NAG Library Function Document

nag_real_general_eigensystem (f02bjc)

1 Purpose

nag_real_general_eigensystem (f02bjc) calculates all the eigenvalues and, if required, all the eigenvectors of the generalized eigenproblem \( Ax = \lambda Bx \) where \( A \) and \( B \) are real, square matrices, using the QZ algorithm.

2 Specification

```c
#include <nag.h>
#include <nagf02.h>
void nag_real_general_eigensystem (Integer n, double a[], Integer tda, 
  double b[], Integer tdb, double tol, Complex alfa[], double beta[], 
  Nag_Boolean wantv, double v[], Integer tdv, Integer iter[], 
  NagError *fail)
```

3 Description

All the eigenvalues and, if required, all the eigenvectors of the generalized eigenproblem \( Ax = \lambda Bx \) where \( A \) and \( B \) are real, square matrices, are determined using the QZ algorithm. The QZ algorithm consists of four stages:

(a) \( A \) is reduced to upper Hessenberg form and at the same time \( B \) is reduced to upper triangular form.

(b) \( A \) is further reduced to quasi-triangular form while the triangular form of \( B \) is maintained.

(c) The quasi-triangular form of \( A \) is reduced to triangular form and the eigenvalues extracted.

(d) This function does not actually produce the eigenvalues \( \lambda_j \), but instead returns \( \alpha_j \) and \( \beta_j \) such that

\[
\lambda_j = \frac{\alpha_j}{\beta_j}, \quad j = 1, 2, \ldots, n.
\]

The division by \( \beta_j \) becomes the responsibility of your program, since \( \beta_j \) may be zero indicating an infinite eigenvalue. Pairs of complex eigenvalues occur with \( \alpha_j/\beta_j \) and \( \alpha_{j+1}/\beta_{j+1} \) complex conjugates, even though \( \alpha_j \) and \( \alpha_{j+1} \) are not conjugate.

(e) If the eigenvectors are required (\( \text{wantv} = \text{Nag_TRUE} \)), they are obtained from the triangular matrices and then transformed back into the original coordinate system.

4 References


5 Arguments

1:    n – Integer

   Input

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

   Constraint: \( n \geq 1 \).
2: \(a[n \times \text{tda}]\) – double  
**Input/Output**  
*Note*: the \((i, j)\)th element of the matrix \(A\) is stored in \(a[(i - 1) \times \text{tda} + j - 1]\).  
*On entry*: the \(n\) by \(n\) matrix \(A\).  
*On exit*: \(a\) is overwritten.

3: \(\text{tda}\) – Integer  
**Input**  
*On entry*: the stride separating matrix column elements in the array \(a\).  
*Constraint*: \(\text{tda} \geq n\).

4: \(b[n \times \text{tdb}]\) – double  
**Input/Output**  
*Note*: the \((i, j)\)th element of the matrix \(B\) is stored in \(b[(i - 1) \times \text{tdb} + j - 1]\).  
*On entry*: the \(n\) by \(n\) matrix \(B\).  
*On exit*: \(b\) is overwritten.

5: \(\text{tdb}\) – Integer  
**Input**  
*On entry*: the stride separating matrix column elements in the array \(b\).  
*Constraint*: \(\text{tdb} \geq n\).

6: \(\text{tol}\) – double  
**Input**  
*On entry*: the tolerance used to determine negligible elements.  
\(\text{tol} > 0.0\)  
An element will be considered negligible if it is less than \(\text{tol}\) times the norm of its matrix.  
\(\text{tol} \leq 0.0\)  
*machine precision* is used in place of \(\text{tol}\).  
A value of \(\text{tol}\) greater than *machine precision* may result in faster execution but less accurate results.

7: \(\text{alfa}[n]\) – Complex  
**Output**  
*On exit*: \(\alpha_j\), for \(j = 1, 2, \ldots, n\).

8: \(\text{beta}[n]\) – double  
**Output**  
*On exit*: \(\beta_j\), for \(j = 1, 2, \ldots, n\).

9: \(\text{wantv}\) – Nag_Boolean  
**Input**  
*On entry*: \(\text{wantv}\) must be set to Nag_TRUE if the eigenvectors are required. If \(\text{wantv}\) is set to Nag_FALSE then the array \(v\) is not referenced.

10: \(v[n \times \text{tdv}]\) – double  
**Output**  
*Note*: the \(i\)th element of the \(j\)th vector \(V\) is stored in \(v[(i - 1) \times \text{tdv} + j - 1]\).  
*On exit*: if \(\text{wantv} = \text{Nag_TRUE}\), then  
(i) if the \(j\)th eigenvalue is real, the \(j\)th column of \(v\) contains its eigenvector;  
(ii) if the \(j\)th and \((j + 1)\)th eigenvalues form a complex pair, the \(j\)th and \((j + 1)\)th columns of \(v\) contain the real and imaginary parts of the eigenvector associated with the first eigenvalue of the pair. The conjugate of this vector is the eigenvector for the conjugate eigenvalue.  
Each eigenvector is normalized so that the component of largest modulus is real and the sum of squares of the moduli equal one.
If \texttt{wantv} = \texttt{Nag\_FALSE}, \(v\) is not referenced and may be \texttt{NULL}.

