NAG C Library Function Document

nag_zgebal (f08nvc)

1 Purpose

nag_zgebal (f08nvc) balances a complex general matrix in order to improve the accuracy of computed eigenvalues and/or eigenvectors.

2 Specification

```c
void nag_zgebal (Nag_OrderType order, Nag_JobType job, Integer n, Complex a[],
                Integer pda, Integer *ilo, Integer *ihi, double scale[], NagError *fail)
```

3 Description

nag_zgebal (f08nvc) balances a complex general matrix \( A \). The term ‘balancing’ covers two steps, each of which involves a similarity transformation of \( A \).

1. The function first attempts to permute \( A \) to block upper triangular form by a similarity transformation:

   \[
   PAP^T = A' = \begin{pmatrix}
   A'_{11} & A'_{12} & A'_{13} \\
   0 & A'_{22} & A'_{23} \\
   0 & 0 & A'_{33}
   \end{pmatrix}
   \]

   where \( P \) is a permutation matrix, and \( A'_{11} \) and \( A'_{33} \) are upper triangular. Then the diagonal elements of \( A'_{11} \) and \( A'_{33} \) are eigenvalues of \( A \). The rest of the eigenvalues of \( A \) are the eigenvalues of the central diagonal block \( A'_{22} \), in rows and columns \( i_{lo} \) to \( i_{hi} \). Subsequent operations to compute the eigenvalues of \( A \) (or its Schur factorization) need only be applied to these rows and columns; this can save a significant amount of work if \( i_{lo} > 1 \) and \( i_{hi} < n \). If no suitable permutation exists (as is often the case), the function sets \( i_{lo} = 1 \) and \( i_{hi} = n \), and \( A'_{22} \) is the whole of \( A \).

2. The function applies a diagonal similarity transformation to \( A' \), to make the rows and columns of \( A'_{22} \) as close in norm as possible:

   \[
   A'' = D A' D^{-1} = \begin{pmatrix}
   I & 0 & 0 \\
   0 & D_{22} & 0 \\
   0 & 0 & I
   \end{pmatrix}
   \begin{pmatrix}
   A'_{11} & A'_{12} & A'_{13} \\
   0 & A'_{22} & A'_{23} \\
   0 & 0 & A'_{33}
   \end{pmatrix}
   \begin{pmatrix}
   I & 0 & 0 \\
   0 & D_{22}^{-1} & 0 \\
   0 & 0 & I
   \end{pmatrix}
   \]

   This scaling can reduce the norm of the matrix (that is, \( \| A''_{22} \| < \| A'_{22} \| \)) and hence reduce the effect of rounding errors on the accuracy of computed eigenvalues and eigenvectors.

4 References


5 Parameters

1: \( \text{order} \) – \( 
\text{Nag OrderType} 
\)  

*Input*

On entry: the \( \text{order} \) parameter specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by \( \text{order} = \text{Nag RowMajor} \). See Section 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

Constraint: \( \text{order} = \text{Nag RowMajor} \) or \( \text{Nag ColMajor} \).
2:  \( \text{job} \) – Nag_JobType  
*Input*

*On entry:* indicates whether \( A \) is to be permuted and/or scaled (or neither), as follows:
- if \( \text{job} = \text{Nag_DoNothing} \), \( A \) is neither permuted nor scaled (but values are assigned to \( \text{ilo} \), \( \text{ihi} \) and \( \text{scale} \));
- if \( \text{job} = \text{Nag_Permute} \), \( A \) is permuted but not scaled;
- if \( \text{job} = \text{Nag_Scale} \), \( A \) is scaled but not permuted;
- if \( \text{job} = \text{Nag_DoBoth} \), \( A \) is both permuted and scaled.

*Constraint:* \( \text{job} = \text{Nag_DoNothing}, \text{Nag_Permute}, \text{Nag_Scale} \) or \( \text{Nag_DoBoth} \).

3:  \( n \) – Integer  
*Input*

*On entry:* \( n \), the order of the matrix \( A \).

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

4:  \( a[\text{dim}] \) – Complex  
*Input/Output*

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

Where \( A(i,j) \) appears in this document, it refers to the array element
- if \( \text{order} = \text{Nag_ColMajor} \), \( a[(j - 1) \times \text{pda} + i - 1] \);
- if \( \text{order} = \text{Nag_RowMajor} \), \( a[(i - 1) \times \text{pda} + j - 1] \).

*On entry:* the \( n \) by \( n \) matrix \( A \).

*On exit:* \( a \) is overwritten by the balanced matrix.

\( a \) is not referenced if \( \text{job} = \text{Nag_DoNothing} \).

5:  \( \text{pda} \) – Integer  
*Input*

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

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

6:  \( \text{ilo} \) – Integer *  

7:  \( \text{ihi} \) – Integer *  
*Output*

*On exit:* the values \( i_{lo} \) and \( i_{hi} \) such that on exit \( A(i,j) \) is zero if \( i > j \) and \( 1 \leq j < i_{lo} \) or \( i_{hi} < i \leq n \).

