NAG Library Function Document

nag_dtgsna (f08ylc)

1 Purpose

nag_dtgsna (f08ylc) estimates condition numbers for specified eigenvalues and/or eigenvectors of a matrix pair in generalized real Schur form.

2 Specification

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

```
void nag_dtgsna (Nag_OrderType order, Nag_JobType job,
     Nag_HowManyType howmny, const Nag_Boolean select[], Integer n,
     const double a[], Integer pda, const double b[], Integer pdb,
     const double vl[], Integer pdvl, const double vr[], Integer pdvr,
     double s[], double dif[], Integer mm, Integer *m, NagError *fail)
```

3 Description

nag_dtgsna (f08ylc) estimates condition numbers for specified eigenvalues and/or right eigenvectors of an n by n matrix pair (S,T) in real generalized Schur form. The function actually returns estimates of the reciprocals of the condition numbers in order to avoid possible overflow.

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_dggesx (f08xbc), or nag_dhgeqz (f08xec) with **job** $=$ Nag_Schur. The diagonal elements, or blocks, define the generalized eigenvalues (α_i,β_i), for $i=1,2,\dots,n$, of the pair (S,T) and the eigenvalues are given by

$$\lambda_i=\alpha_i/\beta_i,$$

so that

$$\beta_i S x_i=\alpha_i T x_i \quad \text{or} \quad S x_i=\lambda_i T x_i,$$

where x_i is the corresponding (right) eigenvector.

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 the eigenvalues and condition numbers of the pair (S,T) are the same as those of the pair (A,B).

Let $(\alpha,\beta)\neq(0,0)$ be a simple generalized eigenvalue of (A,B). Then the reciprocal of the condition number of the eigenvalue $\lambda=\alpha/\beta$ is defined as

$$s(\lambda)=\frac{\left(\left|y^{\mathrm{T}}Ax\right|^2+\left|y^{\mathrm{T}}Bx\right|^2\right)^{1/2}}{\left(\|x\|_2\|y\|_2\right)},$$

where x and y are the right and left eigenvectors of (A,B) corresponding to λ. If both α and β are zero, then (A,B) is singular and $s(\lambda)=-1$ is returned.

The definition of the reciprocal of the estimated condition number of the right eigenvector x and the left eigenvector y corresponding to the simple eigenvalue λ depends upon whether λ is a real eigenvalue, or one of a complex conjugate pair.

If the eigenvalue λ is real and U and V are orthogonal transformations such that

$$U^{\mathrm{T}}(A,B)V = (S,T) = \begin{pmatrix} \alpha & * \\ 0 & S_{22} \end{pmatrix}\begin{pmatrix} \beta & * \\ 0 & T_{22} \end{pmatrix},$$

where S_{22} and T_{22} are $(n-1)$ by $(n-1)$ matrices, then the reciprocal condition number is given by

$$\mathrm{Dif}(x) \equiv \mathrm{Dif}(y) = \mathrm{Dif}((\alpha,\beta),(S_{22},T_{22})) = \sigma_{\min}(Z),$$

where $\sigma_{\min}(Z)$ denotes the smallest singular value of the $2(n-1)$ by $2(n-1)$ matrix

$$Z = \begin{pmatrix} \alpha\otimes I & -1\otimes S_{22} \\ \beta\otimes I & -1\otimes T_{22} \end{pmatrix}$$

and $\otimes$ is the Kronecker product.

If λ is part of a complex conjugate pair and U and V are orthogonal transformations such that

$$U^{\mathrm{T}}(A,B)V = (S,T) = \begin{pmatrix} S_{11} & * \\ 0 & S_{22} \end{pmatrix}\begin{pmatrix} T_{11} & * \\ 0 & T_{22} \end{pmatrix},$$

where S_{11} and T_{11} are two by two matrices, S_{22} and T_{22} are $(n-2)$ by $(n-2)$ matrices, and (S_{11},T_{11}) corresponds to the complex conjugate eigenvalue pair λ, $\bar{\lambda}$, then there exist unitary matrices U_1 and V_1 such that

$$U_1^H S_{11} V_1 = \begin{pmatrix} s_{11} & s_{12} \\ 0 & s_{22} \end{pmatrix} \quad \text{and} \quad U_1^H T_{11} V_1 = \begin{pmatrix} t_{11} & t_{12} \\ 0 & t_{22} \end{pmatrix}.$$

