NAG Library Function Document

nag_moments_ratio_quad_forms (g01nbc)

1 Purpose

nag_moments_ratio_quad_forms (g01nbc) computes the moments of ratios of quadratic forms in Normal variables and related statistics.

2 Specification

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

```
void nag_moments_ratio_quad_forms (Nag_OrderType order,
     Nag_MomentType ratio_type, Nag_IncludeMean mean, Integer n,
     const double a[], Integer pda, const double b[], Integer pdb,
     const double c[], Integer pdc, const double ela[], const double emu[],
     const double sigma[], Integer pdsig, Integer l1, Integer l2,
     Integer *lmax, double rmom[], double *abserr, double eps,
     NagError *fail)
```

3 Description

Let x have an n-dimensional multivariate Normal distribution with mean μ and variance-covariance matrix Σ. Then for a symmetric matrix A and symmetric positive semidefinite matrix B, nag_moments_ratio_quad_forms (g01nbc) computes a subset, l_1 to l_2, of the first 12 moments of the ratio of quadratic forms

$$R = x^{\mathrm{T}}Ax/x^{\mathrm{T}}Bx.$$

The sth moment (about the origin) is defined as

$$E(R^s), \tag{1}$$

where E denotes the expectation. Alternatively, this function will compute the following expectations:

$$E\left(R^s\left(a^{\mathrm{T}}x\right)\right) \tag{2}$$

and

$$E\left(R^s\left(x^{\mathrm{T}}Cx\right)\right), \tag{3}$$

where a is a vector of length n and C is a n by n symmetric matrix, if they exist. In the case of (2) the moments are zero if $\mu = 0$.

The conditions of theorems 1, 2 and 3 of Magnus (1986) and Magnus (1990) are used to check for the existence of the moments. If all the requested moments do not exist, the computations are carried out for those moments that are requested up to the maximum that exist, l_{MAX}.

This function is based on the function QRMOM written by Magnus and Pesaran (1993a) and based on the theory given by Magnus (1986) and Magnus (1990). The computation of the moments requires first the computation of the eigenvectors of the matrix $L^{\mathrm{T}}BL$, where $LL^{\mathrm{T}} = \Sigma$. The matrix $L^{\mathrm{T}}BL$ must be positive semidefinite and not null. Given the eigenvectors of this matrix, a function which has to be integrated over the range zero to infinity can be computed. This integration is performed using nag_1d_quad_inf_1 (d01smc).

4 References

Magnus J R (1986) The exact moments of a ratio of quadratic forms in Normal variables *Ann. Économ. Statist.* **4** 95–109

Magnus J R (1990) On certain moments relating to quadratic forms in Normal variables: Further results *Sankhyā, Ser. B* **52** 1–13

Magnus J R and Pesaran B (1993a) The evaluation of cumulants and moments of quadratic forms in Normal variables (CUM): Technical description *Comput. Statist.* **8** 39–45

Magnus J R and Pesaran B (1993b) The evaluation of moments of quadratic forms and ratios of quadratic forms in Normal variables: Background, motivation and examples *Comput. Statist.* **8** 47–55

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: **ratio_type** – Nag_MomentType *Input*

On entry: indicates the moments of which function are to be computed.

ratio_type = Nag_RatioMoments (Ratio)
$E(R^s)$ is computed.

ratio_type = Nag_LinearRatio (Linear with ratio)
$E(R^s(a^{\mathrm{T}}x))$ is computed.

ratio_type = Nag_QuadRatio (Quadratic with ratio)
$E(R^s(x^{\mathrm{T}}Cx))$ is computed.

Constraint: **ratio_type** = Nag_RatioMoments, Nag_LinearRatio or Nag_QuadRatio.

3: **mean** – Nag_IncludeMean *Input*

On entry: indicates if the mean, μ, is zero.

mean = Nag_MeanZero
μ is zero.

mean = Nag_MeanInclude
The value of μ is supplied in **emu**.

Constraint: **mean** = Nag_MeanZero or Nag_MeanInclude.

4: **n** – Integer *Input*

On entry: n, the dimension of the quadratic form.

Constraint: **n** > 1.

5: **a**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **a** must be at least **pda** $\times$ **n**.

The (i,j)th element of the matrix A is stored in

a$[(j-1)\times$ **pda** $+ i - 1]$ when **order** = Nag_ColMajor;
a$[(i-1)\times$ **pda** $+ j - 1]$ when **order** = Nag_RowMajor.

