NAG Library Function Document

nag_dsygv (f08sac)

1 Purpose

nag_dsygv (f08sac) computes all the eigenvalues and, optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem, of the form

$$Az = \lambda Bz, \quad ABz = \lambda z \quad \text{or} \quad BAz = \lambda z,$$

where A and B are symmetric and B is also positive definite.

2 Specification

```
#include <nag.h>
#include <nagf08.h>
void nag_dsygv (Nag_OrderType order, Integer itype, Nag_JobType job,
     Nag_UploType uplo, Integer n, double a[], Integer pda, double b[],
     Integer pdb, double w[], NagError *fail)
```

3 Description

nag_dsygv (f08sac) first performs a Cholesky factorization of the matrix B as $B = U^{\mathrm{T}}U$, when **uplo** = Nag_Upper or $B = LL^{\mathrm{T}}$, when **uplo** = Nag_Lower. The generalized problem is then reduced to a standard symmetric eigenvalue problem

$$Cx = \lambda x,$$

which is solved for the eigenvalues and, optionally, the eigenvectors; the eigenvectors are then backtransformed to give the eigenvectors of the original problem.

For the problem $Az = \lambda Bz$, the eigenvectors are normalized so that the matrix of eigenvectors, z, satisfies

$$Z^{\mathrm{T}}AZ = \Lambda \quad \text{and} \quad Z^{\mathrm{T}}BZ = I,$$

where Λ is the diagonal matrix whose diagonal elements are the eigenvalues. For the problem $ABz = \lambda z$ we correspondingly have

$$Z^{-1}AZ^{-\mathrm{T}} = \Lambda \quad \text{and} \quad Z^{\mathrm{T}}BZ = I,$$

and for $BAz = \lambda z$ we have

$$Z^{\mathrm{T}}AZ = \Lambda \quad \text{and} \quad Z^{\mathrm{T}}B^{-1}Z = I.$$

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: **itype** – Integer *Input*

On entry: specifies the problem type to be solved.

itype = 1
$Az = \lambda Bz$.

itype = 2
$ABz = \lambda z$.

itype = 3
$BAz = \lambda z$.

Constraint: **itype** = 1, 2 or 3.

3: **job** – Nag_JobType *Input*

On entry: indicates whether eigenvectors are computed.

job = Nag_EigVals
Only eigenvalues are computed.

job = Nag_DoBoth
Eigenvalues and eigenvectors are computed.

Constraint: **job** = Nag_EigVals or Nag_DoBoth.

4: **uplo** – Nag_UploType *Input*

On entry: if **uplo** = Nag_Upper, the upper triangles of A and B are stored.

If **uplo** = Nag_Lower, the lower triangles of A and B are stored.

Constraint: **uplo** = Nag_Upper or Nag_Lower.

5: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

6: **a**$[dim]$ – double *Input/Output*

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

On entry: the n by n symmetric matrix A.

If **order** = 'Nag_ColMajor', A_{ij} is stored in **a**$[(j-1) \times \textbf{pda} + i - 1]$.

If **order** = 'Nag_RowMajor', A_{ij} is stored in **a**$[(i-1) \times \textbf{pda} + j - 1]$.

If **uplo** = 'Nag_Upper', the upper triangular part of A must be stored and the elements of the array below the diagonal are not referenced.

If **uplo** = 'Nag_Lower', the lower triangular part of A must be stored and the elements of the array above the diagonal are not referenced.

On exit: if **job** = Nag_DoBoth, **a** contains the matrix Z of eigenvectors. The eigenvectors are normalized as follows:

if **itype** $= 1$ or 2, $Z^{\mathrm{T}}BZ = I$;

if **itype** $= 3$, $Z^{\mathrm{T}}B^{-1}Z = I$.

If **job** $=$ Nag_EigVals, the upper triangle (if **uplo** $=$ Nag_Upper) or the lower triangle (if **uplo** $=$ Nag_Lower) of **a**, including the diagonal, is overwritten.

7: **pda** – Integer *Input*

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

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

8: **b**$[dim]$ – double *Input/Output*

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

On entry: the n by n symmetric positive definite matrix B.

If **order** $=$ 'Nag_ColMajor', B_{ij} is stored in **b**$[(j-1) \times \mathbf{pdb} + i - 1]$.

If **order** $=$ 'Nag_RowMajor', B_{ij} is stored in **b**$[(i-1) \times \mathbf{pdb} + j - 1]$.

