NAG C 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

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 $Q = x^T A x$. 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 $E(e^{Qt})$. The $s$th cumulant is defined as the coefficient of $t^s/s!$ in the expansion of $\log(E(e^{Qt}))$.

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

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.
2: \( \text{mom} \) – Nag_SelectMoments

\textit{Input}

\textit{On entry:} indicates if moments are computed in addition to cumulants.

If \( \text{mom} = \text{Nag\_CumulantsOnly} \), only cumulants are computed.

If \( \text{mom} = \text{Nag\_ComputeMoments} \), moments are computed in addition to cumulants.

\textit{Constraint:} \( \text{mom} = \text{Nag\_CumulantsOnly} \) or \( \text{Nag\_ComputeMoments} \).

3: \( \text{mean} \) – Nag_IncludeMean

\textit{Input}

\textit{On entry:} indicates if the mean, \( \mu \), is zero.

If \( \text{mean} = \text{Nag\_MeanZero} \), \( \mu \) is zero.

If \( \text{mean} = \text{Nag\_MeanInclude} \), the value of \( \mu \) is supplied in \( \text{emu} \).

\textit{Constraint:} \( \text{mean} = \text{Nag\_MeanZero} \) or \( \text{Nag\_MeanInclude} \).

4: \( n \) – Integer

\textit{Input}

\textit{On entry:} the dimension of the quadratic form, \( n \).

\textit{Constraint:} \( n > 1 \).

5: \( \text{a}[\text{dim}] \) – const double

\textit{Input}

\textit{Note:} the dimension, \( \text{dim} \), of the array \( \text{a} \) must be at least \( \text{pda} \times n \).

If \( \text{order} = \text{Nag\_ColMajor} \), the \((i, j)\)th element of the matrix \( A \) is stored in \( \text{a}[(j-1) \times \text{pda} + i - 1] \) and if \( \text{order} = \text{Nag\_RowMajor} \), the \((i, j)\)th element of the matrix \( A \) is stored in \( \text{a}[(i-1) \times \text{pda} + j - 1] \).

\textit{On entry:} the \( n \) by \( n \) symmetric matrix \( A \). Only the lower triangle is referenced.

6: \( \text{pda} \) – Integer

\textit{Input}

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

\textit{Constraint:} \( \text{pda} \geq n \).

7: \( \text{emu}[\text{dim}] \) – const double

\textit{Input}

\textit{Note:} the dimension, \( \text{dim} \), of the array \( \text{emu} \) must be at least \( n \) when \( \text{mean} = \text{Nag\_MeanInclude} \) and at least 1 otherwise.

\textit{On entry:} if \( \text{mean} = \text{Nag\_MeanInclude} \), \( \text{emu} \) must contain the \( n \) elements of the vector \( \mu \). If \( \text{mean} = \text{Nag\_MeanZero} \), \( \text{emu} \) is not referenced.

8: \( \text{sigma}[\text{dim}] \) – const double

\textit{Input}

\textit{Note:} the dimension, \( \text{dim} \), of the array \( \text{sigma} \) must be at least \( \text{pdsig} \times n \).

If \( \text{order} = \text{Nag\_ColMajor} \), the \((i, j)\)th element of the matrix is stored in \( \text{sigma}[(j-1) \times \text{pdsig} + i - 1] \) and if \( \text{order} = \text{Nag\_RowMajor} \), the \((i, j)\)th element of the matrix is stored in \( \text{sigma}[(i-1) \times \text{pdsig} + j - 1] \).

\textit{On entry:} the \( n \) by \( n \) variance-covariance matrix \( \Sigma \). Only the lower triangle is referenced.

\textit{Constraint:} the matrix \( \Sigma \) must be positive-definite.

9: \( \text{pdsig} \) – Integer

\textit{Input}

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

\textit{Constraint:} \( \text{pdsig} \geq n \).
10:  I – Integer
     On entry: the required number of cumulants, and moments if specified.
     Constraint: $1 \leq I \leq 12.$

11:  rkum[I] – double
     On exit: the I cumulants of the quadratic form.

12:  rmom[dim] – double
     Output
     Note: the dimension, dim, of the array rmom must be at least I when
     mom = Nag_ComputeMoments and at least 1 otherwise.
     On exit: if mom = Nag_ComputeMoments, the I moments of the quadratic form.

13:  fail – NagError *
     Input/Output
     The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT
     On entry, n = ⟨value⟩.
     Constraint: n > 1.
     On entry, pda = ⟨value⟩.
     Constraint: pda > 0.
     On entry, pdsig = ⟨value⟩.
     Constraint: pdsig > 0.
     On entry, I = ⟨value⟩.
     Constraint: $1 \leq I \leq 12.$
     On entry, I = ⟨value⟩.
     Constraint: $I \geq 1.$

NE_INT_2
     On entry, pda = ⟨value⟩, n = ⟨value⟩.
     Constraint: pda $\geq$ n.
     On entry, pdsig = ⟨value⟩, n = ⟨value⟩.
     Constraint: pdsig $\geq$ n.

NE_POS_DEF
     On entry, sigma is not positive-definite.

NE_ALLOC_FAIL
     Memory allocation failed.

NE_BAD_PARAM
     On entry, parameter ⟨value⟩ 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

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

8 Further Comments

None.

9 Example

The example is given by Magnus and Pesaran (1993) 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

\[ Q = \sum_{i=2}^{n} y_i y_{t-1} \]

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

\[ A(i+1, i) = \frac{1}{2}, \quad i = 1, 2, \ldots, n - 1; \]
\[ A(i, j) = 0, \quad \text{otherwise}. \]

The value of \(\Sigma\) can be computed using the relationships

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

and

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

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.

9.1 Program Text

/* nag_moments_quad_form (g01nac) Example Program. */
/* Copyright 2001 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]
    #endif

    // Program code...
}
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;
Vprintf("g01nac Example Program Results\n");
/* Skip heading in data file */
Vscanf("\%*[\'\n\"");
Vscanf("\%f\%f*[\'\n\", &beta, &con");
Vscanf("\%ld\%ld*[\'\n\", &n, &l");

// 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)) )
{
   Vprintf("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);
    }
g01nac(order, Nag_ComputeMoments, Nag_MeanInclude,
   n, a, n, emu, sigma, n, l, rkum, rmom, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from g01nac.\n", fail.message);
    exit_status = 1;
    goto END;
}

Vprintf("\n");
Vprintf(" n = \%1d beta = \%6.3f con = \%6.3f\n", n, beta, con);
Vprintf("\n");
Vprintf(" Cumulants Moments\n");
Vprintf("\n");
for (i = 1; i <= l; ++i)
    Vprintf("\%1d\%12.4e \%12.4e\n", i, rkum[i - 1], rmom[i - 1]);
} END: 
if (a) NAG_FREE(a);
if (emu) NAG_FREE(emu);
if (rkum) NAG_FREE(rkum);
if (rmom) NAG_FREE(rmom);
if (sigma) NAG_FREE(sigma);
return exit_status;
}

9.2 Program Data

g01nac Example Program Data
0.8 1.0 : BETA, CON
10 4 : N, L

9.3 Program Results

g01nac Example Program Results

n = 10 beta = 0.800 con = 1.000

<table>
<thead>
<tr>
<th>Cumulants</th>
<th>Moments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.7517e+01</td>
</tr>
<tr>
<td>2</td>
<td>3.5010e+02</td>
</tr>
<tr>
<td>3</td>
<td>1.6091e+04</td>
</tr>
<tr>
<td>4</td>
<td>1.1700e+06</td>
</tr>
</tbody>
</table>