NAG Library Function Document

nag_mv_promax (g03bdc)

1 Purpose

nag_mv_promax (g03bdc) calculates a ProMax rotation, given information following an orthogonal rotation.

2 Specification

```
#include <nag.h>
#include <nagg03.h>
void nag_mv_promax (Nag_RotationLoading stand, Integer n, Integer m,
     const double x[], Integer pdx, const double ro[], Integer pdro,
     double power, double fp[], Integer pdfp, double r[], Integer pdr,
     double phi[], Integer pdphi, double fs[], Integer pdfs, NagError *fail)
```

3 Description

Let X and Y denote n by m matrices each representing a set of n points in an m-dimensional space. The X matrix is a matrix of loadings as returned by nag_mv_orthomax (g03bac), that is following an orthogonal rotation of a loadings matrix Z. The target matrix Y is calculated as a power transformation of X that preserves the sign of the loadings. Let X_{ij} and Y_{ij} denote the (i,j)th element of matrices X and Y. Given a value greater than one for the exponent p:

$$Y_{ij} = \delta_{ij}\left|\left|X_{ij}\right|\right|^{p},$$

for

$i = 1, 2, \ldots, n;$

$j = 1, 2, \ldots, m;$

$$\delta_{ij} = \begin{cases} -1, & \text{if } X_{ij} < 0; \\ 1, & \text{otherwise.} \end{cases}$$

The above power transformation tends to increase the difference between high and low values of loadings and is intended to increase the interpretability of a solution.

In the second step a solution of:

$$XW = Y, \quad X, Y \in \mathbb{R}^{n\times m},\ W \in \mathbb{R}^{m\times m},$$

is found for W in the least squares sense by use of singular value decomposition of the orthogonal loadings X. The ProMax rotation matrix R is then given by

$$R = OW\tilde{W}, \quad O, \tilde{W} \in \mathbb{R}^{m\times m},$$

where O is the rotation matrix from an orthogonal transformation, and $\tilde{W}$ is a matrix with the square root of diagonal elements of $\left(W^{\mathrm{T}}W\right)^{-1}$ on its diagonal and zeros elsewhere.

The ProMax factor pattern matrix P is given by

$$P = XW\tilde{W}, \quad P \in \mathbb{R}^{n\times m};$$

the inter-factor correlations Φ are given by

$$\Phi = \left(Q^{\mathrm{T}}Q\right)^{-1}, \quad \Phi \in \mathbb{R}^{m\times m};$$

where $Q = W\tilde{W}$; and the factor structure S is given by

$$S = P\Phi, \quad S \in \mathbb{R}^{n\times m}.$$

Optionally, the rows of target matrix Y can be scaled by the communalities of loadings.

4 References

None.

5 Arguments

1: **stand** – Nag_RotationLoading *Input*

On entry: indicates how loadings are normalized.

stand = Nag_RoLoadStand
Rows of Y are (Kaiser) normalized by the communalities of the loadings.

stand = Nag_RoLoadNotStand
Rows are not normalized.

Constraint: **stand** = Nag_RoLoadNotStand or Nag_RoLoadStand.

2: **n** – Integer *Input*

On entry: n, the number of points.

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

3: **m** – Integer *Input*

On entry: m, the number of dimensions.

Constraint: **m** ≥ 1.

4: **x**[**n** $\times$ **pdx**] – const double *Input*

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

On entry: the loadings matrix following an orthogonal rotation, X.

5: **pdx** – Integer *Input*

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

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

6: **ro**[**m** $\times$ **pdro**] – const double *Input*

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

On entry: the orthogonal rotation matrix, O.

7: **pdro** – Integer *Input*

On entry: the stride separating matrix column elements in the array **ro**.

Constraint: **pdro** $\geq$ **m**.

8: **power** – double *Input*

On entry: p, the value of exponent.

Constraint: **power** > 1.0.

