NAG Library Function Document

nag_dggev (f08wac)

1 Purpose

nag_dggev (f08wac) computes for a pair of n by n real nonsymmetric matrices (A, B) the generalized eigenvalues and, optionally, the left and/or right generalized eigenvectors using the QZ algorithm.

2 Specification

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

```
void nag_dggev (Nag_OrderType order, Nag_LeftVecsType jobvl,
     Nag_RightVecsType jobvr, Integer n, double a[], Integer pda, double b[],
     Integer pdb, double alphar[], double alphai[], double beta[],
     double vl[], Integer pdvl, double vr[], Integer pdvr, NagError *fail)
```

3 Description

A generalized eigenvalue for a pair of matrices (A, B) is a scalar λ or a ratio $\alpha/\beta = \lambda$, such that $A - \lambda B$ is singular. It is usually represented as the pair (α, β), as there is a reasonable interpretation for $\beta = 0$, and even for both being zero.

The right eigenvector v_j corresponding to the eigenvalue λ_j of (A, B) satisfies

$$Av_j = \lambda_j Bv_j.$$

The left eigenvector u_j corresponding to the eigenvalue λ_j of (A, B) satisfies

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

where u_j^{H} is the conjugate-transpose of u_j.

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:

1. A is reduced to upper Hessenberg form and at the same time B is reduced to upper triangular form.
2. A is further reduced to quasi-triangular form while the triangular form of B is maintained. This is the real generalized Schur form of the pair (A, B).
3. The quasi-triangular form of A is reduced to triangular form and the eigenvalues extracted. This function does not actually produce the eigenvalues λ_j, but instead returns α_j and β_j such that

   $$\lambda_j = \alpha_j/\beta_j, \quad j = 1, 2, \ldots, n.$$

   The division by β_j becomes your responsibility, since β_j may be zero, indicating an infinite eigenvalue. Pairs of complex eigenvalues occur with α_j/β_j and α_{j+1}/β_{j+1} complex conjugates, even though α_j and α_{j+1} are not conjugate.
4. If the eigenvectors are required they are obtained from the triangular matrices and then transformed back into the original coordinate system.

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

Wilkinson J H (1979) Kronecker's canonical form and the QZ algorithm *Linear Algebra Appl.* **28** 285–303

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: **jobvl** – Nag_LeftVecsType *Input*

On entry: if **jobvl** = Nag_NotLeftVecs, do not compute the left generalized eigenvectors.

If **jobvl** = Nag_LeftVecs, compute the left generalized eigenvectors.

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

3: **jobvr** – Nag_RightVecsType *Input*

On entry: if **jobvr** = Nag_NotRightVecs, do not compute the right generalized eigenvectors.

If **jobvr** = Nag_RightVecs, compute the right generalized eigenvectors.

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

4: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

5: **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 matrix A in the pair (A, B).

On exit: **a** has been overwritten.

6: **pda** – Integer *Input*

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

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

7: **b**[dim] – double *Input/Output*

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

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

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

On entry: the matrix B in the pair (A,B).

On exit: **b** has been overwritten.

8: **pdb** – Integer *Input*

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

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

9: **alphar**[**n**] – double *Output*

On exit: the element $\mathbf{alphar}[j-1]$ contains the real part of α_j.

10: **alphai**[**n**] – double *Output*

On exit: the element $\mathbf{alphai}[j-1]$ contains the imaginary part of α_j.

11: **beta**[**n**] – double *Output*

On exit: $(\mathbf{alphar}[j-1]+\mathbf{alphai}[j-1]\times i)/\mathbf{beta}[j-1]$, for $j=1,2,\ldots,\mathbf{n}$, will be the generalized eigenvalues.

If $\mathbf{alphai}[j-1]$ is zero, then the jth eigenvalue is real; if positive, then the jth and $(j+1)$st eigenvalues are a complex conjugate pair, with $\mathbf{alphai}[j]$ negative.

Note: the quotients $\mathbf{alphar}[j-1]/\mathbf{beta}[j-1]$ and $\mathbf{alphai}[j-1]/\mathbf{beta}[j-1]$ may easily overflow or underflow, and $\mathbf{beta}[j-1]$ may even be zero. Thus, you should avoid naively computing the ratio α_j/β_j. However, $\max(|\alpha_j|)$ will always be less than and usually comparable with $\|\mathbf{a}\|_2$ in magnitude, and $\max(|\beta_j|)$ will always be less than and usually comparable with $\|\mathbf{b}\|_2$.

12: **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 the columns of **vl**, in the same order as the corresponding eigenvalues.

If the jth eigenvalue is real, then $u_j=\mathbf{VL}(:,j)$, the jth column of **vl**.

If the jth and $(j+1)$th eigenvalues form a complex conjugate pair, then $u_j=\mathbf{VL}(:,j)+i\times\mathbf{VL}(:,j+1)$ and $u(j+1)=\mathbf{VL}(:,j)-i\times\mathbf{VL}(:,j+1)$. Each eigenvector will be scaled so the largest component has $|\text{real part}|+|\text{imag. part}|=1$.

