MATRIX EIGENPROBLEM LIBRARY ECC_EISPACK FOR TMS320C6000 REFERENCE MANUAL

e-mail: andrew.nesterov@softhome.net
e-mail: andrew.nesterov@techemail.com

November 7, 2003

Contents

Chapter 1

Introduction

1.1 Installation

1.1.1 DOS systems

1.1.2 Unix systems

1.1.3 OS/2 systems

1.2 Using the library

1.2.1 Header files

1.2.2 Linking

Chapter 2

Recommended Sequences

2.1 Real Generalized Eigenproblem

2.1.1 RGG Generalized Eigenvalues and Eigenvectors

Name

srgg — Driver for solving real generalized eigenproblem, single precision
drgg — Driver for solving real generalized eigenproblem, double precision

Synopsis

```
#include <ceispack.h>
void srgg (n, a, b, alfa, beta, matz, z, fv, ierr)
int n;
int matz;
float **a, **b, **z;
fcomplex *alfa;
float *beta, *fv;
int *ierr;

void drgg (n, a, b, alfa, beta, matz, z, fv, ierr)
int n;
int matz;
double **a, **b, **z;
dcomplex *alfa;
double *beta, *fv;
int *ierr;
```

n	— order of input/output matrices
a	— first square input matrix
b	— second square input matrix
alfa	— length n output array containing numerators of eigenvalues
beta	— length n output array containing denominators of eigenvalues
matz	— flag which causes computation of eigenvectors
z	— output square array containing n transposed (row) eigenvectors
fv	— temporary storage array of length n
ierr	— address of output variable containing error completion code

Diagnostics

On output the variable `*ierr` is set to 0 for normal return. If an error exit has been made, the eigenvalues would be correct for indices $[*ierr, n-1]$. In such a case none of the eigenvectors have yet been determined.

Description

Function rgg invokes the sequence of functions `qzhes, qzit, qzval` and optionally, `qzvec` from the eigensystem library C_EISPACK to find the eigenvalues and eigenvectors of the real generalized eigenproblem:

$$Ax = \lambda Bx,$$

where A and B are real general matrices of order n. The solution is determined using the QZ algorithm. It is *not* required for the B to be *non-singular*. No inversion of the B or its submatrices is performed. The QZ algorithm is a generalization of the QR algorithm and is reduced to the QR algorithm if $B \equiv I$, i.e. B is an identity matrix.

The functions that implement the QZ algorithm may also be applied for solving the generalized eigenproblem of order r:

$$(\lambda^r G_r + \lambda^{r-1} G_{r-1} + \ldots + G_0)x = 0$$

where the matrices A_i are assumed of order n. This eigenproblem can be reduced to the generalized problem

$$Az = \lambda Bz$$

by forming block-matrices A and B of order $m = rn$:

$$A = \begin{pmatrix} G_{r-1} & G_{r-2} & \cdots & G_0 \\ I & 0 & \cdots & 0 \\ \vdots & \cdots & \cdots & \vdots \\ 0 & 0 & I & 0 \end{pmatrix} \text{ and } B = -\begin{pmatrix} G_r & 0 & \cdots & 0 \\ 0 & I & \cdots & 0 \\ \vdots & \cdots & \cdots & \vdots \\ 0 & 0 & 0 & I \end{pmatrix},$$

where I is the identity matrix. The m-vector z can be formed using the formula:

$$z = (\lambda^{r-1}x, \lambda^{r-2}x, \ldots, x)^T$$

The calculations proceed as follows. At first, the pair of input matrices are reduced to the generalized Hessenberg form by orthogonal transformations. After the first step, A has been reduced to upper Hessenberg form, and B has been reduced to upper triangular form. The next step reduces A to upper quasi-triangular form while B is kept in upper triangular form.

On the third step, the generalized eigenvalues are extracted from the pair of matrices in the generalized upper quasi-triangular form.

If specified by the flag *matz*, the generalized eigenvectors are computed by the back substitution in the last step. The eigenvectors are only computed if the input parameter *matz* is set to any non-zero value.

Having returned from the function, the generalized eigenvalues can be computed as the ratios of the corresponding elements of the arrays *alfa* and *beta*:

$$\lambda_j = \frac{\mathrm{Re}(\alpha_j) + i\mathrm{Im}(\alpha_j)}{\beta_j}$$

The real and imaginary parts of the generalized eigenvectors are stored transposed (row-wise) in the array z. If $\mathrm{Im}(\alpha_i) = 0$, the i^{th} eigenvalue is real and the i^{th} row of z contains the corresponding eigenvector. If $\mathrm{Im}(\alpha_i) \neq 0$ the i^{th} eigenvalue would be complex and two cases arise:

If the imaginary part of an eigenvalue is positive, i.e. $\mathrm{Im}(\alpha_i) > 0$, then the eigenvalue is the first of a conjugate pair of eigenvalues. The i^{th} row of matrix z would contain the real part, while $(i+1)^{\mathrm{th}}$ row would contain the imaginary part of the corresponding eigenvector.

If the imaginary part of an eigenvalue is negative, i.e. $\mathrm{Im}(\alpha_i) < 0$, then the eigenvalue is the second of a complex pair and the $(i-1)^{\mathrm{th}}$ and i^{th} rows contain correspondingly the real and the imaginary parts of its eigenvector *conjugate*.

All the eigenvectors are normalized so that the absolute value of each vector's largest component is equal to 1.

Performance

The computational speed of function rgg is determined by the desired level of accuracy. The input parameter *eps1* to the function `qzit` controls that accuracy level within QZ algorithm. An user can specify the desired accuracy level by varying parameter *eps1* when invoking the functions of the QZ algorithm explicitly instead of using the driver function. For details see the Chapter **Real Generalized Eigenproblem**.

Notes

Arrays *a, b* and *z* are expected to be in the format produced by the allocation function `fsquare`.

The input matrices are modified by the function rgg.

Test

The test program is contained in `rggt.c` file. The example matrices are contained in `qz1.xpd, qz2.xpd` files.

References

C. B. Moler, G. W. Stuart, SIAM Journal of Numerical Analysis, Vol. 10, pp.241-256, 1973.

R. C. Ward, SIAM Journal of Numerical Analysis, Vol. 12, pp.835-853, 1975.

R. C. Ward, Technical Note NASA, TN D-7305 (1973).

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

2.2 Real General Eigenproblem

2.2.1 RG Eigenvalues and Eigenvectors

Name

srg — Driver for solving real general eigenproblem, single precision
drg — Driver for solving real general eigenproblem, double precision

Synopsis

```
#include <ceispack.h>
void srg (n, a, w, matz, z, iv, fv, ierr)
int n, matz;
fcomplex *w;
float **a, **z, *fv;
int *iv, *ierr;

void drg (n, a, w, matz, z, iv, fv, ierr)
int n, matz;
dcomplex *w;
double **a, **z, *fv;
int *iv, *ierr;
```

n	— order of input/output matrices
a	— square input matrix
w	— output array containing n eigenvalues
matz	— flag which causes computation of eigenvectors
z	— square output array containing n column eigenvectors
iv	— temporary storage array of length n
fv	— temporary storage array of length n
ierr	— address of output variable containing error completion code

Diagnostics

On output, the variable `*ierr` is set to 0 for normal return. If an error exit has been made, the eigenvalues would be correct for indices $[*ierr, n-1]$. In such a case, none of the eigenvectors have yet been determined.

Description

Function rg invokes the recommended sequence of functions from the eigensystem library C_EISPACK to find all eigenvalues and optionally, eigenvectors of a real general matrix.

Function rg calls the functions `balanc, elmhes, hqr` to determine eigenvalues only, or `balanc, elmhes, hqr2` and `balbak` to determine both the eigenvalues and eigenvectors.

The calculations proceed as follows. At first, the input matrix is balanced. It is then reduced to upper Hessenberg form by the stabilized elementary similarity transformations. Finally the upper Hessenberg matrix is reduced to upper quasi-triangular form by the real QR algorithm to determine eigenvalues. If specified, the eigenvectors are determined by the back substitution process.

Having returned from the function, the vector w contains the eigenvalues. Complex conjugate pairs of the eigenvalues appear consecutively. The eigenvalue having the positive imaginary part is stored first.

If specified by the flag *matz*, the array z contains unnormalized column eigenvectors. If the i^{th} eigenvalue is real, the i^{th} column of the matrix z contains its eigenvector. The eigenvectors are computed if the input flag *matz* is set to any non-zero value. When it is set to 0 only the eigenvalues are computed.

If the i^{th} eigenvalue is complex then two cases arise: if the imaginary part of an eigenvalue is positive, the i^{th} column of array z would contain the real part, while $(i+1)^{\text{th}}$ column of z contains the imaginary part of the corresponding eigenvector. The *conjugate* of this vector is the eigenvector for the *conjugate* eigenvalue.

Stabilized elementary transformations are generally faster than the orthogonal transformations, since they employ a fewer number of arithmetic operations. The disadvantage of elementary transformations is that they may possibly increase l_2 norm (Frobenius norm) of reduced matrix, thus decreasing the accuracy of the computed eigenvalues and eigenvectors, while orthogonal transformations keep 2-norm and condition numbers of the eigenvalues of the reduced matrix unchanged. It follows from the equations:

$$Q^T Q = QQ^T = I$$

where Q is orthogonal, and I is the identity matrix. The upper Hessenberg form H of the original matrix A is evaluated by the following formula:

$$H = Q^T AQ$$

and results in $\|Q^T AQ\| \equiv \|A\|$.

The next sequence can be applied, using orthogonal similarity transformations to reduce the original matrix to upper Hessenberg form and then determining the eigenvalues and optionally, eigenvectors by the QR algorithm: `balanc, orthes, hqr` to find the eigenvalues only, or `balanc, orthes, ortran, hqr2` and `balbak` to find both the eigenvalues and eigenvectors.

The next sequence can be used to find all the eigenvalues and some of the eigenvectors, corresponding to specified eigenvalues, using elementary similarity transformations to reduce the original matrix to upper Hessenberg form, QR algorithm to find the eigenvalues and the inverse iteration technique to determine the eigenvectors: `balanc, elmhes, hqr, invit, elmbak, balbak`, or the sequence using orthogonal similarity transformations to reduce the original matrix to upper Hessenberg form, QR algorithm to compute the eigenvalues and inverse iteration techniques to find the eigenvectors: `balanc, orthes, hqr, invit, ortbak, balbak`.

Performance

See the discussion of performance in the description of related functions `balanc, elmhes, hqr, hqr2, balbak`.

Notes

Arrays a and z are expected to be in the format produced by the allocation function `fsquare`.

The input matrix is modified by the function rg.

Test

The test program is contained in `rgt.c` file. The example matrices are contained in `rmatrix1.xpd, rmatrix2.xpd, rmatrix3.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

2.3 Symmetric Generalized Eigenproblem

2.3.1 RSG Generalized Eigenvalues and Eigenvectors

Name

srsg — Driver for solving generalized symmetric eigenproblem, single precision
drsg — Driver for solving generalized symmetric eigenproblem, double precision

Synopsis

```
#include <ceispack.h>
int srsg (n, a, b, w, matz, z, fv1, fv2, index)
int n, matz;
float **a, **b, *w, **z;
float *fv1, *fv2;
int *index;

int drsg (n, a, b, w, matz, z, fv1, fv2, index)
int n, matz;
double **a, **b, *w, **z;
double *fv1, *fv2;
int *index;
```

n	— order of matrix system
a	— symmetric input matrix
b	— positive definite symmetric input matrix
w	— output array containing n eigenvalues
matz	— flag which causes computation of eigenvectors
z	— output square array containing n transposed (row) eigenvectors
fv1	— temporary storage array of length n
fv2	— temporary storage array of length n
index	— address of output variable containing error completion code

Diagnostics

The function rsg returns:

- 0 for normal return.
- 1 if an error occurs in function `reduc`. It means that the second input matrix is not positive definite. The output variable `*index` is set to $(7n + 1)$.
- 2, 3 if an error occurs either in functions `tqlrat` or `tql2` respectively. The output variable `*index` is set to the index of the eigenvalue which has not been determined. The eigenvalues would be correct for indices $[0, *index - 1]$, but they may not be the smallest eigenvalues. The array z contains the transposed (row) eigenvectors associated with the computed eigenvalues.

Description

Function rsg invokes the recommended sequence of functions from the eigensystem library C_EISPACK to find the eigenvalues and eigenvectors of real symmetric generalized eigensystem:

$$Ax = \lambda Bx,$$

where matrix A is symmetric, matrix B is symmetric positive definite.

Function rsg calls the functions `reduc, tred1, tqlrat` to determine the eigenvalues only, or `reduc, tred2, tql2` and `rebak` to determine both the eigenvalues and eigenvectors.

The calculations proceed as follows. The solution requires matrix B to be positive definite and calculates its Cholesky factorization $B = L^T L$. The original eigensystem is reduced by orthogonal transformations to the standard symmetric eigenproblem:

$$A_1 y = \lambda y,$$

where

$$A_1 = L^{-1} A L^{-T},$$

and

$$y = L^T z$$

Next, the symmetric matrix is reduced to the symmetric tridiagonal form and its eigenvalues and if specified by the input flag *matz*, the eigenvectors are computed. At the last step, the eigenvectors are back transformed to the vectors of the original coordinate system.

Having returned from the function, vector w contains the eigenvalues stored in ascending order. If specified, array z contains the associated transposed (row) eigenvectors. The eigenvectors are normalized such that ${Z_i}^T B Z_i = I$. The eigenvectors are computed if the input parameter *matz* is set to any non-zero value. When it is set to zero, only the eigenvalues are computed.

