NAG Library Function Document

nag_ztrsyl (f08qvc)

1 Purpose

nag_ztrsyl (f08qvc) solves the complex triangular Sylvester matrix equation.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_ztrsyl (Nag_OrderType order, Nag_TransType trana,
     Nag_TransType tranb, Nag_SignType sign, Integer m, Integer n,
     const Complex a[], Integer pda, const Complex b[], Integer pdb,
     Complex c[], Integer pdc, double *scal, NagError *fail)
```

3 Description

nag_ztrsyl (f08qvc) solves the complex Sylvester matrix equation

$$\mathrm{op}(A)X \pm X\,\mathrm{op}(B) = \alpha C,$$

where $\mathrm{op}(A) = A$ or A^{H}, and the matrices A and B are upper triangular; α is a scale factor (≤ 1) determined by the function to avoid overflow in X; A is m by m and B is n by n while the right-hand side matrix C and the solution matrix X are both m by n. The matrix X is obtained by a straightforward process of back-substitution (see Golub and Van Loan (1996)).

Note that the equation has a unique solution if and only if $\alpha_i \pm \beta_j \neq 0$, where $\{\alpha_i\}$ and $\{\beta_j\}$ are the eigenvalues of A and B respectively and the sign ($+$ or $-$) is the same as that used in the equation to be solved.

4 References

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

Higham N J (1992) Perturbation theory and backward error for $AX - XB = C$ *Numerical Analysis Report* University of Manchester

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: **trana** – Nag_TransType *Input*

On entry: specifies the option $\mathrm{op}(A)$.

trana = Nag_NoTrans
$\mathrm{op}(A) = A$.

trana = Nag_ConjTrans
$\mathrm{op}(A) = A^{\mathrm{H}}$.

Constraint: **trana** = Nag_NoTrans or Nag_ConjTrans.

3: **tranb** – Nag_TransType *Input*

On entry: specifies the option $\mathrm{op}(B)$.

tranb = Nag_NoTrans
$\mathrm{op}(B) = B$.

tranb = Nag_ConjTrans
$\mathrm{op}(B) = B^{\mathrm{H}}$.

Constraint: **tranb** = Nag_NoTrans or Nag_ConjTrans.

4: **sign** – Nag_SignType *Input*

On entry: indicates the form of the Sylvester equation.

sign = Nag_Plus
The equation is of the form $\mathrm{op}(A)X + X\,\mathrm{op}(B) = \alpha C$.

sign = Nag_Minus
The equation is of the form $\mathrm{op}(A)X - X\,\mathrm{op}(B) = \alpha C$.

Constraint: **sign** = Nag_Plus or Nag_Minus.

5: **m** – Integer *Input*

On entry: m, the order of the matrix A, and the number of rows in the matrices X and C.

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

6: **n** – Integer *Input*

On entry: n, the order of the matrix B, and the number of columns in the matrices X and C.

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

7: **a**$[dim]$ – const Complex *Input*

Note: the dimension, *dim*, of the array **a** must be at least $\max(1, \mathbf{pda} \times \mathbf{m})$.

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

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

On entry: the m by m upper triangular matrix A.

8: **pda** – Integer *Input*

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

Constraint: $\mathbf{pda} \geq \max(1, \mathbf{m})$.

9: **b**$[dim]$ – const Complex *Input*

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

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

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

On entry: the n by n upper triangular 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**.

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

11: **c**[dim] – Complex *Input/Output*

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

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

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

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

On entry: the m by n right-hand side matrix C.

On exit: **c** is overwritten by the solution matrix X.

12: **pdc** – Integer *Input*

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

