NAG Library Function Document

nag_dspgvx (f08tbc)

1 Purpose

nag_dspgvx (f08tbc) computes selected eigenvalues and, optionally, eigenvectors of a real generalized symmetric-definite eigenproblem, of the form

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

where \( A \) and \( B \) are symmetric, stored in packed storage, and \( B \) is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

2 Specification

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

void nag_dspgvx (Nag_OrderType order, Integer itype, Nag_JobType job,
                 Nag_RangeType range, Nag_UploType uplo, Integer n, double ap[],
                 double bp[], double vl, double vu, Integer il, Integer iu,
                 double abstol, Integer *m, double w[], double z[], Integer pdz,
                 Integer jfail[], NagError *fail)
```

3 Description

nag_dspgvx (f08tbc) first performs a Cholesky factorization of the matrix \( B \) as \( B = U^T U \), when \( \text{uplo} = \text{Nag}_\text{Upper} \) or \( B = LL^T \), when \( \text{uplo} = \text{Nag}_\text{Lower} \). The generalized problem is then reduced to a standard symmetric eigenvalue problem

\[ C x = \lambda x, \]

which is solved for the desired eigenvalues and 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

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

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

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

and for \( BA z = \lambda z \) 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
   \[ A\mathbf{z} = \lambda \mathbf{Bz} \].

   itype = 2
   \[ A\mathbf{Bz} = \lambda \mathbf{z} \].

   itype = 3
   \[ \mathbf{BAz} = \lambda \mathbf{z} \].
   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: range – Nag_RangeType
   Input
   On entry: if range = Nag_AllValues, all eigenvalues will be found.
   If range = Nag_Interval, all eigenvalues in the half-open interval \([v_1, v_2)\] will be found.
   If range = Nag_Indices, the \(i_{th}\) to \(i_{th}\) eigenvalues will be found.
   Constraint: range = Nag_AllValues, Nag_Interval or Nag_Indices.

5: 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.
6: \( n \) – Integer

*On entry:* \( n \), the order of the matrices \( A \) and \( B \).

*Constraint:* \( n \geq 0 \).

7: \( \text{ap}[\text{dim}] \) – double

*Input/Output*

*Note:* the dimension, \( \text{dim} \), of the array \( \text{ap} \) 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 \( \text{order} \) and \( \text{uplo} \) arguments as follows:

- If \( \text{order} = \text{Nag}_\text{ColMajor} \) and \( \text{uplo} = \text{Nag}_\text{Upper} \),
  \( A_{ij} \) is stored in \( \text{ap}[(j - 1) \times j/2 + i - 1] \), for \( i \leq j \);
- If \( \text{order} = \text{Nag}_\text{ColMajor} \) and \( \text{uplo} = \text{Nag}_\text{Lower} \),
  \( A_{ij} \) is stored in \( \text{ap}[(2n - j) \times (j - 1)/2 + i - 1] \), for \( i \geq j \);
- If \( \text{order} = \text{Nag}_\text{RowMajor} \) and \( \text{uplo} = \text{Nag}_\text{Upper} \),
  \( A_{ij} \) is stored in \( \text{ap}[(2n - i) \times (i - 1)/2 + j - 1] \), for \( i \leq j \);
- If \( \text{order} = \text{Nag}_\text{RowMajor} \) and \( \text{uplo} = \text{Nag}_\text{Lower} \),
  \( A_{ij} \) is stored in \( \text{ap}[(i - 1) \times i/2 + j - 1] \), for \( i \geq j \).

*On exit:* the contents of \( \text{ap} \) are destroyed.

8: \( \text{bp}[\text{dim}] \) – double

*Input/Output*

