The QAM Library

A Gauss Implementation for Quantitative Asset Management Modelling

Thierry Roncalli

University of Evry

This version: May 25, 2009

Contents

Chapter 1

Introduction

1.1 Installation

1. The file *gauss-qam.zip* is a zipped archive file. Copy this file under the root directory of Gauss, for example **D:\GAUSS60**.

2. Unzip the file. Directories will then be created and files will be copied over them:

target_path	*readme.txt*
*target_path***dlib**	DLLs
*target_path***lib**	library file
*target_path***qam****[...]**	example and tutorial files
*target_path***qam****src**	source code files
*target_path***src**	source code files

3. If your root of Gauss is **D:\GAUSS60**, the installation is finished, otherwise you have to modify the paths of the library using notepad or the LibTool. Another way to update the library is to run Gauss, l**og on to the *qam**src* directory**, delete the path with the command **lib qam -n** and add the path to the library with the command **lib qam -a**.

1.2 Getting started

Gauss 6.0.57+ for Windows and the library **optmum** are required to use the **QAM** routines.

1.2.1 readme.txt file

The file *readme.txt* contains last minute information on the **QAM** procedures. Please read it before using them.

1.2.2 Setup

In order to use these procedures, the **QAM** library must be active. This is done by including **QAM** in the LIBRARY statement at the top of your program:

```
library qam;
```

To reset global variables in subsequent executions of the program and in order to load DLLs, the following instruction should be used:

qamSet;

1.3 What is QAM ?

QAM is a Gauss library designed to accompany the French lecture notes "Quantitative Asset Management" [1].

QAM contains the procedures whose list is given below.

1. **Backtest Computing**
   (a) **Add_Fees**: Adds managing and performance fees (yearly basis).
   (b) **Compute_Capitalized_Eonia_Plus**: Computes the backtest of a non-risky investment Eonia + x bp (or Libor + x bp).
   (c) **Compute_Leverage_Strategy**: Leverages a strategy.
   (d) **Compute_Loss_Function**: Computes the loss function.
   (e) **Compute_Maximum_Drawdown**: Computes the maximum drawdown.
   (f) **Compute_Monthly_Basktest**: Computes the backtest of a basket of strategies (with a rebalancing at the end of the month).
   (g) **Compute_Monthly_Basktest2**: Computes the backtest of a basket of funded and non-funded strategies (with a rebalancing at the end of the month).
   (h) **Compute_Weekly_Returns**: Computes weekly returns.
   (i) **Currency_Hedging**: Performs currency hedging.
   (j) **Delete_Fees**: Deletes managing and performance fees (yearly basis).
2. **Optimization Methods**
   (a) **Black_Litterman_mu**: Computes the parameter μ_{cond} in the Black-Litterman model.
   (b) **Black_Litterman_pi**: Computes the parameter π in the Black-Litterman model.
   (c) **Black_Litterman_Solve**: Solves the Black-Litterman model.
   (d) **Compute_ERC_Portfolio**: Computes the ERC portfolio.
   (e) **Compute_Diversification_Ratio**: Computes the diversification ratio $D(x)$.
   (f) **Compute_MDP_Portfolio**: Computes the MDP portfolio.
   (g) **Compute_Risk_Contribution**: Computes the risk contribution decomposition of a portfolio.
   (h) **Lprog**: Solves a linear programming problem using an interior-point algorithm.
   (i) **Qprog_Allocation_Solve**: Solves the portfolio allocation problem.
   (j) **Qprog_Allocation_Solve_With_TC**: Solves the portfolio allocation problem with transaction costs.
   (k) **Qprog_Index_Sampling**: Performs a sampling of an equity index (or a benchmark).
   (l) **Qprog_Index_Solve**: Solves the enhanced indexing problem.

(m) **Qprog_Index_130_30**: Solves the 130/30 indexing problem.

(n) **Qprog_Min_Variance**: Computes the Minimum variance portfolio.

(o) **Qprog_Sharpe Maximize**: Solves the Max Sharpe problem.

(p) **regKernelQuantile**: Non-parametric quantile regression.

(q) **regQuantile**: Linear quantile regression.

(r) **regQP**: Linear regression using quandratic programming.

(s) **regSharpeStyle**: Sharpe style analysis.

(t) **Risk_Budgeting_Solve**: Solves the risk budegting allocation problem.

3. **Numerical Algorithms**

(a) **cosMatrix**: Computes the matrix cosine.

(b) **Compute_Discrete_Simplex**: Discretization of the simplex set.

(c) **Compute_Markov_Generator**: Computes the Markov generator of a one-year transition matrix.

(d) **Compute_Nearest_Correlation**: Computes the nearest correlation matrix.

(e) **ConstantCorrelation**: Generates a constant correlation matrix $C_n(\rho)$.

(f) **Estimate_Markov_Generator**: Estimates a valid Markov generator.

(g) **expMatrix**: Computes the matrix exponential.

(h) **fnMatrix**: Computes a general matrix function.

(i) **gaussHermite**: Computes weights and nodes of Hermite quadrature rules.

(j) **gaussJacobi**: Computes weights and nodes of Jacobi quadrature rules.

(k) **gaussLaguerre**: Computes weights and nodes of Laguerre quadrature rules.

(l) **gaussLegendre**: Computes weights and nodes of Legendre quadrature rules.

(m) **lnMatrix**: Computes the matrix logarithm.

(n) **quadHermite1**: Integrates a 1D function using Gauss-Hermite quadrature.

(o) **quadHermite2**: Integrates a 2D function using Gauss-Hermite quadrature.

(p) **quadLaguerre1**: Integrates a 1D function using Gauss-Laguerre quadrature

(q) **quadLaguerre2**: Integrates a 2D function using Gauss-Laguerre quadrature..

(r) **quadLegendre1**: Integrates a 1D function using Gauss-Legendre quadrature.

(s) **quadLegendre2**: Integrates a 2D function using Gauss-Legendre quadrature.

(t) **random_LC**: Uniform LC generator.

(u) **regFLS**: Flexisble least squares.

(v) **regLDP**: Linear dependency analysis.

(w) **regPCA**: Principal component analysis.

(x) **regSpline**: Spline interpolation and smoothing.

(y) **rndmn**: Simulates normal random vectors using the Cholesky decomposition.

(z) **rndmn_eig**: Simulates normal random vectors using the Eigenvalue decomposition.

(aa) **rndmn_svd**: Simulates normal random vectors using the SVD decomposition.

(ab) **rndn_Box_Muller**: Simulates normal random numbers using the Box-Muller algorithm.

(ac) **rndu_Halton**: Simulates quasi random numbers using Halton sequences.

(ad) **rndu_Hammersley**: Simulates quasi random numbers using Hammersley sequences.

(ae) **simulate_Brownian_Bridge**: Simulates a Brownian bridge.

(af) **simulate_Correlation**: Simulates a random correlation matrix using the random orthogonal algorithm.

(ag) **simulate_Correlation2**: Simulates a random correlation matrix using the nearest correlation algorithm.

(ah) **sinMatrix**: Computes the matrix sine.

(ai) **sqrtMatrix**: Computes the matrix square root.

4. **Pricing Methods**

   (a) **Compute_Dynamic_Delta_Hedging**: Computes the 1D dynamic delta hedging of a derivatives portfolio.

   (b) **Compute_Dynamic_Delta_Hedging2**: Computes the 2D dynamic delta hedging of a derivatives portfolio.

5. **Statistical Tools**

   (a) **regCorr1**: Estimates the 1F correlation model.

   (b) **regCorr2**: Estimates the multi-factor correlation model.

   (c) **regCLS**: Estimates the parameters by the method of conditional least squares.

   (d) **regFactorModel**: Estimates the factor model.

   (e) **regGMM**: Estimates the parameters by the generalized method of moments.

   (f) **regHuber**: Estimates the parameters by the Huber robust method.

   (g) **regKernel**: Estimates the model by the non-parametric Kernel method.

   (h) **regKernelDensity**: Estimates the probability density function by the Gaussian Kernel method.

   (i) **regLAD**: Estimates the parameters by the robust method of least absolute deviations.

   (j) **regLogit**: Estimates the Logit model by ML.

   (k) **regMars**: Estimates the MARS model of Friedman.

   (l) **regMarsForecast**: Forecasting with Mars model.

   (m) **regML**: Estimates the parameters by the method of maximum likelihood.

   (n) **regNLS**: Estimates the parameters by the method of non-linear least squares.

   (o) **regOLS**: Estimates the parameters by the method of ordinary least squares.

   (p) **regOU**: Estimates the parameters of the Ornstein-Uhlenbeck process by ML.

   (q) **regProbit**: Estimates the Probit model by ML.

   (r) **regQR**: Estimates the parameters by the robust quantile regression method.

   (s) **regRestrict**: Fixes parameters in regression models (regGMM, regML, regNLS, regOLS).

(t) **regRobust**: Estimates the parameters by the general robust method (M estimation).

(u) **regTobit**: Estimates the Tobit model by ML.

6. **Time Series Analysis**
7. **Quantitative Strategies**
8. **Stock Screening Methods**
9. **Risk Management**

1.4 Using Online Help

QAM library supports Windows Online Help. Before using the browser, you have to verify that the **QAM** library is activated by the `library` command.

Chapter 2

Quantitative Asset Management

see [1].

Chapter 3

Command References

Add_Fees

■ **Purpose**
Adds managing and performance fees (yearly basis).

■ **Format**
NAV = Add_Fees(dates,GAV,mf,reset,benchmark,pf);

■ **Input**

dates	matrix $N \times 1$, the vector of dates
GAV	matrix $N \times 1$, gross asset value
mf	scalar, managing fees
reset	matrix $M \times 1$, reset dates for performance fees
benchmark	matrix $N \times 1$, prices of the benchmark
pf	scalar, performance fees

■ **Output**

NAV	matrix $N \times 1$, net asset value

■ **Globals**

■ **Remarks**

■ **Source**
backtest.src

Black_Litterman_mu

■ **Purpose**
Computes the parameter μ_{cond} in the Black-Litterman model.

■ **Format**
mu_cond = Black_Litterman_mu(pi_,Gamma_,P,Q,Omega);

■ **Input**

pi _	matrix $N \times 1$, the π vector
Gamma_	matrix $N \times N$, the Γ matrix
P	matrix $K \times N$, the P matrix
Q	matrix $K \times 1$, the Q vector
Omega	matrix $K \times K$, the Ω covariance matrix

■ **Output**

mu_cond	matrix $N \times 1$, the solution μ_{cond}

■ **Globals**

■ **Remarks**
We have:

$$\mu_{\text{cond}} = E\left[\mu \mid \nu = Q\right] = \pi + \Gamma P^{\top}\left(P\Gamma P^{\top} + \Omega\right)^{-1}(Q - P\pi)$$

■ **Source**
optimization.src

Black_Litterman_pi

■ **Purpose**
Computes the parameter π in the Black-Litterman model.

■ **Format**
{pi_,phi} = Black_Litterman_pi(x_star,cov,sh,r,phi);

■ **Input**

x_star	matrix $N \times 1$, the initial portfolio $x^\star$
cov	matrix $N \times N$, the covariance matrix Σ
sh	scalar, the Sharpe ratio $\text{sh}^\star$
r	scalar, the risk-free rate r
phi	scalar, the value of ϕ

■ **Output**

pi_	matrix $N \times 1$, the solution π
phi	scalar, the implied value of ϕ

■ **Globals**

■ **Remarks**
The parameter π is computed with the following formula:

$$\pi = 2\frac{1}{\phi}\Sigma x^\star$$

If $\phi = 0$, we estimate the implied value of ϕ based on the Sharpe ratio:

$$\phi = 2\frac{{x^\star}^\top \Sigma x^\star}{r + \text{sh}^\star \sqrt{{x^\star}^\top \Sigma x^\star}}$$

■ **Source**
optimization.src

Black_Litterman_Solve

Purpose
Solves the Black-Litterman model.

Format
{x,phi,pi_,mu_cond,te} = Black_Litterman_Solve(x_star,cov,sh,tau,P,Q,Omega);

Input

x_star	matrix $N \times 1$, the initial portfolio $x^\star$
cov	matrix $N \times N$, the covariance matrix Σ
sh	scalar, the Sharpe ratio $\text{sh}^\star$
tau	scalar, the value of τ
P	matrix $K \times N$, the P matrix
Q	matrix $K \times 1$, the Q vector
Omega	matrix $K \times K$, the Ω covariance matrix

Output

x	matrix $N \times 1$, the solution of the Black-Litterman model
phi	scalar, the implied value of ϕ
pi_	matrix $N \times 1$, the solution π
mu	matrix $N \times 1$, the solution μ_{cond}
te	scalar, the volatility of the tracking error

Globals

Remarks
The Black-Litterman model is a 3-steps process:

1. First we deduce the implied vector π of the inverse Markowitz problem given Σ and $x^\star$.
2. Then, we specify the views of the manager (P, Q and Ω matrices). By assuming that $\Gamma = \tau\Sigma$, we deduce the conditional expected returns μ_{cond}.
3. Finally, we solve the optimization problem:

$$\begin{array}{rcl} x_{\text{BL}} & = & \arg\min x^\top \Sigma x - \phi x^\top \mu_{\text{cond}} \\ & \text{uc.} & \left\{ \begin{array}{l} \mathbf{1}^\top x = 1 \\ \mathbf{0} \leq x \leq \mathbf{1} \end{array} \right. \end{array}$$

Source
optimization.src

Compute_Capitalized_Eonia_Plus

■ **Purpose**
Computes the backtest of a non-risky investment Eonia + x bp (or Libor + x bp).

