# NAG Library Function Document

# nag_mv_ordinal_multidimscale (g03fcc)

## 1    Purpose

nag_mv_ordinal_multidimscale (g03fcc) performs non-metric (ordinal) multidimensional scaling.

## 2    Specification

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

```
void nag_mv_ordinal_multidimscale (Nag_ScaleCriterion type, Integer n,
      Integer ndim, const double d[], double x[], Integer tdx, double *stress,
      double dfit[], Nag_E04_Opt *options, NagError *fail)
```

## 3    Description

For a set of $n$ objects, a distance or dissimilarity matrix $D$ can be calculated such that $d_{ij}$ is a measure of how 'far apart' objects $i$ and $j$ are. If $p$ variables $x_k$ have been recorded for each observation this measure may be based on Euclidean distance, $d_{ij} = \sum_{k=1}^{p} (x_{ki} - x_{kj})^2$, or some other calculation such as the number of variables for which $x_{kj} \neq x_{ki}$. Alternatively, the distances may be the result of a subjective assessment. For a given distance matrix, multidimensional scaling produces a configuration of $n$ points in a chosen number of dimensions, $m$, such that the distance between the points in some way best matches the distance matrix. For some distance measures, such as Euclidean distance, the size of distance is meaningful, for other measures of distance all that can be said is that one distance is greater or smaller than another. For the former, metric scaling can be used, see nag_mv_prin_coord_analysis (g03fac), for the latter, a non-metric scaling is more appropriate.

For non-metric multidimensional scaling, the criterion used to measure the closeness of the fitted distance matrix to the observed distance matrix is known as *stress*. *stress* is given by,

$$\sqrt{\frac{\sum_{i=1}^{n} \sum_{j=1}^{i-1} \left( \hat{d}_{ij} - \tilde{d}_{ij} \right)^2}{\sum_{i=1}^{n} \sum_{j=1}^{i-1} \hat{d}_{ij}^2}}$$

where $\hat{d}_{ij}^2$ is the Euclidean squared distance between points $i$ and $j$ and $\tilde{d}_{ij}$ is the fitted distance obtained when $\hat{d}_{ij}$ is monotonically regressed on $d_{ij}$, that is, $\tilde{d}_{ij}$ is monotonic relative to $d_{ij}$ and is obtained from $\hat{d}_{ij}$ with the smallest number of changes. So *stress* is a measure of by how much the set of points preserve the order of the distances in the original distance matrix. Non-metric multidimensional scaling seeks to find the set of points that minimize the *stress*.

An alternate measure is squared *stress*, $SSTRESS$,

$$\sqrt{\frac{\sum_{i=1}^{n} \sum_{j=1}^{i-1} \left( \hat{d}_{ij}^2 - \tilde{d}_{ij}^2 \right)^2}{\sum_{i=1}^{n} \sum_{j=1}^{i-1} \hat{d}_{ij}^4}}$$

in which the distances in *stress* are replaced by squared distances.

In order to perform a non-metric scaling, an initial configuration of points is required. This can be obtained from principal coordinate analysis, see nag_mv_prin_coord_analysis (g03fac). Given an initial configuration, nag_mv_ordinal_multidimscale (g03fcc) uses the optimization function nag_opt_conj_grad (e04dgc) to find the configuration of points that minimizes *stress* or $SSTRESS$. The function nag_opt_conj_grad (e04dgc) uses a conjugate gradient algorithm. nag_mv_ordinal_multidimscale (g03fcc) will find an optimum that may only be a local optimum, to be more sure of finding a global optimum several different initial configurations should be used; these can be obtained by randomly perturbing the original initial configuration using functions from the g05 Chapter Introduction.

## 4    References

Chatfield C and Collins A J (1980) *Introduction to Multivariate Analysis* Chapman and Hall

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

## 5    Arguments

1:    **type** – Nag_ScaleCriterion                                                                          *Input*

On entry: indicates whether *stress* or $SSTRESS$ is to be used as the criterion.

**type** = Nag_Stress
        *stress* is used.

**type** = Nag_SStress
        $SSTRESS$ is used.

*Constraint*: **type** = Nag_Stress or Nag_SStress.

2:    **n** – Integer                                                                                         *Input*

*On entry*: the number of objects in the distance matrix , $n$.

*Constraint*: **n** > **ndim**.

3:    **ndim** – Integer                                                                                      *Input*

*On entry*: the number of dimensions used to represent the data, $m$.

*Constraint*: **ndim** ≥ 1.

4:    **d**[**n** × (**n** − **1**)/**2**] – const double                                                     *Input*

*On entry*: the lower triangle of the distance matrix $D$ stored packed by rows. That is **d**$[(i-1) \times (i-2)/2 + j - 1]$ must contain $d_{ij}$, for $i = 2, 3, \ldots, n$ and $j = 1, 2, \ldots, i - 1$. If $d_{ij}$ is missing then set $d_{ij} < 0$; For further comments on missing values see Section 9.

