NAG Library Function Document

nag_sum_sqs_combine (g02bzc)

1 Purpose

nag_sum_sqs_combine (g02bzc) combines two sets of sample means and sums of squares and cross-products matrices. It is designed to be used in conjunction with nag_sum_sqs (g02buc) to allow large datasets to be summarised.

2 Specification

```
#include <nag.h>
#include <nagg02.h>
```

```
void nag_sum_sqs_combine (Nag_SumSquare mean, Integer m, double *xsw,
     double xmean[], double xc[], double ysw, const double ymean[],
     const double yc[], NagError *fail)
```

3 Description

Let X and Y denote two sets of data, each with m variables and n_x and n_y observations respectively. Let μ_x denote the (optionally weighted) vector of m means for the first dataset and C_x denote either the sums of squares and cross-products of deviations from μ_x

$$C_x = \left(X - e\mu_x^{\mathrm{T}}\right)^{\mathrm{T}} D_x \left(X - e\mu_x^{\mathrm{T}}\right)$$

or the sums of squares and cross-products, in which case

$$C_x = X^{\mathrm{T}} D_x X$$

where e is a vector of n_x ones and D_x is a diagonal matrix of (optional) weights and W_x is defined as the sum of the diagonal elements of D. Similarly, let μ_y, C_y and W_y denote the same quantities for the second dataset.

Given $\mu_x, \mu_y, C_x, C_y, W_x$ and W_y nag_sum_sqs_combine (g02bzc) calculates μ_z, C_z and W_z as if a dataset Z, with m variables and $n_x + n_y$ observations were supplied to nag_sum_sqs (g02buc), with Z constructed as

$$Z = \begin{pmatrix} X \\ Y \end{pmatrix}.$$

nag_sum_sqs_combine (g02bzc) has been designed to combine the results from two calls to nag_sum_sqs (g02buc) allowing large datasets, or cases where all the data is not available at the same time, to be summarised.

4 References

Bennett J, Pebay P, Roe D and Thompson D (2009) Numerically stable, single-pass, parallel statistics algorithms *Proceedings of IEEE International Conference on Cluster Computing*

5 Arguments

1: **mean** – Nag_SumSquare *Input*

On entry: indicates whether the matrices supplied in **xc** and **yc** are sums of squares and cross-products, or sums of squares and cross-products of deviations about the mean.

mean = Nag_AboutMean
Sums of squares and cross-products of deviations about the mean have been supplied.

mean = Nag_AboutZero
Sums of squares and cross-products have been supplied.

Constraint: **mean** = Nag_AboutMean or Nag_AboutZero.

2: **m** – Integer *Input*

On entry: m, the number of variables.

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

3: **xsw** – double * *Input/Output*

On entry: W_x, the sum of weights, from the first set of data, X. If the data is unweighted then this will be the number of observations in the first dataset.

On exit: W_z, the sum of weights, from the combined dataset, Z. If both datasets are unweighted then this will be the number of observations in the combined dataset.

Constraint: $\mathbf{xsw} \geq 0$.

4: **xmean**[**m**] – double *Input/Output*

On entry: μ_x, the sample means for the first set of data, X.

On exit: μ_z, the sample means for the combined data, Z.

5: **xc**[$(\mathbf{m} \times \mathbf{m} + \mathbf{m})/2$] – double *Input/Output*

On entry: C_x, the sums of squares and cross-products matrix for the first set of data, X, as returned by nag_sum_sqs (g02buc).

nag_sum_sqs (g02buc), returns this matrix packed by columns, i.e., the cross-product between the jth and kth variable, $k \geq j$, is stored in $\mathbf{xc}[k \times (k-1)/2 + j - 1]$.

No check is made that C_x is a valid cross-products matrix.

On exit: C_z, the sums of squares and cross-products matrix for the combined dataset, Z.

This matrix is again stored packed by columns.

6: **ysw** – double *Input*

On entry: W_y, the sum of weights, from the second set of data, Y. If the data is unweighted then this will be the number of observations in the second dataset.

Constraint: $\mathbf{ysw} \geq 0$.

7: **ymean**[**m**] – const double *Input*

On entry: μ_y, the sample means for the second set of data, Y.

