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
#include <nag.h>
#include <nagg02.h>
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 function 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 jth variable be $\bar{x}_j(i - 1)$, the cross-product about the mean for the jth and kth variables be $c_{jk}(i - 1)$ and the sum of weights be $W_{i-1}$. These are updated by the $i$th observation, $x_{ij}$, for $j = 1, 2, \ldots, m$, with weight $w_i$ as follows:

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

and

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

The algorithm is initialized by taking $\bar{x}_j(1) = x_{1j}$, 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 Arguments

1: **order** – Nag_OrderType

*Input*

*On entry:* the order argument 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 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

Constraint: order = Nag_RowMajor or Nag_ColMajor.

2: 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 about the mean.
   mean = Nag_AboutMean
   The sums of squares and cross-products of deviations about the mean are calculated.
   mean = Nag_AboutZero
   The sums of squares and cross-products are calculated.
   Constraint: mean = Nag_AboutMean or Nag_AboutZero.

3: n – Integer
   Input
   On entry: n, the number of observations in the dataset.
   Constraint: n ≥ 1.

4: m – Integer
   Input
   On entry: m, the number of variables.
   Constraint: m ≥ 1.

5: x[dim] – const double
   Input
   Note: the dimension, dim, of the array x must be at least
   \[ \max(1, \text{pdx} \times m) \] when order = Nag_ColMajor;
   \[ \max(1, n \times \text{pdx}) \] when order = Nag_RowMajor.

   Where \( X(i,j) \) appears in this document, it refers to the array element
   \[ x[(j - 1) \times \text{pdx} + i - 1] \] when order = Nag_ColMajor;
   \[ x[(i - 1) \times \text{pdx} + j - 1] \] when order = Nag_RowMajor.

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

6: pdx – Integer
   Input
   On entry: the stride separating row or column elements (depending on the value of order) in the array x.
   Constraints:
   if order = Nag_ColMajor, pdx ≥ n;
   if order = Nag_RowMajor, pdx ≥ m.

7: wt[dim] – const double
   Input
   Note: the dimension, dim, of the array wt must be at least n.

   On entry: the optional weights of each observation. If weights are not provided then wt must be set to NULL, otherwise \( wt[i - 1] \) must contain the weight for the \( i \)th observation.
   Constraint: if wt is not NULL, \( wt[i - 1] \geq 0.0 \), for \( i = 1, 2, \ldots, n \).

8: sw – double *
   Output
   On exit: the sum of weights.
   If wt is NULL, sw contains the number of observations, n.
9: \[ \text{wmean} \] – double
   \textit{On exit:} the sample means. \text{wmean}[j-1] contains the mean for the \( j \)th variable.

10: \[ c[(m \times m + m)/2] \] – double
   \textit{On exit:} the cross-products.
   If \text{mean} = \text{Nag AboutMean}, \text{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 AboutZero}, \text{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 \text{c}[k \times (k - 1)/2 + j - 1].

11: \text{fail} – \text{NagError} *
   The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

\textbf{NE_ALLOC_FAIL}
   Dynamic memory allocation failed.
   See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE_BAD_PARAM}
   On entry, argument \textit{value} had an illegal value.

\textbf{NE_INT}
   On entry, \( m = \text{value} \).
   Constraint: \( m \geq 1 \).
   On entry, \( n = \text{value} \).
   Constraint: \( n \geq 1 \).
   On entry, \( pdx = \text{value} \).
   Constraint: \( pdx > 0 \).

\textbf{NE_INT_2}
   On entry, \( pdx = \text{value} \) and \( m = \text{value} \).
   Constraint: \( pdx \geq m \).
   On entry, \( pdx = \text{value} \) and \( n = \text{value} \).
   Constraint: \( pdx \geq n \).

\textbf{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.
   An unexpected error has been triggered by this function. Please contact NAG.
   See Section 3.6.6 in the Essential Introduction for further information.

\textbf{NE_NO_LICENCE}
   Your licence key may have expired or may not have been installed correctly.
   See Section 3.6.5 in the Essential Introduction for further information.
On entry, \( wt[\text{value}] < 0.0 \).

### 7 Accuracy

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

### 8 Parallelism and Performance

Not applicable.

### 9 Further Comments

\texttt{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 \texttt{nag\_sum\_sqs (g02buc)} may be updated by adding or removing observations using \texttt{nag\_sum\_sqs\_update (g02btc)}.

Two sets of means and cross-products, as produced by \texttt{nag\_sum\_sqs (g02buc)}, can be combined using \texttt{nag\_sum\_sqs\_combine (g02bzc)}.