On entry: the n by n symmetric matrix A. Only the lower triangle is referenced.

6: **pda** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **a**.

Constraint: **pda** $\geq$ **n**.

7: **b**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **b** must be at least **pdb** $\times$ **n**.

The (i, j)th element of the matrix B is stored in

b$[(j-1) \times$ **pdb** $+ i - 1]$ when **order** $=$ Nag_ColMajor;
b$[(i-1) \times$ **pdb** $+ j - 1]$ when **order** $=$ Nag_RowMajor.

On entry: the n by n positive semidefinite symmetric matrix B. Only the lower triangle is referenced.

Constraint: the matrix B must be positive semidefinite.

8: **pdb** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **b**.

Constraint: **pdb** $\geq$ **n**.

9: **c**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **c** must be at least

max(1, **pdc** $\times$ **n**) when **ratio_type** $=$ Nag_QuadRatio;
1 otherwise.

The (i, j)th element of the matrix C is stored in

c$[(j-1) \times$ **pdc** $+ i - 1]$ when **order** $=$ Nag_ColMajor;
c$[(i-1) \times$ **pdc** $+ j - 1]$ when **order** $=$ Nag_RowMajor.

On entry: if **ratio_type** $=$ Nag_QuadRatio, **c** must contain the n by n symmetric matrix C; only the lower triangle is referenced.

If **ratio_type** $\neq$ Nag_QuadRatio, **c** is not referenced.

10: **pdc** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **c**.

Constraints:

if **ratio_type** $=$ Nag_QuadRatio, **pdc** $\geq$ **n**;
otherwise **pdc** ≥ 1.

11: **ela**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **ela** must be at least

n when **ratio_type** $=$ Nag_LinearRatio;
1 otherwise.

On entry: if **ratio_type** $=$ Nag_LinearRatio, **ela** must contain the vector a of length n, otherwise **ela** is not referenced.

12: **emu**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **emu** must be at least

n when **mean** $=$ Nag_MeanInclude;
1 otherwise.

On entry: if **mean** $=$ Nag_MeanInclude, **emu** must contain the n elements of the vector μ.

If **mean** $=$ Nag_MeanZero, **emu** is not referenced.

13: **sigma**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **sigma** must be at least **pdsig** $\times$ **n**.

The (i, j)th element of the matrix is stored in

sigma$[(j-1) \times$ **pdsig** $+ i - 1]$ when **order** $=$ Nag_ColMajor;
sigma$[(i-1) \times$ **pdsig** $+ j - 1]$ when **order** $=$ Nag_RowMajor.

On entry: the n by n variance-covariance matrix Σ. Only the lower triangle is referenced.

Constraint: the matrix Σ must be positive definite.

14: **pdsig** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **sigma**.

Constraint: **pdsig** $\geq$ **n**.

15: **l1** – Integer *Input*

On entry: the first moment to be computed, l_1.

Constraint: $0 <$ **l1** $\leq$ **l2**.

16: **l2** – Integer *Input*

On entry: the last moment to be computed, l_2.

Constraint: **l1** $\leq$ **l2** ≤ 12.

17: **lmax** – Integer * *Output*

On exit: the highest moment computed, l_{MAX}. This will be l_2 on successful exit.

18: **rmom**[**l2** $-$ **l1** $+$ **1**] – double *Output*

On exit: the l_1 to l_{MAX} moments.

19: **abserr** – double * *Output*

On exit: the estimated maximum absolute error in any computed moment.

20: **eps** – double *Input*

On entry: the relative accuracy required for the moments, this value is also used in the checks for the existence of the moments.

If **eps** $= 0.0$, a value of $\sqrt{\epsilon}$ where ϵ is the ***machine precision*** used.

Constraint: **eps** $= 0.0$ or **eps** $\geq$ ***machine precision***.

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ACCURACY

Full accuracy not achieved in integration.

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

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

NE_EIGENVALUES

Failure in computing eigenvalues.

NE_ENUM_INT

On entry, **ratio_type** $= \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} > 0$.

NE_ENUM_INT_2

On entry, **ratio_type** $= \langle value \rangle$, $\mathbf{pdc} = \langle value \rangle$, $\mathbf{n} = \langle value \rangle$.
Constraint: if **ratio_type** $=$ Nag_QuadRatio, $\mathbf{pdc} \geq \mathbf{n}$;
otherwise $\mathbf{pdc} \geq 1$.

