NAG Library Function Document

nag_nearest_correlation_h_weight (g02ajc)

1 Purpose

nag_nearest_correlation_h_weight (g02ajc) computes the nearest correlation matrix, using element-wise weighting in the Frobenius norm and optionally with bounds on the eigenvalues, to a given square, input matrix.

2 Specification

```
#include <nag.h>
#include <nagg02.h>
void nag_nearest_correlation_h_weight (double g[], Integer pdg, Integer n,
     double alpha, double h[], Integer pdh, double errtol, Integer maxit,
     double x[], Integer pdx, Integer *iter, double *norm, NagError *fail)
```

3 Description

nag_nearest_correlation_h_weight (g02ajc) finds the nearest correlation matrix, X, to an approximate correlation matrix, G, using element-wise weighting, this minimizes $\|H \circ (G - X)\|_F$, where $C = A \circ B$ denotes the matrix C with elements $C_{ij} = A_{ij} \times B_{ij}$.

You can optionally specify a lower bound on the eigenvalues, α, of the computed correlation matrix, forcing the matrix to be strictly positive definite, if $0 < \alpha < 1$.

Zero elements in H should be used when you wish to put no emphasis on the corresponding element of G. The algorithm scales H so that the maximum element is 1. It is this scaled matrix that is used in computing the norm above and for the stopping criteria described in Section 7.

Note that if the elements in H vary by several orders of magnitude from one another the algorithm may fail to converge.

4 References

Borsdorf R and Higham N J (2010) A preconditioned (Newton) algorithm for the nearest correlation matrix *IMA Journal of Numerical Analysis* **30(1)** 94–107

Jiang K, Sun D and Toh K-C (To appear) An inexact accelerated proximal gradient method for large scale linearly constrained convex SDP

Qi H and Sun D (2006) A quadratically convergent Newton method for computing the nearest correlation matrix *SIAM J. Matrix AnalAppl* **29(2)** 360–385

5 Arguments

1: **g**[**pdg** × **n**] – double *Input/Output*

Note: the (i, j)th element of the matrix G is stored in **g**$[(j-1) \times$ **pdg** $+ i - 1]$.

On entry: G, the initial matrix.

On exit: G is overwritten.

2: **pdg** – Integer *Input*

On entry: the stride separating matrix row elements in the array **g**.

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

3: **n** – Integer *Input*

On entry: the order of the matrix G.

Constraint: **n** > 0.

4: **alpha** – double *Input*

On entry: the value of α.

If **alpha** < 0.0, 0.0 is used.

Constraint: **alpha** < 1.0.

5: **h**[**pdh** $\times$ **n**] – double *Input/Output*

Note: the (i,j)th element of the matrix H is stored in **h**$[(j-1) \times$ **pdh** $+ i - 1]$.

On entry: the matrix of weights H.

On exit: a symmetric matrix $\frac{1}{2}(H + H^{\mathrm{T}})$ with its diagonal elements set to zero and the remaining elements scaled so that the maximum element is 1.0.

Constraint: $H[(j-1) \times$ **pdh** $+ i - 1] \geq 0.0$, for all i and $j = 1, 2, \ldots, n$, $i \neq j$.

6: **pdh** – Integer *Input*

On entry: the stride separating matrix row elements in the array **h**.

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

7: **errtol** – double *Input*

On entry: the termination tolerance for the iteration. If **errtol** ≤ 0.0 then **n** $\times \sqrt{\textbf{\textit{machine precision}}}$ is used. See Section 7 for further details.

8: **maxit** – Integer *Input*

On entry: specifies the maximum number of iterations to be used.

If **maxit** ≤ 0, 200 is used.

9: **x**[**pdx** $\times$ **n**] – double *Output*

Note: the (i,j)th element of the matrix X is stored in **x**$[(j-1) \times$ **pdx** $+ i - 1]$.

On exit: contains the nearest correlation matrix.

10: **pdx** – Integer *Input*

On entry: the stride separating matrix row elements in the array **x**.

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

11: **iter** – Integer * *Output*

On exit: the number of iterations taken.

12: **norm** – double * *Output*

On exit: the value of $\|H \circ (G - X)\|_F$ after the final iteration.

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

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

Function fails to converge in ⟨*value*⟩ iterations.
Increase **maxit** or check the call to the function.

NE_EIGENPROBLEM

Failure to solve intermediate eigenproblem. This should not occur. Please contact NAG with details of your call.

NE_INT

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} > 0$.

NE_INT_2

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

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

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 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.

NE_REAL

On entry, $\mathbf{alpha} = \langle value \rangle$.
Constraint: $\mathbf{alpha} < 1.0$.

NE_WEIGHTS_NOT_POSITIVE

On entry, one or more of the off-diagonal elements of H were negative.

7 Accuracy

The returned accuracy is controlled by **errtol** and limited by ***machine precision***. If e_i is the value of **norm** at the ith iteration, that is

$$e_i = \|H \circ (G - X)\|_F,$$

where H has been scaled as described above, then the algorithm terminates when:

$$\frac{|e_i - e_{i-1}|}{1 + \max(e_i, e_{i-1})} \leq \textbf{errtol}.$$

8 Parallelism and Performance

nag_nearest_correlation_h_weight (g02ajc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_nearest_correlation_h_weight (g02ajc) 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_h_weight (g02ajc). The total size of these arrays is $15 \times \mathbf{n} + 5 \times \mathbf{n} \times \mathbf{n} + \max(2 \times \mathbf{n} \times \mathbf{n} + 6 \times \mathbf{n} + 1, 120 + 9 \times \mathbf{n})$ double elements and $5 \times \mathbf{n} + 3$ Integer elements. All allocated memory is freed before return of nag_nearest_correlation_h_weight (g02ajc).

