NAG Library Function Document

nag_dggesx (f08xbc)

1 Purpose

nag_dggesx (f08xbc) 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 \((A, B)\).

Estimates of condition numbers for selected generalized eigenvalue clusters and Schur vectors are also computed.

2 Specification

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

void nag_dggesx (Nag_OrderType order, Nag_LeftVecsType jobvsl,
                 Nag_RightVecsType jobvsr, Nag_SortEigValsType sort,
                 Nag_Boolean (*selctg)(double ar, double ai, double b),
                 Nag_RCondType sense, 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, double rconde[ ], double rcondv[ ], NagError *fail)
```

3 Description

The generalized real Schur factorization of \((A, B)\) 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 \(T\) and \(S\) 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_dggesx (f08xbc) 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_dggesx (f08xbc) 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.

The reciprocals of the condition estimates, the reciprocal values of the left and right projection norms, are returned in rconde[0] and rconde[1] respectively, for the selected generalized eigenvalues, together with reciprocal condition estimates for the corresponding left and right deflating subspaces, in rcondv[0] and rcondv[1]. See Section 4.11 of Anderson et al. (1999) for further information.
References


Arguments

1: \textbf{order} – Nag_OrderType \hspace{1cm} \textit{Input}

\textit{On entry:} the \textbf{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 \textbf{order} = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

\textit{Constraint:} \textbf{order} = Nag_RowMajor or Nag_ColMajor.

2: \textbf{jobvsl} – Nag_LeftVecsType \hspace{1cm} \textit{Input}

\textit{On entry:} if \textbf{jobvsl} = Nag_NotLeftVecs, do not compute the left Schur vectors.

If \textbf{jobvsl} = Nag_LeftVecs, compute the left Schur vectors.

\textit{Constraint:} \textbf{jobvsl} = Nag_NotLeftVecs or Nag_LeftVecs.

3: \textbf{jobvsr} – Nag_RightVecsType \hspace{1cm} \textit{Input}

\textit{On entry:} if \textbf{jobvsr} = Nag_NotRightVecs, do not compute the right Schur vectors.

If \textbf{jobvsr} = Nag_RightVecs, compute the right Schur vectors.

\textit{Constraint:} \textbf{jobvsr} = Nag_NotRightVecs or Nag_RightVecs.

4: \textbf{sort} – Nag_SortEigValsType \hspace{1cm} \textit{Input}

\textit{On entry:} specifies whether or not to order the eigenvalues on the diagonal of the generalized Schur form.

\textbf{sort} = Nag_NoSortEigVals \hspace{1cm} \textit{Eigenvalues are not ordered.}

\textbf{sort} = Nag_SortEigVals \hspace{1cm} \textit{Eigenvalues are ordered (see \textbf{selctg}).}

\textit{Constraint:} \textbf{sort} = Nag_NoSortEigVals or Nag_SortEigVals.

5: \textbf{selctg} – function, supplied by the user \hspace{1cm} \textit{External Function}

\textit{If} \textbf{sort} = Nag_SortEigVals, \textbf{selctg} is used to select generalized eigenvalues to the top left of the generalized Schur form.

\textit{If} \textbf{sort} = Nag_NoSortEigVals, \textbf{selctg} is not referenced by \texttt{nag_dggesx (f08xbc)}, and may be specified as NULLFN.

\begin{verbatim}
The specification of \textbf{selctg} is:
Nag_Boolean selctg (double ar, double ai, double b)
\end{verbatim}
On entry: an eigenvalue 
\( (\text{ar}[j-1] + \sqrt{-1} \times \text{ai}[j-1]) / \text{b}[j-1] \) is selected if 
\( \text{selectg} (\text{ar}[j-1], \text{ai}[j-1], \text{b}[j-1]) \) is 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 
\( \text{selectg} (\text{ar}[j-1], \text{ai}[j-1], \text{b}[j-1]) = \text{Nag_TRUE} \) after ordering.

\( \text{fail.code} = \text{NE_SCHUR_REORDER_SELECT} \) in this case.

### Parameters

1. **ar** – double
   **Input**

2. **ai** – double
   **Input**

3. **b** – double
   **Input**

   On entry: determines which reciprocal condition numbers are computed.

   - **sense** = Nag_NotRCond
     None are computed.
   - **sense** = Nag_RCondEigVals
     Computed for average of selected eigenvalues only.
   - **sense** = Nag_RCondEigVecs
     Computed for selected deflating subspaces only.
   - **sense** = Nag_RCondBoth
     Computed for both.

   If **sense** = Nag_RCondEigVals, Nag_RCondEigVecs or Nag_RCondBoth, **sort** = Nag_SortEigVals.