■ **Format**
F = Compute_Capitalized_Eonia_Plus(Dates,eonia,x,cn);

■ **Input**

dates	matrix $N \times 1$, vector of dates
eonia	matrix $N \times 1$, Eonia
x	scalar, value of x (in bp)
cn	scalar 1 if the Eonia is capitalized 0 otherwise

■ **Output**

F	matrix $N \times 1$, backtest

■ **Globals**

■ **Remarks**

■ **Source**
backtest.src

Compute_Diversification_Ratio

■ **Purpose**
Computes the diversification ratio $D(x)$.

■ **Format**
D = Compute_Diversification_Ratio(x,cov);

■ **Input**

x	matrix $N \times 1$, the portfolio x
cov	matrix $N \times N$, the covariance matrix Σ

■ **Output**

D	scalar, value of the diversification ratio $D(x)$

■ **Globals**

■ **Remarks**
The diversification ratio $D(x)$ is defined by:

$$D(x) = \frac{x^{\top}\boldsymbol{\sigma}}{\sqrt{x^{\top}\Sigma x}} = \frac{\sqrt{x^{\top}\Sigma(1)\,x}}{\sqrt{x^{\top}\Sigma x}}$$

■ **Source**
optimization.src

Compute_Discrete_Simplex

■ **Purpose**
Discretization of the simplex set.

■ **Format**
u = Compute_Discrete_Simplex(d,p,s);

■ **Input**

d	scalar, dimension d
p	scalar, number of points p
s	scalar, 0 for no restriction, otherwise s

■ **Output**

u	matrix $N \times d$

■ **Globals**

■ **Remarks**
The simplex set is defined by :

$$\left\{ u : u \in [0,1]^d, \sum_{i=1}^{d} u_i \leq 1 \right\}$$

We may impose a restriction $\sum_{i=1}^{d} u_i = s$. The procedure uses a discretization of $[0,1]$ into $p+1$ points.

■ **Source**
numerics.src

Compute_Dynamic_Delta_Hedging

■ **Purpose**
Computes the 1D dynamic delta hedging of a derivatives portfolio.

■ **Format**
{dh,PnL,PnL1,PnL2,Results} =
Compute_Dynamic_Delta_Hedging(notional,option,optionProc,deltaProc,t,S,r,cn);

■ **Input**

notional	scalar, notional
option	scalar, actual premium of the option
optionProc	scalar, pointer to a procedure that computes the premium
deltaProc	scalar, pointer to a procedure that computes the delta
t	vector $N \times 1$, trading dates
S	vector $N \times 1$, prices of the underlying asset
r	vector $N \times 1$, interest rate
cn	scalar, 1 for saving all the results

■ **Output**

dh	vector $N \times 1$, dynamic hedging
PnL	vector $N \times 1$, PnL of the option and hedging portfolio
PnL1	vector $N \times 1$, PnL part of the option portfolio
PnL2	vector $N \times 1$, PnL part of the hedging portfolio
Results	matrix $N \times 12$, results

■ **Globals**

■ **Remarks**

■ **Source**
pricing.src

Compute_Dynamic_Delta_Hedging2

■ **Purpose**
Computes the 2D dynamic delta hedging of a derivatives portfolio.

■ **Format**
{dh,PnL,PnL1,PnL2,Results} =
Compute_Dynamic_Delta_Hedging(notional,option,optionProc,deltaProc,t,S,r,cn);

■ **Input**

notional	scalar, notional
option	scalar, actual premium of the option
optionProc	scalar, pointer to a procedure that computes the premium
deltaProc	scalar, pointer to a procedure that computes the 2D delta
t	vector $N \times 1$, trading dates
S	matrix $N \times 2$, prices of the underlying asset
r	vector $N \times 1$, interest rate
cn	scalar, 1 for saving all the results

■ **Output**

dh	vector $N \times 1$, dynamic hedging
PnL	vector $N \times 1$, PnL of the option and hedging portfolio
PnL1	vector $N \times 1$, PnL part of the option portfolio
PnL2	vector $N \times 1$, PnL part of the hedging portfolio
Results	matrix $N \times 12$, results

■ **Globals**

■ **Remarks**

■ **Source**
pricing.src

Compute_ERC_Portfolio

■ **Purpose**
Computes the ERC portfolio.

■ **Format**
{x,vol,mr,tr,rc} = Compute_ERC_Portfolio(x0,cov);

■ **Input**

x0	matrix $N \times 1$, starting values
cov	matrix $N \times N$, the covariance matrix Σ

■ **Output**

x	matrix $N \times 1$, the ERC portfolio
vol	scalar, the volatility of the ERC portfolio
mr	matrix $N \times 1$, marginal risk contributions
tr	matrix $N \times 1$, total risk contributions
rc	matrix $N \times 1$, risk contributions (in %)

■ **Globals**

■ **Remarks**
The volatility of the portfolio x verifies the Euler decomposition:

$$\sigma(x) = \sum_{i=1}^{n} x_i \times \frac{\partial \sigma(x)}{\partial x_i}$$

The ERC (*Equally-weighted Risk Contributions*) portfolio is defined as:

$$x_i \times \frac{\partial \sigma(x)}{\partial x_i} = x_j \times \frac{\partial \sigma(x)}{\partial x_j}$$

with $\mathbf{1}^\top x = 1$ et $\mathbf{0} \leq x \leq \mathbf{1}$.

■ **Source**
optimization.src

Compute_Leverage_Strategy

■ **Purpose**
Leverages a strategy

■ **Format**
F = Compute_Leverage_Strategy(dates,S,fc,lev);

■ **Input**

dates	matrix $N \times 1$, vector of dates
S	matrix $N \times 1$, values of the strategy
fc	matrix $N \times 1$, funding costs
lev	scalar, value of the leverage

■ **Output**

F	matrix $N \times 1$, values of the leveraged strategy

■ **Globals**

■ **Remarks**

■ **Source**
backtest.src

Compute_Loss_Function

■ **Purpose**
Computes the loss function.

■ **Format**
{Dt,L} = Compute_Loss_Function(x,Dt);

■ **Input**

x	matrix $N \times M$, times series
Dt	matrix $M \times 1$, values of the time index Δ_t

■ **Output**

Dt	matrix $M \times 1$, values of the time index Δ_t
L	matrix $M \times 1$, values of the loss function

■ **Globals**

■ **Remarks**
If Dt is a positive scalar, the procedure generates the sequence $1, 2, \ldots, Dt$. If Dt is a negative scalar, the procedure compute $L\,(-Dt)$.

■ **Source**
backtest.src

Compute_Markov_Generator

■ **Purpose**
Computes the Markov generator of a one-year transition matrix.

■ **Format**
{lnP,Lambda} = Compute_Markov_Generator(P);

■ **Input**

P	matrix $N \times N$, the one-year transition matrix

■ **Output**

lnP	matrix $N \times N$, the matrix logarithm of P
Lambda	matrix $N \times N$, the Markov generator

■ **Globals**

■ **Remarks**
We use the algorithm described in Israel, R.B., Rosenthal J.S. and Wei J.Z., Finding Generators for Markov Chains via Empirical Transition Matrices with Applications to Credit Ratings, *Mathematical Finance*, 11, 2001.

■ **Source**
numerics.src

Compute_Maximum_Drawdown

■ **Purpose**
Computes the maximum drawdown.

■ **Format**
{maxDD,startDD,endDD} = Compute_Maximum_Drawdown(x,cn);

■ **Input**

x	matrix $N \times 1$, returns or prices
cn	scalar
	0 if returns
	1 if prices

■ **Output**

maxDD	scalar, the maximum drawdown
startDD	scalar, begining of the MDD period
endDD	scalar, end of the MDD period

■ **Globals**

■ **Remarks**

■ **Source**
backtest.src

Compute_MDP_Portfolio

■ **Purpose**
Computes the MDP portfolio.

■ **Format**
{x,vol,mr,tr,rc} = Compute_MDP_Portfolio(x0,cov);

■ **Input**

x0	matrix $N \times 1$, starting values
cov	matrix $N \times N$, the covariance matrix Σ

■ **Output**

x	matrix $N \times 1$, the MDP portfolio
vol	scalar, the volatility of the MDP portfolio
mr	matrix $N \times 1$, marginal risk contributions
tr	matrix $N \times 1$, total risk contributions
rc	matrix $N \times 1$, risk contributions (in %)

■ **Globals**

■ **Remarks**
The MDP (*Most-Diversified Portfolio*) portfolio is the solution of the following optimization problem:

$$\begin{aligned} x^\star &= \arg\max D(x) \\ \text{u.c.} &\quad \mathbf{1}^\top x = 1 \text{ et } \mathbf{0} \leq x \leq \mathbf{1} \end{aligned}$$

where $D(x)$ is the diversification ratio.

■ **Source**
optimization.src

Compute_Monthly_Basket

■ **Purpose**
Computes the backtest of a basket of strategies (with a rebalancing at the end of the month).

■ **Format**
F = Compute_Monthly_Backtest(dates,weights,basket);

■ **Input**

dates	matrix $N \times 1$, vector of dates
weights	matrix $N \times K$, matrix of weights
basket	matrix $N \times K$, basket of strategies

■ **Output**

F	matrix $N \times 1$, backtest

■ **Globals**

■ **Remarks**

■ **Source**
backtest.src

Compute_Monthly_Basket2

■ **Purpose**
Computes the backtest of a basket of funded and non-funded strategies (with a rebalancing at the end of the month).

■ **Format**
F = Compute_Monthly_Backtest((dates,wfs,fs,wnfs,nfs);

■ **Input**

dates	matrix $N \times 1$, vector of dates
wfs	matrix $N \times K$, matrix of weights of the funded strategies
fs	matrix $N \times K$, funded strategies
wnfs	matrix $N \times K$, matrix of weights of the non funded strategies
nfs	matrix $N \times K$, non funded strategies

■ **Output**

F	matrix $N \times 1$, backtest

■ **Globals**

■ **Remarks**

■ **Source**
backtest.src

Compute_Nearest_Correlation

■ **Purpose**
Computes the nearest correlation matrix.

■ **Format**
C = Compute_Nearest_Correlation(A);

■ **Input**

A	matrix $N \times N$

■ **Output**

C	matrix $N \times N$

■ **Globals**

_corr_maxit	scalar, maximum number of iterations (default = 200)
_corr_tol	scalar, tolerance value (default = 1e-10)

■ **Remarks**

■ **Source**
numerics.src

Compute_Risk_Contribution

■ **Purpose**
Computes the risk contribution decomposition of a portfolio.

■ **Format**
{vol,mr,tr,rc} = Compute_Risk_Contribution(x,cov);

■ **Input**

x	matrix $N \times 1$, the portfolio x
cov	matrix $N \times N$, the covariance matrix Σ

■ **Output**

vol	scalar, the volatility $\sigma(x)$
mr	matrix $N \times 1$, marginal risk contributions
tr	matrix $N \times 1$, total risk contributions
rc	matrix $N \times 1$, risk contributions (in %)

■ **Globals**

■ **Remarks**
Let Σ be the covariance matrix of asset returns. The volatility of the portfolio x is defined by:

$$\sigma(x) = \sqrt{x^\top \Sigma x}$$

It verifies the Euler decomposition:

$$\sigma(x) = \sum_{i=1}^{n} x_i \times \frac{\partial \sigma(x)}{\partial x_i}$$

where $\partial_{x_i} \sigma(x)$ is the marginal risk. The total risk contribution is $\sigma_i(x) = x_i \times \sigma_i(x)$ whereas the relative risk contribution is the ratio of the total risk contribution on the volatility.

■ **Source**
optimization.src

Compute_Weekly_Returns

Purpose
Computes weekly returns.

Format
{wd,wr} = Compute_Weekly_Returns(firstDate,lastDate,day,dates,prices);

Input

firstDate	scalar, first date
llstDate	scalar, last date
day	scalar, day of the week (1 for monday, 2 for tuesday, etc.)
dates	matrix $N \times 1$, vector of dates
prices	matrix $N \times 1$, vector of prices

Output

wd	matrix $M \times 1$, weekly dates
wr	matrix $M \times 1$, weekly returns

Globals

Remarks

Source
backtest.src

ConstantCorrelation

■ **Purpose**
Generates a constant correlation matrix $C_n(\rho)$.

■ **Format**
c = ConstantCorrelation(rho,n);

■ **Input**

rho	scalar, value of ρ
n	scalar, dimension of the correlation matrix

■ **Output**

c	matrix $n \times n$

■ **Globals**

■ **Remarks**

■ **Source**
numerics.src

Currency_Hedging

■ **Purpose**
Performs currency hedging.