*Note:* the dimension, \( \text{dim} \), of the array \( \text{bp} \) 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 \( \text{order} \) and \( \text{uplo} \) arguments as follows:

- If \( \text{order} = \text{Nag}_\text{ColMajor} \) and \( \text{uplo} = \text{Nag}_\text{Upper} \),
  \( B_{ij} \) is stored in \( \text{bp}[(j - 1) \times j/2 + i - 1] \), for \( i \leq j \);
- If \( \text{order} = \text{Nag}_\text{ColMajor} \) and \( \text{uplo} = \text{Nag}_\text{Lower} \),
  \( B_{ij} \) is stored in \( \text{bp}[(2n - j) \times (j - 1)/2 + i - 1] \), for \( i \geq j \);
- If \( \text{order} = \text{Nag}_\text{RowMajor} \) and \( \text{uplo} = \text{Nag}_\text{Upper} \),
  \( B_{ij} \) is stored in \( \text{bp}[(2n - i) \times (i - 1)/2 + j - 1] \), for \( i \leq j \);
- If \( \text{order} = \text{Nag}_\text{RowMajor} \) and \( \text{uplo} = \text{Nag}_\text{Lower} \),
  \( B_{ij} \) is stored in \( \text{bp}[(i - 1) \times i/2 + j - 1] \), 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 \( B \).

9: \( \text{vl} \) – double

*Input*

10: \( \text{vu} \) – double

*Input*

*On entry:* if \( \text{range} = \text{Nag}_\text{Interval} \), the lower and upper bounds of the interval to be searched for eigenvalues.

If \( \text{range} = \text{Nag}_\text{AllValues} \) or \( \text{Nag}_\text{Indices} \), \( \text{vl} \) and \( \text{vu} \) are not referenced.

*Constraint:* if \( \text{range} = \text{Nag}_\text{Interval} \), \( \text{vl} < \text{vu} \).

11: \( \text{il} \) – Integer

*Input*

12: \( \text{iu} \) – Integer

*Input*

*On entry:* if \( \text{range} = \text{Nag}_\text{Indices} \), the indices (in ascending order) of the smallest and largest eigenvalues to be returned.

If \( \text{range} = \text{Nag}_\text{AllValues} \) or \( \text{Nag}_\text{Interval} \), \( \text{il} \) and \( \text{iu} \) are not referenced.
Constraints:

- if range = Nag_Indices and n = 0, il = 1 and iu = 0;
- if range = Nag_Indices and n > 0, 1 ≤ il ≤ iu ≤ n.

13: **abstol** – double  
*Input*

On entry: the absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval \([a, b]\) of width less than or equal to

\[
abstol + \epsilon \max(|a|, |b|),
\]

where \(\epsilon\) is the *machine precision*. If abstol is less than or equal to zero, then \(\epsilon \|T\|_1\) will be used in its place, where \(T\) is the tridiagonal matrix obtained by reducing \(C\) to tridiagonal form. Eigenvalues will be computed most accurately when abstol is set to twice the underflow threshold \(2 \times \text{nag\_real\_safe\_small\_number}\), not zero. If this function returns with fail code = NE_CONVERGENCE, indicating that some eigenvectors did not converge, try setting abstol to \(2 \times \text{nag\_real\_safe\_small\_number}\). See Demmel and Kahan (1990).

14: **m** – Integer *  
*Output*

On exit: the total number of eigenvalues found. 0 ≤ m ≤ n.

If range = Nag_AllValues, m = n.
If range = Nag_Indices, m = iu - il + 1.

15: **w[n]** – double  
*Output*

On exit: the first m elements contain the selected eigenvalues in ascending order.

16: **z[dim]** – double  
*Output*

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

\[
\max(1, pdz \times n) \text{ when job = Nag_DoBoth};
1 \text{ otherwise.}
\]

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, then

