NAG Library Function Document

nag_dspgv (f08tac)

1 Purpose

nag_dspgv (f08tac) computes all the eigenvalues and, optionally, all the eigenvectors of a real generalised 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, stored in packed format, and \( B \) is also positive definite.

2 Specification

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

void nag_dspgv (Nag_OrderType order, Integer itype, Nag_JobType job,
                  Nag_UploType uplo, Integer n, double ap[], double bp[], double w[],
                  double z[], Integer pdz, NagError *fail)
```

3 Description

nag_dspgv (f08tac) 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 generalised 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

\[ Z^T AZ = \Lambda \quad \text{and} \quad Z^T BZ = I, \]

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

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

and for \( BAz = \lambda z \) we have

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

4 References


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`:  
     \[ A_z = \lambda B_z. \]

   - `itype = 2`:  
     \[ AB_z = \lambda z. \]

   - `itype = 3`:  
     \[ BA_z = \lambda z. \]

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

3: job – Nag_JobType  
   **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:** \( n \geq 0 \).

6: `ap[dim]` – double  
   **Input/Output**

   **Note:** the dimension, \( dim \), of the array `ap` must be at least \( \max(1, n \times (n + 1)/2) \).

   *On entry:* the upper or lower triangle of the \( n \) by \( n \) symmetric matrix \( A \), packed by rows or columns.

   The storage of elements \( A_{ij} \) depends on the `order` and `uplo` arguments as follows:

   - if `order = Nag_ColMajor` and `uplo = Nag_Upper`, \( A_{ij} \) is stored in \( ap[(j - 1) \times j/2 + i - 1] \), for \( i \leq j \);
   - if `order = Nag_ColMajor` and `uplo = Nag_Lower`, \( A_{ij} \) is stored in \( ap[(2n - j) \times (j - 1)/2 + i - 1] \), for \( i \geq j \);
   - if `order = Nag_RowMajor` and `uplo = Nag_Upper`, \( A_{ij} \) is stored in \( ap[(2n - i) \times (i - 1)/2 + j - 1] \), for \( i \leq j \);
   - if `order = Nag_RowMajor` and `uplo = Nag_Lower`, \( A_{ij} \) is stored in \( ap[(2n - j) \times (j - 1)/2 + i - 1] \), for \( i \geq j \).
if order = Nag_RowMajor and uplo = Nag_Lower, 
A_{ij} is stored in \( a_p[(i-1) \times i/2 + j - 1] \), for \( i \geq j \).

On exit: the contents of \( a_p \) are destroyed.

7: \( b_p[dim] \) – double
     Input/Output

Note: the dimension, \( dim \), of the array \( b_p \) must be at least \( \max(1, n \times (n+1)/2) \).

On entry: the upper or lower triangle of the \( n \) by \( n \) symmetric matrix \( B \), packed by rows or columns.

The storage of elements \( B_{ij} \) depends on the \( order \) and \( uplo \) arguments as follows:

- If \( order = Nag_ColMajor \) and \( uplo = Nag_Upper \), \( B_{ij} \) is stored in \( b_p[(j-1) \times j/2 + i - 1] \), for \( i \leq j \);
- If \( order = Nag_ColMajor \) and \( uplo = Nag_Lower \), \( B_{ij} \) is stored in \( b_p[(2n-j) \times (j-1)/2 + i - 1] \), for \( i \geq j \);
- If \( order = Nag_RowMajor \) and \( uplo = Nag_Upper \), \( B_{ij} \) is stored in \( b_p[(2n-i) \times (i-1)/2 + j - 1] \), for \( i \leq j \);
- If \( order = Nag_RowMajor \) and \( uplo = Nag_Lower \), \( B_{ij} \) is stored in \( b_p[(i-1) \times i/2 + j - 1] \), for \( i \geq j \).

On exit: the triangular factor \( U \) or \( L \) from the Cholesky factorization \( B = U^T U \) or \( B = LL^T \), in the same storage format as \( B \).

8: \( w[n] \) – double
     Output

On exit: the eigenvalues in ascending order.

9: \( z[dim] \) – double
     Output

Note: the dimension, \( dim \), of the array \( z \) must be at least \( \max(1, pdz \times n) \) when \( job = Nag_DoBoth \); 1 otherwise.

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

\( z[(j-1) \times pdz + i - 1] \) when \( order = Nag_ColMajor \);
\( z[(i-1) \times pdz + j - 1] \) when \( order = Nag_RowMajor \).

On exit: if \( job = Nag_DoBoth \), \( z \) contains the matrix \( Z \) of eigenvectors. The eigenvectors are normalized as follows:

- If \( itype = 1 \) or \( 2 \), \( Z^T B Z = I \);
- If \( itype = 3 \), \( Z^T B^{-1} Z = I \).

If \( job = Nag_EigVals \), \( z \) is not referenced.

10: \( pdz \) – Integer
     Input

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

Constraints:

- If \( job = Nag_DoBoth \), \( pdz \geq \max(1, n) \);
- Otherwise \( pdz \geq 1 \).

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.
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 algorithm failed to converge; \(<value>\) off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

NE_ENUM_INT_2
On entry, \(job = \langle value\rangle\), \(pdz = \langle value\rangle\) and \(n = \langle value\rangle\).
Constraint: if \(job = \text{Nag\_Do\_Both}\), \(pdz \geq \max(1, n)\);
otherwise \(pdz \geq 1\).

NE_INT
On entry, \(itype = \langle value\rangle\).
Constraint: \(itype = 1, 2 \text{ or } 3\).
On entry, \(n = \langle value\rangle\).
Constraint: \(n \geq 0\).
On entry, \(pdz = \langle value\rangle\).
Constraint: \(pdz > 0\).

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 \(\text{fail.errnum} = n + \langle value\rangle\), for \(1 \leq \langle value\rangle \leq n\), 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.

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.
parallelism and Performance

f08tac is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

f08tac 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 f08tnc.

10 Example

This example finds all the eigenvalues and eigenvectors of the generalized symmetric eigenproblem $A z = \lambda B z$, 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 f08tcc illustrates solving a generalized symmetric eigenproblem of the form $AB z = \lambda z$.

10.1 Program Text

