

# The Economics Pack

## User Guide

**Thomas Cool, January 2001**

<http://www.dataweb.nl/~cool>

Applications of *Mathematica*

© Thomas Cool, 1995, 2000, 2001

Published by Thomas Cool *Consultancy & Econometrics*

Rotterdamsestraat 69, NL-2586 GH Scheveningen, The Netherlands

NUGI 681, 854

ISBN 90-804774-1-9

*Mathematica* is a registered trademark of Wolfram Research, Inc.

# Brief contents

17	<b>1. The Economics Pack</b>
21	<b>2. Getting started</b>
24	<b>3. System Enhancement</b>
78	<b>4. Logic and Inference</b>
102	<b>5. Economics</b>
266	<b>6. Statistics</b>
384	<b>7. Econometrics</b>
456	<b>Appendix A : Common routines</b>
540	<b>Appendix B : Literature</b>
543	<b>Subject index</b>



# Table of contents

17	<b>1. The Economics Pack</b>
17	<b>1.1 Introduction</b>
17	1.1.1 Origin, aim and result
18	1.1.2 <i>Mathematica</i>
18	1.1.3 Efficiency
18	1.1.4 A guide
18	1.1.5 Acknowledgements
20	<b>1.2 Overview</b>
21	<b>2. Getting started</b>
21	<b>2.1 The first line</b>
21	<b>2.2 The second line</b>
21	<b>2.3 The third line</b>
22	<b>2.4 Using the palettes</b>
23	<b>2.5 All in one line</b>
24	<b>3. System Enhancement</b>
24	<b>3.1 Introduction</b>
24	3.1.1 Always available
24	3.1.2 To be loaded specifically
25	<b>3.2 Money exchange rates and conversion</b>
25	3.2.1 Summary
25	3.2.2 Main routines
26	3.2.3 Exchange rates
27	3.2.4 Exchange rates and interest rates
29	3.2.5 Using an already defined databank
30	3.2.6 Defining a new databank
31	3.2.7 Subroutines
32	3.2.8 MoneyForm
35	3.2.9 Note
36	<b>3.3 Time and Date</b>
36	3.3.1 Summary
36	3.3.2 Date formats
38	3.3.3 Financial date object
39	3.3.4 Input facilities
40	3.3.5 Utilities
42	<b>3.4 Graphically displaying events and plans</b>
42	3.4.1 Summary
42	3.4.2 Examples
43	3.4.3 Main routines
43	3.4.4 Subroutines

45	<b>3.5 Easy construction of arrow diagrams</b>
45	3.5.1 Summary
45	3.5.2 Example
45	3.5.3 User steps
46	3.5.4 The nodes
47	3.5.5 The arrows
47	3.5.6 Show the results
47	3.5.7 Paths
49	<b>3.6 Matrices</b>
49	3.6.1 Summary
49	3.6.2 Block diagonal matrices
50	3.6.3 Definiteness
51	3.6.4 Bordered matrices and Successive Bordered Minors
52	3.6.5 Definiteness for bordered matrices
54	<b>3.7 Calculus, Convexity, Lagrange and Kuhn – Tucker</b>
54	3.7.1 Summary
54	3.7.2 Inverse function
54	3.7.3 Gradient, Hessian and Jacobian
57	3.7.4 Standard analysis of a real function
58	3.7.5 Convexity and concavity
63	3.7.6 Substitute $h[x]$ back in $h'[x]$
65	3.7.7 Constrained optimisation with equality constraints
69	3.7.8 Constrained optimisation with inequality constraints
73	<b>3.8 Databank</b>
73	3.8.1 Summary
73	3.8.2 Introduction
73	3.8.3 Example
75	3.8.4 Databank procedures
76	3.8.5 Show the data
77	3.8.6 Updating

## 4. Logic and Inference

78	<b>4.1 Introduction</b>
78	4.1.1 Decision support environment
78	4.1.2 Propositional logic versus predicate logic
79	4.1.3 Steps and drawbacks
80	4.1.4 About Inference
81	<b>4.2 Propositional logic</b>
81	4.2.1 Summary
81	4.2.2 Loading the package
82	4.2.3 Enhancement of And and Or
83	4.2.4 Decide
83	4.2.5 Englogish : From statement to proper sentence to proposition
83	4.2.6 Conclude
83	4.2.7 Deduce
87	4.2.8 Algebraic structure
88	4.2.9 Truth tables
90	<b>4.3 Logical analysis of longer texts</b>
90	4.3.1 Summary
90	4.3.2 Loading the package
90	4.3.3 Paragraph

91	4.3.4	Utilities
91	4.3.5	Example
93	4.4	<b>Logic laboratory</b>
93	4.4.1	Summary
93	4.4.2	Loading the package
93	4.4.3	LogicLab
93	4.4.4	Example
95	4.5	<b>Inference with the axiomatic method</b>
95	4.5.1	Summary
95	4.5.2	Introduction
95	4.5.3	Infer
98	4.5.4	Axioms and metarules
98	4.5.6	InferenceMachine
99	4.5.7	The axiomatic method and EFSQ
100	4.5.8	Expansion subroutines
100	4.5.8	Different forms
101	4.5.9	Accounting

## 102 5. Economics

102	5.1	<b>Introduction</b>
104	5.2	<b>Social welfare and voting</b>
104	5.2.1	Summary
104	5.2.2	Introduction
105	5.2.3	Key concepts
106	5.2.4	Pareto (efficiency) majority
108	5.2.5	Borda
108	5.2.6	Pairwise majority
110	5.2.7	The Pref[ ...] object
111	5.2.8	The VoteMargin[ ...] object
112	5.2.9	Smaller tools
114	5.2.10	For the link with Graphs
115	5.3	<b>Economic Common routines and other</b>
115	5.3.1	Summary
115	5.3.2	Introduction
116	5.3.3	The main notion of aggregation
118	5.3.4	Cost and profit
119	5.3.5	Aggregator control
119	5.3.6	ResetFactors
122	5.3.7	Other factor control
122	5.3.8	Factor demand
123	5.3.9	Marginal values
123	5.3.10	Elasticities
123	5.3.11	Other properties
124	5.3.12	SetFunction
124	5.3.13	Symbols only
125	5.3.14	Inventory Model
126	5.3.15	Economic`Optimise`
127	5.3.16	Economic`Graphics`
129	5.4	<b>The Constant Elasticity of Substitution function</b>
129	5.4.1	Summary
129	5.4.2	Introduction

130	5.4.3	Canonical and normal form
131	5.4.4	The CES function
132	5.4.5	Flexible input format
133	5.4.6	Check on the elasticity
133	5.4.7	Graphics
134	5.4.8	Utilities
135	5.4.9	Example notebooks
136	5.5	<b>CES – like functions</b>
136	5.5.1	Summary
136	5.5.2	The shifted CES
138	5.5.3	The level CES
141	5.5.4	Indirect Addilog Demand System (IADS)
144	5.6	<b>Applied General Equilibrium analysis</b>
144	5.6.1	Summary
144	5.6.2	Introduction
144	5.6.3	Example
145	5.6.4	Setting up a model
146	5.6.5	Parameters and assignment
146	5.6.6	The equilibrium
147	5.6.7	Plotting
149	5.6.7	Expansion paths
150	5.6.8	Subroutines
152	5.6.9	The Static Leontief Model (SLM)
153	5.6.10	The Dynamic Leontief Model (DLM)
156	5.7	<b>Macro economics</b>
156	5.7.1	Summary
156	5.7.2	Introduction
158	5.7.3	Example model
160	5.7.4	The $I = S$ and $L = M$ model
163	5.7.5	Value added
165	5.7.6	The labour market
169	5.7.7	Undefined symbols
170	5.8	<b>Taxes and premiums</b>
170	5.8.1	Summary
170	5.8.2	Introduction
171	5.8.3	Levy Plot
173	5.8.4	Minimum notions
174	5.8.5	Tax transforms and marginal rates
175	5.8.6	Tax revenue
176	5.8.7	Tax Void Diagram
178	5.8.8	Indexation of rates
179	5.8.9	The Dynamic Marginal Rate
179	5.8.10	Optimal taxation
179	5.8.11	Curved tax
180	5.8.12	Utilities
183	5.9	<b>Game theory and decision theory</b>
183	5.9.1	Summary
183	5.9.2	Introduction
183	5.9.3	Pay – off and Loss
184	5.9.4	LookAt
184	5.9.5	Regret
185	5.9.6	Minimax



185	5.9.7	Mixed strategies and the value of the game
188	5.9.8	Interfacing with the Nash 'Nash' package
188	5.9.9	Minimal due and the value of perfect information
189	5.9.10	Decision theory
191	5.9.11	Relation to prospects and decision trees
194	5.10	<b>Monopoly and cartel</b>
194	5.10.1	Summary
194	5.10.2	Introduction
194	5.10.3	Monopoly
195	5.10.4	Cartel
198	5.11	<b>Peak load pricing</b>
198	5.11.1	Summary
198	5.11.2	Introduction
198	5.11.3	Main routine
199	5.11.4	Subroutines
201	5.11.5	Plotting
202	5.12	<b>Introduction to finance</b>
202	5.12.1	Summary
202	5.12.2	Introduction
203	5.12.3	Symbols only
203	5.12.4	Rate of return
203	5.12.5	Interest rates
204	5.12.6	Portfolio object
205	5.12.7	Simple accounting
206	5.12.8	Accounting: Balance sheet, income statement and cash flow statemen
211	5.12.9	For the CAPM
212	5.13	<b>Finance topics with a flat rate of interest</b>
212	5.13.1	Summary
212	5.13.2	Introduction
212	5.13.3	Discounting
213	5.13.4	CashFlow finance object
214	5.13.5	Present value
215	5.13.6	Yield
215	5.13.7	Periodical payment
217	5.13.8	Payment Tables
217	5.13.9	Schemes on redemption
218	5.13.10	Balance sheet accounting
220	5.14	<b>The Capital Asset Pricing Model</b>
220	5.14.1	Summary
220	5.14.2	Introduction
222	5.14.3	The small models
228	5.14.4	Consequences for the price of an asset
231	5.14.6	Plots and scatter
234	5.14.7	Portfolios
235	5.14.8	CAPMSolve: The Markowitz efficient frontier
239	5.14.9	CAPMSolve: The market portfolio
241	5.14.10	Regression
243	5.14.11	Security Market Line
245	5.14.12	Performance
247	5.15	<b>Finance enhancement</b>
247	5.15.1	Summary
247	5.15.2	Order of contexts

248	5.15.3	Graphics
248	5.15.4	New objects
251	5.15.5	Bonds
254	5.15.6	Portfolio object
255	5.15.7	Utilities
257	5.16	<b>Transport economics and transport science</b>
257	5.16.1	Summary
257	5.16.2	Introduction
258	5.16.3	Vehicle profits
259	5.16.4	Owning or hiring
261	5.16.5	Link capacity
262	5.16.6	Route distances
263	5.16.7	Time equivalent charter
264	5.16.8	Freight

266      **6. Statistics**

266	6.1	<b>Introduction</b>
266	6.1.1	Encouragement
266	6.1.2	Choice of subjects
268	6.2	<b>Data formats</b>
268	6.2.1	Summary
268	6.2.2	Different data formats
269	6.2.3	Data nomenclature
270	6.2.4	Terms
271	6.2.5	Accessing the data
271	6.2.6	Labels
271	6.2.7	Years
272	6.2.8	Exploiting the data structure
273	6.2.9	Setting the data
274	6.2.10	Alternative format : DataList structures
274	6.2.11	Missing data
276	6.3	<b>Data files</b>
276	6.3.1	Summary
276	6.3.2	Using a Dump directory
277	6.3.3	File types
277	6.3.4	Data format
278	6.3.5	DataList format
279	6.3.6	Math format
280	6.3.7	Packages
282	6.3.8	Comma Separated Variable format
283	6.3.9	DIF format
285	6.4	<b>Databases with Dbase`</b>
285	6.4.1	Summary
285	6.4.2	Using a Dump directory
285	6.4.3	Introduction
286	6.4.4	Example data
287	6.4.5	Dbase[ ]
289	6.4.6	Storing the data
291	6.4.7	Reading data
291	6.4.8	Reading an earlier data selection
293	6.4.9	Dictionary

294	6.4.10 Packages that call .dalt's, and more files per context
296	<b>6.5 Chain indices for volumes and prices</b>
296	6.5.1 Summary
296	6.5.2 Introduction
296	6.5.3 The main routines
297	6.5.4 Subroutines
298	<b>6.6 Statistics`Common`</b>
298	6.6.1 Summary
298	6.6.2 Introduction
298	6.6.3 Symbols only
299	6.6.4 Data manipulation
300	6.6.5 Presenting data in a bar chart
301	6.6.6 Statistical measures
304	6.6.7 The normal distribution
305	6.6.8 The lognormal function
306	6.6.9 The Beta function
307	6.6.10 Probability
309	<b>6.7 Probability</b>
309	6.7.1 Summary
309	6.7.2 Prospects
310	6.7.3 Valueing prospects
312	6.7.4 Subroutines
313	6.7.5 Random drawing from Prospects
316	6.7.6 Independent prospects
317	6.7.7 JointProspects
318	6.7.8 Formal development
319	6.7.9 TraditionalForm output printing
319	6.7.10 Event Universe
320	6.7.11 Bayes
320	6.7.12 To conditional
320	6.7.13 From conditional
321	6.7.14 Pr is orderless and ConditionalPr a bit
322	6.7.15 Example of ConditionalPr
324	6.7.16 Utilities
325	6.7.17 Appendix : Rejected formats
327	<b>6.8 Risk</b>
327	6.8.1 Summary
327	6.8.2 Introduction
329	6.8.3 Prospects and risk
331	6.8.4 Risk models
331	6.8.5 Risk concepts
333	6.8.6 Projection of moredimensional prospects and outcomes
335	6.8.7 Subroutines
336	6.8.8 Statistics
337	6.8.9 Prospect plotting
341	6.8.10 JointProspects and risk
343	6.8.11 Continuous densities
344	6.8.12 Symbols only
345	6.8.13 Certainty equivalence
351	6.8.14 Insurance
353	<b>6.9 Statistical decision theory</b>
353	6.9.1 Summary

353	6.9.2	Introduction
354	6.9.3	Example 1
354	6.9.4	Type I and type II errors
356	6.9.5	Technical subroutines
358	6.10	<b>Sampling</b>
358	6.10.1	Summary
358	6.10.2	Introduction
358	6.10.3	Approximating the binomial with a normal distribution
358	6.10.4	Operating characteristic curve
359	6.10.5	Case 1 : Accuracy with confidence for a normal distribution
360	6.10.6	Case 2 : Point hypotheses for a normal distribution
360	6.10.7	Case 3 : Point hypotheses for a binomial, using the normal distribution
362	6.10.8	Apart from these cases : using the Binomial directly
362	6.10.10	Sequential observations and control charts
364	6.11	<b>The <math>\chi^2</math> test</b>
364	6.11.1	Summary
364	6.11.2	Introduction
365	6.11.3	Short example
365	6.11.4	Chi2 and Chi2PValue
367	6.11.5	Example problem with three dimensions
372	6.11.6	The options
373	6.11.7	Subroutine PrEst
374	6.12	<b>Crosstables and the <math>\chi^2</math> test</b>
374	6.12.1	Summary
374	6.12.2	Introduction
375	6.12.3	Example questionnaire
376	6.12.4	Make and QuickTest
378	6.12.5	Using weights
380	6.12.6	Outliers
381	6.12.7	Larger data sets

## 7. Econometrics

384	7.1	<b>Introduction</b>
386	7.2	<b>Dynamic modeling</b>
386	7.2.1	Summary
386	7.2.2	Introduction
387	7.2.3	Models with one lag
388	7.2.4	The Model' package
389	7.2.5	Model[ ]
390	7.2.6	Setting the data
391	7.2.7	Running the model
392	7.2.8	Data management
393	7.2.9	Step by step
394	7.3	<b>Estimation of nonlinear systems</b>
394	7.3.1	Summary
394	7.3.2	Introduction
394	7.3.3	Single equation
396	7.3.4	Systems of equations
398	7.3.5	Subroutines
399	7.4	<b>Timeseries approaches</b>
399	7.4.1	Summary

399	7.4.2	Introduction
399	7.4.3	Backward lag operator
401	7.4.4	Transformation into systems of equations with single lags
402	7.4.5	Comparing Stine's package and WRI's TSP
405	7.5	<b>Neural networks</b>
405	7.5.1	Summary
405	7.5.2	Introduction
405	7.5.3	Freeman's routines
405	7.5.4	Logistic function
405	7.5.5	Hinton diagram
406	7.5.6	Data
407	7.5.7	Input facility for .1 and .9
407	7.5.8	Pattern recognition
408	7.6	<b>Genetic programming</b>
408	7.6.1	Summary
408	7.6.2	Introduction
409	7.6.3	Genesis
411	7.6.4	Estimation fitness criterion
411	7.6.5	Evolution
413	7.6.6	Show and analyse results
414	7.6.7	Subroutines
416	7.6.8	Notes
418	7.7	<b>Operations research</b>
418	7.7.1	Summary
418	7.7.2	Introduction
418	7.7.3	Gantt charts
420	7.7.4	Johnson's rule and Gantt charts
422	7.7.5	Inventory basics
422	7.7.6	Economic Order Quantity
424	7.7.7	Economic Batch Quantity
426	7.7.8	Inventory control : P and Q – systems
429	7.7.9	Piecewise discontinuities
431	7.7.10	One time optimal capacity
434	7.8	<b>Linear programming</b>
434	7.8.1	Summary
434	7.8.2	Introduction
434	7.8.3	Numerical LP analysis
435	7.8.4	Separate routines
437	7.8.5	Unbounded solutions and degeneracy
438	7.8.6	2 D plotting and projecting
439	7.8.7	Boundaries
439	7.8.8	The transport problem
440	7.8.9	Transport problem subroutines
442	7.8.10	Carter's symbolic LP
447	7.9	<b>Queueing</b>
447	7.9.1	Summary
447	7.9.2	Introduction
447	7.9.3	Concepts and symbols
449	7.9.4	The single server model
452	7.9.5	Single server utilities
453	7.9.6	The finite source single server model
453	7.9.7	The multiple servers model

455 7.9.8 Sacrifice ratio plot

456 **Appendix A : Common routines**

457 **A . 1 Numerical calculation**

- 457 A . 1 . 1 Addition and division
- 457 A . 1 . 2 Zero, Indeterminate and Null
- 458 A . 1 . 3 Ranges
- 459 A . 1 . 4 NIntegrate the positive area
- 459 A . 1 . 5 Exponential smoothing
- 460 A . 1 . 6 Some small utilities

461 **A . 2 Symbolic manipulation**

- 461 A . 2 . 1 DQ
- 461 A . 2 . 2 ToSequence
- 462 A . 2 . 3 Expression tests
- 463 A . 2 . 4 Using levels
- 463 A . 2 . 5 Rule basics
- 463 A . 2 . 6 Combine or delete rules
- 464 A . 2 . 7 ReplaceInRule and ReplaceRule
- 465 A . 2 . 8 Replacement by head count
- 465 A . 2 . 9 Other uses of Heads
- 466 A . 2 . 10 Simplification
- 468 A . 2 . 11 Redefine
- 468 A . 2 . 12 UnNumberForm

469 **A . 3 Strings and patterns**

- 469 A . 3 . 1 Names and Symbols
- 469 A . 3 . 2 Lists of strings
- 469 A . 3 . 3 DefinitionToString
- 470 A . 3 . 4 Patterns

471 **A . 4 Lists**

- 471 A . 4 . 1 Quantifiers
- 472 A . 4 . 2 Understanding structures of lists
- 472 A . 4 . 3 Using a mold
- 473 A . 4 . 4 Repeat operations
- 473 A . 4 . 5 Sets
- 475 A . 4 . 6 Matching and pairs
- 475 A . 4 . 7 Indexed sorting and RealSort
- 476 A . 4 . 8 Matrices
- 478 A . 4 . 9 Aggregation and transpose
- 479 A . 4 . 10 ArgMaxQ and ArgMinQ
- 480 A . 4 . 11 Average and standarddeviation, center and radius, and distance
- 481 A . 4 . 12 Angles and polar forms
- 481 A . 4 . 13 Lead, Lag, Dif, UnitIndex, RateOfChange
- 483 A . 4 . 14 LessEqual for list

484 **A . 5 Data input and presentation**

- 484 A . 5 . 1 Data handling
- 484 A . 5 . 2 Presentation with TableForm
- 486 A . 5 . 3 A simple database
- 487 A . 5 . 4 Printing
- 487 A . 5 . 5 Data and DataList formats
- 488 A . 5 . 6 Tabulate

489 **A . 6 Equations and models**

489	A . 6 . 1	Plain symbols
489	A . 6 . 2	Simple symbols with conventions
490	A . 6 . 3	Turning inequalities into equations
491	A . 6 . 4	Variables, variates, symbols, undefined and non – attributed symbols and terms
493	A . 6 . 5	Variations on Solve
494	A . 6 . 6	SolveFrom
496	A . 6 . 7	Presenting and selecting solutions
498	A . 6 . 8	Dynamics
500	A . 6 . 9	Setting values
500	A . 7	<b>Programming</b>
500	A . 7 . 1	Debugging
500	A . 7 . 2	Input Options
501	A . 7 . 3	Output Results
503	A . 7 . 4	Memory constrained execution
504	A . 8	<b>Context</b>
504	A . 8 . 1	Listing the contents of a context or package
504	A . 8 . 2	Listing the contents of a routine
505	A . 8 . 3	Looking for expressions
505	A . 8 . 4	Back Apostrophe
506	A . 8 . 5	\$ContextPath and BeginContext
506	A . 8 . 6	Controlling definitions and adding usage to a key
508	A . 8 . 7	Working in contexts
509	A . 8 . 8	ShadowHull
511	A . 9	<b>Graphics</b>
511	A . 9 . 1	Asymptots
511	A . 9 . 2	Graphics primitives
512	A . 9 . 3	An application of these
514	A . 9 . 4	PlotLine
515	A . 9 . 5	Inverse plot
515	A . 9 . 6	Phase diagram
517	A . 9 . 7	Contours of constraints by polynomial approximation
519	A . 9 . 8	Some utilities
520	A . 10	<b>Economics tools</b>
520	A . 10 . 1	Elasticities and growth
521	A . 10 . 2	Economic functions
522	A . 10 . 3	Some contours
524	A . 10 . 4	Linear programming 2 D Plot
525	A . 10 . 5	Demand and supply cross
527	A . 11	<b>Domain control, inequalities and piecewise</b>
527	A . 11 . 1	Introduction
527	A . 11 . 2	Domain control via piecewise
528	A . 11 . 3	ConditionalApply
529	A . 11 . 4	Domain, Interval and EPS
532	A . 11 . 5	Inequalities and Interval
533	A . 11 . 6	FunctionDomainQ and FunctionRangeQ
533	A . 11 . 7	WhichIFPair : Selecting from various functions and intervals
538	A . 12	<b>Technical note</b>

## Appendix B : Literature

540	B . 1	<b>General literature</b>
-----	-------	---------------------------

541	B . 2 <b>Included papers that use the Pack</b>
-----	--





# 1. The Economics Pack

## 1.1 Introduction

---

### 1.1.1 Origin, aim and result

These notebooks and packages have been developed for economics, business and finance. The software was written while doing research, giving practical decision support and teaching, and it has proven its usefulness many times over. The software is of a basic rather than a grand nature, but it provides a working environment that many will enjoy to have. These applications may help you to get the job done, to get a feel of the discussed problems, or to get a refresher of economics. The software can also be used as a reliable base to create programs of a higher complexity. The name "The Economics Pack" does not mean that you could solve any economic problem with this, but it does mean that when you start doing economics, then you are likely to want to have these tools at your disposal.

To understand the origin of the Pack, you must know that I graduated in 1982, and that I discovered *Mathematica* in 1993, so that I already had a professional life of more than ten years to start from. It was a sensation of some kind to turn this experience towards this new computer language. Since then I have systematically used it in the various projects that I have been involved in. Often the need was felt to provide for some general extensions. Also, I like to have my software well-organised and well documented, so that I can turn to it quickly when I need it again. It was a next observation that others could use the software too.

My experience has been that it frequently takes a lot of time and effort to find a proper formulation for some kind of application, but once this has been found, then it feels rather natural and it becomes relatively easy to develop complexer topics with it. On one hand this is a warning that the simplicity of the various implementations should not deceive you, on the other hand it is a reminder that once you have learned walking, then it is a rather simple thing to do.

Though the focus of the Pack is on economics, business and finance and not on *Mathematica*, the software still comes with all the good properties of the latter.

### 1.1.2 *Mathematica*

*Mathematica* is a language to do mathematics with the computer. Note that mathematics itself is a language that generations of geniusses have been designing to state their theorems and proofs. This elegant and compact language is now being implemented on the computer, and this creates an incredible powerhouse that will likely grow into one of the revolutions of mankind - something that can be compared to the invention of the wheel or the alphabet; at least, it registers with me like that. Note that, actually, it is not the invention of precisely the wheel that mattered, since everybody can see roundness like in irisses, apples or in the Moon; it was the axle that was the real invention. In the same way next generations are likely to speak about the 'computer revolution', but the proper revolution would be this implementation of mathematics.

### 1.1.3 Efficiency

We witness an increasing use of *Mathematica* for economics, business and finance. To reap the rewards of positive network externalities and to increase the productivity of the profession, we should be wise in our programming. There is a lot to be gained from a more disciplined use of *Mathematica* by the economics profession. The programs in the Pack are intended to fit in well with existing programs, and they only deviate when there are compelling reasons to do so.

### 1.1.4 A guide

Since *Mathematica* is such an easy language to program in, it also represents something like a pitfall. It is rather easy to prototype the solution to a problem, or to write a notebook on a subject. But it still appears to be hard work to maintain conciseness, to enhance user friendliness and to document the whole.

Keep in mind the distinction between **(a)** an economic problem, **(b)** how a solution routine has been programmed, **(c)** the way how to use the routines.

The Pack only exists because of ongoing practical applications and original research. See for example the analysis on Social Welfare and the voting paradoxes, or the analysis of the influence of taxes on unemployment, or the estimation of the road freight handling factor. (See appendix B2 for these.)

Nevertheless, this is a guide, and it will concentrate on (c). The problem area (sub a) will be touched on by referring to a common textbook and research papers. A full discussion of the routines (sub b) would take too much time (and one should be able to use `ShowPrivate[]`). Thus, while the proper focus is on the *why*, i.e. the application of these routines, below we will still focus on explaining *how* to use them.

### 1.1.5 Acknowledgements

This guide would not have been possible without the help of others.

First of all, I want to thank Leendert van Gastel (<http://amstel.wins.uva.nl/~gastel/>) and André Heck (<http://amstel.wins.uva.nl/~heck/>) who allowed me since 1994 to visit the Computer Algebra Nederland foundation. I also want to thank Dick Verkerk, originally at CAN and who now directs CANDiensten, for his support in so many ways (<http://www.candiensten.nl>).

I want to thank Asahi Noguchi and Silvio Levy for their kind permission to rework their original package on applied general equilibrium analysis, and to include the "MyFindRoot" package in my Pack.

I want to thank Michael Carter for his equally kind permission to rework his package on linear programming. My own approach originally was only numerical, and concerned a shell around the routines provided by *Mathematica* itself, but Michael showed us how it could be done symbolically, and that is a far more promising approach for non-standard cases.

The makers of *Mathematica*, Wolfram Research Inc. (<http://www.wolfram.com>), provided much software and assistance, and I actually wrote this guide using the tools provided on their Developer CD. Especially Brett H. Barnhart of WRI gave perfect support at crucial moments and I am greatly indebted to him.

I also want to thank Rolf Mertig, since the production of this version of the Pack would neither have been possible without his expert assistance on some technical issues (<http://www.mertig.com>).

A final word of gratitude is due for the users who have provided me with their remarks and with the feedback over the years. This feedback was important and has contributed to a better result. This especially holds for the software reviews by Ron Shone in the Economic Journal. (See my site for details.)

## 1.2 Overview

---

The subjects have been arranged in an order that reflects content. The prime order is Economics, Statistics, Econometrics. The order within these chapters may reflect a course in those subjects. Given this order, it still appears that the guide better starts with the chapters on the System Enhancement since the later chapters build on these. The order of discussion thus is:

- Enhancement
- Economics
- Statistics
- Econometrics

The notebooks included in the Pack have been similarly arranged in subdirectories and help (search) categories. Note that these notebooks are accessible from the **Open File** menu and from the **Help** menu. It is important to be aware that if you evaluate a notebook under the Help menu, then all changes will be lost after closing that file. However, if you directly open a notebook, then the evaluation results can be retained, and your changes will also appear in the Help menu. But you could also delete all output again - see the appropriate command from the Kernel menu in the menu bar. Also, note in the Help Browser window that there is a button to hide the search categories, so that you can have a larger reading area. This makes the help function even more effective.

Note that the size of the printed Guide and the usefulness of above rubrication are reaching some limits. Since the Pack basically grows over the years, there is a question how to proceed in the future. The online Guide is kept up to date with gradual improvements, but the printed Guide has a different production process, and some differences with the most recent online version must be accepted. For additions to the material, the practice has been to include the packages with some working papers and explanatory notebooks, but without entries in the Guide. Thus the list of contents and the subject index show what can be found. But not everything that can be found is in those entries.

Recently, I also completed Cool (2001), *Voting Theory for Democracy, Using Mathematica programs for Direct Single Seat Elections*. This was a major operation that lead to the development of many more voting routines than documented here in The Economics Pack. Also developed were packages on *deontic logic* and *economic fairness*. Interestingly, a topic of *testing* could be included there too. You are referred to that book. A new package on *income inequality* however is documented only in working notebooks.

# 2. Getting started

The Economics Pack becomes fully available by the single command `<<Economics`All``. It is good practice however to use a few separate command lines to better control the working environment. Three lines can be advised in particular.

## 2.1 The first line

---

You start by evaluating:

```
Needs["Economics`Pack`"]
```

This makes the `Economics[]` command available by which you can call specific packages and display their contents. Before you use this, read the following paragraphs first.

## 2.2 The second line

---

`CleanSlate`` is a package provided with *Mathematica* that allows you to reset the system. You thus can delete some or all of the packages that you have loaded and remove other declarations that you have made. The only condition is that `CleanSlate`` resets to the situation that it encounters when it is first loaded. You would normally load `CleanSlate`` after you have loaded some key packages that you would not want to delete. The `ResetAll` command is an easy way to call `CleanSlate``. Your advised second line is:

```
ResetAll
```

<code>ResetAll</code>	<code>ResetAll</code> calls <code>CleanSlate</code> , or if necessary loads it. This means that your notebook does not have to distinguish between calling <code>CleanSlate`</code> and evaluating <code>CleanSlate[]</code>
-----------------------	---

Note that if you first load `CleanSlate`` and then the Economics Pack, then the `ResetAll` will clear the Pack from your working environment, and thus also remove `ResetAll`. If you would happen to call `ResetAll` again after that, then the symbol will be regarded as a `Global`` symbol.

## 2.3 The third line

---

After the above, you could evaluate `EconomicsPack` to find the list of packages.

```
EconomicsPack
```

Select the package of your interest, load it, and investigate what it can do. For example:

```
Economics[Logic]
```

You can suppress printing by an option `Print → False`. You can call more than one package in one call. If you want to work on another package and you want to clear the memory of earlier packages, simply call `ResetAll` first. This also resets the `In[]` and `Out[]` labels.

<code>Economics[xi, ...]</code>	shows the contents of <code>xi</code> and if needed loads the package(s). Input <code>xi</code> can be <code>Symbol</code> or <code>String</code> with or without back– apostrophe. To prevent name conflicts, Symbols are first removed. <code>Economics[ ]</code> doesn't need the <code>Cool</code> , <code>Varianed</code> etc. prefixes
<code>Economics[All]</code>	assigns the <code>Stub–</code> attribute to all routines in the <code>Pack</code> (except some packages)
<code>EconomicsPack</code>	gives the list {directory → packages}

Note: `Economics[All]` will not load the `GenePro`Cart`` package and the packages in the `Data`` and `DataList`` subdirectories since these are too specific.  
Note: `Economics[x, Out → True]` puts out the full name of the context loaded. See A.12 Technical Note for using this feature when making packages that rely on the `Economics Pack`.

## 2.4 Using the palettes

The `Pack` comes with some palettes. These palettes have names and structures that correspond to the chapters in this guide.

- The master palette is `"TheEconomicsPack.nb"` and it provides the commands above and allows you to quickly call the other palettes or to go to the guide under the `help` function.
- The other palettes have `"TEP_"` as part of their name, so that they can easily be recognised as belonging to the `Pack`. These `"TEP_"` palettes contain blue buttons for loading the relevant packages and grey buttons for pasting commands.
- The exception here is `"TEP_Arrowise.nb"` that only deals with the package for making arrow diagrams.

Note: For *Mathematica* 3.0 (but not for 4.0) the palettes have to be copied (but only once) from the `Economics` directory to the *Mathematica* system directory. You would do (only once):

```
InstallEconomicsPalettes[];
```

```
Copying TheEconomicsPack.nb to the system location
```

```
Copying TEP_Arrowise.nb to the system location
```

```
Copying TEP_Common.nb to the system location
```

```
Copying TEP_Enhancement.nb to the system location
```

```
Copying TEP_Economics.nb to the system location
```

```
Copying TEP_Statistics.nb to the system location
```

```
Copying TEP_Econometrics.nb to the system location
```

You may have to restart *Mathematica* 3.0 before the palettes are shown in the palettes menu.

## 2.5 All in one line

---

You can also load the Pack by the following single line.

```
<<Economics`All`
```

This evaluates Needs["Economics`Pack`"] and Economics[All], and opens the palettes. It does not call ResetAll, however.



## 3. System Enhancement

### 3.1 Introduction

---

Although the Economics Pack has been developed for economics, statistics and econometrics, there often appeared a need for routines that enhanced the *Mathematica* system and that could be of more general use. We distinguish two kinds of such enhancements.

#### 3.1.1 Always available

First of all there are the small commands and routines in the `$Economics` common packages, i.e. those packages that are always available.

```
Needs["Economics`Pack`"]

$Economics

{Cool`Inequality`, Cool`Graphics`, Cool`Manager`,
 Cool`Tool`, Cool`List`, Cool`Context`, Cool`Common`}
```

These routines are often crucial for the workings of the higher level routines. Still, when regarded by themselves, they also appear a bit simple, and less relevant for who wants to do economics. For that reason the discussion of these routines is put in the appendices. They are discussed there by topic of interest to the user, and not by package of origin.

#### 3.1.2 To be loaded specifically

Secondly there are the more specific system enhancements, i.e. those projects that warranted packages of their own. These are: `Money``, `Time``, `ShowPlan``, `Arrowise``, `Matrices``, `Calculus``, `Lagrange`` and `Databank``. These more specific enhancement packages will be discussed separately.

## 3.2 Money exchange rates and conversion

### 3.2.1 Summary

The package integrates with *Mathematica's* Convert routine, and allows you to work with a database of money exchange rates.

**Economics [Money]**

### 3.2.2 Main routines

\$Convert allows conversions also with a default (home) currency - at startup set to the Dollar:

$$\begin{aligned} & \$\text{Convert}\left[\frac{\pounds 5.32}{\text{Kilogram}}, \frac{\text{¥}}{\text{Pound}}\right] \quad \& \quad \$\text{Convert}\left[\frac{\pounds 5.32}{\text{Kilogram}}\right] \\ & \frac{374.606 \text{ ¥}}{\text{Pound}} \bigwedge \frac{7.61959 \$}{\text{Kilogram}} \end{aligned}$$

<code>\$Convert [old (, new , sel)]</code>	calls FromMoneyGraphics, MoneyConvert and ToMoneyGraphics (with sel)
<code>ToMoneyGraphics</code>	{Dollar→\$, Yen→¥, PoundSterling→£, Guilder→f, Euro → ■}
<code>FromMoneyGraphics</code>	Reversed ToMoneyGraphics
<code>MoneyConvert [old , new (, sel)]</code>	converts old to new, using the databank entry sel

The default exchange rates are taken from ExchangePrices[]. Another selection *sel* can be given with the format Date → x, Label → y\_String, Databank → z\_Symbol.

At startup \$Convert knows only a few money graphics, i.e. the \$, ¥, £, f and €. These graphics are accessible e.g. by \[.] notations (see e.g. the *Mathematica* book Section 3.10.3). The € is available in *Mathematica* 4.0. In version 3.0 the package uses capital epsilon. For uniformity, the Euro graphic here is also defined by \$Euro (€). You can always correct and extend this list of money graphics.

The list of known money symbols is a bit larger - and also extendable:

**Valuta [EuroBank]**

{ \$Australia, Baht, BEF, Bolivar, \$Canada, DKKrone, DM, \$Drachma, Escudo, Euro, Franc, Guilder, Lira, Markka, \$NewZealand, Peseta, Peso, PoundSterling, Punt, Rand, Real, Rupee, Schilling, \$Singapore, Krona, CHFranc, \$Taiwan, Won, Yen, Yuan }

- MoneyConvert works only with money symbols, and not with money graphics. It is a bit more flexible than *Mathematica's* Convert. You can MoneyConvert to one currency, while you can Convert only to the same mathematical format. The default is conversion to Dollars.

```
MoneyConvert[ $\frac{34 \text{ Yen}}{\text{Pound}}$ , PoundSterling]
```

```
 $\frac{0.219019 \text{ PoundSterling}}{\text{Pound}}$ 
```

```
Convert[ $\frac{\text{Dollar}}{\text{Kilogram}}$ , Euro];
```

```
Convert::incomp : Incompatible units in  $\frac{\text{Dollar}}{\text{Kilogram}}$  and Euro.
```

Before we discuss how you can define a database of exchange rates, it is better to clarify how the rates have been implemented here.

### 3.2.3 Exchange rates

Exchanging uses the price of one (default: foreign) currency in terms of another (default: domestic) currency. The default at startup takes the Dollar as the home currency. Calling ExchangePrices[] gives rules  $\{f \rightarrow x \text{ Dollar}, g \rightarrow y \text{ Dollar}, \dots\}$  for the foreign currencies  $f, g, \dots$ . It follows from the substitution that  $x \text{ Dollar}$  is the domestic price of one unit of the foreign currency  $f$ . The market convention however is to quote how much one unit of the domestic currency can buy of each foreign one, which is  $1/x$ . These then are the ExchangeRates[] or ExchangePrices[] // InvertRates. Working with prices is easiest for substitution.

- ExchangePrices[] are used by MoneyConvert for the current conversion rates. See SetExchangePrices below for setting the prices, including the default Home valuta. Results[ExchangePrices] records when and how these prices have been defined. Since the list of valuta is so long, we don't evaluate ExchangePrices[] now.

```
Results[ExchangePrices]
```

```
{Date → F(10, 26, 2000), Label → IMF & Pac. Com., Databank → EuroBank, Valuta → Dollar}
```

```
ExchangePrices[]
```

```
ExchangeRates[]
```

Thus note: When  $f/d$  is read as a value ratio then it gives  $x$ , but when  $f/d$  is read as the dimension of 'how many  $f$  for 1  $d$ ' then it gives  $1/x$ .

<code>ExchangePrice[ f_Symbol]</code>	selects $f \rightarrow x \ d$
<code>ExchangeRate[f_Symbol]</code>	gives $d \rightarrow 1/x \ f$
<code>ExchangeRate[<math>\frac{f\_Symbol}{d\_Symbol}</math>]</code>	gives $1/x$ from $f \rightarrow x \ d$
<code>ExchangeRate[f <math>\rightarrow</math> x d]</code>	transforms to $\{1/x, f/d\} =$ 'how many f for 1 d'
<code>ExchangeRates[v]</code>	gives the values of foreign currency in terms of home valuta v. If v is not specified then the default of SetHome is taken

Note that the list of quoted rates is sorted differently for each different v.

**ExchangePrice[Euro]**

**ExchangeRate[%]**

**ExchangeRate[Euro]**

**(Euro / Dollar) /. ExchangePrices[]**

**ExchangeRate[Euro / Dollar]**

### 3.2.4 Exchange rates and interest rates

We can use a simple model of interest arbitrage - that e.g. neglects surprises - to explain how the exchange rates have been implemented here. If you earn interest at home, then this should be the same as converting to a foreign currency, earn the foreign interest, and converting back, later, at the expected future exchange price. That price can fall or rise. An appreciation is a rise of a price. The appreciation rate is the rate of growth of the exchange price. If a foreign currency appreciates, the home currency depreciates.

- We use Home and \$Home for the domestic (future) currency, Foreign and \$Foreign for the foreign (future) currency, and HomeRate and ForeignRate for the rates of interest. The AppreciationRate is for the foreign currency in terms of the domestic one.

**ExchangeEquations[Rule] /. ToExchangeSymbols[]**

$$\left\{ \text{Exp}\left[E_{\frac{s}{e}}\right] == (a + 1) E_{\frac{s}{e}}, a == \frac{R_s + 1}{R_e + 1} - 1, R_s + 1 == \frac{\text{Home \$Foreign} (R_e + 1)}{\text{Foreign \$Home}} \right\}$$

`ExchangeModel[{r___Rule}, {elims}, opts]`

applies the model for the appreciation of a currency based on the rates of interest and the (future) exchange prices of the home and foreign currency. It is a `SolveFrom` application that takes its equations from its options

`ExchangeEquations[Rule]`

gives the equations for the appreciation of the foreign currency, based on the rates of interest and the current and future exchange prices of the home and foreign currency. This is the default option of `ExchangeModel[]`.

`ExchangeEquations[All]` extended list of equations

`ToExchangeSymbols[]` contains a list of substitution rules, using \$ and \$Euro

These models must be used with some care because `SolveFrom` uses substitution rules, like  $f \rightarrow x d$ , and then there could be a conflict with the dimensional interpretation  $f/d$ .

If the home interest is 5% (10%), the foreign interest 10% (5%), while the current exchange price is 1.1 then this situation is only in equilibrium if the foreign currency depreciates (appreciates).

`ExchangeModel[{Foreign → 1.1 Home, ForeignRate → .1, HomeRate → 0.05}]`

`ExchangeModel[{Foreign → 1.1 Home, ForeignRate → .05, HomeRate → 0.1}]`

`Appreciation[rd][f → x d, rf]` gives  $f \rightarrow x d (1+rd)/(1+rf)$  as the new rule when the d currency has interest rd and the f currency has interest rf

`Appreciation[rd, rs_List(, ers)]`

does the same using `MapThread`, when rs are the foreign rates of interest, and where ers is the list of exchange rate rules (default `ExchangePrices[]`)

There is also `Depreciation`, a Symbol only.

The domestic interest rate would affect all expected exchange rates if the foreign rates of interest remain the same.

- Let the domestic currency earn 10% and the foreign currency earn 5%.

`Appreciation[0.1][Foreign → 1.1 Home, 0.05]`

- The domestic interest rate should affect all other rates (not evaluated).

`InterestRates[] = Table[0.05 + 0.01 Random[],  
{i, Length[ExchangePrices[]]}];`

Appreciation[0.1, InterestRates[]]

3.2.5 Using an already defined databank

The internal procedure is that \$Convert and MoneyConvert call on ExchangePrices[]. If you would want other rates, it could be sufficient to adjust that list. However, in a systematic framework you would like to define a databank that stores values for different sources and different times. Let us show this by using the existing databank ERBank: we have to set the option Databank → ERBank in order to make this the default bank, and we then can use the various exchange rates in it, identified by source (Label→ ...) and by date (Date → ...). The command SetExchangePrices[...] calls the databank and sets ExchangePrices[]. Results[ExchangePrices] records what selection is being used henceforth. If we use SetHome[...] then also the options are adjusted, and we no longer need to mention all selected options all the time.

SetExchangePrices [opts]

selects the exchange prices from the database of rates, and sets ExchangePrices[].  
You can use options Date → x, Label → y\_String, Databank → z\_Symbol

ExchangePrices [opts] gives the prices. If options are specified,  
then SetExchangePrices is called first

SetHome [opts] sets the options of SetExchangePrices, then calls it

SetHome [v] sets the domestic Valuta → v (Blank gives the Dollar)

Note: The Date can be entered in any date format but will be transformed into the F format of the Time` package. See the Results. MoneyConvert uses Valuta[All] to identify the currency names.

Note: The Money` package thus relies on the Databank` package. This comes with the advantage that some commands are available automatically, like Explain and ShowData. Be aware though that these routines rely on the Databank option of SetDatabank, and not on the Databank option of SetExchangePrices. SetHome helps you to set these options to the same value. Note: Money is a subject of economics, and databanks are not, so this section on money conversion was put before the databank section. Now, however, you may want to start reading section 3.8 on the Databank routines first, before you continue with money conversion.

- Let us take this databank, and let us set the default currency to the Euro.

SetExchangePrices[Date → F[October, 26, 2000], Label → "IMF & Pac.  
Com.", Databank → ERBank, Valuta → Euro]

{DM → 0.511338 Euro, Dollar → 1.2087 Euro,  
Franc → 0.152431 Euro, Guilder → 0.453784 Euro, Lira → 0.000516583 Euro,  
Peseta → 0.00601044 Euro, PoundSterling → 1.73117 Euro, Yen → 0.0111517 Euro}

**Results[ExchangePrices]**

{Date → *F*(10, 26, 2000), Label → IMF & Pac. Com., Databank → ERBank, Valuta → Euro}

- ShowData[] gives the contents of the current databank - in Foreign/Dollar dimensions.

**ShowData[]**

	1	2	3
Date	<i>F</i> (4, 4, 1998)	<i>F</i> (7, 24, 1998)	<i>F</i> (10, 26, 2000)
Label	The Economist	Volkskrant NY	IMF & Pac. Com.
DM	1.85	1.78	2.3638
Euro	1.08	1.10843	1.2087
Franc	6.2	5.966	7.9295
Guilder	2.08	2.0069	2.6636
Lira	1823.	1753.	2339.8
Peseta	157.	151.579	201.1
PoundSterling	0.6	0.602519	0.6982
Yen	133.	141.65	108.387

Note: The financial date format *F*[...] is defined in the `Time`` package.

**3.2.6 Defining a new databank**

At start up time, the databanks EuroBank, ERBank and WorldBank have been defined as `Databank`` objects. The convention for any databank is that it contains quoted rates of foreign valuta to one \$. For example, one \$ would give about 2 DM. If you have evaluated above, then the Results[ExchangePrices] refer to ERBank as the databank that has been used to extract the exchange rates.

ERBank	Databank object for storage of exchange rates. The date has a financial format <i>F</i> [], and the label is a String. Rate data are pure numbers
EuroBank	Databank object for storage of exchange rates. The date has a financial format <i>F</i> [], and the label is a String. Rate data are pure numbers

When something is a `Databank` object, its options explain how it is structured. The `DataMold` here states that data are identified by a date and label, while the valuta are mentioned in the second element of its `DataMold`.

- Since ERBank is a `Databank`` object, it has a `DataMold` option.

**Options[ERBank]**

{DataMold → {{Date, Label}, {DM, Euro, Franc, Guilder, Lira, Peseta, PoundSterling, Yen}}}

3.2.7 Subroutines

<code>FromHomePrices [opts]</code>	inverts the ExchangePrices[opts]
<code>FromHome [expr, v_Symbol, opts]</code>	applies the inverted rate for valuta v
<code>ToHome [expr (, opts)]</code>	converts to the home currency, using ExchangePrices[opts]
<code>Valuta [b]</code>	takes the second element of the DataMold of databank b, giving the list of currencies, excluding Dollar
<code>Valuta []</code>	takes its names from First /@ ExchangePrices[]
<code>Valuta [All]</code>	Prepends Valuta[] with Valuta /. Results[ExchangePrices]

Note that f[... opts] also redefinesExchangePrices[]. Note: When the Dollar is the default Home currency, then also the functions ToDollar, FromDollar and FromDollarPrices can be used.

- Note that ToHome can adjust ExchangePrices[] but not the Options[SetExchangePrices].

**ToHome[35 PoundSterling]**

60.5908 Euro

**ToHome[35 PoundSterling, Date → F[April, 4, 1998], Label → "The Economist", Databank → ERBank]**

58.3333 Dollar

**ToHome[35 PoundSterling]**

58.3333 Dollar

**Results[ExchangePrices]**

{Date → F(4, 4, 1998), Label → The Economist, Databank → ERBank, Valuta → Dollar}

<code>EuroLand []</code>	gives the values of the Euro in local currencies in Euroland per January 1 1999
<code>SDR []</code>	gives the IMF Special Drawing Rights basket per January 1 1999



```
EuroLand[] // InvertRates

(Euro → 40.3399 BEF, Euro → 1.95583 DM, Euro → 200.482 Escudo,
 Euro → 6.55957 Franc, Euro → 2.20371 Guilder, Euro → 1936.27 Lira, Euro → 5.94573 Markka,
 Euro → 166.386 Peseta, Euro → 0.787564 Punt, Euro → 13.7603 Schilling)

SDR[]

0.5821 Dollar + 0.3519 Euro + 0.105 PoundSterling + 27.2 Yen

ToHome[%, Valuta → Dollar, Date → F[1, 1, 1999], Label →
  "EU, IMF, FED NY, DV", Databank → EuroBank]

1.40531 Dollar

$Convert[%, Euro]

1.20447 €

$Convert[%, Dollar]

1.40531 $
```

3.2.8 MoneyForm

If you add money symbols to numbers, then you can do various kinds of operations. The \$Convert routine has been written with this kind of usage in mind indeed.

```
$ 6676734.23 / 58

115116. $
```

- There are some (limited) limitations though.

```
Sqrt[%]

339.288 √$
```

However, if you want to print with the \$ (or any other graphic or symbol) *in front of* the number, and if you possibly want to have the digits grouped in three, then you have to use NumberForm. Using NumberForm however comes at the price of being less able to do arithmetic. The functions Val, DeVal and ReVal should make life a bit easier here.

Val[x]	first removes NumberForm wrappers and then applies MoneyForm
DeVal[x]	removes NumberForm wrappers and explicitly multiplies with the valuta
ReVal[v x]	turns into MoneyForm if v x consists of a number times another term
Val\$Convert[x, y]	unwraps MoneyForm, \$Converts, and wraps again

Note: Val\$Convert is noticeably weaker than \$Convert, because of the final ReVal stage that only recognises at multiplication pair.

```
SetOptions[MoneyForm, Valuta → ¥];
```

- The price for using MoneyForm is that you have to call Val, DeVal or ReVal repeatedly.

```
x = 227641. * 78 // Val
```

¥ 17,755,998.

```
x / 78 // Val
```

¥ 227,641.

```
Sqrt[%] // Val
```

¥ 477.12

- See what happens if you don't call Val.

```
x / 78
```

$$\frac{\text{¥ } 17,755,998.}{78}$$

- Also, you cannot simply \$Convert any more. You have to UnNumberForm and then include a valuta Symbol, since the one shown is only a padding for printing. DeVal does both steps. Then, your result is not in MoneyForm yet. ReVal does that for you.

```
$Convert[x/78 // DeVal, Dollar] // ReVal
```

\$ 2,000.

- In short:

```
Val$Convert[x/78, Dollar]
```

\$ 2,000.

- While the following would lead to nowhere.

```
$Convert[x/78, Dollar]
```

$$\frac{\text{¥ } 17,755,998.}{78}$$

The subroutines of Val are MoneyForm and UnNumberForm. The latter is rather clear. For the first:

```
MoneyForm[x_?NumberQ, opts___Rule]
```

displays the result in money format, i.e. with valuta, comma's and cents.

Options: The Valuta option gives the valuta.

The Integer option gives the column width; default 0 means automatic calculation. The other options adjust those of NumberForm.

```
MoneyRound[x_?NumberQ, opts___Rule]
```

rounds the number to 2 decimals (cents) and displays with MoneyForm. It

differs from MoneyForm in that a really rounded number is returned

Note: A slight drawback is that a zero (0) in the final position will not be printed, e.g. 78.90 becomes 78.9.

Note: The documentation says that NumberForm works only as a wrapper, but this is only so for %, and not when symbols are assigned, like when variable x once has been set `x = 12345.67 // MoneyForm`.

- Note: This prints with Yen and allows working with x. But it reads less nicely as // Val, and the subsequent output does not have the money tag.

```
MoneyForm[x = 3048.34 * 78]
```

```
¥ 237,770.52
```

```
x / 78
```

```
3048.34
```

A similar story holds for PaddedForm. And this form doesn't have the DigitBlock option.

```
MoneyPaddedForm[number, n:0]
```

puts the number in PaddedForm with {n, 2} (cents). Default 0 = with the appropriate number of padding places. Use UnPad to get rid of the PaddedForm for further manipulations

```
MoneyPaddedRound[x_?NumberQ]
```

rounds the number to 2 decimals (cents) and puts it in such PaddedForm. It differs from MoneyPaddedForm in that after UnPad a really rounded number is given

### 3.2.9 Note

The following are new commands that have not been documented nor indexed yet.

**?MoneyBanks**

**?MoneyNames**

**?FindHomeValuta**

**?InvertRates**

**?WorldBank**

This package designs exchange rates as a vector, which is the most common use. Exchange rates actually are more like a matrix rather than a vector, since we cannot presume that arbitrage is perfect. The current design gives some perspective on this more complicated structure.

## 3.3 Time and Date

---

### 3.3.1 Summary

Time and Date routines are important for finance and decision analysis. *Mathematica* comes standardly with the `Miscellaneous`Calendar`` package, while other routines are provided by the Finance Essentials Package. These routines are here extended on.

- Load the package  
`Economics[Time]`

### 3.3.2 Date formats

The package recognises different date input formats and defines transformations between those. The finance date object is defined as `F[m, d, y]`, and the Dutch (European) date object is `Du[d, m, y]`. `CoordiDate[date]` gives the date as a coordinate on the time axis, and `RealToDate[point]` returns the date. There are also tests for dates and leap years.

<code>AnyToStandardDate[x]</code>	converts any date format into the standard format of a list of six integers
<code>AnyToCalendarDate[x]</code>	converts any date format into the Calendar format of a list of three integers
<code>DateQ[x]</code>	tests whether x is a proper date. Options for Calendar and string format apply
<code>DateFormat</code>	Give <code>??DateFormat</code> to see the various date formats in use

Note: For the first two functions, an option `First → year` may be passed on for the String format.

**?? DateFormat**

```
Give ??DateFormat to see the various date formats in use.

DateFormat[] = {Year, Month, Day, Hour, Minute, Second}

DateFormat[Axis] = Real: using equal units for all years

DateFormat[Dutch] = Du[ d, m, y ]

DateFormat[Finance] = F[ m, d, y ]

DateFormat[FinancePackage] = {m, d, y}

DateFormat[Integer] = NumberOfSeconds (sometimes NumberOfDays)

DateFormat[Miscellaneous`Calendar`] = {y, m, d}

DateFormat[Standard] = { y, m, d, h, min, s}

DateFormat[String] = "yymmdd"
```

Subroutines for the various transformations are:

MakeDate[x_List]	appends up to six 0's to get Date format
IntegerToDate[x]	changes 199012 or 19901231 format into Date format. The standard <i>Mathematica</i> integer date format is the number of seconds since January 1 1900, and thus has preference above this use of integers. (See FromDate and ToDate.) However, the present format may be helpful sometimes
CoordiDate[x]	changes date into numerical value on time axis, using years of equal length, and neglecting leap year differences. An option First → year may be passed on to the String format.
RealToDate[x]	changes a real back into a Date; the inverse of CoordiDate
StringToDate[x, opts]	changes a String like 950915 into a proper six digit date list. Default option is First → 1950, meaning that 50mmdd is recognised as 1950 but lower digits as 20 up

"000109"

000109

**AnyToCalendarDate[%]**

{2000, 1, 9}

**CoordiDate[%]**

2000.02

Similarly, when integers stand for the number of days.

DateToDays[CalendarDate, Gregorian | Julian | Islamic]

gives the number of days since {1,1,1}  
according to that calendar. Gregorian is the default.

DaysToDate[integer number, Gregorian | Julian | Islamic]

gives the standard date, counted as the number of days  
since {1,1,1} according to that calendar. Gregorian is the default.

3.3.3 Financial date object

To prevent miscommunication about dates in finance it seems useful to define a finance date object. The object label F can again be removed when one uses routines that don't recognise that F object. Be careful about removal, though:

- use (x /. F → List) for financial dates as in the Finance Essentials package
- use (x /. F → FromF) for dates as in the Miscellaneous`Calendar` package.

F[m, d, y]	finance date object
Du[d, m, y]	inputfacility, transforms into Ft[m, d, y]
Ft[m   y, d, m   y]	or
Ft[F[_]]	checks whether we have a date, and transforms a standard <i>Mathematica</i> date into financial date
FQ[F[_]]	applies DateQ to finance date. Can also be used as in {F[...], F[...], ...} /. F → FQ
ToF[x]	changes a standard date into a financial one
FromF[x]	changes a financial date into a standard one
FDate[]	= ToF[ Date[] ]
FSort[x_List]	sorts a list of financial dates
NoF[x__]	removes F and returns Sequence@@({x} /. F → List)

The Options[FromF] allow you to join a date with information on hour, minute and second. The default is Join → {}.

3.3.4 Input facilities

The months are input facilities with the fixed values 1 till 12.

January	April	July	October
February	May	August	November
March	June	September	December

```
Ft[2000, December, 9]

F(12, 9, 2000)

AnyToCalendarDate[%]

{2000, 12, 9}

CoordiDate[%]

2000.94

RealToDate[%]

{2000, 12, 9, 0, 0, 0}

{F[January, 1, 1999], F[March, 2, 2001]} /. F -> FromF

( 1999 1 1 )
( 2001 3 2 )
```

You may also wish to use labels that are not numeric. The first three letters of the months have been reserved for this (three for neater tables). Also lowercases are used, since otherwise there would be a conflict with May.

MonLabels	{jan, ..., dec} in lower case
MonRule	rule to replace short month labels into number 1, ..., 12
MonSort[x]	sorts a list x of F dates, and replaces months with mon labels

```
MonLabels

{jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec}

{F[may, 1, 2003], F[jan, 3, 2002]}

{F(may, 1, 2003), F(jan, 3, 2002)}

% /. MonRule

{F(5, 1, 2003), F(1, 3, 2002)}
```



```
%% // MonSort
{F(jan, 3, 2002), F(may, 1, 2003)}
```

3.3.5 Utilities

Some small utilities are:

DayFraction[ <i>day</i> ]	explains a real number of days as {days, hours, minutes, seconds}
LeapQ[ <i>year</i> ]	is True when year is a leap year and False otherwise. Default option is Calendar → Gregorian.
Time	Symbol only
Days[ ]	gives the list of days
Months[ ]	gives the list of months as strings

```
DayFraction[3.4]
{3, 9, 35, 59}
```

PlusYears[ <i>date</i> , <i>r</i> ]	adds a number <i>r</i> of years to the date
-------------------------------------	---

Floor[*r*] is taken as the integer number of years, and the remaining fraction is multiplied by 365 or 366 days depending upon whether the final year is a leap year or not. Also day fractions are allocated.

- In 2000, a second is 1 / (366 24 60 60)'s of a year, and adding such fraction to January 31 midnight, flops the calendar to February 1.

```
PlusYears[{2000, 1, 31, 23, 59, 60}, 1/(366 24 60 60)]
{2000, 2, 1, 0, 0, 1}
```

DateToText[ <i>x_list (standard date)</i> ]	gives a text format
DateToText[ ]	gives the current date
TextToDate[ <i>x_String</i> ]	changes a text format into the Calendar date

```
DateToText[]
April 24, 2000
```

The Finance Essentials Package allows one to work with a theoretical environment of years of 360 days, without leap years. This theoretical angle appears to be needed regularly, also for years of 365 days.

```
NormedDuration[{begin, end}, opts] or
NormedDuration[begin, end, opts]
```

gives a duration in days, from begin to end date,  
by using a primitive weighing scheme of standard lengths. This e.g. excludes leap years.  
The defaults define the standard year of 365 days and the standard month of 365/12 days

```
RealToNormedDuration[x, opts___Rule]
```

translates a real duration into the number of years, months, days, etc.

For the latter: Options Year and Month are taken from Options[NormedDuration]. Additional options are: If Drop → True, then only Year, Month and Day are given (Hours etc. dropped). If Print → True, then output contains an explanation

**Options[NormedDuration]**

$$\left\{ \text{Year} \rightarrow 365, \text{Month} \rightarrow \frac{365}{12} \right\}$$

- The normed duration with above settings differs from the DaysBetween that takes account of unequal months and leap years.

```
NormedDuration[{2000, 1, 1}, {2000, 5, 2}]
```

$$\frac{368}{3}$$

```
DaysBetween[{2000, 1, 1}, {2000, 5, 2}]
```

122

```
RealToNormedDuration[1.45, Drop → True, Print → True]
```

$$\left( \begin{array}{ccc} \text{Year} & \text{Month} & \text{Day} \\ 1 & 5 & 12.1667 \end{array} \right)$$

## Note

These routines are also used in these example basic financial mathematics intermediate test and exams.

# 3.4 Graphically displaying events and plans

## 3.4.1 Summary

The following creates a graphical image of the time lapse between events, and of the time span of a plan. Here we take:

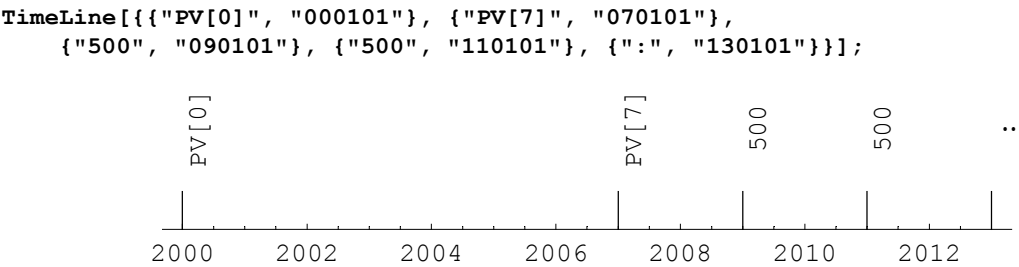
- an event is a *text* and its *date*
- a plan is an *event* and its *date*

The graphical display of an event uses a line for marking the date while the text itself is a label. A plan has the form of a staple, with begin and end line markers and both joined.

- Load the package  
`Economics [ShowPlan]`

## 3.4.2 Examples

If we live on January 1 2000 and we have a 7 year deferred biannual perpetuity of \$500, then the present value can be calculated in these steps: first we calculate the present value at the moment of deferment (year 7), and then we discount to year 0. The timeline is this:



A 10% bond that pays annual interest and matures on September 15, 2000 has a redemption (or par) value of \$1,000. With the settlement day differing from maturity, the bond is actually a plan. Regard the following portfolio database that includes a date on which it will be discussed whether the owner will switch from one bond to another.

```
buyDate = F[9, 15, 1998];
maturity = F[9, 15, 2000];
switchDate = F[7, 30, 1999];
bond = Bond[{10. Percent, maturity, 1000. Dollar, 1}];
bon2 = Bond[{10.5 Percent, F[1, 5, 2001], 1000. Dollar, 1}];
item1 = BondToPlan[bond, buyDate] /. x_String -> "First bond"
item2 = BondToPlan[bon2, F[12, 15, 1998]] /. x_String -> "Second bond"
item3 = {"switch?", FromF[switchDate]}
portfolioDatabase = {item1, item2, item3};

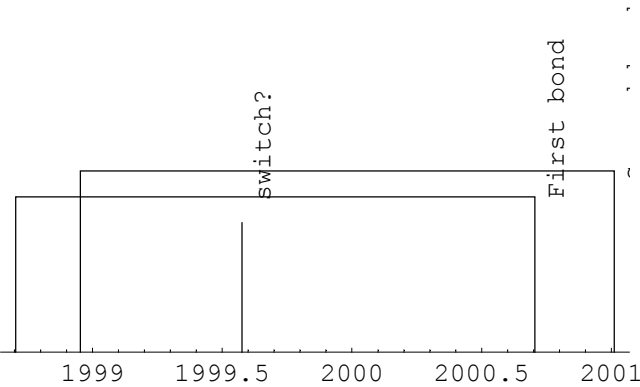
{{First bond, {2000, 9, 15, 0, 0, 0}}, {1998, 9, 15, 0, 0, 0}}

{{Second bond, {2001, 1, 5, 0, 0, 0}}, {1998, 12, 15, 0, 0, 0}}

{switch?, {1999, 7, 30, 0, 0, 0}}
```

ShowEventsOrPlans[portfolioDatabase,  
"Portfolio decisions"]

Portfolio decisions



3.4.3 Main routines

The main routines recognise the different objects, and allow for various date formats.

ShowEventsOrPlans[x, text\_String]

shows database x, with text as plotlabel, using EventsOrPlansToGOs

TimeLine[x, opts]

shows a database of events, with AspectRatio → 1 / 8

3.4.4 Subroutines

For events separately:

<code>ShowEvent [x_List, h:5, j:3, opts___Rule]</code>	shows event {text,date}, h = height of marker, j = position text above that
<code>ShowEvents [database_List, h:5, j:3, opts___Rule]</code>	shows list of events, h = height of markers, j = position texts above that

For plans separately:

<code>ShowPlan [x_List, h:5, j:3, opts___Rule]</code>	show plan {{text,date}, date2}, h = height of marker, j = position text above that
<code>ShowPlans [step:1, database_List, h:5, j:3, opts___Rule]</code>	shows list of plans, h = height of markers, j = position texts above that, step = step rate for horizontal lines

For the transformation into graphics objects (input for Show[ ]):

<code>EventToGO [x_List, h:5, j:3]</code>	makes a Graphics object of an event {text, date}, h = height of marker, j = position text above that
<code>PlanToGO [y_List, h:5, j:3]</code>	makes a Graphics object of a plan {{text, date}, date2}, h = height of marker, j = position text above that
<code>EventsOrPlansToGOs [step:1, x_List, h:5, j:3]</code>	makes a list of graphics objects of x, h = height of markers, j = position texts above that, step = step rate for horizontal lines

Above also calls BondToPlan. If one doesn't have the Finance Essentials Package, then Bond will be in the "Global" context.

<code>BondToPlan [bond, date]</code>	gives a plan {{bond, maturity}, date} where the input dates are in financial and the output dates in standard format
--------------------------------------	--

# 3.5 Easy construction of arrow diagrams

## 3.5.1 Summary

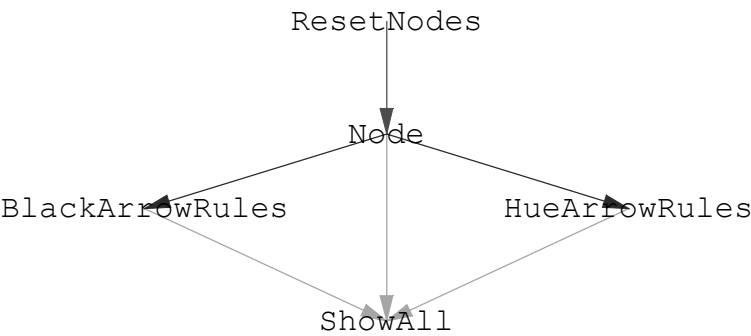
An arrow diagram gives nodes, connections and flow directions. This kind of diagram is used in decision analysis, but also for merely explaining a process. The package allows control over positions, labels, colour, and also facilitates the plotting of paths.

- Load the package  
`Economics [Arrowise]`

Note: Steve Skiena's great package `DiscreteMath`Combinatorica`` implements Graph Theory, and you could use its directed graphs. The current package `Arrowise`` however seems more flexible graphically. Robert Korsan in Varian (1993) creates arrow diagrams too, but does not allow you to select locations.

## 3.5.2 Example

The following diagram explains the process of creating an arrow diagram. A diagram consists of nodes and arrows. The arrows are defined by using the labels of the nodes, as  $\{\text{label}_i \rightarrow \text{label}_j, \dots\}$ . The diagram can be shown using Graphics options. The code for this diagram is reproduced below.



## 3.5.3 User steps

ArrowiseQ	ArrowiseQ tells you how to use the Arrowise package
-----------	---

## ArrowiseQ

Use the package as follows:

- 1) Start with `ResetNodes`.
- 2) Then you give `Node[label1, Text[...1...]]`; `Node[label2, Text[...2...]]`; .... statements. Each statement associates a label with its text, and fills the `NodeLabels` and `Nodes` memory. Text is the normal Mathematica Graphics primitive.
- 3) Then you may give two kinds of lists of rules:
 

```
BlackArrowRules = {labeli -> labelj, ... }
HueArrowRules[r] = {labelk -> labeln, ....}
```

 Such rule statements  $li \rightarrow lj$  will later be translated in arrows for the associated nodes. Here,  $r$  will be a real number from 0 till 1, giving a hue. (See `Rainbow`.)
- 4) The results can be shown by `ShowAll`, or separately by `ShowBlackArrows`, `ShowHueArrows[r]` or `[[r1, r2,...]]`, and finally `ShowText`. Options in these commands are passed on to `Show`.

These steps are also supported by the specific palette `TEP_Arrowise.nb`.

- Creating above arrow diagram:

```
ResetNodes; BlackArrowRules = {};
Node[res, Text["ResetNodes", {0, 5}]];
Node[nd, Text["Node", {0, 2}]];
Node[bar, Text["BlackArrowRules", {-3, 0}]];
Node[har, Text["HueArrowRules", {Plus[3], 0}]];
Node[sh, Text["ShowAll", {0, -3}]];

HueArrowRules[0] = {res -> nd};
HueArrowRules[0.7] = {nd -> bar, nd -> har};
HueArrowRules[0.3] = {nd -> sh, har -> sh, bar -> sh};

ShowAll[{0, 0.3, 0.7}, TextStyle -> FontSize -> 12,
PlotRange -> All]
```

### 3.5.4 The nodes

<code>ResetNodes</code>	<code>ResetNodes</code> sets <code>NodeLabels</code> and <code>Nodes</code> to {}
<code>Node[x, Text[...]]</code>	associates label $x$ with the <code>Text</code> item
<code>NodeLabels</code>	list of the <code>Node</code> labels, corresponding to <code>Nodes</code>
<code>Nodes</code>	list of <code>Text</code> elements <code>{Text[...], Text[...], ...}</code> , corresponding to <code>NodeLabels</code>

3.5.5 The arrows

<code>BlackArrow[x, y]</code>	gives a Graphics primitive for a black arrow from <code>x[[2]]</code> to <code>y[[2]]</code>
<code>BlackArrowRules</code>	list of rules that will become black arrows
<code>HueArrow[x, y, r]</code>	Graphics primitive. An arrow from <code>Node[x][[2]]</code> to <code>Node[y][[2]]</code> with Hue[r].
<code>HueArrowsRules[r]</code>	is a list of rules that will become arrows with Hue r

3.5.6 Show the results

<code>ShowAll[r_List: {}, opts]</code>	submits the data in memory to Show. The various Hues may be given in a list. Text is shown last, thereby printing over
<code>ShowBlackArrows[opts]</code>	submits BlackArrows to Show
<code>ShowHueArrows[r, opts]</code>	submits the hue arrow list to Show. If r is a list of hue values, then this is mapped
<code>ShowText[opts]</code>	submits Nodes to Show

3.5.7 Paths

When the co-ordinates have meaningful values, for example price and quantity, then the development path of subsequent values can be dramatised by using arrows. You can include equal contour lines of the value  $v = p * q$ , or as addition in logarithms  $\text{Log}[v] = \text{Log}[p] + \text{Log}[q]$ .



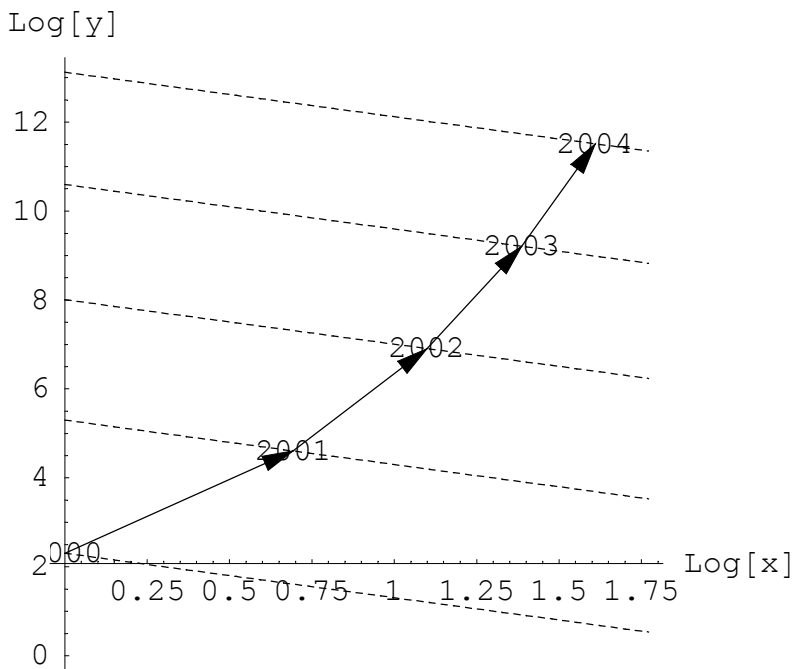
`ShowPath[x_List, y_List, (t_List,) (r_,) opts___Rule]`

gives a path of chained arrows for lists of x and y co-ordinates,  
with possibly t text expressions,  
and hue r ( $0 \leq r \leq 1$ ), and with opts passed on to Show

`ShowPath[Log, x_List, y_List, (t_List,) opts___Rule]`

gives a  $\{\text{Log}[x], \text{Log}[y]\}$  plot, with loglinear contours of  $x * y = \text{constant}$

```
ShowPath[Log, {1, 2, 3, 4, 5},
           {10, 100, 1000, 10000, 100000},
           Range[2000, 2004],
           TextStyle -> {FontSize -> 12},
           AxesLabel -> {"Log[x]", "Log[y]"}]
```



## 3.6 Matrices

---

### 3.6.1 Summary

This section supports the matrix manipulation of the major packages for estimation and optimisation. It generally extends on *Mathematica's* package `LinearAlgebra`MatrixManipulation`` and section A.4.8 below. Since some applications to calculus give more insight, we already call that package too.

`Economics[Matrix, Calculus]`

### 3.6.2 Block diagonal matrices

The routine `BlockDiagonal` is used in the `Estimate`` package for systems of equations.

`BlockDiagonal[x__Vectors | list of these, opts__Rule]`

either x is a single list (matrix) or x is a sequence of various lists.  
Default is `Number → Automatic`, and can be set to `Number → n`

- Various combinations are:

`BlockDiagonal[{1}]`

$\{1\}$

`BlockDiagonal[{1}, Number → 3]`

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

`BlockDiagonal[{1}, {2}, Number → 3] // Transpose`

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

`BlockDiagonal[{1}, {2}, {3}, Number → 3]`

$$\left\{ \text{Cannot be processed, } \begin{pmatrix} \{1\} & \{1\} & \{1\} \\ \{1\} & \{1\} & \{1\} \\ \{1\} & \{1\} & \{1\} \end{pmatrix} \right\}$$

`BlockDiagonal[{{1}, {2}, {3}}, Number → 3] // Transpose`

$$\begin{pmatrix} 1 & 2 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \end{pmatrix}$$

### 3.6.3 Definiteness

A quadratic expression can be represented by a symmetric matrix. These have the property that all eigenvalues are real. If all (real parts of the) eigenvalues of a matrix are positive then the matrix is called positive definite. If some are zero too, then it is positive semi definite. If all are negative then it is negative definite, and if some are zero too, then it is negative semi definite. If none of these apply then the matrix is called 'not definite'. Thus when all are zero, then the matrix is both negative and positive semi definite.

Quadratic expressions are key to optimisation. A condition for an optimum is that the Hessian - the matrix of the second order derivatives - is negative or positive semi definite, for concave resp. convex functions. Note that a test on symmetry of a Hessian is not required, though a symmetry test remains a useful tool and is included below. More important is the determination of the type of definiteness. There are basic theorems on this. We first discuss definiteness for normal matrices and below definiteness for bordered ones, after explaining what is meant by a bordered matrix.

Though *Mathematica* is able to determine the eigenvalues also in symbolic form, those expressions can get quite complicated, so the routine below contains some control on this.

<code>Definiteness[mat, opts]</code>	determines the definiteness of a square matrix
<code>SuccPrincipalMinors[mat]</code>	gives the successive principal minors of mat
<code>SymmetricMatrixQ[mat_List]</code>	tests whether mat is a symmetric matrix

Note: Option Number  $\rightarrow n$  controls the maximum matrix size for symbolic matrices (default 4). Option SymmetricMatrixQ (default Do) controls the test whether the matrix indeed is symmetric. With setting {Do, Return}, the routine stops if the matrix is not symmetric. Option Re (default True) controls taking the real part of the eigenvalues only, which does not apply when SymmetricMatrixQ  $\rightarrow$  True. Option Chop (default True) controls chopping of eigenvalues before their sign is determined. Note: Definiteness[Check, m, opts] provides a test on these topics, and is called by various routines for optimisation.

- By taking the Union of the signs, we are left with only a few possible situations.

**Definiteness[{a, b}, {b, c}]**

```
Hold[Switch][Hold[Union][{sgn(a + c -  $\sqrt{a^2 - 2ca + 4b^2 + c^2}$ ), sgn(a + c +  $\sqrt{a^2 - 2ca + 4b^2 + c^2}$ )},
{1}, {Positive, Definiteness}, {0, 1}, {Positive, Semi, Definiteness}, {-1, 0},
{Negative, Semi, Definiteness}, {-1}, {Negative, Definiteness}, {0},
{All, 0, i.e., Negative, And, Positive, Semi, Definiteness}, {-1, 0, 1}, {In, Definiteness}, _, $Failed]

% /. {a  $\rightarrow$  1, b  $\rightarrow$  -1, c  $\rightarrow$  3} // ReleaseHold

{Positive, Definiteness}
```

- The Definiteness routine directly uses the eigenvalues. Another approach is to look at the successive principal minors.

**SuccPrincipalMinors**[{**a**, **b**}, {**b**, **c**}]

{*a*, *a c* - *b*<sup>2</sup>}

### 3.6.4 Bordered matrices and Successive Bordered Minors

The idea of a bordered matrix is best shown by a calculus example.

<b>ToBorderedMatrix</b> [ <i>h_?SquareMatrixQ</i> , <i>g_?MatrixQ</i> , <i>opts</i> ]  <b>ToBorderedMatrix</b> [ <i>h_?SquareMatrixQ</i> , <i>g_?VectorQ</i> , <i>opts</i> ]	gives the matrix   uses { <i>g</i> } in above
--	--

Note: With option *Reverse* → *True* the zero's are in the lower right corner. Note: Input requires that the number of columns of *h* and *g* are the same, as is the case with output of *Hess* and {*Gra*}.

**bm = ToBorderedMatrix**[**Hess**[**f**[**x**, **y**], {**x**, **y**}], **Gra**[**f**[**x**, **y**], {**x**, **y**}]

$$\begin{pmatrix} f^{(2,0)}(x, y) & f^{(1,1)}(x, y) & f^{(1,0)}(x, y) \\ f^{(1,1)}(x, y) & f^{(0,2)}(x, y) & f^{(0,1)}(x, y) \\ f^{(1,0)}(x, y) & f^{(0,1)}(x, y) & 0 \end{pmatrix}$$

<b>FromBorderedMatrix</b> [ <i>mat</i> , <i>m_Integer</i> , <i>opts</i> ]	decomposes, using border size <i>m</i>
--	--

Note: The routine takes the column border submatrix. Also, *Options*[*ToBorderedMatrix*] control the *Reverse* option. Output uses rules with *Center* and *Border*.

**FromBorderedMatrix**[**bm**, 1]

$$\left\{ \text{Reverse} \rightarrow \text{True}, \text{Center} \rightarrow \begin{pmatrix} f^{(2,0)}(x, y) & f^{(1,1)}(x, y) \\ f^{(1,1)}(x, y) & f^{(0,2)}(x, y) \end{pmatrix}, \text{Border} \rightarrow (f^{(1,0)}(x, y) \ f^{(0,1)}(x, y)) \right\}$$

The successive 'bordered minors' are those required for the next section. (They conform to theorem 1.E.17 in Takayama (1974:126) or theorem M.D.3 in Mas-Colell (1995:938).)

<b>SuccBorderedMinors</b> [ <i>mat_?SquareMatrixQ</i> , <i>m_Integer</i> , <i>opts</i> ]	for a border of size <i>m</i>
<b>SuccBorderedMinors</b> [ <i>h_?SquareMatrixQ</i> , <i>g_?MatrixQ</i> , <i>opts</i> ]	from explicit <i>h</i> and <i>g</i>

Note: The routine identifies where the zero submatrix is. If *Reverse* → *True* (default; controlled by *Options*[*ToBorderedMatrix*]) then the routine can use *-n* to identify the position of the zero submatrix if also the top has a zero submatrix. Note: The routine uses the *Options*[*Definiteness*] for *Number* and *SymmetricMatrixQ* options. Note: Use the routine *BorderedDefiniteness*[ ] to evaluate the sign of these minors.

- This reproduces Mas-Colell et. al. (1995:939). It is just the determinant.

**SuccBorderedMinors**[bm, 1]

$$\{-f^{(2,0)}(x, y) f^{(0,1)}(x, y)^2 + 2 f^{(1,0)}(x, y) f^{(1,1)}(x, y) f^{(0,1)}(x, y) - f^{(0,2)}(x, y) f^{(1,0)}(x, y)^2\}$$

### 3.6.5 Definiteness for bordered matrices

The second order conditions of an optimum under restrictions are expressed in terms of the successive 'bordered minors' of the bordered Hessian.

**BorderedDefiniteness**[mat\_?SquareMatrixQ, m\_Integer <> 0, opts]

determines whether the matrix with border size m<>0  
is negative definite (under the restrictions of the border).  
If so, and if the matrix is the Bordered Hessian of a function,  
then that function is strict quasi concave. If semi definite, then not strictly so.

**BorderedDefiniteness** [ on h and g  
h\_?SquareMatrixQ, g\_?MatrixQ, opts]

**BorderedDefiniteness**[s\_?VectorQ,  
n\_Integer, m\_Integer, numq\_Symbol: Automatic]

uses the SuccBorderedMinors result s, for n normal variables and m constraints.  
Numq can be entered True or False  
if one already knows whether s is numeric or not

Note: The routine uses Options[ToBorderedMatrix] for the Reverse options, and input m can be negative in relation to this. Note: The routine uses Options[Definiteness] for Number and SymmetricMatrixQ options.

Just to restate the textbooks:

1. First the successive bordered minors s are determined.
2. These are premultiplied with  $(-1)^r$  giving  $(-1)^r s(r)$ , for  $r = m+1, \dots, \text{Length}[\text{mat}]-m$
3. The results should all be positive to conclude to negative definiteness (or with some zero for semi definiteness).
4. A direct test on positive definiteness uses the  $(-1)^m s$  similarly.

An example works wonders here (though it would be something for the calculus section). This example also shows that a bordered matrix easily arises from the Hessian of a Lagrangian, so that it indeed is useful to have routines 'to' and 'from' such bordered matrices.

- Suppose that a donkey earns \$10 per hour and requires straw at \$12 per kilogram. Available hours for work are 10, so that leisure is  $10 - y$ . Assume a Cobb-Douglas function with equal weights for leisure and food.

$$\text{Lagrangian}[\mathbf{x}_-, \mathbf{y}_-, \lambda_-] = \mathbf{x}^{1/2} (10 - \mathbf{y})^{1/2} + \lambda (10 \mathbf{y} - 12 \mathbf{x}) ;$$

- The first order conditions are:

$$\text{sol} = \text{Foc}[\text{Apply}, \text{Lagrangian}, \{\mathbf{x}, \mathbf{y}, \lambda\}]$$

$$\left\{ \left\{ x \rightarrow \frac{25}{6}, y \rightarrow 5, \lambda \rightarrow \frac{1}{4\sqrt{30}} \right\} \right\}$$

- The Hessian appears to be a bordered matrix.

$$\mathbf{H} = \text{Hess}[\text{Apply}, \text{Lagrangian}, \{\mathbf{x}, \mathbf{y}, \lambda\}]$$

$$\begin{pmatrix} -\frac{\sqrt{10-y}}{4x^{3/2}} & -\frac{1}{4\sqrt{x}\sqrt{10-y}} & -12 \\ -\frac{1}{4\sqrt{x}\sqrt{10-y}} & -\frac{\sqrt{x}}{4(10-y)^{3/2}} & 10 \\ -12 & 10 & 0 \end{pmatrix}$$

- The bordered Hessian evaluated at the first order solution point must be negative (semi) definite for a maximum.

$$\text{mat} = \mathbf{H} /. \text{sol}[[1]] // \mathbf{N}$$

$$\begin{pmatrix} -0.0657267 & -0.0547723 & -12. \\ -0.0547723 & -0.0456435 & 10. \\ -12. & 10. & 0. \end{pmatrix}$$

$$\text{SuccBorderedMinors}[\text{mat}, 1]$$

$$\{26.2907\}$$

$$\text{BorderedDefiniteness}[\text{mat}, 1]$$

$$\{\text{Negative}, \text{Definiteness}\}$$

## 3.7 Calculus, Convexity, Lagrange and Kuhn-Tucker

---

### 3.7.1 Summary

The `Calculus`` package contains a number of basic tools. The `Convex`` package gives basics for convexity and concavity. The `Lagrange`` and `KuhnTucker`` packages allow symbolic and numerical methods for constrained optimisation with Lagrange multipliers.

- Load the packages

```
Economics[Calculus, Convex, Lagrange, KuhnTucker]
```

### 3.7.2 Inverse function

*Mathematica's* `InverseFunction` is not so strong.

<code>InverseF[f, x_Symbol]</code>	finds the inverse of f for x. Input can be f = g[x], a pure function or an expression containing x
------------------------------------	---

```
g[x] := a x + b
```

```
InverseFunction[g][g[x]]
```

$$g^{(-1)}[b + a x]$$

```
ginv[g_] = InverseF[g, x]
```

$$-\frac{b-g}{a}$$

### 3.7.3 Gradient, Hessian and Jacobian

*Mathematica* does not provide built-in functions for the gradient, Hessian and Jacobian - and probably for the reason that these are easy to program. Still, users like to have these functions available, for example since calling on these names also communicates better what one is doing. We use the originator's names `Hess` and `Jacobi` to prevent name conflicts with other packages.

<code>Gra[f[x], {x1, ..., xn}]</code>	gives the gradient, i.e. $\partial f / \partial x$ . For consistency also defined for a single x. <code>Gra[f, x <math>\rightarrow</math> y]</code> evaluates at y
<code>Hess[f[x], {x1, ..., xn}]</code>	gives the Hessian matrix of $\partial^2 f / (\partial x_i \partial x_j)$ . For consistency also defined for a single x. <code>Hess[f, x <math>\rightarrow</math> y]</code> evaluates at y
<code>BorderedHess[f[x], {x1, ..., xn}]</code>	gives a larger matrix with the Hessian as the center and the gradient as the border
<code>Jacobi[{f1, ..., fn}, {x1, ..., xn}]</code>	gives the Jacobian of the fi with respect to the xi. If option <code>Print <math>\rightarrow</math> True</code> , then the matrix is printed. This matrix can always be found in <code>Results[Jacobi]</code>

Note: See `Calculus`VectorAnalysis`` and notice the differences. Note: `Hess[]` uses symmetry, and composes the matrix from the upper triangle and diagonal. This is not only faster for larger problems but also makes that the result satisfies `SymmetricMatrixQ`.

Note: For  $F \in \{Gra, Hess, BorderedHess\}$  two features have been added. (1) `F[f[x], x  $\rightarrow$  a]` sets x to that value afterwards. (2) If `f[x_]` is defined as a function then you can enter `F[Apply, f, {x, ...}]` so that you don't have to type all x's twice.

An application is in:

<code>ImplicitD[f_List, x_List, xi_Symbol, v_Symbol]</code>
solves <code>D[xi, v]</code> using the implicit function theorem when f is the list of functions, x are the dependent variables, xi is selected from x, and v is one of the independent variables. See <code>Jacobi</code> for input and <code>Results[ImplicitD]</code> for output details

Another application is in:

<code>Foc[f[x], x]</code>	gives the first order condition $f' = 0$ , for x a Symbol or a list of them
<code>Foc[Solve, f[x], x]</code>	solves for unknown x
<code>SoZero[f[x], x]</code>	equals the 2 nd order derivative to zero, $f'' = 0$ , for x a Symbol or such list
<code>SoZero[Solve, f[x], x]</code>	solves for unknown x

Note: The `SoZero` finds only points of inflection. The second order conditions for local extreme values are not the `SoZero`.

Note: If `f[x_]` is defined as a function then you can `Solve` by entering `Foc[Apply, f, {x, ...}]` or `SoZero[Apply, f, {x, ...}]` so that you don't have to type all x's twice.



- Let us regard Buridan's Ass. This is the donkey that is placed right in the middle between two equal stacks of hay, and that dies for not being able to choose which to start eating from.

$$\text{Lagrangian}[\mathbf{x}_-, \mathbf{y}_-, \lambda_-] = (\mathbf{x} + \mathbf{y})^{1/2} + \lambda (1 - \mathbf{x} + 1 - \mathbf{y})$$

$$(-x - y + 2)\lambda + \sqrt{x + y}$$

$$\text{sol} = \text{Foc}[\text{Solve}, \text{Lagrangian}[\mathbf{x}, \mathbf{y}, \lambda], \{\mathbf{x}, \mathbf{y}, \lambda\}]$$

$$\{\{\lambda \rightarrow \frac{1}{2\sqrt{2}}, x \rightarrow 2 - y\}\}$$

- Since we have restrictions, the Hessian already is a bordered matrix.

$$\mathbf{H} = \text{Hess}[\text{Lagrangian}[\mathbf{x}, \mathbf{y}, \lambda], \{\mathbf{x}, \mathbf{y}, \lambda\}]$$

$$\begin{pmatrix} -\frac{1}{4(x+y)^{3/2}} & -\frac{1}{4(x+y)^{3/2}} & -1 \\ -\frac{1}{4(x+y)^{3/2}} & -\frac{1}{4(x+y)^{3/2}} & -1 \\ -1 & -1 & 0 \end{pmatrix}$$

$$\text{mat} = \mathbf{N}[\mathbf{H} /. \text{sol}[[1]]]$$

$$\begin{pmatrix} -0.0883883 & -0.0883883 & -1. \\ -0.0883883 & -0.0883883 & -1. \\ -1. & -1. & 0. \end{pmatrix}$$

- The following shows that we have a local maximum. But, indeed, not unique. Which to take ?

$$\text{BorderedDefiniteness}[\text{mat}, 1]$$

$$\{\text{All}, 0, \text{i.e., Negative, And, Positive, Semi, Definiteness}\}$$

Another application can be seen above in section 3.6.3 and following. There we combine calculus results with matrix results.

3.7.4 Standard analysis of a real function

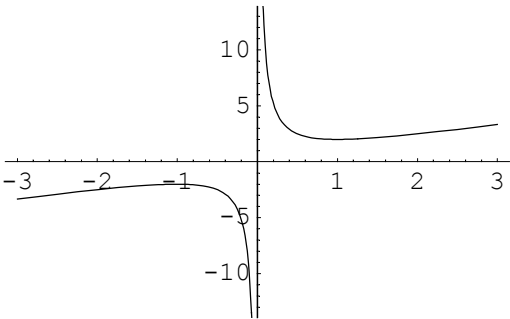
`FunctionAnalysis[f_Symbol, x_Symbol]`  
  
identifies vertical asymptots, increasing and decreasing sections and 2 nd order properties  
  
`FunctionAnalysis[Table]`

gives the sign table using the data from  
FADomain[f] and Results[FunctionAnalysis]

- Take a typically difficult function.

```
f[x_] := x + 1/x
```

```
Plot[f[x], {x, -3, 3}];
```



- The routine requires a domain statement. Let us limit the domain here.

```
Domain[f] = Interval[{-∞, 10}, {100, 200}];
```

- This gives the begin and end values of interval sections, chosen points, the value of the function at those points, and the signs of the first and second derivatives. The font size is reduced here to fit the table on the page.

```
FunctionAnalysis[f, x];
```

```
FunctionAnalysis[Table, TableSpacing -> {1, 1}]
```

	Begin	End	Point	Value	Gra(Sign)	Hess(Sign)
1	$-\infty$	-1.	-500001.	-500001.	1	-1
2	-1	-1	-1	-2	0	-1
3	-0.999999	$-1. \times 10^{-6}$	-0.5	-2.5	-1	-1
4	0	0	0	ComplexInfinity	Indeterminate	Indeterminate
5	$1. \times 10^{-6}$	0.999999	0.5	2.5	-1	1
6	1	1	1	2	0	1
7	1.	10	5.5	5.68182	1	1
8	100	200	150	$\frac{22501}{150}$	1	1

- Since symbolic methods are used, there is more likelihood of finding crucial points. The point of inflection in the next function would not be found if Sqrt[3] was in real format. (Not evaluated.)

```
g[x_] := 1/f[x]; Domain[g] = Domain[f]; Plot[g[x], {x, -5, 5}];
FunctionAnalysis[g, x]
```

Subroutines are:

FunctionAnalysis[Set, f_Symbol]	uses the first order conditions and vertical asymptots to split Domain[f] into FADomain[f]
FunctionAnalysis[Point, f_Symbol]	selects points in those intervals
FunctionAnalysis[D, f_Symbol, x_Symbol]	gives the first and second derivative at those points and their signs
FADomain[f]	holds the domain Interval statement for a function that is subject to FunctionAnalysis[]
VerticalAsymptot[f, x_Symbol]	determines the x for which there is such an asymptot

3.7.5 Convexity and concavity

If a function f[x] is concave then -f[x] is convex. Since economics is more interested in concave shapes, the following routines take concavity as the norm and define convexity on this.

3.7.5.1 Definition

The following definition of (strict, quasi) concavity also allows for less-well-behaved functions. Note that only three points are compared. The f[x] can only be considered concave for the *whole* domain {b<sub>d</sub>, e<sub>d</sub>} if the condition holds for all 0 ≤ h ≤ 1 (or inequality for strictness) and arbitrary b<sub>d</sub> ≤ b < e ≤ e<sub>d</sub> in that domain.

DefinitionConcaveQ[f[x], {x_Symbol, b, e}, h]	
DefinitionConcaveQ[Strict, f[x], {x_Symbol, b, e}, h]	
DefinitionConcaveQ[Quasi, f[x], {x_Symbol, b, e}, h]	
DefinitionConcaveQ[Strict, Quasi, f[x], {x_Symbol, b, e}, h]	
various forms of concavity for a single variable. The expression tested would be f[x], b and e are the begin and end of a domain, and the test point is $h b + (1 - h) e$ .	
DefinitionConcaveQ[f[x], {{x__Symbol}}, b_List, e_List], h]	
DefinitionConcaveQ[Strict, f[x], {{x__Symbol}}, b_List, e_List], h]	
DefinitionConcaveQ[Quasi, f[x], {{x__Symbol}}, b_List, e_List], h]	
DefinitionConcaveQ[Strict, Quasi, f[x], {{x__Symbol}}, b_List, e_List], h]	
	idem for vectors
Strict	Symbol
Quasi	Symbol
DefinitionConvexQ[...]	uses the same input formats as DefinitionConcaveQ. In fact, it uses that routine, while replacing with -f[x]
NLocalConcaveQ[ f[x], x, x0, eps]	is DefinitionConcaveQ[f[x], {x, x0-eps, x0+eps}, 1/2]
NLocalConvexQ[ f[x], x, x0, eps]	similar DefinitionConvexQ. Default eps = 10^-6

■ The following are examples. The condition on h must be read as "for all h in".

**DefinitionConcaveQ[G[x], {x, b, e}, h]**

$$0 \leq h \leq 1 \wedge G(e(1 - h) + b h) \geq h G(b) + (1 - h) G(e)$$

**DefinitionConcaveQ[Quasi, G[x], {x, b, e}, h]**

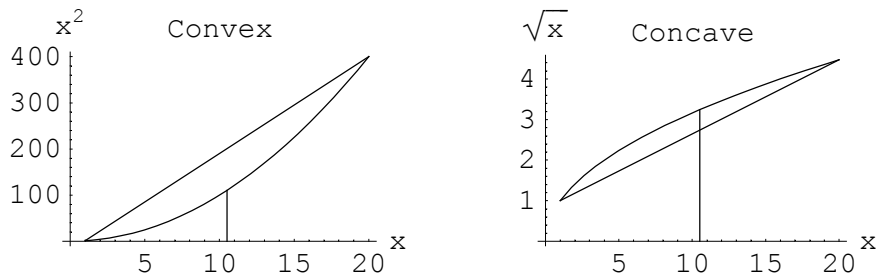
$$0 \leq h \leq 1 \wedge G(b) \geq G(e) \Rightarrow G(e(1 - h) + b h) \geq G(e)$$

3.7.5.2 Example plot

ConvexPlot[f[x], {x\_Symbol, b, e}, h::.5, opts]

plots f[x] over the domain {b, e} for x with a vertical line at  $b\,h + (1-h)\,e$  and a line connecting the values at b and e.  
If the b–e–line is above (below) the value at  $x = b\,h + (1-h)\,e$ , there might be convexity (concavity)  
(no certainty since only three points are looked at).

```
Show[GraphicsArray[{ConvexPlot[x^2, {x, 1, 20}, AxesLabel -> {"x", "x^2"},  
  PlotLabel -> "Convex", DisplayFunction -> Identity],  
  ConvexPlot[Sqrt[x], {x, 1, 20}, AxesLabel -> {"x", "Sqrt[x]"},  
  PlotLabel -> "Concave", DisplayFunction -> Identity}]],  
DisplayFunction -> $DisplayFunction];
```



3.7.5.3 Functions in the real domain

For real functions we can translate the definitions into equivalent conditions in terms of either first or second derivatives. We can perform an function analysis as discussed above in section 3.7.4, and get homogeneous intervals so that the use of a single point is sufficient to say something about the whole interval.

This gives the condition for the first derivative.

DConcaveQ[f[x], {x_Symbol, b, e}]	gives the condition for concavity using f' at b
DConcaveQ[Strict, f[x], {x_Symbol, b, e}]	for strict concavity
DConvexQ[f[x], {x_Symbol, b, e}]	for convexity
DConvexQ[Strict, f[x], {x_Symbol, b, e}]	for strict convexity

```
DConcaveQ[F[x], {x, b, e}]
```

$$F'(b) \geq \frac{F(e) - F(b)}{e - b}$$

The functional analysis is best done with the second derivative included.

D2ConcaveQ[f_Symbol, x_Symbol]	perform a FunctionAnalysis on f ' and return True if the sign of f '' is -1 or 0 in all subintervals
D2ConcaveQ[Strict, f_Symbol, x_Symbol]	strictness allows only -1
D2ConvexQ[f_Symbol, x_Symbol]	create a dummy function g[x] = -f[x] with the same domain and submit this to ContinuousConcaveQ
D2ConvexQ[Strict, f_Symbol, x_Symbol]	idem

Note that input now is f and not f[x]. Note that each call performs the whole FunctionAnalysis, since you may redefine the Domain[f]

- Recall above function, and let us redefine the valid domain.

```
f[x_] := x + 1/x; Domain[f] = Interval[{1, 200}];
```

- These tests give us (for this domain!):

```
D2ConcaveQ[f, x]
D2ConvexQ[Strict, f, x]
```

False

True

ArrowPratt[u[x], x]	gives the Arrow-Pratt measure -u''/u' of utility function u[x]. This is a measure of concavity that is independent of linear transformations of u[x]. Normally u[x] is rising, so that u' > 0. See the discussion on risk.
---------------------	--

```
ArrowPratt[a Utility[x] + b, x]
```

$$-\frac{\text{Utility}''(x)}{\text{Utility}'(x)}$$

3.7.5.4 Using 3 Points in the real domain

The continuously differentiable real functions above are one example where the use of a single point or actually three points suffices. Use of a few points still might work for other discontinuous and nondifferentiable cases. Your judgement is required here. Alternatively, you can use a few points to *disprove* convexity or concavity.

The following is a quick and dirty method that uses Solve or FindRoot to find the  $h$  that causes an equality ( $=$ ) in the concavity or convexity definition. If there are no points of intersection or touching, then only  $h = 0$  or  $h = 1$  result, and all other function values are above or below the line connecting  $b$  and  $e$ . Of course, there is no guarantee that those function values are not crooked, but, as said, your own judgement comes in here.

```
N3PointsConcaveQ[f[x], {x_Symbol, b, e}, h:0.5]
```

is a quick and dirty numerical test whether real valued  $f[x]$  is concave in symbol  $x$  over the range  $[b, e]$ . DefinitionConcaveQ is tried at point  $h b + (1-h) e$ , and there the search for other intersection or touching points starts

```
N3PointsConvexQ[f[x], {x_Symbol, b, e}, h:0.5]                      analogue
```

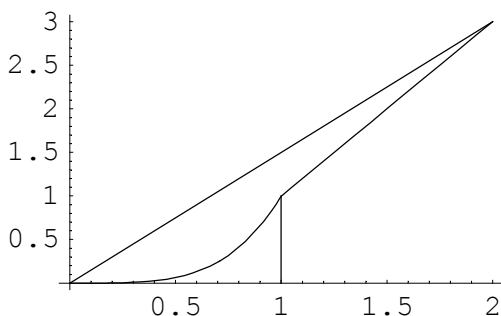
```
NConvexFind[f[x], {x_Symbol, b, e}, h:0.5]
```

tests N3PointsConvexQ and/or N3PointsConcaveQ for the domain  $[b, e]$  for  $x$ . The point  $h b + (1-h) e$  is tested with the definition, and there the search starts for other intersection or touching points. (Note: The result might be wrong if there is touching only.)

Other drawbacks apart from the 'quick and dirtiness' are that the routine does not handle infinity and that it doesn't consider vectors.

- Suppose that a student thinks that joining two convex functions gives a new one.

```
ConvexPlot[If[x ≤ 1, x4, 2 x - 1], {x, 0, 2}, PlotRange → All];
```



- In this case continuous methods don't really work. But we can find points that disprove convexity or concavity. The following disproves convexity.

```
NConvexFind[If[x ≤ 1, x4, 2 x - 1], {x, 0.75, 2}]
```

```
{N3PointsConcaveQ → True, N3PointsConvexQ → False, Indeterminate → False}
```

### 3.7.5.5 Quasi concavity for multivalued twice continuously differentiable functions

The following combines some routines of the section 3.6 on Matrices. The combination of the routines on gradients and Hessians of this section plus the matrices routines is a strong one. We should not define too many combinations when these might not be needed or can simply be composed. Quasi concavity for multivalued functions can be used as an example.

```
ConcaveFind[Quasi, f_Symbol, {x__Symbol}]
```

is another word for BorderedDefiniteness[Hess[f[x], {x}], Gra[f[x], {x}]]. The theorem is that f is quasi concave iff the latter result is negative semi definite. (And strict if definite.)

Note: The theorem referred to is e.g. Mas-Colell (1995:935) theorem M.C.4.

- The following function appears to have bordered minors that are all zero, and hence *ft* is quasi concave.

```
ft[x_, y_] := (x + y)^3

ConcaveFind[Quasi, ft, {x, y}] // ReleaseHold

{Table → {1}, Negative → {All, 0}, Positive → {All, 0}}
```

### 3.7.6 Substitute h[x] back in h'[x]

Substituting a function h[x] back into its derivative h'[x] produces h'[x] = g[x, h[x]], and this can give neater expressions of derivatives.

#### 3.7.6.1 Example

We take the example of the Constant Elasticity of Substitution (CES) function.

```
ces = A (α capital^(1 - 1/σ) +
        β labour^(1 - 1/σ))^(σ/(1 - 1/σ))
```

$$A \left( \alpha \text{capital}^{1-\frac{1}{\sigma}} + \text{labour}^{1-\frac{1}{\sigma}} \beta \right)^{\frac{\sigma}{1-\frac{1}{\sigma}}}$$

```
D[ces, capital]
```

$$A \text{capital}^{-1/\sigma} \sigma \alpha \left( \alpha \text{capital}^{1-\frac{1}{\sigma}} + \text{labour}^{1-\frac{1}{\sigma}} \beta \right)^{\frac{\sigma}{1-\frac{1}{\sigma}}-1}$$



```
dcesdcap = DRule[ces, Power, capital, Take → First];
MatrixForm[dcesdcap]
```

$$\left( \begin{array}{l} \text{Expression} \rightarrow A \text{ Taken}^{\frac{v}{1-\sigma}} \\ \text{Solve} \rightarrow \left\{ \text{Taken} \rightarrow \left( \frac{\text{Expression}}{A} \right)^{-\frac{1-\sigma}{v\sigma}} \right\} \\ \text{Taken} \rightarrow \alpha \text{ capital}^{1-\frac{1}{\sigma}} + \text{labour}^{1-\frac{1}{\sigma}} \beta \\ D \rightarrow A \text{ capital}^{-1/\sigma} v \alpha \left( \alpha \text{ capital}^{1-\frac{1}{\sigma}} + \text{labour}^{1-\frac{1}{\sigma}} \beta \right)^{\frac{v}{1-\sigma}-1} \\ \text{Out} \rightarrow A \text{ capital}^{-1/\sigma} \left( \left( \frac{\text{Expression}}{A} \right)^{-\frac{1-\sigma}{v\sigma}} \right)^{\frac{v}{1-\sigma}-1} v \alpha \\ \text{Simplify} \rightarrow A^{\frac{\sigma-1}{v\sigma}} \text{ capital}^{-1/\sigma} \text{ Expression}^{\frac{(v-1)\sigma+1}{v\sigma}} v \alpha \end{array} \right)$$

- For the equation  $y == \text{ces}$  we find the partial derivative:

```
y == ces
```

$$y == A \left( \alpha \text{ capital}^{1-\frac{1}{\sigma}} + \text{labour}^{1-\frac{1}{\sigma}} \beta \right)^{\frac{v}{1-\sigma}}$$

```
"∂y" / "∂capital" == Simplify /. Last[dcesdcap] /. Expression → y
```

$$\frac{\partial y}{\partial \text{capital}} == A^{\frac{\sigma-1}{v\sigma}} \text{ capital}^{-1/\sigma} v y^{\frac{(v-1)\sigma+1}{v\sigma}} \alpha$$

### 3.7.6.2 Routines

`DRule[f, g, x, opts]` analyses the expression `D[f[... g[... x ...] ...], x]`. It finds `g[...x...]` with `TakeRule`, takes the derivative of `f[x]`, and substitutes `f` or `g` back into the derivative. If `Solve → True`, then `f[]` itself is substituted, else only `g[]`.

`TakeRule[expr, head, term, opts]`

takes the first subexpression under head (within `expr`) which contains `term`. If option `Take → All` (default), then `Taken` is the subexpression starting with `head`; if `Take → First` then `Taken` is the first subexpression within `head`. If `Solve → True` (default) then `expr` is solved for `Taken`.

`Taken` the label in the output of `TakeRule` and `DRule` for what has been taken

### 3.7.7 Constrained optimisation with equality constraints

The `Lagrange`` package uses equality constraints only. The `KuhnTucker`` package allows for inequalities too. They work in a similar way.

The Lagrange method is very common in economics. The inspiration for the present implementation comes from C. Henry Edwards, "Ladders, Moats, and Lagrange Multipliers", The *Mathematica* Journal, Winter 1994 pp 48-52. What I found particularly useful in Edwards' approach was his choice of the starting point by a least squares technique. I have concentrated on making a user-friendly option interface, and there is still the warning that the routine looks at the first order conditions only.

The routine has a fixed input format or a free rule based input format. The following box gives both formats.

Lagrange[opts]

with for example Expression → f, Variables → x,  
Constraints → g, Routine → ... (, StartValues → x0)

Lagrange[f, g, x, x0\_List: {}, mu\_Symbol: Mu]

Output contains Lagrange multipliers m and has the form:  
for FindRoot: {f[x\*], {x → x\*}, {m → m\*}},  
for Solve: {f[yi\*], {yi → yi\*}} for i solutions,  
where y can be x and m intermixed.

See below for a discussion of the settings.

3.7.7.1 Example

```
ResetOptions[Lagrange];

Fxy = x^2 + y^2;
Gxy = {3 x + 6 + y};

Lagrange[Expression → Fxy, Constraints → Gxy,
  Variables → {x, y}, Routine → Solve]

Lagrange::first: Only first order conditions used

{{18/5, {x → -9/5, y → -3/5, Mu(1) → -6/5}}}
```

3.7.7.2 Symbols

The symbol Mu has no definition attached to it, but it is used in the rules that guide input and output, to indicate the multiplier. Other symbols are:

MultiplierSymbol	Constraints	NumberOfConstraints
Multipliers	NumberOfVariables	Routine

### 3.7.7.3 Routines

Part of the userfriendliness is that you can select the routine that is used for finding the optimum. If you use Solve, the solution would be algebraic. If you use FindRoot, the solution would be numeric. You could use the robust MyFindRoot` of Asahi Noguchi and Silvio Levy, in Hal Varian (ed), "Economic and Financial Modeling with *Mathematica*", Telos 1993.

Routines                                      list of known routines {Solve, FindRoot, MyFindRoot}

Lagrange[... , Routine → routi, ...]                      or

Lagrange[... , Routine → { routj, routi},...]

both call Lagrange[routi, ...],  
where routi first is a format and secondly might be an actual routine. Thus,  
when routj uses the input format of a known routine (and the format only),  
then give Routine → {routj, known routine}.

### 3.7.7.4 Multiplier equations and startvalues

The Lagrange routine relies on the following subroutines. They normally set the Options[Lagrange] to the latest values, so that these options give the state of the system.

MultiplierEquations[opts\_\_\_Rule]                                      an input format

MultiplierEquations[f, g, x, mu\_Symbol: Mu]                      an input format

MultiplierEquations → {D[f + mg, x] == 0}

in output. Used for the start values m0 when Routine → {rout, FindRoot}

MultiplierStartValues[f, g, x, x0]

subroutine of Lagrange, finds the starting value m0 of the  
Lagrange multipliers using least squares on MultiplierEquations

### 3.7.7.5 Option settings

Defaults are taken from Options[Lagrange], and output is put there too. Options are cleared by ResetOptions[Lagrange].

Additional inputs (with defaults) are: MultiplierSymbol → Mu, RealRootQ → False (include Complex results of Solve), Routine → FindRoot (see ?Routine), Condition → (Min, Max or First (only first order

conditions: default)). Condition  $\rightarrow \{\text{First}, \text{False}\}$  selects the first order condition and suppresses the message on it. Input options relevant for the solution Routine are passed on to it. Option ResetOptions  $\rightarrow \text{True}$  (default False) resets all options first; this may be handy if you switch from one problem to another and do not want to get mixed up.

Additional outputs are: NumberOfVariables  $\rightarrow \dots$ , NumberOfConstraints  $\rightarrow \dots$ , Multipliers  $\rightarrow \{\dots\}$ , MultiplierEquations  $\rightarrow \{D[f + mg, x] == 0\}$ , MultiplierStartValues  $\rightarrow \{m0, \text{found by least squares on } x0\}$ , Out  $\rightarrow \{\text{format given above}\}$ .

### 3.7.7.6 Ladders

Below reproduces the example given by C. Henry Edwards, cited above.

Minimize  $(L1 + L2)$  subject to  $G[\dots] = 0$ .

A moat of width  $w$  is bridged by ladder  $L1$  from the left and  $L2$  from the right. They meet at a height  $y$ . Ladder  $L1$  is supported at the moat side by a wall with height  $h1$ . With the sun straight up, the shadow of  $L1$  falls on the moat's water for length  $u$  and on the ground (to the left of  $h1$ ) for length  $p$ . Thus the slope of  $L1$  is  $a1 = h1 / p = y / (p+u)$ . The shadow of  $L2$  falls on water for length  $v = w - u$  and on the ground (to the right of  $h2$ ) for length  $q$ . Thus the slope of  $L2$  is  $a2 = y / (v + q) = h2 / q$ . Find the minimum possible value of  $L1 + L2$ .

```
ResetOptions[Lagrange];

aim = L1 + L2; (X = {p, q, y, u, v, L1, L2};)
G = {u + v - w, (*division of the moat width*)
      (p + u)^2 + y^2 - L1^2, (q + v)^2 + y^2 - L2^2, (*Pythagoras on shadows*)
      (y - h1) p - h1 u, (y - h2) q - h2 v}; (*equality of slopes*)
w = 50; h1 = 10; h2 = 15;
x0 = {15, 15, 25, 25, 25, 60, 60};

SetOptions[Lagrange, Expression  $\rightarrow$  aim,
  Variables  $\rightarrow$  X,
  Constraints  $\rightarrow$  G,
  StartValues  $\rightarrow$  x0]

{Condition  $\rightarrow$  First, Constraints  $\rightarrow$ 
  {u + v - 50, -L1^2 + (p + u)^2 + y^2, -L2^2 + (q + v)^2 + y^2, p (y - 10) - 10 u, q (y - 15) - 15 v},
  Expression  $\rightarrow$  L1 + L2, MultiplierEquations  $\rightarrow$  {}, Multipliers  $\rightarrow$  {}, MultiplierStartValues  $\rightarrow$  {},
  MultiplierSymbol  $\rightarrow$  Mu, NumberOfConstraints  $\rightarrow$  0, NumberOfVariables  $\rightarrow$  0,
  Out  $\rightarrow$  {}, RealRootQ  $\rightarrow$  False, ResetOptions  $\rightarrow$  False, Routine  $\rightarrow$  FindRoot,
  StartValues  $\rightarrow$  {15, 15, 25, 25, 25, 60, 60}, Variables  $\rightarrow$  {p, q, y, u, v, L1, L2}}
```

**Lagrange[Routine → FindRoot, Condition → Min]**

*Lagrange::sec : Second order conditions not yet implemented*

```
{102.715, {p → 17.2361, q → 13.118, y → 30.9275, u → 36.0708,
  v → 13.9292, L1 → 61.629, L2 → 41.086}, {Mu(1) → 1.27828, Mu(2) → -0.00811306,
  Mu(3) → -0.0121696, Mu(4) → 0.0413315, Mu(5) → 0.0413315}}
```

## Note: Comparison

Jean-Christophe Culioli in "Optimisation with *Mathematica*", in Varian ed. (1996) also gives a solution method using multipliers. His work seems robust, but for some reason his method cannot find above solution. This is discussed in the example notebook "Lagrange.nb".

### 3.7.8 Constrained optimisation with inequality constraints

The routine has a fixed input format or a free rule based input format. The following box gives both formats.

`KuhnTucker[opts]`      with for example Expression → f, Variables → x,  
                                  Constraints[Equal] → g, Constraints[GreaterEqual] → h,  
                                  Routine → ... (, StartValues → x0)

`KuhnTucker[f, g, h, x, x0_List: {}, mu_Symbol: Mu, kappa_Symbol: Kappa]`

Output contains Lagrange multipliers and has the form:  
   for FindRoot: {f[x\*], {x → x\*}, {m → m\*}, {k → k\*}},  
   for Solve: {f[yi\*], {yi → yi\*}} for i solutions,  
   where y can be x, m and k intermixed.

See the discussion of Lagrange for the modus operandi. An example probably works best.

Let us regard the determination of the Markowitz efficiency frontier in a CAPM model with three assets. The frontier is given by minimum variance given an expected rate of return. The weights of the assets in the efficient portfolios are x, y and z, and let us suppose that the returns of the assets are 1, 2, and 3, while the assets have a unit diagonal covariance matrix.. This example is taken from Luenberger (1998:161).

- The equality constraints are that the weights must add to 1, and that the expected value must be  $r$ .

```
ws = {x, y, z};
rs = {1, 2, 3};
G = {Add[ws] - 1, ws . rs - r};
cov = DiagonalMatrix[{1, 1, 1}];
```

- We minimise  $ws.cov.ws$ . In this case the inequality constraints are just the weights themselves:  $\{x, y, z\} \geq 0$ , since this is sufficient to keep any weight in the  $[0, 1]$  domain.

```
ktsol = KuhnTucker[Expression → ws.cov.ws,  
  Constraints[Equal] → G,  
  Constraints[GreaterEqual] → ws,  
  Variables → ws,  
  Routine → Solve, Condition → Min] // Simplify;  
  
KuhnTucker::sec : Second order conditions not yet implemented
```

- We manipulate output a bit for TableForm printing.

```
ktsol2 = (Prepend[Last[#1], Solution[] → First[#1]] & ) /@ ktsol;
```

- The first order conditions have 4 solution points. Only those with nonnegative  $\kappa$  are real minima.

```
SolveShow[ktsol2]
```

	1	2	3	4
$x$	0	$\frac{4}{3} - \frac{r}{2}$	$2 - r$	$\frac{3-r}{2}$
$y$	$3 - r$	$\frac{1}{3}$	$r - 1$	0
$z$	$r - 2$	$\frac{1}{6} (3r - 4)$	0	$\frac{r-1}{2}$
Kappa(1)	$6r - 16$	0	0	0
Kappa(2)	0	0	0	-1
Kappa(3)	0	0	$8 - 6r$	0
Mu(1)	$26 - 10r$	$\frac{14}{3} - 2r$	$10 - 6r$	$5 - 2r$
Mu(2)	$4r - 10$	$r - 2$	$4r - 6$	$r - 2$
Solution()	$2r^2 - 10r + 13$	$\frac{r^2}{2} - 2r + \frac{7}{3}$	$2r^2 - 6r + 5$	$\frac{1}{2} (r^2 - 4r + 5)$

The following are subroutines that speak for themselves - or see the discussion in the related section of Lagrange. It may be noted that the algebraic solution method (Solve) finds all solutions at the same time, while the numerical method (FindRoot) finds only one solution depending upon the startvalues.

```

KTEquations[opts___Rule]                                or

KTEquations[f, g, h, x,                                update Options[KuhnTucker]
(x0,) mu_Symbol: Mu, kappa_Symbol: Kappa]

KTStartValues[f, g, x, x0, Equal | GreaterEqual]

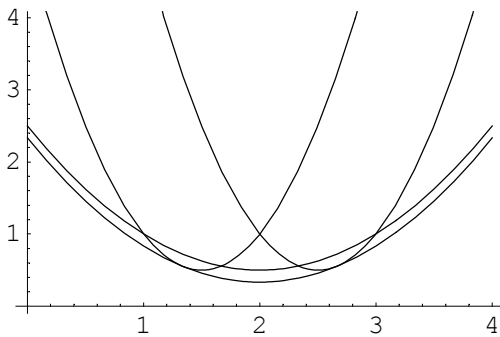
subroutine of KuhnTucker,
finds the starting value m0 of the KuhnTucker
multipliers using least squares on KTEquations.

```

These routines work in the same way as their counterparts for Lagrange, with the added distinction between the equality and inequality multipliers. Though the formats are the same, the routines are fully independent of each other.

- Let us analyse the solutions found above. The following is a plot for certain rate of interest  $r$  in  $[0, 4]$ .

```
Plot @@ {First /@ ktsol, {r, 0, 4}};
```



- To analyse these further, we can use the routine `PrDomain` from the `Probability` package. If we apply `PrDomain` to the weights, then this routine also appends the sum constraint, and we can see that it is always satisfied. (Note that the data are not transposed, while above table is.)

```
Economics[Probability, Print → False]
```

```
ktsol3 = Last /@ ktsol;
```

```
{x, y, z} /. ktsol3;
```

```
ineql = PrDomain /@ % // Simplify
```

$$\left( \begin{array}{cccc} \text{True} & 0 \leq 3 - r \leq 1 & 0 \leq r - 2 \leq 1 & \text{True} \\ 0 \leq \frac{4}{3} - \frac{r}{2} \leq 1 & \text{True} & 0 \leq \frac{1}{6} (3r - 4) \leq 1 & \text{True} \\ 0 \leq 2 - r \leq 1 & 0 \leq r - 1 \leq 1 & \text{True} & \text{True} \\ 0 \leq \frac{3-r}{2} \leq 1 & \text{True} & 0 \leq \frac{r-1}{2} \leq 1 & \text{True} \end{array} \right)$$

- We also need nonnegative kappa's. The last solution does not have a nonnegative multiplier, so drops out. You might verify this by dropping it from the plot indeed.

```
ineq2 = Map[# ≥ 0 &, Array[Kappa, 3] /. ktsol3, {2}]
```

$$\begin{pmatrix} 6r - 16 \geq 0 & \text{True} & \text{True} \\ \text{True} & \text{True} & \text{True} \\ \text{True} & \text{True} & 8 - 6r \geq 0 \\ \text{True} & \text{False} & \text{True} \end{pmatrix}$$

- *Mathematica*'s routine `InequalitySolve` has been loaded by `Probability``. It finds the following solution domains for the  $r$ , for which the separate solutions apply.

```
MapThread[InequalitySolve[And @@ #1 && And @@ #2, r] &, {ineq1, ineq2}]
```

$$\left\{ \frac{8}{3} \leq r \leq 3, \frac{4}{3} \leq r \leq \frac{8}{3}, 1 \leq r \leq \frac{4}{3}, \text{False} \right\}$$

It turns out that `InequalitySolve` is not strong enough to find all binding constraints for complex problems. The point seems to be that the maximand is needed to determine whether a constraint is relevant or not. This kind of problem is tackled in appendix A.11 (and `WhichIFPair` and `IFPairSimplify` in particular). An application is section 5.14 on the CAPM (and `CAPMSolve` in particular).



## 3.8 Databank

---

### 3.8.1 Summary

This implements a databank structure for object oriented programming. There is also a facility to aggregate data in a way that occurs often in economics and statistics.

**Economics [Databank]**

### 3.8.2 Introduction

The Databank package has an object oriented approach to data. A DataMold is a list that shows how a data object is structured, like e.g. `{{Price, Quantity}, Value}`. Each data record in the Databank would have this structure, such as `{{.7, 10}, 7}`. Once the DataMold for a specific Databank has been defined, routines can exploit the knowledge about this structure. For example, there can be a rule that Value is the product of Price and Quantity.

An aggregation structure that is common in economics and statistics is:

1. each record of basic observations can be extended with derived criteria depending upon the basic observations only (e.g. price and quantity create value)
2. then all records are summed, giving the aggregate record
3. the aggregate record is corrected with the aggregate criteria (aggregate price follows from aggregate value divided by aggregate quantity).

This is a rather simple structure that seems to beg for a spreadsheet. Implementing this structure for *Mathematica* however opens the gate for more flexible applications.

Note that the Databank package concentrates on one-record-relationships and on aggregation. It neglects at least two aspects with data series: record dependency and data handling and storage. For the latter, see the `Data`` package. For more complex chain indices of prices and quantities, see the `Chainindex`` package.

An application that uses the `Databank`` package has the advantage that it can use the Databank functions. Applications are for example the `Money`` and `Freight`` packages. The general Databank procedures like `Explain` and `ShowData` thus are also available to those packages.

### 3.8.3 Example

The `SetDatabank` routine only helps structuring the process. You can do all the steps by yourself or you can always redefine options and the like. In the following we use the variable *db* to stand for an arbitrary databank.

```
SetDatabank[x_List, data_List: {}, db_Symbol: Automatic]
```

(1) sets db[Data] = data,

(2) creates or sets Options[db] = {DataMold → x, ...}

(3) does SetOptions[SetDatabank, Databank → db]

Note: If the db symbol is Automatic, then the Databank option in Options[Databank] is taken.  
Note: Currently only DataMolds are accepted that are 1 level deep.

The Options[db] should contain the following options, and are set by SetDatabank:

AfterSum	the operations to be performed after summing the entries.
BeforeSum	the operations to be performed before summing the entries
DataMold	identifies the keys of your databank

Your data will be available in db[Data]. Though it is not required, it is a hint to define your data mold as db[DataMold]. This gives you direct access to that data mold. Let us do this for some transactions, for which we use symbolic data from the alphabet. We allow for incomplete records too.

```
Transactions[DataMold] = {{Price, Quantity}, Value};

SetDatabank[Transactions[DataMold],
  {{{a, b}, c}, {{d, e}}}, Transactions]

{Databank → Transactions}

Transactions[Data]

{{{a, b}, c}, (d e)}

Options[Transactions]

{DataMold → {{Price, Quantity}, Value}, BeforeSum → {}, AfterSum → {}, Upd → False}
```

Available are a large number of operations, using the head Databank[ ].

Databank[x...]	operation x on the databank (see below)
Explain[expr(,db)]	a Flattened explanation keyi → valuei

- The Explain call is useful for readers, but also for ReplaceAll.

```
Explain[ {{a, b}, c} ]

{Price → a, Quantity → b, Value → c}
```

Upd[ ] completes incomplete elements by using the DataMold, and can apply BeforeSum operations.

<code>Upd[x__]</code>	sets <code>db[Data] = Databank[x__]</code> where the <code>db</code> is the last element in <code>x</code> , or taken as the current database
<code>Upd[]</code>	performs <code>Upd[FromMold]</code> and then <code>Upd[ReplaceAll, BeforeSum[]]</code>

**Upd[FromMold]**

$$\left( \begin{array}{cc} \{a, b\} & c \\ \{d, e\} & \text{Indeterminate} \end{array} \right)$$

- Defining before and after sum operations:

```
SetOptions[Transactions,
  BeforeSum → {Value → Price Quantity},
  AfterSum → {Price → Value / Quantity}];
```

Note that `BeforeSum[db:Automatic]` and `AfterSum[db:Automatic]` give quick access to those rules of the `db` databank.

- The aggregate result is:

**Databank[Sum]**

$$\left\{ \left\{ \frac{ab + de}{b + e}, b + e \right\}, ab + de \right\}$$

**Explain[%]**

$$\left\{ \text{Price} \rightarrow \frac{ab + de}{b + e}, \text{Quantity} \rightarrow b + e, \text{Value} \rightarrow ab + de \right\}$$

### 3.8.4 Databank procedures

We distinguish between operations on a single record and operations on the whole databank. The latter are:

Databank[Query, <i>key</i> (, <i>db</i> )]	for retrieving values
Databank[ReplaceAll, { <i>new__Rule</i> } (, <i>db</i> )]	
for replacing the values with new ones. Per data entry you can use existing values in the databank by referring to the keys only	
Databank[ReplaceAll, <i>head</i> (, <i>db</i> )]	for the head of a function that has a list of rules as input (as Explain[entry]) and that creates a new list of rules
Databank[Plus (, <i>db</i> )]	adds all entries (i.e. Plus @@ db[Data])
Databank[Sum (, <i>db</i> )]	first does BeforeSum, then adds, and then does AfterSum
Databank[Add, <i>data_List</i> (, <i>db</i> )]	is similar to Databank[Sum, (,db)] but then for a datalist (not db[Data])
Databank[FromMold (, <i>db</i> )]	adjusts all entries to the DataMold mold
Databank[Switch, <i>db</i> (, <i>data_List</i> )]	switches to db (while the db[Data] may be updated to data)

Note: If Databank is the databank (default) then the data are in Databank[Data].