The accuracy of the computed eigenvectors is commensurate to the magnitudes of the elements of matrix A_1. Thus when L^{-1}, and therefore A_1, having their elements of large magnitudes, the small eigenvalues may have large relative errors. This can happen when B is almost an indefinite matrix and is pointed out by the presence of eigenvalues which are much larger than the elements of the original matrices A and B.

The alternative approach to solving such a problem is to neglect the symmetry and utilize the functions for the real generalized eigenproblem. Although they are in general more computationally expensive and the eigenvalues obtained may have non-zero imaginary parts, the small eigenvalues are usually more precise than those computed by the symmetric sequence.

Notes

Arrays *a, b* and *z* are expected in the format produced by the allocation function `fsquare`.

Only the full upper triangles of the input matrices need be supplied. The strict upper triangle of matrix *a* and the full upper triangle of matrix *b* are unchanged.

Performance

See the discussion of performance in the description of related functions `reduc, tred1, tred2, tqlrat, tql2, rebak`.

Test

The test program is contained in `rsgt.c` file. The example matrices are contained in `rgensym.xpd` file.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

2.3.2 RSGAB Generalized Eigenvalues and Eigenvectors

Name

srsgab — Driver for solving generalized symmetric eigenproblem, single precision
drsgab — Driver for solving generalized symmetric eigenproblem, double precision

Synopsis

```
#include <ceispack.h>
int srsgab (n, a, b, w, matz, z, fv1, fv2, index)
int n, matz;
float **a, **b, *w, **z;
float *fv1, *fv2;
int *index;

int srsgab (n, a, b, w, matz, z, fv1, fv2, index)
int n, matz;
double **a, **b, *w, **z;
double *fv1, *fv2;
int *index;
```

n	— order of matrix system
a	— symmetric input matrix
b	— positive definite symmetric input matrix
w	— output array containing n eigenvalues
matz	— flag which causes computation of eigenvectors
z	— square output array containing n row eigenvectors
fv1	— temporary storage array of length n
fv2	— temporary storage array of length n
index	— address of output variable containing error completion code

Diagnostics

The function rsgab returns:

- 0 for normal return.
- 1 if an error occurs in function *reduc2*. This means that the second input matrix is not positive definite. The output variable `*index` is set to $(7n + 1)$.
- 2, 3 if an error occurs either in the functions *tqlrat* or *tql2* respectively. The output variable `*index` is set to index of the eigenvalue which has not been determined. The eigenvalues would be correct for the indices $[0, *index - 1]$, but they may not be the smallest eigenvalues. The array z contains the transposed (row) eigenvectors associated with the computed eigenvalues.

Description

Function rsgab invokes the recommended sequence of functions from the eigensystem library C_EISPACK to find the eigenvalues and eigenvectors of the real symmetric generalized eigenproblem:

$$ABx = \lambda x,$$

where matrix A is symmetric and matrix B is symmetric positive definite.

Function rsgab calls the functions `reduc2, tred1, tqlrat` to determine eigenvalues only, or `reduc2, tred2, tql2` and `rebak` to determine both the eigenvalues and eigenvectors.

The calculations proceed as follows. The original eigensystem is reduced to the standard symmetric eigenproblem. Since the matrix B is positive definite, its Cholesky factorization $B = LL^T$ exists and allows the original problem to be redefined as:

$$ALL^T z = \lambda z$$

and which is then transformed to the standard symmetric problem:

$$A_1 y = \lambda y,$$

where

$$A_1 = L^T AL$$

and

$$L^T z = y$$

Next, the symmetric matrix is reduced to the symmetric tridiagonal form and its eigenvalues and, if specified by the input flag *matz*, eigenvectors are computed. At the last step, the eigenvectors are back transformed to the vectors of the original coordinate system.

Having returned from the function, vector w contains the eigenvalues stored in ascending order. If specified, array z contains the associated transposed (row) eigenvectors. The eigenvectors are normalized so that ${Z_i}^T BZ_i = I$. The eigenvectors are computed if the input parameter *matz* is set to any non-zero value. When it is set to zero, only the eigenvalues are computed.

Notes

Arrays *a, b* and z are expected in the format produced by the allocation function `fsquare`.

Only the full upper triangles of the input matrices need be supplied. The strict upper triangle of matrix a and the full upper triangle of matrix b are unchanged.

Performance

See the discussion of performance in the description of related functions `reduc2, tred1, tred2, tqlrat, tql2, rebak`.

Test

The test program is contained in `rsgabt.c` file. The example matrices are contained in `rgensym.xpd` file.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

2.3.3 RSGBA Generalized Eigenvalues and Eigenvectors

Name

srsgba — Driver for solving generalized symmetric eigenproblem, single precision
drsgba — Driver for solving generalized symmetric eigenproblem, double precision

Synopsis

```
#include <ceispack.h>
int srsgba (n, a, b, w, matz, z, fv1, fv2, index)
int n, matz;
float **a, **b, *w, **z;
float *fv1, *fv2;
int *index;

int drsgba (n, a, b, w, matz, z, fv1, fv2, index)
int n, matz;
double **a, **b, *w, **z;
double *fv1, *fv2;
int *index;
```

n	— order of matrix system
a	— symmetric input matrix
b	— positive definite symmetric input matrix
w	— output array containing n eigenvalues
matz	— flag which causes computation of eigenvectors
z	— square output array containing n transposed (row) eigenvectors
fv1	— temporary storage array of length n
fv2	— temporary storage array of length n
index	— address of output variable containing error completion code

Diagnostics

The function rsgba returns:

- 0 for normal return.

- 1 if an error occurs in function `reduc2`. This means the second input matrix is not positive definite. The output variable *index* is set to $(7n + 1)$.

- 2, 3 if an error occurs either in the function `tqlrat` or `tql2` respectively. The output variable `*index` is set to index of the eigenvalue which has not been determined. The eigenvalues would be correct for the indices $[0, *index - 1]$, but they may not be the smallest eigenvalues. The array z contains the eigenvectors associated with the computed eigenvalues.

Description

Function rsgba invokes the recommended sequence of functions from the eigensystem library C_EISPACK to find the eigenvalues and eigenvectors for the real symmetric generalized eigenproblem:

$$BAx = \lambda x,$$

where A is symmetric and B is symmetric positive definite.

Function rsgba invokes the sequence of functions `reduc2, tred1, tqlrat` to determine eigenvalues only, or `reduc2, tred2, tql2` and `rebak` to determine both the eigenvalues and eigenvectors.

The calculations proceed as follows. The original eigensystem is reduced to the standard symmetric eigenproblem. Since matrix B is positive definite, its Cholesky factorization $B = LL^T$ exists and allows the original problem to be redefined as:

$$LL^T Az = \lambda z$$

and then transformed to the standard symmetric problem:

$$A_1 y = \lambda y,$$

where

$$A_1 = L^T AL$$

and

$$L^{-1} z = y$$

Next, the symmetric matrix is reduced to the symmetric tridiagonal form and its eigenvalues and, if specified by the input flag *matz*, eigenvectors are computed. At the last step, the eigenvectors are back transformed to the vectors of the original coordinate system.

Having returned from the function, the vector w contains the eigenvalues stored in ascending order. If specified, the array z contains the associated transposed (row) eigenvectors. The eigenvectors are normalized so that ${Z_i}^T B^{-1} Z_i = I$. The eigenvectors are computed if the input parameter *matz* is set to any non-zero value. When it is set to zero, only the eigenvalues are computed.

Notes

Arrays *a, b* and *z* are expected in the format produced by the allocation function `fsquare`.

Only the full upper triangles of the input matrices need be supplied. The strict upper triangle of matrix a and the full upper triangle of matrix b are unchanged.

Performance

See the discussion of performance in the description of related functions `reduc2, tred1, tred2, tqlrat, tql2, rebak`.

Test

The test program is contained in `rsgbat.c` file. The example matrices are contained in `rgensym.xpd` file.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

2.4 Symmetric Eigenproblem

2.4.1 RS Eigenvalues and Eigenvectors

Name

srs — Driver for solving symmetric eigenproblem, single precision
drs — Driver for solving symmetric eigenproblem, double precision

Synopsis

```
#include <ceispack.h>
int srs (n, a, w, matz, z, fv1, fv2, index)
int n, matz;
float **a, *w, **z, *fv1, *fv2;
int *index;

int drs (n, a, w, matz, z, fv1, fv2, index)
int n, matz;
double **a, *w, **z, *fv1, *fv2;
int *index;
```

n	— order of input/output matrices
a	— symmetric input matrix
w	— output array containing n eigenvalues
matz	— flag which causes computation of eigenvectors
z	— output square array containing n transposed (row) eigenvectors
fv1	— temporary storage array of length n
fv2	— temporary storage array of length n
index	— address of output variable containing error completion code

Diagnostics

Function rs returns 0 for normal return or 1 if an error exit has occured. If an error occured, the output variable `*index` is set to index of the eigenvalue which has not been determined. The eigenvalues would be correct for indices $[0, *index - 1]$, but they may not be the smallest eigenvalues. The array z contains transposed (row) eigenvectors associated with the computed eigenvalues.

Description

Function rs invokes the recommended sequence of functions from the eigensystem library C_EISPACK to find the eigenvalues and, optionally eigenvectors of a real symmetric matrix.

Function rs calls the functions `tred1, tqlrat` to determine the eigenvalues only, or `tred2, tql2` to determine both the eigenvalues and eigenvectors.

The calculations proceed as follows. The original matrix is reduced to the symmetric tridiagonal form and its eigenvalues are computed. If specified by the flag *matz*, the transformations are accumulated in the array *z* and the computed eigenvectors are back transformed to the original coordiante system.

Having returned from the function, vector *w* contains the eigenvalues stored in ascending order. If specified, array *z* contains the transposed (row) orthonormal eigenvectors associated with the computed eigenvalues. The eigenvectors are computed if the input parameter *matz* is set to any non-zero value. When it is set to zero, only the eigenvalues are computed.

Notes

Matrix *a* is expected to be in the format produced either by allocation functions `fmatrix`, `fsquare`, `trngl_fmatrix`, `fsym`. If `trngl_fmatrix` is being used for the allocation, the *lower triangle* matrix must be specified. Only the lower triangle of the input matrix need be supplied.

Array *z* is expected to be in the format produced by the allocation function `fsquare`.

If only the eigenvalues are to be computed, the input matrix is modified by the function rs. If computation of the eigenvectors is specified arrays *a* and *z* may coincide (in this case array *a* must be the *full* array). If they are distinct, matrix *a* is preserved by the function.

Performance

See the discussion of performance in the description of related functions `tred1, tred2, tqlrat, tql2`.

Test

The test program is contained in `rstest.c` file. The example matrices are contained in `rsymmet1.xpd, rsymmet2.xpd, rsymmet3.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

2.4.2 RSM All Eigenvalues and Some Eigenvectors

Name

srsm — Driver for solving symmetric eigenproblem, single precision.
drsm — Driver for solving symmetric eigenproblem, double precision.

Synopsis

```
#include <ceispack.h>
int srsm (n, a, w, m, z, fv, iwork, index)
int n, m;
float **a, *w, **z, *fv;
int *iwork, *index;

int drsm (n, a, w, m, z, fv, iwork, index)
int n, m;
double **a, *w, **z, *fv;
int *iwork, *index;
```

n — order of input matrices
a — symmetric input matrix
w — output array containing n eigenvalues
m — number of eigenvalues to be determined
z — rectangular output array containing m transposed (row) eigenvectors
fv — temporary storage array of length $8n$
iwork — temporary storage integer array of length n
index — output variable containing error completion code

Diagnostics

The function rsm returns:

- 0 for normal return.

- 1, 2 if an error occurs in functions `tqlrat` or `imtqlv` respectively. The output variable `index` is set to the index of the eigenvalue which has not been determined. The eigenvalues would be correct for the indices $[0, *index - 1]$, but may not be the smallest eigenvalues.

- 3 if an error occurs in function `tinvit`. The variable `*index` indicates the negative of the last index of the eigenvalue that corresponds to the eigenvector of which fails to converge in 5 iterations. The rows of matrix z corresponding to such an eigenvalues are set to zero.

Description

Function rsm invokes the recommended sequence of functions from the eigensystem library C_EISPACK to find all eigenvalues and some eigenvectors of a real symmetric matrix.

Function rsm calls the functions `tred1, tqlrat` to determine all eigenvalues, or `tred1, imtqlv, tinvit` and `trbak1` to determine all eigenvalues and those eigenvectors, corresponding to the first m eigenvalues.

The calculations proceed as follows. The original matrix is reduced to the symmetric tridiagonal form and its eigenvalues are computed. If specified by the input variable m, the eigenvectors of the symmetric tridiagonal matrix are computed by the inverse iteration technique and further they are back transformed to those of the original coordinate system.

Having returned from the function, vector w contains the eigenvalues stored in ascending order. If specified, array z contains the transposed (row) orthonormal eigenvectors associated with the computed eigenvalues. The eigenvectors are computed if the input parameter m is greater than 0. When it is set to zero, only the eigenvalues are computed.

To find some of the eigenvalues of a real symmetric matrix, one can use another sequence of functions: `tred1` to reduce original matrix to symmetric tridiagonal form and then compute the eigenvalues by either functions `ratqr` to determine an eigenvalue of minimum magnitude, `bisect` to determine eigenvalues which lie in the specified interval or `tridib` to compute eigenvalues having specified indices.

Notes

Matrix a is expected to be in the format produced either by allocation functions `fmatrix`, `fsquare`, `trngl_fmatrix`, `fsym`. If `trngl_fmatrix` is being used for the allocation, the *lower* triangle must be specified.

Only the lower triangle of the original matrix need be supplied. The input matrix is modified by the function rsm.