If \( \text{job} = \text{Nag_DoNothing} \) or \( \text{Nag_Scale} \), \( i_{lo} = 1 \) and \( i_{hi} = n \).

8:  \( \text{scale}[\text{dim}] \) – double  
*Output*

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

*On exit:* details of the permutations and scaling factors applied to \( A \). More precisely, if \( p_j \) is the index of the row and column interchanged with row and column \( j \) and \( d_j \) is the scaling factor used to balance row and column \( j \) then

\[
\text{scale}[j - 1] = \begin{cases} 
  p_j, & j = 1, 2, \ldots, i_{lo} - 1 \\
  d_j, & j = i_{lo}, i_{lo} + 1, \ldots, i_{hi} \ \text{and} \\
  p_j, & j = i_{hi} + 1, i_{hi} + 2, \ldots, n.
\end{cases}
\]

The order in which the interchanges are made is \( n \) to \( i_{hi} + 1 \) then \( 1 \) to \( i_{lo} - 1 \).

9:  \( \text{fail} \) – NagError *  
*Output*

The NAG error parameter (see the Essential Introduction).
6 Error Indicators and Warnings

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

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

**NE_ALLOC_FAIL**
Memory allocation failed.

**NE_BAD_PARAM**
On entry, parameter \( \langle \text{value} \rangle \) had an illegal value.

**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 consult NAG for assistance.

7 Accuracy
The errors are negligible, compared with those in subsequent computations.

8 Further Comments
If the matrix \( A \) is balanced by this function, then any eigenvectors computed subsequently are eigenvectors of the matrix \( A'' \) (see Section 3) and hence nag_zgebak (f08nwc) must then be called to transform them back to eigenvectors of \( A \).

If the Schur vectors of \( A \) are required, then this function must not be called with job = Nag_Scale or Nag_DoBoth, because then the balancing transformation is not unitary. If this function is called with job = Nag_Permute, then any Schur vectors computed subsequently are Schur vectors of the matrix \( A'' \), and nag_zgebak (f08nwc) must be called (with side = Nag_RightSide) to transform them back to Schur vectors of \( A \).

The total number of real floating-point operations is approximately proportional to \( n^2 \).

The real analogue of this function is nag_dgebal (f08nhc).

9 Example
To compute all the eigenvalues and right eigenvectors of the matrix \( A \), where
\[
A = \begin{pmatrix}
1.50 - 2.75i & 0.00 + 0.00i & 0.00 + 0.00i & 0.00 + 0.00i \\
-8.06 - 1.34i & -2.50 - 0.50i & 0.00 + 0.00i & -0.75 + 0.50i \\
-2.09 + 7.56i & 1.39 + 3.97i & -1.25 + 0.75i & -4.82 - 5.67i \\
6.18 + 9.79i & -0.92 - 0.62i & 0.00 + 0.00i & -2.50 - 0.50i
\end{pmatrix}
\]

The program first calls nag_zgebal (f08nvc) to balance the matrix; it then computes the Schur factorization of the balanced matrix, by reduction to Hessenberg form and the QR algorithm. Then it calls nag_ztrevc (f08qxc) to compute the right eigenvectors of the balanced matrix, and finally calls nag_zgebak (f08nwc) to transform the eigenvectors back to eigenvectors of the original matrix \( A \).
9.1 Program Text

/* nag_zgebal (f08nvc) Example Program. */
* Copyright 2001 Numerical Algorithms Group.
* Mark 7, 2001. */

#include <stdio.h>
#include <nag.h>
#include <nagf08.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer i, ihi, ilo, j, m, n, pda, pdh, pdvr;
    Integer scale_len, tau_len, w_len;
    Integer exit_status=0;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    Complex *a=0, *h=0, *tau=0, *vl=0, *vr=0, *w=0;
    double *scale=0;
    Boolean *select=0;

    #ifdef NAG_COLUMN_MAJOR
    #define A(I,J) a[(J-1)*pda+I-1]
    #define H(I,J) h[(J-1)*pdh+I-1]
    #define VR(I,J) vr[(J-1)*pdvr+I-1]
    order = Nag_ColMajor;
    #else
    #define A(I,J) a[(I-1)*pda+J-1]
    #define H(I,J) h[(I-1)*pdh+J-1]
    #define VR(I,J) vr[(I-1)*pdvr+J-1]
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);
    Vprintf("f08nvc Example Program Results\n\n");
    /* Skip heading in data file */
    Vscanf("%*\[^
\] ");
    Vscanf("%ld%*\[^
\] ", &n);
    #ifdef NAG_COLUMN_MAJOR
    pda = n;
    pdh = n;
    pdvr = n;
    #else
    pda = n;
    pdh = n;
    pdvr = n;
    #endif
    scale_len = n;
    tau_len = n;
    w_len = n;