11: \texttt{tdv} – Integer  
\textit{Input}  
\textit{On entry}: the stride separating matrix column elements in the array \(v\).  
\textit{Constraint}: if \texttt{wantv} = \texttt{Nag\_TRUE}, \(tdv \geq n\)

12: \texttt{iter}[n] – Integer  
\textit{Output}  
\textit{On exit}: \(\texttt{iter}[j-1]\) contains the number of iterations needed to obtain the \(j\)th eigenvalue. Note that the eigenvalues are obtained in reverse order, starting with the \(n\)th.

13: \texttt{fail} – NagError *  
\textit{Input/Output}  
The NAG error argument (see Section 3.6 in the Essential Introduction).

6  \textbf{Error Indicators and Warnings}

\textbf{NE\_2\_INT\_ARG\_LT}

\textit{On entry}, \texttt{tda} = \langle \texttt{value} \rangle while \(n = \langle \texttt{value} \rangle\). These arguments must satisfy \texttt{tda} \(\geq n\).

\textit{On entry}, \texttt{tdb} = \langle \texttt{value} \rangle while \(n = \langle \texttt{value} \rangle\). These arguments must satisfy \texttt{tdb} \(\geq n\).

\textit{On entry}, \texttt{tdv} = \langle \texttt{value} \rangle while \(n = \langle \texttt{value} \rangle\). These arguments must satisfy \texttt{tdv} \(\geq n\).

\textbf{NE\_INT\_ARG\_LT}

\textit{On entry}, \(n = \langle \texttt{value} \rangle\).  
\textit{Constraint}: \(n \geq 1\).

\textbf{NE\_ITERATIONS\_QZ}

More than \(n \times 30\) iterations are required to determine all the diagonal 1 by 1 or 2 by 2 blocks of the quasi-triangular form in the second step of the \textit{QZ} algorithm. This failure occurs at the \(i\)th eigenvalue, \(i = \langle \texttt{value} \rangle\). \(\alpha_j\) and \(\beta_j\) are correct for \(j = i+1, i+2, \ldots, n\) but \(v\) does not contain any correct eigenvectors.

The value of \(i\) will be returned in member \texttt{fail.errnum} of the NAG error structure provided \texttt{NAGERR\_DEFAULT} is not used as the error argument.

7  \textbf{Accuracy}

The computed eigenvalues are always exact for a problem \((A + E)x = \lambda(B + F)x\) where \(\|E\|/\|A\|\) and \(\|F\|/\|B\|\) are both of the order of max \((\texttt{tol}, \epsilon)\), \texttt{tol} being defined as in Section 5 and \(\epsilon\) being the \textit{machine precision}.

\textbf{Note}: interpretation of results obtained with the \textit{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  \textbf{Parallelism and Performance}

Not applicable.
9 Further Comments

The time taken by nag_real_general_eigensystem (f02bjc) is approximately proportional to \( n^3 \) and also depends on the value chosen for argument tol.

10 Example

To find all the eigenvalues and eigenvectors of \( Ax = \lambda Bx \) where

\[
A = \begin{pmatrix}
3.9 & 4.3 & 4.3 & 4.4 \\
12.5 & 21.5 & 21.5 & 26.0 \\
-34.5 & -47.5 & -43.5 & -46.0 \\
-0.5 & 7.5 & 3.5 & 6.0
\end{pmatrix}
\quad \text{and} \quad
B = \begin{pmatrix}
1 & 1 & 1 & 1 \\
2 & 3 & 3 & 3 \\
-3 & -5 & -4 & -4 \\
1 & 4 & 3 & 4
\end{pmatrix}.
\]

10.1 Program Text

/* nag_real_general_eigensystem (f02bjc) Example Program. *
* Copyright 2014 Numerical Algorithms Group. *
* Mark 2, 1991. *
* Mark 8 revised, 2004. */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <math.h>
#include <nagf02.h>
#include <nagx02.h>
define A(I, J) a[(I) *tda + J]
define B(I, J) b[(I) *tdb + J]
define V(I, J) v[(I) *tdv + J]

int main(void)
{
    Nag_Boolean wantv;
    Complex *alfa = 0;
    Integer exit_status = 0, i, ip, *iter = 0, j, k, n, tda, tdb, tdv;
    double *a = 0, *b = 0, *beta = 0, tol, *v = 0;
    NagError fail;

