

## NAG Library Function Document

### nag\_multi\_normal\_pdf\_vector (g01lbc)

#### 1 Purpose

nag\_multi\_normal\_pdf\_vector (g01lbc) returns a number of values of the probability density function (PDF), or its logarithm, for the multivariate Normal (Gaussian) distribution.

#### 2 Specification

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

void nag_multi_normal_pdf_vector (Nag_Boolean ilog, Integer k, Integer n,
    const double x[], Integer pdx, const double xmu[], Nag_MatrixType iuld,
    const double sig[], Integer pdsig, double pdf[], Integer *rank,
    NagError *fail)
```

#### 3 Description

The probability density function,  $f(X : \mu, \Sigma)$  of an  $n$ -dimensional multivariate Normal distribution with mean vector  $\mu$  and  $n$  by  $n$  variance-covariance matrix  $\Sigma$ , is given by

$$f(X : \mu, \Sigma) = ((2\pi)^n |\Sigma|)^{-1/2} \exp\left(-\frac{1}{2}(X - \mu)^T \Sigma^{-1}(X - \mu)\right).$$

If the variance-covariance matrix,  $\Sigma$ , is not of full rank then the probability density function, is calculated as

$$f(X : \mu, \Sigma) = ((2\pi)^r \text{pdet}(\Sigma))^{-1/2} \exp\left(-\frac{1}{2}(X - \mu)^T \Sigma^-(X - \mu)\right)$$

where  $\text{pdet}(\Sigma)$  is the pseudo-determinant,  $\Sigma^-$  a generalized inverse of  $\Sigma$  and  $r$  its rank.

nag\_multi\_normal\_pdf\_vector (g01lbc) evaluates the PDF at  $k$  points with a single call.

#### 4 References

None.

#### 5 Arguments

- 1: **ilog** – Nag\_Boolean *Input*  
*On entry:* the value of **ilog** determines whether the logarithmic value is returned in PDF.  
**ilog** = Nag\_FALSE  
 $f(X : \mu, \Sigma)$ , the probability density function is returned.  
**ilog** = Nag\_TRUE  
 $\log(f(X : \mu, \Sigma))$ , the logarithm of the probability density function is returned.
- 2: **k** – Integer *Input*  
*On entry:*  $k$ , the number of points the PDF is to be evaluated at.  
*Constraint:*  $k \geq 0$ .

- 3: **n** – Integer *Input*  
*On entry:*  $n$ , the number of dimensions.  
*Constraint:*  $n \geq 2$ .
- 4: **x**[*dim*] – const double *Input*  
**Note:** the dimension, *dim*, of the array **x** must be at least  $\mathbf{pdx} \times \mathbf{k}$ .  
Where  $\mathbf{X}(i, j)$  appears in this document, it refers to the array element  $\mathbf{x}[(j-1) \times \mathbf{pdx} + i - 1]$ .  
*On entry:*  $X$ , the matrix of  $k$  points at which to evaluate the probability density function, with the  $i$ th dimension for the  $j$ th point held in  $\mathbf{X}(i, j)$ .
- 5: **pdx** – Integer *Input*  
*On entry:* the stride separating matrix row elements in the array **x**.  
*Constraint:*  $\mathbf{pdx} \geq \mathbf{n}$ .
- 6: **xmu**[**n**] – const double *Input*  
*On entry:*  $\mu$ , the mean vector of the multivariate Normal distribution.
- 7: **iuld** – Nag\_MatrixType *Input*  
*On entry:* indicates the form of  $\Sigma$  and how it is stored in **sig**.  
**iuld** = Nag\_LowerMatrix  
**sig** holds the lower triangular portion of  $\Sigma$ .  
**iuld** = Nag\_UpperMatrix  
**sig** holds the upper triangular portion of  $\Sigma$ .  
**iuld** = Nag\_DiagonalMatrix  
 $\Sigma$  is a diagonal matrix and **sig** only holds the diagonal elements.  
**iuld** = Nag\_LowerFactored  
**sig** holds the lower Cholesky decomposition,  $L$  such that  $LL^T = \Sigma$ .  
**iuld** = Nag\_UpperFactored  
**sig** holds the upper Cholesky decomposition,  $U$  such that  $U^T U = \Sigma$ .  
*C o n s t r a i n t :* **iuld** = Nag\_LowerMatrix, Nag\_UpperMatrix, Nag\_DiagonalMatrix, Nag\_LowerFactored or Nag\_UpperFactored.
- 8: **sig**[*dim*] – const double *Input*  
**Note:** the dimension, *dim*, of the array **sig** must be at least  $\mathbf{pdsig} \times \mathbf{n}$ .  
Where  $\mathbf{SIG}(i, j)$  appears in this document, it refers to the array element  $\mathbf{sig}[(j-1) \times \mathbf{pdsig} + i - 1]$ .  
*On entry:* information defining the variance-covariance matrix,  $\Sigma$ .  
**iuld** = Nag\_LowerMatrix or Nag\_UpperMatrix  
**sig** must hold the lower or upper portion of  $\Sigma$ , with  $\Sigma_{ij}$  held in  $\mathbf{SIG}(i, j)$ . The supplied variance-covariance matrix must be positive semidefinite.  
**iuld** = Nag\_DiagonalMatrix  
 $\Sigma$  is a diagonal matrix and the  $i$ th diagonal element,  $\Sigma_{ii}$ , must be held in  $\mathbf{SIG}(1, i)$   
**iuld** = Nag\_LowerFactored or Nag\_UpperFactored  
**sig** must hold  $L$  or  $U$ , the lower or upper Cholesky decomposition of  $\Sigma$ , with  $L_{ij}$  or  $U_{ij}$  held in  $\mathbf{SIG}(i, j)$ , depending on the value of **iuld**. No check is made that  $LL^T$  or  $U^T U$  is a valid variance-covariance matrix. The diagonal elements of the supplied  $L$  or  $U$  must be greater than zero

- 9: **pdsig** – Integer *Input*  
*On entry:* the stride separating matrix row elements in the array **sig**.  
*Constraints:*  
 if **iuld** = Nag\_DiagonalMatrix, **pdsig**  $\geq$  1;  
 otherwise **pdsig**  $\geq$  **n**.
- 10: **pdf[k]** – double *Output*  
*On exit:*  $f(X : \mu, \Sigma)$  or  $\log(f(X : \mu, \Sigma))$  depending on the value of **ilog**.
- 11: **rank** – Integer \* *Output*  
*On exit:*  $r$ , rank of  $\Sigma$ .
- 12: **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 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

### NE\_ARRAY\_SIZE

On entry, **pdsig** =  $\langle value \rangle$ .

Constraint: if **iuld** = Nag\_DiagonalMatrix, **pdsig**  $\geq$  1.

On entry, **pdsig** =  $\langle value \rangle$ .

Constraint: if **iuld**  $\neq$  Nag\_DiagonalMatrix, **pdsig**  $\geq$  **n**.

On entry, **pdx** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

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

### NE\_BAD\_PARAM

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

### NE\_DIAG\_ELEMENTS

On entry, at least one diagonal element of  $\Sigma$  is less than or equal to 0.

### NE\_INT

On entry, **k** =  $\langle value \rangle$ .

Constraint: **k**  $\geq$  0.

On entry, **n** =  $\langle value \rangle$ .

Constraint: **n**  $\geq$  2.

### 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 How to Use the NAG Library and its Documentation for further information.

**NE\_MAT\_NOT\_POS\_DEF**

On entry,  $\Sigma$  is not positive definite and eigenvalue decomposition failed.

**NE\_NO\_LICENCE**

Your licence key may have expired or may not have been installed correctly.  
See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

**NE\_NOT\_POS\_SEM\_DEF**

On entry,  $\Sigma$  is not positive semidefinite.

**7 Accuracy**

Not applicable.

**8 Parallelism and Performance**

nag\_multi\_normal\_pdf\_vector (g01lbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_multi\_normal\_pdf\_vector (g01lbc) 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 prints the value of the multivariate Normal PDF at a number of different points.

**10.1 Program Text**

