Completing correlation matrices of arbitrary order by differential evolution method of global optimization: A Fortran program

SK Mishra

North-Eastern Hill University, Shillong (India)

5. March 2007

Online at http://mpra.ub.uni-muenchen.de/2000/
Introduction: A product moment correlation matrix $R$ of order $n$ is a (square) symmetric positive semi-definite matrix such that $r_{ij} = r_{ji} \in R$ lies between $-1$ and $1$. Moreover, $r_{ii} = 1$. Each $r_{ij}$ is the cosine of angle $\theta$ between two variates, say $x_i$ and $x_j$; $i,j \in \{1,2,\ldots,n\}$. Such matrices have many applications, particularly in marketing and financial economics as reflected in the works of Chesney and Scott (1989), Heston (1993), Schöbel and Zhu (1999), Tyagi and Das (1999), Xu and Evers (2003), etc. The need to forecast demand for a group of products in order to realize savings by properly managing inventories requires the use of correlation matrices (Budden et al. 2007).

In some cases, the matrix available to the analyst/decision-maker is complete, but it is an invalid (not positive semi-definite) correlation matrix. There could be many reasons that give rise to such invalid matrices (Mishra, 2004). In such cases, the problem is to obtain an approximate semi-definite correlation matrix, which, in some sense, is closest to the given invalid matrix. A number of methods have been developed to obtain such nearest correlation matrices. The works of Rebonato and Jäckel (1999), Higham (2002), Anjos et al. (2003), Pietersz and Groenen (2004), Grubisic and Pietersz (2004) and Mishra (2004) are some of them.

In many cases, however, due to paucity of data/information or dynamic nature of the problem at hand, it is not possible to obtain a complete correlation matrix. Some elements of $R$ are unknown. In such cases, the question of validity (semi-definiteness) or otherwise (of an incomplete correlation matrix) does not arise. Instead, the problem is to obtain a valid complete correlation matrix. In absence of sufficient side conditions that are often impracticable to specify, this problem cannot be solved uniquely.

Several methods have been suggested to complete a correlation matrix - that is to obtain a valid complete correlation matrix from an incomplete correlation matrix (some of whose elements are unknown). Works of Johnson (1980), Barett et al. (1989), Helton et al. (1989), Grone et al. (1984), Barett et al. (1998), Laurent (2001), Kahl and Jäckel (2005), Kahl and Günther (2005), etc are notable.

In view of non-unique solutions admissible to the problem of completing the correlation matrix, some authors have suggested numerical methods that provide ranges to different unknown elements. Stanley and Wang (1969), Glass and Collins (1970) and Olkin (1981) have suggested very efficient methods to find such ranges for the unknown elements of very small correlation matrices (of order $n < 4$). Budden (2007) suggests a method to obtain the ranges of missing values of elements of a $4 \times 4$ incomplete correlation matrix whose first row elements are known. With the known elements in the
first row, the method sets the range for $r_{23}$ and one has to specify its value in that range. Once the value of $r_{23}$ is chosen (within the specified range set for it), the method yields the range in which $r_{24}$ would lie. One has to specify the value of $r_{24}$ within the given range, which yields the range for $r_{34}$. Thus the matrix is completed. In this procedure it is obvious that the ranges on latter elements are contingent upon the choice of values of former elements. Further, Budden’s method is limited to a $4 \times 4$ correlation matrix.

**Objective of the Present Paper:** Our objective in this paper is to suggest a method (and provide a Fortran program) that completes a given incomplete correlation matrix of an arbitrary order. The resulting complete matrices are many in number, but all of them are valid (positive semi-definite – with all non-negative eigenvalues). Additionally, the suggested method does not require any pre-assigned pattern as in case of Budden’s method. It allows for holes (unknown elements) in any row and any column. The program that works out such complete matrices does not require any interaction with the user either.

**The Method:** The method proposed here is based on the Differential Evolution (DE) procedure of global optimization (Storn and Price, 1995). It generates a random population of elements that fit the holes ($m$ in number) in the given incomplete correlation matrix, yielding valid correlation matrices whose eigenvalues are all non-negative summing up to the order of the matrix, which is also the trace of the matrix.

The differential Evolution method is perhaps the fastest evolutionary computational procedure yielding most accurate solutions to continuous global optimization problems. It consists of three basic steps: (i) generation of (large enough) population with individuals in the $m$-dimensional space, randomly distributed over the entire domain of the function in question and evaluation of the individuals of the so generated by finding $f(x)$, where $x$ is the decision variable; (ii) replacement of this current population by a better fit new population, and (iii) repetition of this replacement until satisfactory results are obtained or certain criteria of termination are met.

The strength of DE lays on replacement of the current population by a new population that is better fit. Here the meaning of ‘better’ is in the Pareto improvement sense. A set $S_a$ is better than another set $S_b$ iff: (i) no $x_i \in S_a$ is inferior to the corresponding member of $x_i \in S_b$; and (ii) at least one member $x_k \in S_a$ is better than the corresponding member $x_k \in S_b$. Thus, every new population is an improvement over the earlier one. To accomplish this, the DE method generates a candidate individual to replace each current individual in the population. A crossover of the current individual and three other randomly selected individuals obtains the candidate individual from the current population. The crossover itself is probabilistic in nature. Further, if the candidate individual is better fit than the current individual, it takes the place of the current individual else the current individual passes into the next iteration (Mishra, 2006).

In the present application of DE, the ‘complete correlation problem’ is cast into a minimization problem. It may be noted that the problem has innumerably many minima.
and we need multiple solutions. Such problems cannot be solved satisfactorily by conventional optimization procedures. A stochastic population method such as DE or PSO (Particle Swarm Optimization) may, therefore, be a suitable choice. In the scheme of DE, a population of $N$ individuals (each represented by a $m$-dimensional vector, of which each element lies between $-1$ and $1$) is generated by using uniformly distributed random numbers whose each vector provides the candidate values filling in the $m$ number of holes (unknown elements) of the given incomplete matrix. The eigenvalues of the resulting matrices are computed and positive penalties are set if any of them is negative. Minimization of this formulation results into zero penalty, and the solution so obtained yields a valid correlation matrix. Since each individual in the population has gravitational pull to the global optimum, it corresponds to a valid correlation matrix. Thus, we obtain $N$ number of valid correlation matrices.

The Structure of Computer Program and Hints on its Use: The main program (in Fortran) to complete a correlation matrix has eight subroutines. The main program reads the input matrix from a file specified by the user. This file stores the main diagonal and upper diagonal elements of the given matrix. Thus the first row has $n$ elements beginning with 1.0; the second row has $n-1$ elements beginning with 1.0 and so on such that the last ($n^{th}$) row has only one element (=1.0). In making the input matrix file one has to indicate the known and the unknown elements differently. While the known elements naturally lie between $-1$ and $1$ they are put as they really are. However, a number lying beyond the range $[-1, 1]$ represents an unknown element. The value could be any number such as 2, -3, 1.5, etc that cannot be a correlation coefficient. For example, if $r_{ij}$ is known to be 0.73, say, it will be put as 0.73, but if $r_{ij}$ is unknown it may be represented by a number, say 2.0 or $-1.9$ and so on. A number outside the range [-1,1] indicates that it is a hole or an unknown correlation coefficient. When the program runs, it asks for the order of input matrix (morder) and the name of input data file in which the input matrix is already stored. The user has to specify them. The program also asks to name the output file in which the final results (valid correlation matrices) would be stored. The user should specify it. Then the program asks for a random number seed. Any 4-digit odd number (say 1271) can be fed as a seed. Subsequently, the program asks for the number of unknown elements (m) in the input matrix. This also has to be given by the user. The main program calls subroutine DE (differential Evolution optimizer). It asks for inputs from the user; the population size (N) and the number of iteration to be performed. The population size determines the number of valid matrices to be obtained as output. It should be normally 100 or so, but for larger problems, this number should be larger. The number of iterations should be specified at 1000 or larger. Then the program needs another random number seed that could be any 4-digit odd number. Once these inputs are given, DE starts running.

