NAG Library Function Document

nag_nearest_correlation_k_factor (g02aec)

1 Purpose

nag_nearest_correlation_k_factor (g02aec) computes the factor loading matrix associated with the nearest correlation matrix with \( k \)-factor structure, in the Frobenius norm, to a given square, input matrix.

2 Specification

```c
#include <nag.h>
#include <nagg02.h>

void nag_nearest_correlation_k_factor (Nag_OrderType order, double g[],
                Integer pdg, Integer n, Integer k, double errtol, Integer maxit,
                double x[], Integer pdx, Integer *iter, Integer *feval, double *nrmpgd,
                NagError *fail)
```

3 Description

A correlation matrix \( C \) with \( k \)-factor structure may be characterised as a real square matrix that is symmetric, has a unit diagonal, is positive semidefinite and can be written as 
\[
C = XX^T + \text{diag}(I - XX^T),
\]
where \( I \) is the identity matrix and \( X \) has \( n \) rows and \( k \) columns. \( X \) is often referred to as the factor loading matrix.

nag_nearest_correlation_k_factor (g02aec) applies a spectral projected gradient method to the modified problem 
\[
\min_{X} \|G - XX^T + \text{diag}(XX^T - I)\|_F,
\]
such that \( \|x_i^T\| \leq 1 \) for \( i = 1, 2, \ldots, n \), where \( x_i \) is the \( i \)th row of the factor loading matrix, \( X \), which gives us the solution.

4 References


5 Arguments

1: \( \text{order} \) – Nag_OrderType

*Input*

On entry: the \( \text{order} \) argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by \( \text{order} = \text{Nag_RowMajor} \). See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

Constraint: \( \text{order} = \text{Nag_RowMajor} \) or \( \text{Nag_ColMajor} \).

2: \( \text{g}[\text{dim}] \) – double

*Input/Output*

Note: the dimension, \( \text{dim} \), of the array \( \text{g} \) must be at least \( \text{pdg} \times \text{n} \).

The \((i,j)\)th element of the matrix \( G \) is stored in 
\[
\text{g}((j-1) \times \text{pdg} + i - 1) \quad \text{when } \text{order} = \text{Nag_ColMajor};
\]
\[
\text{g}((i-1) \times \text{pdg} + j - 1) \quad \text{when } \text{order} = \text{Nag_RowMajor}.
\]

On entry: \( G \), the initial matrix.
On exit: a symmetric matrix \( \frac{1}{2}(G + G^T) \) with the diagonal elements set to unity.

3: \textbf{pdg} – Integer \hspace{1cm} \textit{Input}

\textit{On entry}: the stride separating row or column elements (depending on the value of \texttt{order}) in the array \texttt{g}.

\textit{Constraint}: \texttt{pdg} \geq \texttt{n}.

4: \textbf{n} – Integer \hspace{1cm} \textit{Input}

\textit{On entry}: \texttt{n}, the order of the matrix \( G \).

\textit{Constraint}: \texttt{n} > 0.

5: \textbf{k} – Integer \hspace{1cm} \textit{Input}

\textit{On entry}: \texttt{k}, the number of factors and columns of \( X \).

\textit{Constraint}: 0 < \texttt{k} \leq \texttt{n}.

6: \textbf{errtol} – double \hspace{1cm} \textit{Input}

\textit{On entry}: the termination tolerance for the projected gradient norm. See references for further details. If \texttt{errtol} \leq 0.0 then 0.01 is used. This is often a suitable default value.

7: \textbf{maxit} – Integer \hspace{1cm} \textit{Input}

\textit{On entry}: specifies the maximum number of iterations in the spectral projected gradient method. If \texttt{maxit} \leq 0, 40000 is used.

8: \textbf{x}[dim] – double \hspace{1cm} \textit{Output}

\textit{Note}: the dimension, \texttt{dim}, of the array \texttt{x} must be at least

\[
\max(1, \texttt{pd} \times \texttt{k}) \quad \text{when} \quad \texttt{order} = \text{Nag}_\text{ColMajor}; \\
\max(1, \texttt{n} \times \texttt{pd} \times \texttt{k}) \quad \text{when} \quad \texttt{order} = \text{Nag}_\text{RowMajor}.
\]

The \((i,j)\)th element of the matrix \( X \) is stored in

\[
\begin{align*}
\texttt{x}(j - 1) \times \texttt{pd} + i - 1] \quad \text{when} \quad \texttt{order} = \text{Nag}_\text{ColMajor}; \\
\texttt{x}(i - 1) \times \texttt{pd} + j - 1] \quad \text{when} \quad \texttt{order} = \text{Nag}_\text{RowMajor}.
\end{align*}
\]

\textit{On exit}: contains the matrix \( X \).

