NAG Library Function Document

nag_zgges (f08xnc)

1 Purpose

nag_zgges (f08xnc) computes the generalized eigenvalues, the generalized Schur form \((S, T)\) and, optionally, the left and/or right generalized Schur vectors for a pair of \(n\) by \(n\) complex nonsymmetric matrices \((A, B)\).

2 Specification

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

3 Description

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

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

where \(Q\) and \(Z\) are unitary, \(T\) and \(S\) are upper triangular. 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_zgges (f08xnc) 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_zgges (f08xnc) computes \(T\) to have real non-negative diagonal entries. The generalized Schur factorization, before reordering, is computed by the \(QZ\) algorithm.

4 References


5 Arguments

1: \texttt{order} – \texttt{Nag\_OrderType} \hspace{1cm} \textit{Input}

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

\textit{Constraint:} \texttt{order} $= \texttt{Nag\_RowMajor}$ or \texttt{Nag\_ColMajor}.

2: \texttt{jobvl} – \texttt{Nag\_LeftVecsType} \hspace{1cm} \textit{Input}

\textit{On entry:} if \texttt{jobvl} $= \texttt{Nag\_NotLeftVecs}$, do not compute the left Schur vectors. If \texttt{jobvl} $= \texttt{Nag\_LeftVecs}$, compute the left Schur vectors.

\textit{Constraint:} \texttt{jobvl} $= \texttt{Nag\_NotLeftVecs}$ or \texttt{Nag\_LeftVecs}.

3: \texttt{jobvsr} – \texttt{Nag\_RightVecsType} \hspace{1cm} \textit{Input}

\textit{On entry:} if \texttt{jobvsr} $= \texttt{Nag\_NotRightVecs}$, do not compute the right Schur vectors. If \texttt{jobvsr} $= \texttt{Nag\_RightVecs}$, compute the right Schur vectors.

\textit{Constraint:} \texttt{jobvsr} $= \texttt{Nag\_NotRightVecs}$ or \texttt{Nag\_RightVecs}.

4: \texttt{sort} – \texttt{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.

\texttt{sort} $= \texttt{Nag\_NoSortEigVals}$

Eigenvalues are not ordered.

\texttt{sort} $= \texttt{Nag\_SortEigVals}$

Eigenvalues are ordered (see \texttt{selctg}).

\textit{Constraint:} \texttt{sort} $= \texttt{Nag\_NoSortEigVals}$ or \texttt{Nag\_SortEigVals}.

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

\textit{If} \texttt{sort} $= \texttt{Nag\_SortEigVals}$, \texttt{selctg} is used to select generalized eigenvalues to the top left of the generalized Schur form.

\textit{If} \texttt{sort} $= \texttt{Nag\_NoSortEigVals}$, \texttt{selctg} is not referenced by \texttt{nag\_zgges} (f08xnc), and may be specified as NULLFN.

The specification of \texttt{selctg} is:

\begin{verbatim}
Nag_Boolean selctg (Complex a, Complex b)
1: a – Complex \hspace{1cm} \textit{Input}
2: b – Complex \hspace{1cm} \textit{Input}
\end{verbatim}

\textit{On entry:} an eigenvalue $a[j-1]/b[j-1]$ is selected if $\texttt{selctg}(a[j-1], b[j-1])$ is \texttt{Nag\_TRUE}.

\textit{Note that in the ill-conditioned case, a selected generalized eigenvalue may no longer satisfy $\texttt{selctg}(a[j-1], b[j-1]) = \texttt{Nag\_TRUE}$ after ordering. \texttt{fail.code} = \texttt{NE\_SCHUR\_REORDER\_SELECT} in this case.}

6: \texttt{n} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} $n$, the order of the matrices $A$ and $B$.

\textit{Constraint:} $n \geq 0$. 