The eigenvalues are given by $\lambda = s_{11}/t_{11}$ and $\bar{\lambda} = s_{22}/t_{22}$. Then the Frobenius norm-based, estimated reciprocal condition number is bounded by

$$\mathrm{Dif}(x) \equiv \mathrm{Dif}(y) \leq \min(d_1, \max(1, |\mathrm{Re}(s_{11})/\mathrm{Re}(s_{22})|), d_2)$$

where $\mathrm{Re}(z)$ denotes the real part of z, $d_1 = \mathrm{Dif}((s_{11},t_{11}),(s_{22},t_{22})) = \sigma_{\min}(Z_1)$, Z_1 is the complex two by two matrix

$$Z_1 = \begin{pmatrix} s_{11} & -s_{22} \\ t_{11} & -t_{22} \end{pmatrix},$$

and d_2 is an upper bound on $\mathrm{Dif}((S_{11},T_{11}),(S_{22},T_{22}))$; i.e., an upper bound on $\sigma_{\min}(Z_2)$, where Z_2 is the $(2n-2)$ by $(2n-2)$ matrix

$$Z_2 = \begin{pmatrix} S_{11}^T\otimes I & -I\otimes S_{22} \\ T_{11}^T\otimes I & -I\otimes T_{22} \end{pmatrix}.$$

See Sections 2.4.8 and 4.11 of Anderson *et al.* (1999) and Kågström and Poromaa (1996) for further details and information.

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 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: **job** – Nag_JobType *Input*

On entry: indicates whether condition numbers are required for eigenvalues and/or eigenvectors.

job = Nag_EigVals
Condition numbers for eigenvalues only are computed.

job = Nag_EigVecs
Condition numbers for eigenvectors only are computed.

job = Nag_DoBoth
Condition numbers for both eigenvalues and eigenvectors are computed.

Constraint: **job** = Nag_EigVals, Nag_EigVecs or Nag_DoBoth.

3: **howmny** – Nag_HowManyType *Input*

On entry: indicates how many condition numbers are to be computed.

howmny = Nag_ComputeAll
Condition numbers for all eigenpairs are computed.

howmny = Nag_ComputeSelected
Condition numbers for selected eigenpairs (as specified by **select**) are computed.

Constraint: **howmny** = Nag_ComputeAll or Nag_ComputeSelected.

4: **select**[dim] – const Nag_Boolean *Input*

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

n when **howmny** = Nag_ComputeSelected;
otherwise **select** may be **NULL**.

On entry: specifies the eigenpairs for which condition numbers are to be computed if **howmny** = Nag_ComputeSelected. To select condition numbers for the eigenpair corresponding to the real eigenvalue λ_j, **select**[$j-1$] must be set Nag_TRUE. To select condition numbers corresponding to a complex conjugate pair of eigenvalues λ_j and λ_{j+1}, **select**[$j-1$] and/or **select**[j] must be set to Nag_TRUE.

If **howmny** = Nag_ComputeAll, **select** is not referenced and may be **NULL**.

5: **n** – Integer *Input*

On entry: n, the order of the matrix pair (S,T).

Constraint: **n** ≥ 0.

6: **a**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **a** must be at least **pda** $\times$ **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 upper quasi-triangular matrix S.

7: **pda** – Integer *Input*

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

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

8: **b**$[dim]$ – const double *Input*

Note: the dimension, *dim*, of the array **b** must be at least **pdb** $\times$ **n**.

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

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

On entry: the upper triangular matrix T.

9: **pdb** – Integer *Input*

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

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

10: **vl**$[dim]$ – const double *Input*

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

pdvl $\times$ **mm** when **job** = Nag_EigVals or Nag_DoBoth and **order** = Nag_ColMajor;
n $\times$ **pdvl** when **job** = Nag_EigVals or Nag_DoBoth and **order** = Nag_RowMajor;
otherwise **vl** may be **NULL**.

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

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

On entry: if **job** = Nag_EigVals or Nag_DoBoth, **vl** must contain left eigenvectors of (S, T), corresponding to the eigenpairs specified by **howmny** and **select**. The eigenvectors must be stored in consecutive columns of **vl**, as returned by nag_dggev (f08wac) or nag_dtgevc (f08ykc).

