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

```
#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 $C \leftarrow \alpha A^{\mathrm{T}}B + \beta C$, 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
$\alpha A^{\mathrm{T}}B + \beta C$ is computed.

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

3: **n** – Integer *Input*

On entry: n, the order of the matrix A.

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

4: **m** – Integer *Input*

On entry: m, the number of columns of matrices B and C.

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

5: **alpha** – double *Input*

On entry: α, the scalar factor in the matrix multiplication.

6: **icolzp**[dim] – const Integer *Input*

Note: the dimension, *dim*, of the array **icolzp** must be at least $\mathbf{n} + 1$.

On entry: **icolzp**[$i - 1$] contains the index in A of the start of a new column. See Section 2.1.3 in the f11 Chapter Introduction.

7: **irowix**[dim] – const Integer *Input*

Note: the dimension, *dim*, of the array **irowix** must be at least **icolzp**[**n**] $- 1$, the number of nonzeros of the sparse matrix A.

On entry: the row index array of sparse matrix A.

8: **a**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **a** must be at least **icolzp**[**n**] $- 1$, the number of nonzeros of the sparse matrix A.

On entry: the array of nonzero values in the sparse matrix A.

9: **b**[dim] – const double *Input*

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

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

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

b[$(j - 1) \times \mathbf{pdb} + i - 1$] when **order** = Nag_ColMajor;
b[$(i - 1) \times \mathbf{pdb} + j - 1$] when **order** = Nag_RowMajor.

On entry: the n by m matrix B.

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

11: **beta** – double *Input*

On entry: the scalar factor β.

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

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

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

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

$\mathbf{c}[(j-1) \times \mathbf{pdc} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{c}[(i-1) \times \mathbf{pdc} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the n by m matrix C.

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

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

14: **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_INT

On entry, $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{m} \geq 0$.

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 0$.

On entry, $\mathbf{pdb} = \langle value \rangle$.
Constraint: $\mathbf{pdb} > 0$.

On entry, $\mathbf{pdc} = \langle value \rangle$.
Constraint: $\mathbf{pdc} > 0$.

NE_INT_2

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

On entry, $\mathbf{pdb} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{pdb} \geq \max(1, \mathbf{n})$.

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

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

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

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 $C = A^{\mathrm{T}}B$, 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");
      exit_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");
      exit_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)
    {
      for (i = 0; i < n; ++i)
        {
#ifdef _WIN32
          scanf_s("%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
```

```
      scanf("%*[^\n] ");
#endif
    }
  /* 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%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Output results */
  printf("\n");
  /* 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%s\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%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Output results */
  printf("\n");
  /* 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%s\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
  5 2                n, m
 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     matrix B
```

10.3 Program Results

```
nag_superlu_matrix_product (f11mkc) Example Program Results


 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
```