GAUSS procedures for computing the nearest correlation matrix and simulating correlation matrices

G. Rapuch & T. Roncalli*
Groupe de Recherche Opérationnelle, Crédit Lyonnais, France

August 8, 2001

Abstract

In this short note, we consider two problems about correlation matrices. The first one is the computation of the nearest correlation matrix, whereas the second one concerns the simulation of random correlation matrices. We provide GAUSS procedures to perform these computations. We use the algorithms developped by N.J. Higham and give some financial applications.

1 Introduction

We consider two problems about correlation matrices:

1. Given a symmetric matrix, what is the nearest correlation matrix?
2. How to generate random correlation matrices?

The first problem is very important in risk management. For example, the analytical method to compute the value-at-risk of a linear portfolio requires the estimation of the covariance matrix of the factors. Because databases contain generally missing values, the estimation is performed using a *listwise* deletion. In some cases, one have to use a pairwise deletion method because of the number of missing values. But this method lead to a major problem because we do not obtain necessarily a **positive definite** matrix covariance matrix. The idea is then to replace the estimated correlation matrix with a symmetric positive (semi) definite matrix.

The second problem concerns the generation of random correlation matrices in high dimension. For example, one would like to test the pricing of a Basket option and to measure the impact of correlation effects. Simulating random correlation matrices could then be useful for multi-asset option pricing and basket credit derivatives.

2 Computing the nearest correlation matrix

2.1 The 'square root' method

Let $\hat{\Sigma}$ be an estimated covariance matrix. We assume that $\hat{\Sigma}$ is not a positive definite matrix. To obtain a positive semidefinite matrix, we adjust the covariance matrix. We consider the square root decomposition $\hat{\Sigma} = A^2$ with $A = A_1 + iA_2$ (A_1 and A_2 are two positive matrices). We then obtain a new estimate $\hat{\Sigma}$ of the covariance matrix with $\hat{\Sigma}^\star = A_1^2$.

Property 1 *$\hat{\Sigma}^\star = A_1^2$ is a positive semidefinite matrix.*

* *Corresponding author:* Groupe de Recherche Opérationnelle, Bercy-Expo — Immeuble Bercy Sud — 4è étage, 90 quai de Bercy — 75613 Paris Cedex 12 — France; *E-mail:* thierry.roncalli@creditlyonnais.fr

Proof. See HORN and JOHNSON [1991]. ■

Property 2 *Let $x \in R^N$. We have*

$$x^\top \hat{\Sigma}^\star x \geq x^\top \hat{\Sigma} x \tag{1}$$

Proof. It comes that

$$\begin{aligned} x^\top \hat{\Sigma} x &= x^\top A_1^2 x - x^\top A_2^2 x \\ &\leq x^\top A_1^2 x \end{aligned} \tag{2}$$

■

Remark 1 *Let $\mathrm{VaR}_\alpha(x;\Sigma)$ be the analytical Value-at-Risk of a linear portfolio x with covariance matrix Σ*

$$\mathrm{VaR}_\alpha(x;\Sigma) = \sqrt{x^\top \Sigma x} \cdot \Phi^{-1}(\alpha) \tag{3}$$

We have

$$\mathrm{VaR}_\alpha\left(x;\hat{\Sigma}^\star\right) \geq \mathrm{VaR}_\alpha\left(x;\hat{\Sigma}\right) \tag{4}$$

We have implemented this method in GAUSS. The procedure `sqrtMatrix` performs the square root decomposition. The algorithm is based on a complex schur decomposition (GOLUB and VAN LOAN [1989]), which is well adapted for **transcendental** functions.

2.2 The 'alternating projections' method

HIGHAM [2000] suggests a method to find the nearest correlation matrix. The problem he considers is, for arbitrary square matrix A, to compute the distance

$$\gamma(A) = \min\{\|A - X\| : X \text{ is a correlation matrix}\} \tag{5}$$

and a matrix achieving this minimum distance. To solve this problem, HIGHAM [2000] proposes to use the 'alternating projections' method, which is an iterative method

$$A \longleftarrow P_U(P_S(A)) \tag{6}$$

where P_U and P_S are the projections onto the sets S and U. These sets are defined by

$$S = \left\{X = X^\top : X \geq 0\right\} \tag{7}$$

and

$$S = \left\{X = X^\top : \mathrm{diag}(X) = \mathbf{1}\right\} \tag{8}$$

Algorithm 1 (Higham [2000, algorithm 3.3 p. 11]) *Given a symmetric square matrix A, this algorithm computes the nearest correlation matrix to A.*

$\Delta S_0 = 0, Y_0 = A$
for $k = 1, 2, \ldots$
  $R_k = Y_{k-1} - \Delta S_{k-1}$
  $X_k = P_S(R_k)$
  $\Delta S_k = X_k - R_k$
  $Y_k = P_U(X_k)$
end

The corresponding GAUSS procedure is `NearestCorrelation`. The implementation uses the following projections

$$P_U(A) = (p_{m,n}), \qquad p_{m,n} = \begin{cases} 1 & \text{if } m = n \\ a_{m,n} & \text{otherwise} \end{cases} \tag{9}$$

and

$$P_S(A) = QD^+Q^\top \tag{10}$$

where $QDQ^\top$ is the Schur decomposition of A and $D^+ = \left(d^+_{m,n}\right)$ where $d^+_{m,n} = \max(d_{m,n}, 0)$.

2.3 Transforming a semidefinite matrix into a definite matrix

With the two previous methods, we obtain a semidefinite matrix. We could then perform simulations using the Cholesky decomposition. Sometimes, it could be useful to obtain a correlation matrix with full rank. The idea is then to replace small eigenvalues with a new value ε. We remark that this method is very similar to the 'square root' method, which can be viewed as a method to correct negative eigenvalues. The GAUSS procedure `DefiniteCorrelation` performs this task.