Operations for *single* records are defined for Query, Replace and FromMold, commonly with the record mentioned as the second argument in the call. See ?Databank for details.

3.8.5 Show the data

Operations based on TableForm are:

ShowData[( <i>db</i> ,) <i>opts</i> ]	plots the data in columns
ShowData[Sum, ( <i>db</i> ,) <i>opts</i> ]	adds the AfterSum result

ShowData [Sum]			
	1	2	3
Price	<i>a</i>	<i>d</i>	$\frac{a\,b+d\,e}{b+e}$
Quantity	<i>b</i>	<i>e</i>	<i>b + e</i>
Value	<i>c</i>	Indeterminate	<i>a b + d e</i>

Complexer formats are:

ShowData[x][(db,) opts]	for a Take[data, x] selection
ShowData[Transpose, (db,) opts]	transposes
ShowData[x][Transpose, (db,) opts]	transposes & selects entries
ShowData[Transpose, data_List, (d b,) opts]	for arbitrary data
ShowData[x][Transpose, data_List, (db,) opts]	selects entries from data

3.8.6 Updating

Note that we used Upd[FromMold] above, and not the full Upd[ ] call. Using the latter redefines the single entries - and in this case replaces c and Indeterminate.

Upd[ ]

$$\left( \begin{array}{cc} \{a, b\} & a\,b \\ \{d, e\} & d\,e \end{array} \right)$$

ShowData[Sum]

	1	2	3
Price	$a$	$d$	$\frac{a\,b+d\,e}{b+e}$
Quantity	$b$	$e$	$b+e$
Value	$a\,b$	$d\,e$	$a\,b+d\,e$

Adding to the databank can be done directly.

Transactions[Data] = Append[Transactions[Data], {p, q}]

$$\left( \begin{array}{cc} \{a, b\} & a\,b \\ \{d, e\} & d\,e \\ p & q \end{array} \right)$$

# 4. Logic and Inference

## 4.1 Introduction

---

### 4.1.1 Decision support environment

*Mathematica* already is a decision engine of a kind. If you run the linear programming routine then there is a lot of deduction before the answer pops up. However, that answer does not come as a neat English expression and does not read as a conclusion in the way that a good speaker would summarise his or her speech. The idea here is, then, to extend *Mathematica*'s powers as a decision machine, and to equip the system with linguistic deduction. Note also that we properly regard *Mathematica* as a language itself too, and hence we are in the subject field of better integrating language and thought.

Suppose that you have a block of text, for example a text that summarises your research results. Would it not be nice to submit it to your computer, and have it verify whether it is logically sound or errs against logic? The `Logic`` and `Inference`` packages are a step into that direction. Admittedly, there still is a long way to go, but if we don't make the first steps we will not get far at all. From the viewpoint of our ambitious goal, we are discussing more than just an enhancement of the *Mathematica* system, and for this reason this discussion has been made a separate chapter.

For an economist it is always nice to remember that John Neville Keynes, the father of John Maynard Keynes, taught logic and wrote a book on the subject. J.M. Keynes was inclined to logic and reasoning, at least, as he himself opposed his way of thinking to that of Jan Tinbergen and Ragnar Frisch who developed econometrics and the methods of number crunching. It seems more suitable to first discuss logic and inference before we start on economics proper.

### 4.1.2 Propositional logic versus predicate logic

There is a distinction between propositional logic and predicate logic. The `Logic`` and `Inference`` packages only deal with propositional logic.

#### 4.1.2.1 Propositional logic

Propositional logic regards normal English sentences as compounds of *atomic* propositions. These atomic propositions can be denoted as  $P[1]$ ,  $P[2]$ ,  $P[3]$ , .... The links in these compounds are provided for by logical operators And, Or, Not and If.... Then..... Regard the following deduction:

**English****Analysis in propositional logic:**

- |   |                   |
|---|-------------------|
| (a) If it rains then the streets are wet. | If P[1] Then P[2] |
| (b) It rains.                             | P[1]              |
| (c) The streets are wet.                  | P[2]              |

Note that statement (a) is a proposition while the steps from a to c are a deduction. The "If ... Then ..." of (a) may be seen as belonging to Statics, while (a) till (c) are Dynamics.

Propositional logic has two advantages for inference. First, it allows expression of any problem in the canonical form, that is, using only the operators Not, And and Or. For example,  $p \Rightarrow q$  can be analysed as  $\text{Or}[\text{Not}[p], q]$ . Secondly, the verification by truth tables is straightforward. These advantages provide an important anchor.

**4.1.2.2 Predicate logic**

Predicate logic looks below the atomic level, and analyses the structure within propositions. The classic example is "All men are mortal", "Socrates is a man", ergo "Socrates is mortal". *Mathematica* already provides a good working environment for finite set operations or for formal testing of predicates. Yet, we will not develop this aspect. The reason is that the development of propositional logic already took a lot of time.

**4.1.3 Steps and drawbacks**

The development of these packages showed the need for some simplifying steps. These then are drawbacks too.

- The first step is to translate normal English into a simple format that the pattern recognition can deal with. This simple English is called "Englogish" here. For example: "I don't like you if you smell" becomes "If you smell then not I like you."

This also means that you will get into problems with subatomic meanings of predicate logic. For example: "I like you, but you smell" becomes something more complicated in order to deal with the "but": "If I like you then it is suggested that I like everything of you. You smell. If you smell then not I like you."

- A second point is that you have to watch the transformation order. For example: "If you jump and not you jump then it rains" is "If (a & !a) Then (b)", while "If not you jump and you jump then it rains" is "If (!(a&a)) Then (b)".
- The system is sensitive to input typing errors.
- The system relies on built-in *Mathematica* features LogicalExpand, Reduce, Collect and the like, and combines some of the weaknesses therein. However, the packages give an enhancement of And and Or (&& and ||).

#### 4.1.3.1 Languages

Note that the `Logic`` package can deal with any language if you use the logical symbols:

- within text:    and, or, not, if, then.    Example: "Je t' aime and il pleut."
- at input level:    &&, ||, !.    Example: "Je t' aime" && "il pleut"

#### 4.1.3.2 Counterfactuals

Much reasoning relies on counterfactuals. The best way to deal with these in the environment of the `Logic`` package is to qualify what one supposes. The following is a valid argument, a *reductio ad absurdum* i.e. proof by contradiction:

**"Suppose that it rains. If suppose that it rains and our suppositions  
fit reality then it rains. If it rains then it is wet. Not it is wet.";**

#### 4.1.4 About Inference

It must be remembered that decision situations can be very complex. For example, you may have accepted  $\{a, b, c\}$  and know that  $\{a, c\}$  is sufficient to conclude to  $\{d\}$ . Your scheme of deduction then must take all subsets of  $\{a, b, c\}$  into account.

One problem is, that it are humans who decide what they regard as conclusions. If you have  $\{p, p \Rightarrow q\}$  as your data, then not only  $\{q\}$  is a conclusion, but also  $\{p \Rightarrow (p \Rightarrow q), \dots\}$ .

Complexity increases by the possible choice between different methods. A common distinction is between the axiomatic method and the truth-table method. In other words, rule based inference and pure computation. The optimum likely uses some aspects of both, since you stop computation of 16000 variables when you already know that  $p$  and  $\neg p$  occur.

But the difference in methods is not just that. The truth-table method is strong for cases where one has a definite idea about the relation to be proven. The axiomatic method has the advantage that forms may be created that one had not thought of before.

*Mathematica* may be seen as a rule based program oriented at pattern recognition, and one would expect the axiomatic method to be superior and easy to implement. The `Inference`` package shows that matters are still complex.



## 4.2 Propositional logic

### 4.2.1 Summary

The main result is `Decide[text]` that can extract new conclusions ("the news"). Secondly, the standard *Mathematica* functions for And & Or can be enhanced. Thirdly, there are various results on text handling and logical operations.

### 4.2.2 Loading the package

```
Economics[Logic]

?RemarkOnParentheses
```

Mathematica does generally not print parentheses around And, Or, and Implies. For example, in Mathematica 3.0 the precedence order printout of Implies in Traditional Form actually is erroneous:  $(p \Rightarrow (q \Rightarrow r))$  is printed the same as  $((p \Rightarrow q) \Rightarrow r)$ , both without parentheses. The `Logic`` package inserts parentheses if you print in `TraditionalForm`

Note: If you evaluate `!p` with `"!"` at the beginning of a line, then *Mathematica* starts reading file `p`.

To start with, the package defines the following small routines.

<code>TertiumNonDatur[x]</code>	gives $(x \parallel \text{Not}[x])$
<code>Equivalent[p, q]</code>	means $(p \Rightarrow q) \&\& (q \Rightarrow p)$
<code>Imp</code>	replaces, Rule $\rightarrow$ Implies. Example: $p \rightarrow (p \rightarrow q)$ /. <code>Imp</code> The use of <code>Imp</code> allows the use of Rule ( $\rightarrow$ ) for logical input readability
<code>Negate[x]</code>	performs <code>LogicalExpand[!x]</code>
<code>NotNot[x]</code>	performs <code>!Negate[x]</code> or <code>!LogicalExpand[!x]</code>
<code>NotpOrq[p, q]</code>	gives $\text{Not}[p] \parallel q$ This function is used to replace <code>If[p, q]</code> and <code>Implies[p, q]</code>
<code>LogicalVariables[x]</code>	provides the list of variables for a logical statement <code>x</code> .
<code>ToDNForm[expr]</code>	changes <code>expr</code> into the disjunctive normal form by substituting <code>If</code> , <code>Implies</code> and <code>Equivalent</code> by <code>And</code> , <code>Or</code> & <code>Not</code>
<code>Implications[x]</code>	gives a list of implications of <code>x</code> . If joined by <code>And</code> , then all implications are equivalent to the original

4.2.3 Enhancement of And and Or

*Mathematica's* And & Or are rather weak. For example, evaluate A && Not[A], and see that it simply remains A && Not[A]. The reason is that these operators are not meant to be entirely 'deductive'. For example, in equations  $x = 0$  and  $x \neq 0$  you want to find a solution set {} and not False. Similarly, If is a control function while Implies belongs to the logical object language. For applications of logic, a favourable aspect of this weak definition is that this allows for a multi-valued logic. However, the standard is two-valued logic, and then the weakness is not useful.

AndOrEnhance[x]	with x = True or False; default True enhances && and
AndOrEnhanced	gives the state of the system
AndOrRules	rules that enhance And & Or

The AndOrRules can be used for replacement in standard *Mathematica*. In AndOrEnhanced mode, they are added to the definitions of And & Or, and then are no longer available for Replacement.

- Examples are:

```
p && Not[p]
```

$(p \wedge ! p)$

```
AndOrEnhance[]
```

*AndOrEnhance::State : Enhanced use of And & Or is set to be True*

```
p && Not[p]
```

False

```
p && q && Not[p] && q
```

False

```
AndOrEnhance[False]
```

*AndOrEnhance::State : Enhanced use of And & Or is set to be False*

- In normal mode, you still can use the AndOrRules.

```
(!q || (p && q)) //. AndOrRules
```

$(! q \vee p)$

- The rules can be looked at by:

```
Begin["Cool`Logic`Private`"]
MatrixForm[AndOrRules]
End[]
```

#### 4.2.4 Decide

The high level routine is Decide.

<code>Decide[x, opts]</code>	performs <code>Conclude[Deduce[Conclude[x], opts]]</code> . The procedure automatically turns on <code>AndOrEnhance</code>
------------------------------	--

- The extraction of a decision may be possible by Decide.

```
Decide["If it rains then the streets are wet. It rains."]
```

```
AndOrEnhance::State : Enhanced use of And & Or is set to be True
```

```
{ the streets are wet}
```

```
Decide["If the apple falls then it hits my head. The apple falls. If it  
hits my head and I am Newton then I invent gravity. If I invent gravity  
then I am Newton. Not I am Newton."]
```

```
{! i invent gravity, it hits my head}
```

- If we regard the counterfactual of section 4.1.3.2 again:

```
"Suppose that it rains. If suppose that it rains and our suppositions  
fit reality then it rains. If it rains then it is wet. Not it is wet."  
// Decide
```

```
{! it rains, ! our suppositions fit reality}
```

```
AndOrEnhance[False]
```

```
AndOrEnhance::State : Enhanced use of And & Or is set to be False
```

#### 4.2.5 Englogish: From statement to proper sentence to proposition

Before we can fully discuss Decide, Conclude and Deduce, we have to understand what "Englogish" is, and how Englogish propositions are created.

First of all, Englogish is the simple English-like language that our text routines can deal with. Only the propositional constants are recognised, and Not has to occur at the beginning of a proposition. (See section 4.1.3.) Englogish statements still can have upper case letters and punctuation. These are transformed into 'proper sentences' (without first capital and final point), in order to allow our routines to recognise the same proposition in different places in a paragraph.

Sentences [statement\_String]  
  
transforms a statement into a List of Proper Sentences. A  
statement is a String that contains English sentences that start with  
a (Blank &) Capital Letter and that end with a Point (& Blank)

```
sents = Sentences["If it rains then the streets are wet. It rains."]
{ if it rains then the streets are wet, it rains}
```

Then, a proper sentence is subjected to Analysis.

Analyse [sentence]                    transform sentence (Standard Sentence Format)  
   into a *Mathematica* Format  
  
Analyse [If || And || Or, sentence]  
  
   subroutines, deal with specifics of If, And or Or sentences  
  
AnalysePrint[x (, props)]    prints the original statement x,  
   may set the propositional analysis in  
   props and gives the PropositionMeaningRule

Subroutines not shown here are ProperSentence, JoinStatements and LocateThen. An example analysis is:

```
Analyse /@ sents
{If[ it rains, the streets are wet], it rains}
```

The routines above are combined in the following routine, that also collects the propositions.

ToPropositionalLogic[statement_String]	
transforms an Englogish statement into a List of <i>Mathematica</i> Logical Expressions. The result can be submitted to Deduce[...]. The routine also generates Propositions = {P[1], ... } and PropositionMeaningRule	
P	P[i] is an atomic sentence, the ith element in Propositions.
Propositions	is the list of atomic sentences P[i]. Enter Propositions /. PropositionMeaningRule to substitute the various meanings
PropositionMeaningRule	This is a rule that gives the meaning of the current atomic sentences.

```
■ For example:
ToPropositionalLogic["If it rains then the streets are wet. It rains."]
{If[P(1), P(2)], P(1)}

Propositions
{P(1), P(2)}

PropositionMeaningRule
{P(1) → it rains, P(2) → the streets are wet}
```

4.2.6 Conclude

Conclusions are those statements that are accepted. Accepted statements are normally linked by && signs at the highest level. It appears useful however to have the Conclusions as a *Mathematica* list, i.e. without those '&&'. The Conclude procedure provides such a list, and puts out new findings ("the news").

Conclude[x]	gives the new conclusions in x, i.e. those that are not already in Conclusions
Conclude[]	is equal to Conclusions = {}, and you will want to do this at a new round of deductions
Conclusions	contains the list of conclusions currently accepted

```
■ An example of a deductive chain is:
Conclude[]; (* sets Conclusions = {}*)
```

```
example = p && q && (g || (d && f && p)) && h && q
```

```
(p ∧ q ∧ (g ∨ (d ∧ f ∧ p)) ∧ h ∧ q)
```

```
Conclude[example]
```

```
{h, p, q, (g ∨ (d ∧ f ∧ p))}
```

- The procedure Conclude gives only the news. Hence, if we try Conclude again on the same proposition, then we don't get any news.

```
Conclude[example]
```

```
{}
```

- However, by some logical manipulations we sometimes can find a hidden conclusion.

```
LogicalExpand[example] //. AndOrRules
```

```
(h ∧ p ∧ q ∧ (g ∨ (d ∧ f)))
```

```
Conclude[%] (* find the news *)
```

```
{(g ∨ (d ∧ f))}
```

```
Conclusions (* has collected all our findings *)
```

```
{h, p, q, (g ∨ (d ∧ f)), (g ∨ (d ∧ f ∧ p))}
```

#### 4.2.7 Deduce

Deduce first transforms a simple English text into a List of *Mathematica* Expressions in propositional logic. It generates the current values of Propositions and PropositionMeaningRule. Default Option is PropVariableList → Propositions, so that Deduce recognises only P[i] statements. If the original statements are not in text form, then the user must supply a variable list. As a second step, the method of deduction is applied. Options[Deduce, Deduce → LogicalExpand] is default, but another setting is Implications.

`Deduce[statement_String, opts___Rule]`

transforms a simple Englogish text into a List of *Mathematica* Expressions in propositional logic, and then applies the deduction routine

`PropVariableList` is an option for `Deduce`;  
default is `PropVariableList → Propositions`

■ Check on enhancement

`AndOrEnhanced`

False

■ Propositional analysis of a clear contradiction

`Deduce["It rains. Not it rains."]`

False

■ Propositional analysis of a more complex contradiction

`Deduce["If it rains then the streets are wet. Not the streets are wet. It rains."]`

False

■ Testing tautologies (Ex Falso Sequitur Quodlibet)

`Deduce["If it rains and not it rains then you get all my money."]`

True

## 4.2.8 Algebraic structure

The following exploits the structural equivalence of {And, Or} with {Times, Plus}. Standard routines like `Collect` and `Expand` that know how to handle `Times` and `Plus` can be used here.

`ToLogic[f, input, parms]`

does `f[input /. {And → Times, Or → Plus}, parms] /. {Times → And, Plus → Or}`

`ToLogic[Collect, p && q || !p && q, q]`

$(q \wedge (p \vee !p))$

The following are more ambitious.

<code>ToAlgebra [x]</code>	turns x into equation by replacing And $\rightarrow$ Times, Or[p,q] $\rightarrow 1 - (1-p) (1-q)$ , Not[p] $\rightarrow 1 - p$ . If the option All $\rightarrow$ True is set, then equalities $p == 1 \parallel p == 0$ are included, so that the output can be offered to Reduce
<code>ToEquationRule [var_List]</code>	gives the rules for transforming propositions into equations
<code>FromEquationRule [var_List]</code>	gives the rules for transforming equations back into propositional logic
<code>ATOP</code>	just the list {And $\rightarrow$ Times, Or $\rightarrow$ Plus}

4.2.9 Truth tables

Truth tables are an invention of Ludwig Wittgenstein. They enumerate all possible True | False combinations, and then the logical constants are defined in terms of operations.

<code>TruthValue [x]</code>	gives the truth share from 0 up to 1. A tautology means 1
<code>TruthTable [x]</code>	gives the combinations of True and False for the variables in x
<code>TruthTableRule [x:Propositions]</code>	If x is a list of (logical) variables then a Rule is created. If x is a statement then the statement is evaluated with that Rule

Options[TruthTable] also apply to Decide, TruthValue, TruthTableRule and LogicalVariables. Options can only be given by SetOptions[TruthTable, ...]. Default option is Rule  $\rightarrow$  Implies, meaning that ' $\rightarrow$ ' in input is read as 'implies'. Other values disable this option.

**AndOrEnhance[False]**

*AndOrEnhance::State : Enhanced use of And & Or is set to be False*

**TruthTable[p && q] // MatrixForm**

$$\left( \begin{array}{l} \{p, q\} \rightarrow \text{True} \\ \{p, !q\} \rightarrow \text{False} \\ \{!p, q\} \rightarrow \text{False} \\ \{!p, !q\} \rightarrow \text{False} \end{array} \right)$$



```
TruthValue[p && q]
```

$$\frac{1}{4}$$

```
TruthValue[p || !p]
```

```
1
```

TruthTableRule[x] can be applied to x = statement or to x = List.

```
try = p && (q || r);
```

```
try = Equivalent[try, LogicalExpand[try]]
```

$$((p \wedge (q \vee r)) \Leftrightarrow ((p \wedge q) \vee (p \wedge r)))$$

```
% // LogicalVariables
```

```
{p, q, r}
```

```
TruthTableRule[%]
```

```
{{{p -> True, q -> True, r -> True}, {p -> True, q -> True, r -> False}},
 {{p -> True, q -> False, r -> True}, {p -> True, q -> False, r -> False}},
 {{p -> False, q -> True, r -> True}, {p -> False, q -> True, r -> False}},
 {{p -> False, q -> False, r -> True}, {p -> False, q -> False, r -> False}}}
```

```
TruthTableRule[try]
```

```
( {True, True} {True, True} )
( {True, True} {True, True} )
```

```
TruthValue[try]
```

```
1
```

## 4.3 Logical analysis of longer texts

---

### 4.3.1 Summary

In analysing longer texts, it appears to be useful to have some text management facilities. In particular, paragraphs can be analysed by themselves, and intermediate logic steps can be eliminated since they need not be relevant for the final conclusion. An example of a longer analysis is given in `LogicExample.nb`, that discusses the Financial Times editorial of Friday July 26 1991.

### 4.3.2 Loading the package

```
Economics[Logic]
```

### 4.3.3 Paragraph

<code>Paragraph[Label]</code>	contains the list of labels
<code>Paragraph[Set]</code>	sets the list of labels to {}
<code>Paragraph[label, "..."]</code>	associates the label with the input text and updates the list of labels
<code>Paragraph[In, label]</code>	contains the associated text. (Label = All gives the joined text, when first Set)
<code>Paragraph[Set, In, All]</code>	joins the input texts associated with the list of labels
<code>Paragraph[Set, Out, head, label(, opts)]</code>	
	applies head with opts to Paragraph[In, label] (e.g. head = Deduce)
<code>Paragraph[Set, Deduce, label(, opts)]</code>	
	does the former for Deduce and then applies the PropositionMeaningRule

```
Paragraph[Out, head, label]
```

the result of Paragraph[Set, Out, head, label (, opts)]

```
Paragraph[Deduce, All, done:True (, opts)]
```

could follow after a number of Paragraph[label, ... ] statements.

If done == False, it calls Deduce on the paragraphs (with the opts).

It fills Paragraph[Out, Deduce, All] as a conjunction of Deduce on the separate paragraphs (without calling Deduce for the whole).

#### 4.3.4 Utilities

There are some utilities that help to remove statements with known values.

```
Choose[x_List, val:True]
```

performs Map[ (# → val)& , x]

```
ChooseP[x_List, val:True]
```

performs Map[ (P[#] → val)& , x]

```
ChoosePrint[x_List, rule, val:True]
```

performs

(Print[Map[P, x] /. rule ]; Map[ (P[#] → val)& , x] )

#### 4.3.5 Example

The following is just a simple example.

```
Paragraph[Set];
```

- A first paragraph:

```
Paragraph[1, "If it rains then the streets are wet."];
```

```
AnalysePrint[Paragraph[In, 1]]
```

If it rains then the streets are wet.

$$\left( \begin{array}{l} P(1) \rightarrow \text{it rains} \\ P(2) \rightarrow \text{the streets are wet} \end{array} \right)$$

```
Paragraph[Set, Deduce, 1]
```

(! it rains ∨ the streets are wet)

- A second paragraph

```
Paragraph[2, "It rains."];
```

```
AnalysePrint[Paragraph[In, 2]]
```

```
It rains.
```

```
( $P(1) \rightarrow$  it rains)
```

```
Paragraph[Set, Deduce, 2]
```

```
it rains
```

- Combine the paragraphs.

```
Paragraph[Set, In, All]
```

```
If it rains then the streets are wet. It rains.
```

- The command below doesn't call Deduce yet. It merely applies the And & Or rules if there is enhancement. We use this command to show the use of ChooseP[ ].

```
result = Paragraph[Deduce, All]
```

```
If it rains then the streets are wet. It rains.
```

```
(
$$\begin{pmatrix} P(1) \rightarrow \text{it rains} \\ P(2) \rightarrow \text{the streets are wet} \end{pmatrix}$$
)
```

```
(( $\neg(P(1)) \vee P(2)$ )  $\wedge$   $P(1)$ )
```

```
ChooseP[{1}, True]
```

```
{ $P(1) \rightarrow$  True}
```

```
result /. %
```

```
 $P(2)$ 
```

```
% /. PropositionMeaningRule
```

```
the streets are wet
```

## 4.4 Logic laboratory

---

### 4.4.1 Summary

You may wish to experiment with various rules on various logical statements. For example, you may wish to extend the AndOrRules used in Enhancement. In that case the procedure LogicLab comes into use.

### 4.4.2 Loading the package

```
Economics[Logic]
```

### 4.4.3 LogicLab

*LogicLab*[*r*, *x*] | *LogicLab*[{*r*, *x*}]

If *r* is a List and *x* not, then all rules are applied simultaneously.

If *r* is a List and *x* too, then the procedure does Map[LogicLab[*r*, #]&, *x*].

If LogicLab[{*r\_List*, *x\_List*}] then Outer is used

Hence, to apply separate rules in *r* on a single *x*, do LogicLab[{*r*, {*x*}]}

LogicLab gives:

- (1) in print form:
- (1a) the input statement *x*,
  - (1b) *y* = LogicalExpand[*x*],
  - (1c) *z* = *y* //. *r*,
  - (d) the truthtable result
- (2) in output: the collection of truthtable results

### 4.4.4 Example

- The following is a valid deduction.

```
LogicalExpand[x || b1 && x && b2]
```

*x*

- Let us define this now as a general rule (that we might want to include in AndOrRules).

```
rule1 = (x_) || ((b1____) && (x_) && (b2____)) → x
(x_ ∨ (b1____ ∧ x_ ∧ b2____)) → x
```

- Here are some sample statements.

```
propos = {a && (b || c), a || (b && a && b)}
{(a ∧ (b ∨ c)), (a ∨ (b ∧ a ∧ b))}
```

- The logic laboratory shows that the rule has no effect on the first proposition.

```
LogicLab[rule1, propos[[1]]]
```

```
(a ∧ (b ∨ c))
```

```
((a ∧ b) ∨ (a ∧ c))
```

```
((a ∧ b) ∨ (a ∧ c))
```

```
(
  {a, b, c} → True
  {a, b, !c} → True
  {a, !b, c} → True
  {a, !b, !c} → True
  {!a, b, c} → True
  {!a, b, !c} → True
  {!a, !b, c} → True
  {!a, !b, !c} → True
)
```

```
(
  {a, b, c} → True
  {a, b, !c} → True
  {a, !b, c} → True
  {a, !b, !c} → True
  {!a, b, c} → True
  {!a, b, !c} → True
  {!a, !b, c} → True
  {!a, !b, !c} → True
)
```

## 4.5 Inference with the axiomatic method

---

### 4.5.1 Summary

This section investigates rule based inference for propositional logic. **Infer** is a large array of rules that has been composed automatically by InferenceMachine. Repeated replacement using Infer gives a logical conclusion. The method still suffers from the general weakness of the axiomatic method, that any truth can be derived from false premisses.

**Economics [Inference]**

### 4.5.2 Introduction

We now investigate the axiomatic method. Some relations are posed as axioms, and conjectures may be proven by repeated application of these axioms. The truth-table method that we discussed above is strong for cases where one has a definite idea about the relation to be proven. The axiomatic method on the other hand has the advantage that forms may be created that one had not thought of before.

In the `Logic`` package we already encountered the set of rules **AndOrRules**. Here we meet **Infer**, a large array of rules that has been composed automatically by InferenceMachine.

Though *Mathematica* is a rule based program by itself and has remarkable capabilities in pattern recognition, it still is not as simple as it might seem to develop an inference machine. The problem resides in the fact that pattern recognition remains a complicated affair.

For example it turns out that `Not[p]` is not always recognised as a proposition on a par with `p`. When we have an axiom `(p_ => !p_) :> !p`, then:

```
axiom = (p_ => !p_) :> !p
```

```
(p_ => !p_) :> !p
```

```
q => !q      /. axiom
```

```
!q
```

```
(!q => q)    /. axiom
```

```
(!q => q)
```

Similar pattern recognition problems seem to exist for other basic properties, like the antisymmetry of `If`.

Given this problem with pattern recognition, we best distinguish between (1) the axioms proper, i.e. the axioms as we wish them to hold, as expressed by a replacement pattern, and (2) the metarules, that allow *Mathematica* to recognise and apply the axioms.

With respect to these metarules, we still have levels of complexity. The first level of metarules in pattern recognition is given for example by patterns defined by hand, as is the case for AndOrRules or SelfImplication. A second level of metarules arises when we use properties like symmetry and embedding within BlankNullSequences, and use brute force by having *Mathematica* generate all combinations. For straightforward patterns both approaches may come down to the same. Of course it is most elegant - a third level - when a general pattern can be defined, for example that Not[p] comes on a par with p, but when the developer has not yet solved his pattern problem, brute force may be a good alternative.

The InferenceMachine provides some brute force, and allows you to formulate your own rules and to expand them.

A problem that is inherent in the axiomatic method is that any truth can be derived from false premisses. In Latin, this is the Ex Falso Sequitur Quodlibet situation. An example is the statement (truth): "If it rains and does not rain at the same time, then I'll give you all my money." This EFSQ problem is not solved here.

4.5.3 Infer

Infer is composed by InferenceMachine when the Inference` package is loaded by *Mathematica*. Infer is standardly based on the axioms SelfImplication, ModusPonens, EFSQ, IfTransitive, and IfToAnd, while using metarules AndSymmetric, OrSymmetric and IfAntiSymmetric.

x //. Infer	tries to solve a logical statement x
InferQ	gives suggestions on the use of the package
InferAndOr	AndOrRules ~Join~ Infer



## InferQ

The  $\Rightarrow$  will here have the function of implication within the object language.

On terms: Object language constants are  $P[1]$ ,  $P[2]$ , ... and variables are  $p1$ ,  $p2$ , .... Commonly, the term 'axiom' is used for an (object language) expression like  $p1 \Rightarrow p1$  which one accepts as true, for variable  $p1$ . In other words  $(p1 \Rightarrow p1) \text{ :> True}$  (concludes to True) for every substitution of a constant. Commonly, next, there are rules that allow manipulation of those axioms. Here however, those replacement rules are much more in focus. It becomes rather natural to use the term 'axiom' for  $p\_ \Rightarrow p\_ \text{ :> True}$ . Our somewhat deviant use of terms is likely caused by the fact that patterns and variables are different.

a) To allow better manipulation of patterns, define axioms in terms of symbols

$p$ ,  $q$ , ... Of course, use logical operators And, Or, Xor, Not,  $\Rightarrow$  (Implies), True and False. Use RuleDelayed for the result. E.g.  $(p \Rightarrow p) \text{ :> True}$ .

You can use patterns, so that your axioms are replacement rules. Note that  $(p\_ \Rightarrow p\_)$  gives a warning message. Then use LogicalPattern[ $p \Rightarrow p$ ]. Perhaps, for the distinction with the metarules, you don't want to use patterns for the axioms, and have these automatically inserted later.

Note: since you would like to use repeated replacement, your axioms should simplify rather than expand.

b) Give metarules (e.g. on symmetry) with normal patterns  $x\_$ ,  $y\_$ , ... in the premisses. These metarules must be single, named and in MetaRuleForm.

c) Each axiom can be inputted in ExpandAxiom with the metarules for its operators. Options allow replacement of symbols  $p$ ,  $q$ , ... with patterns (in the premiss), and If with Rule.

d) Joining these expansions gives YourSetOfRules.

e) Analysis of object language logical expressions like  $x = (p1 \Rightarrow q1)$  works like this:

$x /. \text{YourSetOfRules}$  or  $x //. \text{YourSetOfRules}$

f) The default inference rule is Infer.

It presumes AndOrEnhancement. Otherwise use InferAndOr.

g) Infer has a preference for both using  $\Rightarrow$  and solving by substitution.

■ A simple success is:

```
(p => q) && (q => r) && p // . Infer
```

*r*

■ Perhaps more complex is:

```
(!r => !q) && (!r => p) && (p => q) // . Infer
```

*r*

### 4.5.4 Axioms and metarules

In mathematics, all axioms are treated the same. *Mathematica* forces us to be more specific. Metarules will here be rules that will be applied to the axioms in order to increase the likelihood that the axiomatic pattern is recognised.

ModusPonens	axiom ((p_ => q_) && p_) :> q
IfToIf	axiom (p_ => (q_ => m_)) :> ((p => q) => (p => m))
IfToAnd	axiom ((p_ => q_) => m_) :> ((!p => m) && (q => m))
EFSQ	axiom (!p_ => (p_ => q_)) :> True
IfTransitive	axiom ((p_ => q_) && (q_ => r_)) :> (p => r)
SelfImplication	axiom (!p_ => p_) :> p and (p_ => p_) :> True

Note: EFSQ = Ex Falso Sequitur Quodlibet: From False follows anything you want.

AndSymmetric	metarule (x_ && y_) :> (y && x)
OrSymmetric	metarule (x_    y_) :> (y    x)
IfAntiSymmetric	metarule (x_ => y_) :> (!y => !x)

The following routines are used to get Not[p] on a par with p:

Deny[x]	gives a rule for denial of x: $x \rightarrow \text{Not}[x]$ ; x can be a list
DenyPattern[ <i>pattern</i> , x, prin:True]	replaces x_ with !(x_) within pattern and x with ! x (for conclusions). Default prints

4.5.6 InferenceMachine

The routine that puts everything together is InferenceMachine. By 'expanding' an axiom we will mean the application of the metarules so that an axiom is replicated in all its patterns.

InferenceMachine[ <i>opts</i> ]	expands the axioms using the metarules, creating a list of replacement rules
---------------------------------	---

InferenceMachine is directed by various options. The input options are: HoldAll → {axioms taken as they are}, Hold → {axioms that already conclude to True; expandable}, AxiomToTrue → {axioms that don't conclude to True, and you want to add 'axiom :> True'; expandable}, ExpandAxiom → {axioms which will be expanded}, PlacedProperties → {And → {...}, If → {...}, ...} are the metarules (entered as Strings so that they will not evaluate at the call) that should hold for any instance of the operators.

- An example is the definition of Infer at start up:

```
Infer = InferenceMachine[
  HoldAll → {SelfImplication},
  Hold → {EFSQ},
  AxiomToTrue → {ModusPonens, IfTransitive, IfToAnd},
  ExpandAxiom → {ModusPonens, IfTransitive, IfToAnd},
  PlacedProperties →
    {Or → "OrSymmetric", And → "AndSymmetric",
     Implies → "IfAntiSymmetric"}];
```

#### 4.5.7 The axiomatic method and EFSQ

It turns out that Infer is not without a paradox. We can trace this paradox to the Ex Falso Sequitur Quodlibet situation, or, that from a falsehood anything can be derived. It is useful to be aware of this, for otherwise we might conclude that there is an error in our InferenceMachine. The following is a crucial example.

- The following applies the transitivity of If:

```
(x ⇒ !x) && (!x ⇒ x) /. Infer
```

```
(x ⇒ x)
```

```
% /. Infer
```

```
True
```

- While SelfImplication gives:

```
(x ⇒ !x) && (!x ⇒ x) //. SelfImplication
```

```
(!x ∧ x)
```

```
% //. InferAndOr
```

```
False
```

The logical analysis of the situation is as follows. The proper answer is provided by SelfImplication, and the statement is False. Given that there is a falsehood, anything can be derived, such as the results True (the first approach).

Above problem seems to be related to the fact that IfTransitive dominates SelfImplication. But the true problem is of a more general nature. If we would change the order, we would run into another case where EFSQ causes problems.

It thus turns out that Infer is rather risky for application to real world questions. It may generate True, by *valid* deduction, while the proper answer is False. For this reason there has been no effort to extend

Decide[ ] with Infer, and to create an InferEnhance[ ] mode as with AndOrEnhance[ ]. This does not seem too bad, as one can always fall back on the method with the truth tables.

In fact, if one wishes to save the axiomatic method, then the proper approach would seem to be to have various deductions parallel to each other. Eventually, this amounts to the same as the truth-table method.

#### 4.5.8 Expansion subroutines

```

AndEmbedding[r]           embeds p_ && q_ within BlankNullSequences,
                           while the pattern-names are added
                           to the last part (assumed to be the conclusion)

AxiomToTrue[x, y:Implies, adjustable:True]

    gives (x /. RuleDelayed → y) :> True.
    Constructs the rule: whole axiom :> True.
    If adjustable, then the conclusion is patterned if the premiss was

ExpandAxiom[axiom, opts]

    applies the metarules for Implies, And, Or & Xor,
    for all separate occurrences of these operators.
    The metarules must be given as operator →
    name or operator → {name1, name2, ...} where namei
    are Strings. ($namei will be the copy for the occurrences.)

```

Default options for ExpandAxiom are: MachineForm → True applies MachineForm to the axiom first, First → True means that the metarules only apply to the premiss, LogicalPattern → True gives patterns in output, Rule → True replaces Implies with Rule, AndEmbedding → True embeds p && q within BlankNullSequences, Union → True flattens the result and thereby removes possible duplications. Possible options are default rules for Implies, And, Or & Xor.

#### 4.5.8 Different forms

The analysis gives rise to various forms, namely MachineForm, ObjectForm and MetaRuleForm. Actually, Rule and Pattern are not explicitly called 'Form', but still are forms of a logical statement.

<code>MachineForm[x]</code>	<code>= UnPattern[x] /. Rule → Implies.</code> Axioms are internally set to MachineForm, to facilitate matches
<code>ObjectForm[x]</code>	<code>= UnPattern[x] /. RuleDelayed → Rule.</code> Transforms into object language form, including :>
<code>MetaRuleForm[x]</code>	<code>= LogicalPattern[MachineForm[x]]</code>
<code>ToRule[x]</code>	<code>= LogicalPattern[x] /. Implies → Rule.</code> Transforms object language expressions into rules
<code>LogicalPattern[x]</code>	changes a logical expression x into a logical pattern. Default option First → True applies pattern only to the premiss

#### 4.5.9 Accounting

The InferenceMachine works by assigning a marker to each occurrence of a logical operator. The metarules then are applied to each operator individually. After this, the markers are removed again.

<code>PlacedMetaRule[name_String]</code>	subroutine for ExpandAxiom and PlaceProperties, derives a placed metarule from the general metarule
<code>PlacedProperties[ len, x]</code>	subroutine for ExpandAxiom, gives each placed operator its own list of metarules; len is the number of occurrences of the operator, and x gives the list of general metarules for that operator
<code>Place[Operator, i]</code>	marks the place of the ith operator
<code>PlaceOperator[expr]</code>	gives each operator its place mark
<code>PlaceUndo</code>	<code>= {Place[x_, y_] → x}</code> which removes Place marks

## 5. Economics

### 5.1 Introduction

---

Remarkably, it took 4 chapters of preliminaries, but finally, in this 5th chapter, we get to doing economics. The Pack has collected so many subjects by now that this chapter starts to read like a course in economics.

We start out with the key economics subject: social welfare. Practical work requires us to drop one of Kenneth Arrow's axioms, and we arrive at some reasonable constitutions for group decision making.

We next discuss the `Economic`Common`` package that provides common symbols and that tries to provide for individual optimising behaviour and a general equilibrium environment. The Constant Elasticity of Substitution function is commonly used for utility or production analysis, and the `CES`` and `CES-like` packages are implementations of the structure provided by `Economic`Common``. The subsequent `AGE`` package for applied general equilibrium theory is wholly embedded in the `Economic`Common`` structure and finds employment for the CES and CES-like functions.

The other packages still rely on the `Economic`Common`` symbols, but don't use its structure (yet). The package on macro economics instead relies on the `SolveFrom` construction, that allows the user to set certain values and then let the routine determine the unknowns. The package on taxes and premiums extends the IS-LM context with heterogenous labour and income, and the discussion shows how a wrong tax structure causes an unnecessarily high minimum wage and how that minimum wage causes mass unemployment.

The discussion of game theory might as well have been put at the beginning, since game theory is basic for understanding decision making and social interaction. The relevance of the subject can however be better appreciated when some familiarity has grown with the concepts and apparatus of economics. Discussing game theory here provides a bridge to meso-economic subjects.

At the meso-economic level we meet monopolies and cartels. The package on these exploits the fact that a cartel basically is a monopoly with the additional problem of distributing production and profits over the members of the cartel. Subsequently, a typical problem for a public monopoly is peak load pricing - an example are the day and night tariffs for electricity. Peak load pricing is also an example of market segmentation and the theory of the consumers' surplus.

Wolfram Research Inc. already has a Finance Essentials pack on the market. The `Finance`` package included here in the Economics Pack supports this WRI pack, though it may be noted that many of its routines can be used by themselves. The `FlatRate`` package here introduces the basic finance concepts using the assumption of a flat (constant) rate of interest.

Next to finance, also transport happened to get special attention. There are more packages here, dealing with different aspects of the sector.

The reader may be advised to have the textbook by Andreu Mas-Colell, Michael Whinston and Jerry Green, "Microeconomic Theory", Oxford 1995 at hand. For macro economics a reference text is Dornbusch & Fischer (1994), "Macroeconomics", McGrawHill 6th ed. Please do not forget the books and software written by Hal Varian and his co-authors. Also, enclosed in the Economics Pack are papers with innovations on standard theory.

## 5.2 Social welfare and voting

---

### 5.2.1 Summary

This package develops some consistent constitutions for group decision making and social welfare. These constitutions violate Kenneth Arrow's axiom of "Independence of Irrelevant Alternatives", but still can be reasonable and morally desirable.

**Economics [Voting]**

### 5.2.2 Introduction

The analysis of social welfare is the key subject of economics. One of the major analytical results of 20th century economics would be Arrow's Theorem. In 1951 Kenneth Arrow formulated a set of axioms that many would consider reasonable and morally desirable, and he then showed that these axioms result into a contradiction. The conclusion would be that there would exist no good constitutions - and this is indeed accepted by many economists.

In my analysis, however, the body of current economic analysis on this topic is rather misguided. While Arrow's theorem is mathematically sound, there still is the matter of economic interpretation. It is possible to define good constitutions, i.e. reasonable and morally desirable. This package implements a couple of them.

The constitutions implemented here violate Arrow's axiom of "Independence of Irrelevant Alternatives". A realistic decision maker cannot accept this axiom. For example, in a choice among three possibilities *A*, *B* and *C* (see the Condorcet situation discussed below) the group choice on only two items *must* include the votes on the other item, and thus the choice is not independent of these. The reason is that if one can determine a voting cycle then the group decision is indifference rather than some preference. There is the subtle difference between voting and deciding; it is no problem that voting patterns show a cycle (of indecision), what counts is that the constitution results in a consistent decision. Note that since we reject Arrow's axiom, we are consistent. The "voting paradoxes" are paradoxes i.e. seeming contradictions indeed, and no real contradictions. See my bibliography for more details on this issue.

Consider these three constitutions - included in the `Constitutions[]` call:

- **Pareto (efficiency) majority:** Only those items under choice are considered that benefit some and that are not to the disadvantage of anyone. Note that efficiency depends *absolutely* on the Status Quo, and is not *relative*. If there are more such items without a clear order, then a Fixed Point Borda majority decision is used. If this locates cycles of fixed points, then the vote margin count over the whole budget set is used to break the tie.
- **Borda (-majority):** each voter gets *N* (say *N* = 100) points, and may distribute these across the items under choice. The item with the highest sum of points is selected.



- **Pairwise majority:** Items are brought to the floor in pairs, and decided upon by normal majority of "pro" and "contra". If a cycle occurs then there is indifference (a deadlock).

When there is a deadlock or indifference over the whole set, then the Status Quo is maintained. Then you have to provide additional decision rules, such as random selection (throwing dice) etcetera. The pairwise majority routine e.g. conservatively selects the previous best, since pairwise majority itself has no standard approach to solve deadlocks. In all cases, we concentrate on picking the winner, rather than constructing the collective welfare index - thus we use a 'social decision functions'. Not deriving the full ordering is a matter of efficiency. (Look at this [hyperlink](#) to read more about the book "Voting Theory for Democracy", Cool (2001).)

**?VotingQ**

The principle of the Voting package is as follows. The major objects are preferences, represented by lists. A preference list of a single voter gives a row *v* of values, where *v*[[*i*]] gives the value attached to the *i*th item, with a greater number for a greater value. The Preferences matrix has NumberOfVoters rows for voters and NumberOfItems columns of preference values. These preferences are transformed by Borda[], ParetoMajority[] or PairwiseMajority[] to generate Group results. DuncanBlackPlot[] plots the preferences for all voters.

A preference list (for a single voter, or for the Group result) can be transformed into a VoteMargin[] object that gives another representation of the results of pairwise comparisons. The VoteMargin[] representation allows for cycles like *A > B > C > A*. Importantly, since preferences represented by lists cannot produce cycles, the VoteMargin[] representation is (primarily) relevant for the Group result. VoteMarginToGraph[] allows a translation to the existing Graph functions.

**5.2.3 Key concepts**

There are the items to be voting about:

Items	a list. You would set your own Items = {...}. The default has NumberOfItems elements of the Alphabet
NumberOfItems	Must be set to the number of Items considered
StatusQuo[]	gives the item that represents the status quo. By default the first of Items

Note that the default items are Strings "A", "B", ... and that *Mathematica* does not normally print ""s.

Then there are the voters. A voter does not have to be a single individual, but can also represent a party with a certain percentage of the vote. Each voter is associated with a preference ordering.

NumberOfVoters	must be set to the number of voters considered
Preferences	Preferences is a {NumberOfVoters, NumberOfItems} matrix (list of lists) for the values assigned to the items, in the order of Items. A higher value means a higher priority. Thus {{1, 2}, {1, 2}} means that there are two voters and that both assign a higher value to B rather than A
SetPreferences[x]	checks on x, sets NumberOfVoters and NumberOfItems. It assigns equal voting power if the existing votes don't match.
Votes	gives the list of votes per voter. The sum must add to unity. The default for 3 voters is PM[{.25, .35, Rest}]

Note: PM is the probability measure input facility of `Statistics`Common``.

- The package provides the Condorcet example.

**ShowPrivate[Condorcet]**

```
Condorcet[] sets key parameters to the example
voting paradox given by Marquis de Condorcet 1785

Condorcet[] := (NumberOfItems = 3; NumberOfVoters = 3;
  Items := Table[FromCharacterCode[i], {i, 65, 64 + NumberOfItems}];
  StatusQuo[] := First[Items];
  Preferences = {{1, 2, 3}, {2, 3, 1}, {3, 1, 2}};
  Votes = PM[{0.25, 0.35, Rest}];)
```

**Condorcet[]; CheckVote[]**

{Number of Voters → 3, Number of items → 3, Votes are nonnegative and add up to 1 → True,  
Preferences fit the numbers of Voters and Items → True,  
Type of scale → Ordinal, Preferences give a proper ordering → True,  
Preferences add up to → {6}, Items → {A, B, C}, Votes → {0.25, 0.35, 0.4}}

**5.2.4 Pareto (efficiency) majority**

ParetoMajority[ <i>p</i> :Preferences, <i>v</i> :Votes, <i>i</i> :Items, <i>s</i> :StatusQuo[]]	first selects the Pareto points that dominate the Status Quo, and then applies the BordaFP majority rule to those. Selected points thus always satisfy the efficiency criterion
---	---

If  $B > A$  and  $C > A$  are both Paretian improvements (from the Status Quo  $A$ ), while there is no clear efficiency preference on  $\{B, C\}$ , then there might still be a deadlock. The ParetoMajority rule solves this deadlock by Fixed Point Borda majority voting. A final deadlock of indifference by still remaining equal votes is left to the user. You may select the Status Quo, use dice, etcetera.

- ParetoMajority[ ] applied to the Condorcet situation gives:

**ParetoMajority[ ]**

{StatusQuo  $\rightarrow$  A, Pareto  $\rightarrow$  {A}, Select  $\rightarrow$  A}

There are two other routines that first select the Pareto improvements from the Status Quo. They seem less attractive than the former (see the discussion in the Resume).

ParetoBorda [ <i>p:Preferences, v:Votes</i> ]	first selects the EfficiencyPairs that dominate the Status Quo, and then applies the (plain) Borda majority rule to those
ParetoPairwise [ <i>p:Preferences, v:Votes</i> ]	collapses the preferences to the Pareto points, and then applies the pairwise voting scheme

Subroutines are:

EfficiencyPairs [ <i>p:Preferences</i> ]	
	gives the ordered pairs in items that satisfy the efficiency criterion, i.e. those pairs {item1, item2} where someone improves while nobody is hurt by the group taking item2 instead of item1
ListToOrderedPairs [ <i>preference list, items</i> ]	
	changes the preference list into a list of ordered pairs
NOrderPair [ <i>preference list, {i, j}</i> ]	
	makes the ordered pair for numbered items i and j, based on the preference list for one voter. If there is indifference between item1 and item2, both {item1, item2} and {item2, item1} occur. There is a strict preference if the opposite pair does not occur.

Note that *p:Preferences* indicates a matrix and *preference\_List* indicates a vector.

PPath [{ <i>a, b</i> }]	gives the list of pairs that together form a path from <i>a</i> to <i>b</i> . PairsToPaths has to be performed first.
PairsToPaths [ <i>pairs_List</i> ]	chains pairs to form paths. Output are the pairs that form beginning and end of the longest paths, and with the intermediate pairs thus eliminated. Note: cycles are not looked for. The result is stored in PPath

5.2.5 Borda

<code>Borda[<i>p:Preferences</i>, <i>v:Votes</i>, <i>i:Items</i>]</code>	chooses the items with the maximum in the BordaField <i>v</i> . <i>p</i>
<code>BordaFP[<i>p:Preferences</i>, <i>v:Votes</i>, <i>i:Items</i>]</code>	first collapses to the cycle of fixed point winners, then applies Borda to this selection
<code>BordaField[ <i>p:Preferences</i>, <i>v:Votes</i>]</code>	simply applies the Votes to Preferences: <i>v</i> . <i>p</i>
<code>BordaAnalysis[ <i>p:Preferences</i>, <i>v:Votes</i>, <i>i:Items</i>]</code>	analyses the situation for a Borda type of vote: 1) the selected items 2) the BordaField 3) the positions of the maxima 4) the items sorted from lowest to highest weighted vote 5) whether the selection are fixed points

Note: A fixed point A wins from the alternative winner B - with the alternative defined as the match when A does not participate.  
BordaFPQ tests whether a point is a Borda fixed point.

In the following example there is no difference between Borda and BordaPF. (But see this notebook on preference reversals (hyperlink).)

■ Borda[ ] applied to the Condorcet situation - with the present voting weights:

**Borda[ ]**

A

**BordaAnalysis[ ]**

{Select → A, BordaFPQ → {True},

WeightTotal → {2.15, 1.95, 1.9}, Position → ( 1 ), Ordering →  $\left( \begin{array}{cc} 1.9 & C \\ 1.95 & B \\ 2.15 & A \end{array} \right)$

5.2.6 Pairwise majority

If we count just wins or losses, then this either gives the Condorcet winner or the Status Quo (output indicated by 1). If we add the vote margins, then the highest row sum gives the winner (output indicated by N). If the Condorcet method results into a cycle, then we could use the vote margins to break the tie (output indicated by All.). Sometimes we want the routine to also show: (1) the matrix of voters and the pairs under consideration, (2) the matrix of votes cast per voter and pair, (3) the total vote per pair.

### 5.2.6.1 The main routine

<code>PairwiseMajority[ p:Preferences, v:Votes]</code>	works from the <code>VoteMargin</code> object
<code>PairwiseMajority[ Show, p:Preferences, v:Votes]</code>	takes the <code>PairwiseField[]</code> of <code>NPairs[]</code>

Note: The occurrence of indifference causes output of a List. Note: You can evaluate `VoteMargin["Explain"]`.

Subresults: (1a) Sum → the 1 / 0 row sums, (1b) Max → the maximal value of this. If this equals `NumberOfItems - 1`, then there is a Condorcet winner, (1c) Pref → gives the Pref object (1d) Find → takes the last element in Pref, (1f) LastCycleTest → True / False if the latter is not unique, (1g) Select → either the unique Condorcet winner or the Status Quo. (Na) Sum → the row sums of the `VoteMargin` object, (Nb) Pref → gives the Pref object, (Nc) Select → the item with the highest preference.

#### **PairwiseMajority[Show]**

`VoteMarginToPref::cyc : Cycle {C, A, B, C}`

$$\left\{ \text{Outer} \rightarrow \begin{pmatrix} \{1, \{A, B\}\} & \{1, \{A, C\}\} & \{1, \{B, C\}\} \\ \{2, \{A, B\}\} & \{2, \{A, C\}\} & \{2, \{B, C\}\} \\ \{3, \{A, B\}\} & \{3, \{A, C\}\} & \{3, \{B, C\}\} \end{pmatrix}, \text{Pairwise} \rightarrow \begin{pmatrix} \{0, 0.25\} & \{0, 0.25\} & \{0, 0.25\} \\ \{0, 0.35\} & \{0.35, 0\} & \{0.35, 0\} \\ \{0.4, 0\} & \{0.4, 0\} & \{0, 0.4\} \end{pmatrix}, \right.$$

$$\text{Sum} \rightarrow \begin{pmatrix} \{A, B\} & \{0.4, 0.6\} \\ \{A, C\} & \{0.75, 0.25\} \\ \{B, C\} & \{0.35, 0.65\} \end{pmatrix}, \text{VoteMargin} \rightarrow \text{VoteMargin} \left( \begin{pmatrix} 0 & -0.2 & 0.5 \\ 0.2 & 0 & -0.3 \\ -0.5 & 0.3 & 0 \end{pmatrix} \right),$$

1 → {StatusQuo → A, Sum → {1, 1, 1}, Max → 1, No Condorcet winner → {A, B, C},  
 Pref → Pref({A, B, C}), Find → {A, B, C}, LastCycleTest → True, Select → A},  
 N → {Sum → {0.3, -0.1, -0.2}, Pref → Pref(C, B, A), Select → A}, All → A}

### 5.2.6.2 Pairs and pairwise

<code>Pairs[i:Items]</code>	gives the list of unordered pairs in i
<code>NPairs[n:NumberOfItems]</code>	gives the list of unordered pairs for the set {1, ..., n}
<code>Pairwise[ preference list, pair_List]</code>	gives the element from pair that has highest preference. It is assumed that pair is a list of two numbers, where the numbers give the positions of items in Items
<code>Pairwise[preference list, v]</code>	assigns votes v to the position with highest preference

Note: Pairwise uses the majority rule, but you are free to give another implementation.

5.2.6.3 A pairwise voting field is not yet a decision

<code>PairwiseVoter[n_Integer, pair_List]</code>	performs the pairwise vote for voter n using his preferences and voting power
<code>PairwiseField[list of pairs]</code>	applies PairwiseVoter to all voters, for that list of pairs
<code>GroupOrderQ[x, y]</code>	is set by PairwiseField[] and gives the implied preference order. The order is the same as LessEqual: the last position in the list is the most important

Note: A pair here is a list of two numbers, where the numbers give the positions of items in Items.

5.2.7 The Pref[...] object

The list format assigns values to positions, as in the preference list {1, 3, 2} (that means that element 2 is preferred most, and element 1 is preferred less). An alternative representation is to give positions based on preference values: this gives the Pref[ ] object.

<code>Pref[x1, ..., xn]</code>	gives a preference order from the lowest preferred x1 to the highest preferred xn. The position in the order in fact gives the ordinal preference value. Elements of equal preference are put in sublists, such as Pref[x1, {x2, x3}]
<code>ListToPref[ preference list]</code>	turns a list into a Pref object, using Items
<code>PrefToList[Pref[...]]</code>	turns a Pref object into a list again

The Pref[..] object has not been taken as the basic programming object since it gives less information, and since the size of the gap between the various alternatives can best be put into numbers. However, for pairwise majority voting, the Pref[..] object does good service to help eliminate cycles in the final selection.

- Comparing a Pref object with a list object.

```
ListToPref[{5/2, 5/2, 1}]
```

```
Pref(C, {A, B})
```

There is also a format ListToPref[Order, list (, q)] that uses subroutines:

`ListToPrefOrderQ[preference List, q:PrefOrderQ]`

uses the preference list to define the *q* sorting criterion, with default PrefOrderQ

**PrefOrderQ** PrefOrderQ is a head only, that may get a definition by ListToPrefOrderQ. If an order is defined, then it can be used for Sort. PrefOrderQ[i,j] must give False if the first element in a pair {i,j} comes before the last, according to the stated preference list. A pair is assumed to consist of elements of Items

### 5.2.8 The VoteMargin[...] object

Preferences can also be recorded as the values of pairwise comparing *i* with *j*. This gives a matrix instead of a list.

`VoteMargin[{row1, row2, ..., rown}]`

the outcomes of pairwise comparisons of *n* items. For votes: if  $V[i, j]$  are the votes for *i* in the match with *j*, then  $P[i, j] = V[i, j] - V[j, i]$

`ListToVoteMargin[preference List]`

changes a preference list into a VoteMargin[...] object, by assuming that the utility difference between the items is simply the difference of the values in the preference list

Thus each element  $\text{VoteMargin}[i, j]$  is the outcome of a preference consideration. Assumed is that 0 means Indifference, and it applies to the diagonal (in the plot from bottom left to top right). A positive value means that *i* is better than *j*, a negative value conversely. The size of the value may matter, depending upon the application. If  $\text{VoteMargin}[i, j] + \text{VoteMargin}[j, i] \neq 0$ , then the preference pairs are 'irrational', as sometimes happens in experiments. (Note that it is useful to keep the word 'irrational' between quotes, since science by definition will try to find a rational explanation for what happens.) Evaluate  $\text{VoteMargin}["\text{Explain}"]$ . (Note: An old name for VoteMargin was PrefPairs.)

`ListToVoteMargin[Preferences[[1]]]`

$$\text{VoteMargin} \left( \begin{pmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{pmatrix} \right)$$

Subroutines here are:

<code>VoteMarginToOrderQ[<i>pref_VoteMargin</i>, <i>q</i>]</code>	translates the <code>VoteMargin</code> object into a sorting order criterion <i>q</i> , that can be used for <code>Sort</code> . For example <i>q</i> = <code>PrefOrderQ[i]</code> for the <i>i</i> th voter
<code>SetRandomVoteMargin[<i>n:NumberOfItems</i>, <i>type:Integer</i>, <i>ran_List</i>: {-1, 1}]</code>	creates a <i>n</i> x <i>n</i> matrix of <code>Random[<i>type</i>, <i>ran</i>]</code> values, though with diagonal 0. The Head <code>VoteMargin</code> is added, to distinguish this matrix from the normal preference ordering that is represented by a List
<code>VoteMarginPlot[<i>pp_VoteMargin</i>, <i>opts___Rule</i>]</code>	density plot of a <code>VoteMargin</code> object

5.2.9 Smaller tools

<code>ExamplePrefs[<i>n</i>]</code>	for <i>n</i> =1,2 give example Preferences for 3 voters and 3 items (with e.g. weighted voting)
<code>NPrefOrderQ[<i>pair_List</i>, <i>preference_List</i>]</code>	can be used for <code>Sort</code> , and gives False if the first element in pair comes before the last, according to the stated preference list. Pair is assumed to consist of numbers
<code>SetRandomPreferences[]</code>	sets the Preferences to random orderings
<code>SetRandomPreferences[<i>n</i>]</code>	sets the number of items to <i>n</i> , and then generates random preferences
<code>SetRandomPreferences[<i>m</i>, <i>n</i>]</code>	sets the numbers of voters and items to <i>m</i> and <i>n</i> resp., and then generates random preferences

- Preference list {4, 1, 3, 2} means that the second element has the least value, then comes the last element, then the third, while the first element has highest value.

Sorting:

```
Sort[{1, 2, 3, 4}, NPrefOrderQ[{{#}}, {4, 1, 3, 2}] & ]
{2, 4, 3, 1}
```



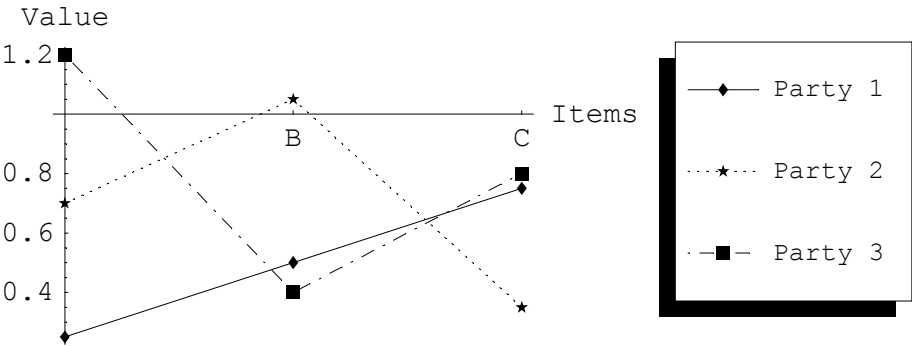
<code>ProperPrefsQ[<i>mat</i>]</code>	Option <i>N</i> → <i>m</i> determines this test: ordinality for <i>m</i> = Automatic, interval/ratio scale for <i>m</i> a number, and cardinality for <i>m</i> Infinity. If Automatic, then <i>m</i> = 1 + ... + <i>n</i> .
<code>StrictRisingPrefsQ[<i>mat</i>]</code>	gives True if the rows are permutations of {1, ..., <i>n</i> }
<code>DefaultItems[<i>(n)</i>]</code>	sets the items to A, B, C, ... and Status Quo[] := First[Items]
<code>EqualVotes[<i>(m)</i>]</code>	for <i>m</i> a Blank, takes NumberOfVotes, for <i>m</i> a Number sets NumberOfVotes, and sets Votes to a list of equal votes 1/ <i>m</i>

Plotting tools are:

<code>DuncanBlackPlot[<i>x</i>, <i>plotting opts</i>]</code>
plots the preferences <i>x</i> with the items on the x-axis and the values on the y-axis. May be used to see single-peakedness. Default for <i>x</i> are the Preferences weighted by the Votes
<code>PreferencesPlot[<i>p</i>:Preferences, <i>plotting opts</i>]</code>
gives a density plot of <i>p</i> (default not weighted with Votes)

- The following gives Duncan Black's plot of the Condorcet example that we have been using. The default plots Votes \* Preferences.

```
DuncanBlackPlot[TextStyle → {FontSize → 11},  
PlotLegend → {"Party 1", "Party 2", "Party 3"},  
LegendShadow → {-0.05, -0.05}];
```



### 5.2.10 For the link with Graphs

The package provides its own solution routines. Note that voting situations can be represented by Graphs, and that Steve Skiena's great `DiscreteMath`Combinatorica`` package deals with these. The `Voting`` package also provides routines to translate to Graphs, and one may benefit from those plotting and solution routines. The relation with Graphs is discussed in my book on voting theory, see this [hyperlink](#).

```
ShowPrefGraph[graph, opts]
```

shows a graph in a typical preference analysis situation: as a labeled directed graph. Adjust with options

```
ShowListGraph[x_List, opts]
```

performs `ListToVoteMargin`, `VoteMarginToGraph`, and `ShowPrefGraph`. The options apply to these and `Show`. For two elements you may want to adjust the `PlotRange`

```
ListToGraph[x_List, opts___Rule]
```

performs `VoteMarginToGraph[ListToVoteMargin[x], opts]`

```
VoteMarginToGraph[x_VoteMargin, SameQ→True]
```

represents indifference by a value 0 in the adjacency matrix. In that case the adjacency values represent the 'distances' between points, so two indifferent items are plotted at the same vertex

```
VoteMarginToGraph[x_VoteMargin, SameQ→False]
```

(default) sets all off-diagonal nonnegative values to 1, including zero values, so that all items can be plotted separately, and indifference is given by two arrows 'to and fro' points. Both results would be plotted with `ShowGraph[graph, Directed]`

## 5.3 Economic Common routines and other

---

### 5.3.1 Summary

The `Economic`Common`` package provides some basic functionality for economics. Only a limited result is possible however.

```
Economics[Economic`Common]
```

### 5.3.2 Introduction

The idea about the `Economic`Common`` package is that it should capture the backbone of doing economics on the computer. As *Mathematica* contains `Plus` and `Times` for doing mathematics, and as a `Statistics`Common`` package contains the `Mean` and `StandardDeviation` functions: in the same way one would want to see the basic functionality for economics.

Economics *does* have a "Standard Model". This is Lionel Robbins's definition of economics, or the Robinson Crusoe model of maximising utility subject to a production function and limited resources. Those limited resources can be either natural resources or a budget with given prices or expected price reactions. A variant is: maximise profits subject to a production function and limited resources. We can abstract from utility and profits by speaking about an *aggregator* in general. The optimum is derived by setting factor-derivatives of the aggregator to zero, and by then solving these ("normal") equations for the unknown optimal factors. Maximising utility or profits and minimising cost are "dual" to each other. Their properties "mirror" each other. Likely the simplest model here is linear programming.

Note, subsequently, that the Economics Pack already contains a Lagrange maximisation routine (second order conditions not yet included), and that the Standard Model is basically covered by this. Or we can refer to linear programming again, and use linear approximations.

- Result 1: To facilitate applications on this Standard Model, the `Economic`Common`` package contains an arbitrary aggregator function with factors and coefficients, Cost and Profit definitions, and demand functions. The functions supplied here are shells: they provide a common framework, but the user has to supply the actual routine. The `CES`` package is an example how one can use the shell for a particular function. The benefit of using a shell is that you can write routines that use the shell, while you can switch to various functions by adjusting the Aggregator option.

The subsequent question is whether we can do more than this. It is out of the question to design routines so that *any* economic problem can be tackled with them, since economic problems come in any shape.

To clarify the difficulty of progress, let us first state some dilemmas:

**Dilemma 1:** Develop a full model or just components ?

- The AGE` package develops a full model. The full model comes at the price of perhaps too simple components. And, the model itself can be criticised for not being dynamic, etcetera.
- The CES` package and its relatives develop just components.

**Dilemma 2:** Use algebraic or numerical methods ?

- *Mathematica* has opened the route for algebraic methods. But for certain applications the numerical methods can still be most efficient.

**Dilemma 3:** Develop mathematical models, or do applied economics ?

- The Estimate` package allows the estimation of systems of models. The AGE` package, though applied, however has not been written in a format so that it can be estimated in this manner ...

These dilemmas clarify that it is no easy task to do more than what we achieved on the Standard Model. However, if we are willing to accept that progress comes by little steps at the time, then it is a result of sorts when we can avoid name conflicts.

- Result 2: The Economic`Common` package contains a set of undefined economics Symbols, that can be employed in various other packages without causing name conflicts.

The combined result of 1 and 2 still is a pretty strong tool.

Since the following discussion is a bit complex, we will start with some main elements before discussing the role of the Options[Aggregator].

### 5.3.3 The main notion of aggregation

Let  $f[\ ]$  be an aggregator for factors  $x_1$  till  $x_n$ . If the  $x_i$  are factors of production, then  $f[\ ]$  is the production function. If the factors are consumption goods, then  $f[\ ]$  would be the utility function. Aggregation results into an aggregate value  $y$ . Assigning prices to all factors and the aggregate, we can determine a residue profit  $\Pi$ . In formulas, the aggregation and budget equation are:

$$y = f[x_1, \dots, x_n]$$

$$\Pi = p y - (p_1 x_1 + \dots + p_n x_n)$$

Aggregate	option for the aggregate value. See <code>List`</code> for added usage
AggregatePrice	option for the price of the Aggregate
BudgetEquation[ <i>opts</i> ]	gives the budget equation using only the keywords of the Options[current aggregator], with Profit giving the difference between $p \cdot y$ and $p \cdot x$

Note: Don't forget the possibility of taking inputs negative and outputs positive.

- The default number of factors is 2, and by default the aggregate and its price are normalised to 1. If we use non-default values:

```
BudgetEquation[Aggregate → y, AggregatePrice → p]
```

$$p \cdot y == \text{Profit} + \text{FactorP}(1) \text{FactorX}(1) + \text{FactorP}(2) \text{FactorX}(2)$$

The aggregator will have various coefficients. By default, we take account of two coefficients per factor, namely  $c_i$  and  $e_i$ . There can be more of course, but then the user must set `ResetFactors`.

Many procedures must recognise variables and parameters, and for such procedural recognition we need canonical symbols. Default symbols are `FactorX` and `FactorP` for factors and their prices, and `FactorC` and `FactorE` for the coefficients. When the `NumberOfFactors` is small then these symbols may seem ugly. However, these names provide a framework for thought that may be useful, especially for larger problems.

Factors	option
FactorX	element head in Factors
FactorPrices	option
FactorP	element head in FactorPrices
FactorCoefficients	option
FactorC	element head in FactorCoefficients
FactorPowers	option that may be added into Options[Aggregator]
FactorE	element head that may be added into Options[Aggregator], likely for exponential terms

In some cases it is useful to check whether the factor coefficients add up to 1.

UnitFactorCoefficientsQ[*opts*\_\_\_Rule]

takes the FactorCoefficients of the current aggregator (in *opts* or taken from Options[Aggregator]) and tests whether they add to 1. If the coefficients are symbolic, then output is the rule that they should add to 1.

5.3.4 Cost and profit

- Cost is regarded as a function of the output level and the factor input prices.
- Profit is regarded as a function of input and output and all prices.
- Cost and Profit need not satisfy efficiency.
- You may add the condition that all factors are on the efficiency frontier.
- Further optimality is provided by MinimalCost and MaximalProfit. Note: If only one factor is considered in profit maximisation, that single result need not correspond to an overall cost minimum and profit maximum.
- Profit maximisation generally presupposes cost minimisation.

These functions are defined for an arbitrary aggregator. They take the option Aggregator in Options[Aggregator] as default. As said, the AggregatePrice option is normalised to 1 by default.

Cost [ <i>opts</i> ]	gives cost $p_x \cdot x$
Cost [Function, <i>opts</i> ]	gives $p_x \cdot x$ where one factor $x_i$ has been solved as a function of the output level and the other input factors. Factor $x_i$ is selected by FactorPosition.
Profit [ <i>opts</i> ]	gives profit: $p \cdot f[x] - p_x \cdot x$
Profit [Function, <i>opts</i> ]	gives profits as a function of output $y = f[x]$ , where cost has been solve for factor $x_i$

**Cost[]**

$$\text{FactorP}(1) \text{FactorX}(1) + \text{FactorP}(2) \text{FactorX}(2)$$

**Profit[]**

$$\text{Aggregator}() - \text{FactorP}(1) \text{FactorX}(1) - \text{FactorP}(2) \text{FactorX}(2)$$

The following are shells. They provide a general mold for any aggregator, and the user can select the specific aggregator either (1) by naming it in the first position, or (2) by adding an input option Aggregator → myagg, or (3) by seting Options[Aggregator] to such a default value. Next to this shell, the

user still has to define himself or herself how the minimum cost or maximum profit is achieved. See the CES ` package for an example of how this can be done.

MinimalCost[(aggregator,) *opts*]

gives the minimal cost to produce Aggregate with given FactorPrices

MaximalProfit[(aggregator,) *opts*]

gives the results of profit maximisation  
with given AggregatePrice and FactorPrices. Output is  
conditional: If[ returns to scale >= 1, (\*then limited by resources\*) Infinity,  
(\*else\*) {Aggregate → ..., MinimalCost → ..., MaximalProfit → ...}]

Note: Minimal cost factor demand  $x^*[y, ps]$  would follow from  $\nabla[MinimalCost[ ]]$ , following Hotelling 1932 and Shephard 1953.

5.3.5 Aggregator control

The main control of the Economic `Common ` routines is exerted by the Options[Aggregator].

Aggregator	to select the aggregator
CurrentAggregator[ <i>opts</i> ]	gives the Aggregator option setting
Options\$Aggregator	stores the options at startup, for a ResetOptions[Aggregator]

The Options[Aggregator] are relevant in three ways:

- 1. The setting Aggregator → f directs routines to a specific aggregator (f = CES, IADS, ...)
- 2. Settings within Options[Aggregator] may work as a defaults for the Options[f].
- 3. The setting ResetFactors → ... gives directions to calls of ResetFactors[ (, f) ].  
Options[Aggregator] must be set in order to have ResetFactors work properly. (Setting the names and number of parameters in a function differs from "working with the function".)

Options [Aggregator]

{Aggregate → 1, AggregatePrice → 1, Aggregator → Aggregator,  
Constant → Scale, Cost → Cost, DemandFunction → Cost, Dual → Cost,  
DualQ → Null, Factor → FactorX(1), FactorPosition → Automatic,  
FactorPrices → {FactorP(1), FactorP(2)}, Factors → {FactorX(1), FactorX(2)},  
NumberOfFactors → 2, ResetFactors → ( FactorCoefficients FactorC ), Return → 1}

For example, when the option had been set to Aggregator → CES, then a routine like MinimalCost[ ] would be instructed to use the minimal cost definitions for the CES functions, and to use the Options[CES] for more information on the number of factors and such.

In general, the options of a specific aggregator, like the Options[CES], define what is required for that aggregator. If a certain piece of information is missing, then `Economic`Common`` routines fall back on the Options[Aggregator]. Let us therefor look at the various option settings.

We have already discussed the options `Aggregate`, `AggregatePrice`, `Factors` and `FactorPrices`. The other options and their settings are:

<i>option</i>	<i>typical default value</i>	
Constant	Scale	the constant markup of the Aggregator
Cost	Cost	the cost value. Here symbolic, but it might be a real number
DemandFunction	Cost	the type of demand function to use for the factors; must be Function, Price, Cost or Profit
Dual	Cost	specifies what kind of duality result is to be taken. An alternative setting is <code>Aggregate</code> .
DualQ	Null	True if the aggregator is indirect (dual), False if direct, Null if unknown
Factor	FactorX[1]	selects the factor for FactorDemand
FactorPosition	Automatic	If Automatic then the position of the factor is searched for. If n, then the nth position is taken; use this when Factors has (equal) numerical values.
NumberOfFactors	2	option for the number of factors
ResetFactors	{ {FactorCoefficients, FactorC} }	when resetting NumberOfFactors, adjust this too
Return	1	a returns to scale parameter. A symbolic value would be RTS

Note: The option values for `ResetFactors` must contain Strings, see `FullForm[ ]`.

5.3.6 **ResetFactors**

<code>ResetFactors[opts]</code>	resets options in <code>Options[currentAggregator]</code> which pertain to the factors
---------------------------------	--

`Options[ResetFactors]` give control over the `NumberOfFactors` and `Clearing`.

Adjusting the number of factors is a transparant operation. The options `Factors`, `FactorCoefficients` and `FactorPrices` have to be set to the length given by `NumberOfFactors`. Canonic inner terms are for example `FactorX` and `FactorP`. Similarly, additional parameters are set, as defined by `ResetFactors`  $\rightarrow$  *val*.



Clearing is a bit more complex. Let us suppose that you find the canonic terms like FactorX and FactorP less didactic. You then set e.g.  $x = \text{FactorX}$  and  $w = \text{FactorP}$ , or  $\text{Labour} = \text{FactorX}[1]$  and  $\text{Wage} = \text{FactorP}[1]$  etcetera. Formulas may become easier to read in this manner, while the routines still operate (should operate) as before. ResetFactors allows you to undo these name assignments, or to keep them while the number of factors changes.

```
Clear[x, p, b]
FactorX = x;
FactorP = p;
FactorC = b;

Profit[]

Aggregator() - p(1) x(1) - p(2) x(2)

ResetFactors[NumberOfFactors -> 3, Clear -> False]

{Aggregate -> 1, AggregatePrice -> 1, Aggregator -> Aggregator, Constant -> Scale,
  Cost -> Cost, DemandFunction -> Cost, Dual -> Cost, DualQ -> Null, Factor -> x(1),
  FactorPosition -> Automatic, FactorPrices -> {p(1), p(2), p(3)}, Factors -> {x(1), x(2), x(3)},
  NumberOfFactors -> 3, ResetFactors -> (FactorCoefficients FactorC ), Return -> 1}
```

The options `Aggregator -> ...` and `ResetFactors -> ...` are taken from `Options[Aggregator]` (only).

<i>option</i>	<i>typical default value</i>	
NumberOfFactors	Automatic	the setting is not changed. If the setting is an integer, the number of factors is adjusted to that value.
Clear	Automatic	If Clear -> True, then the Factor terms are Cleared and thus get their canonic value. If Clear -> False, then they keep their value. If Clear -> Automatic (default), and NumberOfFactors -> Automatic (default), then you are prompted whether you want to Clear Factors.

If `NumberOfFactors -> number`, then you will not be prompted, and then, if `Clear -> Automatic`, the Factors will not be Cleared.

A subroutine is:

<code>ResetFactorsExpand[n]</code>	takes <code>ResetFactors -&gt; val</code> of <code>Options[Aggregator]</code> , and $val = \{\{xi, yi\}, \dots\}$ becomes $\{xi \rightarrow \text{Array}[yi, n], \dots\}$
------------------------------------	---

5.3.7 Other factor control

Some of these are used in higher level packages, like `LevelCES``.

FactorPosition[ { <i>opts</i> }]	gives the position of Factor within Factors
FactorSettings	option for {FactorX[i]→ valuei, ... }
FactorLabels	option to indicate all FactorX[i] labels, also those which are perhaps not explicitly used when there are levels.
Levels	use option Levels → {n1, n2, ..., n} when level i has ni factors.

5.3.8 Factor demand

FactorDemand[ ] again is a shell. It provides a general format that only directs towards routines that have to be defined by the user. For example in the CES` package, the FactorDemand[CES, Price] has been defined. By setting SetOptions[Aggregator, Aggregator → CES] this particular demand function becomes accessible as FactorDemand[Price, opts].

FactorDemand[ <i>case</i> , <i>opts</i> ]	gives the demand for a factor as factor → <i>value</i> , under <i>case</i>
FactorDemand[ <i>opts</i> ]	use the current aggregator, and use the setting of DemandFunction → ...
FactorDemand[ ( <i>f</i> ,) Function, <i>opts</i> ]	use only the functional relationship to select a point on the efficiency frontier
FactorDemand[ ( <i>f</i> ,) Price, <i>opts</i> ]	use $p D[f, x_i] == p_i$ for <i>i</i> only to solve for <i>x<sub>i</sub></i>
FactorDemand[ ( <i>f</i> ,) Cost, <i>opts</i> ]	use cost minimisation
FactorDemand[ ( <i>f</i> ,) Profit, <i>opts</i> ]	use profit maximisation

Note: The factor demanded is selected by the FactorPosition. If DualQ → True then Roy's Rule is applied.

A subroutine is:

DualSelector[{ <i>opts</i> }]	selects known combinations of demand functions and Dual type from <i>opts</i> , updating the latter. Output is {DemandFunction type, Dual type, <i>optsnew</i> }. Default is Cost for all.
-------------------------------	--

5.3.9 Marginal values

MarginalCost[( <i>f</i> ,) <i>opts</i> ]	gives $D[\text{Cost}[\text{Factor} \rightarrow x], x]$ for aggregator <i>f</i>
MarginalProduct[( <i>f</i> ,) <i>opts</i> ]	gives $D[f[x], x]$ for aggregator <i>f</i>

### 5.3.10 Elasticities

The following take the variable of relevance from the setting of Factor → ....

<code>PriceElasticity[opts]</code>	gives the own price elasticity of Factor, i.e. partial derivative $d\log[\text{DemandFunction}[ ]] / d\log[\text{FactorP}[i]]$
<code>CostElasticity[opts]</code>	gives the cost elasticity of Factor in the <code>FactorDemand[Cost, opts]</code> . This would be the most general statement that can be used to determine the income elasticity.
<code>ES[f, xi, xj]</code> <code>ES[g[... , xi, ... , xj, ...], xi, xj]</code>	gives the direct Elasticity of Substitution of $f[x,y]$ or $g[...]$ evaluated at $x_i$ and $x_j$ (McFadden (1963)). for 2 factor functions, the ES is equal to the AES, but in general not.
<code>AES[opts]</code>	gives the Allen Elasticity of Substitution using $AES[i, j] = d\log[x_i] / d\log[p_j] / CS_j$ where $CS$ is the minimal cost share. Default option for the derivative selection is $D \rightarrow \text{FactorP}[2]$
<code>AUES[opts]</code>	gives the Allen Elasticity of Substitution using Uzawa 1962 $AES[i, j] = C * C_{ij} / (C_i C_j)$ where $C$ is minimal cost and $C_i$ the derivative on price $i$ . Default option for the derivative selection is $D \rightarrow \{\text{FactorP}[1], \text{FactorP}[2]\}$

### 5.3.11 Other properties

<code>Properties[opts]</code>	gives some properties of the current aggregator using explicit differentiation. Output is {Return → returns to scale, Share → list of cost shares, PriceElasticity → list of own prices elasticities}
-------------------------------	---

A point of interest is the `DRule` routine discussed in 3.7, that helps to substitute  $y$  back into  $\partial y / \partial x$ . Let  $y$  be an aggregator with scale parameter  $c$  and returns to scale  $r$ . Then normalised production  $y^*$  is useful for duality:

$$y^* = \left( \frac{y}{c} \right)^{\frac{1}{r}}$$

Since the (utility) optimum is conditional to a budget condition  $Z = p_x \cdot x$  (and the budget itself is homogeneous of degree zero), the optimum is homogeneous of degree zero, and a normalised problem may simplify expressions.

The first order conditions give us  $\partial y/\partial x$ , and since  $y$  has a power, we may get simplified expressions by back-substitution of  $y$  into its derivative. And  $y^*$  may be even simpler.

5.3.12 SetFunction

The following shell is used intensely. The usefulness of SetFunction is that it offers key information about a function to other processing routines.

```
SetFunction[(f,) opts]    should give: {Function → f[g[opts]], CoefficientList → h[opts],  
                                Factors → ...}, where g and h are functions dependent upon f.
```

An application of SetFunction is in the AGE` package. Applied general equilibrium analysis should be independent from the functions chosen. Indeed, if the user provides the information about a function as structured by SetFunction, then the AGE` package can run that function.

If no  $f$  is specified, then the Economic`Common` package redirects the call, by using the Options[Aggregator], to SetFunctions[current aggregator, opts]. Presumably, the user has defined this expression for the current aggregator. This for example has been done for the CES function.

5.3.13 Symbols only

The following symbols have no function attached to them. As symbols they still are useful. For example, Demand[Real] and Demand[Nominal] can indicate different variables.

\$Cost	Demand	Premium	Sector
\$Price	Intermediate	Price	Supply
\$Profit	Labour	Production	Tax
\$Quantity	LabourProductivity	Quantity	UnitCost
Capacity	Loss	Rate	Utilisation
Capital	MarkUp	RateOfInterest	Utility
Consumption	Nominal	Resources	ValueAdded
Debt	NumberOfSectors	Revenue	Wealth
Deficit	Optimal	RTS	

- The following are possible relationships. These are not defined internally, and may e.g. be imposed by the user, where he or she should take care for real and nominal values.

```
Revenue == Price Quantity  
  
Quantity == Min[Demand, Supply]  
  
Revenue == (1 + MarkUp) Cost
```

```
UnitCost == Cost / Production

ValueAdded == Production - Intermediate

LabourProductivity == ValueAdded / Labour
```

5.3.14 Inventory Model

The routine SolveFrom is explained in section A.6.5. As remarked there, one of its advantages is that symbols need not be defined as functions. The following InventoryModel is a SolveFrom application with some simple economic relationships on inventories. Note: Production here is seen as Real Value Added.

InventoryModel[opts] is a SolveFrom application, using, in real values:  
Demand + DInventory == Production,  
DInventory == AverageInventory - OldInventory,  
Production == LabourProductivity Labour,  
Turnover == Demand / AverageInventory,  
NPeriodsOfSupply == AverageInventory / Demand.

```
sol = InventoryModel[
  OldInventory → 100,
  LabourProductivity → 1,
  Labour → 1000,
  AverageInventory → 10. Sqrt[Demand]];
```

```
SolveShow[Last[sol]]
```

	1
Demand	814.59
DInventory	185.41
NPeriodsOfSupply	0.350373
Production	1000.
Turnover	2.8541

To use SolveFrom, symbols need not have functional definitions. However, they still may have them. Some of above symbols indeed have definitions attached to them.

Inventory	Symbol for the inventory level
AverageInventory	AverageInventory[q, bi:0] for basic inventory bi (safety, pipeline) and q/2 the average variable inventory
Turnover[d, a, p:l]	for total demand d, average inventory a and period p. Note that turnover is an 'average' concept
NPeriodsOfSupply[d, i, p:l]	for total demand d over the period p, and inventory i. If i is the average inventory over the period (i = a), then this gives the average number of periods of supply
DInventory	Symbol for the change in inventories (s – s(–1))
OldInventory	Symbol for s(–1)

5.3.15 Economic`Optimise`

The following small utility package applies the Lagrange` package to the Economic`Common` environment. It is not fully tested.

```
Economics[Economic`Optimise]
```

MinimiseCost[opts]	uses the Lagrange method to minimise px.x subject to Aggregate = f[x] for the current aggregator f
--------------------	--

Default Print → True prints the optimisation problem. Lagrange options are from Options[Lagrange] or opts. HoldShell is applied to the call of f, also applying opts with defaults ReEvaluate → False and SetResults → True.

5.3.16 Economic`Graphics`

This package gives some graphics routines. First load it:

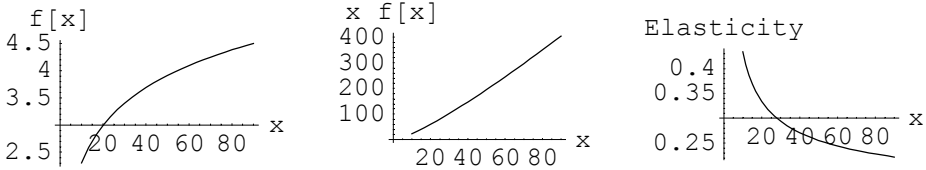
```
Economics[Economic`Graphics]
```

If f[p] is a demand function with p the price, then p f[p] gives revenue. If f[p] = A p<sup>-a</sup> then a is the price elasticity, and revenue will rise when a+1 > 0. These relations are clarified by the following plot, in this case for f[p] = Log[p].

```
ElasticityPlot[f[x], {x, xmin, xmax}, opts]
```

plots  $f[x]$ ,  $x f[x]$  and  $\text{Elasticity}[f[x], x]$  for the given domain

```
ElasticityPlot[Log[x], {x, 10, 90},  
  TextStyle → {FontSize → 10}];
```



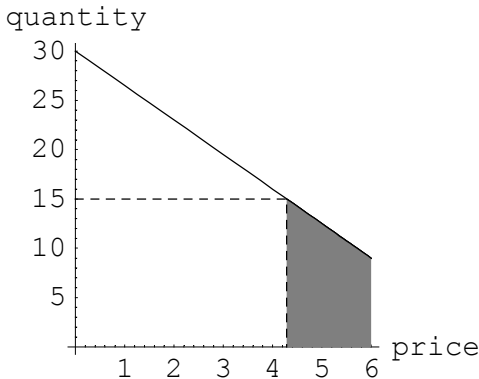
The consumers' surplus is the revenue that (some) consumers would be willing to pay, but that they don't need to pay since the current market price is lower. A producer can try to exploit the consumers' surplus by product differentiation. The latter creates different demand functions.

```
ConsumerSurplusFilledPlot[f, pstar, {pmin, pmax}, opts]
```

plots the demand function  $f[p]$  for the price domain, while the consumers' surplus from  $pstar$  is coloured. If  $pstar = \text{Automatic}$ , then  $pstar$  is computed as the value that maximises revenue  $p f[p]$ . See `Results[ConsumerSurplusFilledPlot]`

```
lin[p_] = 30 - 3.5 p;
```

```
ConsumerSurplusFilledPlot[lin, Automatic, {0, 6}, PlotRange → All,  
  AspectRatio → 1, TextStyle → {FontSize → 11}];
```

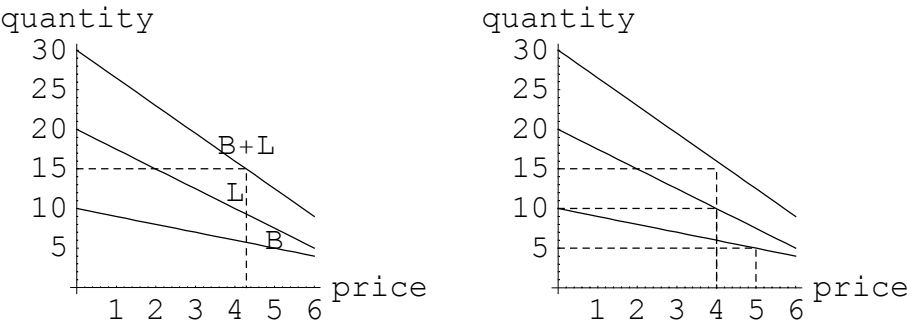


```
ConsumerSurplusPlot[{f, g}, points_List: {}, {pmin, pmax}, labels_List: {}, opts]
```

divides the consolidated demand function  $(f+g)[p]$  in sub demand functions  $f[u]$  and  $g[v]$  so that the consumer surplus can be exploited.

Points = {u, v, p} can be provided, and if points = {} then the price points are determined by revenue maximisation, i.e. by maximising  $p * (f+g)[p]$  and  $u * f[u]$  and  $v * g[v]$  respectively. List {pmin, pmax} is a pair of numbers that provide the beginning and end of the price plotting domain. Labels is a pair of strings that provide identifications of the functions in the plot. Results are in Results[ConsumerSurplusPlot], with Join → ... giving the results for the consolidated demand function and Apart → ... giving the results for the nonconsolidated case. Data are in the order u, v, p.

```
lin1[p_] = 10 - 1 p; lin2[p_] = lin[p] - lin1[p];  
  
ConsumerSurplusPlot[{lin1, lin2}, {}, {0, 6}, {"B", "L"}, AspectRatio → 1];
```





## 5.4 The Constant Elasticity of Substitution function

---

### 5.4.1 Summary

The Constant Elasticity of Substitution (CES) function is often used in utility and production analysis. This package provides a canonical form, and defines the demand functions and dual function in terms of this. Also provided are various utilities such as for the plotting of the contours.

**Economics [CES]**

### 5.4.2 Introduction

The Constant Elasticity of Substitution (CES) function of Arrow, Chenery, Minhas & Solow 1961 is frequently used in textbook analysis of production and utility. It unites various functional forms, while it captures an important economic process, that of substitution, within a single parameter. The CES also has the advantage that its dual is a CES again.

It must be noted though that the CES is primarily useful for those conceptual reasons only. Beyond the textbooks, the conceptual advantages are also drawbacks. The traditional CES is homothetic, e.g. the capital-labour ratio is independent of the level of output and is dependent upon the price ratio only. Alternatively put, the expansion paths are straight lines through the origin, and the contours are radial blow-ups of one-another. Also, the CES is additive, with the consequence of an approximate linear relationship between income and (own) price elasticities.

Barten 1977 writes: "The proportionality of the direct price elasticity and the income elasticity of a good is too restrictive to the taste of an empirical demand analyst. Indeed, under additive or strongly separable preferences, the cross specific substitution effects are, by definition, zero while, empirically, the general substitution effect together with the income effect is negligible. One is left with only the direct substitution effect. The structure (...) imposed on the interactions between demand for commodities is so restrictive that it frustrates one of the main purposes of the construction of empirical demand systems, namely a coherent and empirically valid measurement of those interactions. By suppressing the interactions, one remains coherent, but the empirical validity might be reduced to an unacceptably low level." (Barten, "The systems of consumer demand functions approach: A review", *Econometrica* 1977, p23-51.)

Various alternatives to the CES have been proposed. See, especially on heterothecity: Sato, "The most general class of CES functions", *Econometrica* 1975, p999-1003; Sato, "Homothetic and Non-Homothetic CES production functions", *American Economic Review*, September 1977, p559-569; and Nadiri, "Producers theory", in Arrow & Intrilligator, "Handbook of mathematical economics", Vol II, North Holland 1982. A modern treatment of duality (and with some special attention to the CES) is given by: Blackorby, Primont, Russell, "Duality, Separability & Functional Structure: Theory and Economic

Application", North Holland 1978; and Diewert, "Duality approaches to microeconomic theory", in Arrow & Intriligator, "Handbook of mathematical economics", Vol II, North Holland 1982.

Below we don't stray far from the traditional CES. We only regard the n-level-CES, a shifted-CES, and the Indirect Addilog Demand System (IADS). We develop those functions fully, rather than developing a general function and collapsing this for certain parameter values. The major reason is that it appeared that certain limits were not found by *Mathematica*. So it turned out to be better to first solve the limit, or state the specific functional form, and then repeat the derivation of properties. And thus, while we started out with wondering whether it would be possible to develop a Standard Model, and decided that it would be better to start with a simpler single function, we now find that this function already causes many problems. The exercise however is useful for textbook analysis and conceptual reference, while it also shows how functions can be implemented with the `Economic`Common`` package.

### 5.4.3 Canonical and normal form

The **Canonical Form** of the Constant Elasticity of Substitution function is:

$$y = A (c \text{ labour}^{-\rho} + (1 - c) \text{ capital}^{-\rho})^{-\frac{\nu}{\rho}}$$

where  $\nu$  is the returns to scale parameter (normally  $\nu = 1$ ), and where  $-1 \leq \rho < \infty$ . The elasticity of substitution  $0 \leq \sigma < \infty$  can be derived to be:

$$\sigma = \frac{1}{1 + \rho}$$

Limiting functions are:

- When  $\rho \rightarrow -1$  then  $\sigma \rightarrow \infty$ , giving a straight line
- when  $\rho = 0$  then  $\sigma = 1$ , giving the Cobb-Douglas function
- when  $\rho \rightarrow \infty$  then  $\sigma \rightarrow 0$ , giving the Leontief function.

The canonical form is used a lot in the literature, likely since mathematical manipulations are more economical in terms of reading and writing. However, in *Mathematica*, the computer does most of the work, and hence we rather use the following **Normal Form** that directly addresses the elasticity. Use  $1/\sigma = 1 + \rho$ , or  $\rho = 1/\sigma - 1$ , and find:

$$y = A (c \text{ labour}^{1 - \frac{1}{\sigma}} + (1 - c) \text{ capital}^{1 - \frac{1}{\sigma}})^{\frac{\nu}{1 - \frac{1}{\sigma}}}$$

For more than two dimensions the normalised production  $y^*$  and the **CESterm** for vectorial  $c$  and  $x$  are useful:

$$y^* = \left( \frac{Y}{A} \right)^{\frac{1}{\nu}}$$

$$\text{CESterm} = c \cdot x^{1 - \frac{1}{\sigma}} = (y^*)^{1 - \frac{1}{\sigma}}$$

$$CES = A \text{ CESterm}^{\frac{\nu}{1-1/\sigma}}$$

Note: The discussion may adopt the terminology of production, but the results can be translated easily to utility.

Note: One may set  $\sigma$  at a negative value, to create concave contours. This kind of function can be handy, though it will generally not be a utility or production function.

Note: The "constant" returns to scale or CRTS is taken as  $\nu = 1$ . If  $\nu < 1$  then there are decreasing returns to scale, and if  $\nu > 1$  then there are increasing returns to scale. Note that "proportional" returns to scale would be a better term, since a value  $\nu \neq 1$  could still be *constant*. Variable returns to scale would have the  $\nu$  dependent upon the inputs.

5.4.4 The CES function

The CES function has been implemented with two formats, both a fixed format and a option input format. The latter has the advantage of free floating input and named parameters.

<code>CES [A, c, x, S, v: I]</code>	gives the function with scale factor A, factors x, their coefficients c (usefully summing to 1), the elasticity S, and the returns to scale parameter v.
<code>CES [opts]</code>	calls the Constant Elasticity of Substitution function using default options as set in Options[CES].

■ The Leontief function

`CES[A, {c1, c2}, {x1, x2}, 0]`

$$A \text{ Min}\left[\frac{x1}{c1}, \frac{x2}{c2}\right]$$

■ The Cobb-Douglas function

`CES[A, {c1, c2}, {x1, x2}, 1]`

$$A x1^{c1} x2^{c2}$$

■ Line: infinite substitutionability

`CES[A, {c1, c2}, {x1, x2}, Infinity]`

$$A (c1 x1 + c2 x2)$$

- The full CES function

```
CES[A, {c1, c2}, {x1, x2}, S, v]
```

$$A \left( c1 x1^{1-\frac{1}{S}} + c2 x2^{1-\frac{1}{S}} \right)^{\frac{v}{1-\frac{1}{S}}}$$

Subroutines are:

CESlimitQ[S]	evaluates MemberQ[{0,0.,1,1.,Infinity}, S]. Is called by CES[ ]. When CESlimitQ[S] then the function takes a special form (Leontief, CD or Line). The call of the function also creates the CESTerm which excludes scale and powers.
CESTerm	CESTerm is the inner term, i.e. (Aggregate / A)^(1/v). CESTerm is defined within the call of the CES[] function, and keeps its current value until the next call.
CESTermRuleI[A, s, v:1] or CESTermRule[opts]	gives the rule CESTerm → function[Aggregate] for the relevant form. The latest CESTerm is used, i.e. of the latest CES[ ] call. This means that there need not be a relation with the parameters given in the present call of CESTermRule
CheckES[S]	checks on the Sign of S (nonnegative value expected) and substitutes 0. → 0 and 1. → 1

5.4.5 Flexible input format

When you use the flexible input format, be aware that CES is for the function and Ces → S is for the parameter assignment.

S	default value in Ces → S in Options[CES], denoting the elasticity of substitution between the input factors
Ces	option for the constant elasticity of substitution in the Options[CES]
Options\$CES	makes options available for a ResetOptions[CES]; the current settings of Options[Aggregator] are taken; so perhaps you wish to ResetOptions[Aggregator] first

- For example:

```
CES[Constant → A[NewYork], FactorCoefficients → {.3, .7},  
Ces → 1/2, Return → v]
```

$$A(\text{NewYork}) \left( \frac{0.7}{\text{FactorX}(2)} + \frac{0.3}{\text{FactorX}(1)} \right)^{-v}$$

When the *Ces* parameter has a numerical value, then the *Mathematica* evaluations may render awkward results which take long to compute. Therefore, the default value is symbolic *S*. To speed the evaluation, the default solution method is `Method → SolveFormally`, that first replaces numbers for symbols, then `Solves`, then substitutes back.

#### 5.4.6 Check on the elasticity

The CES package puts the option `Aggregator → CES` in the `Options[Aggregator]`. As a result, you can use `FactorDemand`, `Cost` and `Profit` and other functions without specifying that you use the CES.

```
FactorX = x; FactorP = p; FactorC = c;
```

```
AES[Dual → Aggregate]
```

$$S c(2)^S \left( \frac{c(2)}{p(2)} \right)^{-S} p(2)^{-S} (c(1)^S p(1)^{1-S} + c(2)^S p(2)^{1-S})^{-\frac{S}{1-S} + \frac{1}{1-S} - 1}$$

```
Simplify[PowerExpand[%]]
```

*S*

#### 5.4.7 Graphics

```
CESContours[levels_List, width, {cesopts1}, ..., opts]
```

displays contours with display opts. It maps CESContours onto the cesopts

```
CESContours[levels_List, width, cesopts]
```

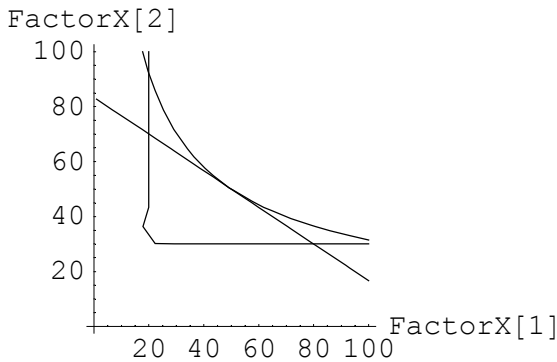
calls `ContourPlot` with `Contours → levels`,  
for `{xi, 1, width}`, where `width` must be a number. `Cesopts`  
are only relevant for the CES function. Display is suppressed

For both, the `Factors → {x1, ...}` must be set such that only two elements are Symbols and all other are numbers.

Let us plot the contours for various values of the elasticity of substitution (0, 1, Infinity), at level 50. We thus plot those combinations of `{x1, x2}` such that `50 = CES[... {x1, x2}...]`. The `Ces → 0` Leontief function is a bit ragged, but the idea of a hook survives.

```
SetOptions[CES, Constant → 1, FactorCoefficients → {.4, .6}];

CESContours[{50}, 100,
  {Ces → 0}, {Ces → 1}, {Ces → Infinity},
  AxesOrigin → {0,0}, TextStyle → {FontSize → 11}];
```



Note the location of the contours. Properly speaking, the Leontief case is not quite the limit for the present CES when  $S \rightarrow 0$ . The proper limit is  $\text{Min}[x_1, x_2, \dots]$  without the  $c$ -weights. The  $c$ 's however are included here, since they provide useful parameters for dimensional adjustment, for example for changing the factor inputs into efficiency units. If you run the same contourplot but with  $\{Ces \rightarrow 0.1\}$ ,  $\{Ces \rightarrow 1\}$ ,  $\{Ces \rightarrow \text{Infinity}\}$ , then you will see that the contours touch in one point.

5.4.8 Utilities

Back-substitution has been discussed in 3.7.4.

<code>DCES[opts___Rule]</code>	takes defaults from Options[CES], takes Factor $\rightarrow$ xi, differentiates the CES function, and substitutes Aggregate == CES[...] back into the result
<code>DCES[Hold[CES[A, c, x, S, v:l]], xi]</code>	
	an alternative input format for the latter

The following is useful for the AGE` package, where there are different functions per sector.

<code>SetOptionsCESSector[i]</code>	performs a <code>SetOptions</code> for Sector <code>i</code> .
<code>SetOptionsCESSector[]</code>	gives options for the aggregate, thus with <code>S[]</code> , <code>Scale[]</code> , etc.
<code>SetOptionsCESSector[None]</code>	returns pure symbols <code>S</code> , <code>Scale</code> , etc.

Note that you cannot use a pure symbol `S` for the aggregate and a functional form `S[i]` for a sector. For, for example, when `S` gets a value, `S = 100`, then you get `100[i]` ...

### 5.4.9 Example notebooks

The example notebooks deal with particular angles:

- `CESUserManual.nb` gives a short user manual. This notebook shows the problems that arise with taking the limits, while also the proper limit to the Leontief case is discussed. It is demonstrated how one can use the `CEStermRule`. One is reminded to use `ResetFactors` of the `Economic`Common`` package to adjust the number of factors. There is also a short review of the factor demand, and minimal cost and maximal profit solutions, as an application of the similar discussion for the `Economic`Common`` package.
- `Graphics.nb` provides more graphical examples.
- `CESdemand.nb` shows the different factor demand functions, under the different assumptions of efficiency, single price optimality, minimal cost and maximal profit.
- `CEScost.nb` derives the cost relations.
- `CESprofit.nb` derives the profit relations.
- `dCESdx.nb` shows how to substitute the aggregate back into marginal productivity.

# 5.5 CES-like functions

## 5.5.1 Summary

We here discuss the shifted CES, the n-level CES and the Indirect Addilog Demand System (IADS). Only general points are treated, and details can be found in the exemplary notebooks ShiftCES.nb, LevelCES.nb and IADS.nb.

## 5.5.2 The shifted CES

Economics[ShiftCES]

### 5.5.2.1 Definition

With the CES as  $y = f[x; A, c, \sigma]$  with factors  $x$ , constant  $A$ , unit weight coefficients  $c$ , and substitution elasticity  $\sigma$ , we get a function where the factors are shifted, as  $y = f[x - c y; A, c, \sigma]$ .

While the CES allows boundary values  $x \rightarrow 0$ , except in the Leontief case, the shifted CES is as strict as the Leontief case, since the boundaries are  $x \rightarrow c y$ .

Note that this is an implicit function, since  $y$  occurs both left and right of the equality sign. This function is no longer a proper CES. We may study it since it helps us to understand the limit properties of the CES, i.e. that the CES turns into a Line, Leontief or Cobb-Douglas function.

ShiftCES[opts]	shifts the function, so that it actually is no CES[]
NShiftCES[Set, opts]	sets parameters at a value and calls ShiftCES[], so that later computations are faster
NShiftCES[opts]	computes the aggregate value of the implicit function by using FindRoot

- As much as possible is taken from the Options[CES]:

```
SetOptions[CES, Constant → A, Return → v, Aggregate → y];
FactorX[1] = Labour; FactorX[2] = Capital;
FactorC[1] = c; FactorC[2] = 1 - c;
```

- Note that the call generates an *equation*:

ShiftCES[]

$$y == A ((1 - c) (Capital - (1 - c) y)^{1-\frac{1}{\sigma}} + c (Labour - c y)^{1-\frac{1}{\sigma}})^{\frac{\sigma}{1-\sigma}}$$



- The special cases of Line, Leontief and Cobb-Douglas have not been developed. Limit calls thus are inaccurate.

**ShiftCES[Ces → 1]**

*ShiftCES::limit: Warning: limit values taken for CES and not ShiftCES*

$$y == A ((\text{Capital} - (1 - c)y)^{1-c} (\text{Labour} - cy)^c)^v$$

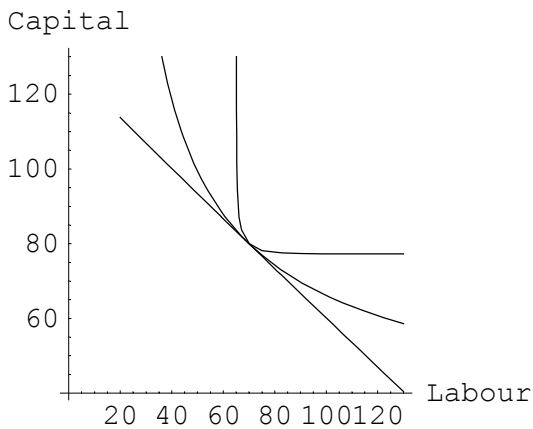
### 5.5.2.2 Graphics

The contour plot is similar to the CES. We take approximate values around 0, 1 and  $\infty$ , since the limits have not been developed.

**ShiftCESContours[]**      See the input format of CESContours[]

```
SetOptions[CES, Constant → 1, Return → 1,
  FactorCoefficients → {0.4, 0.6}];

ShiftCESContours[{50}, 130,
  {Ces → 0.1}, {Ces → 1.1}, {Ces → 99},
  AxesOrigin → {0, 40}, TextStyle → {FontSize → 11}];
```



### 5.5.2.3 Numerical

The following is an example of how one can extract the implicit value of  $y$ .

```
NShiftCES[Set, Constant → 1, Return → 1, Ces → 2,
  FactorCoefficients → {0.4, 0.6}]

f[x1_, x2_] := NShiftCES[Factors → {x1, x2}]
```

`f[4, 5]`

3.02425

The following subroutine is used:

<code>ShiftCESrhs</code>	Label only. <code>FunctionDomainQ[ShiftCESrhs, opts]</code> or <code>FunctionDomainQ[ShiftCESrhs, factors_List, aggregate, factorcoefficients]</code> test on the domain of the right hand side of the ShiftCES, i.e. whether <code>factors – aggregate * factorcoefficients</code> is nonnegative. Defaults are taken from <code>Options[CES]</code> , that also can be set with <code>NShiftCES[Set, opts]</code>
--------------------------	---

5.5.3 The level CES

Because of the discussion before, we first reset the options. Some options must be symbolic too, since they can get different formats and values at each level.

```
ResetOptions[CES]; SetOptions[CES, Return → RTS];

Economics[LevelCES]
```

5.5.3.1 Definition

A level-C.E.S. function arises when factors within a CES format are CES formats themselves.

<code>LevelCES[opts]</code>	analyse the structure given by option <code>LevelCES → structure</code> , define the required <code>Options[LevelCES]</code> , and give the function
<code>Options\$LevelCES</code>	makes options available for a <code>ResetOptions[LevelCES]</code>

`LevelCES[ ]` relies on `CES[ ]` and `Options[CES]`. Options like `Factors → ...` etc. are used to reset the `Options[CES]`. Default values for `Constant`, `Ces` and `Return` for the separate levels are taken from `Options[CES]`. The `LevelCES` stores the `LevelCES` parameter. If input was `LevelCES → {}`, then a call of `LevelCES[ ]` stops without resetting the Options. Use `SetFunction[LevelCES, ...]` for more control of the naming of variables and parameters.

5.5.3.2 Example

Let us define a structure. A very simple example is a function `h` with two factors, `x[1]` and `x[2]`.

```
cs = h[x[1], x[2]];
LevelCES[LevelCES → cs]

((1 - c) Capital1 -  $\frac{1}{\sigma}$  + c Labour1 -  $\frac{1}{\sigma}$ ) $\frac{RTS()}{1 - \frac{1}{\sigma}}$  Scale()
```

Now, what if  $x[1]$  and  $x[2]$  are functions again ? Let us use, for your convenience,  $x, y, z$  for the different levels. The only condition is that all symbols must be undefined. We also keep using  $h$  as an arbitrary head.

```

x1 = h[y1, y2];
x2 = h[y3, y4];
y1 = h[z1, z2, z3, z4];

cs = h[x1, x2]

h(h(h(z1, z2, z3, z4), y2), h(y3, y4))

LevelCES[LevelCES → cs]

```

The latter call generates a large expression, and is therefore not printed.

LevelCES[ ] exploits *Mathematica*'s administrative routines on levels and positions, and thereby allows a flexible format for creating quite complicated level structures. For example, you can use Array[ ] statements to make larger lists of symbols, like f[f[Array[b, 10], ... ]]. Or for diagnostics you can use LevelInspect[cs] and PartInspect[cs] on the above.

The routine also defines the relevant options for factors and parameters. The naming of factors and parameters conforms with *Mathematica*'s convention on positions in the given structure. At the same time, Factors → ... and FactorPrices → ... options remain flat lists, and thus can be used by the common package. Please note too, that parameters (like  $S$ ) now are functions of the level (like  $S[ ]$ ,  $S[2,2]$ ). When you provide values for  $S[...]$  beforehand, the internal call to the CES[ ] routine will be able to mix Leontief, CD, Line and General CES functions.

### 5.5.3.3 Economic`Common`

This implementation of the Level-CES uses the Economic`Common` environment. A simple result is the cost accounting:

```

Cost[LevelCES]

{Cost(1, 1) → FactorP(1, 1, 1) FactorX(1, 1, 1) + FactorP(1, 1, 2) FactorX(1, 1, 2) +
  FactorP(1, 1, 3) FactorX(1, 1, 3) + FactorP(1, 1, 4) FactorX(1, 1, 4),
  Capital → FactorP(2, 1) FactorX(2, 1) + FactorP(2, 2) FactorX(2, 2),
  Labour → Cost(1, 1) + FactorP(1, 2) FactorX(1, 2), Cost → Labour FactorP(1) + Capital FactorP(2)}

```

Only Efficiency and Cost demandfunctions have been implemented. The default demand function is with cost minimisation, as we can check and then determine for a factor:

```

DemandFunction /. Options[LevelCES]

Cost

```

```
FactorDemand[LevelCES, Aggregate → y,  
             Factor → FactorX[1, 1, 3]]  
  
ReEvaluate::rec :  
  ReEvaluate -> True may be slow for recursive calls from FactorDemand  
  
LevelCESAnalyser::within :  
  Warning: the maximum over all NumberOfFactors  
    of the sublevels is found to be 4, which is  
    greater than the present setting 2 in Options[CES]  
  
FactorX(1, 1, 3) →  $\left( \text{Cost}(1, 1) \left( \frac{\text{FactorC}(1, 1, 3)}{\text{FactorP}(1, 1, 3)} \right)^{S(1,1)} \right) /$   
   $(\text{FactorC}(1, 1, 1)^{S(1,1)} \text{FactorP}(1, 1, 1)^{1-S(1,1)} + \text{FactorC}(1, 1, 2)^{S(1,1)} \text{FactorP}(1, 1, 2)^{1-S(1,1)} +$   
   $\text{FactorC}(1, 1, 3)^{S(1,1)} \text{FactorP}(1, 1, 3)^{1-S(1,1)} + \text{FactorC}(1, 1, 4)^{S(1,1)} \text{FactorP}(1, 1, 4)^{1-S(1,1)})$ 
```

The warning on the number of factors is only a warning message. LevelCES[ ] makes equal lists of factors, coefficients, etc. and therefor is quite protected against snitches in the Options[CES].

5.5.3.4 Utilities

Four different utilities are:

LevelCESAnalyser[opts]	analyses the structure given by LevelCES → s, defines required Options[LevelCES], and puts the output on hold: Hold[CES][result]. (LevelCES only calls ReleaseHold.) Within Hold, Position gives the level.
LevelCESRule	develops FactorSettings, and thus gives a set of rules that are basically independent of Options[CES]. See SimplifyBy.
LevelCESlevel[opts]	gives the level within the LevelCES associated with Factor. The Factor option may also be set to implicit level values.
SubCESNoFactors[n]	sets the maximum number of factors in the CES used in sublevels to n

The utility SubCESNoFactors[n] comes in use when you wish to use CES options from the Options[CES]. To adjust the number of factors it is not enough to use SetOptions, since the number of factors has influence on other parameters. SubCESNoFactors adjust such other parameters too.

The LevelCES.nb exemplary notebook also contains an example of how you can evaluate LevelCES[ ] in a HoldShell, so that you can store results and thereafter prevent the reevaluation of the same expression.

5.5.4 Indirect Addilog Demand System (IADS)

Economics [IADS]

Note that the Almost Ideal Demand System (AIDS) has not implemented.

### 5.5.4.1 Definition

A well-known and flexible *dual* aggregation function is the Indirect Addilog Demand System, also called the Expenditure Allocation Model. See Somermeyer & Langhout, "Shapes of Engel curves and demand curves: implications of the expenditure allocation model, applied to Dutch data," European Economic Review 1972, pp351-386. Here we define the IADS, derive factor demand and cost and price elasticities, and show how it can collapse into the (dual) CES for certain parameter values.

<code>IADS [opts]</code>	calls the Indirect Addilog Demand System. If all FactorPowers are equal, this reduces to the CES.
<code>Options\$IADS</code>	makes options available for a <code>ResetOptions[IADS]</code>

Note: `IADS[ ]` uses relative prices  $p(i) / \text{cost}$ , so that it is applicable both for consumption and production. When the IADS is used for production then there is a restriction on the parameters, see `IADS.nb` and Thomas Cool, "On the link between consumption and production duality: using the IADS for production", Central Planning Bureau, The Hague, Internal Memo III-6, July 3 1986.

- The following settings give a useful example:

```
ResetOptions[Aggregator];
SetOptions[Aggregator, Return → v];
ResetFactors[Clear → True, NumberOfFactors → 3];
ResetOptions[IADS];

FactorX = x; FactorP = p; FactorC = b; FactorE = e;

IADS[]
```

$$\text{Scale} \left( \left( \frac{p(1)}{\text{Cost}} \right)^{e(1)} b(1)^{1-e(1)} + b(2)^{1-e(2)} \left( \frac{p(2)}{\text{Cost}} \right)^{e(2)} + b(3)^{1-e(3)} \left( \frac{p(3)}{\text{Cost}} \right)^{e(3)} \right)^{\wedge} \left( -\frac{3v}{e(1)+e(2)+e(3)} \right)$$

<code>IADSPartRule</code>	<code>IADSPartRule</code> gives a rule for simplification, using the <code>IADSPart</code> . See <code>SimplifyBy</code> .
<code>IADSPart</code>	denotes an additive part within the IADS
<code>IADSpriceRule</code>	gives a rule for the prices, using the <code>IADSPart</code>

- A simplified presentation can be found by:

```
IADSPart = Q;
qs = Table[IADSPart[i], {i, NumberOfFactors /. Options[IADS]}];

IADS[] /. IADSPartRule
```

$$\text{Scale} \left( \frac{Q(1)}{e(1)} + \frac{Q(2)}{e(2)} + \frac{Q(3)}{e(3)} \right)^{-\frac{3v}{e(1)+e(2)+e(3)}}$$

- Note that reverse substitution is possible by:

```
qback = Solve[IADSPartRule /. Rule -> Equal, qs]
```

$$\left\{ \left\{ Q(1) \rightarrow b(1)^{1-e(1)} e(1) \left( \frac{p(1)}{\text{Cost}} \right)^{e(1)}, Q(2) \rightarrow b(2)^{1-e(2)} e(2) \left( \frac{p(2)}{\text{Cost}} \right)^{e(2)}, Q(3) \rightarrow b(3)^{1-e(3)} e(3) \left( \frac{p(3)}{\text{Cost}} \right)^{e(3)} \right\} \right\}$$

#### 5.5.4.2 Economic `Common`

Since the IADS is dual, the Cost variable already gives Minimal Cost. Solving  $p x = C$  ( $= \text{Cost}$ ) shows that  $C$  is eliminated, and hence cannot be determined in that manner. The returns to scale in  $y = \text{IADSterm}[C, p]^v$  for example have an effect on  $y$ , but not on costs, since Cost has been given or specified already (and optimal demand  $x[\text{Cost}, p]$  is independent of  $v$ ).

- By consequence:

```
fd1 = FactorDemand[];
fd2 = FactorDemand[IADS];
fd3 = FactorDemand[IADS, Factor -> FactorX[1]];
fd4 = FactorDemand[IADS, Cost, Factor -> FactorX[1]];
```

```
fd1 === fd2 === fd3 === fd4
```

```
True
```

The package operates as follows: the cost parameter is temporarily replaced by a variable, then Roy's rule is applied, and then the cost is set to its option value. This gives for example:

```
fd1 = FactorDemand[]
```

$$x(1) \rightarrow - \left( b(1)^{1-e(1)} e(1) \left( \frac{p(1)}{\text{Cost}} \right)^{e(1)-1} \right) / \left( \text{Cost} \left( - \frac{e(1) p(1) \left( \frac{p(1)}{\text{Cost}} \right)^{e(1)-1} b(1)^{1-e(1)}}{\text{Cost}^2} - \frac{b(2)^{1-e(2)} e(2) p(2) \left( \frac{p(2)}{\text{Cost}} \right)^{e(2)-1}}{\text{Cost}^2} - \frac{b(3)^{1-e(3)} e(3) p(3) \left( \frac{p(3)}{\text{Cost}} \right)^{e(3)-1}}{\text{Cost}^2} \right) \right)$$

```
fd1sol = fd1 /. ((x_)^((y_) + (z_)) -> Hold[x^y x^z]) /.
IADSPartRule // ReleaseHold // Simplify
```

$$x(1) \rightarrow \frac{\text{Cost } Q(1)}{p(1) (Q(1) + Q(2) + Q(3))}$$

#### 5.5.4.3 Collapse to the CES

When the factor powers are equal, then the IADS reduces to a CES. Take  $e(i) = e = 1 - S$ .

- Compare the following result to the dual CES:

```
IADS[FactorPowers →
  (FactorPowers /. Options[IADS] /. FactorE[i_] → e) ] //
  PowerExpand // Simplify // PowerExpand

IADS::CES : Equal powers, giving dual CES, Ces -> 1 - e

Costv Scale (p(1)e b(1)1-e + b(2)1-e p(2)e + b(3)1-e p(3)e)-v/e
```

#### 5.5.4.4 Elasticities

The example notebook IADS.nb shows how one can use *Mathematica* to derive price and cost (income) elasticities. In this derivation the cost shares feature prominently. From above solution to the factor demand, one can determine an expression for those cost shares:

```
costShare[i_] = FactorX[i] FactorP[i] / Cost


$$\frac{p(i)x(i)}{\text{Cost}}$$


costShare[1] /. fd1sol


$$\frac{Q(1)}{Q(1) + Q(2) + Q(3)}$$

```

Though the full derivation leads too far, it remains useful to show how *Mathematica* can deal with rules on the cost shares and for example with the average value of the power coefficients.

```
{Q[i_]/ Add[qs] → share[i], Sum[share[i], {i, 3}] → 1}

{  $\frac{Q(i)}{Q(1) + Q(2) + Q(3)}$  → share(i), share(1) + share(2) + share(3) → 1 }

averageE = Array[share, {NumberOfFactors}] . FactorPowers /.
  Options[IADS]

e(1) share(1) + e(2) share(2) + e(3) share(3)
```

The AUES[ ] routine does not work properly, since Cost here is specified as a constant (and thus gets zero derivatives). It may be useful in the future to include explicit expressions for the elasticities. (Not evaluated.)

```
(* AUES[Aggregate → y, Dual → Price, D → {FactorP[2], FactorP[3]}] *)
```

# 5.6 Applied General Equilibrium analysis

## 5.6.1 Summary

Here we analyse the general equilibrium of an economy with  $n$  commodities (sectors) and  $m$  factors of production, where the optimal social allocation is determined by a Bergson-Samuelson social welfare function. The main package is:

```
Economics [AGE]
```

Wassily Leontief was a pioneer in Applied General Equilibrium analysis. From the wealth of his work:

```
Economics [Leontief]
```

## 5.6.2 Introduction

Here we analyse the general equilibrium of an economy with  $n$  commodities (sectors) and  $m$  factors of production. The optimal social allocation is determined by a Bergson-Samuelson social welfare function. It is optional to let sectors trade intermediate goods. We analyse the economy by means of Consumption Possibility Curves, Production Possibility Curves and Edgeworth-Bowley boxes.

Asahi Noguchi and Silvio Levy already published the same results; see Asahi Noguchi's paper "General Equilibrium Models" in Varian (ed), "Economic and financial modeling with *Mathematica*", Springer Telos 1993. The AGE` package extends on their results and embeds them in the Economics Pack. I thank Asahi Noguchi for his permission for me to actually re-engineer his work. You will still benefit from his paper, since it explains the subject matter very clearly - while I have retained his notation whenever possible. Also, AGE` remains dependent on the "MyFindRoot" routine and package of Silvio Levy. I thank him for his permission to include this package in the Economics Pack.

The AGE` package conforms to canonised names, allows wide choice of utility and production functions, gives easy control of input, makes more output available, and produces the diagrams fast. If you have more than two sectors and two factors, then you can select two which you want to plot. You can also plot the expansion paths that depend upon changing parameters.

## 5.6.3 Example

The discussion below will use an example that is provided with the package.

AGExample

SetModel[AGExample] gives a 2 x2 example with only unknown resource parameters. An AGExpars is created that supplies such resource data



**SetModel [AGExample]**

```
{NumberOfSectors → 2, NumberOfFactors → 2,  
  IntermediatesQ → False, See Results[AGEmodel] for current results}
```

**5.6.4 Setting up a model**

Obvious options for a model are `NumberOfSectors`, `NumberOfFactors` and `IntermediatesQ` (True or False). The options `Utility` and `Production` determine the social utility and the sectoral production functions. A `MaxIterations` option is passed on to `FindRoot` (called by `MyFindRoot`).

`SetModel [opts]`  
`SetModel [ {opts}]`

loads the options for a particular model

- The most structured approach is to use `SetFunction` from `Economic`Common`` with a `Hold` construction. A user can define his or her own `SetFunction[myfunction, ...]`, and assign this to the sectors. The example is:

**AGExample**

```
{NumberOfSectors → 2, NumberOfFactors → 2, Aggregator → {CES},  
  IntermediatesQ → False, Production → {Sector(1) → Hold[SetFunction][CES, Ces → 1,  
    Return → 1, Constant → Scale(1) → 1, Factors → {FactorX(1, 1), FactorX(1, 2)},  
    FactorCoefficients → {FactorC(1, 1) → 0.8, FactorC(1, 2) → 0.2}], Sector(2) → Hold[SetFunction][  
    CES, Ces → 1, Return → 1, Constant → Scale(2) → 1, Factors → {FactorX(2, 1), FactorX(2, 2)},  
    FactorCoefficients → {FactorC(2, 1) → 0.2, FactorC(2, 2) → 0.8}}],  
  Utility → Hold[SetFunction][CES, Ces → 1, Return → 1, Constant → 1,  
    Factors → {Utility(1), Utility(2)}, FactorCoefficients → {FactorE(1) → 0.6, FactorE(2) → 0.4}]}
```

Since the above looks a bit forbidding, some ways for easier standard input have been created. One may enter `Utility → CES`, `Production → CES` for automatic generation for the sectors and factors. Also, one can define such standard types oneself, by setting the `Aggregator → {CES, owntype ...}` in the `Options[SetModel]`. In all such cases, `UFunction[ ]` and `PFunction[ ]` are called.

`PFunction [type, i, m]`  
  
`UFunction [type, m]`

gives the production function  
for Sector *i* with *m* factors of production  
  
gives the social welfare utility  
function with *m* sectors or consumption goods

*Type* must be a member of the `Aggregator` option of `SetModel`. For `type = LevelCES` then *m* is the structure. Note: a *type* function best uses `Hold[SetFunction][type, ...]`.

You can inspect the results of `SetModel[ ]` by evaluating `Results[AGEmodel]`, `$Utility` and `$Production`.

5.6.5 Parameters and assignment

Above SetModel[AGExample] has determined the following (new default) values for the parameters:

```
$Coefficients  
  
{FactorE(1) → 0.6, FactorE(2) → 0.4, Scale(1) → 1, FactorC(1, 1) → 0.8,  
  FactorC(1, 2) → 0.2, Scale(2) → 1, FactorC(2, 1) → 0.2, FactorC(2, 2) → 0.8}
```

Some parameters need to be assigned at the model setting stage, such as the CES parameter if you wish a Cobb-Douglas function. Other parameters may remain symbolic and can be assigned to a particular numerical value only at the latest moment. Usually you collect parameter assignments in a parameter list *pars* = {..., pari → vali, ....}. Then you call the Assign[pars] command. This assignment *replaces* symbols with values, rather than setting those symbols at those values. Assign takes default values from \$Coefficients. All subsequent AGE` functions will use Assign[ ] internally, leaving the user the possibility to work with only a limited list of parameters that are relevant for his or her analysis.

