NAG Library Function Document

nag_dtgsen (f08ygc)

1 Purpose

nag_dtgsen (f08ygc) reorders the generalized Schur factorization of a matrix pair in real generalized Schur form, so that a selected cluster of eigenvalues appears in the leading elements, or blocks 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

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

3 Description

nag_dtgsen (f08ygc) factorizes the generalized real n by n matrix pair (S,T) in real generalized Schur form, using an orthogonal equivalence transformation as

$$S = \hat{Q}\hat{S}\hat{Z}^{\mathrm{T}}, \quad T = \hat{Q}\hat{T}\hat{Z}^{\mathrm{T}},$$

where $\left(\hat{S},\hat{T}\right)$ are also in real generalized Schur form and have the selected eigenvalues as the leading diagonal elements, or diagonal blocks. 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 real generalized Schur form if S is block upper triangular with 1 by 1 and 2 by 2 diagonal blocks and T is upper triangular as returned, for example, by nag_dgges (f08xac), or nag_dhgeqz (f08xec) with **job** = Nag_Schur. The diagonal elements, or blocks, define the generalized eigenvalues (α_i,β_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 (α_i,β_i) in order to avoid possible overflow in computing λ_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, Dif_u and 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)

$$A = QSZ^{\mathrm{T}}, \quad B = QTZ^{\mathrm{T}}$$

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, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, 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-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: **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: $0 \leq$ **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 a real eigenvalue λ_j, **select**$[j-1]$ must be set to Nag_TRUE.

To select a complex conjugate pair of eigenvalues λ_j and λ_{j+1}, corresponding to a 2 by 2 diagonal block, either **select**$[j-1]$ or **select**$[j]$ or both must be set to Nag_TRUE; a complex conjugate pair of eigenvalues must be either both included in the cluster or both excluded.

6: **n** – Integer *Input*

On entry: n, the order of the matrices S and T.

Constraint: **n** ≥ 0.

7: **a**[dim] – double *Input/Output*

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

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

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

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

On exit: the updated matrix $\hat{S}$.

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

9: **b**[dim] – double *Input/Output*

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

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

On exit: the updated matrix $\hat{T}$

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: **alphar**[**n**] – double *Output*

On exit: see the description of **beta**.

12: **alphai**[**n**] – double *Output*

On exit: see the description of **beta**.

13: **beta**[**n**] – double *Output*

On exit: **alphar**[$j-1$]/**beta**[$j-1$] and **alphai**[$j-1$]/**beta**[$j-1$] are the real and imaginary parts respectively of the jth eigenvalue, for $j = 1, 2, \ldots,$ **n**.

If **alphai**[$j-1$] is zero, then the jth eigenvalue is real; if positive then **alphai**[j] is negative, and the jth and $(j+1)$st eigenvalues are a complex conjugate pair.

Conjugate pairs of eigenvalues correspond to the 2 by 2 diagonal blocks of $\hat{S}$. These 2 by 2 blocks can be reduced by applying complex unitary transformations to $\left(\hat{S}, \hat{T}\right)$ to obtain the complex Schur form $\left(\tilde{S}, \tilde{T}\right)$, where $\tilde{S}$ is triangular (and complex). In this form **alphar** $+ i$**alphai** and **beta** are the diagonals of $\tilde{S}$ and $\tilde{T}$ respectively.

14: **q**[dim] – double *Input/Output*

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

$\max(1, \mathbf{pdq} \times \mathbf{n})$ when **wantq** = Nag_TRUE;
1 otherwise.

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

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

On entry: if **wantq** = Nag_TRUE, the n by n matrix Q.

On exit: if **wantq** = Nag_TRUE, the updated matrix $Q\hat{Q}$.

If **wantq** = Nag_FALSE, **q** is not referenced.

15: **pdq** – Integer *Input*

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

Constraints:

if **wantq** = Nag_TRUE, **pdq** $\geq \max(1,$ **n**$)$;
otherwise **pdq** ≥ 1.