Other subroutines in the program are: Normal (generates normally distributed random numbers), Random (generates uniformly distributed random numbers between 0 and 1), Fselect (chooses a function), Func (organizes function calls), Eigen (computes eigenvalues and vectors), Concor (constructs correlation matrices for optimization) and Ncorx (constructs valid correlation matrices and stores them in the output file specified by the user). The output file may be opened in notepad or by any editor program (edit or
Microsoft Word of Microsoft Windows) to obtain the results. The source codes (Fortran programs) are appended here. Directly usable source codes that may be cut and pasted in an editor may be downloaded from http://www1.webng.com/economics/complete-cormat.txt or http://www.freewebs.com/nehu_economics/complete-cormat.txt. A Fortran compiler may be obtained from http://www.thefreecountry.com/compilers/fortran.shtml or http://www.download.com/Force/3000-2069_4-10233344.html freely. The source codes may be pasted in the Force editor directly. Presently, the dimensions in the program are set to deal with the matrices of order 10 or less. If needed, they may be increased suitably for larger matrices.

An Example: An incomplete matrix of order 7 (=morder =n) given in table-1 is used as an example to illustrate an application of the method and program given in this paper. It has 12 (=m) holes or unknown elements (colored red). They have been assigned an invalid number (5), outside the permissible range [-1, 1]. Other numbers in the range [-1, 1] are known elements of the matrix. The program is run for population size N=100 and it gives N valid correlation matrices. Two sample matrices from the output are given in table-2 and table-3. The program also gives the eigenvectors for each valid correlation matrix, but they are not presented here.

<table>
<thead>
<tr>
<th>Table-1. Input Correlation Matrix with Some Unknown Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
</tr>
<tr>
<td>1.00</td>
</tr>
<tr>
<td>1.00</td>
</tr>
<tr>
<td>1.00</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table-2. Sample Output Correlation Matrix and its Eigenvalues</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00000000-0.5000000 0.50000000-0.5000000 0.56000000 0.21000000 0.34000000</td>
</tr>
<tr>
<td>-0.5000000 1.00000000-0.0285722 0.1840863 -0.0967958 0.30000000 0.16000000</td>
</tr>
<tr>
<td>0.50000000-0.0285722 1.00000000-0.0249011 0.3674891 0.14763300 0.89000000</td>
</tr>
<tr>
<td>-0.50000000-0.1840863-0.0249011 1.00000000 0.0894851-0.0430459-0.0959958</td>
</tr>
<tr>
<td>0.50000000-0.0967958 0.3674891 0.0894851 1.00000000 0.2641564 0.2404028</td>
</tr>
<tr>
<td>0.21000000 0.30000000 0.14763300-0.0430459 0.2641564 1.00000000 0.0415002</td>
</tr>
<tr>
<td>0.34000000 0.16000000 0.89000000-0.0959958 0.2404028 0.0415002 1.00000000</td>
</tr>
<tr>
<td>EIGENVALUES, SUM AND PRODUCT OF EIGENVALUES</td>
</tr>
<tr>
<td>2.6161856 1.5874846 1.1577154 1.0120181 0.468504 0.0975220 0.0604329</td>
</tr>
<tr>
<td>7.0000000 0.0134378</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table-3. Sample Output Correlation Matrix and its Eigenvalues</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00000000-0.5000000 0.50000000-0.5000000 0.56000000 0.21000000 0.34000000</td>
</tr>
<tr>
<td>-0.5000000 1.00000000-0.0784965-0.0162682-0.325212 0.30000000 0.16000000</td>
</tr>
<tr>
<td>0.50000000 0.0784965 1.00000000-0.1237942 0.1573758 0.0478572 0.89000000</td>
</tr>
<tr>
<td>-0.50000000-0.0162682-0.1237942 1.00000000 0.0030405 0.0628510 0.0986661</td>
</tr>
<tr>
<td>0.56000000-0.325212 0.1573758-0.0030405 1.00000000 0.0528261 0.0727079</td>
</tr>
<tr>
<td>0.21000000 0.30000000 0.0478572 0.0628510 0.0528261 1.00000000-0.0683791</td>
</tr>
<tr>
<td>0.34000000 0.16000000 0.89000000-0.0986661 0.0727079-0.0683791 1.00000000</td>
</tr>
<tr>
<td>EIGENVALUES, SUM AND PRODUCT OF EIGENVALUES</td>
</tr>
<tr>
<td>2.4707041 1.6609468 1.1648713 1.0422329 0.5538390 0.0926969 0.0147091</td>
</tr>
<tr>
<td>7.0000000 0.0037623</td>
</tr>
</tbody>
</table>
Conclusion: The method (and the program) given here has an advantage over other algorithms due to its ability to present a scenario of valid correlation matrices that might be obtained from a given incomplete matrix of an arbitrary order. The analyst may choose some particular matrices, most suitable to his purpose, from among those output matrices. Further, unlike other methods, it has no restriction on the distribution of holes over the entire matrix, nor the analyst has to interactively feed elements of the matrix sequentially (as in Budden’s scheme) which might be quite inconvenient for larger matrices. It is flexible and by merely choosing larger population size (N) one might obtain a more exhaustive scenario of valid matrices. As the number of holes increases, the program takes longer time no doubt, but for smaller number of holes it takes a small time even if the input matrix is quite large. This is a special advantage of this method.

References


MAIN PROGRAM: GENERATE A SEMIPOSITIVE CORRELATION MATRIX FROM
A GIVEN CORRELATION MATRIX WITH SOME KNOWN ELEMENTS

METHOD: DIFFERENTIAL EVOLUTION

ADJUST THE PARAMETERS SUITABLY IN SUBROUTINES DE
WHEN THE PROGRAM ASKS FOR PARAMETERS, FEED THEM SUITABLY

--- MAIN PROGRAM ---

IMPLICIT DOUBLE PRECISION (A-H, O-Z)
COMMON /KF,KF,KF,NECALL,FTIT
CHARACTER *30 METHOD(1),MATFILE
CHARACTER *1 PROCEED
CHARACTER *70 FTIT,INFILE,OUTFIL
COMMON /XBAS/XBAS
COMMON /RNDM/IU,IV ! RANDOM NUMBER GENERATION (IU = 4-DIGIT SEED)
COMMON /NEAREST/2,MORDER
DIMENSION Z(10,10) ! THE INPUT CORRELATION MATRIX
INTEGER IU,IV
DIMENSION XX(3,50),KKF(3),MM(3),FMINN(3),XBAS(500,50)
DIMENSION X(50) ! X IS THE DECISION VARIABLE X IN F(X) TO MINIMIZE
DIMENSION XORDER(10,50) ! RANDOM NUMBER GENERATION (IU = 4-DIGIT SEED)

