NAG Library Function Document

nag_ztgsen (f08yuc)

1 Purpose

nag_ztgsen (f08yuc) reorders the generalized Schur factorization of a complex matrix pair in generalized
Schur form, so that a selected cluster of eigenvalues appears in the leading elements on the diagonal of
the generalized Schur form. The function also, optionally, computes the reciprocal condition numbers of
the cluster of eigenvalues and/or corresponding deflating subspaces.

2 Specification

```c
#include <nag.h>
#include <nagf08.h>
void nag_ztgsen (Nag_OrderType order, Integer ijob, Nag_Boolean wantq,
    Nag_Boolean wantz, const Nag_Boolean select[], Integer n, Complex a[],
    Integer pda, Complex b[], Integer pdb, Complex alpha[], Complex beta[],
    Complex q[], Integer pdq, Complex z[], Integer pdz, Integer *m,
    double *pl, double *pr, double dif[], NagError *fail)
```

3 Description

nag_ztgsen (f08yuc) factorizes the generalized complex n by n matrix pair \((S, T)\) in generalized Schur form, using a unitary equivalence transformation as

\[
S = Q \hat{S} Z^H, \quad T = Q \hat{T} Z^H,
\]

where \((\hat{S}, \hat{T})\) are also in generalized Schur form and have the selected eigenvalues as the leading
diagonal elements. The leading columns of \(Q\) and \(Z\) are the generalized Schur vectors corresponding to
the selected eigenvalues and form orthonormal subspaces for the left and right eigenspaces (deflating
subspaces) of the pair \((S, T)\).

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 \(\text{job} = \text{Nag Schur}\). The diagonal elements define the
generalized eigenvalues \((\alpha_i, \beta_i)\), for \(i = 1, 2, \ldots, n\), of the pair \((S, T)\). The eigenvalues are given by

\[
\lambda_i = \alpha_i / \beta_i,
\]

but are returned as the pair \((\alpha_i, \beta_i)\) in order to avoid possible overflow in computing \(\lambda_i\). Optionally,
the function returns reciprocals of condition number estimates for the selected eigenvalue cluster, \(p\) and \(q\),
the right and left projection norms, and of deflating subspaces, \(\text{Dif}_u\) and \(\text{Dif}_l\). For more information see
Sections 2.4.8 and 4.11 of Anderson et al. (1999).

If \(S\) and \(T\) are the result of a generalized Schur factorization of a matrix pair \((A, B)\)

\[
A = QSZ^H, \quad B = QTZ^H
\]

then, optionally, the matrices \(Q\) and \(Z\) can be updated as \(Q \hat{Q}\) and \(Z \hat{Z}\). Note that the condition numbers
of the pair \((S, T)\) are the same as those of the pair \((A, B)\).

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A,
Philadelphia http://www.netlib.org/lapack/lug
5 Arguments

1: order – Nag_OrderType  
   Input
   
   On entry: the order argument specifies the two-dimensional storage scheme being used, i.e., row-maj 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: ijob – Integer  
   Input
   
   On entry: specifies whether condition numbers are required for the cluster of eigenvalues (p and q) or the deflating subspaces (Dif_u and Dif_l).

   ijob = 0
   Only reorder with respect to select. No extras.

   ijob = 1
   Reciprocal of norms of ‘projections’ onto left and right eigenspaces with respect to the selected cluster (p and q).

   ijob = 2
   The upper bounds on Dif_u and Dif_l. F-norm-based estimate (stored in dif[0] and dif[1] respectively).

   ijob = 3
   Estimate of Dif_u and Dif_l. 1-norm-based estimate (stored in dif[0] and dif[1] respectively). About five times as expensive as ijob = 2.

   ijob = 4
   Compute pl, pr and dif as in ijob = 0, 1 and 2. Economic version to get it all.

   ijob = 5
   Compute pl, pr and dif as in ijob = 0, 1 and 3.

   Constraint: 0 ≤ ijob ≤ 5.

3: wantq – Nag_Boolean  
   Input
   
   On entry: if wantq = Nag_TRUE, update the left transformation matrix Q. If wantq = Nag_FALSE, do not update Q.

4: wantz – Nag_Boolean  
   Input
   
   On entry: if wantz = Nag_TRUE, update the right transformation matrix Z. If wantz = Nag_FALSE, do not update Z.

5: select[n] – const Nag_Boolean  
   Input
   
   On entry: specifies the eigenvalues in the selected cluster. To select an eigenvalue λ_j, select[j − 1] must be set to Nag_TRUE.

