

# NAG Library Function Document

## nag\_dpqr (f08jgc)

### 1 Purpose

nag\_dpqr (f08jgc) computes all the eigenvalues and, optionally, all the eigenvectors of a real symmetric positive definite tridiagonal matrix, or of a real symmetric positive definite matrix which has been reduced to tridiagonal form.

### 2 Specification

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

void nag_dpqr (Nag_OrderType order, Nag_ComputeZType compz, Integer n,
               double d[], double e[], double z[], Integer pdz, NagError *fail)
```

### 3 Description

nag\_dpqr (f08jgc) computes all the eigenvalues and, optionally, all the eigenvectors of a real symmetric positive definite tridiagonal matrix  $T$ . In other words, it can compute the spectral factorization of  $T$  as

$$T = Z\Lambda Z^T,$$

where  $\Lambda$  is a diagonal matrix whose diagonal elements are the eigenvalues  $\lambda_i$ , and  $Z$  is the orthogonal matrix whose columns are the eigenvectors  $z_i$ . Thus

$$Tz_i = \lambda_i z_i, \quad i = 1, 2, \dots, n.$$

The function may also be used to compute all the eigenvalues and eigenvectors of a real symmetric positive definite matrix  $A$  which has been reduced to tridiagonal form  $T$ :

$$\begin{aligned} A &= QTQ^T, \text{ where } Q \text{ is orthogonal} \\ &= (QZ)\Lambda(QZ)^T. \end{aligned}$$

In this case, the matrix  $Q$  must be formed explicitly and passed to nag\_dpqr (f08jgc), which must be called with **compz** = Nag\_UpdateZ. The functions which must be called to perform the reduction to tridiagonal form and form  $Q$  are:

|                             |   |
|-----------------------------|---|
| full matrix                 | nag_dsytrd (f08fec) and nag_dorgtr (f08ffc)       |
| full matrix, packed storage | nag_dsptd (f08gec) and nag_dopgtr (f08gfc)        |
| band matrix                 | nag_dsbtrd (f08hec) with <b>vect</b> = Nag_FormQ. |

nag\_dpqr (f08jgc) first factorizes  $T$  as  $LDL^T$  where  $L$  is unit lower bidiagonal and  $D$  is diagonal. It forms the bidiagonal matrix  $B = LD^{\frac{1}{2}}$ , and then calls nag\_dbdsqr (f08mec) to compute the singular values of  $B$  which are the same as the eigenvalues of  $T$ . The method used by the function allows high relative accuracy to be achieved in the small eigenvalues of  $T$ . The eigenvectors are normalized so that  $\|z_i\|_2 = 1$ , but are determined only to within a factor  $\pm 1$ .

### 4 References

Barlow J and Demmel J W (1990) Computing accurate eigensystems of scaled diagonally dominant matrices *SIAM J. Numer. Anal.* **27** 762–791

## 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 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: **compz** – Nag\_ComputeZType *Input*  
*On entry:* indicates whether the eigenvectors are to be computed.  
**compz** = Nag\_NotZ  
Only the eigenvalues are computed (and the array **z** is not referenced).  
**compz** = Nag\_UpdateZ  
The eigenvalues and eigenvectors of  $A$  are computed (and the array **z** must contain the matrix  $Q$  on entry).  
**compz** = Nag\_InitZ  
The eigenvalues and eigenvectors of  $T$  are computed (and the array **z** is initialized by the function).  
*Constraint:* **compz** = Nag\_NotZ, Nag\_UpdateZ or Nag\_InitZ.
- 3: **n** – Integer *Input*  
*On entry:*  $n$ , the order of the matrix  $T$ .  
*Constraint:*  $n \geq 0$ .
- 4: **d**[*dim*] – double *Input/Output*  
**Note:** the dimension, *dim*, of the array **d** must be at least  $\max(1, n)$ .  
*On entry:* the diagonal elements of the tridiagonal matrix  $T$ .  
*On exit:* the  $n$  eigenvalues in descending order, unless **fail.code** = NE\_CONVERGENCE or NE\_POS\_DEF, in which case **d** is overwritten.
- 5: **e**[*dim*] – double *Input/Output*  
**Note:** the dimension, *dim*, of the array **e** must be at least  $\max(1, n - 1)$ .  
*On entry:* the off-diagonal elements of the tridiagonal matrix  $T$ .  
*On exit:* **e** is overwritten.
- 6: **z**[*dim*] – double *Input/Output*  
**Note:** the dimension, *dim*, of the array **z** must be at least  
 $\max(1, \mathbf{pdz} \times n)$  when **compz** = Nag\_UpdateZ or Nag\_InitZ;  
1 when **compz** = Nag\_NotZ.  
The ( $i, j$ )th element of the matrix  $Z$  is stored in  
 $\mathbf{z}[(j - 1) \times \mathbf{pdz} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{z}[(i - 1) \times \mathbf{pdz} + j - 1]$  when **order** = Nag\_RowMajor.  
