NAG Library Function Document

nag_ztgsyl (f08yvc)

1 Purpose

nag_ztgsyl (f08yvc) solves the generalized complex triangular Sylvester equations.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_ztgsyl (Nag_OrderType order, Nag_TransType trans, Integer ijob,
     Integer m, Integer n, const Complex a[], Integer pda, const Complex b[],
     Integer pdb, Complex c[], Integer pdc, const Complex d[], Integer pdd,
     const Complex e[], Integer pde, Complex f[], Integer pdf, double *scale,
     double *dif, NagError *fail)
```

3 Description

nag_ztgsyl (f08yvc) solves either the generalized complex Sylvester equations

$$\begin{aligned} AR - LB &= \alpha C \\ DR - LE &= \alpha F, \end{aligned} \tag{1}$$

or the equations

$$\begin{aligned} A^{\mathrm{H}}R + D^{\mathrm{H}}L &= \alpha C \\ RB^{\mathrm{H}} + LE^{\mathrm{H}} &= -\alpha F, \end{aligned} \tag{2}$$

where the pair (A, D) are given m by m matrices in generalized Schur form, (B, E) are given n by n matrices in generalized Schur form and (C, F) are given m by n matrices. The pair (R, L) are the m by n solution matrices, and α is an output scaling factor determined by the function to avoid overflow in computing (R, L).

Equations (1) are equivalent to equations of the form

$$Zx = \alpha b,$$

where

$$Z = \begin{pmatrix} I \otimes A - B^{\mathrm{H}} \otimes I \\ I \otimes D - E^{\mathrm{H}} \otimes I \end{pmatrix}$$

and $\otimes$ is the Kronecker product. Equations (2) are then equivalent to

$$Z^{\mathrm{H}}y = \alpha b.$$

The pair (S, T) are in generalized Schur form if S and T are upper triangular as returned, for example, by nag_zgges (f08xnc), or nag_zhgeqz (f08xsc) with **job** = Nag_Schur.

Optionally, the function estimates $\mathrm{Dif}[(A, D), (B, E)]$, the separation between the matrix pairs (A, D) and (B, E), which is the smallest singular value of Z. The estimate can be based on either the Frobenius norm, or the 1-norm. The 1-norm estimate can be three to ten times more expensive than the Frobenius norm estimate, but makes the condition estimation uniform with the nonsymmetric eigenproblem. The Frobenius norm estimate provides a low cost, but equally reliable estimate. For more information see Sections 2.4.8.3 and 4.11.1.3 of Anderson *et al.* (1999) and Kågström and Poromaa (1996).

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

Kågström B (1994) A perturbation analysis of the generalized Sylvester equation $(AR - LB, DR - LE) = (c, F)$ *SIAM J. Matrix Anal. Appl.* **15** 1045–1060

Kågström B and Poromaa P (1996) LAPACK-style algorithms and software for solving the generalized Sylvester equation and estimating the separation between regular matrix pairs *ACM Trans. Math. Software* **22** 78–103

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: **trans** – Nag_TransType *Input*

On entry: if **trans** = Nag_NoTrans, solve the generalized Sylvester equation (1).

If **trans** = Nag_ConjTrans, solve the 'conjugate transposed' system (2).

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

3: **ijob** – Integer *Input*

On entry: specifies what kind of functionality is to be performed when **trans** = Nag_NoTrans.

ijob = 0
Solve (1) only.

ijob = 1
The functionality of **ijob** = 0 and 3.

ijob = 2
The functionality of **ijob** = 0 and 4.

ijob = 3
Only an estimate of $\mathrm{Dif}[(A, D), (B, E)]$ is computed based on the Frobenius norm.

ijob = 4
Only an estimate of $\mathrm{Dif}[(A, D), (B, E)]$ is computed based on the 1-norm.

If **trans** = Nag_ConjTrans, **ijob** is not referenced.

Constraint: if **trans** = Nag_NoTrans, $0 \leq$ **ijob** ≤ 4.

4: **m** – Integer *Input*

On entry: m, the order of the matrices A and D, and the row dimension of the matrices C, F, R and L.

Constraint: **m** ≥ 0.

5: **n** – Integer *Input*

On entry: n, the order of the matrices B and E, and the column dimension of the matrices C, F, R and L.

Constraint: **n** ≥ 0.

6: **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

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

On entry: the upper triangular matrix A.

7: **pda** – Integer *Input*

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

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

8: **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

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

On entry: the upper triangular matrix B.

9: **pdb** – Integer *Input*

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

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

10: **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

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

On entry: contains the right-hand-side matrix C.

On exit: if **ijob** = 0, 1 or 2, **c** is overwritten by the solution matrix R.

If **trans** = Nag_NoTrans and **ijob** = 3 or 4, **c** holds R, the solution achieved during the computation of the Dif estimate.

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

12: **d**[*dim*] – const Complex *Input*

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

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

d$[(j-1) \times$ **pdd** $+ i - 1]$ when **order** = Nag_ColMajor;
