NAG C Library Function Document

nag_sum_sqs (g02buc)

1 Purpose

nag_sum_sqs (g02buc) calculates the sample means and sums of squares and cross-products, or sums of squares and cross-products of deviations from the mean, in a single pass for a set of data. The data may be weighted.

2 Specification

```c
void nag_sum_sqs (Nag_OrderType order, Nag_SumSquare mean, Integer n, Integer m,
                 const double x[], Integer pdx, const double wt[], double *sw,
                 double wmean[], double c[], NagError *fail)
```

3 Description

nag_sum_sqs (g02buc) is an adaptation of West’s WV2 algorithm; see West (1979). This routine calculates the (optionally weighted) sample means and (optionally weighted) sums of squares and cross-products or sums of squares and cross-products of deviations from the (weighted) mean for a sample of \( n \) observations on \( m \) variables \( X_j \), for \( j = 1, 2, \ldots, m \). The algorithm makes a single pass through the data.

For the first \( i - 1 \) observations let the mean of the \( j \)th variable be \( \bar{x}_j(i-1) \), the cross-product about the mean for the \( j \)th and \( k \)th variables be \( c_{jk}(i-1) \) and the sum of weights be \( W_i \). These are updated by the \( i \)th observation, \( x_{ij} \), for \( j = 1, 2, \ldots, m \), with weight \( w_i \) as follows:

\[
W_i = W_{i-1} + w_i \quad \bar{x}_j(i) = \bar{x}_j(i-1) + \frac{w_i}{W_i} (x_j - \bar{x}_j(i-1)), \quad j = 1, 2, \ldots, m
\]

and

\[
c_{jk}(i) = c_{jk}(i-1) + \frac{w_i}{W_i} (x_j - \bar{x}_j(i-1))(x_k - \bar{x}_k(i-1))W_{i-1}, \quad j = 1, 2, \ldots, m; \quad k = j, j+1, \ldots, m.
\]

The algorithm is initialised by taking \( \bar{x}_j(1) = x_{ij} \), the first observation, and \( c_{ij}(1) = 0.0 \).

For the unweighted case \( w_i = 1 \) and \( W_i = i \) for all \( i \).

Note that only the upper triangle of the matrix is calculated and returned packed by column.

4 References


5 Parameters

1:  
order – Nag_OrderType

On entry: the order parameter specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by order = Nag_RowMajor. See Section 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

Constraint: order = Nag_RowMajor or Nag_ColMajor.
mean – Nag_SumSquare

**Input**

*On entry*: indicates whether nag_sum_sqs (g02buc) is to calculate sums of squares and cross-products, or sums of squares and cross-products of deviations from the mean.

If \texttt{mean} = \texttt{Nag_AboutMean}, the sums of squares and cross-products of deviations from the mean are calculated.

If \texttt{mean} = \texttt{Nag_AboutZero}, the sums of squares and cross-products are calculated.

**Constraint**: \texttt{mean} = \texttt{Nag_AboutMean} or \texttt{Nag_AboutZero}.

\begin{itemize}
  \item **n** – Integer
  \item **m** – Integer
  \item **x[\texttt{dim}]** – const double
  \item **pdx** – Integer
  \item **wt[\texttt{dim}]** – const double
  \item **sw** – double *
  \item **wmean[m]** – double
\end{itemize}

**n** – *Input*

*On entry*: the number of observations in the data set, \(n\).

**Constraint**: \(n \geq 1\).

**m** – *Input*

*On entry*: the number of variables, \(m\).

**Constraint**: \(m \geq 1\).

\texttt{x[\texttt{dim}]} must be at least \(\max(1, \texttt{pdx} \times \texttt{m})\) when \texttt{order} = \texttt{Nag_ColMajor} and at least \(\max(1, \texttt{pdx} \times \texttt{n})\) when \texttt{order} = \texttt{Nag_RowMajor}.

\begin{align*}
  &\text{if } \texttt{order} = \texttt{Nag_ColMajor}, \ x[(j - 1) \times \texttt{pdx} + i - 1]; \\
  &\text{if } \texttt{order} = \texttt{Nag_RowMajor}, \ x[(i - 1) \times \texttt{pdx} + j - 1].
\end{align*}

*On entry*: \(X(i,j)\) must contain the \(i\)th observation on the \(j\)th variable, for \(i = 1, 2, \ldots, n\); \(j = 1, 2, \ldots, m\).

