NAG C Library Function Document

nag_frequency_table (g01aec)

1 Purpose

nag_frequency_table (g01aec) constructs a frequency distribution of a variable, according to either user-supplied, or function-calculated class boundary values.

2 Specification

```
#include <nag.h>
#include <nagg01.h>
```

```
void nag_frequency_table (Integer n, const double x[], Integer num_class,
     Nag_ClassBoundary class, double cint[], Integer ifreq[], double *xmin,
     double *xmax, NagError *fail)
```

3 Description

The data consists of a sample of n observations of a continuous variable, denoted by x_i, for $i = 1, 2, \ldots, n$. Let $a = \min(x_1, \ldots, x_n)$ and $b = \max(x_1, \ldots, x_n)$. The function constructs a frequency distribution with $k(> 1)$ classes denoted by f_i, for $i = 1, 2, \ldots, k$. The boundary values may be either user-supplied, or function-calculated, and are denoted by y_j, for $j = 1, 2, \ldots, k - 1$.

If the boundary values of the classes are to be function-calculated, then they are determined in one of the following ways:

(a) If $k > 2$, the range of x values is divided into $k - 2$ intervals of equal length, and two extreme intervals, defined by the class boundary values $y_1, y_2, \ldots, y_{k-1}$.

(b) If $k = 2$, $y_1 = \frac{1}{2}(a + b)$.

However formed, the values $y_1, \ldots, y_{k-1}$ are assumed to be in ascending order. The class frequencies are formed with

$$f_1 = \text{ the number of } x \text{ values in the interval } (-\infty, y_1)$$
$$f_i = \text{ the number of } x \text{ values in the interval } [y_{i-1}, i_k), \quad i = 2, \ldots, k-1$$
$$f_k = \text{ the number of } x \text{ values in the interval } [y_{k-1}, \infty),$$

where [means inclusive, and) means exclusive. If the class boundary values are function-calculated and $k > 2$, then $f_1 = f_k = 0$, and y_1 and y_{k-1} are chosen so that $y_1 < a$ and $y_{k-1} > b$.

If a frequency distribution is required for a discrete variable, then it is suggested that the user supplies the class boundary values; function-calculated boundary values may be slightly imprecise (due to the adjustment of y_1 and y_{k-1} outlined above) and cause values very close to a class boundary to be assigned to the wrong class.

4 References

None.

5 Arguments

1: **n** – Integer *Input*

On entry: the number of observations, n.

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

2: **x**[**n**] – const double *Input*

On entry: the sample of observations of the variable for which the frequency distribution is required, x_i, for $i = 1, 2, \ldots, n$. The values may be in any order.

3: **num_class** – Integer *Input*

On entry: the number of classes desired in the frequency distribution, k. Whether or not class boundary values are user-supplied, **num_class** must include the two extreme classes which stretch to $\pm\infty$.

Constraint: **num_class** ≥ 2.

4: **class** – Nag_ClassBoundary *Input*

On entry: indicates whether class boundary values are to be calculated within the function, or are supplied by the user.

class = **Nag_ClassBoundaryComp**

The class boundary values are to be calculated within the function.

class = **Nag_ClassBoundaryUser**

They are user-supplied.

Constraint: **class** = **Nag_ClassBoundaryUser** or **Nag_ClassBoundaryComp**.

5: **cint**[**num_class** – **1**] – double *Input/Output*

On entry: if **class** $= 0$, then the elements of **cint** need not be assigned values, as the function calculates $k - 1$ class boundary values.

If **class** $= 1$, the first $k - 1$ elements of **cint** must contain the user-supplied class boundary values, in ascending order.

On exit: the first $k - 1$ elements of **cint** contain the class boundary values in ascending order.

Constraint: if **class** $= 1$, **cint**$[i - 1] <$ **cint**$[i]$, for $i = 1, 2, \ldots, k - 2$.

6: **ifreq**[**num_class**] – Integer *Output*

On exit: the elements of **ifreq** contain the frequencies in each class, f_i, for $i = 1, 2, \ldots, k$. In particular **ifreq**(1) contains the frequency of the class up to **cint**(1), f_1, and **ifreq**$[k - 1]$ contains the frequency of the class greater than **cint**$[k - 2]$, f_k.

7: **xmin** – double * *Output*

On exit: the smallest value in the sample, a.

8: **xmax** – double * *Output*

On exit: the largest value in the sample, b.

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

The NAG error parameter, see the Essential Introduction.

6 Error Indicators and Warnings

NE_BAD_PARAM

On entry, parameter **class** had an illegal value.

NE_INT_ARG_LT

On entry, **num_class** must not be less than 2: **num_class** $= \langle value \rangle$.

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.

NE_NOT_STRICTLY_INCREASING

The sequence **cint** is not strictly increasing: **cint**[$\langle value\rangle$] = $\langle value\rangle$, **cint**[$\langle value\rangle$] = $\langle value\rangle$.

7 Accuracy

The method used is believed to be stable.