■ **Format**
hedged_prices = Currency_Hedging(dates,prices,ir,ir_star);

■ **Input**

dates	matrix $N \times 1$, vector of dates
prices	matrix $N \times 1$, local currency prices
ir	matrix $N \times 1$, interest rate r
ir_star	matrix $N \times 1$, interest rate $r^\star$

■ **Output**

hedged_prices	matrix $N \times 1$, vector of hedged prices

■ **Globals**

■ **Remarks**

■ **Source**
backtest.src

cosMatrix

■ **Purpose**
Computes the matrix cosine.

■ **Format**
B = cosMatrix(A);

■ **Input**

A	matrix $N \times N$

■ **Output**

B	matrix $N \times N$

■ **Globals**

■ **Remarks**
We have:

$$B = \frac{e^{iA} + e^{-iA}}{2}$$

■ **Source**
numerics.src

Delete_Fees

■ **Purpose**
Deletes managing and performance fees (yearly basis).

■ **Format**
GAV = Delete_Fees(dates,NAV,mf,benchmark,pf);

■ **Input**

dates	matrix $N \times 1$, the vector of dates
NAV	matrix $N \times 1$, net asset value
mf	scalar, managing fees
benchmark	matrix $N \times 1$, prices of the benchmark
pf	scalar, performance fees

■ **Output**

GAV	matrix $N \times 1$, gross asset value

■ **Globals**

■ **Remarks**
We assume that reset dates correspond to the end of every year.

■ **Source**
backtest.src

Estimate_Markov_Generator

■ **Purpose**
Estimates a valid Markov generator.

■ **Format**
{Lambda1,Lambda2} = Estimate_Markov_Generator(Lambda);

■ **Input**

Lambda	matrix $N \times N$, the non-valid Markov generator

■ **Output**

Lambda1	matrix $N \times N$, estimate of the Markov generator (algorithm [IRM-1])
Lambda1	matrix $N \times N$, another estimate of the Markov generator (algorithm [IRM-2])

■ **Globals**

■ **Remarks**
We use the algorithms described in Israel, R.B., Rosenthal J.S. and Wei J.Z., Finding Generators for Markov Chains via Empirical Transition Matrices with Applications to Credit Ratings, *Mathematical Finance*, 11, 2001.

■ **Source**
numerics.src

expMatrix

■ **Purpose**
Computes the matrix exponential.

■ **Format**
B = lnMatrix(A);

■ **Input**

A	matrix $N \times N$

■ **Output**

B	matrix $N \times N$

■ **Globals**

■ **Remarks**
The definition of the matrix exponential is:

$$B = I + A + \frac{A^2}{2!} + \frac{A3}{3!} + \ldots$$

We use the algorithm 11.3.1 described in Golub, G.H. and C.F. Van Loan [1989], Matrix Computations, second edition, Johns Hopkins University Press.

■ **Source**
numerics.src

fnMatrix

■ **Purpose**
Computes a general matrix function.

■ **Format**
B = fnMatrix(A,&func);

■ **Input**

A	matrix $N \times N$
&func	scalar, procedure pointer to a function

■ **Output**

B	matrix $N \times N$

■ **Globals**

■ **Remarks**
We use the algorithm 11.1.1 described in Golub, G.H. and C.F. Van Loan [1989], Matrix Computations, second edition, Johns Hopkins University Press.

■ **Source**
numerics.src

gaussHermite

■ **Purpose**
Computes weights and nodes of Hermite quadrature rules.

■ **Format**
{x,w} = gaussHermite(N,alpha);

■ **Input**

N	scalar, quadrature order
alpha	scalar, parameter α

■ **Output**

x	matrix $N \times 1$, value of nodes
w	matrix $N \times 1$, weights

■ **Globals**

_quad_mtd	scalar (default = 1) 1 for the Newton-Raphson algorithm 2 for the SVD algorithm

■ **Remarks**

■ **Source**
numerics.src

gaussJacobi

■ **Purpose**
Computes weights and nodes of Jacobi quadrature rules.

■ **Format**
{x,w} = gaussJacobi(N,alpha,beta);

■ **Input**

N	scalar, quadrature order
alpha	scalar, parameter α
beta	scalar, parameter β

■ **Output**

x	matrix $N \times 1$, value of nodes
w	matrix $N \times 1$, weights

■ **Globals**

_quad_mtd	scalar (default = 1) 1 for the Newton-Raphson algorithm 2 for the SVD algorithm

■ **Remarks**

■ **Source**
numerics.src

gaussLaguerre

■ **Purpose**
Computes weights and nodes of Laguerre quadrature rules.

■ **Format**
{x,w} = gaussLaguerre(N,alpha);

■ **Input**

N	scalar, quadrature order
alpha	scalar, parameter α

■ **Output**

x	matrix $N \times 1$, value of nodes
w	matrix $N \times 1$, weights

■ **Globals**

_quad_mtd	scalar (default = 1) 1 for the Newton-Raphson algorithm 2 for the SVD algorithm

■ **Remarks**

■ **Source**
numerics.src

gaussLegendre

■ **Purpose**
Computes weights and nodes of Legendre quadrature rules.

■ **Format**
{x,w} = gaussLegendre(N);

■ **Input**

N	scalar, quadrature order

■ **Output**

x	matrix $N \times 1$, value of nodes
w	matrix $N \times 1$, weights

■ **Globals**

_quad_mtd	scalar (default = 1) 1 for the Newton-Raphson algorithm 2 for the SVD algorithm

■ **Remarks**

■ **Source**
numerics.src

lnMatrix

■ **Purpose**
Computes the matrix logarithm.

■ **Format**
B = lnMatrix(A);

■ **Input**

A	matrix $N \times N$

■ **Output**

B	matrix $N \times N$

■ **Globals**

■ **Remarks**
We have:

$$\ln\left(e^{A}\right) = A$$

■ **Source**
numerics.src

Lprog

■ **Purpose**
Solves a linear programming problem using an interior-point algorithm.

■ **Format**
x = Lprog(c,A,B,ub,sv);

■ **Input**

c	matrix $N \times 1$, the c vector
A	matrix $P \times N$, the A matrix
B	matrix $P \times 1$, the B vector
ub	matrix $N \times 1$, the upper bounds x_+
sv	matrix $N \times 1$, starting values

■ **Output**

x	matrix $N \times 1$

■ **Globals**

■ **Remarks**
The optimization program is:

$$\min_x \left\{ c^\top x \mid Ax = B, 0 \leq x \leq x_+ \right\}$$

■ **Source**
optimization.src

Qprog_Allocation_Solve

Purpose

Solves the portfolio allocation problem.

Format

{x,mu,sigma,phi,retcode} = Qprog_Allocation_Solve(er,cov,A,B,C,D,Bounds,Targets,Method);

Input

er	matrix $N \times 1$, the vector μ of expected returns
cov	matrix $N \times N$, the covariance matrix Σ
A	matrix $M \times N$, the A matrix
B	matrix $M \times 1$, the B vector
C	matrix $L \times N$, the C matrix
D	matrix $L \times 1$, the D vector
Bounds	matrix $N \times 2$, lower and upper bounds
Targets	matrix $K \times 1$, vector of targets (ϕ, $\mu^\star$ or $\sigma^\star$)
Method	scalar 0 for the ϕ-problem 1 for the μ-problem 2 for the σ-problem

Output

x	matrix $N \times K$, optimal portfolios
mu	matrix $K \times 1$, the expected return of optimal portfolios
sigma	matrix $K \times 1$, the volatility of optimal portfolios
phi	matrix $K \times 1$, the implied value of ϕ
retcode	matrix $K \times 1$, the return code of the Qprog algorithm

Globals

Remarks

The ϕ-problem is defined by:

$$\begin{aligned} x^\star(\phi) &= \arg\min x^\top \Sigma x - \phi x^\top \mu \\ &\text{u.c.} \quad A^\top x = B,\ C^\top x \geq D \text{ and } x_- \leq x \leq x_+ \end{aligned}$$

For the μ- and σ-problems, we have to find the optimal value of ϕ such that:

$$\mu(x^\star(\phi)) = \mu^\star$$

or:

$$\sigma(x^\star(\phi)) = \sigma^\star$$

We define the nature of the allocation problem with the variable `Method` whereas the target values of ϕ, $\mu^\star$ and $\sigma^\star$ are initialized by the variable `Targets`.

Source

optimization.src

Qprog_Allocation_Solve_With_TC

Purpose

Solves the portfolio allocation problem with transaction costs.

Format

{x,x_m,x_p,mu,sigma,phi,retcode} =
Qprog_Allocation_Solve_With_TC(er,cov,A,B,C,D,Bounds,Targets,Method,x0,TC_m,TC_p);

Input

er	matrix $N \times 1$, the vector μ of expected returns
cov	matrix $N \times N$, the covariance matrix Σ
A	matrix $M \times N$, the A matrix
B	matrix $M \times 1$, the B vector
C	matrix $L \times N$, the C matrix
D	matrix $L \times 1$, the D vector
Bounds	matrix $N \times 2$, lower and upper bounds
Targets	matrix $K \times 1$, the vector of targets (ϕ, $\mu^\star$ or $\sigma^\star$)
Method	scalar 0 for the ϕ-problem 1 for the μ-problem 2 for the σ-problem
x0	matrix $N \times 1$, the actual portfolio x^0
TC_m	matrix $N \times 1$, transaction costs c_-
TC_p	matrix $N \times 1$, transaction costs c_+

Output

x	matrix $N \times K$, optimal portfolios
x_m	matrix $N \times K$, values of x_-
x_p	matrix $N \times K$, values of x_+
mu	matrix $K \times 1$, the expected return of optimal portfolios
sigma	matrix $K \times 1$, the volatility of optimal portfolios
phi	matrix $K \times 1$, the implied value of ϕ
retcode	matrix $K \times 1$, the return code of the Qprog algorithm

Globals

Remarks

Let x^0 be the actual portfolio. We note x the optimal portfolio. We have:

$$x_i = x_i^0 - x_i^- + x_i^+$$

where x_i^- and x_i^+ are negative and positive adjustments. Let c_i^- and c_i^+ be the bid and ask transaction costs. The ϕ-problem of portfolio allocation becomes :

$$x^\star = \arg\min x^\top \Sigma x - \phi \left(\sum x_i \mu_i - \sum x_i^- c_i^- - \sum x_i^+ c_i^+ \right)$$

with:

$$\sum x_i + \sum x_i^- c_i^- + \sum x_i^+ c_i^+ = 1$$

Source

optimization.src

Qprog_Index_Sampling

■ **Purpose**
Performs a sampling of an equity index (or a benchmark).

■ **Format**
{x,te} = Qprog_Index_Sampling(weights,cov,size,p);

■ **Input**

weights	matrix $N \times 1$, weights b of the benchmark
cov	matrix $N \times N$, the covariance matrix Σ
size	maximum number of assets with non-zero weights
p	scalar if $p < 1$, the pourcentage of weights to delete at each step of the triming algorithm if $p > 1$, the number of assets to delete at each step of the triming algorithm

■ **Output**

x	matrix $N \times 1$, the portfolio of replication
te	scalar, the volatility of the tracking error

■ **Globals**

■ **Remarks**

■ **Source**
optimization.src

Qprog_Index_Solve

■ **Purpose**
Solves the enhanced indexing problem.

■ **Format**
{x,te,alpha} = Qprog_Index_Solve(weights,cov,A,B,C,D,Bounds,phi,er,active,sv);

■ **Input**

weights	matrix $N \times 1$, weights b of the benchmark
cov	matrix $N \times N$, the covariance matrix Σ
A	matrix $M \times N$, the A matrix
B	matrix $M \times 1$, the B vector
C	matrix $L \times N$, the C matrix
D	matrix $L \times 1$, the D vector
Bounds	matrix $N \times 2$, lower and upper bounds
phi	scalar, value of ϕ
er	matrix $N \times 1$, the vector μ of expected retruns
active	matrix $N \times 1$, the vector of active and non-active assets
sv	matrix $N \times 1$, starting values for the optimization algorithm

■ **Output**

x	matrix $N \times 1$, the optimal portfolio
te	scalar, the volatility of the tracking error
alpha	scalar, the outperformance of the optimal portfolio

■ **Globals**

■ **Remarks**
We consider the optimization problem :

$$x^{\star} = \arg\min \sigma^2 (x \mid b) - \phi\mu (x \mid b)$$

where $\sigma (x \mid b) = \sqrt{(x-b)^{\top} \Sigma (x-b)}$ is the volatility of the tracking error and $\mu (x \mid b) = (x-b)^{\top} \mu$ is the outperformance of the portfolio with respect to the benchmark.

■ **Source**
optimization.src

Qprog_Index_130_30

Purpose

Solves the 130/30 indexing problem.