If **job** = Nag_EigVecs, **vl** is not referenced and may be **NULL**.

11: **pdvl** – Integer *Input*

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

Constraints:

if **order** = Nag_ColMajor,

if **job** = Nag_EigVals or Nag_DoBoth, **pdvl** $\geq$ **n**;
otherwise **pdvl** ≥ 1.;

if **order** = Nag_RowMajor,

if **job** = Nag_EigVals or Nag_DoBoth, **pdvl** $\geq$ **mm**;
otherwise **vl** may be **NULL**..

12: **vr**$[dim]$ – const double *Input*

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

pdvr $\times$ **mm** when **job** = Nag_EigVals or Nag_DoBoth and **order** = Nag_ColMajor;
n $\times$ **pdvr** when **job** = Nag_EigVals or Nag_DoBoth and **order** = Nag_RowMajor;
otherwise **vr** may be **NULL**.

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

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

On entry: if **job** = Nag_EigVals or Nag_DoBoth, **vr** must contain right eigenvectors of (S, T), corresponding to the eigenpairs specified by **howmny** and **select**. The eigenvectors must be stored in consecutive columns of **vr**, as returned by nag_dggev (f08wac) or nag_dtgevc (f08ykc).

If **job** = Nag_EigVecs, **vr** is not referenced and may be **NULL**.

13: **pdvr** – Integer *Input*

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

Constraints:

> if **order** = Nag_ColMajor,
>
> > if **job** = Nag_EigVals or Nag_DoBoth, **pdvr** $\geq$ **n**;
> > otherwise **pdvr** ≥ 1.;
>
> if **order** = Nag_RowMajor,
>
> > if **job** = Nag_EigVals or Nag_DoBoth, **pdvr** $\geq$ **mm**;
> > otherwise **vr** may be **NULL**..

14: $\mathbf{s}[dim]$ – double *Output*

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

> **mm** when **job** = Nag_EigVals or Nag_DoBoth;
> otherwise **s** may be **NULL**.

On exit: if **job** = Nag_EigVals or Nag_DoBoth, the reciprocal condition numbers of the selected eigenvalues, stored in consecutive elements of the array. For a complex conjugate pair of eigenvalues two consecutive elements of **s** are set to the same value. Thus $\mathbf{s}[j-1]$, $\mathbf{dif}[j-1]$, and the jth columns of VL and VR all correspond to the same eigenpair (but not in general the jth eigenpair, unless all eigenpairs are selected).

If **job** = Nag_EigVecs, **s** is not referenced and may be **NULL**.

15: $\mathbf{dif}[dim]$ – double *Output*

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

> **mm** when **job** = Nag_EigVecs or Nag_DoBoth;
> otherwise **dif** may be **NULL**.

On exit: if **job** = Nag_EigVecs or Nag_DoBoth, the estimated reciprocal condition numbers of the selected eigenvectors, stored in consecutive elements of the array. For a complex eigenvector two consecutive elements of **dif** are set to the same value. If the eigenvalues cannot be reordered to compute $\mathbf{dif}[j-1]$, $\mathbf{dif}[j-1]$ is set to 0; this can only occur when the true value would be very small anyway.

If **job** = Nag_EigVals, **dif** is not referenced and may be **NULL**.

16: **mm** – Integer *Input*

On entry: the number of elements in the arrays **s** and **dif**.

Constraints:

> if **howmny** = Nag_ComputeAll, **mm** $\geq$ **n**;
> otherwise **mm** $\geq$ **m**.

17: **m** – Integer * *Output*

On exit: the number of elements of the arrays **s** and **dif** used to store the specified condition numbers; for each selected real eigenvalue one element is used, and for each selected complex conjugate pair of eigenvalues, two elements are used. If **howmny** = Nag_ComputeAll, **m** is set to **n**.

18: **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_ENUM_INT_2

On entry, **job** $= \langle value\rangle$, **pdvl** $= \langle value\rangle$, **mm** $= \langle value\rangle$.
Constraint: if **job** = Nag_EigVals or Nag_DoBoth, **pdvl** $\geq$ **mm**.

On entry, **job** $= \langle value\rangle$, **pdvl** $= \langle value\rangle$ and **n** $= \langle value\rangle$.
Constraint: if **job** = Nag_EigVals or Nag_DoBoth, **pdvl** $\geq$ **n**.