5:    **x**[**n** × **tdx**] – double                                                                  *Input/Output*

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

*On entry*: the $i$th row must contain an initial estimate of the coordinates for the $i$th point, $i = 1, 2, \ldots, n$. One method of computing these is to use nag_mv_prin_coord_analysis (g03fac).

*On exit*: the $i$th row contains $m$ coordinates for the $i$th point, $i = 1, 2, \ldots, n$.

6:    **tdx** – Integer                                                                                       *Input*

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

*Constraint*: **tdx** ≥ **ndim**.

7: **stress** – double * *Output*

On exit: the value of $stress$ or $SSTRESS$ at the final iteration.

8: **dfit**$[2 \times \mathbf{n} \times (\mathbf{n} - 1)]$ – double *Output*

*On exit*: auxiliary outputs. If **type** = Nag_Stress, the first $n(n-1)/2$ elements contain the distances, $\hat{d}_{ij}$, for the points returned in **x**, the second set of $n(n-1)/2$ contains the distances $\hat{d}_{ij}$ ordered by the input distances, $d_{ij}$, the third set of $n(n-1)/2$ elements contains the monotonic distances, $\tilde{d}_{ij}$, ordered by the input distances, $d_{ij}$ and the final set of $n(n-1)/2$ elements contains fitted monotonic distances, $\tilde{d}_{ij}$, for the points in **x**. The $\tilde{d}_{ij}$ corresponding to distances which are input as missing are set to zero. If **type** = Nag_SStress, the results are as above except that the squared distances are returned.

Each distance matrix is stored in lower triangular packed form in the same way as the input matrix $D$.

9: **options** – Nag_E04_Opt * *Input/Output*

*On entry/exit*: a pointer to a structure of type Nag_E04_Opt whose members are optional arguments for nag_opt_conj_grad (e04dgc). These structure members offer the means of adjusting some of the argument values of the algorithm and on output will supply further details of the results. You are referred to the nag_opt_conj_grad (e04dgc) document for further details.

The default values used by nag_mv_ordinal_multidimscale (g03fcc) when the options argument is set to the NAG defined null pointer, E04_DEFAULT, are as follows:

    **optim_tol** = 0.00001;

    **print_level** = Nag_NoPrint;

    **list** = Nag_FALSE;

    **verify_grad** = Nag_FALSE;

    **max_iter** = max$(50, \mathbf{n} \times \mathbf{ndim})$.

If a different value is required for any of these four structure members or if other options available in nag_opt_conj_grad (e04dgc) are to be used, then the structure **options** should be declared and initialized by a call to nag_opt_init (e04xxc) and supplied as an argument to nag_mv_ordinal_multidimscale (g03fcc). In this case, the structure members listed above except for **list** will have the default values as specified above; **list** = Nag_TRUE in this case.

10: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

# 6 Error Indicators and Warnings

**NE_2_INT_ARG_LE**

On entry, $\mathbf{n} = \langle value \rangle$ while $\mathbf{ndim} = \langle value \rangle$. These arguments must satisfy $\mathbf{n} > \mathbf{ndim}$.

**NE_2_INT_ARG_LT**

On entry, $\mathbf{tdx} = \langle value \rangle$ while $\mathbf{ndim} = \langle value \rangle$. These arguments must satisfy $\mathbf{tdx} \geq \mathbf{ndim}$.

**NE_ALLOC_FAIL**

Dynamic memory allocation failed.

**NE_BAD_PARAM**

On entry, argument **type** had an illegal value.

**NE_INT_ARG_LT**

On entry, **ndim** $= \langle value \rangle$.
Constraint: **ndim** $\geq 1$.

**NE_INTERNAL_ERROR**

Additional error messages are output if the optimization fails to converge or if the options are set incorrectly, Details of these can be found in the nag_opt_conj_grad (e04dgc) document.

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_NEG_OR_ZERO_ARRAY**

All elements of array **d** $\leq 0.0$.
Constraint: At least one element of **d** must be positive.

# 7    Accuracy

After a successful optimization, the relative accuracy of *stress* should be approximately $\epsilon$, as specified by **optim_tol**.

# 8    Parallelism and Performance

Not applicable.

# 9    Further Comments

Missing values in the input distance matrix can be specified by a negative value and providing there are not more than about two thirds of the values missing, the algorithm may still work. However, the function nag_mv_prin_coord_analysis (g03fac) does not allow for missing values so an alternative method of obtaining an initial set of coordinates is required. It may be possible to estimate the missing values with some form of average and then use nag_mv_prin_coord_analysis (g03fac) to give an initial set of coordinates.