9: \textbf{pdx} – Integer \hspace{1cm} \textit{Input}

\textit{On entry}: the stride separating row or column elements (depending on the value of \texttt{order}) in the array \texttt{x}.

\textit{Constraints}:

\[
\begin{align*}
\text{if} \quad \texttt{order} = \text{Nag}_\text{ColMajor}, \quad \texttt{pd} \geq \texttt{n}; \\
\text{if} \quad \texttt{order} = \text{Nag}_\text{RowMajor}, \quad \texttt{pd} \geq \texttt{k}.
\end{align*}
\]

10: \textbf{iter} – Integer * \hspace{1cm} \textit{Output}

\textit{On exit}: the number of steps taken in the spectral projected gradient method.

11: \textbf{feval} – Integer * \hspace{1cm} \textit{Output}

\textit{On exit}: the number of evaluations of \( \|G - XX^T + \text{diag}(XX^T - I)\|_F \).

12: \textbf{nrmpgd} – double * \hspace{1cm} \textit{Output}

\textit{On exit}: the norm of the projected gradient at the final iteration.
6 Error Indicators and Warnings

**NE_ALLOC_FAIL**
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

**NE_BAD_PARAM**
On entry, argument `<value>` had an illegal value.

**NE_CONVERGENCE**
Spectral gradient method fails to converge in `<value>` iterations.

**NE_INT**
On entry, \( n = \langle\text{value}\rangle \).
Constraint: \( n > 0 \).

**NE_INT_2**
On entry, \( k = \langle\text{value}\rangle \) and \( n = \langle\text{value}\rangle \).
Constraint: \( 0 < k \leq n \).
On entry, \( \text{pdg} = \langle\text{value}\rangle \) and \( n = \langle\text{value}\rangle \).
Constraint: \( \text{pdg} \geq n \).
On entry, \( \text{pdx} = \langle\text{value}\rangle \) and \( k = \langle\text{value}\rangle \).
Constraint: \( \text{pdx} \geq k \).
On entry, \( \text{pdx} = \langle\text{value}\rangle \) and \( n = \langle\text{value}\rangle \).
Constraint: \( \text{pdx} \geq n \).

**NE_INTERNAL_ERROR**
An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.
An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

**NE_NO_LICENCE**
Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

7 Accuracy
The returned accuracy is controlled by `errtol` and limited by `machine precision`.

8 Parallelism and Performance
`nag_nearest_correlation_k_factor (g02aec)` is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.
`nag_nearest_correlation_k_factor (g02aec)` makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.
Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users’ Note for your implementation for any additional implementation-specific information.

9 Further Comments

Arrays are internally allocated by nag_nearest_correlation_k_factor (g02aec). The total size of these arrays is $n \times n + 4 \times n \times k + (nb + 3) \times n + n + 50$ double elements and $6 \times n$ integer elements. There is an additional $n \times k$ double elements if order = Nag_RowMajor. Here nb is the block size required for optimal performance by nag_dsytrd (f08fec) and nag_dormtr (f08fgc) which are called internally. All allocated memory is freed before return of nag_nearest_correlation_k_factor (g02aec).

See nag_mv_factor (g03cac) for constructing the factor loading matrix from a known correlation matrix.

10 Example

This example finds the nearest correlation matrix with $k = 2$ factor structure to:

$$
G = \begin{pmatrix}
2 & -1 & 0 & 0 \\
-1 & 2 & -1 & 0 \\
0 & -1 & 2 & -1 \\
0 & 0 & -1 & 2
\end{pmatrix}
$$

10.1 Program Text

/* nag_nearest_correlation_k_factor (g02aec) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 23, 2011. */

#include <nag.h>
#include <nag_stdlib.h>
#include <nagf16.h>
#include <nagg02.h>
#include <nagx04.h>

int main(void)
{

    /* Scalars */
    Integer exit_status = 0;
    double errtol, nrmpgd;
    Integer feval, i, iter, j, k, pda, pdg, pdx, maxit, n;

    /* Arrays */
    double *a = 0, *g = 0, *x = 0;

    /* Nag Types */
    Nag_OrderType order;
    NagError fail;

    INIT_FAIL(fail);

    #ifdef NAG_COLUMN_MAJOR
    #define A(I, J) a[(J-1)*pda + I-1]
    #define G(I, J) g[(J-1)*pdg + I-1]
    #define X(I, J) x[(J-1)*pdx + I-1]
    order = Nag_ColMajor;
    #else
    #define A(I, J) a[(I-1)*pda + J-1]
    #define G(I, J) g[(I-1)*pdg + J-1]
    #define X(I, J) x[(I-1)*pdx + J-1]
    order = Nag_RowMajor;
    #endif

