NAG Library Function Document

nag_dgemv (f16pac)

1 Purpose

nag_dgemv (f16pac) performs matrix-vector multiplication for a real general matrix.

2 Specification

```
#include <nag.h>
#include <nagf16.h>
```

```
void nag_dgemv (Nag_OrderType order, Nag_TransType trans, Integer m,
     Integer n, double alpha, const double a[], Integer pda,
     const double x[], Integer incx, double beta, double y[], Integer incy,
     NagError *fail)
```

3 Description

nag_dgemv (f16pac) performs one of the matrix-vector operations

$$y \leftarrow \alpha Ax + \beta y, \qquad \text{or} \qquad y \leftarrow \alpha A^{\mathrm{T}}x + \beta y,$$

where A is an m by n real matrix, x and y are real vectors, and α and β are real scalars.

If $m = 0$ or $n = 0$, no operation is performed.

4 References

Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001) *Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard* University of Tennessee, Knoxville, Tennessee http://www.netlib.org/blas/blast-forum/blas-report.pdf

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: **trans** – Nag_TransType *Input*

On entry: specifies the operation to be performed.

trans = Nag_NoTrans
$y \leftarrow \alpha Ax + \beta y$.

trans = Nag_Trans or Nag_ConjTrans
$y \leftarrow \alpha A^{\mathrm{T}}x + \beta y$.

Constraint: **trans** = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

3: **m** – Integer *Input*

On entry: m, the number of rows of the matrix A.

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

4: **n** – Integer *Input*

On entry: n, the number of columns of the matrix A.

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

5: **alpha** – double *Input*

On entry: the scalar α.

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

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

$\max(1, \mathbf{pda} \times \mathbf{n})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{m} \times \mathbf{pda})$ when **order** = Nag_RowMajor.

If **order** = 'Nag_ColMajor', A_{ij} is stored in $\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$.

If **order** = 'Nag_RowMajor', A_{ij} is stored in $\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$.

On entry: the m by n matrix A.

7: **pda** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor, $\mathbf{pda} \geq \max(1, \mathbf{m})$;
if **order** = Nag_RowMajor, $\mathbf{pda} \geq \mathbf{n}$.

8: **x**[*dim*] – const double *Input*

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

$\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incx}|)$ when **trans** = Nag_NoTrans;
$\max(1, 1 + (\mathbf{m} - 1)|\mathbf{incx}|)$ when **trans** = Nag_Trans or Nag_ConjTrans.

On entry: the vector x.

9: **incx** – Integer *Input*

On entry: the increment in the subscripts of **x** between successive elements of x.

Constraint: $\mathbf{incx} \neq 0$.

10: **beta** – double *Input*

On entry: the scalar β.

11: **y**[*dim*] – double *Input/Output*

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

$\max(1, 1 + (\mathbf{m} - 1)|\mathbf{incy}|)$ when **trans** = Nag_NoTrans;
$\max(1, 1 + (\mathbf{n} - 1)|\mathbf{incy}|)$ when **trans** = Nag_Trans or Nag_ConjTrans.

On entry: the vector y.

If $\mathbf{beta} = 0$, **y** need not be set.

On exit: the updated vector y.

12: **incy** – Integer *Input*

On entry: the increment in the subscripts of **y** between successive elements of y.

Constraint: **incy** $\neq 0$.

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.

NE_BAD_PARAM

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

NE_INT

On entry, **incx** $= \langle$*value*$\rangle$.
Constraint: **incx** $\neq 0$.

On entry, **incy** $= \langle$*value*$\rangle$.
Constraint: **incy** $\neq 0$.

On entry, **m** $= \langle$*value*$\rangle$.
Constraint: **m** ≥ 0.

On entry, **n** $= \langle$*value*$\rangle$.
Constraint: **n** ≥ 0.

NE_INT_2

On entry, **pda** $= \langle$*value*$\rangle$, **m** $= \langle$*value*$\rangle$.
Constraint: **pda** $\geq \max(1, \mathbf{m})$.

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

7 Accuracy

The BLAS standard requires accurate implementations which avoid unnecessary over/underflow (see Section 2.7 of Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001)).

