NAG Library Function Document
nag_superlu_matrix_product (f11mkc)

1 Purpose

nag_superlu_matrix_product (f11mkc) computes a matrix-matrix or transposed matrix-matrix product involving a real, square, sparse nonsymmetric matrix stored in compressed column (Harwell–Boeing) format.

2 Specification

```c
#include <nag.h>
#include <nagf11.h>

void nag_superlu_matrix_product (Nag_OrderType order, Nag_TransType trans,
                                  Integer n, Integer m, double alpha, const Integer icolzp[],
                                  const Integer irowix[], const double a[], const double b[],
                                  Integer pdb, double beta, double c[], Integer pdc, NagError *fail)
```

3 Description

nag_superlu_matrix_product (f11mkc) computes either the matrix-matrix product $C \leftarrow \alpha AB + \beta C$, or the transposed matrix-matrix product $C \leftarrow \alpha A^T B + \beta C$, according to the value of the argument `trans`, where $A$ is a real $n$ by $n$ sparse nonsymmetric matrix, of arbitrary sparsity pattern with $nnz$ nonzero elements, $B$ and $C$ are $n$ by $m$ real dense matrices. The matrix $A$ is stored in compressed column (Harwell–Boeing) storage format. The array `a` stores all nonzero elements of $A$, while arrays `icolzp` and `irowix` store the compressed column indices and row indices of $A$ respectively.

4 References

None.

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 whether or not the matrix $A$ is transposed.

   `trans` = Nag_NoTrans
   $\alpha AB + \beta C$ is computed.

   `trans` = Nag_Trans
   $\alpha A^T B + \beta C$ is computed.

   *Constraint:* `trans` = Nag_NoTrans or Nag_Trans.
3: \[ n \] – Integer

\textit{On entry:} \( n \), the order of the matrix \( A \).

\textit{Constraint:} \( n \geq 0 \).

4: \[ m \] – Integer

\textit{On entry:} \( m \), the number of columns of matrices \( B \) and \( C \).

\textit{Constraint:} \( m \geq 0 \).

5: \[ \alpha \] – double

\textit{On entry:} \( \alpha \), the scalar factor in the matrix multiplication.

6: \[ \text{iolzp}[\dim] \] – const Integer

\textit{Note:} the dimension, \( \dim \), of the array \( \text{iolzp} \) must be at least \( n + 1 \).

\textit{On entry:} \( \text{iolzp}[i - 1] \) contains the index in \( A \) of the start of a new column. See Section 2.1.3 in the \texttt{f11} Chapter Introduction.

7: \[ \text{irowix}[\dim] \] – const Integer

\textit{Note:} the dimension, \( \dim \), of the array \( \text{irowix} \) must be at least \( \text{iolzp}[n] - 1 \), the number of nonzeros of the sparse matrix \( A \).

\textit{On entry:} the row index array of sparse matrix \( A \).

8: \[ a[\dim] \] – const double

\textit{Note:} the dimension, \( \dim \), of the array \( a \) must be at least \( \text{iolzp}[n] - 1 \), the number of nonzeros of the sparse matrix \( A \).

\textit{On entry:} the array of nonzero values in the sparse matrix \( A \).

9: \[ b[\dim] \] – const double

\textit{Note:} the dimension, \( \dim \), of the array \( b \) must be at least

\[
\max(1, \text{pdb} \times m) \quad \text{when order = Nag\_ColMajor;}
\max(1, n \times \text{pdb}) \quad \text{when order = Nag\_RowMajor.}
\]

The \((i,j)\)th element of the matrix \( B \) is stored in

\[
\begin{align*}
 b[(j - 1) \times \text{pdb} + i - 1] & \quad \text{when order = Nag\_ColMajor;}
 b[(i - 1) \times \text{pdb} + j - 1] & \quad \text{when order = Nag\_RowMajor.}
\end{align*}
\]

\textit{On entry:} the \( n \) by \( m \) matrix \( B \).

10: \[ \text{pdb} \] – Integer

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

\textit{Constraints:}

\[
\begin{align*}
 & \text{if order = Nag\_ColMajor, pdb} \geq \max(1, n); \\
 & \text{if order = Nag\_RowMajor, pdb} \geq \max(1, m).
\end{align*}
\]

11: \[ \beta \] – double

\textit{On entry:} the scalar factor \( \beta \).
12: \( c[\text{dim}] \) – double

\textit{Note:} the dimension, \( \text{dim} \), of the array \( c \) must be at least
\[ \max(1, \text{pdc} \times m) \] when \( \text{order} = \text{Nag\_ColMajor} \);
\[ \max(1, n \times \text{pdc}) \] when \( \text{order} = \text{Nag\_RowMajor} \).