```

/* nag_multi_normal_pdf_vector (g01lbc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg01.h>
#include <nagx04.h>

#define X(I, J)    x[(J) *pdx + I]
#define SIG(I, J) sig[(J) *pdsig + I]

int main(void)
{
  /* Integer scalar and array declarations */
  Integer i, j, k, n, pdx, pdsig, rank;
  Integer exit_status = 0;

```

```

/* NAG structures and types */
NagError fail;
Nag_MatrixType iuld;
Nag_Boolean ilog;

/* Double scalar and array declarations */
double *x = 0, *xmu = 0, *sig = 0, *pdf = 0;

/* Character scalar and array declarations */
char ciuld[40], cilog[40];

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

printf("nag_multi_normal_pdf_vector (g01lbc) Example Program Results\n\n");
fflush(stdout);

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

/* Read in the problem size */
#ifdef _WIN32
scanf_s("%" NAG_IFMT "%" NAG_IFMT "%39s%39s%*[\n] ", &k, &n, ciuld,
        (unsigned)_countof(ciuld), cilog, (unsigned)_countof(cilog));
#else
scanf("%" NAG_IFMT "%" NAG_IFMT "%39s%39s%*[\n] ", &k, &n, ciuld, cilog);
#endif
ilog = (Nag_Boolean) nag_enum_name_to_value(cilog);
iuld = (Nag_MatrixType) nag_enum_name_to_value(ciuld);

pdx = n;
pdsig = (iuld == Nag_DiagonalMatrix) ? 1 : n;
if (!(x = NAG_ALLOC(pdx * k, double)) ||
    !(xmu = NAG_ALLOC(n, double)) ||
    !(sig = NAG_ALLOC(pdsig * n, double)) || !(pdf = NAG_ALLOC(k, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read in and echo the vector of means */
for (i = 0; i < n; i++)
#ifdef _WIN32
scanf_s("%lf", &xmu[i]);
#else
scanf("%lf", &xmu[i]);
#endif
#ifdef _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif

printf("Vector of Means:\n");
for (i = 0; i < n; i++)
    printf("%8.4f ", xmu[i]);
printf("\n\n");
fflush(stdout);

