NAG Library Function Document

nag_mv_dendrogram (g03ehc)

1 Purpose

nag_mv_dendrogram (g03ehc) produces a dendrogram from the results of nag_mv_hierar_cluster_analysis (g03ecc).

2 Specification

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

```
void nag_mv_dendrogram (Nag_DendOrient orient, Integer n,
     const double dord[], double dmin, double dstep, Integer nsym, char ***c,
     NagError *fail)
```

3 Description

Hierarchical cluster analysis, as performed by nag_mv_hierar_cluster_analysis (g03ecc) can be represented by a tree that shows at which distance the clusters merge. Such a tree is known as a dendrogram. See Everitt (1974) and Krzanowski (1990) for examples of dendrograms. A simple example is,

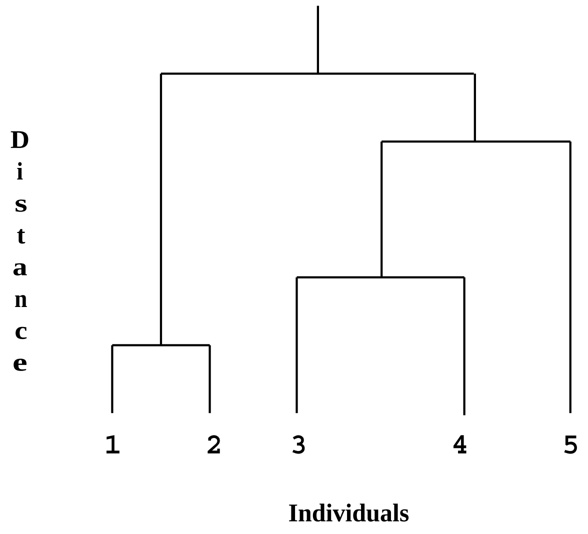

Figure 1

The end-points of the dendrogram represent the objects that have been clustered. They should be in a suitable order as given by nag_mv_hierar_cluster_analysis (g03ecc). Object 1 is always the first object. In the example above the height represents the distance at which the clusters merge.

The dendrogram is produced in an array of character pointers using the ordering and distances provided by nag_mv_hierar_cluster_analysis (g03ecc). Suitable characters are used to represent parts of the tree.

There are four possible orientations for the dendrogram. The example above has the end-points at the bottom of the diagram which will be referred to as south. If the dendrogram was the other way around with the end-points at the top of the diagram then the orientation would be north. If the end-points are at the left-hand or right-hand side of the diagram the orientation is west or east. Different symbols are used for east/west and north/south orientations.

4 References

Everitt B S (1974) *Cluster Analysis* Heinemann

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

5 Arguments

1: **orient** – Nag_DendOrient *Input*

On entry: indicates which orientation the dendrogram is to take.

orient = Nag_DendNorth
The end-points of the dendrogram are to the north.

orient = Nag_DendSouth
The end-points of the dendrogram are to the south.

orient = Nag_DendEast
The end-points of the dendrogram are to the east.

orient = Nag_DendWest
The end-points of the dendrogram are to the west.

Constraint: **orient** = Nag_DendNorth, Nag_DendSouth, Nag_DendEast or Nag_DendWest.

2: **n** – Integer *Input*

On entry: the number of objects in the cluster analysis.

Constraint: $\mathbf{n} \geq 2$.

3: **dord**[**n**] – const double *Input*

On entry: the array **dord** as output by nag_mv_hierar_cluster_analysis (g03ecc). **dord** contains the distances, in dendrogram order, at which clustering takes place.

Constraint: $\mathbf{dord}[\mathbf{n}-1] \geq \mathbf{dord}[i-1]$, for $i = 1, 2, \ldots, \mathbf{n}-1$.

4: **dmin** – double *Input*

On entry: the clustering distance at which the dendrogram begins.

Constraint: $\mathbf{dmin} \geq 0.0$.

5: **dstep** – double *Input*

On entry: the distance represented by one symbol of the dendrogram.

Constraint: $\mathbf{dstep} > 0.0$.

6: **nsym** – Integer *Input*

On entry: the number of character positions used in the dendrogram. Hence the clustering distance at which the dendrogram terminates is given by $\mathbf{dmin} + \mathbf{nsym} \times \mathbf{dstep}$.

Constraint: $\mathbf{nsym} \geq 1$.

