NAG Library Function Document

nag_ztpsv (f16slc)

1 Purpose

nag_ztpsv (f16slc) solves a system of equations given as a complex triangular matrix stored in packed form.

2 Specification

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

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

3 Description

nag_ztpsv (f16slc) performs one of the matrix-vector operations

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

where A is an n by n complex triangular matrix, stored in packed form, x is an n-element complex vector and α is a complex scalar. $A^{-\mathrm{T}}$ denotes $A^{-\mathrm{T}}$ or equivalently $A^{-\mathrm{T}}$; $A^{-\mathrm{H}}$ denotes $\left(A^{\mathrm{H}}\right)^{-1}$ or equivalently $\left(A^{-1}\right)^{H}$.

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
$x \leftarrow \alpha A^{-\mathrm{T}} x$.

trans = Nag_ConjTrans
$x \leftarrow \alpha A^{-\mathrm{H}} x$.

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

6: **alpha** – Complex *Input*

On entry: the scalar α.

7: **ap**[*dim*] – const Complex *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 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 $\mathbf{ap}[(j-1) \times j/2 + i - 1]$, 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 $i \geq j$.

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*] – Complex *Input/Output*

Note: the dimension, *dim*, of the array **x** must be at least $\max(1, 1 + (\mathbf{n}-1)|\mathbf{incx}|)$.

On entry: the vector x.

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_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, **incx** = ⟨*value*⟩.
Constraint: **incx** $\neq 0$.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

NE_INTERNAL_ERROR

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

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

$$A = \begin{pmatrix} 4.78 + 4.56i & & & \\ 2.00 - 0.30i & -4.11 + 1.25i & & \\ 2.89 - 1.34i & 2.36 - 4.25i & 4.15 + 0.80i & \\ -1.89 + 1.15i & 0.04 - 3.69i & -0.02 + 0.46i & 0.33 - 0.26i \end{pmatrix},$$

and

$$y = \begin{pmatrix} -14.78 - 32.36i \\ 2.98 - \ \ 2.14i \\ -20.96 + 17.06i \\ 9.54 + \ \ 9.91i \end{pmatrix}.$$

10.1 Program Text

```
/* nag_ztpsv (f16slc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 8, 2005.
 */

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

int main(void)
{

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

  /* Arrays */
  Complex       *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_ztpsv (f16slc) Example Program Results\n\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Read the problem dimensions */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n] ", &n);
#else
  scanf("%"NAG_IFMT"%*[^\n] ", &n);
#endif

  /* Read the uplo storage parameter */
```

```
#ifdef _WIN32
  scanf_s("%39s%*[^\n] ", nag_enum_arg, _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);
  /* Read the transpose parameter */
#ifdef _WIN32
  scanf_s("%39s%*[^\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n] ", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac), see above. */
  trans = (Nag_TransType) nag_enum_name_to_value(nag_enum_arg);
  /* Read the unit-diagonal parameter */
#ifdef _WIN32
  scanf_s("%39s%*[^\n] ", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n] ", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac), see above. */
  diag = (Nag_DiagType) nag_enum_name_to_value(nag_enum_arg);

  /* Read scalar parameters */
#ifdef _WIN32
  scanf_s(" ( %lf , %lf )%*[^\n] ", &alpha.re, &alpha.im);
#else
  scanf(" ( %lf , %lf )%*[^\n] ", &alpha.re, &alpha.im);
#endif
  /* Read increment parameter */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n] ", &incx);
#else
  scanf("%"NAG_IFMT"%*[^\n] ", &incx);
#endif

  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, Complex)) ||
          !(x = NAG_ALLOC(xlen, Complex)))
        {
          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)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &A_UPPER(i, i).re,
                    &A_UPPER(i, i).im);
#else
```

```
            scanf(" ( %lf , %lf )", &A_UPPER(i, i).re,
                   &A_UPPER(i, i).im);
#endif
          for (j = i+1; j <= n; ++j)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &A_UPPER(i, j).re,
                   &A_UPPER(i, j).im);
#else
            scanf(" ( %lf , %lf )", &A_UPPER(i, j).re,
                   &A_UPPER(i, j).im);
#endif
        }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j < i; ++j)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &A_LOWER(i, j).re,
                   &A_LOWER(i, j).im);
#else
            scanf(" ( %lf , %lf )", &A_LOWER(i, j).re,
                   &A_LOWER(i, j).im);
#endif
          if (diag == Nag_NonUnitDiag)
#ifdef _WIN32
            scanf_s(" ( %lf , %lf )", &A_LOWER(i, i).re,
                   &A_LOWER(i, i).im);
#else
            scanf(" ( %lf , %lf )", &A_LOWER(i, i).re,
                   &A_LOWER(i, i).im);
#endif
        }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
    }
  for (i = 0; i < xlen; ++i)
#ifdef _WIN32
    scanf_s(" ( %lf , %lf )%*[^\n] ", &x[i].re, &x[i].im);
#else
    scanf(" ( %lf , %lf )%*[^\n] ", &x[i].re, &x[i].im);
#endif

  /* nag_ztpsv (f16slc).
   * Solution of complex triangular system of linear equations,
   * using packed storage.
   */
  nag_ztpsv(order, uplo, trans, diag, n, alpha, ap, x, incx, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_ztpsv.\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 , %11f )\n", x[i].re, x[i].im);
    }
```

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

  return exit_status;
}
```

10.2 Program Data

```
nag_ztpsv (f16slc) Example Program Data
  4                            :Value of n
  Nag_Lower                    :Storage of A
  Nag_NoTrans                  :Transpose A?
  Nag_NonUnitDiag              :Unit diagonal elements?
  ( 1.0, 0.0)                  :Value of alpha
  1                            :Value of incx
 ( 4.78, 4.56)
 ( 2.00,-0.30) (-4.11, 1.25)
 ( 2.89,-1.34) ( 2.36,-4.25) ( 4.15, 0.80)
 (-1.89, 1.15) ( 0.04,-3.69) (-0.02, 0.46) ( 0.33,-0.26)  :End of matrix A
 (-14.78,-32.36)
 (  2.98, -2.14)
 (-20.96, 17.06)
 (  9.54,  9.91)                                          :End of vector x
```

10.3 Program Results

```
nag_ztpsv (f16slc) Example Program Results

 Solution x:
(   -5.000000 ,   -2.000000 )
(   -3.000000 ,   -1.000000 )
(    2.000000 ,    1.000000 )
(    4.000000 ,    3.000000 )
```