

NAG Library Function Document

nag_dtpsv (f16plc)

1 Purpose

nag_dtpsv (f16plc) solves a system of equations given as a real triangular matrix stored in packed form.

2 Specification

```
#include <nag.h>
#include <nagf16.h>

void nag_dtpsv (Nag_OrderType order, Nag_UploType uplo, Nag_TransType trans,
               Nag_DiagType diag, Integer n, double alpha, const double ap[],
               double x[], Integer incx, NagError *fail)
```

3 Description

nag_dtpsv (f16plc) performs one of the matrix-vector operations

$$x \leftarrow \alpha A^{-1}x \quad \text{or} \quad x \leftarrow \alpha A^{-T}x,$$

where A is an n by n real triangular matrix, stored in packed form, x is an n -element real vector and α is a real scalar. A^{-T} denotes A^{-T} or equivalently A^{-T} .

No test for singularity or near-singularity of A is included in this function. Such tests must be performed before calling this function.

4 References

Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001) *Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard* University of Tennessee, Knoxville, Tennessee <http://www.netlib.org/blas/blast-forum/blas-report.pdf>

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: **uplo** – Nag_UploType *Input*

On entry: specifies whether A is upper or lower triangular.

uplo = Nag_Upper
 A is upper triangular.

uplo = Nag_Lower
 A is lower triangular.

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

- 3: **trans** – Nag_TransType *Input*
On entry: specifies the operation to be performed.
trans = Nag_NoTrans
 $x \leftarrow \alpha A^{-1}x.$
trans = Nag_Trans or Nag_ConjTrans
 $x \leftarrow \alpha A^{-T}x.$
Constraint: **trans** = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.
- 4: **diag** – Nag_DiagType *Input*
On entry: specifies whether A has nonunit or unit diagonal elements.
diag = Nag_NonUnitDiag
The diagonal elements are stored explicitly.
diag = Nag_UnitDiag
The diagonal elements are assumed to be 1 and are not referenced.
Constraint: **diag** = Nag_NonUnitDiag or Nag_UnitDiag.
- 5: **n** – Integer *Input*
On entry: n , the order of the matrix A .
Constraint: $n \geq 0$.
- 6: **alpha** – double *Input*
On entry: the scalar α .
- 7: **ap**[*dim*] – const double *Input*
Note: the dimension, *dim*, of the array **ap** must be at least $\max(1, n \times (n + 1)/2)$.
On entry: the n by n triangular 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 **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 **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 **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 **ap**[($i - 1$) \times $i/2 + j - 1$], for $i \geq j$.
If **diag** = 'Nag_UnitDiag', the diagonal elements of AP are assumed to be 1, and are not referenced; the same storage scheme is used whether **diag** = 'Nag_NonUnitDiag' or **diag** = 'Nag_UnitDiag'.
- 8: **x**[*dim*] – double *Input/Output*
Note: the dimension, *dim*, of the array **x** must be at least $\max(1, 1 + (n - 1)|\mathbf{incx}|)$.
On entry: the right-hand side vector b .
On exit: the solution vector x .
- 9: **incx** – Integer *Input*
On entry: the increment in the subscripts of **x** between successive elements of x .
Constraint: **incx** $\neq 0$.

10: **fail** – NagError *

Input/Output

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

6 Error Indicators and Warnings

NE_BAD_PARAM

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

NE_INT

On entry, **incx** = $\langle value \rangle$.

Constraint: **incx** $\neq 0$.

On entry, **n** = $\langle value \rangle$.

Constraint: **n** ≥ 0 .

7 Accuracy

The BLAS standard requires accurate implementations which avoid unnecessary over/underflow (see Section 2.7 of Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001)).

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

Solves real triangular system of linear equations, $Ax = y$, where A is a 4 by 4 real triangular matrix, stored in packed storage format, and is given by

$$A = \begin{pmatrix} 4.30 & & & \\ -3.96 & -4.87 & & \\ 0.40 & 0.31 & -8.02 & \\ -0.27 & 0.07 & -5.95 & 0.12 \end{pmatrix}$$

and

$$y = (-12.90, 16.75, -17.55, -11.04)^T.$$

The vector y is stored in **x** and nag_dtpsv (f16plc).