3 Simulating a random correlation matrix

3.1 The Bendel-Mickey algorithm and the correction of Davies-Higham

Given nonnegative scalars $\lambda_1, \ldots, \lambda_N$, Stewart [1980] proposes an algorithm[1] to define random orthogonal matrices from the Haar distribution with singular values $\lambda_1, \ldots, \lambda_N$. It is then easy to simulate a random correlation matrix C with

$$C = \Sigma \setminus \sigma \setminus \sigma^\top \tag{11}$$

where $\sigma := \text{diag}^{\frac{1}{2}}(\Sigma)$ and Σ a random symmetric matrix. However, the singular values of C are not $\lambda_1, \ldots, \lambda_N$.

Bendel and Mickey [1978] give an algorithm to transform the matrix Σ into a correlation matrix with singular values $\lambda_1, \ldots, \lambda_N$. The main idea is to perform Givens rotations (see section 5.1.8 of Golub and Van Loan [1989]). Let $G(i, j; c, s)$ be the Givens matrix

$$G(i,j;c,s) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & & & & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & & \ddots & & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & & & & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \tag{12}$$

such that the (i,i) element of $G(i,j;c,s)^\top \Sigma G(i,j;c,s)$ is 1 — i and j are such that $i < j$ and $\Sigma_{i,i} < 1 < \Sigma_{j,j}$ (or $\Sigma_{i,i} > 1 > \Sigma_{j,j}$). By performing N sucessive orthogonal transformations $\Sigma \longleftarrow G(i,j;c,s)^\top \Sigma G(i,j;c,s)$, we obtain a correlation matrix with singular values $\lambda_1, \ldots, \lambda_N$.

The previous algorithm is extensively presented in Davies and Higham [2000]. Moreover, they show that

$$c = \frac{1}{\sqrt{1+t^2}} \tag{13}$$

[1] see also Demmel and McKenney [1989].

and

$$s = ct \tag{14}$$

where

$$t = \frac{\Sigma_{i,j} + \sqrt{\Sigma_{i,j}^2 - (\Sigma_{i,i} - 1)(\Sigma_{j,j} - 1)}}{(\Sigma_{j,j} - 1)} \tag{15}$$

The corresponding GAUSS procedure is `RandomCorrelation`.

3.2 Using the nearest correlation matrix

Another procedure called `RandomCorrelation2` is provided and is based on the algorithm to compute the nearest correlation matrix. The idea is to simulate a symmetric matrix and to compute its nearest correlation matrix.

4 Some illustrations

We give now some illustrations of the GAUSS procedures. In the following program, we reproduce the results given in Section 4 of Higham [2000].

```
new;
#include corr.src;

let A[4,4] = 2 -1 0 0
             -1 2 -1 0
             0 -1 2 -1
             0 0 -1 2;


B = NearestCorrelation(A);
C = DefiniteCorrelation(B,0.001);

output file = nearest.out reset;

print B; print inv(B);
print C; print invpd(C);

output off;
```

```
     1.000    -0.8084     0.1916     0.1068
   -0.8084      1.000    -0.6562     0.1916
    0.1916    -0.6562      1.000    -0.8084
    0.1068     0.1916    -0.8084      1.000

-1.347e+009 -2.416e+009 -2.416e+009 -1.347e+009
-2.416e+009 -4.336e+009 -4.336e+009 -2.416e+009
-2.416e+009 -4.336e+009 -4.336e+009 -2.416e+009
-1.347e+009 -2.416e+009 -2.416e+009 -1.347e+009

     1.000    -0.8080     0.1918     0.1069
   -0.8080      1.000    -0.6556     0.1918
    0.1918    -0.6556      1.000    -0.8080
```

```
    0.1069     0.1918    -0.8080      1.000

     120.5      213.5      211.4      117.0
     213.5      382.6      380.7      211.4
     211.4      380.7      382.6      213.5
     117.0      211.4      213.5      120.5
```

To simulate random correlation matrices, we may use `RandomCorrelation` or `RandomCorrelation2`. If we do not specify the eigenvalues, they are generated randomly with uniform distribution.

```
new;
#include corr.src;

N = 5;
lambda = seqa(N,-1,N);

rndseed 123;
corr1 = RandomCorrelation2(N,0,0);

rndseed 123;
corr2 = RandomCorrelation2(N,lambda,0);

rndseed 123;
corr3 = RandomCorrelation(N,lambda,0);

output file = simul1.out reset;

print corr1;
print corr2;
print corr3;

output off;
```

```
     1.000    -0.3786     0.5850     0.3789     0.5586
   -0.3786      1.000    -0.2620    -0.1835    -0.1285
    0.5850    -0.2620      1.000    -0.5250     0.9278
    0.3789    -0.1835    -0.5250      1.000    -0.4788
    0.5586    -0.1285     0.9278    -0.4788      1.000

     1.000    -0.1544     0.1665     0.2027     0.3053
   -0.1544      1.000    -0.1156   -0.06559  -0.007752
    0.1665    -0.1156      1.000    -0.1634     0.4874
    0.2027   -0.06559    -0.1634      1.000    -0.2574
    0.3053  -0.007752     0.4874    -0.2574      1.000

     1.000    -0.3729    -0.2923    0.07129    -0.1228
   -0.3729      1.000    -0.2427    -0.4595   -0.06988
   -0.2923    -0.2427      1.000     0.1884  -0.001852
   0.07129    -0.4595     0.1884      1.000    0.01993
   -0.1228   -0.06988  -0.001852    0.01993      1.000
```

