NAG Library Function Document

nag_dgeevx (f08nbc)

1 Purpose

nag_dgeevx (f08nbc) computes the eigenvalues and, optionally, the left and/or right eigenvectors for an n by n real nonsymmetric matrix A.

Optionally, it also computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors, reciprocal condition numbers for the eigenvalues, and reciprocal condition numbers for the right eigenvectors.

2 Specification

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

```
void nag_dgeevx (Nag_OrderType order, Nag_BalanceType balanc,
     Nag_LeftVecsType jobvl, Nag_RightVecsType jobvr, Nag_RCondType sense,
     Integer n, double a[], Integer pda, double wr[], double wi[],
     double vl[], Integer pdvl, double vr[], Integer pdvr, Integer *ilo,
     Integer *ihi, double scale[], double *abnrm, double rconde[],
     double rcondv[], NagError *fail)
```

3 Description

The right eigenvector v_j of A satisfies

$$Av_j = \lambda_j v_j$$

where λ_j is the jth eigenvalue of A. The left eigenvector u_j of A satisfies

$$u_j^{\mathrm{H}} A = \lambda_j u_j^{\mathrm{H}}$$

where u_j^{H} denotes the conjugate transpose of u_j.

Balancing a matrix means permuting the rows and columns to make it more nearly upper triangular, and applying a diagonal similarity transformation DAD^{-1}, where D is a diagonal matrix, with the aim of making its rows and columns closer in norm and the condition numbers of its eigenvalues and eigenvectors smaller. The computed reciprocal condition numbers correspond to the balanced matrix. Permuting rows and columns will not change the condition numbers (in exact arithmetic) but diagonal scaling will. For further explanation of balancing, see Section 4.8.1.2 of Anderson *et al.* (1999).

Following the optional balancing, the matrix A is first reduced to upper Hessenberg form by means of unitary similarity transformations, and the QR algorithm is then used to further reduce the matrix to upper triangular Schur form, T, from which the eigenvalues are computed. Optionally, the eigenvectors of T are also computed and backtransformed to those of A.

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

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: **balanc** – Nag_BalanceType *Input*

On entry: indicates how the input matrix should be diagonally scaled and/or permuted to improve the conditioning of its eigenvalues.

balanc = Nag_NoBalancing
: Do not diagonally scale or permute.

balanc = Nag_BalancePermute
: Perform permutations to make the matrix more nearly upper triangular. Do not diagonally scale.

balanc = Nag_BalanceScale
: Diagonally scale the matrix, i.e., replace A by DAD^{-1}, where D is a diagonal matrix chosen to make the rows and columns of A more equal in norm. Do not permute.

balanc = Nag_BalanceBoth
: Both diagonally scale and permute A.

Computed reciprocal condition numbers will be for the matrix after balancing and/or permuting. Permuting does not change condition numbers (in exact arithmetic), but balancing does.

Constraint: **balanc** = Nag_NoBalancing, Nag_BalancePermute, Nag_BalanceScale or Nag_BalanceBoth.

3: **jobvl** – Nag_LeftVecsType *Input*

On entry: if **jobvl** = Nag_NotLeftVecs, the left eigenvectors of A are not computed.

If **jobvl** = Nag_LeftVecs, the left eigenvectors of A are computed.

If **sense** = Nag_RCondEigVals or Nag_RCondBoth, **jobvl** must be set to **jobvl** = Nag_LeftVecs.

Constraint: **jobvl** = Nag_NotLeftVecs or Nag_LeftVecs.

4: **jobvr** – Nag_RightVecsType *Input*

On entry: if **jobvr** = Nag_NotRightVecs, the right eigenvectors of A are not computed.

If **jobvr** = Nag_RightVecs, the right eigenvectors of A are computed.

If **sense** = Nag_RCondEigVals or Nag_RCondBoth, **jobvr** must be set to **jobvr** = Nag_RightVecs.

Constraint: **jobvr** = Nag_NotRightVecs or Nag_RightVecs.

5: **sense** – Nag_RCondType *Input*

On entry: determines which reciprocal condition numbers are computed.

sense = Nag_NotRCond
: None are computed.

sense = Nag_RCondEigVals
: Computed for eigenvalues only.

sense = Nag_RCondEigVecs
: Computed for right eigenvectors only.

sense = Nag_RCondBoth
Computed for eigenvalues and right eigenvectors.

If **sense** = Nag_RCondEigVals or Nag_RCondBoth, both left and right eigenvectors must also be computed (**jobvl** = Nag_LeftVecs and **jobvr** = Nag_RightVecs).

