1 Purpose

Nag_dtrsyl (f08qhc) solves the real quasi-triangular Sylvester matrix equation.

2 Specification

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

void nag_dtrsyl (Nag_OrderType order, Nag_TransType trana,
                 Nag_TransType tranb, Nag_SignType sign, Integer m, Integer n,
                 const double a[], Integer pda, const double b[], Integer pdb,
                 double c[], Integer pdc, double *scale, NagError *fail)
```

3 Description

Nag_dtrsyl (f08qhc) solves the real Sylvester matrix equation

\[
\text{op}(A)X \pm X\text{op}(B) = \alpha C,
\]

where \(\text{op}(A) = A \text{ or } A^T\), and the matrices \(A\) and \(B\) are upper quasi-triangular matrices in canonical Schur form (as returned by nag_dhseqr (f08pec)); \(\alpha\) is a scale factor (\(\leq 1\)) determined by the function to avoid overflow in \(X\); \(A\) is \(m \times m\) and \(B\) is \(n \times n\) while the right-hand side matrix \(C\) and the solution matrix \(X\) are both \(m \times n\). 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


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 \(\text{op}(A)\).

   \(\text{trana} = \text{Nag\_NoTrans}\), \(\text{op}(A) = A\).
**Constraints:**

- `trans = Nag_Trans or Nag_ConjTrans`
  - `op(A) = A^T`.

**Constraint:** `trans = Nag_NoTrans, Nag_Trans or Nag_ConjTrans`.

3: `tranb` – `Nag_TransType`

**Input**

*On entry:* specifies the option `op(B)`.

- `tranb = Nag_NoTrans`
  - `op(B) = B`.

- `tranb = Nag_Trans or Nag_ConjTrans`
  - `op(B) = B^T`.

**Constraint:** `tranb = Nag_NoTrans, Nag_Trans 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 `op(A)X + X op(B) = αC`.

- `sign = Nag_Minus`
  - The equation is of the form `op(A)X - X op(B) = αC`.

**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:** `m ≥ 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:** `n ≥ 0`.

7: `a[dim]` – const double

**Input**

*Note:* the dimension, `dim`, of the array `a` must be at least `max(1, pda × m)`.

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

- `a[(j - 1) × pda + i - 1]` when `order = Nag_ColMajor`;
- `a[(i - 1) × pda + j - 1]` when `order = Nag_RowMajor`.

*On entry:* the `m` by `m` upper quasi-triangular matrix `A` in canonical Schur form, as returned by `nag_dhseqr (f08pec)`.

8: `pda` – Integer

**Input**

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

**Constraint:** `pda ≥ max(1, m)`.

9: `b[dim]` – const double

**Input**

*Note:* the dimension, `dim`, of the array `b` must be at least `max(1, pdb × n)`.

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

- `b[(j - 1) × pdb + i - 1]` when `order = Nag_ColMajor`;
- `b[(i - 1) × pdb + j - 1]` when `order = Nag_RowMajor`.
On entry: the \( n \) by \( n \) upper quasi-triangular matrix \( B \) in canonical Schur form, as returned by \nag_dhseqr (f08pec).

### 10: \( \textbf{pdb} \) – Integer

*Input*

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

\textit{Constraint:} \( \texttt{pdb} \geq \max(1, n) \).

### 11: \( \mathbf{c}[	ext{dim}] \) – double

*Input/Output*

\textit{Note:} the dimension, \( \text{dim} \), of the array \( \mathbf{c} \) must be at least

\[
\begin{align*}
\max(1, \text{pdb} \times n) & \text{ when } \texttt{order} = \texttt{Nag\_ColMajor}; \\
\max(1, m \times \text{pdb}) & \text{ when } \texttt{order} = \texttt{Nag\_RowMajor}.
\end{align*}
\]

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

\[
\begin{align*}
\mathbf{c}(j - 1) \times \text{pdb} + i - 1] & \text{ when } \texttt{order} = \texttt{Nag\_ColMajor}; \\
\mathbf{c}(i - 1) \times \text{pdb} + j - 1] & \text{ when } \texttt{order} = \texttt{Nag\_RowMajor}.