The following pars1 is an example of a parameter list.

```
pars1 = {Resources → {400, 600}};  
  
Assign[pars1]  
  
{FactorC(1, 1) → 0.8, FactorC(1, 2) → 0.2, FactorC(2, 1) → 0.2, FactorC(2, 2) → 0.8, FactorE(1) → 0.6,  
  FactorE(2) → 0.4, Resources(1) → 400, Resources(2) → 600, Scale(1) → 1, Scale(2) → 1}
```

<code>\$Coefficients</code>	contains the names and defaults of all coefficients in the model
<code>Assign[<i>pars</i>]</code>	assigns specific numerical values to the model's parameters. One can use input {Utility → ..., Production → {Sector[1] → ....}, Resources[1] → ... (and if required: Matrix → ...)}, or just a list for the parameters. Assign can use default parameter values if these have been supplied with SetFunction

If you run the AGExample with IntermediatesQ → True, then you might use the parameters:

```
parsint = pars1 ~Join~ {Matrix → {{.1, .3}, {.4, .1}}};
```

5.6.6 The equilibrium

<code>Equilibrium[<i>pars</i>]</code>	calls Allocation[pars] and solves key equilibrium results
---------------------------------------	---

- The example economy has equilibrium at:

```
Equilibrium[pars1]
{UEq → 293.649, YEq → 492.851, Price → {1, 0.678428},
  FactorPrices → {0.689991, 0.361424}, Consumption → {295.71, 290.584},
  Production → {295.71, 290.584}, Allocation → {FactorX(1, 1) → 342.857,
    FactorX(1, 2) → 163.636, FactorX(2, 1) → 57.1429, FactorX(2, 2) → 436.364}}
```

- A useful subroutine is:

```
AllocationTable[Allocation[pars1]]

```

	FactorX(1)	FactorX(2)
Sector(1)	342.857	163.636
Sector(2)	57.1429	436.364

The equilibrium routine generates the following equilibrium solutions:

CoEq[i][pars]	the consumption of commodity i
FactorXEq[i, j][pars]	the allocation of factor j to sector i
UEq[pars]	the social utility
XEq[i][pars]	the production of commodity i
YEq[i][pars]	national income. Commodity 1 is the numeraire
wEq[i, j][pars]	the marginal productivity of factor j in sector i
pEq[i][pars]	the price of commodity i. Good 1 is the numeraire
vpEq[i][pars]	the marginal value productivity of factor 1 in sector i

5.6.7 Plotting

```
CPCDiagram[pars_List, opts]

```

combines the plots of the Consumption Possibilities Curve,  
the Indifference contour, and the budget line,  
for two sectors assuming the other sectors to be constant.  
The sectors are selected with the Sector → {i, j} option.  
If IntermediatesQ → True,  
then the Production Possibilities Curve is included by giving CPCDiagram → All

```
EdgeworthBowley[pars_List, opts]

```

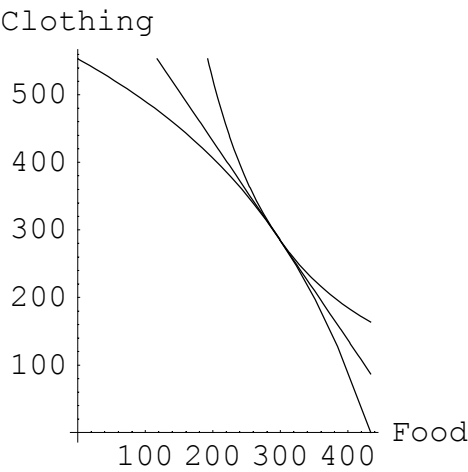
plots the 2 D Edgeworth–Bowley diagram for Sector → {i, j} and Factor → {f, g}

Use Results[CPCDiagram] for separate item plots, while numerical data per item can be found in Results[item].

The plots generate warning messages for difficult points. These messages can normally be neglected. They may however point to convergence problems; if so, you may find indications in Results[item].

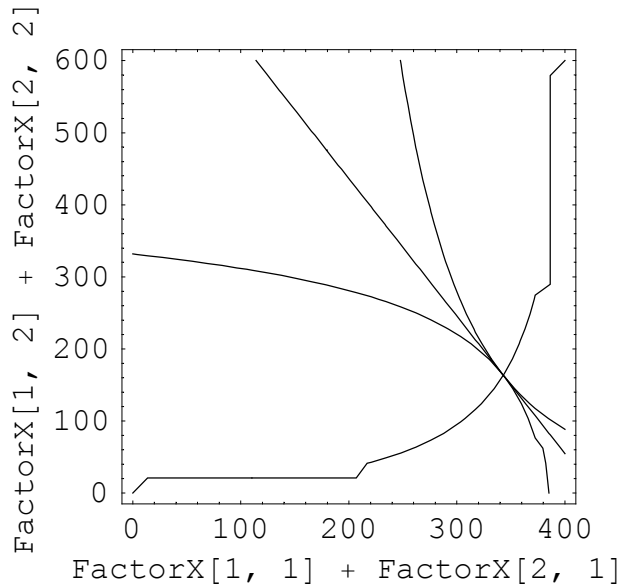
- Let us assume that the first sector makes food, and the other clothing.

```
CPCDiagram[pars1, AxesLabel -> {"Food", "Clothing"},  
AspectRatio -> Automatic, TextStyle -> {FontSize -> 12}]
```



- The following could be improved with the graphics option `PlotPoints`.

`EdgeworthBowley[pars1, TextStyle → {FontSize → 12}]`



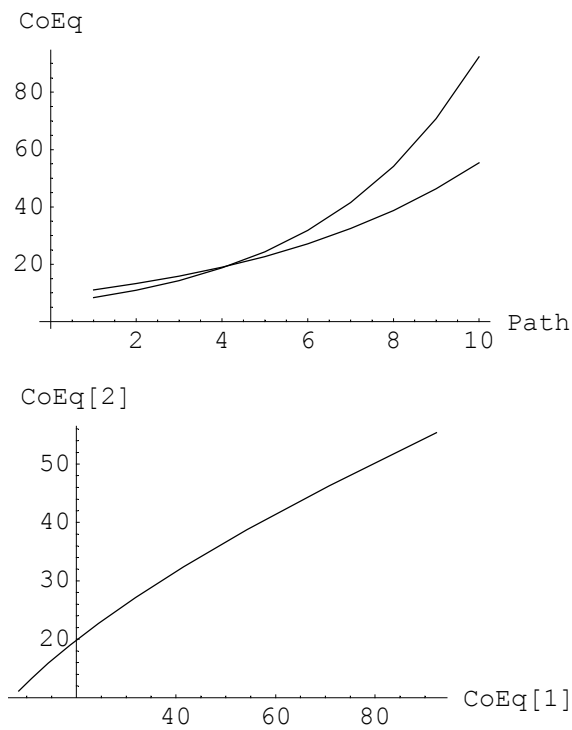
### 5.6.7 Expansion paths

<code>AGEpath[y, {pars1, ..., parsn}]</code>	gives a path plot. <i>y</i> can be a single item or a list, like { <i>z</i> [1], ..., <i>z</i> [ <i>m</i> ]}. The requirement is that each single <i>y</i> is defined like <i>UEq</i> and <i>YEq</i> , thus, <i>y</i> [ <i>parsi</i> ] must render a value. Those values are in <i>Results</i> [ <i>AGEpath</i> ]. <i>ListPlot</i> is used, so small sets of parameters give kinked output
<code>AGEpath[{pars1, ..., parsn}]</code>	determines the allocations (using the former result as startingvalue for the new)

The following plots the equilibrium consumptions for 10 periods. The first plot gives sectoral consumption as a function of time, the second plot gives a projection in the sector 1 - sector 2 space.

- We use the *AGExpars* created by `SetModel[AGExample]`.

`AGEpath[Array[CoEq, 2], Table[AGExpars[i], {i, 1, 100, 10}]]`



5.6.8 Subroutines

The following subsections discuss the subroutines for the above. As a list it is rather boring, but on occasion it can be useful to directly use these subroutines.

For the CPC diagram:

<code>CPCBudget [pars, opts]</code>	plots the equilibrium production budget line for two sectors assuming the others to be constant.
<code>NPPC [pars_List, opts]</code>	gives the consumption possibility contour
<code>PPC [pars_List, opts]</code>	gives the production possibility curve
<code>IndifferenceCurve [pars, opts]</code>	plots the equilibrium indifference contour for two sectors assuming the others to be constant.

For Edgeworth-Bowley:

`ContractCurve [pars_List, opts]`

plots the 2 D contract curve for factor allocations.  
Options are Sector  $\rightarrow \{i, j\}$  and Factor  $\rightarrow \{f, g\}$

`Isoquant [pars_List, opts]`

plots the 2 D isoquant of equilibrium output of a commodity as a function of factor allocations to the sector.  
Options are Sector  $\rightarrow \{i, j\}$  and Factor  $\rightarrow \{f, g\}$ ,  
and Take  $\rightarrow$  First for taking  $i$ , and Take  $\rightarrow$  Last for taking  $j$ .  
Factor allocations for  $i$  are read from the lower left corner in the diagram,  
factor allocations for  $j$  are read from the upper right corner.

`Isoquants [pars_List, opts]`

plots 2 D isoquants for Sector  $\rightarrow \{i, j\}$  and Factor  $\rightarrow \{f, g\}$ .  
If sector  $j === \text{All}$ ,  
then isoquants for  $i$  are given (for all available resources);  
otherwise the isoquants for  $j$  are  
given (for the equilibrium allocation to sectors  
 $i$  and  $j$  and excluding the allocations to the other sectors).

`Isocost [pars_List, opts]`

plots the 2 D isocost line for Sector  $\rightarrow \{i, j\}$  and Factor  $\rightarrow \{f, g\}$

The subroutine Allocation has different possibilities:

`Allocation [pars_List, x_List]`

sets the factors to the values in  $x$ , which may not be optimal.  
If element  $x[i, j]$  is Automatic, then the equilibrium solution for that element is taken.  
Per factor one element  $x[i, j]$  can best be Null so  
that it is computed as the remainder of the other allocations

`Allocation [pars_List, opts]`

for a list of parameters (including resources) gives the equilibrium allocations

There are two special options for Allocation:

<i>option</i>	<i>special value</i>	
ContractCurve	FactorX [.,.] → value	gives the allocation such that other sectors are at equilibrium values, but the sectors Sector → {i,j} and factors Factor → {k,m} are solved so that FactorX[.,.] is on the Contract Curve
Constraint	{FactorX [.,.] → value, ...}	allows disequilibria by constraining factors to specific values. Then the price equalities (vp) are preferred above the mpr' s (w).

Some routines also accept the option Allocation → y. Here y can be the output of Allocation[...], or a matrix of allocations of factors, or the expression ContractCurve → FactorX[.,.] → value.

More details and further examples can be found in the example notebooks AGEmodel.nb and AGEmodelContinued.nb.

5.6.9 The Static Leontief Model (SLM)

The static Leontief model is  $x = f + A x$ . The solution is  $x = (I - A)^{-1} f$ .

Here,  $A$  is a matrix of input-output coefficients, and each element  $A[i, j]$  represents the need of input from sector  $i$  for the production of one unit of output of sector  $j$ . In this model  $x$  is a vector of total output that equals the sum of final demand  $f$  and intermediate deliveries  $A x$ . For the reason that the model concentrates on the input and output of sectors, it also is called the input-output model.

The matrix  $(I - A)^{-1}$  is called the Leontief Inverse, and gives the multiplier coefficients for  $f$ . If the Leontief Inverse is not nonnegative, then some nonnegative  $f$  may result into negative  $x[i]$  elements. To allow for nonnegative  $x$  and  $f$  (i.e. to have meaningful economics) the Frobenius root of  $A$  must be smaller than 1.

IOAnalysis[A]	for basic input–output analysis. The Frobenius root is normalised.
IOAnalysis[A, B]	for the forward Dynamic Leontief Model.

Subroutines are:



FrobeniusRoot	output option for the real eigenvalue with largest absolute value and associated with a nonnegative eigenvector (called the FrobeniusVector)
FrobeniusVector	output option for the vector, normalised so that the first element is 1
IOMatrixQ[x]	tests whether x is a nonnegative square matrix
LeontiefInverse[A]	$= (I - A)^{-1}$
PositiveMatrixQ[x]	tests whether x is a positive matrix
NonNegativeMatrixQ[x]	tests whether x is a nonnegative matrix

The following example has high input-output coefficients and gives a barely productive economy. To generate a small amount of final demand, production must be relatively large.

```

A = {{0.7, 0.4}, {0.2, 0.46}};
f = {1, 4};

ioa = IOAnalysis[A]

{NonNegative Inverse → True, Root smaller than 1 → True, Eigenvalues → {0.887246, 0.272754},
  Eigenvectors →  $\begin{pmatrix} 0.90568 & 0.423962 \\ -0.683447 & 0.73 \end{pmatrix}$ , FrobeniusRoot → 0.887246,
  FrobeniusVector → {1., 0.468115}, LeontiefInverse →  $\begin{pmatrix} 6.58537 & 4.87805 \\ 2.43902 & 3.65854 \end{pmatrix}$ }

x → (LeontiefInverse /. ioa) . f
x → {26.0976, 17.0732}

```

### 5.6.10 The Dynamic Leontief Model (DLM)

The forward looking dynamic Leontief model adds a capital stock requirement  $k = B.x$  with investments  $i = k(+1) - k = B.(x(+1) - x)$ . Final demand then is  $f = c + i$ , where  $c$  is consumption. Expansion gives:

$$x(+1) = -B^{-1}c + (I + B^{-1}.(I - A))x$$

The solution consists of two parts: a *particular* and a *general* part. The particular part consists of  $x = 0$  and a nonnegative value for  $-B^{-1}c$ . The general part comes from the closed economy where  $c = 0$ . This closed form generates the equation  $x(+1) = (I + B^{-1}(I - A))x$ , which is a first order difference equation  $x(t+1) = Mx(t)$  for  $M = (I + B^{-1}(I - A))$ . We will write  $M = (I + D)$  and we will call  $D$  the Dynamic Leontief Matrix.

This system of difference equations runs the risk of 'causal indeterminacy', meaning that nonnegative start values still can result in negative values. A useful condition appears to be that the system must be relatively stable, i.e. that any deviation will return to a nonnegative path. Note that there exists a

Frobenius root  $\text{fr}(C)$  for  $C = (I - A)^{-1} \cdot B = D^{-1}$  - which matrix is called the Dynamic Leontief Inverse. When  $C > 0$ , when its inverse  $C^{-1}$  exists (i.e. when  $B^{-1}$  exists), and when  $r = (1 + 1 / \text{fr}(C))$  dominates the other roots of  $(I + D)$ , then and only then there is a relatively stable solution. Note: There is a condition of indecomposability here, but the current routine does not check on it.

DynamicLeontiefInverse	output option for $(I - A)^{-1} \cdot B$
DynamicLeontiefMatrix	output option for $B^{-1} \cdot (I - A)$
RelativeStabilityQ	output option for relative stability

The following reproduces Example 1 of DOSSO 1958, taken from A. Takayama, "Mathematical economics", The Dryden Press 1974. We use the subroutine `NSolveLinearDynamics` (from Appendix A.6).

- The application is straightforward:

```
A = {{1/3, 1/3}, {1/3, 1/3}};  
B = {{1, 0}, {0, 1}};
```

```
dioa = IOAnalysis[A, B]
```

$$\left\{ \begin{array}{l} \text{Positive Inverse} \rightarrow \text{True}, \text{DynamicLeontiefInverse} \rightarrow \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}, \text{DynamicLeontiefMatrix} \rightarrow \begin{pmatrix} \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} \end{pmatrix}, \\ \text{Eigenvalues} \rightarrow \left\{ 2, \frac{4}{3} \right\}, \text{Eigenvectors} \rightarrow \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}, \text{FrobeniusRoot} \rightarrow \frac{4}{3}, \\ \text{FrobeniusVector} \rightarrow \{1., 1.\}, \text{Matrix} \rightarrow \begin{pmatrix} \frac{5}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{5}{3} \end{pmatrix}, \text{RelativeStabilityQ} \rightarrow \text{False} \end{array} \right\}$$

- An economically sound path is the Frobenius path  $x_f(+1) = M x_f$ .

```
NSolveLinearDynamics[Matrix, FrobeniusVector, 4] /. dioa
```

$$\begin{pmatrix} 1. & 1. \\ 1.33333 & 1.33333 \\ 1.77778 & 1.77778 \\ 2.37037 & 2.37037 \\ 3.16049 & 3.16049 \end{pmatrix}$$

- However, an economically unsound path is the following. For a specific consumption path departing from  $c(0) = \{4, 2\}$  and  $x(0) = \{10, 9\}$ , we find that the economy eats up its capital stock:

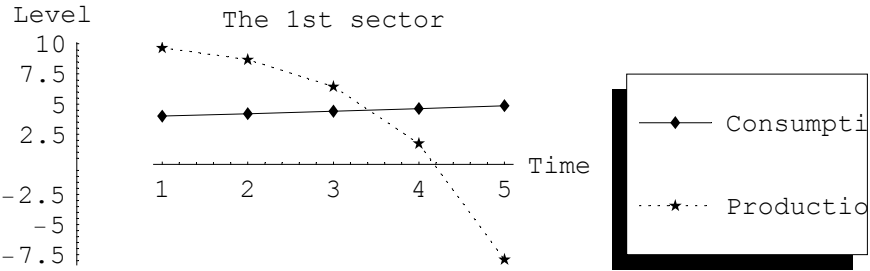
```
cons = Table[{4. 1.05^t, 2. 1.01^t}, {t, 0, 4}];  
minBinvsCons = (- Inverse[B] . #) & /@ cons;
```

```
xs = NSolveLinearDynamics[Matrix, {10, 9}, minBinvCons] /. dioa
```

$$\begin{pmatrix} 9.66667 & 9.66667 \\ 8.68889 & 10.8689 \\ 6.44852 & 13.1783 \\ 1.72426 & 17.7538 \\ -7.90618 & 26.9336 \end{pmatrix}$$

```
Needs["Graphics`MultipleListPlot`"]
```

```
MultipleListPlot[First /@ cons, First /@ xs,
  PlotJoined -> True, AxesOrigin -> {0, 0},
  AxesLabel -> {"Time", "Level"},
  PlotLabel -> "The 1st sector",
  TextStyle -> {FontSize -> 10},
  PlotLegend -> {"Consumption", "Production"},
  LegendShadow -> {-.05, -.05}]
```



## 5.7 Macro economics

---

### 5.7.1 Summary

The major problem in doing macro economics with *Mathematica* was in finding the proper working environment. The solution seems to be to use `SolveFrom` applications.

```
Economics[Economic`Macro]
```

### 5.7.2 Introduction

The major problem in doing macro economics with *Mathematica* was in finding the proper working environment. On one hand it is easy to type a few equations and solve these, but this way of working is hardly systematic and it puts a burden on communication when every economist has his or her own version of a model. On the other hand it is too ambitious to implement a huge model such as used in the big institutions like the IMF or CPB.

An intermediate position, or solution to this dilemma, is to standardise elements and to develop routines that allow one to work with these. We can distinguish (a) variables, (b) equations, (c) models (i.e. problem areas and names for models for these), and we can introduce standardised names for these. If groups of researchers adopt the discipline, then we can reap the benefits of standardisation. For example, if the standard variable name `Investment` is used for private investments and the standard model name `ISLMModel[ ]` is used for the Keynesian model, then I can use `Investment /. ISLMModel[ ]`, even though the actual equations may differ from actual model to actual model.

The `SolveFrom` format seems best for such standardisation. The format allows for a fixed set of basic equations with standardized names ("the model"), and for solving for specific substitutions. For example, `ISLMModel[{Income → 1000, $ISLMParameters}]` would solve for that particular value of `Income` while taking default parameters and values from `$ISLMParameters`. Users of course can define their own model and parameters, but there still are some conventions that help communication.

A canonic model will be in terms of  $\text{Income} = \text{Consumption} + \text{Investment}$ , and a compact model will be in the format  $Y = C + J$ . (And we use `J` here, since `I` in *Mathematica* stands for the complex operator or imaginary number.) Lists of replacement rules and legends allow the transformation and interpretation of extended and compact symbols.

The discussion below concentrates on the development of this working environment. As an application, some chapters of Dornbusch & Fischer (1994), "Macroeconomics", McGrawHill 6th ed. are implemented, adopting their symbols as much as manageable. We will regard an ISLM model, a small Surprise model, a model for value added and a labour market model.

### 5.7.2.1 Compact and extensive formats

A model in compact format is:

$$Y == C + I$$

$$Y == C + S$$

$$C == c Y + C0$$

The advantage of this format is that it fits the textbooks and that it helps understanding the mathematical structure. The set of single letters however is small and quickly runs out. The letter I already stands for the imaginary number,  $C[i]$  is the default form for the  $i$ th constant of integration, and S is used in the CES` package for the elasticity of substitution. One would want to use N or E for employment, but N is the numerical evaluator and E is the natural number. And P for Pricelevel conflicts with the P for Proposition in the Logic` package. And we would like to do economics with some logic.

*Mathematica* has a preference for longer symbols that can be the same throughout different applications. The extensive format of the above is:

$$\text{Income} == \text{Consumption} + \text{Investment}$$

$$\text{Income} == \text{Consumption} + \text{Saving}$$

$$\text{Consumption} == \text{Parameter}["\text{Consumption}"] \text{Income} + \text{Autonomous}["\text{Consumption}"]$$

An additional advantage of this extensive format is that it is directly obvious that Consumption stands for consumption. This advantage however is only limited. Consumption can be developed in more dimensions and e.g. levels of aggregation, so the supposed clarity may be misleading.

The compact and extensive formats are best allowed to co-exist so that their advantages can be enjoyed. To this end we don't assign numerical values but use replacement rules only, and have the following discipline:

- A model will have to be stated in the extensive format.
- A list of rules can be provided so that one can turn that extensive format into a compact format.

### 5.7.2.2 SolveFrom statics structure

Above considerations have resulted in the following programming structure:

1. The modeling routines are SolveFrom applications. Thus, they call on default lists of equations and apply substitutions of variables and parameters before solving for the remaining unknowns (that the routine finds itself).

2. The advantage of Solve is that it allows for symbolic solutions. The disadvantage is that the dynamics are awkward. See the `Model`` package for dynamics with `NSolve`. You may want to change the macro economic equations into dynamic equations.
3. The models themselves are lists of equations. The best way to see these equations is to use `MatrixForm`. The default models have a compact form that can be found by using the replacement rules `ToMacroSymbols`. Use `eluR[ToMacroSymbols]` for their reverse. There are also legends, best looked at in `MatrixForm`.
4. Extensive symbols are generally unprotected to allow for maximal ease of use. However, at start up time the compact symbols are protected, since it has appeared that one is inclined to set such symbols to values while this is not the intended use (e.g. with the legend). To unprotect these symbols, evaluate `Unprotect @@ MacroSymbols`. To protect them again, evaluate `Protect @@ MacroSymbols`.
5. The models supplied here are denoted with a \$ affixed to the name such as `$ISLMEquations` or `$Surprises`. You can of course make your own lists of equations, and set the options of the appropriate model routine to that list. For example: `SetOptions[ISLMModel, Equations → MyOwnISLM]`. (Model developers should have the discipline to make their models like this.)
6. Default parameter values are in `$nameParameters`, again as rules, so that they can be used directly in a model routine call. You of course can make your own lists of parameters, and then solve as `ISLMModel[MyOwnParameters]`
7. Results can be tabulated using `SolveShow[]` for single call solutions, and `SolveFrom[TableForm, ...]` for calls with different parameters.

### 5.7.3 Example model

The method is best explained by a small model. The `ExampleModel[ ]` contains default equations in extensive format, we can get its compact format by symbolic replacement, and we can solve for arbitrary selections of variables.

<code>ExampleModel[{xs}, ys]</code>	is a <code>SolveFrom</code> application, with <code>xs</code> rules, <code>ys</code> eliminables
<code>\$Surprises</code>	a list of some surprise equations
<code>ToMacroSymbols</code>	rules to change extensive symbols into compact symbols
<code>MacroSymbols</code>	the list of protected compact symbols

- Let us look at the full model, the compact model, and the legend.

**`$Surprises // MatrixForm`**

$$\left( \begin{array}{l} \text{Inflation} == \frac{\text{PriceLevel}}{\text{Lag}(\text{PriceLevel})} - 1 \\ \text{Real}[\text{RateOfInterest}] == \text{RateOfInterest} - \text{Inflation} \\ \text{Expected}(\text{Real}[\text{RateOfInterest}]) == \text{RateOfInterest} - \text{Expected}(\text{Inflation}) \\ \text{Surprise}(\text{Inflation}) == \text{Inflation} - \text{Expected}(\text{Inflation}) \\ \text{Surprise}(\text{Real}[\text{RateOfInterest}]) == \text{Real}[\text{RateOfInterest}] - \text{Expected}(\text{Real}[\text{RateOfInterest}]) \end{array} \right)$$

**`$Surprises /. ToMacroSymbols // MatrixForm`**

$$\left( \begin{array}{l} p == \frac{Py}{Py[-1]} - 1 \\ R == r - p \\ Re == r - pe \\ \text{Surprise}(\text{Inflation}) == p - pe \\ \text{Surprise}(\text{Real}[\text{RateOfInterest}]) == R - Re \end{array} \right)$$

**`ListOfSymbols[ExampleModel]`**

$$\left( \begin{array}{ll} \text{Inflation} & p \\ \text{PriceLevel} & Py \\ \text{RateOfInterest} & r \\ \text{Expected}(\text{Real}[\text{RateOfInterest}]) & Re \\ \text{Lag}(\text{PriceLevel}) & Py[-1] \\ \text{Real}[\text{RateOfInterest}] & R \end{array} \right)$$

Complexer symbols are in Strings, like "Py[-1]". The reason is that replacing  $Py \rightarrow 1.4$  for the non-lagged price level otherwise would give  $1.4[-1]$  for the lagged price level.

- This runs the model, in this case eliminating some nonessential variables:

```
sol = ExampleModel[{} ,  
  {Inflation, RateOfInterest, Real["RateOfInterest"]}];
```

**`Simplify[Last[sol]] /. ToMacroSymbols // Transpose // MatrixForm`**

$$\left( \begin{array}{l} Py \rightarrow Py[-1] (pe - \text{Surprise}(\text{Real}[\text{RateOfInterest}]) + 1) \\ \text{Surprise}(\text{Inflation}) \rightarrow -\text{Surprise}(\text{Real}[\text{RateOfInterest}]) \end{array} \right)$$

5.7.4 The I = S and L = M model

The  $I = S$  condition determines equilibrium in the goods market, the  $L = M$  condition determines equilibrium in the money market. Together they determine national income and the rate of interest. The `$ISLMEquations` also contain some equations on the deficit and national debt that provide key indicators to evaluate the results. First regard the list of symbols.

ListOfSymbols[ISLMModel]

Consumption	$C$
DisposableIncome	$YD$
Government	$G$
Income	$Y$
Inflation	$p$
InterestFloor	$r_{min}$
Investment	$J$
Liquidity	$L$
Money	$M$
NetExports	$NX$
PriceLevel	$P_y$
RateOfInterest	$r$
Saving	$S$
Taxes	$T$
TransferIncome	$TR$
Velocity	$V$
Wealth	$NPW$
Autonomous(Consumption)	$C_{aut}$
Autonomous(Investment)	$J_{aut}$
Bonds(Demand, Real)	$RDB$
Bonds(Supply, Nominal)	$NSB$
Lag(Debt)	$Debt[-1]$
Lag(PriceLevel)	$P_y[-1]$
Parameter(ConsumptionFunction)	$c$
Parameter(Investment, RateOfInterest)	$b$
Parameter(LiquidityPreference, Interest)	$h$
Parameter(LiquidityPreference, Transaction)	$k$
Parameter(Taxes, Income)	$\tau$

<code>ISLMModel[{xs}, ys]</code>	is a <code>SolveFrom</code> application with rules <code>xs</code> and eliminables <code>ys</code>
<code>\$ISLMEquations</code>	default equations for a textbook $I = S$ & $L = M$ model
<code>\$ISLMParameters</code>	default parameter settings for <code>\$ISLMEquations</code>

Note that the rate of interest and inflation are measured in perunages.

Though it is possible to call the `ISLMModel[ ]` directly, we show the separate steps.



### 5.7.4.1 Solving for $I = S$

<code>\$ISEquations</code>	equations for the commodity market partial equilibrium $I = S$
<code>IS[Y]</code>	gives the rate of interest $r$ as a function of $Y$ , assuming only partial equilibrium on the commodity market. Has to be defined by the user. The start up value fits the start up model structure and parameter values

- These are the equations for the commodity market:

```
$ISEquations /. ToMacroSymbols // MatrixForm
```

$$\begin{pmatrix} Y == C + G + J + NX \\ YD == -T + TR + Y \\ T == \text{tau } Y \\ C == \text{Caut} + TR + c(YD - TR) \\ S == YD - C \\ J == \text{Jaut} - b r \end{pmatrix}$$

- Solve the equations for the rate of interest as a function of national income:

```
solIS = Solve[$ISEquations, {RateOfInterest},  
{Consumption, Investment, Taxes, DisposableIncome, Saving}];
```

```
IS[] → (RateOfInterest /. solIS[[1]]) /.  
ToMacroSymbols // Simplify
```

$$IS() \rightarrow \frac{\text{Caut} + G + \text{Jaut} + NX + TR + c Y - c \text{tau } Y - Y}{b}$$

- For a normal set of parameters and variable values:

```
IS[Income_] = (RateOfInterest /. solIS[[1]]) /.  
$ISLMPParameters // Simplify
```

0.42 – 0.000088 Income

### 5.7.4.2 Solving for $L = M$

Included is a sensitive liquidity preference schedule.

```
LiquidityPreference[Y, r, {k:0.2, h:0.02, rmin:0.01}]
```

gives  $(1 + h / (r - rmin)) k Y$  as the real demand for money, for income  $Y$  and rate of interest  $r$ . Parameters are transaction parameter  $k$  (default .2), the minimal rate of interest  $rmin$  (default .01), and a sensitivity parameter  $h$  (default .02).

The equations are:

<code>\$LMEquations</code>	equations for the money market partial equilibrium $L = M$
<code>LM[Y]</code>	gives the rate of interest $r$ as a function of $Y$ , assuming only partial equilibrium on the money market. Has to be defined by the user. The start up value fits the start up model structure and parameter values

```
$LMEquations /. ToMacroSymbols // MatrixForm
```

$$\left( \begin{array}{l} NPW == M + NSB \\ L == k \left( \frac{h}{r - rmin} + 1 \right) Y \\ M V == P_Y Y \\ L == \frac{M}{P_Y} \end{array} \right)$$

```
solLM = Solve[$LMEquations, {RateOfInterest},  
              {Liquidity, Wealth, Velocity}];
```

```
LM[] → (RateOfInterest /. solLM[[1]]) /.  
        ToMacroSymbols // Simplify
```

$$LM() \rightarrow \frac{M rmin + k P_Y (h - rmin) Y}{M - k P_Y Y}$$

- For a normal set of parameters and variable values:

```
LM[Income_] = (RateOfInterest /. solLM[[1]]) /.  
               $ISLMPParameters // Simplify
```

$$\frac{0.002 \text{ Income} + 16.}{1600 - 0.2 \text{ Income}}$$

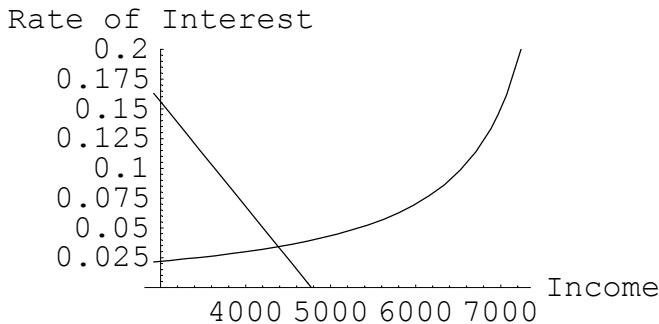
### 5.7.4.3 Together I = S and L = M

The routine ISLMPlotPrepare[ ] takes the IS[ ] and LM[ ] functions as input, determines the plotting domain, and prepares the input for a Plot. The output can be submitted to Plot or be manually adapted by the user first.

```
ISLMPlotPrepare[f:IS, g:LM]
```

prepares a plot of ISLM,  
where the domain is found by solving  $IS[Y] == LM[Y]$  and  $LM[Y] == .2$

```
Plot @@ ISLMPlotPrepare[]
```



#### 5.7.4.4 ISLM comparative statics

We can do comparative statics by running the `ISLMModel[ ]` for different parameters or variable values. The `SelectSolutionRules` help to get rid of non-economic solutions.

`SelectSolutionRules` to select useful solutions. Default eliminated are solutions with negative `RateOfInterest` and `PriceLevel`

```
res1 = ISLMModel[$ISLMParameters] /.  
      SelectSolutionRules;  
  
res2 = ISLMModel[{Parameter["ConsumptionFunction"] -> .9,  
                  $ISLMParameters  
                }] /. SelectSolutionRules;
```

- The following would print these two results (not evaluated here).

```
SolveFrom[TableForm, res1, res2]
```

#### 5.7.5 Value added

While the ISLM model already tells us something about inflation - via the quantity of money - there is also the labour market where wages can drive up costs and prices. The ISLM sectors and the labour market are linked via Value Added. So it is proper to discuss Value Added first.

ValueAdded[{xs}, ys]	is a SolveFrom application with rules xs and eliminables ys
\$ValueAddedEquations	list of equations
\$ValueAddedParameters	example parameter settings

Inflation is regarded here as the relative price change of the PriceLevel, and the latter is the price of Income or Real Value Added. Hence gross output (Production) has a different deflator, here Price.

```
$ValueAddedEquations /. ToMacroSymbols // MatrixForm
```

$$\left( \begin{array}{l} P_y Y == VA \\ VA == rPkK + LE W \\ P_p Y_p == VA + X_m \\ P_p Y_p == (m + 1) (LE W + X_m) \\ Y_p == \frac{X_m}{P_m} + Y \\ X_m == P_m \theta Y_p \\ p == \frac{P_y}{P_y[-1]} - 1 \end{array} \right)$$

```
ListOfSymbols[ValueAdded]
```

$$\left( \begin{array}{ll} \text{Employment} & LE \\ \text{Income} & Y \\ \text{Inflation} & p \\ \text{Intermediate} & X_m \\ \text{MarkUp} & m \\ \text{MaterialsPrice} & P_m \\ \text{Price} & P_p \\ \text{PriceLevel} & P_y \\ \text{Production} & Y_p \\ \text{ValueAdded} & VA \\ \text{Wage} & W \\ \text{Capital(Income)} & rPkK \\ \text{Lag(PriceLevel)} & P_y[-1] \\ \text{Parameter(Intermediate, Production)} & \theta \end{array} \right)$$

- This would produce the default solution and tabulate the results (not evaluated here):

```
ValueAdded[$ValueAddedParameters]
```

```
SolveShow[Last[%]]
```

### 5.7.6 The labour market

<code>LabourMarket[{xs}, ys]</code>	is a <code>SolveFrom</code> application with rules <code>xs</code> and eliminables <code>ys</code>
<code>\$LabourMarketEquations</code>	equations for a textbook labour market model
<code>\$LabourMarketParameters</code>	parameter settings for <code>\$LabourMarketEquations</code>

Wage inflation will be affected with a positive sign by expected inflation, by profits and by forward shifting of taxes, while it will be affected with a negative sign by unemployment. Wage inflation ( $w$ ) and unemployment ( $u$ ) are measured in perunages. The LIQ and NAWIRU are discussed in the next section.

**\$LabourMarketEquations /. ToMacroSymbols // MatrixForm**

$$\left( \begin{array}{l} \text{LIQ} == \frac{\text{LE } W}{\text{Py } Y} \\ \text{yl} == \frac{Y}{\text{LE}} \\ U == \text{LS} - \text{LE} \\ u == 1 - \frac{\text{LE}}{\text{LS}} \\ v == 1 - \frac{\text{LE}}{\text{LD}} \\ W == W[-1] (w + 1) \\ w == \text{pe} + \text{rho tau} + \text{PhillipsCurve}(\text{LIQ}, u) \\ w[-1] == \text{pe} + \text{rho tau} + \text{PhillipsCurve}(\text{LIQ}, \text{NAWIRU}) \end{array} \right)$$

**ListOfSymbols[LabourMarket]**

$$\left( \begin{array}{ll} \text{Employment} & \text{LE} \\ \text{Income} & Y \\ \text{LabourIncomeQuote} & \text{LIQ} \\ \text{LabourProductivity} & \text{yl} \\ \text{Price} & \text{Pp} \\ \text{PriceLevel} & \text{Py} \\ \text{Unemployment} & u \\ \text{UnemploymentLevel} & U \\ \text{Vacancy} & v \\ \text{Wage} & W \\ \text{WageInflation} & w \\ \text{Expected(Inflation)} & \text{pe} \\ \text{Labour(Demand)} & \text{LD} \\ \text{Labour(Supply)} & \text{LS} \\ \text{Lag(Wage)} & W[-1] \\ \text{Lag(WageInflation)} & w[-1] \\ \text{Parameter(WageInflation, Taxes)} & \text{rho} \end{array} \right)$$

- This would produce the default solution with Employment the remaining unknown (not evaluated here):

**LabourMarket[\$LabourMarketParameters]**

SolveShow[Last[%] // Simplify]

5.7.6.1 The Phillipscurve, NAIRU and the NAWIRU

PhillipsCurve[Line    Inverse    Log, LIQ, u, {alpha, beta, gamma}]	
	gives a contribution to the development of wage inflation due to the tensions on the labour market, with u unemployment and LIQ the labour income quote in total value added (as an indication of profits).
NAIRU[expr, u]	symbol for the non-accelerating inflation rate of unemployment, solves u from expr == 0, regarding expr as a function of u
NAWIRU	symbol for the non-accelerating wage-inflation rate of unemployment

Note that some would restrict the Phillipscurve effect in the wage equation to  $u$  only, but it is better to include the labour income quote (LIQ) as an indication of profits. Let us then discuss when the solution of  $NAIRU[expr, u]$  is the Non-Accelerating-Inflation Rate-of-Unemployment.

1. If  $p$  is inflation and  $p[-1]$  was last year's inflation, then the acceleration of inflation is  $p - p[-1]$ . Note that this should be divided by  $p[-1]$  again (for a proper application of the notion of acceleration), but this is not common in the literature.
2. Wage inflation will be  $w = pe + \text{PhillipsCurve}[LIQ, u]$ , for  $pe$  anticipated inflation,  $u$  unemployment and LIQ the labour income quote in total value added.
3. Let  $f[ ]$  give the functional relation of wage inflation  $w$  to general inflation, then:  $p = f[w]$
4. Hence, solve  $0 = p - p[-1] = f[w] - p[-1] = f[pe + \text{PhillipsCurve}[LIQ, NAIRU]] - p[-1]$  for the NAIRU.
5. The latter however is difficult, due to  $f[ ]$ . Therefor one often is satisfied with the NAWIRU (the W for wage) as solved from:

$$w - w[-1] = pe + \text{PhillipsCurve}[LIQ, NAWIRU] - w[-1] = 0$$

Above PhillipsCurve function allows for Line, Inverse and Log formats. The default parameter values for Line are  $\{.10, -.03, -2\}$ , the Inverse version has no defaults, and defaults for Log are  $\{-.35, -.1, -.1\}$ . In Line and Log cases, the LIQ and  $u$  can be seen as measured against some optimal value, e.g.  $\text{Log}[u / u^*]$ , though these additional parameters are here solved out into the constant.

- A Phillipscurve with inverse format, and solved for the NAWIRU:

PhillipsCurve[Inverse, LIQ, u, {α, -β, -γ}]

$$\alpha - \frac{\beta}{LIQ} - \frac{\gamma}{u}$$

NAWIRU  $\rightarrow$  NAIRU[pe + % - w[-1], u]

$$\text{NAWIRU} \rightarrow \frac{\text{LIQ} \gamma}{\text{LIQ} (\text{pe} + \alpha - w(-1)) - \beta}$$

- A Phillipscurve with Log format, and a NAWIRU assuming that  $\text{pe} - w[-1] = .03$

PhillipsCurve[Log, .66, u, {}]

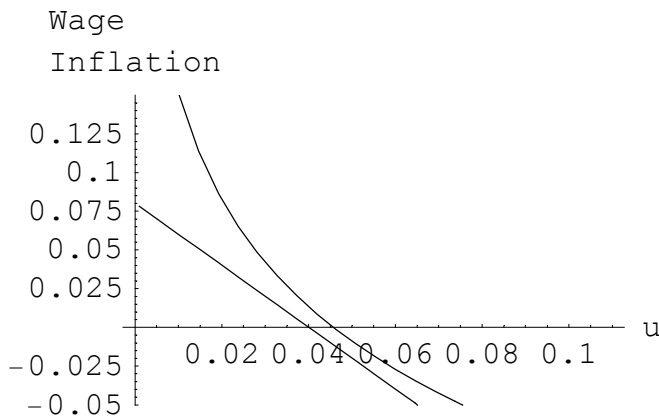
$$-0.1 \log(u) - 0.308448$$

NAWIRU  $\rightarrow$  NAIRU[% + .03, u]

$$\text{NAWIRU} \rightarrow 0.0617609$$

- A plot of Log and Line Phillipscurves with LIQ = 66% and default parameters:

```
Plot[ {PhillipsCurve[Log, .66, u, {}],
      PhillipsCurve[Line, .66, u, {}]}, {u, .001, .11},
      PlotRange  $\rightarrow$  {-0.05, 0.15}, TextStyle  $\rightarrow$  {FontSize  $\rightarrow$  12},
      AxesLabel  $\rightarrow$  {"u", "Wage\nInflation"}]
```



### 5.7.6.2 The Labour Income Quote (LIQ)

Since the labour income quote LIQ enters the Phillipscurve, and since the level of wages affects that quote (both directly by the wages and indirectly by the price level), there is a feedback.

- The feedback for a linear Phillipscurve

```
solLin = Solve[$LabourMarketEquations /.
  {PhillipsCurve  $\rightarrow$  ( $\alpha + \beta \#1 + \gamma \#2\&$ )},
  {LabourIncomeQuote},
  {NAWIRU, UnemploymentLevel, Wage, Income,
   Employment, WageInflation, Vacancy}];
```

```
solLin /. ToMacroSymbols
```

$$\left\{ \left\{ \text{LIQ} \rightarrow \frac{W[-1] (\text{pe} + \rho \tau + \alpha + u \gamma + 1)}{P_y y_l - W[-1] \beta} \right\} \right\}$$

- The feedback for a logarithmic Phillips curve:

```
solLog = Solve[$LabourMarketEquations /.
  {PhillipsCurve -> (\alpha + \beta Log[#1] + \gamma Log[#2]&)},
  {LabourIncomeQuote},
  {NAWIRU, UnemploymentLevel, Wage, Income,
    Employment, WageInflation, Vacancy}];
```

```
InverseFunction::ifun:
```

```
Warning: Inverse functions are being used. Values
may be lost for multivalued inverses.
```

```
Solve::ifun: Inverse functions are being
used by Solve, so some solutions may not be found.
```

```
solLog /. ToMacroSymbols
```

$$\left\{ \left\{ \text{LIQ} \rightarrow - \frac{W[-1] \beta \text{ProductLog} \left[ - \frac{e^{-\frac{\text{pe} + \rho \tau + \alpha + 1}{\beta}} P_y u^{-\frac{\gamma}{\beta}} y_l}{W[-1] \beta} \right]}{P_y y_l} \right\} \right\}$$

It turns out that Solve is strong on polynomials, but weak on more complex structures such as the logarithmic version of the Phillips curve. In this case, we can work around the problem by using above solution for the LIQ. This solution has been copied in below LIQRule[ ] that can be used in LabourMarketModel[ ]. The example notebook LabourMarket.nb discusses this further.

```
LIQRule[Line || Log, {alpha, beta, gamma}]
```

gives a rule for the LabourIncomeQuote as solved from the  
LabourMarket[ ] using the relevant PhillipsCurve, setting rho = 0

- For example, applying the default parameters and variable values:

```
LIQRule[Line, {\alpha, \beta, \gamma}] /. $LabourMarketParameters
```

$$\text{LabourIncomeQuote} \rightarrow \frac{16.3462 (\alpha + \text{Unemployment} \gamma + 1.03)}{25 \text{PriceLevel} - 16.3462 \beta}$$



**5.7.7 Undefined symbols**

The package contains a number of symbols that, apart from being used in equations and such, have no other definition attached to them.

Bonds	Imports	PropensityToConsume
Debt	Inflation	Saving
Deficit	Investment	Taxes
DemandPerCapita	LabourIncomeQuote	TransferIncome
DisposableIncome	Liquidity	Unemployment
Employment	MaterialsPrice	UnemploymentLevel
Exports	Money	Vacancy
ExternalAccount	NetExports	Velocity
GDP	Participation	Wage
GNP	PriceLevel	WageInflation

# 5.8 Taxes and premiums

## 5.8.1 Summary

The `Taxes`` package implements key concepts for the income tax and the social security premiums. You can enter your own functions and parameters, in flexible formats. Topics dealt with are:

- piecewise linearity and / or curvature, and determination and plots of marginal rates
- the minimum wage and the Tax Void Diagram
- revenue calculation based on a lognormal income distribution
- optimal tax subject to revenue conditions
- dynamics such as indexation, dynamic curvature and marginal.

- Load the package:  
`Economics [Taxes]`

## 5.8.2 Introduction

Premiums levied on wages (gross earnings), and taxes are levied on income, i.e. wages after deduction of premiums. Taxes and premiums thus have different bases. The levy is defined as the sum of taxes and premiums, and the levy will be measured at the same base as the premiums.

<code>Tax[y]</code>	at startup <code>PieceWiseLinear[y, TaxRates]</code>
<code>Premium[w]</code>	at startup <code>PieceWiseLinear[w, PremiumRates]</code>
<code>Levy[w, p:Premium, t:Tax]</code>	gives the total burden of tax and premium combined
<code>NetIncome[w, p:Premium, t:Tax]</code>	gives net income
<code>TaxOnWage[w, p:Premium, t:Tax]</code>	gives the tax on the wage <code>w</code> base instead of the income <code>y</code> base

By definition  $\text{Income} = \text{Wage} - \text{Premium}[\text{Wage}]$ , so that taxable income remains after deduction of premiums. Then  $\text{NetWage}$  or  $\text{NetIncome} = \text{Income} - \text{Tax}[\text{Income}]$ . In principle there can be a difference between wages and other incomes when only wages are subjected to premiums. The user will have to introduce different tax groups here.

Taxes and premiums can have any structure. A common form is the piecewise structure.

`PieceWiseLinear[y, rates]`

constructs a piecewise linear function. The rates are given as pairs,  $\{\{x_1, r_1\}, \{x_2, r_2\}, \dots\}$  where  $x_i$  is the point where the slope (marginal rate)  $r_i$  starts to be effective. It is assumed that  $x_1$  is proper exemption, so that the slope from 0 till  $x_1$  is zero. The variable  $y$  is the variable used in the function. The function uses If-statements so that it can be differentiated

`PiecewiseWhich[x, rates]`

is similar to `PieceWiseLinear`, but applies `ToPiecewiseLinear` to create a `Which` object

Reserved names are `TaxRates` and `PremiumRates`. The defaults take the Dutch situation of 1997. Income  $y$  and wages  $w$  are considered gross values, and measured in thousands DFL 1997 with  $\$1 = 1.7$  DFL.

- The rates at startup are:

**TaxRates**

$$\begin{pmatrix} 7.102 & 0.373 \\ 45.96 & 0.5 \\ 97.422 & 0.6 \end{pmatrix}$$

**PremiumRates**

$$\begin{pmatrix} 0 & 0.0975 \\ 20.4 & 0.2545 \\ 58.884 & 0.09745 \\ 60.75 & 0.0285 \end{pmatrix}$$

If one assigns values to `TaxRates` and `PremiumRates`, then the rates are not yet effective: first one has to `Clear[Tax, Premium]` and then define e.g. `Tax[y_] = PieceWiseLinear[y, TaxRates]` and `Premium[w_] = PieceWiseLinear[w, PremiumRates]`. (Other function definitions of course are possible too.) The following automates the process.

`SetTaxAndPremium[p, t]`

clears the `Tax` and `Premium` functions, and defines them as piecewise linear functions, while setting `PremiumRates = p` and `TaxRates = t`

`SetTaxAndPremium[]`

resets to the defaults

### 5.8.3 Levy Plot

The levy plot includes the following elements:

1. the levy line

2. the 45-degrees line, so that the difference between the levy and the 45-degree line gives net income
3. the benefit line parallel to the 45-degrees line, i.e. that part of net income that is required for subsistence and the replacement rate
4. the minimum wage given by the intersection of levy and benefit line.

```
LevyPlot[b:0, e:130, p:Premium, t:Tax, opts]
```

plots the levy situation for the wage b to e

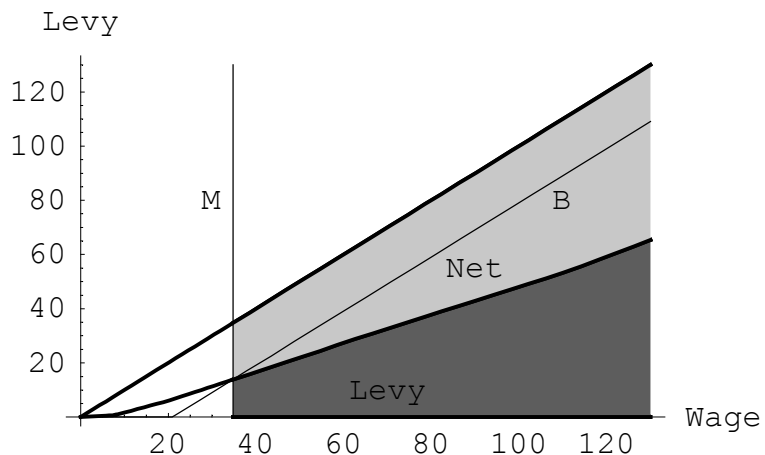
```
NetIncomePlot[b:0, e:130, p:Premium, t:Tax, opts]
```

a plot like the LevyPlot but translated into net income

- The following plots with some useful labels (while the first plot is suppressed).

```
lp = LevyPlot[0, 130, Premium, Tax,
  AxesOrigin → {0,0}, PlotStyle → Thickness[.007],
  AxesLabel → {"Wage", "Levy"}, FilledPlot → True,
  Fills → { Hue[0.55, 1, .7], Hue[0.11, .7, 1]},
  PlotRange → All, TextStyle → {FontSize → 12}];
```

```
Show[lp, Graphics[{Text["M", {30, 80}],
  Text["Levy", {70, 10}], Text["Net", {90, 55}],
  Text["B", {110, 80}]],
  TextStyle → {FontFamily → "Helvetica",
  FontSize → 14}]]
```



Note that there is a levy void below the minimum wage, since people are not allowed to work below the minimum wage, thus those workers don't earn anything that can be taxed. This point is clarified by the option `Filled → True` and by the Tax Void Diagram below.

### 5.8.4 Minimum notions

The levy plot relies on the following notions:

<code>Bentham[w, r:0.5, x:Exemption]</code>	= $r(w - x)$ the basic tax system discussed by Bentham, here with a default marginal rate of 50 %
<code>Exemption</code>	gives theoretical exemption, equal to the net minimum wage, equal to the current value of Subsistence * ReplacementRate
<code>ReplacementRate</code>	a reserved name, with default value 1.10
<code>Subsistence</code>	a reserved name, with default value 19 (1000 DFL)
<code>BenefitLine[x]</code>	evaluates $x - \text{ReplacementRate} \cdot \text{Subsistence}$ . Its intersection with the levy line gives the minimum wage
<code>TaxCredit</code>	symbol only

Note that if you want to assign a Tax function with a tax credit, you might `Clear[Tax]` and define:

$$\text{Tax}[y\_]=\text{Max}[0, \text{PieceWiseLinear}[y, \text{TaxRates}]-\text{TaxCredit}]$$

The following are utilities to determine the gross minimum wage from net considerations.

MinimumIncome	a reserved name
MinimumWage	a reserved name, giving the minimum wage found by FindMinimumWage[] or SolveMinimumWage[]
FindMinimumWage [ <i>p:Premium, t:Tax</i> ]	computes MinimumWage and MinimumIncome, assuming values for Subsistence and ReplacementRate
SolveMinimumWage [ <i>p:Premium, t:Tax</i> ]	tries for an algebraic solution of the MinimumWage and MinimumIncome, assuming values for Subsistence and ReplacementRate. It only works when p and t are proper algebraic relations (e.g. no If statements)

- The (gross) minimum wage in DFL 1000 is:

**FindMinimumWage[]**

34.7493

5.8.5 Tax transforms and marginal rates

Two basic transformations are de average levy and the gross-to-net-ratio.

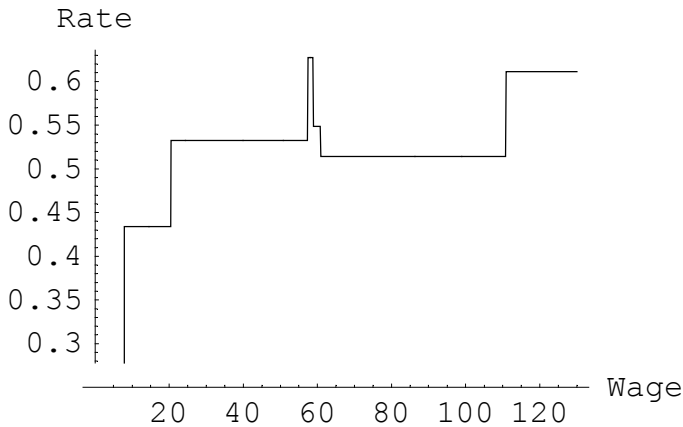
AverageLevyRatio [ <i>w, p:Premium, t:Tax</i> ]	gives the average levy as a ratio to the gross wage: Levy[w, p, t] / w
GrossToNetRatio [ <i>w, p:Premium, t:Tax</i> ]	gives the ratio of the gross to the net wage: w / NetIncome[w, p, t]

Due to the interaction of taxes and premiums, the marginal rate structure is not obvious.

MarginalRate[f, y (, y0)]	gives the marginal rate for a function f,
MarginalRate[f[y], y (, y0)]	either as a function of y or at a point y0
MarginalRate[f, y (y0)]	
MarginalRate[f, y0]	

The rate is computed as  $(f[y + .001] - f[y]) / .001$ , since, when the tax function has Max or Min statements, the derivative gives problems. The 1997 Dutch situation has its wonders:

```
Plot[MarginalRate[Levy, w], {w, 0, 130},
AxesOrigin -> {0, 0.25}, AxesLabel -> {"Wage", "Rate"}, TextStyle ->
{FontSize -> 12}]
```



### 5.8.6 Tax revenue

Tax revenue can be computed by applying the tax function to all the incomes in the country. When there is a neat income distribution, we can use integration. Productivities and wages are commonly distributed in a LogNormal manner. The wage density used here gives the number of labour years per wage level.

```
WageDensity[x, m:3.85, s:0.5, n:8]
```

gives n times the lognormal density at x, for lognormal parameters m and s. Thus n equals NIntegrate[ WageDensity[x, m, s, n], {x, 0, Infinity}] and gives the total number of labour years involved (measured in millions)

**WageDensity[x]**

$$\frac{6.38308 e^{-2 \cdot (\log(x)-3.85)^2}}{x}$$

<code>Total[f:Identity, b:0, e:∞]</code>	integrates f[x] with the WageDensity at x, for the interval of b till e. Identity gives the wage sum
<code>Total[Number, b:0, e:∞]</code>	gives the integral of the wage density, i.e. the total number of labour years. This could also be got by setting f = 1
<code>Total["Wage", b:0, e:∞]</code>	also computes the wage sum, but it is thought that Wage reads better than Identity

Use `SetOptions[NIntegrate, ...]` to adjust the computational parameters.

- Approximate Dutch labour supply in million labour years:

```
LabourSupply = Total[Number]
```

8.

- The OECD has computed Dutch unemployment as approximately 25%.

```
OnBenefit = Total[Number, 0, MinimumWage]
```

2.18422

```
OnBenefit / LabourSupply
```

0.273027

- A different tax structure would increase employment.

```
EmployableOnBenefit = Total[Number, Exemption, MinimumWage]
```

1.76372

- For the revenue we have to exclude the tax void.

```
LevySum = Total[Levy, MinimumWage]
```

169.427

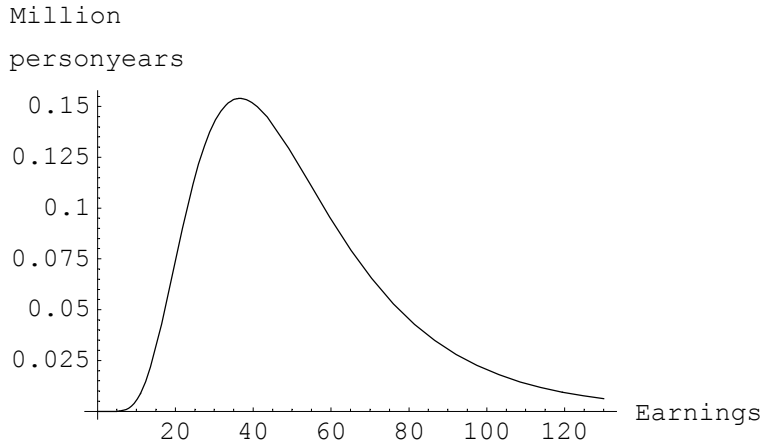
5.8.7 Tax Void Diagram

The levy plot and the wage density plot both have the wage on the x-axis, so we can combine the plots. This will give an instructive view of the effect of the tax void. Not all steps below will be printed.



- A plot of the wage density.

```
wdp = Plot[WageDensity[x], {x, 0, 130}, PlotRange → All,
AxesLabel → {"Earnings", "Million\personyears"}]
```



- Fill the wage density area between net and gross minimum wage.

```
mni = "MinimumNetIncome" /. Results[LevyPlot];
```

```
wdfill = FilledPlot[WageDensity[x], {x, mni, MinimumWage}, Fills →
Hue[.9], Axes → False]
```

- Show the wage density with that filled area. We have to get rid of a AxesFront option that has been defined by FilledPlot but that is not recognised by Show. Also, as explained below, we eliminate the axes.

```
wdna = Show[wdp, Axes → False];
```

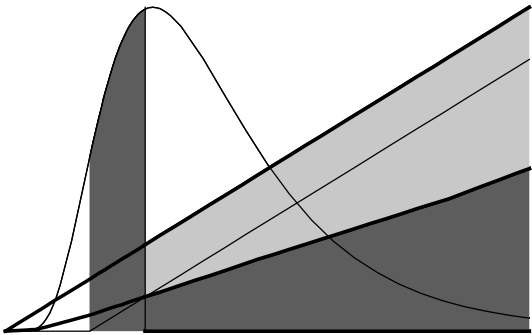
```
wd = Show[wdna, wdfill /. (AxesFront → True) :> ToSequence[]]
```

- We would like to have a plot with a vertical axis on the left for the wage density and a vertical axis on the right for the levy, but this is one feature that *Mathematica* doesn't allow for. We even have to delete all axes in the wage density plot and the levy plot, since axes affect the plotting proportions.

```
lp2 = Show[lp, Axes → False]
```

- The pink area shows the labour years affected by the tax void between the net and gross minimum wages. Since the wage density is skewed, the impact of the tax void is disproportional. If exemption were chosen at the net minimum, the void could disappear. Such a policy would not cost anything, since there are no revenues in this area.

```
Show[GraphicsArray[{wd, lp2, wdna}, GraphicsSpacing -> -1]]
```



5.8.8 Indexation of rates

Taxes are indexed by default as PieceWiseLinear[y, AdjustRates[TaxRates, ti]]

AdjustRates[rates, x]	for rates in TaxRates format, multiplies the tax brackets with factor x
Indexed[f][y, ti]	is f[y] (Tax, Premium) with the rates adjusted by ti

- If the tax brackets are adjusted for p % inflation:

```
AdjustRates[TaxRates, 1 + 0.01 p]
```

$$\begin{pmatrix} 7.102 (0.01 p + 1) & 0.373 \\ 45.96 (0.01 p + 1) & 0.5 \\ 97.422 (0.01 p + 1) & 0.6 \end{pmatrix}$$

There is differential indexation when levies are indexed with PI and wages are indexed with WI. The common situation in OECD countries is that taxes are adjusted for inflation only, while wages are adjusted for prices and productivity or the real rise in welfare, so that  $WI = PI * RWI$ . Note the two effects:

- When subsistence rises with WI and statutory tax exemption with PI, then the tax void grows. In fact, this differential indexation has been a major cause in the creation of the tax void.
- With tax adjustments, we need the *dynamic* marginal tax rates to see the effect on incentives.

### 5.8.9 The Dynamic Marginal Rate

```
DynamicMarginal[w, wi, pri:1, ti:Automatic]
```

gives the d.m.r.  $(\text{newlevy} - \text{Levy}[w]) / (\text{newwage} - w)$ ,  
when wages are indexed with  $w_i$ , premium with  $pri$ ,  
and tax with  $ti$ . The indexed levy is computed using  
`Indexed[...]`. Automatically  $ti$  is taken equal to  $pri$

```
DynamicMarginal[All, w, wi, pri, ti:Automatic]
```

generates more information on the above

When tax brackets are indexed on the wage, then the dynamic marginal equals the average tax burden. This point is developed in the example notebook `DynamicMarginal.nb`. Note that econometric research on the impact of tax rates indeed suggests that average rates are more important than statutory marginal rates.

### 5.8.9 Optimal taxation

The theory of optimal taxation suggests that tax distortions are least when the most productive members of society have the lowest marginal tax disincentives. This point is not developed here, in particular since there are also points of equity to consider that would make for a new programming exercise.

A question of optimality that is in line with the existing routines and that thus can be handled directly is the following. Can one determine a new levy schedule, such that the tax void is abolished, while at least the same revenue is generated? In solving this, we can use the fact that the unemployed now cost benefits, so that we only need to look at 'net revenue'. We could add the condition that the top marginal rate would not increase, given the notion that average rates are more important anyway (see above). The various elements in this topic are discussed in the notebook `OptimalTax.nb`.

### 5.8.10 Curved tax

Tax simplification is a popular subject. A suggestion made here is to replace the piecewise linear tax function by a curved tax with only three parameters. Exemption  $x$  would be determined by the labour market. The limit value of the top marginal rate  $r$  would follow from equity considerations. The third parameter  $c$  would follow from the revenue requirements, and would determine the curvature. The political debate about tax brackets would be killed, while on the balanced growth path both  $x$  and  $c$  would be indexed on the average wage.

<code>CurvedTax[y, x:7.102, r:0.6, c:31]</code>	gives a nonlinear curved tax relation for income y, exemption x, limit value r of the marginal rate, and curve parameter c.
<code>PieceWiseCurved[y, rates]</code>	for rates that are triplets, $\{\{x_1, r_1, c_1\}, \{x_2, r_2, c_2\}, \dots\}$ where $x_i$ is the point where the limit slope (marginal rate) $r_i$ and curvature $c_i$ start to be effective. $x_1$ is proper exemption
<code>CurvedTaxRates</code>	example input for <code>PieceWiseCurved</code>
<code>AdjustRates[rates, x, y]</code>	inflation adjustment for rates in <code>CurvedTaxRates</code> format

- The relation is  $r(y - x) y / (y + c)$ . The Bentham tax is  $r(y - x)$ , and  $y / (y + c)$  adds curvature. A positive  $c$  gives a convex function, a negative  $c$  a concave function. Maximum conditions are applied to guarantee useful output.

**CurvedTax[y, x, r, c]**

$$\frac{r y \text{Max}[0, y - x]}{\text{Max}[0.00001, c + y]}$$

5.8.11 Utilities

5.8.11.1 Additive rates

A frequent situation is that one has to pay various premiums, with different rate structures. If these premiums are all on the same base, e.g. gross income, then they can be added. The total can be represented with a new rate structure.

<code>AddLinearRates[r1, r2, ...]</code>	where rates $r_i$ are in the format of <code>PieceWiseLinear</code>
--	---

- For the Dutch situation some brackets are determined by daily earnings for a certain period. For example, disability premiums (wao) are based on 200 days of work, with a range of DFL 102 till DFL 294.42 per day. The national health insurance (zfw) is topped at DFL 60,750 per annum (and then private insurance kicks in). There are also premiums for unemployment (awf) and early retirement (vut). The various Dutch premiums can be added as follows:

```
wao = {{.102 * 200, 0.0845}, {.29442 200, 0}};  
  
awf = {{.102 * 200, 0.0505 + .022}, {.29442 200, 0}};  
  
vut = {{0, 0.019 + 0.0095}};
```

```
zfw = {{0, .0555 + .0135}, {60.750, 0}};

prem = AddLinearRates[wao, awf, vut, zfw]
```

### 5.8.11.2 Tax groups

The government may distinguish different tax groups. Suppose that group 1 (workers) may subtract a 10% of their income, with a minimum of DFL .2 and a maximum of DFL 2. thousand.

TaxDeduction	reserved option, can be implemented by the user
--------------	---

```
rates = {{0, 0.1}, {20., 0}};

TaxDeduction[y_] = Max[.2, PieceWiseLinear[y, rates]]

Max[0.2, If[y ≤ 0, 0, If[y ≤ 20., 0.1 y, If[y ≤ ∞, 2., Null]]]]

Tax[y_, group1] := Tax[y - TaxDeduction[y]]
```

### 5.8.11.3 Pen's Parade

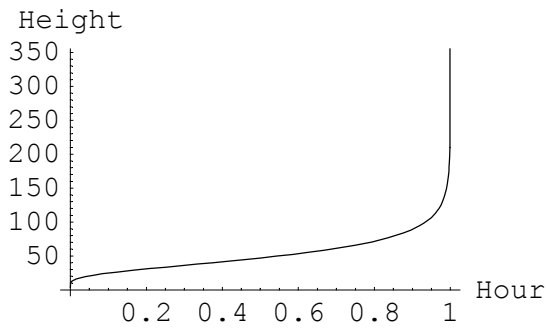
Professor Jan Pen invented a presentation of the income distribution that better conveys its meaning. Think of the population passing by in a parade of one hour, all ordered by the height of their incomes. Even better, let the people be as tall as their income, so that the Parade would start with dwarves. For a common income distribution you have to wait 40 minutes before you encounter people of medium income (height).

ToPensParade[ <i>density</i> , <i>p</i> ]	translates an income density (with integral value of 1) into Pen's Parade with proportion <i>p</i>
---	---

Pen's Parade is the inverse of the integral of the density. Variable *p* is a proportion of the population, and ToPensParade determines the income level associated with it.

```
income[prop_] = ToPensParade[WageDensity[#, 3.85, .5, 1]&, prop];
```

```
Plot[income[prop], {prop, 0, 1}, TextStyle -> {FontSize -> 11},  
AxesLabel -> {"Hour", "Height"}];
```



**Note**

At the close of writing the documentation for this version of the Pack, I included a new package on income inequality measures. Its documentation can be found as `supplementary notebooks` . The package's contents are as follows.

```
Economics[Economic`Inequality]  
  
FromWageDensity  Options$Theil      ToPensParade  
Gini              PensParade       Total  
Lorenz            PensParadePlot   WageDensity  
LorenzPlot        ResetWageDensity WageDistribution  
NLorenz           Theil
```

## 5.9 Game theory and decision theory

---

### 5.9.1 Summary

This implements basics of game theory and decision theory, for two player zero-sum games.

`Economics [Decision]`

`Economics [Minimax]`

### 5.9.2 Introduction

Game theory is basic to economics. It provides a concept for general equilibrium, where agents act under expectations about the acts of others. It is important for voting where agents may vote strategically (sometimes called cheating). It is relevant when the partners in a cartel have to divide the loot. It is relevant for statistical decision making in practical decision support (see below). There are numerous examples as these.

John Dickhaut and Todd Kaplan already made a game theory package, see their "A program for finding Nash equilibria", in Hal Varian (ed), "Economic and financial modeling with *Mathematica*", Springer-Telos 1993. Michael Carter, "Cooperative Games", in the same volume, gives an elegant presentation of the Shapley Value. Their packages can be downloaded from *MathSource*. The game theory packages discussed here have been developed independently, with the decision approach to statistics as explained by Thomas S. Ferguson, "Mathematical statistics", AP 1967, as the inspiration. The packages have been extended now with routines to link to the Dickhaut and Kaplan package.

We regard only two-player zero-sum games. We will use a  $2 \times 2$  example that comes with the `Decision`` package, but the problem can be  $n \times m$ .

### 5.9.3 Pay-off and Loss

The game is represented mathematically by a pay-off table. Your opponent controls the rows (horizontal player), and all positive entries are his or her earnings. You control the columns (vertical player), and the pay-offs are your losses. The package uses "PayOff" for the whole matrix, and "Loss" for elements therein, to indicate the change of perspective, without changing plus/minus signs.

NB: This Loss is taken general and need not be nonnegative. The expected value of this generalised loss is called "Due". Compare this with *prospects*, where we take Loss to stand for nonnegative values only. Here, with *pay-offs*, we have generalised loss that can be negative too (representing a profit). The quite standard statistical decision theory (e.g. described by Ferguson (1967)) uses the term "risk" for this expected generalised loss. It turns out that this use of the term "risk" is unfortunate, and, as said, we will be using "Due".

PayOffTable	array of Loss, with symbolic or numeric values
PayOffToRule [ <i>matrix</i> ]	translates a payoff matrix into a pay-off object (list of rules)
PayOff	option PayOff → <i>matrix</i> occurs in a pay-off object

Note: The list of rules describing a 2 player zero-sum game is called a "pay-off object" here.

**PayOffTable**

Array[Loss, {NumberOfStates, NumberOfActions}]

In *Mathematica* the game situation shall be represented by a 'game object' or 'pay-off object' that not only contains the pay-off table but also some other information that comes in handy. For example, you may want to include (formal) probability distributions with the players.

OddOrEven	example of a pay-off object. Two players show 1 or 2 fingers. The pay off is determined by the sum, you cash even values, while odd values go to your opponent
-----------	--

- In this game, you cash even values while odd values go to your opponent.

**OddOrEven**

{NumberOfStates → 2, NumberOfActions → 2,  
PayOff →  $\begin{pmatrix} -2 & 3 \\ 3 & -4 \end{pmatrix}$ , ColumnPlayer → {}, RowPlayer → {}}

**5.9.4 LookAt**

LookAt [ <i>item</i> , <i>payoff object</i> ]	shows properties of item. See ??LookAt
---	--

- The head Act denote the actions that you can take, the head State those of your opponent (that you don't have influence on).

**LookAt[Rule, OddOrEven]**

	Act(1)	Act(2)
State(1)	-2	3
State(2)	3	-4



### 5.9.5 Regret

Notice that the pay-off table is not sacrosanct. It is just one representation of the game situation, and there may be transformations that are of more interest to the players. For example, *regret* for the vertical player is defined as his relative loss compared to what would have been his best choice had he known the choice of the horizontal player. To determine this regret, one subtracts the minimal value in a row from the whole row.

**Regret [OddOrEven]**

$$\begin{pmatrix} 0 & 5 \\ 7 & 0 \end{pmatrix}$$

### 5.9.6 Minimax

The minimax strategy is relevant in a situation when there is no probability information.

- The minimax strategy of the vertical player is to minimise the maximal loss. Each column has a maximal loss, and you take the minimum of these over the columns. This strategy is found by `Minimax[table]`.
- The minimax strategy for the horizontal player has been implemented as `Maximin[table]`, i.e. maximising (over the rows) the minimum revenue (per row). (Try `Minimax[- Transpose[table]]`.)

<code>Minimax[table]</code>	gives the minimised maximal loss to the vertical player
<code>Maximin[table]</code>	gives the maximised minimal revenue to the horizontal player
<code>MinimaxSolution[table]</code>	compares the strategies of the vertical (minimax) and horizontal (maximin) players.

- For `OddOrEven`, your `Minimax` strategy could be either column (both with a maximum loss of 3) and the `maximin` strategy of your opponent is `State[1]`. `OddOrEven` has no solution such that both strategies result into a single equilibrium value.

**MinimaxSolution[PayOff /. OddOrEven]**

$$\left\{ \begin{array}{l} \text{MinimaxSolutionQ} \rightarrow \text{False}, \text{Value} \rightarrow \text{Indeterminate}, \text{Position} \rightarrow \{\}, \\ \left\{ \begin{array}{l} \text{Minimax} \rightarrow \left\{ \begin{array}{l} \text{Dimensions} \rightarrow \{2, 2\}, \text{Max} \rightarrow \{3, 3\}, \text{Value} \rightarrow 3, \text{Position} \rightarrow \left( \begin{array}{cc} 1 & 2 \\ 2 & 1 \end{array} \right) \end{array} \right\}, \\ \text{Maximin} \rightarrow \left\{ \begin{array}{l} \text{Dimensions} \rightarrow \{2, 2\}, \text{Min} \rightarrow \{-2, -4\}, \text{Value} \rightarrow -2, \text{Position} \rightarrow \left( \begin{array}{cc} 1 & 1 \end{array} \right) \end{array} \right\} \end{array} \right\} \end{array} \right\}$$

5.9.7 Mixed strategies and the value of the game

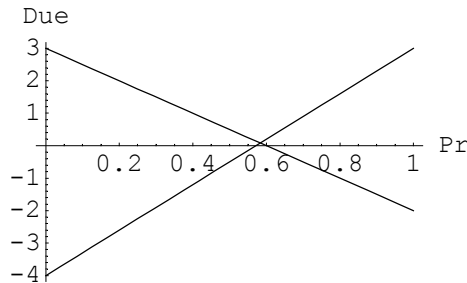
Mixed strategies arise when your opponent chooses his or her States with probability measure  $t$  and when you choose your Actions with probability measure  $p$ . The "Value of the game" is what your opponent may expect to win when both players select optimal mixed strategies. We use the term "Due" to stand for the expected value of your negative revenue. Note that the "value of the game" concept still excludes variance considerations.

<code>GameValue[table, t, p] or GameValue[table, {t, p}]</code>	
$t. table .p$ , if $p$ is a list, and $f[t. table .f[p]$ for $f[q] = PM[\{q, Rest\}]$ otherwise	
<code>Value2By2[table]</code>	determines the value of the game of the 2 by 2 case
<code>GamePlot[Due, table]</code>	makes a 2 D plot of the expected returns
<code>GamePlot[Value2By2, table]</code>	makes a 3 D plot of the value of the game

Note: Value2By2 gives one point and neglect multiple solutions.

- Write the pay-off table as  $\{\{a, b\}, \{c, d\}\}$ . Multiplying with probability measure  $\{t, 1 - t\}$  for the States, gives  $\{a t + c (1 - t), \{b t + d (1 - t)\}$ . This gives two lines that we can plot for  $0 \leq t \leq 1$ . Each line represents the expected loss for one action, depending upon  $t$ . The lines intersect at the "value of the game".

`GamePlot[Due, PayOff /. OddOrEven];`



- The "value of OddOrEven" is  $1/12$ , and it is found when `Act[1]` is chosen with probability  $7/12$  and `State[1]` is chosen with probability  $7/12$ .

`Value2By2[PayOff /. OddOrEven]`

$$\left\{ \frac{1}{12}, \left\{ \Pr[\text{State}[1]] \rightarrow \frac{7}{12}, \Pr[\text{Act}[1]] \rightarrow \frac{7}{12} \right\} \right\}$$

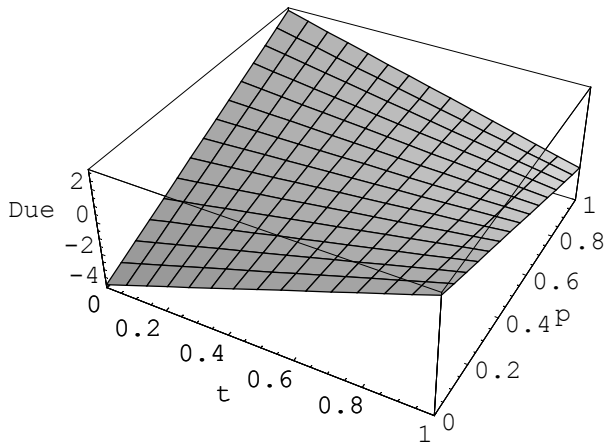
- If we pick arbitrary  $t$  for the states and  $p$  for the acts, then we get an expression for the value.

```
GameValue[PayOff /. OddOrEven, t, p]
```

$$p(3(1-t) - 2t) + (1-p)(3t - 4(1-t))$$

- If we plot the value, then we get a saddlepoint. Note: this 3D plot uses  $t = \text{Pr}[\text{State}[1]]$  and  $p = \text{Pr}[\text{Act}[1]]$ .

```
GamePlot[Value2By2, PayOff /. OddOrEven];
```



Note that the `Decision`` package does not allow you to solve `GameValue[table, p, q]` in general. It only provides `Value2By2` routine. For a general solution, see the `Dickhaut and Kaplan Nash`Nash`` package.

Note that in our visit to Prospects above, we already took the mixed strategy perspective. If we take the mixed strategy perspective, then there is also the consequence that we should take revenues to be probabilistic too.

- When we take the cross-diagonal elements in the prospects, in which the revenues of one player are dependent upon the probabilities chosen by the other, then we can again join those prospects, and get the following result. This indicates that the revenues can be considered to be probabilistic in the first place.

```
JoinIDProspects[Last[RowPlayer /. potps], First[ColumnPlayer /. potps]]
```

$$\text{JointProspect}\left(\left(\begin{pmatrix} 3-5t & 3-5t \\ 7t-4 & 7t-4 \end{pmatrix}, \begin{pmatrix} 5p-3 & 4-7p \\ 5p-3 & 4-7p \end{pmatrix}, \begin{pmatrix} pt & p(1-t) \\ (1-p)t & (1-p)(1-t) \end{pmatrix}\right)\right)$$

- Taking the expected values, we find that these are opposite, and give the value of the game.

```
JointProspectEV[%] // Simplify // Expand

{-12 t p + 7 p + 7 t - 4, 12 t p - 7 p - 7 t + 4}

First[%] == - Last[%]

True
```

5.9.8 Interfacing with the Nash`Nash` package

The Nash`Nash` package representation of a pay-off cell is {X, Y}, where player 1 receives X and player 2 receives Y. These games can be other than zero-sum. The pay-off table format above can be easily transformed into this. See also the example notebook on the Nash`Nash` package and the Nash[ ] routine therein.

PayOffRuleToGame [object]	translates pay-off X into {X, -X}
GameToPayOffRules [game]	splits the pay-off elements {X, -Y} into single pay-offs. Each player plays against a third party

- The result of the following can be submitted to the Nash[ ] routine.

```
OOE = PayOffRuleToGame [OddOrEven]

( {-2, 2}  {3, -3} )
( {3, -3}  {-4, 4} )
```

- The PayOff for your opponent is now presented as the negative transpose, since he or she now plays against a third player.

```
GameToPayOffRules [OOE]

{{PayOff -> ( 2  -3 )
              -3  4 }, NumberOfStates -> 2, NumberOfActions -> 2},
{PayOff -> ( -2  3 )
              3  -4 }, NumberOfStates -> 2, NumberOfActions -> 2}}
```

5.9.9 Minimal due and the value of perfect information

The following are relevant when you know something about the probabilities *p* for the States.

MinimalDue [ for pay-off object {r} using probabilities p for the states  
{ *r*\_\_\_Rule } , *p*\_\_List ]

`MinimalDue[table, p_List]` works directly on that table

- If you assign 70% probability that your opponent draws two fingers, select two fingers yourself:

```
MinimalDue[OddOrEven, {.3, .7}]
```

{Expectation  $\rightarrow \{1.5, -1.9\}$ , Position  $\rightarrow (2)$ , Min  $\rightarrow -1.9$ }

If you had perfect information about your opponent, then you could select the column with the minimal value. Currently you can only weigh these minima with the probabilities. The amount by which this rises above the normal minimal due is the value of perfect information.

EVPerfectInformation[*pay-off object*, *p\_List*]

the expected value of perfect information using probabilities  $p$  for the states

`EVPerfectInformation[payofftable, p_List]` works directly on that table

- If both states are equally likely, the value of perfect information (spying) is:

```
EVPerfectInformation[OddOrEven, {.5, .5}]
```

$$\{\text{MinimalDue} \rightarrow \{\text{Expectation} \rightarrow \{0.5, -0.5\}, \text{Position} \rightarrow (2), \text{Min} \rightarrow -0.5\}, \\ \text{BestInRow} \rightarrow \{-2, -4\}, \text{Expectation}(\text{BestInRow}) \rightarrow -3., \text{Value} \rightarrow -2.5\}$$

### 5.9.10 Decision theory

Suppose that you can probe your opponent, and that you know that the answer is 75% reliable. This is comparable to taking a sample with a presumed error.

These assumptions change the character of the problem. There now are decision rules that can link an estimate to a action. And each decision rule has an expected value, subject to the true choice of the opponent.

An example decision rule is to believe your opponent. Let his answer (the estimate) be  $Est[i]$  and your action be  $Act[j]$ . Believing your opponent, in `OddOrEven`, means  $Est[1] \rightarrow Act[1]$  (i.e. reap Loss -2) and  $Est[2] \rightarrow Act[2]$  (i.e. reap Loss -4). If the true choice will be `State[1]`, then  $Est[1]$  has a 75% chance, and then you reap a Loss of -2. If the true choice is `State[1]`, then  $Est[2]$  has a 25% chance of being said, and the combination of  $\{State[1], Act[2]\}$  gives you a Loss of 3. Thus, `State[1]` comes with a due of 75% (-2)

+ 25% (3) = -3/4. Similarly, the decision rule of believing your opponent comes in State[2] with a due of -2.25.

<code>Assign[<i>item</i>, <i>object</i>]</code>	assigns a numeric value to item; best: item = All
<code>Reset[]</code>	sets decision variables to structural form
<code>DT[<i>object</i>]</code>	retrieves the due table
<code>GamePlot[Hull, <i>table</i>]</code>	plots the convex hull in the {Due[1], Due[2]} dimensions

- Assign values to all relevant functions of the decision problem.

```
Assign[All, OddOrEven, ProbabilityMatrix -> {{3, 1}, {1, 3}}/ 4.];
```

- Assigning All has not assigned Dec[ ], which is useful for this LookAt.

```
LookAt[Due, OddOrEven]
```

	Dec(1)	Dec(2)	Dec(3)	Dec(4)
State(1)	-2.	-0.75	1.75	3.
State(2)	3.	-2.25	1.25	-4.

- Now, regard the possible decisions. Dec[1] and Dec[4] neglect the information in the estimate. Dec[2] is to believe the estimate, and Dec[3] is to do just the opposite.

```
Assign[Dec, OddOrEven]
```

```
??Dec
```

Head of decisions

```
Dec[1] = {Est[1] -> Act[1], Est[2] -> Act[1]}
```

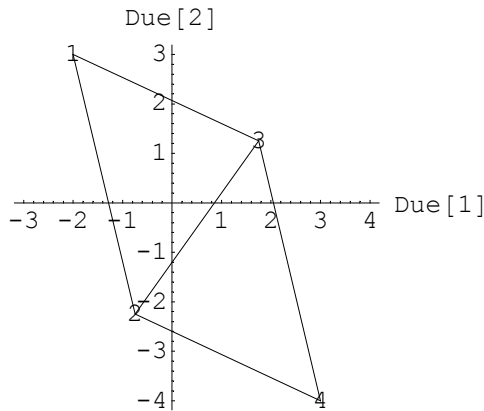
```
Dec[2] = {Est[1] -> Act[1], Est[2] -> Act[2]}
```

```
Dec[3] = {Est[1] -> Act[2], Est[2] -> Act[1]}
```

```
Dec[4] = {Est[1] -> Act[2], Est[2] -> Act[2]}
```

- We can plot the due table in the  $\{\text{Due}[1], \text{Due}[2]\}$  space. If we include all mixed strategies, then we get the convex hull.

```
GamePlot[Hull, DT[OddOrEven]];
```



A rational agent will aspire minimal due. For above convex hull that means that all points on the lower and left side will be taken. For any other point there is a mixed strategy that dominates that point. In fact, we may eliminate all such dominated decision rules in the due table.

```
LE[Transpose, DT[OddOrEven]]
```

$$\begin{pmatrix} 3. & -0.75 & -2. \\ -4. & -2.25 & 3. \end{pmatrix}$$

- Using the minimax due criterion gives that decision rule 2 is best.

```
Minimax[%]
```

```
{Dimensions → {2, 3}, Max → {3., -0.75, 3.}, Value → -0.75, Position → {1 2}}
```

### 5.9.11 Relation to prospects and decision trees

It is instructive to compare the pay-off object with the Prospect object discussed below in relation to risk. In single prospects, we regard only the outcomes for a single player dependent upon a single probability density. In the JointProspects we however allow for interaction. Clearly, the game situation is one of interaction.

- Transforming a pay-off object into a JointProspect gives us the proper revenue matrices (positive values pertaining to the player himself or herself) and the joint probability density accross all states. Clearly the probabilities taken by the players are independently distributed. With  $t$  for your opponent and  $p$  for you:

```
potjp = PayOffToJointProspect[OddOrEven, {t, 1 - t}, {p, 1 - p}]
```

$$\text{JointProspect}\left(\begin{pmatrix} -2 & 3 \\ 3 & -4 \end{pmatrix}, \begin{pmatrix} 2 & -3 \\ -3 & 4 \end{pmatrix}, \begin{pmatrix} pt & (1-p)t \\ p(1-t) & (1-p)(1-t) \end{pmatrix}\right)$$

- If we instead would try for single Prospects, then we have to choose which probability we want to make the revenues dependent on. Since we don't want to decide for you, you get both results.

```
potps = PayOffToProspects[OddOrEven, {t, 1 - t}, {p, 1 - p}]
```

```
RowPlayer → {Prospect(3 - 5 p, 7 p - 4, t), Prospect(3 - 5 t, 7 t - 4, p)},  
ColumnPlayer → {Prospect(5 p - 3, 4 - 7 p, t), Prospect(5 t - 3, 4 - 7 t, p)}
```

It is also useful to note that decision situations can be more complicated, with decisions conditional upon other decisions and probabilistic events. These cases could be handled by making larger matrices. A slightly different approach is the following.

<pre>DecisionTree[   {r__Rule}, f:Max]</pre>	<p>where one rule has to have the format <math>\text{Begin} \rightarrow (\dots)</math>. The tree is reduced by repeated replacement of the rules onto <math>\text{Begin}</math>, and then <math>\text{DecisionTree}[\text{expr}, f]</math> is called</p>
<pre>DecisionTree[expr, f:Max]</pre>	<p>solves for the <math>f</math> solution and the path leading to it</p>

Note: The rhs of a rule must be  $\text{Dec}[e_1, \dots, e_n]$  where  $e_i$  can be a single element or a list, or a value or a Prospect. See the example below. Note: Prospects are replaced by their expected value. If you want another evaluation, such as expected utility, then do such replacement before calling this routine. Note: Other results are in  $\text{Results}[\text{DecisionTree}]$ .

In this routine, the decision is basically made by replacement  $\text{Dec}[e_1, \dots, e_n] \rightarrow f$ , where the  $e_i$  have been selected as the single elements or the last element of lists  $\{\text{label}i_1, \dots, \text{label}m_i, e_i\}$ , depending upon whether single elements or lists have been given or computed. Note that the path commonly can only be indicated if such labels have been provided.

- This reproduces an example of Krajewski and Ritzman (1996:83). A company can expand to a large or a small facility. Demand can turn out to be high (probability 60%) or low (40%). If one expands to a small facility, then expanding later on will be relatively costly. If one expand to a large facility and demand turns out to be low, one might opt for additional advertisement, but the effect of this is probabilistic again.

```
data = {Begin → Dec[{Large, e}, {Small, s}],  
e → Prospect[800, LowDemand, .6],  
LowDemand → Dec[{NotAdvertise, 40}, {Advertise, adv}],  
adv → Prospect[220, 20, .7],  
s → Prospect[Later, 200, .6],  
Later → Dec[{ExpandLater, 270}, {NeverExpand, 223}]};
```



- The proper tree arises from repeated substitution of all rules.

**Begin // . data**

```
Dec({Large, Prospect(800, Dec({NotAdvertise, 40}, {Advertise, Prospect(220, 20, 0.7)}), 0.6)},
    {Small, Prospect(Dec({ExpandLater, 270}, {NeverExpand, 223}), 200, 0.6)})
```

- The path of maximal expected revenue is to expand to the large facility and eventually advertise.

**DecisionTree[data]**

```
{Max → 544., Path → {{Large, 544.}, {Large, Advertise, 160.}}}
```

- These are the other results computed by the routine.

**Results[DecisionTree]**

```
{{Large, 544.}, {Small, 242.}, {Large, Advertise, 160.},
    {Large, NotAdvertise, 40}, {Small, ExpandLater, 270}, {Small, NeverExpand, 223}}
```

# 5.10 Monopoly and cartel

## 5.10.1 Summary

This provides for the basic models of monopoly and cartel formation.

Economics[Cartel]

## 5.10.2 Introduction

There is a monopoly when there is only one supplier. There is a cartel when there are more suppliers who work together as a monopoly. The cartel has the advantage that there is a single demand function, like in the monopoly. It however has two main problems: first to allocate production to the different partners, and secondly to allocate aggregate profits to the partners.

## 5.10.3 Monopoly

The simple monopoly model is as follows. Let demand be  $q[p] = A p^e$ , where  $e$  is the price elasticity of demand  $\frac{\partial \text{Log}(q)}{\partial \text{Log}(p)}$ . With  $F$  the fixed costs and  $c$  the marginal costs, total costs are  $\text{Cost}[q] = F + c q$ . We can derive profits as:

$$\text{Profit}[p] = p q - \text{Cost}[q] = A p^e - F - c (A p^e) = A (p - c) p^e - F$$

The first order condition for maximal profits, that marginal profit is zero, gives  $p^* = c / (1 + 1 / e)$ .

In a complexer model we may take the `LinExp[]` function for demand and allow polynomial  $\text{Cost}[q] = c_0 + c_1 q + c_2 q^2 + \dots$ . In this case there may be no analytical solution, but we can use numerics.

`MonopolyModel` allows you to enter your own demand and cost functions. After making these substitutes, it releases the `Hold` on `D`, determines the marginal cost and the price elasticity, and then makes other substitutions.

<code>MonopolyModel[{rules},{elims},opts]</code>	is a <code>SolveFrom</code> application
<code>MonopolyEquations[]</code>	are called by <code>MonopolyModel</code>

You set the demand function by the rule `Demand → ...`, where your function must depend on `Price`. The default is `Demand → LinExp[Price, Exponent]`, and if you use this (i.e. if you leave out a `Demand → ..` rule), then the rule `Exponent → {A, a, pstar, h}` sets the values for the `LinExp` function. You set the cost function by the option `$Cost → ...`, where your function must depend on `Production`. The default is polynomial cost, and `CoefficientList → {c0, c1, c2, ...}` sets the coefficients. Set other values as you please. This provides a flexible format that allows one to e.g. solve for arbitrary coefficients such as `A`, `a`, ... given the level of `Production` and `Price`.

- Using the default demand and cost functions.

```
MonopolyModel[{Exponent → {A, -2, 1, 0},
  CoefficientList → {c0, 0, c2},
  Price → 850, Production → 104 150,
  Profit → 2 10^6}] // Last
```

$$\left\{ \left\{ c0 \rightarrow 7945000, A \rightarrow 11271000000, c2 \rightarrow \frac{17}{1248}, \right. \right. \\ \left. \left. \text{Cost} \rightarrow 11260000, \text{MarginalCost} \rightarrow 425, \text{Elasticity} \rightarrow -2 \right\} \right\}$$

- Taking a linear demand function.

```
MonopolyModel[{Demand → 350 - 2 Price,
  CoefficientList → {2000., 3, 2}}] // Last
```

```
{{Profit → 958.4, Cost → 4469.92, MarginalCost → 140.6,
  Price → 157.8, Elasticity → -9.17442, Production → 34.4}}
```

The package also contains a function `SetMonopoly[ ]` to set the demand and cost functions, and `Monopoly[ ]` to recover the various results. This format however is unidirectional and less flexible than `MonopolyModel[ ]`. You are referred here to "?".

5.10.4 Cartel

For the cartel, let us suppose for this introduction that there are two partners, with cost functions  $Cost_1[q_1] = F_1 + c_1 q_1$  and  $Cost_2[q_2] = F_2 + c_2 q_2$ . Total output will be  $q = q_1 + q_2$ .

First of all, minimal cost could be achieved by producing at only one facility, giving  $\text{Min}[Cost_1[q], Cost_2[q]]$ . However, the linear cost function is often only an approximation, and really increasing production at one site would affect fixed costs. In addition, it often happens that both partners still want to remain in business, e.g. when national airlines are cooperating. In these cases the different cost functions can be added, giving  $Cost[q] = F_1 + c_1 q_1 + F_2 + c_2 (q - q_1)$ . The condition for an optimum now is that one should produce at each facility at the same marginal cost. The simple linear cost functions still provide a problem, since equalising marginal costs gives  $c_1 = c_2$ , and this will be the case only seldomly. When at least one of the costs is quadratic, then marginal costs can be set equal at all facilities, and an analytical solution can be found. Note however that then average costs will still differ, and hence profits per partner. So the partners will have to agree on a distribution clause. Also, the production levels generated by the solution may still not be agreeable, and there may be additional conditions in practice.

<code>SetCartel[kind, {A, e (, pstar, h)}, {parms__List}, x0:80, x1:3000]</code>	sets up and solves the cartel problem
<code>Cartel[item]</code>	gives <i>item</i> results at the optimal allocation
<code>Partner[i][Cost, qi]</code>	gives the cost of partner i in producing qi

SetCartel does the following tasks:

1. It sets the demand function to `LinExp[p, {A, e, pstar, h}]`, see `FixedLinExp`; if called with only `A` and `e`, then `pstar = 1` and `h = 0`.
2. It defines the cost functions of the different partners, where each parameter list is `{c0, c1, c2, ...}` for polynomial  $cost[i] = c_0 + c_1 q[i] + c_2 q[i]^2 + \dots$
3. It solves for the optimal price and derives the allocation of production over the partners. Here, `x0` and `x1` are the range values for `FindRoot`. When `kind = Plus`, then the cost functions are added, and each produces at the same marginal cost; when `kind = Min` then the minimal cost producer is selected (and the others are zero).
4. It sets the different values for `Cartel[...]`.

```
SetCartel[Plus, {10^7, -1.5, 1, 0},  
           {{90000, 400, 1}, {60000, 500, .75}}]
```

```
{Price → 1729.07, Production → 139.085, Cost → 220444.,  
  MarginalCost → 576.358, Profit → 20044.3, Partner(N) → 2,  
  Partner(1) → {Production → 88.1791, Cost → 133047., MarginalCost → 576.358, Profit → 19421.1},  
  Partner(2) → {Production → 50.9055, Cost → 87396.3, MarginalCost → 576.358, Profit → 623.18}}
```

<code>Shares [criterion, c_List]</code>	reallocates profits of <code>c = Cartel[]</code> formats according to the given criterion
<code>Shares [c_List]</code>	transforms <code>c = Cartel[]</code> formats into a format with shares per partner

Possible criteria are:

- Equal: the shares are taken as  $1 / n$ , with  $n$  the number of partners
- {share1, share2, ...}: a plain list of shares (if not adding up to 1, then it is divided by the sum of these shares)
- Cost: take the shares in the total cost
- Production: take the shares in production

Output is: Profit → the total profit that is to be divided, Rule → the division rule, Share → the shares when applying that rule, In → the profits earned by every partner by his sales, Out → profits as they should be after redistribution, Re → the actual redistribution (Out - In). A positive value is a receipt from the clearing agent, a negative value is a payment to the clearing agent.

**Shares [Equal, %]**

{Profit → 20044.3, Rule → Equal, Share →  $\left\{\frac{1}{2}, \frac{1}{2}\right\}$ ,  
In → {19421.1, 623.18}, Out → {10022.2, 10022.2}, Re → {-9398.97, 9398.97}}

**Shares [%%]**

{Production → {0.633996, 0.366004}, Cost → {0.603543, 0.396457},  
MarginalCost → {0.5, 0.5}, Profit → {0.96891, 0.0310901}}

# 5.11 Peak load pricing

## 5.11.1 Summary

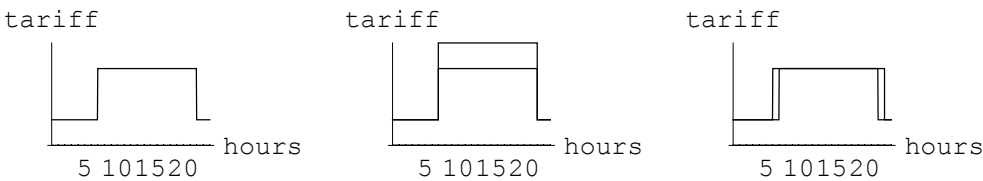
This implements the basic peak load pricing model.

```
Economics[PeakLoad]
```

## 5.11.2 Introduction

A peak load problem arises when services during a particular usage period ('day') are much higher than at other periods ('night') so that capital utilisation is not evenly spread. A general solution is to have high tariffs in the peak load period and low tariffs in the low period, so that users are incited to move from the peak load period to the low period. A condition for this to work is that the good cannot be easily stored, like electricity. Then there are still two questions: (1) how high shall the peak load tariff be, and (2) how long will that tariff be used ? The answer to this depends upon the question whether the operator is a private profit maximiser or a welfare maximising public utility. Note that this problem can be applied to any kind of usage class distinction, as long as the conditions of discrimination and storage apply. We will keep using the language of electricity day and night tariffs though.

```
LowHighTariffPlot[5, {10, {7, 22}}, {15, {7, 22}}, {10, {6, 23}}, {t, 0, 24}, AxesLabel -> {"hours", "tariff"}, PlotRange -> {0, 20}, TextStyle -> {FontSize -> 10}]
```



The following discussion relies on A. Takayama, "Mathematical economics", The Dryden Press 1974. The problem is simplified by letting exogenous circumstances determine the duration of the peak load tariff.

## 5.11.3 Main routine

The user should supply: (1) the inverse demand functions for the 'day' and 'night' prices:  $pd[y]$  and  $pn[y]$ , (2) the fraction of the 'day' period,  $fd$ , and (3) additional cost information.

`PeakLoad[kind, fd, {pd, pn}, {v, b, rpk} (, Profit)]`

gives the optimal output and capital stock as function of the time share of pd,  
for kind = All, Part & for welfare or profit maximisation.

#### **PeakLoadLegend**

y = output = service = sales

p = output price, p(y) is the inverse demand function

pd = first submarket price ('day rate') =>  $pd = pd(y)$

pn = second submarket price ('night rate')=>  $pn = pn(y)$

eps = price elasticity of demand,  $eps < -1$

K = capital stock

b = capital per unit of output

rpk = unit capital cost, or user cost per unit of capital

q = shadow price of capital

v = variable costs per unit of output

T = time period

f = fraction of T

All = operating at peak load for all t

Part = operating at peak load for part t only

– (none) = the objective is welfare maximisation, and  
the epsd in pd(y) and epsn in pn(y) may differ,  
but we substitute  $eps \rightarrow -\text{Infinity}$  in the routines

Profit = the objective is profit maximisation, and this applies  
only for the following conditions:

full capacity: constant & equal price elasticities  $epsd = epsn$

below capacity:  $yn < yd$ , and constant price elasticity for yd

#### **5.11.4 Subroutines**

The main routine calls the subroutines `PeakLoadCapital` and `ImpliedShares`, and the first uses the `PeakLoadEquations` for the solution.

`PeakLoadCaptital [Point, kind,  
fd, {pd, pn}, {v, b, rpk, eps}, val:10, opts__Rule]`

uses FindRoot to determine the implied size  
of the capital stock as a function of the fraction of time for pd.  
Kind = All or Part. Note: if fd = {f1, f2} then one gets a range of K

`PeakLoadEquation[kind, K, fd, {pd, pn}, {v, b, rpk, eps}]`

gives the crucial condition at the optimum, for kind = All, Part.  
If eps = -Infinity then there is welfare maximisation. If eps is entered as {eps},  
then the functional value eps[K/b] is taken

`ImpliedShares[kind, K, {pd, pn}, {v, b, rpk} (,Profit)]`

gives the implied time shares of d and n at given level K,  
for kind = All, Part & for under welfare or profit maximisation.

`PeakLoadEquation[Part, K, f, {pd, pn}, {v, b, rpk, eps}]`

$$pd\left(\frac{K}{b}\right) == \frac{b \text{ rpk}}{f} + \frac{v}{1 + \frac{1}{\text{eps}}}$$

`PeakLoadEquation[All, K, f, {pd, pn}, {v, b, rpk, eps}]`

$$f \text{ pd}\left(\frac{K}{b}\right) + (1 - f) \text{ pn}\left(\frac{K}{b}\right) == \frac{b \text{ rpk} + v}{1 + \frac{1}{\text{eps}}}$$

`PeakLoadEquation[All, K, f, {pd, pn}, {v, b, rpk, {eps}}]`

$$f \text{ pd}\left(\frac{K}{b}\right) + (1 - f) \text{ pn}\left(\frac{K}{b}\right) == \frac{b \text{ rpk} + v}{1 + \frac{1}{\text{eps}\left(\frac{K}{b}\right)}}$$



### 5.11.5 Plotting

```
PeakLoadPlot[All, f, {pd, pn}, y*, {ymin, ymax}, labels_List: {}, opts]
```

gives a plot of the peak load problem at full capacity,  
with  $f$  the fraction for  $pd[]$ , and  $y^*$  the optimal point

```
PeakLoadPlot[Part, f, {pd, pn}, {y*, z*}, {ymin, ymax}, labels_List: {}, opts]
```

gives a plot of the peak load problem at part capacity, with  $f$  the fraction for  $pd[]$ ,  
 $y^*$  the optimal point for  $pd[]$ , and  $z^*$  the optimal point for  $pn[]$ .

These plots reproduce Takayama's diagrams. See the example notebook `PeakLoad.nb`.

## 5.12 Introduction to finance

---

### 5.12.1 Summary

The `Finance`Common`` package provides some basic finance routines like for accounting, and sets a working environment by also loading `Time`` and `ShowPlan``.

```
Economics["Finance`Common`"]
```

### 5.12.2 Introduction

Finance is an inviting subject for *Mathematica*. Wolfram Research Inc. (WRI) has developed the Finance Essentials package that provides a basic working environment for financial analysis with *Mathematica*. The books edited by Hal Varian contain various finance chapters. It is perhaps only natural that there still appeared to be some room for enhancement. The Economics Pack has the following additions:

- The `Finance`Common`` package provides for common concepts and accounting basics.
- The `FlatRate`` package provides for an introduction in finance concepts, using a flat rate of interest. This package can also be used to check on results when complexer formats collapse into simple ones.
- The `Finance`CAPM`` package develops main points of the Capital Asset Pricing Model.
- The `Finance`` package calls all finance packages of the Economics Pack and those of the Finance Essentials. The Finance Essentials example notebooks show a number of object manipulations that are so useful that they may as well be put into standard routines - as is done in the `Finance`` package.

There are also the following supplements:

- Finance needs dating of financial instruments. The `Time`` package extends the time and date routines.
- The `ShowPlan`` package allows the plot of time lines.
- The `Risk`` package develops the concept of risk.
- The `Databank`` package allows the structured use of accounting data.
- The `Concepts.nb` exemplary notebook explains financial concepts and terms. This for example clarifies that the `Bond[ ]` object in the Finance Essentials package is a bullit. A swap construction using flat rates is discussed in `Swap.nb`.

Note that most packages mentioned here can run without the Finance Essentials package. Only the `Finance`` package is used in combination with that package.

5.12.3 Symbols only

The following are symbols only, though can be redefined by a higher application.

Account	EBIT	Premium
AcidTest	Equity	PriceToEarnings
Asset	Income	RetainedIncome
Bond	Interest	ReturnOnAssets
Cash	Investment	ReturnOnEquity
CashFlow	Leverage	ReturnOnSales
CurrentRatio	Liability	Stock
Depreciation	LIBOR	Tax
Dividend	Market	TimesInterestEarned
EarningPerShare	NetWorth	WorthPerShare
Ecart	NumberOfShares	

Asset would be used for cash, bonds and stock equity. Income is the change of wealth; where Hicks's definition (what you can spend during a period while ending up with the same wealth as at the beginning) might be limited to nonnegative values. The EBIT are Earnings Before Interest and Tax (or operating income). NetWorth is the difference between assets (what one owns, credit, right side of the balance sheet) and liabilities (what one owes, debit, left side of the balance sheet). This would be the Equity value if it concerns a common stock company. LIBOR is the London Interbank Offered Rate, at which banks offer to lend funds in the international interbank market. Ecart is a (French) term for the difference between two rates; many use the term 'spread', but a spread is a standard deviation.

5.12.4 Rate of return

As simple as the following is, it still can be useful for operations on lists or for legible statements in routines.

RateOfReturn[ <i>begin, end, duration</i> ]	gives the rate of return when \$ begin is invested for duration, and earns \$ end
--	--

**RateOfReturn[in, out, n]**

$$\left(\frac{\text{out}}{\text{in}}\right)^{\frac{1}{n}} - 1$$

5.12.5 Interest rates

<code>AnnualToDayRate[r, opts]</code>	changes an annual interest rate <i>r</i> into a day rate
<code>DayToAnnualRate[r, opts]</code>	changes a day rate into an annual rate

Default option setting `Count` → 360 takes 360 days per annum; alternatives are 365, 366, or a year, like 1990 (with account of leap years). Default option `Factor` → 10000 gives the result as base points (percents of percents) per day. If the switch of rates per one (years) to rates per 10000 (days) is confusing, set `Factor` → 1.

**AnnualToDayRate[r]**

$$10000 \cdot (\sqrt[360]{r + 1} - 1)$$

**DayToAnnualRate[r, Count → 2000]**

$$(0.0001 \cdot r + 1)^{366} - 1$$

Many trades use a day rate for specific periods.

<code>DayToDayInterestFactor[begindate, enddate, rate, opts]</code>
gives the compounded interest factor from one day to another. The input rate is assumed to be annual, and the day rate is computed by <code>AnnualToDayRate[rate, opts]</code>

`DayToDayInterestFactor[F[March, 2, 1999], F[April, 5, 1999], .1]`  
1.00904

5.12.6 Portfolio object

The following gives a finance object to collect other finance objects. The prime use of this is in applications of a higher level, that can exploit the common structure.

**Portfolio** Finance Object for collecting various finance objects. Formats are:  
 Portfolio[fo1, fo2, ...]  
 Portfolio[n1 fo1, ... ] for numbers ni  
 Portfolio[n1 fo1 + ... ]  
 Portfolio[{fo1, ...}, {n1, ...}]  
 The latter is the preferred and most robust format. The first formats rely on automatic recognition of weight and finance object, and this works only for numeric and simple symbolic weights

**ToStandardPortfolioForm**[*p* *Portfolio*]  
 turns *p* into Portfolio[{fo1, ...}, {n1, ....}] format (if not already)

**PortfolioValue**[*p*, *d*, *r*]  
 gives the value of portfolio object *p*, at settlement date *d*, for interest rate *r*.  
 Option **Function** in **Options**[PortfolioValue] should give the function;  
 default **Value**[*foi*, *d*, *r*] for finance objects *foi*.

The option **Replace** can be used for replacement rules with parameters **Date** and **RateOfInterest**. This appears to be less relevant when you define a proper **Value** function. For example, in the **Finance** package, the function **EffectiveValue** is used, that effectively does this replacement already (in this case: **Bonds** into **CashFlows**).

- At this stage there is no **Value** function to evaluate financial objects. Therefor the following is just a formal result. Such a **Value** function however is available in the **WRI Finance Essentials** pack, see section 5.15.

**Options**[PortfolioValue]

{Function → Value, Replace → {}}

**examp** = **Portfolio**[**Bond**[{}], **Cash**[{}]];

**PortfolioValue**[**examp**, **F**[**December**, **9**, **2000**], **0.05**]

{ClassValues → {Value[Bond({}), {12, 9, 2000}, 0.05], Value[Cash({}), {12, 9, 2000}, 0.05]},

Number → {1, 1}, PortfolioWeights → {Value[Bond({}), {12, 9, 2000}, 0.05]/

(Value[Bond({}), {12, 9, 2000}, 0.05] + Value[Cash({}), {12, 9, 2000}, 0.05]),

Value[Cash({}), {12, 9, 2000}, 0.05]/

(Value[Bond({}), {12, 9, 2000}, 0.05] + Value[Cash({}), {12, 9, 2000}, 0.05]),

Value → Value[Bond({}), {12, 9, 2000}, 0.05] + Value[Cash({}), {12, 9, 2000}, 0.05]}

### 5.12.7 Simple accounting

Accounting is basic to finance. It is essential when transactions get complicated, like in a swap. The following are simple accounting rules.

<code>Pay[from, to, amount, date]</code>	gives a finance object
<code>Transactions[]</code>	should give the list of Pay transactions
<code>Credit[x, y: {}]</code>	selects the transactions of agent x that provide him or her with benefits. Default {} takes Transactions[]
<code>Debit[x, y: {}]</code>	selects the transactions of agent x that provide him or her with costs. Default {} takes Transactions[]
<code>BalanceLine[x, y_List: {}]</code>	determines the final line in the balance sheet of agent x, using transactions y. Default {} takes Transactions[].

- Let us regard three transactions performed by a bank.

```
Transactions[] =
{(*1*) Pay[Market, Bank, 1000, date],
(*2*) Pay[Bank, Market, 200, date],
(*3*) Pay[Company, Bank, 800, date] };

Debit[Bank]

{Pay(Bank, Market, 200, date)}

Credit[Bank]

{Pay(Market, Bank, 1000, date), Pay(Company, Bank, 800, date)}

BalanceLine /@ {Bank, Market, Company}

{1600, -800, -800}
```

**5.12.8 Accounting: Balance sheet, income statement and cash flow statement**

The example implemented below is from Bodie & Merton (1998:64-65).

There is the following implementation of basic accounting:

1. Databank objects (with predefined object[DataMold], object[Data] and object[Equations]) are: `BalanceSheet`, `IncomeStatement` and `CashFlowStatement`.
2. Each object[Data] here is supposed to contain entries for the last year, the current year, and the difference
3. Databank allows you to solve and update these data. Use `ShowData` to show the data.
4. Use `CashFlowStatement[Upd, investment level]` to use the data of the `BalanceSheet` and `IncomeStatement` to find the cash flow.
5. Finally you could use `Ratio[..]` to find ratios like the Price To Earnings Ratio etc.

### ?Accounting

Symbol. There is the following implementation of basic accounting:

- 1) Databank objects (with predefined object[DataMold], object[Data] and object[Equations]) are: BalanceSheet, IncomeStatement, CashFlowStatement
  - 2) Each object[Data] here is supposed to contain entries for the last year, the current year, and the difference
  - 3) Databank allows you to solve and update these data. Use ShowData to show the data.
  - 4) Use CashFlowStatement[Upd, investment level] to use the BalanceSheet and IncomeStatement data to find the cash flow
  - 5) Finally you could use Ratio[...] to find ratios like the Price To Earnings Ratio etc.
- The example implemented is from Bodie & Merton (1998)

When working with Databank objects, bear in mind:

- You switch between Databank objects with Databank[Switch, object].
- Use FullForm to see which complex symbols use quotes.
- Predefined are object[DataMold] and object[Equations] so that Databank[Solve, ...] or Upd[Solve, ...] commands are possible on these default settings.
- Before you calculate critical ratios, you would update the objects. Also, the cash flow statement relies on the balance sheet and the income statement, so you would update in a specific order.
- The Databank[Solve,...] is sensitive so inconsistent numbers, and can best be applied when some elements are indeterminate.
- The equations generate complex infinity, when the number of shares remain the same in the two years (so that the difference is zero - which is used as the denominator for worth per share).

#### 5.12.8.1 BalanceSheet

**BalanceSheet[Equations] // MatrixForm**

$$\left( \begin{array}{l} \text{Account(Receivable) + Cash() + Inventory() == Asset(Current)} \\ \text{Asset(GrossFixed) - Depreciation() == Asset(Fixed)} \\ \text{Asset(Current) + Asset(Fixed) == Asset(Total)} \\ \text{Asset(Total) == Liability(Total)} \\ \text{Account(Payable) + Debt(Short) == Liability(Current)} \\ \text{Debt(Long) + Equity() + Liability(Current) == Liability(Total)} \\ \text{Capital(Paid-in) + RetainedIncome() == Equity()} \\ \text{WorthPerShare == } \frac{\text{Equity()}}{\text{NumberOfShares}} \end{array} \right)$$

**Databank[Switch, BalanceSheet];**

**Upd[Solve];**

*Power::infy : Infinite expression  $\frac{1}{0}$  encountered.*

ShowData[]

	1	2	3
Cash()	100	120	20
Account(Receivable)	50	60	10
Inventory()	150	180	30
Asset(Current)	300	360.	60.
Asset(GrossFixed)	400	490.	90.
Depreciation()	100	130	30
Asset(Fixed)	300	360	60
Asset(Total)	600	720.	120.
Debt(Short)	90	184.6	94.6
Account(Payable)	60	72	12
Liability(Current)	150	256.6	106.6
Debt(Long)	150	150	0
Equity()	300	313.4	13.4
Liability(Total)	600	720.	120.
RetainedIncome()	100	113.4	13.4
Capital(Paid-in)	200	200.	0.
NumberOfShares	1	1	0
WorthPerShare	300	313.4	ComplexInfinity
Price(Share)	200	187.2	-12.8

5.12.8.2 Income statement

```
IncomeStatement[Equations] // MatrixForm
(
  Revenue() - Cost(Product) == Income(Gross)
  Income(Gross) - Cost(General) == EBIT()
  EBIT() - Interest() == Income(Taxable)
  Income(Taxable) - Tax() == Income(Net)
  Income(Net) == Dividend() + RetainedIncome(Delta)
  EarningPerShare ==  $\frac{\text{Income(Net)}}{\text{NumberOfShares}}$ 
)

Databank[Switch, IncomeStatement]

{Databank → IncomeStatement}

Upd[Solve] ;

Power::infy : Infinite expression  $\frac{1}{0}$  encountered.
```



ShowData[]			
	1	2	3
Revenue()	180	200	20
Cost(Product)	100	110	10
Income(Gross)	80	90.	10.
Cost(General)	30	30	0
EBIT()	50	60.	10.
Interest()	15	21	6
Income(Taxable)	35	39.	4.
Tax()	15	15.6	0.6
Income(Net)	20	23.4	3.4
Dividend()	10	10	0
RetainedIncome(Delta)	10	13.4	3.4
NumberOfShares	1	1	0
EarningPerShare	20	23.4	ComplexInfinity

### 5.12.8.3 CashFlow statement

Cash flow has to be consistent with the BalanceSheet and IncomeStatement.

CashFlowStatement [*Investment*, *inv*]

takes BalanceSheet[Data] and IncomeStatement[Data],  
regards the first entries as lagged data and the second entries as the current data,  
and calculates the 2 nd entry for CashFlowStatement[Data],  
presuming a current investment level of inv

CashFlowStatement [*Upd*, *inv*]

redefines CashFlowStatement[Data] as {first untouched,  
CashFlowStatement[Investment, inv], second – first}

```

CashFlowStatement[Equations] // MatrixForm
(
  Account(Payable, Delta) – Account(Receivable, Delta) + Depreciation(Delta) + Income(Net) – Inventory(I
    – Investment == CashFlow(FromInvesting)
    Debt(Short, Delta) – Dividend() == CashFlow(FromFinancing)
    CashFlow(FromFinancing) + CashFlow(FromInvesting) + CashFlow(FromOperating)

Databank[Switch, CashFlowStatement];

CashFlowStatement[Upd, 90];

Upd[Solve];

```

ShowData[]

	1	2	3
Income(Net)	29.	23.4	-5.6
Depreciation(Delta)	30	30	0
Account(Receivable, Delta)	10	10	0
Inventory(Delta)	30	30	0
Account(Payable, Delta)	12	12	0
CashFlow(FromOperating)	31.	25.4	-5.6
Investment	9	90	81
CashFlow(FromInvesting)	-9.	-90.	-81.
Debt(Short, Delta)	100	94.6	-5.4
Dividend()	10	10	0
CashFlow(FromFinancing)	90.	84.6	-5.4
Cash(Delta)	112.	20.	-92.

5.12.8.4 Ratio's

Ratio[x]	is a list of equations for financial ratios based on a balance sheet and the income and cash flow statements. Defined for $x \in \{\text{Return, Profit, Turnover, Leverage, Liquidity, Market, All}\}$
Ratio[All, Rule]	gives the sublists as rules
Ratio[Data, x__]	uses the data in the Databank objects BalanceSheet, IncomeStatement and CashFlowStatement to calculate ratios, taking the second entry as the current year and the first entry as a lagged value. Note that you may have to updated these Databank objects first. x will be a sequence of symbols for which Ratio[x] is defined

Ratio[All] // MatrixForm

$$\left( \begin{array}{l} \text{ReturnOnEquity}() == \frac{\text{Income(Net)}}{\text{Equity}()} \\ \text{Stock(Return)} == \frac{\frac{\text{Dividend}()}{\text{NumberOfShares}} + \text{Price(Share)} - \text{Price(Share,Lag)}}{\text{Price(Share,Lag)}} \\ \text{ReturnOnSales}() == \frac{\text{EBIT}()}{\text{Revenue}()} \\ \text{ReturnOnAssets}() == \frac{2 \text{ EBIT}()}{\text{Asset(Total)} + \text{Asset(Total,Lag)}} \\ \text{ReturnOnEquity}() == \frac{2 \text{ Income(Net)}}{\text{Equity}() + \text{Equity(Lag)}} \\ \text{Turnover(Receivable)} == \frac{2 \text{ Revenue}()}{\text{Account(Receivable)} + \text{Account(Receivable,Lag)}} \\ \text{Turnover(Inventory)} == \frac{2 \text{ Cost(Product)}}{\text{Inventory}() + \text{Inventory(Lag)}} \\ \text{Turnover(Asset)} == \frac{2 \text{ Revenue}()}{\text{Asset(Total)} + \text{Asset(Total,Lag)}} \\ \text{Leverage}() == \frac{\text{Debt(Long)} + \text{Liability(Current)}}{\text{Asset(Total)}} \\ \text{TimesInterestEarned}() == \frac{\text{EBIT}()}{\text{Interest}()} \\ \text{CurrentRatio}() == \frac{\text{Asset(Current)}}{\text{Liability(Current)}} \\ \text{AcidTest}() == \frac{\text{Account(Receivable)} + \text{Cash}()}{\text{Liability(Current)}} \\ \text{PriceToEarnings}() == \frac{\text{NumberOfShares Price(Share)}}{\text{Income(Net)}} \\ \text{Ratio(Market, BookValue)} == \frac{\text{NumberOfShares Price(Share)}}{\text{Equity}()} \end{array} \right)$$

Ratio[Data, Liquidity]

{CurrentRatio() == 1.40296, AcidTest() == 0.701481}

5.12.9 For the CAPM

AssetAlpha	Symbol for the Alpha of a CAPM regression, (r – rf) = Beta (rm – rf) + Alpha
AssetBeta	Symbol for the Beta of a CAPM regression, (r – rf) = Beta (rm – rf) + Alpha
CertainRate	Symbol for the certain or fixed rate (rf) of a CAPM regression: no spread, no risk
SpreadOnlyRate	Symbol for the rate (rf') of a CAPM regression that concerns an asset without risk but still with some spread.

The term 'risk-free' should rather not be used when there can be confusion about one's definition of risk.

## 5.13 Finance topics with a flat rate of interest

---

### 5.13.1 Summary

This package defines the cash flow and bond objects and the present value, yield and annuity functions for a flat rate of interest.

**Economics [FlatRate]**

### 5.13.2 Introduction

The assumption of a flat (constant) rate of interest provides a good introduction in the finance aspects of cash flows and bonds. The assumption of a flat rate also makes for neat algebraic solutions that *Mathematica* should handle nicely.

We assume a sequence of equal periods with a well defined periodical payment and a final payment at maturity. We use the following symbols throughout:

- $r$  rate of interest (coupon rate  $i$  or market rate  $R$ ) per period
- $m$  maturity (number of periods)
- $p$  instalment or periodical payment (possibly with a rate of growth  $g$ )
- $w$  payment at maturity (principal, worth)
- $v$  present value (capital equivalent at the beginning)

### 5.13.3 Discounting

With a cash flow of  $p[t]$  per period, we can discount each payment with a discount factor  $\frac{1}{(1+r)^t}$ . Since we assume equal payments,  $p[t] = p$ , we can add all discount factors:

$$\text{total}[r, m] = \text{Sum}\left[\frac{1}{(1+r)^t}, \{t, m\}\right]$$

$$\frac{(r+1)^{-m}((r+1)^m - 1)}{r}$$

For a perpetuity  $m \rightarrow \infty$ , giving:

$$\text{Limit}[\text{total}[r, m], m \rightarrow \text{Infinity}]$$

$$\lim_{m \rightarrow \infty} \frac{(r+1)^{-m}((r+1)^m - 1)}{r}$$

*Mathematica* stops here, since it does not know that the rate of interest remains a nonnegative number.

`FinanceLimit[expr, x→x0, y, opts]`

if  $x_0$  is Infinity, then  $y^x$  is set to Infinity and  $y^{-x}$  to zero

Note: The routine is not fully algebraic, and thus one should always check the result.

`FinanceLimit[total[r, m], m → Infinity, (1 + r)]`

$$\frac{1}{r}$$

Options[FinanceLimit] contain some rules for a nonnegative rate of interest (assuming that  $y = 1 + r$ ). Another option setting is FullSimplify → True, that makes that *expr* is first simplified; which appears a useful step in determining the limit.

### 5.13.4 CashFlow finance object

The basic object is a cash flow of  $p$  per period, for  $m$  periods, and a final payment of  $w$ . In effect, someone has borrowed  $w$ , pays periodic interest  $p$  at the coupon interest rate  $i = p / w$ , and returns the loan at maturity  $m$ .

<code>CashFlow[p, m, w(g)]</code>	object for cash payments of size $p$ per period, for $m$ periods, and a final payment of $w$ . (The periodical payment can grow with rate $g$ .)
<code>Bullit[i, m, w]</code>	= <code>CashFlow[i w, m, w]</code> , for coupon rate $i$
<code>FloatingRateNote[R, m, w]</code>	= <code>CashFlow[R w, m, w]</code> , for market rate $R$
<code>ZeroCoupon[R, m, v]</code>	= <code>CashFlow[0, m, v (1 + R)<sup>m</sup>]</code>

- Regard these objects for a given set of parameters.

`example = {p → 100 Dollar/Year, m → 10 Year, w → 1000 Dollar};`

`cf = CashFlow[p, m, w] /. example`

`CashFlow( $\frac{100 \text{ Dollar}}{\text{Year}}$ , 10 Year, 1000 Dollar)`

`bul = Bullit[i / Year, m, w] /. example`

`CashFlow( $\frac{1000 \text{ Dollar } i}{\text{Year}}$ , 10 Year, 1000 Dollar)`

```
frn = FloatingRateNote[R / Year, m, w] /. example
```

$$\text{CashFlow}\left(\frac{1000 \text{ Dollar } R}{\text{Year}}, 10 \text{ Year}, 1000 \text{ Dollar}\right)$$

```
zer = ZeroCoupon[i / Year, m, w] /. example
```

$$\text{CashFlow}\left(0, 10 \text{ Year}, 1000 \text{ Dollar}\left(\frac{i}{\text{Year}} + 1\right)^{10 \text{ Year}}\right)$$

<code>CashFlowSumSimplify[expr]</code>	sums CashFlow[] objects of same maturity
--	--

Note: This has not been implemented for growth funds.

```
CashFlowSumSimplify[cf + bul + frn]
```

$$\text{CashFlow}\left(\frac{100 \text{ Dollar}(10 i + 10 R + 1)}{\text{Year}}, 10 \text{ Year}, 3000 \text{ Dollar}\right)$$

5.13.5 Present value

The assumption is that all payments are done at the end of the period. If payments are done at the beginning, then one simply sums the first payment and the present value of the remainder considered as being at the end.

<code>PV[CashFlow[p, m, w(g), r]</code>	the present value at discount rate r
---	--------------------------------------

```
capital == PV[CashFlow[p, m, w], r]
```

$$\text{capital} == w(r + 1)^{-m} + \frac{p(1 - (r + 1)^{-m})}{r}$$

```
capital == PV[Bullit[i, m, w], R]
```

$$\text{capital} == w(R + 1)^{-m} + \frac{i(1 - (R + 1)^{-m})w}{R}$$

```
capital == PV[FloatingRateNote[R, m, w], R] // Simplify
```

$$\text{capital} == w$$

```
capital == PV[CashFlow[p, m, w, g], r]
```

$$\text{capital} == w(r + 1)^{-m} + \frac{p\left(1 - \left(\frac{g+1}{r+1}\right)^m\right)}{r - g}$$

5.13.6 Yield

<code>Yield[CashFlow[p, m, w], price]</code>	solves r from PV[cf, r] == price
--	----------------------------------

- Let someone offer \$900 for above cash flow example. The implied yield is:

```
Yield[ cf /. {Year → 1, Dollar → 1}, 900]

0.117519
```

- For a growth fund:

```
Yield[CashFlow[100, 10, 1000, 0.05], 900]

0.13854
```

EffectiveRate [ <i>i</i> , <i>n_Integer</i> ]	gives the true period rate, when the bank says that it charges rate <i>i</i> per period, but in fact charges <i>i</i> / <i>n</i> over <i>n</i> subperiods
--	---

```
EffectiveRate[0.07, 12]

0.0722901
```

5.13.7 Periodical payment

Suppose you borrow capital *v* now. Without intermediate interest payments, you would have to repay *f v* with factor  $f = (1 + r)^m$  at maturity. Suppose that you only pay *w* at maturity. Then the remainder *f v* - *w* must be paid as interest or redemption in the period before. If the periodical payment is constant and annual then it is called an annuity. If the payment grows with a constant rate per period, then this is called a growth fund.

Instalment [ <i>r</i> , <i>m</i> , <i>w</i> , <i>v</i> ( <i>g</i> )]	general statement for a periodical payment
Annuity[ <i>x__</i> ]	just a Head, replaces with Instalment
Perpetuity[ <i>r</i> , <i>v</i> ]	gives the perpetuity Annuity[ <i>r</i> , Infinity, 0, <i>v</i> ] = <i>r v</i> for the rate of interest <i>r</i> and the principal or present value <i>v</i>
GrowthFund [ <i>r</i> , <i>m</i> , <i>w</i> , <i>v</i> , <i>g</i> ]	gives the <i>first</i> payment for a loan of <i>v</i> , with repayment of <i>w</i> at <i>m</i> , when the period growth rate is <i>g</i>
ConstantRedemption [ <i>r</i> , <i>m</i> , <i>w</i> , <i>v</i> ]	repays debt <i>v</i> with a constant amount ( <i>v</i> - <i>w</i> )/ <i>m</i> per period and an additional <i>w</i> at the end, with interest payments determined by remaining debt

Note: A sinking fund can be created by Instalment[*r*, *m*, -*w*, 0]. Note: If the word Table is appended at the end of the input of Instalment, then a list of rules is created that can be input for InstalmentTable. Appending RoundAt → *n* instead, will give a table with the payment rounded to *n* digits (negative *n* too) - and really rounded, not just printed in PaddedForm.

