NAG Library Function Document

nag_dgemm (f16yac)

1 Purpose

nag_dgemm (f16yac) performs matrix-matrix multiplication for a real general matrix.

2 Specification

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

```
void nag_dgemm (Nag_OrderType order, Nag_TransType transa,
     Nag_TransType transb, Integer m, Integer n, Integer k, double alpha,
     const double a[], Integer pda, const double b[], Integer pdb,
     double beta, double c[], Integer pdc, NagError *fail)
```

3 Description

nag_dgemm (f16yac) performs one of the matrix-matrix operations

$$\begin{array}{lcl} C \leftarrow \alpha AB + \beta C, & & C \leftarrow \alpha A^{\mathrm{T}} B + \beta C, \\ C \leftarrow \alpha AB^{\mathrm{T}} + \beta C & \text{or} & C \leftarrow \alpha A^{\mathrm{T}} B^{\mathrm{T}} + \beta C, \end{array}$$

where A, B and C are real matrices, and α and β are real scalars; C is always m by n.

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: **transa** – Nag_TransType *Input*

On entry: specifies whether the operation involves A or A^{T}.

transa = Nag_NoTrans
It involves A.

transa = Nag_Trans or Nag_ConjTrans
It involves A^{T}.

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

3: **transb** – Nag_TransType *Input*

On entry: specifies whether the operation involves B or B^{T}.

transb = Nag_NoTrans
It involves B.

transb = Nag_Trans or Nag_ConjTrans
It involves B^{T}.

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

4: **m** – Integer *Input*

On entry: m, the number of rows of the matrix C; the number of rows of A if **transa** = Nag_NoTrans, or the number of columns of A if **transa** = Nag_Trans or Nag_ConjTrans.

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

5: **n** – Integer *Input*

On entry: n, the number of columns of the matrix C; the number of columns of B if **transb** = Nag_NoTrans, or the number of rows of B if **transb** = Nag_Trans or Nag_ConjTrans.

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

6: **k** – Integer *Input*

On entry: k, the number of columns of A if **transa** = Nag_NoTrans, or the number of rows of A if **transa** = Nag_Trans or Nag_ConjTrans; the number of rows of B if **transb** = Nag_NoTrans, or the number of columns of B if **transb** = Nag_Trans or Nag_ConjTrans.

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

7: **alpha** – double *Input*

On entry: the scalar α.

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

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

$\max(1, \mathbf{pda} \times \mathbf{k})$ when **transa** = Nag_NoTrans and **order** = Nag_ColMajor;
$\max(1, \mathbf{m} \times \mathbf{pda})$ when **transa** = Nag_NoTrans and **order** = Nag_RowMajor;
$\max(1, \mathbf{pda} \times \mathbf{m})$ when **transa** = Nag_Trans or Nag_ConjTrans and **order** = Nag_ColMajor;
$\max(1, \mathbf{k} \times \mathbf{pda})$ when **transa** = Nag_Trans or Nag_ConjTrans and **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 matrix A; A is m by k if **transa** = Nag_NoTrans, or k by m if **transa** = Nag_Trans or Nag_ConjTrans.

9: **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,

if **transa** = Nag_NoTrans, $\mathbf{pda} \geq \max(1, \mathbf{m})$;
if **transa** = Nag_Trans or Nag_ConjTrans, $\mathbf{pda} \geq \max(1, \mathbf{k})$.;

if **order** = Nag_RowMajor,

if **transa** = Nag_NoTrans, **pda** $\geq$ max(1, **k**);
if **transa** = Nag_Trans or Nag_ConjTrans, **pda** $\geq$ max(1, **m**)..

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

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

max(1, **pdb** $\times$ **n**) when **transb** = Nag_NoTrans and **order** = Nag_ColMajor;
max(1, **k** $\times$ **pdb**) when **transb** = Nag_NoTrans and **order** = Nag_RowMajor;
max(1, **pdb** $\times$ **k**) when **transb** = Nag_Trans or Nag_ConjTrans and **order** = Nag_ColMajor;
max(1, **n** $\times$ **pdb**) when **transb** = Nag_Trans or Nag_ConjTrans and **order** = Nag_RowMajor.

If **order** = Nag_ColMajor, B_{ij} is stored in **b**$[(j-1) \times$ **pdb** $+ i - 1]$.

If **order** = Nag_RowMajor, B_{ij} is stored in **b**$[(i-1) \times$ **pdb** $+ j - 1]$.

On entry: the matrix B; B is k by n if **transb** = Nag_NoTrans, or n by k if **transb** = Nag_Trans or Nag_ConjTrans.

11: **pdb** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor,

if **transb** = Nag_NoTrans, **pdb** $\geq$ max(1, **k**);
if **transb** = Nag_Trans or Nag_ConjTrans, **pdb** $\geq$ max(1, **n**).;

if **order** = Nag_RowMajor,

if **transb** = Nag_NoTrans, **pdb** $\geq$ max(1, **n**);
if **transb** = Nag_Trans or Nag_ConjTrans, **pdb** $\geq$ max(1, **k**)..