**pdx** – *Input*

*On entry*: the stride separating matrix row or column elements (depending on the value of \texttt{order}) in the array \texttt{x}.

**Constraints**:

\begin{align*}
  &\text{if } \texttt{order} = \texttt{Nag_ColMajor}, \ \texttt{pdx} \geq \texttt{n}; \\
  &\text{if } \texttt{order} = \texttt{Nag_RowMajor}, \ \texttt{pdx} \geq \texttt{m}.
\end{align*}

**wt[\texttt{dim}]** must be at least \(\texttt{n}\).

*On entry*: the optional weights of each observation. If weights are not provided then \texttt{wt} must be set to the \texttt{NULL} pointer, i.e., \((\text{double *} )\)\texttt{0}, otherwise \texttt{wt[i]} must contain the weight for the \(i - 1\)th observation.

**Constraint**: if \texttt{wt} is not \texttt{NULL}, \(\texttt{wt[i]} \geq 0.0\) for \(i = 0, 1, \ldots, n - 1\).

**sw** – *Output*

*On exit*: the sum of weights.

If \texttt{wt} is \texttt{NULL}, then \texttt{sw} contains the number of observations, \(n\).

**wmean[m]** – *Output*

*On exit*: the sample means. \(\texttt{wmean}[j - 1]\) contains the mean for the \(j\)th variable.
10: \( c[\text{dim}] \) – double

**Output**

**Note:** the dimension, \( \text{dim} \), of the array \( c \) must be at least \( (m \times m + m)/2 \).

**On exit:** the cross-products.

If \( \text{mean} = \text{Nag\_About\_Mean} \), then \( c \) contains the upper triangular part of the matrix of (weighted) sums of squares and cross-products of deviations about the mean.

If \( \text{mean} = \text{Nag\_About\_Zero} \), then \( c \) contains the upper triangular part of the matrix of (weighted) sums of squares and cross-products.

These are stored packed by columns, i.e., the cross-product between the \( j \)th and \( k \)th variable, \( k \geq j \), is stored in \( c(k \times (k - 1)/2 + j) \).

11: \( \text{fail} \) – NagError *

**Input/Output**

The NAG error parameter (see the Essential Introduction).

### 6 Error Indicators and Warnings

#### NE_INT

On entry, \( n = \langle \text{value} \rangle \).

Constraint: \( n \geq 1 \).

On entry, \( \text{pdx} = \langle \text{value} \rangle \).

Constraint: \( \text{pdx} > 0 \).

On entry, \( m = \langle \text{value} \rangle \).

Constraint: \( m \geq 1 \).

#### NE_INT_2

On entry, \( \text{pdx} = \langle \text{value} \rangle, n = \langle \text{value} \rangle \).

Constraint: \( \text{pdx} \geq n \).

On entry, \( \text{pdx} = \langle \text{value} \rangle, m = \langle \text{value} \rangle \).

Constraint: \( \text{pdx} \geq m \).

#### NE_REAL\_ARRAY\_ELEM\_CONS

On entry, \( \text{wt}[[\langle \text{value} \rangle]] < 0.0 \).

#### NE_ALLOC_FAIL

Memory allocation failed.

#### NE_BAD_PARAM

On entry, parameter \( \langle \text{value} \rangle \) had an illegal value.

#### 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.

### 7 Accuracy

For a detailed discussion of the accuracy of this algorithm see Chan *et al.* (1982) or West (1979).
8 Further Comments

*nag_cov_to_corr* (g02bwc) may be used to calculate the correlation coefficients from the cross-products of deviations about the mean. The cross-products of deviations about the mean may be scaled to give a variance-covariance matrix.

The means and cross-products produced by *nag_sum_sqs* (g02buc) may be updated by adding or removing observations using *nag_sum_sqs_update* (g02btc).

9 Example

A program to calculate the means, the required sums of squares and cross-products matrix, and the variance matrix for a set of 3 observations of 3 variables.

