

# NAG Library Function Document

## nag\_dtgsyl (f08yhc)

### 1 Purpose

nag\_dtgsyl (f08yhc) solves the generalized real quasi-triangular Sylvester equations.

### 2 Specification

```
#include <nag.h>
#include <nagf08.h>

void nag_dtgsyl (Nag_OrderType order, Nag_TransType trans, Integer ijob,
                Integer m, Integer n, const double a[], Integer pda, const double b[],
                Integer pdb, double c[], Integer pdc, const double d[], Integer pdd,
                const double e[], Integer pde, double f[], Integer pdf, double *scale,
                double *dif, NagError *fail)
```

### 3 Description

nag\_dtgsyl (f08yhc) solves either the generalized real Sylvester equations

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

or the equations

$$\begin{aligned} A^T R + D^T L &= \alpha C \\ RB^T + LE^T &= -\alpha F, \end{aligned} \quad (2)$$

where the pair  $(A, D)$  are given  $m$  by  $m$  matrices in real generalized Schur form,  $(B, E)$  are given  $n$  by  $n$  matrices in real generalized Schur form and  $(C, F)$  are given  $m$  by  $n$  matrices. The pair  $(R, L)$  are the  $m$  by  $n$  solution matrices, and  $\alpha$  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

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

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

$$Z^T y = \alpha b.$$

The pair  $(S, T)$  are in real generalized Schur form if  $S$  is block upper triangular with 1 by 1 and 2 by 2 diagonal blocks on the diagonal and  $T$  is upper triangular as returned, for example, by nag\_dgges (f08xac), or nag\_dhgeqz (f08xec) with **job** = Nag\_Schur.

Optionally, the function estimates  $\text{Dif}[(A, D), (B, E)]$ , 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\_Trans, solve the ‘transposed’ system (2).
- Constraint:* **trans** = Nag\_NoTrans or Nag\_Trans.
- 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  $\text{Dif}[(A, D), (B, E)]$  is computed based on the Frobenius norm.
- ijob** = 4  
Only an estimate of  $\text{Dif}[(A, D), (B, E)]$  is computed based on the 1-norm.
- If **trans** = Nag\_Trans, **ijob** is not referenced.
- Constraint:* if **trans** = Nag\_NoTrans,  $0 \leq \text{ijob} \leq 4$ .
- 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**  $\geq$  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**  $\geq$  0.

- 6: **a**[*dim*] – const double *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 quasi-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 double *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 quasi-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*] – double *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 double *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

$$\begin{aligned} & \mathbf{d}[(j-1) \times \mathbf{pdd} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ & \mathbf{d}[(i-1) \times \mathbf{pdd} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$

*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:*  $\mathbf{pdd} \geq \max(1, \mathbf{m})$ .

14: **e**[*dim*] – const double *Input*

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

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

$$\begin{aligned} & \mathbf{e}[(j-1) \times \mathbf{pde} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ & \mathbf{e}[(i-1) \times \mathbf{pde} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$

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

16: **f**[*dim*] – double *Input/Output*

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

$$\begin{aligned} & \max(1, \mathbf{pdf} \times \mathbf{n}) \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ & \max(1, \mathbf{m} \times \mathbf{pdf}) \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$

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

$$\begin{aligned} & \mathbf{f}[(j-1) \times \mathbf{pdf} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ & \mathbf{f}[(i-1) \times \mathbf{pdf} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$

*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:*

$$\begin{aligned} & \text{if } \mathbf{order} = \text{Nag\_ColMajor}, \mathbf{pdf} \geq \max(1, \mathbf{m}); \\ & \text{if } \mathbf{order} = \text{Nag\_RowMajor}, \mathbf{pdf} \geq \max(1, \mathbf{n}). \end{aligned}$$

18: **scale** – double \* *Output*

*On exit:*  $\alpha$ , the scaling factor in (1) or (2).

If  $0 < \mathbf{scale} < 1$ , **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  $\langle value \rangle$  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** =  $\langle value \rangle$  and **ijob** =  $\langle value \rangle$ .

Constraint: if **trans** = Nag\_NoTrans,  $0 \leq \mathbf{ijob} \leq 4$ .

### NE\_INT

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

Constraint: **m**  $\geq 0$ .

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

Constraint: **n**  $\geq 0$ .

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

Constraint: **pda**  $> 0$ .

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

Constraint: **pdb**  $> 0$ .

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

Constraint: **pdc**  $> 0$ .

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

Constraint: **pdd**  $> 0$ .

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

Constraint: **pde**  $> 0$ .

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

Constraint: **pdf**  $> 0$ .

### 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, **n**).

On entry, **pdd** =  $\langle value \rangle$  and **m** =  $\langle value \rangle$ .

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

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

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

On entry, **pdf** =  $\langle value \rangle$  and **m** =  $\langle value \rangle$ .

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

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

Constraint: **pdf**  $\geq$  max(1, **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\_dtgsyl (f08yhc) is not threaded by NAG in any implementation.

nag\_dtgsyl (f08yhc) 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  $2mn(n + m)$ . The Frobenius norm estimate of Dif does not require additional significant computation, but the 1-norm estimate is typically five times more expensive.

The complex analogue of this function is nag\_ztgsyl (f08yvc).

