NAG 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

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

```
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. 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:

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

   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 $i_{\mathrm{lo}} > 1$ and $i_{\mathrm{hi}} < n$. If no suitable permutation exists (as is often the case), the function sets $i_{\mathrm{lo}} = 1$ and $i_{\mathrm{hi}} = n$, 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:

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

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

4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

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: **job** – Nag_JobType *Input*

On entry: indicates whether A is to be permuted and/or scaled (or neither).

job $=$ Nag_DoNothing
: A is neither permuted nor scaled (but values are assigned to **ilo**, **ihi** and **scale**).

job $=$ Nag_Permute
: A is permuted but not scaled.

job $=$ Nag_Scale
: A is scaled but not permuted.

job $=$ Nag_DoBoth
: A is both permuted and scaled.

Constraint: **job** $=$ Nag_DoNothing, Nag_Permute, Nag_Scale or Nag_DoBoth.

3: **n** – Integer *Input*

On entry: n, the order of the matrix A.

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

4: **a**[*dim*] – Complex *Input/Output*

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

Where $\mathbf{A}(i,j)$ appears in this document, it refers to the array element

$\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ when **order** $=$ Nag_ColMajor;
$\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$ when **order** $=$ Nag_RowMajor.

On entry: the n by n matrix A.

On exit: **a** is overwritten by the balanced matrix. If **job** $=$ Nag_DoNothing, **a** is not referenced.

5: **pda** – Integer *Input*

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

Constraint: $\mathbf{pda} \geq \max(1, \mathbf{n})$.

6: **ilo** – Integer * *Output*
7: **ihi** – Integer * *Output*

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

If **job** $=$ Nag_DoNothing or Nag_Scale, $i_{\mathrm{lo}} = 1$ and $i_{\mathrm{hi}} = n$.

8: **scale**[**n**] – double *Output*

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

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

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

9: **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 $\langle$*value*$\rangle$ had an illegal value.

NE_INT

On entry, **n** $= \langle$*value*$\rangle$.
Constraint: **n** ≥ 0.

On entry, **pda** $= \langle$*value*$\rangle$.
Constraint: **pda** > 0.

NE_INT_2

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

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_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 errors are negligible, compared with those in subsequent computations.

8 Parallelism and Performance

nag_zgebal (f08nvc) is not threaded by NAG in any implementation.

nag_zgebal (f08nvc) 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

If the matrix A is balanced by nag_zgebal (f08nvc), 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).

10 Example

This example computes all the eigenvalues and right eigenvectors of the matrix A, where

$$A = \begin{pmatrix} 1.50 - 2.75i & 0.00 + 0.00i & 0.00 + 0.00i & 0.00 + 0.00i \\ -8.06 - 1.24i & -2.50 - 0.50i & 0.00 + 0.00i & -0.75 + 0.50i \\ -2.09 + 7.56i & 1.39 + 3.97i & -1.25 + 0.75i & -4.82 - 5.67i \\ 6.18 + 9.79i & -0.92 - 0.62i & 0.00 + 0.00i & -2.50 - 0.50i \end{pmatrix}.$$

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.

10.1 Program Text

```
/* nag_zgebal (f08nvc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

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

int main(void)
{
  /* Scalars */
  Integer       firstnz, 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;
  Nag_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);

  printf("nag_zgebal (f08nvc) Example Program Results\n\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n] ", &n);
#else
  scanf("%"NAG_IFMT"%*[^\n] ", &n);
#endif

  pda = n;
  pdh = n;
  pdvr = n;
  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, Nag_Boolean)))
    {
      printf("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)
#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

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

  /* Reduce A to upper Hessenberg form H = (Q**H)*A*Q */
```

```
/* nag_zgehrd (f08nsc).
 * Unitary reduction of complex general matrix to upper
 * Hessenberg form
 */
nag_zgehrd(order, n, ilo, ihi, a, pda, tau, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_zgehrd (f08nsc).\n%s\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).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 */
/* nag_zunghr (f08ntc).
 * Generate unitary transformation matrix from reduction to
 * Hessenberg form determined by nag_zgehrd (f08nsc)
 */
nag_zunghr(order, n, 1, n, vr, pdvr, tau, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_zunghr (f08ntc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Calculate the eigenvalues and Schur factorization of A */
/* nag_zhseqr (f08psc).
 * Eigenvalues and Schur factorization of complex upper
 * Hessenberg matrix reduced from complex general matrix
 */
nag_zhseqr(order, Nag_Schur, Nag_UpdateZ, n, ilo, ihi, h, pdh,
           w, vr, pdvr, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_zhseqr (f08psc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
printf(" Eigenvalues\n");
for (i = 0; i < n; ++i)
  printf(" (%7.4f,%7.4f)", w[i].re, w[i].im);
printf("\n");
/* Calculate the eigenvectors of A, storing the result in VR */
/* nag_ztrevc (f08qxc).
 * Left and right eigenvectors of complex upper triangular
 * matrix
 */
nag_ztrevc(order, Nag_RightSide, Nag_BackTransform, select, n,
           h, pdh, vl, 1, vr, pdvr, n, &m, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_ztrevc (f08qxc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
/* nag_zgebak (f08nwc).
 * Transform eigenvectors of complex balanced matrix to
 * those of original matrix supplied to nag_zgebal (f08nvc)
 */
```

```
  nag_zgebak(order, Nag_DoBoth, Nag_RightSide, n, ilo, ihi, scale,
             m, vr, pdvr, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zgebak (f08nwc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Normalize the eigenvectors */
  for(j=1; j<=m; j++)
    {
      for(i=n; i>=1; i--)
        {
          if(VR(i, j).re != 0 || VR(i, j).im != 0)
            {
              firstnz = i;
            }
        }
      for(i=n; i>=1; i--)
        {
          VR(i, j) = nag_complex_divide(VR(i, j), VR(firstnz,j));
        }
    }
  /* Print eigenvectors */
  printf("\n");
  /* nag_gen_complx_mat_print_comp (x04dbc).
   * Print complex general matrix (comprehensive)
   */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(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)
    {
      printf(
              "Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
 END:
  NAG_FREE(a);
  NAG_FREE(h);
  NAG_FREE(scale);
  NAG_FREE(tau);
  NAG_FREE(vl);
  NAG_FREE(vr);
  NAG_FREE(w);
  NAG_FREE(select);

  return exit_status;
}
```

10.2 Program Data

```
nag_zgebal (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
```

10.3 Program Results

```
nag_zgebal (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
                   1                  2                  3                  4
 1  ( 0.0000, 0.0000)  ( 0.0000, 0.0000)  ( 0.0000, 0.0000)  ( 1.0000, 0.0000)
 2  ( 0.0000, 0.0000)  ( 1.0000,-0.0000)  ( 1.0000, 0.0000)  (-1.4269,-1.6873)
 3  ( 1.0000, 0.0000)  (-9.7405,-0.0846)  ( 0.6466, 1.5212)  ( 5.3497, 1.5369)
 4  ( 0.0000, 0.0000)  (-0.9215,-0.6177)  ( 0.9215, 0.6177)  (-0.0819, 3.0107)
```