\hspace{1cm}
7: \(a[\text{dim}]\) – Complex

Input/Output

Note: the dimension, \(\text{dim}\), 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

\[
\begin{align*}
\text{a}[(j - 1) \times \text{pda} + i - 1] & \quad \text{when order} = \text{Nag_ColMajor}; \\
\text{a}[(i - 1) \times \text{pda} + j - 1] & \quad \text{when order} = \text{Nag_RowMajor}.
\end{align*}
\]

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: \(b[\text{dim}]\) – Complex

Input/Output

Note: the dimension, \(\text{dim}\), 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

\[
\begin{align*}
\text{b}[(j - 1) \times \text{pdb} + i - 1] & \quad \text{when order} = \text{Nag_ColMajor}; \\
\text{b}[(i - 1) \times \text{pdb} + j - 1] & \quad \text{when order} = \text{Nag_RowMajor}.
\end{align*}
\]

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 \(\text{sort} = \text{Nag_SortEigVals}\), \(sdim\) = number of eigenvalues (after sorting) for which \(\text{seleig} \) is \(\text{Nag_TRUE}\).

12: \(\text{alpha}[n]\) – Complex

Output

On exit: see the description of \(\text{beta}\).

13: \(\text{beta}[n]\) – Complex

Output

On exit: \(\text{alpha}[j - 1]/\text{beta}[j - 1]\), for \(j = 1, 2, \ldots, n\), will be the generalized eigenvalues. \(\text{alpha}[j - 1]\), for \(j = 1, 2, \ldots, n\) and \(\text{beta}[j - 1]\), for \(j = 1, 2, \ldots, n\), are the diagonals of the complex Schur form \((A, B)\) output by \text{nag_zgges} (f08xnc). The \(\text{beta}[j - 1]\) will be non-negative real.

Note: the quotients \(\text{alpha}[j - 1]/\text{beta}[j - 1]\) 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, \(\text{alpha}\) will always be less than and usually comparable with \(\|a\|\) in magnitude, and \(\text{beta}\) will always be less than and usually comparable with \(\|b\|\).
14: \texttt{vsl[dim]} – Complex

\textbf{Output}

\textbf{Note:} the dimension, \texttt{dim}, of the array \texttt{vsl} must be at least

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

\[
1 \text{ otherwise.}
\]

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

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

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

\textbf{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.

15: \texttt{pdvsl} – Integer

\textbf{Input}

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

\textbf{Constraints:}

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

\[
\text{otherwise } \texttt{pdvsl} \geq 1.
\]

16: \texttt{vsr[dim]} – Complex

\textbf{Output}

\textbf{Note:} the dimension, \texttt{dim}, of the array \texttt{vsr} must be at least

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

\[
1 \text{ otherwise.}
\]

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

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

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

\textbf{On exit:} if \texttt{jobvsr} = \texttt{Nag\_RightVecs}, \texttt{vsr} will contain the right Schur vectors, \(Z\).

If \texttt{jobvsr} = \texttt{Nag\_NotRightVecs}, \texttt{vsr} is not referenced.

17: \texttt{pdvsr} – Integer

\textbf{Input}

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

\textbf{Constraints:}

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

\[
\text{otherwise } \texttt{pdvsr} \geq 1.
\]

18: \texttt{fail} – NagError *

\textbf{Input/Output}

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 \hspace{1em} \textbf{Error Indicators and Warnings}

\textbf{NE\_ALLOC\_FAIL}

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE\_BAD\_PARAM}

On entry, argument \(\langle value\rangle\) 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 did not converge and the matrix pair (A, B) is not in the generalized Schur form. The computed α_i and β_i should be correct for i = ⟨value⟩, ..., ⟨value⟩.
The QZ iteration failed with an unexpected error, please contact NAG.

NE_NO_LI能得到
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

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

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_zgges (f08xnc)` is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

`nag_zgges (f08xnc)` 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 real analogue of this function is `nag_dgges (f08xac)`.