The procedure `RandomCorrelation2` may be used to 'impose' positive or negative correlations.

```
new;
#include corr.src;
```

```
N = 5;

rndseed 123;
corr1 = RandomCorrelation2(N,0,0);

rndseed 123;
corr2 = RandomCorrelation2(N,0,1);

rndseed 123;
corr3 = RandomCorrelation2(N,0,-1);

let e[5,5] = 1 -1 1 1 -1
             -1 1 1 1 1
             1 1 1 1 1
             1 1 1 1 1
            -1 1 1 1 1;

rndseed 123;
corr4 = RandomCorrelation2(N,0,e);

output file = simul2.out reset;

print corr1;
print corr2;
print corr3;
print corr4;

output off;
```

```
     1.000    -0.3786     0.5850     0.3789     0.5586
   -0.3786      1.000    -0.2620    -0.1835    -0.1285
    0.5850    -0.2620      1.000    -0.5250     0.9278
    0.3789    -0.1835    -0.5250      1.000    -0.4788
    0.5586    -0.1285     0.9278    -0.4788      1.000

     1.000     0.2282     0.7843     0.5463     0.6436
    0.2282      1.000     0.4567     0.5096     0.7458
    0.7843     0.4567      1.000     0.8365     0.6745
    0.5463     0.5096     0.8365      1.000     0.3578
    0.6436     0.7458     0.6745     0.3578      1.000

     1.000    -0.4469    -0.2009   -0.09366    -0.4999
   -0.4469      1.000    0.02780  -0.008767    0.06818
   -0.2009    0.02780      1.000    -0.4049    -0.2676
  -0.09366  -0.008767    -0.4049      1.000    -0.4462
   -0.4999    0.06818    -0.2676    -0.4462      1.000

     1.000    -0.5027     0.4542     0.4865    -0.6078
   -0.5027      1.000     0.3851     0.5046     0.8292
    0.4542     0.3851      1.000     0.8959     0.4304
    0.4865     0.5046     0.8959      1.000     0.2865
   -0.6078     0.8292     0.4304     0.2865      1.000
```

In the last example, we consider the Black-Scholes pricing of a Basket option with the following payoff

function

$$\left(S_1(T) - S_2(T) + S_3(T) - S_4(T) - K\right)_+ \mathbf{1}_{[S_5(T)>L]} \tag{16}$$

First, we use a constant-correlation matrix to show the relationship between the price and the constant correlation ρ. Then, we simulate random correlation matrices in order to find option prices bigger than this given by the *minimal correlation matrix*.

```
new;
library pgraph;

#include corr.src;

N = 5;

let sigma = 0.20 0.20 0.20 0.20 0.20;
let S0 = 100 100 100 100 100;
r = 0.05;
tau = 3/12;

K = 5;
L = 105;

proc Payoff(S);
  local S1,S2,S3,S4,S5,payoff;
  S1 = S[.,1];
  S2 = S[.,2];
  S3 = S[.,3];
  S4 = S[.,4];
  S5 = S[.,5];

  payoff = (S1 - S2 + S3 - S4 - K) .* (S5 .> L);
  payoff = payoff .* (payoff .>= 0);

  retp( PayOff );
endp;

rndseed 123;
Ns = 10000|50;

rho = seqa(-1,0.1,21);
rho[8] = -0.2;
option = miss(zeros(rows(rho),1),0);

i = 1;
do until i > rows(rho);

  C = constantCorrelation(rho[i],N);
  if C == error(0);
    i = i + 1;
    continue;
  endif;

  tmp = zeros(Ns[2],1);

  j = 1;
  do until j > Ns[2];
```

```
    S = simulate_mGBM(S0,r,sigma,C,0,tau,Ns[1]);
    tmp[j] = exp(-r*tau)*meanc(Payoff(S));
    j = j + 1;
  endo;

  option[i] = meanc(tmp);

  i = i + 1;
endo;

max = maxindc(option);
max_option = option[max];
max_rho = constantCorrelation(rho[max],N);

M = 25;
option2 = zeros(M,1);
rho2 = zeros(M,N*(N+1)/2);

i = 1;
do until i > M;
  C = RandomCorrelation2(N,0,-1);
  rho2[i,.] = vech(C)';
  tmp = zeros(Ns[2],1);

  j = 1;
  do until j > Ns[2];
    S = simulate_mGBM(S0,r,sigma,C,0,tau,Ns[1]);
    tmp[j] = exp(-r*tau)*meanc(Payoff(S));
    j = j + 1;
  endo;

  option2[i] = meanc(tmp);

  i = i + 1;
endo;

max2 = maxindc(option2);
max2_option = option2[max2];
max2_rho = xpnd(rho2[max2,.]');

M = 25;
option3 = zeros(M,1);
rho3 = zeros(M,N*(N+1)/2);

i = 1;
do until i > M;
  C = RandomCorrelation(N,0,0);
  rho3[i,.] = vech(C)';
  tmp = zeros(Ns[2],1);

  j = 1;
  do until j > Ns[2];
    S = simulate_mGBM(S0,r,sigma,C,0,tau,Ns[1]);
    tmp[j] = exp(-r*tau)*meanc(Payoff(S));
    j = j + 1;
  endo;
```

```
  option3[i] = meanc(tmp);

  i = i + 1;
endo;

max3 = maxindc(option3);
max3_option = option3[max3];
max3_rho = xpnd(rho3[max3,.]');

output file = ma.out reset;

print max_rho; print; print ftos(max_option,''                Option price = %lf'',5,3);
print ''-------------------------------------------------------'';
print max2_rho; print; print ftos(max2_option,''                Option price = %lf'',5,3);
print ''-------------------------------------------------------'';
print max3_rho; print; print ftos(max3_option,''                Option price = %lf'',5,3);

output off;

option2 = selif(option2, option2 .> max_option);
e2 = ones(rows(option2),1);

option3 = selif(option3, option3 .> max_option);
e3 = ones(rows(option3),1);

graphset;
  _pdate = ''''; _pnum = 2; _pframe = 0;
  fonts(''simplex simgrma'');
  xlabel(''\202r\201'');
  ylabel(''Option price'');
  xtics(-1,1,0.25,5);
  ytics(0,4,0.5,5);
  _pline = e2.*(1~6~-1)~option2~e2~option2~e2.*(1~15~0) |
           e3.*(1~3~-1)~option3~e3~option3~e3.*(1~15~0) ;
  graphprt(''-c=1 -cf=ma.ps'');
  xy(rho,option);


proc simulate_mGBM(S0,mu,sigma,rho,t0,TT,Ns);
  local ST,N,dt,dw,Pchol;

  S0 = S0'; mu = mu'; sigma = sigma';

  N = maxc(cols(S0)|cols(mu)|cols(sigma));
  dt = TT - t0;

  if rho == 1;
    dw = sqrt(dt) .* rndn(Ns,1) .* ones(1,N);
  elseif rho == -1 and N == 2;
    dw = sqrt(dt) .* (1~-1) .* rndn(Ns,1);
  else;
    Pchol = chol(rho);
    dw = sqrt(dt) .* rndn(Ns,N)*Pchol;
  endif;
```

```
  ST = S0 .* exp( (mu-0.5*sigma^2) .* dt + sigma .* dw );

  retp(ST);
endp;
```

```
     1.000    -0.2000    -0.2000    -0.2000    -0.2000
  -0.2000       1.000    -0.2000    -0.2000    -0.2000
  -0.2000    -0.2000       1.000    -0.2000    -0.2000
  -0.2000    -0.2000    -0.2000       1.000    -0.2000
  -0.2000    -0.2000    -0.2000    -0.2000       1.000

                Option price = 2.161
------------------------------------------------------

     1.000    -0.4417    0.04430    -0.3713    -0.1110
  -0.4417       1.000    -0.2925   -0.02599    -0.3066
  0.04430    -0.2925       1.000    -0.4690    0.09158
  -0.3713   -0.02599    -0.4690       1.000    -0.5193
  -0.1110    -0.3066    0.09158    -0.5193       1.000

                Option price = 4.108
------------------------------------------------------

     1.000    -0.2024     0.6456    0.09182    0.06828
  -0.2024       1.000    -0.4896     0.3457    -0.1525
   0.6456    -0.4896       1.000     0.1606    -0.2238
  0.09182     0.3457     0.1606       1.000    -0.4085
  0.06828    -0.1525    -0.2238    -0.4085       1.000

                Option price = 3.348
```