6: n – Integer  
   Input
   
   On entry: n, the order of the matrices S and T.

   Constraint: n ≥ 0.

7: a[dim] – Complex  
   Input/Output

   Note: the dimension, dim, of the array a must be at least max(1, pda × n).
The \((i, j)\)th element of the matrix \(A\) is stored in
\[
a[(j - 1) \times \text{pda} + i - 1] \text{ when order = Nag_ColMajor;}
a[(i - 1) \times \text{pda} + j - 1] \text{ when order = Nag_RowMajor.}
\]

**On entry:** the matrix \(S\) in the pair \((S, T)\).

**On exit:** the updated matrix \(\tilde{S}\).

8: \(\text{pda} \rightarrow \text{Integer} \quad \text{Input}\)

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

**Constraint:** \(\text{pda} \geq \max(1, n)\).

9: \(b[\text{dim}] \rightarrow \text{Complex} \quad \text{Input/Output}\)

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

The \((i, j)\)th element of the matrix \(B\) is stored in
\[
b[(j - 1) \times \text{pdb} + i - 1] \text{ when order = Nag_ColMajor;}
b[(i - 1) \times \text{pdb} + j - 1] \text{ when order = Nag_RowMajor.}
\]

**On entry:** the matrix \(T\), in the pair \((S, T)\).

**On exit:** the updated matrix \(\tilde{T}\).

10: \(\text{pdb} \rightarrow \text{Integer} \quad \text{Input}\)

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

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

11: \(\text{alpha}[n] \rightarrow \text{Complex} \quad \text{Output}\)

12: \(\text{beta}[n] \rightarrow \text{Complex} \quad \text{Output}\)

**On exit:** \(\text{alpha}\) and \(\text{beta}\) contain diagonal elements of \(\tilde{S}\) and \(\tilde{T}\), respectively, when the pair \((S, T)\) has been reduced to generalized Schur form. \(\text{alpha}[i - 1]/\text{beta}[i - 1]\), for \(i = 1, 2, \ldots, n\), are the eigenvalues.

13: \(q[\text{dim}] \rightarrow \text{Complex} \quad \text{Input/Output}\)

**Note:** the dimension, \(\text{dim}\), of the array \(q\) must be at least
\[
\max(1, \text{pdq} \times n) \text{ when wantq = Nag_TRUE;}
1 \text{ otherwise.}
\]

The \((i, j)\)th element of the matrix \(Q\) is stored in
\[
q[(j - 1) \times \text{pdq} + i - 1] \text{ when order = Nag_ColMajor;}
q[(i - 1) \times \text{pdq} + j - 1] \text{ when order = Nag_RowMajor.}
\]

**On entry:** if \(\text{wantq} = \text{Nag_TRUE}\), the \(n\) by \(n\) matrix \(Q\).

**On exit:** if \(\text{wantq} = \text{Nag_TRUE}\), the updated matrix \(\tilde{Q}\).

If \(\text{wantq} = \text{Nag_FALSE}\), \(q\) is not referenced.

14: \(\text{pdq} \rightarrow \text{Integer} \quad \text{Input}\)

**On entry:** the stride separating row or column elements (depending on the value of \text{order}) in the array \(q\).
Constraints:

if wantq = Nag_TRUE, pdq ≥ max(1, n);
otherwise pdq ≥ 1.

15: z[dim] — Complex

Note: the dimension, dim, of the array z must be at least
max(1, pdz × n) when wantz = Nag_TRUE;
1 otherwise.

The (i, j)th element of the matrix Z is stored in
z[(j - 1) × pdz + i - 1] when order = Nag_ColMajor;
z[(i - 1) × pdz + j - 1] when order = Nag_RowMajor.

On entry: if wantz = Nag_TRUE, the n by n matrix Z.
On exit: if wantz = Nag_TRUE, the updated matrix ZZ.
If wantz = Nag_FALSE, z is not referenced.

16: pdz — Integer

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

Constraints:

if wantz = Nag_TRUE, pdz ≥ max(1, n);
otherwise pdz ≥ 1.

17: m — Integer *

On exit: the dimension of the specified pair of left and right eigenspaces (deflating subspaces).

Constraint: 0 ≤ m ≤ n.

18: pl — double *
19: pr — double *

On exit: if ijob = 1, 4 or 5, pl and pr are lower bounds on the reciprocal of the norm of ‘projections’ p and q onto left and right eigenspace with respect to the selected cluster. 0 < pl, pr ≤ 1.
If m = 0 or m = n, pl = pr = 1.
If ijob = 0, 2 or 3, pl and pr are not referenced.