d$[(i-1) \times$ **pdd** $+ j - 1]$ when **order** = Nag_RowMajor.

On entry: the upper triangular matrix D.

13: **pdd** – Integer *Input*

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

Constraint: **pdd** $\geq \max(1,$ **m**$)$.

14: **e**$[dim]$ – const Complex *Input*

Note: the dimension, *dim*, of the array **e** must be at least $\max(1,$ **pde** $\times$ **n**$)$.

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

e$[(j-1) \times$ **pde** $+ i - 1]$ when **order** = Nag_ColMajor;
e$[(i-1) \times$ **pde** $+ j - 1]$ when **order** = Nag_RowMajor.

On entry: the upper triangular matrix E.

15: **pde** – Integer *Input*

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

Constraint: **pde** $\geq \max(1,$ **n**$)$.

16: **f**$[dim]$ – Complex *Input/Output*

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

$\max(1,$ **pdf** $\times$ **n**$)$ when **order** = Nag_ColMajor;
$\max(1,$ **m** $\times$ **pdf**$)$ when **order** = Nag_RowMajor.

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

f$[(j-1) \times$ **pdf** $+ i - 1]$ when **order** = Nag_ColMajor;
f$[(i-1) \times$ **pdf** $+ j - 1]$ when **order** = Nag_RowMajor.

On entry: contains the right-hand side matrix F.

On exit: if **ijob** = 0, 1 or 2, **f** is overwritten by the solution matrix L.

If **trans** = Nag_NoTrans and **ijob** = 3 or 4, **f** holds L, the solution achieved during the computation of the Dif estimate.

17: **pdf** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor, **pdf** $\geq \max(1,$ **m**$)$;
if **order** = Nag_RowMajor, **pdf** $\geq \max(1,$ **n**$)$.

18: **scale** – double * *Output*

On exit: α, the scaling factor in (1) or (2).

If $0 <$ **scale** < 1, **c** and **f** hold the solutions R and L, respectively, to a slightly perturbed system but the input arrays **a**, **b**, **d** and **e** have not been changed.

If **scale** $= 0$, **c** and **f** hold the solutions R and L, respectively, to the homogeneous system with $C = F = 0$. In this case **dif** is not referenced.

Normally, **scale** $= 1$.

19: **dif** – double * *Output*

On exit: the estimate of Dif. If **ijob** $= 0$, **dif** is not referenced.

20: **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_EIGENVALUES

(A, D) and (B, E) have common or close eigenvalues and so no solution could be computed.

NE_ENUM_INT

On entry, **trans** $=$ ⟨*value*⟩ and **ijob** $=$ ⟨*value*⟩.
Constraint: if **trans** $=$ Nag_NoTrans, $0 \leq$ **ijob** ≤ 4.

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.

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

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

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

NE_INT_2

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

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

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

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

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

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

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

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

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.

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

See Kågström (1994) for a perturbation analysis of the generalized Sylvester equation.

8 Parallelism and Performance

nag_ztgsyl (f08yvc) is not threaded by NAG in any implementation.

nag_ztgsyl (f08yvc) 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 needed to solve the generalized Sylvester equations is approximately $8mn(n+m)$. The Frobenius norm estimate of Dif does not require additional significant computation, but the 1-norm estimate is typically five times more expensive.

The real analogue of this function is nag_dtgsyl (f08yhc).

10 Example

This example solves the generalized Sylvester equations

$$\begin{aligned} AR - LB &= \alpha C \\ DR - LE &= \alpha F, \end{aligned}$$

where

$$A = \begin{pmatrix} 4.0+4.0i & 1.0+1.0i & 1.0+1.0i & 2.0-1.0i \\ 0 & 2.0+1.0i & 1.0+1.0i & 1.0+1.0i \\ 0 & 0 & 2.0-1.0i & 1.0+1.0i \\ 0 & 0 & 0 & 6.0-2.0i \end{pmatrix},$$

$$B = \begin{pmatrix} 2.0 & 1.0+1.0i & 1.0+1.0i & 3.0-1.0i \\ 0 & 1.0 & 2.0+1.0i & 1.0+1.0i \\ 0 & 0 & 1.0 & 1.0+1.0i \\ 0 & 0 & 0 & 2.0 \end{pmatrix},$$

$$D = \begin{pmatrix} 1.0+1.0i & 1.0-1.0i & 1.0+1.0i & 1.0-1.0i \\ 0 & 6.0-4.0i & 1.0-1.0i & 1.0+1.0i \\ 0 & 0 & 2.0+4.0i & 1.0-1.0i \\ 0 & 0 & 0 & 2.0+3.0i \end{pmatrix},$$

$$E = \begin{pmatrix} 1.0 & 1.0+1.0i & 1.0-1.0i & 1.0+1.0i \\ 0 & 2.0 & 1.0+1.0i & 1.0-1.0i \\ 0 & 0 & 2.0 & 1.0+1.0i \\ 0 & 0 & 0 & 1.0 \end{pmatrix},$$

$$C = \begin{pmatrix} -13.0+9.0i & 2.0+8.0i & -2.0+8.0i & -2.0+5.0i \\ -9.0-1.0i & 0.0+5.0i & -7.0-3.0i & -6.0-0.0i \\ -1.0+1.0i & 4.0+2.0i & 4.0-5.0i & 9.0-5.0i \\ -6.0+6.0i & 9.0+1.0i & -2.0+4.0i & 22.0-8.0i \end{pmatrix}$$

and

$$F = \begin{pmatrix} -6.0+5.0i & 4.0-4.0i & -3.0+11.0i & 3.0-7.0i \\ -5.0+11.0i & 12.0-4.0i & -2.0+2.0i & 0.0+14.0i \\ -5.0-1.0i & 0.0+4.0i & -2.0+10.0i & 3.0-1.0i \\ -6.0-2.0i & 1.0+1.0i & -7.0-3.0i & 4.0+7.0i \end{pmatrix}.$$

10.1 Program Text