■ In rising complexity:

```
p == Instalment[r, m, 0, v]
```

$$p == \frac{rv}{1 - (r + 1)^{-m}}$$

```
p == Instalment[r, m, w, v]
```

$$p == \frac{r(v - (r + 1)^{-m} w)}{1 - (r + 1)^{-m}}$$

```
p == Instalment[r, m, w, v, g]
```

$$p == \frac{(r - g)(v - (r + 1)^{-m} w)}{1 - \left(\frac{g + 1}{r + 1}\right)^m}$$

■ The annuity for a bullit, with w = v and r = i:

```
p == Annuity[i, m, v, v] // Simplify
```

$$p == i v$$

■ The following generates a list of payments, so m best has a numerical value.

```
ConstantRedemption[r, 2, w, v]
```

$$\{\text{Redemption} \rightarrow \left\{\frac{v - w}{2}, \frac{v - w}{2}\right\}, \text{DebtTrajectory} \rightarrow \left\{v + \frac{w - v}{2}, w\right\}, \\ \text{Interest} \rightarrow \left\{rv, r\left(v + \frac{w - v}{2}\right)\right\}, \text{Sum} \rightarrow \left\{rv + \frac{v - w}{2}, \frac{v - w}{2} + r\left(v + \frac{w - v}{2}\right)\right\}\}$$

5.13.8 Payment Tables

A table of payments, of interest and amortisation, and remaining debt, can be made for various schemes. The basic format uses rules with symbols `Redemption`, `Interest` and `DebtTrajectory` or `Debt`.

<code>InstalmentTable[{rules}]</code>	for rules <code>Redemption</code> → ..., <code>Interest</code> → ..., <code>DebtTrajectory</code> → ... , <code>Debt</code> → ...
<code>InstalmentTable[CashFlow[args], r]</code>	for a <code>CashFlow</code> object
<code>InstalmentTable[f[args]]</code>	for f= <code>Instalment</code> , <code>Annuity</code> , <code>GrowthFund</code> , <code>ConstantRedemption</code> .

Note: The last element in the list is w, or better said exclusive of its payment. Also, the table Chops the values at .01. Note: The `DebtTrajectory` can and will be computed from `Debt` and `Amortisation` settings. Note: The `Number` option controls printing. Note: The series are available from `Results[InstalmentTable]`.



**InstalmentTable[Instalment[0.1, 3, 150, 200]]**

period	payment	interest	redemption	debt
1	35.11	20.00	15.11	184.89
2	35.11	18.49	16.62	168.28
3	35.11	16.83	18.28	150.00

**InstalmentTable[Instalment[0.1, 3, 150, 200, 0.05]]**

period	payment	interest	redemption	debt
1	33.51	20.00	13.51	186.49
2	35.19	18.65	16.54	169.95
3	36.95	17.00	19.95	150.00

**InstalmentTable[CashFlow[30, 3, 150], 0.1]**

Present value = 187.303

period	payment	interest	redemption	debt
1	30.00	18.73	11.27	176.03
2	30.00	17.60	12.40	163.64
3	30.00	16.36	13.64	150.00

**InstalmentTable[{Redemption → {10, 20, 20}, Interest → {30, 20, 10}, Debt → 200}]**

period	payment	interest	redemption	debt
1	40.00	30.00	10.00	190.00
2	40.00	20.00	20.00	170.00
3	30.00	10.00	20.00	150.00

### 5.13.9 Schemes on redemption

Above schemes define the period payment by directly determining the period total. A different approach is to define only redemption, and then let the total be derived from redemption and the interest on remaining debt. A quick example is constant redemption, where each period's redemption follows from (principal - final payment) / m.

**InstalmentTable[ConstantRedemption[.10, 3, 150, 200.]]**

period	payment	interest	redemption	debt
1	36.67	20.00	16.67	183.33
2	35.00	18.33	16.67	166.67
3	33.33	16.67	16.67	150.00

If the redemption scheme is regular, there still can be a quick way to determine the present value.

`PVFromRedemption[d, i, R, pva]`

gives the present value for any arbitrary redemption scheme,  
as  $pva + i/R (d - pva)$ , with d the principal of the loan, i the coupon rate,  
and pva the present value at market rate R for that scheme

See Results[PVFromRedemption] for the split between pva and pvi (the PV of the interest payments).

- For above loan with a coupon rate of 10% and with a period redemption of 50 / 3, the present value at a market rate of 7% can be found by first determining the present value of the redemption, and then applying above formula.

```
pva = PV[CashFlow[50/3, 3, 150], 0.07]
```

```
166.183
```

```
PVFromRedemption[200, 0.1, 0.07, pva]
```

```
214.493
```

- These two steps in one:

```
PV[Hold[ConstantRedemption][0.1, 3, 150, 200], 0.07]
```

```
214.493
```

```
Results[PVFromRedemption]
```

```
{166.183, 48.3096}
```

### 5.13.10 Balance sheet accounting

The accounting of simple payments was discussed In section 5.12. By including financial instruments we can handle complicated transactions like in a swap. Let us for example regard the following three transactions performed by a bank.

```
Transactions[] = {(1*) Pay[Market, Bank, w, date],
(*2*) Pay[Bank, Market, FloatingRateNote[LIBOR+.0025, m, w], date],
(*3*) Pay[Company, Bank, Bullit[i + .015, m, w], date] }
```

```
{Pay(Market, Bank, w, date), Pay(Bank, Market, CashFlow((LIBOR + 0.0025) w, m, w), date),
Pay(Company, Bank, CashFlow((i + 0.015) w, m, w), date)}
```

```
BalanceLine[Bank]
```

```
w + CashFlow((i + 0.015) w, m, w) - CashFlow((LIBOR + 0.0025) w, m, w)
```

```
CashFlowSumSimplify[%]
```

```
w + CashFlow((i - LIBOR + 0.0125) w, m, 0)
```

### Note

These routines are also used in these example basic financial mathematics intermediate test and exams.

## 5.14 The Capital Asset Pricing Model

### 5.14.1 Summary

The `Finance`CAPM`` package provides some routines for the Capital Asset Pricing Model.

**Economics** [`Finance`CAPM``]

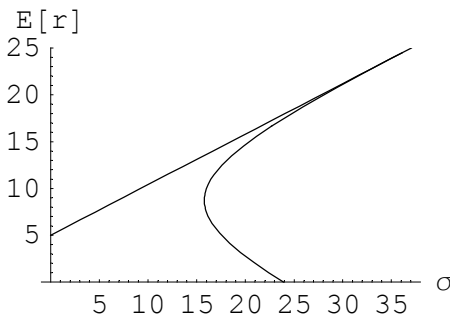
### 5.14.2 Introduction

As Luenberger (1998:173) rightly remarks, there are two main problems in investment science: To determine the prices of assets and the best portfolio of them. These problems are discussed here jointly. We will concentrate on the CAPM that gives an one-period equilibrium relation between the return of an investment and its spread (standard deviation).

The CAPM theory is best explained by reference to a plot in the  $\{\sigma, \mu\}$  space, i.e. of the spreads and expected returns of assets, where the idea is that the spread causes the expected return (or the premium above the certain rate of interest).

- This example has data in percentages to get clean axes. The rate of interest is 5%.

```
CAPMPlot[5, FrontierExample[r], {r, 0, 25}];
```



- With the interest rate of 5%, the market portfolio (i.e. the portfolio optimised over all uncertain assets) has an expected return of 24.5% with a spread of 36.1% points.

```
{sigmaM, rM} = MarketPortfolio /. Results[CAPMPlot]
```

```
{36.1158, 24.5013}
```

The plot uses the following information:

- The certain or fixed rate ( $r_f$ ) can be found at  $\{0, r_f\}$ . Similarly, *cash* is at the origin  $\{0, 0\}$
- All uncertain assets are within the Markowitz efficiency frontier given by the parabola. The efficiency frontier gives minimum spread for a given return.

- The market portfolio can be found at the point  $\{\sigma_M, r_M\}$  in the tangent from the parabola that passes through the certain rate. (This means that the interest rate affects the market portfolio, even though it is not in that portfolio.)
- The line from  $\{0, r_f\}$  to  $\{\sigma_M, r_M\}$  is called the Capital Market Line (CML). The slope of that line is the premium-to-spread ratio or the 'price of spread' or the Sharpe Market Index.
- The CML theorem is that all optimal portfolios that involve the certain rate are on the capital market line. If it is possible to go short, then also points to the right of  $\{\sigma_M, r_M\}$  will be on an extension of the capital market line. Otherwise, if no short position is possible, optimal points are just on the right section of the efficiency frontier. (Note that a short position is always balanced by a long position, so that the market 'as a whole' cannot go short. See the included paper on this issue in appendix B. )

The CML theorem in other words states that an optimal portfolio that involves the certain rate ( $r_f$ ) is a weighted average of that certain rate and the market rate ( $r_M$ ). The investor only has to select his or her preferred weight. Denoting the weight of the market as  $\beta$ , gives the expected return ( $r$ ) of an optimal portfolio as  $r = (1 - \beta)r_f + \beta r_M$ .

The premium (reward or excess return) of a rate is the mark up on the certain rate, i.e.  $\pi = r - r_f$ . The premium ratio is  $\pi_R = (r - r_f) / (r_M - r_f)$ . Basic algebra gives that  $\beta = \pi_R$  for optimal portfolio's that are on the CML.

Note that the premium depends on various causes such as spread (standard deviation based on intrinsic correlations), volatility (spread based on time series) or risk (see the section on risk). Traditional research focusses on the  $\{\sigma, \mu\}$  space. Since the certain rate has spread 0, the spread of an optimal portfolio on the CML is  $\sigma = \beta \sigma_M$ .

- The expected return of an optimal portfolio thus can be expressed as a function of its spread. The coefficient of the spread is the premium-to-spread ratio.

**CapitalMarketLine** $[\sigma, r_f, \sigma_M, r_M]$

$$\text{ExpectedReturn} == r_f + \frac{\sigma (r_M - r_f)}{\sigma_M}$$

- These are the basic equations.

**CAPMEquations** $[\ ]$  /. **ToCAPMSymbols** // **MatrixForm**

$$\left( \begin{array}{l} r == (1 - \beta)r_f + \beta r_M \\ \sigma == \beta \sigma_M \\ S_M == \frac{r_M - r_f}{\sigma_M} \\ \pi == r - r_f \\ \pi_M == r_M - r_f \\ \pi_R == \frac{\pi}{\pi_M} \end{array} \right)$$

- The capital market line gives the expected return of an asset or portfolio as a function of its spread.

```
(Solve[Take[CAPMEquations[], 2],
{ExpectedReturn[Portfolio]}, {Weight[]}] //
Simplify) /. ToCAPMSymbols ;

r == Collect[%[[1, 1, 2]], {σ, σM}]

r == rf +  $\frac{\sigma(r_M - r_f)}{\sigma_M}$ 
```

An issue now is the interaction of the prices of the assets and the efficiency frontier.

Luenberger (1998:177) defines CAPM as the theorem that: If the market portfolio M is efficient, then the expected return  $r_i$  of any asset satisfies  $r_i = (1 - \beta_i) r_f + \beta_i r_M$  and  $\beta_i = \sigma_{i,M} / \sigma_M^2$ .

Note that the CAPM theorem concerns assets *within* the Markowitz frontier, that would be included in the market portfolio. This thus is a different  $\beta$  than the one for portfolios on the Capital Market Line. However, the portfolios on the CML are fully correlated with the market portfolio, and thus the relation holds too. For example,  $\beta = 1$  for the market portfolio. The literature then defines  $\beta_i \equiv \sigma_{i,M} / \sigma_M^2$  even if there would not be optimality. From the properties of covariances it then holds also in inoptimality that the  $\beta$  of a portfolio is a simple weighted average of the  $\beta$ 's of the assets, with the weights of the portfolio.

A consequence of this definition of the betas is that it no longer need hold that  $\beta = \pi_R$  for the off-CML assets and portfolios. If only holds under optimality, and in fact, we can use it to impose optimality.

### 5.14.3 The small models

CAPM[ ] is a SolveFrom application, which means that you can set specific variables and parameters and solve for the unknowns. Here, you can also set the options to a specific set of equations, while the equations above are default.

#### 5.14.3.1 The model interface

CAPM[{x__}, y__]	is a SolveFrom application, using the Equations of the options
CAPMEquations[i]	are predefined sets of equations for i = blank, 1, 2, Price
CAPMSymbols	the list of compact symbols
ToCAPMSymbols	a list of rules to change CAPM symbols into short variables

Note that the subscripts like in  $r_M$  are in Strings ("M") to prevent accidental values. Note that protection of such subscripted variables appears infeasible.

- The three predefined sets of equations all use the following symbols.

**ListOfSymbols [CAPM]**

CertainRate	$r_f$
Correlation	$R$
PremiumToSpreadRatio	$S_M$
AssetBeta(1)	$\beta_1$
AssetBeta(2)	$\beta_2$
Covar(1, 2)	$\sigma_{1,2}$
Covar(i, M)	$\sigma_{i,M}$
ExpectedReturn(1)	$r_1$
ExpectedReturn(2)	$r_2$
ExpectedReturn(Market)	$r_M$
ExpectedReturn(Portfolio)	$r$
Premium(Market)	$\pi_M$
Premium(Portfolio)	$\pi$
PremiumRatio()	$\pi_R$
PremiumRatio(1)	$\pi_{R,1}$
PremiumRatio(2)	$\pi_{R,2}$
Spread(1)	$\sigma_1$
Spread(2)	$\sigma_2$
Spread(Market)	$\sigma_M$
Spread(Portfolio)	$\sigma$
Weight()	$\beta$
Weight(1)	$w$

5.14.3.2 Solving the default model

Since we use percentages to get readable axes in plots, we keep on using percentages in order to get not confused. Note however that routines have generally been written with perunages in mind, as would be the normal use.

- Given the data of above plot, suppose that you desire a portfolio with a premium of 3%.

```
CAPM[{ExpectedReturn[Market] → 24.5,  
      Spread[Market] → 36,  
      CertainRate → 5, Premium[Portfolio] → 3}];
```

- The premium of the market portfolio is 19.5%, so your 3% premium implies that the share of the market portfolio in your portfolio should be about 15%.

**SolveShow[Last[%]]**

	1
PremiumToSpreadRatio	0.541667
ExpectedReturn(Portfolio)	8.
Premium(Market)	19.5
PremiumRatio()	0.153846
Spread(Portfolio)	5.53846
Weight()	0.153846

### 5.14.3.3 The complexer model

The complexer model here gives the typical explanation for the efficiency frontier. If there are two uncertain assets then a portfolio of them has weighted expectation and variance. The market portfolio weights for the two assets arise from the tangency condition.

- Note: CAPMEquations[1] excludes the last three conditions on optimal  $w$ .

**CAPMEquations[2] /. ToCAPMSymbols // MatrixForm**

$$\left( \begin{array}{l} r == (1 - \beta) r_f + \beta r_M \\ \sigma == \beta \sigma_M \\ S_M == \frac{r_M - r_f}{\sigma_M} \\ \pi == r - r_f \\ \pi_M == r_M - r_f \\ \pi_R == \frac{\pi}{\pi_M} \\ r_M == w r_1 + (1 - w) r_2 \\ \sigma_M == \sqrt{w^2 \sigma_1^2 + (1 - w)^2 \sigma_2^2 + 2 (1 - w) w \sigma_{1,2}} \\ R == \frac{\sigma_{1,2}}{\sigma_1 \sigma_2} \\ \pi_{R,1} == \frac{r_1 - r_f}{r_M - r_f} \\ \pi_{R,2} == \frac{r_2 - r_f}{r_M - r_f} \\ \beta_1 == \frac{w \sigma_1^2 + (1 - w) \sigma_{1,2}}{\sigma_M^2} \\ \beta_2 == \frac{(1 - w) \sigma_2^2 + w \sigma_{1,2}}{\sigma_M^2} \\ \beta_1 == \pi_{R,1} \\ \beta_2 == \pi_{R,2} \\ w == \frac{(r_1 - r_f) \sigma_2^2 + (r_f - r_2) \sigma_{1,2}}{(r_2 - r_f) \sigma_1^2 + (r_1 - r_f) \sigma_2^2 + (-r_1 - r_2 + 2 r_f) \sigma_{1,2}} \end{array} \right)$$

- For example, when the two uncertain assets are fully uncorrelated, then it is possible to create an artificial *certain* asset by properly mixing the two.

```
Take[CAPMEquations[2], {8, 9}] /. ToCAPMSymbols
```

$$\left\{ \sigma_M == \sqrt{w^2 \sigma_1^2 + (1-w)^2 \sigma_2^2 + 2(1-w)w \sigma_{1,2}}, R == \frac{\sigma_{1,2}}{\sigma_1 \sigma_2} \right\}$$

```
Solve[%, {w}, {\sigma_{1,2}}];
```

```
Union[% /. {R -> -1, \sigma_M -> 0} // Simplify]
```

$$\left\{ \left\{ w \rightarrow \frac{\sigma_2}{\sigma_1 + \sigma_2} \right\} \right\}$$

- For example, the last equation on the  $w$  can be solved from the earlier ones.

```
eqs = Drop[Drop[CAPMEquations[2], 7], -1];
```

```
Solve[eqs, {Weight[1]}, {Correlation, AssetBeta[1], AssetBeta[2],  
ExpectedReturn[Market], PremiumRatio[1], PremiumRatio[2],  
Spread[Market]}] /. ToCAPMSymbols
```

$$\left\{ \left\{ w \rightarrow \frac{r_1 \sigma_2^2 - r_f \sigma_2^2 - r_2 \sigma_{1,2} + r_f \sigma_{1,2}}{r_2 \sigma_1^2 - r_f \sigma_1^2 + r_1 \sigma_2^2 - r_f \sigma_2^2 - r_1 \sigma_{1,2} - r_2 \sigma_{1,2} + 2 r_f \sigma_{1,2}} \right\} \right\}$$

Note though that the theory runs in a different order. The theory is that the slope of the efficiency frontier in the optimal point must be equal to the slope of the capital market line. From this follows optimal  $w$ , and from this again it follows that the  $\beta_i$  are equal to the premium ratios. We can quickly show this.

$$\text{marketvar}[w_] = w^2 \sigma_1^2 + (1-w)^2 \sigma_2^2 + 2 w (1-w) \sigma_{1,2};$$

$$\text{marketr}[w_] = w r_1 + (1-w) r_2;$$

$$\text{slope}[w_] = (\text{marketr}[w] - r_f) / \text{Sqrt}[\text{marketvar}[w]]$$

$$\frac{w r_1 + (1-w) r_2 - r_f}{\sqrt{w^2 \sigma_1^2 + (1-w)^2 \sigma_2^2 + 2(1-w)w \sigma_{1,2}}}$$

- We can find the slope also as  $\frac{d \mu}{d \sigma} = \frac{d \mu}{d w} / \frac{d \sigma}{d w}$ . Setting these slope expressions equal to one another allows us to solve for the unknown  $w$ , the optimal allocation of the stocks.

```
slope[w] == D[marketr[w], w] / D[Sqrt[marketvar[w]], w];
```

```
optw = Simplify[Solve[%, w]];
```



- The following only collects terms for readability.

**Division[Collect[#1, { $\sigma_1^2$ ,  $\sigma_2^2$ ,  $\sigma_{1,2}$ }] &, optw[[1, 1, 2]], {}]**

$$\frac{(r_f - r_1) \sigma_2^2 + (r_2 - r_f) \sigma_{1,2}}{(r_f - r_2) \sigma_1^2 + (r_f - r_1) \sigma_2^2 + (r_1 + r_2 - 2 r_f) \sigma_{1,2}}$$

The binary model has also been put into a separate routine call. This is useful also because of the Two Funds Theorem, that says that the efficiency frontier can be described by two portfolios.

**BinaryWeights[r, {s1, r1}, {s2, r2}, covar]**

determines the optimal weight of the first asset, given certain rate r,  
for uncertain assets with expected value ri and spread si, and covariance

**BinaryWeights[r, {s1, r1}, {s2, r2}, R, Correlation]** calculates covar = R s1 s2

**BinaryWeights[Solve, r, {s1, r1}, {s2, r2}, c (, Correlation)]**

calculates the CAPM consequences of the optimal weight too

The weight of the other asset is (1 - w), so that no short selling is allowed.

- Of these two formats we evaluate only the second.

**BinaryWeights[Solve, r"f", { $\sigma_1$ ,  $r_1$ }, { $\sigma_2$ ,  $r_2$ },  $\sigma_{1,2}$ ] // Simplify**

**BinaryWeights[Solve, 6, {15, 8}, {2, 14}, -.5, Correlation]**

{Weight → 0.0653728, Variance(Market) → 2.6227,  
Spread(Market) → 1.61947, ExpectedReturn(Market) → 13.6078,  
Slope → 4.69767, AssetBeta(1) → 0.262889, AssetBeta(2) → 1.05156}

Note: It is useful to derive the  $\beta_i$  from the  $\sigma_{i,M}$ , since textbooks seldom do it. Write  $\tilde{r}_i$  for a random event (since  $r_i$  gives the expectation). By definition:  $n \sigma_{1,M} = \Sigma (\tilde{r}_1 - r_1) (\tilde{r}_M - r_M)$ . Note that  $r_M = w r_1 + (1 - w) r_2$ . Hence  $n \sigma_{1,M} = \Sigma (\tilde{r}_1 - r_1) (\tilde{r}_M - r_M) = w \Sigma (\tilde{r}_1 - r_1)^2 + (1 - w) \Sigma (\tilde{r}_1 - r_1) (\tilde{r}_2 - r_2)$ . Hence we find that  $\sigma_{1,M} = w \sigma_1^2 + (1 - w) \sigma_{1,2}$ , and  $\beta_1 = (w \sigma_1^2 + (1 - w) \sigma_{1,2}) / \sigma_M^2$ .

Note that there thus is a fixed point: The  $\beta_i$  depend on the  $w$ , the  $w$  depends upon the  $r_i$ , and the  $r_i$  depend upon the  $\beta_i$ . However, the  $r_i$  are considered given here, so there is a fixed anchor.

#### 5.14.3.4 Comparing solutions

Let us compare the effect of the last conditions on the optimality of the market portfolio.

- We take an interest rate of 6%, correlation of -0.5, and a whole list of other assumptions.

```
params = {Weight[1] → .4, ExpectedReturn[1] → 8,
  ExpectedReturn[2] → 14, CertainRate → 6, Weight[] → 0.5,
  Spread[1] → 15, Spread[2] → 2, Correlation → -.5};
```

- In the first run we set the weight of the first asset to 40%. Note: CAPMEquations[1] excludes the last three conditions on optimal  $w$ .

```
SetOptions[CAPM, Equations → CAPMEquations[1]];
sol[1] = CAPM[params];
```

- In the second run, we let the model determine optimality. It turns out that two of the optimality conditions are difficult for Solve.

```
SetOptions[CAPM, Equations → Drop[CAPMEquations[2], {-3, -2}]];
sol[2] = CAPM[Drop[params, 1]];
```

- We find that the  $\beta$ 's and the premium ratios differ for the first solution, but are properly equal for the second one. Curiously the market return and spread have dropped, but the price of spread increases (Sharpe measure). The share of the first asset rises, but since it earns less, the return of our portfolio drops. Thus we find that optimality *reduces* our return. What is the lesson of this ? The point that we learn here is that apparently we underestimated the  $\beta_1$  in the first solution. In other words, the CAPM equations are dangerous stuff when applied to inoptimal conditions or settings.

```
SolveShow[SolveFrom, sol[1], sol[2]] /. ToCAPMSymbols
```

	1	2
$r_f$	6	6
$R$	-0.5	-0.5
$S_M$	1.01835	4.69767
$\beta_1$	2.67857	0.262889
$\beta_2$	-0.119048	1.05156
$\sigma_{1,2}$	-15.	-15.
$r_1$	8	8
$r_2$	14	14
$r_M$	11.6	13.6078
$r$	8.8	9.80388
$\pi_M$	5.6	7.60776
$\pi$	2.8	3.80388
$\pi_R$	0.5	0.5
$\pi_{R,1}$	0.357143	0.262889
$\pi_{R,2}$	1.42857	1.05156
$\sigma_1$	15	15
$\sigma_2$	2	2
$\sigma_M$	5.49909	1.61947
$\sigma$	2.74955	0.809737
$\beta$	0.5	0.5
$w$	0.4	0.0653728

5.14.4 Consequences for the price of an asset

The CAPM derives its name from the point that it provides prices for assets. The CAPM routine, next to being a SolveFrom application, also allows this price determination.

CAPM[Before, futureprice, rf, rm, $\beta$ ]	the current price as a function of the future price
CAPM[After, currentprice, rf, rm, $\beta$ ]	the future price as a function of the current price

Both at certain rate  $r_f$ , market rate  $r_m$ , and beta  $b$ . These calculations assume that  $r_m$  is not affected by the price change (and the implied weight change).

- The current and future price of an asset are related by the CAPM rate of return.

```
CurrentPrice == CAPM[Before, FuturePrice, rF", rM", beta]
```

$$\text{CurrentPrice} == \frac{\text{FuturePrice}}{r_f + \beta (r_M - r_f) + 1}$$

```
FuturePrice == CAPM[After, CurrentPrice, rF", rM", beta]
```

$$\text{FuturePrice} == \text{CurrentPrice} (r_f + \beta (r_M - r_f) + 1)$$

It is possible to say a little bit more about such prices if we leave the one-period framework. An asset with a current price of \$1000 and a current *perpetual* expected rate of return of 10%, has a theoretical perpetuity equivalent of \$100. If the CAPM rate differs, then this perpetuity can be converted with this rate into a new price. By taking account of the  $\beta$  there can be an immediate appreciation or depreciation of the price. We can also include a certainty equivalent. If someone borrows at the current certain rate and buys the asset, then that rate times the price gives a certainty equivalent for the theoretical perpetuity. Note that this exercise is provisional, and ought to be replaced by proper dynamics.

- This is another application of SolveFrom, where you can set values, and the routine finds the remainder.

```
SetOptions[CAPM, Equations → CAPMEquations[Price]]
```

$$\left\{ \begin{aligned} &\text{Equations} \rightarrow \left\{ \begin{aligned} &\text{Price}() \text{Rate}() == \text{Revenue}(), \text{CertainRate} \text{Price}() == \text{Revenue}(\text{CE}), \\ &\text{Price}(\text{NextPeriod}) == \text{Price}(\text{Appreciation}) (\text{Rate}(\text{CAPM}) + 1), \text{Price}(\text{Appreciation}) == \frac{\text{Revenue}()}{\text{Rate}(\text{CAPM})}, \\ &\text{Rate}(\text{CAPM}) == \text{CertainRate} - \beta (\text{ExpectedReturn}(\text{Market}) - \text{CertainRate}), \\ &\text{Dif}(\text{Rate}) == \text{Rate}() - \text{Rate}(\text{CAPM}), \\ &\text{Dif}(\text{Price}) == \text{Price}(\text{Appreciation}) - \text{Price}(), \text{Dif}(\text{Price}, \text{Rate}) == \frac{\text{Dif}(\text{Price})}{\text{Price}()} \end{aligned} \right\} \end{aligned} \right\}$$

- Note that in this case we must use peruauges for the rates of return.

```
CAPM[{Rate[] → .10, Price[] → 1000,  
      beta → -.5, CertainRate → .07, ExpectedReturn[Market] → .12}];
```

```
SolveShow[Last[%]]
```

	1
Dif(Price)	52.6316
Dif(Rate)	0.005
Dif(Price, Rate)	0.0526316
Price(NextPeriod)	1152.63
Price(Appreciation)	1052.63
Rate(CAPM)	0.095
Revenue()	100.
Revenue(CE)	70.

### 5.14.5 Parametric representation

The  $\sigma[r]$  format of the efficiency frontier is likely the easiest representation. But using Plot on  $\sigma[r]$  would put  $r$  on the horizontal axis - and *Mathematica* does not allow you to simply toggle the axes. The routine InversePlot of the Economics Pack however does that trick. InversePlot works by transforming to  $\{\sigma[r], r\}$  as a (degenerate) parametric representation.

It must be noted that there are two non-degenerate parametric representations. In the first case the  $\{\sigma[w], r[w]\}$  are directly dependent upon weights  $w$  (e.g. in the domain  $[0, 1]$ ). In the second case there is a parametric representation of the weights, as  $\{\sigma[w[\lambda]], r[w[\lambda]]\}$  for a single parameter  $\lambda$ , and  $\lambda$ 's domain e.g. solves from  $0 \leq w \leq 1$ .

Given these non-degenerate representations, it has been decided to implement the frontier always as a parametric pair  $\{\sigma, \mu\}$ , even when  $\sigma[r]$  would be more elegant and  $\{\sigma[r], r\}$  reads pedantic. You of course are free to differ, and then could use InversePlot.

- Expected return and spread in this example are in percentages.

**FrontierExample[]**

$$\{\sqrt{4.24 r^2 - 74. r + 572.12}, r\}$$

The following routines allow manipulation of the parametric forms.

ToNonParametric[ $\{\sigma, r\}, s\_Symbol]$	solves the parametric forms of $\sigma$ and $r$ , eliminating the parameter and making $r[\sigma]$
ToNonParametric[ Spread, $\{\sigma, r\}, s\_Symbol]$	solves the parametric forms of $\sigma$ and $r$ , eliminating the parameter and making $\sigma[r]$
EVRRange[ $\{r, b, e\}, s\_Symbol]$	solves a parametric representation of $r$ for its $b$ to $e$ domain for $s$ , and gives the $s$ domain.

Note: These routines work for the various parametric formats. In practice, the CAPM allows an easy  $\{\sigma[r], r\}$  format, and one would not really use these routines. They still are included here for completeness - and perhaps complexer future developments. See ParametricRange for constructions also involving sigma.

**ToNonParametric[Spread, FrontierExample[r], r]**

$$\{\text{Spread} \rightarrow \sqrt{4.24 \text{ExpectedReturn}^2 - 74. \text{ExpectedReturn} + 572.12}\}$$

```
MPFromParametric[] [rf, {sigma, r}, w_Symbol]
```

solves the market portfolio  $\{\sigma[w_M], r[w_M]\}$  from a given parametric representation of the efficiency frontier, for  $w$  in  $[0, 1]$

```
MPFromParametric[Solve] [rf, {sigma, r}, w_Symbol]
```

determines the parameter  $w_M$  such that the slope of the Capital Market Line in the  $\{\sigma, r\}$  space equals  $(r[w_M] - r_f) / \sigma[w_M]$  with  $w$  in  $[0, 1]$

Give  $\{w\_Symbol, b, e\}$  instead of only the symbol for domain control. Note that there are two non-degenerate parametric representations. In the first case the  $\{\sigma[w], \text{ev}[w]\}$  are directly dependent upon weights  $w$ , e.g. in  $[0, 1]$ . Use symbol `Weight` here. In the second case there is a parametric representation of the weights, as  $\{\sigma[w[\text{lab}]], \text{ev}[w[\text{lab}]]\}$ , and  $\text{lab}$ 's domain solves e.g. from  $0 \leq w \leq 1$ . Use symbol `labda` here. Also, `Options[MPFromParametric]` control adjustment of a solution to a specified domain, and help in plotting. Note: Two simple subroutines are `Max0TolIfNegative` and `ToNonNegative`.

```
MPFromParametric[] [5, FrontierExample[r], r]
```

```
MPFromParametric::dom : Solution 24.5013 isn't in {0, 1}
{36.1158, 24.5013}
```

You can control domain testing by the parameter `Domain`  $\rightarrow$  `True` | `False` in `Options[MPFromParametric]`, with default `False`.

#### 5.14.6 Plots and scatter

Before we continue with the determination of the efficiency frontier and the market portfolio, it is useful to discuss the plotting tools first.

### 5.14.6.1 Plots

```
CAPMPlot[rf, {sigma, r}, {r_Symbol, b, e}]
```

plots the efficiency frontier and capital market line for degenerate {sigma, r}

```
CAPMPlot[rf, {sigma, r}, {b, e}, s_Symbol: labda]
```

plots the efficiency frontier and capital market line, for  $r[s]$  in  $\{b, e\}$ .  
Uses EVRange, and then submits to CAPMParametricPlot

```
CAPMParametricPlot[rf, {sigma, r}, s_Symbol, opts___Rule]
```

for a parametric representation of the market portfolio,  
with  $\sigma[s]$  and expected  $r[s]$  for  $\{s, 0, 1\}$ . The certain rate is  $rf$

```
CAPMParametricPlot[Line, rf, {sigma, r}, s_Symbol, opts___Rule]
```

plots just the Capital Market Line

```
CAPMParametricPlot[{sigma, r}, s_Symbol, opts___Rule]
```

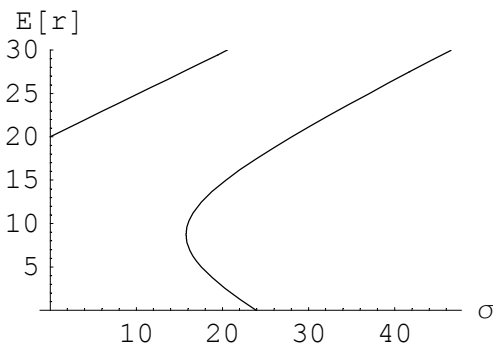
plots just the Capital Market efficiency frontier

Note: If  $s$  is the symbol, then the parametric representation uses  $\sigma = \sigma[s]$  and  $r = \mu[s]$ . Note: For CAPMParametricPlot: give {s\_Symbol, b, e} instead of only the symbol for domain control. Note: CAPMPlot calculates the market portfolio, and the various results are in Results[CAPMPlot].

- An example of domain testing is:

```
CAPMPlot[20, FrontierExample[r], {0, 30}, r];
```

```
MPFromParametric::dom : Solution  $\propto$  isn't in {0, 30}
```

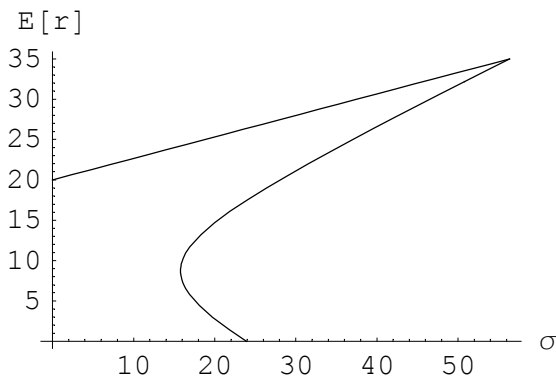


- You don't normally see graphs like the following. If we drop the short selling assumption, then the weight is bounded, then the efficiency frontier is bounded, and then the Capital Market Line is not necessarily tangent to the market portfolio efficiency frontier but it can connect with the extreme point. (Note: We include an arbitrary dimension factor (arbitrarily put at 35) to create another frontier.)

```
SetOptions[MPFromParametric, Domain -> True];

CAPMParametricPlot[20, FrontierExample[Weight 35], {Weight, 0, 1}];

MPFromParametric::dom : Solution  $\omega$  isn't in {0, 1}
```



The following plot can come in handy - though it is little else than Plot with some preset options. It is demonstrated in section 5.14.8 below.

CAPMWeightsPlot[  
w\_List, {s\_Symbol, b, e}]

plots the w(s)

5.14.6.2 Scatter

Let us consider the following data on rates of return, and let us add a randomly generated series that we force to have a specific mean and spread to create more dispersion in the datamatrix. Note that the original data are in perunages, and that we change them into percentages.



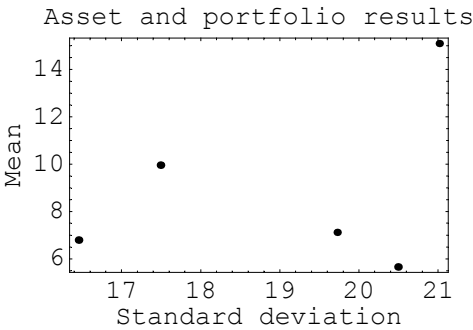
```
AddRandomSeries = False; (*set to True or None if y
ou want to*)
years = Table[i, {i, 1989, 1994}];
level = 100;
datamat =
  {Robeco = level {0.137, -0.166, 0.112, 0.096, 0.3, -0.071},
  Rolinco = level {0.167, -0.224, 0.183, 0.049, 0.325, -0.073},
  MSCI = level {0.123, -0.263, 0.204, 0.011, 0.317, -0.052},
  Holland = level {0.275, -0.131, 0.187, 0.079, 0.472, 0.023},
  If[AddRandomSeries,
  ran = level DrawNormal[{0.1, 0.16}, Length[years], -1],
  ran = level {0.122, 0.332, -0.0833, -0.0067, -0.0459, 0.280},
  ToSequence[]];
```

- Let us take the following market portfolio. Note that StandardDeviation divides by n-1, while Spread divides by n. (We still use the label Spread in texts and graphs.)

```
{sigmaM, rM} = ({StandardDeviation[#1], Mean[#1]}&)[Holland]
{21.0145, 15.0833}
```

<code>AssetScatterPlot[{{x1, y1}, ...}]</code>	plots {xi, yi} with default frame options
<code>AssetScatterPlot[Spread, {r_List}, opts]</code>	plots the {σi, ri} space
<code>AssetScatterPlot[Spread, means, covar, opts]</code>	idem, using the means and covar matrix

```
AssetScatterPlot[Spread, datamat];
```



5.14.7 Portfolios

Without commitment to efficiency, it is possible to calculate the return and spread of a portfolio using the data on the assets. We can find separate assets (e.g. for scatters) or the total weighted portfolio.

```
PortfolioSigmaMu[rs_List, cov?MatrixQ]
```

gives the points in the  $\{\sigma, r\}$  space for mean returns rs

```
PortfolioSigmaMu[rs_List, cov?MatrixQ, w_List]
```

evaluates a portfolio of weights w of assets with expected returns rs and covariance matrix c. Output is  $\{\sigma, r\}$

```
PortfolioSigmaMu[rs_List, cov?MatrixQ, n_Integer]
```

selects the results of only the n-th asset

```
PortfolioSigmaMu[rs_List, cov?MatrixQ, w_Symbol]
```

uses an array of w as weights (preferably use Weight)

- Using the data presented above, and let us divide the covariances with n-1.

```
means = Mean /@ datamat;
```

```
covarm = CovarMat[datamat, -1];
```

```
PortfolioSigmaMu[means, covarm]
```

$$\begin{pmatrix} 16.4549 & 6.8 \\ 19.7241 & 7.11667 \\ 20.492 & 5.66667 \\ 21.0145 & 15.0833 \\ 17.4916 & 9.96833 \end{pmatrix}$$

```
PortfolioSigmaMu[means, covarm, {.4, .1, .5, 0, 0}]
```

```
{18.5924, 6.265}
```

#### 5.14.8 CAPMSolve: The Markowitz efficient frontier

The efficiency frontier should include all assets, but one may of course only look at smaller portfolios.

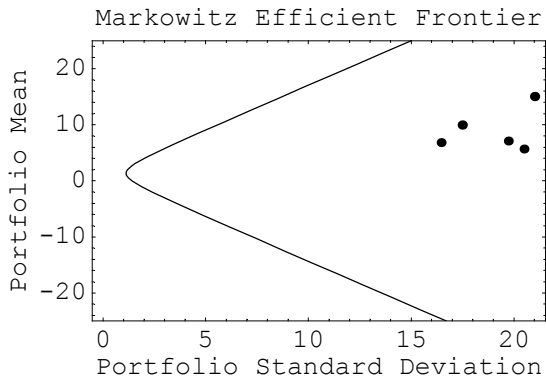
<code>FrontierAnalysis[</code> <code>{r1, r2, ...} , opts]</code>	for ri the list of rates of returns of asset i, gives the means and covariances of the data and the Markowitz Efficient Frontier
<code>Frontier[r]</code>	should give a parametric representation of the Markowitz efficiency frontier $\{\sigma, r\}$

Note: The default options are to plot and to print the frontier formula; option Condition → Short allows short selling; and option N → -1 divides the covariances by n-1. The Results → label option gives Results[label, Solve | Plot | Data], with default label FrontierAnalysis. If no short selling is allowed, then you can also check Options[KuhnTucker].

- The random series is negatively correlated with the others that are all positively correlated. Inclusion of this random series creates a far out frontier, by happenstance rather pointed.

```
FrontierAnalysis[{Robeco, Rolinco, MSCI, Holland, ran},
PlotRange → {-25, 25}, Results → Short]
```

The Efficient Frontier is:  $\{\sqrt{0.399967 r^2 - 1.08764 r + 1.99919}, r\}$

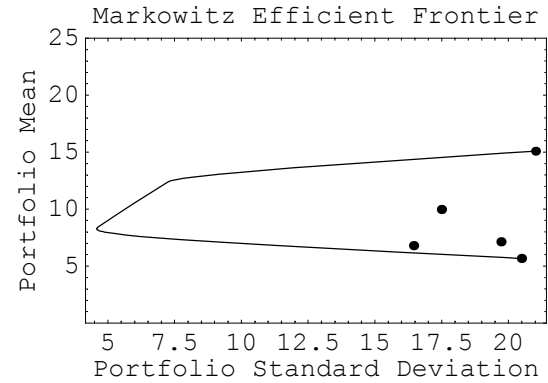


{Mean → {6.8, 7.11667, 5.66667, 15.0833, 9.96833},

CovarMat → 
$$\begin{pmatrix} 270.764 & 317.82 & 320.17 & 332.752 & -245.383 \\ 317.82 & 389.042 & 399.897 & 401.352 & -294.977 \\ 320.17 & 399.897 & 419.923 & 408.879 & -304.759 \\ 332.752 & 401.352 & 408.879 & 441.61 & -262.973 \\ -245.383 & -294.977 & -304.759 & -262.973 & 305.956 \end{pmatrix}$$

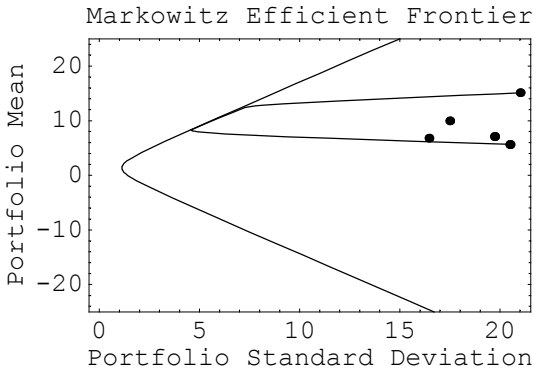
- If we drop the short selling assumption then the efficient area is much smaller. The following represses printing of the frontier (that reads awkwardly). And the calculation is slower.

```
FrontierAnalysis[{Robeco, Rolinco, MSCI, Holland, ran}, Condition → Automatic, Print → False, Results → NonNegative];
```



- This combines the two plots, to show the effect of short selling versus nonnegative weights.

```
Show[Results[Short, Plot], Results[NonNegative, Plot]];
```



- The frontier is a Which object. Read appendix A.11 on this.

```
Frontier[r]
```

```
{√ Which[5.66667 ≤ r ≤ 7.70197, 72.1667 r2 - 1154.82 r + 4646.55, 7.70197 ≤ r ≤ 8.31906,  
38.2643 r2 - 632.589 r + 2635.45, 8.31906 ≤ r ≤ 8.39043, 1.7749 r2 - 25.4746 r + 110.136,  
8.39043 ≤ r ≤ 8.70945, 30.2729 r2 - 503.695 r + 2116.37, 8.70945 ≤ r ≤ 9.3621,  
158.012 r2 - 2728.77 r + 11805.9, 9.3621 ≤ r ≤ 12.3562, 0.561037 r2 - 3.85892 r + 14.3315,  
12.3562 ≤ r ≤ 15.0833, 48.6757 r2 - 1192.89 r + 7360.25, True, Indeterminate], r}
```

The routine that calculates the frontier is CAPMSolve. This same routine also calculates the market portfolio that we discuss in next section.

```
CAPMSolve[Frontier][rs_List, cov?MatrixQ, r_Symbol]
```

determines Frontier[r] and FrontierWeights[r] for uncertain assets with expected values rs and covariance matrix cov

```
CAPMSolve[Frontier, Short][rs_List, cov?MatrixQ, r_Symbol]
```

idem for short selling and (thus, possibly,) negative weights

```
CAPMSolve[MarketPortfolio][rs_List, cov?MatrixQ, rf]
```

gives the optimal weights MPWeights[rf] and MarketPortfolio[rf] = {sigmaM, evM} depending upon certain rate rf

```
CAPMSolve[MarketPortfolio, Short][rs_List, cov?MatrixQ, rf]
```

idem with going short

Note: The versions with nonnegative weights call the KuhnTucker routine, and can give complicated results, in particular if input is symbolic.

- This reproduces the example by Luenberger (1998:160) of three assets with expectations {1, 2, 3} and unit variances and zero covariances.

```
CAPMSolve[Frontier, Short][{1, 2, 3}, DiagonalMatrix[{1, 1, 1}], r]
```

$$\left\{ \text{Frontier} \rightarrow \left\{ \sqrt{\frac{r^2}{2} - 2r + \frac{7}{3}}, r \right\}, \text{FrontierWeights} \rightarrow \left\{ \frac{4}{3} - \frac{r}{2}, \frac{1}{3}, \frac{1}{6}(3r - 4) \right\} \right\}$$

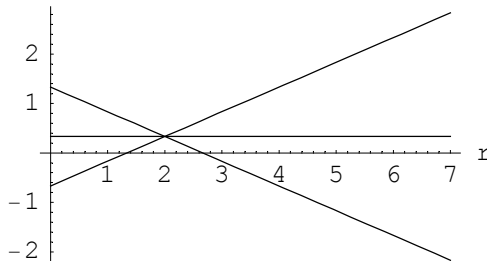
- This would be a plot (not evaluated).

```
p1 = CAPMParametricPlot[Frontier[r], {r, 1, 3}];
```

- The weights are directly available, and can be plotted too.

```
CAPMWeightsPlot[FrontierWeights[r], {r, 0, 7}];
```

Weights



- Note that we get the same result from a Lagrange call (minimising the variance now, just for printing).

```
ws = {x, y, z};
rs = {1, 2, 3};
G = {1 - Add[ws], r - ws . rs};
cov = DiagonalMatrix[{1, 1, 1}];

Lagrange[Expression → ws.cov.ws, Constraints → G, Variables → ws,
  Routine → Solve] // Simplify
```

*Lagrange::first: Only first order conditions used*

```
(  $\frac{r^2}{2} - 2r + \frac{7}{3}$  { $x \rightarrow \frac{4}{3} - \frac{r}{2}$ ,  $y \rightarrow \frac{1}{3}$ ,  $z \rightarrow \frac{1}{6}(3r - 4)$ ,  $\text{Mu}(1) \rightarrow 2r - \frac{14}{3}$ ,  $\text{Mu}(2) \rightarrow 2 - r$ })
```

- This is the result with nonnegativity restrictions on the frontier weights (no short selling).

```
CAPMSolve[Frontier][{1, 2, 3}, DiagonalMatrix[{1, 1, 1}], r]
```

```
{Frontier → { $\sqrt{\text{Which}[1 \leq r \leq \frac{4}{3}, 2r^2 - 6r + 5, \frac{4}{3} \leq r \leq \frac{8}{3},$ 
 $\frac{r^2}{2} - 2r + \frac{7}{3}, \frac{8}{3} \leq r \leq 3, 2r^2 - 10r + 13, \text{True}, \text{Indeterminate}]$ ,  $r$ },
FrontierWeights →  $\text{Which}[1 \leq r \leq \frac{4}{3}, \{2 - r, r - 1, 0\}, \frac{4}{3} \leq r \leq \frac{8}{3},$ 
 $\{\frac{4}{3} - \frac{r}{2}, \frac{1}{3}, \frac{1}{6}(3r - 4)\}, \frac{8}{3} \leq r \leq 3, \{0, 3 - r, r - 2\}, \text{True}, \{\}$ }}
```

- This would be a plot, in comparison with the short selling plot (not evaluated).

```
p2 = CAPMParametricPlot[Frontier[r], {r, 1, 3}];

Show[p1, p2]
```

### 5.14.9 CAPMSolve: The market portfolio

The market portfolio is determined by the maximum slope that starts at  $\{0, r_f\}$ . The required formats have been mentioned in the former section.

If short selling is allowed then there is a straightforward solution that also allows easy formal treatment. With nonnegativity constraints, this paradise is lost to all kinds of inequalities.

- This uses the same example, now at a certain rate of 0.5% (Luendberger (1998: 168)).

```
CAPMSolve[MarketPortfolio, Short][{1, 2, 3}, DiagonalMatrix[{1, 1, 1}],
1/2]
```

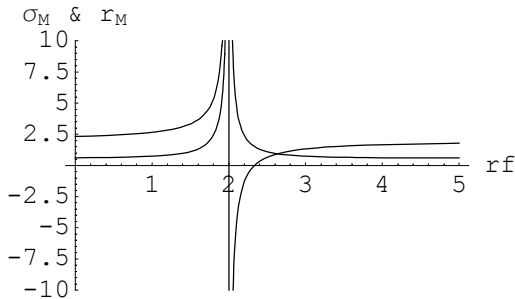
```
{MarketPortfolio → { $\frac{\sqrt{35}}{9}, \frac{22}{9}$ }, MPWeights → { $\frac{1}{9}, \frac{1}{3}, \frac{5}{9}$ }}
```

- It is also possible to use a symbolic certain rate, and plot the result for some domain.

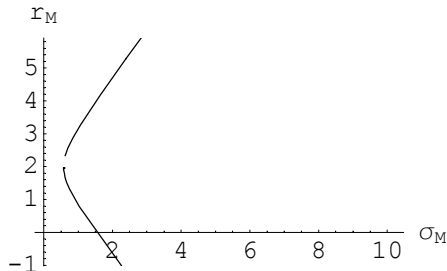
```
CAPMSolve[MarketPortfolio, Short][{1, 2, 3}, DiagonalMatrix[{1, 1, 1}], rf]
```

$$\left\{ \text{MarketPortfolio} \rightarrow \left\{ \frac{1}{3} \sqrt{\frac{3rf^2 - 12rf + 14}{(rf-2)^2}}, \frac{14-6rf}{6-3rf} \right\}, \text{MPWeights} \rightarrow \left\{ \frac{rf-1}{3(rf-2)}, \frac{1}{3}, \frac{rf-3}{3(rf-2)} \right\} \right\}$$

```
Plot@@{MarketPortfolio[rf], {rf, 0, 5},
  AxesLabel -> {rf, "σM & rM"}, PlotRange -> {-10, 10}};
```



```
ParametricPlot[MarketPortfolio[rf], {rf, 0, 20}, AxesLabel -> {"σM", "rM"}];
```



- If we don't allow short selling, the market portfolio will consist of only the third asset. Note that this routine can generate more solution points, so we have lists of points.

```
CAPMSolve[MarketPortfolio][{1, 2, 3}, DiagonalMatrix[{1, 1, 1}], 7]
```

```
{MarketPortfolio -> (1 3), MPWeights -> (0 0 1)}
```

If you were to evaluate the former problem with a formal fixed rate  $r_f$  then you will find that this solution has to be put into Hold, since different domains can apply for different values of  $r_f$ .

#### 5.14.10 Regression

The 'characteristic equation' of an asset is defined as the premium regression equation with a constant, i.e.  $\pi_i = \beta_i \pi_M + \alpha_i + e$ . The asset's beta is the regression coefficient of the expected premium of the asset on the expected market premium. The constant is called the asset's  $\alpha$ . The CAPM theorem tells that optimality is reached when  $\alpha$  is zero. The Jensen Index takes the  $\alpha$  as a measure of closeness to optimality.

A subtle point is that the CAPM concerns the *expected* returns and premia of assets, and these are not the observed returns commonly used in regressions. Perhaps we should write the CAPM relation with  $E[r_i]$  and  $E[r_M]$  instead of  $r_i$  and  $r_M$ . Similarly, the observed covariance of datapoints of an asset and the market portfolio may be a bad estimate of the proper covariance that the theorem refers to.

Note 1: If one would regress with  $\alpha = 0$  *a priori*, then a different  $\beta$  value is found than with free regression.

Note 2: The CAPM relation can be seen as  $r_i = \gamma + \beta_i r_M$ , so that a regression with a constant is possible, and an estimated certain return  $r_f' = \gamma / (1 - \beta_i)$  can be recovered. This seems yet little more than a mathematical property and little else. Having more assets would generate many of those  $(r_f')_i$ , and what then ? It would be a strong assumption that these reflect the expected certain returns for the investors participating in these specific assets.

Note 3: Don't forget about the premium ratio  $\pi_R \equiv \pi_i / \pi_M$ . For a timeseries, one would take geometric averages of the rates of return to find the best estimate of the premia. Alternatively put: regressing could be a wrong way to determine the period average - unless it would be a superior way to get from observations to expectations.



<code>CAPMFit[rf, rm, r]</code>	regresses $r - rf =$ alpha + beta (rm - rf) and renders {alpha, beta}. Also defines CAPMFit[x] that for $x = rm - rf$ gives $r - rf$
<code>AssetAlpha</code>	Symbol for the alpha of a CAPM regression, $(r - rf) = \text{AssetBeta} (rm - rf) + \text{AssetAlpha}$
<code>AssetAlpha[ rf, rm_List, r_List]</code>	gives the $\alpha = (\text{mean}[r] - rf) - \text{beta}[r] (\text{mean}[rm] - rf)$
<code>beta</code>	defined as Greek $\beta$
<code>AssetBeta</code>	Symbol for the Beta of a CAPM regression, $(r - rf) = \text{AssetBeta} (rm - rf) + \text{AssetAlpha}$
<code>AssetBeta[ rm_List, r_List]</code>	calculates the asset beta from the covariance with rm. Both variance and covariance divide by n-1
<code>AssetBeta[ rm_List, {r_List}]</code>	does this for all r
<code>PremiumRatio[ rf, rm, r]</code>	gives the asset's beta in terms of its expected return r, the certain or fixed rate rf, and the expected return rm of the market portfolio
<code>PremiumRatio[ rf_List, rm_List, r_List]</code>	takes geometric averages first, which may presume that the asset is compared to short term bonds
<code>PremiumRatio[ rf, rm_List, r_List]</code>	presumes that the comparison is for a long term bond of the same duration
<code>PremiumRatio[rf_?(!ListQ[#]&amp;), rm_List, {r_List}]</code>	calculates the betas of more assets, including the implied covariances

Note: *Mathematica's* Variance[] routine divides by n-1 (while VarianceMLE divides by n). So the covariance must be computed with n-1 too, to get the proper betas. This is done in AssetBeta, while CAPMFit uses Fit, and gives the same result.

#### ■ Calling Fit:

```
(CAPMFit[7, Holland, #1] & ) /@ {Robeco, Rolinco, MSCI}
```

```
(-6.29078 0.753498 )
(-7.22979 0.90884 )
(-8.81756 0.925884 )
```

```
CAPMFit[x - rf]
```

```
0.925884 (x - rf) - 8.81756
```

- Fitting without a certain rate gives an implied certain rate, as constant /  $(1 - \beta)$

```
CAPMFit[Holland, Robeco]
```

```
{AssetBeta → 0.753498, Constant → -4.56526, CertainRate(Implied) → -18.5202}
```

- If we use the normal covariances, then we can find these betas.

```
AssetBeta[Holland, {Robeco, Rolinco, MSCI}]
```

```
{Method → Covar, Market(Mean) → 15.0833, Market(Spread) → 21.0145,  
Covar → {332.752, 401.352, 408.879}, AssetBeta → {0.753498, 0.90884, 0.925884}}
```

- Conversely, if we presume optimality and regard the premium ratio, then we can find an implied covariance from this and the market variance. Let us take the geometric means as the best estimators of the expectation, and the normal variance of the market portfolio around this mean. The following uses a certain rate of 7%, and from it we determine the premium ratios  $\pi_i / \pi_M$ . Note that geometric averaging is sensitive to levels, so we have to use perunages.

```
PremiumRatio @@ {1/level {7, Holland, {Robeco, Rolinco, MSCI}}}
```

```
{Method → Definition, Market(GeometricMean) → 0.134917,  
Market(GeometricSpread) → 0.192494, Asset(GeometricMean) → {0.0571424, 0.0551918, 0.0389704},  
Covar → {-0.00733898, -0.00845236, -0.0177114},  
PremiumRatio → {-0.198061, -0.228108, -0.477986}}
```

- The premium ratio's thus differ strongly from the beta's. Let us check for Robeco:

```
GeometricMean[Robeco/level + 1] - 1
```

```
0.0571424
```

```
PremiumRatio[0.07, 0.134917, %]
```

```
-0.198062
```

The numbers fit, and apparently the data don't reflect efficiency. The premium ratio conveys quite some information. For example, the rate of return of Robeco was indeed less than the certain rate, and while its premium ratio should be 1 to keep up with the Dutch average, it was actually negative.

### 5.14.11 Security Market Line

The security market line has two representations.

- The security market line gives the expected return of an asset or portfolio as a function of its beta ...

```
ExpectedReturn[1] ==
  CertainRate + AssetBeta[1] (ExpectedReturn[Market] - CertainRate) /.
  ToCAPMSymbols /. 1 -> i
```

$$r_i == r_f + (r_M - r_f) \beta_i$$

- ... or its covariance with the market portfolio.

```
% /.  $\beta_i \rightarrow \text{Covar}["i", "M"] / \text{Spread}[\text{Market}]^2 /. \text{ToCAPMSymbols}$ 
```

$$r_i == r_f + \frac{(r_M - r_f) \sigma_{i,M}}{\sigma_M^2}$$

Of these, only the beta plot has been implemented.

SecurityMarket[ Line, beta_Symbol, rf, evM]	gives the security market line for the beta; defines SecurityMarket[x]
SecurityMarket[Line, cov_Symbol, rf, sigmaM, evM]	idem for the covariance; defines SecurityMarket[Covar, x]
SecurityMarket[Data, rm_List, r_List]	gives the {Beta[r, rm], Mean[r]} point
SecurityMarket[ Data, rm_List, {r_List}]	does this for the r's
SecurityMarket[ Plot, rf, rm_List, {r_List}]	plots the line and the {betai, meani} points
SecurityMarket[Fit, rm_List, r_List, opts]	fits, can plot. Tip: subtract rf first
SecurityMarket[Fit, rm_List, {r_List}, opts]	does this for the r's; plotting now concerns only the betas and means
SecurityMarket[Plot, Fit, rf, rm_List, {r_List}, opts]	fits and plots

Note: The Fit versions use the Options to control printing (the line equation) and plotting. Note: In the final Plot Fit there are two lines in the plot: the proper Security Market line for the market portfolio, and the fit through the {beta, mean} observations. The difference between these lines gives a measure for the efficiency of the market

- Thus, using data set above:

```
SecurityMarket[Line, beta, 5, rM]
```

$$\text{ExpectedReturn} == 10.0833 \beta + 5$$

```
SecurityMarket[x]
```

$10.0833x + 5$

- This is actually a variant of CAPMFit.

```
SecurityMarket[Data, Holland, {Robeco, Rolinco, MSCI}]
```

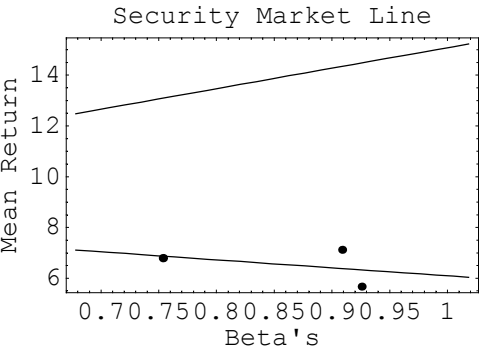
$$\begin{pmatrix} 0.753498 & 6.8 \\ 0.90884 & 7.11667 \\ 0.925884 & 5.66667 \end{pmatrix}$$

- This plot has two lines: (1) The proper market security line. Note that  $\beta = 1$  gives the market portfolio, and that the distance of a point to the line is the  $\alpha_i$  (the Jensen Index). (2) The fit through the observed {beta, mean} points. The differences of these two lines give a measure for the efficiency of the market. Note: The lines are available as SecurityMarket[x] and CAPMFit[x].

```
SecurityMarket[Plot, Fit, 7, Holland, {Robeco, Rolinco, MSCI}];
```

The security market line is:  $8.08333\beta + 7$

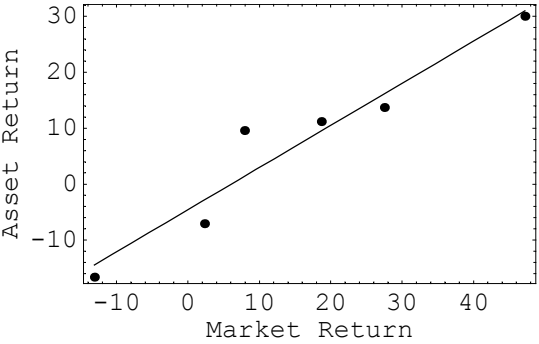
The {beta, mean} fit is:  $9.25111 - 3.15661\beta$



- In case we want to compare only one asset with the market portfolio. Notice the different meanings of the axes.

**SecurityMarket[Fit, Holland, Robeco];**

The fit is:  $0.753498x - 4.56526$



### 5.14.12 Performance

We can also compare the performance of a managed fund that has results  $\{\sigma, r\}$ , to the result of investing with CAPM, using certain rate  $r_f$ , market results  $\{\sigma_M, r_M\}$  and target the same spread  $\sigma$ . The performance is measured by the ratio  $r / r[\text{CAPM}]$ .

**Performance** $[r_f, \{\sigma_M, r_M\}, \{\sigma, r\}]$  gives the  $\beta$ , the expected  $r[\text{CAPM}]$ , and performance of  $r$

**Performance** $[r_f, \{\sigma_M, r_M\}, \{\sigma, r\}]$

$$\left\{ \beta \rightarrow \frac{\sigma}{\sigma_M}, \text{ExpectedReturn} \rightarrow r_f + \frac{\sigma(r_M - r_f)}{\sigma_M}, \text{Performance} \rightarrow \frac{r}{r_f + \frac{\sigma(r_M - r_f)}{\sigma_M}} \right\}$$

**Performance** $[5, \{2.5, 8\}, \{3.1, 9\}]$

$\{\beta \rightarrow 1.24, \text{ExpectedReturn} \rightarrow 8.72, \text{Performance} \rightarrow 1.03211\}$

Another measure of performance is the Sharpe Index, or reward-to-spread ratio, or the price of spread.

**SharpeIndex** $[r_f, \{\sigma, r\}]$  gives the Sharpe Index  
(reward to spread ratio, the price of spread)  
for a single asset

**SharpeIndex** $[r_f, \{\sigma_M, r_M\}, \{\sigma, r\}]$  divides this with the Market Sharpe Index

**SharpeIndex** $[5, \{2.5, 8\}, \{3.1, 9\}]$

$\{\text{Asset} \rightarrow 1.29032, \text{Market} \rightarrow 1.2, \text{Ratio} \rightarrow 1.07527\}$

The Jensen index actually is the  $\alpha$  of the CAPM fit.

```
AssetAlpha[7, Holland, Robeco]
```

```
-6.29078
```

## 5.15 Finance enhancement

### 5.15.1 Summary

The `Finance`` package provides some routines for enhancement of the WRI Finance Essentials (FE) pack written by Leszek Sczaniecki, version June 1999.

- Note: The `Finance`` package of the Economics Pack calls all finance related packages, including all packages of the Finance Essentials package, and makes sure that the packages are in the right order in the `$ContextPath`. You could basically neglect all shadowing messages.

**Economics[Finance]**

### 5.15.2 Order of contexts

Calling `Finance`` causes a number of shadowing messages. For example, `Bond` occurs in `Cool`Finance`Common`` and `Finance`Bonds``. There also is shadowing of the `Finance`Calendar`` package with the standard *Mathematica* package `Miscellaneous`Calendar``. The various packages are best put into a specific order on the `$ContextPath`, and this order is warranted by the use of `F$ContextPath[ ]` - a routine called automatically by `Finance``.

`F$ContextPath[ ]`      adjusts the `$ContextPath` such that the order is { ... `Cool`Finance``, ..., `Cool`Time``, ..., `Finance`Calendar``, ..., `Miscellaneous`Calendar``, ...}.

Note that *Mathematica* 4.0 has a symbol `Value` in the `System`` context, that is replaced by Finance Essentials. In *Mathematica* 3.0, there is no `Value` in the `System`` context, but there is one in the `Cool`Common`` context. The Pack looks at your `$Version` here.

- Just to check that Finance Essentials is properly present, this reproduces a result on p22 of the FE booklet:

**?Bond**

```
Bond[{coupon, maturity, par,
      frequency}] is an object representing a bond.
```

```
bond10 = Bond[{10 Percent, F[9, 15, 2003], 1000, 2}]
```

```
Bond[{1/10, F(9, 15, 2003), 1000, 2}]
```

```
Payment[bond10, F[9, 15, 1993], 0.15] /. F -> List
```

```
745.138
```

```
Val [%]  
  
Dollar 745.14
```

5.15.3 Graphics

Since Finance Objects have a similar structure, we can use the same plotting functions.

ShowFO [ <i>x</i> ]	shows Finance Objects <i>x</i> , single or list of those
FinanceGO [ <i>x</i> ]	turns a Finance Object <i>x</i> into a Graphics Object

5.15.4 New objects

5.15.4.1 Dated DiscountFactors

The following dated DiscountFactors object allows control of when the factors are applied.

DiscountFactors [{F[...], factor1}, {F[...], factor2}, ... ]
is an interest rates object with financial dates and discount factors. The initial date must have discount factor 1.

Use ?DatedDiscountFactors for information.

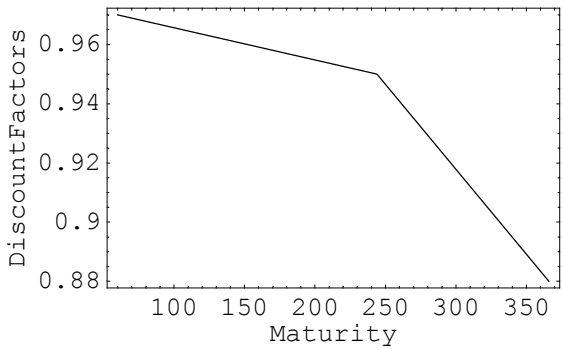
```
ddf = DiscountFactors [{F[1, 1, 2001], .88},  
                        {F[1, 1, 2000], 1. },  
                        {F[3, 1, 2000], .97},  
                        {F[9, 1, 2000], .95} } // FSort ]  
  
DiscountFactors  $\left( \begin{pmatrix} F(1, 1, 2000) & 1. \\ F(3, 1, 2000) & 0.97 \\ F(9, 1, 2000) & 0.95 \\ F(1, 1, 2001) & 0.88 \end{pmatrix} \right)$ 
```

FOToPeriods [ <i>dfo</i> , <i>date</i> :Automatic]
transforms a dated finance object <i>dfo</i> into an object with actual days in between. The first date in <i>dfo</i> is the initial date unless <i>date</i> is specified

```
df = FOToPeriods [ddf]  
  
DiscountFactors  $\left( \begin{pmatrix} 60 & 0.97 \\ 244 & 0.95 \\ 366 & 0.88 \end{pmatrix}, \text{InitialDate} \rightarrow F(1, 1, 2000) \right)$ 
```



`ShowFO[df];`



`IntervalSumFR[`      cumulates the subperiods of the forward rates `x` to find the duration  
`x]`

- The `ForwardRates` transformation gives day interest rates. Note that 2000 is a leap year.

`ForwardRates[ df ]`

$$\text{ForwardRates}\left(\left(\begin{pmatrix} 60 & 0.000507782 \\ 184 & 0.000113235 \\ 122 & 0.000627575 \end{pmatrix}, \text{InitialDate} \rightarrow F(1, 1, 2000)\right)\right)$$

`IntervalSumFR[%]`

366

`InterpolateDF[d]`      interpolates the whole range of the discount factors object `d`

- Though real months are not 30 days long:

$$\text{dfInMonths} = \text{df} /. \{ \mathbf{x\_Integer}, \mathbf{y\_Real} \} \rightarrow \{ \mathbf{x} / 30, \mathbf{y} \}$$

$$\text{DiscountFactors}\left(\left(\begin{pmatrix} 2 & 0.97 \\ \frac{122}{15} & 0.95 \\ \frac{61}{5} & 0.88 \end{pmatrix}, \text{InitialDate} \rightarrow F(1, 1, 2000)\right)\right)$$

`InterpolatedDF[ dfInMonths ]`

DiscountFactors	1	0.984886
	2	0.97
	3	0.966711
	4	0.963432
	5	0.960165
	6	0.956909
	7	0.953664
	8	0.95043
	9	0.934629
	10	0.917203
	11	0.900101
	12	0.883319

For formal interpolations:

`SingleInterpolatedDF[{ {t, f}, {T, F}}, time]`  
  
gives a single interpolation for discount factors that may be in symbols

`SingleInterpolatedDF[{ {t, f}, {T, F}}, time]`

$$\left\{ \text{time}, f \left( \frac{F}{f} \right)^{\frac{\text{time}-t}{T-t}} \right\}$$

5.15.4.2 CashStocks

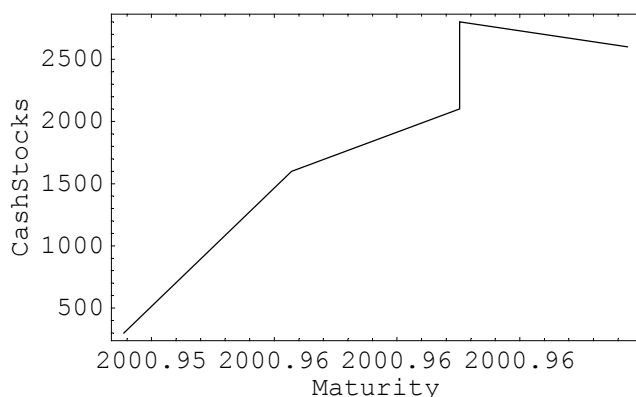
For a simple cash position at the cash register, one explicitly should not discount.

`CashStocks[{ {time1, amount1}, {time2, amount2}, ...}]`  
  
is an object for a cash register, the cumulation of cash flow  
  
`CashCumulate[x]`                      cumulates a CashFlows object into a CashStocks object  
  
`CashDifference[x]`                    changes a CashStocks object into a CashFlows object

```
flow = CashFlows[{ {Ft[December, 15, 2000], 300 },
                  {Ft[December, 16, 2000], 1300},
                  {Ft[December, 17, 2000], 500},
                  {Ft[December, 17, 2000], 700},
                  {Ft[December, 18, 2000], -200}  ]];

cumflow = CashCumulate[flow];
```

ShowFO[cumflow];



### 5.15.5 Bonds

- Below we shall use these settings.

```
settlement = F[January, 1, 2000];
bond = Bond[ {.10, F[January, 1, 2002], 1000 Dollar, 2} ];
subst = {Dollar → 1, F → List};
```

#### 5.15.5.1 Understanding the FE Bond object

There are some crucial points to understand about the FE Bond object.

- Value == Accrued Interest + Payment. (Confusing is that if you do ?Value, then it says that Value and Payment are synonyms; you should neglect this.)
- The YieldToMaturity is determined on Payment, and thus neglects the interest. The price attributed to the bond thus also neglects the interest. The yield also is a nominal one, and not effective.
- The value function uses a settlement date but the calculation uses equal intervals, while the rate is a nominal one. The FE booklet shows this on page 32-33, by converting a Bond to a CashFlows object with equal intervals, and applying the effective rate of interest.

Given this, the following ProperValue function translates a Bond first into a CashFlows object and requires an effective rate as input.

```
EffectiveValue[bond_Bond, settlement, rate, datedq:True]
```

gives the present value of the bond,  
that arises from converting to a cash flows object,  
and with rate the effective rate of interest. If dated ==  
True then actual dates are used, otherwise equal intervals

```
EffectiveValue[x_] is just Value[x] for other objects
```

```
BondToCashFlows[bond_Bond, d_F, datedq:True]
```

default dated; if datedq = False then in cumulated period form

Options[PortfolioValue] currently have Value → EffectiveValue.

- This uses equal intervals and nominal rate 12%.

```
Value[bond, settlement, 0.12] /. F → List
```

965.349 Dollar

- This uses calendar intervals and effective rate 12% per annum.

```
EffectiveValue[bond, settlement, 0.12]
```

971.159 Dollar

- Above relies on the transformation:

```
cf = BondToCashFlows[bond, settlement]
```

$$\text{CashFlows} \left( \begin{pmatrix} F(7, 1, 2000) & 50. \text{ Dollar} \\ F(1, 1, 2001) & 50. \text{ Dollar} \\ F(7, 1, 2001) & 50. \text{ Dollar} \\ F(1, 1, 2002) & 1050. \text{ Dollar} \end{pmatrix} \right)$$

```
BondToCashFlows[bond, settlement, False]
```

$$\text{CashFlows} \left( \begin{pmatrix} \frac{1}{2} & 50. \text{ Dollar} \\ 1 & 50. \text{ Dollar} \\ \frac{3}{2} & 50. \text{ Dollar} \\ 2 & 1050. \text{ Dollar} \end{pmatrix}, \text{InitialDate} \rightarrow F(1, 1, 2000) \right)$$

The difference can also be shown formally. Let the coupon rate be  $i$ , and market rate  $r$ . With maturity in 2005, and annual payments, the difference in value is caused by 2004 being a leap year. If there were more payments per year the difference would be larger.

```
b = Bond[{i, {1, 1, 2005}, w, 1}];
```

`Value[b, {1, 1, 2001}, r]`

$$\frac{\left(i\left(1 + \frac{1}{r+1} + \frac{1}{(r+1)^2} + \frac{1}{(r+1)^3}\right) + \frac{1}{(r+1)^3}\right)w}{r+1}$$

`EffectiveValue[b, {1, 1, 2001}, r]`

$$\frac{iw}{r+1} + \frac{iw}{(r+1)^2} + \frac{iw}{(r+1)^{400769/133590}} + \frac{iw+w}{(r+1)^4}$$

### 5.15.5.2 Yield to maturity

- The "price" (payment) of a Bond here is exclusive of the accrued interest. The yield to maturity also is based upon this. This yield also is a nominal one, and not effective.

```
pm = Payment[bond, settlement, .07] /. subst
```

```
1055.1
```

```
YieldToMaturity[bond, pm, settlement] /. subst
```

```
0.07
```

To better understand the above, we can use the `FlatRate`` package.

`BondToCashFlow[b_Bond, d_F]` creates a `FlatRate` `CashFlow` object with full periods till payment days

- Translate above into the simple flat rate `CashFlow` object.

```
btc = BondToCashFlow[bond, settlement] /. Dollar -> 1
```

```
CashFlow(50., 4, 1000)
```

- The `FlatRate`` `Yield` function gives 3.5%. We conclude that the `Finance Essentials` `Value` function uses a nominal rate of interest for bonds (the 7% generated by `YieldToMaturity` should be divided by the frequency to find the effective rate for a subperiod).

```
Yield[btc, pm]
```

```
0.035
```

- Note: The `FlatRate`` present value was used to find the internal rate of interest.

```
pm == PV[btc, r]
```

$$1055.1 == \frac{50.\left(1 - \frac{1}{(r+1)^4}\right)}{r} + \frac{1000}{(r+1)^4}$$

#### 5.15.5.4 Cash flow dates

The Finance Essentials Package `CashFlowsDates[bond, date]` function returns the future coupon payment dates of the bond. If the settlement falls on a cash flow date then it is counted. However, settlement can also take place after such payments have taken place.

`AfterCashFlowsDates[bond_Bond, settlement]`

gives the cashflow dates excluding settlement if that falls on a cashflow date

- Using the Finance Essentials routine, we find three cash flow dates:

`CashFlowsDates[bond, settlement] /. F → List`

$$\begin{pmatrix} 1 & 1 & 2000 \\ 7 & 1 & 2000 \\ 1 & 1 & 2001 \end{pmatrix}$$

- Finance Essentials however counts only two payments:

`NumberOfCouponPayments[bond, settlement] /. F → List`

2

- The following subroutine is also used to transform Bonds to CashFlows in EffectiveValue.

`AfterCashFlowsDates[bond, settlement] /. F → List`

$$\begin{pmatrix} 7 & 1 & 2000 \\ 1 & 1 & 2001 \end{pmatrix}$$

#### 5.15.6 Portfolio object

The Portfolio object was introduced in section 5.12. It now relies on the EffectiveValue function to base the valuing of Bonds and CashFlows on equal footing.

- `Portfolio[ ]` is just a holder. But we can apply the `Value[ ]` function to it. For example, we collect above bond and cashflow in it.

`pf = Portfolio[bond, cf] /. Dollar → 1`

$$\text{Portfolio} \left( \text{Bond}((0.1, F(1, 1, 2002), 1000, 2)), \text{CashFlows} \left( \begin{pmatrix} F(7, 1, 2000) & 50. \\ F(1, 1, 2001) & 50. \\ F(7, 1, 2001) & 50. \\ F(1, 1, 2002) & 1050. \end{pmatrix} \right) \right)$$

- If the settlement day still is January 1 2000, the Bond and Cashflows objects have equal value.

**Value**[**pf**, **F**[**January**, **1**, **2000**], **.07**]

{ClassValues → {1057.37, 1057.37}, Number → {1, 1},  
PortfolioWeights → {0.5, 0.5}, EffectiveValue → 2114.74}

- But if the settlement day is different, then the values differ, since the Cashflows object is fixed in time, and the Bond object isn't.

**Value**[**pf**, **F**[**January**, **1**, **1999**], **.07**]

{ClassValues → {1083.28, 988.199}, Number → {1, 1},  
PortfolioWeights → {0.52295, 0.47705}, EffectiveValue → 2071.48}

### 5.15.7 Utilities

**ToBond**[**CashFlow**[*pp*, *m*, *w*], *date*, *freq*:*1*]

creates a Bond object with maturity = date + m / freq years ahead.

**ToDatedCash**[**CashFlow**[*pp*, *m*, *w*(*g*)], *date*, *freq*:*1*, *datedq*:**True**]

creates a dated CashFlows object. If *datedq* is  
True then all payments are dated, otherwise equal intervals

Here, *date* is the initial date, *m* are the number of payments, and *freq* gives the number of payments per year. The routines use **PlusYears**, but make sure that the number of coupon payments fit **NumberOfCouponPayments**.

- This calculates the crucial maturity date.

**ToBond**[**CashFlow**[**50**, **5**, **1000**], **F**[**February**, **3**, **2000**], **3**]

**Bond**( $\left(\frac{1}{20}, F(10, 3, 2001), 1000, 3\right)$ )

- This **CashFlow** object translates into a **CashFlows** object with geometrically rising payments.

**ToDatedCash**[**CashFlow**[**50**, **3**, **1000**, **g**], **F**[**February**, **3**, **2000**], **2**]

**CashFlows** $\left(\left(\begin{array}{cc} F(8, 3, 2000) & 50 \\ F(2, 3, 2001) & 50(g+1) \\ F(8, 3, 2001) & 50(g+1)^2 + 1000 \end{array}\right)\right)$

## 5.16 Transport economics and transport science

---

### 5.16.1 Summary

These packages give many routines for transport economics and transport science. They are too specialised to fully document in this user guide. Only some topics are shown.

```
Economics[Transport, Freight, $Freight, Shipping, WorldMap, Physics,
TScience]
```

### 5.16.2 Introduction

In the period 1996-1998 I got involved in transport economics and transport science. I was asked to be the project leader for the Dutch Government freight projections 1997-2002, and had a role as a quality of life adviser in linking the technical performance of vehicles to the effects on the environment. Later I got a teaching position in operations management and transport, and turned much material into syllabi. Above packages are the residue of the role of *Mathematica* in these activities.

These packages are clearly beyond the interest of the average economist. The Economics Pack is a result of practical work and is intended to include such material, but we must be wise in our selection of topics, and it would lead too far, and add too many pages, if these packages were discussed in this guide. So we don't discuss them.

It does no harm, however, to include the software. The packages are available to you, and for documentation you can use the ? operation and the example notebooks. Those interested are advised to look at my internet site. That gives access to the syllabus "Transport economics for operations management" and the paper "An estimator for the road freight handling factor". The diagrams and models there have been developed with these packages. Currently, too, I am working with a colleague on the syllabus on "Transport science for operations management" (not on the internet).

It would be a wrong conclusion not to discuss transport at all. Some transport issues are of interest to the average economist. The following gives some excerpts with graphics.



Note, by the way, that in economics everything hangs together, and that this also happens to hold for the development of economic theory. There have been major influences of transport economics onto finance and other areas of economics. For example, Peter Bernstein, "Capital ideas", The Free Press 1992, gives a wonderful review of the development of finance, and indicates some such influences. For example, on p165: "Modigliani (of the MM theorem) also became friendly with the Dutch economist Tjalling Koopmans. Koopmans was then working in New York for a Dutch shipping company, where he was applying his new technique of linear programming to the company's scheduling problems." Note that linear programming was important for Harry Markowitz's theory. Another example is that Wells Fargo, the bank, originally started as a mail and packages services firm using their famous stagecoaches. Henry Wells and William Fargo also were founders of American Express Company.

### 5.16.3 Vehicle profits

There are many single person trucking companies where the driver is the owner of the vehicle. Regard such a company that maximises profits. Use the following symbols:

- $o$  = operating time, e.g.  $16 \times 365$  hours per year
- $q$  = quantity transported, e.g. carrying capacity  $\times$  loadfactor
- $p$  = price per weight distance, e.g. the competitive market price per tonnekilometer
- $pt$  = price per vehicle-duration, independent of speed, e.g. the competitive rent per hour
- $pf$  = price per speed dependent aspects, e.g. fuel price
- $a$  and  $b$  are parameters for speed dependent aspects, e.g. fuel use per hour

`ProfitPerVKM[{p, q, o}, {pt, pf, a, v_Symbol, b}]`

solves for maximal profit  $(p q o v - o (pt + pf a v^b))$  by speed  $v$

Note: You can enter  $a$  as  $a'[q]$  so dependency upon  $q$  is possible. Note that  $(o v)$  gives the operating distance. Note also that there is no explicit account for empty kilometers. See the full form for this.

- Regard a 10 tonne truck with a fixed cost of \$50 per day (or \$5 per operating hour) that uses diesel at \$ 0.25 per liter. Approximate the market freight rate at \$ 0.10 per tonnekilometer.

`ProfitPerVKM[{.1, 10, 10 365 5/7.}, {50. / 10, .25, .05, v, 2}]`

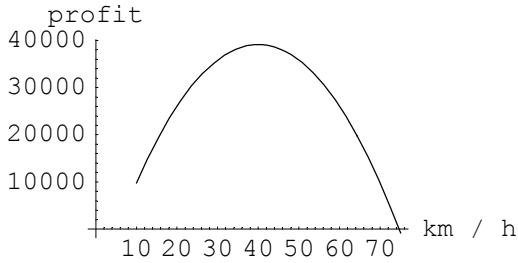
{Function  $\rightarrow 2607.14 v - 2607.14 (0.0125 v^2 + 5.)$ , OptimalSpeed  $\rightarrow 40.$ ,  
 OperatingDistance  $\rightarrow 104286.$ , OperatingWeightDistance  $\rightarrow 1.04286 \times 10^6$ , Revenue  $\rightarrow 104286.$ ,  
 Cost  $\rightarrow 65178.6$ , Profit  $\rightarrow 39107.1$ , Price  $\rightarrow \{0.1, 5., 0.25\}$ , UseAtOptimalSpeed  $\rightarrow 80.$ ,  
 Profit/OperatingTime  $\rightarrow 15.$ , UnitCost/Distance  $\rightarrow 0.625$ , UnitProfit/Distance  $\rightarrow 0.375$ }

- The optimal speed is 40 km / h ! Note how the trucker depends upon the technical properties of his vehicle.

**Function** /. %

2607.14 v - 2607.14 (0.0125 v<sup>2</sup> + 5.)

**Plot**[% , {v, 10, 75}, **AxesLabel** → {"km / h", "profit"},  
**TextStyle** → {FontSize → 10}];



#### 5.16.4 Owning or hiring

Suppose that you have  $c_i$  carloads per week  $i$  to ship, and you can own or hire such cars. Owning a car brings fixed costs  $F$  per week and variable costs  $V$  per week, while hiring costs are  $H$  per week. Your planning horizon is  $P$  weeks, and your objective is to minimise costs.

Let  $n$  be the number of cars that you own,  $n_i$  the number of own cars used in week  $i$ , and let  $h_i$  be the cars hired in week  $i$ . Then the minimisation problem is:

Min over  $n$ :  $P F n + V (n_1 + \dots + n_P) + H (h_1 + \dots + h_P)$  s.t.  $n_i + h_i = c_i$  while  $0 \leq n_i \leq n$  and  $0 \leq h_i$ .

If  $V \geq H$ , then you would only hire. The real problem, more common, is when  $V < H$ .

This is not a linear programming problem, since  $n$  over which we minimise is also in the constraints. However, there is a straightforward algorithm, that is to calculate all possibilities.. For a specific value of  $n$ , calculate the  $n_i = \min[n, c_i]$  and  $h_i = c_i - n_i$ . Then it is simple to calculate all costs. Doing this for all values of  $n$  from 1 till  $\max[c_i]$  gives us an  $n$  that has minimal total costs.

There is an interesting condition for optimality, though. At the optimum we should have  $\frac{\partial Z}{\partial n} = 0$ . Of course these are discrete problems, and we cannot get to zero exactly. But the approximation can be interesting. Suppose that the optimal solution for the minimand  $Z$  has  $T$  periods in which  $n_i = n$  or  $h_i > 0$ . Then one owned car more or less would give us conditions  $\Delta Z / \Delta n = 0$ .

1.  $\Delta n = -1$  causes costs to decrease by  $P F + V T$  and costs to increase by  $H T$ .
2.  $\Delta n = 1$  causes costs to increase by  $P F + V T$  and costs to decrease by  $H T$ .

Assuming that the costs cancel each other,  $T = P F / (H - V)$

`FleetMix[{F, V, H}, c_List]`

determines the size of the own fleet,  
when ownership has fixed cost F and variable usage cost V,  
while hiring has cost H, all per vehicle per period. Also,  
c gives the list of vehicle load demands for the periods over which is  
optimised. The costs for size 0 till maximal demand are in Results[FleetMix]

`FleetMix[{F, V, H}, c_List, n_Integer]` gives cost and schedules for own fleet size n

- If there is a horizon of 4 weeks with these carloads, optimum ownership is 7.

`FleetMix[{1, 0.1, 1.2}, {10, 8, 7, 12}]`

{41.6, {7}}

- With ownership 7, the cost and schedules are as follows:

`FleetMix[{1, 0.1, 1.2}, {10, 8, 7, 12}, 7]`

$$\begin{pmatrix} 11.3 & 8.9 & 7.7 & 13.7 \\ 7 & 7 & 7 & 7 \\ 3 & 1 & 0 & 5 \end{pmatrix}$$

- In above example, we find that the true  $T = 3$ , while the estimate of  $T = P F / (H - V)$  is:

$4 \cdot 1. / (1.2 - 0.1)$

3.63636

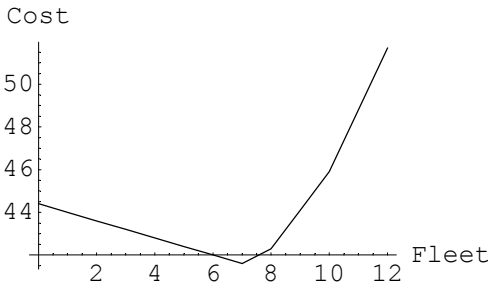
One might want to speed up the algorithm by investigating only fleet sizes around this theoretical size. However, since this kind of problem is relatively simple, one might as well compute all fleet sizes, and use the information about the costs too to check on the sensitivity.

- We can also plot the cost level depending upon the fleet sizes (here 0 till 12):

`Results[FleetMix]`

{44.4, 44., 43.6, 43.2, 42.8, 42.4, 42., 41.6, 42.3, 44.1, 45.9, 48.8, 51.7}

```
PlotLine[Transpose[{Range[0, 12], %}], AxesLabel -> {Fleet, Cost}];
```



5.16.5 Link capacity

The capacity of a road link appears to depend on the possibility of a crash. The latter depends upon the braking accelerations of vehicles (in an emergency), the reaction speeds of the drivers and the lengths of the vehicles.

```
CrashBasedCapacity[{v1, a1}, {v2, a2}, TR, L]
```

gives the capacity of a link when the requirement is that vehicle 2 (backtrailer) does not crash into vehicle 1 (frontrunner). Vehicle i drives at  $v_i$  m/s and brakes at (nonnegative)  $a_i$  m/s<sup>2</sup>. Vehicle 2 has a reaction time of TR, and the average length of the vehicles is L. It is assumed that the vehicles, while braking, do not crash before the frontrunner is fully at rest.

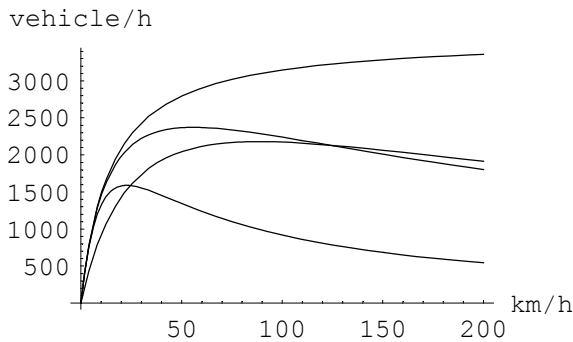
```
CrashBasedCapacity[{v, a}, TR, L]
```

gives the result for an average cruising speed of v and an immediate full stop of the frontrunner i.e. `CrashBasedCapacity[{v, Infinity}, {v, a}, TR, L]`

A well known plot gives the capacity of a link as a function of the speed of the vehicles. There are different contours for different assumptions about the braking accelerations. Normal values are 5.5 m / s<sup>2</sup> for passenger cars and 4.8 m / s<sup>2</sup> for trucks, giving an optimal speed of about 90 km / h. Note that the truck must be at least 7.5 meters behind the passenger car.

- CrashBasedCapacity is both the name of the function and the output label with that result. For readability of this Plot command we use a label cbc:

```
cbc = CrashBasedCapacity;
Plot[{3600 cbc /. cbc[{v/3.6, 5.}, 1, 4],
      3600 cbc /. cbc[{v/3.6, 6.0}, {v/3.6, 5.0}, 1, 4],
      3600 cbc /. cbc[{v/3.6, 5.0}, {v/3.6, 5.0}, 1, 4],
      3600 cbc /. cbc[{v/3.6, 5.5}, {v/3.6, 4.8}, 1, 8]
}, {v, 0, 200},
  AxesLabel -> {"km/h", "vehicle/h"},
  TextStyle -> {FontSize -> 10}]
```



### 5.16.6 Route distances

*Mathematica* comes with a standard package to plot world maps and to determine city locations. There may be occasion to plot a route and determine its distance, as for example in exam questions on shipping.

`RouteDistance[c_List]` gives the distance of a route along a list *c* of cities. Output is {total, {d1, d2, ...}} for total and consecutive distances

```
Route = {"Amsterdam", "Cardiff", "Lisbon", "Gibraltar", "Athens"};

RouteDistance[Route]

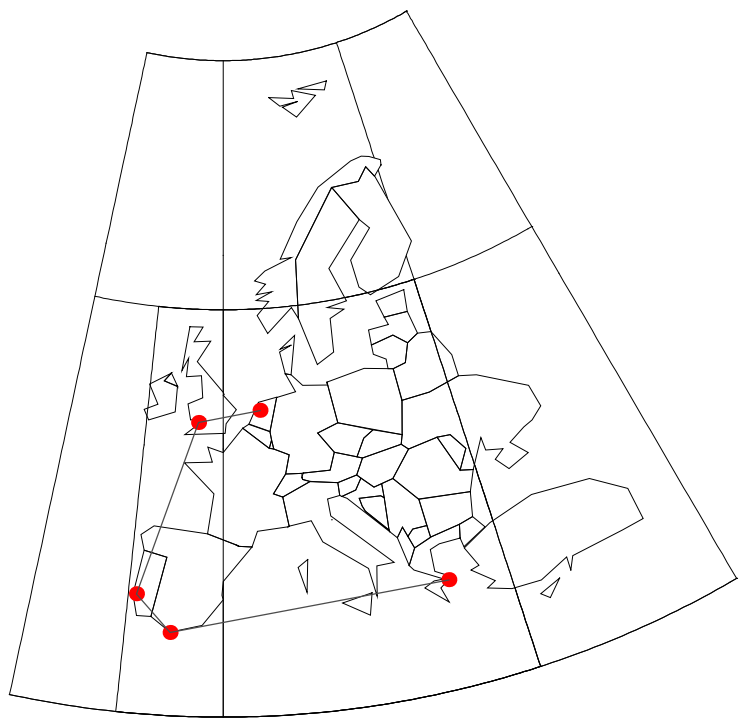
{5078.6, {564.239, 1492.34, 441.842, 2580.18}}
```

Since WorldPlot takes much CPU time, it appears useful to store some results.

<code>SetStage[countries, opts]</code>	sets Stage[] = WorldPlot[countries, options], and remembers the WorldGraphics options as Stage[Options]
<code>ShowRoute[cities]</code>	shows cities with coloured dots and connecting lines, with a Stage[] assumed

```
SetStage[WesternEurope, WorldProjection -> Albers[20, 60] 1;
```

ShowRoute [Route]



5.16.7 Time equivalent charter

Shipping rates are not only relevant for the shippers involved, but they also provide an indication of the state of the world economy. If the Pacific region is depressed and buys less goods from the Atlantic, then ships going to the Pacific will have more empty space, and rates get depressed too. The time equivalent charter transforms the revenue of a single trip for a particular duration into a trip-independent \$/day value, and thus makes different trips economically comparable.

ShippingModel[{xs}, ys, opts]	is a SolveFrom application
ToShippingSymbols	ToShippingSymbols is a list of rules to translate long symbols to short symbols
\$TimeCharter	symbol used in Equations[..], Parameter[...] and ListOfSymbols[...]

```
■ The model (format) could be used as follows (not evaluated here):  
SetOptions[ShippingModel, Equations → Equations[$TimeCharter]];  
  
res = ShippingModel[Parameter[$TimeCharter]]
```

```
SolveFrom[TableForm, res]
```

### 5.16.8 Freight

The output of the freight industry is measured in tonnekilometers, i.e. the product of each tonne times its transported distance. This output is measured relatively accurately.

Problems arise when one wishes to know more. It might for example be interesting to know the integral distance of all transports, and to see whether goods get transported further and further each year. For a causal model of freight developments one actually would need this information.

One complication in freight analysis however is that reports are based on single lifts and not on chains of lifts. A transported weight can be handled by various carriers who all report it for their leg of the journey only. For example, harbours report how much they have handled, but the subsequent freighters do the same for their effort. Therefor the same weight  $w$  shows up as  $n w$  in the statistics, with  $n$  an unknown number of handling incidents.

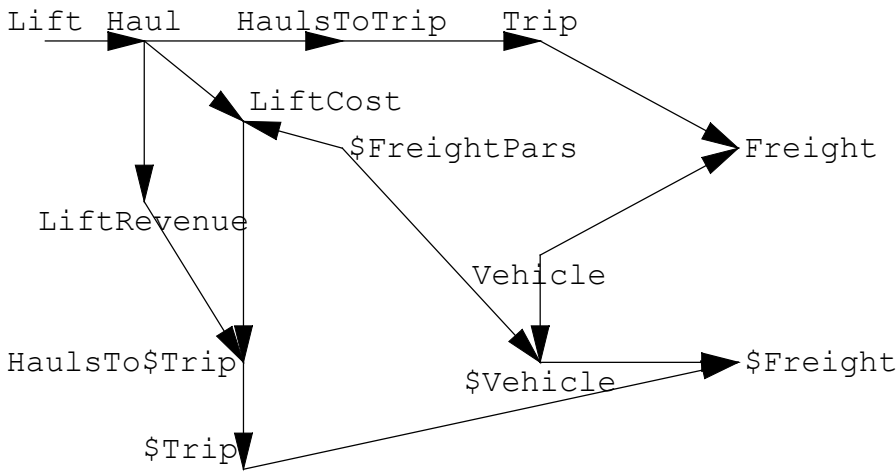
Another complication is that vehicles can be partly loaded part of the time. So if 100 tonnes are transported from A to B and subsequently 50 tonnes from B to C, then the statistics report 150 tonnes lifted. If things had been organised differently, then 50 tonnes could have been transported from A to C directly, and 50 from A to B, and the statistical report might have been about 100 tonnes lifted ....

A further complication is the different nature of cost and revenue calculations. Costs are based on operating hours and the material need for fuel, manpower, etcetera. Costs are related to the vehicles employed. Revenue cannot be taken as cost plus a markup, since the market creates an environment where customers expect to be charged per tonne in some consistent fashion. Revenue in practice is based on some rate structure. Note that there is no uniform rate for the tonnekilometers either, since handling and empty kilometers affect the calculations.

The `Freight`` and `$Freight`` packages define different objects like `Lift`, `Haul` (including handling), `Trip` (hauls with different goods), `Freight` (including the vehicle) and `$Freight` (including costs and revenues) to consistently deal with these aspects. These are `Databank`` applications.

The following diagram shows the complexity of the subject matter.

`$FreightDiagram[]`



To get acquainted with the concepts, there is a FreightModel as a SolveFrom application - though without default parameter and variable settings. Not evaluated here:

```
FreightEquations // MatrixForm

% /. ToFreightSymbols // MatrixForm

FreightModel[{LoadFactor -> 0.6}]
```



## 6. Statistics

### 6.1 Introduction

---

#### 6.1.1 Encouragement

Amazingly, some people seem to get by without statistics. For many of us it is a necessary but difficult subject. The quantitative faculties of the human mind are not well developed, witness the slow advance since the last Ice Age. As the didactic powers of our statistics teachers are poorly developed too, the chaos in the average statistics discussion can be well comprehended.

*Mathematica* might be the cane to the blind person here. We can combine formulas with numerical examples and graphics plots. We can draw white and red balls from an urn, so to speak, without getting tired doing so. We can tinker with parameters and quickly see how they change the results, so that we better understand what the parameters do.

'Being blind' is a relative concept of course. For those of us who do understand some statistics, *Mathematica* remains a useful tool too. I just mention this to be properly didactic, since this is not a sales talk.

#### 6.1.2 Choice of subjects

Below we first tackle the issues of finding a proper data format and of loading and saving datafiles. For larger data sets we might use a database structure. Secondly, with economic data such as prices and quantities we are interested making index numbers, which is a calculation that does not involve parameters and estimation of these.

The `Statistics`Common`` package sets up the framework for the decision theoretical approach to statistics. It provides for common symbols and small routines of general utility, while also loading a good number of the standard statistical packages available with *Mathematica*. The next proper subject is probability theory. It took a surprisingly large effort to find a proper expression of conditional probability, as explained in that section. But having a proper format now, we arrive at an exposition that should be clearer than a normal text, while it is directly accessible on the computer. Having this basis of probability, we proceed with statistical decision theory, and implement game theory for statistical testing.

The `Sampling`` package contains some standard cases of statistical testing. The routines just copy the formulas from the textbooks, and don't derive them. The routines thus are just a facility where one puts in the numbers in the right slots, and records the answer. This is just useful for practical situations. It is also possible to plot the operating characteristic curves, that help to understand the numbers.

The `Chi2`` package develops the  $\chi^2$  test for tabular data, using the Pearson and likelihood ratio test statistics with the default null hypothesis of independence. This is a straightforward application of the discussion in a good textbook. The `CrossTable`` package helps you in making crosstables or contingency tables from questionnaire data, and subjecting these to such  $\chi^2$  tests. It seems that *Mathematica* does not mind whether the dimensions are 2 x 2 or  $n_1 \times n_2 \times n_3 \times \dots$ . You can select dimensions, print tables, identify outliers, read from and write to files while only monitoring key results. The example notebooks help you in understanding the statistical features of the packages.

## 6.2 Data formats

---

### 6.2.1 Summary

This chapter introduces a preferred data format.

- Load the `Data`` package and an example application

```
Economics[Data]
```

```
Economics[Data`ybf]
```

### 6.2.2 Different data formats

We can list a number of different data formats:

- The **Data format** uses unassigned Symbols like `Var`, and the data then are defined and used as `Data[Var] = {...}`. This format has the advantage that the variables can be used as symbols in `SolveFrom` constructions, and as heads as in `Var[1990]`. Secondly there is the possibility of labeling, as in `Data[Var, label]` for example for different model runs. See the `Model`` package. The Data format is the advised format.
- The **DataList format** uses assigned variables like `cons = {11.5, ...}` and the variable directly refers to the data. This format has the advantage of easy operations in *Mathematica*. Many routines for lists beg for assigned names, without the `Data[ ]` interfix. In fact, many routines in the Economics Pack use this format.
- The **DataMatrix format** uses variable names (Strings), and the data are in the format of `{{"var1", 10.3, ...}, {"var2", 7.2, ...}, ...}`. This format is often used in datafiles.
- The **Generalised Data format** would use lists of Symbols, such as `Data[{consumption, cars, Holland, imported}, {volume, rate}, observed]`. In fact, the indicator could be a `Databank`` application. This is not implemented here.
- The **Contexted DataList format** uses one context (e.g. `d``) for datalists, so that `d`consumption = { ...}`, and uses the `Global`` context for symbols, so that `"consumption == 0.95 income"` remains a normal equation.

Next to the data formats, there are other data issues that complicate the discussion. What is the period that the data apply to? What is the indicator of missing data?

### 6.2.3 Data nomenclature

*Mathematica* is wonderfully easy in data formats. Still, data are as complex as the world we live in, and there are advantages from a more disciplined use. The `Data`` package implements a data system that is used by the Dutch Central Planning Bureau (CPB) and that could be preferable in general.

The basic idea about the data system is that one uses a *nomenclature*. Having a systematic nomenclature makes it easier for the user to understand what the data are about, while a programmer can make routines that exploit the data structure.

The format of the (example) nomenclature can be denoted as {concept, type, sector}. While there are various ways to deal with such a classification, the approach in the `Data`` package is to use variable names as Symbols only. For example, concept CCC, type TT and sector SS are concatenated to the variable symbol CCCTTSS. The data then can be found as `Data[CCCTTSS]`, and would generally be a list of values such as {1.9, 7.5, ...}.

- The ybf data loaded above are accessible in at least two ways, directly and by concatenation, e.g. the concatenation based on the nomenclature. In this case *ybf* stands for the gross value added at factor costs, *wn* stands for the Dutch guilder value level (in millions) and *la* stands for agriculture (Dutch "landbouw").

`Data[ybfnla]`

```
{11564, 11036, 11488, 13967, 15135, 15501, 15903,  
 16200, 17599, 16836, 17381, 20060, 20271, 20945, 20004, 17924}
```

`ToValue[ybf, wn, la]`

```
{11564, 11036, 11488, 13967, 15135, 15501, 15903,  
 16200, 17599, 16836, 17381, 20060, 20271, 20945, 20004, 17924}
```

The following table reviews how labels are used to identify the dimensions in the nomenclature.

<i>Dimension</i>	<i>Description</i>	<i>Length</i>	<i>Example Label</i>	<i>Example</i>
Concept	What the variable is about	3	con	consumption
Type	Unit of measurement	2	wn	value, level
Sector	Sector in national accounts	2	la	agriculture

The {concept, type, sector} list of dimensions given here is just an example. It can be redefined and extended by the user. For example, the *type* dimension given here actually consists of two single dimensions that only have been joined for clarity: one for the *category* (value, quantity, price etcetera) and one for the *form* (level, rate of change, quote, etcetera). Note too that the concatenation of Symbols requires the use of fixed lengths for the identifiers. If one would use variable string lengths, then the uniqueness of Symbols would no longer be guaranteed. (One could use variable length identifiers in the Generalised Data format, not implemented here.)

Note that the use of a data nomenclature still leaves freedom for a decision on using either the `Data` or the `DataList` format. The `Data`` package has chosen to implement the `Data` format. Above example of `ToValue[ybf, wn, la]` shows how you would work with that format. In the `DataList` format you could use Strings for variable names, and you would use `ToName[ybf, wn, la]` to get the variable name as a String only, while `ToSymbol[ybf, wn, la]` would generate the data list.

6.2.4 Terms

<code>Concepts [x]</code>	gives a list of Symbols on the label x. At start up, <code>Concepts[] = {}</code>
<code>Types [x]</code>	gives a list of Symbols applicable to x, as given by the associated rule in <code>NamesList[x]</code> . By default <code>Types[]</code> : <code>wn</code> = current value (millions) <code>vn</code> = value in prices last year (Laspeyres volume) <code>vr</code> = volume rate of change (Laspeyres) <code>cn</code> = volume constant prices (chained) <code>pr</code> = price rate of change (Paasche) <code>pi</code> = price index (chained)
<code>Sectors [concept]</code>	list of symbols of the economic sectors

- The application package `Data`ybf`` has introduced the following concepts:

**Concepts []**

`{ybf}`

**?ybf**

gross value added at factor costs

**Options [ybf]**

`{Types → {wn, vn, vr}, Sectors → {la, vg, tk, hb, pg, cr, me, or, de, on, bo,  
    wo, ha, zl, ot, pt, bv, at, kw, na, pl, ob, rc, bu, co, be, ex, sh, oi, tr, te, en, oo}}`

**Types [ybf]**

`{wn, vn, vr}`

**Sectors [ybf]**

`{la, vg, tk, hb, pg, cr, me, or, de, on, bo, wo, ha, zl,  
    ot, pt, bv, at, kw, na, pl, ob, rc, bu, co, be, ex, sh, oi, tr, te, en, oo}`

**?la**

landbouw

<code>NamesList [concept]</code>	gives the list of variable names available for a concept, where a variable name is a String, and where <code>ToValue[variable]</code> generates data. The list of variables is found by combining the <code>Types</code> and <code>Sectors</code> , found in <code>Options[concept]</code> .
----------------------------------	--

Note that it is possible to define and undefine explicit lists, by `Set` and `Unset` on `NamesList[concept]`.

- This creates a long list of names for data series available for the ybf concept (not evaluated here).

```
NamesList[ybf]
```

6.2.5 Accessing the data

The following have already been used above, for the Data format.

<code>Data[x, label]</code>	contains data for the variate x with the given label. The data are a list of values from BeginYear till EndYear
<code>ToValue[x__, {label}]</code>	gives Data[ToSymbol[x], label]

6.2.6 Labels

Though the data series are defined by the nomenclature, it appears useful to allow some more freedom by the use of labels. Admittedly, any label could be defined in the nomenclature, and one could work with variable names extended by those labels. However, this turns out a bit too strict for some applications. For example, when a model has been defined in terms of ybfwnla, and one runs the model three times under three different assumptions, then three data series for ybfwnla are being created. Rather than creating three variables ybfwnla1, ybfwnla2 and ybfwnla3, one can use a label, and use Data[ybfwnla, 1], Data[ybfwnla, 2] and Data[ybfwnla, 3].

<code>Labels</code>	Labels is the list of labels
<code>Store[x, b, e, label]</code>	puts the values of x[i] for the year b up to e into Data[x, label]; and maps over x if it is a list
<code>DataSymbols[label]</code>	gives the symbols for which data have been set

6.2.7 Years

The data are observed from BeginYear up to and including EndYear. The user basically provides a list of years or a routine that generates such a list. Actually, the word "year" should be understood abstractly as *period*, since the same scheme can be applied to e.g. quarters or months.

<code>Years[concept]</code>	list of integer values for which there are data for concept
<code>BeginYear[x, label]</code>	gives the begin year of data symbol x with label
<code>EndYear[x, label]</code>	gives the end year of data symbol x with label

```
Years[ybf]
```

```
{1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993}
```

**BeginYear** [ybfwnla]

1978

**EndYear** [ybfwnla]

1993

Since we have economic series that rely on price and quantity indices:

BaseYear [concept]	gives the year for which the price index of the concept is 1
BaseYear []	or BaseYear [All] is generic

**BaseYear** []

1990

6.2.8 Exploiting the data structure

The nomenclature gives structure to the data names, and this structure can be exploited to perform operations on the data. Two basic operations are aggregation and the determination of rates of change.

SumOverSectors [x, c, y, lis]	sets x <> c <> y to the sum over x <> c <> lis[[i]]
SumOverConcepts [x, c, y, lis]	sets x <> c <> y to the sum over lis[[i]] <> c <> y

For both: if c = Null, then the Sum is mapped over {wn, vn}.

SetROC[x, y, z (, beginyear:StartYear, label)]	applies SetData to x → RateOfChange[y, z], which works when x is an undefined Symbol or String
SetROC [x, pr, y, b, l]	gives x <> pr <> y as the rate of change of wn and vn
SetROC [x, vr, y, b, l]	gives x <> vr <> y as the rate of change of vn and lagged wn

These routines are used in the Data`ybf` package to determine sector aggregates and volume and price indices for these. For example, the "co" sector is the macro total.

Routines like SumOverSectors[ ] and SetROC[ ] rely on common routines ToName and ToSymbol that concatenate names and symbols.

6.2.9 Setting the data

Data are basically set by the user as `Data[symbol] = { .... }`. The symbol should be included in the `NamesList`, and also the `BeginYear` and `EndYear` should be defined. An example is provided by the `Data`ybf`` package. Data definitions can be added to existing ones, or one can reset the data system.

<code>ResetData</code>	clears <code>Data</code> and <code>DataSymbols</code> , and resets <code>BeginYear</code> and <code>EndYear</code> to $-/+$ Infinity
------------------------	---

There may be a specific occasion in which one wishes to access the data as `x[t]` for year `t`.

<code>SetData[x → y (, b:StartYear, label)]</code>	for symbol <code>x</code> and list <code>y</code> , will <code>Clear[x]</code> and assign values so that: <code>{x[b], ..., x[b+n-1]} == y</code> , where <code>n</code> is the length of the datalist <code>y</code>
<code>SetData[x_List (, b:StartYear, label)]</code>	maps over the list
<code>StartYear</code>	gives the year of the first <code>Data</code> , so that <code>var[t - StartYear + 1]</code> gives the proper position and value for year <code>t</code> . Default 1.

Note: `SetData` works for a list `y`, and the `Data[ ]` construction is not essential here.

```
StartYear
1978

SetData[ybfnco → Data[ybfnco]]

ybfnco[1990]
468560

SetData[nominalNationalIncome → Data[ybfnco]]

nominalNationalIncome[1990]
468560
```

6.2.10 Alternative format: DataList stuctures

If you want to use the `DataList` format, then you can load the `DataList`Types`` package, and find routines like above, but now defined for this format. An application is the `DataList`ybf`` package.

```
ResetAll
```



```
Economics[DataList`Types]
```

```
Economics[DataList`ybf]
```

### 6.2.11 Missing data

One aspect of one's data concept concerns the indicator of missing data.

```
Economics[MissingData]
```

```
MissingData[data_List]
```

counts the number of occurrences of the missing data indicator.  
If data have sublists, then the report is mapped over these.

Note: See the discussion for the other input formats.

The default indicator is Indeterminate. Other default options are {Format → Share, All → 100, None → 0}, and these mean that the report gives percentages, with 100 for all missing data and 0 for no missing data. The alternative setting would be {Format → Level, All → All || Number, None → None || Number}, with obvious meanings.

```
Options[MissingData]
```

```
{Symbol → Indeterminate, Format → Share, All → 100, None → 0, Context → {Cool`Common`, Global`}}
```

```
Data[var] = {{23, Indeterminate, 57, 49}, {45, 45, 45}, {33}};
```

```
MissingData[%]
```

```
{25., 0, 0}
```

The `MissingData`` package has some features for data in different contexts.

(1) First of all, if you refer to the variable by a name (String), then the `MissingData` report is not mapped over the sublists, and you can add a place selector for the sublists that you are interested in. You also can use the `Context` option to select such names in different contexts. The first symbol in the call of `MissingData` then is a head, used in `head[ToExpression[context`name]]`, which head normally would be `Data`. If you would use a `DataList` format, the head would be `Identity`.

```
MissingData[Data, "var", {1, 3}]
```

```
{var, 25., 0}
```

(2) If you want to consider various variables at the same time, then you can collect their names in `VariableNames[ ]`. The `Context` option allows one to select `VariableNames[ ]` and its elements in a certain context. (Note that the default situation has a list setting, since the default `VariableNames` are a symbol of the `Economics Pack`, while the variables will be user defined. For an alternative setting, you can use a

single context, not a list.) Finally, when the data have a structure that can be described by a vector, then you can do a selection on that structure.

```
structure = {small, medium, big};

Data[this] = {{1, 2, 3, 4}, {45, 34, Indeterminate}, {4532}};
VariableNames[] = {"this", "var"};

MissingData[Data, {small, big}, "structure"]
```

$$\left( \begin{array}{lcl} \text{Share of Global} & 1 & 1 \\ \text{this} & \{0\} & \{0\} \\ \text{var} & \{25.\} & \{0\} \end{array} \right)$$

Note that while each context can have its own structure and VariableNames[ ], that you may keep the same name selection over the various contexts, as long as those names only occur in those contexts. A call would be MissingData[Data, sel, "structure", Context → #]& /@ contexts.

## 6.3 Data files

---

### 6.3.1 Summary

This section concerns routines to write and read datafiles with extensions .data, .dalt, .math, .m, .csv and .dif.

Data will commonly be stored as a data series, either as time-series or cross-section data. In the "Data format", a series is `Data[symbol] = { ...}`, and in the "DataList format", a series is `var = { ...}`. A matrix format is `{{name1, ...}, {name2, ...}, ... }` where the name of the series is included in the list.

- Load the package

```
Economics[DataFile]
```

### 6.3.2 Using a Dump directory

The Economics Pack comes with a Data subdirectory that contains example files like Sheet2.dif. You can use these files for comparison. Here we will recreate such files in a Dump subdirectory in the root directory.

- Set the directory to, and if necessary create it:

```
CreateDirectory[ToFileName[{$HomeDirectory, "Dump"}]]
```

```
SetDirectory[ToFileName[{$HomeDirectory, "Dump"}]]
```

```
c:\Dump
```

```
FileNames[]
```

```
{}
```

We will use the following data, with names (Strings), variables (Symbols with assignment) and (unassigned) Symbols.

- This is the Data format, with unassigned Symbols:

```
Data[var1] = {610, 600};
```

```
Data[var2] = {611, 363};
```

- This is the DataList format, with assigned Symbols.

```
var3 = {100, 200};
```

```
var4 = {11, 33};
```

- This is the DataMatrix format, with names (Strings).

```
datamat = {{ "var5", 10, 11, 12}, {"var6", 20, 15, 17}};
```

### 6.3.3 File types

The routines recognise the following explicit data files:

- \*.data: readable by Get and ReadList; data are "Data[symbol] = values" per record.
- \*.dalt: readable by Get and ReadList; data are "var = values" per record.
- \*.math: readable by Get and ReadList; each data record is a list { ... }.
- \*.m: readable by Get. A package can hold the same data formats as .math, .data and .dalt.
- \*.csv: for "comma separated variables" format; each record is a new row of data.
- \*.dif: data interchange format (Lotus, Excel).

The routines use ToFileName[dir, file], which means that the directory dir can be given as a name or a list of subdirectories {dir1, ..., dirn}. Since we have set the directory, it suffices to use "". Also, the routines check on the file extension, and one can enter an option to disable this.

```
CheckFileName[fil_String, test_String, opts]
```

checks whether the last letters of fil are test. Set Check → False to avoid testing

A subroutine of the data file routines.

### 6.3.4 Data format

```
DataToFile[dir, filout_String, var_List, opts]
```

the var list is {symbol1, symbol2, ...}, and the data will be found as Data[symbol1], Data[symbol2], ... ReadList[file] or <<file will assign the variables

```
DataToFile["", "test.data", {var1, var2}]
```

```
test.data
```

```
!!test.data
```

```
SetOptions[Data, Data -> Data]
VariableNames[] = {var1, var2}
Data[var1] = {610, 600}
Data[var2] = {611, 363}
```

```

ReadList["test.data"]

{{Symbol → Indeterminate, Data → Data}, {var1, var2}, {610, 600}, {611, 363}}

Plus @@ Data /@ VariableNames[]

{1221, 963}

```

An alternative procedure is to first turn the data into a matrix, and then storing the matrix.

```

DataToMatrix[{var__Symbol}]

    creates a data matrix while assuming that each var
    concerns a variable in Data format, i.e. Data[var] gives a list of data

MatrixToData[dir, filout_String, matrix, opts]

    the file has VariableNames = {name1, name2,...} and then Data[symbol] =
    Rest[{}]. ReadList[file] or <<file will assign the variables

```

```

DataToMatrix[{var1, var2}]

( var1 610 600 )
( var2 611 363 )

MatrixToData["", "test2.data", %]

test2.data

!!test2.data

SetOptions[Data, Data -> Data]
VariableNames[] = {var1, var2}
Data[var1] = {610, 600}
Data[var2] = {611, 363}

```

### 6.3.5 DataList format

The proposed procedure is to first turn the data into a matrix, and then storing the matrix.

```
DataListToMatrix[{names__String}]
```

creates a data matrix while assuming that each name concerns a variable in DataList format, i.e. ToExpression[name] gives a list of data

```
MatrixToDalt[dir, filout_String, matrix, opts]
```

the file has VariableNames = {name1, name2,...} and then var = Rest[...]. ReadList[file] of <<file will assign the variables

```
DataListToMatrix[{"var3", "var4"}]
```

```
(var3 100 200)
(var4 11 33)
```

```
MatrixToDalt["", "test.dalt", %]
```

```
test.dalt
```

```
!!test.dalt
```

```
SetOptions[Data, Data -> List]
VariableNames[] = {"var3", "var4"}
var3 = {100, 200}
var4 = {11, 33}
```

```
ReadList["test.dalt"]
```

```
{{Symbol -> Indeterminate, Data -> List}, {var3, var4}, {100, 200}, {11, 33}}
```

```
var3 + var4
```

```
{111, 233}
```

### 6.3.6 Math format

```
MatrixToMath[dir, filout_String, matrix, opts]
```

writes matrix such that ReadList[filout] === matrix

```
MatrixToMath["", "test.math", datamat]
```

```
test.math
```

```
!!test.math
```

```
{"var5", 10, 11, 12}
{"var6", 20, 15, 17}
```

```
rdlist = ReadList["test.math"]
```

```
( var5 10 11 12 )
( var6 20 15 17 )
```

### 6.3.7 Packages

It is often a good approach to put one's data into a package. An example is the `Data`ybf`` package discussed in section 6.2, where sectors are aggregated, and where volume and price indices are determined. Note however where the real advantage of a package lies. It is not quite that one can manipulate the data. As shown above, one can use any file that `Get` or `ReadList` will properly read. The real advantage of packages is the use of contexts, such that operations can be shielded from other contexts and possible name conflicts.

The `.data`, `.dalt` and `.math` formats can all be put into a package. Since the file extension is uniform `.m`, one now really needs the `Options[Data]` and the `VariableNames[ ]` to determine what kind of package it is.

#### 6.3.7.1 Data package

```
DataToM[dir, context_String, {symbols}, opts]
```

a package with `Data[symbol] = values`

- This assumes that the `Data[symbol]` have been defined already.

```
DataToM["", "datvar`, {var1, var2}]
```

```
datvar`
```

```
!!datvar`
```

```
BeginPackage["datvar`,
{"Cool`Common`, "Cool`Context`"}]
var1::usage = "Data[var1] gives a list of values"
var2::usage = "Data[var2] gives a list of values"
Begin["`Private`"]
SetOptions[Data, Data -> Data]
VariableNames[] = {var1, var2}
Data[var1] = {610, 600}
Data[var2] = {611, 363}
End[]
EndPackage[]
```

### 6.3.7.2 DataList package

```
DataListToM[dir, context_String, {varnames}, opts]
```

a package with var = values

- This assumes that the variables been defined already.

```
DataListToM["", "datlist`", {"var3", "var4"}]
```

```
datlist`
```

```
!!datlist`
```

```
BeginPackage["datlist`",
  {"Cool`Common`", "Cool`Context`"}]
var3::usage = "List of values"
var4::usage = "List of values"
Begin["`Private`"]
SetOptions[Data, Data -> List]
VariableNames[] = {"var3", "var4"}
var3 = {100, 200}
var4 = {11, 33}
End[]
EndPackage[]
```

### 6.3.7.3 Matrix package

```
MatrixToM[dir, context_String, matrix, opts]
```

a package with Data[ ] = Data[context] = matrix

- The VariableNames[] now give a context name, and the matrix is in Data[.]

```
MatrixToM["", "test`", datamat]
```

```
test`
```

```
!!test`
```

```
BeginPackage["test`",
  {"Cool`Common`", "Cool`Context`"}]
Begin["`Private`"]
SetOptions[Data, Data -> Data]
VariableNames[] = "test`"
Data[] = Data["test`"] =
  {"var5", 10, 11, 12}, {"var6", 20, 15, 17}}
End[]
EndPackage[]
```



### 6.3.8 Comma Separated Variable format

The Comma Separated Variable (CSV) format is another popular format to exchange data. Since the comma is used in some countries for the decimal point, the default here is the semi-colon ";".

```
MatrixToCSV[dir, filout_String, datamatrix_List, sep_String: ";"]
```

the data matrix is put on file, line by line. The elements are separated by sep

```
CSVToMatrix[dir, fil_String, sep_String: ";"]
```

create a matrix, assuming that the elements in fil are separated by sep,  
and putting each record in a list

```
MatrixToCSV["", "test.csv", datamat, ";"]
```

```
test.csv
```

```
ReadList["test.csv", String]
```

```
{var5; 10; 11; 12; , var6; 20; 15; 17; }
```

```
FullForm[%]
```

```
List["var5; 10; 11; 12; ", "var6; 20; 15; 17; "]
```

```
CSVToMatrix["", "test.csv", ";"]
```

```
(var5 10 11 12 )
(var6 20 15 17 )
```

### 6.3.9 DIF format

```
ListToDif[dir, filout_String, lis_List]
```

the list is put on a dif-file of the EXCEL type. Null's are changed into  
empty strings. Lis may be a vector or a list of (unequally sized) vectors

```
DifToMatrix[dir, file.dif_String, opts]
```

a dif format file is put into a list

The Data Interchange Format is very old, and dates from the original Lotus program that invented the spreadsheet. The DIF format is somewhat peculiar, but it works, for an array of vectors. Two types are recognised automatically, the original type and the type put out by Excel.

- These are transposed data.

```
lis = {{{"a", "b"}, {12, 13}, {14.5, 15}, {, 1222.5}}
```

$$\begin{pmatrix} a & b \\ 12 & 13 \\ 14.5 & 15 \\ \text{Null} & 1222.5 \end{pmatrix}$$

```
ListToDif["", "sheet2.dif", lis]
```

```
sheet2.dif
```

- Properly reading sheet2.dif

```
result = DifToMatrix["", "sheet2.dif"]
```

$$\begin{pmatrix} a & b \\ 12 & 13 \\ 14.5 & 15 \\ \text{Null} & 1222.5 \end{pmatrix}$$

- One can see the structure by step by step reading sheet2.dif.

```
instr = OpenRead["sheet2.dif"];
```

```
in = {};
```

```
Do[AppendTo[in, Read[instr, String]], {i, 1, 38}];
```

```
Close[instr];
```

```
Print[in]
```

```
{TABLE, 0, 1, "EXCEL", VECTORS, 0, 4, "", TUPLES, 0, 2, "", DATA, 0, 0, "",  
-1, 0, BOT, 1, 0, "a", 1, 0, "b", -1, 0, BOT, 0, 12, V, 0, 13, V, -1, 0,  
BOT, 0, 14.5, V, 0, 15, V, -1, 0, BOT, 1, 0, "", 0, 1222.5, V, -1, 0, EOD}
```

```
DifToMath[" DIF_filename ", " Mathematica_filename "]
```

line by line transformation of DIF file to a *Mathematica*-format file

- Note that there is no 'dir' here. Output is also True and not the file name. Both have to do with the *Dbase`* package.

```
DifToMath["sheet2.dif", "sheet2.math"]
```

```
True
```

```
!!sheet2.math
```

```
{ "a", "b" }  
{ 12, 13 }  
{ 14.5, 15 }  
{ Null, 1222.4999999999999 }
```

```
rdlist = ReadList["sheet2.math"]
```

a	b
12	13
14.5	15
Null	1222.4999999999999

## 6.4 Databases with Dbase`

---

### 6.4.1 Summary

This chapter gives routines to manage a larger database.

```
Economics["Dbase`"]
```

### 6.4.2 Using a Dump directory

- See section 6.3 on the data files for our use of the Dump directory.

```
CreateDirectory[ToFileName[{$HomeDirectory, "Dump"}]]

dir = SetDirectory[ToFileName[{$HomeDirectory, "Dump"}]]

c:\Dump

FileNames[]

{}
```

The routines discussed in this section use the `DataPackage[ ]` routine, so that directories must be entered in the `ToFileName[ ]` format.

### 6.4.3 Introduction

Larger databases require systematic control. If we have a system, then we can use routines that exploit it.

Suppose that you have annual data on population and income for various regions of the world. Storing all information in one package would give an enormous package. An alternative is to store by year, or by property or by region. If the region would be your package context, then it is possible that you have more files per regional context, for example different files for prices and quantities. Also, when analysing the data, you will seldomly use the whole dataset. When you make a selection of variables in one dimension, then you want this selection to hold for all other dimensions. You will also like to store this selection so that it is available when you start up *Mathematica*, so that you can continue where you stopped last time.

We will make the following distinctions:

- There are "projects". For example project 1 = "Analyse the world population on income and regions."
- These projects have various "stages". For example:  
stage 1 = "Start with World and North America to get a feel of the problem."