Rectangular $m \times n$ matrix z is expected to be in the format produced by the allocation function `fmatrix`.

Performance

See the discussion of performance in the description of related functions `tred1, tqlrat, imtqlv, tinvit, trbak1`.

Test

The test program is contained in `rsmt.c` file. The example matrices are contained in `rsymmet1.xpd, rsymmet2.xpd, rsymmet3.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

2.5 Symmetric Band Eigenproblem

2.5.1 RSB Eigenvalues and Eigenvectors

Name

srsb — Driver for solving band symmetric eigenproblem, single precision.
drsb — Driver for solving band symmetric eigenproblem, double precision.

Synopsis

```
#include <ceispack.h>
int srsb (n, mb, a, w, matz, z, fv1, fv2, index)
int n, mb, matz;
float **a, **z, *w;
float *fv1, *fv2;
int *index;

int drsb (n, mb, a, w, matz, z, fv1, fv2, index)
int n, mb, matz;
double **a, **z, *w;
double *fv1, *fv2;
int *index;
```

n	— order of input/output matrices
mb	— number of sub/super diagonals
a	— band symmetric input matrix
w	— output array containing n eigenvalues
matz	— flag which causes computation of eigenvectors
z	— square output array containing n row eigenvectors
fv1	— temporary storage array of length n
fv2	— temporary storage array of length n
index	— address of output variable containing error completion code

Diagnostics

The function rsb returns:

- 0 for normal return.
- 1 if the computation of an eigenvalue fails. The output variable `*index` is set to index of the eigenvalue which has not been determined. The eigenvalues would be correct for the indices $[0, *index - 1]$, but may not be the smallest eigenvalues. The array z contains transposed (row) eigenvectors associated with the computed eigenvalues.
- $12n$ if the number of subdiagonals is $mb < 0$ or $mb > n$.

Description

Function rsb invokes the recommended sequence of functions from the eigensystem library C_EISPACK to find the eigenvalues and optionally, eigenvectors of a real band symmetric matrix.

Function rsb calls the functions `bandr`, and `tqlrat` to determine eigenvalues only, or `bandr`, and `tql2` to determine both the eigenvalues and eigenvectors.

The calculations proceed as follows. The original matrix is reduced to the symmetric tridiagonal form and its eigenvalues are computed. If specified by the input flag *matz* the transformations are accumulated in the array z and the eigenvectors are back transformed to the original coordinate system.

Having returned from the function, vector w contains the eigenvalues stored in ascending order. If specified, array z contains the associated transposed (row) orthonormal eigenvectors. The eigenvectors are computed if the input parameter *matz* is set to any non-zero value. When it is set to zero, only the eigenvalues are computed.

Function rsb invokes one of the possible sequences for the computation eigenvalues and eigenvectors of real band symmetric matrix. The function `bqr` can be applied to find the eigenvalue of smallest magnitude, or the functions `bqr`, and `bandv` to determine the eigenvalues and corresponding eigenvectors.

Other possible sequences can include the functions that apply the implicit QL algorithm to find the eigenvalues of *reduced* symmetric tridiagonal matrix.

Notes

Matrix a is expected to be in the format produced by the allocation functions `trngl_band_fmatrix` or `fbandsym`. If the first function is being used, the *lower* part of band must be specified.

Only the lower part (half band width) of the input matrix need be supplied. The input matrix is modified by the function rsb.

The total number of diagonals in the input matrix is equal to $(2m_b + 1)$.

Array z is expected to be in the format produced by the allocation function `fsquare`.

Performance

See the discussion of performance in the description of related functions `bandr, tqlrat, tql2`.

Test

The test program is contained in `rsbt.c` file. The example matrices are contained in `bsm01.xpd, bsm02.xpd` and `bsm03.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

2.6 Real Special Tridiagonal Eigenproblem

2.6.1 RT Eigenvalues and Eigenvectors

Name

srt — Driver for solving tridiagonal eigenproblem, single precision
drt — Driver for solving tridiagonal eigenproblem, double precision

Synopsis

```
#include <ceispack.h>
int srt (n, a, w, matz, z, fv, index)
int n, matz;
float **a, *w, **z, *fv;
int *index;

int drt (n, a, w, matz, z, fv, index)
int n, matz;
double **a, *w, **z, *fv;
int *index;
```

n	— order of input/output matrices
a	— tridiagonal input matrix
w	— output array containing n eigenvalues
matz	— flag which causes computation of eigenvectors
z	— square output array containing n transposed (row) eigenvectors
fv	— temporary storage array of length n
index	— output variable containing error completion code

Diagnostics

The function rt returns:

- 0 for normal return.

- 1 if an error occurs in the function `figi`. The output variable `*index` is set to $(n + i)$ if the product $a_{i,i-1}a_{i-1,i} < 0$, or to $-(3n + i)$ if the product $a_{i,i-1}a_{i-1,i} = 0$ with one factor not 0. In such a case the eigenvectors of the symmetric tridiagonal matrix are not simply related to those of original matrix and should not be computed.

- 2 if an error occurs in the function `figi2`. The output variable `*index` is set to $(n + i)$ if the product $a_{i,i-1}a_{i-1,i} < 0$, or to $(2n + i)$ if the product $a_{i,i-1}a_{i-1,i} = 0$ with one factor is not 0. In such a case none of the eigenvectors have yet been computed.

- 3, 4 if an error occurs in the functions `imtql1` or `imtql2` respectively. The output variable `*index` is set to index of the eigenvalue which has not been determined. The eigenvalues would be correct for indices $[0, *index - 1]$. If the failure occurs in function `imtql1` these eigenvalues are ordered but they may not be the smallest ones. If the failure occurs in function `imtql2` these eigenvalues are unordered. The array z contains transposed (row) eigenvectors associated with the computed eigenvalues.

Description

Function rt invokes the recommended sequence of functions from the eigensystem library C_EISPACK to find the eigenvalues and, optionally eigenvectors of a special real tridiagonal matrix.

Function rt calls the functions `figi`, and `imtql1` to determine eigenvalues only, or `figi2` and `imtql2` to determine both the eigenvalues and eigenvectors.

The calculations proceed as follows. The original matrix is reduced to the symmetric tridiagonal form and its eigenvalues are computed. If specified by the input flag *matz*, the diagonal similarity transformations are accumulated and later used to back transform the eigenvectors to those of the original system.

Having returned from the function, the vector w contains n eigenvalues stored in ascending order. If specified by the input flag *matz*, array z contains the associated transposed (row) orthonormal eigenvectors. The eigenvectors are computed if the input parameter *matz* is set to any non-zero value. When it is set to zero, only the eigenvalues are computed.

To determine some of the eigenvalues of a special tridiagonal matrix one can use the function `figi` to reduce the original matrix to symmetric tridiagonal form and then compute the eigenvalues by either functions `ratqr`, `bisect` or `tridib`.

To compute some eigenvalues and corresponding eigenvectors one can call the sequence `figi, bisect, tinvit, bakvec` or `figi, tsturm, bakvec`.

Notes

Matrix a is expected to be in the format produced either by the allocation functions `band_fmatrix, fband` or `f3diag`.

The input matrix is unchanged by the function rt.

Array z is expected to be in the format produced by the allocation function `fsquare`.

Performance

See the discussion of performance in the description of related functions `figi, figi2, imtql1, imtql2`.

Test

The test program is contained in `rtt.c` file. The example matrices are contained in `trispc1.xpd, trispc2.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

2.7 Symmetric Tridiagonal Eigenproblem

2.7.1 RST Eigenvalues and Eigenvectors

Name

srst — Driver for solving symmetric tridiagonal eigenproblem, single precision
drst — Driver for solving symmetric tridiagonal eigenproblem, double precision

Synopsis

```
#include <ceispack.h>
int srst (n, d, e, matz, z, index)
int n, matz;
float *d, *e, **z;
int *index;

int drst (n, d, e, matz, z, index)
int n, matz;
double *d, *e, **z;
int *index;
```

n	— order of input/output matrices
t	— input symmetric tridiagonal matrix
d	— length n input vector of diagonal elements/output n ascending eigenvalues
e	— length n input vector containing subdiagonal elements of input matrix
matz	— flag which causes computation of eigenvectors
z	— output square array containing n transposed (row) eigenvectors
index	— address of output variable containing error completion code

Diagnostics

The function returns 0 for normal return, or 1 if an error occurs in the functions `imtql1` or `imtql2`. The output variable `*index` is set to the index of the eigenvalue which has not been determined. The eigenvalues would be correct for the indices $[0, *index - 1]$. If the failure occurs in function `imtql1`, these eigenvalues are ordered but may not be the smallest ones. If the failure occurs in function `imtql2` these eigenvalues are unordered. The array z contains transposed (row) eigenvectors associated with the computed eigenvalues.

Description

Function rst invokes the recommended sequence of functions from the eigensystem library C_EISPACK to find all eigenvalues and, optionally eigenvectors of a real symmetric tridiagonal matrix.

Function rst invokes the function `imtql1` to find the eigenvalues only, or function `imtql2` to determine both the eigenvalues and eigenvectors. The eigenvalues are computed by the implicit QL algorithm.

The input vectors d and e contain diagonal and subdiagonal elements of the original matrix. The subdiagonal elements should be stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array is arbitrary.

Having returned from the function, the vector d contains n eigenvalues stored in ascending order. If specified by the input flag *matz*, array z contains the associated transposed (row) orthonormal eigenvectors. The eigenvectors are computed if the input parameter *matz* is set to any non-zero value. When it is set to zero, only the eigenvalues are computed.

The function rst invokes one of the possible sequences for the computation of the eigenvalues and eigenvectors of a real symmetric tridiagonal matrix. The functions `imtqlv,` `tql1` or `tqlrat` can be used to find all of the eigenvalues of a symmetric tridiagonal matrix.

The functions `bisect, ratqr` and `tridib` can be used to find some of the eigenvalues, namely those which are lie within a specified interval, the algebraically smallest or largest eigenvalues, and the eigenvalues which are lie within specified boundary indices respectively.

The functions `tql2` and `imtql2` find all the eigenvalues and eigenvectors. The function `tsturm` finds those eigenvalues which lie within a specified interval and the eigenvalues corresponding to them.

The functions `bisect, imtqlv, ratqr` and `tridib` associate the submatrix indices with the eigenvalues computed.

The function `tinvit` finds the eigenvectors which correspond to the specified eigenvalues, and is called after that the eigenvalues required have been determined by the functions `bisect, imtqlv, ratqr` or `tridib`.

The function `tridib` is generally faster and more accurate than the function `ratqr` if the eigenvalues are clustered. If more than $n/4$ of the eigenvalues are to be found, the functions `tql1, imtql1` or `tqlrat` are generally faster than either `tridib` and `bisect`.

Generally the functions `tql1, tql2` and `tqlrat`, which are introduce QL algorithm with explicit shift does not performed well on the matrices which have not have subsequent row norms increasing from top to bottom, as opposite to the matrix:

$$T = \begin{cases} t_{ii} = 10^{4i} & \text{for } i \in [1, n] \\ t_{i,i+1} = t_{i+1,i} = i & \text{for } i \in [1, n-1] \\ t_{ij} = 0 & \text{for the rest of the matrix} \end{cases}$$

If the eigenvalues of small magnitude are needed to have the accuracy relative to themselves, not to the norm of the matrix, the functions based on the explicit QL

algorithm are almost certainly fail to determine the eigenvalues with such a high precision. Instead, the functions `imtql1, imtql2, imtqlv` that utilize the implicit QL algorithm are not sensitive so crucially to such a special structure of a matrix and may compute the small eigenvalues with the accuracy still relative to themselves.

Notes

Array z is expected to be in the format produced by the allocation function `fsquare`.

Performance

See the discussion of performance in the description of related functions `imtql1, imtql2`.

Test

The test program is contained in `rstt.c` file. The example matrices are contained in `trisym1.xpd, trisym2.xpd, trisym3.xpd` and `trisym4.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

2.8 Complex General Eigenproblem

2.8.1 CG Eigenvalues and Eigenvectors

Name

ccg — Driver for solving complex general eigenproblem, single precision
zcg — Driver for solving complex general eigenproblem, double precision

Synopsis

```
#include <ceispack.h>
void ccg (n, a, w, matz, z, fv1, cv2, cv3, ierr)
int n;
int matz;
fcomplex **a, **z, *w;
float *fv1;
fcomplex *cv2, *cv3;
int *ierr;

void zcg (n, a, w, matz, z, fv1, cv2, cv3, ierr)
int n;
int matz;
dcomplex **a, **z, *w;
double *fv1;
dcomplex *cv2, *cv3;
int *ierr;
```

n	— order of input/output matrices
a	— square input matrix
w	— output array containing n eigenvalues
matz	— flag which causes computation of eigenvectors
z	— output square array containing n column eigenvectors
fv1	— temporary storage array of length n
cv2	— temporary storage array of length n
cv3	— temporary storage array of length n
ierr	— address of output variable containing error completion code

Diagnostics

On output the variable `*ierr` is set to zero for normal return. If an error exit has been made, the eigenvalues would be correct for the indices $[*ierr, n-1]$. In such a case, none of the eigenvectors have yet been determined.

Description

Function cg invokes the recommended sequence of functions from the the eigensystem library C_EISPACK to find all eigenvalues and optionally, eigenvectors of a complex general matrix.

The function calls the functions `cbal, corth,` and `comqr` to determine the eigenvalues only, or `cbal, corth, comqr2` and `cbabk2` to determine both the eigenvalues and eigenvectors.

The calculations proceed as follows. At first, the input matrix is balanced. Next, it is reduced to upper Hessenberg form by the unitary similarity transformations and finally the upper Hessenberg matrix is reduced to upper triangular form by the complex QR algorithm to determine eigenvalues.