20: dif[dim] — double

Note: the dimension, dim, of the array dif must be at least 2.
On exit: if ijob ≥ 2, dif[0] and dif[1] store the estimates of Dif_u and Dif_l.
If ijob = 2 or 4, dif[0] and dif[1] are F-norm-based upper bounds on Dif_u and Dif_l.
If ijob = 3 or 5, dif[0] and dif[1] are 1-norm-based estimates of Dif_u and Dif_l.
If m = 0 or n, dif[0] and dif[1] = ∥(A, B)∥_F.
If ijob = 0 or 1, dif is not referenced.

21: fail — NagError *

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 \(\text{value}\) had an illegal value.

**NE_CONSTRAINT**
On entry, \(\text{wantq} = \langle\text{value}\rangle, \text{pdq} = \langle\text{value}\rangle\) and \(n = \langle\text{value}\rangle\).
Constraint: if \(\text{wantq} = \text{Nag\_TRUE}\), \(\text{pdq} \geq \max(1, n)\); otherwise \(\text{pdq} \geq 1\).
On entry, \(\text{wantz} = \langle\text{value}\rangle, \text{pdz} = \langle\text{value}\rangle\) and \(n = \langle\text{value}\rangle\).
Constraint: if \(\text{wantz} = \text{Nag\_TRUE}\), \(\text{pdz} \geq \max(1, n)\); otherwise \(\text{pdz} \geq 1\).

**NE_INT**
On entry, \(ijob = \langle\text{value}\rangle\).
Constraint: \(0 \leq ijob \leq 5\).
On entry, \(n = \langle\text{value}\rangle\).
Constraint: \(n \geq 0\).
On entry, \(pda = \langle\text{value}\rangle\).
Constraint: \(pda > 0\).
On entry, \(pdb = \langle\text{value}\rangle\).
Constraint: \(pdb > 0\).
On entry, \(pdq = \langle\text{value}\rangle\).
Constraint: \(pdq > 0\).
On entry, \(pdz = \langle\text{value}\rangle\).
Constraint: \(pdz > 0\).

**NE_INT_2**
On entry, \(pda = \langle\text{value}\rangle\) and \(n = \langle\text{value}\rangle\).
Constraint: \(pda \geq \max(1, n)\).
On entry, \(pdb = \langle\text{value}\rangle\) and \(n = \langle\text{value}\rangle\).
Constraint: \(pdb \geq \max(1, n)\).

**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.
NE_SCHUR

Reordering of \((S, T)\) failed because the transformed matrix pair would be too far from generalized Schur form; the problem is very ill-conditioned. \((S, T)\) may have been partially reordered. If requested, 0 is returned in \(\text{dif}[0]\) and \(\text{dif}[1]\), \(\text{pl}\) and \(\text{pr}\).

7 Accuracy

The computed generalized Schur form is nearly the exact generalized Schur form for nearby matrices \((S + E)\) and \((T + F)\), where

\[
\|E\|_2 = O\epsilon\|S\|_2 \quad \text{and} \quad \|F\|_2 = O\epsilon\|T\|_2,
\]

and \(\epsilon\) is the \textit{machine precision}. See Section 4.11 of Anderson et al. (1999) for further details of error bounds for the generalized nonsymmetric eigenproblem, and for information on the condition numbers returned.

8 Parallelism and Performance

\text{f08yuc} is not threaded by NAG in any implementation.

\text{f08yuc} 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 real analogue of this function is \text{f08ygc}.

10 Example

This example reorders the generalized Schur factors \(S\) and \(T\) and update the matrices \(Q\) and \(Z\) given by

\[
S = \begin{pmatrix}
4.0 + 4.0i & 1.0 + 1.0i & 1.0 + 1.0i & 2.0 - 1.0i \\
0 & 2.0 + 1.0i & 1.0 + 1.0i & 1.0 + 1.0i \\
0 & 0 & 2.0 - 1.0i & 1.0 + 1.0i \\
0 & 0 & 0 & 6.0 - 2.0i \\
\end{pmatrix},
\]

\[
T = \begin{pmatrix}
2.0 & 1.0 + 1.0i & 1.0 + 1.0i & 3.0 - 1.0i \\
0 & 1.0 & 2.0 + 1.0i & 1.0 + 1.0i \\
0 & 0 & 1.0 & 1.0 + 1.0i \\
0 & 0 & 0 & 2.0 \\
\end{pmatrix},
\]