8: **yc**[$(\mathbf{m} \times \mathbf{m} + \mathbf{m})/2$] – const double *Input*

On entry: C_y, the sums of squares and cross-products matrix for the second set of data, Y, as returned by nag_sum_sqs (g02buc).

nag_sum_sqs (g02buc), returns this matrix packed by columns, i.e., the cross-product between the jth and kth variable, $k \geq j$, is stored in $\mathbf{yc}[k \times (k-1)/2 + j - 1]$.

No check is made that C_y is a valid cross-products matrix.

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

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_INT

On entry, $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{m} \geq 1$.

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 contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_REAL

On entry, $\mathbf{xsw} = \langle value \rangle$.
Constraint: $\mathbf{xsw} \geq 0.0$.

On entry, $\mathbf{ysw} = \langle value \rangle$.
Constraint: $\mathbf{ysw} \geq 0.0$.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_sum_sqs_combine (g02bzc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

None.

10 Example

This example illustrates the use of nag_sum_sqs_combine (g02bzc) by dividing a dataset into three blocks of 4, 5 and 3 observations respectively. Each block of data is summarised using nag_sum_sqs (g02buc) and then the three summaries combined using nag_sum_sqs_combine (g02bzc).

The resulting sums of squares and cross-products matrix is then scaled to obtain the covariance matrix for the whole dataset.

10.1 Program Text

```
/* nag_sum_sqs_combine (g02bzc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg02.h>
#include <nagx04.h>

#define X(I,J) x[(order == Nag_ColMajor) ? (J)*pdx + (I) : (I)*pdx + (J)]

int main(void)
{
  /* Integer scalar and array declarations */
  Integer b, i, j, ierr, lc, pdx, m, n, iwt;
  Integer exit_status = 0;

  /* NAG structures and types */
  NagError fail;
  Nag_SumSquare mean;
  Nag_OrderType order = Nag_ColMajor;

  /* Double scalar and array declarations */
  double alpha, xsw, ysw;
  double *wt = 0, *x = 0, *xc = 0, *xmean = 0, *yc = 0, *ymean = 0;

  /* Character scalar and array declarations */
  char cmean[40];

  /* Initialize the error structure */
  INIT_FAIL(fail);

  printf("nag_sum_sqs_combine (g02bzc) Example Program Results\n\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Read in the problem defining variables */
#ifdef _WIN32
  scanf_s("%39s%" NAG_IFMT "%*[^\n] ", cmean, (unsigned)_countof(cmean),
          &m);
#else
  scanf("%39s%" NAG_IFMT "%*[^\n] ", cmean, &m);
#endif
  mean = (Nag_SumSquare) nag_enum_name_to_value(cmean);

  /* Allocate memory for output arrays */
  lc = (m * m + m) / 2;
```

```
  if (!(xmean = NAG_ALLOC(m, double)) ||
      !(ymean = NAG_ALLOC(m, double)) ||
      !(xc = NAG_ALLOC(lc, double)) || !(yc = NAG_ALLOC(lc, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Loop over each block of data */
  for (b = 0;;) {
    /* Read in the number of observations in this block and a flag indicating
     * whether weights have been supplied (iwt = 1) or not (iwt = 0).
     */
#ifdef _WIN32
    ierr = scanf_s("%" NAG_IFMT "%" NAG_IFMT "", &n, &iwt);
#else
    ierr = scanf("%" NAG_IFMT "%" NAG_IFMT "", &n, &iwt);
#endif

    if (ierr == EOF || ierr < 2)
      break;
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif

    /* Keep a running total of the number of blocks of data */
    b++;

    /* Reallocate X to the required size */
    NAG_FREE(x);
    pdx = n;
    if (!(x = NAG_ALLOC(pdx * m, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

    /* Read in the data for this block */
    if (iwt) {
      /* Weights supplied, so reallocate X to the required size */
      NAG_FREE(wt);
      if (!(wt = NAG_ALLOC(n, double)))
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }
      for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
#ifdef _WIN32
          scanf_s("%lf", &X(i, j));
#else
          scanf("%lf", &X(i, j));
#endif
#ifdef _WIN32
        scanf_s("%lf", &wt[i]);
#else
        scanf("%lf", &wt[i]);
#endif
      }
    }
    else {
      /* No weights */
      NAG_FREE(wt);
      wt = 0;

      for (i = 0; i < n; i++)
```

```
        for (j = 0; j < m; j++)
#ifdef _WIN32
          scanf_s("%lf", &X(i, j));
#else
          scanf("%lf", &X(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif

    /* Call nag_sum_sqs (g02buc) to summarise this block of data */
    if (b == 1) {
      /* This is the first block of data, so summarise the data into
       * xmean and xc.
       */
      nag_sum_sqs(order, mean, n, m, x, pdx, wt, &xsw, xmean, xc, &fail);
      if (fail.code != NE_NOERROR) {
        printf("Error from nag_sum_sqs (g02buc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
      }
    }
    else {
      /* This is not the first block of data, so summarise the data into
       * ymean and yc.
       */
      nag_sum_sqs(order, mean, n, m, x, pdx, wt, &ysw, ymean, yc, &fail);
      if (fail.code != NE_NOERROR) {
        printf("Error from nag_sum_sqs (g02buc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
      }

      /* Call nag_sum_sqs_combine (g02bzc) to update the running summaries */
      nag_sum_sqs_combine(mean, m, &xsw, xmean, xc, ysw, ymean, yc, &fail);
      if (fail.code != NE_NOERROR) {
        printf("Error from nag_sum_sqs_combine (g02bzc).\n%s\n",
               fail.message);
        exit_status = 1;
        goto END;
      }
    }
  }

  /* Display results */
  printf(" Means\n ");
  for (i = 0; i < m; i++)
    printf("%14.4f", xmean[i]);
  printf("\n\n");
  fflush(stdout);

  /* Call nag_pack_real_mat_print (x04ccc) to print the sums of squares */
  nag_pack_real_mat_print(Nag_ColMajor, Nag_Upper, Nag_NonUnitDiag, m, xc,
                          "Sums of squares and cross-products", NULL, &fail);

  if (xsw > 1.0 && mean == Nag_AboutMean && fail.code == NE_NOERROR) {
    /* Convert the sums of squares and cross-products to a
       covariance matrix */
    alpha = 1.0 / (xsw - 1.0);
    for (i = 0; i < lc; i++)
      xc[i] *= alpha;

    printf("\n");
    fflush(stdout);
    nag_pack_real_mat_print(Nag_ColMajor, Nag_Upper, Nag_NonUnitDiag, m, xc,
                            "Covariance matrix", NULL, &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;
  }

END:
  NAG_FREE(x);
  NAG_FREE(wt);
  NAG_FREE(xc);
  NAG_FREE(xmean);
  NAG_FREE(yc);
  NAG_FREE(ymean);

  return (exit_status);
}
```