If **uplo** $=$ 'Nag_Upper', the upper triangular part of B must be stored and the elements of the array below the diagonal are not referenced.

If **uplo** $=$ 'Nag_Lower', the lower triangular part of B must be stored and the elements of the array above the diagonal are not referenced.

On exit: if **fail.code** $=$ NE_NOERROR or NE_CONVERGENCE, the part of **b** containing the matrix is overwritten by the triangular factor U or L from the Cholesky factorization $B = U^{\mathrm{T}}U$ or $B = LL^{\mathrm{T}}$.

9: **pdb** – Integer *Input*

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

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

10: **w**$[\mathbf{n}]$ – double *Output*

On exit: the eigenvalues in ascending order.

11: **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_CONVERGENCE

The algorithm failed to converge; ⟨*value*⟩ off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

NE_INT

On entry, **itype** $= \langle value \rangle$.
Constraint: **itype** $= 1$, 2 or 3.

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

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

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

If **fail.errnum** $= \mathbf{n} + \langle value \rangle$, for $1 \leq \langle value \rangle \leq \mathbf{n}$, then the leading minor of order $\langle value \rangle$ of B is not positive definite. The factorization of B could not be completed and no eigenvalues or eigenvectors were computed.

7 Accuracy

If B is ill-conditioned with respect to inversion, then the error bounds for the computed eigenvalues and vectors may be large, although when the diagonal elements of B differ widely in magnitude the eigenvalues and eigenvectors may be less sensitive than the condition of B would suggest. See Section 4.10 of Anderson *et al.* (1999) for details of the error bounds.

The example program below illustrates the computation of approximate error bounds.

8 Parallelism and Performance

nag_dsygv (f08sac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dsygv (f08sac) 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_zhegv (f08snc).

10 Example

This example finds all the eigenvalues and eigenvectors of the generalized symmetric eigenproblem $Az = \lambda Bz$, where

$$A = \begin{pmatrix} 0.24 & 0.39 & 0.42 & -0.16 \\ 0.39 & -0.11 & 0.79 & 0.63 \\ 0.42 & 0.79 & -0.25 & 0.48 \\ -0.16 & 0.63 & 0.48 & -0.03 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 4.16 & -3.12 & 0.56 & -0.10 \\ -3.12 & 5.03 & -0.83 & 1.09 \\ 0.56 & -0.83 & 0.76 & 0.34 \\ -0.10 & 1.09 & 0.34 & 1.18 \end{pmatrix},$$

together with and estimate of the condition number of B, and approximate error bounds for the computed eigenvalues and eigenvectors.

The example program for nag_dsygvd (f08scc) illustrates solving a generalized symmetric eigenproblem of the form $ABz = \lambda z$.