The \((i,j)\)th element of the matrix \( C \) is stored in
\[ c[(j - 1) \times \text{pdc} + i - 1] \] when \( \text{order} = \text{Nag\_ColMajor} \);
\[ c[(i - 1) \times \text{pdc} + j - 1] \] when \( \text{order} = \text{Nag\_RowMajor} \).

On entry: the \( n \) by \( m \) matrix \( C \).

On exit: \( C \) is overwritten by \( \alpha AB + \beta C \) or \( \alpha A^T B + \beta C \) depending on the value of \( \text{trans} \).

13: \( \text{pdc} \) – Integer

\textit{Input}

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

\textit{Constraints:}
\[ \text{if } \text{order} = \text{Nag\_ColMajor}, \text{pdc} \geq \max(1, n); \]
\[ \text{if } \text{order} = \text{Nag\_RowMajor}, \text{pdc} \geq \max(1, m). \]

14: \( \text{fail} \) – NagError*

\textit{Input/Output}

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

6  \textbf{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 \( \langle \text{value} \rangle \) had an illegal value.

**NE\_INT**

On entry, \( m = \langle \text{value} \rangle \).

Constraint: \( m \geq 0 \).

On entry, \( n = \langle \text{value} \rangle \).

Constraint: \( n \geq 0 \).

On entry, \( \text{pdb} = \langle \text{value} \rangle \).

Constraint: \( \text{pdb} > 0 \).

On entry, \( \text{pdc} = \langle \text{value} \rangle \).

Constraint: \( \text{pdc} > 0 \).

**NE\_INT\_2**

On entry, \( \text{pdb} = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \).

Constraint: \( \text{pdb} \geq \max(1, m) \).

On entry, \( \text{pdb} = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).

Constraint: \( \text{pdb} \geq \max(1, n) \).

On entry, \( \text{pdc} = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \).

Constraint: \( \text{pdc} \geq \max(1, m) \).

On entry, \( \text{pdc} = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).

Constraint: \( \text{pdc} \geq \max(1, n) \).
7 Accuracy
Not applicable.

8 Parallelism and Performance
nag_superlu_matrix_product (f11mkc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

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 reads in a sparse matrix $A$ and a dense matrix $B$. It then calls nag_superlu_matrix_product (f11mkc) to compute the matrix-matrix product $C = AB$ and the transposed matrix-matrix product $C = A^T B$, where
\[
A = \begin{pmatrix}
  2.00 & 1.00 & 0 & 0 & 0 \\
  0 & 0 & 1.00 & -1.00 & 0 \\
  4.00 & 0 & 1.00 & 0 & 1.00 \\
  0 & 0 & 0 & 1.00 & 2.00 \\
  0 & -2.00 & 0 & 0 & 3.00
\end{pmatrix}
\quad \text{and} \quad
B = \begin{pmatrix}
  0.70 & 1.40 \\
  0.16 & 0.32 \\
  0.52 & 1.04 \\
  0.77 & 1.54 \\
  0.28 & 0.56
\end{pmatrix}.
\]

10.1 Program Text
/* nag_superlu_matrix_product (f11mkc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 8, 2005. */
*/
#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf11.h>

/* Table of constant values */
static double alpha = 1.;
static double beta = 0.;

int main(void)
{

Integer exit_status = 0, i, j, m, n, nnz;
double *a = 0, *b = 0, *c = 0;
Integer *icolzp = 0, *irowix = 0;
/* Nag types */
NagError fail;
Nag_OrderType order = Nag_ColMajor;
Nag_MatrixType matrix = Nag_GeneralMatrix;
Nag_DiagType diag = Nag_NonUnitDiag;
Nag_TransType trans;

