

## NAG Library Function Document

### nag\_dgtsvx (f07cbc)

#### 1 Purpose

nag\_dgtsvx (f07cbc) uses the  $LU$  factorization to compute the solution to a real system of linear equations

$$AX = B \quad \text{or} \quad A^T X = B,$$

where  $A$  is a tridiagonal matrix of order  $n$  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_dgtsvx (Nag_OrderType order, Nag_FactoredFormType fact,
                Nag_TransType trans, Integer n, Integer nrhs, const double dl[],
                const double d[], const double du[], double dlf[], double df[],
                double duf[], double du2[], Integer ipiv[], const double b[],
                Integer pdb, double x[], Integer pdx, double *rcond, double ferr[],
                double berr[], NagError *fail)
```

#### 3 Description

nag\_dgtsvx (f07cbc) performs the following steps:

1. If **fact** = Nag\_NotFactored, the  $LU$  decomposition is used to factor the matrix  $A$  as  $A = LU$ , where  $L$  is a product of permutation and unit lower bidiagonal matrices and  $U$  is upper triangular with nonzeros in only the main diagonal and first two superdiagonals.
2. If some  $u_{ii} = 0$ , so that  $U$  is exactly singular, then the function returns with **fail.errno** =  $i$ . 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  
**dlf**, **df**, **duf**, **du2** and **ipiv** contain the factorized form of the matrix  $A$ . **dlf**, **df**, **duf**, **du2** and **ipiv** will not be modified.  
**fact** = Nag\_NotFactored  
The matrix  $A$  will be copied to **dlf**, **df** and **duf** and factorized.  
*Constraint:* **fact** = Nag\_Factored or Nag\_NotFactored.
- 3: **trans** – Nag\_TransType *Input*  
*On entry:* specifies the form of the system of equations.  
**trans** = Nag\_NoTrans  
 $AX = B$  (No transpose).  
**trans** = Nag\_Trans or Nag\_ConjTrans  
 $A^T X = B$  (Transpose).  
*Constraint:* **trans** = Nag\_NoTrans, Nag\_Trans or Nag\_ConjTrans.
- 4: **n** – Integer *Input*  
*On entry:*  $n$ , the order of the matrix  $A$ .  
*Constraint:*  $n \geq 0$ .
- 5: **nrhs** – Integer *Input*  
*On entry:*  $r$ , the number of right-hand sides, i.e., the number of columns of the matrix  $B$ .  
*Constraint:* **nrhs**  $\geq 0$ .
- 6: **dl**[ $dim$ ] – const double *Input*  
**Note:** the dimension,  $dim$ , of the array **dl** must be at least  $\max(1, n - 1)$ .  
*On entry:* the  $(n - 1)$  subdiagonal elements of  $A$ .
- 7: **d**[ $dim$ ] – const double *Input*  
**Note:** the dimension,  $dim$ , of the array **d** must be at least  $\max(1, n)$ .  
*On entry:* the  $n$  diagonal elements of  $A$ .
- 8: **du**[ $dim$ ] – const double *Input*  
**Note:** the dimension,  $dim$ , of the array **du** must be at least  $\max(1, n - 1)$ .  
*On entry:* the  $(n - 1)$  superdiagonal elements of  $A$ .

- 9: **dlf**[*dim*] – double *Input/Output*  
**Note:** the dimension, *dim*, of the array **dlf** must be at least  $\max(1, \mathbf{n} - 1)$ .  
*On entry:* if **fact** = Nag\_Factored, **dlf** contains the  $(n - 1)$  multipliers that define the matrix *L* from the *LU* factorization of *A*.  
*On exit:* if **fact** = Nag\_NotFactored, **dlf** contains the  $(n - 1)$  multipliers that define the matrix *L* from the *LU* factorization of *A*.
- 10: **df**[*dim*] – double *Input/Output*  
**Note:** the dimension, *dim*, of the array **df** must be at least  $\max(1, \mathbf{n})$ .  
*On entry:* if **fact** = Nag\_Factored, **df** contains the *n* diagonal elements of the upper triangular matrix *U* from the *LU* factorization of *A*.  
*On exit:* if **fact** = Nag\_NotFactored, **df** contains the *n* diagonal elements of the upper triangular matrix *U* from the *LU* factorization of *A*.
- 11: **duf**[*dim*] – double *Input/Output*  
**Note:** the dimension, *dim*, of the array **duf** must be at least  $\max(1, \mathbf{n} - 1)$ .  
*On entry:* if **fact** = Nag\_Factored, **duf** contains the  $(n - 1)$  elements of the first superdiagonal of *U*.  
*On exit:* if **fact** = Nag\_NotFactored, **duf** contains the  $(n - 1)$  elements of the first superdiagonal of *U*.
- 12: **du2**[*dim*] – double *Input/Output*  
**Note:** the dimension, *dim*, of the array **du2** must be at least  $\max(1, \mathbf{n} - 2)$ .  
*On entry:* if **fact** = Nag\_Factored, **du2** contains the  $(n - 2)$  elements of the second superdiagonal of *U*.  
*On exit:* if **fact** = Nag\_NotFactored, **du2** contains the  $(n - 2)$  elements of the second superdiagonal of *U*.
- 13: **ipiv**[*dim*] – Integer *Input/Output*  
**Note:** the dimension, *dim*, of the array **ipiv** must be at least  $\max(1, \mathbf{n})$ .  
*On entry:* if **fact** = Nag\_Factored, **ipiv** contains the pivot indices from the *LU* factorization of *A*.  