### 10 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.

#### 10.1 Program Text

```c
#include <stdio.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf16.h>
#include <nagg02.h>
#include <nagx04.h>

int main(void)
{
    /* Arrays */
    char nag_enum_mean[40], nag_enum_weight[40];
    double *c = 0, *v = 0, *wmean = 0, *wt = 0, *x = 0;
    double *wtptr = 0;
    /* Scalars */
    double alpha, sw;
    Integer exit_status, j, k, m, mm, n, pdx;
    Nag_SumSquare mean;
    Nag_Boolean weight;
    Nag_OrderType order;
    NagError fail;

    #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
```
order = Nag_RowMajor;
#endif

INIT_FAIL(fail);
exit_status = 0;
printf("nag_sum_sqs (g02buc) Example Program Results\n");
/* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n ] ");
#else
    scanf("%*[\n ] ");
#endif
#ifdef _WIN32
while (scanf_s("%39s %39s %"NAG_IFMT"%"NAG_IFMT"%*[\n], nag_enum_mean,
        _countof(nag_enum_mean), nag_enum_weight,
        _countof(nag_enum_weight), &m, &n) != EOF)
{
#else
while (scanf("%39s %39s %"NAG_IFMT"%"NAG_IFMT"%*[\n], nag_enum_mean,
        nag_enum_weight, &m, &n) != EOF)
{
#endif
    /* nag_enum_name_to_value (x04nac).
     * Converts NAG enum member name to value
     */
    mean = (Nag_SumSquare) nag_enum_name_to_value(nag_enum_mean);
    weight = (Nag_Boolean) nag_enum_name_to_value(nag_enum_weight);
    /* 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))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
#ifdef NAG_COLUMN_MAJOR
    pdx = n;
#else
    pdx = m;
#endif
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
        scanf_s("%lf", &wt[j-1]);
#else
        scanf("%lf", &wt[j-1]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n ] ");
#else
    scanf("%*[\n ] ");
#endif
    for (j = 1; j <= n; ++j)
    {
#ifdef _WIN32
        scanf_s("%lf", &X(j, k));
#else
        scanf("%lf", &X(j, k));
#endif
    }
#else
    scanf("%*[\n ]");
#endif

if (weight)
    wtptr = wt;

/* Calculate sums of squares and cross-products matrix */
/* nag_sum_sqs (g02buc). */
/* Computes a weighted sum of squares matrix */
nag_sum_sqs(order, mean, n, m, x, pdx, wtptr, &sw, wmean, c, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_sum_sqs (g02buc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

printf("\n");
printf("Means\n");
for (j = 1; j <= m; ++j)
    printf("%14.4f%s", wmean[j-1], j%6 == 0 || j == m?"\n":" ");
if (wtptr)
{
    printf("\n");
    printf("Weights\n");
    for (j = 1; j <= n; ++j)
        printf("%14.4f%s", wt[j-1], j%6 == 0 || j == n?"\n":" ");
    printf("\n");
}

/* Print the sums of squares and cross products matrix */
/* nag_pack_real_mat_print (x04ccc). */
/* Print real packed triangular matrix (easy-to-use) */
fflush(stdout);
nag_pack_real_mat_print(Nag_ColMajor, Nag_Upper, Nag_NonUnitDiag, m, c,
    "Sums of squares and cross-products", 0, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_pack_real_mat_print (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;
    /* v[] = alpha*c[] using */
    /* nag_daxpby (f16ecc) */
    /* Multiply real vector by scalar, preserving input vector */
    nag_daxpby(mm, alpha, c, 1, 0.0, v, 1, &fail);
    /* Print the variance matrix */
    printf("\n");
    /* nag_pack_real_mat_print (x04ccc), see above. */
    /* nag_pack_real_mat_print(Nag_ColMajor, Nag_Upper, Nag_NonUnitDiag, m,
    v, "Variance matrix", 0, &fail); */
    if (fail.code != NE_NOERROR)
    {
        printf( /* Error from nag_pack_real_mat_print (x04ccc).\n%s\n" */
            fail.message);
        exit_status = 1;
    }
}
10.2 Program Data

nag_sum_sqs (g02buc) Example Program Data
Nag_AboutMean Nag_TRUE 3 3
0.1300 1.3070 0.3700
9.1231 3.7011 4.5230
0.9310 0.0900 0.8870
0.0009 0.0099 0.0999

10.3 Program Results

nag_sum_sqs (g02buc) Example Program Results

Means
1.3299 0.3334 0.9874

Weights
0.1300 1.3070 0.3700

Sums of squares and cross-products

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.7569</td>
<td>3.6978</td>
<td>4.0707</td>
</tr>
<tr>
<td>2</td>
<td>1.5905</td>
<td>1.6861</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.9297</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variance matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.8512</td>
<td>4.5822</td>
<td>5.0443</td>
</tr>
<tr>
<td>2</td>
<td>1.9709</td>
<td>2.0893</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2.3912</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>