Format

{x,weight_in,lev,beta,info} = Qprog_Index_130_30(weights,cov,max_te,size,p,er,active,sl);

Input

weights	matrix $N \times 1$, weights of the benchmark
cov	matrix $N \times N$, the covariance matrix Σ
max_te	scalar, maximum of the volatility of tracking error
size	scalar, minimum number of assets with non-zero weights
p	scalar, if $p < 1$, the pourcentage of weights to delete at each step of the triming algorithm if $p > 1$, the number of assets to delete at each step of the triming algorithm
er	matrix $N \times 1$, the vector μ of expected returns
active	matrix $N \times 1$, the vector of active and non-active assets
sl	scalar, the value of the short leverage ℓ

Output

x	matrix $N \times 1$, the optimal portfolio
weight_in	scalar, overlapping of weights
lev	scalar, the effective leverage
beta	scalar, the beta of the optimal portfolio
info	matrix 7×1 info[1] = the number of assets in the benchmark info[2] = the volatility of the benchmark info[3] = the number of assets with non-zero weights info[4] = the volatility of the optimal portfolio info[5] = the volatility of the tracking error info[6] = the number of assets of the short leg info[7] = the number of assets of the long leg

Remarks

We consider the classic enhanced indexing problem:

$$\begin{array}{rcl} x^\star & = & \arg\max \mu\left(x \mid b\right) \\ & \text{u.c..} & \mathbf{1}^\top x = 1 \text{ and } \sigma\left(x \mid b\right) \leq \sigma^\star \end{array}$$

But we replace the long-only constraint $0 \leq x \leq 1$ by the following restrictions :

$$\left\{ \begin{array}{l} -1 \leq x \leq 1 \\ \displaystyle\sum_{i=1}^{n} \max\left(0, x_i\right) = 1 + \ell \\ \displaystyle\sum_{i=1}^{n} \max\left(0, -x_i\right) = \ell \end{array} \right.$$

ℓ is the short leverage and we have $\sum_{i=1}^{n} |x_i| = 1 + 2\ell$.

Source

optimization.src

Qprog_Min_Variance

■ **Purpose**
Computes the Minimum variance portfolio.

■ **Format**
{x,sigma,retcode} = Qprog_Min_Variance(cov,A,B,C,D,Bounds);

■ **Input**

cov	matrix $N \times N$, the covariance matrix Σ
A	matrix $M \times N$, the A matrix
B	matrix $M \times 1$, the B vector
C	matrix $L \times N$, the C matrix
D	matrix $L \times 1$, the D vector
Bounds	matrix $N \times 2$, lower and upper bounds

■ **Output**

x	matrix $N \times 1$, the optimal portfolio
sigma	scalar, the volatility of the optimal portfolio
retcode	scalar, the return code of the Qprog algorithm

■ **Globals**

■ **Remarks**
We have:

$$\begin{array}{rcl} x^\star & = & \arg\min x^\top \Sigma x \\ & \text{u.c.} & A^\top x = B,\, C^\top x \geq D \text{ and } x_- \leq x \leq x_+ \end{array}$$

■ **Source**
optimization.src

Qprog_Sharpe Maximize

■ **Purpose**
Solves the Max Sharpe problem.

■ **Format**
{x,mu,sigma,Sharpe,phi} = Qprog_Sharpe_Maximize(er,cov,A,B,C,D,Bounds,r,min _vol,sv);

■ **Input**

er	matrix $N \times 1$, the vector of μ expected returns
cov	matrix $N \times N$, the covariance matrix Σ
A	matrix $M \times N$, the A matrix
B	matrix $M \times 1$, the B vector
C	matrix $L \times N$, the C matrix
D	matrix $L \times 1$, the D vector
Bounds	matrix $N \times 2$, lower and upper bounds
r	scalar, the risk-free interest rate
min_vol	scalar, the minimum of volatility to consider
sv	matrix $N \times 1$, starting values

■ **Output**

x	matrix $N \times 1$, the optimal portfolio
mu	scalar, the expected return of the optimal portfolio
sigma	scalar, the volatility of the optimal portfolio
Sharpe	scalar, the Sharpe ratio of the optimal portfolio
phi	scalar, the implied value of ϕ

■ **Globals**

■ **Remarks**
We have:

$$\begin{aligned} x^{\star} &= \arg\max \frac{\mu(x) - r}{\sigma(x)} \\ &\text{u.c.} \quad A^{\top}x = B,\ C^{\top}x \geq D \text{ and } x_{-} \leq x \leq x_{+} \end{aligned}$$

■ **Source**
optimization.src

quadHermite1

■ **Purpose**
Integrates a 1D function using Gauss-Hermite quadrature.

■ **Format**
I = quadHermite1(&f,N);

■ **Input**

&f	scalar, pointer to the procedure containing the function $f(x)$ to be integrated
N	scalar, order of the quadrature

■ **Output**

I	scalar, estimated integral

■ **Globals**

_quad_mtd	scalar (default = 1) 1 for the Newton-Raphson algorithm 2 for the SVD algorithm

■ **Remarks**
The lower integration bound is $-\infty$ whereas the upper integration bound is $+\infty$.

■ **Source**
numerics.src

quadHermite2

Purpose

Integrates a 2D function using Gauss-Hermite quadrature.

Format

I = quadHermite2(&f,N);

Input

&f	scalar, pointer to the procedure containing the function $f(x, y)$ to be integrated
N	scalar, order of the quadrature

Output

I	scalar, estimated integral

Globals

_quad_mtd	scalar (default = 1) 1 for the Newton-Raphson algorithm 2 for the SVD algorithm

Remarks

Lower integration bounds are $-\infty$ whereas upper integration bounds are $+\infty$.

Source

numerics.src

quadLaguerre1

■ **Purpose**
Integrates a 1D function using Gauss-Laguerre quadrature.

■ **Format**
I = quadLaguerre1(&f,N);

■ **Input**

&f	scalar, pointer to the procedure containing the function $f(x)$ to be integrated
N	scalar, order of the quadrature

■ **Output**

I	scalar, estimated integral

■ **Globals**

_quad_mtd	scalar (default = 1) 1 for the Newton-Raphson algorithm 2 for the SVD algorithm

■ **Remarks**
The lower integration bound is 0 whereas the upper integration bound is $+\infty$.

■ **Source**
numerics.src

quadLaguerre2

■ **Purpose**
Integrates a 2D function using Gauss-Laguerre quadrature.

■ **Format**
I = quadLaguerre2(&f,N);

■ **Input**

&f	scalar, pointer to the procedure containing the function $f(x, y)$ to be integrated
N	scalar, order of the quadrature

■ **Output**

I	scalar, estimated integral

■ **Globals**

_quad_mtd	scalar (default = 1) 1 for the Newton-Raphson algorithm 2 for the SVD algorithm

■ **Remarks**
Lower integration bounds are 0 whereas upper integration bounds are $+\infty$.

■ **Source**
numerics.src

quadLegendre1

■ **Purpose**
Integrates a 1D function using Gauss-Legendre quadrature.

■ **Format**
I = quadLegendre1(&f,xl,N);

■ **Input**

&f	scalar, pointer to the procedure containing the function $f(x)$ to be integrated
xl	matrix $2 \times M$, the limits of x
N	scalar, order of the quadrature

■ **Output**

I	matrix $M \times 1$, estimated integrals

■ **Globals**

_quad_mtd	scalar (default = 1) 1 for the Newton-Raphson algorithm 2 for the SVD algorithm

■ **Remarks**

■ **Source**
numerics.src

quadLegendre2

Purpose

Integrates a 2D function using Gauss-Legendre quadrature.

Format

I = quadLegendre1(&f,xl,yl,N);

Input

&f	scalar, pointer to the procedure containing the function $f(x, y)$ to be integrated
xl	matrix $2 \times M$, the limits of x
yl	matrix $2 \times M$, the limits of y
N	scalar, order of the quadrature

Output

I	matrix $M \times 1$, estimated integrals

Globals

_quad_mtd	scalar (default = 1) 1 for the Newton-Raphson algorithm 2 for the SVD algorithm

Remarks

Source

numerics.src

quadLegendre3

■ **Purpose**
Integrates a 3D function using Gauss-Legendre quadrature.

■ **Format**
I = quadLegendre1(&f,xl,yl,zl,N);

■ **Input**

&f	scalar, pointer to the procedure containing the function $f(x, y, z)$ to be integrated
xl	matrix $2 \times M$, the limits of x
yl	matrix $2 \times M$, the limits of y
zl	matrix $2 \times M$, the limits of z
N	scalar, order of the quadrature

■ **Output**

I	matrix $M \times 1$, estimated integrals

■ **Globals**

_quad_mtd	scalar (default = 1) 1 for the Newton-Raphson algorithm 2 for the SVD algorithm

■ **Remarks**

■ **Source**
numerics.src

random_LC

■ **Purpose**
Uniform LC generator

■ **Format**
{x,u,seed} = random_LC(seed,a,b,m,n);

■ **Input**

seed	scalar, seed value
a	scalar, value of a
b	scalar, value of b
m	scalar, value of m
n	scalar, number of simulates

■ **Output**

x	matrix $n \times 1$, integrer random numbers
u	matrix $n \times 1$, iuniform random numbers
newseed	scalar, new seed value

■ **Globals**

■ **Remarks**
To obtain the Gauss LC generator, you may specify `m = error(0)`.

■ **Source**
numerics.src

regCorr1

■ **Purpose**
Estimates the 1F correlation model.

■ **Format**
{rho,esc,emc} = regCorr1(data);

■ **Input**

data	matrix $N \times K$

■ **Output**

rho	scalar, estimated value of ρ
esc	matrix $K \times K$, estimated correlation matrix
emc	matrix $K \times K$, empirical correlation matrix

■ **Globals**

■ **Remarks**

■ **Source**
statistics.src

regCorr2

Purpose

Estimates the multi-factor correlation model.

Format

{rho_c,rho_f,esc,emc} = regCorr2(data,common,factors,sv);

Input

data	matrix $N \times K$
common	scalar, value of the correlation for the common factor (1 if you want to estimate this parameter)
factors	vector $K \times 1$, indicates the link between variables and factors
sv	vector $F \times 1$ or $(F+1) \times 1$, starting values

Output

rho_c	scalar, estimated value of ρ
rho_f	vector $F \times 1$, estimated value of ρ_f
esc	matrix $K \times K$, estimated correlation matrix
emc	matrix $K \times K$, empirical correlation matrix

Globals

Remarks

Source

statistics.src

regCLS

Purpose

Estimates the parameters by the method of conditional least squares.

Format

{theta,stderr,cov,u} = regCLS(&f,sv,RR,r);

Input

&f	scalar, pointer to a procedure which computes the residuals $u = f(\theta)$
sv	vector $G \times 1$, starting values for the optimization algorithm
RR	matrix $K \times G$, matrix R of the implied restrictions $\theta = R\gamma + r$
r	vector $G \times 1$, vector r of the implied restrictions $\theta = R\gamma + r$

Output

theta	vector $K \times 1$, estimated parameters $\hat{\theta}$
stderr	vector $K \times 1$, standard errors of the estimated parameters
cov	matrix $K \times K$, covariance matrix of the estimated parameters
u	vector $N \times 1$, residuals

Globals

_ParNames	vector $K \times 1$, name of the parameters (default = 0)
_reg_cov	scalar 0, do not estimate the covariance matrix scalar 1 (default), OPG estimator
_reg_gradp	scalar, pointer to a procedure which computes the Jacobian matrix of $f(\theta)$ 0 to use the numerical gradient
_reg_NoIter	scalar, 1 to suppress the optimisation step
_reg_opalgr	scalar, numerical algorithm (default = 2) 1 for the SD algorithm (steepest descent) 2 for the BFGS algorithm (Broyden, Fletcher, Goldfarb, Shanno) 3 for the scaled BFGS algorithm 4 for the Self-Scaling DFP algorithm (Davidon, Fletcher, Powell) 5 for the Newton-Raphson algorithm 6 for the Polak-Ribiere Conjugate Gradient algorithm
_reg_PrintIters	scalar 1 (default), print the iterations scalar 0, do not print the iterations
_reg_Output	string, name of the file to save the iterations
_reg_weight	vector $N \times 1$, observation weights (default = 0)
__output	scalar 1 (default), print the results scalar 0, do not print the results
__title	string, name of the model
_reg_df	scalar, degrees of freedom

Remarks

Source

statistics.src

regFactorModel

■ **Purpose**
Estimates the factor model.

■ **Format**
{A,D} = regFactorModel(cov,n);

■ **Input**

cov	matrix $K \times K$, covariance matrix
n	scalar, number of factors

■ **Output**

A	matrix $K \times n$, factor loadings
D	matrix $K \times K$, specific variances matrix

■ **Globals**

■ **Remarks**
The estimate of Σ is $V = AA + D$. This procedure is adapted from the Gauss code developed by G. Arminger, U. Kuesters and R. Schlittgen.

■ **Source**
statistics.src

regFLS

■ **Purpose**
Flexible least squares.

■ **Format**
{beta,u,r2M,r2D} = regFLS(y,x,W);

■ **Input**

y	matrix $N \times 1$, dependent variable
x	matrix $N \times K$, independent variables
W	scalar or vector $K \times 1$ or matrix $K \times K$, mu-weight parameters

■ **Output**

beta	matrix $N \times K$, FLS estimates
u	vector $N \times 1$, residuals
r2M	scalar, sum of squared residual measurement errors
r2D	scalar, sum of squared residual dynamic errors

■ **Globals**

__output	scalar 1 (default), print the statistics
	scalar 0, do not print the statistics

■ **Remarks**

■ **Source**
numerics.src

regGMM

Purpose

Estimates the parameters by the generalized method of moments.

