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

```
#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 $C = XX^{\mathrm{T}} + \mathrm{diag}(I - XX^{\mathrm{T}})$, 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 $\min(\|G - XX^{\mathrm{T}} + \mathrm{diag}(XX^{\mathrm{T}} - I)\|)_F$ such that $\left\|x_i^{\mathrm{T}}\right\|_2 \leq 1$, for $i = 1, 2, \ldots, n$, where x_i is the ith row of the factor loading matrix, X, which gives us the solution.

4 References

Birgin E G, MartÕnez J M and Raydan M (2001) Algorithm 813: SPG–software for convex-constrained optimization *ACM Trans. Math. Software* **27** 340–349

Borsdorf R, Higham N J and Raydan M (2010) Computing a nearest correlation matrix with factor structure. *SIAM J. Matrix Anal. Appl.* **31(5)** 2603–2622

5 Arguments

1: **order** – Nag_OrderType *Input*

On entry: the **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 **order** = Nag_RowMajor. See Section 2.3.1.3 in How to Use the NAG Library and its Documentation for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

2: **g**[dim] – double *Input/Output*

Note: the dimension, *dim*, of the array **g** must be at least **pdg** × **n**.

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

g[$(j-1) \times$ **pdg** $+ i - 1$] when **order** = Nag_ColMajor;
g[$(i-1) \times$ **pdg** $+ j - 1$] when **order** = Nag_RowMajor.

On entry: G, the initial matrix.

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

3: **pdg** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **g**.

Constraint: **pdg** $\geq$ **n**.

4: **n** – Integer *Input*

On entry: n, the order of the matrix G.

Constraint: **n** > 0.

5: **k** – Integer *Input*

On entry: k, the number of factors and columns of X.

Constraint: $0 <$ **k** $\leq$ **n**.

6: **errtol** – double *Input*

On entry: the termination tolerance for the projected gradient norm. See references for further details. If **errtol** ≤ 0.0 then 0.01 is used. This is often a suitable default value.

7: **maxit** – Integer *Input*

On entry: specifies the maximum number of iterations in the spectral projected gradient method.

If **maxit** ≤ 0, 40000 is used.

8: **x**$[dim]$ – double *Output*

Note: the dimension, *dim*, of the array **x** must be at least

max$(1,$ **pdx** $\times$ **k**$)$ when **order** $=$ Nag_ColMajor;
max$(1,$ **n** $\times$ **pdx**$)$ when **order** $=$ Nag_RowMajor.

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

x$[(j-1) \times$ **pdx** $+ i - 1]$ when **order** $=$ Nag_ColMajor;
x$[(i-1) \times$ **pdx** $+ j - 1]$ when **order** $=$ Nag_RowMajor.

On exit: contains the matrix X.

9: **pdx** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **x**.

Constraints:

if **order** $=$ Nag_ColMajor, **pdx** $\geq$ **n**;
if **order** $=$ Nag_RowMajor, **pdx** $\geq$ **k**.

10: **iter** – Integer * *Output*

On exit: the number of steps taken in the spectral projected gradient method.

11: **feval** – Integer * *Output*

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

12: **nrmpgd** – double * *Output*

On exit: the norm of the projected gradient at the final iteration.

13: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation 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, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} > 0$.

NE_INT_2

On entry, $\mathbf{k} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $0 < \mathbf{k} \leq \mathbf{n}$.

On entry, $\mathbf{pdg} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{pdg} \geq \mathbf{n}$.

On entry, $\mathbf{pdx} = \langle value \rangle$ and $\mathbf{k} = \langle value \rangle$.
Constraint: $\mathbf{pdx} \geq \mathbf{k}$.

On entry, $\mathbf{pdx} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{pdx} \geq \mathbf{n}$.

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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation 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 $\mathbf{n} \times \mathbf{n} + 4 \times \mathbf{n} \times \mathbf{k} + (nb + 3) \times \mathbf{n} + \mathbf{n} + 50$ double elements and $6 \times \mathbf{n}$ Integer elements. There is an additional $\mathbf{n} \times \mathbf{k}$ 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.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#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]
  order = Nag_ColMajor;
#else
```

```
#define A(I, J) a[(I-1)*pda + J-1]
#define G(I, J) g[(I-1)*pdg + J-1]
  order = Nag_RowMajor;
#endif

  /* 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_s("%*[^\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("\nNumber of steps taken: %" NAG_IFMT "\n", iter);
  printf("Number of function evaluations: %" NAG_IFMT "\n\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                          :: n
 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
```

```
          1       2       3       4
 1   1.0000 -0.8935  0.2419  0.1943
 2  -0.8935  1.0000 -0.6388  0.2419
 3   0.2419 -0.6388  1.0000 -0.8935
 4   0.1943  0.2419 -0.8935  1.0000
```