    INIT_FAIL(fail);
    printf("
    "nag_real_general_eigensystem (f02bjc) Example Program Results\n"");

#define _WIN32
    scanf_s("%*[\n"]); /* Skip heading in data file */
#else
    scanf("%*[\n"]); /* Skip heading in data file */
#endif
#define _WIN32
    scanf_s("%"NAG_IFMT", &n);
#else
    scanf("%"NAG_IFMT", &n);
#endif
    if (n >= 1)
    {
        if (!beta = NAG_ALLOC(n, double)) ||
            !a = NAG_ALLOC(n*n, double)) ||
            !b = NAG_ALLOC(n*n, double)) ||
            !v = NAG_ALLOC(n*n, double)) ||
            !(iter = NAG_ALLOC(n, Integer)) ||
            !(alfa = NAG_ALLOC(n, Complex)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
        }
goto END;
}
<tda = n;
<tdb = n;
<tdv = n;
}
else
{
    printf("Invalid n.\n\n");
    exit_status = 1;
    return exit_status;
}
for (i = 0; i < n; ++i)
for (j = 0; j < n; ++j)
#endif _WIN32
    scanf_s("%lf", &A(i, j));
#else
    scanf("%lf", &A(i, j));
#endif
for (i = 0; i < n; ++i)
for (j = 0; j < n; ++j)
#endif _WIN32
    scanf_s("%lf", &B(i, j));
#else
    scanf("%lf", &B(i, j));
#endif

wantv = Nag_TRUE;
/* nag_machine_precision (x02ajc).
 * The machine precision */
tol = nag_machine_precision;
/* nag_real_general_eigensystem (f02bjc).
 * All eigenvalues and optionally eigenvectors of real
 * generalized eigenproblem, by QZ algorithm */
nag_real_general_eigensystem(n, a, tda, b, tdb, tol,
alfa, beta, wantv, v, tdv, iter,
&fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_real_general_eigensystem (f02bjc).
%s
", fail.message);
    exit_status = 1;
    goto END;
}
ip = 0;
for (i = 0; i < n; ++i)
{
    printf("Eigensolution %4"NAG_IFMT"
", i+1);
    printf("alfa[%"NAG_IFMT"].re %7.3f", i, alfa[i].re);
    printf("beta[%"NAG_IFMT"] %7.3f\n", i, beta[i]);
    if (beta[i] == 0.0)
        printf("lambda is infinite");
    else
        if (alfa[i].im == 0.0)
            printf("lambda %7.3f\n", alfa[i].re/beta[i]);
        else
            printf("lambda %7.3f %7.3f\n",
                    alfa[i].re/beta[i], alfa[i].im/beta[i]);
            printf("Eigenvector\n");
    for (j = 0; j < n; ++j)
        printf("%7.3f\n", V(j, i));
}
else
{
    printf("lambda %7.3f \n",
            alfa[i].re/beta[i], alfa[i].im/beta[i]);
    printf("Eigenvector\n");
    k = (Integer) pow((double) -1, (double)(ip+2));
    for (j = 0; j < n; ++j)
    {

```c
    printf("%7.3f", V[j, i-ip]);
    printf("%7.3f\n", k*V[j, i-ip+1]);
}
    ip = 1-ip;
}

printf("Number of iterations (machine-dependent)\n");
for (i = 0; i < n; ++i)
    printf("%2"NAG_IFMT", iter[i]);
printf("\n");
END:
    NAG_FREE(beta);
    NAG_FREE(a);
    NAG_FREE(b);
    NAG_FREE(v);
    NAG_FREE(iter);
    NAG_FREE(alfa);
    return exit_status;
}

10.2 Program Data

nag_real_general_eigensystem (f02bjc) Example Program Data
4
  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
  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

10.3 Program Results

nag_real_general_eigensystem (f02bjc) Example Program Results

Eigensolution 1
    alfa[0].re 3.801 alfa[0].im 0.000 beta[0] 1.900
    lambda 2.000
    Eigenvector
      0.996
      0.006
      0.063
      0.063

Eigensolution 2
    alfa[1].re 1.563 alfa[1].im 2.084 beta[1] 0.521
    lambda 3.000 4.000
    Eigenvector
      0.945 0.000
      0.189 0.000
      0.113 -0.151
      0.113 -0.151

Eigensolution 3
    lambda 3.000 -4.000
    Eigenvector
      0.945 -0.000
      0.189 -0.000
      0.113 0.151
      0.113 0.151

Eigensolution 4
    alfa[3].re 4.000 alfa[3].im 0.000 beta[3] 1.000
    lambda 4.000
    Eigenvector
      0.988
```
<table>
<thead>
<tr>
<th>0.011</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.033</td>
</tr>
<tr>
<td>0.154</td>
</tr>
</tbody>
</table>

Number of iterations (machine-dependent)
0 0 5 0