Format

{theta,stderr,cov,Qmin} = regGMM(dataset,vars,&f,sv,RR,r);

Input

dataset	string, name of the dataset
	scalar 0, do not use Gauss datasets
vars	vector $N \times 1$, names of the variables
	– or –
	matrix $N \times M$, data matrix
	– or –
	0 if the data are already defined in the GMM function
&f	scalar, pointer to a procedure which computes the moments
sv	vector $G \times 1$, starting values for the optimization algorithm
RR	matrix $K \times G$, matrix R of the implied restrictions $\theta = R\gamma + r$
r	vector $G \times 1$, vector r of the implied restrictions $\theta = R\gamma + r$

Output

theta	vector $K \times 1$, estimated parameters $\hat{\theta}$
stderr	vector $K \times 1$, standard errors of the estimated parameters
cov	matrix $K \times K$, covariance matrix of the estimated parameters
Qmin	scalar, value of the Q function

■ **Globals**

_ParNames	vector $K \times 1$, name of the parameters (default = 0)
_reg_cov	scalar 0, do not estimate the covariance matrix
	scalar 1, OPG estimator
_regGMM_invW	matrix $M \times M$, weights matrix
	– or –
	0 to use the Newey-West estimator (default = 0)
_regGMM_iters	scalar, maximum number of iterations (default = 20)
_regGMM_lags	scalar, number of lags for the Bartlett window (default = 0)
_regGMM_tol	scalar, convergence tolerance (default = 0.001)
_regGMM_pinv	scalar, 1 to use the Moore-Penrose inverse (default = 0)
_regGMM_mtd	scalar, method to define the matrix W of weights
	1 for a full matrix (default value)
	2 for a diagonal matrix
_reg_NoIter	scalar, 1 to suppress the optimisation step
_reg_opalgr	scalar, numerical algorithm (default = 2)
	1 for the SD algorithm
	2 for the BFGS algorithm
	3 for the scaled BFGS algorithm
	4 for the Self-Scaling DFP algorithm
	5 for the Newton-Raphson algorithm
	6 for the Polak-Ribiere Conjugate Gradient algorithm
_reg_PrintIters	scalar 1 (default), print the iterations
	scalar 0, do not print the iterations
_reg_Output	string, name of the file to save the iterations
__output	scalar 1 (default), print the results
	scalar 0, do not print the results
__title	string, name of the model
_reg_df	scalar, degrees of freedom
_regGMM_Jtest	vector 2×1, Hansen J statistic and p-value

■ **Remarks**

■ **Source**

statistics.src

regHuber

Purpose

Estimates the parameters by the Huber robust method.

Format

{beta,u,retcode} = regHuber(y,x,c);

Input

y	matrix $N \times 1$, dependent variable
x	matrix $N \times K$, independent variables
c	scalar, value of the threshold

Output

beta	vector $K \times K$, estimated values
u	vector $N \times 1$, residuals
retcode	scalar, return code

Globals

__eps	scalar, convergence tolerance
_regRobust_maxit	scalar, maximum number of iterations

Remarks

Source

numerics.src

regKernel

■ **Purpose**
Estimates the parameters by the non-parametric Kernel method.

■ **Format**
yhat = regHuber(y,x,p,&kernel,h,z);

■ **Input**

y	vector $N \times 1$, dependent variable
x	vector $N \times 1$, independent variable
p	scalar, polynomial order
&kernel	scalar, pointer to a function that compute the Kernel function 0 to use the Gaussian Kernel function
h	scalar, window (default = 0)
z	vector $M \times 1$, points to be evaluated

■ **Output**

yhat	vector $M \times 1$, estimated values

■ **Globals**

■ **Remarks**

■ **Source**
numerics.src

regKernelDensity

Purpose

Estimates the probability density function by the Gaussian Kernel method.

Format

{pdf,cdf} = regKernelDensity(y,x);

Input

y	vector $N \times 1$, data
x	vector $M \times 1$, points to be evaluated

Output

pdf	vector $M \times 1$, estimated values of the pdf
cdf	vector $M \times 1$, estimated values of the cdf

Globals

Remarks

Source

numerics.src

regKernelQuantile

■ **Purpose**
Non-parametric quantile regression.

■ **Format**
q = regKernelQuantile(y,x,alpha,d,z);

■ **Input**

y	matrix $N \times 1$, dependent variable
x	matrix $N \times 1$, independent variable
alpha	scalar, quantile level α
d	scalar 1 for linear polynomial regression scalar 2 for quadratic polynomial regression
z	matrix $M \times 1$, points to be evaluated

■ **Output**

q	matrix $M \times 1$, estimated values

■ **Globals**

■ **Remarks**

■ **Source**
optimization.src

regLAD

Purpose

Estimates the parameters by the robust method of least absolute deviations.

Format

{beta,u,retcode} = regLAD(y,x);

Input

y	matrix $N \times 1$, dependent variable
x	matrix $N \times K$, independent variables

Output

beta	vector $K \times 1$, estimated values
u	vector $N \times 1$, residuals
retcode	scalar, return code

Globals

__eps	scalar, convergence tolerance
_regRobust_maxit	scalar, maximum number of iterations

Remarks

Source

numerics.src

regLDP

■ **Purpose**
Linear dependency analysis.

■ **Format**
{dep,indep,A,tols} = regLDP(M,tol);

■ **Input**

M	matrix $N \times K$ or matrix $K \times K$
tol	scalar, tolerance value

■ **Output**

dep	vector $K_1 \times 1$, dependent variables y
indep	vector $K_2 \times 1$, independent vacraibles x
A	matrix $K_1 \times K_2$, linear dependency matrix $y = Ax$
tols	vector $K_1 \times 1$, tolerances

■ **Globals**

__output	scalar 1 (default), print the statistics
	scalar 0, do not print the statistics

■ **Remarks**
The procedure is adapted from the original code written by Ron Schoenberg.

■ **Source**
numerics.src

regLogit

Purpose

Estimates the Logit model by ML.

Format

{theta,stderr,cov,logl} = regLogit(dataset,y,x,&f,sv,RR,r);

Input

dataset	string, name of the dataset
	– or – scalar 0, do not use Gauss datasets
y	string, name of the y variable
	– or – vector $N \times 1$, dependent variable
x	vector $K \times 1$, name of the x variables
	– or – matrix $N \times K$, independent variables
sv	vector $G \times 1$, starting values for the optimization algorithm
RR	matrix $K \times G$, matrix R of the implied restrictions $\theta = R\gamma + r$
r	vector $G \times 1$, vector r of the implied restrictions $\theta = R\gamma + r$

Output

theta	vector $K \times 1$, estimated parameters $\hat{\theta}$
stderr	vector $K \times 1$, standard errors of the estimated parameters
cov	matrix $K \times K$, covariance matrix of the estimated parameters
logl	scalar, value of the Log-likelihood function

Globals

_ParNames	vector $K \times 1$, name of the parameters (default = 0)
_reg_cov	scalar 0, do not estimate the covariance matrix
	scalar 1, Hessian estimator (default value)
	scalar 2, OPG estimator
	scalar 3, White estimator
_reg_NoIter	scalar, 1 to suppress the optimisation step
_reg_opalgr	scalar, numerical algorithm (default = 2)
	1 for the SD algorithm
	2 for the BFGS algorithm
	3 for the scaled BFGS algorithm
	4 for the Self-Scaling DFP algorithm
	5 for the Newton-Raphson algorithm
	6 for the Polak-Ribiere Conjugate Gradient algorithm
_reg_PrintIters	scalar 1 (default), print the iterations
	scalar 0, do not print the iterations
_reg_Output	string, name of the file to save the iterations
_reg_weight	vector $N \times 1$, observation weights (default = 0)
__output	scalar 1 (default), print the results
	scalar 0, do not print the results
__title	string, name of the model
_reg_df	scalar, degrees of freedom

Source

statistics.src

regMars

Purpose

Estimates the MARS model.

Format

{fm,im} = mars_estimate(x,y,w,nB,nI,Lx);

Input

y	vector $N \times 1$, dependent variable
x	matrix $N \times P$, independent variables
w	vector $N \times 1$, observation weights – or – scalar 0 for uniform weights
nB	scalar, maximum number of basis functions
nI	scalar, minimum number of variables by basis function
Lx	vector $P \times 1$, variable status 0 if the variable is not used in the model 1 if the variable is numerical (no restriction) 2 if the variable is numerical and can be only used in an additive manner, withour interaction with other variables 3 if the variable is numerical and can be only used in a linear manner -1 if the variable is qualitative (no restriction) -2 if the variable is qualitative and can be only used in an additive manner, withour interaction with other variables (if Lx = 0, then Lx = ones(K,1);)

Output

fm	vector $im\,[1] \times 1$, MARS model output
im	vector $im\,[2] \times 1$, MARS model output

Globals

_mars_speed	scalar, speed factor (1-5) (default = 4)
_mars_logit	scalar 0 for the OLS regression (default) scalar 1 for the Logit regression
_mars_df	scalar, degrees of freedom (default = 3.0)
_mars_estimate_df	control parameter for sample reuse technique (default = 0)
_mars_seed	seed for internal random number generator (default = 0)
_mars_penalty	incremental penalty for increasing the number of variables (default = 0)
_mars_categorical	qualitative variable status 0 if no effect (drfault) 1 if no interaction between numerical and qualitative variables 2 if the interaction between numerical and qualitative variables is limited to two qualitative variables
_mars_minimum_span	scalar, minimum number of observations between two nodes (default = 0)

Remarks

Here are the mapping between Gauss variables and Fortran variables of the Friedman program:

```
n,p,x,y,w,lx,fm,im = same variables than the original Fortran code

nB = nk
     maximum number of basis functions (Ref[2] Sec. 3.6, Ref[3] Sec. 2.3)

nI = mi
     maximum number of variables per basis function (interaction level).
           mi = 1 => additive modeling (main effects only);
           mi > 1 => up to mi-variable interactions allowed.

Lx = lx
     predictor variable flags; lx(i) corresponds to the ith variable:
           lx(i) =  0 : exclude variable from model.
                    1 : ordinal variable - no restriction.
                    2 : ordinal variable that can only enter additively;
                        no interactions with other variables.
                    3 : ordinal variable that can enter only linearly.
                   -1 : categorical variable - no restriction.
                   -2 : categorical variable that can only enter additively;
                        no interactions with other variables.

fm(3+nk*(5*mi+nmcv+6)+2*p+ntcv),im(21+nk*(3*mi+8)) = mars model.
     (nmcv = maximum number of distinct values for any categorical variable;
      ntcv = total number of distinct values over all categorical variables.)

note: upon return im(1) and im(2) contain the lengths of the fm and
      im arrays (respectively) actually used by the program.

_mars_speed = is
              call speed(is):
              is = speed acceleration factor (1-5).
              larger values progressively sacrifice optimization thuroughness for
              computational speed advantage. this usually results in marked decrease
              in computing time with little or no effect on resulting approximation
              accuracy (especially useful for exploratory work).
              is = 1 => no acceleration.
              is = 5 => maximum speed advantage.
              (default: is=4) (Ref [4] Secs. 3.0 - 4.0)

_mars_logit = il
              call logit(il):
              il = ordinary/logistic regression flag.
              il = 0 => ordinary least-squares regression.
              il = 1 => logistic regression.
              (default: il=0). if logistic regression is selected (il=1) then
              the response variable is assumed to take on only the values 0/1 and
              the mars model is for the log-odds: f(x) = log (Pr(Y=1:x)/Pr(Y=0:x)).
              (Ref[2] Sec. 4.5)

_mars_df = df
```

```
                call setdf(df):
                df = number of degrees-of-freedom charged for (unrestricted)
                knot optimization. (default: df=3.0)
                (Ref[2] Sec. 3.6, Ref[3] Sec. 2.3)

_mars_estimate_df = ix
                call xvalid(ix):
                ix = control parameter for sample reuse technique used to automatically
                estimate smoothing parameter df (see above) from the data.
                ix = 0 => no effect (default). value used for df is set by user if
                setdf(df) is called (see above), otherwise default value
                (df=3.0) is used.
                ix > 0 => ix - fold cross-validation.
                ix < 0 => single validation pass using every (-ix)th (randomly selected)
                observation as an independent test set.
                if ix.ne.0 then call setdf(df) (see above) has no effect. if ix > 0,
                computation increases roughly by a factor of ix over that for ix = 0.
                for ix < 0 computation increases approximately by a factor of two.
                (Ref[3] Sec. 2.3)

_mars_seed = is
                call stseed(is):
                is = seed for internal random number generator used to group observation
                subsets for validation (ix.ne.0). (default: is=987654321).

_mars_penalty = fv
                call setfv(fv):
                fv = (fractional) incremental penalty for increasing the number of variables
                in the mars model. sometimes useful with highly collinear designs as it
                may produce nearly equivalent models with fewer predictor variables,
                aiding in interpretation. (fv .ge. 0)
                fv = 0.0  => no penalty (default).
                fv = 0.05 => moderate penalty.
                fv = 0.1  => heavy penality.
                the best value depends on the specific situation and some user
                experimentation using different values is usually required. this option
                should be used with some care. (Ref[2] Sec. 5.3)

_mars_categorical = ic
                call setic(ic):
                ic = flag restricting categorical - ordinal interactions.
                ic = 0 => no effect (default).
                ic = 1 => interactions between categorical and ordinal variables prohibited.
                ic = 2 => maximum number of ordinal variables participating in any
                interaction is restricted to two. categorical interactions are
                unrestricted.
                the restrictions associated with a value of ic are imposed in addition
                to those that are controlled by the mi and lx flags (see above).

_mars_minimum_span = ms
                call setms(ms):
                ms = minimum span (minimum number of observations between each knot).
                ms .le. 0 => default value (depending on n and p) is used.
                (default: ms=0). (Ref[2] Sec. 3.8)
```

References

1. Friedman, J. H. (1988). Fitting functions to noisy data in high dimensions. proc., Twentyth Symposium on the Interface, Wegman, Gantz, and Miller, eds. American Statistical Association, Alexandria, VA. 3-43.
2. Friedman, J. H. (1991a). Multivariate adaptive regression splines (with discussion). Annals of Statistics, 19, 1-141 (March).
3. Friedman, J. H. (1991b). Estimating functions of mixed ordinal and categorical variables using adaptive splines. Department of Statistics, Stanford University, Tech. Report LCS108.
4. Friedman, J. H. (1993). Fast MARS. Department of Statistics, Stanford University, Tech. Report LCS110.
5. Friedman, J. H. and Silverman, B. W. (1989). Flexible parsimonious smoothing and additive modeling (with discussion). TECHNOMETRICS, 31, 3-39 (Feburary).