7: **c** – char *** *Input/Output*

On entry/exit: a pointer to an array of character pointers, containing consecutive lines of the dendrogram. The memory to which **c** points is allocated internally.

orient = Nag_DendNorth or Nag_DendSouth
The number of lines in the dendrogram is **nsym**.

orient = Nag_DendEast or Nag_DendWest
The number of lines in the dendrogram is **n**.

The storage pointed to by this pointer must be freed using nag_mv_dend_free (g03xzc).

8: **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_BAD_PARAM

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

NE_DENDROGRAM_ARRAY

On entry, $\mathbf{n} = \langle value \rangle$, $\mathbf{dord}[\langle value \rangle] = \langle value \rangle$.
Constraint: $\mathbf{dord}[\mathbf{n} - 1] \geq \mathbf{dord}[i - 1]$, $i = 1, 2, \ldots, \mathbf{n} - 1$.

NE_INT_ARG_LT

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 2$.

On entry, $\mathbf{nsym} = \langle value \rangle$.
Constraint: $\mathbf{nsym} \geq 1$.

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, **dstep** must not be less than or equal to 0.0: $\mathbf{dstep} = \langle value \rangle$.

NE_REAL_ARG_LT

On entry, **dmin** must not be less than 0.0: $\mathbf{dmin} = \langle value \rangle$.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_mv_dendrogram (g03ehc) is not threaded in any implementation.

9 Further Comments

The scale of the dendrogram is controlled by **dstep**. The smaller the value of **dstep** the greater the amount of detail that will be given. However, **nsym** will have to be larger to give the full dendrogram. The range of distances represented by the dendrogram is **dmin** to $\mathbf{nsym} \times \mathbf{dstep}$. The values of **dmin**, **dstep** and **nsym** can thus be set so that only part of the dendrogram is produced.

The dendrogram does not include any labelling of the objects. You can print suitable labels using the ordering given by the array **iord** returned by nag_mv_hierar_cluster_analysis (g03ecc).

10 Example

Data consisting of three variables on five objects are read in. Euclidean squared distances are computed using nag_mv_distance_mat (g03eac) and median clustering performed by nag_mv_hierar_cluster_analysis (g03ecc). nag_mv_dendrogram (g03ehc) is used to produce a dendrogram with orientation east and a dendrogram with orientation south. The two dendrograms are printed.

Note the use of nag_mv_dend_free (g03xzc) to free the memory allocated internally to the character array pointed to by **c**.