FMIN IS THE MIN VALUE OF F(X) OBTAINED FROM DE
FMINN IS THE MIN VALUE OF F(X) OBTAINED FROM DE

WRITE(*,*)'======= OPTIMIZATION BY DIFFERENTIAL EVOLUTION ========'

WRITE(*,*)'ORDER OF INPUT MATRIX (MORDER) & NAME OF INPUT FILE ?'
READ(*,*) MORDER,INFILE
WRITE(*,*)'SPECIFY THE OUTPUT FILE TO STORE VALID OUTPUT MATRICES'
READ(*,*) OUTFIL
WRITE(*,*)'READ THE GIVEN CORRELATION MATRIX (UPPER DIAGONAL ONLY)
OPEN(7,FILE=INFILE) ! OPEN INPUT FILE AND READ THE MATRIX
DO I=1,MORDER
READ(7,*) (I,J),J=I,MORDER
ENDDO
CLOSE(7)

WRITE(*,*)'********** CONSTRUCTION OF VALID CORRELATION MATRIX ====='
WRITE(*,*)'********** OPTIMIZATION BY DIFFERENTIAL EVOLUTION ======='

WRITE(*,*)'ORDER OF INPUT MATRIX (MORDER) & NAME OF INPUT FILE ?'
READ(*,*) MORDER,INFILE
WRITE(*,*)'SPECIFY THE OUTPUT FILE TO STORE VALID OUTPUT MATRICES'
READ(*,*) OUTFIL
WRITE(*,*)'READ THE GIVEN CORRELATION MATRIX (UPPER DIAGONAL ONLY)
OPEN(7,FILE=INFILE) ! OPEN INPUT FILE AND READ THE MATRIX
DO I=1,MORDER
READ(7,*) (I,J),J=I,MORDER
ENDDO
CLOSE(7)

WRITE(*,*)'********** CONSTRUCTION OF VALID CORRELATION MATRIX ====='
WRITE(*,*)'********** OPTIMIZATION BY DIFFERENTIAL EVOLUTION ======='

WRITE(*,*)'ORDER OF INPUT MATRIX (MORDER) & NAME OF INPUT FILE ?'
READ(*,*) MORDER,INFILE
WRITE(*,*)'SPECIFY THE OUTPUT FILE TO STORE VALID OUTPUT MATRICES'
READ(*,*) OUTFIL
WRITE(*,*)'READ THE GIVEN CORRELATION MATRIX (UPPER DIAGONAL ONLY)
OPEN(7,FILE=INFILE) ! OPEN INPUT FILE AND READ THE MATRIX
DO I=1,MORDER
READ(7,*) (I,J),J=I,MORDER
ENDDO
CLOSE(7)

WRITE(*,*)'********** CONSTRUCTION OF VALID CORRELATION MATRIX ====='
WRITE(*,*)'********** OPTIMIZATION BY DIFFERENTIAL EVOLUTION ======='

WRITE(*,*)'ORDER OF INPUT MATRIX (MORDER) & NAME OF INPUT FILE ?'
READ(*,*) MORDER,INFILE
WRITE(*,*)'SPECIFY THE OUTPUT FILE TO STORE VALID OUTPUT MATRICES'
READ(*,*) OUTFIL
WRITE(*,*)'READ THE GIVEN CORRELATION MATRIX (UPPER DIAGONAL ONLY)
OPEN(7,FILE=INFILE) ! OPEN INPUT FILE AND READ THE MATRIX
DO I=1,MORDER
READ(7,*) (I,J),J=I,MORDER
ENDDO
CLOSE(7)

WRITE(*,*)'********** CONSTRUCTION OF VALID CORRELATION MATRIX ====='
WRITE(*,*)'********** OPTIMIZATION BY DIFFERENTIAL EVOLUTION ======='

WRITE(*,*)'ORDER OF INPUT MATRIX (MORDER) & NAME OF INPUT FILE ?'
READ(*,*) MORDER,INFILE
WRITE(*,*)'SPECIFY THE OUTPUT FILE TO STORE VALID OUTPUT MATRICES'
READ(*,*) OUTFIL
WRITE(*,*)'READ THE GIVEN CORRELATION MATRIX (UPPER DIAGONAL ONLY)
OPEN(7,FILE=INFILE) ! OPEN INPUT FILE AND READ THE MATRIX
DO I=1,MORDER
READ(7,*) (I,J),J=I,MORDER
ENDDO
CLOSE(7)

WRITE(*,*)'********** CONSTRUCTION OF VALID CORRELATION MATRIX ====='
WRITE(*,*)'********** OPTIMIZATION BY DIFFERENTIAL EVOLUTION ======='

WRITE(*,*)'ORDER OF INPUT MATRIX (MORDER) & NAME OF INPUT FILE ?'
READ(*,*) MORDER,INFILE
WRITE(*,*)'SPECIFY THE OUTPUT FILE TO STORE VALID OUTPUT MATRICES'
READ(*,*) OUTFIL
WRITE(*,*)'READ THE GIVEN CORRELATION MATRIX (UPPER DIAGONAL ONLY)
OPEN(7,FILE=INFILE) ! OPEN INPUT FILE AND READ THE MATRIX
DO I=1,MORDER
READ(7,*) (I,J),J=I,MORDER
ENDDO
CLOSE(7)

WRITE(*,*)'********** CONSTRUCTION OF VALID CORRELATION MATRIX ====='
WRITE(*,*)'********** OPTIMIZATION BY DIFFERENTIAL EVOLUTION ======='

WRITE(*,*)'ORDER OF INPUT MATRIX (MORDER) & NAME OF INPUT FILE ?'
READ(*,*) MORDER,INFILE
WRITE(*,*)'SPECIFY THE OUTPUT FILE TO STORE VALID OUTPUT MATRICES'
READ(*,*) OUTFIL
WRITE(*,*)'READ THE GIVEN CORRELATION MATRIX (UPPER DIAGONAL ONLY)
OPEN(7,FILE=INFILE) ! OPEN INPUT FILE AND READ THE MATRIX
DO I=1,MORDER
READ(7,*) (I,J),J=I,MORDER
ENDDO
CLOSE(7)

WRITE(*,*)'********** CONSTRUCTION OF VALID CORRELATION MATRIX ====='
WRITE(*,*)'********** OPTIMIZATION BY DIFFERENTIAL EVOLUTION ======='

WRITE(*,*)'ORDER OF INPUT MATRIX (MORDER) & NAME OF INPUT FILE ?'
READ(*,*) MORDER,INFILE
WRITE(*,*)'SPECIFY THE OUTPUT FILE TO STORE VALID OUTPUT MATRICES'
READ(*,*) OUTFIL
WRITE(*,*)'READ THE GIVEN CORRELATION MATRIX (UPPER DIAGONAL ONLY)
OPEN(7,FILE=INFILE) ! OPEN INPUT FILE AND READ THE MATRIX
DO I=1,MORDER
READ(7,*) (I,J),J=I,MORDER
ENDDO
CLOSE(7)

WRITE(*,*)'********** CONSTRUCTION OF VALID CORRELATION MATRIX ====='
WRITE(*,*)'********** OPTIMIZATION BY DIFFERENTIAL EVOLUTION ======='