    /* Allocate memory */
    if ( !(a = NAG_ALLOC(n * n, Complex)) ||
        !(h = NAG_ALLOC(n * n, Complex)) ||
        !(scale = NAG_ALLOC(scale_len, double)) ||
        !(tau = NAG_ALLOC(tau_len, Complex)) ||
        !(vl = NAG_ALLOC(1 * 1, Complex)) ||
        !(vr = NAG_ALLOC(n * n, Complex)) ||
        !(w = NAG_ALLOC(w_len, Complex)) ||
        !(select = NAG_ALLOC(1, Boolean)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = -1;
    }
}

f08nvc.4 [NP3645/7]
/* Read A from data file */
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= n; ++j)
        Vscanf(" ( %lf , %lf )", &A(i,j).re, &A(i,j).im);
} Vscanf("%*[\n ] ");

/* Balance A */
f08nvc(order, Nag_DoBoth, n, a, pda, &ilo, &ihi, scale, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08nvc.
%s
", fail.message);
    exit_status = 1;
    goto END;
}

/* Reduce A to upper Hessenberg form H = (Q**H)*A*Q */
f08nsc(order, n, ilo, ihi, a, pda, tau, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08nsc.
%s
", fail.message);
    exit_status = 1;
    goto END;
}

/* Copy A to H and VR */
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= n; ++j)
    {
        H(i,j).re = A(i,j).re;
        H(i,j).im = A(i,j).im;
        VR(i,j).re = A(i,j).re;
        VR(i,j).im = A(i,j).im;
    }
}

/* Form Q explicitly, storing the result in VR */
f08ntc(order, n, 1, n, vr, pdvr, tau, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08ntc.
%s
", fail.message);
    exit_status = 1;
    goto END;
}

/* Calculate the eigenvalues and Schur factorization of A */
f08psc(order, Nag_Schur, Nag_UpdateZ, n, ilo, ihi, h, pdh, w, vr, pdvr, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08psc.
%s
", fail.message);
    exit_status = 1;
    goto END;
}

Vprintf( " Eigenvalues\n");
for (i = 0; i < n; ++i)
    Vprintf( " (%7.4f,%7.4f)" , w[i].re , w[i].im);
Vprintf("\n");
/* Calculate the eigenvectors of A, storing the result in VR */
f08qxc(order, Nag_RightSide, Nag_BackTransform, select, n, h, pdh, vl, l, vr, pdvr, n, &m, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08qxc.
%s
", fail.message);
    exit_status = 1;
    goto END;
}
f08nvc

NAG C Library Manual

}  

f08nwc(order, Nag_DoBoth, Nag_RightSide, n, ilo, ihi, scale, 
m, vr, pdvr, &fail);  
if (fail.code != NE_NOERROR)  
{  
  Vprintf("Error from f08nwc.\n%s\n", fail.message);  
  exit_status = 1;  
  goto END;  
}

/* Print eigenvectors */
Vprintf("\n");
x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, m, vr, pdvr, 
    Nag_BracketForm, "%7.4f", "Contents of array VR",
    Nag_IntegerLabels, 0, Nag_IntegerLabels, 0, 80, 0, 0, &fail);
if (fail.code != NE_NOERROR)  
{  
  Vprintf("Error from x04dbc.\n%s\n", fail.message);  
  exit_status = 1;  
  goto END;  
}

END:
if (a) NAG_FREE(a);
if (h) NAG_FREE(h);
if (scale) NAG_FREE(scale);
if (tau) NAG_FREE(tau);
if (vl) NAG_FREE(vl);
if (vr) NAG_FREE(vr);
if (w) NAG_FREE(w);
if (select) NAG_FREE(select);

return exit_status;

9.2 Program Data

f08nvc Example Program Data

4 :Value of N
  ( 1.50,-2.75) ( 0.00, 0.00) ( 0.00, 0.00) ( 0.00, 0.00)
(-8.06,-1.24) (-2.50,-0.50) ( 0.00, 0.00) (-0.75, 0.50)
(-2.09, 7.56) ( 1.39, 3.97) (-1.25, 0.75) (-4.82,-5.67)
( 6.18, 9.79) (-0.92,-0.62) ( 0.00, 0.00) (-2.50,-0.50) :End of matrix A

9.3 Program Results

f08nvc Example Program Results

Eigenvalues
(-1.2500, 0.7500) (-1.5000,-0.4975) (-3.5000,-0.5025) ( 1.5000,-2.7500)

Contents of array VR

<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>( 0.0000, 0.0000)</td>
<td>( 0.0000, 0.0000)</td>
<td>( 0.0000, 0.0000)</td>
<td>( 0.1452, 0.0000)</td>
</tr>
<tr>
<td>2</td>
<td>( 0.0000, 0.0000)</td>
<td>(-0.0616, 0.0413)</td>
<td>( 0.4613,-0.0000)</td>
<td>(-0.2072,-0.2450)</td>
</tr>
<tr>
<td>3</td>
<td>( 1.0000, 0.0000)</td>
<td>( 0.6032,-0.3968)</td>
<td>( 0.2983, 0.7017)</td>
<td>( 0.7768, 0.2232)</td>
</tr>
<tr>
<td>4</td>
<td>( 0.0000, 0.0000)</td>
<td>( 0.0822, 0.0000)</td>
<td>( 0.4251, 0.2850)</td>
<td>(-0.0119, 0.4372)</td>
</tr>
</tbody>
</table>