*On exit:* if **fact** = Nag\_NotFactored, **ipiv** contains the pivot indices from the *LU* factorization of *A*; row *i* of the matrix was interchanged with row **ipiv**[*i* - 1]. **ipiv**[*i* - 1] will always be either *i* or *i* + 1; **ipiv**[*i* - 1] = *i* indicates a row interchange was not required.
- 14: **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  
**b**[(*j* - 1) × **pdb** + *i* - 1] when **order** = Nag\_ColMajor;  
**b**[(*i* - 1) × **pdb** + *j* - 1] when **order** = Nag\_RowMajor.  
*On entry:* the *n* by *r* right-hand side matrix *B*.
- 15: **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, **pdb**  $\geq$  max(1, **n**);  
 if **order** = Nag\_RowMajor, **pdb**  $\geq$  max(1, **nrhs**).

16: **x**[*dim*] – double

*Output*

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

max(1, **pdx**  $\times$  **nrhs**) when **order** = Nag\_ColMajor;  
 max(1, **n**  $\times$  **pdx**) when **order** = Nag\_RowMajor.

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

**x**[(*j* – 1)  $\times$  **pdx** + *i* – 1] when **order** = Nag\_ColMajor;  
**x**[(*i* – 1)  $\times$  **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*.

17: **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, **n**);  
 if **order** = Nag\_RowMajor, **pdx**  $\geq$  max(1, **nrhs**).

18: **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.

19: **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 *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 **rcond**, and is almost always a slight overestimate of the true error.

20: **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).

21: **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,  $\mathbf{n} = \langle value \rangle$ .

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

On entry,  $\mathbf{nrhs} = \langle value \rangle$ .

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

On entry,  $\mathbf{pdb} = \langle value \rangle$ .

Constraint:  $\mathbf{pdb} > 0$ .

On entry,  $\mathbf{pdx} = \langle value \rangle$ .

Constraint:  $\mathbf{pdx} > 0$ .

**NE\_INT\_2**

On entry,  $\mathbf{pdb} = \langle value \rangle$  and  $\mathbf{n} = \langle value \rangle$ .

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

On entry,  $\mathbf{pdb} = \langle value \rangle$  and  $\mathbf{nrhs} = \langle value \rangle$ .

Constraint:  $\mathbf{pdb} \geq \max(1, \mathbf{nrhs})$ .

On entry,  $\mathbf{pdx} = \langle value \rangle$  and  $\mathbf{n} = \langle value \rangle$ .

Constraint:  $\mathbf{pdx} \geq \max(1, \mathbf{n})$ .

On entry,  $\mathbf{pdx} = \langle value \rangle$  and  $\mathbf{nrhs} = \langle value \rangle$ .

Constraint:  $\mathbf{pdx} \geq \max(1, \mathbf{nrhs})$ .

**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  $U$  is exactly singular, so the solution and error bounds could not be computed.  $\mathbf{rcond} = 0.0$  is returned.

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

**NE\_SINGULAR\_WP**

$U$  is nonsingular, but  $\mathbf{rcond}$  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  $\mathbf{rcond}$  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

$$|E| \leq c(n)\epsilon|L|U|,$$

$c(n)$  is a modest linear function of  $n$ , and  $\epsilon$  is the *machine precision*. See Section 9.3 of Higham (2002) for further details.

If  $x$  is the true solution, then the computed solution  $\hat{x}$  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  $\text{cond}(A, \hat{x}, b) = \frac{\| |A^{-1}|(|A|\|\hat{x}\| + |b|) \|_\infty}{\|\hat{x}\|_\infty} \leq \text{cond}(A) = \| |A^{-1}| |A| \|_\infty \leq \kappa_\infty(A)$ . If  $\hat{x}$  is the  $j$ th 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\_dgtsvx (f07cbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_dgtsvx (f07cbc) 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 total number of floating-point operations required to solve the equations  $AX = B$  is proportional to  $nr$ .

The condition number estimation typically requires between four and five solves and never more than eleven solves, following the factorization. The solution is then refined, and the errors estimated, using iterative refinement.

In practice the condition number estimator is very reliable, but it can underestimate the true condition number; see Section 15.3 of Higham (2002) for further details.

The complex analogue of this function is nag\_zgtsvx (f07cpc).

## 10 Example

This example solves the equations

$$AX = B,$$

where  $A$  is the tridiagonal matrix

$$A = \begin{pmatrix} 3.0 & 2.1 & 0 & 0 & 0 \\ 3.4 & 2.3 & -1.0 & 0 & 0 \\ 0 & 3.6 & -5.0 & 1.9 & 0 \\ 0 & 0 & 7.0 & -0.9 & 8.0 \\ 0 & 0 & 0 & -6.0 & 7.1 \end{pmatrix}$$

and

$$B = \begin{pmatrix} 2.7 & 6.6 \\ -0.5 & 10.8 \\ 2.6 & -3.2 \\ 0.6 & -11.2 \\ 2.7 & 19.1 \end{pmatrix}.$$

Estimates for the backward errors, forward errors and condition number are also output.

### 10.1 Program Text

