NAG Library Function Document
nag_dgges (f08xac)

1 Purpose
nag_dgges (f08xac) computes the generalized eigenvalues, the generalized real Schur form \((S,T)\) and, optionally, the left and/or right generalized Schur vectors for a pair of \(n\) by \(n\) real nonsymmetric matrices \((A, B)\).

2 Specification

```c
#include <nag.h>
#include <nagf08.h>
void nag_dgges (Nag_OrderType order, Nag_LeftVecsType jobvsl,
        Nag_RightVecsType jobvsr, Nag_SortEigValsType sort,
        Nag_Boolean (*selctg)(double ar, double ai, double b),
        Integer n, double a[], Integer pda, double b[],
        Integer pdb, Integer *sdim, double alphar[], double alphai[],
        double beta[], double vsl[], Integer pdvsl, double vsr[],
        Integer pdvsr, NagError *fail)
```

3 Description

The generalized Schur factorization for a pair of real matrices \((A, B)\) is given by

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

where \(Q\) and \(Z\) are orthogonal, \(T\) is upper triangular and \(S\) is upper quasi-triangular with 1 by 1 and 2 by 2 diagonal blocks. The generalized eigenvalues, \(\lambda\), of \((A, B)\) are computed from the diagonals of \(S\) and \(T\) and satisfy

\[
Az = \lambda Bz,
\]

where \(z\) is the corresponding generalized eigenvector. \(\lambda\) is actually returned as the pair \((\alpha, \beta)\) such that

\[
\lambda = \alpha / \beta
\]

since \(\beta\), or even both \(\alpha\) and \(\beta\) can be zero. The columns of \(Q\) and \(Z\) are the left and right generalized Schur vectors of \((A, B)\).

Optionally, nag_dgges (f08xac) can order the generalized eigenvalues on the diagonals of \((S, T)\) so that selected eigenvalues are at the top left. The leading columns of \(Q\) and \(Z\) then form an orthonormal basis for the corresponding eigenspaces, the deflating subspaces.

nag_dgges (f08xac) computes \(T\) to have non-negative diagonal elements, and the 2 by 2 blocks of \(S\) correspond to complex conjugate pairs of generalized eigenvalues. The generalized Schur factorization, before reordering, is computed by the QZ algorithm.

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: jobvsl – Nag_LeftVecsType

*Input*

*On entry:* if jobvsl = Nag_NotLeftVecs, do not compute the left Schur vectors.
If jobvsl = Nag_LeftVecs, compute the left Schur vectors.

*Constraint:* jobvsl = Nag_NotLeftVecs or Nag_LeftVecs.

3: jobvsr – Nag_RightVecsType

*Input*

*On entry:* if jobvsr = Nag_NotRightVecs, do not compute the right Schur vectors.
If jobvsr = Nag_RightVecs, compute the right Schur vectors.

*Constraint:* jobvsr = Nag_NotRightVecs or Nag_RightVecs.

4: sort – Nag_SortEigValsType

*Input*

*On entry:* specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form.

sort = Nag_NoSortEigVals
Eigenvalues are not ordered.

sort = Nag_SortEigVals
Eigenvalues are ordered (see selctg).

*Constraint:* sort = Nag_NoSortEigVals or Nag_SortEigVals.

5: selctg – function, supplied by the user

*External Function*

If sort = Nag_SortEigVals, selctg is used to select generalized eigenvalues to the top left of the generalized Schur form.

If sort = Nag_NoSortEigVals, selctg is not referenced by nag_dgges (f08xac), and may be specified as NULLFN.

The specification of selctg is:

Nag_Boolean selctg (double ar, double ai, double b)

1: ar – double

2: ai – double

3: b – double

*Input*

*On entry:* an eigenvalue \((ar[j-1] + \sqrt{-1} \times ai[j-1])/b[j-1]\) is selected if selctg\((ar[j-1], ai[j-1], b[j-1]) = \text{Nag\_TRUE}\). If either one of a complex conjugate pair is selected, then both complex generalized eigenvalues are selected.

