NAG Library Function Document

nag_dggev3 (f08wcc)

1 Purpose

nag_dggev3 (f08wcc) 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_dggev3 (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 (2012) *Matrix Computations* (4th 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 2.3.1.3 in How to Use the NAG Library and its Documentation 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

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

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

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

On exit: the element **alphar**$[j-1]$ contains the real part of α_j.

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

On exit: the element **alphai**$[j-1]$ contains the imaginary part of α_j.

11: **beta**[**n**] – double *Output*

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

If **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 **alphai**$[j]$ negative.

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

12: **vl**[*dim*] – double *Output*

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

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

Where **VL**(i,j) appears in this document, it refers to the array element

vl$[(j-1) \times$ **pdvl** $+ i - 1]$ when **order** = Nag_ColMajor;
vl$[(i-1) \times$ **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 $u_j =$ **VL**$(:, j)$, the jth column of **vl**.

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

14: **vr**[*dim*] – double *Output*

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

max(1, **pdvr** × **n**) when **jobvr** = Nag_RightVecs;
1 otherwise.

Where **VR**(i, j) appears in this document, it refers to the array element

vr$[(j-1) \times$ **pdvr** $+ i - 1]$ when **order** = Nag_ColMajor;
vr$[(i-1) \times$ **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 $v_j =$ **VR**$(:, j)$, the jth column of VR.

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

16: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

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, **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, **pdb** = ⟨*value*⟩.
Constraint: **pdb** > 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** $\geq \max(1, \mathbf{n})$.

On entry, **pdb** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

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 ⟨*value*⟩.

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

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

The computed eigenvalues and eigenvectors are exact for 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_dggev3 (f08wcc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dggev3 (f08wcc) 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

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

The complex analogue of this function is nag_zggev3 (f08wqc).

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_dggev3 (f08wcc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <math.h>
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagx02.h>
#include <nagx04.h>
#include <nagm01.h>
#include <naga02.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static Integer NAG_CALL compare(const Nag_Pointer a, const Nag_Pointer b);
  static Integer normalize_vectors(Integer n, double alphai[], double v[],
                                   Complex w[], size_t rank[],
                                   const char *title);
  static Integer sort_values (Integer n, Complex alpha[], size_t rank[],
                              double temp[]);
#ifdef __cplusplus
}
#endif

int main(void)
{
  /* Scalars */
  Integer           i, isinf, j, n, pda, pdb, pdvl, pdvr;
  Integer           exit_status = 0;

  /* Arrays */
  Complex           *eval=0, *evec=0;
  double            *a = 0, *alphai = 0, *alphar = 0, *b = 0, *beta = 0;
  double            *vl = 0, *vr = 0, *ea = 0;
  char              nag_enum_arg[40];
  size_t            *rank = 0;
```

```
  /* 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]
  order = Nag_ColMajor;
#else
#define A(I, J)  a[(I-1)*pda + J - 1]
#define B(I, J)  b[(I-1)*pdb + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dggev3 (f08wcc) 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
  if (n < 0) {
    printf("Invalid n\n");
    exit_status = 1;
    goto END;
  }
#ifdef _WIN32
  scanf_s(" %39s%*[^\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf(" %39s%*[^\n]", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  jobvl = (Nag_LeftVecsType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s(" %39s%*[^\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf(" %39s%*[^\n]", nag_enum_arg);
#endif
  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)) ||
      !(ea = NAG_ALLOC(n, double)) ||
      !(eval = NAG_ALLOC(n, Complex)) ||
      !(evec = NAG_ALLOC(n * n, Complex)) ||
      !(rank = NAG_ALLOC(n, size_t)))
   {
    printf("Allocation failure\n");
    exit_status = -1;
```

