NAG Library Function Document

nag_nearest_correlation_bounded (g02abc)

1 Purpose

nag_nearest_correlation_bounded (g02abc) computes the nearest correlation matrix, in the Frobenius norm or weighted 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_bounded (Nag_OrderType order, double g[],
     Integer pdg, Integer n, Nag_NearCorr_ProbType opt, double alpha,
     double w[], double errtol, Integer maxits, Integer maxit, double x[],
     Integer pdx, Integer *iter, Integer *feval, double *nrmgrd,
     NagError *fail)
```

3 Description

Finds the nearest correlation matrix X by minimizing $\frac{1}{2}\|G-X\|^2$ where G is an approximate correlation matrix.

The norm can either be the Frobenius norm or the weighted Frobenius norm $\frac{1}{2}\left\|W^{\frac{1}{2}}(G-X)W^{\frac{1}{2}}\right\|_F^2$.

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

Note that if the weights 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

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: **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 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

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

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

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

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

On entry: G, the initial matrix.

On exit: G is overwritten.

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: the order of the matrix G.

Constraint: **n** > 0.

5: **opt** – Nag_NearCorr_ProbType *Input*

On entry: indicates the problem to be solved.

opt = Nag_LowerBound
The lower bound problem is solved.

opt = Nag_WeightedNorm
The weighted norm problem is solved.

opt = Nag_Both
Both problems are solved.

Constraint: **opt** = Nag_LowerBound, Nag_WeightedNorm or Nag_Both.

6: **alpha** – double *Input*

On entry: the value of α.

If **opt** = Nag_WeightedNorm, **alpha** need not be set.

Constraint: $0.0 <$ **alpha** < 1.0.

7: **w**[**n**] – double *Input/Output*

On entry: the square roots of the diagonal elements of W, that is the diagonal of $W^{\frac{1}{2}}$.

If **opt** = Nag_LowerBound, **w** need not be set.

On exit: if **opt** = Nag_WeightedNorm or Nag_Both, the array is scaled so $0 <$ **w**$[i-1] \leq 1$, for $i = 1, 2, \ldots, n$.

Constraint: **w**$[i-1] > 0.0$, for $i = 1, 2, \ldots, n$.

8: **errtol** – double *Input*

On entry: the termination tolerance for the Newton iteration. If **errtol** ≤ 0.0 then **n** $\times \sqrt{\textit{machine precision}}$ is used.

9: **maxits** – Integer *Input*

On entry: specifies the maximum number of iterations to be used by the iterative scheme to solve the linear algebraic equations at each Newton step.

If **maxits** ≤ 0, $2 \times$ **n** is used.

10: **maxit** – Integer *Input*

On entry: specifies the maximum number of Newton iterations.

If **maxit** ≤ 0, 200 is used.

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

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

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

On exit: contains the nearest correlation matrix.

12: **pdx** – Integer *Input*

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

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

13: **iter** – Integer * *Output*

On exit: the number of Newton steps taken.

14: **feval** – Integer * *Output*

On exit: the number of function evaluations of the dual problem.

15: **nrmgrd** – double * *Output*

On exit: the norm of the gradient of the last Newton step.

16: **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.

NE_BAD_PARAM

On entry, argument $\langle$*value*$\rangle$ had an illegal value.

NE_CONVERGENCE

Newton iteration fails to converge in $\langle$*value*$\rangle$ iterations. Increase **maxit** or check the call to the function.

The ***machine precision*** is limiting convergence. In this instance the returned matrix X may be useful.

NE_EIGENPROBLEM

Failure to solve intermediate eigenproblem.

NE_INT

On entry, **n** = $\langle$*value*$\rangle$.
Constraint: **n** > 0.

NE_INT_2

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

On entry, **pdx** = $\langle$*value*$\rangle$ and **n** = $\langle$*value*$\rangle$.
Constraint: **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.

NE_REAL

On entry, **alpha** $= \langle value \rangle$.
Constraint: $0.0 <$ **alpha** < 1.0.

NE_WEIGHTS_NOT_POSITIVE

On entry, all elements of **w** were not positive.

7 Accuracy

The returned accuracy is controlled by **errtol** and limited by ***machine precision***.

8 Parallelism and Performance

nag_nearest_correlation_bounded (g02abc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_nearest_correlation_bounded (g02abc) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

Arrays are internally allocated by nag_nearest_correlation_bounded (g02abc). The total size of these arrays is $12 \times \mathbf{n} + 3 \times \mathbf{n} \times \mathbf{n} + \max(2 \times \mathbf{n} \times \mathbf{n} + 6 \times \mathbf{n} + 1, 120 + 9 \times \mathbf{n})$ double elements and $5 \times \mathbf{n} + 3$ Integer elements. All allocated memory is freed before return of nag_nearest_correlation_bounded (g02abc).

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 $W^{\frac{1}{2}} = \mathrm{diag}(100, 20, 20, 20)$ with minimum eigenvalue 0.02.

10.1 Program Text