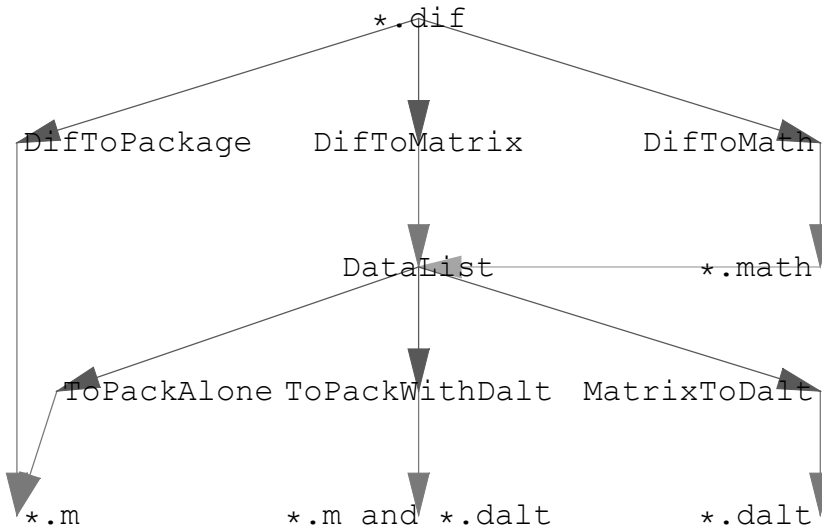
stage 2 = "Use World, NAM and Europe to make routines that deal with Lists."

stage 3 = "Try the whole data set."

Each stage will be associated with a "Dbase" selection package. The combination gives "project(i)Dbase(j) files", as for example "Pro1Dba1.m". These different data packages will be associated with different contexts, such as Pro1Dba1' and Pro1Dba2'. Thanks to *Mathematica's* context structure, one can have similar data available under different contexts.

The actual data can be arranged in various orders. Note that each multidimensional array can be mapped into a twodimensional array. In the current example each region could have its own data package, giving "wld.m", "nam.m", etc. The data that do not change over the project stages would be in a basic data directory. Typically, you would create one directory for each project\stage, and store there the dbase selection file and the results of that stage. Each project\stage accesses this directory to select the files and data relevant at that time.

Some routines in the utility package `DbaseDataFiles`` allow the use of data packages in combination with the `Dbase`` package. These data packages can be generated automatically, and some of these packages can contain instructions to read `.dalt` files. Note that the `Dbase`` package has been defined for the `DataList` format only. The following diagram gives types of data files and routines, presumably taking the `*.dif` files from some other data generating process.



#### 6.4.4 Example data

The following are data for the world population in billion persons for different regions and years.

```

regions = {"nr", "wld", "nam", "weu", "jap", "nme"};
population[1990] = {"po1990", 5290, 330, 460, 120, 390};
population[2015] = {"po2015", 7640, 400, 490, 130, 450};

db = {regions, population[1990], population[2015]};

```

**Transpose [db]**

nr	po1990	po2015
wld	5290	7640
nam	330	400
weu	460	490
jap	120	130
nme	390	450

The procedure ToPackAlone (discussed below) will put these data in a package. However, the main focus of our discussion will be on the Dbase selection file, that stores the project\stage particulars.

**6.4.5 Dbase[ ]**

Dbase [ {r___Rule} ]	makes a package with the data contexts and selected variables
Dbase [ ]	is the same as Dbase[Options[Dbase]]

The DbaseFile option default is ToFileName[{\$HomeDirectory, "Dump"}].

**Options [Dbase]**

{DbaseFile → c:\Dump\Pro1Dbal.m, DbaseContext → Pro1Dbal`, FromFileQ → False,  
SelectedVariables → {nr, datum1, datum2}, DataContexts → {d1`, d2`}, DbaseTest →  $\begin{pmatrix} \text{True} & 1 \\ \text{nr} & 2 \end{pmatrix}$ }

DbaseFile	the full name of the project & data selection package
DbaseContext	the name of the database context where the selection of variables is kept. Default is Project 1's Dbase 1: Pro1Dbal`
DataContexts	list of String codes for the data contexts. These context names are generally the same as the actual data package names. Note: the last ` may be dropped. Default is: {d1`, d2`}
SelectedVariables	for the variable Selection list. FromFileQ controls whether this selection is taken from Options or from DbaseContext`FromFile.
FromFileQ	If True, then options are taken from FromFile (which is set to a value by reading a project-stage DbaseFile package first) . If False, then these options are taken from the Options[Dbase] . FromFileQ has no role when creating a project-stage DbaseFile.
DbaseTest	for control of data file routines like ToPackAlone[]

- Let us regard a call of `Dbase[ ]`, with the default settings of the options. Note that a `Dbase[ ]` call can generate shadowing messages.

#### 6.4.5.1 The default settings

**Dbase[ ]**

*FromFile::shdw :*

*Symbol FromFile appears in multiple contexts {Pro1Dbal`, Cool`Dbase`};  
definitions in context Pro1Dbal` may shadow  
or be shadowed by other definitions.*

*Dbase::good : Dbase selection file created: c:\Dump\Pro1Dbal.m*

True

**!!Pro1Dbal`**

```
BeginPackage["Pro1Dbal`",
  {"Cool`Common`, "Cool`Context`, "Cool`Dbase`"}]
Pro1Dbal`FromFile = {
  DataContexts -> {"d1`, "d2`"}
, (* comma *)
  SelectedVariables -> {"datum1", "datum2", "nr"}
} (* end Set FromFile *)
EndPackage[]
```

Normally, the project would be in a directory, and the various stages in subdirectories. Then the project\stage context may differ from the directory. But you can enforce equality if you want to.

`Dbase[Options]`                      to show where Strings are required

`Dbase[SetOptions, dir, context, lfn:Null]`

if you want to make the options for project filename and context the same.  
If `lfn === Null`, then context and lfn are the same.

`Dbase[Query, q_String, x__]`

inserts the Dbase contexts in the Query. Try `Dbase[Query, "??", ""]`

#### 6.4.5.2 For the world population case

- A call of `Dbase[ ]` for our world population problem does not need the actual data.

`Dbase[SetOptions, dir, "Pro1Dbal2"];`

```

opts = SetOptions[Dbase, DataContexts → {"po`"},
                  SelectedVariables → {"nr", "nam", "wld"}]

{DbaseFile → c:\Dump\Pro1Dba2.m, DbaseContext → Pro1Dba2`, FromFileQ → False,
 SelectedVariables → {nr, nam, wld}, DataContexts → {po`}, DbaseTest →  $\begin{pmatrix} \text{True} & 1 \\ \text{nr} & 2 \end{pmatrix}$ }

Dbase[];

FromFile::shdw : Symbol FromFile appears in multiple contexts
{Pro1Dba2`, Cool`Dbase`, Pro1Dba1`}; definitions in context
Pro1Dba2` may shadow or be shadowed by other definitions.

Dbase::good : Dbase selection file created: c:\Dump\Pro1Dba2.m

!!Pro1Dba2`

BeginPackage["Pro1Dba2`",
             {"Cool`Common`", "Cool`Context`", "Cool`Dbase`"}]
Pro1Dba2`FromFile = {
DataContexts -> {"po`"}
, (* comma *)
SelectedVariables -> {"nam", "nr", "wld"}
} (* end Set FromFile *)
EndPackage[]

```

### 6.4.6 Storing the data

For storage, we can use a package. For small cases, we can put everything within a single package. For large data sets, we can have "case.dalt" files, and have the package call these files in order to make the selection. In both situations we cannot use the `DataFile`` routines, since we now want to add a selection process. To prevent confusion we also use `DbaseVariables` instead of `VariableNames`.

The routine `ToPackAlone` allows one to make a single package. The procedure is as follows:

- By default, the first data series must have the name "nr" and contain only Strings as elements. This can be seen as a column margin, with "nr" the "observation number code".
- By default, the name of the package is constructed as the first two letters of the first element of the "nr" series, with ".m" affixed. Thus with "po1990" we get "po.m".
- These defaults can be adapted by the option `DbaseTest`. This option has the structure  $\{\{\text{True}, 1\}, \{\text{"nr"}, 2\}\}$ . If the  $[[1, 1]]$  element is True, then the data in the second list are taken for the codes, with  $[[2, 1]]$  the name of the series, and  $[[2, 2]]$  the number of letters taken for the context and package name.



<code>ToPackAlone[dir, data, opts]</code>	for a package with <code>vari = Rest[data[[i]]]</code> . DbaseTest controls the context` name
<code>DbaseVariables</code>	the list of variables in the Dbase

- To allow for memory constrained running (see below), the output is True or not.

```
ToPackAlone["", Transpose[db]]
```

```
ToPackAlone::good : File written po.m
```

```
True
```

```
!!po`
```

```
BeginPackage["po`",
{"Cool`Common`, "Cool`Context`, "Cool`Dbase`"}]
DbaseVariables::usage = "List of names (Strings)"
nr::usage = "List of values"
wld::usage = "List of values"
nam::usage = "List of values"
weu::usage = "List of values"
jap::usage = "List of values"
nme::usage = "List of values"
Begin["`Private`"]
DbaseVariables = {"nr", "wld", "nam", "weu", "jap", "nme"}
If[ MemberQ[DB[SelectedVariables], "nr"],
nr = {"po1990", "po2015"}
]
If[ MemberQ[DB[SelectedVariables], "wld"],
wld = {5290, 7640}
]
If[ MemberQ[DB[SelectedVariables], "nam"],
nam = {330, 400}
]
If[ MemberQ[DB[SelectedVariables], "weu"],
weu = {460, 490}
]
If[ MemberQ[DB[SelectedVariables], "jap"],
jap = {120, 130}
]
If[ MemberQ[DB[SelectedVariables], "nme"],
nme = {390, 450}
]
End[]
EndPackage[]
```

### 6.4.7 Reading data

If we read the data, then the data selection becomes active. Each data entry in the package has been subjected to a selector. The `FromFileQ` option in `Options[Dbase]` has the following effect:

- If `FromFileQ` → False then the `SelectedVariables` option of `Options[Dbase]` is taken.

- If `FromFileQ` → `True` then the `SelectedVariables` option of the `DbaseContext`FromFile` is taken.

<code>DB[x]</code>	selects options (like <code>DataContexts</code> and <code>SelectedVariables</code> ) either from <code>DbaseContext</code> or from <code>Options[Dbase]</code> . This selection is controlled by <code>FromFileQ</code> in <code>Options[Dbase]</code>
--------------------	--

- The selection has been set above by the `SetOptions[Dbase, ...]`.

```
DB[SelectedVariables]
```

```
{nr, nam, wld}
```

- A package may be read in standard manner.

```
<<po`
```

- Since `nam` and `wld` are selected, we find the share of North America in the World population.

```
nam / wld // N
```

```
{0.0623819, 0.052356}
```

- This would give the share of Western Europe: but these data have not been selected.

```
weu / wld // N
```

```
{0.000189036 weu, 0.00013089 weu}
```

#### 6.4.8 Reading an earlier data selection

If we close *Mathematica* and start another session, or do a `ResetAll`, then our data selections have been cleared from memory, and they are only available in the various `Pro(i)Dba(j).m` files. Restarting the session, we could set the option that the selection should be read from file.

```
ReadDbase[dir_String, opts]
```

maps `ReadPackage` to `DataContexts`, using a `ShadowHull`

```
ReadPackage[dir_String, context_String, opts]
```

reads the `DataPackage[dir, context]` using the selection created by `Dbase`

Note: Default option is `MessagesQ`→`False`, for both in `Options[ReadDbase]`.

- Let us first reset all, and then set up the same start situation. We set the default selection to {} and set FromFileQ → True.

**ResetAll**

**Economics**["Dbase`"]

**dir** = SetDirectory[ToFileName[{\$HomeDirectory, "Dump"}]];

**Dbase**[SetOptions, dir, "Pro1Dba2"];

**opts** = SetOptions[Dbase, DataContexts → {"po`"},  
FromFileQ → True, SelectedVariables → {}]

{DbaseFile → c:\Dump\Pro1Dba2.m, DbaseContext → Pro1Dba2`, FromFileQ → True,  
SelectedVariables → {}, DataContexts → {po`}, DbaseTest →  $\begin{pmatrix} \text{True} & 1 \\ \text{nr} & 2 \end{pmatrix}$ }

**<<Pro1Dba2`**

*FromFile::shdw :*

*Symbol FromFile appears in multiple contexts {Pro1Dba2`, Cool`Dbase`};  
definitions in context Pro1Dba2` may shadow  
or be shadowed by other definitions.*

- Now call ReadDbase. This will recover the data selection that we had before.

**ReadDbase**[""]

{Null}

- These data had been removed by ResetAll, but are available again. Note that the selection of variables has been taken from the Pro1`Dba2` context.

**??wld**

List of values

wld = {5290, 7640}

### 6.4.9 Dictionary

Having a larger database also generates the need for some dictionary. This especially happens when the same kind of variable occurs in more files: rather than using the same long explanation every time again, one uses an acronym, and stores the explanation in a different place.

<code>Dictionary[Clear]</code>	clears the entries
<code>Dictionary[File]</code>	gives the name of the current dictionary file
<code>Dictionary[Get]</code>	reads from file
<code>Dictionary[var_String]</code>	gives the dictionary entry for that variable name
<code>Dictionary[Variables]</code>	gives the variable names with a dictionary entry
<code>Dictionary[Write]</code>	writes to <code>Dictionary[File]</code>
<code>DictionaryQ</code>	Initially False; True denotes the existence of a Dictionary

Note: `Clear[Dictionary]` may cause you to read the package again.

If not read from file, the variables and entries of course must be supplied by the user, as `Dictionary[Variables] = {...}` and `Dictionary["name_i"] = "entry_i"`

- Load the package.

**Economics[Dictionary]**

- Create a dictionary for fraud research data, where we recognise two dimensions, namely the country where the fraud takes place, and the type of fraud.

```
Dictionary[Variables] = {"country", "fraud"};
Dictionary["country"] = {"uk1 England", "uk2 Wales"};
Dictionary["fraud"] = {"101 theft", "102 writing error", "103
oversight"};
DictionaryQ = True;

Dictionary[File] = ToFileName[{$HomeDirectory, "Dump"}, "fraudict.ry"];
Dictionary[Write];

!!fraudict.ry

Dictionary[Variables] = {"country", "fraud"}
Dictionary["country"] = {"uk1 England", "uk2 Wales"}
Dictionary["fraud"] = {"101 theft", "102 writing error", "103
oversight"}
```

- Clear the dictionary.

**Dictionary[Clear]**

- Load the dictionary from file, still known as Dictionary[File].

```
Dictionary[Get]
```

```
True
```

- Any time when we have a question about an entry, we can ask additional information.

```
Dictionary[Variables]
```

```
{country, fraud}
```

```
Dictionary["fraud"] // MatrixForm
```

```

$$\begin{pmatrix} 101 \text{ theft} \\ 102 \text{ writing error} \\ 103 \text{ oversight} \end{pmatrix}$$

```

#### 6.4.10 Packages that call .dalt's, and more files per context

A package can use a routines to make a selection, and then read this selection from another .dalt file. It is also possible that one has more files per context, for example separate files on prices and volumes. The `DbaseDataFile`` package written for such larger databases.

```
ToPackWithDalt[dir_String, ps_String, varnames_List, flndat:Automatic]
```

makes a package which reads data with `ReadData[ ... flndat ...]`.  
The package is ps <> .m and automatic flndat is ps <> .dat

- Load the package.

```
Economics[DbaseDataFile]
```

```
ToPackWithDalt["", "pwdtest", {"var1", "var2"}]
```

```
ToPackWithDalt::good : File written pwdtest.m
```

```
True
```

```
!!pwdtest`
```

```
BeginPackage["pwdtest`",
{"Cool`Common`", "Cool`Context`", "Cool`Dbase`"}]
DbaseVariables::usage = "List of names (Strings)"
var1::usage = "List of values"
var2::usage = "List of values"
Begin["`Private`"]
DbaseVariables = {"var1", "var2"}
w = OpenRead["pwdtest.dalt"]
Map[ReadData[w], DB[SelectedVariables]]
Close[w]
End[]
EndPackage[]
```

When one has more files per context, as said for example separate files on prices and volumes, then the routines `MathToUnionDataList`, `DifToPackage` and `DifBigToM` are in order. These routines can be run with `MemoryConstrainedQ`  $\rightarrow$  `True` (which explains why some datafile routines return `True` when successfully completed). There is the note of warning that these routines have *not* been tested under *Mathematica* 3.0, since the occasion simply did not occur.

- `DifToPackage`: numbered DIF-files are first transformed line by line to *Mathematica* files, and then combined to a package. The "nr" series is used to check that all datafiles have the same kind of data.
- `DifBigToM`: the numbered DIF-files are first transformed line by line to *Mathematica* files, and then combined to both a datafile and a package (with `ToPackWithDalt`)
- `MathToUnionDataList`: subroutine for the above. *Mathematica* files are combined into a dataset in memory. If `CleanUp` then only-Nulls are reduced and dictionary variables are abbreviated. If `UpdateDictionaryQ` then the dictionary is Updated.

# 6.5 Chain indices for volumes and prices

## 6.5.1 Summary

This package generates an aggregate Laspeyres volume and Paasche price index, while indices for subsequent periods can be chained.

`Economics["Chainindex`"]`

## 6.5.2 Introduction

The use of aggregate indices is a second best. If each market has quantity and price results  $\{q_i, p_i\}$  so that the aggregate value is  $p_1 q_1 + \dots + p_n q_n$ , then the best indicator of the aggregate situation is the social welfare function value  $SWF[q_1, \dots, q_n]$ , and there is no need for any other additional index. The problem however is that the SWF is unknown and has to be estimated. Given the uncertainties here, a preference has developed to use parameter free measures that are sufficiently adequate to indicate aggregate volumes and prices.

This package implements the time honoured method of the Dutch Central Planning Bureau of using chain Laspeyres volume indices and chain Paasche price indices. Advantages of this are:

- Volume and price changes add up to value changes.
- The weights for volumes and prices are adjusted over time.
- The choice of a different base period does not lead to different values for the aggregate data.

A consequence however is that the aggregate chained volume index differs from the sum of the disaggregate chained volume indices.

The discussion below uses the same symbols for the dimensions of variables as section 6.2.4 above.

## 6.5.3 The main routines

`ChainIndex[q, p, pos:1]` gives a chain index for quantities  $q$  and associated price indices  $p$ . Elements  $q[[i]]$  and  $p[[i]]$  must be vectors. Position  $pos$  in the price vectors is normalized to 1

The different steps are, with  $q$  now  $q_n$  to express that its dimension is 'level':

1.  $wn = pn * qn$  gives values, and summing these gives aggregate  $wntot$ .
2.  $pu = \text{UnitIndex}[pn]$  are the unit price indices (per period). A unit index is  $pu = (1 + pp / 100)$  where  $pp$  is the percentage price change from the lagged period to this period.
3.  $vn = wn / pu$  deflates to volumes (value in terms of lagged period prices).

- 4. summing volumes gives aggregate volume  $vntot$ , and (aggregate)  $wntot / vntot$  gives the unit price indices for the aggregate (per period).
- 5. chaining the aggregate unit price indices gives aggregate  $pitot$ , and chained volume  $cntot = wntot / pitot$  (in prices of the base year). Output is that pair  $\{cntot, pitot\}$ .

<i>Symbols used in this discussion</i>	pd = price change in level
vd = volume change in level	pp = price change in percentage of vn
vp = volume change in percentage	wn = value in level, current prices
vn = volume level (value in former prices)	wn - 1 = lagged value in level

An additional advantage of this aggregation method is that there is an easy table for the construction or presentation of the indices. If we use above steps, we can create the following columns, the sum of which (or weighted sum), i.e. the last row, gives the aggregated total:

ChainIndexTable[q, p, pos: I]

the same as ChainIndex but prints the results in tabular form. See Results[ChainIndexTable] for more data results

- Let the number of sold radio's rise from 100 to 110, while the price remains fixed at \$10 per radio. Let the number of sold television sets fall from 45 to 43, while the price rises from \$130 to \$140. The aggregate indices for "electronic equipment" are a -2.3 % drop in volume and a 6.4 % rise in price. The drop in sales in television sets weighs heavily since the value of sales is relatively large.

ChainIndexTable[{{100, 110}}, {{45, 43.}}, {{10, 10}}, {{130, 140.}}]

	wn-1	vd	vp	vn	pd	pp	wn
1	1000	100	10	1100	0	0	1100
2	5850	-260.	-4.44444	5590.	430.	7.69231	6020.
3	6850	-160.	-2.33577	6690.	430.	6.4275	7120.

6.5.4 Subroutines

Chain[x, pos: I]

chains a unit change vector while normalizing to 1 in pos

Note: Common routines are Lag, UnitIndex and RateOfChange. Note: Also defined are AllNonChained, FischerP, FischerQ, LaspeyresP, LaspeyresQ, PaascheP, PaascheQ, ValueIndex, all with obvious use. Note: The indicator for missing data is Indeterminate.



# 6.6 Statistics`Common`

## 6.6.1 Summary

This package provides some basic routines and undefined common symbols.

```
Economics["Statistics`Common`"]

Needs["Graphics`Graphics`"]
```

## 6.6.2 Introduction

The `Statistics`Common`` package provides some basic routines and undefined common symbols. The routines will be discussed below, and their use will become clear. The use of the undefined common symbols is less obvious. Their use can best be understood in relation to the other statistical packages. Whenever such a package required a symbol that could be suspected to be useful for more cases, that symbol has been put in this common package. Two of such packages are `Probability`` and `Decision``.

- Probability is basic to the development of statistics. The head `Pr` is used to denote probability, so `Pr[A]` would be the probability of event `A`. The head `Pr` is put in the common package, and the `Probability`` package develops the basics of probability theory.
- We take the decision theoretical approach to statistics. The statistician and nature play a game, where nature selects the state of the system and where the statistician selects an action. This framework is further developed in the `Decision`` package, while the symbols like `NumberOfStates` and `NumberOfActions` are put into the common package.

Note: While *Mathematica* 3.0.0 did not provide for a Multinomial distribution and the `Pack` did, now 4.0.0 does and thus the `Pack` doesn't any more.

## 6.6.3 Symbols only

Accept	Est	Pr
Act	ID	Reject
Categories	NumberOfActions	State
Dec	NumberOfObservations	Test
DegreesOfFreedom	NumberOfStates	TestStatistic
Error	Obs	TValues

`State[i]` is a state, `Act[i]` is an action, `Dec[i]` is a decision. `Accept` and `Reject` are possible decisions in hypothesis testing. `Obs[i]` is an observation, `Est[i]` is an estimate. `ID` is a label for indicating the

assumption of independently distributed variables. Pr stands for probability, with Pr[A] the probability of event A. The only real definition here is:

NumberOfDecisions

NumberOfActions

<sup>NumberOfStates</sup>

6.6.4 Data manipulation

RelativeFreq[ <i>data_List</i> ]	gives the relative frequency
FrequencyCategories[ <i>freq_List</i> ]	selects the categories
FrequencyValues[ <i>freq_List</i> ]	selects the frequency values

The later two take the output of Statistics`DataManipulation`Frequency[] as input.

RelativeFreq[{a, b, c, a, b}]

$$\left(\begin{array}{c} \frac{2}{5} \ a \\ \frac{2}{5} \ b \\ \frac{1}{5} \ c \end{array}\right)$$

MissingToZero[ <i>var, categories_List, opts</i> ]
includes {0, category} for missing categories.

If var is a String, then it is taken as the variable in the context specified by the Context → option (default Global` ). Since ShadowHull is used, option MessagesQ → False (default) suppresses shadowing messages.

MissingToZero[%, {c, d}]

$$\left(\begin{array}{c} \frac{2}{5} \ a \\ \frac{2}{5} \ b \\ \frac{1}{5} \ c \\ 0 \ d \end{array}\right)$$

If the original data are numeric, then the following applies.

FrequencyDispersion[ <i>freq_List</i> ] or FrequencyDispersion[ListPlot, <i>x_List</i> ]
gives a dispersion report with the statistics weighted by the frequencies

The first format takes output from Frequency[], the other format is like the one of ListPlot[ ]. The result can be input of NormalProxy to give a proxy by the Normal distribution with mean and standard deviation.

### 6.6.5 Presenting data in a bar chart

```
BarChartServer[data_List, stepfactor:0.1]

gives
{Factor → stepfactor, Range → {Min, Max, step}, BinCounts → freq, Point → midpoints}
```

This routine gives rough and ready settings for a bar chart. It can be useful when a proper bin selection is yet unknown, or when defining a known bin selection is not worth the effort.

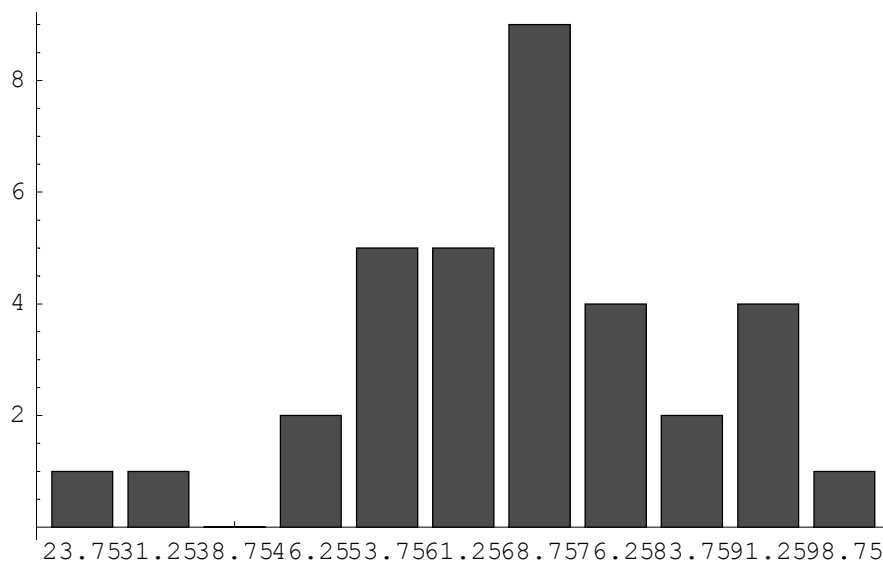
- These are results of a mathematics exam (on a scale of 0 till 100).

```
class = {55, 76, 95, 44, 55, 61, 50, 94, 86, 71, 46, 60, 53, 32, 70,
89, 71, 68, 68, 76, 86, 72, 59, 92, 88, 77, 69, 72, 51, 60, 66, 20, 64,
73};
```

```
bcs = BarChartServer[class]
```

```
{Factor → 0.1, Range → {19.8, 95.95, 7.5}, BinCounts → {1, 1, 0, 2, 5, 5, 9, 4, 2, 4, 1},
Point → {23.75, 31.25, 38.75, 46.25, 53.75, 61.25, 68.75, 76.25, 83.75, 91.25, 98.75}}
```

```
BarChart[Transpose[{BinCounts, Point} /. bcs ]];
```



### 6.6.6 Statistical measures

*Mathematica* provides a wealth of statistical measures. The supplement provided here can be categorised into (1) A slightly different collection of output, (2) Some names that remain from earlier times when *Mathematica* did not provide this wealth (such as Covar below, which remains still handy as a short name), (3) The inclusion of weights (such as CovarPr).

<code>BasicStatistics[ x_List]</code>	gives basic statistics like mean etc.
---	---------------------------------------

- Using math exam data (above).

```
disrep = BasicStatistics[class] // N
```

```
{Mean → 66.7353, Median → 68.5, Min → 20., Max → 95., NumberOfObservations → 34.,  
Mode → {55., 60., 68., 71., 72., 76., 86.}, Variance → 297.534, StandardDeviation → 17.2492,  
SampleRange → 75., MeanDeviation → 13.237, MedianDeviation → 9., QuartileDeviation → 10.5}
```

Frequency data can give a mean and variance without referring to the original data.

<code>WeightedMean[freq_List]</code>	gives the weighted mean
<code>WeightedMean[x_List, fr_List]</code>	weights x by using the frequencies fr (note the other order of x and fr)

Note: A routine MeanPr[x, p] would just be  $x \cdot p$  and thus has not been defined.

- This uses Frequencies, but sometimes only such output is available, and not the original data.

```
WeightedMean[Frequencies[{a, b, c, a, b, e}]]
```

$$\frac{a}{3} + \frac{b}{3} + \frac{c}{6} + \frac{e}{6}$$

Options[WeightedMean] determine the weighing function, with default WeightUnit → (# / Add[#])&. Another setting is DivSmallestNonZero, which divides the weights by the smallest nonzero element (that might be negative). In manipulating weights, one can use symbols Weight for a single weight and WeightTotal for the total of all weights.

<code>VariancePr[x_List, p_List]</code>	gives the variance of a list of values x using the list of associated probabilities p
---	--

```
VariancePr[{x1, x2}, {p1, p2}]
```

$$p1(-p1 x1 + x1 - p2 x2)^2 + p2(-p1 x1 - p2 x2 + x2)^2$$

The CovarianceMatrix routine supplied with *Mathematica* is quite powerful. Covar remains from earlier times, but still is a useful label - see Prospects. CovarPr gives the weighted results.

<code>Covar[x_List, y_List (-1)]</code>	gives the covariance of x and y. Including – 1 divides by n–1 instead of n
<code>CovarMat[{x1_List, ..., xn_List} (-1)]</code>	gives the covariance matrix of the xi data lists. Including – 1 divides by n–1 instead of n
<code>CovarPr[x_List, y_List, p_List]</code>	gives the covariance of x and y, assuming simultaneous probabilities p
<code>CovarMatPr[{x1_List, ..., xn_List}, p_List]</code>	gives the covariance matrix of the xi data lists, assuming simultaneous probabilities p

**CovarPr[{10, 11, 12}, {4, 15, 6}, {.1, .3, .6}]**  
-1.05

<code>CorrelationPr[x_List, y_List, p_List]</code>	gives the correlation coefficient of x and y, assuming simultaneous probabilities p
<code>ToCorrelation[m_?MatrixQ]</code>	turns a covariance matrix into a correlation matrix

**CorrelationPr[{10, 11, 12}, {4, 15, 6}, {.1, .3, .6}]**  
-0.364405

<code>GeometricSpread[lis_List (-1)]</code>	determines the standard deviation based on the geometric mean. Inclusion of –1 divides by n–1. Note: for growth rates or interest rates, enter (lis+1)
<code>GeometricCovar[x_List, y_List (-1)]</code>	gives the covariance of x and y around their geometric means. Including –1 divides by n–1 instead of n

**GeometricSpread[{1.05, 1.04, 0.99}]**  
0.0262489

<code>Outliers[k, dx_List, mu:0]</code>	for k times the standard deviation, $Abs[dx-\mu] > k \text{ sigma}$
---	---

The outliers routine finds those values that are at more than k sigma distance of the mean. The standard situation has  $dx = x - \text{Mean}[x]$ , so that  $\mu = 0$ . Output is {Range → k, N → number of outliers, Quotient → rate of occurrence of outliers, StandardDeviation → sigma, Normal → normalized data  $(dx-\mu)/\text{sigma}$ , Position → positions of the outliers}.

```
varx = {1, 3, 4, 5, 3, 2, 10, 4, 6};  
  
Outliers[2, varx - Mean[varx]]  
  
{Range → 2, StandardDeviation → 2.63523,  
  Normal → {-1.22275, -0.463801, -0.0843274, 0.295146, -0.463801, -0.843274,  
    2.19251, -0.0843274, 0.674619}, Position → (7), N → 1, Quotient → 0.111111}
```

PrBelowOutMean [ <i>x_Symbol</i> , <i>m_Symbol</i> ]	is measure of skewness
PrBelowOutMean [ <i>list</i> ]	the measure for a list of data.
PrBelowOutMean [ <i>freq_List</i> , <i>values_List</i> ]	the measure for frequency data

Results[PrBelowOutMean] contain the mean, the  $\Pr[x < m]$  and the  $\Pr[x = m]$ .

PrBelowOutMean[ ] tells how far apart the two conditional means  $E[x \mid x > m]$  and  $E[x \mid x < m]$  are from the mean  $m$  of a distribution. The measure is simply the area in the density function on the left of the mean, adjusted for the area taken in by the mean itself. For continuous densities  $\Pr[x = m] = 0$ , so there is no adjustment. For discrete densities the adjustment is  $(1 - \Pr[x = m])$ . This skewness measure has basically been proposed by Tajuddin, Statistica Neerlandica 1996, pp 362-366.

```
PrBelowOutMean[x, m]
```

$$\frac{\Pr(x < m)}{1 - \Pr(x == m)}$$

Even with discrete data, the probability  $\Pr[x = m]$  is often null, since  $m$  is a fraction. With discrete data, one might prefer to take the proper mean as the integer closest to that fraction. Use the option RoundAt →  $n$  for the number of digits  $n$ . Default RoundAt → False.

```
PrBelowOutMean[{1, 3, 4, 5, 3, 2, 1, 4, 6}]  
  
 $\frac{5}{9}$   
  
PrBelowOutMean[{1, 3, 4, 5, 3, 2, 1, 4, 6}, RoundAt → 0]  
  
 $\frac{3}{7}$ 
```

### 6.6.7 The normal distribution

```
NormalProxy[x_Symbol, d_List, dcf:Automatic]
```

constructs a normal distribution that approximates the original data,  
with *d* dispersion statistics

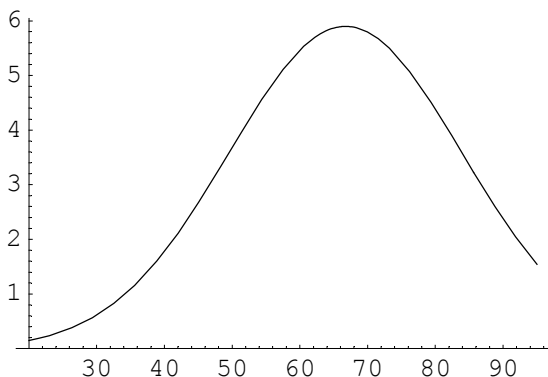
The display correction factor *dcf* allows a better comparison of the plot of the PDF with the ListPlot of the original data. If *dcf* = Automatic, then the *dcf* is computed as SampleRange / NumberOfStates (e.g. bins).

- Using the basic statistics of the math exam, computed above.

```
np = NormalProxy[x, disrep ~Join~ {NumberOfStates → 10}]
```

```
{PDF → 0.786359 e-0.00168048 (x-66.7353)2, Factor → 7.5,  
Plot → Hold[Plot][0.786359 e-0.00168048 (x-66.7353)2 Hold[7.5],  
{x, 20., 95.}, PlotRange → All, AxesOrigin → {20., 0}]}
```

```
ReleaseHold[Plot /. np];
```



```
CNormalInverse[{mu, sigma}, level]
```

gives the value on the x-axis for the cumulative normal = level

```
CNormalInverse[{0, 1}, 0.95]
```

```
1.64485
```

<code>DrawNormal[{mu, sigma}, n_Integer]</code>	makes a series of n draws of the normal distribution with mu and sigma
<code>DrawNormal[{mu, sigma}, n_Integer, True]</code>	draws, and adjusts so that the series mean and standard deviation are exactly the mu and sigma values
<code>DrawNormal[{mu, sigma}, n_Integer, -1]</code>	as True, but now using division by n-1 for $\sigma$

```
DrawNormal[{0, 1}, 4, True]
{-0.970629, 0.179541, -0.769087, 1.56018}

StandardDeviationMLE[%]
1.
```

6.6.8 The lognormal function

The lognormal distribution is characterised by two parameters, and these are normally written as  $\mu$  and  $\sigma$ . This is a bit confusing, since commonly the  $\mu$  and  $\sigma$  are the mean and standard deviation, and here they are not. Best is to define the lognormal function with parameters  $\theta$  and  $\tau$ , and try to relate these to the mean and standard deviation. One will find that  $\theta$  even differs from  $\text{Log}[\text{mean}]$ .

<code>ToLogNormal[mean, stdev, median:Automatic]</code>	takes sample observations mean and stdev, and generates the parameters of the associated <code>LogNormal[theta, tau]</code> distribution.
<code>FromLogNormal[theta, tau]</code>	takes the theta and tau parameters of the <code>LogNormal</code> distribution, and translates these in the proper mean and standarddeviation

```
ToLogNormal[mu, sigma]
{Theta -> 1/2 (4 log(mu) - log(mu^2 + sigma^2)), Tau -> Sqrt[log(mu^2 + sigma^2) - 2 log(mu)]}

FromLogNormal[theta, tau]
{Mean -> e^(tau^2/2 + theta), StandardDeviation -> e^(tau^2/2 + theta) Sqrt[-1 + e^tau^2],
Median -> e^theta, Mean / Median -> e^(tau^2/2)}
```



If the sample median is supplied, then Log[median] could be the best estimator of theta - though of course the sample data may not fit the LogNormal distribution. For that latter reason ToLogNormal sticks to the use of the mean.

### 6.6.9 The Beta function

The Beta function is bell shaped like the normal distribution, but has a bounded domain, with a minimum and a maximum value. An example application of the Beta distribution is in the Critical Path Method in operations research. Individual projects are scored on their minimal, modal and maximal duration, the means and variances are summed, and, with an appeal to the law of large numbers, one then uses the normal distribution to discuss the stochastic properties of the total critical path.

While the standard Beta distribution is defined over the [0, 1] domain, the following routines extend this.

ToBeta[*min*, *mode*, *max*]
gives estimates of the mean  
and variance of the Beta distribution

ToBeta[PDF | CDF, *mu*, *var*, {*x\_Symbol*, *min*, *max*}]
  
  
generates the PDF or CDF for *x* in [*min*, *max*]

ToBeta[PDF | CDF, *mode*, {*x\_Symbol*, *min*, *max*}]
  
  
combines, using estimates for *mu* and *var* from ToBeta

Note: These functions actually are defined in Economics[Statistics`Beta].

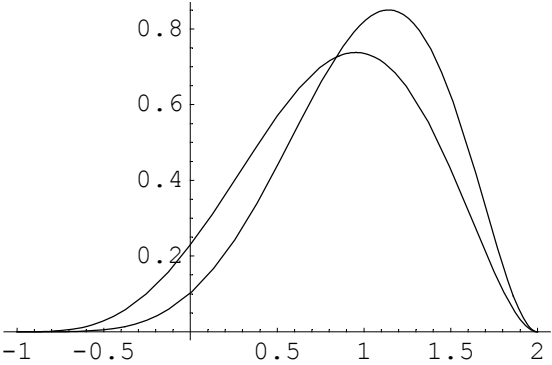
```

dist = {ToBeta[PDF, 1, 0.2, {x, -1, 2}], ToBeta[PDF, 1, {x, -1, 2}]}

{56. (2/3 - x/3)^2. (x/3 + 1/3)^5. (2/3 - x/3)^157/81 (x/3 + 1/3)^293/81 Γ(68/9)}
{3 Γ(238/81) Γ(374/81)}

Plot[Evaluate[dist], {x, -1, 2}];

```



The following subroutine is used:

<code>ToBetaDistribution[{mu, var}, {min, max}]</code>	
<code>ToBetaDistribution[min, mode, max]</code>	determine the {alpha, beta} parameters from those input data

Note: The generated distribution B still is defined only on [0, 1]. The proper mean is found by (max-min) Mean[B] + min, and the variance is recovered from (max-min)^2 Variance[B].

Note: ToBeta[min,mode,max,All] gives {{mu,var},{alpha,beta}} where the alpha and beta values derive from rules of thumb that are less reliable than found by ToBetaPDF and ToBetaDistribution.

<code>SubsequentBetaCDF[lis_List, t, printq:True]</code>	
	gives the distribution of the cumulative duration t of a list of subsequent Beta events or activities. The activities are scored on the minimal, mode, and maximal duration, so that lis = {..., {mini, modei, maxi}, ...}.

Note: Output is the Normal CDF with this total mean and standarddeviation, which are printed if printq == True. Variable t can be symbolic or have a numeric value.

6.6.10 Probability

<code>PM[x]</code>	makes a probability measure of x. 'Rest' in the list is set to 1 - sum[x] Not is applied when ProbabilityMeasureQ fails.
<code>ProbabilityMeasureQ[x]</code>	tests whether x is a probability measure, i.e. x[i] >= 0 and sum[x] = 1

```
PM[{.1, .3, .4, .07, Rest}]
{0.1, 0.3, 0.4, 0.07, 0.13}

PM[{a, b, Rest, d}]
{a, b, -a - b - d + 1, d}

PM[{.1, .3, Rest, .7, .4, .07}]
Not::argx : Not called with 6 arguments; 1 argument is expected.
Not[0.1, 0.3, -0.57, 0.7, 0.4, 0.07]

PM[{a, b, c}]
Not::argx : Not called with 3 arguments; 1 argument is expected.
Not[a, b, c]
```

**ProbabilityMeasureQ**[{a, b, c, 1 - a - b - c}]

NonNegative[a]  $\wedge$  NonNegative[b]  $\wedge$  NonNegative[c]  $\wedge$  NonNegative[-a - b - c + 1]

**PrTable**[t, p, ...] or **PrTable**[{t, p, ...}]

= **Outer**[Times, t, p, ...] gives the probability of occurrence of cells, when the states have distribution t and the actions have distribution p, etc.

**IDPrMatrixQ**[x] test whether the matrix represents independent distributions. It does not test on the latter being probability measures

**PrTable**[{t, 1 - t}, {p, 1 - p}]

$$\begin{pmatrix} p t & (1-p)t \\ p(1-t) & (1-p)(1-t) \end{pmatrix}$$

**IDPrMatrixQ**[%]

True

Next to **Pr**, there are also undefined symbols **ConditonalPr**, **MarginalPr** and **ProbabilityMatrix**. For the latter, it depends upon the routine call whether all elements in the matrix add up to one (as in **PrTable**), or whether this holds only for rows or columns.

# 6.7 Probability

## 6.7.1 Summary

This package implements basic probability theory. The Prospect object allows easy handling of discrete probability situations, and Draw helps you to find a proper playing strategy. For symbolic manipulation, it was a lot of work to find a good representation for conditional probability. That has become: ConditionalPr[x][y].

```
Economics[Probability]
```

## 6.7.2 Prospects

A prospect is an object that collects both the events that can occur and the probabilities that they occur. A binary prospect recognises only two events; if the probability of the first is  $p$ , then the probability of the other is  $(1 - p)$ . The multidimensional prospect generalises from this. Only lists have been implemented here, not continuous variables.

<code>Prospect[event1, event2, Pr[event1]]</code>	a binary prospect
<code>Prospect[x_List, p_List]</code>	is an object with values x and associated probabilities p
<code>ProspectQ[q]</code>	tests whether q is a prospect
<code>ProspectEV[x_Prospect]</code>	gives the expected value of prospect x

The function Spread recognises Prospects too.

```
ex1 = Prospect[a, b, p];

ProspectEV[ex1]

b (1 - p) + a p

ex2 = Prospect[{c, d, e}, {.3, .3, .4}];

ProspectEV[%]

0.3 c + 0.3 d + 0.4 e
```

The following is an example application. A ship arriving at the harbour is checked on its official papers, its physical condition, and its crew. Ships with adequate papers may suffer problems with either the physical condition or the crew, but not both. In fact, 5% has physical problems causing a delay of 3 days, and in addition 3% has problems with the crew, causing a delay of 1 day. About 5% of the ships have deficient papers, and there is one day of delay on average in retrieving the missing information. This

delay comes on top of what other delays there will be. Of the ships with deficient papers, 10% have physical problems, and of these 20% have problems with the crew causing a delay of 5 days while those without crew problems have a delay of only 1 day. Of the ships with deficient papers but without physical problems, 10% has problems with the crew, causing an additional delay of 3 days.

- We first record all information as equations and rules of the form `label → Prospect[ ]`. We then separate equations and rules, and use them in a `SolveFrom` construction. We find that the expected delay is 0.2435 days.

```
info = {EVstart == ProspectEV[start],
start → Prospect[DefectivePapers, GoodPapers, .05],
DefectivePapers == 1 + ProspectEV[DP],
DP → Prospect[BadCondition, FairCondition, .10],
BadCondition == ProspectEV[Crew1],
Crew1 → Prospect[5, 1, .20],
FairCondition == ProspectEV[Crew2],
Crew2 → Prospect[3, 0, .10],
GoodPapers == ProspectEV[GP],
GP → Prospect[{3, 1}, {0.05, 0.03}]};

SolveFrom[Select[info, EquationQ], Select[info, RuleQ]]

{{EVstart == 0.05 DefectivePapers + 0.95 GoodPapers,
DefectivePapers == 0.1 BadCondition + 0.9 FairCondition + 1, BadCondition == 1.8,
FairCondition == 0.3, GoodPapers == 0.18}, {start → Prospect(DefectivePapers, GoodPapers, 0.05),
DP → Prospect(BadCondition, FairCondition, 0.1), Crew1 → Prospect(5, 1, 0.2),
Crew2 → Prospect(3, 0, 0.1), GP → Prospect({3, 1}, {0.05, 0.03})}, {{BadCondition → 1.8,
DefectivePapers → 1.45, EVstart → 0.2435, FairCondition → 0.3, GoodPapers → 0.18}}}
```

Note: See section 5.9.11 for a decision tree that has not just probabilistic data.

### 6.7.3 Valuing prospects

States of the world must have the same dimensions or be valued in order for us to be able to compare them or add them. If the dimensions are not the same then we use money or utility.

We use Utility to stand for non-stochastic utility that evaluates certain states of the world, and ProspectUtility for stochastic utility that for example weighs expected value and spread. See the use of  $U(\sigma, \mu)$  in finance. See the discussion on certainty equivalence in section 6.8 below for more details.

<code>CreateProspect [n_Integer]</code>	creates a prospect with n states and probabilities
<code>CreateProspect [Price, n_Integer]</code>	idem with n priced states and probabilities
<code>CreateProspect[Utility, n_Integer]</code>	idem with n utility states and probabilities
<code>ToUtility[q_Prospect]</code>	makes a Von Neumann – Morgenstern prospect with utilities of the outcomes
<code>ToExpectedUtility[q_Prospect]</code>	combines ToUtility and ProspectEV
<code>ProspectUtility[x_Prospect, crit__]</code>	gives the utility of prospect x based the criteria.

If the Risk package is loaded, then default criteria for ProspectUtility are ProspectEV, Spread and Risk.

- Though states can have different dimensions, evaluation requires that they must be valued by money or utility - or they must have the same dimension (like dimensionless rates of return).

```
moreDims = CreateProspect[2]
```

```
Prospect({State(1), State(2)}, {Pr(1), Pr(2)})
```

```
withMoney = CreateProspect[Price, 2]
```

```
Prospect({Price(1) State(1), Price(2) State(2)}, {Pr(1), Pr(2)})
```

```
withUtility = CreateProspect[Utility, 2]
```

```
Prospect({Utility(State(1)), Utility(State(2))}, {Pr(1), Pr(2)})
```

- The following might be conceptually dangerous since it means that we take the utilities of sums of money. This could be alright, though, if we were to use (money) (chain) indices.

```
ToExpectedUtility[withMoney]
```

```
Pr(1) Utility(Price(1) State(1)) + Pr(2) Utility(Price(2) State(2))
```

- This is the Von Neumann - Morgenstern criterion without the latter money complexity.

```
ProspectEV[withUtility]
```

```
Pr(1) Utility(State(1)) + Pr(2) Utility(State(2))
```

- The following would be the utility evaluation in the  $\{\sigma, \mu\}$  space as is common in finance. If we limit our attention to dimensionless rates of return, then the addition is no problem.

**ProspectUtility[ex1, Spread, ProspectEV]**

$$\text{Utility}\left(\sqrt{p(-pa + a - b(1-p))^2 + (1-p)(-(1-p)b + b - ap)^2}, b(1-p) + ap\right)$$

### 6.7.4 Subroutines

The following is useful when one writes routines and wants a uniform input format.

<b>ProspectReList[q]</b>	changes a binary prospect q into a prospect with lists
--------------------------	--

**ProspectReList[ex1]**

Prospect({a, b}, {p, 1 - p})

<b>ProspectInnerSort[q_Prospect(, p)]</b>
sorts the states and keeps the probabilities right, with p an ordering function

**ProspectInnerSort[Prospect[{x, q, a}, {.5, .1, .4}]]**

Prospect({a, q, x}, {0.4, 0.1, 0.5})

<b>ProspectApply[f, x]</b>	applies f to Prospects in x
----------------------------	-----------------------------

**ProspectApply[Spread, {}, func[σ → Prospect[1, 2, .3]]]**

{}, func(σ → 0.458258)}

In later discussions it appears useful to 'put in' or 'take out' values.

<code>PutIn [x+ ...+q_Prospect+ ...+y]</code>	for a single q includes the additions into the prospect
<code>PutIn [a_List, q_Prospect]</code>	takes in the strategy for playing the prospect
<code>TakeOut [q_Prospect, f]</code>	takes out a f[profit, loss]; f = Max takes out profits, f = Min takes out losses.
<code>TakeOut [q, Profit     Loss]</code>	for simple prospects: uses positions

```
Prospect[10] + 56 + test

test + Prospect(10, 0, 1) + 56

PutIn[%]

Prospect(test + 66, test + 56, 1)

TakeOut[%, Profit]

test + Prospect(0, -10, 1) + 66

Prospect[{1, 2, 3, 4}, Laplace[4]]

Prospect({1, 2, 3, 4}, {1/4, 1/4, 1/4, 1/4})

TakeOut[%, Max]

Prospect({-3, -2, -1, 0}, {1/4, 1/4, 1/4, 1/4}) + 4

PutIn[%]

Prospect({1, 2, 3, 4}, {1/4, 1/4, 1/4, 1/4})
```

6.7.5 Random drawing from Prospects

The following is a small but powerful routine to actually draw randomly from a prospect. You can also define a strategy how much to bet at each turn.



<code>Draw[q_Prospect]</code>	draws from the probability density and returns the drawn outcome.
<code>Draw[a_List, Prospect[v, p]]</code>	draws with strategy a. Part of wealth (1 - Add[a]) is retained, and a * v are the adjusted rewards with probabilities p. Multiply the outcomes to get the wealth after so many tries
<code>Draw[]</code>	draws from the prospect or strategy given earlier
<code>Draw[n_Integer, x___]</code>	draws n times

- This is a single draw for a "50% win 10, lose 10" proposition.

```
Draw[Prospect[10, -10, 1/2]]  
  
-10
```

- Here we draw 100 times, and then determine the frequencies (in this case percentages).

```
res = Draw[100, Prospect[{a, b, c}, {0.1, 0.3, 0.6}]];  
Frequencies[res]  
  

$$\begin{pmatrix} 6 & a \\ 32 & b \\ 62 & c \end{pmatrix}$$

```

Drawing repeatedly can be done with a strategy. Luenberger (1998) ch. 15 is essential for your ticket to wealth here. Suppose there is a wheel with areas {1/2, 1/3, 1/6} that pays out 3, 2, or 6 times the bet on the respective area. A player bets on these areas proportions of his wealth {a, b, c}, and thus has a Prospect[ {3a, 2b, 6c}, {1/2, 1/3, 1/6}], while he retains 1 - a - b - c. The trick is that retaining part of winnings biases the process towards growth. The outcomes can be multiplied to generate the wealth after repeated draws.

- Let us play the wheel for 3 times with {a, b, c}.

```
Draw[3, {a, b, c}, Prospect[{3, 2, 6}, {1/2, 1/3, 1/6}]]  
  
{-a + b - c + 1, 2a - b - c + 1, -a - b + 5c + 1}
```

- Let us play now 100 times with  $\{1/4, 0, 0\}$ , and determine the wealth remaining at the end by multiplication. We also take the  $1/100$  root, to determine the average growth rate.

```
res = Draw[100, {1/4, 0, 0}, Prospect[{3, 2, 6}, {1/2, 1/3, 1/6}]];
N[Times @@ res]^(1/100)
```

1.06804

- Let us find the optimal strategy. First define the formal prospect situation.

```
fp = PutIn[{a, b, c}, Prospect[{3, 2, 6}, {1/2, 1/3, 1/6}]]
```

$$-a - b - c + \text{Prospect}\left(\{3a, 2b, 6c\}, \left\{\frac{1}{2}, \frac{1}{3}, \frac{1}{6}\right\}\right) + 1$$

```
properfp = PutIn[fp]
```

$$\text{Prospect}\left(\{2a - b - c + 1, -a + b - c + 1, -a - b + 5c + 1\}, \left\{\frac{1}{2}, \frac{1}{3}, \frac{1}{6}\right\}\right)$$

- Take logarithmic utility to warrant the highest rate of growth

```
eu = ToExpectedUtility[properfp] /. Utility → Log
```

$$\frac{1}{2} \log(2a - b - c + 1) + \frac{1}{3} \log(-a + b - c + 1) + \frac{1}{6} \log(-a - b + 5c + 1)$$

- Setting up the first order conditions for a maximum.

```
Economics[Calculus, Print → False]
```

```
foc = Foc[eu, {a, b, c}]
```

$$\left\{ \begin{aligned} -\frac{1}{3(-a+b-c+1)} - \frac{1}{6(-a-b+5c+1)} + \frac{1}{2a-b-c+1} &== 0, \\ \frac{1}{3(-a+b-c+1)} - \frac{1}{6(-a-b+5c+1)} - \frac{1}{2(2a-b-c+1)} &== 0, \\ -\frac{1}{3(-a+b-c+1)} + \frac{5}{6(-a-b+5c+1)} - \frac{1}{2(2a-b-c+1)} &== 0 \end{aligned} \right\}$$

- We can show already that  $\{1/4, 0, 0\}$  is not optimal.

```
foc /. Thread[{a, b, c} → {1/4, 0, 0}]
```

{True, False, False}

- Let us make sure that we are dealing with a probability measure.

```
pm = foc /. c → 1 - a - b;
```

```
Solve[pm, {a, b}]
```

$$\left\{ \left\{ a \rightarrow \frac{1}{2}, b \rightarrow \frac{1}{3} \right\} \right\}$$

- We can compare the single period expected growth factors (where a factor = 1 + rate), see Luenberger (1998) for a discussion.

```
E^eu /. Thread[{a, b, c} → {1/4, 0, 0}]/N
```

```
1.06066
```

```
E^eu /. Thread[{a, b, c} → {1/2, 1/3, 1/6}]/N
```

```
1.06991
```

- Let us try this new strategy again. Well, bad luck !

```
res = Draw[100, {1/2, 1/3, 1/6}, Prospect[{3, 2, 6}, {1/2, 1/3, 1/6}]];
N[Times @@ res]^(1/100)
```

```
1.04561
```

### 6.7.6 Independent prospects

If there are more prospects, the question arises whether the probabilities are dependent or not. The following starts by assuming independence.

```
IDProspectsPrMatrix[x_Prospect, y_Prospect]
```

gives the probability matrix of the joint distribution of prospects x and y if it is assumed that they are independently distributed and the probabilities of the prospects are the marginals

```
IDProspectsPrMatrix[ex1, ex2]
```

$$\begin{pmatrix} 0.3 p & 0.3 p & 0.4 p \\ 0.3 (1 - p) & 0.3 (1 - p) & 0.4 (1 - p) \end{pmatrix}$$

Joining prospects gives rise to the JointProspect object (see below).

`JoinIDProspects [p_Prospect, q_Prospect]`

joins prospects p and q assuming that they have independent marginal probabilities. Note: the prospects are Flattened by default; add Not as a final argument if you don't want this

`JoinIDProspects[ex1, ex2]`

`JointProspect` $\left(\begin{pmatrix} a & a & a \\ b & b & b \end{pmatrix}, \begin{pmatrix} c & d & e \\ c & d & e \end{pmatrix}, \begin{pmatrix} 0.3p & 0.3p & 0.4p \\ 0.3(1-p) & 0.3(1-p) & 0.4(1-p) \end{pmatrix}\right)$

### 6.7.7 JointProspects

For simplicity, assume a two dimensional world, with different outcomes of the weather and of the oil price. Let us also assume two persons, who have different revenues depending upon different outcomes of the weather and the oil price. This is basically a game situation, but with the players not in control of the probabilities. Let us for simplicity assume that there are three classes for the weather and similarly for oil, so that we get 3 by 3 matrices of possible states of the world, their probabilities and revenues for either person. We collect all this information in a `JointProspect` object, that lists the revenue matrices and the probability matrix.

`JointProspect [x1_list, ..., xn_List, p_List]`

are returns to agents 1, ..., n all with probability p.  
The xi and p must be of equal dimension, of arbitrary sizes

`JointToMarginalProspects [q_JointProspect]`

makes single prospects that only use the marginal probabilities

`jp = JointProspect[{{100, -10, 5}, {1, 0, 0}, {-10, -10, 20}},  
                  {{-10, -10, 1}, {0, 0, 100}, {100, 10, 0}},  
                  {{2, 6, 6}, {6, 6, 12}, {8, 12, 12}}/70]`

`JointProspect` $\left(\begin{pmatrix} 100 & -10 & 5 \\ 1 & 0 & 0 \\ -10 & -10 & 20 \end{pmatrix}, \begin{pmatrix} -10 & -10 & 1 \\ 0 & 0 & 100 \\ 100 & 10 & 0 \end{pmatrix}, \begin{pmatrix} \frac{1}{35} & \frac{3}{35} & \frac{3}{35} \\ \frac{3}{35} & \frac{3}{35} & \frac{6}{35} \\ \frac{4}{35} & \frac{6}{35} & \frac{6}{35} \end{pmatrix}\right)$

One possibility is to define separate routines for joint prospects. Like for the expected values:

<code>JointProspectPrValue[q_JointProspect]</code>	gives the probability values
<code>JointProspectEV[q_JointProspect]</code>	gives the expected values of the various return matrices

`N[JointProspectEV[jp]]`  
  
`{3.08571, 29.2286}`

However, it appears to be more fruitful to decompose the joint prospect, and call on the existing routines that already work on prospects. Decomposing can be done by the following routines.

<code>JointToProspects[q_JointProspect]</code>	makes single prospects
<code>JointToProspects[q_JointProspect, weights_List]</code>	adds the qs with the weights
<code>JointToProspects[Flatten, q_JointProspect]</code>	also Flattens

These routines will be important in the subsequent discussion on risk, in section 6.8.

6.7.8 Formal development

The following turns to symbolic manipulation.

With  $\Pr[A]$  the probability of event A, we find for two events A and B:

- the joint probability  $\Pr[A, B] = \Pr[A \ \&\& \ B] = \Pr[A \cap B]$
- $\Pr[A \text{ or } B] = \Pr[A \cup B] = \Pr[A] + \Pr[B] - \Pr[A \cap B]$
- the conditional probability of A given B:  $\Pr[A; B] = \Pr[A | B] = \Pr[A, B] / \Pr[B]$
- the probability of the ordered event of first A and then B is  $\Pr[\{A, B\}]$

While these formulas are obvious enough, and while *Mathematica* is versatile on input formats, we, surprisingly, still have problems with finding a format for conditional probability.

- the semi-colon ";" is not accepted by *Mathematica* and
- "|" is an infix between two variables, so that  $\Pr[A, B | C, D]$  is interpreted as  $\Pr[A, (B | C), D]$ .

I have tried some five formats before settling down to the current format. My conclusions are:

- The conditional probability is best regarded as different from  $\Pr$ . The family resemblance does not mean identity. The conditional probability of A given B is denoted as  $\text{ConditionalPr}[A][B]$ . This notation will give no problems since  $\text{ConditionalPr}[A]$  will not be defined by itself.
- We can use  $\text{Prob}[A | B]$  as an output printing facility for structural  $\text{ConditionalPr}[A][B]$ .
- We may also use *TraditionalForm* to display nice output, without affecting the proper form.

<code>Pr[x__]</code>	gives the joint probability of events x
<code>ConditionalPr[x__][y__]</code>	gives the conditional probability of events x given events y
<code>Prob[x__]</code>	is a format for probability that humans can read better, but it is less tractible for <i>Mathematica</i>
<code>ToProb</code>	rules to turn <code>Pr[...]</code> into <code>Prob[...]</code> format
<code>FromProb</code>	rules to turn <code>Prob[...]</code> into <code>Pr[...]</code> format
<code>ConditionalPrForm[x]</code>	control TraditionalForm printing. x = On, Off or Blank

Users may, alternatively, opt to use `Prob` as the basic function, then use `FromProb` to get to the structural form that the routines recognise, and then apply `ToProb` again.

Of course, Wolfram Research may one day allow for an input format with `;` or `|`, but in the mean time we have found a representation that is structurally sound.

6.7.9 TraditionalForm output printing

- This gives untampered *Mathematica*.  
`ConditionalPrForm[Off]`  
`ConditionalPr[A, B][C, D]`  
  
`ConditionalPr(A, B)(C, D)`
- This gives a traditional display.  
`ConditionalPrForm[On]`  
`ConditionalPr[A, B][C, D]`  
  
`Pr[ A, B | C, D]`
- This just prints neatly - the default situation.  
`ConditionalPrForm[]`  
`ConditionalPr[A, B][C, D]`  
  
`ConditionalPr[ A, B][C, D]`

6.7.10 Event Universe

Success of an event is denoted by a symbol (say x) and failure is denoted by its negation (`!x` or `Not[x]`). An event can also be a composite event, such as `{x1, !x2}`. Some routines also accept `x1 && !x2`.

<code>FromEvent[p_Symbol]</code>	gives {p, Not[p]}
<code>Universe[p_Symbol]</code>	gives the {p, Not[p]} combinations for the various p symbols (events)

```
u = Universe[g1, g2, g3]
```

$$\begin{pmatrix} g1 & g2 & g3 \\ g1 & g2 & !g3 \\ g1 & !g2 & g3 \\ g1 & !g2 & !g3 \\ !g1 & g2 & g3 \\ !g1 & g2 & !g3 \\ !g1 & !g2 & g3 \\ !g1 & !g2 & !g3 \end{pmatrix}$$

### 6.7.11 Bayes

<p>Bayes[A, B][x__Rule]</p>	<p>is a SolveFrom application, using the equations</p> $\text{Pr}[A, B] == \text{Pr}[B] \text{ ConditionalPr}[A][B]$ $\text{Pr}[A, B] == \text{Pr}[A] \text{ ConditionalPr}[B][A]$
-----------------------------	--

```
Bayes[A, B][Pr[A, B] -> .15, Pr[A] -> .2, Pr[B] -> .3] // Last
{{ConditionalPr[A][B] -> 0.5, ConditionalPr[B][A] -> 0.75}}

Bayes[A, B][ConditionalPr[A][B] -> .5, Pr[A] -> .2, Pr[B] -> .3] // Last
{{Pr(A, B) -> 0.15, ConditionalPr[B][A] -> 0.75}}
```

### 6.7.12 To conditional

<p>ToConditional[expr]</p>	<p>turns the joint probability Pr in expr into the marginal probability times the conditional probability for as many variables in Pr</p>
<p>ToConditional[expr, symb]</p>	<p>does the above for symb only</p>

```
abc = ToConditional[Pr[a, b, c]]

Pr(a) (ConditionalPr[b][a]) (ConditionalPr[c][b, a])

% /. ToProb

Prob(a) Prob(b | a) Prob(c | a, b)
```

### 6.7.13 From conditional

<p>FromConditional[expr]</p>	<p>tries to determine the joint probabilities starting from the conditional and marginal ones</p>
------------------------------	---

```
FromConditional[abc]
```

```
Pr(a, b, c)
```

```
tes = ConditionalPr[a, e][t] ConditionalPr[a, e, d][b, c] //
ToConditional
```

```
(ConditionalPr[a][t]) (ConditionalPr[a][b, c])
  (ConditionalPr[d][a, b, c]) (ConditionalPr[e][a, t]) (ConditionalPr[e][d, a, b, c])
```

```
tes /. ToProb
```

```
Prob(a | t) Prob(a | b, c) Prob(e | a, t) Prob(d | a, b, c) Prob(e | a, b, c, d)
```

```
FromConditional[tes]
```

```
(ConditionalPr[a, e][t]) (ConditionalPr[a, d, e][b, c])
```

#### 6.7.14 Pr is orderless and ConditionalPr a bit

At start up, the attributes of Pr and ConditionalPr have been set at Orderless.

- This is as it should be.

```
Pr[A, B] === Pr[B, A]
```

```
True
```

```
ConditionalPr[A, B][C, D] === ConditionalPr[B, A][C, D]
```

```
True
```

- This is not as it should be.

```
ConditionalPr[A, B][C, D] === ConditionalPr[A, B][D, C]
```

```
False
```

Ordered probabilities are denoted with lists, so that  $\text{Pr}\{A, B\}$  gives the probability of the ordered sequence  $\{A, B\}$ . This is much better than `ClearAttributes[Pr, Orderless]` and `ClearAttributes[ConditionalPr, Orderless]`.

Be aware of the subtleties of order. Traditional notation is awkward here. The probability of first a black card and then a red one  $\text{Pr}\{B, R\} = \text{Pr}[B] \text{Pr}[R | \{B\}]$  differs from the probability of just a black and red card  $\text{Pr}\{B, R\}$  or  $\{R, B\} = \text{Pr}\{B, R\} + \text{Pr}\{R, B\}$ . And the latter differs from the probability  $\text{Pr}[B, R]$ , since the latter has nothing to do with sequential draws. By using the  $\{\}$  in the conditional part, as in  $\text{Pr}[R | \{B\}]$ , we can express that sequential draws are at hand.

An example can help. Let the universe have  $n$  elements,  $R$   $r$  elements,  $B$   $b$  elements and the intersection of  $R$  and  $B$  have  $m$  elements. Then:



$$\Pr[R] = r/n, \quad \Pr[B] = b/n, \quad \Pr[R, B] = m/n, \quad \Pr[R | B] = m/b$$

If the events are independent then  $m = 0$  (as would happen with black and red cards):

$$\Pr[\{B, R\}] = \Pr[B] \Pr[R | \{B\}] = \frac{b}{n} \frac{r}{n-1}$$

$$\Pr[\{R, B\}] = \Pr[R] \Pr[B | \{R\}] = \frac{r}{n} \frac{b}{n-1}$$

The sum  $\Pr[\{B, R\}] + \Pr[\{R, B\}]$  clearly may differ from  $\Pr[R, B] = 0$ .

- In the special case that  $n = b + r$ , this sum is:

$$\frac{b r}{n (n - 1)} + \frac{r b}{n (n - 1)} \quad /. \quad b \rightarrow n - r \quad // \quad \text{Simplify}$$

$$\frac{2 (n - r) r}{(n - 1) n}$$

- This is actually from the hypergeometric distribution.

$$\text{Binomial}[n - r, 1] \text{Binomial}[r, 1] / \text{Binomial}[n, 2]$$

$$\frac{2 (n - r) r}{(n - 1) n}$$

If the events are dependent then  $m \neq 0$  (as would happen with black cards and picture cards):

$$\Pr[\{B, R\}] = \Pr[B] \Pr[R | \{B\}] = \frac{b}{n} \left( \frac{m}{b} \frac{r-1}{n-1} + \frac{b-m}{b} \frac{r}{n-1} \right)$$

$$\Pr[\{R, B\}] = \Pr[R] \Pr[B | \{R\}] = \frac{r}{n} \left( \frac{m}{r} \frac{b-1}{n-1} + \frac{r-m}{r} \frac{b}{n-1} \right)$$

### 6.7.15 Example of ConditionalPr

In "event trees", where one event branches out in different subsequent events, the total probability of a range of events is best computed by using the conditional probabilities. The following is an example of this. We first need to introduce the SequentialDraws routine.

```
SequentialDraws[x, y:{}, opts]
```

gives the probability of event x given that event y has occurred before

Options[SequentialDraws] state the start values for the universe NumberOfElements  $\rightarrow$  n and the NumberOfFailures  $\rightarrow$  f therein. Without earlier draws, the chance of success is  $1 - f/n$ .

If x is a composite event, like  $\{x_1, x_2, !x_3\}$ , then this is taken by default as an ordered sequence.

**Options[SequentialDraws]**

{NumberOfElements → 100, NumberOfFailures → 5, Order → True}

- The default options say that there is a 5 % chance on failure. Application of the routine with these parameters to above universe u gives:

**SequentialDraws /@ u**

$\left\{ \frac{27683}{32340}, \frac{893}{19404}, \frac{893}{19404}, \frac{19}{9702}, \frac{893}{19404}, \frac{19}{9702}, \frac{19}{9702}, \frac{1}{16170} \right\}$

**Add[%]**

1

Let us regard the example, where a batch of products is accepted (a) when two subsequent draws are good, (b) or one of these two is bad, but a third draw is good. This appears a lax testing rule, with only a 0.6 % chance of rejecting a batch.

- If draw  $i$  is good, then  $g_i$ , otherwise  $!g_i$ . Two subsequent good draws are  $\{g_1, g_2\}$  and this will be a (composite) event for SequentialDraws.

**Pr[Accept] = Pr[{g1, g2}] + Pr[{g1, !g2, g3}] + Pr[{!g1, g2, g3}]**

$\text{Pr}(\{g_1, g_2\}) + \text{Pr}(\{g_1, !g_2, g_3\}) + \text{Pr}(\{!g_1, g_2, g_3\})$

- This computes the total probability directly from Pr.

**Pr[Accept] /. Pr[{x\_\_}] := SequentialDraws[{x}] // N**

0.994063

**Pr[Reject] = 1 - %**

0.00593692

- We can find the same result when we translate all Pr statements into ConditionalPr statements. However, ToConditional[ ] has not been defined for Pr[{x, y}] formats. We can get rid of the { } if we keep in mind that we have sequential draws.

**pra = ToConditional[Pr[Accept] /. Pr[{x\_\_}] := Pr[x]]**

$\text{Pr}(g_1) (\text{ConditionalPr}[g_2][g_1]) + \text{Pr}(g_2) (\text{ConditionalPr}[g_3][g_2]) (\text{ConditionalPr}[!g_1][g_3, g_2]) +$   
 $\text{Pr}(g_1) (\text{ConditionalPr}[g_3][g_1]) (\text{ConditionalPr}[!g_2][g_3, g_1])$

- The conditional probabilities may read better in Prob printing form.

**pra /. ToProb**

$\text{Prob}(g_1) \text{Prob}(g_2 | g_1) + \text{Prob}(g_2) \text{Prob}(g_3 | g_2) \text{Prob}(!g_1 | g_2, g_3) +$   
 $\text{Prob}(g_1) \text{Prob}(g_3 | g_1) \text{Prob}(!g_2 | g_1, g_3)$

- If we assign  $\text{Pr}[A \mid B]$  to `SequentialDraws` in the following manner, it actually means interpreting  $\text{Pr}[A \mid B]$  as  $\text{Pr}[A \mid \{B\}]$ . This completes the  $\text{Pr}[\{x\}] \rightarrow \text{Pr}[x]$  trick.

```
pra /. {Pr[x__] :> SequentialDraws[{x}],
       ConditionalPr[x__][y__] :> SequentialDraws[{x}, {y}]} // N

0.994063
```

Note: Alternatively, if you want a statement  $x$  to be exemplary of any order, with only the number of failures as the criterion, then set the option `Order`  $\rightarrow$  `False`; the probability results then are equal to the use of the `HypergeometricDistribution`.

### 6.7.16 Utilities

<code>UnitPr[n_Integer, len_Integer]</code>	creates a vector of length <code>len</code> , with all weight on position <code>n</code>
<code>NormalisePr[x_List]</code>	normalises to the unit simplex while first cutting of values below 0 and above 1
<code>Laplace[n_Integer: 2]</code>	uniform distribution over <code>n</code> states

Note: The normalisation  $x / \text{Add}[x]$  is too simple to implement with a separate routine.

```
UnitPr[4, 5]
```

```
{0, 0, 0, 1, 0}
```

```
NormalisePr[Append[%, 10]]
```

```
{0, 0, 0, 1/2, 0, 1/2}
```

```
Laplace[]
```

```
{1/2, 1/2}
```

<code>PrDomain[p_List]</code>	produces the conditions that the $p$ is on the unit simplex
-------------------------------	---

```
PrDomain[{p, q}]
```

```
{0 ≤ p ≤ 1, 0 ≤ q ≤ 1, 0 ≤ p + q ≤ 1}
```

Also relevant is the parametric presentation of probabilities. An application in this way can be found in section 3.7.8 on Kuhn-Tucker optimisation.

<code>PrDomain[Solve, p_List, labda_Symbol]</code>	solves for labda, requiring that the p (labda) are on the unit simplex
<code>PrDomain[Solve, List, p_List, labda_Symbol]</code>	maps over the separate entries only
<code>PrDomain[Inverse, p_List, labda_Symbol]</code>	solves the $p(\text{labda}) == \text{Pr}$ , and gives $\text{labda} = p^{(-1)}[\text{Pr}]$
<code>PrDomain[Domain, Min   Max, p_List, labda_Symbol]</code>	produces a domain statement that can be used in a plot
<code>PrDomain[Plot, p_List, {labda_Symbol, n, m}]</code>	plots the values of $p(\text{labda})$ with special attention to the unit simplex
<code>PrDomain[Plot, Inverse, p_List, labda_Symbol]</code>	looks only at the $[0, 1]$ domain, and if the $p(\text{labda})$ don't intersect, then there is no useful value for labda

6.7.17 Appendix: Rejected formats

One format was `Conditional[Pr]`. This was rejected because of the brackets.

One format was `Pr[Conditional[A, ..., B][C, ..., D]]`. This adds brackets too. Also, there is a possible confusion with the 'marginal probability of `Conditional[A, ..., B][C, ..., D]`'.

One format was `Pr[A, ..., B][C, ..., D]` and thus without `Conditiona[Pr]`. This required `Prob[A, ..., B][ ]` for the joint probability. But the `[ ]` is not natural. Also `Pr[A, ..., B]` would have to remain undefined otherwise it gives a result as `(value)[C, ..., D]`.

One format is:

- `Pr[A && B]` for the joint probability of A and B
- `Pr[A && B, C && D]` for the probability of A & B conditional on C & D

This was rejected (a) for readability, especially for longer `A && B && C && D && E` or `And[A, B, C, D, E]`, and (b) for the abnormal use of the comma. But, perhaps you find this acceptable:

```
Pr[And[A, B, C] | And[D, E, F]]
```

One format is  $\text{Pr}[\{A, B\} \mid \{C, D\}]$  giving  $\text{Prob}[A, B \mid C, D]$ . But lists are not orderless, and making them orderless would upset many other routines. Also, all those brackets don't read nicely. Not tried was  $\text{Pr}[A, B, \text{"|"}, C, D]$ .

One format is  $\text{Pr}[A, \dots, B, \text{Cond}[C, \dots, D]]$ , but this breaks down on Orderlessness of Pr.

Then there is  $\text{Pr}[\text{Var}[A, \dots B], \text{Cond}[C, \dots, D]]$ . By defining Var and Cond as Orderless, one could easily determine that  $\text{Pr}[\text{Var}[A, B]] = \text{Pr}[\text{Var}[B, A]]$ . However, this actually comes down on using  $\text{Pr}[A \&\& B]$ , with  $\text{Var} = \text{And}$ .

### 6.7.18 Appendix: bordered 2D Pr

A bordered 2D probability matrix is a  $\{n, m\}$  matrix of which the last row (column) is the sum of the preceding rows (columns).

```
Bordered2DPrSolve[x_?MatrixQ , X_Symbol:XYXZX] solves □
Bordered2DPrToConditionals[x_?MatrixQ]
Bordered2DPrToEquations[x_?MatrixQ]
subroutines, for inner cell conditional probabilities and equations
```

**.29** □ **a**  
**Bordered2DPrSolve**[ □ □ □ ]  
**.35** □ **1**

$$\begin{pmatrix} 0.29 & 1.a - 0.29 & a \\ 0.06 & 0.94 - 1.a & 1. - 1.a \\ 0.35 & 0.65 & 1 \end{pmatrix}$$

# 6.8 Risk

## 6.8.1 Summary

This section develops the notion of risk. There is also a short discussion of insurance.

```
Economics[Risk, Finance`Insurance]
```

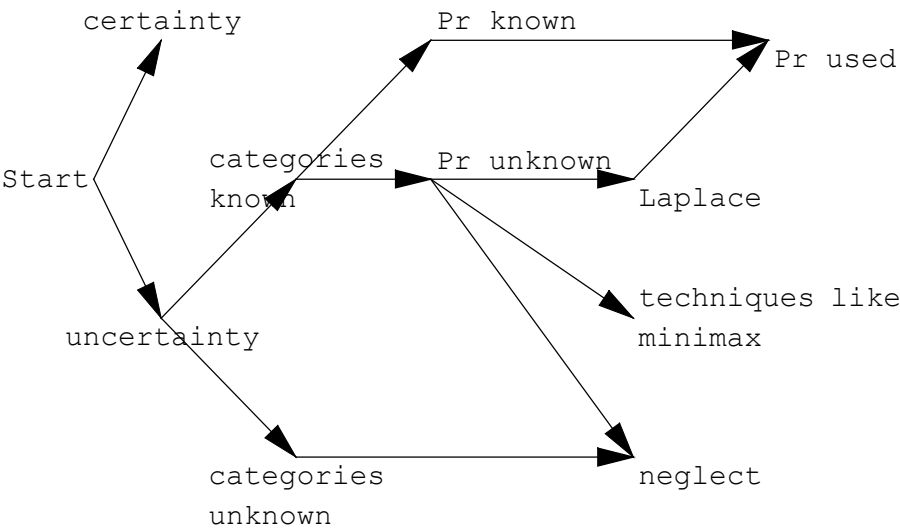
## 6.8.2 Introduction

We better understand our subject when we first have a foundation for uncertainty and probability:

- (1) First there is the distinction between *certainty* and *uncertainty*.
- (2) Uncertainty forks into *known categories* and *unknown categories*.
- (4) Known categories forks into *known* and *unknown probabilities*.
- (3) Unknown probabilities forks into *assuming a uniform distribution* (Laplace) or use non-probabilistic *techniques like minimax* or *neglect*.

These definitions can be clarified by the following plot.

```
UncertaintyDefinitionsPlot[];
```



What do we mean by 'risk' ? A.S. Hornby's (1985) "Oxford Advanced Learner's Dictionary of Current English" defines 'risk' as: "(instance of) possibility or chance of meeting danger, suffering loss, injury, etc." Also: "at the ~ risk of / at ~ of, with the possibility of (loss etc.)".

Thus, if there are possible outcomes  $O = \{o_1, o_2, \dots, o_n\}$ , then the situation is risky if at least one of the  $o$ 's represents a loss. The risks themselves are the  $o_i$  that are those losses. The risks factors are the dimensions or positions of the risky outcomes, the  $i$ 's (or the causes that make such positions to be filled).

We will use the term 'valued risk' when a risk is valued with money or utility. When all risks have been made comparable by valuing them, then we can add them, and we will use the term *expected risk value* for the *expected value of the 'valued risks'*. Then, crucially, once these definitions are well understood, then we may also use '*the risk*' for the expected risk value.

With such understanding, risk will be  $\rho = -E[x < 0]$ . Note that the term 'risk' has not been used in the 4 points above, so that an independent definition is possible.

*Relative* risk is defined as  $\rho(t) = t - E[x < t]$  for some target level  $t$ . Risk (or *absolute* risk) takes  $t = 0$ , and relative risk would allow for a different target level. An interesting application is when  $x$  is a stochastic rate of return and  $r$  the certain rate, so that there is relative risk  $\rho(r) = r - E[x < r]$ . This relative risk answers the question: What is the probable loss with respect to a target return of  $r$ ? Here,  $r - \rho(r) = E[x < r]$  gives the weight of underperformance in the total target return (which weight has to be compensated by probable profits to achieve the target).

*Conditional* (relative) risk is defined as  $\kappa(t) = t - E[x | x < t]$  for some target level  $t$ . With respect to rates of return, conditional risk  $\kappa(r)$  answers the question: What would one expect to lose with respect to  $r$ , if earnings actually underperform and fall below  $r$ . Indeed,  $r - \kappa(r)$  would give your expected return when actually underperforming.

Conditional risk is related to relative risk by the property that  $E[x | x < t] = E[x < t] / \Pr[x < t]$ . The probable loss thus is corrected for the probability of the loss. Or, the probability measure in the expectation is corrected so that a density is taken that sums to 1.

Note that above definitions are proper in the sense that they conform to every day parlance and the definitions provided by Hornby's dictionary op. cit.. The definitions provided here however differ from other definitions within the economics literature. First there are the definitions of Knight (1921) that have been adopted widely in economics, as for example in The New Palgrave (1998:III:358). Or it has become custom in finance to associate risk with the standard deviation. Cool (1999), "Proper definitions of risk and uncertainty" (available in the Pack and on the internet) further discusses why such alternatives generate conceptual problems and why the current definitions are preferable.

Below we will develop these notions for simple binary prospects, multidimensional prospects, joint prospects, and continuous probability densities.

6.8.3 Prospects and risk

Prospects have been discussed in section 6.7 on probability. These prospects are perfect to continue the discussion on risk. The typical binary risky prospect recognises only win or lose situations. Let *profit*  $\geq 0$  stand for the positive return of a prospect, and  $-loss \leq 0$  for the negative return of a prospect, where loss is the absolute value of that negative return. The probability of a profit is *p*, the probability of a loss is (1 - *p*). The multidimensional prospect generalises from this. In the discussion below we will also investigate the projection of multidimensional prospects into the simply binary risky prospect, because of the idea that the latter is a good summary or psychological vehicle to convey the information about the risk.

Prospect [*profit*,  $-loss$ , Pr[Profit]]    convention !

Note that loss is an absolute value, and that a real loss must be entered as a negative value.

```
eg = Prospect[Profit, -Loss, p];

ProspectEV[eg]

p Profit - Loss (1 - p)
```

It is important to see that there is nothing in the concept of a Prospect that requires that the second position is reserved for a loss. A binary prospect may well give two profits or two losses. Therefor, the proper definition of Risk requires a formal test on the values of the entries. For formal discussions, however, the formal test reads awkward, and we indeed may resort to the positional convention.

Risk [ <i>q</i> ]	gives the risk, i.e. the expected absolute loss
RiskPr [ <i>q</i> ]	gives the cumulative probability of a loss in prospect <i>q</i>
RiskyQ [ <i>q</i> ]	gives False if not risky, True if <i>q</i> is risky (at least one negative possible outcome with nonzero probability)

All defined on prospects *q*. These routines use an If[Negative...] construction since its derivative is defined. For reading, use IfNegativeToMinRule or IfNegativeToMaxRule. On the binary Prospect, if Options[Risk] have Position  $\rightarrow$  True, the second position is taken as the loss.

- This is the proper risk definition.

```
Risk[eg]

-(1 - p) If[Negative[-Loss], -Loss, 0] - p If[Negative[Profit], Profit, 0]

RiskyQ[eg]

Negative[-Loss]  $\Rightarrow$  True
```



- For formal analysis, however, we may resort to the convention that the second position is the loss. The Position option works only for binary risk (in non-list-format).

```
SetOptions[Risk, Position → True];
Risk[eg]

Loss(1 - p)

RiskyQ[eg]

True
```

The default option setting is Position → False, since it is not obvious that you would be using Prospects formally. It also reminds you about the positional convention.

- If you set the option to True, you can still have formal testing by using the list format. Also, routines like ProspectPrValue internally call ProspectReList, and thus are not affected by the Position option.

```
Risk[ProspectReList[eg]]

-(1 - p) If[Negative[-Loss], -Loss, 0] - p If[Negative[Profit], Profit, 0]
```

- This gives the risk probabilities.

```
RiskPr[eg]

1 - p

eglis = Prospect[{-1, 2, -2, 3}, {0.2, 0.4, 0.3, 0.1}];

RiskPr[eglis]

0.5
```

Since risk selects the negative values from the states of the world, and since some of the prospects are algebraic rather than numeric, the Risk function basically uses a conditional statement. Where the choice was between Negative[ ], Min[ ] or Max[ ], the use of Negative[ ] has been chosen, since the derivative D[ ] still applies. Since an If[Negative[ ], ..] statements reads difficult at times, there is the possibility to replace it by the following.

IfNegativeToMinRule	A rule that changes an If[Negative...] statement into a Min condition. The derivative of If[Negative...] is defined, but Min reads better
IfNegativeToMaxRule	A rule that changes an If[Negative...] statement into a Max condition. The derivative of If[Negative...] is defined, but Max reads better

6.8.4 Risk models

The first model presumes binary risk, and therefor is a good introduction to the subject.

<code>RiskModel[{rules}]</code>	is a <code>SolveFrom</code> application, default with <code>RiskEquations</code>
<code>RiskEquations</code>	used in <code>RiskModel</code> . Profit, Loss and Risk are nonnegative values, and the expected value is <code>Prob Profit – (1–Prob) Loss</code>
<code>RelativeRiskEquations</code>	show the relationship between relative and conditional risk

```
RiskEquations  
  
{Risk == Loss (1 – Prob), ExpectedValue == Prob Profit – Risk}  
  
RiskModel[{Profit → 0.6, Prob → 0.5, Risk → 0.2}] // Last  
  
{ExpectedValue → 0.1, Loss → 0.4}
```

The relative risk model shows the relationship between relative and conditional risk.

```
SetOptions[RiskModel, Equations → RelativeRiskEquations]  
  
{Equations → {RelativeRisk == Target – Expectation(LessThanTarget),  
ConditionalRisk == Target – Expectation(Conditional),  
Expectation(Conditional) ==  $\frac{\text{Expectation(LessThanTarget)}}{\text{RiskPr}}$ }}
```

```
RiskModel[{}, {Expectation["LessThanTarget"],  
Expectation["Conditional"]}] // Last // FullSimplify  
  
{ConditionalRisk →  $\frac{\text{RelativeRisk} + (\text{RiskPr} - 1) \text{Target}}{\text{RiskPr}}$ }
```

6.8.5 Risk concepts

<code>RiskRatio[q]</code>	gives the risk ratio as <code>Risk[q] / (E[q] + Risk[q])</code>
<code>RelativeRisk[q, t:Automatic]</code>	gives <code>t – E[x &lt; t]</code> for target <code>t</code> (default <code>E[x]</code> )
<code>ConditionalRisk[q, t:Automatic]</code>	gives <code>t – E[x   x &lt; t]</code> for target <code>t</code> (default <code>E[x]</code> )

Note: These are defined, like `Risk[q]`, for `q` a simple or list prospect or probability density function (see below). `Risk` and `RiskRatio` are also defined for riskets (see below). Note: `E[ ]` is not a function here, but an abbreviation for the expected value. Note: `Risk[pdf]` for continuous density pdf uses `RelativeRisk[pdf, 0]`. Note: If `q` is a pdf then `Domain[pdf] = {min, max}` must have been defined.

**Risk[eg]**

Loss (1 - p)

**RiskRatio[eg]**

$$\frac{\text{Loss}(1 - p)}{p \text{ Profit}}$$

**RelativeRisk[eg]**

*RelativeRisk::form: RelativeRisk is not defined for formal prospects*

<b>RiskRatioSplit</b> [ <i>q, crit</i> ]	splits a prospect <i>q</i> into {−loss/profit, 1/ <i>p</i> − 1}. Multiplying these elements gives the RiskRatio.
---	--

Note: If *q* is an object with Lists, then *q* is projected first, using *crit*. If *q* itself is a list, then RiskRatioSplit is mapped over it.

**RiskRatioSplit[eg]**

$$\left\{ \frac{\text{Loss}}{\text{Profit}}, \frac{1}{p} - 1 \right\}$$

Given the relevance of expected value, spread, risk and loss probability, we define the Risket object.

<b>Risket</b> [ <i>μ, σ, ρ, 1 - p</i> ]	is an object with expected value <i>μ</i> , spread <i>σ</i> , risk <i>ρ</i> and probability <i>p</i> of profit
<b>ToRisket</b> [ <i>q</i> ]	turns a Prospect object or <i>pdf</i> or data vector into a Risket object
<b>ToProspect</b> [ <i>x</i> ]	turns a Risket object or <i>pdf</i> or data vector into a Prospect object

Note: ToProspect[*pdf*] turns the probability density into a prospect, assuming that risk and expected value remain the same. ToProspect is here a projection like ProspectProjection (see below). For a vector of outcomes the assumption is equal probabilities. ToRisket has default option Spread → StandardDeviation (division by *n*-1). Use Spread to divide by *n* and False if you want the spread implied by applying ToProspect again.

Importantly, the Risket object can contain more information than a Prospect. Going from a Risket to a Prospect in generally is a projection, where information about the true spread can be lost.

- This gives a clear formal result since we now work with Position → True in Options[Risk].

**ToRisket[eg]**

$$\text{Risket}(p \text{ Profit} - \text{Loss}(1 - p), \sqrt{((1 - p)((1 - p) \text{ Loss} - \text{Loss} - p \text{ Profit})^2 + p (\text{Loss}(1 - p) - p \text{ Profit} + \text{Profit})^2)}, \text{Loss}(1 - p), 1 - p)$$

- This function neglects the spread, since for true binary prospects all information is in the expected value, risk and risk probability. If the original is not truly binary, then information is lost.

**ToProspect[%]**

Prospect(Profit, -Loss, *p*)

If one wishes to determine a prospect by using the spread instead, the following would be useful.

**ProspectInverse[*mean, spread, risk*]** gives a binary prospect with these properties

Note that there can be more solutions.

**ProspectInverse[0.5, 0.6, 0.1]**

{Prospect(0.626795, -2.33923, 0.957251), Prospect(0.973205, -0.26077, 0.61652)}

### 6.8.6 Projection of moredimensional prospects and outcomes

**ProspectProjection[*q (, crit)*]** projects a prospect *q* into a binary prospect using criterion *crit*

Note: The default projection criterion uses expected value, risk and the probability. A criterion is Spread, and uses expected value, risk and spread, using ProjectInverse. You can define ProspectProjection for other criteria.

**ProspectProjection[eglis]**

Prospect(2.2, -1.6, 0.5)

- Compare the spreads of the projection and the original one:

**ToRisket /@ {%, eglis}**

{Risket(0.3, 1.9, 0.8, 0.5), Risket(0.3, 1.95192, 0.8, 0.5)}

The issue of projection relates directly to the issue of statistical analysis of a series of outcomes. A series of data on some random event can be seen as a prospect where all outcomes had a chance of occurring.

- Some data on the rate of return of a company stock.

**Robeco = {13.7, -16.6, 11.2, 9.6, 30., -7.1};**

- These data can be turned into a prospect - default with Laplace's assumption.

**res1 = ToProspect[Robeco]**

Prospect({13.7, -16.6, 11.2, 9.6, 30., -7.1}, { $\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}$ })

Of course, the translation of a series of outcomes to a prospect basically depends upon your own judgement, and you have to decide about e.g. bin sizes and priors. This discussion just may help shaping your thoughts. The simple point made here is that if we try to squeeze all data into a binary prospect, then we may lose information on the spread and/or risk. It may still be, though, that a simple binary prospect better conveys the risk, psychologically, than the spread.

- The risket object that follows from above Laplace:

```
ToRiset[res1]
```

```
Riset(6.8, 15.0212, 3.95,  $\frac{1}{3}$ )
```

- A risket object can also be found directly - but the standard deviation now differs because of the default option setting Spread → StandardDeviation has a division by n-1.

```
res2 = ToRiset[Robeco]
```

```
Riset(6.8, 16.4549, 3.95,  $\frac{1}{3}$ )
```

- If we squeeze above data into a binary prospect, then we find that we lose information on the standard deviation. To information exchange is based on  $\mu$ ,  $\rho$  and p only.

```
ToProspect[%]
```

```
Prospect(16.125, -11.85,  $\frac{2}{3}$ )
```

```
ToRiset[%]
```

```
Riset(6.8, 13.1875, 3.95,  $\frac{1}{3}$ )
```

- Another example is a ProspectProjection that wants to maintain the same mean, spread and risk: other probability estimates arise, and there are more estimates to choose from.

```
ProspectProjection[res1, Spread]
```

```
{Prospect(12.0764, -35.9634, 0.890166), Prospect(22.5131, -7.55982, 0.477501)}
```

```
ToRiset /@ %
```

```
{Riset(6.8, 15.0212, 3.95, 0.109834), Riset(6.8, 15.0212, 3.95, 0.522499)}
```

6.8.7 Subroutines

ProspectSort [*x\_List*, *crit\_* : *Risk*]

sorts the list of prospects *x* according to criterion *crit*,  
default *Risk*. The criterion function must generate a real value on a prospect. Output is  
an option list with the criterion values, the order sequence, and the reordered list *x*

**ProspectSort**[{*eg*, *eglis*}]

*Risk* → {0.8, Loss (1 – *p*)}, *Order* → {2, 1},  
*Sort* → {Prospect({–1, 2, –2, 3}, {0.2, 0.4, 0.3, 0.1}), Prospect(Profit, –Loss, *p*)}

ProspectPrValue [*q*]      gives the 'probability values'  
                                 of the prospect *q* (probability times value)

**ProspectPrValue**[*eglis*]

{–0.2, 0.8, –0.6, 0.3}

This is useful when you see the profit as a result of taking the risk.

ProspectSplit [*q*]      splits prospect *q* into binary probability  
                                 values {(1–*p*) loss, *p* profit}. Maps over *q* if it is a List

**ProspectSplit**[*eglis*]

{0.8, 1.1}

Remember that taking 'risk' as the expected value of the valued risks is only a special application. Normally the risks are the events by themselves. The following helps to identify the risk factors.

Risks [{*xi*, ...}, {*pi*, ...}, {*posi*, ...}]

is a risks object that gives the risks, with the *xi* only positive,  
the *pi* >= 0 not necessarily adding up to 1,  
and the positions in a prospect (thus identifying the risk factors)

Risks [*q*]      selects the risks in *q*, making a Risks object,  
                 with the States that are risky, in absolute values,  
                 with their probabilities and dimensions (positions, risk factors)

RisksFactors [*q*]      gives the dimensions (positions) of the risks

Note that binary prospects are first turned into list prospects.

```
Risks[eglis]

Risks({-1, -2}, {0.2, 0.3},  $\begin{pmatrix} 1 \\ 3 \end{pmatrix}$ )

RiskFactors[eglis]

 $\begin{pmatrix} 1 \\ 3 \end{pmatrix}$ 
```

### 6.8.8 Statistics

The following generates various statistics about prospects.

ProspectStatistics[  
q]

gives various statistics of prospect q;  
or maps over q if q is a list

ProspectStatistics[TableForm, q\_List, rowheadings:Character]

computes the statistics and directly shows them in TableForm. Default  
rowheadings are characters, Automatic will give numbers,  
and a list with values will use that list. The statistics are in Results

Let us define 5 example prospects, with PM a function that completes a probability measure.

```
pr[1] = Prospect[{5, -5, 10, 4}, {0.1, 0.1, 0.2, 0.6}];
pr[2] = Prospect[{0.5, -15, 10, -0.4}, {0.2, 0.1, 0.2, 0.5}];
pr[3] = Prospect[{0, 1.1, -3, 2}, PM[{0.21, Rest, 0.4, 0.3}]];
pr[4] = Prospect[{13.8, -1., 0.6, 1}, PM[{0.2, 0.2, 0.3, 0.3}]];
pr[5] = Prospect[{5, 1, 1, 2.}, PM[{0.1, Rest, 0.2, 0.06}]];

ProspectStatistics[TableForm, Array[pr, 5]]
```

	ProspectEV	Spread	Risk	RiskRatio	RiskPr
A	4.4	3.90384	0.5	0.102041	0.1
B	0.4	6.5169	1.7	0.809524	0.6
C	-0.501	2.15822	1.2	1.71674	0.4
D	3.04	5.42719	0.2	0.0617284	0.2
E	1.46	1.20349	0.	0.	0.

6.8.9 Prospect plotting

A useful feature is that the probabilities of 3D prospects can be plotted in a 2D triangle, that essentially is a transform of the 3D unit simplex. If the dimension is less than 3 then the 3rd dimension is set to 0. If the dimension is larger than 3 then the higher dimensions can be added. The triangle has sides  $2 / \sqrt{3}$ , and the corners are at  $c1 = \{0, 0\}$ ,  $c2 = \{2 / \sqrt{3}, 0\}$  and  $c3 = \{1 / \sqrt{3}, 1\}$ . A point in the triangle has the property that the distances to the sides add up to one. The prospect probabilities  $\{p1, p2, p3\}$  are plotted such that the distance from the plotted point to the side opposite to  $ci$  gives the probability  $pi$  of outcome  $i$ .

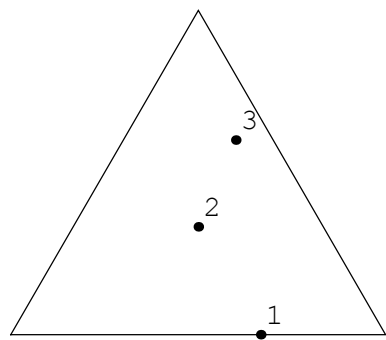
```
Prospect3DPrTriangle[  
  lis_List, opts]
```

plots probabilities of prospects in *lis* in a triangle

Note: The points are in Results[ProspectPr3DTriangle]. Note: Options are: Point → True (default) plots points, PointSize→ size (default .025) gives the size of these points, Label → Automatic (default) gives labels. The latter can also be a list, or None. If Point → False, then the labels are printed right on the co-ordinates. Note: Subroutines are: ProspectPr3DTriangle[] gives the graphics primitive of the triangle; while ProspectPr3DTriangle[Point, lis] gives those of the points.

- Point 1 of the simple prospect plots on the bottom side, since the distance from the bottom side is zero. The distance to the right side is 1/3 and to the left side 2/3. Point 2 plots in the middle, and lies on a line through point 1 parallel to the right side. Point 3 then is clear.

```
Prospect3DPrTriangle[{Prospect[3, -2, 1/3],  
  Prospect[{a, b, c}, {1, 1, 1}/3],  
  Prospect[{a, b, c}, {0.1, 0.3, 0.6}]}];
```



Above plots just the probabilities. The following include the values.

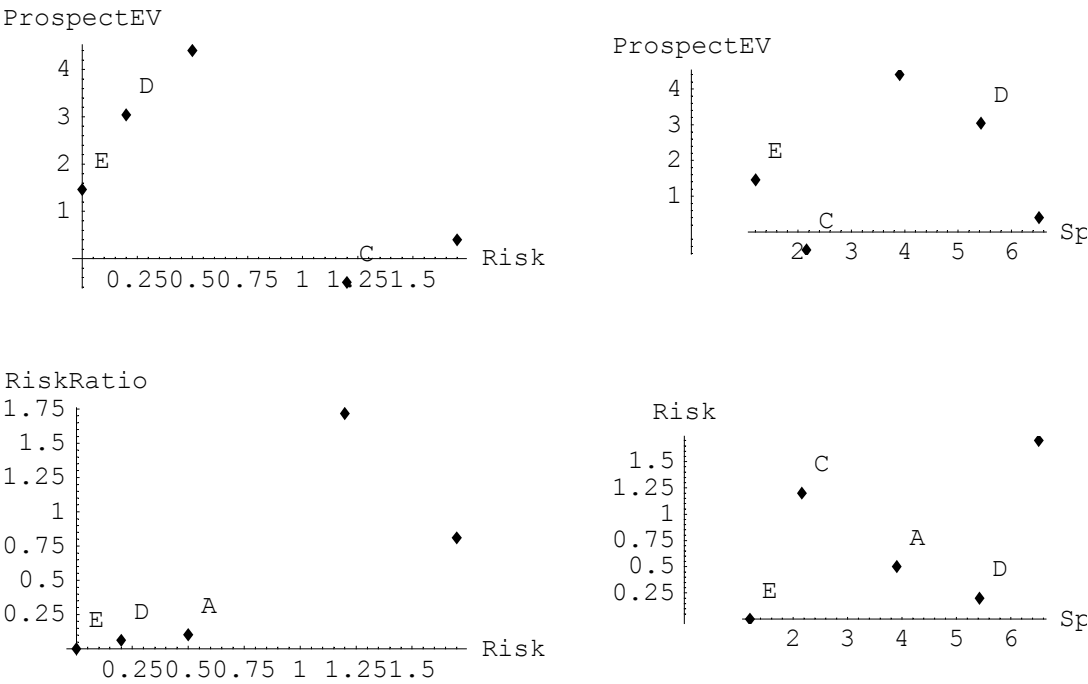


<code>ProspectPlot[x_List]</code>	plots the prospects x in the Expected Value, Risk and Spread space. Subroutines are:
<code>ProspectPlot[Set, x]</code>	sets the data to be plotted ( $\Leftrightarrow$ <code>SetOptions[ProspectPlot, Data <math>\rightarrow</math> ProspectStatistics[x]]</code> )
<code>ProspectPlot[x_Symbol, y_Symbol, opts___Rule]</code>	plots the keys x and y of these
<code>ProspectPlot[All]</code>	plots for the mentioned three keys
<code>ProspectPlot[q_Prospect, a_Symbol]</code>	plots Expected Value, Risk and Spread values of a prospect that is a function of a (in the domain [0, 1])

We can plot the five example prospects of the former section into various twodimensional spaces.

- The common finance analysis in the  $\{\sigma, \mu\}$  space will regard the line A-E as the efficiency border (that dominates the below and right). The point D would be considered to be dominated. However, in the  $\{\rho, \mu\}$  plot, the D takes a dominant position.

```
ProspectPlot[Array[pr, 5], AxesOrigin -> {0, 0}];
```

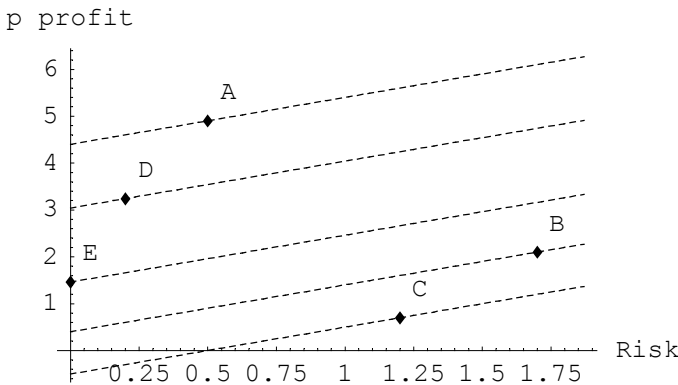


Another plot is for the ProspectSplit[ ] function discussed above.

<code>ProspectSplitPlot[x_List]</code>	sets data and plots them
<code>ProspectSplitPlot[Set, x_List]</code>	for a list x of prospects, sets the split data in the options
<code>ProspectSplitPlot[Data]</code>	plots these data
<code>ProspectSplitPlot[Line, x_List]</code>	sets, plots, and adds equal sum contours

- The following plots in the space of the probability values  $\{(1-p) \text{ loss}, p \text{ profit}\}$ . In this split space, the expected values can be found by subtracting the horizontal co-ordinate from the vertical co-ordinate. Contours of equal expected value can be found in the same way. The following plot clarifies that A has a much higher expected value than D, but a higher risk.

```
ProspectSplitPlot[Line, Array[pr, 5], AxesOrigin -> {0, 0}];
```



Another plot is for the ratio split (not plotted here).

```
RiskRatioSplit[eglis]
```

```
{0.727273, 1.}
```

<code>RiskRatioSplitPlot[x_List]</code>	sets data and plots them
<code>RiskRatioSplitPlot[Set, x_List]</code>	
	for a list x of prospects, sets the split ratio data in the options
<code>RiskRatioSplitPlot[Data]</code>	plots these data
<code>RiskRatioSplitPlot[Log, Data]</code>	changes the data in logs, and plots
<code>RiskRatioSplitPlot[Log, x_List]</code>	sets, turns in logs, plots
<code>RiskRatioSplitPlot[Log, Line, x_List]</code>	
	sets, turns in logs, plots, and adds equal sum contours

For the logs: zero elements are removed, and the log data are in Results[RiskRatioSplitPlot]

### 6.8.10 JointProspects and risk

We can further illustrate the meaning of the definition of risk by reproducing a small part of the common analysis of the Markowitz efficiency frontier in the  $\{\sigma, \mu\}$  space of two assets.

Reconsider the case introduced in section 6.7.7 above. We assume a two dimensional world, with different outcomes of the weather and of the oil price. Let us also assume two persons, who have different revenues depending upon different outcomes of the weather and the oil price; and we assume that there are three classes for the weather and similarly for oil, so that we get 3 by 3 matrices. We collect all this information in a JointProspect object, that lists the revenue matrices and the probability matrix.

```
jp = JointProspect[{{100, -10, 5}, {1, 0, 0}, {-10, -10, 20}},
                  {{-10, -10, 1}, {0, 0, 100}, {100, 10, 0}},
                  {{2, 6, 6}, {6, 6, 12}, {8, 12, 12}}/70]
```

$$\text{JointProspect} \left( \begin{pmatrix} 100 & -10 & 5 \\ 1 & 0 & 0 \\ -10 & -10 & 20 \end{pmatrix}, \begin{pmatrix} -10 & -10 & 1 \\ 0 & 0 & 100 \\ 100 & 10 & 0 \end{pmatrix}, \begin{pmatrix} \frac{1}{35} & \frac{3}{35} & \frac{3}{35} \\ \frac{3}{35} & \frac{3}{35} & \frac{6}{35} \\ \frac{4}{35} & \frac{6}{35} & \frac{6}{35} \end{pmatrix} \right)$$

We can find the various prospects statistics by creating separate (full) prospects:

```
ps = JointToProspects[jp]
```

$$\left\{ \text{Prospect} \left( \begin{pmatrix} 100 & -10 & 5 \\ 1 & 0 & 0 \\ -10 & -10 & 20 \end{pmatrix}, \begin{pmatrix} \frac{1}{35} & \frac{3}{35} & \frac{3}{35} \\ \frac{3}{35} & \frac{3}{35} & \frac{6}{35} \\ \frac{4}{35} & \frac{6}{35} & \frac{6}{35} \end{pmatrix} \right), \text{Prospect} \left( \begin{pmatrix} -10 & -10 & 1 \\ 0 & 0 & 100 \\ 100 & 10 & 0 \end{pmatrix}, \begin{pmatrix} \frac{1}{35} & \frac{3}{35} & \frac{3}{35} \\ \frac{3}{35} & \frac{3}{35} & \frac{6}{35} \\ \frac{4}{35} & \frac{6}{35} & \frac{6}{35} \end{pmatrix} \right) \right\}$$

**ProspectStatistics[TableForm, N[ps]]**

	ProspectEV	Spread	Risk	RiskRatio	RiskPr
A	3.08571	19.5994	3.71429	0.546218	0.371429
B	29.2286	45.0721	1.14286	0.0376294	0.114286

An investor with a budget will allocate that budget over the various prospects, and will be interested in the optimal mix. Allocate share  $S$  to one prospect, and  $1 - S$  to the other.

**prs = JointToProspects[jp, {S, 1 - S}] // Simplify**

$$\text{Prospect} \left( \begin{pmatrix} 110S - 10 & -10 & 4S + 1 \\ S & 0 & -100(S - 1) \\ 100 - 110S & 10 - 20S & 20S \end{pmatrix}, \begin{pmatrix} \frac{1}{35} & \frac{3}{35} & \frac{3}{35} \\ \frac{3}{35} & \frac{3}{35} & \frac{6}{35} \\ \frac{4}{35} & \frac{6}{35} & \frac{6}{35} \end{pmatrix} \right)$$

The following shows only some of the statistics results. The key point is that these results are parametric functions of  $S$ , and that we can plot in the various spaces for  $S$  in the  $[0, 1]$  range (no borrowing).

**BinaryRiskSimplifyRule[expr, s]**

assumes a binary prospect with weight  $s$  in  $[0, 1]$ ,  
simplifies If[Negative...] using linearity, replaces by Max conditions, and Chops

Note: Simplify[Negative[20 S]] gives Negative[S], and may hinder IfNegativeToMaxRule at occasion

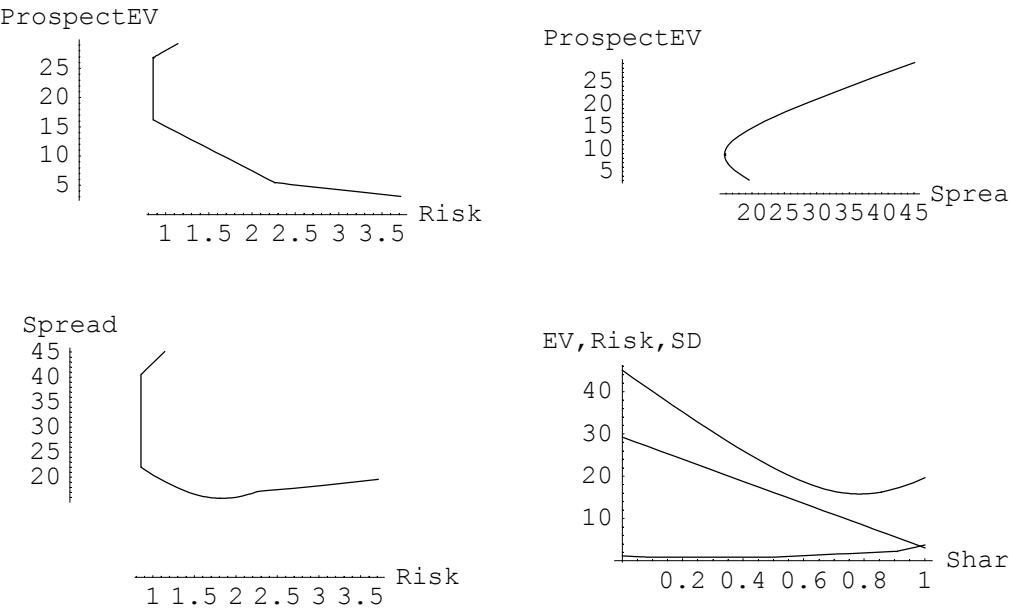
**BinaryRiskSimplify[**

**Take[Simplify[ProspectStatistics[prs]], 3], S]**

$$\left\{ \begin{aligned} \text{ProspectEV} &\rightarrow -\frac{3}{35} (305S - 341), \text{ Spread} \rightarrow \frac{1}{35} \sqrt{2} \sqrt{1775030S^2 - 2784035S + 1244288}, \\ \text{Risk} &\rightarrow \frac{1}{35} (-6 \text{ If}[\text{Negative}[1 - S], -100(S - 1), 0] - 6 \text{ If}[\text{Negative}[S], 20S, 0] + \\ &\quad \text{Max}(0, 10 - 110S) + 3 \text{ Max}(0, -S) + 6 \text{ Max}(0, 20S - 10) + 4 \text{ Max}(0, 110S - 100) + 30) \end{aligned} \right\}$$

- The following plots for S in [0,1]. The finance community is familiar with the upper right hand plot, the other plots are novel.

```
ProspectPlot[prs, S];
```



6.8.11 Continuous densities

The main routines discussed above can also handle continuous probability densities.

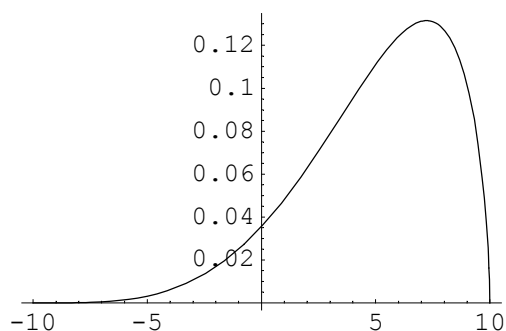
- Define a probability density.

```
Domain[pdf] = Interval[{min = -10, max = 10}];
```

```
pdf[x_] = ToBeta[PDF, 5, 10, {x, min, max}]
```

$$\frac{2079 \sqrt{\pi} \left(\frac{1}{2} - \frac{x}{20}\right)^{5/8} \left(\frac{x}{20} + \frac{1}{2}\right)^{31/8}}{256 \Gamma\left(\frac{13}{8}\right) \Gamma\left(\frac{39}{8}\right)}$$

```
Plot[pdf[x], {x, min, max}];
```



- The following commands presume absolute risk.

```
ToProspect[pdf]
```

```
Prospect(5.59606, -1.80958, 0.919513)
```

```
ToRisket[pdf]
```

```
Risket(5., 3.16228, 0.145648, 0.0804873)
```

- The following commands presume a target level, by default the expected value.

```
RelativeRisk[pdf]
```

```
4.08857
```

```
ConditionalRisk[pdf]
```

```
2.92301
```

- A check on consistency:

```
SetOptions[RiskModel, Equations → RelativeRiskEquations];
```

```
RiskModel[Results[RelativeRisk]]
```

```
{{4.08857 == 5. - Expectation(LessThanTarget), ConditionalRisk == 5. - Expectation(Conditional),  
  Expectation(Conditional) == 2.27882 Expectation(LessThanTarget)},  
 {Target → 5., RelativeRisk → 4.08857, RiskPr → 0.438823}, {{ConditionalRisk → 2.92301,  
  Expectation(Conditional) → 2.07699, Expectation(LessThanTarget) → 0.911434}}}
```

### 6.8.12 Symbols only

The following are only symbols.

RiskAversion	SpreadAversion	UncertaintyAversion
RiskPremium	SpreadPremium	UncertaintyPremium
Uncertainty		

6.8.13 Certainty equivalence

This section discusses how one could recover an ordinal utility scale from experiments with prospects. The current assumption is that there is no difference between the buying or selling price of a prospect, although in practice there can be this difference (since people tend to overcharge for what they have, even when they think that the grass of the neighbour is greener).

6.8.13.1 A standard approach

The idea is that you can choose between (1) a value  $C$  for certain or (2) a gamble between  $A$  ("above") and  $B$  ("below") with probability  $p$  for  $A$ . This means in fact that certainty equivalent  $C$  is already part of your wealth, and that there is an additional prospect of winning  $A - C$  or losing  $C - B$  (as absolute loss).

CertaintyEq[q_Prospect, c]	gives the condition so that $c$ is the certainty equivalent of the prospect
CertaintyEq[q_Prospect, c, None]	gives the same but excluding wealth
CertaintyEq[Prospect[A, B, p], C, ' ' Budget ' ']	
gives the budget for who reasons as follows: certain is $Wealth + C$ , and then there is the prospect of winning $A - C$ or losing (absolute) $C - B$ .	

```
cond = Prospect[A, B, p] ~CertaintyEq~ C

Utility(C + Wealth) == p Utility(A + Wealth) + (1 - p) Utility(B + Wealth)

CertaintyEq[Prospect[A, B, p], C, "Budget"]

C + Wealth + Prospect(A - C, B - C, p)
```

CertaintyEq[Equations]	gives the equations for the linear transform of utility
------------------------	---



- Note that the certainty equivalence condition is invariant for a linear transformation. This means that we can recover only ordinal utility.

```
cond /. Utility[x_] => a U[x] + b // FullSimplify
a (p U(A + Wealth) - (p - 1) U(B + Wealth) - U(C + Wealth)) == 0
```

- We can normalise our findings by setting the utility levels of the extremes of the prospect equal to those extremes.

```
CertaintyEq[Equations]
{c2 + c1 Utility(Min) == Min, c2 + c1 Utility(Max) == Max}
```

CertaintyEq[ Set, max, min, p_List]	sets the options for the maximum and minimum points, the probabilities p that are being considered, and the implied expected values
CertaintyEq[Data, CE_List]	sets CertaintyEq[Data] and creates in interpolation function for the certainty equivalent values CE
CertaintyEq[Data]	contains pairs {expected value, certainty equivalent}

Note: The data are recorded in Options[CertaintyEq].

- This example is taken from Luenberger (1998:236).

```
CertaintyEq[Set, 9, 1, Range[0, 10]/10.]
{1, 1.8, 2.6, 3.4, 4.2, 5., 5.8, 6.6, 7.4, 8.2, 9.}

func = CertaintyEq[Data,
  {1, 1.44, 1.96, 2.56, 3.24, 4, 4.84, 5.76, 6.76, 7.84, 9}]
InterpolatingFunction[({1. 9. }, <>)]

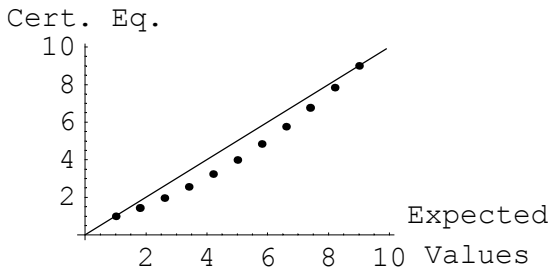
func[2]
2.65701
```

- It also appears possible to differentiate this interpolated function. The Arrow-Pratt measure for 'risk aversion' becomes:

```
ap[x_] = ArrowPratt[func[x], x];
```

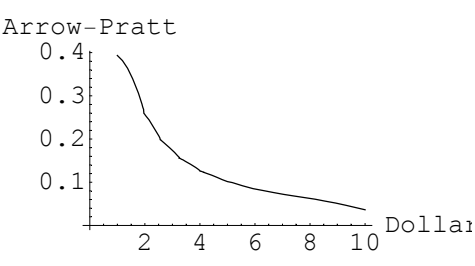
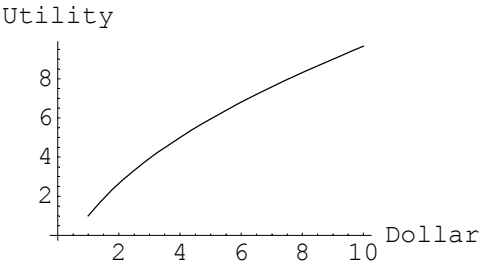
<code>CertaintyEq[ListPlot, opts]</code>	plots <code>CertaintyEq[Data]</code>
<code>CertaintyEq[Plot, opts]</code>	adds a diagonal line for reference

`CertaintyEq[Plot];`



■ The following plots the estimated utility function as well as the Arrow-Pratt measure.

```
Show[GraphicsArray[{
  Plot[func[x], {x, 1, 10}, AxesLabel -> {Dollar, Utility},
    DisplayFunction -> Identity],
  Plot[ap[x], {x, 1, 10}, AxesLabel -> {Dollar, "Arrow-Pratt"},
    DisplayFunction -> Identity]}],
  DisplayFunction -> $DisplayFunction];
```



6.8.13.2 A non-standard approach

The above (standard) approach can be criticised for using nonstochastic utility for a stochastic situation. The curvature of the utility function that applies for certain changes, now is applied to stochastic differences. The standard approach also does not explicitly state the risk which is exemplified by the budget line. An alternative (non-standard) approach is to regard a utility function that includes stochastic data and that uses that explicit risk.

- First recall the budget above.

```
CertaintyEq[Prospect[A, B, p], C, "Budget"]
```

$$C + \text{Wealth} + \text{Prospect}(A - C, B - C, p)$$

- Let us suppose that utility depends upon the wealth level and stochastic data. Note that the prospect normally has an expected value that differs from zero, and apparently the value of this is balanced by its risk. This means that the certainty equivalent condition above is mistaken, and we must replace it with the following.

```
cond2 = Utility[Wealth + C, 0, 0] ==
```

```
ProspectUtility[Prospect[A - C, B - C, p], Wealth + C &, ProspectEV, Risk]
```

$$\text{Utility}(C + \text{Wealth}, 0, 0) == \text{Utility}(C + \text{Wealth}, (B - C)(1 - p) + (A - C)p, -(B - C)(1 - p))$$

To tackle this, we need to have access to the prospects implied by this budget.

CertaintyEq[Set, Prospect]	creates the prospects implied by the options
----------------------------	--

CertaintyEq[Prospect, f]	applies function f to the prospects in the options
--------------------------	--

```
CertaintyEq[Set, Prospect]
```

```
{Prospect(8, 0, 0), Prospect(7.56, -0.44, 0.1), Prospect(7.04, -0.96, 0.2), Prospect(6.44, -1.56, 0.3),  
Prospect(5.76, -2.24, 0.4), Prospect(5, -3, 0.5), Prospect(4.16, -3.84, 0.6),  
Prospect(3.24, -4.76, 0.7), Prospect(2.24, -5.76, 0.8), Prospect(1.16, -6.84, 0.9), Prospect(0, -8, 1.)}
```

- The following determines the ProspectEV and Risk implied by the current Options[CertaintyEq], that are to be used for above ProspectUtility.

```
lisev = CertaintyEq[Prospect, ProspectEV] ;
```

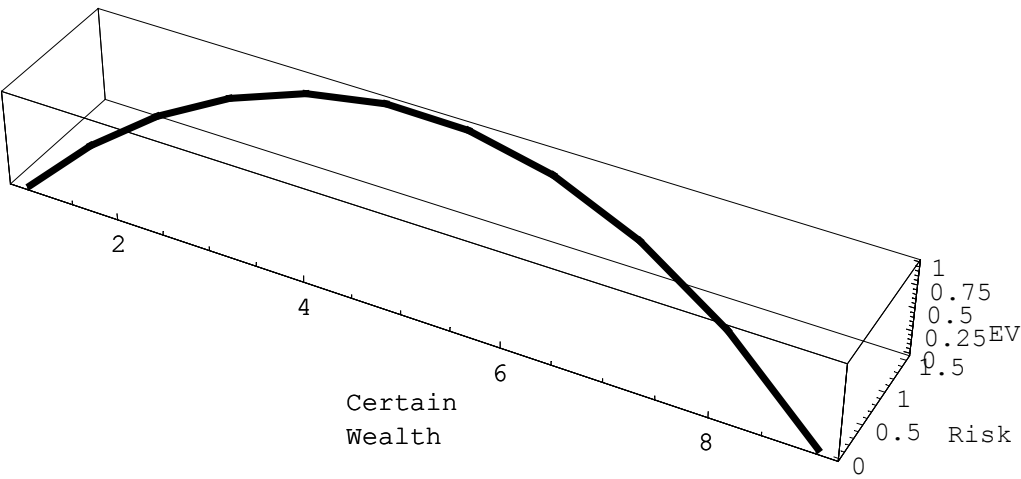
```
lisr = CertaintyEq[Prospect, Risk] ;
```

```
lisc = CertaintyEq /. Options[CertaintyEq] ;
```

- A higher risk needs compensation with higher expected addition to wealth.

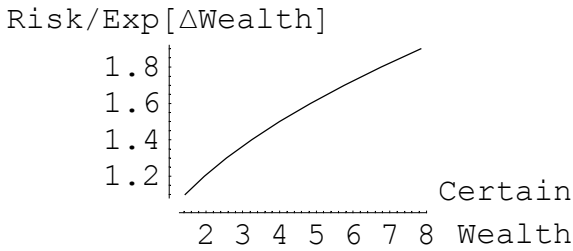
```
<< "Graphics`Graphics3D`"
```

```
ScatterPlot3D[Transpose[{lisc, lisr, lisev}],
  PlotJoined → True, PlotStyle → Thickness[0.008],
  AxesLabel → {"Certain\nWealth", Risk, "EV"}];
```



- ProspectEV is the expected  $\Delta$ Wealth forgone because of the risk in the prospect. With rising wealth, the experimental person is willing to accept more risk for the same amount of expected income =  $\text{Exp}[\Delta\text{Wealth}] = \text{ProspectEV}$ . The investor can use this relationship for additional decisions and predictions.

```
PlotLine[lisc, lisr/lisev, AxesLabel →
  {"Certain\n Wealth", "Risk/Exp[ΔWealth]"}, AxesOrigin → {1, 1}];
```



**6.8.13.3 A note on independence**

Above difference with the standard approach can also be clarified by reference to Mas-Colell c.s. (1995:171) and their theorem 6.B.4 on the independence axiom of preferences on prospects. The discussion on prospects there is a bit sterile, since it concentrates on the probabilities (or is in danger of confusion with that) while economically we are rather interested in the commodity space. The authors overstate it when they write that the independence theorem is "at the heart of the theory of choice under uncertainty". Similarly, the discussion there on p 179-180 on the Allais paradox leaves much to be said.

Let us regard three situations: Good, Normal and Bad. Use  $\succeq$  for the preference relation 'at least as good as'. The axiom states that the preference between prospects P and P' is independent of any third; or:  $P \succeq P' \Leftrightarrow \alpha P + (1 - \alpha) P'' \succeq \alpha P' + (1 - \alpha) P''$  for  $\alpha \in (0, 1)$ . This sounds seductively true if we look at probabilities only. But economically, we cannot neglect the wealth effect. Clearly my preference generally is Good  $\succeq$  Normal  $\succeq$  Bad, and when I am in a bad situation then clearly I am willing to gamble on getting better. But when I am in a Normal situation, then I will hesitate on a gamble with the Bad risk. The independence axiom would force me to gamble though !

In formula's: When at Bad, there can be an  $\alpha$  so that  $\alpha \text{ Good} + (1 - \alpha) \text{ Bad} \succeq \text{Normal}$ . Let us take a  $\beta$  and check independence from Normal itself (which the axiom allows). Then  $\beta (\alpha \text{ Good} + (1 - \alpha) \text{ Bad}) + (1 - \beta) \text{ Normal} \succeq \text{Normal}$ . But when I am at Normal, I may not wish to gamble when Bad is a possible outcome !

Note: The Allais paradox is included for completeness.

AllaisParadox[] contains the four prospects for the Allais paradox, discussed by Mas-Colell c.s. (1995), p179–180.

- People are offered two choices, one between 1 and 2, and one between 3 and 4. They tend to prefer  $1 \succeq 2$  and  $4 \succeq 3$ , though this violates the independence axiom. Below shows that their choice is not really irrational.

**alpar = AllaisParadox[]**

```
Prospect({2.5 × 106, 500000., 0}, {0, 1, 0}), Prospect({2.5 × 106, 500000., 0}, {0.1, 0.89, 0.01}),
Prospect({2.5 × 106, 500000., 0}, {0, 0.11, 0.89}), Prospect({2.5 × 106, 500000., 0}, {0.1, 0, 0.9})
```

- Properly seen, the choice between 1 and 2 gives a certain outcome with the first prospect. So the other prospect is one with risk.

**PutIn[alpar[[2]] - 500000]**

```
Prospect({2. × 106, 0., -500000}, {0.1, 0.89, 0.01})
```

**ToRisket[%]**

```
Risket(195000., 603718., 5000., 0.01)
```

- For the choice between 3 and 4, we can take a reference point in the certainty equivalence of the least attractive option. What this is, depends upon the agent. Let us here take the minimal expected value as the reference point. First determine the expected values.

