

# NAG Library Function Document

## nag\_dsygst (f08sec)

### 1 Purpose

nag\_dsygst (f08sec) 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\_dpotrf (f07fdc).

### 2 Specification

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

void nag_dsygst (Nag_OrderType order, Nag_ComputeType comp_type,
                 Nag_UploType uplo, Integer n, double a[], Integer pda, const double b[],
                 Integer pdb, 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  $Cy = \lambda y$ , nag\_dsygst (f08sec) must be preceded by a call to nag\_dpotrf (f07fdc) 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.

			order = Nag_ColMajor			order = Nag_RowMajor		
comp_type	Problem	uplo	$B$	$C$	$z$	$B$	$C$	$z$
1	$Az = \lambda Bz$	Nag_Upper Nag_Lower	$U^T U$ $LL^T$	$U^{-T} A U^{-1}$ $L^{-1} A L^{-T}$	$U^{-1} y$ $L^{-T} y$	$U U^T$ $L^T L$	$U^{-1} A U^{-T}$ $L^{-T} A L^{-1}$	$U^{-T} y$ $L^{-1} y$
2	$ABz = \lambda z$	Nag_Upper Nag_Lower	$U^T U$ $LL^T$	$U A U^T$ $L^T A L$	$U^{-1} y$ $L^{-T} y$	$U U^T$ $L^T L$	$U^T A U$ $L A L^T$	$U^{-T} y$ $L^{-1} y$
3	$BAz = \lambda z$	Nag_Upper Nag_Lower	$U^T U$ $LL^T$	$U A U^T$ $L^T A L$	$U^T y$ $L y$	$U U^T$ $L^T L$	$U^T A U$ $L A L^T$	$U y$ $L^T y$

### 4 References

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

### 5 Arguments

- 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 2.3.1.3 in How to Use the NAG Library and its Documentation 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,  $C = U^{-T}AU^{-1}$  when **order** = Nag\_ColMajor and  $C = U^{-1}AU^{-T}$  when **order** = Nag\_RowMajor;

if **uplo** = Nag\_Lower,  $C = L^{-1}AL^{-T}$  when **order** = Nag\_ColMajor and  $C = L^{-T}AL^{-1}$  when **order** = Nag\_RowMajor.

**comp\_type** = Nag\_Compute\_2 or Nag\_Compute\_3

if **uplo** = Nag\_Upper,  $C = UAU^T$  when **order** = Nag\_ColMajor and  $C = U^T AU$  when **order** = Nag\_RowMajor;

if **uplo** = Nag\_Lower,  $C = L^T AL$  when **order** = Nag\_ColMajor and  $C = LAL^T$  when **order** = Nag\_RowMajor.

*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  $B = U^T U$  when **order** = Nag\_ColMajor and  $B = UU^T$  when **order** = Nag\_RowMajor.

**uplo** = Nag\_Lower

The lower triangular part of  $A$  is stored and  $B = LL^T$  when **order** = Nag\_ColMajor and  $B = L^T L$  when **order** = Nag\_RowMajor.

*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: **a**[*dim*] – double *Input/Output*

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

*On entry:* the  $n$  by  $n$  symmetric matrix  $A$ .

If **order** = Nag\_ColMajor,  $A_{ij}$  is stored in **a**[( $j - 1$ )  $\times$  **pda** +  $i - 1$ ].

If **order** = Nag\_RowMajor,  $A_{ij}$  is stored in **a**[( $i - 1$ )  $\times$  **pda** +  $j - 1$ ].

If **uplo** = Nag\_Upper, the upper triangular part of  $A$  must be stored and the elements of the array below the diagonal are not referenced.

If **uplo** = Nag\_Lower, the lower triangular part of  $A$  must be stored and the elements of the array above the diagonal are not referenced.

*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**.

- 6: **pda** – Integer *Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) of the matrix *A* in the array **a**.  
*Constraint:* **pda**  $\geq \max(1, \mathbf{n})$ .
- 7: **b[dim]** – const double *Input*  
**Note:** the dimension, *dim*, of the array **b** must be at least  $\max(1, \mathbf{pdb} \times \mathbf{n})$ .  
*On entry:* the Cholesky factor of *B* as specified by **uplo** and returned by nag\_dpotr (f07fdc).
- 8: **pdb** – Integer *Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) of the matrix *B* in the array **b**.  
*Constraint:* **pdb**  $\geq \max(1, \mathbf{n})$ .
- 9: **fail** – NagError \* *Input/Output*  
The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.  
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_INT

On entry, **n** =  $\langle value \rangle$ .  
*Constraint:* **n**  $\geq 0$ .  
On entry, **pda** =  $\langle value \rangle$ .  
*Constraint:* **pda**  $> 0$ .  
On entry, **pdb** =  $\langle value \rangle$ .  
*Constraint:* **pdb**  $> 0$ .

