NAG Library Function Document

nag_dsbgst (f08uec)

1 Purpose

nag_dsbgst (f08uec) reduces a real symmetric-definite generalized eigenproblem \(Az = \lambda Bz\) to the standard form \(Cy = \lambda y\), where \(A\) and \(B\) are band matrices, \(A\) is a real symmetric matrix, and \(B\) has been factorized by nag_dpbstf (f08ufc).

2 Specification

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

void nag_dsbgst (Nag_OrderType order, Nag_VectType vect, Nag_UploType uplo,
                Integer n, Integer ka, Integer kb, double ab[],
                Integer pdab, const double bb[], Integer pdbb, double x[],
                Integer pdx, NagError *fail)
```

3 Description

To reduce the real symmetric-definite generalized eigenproblem \(Az = \lambda Bz\) to the standard form \(Cy = \lambda y\), where \(A\), \(B\), and \(C\) are banded, nag_dsbgst (f08uec) must be preceded by a call to nag_dpbstf (f08ufc) which computes the split Cholesky factorization of the positive definite matrix \(B\): \(B = S^T S\). The split Cholesky factorization, compared with the ordinary Cholesky factorization, allows the work to be approximately halved.

This function overwrites \(A\) with \(C = X^T A X\), where \(X = S^{-1} Q\) and \(Q\) is a orthogonal matrix chosen (implicitly) to preserve the bandwidth of \(A\). The function also has an option to allow the accumulation of \(X\), and then, if \(z\) is an eigenvector of \(C\), \(Xz\) is an eigenvector of the original system.

4 References


5 Arguments

1: \(\mathbf{order}\) – Nag_OrderType

\(\text{Input}\)

\(\text{On entry: the order argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by order = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.}\)

\(\text{Constraint: order = Nag_RowMajor or Nag_ColMajor.}\)

2: \(\mathbf{vect}\) – Nag_VectType

\(\text{Input}\)

\(\text{On entry: indicates whether X is to be returned.}\)

\(\text{vect = Nag_DoNotForm}\)

\(X\) is not returned.
vect = Nag_FormX
X is returned.

Constraint: vect = Nag_DoNotForm or Nag_FormX.

3: uplo – Nag_UploType

On entry: indicates whether the upper or lower triangular part of \( A \) is stored.

uplo = Nag_Upper
The upper triangular part of \( A \) is stored.

uplo = Nag_Lower
The lower triangular part of \( A \) is stored.

Constraint: uplo = Nag_Upper or Nag_Lower.

4: n – Integer

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

Constraint: \( n \geq 0 \).

5: ka – Integer

On entry: if uplo = Nag_Upper, the number of superdiagonals, \( k_a \), of the matrix \( A \).
If uplo = Nag_Lower, the number of subdiagonals, \( k_a \), of the matrix \( A \).

Constraint: \( ka \geq 0 \).

6: kb – Integer

On entry: if uplo = Nag_Upper, the number of superdiagonals, \( k_b \), of the matrix \( B \).
If uplo = Nag_Lower, the number of subdiagonals, \( k_b \), of the matrix \( B \).

Constraint: \( ka \geq kb \geq 0 \).

7: ab[dim] – double

Input/Output

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

On entry: the upper or lower triangle of the \( n \) by \( n \) symmetric band matrix \( A \).

This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements of \( A_{ij} \), depends on the order and uplo arguments as follows:

if order = Nag_ColMajor and uplo = Nag_Upper,
\( A_{ij} \) is stored in \( ab[k_a + i - j + (j - 1) \times pdab] \), for \( j = 1, \ldots, n \) and
\( i = \max(1, j - k_a), \ldots, j \);
if order = Nag_ColMajor and uplo = Nag_Lower,
\( A_{ij} \) is stored in \( ab[i - j + (j - 1) \times pdab] \), for \( j = 1, \ldots, n \) and
\( i = j, \ldots, \min(n, j + k_a) \);
if order = Nag_RowMajor and uplo = Nag_Upper,
\( A_{ij} \) is stored in \( ab[j - i + (i - 1) \times pdab] \), for \( i = 1, \ldots, n \) and
\( j = i, \ldots, \min(n, i + k_a) \);
if order = Nag_RowMajor and uplo = Nag_Lower,
\( A_{ij} \) is stored in \( ab[k_a + j - i + (i - 1) \times pdab] \), for \( i = 1, \ldots, n \) and
\( j = \max(1, i - k_a), \ldots, i \).

