NAG Library Function Document

nag_dsygvd (f08scc)

1 Purpose

nag_dsygvd (f08scc) computes all the eigenvalues and, optionally, the eigenvectors of a real generalized symmetric-definite eigenproblem, of the form

\[ \begin{align*}
    A z &= \lambda B z, \\
    A B z &= \lambda z \\text{or} \quad B A z &= \lambda z,
\end{align*} \]

where \( A \) and \( B \) are symmetric and \( B \) is also positive definite. If eigenvectors are desired, it uses a divide-and-conquer algorithm.

2 Specification

```c
#include <nag.h>
#include <nagf08.h>
void nag_dsygvd (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_dsygvd (f08scc) first performs a Cholesky factorization of the matrix \( B \) as \( B = U^T U \), when \( \text{uplo} = \text{Nag_Upper} \) or \( B = L L^T \), when \( \text{uplo} = \text{Nag_Lower} \). The generalized problem is then reduced to a standard symmetric eigenvalue problem

\[ C x = \lambda x, \]

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 \( A z = \lambda B z \), the eigenvectors are normalized so that the matrix of eigenvectors, \( z \), satisfies

\[ Z^T A Z = A \quad \text{and} \quad Z^T B Z = I, \]

where \( A \) is the diagonal matrix whose diagonal elements are the eigenvalues. For the problem \( A B z = \lambda z \) we correspondingly have

\[ Z^{-1} A Z^{-T} = A \quad \text{and} \quad Z^T B Z = I, \]

and for \( B A z = \lambda z \) we have

\[ Z^T A Z = A \quad \text{and} \quad Z^T B^{-1} Z = I. \]

4 References


5 Arguments

1: order – Nag_OrderType  
   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  
   On entry: specifies the problem type to be solved.
   itype = 1
   \[ A_z = \lambda B_z. \]
   itype = 2
   \[ AB_z = \lambda z. \]
   itype = 3
   \[ BA_z = \lambda z. \]
   Constraint: itype = 1, 2 or 3.

3: job – Nag_JobType  
   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  
   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  
   On entry: n, the order of the matrices A and B.
   Constraint: n \geq 0.

6: a[dim] – double  
   Note: the dimension, dim, of the array a must be at least max(1, pda \times n).
   On entry: the n by n symmetric matrix A.
   If order = Nag_ColMajor, \( A_{ij} \) is stored in \( a[(j - 1) \times pda + i - 1] \).
   If order = Nag_RowMajor, \( A_{ij} \) is stored in \( a[(i - 1) \times pda + j - 1] \).
   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 $\text{itype} = 1$ or $2$, $Z^T B Z = I$;
if $\text{itype} = 3$, $Z^T B^{-1} Z = I$.

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: $\text{pda}$ – Integer

*Input*

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

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

8: $\text{b}[\text{dim}]$ – double

*Input/Output*

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

On entry: the $n$ by $n$ symmetric matrix $B$.

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

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: the triangular factor $U$ or $L$ from the Cholesky factorization $B = U^T U$ or $B = LL^T$.

9: $\text{pdb}$ – Integer

*Input*

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

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

10: $\text{w}[n]$ – double

*Output*

On exit: the eigenvalues in ascending order.

11: $\text{fail}$ – 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 $\langle \text{value} \rangle$ had an illegal value.

**NE_CONVERGENCE**

The algorithm failed to converge; $\langle \text{value} \rangle$ off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

**NE_INT**

On entry, $\text{itype} = \langle \text{value} \rangle$.

*Constraint:* $\text{itype} = 1, 2$ or $3$. 

---

Mark 25
On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \geq 0 \).

On entry, \( pda = \langle \text{value} \rangle \).
Constraint: \( pda > 0 \).

On entry, \( pdb = \langle \text{value} \rangle \).
Constraint: \( pdb > 0 \).

**NE_INT_2**

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

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

**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_MAT_NOT_POS_DEF**

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

**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**

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**

\( \text{nag_dsygvd (f08scc)} \) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\( \text{nag_dsygvd (f08scc)} \) 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 \( \text{nag_zhegvd (f08sqc)} \).