INIT_FAIL(fail);

printf("nag_superlu_matrix_product (f11mkc) Example Program Results\n\n");
/* Skip heading in data file */
#ifdef _WIN32
scanf_s("%*[\n ] ");
#else
scanf("%*[\n ] ");
#endif /* Read order of matrix */
#ifdef _WIN32
scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[\n ] ", &n, &m);
#else
scanf("%"NAG_IFMT"%"NAG_IFMT"%*[\n ] ", &n, &m);
#endif /* Read the matrix A */
if (!(icolzp = NAG_ALLOC(n+1, Integer)))
{
printf("Allocation failure\n");
extit_status = -1;
goto END;
}
for (i = 0; i < n + 1; ++i)
#ifdef _WIN32
scanf_s("%"NAG_IFMT"%*[\n ] ", &icolzp[i]);
#else
scanf("%"NAG_IFMT"%*[\n ] ", &icolzp[i]);
#endif
nnz = icolzp[n] - 1;
/* Allocate memory */
if (!(!irowix = NAG_ALLOC(nnz, Integer)) ||
    !(a = NAG_ALLOC(nnz, double)) ||
    !(b = NAG_ALLOC(n * m, double)) ||
    !(c = NAG_ALLOC(n * m, double)))
{
printf("Allocation failure\n");
extit_status = -1;
goto END;
}
for (i = 0; i < nnz; ++i)
#ifdef _WIN32
scanf_s("%lf"NAG_IFMT"%*[\n ] ", &a[i], &irowix[i]);
#else
scanf("%lf"NAG_IFMT"%*[\n ] ", &a[i], &irowix[i]);
#endif /* Read the matrix B */
for (j = 0; j < m; ++j)
{
#ifdef _WIN32
scanf("%lf", &b[j*n + i]);
#else
scanf("%lf", &b[j*n + i]);
#endif
c[j*n + i] = 0.0;
#ifdef _WIN32
scanf_s("%*[\n ] ");
#else
...
/* Calculate matrix-matrix product */
trans = Nag_NoTrans;
/* nag_superlu_matrix_product (f11mkc).
* Real sparse nonsymmetric matrix matrix multiply,
* compressed column storage
*/
nag_superlu_matrix_product(order, trans, n, m, alpha, icolzp, irowix, a, b,
    n, beta, c, n, &fail);
if (fail.code != NE_NOERROR)
    { printf("Error from nag_superlu_matrix_product (f11mkc).\n", fail.message);
      exit_status = 1;
      goto END;
    }

/* Output results */
printf("
");
/* nag_gen_real_mat_print (x04cac).
* Print real general matrix (easy-to-use)
*/
fflush(stdout);
nag_gen_real_mat_print(order, matrix, diag, n, m, c, n,
    "Matrix-vector product", 0, &fail);
if (fail.code != NE_NOERROR)
    { printf("Error from nag_gen_real_mat_print (x04cac).\n", fail.message);
      exit_status = 1;
      goto END;
    }

/* Calculate transposed matrix-matrix product */
trans = Nag_Trans;
/* nag_superlu_matrix_product (f11mkc), see above. */
nag_superlu_matrix_product(order, trans, n, m, alpha, icolzp, irowix, a, b,
    n, beta, c, n, &fail);
if (fail.code != NE_NOERROR)
    { printf("Error from nag_superlu_matrix_product (f11mkc).\n", fail.message);
      exit_status = 1;
      goto END;
    }

/* Output results */
printf("
");
/* nag_gen_real_mat_print (x04cac), see above. */
fflush(stdout);
nag_gen_real_mat_print(order, matrix, diag, n, m, c, n,
    "Transposed matrix-vector product", 0, &fail);
if (fail.code != NE_NOERROR)
    { printf("Error from nag_gen_real_mat_print (x04cac).\n", fail.message);
      exit_status = 1;
      goto END;
    }

END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(c);
NAG_FREE(icolzp);
NAG_FREE(irowix);
return exit_status;
10.2 Program Data

nag_superlu_matrix_product (f11mkc) Example Program Data

\begin{verbatim}
5 2
1 3
5 7
9
12
icolzp(i) i=0..n
2. 1
4. 3
1. 1
-2. 5
1. 2
1. 3
-1. 2
1. 4
1. 3
-2. 4
3. 5
a(i) irowix(i) i=0..nnz-1
0.70 0.16 0.52 0.77 0.28
1.40 0.32 1.04 1.54 0.56
\end{verbatim}

matrix B

10.3 Program Results

nag_superlu_matrix_product (f11mkc) Example Program Results

\begin{verbatim}
Matrix-vector product
1 2
1 1.5600 3.1200
2 -0.2500 -0.5000
3 3.6000 7.2000
4 1.3300 2.6600
5 0.5200 1.0400

Transposed matrix-vector product
1 2
1 3.4800 6.9600
2 0.1400 0.2800
3 0.6800 1.3600
4 0.6100 1.2200
5 2.9000 5.8000
\end{verbatim}