9: **fp**[**n** × **pdfp**] – double *Output*

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

On exit: the factor pattern matrix, P.

10: **pdfp** – Integer *Input*

On entry: the stride separating matrix column elements in the array **fp**.

Constraint: **pdfp** $\geq$ **m**.

11: **r**[**m** × **pdr**] – double *Output*

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

On exit: the ProMax rotation matrix, R.

12: **pdr** – Integer *Input*

On entry: the stride separating matrix column elements in the array **r**.

Constraint: **pdr** $\geq$ **m**.

13: **phi**[**m** × **pdphi**] – double *Output*

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

On exit: the matrix of inter-factor correlations, Φ.

14: **pdphi** – Integer *Input*

On entry: the stride separating matrix column elements in the array **phi**.

Constraint: **pdphi** $\geq$ **m**.

15: **fs**[**n** × **pdfs**] – double *Output*

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

On exit: the factor structure matrix, S.

16: **pdfs** – Integer *Input*

On entry: the stride separating matrix column elements in the array **fs**.

Constraint: **pdfs** $\geq$ **m**.

17: **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_INT

On entry, **m** $= \langle value \rangle$.
Constraint: **m** ≥ 1.

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

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

On entry, **pdphi** = ⟨*value*⟩.
Constraint: **pdphi** > 0.

On entry, **pdr** = ⟨*value*⟩.
Constraint: **pdr** > 0.

On entry, **pdro** = ⟨*value*⟩.
Constraint: **pdro** > 0.

On entry, **pdx** = ⟨*value*⟩.
Constraint: **pdx** > 0.

NE_INT_2

On entry, **n** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: **n** ≥ **m**.

On entry, **pdfp** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: **pdfp** ≥ **m**.

On entry, **pdfs** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: **pdfs** ≥ **m**.

On entry, **pdphi** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: **pdphi** ≥ **m**.

On entry, **pdr** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: **pdr** ≥ **m**.

On entry, **pdro** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: **pdro** ≥ **m**.

On entry, **pdx** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: **pdx** ≥ **m**.

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_ARG_LE

On entry, **power** = ⟨*value*⟩.
Constraint: **power** > 1.0.

NE_SVD_FAIL

SVD failed to converge.

7 Accuracy

The calculations are believed to be stable.

8 Parallelism and Performance

nag_mv_promax (g03bdc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_mv_promax (g03bdc) 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

None.

10 Example

This example reads a loadings matrix and calculates a varimax transformation before calculating P, R and σ for a ProMax rotation.

