nag_mv_procustes (g03bcc)

1. Purpose

nag_mv_procustes (g03bcc) computes Procrustes rotations in which an orthogonal rotation is found so that a transformed matrix best matches a target matrix.

2. Specification

```
#include <nag.h>
#include <nagg03.h>

void nag_mv_procustes(Nag_TransNorm stand, Nag_RotationScale pscale, Integer n,
                      Integer m, double x[], Integer tdx, double y[], Integer tdy,
                      double yhat[], double r[], Integer tdr, double *alpha, double *rss,
                      double res[], NagError *fail)
```

3. Description

Let X and Y be n by m matrices. They can be considered as representing sets of n points in an m-dimensional space. The X matrix may be a matrix of loadings from say factor or canonical variate analysis, and the Y matrix may be a postulated pattern matrix or the loadings from a different sample. The problem is to relate the two sets of points without disturbing the relationships between the points in each set. This can be achieved by translating, rotating and scaling the sets of points. The Y matrix is considered as the target matrix and the X matrix is rotated to match that matrix.

First the two sets of points are translated so that their centroids are at the origin to give X_c and Y_c, i.e., the matrices will have zero column means. Then the rotation of the translated X_c matrix which minimizes the sum of squared distances between corresponding points in the two sets is found. This is computed from the singular value decomposition of the matrix:

$$X_c^T Y_c = UDV^T,$$

where U and V are orthogonal matrices and D is a diagonal matrix. The matrix of rotations, R, is computed as:

$$R = UV^T.$$

After rotation, a scaling or dilation factor, α, may be estimated by least-squares. Thus, the final set of points that best match Y_c is given by:

$$\hat{Y}_c = \alpha X_c R.$$

Before rotation, both sets of points may be normalized to have unit sums of squares or the X matrix may be normalized to have the same sum of squares as the Y matrix. After rotation, the results may be translated to the original Y centroid.

The ith residual, r_i, is given by the distance between the point given in the ith row of Y and the point given in the ith row of $\hat{Y}$. The residual sum of squares is also computed.

4. Parameters

stand

Input: indicates if translation/normalization is required.

If **stand** = **Nag_NoTransNorm** there is no translation or normalization.

If **stand** = **Nag_Orig** there is translation to the origin.

If **stand** = **Nag_OrigCentroid** there is translation to the origin and then to the Y centroid after rotation.

If **stand** = **Nag_Norm** there is unit normalization.

If **stand** = **Nag_OrigNorm** there is translation and normalization.

If **stand** = **Nag_OrigNormCentroid** there is translation and normalization to Y scale, then translation to the Y centroid after rotation.

Constraint: **stand** = **Nag_NoTransNorm**, **Nag_Orig**, **Nag_OrigCentroid**, **Nag_Norm**, **Nag_OrigNorm** or **Nag_OrigNormCentroid**.

pscale
Input: indicates if least-squares scaling is applied after rotation.
If **pscale** = **Nag_LsqScale** then scaling is to be applied.
If **pscale** = **Nag_NotLsqScale** then no scaling is applied.
Constraint: **pscale** = **Nag_LsqScale** or **Nag_NotLsqScale**.

n
Input: the number of points, n.
Constraint: $\mathbf{n} \geq 1$.

m
Input: the number of dimensions, m.
Constraints:
$\mathbf{m} \geq 1$.
$\mathbf{m} \leq \mathbf{n}$.

x[n][tdx]
Input: the matrix to be rotated, X.
Output: if **stand** = **Nag_NoTransNorm**, then **x** will be unchanged.
If **stand** = **Nag_Orig**, **Nag_OrigCentroid**, **Nag_OrigNorm** or **Nag_OrigNormCentroid**, then **x** will be translated to have zero column means.
If **stand** = **Nag_Norm** or **Nag_OrigNorm**, then **x** will be scaled to have unit sum of squares.
If **stand** = **Nag_OrigNormCentroid**, then **x** will be scaled to have the same sum of squares as **y**.

