NAG C Library Function Document

nag_dspgst (f08tec)

1 Purpose

nag_dspgst (f08tec) reduces a real symmetric-definite generalized eigenproblem $Az = \lambda Bz$, $ABz = \lambda z$ or $BAz = \lambda z$ to the standard form $Cy = \lambda y$, where A is a real symmetric matrix and B has been factorized by nag_dpptrf (f07gdc), using packed storage.

2 Specification

```
void nag_dspgst (Nag_OrderType order, Nag_ComputeType comp_type,
     Nag_UploType uplo, Integer n, double ap[], const double bp[], NagError *fail)
```

3 Description

To reduce the real symmetric-definite generalized eigenproblem $Az = \lambda Bz$, $ABz = \lambda z$ or $BAz = \lambda z$ to the standard form $Cy = \lambda y$ using packed storage, this function must be preceded by a call to nag_dpptrf (f07gdc) which computes the Cholesky factorization of B; B must be positive-definite.

The different problem types are specified by the parameter **comp_type**, as indicated in the table below. The table shows how C is computed by the function, and also how the eigenvectors z of the original problem can be recovered from the eigenvectors of the standard form.

comp_type	Problem	**uplo**	B	C	z
1	$Az = \lambda Bz$	**Nag_Upper** **Nag_Lower**	U^TU LL^T	$U^{-T}AU^{-1}$ $L^{-1}AL^{-T}$	$U^{-1}y$ $L^{-T}y$
2	$ABz = \lambda z$	**Nag_Upper** **Nag_Lower**	U^TU LL^T	UAU^T L^TAL	$U^{-1}yL^{-T}y$
3	$BAz = \lambda z$	**Nag_Upper** **Nag_Lower**	U^TU LL^T	UAU^T L^TAL	U^TyLy

4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

5 Parameters

1: **order** – Nag_OrderType *Input*

On entry: the **order** parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

Constraint: **order** = **Nag_RowMajor** or **Nag_ColMajor**.

2: **comp_type** – Nag_ComputeType *Input*

On entry: indicates how the standard form is computed as follows:

if **comp_type** = **Nag_Compute_1**,

if **uplo** = **Nag_Upper**, $C = U^{-T}AU^{-1}$;

if **uplo** = **Nag_Lower**, $C = L^{-1}AL^{-T}$;

if **comp_type** = **Nag_Compute_2** or **Nag_Compute_3**,

if **uplo** = **Nag_Upper**, $C = UAU^T$;

if **uplo** = **Nag_Lower**, $C = L^TAL$.

Constraint: **comp_type** = **Nag_Compute_1**, **Nag_Compute_2** or **Nag_Compute_3**.

3: **uplo** – Nag_UploType *Input*

On entry: indicates whether the upper or lower triangular part of A is stored and how B has been factorized, as follows:

if **uplo** = **Nag_Upper**, the upper triangular part of A is stored and $B = U^TU$;

if **uplo** = **Nag_Lower**, the lower triangular part of A is stored and $B = LL^T$.

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

4: **n** – Integer *Input*

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

Constraint: **n** ≥ 0.

5: **ap**[dim] – double *Input/Output*

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

On entry: the symmetric matrix A, packed by rows or columns. The storage of elements a_{ij} depends on the **order** and **uplo** parameters 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**$[(i-1) \times i/2 + j - 1]$, for $i \geq j$.

On exit: the upper or lower triangle of A is overwritten by the corresponding upper or lower triangle of C as specified by **comp_type** and **uplo**, using the same packed storage format as described above.

6: **bp**[dim] – const double *Input*

Note: the dimension, dim, of the array **bp** must be at least $\max(1, \mathbf{n} \times (\mathbf{n}+1)/2)$.

On entry: the Cholesky factor of B as specified by **uplo** and returned by nag_dpptrf (f07gdc).

7: **fail** – NagError * *Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 0.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter ⟨*value*⟩ had an illegal value.

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 consult NAG for assistance.

7 Accuracy

Forming the reduced matrix C is a stable procedure. However it involves implicit multiplication by B^{-1} if (**comp_type** = **Nag_Compute_1**) or B (if **comp_type** = **Nag_Compute_2** or **Nag_Compute_3**). When the function is used as a step in the computation of eigenvalues and eigenvectors of the original problem, there may be a significant loss of accuracy if B is ill-conditioned with respect to inversion.

