NAG Library Function Document

nag_zggev (f08wnc)

1 Purpose

nag_zggev (f08wnc) computes for a pair of $n$ by $n$ complex nonsymmetric matrices $(A, B)$ the generalized eigenvalues and, optionally, the left and/or right generalized eigenvectors using the QZ algorithm.

2 Specification

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

void nag_zggev (Nag_OrderType order, Nag_LeftVecsType jobvl,
                Nag_RightVecsType jobvr, Integer n, Complex a[], Integer pda,
                Complex b[], Integer pdb, Complex alpha[], Complex beta[], Complex vl[],
                Integer pdvl, Complex vr[], Integer pdvr, NagError *fail)
```

3 Description

A generalized eigenvalue for a pair of matrices $(A, B)$ is a scalar $\lambda$ or a ratio $\alpha / \beta = \lambda$, such that $A - \lambda B$ is singular. It is usually represented as the pair $(\alpha, \beta)$, as there is a reasonable interpretation for $\beta = 0$, and even for both being zero.

The right generalized eigenvector $v_j$ corresponding to the generalized eigenvalue $\lambda_j$ of $(A, B)$ satisfies

$$Av_j = \lambda_j Bv_j.$$

The left generalized eigenvector $u_j$ corresponding to the generalized eigenvalue $\lambda_j$ of $(A, B)$ satisfies

$$u_j^H A = \lambda_j u_j^H B,$$

where $u_j^H$ is the conjugate-transpose of $u_j$.

All the eigenvalues and, if required, all the eigenvectors of the complex generalized eigenproblem $Ax = \lambda BX$, where $A$ and $B$ are complex, square matrices, are determined using the QZ algorithm. The complex QZ algorithm consists of three stages:

1. $A$ is reduced to upper Hessenberg form (with real, non-negative subdiagonal elements) and at the same time $B$ is reduced to upper triangular form.
2. $A$ is further reduced to triangular form while the triangular form of $B$ is maintained and the diagonal elements of $B$ are made real and non-negative. This is the generalized Schur form of the pair $(A, B)$. This function does not actually produce the eigenvalues $\lambda_j$, but instead returns $\alpha_j$ and $\beta_j$ such that

$$\lambda_j = \alpha_j / \beta_j, \quad j = 1, 2, \ldots, n.$$

The division by $\beta_j$ becomes your responsibility, since $\beta_j$ may be zero, indicating an infinite eigenvalue.
3. If the eigenvectors are required they are obtained from the triangular matrices and then transformed back into the original coordinate system.
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: 
  jobvl – Nag_LeftVecsType

   Input

   On entry: if jobvl = Nag_NotLeftVecs, do not compute the left generalized eigenvectors.

   If jobvl = Nag_LeftVecs, compute the left generalized eigenvectors.

   Constraint: jobvl = Nag_NotLeftVecs or Nag_LeftVecs.

3: 
  jobvr – Nag_RightVecsType

   Input

   On entry: if jobvr = Nag_NotRightVecs, do not compute the right generalized eigenvectors.

   If jobvr = Nag_RightVecs, compute the right generalized eigenvectors.

   Constraint: jobvr = Nag_NotRightVecs or Nag_RightVecs.

4: 
  n – Integer

   Input

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

   Constraint: n ≥ 0.

5: 
  a[dim] – Complex

   Input/Output

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

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

   a[(j - 1) × pda + i - 1] when order = Nag_ColMajor;

   a[(i - 1) × pda + j - 1] when order = Nag_RowMajor.

   On entry: the matrix A in the pair (A, B).

   On exit: a has been overwritten.

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 ≥ max(1, n).

7: 
  b[dim] – Complex

   Input/Output

   Note: the dimension, dim, of the array b must be at least max(1, pdb × n).