10 Example

This example finds all the eigenvalues and eigenvectors of the generalized symmetric eigenproblem
\[ ABz = \lambda Cz, \]
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 an estimate of the condition number of \( B \), and approximate error bounds for the computed eigenvalues and eigenvectors.

The example program for nag_dsygv (f08sac) illustrates solving a generalized symmetric eigenproblem of the form \( Az = \lambda Bz \).

10.1 Program Text

/* nag_dsygv (f08scc) Example Program.

* Copyright 2014 Numerical Algorithms Group.
* Mark 23, 2011.
*/

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

int main(void)
{
    /* Scalars */
    double anorm, bnorm, eps, rcond, rcondb, t1, t2, t3;
    Integer i, j, n, pda, pdb;
    Integer exit_status = 0;
    /* 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 (f08scc) Example Program Results\n\n");

    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\n]");
    #else
    scanf("%*[\n]");
    */
#ifndef __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
 */
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);
pda = n;
pdb = n;
#endif __WIN32
if (a = NAG_ALLOC(n * n, double)) ||
(b = NAG_ALLOC(n * n, double)) ||
(eerbd = NAG_ALLOC(n, double)) ||
(rcondz = NAG_ALLOC(n, double)) ||
(w = NAG_ALLOC(n, double)) ||
(zerbd = 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)
#ifdef __WIN32
        for (j = i; j <= n; ++j) scanf_s("%lf", &A(i, j));
#else
        for (j = i; 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 <= i; ++j) scanf_s("%lf", &B(i, j));
#else
        for (j = 1; j <= i; ++j) scanf("%lf", &B(i, j));
#endif
} else 
{
    for (i = 1; i <= n; ++i)
#ifdef __WIN32
        for (j = 1; j <= i; ++j) scanf_s("%lf", &A(i, j));
#else
        for (j = 1; j <= i; ++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 <= i; ++j) scanf_s("%lf", &B(i, j));
#else
    for (j = 1; j <= i; ++j) scanf("%lf", &B(i, j));
#endif
}
#ifdef _WIN32
    scanf_s("%*[\n"]);
#else
    scanf("%*[\n"]);
#endif

/* Compute the one-norms of the symmetric matrices A and B using
   nag_dsy_norm (f16rcc).
*/
    nag_dsy_norm(order, Nag_OneNorm, uplo, n, a, pda, &anorm, &fail);
    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*B*x = lambda*x
   using nag_dsygvd (f08scc).
*/
    nag_dsygvd(order, 2, Nag_DoBoth, uplo, n, a, pda, b, pdb, w, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_dsygvd (f08scc).\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");
    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 (f07tgc)
   * 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;
    }

/* Print the reciprocal condition number of B */
    rcondb = rcond * rcond;
    printf("\nEstimate of reciprocal condition number for B\n %11.1e\n", rcondb,
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");
go_to 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;
go_to END;
}

/* Compute the error estimates for the eigenvalues and eigenvectors. */
t1 = 1.0 / rcond;
t2 = eps * t1;
t3 = anorm * bnorm;
for (i = 0; i < n; ++i)
{
    eerbnd[i] = eps * (t3 + abs(w[i])/rcondb);
    zerbnd[i] = t2 * (t3/rcondz[i] + t1);
}

/* 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
Error 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_dsygvd (f08scc) Example Program Data

<table>
<thead>
<tr>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uplo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nag_Upper</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.24 0.39 0.42 -0.16</td>
</tr>
<tr>
<td>-0.11 0.79 0.63</td>
</tr>
<tr>
<td>-0.25 0.48</td>
</tr>
<tr>
<td>-0.03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.16 -3.12 0.56 -0.10</td>
</tr>
<tr>
<td>5.03 -0.83 1.09</td>
</tr>
<tr>
<td>0.76 0.34</td>
</tr>
<tr>
<td>1.18</td>
</tr>
</tbody>
</table>
10.3 Program Results

nag_dsygvd (f08scc) Example Program Results

Eigenvalues

-3.5411  -0.3347  0.2983  2.2544

Eigenvectors

1 2 3 4
1 1.0000 1.0000 1.0000 1.0000
2 -10.6846 -4.1593 1.0517 1.8547
3 8.2568 -15.0567 0.9577 2.9680
4 8.9384 10.2472 -1.4993 2.0218

Estimate of reciprocal condition number for B

5.8e-03

Error estimates for the eigenvalues

6.0e-14  2.2e-15  2.2e-15  4.1e-14

Error estimates for the eigenvectors

2.8e-14  6.4e-14  6.4e-14  3.4e-14