On entry, **job** $= \langle value\rangle$, **pdvr** $= \langle value\rangle$, **mm** $= \langle value\rangle$.
Constraint: if **job** = Nag_EigVals or Nag_DoBoth, **pdvr** $\geq$ **mm**.

On entry, **job** $= \langle value\rangle$, **pdvr** $= \langle value\rangle$ and **n** $= \langle value\rangle$.
Constraint: if **job** = Nag_EigVals or Nag_DoBoth, **pdvr** $\geq$ **n**.

NE_ENUM_INT_3

On entry, **howmny** $= \langle value\rangle$, **n** $= \langle value\rangle$, **mm** $= \langle value\rangle$ and **m** $= \langle value\rangle$.
Constraint: if **howmny** = Nag_ComputeAll, **mm** $\geq$ **n**;
otherwise **mm** $\geq$ **m**.

NE_INT

On entry, **n** $= \langle value\rangle$.
Constraint: **n** ≥ 0.

On entry, **pda** $= \langle value\rangle$.
Constraint: **pda** > 0.

On entry, **pdb** $= \langle value\rangle$.
Constraint: **pdb** > 0.

On entry, **pdvl** $= \langle value\rangle$.
Constraint: **pdvl** > 0.

On entry, **pdvr** $= \langle value\rangle$.
Constraint: **pdvr** > 0.

NE_INT_2

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

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

7 Accuracy

None.

8 Parallelism and Performance

nag_dtgsna (f08ylc) is not threaded by NAG in any implementation.

nag_dtgsna (f08ylc) 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

An approximate asymptotic error bound on the chordal distance between the computed eigenvalue $\tilde{\lambda}$ and the corresponding exact eigenvalue λ is

$$\chi(\tilde{\lambda}, \lambda) \leq \epsilon \|(A, B)\|_F / S(\lambda)$$

where ϵ is the ***machine precision***.

An approximate asymptotic error bound for the right or left computed eigenvectors $\tilde{x}$ or $\tilde{y}$ corresponding to the right and left eigenvectors x and y is given by

$$\theta(\tilde{z}, z) \leq \epsilon \|(A, B)\|_F / \mathrm{Dif}.$$

The complex analogue of this function is nag_ztgsna (f08yyc).

10 Example

This example estimates condition numbers and approximate error estimates for all the eigenvalues and eigenvalues and right eigenvectors of the pair (S, T) given by

$$S = \begin{pmatrix} 4.0 & 1.0 & 1.0 & 2.0 \\ 0 & 3.0 & -1.0 & 1.0 \\ 0 & 1.0 & 3.0 & 1.0 \\ 0 & 0 & 0 & 6.0 \end{pmatrix} \quad \text{and} \quad T = \begin{pmatrix} 2.0 & 1.0 & 1.0 & 3.0 \\ 0 & 1.0 & 0.0 & 1.0 \\ 0 & 0 & 1.0 & 1.0 \\ 0 & 0 & 0 & 2.0 \end{pmatrix}.$$

The eigenvalues and eigenvectors are computed by calling nag_dtgevc (f08ykc).