/* Read in and echo the covariance matrix */
if (iuld == Nag_DiagonalMatrix) {
    for (i = 0; i < n; i++)
#ifdef _WIN32
scanf_s("%lf", &SIG(1, i));

```

```

#else
    scanf("%lf", &SIG(1, i));
#endif
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

    printf("Diagonal Elements of Covariance Matrix:\n");
    for (i = 0; i < n; i++)
        printf("%8.4f ", SIG(1, i));
    printf("\n\n");

}
else if (iuld == Nag_LowerMatrix || iuld == Nag_LowerFactored) {

    for (i = 0; i < n; i++) {
        for (j = 0; j <= i; j++)
#ifdef _WIN32
            scanf_s("%lf", &SIG(i, j));
#else
            scanf("%lf", &SIG(i, j));
#endif
#ifdef _WIN32
            scanf_s("%*[\n] ");
#else
            scanf("%*[\n] ");
#endif
    }
    /* Print details of Covariance matrix using
    * nag_gen_real_mat_print (x04cac)
    */
    if (iuld == Nag_LowerMatrix)
        nag_gen_real_mat_print(Nag_ColMajor, Nag_LowerMatrix, Nag_NonUnitDiag,
                               n, n, sig, pdsig, "Covariance Matrix:", NULL,
                               &fail);
    else
        nag_gen_real_mat_print(Nag_ColMajor, Nag_LowerMatrix, Nag_NonUnitDiag,
                               n, n, sig, pdsig,
                               "Lower Triangular Cholesky Factor "
                               "of Covariance Matrix:", NULL, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
              fail.message);
        exit_status = 1;
        goto END;
    }

}
else {
    /* Upper triangle of matrix or factor is stored */

    for (i = 0; i < n; i++) {
        for (j = i; j < n; j++)
#ifdef _WIN32
            scanf_s("%lf", &SIG(i, j));
#else
            scanf("%lf", &SIG(i, j));
#endif
#ifdef _WIN32
            scanf_s("%*[\n] ");
#else
            scanf("%*[\n] ");
#endif
    }
    /* Print details of Covariance matrix using
    * nag_gen_real_mat_print (x04cac)
    */
    if (iuld == Nag_UpperMatrix)
        nag_gen_real_mat_print(Nag_ColMajor, Nag_UpperMatrix, Nag_NonUnitDiag,

```

```

        n, n, sig, pdsig, "Covariance Matrix:", NULL,
        &fail);
    else
        nag_gen_real_mat_print(Nag_ColMajor, Nag_UpperMatrix, Nag_NonUnitDiag,
            n, n, sig, pdsig,
            "Upper Triangular Cholesky Factor "
            "of Covariance Matrix:", NULL, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n",
            fail.message);
        exit_status = 1;
        goto END;
    }
}

/* Read in the points at which to evaluate the PDF */
for (j = 0; j < k; j++)
    for (i = 0; i < n; i++)
#ifdef _WIN32
        scanf_s("%lf", &X(i, j));
#else
        scanf("%lf", &X(i, j));
#endif

/* nag_multi_normal_pdf_vector (g01lbc): evaluate the PDF */
nag_multi_normal_pdf_vector(ilog, k, n, x, pdx, xmu, iuld, sig, pdsig,
    pdf, &rank, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_multi_normal_pdf_vector (g01lbc).\n%s\n",
        fail.message);
    exit_status = 1;
    goto END;
}

/* Display results */
printf("\nRank of the covariance matrix: %" NAG_IFMT "\n\n", rank);
if (ilog)
    printf("      log(PDF)                X\n");
else
    printf("      PDF                X\n");
printf("-----\n");
for (j = 0; j < k; j++) {
    printf(" %13.4e", pdf[j]);
    for (i = 0; i < n; i++)
        printf(" %8.4f", X(i, j));
    printf("\n");
}

END:
    NAG_FREE(x);
    NAG_FREE(xmu);
    NAG_FREE(sig);
    NAG_FREE(pdf);

    return exit_status;
}

```

## 10.2 Program Data

```

nag_multi_normal_pdf_vector (g01lbc) Example Program Data
  2 4 Nag_LowerMatrix Nag_FALSE : k,n,iuld,ilog
  0.10 0.20 0.30 0.40 : End xmu
  4.16
 -3.12 5.03
  0.56 -0.83 0.76
 -0.10 1.18 0.34 1.18 : End sig
  1.00 1.00 1.00 1.00
  1.00 2.00 3.00 4.00 : End of x

```

### 10.3 Program Results

nag\_multi\_normal\_pdf\_vector (g01lbc) Example Program Results

Vector of Means:

0.1000 0.2000 0.3000 0.4000

Covariance Matrix:

	1	2	3	4
1	4.1600			
2	-3.1200	5.0300		
3	0.5600	-0.8300	0.7600	
4	-0.1000	1.1800	0.3400	1.1800

Rank of the covariance matrix: 4

PDF	X			
	1.0000	2.0000	3.0000	4.0000
3.0307e-03	1.0000	1.0000	1.0000	1.0000
4.5232e-06	1.0000	2.0000	3.0000	4.0000

---