```

/* nag_dgtsvx (f07cbc) 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 *b = 0, *berr = 0, *d = 0, *df = 0, *dl = 0, *dlf = 0, *du = 0;
    double *du2 = 0, *duf = 0, *ferr = 0, *x = 0;
    Integer *ipiv = 0;

    /* Nag Types */
    NagError fail;
    Nag_OrderType order;

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

    INIT_FAIL(fail);

    printf("nag_dgtsvx (f07cbc) 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;
    }

```

```

}
/* Allocate memory */
if (!(b = NAG_ALLOC(n * nrhs, double)) ||
    !(berr = NAG_ALLOC(nrhs, double)) ||
    !(d = NAG_ALLOC(n, double)) ||
    !(df = NAG_ALLOC(n, double)) ||
    !(dl = NAG_ALLOC(n - 1, double)) ||
    !(dlf = NAG_ALLOC(n - 1, double)) ||
    !(du = NAG_ALLOC(n - 1, double)) ||
    !(du2 = NAG_ALLOC(n - 2, double)) ||
    !(duf = NAG_ALLOC(n - 1, 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 tridiagonal matrix A from data file */
#ifdef _WIN32
    for (i = 0; i < n - 1; ++i)
        scanf_s("%lf", &du[i]);
#else
    for (i = 0; i < n - 1; ++i)
        scanf("%lf", &du[i]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
#ifdef _WIN32
    for (i = 0; i < n; ++i)
        scanf_s("%lf", &d[i]);
#else
    for (i = 0; i < n; ++i)
        scanf("%lf", &d[i]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
#ifdef _WIN32
    for (i = 0; i < n - 1; ++i)
        scanf_s("%lf", &dl[i]);
#else
    for (i = 0; i < n - 1; ++i)
        scanf("%lf", &dl[i]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
/* Read the right hand matrix B */
    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 using nag_dgtsvx (f07cbc). */
nag_dgtsvx(order, Nag_NotFactored, Nag_NoTrans, n, nrhs, dl, d, du, dlf, df,
           duf, du2, ipiv, b, pdb, x, pdx, &rcond, ferr, berr, &fail);
if (fail.code != NE_NOERROR && fail.code != NE_SINGULAR) {
    printf("Error from nag_dgtsvx (f07cbc).\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 solution, 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_dgtsvx (f07cbc).\n%s\n", fail.message);
END:
NAG_FREE(b);
NAG_FREE(berr);
NAG_FREE(d);
NAG_FREE(df);
NAG_FREE(dl);
NAG_FREE(dlf);
NAG_FREE(du);
NAG_FREE(du2);
NAG_FREE(duf);
NAG_FREE(ferr);
NAG_FREE(x);
NAG_FREE(ipiv);

return exit_status;
}
#endif B

```

## 10.2 Program Data

```

nag_dgtsvx (f07cbc) Example Program Data
  5          2          : n and nrhs
    2.1  -1.0   1.9   8.0
  3.0   2.3  -5.0  -0.9   7.1
  3.4   3.6   7.0  -6.0          : matrix A (super, main, sub)-diags
  2.7   6.6
 -0.5  10.8
  2.6  -3.2
  0.6 -11.2
  2.7  19.1          : matrix B

```

### 10.3 Program Results

nag\_dgtsvx (f07cbc) Example Program Results

Solution(s)

	1	2
1	-4.0000	5.0000
2	7.0000	-4.0000
3	3.0000	-3.0000
4	-4.0000	-2.0000
5	-3.0000	1.0000

Backward errors (machine-dependent)

7.2e-17	5.9e-17
---------	---------

Estimated forward error bounds (machine-dependent)

9.4e-15	1.4e-14
---------	---------

Estimate of reciprocal condition number

1.1e-02
---------

---