10 Example

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

\[
A = \begin{pmatrix}
-21.10 - 22.50i & 53.50 - 50.50i & -34.50 + 127.50i & 7.50 + 0.50i \\
-0.46 - 7.78i & -3.50 - 37.50i & -15.50 + 58.50i & -10.50 - 1.50i \\
4.30 - 5.50i & 39.70 - 17.10i & -68.50 + 12.50i & -7.50 - 3.50i \\
5.50 + 4.40i & 14.40 + 43.30i & -32.50 - 46.00i & -19.00 - 32.50i \\
\end{pmatrix}
\]

and

\[
B = \begin{pmatrix}
1.00 - 5.00i & 1.60 + 1.20i & -3.00 + 0.00i & 0.00 - 1.00i \\
0.80 - 0.60i & 3.00 - 5.00i & -4.00 + 3.00i & -2.40 - 3.20i \\
1.00 + 0.00i & 2.40 + 1.80i & -4.00 - 5.00i & 0.00 - 3.00i \\
0.00 + 1.00i & -1.80 + 2.40i & 0.00 - 4.00i & 4.00 - 5.00i \\
\end{pmatrix}
\]

10.1 Program Text

/* nag_zgges (f08xnc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* * Mark 25, 2014. */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx02.h>
#include <nagx04.h>
int main(void)
{
    /* Scalars */
    Complex alph, bet, z;
    double norma, normb, normd, norme, eps;
    Integer i, j, n, sdim, pda, pdb, pdc, pdd, pde, pdvsl, pdvsr;
    Integer exit_status = 0;

    /* Arrays */
    Complex *a = 0, *alpha = 0, *b = 0, *beta = 0, *c = 0;
    Complex *d = 0, *e = 0, *vsl = 0, *vsr = 0;
    Integer exit_status = 0;

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

    INIT_FAIL(fail);
    printf("nag_zgges (f08xnc) Example Program Results\n\n");

    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\n]");
    #else
    scanf("%*[\n]");
    #endif
    #ifdef _WIN32
    scanf_s("%39s%*[\n]", &n);
    #else
    scanf("%39s%*[\n]", &n);
    #endif
    if (n < 0)
    {
        printf("Invalid n\n");
        exit_status = 1;
        return exit_status;
    }

    char nag_enum_arg[40];
    jobvsl = (Nag_LeftVecsType) nag_enum_name_to_value(nag_enum_arg);
    jobvsr = (Nag_RightVecsType) nag_enum_name_to_value(nag_enum_arg);
    pdvsl = (jobvsl==Nag_LeftVecs?n:1);
    pdvsr = (jobvsr==Nag_RightVecs?n:1);
    pdvsl = n;
pdb = n;
pdc = n;
pdd = n;
pde = n;
/* Allocate memory */
if (!((a = NAG_ALLOC(n * n, Complex)) ||
    (b = NAG_ALLOC(n * n, Complex)) ||
    (c = NAG_ALLOC(n * n, Complex)) ||
    (d = NAG_ALLOC(n * n, Complex)) ||
    (e = NAG_ALLOC(n * n, Complex)) ||
    (alpha = NAG_ALLOC(n, Complex)) ||
    (beta = NAG_ALLOC(n, Complex)) ||
    (vsl = NAG_ALLOC(pdvsl*pdvsl, Complex)) ||
    (vsr = NAG_ALLOC(pdvsr*pdvsr, Complex)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Read in the matrices A and B */
for (i = 1; i <= n; ++i)
    for (j = 1; j <= n; ++j)
#ifdef _WIN32
    scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
    scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
#ifdef _WIN32
    scanf_s(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#else
    scanf(" ( %lf , %lf )", &B(i, j).re, &B(i, j).im);
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
/* Copy A and B to D and E respectively: nag_zge_copy (f16tfc),
   * Complex valued general matrix copy. */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zge_copy (f16tfc).\n\n", fail.message);
    exit_status = 1;
    goto END;
}
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zge_copy (f16tfc).\n\n", fail.message);
    exit_status = 1;
    goto END;
}
/* nag_zge_norm (f16uac): Find norms of input matrices A and B. */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zge_norm (f16uac).\n\n", fail.message);
    exit_status = 1;
    goto END;
}
nag_zge_norm(order, Nag_OneNorm, n, n, b, pdb, &normb, &fail);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_zge_norm (f16uac)\n%sn", fail.message);
  exit_status = 1;
  goto END;
}