If specified by the flag `matz`, the eigenvectors are determined by the back substitution process. The eigenvectors are computed if the input parameter `matz` is set to any non-zero value. When it is set to zero, only the eigenvalues are computed.

Having returned from the function, vector w contains n eigenvalues. If specified, the array z contains n column eigenvectors. They are unnormalized.

Function cg invokes one of the possible sequences for the determination of the eigenvalues and eigenvectors of complex general matrix. Another sequence can be applied, using stabilized elementary similarity transformation to reduce original matrix to upper Hessenberg form and determine the eigenvalues and eigenvectors by LR algorithm: `cbal, comhes,` and `comlr` to find the eigenvalues only, or `cbal, comhes, comlr2` and `cbabk2` to find both the eigenvalues and eigenvectors.

Stabilized elementary transformations are generally faster than unitary transformations, since they employ fewer arithmetic operations. The disadvantage of elementary transformations is that they possibly increase the l_2 norm (Frobenius norm) of the reduced matrix, thus decreasing the accuracy of the computed eigenvalues and eigenvectors, while unitary transformations keep the 2-norm and condition numbers of the eigenvalues of reduced matrix unchanged.

The next sequences can be used to find all the eigenvalues and some eigenvectors, corresponding to the specified eigenvalues, using elementary similarity transformations to reduce the original matrix to upper Hessenberg form, the LR algorithm to find the eigenvalues and the inverse iteration technique to find the eigenvectors: `cbal, comhes, comlr, cinvit, combak, cbabk2`, or the sequence utilizing unitary similarity transformations to reduce the original matrix to upper Hessenberg form, the QR algorithm to find the eigenvalues and the inverse iteration technique to find the eigenvectors: `cbal, corth, comqr, cinvit, cortb, cbabk2`.

Performance

See the discussion of performance in the description of related functions `cbal, corth, comqr, comqr2, cbabk2`.

Notes

Arrays a and z are expected to be in the format produced by the allocation function `csquare`.

The input matrix is modified by the function cg.

Test

The test program is contained in `cgt.c` file, the example matrices are contained in `cmatrix1.xpd, cmatrix2.xpd, cmatrix3.xpd, cmatrix4.xpd` and `cmatrix5.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

2.9 Hermitian Eigenproblem

2.9.1 CH Eigenvalues and Eigenvectors

Name

cch — Driver for solving Hermitian eigenproblem, single precision.
zch — Driver for solving Hermitian eigenproblem, double precision.

Synopsis

```
#include <ceispack.h>
int cch (n, a, w, matz, zr, zi, fm1, fv1, fv2, index)
int n, matz;
fcomplex **a;
float *w, **zr, **zi;
float **fm1, *fv1, *fv2;
int *index;

int zch (n, a, w, matz, zr, zi, fm1, fv1, fv2, index)
int n, matz;
dcomplex **a;
double *w, **zr, **zi;
double **fm1, *fv1, *fv2;
int *index;
```

n	— order of input/output matrices
a	— input Hermitian matrix
w	— output array containing n eigenvalues
matz	— flag which causes computation of eigenvectors
zr	— square output array containing real parts of n transposed (row) eigenvectors
zi	— square output array containing imaginary parts of n transposed (row) eigenvectors
fm1	— temporary storage rectangular $2 \times n$ array
fv1	— temporary storage array of length n
fv2	— temporary storage array of length n
index	— address of output variable containing error completion code

Diagnostics

Function ch returns 0 for normal return or 1 if an error exit has been made. The output variable `*index` is set to the index of the eigenvalue which has not been determined. The eigenvalues having their indices in $[0, *index - 1]$ would be correct, but may not be the smallest eigenvalues. If specified, the arrays *zr, zi* contains the transposed (row) eigenvectors associated with the computed eigenvalues.

Description

Function ch invokes the recommended sequence of functions from the eigensystem library C_EISPACK to find the eigenvalues and, optionally eigenvectors of Hermitian matrix.

Function ch calls the functions `htridi`, and `tqlrat` to determine eigenvalues only, or `htridi, tql2`, and `htribk` to determine both the eigenvalues and eigenvectors.

The calculations proceed as follows. At first the original matrix is reduced to the symmetric tridiagonal form and its eigenvalues are computed. If specified by the flag *matz*, the eigenvectors of the symmetric tridiagonal matrix are computed and back transformed to those of the original system. The eigenvectors are computed if the input parameter *matz* is set to any non-zero value. When it is set to zero, only the eigenvalues are computed.

Having returned from the function, vector w contains n eigenvalues stored in ascending order. If specified, arrays *zr, zi* contain the real and imaginary parts respectively, of the associated orthonormal transposed (row) eigenvectors.

The variations of the recommended sequence for determining eigenvalues and eigenvectors of Hermitian matrix would be the substitution of the implicit QL algorithm and bisection technique for the explicit and rational QL algorithms. See the description of function `rst` for the discussion of the properties of implicit and explicit QL algorithms and the description of related functions as well.

If the original Hermitian matrix is stored in a single real array the reduction function `htrid3` need to be substituted for the function `htridi` and the function `htrib3` need to be substituted for the function `htribk`. See the description of the functions `htrid3` and `htrib3`.

Notes

Array a is expected to be in the format produced either by the allocation functions `cmatrix`, `csquare`, `trngl_cmatrix`, `chermit`. If `trngl_cmatrix` is being used for the allocation, the *lower* triangle must be specified.

Only the *lower* triangle of the original matrix need be supplied. The function modified it on return.

Arrays *zr, zi* are expected to be in the format produced by the allocation function `fsquare`.

Rectangular 2-rows, n-columns array *fm1* is expected to be in the format produced by the allocation function `fmatrix`.

Performance

See the discussion of performance in the description of related functions `htridi, tqlrat, tql2, htribk`.

Test

The test program is contained in `cht.c` file, the example matrices are contained in `hermiti0.xpd` and `hermiti1.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

Chapter 3

Real Generalized Eigenproblem

3.1 QZ Algorithm

3.1.1 QZHES Reduction to Hessenberg Form

Name

sqzhes — Reduction of pair of matrices to generalized Hessenberg form, single precision
dqzhes — Reduction of pair of matrices to generalized Hessenberg form, double precision

Synopsis

```
#include <ceispack.h>
void sqzhes (n, a, b, matz, z, rv)
int n;
int matz;
float **a, **b, **z;
float *rv;

void dqzhes (n, a, b, matz, z, rv)
int n;
int matz;
double **a, **b, **z;
double *rv;
```

n — order of input/output matrices
a — first square input/output matrix
b — second square input/output matrix
matz — input flag which causes accumulation of transformations
z — output transposed orthogonal transformation matrix
rv — temporary storage array of length n

Description

Function qzhes performs the first phase of the QZ algorithm for solving the generalized matrix eigenvalue problem:

$$Ax = \lambda Bx$$

Function qzhes accepts a pair of a real general matrices and reduces one of them to upper Hessenberg form and the other to upper triangular form using Householder transformations and rotations:

$$(A, B) \to (A', B')$$

where the matrix A' is in upper Hessenberg form, matrix B' is in upper triangular form.

At first, the matrix B is reduced to an upper triangular form using a sequence of $(n-1)$ Householder reflections that are simultaneously applied to matrix A:

$$\tilde{B} = HB \text{ and } \tilde{A} = HA,$$

where matrix H is the product of Householder transformations. Then the matrix $\tilde{A}$ is reduced to an upper Hessenberg form by a sequence of $(n-1)(n-2)/2$ left-hand rotations R. Right-hand rotations R' are applied to both the matrices $\tilde{A}$ and $\tilde{B}$ to preserve upper triangular form of the B:

$$A' = R\tilde{A}R' \text{ and } B' = R\tilde{B}R'$$

A simultaneous reduction is possible if the entries of $\tilde{A}$ are eliminated column-wise from left to right and from bottom to top in a column. This process is illustrated for a matrix of order 6:

$$\begin{pmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ (4) & \times & \times & \times & \times & \times \\ (3) & (7) & \times & \times & \times & \times \\ (2) & (6) & (9) & \times & \times & \times \\ (1) & (5) & (8) & (10) & \times & \times \end{pmatrix}$$

Each rotation which eliminates an element of the $\tilde{A}$ when applied to the $\tilde{B}$ sets its subdiagonal entry to a non-zero value. Then this element is annihilated by the right-hand rotation which does not affect the already eliminated entries of the matrix $\tilde{A}$. Total number of right-hand rotations is the same as of left-hand ones and equal to $(n-1)(n-2)/2$.

If specified by the input flag *matz*, the function accumulates right-hand transformations used in the reduction process for later use for eigenvectors computation. They are stored transposed into the array *z*. The transformations are accumulated when *matz* is set to a non-zero value. When *matz* is set to 0, the output matrix *z* is *not referenced.*

Having returned from the function the matrix *a* is in upper Hessenberg form, and the matrix *b* is in upper triangular form.

Performance

The reduction process is numerically stable. It requires about $\frac{17}{3}n^3$ multiplications, $\frac{17}{3}n^3$ additions and n^2 square roots. If accumulation of the transformation is specified, it requires additionally $\frac{3}{2}n^3$ multiplications and $\frac{3}{2}n^3$ additions.

Notes

Arrays a, b and z are expected to be in the format produced by the allocation function `fsquare`.

Test

The test program is contained in `rggt.c` file. The example matrices are contained in `qz1.xpd, qz2.xpd` files.

References

C. B. Moler, G. W. Stuart, SIAM Journal of Numerical Analysis, Vol. 10, pp.241-256, 1973.

R. C. Ward, Technical Note NASA, TN D-7305 (1973).

R. C. Ward, SIAM Journal of Numerical Analysis, Vol. 12, pp.835-853, 1975.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

3.1.2 QZIT Reduction to Generalized Triangular Form

Name

sqzit — Combination shift QZ iterations, single precision
dqzit — Combination shift QZ iterations, double precision

Synopsis

```
#include <ceispack.h>
void sqzit (n, a, b, eps1, matz, z, ierr)
int n;
float eps1;
float **a, **b, **z;
int matz, *ierr;

void dqzit (n, a, b, eps1, matz, z, ierr)
int n;
double eps1;
double **a, **b, **z;
int matz, *ierr;
```

n — order of input/output matrices
eps1 — input variable containing absolute error tolerance
a — input upper Hessenberg/output quasi-triangular matrix
b — input/output matrix in upper triangular form
matz — flag which causes accumulation of transformations
z — input/output transposed orthogonal transformation matrix
ierr — address of output variable containing error completion code

Diagnostics

On output the variable `*ierr` is set to 0 for normal return. If the limit of `MAXITER` iterations is exceeded while the i^{th} diagonal block of order 1 or 2 is being determined, the error flag `*ierr` is set to $(i+1)$. The principal minors in rows and columns between indices $[*ierr, n-1]$ would be appropriate for eigenvalues computation by `qzval` function. No eigenvectors can be determined.

Description

Function qzit performs the second phase of the QZ algorithm for solving the generalized matrix eigenvalue problem:

$$Ax = \lambda Bx$$

Function qzit accepts a pair of real matrices, one of them in upper Hessenberg form and the other in upper triangular form. Then qzit reduces the Hessenberg matrix to a quasi-triangular form using orthogonal transformations while keeping the triangular form of the second matrix.

The QR algorithm with a shift of origin is applied effectively to the matrix in upper Hessenberg form:

$$C = AB^{-1},$$

but the matrix C is never formed. Orthogonal transformations are such that A is kept in upper Hessenberg form and B is kept in upper triangular form at each iteration.

A shift of the origin at each iteration is computed as the eigenvalues of the lowermost 2×2 principal minor if they are complex or that eigenvalue which is closer to a lower diagonal element of this minor if they are real. When a lowermost principal minor of order 1 or 2 has split from the rest of the matrix, the iteration continues with the remaining part of the matrix. The tolerance in the test for splitting is commensurate to the absolute error tolerance ε_1. The iterations proceed until the whole matrix has finally split into minors of 1 and 2 order.

If specified by the input flag *matz*, the function accumulates the transformations used in the reduction process for later use in the eigenvectors computation. The transformations are stored transposed into the array *z*. The transformations are accumulated when *matz* is set to a non-zero value. When it is set to zero, the matrix *z* is *not referenced.*

The input constant *eps1* is an absolute tolerance used to determine negligible elements. A non-positive tolerance may be entered, however in this case an element will be neglected only if it less than roundoff error times the norm of its submatrix. If the *eps1* is positive, then an element would be considered negligible if it is less than *eps1* times the norm of its submatrix. A positive value of *eps1* would lead to a faster execution, but with less accurate results.

The input matrix *a* is in upper Hessenberg form, and the matrix *b* is in upper triangular form.

If specified by the flag *matz*, the input matrix *z* contains transposed transformation matrix obtained in the reduction by the function `qzhes`. Otherwise *z* is not referenced.

Having returned from the function, the matrix *a* is in upper quasi-triangular form. The matrix *b* is still in upper triangular form, but its entries have been altered. If specified by the *matz* flag, the matrix *z* would contain information about the transformations used in both phases.

The entry `b[n-1][0]` contains ε_1 times the norm of output matrix B for later use by the functions `qzval` and `qzvec`.

Performance

The rate of convergence is proportional to a third order. The computation time is affected by the value of the absolute error tolerance *eps1*. It can be reduced by increasing the value of *eps1* but the accuracy of computed eigenvalues and eigenvectors would be reduced as well.

Notes

Matrices *a, b* and *z* are expected to be in the format produced by the allocation function `fsquare`.

Test

The test program is contained in `rggt.c` file. The example matrices are contained in `qz1.xpd, qz2.xpd` files.

