# NAG Library Function Document

# nag_real_symm_general_eigensystem (f02aec)

## 1 Purpose

nag_real_symm_general_eigensystem (f02aec) calculates all the eigenvalues and eigenvectors of $Ax = \lambda Bx$, where $A$ is a real symmetric matrix and $B$ is a real symmetric positive definite matrix.

## 2 Specification

```
#include <nag.h>
#include <nagf02.h>
```

```
void nag_real_symm_general_eigensystem (Integer n, double a[], Integer tda,
     double b[], Integer tdb, double r[], double v[], Integer tdv,
     NagError *fail)
```

## 3 Description

The problem is reduced to the standard symmetric eigenproblem using Cholesky's method to decompose $B$ into triangular matrices $B = LL^{\mathrm{T}}$, where $L$ is lower triangular. Then $Ax = \lambda Bx$ implies $\left(L^{-1}AL^{-T}\right)(L^{\mathrm{T}}x) = \lambda(L^{\mathrm{T}}x)$; hence the eigenvalues of $Ax = \lambda Bx$ are those of $Py = \lambda y$, where $P$ is the symmetric matrix $L^{-1}AL^{-T}$. Householder's method is used to tridiagonalise the matrix $P$ and the eigenvalues are found using the $QL$ algorithm. An eigenvector $z$ of the derived problem is related to an eigenvector $x$ of the original problem by $z = L^{\mathrm{T}}x$. The eigenvectors $z$ are determined using the $QL$ algorithm and are normalized so that $z^{\mathrm{T}}z = 1$; the eigenvectors of the original problem are then determined by solving $L^{\mathrm{T}}x = z$, and are normalized so that $x^{\mathrm{T}}Bx = 1$.

## 4 References

Wilkinson J H and Reinsch C (1971) *Handbook for Automatic Computation II, Linear Algebra* Springer–Verlag

## 5 Arguments

1: **n** – Integer *Input*

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

*Constraint*: **n** $\geq 1$.

2: **a**[**n** $\times$ **tda**] – double *Input/Output*

**Note**: the $(i,j)$th element of the matrix $A$ is stored in **a**$[(i-1) \times$ **tda** $+ j - 1]$.

*On entry*: the upper triangle of the $n$ by $n$ symmetric matrix $A$. The elements of the array below the diagonal need not be set.

*On exit*: the lower triangle of the array is overwritten. The rest of the array is unchanged. See also Section 9

3: **tda** – Integer *Input*

*On entry*: the stride separating matrix column elements in the array **a**.

*Constraint*: **tda** $\geq$ **n**.

4: **b**[**n** × **tdb**] – double *Input/Output*

**Note**: the $(i, j)$th element of the matrix $B$ is stored in **b**$[(i-1) \times$ **tdb** $+ j - 1]$.

*On entry*: the upper triangle of the $n$ by $n$ symmetric positive definite matrix $B$. The elements of the array below the diagonal need not be set.

*On exit*: the elements below the diagonal are overwritten. The rest of the array is unchanged.

5: **tdb** – Integer *Input*

*On entry*: the stride separating matrix column elements in the array **b**.

*Constraint*: **tdb** $\geq$ **n**.

6: **r**[**n**] – double *Output*

*On exit*: the eigenvalues in ascending order.

7: **v**[**n** × **tdv**] – double *Output*

**Note**: the $(i, j)$th element of the matrix $V$ is stored in **v**$[(i-1) \times$ **tdv** $+ j - 1]$.

*On exit*: the normalized eigenvectors, stored by columns; the $i$th column corresponds to the $i$th eigenvalue. The eigenvectors $x$ are normalized so that $x^{\mathrm{T}}Bx = 1$. See also Section 9

8: **tdv** – Integer *Input*

*On entry*: the stride separating matrix column elements in the array **v**.

*Constraint*: **tdv** $\geq$ **n**.

9: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

## 6 Error Indicators and Warnings

**NE_2_INT_ARG_LT**

On entry, **tda** = ⟨*value*⟩ while **n** = ⟨*value*⟩. These arguments must satisfy **tda** $\geq$ **n**.

On entry, **tdb** = ⟨*value*⟩ while **n** = ⟨*value*⟩. These arguments must satisfy **tdb** $\geq$ **n**.

On entry, **tdv** = ⟨*value*⟩ while **n** = ⟨*value*⟩. These arguments must satisfy **tdv** $\geq$ **n**.

**NE_ALLOC_FAIL**

Dynamic memory allocation failed.

**NE_INT_ARG_LT**

On entry, **n** = ⟨*value*⟩.
Constraint: **n** $\geq 1$.

**NE_NOT_POS_DEF**

The matrix $B$ is not positive definite, possibly due to rounding errors.