Note that in the ill-conditioned case, a selected complex generalized eigenvalue may no longer satisfy selctg\((ar[j-1], ai[j-1], b[j-1]) = \text{Nag\_TRUE}\) after ordering.

fail.code = NE_SCHUR_REORDER_SELECT in this case.
6:   n – Integer

On entry: $n$, the order of the matrices $A$ and $B$.

Constraint: $n \geq 0$.

7:   a[dim] – double

Input/Output

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

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

$a[(j-1) \times pda + i - 1]$ when $\text{order} = \text{Nag\_ColMajor};$

$a[(i-1) \times pda + j - 1]$ when $\text{order} = \text{Nag\_RowMajor}.$

On entry: the first of the pair of matrices, $A$.
On exit: $a$ has been overwritten by its generalized Schur form $S$.

8:   pda – Integer

Input

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

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

9:   b[dim] – double

Input/Output

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

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

$b[(j-1) \times pdb + i - 1]$ when $\text{order} = \text{Nag\_ColMajor};$

$b[(i-1) \times pdb + j - 1]$ when $\text{order} = \text{Nag\_RowMajor}.$

On entry: the second of the pair of matrices, $B$.
On exit: $b$ has been overwritten by its generalized Schur form $T$.

10:   pdb – Integer

Input

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

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

11:   sdim – Integer *

Output

On exit: if $\text{sort} = \text{Nag\_NoSortEigVals}$, $sdim = 0$.

If $\text{sort} = \text{Nag\_SortEigVals}$, $sdim = \text{number of eigenvalues (after sorting) for which } selctg \text{ is Nag\_TRUE. (Complex conjugate pairs for which } selctg \text{ is Nag\_TRUE for either eigenvalue count as 2.)}$

12:   alphar[n] – double

Output

On exit: see the description of $\beta$.

13:   alphai[n] – double

Output

On exit: see the description of $\beta$.

14:   beta[n] – double

Output

On exit: $[\frac{\text{alphar}[j-1] + \text{alphai}[j-1] \times i}{\beta[j-1]}$, for $j = 1, 2, \ldots, n$, will be the generalized eigenvalues. $\text{alphar}[j-1] + \text{alphai}[j-1] \times i$, and $\text{beta}[j-1]$, for $j = 1, 2, \ldots, n$, are the diagonals of the complex Schur form $(S,T)$ that would result if the 2 by 2 diagonal blocks of
the real Schur form of \((A, B)\) were further reduced to triangular form using 2 by 2 complex unitary transformations.

If \(\text{alphai}[j-1]\) is zero, then the \(j\)th eigenvalue is real; if positive, then the \(j\)th and \((j+1)\)st eigenvalues are a complex conjugate pair, with \(\text{alphai}[j]\) negative.

**Note:** the quotients \(\text{alphar}[j-1]/\text{beta}[j-1]\) and \(\text{alphai}[j-1]/\text{beta}[j-1]\) may easily overflow or underflow, and \(\text{beta}[j-1]\) may even be zero. Thus, you should avoid naively computing the ratio \(\alpha/\beta\). However, \(\text{alphar}\) and \(\text{alphai}\) will always be less than and usually comparable with \(\|a\|_2\) in magnitude, and \(\text{beta}\) will always be less than and usually comparable with \(\|b\|_2\).

15: \(\text{vsl}[\text{dim}]\) – double

**Output**

*Note:* the dimension, \(\text{dim}\), of the array \(\text{vsl}\) must be at least

\[\max(1, \text{pdvsl} \times n)\] when \(\text{jobvsl} = \text{Nag LeftVecs}\);

1 otherwise.

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

\[\text{vsl}[(j-1) \times \text{pdvsl} + i - 1]\] when \(\text{order} = \text{Nag ColMajor}\);

\[\text{vsl}[(i-1) \times \text{pdvsl} + j - 1]\] when \(\text{order} = \text{Nag RowMajor}\).