Constraint: **sense** = Nag_NotRCond, Nag_RCondEigVals, Nag_RCondEigVecs or Nag_RCondBoth.

6: **n** – Integer *Input*

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

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

7: **a**[dim] – double *Input/Output*

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

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

$\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ 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** has been overwritten. If **jobvl** = Nag_LeftVecs or **jobvr** = Nag_RightVecs, A contains the real Schur form of the balanced version of the input matrix A.

8: **pda** – Integer *Input*

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

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

9: **wr**[dim] – double *Output*
10: **wi**[dim] – double *Output*

Note: the dimension, *dim*, of the arrays **wr** and **wi** must be at least $\max(1, \mathbf{n})$.

On exit: **wr** and **wi** contain the real and imaginary parts, respectively, of the computed eigenvalues. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.

11: **vl**[dim] – double *Output*

Note: the dimension, *dim*, of the array **vl** must be at least

$\max(1, \mathbf{pdvl} \times \mathbf{n})$ when **jobvl** = Nag_LeftVecs;
1 otherwise.

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

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

On exit: if **jobvl** = Nag_LeftVecs, the left eigenvectors u_j are stored one after another in **vl**, in the same order as their corresponding eigenvalues. If the jth eigenvalue is real, then $u_j = \mathbf{VL}(i,j)$, for $i = 1, 2, \ldots, \mathbf{n}$. If the jth and $(j+1)$st eigenvalues form a complex conjugate pair, then $u_j = \mathbf{VL}(i,j) + i \times \mathbf{VL}(i,j+1)$ and $u_{j+1} = \mathbf{VL}(i,j) - i \times \mathbf{VL}(i,j+1)$, for $i = 1, 2, \ldots, \mathbf{n}$.

If **jobvl** = Nag_NotLeftVecs, **vl** is not referenced.

12: **pdvl** – Integer *Input*

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

Constraints:

if **jobvl** $=$ Nag_LeftVecs, **pdvl** $\geq \max(1, \mathbf{n})$;
otherwise **pdvl** ≥ 1.

13: **vr**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **vr** must be at least

$\max(1, \mathbf{pdvr} \times \mathbf{n})$ when **jobvr** $=$ Nag_RightVecs;
1 otherwise.

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

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

On exit: if **jobvr** $=$ Nag_RightVecs, the right eigenvectors v_j are stored one after another in **vr**, in the same order as their corresponding eigenvalues. If the jth eigenvalue is real, then $v_j = \mathbf{VR}(i, j)$, for $i = 1, 2, \ldots, \mathbf{n}$. If the jth and $(j+1)$st eigenvalues form a complex conjugate pair, then $v_j = \mathbf{VR}(i, j) + i \times \mathbf{VR}(i, j+1)$ and $v_{j+1} = \mathbf{VR}(i, j) - i \times \mathbf{VR}(i, j+1)$, for $i = 1, 2, \ldots, \mathbf{n}$.

If **jobvr** $=$ Nag_NotRightVecs, **vr** is not referenced.

14: **pdvr** – Integer *Input*

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

Constraints:

if **jobvr** $=$ Nag_RightVecs, **pdvr** $\geq \max(1, \mathbf{n})$;
otherwise **pdvr** ≥ 1.

15: **ilo** – Integer * *Output*
16: **ihi** – Integer * *Output*

On exit: **ilo** and **ihi** are integer values determined when A was balanced. The balanced A has $a_{ij} = 0$ if $i > j$ and $j = 1, 2, \ldots, \mathbf{ilo} - 1$ or $i = \mathbf{ihi} + 1, \ldots, \mathbf{n}$.

17: **scale**[*dim*] – double *Output*

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

On exit: details of the permutations and scaling factors applied when balancing A.

If p_j is the index of the row and column interchanged with row and column j, and d_j is the scaling factor applied to row and column j, then

$\mathbf{scale}[j-1] = p_j$, for $j = 1, 2, \ldots, \mathbf{ilo} - 1$;

$\mathbf{scale}[j-1] = d_j$, for $j = \mathbf{ilo}, \ldots, \mathbf{ihi}$;

$\mathbf{scale}[j-1] = p_j$, for $j = \mathbf{ihi} + 1, \ldots, \mathbf{n}$.

The order in which the interchanges are made is **n** to $\mathbf{ihi} + 1$, then 1 to $\mathbf{ilo} - 1$.

18: **abnrm** – double * *Output*

On exit: the 1-norm of the balanced matrix (the maximum of the sum of absolute values of elements of any column).