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

13: **pdvl** – Integer *Input*

On entry: the stride used in the array **vl**.

Constraints:

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

14: **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 the columns of **vr**, in the same order as the corresponding eigenvalues.

If the jth eigenvalue is real, then $v_j = \mathbf{VR}(:,j)$, the jth column of VR.

If the jth and $(j+1)$th eigenvalues form a complex conjugate pair, then $v_j = \mathbf{VR}(:,j) + i \times \mathbf{VR}(:,j+1)$ and $v_{j+1} = \mathbf{VR}(:,j) - i \times \mathbf{VR}(:,j+1)$. Each eigenvector will be scaled so the largest component has $|\text{real part}| + |\text{imag. part}| = 1$.

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

15: **pdvr** – Integer *Input*

On entry: the stride used in the array **vr**.

Constraints:

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

16: **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.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_EIGENVECTORS

A failure occurred in nag_dtgevc (f08ykc) while computing generalized eigenvectors.

NE_ENUM_INT_2

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

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

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: $\mathbf{n} \geq 0$.

On entry, **pda** = ⟨*value*⟩.
Constraint: $\mathbf{pda} > 0$.

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

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

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

NE_INT_2

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

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

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.

NE_ITERATION_QZ

The QZ iteration failed. No eigenvectors have been calculated but **alphar**$[j]$, **alphai**$[j]$ and **beta**$[j]$ should be correct from element $\langle value \rangle$.

The QZ iteration failed with an unexpected error, please contact NAG.

7 Accuracy

The computed eigenvalues and eigenvectors are exact for a nearby matrices $(A + E)$ and $(B + F)$, where

$$\|(E, F)\|_F = O(\epsilon)\|(A, B)\|_F,$$

and ϵ is the ***machine precision***. See Section 4.11 of Anderson *et al.* (1999) for further details.

Note: interpretation of results obtained with the 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 α_j and β_j. It should be noted that if α_j and β_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 Parallelism and Performance

nag_dggev (f08wac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dggev (f08wac) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

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

The complex analogue of this function is nag_zggev (f08wnc).

10 Example

This example finds all the eigenvalues and right eigenvectors of the matrix pair (A, B), where

$$A = \begin{pmatrix} 3.9 & 12.5 & -34.5 & -0.5 \\ 4.3 & 21.5 & -47.5 & 7.5 \\ 4.3 & 21.5 & -43.5 & 3.5 \\ 4.4 & 26.0 & -46.0 & 6.0 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 1.0 & 2.0 & -3.0 & 1.0 \\ 1.0 & 3.0 & -5.0 & 4.0 \\ 1.0 & 3.0 & -4.0 & 3.0 \\ 1.0 & 3.0 & -4.0 & 4.0 \end{pmatrix}.$$