NE_INT

On entry, $\mathbf{l1} = \langle value \rangle$.
Constraint: $\mathbf{l1} \geq 1$.

On entry, $\mathbf{l2} = \langle value \rangle$.
Constraint: $\mathbf{l2} \leq 12$.

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

On entry, $\mathbf{pda} = \langle value \rangle$.
Constraint: $\mathbf{pda} > 0$.

On entry, $\mathbf{pdb} = \langle value \rangle$.
Constraint: $\mathbf{pdb} > 0$.

On entry, $\mathbf{pdc} = \langle value \rangle$.
Constraint: $\mathbf{pdc} > 0$.

On entry, $\mathbf{pdsig} = \langle value \rangle$.
Constraint: $\mathbf{pdsig} > 0$.

NE_INT_2

On entry, $\mathbf{l1} = \langle value \rangle$ and $\mathbf{l2} = \langle value \rangle$.
Constraint: $0 < \mathbf{l1} \leq \mathbf{l2}$.

On entry, $\mathbf{l1} = \langle value \rangle$ and $\mathbf{l2} = \langle value \rangle$.
Constraint: $\mathbf{l1} \leq \mathbf{l2} \leq 12$.

On entry, $\mathbf{l1} = \langle value \rangle$ and $\mathbf{l2} = \langle value \rangle$.
Constraint: $\mathbf{l2} \geq \mathbf{l1}$.

On entry, $\mathbf{pda} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{pda} \geq \mathbf{n}$.

On entry, **pdb** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pdb** $\geq$ **n**.

On entry, **pdc** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pdc** $\geq$ **n**.

On entry, **pdsig** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pdsig** $\geq$ **n**.

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.

NE_MOMENTS

Only $\langle value \rangle$ moments exist, less than **l1** $= \langle value \rangle$.

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.

NE_POS_DEF

On entry, **sigma** is not positive definite.

NE_POS_SEMI_DEF

On entry, **b** is not positive semidefinite or is null.

The matrix $L^{\mathrm{T}}BL$ is not positive semidefinite or is null.

NE_REAL

On entry, **eps** $= \langle value \rangle$.
Constraint: if **eps** $\neq 0.0$, **eps** $\geq$ ***machine precision***.

NE_SOME_MOMENTS

Only $\langle value \rangle$ moments exist, less than **l2** $= \langle value \rangle$.

7 Accuracy

The relative accuracy is specified by **eps** and an estimate of the maximum absolute error for all computed moments is returned in **abserr**.

8 Parallelism and Performance

nag_moments_ratio_quad_forms (g01nbc) is not threaded by NAG in any implementation.

nag_moments_ratio_quad_forms (g01nbc) 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 is given by Magnus and Pesaran (1993b) and considers the simple autoregression:

$$y_t = \beta y_{t-1} + u_t, \quad t = 1, 2, \ldots, n,$$

where $\{u_t\}$ is a sequence of independent Normal variables with mean zero and variance one, and y_0 is known. The least squares estimate of β, $\hat{\beta}$, is given by

$$\hat{\beta} = \frac{\sum_{t=2}^{n} y_t y_{t-1}}{\sum_{t=2}^{n} y_t^2}.$$

Thus $\hat{\beta}$ can be written as a ratio of quadratic forms and its moments computed using nag_moments_ratio_quad_forms (g01nbc). The matrix A is given by

$$A(i+1, i) = \tfrac{1}{2}, \quad i = 1, 2, \ldots n-1;$$

$$A(i, j) = 0, \qquad \text{otherwise},$$

and the matrix B is given by

$$B(i, i) = 1, \quad i = 1, 2, \ldots n-1;$$

$$B(i, j) = 0, \qquad \text{otherwise}.$$

The value of Σ can be computed using the relationships

$$\text{var}\,(y_t) = \beta^2\,\text{var}\,(y_{t-1}) + 1$$

and

$$\text{cov}(y_t y_{t+k}) = \beta\,\text{cov}(y_t y_{t+k-1})$$

for $k \geq 0$ and $\text{var}(y_1) = 1$.

The values of β, y_0, n, and the number of moments required are read in and the moments computed and printed.