References

C. B. Moler, G. W. Stuart, SIAM Journal of Numerical Analysis, Vol. 10, pp.241-256, 1973.

R. C. Ward, Technical Note NASA, TN D-7305 (1973).

R. C. Ward, SIAM Journal of Numerical Analysis, Vol. 12, pp.835-853, 1975.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

3.1.3 QZVAL Generalized Eigenvalues

Name

sqzval — Forms generalized eigenvalues, single precision
dqzval — Forms generalized eigenvalues, double precision

Synopsis

```
#include <ceispack.h>
void sqzval (n, a, b, alfa, beta, matz, z)
int n;
float **a, **b;
fcomplex *alfa;
float *beta;
int matz;
float **z;

void dqzval (n, a, b, alfa, beta, matz, z)
int n;
double **a, **b;
dcomplex *alfa;
double *beta;
int matz;
double **z;
```

n	— order of input/output matrices
a	— quasi-triangular input/output matrix
b	— input matrix in upper triangular form
alfa	— length n output array containing numerators of eigenvalues
beta	— length n output array containing denominators of eigenvalues
matz	— flag which causes accumulation of transformations
z	— transposed orthogonal transformation input/output matrix

Description

Function qzval performs the third phase of the QZ algorithm for solving the generalized matrix eigenvalue problem:

$$Ax = \lambda Bx$$

Function qzval accepts a pair of real matrices, one of them in quasi-triangular form and the other in upper triangular form.

The function proceed as follows. First, the diagonal elements of matrix a that correspond to a principal minors of order 1 are stored in *real* parts of entries of array *alfa*. The signs of these elements may be inverted in order to store *absolute values* of corresponding diagonals of matrix *b* into array *beta*.

At the next step, the function reduces remaining 2×2 blocks by orthogonal transformations into blocks of 1 and 2 order, so that any remaining 2×2 blocks correspond to pairs of *complex* eigenvalues. The transformations are applied if a block can be triangularized in a real arithmetic. If a block corresponds to a pair of complex eigenvalues, the transformations are accumulated in the transformation matrix *z* and complex roots are stored into the array *alpha* with a sign possibly inverted in order to store the *absolute values* of corresponding diagonals of matrix *b* into array *beta*. In such a case, the transformations are not applied. The matrix *b* is being kept upper triangular during the computation process.

The ratios of the corresponding elements of vectors α and β forms the generalized eigenvalues of the original eigenproblem:

$$\lambda_j = \frac{\mathrm{Re}(\alpha_j) + i\mathrm{Im}(\alpha_j)}{\beta_j}$$

If specified by the input flag *matz*, the function accumulates the transformations used in the reduction process for later use in the eigenvectors computation. The transformations are stored transposed into array *z*. The transformations are accumulated when *matz* is set to a non-zero value. When it is set to zero, the matrix *z* is *not referenced*.

Having returned from the function, the matrix *a* has been reduced so that the remaining diagonal blocks of order 2 are corresponding to *complex* eigenvalues. The matrix *b* is still upper triangular but its elements have been modified.

The array *alfa* which contains the real and imaginary parts of the diagonal elements of the triangular matrix that would be obtained if *a* were reduced completely to triangular form by unitary transformations. Non-zero values of $\mathrm{Im}(\alpha_i)$ occur in pairs, the first member positive and the second negative. The entries of the *alfa* array introduce the numerators of the generalized eigenvalues.

The array *beta* contains the diagonal elements of the corresponding matrix *b*, normalized to be real and non-negative. The entries of the *beta* array introduce the denominators of the generalized eigenvalues.

The output array *z* contains the product of the right-hand transformations for all three stages, if specified by the flag *matz*.

Performance

The computed eigenvalues are exact for a system which is a small perturbation of the original matrix system.

Notes

Matrices *a, b* and *z* are expected to be in the format produced by the allocation function `fsquare`.

Test

The test program is contained in `rggt.c` file. The example matrices are contained in `qz1.xpd, qz2.xpd` files.

References

C. B. Moler, G. W. Stuart, SIAM Journal of Numerical Analysis, Vol. 10, pp.241-256, 1973.

R. C. Ward, Technical Note NASA, TN D-7305 (1973).

R. C. Ward, SIAM Journal of Numerical Analysis, Vol. 12, pp.835-853, 1975.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

3.1.4 QZVEC Generalized Eigenvectors

Name

sqzvec — Forms generalized eigenvectors, single precision
dqzvec — Forms generalized eigenvectors, double precision

Synopsis

```
#include <ceispack.h>
void sqzvec (n, a, b, alfa, beta, z, rv)
int n;
float **a, **b;
fcomplex *alfa;
float *beta, **z, *rv;

void dqzvec (n, a, b, alfa, beta, z, rv)
int n;
double **a, **b;
dcomplex *alfa;
double *beta, **z, *rv;
```

n	— order of input/output matrices
a	— square input array containing upper quasi-triangular matrix
b	— square input array containing upper triangular matrix
alfa	— length n input array containing n numerators of eigenvalues
beta	— length n input array containing n denominators of eigenvalues
z	— input transposed transformation/output transposed (row) eigenvector matrix
rv	— temporary storage array of length n

Description

Function qzvec performs the fourth phase of the QZ algorithm for solving the generalized matrix eigenvalue problem:

$$Ax = \lambda Bx$$

Function qzvec accepts a pair of real matrices, one of them in quasi-triangular form in which each 2×2 block corresponds to a pair of complex eigenvalues, and the other in upper triangular form. Then qzvec computes the eigenvectors of the triangular problem by the back substitution process and transforms the results back to the eigenvectors of the original matrix system.

The input matrix z contains the transposed transformation matrix produced in the reduction by the functions `qzhes, qzit` and `qzval`, if they have been performed. If the eigenvectors of the triangular problem are required, z must contain the identity matrix.

The real and imaginary parts of the generalized eigenvectors are stored transposed (row-wise) in the array z. If $\mathrm{Im}(\alpha_i) = 0$, the i^{th} eigenvalue is real and the i^{th} row of z contains the corresponding eigenvector. If $\mathrm{Im}(\alpha_i) \neq 0$ the i^{th} eigenvalue would be complex and two cases arise:

If the imaginary part of an eigenvalue is positive, i.e. $\mathrm{Im}(\alpha_i) > 0$, then the eigenvalue is the first of a conjugate pair of eigenvalues. The i^{th} row of matrix z would contain the real part, while the $(i+1)^{\mathrm{th}}$ row would contain the imaginary part of the corresponding eigenvector.

If the imaginary part of an eigenvalue is negative, i.e. $\mathrm{Im}(\alpha_i) < 0$, then the eigenvalue is the second of a complex pair and the $(i-1)^{\mathrm{th}}$ and i^{th} rows contain correspondingly the real and the imaginary parts of its eigenvector *conjugate*.

All the eigenvectors are normalized so that the absolute value of each vector's largest component is equal to 1.

Performance

The computed eigenvectors are exact for a system which is a small perturbation of the original matrix system.

Notes

Arrays a, b and z are expected to be in the format produced by the allocation function `fsquare`.

The function preserves the input matrix a. Its subdiagonal elements provide the information about the storage of the complex eigenvectors. The vectors α and β are also unaltered.

The matrix b has been modified.

Test

The test program is contained in `rggt.c` file. The example matrices are contained in `qz1.xpd, qz2.xpd` files.

References

C. B. Moler, G. W. Stuart, SIAM Journal of Numerical Analysis, Vol. 10, pp.241-256, 1973.

R. C. Ward, Technical Note NASA, TN D-7305 (1973).

R. C. Ward, SIAM Journal of Numerical Analysis, Vol. 12, pp.835-853, 1975.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

Chapter 4

Real General Eigenproblem

4.1 Matrix Balancing

4.1.1 BALANC Balancing

Name

sbalanc — Balances real matrix, single precision
dbalanc — Balances real matrix, double precision

Synopsis

```
#include <ceispack.h>
void sbalanc (n, a, low, igh, scale)
int n;
float **a;
int *low, *igh;
float *scale;

void dbalanc (n, a, low, igh, scale)
int n;
double **a;
int *low, *igh;
double *scale;
```

n — order of input/output matrices
a — unbalanced input/balanced output matrix
low — address of output variable containing lower index of balanced submatrix
igh — address of output variable containing higher index of balanced submatrix
scale — length n output array containing scaling factors and permutation indices

Description

Function `balanc` balances a real square general matrix and extracts the eigenvalues of the original matrix when they can be found exact.

The reduction performed as follows. At first the matrix is converted to block-triangular form (if it is possible):

$$PAP = \begin{pmatrix} R & \vdots & W & \vdots & X \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & B & \vdots & Y \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & 0 & \vdots & T \end{pmatrix},$$

by applying the permutation similarity transformations P. The matrix B is of order $(igh - low + 1)$ and located in the rows and columns from low to igh. B has the property such that if B_0 is equal to B with its diagonal elements set to zero, then the B_0 does not have any rows or columns of zero 1-norm. Matrices R and T are upper triangular. The other matrices are rectangular of the respective dimensions. The diagonal elements of matrices R and T are the eigenvectors of the original matrix that are found exact.

There are two boundary cases arise. The first one is occur when B is of order 0. In such a case, the function sets the output variables low and igh both to 0. The second case occurs when B is of order n and all the other submatrices are empty. In this case, $P = I$, and the function sets $low = 0$ and $igh = n - 1$.

At the next step the elements of B are transformed by the diagonal similarity transformation:

$$B_f = D^{-1}BD,$$

such that the 1-norms of the rows and, corresponding to them, the columns in the B_f are nearly equal. The entries of the diagonal matrix D (that are scaling factors) are chosen to be integer powers of the base of the machine arithmetic, so that the transformation does not produce any rounding errors. The whole matrix has the form:

$$D^{-1}PAPD = \begin{pmatrix} R & \vdots & WD & \vdots & X \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & D^{-1}BD & \vdots & D^{-1}Y \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & 0 & \vdots & T \end{pmatrix},$$

The 1-norm of the original matrix also is reduced by the diagonal transformation when 1-norms of the columns and, corresponding to them, the rows of the original matrix are significantly different. Generally reducing the 1-norm of a matrix improves the accuracy in determining eigenvalues and eigenvectors.

The similarity transformations applied keep the eigenvalues of the balanced matrix equal to those of the original matrix.

Having returned from the function, the array *scale* contains the information about permutations and diagonal transformations used by the function balanc and should be interpreted as follows:

1. The rows and columns j and $scale_j$ have been interchanged for indices $j \in [0, low-1]$ and $j \in [igh+1, n-1]$. The interchanges are performed in indices first from $(n-1)$ to $(igh+1)$, then from 0 to $(low-1)$.
2. The elements from low to igh of the diagonal matrix D are stored in the locations $scale_j$, where $j \in [low, igh]$.

Performance

Since the non-unitary entries in the diagonal matrix D are integer powers of the base of machine arithmetic, the function balanc produces no rounding errors.

Notes

Array a is expected to be in the format produced by the allocation function `fsquare`.

The function `balbak` should be performed after the computation of eigenvectors of balanced matrix in order to reconstruct the eigenvectors of the original (unbalanced) matrix.

Test

The test programs are contained in `rgt.c, elmbakt.c, ortbakt.c, ortrant.c` files. The example matrices are contained in `rmatrix1.xpd, rmatrix2.xpd, rmatrix3.xpd` files.

References

B. N. Parlett, C. Reinsch, Numerical Mathematics, Vol. 13, pp.293-304, 1969.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

4.2 Reduction to Hessenberg form

4.2.1 ORTHES Reduction by Orthogonal Transformations

Name

sorthes — Reduction by orthogonal transformations, single precision
dorthes — Reduction by orthogonal transformations, double precision

Synopsis

```
#include <ceispack.h>
void sorthes (n, low, igh, a, ort, rv)
int n, low, igh;
float **a, *ort, *rv;

void dorthes (n, low, igh, a, ort, rv)
int n, low, igh;
double **a, *ort, *rv;
```

n — order of input/output matrices
low — input variable set to lower index of balanced submatrix
igh — input variable set to higher index of balanced submatrix
a — balanced input/upper Hessenberg output and transformations matrix
ort — output array containing rest of information about transformations
rv — temporary storage array of length n

Description

Function orthes reduces a submatrix located in the rows and columns from *low* to *igh* of given complex general matrix to upper Hessenberg form using and accumulating orthogonal similarity transformations.

The above submatrix is derived usually from balancing the original general matrix using the function `balanc`, that reduces the original matrix into the block-triangular form:

$$\left(\begin{array}{ccccc} R & \vdots & W & \vdots & X \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & B & \vdots & Y \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & 0 & \vdots & T \end{array}\right),$$

where matrix B is of order $(igh - low + 1)$ and located in the rows and columns from *low* to *igh*, matrices R and T are upper triangular, and the other matrices are rectangular of the respective dimensions.

The reduction is performed as follows. At first, the elements in the i^{th} column below the principal diagonal are scaled to avoid possible underflow that would result in destroying the orthogonality of the transformation.

At the next phase, the element in the $(i+1,i)$ location is expanded by adding the square root of the sum of the squares S_i of scaled elements. Then the elements of the i^{th} column below the principal diagonal form a vector w, that defines Householder reflection:

$$Q = I - ww^T/h, \text{ where } h = w^T w/2,$$

I is the identity matrix, and Q is orthogonal and symmetric. The transformation QFQ eliminates the elements in i^{th} column below the first subdiagonal.

As described above, Hessenberg reflections are applied successively to eliminate elements below the subdiagonal in the columns from *low* to *igh* of the input matrix.

The product of the Householder reflections is accumulated in the *strict lower* triangle below the upper Hessenberg matrix in the array *a* and in the vector *ort*.