**On exit:** if \(\text{jobvsl} = \text{Nag LeftVecs}, \text{vsl}\) will contain the left Schur vectors, \(Q\).

If \(\text{jobvsl} = \text{Nag NotLeftVecs}, \text{vsl}\) is not referenced.

16: \(\text{pdvsl}\) – Integer

**Input**

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

**Constraints:**

\[\text{if } \text{jobvsl} = \text{Nag LeftVecs}, \text{pdvsl} \geq \max(1, n);\]

otherwise \(\text{pdvsl} \geq 1\).

17: \(\text{vsr}[\text{dim}]\) – double

**Output**

*Note:* the dimension, \(\text{dim}\), of the array \(\text{vsr}\) must be at least

\[\max(1, \text{pdvsr} \times n)\] when \(\text{jobvsr} = \text{Nag RightVecs}\);

1 otherwise.

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

\[\text{vsr}[(j-1) \times \text{pdvsr} + i - 1]\] when \(\text{order} = \text{Nag ColMajor}\);

\[\text{vsr}[(i-1) \times \text{pdvsr} + j - 1]\] when \(\text{order} = \text{Nag RowMajor}\).

**On exit:** if \(\text{jobvsr} = \text{Nag RightVecs}, \text{vsr}\) will contain the right Schur vectors, \(Z\).

If \(\text{jobvsr} = \text{Nag NotRightVecs}, \text{vsr}\) is not referenced.

18: \(\text{pdvsr}\) – Integer

**Input**

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

**Constraints:**

\[\text{if } \text{jobvsr} = \text{Nag RightVecs}, \text{pdvsr} \geq \max(1, n);\]

otherwise \(\text{pdvsr} \geq 1\).

19: \(\text{fail}\) – 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_ENUM_INT_2
On entry, jobvsl = ⟨value⟩, pdvsl = ⟨value⟩ and n = ⟨value⟩.
Constraint: if jobvsl = Nag_LeftVecs, pdvsl ≥ max(1, n);
otherwise pdvsl ≥ 1.

On entry, jobvsr = ⟨value⟩, pdvsr = ⟨value⟩ and n = ⟨value⟩.
Constraint: if jobvsr = Nag_RightVecs, pdvsr ≥ max(1, n);
otherwise pdvsr ≥ 1.

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

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

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

On entry, pdvsl = ⟨value⟩.
Constraint: pdvsl > 0.

On entry, pdvsr = ⟨value⟩.
Constraint: pdvsr > 0.

NE_INT_2
On entry, pda = ⟨value⟩ and n = ⟨value⟩.
Constraint: pda ≥ max(1, n).

On entry, pdb = ⟨value⟩ and n = ⟨value⟩.
Constraint: pdb ≥ 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_ITERATION_QZ
The QZ iteration failed. No eigenvectors have been calculated but alphar[j], alphai[j] and beta[j]
should be correct from element ⟨value⟩.

The QZ iteration failed with an unexpected error, please contact NAG.

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_REORDER

The eigenvalues could not be reordered because some eigenvalues were too close to separate (the problem is very ill-conditioned).

NE_SCHUR_REORDER_SELECT

After reordering, roundoff changed values of some complex eigenvalues so that leading eigenvalues in the generalized Schur form no longer satisfy selectg = Nag_TRUE. This could also be caused by underflow due to scaling.

7 Accuracy

The computed generalized Schur factorization satisfies

\[ A + E = QSZ^T, \quad B + F = QTZ^T, \]

where

\[ \|(E, F)\|_F = O(\epsilon)\|(A, B)\|_F \]

and \( \epsilon \) is the machine precision. See Section 4.11 of Anderson et al. (1999) for further details.

8 Parallelism and Performance

nag_dgges (f08xac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dgges (f08xac) 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 proportional to \( n^3 \).

The complex analogue of this function is nag_zgges (f08xnc).