Constraints:

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

13: **scal** – double * *Output*

On exit: the value of the scale factor α.

14: **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.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

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

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

On entry, **pda** = ⟨*value*⟩.
Constraint: **pda** > 0.

On entry, **pdb** = ⟨*value*⟩.
Constraint: **pdb** > 0.

On entry, **pdc** = ⟨*value*⟩.
Constraint: **pdc** > 0.

NE_INT_2

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

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

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

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

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.

NE_PERTURBED

A and B have common or close eigenvalues, perturbed values of which were used to solve the equation.

7 Accuracy

Consider the equation $AX - XB = C$. (To apply the remarks to the equation $AX + XB = C$, simply replace B by $-B$.)

Let $\tilde{X}$ be the computed solution and R the residual matrix:

$$R = C - \left(A\tilde{X} - \tilde{X}B\right).$$

Then the residual is always small:

$$\|R\|_F = O(\epsilon)\left(\|A\|_F + \|B\|_F\right)\left\|\tilde{X}\right\|_F.$$

However, $\tilde{X}$ is **not** necessarily the exact solution of a slightly perturbed equation; in other words, the solution is not backwards stable.

For the forward error, the following bound holds:

$$\left\|\tilde{X} - X\right\|_F \leq \frac{\|R\|_F}{sep(A, B)}$$

but this may be a considerable over estimate. See Golub and Van Loan (1996) for a definition of $sep(A, B)$, and Higham (1992) for further details.

These remarks also apply to the solution of a general Sylvester equation, as described in Section 9.

8 Parallelism and Performance

nag_ztrsyl (f08qvc) is not threaded by NAG in any implementation.

nag_ztrsyl (f08qvc) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The total number of real floating-point operations is approximately $4mn(m + n)$.

To solve the **general** complex Sylvester equation

$$AX \pm XB = C$$

where A and B are general matrices, A and B must first be reduced to Schur form :

$$A = Q_1 \tilde{A} Q_1^{\mathrm{H}} \quad \text{and} \quad B = Q_2 \tilde{B} Q_2^{\mathrm{H}}$$

where $\tilde{A}$ and $\tilde{B}$ are upper triangular and Q_1 and Q_2 are unitary. The original equation may then be transformed to:

$$\tilde{A}\tilde{X} \pm \tilde{X}\tilde{B} = \tilde{C}$$

where $\tilde{X} = Q_1^{\mathrm{H}} X Q_2$ and $\tilde{C} = Q_1^{\mathrm{H}} C Q_2$. $\tilde{C}$ may be computed by matrix multiplication; nag_ztrsyl (f08qvc) may be used to solve the transformed equation; and the solution to the original equation can be obtained as $X = Q_1 \tilde{X} Q_2^{\mathrm{H}}$.

The real analogue of this function is nag_dtrsyl (f08qhc).

10 Example

This example solves the Sylvester equation $AX + XB = C$, where

$$A = \begin{pmatrix} -6.00 - 7.00i & 0.36 - 0.36i & -0.19 + 0.48i & 0.88 - 0.25i \\ 0.00 + 0.00i & -5.00 + 2.00i & -0.03 - 0.72i & -0.23 + 0.13i \\ 0.00 + 0.00i & 0.00 + 0.00i & 8.00 - 1.00i & 0.94 + 0.53i \\ 0.00 + 0.00i & 0.00 + 0.00i & 0.00 + 0.00i & 3.00 - 4.00i \end{pmatrix},$$

$$B = \begin{pmatrix} 0.50 - 0.20i & -0.29 - 0.16i & -0.37 + 0.84i & -0.55 + 0.73i \\ 0.00 + 0.00i & -0.40 + 0.90i & 0.06 + 0.22i & -0.43 + 0.17i \\ 0.00 + 0.00i & 0.00 + 0.00i & -0.90 - 0.10i & -0.89 - 0.42i \\ 0.00 + 0.00i & 0.00 + 0.00i & 0.00 + 0.00i & 0.30 - 0.70i \end{pmatrix}$$

and

$$C = \begin{pmatrix} 0.63 + 0.35i & 0.45 - 0.56i & 0.08 - 0.14i & -0.17 - 0.23i \\ -0.17 + 0.09i & -0.07 - 0.31i & 0.27 - 0.54i & 0.35 + 1.21i \\ -0.93 - 0.44i & -0.33 - 0.35i & 0.41 - 0.03i & 0.57 + 0.84i \\ 0.54 + 0.25i & -0.62 - 0.05i & -0.52 - 0.13i & 0.11 - 0.08i \end{pmatrix}.$$

10.1 Program Text