The \((i, j)\)th element of the matrix \(B\) is stored in
\[
\begin{align*}
b[(j - 1) \times \text{pdb} + i - 1] & \quad \text{when } \text{order} = \text{Nag\_ColMajor}; \\
b[(i - 1) \times \text{pdb} + j - 1] & \quad \text{when } \text{order} = \text{Nag\_RowMajor}.
\end{align*}
\]

On entry: the matrix \(B\) in the pair \((A, B)\).

On exit: \(b\) has been overwritten.

8: \textbf{pdb} – Integer

\textit{Input}

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

\textit{Constraint:} \(\text{pdb} \geq \max(1, n)\).

9: \textbf{alpha}[n] – Complex

\textit{Output}

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

10: \textbf{beta}[n] – Complex

\textit{Output}

On exit: \(\alpha[j - 1]/\beta[j - 1]\), for \(j = 1, 2, \ldots, n\), will be the generalized eigenvalues.

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

11: \textbf{vl}[\text{dim}] – Complex

\textit{Output}

\textbf{Note:} the dimension, \text{dim}, of the array \text{vl} must be at least
\[
\max(1, \text{pdvl} \times n) & \quad \text{when } \text{jobvl} = \text{Nag\_LeftVecs}; \\
1 & \quad \text{otherwise}.
\]

The \(i\)th element of the \(j\)th vector is stored in
\[
\begin{align*}
\text{vl}[(j - 1) \times \text{pdvl} + i - 1] & \quad \text{when } \text{order} = \text{Nag\_ColMajor}; \\
\text{vl}[(i - 1) \times \text{pdvl} + j - 1] & \quad \text{when } \text{order} = \text{Nag\_RowMajor}.
\end{align*}
\]

On exit: if \(\text{jobvl} = \text{Nag\_LeftVecs},\) the left generalized eigenvectors \(u_j\) are stored one after another in the columns of \text{vl}, in the same order as the corresponding eigenvalues. Each eigenvector will be scaled so the largest component will have \(|\text{real part}| + |\text{imag. part}| = 1\).

If \(\text{jobvl} = \text{Nag\_NotLeftVecs},\) \text{vl} is not referenced.

12: \textbf{pdvl} – Integer

\textit{Input}

On entry: the stride used in the array \text{vl}.

\textit{Constraints:}
\[
\text{if } \text{jobvl} = \text{Nag\_LeftVecs, } \text{pdvl} \geq \max(1, n); \\
\text{otherwise } \text{pdvl} \geq 1.
\]

13: \textbf{vr}[\text{dim}] – Complex

\textit{Output}

\textbf{Note:} the dimension, \text{dim}, of the array \text{vr} must be at least
\[
\max(1, \text{pdvr} \times n) & \quad \text{when } \text{jobvr} = \text{Nag\_RightVecs}; \\
1 & \quad \text{otherwise}.
\]

The \(i\)th element of the \(j\)th vector is stored in
\[
\begin{align*}
\text{vr}[(j - 1) \times \text{pdvr} + i - 1] & \quad \text{when } \text{order} = \text{Nag\_ColMajor}; \\
\text{vr}[(i - 1) \times \text{pdvr} + j - 1] & \quad \text{when } \text{order} = \text{Nag\_RowMajor}.
\end{align*}
\]
On exit: if \( \text{jobvr} = \text{Nag}_\text{RightVecs} \), the right generalized eigenvectors \( v_j \) are stored one after another in the columns of \( \mathbf{v}_r \), in the same order as the corresponding eigenvalues. Each eigenvector will be scaled so the largest component will have \( |\text{real part}| + |\text{imag. part}| = 1 \).

If \( \text{jobvr} = \text{Nag}_\text{NotRightVecs} \), \( \mathbf{v}_r \) is not referenced.

14: \( \text{pdvr} \) – Integer

Input

On entry: the stride used in the array \( \mathbf{v}_r \).

Constraints:

if \( \text{jobvr} = \text{Nag}_\text{RightVecs} \), \( \text{pdvr} \geq \max (1, \mathbf{n}) \);
otherwise \( \text{pdvr} \geq 1 \).

15: \( \text{fail} \) – 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 \( \langle\text{value}\rangle \) had an illegal value.

**NE_EIGENVECTORS**

A failure occurred in nag_dtgevc (f08ykc) while computing generalized eigenvectors.