19: **rconde**[*dim*] – double *Output*

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

On exit: $\mathbf{rconde}[j-1]$ is the reciprocal condition number of the jth eigenvalue.

20: **rcondv**[*dim*] – double *Output*

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

On exit: **rcondv**[$j - 1$] is the reciprocal condition number of the jth right eigenvector.

21: **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 ⟨*value*⟩ had an illegal value.

NE_CONVERGENCE

The QR algorithm failed to compute all the eigenvalues, and no eigenvectors or condition numbers have been computed; elements 1 to **ilo** − 1 and ⟨*value*⟩ to **n** of **wr** and **wi** contain eigenvalues which have converged.

NE_ENUM_INT_2

On entry, **jobvl** = ⟨*value*⟩, **pdvl** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **jobvl** = Nag_LeftVecs, **pdvl** $\geq \max(1, \mathbf{n})$;
otherwise **pdvl** ≥ 1.

On entry, **jobvr** = ⟨*value*⟩, **pdvr** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: if **jobvr** = Nag_RightVecs, **pdvr** $\geq \max(1, \mathbf{n})$;
otherwise **pdvr** ≥ 1.

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

On entry, **pda** = ⟨*value*⟩.
Constraint: **pda** > 0.

On entry, **pdvl** = ⟨*value*⟩.
Constraint: **pdvl** > 0.

On entry, **pdvr** = ⟨*value*⟩.
Constraint: **pdvr** > 0.

NE_INT_2

On entry, **pda** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
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 computed eigenvalues and eigenvectors are exact for a nearby matrix $(A + E)$, where

$$\|E\|_2 = O(\epsilon)\|A\|_2,$$

and ϵ is the ***machine precision***. See Section 4.8 of Anderson *et al.* (1999) for further details.

8 Parallelism and Performance

nag_dgeevx (f08nbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dgeevx (f08nbc) 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

Each eigenvector is normalized to have Euclidean norm equal to unity and the element of largest absolute value real and positive.

The total number of floating-point operations is proportional to n^3.

The complex analogue of this function is nag_zgeevx (f08npc).

10 Example

This example finds all the eigenvalues and right eigenvectors of the matrix

$$A = \begin{pmatrix} 0.35 & 0.45 & -0.14 & -0.17 \\ 0.09 & 0.07 & -0.54 & 0.35 \\ -0.44 & -0.33 & -0.03 & 0.17 \\ 0.25 & -0.32 & -0.13 & 0.11 \end{pmatrix},$$

together with estimates of the condition number and forward error bounds for each eigenvalue and eigenvector. The option to balance the matrix is used. In order to compute the condition numbers of the eigenvalues, the left eigenvectors also have to be computed, but they are not printed out in this example.

10.1 Program Text

```
/* nag_dgeevx (f08nbc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

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

```
int main(void)
{

  /* Scalars */
  double        abnrm, eps, rcnd, tol;
  Integer       exit_status = 0, i, ihi, ilo, j, n, pda, pdvr, pdvl;
  Complex       eig;

  /* Arrays */
  double        *a = 0, *rconde = 0, *rcondv = 0, *scale = 0, *vl = 0, *vr = 0;
  double        *wi = 0, *wr = 0;

  /* Nag Types */
  NagError      fail;
  Nag_OrderType order;

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

  INIT_FAIL(fail);

  printf("nag_dgeevx (f08nbc) Example Program Results\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;
  pdvr = n;
  pdvl = n;
  /* Allocate memory */
  if (!(a      = NAG_ALLOC(n * n, double)) ||
      !(rconde = NAG_ALLOC(n, double)) ||
      !(rcondv = NAG_ALLOC(n, double)) ||
      !(scale  = NAG_ALLOC(n, double)) ||
      !(vl     = NAG_ALLOC(n * n, double)) ||
      !(vr     = NAG_ALLOC(n * n, double)) ||
      !(wi     = NAG_ALLOC(n, double)) ||
      !(wr     = NAG_ALLOC(n, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read the matrix A from data file */
  for (i = 1; i <= n; ++i)
#ifdef _WIN32
    for (j = 1; j <= n; ++j) scanf_s("%lf", &A(i, j));
#else
    for (j = 1; j <= n; ++j) scanf("%lf", &A(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
```

```
  scanf("%*[^\n]");
#endif

  /* Solve the eigenvalue problem using nag_dgeevx (f08nbc). */
  nag_dgeevx(order, Nag_BalanceBoth, Nag_LeftVecs, Nag_RightVecs,
             Nag_RCondBoth, n, a, pda, wr, wi, vl, pdvl, vr, pdvr, &ilo, &ihi,
             scale, &abnrm, rconde, rcondv, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dgeevx (f08nbc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Compute the machine precision */
  eps = nag_machine_precision;
  tol = eps * abnrm;