```
/* nag_ztgsyl (f08yvc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf08.h>

int main(void)
{
  /* Scalars */
  double        dif, scale;
  Integer       i, ijob, j, m, n, pda, pdb, pdc, pdd, pde, pdf;
  Integer       exit_status = 0;

  /* Arrays */
  Complex       *a = 0, *b = 0, *c = 0, *d = 0, *e = 0, *f = 0;
  char          nag_enum_arg[40];
```

```
  /* Nag Types */
  NagError      fail;
  Nag_OrderType order;
  Nag_TransType trans;

  /* K(I,J) maps matrix element (I,J) to array storage element k */
#ifdef NAG_COLUMN_MAJOR
#define K(I, J, PD) (J-1)*PD + I - 1
  order = Nag_ColMajor;
#else
#define K(I, J, PD) (I-1)*PD + J - 1
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_ztgsyl (f08yvc) 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]", &m, &n);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &m, &n);
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]", &ijob);
#else
  scanf("%"NAG_IFMT"%*[^\n]", &ijob);
#endif
  if (m < 0 || n < 0 || ijob < 0 || ijob > 4)
    {
      printf("Invalid m, n or ijob\n");
      exit_status = 1;
      goto END;
    }

#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
   */
  trans = ( Nag_TransType) nag_enum_name_to_value(nag_enum_arg);

  pda = m;
  pdb = n;
  pdd = m;
  pde = n;
#ifdef NAG_COLUMN_MAJOR
  pdc = m;
  pdf = m;
#else
  pdc = n;
  pdf = n;
#endif

