NAG Library Function Document

nag_dspsvx (f07pbc)

1 Purpose

nag_dspsvx (f07pbc) uses the diagonal pivoting factorization

$$A = UDU^{\mathrm{T}} \quad \text{or} \quad A = LDL^{\mathrm{T}}$$

to compute the solution to a real system of linear equations

$$AX = B,$$

where A is an n by n symmetric matrix stored in packed format and X and B are n by r matrices. Error bounds on the solution and a condition estimate are also provided.

2 Specification

```
#include <nag.h>
#include <nagf07.h>
```

```
void nag_dspsvx (Nag_OrderType order, Nag_FactoredFormType fact,
     Nag_UploType uplo, Integer n, Integer nrhs, const double ap[],
     double afp[], Integer ipiv[], const double b[], Integer pdb, double x[],
     Integer pdx, double *rcond, double ferr[], double berr[],
     NagError *fail)
```

3 Description

nag_dspsvx (f07pbc) performs the following steps:

1. If **fact** $=$ Nag_NotFactored, the diagonal pivoting method is used to factor A as $A = UDU^{\mathrm{T}}$ if **uplo** $=$ Nag_Upper or $A = LDL^{\mathrm{T}}$ if **uplo** $=$ Nag_Lower, where U (or L) is a product of permutation and unit upper (lower) triangular matrices and D is symmetric and block diagonal with 1 by 1 and 2 by 2 diagonal blocks.
2. If some $d_{ii} = 0$, so that D is exactly singular, then the function returns with **fail**.**errnum** $= i$ and **fail**.**code** = NE_SINGULAR. Otherwise, the factored form of A is used to estimate the condition number of the matrix A. If the reciprocal of the condition number is less than ***machine precision***, **fail**.**code** = NE_SINGULAR_WP is returned as a warning, but the function still goes on to solve for X and compute error bounds as described below.
3. The system of equations is solved for X using the factored form of A.
4. Iterative refinement is applied to improve the computed solution matrix and to calculate error bounds and backward error estimates for it.

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

Higham N J (2002) *Accuracy and Stability of Numerical Algorithms* (2nd Edition) SIAM, Philadelphia

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 2.3.1.3 in How to Use the NAG Library and its Documentation for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

2: **fact** – Nag_FactoredFormType *Input*

On entry: specifies whether or not the factorized form of the matrix A has been supplied.

fact = Nag_Factored
afp and **ipiv** contain the factorized form of the matrix A. **afp** and **ipiv** will not be modified.

fact = Nag_NotFactored
The matrix A will be copied to **afp** and factorized.

Constraint: **fact** = Nag_Factored or Nag_NotFactored.

3: **uplo** – Nag_UploType *Input*

On entry: if **uplo** = Nag_Upper, the upper triangle of A is stored.

If **uplo** = Nag_Lower, the lower triangle of A is stored.

Constraint: **uplo** = Nag_Upper or Nag_Lower.

4: **n** – Integer *Input*

On entry: n, the number of linear equations, i.e., the order of the matrix A.

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

5: **nrhs** – Integer *Input*

On entry: r, the number of right-hand sides, i.e., the number of columns of the matrix B.

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

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

Note: the dimension, *dim*, of the array **ap** must be at least $\max(1, \mathbf{n} \times (\mathbf{n} + 1)/2)$.

On entry: the n by n symmetric matrix A, packed by rows or columns.

The storage of elements A_{ij} depends on the **order** and **uplo** arguments as follows:

if **order** = Nag_ColMajor and **uplo** = Nag_Upper,
A_{ij} is stored in $\mathbf{ap}[(j-1) \times j/2 + i - 1]$, for $i \leq j$;
if **order** = Nag_ColMajor and **uplo** = Nag_Lower,
A_{ij} is stored in $\mathbf{ap}[(2n-j) \times (j-1)/2 + i - 1]$, for $i \geq j$;
if **order** = Nag_RowMajor and **uplo** = Nag_Upper,
A_{ij} is stored in $\mathbf{ap}[(2n-i) \times (i-1)/2 + j - 1]$, for $i \leq j$;
if **order** = Nag_RowMajor and **uplo** = Nag_Lower,
A_{ij} is stored in $\mathbf{ap}[(i-1) \times i/2 + j - 1]$, for $i \geq j$.