WRITE(*,*)'ORDER OF INPUT MATRIX (MORDER) & NAME OF INPUT FILE ?'
READ(*,*) MORDER,INFILE
WRITE(*,*)'SPECIFY THE OUTPUT FILE TO STORE VALID OUTPUT MATRICES'
READ(*,*) OUTFIL
WRITE(*,*)'READ THE GIVEN CORRELATION MATRIX (UPPER DIAGONAL ONLY)
OPEN(7,FILE=INFILE) ! OPEN INPUT FILE AND READ THE MATRIX
DO I=1,MORDER
READ(7,*) (I,J),J=I,MORDER
ENDDO
CLOSE(7)

WRITE(*,*)'********** CONSTRUCTION OF VALID CORRELATION MATRIX ====='
WRITE(*,*)'********** OPTIMIZATION BY DIFFERENTIAL EVOLUTION ======='

WRITE(*,*)'ORDER OF INPUT MATRIX (MORDER) & NAME OF INPUT FILE ?'
READ(*,*) MORDER,INFILE
WRITE(*,*)'SPECIFY THE OUTPUT FILE TO STORE VALID OUTPUT MATRICES'
READ(*,*) OUTFIL
WRITE(*,*)'READ THE GIVEN CORRELATION MATRIX (UPPER DIAGONAL ONLY)
OPEN(7,FILE=INFILE) ! OPEN INPUT FILE AND READ THE MATRIX
DO I=1,MORDER
READ(7,*) (I,J),J=I,MORDER
ENDDO
CLOSE(7)

WRITE(*,*)'********** CONSTRUCTION OF VALID CORRELATION MATRIX ====='
WRITE(*,*)'********** OPTIMIZATION BY DIFFERENTIAL EVOLUTION ======='

WRITE(*,*)'ORDER OF INPUT MATRIX (MORDER) & NAME OF INPUT FILE ?'
READ(*,*) MORDER,INFILE
WRITE(*,*)'SPECIFY THE OUTPUT FILE TO STORE VALID OUTPUT MATRICES'
READ(*,*) OUTFIL
WRITE(*,*)'READ THE GIVEN CORRELATION MATRIX (UPPER DIAGONAL ONLY)
OPEN(7,FILE=INFILE) ! OPEN INPUT FILE AND READ THE MATRIX
DO I=1,MORDER
READ(7,*) (I,J),J=I,MORDER
ENDDO
CLOSE(7)

WRITE(*,*)'********** CONSTRUCTION OF VALID CORRELATION MATRIX ====='
WRITE(*,*)'********** OPTIMIZATION BY DIFFERENTIAL EVOLUTION ======='

WRITE(*,*)'ORDER OF INPUT MATRIX (MORDER) & NAME OF INPUT FILE ?'
READ(*,*) MORDER,INFILE
WRITE(*,*)'SPECIFY THE OUTPUT FILE TO STORE VALID OUTPUT MATRICES'
READ(*,*) OUTFIL
WRITE(*,*)'READ THE GIVEN CORRELATION MATRIX (UPPER DIAGONAL ONLY)
OPEN(7,FILE=INFILE) ! OPEN INPUT FILE AND READ THE MATRIX
DO I=1,MORDER
READ(7,*) (I,J),J=I,MORDER
ENDDO
CLOSE(7)

WRITE(*,*)'********** CONSTRUCTION OF VALID CORRELATION MATRIX ====='
WRITE(*,*)'********** OPTIMIZATION BY DIFFERENTIAL EVOLUTION ======='

WRITE(*,*)'ORDER OF INPUT MATRIX (MORDER) & NAME OF INPUT FILE ?'
READ(*,*) MORDER,INFILE
WRITE(*,*)'SPECIFY THE OUTPUT FILE TO STORE VALID OUTPUT MATRICES'
READ(*,*) OUTFIL
WRITE(*,*)'READ THE GIVEN CORRELATION MATRIX (UPPER DIAGONAL ONLY)
OPEN(7,FILE=INFILE) ! OPEN INPUT FILE AND READ THE MATRIX
DO I=1,MORDER
READ(7,*) (I,J),J=I,MORDER
ENDDO
CLOSE(7)
SUBROUTINE DE(M,A,FBEST,G0,G1)

PROGRAM "DIFFERENTIAL EVOLUTION ALGORITHM" OF GLOBAL OPTIMIZATION
This method was proposed by R. Storn and K. Price in 1995. Ref --
"DIFFERENTIAL EVOLUTION - A SIMPLE AND EFFICIENT ADAPTIVE SCHEME
FOR GLOBAL OPTIMIZATION OVER CONTINUOUS SPACES" : TECHNICAL REPORT
PROGRAM BY SK MISHRA, DEPT. OF ECONOMICS, NEHU, SHILLONG (INDIA)
-------------------------------------------------------------------
C     RANDOM NUMBERS BETWEEN -RRANGE AND +RRANGE (BOTH EXCLUSIVE)
C     CALL RANDOM(RAND) ! GENERATES INITIATION X WITHIN
C     INITIALIZATION: GENERATE X(N,M) RANDOMLY
C     READ(*,*) PCROS,FACT
C     SPECIFY OTHER PARAMETERS ---------------------------------------
C     ------- SELECT THE FUNCTION TO MINIMIZE AND ITS DIMENSION -------
C     ----------------------------------------------------------------
C     (7) RANDOM NUMBER SEED (4 DIGITS INTEGER)
C     (6) ITER = MAXIMUM NUMBER OF ITERATIONS PERMITTED (5000 OR MORE)
C     (5) FACT = SCALE (SUGGESTED 0.5 TO .95 OR 1, ETC);
C     (4) PCROS = PROB. OF CROSS-OVER (SUGGESTED: ABOUT 0.85 TO .99);
C         FOR SMALLER PROBLEMS N=100 WORKS VERY WELL);
C     (3) N=POPULATION SIZE (SUGGESTED 10 TIMES OF NO. OF VARIABLES, M,
C     (1) FUNCTION CODE (KF), (2) NO. OF VARIABLES IN THE FUNCTION (M);
C     THE PROGRAM REQUIRES INPUTS FROM THE USER ON THE FOLLOWING ------
C     -----------------------------------------------------------------
C     ULTIMATELY "DID NOT CONVERGE" IS REPORTED.
C     WOULD BE UNDERGONE EVEN IF NO IMPROVEMENT IN RESULTS IS THERE.
C     IT PRINTS THE INTERMEDIATE RESULTS AFTER EACH IPRINT ITERATION AND
C     PARAMETER(NCROSS=2) ! CROSS-OVER SCHEME (NCROSS <=0 OR =1 OR =>2)
C     SCHEME 3. THUS RX1 = .2 AND RX2 = .8 MEANS 0-20% SCHEME1, 20 TO 80
C     SCHEME 2; RX2 IS THE UPPER LIMIT OF SCHEME 2 AND LOWER LIMIT OF
C     RX1 DETERMINES THE UPPER LIMIT OF SCHEME 1 (AND LOWER LIMIT OF
C     PROGRAM DE
C     -----------------------------------------------------------------
C     FOR GLOBAL OPTIMIZATION OVER CONTINUOUS SPACES" : TECHNICAL REPORT
C     "DIFFERENTIAL EVOLUTION - A SIMPLE AND EFFICIENT ADAPTIVE SCHEME
C     THIS METHOD WAS PROPOSED BY R. STORN AND K. PRICE IN 1995. REF --
C     PROGRAM: "DIFFERENTIAL EVOLUTION ALGORITHM" OF GLOBAL OPTIMIZATION

