NAG Library Function Document

nag_dsysvx (f07mbc)

1 Purpose

nag_dsysvx (f07mbc) uses the diagonal pivoting factorization to compute the solution to a real system of linear equations

\[ AX = B, \]

where \( A \) is an \( n \) by \( n \) symmetric matrix and \( X \) and \( B \) are \( n \) by \( r \) matrices. Error bounds on the solution and a condition estimate are also provided.

2 Specification

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

void nag_dsysvx (Nag_OrderType order, Nag_FactoredFormType fact,
                  Nag_UploType uplo, Integer n, Integer nrhs, const double a[],
                  Integer pda, double af[], Integer pdaf, Integer ipiv[],
                  const double b[], Integer pdb, double x[], Integer pdx, double *rcond,
                  double ferr[], double berr[], NagError *fail)
```

3 Description

nag_dsysvx (f07mbc) performs the following steps:

1. If \( \text{fact} = \text{Nag_NotFactored} \), the diagonal pivoting method is used to factor \( A \). The form of the factorization is \( A = U D U^T \) if \( \text{uplo} = \text{Nag_Upper} \) or \( A = L D L^T \) if \( \text{uplo} = \text{Nag_Lower} \), 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 \( \text{fail.errnum} = i \) and \( \text{fail.code} = \text{NE_SINGULAR} \). 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 \text{machine precision}, \text{fail.code} = \text{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


5 Arguments

1: order – Nag_OrderType

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: fact – Nag_FactoredFormType

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

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

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

Constraint: fact = Nag_Factored or Nag_NotFactored.

3: uplo – Nag_UploType

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

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

Constraint: n ≥ 0.

5: nrhs – Integer

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

Constraint: nrhs ≥ 0.

6: a[dim] – const double

Note: the dimension, dim, of the array a must be at least max(1, pda × n).

On entry: the n by n symmetric matrix A.

If order = Nag_ColMajor, Aij is stored in a[(j - 1) × pda + i - 1].

If order = Nag_RowMajor, Aij is stored in a[(i - 1) × pda + j - 1].

If uplo = Nag_Upper, the upper triangular part of A must be stored and the elements of the array below the diagonal are not referenced.

If uplo = Nag_Lower, the lower triangular part of A must be stored and the elements of the array above the diagonal are not referenced.

7: pda – Integer

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

Constraint: pda ≥ max(1, n).

8: af[dim] – double

Note: the dimension, dim, of the array af must be at least max(1, pdaf × n).
The \((i, j)\)th element of the matrix is stored in
\[
\begin{align*}
\text{af}[ (j - 1) \times \text{pdaf} + i - 1 ] & \quad \text{when order = Nag\_ColMajor}; \\
\text{af}[ (i - 1) \times \text{pdaf} + j - 1 ] & \quad \text{when order = Nag\_RowMajor}.
\end{align*}
\]

On entry: if \text{fact} = Nag\_Factored, \text{af} contains the block diagonal matrix \(D\) and the multipliers used to obtain the factor \(U\) or \(L\) from the factorization \(a = UD^T\) or \(a = LDL^T\) as computed by \text{nag\_dsytrf} (f07mdc).

On exit: if \text{fact} = Nag\_NotFactored, \text{af} returns the block diagonal matrix \(D\) and the multipliers used to obtain the factor \(U\) or \(L\) from the factorization \(a = UD^T\) or \(a = LDL^T\).

9: \text{pdaf} – Integer

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

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

10: \text{ipiv}[\text{dim}] – Integer

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

On entry: if \text{fact} = Nag\_Factored, \text{ipiv} contains details of the interchanges and the block structure of \(D\), as determined by \text{nag\_dsytrf} (f07mdc).

if \text{ipiv}[i - 1] = k > 0, \(d_{kk}\) is a 1 by 1 pivot block and the \(i\)th row and column of \(A\) were interchanged with the \(k\)th row and column;