Source

statistics.src

Important remark

To use this procedure, you need the DLL file mars36.dll built from the Fortran code of Jerome Friedman. It seems that there is a copyright problem with the Fortran code which was freely available some years ago. For this reason, I have decided to delete the DLL file until I get more information about the license of the Fortran code.

regMarsForecast

■ **Purpose**
Forecasting with Mars model.

■ **Format**
y = regMarsForecast(x,fm,im,m);

■ **Input**

x	matrix $N \times P$
fm	vector $im\,[1] \times 1$, MARS model output
im	vector $im\,[2] \times 1$, MARS model output
m	scalar, forecasting technique 1 - no smoothing (piecewise linear model) 2 - smoothing (piecewise cubic model)

■ **Output**

Y	vector $N \times 1$, model responses

■ **Globals**

■ **Remarks**

■ **Source**
statistics.src

regML

Purpose

Estimates the parameters by the method of maximum likelihood.

Format

{theta,stderr,cov,logl} = regML(dataset,vars,&f,sv,RR,r);

Input

dataset	string, name of the dataset – or – scalar 0, do not use Gauss datasets
vars	vector $N \times 1$, name of the variables – or – matrix $N \times M$, data matrix – or – scalar 0 if the data are already defined in the ML function
&f	scalar, pointer to a procedure which computes the log-likelihood vector
sv	vector $G \times 1$, starting values for the optimization algorithm
RR	matrix $K \times G$, matrix R of the implied restrictions $\theta = R\gamma + r$
r	vector $G \times 1$, vector r of the implied restrictions $\theta = R\gamma + r$

Output

theta	vector $K \times 1$, estimated parameters $\hat{\theta}$
stderr	vector $K \times 1$, standard errors of the estimated parameters
cov	matrix $K \times K$, covariance matrix of the estimated parameters
logl	scalar, value of the Log-likelihood function

Globals

_ParNames	vector $K \times 1$, name of the parameters (default = 0)
_reg_cov	scalar 0, do not estimate the covariance matrix scalar 1, Hessian estimator (default value) scalar 2, OPG estimator scalar 3, White estimator
_reg_NoIter	scalar, 1 to suppress the optimisation step
_reg_opalgr	scalar, numerical algorithm (default = 2) 1 for the SD algorithm 2 for the BFGS algorithm 3 for the scaled BFGS algorithm 4 for the Self-Scaling DFP algorithm 5 for the Newton-Raphson algorithm 6 for the Polak-Ribiere Conjugate Gradient algorithm
_reg_PrintIters	scalar 1 (default), print the iterations scalar 0, do not print the iterations
_reg_Output	string, name of the file to save the iterations
_reg_weight	vector $N \times 1$, observation weights (default = 0)
__output	scalar 1 (default), print the results scalar 0, do not print the results
__title	string, name of the model
_reg_df	scalar, degrees of freedom

Source

statistics.src

regNLS

Purpose

Estimates the parameters by the method of non-linear least squares.

Format

{theta,stderr,cov,u} = regNLS(y,&f,sv,RR,r);

Input

y	vector $N \times 1$, dependent variable
&f	scalar, pointer to a procedure which computes the function $f(\theta)$
sv	vector $G \times 1$, starting values for the optimization algorithm
RR	matrix $K \times G$, matrix R of the implied restrictions $\theta = R\gamma + r$
r	vector $G \times 1$, vector r of the implied restrictions $\theta = R\gamma + r$

Output

theta	vector $K \times 1$, estimated parameters $\hat{\theta}$
stderr	vector $K \times 1$, standard errors of the estimated parameters
cov	matrix $K \times K$, covariance matrix of the estimated parameters
u	vector $N \times 1$, residuals

Globals

_ParNames	vector $K \times 1$, name of the parameters (default = 0)
_reg_gradp	scalar, pointer to a procedure which computes the Jacobian matrix of $f(\theta)$ – or – 0 to use the numerical gradient
_reg_NoIter	scalar, 1 to suppress the optimisation step
_reg_opalgr	scalar,, numerical algorithm (default = 2) 1 for the SD algorithm 2 for the BFGS algorithm 3 for the scaled BFGS algorithm 4 for the Self-Scaling DFP algorithm 5 for the Newton-Raphson algorithm 6 for the Polak-Ribiere Conjugate Gradient algorithm
_reg_PrintIters	scalar 1 (default), print the iterations scalar 0, do not print the iterations
_reg_Output	string, name of the file to save the iterations
_reg_weight	vector $N \times 1$, observation weights (default = 0)
__output	scalar 1 (default), print the results scalar 0, do not print the results
__title	string, name of the model
_reg_df	scalar, degrees of freedom

Remarks

Source

statistics.src

regOLS

Purpose

Estimates the parameters by the method of ordinary least squares.

Format

{beta,stderr,cov,u} = regOLS(y,x,RR,r);

Input

y	matrix $N \times 1$, dependent variable
x	matrix $N \times K$, independent variables
RR	matrix $K \times G$, matrix R of the implied restrictions $\theta = R\gamma + r$
r	vector $G \times 1$, vector r of the implied restrictions $\theta = R\gamma + r$

Output

beta	vector $K \times 1$, estimated parameters
stderr	vector $K \times 1$, standard errors of the estimated parameters
cov	matrix $K \times K$, covariance matrix of the estimated parameters
u	vector $N \times 1$, residuals

Globals

_ParNames	vector $K \times 1$, name of the parameters (default = 0)
_reg_PrintIters	scalar 1 (default), print the iterations
	scalar 0, do not print the iterations
_reg_weight	vector $N \times 1$, observation weights (default = 0)
__output	scalar 1 (default), print the results
	scalar 0, do not print the results
__title	string, name of the model
_reg_df	scalar, degrees of freedom

Remarks

Source

statistics.src

regOU

Purpose

Estimates the parameters of the Ornstein-Uhlenbeck process by ML.

Format

{theta,stderr,cov,logl} = regOU(x,k,sv,RR,r);

Input

x	vector $N \times 1$, data
k	scalar, time step dt
sv	vector $G \times 1$, starting values for the optimization algorithm
RR	matrix $3 \times G$, matrix R of the implied restrictions $\theta = R\gamma + r$
r	vector $G \times 1$, vector r of the implied restrictions $\theta = R\gamma + r$

Output

theta	vector 3×1, estimated parameters $\hat{\theta}$
stderr	vector 3×1, standard errors of the estimated parameters
cov	matrix $3 \times K$, covariance matrix of the estimated parameters
logl	scalar, value of the Log-likelihood function

Globals

_ParNames	vector $K \times 1$, name of the parameters (default = 0)
_reg_cov	scalar covariance matrix estimator (default = 1)
	scalar 0, do not estimate the covariance matrix
	scalar 1, Hessian estimator
	scalar 2, OPG estimator
	scalar 3, White estimator
_reg_NoIter	scalar, 1 to suppress the optimisation step
_reg_opalgr	scalar, numerical algorithm (default = 2)
	1 for the SD algorithm
	2 for the BFGS algorithm
	3 for the scaled BFGS algorithm
	4 for the Self-Scaling DFP algorithm (
	5 for the Newton-Raphson algorithm
	6 for the Polak-Ribiere Conjugate Gradient algorithm
_reg_PrintIters	scalar 1 (default), print the iterations
	scalar 0, do not print the iterations
_reg_Output	string, name of the file to save the iterations
__output	scalar 1 (default), print the results
	scalar 0, do not print the results
__title	string, name of the model
_reg_df	scalar, degrees of freedom

Remarks

Source

statistics.src

regPCA

Purpose

Principal component analysis.

Format

{va,ve,pcaMatrix} = regPCA(x,M);

Input

x	matrix $N \times K$, data
	– or –
	matrix $K \times K$, covariance or correlation matrix
M	scalar, number of factors

Output

va	vector $M \times 1$, eigenvalues
va	vector $M \times K$, eigenvectors
pcaMatrix	data buffer

Globals

__output	scalar 1 (default), print the statistics
	scalar 0, do not print the statistics

Remarks

You may use the `vread` or `vget` command to extract the following statistics: "QLT" (representation quality of each factor), "QLTc" (cumulated representation quality of each factor), "SAT" (saturation table), "QLTvar" (representation quality of each variable), "CTRvar" (contribution of each variable) and "corr" (correlation matrix).

Source

numerics.src

regProbit

Purpose

Estimates the Probit model by ML.

Format

{theta,stderr,cov,logl} = regProbit(dataset,y,x,&f,sv,RR,r);

Input

dataset	string, name of the dataset
	– or – scalar 0, do not use Gauss datasets
y	string, name of the y variable
	– or – vector $N \times 1$, dependent variable
x	vector $K \times 1$, name of the x variables
	– or – matrix $N \times K$, independent variables
sv	vector $G \times 1$, starting values for the optimization algorithm
RR	matrix $K \times G$, matrix R of the implied restrictions $\theta = R\gamma + r$
r	vector $G \times 1$, vector r of the implied restrictions $\theta = R\gamma + r$

Output

theta	vector $K \times 1$, estimated parameters $\hat{\theta}$
stderr	vector $K \times 1$, standard errors of the estimated parameters
cov	matrix $K \times K$, covariance matrix of the estimated parameters
logl	scalar, value of the Log-likelihood function

Globals

_ParNames	vector $K \times 1$, name of the parameters (default = 0)
_reg_cov	scalar 0, do not estimate the covariance matrix
	scalar 1, Hessian estimator (default value)
	scalar 2, OPG estimator
	scalar 3, White estimator
_reg_NoIter	scalar, 1 to suppress the optimisation step
_reg_opalgr	scalar, numerical algorithm (default = 2)
	1 for the SD algorithm
	2 for the BFGS algorithm
	3 for the scaled BFGS algorithm
	4 for the Self-Scaling DFP algorithm
	5 for the Newton-Raphson algorithm
	6 for the Polak-Ribiere Conjugate Gradient algorithm
_reg_PrintIters	scalar 1 (default), print the iterations
	scalar 0, do not print the iterations
_reg_Output	string, name of the file to save the iterations
_reg_weight	vector $N \times 1$, observation weights (default = 0)
__output	scalar 1 (default), print the results
	scalar 0, do not print the results
__title	string, name of the model
_reg_df	scalar, degrees of freedom

Source

statistics.src

regQP

Purpose

Linear regression using quadratic programming.

Format

{beta,u,R2} = regQP(y,x,w,sv,A,B,C,D,Bounds);

Input

y	matrix $N \times 1$, dependent variable
x	matrix $N \times K$, independent variables
w	matrix $N \times 1$, weights (0 if no weight)
sv	matrix $K \times 1$, starting values
A	matrix $M \times K$, the A matrix
B	matrix $M \times 1$, the B vector
C	matrix $L \times K$, the C matrix
D	matrix $L \times 1$, the D vector
Bounds	matrix $K \times 2$, lower and upper bounds

Output

beta	matrix $K \times 1$, estimated coefficients
u	matrix $N \times 1$, residuals
R2	matrix 2×1, statistics R^2 and $\bar{R}^2$

Globals

Remarks

We solve the linear regression model:

$$\begin{aligned} \hat{\beta} &= \arg\min (Y - X\beta)^\top (Y - X\beta) \\ &\text{u.c.} \quad A^\top \beta = B,\ C^\top \beta \geq D \text{ and } \beta_- \leq \beta \leq \beta_+ \end{aligned}$$

Source

optimization.src

regQR

■ **Purpose**
Estimates the parameters by the robust quantile regression method.