12: **beta** – double *Input*

On entry: the scalar β.

13: **c**[*dim*] – double *Input/Output*

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

max(1, **pdc** $\times$ **n**) when **order** = Nag_ColMajor;
max(1, **m** $\times$ **pdc**) when **order** = Nag_RowMajor.

If **order** = Nag_ColMajor, C_{ij} is stored in **c**$[(j-1) \times$ **pdc** $+ i - 1]$.

If **order** = Nag_RowMajor, C_{ij} is stored in **c**$[(i-1) \times$ **pdc** $+ j - 1]$.

On entry: the m by n matrix C.

If **beta** = 0, **c** need not be set.

On exit: the updated matrix C.

14: **pdc** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor, **pdc** $\geq$ max(1, **m**);
if **order** = Nag_RowMajor, **pdc** $\geq$ max(1, **n**).

15: **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_ENUM_INT_2

On entry, **transa** = ⟨*value*⟩, **k** = ⟨*value*⟩, **pda** = ⟨*value*⟩.
Constraint: if **transa** = Nag_NoTrans, **pda** $\geq \max(1, \mathbf{k})$.

On entry, **transa** = ⟨*value*⟩, **m** = ⟨*value*⟩, **pda** = ⟨*value*⟩.
Constraint: if **transa** = Nag_Trans or Nag_ConjTrans, **pda** $\geq \max(1, \mathbf{m})$.

On entry, **transa** = ⟨*value*⟩, **pda** = ⟨*value*⟩, **k** = ⟨*value*⟩.
Constraint: if **transa** = Nag_Trans or Nag_ConjTrans, **pda** $\geq \max(1, \mathbf{k})$.

On entry, **transa** = ⟨*value*⟩, **pda** = ⟨*value*⟩, **m** = ⟨*value*⟩.
Constraint: if **transa** = Nag_NoTrans, **pda** $\geq \max(1, \mathbf{m})$.

On entry, **transb** = ⟨*value*⟩, **k** = ⟨*value*⟩, **pdb** = ⟨*value*⟩.
Constraint: if **transb** = Nag_NoTrans, **pdb** $\geq \max(1, \mathbf{k})$.

On entry, **transb** = ⟨*value*⟩, **k** = ⟨*value*⟩, **pdb** = ⟨*value*⟩.
Constraint: if **transb** = Nag_Trans or Nag_ConjTrans, **pdb** $\geq \max(1, \mathbf{k})$.

On entry, **transb** = ⟨*value*⟩, **n** = ⟨*value*⟩, **pdb** = ⟨*value*⟩.
Constraint: if **transb** = Nag_NoTrans, **pdb** $\geq \max(1, \mathbf{n})$.

On entry, **transb** = ⟨*value*⟩, **n** = ⟨*value*⟩, **pdb** = ⟨*value*⟩.
Constraint: if **transb** = Nag_Trans or Nag_ConjTrans, **pdb** $\geq \max(1, \mathbf{n})$.

NE_INT

On entry, **k** = ⟨*value*⟩.
Constraint: **k** ≥ 0.

On entry, **m** = ⟨*value*⟩.
Constraint: **m** ≥ 0.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

NE_INT_2

On entry, **pdc** = ⟨*value*⟩, **m** = ⟨*value*⟩.
Constraint: **pdc** $\geq \max(1, \mathbf{m})$.

On entry, **pdc** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: **pdc** $\geq \max(1, \mathbf{n})$.

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.