10.1 Program Text

```
/* nag_mv_promax (g03bdc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 9, 2009.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg03.h>
#include <nagx04.h>

int main(void)
{
  /*Integer scalar and array declarations */
  Integer             exit_status = 0;
  Integer             i, iter, j, m, maxit, n;
  Integer             pdfp, pdfs, pdphi, pdr, pdro, pdx;
  /*Double scalar and array declarations */
  double              acc, g, power;
  double              *fp = 0, *fs = 0, *phi = 0, *r = 0, *ro = 0, *x = 0;
  /*Character scalar and array declarations */
  char                sstand[40];
  /*NAG types */
  Nag_OrderType       order;
  Nag_RotationLoading stand;
  NagError            fail;

  INIT_FAIL(fail);

  printf("%s\n", "nag_mv_promax (g03bdc) Example Program Results");
  /* Skip heading in data file*/
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
  scanf_s("%39s %"NAG_IFMT"%"NAG_IFMT"%lf%*[^\n] ", sstand, _countof(sstand), &n,
&m, &power);
#else
  scanf("%39s %"NAG_IFMT"%"NAG_IFMT"%lf%*[^\n] ", sstand, &n, &m, &power);
#endif
  stand = (Nag_RotationLoading) nag_enum_name_to_value(sstand);

  pdfp = m;
  #define FP(I, J)  fp[(I-1)*pdfp + J-1]
  pdfs = m;
  #define FS(I, J)  fs[(I-1)*pdfs + J-1]
  pdphi = m;
  #define PHI(I, J) phi[(I-1)*pdphi + J-1]
  pdr = m;
  #define R(I, J)   r[(I-1)*pdr + J-1]
  pdro = m;
  #define RO(I, J)  ro[(I-1)*pdro + J-1]
```

```
  pdx = pdfp;
  #define X(I, J)   x[(I-1)*pdx + J-1]

  if (!(fp = NAG_ALLOC(pdfp*n, double)) ||
      !(fs = NAG_ALLOC(pdfs*n, double)) ||
      !(phi = NAG_ALLOC(pdphi*m, double)) ||
      !(r = NAG_ALLOC(pdr*m, double)) ||
      !(ro = NAG_ALLOC(pdro*m, double)) ||
      !(x = NAG_ALLOC(pdx*n, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  /* Read loadings matrix.*/
  for (i = 1; i <= n; i++)
    {
      for (j = 1; j <= m; j++)
#ifdef _WIN32
        scanf_s("%lf ", &FP(i, j));
#else
        scanf("%lf ", &FP(i, j));
#endif
    }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
  /*
   * nag_mv_orthomax (g03bac)
   * Orthogonal rotations
   */
  g = 1.0e0;
  acc = 1.0e-5;
  maxit = 200;
  nag_mv_orthomax(stand, g, n, m, fp, pdx, x, ro, pdro, acc, maxit, &iter,
                  &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_mv_orthomax (g03bac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  /*
   * nag_mv_promax (g03bdc)
   * ProMax rotations
   */
  nag_mv_promax(stand, n, m, x, pdx, ro, pdro, power, fp, pdfp, r, pdr,
                phi, pdphi, fs, pdfs, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_mv_promax (g03bdc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  printf("\n");
  /*
   * nag_gen_real_mat_print (x04cac)
   * Print real general matrix (easy-to-use)
   */
  order = Nag_RowMajor;
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, m,
                         fp, pdfp, "Factor pattern", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
              fail.message);
      exit_status = 1;
```

```
      goto END;
    }
  printf("\n");
  /*
   * nag_gen_real_mat_print (x04cac)
   * Print real general matrix (easy-to-use)
   */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m, m,
                         r, pdr, "ProMax rotation", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  printf("\n");
  /*
   * nag_gen_real_mat_print (x04cac)
   * Print real general matrix (easy-to-use)
   */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m, m,
                         phi, pdphi, "Inter-factor correlations", 0,
                         &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  printf("\n");
  /*
   * nag_gen_real_mat_print (x04cac)
   * Print real general matrix (easy-to-use)
   */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, m,
                         fs, pdfs, "Factor structure", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

 END:
  NAG_FREE(fp);
  NAG_FREE(fs);
  NAG_FREE(phi);
  NAG_FREE(r);
  NAG_FREE(ro);
  NAG_FREE(x);

  return exit_status;
}
```

10.2 Program Data

```
nag_mv_promax (g03bdc) Example Program Data
Nag_RoLoadStand 5 2 3.0  : stand n m power
0.74215 -0.57806
0.71370 -0.55515
0.87899 -0.15847
0.62533  0.76621
0.71447  0.67936
```

10.3 Program Results

```
nag_mv_promax (g03bdc) Example Program Results

 Factor pattern
           1          2
 1      0.9556    -0.0979
 2      0.9184    -0.0935
 3      0.7605     0.3393
 4     -0.0791     1.0019
 5      0.0480     0.9751

 ProMax rotation
         1       2
 1   0.7380  0.5420
 2  -0.7055  0.8653

 Inter-factor correlations
         1       2
 1   1.0000  0.2019
 2   0.2019  1.0000

 Factor structure
         1       2
 1   0.9358  0.0950
 2   0.8995  0.0919
 3   0.8290  0.4928
 4   0.1232  0.9860
 5   0.2448  0.9848
```