7: **afp**[*dim*] – double *Input/Output*

Note: the dimension, *dim*, of the array **afp** must be at least $\max(1, \mathbf{n} \times (\mathbf{n} + 1)/2)$.

On entry: if **fact** = Nag_Factored, **afp** contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = UDU^{\mathrm{T}}$ or $A = LDL^{\mathrm{T}}$ as computed by nag_dsptrf (f07pdc), stored as a packed triangular matrix in the same storage format as A.

On exit: if **fact** = Nag_NotFactored, **afp** contains the block diagonal matrix D and the multipliers used to obtain the factor U or L from the factorization $A = UDU^{\mathrm{T}}$ or $A = LDL^{\mathrm{T}}$ as computed by nag_dsptrf (f07pdc), stored as a packed triangular matrix in the same storage format as A.

8: **ipiv**[**n**] – Integer *Input/Output*

On entry: if **fact** = Nag_Factored, **ipiv** contains details of the interchanges and the block structure of D, as determined by nag_dsptrf (f07pdc).

if $\mathbf{ipiv}[i-1] = k > 0$, d_{ii} is a 1 by 1 pivot block and the ith row and column of A were interchanged with the kth row and column;

if **uplo** = Nag_Upper and $\mathbf{ipiv}[i-2] = \mathbf{ipiv}[i-1] = -l < 0$, $\begin{pmatrix} d_{i-1,i-1} & \bar{d}_{i,i-1} \\ \bar{d}_{i,i-1} & d_{ii} \end{pmatrix}$ is a 2 by 2 pivot block and the $(i-1)$th row and column of A were interchanged with the lth row and column;

if **uplo** = Nag_Lower and $\mathbf{ipiv}[i-1] = \mathbf{ipiv}[i] = -m < 0$, $\begin{pmatrix} d_{ii} & d_{i+1,i} \\ d_{i+1,i} & d_{i+1,i+1} \end{pmatrix}$ is a 2 by 2 pivot block and the $(i+1)$th row and column of A were interchanged with the mth row and column.

On exit: if **fact** = Nag_NotFactored, **ipiv** contains details of the interchanges and the block structure of D, as determined by nag_dsptrf (f07pdc), as described above.

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

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

$\max(1, \mathbf{pdb} \times \mathbf{nrhs})$ 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

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

On entry: the n by r right-hand side 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, $\mathbf{pdb} \geq \max(1, \mathbf{nrhs})$.

11: **x**[*dim*] – double *Output*

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

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

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

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

On exit: if **fail.code** = NE_NOERROR or NE_SINGULAR_WP, the n by r solution matrix X.

12: **pdx** – Integer *Input*

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

Constraints:

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

13: **rcond** – double * *Output*

On exit: the estimate of the reciprocal condition number of the matrix A. If **rcond** $= 0.0$, the matrix may be exactly singular. This condition is indicated by **fail.code** = NE_SINGULAR. Otherwise, if **rcond** is less than the ***machine precision***, the matrix is singular to working precision. This condition is indicated by **fail.code** = NE_SINGULAR_WP.

14: **ferr**[**nrhs**] – double *Output*

On exit: if **fail.code** = NE_NOERROR or NE_SINGULAR_WP, an estimate of the forward error bound for each computed solution vector, such that $\|\hat{x}_j - x_j\|_\infty / \|x_j\|_\infty \leq \mathbf{ferr}[j-1]$ where $\hat{x}_j$ is the jth column of the computed solution returned in the array **x** and x_j is the corresponding column of the exact solution X. The estimate is as reliable as the estimate for **rcond**, and is almost always a slight overestimate of the true error.

15: **berr**[**nrhs**] – double *Output*

On exit: if **fail.code** = NE_NOERROR or NE_SINGULAR_WP, an estimate of the component-wise relative backward error of each computed solution vector $\hat{x}_j$ (i.e., the smallest relative change in any element of A or B that makes $\hat{x}_j$ an exact solution).

16: **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_BAD_PARAM

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

NE_INT

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

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

On entry, **pdb** $= \langle value \rangle$.
Constraint: **pdb** > 0.

On entry, **pdx** $= \langle value \rangle$.
Constraint: **pdx** > 0.