   **Constraint:** **sense** = Nag_NotRCond, Nag_RCondEigVals, Nag_RCondEigVecs or Nag_RCondBoth.

4. **n** – Integer
   **Input**

   On entry: \( n \), the order of the matrices \( A \) and \( B \).

   **Constraint:** \( n \geq 0 \).

5. **a[dim]** – double
   **Input/Output**

   On entry: the first of the pair of matrices, \( A \).

   On exit: \( a \) has been overwritten by its generalized Schur form \( S \).

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

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

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

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

   On entry: the first of the pair of matrices, \( A \).

   On exit: \( a \) has been overwritten by its generalized Schur form \( S \).

6. **pda** – Integer
   **Input**

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

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

7. **b[dim]** – double
   **Input/Output**

   On entry: the second of the pair of matrices, \( B \).

   **Note:** the dimension, \( dim \), 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

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

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

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

### 11: \texttt{pdb} – Integer

*Input*

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

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

### 12: \texttt{sdim} – Integer *

*Output*

On exit: if \texttt{sort} = \texttt{Nag\_NoSortEigVals}, \(sdim = 0\).

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

### 13: \texttt{alphar}[n] – double

*Output*

On exit: see the description of \texttt{beta}.

### 14: \texttt{alphai}[n] – double

*Output*

On exit: see the description of \texttt{beta}.

### 15: \texttt{beta}[n] – double

*Output*

On exit: \(\frac{\text{alphar}[j-1] + \text{alphai}[j-1] \times i}{\text{beta}[j-1]}, \) 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 \(\frac{\text{alphar}[j-1]}{\text{beta}[j-1]}\) and \(\frac{\text{alphai}[j-1]}{\text{beta}[j-1]}\) 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, \texttt{alphar} and \texttt{alphai} will always be less than and usually comparable with \(\|a\|_2\) in magnitude, and \texttt{beta} will always be less than and usually comparable with \(\|b\|_2\).*

### 16: \texttt{vsl}[dim] – double

*Output*

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

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

1 otherwise.

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

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

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

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

If \texttt{jobvsl} = \texttt{Nag\_NotLeftVecs}, \texttt{vsl} is not referenced.

### 17: \texttt{pdvsl} – Integer

*Input*

On entry: the stride used in the array \(vsl.\)

*Constraints:*

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

otherwise \(pdvsl \geq 1.\)
18: `vsr[dim]` – double

**Output**

Note: the dimension, `dim`, of the array `vsr` must be at least

\[ \max(1, pdvsr \times n) \] when `jobvsr` = `Nag_RightVecs`;

1 otherwise.

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

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

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

On exit: if `jobvsr` = `Nag_RightVecs`, `vsr` will contain the right Schur vectors, \( Z \).

If `jobvsr` = `Nag_NotRightVecs`, `vsr` is not referenced.

19: `pdvsr` – Integer

**Input**

On entry: the stride used in the array `vsr`.

**Constraints:**

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

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


**Output**

On exit: if `sense` = `Nag_RCondEigVals` or `Nag_RCondBoth`, `rconde[0]` and `rconde[1]` contain the reciprocal condition numbers for the average of the selected eigenvalues.

If `sense` = `Nag_NotRCond` or `Nag_RCondEigVecs`, `rconde` is not referenced.


**Output**

On exit: if `sense` = `Nag_RCondEigVecs` or `Nag_RCondBoth`, `rcondv[0]` and `rcondv[1]` contain the reciprocal condition numbers for the selected deflating subspaces.

If `sense` = `Nag_NotRCond` or `Nag_RCondEigVals`, `rcondv` 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 \( \langle \text{value} \rangle \) had an illegal value.

**NE_ENUM_INT_2**

On entry, `jobvs` = \( \langle \text{value} \rangle \), `pdvs` = \( \langle \text{value} \rangle \) and `n` = \( \langle \text{value} \rangle \).