16: **z**$[dim]$ – double *Input/Output*

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

$\max(1,$ **pdz** $\times$ **n**$)$ when **wantz** = Nag_TRUE;
1 otherwise.

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

z$[(j-1) \times$ **pdz** $+ i - 1]$ when **order** = Nag_ColMajor;
z$[(i-1) \times$ **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 $Z\hat{Z}$.

If **wantz** = Nag_FALSE, **z** is not referenced.

17: **pdz** – Integer *Input*

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** $\geq \max(1,$ **n**$)$;
otherwise **pdz** ≥ 1.

18: **m** – Integer * *Output*

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

19: **pl** – double * *Output*
20: **pr** – double * *Output*

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 eigenspaces 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.

21: **dif**$[dim]$ – double *Output*

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.

22: **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_CONSTRAINT

On entry, **wantq** $=$ ⟨*value*⟩, **pdq** $=$ ⟨*value*⟩ and **n** $=$ ⟨*value*⟩.
Constraint: if **wantq** $=$ Nag_TRUE, **pdq** $\geq \max(1, \mathbf{n})$;
otherwise **pdq** ≥ 1.

On entry, **wantz** $=$ ⟨*value*⟩, **pdz** $=$ ⟨*value*⟩ and **n** $=$ ⟨*value*⟩.
Constraint: if **wantz** $=$ Nag_TRUE, **pdz** $\geq \max(1, \mathbf{n})$;
otherwise **pdz** ≥ 1.

NE_INT

On entry, **ijob** $=$ ⟨*value*⟩.
Constraint: $0 \leq$ **ijob** ≤ 5.

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

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

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

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

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

NE_INT_2

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

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

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.

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 **dif**$[0]$ and **dif**$[1]$, **pl** and **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 ϵ is the ***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

nag_dtgsen (f08ygc) is not threaded by NAG in any implementation.

nag_dtgsen (f08ygc) 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 complex analogue of this function is nag_ztgsen (f08yuc).

10 Example

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

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

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

selecting the first and fourth 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_dtgsen (f08ygc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

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

int main(void)
{

  /* Scalars */
  double        pl, pr, small;
  Integer       i, ijob, j, m, n, pdq, pds, pdt, pdz;
  Integer       exit_status = 0;

  /* Arrays */
  double        *alphai = 0, *alphar = 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 S(I, J) s[(J-1)*pds + I - 1]
#define T(I, J) t[(J-1)*pdt + I - 1]
#define Q(I, J) q[(J-1)*pdq + I - 1]
#define Z(I, J) z[(J-1)*pdz + I - 1]
  order = Nag_ColMajor;
#else
#define S(I, J) s[(I-1)*pds + J - 1]
#define T(I, J) t[(I-1)*pdt + J - 1]
#define Q(I, J) q[(I-1)*pdq + J - 1]
#define Z(I, J) z[(I-1)*pdz + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dtgsen (f08ygc) 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, double)) ||
      !(t      = NAG_ALLOC(n*n, double)) ||
      !(alphai = NAG_ALLOC(n, double)) ||
      !(alphar = NAG_ALLOC(n, double)) ||
      !(beta   = NAG_ALLOC(n, double)) ||
      !(select = NAG_ALLOC(n, Nag_Boolean)) ||
      !(q      = NAG_ALLOC(pdq*pdq, double)) ||
      !(z      = NAG_ALLOC(pdz*pdz, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  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
      select[i] = (Nag_Boolean) nag_enum_name_to_value(nag_enum_arg);
    }
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Read S, T, Q, Z and the logical array select from data file */
  for (i = 1; i <= n; ++i)
#ifdef _WIN32
    for (j = 1; j <= n; ++j) scanf_s("%lf", &S(i, j));
#else
    for (j = 1; j <= n; ++j) scanf("%lf", &S(i, j));
#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", &T(i, j));
```