```
/* nag_ztrsyl (f08qvc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx04.h>
#include <nagx02.h>

int main(void)
{
  /* Scalars */
  Complex       alpha, beta;
  Integer       i, j, m, n, pda, pdb, pdc, pdd, pde, pdf;
  Integer       exit_status = 0;
  double        norm, scale;
  NagError      fail;
```

```
  Nag_OrderType order;
  Nag_SignType  sign = Nag_Minus;
  /* Arrays */
  Complex       *a = 0, *b = 0, *c = 0, *d = 0, *e = 0, *f = 0;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda + I - 1]
#define B(I, J) b[(J-1)*pdb + I - 1]
#define C(I, J) c[(J-1)*pdc + I - 1]
#define D(I, J) c[(J-1)*pdd + I - 1]
#define E(I, J) c[(J-1)*pde + I - 1]
#define F(I, J) c[(J-1)*pdf + I - 1]
  order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + J - 1]
#define B(I, J) b[(I-1)*pdb + J - 1]
#define C(I, J) c[(I-1)*pdc + J - 1]
#define D(I, J) d[(I-1)*pdd + J - 1]
#define E(I, J) e[(I-1)*pde + J - 1]
#define F(I, J) f[(I-1)*pdf + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_ztrsyl (f08qvc) Example Program Results\n\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  scanf("%ld%ld%*[^\n] ", &m, &n);
#ifdef NAG_COLUMN_MAJOR
  pda = m;
  pdb = n;
  pdc = m;
  pdd = m;
  pde = m;
  pdf = m;
#else
  pda = m;
  pdb = n;
  pdc = n;
  pdd = n;
  pde = n;
  pdf = n;
#endif

  /* Allocate memory */
  if (!(a = NAG_ALLOC(m * m, Complex)) ||
      !(b = NAG_ALLOC(n * m, Complex)) ||
      !(c = NAG_ALLOC(m * n, Complex)) ||
      !(d = NAG_ALLOC(m * n, Complex)) ||
      !(e = NAG_ALLOC(m * n, Complex)) ||
      !(f = NAG_ALLOC(m * n, Complex))
      )
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read A, B and C from data file */
  for (i = 1; i <= m; ++i)
    {
      for (j = 1; j <= m; ++j)
        scanf(" ( %lf , %lf ) ", &A(i, j).re, &A(i, j).im);
    }
  scanf("%*[^\n] ");
  for (i = 1; i <= n; ++i)
    {
      for (j = 1; j <= n; ++j)
        scanf(" ( %lf , %lf ) ", &B(i, j).re, &B(i, j).im);
```

```
  }
scanf("%*[^\n] ");
for (i = 1; i <= m; ++i)
  {
    for (j = 1; j <= n; ++j)
      scanf(" ( %lf , %lf ) ", &C(i, j).re, &C(i, j).im);
  }
scanf("%*[^\n] ");

/* Copy C into F */
for(i = 1; i <= m; i++)
  {
    for(j = 1; j <= m; j++)
      {
        F(i, j).re = C(i, j).re;
        F(i, j).im = C(i, j).im;
      }
  }

/* nag_gen_complx_mat_print_comp (x04dbc): Print matrix C */
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m,
                              n, c, pdc, Nag_BracketForm, "%7.4f",
                              "Matrix C", Nag_IntegerLabels, 0,
                              Nag_IntegerLabels, 0, 80, 0, 0, &fail);
printf("\n");
if (fail.code != NE_NOERROR)
  {
    printf(
           "Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