```
/* nag_nearest_correlation_bounded (g02abc) Example Program.
 *
 * Copyright 2011, Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

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

```
int main(void)
{

  /* Scalars */
  Integer               exit_status = 0;
  double                alpha, errtol, nrmgrd;
  Integer               feval, i, iter, j, maxit, maxits, n, pdeig, pdg, pdx;

  /* Arrays */
  char                  nag_enum_arg[100];
  double                *eig = 0, *g = 0, *w = 0, *x = 0;

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

  INIT_FAIL(fail);

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

  /* Output preamble */
  printf("nag_nearest_correlation_bounded (g02abc)");
  printf(" Example Program Results\n\n");
  fflush(stdout);

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  /* Read in the problem size, opt and alpha */
  scanf("%ld", &n);
  scanf("%39s", nag_enum_arg);
  /*
   * nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  opt = (Nag_NearCorr_ProbType) nag_enum_name_to_value(nag_enum_arg);
  scanf("%lf%*[^\n]", &alpha);

  pdg = n;
  pdx = n;
  if( order == Nag_ColMajor)
    pdeig = 1;
  else
    pdeig = n;

  if (
      !(g = NAG_ALLOC((pdg)*(n), double)) ||
      !(w = NAG_ALLOC((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++)
      scanf("%lf", &G(i, j));
  scanf("%*[^\n] ");
  /* Read in the vector w */
  for (i = 0; i < n; i++)
    scanf("%lf", &w[i]);
  scanf("%*[^\n] ");

  /* Use the defaults for errtol, maxits and maxit */
  errtol = 0.0;
  maxits = 0;
  maxit = 0;

  /*
   * nag_nearest_correlation_bounded (g02abc).
   * Computes the nearest correlation matrix incorporating weights
   * and/or bounds
   */
  nag_nearest_correlation_bounded(order, g, pdg, n, opt, alpha, w, errtol,
                                  maxits, maxit, x, pdx, &iter, &feval,
                                  &nrmgrd, &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, n, x,
                         pdx, "Nearest Correlation Matrix x", NULL, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  printf("\nNumber of Newton steps taken: %11ld\n", iter);
  printf("Number of function evaluations: %9ld\n\n", feval);
  printf("alpha: %37.3f \n\n", 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("%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, 1, n,
                         eig, pdeig, "Eigenvalues of x", NULL, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("%s\n", fail.message);
      exit_status = 1;
    }

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

10.2 Program Data

```
nag_nearest_correlation_bounded (g02abc) Example Program Data
 4  Nag_Both  0.02            :: n, opt, 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
 100.0   20.0   20.0   20.0  :: w
```

10.3 Program Results

```
nag_nearest_correlation_bounded (g02abc) Example Program Results

 Nearest Correlation Matrix x
         1       2       3       4
 1   1.0000 -0.9187  0.0257  0.0086
 2  -0.9187  1.0000 -0.3008  0.2270
 3   0.0257 -0.3008  1.0000 -0.8859
 4   0.0086  0.2270 -0.8859  1.0000

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

alpha:                                0.020

 Eigenvalues of x
            1          2          3          4
 1      0.0392     0.1183     1.6515     2.1910
```