if \text{uplo} = Nag\_Upper and \text{ipiv}[i - 2] = \text{ipiv}[i - 1] = -l < 0, \begin{pmatrix} d_{i-1,i-1} & d_{i,i-1} \\ 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 \(l\)th row and column;

if \text{uplo} = Nag\_Lower and \text{ipiv}[i - 1] = \text{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 \(m\)th row and column.

On exit: if \text{fact} = Nag\_NotFactored, \text{ipiv} contains details of the interchanges and the block structure of \(D\), as determined by \text{nag\_dsytrf} (f07mdc), as described above.

11: \text{b}[\text{dim}] – const double

Note: the dimension, \text{dim}, of the array \text{b} must be at least
\(\max(1, \text{pd} \times \text{nrhs})\) when \text{order} = Nag\_ColMajor;
\(\max(1, n \times \text{pd})\) when \text{order} = Nag\_RowMajor.

The \((i, j)\)th element of the matrix \(B\) is stored in
\[
\begin{align*}
\text{b}[ (j - 1) \times \text{pd} + i - 1 ] & \quad \text{when order = Nag\_ColMajor}; \\
\text{b}[ (i - 1) \times \text{pd} + j - 1 ] & \quad \text{when order = Nag\_RowMajor}.
\end{align*}
\]

On entry: the \(n\) by \(r\) right-hand side matrix \(B\).

12: \text{pd} – Integer

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

Constraints:
\[
\begin{align*}
\text{if order = Nag\_ColMajor, pd} & \geq \max(1, n); \\
\text{if order = Nag\_RowMajor, pd} & \geq \max(1, \text{nrhs}).
\end{align*}
\]
13: \( x[\text{dim}] \) – double  

**Output**

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

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

On exit: if \( \text{fail\.code} = \text{NE\_NOERROR} \) or \( \text{NE\_SINGULAR\_WP} \), the \( n \) by \( r \) solution matrix \( X \).

14: \( \text{pdx} \) – Integer  

**Input**

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

**Constraints:**

- if \( \text{order} = \text{Nag\_ColMajor} \), \( \text{pdx} \geq \max(1, \text{n}) \);  
- if \( \text{order} = \text{Nag\_RowMajor} \), \( \text{pdx} \geq \max(1, \text{nrhs}) \).

15: \( \text{rcond} \) – double *  

**Output**

**On exit:** the estimate of the reciprocal condition number of the matrix \( A \). If \( \text{rcond} = 0.0 \), the matrix may be exactly singular. This condition is indicated by \( \text{fail\.code} = \text{NE\_SINGULAR} \). Otherwise, if \( \text{rcond} \) is less than the machine precision, the matrix is singular to working precision. This condition is indicated by \( \text{fail\.code} = \text{NE\_SINGULAR\_WP} \).

16: \( \text{ferr\[\text{dim}\]} \) – double  

**Output**

**Note:** the dimension, \( \text{dim} \), of the array \( \text{ferr} \) must be at least \( \max(1, \text{nrhs}) \).

**On exit:** if \( \text{fail\.code} = \text{NE\_NOERROR} \) or \( \text{NE\_SINGULAR\_WP} \), an estimate of the forward error bound for each computed solution vector, such that  
\[ \| \hat{x}_j - x_j \| / \| x_j \| \leq \text{ferr}[j - 1] \]  
where \( \hat{x}_j \) is the \( j \)th 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 \( \text{rcond} \), and is almost always a slight overestimate of the true error.

17: \( \text{berr\[\text{dim}\]} \) – double  

**Output**

**Note:** the dimension, \( \text{dim} \), of the array \( \text{berr} \) must be at least \( \max(1, \text{nrhs}) \).

**On exit:** if \( \text{fail\.code} = \text{NE\_NOERROR} \) or \( \text{NE\_SINGULAR\_WP} \), 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).

18: \( \text{fail} \) – 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 \( \langle \text{value} \rangle \) had an illegal value.
**NE_INT**

On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \geq 0 \).

On entry, \( \text{nrhs} = \langle \text{value} \rangle \).
Constraint: \( \text{nrhs} \geq 0 \).

On entry, \( \text{pda} = \langle \text{value} \rangle \).
Constraint: \( \text{pda} > 0 \).

On entry, \( \text{pdaf} = \langle \text{value} \rangle \).
Constraint: \( \text{pdaf} > 0 \).

On entry, \( \text{pdb} = \langle \text{value} \rangle \).