/* Reorder the Schur factorization T */
/* nag_ztrsyl (f08qvc).
 * Solve complex Sylvester matrix equation AX + XB = C, A
 * and B are upper triangular or conjugate-transposes
 */
nag_ztrsyl(order, Nag_NoTrans, Nag_NoTrans, sign, m, n, a, pda,
           b, pdb, c, pdc, &scale, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_ztrsyl (f08qvc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* nag_zgemm (f16zac): Compute aC - (A*X + X*B*sign) from the solution */
/* and store in matrix E*/
alpha.re = 1.0;
alpha.im = 0.0;
beta.re = 0.0;
beta.im = 0.0;
nag_zgemm(order, Nag_NoTrans, Nag_NoTrans, m, n, m, alpha, a, pda,
          c, pdc, beta, d, pdd, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_zgemm (f16zac).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }
if(sign == Nag_Minus)
  alpha.re = -1.0;
else
  alpha.re = 1.0;
beta.re = 1.0;
nag_zgemm(order, Nag_NoTrans, Nag_NoTrans, m, n, n, alpha, c, pdc,
          b, pdb, beta, d, pdd, &fail);
```

```
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zgemm (f16zac).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
  for(i = 1; i <= m; i++)
    {
      for(j = 1; j <= n; j++)
        {
          E(i, j).re = scale * F(i, j).re - D(i, j).re;
          E(i, j).im = scale * F(i, j).im - D(i, j).im;
        }
    }

  /* nag_zge_norm (f16uac): Find norm of matrix E and print warning if */
  /* it is too large */
  nag_zge_norm(order, Nag_OneNorm, m, n, e, pde, &norm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dge_norm (f16rac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  if (norm > pow(x02ajc(),0.8))
    {
      printf("%s\n%s\n","Norm of aC - (A*X + X*B*sign) is much greater than 0.",
             "Schur factorization has failed.");
    }
  else
    {
      printf(" SCALE = %11.2e\n", scale);
     }

 END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(c);
  NAG_FREE(d);
  NAG_FREE(e);
  NAG_FREE(f);

  return exit_status;
}
```

10.2 Program Data

```
nag_ztrsyl (f08qvc) Example Program Data
  4  4                                                        :Values of M and N
 (-6.00,-7.00) ( 0.36,-0.36) (-0.19, 0.48) ( 0.88,-0.25)
 ( 0.00, 0.00) (-5.00, 2.00) (-0.03,-0.72) (-0.23, 0.13)
 ( 0.00, 0.00) ( 0.00, 0.00) ( 8.00,-1.00) ( 0.94, 0.53)
 ( 0.00, 0.00) ( 0.00, 0.00) ( 0.00, 0.00) ( 3.00,-4.00)   :End of matrix A
 ( 0.50,-0.20) (-0.29,-0.16) (-0.37, 0.84) (-0.55, 0.73)
 ( 0.00, 0.00) (-0.40, 0.90) ( 0.06, 0.22) (-0.43, 0.17)
 ( 0.00, 0.00) ( 0.00, 0.00) (-0.90,-0.10) (-0.89,-0.42)
 ( 0.00, 0.00) ( 0.00, 0.00) ( 0.00, 0.00) ( 0.30,-0.70)   :End of matrix B
 ( 0.63, 0.35) ( 0.45,-0.56) ( 0.08,-0.14) (-0.17,-0.23)
 (-0.17, 0.09) (-0.07,-0.31) ( 0.27,-0.54) ( 0.35, 1.21)
 (-0.93,-0.44) (-0.33,-0.35) ( 0.41,-0.03) ( 0.57, 0.84)
 ( 0.54, 0.25) (-0.62,-0.05) (-0.52,-0.13) ( 0.11,-0.08)   :End of matrix C
```

10.3 Program Results

```
nag_ztrsyl (f08qvc) Example Program Results

 Matrix C
                     1                   2                   3                   4
 1  ( 0.6300, 0.3500)  ( 0.4500,-0.5600)  ( 0.0800,-0.1400)  (-0.1700,-0.2300)
 2  (-0.1700, 0.0900)  (-0.0700,-0.3100)  ( 0.2700,-0.5400)  ( 0.3500, 1.2100)
 3  (-0.9300,-0.4400)  (-0.3300,-0.3500)  ( 0.4100,-0.0300)  ( 0.5700, 0.8400)
 4  ( 0.5400, 0.2500)  (-0.6200,-0.0500)  (-0.5200,-0.1300)  ( 0.1100,-0.0800)

 SCALE =    1.00e+00
```