10 Example

This example finds the generalized Schur factorization of the matrix pair \((A, B)\), where

\[
A = \begin{pmatrix}
3.9 & 12.5 & -34.5 & -0.5 \\
4.3 & 21.5 & -47.5 & 7.5 \\
4.3 & 21.5 & -43.5 & 3.5 \\
4.4 & 26.0 & -46.0 & 6.0
\end{pmatrix}
\quad \text{and} \quad
B = \begin{pmatrix}
1.0 & 2.0 & -3.0 & 1.0 \\
1.0 & 3.0 & -5.0 & 4.0 \\
1.0 & 3.0 & -4.0 & 3.0 \\
1.0 & 3.0 & -4.0 & 4.0
\end{pmatrix},
\]

such that the real positive eigenvalues of \((A, B)\) correspond to the top left diagonal elements of the generalized Schur form, \((S, T)\).

10.1 Program Text

/* nag_dgges (f08xac) Example Program. * * Copyright 2014 Numerical Algorithms Group. * * Mark 25, 2014. */
#include <stdio.h>

f08xac.6

f08xac

NAG Library Manual

Mark 25
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx02.h>
#include <nagx04.h>

#ifdef __cplusplus
extern "C" {
#endif
static Nag_Boolean NAG_CALL selctg(const double ar, const double ai, const double b);
#ifdef __cplusplus
}
#endif

int main(void) {

/* Scalars */
double dg_a, dg_b, eps, norma, normb, normd, norme;
Integer i, j, n, sdim, pda, pdb, pdc, pdd, pde, pdvsl, pdvsr;
Integer exit_status = 0;

/* Arrays */
double *a = 0, *alphai = 0, *alphar = 0, *b = 0, *beta = 0;
double *c = 0, *d = 0, *e = 0, *vsl = 0, *vsr = 0;
char nag_enum_arg[40];

/* Nag Types */
NagError fail;
Nag_OrderType order;
Nag_LeftVecsType jobvsl;
Nag_RightVecsType jobvsr;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda +I-1]
#define B(I, J) b[(J-1)*pdb +I-1]
#else
#define A(I, J) a[(I-1)*pda+J-1]
#define B(I, J) b[(I-1)*pdb +J-1]
#endif

INIT_FAIL(fail);
printf("nag_dgges (f08xac) Example Program Results

");
#ifdef _WIN32
scanf_s("%*\[^
"]");
#else
scanf("%*\[^
"]");
#endif
if (n < 0) {
    printf("Invalid n\n");
    exit_status = 1;
    return exit_status;
}
#ifdef _WIN32
scanf_s("%39s%*\[^
"]", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf(" %39s%*\[^
"]", nag_enum_arg);

Mark 25

f08xac.7

f08 – Least-squares and Eigenvalue Problems (LAPACK)

f08xac
#ifndef _WIN32
scans(_%39s*_\n", nag_enum_arg, _countof(nag_enum_arg));
#else
scans(_%39s*_\n", nag_enum_arg);
#endif
jobvsr = (Nag_RightVecsType) nag_enum_name_to_value(nag_enum_arg);

pdvsl = (jobvsl==Nag_LeftVecs?) n:1);  
pdvsr = (jobvsr==Nag_RightVecs?) n:1);  
pda = n;  
pdb = n;  
pdc = n;  
pdd = n;  
pde = n;
/* Allocate memory */
if (! (a = NAG_ALLOC(n * n, double)) ||
!(b = NAG_ALLOC(n * n, double)) ||
!(c = NAG_ALLOC(n * n, double)) ||
!(d = NAG_ALLOC(n * n, double)) ||
!(e = NAG_ALLOC(n * n, double)) ||
!(alphai = NAG_ALLOC(n, double)) ||
!(alphar = NAG_ALLOC(n, double)) ||
!(beta = NAG_ALLOC(n, double)) ||
!(vsl = NAG_ALLOC(pdvsl*pdvsl, double)) ||
!(vsr = NAG_ALLOC(pdvsr*pdvsr, double)))
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}
/* Read in the matrices A and B */
for (i = 1; i <= n; ++i)
#endif _WIN32
for (j = 1; j <= n; ++j) scans(_%lf", &A(i, j));
#else
for (j = 1; j <= n; ++j) scans(_%lf", &A(i, j));
#endif _WIN32
scans(_%*[\n"]");
#else
scans(_%*[\n"]");
#endif _WIN32
scans(_%*[\n"]");
#else
scans(_%*[\n"]");
#endif _WIN32
scans(_%*[\n"]");
#endif _WIN32
scans(_%*[\n"]");
/* Copy matrices A and B to matrices D and E using nag_dge_copy (f16qfc),
 * real valued general matrix copy.
 * The copies will be used as comparison against reconstructed matrices.
 */
nag_dge_copy(order, Nag_NoTrans, n, n, a, pda, d, pdd, &fail);
if (fail.code != NE_NOERROR)
{
printf("Error from nag_dge_copy (f16qfc)\n", fail.message);
exit_status = 1;
goto END;
}
nag_dge_copy(order, Nag_NoTrans, n, n, b, pdb, e, pde, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dge_copy (f16qfc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_dge_norm (f16rac): Find norms of input matrices A and B. */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