On exit: the upper or lower triangle of \( ab \) is overwritten by the corresponding upper or lower triangle of \( C \) as specified by uplo.
8:    **pdab** – Integer

    On entry: the stride separating row or column elements (depending on the value of **order**) of the matrix **A** in the array **ab**.

    Constraint: **pdab** ≥ **ka** + 1.

9:    **bb[dim]** – const double

    Note: the dimension, **dim**, of the array **bb** must be at least max(1, **pddb** × **n**).

    On entry: the banded split Cholesky factor of **B** as specified by **uplo**,** n** and **kb** and returned by **nag_dpbstf** (**f08ufc**).

10:   **pddb** – Integer

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

    Constraint: **pddb** ≥ **kb** + 1.

11:   **x[dim]** – double

    Note: the dimension, **dim**, of the array **x** must be at least
    
    max(1, **pdx** × **n**) when **vect** = Nag_FormX;
    1 when **vect** = Nag_DoNotForm.

    The (i,j)th element of the matrix **X** is stored in
    
    **x**[(j−1) × **pdx** + i − 1] when **order** = Nag_ColMajor;
    **x**[(i−1) × **pdx** + j − 1] when **order** = Nag_RowMajor.

    On exit: the n by n matrix **X** = **S**⁻¹**Q**, if **vect** = Nag_FormX.

    If **vect** = Nag_DoNotForm, **x** is not referenced.

12:   **pdx** – Integer

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

    Constraints:
    
    if **vect** = Nag_FormX, **pdx** ≥ max(1, **n**);
    if **vect** = Nag_DoNotForm, **pdx** ≥ 1.

13:   **fail** – NagError*

    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_ENUM_INT_2**

On entry, **vect** = ⟨**value**⟩, **pdx** = ⟨**value**⟩ and **n** = ⟨**value**⟩.  Constraint: if **vect** = Nag_FormX, **pdx** ≥ max(1, **n**);

if **vect** = Nag_DoNotForm, **pdx** ≥ 1.
On entry, $ka = \langle value \rangle$.  
Constraint: $ka \geq 0$.  

On entry, $n = \langle value \rangle$.  
Constraint: $n \geq 0$.  

On entry, $pdab = \langle value \rangle$.  
Constraint: $pdab > 0$.  

On entry, $pdbb = \langle value \rangle$.  
Constraint: $pdbb > 0$.  

On entry, $pdx = \langle value \rangle$.  
Constraint: $pdx > 0$.  

On entry, $ka = \langle value \rangle$ and $kb = \langle value \rangle$.  
Constraint: $ka \geq kb \geq 0$.  

On entry, $pdab = \langle value \rangle$ and $ka = \langle value \rangle$.  
Constraint: $pdab \geq ka + 1$.  

On entry, $pdbb = \langle value \rangle$ and $kb = \langle value \rangle$.  
Constraint: $pdbb \geq kb + 1$.  

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.  

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.  

Forming the reduced matrix $C$ is a stable procedure. However it involves implicit multiplication by $B^{-1}$.  
When nag_dsbgst (f08uec) 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.  

nag_dsbgst (f08uec) is not threaded by NAG in any implementation.  
nag_dsbgst (f08uec) 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.  

The total number of floating-point operations is approximately $6n^2k_B$, when vect = Nag_DoNotForm, assuming $n \gg k_A, k_B$; there are an additional $(3/2)n^3(k_B/k_A)$ operations when vect = Nag_FormX.
The complex analogue of this function is nag_zhbgst (f08usc).

10 Example

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

\[
A = \begin{pmatrix}
0.24 & 0.39 & 0.42 & 0.00 \\
0.39 & -0.11 & 0.79 & 0.63 \\
0.42 & 0.79 & -0.25 & 0.48 \\
0.00 & 0.63 & 0.48 & -0.03
\end{pmatrix}
\quad \text{and} \quad
B = \begin{pmatrix}
2.07 & 0.95 & 0.00 & 0.00 \\
0.95 & 1.69 & -0.29 & 0.00 \\
0.00 & -0.29 & 0.65 & -0.33 \\
0.00 & 0.00 & -0.33 & 1.17
\end{pmatrix}.
\]

Here \(A\) is symmetric, \(B\) is symmetric positive definite, and \(A\) and \(B\) are treated as band matrices. \(B\) must first be factorized by nag_dpbstf (f08ufc). The program calls nag_dsbgst (f08uec) to reduce the problem to the standard form \(C y = \lambda y\), then nag_dsbtrd (f08hec) to reduce \(C\) to tridiagonal form, and nag_dsterf (f08jfc) to compute the eigenvalues.

10.1 Program Text