Having returned from the function the array *a* contains the upper Hessenberg matrix. Information about the orthogonal transformations used in the reduction is stored in the *strict lower* triangle below the Hessenberg matrix.

The output vector *ort* contains the rest of information about the transformations.

Performance

The reduction process is numerically stable. The computed eigenvalues and eigenvectors would be exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

The whole reduction process involves about $\frac{5}{3}m^3$ multiplications, where $m = igh - low + 1$.

Notes

Matrix *a* is expected to be in the format produced by the allocation function `fsquare`.

Array *ort* should be allocated as a real array of length at least *igh*. The function references only the entries in $[low+1, igh]$ locations of the array *ort*.

If the function `balanc` has not been used to balance the original matrix, the input parameters *low* and *igh* must be set to 0 and $(n-1)$ respectively.

Test

The test programs are contained in `ortbakt.c, ortrant.c` files. The example matrices are contained in `rmatrix1.xpd, rmatrix2.xpd, rmatrix3.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 12, pp.349-368, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

4.2.2 ORTRAN Accumulation of Orthogonal Transformations

Name

sortran — Accumulation of orthogonal transformations, single precision
dortran — Accumulation of orthogonal transformations, double precision

Synopsis

```
#include <ceispack.h>
void sortran (n, low, igh, a, ort, z, rv)
int n, low, igh;
float **a, **z, *ort, *rv;

void dortran (n, low, igh, a, ort, z, rv)
int n, low, igh;
double **a, **z, *ort, *rv;
```

n	— order of input/output matrices
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
a	— upper Hessenberg and transformations input matrix
ort	— input array containing rest of information about transformations
z	— square output array containing transformation matrix
rv	— temporary storage array of length n

Description

Function ortran accumulates the orthogonal similarity transformations used in the reduction of a real general matrix to upper Hessenberg form by the function `orthes`.

If submatrix B, stored in the input array a has been reduced to upper Hessenberg form by the orthogonal similarity transformations:

$$H = Q^T BQ,$$

where Q is the product of the orthogonal transformations that has been stored in array *ort* and in the *strict lower* triangle below upper Hessenberg matrix, then the function ortran accumulates the transformations in the output matrix z for later use in computation of eigenvectors of the original matrix.

Having returned from the function the input array *ort* has been modified.

Performance

This function is numerically stable. The computed eigenvalues and eigenvectors would be exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Arrays a and z are expected to be in the format produced by the allocation function `fsquare`.

Array *ort* should be allocated as a real array of length at least igh. The function references only the entries in $[low + 1, igh]$ locations of the array *ort*.

If the function `balanc` has not been used to balance the original matrix, the input parameters *low* and *igh* must be set to 0 and $(n - 1)$ respectively.

Test

The test program is contained in `ortrant.c` file. The example matrices are contained in `rmatrix1.xpd, rmatrix2.xpd, rmatrix3.xpd` files.

References

G .Peters, J .H .Wilkinson, Numerical Mathematics, Vol. 16, pp.181-204, 1970.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

4.2.3 ELMHES Reduction by Elementary Transformation

Name

selmhes — Reduction by stabilized elementary transformations, single precision
delmhes — Reduction by stabilized elementary transformations, double precision

Synopsis

```
#include <ceispack.h>
void selmhes (n, low, igh, a, iperm)
int n, low, igh;
float **a;
int *iperm;

void delmhes (n, low, igh, a, iperm)
int n, low, igh;
double **a;
int *iperm;
```

n — order of input/output matrices
low — input constant set to lower index of balanced submatrix
igh — input constant set to higher index of balanced submatrix
a — balanced input/upper Hessenberg and transformations output matrix
iperm — output integer array of length at least *igh* containing permutation indices

Description

Function elmhes reduces a submatrix located in the rows and columns from *low* to *igh* of a given complex general matrix to upper Hessenberg form using and accumulates elementary similarity transformations stabilized by permutation transformations.

The above submatrix is derived usually from balancing the original general matrix performed by the function `balanc`, that reduces the original matrix into the block-triangular form:

$$\begin{pmatrix} R & \vdots & W & \vdots & X \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & B & \vdots & Y \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & 0 & \vdots & T \end{pmatrix},$$

where B is of order $(igh - low + 1)$ and located in the rows and columns from *low* to *igh*, R and T are upper triangular, and the other matrices are rectangular of the respective dimensions.

The reduction is performed as follows. The element having the maximum absolute value is searched for in the i^{th} column below the principal diagonal and moved to the $(i+1,i)$ location by the permutation transformations to stabilize the reduction process. Then the elements of the i^{th} column below the first subdiagonal are annihilated by the elementary row transformations.

Column transformations are applied to both of the permutation and elementary transformations to accomplish the similarity transformation. The information about the permutation transformations is stored in the vector *iperm* while the multipliers that define the elementary transformations are stored in place of the eliminated elements of the matrix.

The steps described above are performed successively on the columns from *low* to $(igh-2)$. Having returned from the function, the submatrix B of the input matrix a has been reduced to upper Hessenberg form. The information about permutation transformations is stored into the locations from *low* to *igh* of vector *iperm*. The other entries of the array are not used. The multipliers that have been used in the elementary transformations are stored into the *strict lower* triangle below the upper Hessenberg submatrix in the matrix a.

Performance

The reduction process is numerically stable. The computed eigenvalues and eigenvectors would be exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

The whole reduction process involves about $\frac{5}{6}m^3$ multiplications, where $m = igh - low + 1$.

This function is generally faster than the function `orthes`, since it involves fewer number of arithmetic operations. The disadvantage of the function `elmhes` is in the fact that the transformations used are not orthogonal and possibly may increase the norm of reduced matrix, therefore affecting the precision of the computed eigenvalues and eigenvectors. In practice, such a matrices are rarely arise.

Notes

Array a is expected to be in the format produced by the allocation function `fsquare`.

The function references only the entries in $[low+1, igh-1]$ locations of the array *iperm*.

If the function `balanc` has not been used to balance the original matrix, the input parameters *low* and *igh* must be set to 0 and $(n-1)$ respectively.

Test

The test programs are contained in `rgt.c, elmbakt.c` files. The example matrices are contained in `rmatrix1.xpd, rmatrix2.xpd, rmatrix3.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 12, pp.349-368, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

4.2.4 ELTRAN Accumulation of Elementary Transformations

Name

seltran — Accumulation of elementary transformations, single precision
deltran — Accumulation of elementary transformations, double precision

Synopsis

```
#include <ceispack.h>
void seltran (n, low, igh, a, iperm, z)
int n, low, igh;
float **a, **z;
int *iperm;

void deltran (n, low, igh, a, iperm, z)
int n, low, igh;
double **a, **z;
int *iperm;
```

n	— order of input/output matrices
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
a	— upper Hessenberg and transformations input matrix
iperm	— integer input array of length at least *igh* containing permutation indices
z	— square output array containing the transformation matrix

Description

Function eltran accumulates the stabilized elementary similarity transformations that have been used by the function `elmhes` in the reduction of a real general matrix to upper Hessenberg form.

If submatrix B, stored in the input array a has been reduced to upper Hessenberg form by the permutation and stabilized elementary similarity transformations:

$$H = S^{-1}BS,$$

where S is the product of the permutation matrices and elementary transformations, that has been stored in array *iperm* and in the *strict lower* triangle below the upper Hessenberg matrix, then the function eltran transfers the multipliers stored in the matrix a into the output matrix z for later use in computation of eigenvectors of the original matrix.

Having returned from the function, the array z contains the transformation matrix produced in the reduction process by the function `elmhes`.

Performance

Since the function performs only a copying of the elements of the strict lower triangle of the matrix a into matrix z and interchanges the rows and columns of z determined by vector *iperm* it produces no rounding errors.

Notes

Matrices a and z are expected to be in the format produced by the allocation function `fsquare`.

The function references only the entries in $[low + 1, igh - 1]$ locations of the array *iperm*.

If the function `balanc` has not been used to balance the original matrix, the input parameters *low* and *igh* must be set to 0 and $(n - 1)$ respectively.

Test

The test program is contained in `rgt.c` file. The example matrices are contained in `rmatrix1.xpd, rmatrix2.xpd, rmatrix3.xpd` files.

References

G .Peters, J .H .Wilkinson, Numerical Mathematics, Vol. 16, pp.181-204, 1970.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

4.3 Eigenvalues and Eigenvectors

4.3.1 HQR Eigenvalues, QR Algorithm

Name

shqr — Forms eigenvalues by QR algorithm, single precision
dhqr — Forms eigenvalues by QR algorithm, double precision

Synopsis

```
#include <ceispack.h>
void shqr (n, low, igh, h, w, ierr)
int n, low, igh;
float **h;
fcomplex *w;
int *ierr;

void dhqr (n, low, igh, h, w, ierr)
int n, low, igh;
double **h;
dcomplex *w;
int *ierr;
```

n	— order of input matrix
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
h	— input matrix in upper Hessenberg form
w	— length n output array containing eigenvalues
ierr	— address of output variable containing error completion code

Diagnostics

On output, the variable `*ierr` is set to 0 for normal return. If the limit of `MAXITER*n` iterations is exceeded while the i^{th} eigenvalue is being computed, the error flag `*ierr` is set to $(i+1)$. Eigenvalues corresponding to indices $j \in [*ierr, n-1]$ would be correct.

Description

Function hqr computes all the eigenvalues of a real matrix in upper Hessenberg form by the QR algorithm. This algorithm iterates a sequence of upper Hessenberg matrices that are orthogonally similar to the original matrix. The sequence of upper Hessenberg matrices converges to an upper quasi-triangular form, such that its principal diagonal contains only blocks of first and second order. The eigenvalues of such a matrix are those of the principal 1×1 and 2×2 diagonal blocks.

A shift of origin before the next iteration generally improves the rate of convergence. The current upper Hessenberg form is tested before the next iteration for possible splitting to submatrices. The tolerance for the test is proportional to relative machine precision. If the matrix splits, the next iterations continue with the lower submatrix.

The value of the origin shift is chosen be equal to the eigenvalues of the 2×2 lowermost principal minor of the currently iterating submatrix. Iterations proceed until the whole matrix has finally split into minors of first and second order.

If an eigenvalue has not converged within 10 consecutive iterations, the additional shift of origin is applied to improve convergence.

The arithmetic used in function hqr is kept real during the whole computation process by using a double QR step and two real or a pair of complex conjugate shifts of origin.

The hqr function iterates the submatrix which is located in the rows and columns low through igh (see the description of the `balanc` function for more details about the variables low and igh) and concerns the diagonal elements of the submatrices located in the rows and columns 0 through low and $(igh + 1)$ through $(n - 1)$ as the eigenvalues of original matrix. These eigenvalues are computed exact. If the `balanc` function has not been used, the input variables low and igh must be set to 0 and $(n - 1)$ respectively.

Having returned from the function, the array w contains the eigenvalues of the original matrix. The complex conjugate pairs of eigenvalues are stored such that the eigenvalue having the positive imaginary part is stored first.

The eigenvalues of a real general matrix can also be found if the functions `elmhes` or `orthes` have been used to reduce the original general matrix into upper Hessenberg form. Generally it is recommended to use the function `balanc` to balance the original general matrix before the reduction to upper Hessenberg form.

Performance

It is expected that the QR algorithm converges in most the cases. Appropriate shift of origin makes the convergence rate very fast.

The computed eigenvalues are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of the original matrix.

Notes

Matrix h is expected to be in the format produced by the allocation function `fsquare`.

On input, the lower triangle below the first subdiagonal of the input matrix h contains the information about the orthogonal transformations used in the reduction of a general matrix to upper Hessenberg form by the functions `orthes` or `elmhes` if they have been used.

The upper Hessenberg part and two adjacent subdiagonals of the input matrix h are modified by the function hqr.

Test

The test programs are contained in `hqrt.c, elmbakt.c, ortbakt.c` files. The example matrices are contained in `rmatrix1.xpd, rmatrix2.xpd, rmatrix3.xpd` and `rhessen1.xpd, rhessen2.xpd, rhessen3.xpd, rhessen4.xpd, rhessen5.xpd, rhessen6.xpd` files.

References

J. G. F. Francis, Computer Journal, Vol. 4, pp.332-345, 1962.

R. S. Martin, G .Peters, J. H. Wilkinson, Numerical Mathematics, Vol. 14, pp.219-231, 1970.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

4.3.2 HQR2 Eigenvalues and Eigenvectors, QR Algorithm

Name

shqr2 — Forms eigenvalues and eigenvectors, single precision
dhqr2 — Forms eigenvalues and eigenvectors, double precision

Synopsis

```
#include <ceispack.h>
void shqr2 (n, low, igh, h, w, z, ierr)
int n;
int low, igh;
float **h, **z;
fcomplex *w;
int *ierr;

void dhqr2 (n, low, igh, h, w, z, ierr)
int n; int low, igh;
double **h, **z;
dcomplex *w;
int *ierr;
```

n	— order of input/output matrices
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
h	— input matrix in upper Hessenberg form
w	— length n output array containing eigenvalues
z	— input transformations (identity)/output column eigenvectors matrix
ierr	— address of output variable containing error completion code

Diagnostics

On output, the variable `*ierr` is set to 0 for normal return. If the limit of `MAXITER*n` iterations is exceeded while the i^{th} eigenvalue is being computed, the error flag `*ierr` is set to $(i+1)$. Eigenvalues corresponding to the indices $[*ierr, n-1]$ would be correct. None of the eigenvectors have yet been determined.

Description

Function hqr2 computes all the eigenvalues and corresponding eigenvectors of a real matrix in upper Hessenberg form by the QR algorithm. This algorithm iterates a sequence of upper Hessenberg matrices that are orthogonally similar to the original matrix. The sequence of upper Hessenberg matrices converges to an upper quasi-triangular form, such that its principal diagonal contains only blocks of first and second order. The eigenvalues of such a matrix are those of the principal 1 × 1 and 2 × 2 diagonal blocks.