References

[BM] Bendel, R.B. and M.R. Mickey [1978], Population correlation matrices for sampling experiments, *Communications in Statistics – Simulation and Computation*, **7(2)**, 163-182

[BS] Beneš, V. and J. Štěpán [1997], Distributions with Given Marginals and Moment Problems, Kluwer Academic Publishers, Dordrecht

[DH] Davies, P.I. and N.J. Higham [2000], Numerically stable generation of correlation matrices and their factors, *BIT*, **40(4)**, 640-651 (available from `http://www.ma.man.ac.uk/~higham/`)

[DM] Demmel, J.W. and A. McKenney [1989], A test matrix generation suite — LAPACK Working Note 9, Courant Institute, *Technical Report*

[GL] Golub, G.H. and C.F. Van Loan [1989], Matrix Computations, John Hopkins University Press, second edition, Baltimore

[Higham] Higham, N.J. [2000], Computing the nearest correlation matrix — A problem from finance, Department of Mathematics, University of Manchester, *Numerical Analysis Report*, **369** (available from `http://www.ma.man.ac.uk/~higham/`)

[HJ] Horn, R.A. and C.R. Johnson [1991], Topics in Matrix Analysis, Cambridge University Press, Cambridge

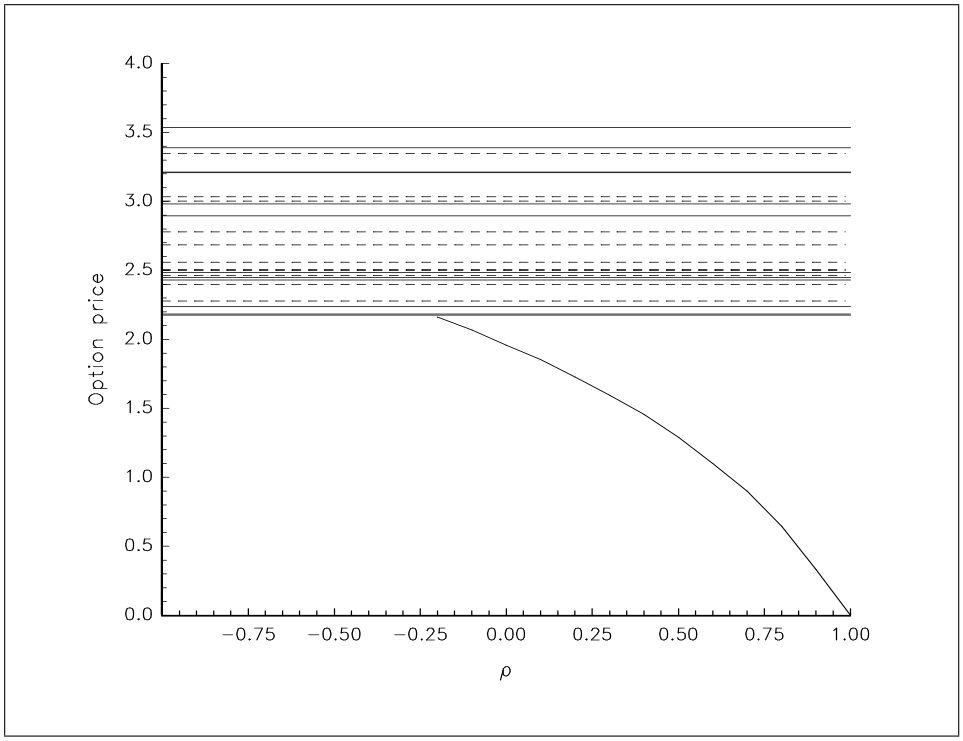

Figure 1: Price of the Basket option

[LB] Lin, S.P. and R.B. Bendel [1985], Algorithm AS213: Generation of population correlation on matrices with specified eigenvalues, *Applied Statistics*, **34**, 193-198