  /* Print the eigenvalues/vectors, associated condition number and bounds. */
  for (j = 0; j < n; ++j) {
    /* Print information on jth eigenvalue */
    printf("\n\nEigenvalue %3"NAG_IFMT"%14s= ", j+1, "");
    if (wi[j] == 0.)
      printf("%12.4e\n", wr[j]);
    else
      printf("(%13.4e, %13.4e)\n", wr[j], wi[j]);

    rcnd = rconde[j];
    printf("\nReciprocal condition number = %9.1e\n", rcnd);
    if (rcnd > 0.0)
      printf("Error bound                 = %9.1e\n", tol/rcnd);
    else
      printf("Error bound is infinite\n");

    /* Normalize and print information on jth eigenvector */
    printf("\nEigenvector %2"NAG_IFMT"\n", j+1);
    if (wi[j] == 0.0)
      for (i = 0; i < n; ++i)
        printf("%29s%13.4e\n", "", VR(i, j)/ VR(n-1, j));
    else if (wi[j] > 0.)
      for (i = 0; i < n; ++i) {
        eig = nag_complex_divide(nag_complex(VR(i, j), VR(i, j + 1)),
                                 nag_complex(VR(n-1, j), VR(n-1, j + 1)));
        printf("%30s(%13.4e, %13.4e)\n", "", eig.re, eig.im);
      }
    else
      for (i = 0; i < n; ++i) {
        eig = nag_complex_divide(nag_complex(VR(i, j - 1), VR(i, j)),
                                 nag_complex(VR(n-1, j - 1), VR(n-1, j)));
        printf("%30s(%13.4e, %13.4e)\n", "", eig.re, -eig.im);
      }

    rcnd = rcondv[j];
    printf("\nReciprocal condition number = %9.1e\n", rcnd);
    if (rcnd > 0.0)
      printf("Error bound                 = %9.1e\n", tol/rcnd);
    else
      printf("Error bound is infinite\n");
  }

 END:
  NAG_FREE(a);
  NAG_FREE(rconde);
  NAG_FREE(rcondv);
  NAG_FREE(scale);
  NAG_FREE(vl);
  NAG_FREE(vr);
  NAG_FREE(wi);
  NAG_FREE(wr);
```

```
  return exit_status;
}
#undef A
#undef VR
```

10.2 Program Data

```
nag_dgeevx (f08nbc) Example Program Data

  4                         : n

  0.35   0.45  -0.14  -0.17
  0.09   0.07  -0.54   0.35
 -0.44  -0.33  -0.03   0.17
  0.25  -0.32  -0.13   0.11 : matrix A
```

10.3 Program Results

```
nag_dgeevx (f08nbc) Example Program Results


Eigenvalue   1              =   7.9948e-01

Reciprocal condition number =   9.9e-01
Error bound                 =   1.3e-16

Eigenvector  1
                                6.8519e+00
                                5.4769e+00
                               -5.6086e+00
                                1.0000e+00

Reciprocal condition number =   6.3e-01
Error bound                 =   2.1e-16


Eigenvalue   2              = (  -9.9412e-02,    4.0079e-01)

Reciprocal condition number =   7.0e-01
Error bound                 =   1.8e-16

Eigenvector  2
                              (  -2.8597e-01,    3.7670e-01)
                              (   3.7259e-01,   -7.7284e-01)
                              (   1.4377e-01,   -4.5622e-01)
                              (   1.0000e+00,    0.0000e+00)

Reciprocal condition number =   4.0e-01
Error bound                 =   3.3e-16


Eigenvalue   3              = (  -9.9412e-02,   -4.0079e-01)

Reciprocal condition number =   7.0e-01
Error bound                 =   1.8e-16

Eigenvector  3
                              (  -2.8597e-01,   -3.7670e-01)
                              (   3.7259e-01,    7.7284e-01)
                              (   1.4377e-01,    4.5622e-01)
                              (   1.0000e+00,   -0.0000e+00)

Reciprocal condition number =   4.0e-01
Error bound                 =   3.3e-16


Eigenvalue   4              =  -1.0066e-01

Reciprocal condition number =   5.7e-01
```

```
Error bound                  =   2.3e-16

Eigenvector  4
                                 1.7357e-01
                                 4.5981e-01
                                 8.2239e-01
                                 1.0000e+00

Reciprocal condition number =   3.1e-01
Error bound                  =   4.2e-16
```