```
#else
    for (j = 1; j <= n; ++j) scanf("%lf", &T(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  if (wantq) {
    for (i = 1; i <= n; ++i)
#ifdef _WIN32
      for (j = 1; j <= n; ++j) scanf_s("%lf", &Q(i, j));
#else
      for (j = 1; j <= n; ++j) scanf("%lf", &Q(i, j));
#endif
#ifdef _WIN32
    scanf_s("%*[^\n]");
#else
    scanf("%*[^\n]");
#endif
  }
  if (wantz) {
    for (i = 1; i <= n; ++i)
#ifdef _WIN32
      for (j = 1; j <= n; ++j) scanf_s("%lf", &Z(i, j));
#else
      for (j = 1; j <= n; ++j) scanf("%lf", &Z(i, j));
#endif
#ifdef _WIN32
    scanf_s("%*[^\n]");
#else
    scanf("%*[^\n]");
#endif
  }

  /* Reorder the Schur factors S and T and update the matrices Q and Z */
  nag_dtgsen(order, ijob, wantq, wantz, select, n, s, pds, t, pdt, alphar,
             alphai, beta, q, pdq, z, pdz, &m, &pl, &pr, dif, &fail);

  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dtgsen (f08ygc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* nag_real_safe_small_number (x02amc). */
  small = nag_real_safe_small_number;

  /* Print the eigenvalues */
  printf("Selected Eigenvalues\n");
  for (j = 0; j < m; ++j)
    {
      printf("%2"NAG_IFMT" ", j+1);
      if ((fabs(alphar[j]) + fabs(alphai[j])) * small >= fabs(beta[j]))
        printf("  infinite or undetermined, alpha = (%13.4e, %13.4e), "
               "beta = %13.4e\n", alphar[j], alphai[j], beta[j]);
      else if (alphai[j] == 0.0)
        printf("  %12.4e\n", alphar[j]/beta[j]);
      else
        printf("  (%13.4e, %13.4e)\n", alphar[j]/beta[j], alphai[j]/beta[j]);
    }

  if (ijob==1 || ijob==4 || ijob == 5) {
    printf("\n");
    printf("For the selected eigenvalues,\nthe reciprocals of projection "
           "norms onto the deflating subspaces are\n");
    printf(" for left  subspace, pl = %11.2e\n for right subspace, pr = "
           "%11.2e\n\n", pl, pr);
  }
```

```
  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(alphai);
  NAG_FREE(alphar);
  NAG_FREE(beta);
  NAG_FREE(select);
  NAG_FREE(q);
  NAG_FREE(z);

  return exit_status;
}
```

10.2 Program Data

```
nag_dtgsen (f08ygc) Example Program Data

  4    4                        : n and ijob

  Nag_TRUE                      : wantq
  Nag_TRUE                      : wantz

  Nag_TRUE  Nag_FALSE
  Nag_FALSE Nag_TRUE            : select[i], i = 0, n-1


  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            : S

  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            : T

  1.0  0.0  0.0  0.0
  0.0  1.0  0.0  0.0
  0.0  0.0  1.0  0.0
  0.0  0.0  0.0  1.0            : Q

  1.0  0.0  0.0  0.0
  0.0  1.0  0.0  0.0
  0.0  0.0  1.0  0.0
  0.0  0.0  0.0  1.0            : Z
```

10.3 Program Results

```
nag_dtgsen (f08ygc) Example Program Results

Selected Eigenvalues
 1     2.0000e+00
 2     3.0000e+00

For the selected eigenvalues,
the reciprocals of projection norms onto the deflating subspaces are
 for left  subspace, pl =    3.71e-01
 for right subspace, pr =    6.67e-01

 upper bound on Difu    =    2.52e-01
 upper bound on Difl    =    2.45e-01

Upper bounds on Difl, Difu are based on the Frobenius norm
```