10 Example

This example finds the nearest correlation matrix to:

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

weighted by:

$$H = \begin{pmatrix} 0.0 & 10.0 & 0.0 & 0.0 \\ 10.0 & 0.0 & 1.5 & 1.5 \\ 0.0 & 1.5 & 0.0 & 0.0 \\ 0.0 & 1.5 & 0.0 & 0.0 \end{pmatrix}$$

with minimum eigenvalue 0.04.

10.1 Program Text

```
/* nag_nearest_correlation_h_weight (g02ajc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagg02.h>
#include <nagx04.h>

int main(void)
{

#define G(I,J) g[(J-1)*pdg + I-1]
#define H(I,J) h[(J-1)*pdh + I-1]
```

```
  /*  Scalars */
  Integer exit_status = 0;
  Integer one=1;
  double alpha, errtol, norm;
  Integer i, j, iter, maxit, n, pdg, pdh, pdx;

  /*  Arrays */
  double  *eig = 0,  *g = 0,  *h = 0,  *x = 0;

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

  INIT_FAIL(fail);

  /* Output preamble */
  printf("nag_nearest_correlation_h_weight (g02ajc)");
  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 and alpha*/
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%lf%*[^\n] ", &n,  &alpha);
#else
  scanf("%"NAG_IFMT"%lf%*[^\n] ", &n,  &alpha);
#endif

  pdg = n;
  pdh = n;
  pdx = n;
  if (
      !(g = NAG_ALLOC(pdg*n, double))||
      !(h = NAG_ALLOC(pdh*n, double))||
      !(x = NAG_ALLOC(pdx*n, double))||
      !(eig = NAG_ALLOC(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

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

```
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Use the defaults for ERRTOL and MAXIT*/
  errtol = 0.0;
  maxit = 0;

  /*
   * nag_nearest_correlation_h_weight (g02ajc).
   * Calculate nearest correlation matrix with element-wise weighting
   */
  nag_nearest_correlation_h_weight(g, pdg, n, alpha, h, pdh, errtol, maxit,
                                   x, pdx, &iter, &norm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_nearest_correlation_h_weight (g02ajc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  /* Display results*/

  order = Nag_ColMajor;
  /*
   * nag_gen_real_mat_print (x04cac).
   * Prints real general matrix
   */
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, h,
                         pdh, "Returned h Matrix", NULL, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
      exit_status = 2;
      goto END;
    }

  printf("\n");
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, x,
                         pdx, "Nearest Correlation Matrix x", NULL, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
      exit_status = 3;
      goto END;
    }

  printf("\n%s %11"NAG_IFMT" \n\n","Number of iterations:",  iter);
  printf("Norm value:%27.4f \n\n", norm);
  printf("%s %30.4f \n","alpha: ",  alpha);

  /*
   * nag_dsyev (f08fac).
   * Computes all eigenvalues and, optionally, eigenvectors of a real
   * symmetric matrix
   */
  nag_dsyev(order, Nag_EigVals, Nag_Upper, n, x, pdx, eig, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dsyev (f08fac).\n%s\n", fail.message);
      exit_status = 4;
      goto END;
    }

  printf("\n");
  fflush(stdout);
```

```
  /*
   * nag_gen_real_mat_print (x04cac).
   * Print real general matrix (easy-to-use)
   */
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, one, n,
                         eig, one, "Eigenvalues of x", NULL, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
      exit_status = 5;
      goto END;
    }

 END:
  NAG_FREE(eig);
  NAG_FREE(g);
  NAG_FREE(h);
  NAG_FREE(x);
  return exit_status;
}
```

10.2 Program Data

```
nag_nearest_correlation_h_weight (g02ajc) Example Program Data
 4  0.04                      :: n, alpha
   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
   0.0   10.0    0.0    0.0
  10.0    0.0    1.5    1.5
   0.0    1.5    0.0    0.0
   0.0    1.5    0.0    0.0  :: End of h
```

10.3 Program Results

```
nag_nearest_correlation_h_weight (g02ajc) Example Program Results

 Returned h Matrix
         1       2       3       4
 1   0.0000  1.0000  0.0000  0.0000
 2   1.0000  0.0000  0.1500  0.1500
 3   0.0000  0.1500  0.0000  0.0000
 4   0.0000  0.1500  0.0000  0.0000

 Nearest Correlation Matrix x
              1            2            3            4
 1    1.0000e+00  -9.2285e-01   7.7335e-01   2.5854e-03
 2   -9.2285e-01   1.0000e+00  -7.8433e-01  -8.4891e-07
 3    7.7335e-01  -7.8433e-01   1.0000e+00  -6.1477e-02
 4    2.5854e-03  -8.4891e-07  -6.1477e-02   1.0000e+00

Number of iterations:          66

Norm value:                     0.1183

alpha:                          0.0400

 Eigenvalues of x
           1          2          3          4
 1      0.0769     0.2637     1.0031     2.6563
```