[Stewart] Stewart, G.W. [1980], On efficient generation of random orthogonal matrices with an application to condition estimation, *SIAM Journal of Numerical Analysis*, **17(3)**, 403-409

[TH] Tiit, E-M. and H-L. Helemäe [1997], Boundary distributions with fixed marginals, in V. Beneš and J. Štěpán (eds.), *Distributions with Given Marginals and Moment Problems*, Kluwer Academic Publishers, Dordrecht

A Command reference

A.1 `constantCorrelation`

■ **Purpose**
Defines a constant-correlation matrix.

■ **Format**
corr = constantCorrelation(rho,N);

■ **Input**

rho	scalar
N	scalar

■ **Output**

corr	$N \times N$ matrix

■ **Remark**
$C(\rho)$ is a constant-correlation matrix iff

$$1 \geq \rho \geq -(N-1)^{-1}$$

(see Tiit and Helemäe [1997]).

■ **Source**
corr.src

A.2 DefiniteCorrelation

■ **Purpose**
Transforms a semidefinite correlation matrix into a positive correlation matrix.

■ **Format**
B = DefiniteCorrelation(A,lambda);

■ **Input**

A	$N \times N$ matrix
lambda	scalar or $N \times 1$ vector

■ **Output**

B	$N \times N$ matrix

■ **Globals**

_corr_eps	scalar, epsilon value (default = 10^{-10})

■ **Remark**
if `lambda` is set to 0, the eigenvalues which are smaller than `_corr_eps` are initialized to `_corr_eps`.

■ **Source**
corr.src

A.3 NearestCorrelation

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

■ **Format**
corr2 = NearestCorrelation(corr1);

■ **Input**

corr1	$N \times N$ matrix

■ **Output**

corr2	$N \times N$ matrix

■ **Globals**

_corr_maxit	scalar, maximum number of iterations (default = 200)
_corr_tol	scalar, tolerance value (default = 10^{-10})

■ **Remark**
The algorithm stops when the number of iterations exceeds `_corr_maxit` or when

$$\frac{\|P_U(P_S(A)) - A\|_2}{\|A\|_2} \leq \texttt{\_corr\_tol}$$

■ **Source**
corr.src

A.4 RandomCorrelation

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

■ **Format**
corr = RandomCorrelation(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**
The procedure is based on Algorithm 3.1 of Davies and Higham [2000]. The random symmetric orthogonal matrix from the Haar distribution is simulated using the LAPACK/Matlab implementation of Demmel and McKenney [1989].

■ **Source**
corr.src

A.5 RandomCorrelation2

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

■ **Format**
corr = RandomCorrelation2(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

A.6 sqrtMatrix

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

■ **Format**
B = sqrtMatrix(A);

■ **Input**

A	$N \times N$ matrix

■ **Output**

B	$N \times N$ matrix

■ **Source**
corr.src

A.7 vcx2

■ **Purpose**
Computes the covariance matrix from the data matrix.

■ **Format**
{vc,m} = vcx2(x,mtd);

■ **Input**

x	$N \times K$ matrix, data
mtd	scalar 0 if no missing data 1 for a listwise deletion 2 for a pairwise deletion 3 for a complete pairwise deletion 4 for a modified listwise deletion 5 for a modified complete pairwise deletion

■ **Output**

vc	$K \times K$ matrix, covariance matrix
m	$K \times 1$ vector, means

■ **Remark**
There are different methods to compute the covariance matrix Σ of the matrix X. Let $\mathbf{m}$ and $\boldsymbol{\sigma}$ denote the vector of means and the vector of standard deviations, we have

$$\Sigma = \frac{1}{T-1}\left(X - \mathbf{1} \odot \mathbf{m}^\top\right)^\top \left(X - \mathbf{1} \odot \mathbf{m}^\top\right)$$

where T is the number of observations and

$$\mathbf{m} = \frac{1}{T} X^\top \mathbf{1}$$

The *listwise deletion* method consists in deleting all observations from the dataset that contains a missing value. When *pairwise deletion* is used, any element of the matrix that is missing is excluded from the computation of the covariance matrix V, but not for the means $\mathbf{m}$. In this case, we have

$$\Sigma_{i,j} = \frac{1}{T(i,j) - 1} \sum_{t=1}^{T} 1_{[\text{ɔ}(i,j,t)]} \cdot \left(X_{t,i} - m_{i|j}\right)\left(X_{t,j} - m_{j|i}\right)$$

with $\exists(i,j,t) = (X_{t,i} \; != NaN \quad \&\& \quad X_{t,j} \; != NaN)$, $T(i,j) = \sum_{t=1}^{T} 1_{[\exists(i,j,t)]}$ and

$$m_{i|j} = \frac{1}{T(i,i)} \sum_{t=1}^{T} 1_{[\exists(i,i,t)]} \cdot X_{t,i}$$

In this case, the mean $m_{i|j}$ does not depend on the variable j. When *complete paiwise deletion* is used, we have