PARAMETER(NMAX=500,MMAX=50)! MAXIMUM DIMENSION PARAMETERS
PARAMETER(RX1=1.0, RX2=0.0)! TO BE ADJUSTED SUITABLY, IF NEEDED
RX1 DETERMINES THE UPPER LIMIT OF SCHEME 1 (AND LOWER LIMIT OF
SCHEME 2; RX2 IS THE UPPER LIMIT OF SCHEME 2 AND LOWER LIMIT OF
SCHEME 3. THUS RX1 = .2 AND RX2 = .8 MEANS 0-20% SCHEME1, 20 TO 80
PERCENT SCHEME 2 AND THE REST (80 TO 100 %) SCHEME 3.
PARAMETER(NCROSS=2)! CROSS-OVER SCHEME (NCROSS <=0 OR =1 OR =>2)
PARAMETER(IPRINT=500,EPS=1.D-08)! FOR WATCHING INTERMEDIATE RESULTS
IT PRINTS THE INTERMEDIATE RESULTS AFTER EACH IPRINT ITERATION AND
EPS DETERMINES ACCURACY FOR TERMINATION. IF EPS= 0, ALL ITERATIONS
WOULD BE UNDERGONE EVEN IF NO IMPROVEMENT IN RESULTS IS THERE.
ULTIMATELY "DID NOT CONVERGE" IS REPORTED.
COMMON /RNDM/IU,IV ! RANDOM NUMBER GENERATION (IU = 4-DIGIT SEED)
COMMON /KFF/KF,NCALL,FTIT ! FUNCTION CODE, NO. OF CALLS * TITLE
COMMON /XBASE/XBAS
CHARACTER *70 FTIT ! TITLE OF THE FUNCTION
CHARACTER *15 CFIL ! OUTPUT FILE
THE PROGRAM REQUIRES INPUTS FROM THE USER ON THE FOLLOWING -------
(1) FUNCTION CODE (KF), (2) NO. OF VARIABLES IN THE FUNCTION (M);
N=POPULATION SIZE (SUGGESTED 10 TIMES OF NO. OF VARIABLES, M,
FOR SMALLER PROBLEMS N=100 WORKS VERY WELL));
PCROS = PROB. OF CROSS-OVER (SUGGESTED : ABOUT 0.85 TO .99);
FACT = SCALE (SUGGESTED 0.5 TO .95 OR 1, ETC);
ITER = MAXIMUM NUMBER OF ITERATIONS PERMITTED (5000 OR MORE)
RANDOM NUMBER SEED (4 DIGITS INTEGER)
DIMENSION X(NMAX,MMAX),Y(NMAX,MMAX),A(MMAX),FV(NMAX)
DIMENSION IR(3),XBASE(500,50)
SELECT THE FUNCTION TO MINIMIZE AND ITS DIMENSION -------
CALL FSELECT(KF,M,FTIT)
CFIL='CORRESULTS'! IT IS AN INTERMEDIATE FILE
SPECIFY OTHER PARAMETERS -----------------------------------------------
WRITE(*,*)'POPULATION SIZE [N] AND NO. OF ITERATIONS [ITER] ?'
WRITE(*,*)'SUGGESTED: N=>100 OR =>10.M; ITERATION 500 OR LARGER'
READ(*,*) N,ITER
WRITE(*,*)'CROSSOVER PROBABILITY [PCROS] AND SCALE [FACT] ?'
WRITE(*,*)'SUGGESTED: PCROS ABOUT 0.9; FACT=.5 OR LARGER BUT <=1'
READ(*,*) PCROS,FACT
PCROS=0.9
FACT=0.5
WRITE(*,*)'RANDOM NUMBER SEED ?'
WRITE(*,*)'A FOUR-DIGIT POSITIVE ODD INTEGER, SAY, 1171'
READ(*,*) IU
NFCALL=0 ! INITIALIZE COUNTER FOR FUNCTION CALLS
GBEST=1.D30! TO BE USED FOR TERMINATION CRITERION
INITIALIZATION : GENERATE X(N,M) RANDOMLY
DO I=1,N
DO J=1,M
CALL RANDOM(RAND) ! GENERATES INITIATION X WITHIN
X(I,J)=(RNDM-.5000)*2000 ! GENERATES INITIATION X WITHIN
RANDOM NUMBERS BETWEEN -RRANGE AND +RRANGE (BOTH EXCLUSIVE)
X(I,J)=XBASE(I,J) TAKES THESE NUMBERS FROM THE MAIN PROGRAM
ENDDO
ENDDO
WRITE(*,*) 'COMPUTING --- PLEASE WAIT '
IPCOUNT=0
DO 100 ITR=1,ITER  ! ITERATION BEGINS
           ! EVALUATE ALL X FOR THE GIVEN FUNCTION
   DO I=1,N
   DO J=1,M
      A(J)=X(I,J)
   ENDDO
   CALL FUNC(A,M,F)
   C     STORE FUNCTION VALUES IN FV VECTOR
   FV(I)=F
   ENDDO

