NAG 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

```c
#include <nag.h>
#include <nagf08.h>
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, nag_dspgst (f08tec) 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 argument `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.

<table>
<thead>
<tr>
<th><code>comp_type</code></th>
<th>Problem</th>
<th><code>uplo</code></th>
<th>$B$</th>
<th>$C$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nag_Compute_1</td>
<td>$Az = \lambda Bz$</td>
<td>Nag_Upper</td>
<td>$U^T U$</td>
<td>$U^{-T}A U^{-1}$</td>
<td>$U^{-1}_L y$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nag_Lower</td>
<td>$L L^T$</td>
<td>$L^{-1}_A L^{-T}$</td>
<td>$L^{-1}_L y$</td>
</tr>
<tr>
<td>Nag_Compute_2</td>
<td>$ABz = \lambda z$</td>
<td>Nag_Upper</td>
<td>$U^T U$</td>
<td>$U A U^T$</td>
<td>$U^{-1}_L y$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nag_Lower</td>
<td>$L L^T$</td>
<td>$L^T A L$</td>
<td>$L^{-1}_L y$</td>
</tr>
<tr>
<td>Nag_Compute_3</td>
<td>$BAz = \lambda z$</td>
<td>Nag_Upper</td>
<td>$U^T U$</td>
<td>$U A U^T$</td>
<td>$U^T_L y$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nag_Lower</td>
<td>$L L^T$</td>
<td>$L^T A L$</td>
<td>$L_L y$</td>
</tr>
</tbody>
</table>

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:  comp_type – Nag_ComputeType
    Input
    On entry: indicates how the standard form is computed.
    comp_type = Nag_Compute_1
        if uplo = Nag_Upper, C = U^{-T}AU^{-1};
        if uplo = Nag_Lower, C = L^{-1}AL^{-T}.
    comp_type = Nag_Compute_2 or Nag_Compute_3
        if uplo = Nag_Upper, C = UAU^{T};
        if uplo = Nag_Lower, C = L^{T}AL.
    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.
    uplo = Nag_Upper
        The upper triangular part of A is stored and B = U^{T}U.
    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 \geq 0.

5:  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[(i - 1) \times i/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 ap 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, n \times (n + 1)/2).
    On entry: the Cholesky factor of B as specified by uplo and returned by nag_dpptrf (f07gdc).

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

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_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
Forming the reduced matrix \( C \) is a stable procedure. However it involves implicit multiplication by \( B^{-1} \) if \( \langle \text{comp_type} \rangle = \text{Nag_Compute}_1 \) or \( B \) (if \( \langle \text{comp_type} \rangle = \text{Nag_Compute}_2 \) or \( \text{Nag_Compute}_3 \)). When nag_dspgst (f08tec) 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 Parallelism and Performance
nag_dspgst (f08tec) is not threaded by NAG in any implementation.
nag_dspgst (f08tec) 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 approximately \( n^3 \).
The complex analogue of this function is nag_zhpgst (f08tsc).
10 Example

This example computes all the eigenvalues of $Az = \lambda Bz$, where

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

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.

10.1 Program Text

/* nag_dspgst (f08tec) Example Program.
 * Copyright 2014 Numerical Algorithms Group.
 */
#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 nag_enum_arg[40];
    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]
    #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]
    #endif

    order = Nag_ColMajor;
    #else
    order = Nag_RowMajor;
    #endif

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

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

    return exit_status;
}

f08tec

NAG Library Manual

Mark 25
scanf("%"NAG_IFMT"%*[\n] ", &n);
#endif
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))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Read A and B from data file */
#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);
if (uplo == Nag_Upper)
{
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)
#ifdef _WIN32
            scanf_s("%lf", &A_UPPER(i, j));
#else
            scanf("%lf", &A_UPPER(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    for (i = 1; i <= n; ++i)
    {
        for (j = i; j <= n; ++j)
#ifdef _WIN32
            scanf_s("%lf", &B_UPPER(i, j));
#else
            scanf("%lf", &B_UPPER(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
}
else
{
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= i; ++j)
#ifdef _WIN32
            scanf_s("%lf", &A_LOWER(i, j));
#else
            scanf("%lf", &A_LOWER(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
} else
{ for (i = 1; i <= n; ++i)
    { for (j = 1; j <= i; ++j)
#ifdef _WIN32
            scanf_s("%lf", &A_LOWER(i, j));
#else
            scanf("%lf", &A_LOWER(i, j));
#endif
}
#ifdef _WIN32
    scanf_s("%*[\n ] ");
#else
    scanf("%*[\n ] ");
#endif
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= i; ++j)
    { #ifdef _WIN32
        scanf_s("%lf", &B_LOWER(i, j));
    #else
        scanf("%lf", &B_LOWER(i, j));
    #endif
    }
    #ifdef _WIN32
    scanf_s("%*[\n ] ");
    #else
    scanf("%*[\n ] ");
    #endif
}
/* Compute the Cholesky factorization of B */
/* nag_dpptrf (f07gdc). */
/* Cholesky factorization of real symmetric */
/* positive-definite matrix, packed storage */
/* */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dpptrf (f07gdc).
    ", fail.message);
    exit_status = 1;
    goto END;
}
/* Reduce the problem to standard form C*y = lambda*y, storing */
/* the result in A */
/* nag_dspgst (f08tec). */
/* Reduction to standard form of real symmetric-definite */
/* generalized eigenproblem Ax = lambda Bx, ABx = lambda x */
/* or BAx = lambda x, packed storage, B factorized by */
/* nag_dpptrf (f07gdc) */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dspgst (f08tec).
    ", fail.message);
    exit_status = 1;
    goto END;
}
/* Reduce C to tridiagonal form T = (Q**T)*C*Q */
/* nag_dsptrd (f08gef). */
/* Orthogonal reduction of real symmetric matrix to */
/* symmetric tridiagonal form, packed storage */
/* */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dsptrd (f08gef).
    ", fail.message);
    exit_status = 1;
    goto END;
}
/* Calculate the eigenvalues of T (same as C) */
/* nag_dsterf (f08jfc). */
/* All eigenvalues of real symmetric tridiagonal matrix, */
/* root-free variant of QL or QR */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dsterf (f08jfc).
    ", fail.message);
    exit_status = 1;
    goto END;
}
/* Print eigenvalues */
printf("Eigenvalues\n");
for (i = 1; i <= n; ++i)
    printf("%8.4f%s", d[i-1], i%9 == 0 || i == n?"\n":" ");
printf("\n");

END:
NAG_FREE(ap);
NAG_FREE(bp);
NAG_FREE(d);
NAG_FREE(e);
NAG_FREE(tau);

return exit_status;

10.2 Program Data

nag_dspgst (f08tec) Example Program Data
4 :Value of n
Nag_Lower :Value of uplo
0.24
0.39 -0.11
0.42 0.79 -0.11
-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

10.3 Program Results

nag_dspgst (f08tec) Example Program Results

Eigenvalues
-2.2254 -0.4548 0.1001 1.1270