■ **Format**
{beta,u,retcode} = regQR(y,x,alpha);

■ **Input**

y	matrix $N \times 1$, dependent variable
x	matrix $N \times K$, independent variables
alpha	scalar, value of the quantile α

■ **Output**

beta	vector $K \times 1$, estimated coefficients
u	vector $N \times 1$, residuals
retcode	scalar, return code

■ **Globals**

__eps	scalar, convergence tolerance
_regRobust_maxit	scalar, maximum number of iterations

■ **Remarks**

■ **Source**
numerics.src

regQuantile

Purpose

Linear quantile regression.

Format

{beta,u_plus,u_minus} = regQuantile(y,x,alpha,w);

Input

y	matrix $N \times 1$, dependent variable
x	matrix $N \times K$, independent variables
alpha	scalar, quantile level α
w	matrix $N \times 1$, weights

Output

beta	matrix $K \times 1$, estimated coefficients
u_plus	matrix $N \times 1$, positive value of residuals
u_minus	matrix $N \times 1$, negative value of residuals

Globals

Remarks

Source

optimization.src

regRestrict

Purpose

Fixes parameters in regression models.

Format

{RR,r,gamma_} = regRestrict(theta,indx);

Input

theta	vector $K \times 1$, vector θ
indx	vector $G \times 1$, list of fixed parameters

Output

RR	matrix $K \times G$, matrix R of the implied restrictions $\theta = R\gamma + r$
r	vector $G \times 1$, vector r of the implied restrictions $\theta = R\gamma + r$
gamma_	vector $G \times 1$, vector γ

Globals

Remarks

Source

numerics.src

regRobust

Purpose

Estimates the parameters by the general robust method (M estimation).

Format

{beta,u,retcode} = regRobust(y,x,&rho,&psi);

Input

y	matrix $N \times 1$, dependent variable
x	matrix $N \times K$, independent variables
&rho	scalar, pointer to a procedure which computes the function $\rho(u)$
&psi	scalar, pointer to a procedure which computes the function $\psi(u)$

Output

beta	vector $K \times 1$, estimated values
u	vector $N \times 1$, residuals
retcode	scalar, return code

Globals

__eps	scalar, convergence tolerance
_regRobust_maxit	scalar, maximum number of iterations

Remarks

Source

numerics.src

regSharpeStyle

■ **Purpose**
Sharpe style analysis.

■ **Format**
{w,u,R2} = regSharpeStyle(fund,factors);

■ **Input**

fund	matrix $N \times 1$, returns of the fund
factors	matrix $N \times K$, returns of factors

■ **Output**

w	matrix $K \times 1$, factors weights
u	matrix $N \times 1$, residuals
R2	scalar, the statistic R^2

■ **Globals**

■ **Remarks**
Let $R(t)$ and $R_i(t)$ be respectively the return of the fund and the return of the factor i at time t. The Sharpe style regression is:

$$\begin{aligned} R(t) &= \sum_{i=1}^{n} w_i R_i(t) + \varepsilon_t \\ \text{u.c.} \quad & w_i \geq 0 \text{ et } \sum_{i=1}^{n} w_i = 1 \end{aligned}$$

■ **Source**
optimization.src

regSpline

■ **Purpose**
Spline interpolation and smoothing.

■ **Format**
s = regSpline(x,y,p,z);

■ **Input**

x	matrix $N \times 1$, independent variable
y	matrix $N \times 1$, dependent variable
p	scalar, smoothing parameter
z	matrix $M \times 1$, points to be evaluated

■ **Output**

s	matrix $M \times 1$, estimated values

■ **Globals**

■ **Remarks**
The procedure is adapted from the original code written by David Baird.

■ **Source**
numerics.src

regTobit

Purpose

Estimates the Tobit model by ML.

Format

{theta,stderr,cov,logl} = regTobit(dataset,y,x,&f,sv,RR,r);

Input

dataset	string, name of the dataset
	– or – scalar 0, do not use Gauss datasets
y	string, name of the y variable
	– or – vector $N \times 1$, dependent variable
x	vector $K \times 1$, name of the x variables
	– or – matrix $N \times K$, independent variables
sv	vector $G \times 1$, starting values for the optimization algorithm
RR	matrix $K \times G$, matrix R of the implied restrictions $\theta = R\gamma + r$
r	vector $G \times 1$, vector r of the implied restrictions $\theta = R\gamma + r$

Output

theta	vector $K \times 1$, estimated parameters $\hat{\theta}$
stderr	vector $K \times 1$, standard errors of the estimated parameters
cov	matrix $K \times K$, covariance matrix of the estimated parameters
logl	scalar, value of the Log-likelihood function

Globals

_ParNames	vector $K \times 1$, name of the parameters (default = 0)
_reg_cov	scalar 0, do not estimate the covariance matrix
	scalar 1, Hessian estimator (default value)
	scalar 2, OPG estimator
	scalar 3, White estimator
_reg_NoIter	scalar, 1 to suppress the optimisation step
_reg_opalgr	scalar, numerical algorithm (default = 2)
	1 for the SD algorithm
	2 for the BFGS algorithm
	3 for the scaled BFGS algorithm
	4 for the Self-Scaling DFP algorithm
	5 for the Newton-Raphson algorithm
	6 for the Polak-Ribiere Conjugate Gradient algorithm
_reg_PrintIters	scalar 1 (default), print the iterations
	scalar 0, do not print the iterations
_reg_Output	string, name of the file to save the iterations
_reg_weight	vector $N \times 1$, observation weights (default = 0)
__output	scalar 1 (default), print the results
	scalar 0, do not print the results
__title	string, name of the model
_reg_df	scalar, degrees of freedom

Source

statistics.src

Risk_Budgeting_Solve

Purpose

Solves the risk budegting allocation problem.

Format

{x,vol,mr,tr,rc} = Risk_Budgeting_Solve(sv,er,cov,A,B,C,D,Bounds,ac,rb,vol_target);

Input

sv	matrix $N \times 1$, starting values
er	matrix $N \times 1$, the vector μ of expected returns
cov	matrix $N \times N$, the covariance matrix Σ
A	matrix $M \times N$, the A matrix
B	matrix $M \times 1$, the B vector
C	matrix $L \times N$, the C matrix
D	matrix $L \times 1$, the D vector
Bounds	matrix $N \times 2$, lower and upper bounds
ac	matrix $N \times C$, specification of asset classes
rb	matrix $C \times 1$, rhe risk budget of asset classes
vol_targes	scalar, the volatility target

Output

x	matrix $N \times 1$, the optimal portfolio
vol	scalar, the volatility of the optimal portfolio
mr	matrix $N \times 1$, marginal risk contributions
tr	matrix $N \times 1$, total risk contributions
rc	matrix $N \times 1$, risk contributions (in %)

Globals

Remarks

The optimization problem is:

$$x^\star = \arg\max \mu(x) \quad \text{u.c.} \quad \begin{cases} \sigma(x) \leq \sigma^\star \\ \sum_{i \in \mathcal{C}_j} \sigma_i(x) \leq c_j \sigma^\star \\ \mathbf{1}^\top x = 1 \\ \mathbf{0} \leq x \leq \mathbf{1} \end{cases}$$

c_j is the relative risk budget for the asset class $\mathcal{C}_j$.

Source

optimization.src

rndmn

■ **Purpose**
Simulates normal random vectors using the Cholesky decomposition.

■ **Format**
u = rndmn(mu,cov,N);

■ **Input**

mu	vector $K \times 1$, vector μ of expected returns
cov	matrix $K \times K$, covariance matrix Σ
N	scalar, number of simulations

■ **Output**

u	matrix $N \times K$, random vectors

■ **Globals**

■ **Remarks**

■ **Source**
numerics.src

rndmn_eig

Purpose

Simulates normal random vectors using the Eigenvalue decomposition.

Format

u = rndmn_eig(mu,cov,N);

Input

mu	vector $K \times 1$, vector μ of expected returns
cov	matrix $K \times K$, covariance matrix Σ
N	scalar, number of simulations

Output

u	matrix $N \times K$, random vectors

Globals

Remarks

Source

numerics.src

rndmn_svd

■ **Purpose**
Simulates normal random vectors using the SVD decomposition.

■ **Format**
u = rndmn_svd(mu,cov,N);

■ **Input**

mu	vector $K \times 1$, vector μ of expected returns
cov	matrix $K \times K$, covariance matrix Σ
N	scalar, number of simulations

■ **Output**

u	matrix $N \times K$, random vectors

■ **Globals**

■ **Remarks**

■ **Source**
numerics.src

rndn_Box_Muller

■ **Purpose**
Simulates normal random numbers using the Box-Muller algorithm.

■ **Format**
{n1,n2} = rndn_Box_Muller(u1,u2);

■ **Input**

u1	matrix $E \times E$, uniform random numbers
u2	matrix $E \times E$, uniform random numbers

■ **Output**

n1	matrix $E \times E$, normal random numbers
nu2	matrix $E \times E$, normal random numbers

■ **Globals**

■ **Remarks**

■ **Source**
numerics.src

rndu_Halton

■ **Purpose**
Simulates quasi random numbers using Halton sequences.

■ **Format**
u = rndu_Halton(n,d);

■ **Input**

n	scalar, number of simulations
d	scalar, dimension

■ **Output**

u	matrix $n \times d$

■ **Globals**

_rnd_primes	matrix $d \times 1$, prime numbers (if 0, we use the first d prime numbers)

■ **Remarks**

■ **Source**
numerics.src

rndu_Hammersley

Purpose
Simulates quasi random numbers using Hammersley sequences.

Format
u = rndu_Hammersley(n,d);

Input

n	scalar, number of simulations
d	scalar, dimension

Output

u	matrix $n \times d$

Globals

_rnd_primes	matrix $d \times 1$, prime numbers (if 0, we use the first d prime numbers)

Remarks

Source
numerics.src

simulate_Brownian_Bridge

■ **Purpose**
Simulates a Brownian Bridge.

■ **Format**
{t,B} = Simulate_Brownian_Bridge(t_x,x,t,ns);

■ **Input**

t_x	matrix $N \times 1$, time indices t_i
x	matrix $N \times 1$, values of x_i
t	matrix $T \times 1$, time indices t
ns	scalar, number of simulations N_s

■ **Output**

t	matrix $T \times 1$, time indices t
B	matrix $T \times N_s$, the path of the Brownian bridge

■ **Globals**

■ **Remarks**
We have $B(t_i) = x_i$.

■ **Source**
numerics.src

simulate_Correlation

Purpose

Simulates a random correlation matrix using the random orthogonal algorithm.

Format

corr = Simulate_Correlation(N,lambda,kappa);

Input

N	scalar, dimension of the correlation matrix
lambda	$N \times 1$ vector, specified singular values (0 for random uniform singular values) (1 for one large singular value) (2 for one small singular value) (3 for geometrically distributed singular values) (4 for arithmetically distributed singular values) (5 for exponential distributed singular values)
kappa	scalar, estimate of the condition number of the correlation matrix

Output

corr	$N \times N$ matrix

Remark

Source

numerics.src

simulate_Correlation2

■ **Purpose**
Simulates a random correlation matrix using the nearest correlation algorithm.

■ **Format**
corr = Simulate_Correlation2(N,lambda,e);

■ **Input**

N	scalar, dimension of the correlation matrix
lambda	$N \times 1$ vector, specified eigenvalues (0 for random uniform eigenvalues)
e	$N \times N$ matrix, signs of the correlation values 0 for random signs 1 for positive signs -1 for negative signs

■ **Output**

corr	$N \times N$ matrix

■ **Source**
corr.src

sinMatrix

■ **Purpose**
Computes the matrix sine.

■ **Format**
B = sinMatrix(A);

■ **Input**

A	matrix $N \times N$

■ **Output**

B	matrix $N \times N$

■ **Globals**

■ **Remarks**
We have:

$$B = -\frac{e^{iA} - e^{-iA}}{2} i$$

■ **Source**
numerics.src

sqrtMatrix

■ **Purpose**
Computes the matrix square root.

■ **Format**
B = sinMatrix(A);

■ **Input**

A	matrix $N \times N$

■ **Output**

B	matrix $N \times N$

■ **Globals**

■ **Remarks**
The matrix square root of A is the matrix B such that:

$$B \times B = A$$

■ **Source**
numerics.src

Chapter 4

Examples

Some programs require the following Gauss library:

- gWizard
- nnet
- option
- pde2d

All these libraries are available in the web page :

`http://www.thierry-roncalli.com/#gauss`

4.1 Examples in the *backtest* directory

1. **backtest1.prg**
   Performs a backtest example (Chapter 1 – Tables 1 to 5).

2. **ch1.prg**
   An example of currency hedging (Chapter 1 – Figure 3).

3. **fees0.prg**
   An example of adding managing and performance fees (Chapter 1 – Table 6).

4. **fees1.prg**
   Impact of managing and performance fees on the backtest (Chapter 1 – Figure 1).