```c
/* nag_dsbgst (f08uec) Example Program. */

/* Copyright 2014 Numerical Algorithms Group. */

/* Mark 7, 2001. */

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

int main(void)
{
    /* Scalars */
    Integer i, j, k1, k2, ka, kb, n, pdab, pdbb, pdx, d_len, e_len;
    Integer exit_status = 0;
    NagError fail;
    Nag_UploType uplo;
    Nag_OrderType order;
    /* Arrays */
    char nag_enum_arg[40];
    double *ab = 0, *bb = 0, *d = 0, *e = 0, *x = 0;

    #ifdef NAG_COLUMN_MAJOR
        #define AB_UPPER(I, J) ab[(J-1)*pdab + k1 + I-J-1]
        #define AB_LOWER(I, J) ab[(I-1)*pdab + J-I]
        #define BB_UPPER(I, J) bb[(J-1)*pdbb + k2 + I-J-1]
        #define BB_LOWER(I, J) bb[(I-1)*pdbb + J-I-1]
        order = Nag_ColMajor;
    #else
        #define AB_UPPER(I, J) ab[(J-1)*pdab + I-J]
        #define AB_LOWER(I, J) ab[(I-1)*pdab + J-I]
        #define BB_UPPER(I, J) bb[(J-1)*pdbb + J-I]
        #define BB_LOWER(I, J) bb[(I-1)*pdbb + k2 + J-I-1]
        order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);
    printf("nag_dsbgst (f08uec) 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"%NAG_IFMT"%NAG_IFMT"%NAG_IFMT"%[^\n] ", &n, &ka, &kb);
    #endif
    return exit_status;
}
```

Mark 25
```c
/* Read whether Upper or Lower part of A is stored */
#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);
/* Read A and B from data file */
k1 = ka + 1;
k2 = kb + 1;
if (uplo == Nag_Upper)
    for (i = 1; i <= n; ++i)
        { for (j = i; j <= MIN(i+ka, n); ++j)
            #ifdef _WIN32
                scanf_s("\%lf", &AB_UPPER(i, j));
            #else
                scanf("\%lf", &AB_UPPER(i, j));
            #endif
        }
    #ifdef _WIN32
        scanf_s("\%\n ");
    #else
        scanf("\%\n ");
    #endif
else
    { for (i = 1; i <= n; ++i)
        { for (j = MAX(1, i-ka); j <= i; ++j)
            #ifdef _WIN32
                scanf_s("\%lf", &AB_LOWER(i, j));
            #else
                scanf("\%lf", &AB_LOWER(i, j));
            #endif
        }
    #ifdef _WIN32
        scanf_s("\%\n ");
    #else
        scanf("\%\n ");
    #endif
    if (uplo == Nag_Upper)
        { for (i = 1; i <= n; ++i)
            { for (j = i; j <= n; ++j)
                #ifdef _WIN32
                    scanf_s("\%lf", &AB_UPPER(i, j));
                #else
                    scanf("\%lf", &AB_UPPER(i, j));
                #endif
            }
        #ifdef _WIN32
            scanf_s("\%\n ");
        #else
            scanf("\%\n ");
        #endif
        if (uplo == Nag_Upper)
            { for (i = 1; i <= n; ++i)
                { for (j = i; j <= n; ++j)
                    #ifdef _WIN32
                        scanf_s("\%lf", &AB_UPPER(i, j));
                    #else
                        scanf("\%lf", &AB_UPPER(i, j));
                    #endif
                }
            #ifdef _WIN32
                scanf_s("\%\n ");
            #else
                scanf("\%\n ");
            #endif
        }
    }
#endif
```
for (i = 1; i <= n; ++i)
{
    for (j = MAX(1, i-kb); j <= i; ++j)
    {
        scanf("%lf", &BB_LOWER(i, j));
    }
}

/* Compute the split Cholesky factorization of B */
/* nag_dpbbst (f08ufc). */
/* Computes a split Cholesky factorization of real symmetric */
/* positive-definite band matrix A */
/* nag_dpbbstf(order, uplo, n, kb, bb, pdbb, &fail); */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dpbbstf (f08ufc).\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 */
/* nag_dsbbgst (f08uec). */
/* Reduction of real symmetric-definite banded generalized */
/* eigenproblem Ax = lambda Bx to standard form */
/* Cy = lambda y, such that C has the same bandwidth as A */
/* nag_dsbbgst(order, Nag_DoNotForm, uplo, n, ka, kb, ab, pdab, bb, pdbb, */
/* x, pdx, &fail); */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dsbbgst (f08uec).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Reduce C to tridiagonal form T = (Q**T)*C*Q */
/* nag_dsbtrd (f08hec). */
/* Orthogonal reduction of real symmetric band matrix to */
/* symmetric tridiagonal form */
/* nag_dsbtrd(order, Nag_DoNotForm, uplo, n, ka, ab, pdab, d, e, */
/* x, pdx, &fail); */
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dsbtrd (f08hec).\n%s\n", 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 
 */
 nag_dsterf(n, d, e, &fail);
 if (fail.code != NE_NOERROR)
 {
   printf("Error from nag_dsterf (f08jfc).\n%s\n", fail.message);
   exit_status = 1;
   goto END;
 }
 /* Print eigenvalues */
 printf(" Eigenvalues\n");
 for (i = 0; i < n; ++i)
   printf(" %8.4lf", d[i]);
 printf("\n");
END:
NAG_FREE(ab);
NAG_FREE(bb);
NAG_FREE(d);
NAG_FREE(e);
NAG_FREE(x);
return exit_status;
}

10.2 Program Data

nag_dsbgst (f08uec) Example Program Data
4  2  1  :Values of n, ka and kb
   Nag_Lower  :Value of uplo
0.24
0.39 -0.11
0.42  0.79 -0.25
   0.63  0.48 -0.03  :End of matrix A
2.07
0.95  1.69
   -0.29  0.65
   -0.33  1.17  :End of matrix B

10.3 Program Results

nag_dsbgst (f08uec) Example Program Results

Eigenvalues
-0.8305 -0.6401  0.0992  1.8525