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, 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,

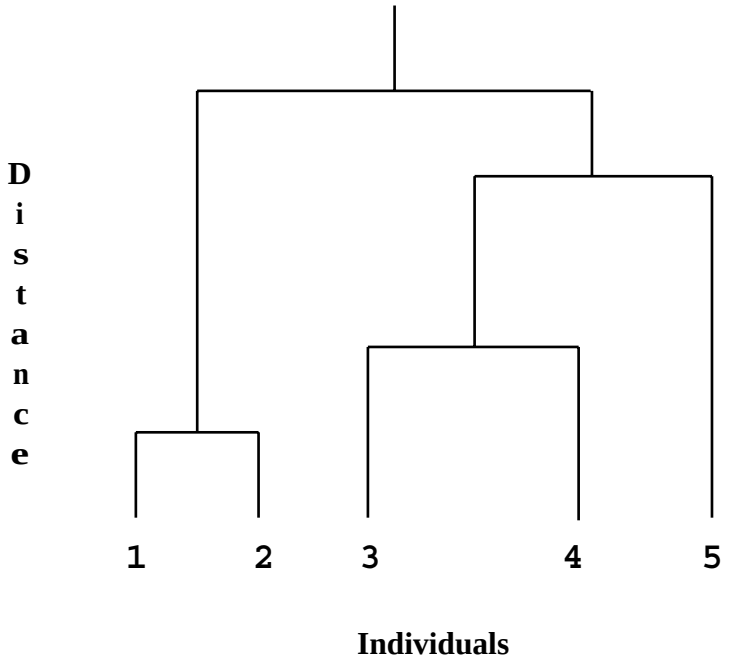


Figure 1

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. Parameters

orient

Input: indicates which orientation the dendrogram is to take.

If **orient** = **Nag_DendNorth**, then the end-points of the dendrogram are to the north.
If **orient** = **Nag_DendSouth**, then the end-points of the dendrogram are to the south.
If **orient** = **Nag_DendEast**, then the end-points of the dendrogram are to the east.
If **orient** = **Nag_DendWest**, then the end-points of the dendrogram are to the west.

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

n

Input: the number of objects in the cluster analysis.

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

dord[n]

Input: 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 $i = 1, 2, \ldots, \mathbf{n}-1$.

dmin

Input: the clustering distance at which the dendrogram begins.

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

dstep

Input: the distance represented by one symbol of the dendrogram.

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

nsym

Input: the number of character positions used in the dendrogram. Hence the clustering distance at which the dendrogram terminates is given by **dmin** + **nsym** × **dstep**.

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

c

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

If **orient** = **Nag_DendNorth** or **Nag_DendSouth**, then the number of lines in the dendrogram is **nsym**.
If **orient** = **Nag_DendEast** or **Nag_DendWest**, then 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).

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 **orient** had an illegal value.

NE_INT_ARG_LT

On entry, **n** must not be less than 2: **n** = ⟨*value*⟩.
On entry, **nsym** must not be less than 1: **nsym** = ⟨*value*⟩.

NE_REAL_ARG_LT

On entry, **dmin** must not be less than 0.0: **dmin** = ⟨*value*⟩.

NE_REAL_ARG_LE

On entry, **dstep** must not be less than or equal to 0.0: **dstep** = ⟨*value*⟩.

NE_DENDROGRAM_ARRAY

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

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 consult NAG for assistance.

6. 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 **nsym** × **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. The user can print suitable labels using the ordering given by the array **iord** returned by nag_mv_hierar_cluster_analysis (g03ecc).

6.1. Accuracy

Not applicable.

6.2. References

Everitt B S (1974) *Cluster Analysis* Heinemann.
Krzanowski W J (1990) *Principles of Multivariate Analysis* Oxford University Press.

7. See Also

nag_mv_hierar_cluster_analysis (g03ecc)
nag_mv_dend_free (g03xzc)

8. 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**.

