

# 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  $p$ :

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

for

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

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

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

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:

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

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  $(W^T W)^{-1}$  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  $\Phi$  are given by

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

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**  $\geq$  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**[( $i - 1$ ) × **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** ≥ **m**.
- 11: **r**[**m** × **pdr**] – double *Output*  
**Note:** the  $(i, j)$ th element of the matrix  $R$  is stored in **r**[( $i - 1$ ) × **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** ≥ **m**.
- 13: **phi**[**m** × **pdphi**] – double *Output*  
**Note:** the  $(i, j)$ th element of the matrix is stored in **phi**[( $i - 1$ ) × **pdphi** +  $j - 1$ ].  
*On exit:* the matrix of inter-factor correlations,  $\Phi$ .
- 14: **pdphi** – Integer *Input*  
*On entry:* the stride separating matrix column elements in the array **phi**.  
*Constraint:* **pdphi** ≥ **m**.
- 15: **fs**[**n** × **pdfs**] – double *Output*  
**Note:** the  $(i, j)$ th element of the matrix is stored in **fs**[( $i - 1$ ) × **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** ≥ **m**.
- 17: **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.

### 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** =  $\langle value \rangle$ .

Constraint: **pdphi** > 0.

On entry, **pdr** =  $\langle value \rangle$ .

Constraint: **pdr** > 0.

On entry, **pdro** =  $\langle value \rangle$ .

Constraint: **pdro** > 0.

On entry, **pdx** =  $\langle value \rangle$ .

Constraint: **pdx** > 0.

## NE\_INT\_2

On entry, **n** =  $\langle value \rangle$  and **m** =  $\langle value \rangle$ .

Constraint: **n** ≥ **m**.

On entry, **pdfp** =  $\langle value \rangle$  and **m** =  $\langle value \rangle$ .

Constraint: **pdfp** ≥ **m**.

On entry, **pdfs** =  $\langle value \rangle$  and **m** =  $\langle value \rangle$ .

Constraint: **pdfs** ≥ **m**.

On entry, **pdphi** =  $\langle value \rangle$  and **m** =  $\langle value \rangle$ .

Constraint: **pdphi** ≥ **m**.

On entry, **pdr** =  $\langle value \rangle$  and **m** =  $\langle value \rangle$ .

Constraint: **pdr** ≥ **m**.

On entry, **pdro** =  $\langle value \rangle$  and **m** =  $\langle value \rangle$ .

Constraint: **pdro** ≥ **m**.

On entry, **pdx** =  $\langle value \rangle$  and **m** =  $\langle value \rangle$ .

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** =  $\langle value \rangle$ .

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  $\sigma$  for a ProMax rotation.

### 10.1 Program Text

```
/* nag_mv_promax (g03bdc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
/* 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,
            (unsigned)_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;
```

```

pdphi = m;
pdr = m;
pdro = m;
pdx = pdfp;

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(pdرو * 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

---