**NE_ENUM_INT_2**

On entry, \( \text{jobvl} = \langle\text{value}\rangle \), \( \text{pdvl} = \langle\text{value}\rangle \) and \( \mathbf{n} = \langle\text{value}\rangle \).

Constraint: if \( \text{jobvl} = \text{Nag}_\text{LeftVecs} \), \( \text{pdvl} \geq \max (1, \mathbf{n}) \);
otherwise \( \text{pdvl} \geq 1 \).

On entry, \( \text{jobvr} = \langle\text{value}\rangle \), \( \text{pdvr} = \langle\text{value}\rangle \) and \( \mathbf{n} = \langle\text{value}\rangle \).

Constraint: if \( \text{jobvr} = \text{Nag}_\text{RightVecs} \), \( \text{pdvr} \geq \max (1, \mathbf{n}) \);
otherwise \( \text{pdvr} \geq 1 \).

**NE_INT**

On entry, \( \mathbf{n} = \langle\text{value}\rangle \).

Constraint: \( \mathbf{n} \geq 0 \).

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, \( \text{pdvl} = \langle\text{value}\rangle \).

Constraint: \( \text{pdvl} > 0 \).

On entry, \( \text{pdvr} = \langle\text{value}\rangle \).

Constraint: \( \text{pdvr} > 0 \).

**NE_INT_2**

On entry, \( \text{pda} = \langle\text{value}\rangle \) and \( \mathbf{n} = \langle\text{value}\rangle \).

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

Constraint: \( \text{pdb} \geq \max(1, \text{n}) \).

**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 \( \text{alpha} \) and \( \text{beta} \) 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.

7 **Accuracy**

The computed eigenvalues and eigenvectors are exact for a nearby matrices \( (A + E) \) and \( (B + F) \), 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.

**Note:** interpretation of results obtained with the \( QZ \) algorithm often requires a clear understanding of the effects of small changes in the original data. These effects are reviewed in Wilkinson (1979), in relation to the significance of small values of \( \alpha_j \) and \( \beta_j \). It should be noted that if \( \alpha_j \) and \( \beta_j \) are both small for any \( j \), it may be that no reliance can be placed on any of the computed eigenvalues \( \lambda_i = \alpha_i/\beta_i \). You are recommended to study Wilkinson (1979) and, if in difficulty, to seek expert advice on determining the sensitivity of the eigenvalues to perturbations in the data.

8 **Parallelism and Performance**

\text{nag_zggev (f08wnc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\text{nag_zggev (f08wnc)} 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 \text{nag_dggev (f08wac)}. 
10 Example

This example finds all the eigenvalues and right eigenvectors 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_zggev (f08wnc) Example Program.  
*  Copyright 2014 Numerical Algorithms Group.  
*  Mark 23, 2011.  
*/

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <naga02.h>
#include <nagx02.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Complex z;
    double small;
    Integer i, j, n, pda, pdb, pdvl, pdvr;
    Integer exit_status = 0;

    /* Arrays */
    Complex *a = 0, *alpha = 0, *b = 0, *beta = 0, *vl = 0, *vr = 0;
    char nag_enum_arg[40];

    /* Nag Types */
    NagError fail;
    Nag_OrderType order;
    Nag_LeftVecsType jobvl;
    Nag_RightVecsType jobvr;

    /* Define A and B using Nag column major order. */
    ifdef NAG_COLUMN_MAJOR
    #define A(I, J) a[(J-1)*pda + I - 1]
    #define B(I, J) b[(J-1)*pdb + I - 1]
    order = Nag_ColMajor;
    #else
    #define A(I, J) a[(I-1)*pda + J - 1]
    #define B(I, J) b[(I-1)*pdb + J - 1]
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);

    printf("nag_zggev (f08wnc) Example Program Results\n");

    /* Skip heading in data file */
    ifndef _WIN32