8.1. Program Text

```
/* nag_mv_dendrogram (g03ehc) 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 10
#define MMAX 10
#define LDC 100

main()
{

  double cd[NMAX-1], d[NMAX*(NMAX-1)/2], dord[NMAX],
  s[MMAX], x[NMAX][MMAX];
  double dmin_;
  double dstep;

  Integer ilc[NMAX-1], iord[NMAX], isx[MMAX], iuc[NMAX-1];
  Integer nsym;
```

```
  Integer i, j;
  Integer m, n;
  Integer int_method;
  Integer tdx;

  char char_dist[2];
  char char_scale[2];
  char char_update[2];
  char **c=0;

  Nag_ClusterMethod method;
  Nag_MatUpdate update;
  Nag_DistanceType dist;
  Nag_VarScaleType scale;

  tdx = MMAX;
  Vprintf("g03ehc Example Program Results\n\n");

  Vscanf("%*[^\n]");
  Vscanf("%ld",&n);
  Vscanf("%ld",&m);

  if (n <= NMAX && m <= MMAX)
    {
      Vscanf("%ld",&int_method);
      if (int_method == 1)
        method = Nag_SingleLink;
      else  if (int_method == 2)
        method = Nag_CompleteLink;
      else  if (int_method == 3)
        method = Nag_GroupAverage;
      else if (int_method == 4)
        method = Nag_Centroid;
      else  if (int_method == 5)
        method = Nag_Median;
      else
        method = Nag_MinVariance;


      Vscanf("%s",char_update);
      if (*char_update == 'U')
        update = Nag_MatUp;
      else
        update = Nag_NoMatUp;

      Vscanf("%s",char_dist);
      if (*char_dist == 'A')
        dist = Nag_DistAbs;
      else if (*char_dist == 'E')
        dist = Nag_DistEuclid;
      else
        dist = Nag_DistSquared;

      Vscanf("%s",char_scale);
      if (*char_scale == 'S')
        scale = Nag_VarScaleStd;
      else if (*char_scale == 'R')
        scale = Nag_VarScaleRange;
      else if (*char_scale == 'G')
        scale = Nag_VarScaleUser;
      else
        scale = Nag_NoVarScale;

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

```
        for (i = 0; i < m; ++i)
          Vscanf("%lf",&s[i]);
        Vscanf("%lf",&dmin_);
        Vscanf("%lf",&dstep);
        Vscanf("%ld",&nsym);

        /*  Compute the distance matrix */
        g03eac(update, dist, scale, n, m, (double *)x, tdx, isx, s, d, NAGERR_DEFAULT);

        /*  Perform clustering */
        g03ecc(method, n, d, ilc, iuc, cd, iord, dord, NAGERR_DEFAULT);
        /*  Produce dendrograms */

        g03ehc(Nag_DendEast, n, dord, dmin_, dstep, nsym, &c, NAGERR_DEFAULT);

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

        Vscanf("%lf",&dmin_);
        Vscanf("%lf",&dstep);
        Vscanf("%ld",&nsym);
        g03xzc(&c);
        g03ehc(Nag_DendSouth, n, dord, dmin_, dstep, nsym, &c, NAGERR_DEFAULT);
        Vprintf("\n\n Dendrogram, Orientation South\n\n");
        for (i=0; i<nsym; i++)
          {
            Vprintf("%s\n", c[i]);
          }
        g03xzc(&c);
        exit(EXIT_SUCCESS);
      }
      else
      {
        Vprintf("Incorrect input value of n or m.\n");
        exit(EXIT_FAILURE);
      }
}
```

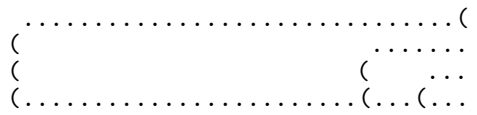
8.2. Program Data

```
g03ehc Example Program Data
5 3
5
I S U
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
```

8.3. Program Results

```
g03ehc Example Program Results

Dendrogram, Orientation East
```



```
 Dendrogram, Orientation South
```