/* nag_gen_complex_mat_print_comp (x04dbc): Print matrices A and B. */
fflush(stdout);
nag_gen_complex_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
  n, a, pda, Nag_BracketForm, "%6.2f",
  "Matrix A", Nag_IntegerLabels, 0,
  Nag_IntegerLabels, 0, 80, 0, 0, &fail);
printf("\n");
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_gen_complex_mat_print_comp (x04dbc)\n%sn", fail.message);
  exit_status = 1;
  goto END;
}

fflush(stdout);
nag_gen_complex_mat_print_comp(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
  n, b, pdb, Nag_BracketForm, "%6.2f",
  "Matrix B", Nag_IntegerLabels, 0,
  Nag_IntegerLabels, 0, 80, 0, 0, &fail);
printf("\n");
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_gen_complex_mat_print_comp (x04dbc)\n%sn", fail.message);
  exit_status = 1;
  goto END;
}

/* Find the generalized Schur form using nag_zgges (f08xnc). */
nag_zgges(order, jobvsl, jobvsr, Nag_NoSortEigVals, NULLFN, n, a, pda, b,
  pdb, &sdim, alpha, beta, vsl, pdvsl, vsr, pdvsr, &fail);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_zgges (f08xnc)\n%sn", 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^H and subtract from original (D) using the steps
* C = Q (Q in vsl) using nag_zge_copy (f16tfc).
* C = C*S (S in a, upper triangular) using nag_ztrmm (f16zfc).
* D = D - C*Z^H (Z in vsr) using nag_zgemm (f16zac).
*/
nag_zge_copy(order, Nag_NoTrans, n, n, vsl, pdvsl, c, pdc, &fail);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_zge_copy (f16tfc)\n%sn", fail.message);
  exit_status = 1;
  goto END;
}
alph = nag_complex(1.0,0.0);
/* nag_ztrmm (f16zfc) Triangular complex matrix-matrix multiply. */
nag_ztrmm(order, Nag_RightSide, Nag_Upper, Nag_NoTrans, Nag_NoUnitDiag, n,
n, alph, a, pda, c, pdc, &fail);  
if (fail.code != NE_NOERROR)  
{  
    printf("Error from nag_ztrmm (f16zfc).\n%s\n", fail.message);  
    exit_status = 1;  
    goto END;  
}  
alph = nag_complex(-1.0,0.0);  
bet = nag_complex(1.0,0.0);  
nag_zgemm(order, Nag_NoTrans, Nag_ConjTrans, n, n, n, alph, c, pdc, vsr,  
        pdvsr, bet, d, pdd, &fail);  
if (fail.code != NE_NOERROR)  
{  
    printf("Error from nag_zgemm (f16zac).\n%s\n", fail.message);  
    exit_status = 1;  
    goto END;  
}  
/* Reconstruct B as Q*T*Z^H and subtract from original (E) using the steps  
* Q = Q*T (Q in vsl, T in b, upper triangular) using nag_ztrmm (f16zfc).  
* E = E - Q*Z^H (Z in vsr) using nag_zgemm (f16zac).  
*/  
alph = nag_complex(1.0,0.0);  
nag_ztrmm(order, Nag_RightSide, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, n,  
        n, alph, b, pdb, vsl, pdvsl, &fail);  
if (fail.code != NE_NOERROR)  
{  
    printf("Error from nag_ztrmm (f16zfc).\n%s\n", fail.message);  
    exit_status = 1;  
    goto END;  
}  
alph = nag_complex(-1.0,0.0);  
bet = nag_complex(1.0,0.0);  
nag_zgemm(order, Nag_NoTrans, Nag_ConjTrans, n, n, n, alph, vsl, pdvsl, vsr,  
        pdvsr, bet, e, pde, &fail);  
if (fail.code != NE_NOERROR)  
{  
    printf("Error from nag_zgemm (f16zac).\n%s\n", fail.message);  
    exit_status = 1;  
    goto END;  
}  
/* nag_zge_norm (f16uac): Find norms of difference matrices D and E. */  
nag_zge_norm(order, Nag_OneNorm, n, n, d, pdd, &normd, &fail);  
if (fail.code != NE_NOERROR)  
{  
    printf("Error from nag_zge_norm (f16uac).\n%s\n", fail.message);  
    exit_status = 1;  
    goto END;  
}  
nag_zge_norm(order, Nag_OneNorm, n, n, e, pde, &norme, &fail);  
if (fail.code != NE_NOERROR)  
{  
    printf("Error from nag_zge_norm (f16uac).\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.\nnThe Schur factorization has failed.\n"");  
    exit_status = 1;  
    goto END;  
}  
/* Print details on eigenvalues */  
printf("Generalized eigenvalues are:\n");  
for (i=0;i<n;i++)  
{
if (beta[i].re != 0.0 || beta[i].im != 0.0) {
    z = nag_complex_divide(alpha[i], beta[i]);
    printf("%3"NAG_IFMT" (%13.4e, %13.4e)n", i + 1, z.re, z.im);
} else
    printf("%3"NAG_IFMT" Eigenvalue is infinite
", i + 1);
}