NE_INT_2

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

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

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

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

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_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_SINGULAR

Element $\langle value \rangle$ of the diagonal is exactly zero. The factorization has been completed, but the factor D is exactly singular, so the solution and error bounds could not be computed. **rcond** $= 0.0$ is returned.

NE_SINGULAR_WP

D is nonsingular, but **rcond** is less than ***machine precision***, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of **rcond** would suggest.

7 Accuracy

For each right-hand side vector b, the computed solution $\hat{x}$ is the exact solution of a perturbed system of equations $(A + E)\hat{x} = b$, where

$$\|E\|_1 = O(\epsilon)\|A\|_1,$$

where ϵ is the ***machine precision***. See Chapter 11 of Higham (2002) for further details.

If $\hat{x}$ is the true solution, then the computed solution x satisfies a forward error bound of the form

$$\frac{\|x - \hat{x}\|_\infty}{\|\hat{x}\|_\infty} \leq w_c \operatorname{cond}(A, \hat{x}, b)$$

where $\operatorname{cond}(A, \hat{x}, b) = \left\| |A^{-1}|(|A||\hat{x}| + |b|) \right\|_\infty / \|\hat{x}\|_\infty \leq \operatorname{cond}(A) = \left\| |A^{-1}||A| \right\|_\infty \leq \kappa_\infty(A)$. If $\hat{x}$ is the jth column of X, then w_c is returned in **berr**$[j-1]$ and a bound on $\|x - \hat{x}\|_\infty / \|\hat{x}\|_\infty$ is returned in **ferr**$[j-1]$. See Section 4.4 of Anderson *et al.* (1999) for further details.

8 Parallelism and Performance

nag_dspsvx (f07pbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dspsvx (f07pbc) 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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

The factorization of A requires approximately $\frac{1}{3}n^3$ floating-point operations.

For each right-hand side, computation of the backward error involves a minimum of $4n^2$ floating-point operations. Each step of iterative refinement involves an additional $6n^2$ operations. At most five steps of iterative refinement are performed, but usually only one or two steps are required. Estimating the forward error involves solving a number of systems of equations of the form $Ax = b$; the number is usually 4 or 5 and never more than 11. Each solution involves approximately $2n^2$ operations.

The complex analogues of this function are nag_zhpsvx (f07ppc) for Hermitian matrices, and nag_zspsvx (f07qpc) for symmetric matrices.

10 Example

This example solves the equations

$$AX = B,$$

where A is the symmetric matrix

$$A = \begin{pmatrix} -1.81 & 2.06 & 0.63 & -1.15 \\ 2.06 & 1.15 & 1.87 & 4.20 \\ 0.63 & 1.87 & -0.21 & 3.87 \\ -1.15 & 4.20 & 3.87 & 2.07 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 0.96 & 3.93 \\ 6.07 & 19.25 \\ 8.38 & 9.90 \\ 9.50 & 27.85 \end{pmatrix}.$$

Error estimates for the solutions, and an estimate of the reciprocal of the condition number of the matrix A are also output.