10.1 Program Text

```
/* nag_dtgsna (f08ylc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

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

int main(void)
{
  /* Scalars */
  double        eps, snorm, stnrm, tnorm, tol;
  Integer       i, j, m, n, pds, pdt, pdvl, pdvr;
  Integer       exit_status = 0;

  /* Arrays */
  double        *dif = 0, *s = 0, *scon = 0, *t = 0, *vl = 0, *vr = 0;

  /* Nag Types */
  NagError      fail;
  Nag_OrderType order;

#ifdef NAG_COLUMN_MAJOR
#define S(I, J) s[(J-1)*pds + I - 1]
#define T(I, J) t[(J-1)*pdt + 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]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dtgsna (f08ylc) 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"%*[^\n]", &n);
#else
  scanf("%"NAG_IFMT"%*[^\n]", &n);
#endif
  if (n < 0)
    {
      printf("Invalid n\n");
      exit_status = 1;
      goto END;
    }
  m = n;
  pds = n;
  pdt = n;
  pdvl = n;
  pdvr = n;

  /* Allocate memory */
  if (!(dif  = NAG_ALLOC(n, double)) ||
      !(scon = NAG_ALLOC(n, double)) ||
```

```
      !(s    = NAG_ALLOC(n*n, double)) ||
      !(t    = NAG_ALLOC(n*n, double)) ||
      !(vl   = NAG_ALLOC(n*m, double)) ||
      !(vr   = NAG_ALLOC(n*m, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read S and T 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

  /* Calculate the left and right generalized eigenvectors of the
   * matrix pair (S,T) using nag_dtgevc (f08ykc).
   * NULL may be passed here in place of the select array since all
   * eigenvectors are requested.
   */
  nag_dtgevc(order, Nag_BothSides, Nag_ComputeAll, NULL, n, s, pds, t, pdt,
             vl, pdvl, vr, pdvr, n, &m, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dtgevc (f08ykc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Estimate condition numbers for all the generalized eigenvalues and right
   * eigenvectors of the pair (S,T) using nag_dtgsna (f08ylc).
   * NULL may be passed here in place of the select array since all
   * eigenvectors are requested.
   */
  nag_dtgsna(order, Nag_DoBoth, Nag_ComputeAll, NULL, n, s, pds, t, pdt,
             vl, pdvl, vr, pdvr, scon, dif, n, &m, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dtgsna (f08ylc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print condition numbers of eigenvalues and right eigenvectors */
  printf("Condition numbers of eigenvalues (scon) and right eigenvectors "
         "(diff),\n");
  printf("scon:   ");
  for (i = 0; i < m; ++i)
    printf(" %10.1e%s", scon[i], i%7 == 6?"\n        ":"");
  printf("\ndif:    ");
  for (i = 0; i < m; ++i)
    printf(" %10.1e%s", dif[i], i%7 == 6?"\n        ":"");
```

```
  /* Compute the norm of (S,T) using nag_dge_norm (f16rac). */
  eps = nag_machine_precision;
  nag_dge_norm(order, Nag_OneNorm, n, n, s, pds, &snorm, &fail);
  nag_dge_norm(order, Nag_OneNorm, n, n, t, pdt, &tnorm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  if (snorm == 0.0)
    stnrm = ABS(tnorm);
  else if (tnorm == 0.0)
    stnrm = ABS(snorm);
  else if (ABS(snorm) >= ABS(tnorm))
    stnrm = ABS(snorm)*sqrt(1.0+(tnorm/snorm)*(tnorm/snorm));
  else
    stnrm = ABS(tnorm)*sqrt(1.0+(snorm/tnorm)*(snorm/tnorm));

  /* Calculate approximate error estimates */
  tol = eps*stnrm;

  printf("\n\nError estimates for eigenvalues (errval) and right eigenvectors"
         " (errvec),\n");
  printf("errval: ");
  for (i = 0; i < m; ++i)
    printf(" %10.1e%s", tol/scon[i], i%7 == 6?"\n        ":"");
  printf("\nerrvec: ");
  for (i = 0; i < m; ++i)
    printf(" %10.1e%s", tol/dif[i], i%7 == 6?"\n        ":"");

 END:
  NAG_FREE(dif);
  NAG_FREE(scon);
  NAG_FREE(s);
  NAG_FREE(t);
  NAG_FREE(vl);
  NAG_FREE(vr);

  return exit_status;
}
```

10.2 Program Data

```
nag_dtgsna (f08ylc) Example Program Data

  4                    : n

  4.0  1.0  1.0  2.0
  0.0  3.0 -1.0  1.0
  0.0  1.0  3.0  1.0
  0.0  0.0  0.0  6.0   : matrix S

  2.0  1.0  1.0  3.0
  0.0  1.0  0.0  1.0
  0.0  0.0  1.0  1.0
  0.0  0.0  0.0  2.0   : matrix T
```

10.3 Program Results

```
nag_dtgsna (f08ylc) Example Program Results

Condition numbers of eigenvalues (scon) and right eigenvectors (diff),
scon:        1.6e+00    1.7e+00    1.7e+00    1.4e+00
dif:         5.4e-01    1.5e-01    1.5e-01    1.2e-01

Error estimates for eigenvalues (errval) and right eigenvectors (errvec),
errval:      8.7e-16    7.8e-16    7.8e-16    9.9e-16
errvec:      2.5e-15    9.0e-15    9.0e-15    1.1e-14
```