$$m_{i|j} = \frac{1}{T(i,j)} \sum_{t=1}^{T} 1_{[\exists(i,j,t)]} \cdot X_{t,i}$$

In this case, the mean $m_{i|j}$ depends on the variable j. In *modified deletions*, the covariance matrix Σ is estimated using the following formula

$$\Sigma = \boldsymbol{\sigma} C \boldsymbol{\sigma}^{\top}$$

where C is the correlation matrix. In this case, we may compute C with *listwise deletion* or *complete paiwise deletion*, but the standard deviations are estimated with all the available data

$$\boldsymbol{\sigma}_i = \frac{1}{T(i,i) - 1} \sum_{t=1}^{T} 1_{[\exists(i,i,t)]} \cdot \left(X_{t,i} - \frac{1}{T(i,i)} \sum_{t=1}^{T} 1_{[\exists(i,i,t)]} \cdot X_{t,i} \right)^2$$

■ **Source**
corr.src

B The source code

```
/*
**  corr.src - corr library.
**
**  Procedure                   Object                                    Line
** ============================================================================
**  constantCorrelation         Defines a constant-correlation matrix      33
**
**  DefiniteCorrelation         Transforms a semidefinite correlation matrix
**                              into a positive correlation matrix         56
**
**  NearestCorrelation          Computes the nearest correlation matrix   118
**
**  RandomCorrelation           Simulates a random correlation matrix
**                              using the random orthogonal algorithm     169
**
**  RandomCorrelation2          Simulates a random correlation matrix
**                              using the nearest correlation algorithm   340
**
**  sqrtMatrix                  Computes the matrix square root           429
**
**  vcx2                        Computes the covariance matrix from
**                              the data matrix                           453
**
*/
```

```
declare matrix _corr_maxit  = 200;
declare matrix _corr_tol = 1e-10;
declare matrix _corr_eps = 1e-10;


/*
**> constantCorrelation
**
** Purpose: Defines a constant-correlation matrix.
**
** Format:  corr = constantCorrelation(c,N);
**
** Input:       c - scalar
**              N - scalar
**
** Output:   corr - matrix N*N
**
*/

proc (1) = constantCorrelation(c,N);
  if c >= -1/(N-1);
    retp( diagrv(c .* ones(N,N),ones(N,1)) );
  else;
    retp( error(0) );
  endif;
endp;


/*
**> DefiniteCorrelation
**
** Purpose: Transforms a semidefinite correlation matrix
**          into a positive correlation matrix.
**
** Format:  B = DefiniteCorrelation(A,lambda);
**
** Input:       A - matrix N*N
**         lambda - vector N*1 or scalar
**
** Output:      B - matrix N*N
**
** Globals:
**    _corr_eps    - scalar, epsilon value (default = 1e-10)
**
** Remark:
**   If lambda is set to 0, the eigenvalues which are smaller than _corr_eps
**   are initialized to _corr_eps.
**
*/

proc DefiniteCorrelation(A,lambda);
  local N,S,Q,B,factor,e,sigma;

  N = rows(A);
  {S,Q} = eighv(A);
  S = real(S);
```

```
  Q = real(Q);

  if lambda == 0;
    lambda = _corr_eps;
  endif;

  if rows(lambda) == N;
    S =  diagrv(zeros(N,N),lambda);
    B = Q*S*inv(Q);
    sigma = sqrt(diag(B));
    B = B ./ sigma ./ sigma';
  elseif rows(lambda) == 1;
    e = S .> lambda;
    factor = sumc(S - (1-e)*lambda) / sumc(e.*S);
    S = diagrv(zeros(N,N),factor * (e .* S) + (1-e) * lambda);
    B = Q*S*inv(Q);
    sigma = sqrt(diag(B));
    B = B ./ sigma ./ sigma';
  else;
    B = error(0);
  endif;

  retp( real(B) );
endp;


/*
**> NearestCorrelation
**
** Object: Computes the nearest correlation matrix.
**
** Format: corr2 = NearestCorrelation(corr1);
**
** Input:    corr1 - N*N matrix
**
** Output:   corr2 - N*N matrix
**
** Globals:
**    _corr_maxit  - scalar, maximum number of iterations (default = 200)
**    _corr_tol    - scalar, tolerance value (default = 1e-10)
**
** Reference:
**    Higham, N.J. [2000], Computing the nearest correlation matrix ---
**    A problem from finance, Department of Mathematics, University of Manchester,
**    Numerical Analysis Report, 369
*/

proc (1) = NearestCorrelation(Y);
  local dS,k,R,X,Y_,tol;

  dS = 0;
  tol = _corr_tol + 1;

  k = 0;
  do until k > _corr_maxit or tol < _corr_tol;
    R = Y - dS;
```

```
    X = _NearestCorrelation_PS(R);
    dS = X - R;
    Y_ = _NearestCorrelation_PU(X);
    tol = sqrt(sumc(sumc((Y_-Y)^2)))/sqrt(sumc(sumc((Y_)^2)));
    Y = Y_;
    k = k + 1;
  endo;

  retp(Y);
endp;

proc _NearestCorrelation_PU(C);