```c
scanf_s("%*[\n"]);
#else
    scanf("%*[\n"]);
#endif
#endif
#if _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;
    }
#endif
#if _WIN32
    scanf_s(" %39s%*[\n"]", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf(" %39s%*[\n"]", nag_enum_arg);
#endif
#ifdef _WIN32
    jobvl = (Nag_LeftVecsType) nag Enum name_to_value(nag_enum_arg);
#else
    jobvl = (Nag_LeftVecsType) nag Enum name_to_value(nag_enum_arg);
#endif
jobvr = (Nag_RightVecsType) nag Enum name_to_value(nag_enum_arg);
#endif
pda = n;
pdb = n;
pdvl = (jobvl==Nag_LeftVecs?n:1);
pdvr = (jobvr==Nag_RightVecs?n:1);
/* Allocate memory */
if (!(a = NAG_ALLOC(n*n, Complex)) ||
    !(alpha = NAG_ALLOC(n, Complex)) ||
    !(b = NAG_ALLOC(n*n, Complex)) ||
    !(beta = NAG_ALLOC(n, Complex)) ||
    !(vl = NAG_ALLOC(pdvl*pdvl, Complex)) ||
    !(vr = NAG_ALLOC(pdvr*pdvr, 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)
#if _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
#if _WIN32
    scanf_s("%*[\n"]);
#else
    scanf("%*[\n"]);
#endif
for (i = 1; i <= n; ++i)
    for (j = 1; j <= n; ++j)
#if _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
```
/* Solve the generalized eigenvalue problem using nag_zggev (f08wnc). */

nag_zggev(order, jobvl, jobvr, n, a, pda, b, pdb, alpha, beta, vl, pdvl, vr, pdvr, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zggev (f08wnc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_real_safe_small_number (x02amc). */

small = nag_real_safe_small_number;

printf("\n Eigenvalues\n");
for (j = 0; j < n; ++j) {
    if (nag_complex_abs(alpha[j]) * small >= nag_complex_abs(beta[j])) {
        printf("%2"NAG_IFMT" numerically infinite or undetermined\n", j+1);
        printf(" alpha = (%9.4f, %9.4f), beta = (%9.4f, %9.4f)\n",  
            alpha[j].re, alpha[j].im, beta[j].re, beta[j].im);
    } else {
        z = nag_complex_divide(alpha[j], beta[j]);
        printf("%2"NAG_IFMT" (%13.4e, %13.4e)\n", j+1, z.re, z.im);
    }
}

if (jobvl==Nag_LeftVecs) {
    printf("\n");
    /* Print left eigenvectors using nag_gen_complx_mat_print (x04dac). */
    fflush(stdout);
    nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,  
        n, vl, pdvl, " Left eigenvectors (columns)" ,
        0, &fail);
}

if ( jobvr==Nag_RightVecs && fail.code == NE_NOERROR) {
    printf("\n");
    /* Print rightt eigenvectors using nag_gen_complx_mat_print (x04dac). */
    fflush(stdout);
    nag_gen_complx_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,  
        n, vr, pdvr, " Right eigenvectors (columns)" ,
        0, &fail);
}

if (fail.code != NE_NOERROR) {
    printf("Error from nag_gen_complx_mat_print (x04dac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

END:
NAG_FREE(a);
NAG_FREE(alpha);
NAG_FREE(b);
NAG_FREE(beta);
NAG_FREE(vl);
NAG_FREE(vr);

return exit_status;
10.2 Program Data

nag_zggev (f08wnc) Example Program Data

4

: n

Nag_NotLeftVecs : jobvl
Nag_RightVecs : jobvr

(-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_zggev (f08wnc) Example Program Results

Eigenvalues
1 (3.0000e+00, -9.0000e+00)
2 (2.0000e+00, -5.0000e+00)
3 (3.0000e+00, -1.0000e+00)
4 (4.0000e+00, -5.0000e+00)

Right eigenvectors (columns)

1 2 3 4
1 -0.8238 0.6397 0.9775 -0.9062
-0.1762 0.3603 0.0225 0.0938
2 -0.1530 0.0042 0.1591 -0.0074
 0.0707 -0.0005 -0.1137 0.0069
3 -0.0707 0.0402 0.1209 0.0302
-0.1530 0.0226 -0.1537 -0.0031
4 0.1530 -0.0226 0.1537 -0.0146
-0.0707 0.0402 0.1209 -0.1410