*On entry:* if **compz** = Nag\_UpdateZ, **z** must contain the orthogonal matrix  $Q$  from the reduction to tridiagonal form.  
If **compz** = Nag\_InitZ, **z** need not be set.

On exit: if **compz** = Nag\_UpdateZ or Nag\_InitZ, the  $n$  required orthonormal eigenvectors stored as columns of  $Z$ ; the  $i$ th column corresponds to the  $i$ th eigenvalue, where  $i = 1, 2, \dots, n$ , unless **fail.code** = NE\_CONVERGENCE or NE\_POS\_DEF.

If **compz** = Nag\_NotZ, **z** is not referenced.

7: **pdz** – Integer *Input*

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

Constraints:

if **compz** = Nag\_UpdateZ or Nag\_InitZ, **pdz**  $\geq \max(1, n)$ ;  
if **compz** = Nag\_NotZ, **pdz**  $\geq 1$ .

8: **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\_CONVERGENCE

The algorithm to compute the singular values of the Cholesky factor  $B$  failed to converge;  $\langle value \rangle$  off-diagonal elements did not converge to zero.

### NE\_ENUM\_INT\_2

On entry, **compz** =  $\langle value \rangle$ , **pdz** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

Constraint: if **compz** = Nag\_UpdateZ or Nag\_InitZ, **pdz**  $\geq \max(1, n)$ ;  
if **compz** = Nag\_NotZ, **pdz**  $\geq 1$ .

### NE\_INT

On entry, **n** =  $\langle value \rangle$ .

Constraint: **n**  $\geq 0$ .

On entry, **pdz** =  $\langle value \rangle$ .

Constraint: **pdz**  $> 0$ .

### 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 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.

**NE\_POS\_DEF**

The leading minor of order *<value>* is not positive definite and the Cholesky factorization of  $T$  could not be completed. Hence  $T$  itself is not positive definite.

**7 Accuracy**

The eigenvalues and eigenvectors of  $T$  are computed to high relative accuracy which means that if they vary widely in magnitude, then any small eigenvalues (and corresponding eigenvectors) will be computed more accurately than, for example, with the standard  $QR$  method. However, the reduction to tridiagonal form (prior to calling the function) may exclude the possibility of obtaining high relative accuracy in the small eigenvalues of the original matrix if its eigenvalues vary widely in magnitude.

To be more precise, let  $H$  be the tridiagonal matrix defined by  $H = DTD$ , where  $D$  is diagonal with  $d_{ii} = t_{ii}^{-\frac{1}{2}}$ , and  $h_{ii} = 1$  for all  $i$ . If  $\lambda_i$  is an exact eigenvalue of  $T$  and  $\tilde{\lambda}_i$  is the corresponding computed value, then

$$|\tilde{\lambda}_i - \lambda_i| \leq c(n)\epsilon\kappa_2(H)\lambda_i$$

where  $c(n)$  is a modestly increasing function of  $n$ ,  $\epsilon$  is the **machine precision**, and  $\kappa_2(H)$  is the condition number of  $H$  with respect to inversion defined by:  $\kappa_2(H) = \|H\| \cdot \|H^{-1}\|$ .

If  $z_i$  is the corresponding exact eigenvector of  $T$ , and  $\tilde{z}_i$  is the corresponding computed eigenvector, then the angle  $\theta(\tilde{z}_i, z_i)$  between them is bounded as follows:

$$\theta(\tilde{z}_i, z_i) \leq \frac{c(n)\epsilon\kappa_2(H)}{relgap_i}$$

where  $relgap_i$  is the relative gap between  $\lambda_i$  and the other eigenvalues, defined by

$$relgap_i = \min_{i \neq j} \frac{|\lambda_i - \lambda_j|}{(\lambda_i + \lambda_j)}.$$

**8 Parallelism and Performance**

nag\_dpqr (f08jgc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag\_dpqr (f08jgc) 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 typically about  $30n^2$  if **compz** = Nag\_NotZ and about  $6n^3$  if **compz** = Nag\_UpdateZ or Nag\_InitZ, but depends on how rapidly the algorithm converges. When **compz** = Nag\_NotZ, the operations are all performed in scalar mode; the additional operations to compute the eigenvectors when **compz** = Nag\_UpdateZ or Nag\_InitZ can be vectorized and on some machines may be performed much faster.

The complex analogue of this function is nag\_zpqr (f08juc).

## 10 Example

This example computes all the eigenvalues and eigenvectors of the symmetric positive definite tridiagonal matrix  $T$ , where

$$T = \begin{pmatrix} 4.16 & 3.17 & 0.00 & 0.00 \\ 3.17 & 5.25 & -0.97 & 0.00 \\ 0.00 & -0.97 & 1.09 & 0.55 \\ 0.00 & 0.00 & 0.55 & 0.62 \end{pmatrix}.$$

### 10.1 Program Text