\end{align*}
\]

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

On exit: \( \mathbf{c} \) is overwritten by the solution matrix \( X \).

### 12: \( \textbf{pdc} \) – Integer

*Input*

On entry: the stride separating row or column elements (depending on the value of \texttt{order}) in the array \( \mathbf{c} \).

\textit{Constraints:}

\[
\begin{align*}
\text{if } \texttt{order} = \texttt{Nag\_ColMajor}, & \quad \text{pdb} \geq \max(1, m); \\
\text{if } \texttt{order} = \texttt{Nag\_RowMajor}, & \quad \text{pdc} \geq \max(1, n).
\end{align*}
\]

### 13: \( \textbf{scale} \) – double *

*Output*

On exit: the value of the scale factor \( \alpha \).

### 14: \( \textbf{fail} \) – 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 \textit{value} \rangle had an illegal value.

**NE_INT**

On entry, \( m = \langle \textit{value} \rangle \).

\textit{Constraint:} \( m \geq 0 \).

On entry, \( n = \langle \textit{value} \rangle \).

\textit{Constraint:} \( n \geq 0 \).

On entry, \( \texttt{pda} = \langle \textit{value} \rangle \).

\textit{Constraint:} \( \texttt{pda} > 0 \).
On entry, \( \text{pdb} = \langle \text{value} \rangle \).
Constraint: \( \text{pdb} > 0 \).

On entry, \( \text{pdc} = \langle \text{value} \rangle \).
Constraint: \( \text{pdc} > 0 \).

**NE_INT_2**

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

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

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

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

**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.

**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 \( \hat{X} \) be the computed solution and \( R \) the residual matrix:

\[
R = C - (A\hat{X} - \hat{X}B).
\]

Then the residual is always small:

\[
\| R \|_F = O(\epsilon)(\| A \|_F + \| B \|_F)\|\hat{X}\|_F.
\]

However, \( \hat{X} \) 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:

\[
\| \hat{X} - X \|_F \leq \frac{\| R \|_F}{\text{sep}(A, B)}
\]

but this may be a considerable over estimate. See Golub and Van Loan (1996) for a definition of \( \text{sep}(A, B) \), 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_dtrsyl (f08qhc) is not threaded by NAG in any implementation.

nag_dtrsyl (f08qhc) 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 is approximately $mn(m + n)$.

To solve the general real Sylvester equation

$$AX \pm XB = C$$

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

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

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

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

where $\tilde{X} = Q_1^T X Q_2$ and $\tilde{C} = Q_1^T C Q_2$. $\tilde{C}$ may be computed by matrix multiplication; nag_dtrsyl (f08qhc) may be used to solve the transformed equation; and the solution to the original equation can be obtained as $X = Q_1 \tilde{X} Q_2^T$.

The complex analogue of this function is nag_ztrsyl (f08qvc).

10 Example

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

$$A = \begin{pmatrix}
0.10 & 0.50 & 0.68 & -0.21 \\
-0.50 & 0.10 & -0.24 & 0.67 \\
0.00 & 0.00 & 0.19 & -0.35 \\
0.00 & 0.00 & 0.00 & -0.72
\end{pmatrix},$$

$$B = \begin{pmatrix}
-0.99 & -0.17 & 0.39 & 0.58 \\
0.00 & 0.48 & -0.84 & -0.15 \\
0.00 & 0.00 & 0.75 & 0.25 \\
0.00 & 0.00 & -0.25 & 0.75
\end{pmatrix},$$

and

$$C = \begin{pmatrix}
0.63 & -0.56 & 0.08 & -0.23 \\
-0.45 & -0.31 & 0.27 & 1.21 \\
0.20 & -0.35 & 0.41 & 0.84 \\
0.49 & -0.05 & -0.52 & -0.08
\end{pmatrix}.$$
10.1 Program Text

/* nag_dtrsyl (f08qhc) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
*/

#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 */
    Integer i, j, m, n, pda, pdb, pdc, pdd, pde, pdf;
    Integer exit_status = 0;
    double alpha, beta, norm, scale;
    Nag_SignType sign = Nag_Minus;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    double *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) d[(J-1)*pdd +I-1]
    #define E(I, J) e[(J-1)*pde +I-1]
    #define F(I, J) f[(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_dtrsyl (f08qhc) 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 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, double)) ||
    !(b = NAG_ALLOC(n * m, double)) ||
    !(c = NAG_ALLOC(m * n, double)) ||
    !(d = NAG_ALLOC(m * n, double)) ||
    !(e = NAG_ALLOC(m * n, double)) ||
    !(f = NAG_ALLOC(m * n, double))
)
{
    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)
        #ifdef _WIN32
        scanf_s("%lf", &A(i, j));
        #else
        scanf("%lf", &A(i, j));
        #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", &B(i, j));
            #else
            scanf("%lf", &B(i, j));
            #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", &C(i, j));
                #else
                scanf("%lf", &C(i, j));
                #endif
            #ifdef _WIN32
            scanf_s("%*[\n]");
            #else
            scanf("%*[\n]");
            #endif
        } /* Copy C into F */
    for (i = 1; i <= m; ++i)
    {
        for (j = 1; j <= m; ++j)
            F(i, j) = C(i, j);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, m, n, c, pdc, "Matrix C", 0, &fail);
printf("\n");
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print (x04cac).\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Reorder the Schur factorization T */
/* nag_dtrsyl (f08qhc).