tdx
Input: the last dimension of the array **x** as declared in the calling program.
Constraint: $\mathbf{tdx} \geq \mathbf{m}$.

y[n][tdy]
Input: the target matrix, Y.
Output: if **stand** = **Nag_NoTransNorm**, then **y** will be unchanged.
If **stand** = **Nag_Orig** or **Nag_OrigNorm**, then **y** will be translated to have zero column means.
If **stand** = **Nag_Norm** or **Nag_OrigNorm**, then **y** will be scaled to have unit sum of squares.
If **stand** = **Nag_OrigCentroid** or **Nag_OrigNormCentroid**, then **y** will be translated and then after rotation, translated back. The output **y** should be the same as the input **y** except for rounding errors.

tdy
Input: the last dimension of the arrays **y** and **yhat** as declared in the calling program.
Constraint: $\mathbf{tdy} \geq \mathbf{m}$.

yhat[n][tdy]
Output: the fitted matrix, $\hat{Y}$.

r[m][tdr]
Output: the matrix of rotations, R, see Section 6.

tdr
Input: the last dimension of the array **r** as declared in the calling program.
Constraint: $\mathbf{tdr} \geq \mathbf{m}$.

alpha
Output: if **pscale** = **Nag_LsqScale** the scaling factor, α; otherwise **alpha** is not set.

rss
Output: the residual sum of squares.

res[n]
Output: the residuals, r_i, for $i = 1, 2, \ldots, n$.

fail
The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings

NE_BAD_PARAM
On entry, parameter **stand** had an illegal value.
On entry, parameter **pscale** had an illegal value.

NE_INT_ARG_LT
On entry, **n** must not be less than 1: **n** = ⟨*value*⟩.
On entry, **m** must not be less than 1: **m** = ⟨*value*⟩.

NE_2_INT_ARG_LT
On entry, **tdx** = ⟨*value*⟩ while **m** = ⟨*value*⟩.
These parameters must satisfy **tdx** ≥ **m**.
On entry, **tdy** = ⟨*value*⟩ while **m** = ⟨*value*⟩.
These parameters must satisfy **tdy** ≥ **m**.
On entry, **tdr** = ⟨*value*⟩ while **m** = ⟨*value*⟩.
These parameters must satisfy **tdr** ≥ **m**.

NE_2_INT_ARG_GT
On entry, **m** = ⟨*value*⟩ while **n** = ⟨*value*⟩.
These parameters must satisfy **m** ≤ **n**.

NE_SVD_NOT_CONV
The singular value decomposition has failed to converge.
This is an unlikely error exit.

NE_NORM_ZERO_PTS
On entry, either **x** or **y** contains only zero-points (possibly after translation) when normalization is to be applied.

NE_LSQ_SCAL_ZERO_PTS
The fitted matrix $\hat{Y}$, contains only zero-points when least-squares scaling is applied.

NE_ALLOC_FAIL
Memory allocation failed.

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 consult NAG for assistance.

6. Further Comments

Note that if the matrix $X_c^T Y$ is not of full rank, then the matrix of rotations, R, may not be unique even if there is a unique solution in terms of the rotated matrix, $\hat{Y}_c$. The matrix R may also include reflections as well as pure rotations, see Krzanowski (1990).

If the column dimensions of the X and Y matrices are not equal, the smaller of the two should be supplemented by columns of zeros. Adding a column of zeros to both X and Y will have the effect of allowing reflections as well as rotations.

6.1. Accuracy

The accuracy of the calculation of the rotation matrix largely depends upon the singular value decomposition. See nag_real_svd (f02wec) for further details.

6.2. References

Krzanowski W J (1990) *Principles of Multivariate Analysis* Oxford University Press.
Lawley D N and Maxwell A E (1971) *Factor Analysis as a Statistical Method* Butterworths (2nd Edition).

7. See Also

nag_real_svd (f02wec)

8. Example

Three points representing the vertices of a triangle in two dimensions are input. The points are translated and rotated to match the triangle given by (0,0),(1,0),(0,2) and scaling is applied after rotation. The target matrix and fitted matrix are printed along with additional information.