nag_dge_norm(order, Nag_OneNorm, n, n, a, pda, &norma, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

nag_dge_norm(order, Nag_OneNorm, n, n, b, pdb, &normb, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_gen_real_mat_print (x04cac): Print Matrices A and B. */
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
a, pda, "Matrix A", 0, &fail);
printf("\n");
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
b, pdb, "Matrix B", 0, &fail);
printf("\n");
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Find the generalized Schur form using nag_dgges (f08xac). */
nag_dgges(order, jobvsl, jobvsr, Nag_SortEigVals, selctg, n, a, pda, b, pdb,
&sdim, alphar, alphai, beta, vsl, pdvsl, vsr, pdvsr, &fail);

if (fail.code != NE_NOERROR && fail.code != NE_SCHUR_REORDER_SELECT)
{
    printf("Error from nag_dgges (f08xac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Check generalized Schur Form by reconstruction of Schur vectors are
available. */
if (jobvsl==Nag_NotLeftVecs || jobvsr==Nag_NotRightVecs)
{
    /* Cannot check factorization by reconstruction Schur vectors. */
    goto END;
}

/* Reconstruct A as Q*S*Z^T and subtract from original (D) using the steps
C = Q*S (Q in vsl, S in a) using nag_dgemm (f16yac).
Note: not nag_dtrmm since S may not be strictly triangular.
D = D - C*Z^T (Z in vsr) using nag_dgemm (f16yac). */
dg_a = 1.0;
dg_b = 0.0;
nag_dgemm(order, Nag_NoTrans, Nag_NoTrans, n, n, n, dg_a, vsl, pdvsl, a, pda, 
dg_b, c, pdc, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgemm (f16yac).\n%s\n", fail.message);
    exit_status = 1;
goto END;
}
dg_a = -1.0;
dg_b = 1.0;
nag_dgemm(order, Nag_NoTrans, Nag_Trans, n, n, n, dg_a, c, pdc, vsr, pdvsr, 
dg_b, d, pdd, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgemm (f16yac).\n%s\n", fail.message);
    exit_status = 1;
goto END;
}
/* Reconstruct B as Q*T*Z^T and subtract from original (E) using the steps
 * C = Q*T (Q in vsl, T in b) using nag_dgemm (f16yac).
 * E = E - C*Z^T (Z in vsr) using nag_dgemm (f16yac).
 */
dg_a = 1.0;
dg_b = 0.0;
nag_dgemm(order, Nag_NoTrans, Nag_NoTrans, n, n, n, dg_a, vsl, pdvsl, b, pdb, 
dg_b, c, pdc, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgemm (f16yac).\n%s\n", fail.message);
    exit_status = 1;
goto END;
}
dg_a = -1.0;
dg_b = 1.0;
nag_dgemm(order, Nag_NoTrans, Nag_Trans, n, n, n, dg_a, c, pdc, vsr, pdvsr, 
dg_b, e, pde, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgemm (f16yac).\n%s\n", fail.message);
    exit_status = 1;
goto END;
}
/* nag_dge_norm (f16rac): Find norms of difference matrices D and E. */
nag_dge_norm(order, Nag_OneNorm, n, n, d, pdd, &normd, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
    exit_status = 1;
goto END;
}
nag_dge_norm(order, Nag_OneNorm, n, n, e, pde, &norme, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dge_norm (f16rac).\n%s\n", fail.message);
    exit_status = 1;
goto END;
}
/* Get the machine precision, using nag_machine_precision (x02ajc) */
eps = nag_machine_precision;
if (MAX(normd,norme) > pow(eps,0.8)*MAX(norma,normb))
{
    printf("The norm of the error in the reconstructed matrices is greater " 
    "than expected.\nThe Schur factorization has failed.\n");
    exit_status = 1;
goto END;
}
/* Print details on eigenvalues */
printf("Number of sorted eigenvalues = %4"NAG_IFMT"
\n", sdim);
if (fail.code == NE_SCHUR_REORDER_SELECT) {
  printf("*** Note that rounding errors mean that leading eigenvalues in the"
    " generalized Schur form no longer satisfy selctg = Nag_TRUE"
    "\n\n");
} else {
  printf("The selected eigenvalues are:\n");
  for (i=0;i<sdim;i++) {
    if (beta[i] != 0.0)
      printf("%3"NAG_IFMT" (%13.4e, %13.4e)\n",
          i+1, alphar[i]/beta[i], alphai[i]/beta[i]);
    else
      printf("%3"NAG_IFMT" Eigenvalue is infinite\n", i + 1);
  }
}