9.1 Program Text

```c
/* nag_sum_sqs (g02buc) Example Program. 
 * 
 * Copyright 2002 Numerical Algorithms Group. 
 * 
 * Mark 7, 2002. 
 */
#include <stdio.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf06.h>
#include <nag02.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    double alpha, sw;
    Integer exit_status, j, k, m, mm, n, pdx;
    NagError fail;
    Nag_SumSquare mean_enum;
    /* Arrays */
    char mean[2], weight[2];
    double *c=0, *v=0, *wmean=0, *wt=0, *x=0;
    double *wtptr=0;
    Nag_OrderType order;
    #ifdef NAG_COLUMN_MAJOR
    #define X(I,J) x[(J-1)*pdx+I-1]
    order = Nag_ColMajor;
    #else
    #define X(I,J) x[(I-1)*pdx+J-1]
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);
    exit_status = 0;
    Vprintf("g02buc Example Program Results\n");
    /* Skip heading in data file */
    Vscanf("%*[^
] ");

    while (scanf("' %1s ' ' %1s '%ld%ld%*[^
]", mean, weight, &m, &n) != EOF)
    {
        /* Allocate memory */
        if ( !(c = NAG_ALLOC((m*m+m)/2, double)) ||
            !(v = NAG_ALLOC((m*m+m)/2, double)) ||
            !(wmean = NAG_ALLOC(m, double)) ||
            !(wt = NAG_ALLOC(n, double)) ||
```
(!(x = NAG_ALLOC(n * m, double)) )
{
    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
#endif NAG_COLUMN_MAJOR
pdx = n;
#else
    pdx = m;
#endif

for (j = 1; j <= n; ++j)
    Vscanf("%lf", &wt[j-1]);
Vscanf("%*[\n ]");
for (j = 1; j <= n; ++j)
{
    for (k = 1; k <= m; ++k)
        Vscanf("%lf", &X(j,k));
    Vscanf("%*[\n ]");
}

if (mean[0] == 'M')
    mean_enum = Nag_AboutMean;
else if (mean[0] == 'Z')
    mean_enum = Nag_AboutZero;
else
{
    Vprintf("Incorrect value for mean\n");
    exit_status = -1;
    goto END;
}

if (weight[0] == 'W')
    wtptr = wt;

/* Calculate sums of squares and cross-products matrix */
g02buc(order, mean_enum, n, m, x, pdx, wtptr, &sw, wmean, c, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from g02buc.\n\s\n", fail.message);
    exit_status = 1;
    goto END;
}
Vprintf("\n");
Vprintf("Means\n");
for (j = 1; j <= m; ++j)
    Vprintf("%14.4f\%s", wmean[j-1], j%6 == 0 || j == m ? "\n": "");
if (wtptr)
{
    Vprintf("\n");
    Vprintf("Weights\n");
    for (j = 1; j <= n; ++j)
        Vprintf("%14.4f\%s", wt[j-1], j%6 == 0 || j == n ? "\n": "");
    Vprintf("\n");
}

/* Print the sums of squares and cross-products matrix */
x04ccc(Nag_ColMajor, Nag_Upper, Nag_NonUnitDiag, m, c,
    "Sums of squares and cross-products", 0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from x04ccc.\n\s\n", fail.message);
    exit_status = 1;
    goto END;
}
if (sw > 1.0)
{
    /* Calculate the variance matrix */
alpha = 1.0 / (sw - 1.0);
mm = m * (m + 1) / 2;
f06fdd(mm, alpha, c, l, v, l);
/* Print the variance matrix */
Vprintf("\n");
x04ccc(Nag_ColMajor, Nag_Upper, Nag_NonUnitDiag, m, v,
   "Variance matrix", 0, &fail);
if (fail.code != NE_NOERROR)
{
   Vprintf("Error from x04ccc.\n%s\n", fail.message);
   exit_status = 1;
   goto END;
}
END:
if (c) NAG_FREE(c);
if (v) NAG_FREE(v);
if (wmean) NAG_FREE(wmean);
if (wt) NAG_FREE(wt);
if (x) NAG_FREE(x);

9.2 Program Data
<Program Data>
'M' 'W'
0.1300 1.3070 0.3700
9.1231 3.7011 4.5230
0.9310 0.0900 0.8870
0.0009 0.0099 0.0999

9.3 Program Results
<Program Results>
Means
1.3299 0.3334 0.9874
Weights
0.1300 1.3070 0.3700
Sums of squares and cross-products
1  8.7569  3.6978  4.0707
2  1.5905  1.6861
3  1.9297
Variance matrix
1  10.8512  4.5822  5.0443
2  1.9709  2.0893
3  2.3912