   C     ----------------------------------------------------------------
   C     FIND THE FITTEST (BEST) INDIVIDUAL AT THIS ITERATION
   FBEST=FV(1)
   KB=1
   DO IB=2,N
      IF(FV(IB).LT.FBEST) THEN
         FBEST=FV(IB)
         KB=IB
      ENDIF
   ENDDO
   C     BEST FITNESS VALUE = FBEST : INDIVIDUAL X(KB)
   C     ----------------------------------------------------------------
   C     GENERATE OFFSPRINGS
   DO I=1,N  ! I LOOP BEGINS
      C     INITIALIZE CHILDREN IDENTICAL TO PARENTS; THEY WILL CHANGE LATER
      DO J=1,M
         Y(I,J)=X(I,J)
      ENDDO
      C     SELECT RANDOMLY THREE OTHER INDIVIDUALS
      20   DO IRI=1,3  ! IRI LOOP BEGINS
      IR(IRI)=0
      CALL RANDOM(RAND)
      IRJ=INT(RAND*N)+1
      C     CHECK THAT THESE THREE INDIVIDUALS ARE DISTINCT AND OTHER THAN I
      IF(IRI.EQ.1.AND.IRJ.NE.1) THEN
         IR(IRI)=IRJ
      ENDIF
      IF(IRI.EQ.2.AND.IRJ.NE.1.AND.IRJ.NE.IR(1)) THEN
         IR(IRI)=IRJ
      ENDIF
      IF(IRI.EQ.3.AND.IRJ.NE.1.AND.IRJ.NE.IR(1).AND.IRJ.NE.IR(2)) THEN
         IR(IRI)=IRJ
      ENDIF
      ENDDO  ! IRI LOOP ENDS
      C     CHECK IF ALL THE THREE IR ARE POSITIVE (INTEGERS)
      DO IX=1,3
         IF(IR(IX).LE.0) THEN
            GOTO 20  ! IF NOT THEN REGENERATE
         ENDIF
      ENDDO
      C     THREE RANDOMLY CHOSEN INDIVIDUALS DIFFERENT FROM I AND DIFFERENT
      FROM EACH OTHER ARE IR(1),IR(2) AND IR(3)
      NCROSS=0
      CALL RANDOM(RAND)
      IF(RAND.GT.RX1) NCROSS=1  ! IF RX1=>1, SCHEME 2 NEVER IMPLEMENTED
      IF(RAND.GT.RX2) NCROSS=2  ! IF RX2=>1, SCHEME 3 NEVER IMPLEMENTED
      C     ---------------------- SCHEME 1 ----------------------------------
C  NO CROSS OVER, ONLY REPLACEMENT THAT IS PROBABILISTIC
202:  C IF (NCCROSS.LE.0) THEN
203:      IF (J-1,M) DO J=1,M END DO ! J LOOP BEGINS
204:     CALL RANDOM(RAND)
205:     IF (RAND.LE.PCROS) THEN ! REPLACE IF RAND < PCROS
206:         A(J)=X(IR(1),J)+X(IR(2),J)-X(IR(3),J))*FACT ! CANDIDATE CHILD
207:     ENDIF
208: ENDIF
209: ENDDO ! J LOOP ENDS
210: ENDDO
211: ENDDO
212: C --------------------------------------------------------------------
213: C PERSONAL LETTER TO THE AUTHOR (DATED OCTOBER 18, 2006)
214: C CROSSOVER SCHEME (NEW) SUGGESTED BY KENNETH PRICE IN HIS
215: C ESPECIALLY SUITABLE TO NON-DECOMPOSABLE (NON-SEPERABLE) FUNCTIONS
216: C ------------------------ SCHEME 3 --------------------------------
217: C CROSSOVER SCHEME (EXPONENTIAL) SUGGESTED BY KENNETH PRICE IN HIS
218: C THE STANDARD CROSSOVER SCHEME
219: C NO CROSS OVER, ONLY REPLACEMENT THAT IS PROBABILISTIC
220: C PERSONAL LETTER TO THE AUTHOR (DATED SEPTEMBER 29, 2006)
221: C ESPECIALLY SUITABLE TO NON-DECOMPOSABLE (NON-SEPERABLE) FUNCTIONS
222: C CROSSOVER SCHEME (NEW) SUGGESTED BY KENNETH PRICE IN HIS
223: C PERSONAL LETTER TO THE AUTHOR (DATED OCTOBER 18, 2006)
224: IF (NCCROSS.EQ.1) THEN
225:     CALL RANDOM(RAND)
226:     JR=INT(RAND*M)+1
227:     J=JR
228:     A(J)=X(IR(1),J)+X(IR(2),J)-X(IR(3),J))
229:     IF (J-GT.M) J=1
230:     4 IF (J.EQ.JR) GOTO 10
231:     IF (J.EQ.JR) GOTO 10
232:     CALL RANDOM(RAND)
233:     IF (PCROS.LE.RAND) GOTO 2
234:     6 A(J)=X(I,J)
235:     7 J=J+1
236:     IF (J-GT.M) J=1
237:     8 IF (J.EQ.JR) GOTO 10
238:     9 GOTO 6
239: 10 CONTINUE
240: ENDF
241: C --------------------------------------------------------------------
242: C ESPECIALLY SUITABLE TO NON-DECOMPOSABLE (NON-SEPERABLE) FUNCTIONS
243: C CROSSOVER SCHEME (NEW) SUGGESTED BY KENNETH PRICE IN HIS
244: C PERSONAL LETTER TO THE AUTHOR (DATED SEPTEMBER 29, 2006)
245: C ESPECIALLY SUITABLE TO NON-DECOMPOSABLE (NON-SEPERABLE) FUNCTIONS
246: C CROSSOVER SCHEME (NEW) SUGGESTED BY KENNETH PRICE IN HIS
247: C PERSONAL LETTER TO THE AUTHOR (DATED OCTOBER 18, 2006)
248: IF (NCCROSS.GE.2) THEN
249:     CALL RANDOM(RAND)
250:     IF (RAND.LE.PCROS) THEN
251:         CALL NORMAL(RN)
252:         DO J=1,M
253:             A(J)=X(I,J)+X(IR(1),J)+X(IR(2),J)-2*X(I,J)*RN
254:         ENDDO
255: ELSE
256: DO J=1,M
257:     A(J)=X(I,J)+X(IR(1),J)-X(IR(2),J))
258: ENDIF
259: ENDDO
260: ENDF
261: C --------------------------------------------------------------------
262: C EVALUATE THE OFFSPRING
263: IF (F.LT.FV(I)) THEN ! IF BETTER, REPLACE PARENTS BY THE CHILD
264:     FV(I)=F
265:     DO J=1,M
266:         Y(I,J)=A(J)
267:     ENDDO
268: ENDIF
269: ENDDO ! I LOOP ENDS
270: DO I=1,N
271: DO J=1,M
272: X(I,J)=Y(I,J) ! NEW GENERATION IS A MIX OF BETTER PARENTS AND
273: C BETTER CHILDREN
274: ENDDO
275: ENDDO
276: IF (IPCOUNT-IPCOUNT+1
277: IF (IPCOUNT.EQ.IPRINT) THEN
278: DO J=1,M
279:     A(J)=X(KB,J)
ENDDO
WRITE(*,*)'(X(KB,J),J=1,M), FBEST UPTO NOW = ',FBEST
WRITE(*,*)' TOTAL NUMBER OF FUNCTION CALLS =',NFCALL
IF(DABS(FBEST-GBEST).LT.EPS) THEN
   WRITE(*,*)' COMPUTATION OVER'
   GOTO 999
ELSE
   GBEST=FBEST
ENDIF
IPCOUNT=0
ENDIF

100 ENDDO  ! ITERATION ENDS : GO FOR NEXT ITERATION, IF APPLICABLE
C
WRITE(*,*)'DID NOT CONVERGE. REDUCE EPS OR RAISE ITER OR DO BOTH'
WRITE(*,*)'INCREASE N, PCROS, OR SCALE FACTOR (FACT)'
999 OPEN(7,FILE=CFIL)
WRITE(*,*)' N
DO I=1,N
WRITE(*,*)'(X(I,J),J=1,M),FV(I)
ENDDO
CLOSE(7)
RETURN
END

COMMON /RNDM/IU,IV
INTEGER IU,IV

CALL RANDOM(RAND) ! INVOKES RANDOM TO GENERATE UNIFORM RAND [0, 1]
U1=RAND ! U1 IS UNIFORMLY DISTRIBUTED [0, 1]
CALL RANDOM(RAND) ! INVOKES RANDOM TO GENERATE UNIFORM RAND [0, 1]
U2=RAND ! U1 IS UNIFORMLY DISTRIBUTED [0, 1]
R=DLOG(-2.0D0*DLOG(U1))
R=R*DOSR(U2*6.283185307179586476925286766559D00)
RETURN
END

RANDOM NUMBER GENERATOR (UNIFORM BETWEEN 0 AND 1 - BOTH EXCLUSIVE)

SUBROUTINE RANDOM(RAND1)
COMMON /RNDM/IU,IV
INTEGER IU,IV
RAND=REAL(RAND1)
IV=IU*65539
IF(IV.LT.0) THEN
   IV=IV+2147483647+1
ENDIF
RAND=IV
IU=IV
RAND=RAND*0.4656613E-09
RAND1= RAND
SUBROUTINE FSELECT(KF,M,FTIT)

PARAMETER (NFUNCT=1) ! NO. OF FUNCTIONS IN THE LIST

THE PROGRAM REQUIRES INPUTS FROM THE USER ON THE FOLLOWING -------

(1) FUNCTION CODE (KF), (2) NO. OF VARIABLES IN THE FUNCTION (M);