Constraint: \( \text{pdb} > 0 \).

On entry, \( \text{pdx} = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} > 0 \).

**NE_INT_2**

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

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

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

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

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

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

**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 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.

**NE_SINGULAR**

Element \( \langle \text{value} \rangle \) 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. \( \text{rcond} = 0.0 \) is returned.

**NE_SINGULAR_WP**

\( D \) is nonsingular, but \( \text{rcond} \) is less than \textit{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 \( \text{rcond} \) 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 $\epsilon$ 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

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

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

8 Parallelism and Performance

nag_dsysvx (f07mbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dsysvx (f07mbc) 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’ Note for 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_zhesvx (f07mpc) for Hermitian matrices, and nag_zsysvx (f07npc) for symmetric matrices.

10 Example

This example solves the equations

$$AX = B,$$

where $A$ is the symmetric matrix

$$A = \begin{pmatrix}
-1.81 & 2.06 & 0.63 & -1.15 \\
2.06 & 1.15 & 1.87 & 4.20 \\
0.63 & 1.87 & -0.21 & 3.87 \\
-1.15 & 4.20 & 3.87 & 2.07
\end{pmatrix}$$

and $B = \begin{pmatrix}
0.96 & 3.93 \\
6.07 & 19.25 \\
8.38 & 9.90 \\
9.50 & 27.85
\end{pmatrix}$.

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_dsysvx (f07mbc) Example Program. 
 * Copyright 2014 Numerical Algorithms Group. 
 * Mark 23, 2011. 
 */

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

int main(void)
{
    /* Scalars */
    double rcond;
    Integer exit_status = 0, i, j, n, nrhs, pda, pdaf, pdb, pdx;

    /* Arrays */
    double *a = 0, *af = 0, *b = 0, *berr = 0, *ferr = 0;
    double *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(I, J) a[(J-1)*pda +I - 1]
    #define B(I, J) b[(J-1)*pdb +I - 1]
    #endif
    order = Nag_ColMajor;
    #else
    #define A(I, J) a[(I-1)*pda +J - 1]
    #define B(I, J) b[(I-1)*pdb +J - 1]
    #endif
    order = Nag_RowMajor;

    INIT_FAIL(fail);
    printf("nag_dsysvx (f07mbc) Example Program Results\n\n");

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

    /* 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 (!(a = NAG_ALLOC(n * n, double)) ||
    !(af = NAG_ALLOC(n * n, 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;
}

pda = n;
pdaf = n;
#endif 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(i, j));
#else
        for (j = i; j <= n; ++j) scanf("%lf", &A(i, j));
#endif
#else _WIN32
    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("%*[\n"]);
#else
    scanf("%*[\n"]);
#endif

/* Solve the equations AX = B for X using nag_dsysvx (f07mbc). */
if (fail.code != NE_NOERROR && fail.code != NE_SINGULAR)
{
    printf("Error from nag_dsysvx (f07mbc).\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("Backward errors (machine-dependent)\n");
for (j = 0; j < nrhs; ++j) printf("%11.1e%s", berr[j], j%7 == 6?"\n":" ");

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

printf("Estimate of reciprocal condition number\n%11.1e\n", rcond);
if (fail.code == NE_SINGULAR)
{
    printf("Error from nag_dsysvx (f07mbc).\n%s\n", fail.message);
    exit_status = 1;
}

END:
NAG_FREE(a);
NAG_FREE(af);
NAG_FREE(b);
NAG_FREE(berr);
NAG_FREE(ferr);
NAG_FREE(x);
NAG_FREE(ipiv);
return exit_status;

#define B
#define A

10.2 Program Data

nag_dsysvx (f07mbc) Example Program Data
4 2 : n and 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_dsysvx (f07mbc) 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