\[
Q = \begin{pmatrix}
1.0 & 0 & 0 & 0 \\
0 & 1.0 & 0 & 0 \\
0 & 0 & 1.0 & 0 \\
0 & 0 & 0 & 1.0 \\
\end{pmatrix} \quad \text{and} \quad Z = \begin{pmatrix}
1.0 & 0 & 0 & 0 \\
0 & 1.0 & 0 & 0 \\
0 & 0 & 1.0 & 0 \\
0 & 0 & 0 & 1.0 \\
\end{pmatrix},
\]

selecting the second and third generalized eigenvalues to be moved to the leading positions. Bases for the left and right deflating subspaces, and estimates of the condition numbers for the eigenvalues and Frobenius norm based bounds on the condition numbers for the deflating subspaces are also output.
10.1 Program Text

/* nag_ztgsen (f08yuc) Example Program. *
 * Copyright 2014 Numerical Algorithms Group. *
 * Mark 23, 2011. */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf08.h>
#include <nagx02.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    double abs_a, abs_b, pl, pr, small;
    Complex eig;
    Integer i, ijob, j, m, n, pds, pdt, pdq, pdz;
    Integer exit_status = 0;

    /* Arrays */
    Complex *alpha = 0, *beta = 0, *q = 0, *s = 0, *t = 0, *z = 0;
    double dif[2];
    char nag_enum_arg[40];

    /* Nag Types */
    NagError fail;
    Nag_OrderType order;
    Nag_Boolean wantq, wantz;
    Nag_Boolean *select = 0;

    #ifdef NAG_COLUMN_MAJOR
    #define Q(I, J) q[(J-1)*pdq +I-1]
    #define Z(I, J) z[(J-1)*pdt +I-1]
    #define S(I, J) s[(J-1)*pds +I-1]
    #define T(I, J) t[(J-1)*pdq +I-1]
    #else
    #define Q(I, J) q[(I-1)*pdq+J-1]
    #define Z(I, J) z[(I-1)*pdt +J-1]
    #define S(I, J) s[(I-1)*pds +J-1]
    #define T(I, J) t[(I-1)*pdq +J-1]
    #endif

    order = Nag_ColMajor;

    INIT_FAIL(fail);

    printf("nag_ztgsen (f08yuc) 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]", &n, &ijob);
    #else
    scanf("%"NAG_IFMT"%"NAG_IFMT"%*[\n]", &n, &ijob);
    #endif
    if (n < 0 || ijob<0 || ijob>5)
    {
        printf("Invalid 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
 */
wantq = (Nag_Boolean) nag_enum_name_to_value(nag_enum_arg);
ifdef _WIN32
    scanf_s("%39s\n", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%39s\n", nag_enum_arg);
#endif

wantz = (Nag_Boolean) nag_enum_name_to_value(nag_enum_arg);

pds = n;
pdt = n;
pdq = (wantq?n:1);
pdz = (wantz?n:1);

/* Allocate memory */
if (!(s = NAG_ALLOC(n*n, Complex)) ||
    !(t = NAG_ALLOC(n*n, Complex)) ||
    !(alpha = NAG_ALLOC(n, Complex)) ||
    !(beta = NAG_ALLOC(n, Complex)) ||
    !(select = NAG_ALLOC(n, Nag_Boolean)) ||
    !(q = NAG_ALLOC(pdq*pdq, Complex)) ||
    !(z = NAG_ALLOC(pdz*pdz, Complex)))
{
    printf("Allocation failure\n");
    exit_status = -1;
goto END;
}

/* Read S, T, Q, Z and the logical array select from data file */
for (i = 0; i < n; ++i)
{
    ifdef _WIN32
        scanf_s("%39s", nag_enum_arg, _countof(nag_enum_arg));
    #else
        scanf("%39s", nag_enum_arg);
    #endif
    scanf("%39s", nag_enum_arg);
}