8.1. Program Text

```
/* nag_mv_procustes (g03bcc) Example Program.
 *
 * Copyright 1998 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 *
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagg03.h>

#define NMAX 3
#define MMAX 2

main()
{
  double r[MMAX][MMAX], res[NMAX],
  x[NMAX][MMAX], y[NMAX][MMAX], yhat[NMAX][MMAX];
  double alpha;
  double  rss;

  Integer i, j, m, n;
  Integer tdx = MMAX, tdr = MMAX, tdy = MMAX;

  char char_scale[2];
  char char_stand[2];

  Nag_TransNorm stand;
  Nag_RotationScale scale;

  Vprintf("g03bcc Example Program Results\n\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n]");
  Vscanf("%ld",&n);
  Vscanf("%ld",&m);
  Vscanf("%s",char_stand);
  Vscanf("%s",char_scale);

  if (n <= NMAX && m <= MMAX)
    {
      for (i = 0; i < n; ++i)
        {
          for (j = 0; j < m; ++j)
            Vscanf("%lf",&x[i][j]);
        }
      for (i = 0; i < n; ++i)
        {
          for (j = 0; j < m; ++j)
```

```
            Vscanf("%lf",&y[i][j]);
        }

      if (*char_stand == 'N')
        {
          stand = Nag_NoTransNorm;
        }
      else if (*char_stand == 'Z')
        {
          stand = Nag_Orig;
        }
      else if (*char_stand == 'C')
        {
          stand = Nag_OrigCentroid;
        }
      else if (*char_stand == 'U')
        {
          stand = Nag_Norm;
        }
      else if (*char_stand == 'S')
        {
          stand = Nag_OrigNorm;
        }
      else if (*char_stand == 'M')
        {
          stand = Nag_OrigNormCentroid;
        }

      if (*char_scale == 'S')
        {
          scale = Nag_LsqScale;
        }
      else if (*char_scale == 'U')
        {
          scale = Nag_NotLsqScale;
        }

      g03bcc(stand, scale, n, m, (double *)x, tdx, (double *)y, tdy,
             (double *)yhat, (double *)r, tdr,
             &alpha, &rss, res, NAGERR_DEFAULT);

      Vprintf("\n          Rotation Matrix\n\n");
      for (i = 0; i < m; ++i)
        {
          for (j = 0; j < m; ++j)
            Vprintf("  %7.3f  ",r[i][j]);
          Vprintf("\n");
        }
      if (*char_scale == 'S' || *char_scale == 's')
        {
          Vprintf("\n%s%10.3f\n"," Scale factor = ",alpha);
        }
      Vprintf("\n         Target Matrix \n\n");
      for (i = 0; i < n; ++i)
        {
          for (j = 0; j < m; ++j)
            Vprintf("  %7.3f  ",y[i][j]);
          Vprintf("\n");
        }
      Vprintf("\n         Fitted Matrix\n\n");
      for (i = 0; i < n; ++i)
        {
          for (j = 0; j < m; ++j)
            Vprintf("  %7.3f  ",yhat[i][j]);
          Vprintf("\n");
        }
      Vprintf("\n%s%10.3f\n","RSS = ",rss);
      exit(EXIT_SUCCESS);
    }
  else
```

```
    {
      Vprintf("Incorrect input value of n or m.\n");
      exit(EXIT_FAILURE);
    }
}
```

8.2. Program Data

```
g03bcc Example Program Data
 3 2 C S
 0.63 0.58
 1.36 0.39
 1.01 1.76
 0.0 0.0
 1.0 0.0
 0.0 2.0
```

8.3. Program Results

```
g03bcc Example Program Results


          Rotation Matrix

    0.967      0.254
   -0.254      0.967

 Scale factor =      1.556

         Target Matrix

    0.000      0.000
    1.000      0.000
    0.000      2.000

         Fitted Matrix

   -0.093      0.024
    1.080      0.026
    0.013      1.950

RSS =      0.019
```