A shift of origin before the next iteration generally improves the rate of convergence. The current upper Hessenberg form is tested before the next iteration for possible splitting to submatrices. The tolerance for the test is proportional to relative machine precision. If the matrix splits, the next iterations continue with the lower submatrix.

The value of the origin shift is chosen be equal to the eigenvalues of the 2 × 2 lowermost principal minor of the currently iterating submatrix. Iteration proceeds until the whole matrix has finally split into minors of first and second order.

If an eigenvalue has not converged within 10 consecutive iterations, the additional shift of origin is applied to improve the convergence rate.

The arithmetic used in function hqr2 is kept real during the whole computation process by using a double QR step and two real or a pair of complex conjugate shifts of origin.

The hqr2 function iterates the submatrix which is located in the rows and columns *low* through *igh* (see the description of the `balanc` function for more details about the variables *low* and *igh*) and concerns the diagonal elements of the submatrices located in the rows and columns 0 through *low* and $(igh + 1)$ through $(n - 1)$ as the eigenvalues of the original matrix. These eigenvalues are computed exact. If the `balanc` function has not been used, the input variables *low* and *igh* must be set to 0 and $(n - 1)$ respectively.

The eigenvectors of the finally converged quasi-triangular matrix are computed by the back substitution process and multiplied by the transformation matrix to be back transformed into the eigenvectors of the original matrix.

The eigenvalues and eigenvectors of a real general matrix can also be found if the functions `elmhes` or `orthes` have been used to reduce the original general matrix into upper Hessenberg form and the functions `eltran` or `ortran` have been used to accumulate the similarity transformations in the array z.

Generally, it is recommended to use the function `balanc` to balance the original general matrix before the reduction to upper Hessenberg form. Note that the function `balbak` must follow hqr2 if `balanc` has been used.

Having returned from the function, the array w contains the eigenvalues of the original matrix. The complex conjugate pairs of eigenvalues are stored such that the eigenvalue having the positive imaginary part is stored first.

On input, matrix z contains the transformation matrix produced by the function `eltran` after the reduction by the function `elmhes`, or by the function `ortran` after the reduction by the function `orthes` if they have been used. If the eigenvectors of the Hessenberg matrix are required, z must contain the identity matrix.

On output, the columns of matrix z contains the real and imaginary parts of the eigenvectors. If the i^{th} eigenvalue is real, the i^{th} column of matrix z contains its eigenvector.

If the i^{th} eigenvalue is complex then two cases arise: if the imaginary part of an eigenvalue is positive, the i^{th} column of array z would contain the real part, while the $(i+1)^{\text{th}}$ column of z contains the imaginary part of the corresponding eigenvector. The *conjugate* of this vector is the eigenvector for the *conjugate* eigenvalue. The eigenvectors are unnormalized.

Performance

It is expected that the QR algorithm converges in most the cases. Appropriate shift of origin makes the convergence rate very fast.

The computed eigenvalues and eigenvectors are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of the original matrix.

Notes

Matrices h and z are expected to be in the format produced by the allocation function `fsquare`.

The upper Hessenberg part of the input matrix h is modified by the function hqr2.

Test

The test programs are contained in `hqr2t.c, ortrant.c` files. The example matrices are contained in `rmatrix1.xpd, rmatrix2.xpd, rmatrix3.xpd` and `rhessen1.xpd, rhessen2.xpd, rhessen3.xpd, rhessen4.xpd, rhessen5.xpd, rhessen6.xpd` files.

References

J. G. F. Francis, Computer Journal, Vol. 4, pp.332-345, 1962.

G .Peters, J. H. Wilkinson, Numerical Mathematics, Vol. 16, pp.181-204, 1971.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

4.3.3 INVIT Eigenvectors Corresponding to Specified Eigenvalues

Name

sinvit — Finds eigenvectors corresponding to specified eigenvalues, single precision
dinvit — Finds eigenvectors corresponding to specified eigenvalues, double precision

Synopsis

```
#include <ceispack.h>

void sinvit (n, a, w, select, mm, m, mz, z, ierr, rm, rv1, rv2)
int n;
float **a, **z;
fcomplex *w;
int *select, mm, *m, mz, *ierr;
float **rm, *rv1, *rv2;

void dinvit (n, a, w, select, mm, m, mz, z, ierr, rm, rv1, rv2)
int n;
double **a, **z;
dcomplex *w;
int *select, mm, *m, mz, *ierr;
double **rm, *rv1, *rv2;
```

n	— order of input matrix a and number of rows in eigenvector matrix
a	— input matrix in upper Hessenberg form
w	— length n input array containing n eigenvalues
select	— length n input array which specifies required eigenvectors
mm	— input constant specifying upper bound for number of required eigenvectors
m	— address of output variable containing number of eigenvectors actually computed
mz	— number of columns in eigenvector matrix z
z	— rectangular output matrix containing column eigenvectors
ierr	— address of output variable containing error completion code
rm	— order n temporary storage square matrix
rv1	— temporary storage array of length n
rv2	— temporary storage array of length n

Diagnostics

The function returns 0 for normal return or 1 if an error exit has been made. The output error flag `*ierr` is set to:

- 0 for normal return.

- $-(2n+1)$ if more than mm eigenvectors have been specified. The output variable `*m` is set to either mm or $mm-1$ and specifies the number of eigenvectors that have been found.
- $-k$ if the iteration corresponding to the k^{th} eigenvalue failed.
- $-(n+k)$ if both the error situations occurred.

Description

Function invit computes the eigenvectors of a real matrix in upper Hessenberg form, which corresponds to the specified eigenvalues, using the inverse iteration technique. Computed eigenvectors are normalized so that the component of the largest magnitude is set equal to 1.

The function invit solves the linear system

$$Uy = y_0$$

in order to obtain the eigenvector corresponding to the i^{th} eigenvalue, where U is the upper triangular multiplier in the LU decomposition of the matrix

$$F = B - \mu_i I$$

where B is the leading order p submatrix of the input upper Hessenberg matrix. μ_i is the approximate eigenvalue of the matrix A. Starting from an initial vector, the approximate eigenvector is obtained by the back substitution process. The solution vector y is accepted as the eigenvector of the F if its norm satisfies the following acceptance test:

$$\sqrt{p}\,\varepsilon\|F\|_\infty\|y\|_\infty > \frac{\|y_0\|_\infty}{10}$$

If the acceptance test failed, up to p orthogonal initial vectors are tried successively to obtain an appropriate norm growth. The accepted vector is transformed to the eigenvector of the original matrix by adding $(n-p)$ zeros after its last component.

If all the initial vectors have not produce an accepted eigenvector, the function invit sets the error flag *ierr* to $-i$, where i is the index of corresponding eigenvalue, terminates the iterations for this eigenvector and continue the computation process for the next eigenvalue. Components of the unaccepted eigenvector are set to zero. If such a situation occurs more than once, the output error flag would contain the last index of the eigenvalue for which the computation of an eigenvector failed.

The real parts of multiple eigenvalues or a group of close eigenvalues are perturbed by adding small multiples of $\varepsilon\|B\|$ in order to obtain linearly independent eigenvectors.

The input array w contains eigenvectors of the input upper Hessenberg matrix. They are stored unordered except that complex conjugate pairs must be stored successively, and eigenvalues of any submatrix of the input upper Hessenberg matrix must have indices in the array w between the boundary indices of this submatrix. If `hqr` function has been used to find eigenvalues of an upper Hessenberg matrix, its output array of eigenvectors is arranged in the required order.

The entries of the input vector *select* specify the eigenvectors to be found. The entry corresponding to the i^{th} eigenvalue must be set to a non-zero value if the i^{th} eigenvector is required or to 0 otherwise.

Having returned from the function, the real parts of multiple eigenvalues or a group of close eigenvalues may have been perturbed a little in order to compute linearly independent eigenvectors.

The output vector *select* may have been altered. If the entries of the vector corresponding to a pair of complex conjugate eigenvalues were each initially set to a non-zero value, the function invit resets the second of the two elements to 0.

The output variable m is set to the number of columns actually used to store the eigenvectors. Note that an eigenvector corresponding to a complex eigenvalue require two columns to store its real and imaginary parts.

The columns of output matrix z contains the real and imaginary parts of the eigenvectors. If the next selected eigenvalue is real then the next column of matrix z contains its eigenvector. If the eigenvalue is complex, the next two columns of z contain the real and imaginary parts of its eigenvector. The eigenvectors are normalized so that the component of the largest magnitude is 1. Any vector which does not satisfy the acceptance test is set to zero.

Performance

The rate of convergence of the inverse iterations is linear. Computed eigenvectors are the exact ones of the perturbed original matrix. The value of the perturbation is proportional to the machine precision times the norm of original matrix.

Notes

Matrices a and rm are expected to be in the format produced by the allocation function `fsquare`.

Matrix z of dimension $n \times mz$ is expected to be in the format produced by the allocation function `fmatrix`.

The input matrix a is preserved by the function invit.

Test

The test programs are contained in `hqrt.c, elmbakt.c, ortbakt.c` files. The example matrices are contained in `rmatrix1.xpd, rmatrix2.xpd, rmatrix3.xpd` and `rhessen1.xpd, rhessen2.xpd, rhessen3.xpd, rhessen4.xpd, rhessen5.xpd, rhessen6.xpd` files.

References

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

4.3.4 ORTBAK Eigenvectors of Original Matrix

Name

sortbak — Forms eigenvectors of original matrix, single precision
dortbak — Forms eigenvectors of original matrix, double precision

Synopsis

```
#include <ceispack.h>
void sortbak (n, low, igh, a, ort, m, z, rv)
int n, low, igh;
float **a, *ort;
int m;
float **z, *rv;

void dortbak (n, low, igh, a, ort, m, z, rv)
int n, low, igh;
double **a, *ort;
int m;
double **z, *rv;
```

n	— order of input array a and number of rows in eigenvector array
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
a	— input upper Hessenberg and transformations matrix
ort	— input array containing rest of information about transformations
m	— number of columns in eigenvector array
z	— input/output eigenvector array
rv	— temporary storage array of length n

Description

Function ortbak forms the eigenvectors of a real general matrix by back transforming the eigenvectors of the corresponding upper Hessenberg matrix determined by the function `orthes`.

If submatrix B, stored in the input array a has been reduced to upper Hessenberg form by orthogonal similarity transformations:

$$H = Q^T BQ,$$

where Q is the product of the orthogonal transformations, that has been stored in array *ort* and in the *strict lower* triangle below upper Hessenberg matrix, then the function ortbak computes for each eigenvector x_i of the upper Hessenberg matrix the product:

$$z_i = Qx_i, \text{ where } i \in [0, m-1]$$

thus forming the eigenvectors of the original matrix B. The transformed eigenvectors are stored column-wise in the matrix z, overwriting the input eigenvectors of the upper Hessenberg matrix.

Performance

The computed eigenvectors are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of the original matrix.

Notes

Array a is expected to be in the format produced by the allocation function `fsquare`.

Rectangular $n \times m$ matrix z is expected to be in the format produced by the allocation function `fmatrix`.

It is not required by the function that the real and imaginary parts of an eigenvector are stored in the sequential columns of the matrix z.

Array *ort* should be allocated as a real array of length at least *igh*.

If the function `balanc` has not been used to balance the original matrix, the input parameters *low* and *igh* must be set to 0 and $(n-1)$ respectively.

Test

The test program is contained in `ortbakt.c` file. The example matrices are contained in `rmatrix1.xpd, rmatrix2.xpd, rmatrix3.xpd` files.

References

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 12, pp.349-368, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

4.3.5 ELMBAK Eigenvectors of Original Matrix

Name

selmbak — Forms eigenvectors of original matrix, single precision
delmbak — Forms eigenvectors of original matrix, double precision

Synopsis

```
#include <ceispack.h>
void selmbak (n, low, igh, a, iperm, m, z)
int n, low, igh;
float **a;
int *iperm, m;
float **z;

void delmbak (n, low, igh, a, iperm, m, z)
int n, low, igh;
double **a;
int *iperm, m;
double **z;
```

n	— order of input matrix *a* and number of rows in eigenvector matrix
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
a	— upper Hessenberg and transformations input matrix
iperm	— length *n* input integer array containing permutation indices
m	— number of columns in eigenvector matrix
z	— column eigenvector input/output matrix

Description

Function elmbak forms the eigenvectors of a real general matrix by back transforming the eigenvectors of the corresponding upper Hessenberg matrix determined by the function `elmhes`.

If submatrix B, stored in the input array a has been reduced to upper Hessenberg form by permutation and stabilized elementary similarity transformations:

$$H = S^{-1}BS,$$

where S is the product of the permutation matrices and elementary transformations, that have been stored in array *iperm* and in the *strict lower* triangle below upper Hessenberg matrix, then the function elmbak computes for each eigenvector x_i of the upper Hessenberg matrix the product:

$$z_i = Sx_i, \text{ where } i \in [0, m-1]$$

thus forming the eigenvectors of the original matrix B. The transformed eigenvectors are stored column-wise in the matrix z, overwriting the input eigenvectors of the upper Hessenberg matrix.

Performance

The computed eigenvectors are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of the original matrix.

Notes

Matrix a is expected to be in the format produced by the allocation function `fsquare`.

Rectangular $n \times m$ matrix z is expected to be in the format produced by the allocation function `fmatrix`.

It is not required by the function that the real and imaginary parts of an eigenvector are stored in the sequential columns of the matrix z.

The function references only the entries in $[low + 1, igh - 1]$ locations of the array *iperm*.