10.1 Program Text

```

/* nag_dtpsv (f16plc) Example Program.
 *
 * Copyright 2005 Numerical Algorithms Group.
 *
 * Mark 8, 2005.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf16.h>

int main(void)
{

```

```

/* Scalars */
double      alpha;
Integer     ap_len, exit_status, i, incx, j, n, xlen;

/* Arrays */
double      *ap = 0, *x = 0;
char        nag_enum_arg[40];

/* Nag Types */
NagError    fail;
Nag_OrderType order;
Nag_TransType trans;
Nag_UploType uplo;
Nag_DiagType diag;

#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]
    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]
    order = Nag_RowMajor;
#endif

    exit_status = 0;
    INIT_FAIL(fail);

    printf("nag_dtpsv (f16plc) Example Program Results\n\n");

    /* Skip heading in data file */
    scanf("%*[\n] ");

    /* Read the problem dimensions */
    scanf("%ld%*[\n] ", &n);

    /* Read the uplo storage parameter */
    scanf("%39s%*[\n] ", nag_enum_arg);
    /* nag_enum_name_to_value (x04nac).
     * Converts NAG enum member name to value
     */
    uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
    /* Read the transpose parameter */
    scanf("%39s%*[\n] ", nag_enum_arg);
    /* nag_enum_name_to_value (x04nac), see above. */
    trans = (Nag_TransType) nag_enum_name_to_value(nag_enum_arg);
    /* Read the unit-diagonal parameter */
    scanf("%39s%*[\n] ", nag_enum_arg);
    /* nag_enum_name_to_value (x04nac), see above. */
    diag = (Nag_DiagType) nag_enum_name_to_value(nag_enum_arg);

    /* Read scalar parameters */
    scanf("%lf%*[\n] ", &alpha);
    /* Read increment parameter */
    scanf("%ld%*[\n] ", &incx);

    ap_len = n*(n+1)/2;
    xlen = MAX(1, 1 + (n - 1)*ABS(incx));

    if (n > 0)
    {
        /* Allocate memory */
        if (!(ap = NAG_ALLOC(ap_len, double)) ||
            !(x = NAG_ALLOC(xlen, double)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
}

```

```

else
{
    printf("Invalid n\n");
    exit_status = 1;
    return exit_status;
}

/* Input matrix A and vector x*/

if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        if (diag == Nag_NonUnitDiag)
            scanf("%lf", &A_UPPER(i, i));
        for (j = i+1; j <= n; ++j)
            scanf("%lf", &A_UPPER(i, j));
    }
    scanf("%*[\n] ");
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j < i; ++j)
            scanf("%lf", &A_LOWER(i, j));
        if (diag == Nag_NonUnitDiag)
            scanf("%lf", &A_LOWER(i, i));
    }
    scanf("%*[\n] ");
}
for (i = 0; i < xlen; ++i)
    scanf("%lf%*[\n] ", &x[i]);

/* nag_dtpsv (f16plc).
 * Solution of real triangular system of linear equations,
 * using packed storage.
 */
nag_dtpsv(order, uplo, trans, diag, n, alpha, ap, x, incx, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dtpsv.\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print output vector x */
printf("%s\n", " Solution x:");
for (i = 0; i < xlen; ++i)
{
    printf("%11f\n", x[i]);
}

END:
NAG_FREE(ap);
NAG_FREE(x);

return exit_status;
}

```

10.2 Program Data

```

nag_dtpsv (f16plc) Example Program Data
4                               :Value of n
Nag_Lower                       :Storage of A
Nag_NoTrans                     :Transpose A?
Nag_NonUnitDiag                 :Unit diagonal elements?
1.0                             :Value of alpha
1                               :Value of incx
4.30

```

```
-3.96  -4.87
  0.40   0.31  -8.02
-0.27   0.07  -5.95   0.12   :End of matrix A
-12.90
  16.75
-17.55
-11.04                               :End of vector x
```

10.3 Program Results

nag_dtpsv (f16plc) Example Program Results

```
Solution x:
-3.000000
-1.000000
 2.000000
 1.000000
```