  local B;
  B = diagrv(C,ones(rows(C),1));
  retp(B);
endp;

Proc _NearestCorrelation_PS(C);
  local m,D,Q,lambda;

  m = rows(C);
  {D,Q} = schur(C);
  lambda = diag(D);
  D = diagrv(eye(m),lambda .* (lambda .> 0));
  C = Q * D * Q';
  retp(C);
endp;


/*
**> RandomCorrelation
**
** Object: Simulates a random correlation matrix using the random
**         orthogonal algorithm.
**
** Format: corr = RandomCorrelation(N,lambda,kappa);
**
** Input:        N - scalar, dimension of the correlation matrix
**          lambda - N*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 geoemtrically 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*N matrix
**
** Reference:
**   Davies, P.I. and N.J. Higham [2000], Numerically stable generation of correlation
**   matrices and their factors, BIT, 40(4), 640-651
**
**
*/
```

```
proc (1) = RandomCorrelation(n,lambda,kappa);
  local corr,sigma;
  local d,i,j,a,G,iter;

  if lambda == 0;
    lambda = rndu(n,1);
    lambda = n * lambda /sumc(lambda);
  else;
    lambda = n * lambda /sumc(lambda);
  endif;

  corr = _RandomCorrelation_svd(n,kappa,lambda);

  iter = 1;
  do until iter > n-1;
    d = diag(corr);

    if d == ones(n,1);
      break;
    endif;

    i = minc(indexcat(d .< 1,1));
    j = maxc(indexcat(d .> 1,1));
    if i > j;
      i = minc(indexcat(d .> 1,1));
      j = maxc(indexcat(d .< 1,1));
    endif;
    a = corr[i,i]~corr[i,j] |
        corr[j,i]~corr[j,j];
    G = _randomcorrelation_givens(a,n,i,j);
    corr = G'*corr*G;
    corr[i,i] = 1;

    iter = iter + 1;
  endo;

  retp(corr);
endp;

proc _randomcorrelation_givens(a,n,i,j);
  local aii,aij,ajj,t,c,s,G,Givens;

  aii = a[1,1];
  aij = a[1,2];
  ajj = a[2,2];

  t = (aij + sqrt(aij^2 - (aii-1)*(ajj-1) )) / (ajj - 1);
  c = 1/sqrt(1+t^2);
  s = c*t;

  G = eye(n);
  G[i,i] = c;
  G[i,j] = s;
  G[j,i] = -s;
  G[j,j] = c;
```

```
  retp(G);
endp;


proc (1) = _randomcorrelation_svd(n,kappa,mode);
  local absmode,sgnmode,sigma,Q,A;

  if kappa == 0;
    kappa = sqrt(1/__macheps);
  endif;

  if kappa < 1;
    errorlog "error: condition number must greater than one.";
    end;
  endif;

  absmode = abs(mode); sgnmode = mode[1] .< 0;

  if absmode == 1;
    sigma = 1 | ones(n-1,1)/kappa;
  elseif absmode == 2;
    sigma = ones(n-1,1) | (1/kappa);
  elseif absmode == 3;
    sigma = (kappa^(-1/(n-1))) .^ seqa(0,1,n);
  elseif absmode == 4;
    sigma = ones(n,1) - seqa(0,1,n)/(n-1)*(1-1/kappa);
  elseif absmode == 5;
    sigma = exp( -rndu(n,1) * ln(kappa) );
  elseif rows(mode) == n;
    sigma = mode;
  endif;

  if sgnmode < 0 and rows(mode) /= n;
    sigma = rev(sigma);
  endif;
  sigma = diagrv(zeros(n,n),sigma);

  Q = _randomcorrelation_Qmult(n);
  A = Q'sigma*Q;
  A = 0.5*(A + A');

  retp(A);
endp;

proc (1) = _randomcorrelation_Qmult(n);
  local A,d,k,x,s,sgn,beta,y,i;

  A = eye(n);
  d = zeros(n,n);

  k = n-1;
  do until k < 1;
    x = rndn(n-k+1,1);
    s = sqrt(x'x);
    sgn = _randomcorrelation_sign( x[1] ) + (x[1] == 0);
    s = sgn*s;
```

```
    d[k,1] = -sgn;
    x[1] = x[1] + s;
    beta = s*x[1];

    y = x'A[k:n,.];
    A[k:n,.] = A[k:n,.] - x*(y/beta);

    k = k - 1;
  endo;

  i = 1;
  do until i > n-1;
    A[i,.] = d[i,1] *A[i,.];
   i = i + 1;
  endo;

  A[n,.] = _randomcorrelation_sign(rndn(1,1)) * A[n,.];
  retp(A);
endp;

proc (1) = _randomcorrelation_sign(A);
  local e1,e2,s;

  e1 = A .> 0;
  e2 = A .< 0;

  s = e1 - e2;
  retp(s);
endp;



/*
**> RandomCorrelation2
**
** Object: Simulates a random correlation matrix using the nearest
**         correlation algorithm.
**
** Format: corr = RandomCorrelation2(N,lambda,e);
**
** Input:        N - scalar, dimension of the correlation matrix
**          lambda - N*1 vector, specified eigenvalues
**                   (0 for random uniform eigenvalues)
**               e - N*N vector, matrix of the signs of the correlation values
**                   0 for random correlations
**                   1 for positive correlations
**                   -1 for negative correlations
**
** Output:   corr - N*N matrix
**
** Remark: RandomCorrelation uses the procedure NearestCorrelation.
**
*/

proc (1) = RandomCorrelation2(n,lambda,e);
  local corr;
```

```
  if e == 0;
    corr = _RandomCorrelation_1(n,lambda);
  elseif e == 1;
    corr = _RandomCorrelation_2(n,lambda,1);
  elseif e == -1;
    corr = _RandomCorrelation_2(n,lambda,-1);