5. **fees2.prg**
   Impact of managing and performance fees on risk, return and Sharpe ratio (Chapter 1 – Figure 2).

6. **leverage1.prg**
   Computes the backtest of a leverage strategy (Chapter 1 – Figure 3).

7. **reporting1.prg**
   Illustrates the volatility bias (Chapter 1 – Figure 4).

8. **reporting2.prg**
   Illustrates the volatility bias (Chapter 1).

9. **reporting3.prg**
   Computes the maximum drawdown and the loss function (Chapter 1 – Figures 5 and 6).

4.2 Examples in the *optimization* directory

1. **bl1.prg**
   Computes π and μ_{cond} (Chapter 2 – Example of the section 3.6.3).

2. **bl2.prg**
   Black-Litterman inverse problem (Chapter 2 – Figure 16).

3. **erc1.prg**
   An example of computing Equally-Risk Contribution portfolios.

4. **index1.prg**
   In this program, we consider the sampling of the S&P 500 stock index (Chapter 2 – Figure 12).

5. **index2.prg**
   In this program, we compare the efficiency of the sampling method on S&P 500 and CAC 40 indices (Chapter 2 – Figure 13).

6. **index3.prg**
   Solves an enhanced indexing problem (Chapter 2 – Figures 10 and 11).

7. **index4.prg**
   Solves an enhanced indexing problem (Chapter 2 – Table 8).

8. **index5.prg**
   Solves a 130/30 indexing problem.

9. **index6.prg**
   Solves another 130/30 indexing problem.

10. **index7.prg**
    Solves several 130/30 indexing problems (Chapter 2 – Table 9)

11. **ip1.prg**
    An exemple of inverse problem.

12. **ls1.prg**
    Calibrates Long/Short portfolios with a volatility target (Chapter 2 – Table 7).

13. **markowitz1.prg**
    Solves a Markowitz allocation problem (Chapter 2 – Table 3 and Figure 7).

14. **markowitz2.prg**
    Solves μ and σ-problems (Chapter 2 – Tables 4 and 5).

15. **minvar1.prg**
    Solves a minimum variance portfolio problem (Chapter 2 – Table 2).

16. **minvar2.prg**
    Illustrates the diversification effect (Chapter 2 – Figure 5).

17. **minvar3.prg**
    Backtest of a minimum variance strategy on a global asset classes basket (Chapter 2 – Figure 6).

18. **qprog1.prg**
    Interpretation of the Lagrange coefficients in the Qprog procedure (Chapter 2 – Figure 2).

19. **rq1.prg**
    Estimation of skew-beta returns by quantile regression (Chapter 2 – Figure 1).

20. **rb1.prg**
    Solves a risk-budgeting allocation problem (Chapter 2 – Tables 11 and 12).

21. **sharpe1.prg**
    Solves the Max Sharpe problem (Chapter 2 – Figures 8 and 9).

22. **sharpe2.prg**
    Solves the Max Sharpe problem (Chapter 2 – Table 6).

23. **style1.prg**
    Performs a Sharpe style regression (Chapter 2 – Figure 3).

24. **style2.prg**
    Performs a Sharpe style regression (Chapter 2 – Figure 4).

25. **tc1.prg**
    Impact of transaction costs (Chapter 2 – Section 2.7).

26. **tc2.prg**
    Solves a portfolio allocation problem with transaction costs (Chapter 2 – Table 10).

4.3 Examples in the *numerics* directory

1. **band1.prg**
   Storage comparison of band and dense matrices (Chapter 3 – Section 1.2).

2. **band2.prg**
   Speed comparison of band and dense matrices (Chapter 3 – Section 1.2)/

3. **cov1.prg**
   Comparison of eigenvalue and square root matrix decompositions (Chapter 3 – Section 2.3.1).

4. **cov2.prg**
   Computes the nearest correlation matrix (Chapter 3 – Section 2.3.2).

5. **cov3.prg**
   Simulates a correlation matrix (Chapter 3 – Section 3.3).

6. **cov4.prg**
   Computes the basket option price (Chapter 3 – Figure 22).

7. **eig1.prg**
   Eigenvalue decomposition of a covariance matrix (Chapter 3 – Section 1.1.1).

8. **eig2.prg**
   Comparison of eigenvalue and cholesky decompositions for simulating Gaussian random vectors (Chapter 3 – Section 1.1.1).

9. **eig3.prg**
   Eigenvalue decomposition for simulating Gaussian random vectors (Chapter 3 – Section 1.1.1).

10. **fls1.prg**
    Band matrices and flexible least squares (Chapter 3 – Figures 6 and 7).

11. **ldp1.prg**
    Linear dependency analysis (Chapter 3 – Section 1.1.3).

12. **mc1.prg**
    Computation of π by simulations (Chapter 3 – Figure 23).

13. **mc2.prg**
    Non-parametric density of MC estimators (Chapter 3 – Figure 24).

14. **mc3.prg**
    Computation of π by simulations (Chapter 3 – Section 3.4.1).

15. **mc4.prg**
    Convergence of MC estimators (Chapter 3 – Figure 25).

16. **mc5.prg**
    Antithetic simulation of GBM processes (Chapter 3 – Figure 26).

17. **mc6.prg**
    Non-parametric density of MC and MC+AV estimators (Chapter 3 – Figure 27).

18. **pca1.prg**
    Principal component analysis of the yield curve (Chapter 3 – Tables 1, 2 and 3).

19. **pca2.prg**
    Computes the first 3 factors of the yield curve (Chapter 3 – Figure 1).

20. **pde1.prg**
    Numerical errors when solving the Vasicek model by PDE (Chapter 3 – Figure 12).

21. **pde2.prg**
    Comparison of the densities of the Wiener process obtained by Feynman-Kac and Fokker-Planck algorithms (Chapter 3 – Section 2.5.3).

22. **pde3.prg**
    Comparison of the densities of the GBM process obtained by Feynman-Kac and Fokker-Planck algorithms (Chapter 3 – Figure 14).

23. **pde4.prg**
    Comparison of the densities of the OU process obtained by Feynman-Kac and Fokker-Planck algorithms (Chapter 3 – Figure 13).

24. **pde5.prg**
    Computation of the density in the Heston model by solving the 2D Fokker-Planck PDE (Chapter 3 – Section 2.5.3).

25. **pde6.prg**
    Computation of the density in the Heston model by solving the 2D Fokker-Planck PDE (Chapter 3 – Figure 15).

26. **pde7.prg**
    Computation of the density in the SABR model by solving the 2D Fokker-Planck PDE (Chapter 3 – Figure 16).

27. **qmc1.prg**
    Comparison of LCG, Hammersley, Halton and Faure random generators (Chapter 3 – Figure 28).

28. **qmc2.prg**
    Illustration of the Sobol random generator (Chapter 3 – Figure 29).

29. **qmc3.prg**
    Projection of the 3D Faure random generator (Chapter 3 – Section 3.4.3).

30. **qmc4.prg**
    Projection of several random generators on a sphere (Chapter 3 – Figure 30).

31. **qmc5.prg**
    Computation of the Spread option using QMC (Chapter 3 – Table 9).

32. **quad1.prg**
    Knots and weights of gauss quadratures (Chapter 3 – Figure 9).

33. **quad2.prg**
    Gauss-Legendre numerical integration (Chapter 3 – Figure 10).

34. **quad3.prg**
    Gauss-Legendre numerical integration of a sine function (Chapter 3 – Figure 11).

35. **quad4.prg**
    Computation of knots and weights using eigenvalue decomposition (Chapter 3 – Section 2.4.2).

36. **quad5.prg**
    Computation of the spread option price using 1D and 2D numerical quadratures (Chapter 3 – Table 7).

37. **rnd1.prg**
    Simulation of uniform random numbers (Chapter 3 – Section 3.1.1).

38. **rnd2.prg**
    Comparison of Matlab and Gauss random generators (Chapter 3 – Table 8).

39. **rnd3.prg**
    Lattice structure of LC generators (Chapter 3 – Figure 17).

40. **rnd4.prg**
    Comparison of exact and euler schemes for the GBM process (Chapter 3 – Figure 18).

41. **rnd5.prg**
    Density of MC estimators (Chapter 3 – Figure 19).

42. **rnd6.prg**
    Simulates a Brownian bridge (Chapter 3 – Figure 20).

43. **rnd7.prg**
    Simulates a constrained GBM process (Chapter 3 – Figure 21).

44. **schur1.prg**
    Schur decomposition (Chapter 3 – Section 1.1.2).

45. **schur2.prg**
    Computes the Markov generator of the transition probability matrix of fund ratings (Chapter 3 – Section 1.1.2).

46. **schur3.prg**
    Estimates the Markov generator using [IRW-1] and [IRW-2] algorithms (Chapter 3 – Section 1.1.2).

47. **schur4.prg**
    Computes transition probabilities (Chapter 3 – Figure 2).

48. **schur5.prg**
    Computes transition probabilities (Chapter 3 – Figure 3).

49. **schur6.prg**
    Computes transition probabilities (Chapter 3 – Figure 4).

50. **schur7.prg**
    Computes transition probability matrices (Chapter 3 – Section 1.1.2).

51. **schur8.prg**
    Computes the probability of persistence (Chapter 3 – Figure 5).

52. **simplex1.prg**
    Solves a portfolio allocation problem using discretization of the simplex set (Chapter 3 – Table 5).

53. **spline1.prg**
    Spline interpolation and smoothing of a GBM process (Chapter 3 – Figure 8).

4.4 Examples in the *pricing* directory

1. **bs1.prg**
   Call option pricing (Chapter 4 – Figure 1).

2. **bs2.prg**
   Computes the Delta coefficients (Chapter 4 – Figure 4).

3. **bs3.prg**
   Computes the Gamma coefficients (Chapter 4 – Figure 5).

4. **bs4.prg**
   Volatility smile (Chapter 4 – Figure 6).

5. **bs5.prg**
   Risk-neutral distribution and volatility smile (Chapter 4 – Figure 6).

6. **hedging1.prg**
   Dynamic delta hedging with a negative final PnL (Chapter 4 – Table 1).

7. **hedging2.prg**
   Dynamic delta hedging with a positive final PnL (Chapter 4 – Table 2).

8. **hedging3.prg**
   Density of the PnL ratio (Chapter 4 – Figure 2).

9. **hedging4.prg**
   Reproduces the hedging example presented in the book of John Hull (Chapter 4 – Section 2.2).

10. **hedging5.prg**
    Reproduces the hedging example presented in the book of John Hull (Chapter 4 – Section 2.2).

11. **hedging6.prg**
    Impact of the hedging frequency on the efficiency $\sigma(\pi)$ (Chapter 4 – Figure 3).

12. **mtm1.prg**
    Explains the Marked-to-Market pricing method (Chapter 4 – Tables 3 and 4).

4.5 Examples in the *statistics* directory

1. **ann1.prg**
   Sigmoid functions (Chapter 5 – Figure 7).

2. **ann2.prg**
   Graphic representation of the artifical neural network (Chapter 5 – Figure 8).

3. **ann3.prg**
   Structure of artifical neural networks (Chapter 5 – Figure 9).

4. **ann4.prg**
   Graphic representation of the dense ann (Chapter 5 – Figure 10).

5. **ann5.prg**
   Graphic representation of the contrained ann (Chapter 5 – Figure 11).

6. **ann6.prg**
   Partial R^2 and omission costs analysis (Chapter 5 – Figure 12).

7. **ann7.prg**
   Graphic representation of the optimal ann (Chapter 5 – Figure 13).

8. **ann8.prg**
   The XOR problem (Chapter 5 – Figure 14).

9. **ann9.prg**
   The T-C problem (Chapter 5 – Figures 15 and 16).

10. **ann11.prg**
    An example of classification (Chapter 5 – Figure 17).

11. **ann12.prg**
    A more complex example of classification (Chapter 5 – Figure 18).

12. **em1.prg**
    The EM algorithm (Chapter 5 – Section 1.2.5).

13. **mars1.prg**
    The Mars example of Friedman in *Annals of Statistics* (Chapter 5 – Section 1).

14. **ml-corr1.prg**
    Estimates the 1F correlation model (Chapter 5 – Figure 6).

15. **ml-corr2.prg**
    Compares the 1F and the multi-factor correlation models (Chapter 5 – Section 2.1.2).

16. **ml-corr3.prg**
    Estimate the factor model (Chapter 5 – Section 2.1.3).

17. **np1.prg**
    Computes a frequency histogram (Chapter 5 – Figure 2).

18. **np2.prg**
    Estimates the pdf using the Kernel method (Chapter 5 – Figure 3).

19. **np3.prg**
    Compares the pdf of the order statistics using Kernel and parametric models (Chapter 5 – Figure 4).

20. **np4.prg**
    Non-parametric Kernel regression (Chapter 5 – Figure 5).

21. **reg1.prg**
    Robust estimation of beta stocks (Chapter 5 – Figure 1).

Bibliography

[1] Roncalli, T [1995], Stratégies Quantitatives de Gestion, *Lecture Notes*, University of Evry, Available at `http://www.thierry-roncalli.com/download/qambook.pdf`.