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

This example computes the matrix-vector product

$$y = \alpha Ax + \beta y$$

where

$$A = \begin{pmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \\ 5.0 & 6.0 \end{pmatrix},$$

$$x = \begin{pmatrix} -1.0 \\ 2.0 \end{pmatrix},$$

$$y = \begin{pmatrix} 1.0 \\ 2.0 \\ 3.0 \end{pmatrix},$$

$$\alpha = 1.5 \text{ and } \beta = 1.0.$$

10.1 Program Text

```
/* nag_dgemv (f16pac) Example Program.
 *
 * Copyright 2005 Numerical Algorithms Group.
 *
 * Mark 8, 2005.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf16.h>

int main(void)
{

  /* Scalars */
  double        alpha, beta;
  Integer       exit_status, i, incx, incy, j, m, n, pda, xlen, ylen;

  /* Arrays */
  double        *a = 0, *x = 0, *y = 0;
  char          nag_enum_arg[40];

  /* Nag Types */
  NagError      fail;
  Nag_OrderType order;
  Nag_TransType trans;

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

  exit_status = 0;
  INIT_FAIL(fail);

  printf("nag_dgemv (f16pac) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  /* Read the problem dimensions */
  scanf("%ld%ld%*[^\n] ", &m, &n);

  /* Read the transpose parameter */
  scanf("%39s%*[^\n] ", nag_enum_arg);
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  trans = (Nag_TransType) nag_enum_name_to_value(nag_enum_arg);
  /* Read scalar parameters */
  scanf("%lf%lf%*[^\n] ", &alpha, &beta);
  /* Read increment parameters */
  scanf("%ld%ld%*[^\n] ", &incx, &incy);

#ifdef NAG_COLUMN_MAJOR
  pda = m;
#else
  pda = n;
#endif

  if (trans == Nag_NoTrans)
    {
      xlen = MAX(1, 1 + (n - 1)*ABS(incx));
      ylen = MAX(1, 1 + (m - 1)*ABS(incy));
    }
  else
    {
      xlen = MAX(1, 1 + (m - 1)*ABS(incx));
      ylen = MAX(1, 1 + (n - 1)*ABS(incy));
    }

  if (m > 0 && n > 0)
    {
      /* Allocate memory */
      if (!(a = NAG_ALLOC(m*n, double)) ||
          !(x = NAG_ALLOC(xlen, double)) ||
          !(y = NAG_ALLOC(ylen, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
    }
  else
    {
      printf("Invalid m or n\n");
      exit_status = 1;
      return exit_status;
    }

  /* Input matrix A and vectors x and y */

  for (i = 1; i <= m; ++i)
    {
      for (j = 1; j <= n; ++j)
        scanf("%lf", &A(i, j));
      scanf("%*[^\n] ");
    }
  for (i = 1; i <= xlen; ++i)
    scanf("%lf%*[^\n] ", &x[i - 1]);
  for (i = 1; i <= ylen; ++i)
    scanf("%lf%*[^\n] ", &y[i - 1]);

  /* nag_dgemv (f16pac).
   * Matrix-vector multiply.
   *
   */
  nag_dgemv(order, trans, m, n, alpha, a, pda, x, incx,
            beta, y, incy, &fail);
  if (fail.code != NE_NOERROR)
```

```
    {
      printf("Error from nag_dgemv.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print output vector y */
  printf("%s\n", "  y");
  for (i = 1; i <= ylen; ++i)
    {
      printf("%11f\n", y[i-1]);
    }

 END:
  NAG_FREE(a);
  NAG_FREE(x);
  NAG_FREE(y);

  return exit_status;
}
```

10.2 Program Data

```
nag_dgemv (f16pac) Example Program Data
  3 2                 : m, n the dimensions of matrix A
  Nag_NoTrans         : trans
  1.5 1.0             : alpha, beta
  1 1                 : incx, incy
  1.0 2.0
  3.0 4.0
  5.0 6.0             : the end of matrix A
   -1.0
    2.0               : the end of vector x
    1.0
    2.0
    3.0               : the end of vector y
```

10.3 Program Results

```
nag_dgemv (f16pac) Example Program Results

  y
   5.500000
   9.500000
  13.500000
```