10.1 Program Text

```
/* nag_dspsvx (f07pbc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf07.h>

int main(void)
{

  /* Scalars */
  double rcond;
  Integer exit_status = 0, i, j, n, nrhs, pdb, pdx;

  /* Arrays */
  double *afp = 0, *ap = 0, *b = 0, *berr = 0, *ferr = 0, *x = 0;
  Integer *ipiv = 0;
  char nag_enum_arg[40];
```

```
  /* Nag Types */
  NagError fail;
  Nag_OrderType order;
  Nag_UploType uplo;

#ifdef NAG_COLUMN_MAJOR
#define A_UPPER(I, J) ap[J*(J-1)/2 + I - 1]
#define A_LOWER(I, J) ap[(2*n-J)*(J-1)/2 + I - 1]
#define B(I, J)       b[(J-1)*pdb + I - 1]
  order = Nag_ColMajor;
#else
#define A_LOWER(I, J) ap[I*(I-1)/2 + J - 1]
#define A_UPPER(I, J) ap[(2*n-I)*(I-1)/2 + J - 1]
#define B(I, J)       b[(I-1)*pdb + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dspsvx (f07pbc) Example Program Results\n\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &n, &nrhs);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &n, &nrhs);
#endif
  if (n < 0 || nrhs < 0) {
    printf("Invalid n or nrhs\n");
    exit_status = 1;
    goto END;
  }
#ifdef _WIN32
  scanf_s(" %39s%*[^\n]", nag_enum_arg, (unsigned)_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
   */
  uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

  /* Allocate memory */
  if (!(afp = NAG_ALLOC(n * (n + 1) / 2, double)) ||
      !(ap = NAG_ALLOC(n * (n + 1) / 2, double)) ||
      !(b = NAG_ALLOC(n * nrhs, double)) ||
      !(berr = NAG_ALLOC(nrhs, double)) ||
      !(ferr = NAG_ALLOC(nrhs, double)) ||
      !(x = NAG_ALLOC(n * nrhs, double)) || !(ipiv = NAG_ALLOC(n, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

#ifdef NAG_COLUMN_MAJOR
  pdb = n;
  pdx = n;
#else
  pdb = nrhs;
  pdx = nrhs;
#endif

  /* Read the triangular part of the matrix A from  data file */
  if (uplo == Nag_Upper)
    for (i = 1; i <= n; ++i)
```

```
#ifdef _WIN32
      for (j = i; j <= n; ++j)
        scanf_s("%lf", &A_UPPER(i, j));
#else
      for (j = i; j <= n; ++j)
        scanf("%lf", &A_UPPER(i, j));
#endif
  else if (uplo == Nag_Lower)
    for (i = 1; i <= n; ++i)
#ifdef _WIN32
      for (j = 1; j <= i; ++j)
        scanf_s("%lf", &A_LOWER(i, j));
#else
      for (j = 1; j <= i; ++j)
        scanf("%lf", &A_LOWER(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Read B from data file */
  for (i = 1; i <= n; ++i)
#ifdef _WIN32
    for (j = 1; j <= nrhs; ++j)
      scanf_s("%lf", &B(i, j));
#else
    for (j = 1; j <= nrhs; ++j)
      scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Solve the equations AX = B for X using nag_dspsvx (f07pbc). */
  nag_dspsvx(order, Nag_NotFactored, uplo, n, nrhs, ap, afp, ipiv, b, pdb,
             x, pdx, &rcond, ferr, berr, &fail);
  if (fail.code != NE_NOERROR && fail.code != NE_SINGULAR) {
    printf("Error from nag_dspsvx (f07pbc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print solution using nag_gen_real_mat_print (x04cac). */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs,
                         x, pdx, "Solution(s)", 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;
  }

  /* Print error bounds and condition number */
  printf("\nBackward errors (machine-dependent)\n");
  for (j = 0; j < nrhs; ++j)
    printf("%11.1e%s", berr[j], j % 7 == 6 ? "\n" : " ");

  printf("\n\nEstimated forward error bounds (machine-dependent)\n");
  for (j = 0; j < nrhs; ++j)
    printf("%11.1e%s", ferr[j], j % 7 == 6 ? "\n" : " ");

  printf("\n\nEstimate of reciprocal condition number\n%11.1e\n", rcond);
  if (fail.code == NE_SINGULAR) {
    printf("Error from nag_dspsvx (f07pbc).\n%s\n", fail.message);
    exit_status = 1;
  }
END:
```

```
  NAG_FREE(afp);
  NAG_FREE(ap);
  NAG_FREE(b);
  NAG_FREE(berr);
  NAG_FREE(ferr);
  NAG_FREE(x);
  NAG_FREE(ipiv);

  return exit_status;
}

#undef A_UPPER
#undef A_LOWER
#undef B
```

10.2 Program Data

```
nag_dspsvx (f07pbc) Example Program Data
  4      2                   : n, nrhs
  Nag_Upper                  : uplo
 -1.81   2.06   0.63  -1.15
         1.15   1.87   4.20
               -0.21   3.87
                       2.07 : matrix A
  0.96   3.93
  6.07  19.25
  8.38   9.90
  9.50  27.85                : matrix B
```

10.3 Program Results

```
nag_dspsvx (f07pbc) Example Program Results

 Solution(s)
            1          2
 1     -5.0000     2.0000
 2     -2.0000     3.0000
 3      1.0000     4.0000
 4      4.0000     1.0000

Backward errors (machine-dependent)
    1.4e-16     1.0e-16

Estimated forward error bounds (machine-dependent)
    2.5e-14     3.2e-14

Estimate of reciprocal condition number
    1.3e-02
```