  /* Allocate memory */
  if (!(a = NAG_ALLOC(m*m, Complex)) ||
      !(b = NAG_ALLOC(n*n, Complex)) ||
      !(c = NAG_ALLOC(m*n, Complex)) ||
      !(d = NAG_ALLOC(m*m, Complex)) ||
      !(e = NAG_ALLOC(n*n, Complex)) ||
      !(f = NAG_ALLOC(m*n, Complex)))
    {
```

```
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read A, B, D, E, C and F from data file */
  for (i = 1; i <= m; ++i)
    for (j = 1; j <= m; ++j)
#ifdef _WIN32
      scanf_s(" ( %lf , %lf )", &a[K(i, j, pda)].re, &a[K(i, j, pda)].im);
#else
      scanf(" ( %lf , %lf )", &a[K(i, j, pda)].re, &a[K(i, j, pda)].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  for (i = 1; i <= n; ++i)
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
      scanf_s(" ( %lf , %lf )", &b[K(i, j, pdb)].re, &b[K(i, j, pdb)].im);
#else
      scanf(" ( %lf , %lf )", &b[K(i, j, pdb)].re, &b[K(i, j, pdb)].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  for (i = 1; i <= m; ++i)
    for (j = 1; j <= m; ++j)
#ifdef _WIN32
      scanf_s(" ( %lf , %lf )", &d[K(i, j, pdd)].re, &d[K(i, j, pdd)].im);
#else
      scanf(" ( %lf , %lf )", &d[K(i, j, pdd)].re, &d[K(i, j, pdd)].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  for (i = 1; i <= n; ++i)
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
      scanf_s(" ( %lf , %lf )", &e[K(i, j, pde)].re, &e[K(i, j, pde)].im);
#else
      scanf(" ( %lf , %lf )", &e[K(i, j, pde)].re, &e[K(i, j, pde)].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  for (i = 1; i <= m; ++i)
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
      scanf_s(" ( %lf , %lf )", &c[K(i, j, pdc)].re, &c[K(i, j, pdc)].im);
#else
      scanf(" ( %lf , %lf )", &c[K(i, j, pdc)].re, &c[K(i, j, pdc)].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  for (i = 1; i <= m; ++i)
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
      scanf_s(" ( %lf , %lf )", &f[K(i, j, pdf)].re, &f[K(i, j, pdf)].im);
#else
```

```
      scanf(" ( %lf , %lf )", &f[K(i, j, pdf)].re, &f[K(i, j, pdf)].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Solve the Sylvester equations:
   * A*R - L*B = scale*C
   * D*R - L*E = scale*F
   *                     for R and L using  nag_ztgsyl (f08yvc).
   */
  nag_ztgsyl(order, trans, ijob, m, n, a, pda, b, pdb, c, pdc, d, pdd, e, pde,
             f, pdf, &scale, &dif, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_ztgsyl (f08yvc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print the solution matrices R and L */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m,
                                n, c, pdc, Nag_BracketForm, "%7.4f",
                                "Solution matrix R", Nag_IntegerLabels, NULL,
                                Nag_IntegerLabels, NULL, 80, 0, NULL, &fail);
  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;
    }
  printf("\n");
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m,
                                n, f, pdf, Nag_BracketForm, "%7.4f",
                                "Solution matrix L", Nag_IntegerLabels, NULL,
                                Nag_IntegerLabels, NULL, 80, 0, NULL, &fail);
  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;
    }
  printf("\nscale = %11.2e\n", scale);

  if (ijob>0 && scale>0.0) {
    printf("\ndif   = %11.2e\n\n", dif);
    printf("This estimate of Dif((A,D),(B,E)) was computed based on the ");
    if (ijob==1 || ijob==3) {
      printf("Frobenius norm.\n");
    } else {
      printf("one norm.\n");
    }
  }

 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_ztgsyl (f08yvc) Example Program Data

    4             4                                          : m and n
    0                                                        : ijob
    Nag_NoTrans                                              : trans