  elseif rows(e) == n and cols(e) == n;
    corr = _RandomCorrelation_2(n,lambda,e);
  else;
    corr = error(0);
  endif;

  retp(corr);
endp;

proc _RandomCorrelation_1(n,lambda);
  local c,d,q;

  if lambda /= 0;
    n = rows(lambda);
    lambda = n * lambda /sumc(lambda);
    c = 2*rndu(n*(n+1)/2,1)-1;
    c = diagrv(xpnd(c),ones(n,1));
    c = NearestCorrelation(c);
    {D,Q} = schur(c);
    D = diagrv(eye(rows(c)),lambda);
    c = Q * D * Q';
    c = NearestCorrelation(c);
  else;
    c = 2*rndu(n*(n+1)/2,1)-1;
    c = diagrv(xpnd(c),ones(n,1));
    c = NearestCorrelation(c);
    c = DefiniteCorrelation(c,_corr_eps);
  endif;

  retp(c);
endp;

proc _RandomCorrelation_2(n,lambda,e);
  local c,d,q;

  e = e .> 0;

  if lambda /= 0;
    n = rows(lambda);
    lambda = n * lambda /sumc(lambda);
    c = vech(e .* rndu(n,n) - (1-e) .* rndu(n,n));
    c = diagrv(xpnd(c),ones(n,1));
    c = NearestCorrelation(c);
    {D,Q} = schur(c);
    D = diagrv(eye(rows(c)),lambda);
    c = Q * D * Q';
    c = NearestCorrelation(c);
  else;
    c = vech(e .* rndu(n,n) - (1-e) .* rndu(n,n));
```

```
    c = diagrv(xpnd(c),ones(n,1));
    c = NearestCorrelation(c);
    c = DefiniteCorrelation(c,_corr_eps);
  endif;

  retp(c);
endp;


/*
**> sqrtMatrix
**
** Purpose: computes the matrix square root.
**
** Format:  B = sqrtMatrix(A);
**
** Input:   A - matrix N*N
**
** Output:  B - matrix N*N
**
*/

proc sqrtMatrix(A);
  local N,T,Q,B;

  N = rows(A);
  {T,Q} = schtoc(schur(A));
  B = Q * diagrv(zeros(N,N),sqrt(diag(T))) * Q';

  retp(B);
endp;


/*
**> vcx2
**
** Purpose: Computes the covariance matrix from the data matrix.
**
** Format:  {vc,m} = vcx2(x,mtd);
**
** Input:       x - matrix N*K
**            mtd - scalar
**                  0 if no missing data
**                  1 for a listwise deletion
**                  2 for a pairwise deletion
**                  3 for a complete pairwise deletion
**                  4 for a modified listwise deletion
**                  5 for a modified complete pairwise deletion
**
** Output:     vc - matrix K*K, covariance matrix
**              m - vector K*1, means
**
*/

proc (2) = vcx2(x,mtd);
  local e,m,xc,r,Mcov,sigma,stderr,Mcor;
```

```
local ee,y,i,means;

if mtd == 0;

  if ismiss(x);
    ERRORLOG "error: the data contains missing values.";
    retp(error(0));
  endif;

  m = meanc(x);
  xc = x - m';
  Mcov = moment(xc,0) / (rows(x) - 1);

elseif mtd == 1;             /* Listwise deletion */

  e = x ./= miss(0,0);
  m = sumc(missrv(x,0)) ./ sumc(e);
  xc = x - m';
  xc = packr(xc);
  Mcov = (xc'xc) / (rows(xc) - 1);

elseif mtd == 2;             /* Pairwise deletion */

  e = x ./= miss(0,0);
  m = sumc(missrv(x,0)) ./ sumc(e);
  xc = x - m';
  ee = e'e;
  Mcov = moment(xc,2) ./ (ee - 1);
  Mcov = Mcov .* ( ee .> 1 );

elseif mtd == 3;             /* Complete pairwise deletion */

  e = x ./= miss(0,0);
  x = missrv(x,0);

  Mcov = zeros(cols(e),cols(e));

  i = 1;
  do until i > cols(e);
    ee = e[.,i] .and e;
    y = x .* ee;
    m = sumc(y) ./ sumc(ee);
    y = y - m';
    y = y .* ee;
    Mcov[.,i] = (y'y[.,i])./ ( sumc(ee) - 1 );
    i = i + 1;
  endo;

  ee = e'e;
  Mcov = Mcov .* ( ee .> 1 );
  Mcov = missrv(Mcov,0);

elseif mtd == 4;            /* Modified listwise deletion*/

  e = x ./= miss(0,0);
  m = sumc(missrv(x,0)) ./ sumc(e);
```

```
    xc = x - m';
    sigma = sqrt(sumc(missrv(xc,0)^2) ./ (sumc(e) - 1));

    xc = packr(xc);
    Mcov = (xc'xc) / (rows(xc) - 1);
    stderr = sqrt(diag(Mcov));
    Mcor = Mcov ./ stderr ./ stderr';

    Mcov = sigma * Mcor * sigma';

  elseif mtd == 5;             /* Modified complete pairwise deletion*/

    e = x ./= miss(0,0);
    x = missrv(x,0);

    Mcor = zeros(cols(e),cols(e));

    i = 1;
    do until i > cols(e);
      ee = e[.,i] .and e;
      y = x .* ee;
      m = sumc(y) ./ sumc(ee);
      y = y - m';
      y = y .* ee;
      Mcor[.,i] = ( y'y[.,i] ) ./ sqrt( sumc(y^2)*sumc(y[.,i]^2) );
      i = i + 1;
    endo;

    m = sumc(missrv(x,0)) ./ sumc(e);
    xc = x - m';
    sigma = sqrt(sumc(missrv(xc,0)^2) ./ (sumc(e) - 1));

    ee = e'e;
    Mcov = Mcor .* sigma .* sigma';
    Mcov = Mcov .* ( ee .> 1 );
    Mcov = missrv(Mcov,0);

  else;

    retp(error(0),error(0));

  endif;

  e = x ./= miss(0,0);
  means = sumc(missrv(x,0)) ./ sumc(e);

  retp(Mcov,means);
endp;
```