10.2 Program Data

```
nag_sum_sqs_combine (g02bzc) Example Program Data
Nag_AboutMean  5                         :: mean,m
4 0                                      :: n,iwt (1st block)
-1.10  4.06  -0.95  8.53 10.41
 1.63 -3.22  -1.15 -1.30  3.78
-2.23 -8.19  -3.50  4.31 -1.11
 0.92  0.33  -1.60  5.80 -1.15           :: End of X for 1st block
5 1                                      :: n,iwt (2nd block)
 2.12  5.00 -11.69 -1.22  2.86 2.00
 4.82 -7.23  -4.67  0.83  3.46 0.89
-0.51 -1.12  -1.76  1.45  0.26 0.32
-4.32  4.89   1.34 -1.12 -2.49 4.19
 0.02 -0.74   0.94 -0.99 -2.61 4.33      :: End of X,WT for 2nd block
3 0                                      :: n,iwt (3rd block)
 1.37  0.00  -0.53 -7.98  3.32
 4.15 -2.81  -4.09 -7.96 -2.13
13.09 -1.43   5.16 -1.83  1.58           :: End of X for 3rd block
```

10.3 Program Results

```
nag_sum_sqs_combine (g02bzc) Example Program Results

 Means
        0.4369        0.4929       -1.3387       -0.5684        0.0987

 Sums of squares and cross-products
          1          2          3          4          5
 1    304.5052  -123.7700   -27.1830   -60.7092    83.4830
 2               298.9148   -17.3196    -2.1710     5.2072
 3                          332.1639    -3.9445   -96.9299
 4                                     264.7684    79.6211
 5                                                225.5948

 Covariance matrix
          1          2          3          4          5
 1     17.1746    -6.9808    -1.5332    -3.4241     4.7086
 2                16.8593    -0.9769    -0.1224     0.2937
 3                           18.7346    -0.2225    -5.4670
 4                                      14.9334     4.4908
 5                                                 12.7239
```