8 Further Comments

The time taken by nag_frequency_table (g01aec) increases with **num_class** and **n**. It also depends on the distribution of the sample observations.

9 Example

In the example program, **nprob** determines the number of sets of data to be analysed. For each analysis the sample observations and optionally class boundary values, are read. After calling the function the calculated frequency distribution and largest and smallest observations values are printed. In the example, there is one problem to be analysed, with 70 observations to be grouped into 5 function-calculated classes.

9.1 Program Text

```
/* nag_frequency_table (g01aec) Example Program.
 *
 * Copyright 2000 Numerical Algorithms Group.
 *
 * Mark 6a revised, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg01.h>

int main(void)
{
  Integer exit_status=0, i, iclass, j, *jfreq=0, n, nprob, num_class;
  NagError fail;
  Nag_ClassBoundary iclass_enum;
  double *a=0, *c=0, xmax, xmin;

  INIT_FAIL(fail);
  Vprintf("nag_frequency_table (g01aec) Example Program Results\n\n");

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

  Vscanf("%ld", &nprob);
  for (i = 1; i <= nprob; ++i)
    {
      Vscanf("%ld %ld %ld", &n, &iclass, &num_class);
      if (!(a = NAG_ALLOC(n, double))
          || !(c = NAG_ALLOC(num_class-1, double))
          || !(jfreq = NAG_ALLOC(num_class, Integer)))
        {
          Vprintf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
      for (j = 1; j <= n; ++j)
        Vscanf("%lf", &a[j - 1]);
```

```
      Vprintf("Problem %ld\n", i);
      Vprintf("Number of cases %ld\n", n);
      Vprintf("Number of classes, including extreme classes %ld\n",
              num_class);
      if (iclass != 1)
        Vprintf("Routine-supplied class boundaries\n\n");
      else
        {
          for (j = 1; j <= num_class-1; ++j)
            Vscanf("%lf", &c[j - 1]);
          Vprintf("User-supplied class boundaries\n");
        }
      if (iclass == 1)
        iclass_enum = Nag_ClassBoundaryUser;
      else if (iclass == 0)
        iclass_enum = Nag_ClassBoundaryComp;
      else
        iclass_enum = (Nag_ClassBoundary)-999;
      /* nag_frequency_table (g01aec).
       * Frequency table from raw data
       */
      nag_frequency_table(n, a, num_class, iclass_enum, c, jfreq, &xmin, &xmax,
                          &fail);
      if (fail.code == NE_NOERROR)
        {
          Vprintf("Successful call of g01aec\n\n");
          Vprintf("*** Frequency  distribution ***\n\n");
          Vprintf("       Class            Frequency\n\n");
          Vprintf("   Up to    %8.2f %11ld\n", c[0], jfreq[0]);
          if (num_class-1 > 1)
            {
              for (j = 2; j <= num_class-1; ++j)
                Vprintf("%8.2f to %8.2f %11ld\n", c[j - 2], c[j - 1],
                        jfreq[j - 1]);
            }
          Vprintf("%8.2f     and over  %9ld\n\n", c[num_class - 2],
                  jfreq[num_class-1]);
          Vprintf("Total frequency = %ld\n", n);
          Vprintf("Minimum = %9.2f\n", xmin);
          Vprintf("Maximum = %9.2f\n", xmax);
        }
      else
        {
          Vprintf("Error from nag_frequency_table (g01aec).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }
    }
 END:
  if (a) NAG_FREE(a);
  if (c) NAG_FREE(c);
  if (jfreq) NAG_FREE(jfreq);
  return exit_status;
}
```

9.2 Program Data

```
nag_frequency_table (g01aec) Example Program Data
 1
   70    0        7
22.3   21.6   22.6   22.4   22.4   22.4   22.1   21.9   23.1   23.4
23.4   22.6   22.5   22.5   22.1   22.6   22.3   22.4   21.8   22.3
22.1   23.6   20.8   22.2   23.1   21.1   21.7   21.4   21.6   22.5
21.2   22.6   22.2   22.2   21.4   21.7   23.2   23.1   22.3   22.3
21.1   21.4   21.5   21.8   22.8   21.4   20.7   21.6   23.2   23.6
22.7   21.7   23.0   21.9   22.6   22.1   22.2   23.4   21.5   23.0
22.8   21.4   23.2   21.8   21.2   22.0   22.4   22.8   23.2   23.6
```

9.3 Program Results

```
nag_frequency_table (g01aec) Example Program Results

Problem 1
Number of cases 70
Number of classes, including extreme classes 7
Routine-supplied class boundaries

Successful call of g01aec

*** Frequency  distribution ***

       Class            Frequency

   Up to       20.70          0
   20.70 to    21.28          6
   21.28 to    21.86         16
   21.86 to    22.44         21
   22.44 to    23.02         14
   23.02 to    23.60         13
   23.60     and over         0

Total frequency = 70
Minimum =     20.70
Maximum =     23.60
```