```
/* nag_dpteqr (f08jgc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf08.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer i, j, n, pdz, d_len, e_len;
    Integer exit_status = 0;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    double *z = 0, *d = 0, *e = 0;

#ifdef NAG_COLUMN_MAJOR
#define Z(I, J) z[(J - 1) * pdz + I - 1]
    order = Nag_ColMajor;
#else
#define Z(I, J) z[(I - 1) * pdz + J - 1]
    order = Nag_RowMajor;
#endif

    INIT_FAIL(fail);

    printf("nag_dpteqr (f08jgc) 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
    pdz = n;
    d_len = n;
    e_len = n - 1;

    /* Allocate memory */
    if (!(z = NAG_ALLOC(n * n, double)) ||
        !(d = NAG_ALLOC(d_len, double)) || !(e = NAG_ALLOC(e_len, double)))
    {
        printf("Allocation failure\n");
    }
}
```

```

        exit_status = -1;
        goto END;
    }
    /* Read T from data file */
    for (i = 0; i < d_len; ++i)
#ifdef _WIN32
        scanf_s("%lf", &d[i]);
#else
        scanf("%lf", &d[i]);
#endif
    for (i = 0; i < e_len; ++i)
#ifdef _WIN32
        scanf_s("%lf", &e[i]);
#else
        scanf("%lf", &e[i]);
#endif
    /* Calculate all the eigenvalues and eigenvectors of T */
    /* nag_dpqr (f08jgc).
     * All eigenvalues and eigenvectors of real symmetric
     * positive-definite tridiagonal matrix, reduced from real
     * symmetric positive-definite matrix
     */
    nag_dpqr(order, Nag_InitZ, n, d, e, z, pdz, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_dpqr (f08jgc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
    /* Normalize the eigenvectors */
    for (j = 1; j <= n; j++) {
        for (i = n; i >= 1; i--) {
            Z(i, j) = Z(i, j) / Z(1, j);
        }
    }
    /* Print eigenvalues and eigenvectors */
    printf(" Eigenvalues\n");
    for (i = 0; i < n; ++i)
        printf("  %7.4lf", d[i]);
    printf("\n\n");
    /* nag_gen_real_mat_print (x04cac).
     * Print real general matrix (easy-to-use)
     */
    fflush(stdout);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, n,
                           z, pdz, "Eigenvectors", 0, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
}
END:
    NAG_FREE(d);
    NAG_FREE(e);
    NAG_FREE(z);
    return exit_status;
}

```

## 10.2 Program Data

```

nag_dpqr (f08jgc) Example Program Data
4                               :Value of N
4.16   5.25   1.09   0.62
3.17  -0.97   0.55           :End of matrix T

```

**10.3 Program Results**

nag\_dpteqr (f08jgc) Example Program Results

Eigenvalues

|        |        |        |        |
|--------|--------|--------|--------|
| 8.0023 | 1.9926 | 1.0014 | 0.1237 |
|--------|--------|--------|--------|

Eigenvectors

|   | 1       | 2       | 3       | 4       |
|---|---------|---------|---------|---------|
| 1 | 1.0000  | 1.0000  | 1.0000  | 1.0000  |
| 2 | 1.2121  | -0.6837 | -0.9964 | -1.2733 |
| 3 | -0.1711 | 0.9721  | -1.0962 | -3.4611 |
| 4 | -0.0127 | 0.3895  | -1.5807 | 3.8354  |

---