CHARACTER *70 TIT(100),FTIT

KF=1, I NO. OF FUNCTIONS

DATA TIT(1)/'KF=1 CONSTRUCT CORRELATION MATRIX:M-VARIABLES M=?'/

C     ------------------------------------------------------------

DO I=1,NFUNCT
  WRITE(*,*)TIT(I)
ENDDO

WRITE(*,*)'*****************************************************'
WRITE(*,*)'NO. OF VARIABLES=UNKNOWN CORRELATION COEFFICIENTS [M]?'
READ(*,*)M
FTIT=TIT(KF) ! STORE THE NAME OF THE CHOSEN FUNCTION IN FTIT
RETURN
END

C     ------------------------------------------------------------

SUBROUTINE FUNC(X,M,F)

TEST FUNCTIONS FOR GLOBAL OPTIMIZATION PROGRAM

IMPLICIT DOUBLE PRECISION (A-H,O-Z)

COMMON /RNDM/IU,IV
COMMON /KFF/KF,NFCALL,FTIT
INTEGER IU,IV

DIMENSION X(*)
CHARACTER*70 FTIT

NFCALL=NFCALL+1 ! INCREMENT TO NUMBER OF FUNCTION CALLS

C     KF IS THE CODE OF THE TEST FUNCTION

IF(KF.EQ.1) THEN
  CONSTRUCT CORRELATION MATRIX
  CALL CONCOR(M,X,F)
  RETURN
ENDIF

WRITE(*,*)'FUNCTION NOT DEFINED. PROGRAM ABORTED'
STOP
END

C     ------------------------------------------------------------

SUBROUTINE EIGEN(A,N,V,W)

COMPUTES EIGENVALUES AND VECTORS OF A REAL SYMMETRIC MATRIX

A(N,N) =GIVEN REAL SYMMETRIC MATRIX WHOSE EIGENVALUES AND VECTORS
ARE BE FOUND. ITS ORDER IS N X N
W(N,N) CONTAINS EIGENVALUES IN ITS MAIN DIAGONAL. OTHER ELEMENTS=0
V(N,N) CONTAINS EIGENVECTORS

DOUBLE PRECISION A(10,10),V(10,10),W(10,10),P(10)
DOUBLE PRECISION PMAX,EPLN,TAN,SIN,COS,AI,TT,TA,GB

DIMENSION MM(10)