    /* ... Rest of the program ... */

    return exit_status;
}
/* Output preamble */
printf("nag_nearest_correlation_k_factor (g02aec) ");
printf("Example Program Results\n\n");
fflush(stdout);

/* Skip heading in data file */
#ifdef _WIN32
scanf_s("%*\n");
#else
scanf("%*\n");
#endif
/* Read in the problem size */
#ifdef _WIN32
scanf_s("%"NAG_IFMT"%*\n", &n);
#else
scanf("%"NAG_IFMT"%*\n", &n);
#endif
pda = n;
pdg = n;
pdx = n;
if (!
(a = NAG_ALLOC((pda)*(n), double) ||
(g = NAG_ALLOC((pdg)*(n), double) ||
(x = NAG_ALLOC((pdx)*(n), double))
)
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}

/* Read in the matrix g */
for (i = 1; i <= n; i++)
for (j = 1; j <= n; j++)
#ifdef _WIN32
scanf_s("%lf", &G(i, j));
#else
scanf("%lf", &G(i, j));
#endif
#ifdef _WIN32
scanf("%*\n");
#else
scanf("%*\n");
#endif

/* Use the defaults for errtol and maxit */
errtol = 0.0;
maxit = 0;
/* Set k value*/
k = 2;

/* nag_nearest_correlation_k_factor (g02aec).
* Computes the nearest correlation matrix with k-factor structure
* to a real square matrix
*/
nag_nearest_correlation_k_factor(order, g, pdg, n, k, errtol, maxit, x,
pdx, &iter, &feval, &nrmpgd, &fail);
if (fail.code != NE_NOERROR)
{
printf("%s\n", fail.message);
exit_status = 1;
goto END;
}

/* nag_gen_real_mat_print (x04cac).
* Print real general matrix (easy-to-use)
*/
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, k, x,
pdx, "Factor Loading Matrix X", NULL, &fail);
if (fail.code != NE_NOERROR) {
  printf("%s\n", fail.message);
  exit_status = 1;
  goto END;
}

printf("Number of steps taken: %"NAG_IFMT"\n", iter);
printf("Number of function evaluations: %"NAG_IFMT"\n", feval);
fflush(stdout);

/* Generate Nearest k factor correlation matrix */
/* nag_dgemm (f16yac) performs matrix-matrix multiplication for a */
/* real general matrix */

nag_dgemm(order, Nag_NoTrans, Nag_Trans, n, n, k, 1.0, x, pdx, x, pdx,
        0.0, a, pda, &fail);
if (fail.code != NE_NOERROR) {
  printf("%s\n", fail.message);
  exit_status = 1;
  goto END;
}

for (i = 1; i <= n; i++)
  A(i, i) = 1.0;

/* nag_gen_real_mat_print (x04cac). */
/* Print real general matrix (easy-to-use) */

nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, a,
                      pda, "Nearest Correlation Matrix", NULL, &fail);
if (fail.code != NE_NOERROR) {
  printf("%s\n", fail.message);
  exit_status = 1;
}

END:
NAG_FREE(a);
NAG_FREE(g);
NAG_FREE(x);
return exit_status;

10.2 Program Data

nag_nearest_correlation_k_factor (g02aec) Example Program Data

4
  2.0 -1.0 0.0 0.0
-1.0 2.0 -1.0 0.0
0.0 -1.0 2.0 -1.0
0.0 0.0 -1.0 2.0 :: End of g

10.3 Program Results

nag_nearest_correlation_k_factor (g02aec) Example Program Results

Factor Loading Matrix X

1 2
1 0.7665 -0.6271
2 -0.4250 0.9052
3 -0.4250 -0.9052
4 0.7665 0.6271

Number of steps taken: 5
Number of function evaluations: 6

Nearest Correlation Matrix
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000</td>
<td>-0.8935</td>
<td>0.2419</td>
<td>0.1943</td>
</tr>
<tr>
<td>2</td>
<td>-0.8935</td>
<td>1.0000</td>
<td>-0.6388</td>
<td>0.2419</td>
</tr>
<tr>
<td>3</td>
<td>0.2419</td>
<td>-0.6388</td>
<td>1.0000</td>
<td>-0.8935</td>
</tr>
<tr>
<td>4</td>
<td>0.1943</td>
<td>0.2419</td>
<td>-0.8935</td>
<td>1.0000</td>
</tr>
</tbody>
</table>