END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(c);
NAG_FREE(d);
NAG_FREE(e);
NAG_FREE(alphai);
NAG_FREE(alphar);
NAG_FREE(beta);
NAG_FREE(vsl);
NAG_FREE(vsr);

return exit_status;
}

static Nag_Boolean NAG_CALL selctg(const double ar, const double ai, const double b)
{
  /* Logical function selctg for use with nag_dgges (f08xac).
   * Returns the value Nag_TRUE if the eigenvalue is real and positive.
   */
  return (ar > 0.0 && ai == 0.0 && b != 0.0 ? Nag_TRUE : Nag_FALSE);
}

10.2 Program Data

f08_dgges (f08xac) Example Program Data

4

nag_LeftVecs : jobvsl
Nag_RightVecs : jobvsr

3.9 12.5 -34.5 -0.5
4.3 21.5 -47.5 7.5
4.3 21.5 -43.5 3.5
4.4 26.0 -46.0 6.0 : A

1.0 2.0 -3.0 1.0
1.0 3.0 -5.0 4.0
1.0 3.0 -4.0 3.0
1.0 3.0 -4.0 4.0 : B
10.3 Program Results

nag_dgges (f08xac) Example Program Results

Matrix A

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.9000</td>
<td>12.5000</td>
<td>-34.5000</td>
<td>-0.5000</td>
</tr>
<tr>
<td>2</td>
<td>4.3000</td>
<td>21.5000</td>
<td>-47.5000</td>
<td>7.5000</td>
</tr>
<tr>
<td>3</td>
<td>4.3000</td>
<td>21.5000</td>
<td>-43.5000</td>
<td>3.5000</td>
</tr>
<tr>
<td>4</td>
<td>4.4000</td>
<td>26.0000</td>
<td>-46.0000</td>
<td>6.0000</td>
</tr>
</tbody>
</table>

Matrix B

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000</td>
<td>2.0000</td>
<td>-3.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>2</td>
<td>1.0000</td>
<td>3.0000</td>
<td>-5.0000</td>
<td>4.0000</td>
</tr>
<tr>
<td>3</td>
<td>1.0000</td>
<td>3.0000</td>
<td>-4.0000</td>
<td>3.0000</td>
</tr>
<tr>
<td>4</td>
<td>1.0000</td>
<td>3.0000</td>
<td>-4.0000</td>
<td>4.0000</td>
</tr>
</tbody>
</table>

Number of sorted eigenvalues = 2

The selected eigenvalues are:
1 ( 2.00000e+00, 0.00000e+00)
2 ( 4.00000e+00, 0.00000e+00)