# 10    Example

The data, given by Krzanowski (1990), are dissimilarities between water vole populations in Europe. Initial estimates are provided by the first two principal coordinates computed.

## 10.1 Program Text

```
/* nag_mv_ordinal_multidimscale (g03fcc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 * Mark 6 revised, 2000.
 * Mark 7 revised, 2001.
 * Mark 8 revised, 2004.
 *
 */

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

#define NMAX  14
#define NNMAX NMAX*(NMAX-1)/2

#define X(I, J) x[(I-1)*tdx + (J-1)]
```

```
#define XTMP(I) xtmp[(I) -1]
#define YTMP(I) ytmp[(I) -1]

int main(void)
{
  Integer            exit_status = 0, i, j, n, ndim, nn, tdx = NMAX;
  double             *d = 0, *dfit = 0, *e = 0, stress, *x = 0;
  char               nag_enum_arg[40];
  Nag_ScaleCriterion type;
  NagError           fail;

  INIT_FAIL(fail);

  printf(
          "nag_mv_ordinal_multidimscale (g03fcc) Example Program Results\n\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"", &n);
#else
  scanf("%"NAG_IFMT"", &n);
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"", &ndim);
#else
  scanf("%"NAG_IFMT"", &ndim);
#endif
#ifdef _WIN32
  scanf_s("%39s", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  type = (Nag_ScaleCriterion) nag_enum_name_to_value(nag_enum_arg);

  if (ndim >= 1 && n > ndim)
    {
      if (!(d = NAG_ALLOC(n*(n-1)/2, double)) ||
          !(dfit = NAG_ALLOC(4*(n*(n-1)/2), double)) ||
          !(e = NAG_ALLOC(n, double)) ||
          !(x = NAG_ALLOC(n*n, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
      tdx = n;
    }
  else
    {
      printf("Invalid ndim or n.\n");
      exit_status = 1;
      return exit_status;
    }
  nn = n * (n - 1) / 2;
  for (i = 1; i <= nn; ++i)
#ifdef _WIN32
    scanf_s("%lf", &d[i-1]);
#else
    scanf("%lf", &d[i-1]);
#endif

  /* nag_mv_prin_coord_analysis (g03fac).
   * Principal co-ordinate analysis
```

```
    */
  nag_mv_prin_coord_analysis(Nag_LargeEigVals, n, d, ndim, x, tdx, e, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_mv_prin_coord_analysis (g03fac).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  /* nag_mv_ordinal_multidimscale (g03fcc).
   * Multidimensional scaling
   */
  nag_mv_ordinal_multidimscale(type, n, ndim, d, x, tdx, &stress, dfit,
                               E04_DEFAULT, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_mv_ordinal_multidimscale (g03fcc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  printf("\n          STRESS = %13.4e\n\n", stress);
  printf("Co-ordinates\n\n");

  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= ndim; ++j)
        printf("%10.4f", X(i, j));
      printf("\n");
    }

 END:
  NAG_FREE(d);
  NAG_FREE(dfit);
  NAG_FREE(e);
  NAG_FREE(x);

  return exit_status;
}
```

## 10.2 Program Data

```
nag_mv_ordinal_multidimscale (g03fcc) Example Program Data

14 2 Nag_Stress

0.099
0.033 0.022
0.183 0.114 0.042
0.148 0.224 0.059 0.068
0.198 0.039 0.053 0.085 0.051
0.462 0.266 0.322 0.435 0.268 0.025
0.628 0.442 0.444 0.406 0.240 0.129 0.014
0.113 0.070 0.046 0.047 0.034 0.002 0.106 0.129
0.173 0.119 0.162 0.331 0.177 0.039 0.089 0.237 0.071
0.434 0.419 0.339 0.505 0.469 0.390 0.315 0.349 0.151 0.430
0.762 0.633 0.781 0.700 0.758 0.625 0.469 0.618 0.440 0.538 0.607
0.530 0.389 0.482 0.579 0.597 0.498 0.374 0.562 0.247 0.383 0.387 0.084
0.586 0.435 0.550 0.530 0.552 0.509 0.369 0.471 0.234 0.346 0.456 0.090 0.038
```

## 10.3 Program Results

```
nag_mv_ordinal_multidimscale (g03fcc) Example Program Results


          STRESS =    1.2557e-01

Co-ordinates
```

```
    0.2060     0.2439
    0.1063     0.1418
    0.2224     0.0817
    0.3032     0.0355
    0.2645    -0.0698
    0.1554    -0.0435
   -0.0070    -0.1612
    0.0749    -0.3275
    0.0488     0.0289
    0.0124    -0.0267
   -0.1649    -0.2500
   -0.5073     0.1267
   -0.3093     0.1590
   -0.3498     0.0700
```