ifdef _WIN32
    scanf_s("%*\n");
#else
    scanf("%*\n");
#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 )", &S(i, j).re, &S(i, j).im);
        #else
            scanf("( %lf , %lf )", &S(i, j).re, &S(i, j).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 )", &T(i, j).re, &T(i, j).im);
        #else
            scanf_s("( %lf , %lf )", &T(i, j).re, &T(i, j).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 )", &Q(i, j).re, &Q(i, j).im);
        #else
            scanf("( %lf , %lf )", &Q(i, j).re, &Q(i, j).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 )", &Z(i, j).re, &Z(i, j).im);
        #else
            scanf("( %lf , %lf )", &Z(i, j).re, &Z(i, j).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 )", &L(i, j).re, &L(i, j).im);
        #else
            scanf("( %lf , %lf )", &L(i, j).re, &L(i, j).im);
        #endif
```c
scanf(" ( %lf , %lf )", &T(i, j).re, &T(i, j).im);
#endif
#endif
scanf_s("%*[\n]");
#else
scanf("%*[\n]");
#endif
if (wantq) {
    for (i = 1; i <= n; ++i)
        for (j = 1; j <= n; ++j)
            #ifdef _WIN32
                scanf_s(" ( %lf , %lf )", &Q(i, j).re, &Q(i, j).im);
            #else
                scanf(" ( %lf , %lf )", &Q(i, j).re, &Q(i, j).im);
            #endif
            #ifdef _WIN32
                scanf_s("%*[\n]");
            #else
                scanf("%*[\n]");
            #endif
}
if (wantz) {
    for (i = 1; i <= n; ++i)
        for (j = 1; j <= n; ++j)
            #ifdef _WIN32
                scanf_s(" ( %lf , %lf )", &Z(i, j).re, &Z(i, j).im);
            #else
                scanf(" ( %lf , %lf )", &Z(i, j).re, &Z(i, j).im);
            #endif
            #ifdef _WIN32
                scanf_s("%*[\n]");
            #else
                scanf("%*[\n]");
            #endif
}
/* Reorder the Schur factors S and T and update the matrices Q and Z. */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_ztgsen (f08yuc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Reorder the Schur factors S and T and update the matrices Q and Z. */
if (ijob==1 || ijob==4 || ijob == 5) {
    printf("For the selected eigenvalues, the reciprocals of projection 
        norms onto the deflating subspaces are\n"");
    printf(" for left subspace, pl = %11.2e for right subspace, pr = 
        %11.2e\n", pl, pr);
}
scanf("%*[\n]", &T(i, j).re, &T(i, j).im);
```

This code snippet is from the LAPACK library, specifically from the function `f08yuc` which deals with complex Schur decomposition and updating Schur factors. It includes handling of input/output for matrices and vectors, error checking, and printing of eigenvalues and deflating subspaces.
if (ijob>1) {
    printf(" upper bound on Difu = %11.2e\n", dif[0]);
    printf(" upper bound on Difl = %11.2e\n", dif[1]);
    if (ijob==2 || ijob==4) {
        printf("\nUpper bounds on Difl, Difu are based on the Frobenius norm\n");
    }
    if (ijob==3 || ijob==5) {
        printf("\nUpper bounds on Difl, Difu are based on the one norm.\n");
    }
}

END:
NAG_FREE(s);
NAG_FREE(t);
NAG_FREE(alpha);
NAG_FREE(beta);
NAG_FREE(select);
NAG_FREE(q);
NAG_FREE(z);
return exit_status;

10.2 Program Data

nag_ztgsen (f08yuc) Example Program Data

4 4 : n, ijob

Nag_TRUE : wantq
Nag_TRUE : wantz

Nag_FALSE Nag_TRUE Nag_TRUE Nag_FALSE : select

( 4.0, 4.0) ( 1.0, 1.0) ( 1.0, 1.0) ( 2.0,-1.0) : matrix S
( 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 T
( 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 Q
( 1.0, 0.0) ( 0.0, 0.0) ( 0.0, 0.0) ( 0.0, 0.0)
( 0.0, 0.0) ( 1.0, 0.0) ( 0.0, 0.0) ( 0.0, 0.0)
( 0.0, 0.0) ( 0.0, 0.0) ( 1.0, 0.0) ( 0.0, 0.0)
( 0.0, 0.0) ( 0.0, 0.0) ( 0.0, 0.0) ( 1.0, 0.0) : matrix Z

10.3 Program Results

nag_ztgsen (f08yuc) Example Program Results

Selected Eigenvalues
1 ( 2.0000e+00, 1.0000e+00)
2 ( 2.0000e+00, -1.0000e+00)

For the selected eigenvalues,
the reciprocals of projection norms onto the deflating subspaces are
for left subspace, pl = 1.12e-01
for right subspace, pr = 1.42e-01
upper bound on Difu = 2.18e-01
upper bound on Dif1 = 2.62e-01

Upper bounds on Dif1, Difu are based on the Frobenius norm.