Constraint: if `jobvs` = `Nag_LeftVecs`, `pdvs` \( \geq \max(1, n) \);
otherwise `pdvs` \( \geq 1 \).

On entry, `jobvsr` = \( \langle \text{value} \rangle \), `pdvsr` = \( \langle \text{value} \rangle \) and `n` = \( \langle \text{value} \rangle \).

Constraint: if `jobvsr` = `Nag_RightVecs`, `pdvsr` \( \geq \max(1, n) \);
otherwise `pdvsr` \( \geq 1 \).
NE_INT
On entry, \( n = \langle \text{value} \rangle \).  
Constraint: \( n \geq 0 \).

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

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

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

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

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

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

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_ITERATION_QZ
The QZ iteration failed. No eigenvectors have been calculated but \( \text{alphar}[j] \), \( \text{alphai}[j] \) and \( \text{beta}[j] \) should be correct from element \( \langle \text{value} \rangle \).

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 \( \text{selectg} = \text{Nag_TRUE} \). 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_dggesx (f08xbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dggesx (f08xbc) 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_zggesx (f08xpc).

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, $(S, T)$. Estimates of the condition numbers for the selected eigenvalue cluster and corresponding deflating subspaces are also returned.

10.1 Program Text

/* nag_dggesx (f08xbc) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
* Mark 25, 2014.
*/

#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>

#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 abnorm, dg_a, dg_b, eps, norma, normb, normd, norme, tol;
    Integer i, j, n, sdim, pda, pdb, pdc, pdd, pde, pdvs1, pdvsr;
    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;
double  rconde[2], rcondv[2];
char     nag_enum_arg[40];

/* Nag Types */
NagError    fail;
Nag_OrderType  order;
Nag_LeftVecsType  jobvsl;
Nag_RightVecsType  jobvsr;
Nag_SortEigValsType  sort = Nag_SortEigVals;
Nag_RCondType   sense = Nag_RCondBoth;

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

INIT_FAIL(fail);

printf("nag_dggesx (f08xbc) Example Program Results\n\n");

/* Skip heading in data file */
#endif _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;
    return exit_status;
}

#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 */

jobvsl = (Nag_LeftVecsType) nag_enum_name_to_value(nag_enum_arg);
#endif _WIN32

#ifdef _WIN32
scanf_s("%39s%*[\n"]", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf("%39s%*[\n"]", nag_enum_arg);
#endif

jobvsr = (Nag_RightVecsType) nag_enum_name_to_value(nag_enum_arg);
#endif _WIN32

#ifdef _WIN32
scanf_s("%39s%*[\n"]", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf("%39s%*[\n"]", nag_enum_arg);
#endif

dad = n;

f08xbc
NAG Library Manual
f08xbc.8 Mark 25
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)  
    #ifdef _WIN32  
        for (j = 1; j <= n; ++j) scanf_s("%lf", &A(i, j));  
    #else  
        for (j = 1; j <= n; ++j) scanf("%lf", &A(i, j));  
    #endif  
    #ifdef _WIN32  
        scanf_s("%*[\n]");  
    #else  
        scanf("%*[\n]");  
    #endif  
    #ifdef _WIN32  
        for (j = 1; j <= n; ++j) scanf_s("%lf", &B(i, j));  
    #else  
        for (j = 1; j <= n; ++j) scanf("%lf", &B(i, j));  
    #endif  
    #ifdef _WIN32  
        scanf_s("%*[\n]");  
    #else  
        scanf("%*[\n]");  
    #endif  
/* 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", fail.message);  
    exit_status = 1;  
    goto END;  
}  
/* nag_dge_norm (f16rac): Find norms of input matrices A and B. */  
nag_dge_norm(order, Nag_PronobiusNorm, n, n, a, pda, &norma, &fail);  
if (fail.code != NE_NOERROR)  
{  
    printf("Error from nag_dge_norm (f16rac)\n", fail.message);  
    exit_status = 1;  
    goto END;  
}
nag_dge_norm(order, Nag_ProbieniusNorm, n, n, b, pdb, &normb, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dge_norm (f16rac).\n\n\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\n\n", fail.message);
    exit_status = 1;
    goto END;
}