8 Further Comments

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

The complex analogue of this function is nag_zhpgst (f08tsc).

9 Example

To compute all the eigenvalues of $Az = \lambda Bz$, 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

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

using packed storage. Here B is symmetric positive-definite and must first be factorized by nag_dpptrf (f07gdc). The program calls nag_dspgst (f08tec) to reduce the problem to the standard form $Cy = \lambda y$; then nag_dsptrd (f08gec) to reduce C to tridiagonal form, and nag_dsterf (f08jfc) to compute the eigenvalues.

9.1 Program Text

```
/* nag_dspgst (f08tec) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagf08.h>

int main(void)
{
```

```
  /* Scalars */
  Integer  i, j, n, ap_len, bp_len, d_len, e_len, tau_len;
  Integer  exit_status=0;
  NagError fail;
  Nag_UploType uplo;
  Nag_OrderType order;

  /* Arrays */
  char   uplo_char[2];
  double *ap=0, *bp=0, *d=0, *e=0, *tau=0;

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

  INIT_FAIL(fail);
  Vprintf("f08tec Example Program Results\n\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");
  Vscanf("%ld%*[^\n] ", &n);
  ap_len = n * (n +1 )/2;
  bp_len = n * (n +1 )/2;
  d_len = n;
  e_len = n-1;
  tau_len = n;

  /* Allocate memory */
  if ( !(ap = NAG_ALLOC(ap_len, double)) ||
       !(bp = NAG_ALLOC(bp_len, double)) ||
       !(d = NAG_ALLOC(d_len, double)) ||
       !(e = NAG_ALLOC(e_len, double)) ||
       !(tau = NAG_ALLOC(tau_len, double)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read A and B from data file */
  Vscanf(" ' %1s '%*[^\n] ", uplo_char);
  if (*(unsigned char *)uplo_char == 'L')
    uplo = Nag_Lower;
  else if (*(unsigned char *)uplo_char == 'U')
    uplo = Nag_Upper;
  else
    {
      Vprintf("Unrecognised character for Nag_UploType type\n");
      exit_status = -1;
      goto END;
    }
  if (uplo == Nag_Upper)
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = i; j <= n; ++j)
            Vscanf("%lf", &A_UPPER(i,j));
        }
      Vscanf("%*[^\n] ");
      for (i = 1; i <= n; ++i)
        {
```

```
          for (j = i; j <= n; ++j)
            Vscanf("%lf", &B_UPPER(i,j));
        }
      Vscanf("%*[^\n] ");
    }
  else
    {
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
            Vscanf("%lf", &A_LOWER(i,j));
        }
      Vscanf("%*[^\n] ");
      for (i = 1; i <= n; ++i)
        {
          for (j = 1; j <= i; ++j)
            Vscanf("%lf", &B_LOWER(i,j));
        }
      Vscanf("%*[^\n] ");
    }
  /* Compute the Cholesky factorization of B */
  f07gdc(order, uplo, n, bp, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f07gdc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Reduce the problem to standard form C*y = lambda*y, storing */
  /* the result in A */
  f08tec(order, Nag_Compute_1, uplo, n, ap, bp, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08tec.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Reduce C to tridiagonal form T = (Q**T)*C*Q */
  f08gec(order, uplo, n, ap, d, e, tau, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08gec.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Calculate the eigenvalues of T (same as C) */
  f08jfc(n, d, e, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from f08jfc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  /* Print eigenvalues */
  Vprintf("Eigenvalues\n");
  for (i = 1; i <= n; ++i)
    Vprintf("%8.4f%s", d[i-1], i%9==0 || i==n ?"\n":" ");
  Vprintf("\n");
 END:
  if (ap) NAG_FREE(ap);
  if (bp) NAG_FREE(bp);
  if (d) NAG_FREE(d);
  if (e) NAG_FREE(e);
  if (tau) NAG_FREE(tau);

  return exit_status;
}
```

9.2 Program Data

```
f08tec Example Program Data
  4                           :Value of N
  'L'                         :Value of UPLO
  0.24
  0.39  -0.11
  0.42   0.79  -0.25
 -0.16   0.63   0.48  -0.03   :End of matrix A
  4.16
 -3.12   5.03
  0.56  -0.83   0.76
 -0.10   1.09   0.34   1.18   :End of matrix B
```

9.3 Program Results

```
f08tec Example Program Results

Eigenvalues
 -2.2254  -0.4548   0.1001   1.1270
```