NAG Library Function Document

nag_nearest_correlation (g02aac)

1 Purpose

nag_nearest_correlation (g02aac) computes the nearest correlation matrix, in the Frobenius norm, to a given square, input matrix.

2 Specification

```
#include <nag.h>
#include <nagg02.h>
void nag_nearest_correlation (Nag_OrderType order, double g[], Integer pdg,
     Integer n, double errtol, Integer maxits, Integer maxit, double x[],
     Integer pdx, Integer *iter, Integer *feval, double *nrmgrd,
     NagError *fail)
```

3 Description

A correlation matrix may be characterised as a real square matrix that is symmetric, has a unit diagonal and is positive semidefinite.

nag_nearest_correlation (g02aac) applies an inexact Newton method to a dual formulation of the problem, as described by Qi and Sun (2006). It applies the improvements suggested by Borsdorf and Higham (2010).

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 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 set to I.

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 size of the matrix G.

Constraint: **n** > 0.

5: **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.

6: **maxits** – Integer *Input*

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

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

7: **maxit** – Integer *Input*

On entry: specifies the maximum number of Newton iterations.

If **maxit** ≤ 0, 200 is used.

8: **x**$[dim]$ – double *Output*

Note: the dimension, *dim*, of the array **x** must be at least **pdx** $\times$ **n**.

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.

9: **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**.

10: **iter** – Integer * *Output*

On exit: the number of Newton steps taken.

11: **feval** – Integer * *Output*

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

12: **nrmgrd** – double * *Output*

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

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

Machine precision is limiting convergence.
The array returned in **x** may still be of interest.

Newton iteration fails to converge in ⟨*value*⟩ iterations.

NE_EIGENPROBLEM

An intermediate eigenproblem could not be solved. 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, $\mathbf{pdx} = \langle value \rangle$ 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 (g02aac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_nearest_correlation (g02aac) 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 (g02aac). The total size of these arrays is $11 \times \mathbf{n} + 3 \times \mathbf{n} \times \mathbf{n} + \max(2 \times \mathbf{n} \times \mathbf{n} + 6 \times \mathbf{n} + 1, 120 + 9 \times \mathbf{n})$ real elements and $5 \times \mathbf{n} + 3$ integer elements.

10 Example

This example finds the nearest correlation matrix to:

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

10.1 Program Text

```
/* nag_nearest_correlation (g02aac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg02.h>

int main(void)
{
  /*Integer scalar and array declarations */
  Integer exit_status = 0;
  Integer feval, i, iter, j, maxit, maxits, n;
  Integer ldg, pdg, pdx;
  /*Double scalar and array declarations */
  double errtol, nrmgrd;
  double *g = 0, *x = 0;
  Nag_OrderType order;
  NagError fail;

  INIT_FAIL(fail);

  printf("%s\n", "nag_nearest_correlation (g02aac) Example Program Results");
  printf("\n");
  n = 4;
  ldg = 5;
#ifdef NAG_COLUMN_MAJOR
  pdg = ldg;
#define G(I, J) g[(J-1)*pdg + I-1]
  pdx = n;
#define X(I, J) x[(J-1)*pdx + I-1]
  order = Nag_ColMajor;
#else
  pdg = n;
#define G(I, J) g[(I-1)*pdg + J-1]
  pdx = n;
#define X(I, J) x[(I-1)*pdx + J-1]
```

```
  order = Nag_RowMajor;
#endif
  if (!(g = NAG_ALLOC(ldg * n, double)) || !(x = NAG_ALLOC(n * n, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Set up matrix G */
  for (j = 1; j <= n; j++) {
    for (i = 1; i <= n; i++)
      G(i, j) = 0.0;
    G(j, j) = 2.00e0;
  }
  for (j = 2; j <= n; j++) {
    G(j - 1, j) = (-(1.00e0));
    G(j, j - 1) = (-(1.00e0));
  }
  /* Set up method parameters */
  errtol = 1.00e-7;
  maxits = 200;
  maxit = 10;
  /*
   * nag_nearest_correlation (g02aac)
   * Computes the nearest correlation matrix to a real square matrix,
   * using the method of Qi and Sun
   */
  nag_nearest_correlation(order, g, pdg, n, errtol, maxits, maxit, x, pdx,
                          &iter, &feval, &nrmgrd, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_nearest_correlation (g02aac).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }
  printf("%s\n", "     Nearest Correlation Matrix");
  for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++)
      printf("%11.5f%s", X(i, j), j % 4 ? " " : "\n");
  }
  printf("\n");
  printf("\n");
  printf("%s%11" NAG_IFMT "\n", " Number of Newton steps taken:", iter);
  printf("%s%9" NAG_IFMT "\n", " Number of function evaluations:", feval);
  if (nrmgrd > errtol)
    printf("%s %12.3e\n", " Norm of gradient of last Newton step:", nrmgrd);
  printf("\n");

END:
  NAG_FREE(g);
  NAG_FREE(x);

  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_nearest_correlation (g02aac) Example Program Results

     Nearest Correlation Matrix
    1.00000    -0.80841     0.19159     0.10678
   -0.80841     1.00000    -0.65623     0.19159
    0.19159    -0.65623     1.00000    -0.80841
    0.10678     0.19159    -0.80841     1.00000
```

```
Number of Newton steps taken:          3
Number of function evaluations:        4
```
