NAG C Library Function Document

nag_order_data (g10zac)

1 Purpose

nag_order_data (g10zac) orders and weights data which is entered unsequentially, weighted or unweighted.

2 Specification

```
#include <nag.h>
#include <nagg10.h>

void nag_order_data (Integer n, const double x[], const double y[],
          const double weights[], Integer *nord, double xord[], double yord[],
          double wtord[], double *rss, NagError *fail)
```

3 Description

Given a set of observations (x_i, y_i) for $i = 1, 2, \ldots, n$, with corresponding weights w_i, nag_order_data rearranges the observations so that the x_i are in ascending order.

For any equal x_i in the ordered set, say $x_j = x_{j+1} = \ldots = x_{j+k}$, a single observation x_j is returned with a corresponding y' and w', calculated as:

$$w' = \sum_{l=0}^{k} w_{i+l}$$

and

$$y' = \frac{\sum_{l=0}^{k} w_{i+l} y_{i+l}}{w'}.$$

Observations with zero weight are ignored. If no weights are supplied by the user, then unit weights are assumed; that is $w_i = 1$ for $i = 1, 2, \ldots, n$.

In addition, the within group sum of squares is computed for the tied observations using West's algorithm (see West (1979)).

4 Parameters

1: **n** – Integer *Input*

On entry: the number of observations, n.

Constraint: **n** ≥ 1.

2: **x[n]** – const double *Input*

On entry: the values x_i, for $i = 1, 2, \ldots, n$.

3: **y[n]** – const double *Input*

On entry: the values y_i, for $i = 1, 2, \ldots, n$.

4: **weights[n]** – const double *Input*

On entry: **weights** must contain the n weights, if they are required. Otherwise, **weights** must be set to the null pointer (double*) 0.

Constraint: if **weights** are required, then **weights**$[i-1] \geq 0.0$, for $i = 1, 2, \ldots, n$, and at least one **wt**$[i-1] > 0.0$, for some i.

5: **nord** – Integer * *Output*

On exit: the number of distinct observations.

6: **xord[n]** – double *Output*

On exit: the first **nord** elements contain the ordered and distinct x_i.

7: **yord[n]** – double *Output*

On exit: the first **nord** elements contain the values y' corresponding to the values in **xord**.

8: **wtord[n]** – double *Output*

On exit: the first **nord** elements contain the values w' corresponding to the values of **xord** and **yord**.

9: **rss** – double * *Output*

On exit: the within group sum of squares for tied observations.

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

The NAG error parameter (see the Essential Introduction).

5 Error Indicators and Warnings

NE_INT_ARG_LT

On entry, **n** must not be less than 1: **n** = <*value*>.

NE_REAL_ARRAY_CONS

On entry, **weights**[<*value*>] = <*value*>.

Constraint: **weights**$[i] \geq 0$, for $i = 0, 1, \ldots, n-1$.

NE_ARRAY_CONS

The contents of array **weights** are not valid.

Constraint: at least one element of **weights** must be > 0.

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

The routine may be used to compute the pure error sum of squares in simple linear regression along with nag_regsn_mult_linear (g02dac), see Draper and Smith (1985).

6.1 Accuracy

For a discussion on the accuracy of the algorithm for computing mean and variance see West (1979).

6.2 References

Draper N R and Smith H (1985) *Applied Regression Analysis* Wiley (2nd Edition)

West D H D (1979) Updating mean and variance estimates: An improved method *Comm. ACM* **22** 532–555

7 See Also

None.

8 Example

A set of unweighted observations are input and nag_order_data used to produce a set of strictly increasing weighted observations.

8.1 Program Text

```
/* nag_order_data (g10zac) Example Program.
 *
 * Copyright 2000 Numerical Algorithms Group.
 *
 * Mark 6, 2000.
 */


#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg10.h>

int main (void)
{
  char weight[2];
  double rss, *weights=0, *wtord=0, *x=0, *xord=0, *y=0, *yord=0, *wtptr;
  Integer i, *iwrk=0, n, nord;
  Integer exit_status=0;
  NagError fail;

  INIT_FAIL(fail);
  Vprintf("g10zac Example Program Results\n");

/*  Skip heading in data file */
  Vscanf("%*[^\n]");

  Vscanf("%ld", &n);
  if (!(x = NAG_ALLOC(n, double))
      || !(y = NAG_ALLOC(n, double))
      || !(weights = NAG_ALLOC(n, double))
      || !(xord = NAG_ALLOC(n, double))
      || !(yord = NAG_ALLOC(n, double))
      || !(wtord = NAG_ALLOC(n, double))
      || !(iwrk = NAG_ALLOC(n, Integer)))
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
```

```
      Vscanf(" %s ", weight);
      for (i = 1; i <= n; ++i)
   Vscanf("%lf %lf", &x[i - 1], &y[i - 1]);
      if (*weight == 'W')
   wtptr = weights;
      else
   wtptr = 0;

      g10zac(n, x, y, wtptr, &nord, xord, yord, wtord, &rss, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from g10zac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

/*  Print results */
      Vprintf("\n");
      Vprintf("%s%6ld\n", "Number of distinct observations = ",
      nord);
      Vprintf("%s%13.5f\n", "Residual sum of squares = ", rss);
      Vprintf("\n");
      Vprintf("             %s\n", " X                 Y              WEIGHTS");
      for (i = 1; i <= nord; ++i)
   Vprintf("     %13.5f     %13.5f     %13.5f\n", xord[i - 1], yord[i - 1],
   wtord[i - 1]);
 END:
      if (x) NAG_FREE(x);
      if (y) NAG_FREE(y);
      if (weights) NAG_FREE(weights);
      if (xord) NAG_FREE(xord);
      if (yord) NAG_FREE(yord);
      if (wtord) NAG_FREE(wtord);
      if (iwrk) NAG_FREE(iwrk);
      return exit_status;
}
```

8.2 Program Data

```
g10zac Example Program Data
10
U
1.0 4.0
3.0 4.0
5.0 1.0
5.0 2.0
3.0 5.0
4.0 3.0
9.0 4.0
6.0 9.0
9.0 7.0
9.0 4.0
```

8.3 Program Results

```
g10zac Example Program Results

Number of distinct observations =      6
Residual sum of squares =       7.00000

             X                 Y                WEIGHTS
          1.00000           4.00000           1.00000
          3.00000           4.50000           2.00000
          4.00000           3.00000           1.00000
          5.00000           1.50000           2.00000
          6.00000           9.00000           1.00000
          9.00000           5.00000           3.00000
```