**ProspectEV /@ alpar**

```
{500000., 695000., 55000., 250000.}
```

- It turns out that 3 has the minimum expected value. Taking this as the addition to wealth, we can determine the riskets, and find that option 4 clearly is better.

```
ToRisket /@ PutIn /@ (Take[alpar, -2] - 55000)
{Risket(0., 156445., 48950., 0.89), Risket(195000., 750000., 49500., 0.9)}
```

- Not evaluated:

```
Prospect3DPrTriangle[AllaisParadox[]]
```

6.8.14 Insurance

Insurance[Loss, Pr, Coverage, Premium, Wealth]

is an object for an insurance situation, when a Loss with probability Pr is insured, at the Coverage ≤ Loss at Premium, while the client has Wealth.  
Loss and Pr can be lists, and in that case a Min[Coverage, Loss] condition is to be used

Coverage                                      Symbol for the level of insurance coverage for a potential loss

CreateInsurance[n]                          creates an insurance object with loss lists of length n

CreateInsurance[L:Loss, pr:Pr, c:Coverage, p:Premium, w:Wealth]

creates the insurance object when insuring loss L that has probability pr, at the coverage c ≤ L at premium p, while the client's wealth is w.  
Loss L and probabilities pr can be lists, and in that case a Min[c,L] condition will be used

Note: This does not provide for interdependent losses and coverages yet.

- An Insurance is just an object.

```
ins = CreateInsurance[]
Insurance(Loss, Pr, Coverage, Premium, Wealth)
```

An insurance situation can be looked at from the viewpoint of the client or the firm.

ToClient[q_Insurance]	turns the object into a prospect for the client
ToFirm[q_Insurance]	turns the object into a prospect for the insurance firm

- The client's prospect is this:

```
ic = ToClient[ins]
```

```
Prospect(Wealth - CoveragePremium, -Loss + Coverage(1 - Premium) + Wealth, 1 - Pr)
```

- Note that the client's prospect can be simplified as follows.

```
TakeOut[ic, Profit]
```

```
-CoveragePremium + Wealth + Prospect(0, Coverage - Loss, 1 - Pr)
```

- The clients expected value and expected utility value are:

```
ProspectEV[ic] // Simplify
```

```
-Loss Pr + Coverage (Pr - Premium) + Wealth
```

```
ToExpectedUtility[ic]
```

```
Pr Utility(-Loss + Coverage(1 - Premium) + Wealth) + (1 - Pr) Utility(Wealth - CoveragePremium)
```

- The firm that sells the insurance has the following prospect and expected revenue.

```
ToFirm[ins]
```

```
Prospect(CoveragePremium, -Coverage(1 - Premium), 1 - Pr)
```

```
ProspectEV[%] // Simplify
```

```
Coverage (Premium - Pr)
```

- The following is an example when the loss can take different levels with different probabilities, and the firm's revenue thus is more complicated.

```
Simplify[ProspectEV[ToFirm[CreateInsurance[2]]]]
```

```
CoveragePremium - Min(Coverage, Loss(1)) Pr(1) - Min(Coverage, Loss(2)) Pr(2)
```

## 6.9 Statistical decision theory

---

### 6.9.1 Summary

This develops statistical decision theory by applying the results of section 5.9 to statistics.

`Economics[Decision]`

### 6.9.2 Introduction

The statistical decision problem is regarded as a game of the Statistician against Nature. The game is represented by a PayOff table, where Nature controls the rows (horizontal player), and all positive entries are earnings of Nature. These same PayOff's are losses to the Statistician, who controls the columns (vertical player).

NB. See section 5.9 for the terms 'loss', 'due' and 'risk'.

The state of nature is denoted in the literature by  $\theta$ , and here gets the structural symbol `State`. Actions are denoted by  $a$ , and here have structural symbol `Act`. The loss is a function of both, `Loss[ $\theta$ ,  $a$ ]`. Decision rules are denoted in the literature by  $d$ , and here by `Dec`.

- Let us first reset the `Decision`` package, and then choose traditional symbols:

```
Reset[]
```

```
State =  $\theta$ ; Act =  $a$ ; Dec =  $d$ ;
```

- If we now look at an arbitrary problem:

```
LookAt[Loss, {NumberOfStates  $\rightarrow$  2, NumberOfActions  $\rightarrow$  3}]
```

	$a(1)$	$a(2)$	$a(3)$
$\theta(1)$	Loss(1, 1)	Loss(1, 2)	Loss(1, 3)
$\theta(2)$	Loss(2, 1)	Loss(2, 2)	Loss(2, 3)

By taking a sample  $X$  we can probe into the state of nature. `Est[X]` or  $\hat{\theta}$  will be the estimate of  $\theta$ . The estimation has a probability of success, which rate however may depend upon the state of nature too. We have, specifically:

- Probability measure  $\Pr[X \mid \theta] = \Pr[\text{Est}[X] \mid \theta]$ , here denoted as  $\Pr[\theta \rightarrow \text{Est}]$
- Decision rules  $d[\text{Est} \rightarrow a]$

Since the  $\Pr[X \mid \theta]$  are dependent upon the states of nature, we find

- intermediate step  $\Pr[\text{Est} \mid \theta] d[\text{Est} \rightarrow a]$



- $\text{Due}[d] = \text{Expected generalised loss} = \text{Sum}[\text{Pr}[\text{Est} | \theta] \text{ Loss}[\theta, a]]$

We may minimize the due, subject to the decision rules. Note that this 'due concept' does not consider the effect of the variance.

### 6.9.3 Example 1

Let us define a small example decision making problem, with the following pay off table.

**Example1 = PayOffToRule[{{1, 0, 2},{0,-1, -1 }}]**

$\left\{ \text{NumberOfStates} \rightarrow 2, \text{NumberOfActions} \rightarrow 3, \text{PayOff} \rightarrow \begin{pmatrix} 1 & 0 & 2 \\ 0 & -1 & -1 \end{pmatrix} \right\}$

With reference to section 5.9 above, we are tempted to go through the different steps, such as assigning a matrix of conditional probabilities and determining the convex hull:

- **Assign[All, Example1, ProbabilityMatrix  $\rightarrow \{\{.75, .25\}, \{.05, .95\}\}$ ];**
- **GamePlot[Hull, DT[Example1]]**

However, we better test whether the problem isn't dominated in the first place. This appears to be the case, and this saves us a lot of work.

**LE[Transpose, PayOff /. Example1]**  
(0 -1)

### 6.9.4 Type I and type II errors

A typical 2 x 2 problem uses type I and type II errors.

- Let Act[1] be best with a loss of zero if  $\theta > \theta_0$ , so that  $\text{Loss}[\theta > \theta_0, \text{Act}[1]] = 0$ , and let Act[1] for  $\theta \leq \theta_0$  have loss  $L[1] = \text{Loss}[\theta \leq \theta_0, \text{Act}[1]]$ .
- Also, Act[2] would be best with a loss of zero if  $\theta \leq \theta_0$ , while Act[2] would have a loss of  $L[2]$  for other values.
- The decision rule is to take Act[1] if it seems that  $\theta > \theta_0$ , and take Act[2] otherwise.

The area  $\theta \leq \theta_0$  is called the test region or the critical region, and  $\text{Pw}[\theta, \theta_0] = \text{Pr}[\theta \leq \theta_0 | \theta]$  is the power function of the test  $\theta > \theta_0$ . One often uses abstract  $\omega$  for the test region.

The expected loss function for the current special problem is (try  $\theta > \theta_0$  and  $\theta \leq \theta_0$ ):

$$\text{Due}[\theta, \theta_0] = (1 - \text{Pr}[\theta \leq \theta_0 | \theta]) \text{Loss}[\theta, \text{Act}[1]] + \text{Pr}[\theta \leq \theta_0 | \theta] \text{Loss}[\theta, \text{Act}[2]]$$

Using  $\text{Pw}[\theta, \theta_0]$  and writing i for Act[i]:

$$\text{Due}[\theta, \theta_0] = \text{Loss}[\theta, 1] + \text{Pw}[\theta, \theta_0] \{\text{Loss}[\theta, 2] - \text{Loss}[\theta, 1]\}$$

Let us assume that our basic hypothesis  $H_0 = \theta > \theta(0)$  and  $\text{Act}[1]$ . Then the error of type I is to reject the hypothesis while it is true, and the error of type II is to accept the hypothesis while it is false. For the probabilities of these errors:

- $\alpha = \text{Pr}[\text{Error type I}] = \text{Pr}[\text{Act}[2] \mid \theta > \theta(0)]$  is the significance level or size of the test
- $\beta = \text{Pr}[\text{Error type II}] = \text{Pr}[\text{Act}[1] \mid \theta \leq \theta(0)]$  with  $1 - \beta$  the power of the test
- We Reset, and adapt the display of the loss and due tables (with a rule and not Set, since subfunction ToLoss needs to recognise  $\text{State}[]$ ).

```
Reset[]
typr = PayOffToRule[ {{0, L[1]}, {L[2], 0}} ];
Assign[Loss, typr];
disp = {State[1] → "θ > θ(0)", State[2] → "θ ≤ θ(0)"};

LookAt[Loss, typr] /. disp
```

	Act(1)	Act(2)
$\theta > \theta(0)$	0	$L(1)$
$\theta \leq \theta(0)$	$L(2)$	0

- Assigning these probabilities to the proper places:

```
Assign[All, typr, ProbabilityMatrix → {{1 - α, α}, {β, 1 - β}}];

LookAt[Due, typr] /. disp // Simplify
```

	Dec(1)	Dec(2)	Dec(3)	Dec(4)
$\theta > \theta(0)$	0	$\alpha L(1)$	$L(1) - \alpha L(1)$	$L(1)$
$\theta \leq \theta(0)$	$L(2)$	$\beta L(2)$	$L(2) - \beta L(2)$	0

One may now apply various decision criteria to this table, such as minimax due etcetera. Since  $\alpha$  and  $\beta$  generally are small, we already can conclude that Dec[2] dominates Dec[3]. With  $p$  the probability of  $\text{State}[1]$ , we can also find a range for it if we want Dec[2] to be superior. The total due of Dec[1], Dec[2] and Dec[4] would be  $(1 - p) L[2]$ ,  $p \alpha L(1) + (1 - p) \beta L(2)$  and  $p L[1]$  respectively. Dec[2] is only superior in total due, if  $p$  is in the range:

$$p \alpha L(1) + (1 - p) \beta L(2) < (1 - p) L[2] \Rightarrow p < (1 - \beta) L(2) / \{\alpha L(1) + (1 - \beta) L(2)\}$$

$$p \alpha L(1) + (1 - p) \beta L(2) < p L[1] \Rightarrow p > \beta L(2) / \{(1 - \alpha) L(1) + \beta L(2)\}$$

A general point is that the decision must be made with the loss function taken in consideration. Thus: ask for the loss function, and don't simply take a significance level of 5%. Also, the accuracy of measurement of  $\alpha$  and  $\beta$  will come with costs, and these costs could be included into the decision.

- PM. The following subroutine quickly generates above probability matrix.

**PowerOfTheTest**[ $\alpha$ ,  $\beta$ ]

$$\begin{pmatrix} 1 - \alpha & \alpha \\ \beta & 1 - \beta \end{pmatrix}$$

### 6.9.5 Technical subroutines

These subroutines are of a technical nature. Basically one would use the Assign[All, ...] feature. But some might like to understand more about how the routines work.

<b>DefDec</b> [ <i>payoff object</i> ]	constructs the various decision rules
<b>ProbMat</b> [ <i>payoff object</i> ]	constructs the probability matrix Pr[State→ Est]. Since states of nature are estimated, this matrix is always square

**Reset**[]

- DefDec only constructs the decision rules. Use Assign to fill Dec[ ].

**DefDec**[**OddOrEven**]

$$\begin{pmatrix} \{\text{Est}(1) \rightarrow \text{Act}(1), \text{Est}(2) \rightarrow \text{Act}(1)\} & \{\text{Est}(1) \rightarrow \text{Act}(1), \text{Est}(2) \rightarrow \text{Act}(2)\} \\ \{\text{Est}(1) \rightarrow \text{Act}(2), \text{Est}(2) \rightarrow \text{Act}(1)\} & \{\text{Est}(1) \rightarrow \text{Act}(2), \text{Est}(2) \rightarrow \text{Act}(2)\} \end{pmatrix}$$

**Assign**[**Dec**, **OddOrEven**]

**??Dec**

Head of decisions

**Dec**(1) = {**Est**(1) → **Act**(1), **Est**(2) → **Act**(1)}

**Dec**(2) = {**Est**(1) → **Act**(1), **Est**(2) → **Act**(2)}

**Dec**(3) = {**Est**(1) → **Act**(2), **Est**(2) → **Act**(1)}

**Dec**(4) = {**Est**(1) → **Act**(2), **Est**(2) → **Act**(2)}

- ProbMat only constructs the matrix. Use Assign to fill Pr[ ].

**ProbMat**[**OddOrEven**]

$$\begin{pmatrix} \text{Pr}(\text{State}(1) \rightarrow \text{Est}(1)) & \text{Pr}(\text{State}(1) \rightarrow \text{Est}(2)) \\ \text{Pr}(\text{State}(2) \rightarrow \text{Est}(1)) & \text{Pr}(\text{State}(2) \rightarrow \text{Est}(2)) \end{pmatrix}$$

**Assign**[**Pr**, **OddOrEven**, **ProbabilityMatrix** → {{.70, .30}, {.25, .75}}]

??Pr

```
Attributes[Pr] = {Orderless}

Pr(State(1) → Est(1)) = 0.7

Pr(State(1) → Est(2)) = 0.3

Pr(State(2) → Est(1)) = 0.25

Pr(State(2) → Est(2)) = 0.75
```

DueMat [ <i>payoff object</i> ]	determines the due matrix
ToLoss	a replacement rule for the loss related to a decision

Note: ToLoss works only for unassigned Pr. It is easiest to use the Assign[All, ...] command, see section 5.9.

- Note: The allocation of Loss[x, y] values to the due matrix works only for unassigned probability. So in this case we have to clear Pr again. Just to display, we flatten.

```
Clear[Pr]; DueMat[OddOrEven] /. ToLoss // Flatten
```

```
{Loss(1, 1) Pr(State(1) → Est(1)) + Loss(1, 1) Pr(State(1) → Est(2)),
 Loss(2, 1) Pr(State(2) → Est(1)) + Loss(2, 1) Pr(State(2) → Est(2)),
 Loss(1, 1) Pr(State(1) → Est(1)) + Loss(1, 2) Pr(State(1) → Est(2)),
 Loss(2, 1) Pr(State(2) → Est(1)) + Loss(2, 2) Pr(State(2) → Est(2)),
 Loss(1, 2) Pr(State(1) → Est(1)) + Loss(1, 1) Pr(State(1) → Est(2)),
 Loss(2, 2) Pr(State(2) → Est(1)) + Loss(2, 1) Pr(State(2) → Est(2)),
 Loss(1, 2) Pr(State(1) → Est(1)) + Loss(1, 2) Pr(State(1) → Est(2)),
 Loss(2, 2) Pr(State(2) → Est(1)) + Loss(2, 2) Pr(State(2) → Est(2))}
```

- Use Assign[Due, ... ] to create the actual due table. Since we have not assigned the Loss yet, we get the abstract formulation. In this case, to get all on one page, we transpose the result.

```
Assign[Due, OddOrEven,
  ProbabilityMatrix → PowerOfTheTest[α, β]] // Simplify // Transpose
```

$$\begin{pmatrix} \text{Loss}(1, 1) & \text{Loss}(2, 1) \\ \alpha \text{Loss}(1, 2) - (\alpha - 1) \text{Loss}(1, 1) & \beta (\text{Loss}(2, 1) - \text{Loss}(2, 2)) + \text{Loss}(2, 2) \\ \alpha (\text{Loss}(1, 1) - \text{Loss}(1, 2)) + \text{Loss}(1, 2) & \beta \text{Loss}(2, 2) - (\beta - 1) \text{Loss}(2, 1) \\ \text{Loss}(1, 2) & \text{Loss}(2, 2) \end{pmatrix}$$

# 6.10 Sampling

## 6.10.1 Summary

This package implements some standard cases of sampling theory.

```
Economics[Sampling]
```

## 6.10.2 Introduction

The `Decision`` package discusses the theory of statistical decision making. There are some standard cases of hypothesis testing that may as well be put in a separate `Sampling`` package. The general format is `Case[i, ...]` while `Case[Plot, i, ...]` plots.

<code>Case[i, ...]</code>	the sample size rule for case $i$ , with $\alpha$ and $\beta$ the probabilities of error type I and II
<code>SampleSize</code>	symbol, in the text symbol $n$
<code>CriticalValue</code>	symbol, in the text symbol $v$
<code>Number</code>	added usage: the critical number, in the text $c = v n$

Generally, the input distributions concern the separate observations  $x[k]$  and the critical value is for the  $mean = \text{Sum}[x[k]] / n$ . The variance of the mean depends upon the sample size.

## 6.10.3 Approximating the binomial with a normal distribution

If we take a sample of size  $n$ , and each draw has a probability of success  $p$ , then the sample is described by a binomial distribution. For larger  $n$ , we can approximate the distribution by a normal distribution. This holds especially for the distribution of the mean.

<code>BiNoPars[p, n : l]</code>	gives {p, Sqrt[p (1 - p) / n]}
<code>BiNoProxy[p, n]</code>	= NormalDistribution[p, Sqrt[p (1-p)/n]]

## 6.10.4 Operating characteristic curve

The operating characteristic curve (OC curve) defined here gives the probability of acceptance as a function of the population parameter. Note that the literature also has OC curves that show the probability of rejection - that then have another shape.

<code>OCCurve[p, {n, c}]</code>	gives <code>CDF[BinomialDistribution[p, n], c]</code> for the proportion of success p and for given n and c
<code>OCCurve[p, {n, c}, PoissonDistribution]</code>	
	approximates with <code>CDF[PoissonDistribution[p * n], c]</code> , for $n > 20$ and $p < .05$
<code>OCCurve[p, {n, c}, NormalDistribution]</code>	
	approximates with <code>CDF[NormalDistribution[p, Sqrt[p (1-p)/n], c/n]</code>
<code>OCCurveInverse[y_?NumberQ, {{n, c} (, ____)]</code>	
	gives the probability p that makes the probability of acceptance equal to y

Note: For continuous cases, use `CDF[NormalDistribution[mu, sigma], decisionvalue]` as a function of mu.

6.10.5 Case 1: Accuracy with confidence for a normal distribution

We can impose that the mean has some accuracy, and with a certain confidence level. The confidence level is the integral over the range determined by the accuracy.

<code>Case[1, <math>\pi</math>, r, {<math>\mu</math>, <math>\sigma</math>}]</code>	gives the sample size for a normal distribution, for relative accuracy r such that, $\Pr[\mu (1-r) \leq \text{mean} \leq \mu (1+r)] = \pi$ confidence level
--	--

Since the sample size n can be solved from the confidence level for the sample mean, it remains to specify the mean and sigma for the individual data. These can be estimated from the proportion of successes or from looking at the coefficient of variation.

- With 99.7 % confidence for various accuracies r:

```
Case[1, .997 , r, {p, Sqrt[p (1 - p)]}]
```

$$\left\{ \text{SampleSize} \rightarrow \frac{8.80747 (1 - p)}{p r^2} \right\}$$

- With the coefficient of variation = sigma / mu, then the population mean cancels from the formula. With a sigma = 0.8 mu and with  $\pi = 95\%$  and  $r = 5\%$ :

```
Case[1, .95 , .05, {mu, .8 mu}]
```

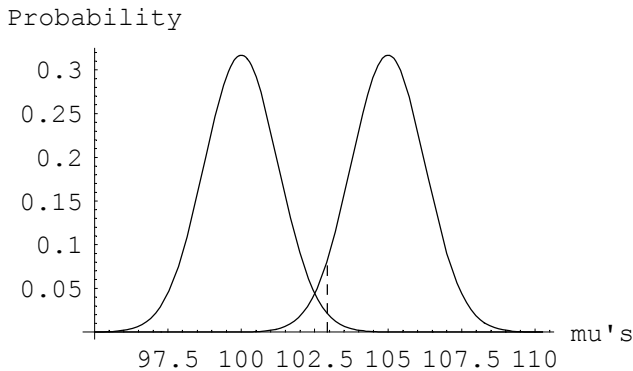
$$\{ \text{SampleSize} \rightarrow 983.413 \}$$

6.10.6 Case 2: Point hypotheses for a normal distribution

This is the classic case of two hypotheses about a normal distribution. Hypothesis 0 concerns the parameter point  $\{\mu_0, \sigma_0\}$ , the alternative is the point  $\{\mu_1, \sigma_1\}$ ,  $\alpha$  is the probability of error type I of rejecting  $H_0$  when it is valid, and  $\beta$  is the probability of error type II of accepting  $H_0$  when it is invalid.

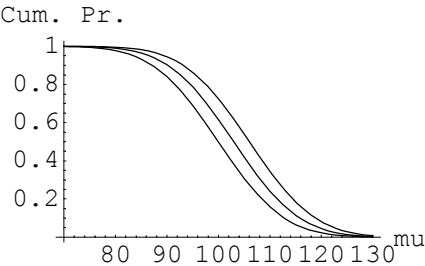
```
Case[2,  $\alpha$ ,  $\beta$ ,  $\{\mu_0, \sigma_0\}$ ,  $\{\mu_1, \sigma_1\}$ ]      for two normal distributions
```

```
Case[2, .01, .05, {100,10}, {105, 10}]  
  
{SampleSize → 63.0818, CriticalValue → 102.929}  
  
Case[Plot, 2, .01, .05, {100,10}, {105, 10}, TextStyle → {FontSize → 10}]
```



Some operating characteristic curves for this case are:

```
Plot[{ CDF[NormalDistribution[mu, 10], 100],  
       CDF[NormalDistribution[mu, 10], 103],  
       CDF[NormalDistribution[mu, 10], 106]},  
      {mu, 70, 130}, AxesLabel → {"mu", "Cum. Pr."}]
```



6.10.7 Case 3: Point hypotheses for a binomial, using the normal distribution

```
Case[3,  $\alpha$ ,  $\beta$ ,  $p_0$ ,  $p_1$ ]      approximates binomials by normal distributions
```

Let us sample an urn with white and black balls, with  $p$  = chance on white. This  $p$  may be .4 or .5. We want to know how many times we must draw to decide on  $p$ , and what the critical value is for the number of white balls found. Take  $p = .5$  as  $H_0$ , and  $\alpha = .05$  and  $\beta = .10$

```
c3 = Case[3, .05, .10, .5, .4]

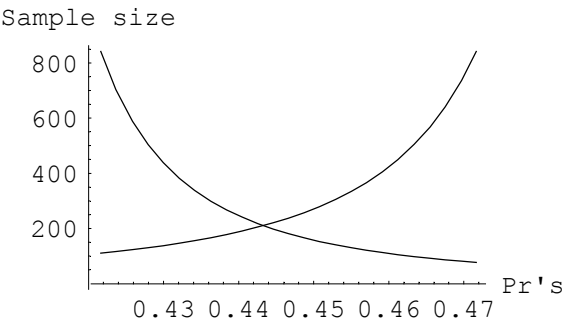
{SampleSize → 210.324, CriticalValue → 0.443291, Number → 93.2349}

c3 = Case[3, .05, .10, .5, .4]

{SampleSize → 210.324, CriticalValue → 0.443291, Number → 93.2349}
```

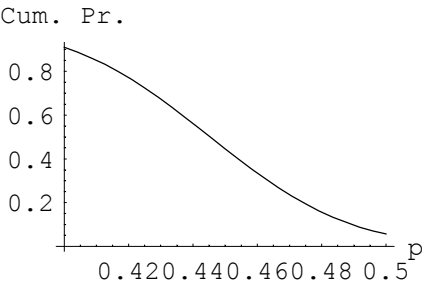
- Since  $1 - \alpha$  is the probability of accepting  $p_0$  when it is true, and since we take a one-sided approach, we find  $1 - \alpha = \text{CDF}[\text{BiNoProxy}[p_0, n], p]$ . This implies a relationship  $n[p]$  or  $p[n]$ . Similarly for  $\beta$  and  $p_1$ . The following plot gives these implied sample sizes as a function of  $p$ . The intersection gives the sample size and  $p$ -value that agree with both assumptions.

```
Case[Plot, 3, .05, .10, .5, .4, TextStyle → {FontSize → 10}]
```



- The OC curve here is:

```
Plot[OCCurve[p, {210, 93}], {p, .4, .5}, TextStyle → {FontSize → 10},
AxesLabel → {"p", "Cum. Pr."}]
```





- Since we solved the problem with the normal approximation, the sample size and critical number are not integers. We should use the proper OC curve then.

```
OCCurve[CriticalValue, {SampleSize, Number}, NormalDistribution] /. c3
0.5
```

### 6.10.8 Apart from these cases: using the Binomial directly

Given current computing power we may as well use the Binomial distribution itself when appropriate. The following routine gives sample results for a sample size of  $n$  with  $c$  successes, where one may substitute  $c = p n$  if  $p$  is known. Output gives results for the Bernoulli distribution with  $p$ , for the sum of successes  $c = \sum x$  and for the  $mean = c / n = \sum x / n$ . Input parameter  $r$  gives the accuracy of  $c$ , with the Spread giving the range of  $(1-r) c$  till  $(1+r) c$ . This Spread also comes with the cumulative probabilities  $\{\pi_1 = \Pr[0 \leq y \leq (1-r) c], 1 - \pi_1 - \pi_2 = \Pr[(1-r) c \leq y \leq (1+r) c], \pi_2 = \Pr[(1+r) c \leq y \leq n]\}$ .

```
BinomialSample[n, c, r:0.2, opts]
```

gives the sample results for a sample size of  $n$  and  $c$  successes

The StudentTCI gives the confidence interval for the mean, assuming the normal approximation; opts can include the ConfidenceLevel option.

```
BinomialSample[100, 20] // N

{Proportion → {Mean → 0.2, Variance → 0.16, StandardDeviation → 0.4},
 Sum → {Mean → 20., Variance → 16., StandardDeviation → 4.},
 Mean → {Mean → 0.2, Variance → 0.0016, StandardDeviation → 0.04,
 StudentTCI → {0.120621, 0.279379}, Spread[0.2, {16., 24.}] → {0.192338, 0.676309, 0.131353}}}
```

### 6.10.10 Sequential observations and control charts

```
ControlChart[data_List, LCL, mean, UCL, opts]
```

plots the control chart with lower control limit LCL and upper control limit UCL. The chart is centered at the mean. Options are passed on to Show[]

The RangeLists option in output divides the data in  $\{x \leq LCL, LCL < x \leq \text{mean}, \text{mean} < x \leq UCL, UCL < x\}$ .

Control charts concern sequential samples. Typically each sample gives an *average value* and a *range* of the values in the sample. Quality criteria result into values for the lower control limit LCL, total mean and the upper control limit UCL, for both the sample averages and the ranges. How these LCL and UCL are determined is not discussed here. A process is in control when both statistics, average and range, remain between their boundaries and don't show a tendency to get out of range. A typical report contains a plot for the averages and a plot for the ranges.

- Let us assume that we have five samples, with give the following averages and ranges.

```
averages = {44, 40, 46, 45, 48};
```

```
ranges = {14, 10, 12, 8, 15};
```

- The following is a control chart for the averages.

```
pav = ControlChart[averages, 39.77, 43.5, 47.23, TextStyle → {FontSize  
→ 10}, DisplayFunction → Identity]
```

```
{RangeLists → {{}, {40}, {44, 46, 45}, {48}},  
Too Low → {}, Too High → {48}, In Control → False}
```

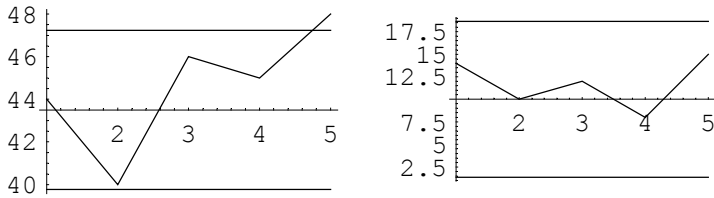
- The following is a control chart for the ranges.

```
pran = ControlChart[ranges, 1.36, 10, 18.64, TextStyle → {FontSize →  
10}, DisplayFunction → Identity]
```

```
{RangeLists → {{}, {8}, {14, 10, 12, 15}, {}},  
Too Low → {}, Too High → {}, In Control → True}
```

- Taking the charts together:

```
Show[GraphicsArray[{pav, pran}], DisplayFunction → $DisplayFunction]
```



# 6.11 The $\chi^2$ test

## 6.11.1 Summary

This package implements the  $\chi^2$  test for tabular data, using the Pearson and Likelihood Ratio test statistics.

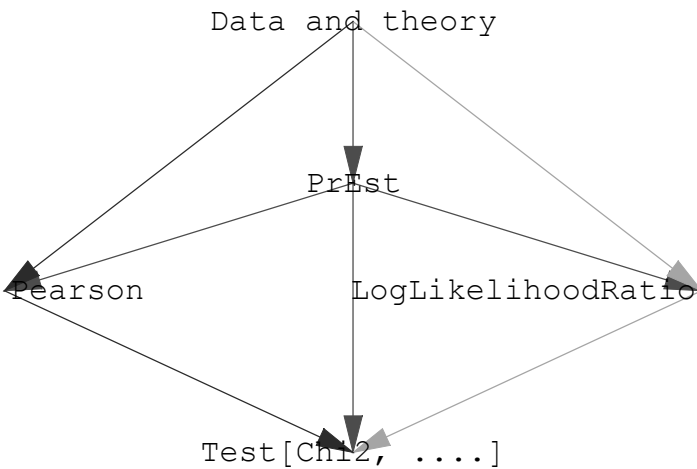
**Economics [Chi2]**

## 6.11.2 Introduction

The procedure is generally like this:

1. The user provides a null-hypothesis on a theoretical distribution.
  - (-) This may be based on later information, but at the cost of degrees of freedom.
  - (-) A common hypothesis is that of independence. We will use the symbol ID to indicate this assumption of independence. Notice though that this is not the only possible assumption.
2. Sample data are collected.
3. Hypothesis and data are translated into a test statistic.
4. The user provides a significance level for the test, derived from cost and benefit considerations.
5. The test statistic is subjected to a  $\chi^2$  test on that significance level.

This procedure is depicted in the following diagram. The test statistics considered here are the Pearson ratio and the LogLikelihoodRatio. The PrEst routine provides probabilities to the test statistics, and provides degrees of freedom to the actual test routine Test[Chi2, ...].



6.11.3 Short example

The preferred routine is `Test[ID, x_List, opts]` that applies the likelihood ratio test on the assumption of independence. The null hypothesis is independence, and possible decisions are `Accept` and `Reject`. One may set `Accept = FailToReject`.

<code>Test[ID, data, opts]</code>	computes the theoretical distribution under independence, determines $-2 \text{Log}[\text{LikelihoodRatio}]$ , and applies the Chi2 test
<code>Test[Chi2, {rules}, opts]</code>	applies the Chi2 test on the the Test and TestStatistic with SignificanceLevel

Note: Defaults are in `Options[Chi2]`. The major controls are `Partition` and `SignificanceLevel`, while `Show` and `RealQ` control output formats.

- This generates some superfluous statistics since the dimension of the problem is 1. Rather than defining a separate routine, it has seemed better to use the most general description.

```
Test[ID, {1, 2, 3, 34}]

Chi2::prfi: Input length 1: no useful border
totals. Equal probability is attached to all categories.

Observed

1      2      3      34

Theoretical

0.25    0.25    0.25    0.25

Log[(t / t(est))^ni]

2.30259      3.21888      3.61192      -41.6084

{BorderSums -> ( 1  2  3  34 ), Chi2PValue -> 5.14033 x 10^-14, DegreesOfFreedom -> 3,
Dimensions -> {4}, Do -> Reject, MarginalPr -> See ProbabilityMatrix,
MLEstimate -> {0.025, 0.05, 0.075, 0.85}, NumberOfObservations -> 40,
Partition -> {1}, ProbabilityMatrix -> {0.25, 0.25, 0.25, 0.25}, SignificanceLevel -> 0.05,
TensorRank -> 1, Test -> LogLikelihoodRatio, TestStatistic -> 64.95}
```

6.11.4 Chi2 and Chi2PValue

*Mathematica* already gives the `CDF[ChiSquareDistribution[df], x]`. The `Chi2` function here is a just a useful shorter format. The `Chi2PValue` has the same formats as `Chi2`, but more content.

<code>Chi2[x_Real, df_Integer]</code>	<code>CDF[ChiSquareDistribution[df], x]</code>
<code>Chi2[x, DegreesOfFreedom→df]</code>	<code>CDF[ChiSquareDistribution[df], x]</code>
<code>Chi2[x]</code>	<code>CDF[ChiSquareDistribution[df], x]</code> , with default df from <code>Options[Chi2]</code>
<code>Chi2PValue[x__]</code>	same input formats as <code>Chi2</code> , gives a rule with the cumulated probability beyond x, <code>Chi2PValue</code> → <code>1 - Chi2[x,df]</code> .

Note: The `OneSidedPValue` in the *Mathematica* statistical packages switches meaning for below 50% and above 50% probability. For example, do the following `Plot[Last[ChiSquarePValue[x, 3]], {x, .001, 10}]`.

- Comparing *Mathematica* and our result: This is the same:

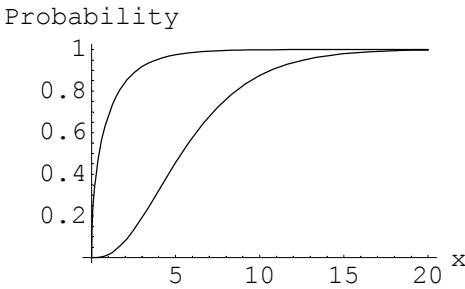
```
{ChiSquarePValue[10, 6], Chi2PValue[10., 6]}  
  
{OneSidedPValue → 0.124652, Chi2PValue → 0.124652}
```

- This however differs because of the *Mathematica* switch of perspective.

```
{ChiSquarePValue[5, 6], Chi2PValue[5., 6]}  
  
{OneSidedPValue → 0.456187, Chi2PValue → 0.543813}
```

- The following plots the  $\chi^2$  for 1 and 6 degrees of freedom.

```
Plot[{Chi2[x, 1], Chi2[x, 6]}, {x, 0, 20}, AxesOrigin → {0, 0},  
AxesLabel → {"x", "Probability"}]
```



<code>Chi2CriticalValue[y]</code>	for <code>y = Sequence[df], [df, signif],</code> or <code>[opts]</code> , computes the Chi2 critical value for the pair <code>{df, signif}</code> , i.e the value of x with cumulated probability up to <code>(1 - signif)</code>
<code>InverseChi2[df, p]</code>	gives the inverse $\text{Chi2}^{-1}[\text{df}, p]$ for given df and cumulated probability p up to x

```
Chi2CriticalValue[6, .05]
```

```
12.5916
```

```
InverseChi2[6, .95]
```

```
12.5916
```

6.11.5 Example problem with three dimensions

6.11.5.1 The problem

H. Rijken van Olst, "Inleiding tot de statistiek", Van Gorcum 1969, p137-139, contains the example of a city police arresting drunks at various days in the week. For a certain week the number of arrests is 70, and under the hypothesis that the day of the week has no influence on the number of arrests of drunks, the theoretical frequency of each day is 10 arrests.

```
days = {Sun, Mon, Tue, Wed, Thu, Fri, Sat};
obsFreq = arresteddrunks = {5, 12, 16, 7, 10, 15, 5};
theoFreq = Table[10, {7}];
```

An alternative theory is that market days are important.

```
marketdays = {no, no, yes, no, no, yes, no};

data = {days, marketdays, obsFreq, theoFreq};
TableForm[data, TableHeadings ->
  {"days", "market", "arrests", "theory 1"}]
```

days	Sun	Mon	Tue	Wed	Thu	Fri	Sat
market	no	no	yes	no	no	yes	no
arrests	5	12	16	7	10	15	5
theory 1	10	10	10	10	10	10	10

To oppose market days to normal days, we need to aggregate the data.

- This trick does the job.

```
market = data . marketdays /. {yes -> 1, no -> 0};
normal = data . marketdays /. {yes -> 0, no -> 1};

TableForm[Transpose[{market, normal}],
TableHeadings -> {"days", "market", "arrests", "theory 2"}]
```

days	Fri + Tue	Mon + Sat + Sun + Thu + Wed
market	2	5
arrests	31	39
theory 2	20	50

- Retrieve the data, and assign these to separate variables.

```
obsMarket = Transpose[{market, normal}][[3]]
theoMarket = Transpose[{market, normal}][[4]]
```

```
{31, 39}
```

```
{20, 50}
```

The  $\chi^2$  test is sensitive to the number of observations per degree of freedom. In the second case, there are more observations per degree of freedom, and thus the test is stronger. (In the same manner, by increasing the sample size, acceptance tends to turn into rejection, and by reducing the size, rejection can turn into acceptance.)

### 6.11.5.2 Executive summary

Since there will be more tests below, the reader might lose track of what is happening, and it is useful to first give a summary of what we are doing and what our conclusions are.

There appear to be four hypotheses:

- "normal days matter" versus "normal days don't matter"
- "market days matter" versus "market days don't matter"

We take the "don't matter" 's as Null Hypotheses, since these reflect *independence*. When independence is rejected, the difficult question arises how things then actually depend - but that is of no concern here. (In a subsequent research project we might check on the actual number of drunks, not only the arrested ones, to see if the police has a different policy each day on making arrests.)

The results of our testing H0 at 5% significance are:

- If 'independence' means that each day has a 1/7th chance, then the kind of day:
  - (a1) has no influence under the Pearson test,
  - (a2) has an influence however according to the likelihood ratio test.
- If 'independence' refers to the distinction between "market days" and "non market days" then there is a significant dependence under both tests.

Our conclusions are:

- It matters what we call "kind of day" (how we aggregate date).
- It matters what test we do and what level of significance we take.

In the following discussion we will store the outcome of the tests as result[hypothesis, kind of test] so that we later can make a summary table.

6.11.5.3 Degrees of freedom and Plot of the  $\chi^2$

The first problem has  $6 = 7 - 1$  degrees of freedom, the second only  $1 = 2 - 1$ . The plot of the  $\chi^2$  functions above actually serves this situation.

6.11.5.4 The Pearson test statistic

The Pearson test statistic is  $(o - t)^2 / t$ , with  $o$  the observed and  $t$  the theoretical frequency.

<code>Pearson[o, t, opts]</code>	computes <code>Sum[(o - t)^2/t]</code> with <code>o</code> the observations and <code>t</code> the theoretical values. Inputs <code>o</code> and <code>t</code> may be multidimensional as long as they have the same shape
<code>Pearson[Table]</code>	contains the test cells after running the routine

Note: The DegreesOfFreedom in output are just the length of the number of observations minus 1. If your theoretical frequencies have been derived without using the data, test with one more degree of freedom.

- Evaluating this expression gives you a rough check on the formulas (not evaluated here).

```
Pearson[Hold[of], Hold[tf]]
```

- Let us first compute the test statistic for the first problem. Since the default option is Show → All, we get to see all information.

```
pears = Pearson[obsFreq, theoFreq]
```

Observed						
5	12	16	7	10	15	5
Theoretical						
10	10	10	10	10	10	10
Pearson (o - t)^2 / t						
2.5	0.4	3.6	0.9	0	2.5	2.5

```
{Test → Pearson, TestStatistic → 12.4, NumberOfObservations → 70, DegreesOfFreedom → 6}
```

```
Pearson[Table]
```

```
{2.5, 0.4, 3.6, 0.9, 0, 2.5, 2.5}
```



- *Testing* means confronting the test statistic with the  $\chi^2$  critical value. Alternatively put, if the PValue for that test statistic is smaller than the significance level, then we reject the hypothesis.

```
result[daysDontMatter, Pearson] = Test[Chi2, pears]
```

```
{Test → Pearson, TestStatistic → 12.4, DegreesOfFreedom → 6,  
  Chi2PValue → 0.0536176, SignificanceLevel → 0.05, Do → Accept}
```

- The similar steps for the hypothesis that market days don't matter.

```
pears2 = Pearson[obsMarket, theoMarket]
```

```
Observed
```

```
31      39
```

```
Theoretical
```

```
20      50
```

```
Pearson (o - t)^2 / t
```

```
6.05      2.42
```

```
{Test → Pearson, TestStatistic → 8.47, NumberOfObservations → 70, DegreesOfFreedom → 1}
```

```
result[marketsDontMatter, Pearson] = Test[Chi2, pears2]
```

```
{Test → Pearson, TestStatistic → 8.47, DegreesOfFreedom → 1,  
  Chi2PValue → 0.00361051, SignificanceLevel → 0.05, Do → Reject}
```

### 6.11.5.5 The likelihood ratio test statistic

```
LogLikelihoodRatio[o, t, opts]
```

computes  $-2 \log[\text{LikelihoodRatio}[o, t]]$  with  $o$  the relative or absolute observed frequencies and  $t$  the theoretical probabilities. Inputs  $o$  and  $t$  can be multidimensional as long as they have the same shape

```
LogLikelihoodRatio[Table]                      contains the test cells after evaluation
```

Note: The DegreesOfFreedom in output is just the length of Flatten[t] minus 1. The MLEstimate option in output gives the maximum likelihood estimate against which was tested.

- The test statistic for the first problem.

```
LLR1 = LogLikelihoodRatio[obsFreq, theoFreq / 70]

Observed

5      12      16      7      10      15      5

Theoretical

1/7      1/7      1/7      1/7      1/7      1/7      1/7

Log[(t / t(est))^ni]

3.46574  -2.18786  -7.52006  2.49672  0.  -6.08198  3.46574

{Test → LogLikelihoodRatio, TestStatistic → 12.7234,
NumberOfObservations → 70, DegreesOfFreedom → 6,
MLEstimate → {0.0714286, 0.171429, 0.228571, 0.1, 0.142857, 0.214286, 0.0714286}}
```

- The test for the the first problem.

```
result[daysDontMatter, LLR] = Test[Chi2, LLR1]

{Test → LogLikelihoodRatio, TestStatistic → 12.7234, DegreesOfFreedom → 6,
Chi2PValue → 0.0476449, SignificanceLevel → 0.05, Do → Reject}
```

- The test statistic for the second problem.

```
LLR2 = LogLikelihoodRatio[obsMarket, theoMarket / 70]

Observed

31      39

Theoretical

2/7      5/7

Log[(t / t(est))^ni]

-13.5859      9.68999

{Test → LogLikelihoodRatio, TestStatistic → 7.79182, NumberOfObservations → 70,
DegreesOfFreedom → 1, MLEstimate → {0.442857, 0.557143}}
```

- The test for the the second problem.

```
result[marketsDontMatter, LLR] = Test[Chi2, LLR2]

{Test → LogLikelihoodRatio, TestStatistic → 7.79182, DegreesOfFreedom → 1,
Chi2PValue → 0.00524833, SignificanceLevel → 0.05, Do → Reject}
```

6.11.5.6 Collecting all test results in a table

We have used the array "result[ H0, kind of test]" to store the results. We use this now to construct a summary table, by employing the routine InsideTable.

```
heading = TableHeadings → {{daysDontMatter, marketsDontMatter},
{Pearson, LLR}};

InsideTable[Set, result, heading]

InsideTable[Show, Do]
```

	Pearson	LLR
daysDontMatter	Accept	Reject
marketsDontMatter	Reject	Reject

6.11.6 The options

With the LogLikelihoodRatio it does not matter whether one of the classes has zero observations. We can extend this for the case when a theoretical probability is zero.

```
Limit[n Log[n], n → 0]
```

0

For the Pearson statistic, we get Indeterminate ( $t \rightarrow 0$ ) and ComplexInfinity ( $t \rightarrow 0, n \rightarrow 0$ ).

```
Limit[(o - t)^2 / t, t → 0]
```

Indeterminate

Since a zero number of observations can be interpreted as that these are irrelevant, the defaults of Indeterminate, ComplexInfinity and Infinity in Options[Chi2] replace these values with 0. MessagesQ controls messages Infinity::indet and Power::infy for divisions by zero. So you may set everything to zero but still be warned when this occurs. By default, you are not warned.

Defined separately are the routines SumLog[x], xLogX[x], xLogP[p, x] and xPowerX[x] that are defined to perform like this always (i.e. without such option control).

Other options of Chi2 are given in the following table. Some of the options will be set by routines, but may still be mentioned.

<i>option</i>	<i>typical default value</i>	
SameQ	True	o and t are tested on consistency
RealQ	True	Rationals are set to Reals
Show	All	print all information; None suppresses output on ProbabilityMatrix
SignificanceLevel	0.05	
DegreesOfFreedom	Null	
NumberOfObservations	Null	
Partition	All	independence for all dimensions, otherwise give a list
Test	Null	LogLikelihoodRatio or Pearson
TestStatistic	Null	

6.11.7 Subroutine PrEst

PrEst[ID, data, opts]	the theoretical probabilities under assumed independence
PrEst[ID, Table]	contains the results after evaluation

The option Partition applies to this routine. For output, Show → None suppresses the sometimes long Probabilitymatrix → PrEst[ID, Table] option.

# 6.12 Crosstables and the $\chi^2$ test

## 6.12.1 Summary

This enables one to make crosstables from questionnaires and apply the  $\chi^2$  test. Larger files can be automatically read, and only the key results will be reported.

`Economics [CrossTable]`

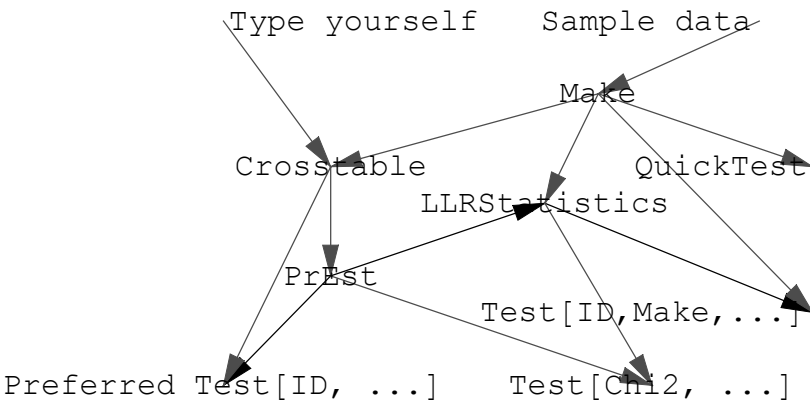
## 6.12.2 Introduction

The general statistical procedure has been discussed in section 6.10 above. We now consider crosstables or contingency tables with dimensions  $\{n_1, n_2, \dots\}$ . The assumption is that the  $n$  observations are sampled from a multinomial distribution.

Some general properties of the package are:

- a. You can get a crosstable by typing one yourself or by starting with sample data and then apply the procedure `Make`. You can show these tables with `ShowCT`.
- b. You can submit the crosstable to `Test[ID,...]`, which tests whether the classifications are independent from each other.
- c. Note though that `Make` already has computed a number of results from the original raw data. So, from `Make`, you may wish to apply a `QuickTest`, or follow up with `LLRStatistics` and other test routines.

The main relations are given in the following diagram.



For the readability of above scheme and diagram we have applied some simplification. Additional details are:

- i. `Make[ ]` has the additional feature that it can use weights on the data. One of the sample data variables can be selected as such a weight. With prices and quantities, or number of cases and values of the cases, one might wish to test on both volumes and values. As a result, a "system of two crosstables" comes into existence, with test results in parallel. For this, you would use `Test[ID, Make, ...]`.
- ii. There is also the procedure `MakeAndSave` that allows you to work in various contexts and save results of large crosstables. You can review summary results and find more detail in files on your hard disk.
- iii. You can do some of the steps yourself. When you do so, you need the procedure `PrEst`, which estimates the probabilities under independence.
- iv. `Test[ID,...]` calls (1) `PrEst` for the probabilities and (2) `Test[Chi2,...]` for the hypothesis test.
- v. `Test[ID, Make, ...]` calls `LLRStatistics` for the test statistics and `Test[Chi2,...]` for testing the separate components. With a quick test, only border sum results are used. Under the option `QuickTest → False`, also `PrEst` is called, and then inner cell likelihood ratios are computed.

The package will be discussed while using an example questionnaire.

### 6.12.3 Example questionnaire

We want to check whether the weather is better in the weekend. Our idea is that pollution is less in the weekend, especially on Sundays. Questionnaires are put out on Thursdays and Fridays (with worst pollution) and Sundays (with least pollution). We do this in January, May, August and October to get a seasonal effect. Weather can be Good or Normal, conditional of course to the season. We add income levels of respondents in \$ thousands per annum for control, expecting that weather views are independent of income. We include a panel opinion, and include the conclusion whether people are telling the truth or not (T or F) according to the panel. The first item in the questionnaire is the number in the list. The questionnaire data are entirely fictitious.

- Our data thus are: Number, Month, Day, Weather, Truth, \$1000

```
questionnaire =
{{10, Jan, Thu, Good, F, 10.},
 {11, Jan, Thu, Normal, T, 43.},
 {20, Jan, Sun, Normal, T, 8.},
 {21, Jan, Sun, Good, F, 18.},
 {22, Jan, Sun, Good, F, 11.},
 {23, Jan, Sun, Good, F, 39.},
 {30, May, Sun, Good, T, 23. },
 {40, Aug, Fri, Normal, T, 11.},
 {44, Aug, Fri, Good, F, 16.},
 {45, Aug, Fri, Good, F, 3.},
 {50, Oct, Sun, Good, F, 50.},
 {60, Oct, Thu, Normal, T, 22.}};
```

- To become useful, these data must be transposed. The proper data exclude the observation number, and we take the weights apart. We also assign separate names to the various dimensions, so that we can access them directly.

```
temp = Rest[Transpose[questionnaire]];
tocross = Drop[temp, -1];
{months, days, weather, truth} = tocross;
income = Last[temp]

{10., 43., 8., 18., 11., 39., 23., 11., 16., 3., 50., 22.}
```

- We may now make the following twodimensional selection.

```
data = {months, truth}

( Jan Jan Jan Jan Jan Jan May Aug Aug Aug Oct Oct )
( F T T F F F T T F F F T )
```

6.12.4 Make and QuickTest

Make [data, opts]

makes a list of rules containing all  
basic information about a crosstable of the data

Note: Data normally is a list of Vectors of equal length. It can be a list of Strings too (i.e. the names of variables that contain such vectors), and then the names are concatenated with the Context in current options; in this manner a structural form can be given for various contexts.

```
restwo = Make[data, CrossEntriesQ → True ]

{Variables → In, Dimensions → {4, 2}, NumberOfObservations → 12,

Frequencies → {  $\begin{pmatrix} 3 & \text{Aug} \\ 6 & \text{Jan} \\ 1 & \text{May} \\ 2 & \text{Oct} \end{pmatrix}$ ,  $\begin{pmatrix} 7 & F \\ 5 & T \end{pmatrix}$  }, Categories → {{Aug, Jan, May, Oct}, {F, T}},

BorderSums → {{3, 6, 1, 2}, {7, 5}}, CrossEntries →  $\begin{pmatrix} \{\text{Aug}, F\} & \{\text{Aug}, T\} \\ \{\text{Jan}, F\} & \{\text{Jan}, T\} \\ \{\text{May}, F\} & \{\text{May}, T\} \\ \{\text{Oct}, F\} & \{\text{Oct}, T\} \end{pmatrix}$ , CrossTable →  $\begin{pmatrix} 2 & 1 \\ 4 & 2 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$  }
```

option	typical default value	
Context	Global`	to concatenate with name input
Position	All	otherwise a list by Position[ ]
Weight	Null	data list or a variable name
WeightUnit	DivSmallestNonZero	or e.g. Identity or N[# / Add[#]]&
CrossEntriesQ	False	CrossEntries in output

The convention in *Mathematica* is to control printing with options, and not with code numbers. However, in this case the codes provide a good overview of the combinations.

ShowCT[{rules}, type: I] prints the crosstable with TableForm. Type codes are:  
1: cases observed,  
2: cases, observed and theory  
3: cases and weights, observed  
4: cases and weights: observed and theory  
5: weights: observed and theory.

Note 1: Theoretical values are in levels, i.e. probabilities times the number of observations.

Note 2: When theoretical probabilities or weights are not in the list, but according to the type code should be used, then values are taken from PrEst[ID, Table || Weight] and LLRStatistics[Table || Weight].

Note 3: Non-integer data are rounded till 0.1. You may use width printing or try PaddedForm to get a neat output.

■ All this amounts to, for above case:

```
ShowCT[restwo]

      F      T
Aug    2      1
Jan    4      2
May    0      1
Oct    1      1
```



- We have direct access to the crosstable.

```
CrossTable[May, T]

1
```

QuickTest[ID, {rules}]      applies a likelihood ratio test with the Full ID statistic

- A quick test shows that telling the truth is independent of the season, at a high significance level.

```
QuickTest[ID, restwo]

{Test → FIDLLRStatistic, TestStatistic → 2.07079, DegreesOfFreedom → 3,
  Chi2PValue → 0.557844, SignificanceLevel → 0.05, Do → Accept}
```

- We get more information as follows, and it is nice to see that the different computations give the same result (not evaluated here).

```
Test[ID, CrossTable /. restwo]
```

QuickTest relies on the "Full ID LLR statistic" routine. For the computation of the LLR test statistic for crosstables, the use of the inner cells can be avoided with the xLogX routine.

FIDLLRStatistic[x\_List, r: {}]

where x can be a crosstable or a list of rules and r a list of rules

FIDLLRStatistic[CrossTable → x, r: {}]      idem

FIDLLRStatistic[Rule → x, r: {}]      idem

Note: The input rules must mention BorderSums, CrossTable, Dimensions, and NumberOfObservations. Also, r is a list of rules that will be used with priority, thereby possibly preventing recomputations on the original table.

6.12.5 Using weights

In the routines discussed here, the weight option means that the probabilities are taken from the weights but the powers are still taken from the frequencies. One reason is that weights may be inflated, e.g. income will be affected by inflation, while the test is sensitive to the sample size.

Suppose that you do research on embezzlement. Then you have information not only on the number of cases established in court, but also the amounts of dollars involved. If you classify the cases by type, it may be that the values are more important than just the number of cases.

Thus for a  $n \times m$  table with frequencies  $f[i, j]$ , weights  $w[i, j]$  and total weight  $W$ :

- the  $w[i, j] / W$  are the observed or maximum likelihood probabilities
  - the  $w[i, .] / W$  and  $w[., j] / W$  are the marginal probabilities under independence, and for the cells we take the products of these
  - the numbers of observations or powers in the formulas remain the  $f[i, j]$ .
- In our questionnaire the income level can be associated with the truthfulness of the answers.

```
restwoW = Make[data , Weight → income];
```

```
ShowCT[restwoW , 3]
```

Cases: crosstabulation as found

	<i>F</i>	<i>T</i>
Aug	2	1
Jan	4	2
May	0	1
Oct	1	1

Weight: crosstabulation as found, rounded .1

	<i>F</i>	<i>T</i>
Aug	6.3	3.7
Jan	26.	17.
May	0	7.7
Oct	16.7	7.3

- We have direct access to the weight data.

```
CrossWeight[Jan, F]
```

26.

The input weights appear to be different from the output ones. Almost 170 thousand dollars have disappeared. How's that ?

```
{In → Add[income], Out → WeightTotal /. restwoW}
```

```
{In → 254., Out → 84.6667}
```

The reason is the WeightUnit option. It has normalised incomes to the smallest income level. Other settings could have been WeightUnit → Identity or WeightUnit → N[# / Add[#]]&.

If we now submit the data with the weights to a test, then we find that the TestStatistic has dropped and that the PValue thus has risen, which means that the independence of truthfulness (conforming to the panel) and the season is even more likelier if we correct for income. (This does not tell yet what level of income is more truthful.)

```
Test[ID, Make, restwoW]

{Test → FIDLLRStatistic, TestStatistic → 2.07079, DegreesOfFreedom → 3,
  Chi2PValue → 0.557844, SignificanceLevel → 0.05, Do → Accept,
  Weight → {Test → LLR(W): probabilities from Weight, TestStatistic → 1.63073,
    DegreesOfFreedom → 3, Chi2PValue → 0.652442, SignificanceLevel → 0.05, Do → Accept}}
```

### 6.12.6 Outliers

Let  $d$  be the difference between the crosstable and its theoretical value. Here  $d / \sigma[d]$  might be standard normally distributed, and crosstable entries that are more than  $k \sigma$  from the average (i.e. zero) then would be outliers that require our attention. Admittedly, this approach is weak, since we already use the likelihood ratio and it would seem to be more appropriate to determine outliers in those terms. However, the Pearson criterion is based on the assumption that the theoretical distribution of the  $t[i, j]$  conditional on the marginals is a multivariate normal. So the approach taken below is not without some sense.

<pre>CrossOutliers[   k, opts]</pre>	performs Outliers on crosstable data and gives the selection of inner cells. Required are options CrossEntries, CrossTable, ProbabilityMatrix and NumberOfObservations.
--------------------------------------	---

```
Make[tocross, CrossEntriesQ → True];

PrEst[ID, CrossTable /. %];

co = CrossOutliers[1.645, Join[%, %%]];
```

- N gives the number of outliers, and Select tells us which.

```
{N, Select} /. co
```

$$\left\{4, \left\{ \text{CrossEntries} \rightarrow \begin{pmatrix} \text{Aug} & \text{Fri} & \text{Good} & F \\ \text{Aug} & \text{Fri} & \text{Normal} & T \\ \text{Jan} & \text{Sun} & \text{Good} & F \\ \text{Oct} & \text{Thu} & \text{Normal} & T \end{pmatrix}, \text{CrossTable} \rightarrow \{2, 1, 3, 1\} \right\} \right\}$$

- We can make a bar chart of the normalised differences as follows.

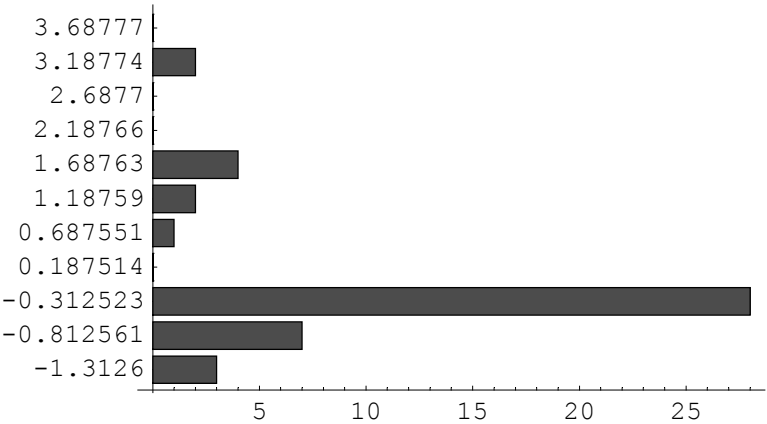
```
zs = Flatten[Normal /. co];

bcs = BarChartServer[zs]

{Factor → 0.1, Range → {-1.54699, 3.47213, 0.500037},
  BinCounts → {3, 7, 28, 0, 1, 2, 4, 0, 0, 2, 0}, Point → {-1.3126, -0.812561, -0.312523,
    0.187514, 0.687551, 1.18759, 1.68763, 2.18766, 2.6877, 3.18774, 3.68777}}
```

<<Graphics`Graphics`

```
BarChart[Transpose[{BinCounts, Point} /. bcs], PlotRange -> All,  
BarOrientation -> Horizontal]
```



6.12.7 Larger data sets

The following aspects are relevant for practical work:

- There may be many variables, combinations, aggregations, and partitions.
- Computing takes time but can be done in spare time.
- Not everything needs to be on screen, if it would be available anyhow if needed.

The following are routines that help us handle these cases.

6.12.7.1 QuickTest[ ], LLRStatistics[ ] and Test[ID, Make, ...]

The output  $y = \text{Make}[x]$  is input for  $\text{Test}[\text{ID}, \text{CrossTable} /. y]$ . Test will however re-compute results already known by Make. Especially when you have large and many cross tables, then it is quicker to directly use the already existing results.

- QuickTest[ ] and LLRStatistics[ ] using the QuickTest -> True option use only the cases, do not use the weights, and do not give ratios of inner cells. Note that in this case ShowCT may not give some results, since cells have not been computed.
- If you have weights and want to perform the parallel test, use Test[ID, Make, ...]. The latter is the same as calling LLRStatistics[ ] once for the test statistics, and the Test[Chi2, ...] for the actual test both on cases and weights.

```
LLRStatistics[{rules}, opts]
```

gives the LLR test statistics on independence using the output of `Make`. If `QuickTest` → `True` (default) the Full ID formula is used. When weights are present, these results are under `Weight` → ... .

```
LLRStatistics[Table || Weight]
```

contains the log likelihood ratio cells when `QuickTest` → `False`. When `Show` → `None`, see `PrEst[ID, Table || Weight]` for the probability matrices

- With weights we here only get the test statistics, and not the test.

```
LLRStatistics[resttwoW, QuickTest → True]
```

```
{Test → FIDLLRStatistic, TestStatistic → 2.07079,
DegreesOfFreedom → 3, NumberOfObservations → 12, Weight →
{Test → LLR(W): probabilities from Weight, TestStatistic → 1.63073, DegreesOfFreedom → 3}}
```

### 6.12.7.2 MakeAndSave[ ]

`MakeAndSave` puts only summary statistics on screen, and detail results can be recovered from file. The file can be read with `!!` and `<<`, and in the latter case works as input for *Mathematica* so that one has the rules directly available.

```
MakeAndSave[kRange, data, opts]
```

computes a crosstable for the data and performs the likelihood ratio test on full independence. Then it gives the outliers in `kRange` and writes results to file.

Note: Default are `Options[Make]` and `Options[MakeAndSave]`. The filename can be given as `File` → `String`, or `File` → {`dir_String`, `n_Integer`, `name_String`}. The integer gives the first places of the Context (default 3 letters of "Global"). The file type is ".mac". The default file name written is "Glocross.mac".

- For the following example run, we first check the options settings.

```
Options[Make]
```

```
{Context → Global', Position → All, Weight → Null,
WeightUnit → DivSmallestNonZero, CrossEntriesQ → False}
```

```
Options[MakeAndSave]
```

```
{File → {c:\Dump\, 3, cross}}
```

- A call of MakeAndSave with  $1.5\sigma$  range.

```
MakeAndSave[1.5, {{w,w,w,r,t,r,t}, {y,y,y,n,n,y,y}},
             Show → None, RealQ → True,
             Weight → {10., 65., 45., 3., 3., 4., 6.}];
```

Context: Global`

Filename: c:\Dump\Glocross.mac

Test on cases, number: 7

```
{Test → FIDLLRStatistic, TestStatistic → 2.8306, DegreesOfFreedom → 2,
  Chi2PValue → 0.242853, SignificanceLevel → 0.05, Do → Accept}
```

A rate of 0 outliers at  $k = 1.5$

This concerns crosstable cells:

```
{}
```

Test on weights

```
{Test → LLR(W): probabilities from Weight,
  TestStatistic → 7.1128, DegreesOfFreedom → 2,
  Chi2PValue → 0.0285413, SignificanceLevel → 0.05, Do → Reject}
```

A rate of 0.166667 outliers at  $k = 1.5$

This concerns crosstable cells:

```
( w y )
```

Reading the file shows that it contains *Mathematica* statements. Reading the file with << assigns variables that you can work with, in particular, for the default parameter settings, a variable GlocrossInput that contains the input data, and GlocrossResult that contains results.

- Reading with !! and not << (not evaluated here)

```
!!C:\Dump\glocross.mac
```

# 7. Econometrics

## 7.1 Introduction

---

Economics is crucially dynamical. We may do statics, and at times statics may even be advisable, but we do statics only to get the dynamics straight.

My university education dates from before David Hendry conquered the textbooks. From those years I recall a sense of wonder that the two methods of estimation, in levels or in rates of change, somehow were different but should rather be the same, but that is all. Perhaps regrettably, my perception still is very much shaped by those pre-Hendry days. To me, the analysis starts with an economic model, and not with a Data Generating Process. You first want to understand the model, so that you know what you are going to test. You talk to practitioners about their experiences, and compare these with the model. Then you collect the data. Of course, for much research like on the national economy the data collection has been institutionalised, and you step onto a bandwagon where existing observations guide your modeling. Yet, also with my long involvement in precisely this macro economic modeling, I entertain a deep distrust of the data. All in all: while the error correction methods came as an enlightenment, and while David Hendry's, "Dynamic econometrics", Oxford 1995 was a valuable buy, and while I hope that such methods will help to establish the empirical validity of economic theories including my own, and while such methods indeed may be crucial given the opening statement above, the fact remains that I have less empathy with time series analysis beyond some essentials. And the following results reflect this attitude.

These results may be categorised as follows:

- We start out with dynamic modeling and the traditional estimation of nonlinear systems, still without tools for error correction and such. The section on timeseries is basically a reference to the work of others, namely the Time Series pack of Wolfram Research Inc. and the chapter by Robert A. Stine in Varian 1993.
- The sections on neural networks and genetic programming are included since these are the more promising unconventional tools. For neural networks we basically refer to James A. Freeman's book on the subject. For genetic programming package I got inspired by the *MathSource* package of Jonathan Kleid, and below gives my own version of this increasingly popular technique.
- Econometrics, as the quantification of economics, includes operations research / management science. The three sections on this subject with topics like linear programming and queueing complete the chapter.

Notice that the allocation of subjects remains rather arbitrary. For example, linear programming has been used to prove the existence of a competitive equilibrium, and it might be included in the section on

applied general equilibrium analysis. As another example, one may argue that the dynamic modeling package has little to do with economics per se, and is just a technique that belongs to the Enhancement chapter. Again, I here just have followed the outline of my own education.



## 7.2 Dynamic modeling

---

### 7.2.1 Summary

This provides a basic environment for dynamic modeling. The main package is `Model`` that applies `NSolve` for solutions over a period, and that provides for the data management of subsequent periods and of different simulation runs. A small package is `OneLag`` that uses `Solve` and `NestList` for models with only one lag.

- Load the packages.

```
Economics[OneLag]
```

```
Economics[Model]
```

```
Contents["Graphics`MultipleListPlot"]
```

### 7.2.2 Introduction

A model is a system of equations, i.e. a list of equations and rules how to use these. A model has *endogenous* variables  $y$  explained by the model, and *exogenous*  $x$  variables explained by another source. The model also has *coefficients* that are supposed to be constant (otherwise these should be variables). The model has a *structural form* when its structure reflects the causal process and when the coefficients thus have a clear causal interpretation. For example there are definitions and institutional equations with perfectly known coefficients and there are behavioural equations with estimated coefficients. If a model is not in the structural form then it is in a *reduced form*. When an equation has been estimated with an error term (that stands for left out variables), then the equation in the model should have an error term too. This is often called an *autonomous* variable, and it is one of the exogenous variables.

For a *dynamic* model, the variables  $y[t]$  and  $x[t]$  are time dependent, and the *predetermined* variables are not only the exogenous variables but also the lagged endogenous  $y[t-r]$  and exogenous  $x[t-r]$ . The solution of the model is not just a list of variables, but a time series or time path for each. Some calculation methods solve for the whole time path directly (like multiple shooting). Common methods work sequentially: Given the lagged variables  $y[t-r]$  and  $x[t-r]$  and current exogenous  $x[t]$ , one solves for  $y[t]$ , stores the data, updates  $t$  one period, and repeats the process, up to desired  $t$ .

Economic equilibrium occurs when expectations of demand and supply are satisfied. With expectations in a model, the future values of the variables could feed back into the current solution. This would be the case in particular when the expectations are model-consistent, i.e. described precisely by the model. In that case the method of solving for the whole time path directly seems the logical approach. Alternatively, one can show that forward looking expectations can also be represented by a scheme on lagged values. In practice expectations will not be model-consistent, agents will have different schemes and asymmetric information, and the common sequential solution method still has its value. Admittedly, if *Mathematica*

would make whole path solution methods available, then this would be of great value. Currently, the packages considered here implement the sequential method.

*Mathematica* provides routines `Solve`, `NSolve` and `FindRoot` to solve *static* models. The routines presented here embed these routines into dynamics.

7.2.3 Models with one lag

When models have only one lag and can be solved by `Solve`, then we can use `NestList`. We only have to specify the endogenes and the startvalues, and can run the model directly. This approach is surprisingly strong, both in ease of use and range of application. Not only are many models of this type without any additional manipulation, but we may also note that systems with longer lags can be written as systems with one lag, by using intermediate variables. For example  $x1 = x[t-1]$ ,  $x2 = x1[t-1]$ , ...

See also the routine `NSolveLinearDynamics` for systems of linear first order difference equations  $x[t+1] = M x[t] + e[t]$ , where  $M$  is a vector and  $x$  and  $e$  are vectors.

OneLag[equations\_List, endogenes\_List, startvalues\_List, iter, t\_Symbol]

assumes that the endogenes[t] in the equations have only one lag,  
then (a) solves for the endogenes,  
and (b) uses NestList to iterate iter times starting from startvalues,  
and (c) stores the solution in \$OneLag Null

OneLag[Set, eqs\_List, endos\_List, t\_Symbol]step (a)

OneLag[N, start\_List, iter]step (b) and (c)

\$OneLagthe solution

■ An example is:

```
eq1 =    x[t] + y[t - 1] == y[t]/2

x(t) + y(t - 1) ==  $\frac{y(t)}{2}$ 

eq2 =    y[t] == x[t - 1] + 1

y(t) == x(t - 1) + 1
```

```
OneLag[{eq1, eq2}, {x, y}, {2, 3}, 4, t]
```

$$\begin{pmatrix} 2 & 3 \\ -\frac{3}{2} & 3 \\ -\frac{13}{4} & -\frac{1}{2} \\ -\frac{5}{8} & -\frac{9}{4} \\ \frac{39}{16} & \frac{3}{8} \end{pmatrix}$$

```
??$OneLag
```

\$OneLag contains the model solution created by OneLag[...]

$$\$OneLag(\{x_, y_ \}) = \{ \frac{1}{2} (x - 2 y + 1), x + 1 \}$$

7.2.4 The Model` package

For systems with various and longer lags, the Model` approach is more user friendly. Model[ ] sets up the model, and NSolveYear[ ] and NSolvePeriod[ ] actually run it, with NSolve.

Endogenes	list of Symbols used for the endogenes
Exogenes	list of Symbols used for the exogenes
ModelVariates[ ]	simply joins the existing sets of Endogenes and Exogenes
ModelVariates[ Equations]	determines the Endogenes and the Exogenes from the Equations. The model-canonical form is used, see \$Endogenes.
ModelVariates[ Store]	stores UndefinedSymbols (see body of the text)
EndogenesRule[ ]	gives a rule so that one can switch from the non-model-canonical form to the model-canonical form. See eluR

Two canonical forms are recognised:

- Model[ ]-Canonical Form: when the "[t]" is not affixed to the endogenes. Example:  $y == 1.2 x[t] + y[t-1]$ . In this case Model[ ] can find the endogenes itself.
- Non-Model[ ]-Canonical Form: when the "[t]" affixes to the endogenes too. Example:  $y[t] + x[t] == 0$ . In this case, Model[ ] can no longer automatically determine the endogenes and exogenes, and thus one has to specify them.

Which format one chooses depends upon legibility and variability of the model specification. A fixed model can be better legible if one drops the '[t]'s. When one varies the exogenes from run to run then it may be easier to adjust \$Endogenes in Options[Model] than adapting those [t] parts.

It has been considered to use the method  $y == 1.2 x[0] + y[-1] + a x[-2]$ , without a time indicator at all. The readability clearly increases. However, we want to be able to include other functions dependent upon time, such as a trend. It also is a strong point that a variable call as  $y[2000]$  will be possible, as you

may note below. In the model-canonical form you can always give  $t \rightarrow 0$  if you want to get rid of the 't' for a reading check.

7.2.5 Model[ ]

Model[*rules*]     *or*   Model[{*rules*}, *opts*]

sets up Equations[t\_], finds the Endogenes and Exogenes in the Equations,  
and fills the Data. The rules must be as in Options[Model]

- The time indicator 't' is taken from the Timing option of backlog operator B.

**Timing /. Options[B]**

*t*

The coefficients in your model may be unassigned Symbols. Options[Model] allow you to give default values, to be entered as rules for replacement. The endogenes then can be identified as the only symbols left without '[ ]'. In this manner you also can run the model with different coefficients, since you can change the defaults.

<i>option</i>	<i>typical default value</i>	
Begin	1	otherwise a year like 1999
Coefficients	{ }	alternatively a list of rules for replacing coefficients with values {coef → val, ...}
Equations	{ }	list of equations
SetData	{ }	{x1 → {data1}, x2 → {data2}, ....}
\$Endogenes	Automatic	the endogenes are identified as symbols without a time indicator; alternative is a list of symbols
Find	NonAttributedTerms	routine to find variables (see body of the text)

- Setting the equations, and for generality we include a trend term.

**SetOptions[Model, Equations →**  
    {y1 == a y2 + Exp[x2[t]] + x1[t-1] + .001 t,  
    y2 + y1 == a x2[t-2] + c y1 + d y1[t-2] }];

- Setting the coefficients.

**SetOptions[Model, Coefficients → {a → 0.3, c → 0.4, d → -0.5}];**

- Call `model[ ]`, to set up everything for computation.

```
Model[ ]
```

- We can peek into the results. The equations are printed, the rules are output.

```
Results[Model]
```

```
Equations(t_) := {y1 == 0.001 t + ex2(t) + a y2 + x1 (t - 1) ,
  y1 + y2 == c y1 + a x2 (t - 2) + d y1 (t - 2) } /.
(Coefficients /. Options[Model])
```

```
{$Endogenes → {y1, y2}, $Exogenes → {x1, x2},
  Begin → 1, TimeZone → {3, Min(EndYear(x1), EndYear(x2))}}
```

- Note that `Model[ ]` has defined `Equations[t_]`.

```
Equations[0]
```

```
{y1 == 0.3 y2 + ex2(0) + x1(-1), y1 + y2 == 0.4 y1 + 0.3 x2(-2) - 0.5 y1(-2)}
```

### 7.2.6 Setting the data

`Model[ ]` relies on the `Data`` package that is automatically called by `Model``. If the `Model[ ]` option `Begin` has been set to a meaningful value and when `SetData[ ]` has been called, then the maximal simulation period can be computed, counting the `LongestLag` in the model and the available data. This maximal simulation period is called "TimeZone" in `Results[Model]`.

- We can set the data by calling `Model` again. The exogenes need data for the full simulation period, the endogenes only for the lagged startperiod.

```
dat = {x1 → Table[Random[], {10}],
  x2 → Table[Random[], {10}],
  y1 → Table[Random[], {3}],
  y2 → Table[Random[], {3}] };
```

```
Model[Begin → 1997, SetData → dat]
```

- Having set the data, we now can directly access them.

```
x2[1997]
```

```
0.724854
```

- The equations for a particular year have been greatly simplified.

```
Equations[1999]
```

```
{y1 == 0.3 y2 + 3.50155, y1 + y2 == 0.4 y1 + 0.199734}
```

If the data set is large then it is more efficient to set `SetData → {}` in `Options[Model]`, and have an option in `Model[ ]` or a separate call of `SetData[ ]` after `Model[ ]`. The reason is, that `Options[Model]` are called for every year to vary the parameters. A smaller `Options[Model]` statement makes for a simpler call. The current options are:

#### **Options[Model]**

```

{Begin → 1, Coefficients → {a → 0.3, c → 0.4, d → -0.5},
  Equations → {y1 == 0.001 t + ex2(t) + a y2 + x1(t - 1), y1 + y2 == c y1 + a x2(t - 2) + d y1(t - 2)},
  Find → NonAttributedTerms, SetData → {}, $Endogenes → Automatic}

```

The method of automatically finding the variables in the model has a relation with the setting of the data. Once the data have been set, a method like `UndefinedSymbols` would no longer work. The default `Find` method `NonAttributedTerms` does not suffer from that problem.

If you wish to use the method `UndefinedSymbols` for finding the variables, though, then you should know that `Model[ ]` has found a way to allow you to do so. The trick is that `ModelVariates[ ]` keeps a memory of the undefined symbols used at any call of `ModelVariates[Equations]` (by `Model[ ]`). This memory is put in `ModelVariates[Store]`. When the data have been set and you use the variables later again, for example for a model variant, then `ModelVariates[ ]` can always recognise them from the memory. Thus the only thing to remember for the method `UndefinedSymbols` is that for a *first* call you should always call `Model[ ]` *before* `SetData[ ]`. This actually happens if you use the `Model[ ]` routine with active `SetData[ ]` options. If you would have done a `SetData[ ]` before `Model[ ]`, either of the following repairs the error: (a) Clear the variable, and proceed in the correct order, (b) `AppendTo[ModelVariates[Store], variable]`, (c) set the `$Endogenes` option to the full list of endogenes. All this, just in case you wish to use the option setting `UndefinedSymbols`.

### **7.2.7 Running the model**

`NSolve` has been chosen rather than `Solve`, since `NSolve` as a numerical routine is more robust in finding solutions. There is a price, though. With `Solve` one might solve the model once, and use that solution for the whole period. `NSolve` however needs to solve the model each period. However, since the coefficients and predetermineds are set at values, in this stage, the real model to solve is smaller and often less complicated. For example, a call of `Equations[1999]` directly uses the value of `x2[1997]` that has been set before.

<code>NSolveYear[t]</code>	NSolves for t and sets the endogenes {y1[t], ... }. It does not check on StartYear, as NSolvePeriod does
<code>NSolvePeriod[t (, s), opts]</code>	calls NSolveYear for the given period, default s = StartYear + LongestLag. The minimum of {t, s} is taken as the beginning, the maximum as the end
<code>Solution[t]</code>	contains the all solutions for year t

Note: NSolve can generate more solution points. Above routines take the first solution point.

```
nsp = NSolvePeriod[1997, 2001]
```

```
NSolvePeriod::begin : Earliest year is 1999
```

$$\begin{pmatrix} 3.0182 & -1.61118 \\ 3.46299 & -2.09448 \\ 3.38765 & -3.47111 \end{pmatrix}$$

- The simulation data are directly available.

```
y1[2000]
```

```
3.46299
```

## 7.2.8 Data management

- Store[ ] now has the AddedUsage that Store[label] puts the current values of the Endogenes into Data[#, label].

```
Store[run1]
```

- Let us choose new data for an exogene, and run the model again, with label run2.  
Note that at this moment you might also change the coefficients in Options[Model].

```
SetData[x1 → Table[Random[], {10}], 1997, run2]
```

```
nsp2 = NSolvePeriod[1997, 2001]
```

```
NSolvePeriod::begin : Earliest year is 1999
```

$$\begin{pmatrix} 3.0549 & -1.63321 \\ 4.23409 & -2.55713 \\ 2.69379 & -3.07315 \end{pmatrix}$$

```
Store[run2]
```

- Evaluating this will give a lot of `Data[var (, label)]` statements (not evaluated).

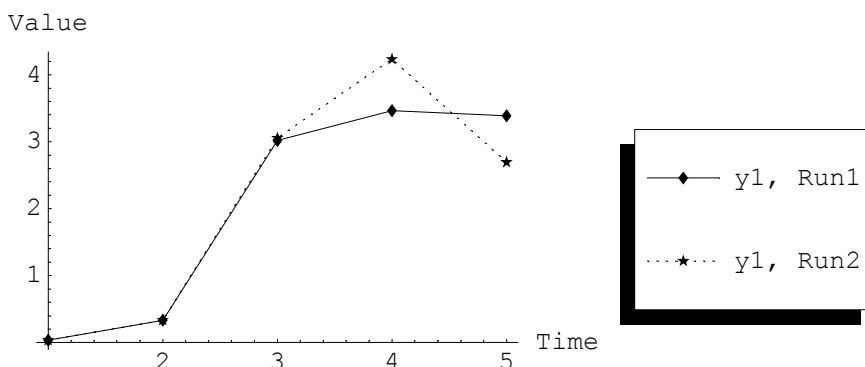
??Data

- We can compare the two runs, e.g. for variable `y1`.

```
{Data[y1, run1], Data[y1, run2]}
```

```
(0.0354442 0.333271 3.0182 3.46299 3.38765)
(0.0354442 0.333271 3.0549 4.23409 2.69379)
```

```
MultipleListPlot[Sequence @@ DataFilter @@ %, PlotJoined → True,
AxesLabel → {"Time", "Value"}, PlotLegend -> {"y1, Run1", "y1, Run2"},
LegendShadow → {-0.05, -0.05}];
```



### 7.2.9 Step by step

The following does not call the `Model[ ]` routine, but does the steps one by one.

- Note that if you set the `Timing` option at another value, such as `Timing → T`, then you may have to `Clear[Equations]` first.

```
Exogenes = {x1, x2};
Endogenes = {y1, y2};
StartYear = 1997;
```

```
SetData[{x1 → Table[Random[], {5}],
x2 → Table[Random[], {5}],
y1 → Table[Random[], {3}],
y2 → Table[Random[], {3}] }];
```

```
Equations[t_] := {y1 == a y2 + x1[t-1] ,
y2 + y1 == a x2[t-2] + c y1 + d y1[t-2]} /.
{a → .3, b → .4, c → .1, d → .5}
```



**NSolvePeriod[1999, 2000]**

$\begin{pmatrix} 0.534393 & -0.128037 \\ 0.920349 & -0.140529 \end{pmatrix}$

## 7.3 Estimation of nonlinear systems

---

### 7.3.1 Summary

The `Estimate`` routine is a shell for *Mathematica*'s routine `NonlinearRegress`, with the possibility of a system of equations, with easier input, and with approximate t-values in output.

**Economics [Estimate]**

### 7.3.2 Introduction

This package uses *Mathematica*'s routine `NonlinearRegress` (see `Statistics`NonlinearFit``), and adds the following features:

- a. systems of equations
- b. more natural format for input
- c. other output, such as t-values.

By a more natural format for input we mean that you can enter an equation like  $y == A lab^a cap^b$  for example, or a list of equations like these. The only limitation here appears to be that you cannot use the `Max` and `Min` functions. You also can enter the data as `{lab → {30.3, 31.5, ..} ...}` or `{lab → Data[lab], ...}` so that you don't have to worry about the order of  $y$ ,  $lab$  and  $cap$ .

Since `NonlinearRegress` is the main engine, its options remain important. Also, the formats for the coefficients (parameters) are just the same.

`Estimate` calls `Estats[ ]` for its statistical report. Option `CovarianceMatrix → True` (default) gives an estimated covariance matrix using the inverse Hessian. When input option `Estimate → False`, then no estimation is tried, and only t-values are computed for startvalues for the coefficients. When option `Out → False` then the original output of `NonlinearRegress` is given.

### 7.3.3 Single equation

```
Estimate[y_Symbol == rhs, {y → data1, var2 → data2, ...}, coefficients_List, opts]
```

estimates coefficients of a single-equation model using the data

Let's take the example of the dual CES demand function for a factor of production given total costs and the prices of all factors. This demand has the substitution parameter  $S$  and, for two factors, only a single parameter  $c$  for the weight of the factors. Note that the expression has been created with the `CES`` package.

- Let's set some arbitrary data series.

```
labour = 2 + Table[Random[], {10}];
capital = 3 + Table[Random[], {10}];
wages = 30 + Table[Random[], {10}];
interest = 10 + Table[Random[], {10}];
cost = labour wages + capital interest;
```

- For estimation, we simply give the relation in formal variables and parameters, have rules that assign data to the variables, and lists for the coefficients. All variables must have the same number of observations. In this case we also supply startvalues for the parameters.

```
Estimate[lab ==  $\frac{\text{Cost} \left(\frac{c}{w}\right)^S}{(1-c)^S \text{rp}k^{1-S} + c^S w^{1-S}}$  ,
  {lab → labour, w → wages, rp k → interest, Cost → cost},
  {{c, .5}, {S, 1.5}}]

{AdjustedRSquared → 0.908871, BestFitParameters → {c → 0.723066, S → 3.37172},
  Correlation → 0.958643, CovarianceMatrix →  $\begin{pmatrix} 0.000138314 & 0.0222873 \\ 0.0222873 & 3.83059 \end{pmatrix}$ , DegreesOfFreedom → 8,
  EstimatedVariance → 0.0161837, NumberOfObservations → 10, ReducedFormQ → True,
  RSquared → 0.918996, StandardDeviation → {0.0117607, 1.95719}, TValues → {61.4815, 1.72273}}
```

#### Results[Estimate]

```
{FitResiduals → {0.160343, -0.0720315, -0.179516, -0.203132, 0.024791,
  0.00132486, 0.0535955, -0.0413753, 0.0671658, 0.123979}, PredictedResponse →
  {2.5072, 2.31449, 2.25496, 2.41034, 2.74271, 2.40252, 2.56312, 2.08563, 2.78267, 2.82664}}
```

Note: Output of Estimate contains RSquared, Correlation and AdjustedRSquared. RSquared is just Correlation<sup>2</sup> (the squared (centered) correlation coefficient  $R^2$  between  $\hat{y}$  and  $y$ ). AdjustedRSquared takes that latter correlation coefficient, squares it, and corrects for the degrees of freedom.

- The input option Out → False causes the output of NonlinearRegress.

```
Estimate[lab ==  $\frac{\text{Cost} (\frac{c}{w})^S}{(1 - c)^S \text{rp}k^{1-S} + c^S w^{1-S}}$  ,
  {lab → labour, w → wages, rp k → interest, Cost → cost},
  {{c, .5}, {S, 1.5}}, Out → False]

{BestFitParameters → {c → 0.723066, S → 3.37172},

      Estimate      Asymptotic SE      CI
ParameterCITable → c      0.723066      0.0118056      {0.695842, 0.75029},
                  S      3.37172      1.96492      {-1.1594, 7.90283}

EstimatedVariance → 0.0161837,

      DF      SumOfSq      MeanSq
      Model      2      62.4856      31.2428
ANOVA Table → Error      8      0.12947      0.0161837,
      Uncorrected Total      10      62.615
      Corrected Total      9      0.984896

AsymptoticCorrelationMatrix →  $\begin{pmatrix} 1. & 0.968501 \\ 0.968501 & 1. \end{pmatrix}$ ,

      Curvature
FitCurvatureTable → Max Intrinsic      0.0263775 }
                  Max Parameter-Effects      6.63757
                  95. % Confidence Region      0.473568
```

7.3.4 Systems of equations

Estimate[{eq1, eq2, ...}, {var1 → data1, var2 → data2, ...}, coefficients\_List, opts]

estimates a system of equations

Let us include the demand function for capital, and, with labour and capital, include the production function and its output value. We substitute the estimated factors into the production function, to get rid of errors in the observed variables.

- The equations have been produced with the CES` package.

```
estimationModel =
{output ==
```

$$\text{Scale} \left( (1 - c) \left( \frac{\text{Cost} \left( \frac{1-c}{\text{rpk}} \right)^S}{(1 - c)^S \text{rpk}^{1-S} + c^S w^{1-S}} \right)^{1 - \frac{1}{S}} + c \left( \frac{\text{Cost} \left( \frac{c}{w} \right)^S}{(1 - c)^S \text{rpk}^{1-S} + c^S w^{1-S}} \right)^{1 - \frac{1}{S}} \right)^{\frac{1}{1 - \frac{1}{S}}}, \text{lab} == \frac{\text{Cost} \left( \frac{c}{w} \right)^S}{(1 - c)^S \text{rpk}^{1-S} + c^S w^{1-S}}, \text{cap} == \frac{\text{Cost} \left( \frac{1-c}{\text{rpk}} \right)^S}{(1 - c)^S \text{rpk}^{1-S} + c^S w^{1-S}} \};$$

- Note that the use of the Data format allows the following short statement.

```
ToDataRule[output, lab, w, cap, rpk, Cost]
```

```
{output → Data(output), lab → Data(lab),
w → Data(w), cap → Data(cap), rpk → Data(rpk), Cost → Data(Cost)}
```

- So, assign some data.

```
Data[output] = 5 + Table[Random[], {10}];
Data[lab] = 2 + Table[Random[], {10}];
Data[cap] = 3 + Table[Random[], {10}];
Data[w] = 30 + Table[Random[], {10}];
Data[rpk] = 10 + Table[Random[], {10}];
Data[Cost] = lab w + cap rpk // ToData;
```

- And then estimate. Since we are not really estimating, set S at .4.

```
Estimate[estimationModel /. S → .4,
ToDataRule[output, lab, w, cap, rpk, Cost],
{{c, 0.5}, {Scale, 1.}}]
```

```
{AdjustedRSquared → 0.917252, BestFitParameters → {c → 0.547015, Scale → 2.02175},
```

```
Correlation → 0.959221, CovarianceMatrix →  $\begin{pmatrix} 0.00106038 & 0.000715636 \\ 0.000715636 & 0.00261927 \end{pmatrix}$ ,
```

```
DegreesOfFreedom → 28, EstimatedVariance → 0.167623, NumberOfEquations → 3,
```

```
NumberOfObservations → 30, ReducedFormQ → True, RSquared → 0.920106,
```

```
StandardDeviation → {0.0325635, 0.0511788}, TValues → {16.7984, 39.5036}}
```

**Results[Estimate]**

{FitResiduals →  
  {Error(1) → {0.591224, 0.207924, -0.141746, 0.41046, 0.105977, -0.571329, -0.800587,  
    0.745034, -0.694739, 0.557416}, Error(2) → {0.00264704, -0.201063, -0.127716,  
    -0.217412, 0.0516993, 0.0705512, 0.229598, -0.0420353, 0.158734, 0.0279134},  
  Error(3) → {-0.00788325, 0.605246, 0.35766, 0.632403, -0.155193, -0.210857,  
    -0.631922, 0.120271, -0.458308, -0.0794116}}, PredictedResponse →  
  {output → {5.11636, 5.44204, 5.95814, 5.23308, 5.81659, 5.72163, 6.27239, 5.041, 6.49982, 5.33568}  
  lab → {2.20002, 2.33681, 2.58282, 2.25715, 2.49854,  
    2.45915, 2.72536, 2.17903, 2.80626, 2.30815}, cap →  
  {3.15674, 3.36742, 3.61592, 3.20832, 3.59646, 3.53356, 3.78911, 3.07692, 3.97701, 3.25182}}}

**7.3.5 Subroutines**

Estats[ <i>equation_Equal</i> , { <i>data_Rule</i> }, <i>coefs_List</i> , { <i>estimates_Rule</i> }, <i>opts</i> ]	computes basic regression statistics for the equation given the estimate
InverseHessian[ <i>equation_Equal</i> , { <i>data_Rule</i> }, <i>coefs_List</i> , { <i>estimates_Rule</i> }]	gives the inverse hessian of the sum of squared error at the estimated point, determined by formal differentiation
ModelY	(hidden) symbol in Estimate for the collected equation ModelY == ... (for systems of equations)
ReducedFormQ[x]	where x can be an equation or a list of equations. True iff all left hand sides contain single variables (called endogenes) and iff all endogenes only occur in the lhs's.

# 7.4 Timeseries approaches

## 7.4.1 Summary

The `Lags`` package supports the formal analysis of linear functions with lags.

`Economics [Lags]`

There is also an `Arima`` package that transforms the formats of the WRI Time Series pack into Stine's format. These packages will be loaded below when we compare the formats.

## 7.4.2 Introduction

The `Lags`` package allows a more formal analysis of lag structures. In timeseries analysis, discrete time lags are obviously important, and difference analysis outweighs differential analysis. The notions remain basically the same though. A lag structure like  $y[t] = a x[t] + b x[t-1] + c x[t-2]$  has a solution that consists of a particular part and a general part, where the latter is the solution of  $y[t] = 0$ . The lag structure can be represented by a polynomial, in this case  $\text{Pln}[B] = a \text{Identity} + b B + c B^2$ . Here  $\text{Pln}[B] = 0$  is called the characteristic equation, and its solution points provide the basis for the general solution of the lag structure. Analysis of the roots of the polynomial tells us more about cycles and dampening or explosive behaviour.

A longer lag can always be represented as a system of equations with single lags, and some may prefer to perform the analysis with matrices and eigenvalues of these.

Robert A. Stine wrote a chapter on timeseries analysis in Varian (1993) and Wolfram Research Inc. later put out a Time Series pack. Both provide a wealth of routines, and we will shortly compare these existing packages and interface with them.

## 7.4.3 Backward lag operator

<code>B[x[u], u]</code>	gives $x[u-1]$
<code>B[x[t]]</code>	gives $x[t-1]$ when <code>Options[B]</code> give <code>Timing → t</code>
<code>ApplyB[x]</code>	turns on the functional use of <code>B</code> when <code>x = True</code> or <code>Null</code> , turns this off when <code>x = False</code>
<code>Delta[x[t] (, t)]</code>	<code>AddedUsage</code> , gives now $x[t] - x[t-1]$

Note: `ApplyB[]` adjusts the internal definitions of `Plus` and `Times`, `ApplyB[False]` removes these.

- The Timing option is set at Global`t.

**Options[B]**

{Timing → t}

- An example is.

**Delta[x[p] y[p], p]**

$$x(p) y(p) - x(p - 1) y(p - 1)$$

Note: If B has already been claimed for the Global` context then its removal is a bit tricky, in particular since the Lags` packages redefines the Delta routine. RemoveGlobalB however does the necessary steps. It removes this Global`B, calls CleanSlate["Cool`Lags`"] and forces a reload of Cool`Common` and Cool`Lags`.

<b>PowersOfB[n]</b>	gives a list of the powers of B up to n, with n a nonnegative integer and with B^0 = Identity
<b>PolynomialOf[x, a, n]</b>	gives the polynomial of x with coefficients a[i]. For symbol B, the first term is a[0] Identity instead of a[0]

- Create a lag structure with some given coefficients.

**PowersOfB[2]**

{Identity, B, B^2}

**{-0.5, 1.3, -0.4} . PowersOfB[2]**

$$-0.4 B^2 + 1.3 B - 0.5 \text{ Identity}$$

**%[x[t]]**

$$-0.4 x(t - 2) + 1.3 x(t - 1) - 0.5 x(t)$$

- Create an abstract polynomial in B of degree 4.

**PolynomialOf[B, a, 4]**

$$a(4) B^4 + a(3) B^3 + a(2) B^2 + a(1) B + \text{Identity } a(0)$$

- Create an abstract structure with 4 lags.

**PolynomialOf[B, a, 4][x[t]]**

$$a(4) x(t - 4) + a(3) x(t - 3) + a(2) x(t - 2) + a(1) x(t - 1) + a(0) x(t)$$

The following is an example of the determination of a general part of a solution.



- Regard a lag structure of degree 6, with the following coefficients.

```
pol = {1, 0, 3, 0, 3, 0, 1} . PowersOfB[6]
```

$$B^6 + 3 B^4 + 3 B^2 + \text{Identity}$$

- The characteristic equation is as follows.

```
chareq = 0 == pol /. Identity -> 1
```

$$0 == B^6 + 3 B^4 + 3 B^2 + 1$$

- The solution points can be found by Solve.

```
Solve[chareq, B]
```

```
{{B -> -i}, {B -> -i}, {B -> -i}, {B -> i}, {B -> i}, {B -> i}}
```

- The roots  $i$  and  $-i$  have multiplicity 3. In the general solution, their coefficients are polynomials of one degree less. With complex  $a$  and  $b$ :

```
x[t] == PolynomialOf[t, a, 3 - 1] I^t +  
        PolynomialOf[t, b, 3 - 1] (-I)^t
```

$$x(t) == i^t (a(2) t^2 + a(1) t + a(0)) + (-i)^t (b(2) t^2 + b(1) t + b(0))$$

- This implies the following real solution.

```
x[t] == PolynomialOf[t, c, 3 - 1] Cos[Pi/2 t] +  
        PolynomialOf[t, d, 3 - 1] Sin[Pi/2 t]
```

$$x(t) == (c(2) t^2 + c(1) t + c(0)) \cos\left(\frac{\pi t}{2}\right) + (d(2) t^2 + d(1) t + d(0)) \sin\left(\frac{\pi t}{2}\right)$$

#### 7.4.4 Transformation into systems of equations with single lags

As said, equations with more lags can be transformed into systems of equations with a single lag. In section 7.2 we discussed the package `OneLag`` that directly handles equations. It is also possible to construct matrices, and then analyse the system with matrix algebra. The following is an example of an error correction model where one variable is on a constant growth path. We only show the approach and don't go into the matrix algebra itself.

- Before we transform the equations, we use the undefined symbol  $\Delta$  to indicate the error correction structure.

```
 $\Delta[y] == b \Delta[x] + (1 - c) (x[t-1] - y[t-1]) + e[t]$ 
```

$$\Delta(y) == e(t) + (1 - c) (x(t - 1) - y(t - 1)) + b \Delta(x)$$

```
x[t] == (1 + g) x[t-1]
```

$$x(t) == (g + 1)x(t - 1)$$

- Develop the error correction into a proper statement.

```
Delta[y[t]] == b Delta[x[t]] + (1 - c) (x[t-1] - y[t-1]) + e[t]
```

$$y(t) - y(t - 1) == e(t) + b(x(t) - x(t - 1)) + (1 - c)(x(t - 1) - y(t - 1))$$

```
Solve[%, y[t]]
```

```
{y(t) -> e(t) - b x(t - 1) - c x(t - 1) + x(t - 1) + b x(t) + c y(t - 1)}
```

```
y[t] == Collect[{y[t] /. %[[1]]}, x[t-1]]
```

$$y(t) == e(t) + (-b - c + 1)x(t - 1) + b x(t) + c y(t - 1)$$

- If we drop the error term and take  $d = 1 - b - c$ , the system is.

```
mat = {{c, d}, {0, 1 + g}}
```

$$\begin{pmatrix} c & d \\ 0 & g + 1 \end{pmatrix}$$

```
{y[t], x[t]} == mat . {y[t-1], x[t-1]}
```

$$\{y(t), x(t)\} == \{d x(t - 1) + c y(t - 1), (g + 1)x(t - 1)\}$$

- Running it from the point  $\{y[0], x[0]\} = \{10, 1\}$ .

```
NSolveLinearDynamics[mat, {10, 1}, 3]
```

$$\begin{pmatrix} 10 & 1 \\ 10c + d & g + 1 \\ c(10c + d) + d(g + 1) & (g + 1)^2 \\ d(g + 1)^2 + c(c(10c + d) + d(g + 1)) & (g + 1)^3 \end{pmatrix}$$

#### 7.4.5 Comparing Stine's package and WRI's TSP

The following comparison of the two programs is not deep.

1. The prime comment is that both programs seem to provide the same solutions. One has to be alert on the input specifications though. For example, the TSP CovarianceFunction on an ARModel (up to lag h) gives the same result as Stine's ArimaCovariance (for h + 1), but one has to use h and h+1 respectively.
2. Though the specifications of Stine and TSP differ, the basics are the same: i.e. the coefficient lists {p1, ...} are given with the same signs, and the variance is given (and not the standard deviation).

3. Stine's package requires names for the Arima process and the noise, and this makes for less flexible programming.
4. Stine's canonical format already has the transfer function specified, while the TSP just contains the original lists of parameters. For programming, the TSP thus is more flexible.
5. The TSP can deal with vectors, while Stine's package has not been tried with those.
6. Stine's package contains some useful printing and plotting routines.
7. Neither `pack(age)` allows the full use of polynomial functionals. Thus, one cannot write  $(I + 2B - 5B^3)x[t]$ , and then have *Mathematica* find  $x[t] + 2x[t-1] - 5x[t-3]$ .

On balance, my tendency is to use the TSP, since it doesn't use names, doesn't require the transfer function, and can use vectors. Also, Stine's package is not really a package, but rather a notebook, and thus it does not create a proper context. Also, where it defines a B operator, then this interferes with the `Lags`` package. However, one may wish to transform results of the TSP to Stine's format, and then apply relevant procedures. This especially goes for printing and plotting. The `Arima`` package therefore contains a routine `ToStine`, that can be applied to TSP formats.

- This call should be sufficient to load both the TSP and Stine's package too. There will be some warning messages that we basically can neglect.

```
Economics[Arima]
```

- Stine's functions can be found by `?Global`*`` and in particular by:

```
?Define*
```

```
?Arima*
```

- The TSP functions can be found by:

```
Contents["TimeSeries`TimeSeries`"]
```

```
ToStine[model, y_Symbol, e_Symbol]
```

gives the Stine format of the model  $y$  with white noise  $e$

- The `ARModel[coefs, var]` function of the TSP gives the  $AR(p)$  model with  $p$  coefficients and `Normal[0, var]` noise.

```
xar = ARModel[{.5, -1.2}, sigma2]
```

```
ARModel({0.5, -1.2}, sigma2)
```

- Transforming to Stine's format.

```
ToStine[xar, yar, ers]
```

- The latter has defined yar and ers.

**yar && ers**

$$\text{ARIMA}\left(\frac{1}{1.2z^2 - 0.5z + 1}, 0, 0, \text{sigma2}, \text{yar}, \text{ers}\right) \bigwedge \text{ARIMA}(1, 0, 0, \text{sigma2}, \text{ers}, \text{WN})$$

Stine's representation of Arima objects is:

$$\text{ARIMA}[T(z) / P(z), d, \text{mean}, \text{scale}, \text{label}, \text{error label}]$$

The ratio  $T(z) / P(z)$  is the transfer function and  $d$  is the order of differencing, where it is assumed (and later enforced) that none of the zeros of  $P(z)$  are equal to 1. Following the usual convention, such zeros are incorporated into the difference value  $d$ . The scale is the error variance.

- From Stine's package.

**ArimaPrint[yar]**

$$\text{yar}_t = + \text{ers}_t - 1.2 \text{yar}_{t-2} + 0.5 \text{yar}_{t-1}$$

- `ARIMAModel[d, phi's, theta's, var]` specifies an  $\text{ARIMA}(p, d, q)$  model with  $p$  AR and  $q$  MA coefficients, and noise distributed as  $\text{Normal}[0, \text{var}]$ .

**xarima = ARIMAModel[1, {p1, p2}, {th1}, sig2]**

`ARIMAModel(1, {p1, p2}, {th1}, sig2)`

**ToStine[xarima, yam, es]**

- And the latter has defined:

**yam && es**

$$\text{ARIMA}\left(\frac{1 - \text{th1} z}{-p2 z^2 - p1 z + 1}, 1, 0, \text{sig2}, \text{yam}, \text{es}\right) \bigwedge \text{ARIMA}(1, 0, 0, \text{sig2}, \text{es}, \text{WN})$$

- Here,  $(1-B)y[t]$  does not evaluate to  $y[t] - y[t-1]$  since the Stine package has e.g. cleared the  $B$  from `Lags``.

**ArimaPrint[yam]**

$$(1 - B) \text{yam}_t = - \text{th1} \text{es}_{t-1} + \text{es}_t + (1 - B) p2 \text{yam}_{t-2} + (1 - B) p1 \text{yam}_{t-1}$$

## 7.5 Neural networks

---

### 7.5.1 Summary

The `Neural`` and `Freeman`Master`` packages only support the use of the neural network packages of J.A. Freeman (1994).

```
Economics[Neural]

Needs["Freeman`Master`"]
```

### 7.5.2 Introduction

Neural networks are alternative to common procedures. Rather than constructing a model with explicit parameters and estimate the latter on a set of data, you construct a network, and "train" it on those data. The theory and some of its applications are discussed in a clear and accessible manner by James A. Freeman, "Simulating neural networks with *Mathematica*", Addison Wesley, 1994. Freeman's packages can also be found on *MathSource*. The routines provided by the Economics Pack are only a few, and they concern very basic utilities. The following discussion also is short; a longer discussion is in the example notebooks.

### 7.5.3 Freeman's routines

- The following produces an overview of Freeman's routines (not evaluated here).

```
freeman = CasesMatch[$ContextPath, "Freeman*"]
Contents[freeman]
```

### 7.5.4 Logistic function

The Logistic function already is in the common packages. One may prefer this to defining a separate Sigmoid function.

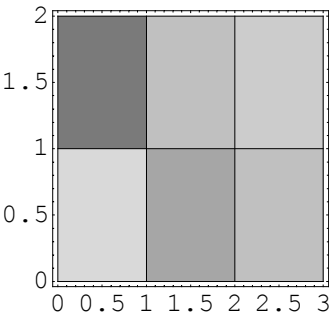
### 7.5.5 Hinton diagram

```
HintonDiagram[x, p:Automatic, opts]
```

makes a Hinton diagram. The `p` parameter allows a partition first

The options are passed on to `ListDensityPlot`. The `PlotRange` option affects the degree of shading.

```
HintonDiagram[{{-.4, 5, 6.},{7, 3, 5}}, Automatic, PlotRange -> {-10, 10}]
```



7.5.6 Data

ToFreemansDataFormat [data]

transforms data  
{outputvar -> {out1, out2, ...}, inputvar1 -> {in11, ....}, inputvar2 -> .....}  
into the format required for the Freeman packages

- For a perceptron for a decision surface for 'AND', let x1 and x2 provide various combinations of true and false, and y = x1 AND x2 be the result. We use these AND data to test Xor, and can calculate the error measure.

```
ToFreemansDataFormat[{y -> {1,0,0,0}, x1 -> {1,1,0,0}, x2 -> {1,0,1,0}} ]
```

$$\begin{pmatrix} \{1, 1\} & \{1\} \\ \{1, 0\} & \{0\} \\ \{0, 1\} & \{0\} \\ \{0, 0\} & \{0\} \end{pmatrix}$$

```
testXor[%, { .5, 1.}]
```

Inputs =  $\begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$

Outputs = {1.5, 0.5, 1., 0.}

Errors =  $\begin{pmatrix} -0.5 \\ -0.5 \\ -1. \\ 0. \end{pmatrix}$

Mean squared error = 0.375

7.5.7 Input facility for .1 and .9

Typing and reading .1 and .9 may cause more errors than LL and GG.

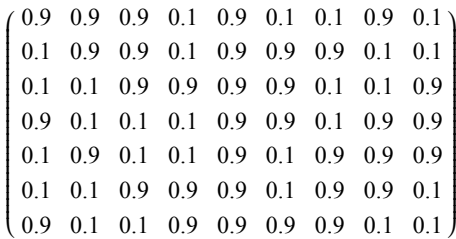
LL	LL = .1; for readability and error prevention
GG	GG = .9; for readability and error prevention

7.5.8 Pattern recognition

Freeman uses an example of pattern recognition, where a list with 9 elements represents a twodimensional image. Turning and plotting the image comes in handy.

TurnList9[x_List]	takes a list with 9 elements, regards this as a 3 X3 block with the middle as the center, and turns the elements one step to the right
TurnList9[x_List, n]	is the same as Nest[TurnList9, x, n]
CList	Representation of C as a list
TList	Representation of T as a list

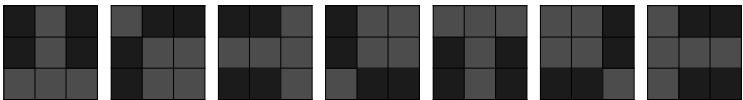
```
ts = NestList[TurnList9, TList, 6]
```



```
tsta = Map[ListDensityPlotSquared[#, Frame -> False,  
DisplayFunction -> Identity]& , ts]
```

```
GraphicsArray[tsta]
```

```
Show[%, DisplayFunction -> $DisplayFunction]
```



# 7.6 Genetic programming

## 7.6.1 Summary

The `GenePro`` package provides routines `Genesis` and `Evolve` for a population of mathematical expressions. `Evolve` calls a fitness test routine to establish the fitness of the individuals, and this criterion can be basically everything. The `GenePro`Estimation`` package provides a fitness criterion based on the sum of squared errors.

```
Economics[GenePro]

Economics[GenePro`Estimation]
```

## 7.6.2 Introduction

Genetic Programming (GP) is another relatively new modeling technique. The modeler provides rules for the creation, survival and procreation of objects, and then lets the program run its course. Essentially the process of evolution is mimicked. The result that is generated is 'fit' according to the rules. This mimicking of the evolutionary process is so strong, in fact, that it has appeared useful to adopt similar names for the routines, like `Population`, `Evolve[ ]` etcetera.

Population	a list of individuals. The first generation is generated by <code>Genesis</code> , the other generations are generated by <code>Evolve</code> .
------------	---

The combination of GP and *Mathematica* is especially interesting, since *Mathematica* is so flexible in the manipulation of symbols.

The first fruitful application is estimation. The goal is to find the equation that minimizes the sum or squared errors (SSE) of  $y = f[x, \dots] + \text{error}$ . This gives us a clear criterion for the 'fitness' of an individual of the population, namely  $1 / \text{SSE}$ . We shall discuss GP using this problem as illustration. The package `GenePro`Estimation`` has been written for this problem, and the options of the main GP engine `Evolve[ ]` are set so that this routine is used by default. Note though that `Evolve[ ]` can be set to call other fitness criteria too, and the user may want to supply these.

Another promising application area, not implemented here, concerns the evolution of society. Richard Gaylord & Louis D'Andria, "Simulating society", Springer-Telos 1998, provide a *Mathematica* toolkit to study the evolutionary behaviour of society. Their approach currently is rather numerical, and can be extended with crossbreeding processes as discussed below.



7.6.3 Genesis

Genesis[ <i>opts</i> ]	creates a new Population from scratch, using both random depths and relationships
Birth[ <i>n</i> ]	creates a new random individual with n levels
Birth[]	gives an individual with random depth

Note: Birth[] relies on internal settings created by Genesis, and hence Genesis must be called at least once for changes to take effect. The reason is that Birth can be called hundreds of times, and hence it is more efficient to exclude all kinds of calls and checks.

Birth relies on Genesis. The latter has a whole set of options.

Options[Genesis]

$\left\{ \text{Functions} \rightarrow \left( \begin{matrix} \text{Plus} & 2 \\ \text{Times} & 2 \end{matrix} \right), \text{MaximumLevel} \rightarrow 3, \right.$   
 $\left. \text{NumberOfIndividuals} \rightarrow 5, \text{Seed} \rightarrow \{\}, \text{TestPrintQ} \rightarrow \text{False}, \text{Variables} \rightarrow \{\text{one}, \text{two}\} \right\}$

<i>option</i>	<i>typical default value</i>	
NumberOfIndividuals	5	the number to be created
MaximumLevel	3	maximum depth per individual
Functions	$\{\{\text{Plus}, 2\}, \{\text{Times}, 2\}\}$	list of functions and their number of arguments that can be used in the creation of an individual
Variables	$\{\text{one}, \text{two}\}$	symbols
Seed	$\{\}$	force the initial population to include these individuals
PrintTestQ	False	whether or not to print individuals with their number

Note: MaximumLevel gives the maximum level. The actual level is determined at birth.

Genesis[]

$\{\text{two}^{2\text{one}}, \text{one two}, \text{one}(\text{one} + \text{two}), \text{two}^{\text{one two}}, \text{one} + 2\text{two}\}$

Birth[5]

$4\text{one}^{\text{one two}}$

Let us now regard a particular estimation problem, namely the estimation of  $x = g[y, z] + e$ . It is advisable to have non-integer data. Otherwise the complexity of some random combinations may result into long computing times. The estimation routine sets data to real numbers, but we can also enter the data as reals.

- The following provides the data.

```
dat = {x → {10,2,3,56,1,7},
      y → {4,5,6,2,1,6}, z → {4,5,3,2,4,8}} // N;
```

The relationship  $g[ ]$  is assumed to consist of normal operations. We can enter both {Plus, 2} and {Plus, 4}, but not {Power, 3}. To enter constants into the equation, we can (1) use a Hold construction, or (2) use a 'data' vector of constant values, or (3) define new functions with constant values.

- Define some constant functions.

```
con1[x_] := .1
con5[] := .5
```

- Create an initial population.

```
Genesis[MaximumLevel → 5,
  Functions → {{Divide, 2}, {Times, 2}, {Minus, 1}, {Plus, 2}, {Plus,
4}, {Power, 2}, {con5, 0}},
  Seed → {},
  Variables → {y, z},
  NumberOfIndividuals → 6, TestPrintQ → True];
```

1:  $4^y$

2: 0.5

3: 0.5

4: 1

5:  $0.5 z$

6:  $\frac{z^2}{y}$

**Population**

$\{4^y, 0.5, 0.5, 1, 0.5 z, \frac{z^2}{y}\}$

### 7.6.4 Estimation fitness criterion

<code>Estimation[<i>individual</i>]</code>	is a fitness test for estimation, and gives {hit, score} = {indicator, sse}
<code>RuleToR2[<i>rules</i>] or RuleToR2[{<i>rules</i>}]</code>	
	gives the R2 statistics associated with the rule(s) rhs → sse. Coefficients that occur in different places are counted as one for the degrees of freedom.

Note: The indicator is 1 iff the endogenous variable occurs in the rhs. The population is normally created with the lhs excluded. If the lhs is included, then the risk exists that lhs = lhs is the best estimate.

- Note that TestPrintQ here affects the printing of Estimation.

`SetOptions[Estimation, TestPrintQ → True, Data → dat ]`

{Data → {x → {10., 2., 3., 56., 1., 7.}, y → {4., 5., 6., 2., 1., 6.}, z → {4., 5., 3., 2., 4., 8.}},  
EvaluationPerTest → 1, ExeQTime → 0.01, NumberOfTests → 1, TestPrintQ → True}

NumberOfTests gives the number of tests per individual. If that number is 10 and the NumberOfIndividuals → 200, then there are 2000 tests per generation. The current Estimation routine does only one test per individual, namely the computation of the SSE. EvaluationPerTest gives the number of evaluations per test call. ExeQTime gives an estimate of the execution time of a single evaluation. The subroutine Duration, that is called by Evolve, uses these settings, and the numbers of the generations and individuals in the population, to estimate the number of minutes for the whole evolution. While a single estimate might not be accurate, changes in the settings have a systematic effect.

### 7.6.5 Evolution

<code>Evolve[<i>opts</i>]</code>	takes output of Genesis, and evolves
----------------------------------	--------------------------------------

Evolve also has a set of options.

`Options[Evolve]`

{FitnessTest → Null, GenerationQ → True,  
Method → Random, NumberOfGenerations → 2, TestPrintQ → False}

<i>option</i>	<i>typical default value</i>	
FitnessTest	Null	the routine that computes Fitness
Method	Random	selection for cross–breeding
GenerationQ	True	store Generation[i]
NumberOfGenerations	2	normally a serious number (50)
PrintTestQ	False	for prints of ApplyTest

Note: The other Method value is Mean, meaning that better than average are selected.

- Note that the detail printing is determined by Estimation and its option settings, while Evolve has only control over its own printing.

```
res = Evolve[FitnessTest → Estimation,  
NumberOfGenerations → 2, TestPrintQ → False]
```

```
Estimation prints the sum of squared errors.  
If the lhs is on the rhs, then 'hit' indicates 1.  
  
Estimated minutes for evolution: 0.204472  
  
>>>>>>>> Generation: 1  
  
Test: 3.45792×107  
  
Test: 3221.5  
  
Test: 3221.5  
  
Test: 3147.  
  
Test: 3101.5  
  
Test: 3201.69  
  
>>> Report:  
  
Minimum score: 3101.5  
  
Average score: 5.76585×106
```

```
>>>>>>>>> Generation: 2

NumberOfIndividuals::chng : Number of individuals changed from 6 to 7

Test: 3101.5

Test: 3221.5

Test: 3147.

Test: 3233.27

Test: 6779.

Test: 20919.

Test: 3215.93

>>> Report:

Minimum score: 3101.5

Average score: 6231.03

***** Result *****

CPU minutes used: 0.0136667

Fittest are/is: individual -> test score

{0.5z → 3101.5}
```

7.6.6 Show and analyse results

- The current population contains 'look-alikes' as a result of cross-breeding and fitness selection.

Population

$$\left\{0.5z, 0.5, 1, \frac{1}{y}, z^2, 2z^2, \frac{z^{\frac{1}{y}}}{y}\right\}$$

Score	list of the scores of each individual
Fitness	transform $1 / (1 + \text{Score})$ , and then normalized so that the sum of all fitnesses is 1
Normalize[Score]	transforms Score to Fitness
Death	the score that causes an exit. Default is $10^{10}$ , but can be redefined in the FitnessTest option routine.

- A lower SSE score means a higher fitness.

**Score**

```
{3101.5, 3221.5, 3147., 3233.27, 6779., 20919., 3215.93}
```

```
pos = Position[Score, Min[Score]]
```

```
(1)
```

**Fitness**

```
{0.182552, 0.175754, 0.179913, 0.175114, 0.083535, 0.027073, 0.176059}
```

```
Position[Fitness, Max[Fitness]]
```

```
(1)
```

- This is a check on the current criterion of the sum of squared errors.

```
Add[(x - Population[[ pos[[1,1]] 1] ) /. dat // N]^2]
```

```
3101.5
```

- This is provided by *Mathematica*'s standard statistical packages.

```
DispersionReport[Score]
```

```
{Variance →  $4.37461 \times 10^7$ , StandardDeviation → 6614.08, SampleRange → 17817.5,  
MeanDeviation → 4353.13, MedianDeviation → 74.5, QuartileDeviation → 1364.17}
```

- This procedure has been provided by *GenePro`Estimation`*.

```
RuleToR2[res]
```

```
{{AdjustedRSquared → 0.257222, Correlation → -0.50717, DegreesOfFreedom → 5,  
EstimatedVariance → 620.3, FitResiduals → {8., -0.5, 1.5, 55., -1., 3.}, Function → 0.5 z,  
NumberOfObservations → 6, PredictedResponse → {2., 2.5, 1.5, 1., 2., 4.}, RSquared → 0.0598666}}
```

### 7.6.7 Subroutines

Each separate kind of problem requires its own fitness test. That test must produce a score and a hit variable. A low score means a fit individual. The hit is a printing indicator only. The fitness test routine is indicated by the option of *Evolve*, and *Evolve* calls *ApplyTest* to evaluate the fitness routine.

`ApplyTest [Population, PrintQ]`

calls the fitness test for each member of the population and fills the Score array. PrintQ is a boolean value for printing.

For efficiency reasons, some of the major routines produce output in the `GenePro`` context, and not just in their last line, while others can get their input also from that context and not just from the input line. For example, `ToNewGeneration` has only the population as direct input. Other information, like `Fitness` and `Mutate`, is called from the context.

Mutation is a bit contra-Darwinian. It is commonly thought that mutation has less chance of continuation in more homogenous populations. In our case, we then want to force mutations, since we are not interested in copies of the same formula.

`ToNewGeneration [Population, forceEqualFitnessQ:False]`

makes a new generation from the current population

`Mutate [Population]`

forces a new Birth if one type of individual has more than 30 % of the population. If there has been a population size overflow, then multiplicity is also used for weeding.

Note: `forceEqualFitnessQ` is here for testing. If it is `True`, then the `Fitness` array is reset to (a priori) equal fitness.

`CrossBreed [individual1, individual2]`

creates two new individuals by switching random parts between old ones

`CrossBreed [{i1, i2}]`

does the same (allowing for `NestList`).

`ChoosePointPr`

affects Birth and CrossBreed

Basically, each individual is a formula consisting of elements at a certain position. (See `PartInspect`.) A random generator selects points in those formulas, and adds or switches the elements. The package has two different strategies for that random generator, and allows a random mix. If `ChoosePointPr` = 1, then all parts of a formula have equal probability, not just the elements but also the combinations that occur in the formula. If `ChoosePointPr` = 0, then the random generator cruises from the bottom to the highest level, and thereby gives more probability to the lowest elements. It appears in practice that the first strategy gives more variety while the second strategy gives more stability. For that reason, `ChoosePointPr` has been set at the default value of .5. If for an individual a random draw  $\leq$  `ChoosePointPr`, then the first strategy is selected, and otherwise the second.

```
CrossBreed[Population[[1]], Population[[2]]]
{0.5 z, 0.25}
```

## 7.6.8 Notes

### 7.6.8.1 Testing

Note that you can call various routines in a simple manner once Genesis and Evolve have run. Genesis and Evolve create settings in the GenePro` context that other routines rely on.

- Even though Log[ ] has not been defined in Functions, you could use it in a test call like this.

```
Estimation[z Log[x]]
Test: 2376.39
{1, 2376.39}
```

### 7.6.8.2 Use old results

If Evolve has stopped, and you wish to continue with the last population, you basically can do so. You may also clean up that population first by applying Union, and add new individuals or adapt the NumberOfIndividuals. If you don't add new individuals, then you might as well do a cross-over first, since the old population has already been tested on the fitness values.

- Thus for example:

```
Population = ToNewGeneration[Union[Population], True]
 $\left\{ \frac{z^{\frac{1}{y}}}{y}, 1, 0.5, 0.25, 0.5 z, 1, 0.5 \right\}$ 
SetOptions[Genesis, NumberOfIndividuals →
  Length[Population]];
```

### 7.6.8.3 Another formulation of constants

You can enter constants also as variables with a constant value.



```

Genesis[MaximumLevel → 5,
  Functions → {{Divide, 2}, {Times, 2}, {Minus, 1}, {Plus, 2},
{Power, 2} },
  Variables → {y, z, c1, c2, c3},
  Seed → {(c3 (c1 + y))(-y)},
  NumberOfIndividuals → 3,
  TestPrintQ → True];

1: (c3 (c1 + y))-y

2: c2-c1 c3

3: c2 yc1

newdat = dat ~Join~ {c1 → {1,1,1,1,1,1}*.75,
  c2 → {1,1,1,1,1,1}* 15., c3 → {1,1,1,1,1,1}*.1};

SetOptions[Estimation, Data → newdat];

```

- Not evaluated here.

```

res = Evolve[FitnessTest → Estimation,
  NumberOfGenerations → 2, TestPrintQ → False]

```

#### 7.6.8.4 On performance

The package has not been used for actual estimation yet. It has been developed because of the wish to understand the subject and from the notion that such a package could be useful someday. It seems to perform fairly well, but this actually would require corroboration. Some notes on performance are:

- Tests on identical individuals can be eliminated by using Union[Population]. This would speed up computations when a generations gets caught in little variation (say, constants 4 and 5 that cannot crossover since they don't have parts). However, then the probability of selection might need to be adjusted with the count for that type of individual.
- Other statistics on the dispersion in the population can be of value.
- The package doesn't yet use logical analysis and pattern recognition to get rid of silly individuals.
- Perhaps we should seed with a linear regression outcome.

## 7.7 Operations research

---

### 7.7.1 Summary

This section gives an introduction into the operations research sections, and provides basic routines for scheduling, inventory management and one time capacity decision making.

**Economics [Operations]**

**Economics [Economic`Graphics]**

### 7.7.2 Introduction

Operations research (OR) aspires at better control of operations by using mathematical and statistical modeling. Another term for the field is Management Science (MS). Key examples of the field are linear programming and queueing theory. Another department is logistics, which concerns operations for getting the products in the right quantities and right qualities to the right place at the right time. Our reference is Hillier and Lieberman, "Introduction to operations research", Holden Day 1967/1972. An accessible intermediate text is given by Krajewski and Ritzman, "Operations Management", Addison-Wesley 1996. A text for logistics is Ballou, "Business logistics management", Prentice Hall 1992.

We are in the fortunate situation that standard *Mathematica* already helps in doing OR. *Mathematica* supplies routines in linear programming, provides statistical packages that one can use for testing - in quality management for example - and it is relatively easy to punch in a formula and derive results - e.g. for the Economic Order Quantity in inventory policy. Also, application packages are available, like the `DiscreteMath`Combinatorica`` package by Steve Skiena for graph theory for networks. And when we review the whole Economics Pack, we will see various topics, like decision theory and acceptance sampling, that are relevant for OR too.

We however still can add to these features and make OR even more accessible. The next two sections shall deal with linear programming and queueing separately. This current section covers some smaller topics that don't require such prominence: scheduling and Gantt charts, inventory policy and one time optimal capacity decisions.

### 7.7.3 Gantt charts

Gantt charts are used to display the sequence of interdependent activities. For example, various products may use the same machines but in different intensity and different order. The chart gives a graphical map of the intended schedule.

The data consist of lists of recipes. A recipe = {productlabel, {machine<sub>i</sub>, begin<sub>i</sub>, end<sub>i</sub>}, {machine<sub>j</sub>, begin<sub>j</sub>, end<sub>j</sub>}, ...}. The product label cannot be a number, and preferably is a string, while the machine labels must be integers. Recipes can be of different length, where one would leave out machines that are not

used. The machines can also be mentioned in arbitrary order, though one might prefer the time sequence of the activities.

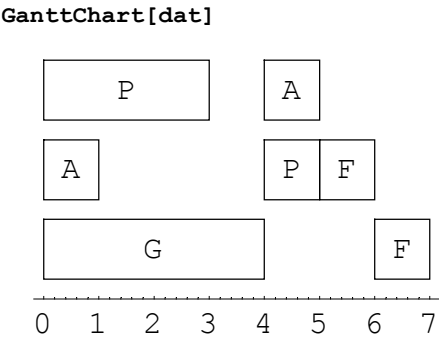
`GanttChart[data, opts]`

gives a Gantt chart of activities on various machines (rows) for periods (columns)

There are 'plot' and 'table' formats. In the plot format it is possible to choose the beginning of an activity as equal to the end of the former activity, while these values can be non-integer.

■ An example plot.

```
dat = {{ "P", {1, 0, 3}, {2, 4, 5}},
      { "A", {2, 0, 1}, {1, 4, 5}},
      { "F", {2, 5, 6}, {3, 6, 7} },
      { "G", {3, 0, 4}}};
```



An alternative format is the tabular one. The beginnings and endings of activities must now be nonnegative integers, and they may not be equal.

`GanttChart[TableForm, data, h:1]`

from plot to table format

`GanttChart[Plot, data, h:1]`

from table to plot format

Note: The change involves adding or subtracting h units to or from the x-axis co-ordinates.

```
dattable = GanttChart[TableForm, dat]

{{P, {1, 1, 3}, {2, 5, 5}}, {A, {2, 1, 1}, {1, 5, 5}}, {F, {2, 6, 6}, {3, 7, 7}}, {G, {3, 1, 4}}}
```

`GanttChart[Table, dattable]`

$$\begin{pmatrix} P & P & P & . & A & . & . \\ A & . & . & . & P & F & . \\ G & G & G & G & . & . & F \end{pmatrix}$$

7.7.4 Johnson's rule and Gantt charts

The GanttChart routine only displays a schedule. The schedule is not necessarily optimal, in the sense of using the least time ("makespan") for a given set of products. When there are only two machines, then Johnson's rule gives a direct solution for that optimum.

JohnsonsRule[Sort, ps]

reorders ps into the best schedule

Here *ps* is a list of pairs {{t1, t2}, ...}. Assumed is that: (a) All products are to be handled first on machine 1 and then on machine 2. (b) For product *i*, the pair {t1*i*, t2*i*} gives the time t1*i* on machine 1 and the time t2*i* on machine.