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-matrix product

$$C = \alpha AB + \beta C$$

where

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

$$B = \begin{pmatrix} 1.0 & 2.0 \\ -2.0 & 1.0 \\ 3.0 & -1.0 \end{pmatrix},$$

$$C = \begin{pmatrix} -2.0 & 1.0 \\ 1.0 & 3.0 \\ 2.0 & -1.0 \end{pmatrix},$$

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

10.1 Program Text

```
/* nag_dgemm (f16yac) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 8, 2005.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf16.h>
#include <nagx04.h>

int main(void)
{
```

```
  /* Scalars */
  double        alpha, beta;
  Integer       exit_status, i, j, k, m, n, pda, pdb, pdc;

  /* Arrays */
  double        *a = 0, *b = 0, *c = 0;
  char          nag_enum_arg[40];

  /* Nag Types */
  NagError      fail;
  Nag_OrderType order;
  Nag_TransType transa;
  Nag_TransType transb;

#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 C(I, J) c[(J-1)*pdc + 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 C(I, J) c[(I-1)*pdc + J - 1]
  order = Nag_RowMajor;
#endif

  exit_status = 0;
  INIT_FAIL(fail);

  printf("nag_dgemm (f16yac) Example Program Results\n\n");

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

  /* Read the problem dimensions */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ",
          &m, &n, &k);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ",
          &m, &n, &k);
#endif

  /* Read the transpose parameters */
#ifdef _WIN32
  scanf_s("%39s%*[^\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n] ", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  transa = (Nag_TransType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s("%39s%*[^\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n] ", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  transb = (Nag_TransType) nag_enum_name_to_value(nag_enum_arg);
  /* Read scalar parameters */
#ifdef _WIN32
  scanf_s("%lf%lf%*[^\n] ", &alpha, &beta);
#else
  scanf("%lf%lf%*[^\n] ", &alpha, &beta);
```

```
#endif

#ifdef NAG_COLUMN_MAJOR
  pdc = m;
  if (transa == Nag_NoTrans && transb == Nag_NoTrans)
    {
      pda = m;
      pdb = k;
    }
  else if ((transa == Nag_Trans || transa == Nag_ConjTrans)
          && transb == Nag_NoTrans)
    {
      pda = k;
      pdb = k;
    }
  else if (transa == Nag_NoTrans &&
           (transb == Nag_Trans || transb == Nag_ConjTrans))
    {
      pda = m;
      pdb = n;
    }
  else
    {
      pda = k;
      pdb = n;
    }
#else
  pdc = n;
  if (transa == Nag_NoTrans && transb == Nag_NoTrans)
    {
      pda = k;
      pdb = n;
    }
  else if ((transa == Nag_Trans || transa == Nag_ConjTrans)
          && transb == Nag_NoTrans)
    {
      pda = m;
      pdb = n;
    }
  else if (transa == Nag_NoTrans &&
           (transb == Nag_Trans || transb == Nag_ConjTrans))
    {
      pda = k;
      pdb = k;
    }
  else
    {
      pda = m;
      pdb = k;
    }
#endif

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

```
  /* Input matrix A */
  if (transa == Nag_NoTrans)
    {
      for (i = 1; i <= m; ++i)
        {
          for (j = 1; j <= k; ++j)
#ifdef _WIN32
            scanf_s("%lf", &A(i, j));
#else
            scanf("%lf", &A(i, j));
#endif
#ifdef _WIN32
          scanf_s("%*[^\n] ");
#else
          scanf("%*[^\n] ");
#endif
        }
    }
  else
    {
      for (i = 1; i <= k; ++i)
        {
          for (j = 1; j <= m; ++j)
#ifdef _WIN32
            scanf_s("%lf", &A(i, j));
#else
            scanf("%lf", &A(i, j));
#endif
#ifdef _WIN32
          scanf_s("%*[^\n] ");
#else
          scanf("%*[^\n] ");
#endif
        }
    }

  /* Input matrix B */
  if (transb == Nag_NoTrans)
    {
      for (i = 1; i <= k; ++i)
        {
          for (j = 1; j <= n; ++j)
#ifdef _WIN32
            scanf_s("%lf", &B(i, j));
#else
            scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
          scanf_s("%*[^\n] ");
#else
          scanf("%*[^\n] ");
#endif
        }
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= k; ++j)
#ifdef _WIN32
            scanf_s("%lf", &B(i, j));
#else
            scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
          scanf_s("%*[^\n] ");
#else
          scanf("%*[^\n] ");
#endif
        }
    }
```

```
  /* Input matrix C */
  for (i = 1; i <= m; ++i)
    {
      for (j = 1; j <= n; ++j)
#ifdef _WIN32
        scanf_s("%lf", &C(i, j));
#else
        scanf("%lf", &C(i, j));
#endif
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
    }

  /* nag_dgemm (f16yac).
   * Matrix-matrix multiply.
   *
   */
  nag_dgemm(order, transa, transb, m, n, k, alpha, a, pda,
            b, pdb, beta, c, pdc, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dgemm.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print result */
  /* nag_gen_real_mat_print (x04cac).
   * Print real general matrix (easy-to-use)
   */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag,
                         m, n, c, pdc, "Matrix Matrix Product",
                         0, &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;
    }

 END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(c);

  return exit_status;
}
```

10.2 Program Data

```
nag_dgemm (f16yac) Example Program Data
  3 2 3               :Values of m, n, k
  Nag_NoTrans         : transa
  Nag_NoTrans         : transb
  1.5  1.0            : alpha, beta
  1.0  2.0  3.0
  3.0  4.0  5.0
  5.0  6.0 -1.0       :End of matrix A
  1.0  2.0
```

```
-2.0  1.0
 3.0 -1.0             :End of matrix B
-2.0  1.0
 1.0  3.0
 2.0 -1.0             :End of matrix C
```

10.3 Program Results

```
nag_dgemm (f16yac) Example Program Results

 Matrix Matrix Product
             1          2
 1      7.0000     2.5000
 2     16.0000    10.5000
 3    -13.0000    24.5000
```