**NE_TOO_MANY_ITERATIONS**

More than ⟨*value*⟩ iterations are required to isolate all the eigenvalues.

## 7 Accuracy

In general this function is very accurate. However, if $B$ is ill-conditioned with respect to inversion, the eigenvectors could be inaccurately determined. For a detailed error analysis see pages 310, 222 and 235 of Wilkinson and Reinsch (1971).

## 8 Parallelism and Performance

Not applicable.

## 9 Further Comments

The time taken by nag_real_symm_general_eigensystem (f02aec) is approximately proportional to $n^3$.

The function may be called with the same actual array supplied for arguments **a** and **v**, in which case the eigenvectors will overwrite the original matrix $A$.

## 10 Example

To calculate all the eigenvalues and eigenvectors of the general symmetric eigenproblem $Ax = \lambda Bx$ where $A$ is the symmetric matrix

$$\begin{pmatrix} 0.5 & 1.5 & 6.6 & 4.8 \\ 1.5 & 6.5 & 16.2 & 8.6 \\ 6.6 & 16.2 & 37.6 & 9.8 \\ 4.8 & 8.6 & 9.8 & -17.1 \end{pmatrix}$$

and $B$ is the symmetric positive definite matrix

$$\begin{pmatrix} 1 & 3 & 4 & 1 \\ 3 & 13 & 16 & 11 \\ 4 & 16 & 24 & 18 \\ 1 & 11 & 18 & 27 \end{pmatrix}.$$

### 10.1 Program Text

```
/* nag_real_symm_general_eigensystem (f02aec) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 2 revised, 1992.
 * Mark 8 revised, 2004.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagf02.h>

#define A(I, J) a[(I) *tda + J]
#define B(I, J) b[(I) *tdb + J]
#define V(I, J) v[(I) *tdv + J]

int main(void)
{

  Integer  exit_status = 0, i, j, n, tda, tdb, tdv;
  NagError fail;
  double   *a = 0, *b = 0, *r = 0, *v = 0;

  INIT_FAIL(fail);

  printf("nag_real_symm_general_eigensystem (f02aec) Example Program"
          " Results\n");
  /* Skip heading in data file */
```

```
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"", &n);
#else
  scanf("%"NAG_IFMT"", &n);
#endif
  if (n >= 1)
    {
      if (!(a = NAG_ALLOC(n*n, double)) ||
          !(b = NAG_ALLOC(n*n, double)) ||
          !(r = NAG_ALLOC(n, double)) ||
          !(v = NAG_ALLOC(n*n, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
      tda = n;
      tdb = n;
      tdv = n;
    }
  else
    {
      printf("Invalid n.\n");
      exit_status = 1;
      return exit_status;
    }
  for (i = 0; i < n; i++)
    {
      for (j = 0; j < n; j++)
#ifdef _WIN32
        scanf_s("%lf", &A(i, j));
#else
        scanf("%lf", &A(i, j));
#endif
      for (j = 0; j < n; j++)
#ifdef _WIN32
        scanf_s("%lf", &B(i, j));
#else
        scanf("%lf", &B(i, j));
#endif
    }
  /* nag_real_symm_general_eigensystem (f02aec).
   * All eigenvalues and eigenvectors of generalized real
   * symmetric-definite eigenproblem
   */
  nag_real_symm_general_eigensystem(n, a, tda, b, tdb, r, v, tdv, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf(
              "Error from nag_real_symm_general_eigensystem (f02aec).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  printf("Eigenvalues\n");
  for (i = 0; i < n; i++)
    printf("%9.4f%s", r[i], (i%8 == 7 || i == n-1)?"\n":" ");
  printf("Eigenvectors\n");
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      printf("%9.4f%s", V(i, j), (j%8 == 7 || j == n-1)?"\n":" ");
 END:
  NAG_FREE(a);
```

```
  NAG_FREE(b);
  NAG_FREE(r);
  NAG_FREE(v);
  return exit_status;
}
```

## 10.2 Program Data

```
nag_real_symm_general_eigensystem (f02aec) Example Program Data
  4
   0.5   1.5   6.6   4.8     1.0   3.0   4.0   1.0
   1.5   6.5  16.2   8.6     3.0  13.0  16.0  11.0
   6.6  16.2  37.6   9.8     4.0  16.0  24.0  18.0
   4.8   8.6   9.8 -17.1     1.0  11.0  18.0  27.0
```

## 10.3 Program Results

```
nag_real_symm_general_eigensystem (f02aec) Example Program Results
Eigenvalues
  -3.0000   -1.0000    2.0000    4.0000
Eigenvectors
  -4.3500   -2.0500   -3.9500    2.6500
   0.0500    0.1500    0.8500    0.0500
   1.0000    0.5000    0.5000   -1.0000
  -0.5000   -0.5000   -0.5000    0.5000
```