10.1 Program Text

```
/* nag_dggev (f08wac) Example Program.
 *
 * Copyright 2011 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <math.h>
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagx02.h>
#include <nagx04.h>
#include <naga02.h>

int main(void)
{
  /* Scalars */
  Complex           eig, eigl, eigr;
  double            sign, small;
  Integer           i, j, k, n, pda, pdb, pdvl, pdvr;
  Integer           exit_status = 0;

  /* Arrays */
  double            *a = 0, *alphai = 0, *alphar = 0, *b = 0, *beta = 0;
  double            *vl = 0, *vr = 0;
  char              nag_enum_arg[40];

  /* Nag Types */
  NagError          fail;
  Nag_OrderType     order;
  Nag_LeftVecsType  jobvl;
  Nag_RightVecsType jobvr;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J)  a[(J-1)*pda + I - 1]
#define B(I, J)  b[(J-1)*pdb + I - 1]
#define VL(I, J) vl[(J-1)*pdvl + 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 B(I, J)  b[(I-1)*pdb + J - 1]
#define VL(I, J) vl[(I-1)*pdvl + J - 1]
#define VR(I, J) vr[(I-1)*pdvr + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dggev (f08wac) Example Program Results\n");

  /* Skip heading in data file */
  scanf("%*[^\n]");
  scanf("%ld%*[^\n]", &n);
  if (n < 0)
    {
```

```
      printf("Invalid n\n");
      exit_status = 1;
      goto END;
    }
  scanf(" %39s%*[^\n]", nag_enum_arg);
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  jobvl = (Nag_LeftVecsType) nag_enum_name_to_value(nag_enum_arg);
  scanf(" %39s%*[^\n]", nag_enum_arg);
  jobvr = (Nag_RightVecsType) nag_enum_name_to_value(nag_enum_arg);
  pda = n;
  pdb = n;
  pdvl = (jobvl==Nag_LeftVecs?n:1);
  pdvr = (jobvr==Nag_RightVecs?n:1);

  /* Allocate memory */
  if (!(a      = NAG_ALLOC(n*n, double)) ||
      !(alphai = NAG_ALLOC(n, double)) ||
      !(alphar = NAG_ALLOC(n, double)) ||
      !(b      = NAG_ALLOC(n*n, double)) ||
      !(beta   = NAG_ALLOC(n, double)) ||
      !(vl     = NAG_ALLOC(pdvl*pdvl, double)) ||
      !(vr     = NAG_ALLOC(pdvr*pdvr, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read in the matrices A and B */
  for (i = 1; i <= n; ++i)
    for (j = 1; j <= n; ++j) scanf("%lf", &A(i, j));
  scanf("%*[^\n]");
  for (i = 1; i <= n; ++i)
    for (j = 1; j <= n; ++j) scanf("%lf", &B(i, j));
  scanf("%*[^\n]");

  /* Solve the generalized eigenvalue problem */
  nag_dggev(order, jobvl, jobvr, n, a, pda, b, pdb, alphar, alphai, beta, vl,
            pdvl, vr, pdvr, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_dggev (f08wac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  small = nag_real_safe_small_number;
  for (j = 0; j < n; ++j)
    {
      printf("\n");
      if ((fabs(alphar[j]) + fabs(alphai[j])) * small >= fabs(beta[j]))
        {
          printf("Eigenvalue %2ld is numerically infinite or "
                 "undetermined\n", j+1);
          printf("alpha = (%13.4e, %13.4e), beta = %13.4e\n", alphar[j],
                 alphai[j], beta[j]);
        }
      else if (alphai[j] == 0.0)
        {
          printf("Eigenvalue %2ld = %13.4e\n", j+1, alphar[j]/beta[j]);
        }
      else
        {
          eig.re = alphar[j]/beta[j], eig.im = alphai[j]/beta[j];
          printf("Eigenvalue %2ld = (%13.4e, %13.4e)\n", j+1, eig.re,
                 eig.im);
        }
      printf("\n");
      if (jobvl==Nag_LeftVecs) printf("%20s%8s", "Left Eigenvector", "");
```

```
      if (jobvr==Nag_RightVecs) printf("%20s", "Right Eigenvector");
      printf(" %2ld\n", j+1);
      if (alphai[j] == 0.0)
        for (i = 1; i <= n; ++i) {
          if (jobvl==Nag_LeftVecs)
            printf("%6s%13.4e%12s", "", VL(i, j+1)/VL(n, j+1), "");
          if (jobvr==Nag_RightVecs)
            printf("%6s%13.4e", "", VR(i, j+1)/VR(n, j+1));
          printf("\n");
        }
      else
        {
          k = (alphai[j]>0.0?j+1:j);
          sign = (alphai[j]>0.0?1.0:-1.0);
          if (jobvl==Nag_LeftVecs) eigl = nag_complex(VL(n,k), VL(n,k+1));
          if (jobvr==Nag_RightVecs) eigr = nag_complex(VR(n,k), VR(n,k+1));
          for (i = 1; i <= n; ++i)
            {
              if (jobvl==Nag_LeftVecs) {
                eig = nag_complex_divide(nag_complex(VL(i,k), VL(i,k+1)),
                                         eigl);
                printf("(%13.4e,%13.4e) ", eig.re, sign*eig.im);
              }
              if (jobvr==Nag_RightVecs) {
                eig = nag_complex_divide(nag_complex(VR(i,k), VR(i,k+1)),
                                         eigr);
                printf("(%13.4e,%13.4e)", eig.re, sign*eig.im);
              }
              printf("\n");
            }
        }
    }

 END:
  NAG_FREE(a);
  NAG_FREE(alphai);
  NAG_FREE(alphar);
  NAG_FREE(b);
  NAG_FREE(beta);
  NAG_FREE(vl);
  NAG_FREE(vr);

  return exit_status;
}
```

10.2 Program Data

```
nag_dggev (f08wac) Example Program Data
   4                    : n

   Nag_NotLeftVecs      : jobvl
   Nag_RightVecs        : jobvr

   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 : matrix A
   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 : matrix B
```

10.3 Program Results

```
nag_dggev (f08wac) Example Program Results

Eigenvalue  1 =    2.0000e+00

   Right Eigenvector  1
         1.5909e+01
         9.0909e-02
         1.0000e+00
         1.0000e+00

Eigenvalue  2 = (   3.0000e+00,    4.0000e+00)

   Right Eigenvector  2
(   3.0000e+00,   4.0000e+00)
(   6.0000e-01,   8.0000e-01)
(   1.0000e+00,  -7.6096e-17)
(   1.0000e+00,  -0.0000e+00)

Eigenvalue  3 = (   3.0000e+00,   -4.0000e+00)

   Right Eigenvector  3
(   3.0000e+00,  -4.0000e+00)
(   6.0000e-01,  -8.0000e-01)
(   1.0000e+00,   7.6096e-17)
(   1.0000e+00,   0.0000e+00)

Eigenvalue  4 =    4.0000e+00

   Right Eigenvector  4
         6.4286e+00
         7.1429e-02
        -2.1429e-01
         1.0000e+00
```
