NAG C Library Function Document

nag_dgebal (f08nhc)

1 Purpose

nag_dgebal (f08nhc) balances a real general matrix in order to improve the accuracy of computed eigenvalues and/or eigenvectors.

2 Specification

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

3 Description

nag_dgebal (f08nhc) balances a real general matrix $A$. The term ‘balancing’ covers two steps, each of which involves a similarity transformation of $A$. The function can perform either or both of these steps.

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

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

   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: \hspace{1cm} order \hspace{1cm} Nag_OrderType

   \textit{Input}

   \textit{On entry}: the \texttt{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 \texttt{order = Nag_RowMajor}. See Section 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

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

[NP3645/7] f08nhc.1
2: \( \text{job} \) – Nag_JobType
   \( \text{Input} \)
   \( \text{On entry:} \) 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.
   \( \text{Constraint:} \ \text{job} = \text{Nag_DoNothing}, \text{Nag_Permute}, \text{Nag_Scale} \) or \( \text{Nag_DoBoth} \).

3: \( n \) – Integer
   \( \text{Input} \)
   \( \text{On entry:} \) \( n \), the order of the matrix \( A \).
   \( \text{Constraint:} \ n \geq 0 \).

4: \( \text{a[dim]} \) – double
   \( \text{Input/Output} \)
   \( \text{Note:} \) the dimension, \( \text{dim} \), of the array \( \text{a} \) must be at least \( \max(1, \text{pda} \times n) \).
   \( \text{Where} \ A(i,j) \ \text{appears in this document, it refers to the array element} \)
   - if \( \text{order} = \text{Nag_ColMajor} \), \( \text{a}[(j-1) \times \text{pda} + i - 1] \);
   - if \( \text{order} = \text{Nag_RowMajor} \), \( \text{a}[(i-1) \times \text{pda} + j - 1] \).
   \( \text{On entry:} \) the \( n \) by \( n \) matrix \( A \).
   \( \text{On exit:} \) the \( n \) by \( n \) matrix \( A \).
   \( \text{a} \) is not referenced if \( \text{job} = \text{Nag_DoNothing} \).

5: \( \text{pda} \) – Integer
   \( \text{Input} \)
   \( \text{On entry:} \) the stride separating matrix row or column elements (depending on the value of \( \text{order} \)) in the array \( \text{a} \).
   \( \text{Constraint:} \ \text{pda} \geq \max(1, n) \).

6: \( \text{ilo} \) – Integer
   \( \text{Output} \)

7: \( \text{ihi} \) – Integer
   \( \text{Output} \)
   \( \text{On exit:} \) 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 \( i_{hi} < i \leq n \).
   \( \text{If} \ \text{job} = \text{Nag_DoNothing} \) or \( \text{Nag_Scale} \), \( i_{lo} = 1 \) and \( i_{hi} = n \).

8: \( \text{scale[dim]} \) – double
   \( \text{Output} \)
   \( \text{Note:} \) the dimension, \( \text{dim} \), of the array \( \text{scale} \) must be at least \( \max(1, n) \).
   \( \text{On exit:} \) details of the permutations and scaling factors applied to \( A \). More precisely, if \( p_j \) is the index of the row and \( d_j \) is the scaling factor used to balance row and column \( j \) then
   \[
   \text{scale}[j-1] = \begin{cases} 
   p_j, & j = 1, 2, \ldots, i_{lo} - 1 \\
   d_j, & j = i_{lo}, i_{lo} + 1, \ldots, i_{hi} \\
   p_j, & j = i_{hi} + 1, i_{hi} + 2, \ldots, n.
   \end{cases}
   \]
   The order in which the interchanges are made is \( n \) to \( i_{hi} + 1 \) then \( 1 \) to \( i_{lo} - 1 \).

9: \( \text{fail} \) – NagError
   \( \text{Output} \)
   \( \text{The NAG error parameter (see the Essential Introduction).} \)
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.

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_dgebak (f08njc) 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 \( \text{job} = \text{Nag\_Scale} \) or \( \text{Nag\_DoBoth} \), because then the balancing transformation is not orthogonal. If this function is called with \( \text{job} = \text{Nag\_Permute} \), then any Schur vectors computed subsequently are Schur vectors of the matrix \( A'' \), and nag_dgebak (f08njc) must be called (with \( \text{side} = \text{Nag\_RightSide} \)) to transform them back to Schur vectors of \( A \).

The total number of floating-point operations is approximately proportional to \( n^2 \).
The complex analogue of this function is nag_zgebal (f08nvc).