- Assume that product 1 takes 12 minutes on machine 1, and 22 minutes on machine 2. Product 2 takes 4 minutes on machine 1, and 5 minutes on 2. Etcetera.

```
ps = {{12, 22}, {4, 5}, {5, 3}, {15, 16}, {10, 8}};
```

```
JohnsonsRule[Sort, ps]
```

$$\begin{pmatrix} 4 & 5 \\ 12 & 22 \\ 15 & 16 \\ 10 & 8 \\ 5 & 3 \end{pmatrix}$$

- Results[ JohnsonsRule, Order] contains the permutation.

```
Results[JohnsonsRule, Order]
```

```
{2, 1, 4, 5, 3}
```

The following allows you to plot or tabulate the optimal solution.

JohnsonsRule[ps, labels]

produces the data that  
can be put into GanttChart[data]

JohnsonsRule[Table, ps, labels]

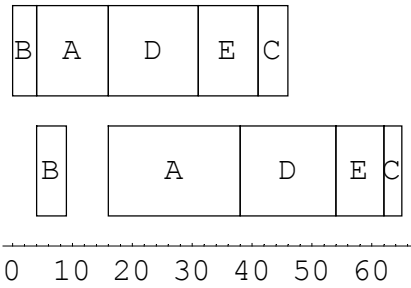
produces the data that can be put into GanttChart[Table, data]

Note: In both cases, labels is a list of preferably strings.

```
labs = CharacterRange["A", "E"]
```

```
{A, B, C, D, E}
```

**GanttChart[JohnsonsRule[ps, labs]]**



The routine also provides some statistics. The job flow time is the cumulative time from beginning to the end of an activity, including waiting as 'work in progress' but excluding 'waiting as finished product'. When averaging over more activities, it is presumed that time measurement starts at the same moment for all, so that many activities may start with waiting. Indeed, it appears conceptually useful to relate some statistics to queueing theory. The MakeSpan is the total duration (at its minimised value), and with  $n$  products the production rate is  $\lambda = n / \text{MakeSpan}$ . The average JobFlowTime is the mean process time  $W$ . Hence the work in progress (WIP) is the number of units in the process,  $\text{WIP} = L = \lambda W = \text{JobFlowTime} * \text{NumberOfActivities} / \text{MakeSpan}$ .

■ Results[JohnsonsRule, MakeSpan] gives the statistics.

**Results[JohnsonsRule, MakeSpan]**

$$\left\{ \begin{array}{l} \text{MakeSpan} \rightarrow 65, \text{Slack} \rightarrow \{19, 11\}, \text{JobFlowTime(Average)} \rightarrow \frac{228}{5}, \\ \text{WorkInProgress(Average)} \rightarrow \frac{228}{65}, \text{Data} \rightarrow \left\{ \begin{array}{ccccc} \{0, 4\} & \{4, 16\} & \{16, 31\} & \{31, 41\} & \{41, 46\} \\ \{4, 9\} & \{16, 38\} & \{38, 54\} & \{54, 62\} & \{62, 65\} \end{array} \right\} \end{array} \right\}$$

7.7.5 Inventory basics

For inventory policy, we may start with a set of undefined symbols that are useful as indicators. OrderCost indicates the total costs incurred by ordering. StorageCost indicates the total costs incurred by storage The TimeBetweenOrders indicates the time between orders. The OrderLeadTime indicates the time required for an order to materialise in the arrival of the products.

Backorder	OrderCost	SafetyStock
MaxInventory	OrderLeadTime	StorageCost
NumberOfOrders	OrderQuantity	TimeBetweenOrders

Some short routines express some basic relations that will be useful throughout.

```
InventoryPolicyCost[n, c, a, s]
```

sums order and storage costs, with  $n$  = number of orders,  
 $c$  = cost per order (or setup costs),  
 $a$  = average inventory and  $s$  = storage costs per unit average inventory

```
Turnover[d, a, p:l]      for d = total demand, a = average inventory and p = period
```

```
InventoryPolicyCost[n, c, a, s]
```

```
{OrderCost → c n, StorageCost → a s, InventoryPolicyCost → c n + a s}
```

```
Turnover[d, a]
```

```
{AverageInventory → a, Turnover →  $\frac{d}{a}$ , NPeriodsOfSupply →  $\frac{a}{d}$ }
```

Note that the `Economic`Common`` package contains an `InventoryModel`, and that this becomes available when loading the `Operations`` package.

### 7.7.6 Economic Order Quantity

The formula for what now is called the "Economic Order Quantity" is attributed to Harris 1913 and Camp 1929. It assumes known demand  $d$ , order costs  $c$  and storage costs  $s$ . For arbitrary order quantity  $q$ , the number of orders is  $d/q$  and the average inventory level is  $q/2$ , so that total costs are  $c(d/q) + s(q/2)$ . The optimal order quantity follows from the first order condition of a cost minimum.

```
foc = D[c (d / q) + s (q / 2), q] == 0
```

$$\frac{s}{2} - \frac{cd}{q^2} == 0$$

```
Solve[foc, q]
```

$$\left\{ \left\{ q \rightarrow -\frac{\sqrt{2} \sqrt{c} \sqrt{d}}{\sqrt{s}} \right\}, \left\{ q \rightarrow \frac{\sqrt{2} \sqrt{c} \sqrt{d}}{\sqrt{s}} \right\} \right\}$$

The `EOQ[ ]` routine provides the nonnegative solution, and fills `Results[EOQ]` with the implied properties for turnover etcetera. A variant of the model allows backorders with cost  $bc$ , and if  $bc \rightarrow \text{Infinity}$ , then this variant becomes the EOQ again.

`EOQ[d, c, s]` gives the economic order quantity, for  $d$  = demand,  
 $c$  = cost per order,  $s$  = storage costs per unit

`EOQ[Plot, d, c, s, {qmin, qmax}]`

gives a plot of the order and storage costs

`EOQ[d, c, s, bc]` for backorders with per time unit cost  $bc$

- This gives the formal result.

`EOQ[d, c, s]`

$$\sqrt{2} \sqrt{\frac{cd}{s}}$$

- We don't evaluate `Results[EOQ]` since this is an exercise in printing square roots. If you do evaluate it, however, you might note that order and storage costs are equal, each just half of total costs. (Which is another way to determine the quantity.)

`Results[EOQ]`

- A numerical example is the following. We now evaluate `Results[EOQ]` so that you can see what output is available.

`EOQ[1000, 20, 2] // N`

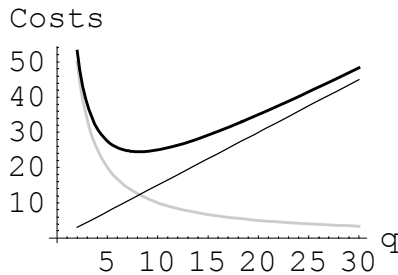
141.421

`Results[EOQ] // N`

{OrderCost → 141.421, StorageCost → 141.421, InventoryPolicyCost → 282.843,  
 AverageInventory → 70.7107, Turnover → 14.1421, NPeriodsOfSupply → 0.0707107,  
 TimeBetweenOrders → 0.141421, NumberOfOrders → 7.07107, OrderQuantity → 141.421}

- Note that the time parameters of the EOQ[ ] routine are defined in terms of the demand period. It is often conceptually better to introduce an explicit number of periods. Instead of demand over a year, we would get demand over 50 weeks. The time between orders then can become a whole integer instead of a small fraction. See the discussion in section 7.7.8.1.
- This is a typical EOQ plot, with storage costs rising linearly with  $q$  and order costs decreasing nonlinearly with  $q$ . The minimum of total costs is at the intersection of both subcosts.

```
EOQ[Plot, 100, 1, 3, {2, 30}]
```



- Backorders reduce storage costs but add their own costs.

```
EOQ[d, c, s, b]
```

$$\sqrt{2} \sqrt{\frac{cd(\frac{s}{b} + 1)}{s}}$$

7.7.7 Economic Batch Quantity

The Economic Batch Quantity (EBQ), also called the Economic production Lot Size (ELS), arises when the quantity is not ordered but produced, so that both demand and supply are continuous. The demand speed  $d$  will be  $r$  times the production speed  $p$ , i.e.  $d = r p$ , normally  $0 < r < 1$ . It appears that the  $EBQ = EOQ[d, c, s(1 - r)]$ . There is a difference in the other statistics, though. While average inventory still is half of maximal inventory, this maximal inventory no longer is equal to the batch size  $q$  (order size) but it is only its fraction  $(1 - r) q$ .

$EBQ[d, c, s, r]$	gives the economic batch quantity when the commodity is not ordered but produced at rate $d / r$
-------------------	--

```
EBQ[d, c, s, r]
```

$$\sqrt{2} \sqrt{\frac{cd}{(1 - r)s}}$$



- Let us compare this with  $EOQ[1000, 20, 2]$  in the above. Let us also double storage costs to make things comparable. Note the result that the average inventory level is being halved.

```
EBQ[1000, 20, 2/r, r] /. r -> 1/2 // N
```

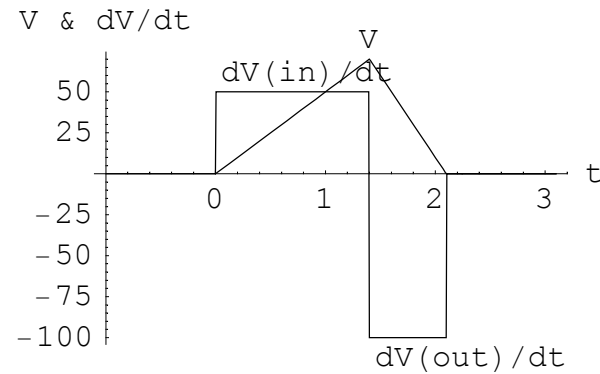
```
141.421
```

```
Results[EBQ] // N
```

```
{ProductionTimePerCycle -> 0.0707107, MaxInventory -> 70.7107,
 OrderCost -> 141.421, StorageCost -> 141.421, InventoryPolicyCost -> 282.843,
 AverageInventory -> 35.3553, Turnover -> 28.2843, NPeriodsOfSupply -> 0.0353553,
 TimeBetweenOrders -> 0.141421, NumberOfOrders -> 7.07107, OrderQuantity -> 141.421}
```

- A dynamic plot can be made with the function `SawTooth` of section A.10.2
- The following is just for one cycle. Regard a storage place  $V_{max}$  that has an inflow and outflow. Let inventory be  $V$ , its change  $dV/dt$ . If net inflow is  $p - d$  then its duration is  $V_{max} / (p - d)$ , and if net outflow is  $d$  then its duration is  $V_{max} / d$ . In above example,  $d = 1000$  and  $p = d / r = 2000$ , so net inflow is first 1000 and net outflow then is 1000. The plot for these parameters is not so neat because of the size of the parameter values. To get a useful plot, we make net inflow 50 and net outflow 100. Note in this plot that an inflow of 50 per time unit precisely gives  $V = 50$  at  $t = 1$ .

```
VdVdtPlot[70, 50, 100]
```



### 7.7.8 Inventory control: P and Q-systems

The EOQ only gives an optimal order quantity (under specific conditions). The next question is how inventories should be monitored. We recognise two approaches here:

- The P-system uses periodic reviews, so that the TimeBetweenOrders (TBO) has a fixed value, and the amount actually ordered will depend upon the TargetInventory.
- The Q-system tracks the remaining inventory and triggers a fixed order at a ReOrderPoint (ROP).

To discuss these systems, we first set some useful parameter values based on a solution of the EOQ model. The list of parameters R will contain algebraic symbols, and the list NR will contain numerical values.

7.7.8.1 Period parameters

- Let us assume that a year has 50 weeks, and that our periodic review requires an integer number of weeks between orders.

```
EOQ[d, c, s]; R = Results[EOQ];

TBO = Floor[periods TimeBetweenOrders /. R]


$$\left\lfloor \frac{\sqrt{2} \text{ periods } \sqrt{\frac{c d}{s}}}{d} \right\rfloor$$


EOQ[1000, 20, 2]; NR = Results[EOQ]//N; NPeriods = 50;

NTBO = Floor[NPeriods TimeBetweenOrders /. NR]

7
```

7.7.8.2 P-System

TargetInventory[f, parms, TBO, OLT, CSL:0.95]

for a periodic (P) inventory control system,  
using the distribution f[parms] for demand, the Time Between Orders TBO,  
the Order Lead Time OLT, and the Cycle ServiceLevel (default 95 %)

Note: Only implemented for f = Normal, and parms = {μ, σ}.

Note: Results[TargetInventory] give the average demand during TBO + OLT and the safety stock.

- This would give a formal expression (not evaluated here).
- This is a numerical result for a μ = 1000 / 50 = 20 and σ = 5, a TBO = 7, an OLT = 3, and a cycle service level of 85%.

```
TargetInventory[Normal, {d / periods, σ}, TBO, OLT]

TargetInventory[Normal, {1000 / NPeriods, 5}, NTBO, 3, .85]

216.387
```

**Results [TargetInventory]**

{TimeBetweenOrders → 7, AverageDemandDuringTBO+OLT → 200,  
SafetyStock → 16.3875, TargetInventory → 216.387}

**7.7.8.3 Q-System**

ReOrderPoint [*f*, *parms*, *OLT*, *CSL:0.95*]

for a continuous (Q) inventory control system,  
using distribution *f*[*parms*] for demand, the Order Lead Time *OLT*,  
and the Cycle ServiceLevel (default 95 %)

Note: Only implemented for *f* = Normal, and *parms* = {*μ*, *σ*}.

Note: Results[ReOrderPoint] give the average demand during OLT and the safety stock.

- The formal result is more tractable.

**ReOrderPoint**[Normal, {*d* / periods, *σ*}, OLT]

$$\frac{d \text{ OLT}}{\text{periods}} + 1.64485 \sigma \sqrt{\text{OLT}}$$

- The ROP for the same settings as the TI above.

**ReOrderPoint**[Normal, {1000/ NPeriods, 5}, 3, .85]

68.9758

**Results [ReOrderPoint]**

{OrderLeadTime → 3, AverageDemandDuringOLT → 60,  
SafetyStock → 8.97578, ReOrderPoint → 68.9758}

**7.7.8.4 P and Q-systems jointly analysed**

The P and Q-systems both assume independent demand. We can jointly analyse both systems given the characteristics of demand and proper values for the order lead time and cycle service level.

IDInventory [*EOQ*, {*d*, *c*, *s*}, *f*, *parms*, *OLT*, *CSL:0.95*]

performs the steps of EOQ, TI and ROP

Note: ID stands for independent demand here. Only implemented for *f* = Normal, and *parms* = {*μ*, *σ*}.

This routine determines  $q = \text{EOQ}[d, c, s]$ , and solves the P-system with a Time Between Orders  $\text{TBO} = \text{Floor}[q / \mu]$ . Note that  $\sigma$  is measured for the periodic demand that has an average  $\mu$ , for  $d / \mu$  periods. The inconsistency of fixed total demand  $d$  and probabilistic periodic demand is overlooked.

- This gives a lot of square roots and Floors (not evaluated here).

```
IDInventory[EOQ, {d, c, s}, Normal, {μ, σ}, OLT]
```

- This reproduces the above, in one call. Note that some output options are Strings.

```
IDInventory[EOQ, {1000., 20, 2}, Normal, {20, 5}, 3, .85]
```

```
{Periods → 50., EOQ → {OrderCost → 141.421, StorageCost → 141.421, InventoryPolicyCost → 282.843,
  AverageInventory → 70.7107, Turnover → 14.1421, NPeriodsOfSupply → 0.0707107,
  TimeBetweenOrders → 0.141421, NumberOfOrders → 7.07107, OrderQuantity → 141.421},
Q-System → {AverageDemandDuringOLT → 60, AverageInventory → 139.686,
  InventoryPolicyCost → 420.794, NPeriodsOfSupply → 6.98432, OrderCost → 141.421,
  OrderLeadTime → 3, OrderQuantity → 141.421, ReOrderPoint → 68.9758, SafetyStock → 8.97578,
  StorageCost → 279.373, TargetInventory → 210.397, Turnover → 7.15889,
  NumberOfOrders(Average) → 7.07107, TimeBetweenOrders(Average) → 7.07107},
P-System → {AverageDemandDuringTBO+OLT → 200, AverageInventory → 146.387,
  InventoryPolicyCost → 435.632, NPeriodsOfSupply → 7.31937, NumberOfOrders → 7.14286,
  OrderCost → 142.857, SafetyStock → 16.3875, StorageCost → 292.775, TargetInventory → 216.387,
  TimeBetweenOrders → 7, Turnover → 6.83119, OrderQuantity(Average) → 140.}}
```

### 7.7.9 Piecewise discontinuities

Paul Rubin, "Optimizing with piecewise smooth functions", in Varian (1996), discusses, among other topics, the discontinuities that arise when order lot sizes differ because of all unit discounts with breakpoints. Krajewski & Ritzman (1996), one of the volumes that I use for teaching, give examples in Supplement H. Rubin develops some routines (in particular PWS, IA, IB) to deal with these discontinuities. My preference however is to stay close to the *Mathematica* System, since this warrants the use by more people and likely, evolutionary, the robustness of the methods. One alternative is to use the Interval function, as is done with NDomain and EPS. Another alternative is the use of Piecewise. A general discussion can be found in appendix A.11 below. For the discussion in the remainder of this current subsection I thank Paul Rubin for his permission to use his example and some of his ideas.

Let us regard the total cost that consists of purchase cost  $p[q] d$  and inventory policy cost  $c n + a s$  (see the discussion in section 7.7.5). Demand is  $d$ , and the unit price  $p[q]$  depends upon the order lot size  $q$ . The number of orders is  $n = d / q$ , the average inventory level is  $a = q / 2$ , and storage costs are a fixed unit cost  $s_0$  and a part that depends upon the rate of interest  $r$ .

```
total = p[q] d + c n + a s /. {n → d/q, a → q/2, s → s0 + r p[q]};
```

- To simplify the discussion we set some less relevant variables at numeric values, and then define the total cost function. Note that  $p$  is a function here.

```
tc[q_, p_] = total /. {d → 12500, c → 125, s0 → .36, r → .07} //
Simplify
```

$$0.18 q + (0.035 q + 12500.) p(q) + \frac{1.5625 \times 10^6}{q}$$

Let *lis* contain the "domain and price" data {upper limit  $i$ , price  $i$ , ...}. We then use the function `ToPiecewise` (section A.11) to reconstruct Rubin's price example.

```
lis = {1, 100, 600, 35.5, 1000, 33.72, 2000, 31.45, True, 28.5};
```

```
price[q_] = ToPiecewise[q < # &, lis]
```

```
Which[q < 1, 100, q < 600, 35.5, q < 1000, 33.72, q < 2000, 31.45, True, 28.5]
```

- Applying this to the total cost function gives us a function of  $q$  only.

```
tc[q, price]
```

```
0.18 q + (0.035 q + 12500.)
```

```
Which[q < 1, 100, q < 600, 35.5, q < 1000, 33.72, q < 2000, 31.45, True, 28.5] + \frac{1.5625 \times 10^6}{q}
```

*Mathematica* handles the latter expression nicely for many kinds of operations.

- This will for example plot the function. (Not evaluated.)

```
Plot[tc[q, price], {q, 0, 4000}]
```

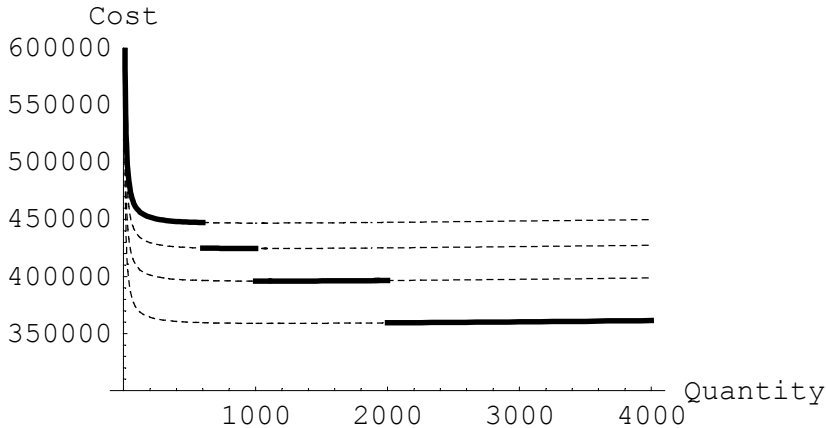
However, if you want to present the (rather commonly used) plot where the price ranges are thick sections of otherwise dashed lines, then some extra effort is required. We follow Rubin here again.

- First it is useful to recover the ranges and prices from *lis*.

```
{up, ps} = Transpose[Partition[lis, 2]];
```

- Note that we use the '&' to enter a function with a fixed value.

```
Show[Table[Plot @@ {tc[q, ps[[i+1]]&,
  {q, up[[j]], up[[j+1]]} /. True -> 4000,
  DisplayFunction -> Identity,
  PlotRange -> {300000, 600000},
  PlotStyle -> If[ i == j, Thickness[0.01], Dashing[{0.01}]]],
  {i, 4}, {j, 4}], DisplayFunction -> $DisplayFunction,
  AxesLabel -> {"Quantity", "Cost"}, TextStyle -> {FontSize -> 11}];
```



When desiring to minimise cost, we find that FindMinimum cannot handle such discontinuities.

- A solution is to call FindMinimum for each price range, and then select the overall minimum.

```
Module[{m, minima, pos, fm},
  fm = FindMinimum[tc[q, price], {q, # + 10, # + 100}]& /@ Drop[up,
-1];
  minima = First /@ fm; m = Min[minima];
  pos = Position[minima, m]; Extract[fm, pos]
(359386. {q -> 2000.03})
```

### 7.7.10 One time optimal capacity

There are also decisions that have a one time character, like concerning special editions of newspapers that are of no value the next day.

### 7.7.9.1 From probability

The following is from Ballou, p418. The assumption is that expected marginal profits at the optimal point  $Q$  are zero. At the margin there are two possible events: selling the product or not selling it. Let  $Pr$  stand for the cumulative probability of selling at least  $Q$ , but not selling that marginal unit anymore (since that is the point where losses really start), then expected marginal profits would be  $(1 - Pr) \text{Profit(selling)} + Pr \text{Profit(not selling)} = 0$ . We now can write:

- $\text{Profit(selling)} = \pi = p - c$  = price minus marginal costs. Note that this profit forgone can also be classified as the unit cost of undercapacity (at this price  $p$ ).
- $\text{Profit(not selling)} = -\gamma$  where  $\gamma = c - s$  = marginal loss when not selling = marginal costs minus salvage value.

The condition for the optimum then can be read as  $(1 - Pr) \pi = Pr \gamma$ , meaning that the expected marginal profit on the next unit sold equals the expected marginal loss of not selling the unit. This solves as  $Pr^* = \pi / (\pi + \gamma)$ . The optimal capacity level  $Q$  then can be determined by setting the cumulative demand probability equal to this probability. Thus Solve[  $Pr^* = Pr[q < Q]$ ,  $Q$ ].

`PrDemandLessThanLevel[ $cu, co$ ]`

gives  $Pr[q < Q]$  or the cumulative probability up to optimal capacity level  $Q$ , as a function of the unit costs of undercapacity and overcapacity

`PrDemandLessThanLevel[ $\pi, \gamma$ ]`

$$\frac{\pi}{\gamma + \pi}$$

`PrDemandLessThanLevel[ $\pi, \gamma, \text{Constant}, \{min, max\}$ ]`

gives  $\{Pr, Q\}$  for the uniform distribution over  $\{min, max\}$

`PrDemandLessThanLevel[ $\pi, \gamma, \text{Normal}, \{mu, sigma\}$ ]`

gives  $\{Pr, Q\}$  for the normal distribution

`PrDemandLessThanLevel[ $\pi, \gamma, PM, \{pr\_List, st\_List\}$ ]`

gives  $\{Pr, Q\}$  for a discrete probability measure  $pr$  over states  $st$

Note: The NewsBoyProblem[] gives a more robust result for the last case.

`PrDemandLessThanLevel[40, 10, Constant, {600, 1400}]`

$$\left\{ \frac{4}{5}, 1240 \right\}$$

```
PrDemandLessThanLevel[40, 10, Normal, {1000, 133.}]
```

$$\left\{\frac{4}{5}, 1111.94\right\}$$

### 7.7.9.2 The newspaper boy's problem

Regard a newsboy who has to buy  $q$  newspapers at cost  $c$  per paper, and who sells at price  $p$  per paper. Additional unit costs are  $\psi$  for overcapacity (demand  $< q$ ) and  $\phi$  for undercapacity (demand  $> q$ ). If he can assign probabilities  $pm$  to the various states of demand, then we can take the payoff table format and MinimalDue routine from the Decision` package.

```
NewsBoyProblem[p, c,  $\phi$ ,  $\psi$ , {pm_List, states_List}]
```

gives the minimal expected loss strategy for a newsboy

```
NewsBoyProblem[Set, p, c,  $\phi$ ,  $\psi$ ]
```

sets the Options[Aggregator] for this problem. See Profit[] and Cost[]

- The profit relation can be found as follows.

```
NewsBoyProblem[Set, p, c,  $\phi$ ,  $\psi$ ] /. {Demand[] → d, Quantity[] → q}
```

$$-cq - \phi \text{Max}[0, d - q] - \psi \text{Max}[0, q - d] + p \text{Min}[d, q]$$

- Let the boy sell 10 till 50 papers with these probabilities.

```
lis = {{.2, .3, .3, .1, .1}, {10, 20, 30, 40, 50}};
```

Let us assume that the papers sell at a dime, cost 4 cents, have an opportunity cost of 1 cent for unsatisfied customers, and cost 2 cents to dump.

- The minimal expected loss approach tells the newspaper boy to buy 30 papers, to expect sales of 26 and to expect a profit of 93 cents. Note that positive values are for nature, so negative values are positive income for the newsboy.

```
NewsBoyProblem[10, 4, 1, 2, lis]
```

```
{NumberOfStates → 5, NumberOfActions → 5,
```

$$\text{PayOff} \rightarrow \begin{pmatrix} -60 & 0 & 60 & 120 & 180 \\ -50 & -120 & -60 & 0 & 60 \\ -40 & -110 & -180 & -120 & -60 \\ -30 & -100 & -170 & -240 & -180 \\ -20 & -90 & -160 & -230 & -300 \end{pmatrix}, \text{Expectation} \rightarrow \{-44., -88., -93., -59., -12.\},$$

```
Position → (3), Min → -93., Demand(Mean) → 26., Quantity() → 30}
```



- Let us compare this result with the approach in the former paragraph. Using the assumption of zero expected marginal profits, the boy would buy 20 papers.

`PrDemandLessThanLevel[10-4-1, 4+2, PM, lis]`

$$\left\{ \frac{5}{11}, 20 \right\}$$

- The profit made in this case can be read from the second column of the `NewsboyProblem[ ]` output. Expected profits are 88 cents.
- PM. Note that if you take the `Profit[ ]` function defined by `NewsboyProblem[ ]` and then substitute the expected demand level, then you would get a statistically misleading result. Proper is `Exp[Profit[d,q]]` and not `Profit[Exp[d], q]`.

`Profit[]`

$$-\text{Max}[0, \text{Demand}() - \text{Quantity}] - \\ 2 \text{Max}[0, \text{Quantity}() - \text{Demand}()] + 10 \text{Min}[\text{Demand}(), \text{Quantity}()] - 4 \text{Quantity}()$$

`Profit[] /. {Demand[] → 26, Quantity[] → 20}`

## 7.8 Linear programming

---

### 7.8.1 Summary

These packages extend *Mathematica*'s LP routines with shadow prices, ranges of feasibility and optimality, printing of tableaus, and the plotting in the 2D projection plane. Also implemented is the transport problem.

#### Economics [LP]

Note: The `LP`` package does not contain routines itself but is a master package for `LP`NumLP``, `LP`Transport``, `LP`Common``, and `LP`Carter``.

### 7.8.2 Introduction

*Mathematica* already supplies routines for linear programming and constrained optimisation. However, these routines only put out the solution point and solution value. The `LP`` package extends on this with:

- the shadow prices and the ranges of feasibility and optimality
- plotting in the 2D projection plane, assuming optimality for the other variables
- implementation of the specific transport problem.

The routines are only shells around *Mathematica*'s linear programming routines. A consequence is that the routines are numerical and not symbolic. The routine names therefor are `NumLP` and `NumLPAnalysis`, for Numeric-LP (with NLP for Nonlinear Programming).

This contrasts with the symbolic routines by Michael Carter, "Linear programming with *Mathematica*" (two chapters), in Varian (1996). Carter's routine is called `LP`, and given the symbolic nature of *Mathematica*, he has right of way. The symbolic approach is very powerful since it gives more freedom for tackling non-standard problems. The numerical approach however allows for quicker computation of the standard cases.

Michael Carter recently gave me permission to rework his package and to include it here. This has become `LP`Carter``. I thank him for this, and users will surely benefit. Carter's discussion referred to above still remains the main text on the package. The improvements here are not in the algorithm but in ease of use, error handling and embedding.

### 7.8.3 Numerical LP analysis

In the following  $c$  is a  $n$ -vector,  $A$  a  $\{m, n\}$  matrix, and  $b$  a  $m$ -vector, and all are nonnegative. The problem is to find the vector  $x$  which maximises the quantity  $c.x$  subject to the constraints  $A.x \leq b$  and  $x \geq 0$ .

NumLPAnalysis[c, A, b]

calls NumLP, SimplexTableau,  
RangeOfFeasibility and RangeOfOptimality

Note: Here *c* is a *n*-vector, *A* a *m* × *n* matrix, and *b* a *m*-vector. The problem is to find the vector *x* which maximises the quantity *c*·*x* subject to the constraints *A*·*x* ≤ *b* and *x* ≥ 0. This thus has the conventional LP format (as compared to *Mathematica*'s LinearProgramming routine that gives the dual formulation).

Though the routine can handle more dimensions, let us regard the 2D example from Hillier and Lieberman, p139. The tableau is printed, the other is normal output.

```
c = {3, 5};
A = {{1, 0}, {0, 1}, {3, 2}} ;
b = {4, 6, 18};
```

NumLPAnalysis[c, A, b]

	1	1	1	0	0	sol
obj	0	0	0	3	1	36
1	1	0	0	$-\frac{2}{3}$	$\frac{1}{3}$	2
2	0	1	0	1	0	6
3	0	0	1	$\frac{2}{3}$	$-\frac{1}{3}$	2

{Solve → 36, Variables → {2, 6}, Slack → {2, 0, 0}, Basis → {1, 2, 3},

ShadowPrices → {0, 3, 1}, RangeOfFeasibility →  $\begin{pmatrix} 2 & \infty \\ 3 & 9 \\ 12 & 24 \end{pmatrix}$ , RangeOfOptimality →  $\begin{pmatrix} 0 & \frac{15}{2} \\ 2 & \infty \end{pmatrix}$ }

The first column of the Tableau contains the numbers of the basic variables. The first row of the Tableau prints 1 for basic variables and 0 for the non-basic variables. The top right hand value is the total value score. The second row contains the net values of the objective function, with negative values indicating inoptimality. The last column gives the values of the basic variables, with negative values indicating inoptimality.

7.8.4 Separate routines

NumLP[c, A, b]

finds the vector *x* which maximises the  
quantity *c*·*x* subject to the constraints *A*·*x* ≤ *b* and *x* ≥ 0

NumLP[c, A, b]

{Solve → 36, Variables → {2, 6}, Slack → {2, 0, 0}, Basis → {1, 2, 3}, ShadowPrices → {0, 3, 1}}

<code>RangeOfFeasibility[]</code>	gives the ranges over which the resources can vary while their shadow price remains the same. The information of the last call of <code>NumLP[...]</code> is used
<code>RangeOfFeasibility[i, v, invB]</code>	gives the relative range of feasibility of the <i>i</i> th constraint using vector <i>v</i> with the values of the basic variables and the matrix <i>invB</i>
<code>RangeOfFeasibility[v, invB]</code>	gives all relative ranges of feasibility

Note: The latter only hold for the optimum.

**RangeOfFeasibility[]**

$$\begin{pmatrix} 2 & \infty \\ 3 & 9 \\ 12 & 24 \end{pmatrix}$$

<code>RangeOfOptimality[]</code>	gives the ranges for which each coefficient in the objective function can change without affecting the solution, given the other coefficient values. The information of the last call of <code>NumLP[...]</code> is used
<code>RangeOfOptimality[c, A, basis]</code>	does this for the given basis
<code>RangeOfOptimality["2D", r1, {c1, c2}, r2]</code>	gives the ranges for a 2 D problem

**RangeOfOptimality[]**

$$\begin{pmatrix} 0 & \frac{15}{2} \\ 2 & \infty \end{pmatrix}$$

The 2D problem is often used to explain the notion of the range of optimality. The "2D" routine application solves the relationship  $r1 \leq -c1' / c2' \leq r2$  for alternatively  $c2' = c2$  (giving the range for  $c1'$ ) and  $c1' = c1$  (giving the range for  $c2'$ ). Here  $-c1 / c2$  is the slope of the objective function. The coefficients must stay in the deduced ranges if the current solution is to remain optimal. The deduction assumes  $c1, c2 \geq 0$ .

```
SimplexTableau[c, A, b, basis]

prints the Simplex Tableau for max c.x subject to A.x ≤ b and x ≥ 0
```

The list of basic variables must be entered as the column numbers in (A I), with as many numbers as there are rows in A. Next to printing the tableau, the routine also produces some results in Results[SimplexTableau]. As the routine has been called as a subroutine above, we can look at this.

- Results[SimplexTableau] tells you about entering and leaving variables.

```
Results[SimplexTableau]

{Value → 36, Basis → {1, 2, 3}, BasisValues → {2, 6, 2}, FirstRow → {0, 0, 0, 3, 1, 36},
  EnteringBasicVariable → None, LeavingBasicVariable → None}
```

- Results[SimplexTableau, Matrix] gives matrices that allow you to check how the shadowprices have been calculated.

```
("cB" . Inverse) /. Results[SimplexTableau, Matrix]

{0, 3, 1}
```

Basis[A, basis]	selects the columns in A for the given basis, see NumLP
Slack	symbol only
ShadowPrices[c, A, basis]	gives the implied shadowprices associated with the selected basis, see NumLP

7.8.5 Unbounded solutions and degeneracy

- NumLP recognises unbounded solutions.

```
NumLP[{1, 1}, {{1, -1}}, {10}]

ConstrainedMin::nbdd: Specified domain appears unbounded.

Dot::mindet: Input matrix contains an indeterminate entry.

NumLP::deg : Degenerate around basis variables {}

{Solve → Indeterminate, Variables → {Indeterminate, Indeterminate},
  Slack → {Indeterminate}, Basis → {1}, ShadowPrices → {1}}
```

- NumLP can spot degeneracy (when one or more constraints can be relaxed without altering the solution).

```
NumLP[{1, 0, 0},
      {{1, 0, 0}, {1, 1, 0}, {1, 1, 1}},
      {1, 1, 1}]
```

*NumLP::deg : Degenerate around basis variables {1}*

{Solve → 1, Variables → {1, 0, 0}, Slack → {0, 0, 0}, Basis → {1, 2, 3}, ShadowPrices → {1, 0, 0}}

### 7.8.6 2D plotting and projecting

The routine `LP2DPlot` is in the common `Graphics`` package (see the appendix A.10.4). Any problem of a higher dimension can be projected in the 2D plane, and this is done by the following routines.

```
NumLPPlot[c, A, b, {i:1, j:2}, opts]
```

determines  $c'$ ,  $A'$  and  $b'$  using `ProjectNumLP[c, A, b, {i, j}]`,  
and evaluates `LP2DPlot[-c', -A', -b', opts]`

```
ProjectNumLP[c, A, b, {i, j}]
```

determines parameters  $c'$ ,  $A'$  and  $b'$  for problem `NumLP[c', A', b']` for integers  $i$  and  $j$ ,  
assuming that the other variables are at the optimum of `NumLP[c, A, b]`

```
ProjectLP[c, A, b, {i, j}, point]
```

determines parameters  $c'$ ,  $A'$  and  $b'$  for integers  $i$  and  $j$ , assuming the point

- Projection is straightforward.

```
ProjectLP[{c1, c2, c3}, {{a1, a2, a3}, {a4, a5, a6}}, {b1, b2}, {1, 2},
{x1, x2, x3}]
```

$\left\{ \text{Function} \rightarrow \{c1, c2\}, \text{Matrix} \rightarrow \begin{pmatrix} a1 & a2 \\ a4 & a5 \end{pmatrix}, \text{Constraint} \rightarrow \{b1 - a3 x3, b2 - a6 x3\} \right\}$

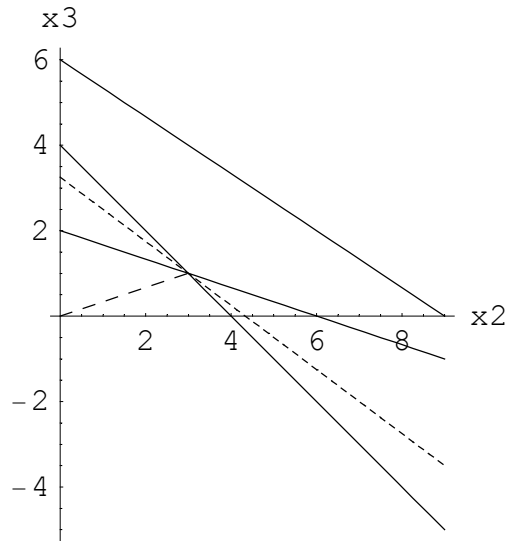
- For a particular problem.

```
c = {3, 3, 4, 1};
A = {{2, 1, 1, 1}, {1, 1, 3, 2}, {3, 2, 3, 4}};
b = {4, 6, 18};
```

```
NumLP[c, A, b]
```

$\left\{ \text{Solve} \rightarrow 13, \text{Variables} \rightarrow \{0, 3, 1, 0\}, \text{Slack} \rightarrow \{0, 0, 9\}, \text{Basis} \rightarrow \{2, 3, 7\}, \text{ShadowPrices} \rightarrow \left\{ \frac{5}{2}, \frac{1}{2}, 0 \right\} \right\}$

```
NumLPPlot[c, A, b, {2, 3}, AxesLabel → {"x2", "x3"}];
```



### 7.8.7 Boundaries

Sometimes, the domains of the variables are determined by lists of simple inequalities. These can be solved with existing packages. The package below is also loaded when you load `LP``.

```
Needs["Algebra`InequalitySolve`"]

InequalitySolve[And[ -29 ≤ x < 10, 100 > -x > -5 ], x]

-29 ≤ x < 5
```

### 7.8.8 The transport problem

The transport problem is to allocate goods that can come from  $m$  origins and that can go to  $n$  destinations. For example, there are  $m$  factories and  $n$  warehouses (that represent demand in the various cities). There are transport costs and capacity and demand constraints to take account of. Allocations  $X = \{x_{ij}\}$  must satisfy row sums  $X.1 = a$  (capacity) and column sums  $1'.X = b$  (demand), while the optimand is minimal cost  $1'.(C * X).1$ . In case of overcapacity, add a dummy column with zero costs; and with undercapacity, a dummy row. The following routine also requires you to supply two undefined symbolic variables for the shadow price analysis.

<code>TransportNumLP[C, b, a, u_Symbol, v_Symbol]</code>	solves the transport problem
--	------------------------------

```
TransportNumLP[{{1, 2}, {3, 2}, {3, 1}},
               {30, 60},
               {30, 40, 20}, u, v]
```

```
Solve::svars :
  Equations may not give solutions for all "solve" variables.
```

$$\begin{aligned} \{ \text{Min} \rightarrow 130, \text{Solve} \rightarrow \begin{pmatrix} 30 & 0 \\ 0 & 40 \\ 0 & 20 \end{pmatrix}, \text{CostMatrix} \rightarrow \begin{pmatrix} 30 & 0 \\ 0 & 80 \\ 0 & 20 \end{pmatrix}, \\ \text{Cost} \rightarrow 130, \text{Chosen}[cij=(ui+vj)] \rightarrow \begin{matrix} & & 0 & 1-u3 \\ & 1 & 0 & . \\ u3+1 & . & 0 & . \\ u3 & . & 0 & . \end{matrix}, \\ \text{NotChosen}[cij-(ui+vj)] \rightarrow \begin{matrix} & 0 & 1-u3 \\ & . & u3 \\ u3+1 & 2-u3 & . \\ u3 & 3-u3 & . \end{matrix} \} \end{aligned}$$

7.8.9 Transport problem subroutines

7.8.9.1 Shadow prices

In the shadow price analysis, the  $u_i$  (rows) and  $v_j$  (columns) are taken such that  $c_{ij} = u_i + v_j$  for the allocated or chosen  $x_{ij} > 0$ . Applying these values to the non-chosen  $x_{ij} = 0$ , should give nonnegative  $c_{ij} - (u_i + v_j)$ . Negative cells indicate a non-optimal solution. Sometimes more than one solution is possible, and Solve balks. This actually happened in the problem above; check there though that we may substitute  $u_3 = 0$  so that above solution is optimal.



`TransportNumLPShadowPrices[C, b, a, u_Symbol, v_Symbol, X]`

is a subroutine of `TransportNumLP`, and the input `a`, `b`, `C` and `X` are explained there

- This shows that the current allocation is non-optimal.

`TransportNumLPShadowPrices[{{1, 2}, {3, 2}, {3, 1}},  
{30, 60}, {30, 40, 20}, u, v,  
{{0, 30}, {30, 10}, {0, 20}}]`

$$\{\text{CostMatrix} \rightarrow \begin{pmatrix} 0 & 60 \\ 90 & 20 \\ 0 & 20 \end{pmatrix}, \text{Cost} \rightarrow 190,$$

$$\text{Chosen}[cij=(ui+v_j)] \rightarrow \begin{matrix} & 0 & -1 \\ 3 & . & 0 \\ 3 & 0 & 0 \\ 2 & . & 0 \end{matrix}, \text{NotChosen}[cij-(ui+v_j)] \rightarrow \begin{matrix} & 0 & -1 \\ 3 & -2 & . \\ 3 & . & . \\ 2 & 1 & . \end{matrix}$$

### 7.8.9.2 Input checks

Note that you can get the row and column sums `{a, b}` with common routine `BorderTotals`.

`BorderTotals[{{0, 30}, {30, 10}, {0, 20}}]`  
`{{30, 40, 20}, {30, 60}}`

The following helps checking on input values.

`TransportNumLPMatrixQ[x, b, a, test:False]`

tests the matrix `x`, column sums `b` and row sums `a` for the transport problem

If `x` is the allocation matrix `X`, set the test at `Plus` or `Print`. `Results[TransportLPMatrixQ]` then contains the differences between the true column and row sums and `a` and `b`. If test is `Print`, then this is printed. If `x` is the cost matrix `C`, set test at any other value (default). Output is `True` if the test passes, `False` otherwise.

### 7.8.9.3 Vogels approximation method

With these solution methods available, Vogel's approximation method (VAM) to find an initial allocation is hardly relevant any more. But it is useful in education to help students become more familiar with the transport problem and the meaning of the concepts involved.

<code>Vogel [ C , b , a ]</code>	applies Vogel's approximation method to the transport problem
<code>Vogel [ x_List ]</code>	gives the Vogel penalty value of that list, i.e. the difference from the lowest value to the next value

- In above example, VAM generates also the solution point.

```
Vogel[{ {1, 2}, {3, 2}, {3, 1}},  
      {30, 60}, {30, 40, 20}]
```

$$\begin{pmatrix} 30 & 0 \\ 0 & 40 \\ 0 & 20 \end{pmatrix}$$

- Results[Vogel] show how the successive allocations have been. In the Matrix output option the cells are {step, assigned}. In the Rims output option each row presents a step, and a row consists of the column and row rims at that step. A rim gives the Vogel penalties taken on the cost matrix. A value of -1 stands for an eliminated row or column.

```
Results[Vogel]
```

$$\left\{ \text{Matrix} \rightarrow \begin{pmatrix} \{1, 30\} & 0 \\ 0 & \{2, 40\} \\ 0 & \{3, 20\} \end{pmatrix}, \text{Rims} \rightarrow \begin{pmatrix} \{2, 1\} & \{1, 1, 2\} \\ \{-1, 1\} & \{-1, 2, 1\} \\ \{-1, 1\} & \{-1, -1, 1\} \end{pmatrix} \right\}$$

7.8.10 Carter's symbolic LP

As said in the introduction, there is also a symbolic approach to LP, thanks to Carter.

- The following would give you an overview of the contents of the package. There is now no name shadowing and no problem with \$PrePrint. (Not evaluated.)

```
Economics[LP`Carter]
```

7.8.10.1 Setting up a problem

One simplification has been that the lists of decision variables and slack variables are always recorded in Options[Problem]. At the beginning you set the problem by Problem[tableau] or Problem[objective, constraints] or by matrices (see below), and then the options are set, and the Problem[ ] object is defined that will be the default for Simplex[ ] and LP[ ]. The latter generate Solution[ ], that is default for ShadowPrices[ ] and SensitivityAnalysis.

<code>Problem[ tableau_?EquationListQ]</code>	sets the <code>Problem[]</code> and fills the <code>Options[Problem]</code> with the lists of decision variables and slacks
<code>Problem[objective, constraints_List]</code>	also constructs the tableau
<code>Problem[SetOptions]</code>	if <code>Problem[]</code> already exists.
<code>Problem[]</code>	is the current problem, used as default for many LP routines
<code>Solution[]</code>	is the default tableau of ShadowPrice(s) and SensitivityAnalysis

Note: Problem uses NonAttributedTerms to identify the variables. Doing this only once will speed up all later routines. Slack is introduced as Slack[label], where the label is determined by the options of ToEquation, by default two random letters. The identifier for the objective is Objective[Max] since Simplex and LP only maximise. The storage of the objective function in the Options[Problem] is crucial for later reference by ShadowPrices.

- You can enter equations using terms like `x[1]`, `x[2]`, or simple symbols as `x`, `y`, `z`. The following is an example of an unbounded problem.

```

Problem[x + y, {x - y ≤ 10}]

{Variables → {x, y}, Slack → {Slack(WQ)}, Objective → x + y}

Simplex[]

LP::unb : The problem is unbounded

{Objective(Max) == x + y, Slack(WQ) == -x + y + 10}

```

Input can also be generated by matrices.

<code>LPMatricesToEquations[c, A, b, {xlabels___Symbol}, {slabls___Symbol}]</code>	transforms the NumLP[c, A, b] problem into the format used by M. Carter 1996, using the symbol Objective for the objective function, X for the proper variables, Slack for the slack variables, xlabels for indexing x and slabls for indexing s. If the latter are empty, then numbers are taken
<code>X[label]</code>	is the symbol representation of the level of output or consumption of commodity label
<code>EquationsToLPMatrices[eqs_List]</code>	transforms the LP equations (slack==..) back into matrix format
<code>LinearEquationQ[eq    {eqs}]</code>	gives True if the input gives all linear equations

Note: The latter uses LinearEquationsToMatrices[] of LinearAlgebra`MatrixManipulation`, and can use CheckLinearity → True. Note that Problem[] does not check on linearity.

- Carter uses the following problem.

```
c = {3, 1, 3};
A = {{2, 2, 1}, {1, 2, 3}, {2, 1, 1}};
b = {30, 25, 20};
```

- The canonical form uses X and Slack headers.

```
LPMatricesToEquations[c, A, b, {bookcases, chairs, desks}, {finishing,
labour, machining}] // TableForm
```

```
Objective(Max) == 3 X(bookcases) + X(chairs) + 3 X(desks)
Slack(finishing) == -2 X(bookcases) - 2 X(chairs) - X(desks) + 30
Slack(labour) == -X(bookcases) - 2 X(chairs) - 3 X(desks) + 25
Slack(machining) == -2 X(bookcases) - X(chairs) - X(desks) + 20
```

- Above also set the options.

```
Options[Problem]
```

```
{Variables → {X(bookcases), X(chairs), X(desks)},
Slack → {Slack(finishing), Slack(labour), Slack(machining)},
Objective → 3 X(bookcases) + X(chairs) + 3 X(desks)}
```

- If you would wish to return to matrices:

```
EquationsToLPMatrices[Problem[]]
```

$$\left\{ \text{Objective} \rightarrow \{3, 1, 3\}, \text{Matrix} \rightarrow \begin{pmatrix} 2 & 2 & 1 \\ 1 & 2 & 3 \\ 2 & 1 & 1 \end{pmatrix}, \text{Constraints} \rightarrow \{30, 25, 20\} \right\}$$

### 7.8.10.2 Solving it

LP calls Simplex twice, in order to find a suitable starting point.

<code>LP [ ]</code>	solves <code>Problem[ ]</code>
<code>LP [tableau]</code>	does the same to a problem in tableau form
<code>LP [objective, constraints]</code>	maximizes objective subject to the constraints using the two phase simplex algorithm
<code>Simplex[same formats]</code>	maximizes objective subject to the constraints (one phase)
<code>CurrentPoint [tableau]</code>	sets the nonbasic variables of the tableau to zero

- Let us first check on the value of the unsolved problem.

`CurrentPoint[Problem[]]`

{Objective(Max) → 0, Slack(finishing) → 30, Slack(labour) → 25,  
Slack(machining) → 20,  $X(\text{bookcases}) \rightarrow 0$ ,  $X(\text{chairs}) \rightarrow 0$ ,  $X(\text{desks}) \rightarrow 0$ }

- Let us solve it with Carter's robust method.

`LP[] // TableForm`

Objective(Max) ==  $-\frac{3 \text{Slack}(\text{labour})}{5} - \frac{6 \text{Slack}(\text{machining})}{5} - \frac{7 X(\text{chairs})}{5} + 39$   
Slack(finishing) ==  $\text{Slack}(\text{machining}) - X(\text{chairs}) + 10$   
 $X(\text{desks}) == -\frac{2 \text{Slack}(\text{labour})}{5} + \frac{\text{Slack}(\text{machining})}{5} - \frac{3 X(\text{chairs})}{5} + 6$   
 $X(\text{bookcases}) == \frac{\text{Slack}(\text{labour})}{5} - \frac{3 \text{Slack}(\text{machining})}{5} - \frac{X(\text{chairs})}{5} + 7$

- And then look at the solution (also identifiable in above table).

`CurrentPoint[Solution[]]`

{Objective(Max) → 39, Slack(finishing) → 10,  $X(\text{desks}) \rightarrow 6$ ,  
 $X(\text{bookcases}) \rightarrow 7$ , Slack(labour) → 0, Slack(machining) → 0,  $X(\text{chairs}) \rightarrow 0$ }

Another approach is `FinalTableau`, that is a shell around `ConstrainedMax` in a similar fashion as `NumLP`.

<code>FinalTableau [objective, constraints]</code>	computes the final tableau when maximizing objective subject to constraints. Uses <code>ConstrainedMax</code>
--	---

- `FinalTableau` sets `Problem[ ]` and `Solution[ ]`. (Not evaluated.)

`FinalTableau[x + y, {0.5 x + 0.3 y ≤ 9, x - 5 y ≥ 10}]`

7.8.10.3 Sensitivity analysis

SensitivityAnalysis relies on properly defined Options[Problem], and its default input is Solution[ ].

SensitivityAnalysis[tableau]	gives the ranges of validity for the prices of the decision variables and the shadow prices of the resources, for a final tableau
SensitivityAnalysis[ShadowPrices (, tableau)]	performs a sensitivity analysis with respect to the constraints
SensitivityAnalysis[Price (, tableau)]	does the same for the prices of the decision variables

- The ranges of optimality for the prices of the decision variables and the ranges of feasibility of the constraints (of validity of the shadow prices) are in delta format.

SensitivityAnalysis[]

$X(\text{bookcases})$	3	$-\infty \leq \text{Delta}(\text{bookcases}) \leq \infty$
$X(\text{chairs})$	1	$-\infty \leq \text{Delta}(\text{chairs}) \leq \frac{7}{5}$
$X(\text{desks})$	3	$-\infty \leq \text{Delta}(\text{desks}) \leq \infty$
Slack(finishing)	0	$-10 \leq \text{Delta}(\text{finishing}) \leq \infty$
Slack(labour)	$\frac{3}{5}$	$-15 \leq \text{Delta}(\text{labour}) \leq 35$
Slack(machining)	$\frac{6}{5}$	$-\frac{35}{3} \leq \text{Delta}(\text{machining}) \leq 10$

7.8.10.4 Subroutines

BasicVariables[tableau]	lists the basic variables in tableau.
NonBasicVariables[tableau]	lists the non–basic variables in tableau

Default tableau is Solution[] while the list of all variables is taken from Options[Problem].

- For the Solution[ ] generated above.

BasicVariables[]

{Slack(finishing),  $X(\text{desks})$ ,  $X(\text{bookcases})$ }

NonBasicVariables[]

{Slack(labour), Slack(machining),  $X(\text{chairs})$ }

While the following speak for themselves.

<code>Pivot [(<i>tableau</i>)]</code>	pivots the tableau
<code>Pivot [<i>v</i> (, <i>tableau</i>)]</code>	pivots variable <i>v</i> into tableau
<code>Pivot [<i>v</i>, <i>w</i> (, <i>tableau</i>)]</code>	pivots <i>v</i> for <i>w</i>
<code>LimitingResource [<i>v</i> (, <i>tableau</i>)]</code>	gives the limiting resource <i>w</i> of <i>v</i>

Default tableau is Problem[]. Variables can be decision variables or slack.

<code>ShadowPrice [<i>var</i> (, <i>tableau</i>)]</code>	computes the shadow price of <i>var</i> in tableau.
<code>ShadowPrices [(<i>tableau</i>)]</code>	extracts the shadowprices of variables in the first row of tableau
<code>ShadowPrices [<i>vars</i>, <i>tableau</i>]</code>	computes the shadowprices of <i>vars</i> in tableau – no default tableau

Default tableau is Solution[].

# 7.9 Queueing

---

## 7.9.1 Summary

This package provides for some queueing basics, with a development of the concepts and the single server and multiple servers models.

```
Economics [Queue]
```

## 7.9.2 Introduction

Fundamentals in queueing concern (a) the concepts, and (b) the following models:

- the single server model, with exponential interarrival times and a service time with known mean and variance
- the finite source single server model, with exponential interarrival and service times
- the multiple server model, with exponential interarrival and k such service times.

Below we develop the concepts, provide for the models and give some utilities for plotting.

We do not use *Mathematica* here to derive results, and the functional relations are directly adopted from Hillier and Lieberman (1967).

## 7.9.3 Concepts and symbols

Expositions on queueing usually adopt the same symbols, and we are wise to adhere to that practice. A good use of *Mathematica* however demands of us that we also use symbolic names that clearly express what the symbols stands for. We can satisfy both conditions by a ListOfSymbols and a list of rules ToQueueSymbols. The short symbols are all protected to prevent assignments. The longer symbols all express that they concern *averages*.

```
ListOfSymbols["Queue"]
```

MeanArrivalRate	$\lambda$
MeanServiceRate	$\mu$
MeanUtilisationRate	$\rho$
MeanNSystem	$L$
MeanNQueue	$L_q$
MeanNBeingServed	$L_s$
MeanProcessTime	$W$
MeanQueueWait	$W_q$
MeanSacrificeRatio	$R$



Some relations such as  $L = \lambda W$  hold under essentially general conditions in a steady state. An example for  $L = \lambda W$  is the following. In the steady state the output rate equals the input rate. If a couch factory has an output of  $\lambda = 100$  couches a week, and each couch takes  $W = 2$  weeks to make, then there are  $L = 200$  couches in the factory (namely 100 in the first week of processing, 100 in the second week of processing). If we collect such steady state conditions and some definitions, then we get a model with queueing basics.

<code>QueueBasics[{x}, elims, opts]</code>	is a <code>SolveFrom</code> application
<code>QueueBasics[Equations]</code>	the steady state queueing equations

`QueueBasics[Equations] /. ToQueueSymbols // MatrixForm`

$$\left( \begin{array}{l} L == W \lambda \\ Lq == Wq \lambda \\ Ls == L - Lq \\ W == Wq + \frac{1}{\mu} \\ \text{MeanArrivalTime} == \frac{1}{\lambda} \\ \text{MeanServiceTime} == \frac{1}{\mu} \\ R == \frac{W}{\text{MeanServiceTime}} \end{array} \right)$$

The mean sacrifice ratio is a less standard concept. It identifies part of customer satisfaction, with the sacrifice given as the ratio of total process time to proper service time. We can solve the `QueueBasics` model for this  $R$ , and find a clear expression that we shall be using for plotting below.

```
MeanSacrificeRatio == (MeanSacrificeRatio /.
  First[Last[QueueBasics[{}], {MeanProcessTime}]]]) /.
  ToQueueSymbols

Solve::svars :
  Equations may not give solutions for all "solve" variables.

R == Wq μ + 1
```

We shall use the notation  $\text{Pr}[N, n]$  for  $\text{Pr}(N = n)$  or  $P_n$ , i.e. the probability that there are  $n$  units in the system. For the probability that the process time is longer than a certain value,  $\text{Pr}(\text{ProcessTime} > t)$ , we use  $1 - \text{Pr}[\text{CDF}, t]$ . In output lists, we can use such expressions in Strings.

To emphasise the distinction between the mean values and stochastic variables with arbitrary values, the following symbols have been defined too (as Symbols only).

<code>NQueue</code>	number of units in the queue
<code>NBeingServed</code>	number of units being served
<code>NSystem</code>	number of units in the system = <code>NQueue</code> + <code>NBeingServed</code>

We shall use `NServers` to indicate the number of servers. Apparently we do not need a symbol for the number of phases.

### 7.9.4 The single server model

Assumptions for the single server model are:

- a. the inter-arrival time has an independent exponential distribution with mean  $\lambda$
- b. the server has a distribution with mean service time  $1 / \mu$  and variance `var`. Alternatively, there are more servers with a common distribution such that they can be regarded as a single server.

This solves the model in terms of the mentioned three parameters.

```
SingleServer[ $\lambda$ ,  $\mu$ , var:Automatic]
```

gives the steady state for the single server model

Note: There is no useful output when the utilisation rate  $\rho = \lambda / \mu > 1$  (more arrivals than departures). The proper name for  $\mu$  would be 'mean maximal service rate' since actual service is limited by arrivals.

This routine covers the following cases:

- i. When `var` is not specified, then it automatically is assumed that the 'aggregate server' has an exponential service time, and hence `var` =  $1 / \mu^2$ .
- ii. For a constant service rate, give `var` = 0.
- iii. With Poisson input and more phases with a joint Erlang (gamma) service time, give `var` =  $1 / (k \mu^2)$  with  $k$  the number of service phases. Note that each phase has a mean service time of  $1 / (k \mu)$ .

When `var` = Automatic, i.e. an exponential service time, then the routine also defines the following probabilities:

- $\Pr[N, n] = (1 - \rho) \rho^n$
- $\Pr[\text{PDF}, t] = (\mu - \lambda) e^{(\lambda - \mu) t}$
- $\Pr[\text{CDF}, t] = 1 - e^{(\lambda - \mu) t}$

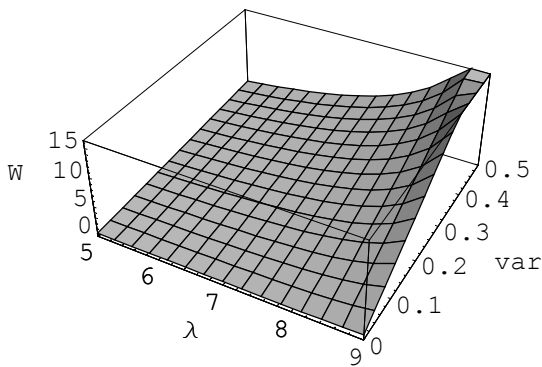
- The following call is a bit curious, with `labda` and `mu` appearing as words and as symbols, but it clarifies what the call produces.

**SingleServer[labda, mu] /. ToQueueSymbols // Simplify**

$$\left\{ \begin{aligned} &P\{\text{ProcessTime} > t\} \rightarrow e^{t(\text{labda}-\mu)}, \Pr[N, 0] \rightarrow 1 - \frac{\text{labda}}{\mu}, \lambda \rightarrow \text{labda}, \text{MeanArrivalTime} \rightarrow \frac{1}{\text{labda}}, \\ &L_s \rightarrow \frac{\text{labda}}{\mu}, L_q \rightarrow \frac{\text{labda}^2}{\mu^2 - \text{labda} \mu}, L \rightarrow \frac{\text{labda}}{\mu - \text{labda}}, W \rightarrow \frac{1}{\mu - \text{labda}}, W_q \rightarrow \frac{\text{labda}}{\mu^2 - \text{labda} \mu}, \\ &R \rightarrow \frac{\mu}{\mu - \text{labda}}, \mu \rightarrow \mu, \text{MeanServiceTime} \rightarrow \frac{1}{\mu}, \rho \rightarrow \frac{\text{labda}}{\mu}, \text{NServers} \rightarrow 1, \text{Variance} \rightarrow \frac{1}{\mu^2} \end{aligned} \right\}$$

- This plots  $W(\lambda, \sigma^2)$  for  $\mu = 10$ . If  $\lambda$  and  $\mu$  give the number of units per hour, then  $W$  is in hours too. The effect of the variance becomes dramatic when the rate of utilisation gets close to 90 %.

**Plot3D[ MeanProcessTime /. SingleServer[labda, 10, var], {labda, 5, 9}, {var, 0, .5}, AxesLabel → {λ, " var", "W "}]**



The `SingleServer` routine is a one way street. The following has multiple entries.

```
SingleServerModel[{rules}, elims, opts]
```

is a SolveFrom application based on SingleServer

- Suppose that you have a situation with a mean arrival rate of 100 customers per hour, an average of 2 customers in the system, while the standard deviation of the service time is 0.3 minutes. What are the other queueing properties predicted by the single server model? Selecting only the first of the set of solutions produced by Solve, gives:

```
(SingleServerModel[{MeanNSystem → 2.,
  MeanArrivalRate → 100/60.,
  Variance → .3^2}] // Last // First) /. ToQueueSymbols

{MeanArrivalTime → 0.6, Ls → 0.708712, Lq → 1.29129, W → 1.2, Wq → 0.774773,
 R → 2.82202, ρ → 0.708712, NServers → 1., MeanServiceTime → 0.425227, μ → 2.35168}
```

Note that the single service model has a nice heuristic explanation. With  $S = 1 / \mu$  the mean service time, a new customer can expect a mean process time  $W$  as the total of the service time for those already in the system and his or her own service time, thus  $W = S L + S$ . Rewrite this as  $W = (L + 1) / \mu$ . If we combine this with  $L = \lambda W$ , we can solve:

```
Solve[{W == (L + 1) / μ, L == λ W}, {W, L}]
```

$$\left\{ \left\{ W \rightarrow -\frac{1}{\lambda - \mu}, L \rightarrow -\frac{\lambda}{\lambda - \mu} \right\} \right\}$$

This heuristic explanation however only holds for exponential service times. We can show this by applying the general model, derive the implied variance, and recognise it as the one of the exponential distribution.

```
sol = SingleServer[λ, μ, var] /. ToQueueSymbols // Simplify;
```

```
W == (L + 1) / μ /. sol
```

$$\frac{-\text{var } \lambda \mu^2 - 2 \mu + \lambda}{2 \lambda \mu - 2 \mu^2} == \frac{\frac{\lambda (-\text{var } \lambda \mu^2 - 2 \mu + \lambda)}{2 (\lambda - \mu) \mu} + 1}{\mu}$$

```
Simplify[%]
```

$$\frac{\text{var } \lambda}{2} - \frac{\lambda}{2 \mu^2} == 0$$

```
Solve[%, var]
```

$$\left\{ \left\{ \text{var} \rightarrow \frac{1}{\mu^2} \right\} \right\}$$

### 7.9.5 Single server utilities

There are two utilities for the the single server model.

`AdjustedRates` [ $\lambda$ ,  $\mu_{th}$ , *perc*]

adjusts  $\rho$  for average lost days in the total of working days

This adjustment is as follows. A theoretical value for  $\rho_{th}$  can be determined from the actual arrival rate  $\lambda$  and the theoretical service rate  $\mu_{th}$ , but the proper values of the utilisation and service rates should include a proportion of lost time. For example, if there are 360 working days per year and a total of 10 lost days, then  $\rho = \lambda / \mu_{th} + 10 / 360$ , and then the actual service rate would be deduced again from  $\mu = \lambda / \rho$ . Hence, in this case, choose *perc* = 10 / 360.

`AdjustedRates` [ $\lambda$ ,  $\mu_{th}$ ,  $\alpha$ ]

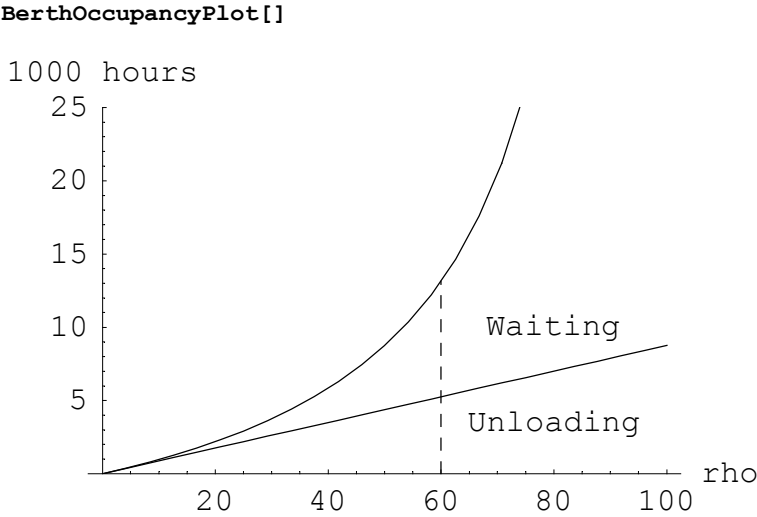
$$\left\{ \text{MeanArrivalRate} \rightarrow \lambda, \text{MeanServiceRate} \rightarrow \frac{\lambda}{\alpha + \frac{\lambda}{\mu_{th}}}, \text{MeanUtilisationRate} \rightarrow \alpha + \frac{\lambda}{\mu_{th}} \right\}$$

`BerthOccupancyPlot` [ $\lambda: \frac{2}{219}$ , *nships:80*, *opts*]

plots a typical situation at a single berth harbour

Note: The default labels are not printed for other input values.

Regard the default example that there are 80 ships of 100,000 Tonnes annually visiting a berth, giving  $\lambda = 80 \text{ Ship} / (365 * 24 \text{ Hour}) = 2 / 219$ . The service or unloading time is  $1 / \mu = \rho / \lambda$ . Both the unloading time and the process time  $W$  can be plotted as a function of  $\rho$ , with waiting time  $Wq$  as the portion inbetween.



A conclusion is that increasing the unloading capacity, at the cost of a lower utilisation rate, could significantly reduce harbour time per ship

**7.9.6 The finite source single server model**

`FiniteSourceSS[ $\lambda$ ,  $\mu$ , M]` gives the single server results when the number of units in the source is exactly M

`FiniteSourceSS[100/60, 2.4, 10] /. ToQueueSymbols`

$\{ \text{Pr}[N, 0] \rightarrow 2.5031 \times 10^{-6}, \lambda \rightarrow \frac{5}{3}, \text{MeanArrivalTime} \rightarrow \frac{3}{5}, \text{Ls} \rightarrow 0.999997, \\ \text{Lq} \rightarrow 7.56001, L \rightarrow 8.56, W \rightarrow 3.56668, \text{Wq} \rightarrow 3.15001, R \rightarrow 8.56003, \\ \mu \rightarrow 2.4, \text{MeanServiceTime} \rightarrow 0.416667, \rho \rightarrow 0.999997, \text{NServers} \rightarrow 1 \}$

**7.9.7 The multiple servers model**

Assumptions for the multiple servers model are:

- a. the inter-arrival time has an independent exponential distribution with mean  $\lambda$
- b. the  $s$  servers are independent with each an exponential distribution with mean service time  $1 / \mu$ .

This solves the model in terms of the mentioned three parameters.

`MultipleServers[ $\lambda$ ,  $\mu$ , s]` gives the steady state for the multiple servers model

Note: There is no useful output when the utilisation rate  $\rho = \lambda / (s \mu) > 1$  (more arrivals than departures).

The routine also defines the probabilities:

- $\Pr[N, n]$  with different formulas for  $n = 0$ ,  $0 < n \leq s$ , and  $n > s$
- $\Pr[\text{CDF}, t]$

The following is an example of a barbershop, where an operations manager would advise to start using appointments and schedules to increase profitability.

- Regard a barbershop with  $\lambda = 9$  customers per hour (Poisson) and 3 chairs where a haircut takes 15 minutes (exponential), so that  $\mu = 4$  customers/hour.

**MultipleServers[9., 4., 3] /. ToQueueSymbols**

```
{P{ProcessTime > t} → e-4.t (1 - 2.27103 (1 - e1.t)), Pr[N, 0] → 0.0747664, λ → 9.,
MeanArrivalTime → 0.111111, Ls → 2.25, Lq → 1.70327, L → 3.95327, W → 0.439252,
Wq → 0.189252, R → 1.75701, μ → 4., MeanServiceTime → 0.25, ρ → 0.75, NServers → 3}
```

- What is the probability that 5 or more customers are in the shop ?

```
1 - Sum[Pr[N, i], {i, 0, 4}]
```

0.319363

- The following shows that the mean utilisation rate equals 1 - (the weighted sum of the probabilities of non-use). For  $\Pr[\text{zero customers}]$  all 3 chairs are empty, for  $\Pr[\text{only 1 customer}]$  2 chairs are empty and for  $\Pr[\text{only 2 customers}]$  only 1 chair is empty.

```
1 - (3 Pr[N, 0] + 2 Pr[N, 1] + 1 Pr[N, 2]) / 3
```

0.75

While `MultipleServers[ ]` is a one way street routine, we should be able to solve in many more ways with a `SolveFrom` application. However, the model contains transcendental terms that `Solve` cannot tackle. Also `NSolve` stops here. A future approach would involve `FindRoot`.

```
MultipleServersModel[{rules}, elims, opts]
```

is a `SolveFrom` application

Note: It appears that `Solve` cannot really tackle this kind of model.

- This is not evaluated here, since `Solve` meets transcendental terms.

```
MultipleServersModel[{MeanQueueWait → 0.05625,
MeanServiceRate → 10., NServers → 2}]
```

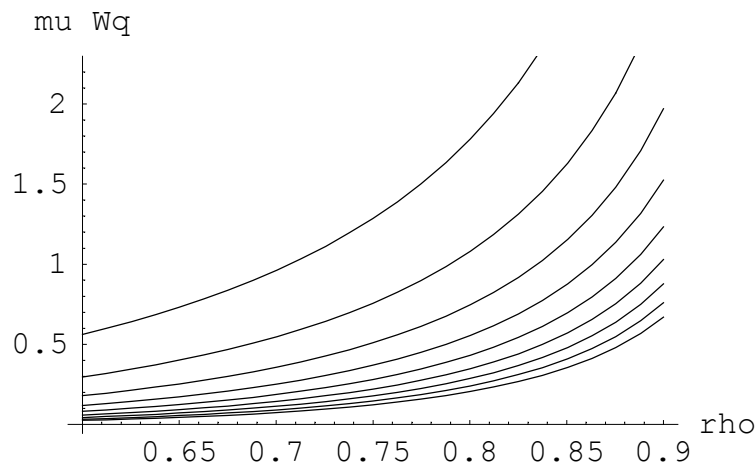
7.9.8 Sacrifice ratio plot

Above we deduced for the sacrifice ratio that  $R = \mu Wq + 1$ . We may subtract 1, to get the ratio of waiting to being served. The following routine plots these ratio's for various values of  $s$  and  $\rho$ . The plot should help one in determining the number of service units, while balancing customer service with a sound rate of utilisation. Interestingly, the plot is independent of  $\lambda$  and  $\mu$  (and depends only on their ratio).

```
MultipleServersContours[smin, smax, domain, opts__Rule]
```

plots contours of  $s$  from  $smin$  to  $smax$  (integer values),  
for  $\rho = \lambda / (s \mu)$  in domain (with default {0, 1}) and for  $y = \mu Wq$

```
MultipleServersContours[2, 10, {.6, .9}]
```





# Appendix A: Common routines

The common routines are always available once `Needs["Economics`Pack`"]` has been evaluated successfully.

The routines extend *Mathematica*'s support on the following topics:

1. Numerical calculation
2. Symbolic manipulation
3. Strings and Patterns
4. Lists
5. Data input and presentation
6. Equations and models
7. Programming
8. Context working environment
9. Graphics
10. Economics tools
11. Technical note

# A.1 Numerical calculation

## A.1.1 Addition and division

<code>Add[x]</code>	ruthless addition, namely <code>Apply[Plus, Flatten[{x}]]</code>
<code>Division[x, y]</code>	can be useful when <code>Indeterminate</code> stands for missing data. It sets <code>Infinity</code> and <code>ComplexInfinity</code> to <code>Indeterminate</code> . With <code>Messages → Off</code> , by default in <code>Options[Division]</code> , messages on division by 0 are suppressed
<code>Division[h, z, {n___}, {d___}]</code>	applies head <code>h</code> on <code>z</code> with separate sequences of arguments <code>d</code> and <code>n</code> , giving <code>Division[h[Numerator[z], n], h[Denominator[z], d]]</code>
<code>Division[h, z, {both___}]</code>	uses <code>n = d = both</code>

NB. Divide is a *Mathematica* function.

```
Add[{{1, 2}, {1, 2}, {1, 2}}]

9

vb = (1/x + 1/ (x y)) / (1/x^2 + 1/x^3)


$$\frac{\frac{1}{yx} + \frac{1}{x}}{\frac{1}{x^2} + \frac{1}{x^3}}$$


Simplify[vb]


$$\frac{x^2 (y + 1)}{(x + 1) y}$$


Division[Simplify[Times[##]] &, vb, {x}]


$$\frac{x^2 \left(1 + \frac{1}{y}\right)}{x + 1}$$

```

## A.1.2 Zero, Indeterminate and Null

Zero means the absence of a proper number. The best indicator of missing values likely is `Indeterminate`. `Null` means the absence of even these.

There is a proper distinction between 0 and 0. which in many cases however is awkward.

<code>RealN[x, n Integer: \$MachinePrecision]</code>	ruthless version of <code>N[ ]</code> , also converses "1" to 1..
--	---

<code>NonZero[x]</code>	deletes 0 and 0. from x without changing the structure of x (that can be lists). Only when x = 0 or 0., then output is Null
<code>NonZeroQ[x]</code>	<code>x === NonZero[x]</code>

<code>NullQ[x]</code>	gives a warning when x contains Null somewhere.
<code>NoNulls[x]</code>	removes Null elements from x
<code>OnlyNulls[n_Integer]</code>	gives <code>Table[Null, {i, n}]</code>
<code>Nullify[x]</code>	sets x to Null if it is any kind of function of Null, and if x is a list, then it applies to elements.

Note: Users who use Null to stand for missing data, would use Nullify to make sure that operations on Null give Null too. Note too that it is easier to use Indeterminate for missing data, since operations on Indeterminata automatically give Indeterminate again.

■ In *Mathematica* 3.0 `N[0]` gives 0 and not 0..

```
{N[0], RealN[0], RealN["1"]}
{0, 0., 1.}

NonZero[{ax, bx, 0, {2, 0., cx}}]
{ax, bx, {2, cx}}

Nullify[{func[Null], {Null^Null, 3 Null}, 4}]
{Null, {Null, Null}, 4}
```

### A.1.3 Ranges

<code>Between[x_?NumberQ, y_List]</code>	
	finds the pairs of subsequent values <code>{y[[i]], y[[i+1]]}</code> for which x is between these. If y is sorted then this gives maximal ymin and minimal ymax so that <code>(ymin &lt; x &lt; ymax)</code>
<code>InRangeQ[x, {lt, ge}]</code>	tests <code>(lt &lt; x &lt;= ge)</code>
<code>InRange[x_?NumberQ, pairs_List]</code>	
	finds the pairs in the list-of-pairs that makes <code>InRangeQ[x, pair]</code> True

```
Between[13, {11, 2, 20, 12, 14, 15}]

$$\begin{pmatrix} 2 & 20 \\ 12 & 14 \end{pmatrix}$$

```

```
InRange[3, {{1, 4}, {5, 6}, {2, 6}}]
```

$$\begin{pmatrix} 1 & 4 \\ 2 & 6 \end{pmatrix}$$

Note that section 3.9 on domain control, inequalities and piecewise, also contains routines on ranges.

**A.1.4 NIntegrate the positive area**

Integration takes areas above the horizontal axis as positive and areas below as negative. Many economics problems only concern the positive part. The following automates the integration to the next intersection of the integrand with the horizontal axis.

```
NIntegrateToFirstZero[f, p_?NumberQ, opts___Rule]
```

finds the intersection with the horizontal axis closest to p,  
and integrates from p to this point. Options are passed on to NIntegrate

Subresults of NIntegrateToFirstZero[ ] are in Results[NIntegrateToFirstZero], namely all intersections with the horizontal axis, the selected one, and the direction of the operation (i.e. to the left or to the right). Also provided is the startpoint Point → {p, f[p]}.

- For a demand function  $D[p] = 10 - 1.5 p^2$  the total value beyond  $p = 1$  is:

```
NIntegrateToFirstZero[10 - 1.5 #^2 &, 1]

7.71326

Results[NIntegrateToFirstZero]

{Point → {1, 8.5}, Solve[f[x] == 0] → {-2.58199, 2.58199},
 Select → 2.58199, Direction → Right, NIntegrate → 7.71326}
```

**A.1.5 Exponential smoothing**

<pre>Smooth[y_List, f:0.75 Null]</pre>	develops $z[t] = f z[t-1] + (1 - f) y[[t]]$ , with $z[1] = y[[1]]$ . The default $f = 0.75$ is a good smoother
<pre>Smooth[r][y_List, f:.75]</pre>	develops $z[t] = (1 + r) (f z[t-1] + (1 - f) y[[t]])$
<pre>Smooth[Find][x_List, a_List: {}, b_List: {}, opts]</pre>	finds the parameters $f[1]$ and $c$ that generate the best exponential forecast series for $x$ .

- Smooth is a particular application of FoldList. Examples are:

```
Smooth[{975, 1100, 1400, 1000}]  
  
{975, 1006.25, 1104.69, 1078.52}  
  
Smooth[.1][{975, 1100, 1400, 1000}]  
  
{975, 1106.88, 1298.17, 1345.99}
```

The forecast requires more explanation. The forecast series starts with  $f[1]$  and the next forecast is the exponential smoother  $f[2] = c f[1] + (1 - c) x[[1]]$ . Etcetera. Numerical estimation is used, since the symbolic approach appears very slow. List  $a = \{f0, f1\}$  gives the estimation start values for  $f[1]$ , and list  $b = \{c1, c2\}$  gives the estimation startvalues for the exponential update parameter  $c$ . Empty lists use default values. Since the startvalue  $f[1]$  may be less relevant for long lists, you may enter  $a = \{\text{Set}, f[1]\}$  or  $a = \{\text{Set}, \text{Automatic}\}$ , and in the latter case  $f[1] = \text{Average}[x]$ . Options are passed on to FindMinimum. Results are: List → the smoothed series, First →  $f[1]$ , Factor → the smoothing factor  $c$ , Last → forecast for the period beyond the list, Min → SSE

```
Smooth[Find][{975, 1100, 1400, 1000}, {Set, 1000}]  
  
{Last → 1054.96, First → 1000, Factor → 0.863999,  
List → {1000, 996.6, 1010.66, 1063.61, 1054.96}, Min → 166947.}
```

A.1.6 Some small utilities

IntegerSCM[s_List]	finds the Smallest Common Integer multiplier k such that k s are all integer. Works for rational numbers in s only
PointOfIntersection[f, g, k:1]	for Real valued functions f and g finds the k–th solution x = xs for f[x] = g[x] and renders {xs, f[xs]}
RoundAt[x, n_Integer: 0]	round at some decimal value i.e. Round[10^n x] / (10^n)
Decimals	symbol

# A.2 Symbolic manipulation

## A.2.1 DQ

If  $D[expr, x]$  is zero, then we conclude that  $expr$  is no function of  $x$ . In the same way as 0 stands for the absence of a proper number. However, it may sometimes be useful to have an indication whether taking the derivative is relevant at all. In economics everything hangs together, but taking the derivative of the value of gold with respect to the mass of the universe might require an error message.

<code>DQ[expr, x]</code>	tests whether $expr$ is a function of $x$ . If False, then the message is printed that taking the (partial) derivative could be irrelevant.
--------------------------	---

Note: DQ can be seen as !FreeQ with a message.

- See these examples and compare this with FreeQ.

```
DQ[f[x] + 1, y]

D::used : D[f(x) + 1, y] may be irrelevant
        since f(x) + 1 is not an explicit function of y

False

DQ[f[x] + 1, x]

True

FreeQ[f[x] + 1, y]

True
```

## A.2.2 ToSequence

The following is useful for example in the conditional inclusion of options.

<code>ToSequence[ ]</code>	generates a Sequence[] only when evaluated
----------------------------	--

- This gives Null within the list of options.  
`{opt1 → a, If[2 < 1, opt3 → c, Sequence[]]}`  
`{opt1 → a, Null}`

```
■ This gives Sequence[], and then disappears.  
{opt1 → a, If[2 < 1, opt3 → c, ToSequence[]]}  
  
{opt1 → a}
```

A.2.3 Expression tests

These functions test some variable, and render True or False.

ConstantQ[x]	tests whether x is a constant (or function of those)
DefinedQ[x]	gives True iff x has been used or defined in some fashion or other so that it has a value or that it is the head of some function
EquationQ[x]	is False when x reduces to True or False, when Infinity, Indeterminate or ComplexInfinity occur, and when x is a form of 1/y == 0. Otherwise it is True.
LessComplicatedQ[x, y]	is True, if x is less complicated than y, otherwise False
RealNumberQ[x]	gives True if x is really a number (i.e. NumberQ[N[x]])
RealQ[x]	gives True if x is real number (Head[x] === Real)
RuleQ[x]	tests for a rule, True if (Head[x] === Rule)
SmatchQ[a, m]	uses either MatchQ or StringMatchQ. If the option Map → True (default) is set, it evaluates Smatch[#, m]& /@ a

```
■ Note the subtle difference (and try this for Sqrt[2] too):  
  
{NumberQ[##], RealQ[##], RealNumberQ[##]}&[Pi]  
  
{False, False, True}
```

The command Intercept relies on ConstantQ, and allows the retrieval of symbolic intercepts.

Intercept[expr]	gives the constant term in a linear expr
-----------------	--

```
a^2 + 223 + Pi + Log[z] + C[2] // Intercept  
  
c2 + π + 223
```

A.2.4 Using levels

These routines concern positions of terms in expressions.

<code>LevelInspect[expr]</code>	prints by level the levelspec and level of <code>expr</code>
<code>PartInspect[expr, opts]</code>	gives a list of the elements and their positions in <code>expr</code>
<code>MapLevel[f, x, levelspec:{1}, opts]</code>	would be more accurate in mapping to levels than <code>Map</code> ; gives <code>Map[f, Level[x, levelspec, opts]]</code>

`PartInspect` gives the positions as lists, and the elements can be extracted by applying `PartList` on them. A default option is `Expand → True`, so that same elements are recorded by position; otherwise, the positions are collected in a list. Set option `Heads → False` if you do not want to include `Heads`.

$$\text{LevelInspect}\left[\frac{\text{Cost}\left(\frac{c}{w}\right)^S}{(1-c)^S \text{rpk}^{1-S} + c^S w^{1-S}}\right]$$

1

$\left\{\text{Cost}, \left(\frac{c}{w}\right)^S, \frac{1}{(1-c)^S \text{rpk}^{1-S} + c^S w^{1-S}}\right\}$

2

$\left\{\frac{c}{w}, S, (1-c)^S \text{rpk}^{1-S} + c^S w^{1-S}, -1\right\}$

3

$\left\{c, \frac{1}{w}, (1-c)^S \text{rpk}^{1-S}, c^S w^{1-S}\right\}$

4

$\{w, -1, (1-c)^S, \text{rpk}^{1-S}, c^S, w^{1-S}\}$

5

$\{1-c, S, \text{rpk}, 1-S, c, S, w, 1-S\}$

6

$\{1, -c, 1, -S, 1, -S\}$

7

$\{-1, c, -1, S, -1, S\}$

8

$\{\}$

A.2.5 Rule basics

There are some simple routines.

<code>eluR[expr]</code>	reverses all rules, giving <code>expr /. Rule[a_, b_] → Rule[b, a]</code>
<code>Delta[x → y, x → z]</code>	gives <code>x → (y-z)</code>
<code>Delta[x → y, x → z, h]</code>	gives <code>x → h[y,z]</code>
<code>SequenceToRules[{t___}, val]</code>	is shorthand for <code>Rule[#, val]&amp; /@ {t}</code>

`SequenceToRules[{x, y, z}, 0]`

$\{x \rightarrow 0, y \rightarrow 0, z \rightarrow 0\}$

A.2.6 Combine or delete rules

There are some simple routines.



<code>CombineRules[{r__Rule}]</code>	gives combinations of rules including {}
<code>DeleteRule[x, h__]</code>	deletes (h → To) in x (more h's allowed)

```
CombineRules[{a → b, c → d}]
```

```
{{}, {c → d}}, {{a → b}, {a → b, c → d}}
```

```
DeleteRule[%, a]
```

```
{{}, {c → d}}, {{}, {c → d}}
```

### A.2.7 ReplaceInRule and ReplaceRule

The following replaces in the RHS of the rule.

```
ReplaceInRule[{x__Rule}, {y__Rule}]
```

replaces the RHS's of the  $x$  by rules  $y$

```
ReplaceInRule[{x → y, y → y^2}, {y → Sqrt[a]}]
```

```
{x →  $\sqrt{a}$ , y → a}
```

`ReplaceRule` does the replacement of a rule  $x \rightarrow \text{oldvalue}$  with a rule  $x \rightarrow \text{newvalue}$ .

```
ReplaceRule[old_List, new_List, opts]
```

where old and new are lists of rules (From → To, ...). For equal From's, the To's in old are replaced with those in new. If option `Union` → True (default) then the new From's unknown to old are included in the final result

```
ReplaceRule[x_List, opts]
```

where  $x$  is a list {old, new, newest, hotnew, ...}. Replacement goes pairwise from the right. If `Union` → False and the middle rules do not contain items of the right, then the middle works as a sink, through which the hot news doesn't pass

- This is a default effect.

```
ReplaceRule[{{a → b, u → j, k → u},
             {a → c, f → g},
             {u → y}}]
```

```
{a → c, f → g, k → u, u → y}
```

- In this example  $u \rightarrow y$  does not pass, since the middle doesn't contain a  $u$ , and such a  $u$  is neither added to that middle list since  $\text{Union} \rightarrow \text{False}$ .

```
ReplaceRule[{{a → b, u → j, k → u},
             {a → c, f → g},
             {u → y}}, Union → False]
```

$\{a \rightarrow c, u \rightarrow j, k \rightarrow u\}$

### A.2.8 Replacement by head count

If an expression contains an item more than once, and one wishes to replace only some particular occurrences of these, then the common approach is to determine the position and to replace by position. In some cases it is also possible to replace with level conditions. These approaches can require a lot of attention when expressions grow to be complex. Replacing one subexpression may affect the position of other subexpressions, and one would have to update the position and level information - and check whether these still concern the subexpressions that one is concerned about. In the following procedure the various occurrences are simply counted, and can be directly addressed as unique formats. Once the replacement has been performed, the count is easily removed.

`ReplaceByHeadCount[expr, x, head]`

replaces  $x$  in  $\text{expr}$  by  $\text{head}[x, \text{int}]$  where  $\text{int}$  counts occurrences.  
If  $x$  is a list, then replacement is folded over  $x$ .

Remove the counts by  $\text{expr} /. \text{head}[x\_ , n\_ ] := x$ .

$$\text{Jansen} \&\& \frac{\sqrt{\{y \text{ Jansen}\}}}{y};$$

`ReplaceByHeadCount[%, {Jansen, y}, FOUND]`

$$\text{FOUND}(\text{Jansen}, 1) \bigwedge \left\{ \frac{\sqrt{\text{FOUND}(\text{Jansen}, 2) \text{FOUND}(y, 2)}}{\text{FOUND}(y, 1)} \right\}$$

`% /. FOUND[Jansen, 2] → Johansen /. FOUND[x_, n_] := x`

$$\text{Jansen} \bigwedge \left\{ \frac{\sqrt{\text{Johansen } y}}{y} \right\}$$

### A.2.9 Other uses of Heads

`NestHead` is used to create complex structures, for example for a CES function with more levels. Of course such results can also be gotten in many more ways.

`NestHead[head, lis]` maps *head* over *lis* and the generated (sub–) result, until the (sub–) result of any level no longer is a List

```
f[x] = {x1, x2, x3};
f[x1] = {y1, y2};
f[y2] = {z1, z2, z3};
f[x3] = {z1, x1};
NestHead[f, x]

{{y1, {z1, z2, z3}}, x2, {z1, {y1, {z1, z2, z3}}}}
```

`ExtractWithHead[expr, head:Rule]` extracts from *expr* all subexpressions with *head*

```
ExtractWithHead[{aa → bb, cc, dd}]

{aa → bb}
```

## A.2.10 Simplification

Simplification appears complex.

`SimplifyBy[expr, {rules___Rule}, opts]`

applies rules repeatedly and FullSimplifies using *opts*

`SimplifyBySolve[expr, By:Automatic, expandq:True, printq:True, opts]`

tries to simplify an expression by first solving for *By*, and then back–solving for the expression again. This works at least in some cases where powers are involved. If *expandq*, first FullSimplify[PowerExpand[*expr*]]. If *By* === Automatic, then Return. If *printq*, print intermediate results. The options are for Solve.

`SimplifySqrt[expr]` brings a number within a Sqrt

- It depends upon what one considers simpler.

```
2 Sqrt[r/2. + 3] // FullSimplify
```

$$2\sqrt{0.5r+3}$$

```
% // SimplifySqrt
```

$$\sqrt{2.r+12}$$

- An example are logarithms:

```
FullSimplify[ $\frac{\text{Log}[(a^5 + b^4)^5]}{\text{Log}[a^5 + b^4]}$ ]
```

$$\frac{\log((a^5 + b^4)^5)}{\log(a^5 + b^4)}$$

```
SimplifyBy[% , LogRule]
```

```
5
```

- Another example for a CES-like situation:

```
 $\left(\left(\frac{a}{c}\right)^e + \left(\frac{b}{c}\right)^e\right)^{v/e}$  // PowerExpand // FullSimplify
```

$$((a^e + b^e)c^{-e})^{\frac{v}{e}}$$

```
SimplifyBySolve[% , c , True , False]
```

```
Solve::ifun : Inverse functions are being  
used by Solve, so some solutions may not be found.
```

```
Solve::ifun : Inverse functions are being  
used by Solve, so some solutions may not be found.
```

$$(a^e + b^e)^{\frac{v}{e}} c^{-v}$$

Some simplification can depend upon the domains involved. Such testing will increasingly be done by *Mathematica* itself, witness the new routines in version 4.0. The following thus may be more of a farewell statement. For a test function `fq`, that tests a function `g[s]` on domain `s` in `[min, max]`, so that `fq[g[s]]` is `True` or `False`, we may test only the extreme values if `g[s]` is a monotone function of `s`, or linear in particular.

```
Positive[1 + s]
```

```
Positive[s + 1]
```

```
SimplifyTestByLinear[Positive[1 + s], {s, 0, 1}]
```

```
True
```

### A.2.11 Redefine

`Redefine[x, old_List, new_List]`

Switches x from old to new. Normally x is an element of old, but when x is a list, then x[[1]] is taken as the Switch expression while the other elements are part of the Switch body

- This is just a server for `Switch[ ]`.

`Redefine[x, {a, c}, {d, e}]`

`Switch[x,`  
`a, d,`  
`c, e]`

`Redefine[{x, y, z}, {}, {}]`

`Switch[x,`  
`y, z]`

### A.2.12 UnNumberForm

`UnNumberForm[expr]`      removes all `NumberForm` heads  
in `expr` and leaves only the first elements

`NumberForm[2 10^6/3., 8, DigitBlock → 3, NumberPadding → "$"]`

\$666,666.67

`UnNumberForm[%]`

666667.

## A.3 Strings and patterns

---

### A.3.1 Names and Symbols

Since Strings cannot have values, they are useful to manipulate variables.

<code>ToName[ x__ ]</code>	gives the concatenation of {x} for Strings or undefined Symbols
<code>ToProperName[ x_String ]</code>	gives a String with the first letter a capital and the rest in lower case.
<code>ToProperName[ x__ ]</code>	concatenates (after first applying ToString)
<code>ToSymbol[ x__ ]</code>	make a Symbol by ToExpression of the Cleared concatenation of {x}

- The first line sets a value, the second line creates the symbol and clears it.

```
MyDataList = {1, 2, 3};  
  
ToSymbol[My, Data, List]  
  
MyDataList
```

### A.3.2 Lists of strings

The Logic` and Inference` packages use sentences.

<code>StringListQ[ a_List ]</code>	tests whether the a[[i]] are Strings
<code>StringListPralse[ x_List ]</code>	Pralse[] when the elements of x are not Strings
<code>StringToList[ x_String ]</code>	turns a sentence into a list of words, while blanks are regarded as separators
<code>GeneralizedStringTake[ x, k_Integer, rigorous_Symbol: All ]</code>	takes the first k positions, even in lists. If All, then expressions are transformed into strings

### A.3.3 DefinitionToString

The following can be used to store a definition and to be able to recall it.

```
DefinitionToString[x_String]
```

gives the Definition of *x* as a Stringed List  
 {..., ..., ...}, which might be submitted again to ToExpression

Alternative to storing DownValues[x].

```
MyDataList = {1, 2, 3};

DefinitionToString["MyDataList"]

{MyDataList = {1, 2, 3}}

FullForm[%]

"{MyDataList = {1, 2, 3}}"
```

### A.3.4 Patterns

```
ToPattern[x_List, head_: Blank]
```

creates a rule for the variables in *x*, that can be used to change an  
 expression into a pattern. Evaluates Map[ Rule[#, Pattern[#, head[]]]&, *x*]

```
UnPattern[y]                      evaluates y /. Pattern → Composition[First, List]
```

- An example is:

```
ToPattern[{y, u]}
```

```
{y → y_, u → u_}
```

```
y2 + u        /. %
```

```
y_2 + u_
```

# A.4 Lists

Lists have many applications. They can be used as ordered lists or just sets. They can be neat matrices or represent object structures. The following list of routines thus covers many areas.

## A.4.1 Quantifiers

A Hintikka observation is that a quantifier always requires a range.

AllQ[list, crit, levelspec:{1}, opts\_\_\_Rule]  
  
gives True if list is nonempty and all its elements cause crit to be True  
  
ExistQ[list, crit, levelspec:{1}, opts\_\_\_Rule]  
  
gives True if list is nonempty and some of its elements cause crit to be True

Note: These routines are comparable to Select, Cases & Count.

Note: MapLevel is used. Levelspec (default 1) and opts are for Level. Use levelspec Infinity when crit is to apply to any level.

- AllQ[expr, test] is not equivalent to VectorQ[expr, test]. The latter gives True on the empty set, in the same way as the empty set is a subset of every set. AllQ gives False here, since in a programming environment one wishes the True value only for proper sets.

VectorQ[{}, ListQ]

True

AllQ[{}, ListQ]

False

- Note, incidently, that a test with VectorQ can be True even when expr is not a vector. A "vector with list elements" would be a contradiction in terms, but Mathematica still generates True. The proper test whether expr is a vector and satisfies test appears to be VectorQ[expr] && VectorQ[expr, test].

VectorQ[{{a}, {b}}, ListQ]

True

VectorQ[{{a}, {b}}] && VectorQ[{{a}, {b}}, ListQ]

False



### A.4.2 Understanding structures of lists

Tip: compare `DimensionForm[ ]` with `LevelInspect[ ]`.

```
DimensionMap[x, onlyLists:True]
```

similar to mapping `Dimensions` on `x`. By default only Lists are recognised

```
DimensionNest[x, onlyLists:True, sor:True]
```

repeated application of `DimensionMap`

```
DimensionForm[x, onlyLists:True, sor:True]
```

prints Nested `DimensionMap` in `TableForm` (for vectors transposed)

An example is:

```
DimensionForm[{1, {2, {3, {4}}}}]
```

```

2
1   2
   1
1   2
   1
1   1  1
```

### A.4.3 Using a mold

The following basic routine is very important for the Databank package.

```
FromMold[mold_List, y_List: {}]
```

supplements `y` with a list structure comparable to `mold`.

Indeterminate's are used as placeholders.

If `y` has an Indeterminate where `mold` has a list, then this is replaced too

```
FromMold[{a, {b}, {c, {d}}}, {x}]
```

```
{x, {Indeterminate}, {Indeterminate, {Indeterminate}}}
```

### A.4.4 Repeat operations

For functions like these there are good *Mathematica* equivalents, so that these are only shorthand. The main reason to define these anyhow is that they help remember how the *Mathematica* constructions are.

<code>Cumulate[x]</code>	cumulates the entries in the list <code>x</code>
<code>Cumulate[x, y]</code>	cumulates <code>x</code> , divides by the last result, multiplies with <code>y</code>
<code>Repeat[head, x]</code>	<code>FoldList[head, First[x], Rest[x]]</code> . Head examples are: <code>Plus</code> for Cumulation, <code>Times</code> for multiplying throughout and <code>Divide</code> for dividing throughout

Cumulative percentages are:

```
Cumulate[{1, 2, 3}, 100.]  
  
{16.6667, 50., 100.}
```

`FoldList` can use a reverse.

<code>FoldListReverse[f, x, lis_List, opts]</code>	is a reverse application of <code>FoldList</code>
<code>FoldReverse[f, x, lis_List, opts]</code>	the last element of <code>FoldListReverse</code>

Note: The `opts` are meant for `f` so one also gets `f[u, v, opts]`.

```
FoldList[f, x, {a, b, c}]  
  
{x, f(x, a), f(f(x, a), b), f(f(f(x, a), b), c)}
```

```
FoldListReverse[f, x, {a, b, c}]  
  
{c, f(b, c), f(a, f(b, c)), f(x, f(a, f(b, c)))}
```

A.4.5 Sets

`Join[a, b]` does not check whether *a* and *b* have the same elements. `Union[a, b]` does, but also sorts. `JoinNews[a, b]` checks, adds only the new elements, but does not sort. For example when a quality inspector is working on complaints, using a First In First Out (FIFO) rule, it does not matter whether old complaints keep arriving in, but the order still is relevant.

<code>JoinNews[x_List, y_List]</code>	shorthand for <code>Join[x, Complement[y, x]]</code>
<code>MoveToFirst[x_List, y]</code>	moves element y to first position in list x
<code>PartList[expr, y_List]</code>	is Part with a list as input rather than a sequence = <code>Part[expr, Sequence @@ y]</code>
<code>TakeElements[lis, p_List]</code>	takes the elements of list lis from positions which have been found by <code>Position[lis, p]</code> . Useful for parallel structures: <code>g = { w, {y,e,w}}</code> <code>g2 = { w1, {y1,e1,w1}}</code> <code>pos = Position[g, w]</code> <code>TakeElements[g2, pos]</code>

Note: Use *Mathematica* 3.0.0's `Extract` now instead of `TakeElements`; it is mentioned here since older notebooks may still use it. Note the difference with `PartList`.

For deletion:

<code>DeleteElement[lis, e, normal:True]</code>	normally (True) deletes e from the list, otherwise maps this deletion over the various occurrences of e
<code>DeleteVectorPosition[lis, n_Integer]</code>	deletes position n in any vector in the list

`DeleteElement[{{a, b, c}, {b, d, e}}, b]`

$$\begin{pmatrix} a & c \\ d & e \end{pmatrix}$$

`DeleteVectorPosition[{{a, b, c}, {b, d, e}}, 2]`

$$\begin{pmatrix} a & c \\ b & e \end{pmatrix}$$

A.4.6 Matching and pairs

<code>CasesMatch[x_List, m]</code>	uses <code>SmatchQ</code>
<code>Polygamy[x_list]</code>	finds those pairs in <code>x</code> that have the same first element
<code>SortSameQ[x, y, test]</code>	is the same as <code>test[x,y] &amp;&amp; test[y,x]</code> , where the test can also be used in <code>Sort</code>
<code>Min2ndOf2[x]</code>	selects the pair in <code>x</code> of which the 2 nd element is minimal of all 2 nd elements.If more,the last is taken

- `Polygamy[ ]` is used in testing preference patterns - although that is a curious way of expressing.

```
Polygamy[{{Jan, Mary}, {Peter, Mary}, {Jan, Suzy}}]
```

```
(Jan Mary)
(Jan Suzy)
```

- This subroutine selects the pair of which the second element is the minimal of all second elements. The first element might be a label, and the other the criterion score.

```
Min2ndOf2[{{2, 3}, {aa, 4}, {1, 2}}]
```

```
{1, 2}
```

A.4.7 Indexed sorting and RealSort

`Sort[ ]` simply forgets where everything came from. By adding an index, positions can be recovered and results can be duplicated.

<code>IndexedSort[x_List]</code>	sorts <code>x</code> and gives the list of indices of the original position
<code>IndexedSort[x, p]</code>	sorts using the ordering function <code>p</code> . Note that the actual sort is on the pairs <code>{x1, 1}</code> , <code>{x2, 2}</code> , ...

With the result `r`, you can sort a list `s` by `s[[#]]& /@ Last[r]`.

An example is:

```
IndexedSort[{E, Pi, Log, I}]
```

```
(i e Log pi)
(4 1 3 2)
```

An application is `RealSort`. Note that `x` and `N[x]` sort differently in principle. Sometimes we want the order of `N[x]` but still see `x`.

<code>RealSort[x_List]</code>	sorts according to the $N[x]$ representation (that differs from the symbolic one) but does not give $N[x]$ in the result. If $x$ is a matrix, then only the first row is taken to sort the matrix
<code>RealSort[Transpose, x]</code>	sorts for the first column (matrixes only)

```
RealSort[Transpose, {{1 ≤ x ≤ 2, 1 - x}, {1/2 (-1 + Sqrt[5]) ≤ x ≤ 1, x^2}}]
```

$$\left( \begin{array}{cc} \frac{1}{2}(-1 + \sqrt{5}) \leq x \leq 1 & x^2 \\ 1 \leq x \leq 2 & 1 - x \end{array} \right)$$

A.4.8 Matrices

Don't forget about `LinearAlgebra`MatrixManipulation``.

`ToMatrix[y]` or `ToMatrix[y, number]`

transforms a list of unequally sized lists into a matrix,  
by appending Indeterminate's. If no number is specified,  
then the maximum length of these lists is taken. Also,  
a single vector is turned into a matrix

`MatrixRowSums[matrix_List, borders:False]`

adds the elements per row;  
if `borders===True` then the first row and column of  
the matrix are excluded (these can be labels)

`MatrixTimesDiagonal[matrix_List, diagonal_List, borders:False]`

multiplies a twodimensional matrix with  
a diagonal matrix formed from vector diagonal;  
if `borders===True` then the first row and column of  
the matrix are excluded (so that the vector diagonal  
must have one element less than the matrix row dimension)

`BorderTotals[x_List]` gives the border sums of x using `TensorRank`.

`BorderTotals[x, Eliminate→{n_Integer, m___Integer}, hd:Plus, opts___]`

eliminates dimensions n to m from the tensor, default by applying `Plus`

`BorderTotals[x, Partition→y_List, hd:Plus, opts]`

partitions the border sums according to `y =`  
`{{imin1, imax1}, {imin2, imax2}, ...}` where the `imaxj + 1 = imin(j+1)`.  
A single number `i = mini = maxi` may be used for `{mini, maxi}`,  
as in `{1, {2, 4}, 5}`. Per part, the complement is eliminated.

`BorderTotalsQ[tr_Integer, Partition→y_List]`

provides merely a check on the partition. It is `True` if  
the partition is in right order and fully exhausts the tensor rank `tr`

Note for `BorderTotals`: `Eliminate` and `Partition` are pseudo-options: they must be mentioned as shown and cannot be set by `SetOptions` or be combined in arbitrary order. The routine employs `Apply[hd, x, {n, m}-1, opts]`. Use `Transpose` to get the dimensions of x in consecutive order.

- First make a probability measure.

```
pm = Outer[Times, {0.1, 0.9}, {0.4, 0.6}, {0.5, 0.2, 0.3}]


$$\begin{pmatrix} \{0.02, 0.008, 0.012\} & \{0.03, 0.012, 0.018\} \\ \{0.18, 0.072, 0.108\} & \{0.27, 0.108, 0.162\} \end{pmatrix}$$

```

- The third dimension is summed out.

```
BorderTotals[pm, Eliminate -> {3}]


$$\begin{pmatrix} 0.04 & 0.06 \\ 0.36 & 0.54 \end{pmatrix}$$

```

### A.4.9 Aggregation and transpose

Aggregator matrices have only 0 and 1 elements and the column sums are all 1. Determining row sums or border totals could also be done by multiplying with such an aggregator. The point then is, though, the construction of the aggregator. When matrices get more dimensions, the aggregator gets complicated too. It then is simpler to transpose the matrix so that the desired dimension gets on top.

Aggregate[x, y, levelspec\_:{l}]

aggregates y at levelspec. That is: transpose level to first, then x . y', then transpose back. TransposeTensor is used

AggregatorQ[x]

gives True when x is an aggregator, i.e. a matrix of only 1's and 0's, and the column sums all 1

TransposeTensor[x, y\_\_\_]

doesn't fail to transpose vectors too, e.g. {1,1}

Transposant[level\_Integer, len\_Integer]

gives the permutation (the second argument of Transpose), when the total number of levels is len while {1, level} must be transposed

TransposeToFirst[x\_List, y\_List]

transposes x, with the dimensions mentioned in y put on top

```
Aggregate[{1, 1}, {{1, 2}, {3, 4}}]


$$\begin{pmatrix} 4 \\ 6 \end{pmatrix}$$

```

```
Aggregate[{{1, 1, 1}}, pm, {3}]
```

$$\begin{pmatrix} \{0.04\} & \{0.06\} \\ \{0.36\} & \{0.54\} \end{pmatrix}$$

```
Transposant[3, 5]
```

$$\{3, 2, 1, 4, 5\}$$

```
Outer[Times, {p, pp}, {q, qq}, {r, rr, rrr}];  
MatrixForm[%]  
MatrixForm[TransposeToFirst[%%, {3}]]
```

$$\begin{pmatrix} \begin{pmatrix} p q r \\ p q rr \\ p q rrr \end{pmatrix} & \begin{pmatrix} p qq r \\ p qq rr \\ p qq rrr \end{pmatrix} \\ \begin{pmatrix} pp q r \\ pp q rr \\ pp q rrr \end{pmatrix} & \begin{pmatrix} pp qq r \\ pp qq rr \\ pp qq rrr \end{pmatrix} \end{pmatrix}$$
  
$$\begin{pmatrix} \begin{pmatrix} p q r \\ p qq r \end{pmatrix} & \begin{pmatrix} pp q r \\ pp qq r \end{pmatrix} \\ \begin{pmatrix} p q rr \\ p qq rr \end{pmatrix} & \begin{pmatrix} pp q rr \\ pp qq rr \end{pmatrix} \\ \begin{pmatrix} p q rrr \\ p qq rrr \end{pmatrix} & \begin{pmatrix} pp q rrr \\ pp qq rrr \end{pmatrix} \end{pmatrix}$$

**A.4.10 ArgMaxQ and ArgMinQ**

ArgMaxQ[x, i_Integer]	tests whether the ith position in x equals Max[x]
ArgMinQ[x, i_Integer]	tests whether the ith position in x equals Min[x]



### A.4.11 Average and standarddeviation, center and radius, and distance

<code>Average[lis]</code>	gives the average value
<code>Average[lis, n]</code>	gives the n period moving average
<code>Spread[lis]</code>	gives the maximum likelihood standard deviation of vector lis using Average and NRadius
<code>NRadius[lis, c]</code>	gives the radius of point lis with center c. If c is a number, then the point {c, c, c, ...} is taken. The default c = 0
<code>Distance[f]</code>	can be used to generate a matrix of distances between various points. The default distance measure is NRadius i.e. the Euclidean distance
<code>Distance[f : NRadius][x_List, y_List]</code>	f[x, y], thus default NRadius[x, y]
<code>Distance[f : NRadius][x_List]</code>	Outer[f, {x}, {x}, 1]
<code>Distance[Right][x]</code>	gives the rectilinear distance (Manhattan grid)
<code>Distance[Min, m_List, w_List: {}]</code>	identifies the point with the minimal total distance to the other points. Here, m is a distance matrix m[i, j] with origin i and destination j, and w are weights (default set to units). The routine thus determines the minimal column sum of (diag[w] . m)

Note: the name NRadius has been chosen to prevent conflicts with Radius in other *Mathematica* packages.

- The standard deviation is some kind of average radius from the center of the dot cloud.

`Spread[Range[1, 11]]`

$$\sqrt{10}$$

- An example is:

`Distance[][{1, 2}, {3, 4}, {5, 6}]`

$$\begin{pmatrix} 0 & 2\sqrt{2} & 4\sqrt{2} \\ 2\sqrt{2} & 0 & 2\sqrt{2} \\ 4\sqrt{2} & 2\sqrt{2} & 0 \end{pmatrix}$$

**A.4.12 Angles and polar forms**

<code>Angle[a_List, b_List]</code>	gives the angle between vectors a and b
<code>Angle[a_List]</code>	gives the angle of vector a with the horizontal plane
<code>FromPolar[a_List, opts]</code>	for a = {r, sequence of angles} gives the point in Cartesian co-ordinates.
<code>ToPolar[a_List, opts]</code>	gives {r, sequence of angles}. The angles are taken for the projections on planes. With a = {x, y, z} then the angle in the {x, y} plane is $\arctan[y/x]$ .

Option Method → Complex is default, which means that a complex notation is used. Alternative settings are ArcTan and {ArcTan, Symbol}.

**A.4.13 Lead, Lag, Dif, UnitIndex, RateOfChange**

See the Chainindex package for chained indices.

<code>Lag[x_List]</code>	gives the lagged list (Indeterminate prepended, last element dropped)
<code>Lag[x, n]</code>	moves elements n places to the right
<code>Lag[x_String]</code>	can be used as a formal representation for lagged values
<code>Lead[x_List]</code>	gives the lead list (Indeterminate appended, first element dropped).
<code>Lead[x, n]</code>	moves elements n places to the left
<code>Dif[x_List]</code>	gives $x - \text{Lag}[x]$ , and thus takes the first difference
<code>Dif[x, n]</code>	takes the n–th difference

<code>UnitIndex[x__]</code>	gives $\text{RateOfChange}[x] + 1$ . Applies to lists as well, or primarily so, and uses Division
<code>RateOfChange[x, y]</code>	gives $x / y - 1$ , while missing elements represented by Indeterminate generate Indeterminate again
<code>RateOfChange[x]</code>	gives $\text{RateOfChange}[x, \text{Lag}[x]]$
<code>RateOfChange[Drop, x (, y)]</code>	 drops the resultant first element
<code>RateOfChange[Range, x_List]</code>	 gives of the rates of change the minimum, maximum, average, and the geometric mean growth from the first to the last element
<code>RateOfChange[Lag, n, x]</code>	 gives the geometric mean rate of change of x with respect to $\text{Lag}[x, n]$ . Variables x and y are tested with VectorQ

Note: `Lag[x_String,]` as in `Lag["var"]`, can be used as a formal representation for lagged values. Replacing as in `({Lag["var"], var} /. var -> 100)` then has the proper effect.

```
Lag[{1, 2, 3, 4, 5, 6}]
```

```
{Indeterminate, 1, 2, 3, 4, 5}
```

```
RateOfChange[{1, 2, 3, 4, 5, 6}]
```

$$\left\{ \text{Indeterminate}, 1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5} \right\}$$

A.4.14 LessEqual for list

The inequality  $x \leq y$  can be quickly extended by `And @@ Thread[x ≤ y]` for lists. We want to do a bit more, though. An application of removing dominance is in section 5.9.10 Decision theory.

<code>LE[x, y]</code>	gives True when $x_i \leq y_i$ for all $i$ , otherwise False. Vectors $x, y$
<code>LE[Remove, {x}]</code>	removes all domination up to a point, list of vectors {x}
<code>LE[Transpose, {x}]</code>	transposes, removes, and transposes back

Note: See ?LE for more information.

- Using the infix format.

```
{1, 1} ~LE~ {1, 3}

True
```

- Imagine dominance as a wedge like the first quadrant.

```
LE[Remove, {{1, 1}, {1, 3}, {3, 3}, {0, 4}}]

 $\begin{pmatrix} 0 & 4 \\ 1 & 1 \end{pmatrix}$ 

Sort[{{1, 1}, {1, 3}, {3, 3}, {0, 4}}, LE]

 $\begin{pmatrix} 0 & 4 \\ 1 & 1 \\ 1 & 3 \\ 3 & 3 \end{pmatrix}$ 
```

# A.5 Data input and presentation

## A.5.1 Data handling

These data handling routines are useful for estimation and plotting.

<code>DataFilter[x__]</code>	for x a vector or a sequence of equal vectors, deletes equal positions where there are missing data. Default option is Symbol → Indeterminate
<code>Pick[x__List]</code>	takes from positions set by Options[Pick], default Take → {1, Infinity}. The latter means that the length of the smallest list is taken
<code>Pick[i (, j)] [x__List]</code>	maps Take[#, i]& or Take[#, {i, j}]& over {x}, similarly taking account of the minimal length in {x}

An example is:

```
Pick[2, 4][{10, 20, 30, 40, 50}, {100, 80, 200, Indeterminate}]
( 20 30      40
  80 200 Indeterminate )

DataFilter @@ %
( 20 30
  80 200 )
```

## A.5.2 Presentation with TableForm

In running various tests one normally has various combinations of inputs (here "labels"), and the results are classified in various dimensions too (here "keys"). This basically gives an "input-output table". Such a table can be presented by TableForm, and it can also be used to extract elements. What the user should do is to record the results of each test as Entry[label1, label2, ...] = {key1 → value1, key2 → value2, ...}.

InsideTable[Set, entry, labels]

fills the memory with entry and labels.  
The head 'entry' is used to find the various lists of rules. The labels input can be the TableForm rule TableHeadings → labels, or just labels compatible with such use, labels = {label1, label2, ...}. Each Set destroys earlier settings

InsideTable[Show, keyi]

shows keyi in TableForm  
under the earlier provided labels

InsideTable[head, labels]

creates the inside of a table of entries head[... ..]  
by applying Outer on the labels. Head need not be defined

TensorForm[x, opts]

is TableForm though for Rank 1 with TableDirection → Row (horizontal instead of vertical)

Note: See also the Bank[ ] routine.

The following example is taken from the discussion of the `Chi2`` package. When we apply the Pearson and Log-Likelihood-Ratio test to either 'normal days' or 'market days', and record the results of the test statistic and some other variable.

- This is the general principle of TableForm printing.

```
heading = TableHeadings → {{days, markets}, {Pearson, LLR}};

TableForm[InsideTable[try, heading], heading]

      Pearson      LLR
days   try(days, Pearson)  try(days, LLR)
markets try(markets, Pearson) try(markets, LLR)

InsideTable[Set, try, {{days, markets}, {Pearson, LLR}}]
```

- We now perform the tests with the different criteria. We record our experiment results - normally not stated together but rather in the section where the test was performed:

```
try[days, Pearson] = {test → 0, other → alpha};
try[days, LLR] = {test → 1, other → beta};
try[markets, Pearson] = {test → 5, other → alpha};
try[markets, LLR] = {test → 100, other → gamma};
```

- Neatly presenting all results:

<b>InsideTable[Show, test]</b>		
	Pearson	LLR
days	0	1
markets	5	100

<b>InsideTable[Show, other]</b>		
	Pearson	LLR
days	alpha	beta
markets	alpha	gamma

**A.5.3 A simple database**

The following concerns a small database routine and / or object Bank[ ].

Bank[x]	is a function for x = Take or Show, and it is a user defined object for x = Data or TableHeadings or Range
Bank[Data]	must be defined by the user, and will contain the various values
Bank[TableHeadings]	must be defined by the user for display with TableForm and for searches
Bank[Range]	must be defined by the user, for searches
Bank[Show]	the command to show the database
Bank[Take, u, BB, 70]	the command to retrieve the entry defined by these rim labels

- An example with range rims is:

```
Bank[Data] = Array[entry, {2, 3}];
Bank[TableHeadings] = {{{0, 10}, {10, 20}}, {AA, BB, CC}};
Bank[Range] = {True, False};

Bank[Show]

      AA          BB          CC
{0, 10}  entry(1, 1)  entry(1, 2)  entry(1, 3)
{10, 20} entry(2, 1)  entry(2, 2)  entry(2, 3)

Bank[Take, 5, BB]

entry(1, 2)
```

- An example that mimics the result of `InsideTable[ ]` above, is:

```
Bank[TableHeadings] = {{days, markets}, {Pearson, LLR}};  
Bank[Data] = Outer[try, Sequence @@ Bank[TableHeadings]];  
Bank[Range] = {};
```

```
Bank[Show]
```

	Pearson	LLR
days	test → 0	test → 1
	other → alpha	other → beta
markets	test → 5	test → 100
	other → alpha	other → gamma

```
Bank[Take, days, Pearson]
```

```
{test → 0, other → alpha}
```

A.5.4 Printing

This was useful for *Mathematica* 2.2.2 and still may be, for example as a programming idea.

<code>PrintPrepare[x, r__Rule]</code>	padds strings, integers and reals with blanks for printing in columns (PaddedForm seems unreliable)
<code>PaddedString[x_String, p_Integer, d_Integer]</code>	adds blanks around a string: d to the right and possibly some to the left – up to a total of p places altogether

A.5.5 Data and DataList formats

The distinction between the Data and DataList formats is discussed extensively in section 6.2. Also, the Data head is discussed in A.6 in the section on equations since this is the area where it has its best use. Here we can get acquainted with some of the notions and regard some simple common routines.

Data in DataList format are e.g. `x = {1.7, 3.2, ...}`. Individual data are accessible as `x[[i]]` for some integer `i = 1, 2, 3, ....` Data in Data format are then `Data[x] = {1.7, 3.2, ...}`, where `x` still is available as a Symbol for symbol transformations, and the individual data are accessible by `x[1999]` (i.e. for meaningful years). For the Data format, the disadvantage is that one needs the Data head to access the data list. Some small routines may help here.



<code>ToData[expr]</code>	substitutes <code>Data[x]</code> for all <code>NonAttributedSymbols</code> <code>x</code> in <code>expr</code>
<code>ToDataRule[x__Symbol]</code> or <code>ToDataRule[{x__Symbol}]</code>	makes the list of replacement rules <code>{x1 → Data[x1], ...}</code>

- `ToData` uses `NonAttributedSymbols`.

```
Data[labour] = 2 + Table[Random[], {4}];
Data[wage] = 30 + Table[Random[], {4}];

Data[labourIncome] = ToData[labour wage]

{70.4415, 74.3011, 80.6429, 67.1284}
```

- `ToDataRule` creates rules for the specified symbols.

```
subst = ToDataRule[labour, wage, labourIncome]

{labour → {2.29292, 2.46643, 2.66213, 2.19427}, wage → {30.7213, 30.1249, 30.2926, 30.5926},
 labourIncome → {70.4415, 74.3011, 80.6429, 67.1284}}
```

$$\text{labour} == -1.05 \text{ Log}[\text{wage}] + 1.5 \text{ Log}[\text{labourIncome}] + \text{error}$$

$$\text{labour} == \text{error} + 1.5 \log(\text{labourIncome}) - 1.05 \log(\text{wage})$$

```
Data[error] = (error /. Solve[%, error][[1]]) /. subst

{-0.493047, -0.420136, -0.341467, -0.523846}
```

### A.5.6 Tabulate

If you want to look at a list of data, then it may be useful to print these in columns. This requires a partition, and the partition again requires that you add dummy values. `Tabulate[Sort[x], n]` is a typical application for looking through a list of words.

<code>Tabulate[x_List, n_Integer]</code>	gives <code>x</code> in transposed form, with <code>n</code> columns
<code>PartitionKeep[x_List, n_Integer]</code>	calls <code>Partition[x', n]</code> where <code>x'</code> is <code>x</code> but with sufficient elements appended

`Tabulate` uses `PartitionKeep`, that itself takes the indicator of missing data from `Symbol` in `Options[Data]`.

```
Tabulate[Table[Random[], {11}], 6]

( 0.651056 0.746624 0.458959 0.884892 0.0575874 0.587043
  0.856399 0.544312 0.217216 0.535312 0.961941 Indeterminate )
```

## A.6 Equations and models

### A.6.1 Plain symbols

The following words exist as plain symbols. There are no definitions for them, but they can be used for example as options in {symbol → value} or function[symbol] statements. Since these are declared in the common packages, name conflicts are prevented between different packages.

Coefficients	Equations	StartValues
Constraint	Ratio	

In *Mathematica* 4.0 there is an undefined Value symbol in the System` context that replaces the one of the Pack.

See for example Bank[ ] for an application of the unassigned Data symbol.

### A.6.2 Simple symbols with conventions

The following have a tiny definition and a stricter convention:

Autonomous	Expected	Solution
Data	ListOfSymbols	Surprise
Expectation	Problem	

Autonomous["f"] could be used as the autonomous term of a function. ListOfSymbols[model] would specify the legend for the model. Routines can exploit the use of a conventional structure.

Expectation[ ] will be used in statistics, Expected[ ] in economics. The best convention is to use strings, as in Expected["Inflation"]. In that case the assignment of Inflation = 5 does not result into Expected[5]. Surprise has been defined in that way.

<code>f[x_] := ... + Autonomous["f"]</code>	the autonomous part can be adjusted without redefining the function. You could use any variable of course, but Autonomous[..] would be of help in organising models.
<code>Parameter["y", "x" (, i)]</code>	user defined, identifies the particular parameter associated with variable x in the function call of y, using index term i
<code>ListOfSymbols[expr]</code>	{{symbol, explanation}, ....}
<code>Surprise[x_?StringQ]</code>	gives the surprise equation for the variable x, with the surprise value given as Surprise[ToString[x]]

Note: The use of Strings here is advised since the terms are subjected to replacements.

**Surprise[x]**

$$\text{Surprise}(x) == x - \text{Expected}(x)$$

Data has some options assigned to it. Calling `Clear[Data]` still keeps those options available. The `Symbol` option denotes the indicator for missing data, the `Data` option indicates what kind of data format is being used. Settings are `Data` for the `Data` format and `List` for the `DataList` format. The data format is that `Data[x]` would generate the list of data associated with symbol `x`. See sections 6.2 and A.5 for more discussion.

**Options[Data]**

{Symbol → Indeterminate, Data → Null}

`Problem[ ]` could signify the current problem that you are working on, and `Solution[ ]` can be used for results, as in `Solution[ ] = Solve[f[x], x]`. An application of this is in LP.

### A.6.3 Turning inequalities into equations

The following originates from a linear programming setting, but seems best mentioned here. You might want to solve a set of inequalities by introducing slack and then call `Solve`.

<code>ToEquation[ LHS &lt;   ≤   ==   ≥   &gt; RHS (, label)]</code>	turns the inequality to an equation by adding <code>Slack[label]</code> and/or a small positive <code>EPS</code>
<code>ToEquation[ineq_List (, label_List) (, opts)]</code>	sets the options and then <code>MapThread</code>
<code>RandomLabel[n_Integer]</code>	makes a <code>Symbol</code> of <code>n</code> random capitals from A–Z

Note: `ToEquation` has options (default): `NumberQ` (False), `N` (2) and `Equal` (None). These are discussed in the text. Note: Thus for a single inequality one has to set the options oneself. Note: With `N` → 3, `EPS` will not be a label.

You will generally have your own labels for the `Slack[label]` identifiers, but if no label is given then a random label is generated, with option `N` controlling how many capital letters or digits (default 2). You can choose between numeric or alphabetic labels with option `NumberQ` (False). For lists of equations, `NumberQ` → `True` causes the use of just the Range 1, ..., `n`; while for random alphabetic labels care is taken that these differ. These labels are also available in `Results[ToEquation]`.

With `ToEquation[LHS == RHS]` you might guess that nothing need be done. Wrong; this is only the default setting, with `Equal` → `None` (though the routine still introduces the 0 to emphasise that nothing is done). Alternative settings still introduce `Slack[label]`.

- 0: {`Slack[label] == LHS - RHS`, `Slack[label] == 0`},
- All: {`Slack[label] == LHS - RHS`, `Slack[Label] == RHS - LHS`, `Slack[label] == 0`}
- all other values: The first two of All

- An example for a list of statements is:

```
ToEquation[{x ≤ y, t > u, w == v}]  
  
{Slack(QY) == y - x, Slack(YR) == -EPS + t - u, 0 == w - v}
```

- The list of labels still includes the dummy reference to equality slack.

```
Results[ToEquation]  
  
{QY, YR, KQ}
```

Note that you can transform linear equations into matrices by the routine `LinearEquationsToMatrices` of `LinearAlgebra`MatrixManipulation``.

**A.6.4 Variables, variates, symbols, undefined and non-attributed symbols and terms**

This section serves the automated recognition of variables in equations.

- *Mathematica's* `Variables[ ]` only selects from polynomial Equations.

```
Variables[1 + x^2 + 100*y^6]  
  
{x, y}
```

- `Variables[ ]` does not select from equations like the following.

```
eqs = {y1 == 1.2 Log[y2] + x1[t - 1] + v[h],  
      Exp[y2] + y1 == 3.2 x2[t - 2] + 4.5 y1[t]}  
  
{y1 == 1.2 log(y2) + v(h) + x1(t - 1), y1 + e^y2 == 3.2 x2(t - 2) + 4.5 y1(t)}  
  
Variables /@ eqs  
  
{}, {}
```

<code>Symbols[expr]</code>	gives the symbols in <code>expr</code>
<code>UndefinedSymbols[expr]</code>	selects the Symbols from <code>expr</code> with <code>DefinedQ False</code>
<code>NonAttributedSymbols[expr]</code>	gives the symbols <code>x</code> for which <code>(Attributes[x] == {})</code>

The use of the `Attributes[var]` is essentially a trick. Many mechanisms have been tried, but this trick is the best there is. The trick is not without reason. *Mathematica's* `System`` variables will generally have attributes, and well-defined functions too. For variables, conventionally, there are no attributes.

- `Symbols[ ]` also generates *Mathematica* components of an expression.