## 10 Example

This example solves the generalized Sylvester equations

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

where

$$A = \begin{pmatrix} 4.0 & 1.0 & 1.0 & 2.0 \\ 0 & 3.0 & 4.0 & 1.0 \\ 0 & 1.0 & 3.0 & 1.0 \\ 0 & 0 & 0 & 6.0 \end{pmatrix}, \quad B = \begin{pmatrix} 1.0 & 1.0 & 1.0 & 1.0 \\ 0 & 3.0 & 4.0 & 1.0 \\ 0 & 1.0 & 3.0 & 1.0 \\ 0 & 0 & 0 & 4.0 \end{pmatrix},$$

$$D = \begin{pmatrix} 2.0 & 1.0 & 1.0 & 3.0 \\ 0 & 1.0 & 2.0 & 1.0 \\ 0 & 0 & 1.0 & 1.0 \\ 0 & 0 & 0 & 2.0 \end{pmatrix}, \quad E = \begin{pmatrix} 1.0 & 1.0 & 1.0 & 2.0 \\ 0 & 1.0 & 4.0 & 1.0 \\ 0 & 0 & 1.0 & 1.0 \\ 0 & 0 & 0 & 1.0 \end{pmatrix},$$

$$C = \begin{pmatrix} -4.0 & 7.0 & 1.0 & 12.0 \\ -9.0 & 2.0 & -2.0 & -2.0 \\ -4.0 & 2.0 & -2.0 & 8.0 \\ -7.0 & 7.0 & -6.0 & 19.0 \end{pmatrix} \quad \text{and} \quad F = \begin{pmatrix} -7.0 & 5.0 & 0.0 & 7.0 \\ -5.0 & 1.0 & -8.0 & 0.0 \\ -1.0 & 2.0 & -3.0 & 5.0 \\ -3.0 & 2.0 & 0.0 & 5.0 \end{pmatrix}.$$

## 10.1 Program Text

```

/* nag_dtgsyl (f08yhc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagx04.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 */
    double      *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_dtgsyl (f08yhc) 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, double)) ||
        !(b = NAG_ALLOC(n*n, double)) ||
        !(c = NAG_ALLOC(m*n, double)) ||
        !(d = NAG_ALLOC(m*m, double)) ||
        !(e = NAG_ALLOC(n*n, double)) ||
        !(f = NAG_ALLOC(m*n, double)))
    {
        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)
#ifdef _WIN32
        for (j = 1; j <= m; ++j) scanf_s("%lf", &a[K(i, j, pda)]);
#else
        for (j = 1; j <= m; ++j) scanf("%lf", &a[K(i, j, pda)]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
    for (i = 1; i <= n; ++i)
#ifdef _WIN32
        for (j = 1; j <= n; ++j) scanf_s("%lf", &b[K(i, j, pdb)]);
#else
        for (j = 1; j <= n; ++j) scanf("%lf", &b[K(i, j, pdb)]);

```



```

#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
    for (i = 1; i <= m; ++i)
#ifdef _WIN32
        for (j = 1; j <= m; ++j) scanf_s("%lf", &d[K(i, j, pdd)]);
#else
        for (j = 1; j <= m; ++j) scanf("%lf", &d[K(i, j, pdd)]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
    for (i = 1; i <= n; ++i)
#ifdef _WIN32
        for (j = 1; j <= n; ++j) scanf_s("%lf", &e[K(i, j, pde)]);
#else
        for (j = 1; j <= n; ++j) scanf("%lf", &e[K(i, j, pde)]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
    for (i = 1; i <= m; ++i)
#ifdef _WIN32
        for (j = 1; j <= n; ++j) scanf_s("%lf", &c[K(i, j, pdc)]);
#else
        for (j = 1; j <= n; ++j) scanf("%lf", &c[K(i, j, pdc)]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
    for (i = 1; i <= m; ++i)
#ifdef _WIN32
        for (j = 1; j <= n; ++j) scanf_s("%lf", &f[K(i, j, pdf)]);
#else
        for (j = 1; j <= n; ++j) scanf("%lf", &f[K(i, j, pdf)]);
#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_dtgsyl (f08yhc).
 */
nag_dtgsyl(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_dtgsyl (f08yhc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print the solution matrices R and L */
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m, n,
                      c, pdc, "Solution matrix R", 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;
}
printf("\n");
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m, n,
                       f, pdf, "Solution matrix L", 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;
}
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\_dtgsyl (f08yhc) Example Program Data

```

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

4.0    1.0    1.0    2.0
0.0    3.0    4.0    1.0
0.0    1.0    3.0    1.0
0.0    0.0    0.0    6.0 : matrix A

1.0    1.0    1.0    1.0
0.0    3.0    4.0    1.0
0.0    1.0    3.0    1.0
0.0    0.0    0.0    4.0 : matrix B

2.0    1.0    1.0    3.0
0.0    1.0    2.0    1.0
0.0    0.0    1.0    1.0
0.0    0.0    0.0    2.0 : matrix D

1.0    1.0    1.0    2.0
0.0    1.0    4.0    1.0
0.0    0.0    1.0    1.0
0.0    0.0    0.0    1.0 : matrix E

-4.0   7.0    1.0   12.0
-9.0   2.0   -2.0   -2.0
-4.0   2.0   -2.0    8.0

```

```
-7.0  7.0  -6.0  19.0   : matrix C
-7.0  5.0   0.0   7.0
-5.0  1.0  -8.0   0.0
-1.0  2.0  -3.0   5.0
-3.0  2.0   0.0   5.0   : matrix F
```

### 10.3 Program Results

nag\_dtgsyl (f08yhc) Example Program Results

```
Solution matrix R
      1      2      3      4
1      1.0000  1.0000  1.0000  1.0000
2     -1.0000  2.0000 -1.0000 -1.0000
3     -1.0000  1.0000  3.0000  1.0000
4     -1.0000  1.0000 -1.0000  4.0000
```

```
Solution matrix L
      1      2      3      4
1      4.0000 -1.0000  1.0000 -1.0000
2      1.0000  3.0000 -1.0000  1.0000
3     -1.0000  1.0000  2.0000 -1.0000
4      1.0000 -1.0000  1.0000  1.0000
```

scale = 1.00e+00

---