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\n\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Find the generalized Schur form using nag_dggesx (f08xbc). */

nag_dggesx(order, jobvsl, jobvsr, sort, selctg, sense, n, a, pda, b, pdb,
  &sdim, alphar, alphai, beta, vsl, pdvsl, vsr, pdvsr, rconde, rcondv, &fail);
if (fail.code != NE_NOERROR && fail.code != NE_SCHUR_REORDER_SELECT)
{
    printf("Error from nag_dggesx (f08xbc).\n\n\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.
* \( D = D - C^2 Z^T \) (\( Z \) in vsr) using nag_dgemm (f16yac).
*/
dg_a = 1.0;
dg_b = 0.0;
nag_dgemm(order, Nag_NoTrans, Nag_NoTrans, n, n, dg_a, vsl, pdvsl, a, pda,
dg_b, c, pdc, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgemm (f16yac).\n\n\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, dg_a, c, pdc, vsr, pdvsr,
dg_b, d, pdd, &fail);

if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgemm (f16yac).\n\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\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\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_FrobeniusNorm, n, n, d, pdd, &normd, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dge_norm (f16rac).\n\n", fail.message);
    exit_status = 1;
    goto END;
}
nag_dge_norm(order, Nag_FrobeniusNorm, n, n, e, pde, &norme, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dge_norm (f16rac).\n\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\n    Schur form no longer satisfy selctg = Nag_TRUE" "\n
");
} else {
    printf("The selected eigenvalues are:\n\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
        i+1, alpha[i]/beta[i], alpha[i]/beta[i]);

Mark 25 f08xbc

f08xbc.11
printf("%3"NAG_IFMT" Eigenvalue is infinite\n", i + 1);
}

abnorm = sqrt(pow(norma, 2) + pow(normb, 2));
tol = eps*abnorm;

if (sense==Nag_RCondEigVals || sense==Nag_RCondBoth) {
    /* Print out the reciprocal condition number and error bound */
    printf("\n");
    printf("For the selected eigenvalues,\nthe reciprocals of projection "
    "norms onto the deflating subspaces are:\n");
    printf(" for left subspace, rcond = %.1e\n for right subspace, rcond = "
    "%.1e\n", rconde[0], rconde[1]);
    printf(" asymptotic error bound = %.1e", tol / rconde[0]);
}
if (sense==Nag_RCondEigVecs || sense==Nag_RCondBoth) {
    /* Print out the reciprocal condition numbers and error bound. */
    printf("\n");
    printf("For the left and right deflating subspaces,\n");
    printf(" reciprocals condition numbers are:\n");
    printf(" for left subspace, rcond = %.1e\n for right subspace, rcond = "
    "%.1e\n", rcondv[0], rcondv[1]);
    printf(" approximate error bound = %.1e", tol / rcondv[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)
{
    /* Boolean function selctg for use with nag_dggesx (f08xbc)
     * 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

nag_dggesx (f08xbc) Example Program Data

4 : n

Nag_LeftVecs : jobvsl
Nag_RightVecs : jobvsr
Nag_RCondBoth : sense

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 : matrix 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 : matrix B
10.3 Program Results

nag_dggesx (f08xbc) 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>3.900</td>
<td>12.500</td>
<td>-34.500</td>
<td>-0.500</td>
</tr>
<tr>
<td>2</td>
<td>4.300</td>
<td>21.500</td>
<td>-47.500</td>
<td>7.500</td>
</tr>
<tr>
<td>3</td>
<td>4.300</td>
<td>21.500</td>
<td>-43.500</td>
<td>3.500</td>
</tr>
<tr>
<td>4</td>
<td>4.400</td>
<td>26.000</td>
<td>-46.000</td>
<td>6.000</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>1.000</td>
<td>2.000</td>
<td>-3.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>1.000</td>
<td>3.000</td>
<td>-5.000</td>
<td>4.000</td>
</tr>
<tr>
<td>3</td>
<td>1.000</td>
<td>3.000</td>
<td>-4.000</td>
<td>3.000</td>
</tr>
<tr>
<td>4</td>
<td>1.000</td>
<td>3.000</td>
<td>-4.000</td>
<td>4.000</td>
</tr>
</tbody>
</table>

Number of sorted eigenvalues = 2

The selected eigenvalues are:
1 ( 2.0000e+00, 0.0000e+00)
2 ( 4.0000e+00, 0.0000e+00)

For the selected eigenvalues, the reciprocals of projection norms onto the deflating subspaces are
for left subspace, rcond = 1.9e-01
for right subspace, rcond = 1.8e-02
asymptotic error bound = 5.7e-14

For the left and right deflating subspaces, reciprocal condition numbers are:
for left subspace, rcond = 5.4e-02
for right subspace, rcond = 9.0e-02

approximate error bound = 1.2e-13