**`Symbols[eqs]`**

`{e, Equal, h, List, Log, Plus, Power, t, Times, v, x1, x2, y1, y2}`

- These don't, but they differ when there has been a definition. To produce this difference, let us give a definition for `x2`:

**`x2[1998] = 0.12;`**

**`UndefinedSymbols[eqs]`**

`{h, t, v, x1, y1, y2}`

**`NonAttributedSymbols[eqs]`**

`{h, t, v, x1, x2, y1, y2}`

Above symbols are not really the variables in the equations. These are  $x_1[t-1]$  and  $v[h]$  too. In making symbolic models, we wish to employ expressions like `Parameter["Consumption"]` or `Autonomous["Investment"]`. All this means that function calls need to be recognised too.

<code>FunctionCalls [expr, x]</code>	gives the list of $x[y]$ for any sequence $y$
<code>FunctionCalls [expr]</code>	gives the list for any $x$ (including Plus, Times, ...).
<code>FunctionCalls [expr, UndefinedSymbols]</code>	only for undefined $x$
<code>FunctionCalls [expr, NonAttributedSymbols]</code>	only for nonattributed $x$
<code>Variates [expr, t]</code>	gives a list of variates in $t$ . Symbol $x$ is a variate in $t$ if $x[t]$ occurs for a single $t$ . <code>Variates [label]</code> can be used to store such lists too

- Note that we define variates here as single argument functions, of specified symbol.

`Variates [eqs, t]`

$\{y_1(t)\}$

- If we regard function calls indiscriminately, we get a too large set. For example, the equations themselves are a function call too, i.e. `Equal [...]`.

`FunctionCalls [eqs]`

$\{\log(y_2), 1.2 \log(y_2), v(h), t-1, x_1(t-1), 1.2 \log(y_2) + v(h) + x_1(t-1),$   
 $y_1 == 1.2 \log(y_2) + v(h) + x_1(t-1), e^{y_2}, y_1 + e^{y_2}, t-2, x_2(t-2), 3.2 x_2(t-2),$   
 $y_1(t), 4.5 y_1(t), 3.2 x_2(t-2) + 4.5 y_1(t), y_1 + e^{y_2} == 3.2 x_2(t-2) + 4.5 y_1(t),$   
 $\{y_1 == 1.2 \log(y_2) + v(h) + x_1(t-1), y_1 + e^{y_2} == 3.2 x_2(t-2) + 4.5 y_1(t)\}$

- This is the best approach

`FunctionCalls [eqs, NonAttributedSymbols]`

$\{v(h), x_1(t-1), x_2(t-2), y_1(t)\}$

The union of `Symbols` and `FunctionCalls` gives us "Terms".

<code>UndefinedTerms [expr]</code>	selects symbols and functional calls of them
<code>NonAttributedTerms [expr]</code>	gives non-attributed symbols and functional calls of them

Thus, we now have a general routine that can detect symbols and symbolic functional calls, and that is smart enough to see that the variables in " $y[t] == x[t] + c$ " are only  $y[t]$ ,  $x[t]$  and  $c$  (and not  $y$  and  $x$  separately too, and neither  $t$ ).

**UndefinedTerms**[eqs]

{t, y1, y2, v(h), x1(t - 1), y1(t)}

**NonAttributedTerms**[eqs]

{y1, y2, v(h), x1(t - 1), x2(t - 2), y1(t)}

### A.6.5 Variations on Solve

There are a some solution approaches built around `Solve[ ]`.

<code>SolveFormally</code> [eqns, vars (, elims), options]	
same format as <code>Solve</code> . This replaces numbers by dummy variables, applies <code>Solve</code> with options, and substitutes the numbers back. <code>Solve</code> sometimes appears stronger when no numbers are present.	
<code>SolveGeneral</code> [x, {t___}]	means <code>If</code> [x, (*true*) {{t → All}}, (*false*) {{t → Null}}, <code>Solve</code> [x, {t}]] ; x can be single or a List
<code>Solve2List</code> [x_List, {t___}]	means <code>SolveGeneral</code> [ <code>Simplify</code> [ <code>Apply</code> [ <code>Equal</code> , x]], {t}], sets the two elements of x to an equation and solves for t
<code>SolveLHS</code> [x__]	solves equations x for the left hand variables
<code>SolveRecurse</code> [x__]	applies rules to equations, and equations that generate a number are again used for replacement

■ Compare the different results:

```

Solve[{y != y + 0 z, z + y == 5}, {y, z}]
SolveGeneral[{y != y + 0 z, z + y == 5}, {y, z}]

{}

{y → Null, z → Null}

Solve[{y == y + 0 z}, {y, z}]
SolveGeneral[{y == y + 0 z}, {y, z}]

{}

{y → All, z → All}

```

- This equalises the lists first.

```
Solve2List[{{x, z + x}, {6, 2}}, {x, z}]
{{x → 6, z → -4}}
```

### A.6.6 SolveFrom

SolveFrom[ ] is a small but powerful application of Solve[ ] that has found its way into many packages in the Economics Pack.

Frequently, (a) a set of equations remains the same, (b) but one has different selections of knowns and unknowns. Given the selection of the knowns, the computer should be able to find the unknowns. Solve[ ] itself is not so smart, SolveFrom[ ] is.

- Regard for example the two physics equations that many will know, with 4 variables (taking  $c$  as a constant).

```
model[{x___Rule}] := SolveFrom[{F == m a, E == m c^2}, {x}]
```

- Then investigate different model[{...}] solutions. Output has the following structure: {model with replaced values, the substitution rules used, the solution}. Joining the latter two sublists gives all settings.

```
model[{m → 10, c → 3 10^8, E → 10^15 F}]
{{F == 10 a, 1000000000000000 F == 90000000000000000},
 {m → 10, c → 300000000, e → 1000000000000000 F}, {{a → 90, F → 900}}}
```

Note: You can use SolveShow[Last[%]] to present the solution.

```
SolveFrom[eqs_list, {x___Rule}, (elims___List,) opts___Rule]
```

determines the equations to solve as neweqs = eqs /. {x},  
finds the remaining variables in these neweqs, and solves for these. The  
elims list have two functions: elims1 is passed on as the elims of Solve,  
elims2 are only eliminated from the variables that the routine has noted. For example,  
if variables have dimensions such as Meter or Second, then these are neither  
to solve for nor to eliminate. Output consists of neweqs and the solution

```
SolveFrom[TableForm, results___List]
```

takes the outputs from various calls of SolveFrom and gives a  
TableForm presentation of these, including the different input parameters {x}

Note: See Model' for the use of lags.



SolveFrom has two important options.

<i>option</i>	<i>typical default value</i>	
Find	NonAttributedTerms	routine to find the variables
Rule	{Equal → List}	alternative <i>add</i> e.g. {Log → List, ....}

The Find method gives the routine that finds the variables to solve. Advised is Find → NonAttributedTerms. If the settings are Variables and UndefinedSymbols and you have an equation  $y[t] == x[t] + c$ , then  $y[t]$  and  $x[t]$  will not be recognised as variables to solve for. Variables will find none, and UndefinedSymbols only  $y$  and  $x$ . If you use UndefinedTerms, then  $y[t]$  will not be found if there is a definition, such as  $y[2000] = 1/2$ . You have the option Rule for further control. E.g. Rule → {...,  $t \rightarrow \{\}$ , ...} will cause that  $t$  in  $y[t] == x[t] + c$  is not included in the variables to solve for. (Be sure, here, that  $y[\{\}]$  does not evaluate to something.) If you use the setting Variables, then you have to use e.g. Rule → {Equal → {}, Log → {}} to get rid of non-polynomial terms like Log[ ]. Note that the setting Equal → List must be included for Variables, in order to remove the heads of the equations.

A.6.7 Presenting and selecting solutions

For presenting results:

<code>SolveShow[r_List, opts___Rule]</code>	
	presents the results $r$ of Solve in TableForm, sorted. Missing values in certain solutions are represented by a dot. Options are passed on to TableForm

Solve generates complex solutions in general, and also can generate Root expressions when all numbers are rational (when output remains in rational form too). These Root expressions however don't test in RealQ or NonNegative. For the selection of real roots the Root expressions have to be set to a number.

<code>RealRoots[sols]</code>	selects the real roots (sols provided by Solve). It uses RootToN
<code>RealRootQ[x, y]</code>	tests whether $y$ is noncomplex, if so then output is $x \rightarrow y$ , else False
<code>RootToN[x]</code>	replaces Root[...] terms by a number

- If you want to find the percentage rate of interest by which a sum of \$1000 grows to \$2000 in 10 years, and if you use Solve, then there are a lot of root expressions. (Output suppressed.)

```
sol = Solve[(1 + r / 100)10 1000 == 2000, r];
```

```
RootToN[sol]

{{r → -207.177}, {r → 7.17735}, {r → -133.12 + 101.932 i}, {r → -66.8804 - 101.932 i},
 {r → -133.12 - 101.932 i}, {r → -66.8804 + 101.932 i}, {r → -186.708 + 62.9973 i},
 {r → -13.2917 - 62.9973 i}, {r → -186.708 - 62.9973 i}, {r → -13.2917 + 62.9973 i}}
```

- RealRoots calls RootToN and selects only the real roots:

```
RealRoots[sol, Messages → True]

RealRoots::act: Real roots taken only. Enter Dump[RealRoots] to see all.

{{r → -207.177}, {r → 7.17735}}
```

- A dump (not evaluated).

```
Dump[RealRoots]
```

As we are only interested in nonnegative rates of interest:

NonNegativeSolution[s, x]	selects solutions with $x \geq 0$ , for s the output of Solve
SelectNonNegativeSolution[s, x]	does not give the whole set of nonnegative solutions, but picks out only the x

Note: x can be one variable or a list of variables. If no x is mentioned, then all variables are taken. S is first reduced to real roots and all Root expressions are turned into real numbers. Thus the solutions of the non-selected variables are also real, though they may still be negative. Use option Messages → True if you want a message that other solutions have been deleted.

- This gives us the nonnegative percentage rate of interest.

```
NonNegativeSolution[sol]

{{r → 7.17735}}
```

A.6.8 Dynamics

The ideas for these basic tools have been distilled from Eckalbar's chapter in Varian ed. (1993). NSolveLinearDynamics has the 'NSolve' in its name since it is best for numerical solutions; it uses NestList.

```
NSolveLinearDynamics [M_List, x0_List, T_Integer]
```

computes the linear dynamic system  $x[t] = M^t x_0$  for  $t = 0, \dots, T$

```
NSolveLinearDynamics [M_List, x0_List, e_List]
```

sets  $x[t] = M x[t-1] + e[[t]]$  and then gives `Array[x, Length[e]]`

```
XXPlusSpace [x_List, tr:True]
```

projects  $x$  in the  $\{x[[i]], x[[i+1]]\}$  space.

If True, output is  $\{\{x[[1]], x[[2]]\}, \{x[[2]], x[[3]]\}, \dots\}$  (sequential).

If False,

output is  $\{\{x[[1]], x[[2]]\}, \{x[[2]], x[[2]]\}, \{x[[2]], x[[3]]\}, \dots\}$  (block sequential)

Note: `XXPlusSpace` stores in `Results[XXPlusSpace, tr]`. See `PhaseDiagram[]`.

See the discussion of the applied general equilibrium models, and especially the Leontief models, for applications of `NSolveLinearDynamics`.

### A.6.9 Setting values

You may wish to set variables to zero.

```
ReplaceByZero[x]            gives  $x \rightarrow 0$ . Listable
```

```
ReplaceByZero[{a, b, c}]
```

```
{a → 0, b → 0, c → 0}
```

- A bit overdone. We already had the following routine in section A.2.5.

```
SequenceToRules[{a, b, c}, 0]
```

```
{a → 0, b → 0, c → 0}
```

# A.7 Programming

---

These routines concern the programming and running of routines.

## A.7.1 Debugging

When debugging a routine, it is often useful to print an intermediate result of some location. Sometimes, one even wishes to know whether a location was found at all.

<code>Kilroy[x]</code>	messages and prints x, where x may be Blank[]
------------------------	---

An example is:

```
Module[{}, If[a == a, Kilroy[True], Kilroy[False]]]

Kilroy::Was : Here: True

True
```

## A.7.2 Input Options

If you define your base options then it is possible to always return to them.

<code>ResetOptions[x]</code>	sets Options[x] = ToExpression["Options\$" <> ToString[x]], thus sets the options to a predefined value, presumably given at startup in a package or saved inbetween
------------------------------	--

- The following three steps show how to set the options, and change and reset them.

```
Options[myfunction] = Options$myfunction = {Method → Automatic,
MaxIterations → 10}

{Method → Automatic, MaxIterations → 10}

SetOptions[myfunction, Method → None]

{Method → None, MaxIterations → 10}

ResetOptions[myfunction]

{Method → Automatic, MaxIterations → 10}
```

```
CurrentOptions[symbol, {news__Rule}, opts]
```

applies ReplaceRule to Options[symbol] and the news

- Without doing a SetOptions, you still can produce a list of updated options.

```
CurrentOptions[myfunction, {Method → None}]
```

```
{MaxIterations → 10, Method → None}
```

The following is adapted from and inspired by Utilities`FilterOptions`, and supposed to be more flexible.

```
OptionsFilter[symbol, opts]
```

returns a sequence of those options that are valid for  
symbol. The options may be interchanged with *lists* of options

```
FilterSetOptions[symbol, opts]
```

sets the options after first filtering out the irrelevant ones

### A.7.3 Output Results

The opposite for input Options are output Results. Output often is singular, like  $1 + 1$  gives 2, but for complexer problems one wishes to know more. For a function  $f[x]$ , the internal routine code usefully contains a line `Results[f] = {symbol1 → value1, symbol2 → value2, ...}`, for example as the last line or the penultimate one. Similarly, the label Dump can be used to dump information that still might be useful on a rare occasion.

```
Dump[x]                    a generic place for routines to dump information (e.g. for error checks)
```

```
Results[f]                is a list of rules to store results of keyword f
```

Results[ ] is just a simple object. Once it exists, we can do more with it. If we predefine a Results\$myfunction, then we can proceed along the same lines as with ResetOptions[ ].

SetResults[f, {forf\_\_Rule}, opts]

sets rules in Results[f] to new values supplied in forf.  
The opts are for ReplaceRule. If Clear → True is given,  
with or without forf–rules, then there is a ResetResults (first).

ResetResults[f]                      resets Results[f] to either {} or a predefined Results\$f.

Routines may call on Results instead of recomputing results.

HoldShell[f, expr\_Hold, opts]

allows the selection of an earlier result  
from Results[f] rather than doing ReleaseHold[expr]

ReEvaluate      Option that may indicate whether routines should re–  
evaluate functions and options or that they can take existing values

The HoldShell[ ] is applied in the Economic`Optimise` package.

The options for HoldShell[ ] are a bit complicated. Typical defaults are:

option	default value	
ReEvaluate	True	activate options ReleaseHold and SetResults
ReleaseHold	True	do not use Results
SetResults	False	do not update Results[f]
Select	Automatic	select Results keywords from expr that has format Hold[{key → g[y], ...}], i.e. a list of rules within Hold.
Union	True	this is passed on to RuleReplace

And alternatives are:

option	alternative	
ReEvaluate	False	override options ReleaseHold and SetResults, prevent reevaluation whenever possible, use all earlier Results whenever possible, do ReleaseHold only when the Results[f] give a Null answer, and store new results
Select	{key, ...}	evaluation of expr generates {key → x, ...} such that Results[f] can be updated

#### A.7.4 Memory constrained execution

In older versions of *Mathematica* the memory constraint was stronger, but then again, we may grow more endaring in our problems.

```
MemoryConstrainedQ[proc, limit_Integer]
```

execute *proc*, and return True if the memory constraint applied

## A.8 Context

---

These routines help to better take advantage of *Mathematica*'s excellent context structure.

### A.8.1 Listing the contents of a context or package

Contexts are mysterious when you don't know what's in there. If you load a package, you want to know what you are getting. These routines help you to find out.

`Contents[x_context | ListOfContexts, r__Rule]`

shows the names in the (list of) contexts.  
The default list of contexts is `$ContextPath` exclusive of the System context. When `NeedsQ`  $\rightarrow$  True (default), Needs is applied, i.e. a context is loaded from package if it does not exist yet

`NeedsQ` option of `Contents`. If `NeedsQ`  $\rightarrow$  True (default), then Needs is applied to the contexts, so that nonexistent contexts are gotten from packages. For file locations you may set the `$Path`

`Load[file_String]` gets file.m and shows the contents of the first context of `$ContextPath`. If file contains ``` then `ContextToFileName` is used, otherwise `ToFileName`

An example is:

```
Contents["Economics`Pack`"]
```

```
?Economics`Pack`*
```

```
Economics           InstallEconomicsPalettes
EconomicsPack       $Economics
EconomicsPalettes
```

### A.8.2 Listing the contents of a routine

Documentation of software is difficult. Often a description only gives a general idea, but not the specifics. *Mathematica* allows you to look at the code by evaluating `??routine`. But then you get the context labels of the variables too. `ShowPrivate[ ]` gives a peek without these labels.



```
ShowPrivate[x, q:"??", priv:"Private`"]
```

for x that can be of Symbol, String or HoldPattern[expr]. Checks whether there exists a private context, enters it, evaluates, and leaves again. In this manner Private context heads are not shown in output. For Head[x] = Symbol or String the default item is "??", but can be "?" while priv might be "". For x = HoldPattern[expr] output is Information[expr].

Note: Enter "??" and "?" and "Private" as Strings.

- For example:

```
ShowPrivate[Kilroy]
```

```
Cool`Manager`Private`
```

Debugging tool to temporarily put into loops. Kilroy[x] messages and prints x. If x is Blank, then the Kilroy statement.

```
Kilroy[x_:"Kilroy was here"] := (Message[Kilroy::Was, x]; Print[x])
```

### A.8.3 Looking for expressions

The converse of looking for the contents of a context, is to start with an expression and to look what context it belongs to. The following extends on Context[expr] and ?expr and ??expr.

```
Query[q_String, (C,) x___ (Strings || ListsOfStrings) _inAnyOrder]
```

makes the outer product of the strings||Lists, concatenates, prepends q, appends \*, and applies ToExpression. By taking q = "?" or "??", one can look for variables in contexts. Mentioning C includes the \$ContextPath exclusive of System`.

Note: In *Mathematica* version 3.0.0 the printing order is a bit mixed up.

- Some examples are (not evaluated here):

```
Query["??", {"Add", "Division"}];
```

```
Query["??", C, "Add"];
```

### A.8.4 Back Apostrophe

Working with contexts requires the use of "'", a symbol that sometimes reads awkward as in "f" or that is just forgotten. In interactive settings *Mathematica* will give error messages. The following routines can be useful in packages.

Cure [ <i>c_String</i> ]	adds a ` if c doesn't have one. Cure comes from ContextSure
DeCure [ <i>c_String</i> ]	drops a ` if c does have one
CtE [ <i>context_String</i> , <i>var_String</i> ]	
	concatenate these strings and then evalutates ToExpression
DataPackage [ <i>dir_String</i> , <i>ps_String</i> ]	
	gives dir<>ps<>.m ; If dir contains ` then ContextToFileName is used, otherwise ToFileName

**A.8.5 \$ContextPath and BeginContext**

A context declaration `Begin[]` does not adjust the `$ContextPath`. `BeginPackage[]` does that. But `BeginPackage["a"]` limits the contexts to only `System`` and the explicitly mentioned other contexts. Occasionally we may want to begin a context while updating `$ContextPath`. The following does so, while actually keeping track of the history of these updates.

<code>BeginContext [<i>x_String</i>]</code>	is similar to <code>Begin[x]</code> and then <code>Prepend[\$ContextPath, x]</code> BackApostrophe `` may be appended to x
<code>BeginContext [N]</code>	gives the number of additions to <code>\$ContextPath</code>
<code>BeginContext [Plus]</code>	increases <code>BeginContext[N]</code> and updates the history
<code>BeginContext [<i>n_NumberQ</i>]</code>	gives the history, i.e. the <code>\$ContextPath</code> at the n–th <code>BeginContext[ ]</code>
<code>BeginContext [ Needs, <i>x_String</i>]</code>	applies <code>Needs[x]</code> , while updating <code>BeginContext[N]</code> with <code>BeginContext[Plus]</code>

By setting `$ContextPath = BeginContext[n]` one resets the list of known contexts to an earlier phase in history.

**A.8.6 Controlling definitions and adding usage to a key**

The use of `Key[]` and `AddedUsage[]` allows you to extend the definitions of keywords in a context or package.

In everyday English, words change their meaning depending upon the context of use. In *Mathematica*, the definition and usage is stricter. Still, we can allow for the use of the same symbols in different ways, as long as these usages are clear and don't conflict. An example is the extension of the use of `Center` in the `Tool`` package:

**?Center**

Center is used to specify alignment  
in printforms such as ColumnForm and TableForm.  
\*\*\* AddedUsage by Cool`Tool`: \*\*\*  
Option of Logistic, for the  
upward shift of the Logistic. Default is 0.

These extensions would normally be defined in packages and users will hardly notice what has happened.

In the following, we use *key* or *keyword* to stand for the symbolic head for a function or operation.

Key[AddedUsage, keyword_Symbol, txt_String, unknownProtection_: False]	
is used as the RHS of keyword::usage = RHS. It adds txt to the current usage statement if \$Context does not yet occur in that usage statement, while item is also included in the AddedUsage[\$Context]. One can enter True if one wishes to have warning messages on protected symbols	
AddedUsage[context]	is a list of keywords for which the usage is extended. Context is where the extension took place.
AddedUsage[]	lists the extended keywords for the whole \$ContextPath
AddedUsage[x_List]	puts the elements of x into the AddedUsage[\$Context]. Normally you would put this statement at the end of a package where the use has been extended

Note: \$Context is the current context, in a package the package itself.

■ Adding the usage of Center again:

**Center::usage = Key[AddedUsage, Center, "Just an example"]**

Center is used to specify alignment in printforms such as ColumnForm and TableForm.  
\*\*\* AddedUsage by Cool`Tool`: \*\*\*  
Option of Logistic, for the upward shift of the Logistic. Default is 0.  
\*\*\* AddedUsage by Global`: \*\*\*  
Just an example

An even stronger control is to record the history of definitions, and to be able to recall a certain definition.

<code>Key[Install, context_String, x_List]</code>	installs <code>Key[context]</code> as a list of Strings for the elements in <code>x</code> , and sets its counter <code>Key[context, N]</code> to zero.
<code>Key[context, N]</code>	the counter for the context
<code>Key[context, name, counter]</code>	gives the definition of name at counter as a String
<code>Key[Set, context]</code>	increases the counter and sets the elements at the current definitions
<code>Key[Take, context, int_Integer: 0, clrq: True]</code>	submits these definitions to <code>ToExpression</code> , which executes them. If <code>int</code> is 0 then the current counter is taken, a negative value is regarded as steps back, and otherwise one has to provide a nonnegative integer. Default <code>clrq = True</code> , meaning that the variables are first Cleared
<code>Key[Begin, c, c2]</code>	records, with <code>{Key[c, N], Date[]}</code> , when context <code>c2</code> was started and what counter applies

An example is given in the example notebook `ContextQ.nb`

### A.8.7 Working in contexts

One would normally work in contexts by using `Begin[ ]`, `BeginContext[ ]` or `BeginPackage[ ]`. Occasionally one could try to work directly.

<code>AppendToInContext[context_String, var_String, value_String]</code>	appends value to var in the context
<code>ClearInContext[c_String, x_String]</code>	tries to clear variable x in context c using <code>Unset</code>
<code>ClearInContext[c_String, x_List]</code>	first maps a concatenation and then applies <code>Clear</code>
<code>ClearInContext[c_List, x_String]</code>	maps <code>ClearInContext</code> to the list of contexts using <code>Unset</code>

There must be a warning on `ClearInContext`. It is not always sure that the procedure works. It seems that removing is more robust.

`ClearFunctionByString[c_String][x_String]`      may be tried when `Clear[]` doesn't work

`PutIntoContexts[y_String, f_String, x_String, c_ListOfStrings(, opts)]`

maps  $y = f[x]$  into the contexts mentioned in `c`.

By default `x` is in context, and there are no shadowing messages

`ArgumentQ`      option for `PutIntoContexts`, standard `True`,  
holding that the argument is also contexted

`Scoring[context_String, scorevar_String, func_, score_:"Null"]`

gives score to the `scorevar` in the positions determined by the function on the context

`Discard[context_String]`      goes to `Global``,  
clears and removes the mentioned context.  
NB. if ``` lacks then this is appended

`KeepVariables[context_String, variableNames_List, keep_List]`

removes variables from a context` by mentioning what is to be kept

`RemoveVariables[context_String, variableNames_List]`

removes variables from a context` by mentioning what is to remove

Note: If `CleanSlate`` has been called, or `ResetAll`, then `CleanSlate[context]` is preferable.

### A.8.8 ShadowHull

Working with contexts may cause messages on name shadowing. Turning these messages off universally is a bad idea. Turning these messages off locally for each routine, and turning them on after completing, is better, but also a lot of programming. Using a hull is a compromise.

`ShadowHull[{head, rule}, procedure]`

may suppress all messages within the procedure, and notably the name shadowing when the procedure applies `ToExpression` with contexts. Normally `head` is the name of the calling routine. `ShadowHull` requires that `Options[head]` or `rule` contain `MessagesQ → False` (suppress) or `True`. Note that the rule can be used in the procedure again.

<code>MessagesQ</code>	option for <code>ShadowHull</code> . If <code>MessagesQ → False</code> , then all Messages, notably of shadowing, are repressed. Only <code>True</code> or <code>False</code> are allowed.
<code>Pralse[x_String]</code>	'print False', in effect Message <i>x</i> , and <code>Throw[False]</code>
<code>Pralse[x_String, Print]</code>	prints instead of a message, e.g. when there is a <code>ShadowHull</code>

■ Setting up.

```
Options[shafunc] = {MessagesQ → False};

shafunc[y_, r___Rule] := ShadowHull[{shafunc, r},
    Module[{x}, x = ToExpression[y];
    If[ x == 1 / 0, Infinity, x] ]]
```

■ A run within the shell.

```
shafunc["1 / 0"]
```

$\infty$

■ A test run that still allows messages.

```
shafunc["1 / 0", MessagesQ → True]
```

*Power::infy : Infinite expression  $\frac{1}{0}$  encountered.*

*Power::infy : Infinite expression  $\frac{1}{0}$  encountered.*

$\infty$

# A.9 Graphics

We distinguish *graphics primitives* (GP), *graphics objects* (GO) and *plots*. Standard *Mathematica* graphics primitives are `Line[ ]`, `Circle[ ]`, etc. The general structure is that `GO = Graphics[{ GP }]`, and that a plot comes from `Show[GO]`.

## A.9.1 Asymptots

<code>Asymptot[{x, y}, direction(, options)]</code>	plots horizontal and vertical lines to {x, y} if direction = True then the asymptot is vertical if direction = False then the asymptot is horizontal if direction = Automatic then both the default option is <code>DashingValues → {.02, .02}</code>
<code>DashingValues</code>	Option of <code>Asymptot</code> , fills <code>Dashing[...]</code>

See the "Demand and Supply Cross" diagram below.

## A.9.2 Graphics primitives

Many of the following primitives consist of lists of primitives.

`Arcs[c, f, g]` gives a graphics primitive of two arcs with the angle made from center  $c = \{cx, cy\}$  and legs towards  $f = \{fx, fy\}$  and  $g = \{gx, gy\}$ . `Results[Arcs]` gives the size of that angle

`LineRasterGP[xs_List, ys_List, {bx, ex}, {by, ey}]`

gives a graphics primitive of a raster of lines at x's and y's positions, within the rectangular area determined by begin and end values  $\{bx, ex\}$ ,  $\{by, ey\}$

`BlockGP[w, h, a, c]` gives a graphics primitive of a 'block', with width  $w$ , height  $h$ , angle with the horizontal axis of  $a$  (default 0), and the top centered at  $c$ , default  $\{0,0\}$

`Parallelogram[a, b, c, opts___Rule]`

is a graphics primitive of a parallelogram through the three points. If option `Arcs`  $\rightarrow$  `True` (default `False`), then the angle at  $a$  is 'arced'. Use as `Show[Graphics[Parallelogram[...]]]`

`DashedCircle[c, r, {th1, th2}]`

gives a dashed circle arc with center  $c$ , radius  $r$ , for angle  $th1$  till  $th2$ .

`DashedCircle[x_List, c_:{0, 0}]`

with  $x$  a list of pairs  $\{pi, qi\}$ , gives the dashed circle arcs with center  $c$  through those points

`TriAngle[a, b, c, opts]` is a graphics primitive of a triangle of the three points. If option `Arcs`  $\rightarrow$  `True`, then the angle at  $a$  is 'arced'.

Note: When displaying, use `Show[Graphics[TriAngle[...]], AspectRatio  $\rightarrow$  Automatic]` to get a proper print. Note also that the name differs from `Triangle` in `MultipleListPlot`.

### A.9.3 An application of these

The following application of above graphics primitives concerns a more philosophical point. There is the paradoxical situation that we in some cases take great pains to prove something that in another point of view is merely a matter of definition.

- Pythagoras convinces us that we have to *prove* that  $c^2 = a^2 + b^2$
- For a circle, it holds *by definition* that  $c^2 = a^2 + b^2$



We encounter the same "prove or define" paradox in economics. The example provided by Pythagoras's theorem exemplifies the situation, and clears the road for a good economics discussion.

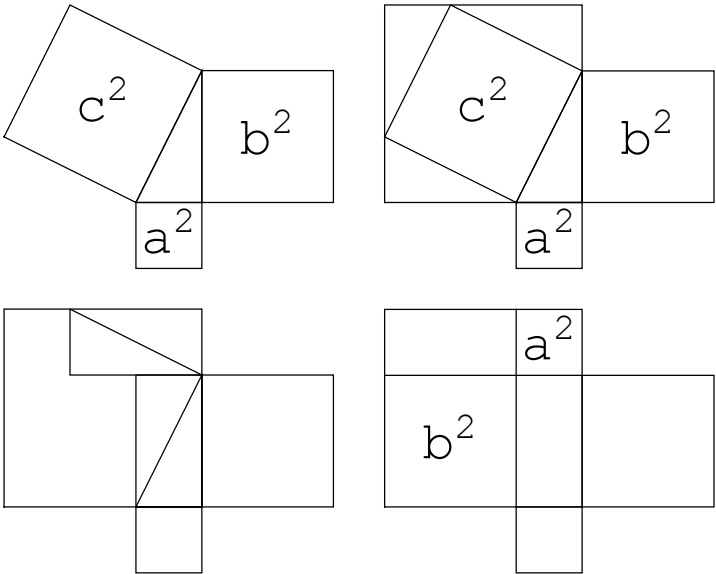
**Economics [Pythagoras]**

Pythagoras[*a*, *b*, *opts*]

gives the graphical proof of Pythagoras's theorem for a straight angled triangle with sides *a* and *b*, i.e. the theorem that the square of the hypotenusa equals the sum of squares of the sides ( $c^2 = a^2 + b^2$ ).

■ Prove the theorem for  $a == 1$  and  $b == 2$ .

**Pythagoras[1, 2]**



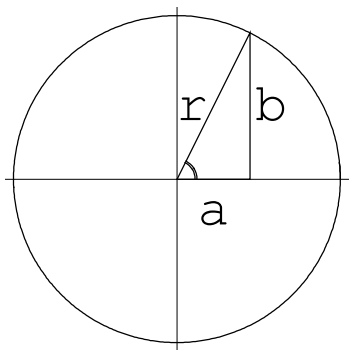
CircleDefinedByPythagoras[a, b, opts]

gives the circle with center {0, 0} through {a, b}.

The radius is given by Pythagoras's theorem  $r = \text{Sqrt}[a^2 + b^2]$ .

For the angle  $\alpha$  we define  $\text{Sin}[\alpha] = b/r$  and  $\text{Cos}[\alpha] = a/r$ , such  
that  $\text{Sin}[\alpha]^2 + \text{Cos}[\alpha]^2 = (b/r)^2 + (a/r)^2 = 1$

- Draw the circle through {1, 2}.
- CircleDefinedByPythagoras[1, 2];**



**A.9.4 PlotLine**

PlotLine is just steno for some standard applications of ListPlot.

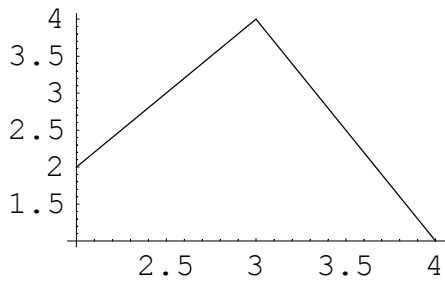
PlotLine[x\_List, opts]

is ListPlot with PlotJoined → True

PlotLine[x\_List, y\_List, opts]

idem for two variables, with application of Pick, DataFilter and Transpose

**PlotLine[{1, 2, 3, 4}, {10, 2, 4, 1}, Take -> {2, 4}]**

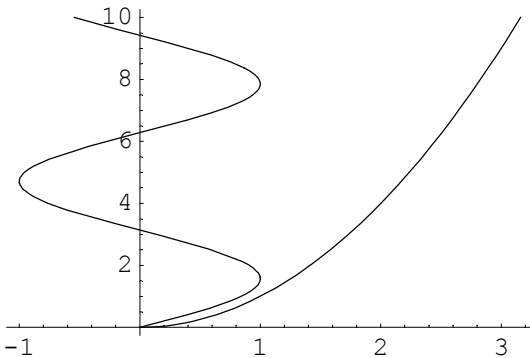


### A.9.5 Inverse plot

This following is useful when the horizontal axis is a function of the vertical axis. The plot toggles the axes.

```
InversePlot[                                plots  $x = x[y]$  for  $y$  on the vertical axis in  $\{b, e\}$ 
 $x, \{y, b, e\}, opts]$ 
```

```
InversePlot[{Sin[y], Sqrt[y]}, {y, 0, 10}];
```



### A.9.6 Phase diagram

If the list  $x$  contains subsequent values, then there are two basic kinds of phase diagrams. A first possibility is to look at  $\{x[[i]], x[[i + 1]]\}$ , a second possibility is to take intermediate rest points at  $\{x[[i]], x[[i]]\}$  before linking to the next step.

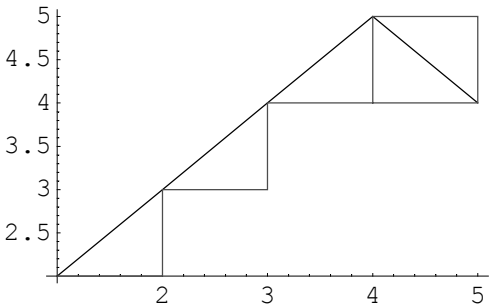
```
PhaseDiagram[ $x\_List, t:True, opts]$ 
```

plots in black ( $t = True$ ) or red ( $t = False$ ). For both,  $t = All$

Note: This uses the routine `XXPlusSpace[x, t]` that projects  $x$  in the  $\{x[[i]], x[[i+1]]\}$  space. Note that this result is stored in `Results[XXPlusSpace, t]`.

- This simple example should help to understand the two kinds of phase diagrams.

```
PhaseDiagram[{1, 2, 3, 4, 5, 4}, All]
```



The following example is taken from the chapter by Eckalbar in Varian (1993).

- Take the chaotic relationship where  $g[x[t+1]] = 4 x[t] (1 - x[t])$ , for  $0 \leq x \leq 1$ , and let us look at the dynamic path from the point  $x[0] = .6$ .

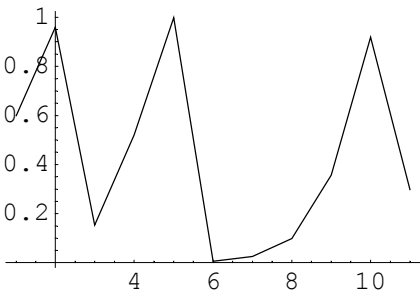
```
g[x_] := 4 x (1 - x)

path = NestList[g, .6, 10]

{0.6, 0.96, 0.1536, 0.520028, 0.998395, 0.00640774, 0.0254667, 0.0992726, 0.35767, 0.918969, 0.29786}
```

- A plot in time shows a chaotic, non-convergent, path.

```
path // PlotLine
```

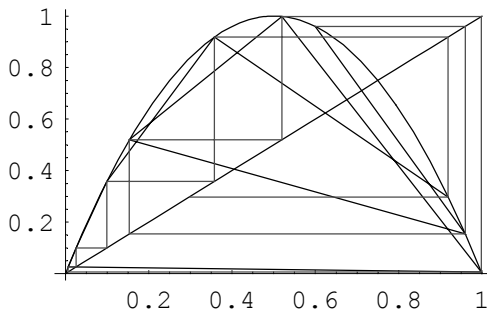


- For plotting of the PhaseDiagram we take as background the plots of  $g[x]$  and the diagonal  $y = x$ .

```
bg = Plot[{x, g[x]}, {x, 0, 1}, DisplayFunction -> Identity];

pd = PhaseDiagram[path, All, DisplayFunction -> Identity];
```

```
Show[bg, pd, DisplayFunction -> $DisplayFunction, TextStyle -> {FontSize -> 10}]
```



### A.9.7 Contours of constraints by polynomial approximation

The routine `SemialgebraicComponents, Algebra`AlgebraicInequalities`` gives you points in feasible areas for strict inequality constraints. The drawback is that this only recognises polynomials. A solution is to approximate functions by polynomials first.

```
ApproxPolynomial[f, {x_Symbol, mn, mx, n_Integer: 20}, m_Integer: 5, opts]
```

approximates  $f[x]$  by polynomial `Proxy[f][x]` of degree  $m$  for domain  $mn$  till  $mx$ ,  
 $n$  datapoints

```
ApproxPolynomial[f, {x_Symbol, mn, mx, n_Integer: 10},  
{y_Symbol, ymn, ymx, yn_Integer: 10}, m_Integer: 4, opts]
```

for 2 D functions  $f[x, y]$ . If `Rationalize -> True` then the coefficients are rationalised

`Proxy[f]` is the function created by `ApproxPolynomial`,  
 so that `Proxy[f][x]` approximates  $f[x]$

```
ApproxFeasibility2D[f_List, for f a list of 2 D fi,  
{x_Symbol, mn_, mx_, n_Integer: 10}, tests fi > 0  
{y_Symbol, ymn_, ymx_, yn_Integer: 10}, m_Integer: 4]
```

```
FeasibleQ[f_List, x_?VectorQ, g_Symbol: Greater]
```

tests whether all  $f_i[x] > 0$  (default  $>$ ). If  $x$  is a list of vectors, then it maps over that

- A set of possible constraints  $g[i][x, y] > 0$  is:

```
g[1][x_, y_] := 4 Log[Abs[x y]] - x^2  
g[2][x_, y_] := 2 x + 3 y - 8  
g[3][x_, y_] := E^x - 3 y
```

- This finds polynomial approximations and then identifies feasible points.

```
a2f = ApproxFeasibility2D[Table[g[i], {i, 3}], {x, -2, 6}, {y, -2, 6}]
```

$$\begin{pmatrix} -\frac{5}{2} & 9 \\ -\frac{9}{4} & \frac{13}{3} \\ 1 & 3 \\ 10 & 11 \\ \frac{37}{3} & -2 \\ \frac{37}{3} & 7 \\ \frac{25}{2} & 6 \end{pmatrix}$$

- This is not alarming yet, since we took an approximation.

```
FeasibleQ[Table[g[i], {i, 3}], a2f]
```

```
{False, False, False, False, False, False, False}
```

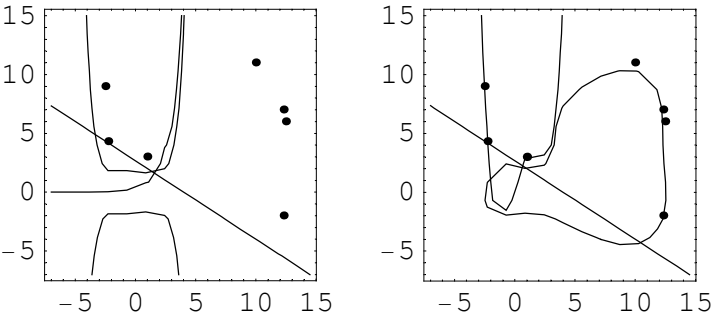
```
ConstraintsPlot[{f1[x, y], ...},  
({Proxy[f1][x, y], ...}, {x, xmin, xmax},  
{y ymin, ymax}, (data,) opts]
```

plots the contours of the  $f_1[x, y] = 0$ , including possible (feasible) data points

Note: The brackets mean that you can plot only one list of functions and need not plot the data points. But if you plot both lists of functions then you must include the data points.

- The true constraints are on the left, the approximation on the right. This shows that we have to make the grid finer and increase the degree of the polynomials - if such approximation is warranted.

```
ConstraintsPlot[Table[g[i][x, y], {i, 3}],  
Table[Proxy[g[i]][x, y], {i, 3}],  
{x, -7, 15}, {y, -7, 15}, a2f];
```



### A.9.8 Some utilities

These utilities are not connected.

`DisplayFunctionFilter[{opts___Rule}, ownopts___Rule]`

creates a pair {r, s}, where:  
 s is {opts} with all DisplayFunction option statements deleted  
 r is a DisplayOption  $\rightarrow$  value statement, either selected from opts,  
 or at a default setting controlled by Options[DisplayFunction] or ownopts

`ListDensityPlotSquared[x]`

partitions x into a square matrix and makes a density plot

`ParametricRange[{fx, fy}, t, xrange, yrange]`

gives the {t, tmin, tmax} range for the t  
 variable in the ParametricPlot[{fx, fy}, {t, tmin, tmax}],  
 with given {min, max} ranges for the x and y axes.

`Rainbow`

Rainbow shows Hue numbers in their Hue, valid from 0 till 1.  
 It actually is a transposed rainbow...

# A.10 Economics tools

The following routines are more common to economics.

## A.10.1 Elasticities and growth

Elasticity[ ] gives the instantaneous value.

Elasticity[f, x]	evaluates $x / f[x] * D[f[x], x]$
Elasticity[f, g, t]	evaluates $dLog[f] / dLog[g]$ with common variable t

Note: One approach for multivariates is the use of direct functions, e.g. f[x1, x2, #, x4]&.

BowElasticity[ ] gives the value over a certain range. When data are observed at discrete points, the curve taken is larger than infinitesimal.

BowElasticity[f, p, p2]	
	gives the bow or arc elasticity when {p, f[p]} moves to {p2, f[p2]}
BowElasticity[{p, y}, {p2, y2}]	
	when only data are given; the bow elasticity B and coefficient c solve $y == c p^B$ and $y2 == c p2^B$

- For a demand function:

**BowElasticity[Demand, p, p2]**

$$\frac{\log\left(\frac{\text{Demand}(p2)}{\text{Demand}(p)}\right)}{\log\left(\frac{p2}{p}\right)}$$

- If the budget is  $p q[p]$  and if there is a price increase  $dp$  that is fully absorbed by  $q$ :

**BowElasticity[{p, q[p]}, {p + dp,  $\frac{p q[p]}{p + dp}$ }]**

$$\frac{\log\left(\frac{p}{dp+p}\right)}{\log\left(\frac{dp+p}{p}\right)}$$

**SimplifyBy[% , LogRule]**



An elasticity can be seen as a ratio of growth rates. Here we can use `GrowthRate[ ]`.

<code>GrowthRate[f, x]</code>	evaluates $D[\text{Log}[f[x]], x] = D[f[x], x] / f[x]$
<code>GrowthRate[f_[y__], x]</code>	if x is only one of the y__
<code>GrowthRate[f, g, x]</code>	if f[x] and g[x] are given but the growth of f with respect to g is required

In same cases the use of logarithms will be useful here.

<code>LogRule</code>	a list of rules to expand logarithms
----------------------	--------------------------------------

To find out about these rules:

```
ShowPrivate["LogRule"]

Cool`Tool`Private`

A list of rules to expand logarithms

LogRule = {Log[(x_) / (y_)] -> Log[x] - Log[y],
  Log[(x_) * (y_)] -> Log[x] + Log[y], Log[(x_) ^ (y_)] -> y * Log[x]}
```

**A.10.2 Economic functions**

The exponential function is often used in simple demand analysis, as  $q[p] = A p^{\text{elas}}$ . The advantage of this function is the constant price elasticity of demand. This advantage however quickly turns into a disadvantage, when the constant price elasticity causes uneconomic results. A possible compromise is to subtract a constant, and adapt the other parameters such that functional value and elasticity at a particular point are maintained.

<code>LinExp[p, {A, elas, pstar, h}]</code>	translates $q[p] = A p^{\text{elas}}$ (i.e. with elasticity elas) with a linear additive constant $c = -h$ $A pstar^{\text{elas}}$ such that the functional value and elasticity at point pstar remain the same
<code>LinExp[p, {pstar, h}, qstar]</code>	gives the function so that maximisation of revenue $p q[p]$ is at $p = pstar$ , with demand $qstar = q[pstar]$

Another function is the logistic or sigmoid function.

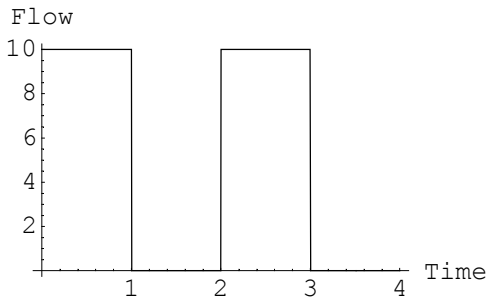
<code>Logistic[x]</code>	uses the parameter values given by <code>Options[Logistic]</code>
<code>Logistic[x, {right, center, range, slope}]</code>	uses the mentioned parameter values
<code>Options\$Logistic</code>	base options, see <code>ResetOptions[ ]</code>
<code>Slope</code>	option of <code>Logistic</code> , for the slope of the Logistic. Default is 1.

Note that 1 parameter is redundant. Users may however have different points of view about which.

Two cyclic functions are `OnOffFlow[ ]` and `SawTooth[ ]`.

<code>OnOffFlow[x, {t1, t2, ..., tn}][t]</code>	gives value $x$ if $t$ is in the interval $[ti, ti+1)$ for uneven $i$ , and 0 otherwise. Value $x$ may be a function of $t$
<code>SawTooth[t, q, d, pp:0]</code>	gives the saw-tooth function common for inventories (see text)

```
onf = OnOffFlow[10, {0, 1, 2, 3}]  
  
OnOffFlow(10, {0, 1, 2, 3})  
  
Plot[onf[t], {t, 0, 4}, AxesLabel -> {"Time", "Flow"}]
```



If the Sawtooth function is applied to inventories, then  $q$  is the order quantity and  $d$  is the demand rate, so that  $q/d$  is the period of supply. The cycle time variable then is  $t' = \text{Mod}[t, q/d]$ , and the level is  $q - d t'$ . If the production period  $pp$  of the commodity is not zero, then there first is a supply phase  $(q / pp - d) t'$ .

**A.10.3 Some contours**

The contours of the degree of utilisation can be made dependent upon the units of measurement.

```
UtilisationContours[ $\frac{f \ o}{c}$ , {c, cmin, cmax}, {o, omin, omax}, opts]
```

with  $u = f \ o / c$  = utilisation degree, for  $c$  = capacity,  
 $o$  = output, and  $f$  = adaptation of units

Note: When  $c$  is measured in units per day and  $o$  in 1000 units per year, then  $f = 1000 / 365$ .

With  $p$  a price and  $q$  a quantity,  $v = p * q$  gives the value or expenditure. Contours for an equal value or for equal expenditure show how much of the product could be gotten when the price changes, excluding the substitution and real income effects. In logarithms we have  $\text{Log}[v] = \text{Log}[p] + \text{Log}[q]$ , with the easy interpretation that the contours are straight lines. Plotting the  $\{p, q\}$  values for various products in contour plots like these give an impression of how the expenditure are positioned relative to each other.

This kind of plot is also useful for other kinds of problems. In frauds, there is the value of the fraud ( $v$ ) and the number of cases per category ( $n$ ), implying an average value ( $p$ ). In transport, there are the tonne-kilometers ( $tk$ ) as the indication of effort, composed of the tonnes ( $t$ ) and kilometers transported ( $k$ ).

```
EqualValueContour[p_List, q_List, number, type]
```

gives a graphics object with property `DisplayFunction`  $\rightarrow$  Identity,  
 for 1 or All equal value contours of either  $v =$   
 $p + q$  or  $v = p*q$ . Only known values are `number` = {1, All},  
 and `type` = {Plus, Times} (and 1 means average)

An example is given by `ShowPath`.

The `EqualValueContour[ ]` function actually sorts out the problem, and redirects the input to one of the following subroutines: `EqualProductContour[ ]` or `EqualSumContour[ ]`. In these routines, the options are passed on to `Plot[ ]` and `Show[ ]`.

By *equal product* we here mean giving an equal result after multiplication.

`EqualProductContour[x_List, opts]`

for  $x$  a list of pairs  $\{p_i, q_i\}$  (prices and quantities)  
gives the contours of equal products (values  $v_i = p_i * q_i$ )

`EqualProductContour[1, {p, q}, {min, max}, opts]`

gives a single contour graphics object for point  $\{p, q\}$  for the min  
to max range of  $p$  (with property `DisplayFunction`  $\rightarrow$  Identity).

`EqualProductContour[2, x_List, opts]`

gives a list of contour graphics objects.

By *equal sum* we here mean giving an equal result after summation. This will only be a logarithmic plot if the inputs are logarithms.

`EqualSumContour[x_List, opts]`

for  $x$  a list of pairs  $\{p_i, q_i\}$  gives the contours  
of equal sums (with Log of prices and quantities: values  
 $\text{Log}[v_i] = \text{Log}[p_i * q_i] = \text{Log}[p_i] + \text{Log}[q_i]$  ).

`EqualSumContour[1, {p, q}, {min, max}, opts]`

gives a single contour graphics object for point  $\{p, q\}$  for the  
min to max range of  $p$  (with property `DisplayFunction`  $\rightarrow$  Identity).

`EqualSumContour[2, x_List, opts]`

gives a list of contour graphics objects

`EqualSumContour[x_List, y_List, opts]`

gives a single contour based on the minima and maxima in  $x$  and  $y$ ,  
where  $x$  and  $y$  can have arbitrary equal length.

#### A.10.4 Linear programming 2D Plot

Linear programming is essentially multidimensional, but 2D cases still are instructive to plot.

`LP2DPlot[c_List, m_List, b_List, opts___Rule]`

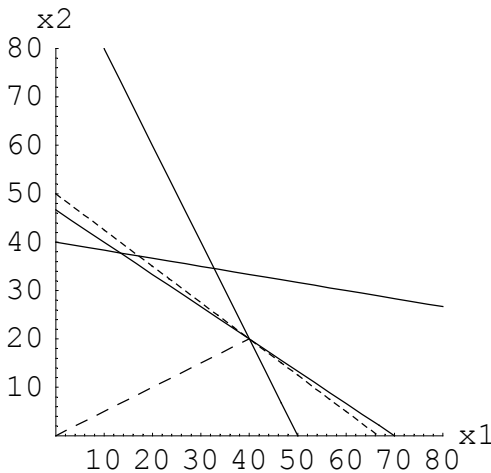
calls `LinearProgramming[c, m, b]`, stores the results in `Results[LP2DPlot]`, and makes a 2 D plot of the restrictions and solution. The solution is graphically identified by a dashed ray from the origin. The routine works only for 2 D problems (m has 2 columns). Options are past on to `Plot`.

Note: `LinearProgramming` essentially minimises. A maximising problem of  $\max c.x$  subject to  $m.x \leq b$  and  $x \geq 0$  can be entered as `LinearProgramming[-c, -m, -b]`, and the same can be done for `LP2DPlot`. Be aware, though, that the maximal value found still is  $c.x$  and not  $-c.x$ .

Note: The automatic plotting range is from 0 till some implied maximum value, which may be infinite when there are zero matrix elements. Use the standard option `PlotRange` to control the plot range.

■ An example is:

```
LP2DPlot[-{1.5, 2},
         -{{2, 3}, {2, 1}, {1/2, 3}},
         -{140, 100, 120},
         AspectRatio → 1, AxesLabel → {"x1", "x2"},
         PlotRange → {{0, 80}, {0, 80}}]
```



`Results[LP2DPlot]`

```
{{40., 20.}, -100.}
```

### A.10.5 Demand and supply cross

The following uses some of the other routines to plot a demand-supply cross. Some people would like to see prices on the vertical axis, but it is less confusing to students when we conform to the world standard of putting the cause on the horizontal axis.

```

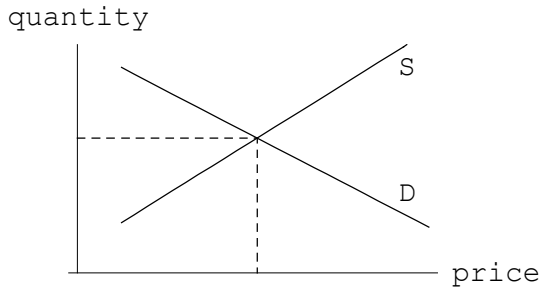
demand[price_] := -1. price + 100
supply[price_] := 1.2 price + 10

cross = Plot[{demand[price], supply[price]}, {price, 10, 80},
DisplayFunction -> Identity];
point = {x1, y1} = PointOfIntersection[demand, supply]

Show[cross,
  Asymptot[point, Automatic, DisplayFunction -> Identity],
  Graphics[{Text["D", {75,35}], Text["S", {75,90}]}],
  DisplayFunction -> $DisplayFunction,
  PlotRange -> {0,100},
  AxesOrigin -> {0,0},
  Ticks -> None,
  AxesLabel -> {"price", "quantity"},
  TextStyle -> {FontSize -> 11}];

{40.9091, 59.0909}

```



## A.11 Domain control, inequalities and piecewise

### A.11.1 Introduction

*Mathematica* already provides much help on domain control and inequalities, for example with the `Interval[]` object and `InequalitySolve[]`, `Algebra`InequalitySolve``. The Economics Pack enhances this with a few smaller routines on ranges and piecewise functions, such as `ToPiecewise[]` and `ConditionalApply[]`, and some larger routines on the `IFPair[]` object (i.e. 'inequality and function pair' object).

Though functions like `ToPiecewise` and `ConditionalApply` allow other conditions than just inequalities, the discussion below will concentrate on those.

### A.11.2 Domain control via piecewise

Piecewise functions can be a useful application of *Mathematica*'s `Which[]` object. Differentiation then affects only the values and not the domains. While `Which[]` objects can be defined easily, we still can provide for some utilities that can be called by other routines or that can be used for larger objects. For example, if the domains touch (as in  $x_0 \leq x < x_1$ ,  $x_1 \leq x < x_2$ , ...) as in tax functions, then we only want to type  $\{x_0, x_1, x_2, \dots\}$  once  $n$  reaches some critical size.

```
ToPiecewise[f_, {limit1, value1, limit2, value2, ...}]
```

creates a `Which` object for criteria `f[limiti]`. Note: normally the last criterion is `True`; if this is present, then `f` is not applied to it; if it is not present, `{True, Indeterminate}` is appended.

```
ToPiecewiseLinear[x_Symbol, {limit0, r0, c}, {limit1, r1, limit2, r2, ...}, opts]
```

creates a `Which` object for a piecewise linear function of `x`.  
To the utmost left, the function starts as  $r_0 x + c$ , for  $x < \text{limit}_0$ .  
In steps to the right, the slope  $r_i$  is valid for  $\text{limit}_{(i-1)}$  till  $\text{limit}_{(i)}$

Note that `ToPiecewise` is not restricted to inequality conditions; this is only an obvious application.

The default option of `ToPiecewiseLinear[]` is `Head → Less` and `Append → {True, Indeterminate}`. If `Head → f`, then this is used as `f[x, limiti]`.

- The following calls assume touching intervals.

```
fp[x_] = ToPiecewise[x < # &, {0, 0, 1, x^2, 2, x, True, x^2}]
```

```
Which[x < 0, 0, x < 1, x^2, x < 2, x, True, x^2]
```

```
D[fp[x], x]
```

```
Which[x < 0, 0, x < 1, 2 x, x < 2, 1, True, 2 x]
```

```
fp2[x_] = ToPiecewiseLinear[x, {a, b, c}, {lim1, r1}]
```

```
Which[x < a, c + b x, x < lim1, a b + c + r1 (x - a), True, Indeterminate]
```

- The following has non-touching intervals.

```
fp3[x_] =
```

```
ToPiecewise[#[[1]] ≤ x ≤ #[[2]] &, {{-1, 0}, 0, {1, 2}, x2, {3, 4}, x}]
```

```
Which[-1 ≤ x ≤ 0, 0, 1 ≤ x ≤ 2, x2, 3 ≤ x ≤ 4, x, True, Indeterminate]
```

### A.11.3 ConditionalApply

Let  $f_i(x)$  be defined over the interval  $I_i(x)$  for real  $x$ . Suppose that you want to find the maximum of all these over each interval, i.e.  $\text{Max}(f_1(x), \dots, f_n(x))$ , where you have to take account of interval overlaps and of the possibility that the functions intersect. A problem like this appears to occur in the Capital Asset Pricing Model, where an efficiency frontier with nonnegative weights has such a fractured format (with a Min condition).

You could implement the problem by defining each  $f_i$  as a `Which[ ]` object as above, and then calling `Max` on it. A small problem are the `Indeterminates` that arise for intervals for which a function is not defined. You get rid of these by `Indeterminate → ToSequence[ ]` before you apply the `Max`. Somehow, you lose on clarity and manageability though. The serious problem however is that a `Which` object takes the first occurrence of a `True`, and not *all* such occurrences. If intervals have been created by a routine over which you don't have control (such as `InequalitySolve`), and your `Which` object would look like `Which[0 ≤ x ≤ 3, f1[x], 1 ≤ x ≤ 2, f2[x]]` so that sorting does not help, then you are in a problem. The next function starts helping you out.

```
ConditionalApply[f, {c1, x1}, ..., {cn, xn}]    gives f[xi, ...] for the xi with True ci
ConditionalApply[                               the same. Note the reversed order
f, {x1, ..., xn}, {c1, ..., cn}]
```

Note that `ConditionalApply` is not restricted to inequality conditions; this is only an obvious application.

This does only evaluate when the `ci` indeed are `True` or `False`, and otherwise waits for this to happen.

- `ConditionalApply` works only for specific values of  $x$ .

```
ca[x_] = ConditionalApply[Max, {x ≤ 20, 1 - x}, {0 ≤ x ≤ 3, x2}]
```

```
ConditionalApply(Max, {x ≤ 20, 1 - x}, {0 ≤ x ≤ 3, x2})
```

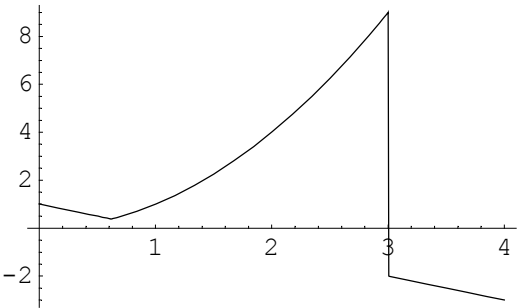


```
ca[1]
```

```
1
```

- Plotting helps identifying the intervals - but what about an analytical simplification ?

```
Plot[ca[x], {x, 0, 4}];
```



`ConditionalApply[ ]` is only a small step in the right direction. It does not provide an analytical equivalent to the latter plot. If you want to use an expression that states *where* the intervals overlap and *where* the functions intersect, then you are in the wilderness again. However, the following sections help to simplify a `ConditionalApply[ ]` statement into a `Which[ ]` object that explicitly states which function dominates for which interval. We need to cover some terrain before we get there, though.

### A.11.4 Domain, Interval and EPS

The `Statistics`Common`DistributionsCommon`` package provides a `Domain[ ]` call, and the convention is that `Domain[f]` gives the domain of function `f`. For this reason the Pack always loads this distributions package.

*Mathematica* provides a rather strong **Interval** function that comes with a union, intersection and membership test. The convention is that `Domain[f]` refers to such an interval object.

A supplement of the interval operations is:

<code>IntervalComplementAll[ x_Interval, y_Interval]</code>	gives the subinterval in <code>x</code> that is not in any of the <code>y</code>
<code>IntervalComplement2[ x_Interval, y_Interval]</code>	for binary <code>x = Interval[{a, b}]</code> and <code>y = Interval[{c, d}]</code>
<code>IntervalOverlapQ[ x_Interval, y_Interval]</code>	tests whether the two intervals overlap

Note: Inclusion of the letters "All" is to prevent conflicts if *Mathematica* later provides its own `IntervalComplement`.

```
IntervalOverlapQ[Interval[{1, 2}, {3, 10}], Interval[{4, 5}]]

True

IntervalComplementAll[Interval[{1, 2}, {3, 10}],
  Interval[{1, 2}, {4, 5}], Interval[{7, 9}]]

Interval[{3, 4}, {5, 7}, {9, 10}]
```

The big drawback of Interval is that it has only closed intervals and not open ones. Thus only  $[a, b]$  and not  $(a, b]$ , neither  $[a, b)$  nor  $(a, b)$ . The suggestion made here is to define `Domain[f]` with Interval and an undefined EPS. You still call `Domain[f]` for symbolic manipulations, but for numeric manipulations you call `NDomain`.

<code>NDomain[f]</code>	gives <code>Domain[f]</code> /. <code>Options[NDomain]</code> . Default <code>{EPS → 10.^(-6)}</code>
<code>EPS</code>	Symbol for a small number, which symbol can be replaced by the actual small number in <code>Options[NDomain]</code> . It is useful to keep EPS undefined in Domain and Interval statements, since it better conveys exclusion
<code>NoEPS</code>	<code>NoEPS[expr]</code> replaces EPS with <code>Options[NDomain]</code>

Note: Domain uses the context "Statistics`Common`DistributionsCommon".

- A domain with 10 excluded.

```
Domain[func] = Interval[{-Infinity, 10 - EPS}, {10 + EPS, 14}]

Interval[{-∞, 10 - EPS}, {EPS + 10, 14}]
```

- Interval does not handle open domains.

```
Domain[func] /. EPS → 0

Interval[{-∞, 14}]
```

- But the combination of NDomain and EPS does.

```
NDomain[func]

Interval[{-∞, 10.}, {10., 14}]
```

A straightforward application of EPS is to single out domain sections e.g. for rising and decreasing parts of functions.

```
IntervalSplit[Interval[x_List], y_?NumberQ, opts]
```

splits the interval in which y occurs. If option Method  $\rightarrow$  Min (default), then {b, e} becomes ({b, y-EPS}, {y, e}), otherwise ({b, y}, {y+EPS, e})

```
IntervalSplit[f_Symbol, {x_?NumberQ}]
```

splits Domain[f] for x's

- This leaves -5 and 5 still in the domain - that we then have to redefine.

```
Domain[func] = IntervalSplit[func, {-5, 5}]
```

```
Interval[{-∞, -EPS - 5}, {-5, 5 - EPS}, {5, 10 - EPS}, {EPS + 10, 14}]
```

Since EPS is a new symbolic object on the block, we need some more routines to let it fit in.

```
IntervalEPSMemberQ[x_Interval, y]
```

tests whether y is within EPS of interval x

```
IntervalEPSOverlapQ[  
x_Interval, y_Interval]
```

tests whether the two intervals overlap within EPS

If the intervals themselves already include EPS, the expression will only substitute EPS. Otherwise, an EPS is included around each interval.

- This gets stuck since IntervalIntersection meets with a symbolic expression.

```
IntervalOverlapQ[Interval[{1, 3 + EPS}], Interval[{3, 4}]]
```

```
IntervalIntersection[Interval[{1, EPS + 3}], Interval[{3, 4}]]  $\neq$  Interval[]
```

- These however handle EPS.

```
IntervalEPSOverlapQ[Interval[{1, 3 + EPS}], Interval[{3, 4}]]
```

```
True
```

```
IntervalEPSOverlapQ[Interval[{1, 3 - 10-7}], Interval[{3, 4}]]
```

```
True
```

```
IntervalEPSMemberQ[Interval[{0, 1}], 1 - EPS]
```

```
True
```

Given our reliance on Interval for domain statements, and the ease of inequalities like  $x > a$  for the human eye, we need a routine to go from one format to another. This is discussed in next section.

Note that Interval is not really defined for vectors. There are two remedies: (1) The use of FunctionDomainQ, which is just a symbol that allows you to define your own FunctionDomainQ[f]. (2)

LessEqual for lists, LE discussed in A4. Let us discuss some of these aspects, as an intermezzo, before we continue the discussion of simplifying ConditionalApply.

### A.11.5 Inequalities and Interval

*Mathematica* appears to have different representations of inequalities that however display the same. The display of  $a < b$  i.e. `Less[a, b]` and `Inequality[a, Less, b]` is the same, but the `FullForm` not. Also given the variety of  $\geq, >, ==, <$  and  $\leq$ , the complexity increases and the pattern test then takes 32 lines of code.

<code>ToInterval[s, expr]</code>	translates equalities and inequalities in <code>s</code> into interval statements, using <code>EPS</code> for strict inequalities. If there are more <code>expr</code> then a joint interval is created
<code>FromInterval[s, expr]</code>	translates Interval statements that occur in <code>expr</code> into equalities and inequalities in terms of symbol <code>s</code>

Note: `s == a` is also recognised. The term `ConditionToInterval` would suggest too general conditions.

- Note that this gives disjoint interval statements, and you would have to apply `IntervalUnion` to get a joint interval; but the latter only works for proper values.

```
ti = ToInterval[x, {a ≤ x, x == b, x > c}]

{Interval[{a, ∞}], Interval[{b, b}], Interval[{c + EPS, ∞}]}

IntervalUnion @@ %

IntervalUnion[Interval[{a, ∞}], Interval[{b, b}], Interval[{c + EPS, ∞}]]

% /. {a → 0, b → -1, c → 10} /. Options[NDomain]

Interval[{-1, -1}, {0, ∞}]
```

- A joint interval can also be created in this manner.

```
ti2 = ToInterval[x, a ≤ x, x == b, x > c]

Interval[{a, ∞}, {b, b}, {c + EPS, ∞}]
```

- `FromInterval` just substitutes into  $a_i \leq y \leq b_i$ . Also `Simplify` is not strong in this area.

```
FromInterval[y, ti2] // FullSimplify

{a ≤ y ≤ ∞, y == b, c + EPS ≤ y ≤ ∞}
```

### A.11.6 FunctionDomainQ and FunctionRangeQ

`FunctionDomainQ` is primarily a `Head`, that you have to define yourself for the uses that you make of it. Only two calls have been predefined, also to clarify how it could be used.

<code>FunctionDomainQ[...]</code>	general idea: renders True when the argument is in the domain of function, False if not
<code>FunctionDomainQ[Real, f, x]</code>	predefined, presumes Domain[f]
<code>FunctionDomainQ[List, f, {x}]</code>	predefined, presumes Domain[f], for vector functions
<code>FunctionDomainQ[function, args]</code>	user defined
<code>FunctionRangeQ[function, value]</code>	renders True when value is in the range of function, False if not.

Note: The latter two must be defined by the user.

- For above Domain[func], you for example could define:

```

FunctionDomainQ[func, x_] := IntervalMemberQ[NDomain[func], x]
FunctionDomainQ[func, #]& /@ {-100, 10}

{True, False}

```

- For a 2D vector function, the 'intervals' are rectangles with lower lhs and upper rhs corners.

```

ff[x_, y_] := x^2 + y^2

Domain[ff] = Interval[{{-10, -10}, {2, 3}}, {{5, 4}, {7, 6}}]

Interval[{{-10, -10}, {2, 3}}, {{5, 4}, {7, 6}}]

FunctionDomainQ[List, ff, {2, 4}]

False

```

### A.11.7 WhichIFPair: Selecting from various functions and intervals

Let us reconsider the problem stated above in A.11.3 of taking the Max of  $f_i(x)$  that are defined over the interval  $I_i(x)$  for real  $x$ . We have to take account of interval overlaps and of the possibility that the functions intersect (in that interval). The following function does the trick.

<code>WhichIFPair[f, x, {ineq<sub>i</sub>(x), f<sub>i</sub>(x)}, ...]</code>	makes a Which object of IFPairSimplify[f, x, {...}]
--	--

Here f is the overall function that determines which f<sub>i</sub> applies in what domain, and x is the symbol connecting domains and functions.

WhichIFPair is the main routine for reducing or simplifying 'inequality and function pairs', where each inequality will be reduced to the form  $a_i \leq x \leq b_i$ , and where each function  $f_i$  must be a function of x. The overall function f (typically Min or Max) determines which  $f_i$  applies in which interval.

- Note that intervals are adjusted for intersections of domains and functions.

```
wmax[x_] = WhichIFPair[Max, x, {-1 ≤ x ≤ 2, 1 - x}, {0 ≤ x ≤ 1, x2}]
```

```
Which[-1 ≤ x ≤  $\frac{1}{2}(-1 + \sqrt{5})$ , 1 - x,  $\frac{1}{2}(-1 + \sqrt{5}) ≤ x ≤ 1$ , x2, 1 ≤ x ≤ 2, 1 - x, True, Indeterminate]
```

The inequalities are internally transformed into "interval and function pairs" IFPair objects that use Interval. In the name IFPair the I is used for interval and inequality; this appears to be no problem. The remainder of this section concerns the properties of IFPair objects and their routines.

**A.11.7.1 The IFPair object**

The basic concept is the "Interval and Function Pair", shortened as IFPair. An IFPair[ ] object has as first element a simple interval object Interval[{b, e}], that is simple in the sense that there are no other intervals (lists). The second element is a function. In function calls you specify the variable *x* for which the IFPair is evaluated. You can also create such an IFPair object by first stating {*a* ≤ *x* ≤ *b*, *f*[*x*]}, and then calling ToIFPair. Only inequalities in precisely that format are recognised.

IFPair[ Interval[{x, y}], f]	is an 'interval & function pair' object that expresses that f[s] is defined for s in [x, y].
IFPair[x_List, f]	saves you typing Interval, for x a vector or matrix

IFPair is 'stubborn': when you add more lists to the interval, then it joins or splits. The idea is that we want the intervals as atomic as possible, so that we can concentrate on those intervals to select the f.

```
IFPair[{a, b}, {c, d}], f]

{IFPair(Interval[{a, b}], f), IFPair(Interval[{c, d}], f)}
```

FromIFPair[expr]	removes head IFPair and and returns to inequalities
ToIFPair[ Interval[x_List], f]	creates IFPair[Interval[xi], f] objects
ToIFPair[s, x_List]	turns a list of {a ≤ s ≤ b, f[s]} objects into proper IFPair objects

General inequalities are allowed. Importantly: the symbol *s* is retained internally to be able to recognise the f[s].

```
res = ToIFPair[x, {{x ≤ 1/2, 1 - x}, {0 ≤ x, x2}}]

{IFPair(Interval[{-∞,  $\frac{1}{2}$ }], 1 - x), IFPair(Interval[{0, ∞}], x2)}
```

SetIFPairSymbol[s]	sets the symbol of IFPair routines for which the intervals and functions are defined
--------------------	--

You use this if you use IFPair objects directly rather than WhichIFPair on inequalities  $a \leq s \leq b$ .

### A.11.7.2 IFPairSimplify

The main subroutine is IFPairSimplify, that can handle both inequalities and proper IFPair objects.

IFPairSimplify[ $f, x, \{ineq_1(x), f_1(x)\}, \dots]$	the main subroutine
IFPairSimplify[f, {p__IFPair}]	for IFPair objects

If you use the latter format then you must use SetIFPairSymbol[x] to set the symbol. If Options[IFPair] contain EPS[Remove] → True (default) then interval lengths smaller than EPS are removed from the solution.

- This gives the same as above, but without the Which wrapper.

**IFPairSimplify[Max, x, {{-1 ≤ x ≤ 2, 1 - x}, {0 ≤ x ≤ 1, x<sup>2</sup>}}]**

$$\left( \begin{array}{cc} -1 \leq x \leq \frac{1}{2}(-1 + \sqrt{5}) & 1 - x \\ \frac{1}{2}(-1 + \sqrt{5}) \leq x \leq 1 & x^2 \\ 1 \leq x \leq 2 & 1 - x \end{array} \right)$$

- This is directly on IFPairs, and properly speaking we should call SetIFPairSymbol[x].

**IFPairSimplify[Min, res]**

{IFPair(Interval[{0, ∞}], x<sup>2</sup>), IFPair(Interval[{-∞, 0}], 1 - x)}

Note that the intervals are sorted with RealSort - since symbolic sorting gives a different order. This is less crucial for the Which object since we have  $a \leq x \leq b$  conditions, but it reads better.

### A.11.7.3 Subroutines

The subroutines are a variation of themes. We have Intersection, Union, Complement, Split etcetera now for IFPair objects. The basic idea is that the functions are piggybacking on their domain while the domains are being fractured into nonoverlapping ones. When all domains have been fractured, then we take midpoint values, and apply the overall function to these function values, finding each function that maximises or minimises over the respective domain fractions.

Given that the main routines WhichIFPair and IFPairSimplify work, there is little chance that you would be interested in the details of these subroutines. You can find details by "?" and ShowPrivate. The following are only examples.

```
IFPairComplement[IFPair[{0, 10}, f], Interval[{1, 2}]]
{IFPair(Interval[{0, 1}], f), IFPair(Interval[{2, 10}], f)}
```

- These may have different functions f and g.

```
IFPairOverlapQ[IFPair[{a, b}, f], IFPair[{b, c}, g]]
IntervalIntersection[Interval[{a, b}], Interval[{b, c}]] ≠ Interval[]
```

- Joining works only if the objects have the same function f and touching intervals.

```
IFPairJoin[IFPair[{a, b}, f], IFPair[{b, c}, f]]
IFPair(Interval[{a, c}], f)
```

- Splitting can be done at a point, a list of points or around interval.

```
IFPairSplit[IFPair[{0, 10}, f], {-1, 2, 3, 20}]
{IFPair(Interval[{0, 2}], f), IFPair(Interval[{2, 3}], f), IFPair(Interval[{3, 10}], f)}

IFPairSplit[IFPair[{0, 10}, f], Interval[{4, 7}]]
{IFPair(Interval[{0, 4}], f), IFPair(Interval[{4, 7}], f), IFPair(Interval[{7, 10}], f)}
```

- For two IFPairs with the same domain x, this solves  $f = g$ , and bisects the interval. (Not evaluated.)

```
IFPairSolve[IFPair[{0, 1}, 1 - x], IFPair[{0, 1}, x^2]]
```

Note: Options[IFPair] has default {MemberQ → IntervalMemberQ}; IFPairSolve uses this to test whether the solution point is indeed in the interval and whether bisection is required. Use IntervalEPSMember if you want to increase the number of bisections around particular points.

- ToNonIntersectingIFPair[ ] has a recursive format. There are various input formats to allow for this recursion. The main call is for a list of objects. (Not evaluated.)

```
ToNonIntersectingIFPair[{IFPair[{-1, 2}, 1 - x],
                          IFPair[{0, 1}, x^2]}]
```

- Union is defined for a single f (and if not the same then nothing happens), and the routine can find the same functions. (Not evaluated.)

```
IFPairUnion[%, 1-x]

IFPairUnion[%]
```



- This gives the  $\{x, f[x]\}$  value for midpoint  $x$ . The output format is a bit complicated, but it helps pattern recognition. The value for Infinity is taken from Options[IFPair].

**IFPair[{0, 1}, x^2] /. IFPair → IFPairMidPoint**

IFPair(IFPairMidPoint( $\frac{1}{2}, \frac{1}{4}$ ), IFPair(Interval[{0, 1}],  $x^2$ ))

# A.12 Technical note

The discussion of the common routines above has concentrated on the material content. The following explains more about the technical structure.

`Needs["Economics`Pack`"]` makes available the following contexts, that are also shown in the `$ContextPath`:

<code>"Economics`Pack`"</code>	for overall control
<code>"Cool`Common`"</code>	for control of the Cool` packages and very basic routines
<code>"Cool`Context`"</code>	for dealing with Contexts
<code>"Cool`Declare`"</code>	for declarations of the Pack's packages and routines
<code>"Cool`Graphics`"</code>	for graphics routines
<code>"Cool`Inequality`"</code>	for inequalities and domain control
<code>"Cool`List`"</code>	for operations on lists
<code>"Cool`Manager`"</code>	for programming tools and control of options and results
<code>"Cool`Tool`"</code>	for other utilities

If you wish to access these basic common packages when writing your own packages, you would use `$Economics`. Your package then would have to start with a `Needs["Economics`Pack`"]` statement to make sure that the parameter is known.

<code>\$Economics</code>	The list of the basic common packages in the correct processing order. Use as: <code>Needs["Economics`Pack`"] (*define \$Economics*)</code> <code>BeginPackage[newpackage, Join[{...}, \$Economics]]</code>
--------------------------	---

Note: In the Traditional Form of Mathematica 3.0.0 the printing order is messed up when more than one package is called in one statement, as would happen with `Economics @@ $Economics`.

If you want to make a package, say `"MyPack`"`, and use some of the packages of the Economics Pack therein, then your package first must make these contexts available in some prologue statements.

- Calling `Economics[x's, Out → True]` will load the packages `x's` and produce the full context names in output. Your package then includes the list of the full context names in the `BeginPackage[ ]` statement.

```
(* prologue *)
Needs["Economics`Pack`"]
Dump["MyPack`"] = Economics["Economic`Common`", "CES`", Print → False,
Out → True];
(* begin *)
BeginPackage["MyPack`", Join[{(*...*)}, Dump["MyPack`"], $Economics]]
(* ... *)
EndPackage[]

MyPack`
```

Some exemplary notebooks may use `Cool[ ]`, `CoolPack` and `$Cool` statements. These are now internal to `Economics[ ]`, `EconomicsPack` and `$Economics`.

<code>Cool[xi, ...]</code>	shows the contents of <code>Cool`xi`</code> , and if needed loads the package(s). Input <code>xi</code> can be <code>Symbol</code> , or <code>String</code> with or without back-apostrophe. To prevent name conflicts, input <code>xi_Symbol</code> is first removed
<code>Cool[All]</code>	assignes the <code>Stub</code> -attribute to the packages in <code>CoolPack</code>
<code>CoolPack</code>	<code>CoolPack</code> gives the list of <code>Cool`</code> packages except <code>CoolExcept</code>
<code>CoolExcept</code>	list of <code>Cool`</code> packages not loaded by <code>Cool[All]</code> or <code>Economics[All]</code>

Note: In the Traditional Form of Mathematica 3.0.0 the printing order is messed up when more than one package is called in one statement.

In *Mathematica* 3.0.0 it appears that `CleanSlate`` and `FilledPlot`` don't mix well. `FilledPlot`` adds a graphics option, that `CleanSlate`` doesn't reset, so that later graphics cannot be made. The following subroutines fix this particular problem, and are part of `ResetAll`.

<code>ResetGraphics</code>	uses <code>Options[ResetStore]</code> to restore the <code>Options[Graphics]</code> to the value before the loading of <code>CleanSlate`</code>
<code>ResetStore</code>	<code>Options[ResetStore]</code> is a list of options that in case of some special packages like <code>FilledPlot.m</code> appears to be necessary to store options for <code>ResetAll</code> . This list now concerns graphics, but may be extended if other problems are found

# Appendix B: Literature

## B.1 General literature

---

- K. Arrow & F. Hahn (1971), *General competitive analysis*, North Holland
- O. Blanchard & S. Fischer (1989), *Lectures on macroeconomics*, MIT
- R. Barro & X. Sala-I-Martin (1995), *Economic growth*, McGraw-Hill
- D. Begg, S. Fischer & R. Dornbusch (1984, 1997), *Economics*, McGraw-Hill
- Th. Cool (2000), *Definition and Reality in the General Theory of Political Economy*, Samuel van Houten Genootschap, ISBN, 90-802263-2-7 (JEL 2000-1325)
- Th. Cool (2001), *Voting Theory for Democracy, Using Mathematica programs for Direct Single Seat Elections*, Thomas Cool Consultancy & Econometrics, ISBN 90-804774-3-5
- J. Cullis & P. Jones (1992), *Public finance & public choice*, McGraw-Hill
- R. Dornbusch & S. Fischer (1994), *Macroeconomics*, McGraw-Hill
- Th. Ferguson (1967), *Mathematical statistics*, Academic Press
- J. Freeman (1994), *Simulating neural networks with Mathematica*, Addison-Wesley
- A. Good & F. Graybill (1950, 1963), *Introduction to the theory of statistics*, McGraw-Hill
- D. Hendry (1995), *Dynamic econometrics*, Oxford
- F. Hillier & G. Lieberman (1967), *Introduction to operations research*, Holden Day
- C. Huang & P. Crooke (1997), *Mathematics and Mathematica for economists*, Blackwell
- J. Johnston (1963, 1972), *Econometric methods*, McGraw-Hill
- P. Korliras & R. Thorn (ed) (1979), *Modern macroeconomics*, Harper & Row
- L. Krajewski & L. Ritzman (1996), *Operations management*, Addison-Wesley
- P. Lambert (1985), *Advanced mathematics for economists*, Blackwell
- R. Lucas & Th. Sargent (ed) (1981), *Rational expectations and econometric practice*, Vol 1, Minnesota
- D. Luenberger (1998), *Investment science*, Oxford
- R. Maeder (1991), *Programming in Mathematica*, Addison-Wesley
- A. Mas-Colell, M. Whinston & J. Green (1995), *Microeconomic theory*, Oxford
- N. Mankiw (1992), *Macroeconomics*, Worth

- H. Rijken van Olst (1969), *Inleiding tot de statistiek*, Part I-II-III, Van Gorcum
- A. Sen (1970), *Collective choice and social welfare*, North Holland
- A. Takayama, (1974), *Mathematical economics*, The Dryden Press
- R. Teigen (ed) (1978), *Readings in money, national income and stabilisation policy*, Irwin
- H. Theil (1971), *Principles of econometrics*, John Wiley
- R. Shone (1998), *Economic dynamics*, Cambridge
- H. Varian (ed) (1993), *Economic and financial modeling with Mathematica*, Springer Telos
- H. Varian (ed) (1996), *Computational economics and finance*, Springer Telos
- S. Wolfram (1996), *The Mathematica book*, Wolfram Media & Cambridge

## B.2 Included papers that use the Pack

---

The Economics Pack comes with a set of papers, either in notebook or in HTML format, that rely on some work done with *Mathematica*. The papers can best be accessed via the ReadMe.html in the main Economics directory. Most papers are also available on the world wide web, in the Economics Working Papers Archive at the Washington University of St. Louis (<http://econwpa.wustl.edu>). Note, though, that a number of these papers have found their revised and definite form in Cool (2000).

### General Economics

*Unemployment Solved ! A breakthrough in economics*, ewp-get/9604002

*On the nature and significance of a free lunch*, ewp-get/9607003

*The solution to Arrow's difficulty in social welfare*, ewp-get/9707001

*Proper definitions for risk and uncertainty*, ewp-get/9902002

*A note on the CAPM: The market as a whole cannot have negative weights, but submarkets might*, ewp-get/9908001

*Understanding the impact of the minimum wage on the Gini index*, October 27 2000

### Political Economy

*How a dead wrong OECD tax policy causes mass unemployment*, ewp-mac/9508003

*On the political economy of employment in the welfare state*, ewp-mac/9509001

*Individual labour supply: A suggestion*, ewp-lab/9509001

*Dynamic curvature of the tax wedge*, ewp-pe/9604001

*Differential impact of the minimum wage on exposed and sheltered sectors*, ewp-get/9608001

*Summary of the solution to the current mass unemployment in the OECD area*, html 1997

*On the paradox of efficiency improvement at the micro level and Productivity Slowdown at the macro level: The case of Efficient Inventory Control*, ewp-get/9805003

## **Other subjects**

*Mathematica in the Fight against Fraud*, 1995

*Economics programs written in Mathematica*, ewp-prog/9508001

*A graduated transport fuel excise for metropolitan areas*, ewp-pe/9605001

*An estimator for the road freight handling factor*, ewp-urb/9703001

*The disappointment and embarrassment of MathML*, ewp-get/0004002

*Aspects of testing (matching, ranking and rating)*, ewp-get/0004004

*The choice on sustainability: information or the meta-SWF approach to a shift of preferences*, ewp-othr/0004004

*The seminal contribution of Roefie Hueting to economic science: Theory and measurement of Sustainable National Income*, November 16 2000

# Subject Index

## A

Accept, 298  
 Accounting: Balance  
     sheet etc., 206  
 AcidTest, 203  
 Act, 299  
 Actions, 185  
 Add, 457  
 AddedUsage, 506  
 AddLinearRates, 180  
 AdjustedRates, 452  
 AdjustRates, 178  
 AES, 123  
 AfterCashFlowsDates, 254  
 AfterSum, 75  
 AGEmodel, 145  
 AGEpath, 149  
 AGEexample, 144  
 Aggregate, 478  
 AggregatePrice, 117  
 Aggregation and  
     transpose, 478  
 Aggregator, 116  
 AggregatorQ, 478  
 AIDS, 141  
 Algebraic structure, 87  
 AllaisParadox, 350  
 Allocation, 146  
 AllocationTable, 147  
 AllQ, 471  
 Amortisation, 217  
 AnalysePrint, 84  
 An application of these,  
     512  
 AndEmbedding, 100  
 AndOrEnhance, 82  
 AndOrEnhanced, 82  
 AndOrRules, 82  
 AndSymmetric, 96  
 Angles and polar forms,  
     481  
 AnnualToDayRate, 203  
 Annuity, 216  
 AnyToCalendarDate, 36  
 AnyToStandardDate, 36  
 AppendToInContext, 508  
 Applied General  
     Equilibrium analysis, 144  
 ApplyB, 399  
 ApplyTest, 415  
 Appreciation, 28  
 AppreciationRate, 27  
 ApproxFeasibility2D, 517  
 ApproxPolynomial, 517  
 April, 39  
 Arcs, 512  
 ArgMaxQ, 479  
 ArgMinQ, 479  
 ArgumentQ, 509  
 Arrow diagrams, 45  
 ArrowwiseQ, 45  
 ArrowPratt, 61  
 Asset, 202  
 AssetAlpha, 211, 242  
 AssetBeta, 211, 225  
 AssetScatterPlot, 234  
 Assign, 146, 189  
 Asymptot, 511  
 Asymptots, 511  
 ATOP, 88  
 AUES, 123  
 August, 39  
 Autonomous, 489  
 Average, 480

AverageInventory, 126  
 AverageLevyRatio, 174  
 Axiomatic method and  
     EFSQ, 99  
 Axioms and metarules, 98  
 AxiomToTrue, 100

## B

B, 399  
 Back Apostrophe, 505  
 Backorder, 422  
 Backward lag operator,  
     399  
 BalanceLine, 206  
 BalanceSheet, 207  
 Bank, 486  
 BarChartServer, 300  
 BaseYear, 272  
 BasicStatistics, 301  
 BasicVariables, 446  
 Basis, 437  
 Bayes, 320  
 BeforeSum, 75  
 BeginContext, 506  
 BeginYear, 272  
 BenefitLine, 173  
 Bentham, 173  
 BerthOccupancyPlot, 452  
 beta, 242  
 Beta function, 306  
 Between, 458  
 BinaryRiskSimplify, 342  
 BinaryWeights, 226  
 BinomialSample, 362  
 BiNoPars, 358  
 BiNoProxy, 358  
 Birth, 409

- BlackArrow, 47
- BlackArrowRules, 46
- BlockDiagonal, 49
- Block diagonal matrices, 49
- BlockGP, 512
- Bond, 202, 247
- Bonds, 251
- BondToCashFlow, 253
- BondToCashFlows, 252
- BondToPlan, 43
- Borda, 108
- BordaAnalysis, 108
- BordaField, 108
- Border, 51
- BorderedDefiniteness, 51
- BorderedHess, 55
- BorderTotals, 477
- BorderTotalsQ, 477
- Boundaries, 439
- BowElasticity, 520
- Brief TOC, 3
- BudgetEquation, 117
- Bullit, 213
  
- C**
- Capacity, 125
- Capital, 125
- Capital Asset Pricing Model, 220
- CapitalMarketLine, 221
- CAPM, 222
- CAPMEquations, 221
- CAPMFit, 242
- CAPMParametricPlot, 232
- CAPMPlot, 220
- CAPMSolve, 238
- CAPMSolve : The market portfolio, 239
- CAPMSolve : The Markowitz efficient frontier, 235
- CAPMSymbols, 221
- CAPMWeightsPlot, 233
- Cartel, 195
- Carter's symbolic LP, 442
- Case, 358
- CasesMatch, 475
- Cash, 205
- CashCumulate, 250
- CashDifference, 250
- CashFlow, 203, 213, 253
- CashFlow statement, 209
- CashFlowStatement, 206
- CashFlowSumSimplify, 214
- CashStocks, 250
- Categories, 298
- CertainRate, 211, 223
- CertaintyEq, 345
- Certainty equivalence, 345
- Ces, 132
- CES, 131
- CESContours, 133
- CES – like functions, 136
- CESlimitQ, 132
- CESterm, 131
- CEStermRule, 132
- Chain, 297
- ChainIndex, 296
- ChainIndexTable, 297
- CheckES, 132
- CheckFileName, 277
- CheckVote, 106
- Chi2, 364
- Chi2CriticalValue, 366
- Chi2PValue, 366
- ChooseP, 91
- ChoosePointPr, 415
- ChoosePrint, 91
- CircleDefinedBy – Pythagoras, 514
- ClearFunctionByString, 509
- ClearInContext, 508
- CList, 407
- CNormalInverse, 304
- Coefficients, 489
- CoEq, 147
- ColumnPlayer, 187
- CombineRules, 464
- Comma Separated Variable format, 282
- ConcaveFind, 63
- Concepts and symbols, 447
- Conclude, 83, 85
- Conclusions, 85
- ConditionalApply, 528
- ConditionalPr, 308, 309
- ConditionalPrForm, 318
- ConditionalRisk, 331
- Condorcet, 106
- ConstantQ, 462
- ConstantRedemption, 216
- Constrained optimisation with equality constraints, 65
- Constrained optimisation with inequality constraints, 69
- Constraint, 489
- Constraint contours, 517
- ConstraintsPlot, 518
- ConsumerSurplus – FilledPlot, 127
- ConsumerSurplusPlot, 128
- Consumption, 125
- Contents, 504
- Context, 504
- Continuous densities, 343
- ContractCurve, 151
- ControlChart, 362
- Convexity and concavity, 58
- ConvexPlot, 60
- Cool, 539
- CoordiDate, 36



- CorrelationPr, 302
- Cost, 118
- CostElasticity, 123
- Covar, 302
- CovarMat, 302
- CovarMatPr, 302
- CovarPr, 302
- Coverage, 351
- CPC, 147
- CPCBudget, 150
- CPCDiagram, 147
- CrashBasedCapacity, 261
- CreateInsurance, 351
- CreateProspect, 311
- Credit, 206
- CriticalValue, 358
- CrossBreed, 415
- CrossEntries, 376
- CrossEntriesQ, 376
- CrossOutliers, 380
- Crosstables, chi2 test, 374
- CrossWeight, 379
- CSVToMatrix, 282
- CtE, 506
- Cumulate, 473
- Cure, 506
- CurrentAggregator, 119
- CurrentOptions, 501
- CurrentPoint, 444
- CurvedTax, 180
- CurvedTaxRates, 180
- D**
- D2ConcaveQ, 61
- D2ConvexQ, 61
- DashedCircle, 512
- DashingValues, 511
- Data, 406, 489
- Data and DataList
  - formats, 487
- Databank, 73, 74
- Databank procedures, 75
- Databases with Dbase`, 285
- DataContexts, 287
- Data files, 276
- DataFilter, 484
- Data format, 277
- Data formats, 268
- Data handling, 484
- DataList format, 278
- DataList package, 281
- DataListToM, 281
- DataListToMatrix, 279
- Data management, 392
- Data manipulation, 299
- DataMold, 73
- Data nomenclature, 269
- Data package, 280
- DataPackage, 506
- DataSymbols, 271
- DataToFile, 277
- DataToM, 280
- DataToMatrix, 278
- DatedDiscountFactors, 248
- DateFormat, 36
- Date formats, 36
- DateQ, 36
- DateToDays, 38
- DayFraction, 40
- Days, 40
- DaysToDate, 38
- DayToAnnualRate, 203
- DayToDayInterestFactor, 204
- DB, 291
- Dbase, 287
- Dbase[], 287
- DbaseContext, 287
- DbaseFile, 287
- DbaseTest, 287
- DbaseVariables, 289
- DCES, 134
- DConcaveQ, 60
- DConvexQ, 60
- Death, 413
- Debit, 206
- Debt, 125, 169
- DebtTrajectory, 217
- Debugging, 500
- Dec, 299
- December, 39
- Decide, 81, 83
- Decimals, 460
- Decision support
  - environment, 78
- Decision theory, 189
- DecisionTree, 192
- DeCure, 506
- Deduce, 83, 86
- DefDec, 356
- Deficit, 125, 169
- DefinedQ, 462
- Definiteness, 50
- Definiteness for
  - bordered matrices, 52
- Definition, 58
- DefinitionConcaveQ, 59
- DefinitionConvexQ, 59
- DefinitionToString, 469
- DegreesOfFreedom, 298
- DeleteElement, 474
- DeleteRule, 464
- DeleteVectorPosition, 474
- Delta, 463
- Demand, 122
- Demand and supply cross, 525
- DemandFunction, 123
- DemandPerCapita, 169
- Deny, 98
- DenyPattern, 98
- Depreciation, 28
- DeVal, 32
- Dictionary, 293

- DictionaryQ, 293
- Dif, 482
- DifBigToM, 295
- DIF format, 283
- DifToMath, 284
- DifToMatrix, 283
- DifToPackage, 295
- DimensionForm, 472
- DimensionMap, 472
- DimensionNest, 472
- DInventory, 126
- Discard, 509
- Discounting, 212
- DisplayFunctionFilter, 519
- Distance, 480
- Division, 457
- DivSmallestNonZero, 301
- Dollar, 25, 31
- Domain, 529
- Domain control, 527
- DQ, 461
- Draw, 314
- DrawNormal, 305
- DRule, 64
- DT, 189
- Du, 36
- Dual, 120
- DualQ, 120
- DualSelector, 122
- Due, 189
- DueMat, 357
- Dump, 501
- Dump directory, 276, 285
- DuncanBlackPlot, 113
- Duration, 411
- DynamicLeontiefInverse, 154
- DynamicLeontiefMatrix, 154
- DynamicMarginal, 179
- Dynamic modeling, 386
- Dynamics, 498
- E**
- EBIT, 203
- EBQ, 424
- Ecart, 203
- Econometrics, 384
- Economic Batch Quantity, 424
- Economic functions, 521
- Economic`Graphics`, 127
- Economic`Optimise`, 126
- Economic Order Quantity, 422
- Economics, 21, 102, 513
- EconomicsPack, 22
- Economics tools, 520
- EdgeworthBowley, 147
- EffectiveRate, 215
- EffectiveValue, 252
- Efficiency majority, 107
- EfficiencyMajority, 107
- EfficiencyPairs, 107
- EFSQ, 96
- Elasticities, 123
- Elasticity, 520
- ElasticityPlot, 127
- eluR, 463
- Employment, 166
- Endogenes, 388
- EndogenesRule, 388
- EndYear, 272
- Englogish, 83
- EOQ, 423
- EPS, 530
- EqualProductContour, 523
- EqualSumContour, 523
- EqualValueContour, 523
- EquationQ, 462
- Equations, 489
- Equations and models, 489
- EquationsToLPMatrices, 443
- Equilibrium, 146
- Equity, 203
- Equivalent, 81
- ERBank, 30
- Error, 298
- ES, 123
- Est, 299
- Estats, 394
- Estimate, 394
- Estimation, 411
- EuroBank, 25, 30
- EuroLand, 31
- EvaluationPerTest, 411
- EventsOrPlansToGOs, 44
- EventToGo, 44
- EventToGO, 44
- Event Universe, 319
- Evolution, 411
- Evolve, 408
- EVPerfectInformation, 189
- EVRRange, 231
- ExampleModel, 159
- Example plot, 60
- ExamplePrefs, 112
- Examples, 42
- ExchangeEquations, 27
- ExchangeModel, 27
- ExchangePrice, 26
- ExchangePrices, 25, 29
- Exemption, 173
- ExeQTime, 411
- ExistQ, 471
- Exogenes, 388
- ExpandAxiom, 100
- Expansion paths, 149
- Expectation, 489
- Expected, 489
- ExpectedReturn, 222
- Explain, 74
- Exponential smoothing, 459

Exports, 169  
 ExternalAccount, 169  
 ExtractWithHead, 466

## F

F, 36  
 FactorC, 117  
 FactorCoefficients, 117  
 FactorDemand, 122  
 FactorE, 117  
 FactorLabels, 122  
 FactorP, 121  
 FactorPosition, 122  
 FactorPowers, 117  
 FactorPrices, 117  
 Factors, 119  
 FactorSettings, 122  
 FactorX, 120  
 FactorXEq, 147  
 FADomain, 57  
 FDate, 38  
 FeasibleQ, 517  
 February, 39  
 FIDLLRStatistic, 378  
 File types, 277  
 FilterSetOptions, 501  
 FinalTableau, 445  
 Finance enhancement, 247  
 FinanceGO, 248  
 FinanceLimit, 213  
 Financial date object, 38  
 FindMinimumWage, 174  
 FiniteSourceSS, 453  
 Fitness, 413  
 FitnessTest, 412  
 FixedLinExp, 196  
 FleetMix, 260  
 FloatingRateNote, 213  
 Foc, 55  
 FoldListReverse, 473  
 FoldReverse, 473  
 Foreign, 27

ForeignRate, 27  
 Formal development, 318  
 FOToperiods, 248  
 FQ, 38  
 Freeman's routines, 405  
 Freight, 264  
 FreightEquations, 265  
 FreightModel, 265  
 FreightSymbols, 265  
 FrequencyCategories, 299  
 FrequencyDispersion, 300  
 FrequencyValues, 299  
 FrobeniusRoot, 153  
 FrobeniusVector, 153  
 FromBorderedMatrix, 51  
 FromConditional, 320  
 FromDollar, 31  
 FromDollarPrices, 31  
 FromEquationRule, 88  
 FromEvent, 319  
 FromF, 38  
 FromFile, 287  
 FromFileQ, 287  
 FromIFPair, 534  
 FromInterval, 532  
 FromLogNormal, 305  
 FromMold, 472  
 FromMoneyGraphics, 25  
 FromPolar, 481  
 FromProb, 318  
 Frontier, 236  
 FrontierAnalysis, 236  
 FrontierExample, 220  
 FrontierWeights, 238  
 FSort, 38  
 Ft, 38  
 FunctionAnalysis, 57  
 FunctionCalls, 492  
 FunctionDomainQ, 532  
 FunctionRangeQ, 533  
 Functions, 409  
 F\$ContextPath, 247

## G

GamePlot, 186  
 Game theory and decision  
   theory, 183  
 GameToPayOffRules, 188  
 GameValue, 186  
 GanttChart, 419  
 GDP, 169  
 GeneralizedStringTake,  
   469  
 Generation, 412  
 GenerationQ, 412  
 Genesis, 409  
 Genetic programming, 408  
 GeometricCovar, 302  
 GeometricSpread, 302  
 GG, 407  
 GNP, 169  
 Gra, gradient, 55  
 Graphics, 133, 248, 511  
 GrossToNetRatio, 174  
 GroupOrderQ, 110  
 GrowthFund, 216  
 GrowthRate, 521

## H

Haul, 265  
 Heads, 465  
 Hess, Hessian, 55  
 HintonDiagram, 405  
 HoldShell, 502  
 Home, 27  
 HomeRate, 27  
 HueArrow, 47  
 HueArrowRules, 46

## I

IADS, 141  
 IADSpert, 142  
 IADSpertRule, 142  
 IADSpriceRule, 142



ID, 298  
 IDInventory, 428  
 IDPrMatrixQ, 308  
 IDProspectsPrMatrix, 316  
 IfAntiSymmetric, 96  
 IfNegativeToMaxRule, 329  
 IfNegativeToMinRule, 329  
 IFPair, 527  
 IFPairComplement, 536  
 IFPairJoin, 536  
 IFPairMidPoint, 537  
 IFPairOverlapQ, 536  
 IFPairSimplify, 533  
 IFPairSolve, 536  
 IFPairSplit, 536  
 IFPairUnion, 536  
 IfToAnd, 96  
 IfToIf, 98  
 IfTransitive, 96  
 Imp, 81  
 Implications, 81  
 ImplicitD, 55  
 ImpliedShares, 200  
 Imports, 169  
 Income, 203  
 Income statement, 208  
 IncomeStatement, 208  
 Independent prospects, 316  
 Indexation of rates, 178  
 Indexed, 178  
 IndexedSort, 475  
 Indexed sorting and  
   RealSort, 475  
 IndifferenceCurve, 150  
 Indirect Addilog Demand  
   System(IADS), 141  
 Inequalities and  
   equations, 490  
 Inequalities and  
   Interval, 532  
 Infer, 95, 96

InferAndOr, 96  
 InferenceMachine, 98  
 InferQ, 96  
 Inflation, 159  
 InRange, 458  
 InRangeQ, 458  
 InsideTable, 485  
 Instalment, 216  
 InstalmentTable, 217  
 Insurance, 351  
 IntegerSCM, 460  
 IntegerToDate, 37  
 Intercept, 462  
 Interest rates, 203  
 Intermediate, 125  
 IntermediatesQ, 145  
 InterpolatedDF, 249  
 IntervalComplement2, 529  
 IntervalComplementAll, 529  
 IntervalEPSMemberQ, 531  
 IntervalEPSOverlapQ, 531  
 IntervalOverlapQ, 529  
 IntervalSplit, 531  
 IntervalSumFR, 249  
 Inventory, 126  
 Inventory basics, 422  
 Inventory control : P and  
   Q – systems, 426  
 InventoryModel, 125  
 InventoryPolicyCost, 422  
 InverseChi2, 366  
 InverseF, 54  
 Inverse function, 54  
 InverseHessian, 398  
 InversePlot, 515  
 IOAnalysis, 152  
 IOMatrixQ, 153  
 IS, 161  
 ISLMModel, 156  
 ISLMPlotPrepare, 163  
 Isocost, 151

Isoquant, 151  
 Isoquants, 151  
 Items, 104

## J

Jacobi, Jacobian, 55  
 January, 39  
 JobFlowTime, 422  
 JohnsonsRule, 420  
 JoinIDProspects, 316  
 JoinNews, 473  
 JoinStatements, 84  
 JointProspect, 317  
 JointProspectEV, 317  
 JointProspectPrValue, 317  
 JointProspects, 317  
 JointProspects and risk, 341  
 JointToMarginalProspects, 317  
 JointToProspects, 318  
 July, 39  
 June, 39

## K

KeepVariables, 509  
 Key, 506  
 Kilroy, 500  
 KTEquations, 71  
 KTStartValues, 71  
 KuhnTucker, 69

## L

Labels, 271  
 Labour, 121  
 LabourMarket, 165  
 LabourProductivity, 125  
 Lag, 482  
 Lagrange, 65, 69  
 Lagrangian, 56  
 Laplace, 313  
 LE, 483



- Lead, 482
- Lead, Lag, Dif,
  - UnitIndex, RateOfChange, 481
- LeapQ, 40
- Leontief model,
  - LeontiefInverse, 153
- LessComplicatedQ, 462
- LessEqual for list, 483
- LevelCES, 138
- LevelCESAnalyser, 140
- LevelCESlevel, 140
- LevelCESRule, 140
- LevelInspect, 463
- Levels, 122
- Levy, 170
- LevyPlot, 171
- LevyPlot, 172
- Liability, 203
- LIBOR, 203
- Lift, 265
- LimitingResource, 446
- LinearEquationQ, 443
- Linear programming, 434
- Linear programming 2 D
  - Plot, 524
- LineRasterGP, 512
- LinExp, 521
- Link capacity, 261
- LIQRule, 168
- LiquidityPreference, 162
- ListDensityPlotSquared, 519
- ListOfSymbols, 489
- Lists, 471
- Lists of strings, 469
- ListToDif, 283
- ListToGraph, 114
- ListToOrderedPairs, 107
- ListToPref, 110
- ListToPrefOrderQ, 111
- ListToVoteMargin, 111
- LL, 407
- LLRStatistics, 377
- LM, 162
- Load, 504
- LoadFactor, 265
- LocateThen, 84
- LogicalPattern, 101
- LogicalVariables, 81
- Logic and Inference, 78
- LogicLab, 93
- Logistic, 522
- Logistic function, 405
- LogLikelihoodRatio, 370
- Lognormal function, 305
- LogRule, 521
- LongestLag, 390
- LookAt, 184
- Loss, 125, 188
- LowHighTariffPlot, 198
- LP2DPlot, 525
- LPMatricesToEquations, 443
- M**
- MachineForm, 101
- Macro economics, 156
- MacroSymbols, 158
- Make, 376
- MakeAndSave, 382
- MakeDate, 37
- MakeSpan, 422
- MapLevel, 463
- March, 39
- MarginalCost, 123
- MarginalPr, 308, 317
- MarginalProduct, 123
- MarginalRate, 174
- Marginal values, 123
- Market, 203
- MarketPortfolio, 238
- MarkUp, 125
- Matching and pairs, 475
- Math file format, 279
- MathToUnionDataList, 295
- Matrices, 49, 476
- Matrix package, 281
- MatrixRowSums, 477
- MatrixTimesDiagonal, 477
- MatrixToCSV, 282
- MatrixToDalt, 279
- MatrixToData, 278
- MatrixToM, 281
- MatrixToMath, 279
- Max0ToIfNegative, 231
- MaximalProfit, 119
- MaximumLevel, 409
- MaxInventory, 422
- May, 39
- MeanArrivalRate, 451
- MeanNSystem, 451
- MeanProcessTime, 448
- MeanQueueWait, 454
- MeanServiceRate, 454
- MemoryConstrainedQ, 503
- MessagesQ, 510
- MessagesQ, 510
- MetaRuleForm, 101
- Min2ndOf2, 475
- MinimalCost, 119
- MinimalDue, 188
- Minimax, 185
- MinimiseCost, 126
- MinimumIncome, 174
- MinimumWage, 174
- Missing data, 274
- MissingData, 274
- MissingToZero, 299
- MLEstimate, 370
- Model, 388
- Models with one lag, 387
- ModelVariates, 388
- ModelY, 398
- ModusPonens, 96
- Mold, 472

- MoneyConvert, 25
- Money exchange rates and conversion, 25
- MoneyForm, 32, 34
- MoneyPaddedForm, 35
- MoneyPaddedRound, 35
- MoneyRound, 34
- MonLabels, 39
- MonopolyEquations, 194
- MonopolyModel, 194
- MonRule, 39
- MonSort, 39
- MoveToFirst, 474
- MPFromParametric, 231
- MPWeights, 238
- MultipleServers, 453
- MultipleServersContours, 455
- MultipleServersModel, 454
- MultiplierEquations, 67
- Multipliers, 65
- MultiplierStartValues, 67
- MultiplierSymbol, 66
- Mutate, 415
  
- N**
- N3PointsConcaveQ, 62
- N3PointsConvexQ, 62
- NAIRU, 166
- NamesList, 270
- Nash`Nash` package, 188
- NAWIRU, 166
- NBeingServed, 448
- NConvexFind, 62
- NDomain, 530
- NeedsQ, 504
- Negate, 81
- NestHead, 466
- NetExports, 169
- NetIncome, 170
- NetIncomePlot, 172
- NetWorth, 203
  
- Neural networks, 405
- New objects, 248
- NewsBoyProblem, 432
- NIntegrateToFirstZero, 459
- NLocalConcaveQ, 59
- NLocalConvexQ, 59
- Node, 46
- NodeLabels, 46
- Nodes, 46
- NoEPS, 530
- NoF, 38
- Nominal, 125
- NonAttributedSymbols, 491
- NonAttributedTerms, 493
- NonBasicVariables, 446
- NonNegativeMatrixQ, 153
- NonNegativeSolution, 498
- NoNulls, 458
- NonZero, 458
- NonZeroQ, 458
- NOrderPair, 107
- NormalisePr, 324
- Normalize, 413
- NormalProxy, 304
- NormedDuration, 41
- NotNot, 81
- NotpOrq, 81
- November, 39
- NPairs, 109
- NPeriodsOfSupply, 126
- NPrefOrderQ, 112
- NQueue, 448
- NRadius, 480
- NServers, 449
- NShiftCES, 136
- NSolveLinearDynamics, 498
- NSolvePeriod, 388
- NSolveYear, 388
- NSystem, 448
- Nullify, 458
- NullQ, 458
  
- NumberOfActions, 298
- NumberOfConstraints, 66
- NumberOfDecisions, 299
- NumberOfElements, 322
- NumberOfFactors, 117
- NumberOfFailures, 322
- NumberOfGenerations, 412
- NumberOfIndividuals, 409
- NumberOfItems, 106
- NumberOfObservations, 298
- NumberOfOrders, 422
- NumberOfSectors, 125
- NumberOfStates, 298
- NumberOfTests, 411
- NumberOfVariables, 66
- NumberOfVoters, 106
- Numerical LP analysis, 434
- NumLP, 435
- NumLPAnalysis, 435
- NumLPPlot, 439
  
- O**
- ObjectForm, 101
- Obs, 299
- OCCurve, 359
- OCCurveInverse, 359
- October, 39
- OddOrEven, 184
- OldInventory, 126
- OneLag, 387
- OnlyNulls, 458
- OnOffFlow, 522
- Operating characteristic curve, 358
- Operations research, 418
- Optimal, 125
- Optimal taxation, 179
- OptionsFilter, 501
- Options\$Aggregator, 119
- Options\$CES, 132
- Options\$IADS, 141



Options\$LevelCES, 138  
 OrderCost, 422  
 OrderLeadTime, 422  
 Order of contexts, 247  
 OrderQuantity, 422  
 OrSymmetric, 96  
 Outliers, 303, 380  
 Owning or hiring, 259

## P

P, 85  
 Packages, 280  
 PaddedString, 487  
 Pairs, 109  
 PairsToPaths, 108  
 Pairwise, 109  
 PairwiseField, 109  
 Pairwise majority, 109  
 PairwiseMajority, 109  
 PairwiseVoter, 110  
 Palettes, 22  
 Paragraph, 90  
 Parallelogram, 512  
 Parameter, 489  
 ParametricRange, 519  
 Participation, 169  
 PartInspect, 463  
 PartitionKeep, 488  
 PartList, 474  
 Partner, 196  
 Paths in arrow diagrams, 47  
 Pattern recognition, 407  
 Patterns, 470  
 Pay, 206, 219  
 Payment Tables, 217  
 PayOff, 183  
 Pay – off and Loss, 183  
 PayOffRuleToGame, 188  
 PayOffTable, 184  
 PayOffToJointProspect, 192  
 PayOffToProspects, 192  
 PayOffToRule, 184  
 PeakLoad, 199  
 PeakLoadCapital, 199  
 PeakLoadEquation, 200  
 PeakLoadLegend, 199  
 PeakLoadPlot, 201  
 Pearson, 369  
 pEq, 147  
 Performance, 245, 246  
 Periodical payment, 215  
 Perpetuity, 216  
 PFunction, 145  
 PhaseDiagram, 515  
 PhillipsCurve, 166  
 Pick, 484  
 PieceWiseCurved, 180  
 Piecewise  
   discontinuities, 429  
 PieceWiseLinear, 170  
 PiecewiseWhich, 171  
 Pivot, 446  
 Place, 101  
 PlacedMetaRule, 101  
 PlacedProperties, 101  
 PlaceOperator, 101  
 PlaceUndo, 101  
 Plain symbols, 489  
 PlanToGO, 44  
 PlotLine, 514  
 PlusYears, 40  
 PM, 307  
 PointOfIntersection, 460  
 Polygamy, 475  
 PolynomialOf, 400  
 Population, 414  
 Portfolio, 205, 255  
 PortfolioAnalysis, 236  
 Portfolio object, 204, 254  
 Portfolios, 234  
 PortfolioSigmaMu, 235  
 PortfolioValue, 205, 252  
 PositiveMatrixQ, 153  
 PowerOfTheTest, 356  
 PowersOfB, 400  
 PPath, 108  
 PPC, 150  
 Pr, 298, 309  
 Pralse, 510  
 PrBelowOutMean, 303  
 PrDemandLessThanLevel, 431  
 PrDomain, 324  
 Pref, 110  
 Preferences, 111  
 PreferencesPlot, 113  
 PrefOrderQ, 111  
 PrefPairs, 111  
 PrefPairsPlot, 112  
 PrefPairsToGraph, 114  
 PrefPairsToOrderQ, 112  
 Premium, 125, 170, 203  
 PremiumRates, 170  
 PremiumRatio, 225  
 PremiumToSpreadRatio, 223  
 Presentation with  
   TableForm, 484  
 Presenting and selecting  
   solutions, 496  
 Presenting data in a bar  
   chart, 300  
 Present value, 214  
 PrEst, 373  
 Price, 125  
 PriceElasticity, 123  
 PriceLevel, 163  
 Price of an asset, 228  
 Printing, 487  
 PrintPrepare, 487  
 PrMarginal, 308  
 Prob, 318  
 Probability, 307, 309  
 ProbabilityMatrix, 308

ProbabilityMeasureQ, 307  
 Problem, 490  
 ProbMat, 356  
 Production, 125  
 Profit, 118  
 ProfitPerVKM, 258  
 Programming, 500  
 Projection, prospects,  
     333  
 ProjectLP, 438  
 ProjectNumLP, 438  
 PropensityToConsume, 169  
 ProperSentence, 84  
 Properties, 124  
 Propositional logic, 81  
 PropositionMeaningRule,  
     84  
 Propositions, 85  
 PropVariableList, 86  
 Prospect, 309  
 Prospect3DPrTriangle, 337  
 ProspectApply, 312  
 ProspectEV, 309  
 ProspectInnerSort, 312  
 ProspectInverse, 333  
 ProspectPlot, 338  
 Prospect plotting, 337  
 ProspectProjection, 333  
 ProspectPrValue, 335  
 ProspectQ, 309  
 ProspectReList, 312  
 Prospects and risk, 329  
 ProspectSort, 335  
 ProspectSplit, 335  
 ProspectSplitPlot, 340  
 ProspectStatistics, 336  
 ProspectUtility, 311  
 Proxy, 517  
 PrTable, 308  
 PutIn, 313  
 PutIntoContexts, 509  
 PV, 214

PVFromRedemption, 218  
 Pythagoras, 513

## Q

Quantifiers, 471  
 Quantity, 125  
 Quasi concavity, 63  
 Query, 505  
 Questionnaire, 375  
 QueueBasics, 448  
 Queueing, 447  
 QueueSymbols, 447  
 QuickTest, 377

## R

Rainbow, 519  
 Random drawing from  
     Prospects, 313  
 RandomLabel, 490  
 RangeOfFeasibility, 436  
 RangeOfOptimality, 436  
 Ranges, 458  
 Rate, 125  
 RateOfChange, 482  
 RateOfInterest, 125  
 Rate of return, 203  
 RateOfReturn, 203  
 Ratio, 489  
 Ratio's, 210  
 ReadData, 294  
 ReadDbase, 292  
 ReadPackage, 292  
 RealN, 458  
 RealNumberQ, 462  
 RealQ, 462  
 RealRootQ, 497  
 RealRoots, 497  
 RealSort, 476  
 RealToDate, 36  
 Redefine, 468  
 Redemption, 216  
 ReducedFormQ, 398

ReEvaluate, 502  
 Regression, 241  
 Regret, 184  
 Reject, 298  
 RelativeFreq, 299  
 RelativeRisk, 331  
 RelativeRiskEquations,  
     331  
 RelativeStabilityQ, 154  
 RemarkOnParentheses, 81  
 RemoveGlobalB, 400  
 RemoveVariables, 509  
 ReOrderPoint, 427  
 Repeat, 473  
 ReplaceByHeadCount, 465  
 ReplaceByZero, 500  
 ReplaceInRule, 464  
 Replacement by head  
     count, 465  
 ReplacementRate, 173  
 Reset, 189  
 ResetAll, 21  
 ResetData, 273  
 ResetFactors, 119, 120  
 ResetFactorsExpand, 121  
 ResetGraphics, 539  
 ResetNodes, 46  
 ResetOptions, 500, 522  
 ResetResults, 502  
 ResetStore, 539  
 Resources, 125  
 Results, 501  
 ReVal, 32  
 Revenue, 125  
 Risk, 327, 329  
 RiskAversion, 345  
 Risk concepts, 331  
 RiskEquations, 331  
 Risket, 332  
 RiskFactors, 336  
 RiskModel, 331  
 Risk models, 331

- RiskPr, 329
- RiskPremium, 345
- RiskRatio, 331
- RiskRatioSplit, 332
- RiskRatioSplitPlot, 341
- Risks, 336
- RiskyQ, 329
- RootToN, 497
- RoundAt, 460
- RouteDistance, 262
- Routine, 65
- Routines, 65
- RowPlayer, 187
- RTS, 120
- Rule basics, 463
- RuleQ, 462
- RuleToR2, 411
  
- S**
- S, 131
- Sacrifice ratio plot, 455
- SafetyStock, 422
- SampleSize, 358
- Sampling, 358
- SawTooth, 522
- Scale, 120
- Score, 413
- Scoring, 509
- SDR, 31
- Sector, 125
- Sectors, 270
- SecurityMarket, 244
- Security Market Line, 243
- Seed, 409
- SelectedVariables, 287
- SelectSolutionRules, 163
- SelfImplication, 96
- Sensitivity analysis, 446
- SensitivityAnalysis, 446
- Sentences, 84
- September, 39
- SequenceToRules, 463, 500
- SequentialDraws, 322
- SetCartel, 196
- SetData, 273
- SetDatabank, 74
- SetExchangePrices, 29
- SetFunction, 124
- SetIFPairSymbol, 535
- SetModel, 145
- SetMonopoly, 195
- SetOptionsCESSector, 135
- SetRandomPreferences, 112
- SetRandomPrefPairs, 112
- SetResults, 502
- SetROC, 273
- Sets, 473
- SetStage, 263
- SetTaxAndPremium, 171
- Setting up a problem, 442
- Setting values, 500
- ShadowHull, 509
- ShadowPrice, 446
- ShadowPrices, 437
- Shares, 196
- SharpeIndex, 246
- ShiftCES, 136
- ShiftCESContours, 137
- ShiftCESRhs, 138
- ShowAll, 46
- ShowBlackArrows, 47
- ShowCT, 377
- ShowData, 76
- ShowEvent, 44
- ShowEvents, 44
- ShowEventsOrPlans, 43
- ShowFO, 248
- ShowHueArrows, 47
- ShowListGraph, 114
- ShowPath, 48
- ShowPlan, 44
- ShowPlans, 44
- ShowPrefGraph, 114
- ShowPrivate, 505
- ShowRoute, 263
- ShowText, 47
- Simplex, 442
- SimplexTableau, 437
- SimplifyBy, 466
- SimplifyBySolve, 466
- SimplifySqrt, 466
- SimplifyTestByLinear, 467
- SingleInterpolatedDF, 250
- SingleServer, 449
- SingleServerModel, 451
- Slack, 437
- Slope, 522
- SmatchQ, 462
- Smooth, 459
- Social welfare and  
voting, 104
- Solution, 391, 490
- Solve2List, 494
- SolveFormally, 494
- SolveFrom, 494
- SolveGeneral, 494
- SolveLHS, 494
- SolveMinimumWage, 174
- SolveRecurse, 494
- SolveShow, 497
- Solving it, 444
- SortSameQ, 475
- SoZero, 55
- Spread, 480
- SpreadAversion, 345
- SpreadOnlyRate, 211
- SpreadPremium, 345
- Stage, 263
- StartValues, 489
- StartYear, 273
- State, 299
- States, 185
- Statistical decision  
theory, 353
- Statistical measures, 301
- Statistics, 266, 336

Statistics`Common`, 298  
 StatusQuo, 106  
 Stock, 203  
 StorageCost, 422  
 Store, 271  
 StringListPralse, 469  
 StringListQ, 469  
 StringToDate, 37  
 StringToList, 469  
 SubCESNoFactors, 140  
 Subject Index, 543  
 SubsequentBetaCDF, 307  
 Subsistence, 173  
 Substitute  $h[x]$  back in  
      $h'[x]$ , 63  
 SuccBorderedMinors, 51  
 SuccPrincipalMinors, 50  
 SumLog, 372  
 SumOverConcepts, 272  
 SumOverSectors, 272  
 Supply, 125  
 Surprise, 489  
 Symbolic manipulation,  
     461  
 Symbols, 491  
 SymmetricMatrixQ, 50  
 System Enhancement, 24  
 Systems of equations, 396

## T

Tabulate, 488  
 Tajuddin, 303  
 TakeElements, 474  
 Taken, 65  
 TakeOut, 313  
 TakeRule, 65  
 TargetInventory, 426  
 Tax, 125, 170  
 TaxCredit, 173  
 TaxDeduction, 181  
 TaxOnWage, 170  
 TaxRates, 170  
 Tax revenue, 175  
 Tax Void Diagram, 176  
 TensorForm, 485  
 Terms, 270  
 TertiumNonDatur, 81  
 Test, 298  
 TestPrintQ, 410  
 TestStatistic, 298  
 Time and Date, 36  
 TimeBetweenOrders, 422  
 Time equivalent charter,  
     263  
 TimeLine, 43  
 Timeseries approaches,  
     399  
 TList, 407  
 ToAlgebra, 88  
 ToBeta, 306  
 ToBetaDistribution, 307  
 ToBond, 255  
 ToBorderedMatrix, 51  
 ToCAPMSymbols, 221  
 ToClient, 351  
 ToConditional, 320  
 ToCorrelation, 302  
 ToData, 488  
 ToDataRule, 488  
 ToDatedCash, 255  
 ToDNForm, 81  
 ToDollar, 31  
 ToEquation, 490  
 ToEquationRule, 88  
 ToExpectedUtility, 311  
 ToF, 38  
 ToFirm, 351  
 ToFreemansDataFormat, 406  
 ToFreightSymbols, 265  
 ToIFPair, 534  
 ToInterval, 532  
 ToLogic, 87  
 ToLogNormal, 305  
 ToLoss, 355  
 ToMacroSymbols, 158  
 ToMatrix, 477  
 ToMoneyGraphics, 25  
 ToName, 469  
 ToNewGeneration, 415  
 ToNonIntersectingIFPair,  
     536  
 ToNonNegative, 231  
 ToNonParametric, 231  
 ToPackAlone, 287  
 ToPackWithDalt, 294  
 ToPattern, 470  
 ToPensParade, 181  
 ToPiecewise, 527  
 ToPiecewiseLinear, 527  
 ToPolar, 481  
 ToProb, 318  
 ToProperName, 469  
 ToPropositionalLogic, 85  
 ToProspect, 332  
 ToQueueSymbols, 447  
 ToRisket, 332  
 ToRule, 101  
 ToSequence, 461  
 ToStandardPortfolioForm,  
     205  
 ToStine, 403  
 ToSymbol, 469  
 Total, 176  
 ToUtility, 311  
 ToValue, 271  
 Transactions, 206, 219  
 TransportNumLP, 440  
 TransportNumLPMatrixQ,  
     441  
 TransportNumLP –  
     ShadowPrices, 441  
 Transposant, 478  
 TransposeTensor, 478  
 TransposeToFirst, 478  
 Trees, prospects, 191  
 TriAngle, 512

Trip, 265  
 TruthTable, 88  
 TruthTableRule, 88  
 TruthValue, 88  
 TurnList9, 407  
 Turnover, 126  
 TValues, 298  
 Type I and type II  
   errors, 354  
 Types, 270

## U

UEq, 147  
 UFunction, 145  
 UKPound, 25  
 Uncertainty, 327  
 UncertaintyAversion, 345  
 Uncertainty –  
   DefinitionsPlot, 327  
 UncertaintyPremium, 345  
 UndefinedSymbols, 491  
 UndefinedTerms, 493  
 Unemployment, 165  
 UnemploymentLevel, 168  
 UnitCost, 125  
 UnitFactorCoefficientsQ,  
   118  
 UnitIndex, 482  
 UnitPr, 324  
 Universe, 319  
 UnNumberForm, 468  
 UnPattern, 470  
 Upd, 74  
 Updating, 77  
 Utilisation, 125  
 UtilisationContours, 523  
 Utility, 125

## V

Vacancy, 168  
 Val, 32  
 Value, 489

Value2By2, 186  
 ValueAdded, 125, 164  
 Valueing prospects, 310  
 Valuta, 31  
 Val\$Convert, 33  
 Variables, variates,  
   symbols, 491  
 VariancePr, 301  
 Variates, 492  
 Variations on Solve, 493  
 VdVdtPlot, 425  
 Vehicle, 261  
 VerticalAsymptot, 58  
 Vogel, 442  
 Votes, 106  
 VotingQ, 105  
 vpEq, 147

## W

WageDensity, 175  
 Wealth, 125  
 Weight, 301  
 WeightedMean, 301  
 WeightTotal, 301  
 WeightUnit, 301  
 wEq, 147  
 WesternEurope, 263  
 WhichIFFPair, 533

## X

XEq, 147  
 xLogP, 372  
 xLogX, 372  
 xPowerX, 372  
 XXPlusSpace, 498

## Y

Years, 271  
 YEq, 147  
 Yield, 215

## Z

ZeroCoupon, 213  
 Zero, Indeterminate and  
   Null, 457

## B

$\beta$ , 221

## \$

\$Coefficients, 146  
 \$Convert, 25  
 \$Cool, 539  
 \$Cost, 125  
 \$Economics, 538  
 \$Endogenes, 388  
 \$Euro, 25  
 \$Foreign, 27  
 \$Freight, 257  
 \$FreightDiagram, 265  
 \$Home, 27  
 \$ISEquations, 161  
 \$ISLMEquations, 158  
 \$ISLMPParameters, 160  
 \$LabourMarketEquations,  
   165  
 \$LabourMarketParameters,  
   165  
 \$LMEquations, 162  
 \$OneLag, 387  
 \$Price, 125  
 \$Production, 145  
 \$Profit, 125  
 \$Quantity, 125  
 \$Surprises, 158  
 \$Utility, 145  
 \$ValueAddedEquations, 164  
 \$ValueAddedParameters,  
   164