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

return exit_status;

10.2 Program Data

nag_zgges (f08xnc) Example Program Data

4 : n
Nag_LeftVecs : jobvsl
Nag_RightVecs : jobvsr

(-21.10,-22.50) ( 53.50, -50.50) (-34.50,127.50) ( 7.50, 0.50)
(-0.46, -7.78) ( -3.50,-37.50) (-15.50, 58.50) (-10.50, -1.50)
( 4.30, -5.50) ( 39.70,-17.10) (-68.50, 12.50) (-7.50, -3.50)
( 5.50, 4.40) ( 14.40, 43.30) (-32.50,-46.00) (-19.00,-32.50) : A

( 1.00, -5.00) ( 1.60, 1.20) ( -3.00, 0.00) ( 0.00, -1.00)
( 0.80, -0.60) ( 3.00, -5.00) ( -4.00, 3.00) (-2.40, -3.20)
( 1.00, 0.00) ( 2.40, 1.80) ( -4.00, -5.00) ( 0.00, -3.00)
( 0.00, 1.00) ( -1.80, 2.40) ( 0.00, -4.00) ( 4.00, -5.00) : B

10.3 Program Results

nag_zgges (f08xnc) 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>(-21.10,-22.50)</td>
<td>(53.50,-50.50)</td>
<td>(-34.50,127.50)</td>
<td>(7.50, 0.50)</td>
</tr>
<tr>
<td>2</td>
<td>(-0.46, -7.78)</td>
<td>(-3.50,-37.50)</td>
<td>(-15.50, 58.50)</td>
<td>(-10.50, -1.50)</td>
</tr>
<tr>
<td>3</td>
<td>(4.30, -5.50)</td>
<td>(39.70,-17.10)</td>
<td>(-68.50, 12.50)</td>
<td>(-7.50, -3.50)</td>
</tr>
<tr>
<td>4</td>
<td>(5.50, 4.40)</td>
<td>(14.40, 43.30)</td>
<td>(-32.50,-46.00)</td>
<td>(-19.00,-32.50)</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>(1.00, -5.00)</td>
<td>(1.60, 1.20)</td>
<td>(-3.00, 0.00)</td>
<td>(0.00, -1.00)</td>
</tr>
<tr>
<td>2</td>
<td>(0.80, -0.60)</td>
<td>(3.00, -5.00)</td>
<td>(-4.00, 3.00)</td>
<td>(-2.40, -3.20)</td>
</tr>
<tr>
<td>3</td>
<td>(1.00, 7.00)</td>
<td>(2.40, 1.80)</td>
<td>(-4.00, -5.00)</td>
<td>(0.00, -3.00)</td>
</tr>
<tr>
<td>4</td>
<td>(0.00, 1.00)</td>
<td>(-1.80, 2.40)</td>
<td>(0.00, -4.00)</td>
<td>(4.00, -5.00)</td>
</tr>
</tbody>
</table>

Generalized eigenvalues are:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(3.0000e+00, -9.0000e+00)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(2.0000e+00, -5.0000e+00)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(3.0000e+00, -1.0000e+00)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(4.0000e+00, -5.0000e+00)</td>
<td></td>
</tr>
</tbody>
</table>