 (  4.0,  4.0) (  1.0,  1.0) (  1.0,  1.0) (  2.0, -1.0)
 (  0.0,  0.0) (  2.0,  1.0) (  1.0,  1.0) (  1.0,  1.0)
 (  0.0,  0.0) (  0.0,  0.0) (  2.0, -1.0) (  1.0,  1.0)
 (  0.0,  0.0) (  0.0,  0.0) (  0.0,  0.0) (  6.0, -2.0)   : matrix A

 (  2.0,  0.0) (  1.0,  1.0) (  1.0,  1.0) (  3.0, -1.0)
 (  0.0,  0.0) (  1.0,  0.0) (  2.0,  1.0) (  1.0,  1.0)
 (  0.0,  0.0) (  0.0,  0.0) (  1.0,  0.0) (  1.0,  1.0)
 (  0.0,  0.0) (  0.0,  0.0) (  0.0,  0.0) (  2.0,  0.0)   : matrix B

 (  1.0,  1.0) (  1.0, -1.0) (  1.0,  1.0) (  1.0, -1.0)
 (  0.0,  0.0) (  6.0, -4.0) (  1.0, -1.0) (  1.0,  1.0)
 (  0.0,  0.0) (  0.0,  0.0) (  2.0,  4.0) (  1.0, -1.0)
 (  0.0,  0.0) (  0.0,  0.0) (  0.0,  0.0) (  2.0,  3.0)   : matrix D

 (  1.0,  0.0) (  1.0,  1.0) (  1.0, -1.0) (  1.0,  1.0)
 (  0.0,  0.0) (  2.0,  0.0) (  1.0,  1.0) (  1.0, -1.0)
 (  0.0,  0.0) (  0.0,  0.0) (  2.0,  0.0) (  1.0,  1.0)
 (  0.0,  0.0) (  0.0,  0.0) (  0.0,  0.0) (  1.0,  0.0)   : matrix E

 (-13.0,  9.0) (  2.0,  8.0) ( -2.0,  8.0) ( -2.0,  5.0)
 ( -9.0, -1.0) (  0.0,  5.0) ( -7.0, -3.0) ( -6.0,  0.0)
 ( -1.0,  1.0) (  4.0,  2.0) (  4.0, -5.0) (  9.0, -5.0)
 ( -6.0,  6.0) (  9.0,  1.0) ( -2.0,  4.0) ( 22.0, -8.0)   : matrix C

 ( -6.0,  5.0) (  4.0, -4.0) ( -3.0, 11.0) (  3.0, -7.0)
 ( -5.0, 11.0) ( 12.0, -4.0) ( -2.0,  2.0) (  0.0, 14.0)
 ( -5.0, -1.0) (  0.0,  4.0) ( -2.0, 10.0) (  3.0, -1.0)
 ( -6.0, -2.0) (  1.0,  1.0) ( -7.0, -3.0) (  4.0,  7.0)   : matrix F
```

10.3 Program Results

```
nag_ztgsyl (f08yvc) Example Program Results

 Solution matrix R
                     1                    2                    3                    4
 1  ( 1.0000, 1.0000)  ( 1.0000, 1.0000)  ( 1.0000, 1.0000)  ( 1.0000, 1.0000)
 2  (-1.0000, 1.0000)  ( 2.0000, 1.0000)  (-1.0000, 1.0000)  (-1.0000, 1.0000)
 3  (-1.0000, 1.0000)  ( 1.0000, 1.0000)  ( 3.0000, 1.0000)  ( 1.0000, 1.0000)
 4  (-1.0000, 1.0000)  ( 1.0000, 1.0000)  (-1.0000, 1.0000)  ( 4.0000, 1.0000)

 Solution matrix L
                     1                    2                    3                    4
 1  ( 4.0000, 1.0000)  (-1.0000, 1.0000)  ( 1.0000, 1.0000)  (-1.0000, 1.0000)
 2  ( 1.0000, 1.0000)  ( 3.0000, 1.0000)  (-1.0000, 1.0000)  ( 1.0000, 1.0000)
 3  (-1.0000, 1.0000)  ( 1.0000, 1.0000)  ( 2.0000, 1.0000)  (-1.0000, 1.0000)
 4  ( 1.0000, 1.0000)  (-1.0000, 1.0000)  ( 1.0000, 1.0000)  ( 1.0000, 1.0000)

scale =    1.00e+00
```