9 Example
To compute all the eigenvalues and right eigenvectors of the matrix \( A \), where

\[
A = \begin{pmatrix}
5.14 & 0.91 & 0.00 & -32.80 \\
0.91 & 0.20 & 0.00 & 34.50 \\
1.90 & 0.80 & -0.40 & -3.00 \\
-0.33 & 0.35 & 0.00 & 0.66
\end{pmatrix}.
\]

The program first calls nag_dgebal (f08nhc) 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_dtrevc (f08qkc) to compute the right eigenvectors of the balanced matrix, and finally calls nag_dgebak (f08njc) to transform the eigenvectors back to eigenvectors of the original matrix \( A \).
/* nag_dgebal (f08nhc) Example Program.  
*  * Copyright 2001 Numerical Algorithms Group.  
*  * Mark 7, 2001.  
*/

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.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, wi_len, wr_len;
    Integer exit_status=0;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    double *a=0, *h=0, *scale=0, *tau=0, *vl=0, *vr=0, *wi=0, *wr=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("f08nhc 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;
    wi_len = n;
    wr_len = n;

    /* Allocate memory */
    if ( !(a = NAG_ALLOC(n * n, double)) ||
        !(h = NAG_ALLOC(n * n, double)) ||
        !(scale = NAG_ALLOC(scale_len, double)) ||
        !(tau = NAG_ALLOC(tau_len, double)) ||
        !(vl = NAG_ALLOC(l * l, double)) ||
        !(vr = NAG_ALLOC(n * n, double)) ||
        !(wi = NAG_ALLOC(wi_len, double)) ||
        !(wr = NAG_ALLOC(wr_len, double)) ||
        !(select = NAG_ALLOC(1, Boolean)) )
    {
        Vprintf("Allocation failure\n");
    }
exit_status = -1;
goto END;

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

/* Balance A */
f08nhc(order, Nag_DoBoth, n, a, pda, &ilo, &ihi, scale, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08nhc.\n\n", fail.message);
    exit_status = 1;
goto END;
}

/* Reduce A to upper Hessenberg form H = (Q**T)*A*Q */
f08nec(order, n, ilo, ihi, a, pda, tau, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08nec.\n\n", 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) = A(i,j);
        VR(i,j) = A(i,j);
    }
}

/* Form Q explicitly, storing the result in VR */
f08nfc(order, n, 1, n, vr, pdvr, tau, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08nfc.\n\n", fail.message);
    exit_status = 1;
goto END;
}

/* Calculate the eigenvalues and Schur factorization of A */
f08pec(order, Nag_Schur, Nag_UpdateZ, n, ilo, ihi, h, pdh,
        wr, wi, vr, pdvr, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08pec.\n\n", fail.message);
    exit_status = 1;
goto END;
}

Vprintf(" Eigenvalues\n");
for (i = 1; i <= n; ++i)
Vprintf("(\%.4f,\%.4f)\n", wr[i-1], wi[i-1]);

/* Calculate the eigenvectors of A, storing the result in VR */
f08qkc(order, Nag_RightSide, Nag_BackTransform, select, n,
        h, pdh, vl, l, vr, pdvr, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08qkc.\n\n", fail.message);
    exit_status = 1;
goto END;
}
f08njc(order, Nag_DoBoth, Nag_RightSide, n, ilo, ihi, scale,
m, vr, pdvr, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f08njc.\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 (wi) NAG_FREE(wi);
if (wr) NAG_FREE(wr);
if (select) NAG_FREE(select);
return exit_status;

9.2 Program Data

f08nhc Example Program Data

<table>
<thead>
<tr>
<th>Value of N</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>5.14 0.91 0.00 -32.80</td>
</tr>
<tr>
<td>0.91 0.20 0.00 34.50</td>
</tr>
<tr>
<td>1.90 0.80 -0.40 -3.00</td>
</tr>
<tr>
<td>-0.33 0.35 0.00 0.66</td>
</tr>
</tbody>
</table>

9.3 Program Results

f08nhc Example Program Results

Eigenvalues

- (-0.4000, 0.0000)
- (-4.0208, 0.0000)
- (3.0136, 0.0000)
- (7.0072, 0.0000)

Contents of array VR

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0000</td>
<td>-3.9281</td>
<td>-1.1688</td>
</tr>
<tr>
<td>2</td>
<td>0.0000</td>
<td>8.0000</td>
<td>-1.9812</td>
</tr>
<tr>
<td>3</td>
<td>1.0000</td>
<td>-0.4314</td>
<td>-1.0000</td>
</tr>
<tr>
<td>4</td>
<td>0.0000</td>
<td>-0.8751</td>
<td>-0.1307</td>
</tr>
</tbody>
</table>