# NAG Library Function Document

# nag_moments_quad_form (g01nac)

## 1 Purpose

nag_moments_quad_form (g01nac) computes the cumulants and moments of quadratic forms in Normal variates.

## 2 Specification

```
#include <nag.h>
#include <nagg01.h>
void nag_moments_quad_form (Nag_OrderType order, Nag_SelectMoments mom,
     Nag_IncludeMean mean, Integer n, const double a[], Integer pda,
     const double emu[], const double sigma[], Integer pdsig, Integer l,
     double rkum[], double rmom[], NagError *fail)
```

## 3 Description

Let $x$ have an $n$-dimensional multivariate Normal distribution with mean $\mu$ and variance-covariance matrix $\Sigma$. Then for a symmetric matrix $A$, nag_moments_quad_form (g01nac) computes up to the first 12 moments and cumulants of the quadratic form $Q = x^{\mathrm{T}}Ax$. The $s$th moment (about the origin) is defined as

$$E(Q^s),$$

where $E$ denotes expectation. The $s$th moment of $Q$ can also be found as the coefficient of $t^s/s!$ in the expansion of $E\left(e^{Qt}\right)$. The $s$th cumulant is defined as the coefficient of $t^s/s!$ in the expansion of $\log\left(E\left(e^{Qt}\right)\right)$.

The function is based on the function CUM written by Magnus and Pesaran (1993a) and based on the theory given by Magnus (1978), Magnus (1979) and Magnus (1986).

## 4 References

Magnus J R (1978) The moments of products of quadratic forms in Normal variables *Statist. Neerlandica* **32** 201–210

Magnus J R (1979) The expectation of products of quadratic forms in Normal variables: the practice *Statist. Neerlandica* **33** 131–136

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 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: **mom** – Nag_SelectMoments *Input*

*On entry*: indicates if moments are computed in addition to cumulants.

**mom** = Nag_CumulantsOnly
Only cumulants are computed.

**mom** = Nag_ComputeMoments
Moments are computed in addition to cumulants.

*Constraint*: **mom** = Nag_CumulantsOnly or Nag_ComputeMoments.

3: **mean** – Nag_IncludeMean *Input*

*On entry*: indicates if the mean, $\mu$, is zero.

**mean** = Nag_MeanZero
$\mu$ is zero.

**mean** = Nag_MeanInclude
The value of $\mu$ 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*: $\mathbf{n} > 1$.

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

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

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

**a**$[(j-1) \times \mathbf{pda} + i - 1]$ when **order** = Nag_ColMajor;
**a**$[(i-1) \times \mathbf{pda} + j - 1]$ 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*: $\mathbf{pda} \geq \mathbf{n}$.

7: **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 $\mu$.

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

8: **sigma**$[dim]$ – const double *Input*

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

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 $\Sigma$. Only the lower triangle is referenced.

*Constraint*: the matrix $\Sigma$ must be positive definite.

9: **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**.

10: **l** – Integer *Input*

*On entry*: the required number of cumulants, and moments if specified.

*Constraint*: $1 \leq$ **l** $\leq 12$.

11: **rkum**[**l**] – double *Output*

*On exit*: the **l** cumulants of the quadratic form.

12: **rmom**[$dim$] – double *Output*

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

**l** when **mom** = Nag_ComputeMoments;
1 otherwise.

*On exit*: if **mom** = Nag_ComputeMoments, the **l** moments of the quadratic form.

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

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

## 6 Error Indicators and Warnings

**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 ⟨*value*⟩ had an illegal value.

**NE_INT**

On entry, **l** = ⟨*value*⟩.
Constraint: **l** $\leq 12$.

On entry, **l** = ⟨*value*⟩.
Constraint: **l** $\geq 1$.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** $> 1$.

On entry, **pda** = ⟨*value*⟩.
Constraint: **pda** $> 0$.

On entry, **pdsig** = ⟨*value*⟩.
Constraint: **pdsig** $> 0$.

**NE_INT_2**

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

## 7 Accuracy

In a range of tests the accuracy was found to be a modest multiple of ***machine precision***. See Magnus and Pesaran (1993b).

## 8 Parallelism and Performance

nag_moments_quad_form (g01nac) is not threaded by NAG in any implementation.

nag_moments_quad_form (g01nac) 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 moments of the quadratic form

$$Q = \sum_{t=2}^{n} y_t y_{t-1}$$

are computed using nag_moments_quad_form (g01nac). The matrix $A$ is given by:

$$\begin{aligned} A(i+1,i) &= \tfrac{1}{2}, \quad i = 1, 2, \ldots n-1; \\ A(i,j) &= 0, \quad \text{otherwise.} \end{aligned}$$

The value of $\Sigma$ 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 $\beta$, $y_0$, $n$, and the number of moments required are read in and the moments and cumulants printed.

## 10.1 Program Text

```
/* nag_moments_quad_form (g01nac) 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        beta, con;
  Integer       exit_status, i, j, l, n, pda, pdsigma;
  NagError      fail;
  Nag_OrderType order;

  /* Arrays */
  double        *a = 0, *emu = 0, *rkum = 0, *rmom = 0, *sigma = 0;

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

  INIT_FAIL(fail);

  exit_status = 0;
  printf("nag_moments_quad_form (g01nac) 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, &con);
#else
  scanf("%lf%lf%*[^\n] ", &beta, &con);
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &n, &l);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ", &n, &l);
#endif
```

```
  /* Allocate memory */
  if (!(a = NAG_ALLOC(n * n, double)) ||
      !(emu = NAG_ALLOC(n, double)) ||
      !(rkum = NAG_ALLOC(l, double)) ||
      !(rmom = NAG_ALLOC(l, double)) ||
      !(sigma = NAG_ALLOC(n * n, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  pda = n;
  pdsigma = n;

  if (l <= 12)
    {
      /* 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;
        }
      for (i = 1; i <= n - 1; ++i)
        A(i + 1, i) = 0.5;
      emu[0] = con * 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);
        }

      /* nag_moments_quad_form (g01nac).
       * Cumulants and moments of quadratic forms in Normal
       * variables
       */
      nag_moments_quad_form(order, Nag_ComputeMoments, Nag_MeanInclude,
                            n, a, n, emu, sigma, n, l, rkum, rmom, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_moments_quad_form (g01nac).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }

      printf("\n");
      printf(" n = %3"NAG_IFMT" beta = %6.3f con = %6.3f\n", n, beta,
              con);
      printf("\n");

      printf("    Cumulants       Moments\n");
      printf("\n");
      for (i = 1; i <= l; ++i)
        printf("%3"NAG_IFMT"%13.4e    %13.4e\n", i, rkum[i - 1],
                rmom[i - 1]);
    }

 END:
  NAG_FREE(a);
  NAG_FREE(emu);
  NAG_FREE(rkum);
```

```
  NAG_FREE(rmom);
  NAG_FREE(sigma);

  return exit_status;
}
```

## 10.2 Program Data

```
nag_moments_quad_form (g01nac) Example Program Data
0.8  1.0   : BETA, CON
10   4     : N, L
```

## 10.3 Program Results

```
nag_moments_quad_form (g01nac) Example Program Results

 n =  10 beta =  0.800 con =  1.000

    Cumulants       Moments

  1  1.7517e+01      1.7517e+01
  2  3.5010e+02      6.5695e+02
  3  1.6091e+04      3.9865e+04
  4  1.1700e+06      3.4039e+06
```