 * Solve real Sylvester matrix equation AX + XB = C, A and B
 * are upper quasi-triangular or transposes
 */
nag_dtrsyl(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_dtrsyl (f08qhc).\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_dgemm (f16yac): Compute aC - (A*X + X*B*sign) from solution*/
/* and store in matrix E*/
alpha = 1.0;
beta = 0.0;
nag_dgemm(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_dgemm (f16yac).\n", fail.message);
    exit_status = 1;
    goto END;
}
if(sign == Nag_Minus)
    alpha = -1.0;
else
    alpha = 1.0;
beta = 1.0;
nag_dgemm(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_dgemm (f16yac).\n", fail.message);
    exit_status = 1;
    goto END;
}
for(i=1; i<=m; i++)
{
    for (j=1; j<=n; j++)
        E(i, j) = scale * F(i, j) - D(i, j);
}

/* nag_dge_norm (f16rac): Find norm of matrix E and print warning if */
/* it is too large */
nag_dge_norm(order, Nag_OneNorm, n, n, e, pde, &norm, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dge_norm (f16rac).\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.",
            "nag_dtrsyl (f08qhc) 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_dtrsyl (f08qhc) Example Program Data

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>Values of M and N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.10</td>
<td>0.50</td>
<td>0.68</td>
<td>-0.21</td>
<td></td>
</tr>
<tr>
<td>-0.50</td>
<td>0.10</td>
<td>-0.24</td>
<td>0.67</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.19</td>
<td>-0.35</td>
<td></td>
</tr>
<tr>
<td>-0.99</td>
<td>-0.17</td>
<td>0.39</td>
<td>0.58</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.48</td>
<td>-0.84</td>
<td>-0.15</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.75</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>-0.25</td>
<td>0.75</td>
<td></td>
</tr>
<tr>
<td>0.63</td>
<td>-0.56</td>
<td>0.08</td>
<td>-0.23</td>
<td></td>
</tr>
<tr>
<td>-0.45</td>
<td>-0.31</td>
<td>0.27</td>
<td>1.21</td>
<td></td>
</tr>
<tr>
<td>0.20</td>
<td>-0.35</td>
<td>0.41</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>0.49</td>
<td>-0.05</td>
<td>-0.52</td>
<td>-0.08</td>
<td></td>
</tr>
</tbody>
</table>

10.3 Program Results

nag_dtrsyl (f08qhc) Example Program Results

Matrix C

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>0.6300</td>
<td>-0.5600</td>
<td>0.0800</td>
<td>-0.2300</td>
<td></td>
</tr>
<tr>
<td>-0.4500</td>
<td>-0.3100</td>
<td>0.2700</td>
<td>1.2100</td>
<td></td>
</tr>
<tr>
<td>0.2000</td>
<td>-0.3500</td>
<td>0.4100</td>
<td>0.8400</td>
<td></td>
</tr>
<tr>
<td>0.4900</td>
<td>-0.0500</td>
<td>-0.5200</td>
<td>-0.0800</td>
<td></td>
</tr>
</tbody>
</table>

SCALE = 1.00e+00