### NE\_INT\_2

On entry, **pda** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .  
*Constraint:* **pda**  $\geq \max(1, \mathbf{n})$ .  
On entry, **pdb** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .  
*Constraint:* **pdb**  $\geq \max(1, \mathbf{n})$ .

### 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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

## NE\_NO\_LICENCE

Your licence key may have expired or may not have been installed correctly.  
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

## 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 nag\_dsygst (f08sec) 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\_dsygst (f08sec) 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\_zhegst (f08ssc).

## 10 Example

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

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

Here  $B$  is symmetric positive definite and must first be factorized by nag\_dpotrf (f07fdc). The program calls nag\_dsygst (f08sec) to reduce the problem to the standard form  $Cy = \lambda y$ ; then nag\_dsytrd (f08fec) to reduce  $C$  to tridiagonal form, and nag\_dsterf (f08jfc) to compute the eigenvalues.

### 10.1 Program Text

```
/* nag_dsygst (f08sec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

int main(void)
{
    /* Scalars */
    Integer i, j, n, pda, pdb, 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 *a = 0, *b = 0, *d = 0, *e = 0, *tau = 0;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda + I - 1]
#define B(I, J) b[(J-1)*pdb + I - 1]
    order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + J - 1]
#define B(I, J) b[(I-1)*pdb + J - 1]
    order = Nag_RowMajor;
#endif

    INIT_FAIL(fail);

    printf("nag_dsygst (f08sec) 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
#ifdef NAG_COLUMN_MAJOR
    pda = n;
    pdb = n;
#else
    pda = n;
    pdb = n;
#endif
    d_len = n;
    e_len = n - 1;
    tau_len = n - 1;

    /* Allocate memory */
    if (!(a = NAG_ALLOC(n * n, double)) ||
        !(b = NAG_ALLOC(n * n, 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, (unsigned)_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(i, j));
#else
                scanf("%lf", &A(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(i, j));
#else
            scanf("%lf", &B(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(i, j));
#else
                scanf("%lf", &A(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(i, j));
#else
                scanf("%lf", &B(i, j));
#endif
        }
#ifdef _WIN32
        scanf_s("%*[^\\n] ");
#else
        scanf("%*[^\\n] ");
#endif
    }

    /* Compute the Cholesky factorization of B */
    /* nag_dpotrf (f07fdc).
     * Cholesky factorization of real symmetric
     * positive-definite matrix
     */
    nag_dpotrf(order, uplo, n, b, pdb, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_dpotrf (f07fdc).\\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_dsygst (f08sec).
     * Reduction to standard form of real symmetric-definite
     * generalized eigenproblem Ax = lambda Bx, ABx = lambda x
     * or BAx = lambda x, B factorized by nag_dpotrf (f07fdc)
     */
    nag_dsygst(order, Nag_Compute_1, uplo, n, a, pda, b, pdb, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_dsygst (f08sec).\\n%s\\n", fail.message);
        exit_status = 1;
    }

```

```

    goto END;
}
/* Reduce C to tridiagonal form T = (Q^T)*C*Q */
/* nag_dsytrd (f08fec).
 * Orthogonal reduction of real symmetric matrix to
 * symmetric tridiagonal form
 */
nag_dsytrd(order, uplo, n, a, pda, d, e, tau, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_dsytrd (f08fec).\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 = 1; i <= n; ++i)
    printf("%8.4f%s", d[i - 1], i % 9 == 0 ? "\n" : " ");
printf("\n");
END:
    NAG_FREE(a);
    NAG_FREE(b);
    NAG_FREE(d);
    NAG_FREE(e);
    NAG_FREE(tau);

    return exit_status;
}

```

## 10.2 Program Data

```

nag_dsygst (f08sec) Example Program Data
4                               :Value of n
Nag_Lower                      :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

```

## 10.3 Program Results

```

nag_dsygst (f08sec) Example Program Results

```

```

Eigenvalues
-2.2254  -0.4548   0.1001   1.1270

```

---