- if fail code = NE_NOERROR, the first m columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the ith column of Z holding the eigenvector associated with \(w[i - 1]\). 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 an eigenvector fails to converge (fail code = NE_CONVERGENCE), then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in jfail.

If job = Nag_EigVals, z is not referenced.

17: **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 \(pdz \geq 1\).

18: \(jfail[dim] – Integer\)

Note: the dimension, \(dim\), of the array \(jfail\) must be at least \(\max(1, n)\).

On exit: if \(job = Nag_DoBoth\), then

if \(fail\.code = NE\_NOERROR\), the first \(m\) elements of \(jfail\) are zero;

if \(fail\.code = NE\_CONVERGENCE\), \(jfail\) contains the indices of the eigenvectors that
failed to converge.

If \(job = Nag\.EigVals\), \(jfail\) is not referenced.

19: \(fail – Nag\_Error\) *

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 \(\langle value\rangle\) had an illegal value.

NE_CONVERGENCE

The algorithm failed to converge; \(\langle value\rangle\) eigenvectors failed to converge.

NE_ENUM_INT_2

On entry, \(\langle value\rangle\), \(pdz = \langle value\rangle\) and \(n = \langle value\rangle\).
Constraint: if \(job = Nag\_DoBoth\), \(pdz \geq \max(1, n)\); otherwise \(pdz \geq 1\).

NE_ENUM_INT_3

On entry, \(\langle value\rangle\), \(il = \langle value\rangle\), \(iu = \langle value\rangle\) and \(n = \langle value\rangle\).
Constraint: if \(range = Nag\_Indices\) and \(n = 0\), \(il = 1\) and \(iu = 0\);
if \(range = Nag\_Indices\) and \(n > 0\), \(1 \leq il \leq iu \leq n\).

NE_ENUM_REAL_2

On entry, \(\langle value\rangle\), \(vl = \langle value\rangle\) and \(vu = \langle value\rangle\).
Constraint: if \(range = Nag\_Interval\), \(vl < vu\).

NE_INT

On entry, \(itype = \langle value\rangle\).
Constraint: \(itype = 1, 2\) 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 fail.errnum = n + (value), for 1 ≤ (value) ≤ n, then the leading minor of order (value) 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.

8 Parallelism and Performance

nag_dspgvx (f08tbc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dspgvx (f08tbc) 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_zhpgvx (f08tpc).

10 Example

This example finds the eigenvalues in the half-open interval $(-1.0, 1.0]$, and corresponding eigenvectors, of the generalized symmetric eigenproblem $Az = \lambda Bz$, where

$$
A = \begin{pmatrix}
0.24 & 0.39 & 0.42 & -0.16 \\
0.39 & -0.11 & 0.79 & 0.63 \\
0.42 & 0.79 & -0.25 & 0.48 \\
-0.16 & 0.63 & 0.48 & -0.03 \\
\end{pmatrix}
$$

and

$$
B = \begin{pmatrix}
4.16 & -3.12 & 0.56 & -0.10 \\
-3.12 & 5.03 & -0.83 & 1.09 \\
0.56 & -0.83 & 0.76 & 0.34 \\
-0.10 & 1.09 & 0.34 & 1.18 \\
\end{pmatrix}.
$$

The example program for nag_dspgvd (f08tcc) illustrates solving a generalized symmetric eigenproblem of the form $ABz = \lambda z$. 
10.1 Program Text

/* nag_dspgvx (f08tbc) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
* Mark 23, 2011. */

#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf08.h>

typedef int int;

typedef double double;

typedef Integer Integer;

typedef NagError NagError;

typedef Nag_OrderType Nag_OrderType;

typedef Nag_UploType Nag_UploType;

typedef Nag.enum_arg Nag.enum_arg;

int main(void)
{
    /* Scalars */
    double abstol, vl, vu;
    Integer i, il = 0, iu = 0, j, m, n, pdz;
    Integer exit_status = 0;
    /* Arrays */
    double *ap = 0, *bp = 0, *w = 0, *z = 0;
    Integer *index = 0;
    char nag_enum_arg[40];
    /* Nag Types */