10.1 Program Text

```
/* nag_dsygv (f08sac) Example Program.
 *
 * Copyright 2011 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagf08.h>
#include <nagf16.h>
#include <nagx04.h>
#include <nagx02.h>

int main(void)
{

  /* Scalars */
  double        anorm, bnorm, eps, rcond, rcondb, t1, t2, t3;
  Integer       exit_status = 0, i, j, n, pda, pdb;

  /* Arrays */
  double        *a = 0, *b = 0, *eerbnd = 0, *rcondz = 0, *w = 0, *zerbnd = 0;
  char          nag_enum_arg[40];

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

#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_dsygv (f08sac) Example Program Results\n\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
 */
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

pda = n;
pdb = n;
/* Allocate memory */
if (!(a      = NAG_ALLOC(n * n, double)) ||
    !(b      = NAG_ALLOC(n * n, double)) ||
    !(eerbnd = NAG_ALLOC(n, double)) ||
    !(rcondz = NAG_ALLOC(n, double)) ||
    !(w      = NAG_ALLOC(n, double)) ||
    !(zerbnd = NAG_ALLOC(n, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

/* Read the triangular parts of the matrices A and B */
if (uplo == Nag_Upper)
  {
    for (i = 1; i <= n; ++i)
      for (j = i; j <= n; ++j) scanf("%lf", &A(i, j));
    scanf("%*[^\n]");
    for (i = 1; i <= n; ++i)
      for (j = i; j <= n; ++j) scanf("%lf", &B(i, j));
  }
else
  {
    for (i = 1; i <= n; ++i)
      for (j = 1; j <= i; ++j) scanf("%lf", &A(i, j));
    scanf("%*[^\n] ");
    for (i = 1; i <= n; ++i)
      for (j = 1; j <= i; ++j) scanf("%lf", &B(i, j));
  }
scanf("%*[^\n] ");

/* Compute the one-norms of the symmetric matrices A and B */
nag_dsy_norm(order, Nag_OneNorm, uplo, n, a, pda, &anorm, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dsy_norm (f16rcc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

nag_dsy_norm(order, Nag_OneNorm, uplo, n, b, pdb, &bnorm, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dsy_norm (f16rcc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Solve the generalized symmetric eigenvalue problem  A*x = lambda*B*x
 * using nag_dsygv (f08sac).
 */
nag_dsygv(order, 1, Nag_DoBoth, uplo, n, a, pda, b, pdb, w, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dsygv (f08sac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
```

```
/* Normalize the eigenvectors */
for(j=1; j<=n; j++)
  for(i=n; i>=1; i--) A(i, j) = A(i, j) / A(1,j);

/* Print eigensolution */
printf(" Eigenvalues\n   ");
for (j = 0; j < n; ++j) printf(" %10.4f%s", w[j], j%6 == 5?"\n":"");
printf("\n\n");

/* Print the normalized eigenvectors using nag_gen_real_mat_print (x04cac). */
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n, a,
                       pda, "Eigenvectors", 0, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Estimate the reciprocal condition number of the Cholesky factor of B.
 * nag_dtrcon (f07tgf)
 * Note that: cond(B) = 1.0/(rcond*rcond)
 */
nag_dtrcon(order, Nag_OneNorm, uplo, Nag_NonUnitDiag, n, b, pdb, &rcond,
           &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_dtrcon (f07tgc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
rcondb = rcond * rcond;
printf("\nEstimate of reciprocal condition number for B\n%15.1e\n", rcondb);

/* Get the machine precision, using nag_machine_precision (x02ajc) */
eps = nag_machine_precision;
if (rcond < eps)
  {
    printf("\nB is very ill-conditioned, error estimates have not been "
           "computed\n");
    goto END;
  }

/* Estimate reciprocal condition numbers for the eigenvectors of A-lambda*B
 * nag_ddisna (f08flc).
 */
nag_ddisna(Nag_EigVecs, n, n, w, rcondz, &fail);
if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_ddisna (f08flc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Compute the error estimates for the eigenvalues and eigenvectors */
t1 = eps / rcondb;
t2 = anorm / bnorm;
t3 = t2 / rcond;
for (i = 0; i < n; ++i)
  {
    eerbnd[i] = t1 * (t2 + abs(w[i]));
    zerbnd[i] = t1 * (t3 + abs(w[i]))/rcondz[i];
  }

/* Print the approximate error bounds for the eigenvalues and vectors */
printf("\nError estimates for the eigenvalues\n    ");
for (i = 0; i < n; ++i) printf(" %10.1e%s", eerbnd[i], i%6 == 5?"\n":"");

printf("\n\nError estimates for the eigenvectors\n    ");
```

```
  for (i = 0; i < n; ++i) printf(" %10.1e%s", zerbnd[i], i%6 == 5?"\n":"");
  printf("\n");

 END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(eerbnd);
  NAG_FREE(rcondz);
  NAG_FREE(w);
  NAG_FREE(zerbnd);

  return exit_status;
}
```

10.2 Program Data

```
nag_dsygv (f08sac) Example Program Data

  4                          : n
  Nag_Upper                  : uplo

  0.24   0.39   0.42  -0.16
        -0.11   0.79   0.63
               -0.25   0.48
                      -0.03 : matrix A

  4.16  -3.12   0.56  -0.10
         5.03  -0.83   1.09
                0.76   0.34
                       1.18 : matrix B
```

10.3 Program Results

```
nag_dsygv (f08sac) Example Program Results

 Eigenvalues
       -2.2254    -0.4548     0.1001     1.1270

 Eigenvectors
            1          2          3          4
 1      1.0000     1.0000     1.0000     1.0000
 2      8.3184     1.7303     0.0830     1.2240
 3     22.3569    -1.1354    -0.1129     1.6780
 4    -20.2941    -2.0169    -1.0611    -0.4540

Estimate of reciprocal condition number for B
        5.8e-03

Error estimates for the eigenvalues
        4.2e-14    3.7e-15    3.7e-15    2.3e-14

Error estimates for the eigenvectors
        4.9e-14    8.8e-14    8.8e-14    6.6e-14
```