```
    goto END;
  }

  /* Read in the matrices A and B */
  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
  for (i = 1; i <= n; ++i)
#ifdef _WIN32
    for (j = 1; j <= n; ++j)
      scanf_s("%lf", &B(i, j));
#else
    for (j = 1; j <= n; ++j)
      scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Solve the generalized eigenvalue problem Ax = lambda Bx using the
   * level 3 blocked routine nag_dggev3 (f08wcc) which returns:
   *  - eigenvalues as (alphar[] + i*alphai[])./beta[];
   *  - left and right eigenvectors in vl and vr respectively.
   */
  nag_dggev3(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_dggev3 (f08wcc).\n%s\n", fail.message);
    exit_status = 2;
    goto END;
  }

  isinf = 0;
  for (j = 0; j < n; ++j) {
    /* Check for infinite, real and complex eigenvalues in that order */
    if (fabs(beta[j]) < x02ajc()) {
      isinf  = j + 1;
    } else {
      eval[j].re = alphar[j]/beta[j];
      eval[j].im = alphai[j]/beta[j] + 10.0*x02ajc();
    }
  }
  if (isinf) {
    printf("Eigenvalue %2" NAG_IFMT " is numerically infinite.\n",isinf);
  } else {
    /* Print the ordered (finite) eigenvalues. */
    exit_status = sort_values(n, eval, rank, ea);
    if (exit_status) {
      goto END;
    }
  }

  if (jobvl == Nag_LeftVecs) {
    /* Normalize and print the left eigenvectors */
    exit_status = normalize_vectors(n, alphai, vl, evec, rank,
                                    "Left eigenvectors:");
    printf("\n");
  }
  if (jobvr == Nag_RightVecs) {
```

```
    /* Normalize and print the right eigenvectors */
    exit_status = normalize_vectors(n, alphai, vr, evec, rank,
                                    "Right eigenvectors:");
    printf("\n");
  }

END:
  NAG_FREE(a);
  NAG_FREE(alphai);
  NAG_FREE(alphar);
  NAG_FREE(b);
  NAG_FREE(beta);
  NAG_FREE(vl);
  NAG_FREE(vr);
  NAG_FREE(ea);
  NAG_FREE(eval);
  NAG_FREE(evec);
  NAG_FREE(rank);

  return exit_status;
}

static Integer normalize_vectors(Integer n, double alphai[], double v[],
                                 Complex w[], size_t rank[],
                                 const char *title)
{
  /* Each complex eigenvector is normalized so that the element of largest
   * magnitude is scaled to be real and positive.
   */

  Complex   scal;
  double    r, rr;
  Integer   i, j, jj, k, errors = 0;
  NagError  fail;

  INIT_FAIL(fail);

#ifdef NAG_COLUMN_MAJOR
#define V(I, J) v[(J-1)*n + I - 1]
#else
#define V(I, J) v[(I-1)*n + J - 1]
#endif
#define W(I, J) w[(I-1)*n + J - 1]

  /* Re-normalize the eigenvectors, largest absolute element real. */
  k = 0;
  for (i = 1; i<=n; i++) {
    if (fabs(alphai[i-1])<x02ajc()) {
      jj = 1;
      r = 0.0;
      for (j = 1; j <= n; j++) {
        W(j,i).re = V(j,i);
        W(j,i).im = 0.0;
        rr = fabs(V(j,i));
        if (rr>r) {
          r = rr;
          jj = j;
        }
      }
    } else if (k==0) {
      jj = 1;
      r = 0.0;
      for (j = 1; j <= n; j++) {
        W(j,i).re = V(j,i);
        W(j,i).im = V(j,i+1);
        rr = sqrt(V(j,i)*V(j,i) + V(j,i+1)*V(j,i+1));
        if (rr>r) {
          r = rr;
          jj = j;
        }
      }
```