```c
#include <stdio.h>
#include <nag.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;
    Integer i, j, n;
    Integer exit_status = 0;

    /* Arrays */
    double *ap = 0, *bp = 0, *dummy = 0, *eerbnd = 0, *w = 0;
    char nag_enum_arg[40];

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

Mark 25
Nag_UploType uplo;

#ifdef NAG_COLUMN_MAJOR
#define A_UPPER(I, J) ap[J*(J-1)/2 + I - 1]
#define A_LOWER(I, J) ap[(2*n-J)*(J-1)/2 + I - 1]
#define B_UPPER(I, J) bp[J*(J-1)/2 + I - 1]
#define B_LOWER(I, J) bp[(2*n-J)*(J-1)/2 + I - 1]
#else
#define A_UPPER(I, J) ap[(2*n-I)*(I-1)/2 + J - 1]
#define A_LOWER(I, J) ap[I*(I-1)/2 + J - 1]
#define B_UPPER(I, J) bp[(2*n-I)*(I-1)/2 + J - 1]
#define B_LOWER(I, J) bp[I*(I-1)/2 + J - 1]
#endif

order = Nag_ColMajor;
#else
#define A_UPPER(I, J) ap[(2*n-I)*(I-1)/2 + J - 1]
#define A_LOWER(I, J) ap[I*(I-1)/2 + J - 1]
#define B_UPPER(I, J) bp[(2*n-I)*(I-1)/2 + J - 1]
#define B_LOWER(I, J) bp[I*(I-1)/2 + J - 1]
#endif

INIT_FAIL(fail);

printf("nag_dspgv (f08tac) Example Program Results\n\n");

/* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif

/* nag_enum_name_to_value (x04nac).
This converts NAG enum member name to value */
uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

/* Allocate memory */
if (!(ap = NAG_ALLOC(n*(n+1)/2, double)) ||
    !(bp = NAG_ALLOC(n*(n+1)/2, double)) ||
    !(dummy = NAG_ALLOC(1 * 1, double)) ||
    !(eerbnd = NAG_ALLOC(n, double)) ||
    !(w = NAG_ALLOC(n, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read the triangular parts of the matrices A and B from data file. */
if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
        for (j = i; j <= n; ++j) scanf("%lf", &A_UPPER(i, j));
}
```c
scanf("%*[\n\"]");
#endif
for (i = 1; i <= n; ++i)
#endif
for (j = i; j <= n; ++j) scanf("%lf", &B_UPPER(i, j));
#else
for (j = i; j <= n; ++j) scanf("%lf", &B_UPPER(i, j));
#endif
}
else if (uplo == Nag_Lower)
{
#endif
scanf_s("%*[\n\"]");
#endif
for (i = 1; i <= n; ++i)
#endif
for (j = 1; j <= i; ++j) scanf_s("%lf", &A_LOWER(i, j));
#else
for (j = 1; j <= i; ++j) scanf("%lf", &A_LOWER(i, j));
#endif
}
#else
scanf("%*[\n\"]");
#endif
for (i = 1; i <= n; ++i)
#endif
for (j = 1; j <= i; ++j) scanf_s("%lf", &B_LOWER(i, j));
#else
for (j = 1; j <= i; ++j) scanf("%lf", &B_LOWER(i, j));
#endif
}
#ifdef _WIN32
scanf_s("%*[\n\"]");
#else
scanf("%*[\n\"]");
#endif
for (i = 1; i <= n; ++i)
#endif
for (j = 1; j <= i; ++j) scanf_s("%lf", &B_LOWER(i, j));
#else
for (j = 1; j <= i; ++j) scanf("%lf", &B_LOWER(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_dsp_norm (f16rdc).
 */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dsp_norm (f16rdc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Solve the generalized symmetric eigenvalue problem A*x = lambda*B*x
 * using nag_dspgv (f08tac).
 */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dspgv (f08tac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* Print solution */
printf("Eigenvalues\n");
for (j = 0; j < n; ++j) printf(" %11.4f\n", w[j], j%6 == 5?"\n":"");
printf("\n");
/* Estimate the reciprocal condition number of the Cholesky factor of B.
 * nag_dtpcon (f07ugc)
 * Note that: cond(B) = 1/(rcond*rcond).
 */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dtpcon (f07ugc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
```

`f08 – Least-squares and Eigenvalue Problems (LAPACK)`

Mark 25
/* Print the reciprocal condition number of B */
  rcondb = rcond * rcond;
  printf("\nEstimate of reciprocal condition number for B
  \%11.1e\n", rcondb);

/* Get the machine precision, using nag_machine_precision (x02ajc) */
  eps = nag_machine_precision;
  if (rcond < eps)
  {
    printf("B is very ill-conditioned, error estimates have not been "
           "computed\n");
  }
  t1 = eps / rcondb;
  t2 = anorm / bnorm;
  for (i = 0; i < n; ++i) eerbnd[i] = t1 * (t2 + abs(w[i]));

/* Print the approximate error bounds for the eigenvalues */
  printf("\nError estimates for the eigenvalues
  \%11.1e\n", eerbnd[i], i%6 == 5?'\n':"");
  printf("\n");

END:
  NAG_FREE(ap);
  NAG_FREE(bp);
  NAG_FREE(dummy);
  NAG_FREE(eerbnd);
  NAG_FREE(w);
  return exit_status;
}