10.1 Program Text

```
/* nag_moments_ratio_quad_forms (g01nbc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

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

int main(void)
{

  /* Scalars */
  double        abserr, beta, y0, eps;
  Integer       exit_status, i, j, l1, l2, lmax, n, pda, pdb, pdsigma;
  NagError      fail;
  Nag_OrderType order;
```

```
  /* Arrays */
  double        *a = 0, *b = 0, *c = 0, *ela = 0, *emu = 0, *rmom = 0;
  double        *sigma = 0;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J)     a[(J-1)*pda + I - 1]
#define B(I, J)     b[(J-1)*pdb + I - 1]
#define SIGMA(I, J) sigma[(J-1)*pdsigma + I - 1]
  order = Nag_ColMajor;
#else
#define A(I, J)     a[(I-1)*pda + J - 1]
#define B(I, J)     b[(I-1)*pdb + J - 1]
#define SIGMA(I, J) sigma[(I-1)*pdsigma + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  exit_status = 0;
  printf(
          "nag_moments_ratio_quad_forms (g01nbc) Example Program Results\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
  scanf_s("%lf%lf%*[^\n] ", &beta, &y0);
#else
  scanf("%lf%lf%*[^\n] ", &beta, &y0);
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &n, &l1, &l2);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &n, &l1, &l2);
#endif


  /* Allocate memory */
  if (!(a = NAG_ALLOC(n * n, double)) ||
      !(b = NAG_ALLOC(n * n, double)) ||
      !(c = NAG_ALLOC(n * n, double)) ||
      !(ela = NAG_ALLOC(n, double)) ||
      !(emu = NAG_ALLOC(n, double)) ||
      !(rmom = NAG_ALLOC(l2-l1+1, double)) ||
      !(sigma = NAG_ALLOC(n * n, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  pda = n;
  pdb = n;
  pdsigma = n;

  /* Compute A, EMU, and SIGMA for simple autoregression */

  for (i = 1; i <= n; ++i)
    {
      for (j = i; j <= n; ++j)
        {
          A(j, i) = 0.0;
          B(j, i) = 0.0;
        }
    }
  for (i = 1; i <= n - 1; ++i)
    {
      A(i + 1, i) = 0.5;
```

```
      B(i, i) = 1.0;
    }
  emu[0] = y0 * beta;
  for (i = 1; i <= n - 1; ++i)
    emu[i] = beta * emu[i - 1];
  SIGMA(1, 1) = 1.0;
  for (i = 2; i <= n; ++i)
    SIGMA(i, i) = beta * beta * SIGMA(i - 1, i - 1) + 1.0;
  for (i = 1; i <= n; ++i)
    {
      for (j = i + 1; j <= n; ++j)
        SIGMA(j, i) = beta * SIGMA(j - 1, i);
    }

  eps = 0.0;
  /* nag_moments_ratio_quad_forms (g01nbc).
   * Moments of ratios of quadratic forms in Normal variables,
   * and related statistics
   */
  nag_moments_ratio_quad_forms(order, Nag_RatioMoments, Nag_MeanInclude, n,
                               a, n, b, n, c, n, ela, emu, sigma, n, l1, l2,
                               &lmax, rmom, &abserr, eps, &fail);

  if (fail.code == NE_NOERROR || fail.code == NE_SOME_MOMENTS
      || fail.code == NE_ACCURACY)
    {
      printf("\n");
      printf(" n = %3"NAG_IFMT" beta = %6.3f y0 = %6.3f\n", n, beta, y0);
      printf("\n");
      printf("      Moments\n");
      printf("\n");

      j = 0;
      for (i = l1; i <= lmax; ++i)
        {
          ++j;
          printf("%3"NAG_IFMT"%12.3e\n", i, rmom[j - 1]);
        }
    }
  else
    {
      printf("Error from nag_moments_ratio_quad_forms (g01nbc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(c);
  NAG_FREE(ela);
  NAG_FREE(emu);
  NAG_FREE(rmom);
  NAG_FREE(sigma);

  return exit_status;
}
```

10.2 Program Data

```
nag_moments_ratio_quad_forms (g01nbc) Example Program Data

0.8 1.0     :  Beta Y0

10  1   3   : N  L1  L1
```

10.3 Program Results

```
nag_moments_ratio_quad_forms (g01nbc) Example Program Results

 n =  10 beta =  0.800 y0 =  1.000

      Moments

  1   6.820e-01
  2   5.357e-01
  3   4.427e-01
```