```
      k = 1;
    } else {
      for (j = 1; j <= n; j++) {
        W(j,i) = nag_complex_conjg(W(j,i-1));
      }
      k = 0;
    }
    scal = nag_complex_conjg(W(jj,i));
    scal.re = scal.re/r;
    scal.im = scal.im/r;
    for (j = 1; j <= n; j++) {
      /* nag_complex_multiply (a02ccc), multiply two complex numbers */
      W(j,i) = nag_complex_multiply(W(j,i),scal);
    }
  }
  for (j = 1; j <=n; j++) {
    /* Sort eigenvectors by eigenvalue rank using
     * nag_reorder_vector (m01esc).
     */
    nag_reorder_vector((Pointer) &W(j,1), (size_t) n, sizeof(Complex),
                       (ptrdiff_t) sizeof(Complex), rank, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_reorder_vector (m01esc).\n%s\n", fail.message);
      errors = 2;
      goto END;
    }
  }
  /* Print the normalized eigenvectors using
   * nag_gen_complx_mat_print_comp (x04dbc)
   */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(Nag_RowMajor, Nag_GeneralMatrix,
                                Nag_NonUnitDiag,
                                n, n, w, n, Nag_BracketForm, "%7.4f",
                                title, Nag_NoLabels, 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);
    errors = 3;
  }
END:
#undef V
#undef W
  return errors;
}

static Integer sort_values (Integer n, Complex vec[], size_t rank[],
                            double temp[])
{
  Integer  i, errors = 0;
  NagError fail;

  INIT_FAIL(fail);

  /* Accumulate eigenvalue modulii in temp. */
  for (i = 0; i < n; ++i) {
    /* nag_complex_abs (a02cdc) - modulus of complex number. */
    temp[i] = nag_complex_abs(vec[i]);
  }
  /* Rank sort eigenvalues by absolute values using nag_rank_sort (m01dsc). */
  nag_rank_sort((Pointer) temp, (size_t) n, (ptrdiff_t) (sizeof(double)),
                compare, Nag_Descending, rank, &fail);
  /* Turn ranks into indices using nag_make_indices (m01zac). */
  nag_make_indices(rank, (size_t) n, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_make_indices (m01zac).\n%s\n", fail.message);
    errors = 1;
    goto END;
  }
  /* Sort eigenvalues using nag_reorder_vector (m01esc). */
```

```
  nag_reorder_vector((Pointer) vec, (size_t) n, sizeof(Complex),
                     (ptrdiff_t) sizeof(Complex), rank, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_reorder_vector (m01esc).\n%s\n", fail.message);
    errors = 2;
    goto END;
  }
  printf("\n Eigenvalues:\n");
  for (i = 0; i < n; ++i) {
    printf(" %4" NAG_IFMT "     (%7.3f,%7.3f)\n", i + 1, vec[i].re, vec[i].im);
  }
  printf("\n");
 END:
  return errors;
}

static Integer NAG_CALL compare(const Nag_Pointer a, const Nag_Pointer b)
{
  double x = *((const double *) a) - *((const double *) b);
  return (x < 0.0 ? -1 : (x == 0.0 ? 0 : 1));
}
```

10.2 Program Data

```
nag_dggev3 (f08wcc) 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_dggev3 (f08wcc) Example Program Results

 Eigenvalues:
    1     (  3.000,  4.000)
    2     (  3.000, -4.000)
    3     (  4.000,  0.000)
    4     (  2.000,  0.000)

 Right eigenvectors:
                  1                   2                   3                   4
   ( 0.7122, 0.0000)  ( 0.7122, 0.0000)  ( 1.0000, 0.0000)  ( 1.0000, 0.0000)
   ( 0.1424, 0.0000)  ( 0.1424,-0.0000)  ( 0.0111, 0.0000)  ( 0.0057, 0.0000)
   ( 0.0855,-0.1140)  ( 0.0855, 0.1140)  (-0.0333,-0.0000)  ( 0.0629, 0.0000)
   ( 0.0855,-0.1140)  ( 0.0855, 0.1140)  ( 0.1556, 0.0000)  ( 0.0629, 0.0000)
```