DO I=1,N
  DO J=1,N
    V(I,J)=0.0
    W(I,J)=A(I,J)
  ENDDO
  P(I)=0.0
ENDDO
PMAX=0
EPLN=0
TAN=0
SIN=0
COS=0

C     ------------------------------------------------------------

RETURN
END
402:     AI=0
403:     TT=0
404:     NN=1
405:     EPLN=1.0D-100
406:     C
407:     IF(NN.NE.0) THEN
408:     DO I=1,N
409:     DO J=1,N
410:     V(I,J)=0.0
411:     IF(I.EQ.J) V(I,J)=1.0
412:     ENDDO
413:     ENDDO
414:     ENDF
415:     2 NR=0
416:     5 MI=N-1
417:     DO I=1,MI
418:     P(I)=0.0
419:     MJ=I+1
420:     DO J=MJ,N
421:     IF(P(I).LE.DABS(A(I,J))) THEN
422:     P(I)=DABS(A(I,J))
423:     MM(I)=J
424:     ENDF
425:     ENDDO
426:     ENDDO
427:     7 DO 8 I=1,MI
428:     IF(I.LE.I) GOTO 10
429:     IF(PMAX.GT.P(I)) GOTO 8
430:     10 PMAX=P(I)
431:     IP=I
432:     JP=MM(I)
433:     8 CONTINUE
434:     C     EPLN=DABS(PMAX)*1.0D-09
435:     IF (PMAX.LE.EPLN) THEN
436:     C     WRITE(*,*)'PMAX EPLN',PMAX, EPLN
437:     C     PAUSE'CONVERGENCE CRITERION IS MET'
438:     C
439:     GO TO 12
440:     ENDIF
441:     NR=NR+1
442:     13 TA=2.0*A(IP,JP)
443:     TB=(DABS(A(IP,IP)-A(IP,JP)) +
444:     1DSQRT((A(IP,IP)-A(IP,JP))**2+4.0*A(IP,JP)**2))
445:     TAN=TA/TB
446:     IF(A(IP,IP).LT.A(IP,JP)) TAN=-TAN
447:     14 COS=1.0/DSQRT(1.0+TAN**2)
448:     SIN=TAN*COS
449:     AI=A(IP,IP)
450:     A(IP,IP)=(COS**2)*(AI+TAN*(2.0*A(IP,JP)+TAN*A(IP,JP)))+
452:     A(IP,JP)=0.0
453:     IF(A(IP,IP).GE.A(IP,JP)) GO TO 15
454:     TT=A(IP,IP)
455:     A(IP,IP)=A(IP,JP)
456:     A(IP,JP)=TT
457:     IF(SIN.GE.0) GO TO 16
458:     TT=COS
459:     GO TO 17
460:     16 TT=-COS
461:     17 COS=DABS(SIN)
462:     SIN=TT
463:     15 DO 18 I=1,MI
464:     IF(I-IPI) 19, 18, 20
465:     20 IF(I.EQ.JP) GO TO 18
466:     19 IF(MM(I).EQ.IP) GO TO 21
467:     IF(MM(I).NE.IP) GO TO 18
468:     K=MM(I)
469:   TT=A(I,K)
470:   A(I,K)=0.0
471:   MJ=I+1
472:   P(I)=0.0
473:   DO 22 J=MJ,N
474:   IF(P(I).GT.DABS(A(I,J))) GO TO 22
475:   P(I)=DABS(A(I,J))
476:   MM(I)=J
477:   22 CONTINUE
478:   A(I,K)=IT
479:   18 CONTINUE
480:   P(IP)=0.0
481:   P(JP)=0.0
482:   DO 23 I=1,N
483:   IF(I-IP) 24, 23, 25
484:   24 TT=A(I,IP)
485:   A(I,IP)=COS*TT+SIN*A(I,JP)
486:   IF(P(IP).GE.DABS(A(I,IP))) GO TO 26
487:   P(IP)=DABS(A(I,IP))
488:   MM(IP)=I
489:   26 A(I,JP)=-SIN*TT+COS*A(I,JP)
490:   IF(P(IP).GE.DABS(A(I,JP))) GO TO 23
491:   30 P(I)=DABS(A(I,JP))
492:   GO TO 23
493:   25 IF(I.LT.JP) GO TO 27
494:   IF(I.GT.JP) GO TO 28
495:   IF(I.EQ.JP) GO TO 23
497:   27 TT=A(IP,I)
498:   A(IP,I)=COS*TT+SIN*A(IP,I)
499:   IF(P(IP).GE.DABS(A(IP,I))) GO TO 29
500:   P(IP)=DABS(A(IP,I))
501:   MM(IP)=I
502:   29 A(IP,JP)=-TT*SIN+COS*A(IP,JP)
503:   IF(P(IP).GE.DABS(A(IP,JP))) GO TO 23
504:   GO TO 30
505:   28 TT=A(IP,I)
506:   A(IP,I)=TT*COS+SIN*A(IP,J)
507:   IF(P(IP).GE.DABS(A(IP,I))) GO TO 31
508:   P(IP)=DABS(A(IP,I))
509:   MM(IP)=I
510:   31 A(IP,JP)=-TT*SIN+COS*A(IP,J)
511:   IF(P(IP).GE.DABS(A(IP,J))) GO TO 23
512:   P(IP)=DABS(A(IP,J))
513:   MM(JP)=I
514:   23 CONTINUE
515:   IF(NN.EQ.0) GOTO 7
516:   DO 32 I=1,N
517:   TT=V(I,IP)
518:   V(I,IP)=TT*COS+TIN*V(I,JP)
519:   V(I,JP)=-TT*SIN+COS*V(I,JP)
520:   32 CONTINUE
521:   GO TO 7
522:   12 DO I=1,N
523:   P(I)=A(I,I)
524:   ENDDO
525:   DO I=1,N
526:   DO J=1,N
527:   A(I,J)=W(I,J)
528:   W(I,J)=0.DO
529:   ENDDO
530:   W(I,I)=P(I)
531:   ENDDO
532:   RETURN
533:   END
534:
535:
C SUBROUTINE CONCOR(M,X,F)  
C CONSTRUCTING VALID CORRELATION MATRICES  
C IMPLICIT DOUBLE PRECISION (A-H,O-Z)  
C COMMON /RNDM/IU,IV  
COMMON /NEAREST/Z,MORDER  
DIMENSION Z(10,10),A(10,10)  
INTEGER IU,IV  
DIMENSION X(*),V(10,10),W(10,10),AA(10,10),P(10)  
C CHECK THE NUMBER OF INVALID ELEMENTS  
C CONSTRUCTING VALID CORRELATION MATRICES  
MINVAL=0  
DO I=1,MORDER  
DO J=I,MORDER  
IF(DABS(Z(I,J)).GT.1.D0) MINVAL=MINVAL+1  
ENDDO  
ENDDO  
IF(M.NE.MINVAL) THEN  
WRITE(*,*)'THE VALUE OF M SHOULD BE=',MINVAL  
WRITE(*,*)'RERUN THE PROGRAM WITH M=', MINVAL  
STOP  
ENDIF  
C  
DO I=1,M  
IF(X(I).LT.-1.D0 .OR. X(I).GT.1.D0) THEN  
CALL RANDOM(RAND)  
X(I)=(RAND-.5)*2  
ENDIF  
ENDDO  
C  
DO I=1,MORDER  
A(I,I)=1.0  
DO J=I+1,MORDER  
IF(DABS(Z(I,J)).LT.1.D0) THEN  
ICOUNT=ICOUNT+1  
A(I,J)=X(ICOUNT)  
ELSE  
A(I,J)=Z(I,J)  
ENDIF  
ENDDO  
C  
WRITE(*,*)'CONSTRUCTED MATRIX'  
DO I=1,MORDER  
DO J=I,MORDER  
AA(I,J)=A(I,J) ! STORES THE MATRIX A IN AA  
ENDDO  
ENDDO  
C  
FORMAT(10F8.4)  
CALL EIGEN(A,MORDER,V,W)  
C  
CALL EIGEN(A,MORDER,V,W)  
STORE EIGENVALUES (DIAGONAL OF RETURNING W) INTO P  
P=0.D0  
PSUM=0.D0 ! SUM OF MAGNITUDE OF EIGENVALUES  
DO I=1,MORDER  
P(I)=W(I,I)  
IF(P(I).LT.0.D0) PSUM=PSUM+DABS(P(I))
603:     ENDDO
604:     PROD=1.D0
605:     DO I=1,MORDER
606:     IF(P(I),LT,0.D0) THEN
607:     F=F+P(I)**2
608:     PROD=PROD*P(I)
609:     ENDIF
610:     ENDDO
611:     IF(PROD.LT.0.D0.OR.PROD.GT.1.D0) F=(F+PSUM+PROD**2)**2
612:     RETURN
613:     END
614:     C -------------------------------
615:     SUBROUTINE NCORX(X,M,OUTFIL)
616:     C NEAREST CORRELATION MATRIX PROBLEM
617:     C IMPLICIT DOUBLE PRECISION (A-H,O-Z)
618:     COMMON /RNDM/IU,IV
619:     COMMON /NEAREST/Z,MORDER
620:     DIMENSION Z(10,10),X(*)
621:     INTEGER IU,IV
622:     DIMENSION A(10,10),V(10,10),W(10,10)
623:     CHARACTER *70 OUTFIL
624:     CHARACTER *15 CFIL
625:     C -------------------------------
626:     CFIL = 'CORRESULTS' ! IT IS AN INTERMEDIATE FILE
627:     C -------------------------------
628:     OPEN (7, FILE=CFIL) ! OPENS INTERMEDIATE FILE FOR INPUT
629:     OPEN (8, FILE=OUTFIL) ! OPENS OUTPUT FILE TO STORE VALID MATRICES
630:     C CONSTRUCT THE CORRELATION MATRIX
631:     READ(7,* ) N ! READS POPULATION SIZE FROM INTERMEDIATE FILE
632:     DO IC=1,N
633:     READ(7,* ) (X(J),J=1,M),VALU ! READS VECTOR FROM INTERMEDIATE FILE
634:     ICOUNT=0
635:     DO I=1,MORDER
636:     A(I,I)=1.0
637:     DO J=I+1,MORDER
638:     IF(DABS(Z(I,J)).GT.1.D0) THEN
639:     ICOUNT=ICOUNT+1
640:     A(I,J)=X(ICOUNT)
641:     ELSE
642:     A(I,J)=Z(I,J)
643:     ENDIF
644:     ENDDO
645:     ENDDO
646:     ENDDO
647:     C FILLING THE LOWER DIAGONAL
648:     DO I=1,MORDER
649:     DO J=1,I
650:     A(I,J)=A(J,I)
651:     ENDDO
652:     ENDDO
653:     WRITE(*,*) ',',
654:     WRITE(8,*)'********************************************************************'
655:     WRITE(8,*)'A VALID CORRELATION MATRIX'
656:     DO I=1,MORDER
657:     WRITE(8,*) (A(I,J),J=1,MORDER)
658:     ENDDO
659:     WRITE(*,*) ',',
660:     CALL EIGEN(A,MORDER,V,W)
661:     MSIGN=0
662:     SUMW=0.D0
663:     PROD=1.D0
664:     DO I=1,MORDER
665:     SUMW=SUMW+W(I,I)
666:     PROD=PROD*W(I,I)
667:     IF(W(I,I).LT.0) MSIGN=1
668:     ENDDO
669:     WRITE(8,*)'EIGENVALUES, SUM AND PRODUCT OF EIGENVALUES'
WRITE(8,1)(W(I,I),I=1,MORDER),SUMW,PROD
671: WRITE(8,*),'EIGENVECTORS '
672: DO I=1,MORDER
673: WRITE(8,1) (V(I,J),J=1,MORDER)
674: ENDDO
675: 1 FORMAT(8F10.7)
676: IF(MSIGN.EQ.1) THEN
677: WRITE(8,*),'FAILURE OF THE METHOD'
678: ELSE
679: WRITE(8,*),'SUCCESS OF THE METHOD'
680: ENDF
681: ENDDO
682: CLOSE(7)
683: CLOSE(8)
684: RETURN
685: END