10.1 Program Text

```
/* nag_mv_dendrogram (g03ehc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 *
 */

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

#define X(I, J) x[(I) *tdx + J]
int main(void)
{
  Integer exit_status = 0, i, j, m, n, nsym, tdx;
  Integer *ilc = 0, *iord = 0, *isx = 0, *iuc = 0;
  char **c = 0;
  double *cd = 0, *d = 0, dmin_, *dord = 0, dstep, *s = 0, *x = 0;
  char nag_enum_arg[40];
  Nag_ClusterMethod method;
  Nag_DistanceType dist;
  Nag_MatUpdate update;
  Nag_VarScaleType scale;
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_mv_dendrogram (g03ehc) Example Program Results\n\n");

#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 "", &m);
#else
  scanf("%" NAG_IFMT "", &m);
#endif

  if (n >= 2 && m >= 1) {
    if (!(cd = NAG_ALLOC(n - 1, double)) ||
        !(d = NAG_ALLOC(n * (n - 1) / 2, double)) ||
        !(dord = NAG_ALLOC(n, double)) ||
        !(s = NAG_ALLOC(m, double)) ||
        !(x = NAG_ALLOC(n * m, double)) ||
        !(ilc = NAG_ALLOC(n - 1, Integer)) ||
        !(iord = NAG_ALLOC(n, Integer)) ||
        !(isx = NAG_ALLOC(m, Integer)) || !(iuc = NAG_ALLOC(n - 1, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
    tdx = m;
  }
  else {
    printf("Invalid n or m.\n");
    exit_status = 1;
    return exit_status;
  }
#ifdef _WIN32
  scanf_s("%39s", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%39s", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  method = (Nag_ClusterMethod) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s("%39s", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%39s", nag_enum_arg);
#endif
  update = (Nag_MatUpdate) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s("%39s", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%39s", nag_enum_arg);
#endif
  dist = (Nag_DistanceType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s("%39s", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%39s", nag_enum_arg);
#endif
  scale = (Nag_VarScaleType) nag_enum_name_to_value(nag_enum_arg);

  for (j = 0; j < n; ++j) {
    for (i = 0; i < m; ++i)
#ifdef _WIN32
      scanf_s("%lf", &X(j, i));
#else
      scanf("%lf", &X(j, i));
#endif
  }
  for (i = 0; i < m; ++i)
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &isx[i]);
#else
    scanf("%" NAG_IFMT "", &isx[i]);
#endif
  for (i = 0; i < m; ++i)
#ifdef _WIN32
    scanf_s("%lf", &s[i]);
#else
    scanf("%lf", &s[i]);
#endif
#ifdef _WIN32
  scanf_s("%lf", &dmin_);
#else
  scanf("%lf", &dmin_);
#endif
#ifdef _WIN32
  scanf_s("%lf", &dstep);
#else
  scanf("%lf", &dstep);
#endif
```

```
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "", &nsym);
#else
  scanf("%" NAG_IFMT "", &nsym);
#endif

  /* Compute the distance matrix */
  /* nag_mv_distance_mat (g03eac).
   * Compute distance (dissimilarity) matrix
   */
  nag_mv_distance_mat(update, dist, scale, n, m, x, tdx, isx, s, d, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_mv_distance_mat (g03eac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Perform clustering */
  /* nag_mv_hierar_cluster_analysis (g03ecc).
   * Hierarchical cluster analysis
   */
  nag_mv_hierar_cluster_analysis(method, n, d, ilc, iuc, cd, iord, dord,
                                 &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_mv_hierar_cluster_analysis (g03ecc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Produce dendrograms */
  /* nag_mv_dendrogram (g03ehc).
   * Construct dendrogram following
   * nag_mv_hierar_cluster_analysis (g03ecc)
   */
  nag_mv_dendrogram(Nag_DendEast, n, dord, dmin_, dstep, nsym, &c, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_mv_dendrogram (g03ehc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  printf("\nDendrogram, Orientation East\n\n");
  for (i = 0; i < n; i++) {
    printf("%s\n", c[i]);
  }

#ifdef _WIN32
  scanf_s("%lf", &dmin_);
#else
  scanf("%lf", &dmin_);
#endif
#ifdef _WIN32
  scanf_s("%lf", &dstep);
#else
  scanf("%lf", &dstep);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "", &nsym);
#else
  scanf("%" NAG_IFMT "", &nsym);
#endif
  /* nag_mv_dend_free (g03xzc).
   * Frees memory allocated to the dendrogram array in
   * nag_mv_dendrogram (g03ehc)
   */
  nag_mv_dend_free(&c);
  /* nag_mv_dendrogram (g03ehc), see above. */
  nag_mv_dendrogram(Nag_DendSouth, n, dord, dmin_, dstep, nsym, &c, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_mv_dendrogram (g03ehc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  printf("\n\n Dendrogram, Orientation South\n\n");
  for (i = 0; i < nsym; i++) {
    printf("%s\n", c[i]);
  }
  /* nag_mv_dend_free (g03xzc), see above. */
  nag_mv_dend_free(&c);

END:
  NAG_FREE(cd);
  NAG_FREE(d);
  NAG_FREE(dord);
  NAG_FREE(s);
  NAG_FREE(x);
  NAG_FREE(ilc);
  NAG_FREE(iord);
  NAG_FREE(isx);
  NAG_FREE(iuc);

  return exit_status;
}
```

10.2 Program Data

```
nag_mv_dendrogram (g03ehc) Example Program Data
5 3
Nag_Median
Nag_NoMatUp Nag_DistSquared Nag_NoVarScale
1 1.0 1.0
2 1.0 2.0
3 6.0 3.0
4 8.0 2.0
5 8.0 0.0
0 1 1
1 1 1
0.0 1.1 40
0.0 1.0 40
```

10.3 Program Results

```
nag_mv_dendrogram (g03ehc) Example Program Results


Dendrogram, Orientation East


        ...............................(
       (                         .......
       (                        (    ...
       (........................(...(...


 Dendrogram, Orientation South




   ----------
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I        I
   I  ------*
   I  I     I
   I  I     I
   I  I     I
   I  I  ---*
   I  I  I  I
   I  I  I  I
---*  I  I  I
I  I  I  I  I
```