Module 28.3: nag_mv_rotation

Rotations

`nag_mv_rotation` contains a procedure to compute rotations for sets of data values.

Contents

Introduction

There are two principal reasons for using rotations:

(a) simplifying the structure to aid interpretation of derived variables, or

(b) comparing two or more data sets or sets of derived variables.

The most common type of rotations used for (a) are *orthogonal rotations*. If Λ is the p by k loading matrix from a variable-directed multivariate method, then the rotations are selected such that the elements, λ_{ij}^*, of the rotated loading matrix, Λ^*, are either relatively large or small.

For (b) *Procrustes* rotations are used. Let A and B be two l by m matrices, which can be considered as representing l points in m dimensions. One example is if A is the loading matrix from a variable-directed multivariate method and B is a hypothesised pattern matrix. In order to try to match the points in A and B there are three steps:

- translate so that centroids of both matrices are at the origin,
- find a rotation that minimizes the sum of squared distances between corresponding points of the matrices,
- scale the matrices.

For a more detailed description, see Krzanowski [1].

The procedures in `nag_fac_analysis` and `nag_canon_analysis` may reduce the dimensionality of the data from m variables to a smaller number, k, of derived variables that can adequately represent the data. In general these k derived variables will be unique only up to an *orthogonal rotation*. Therefore it may be useful to see if there exist suitable rotations of these variables that lead to a simple interpretation of the new variables in terms of the original variables.

Only one procedure is available at this release.

- `nag_orthomax` calculates the orthogonal rotations that satisfy a generalised orthomax criterion.

Procedure: nag_orthomax

1 Description

nag_orthomax calculates the orthogonal rotations for a matrix of loadings, using a generalised orthomax criterion.

2 Usage

```
USE nag_mv_rotation
CALL nag_orthomax(loading, rotated_loading  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathtt{x}(n)$' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$m \geq 1$ — the number of variables.

$k \geq 1$ — the number of factors.

3.1 Mandatory Arguments

loading(m,k) — real(kind=*wp*), intent(in)

Input: loading(i,j) must contain the loading for the ith variable on the jth factor, for $i = 1,2,\ldots,m$ and $j = 1,2,\ldots,k$.

rotated_loading(m,k) — real(kind=*wp*), intent(out)

Output: the rotated loading, Λ^*. The element rotated_loading(i,j) contains the rotated loading for the ith variable on the jth factor, for $i = 1,2,\ldots,m$ and $j = 1,2,\ldots,k$.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

rotation(k,k) — real(kind=*wp*), intent(out), optional

Output: the matrix of rotations.

gamma — real(kind=*wp*), intent(in), optional

Input: the criterion constant, γ, with $\gamma = 1.0$ giving varimax rotations and $\gamma = 0.0$ giving quartimax rotations.

Default: gamma = 0.0.

Constraints: gamma $\geq$ 0.0.

row_std — logical, intent(in), optional

Input: specifies whether the matrix of loadings is to be row standardised before rotation:

if row_std = .true., the loadings are row standardised;

if row_std = .false., the loadings are unstandardised.

Default: row_std = .true..

tol — real(kind=*wp*), intent(in), optional

Input: specifies the accuracy required. The iteration is stopped when the change in V is less than `tol`.

Default: `tol` = 0.01 `SQRT(EPSILON(1.0_`*wp*`))`.

Constraints: `EPSILON(1.0_`*wp*`)` $\leq$ `tol` $<$ 1.0.

max_iter — integer, intent(in), optional

Input: specifies the maximum number of iterations.

Default: `max_iter` = 50.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	The algorithm has failed to converge. Convergence may be obtained by increasing the values of either `tol` or `max_iter`.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Mathematical Background

Let Λ be the m by k matrix of loadings from a variable-directed multivariate method, i.e.,canonical variate analysis or factor analysis. This matrix represents the relationship between the original m variables and the k orthogonal linear combinations of these variables, the canonical variates or factors. The latter are only unique up to a rotation in the k-dimensional space they define. A rotation can then be found that simplifies the structure of the matrix of loadings, and hence the relationship between the original and the derived variables. That is the elements, λ^*_{ij}, of the rotated matrix, Λ^*, are either relatively large or small. The rotations may be found by maximizing the criterion

$$V = \sum_{j=1}^{k}\sum_{i=1}^{m}(\lambda^*_{ij})^4 - \frac{\gamma}{m}\sum_{j=1}^{k}\left(\sum_{i=1}^{m}(\lambda^*_{ij})^2\right)^2,$$

where the constant γ gives a family of rotations with $\gamma = 1$ giving varimax rotations and $\gamma = 0$ giving quartimax rotations.

It is generally advised that factor loadings should be standardised, so that the sum of squared elements for each row is one, before computing the rotations.

The matrix of rotations, R, such that $\Lambda^* = \Lambda R$, is computed using first an algorithm based on that described by Cooley and Lohnes [2], which involves the pairwise rotation of the factors.

6.2 Accuracy

The accuracy is determined by the value of the argument `tol`.

Example 1: Calculation of the Varimax Rotation

The example is taken from Lawley and Maxwell [3]. The results from a factor analysis of ten variables using three factors are input and rotated using varimax rotations without standardising rows.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_mv_rotation_ex01

  ! Example Program Text for nag_mv_rotation
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_write_mat, ONLY : nag_write_gen_mat
  USE nag_mv_rotation, ONLY : nag_orthomax
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, k, p
  REAL (wp) :: gamma
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: loading(:,:), rotated_loading(:,:), &
   rotation(:,:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_mv_rotation_ex01'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) p, k, gamma

  ALLOCATE (loading(p,k),rotated_loading(p,k), &
   rotation(k,k))              ! Allocate storage

  READ (nag_std_in,*) (loading(i,:),i=1,p)

  CALL nag_orthomax(loading,rotated_loading,rotation=rotation, &
   row_std=.FALSE.,gamma=gamma)

  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(rotated_loading,format='f9.3', &
   title='Rotated factor loading')

  WRITE (nag_std_out,*)

  CALL nag_write_gen_mat(rotation,format='f9.3',title='Rotation matrix')

  DEALLOCATE (rotation,rotated_loading,loading) ! Deallocate storage

END PROGRAM nag_mv_rotation_ex01
```

2 Program Data

```
Example Program Data for nag_mv_rotation_ex01
  10 3 1.0           : p, k, gamma
 0.788 -0.152 -0.352
 0.874  0.381  0.041
 0.814 -0.043 -0.213
 0.798 -0.170 -0.204
 0.641  0.070 -0.042
 0.755 -0.298  0.067
 0.782 -0.221  0.028
 0.767 -0.091  0.358
 0.733 -0.384  0.229
 0.771 -0.101  0.071 : loading
```

3 Program Results

```
 Example Program Results for nag_mv_rotation_ex01

 Rotated factor loading
      0.329   -0.289   -0.759
      0.849   -0.273   -0.340
      0.450   -0.327   -0.633
      0.345   -0.397   -0.657
      0.453   -0.276   -0.370
      0.263   -0.615   -0.464
      0.332   -0.561   -0.485
      0.472   -0.684   -0.183
      0.209   -0.754   -0.354
      0.423   -0.514   -0.409

 Rotation matrix
      0.633   -0.534   -0.560
      0.758    0.573    0.311
      0.155   -0.622    0.768
```

References

[1] Krzanowski W J (1990) *Principles of Multivariate Analysis* Oxford University Press

[2] Cooley W C and Lohnes P R (1971) *Multivariate Data Analysis* Wiley

[3] Lawley D N and Maxwell A E (1971) *Factor Analysis as a Statistical Method* Butterworths (2nd Edition)