    NagError fail;
    Nag_OrderType order;
    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]
    define Z(I, J) z[(J-1)*pdz + I - 1]
    define Z(I, J) z[(J-1)*pdz + I - 1]
    define Z(I, J) z[(J-1)*pdz + I - 1]
    endif
    define Z(I, J) z[(J-1)*pdz + I - 1]
    endif

    INIT_FAIL(fail);

    printf("nag_dspgvx (f08tbc) Example Program Results\n\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(" %9s%*\[\n", nag_enum_arg, _countof(nag_enum_arg));
    else
    scanf("%9s%*\[\n", nag_enum_arg, _countof(nag_enum_arg));
    endif

    END:
    exit_status = 1;
    goto END; ;
    END:
    exit_status = 1;
    goto END; ;
#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);

m = n;
pdz = n;
/* Allocate memory */
if (! (ap = NAG_ALLOC(n*(n+1)/2, double)) ||
    ! (bp = NAG_ALLOC(n*(n+1)/2, double)) ||
    ! (w = NAG_ALLOC(n, double)) ||
    ! (z = NAG_ALLOC(n * m, double)) ||
    ! (index = NAG_ALLOC(n, Integer)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Read the lower and upper bounds of the interval to be searched. */
#ifdef _WIN32
    scanf_s("%lf%lf%*[\n]", &vl, &vu);
#else
    scanf("%lf%lf%*[\n]", &vl, &vu);
#endif
/* Read the triangular parts of the matrices A and B from data file. */
if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    #ifdef _WIN32
        for (j = i; j <= n; ++j) scanf_s("%lf", &A_UPPER(i, j));
    #else
        for (j = i; j <= n; ++j) scanf("%lf", &A_UPPER(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_UPPER(i, j));
    #else
        for (j = 1; j <= i; ++j) scanf("%lf", &B_UPPER(i, j));
    #endif
}
else if (uplo == Nag_Lower)
{
    for (i = 1; i <= n; ++i)
    #ifdef _WIN32
        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
    #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_LOWER(i, j));
    #else
        for (j = 1; j <= i; ++j) scanf("%lf", &B_LOWER(i, j));
    #endif
}
#else
    scanf_s("%*[\n]");
#endif
/* use the default absolute error tolerance for eigenvalues. */
abstol = 0.0;

/* Solve the generalized symmetric eigenvalue problem A*x = lambda*B*x */
/* using nag_dspgvx (f08tbc). */
nag_dspgvx(order, 1, Nag_DoBoth, Nag_Interval, uplo, n, ap, bp, vl, vu, il, 
iu, abstol, &m, w, z, pdz, index, &fail);

if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dspgvx (f08tbc).\n%s\n", fail.message);
    exit_status = 1;
go to END;
}

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

/* Print eigensolution */
printf("Number of eigenvalues found =%5"NAG_IFMT","m);
printf(" Eigenvalues\n ");
for (j = 0; j < m; ++j) printf(" %8.4f%s", w[j], j%8 == 7?"\n":"");
printf("\n");

/* Print normalized vectors using nag_gen_real_mat_print (x04cac). */
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, m, 
z, pdz, "Selected 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;
}

END:
NAG_FREE(ap);
NAG_FREE(bp);
NAG_FREE(w);
NAG_FREE(z);
NAG_FREE(index);

return exit_status;


10.2 Program Data

nag_dspgvx (f08tbc) Example Program Data

4 : n
Nag_Upper : uplo
-1.0 1.0 : VL and VU
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_dspgvx (f08tbc) Example Program Results

Number of eigenvalues found = 2

Eigenvalues
-0.4548  0.1001

Selected eigenvectors

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>2</td>
<td>1.7303</td>
<td>0.0830</td>
</tr>
<tr>
<td>3</td>
<td>-1.1354</td>
<td>-0.1129</td>
</tr>
<tr>
<td>4</td>
<td>-2.0169</td>
<td>-1.0611</td>
</tr>
</tbody>
</table>