If the function `balanc` has not been used to balance the original matrix, the input parameters *low* and *igh* must be set to 0 and $(n - 1)$ respectively.

Test

The test program is contained in `elmbakt.c` file. The example matrices are contained in `rmatrix1.xpd, rmatrix2.xpd, rmatrix3.xpd` files.

References

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 12, pp.349-368, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

4.3.6 BALBAK Eigenvectors of Non-balanced Matrix

Name

sbalbak — Forms eigenvectors of non-balanced matrix, single precision
dbalbak — Forms eigenvectors of non-balanced matrix, double precision

Synopsis

```
#include <ceispack.h>
void sbalbak (n, low, igh, scale, m, z)
int n, low, igh;
float *scale;
int m;
float **z;

void dbalbak (n, low, igh, scale, m, z)
int n, low, igh;
double *scale;
int m;
double **z;
```

n	— number of rows in eigenvector matrix
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
scale	— length n input array containing scaling factors and permutation indices
m	— number of columns in eigenvector matrix
z	— column eigenvector input/output matrix

Description

Function balbak forms the eigenvectors of a real general matrix by back transforming the corresponding eigenvectors of the balanced matrix that has been determined by the function `balanc`.

The function `balanc` transforms the original matrix A into the balanced block-triangular matrix A_b applying the similarity transformations:

$$A_b = D^{-1} P^T A P D,$$

where P is the product of permutation transformations. D is the diagonal matrix of scaling factors, such that its locations within the indices $j \in [low, igh]$ are integer powers of the base of machine arithmetic and all other locations are 1.

Function balbak back transforms the eigenvectors of the balanced matrix to the eigenvectors of the original matrix by forming the product:

$$z_i = PDx_i,$$

where x_i is the i^{th} eigenvector of the balanced matrix, and z_i is the corresponding eigenvector of the original matrix.

The back transformed eigenvectors are stored column-wise in the matrix z, overwriting the input eigenvectors of the balanced matrix.

Performance

Since the non-unitary entries in the diagonal matrix D are integer powers of the base of machine arithmetic, the function balbak produces no rounding errors.

Notes

Rectangular $n \times m$ matrix z is expected to be in the format produced by the allocation function `fmatrix`.

It is not required by the function that the real and imaginary parts of an eigenvector are stored in the sequential columns of the matrix z.

Test

The test programs are contained in `rgt.c, elmbakt.c, ortbakt.c, ortrant.c` files. The example matrices are contained in `rmatrix1.xpd, rmatrix2.xpd, rmatrix3.xpd` files.

References

B. N. Parlett, C. Reinsch, Numerical Mathematics, Vol. 13, pp.293-304, 1969.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

4.3.7 BALBKL Left-hand Eigenvectors of Non-balanced Matrix

Name

sbalbkl — Forms left-hand eigenvectors of non-balanced matrix, single precision
dbalbkl — Forms left-hand eigenvectors of non-balanced matrix, double precision

Synopsis

```
#include <ceispack.h>
void sbalbkl (n, low, igh, scale, m, z)
int n, low, igh;
float *scale;
int m;
float **z;

void dbalbkl (n, low, igh, scale, m, z)
int n, low, igh;
double *scale;
int m;
double **z;
```

n	— number of rows in left-hand eigenvector matrix
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
scale	— length n input array containing scaling factors and permutation indices
m	— number of columns in left-hand eigenvector matrix
z	— column left-hand eigenvector input/output matrix

Description

Function balbkl forms the left-hand eigenvectors of a real general matrix by back transforming corresponding left-hand eigenvectors of balanced matrix that have been determined by the function `balanc`.

The function `balanc` transforms the original matrix A into the balanced block-triangular matrix A_b applying the similarity transformations:

$$A_b = D^{-1}P^TAPD,$$

where P is the product of permutation transformations. D is the diagonal matrix of scaling factors, such that its locations within the indices $j \in [low, igh]$ are integer powers of the base of machine arithmetic and all other locations are 1.

Function balbkl back transforms the left-hand eigenvectors of the balanced matrix to the left-hand eigenvectors of the original matrix by forming the product:

$$z_i = PD^{-1}x_i,$$

where x_i is the i^{th} left-hand eigenvector of the balanced matrix, and z_i is the corresponding left-hand eigenvector of the original matrix.

The back transformed left-hand eigenvectors are stored column-wise in the matrix z, overwriting the input left-hand eigenvectors of the balanced matrix.

Performance

Since the non-unitary entries in the diagonal matrix D are integer powers of the base of machine arithmetic, the function balbkl produces no rounding errors.

Notes

Rectangular $n \times m$ matrix z is expected to be in the format produced by the allocation function `fmatrix`.

It is not required by the function that the real and imaginary parts of an eigenvector are stored in the sequential columns of the matrix z.

Function balbkl is a modification of the C function `balbak`.

References

B. N. Parlett, C. Reinsch, Numerical Mathematics, Vol. 13, pp.293-304, 1969.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

Chapter 5

Symmetric Generalized Eigenproblem

5.1 Reduction to Standard Symmetric Eigenproblem

5.1.1 REDUC Reduction Using Cholesky Factorization

Name

sreduc — Reduction to standard symmetric eigenproblem, single precision
dreduc — Reduction to standard symmetric eigenproblem, double precision

Synopsis

```
#include <ceispack.h>
void sreduc (n, a, b, dl, ierr)
int n;
float **a, **b, *dl;
int *ierr;

void dreduc (n, a, b, dl, ierr)
int n;
double **a, **b, *dl;
int *ierr;
```

n	— order of matrices; flag which prevents from computation of Cholesky factor
a	— symmetric input/reduced output symmetric matrix
b	— positive definite input matrix/output Cholesky factor
dl	— length n output array containing diagonals of Cholesky factor
ierr	— address of output variable containing error completion code

Diagnostics

The output error flag `*ierr` is set to:

- 0 for normal return.
- $(7n + 1)$ if matrix b is not positive definite. In such a case, the Cholesky decomposition of the matrix is not exist.

Description

Function reduc reduces the generalized symmetric eigenproblem:

$$Ax = \lambda Bx,$$

where B is positive definite, to the standard symmetric eigenproblem:

$$Gz = \lambda z,$$

where G is the matrix product:

$$G = L^{-1}AL^{-T},$$

using Cholesky factorization of the matrix B. Vector z is defined as $z = L^T x$.

At first, the function estimates the Cholesky decomposition:

$$B = LL^T$$

where L is lower triangular. If the Cholesky factor L of B is already known (this is specified by setting the input parameter n to be negative), it can be passed to the function in place of matrix B, therefore the first phase of the function reduc is omitted.

At the second phase, the function calculates the matrix product G, that has the same eigenvalues as the original system. The eigenvectors of the reduced system can be back transformed into the eigenvectors of the original system, using the function `rebak`.

The input constant n is set to the order of the matrices a and b. If the Cholesky factor of matrix b is already available then n should be negative.

The input matrix a contains the symmetric input matrix. Only the *full upper* triangle of the matrix need be supplied.

The input matrix b contains the symmetric positive input matrix. Only the *full upper* triangle of the matrix need be supplied. If the input n is negative, the *strict lower* triangle of matrix b contains the *strict lower* triangle of its Cholesky factor. In this case, the input vector dl contains the diagonal elements of the Cholesky factor.

Having returned from the function, matrix *a* contains, in its *full lower* triangle, the em full lower triangle of the symmetric matrix derived from the reduction to the standard form. The *strict upper* triangle of the matrix is unaltered.

In its *strict lower* triangle, matrix *b* contains the *strict lower* triangle of its Cholesky factor. The *full upper* triangle of the matrix is unaltered.

Array *dl* contains the diagonal elements of the Cholesky factor of *b*.

Performance

The computed Cholesky factor is exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of the original matrix.

There are $\frac{1}{6}n^3$ multiplications required in computation of the Cholesky factor.

The matrix product G formed by the function reduc also is computed within very small error level. There are about $\frac{2}{3}n^3$ (due to the symmetry property) multiplications required in the computation of the lower triangle of the product.

Notes

Matrices *a* and *b* are expected to be in the format produced by the allocation function `fsquare`.

Test

The test program is contained in `rsgt.c` file. The example matrices are contained in `rgensym.xpd` file.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 11, pp.99-110, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

5.1.2 REDUC2 Reduction Using Cholesky Factorization

Name

sreduc2 — Reduction to standard symmetric eigenproblem, single precision
dreduc2 — Reduction to standard symmetric eigenproblem, double precision

Synopsis

```
#include <ceispack.h>
void sreduc2 (n, a, b, dl, ierr)
int n;
float **a, **b, *dl;
int *ierr;

void dreduc2 (n, a, b, dl, ierr)
int n;
double **a, **b, *dl;
int *ierr;
```

n — order of matrices; flag which prevents from computation of Cholesky factor
a — symmetric input/reduced symmetric output matrix
b — positive definite input matrix/output Cholesky factor
dl — length n output array containing diagonals of Cholesky factor
ierr — address of output variable containing error completion code

Diagnostics

The output error flag `*ierr` is set to:

- 0 for normal return.
- $(7n+1)$ if matrix b is not positive definite. In such a case, the Cholesky decomposition of the matrix is not exist.

Description

Function reduc2 reduces either the generalized symmetric eigenproblems:

$$ABx = \lambda x \text{ or } BAx = \lambda x,$$

where B is positive definite, to the standard symmetric eigenproblem:

$$Gz = \lambda z,$$

where G is the matrix product:

$$G = L^T AL,$$

using Cholesky factorization of matrix B. Vector z is defined as

$$z = L^T x \text{ or } z = L^{-1} x,$$

respectively.

At first, the function estimates the Cholesky decomposition:

$$B = LL^T$$

where L is lower triangular. If the Cholesky factor L of matrix B is already known (this is specified by setting the input parameter n to be negative), it can be passed to the function in place of B, therefore the first phase of the function reduc2 is omitted.

At the second phase, the function calculates the matrix product G, that has the same eigenvalues as the original system. The eigenvectors of the reduced system can be back transformed into the eigenvectors of the original system, using the functions `rebak` or `rebakb`.

The input constant n is set to the order of the matrices a and b. If the Cholesky factor of matrix b is already available then n should be negative.

The input matrix a contains the symmetric input matrix. Only the *full upper* triangle of the matrix need be supplied.

The input matrix b contains the symmetric positive input matrix. Only the *full upper* triangle of the matrix need be supplied. If the input n is negative, the *strict lower* triangle of matrix b contains the *strict lower* triangle of its Cholesky factor. In this case the input vector *dl* contains the diagonal elements of the Cholesky factor.

Having returned from the function, matrix a contains, in its *full lower* triangle, the *full lower* triangle of the symmetric matrix derived from the reduction to the standard form. The *strict upper* triangle of the matrix is unaltered.

In its *strict lower* triangle, matrix b contains the *strict lower* triangle of its Cholesky factor. The *full upper* triangle of the matrix is unaltered.

Array *dl* contains the diagonal elements of the Cholesky factor of b.

Performance

The computed Cholesky factor is exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of the original matrix.

There are $\frac{1}{6}n^3$ multiplications required in computation of the Cholesky factor.

The matrix product G formed by the function reduc2 also is computed within very small error level. There are about $\frac{2}{3}n^3$ (due to the symmetry property) multiplications required in the computation of the lower triangle of the product.

Notes

Matrices a and b are expected to be in the format produced by the allocation function `fsquare`.

Test

The test programs are contained in `rsgabt.c, rsgbat.c` files. The example matrices are contained in `rgensym.xpd` file.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 11, pp.99-110, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

5.2 Generalized Eigenvectors

5.2.1 REBAK Eigenvectors of Original Matrix System

Name

srebak — Forms generalized eigenvectors, single precision
drebak — Forms generalized eigenvectors, double precision

Synopsis

```
#include <ceispack.h>
void srebak (n, b, dl, m, z)
int n;
float **b, *dl;
int m;
float **z;

void drebak (n, b, dl, m, z)
int n;
double **b, *dl;
int m;
double **z;
```

n — order of matrix b and number of columns in eigenvector matrix
b — input Cholesky factor
dl — length n input array containing diagonals of Cholesky factor
m — number of eigenvectors to be back transformed
z — input eigenvector/output transposed (row) generalized eigenvectors matrix

Description

Function rebak computes the eigenvectors of either the generalized symmetric eigensystems:

$$Ax = \lambda Bx \text{ or } ABx = \lambda x$$

by back transforming the eigenvectors of the reduced symmetric matrix determined by either functions `reduc` or `reduc2`.

If the first generalized symmetric eigensystem with the positive definite matrix B has been reduced to the standard symmetric eigenproblem:

$$Gz = \lambda z$$

by the transformation $G = L^{-1}AL^{-T}$ and the second generalized eigensystem has been reduced by the transformation $G = L^{T}AL$, where L is the Cholesky factor of the positive definite matrix B, then the function `rebak` back transforms the eigenvectors of the reduced eigensystem to those of the original system. The eigenvectors of the original system are obtained by calculating the product:

$$x = L^{-T} z$$

If the input eigenvectors are normalized having their Euclidean norms equal to 1, then the back transformed eigenvectors would satisfy the generalized normalization:

$$x^T B x = 1$$

In its strict lower triangle,input matrix b contains information about the transformation (namely, the Cholesky factor of B) used in the reduction by the either functions `reduc` or `reduc2`.

The input vector dl contains the diagonals of the Cholesky factor.

The input matrix z contains the m transposed (row) eigenvectors to be back transformed.

Having returned from the function, matrix z contains the transformed (transposed) orthogonal eigenvectors in its first m rows.

Performance

The computed eigenvectors are close to the exact eigenvectors of a system which is small perturbation of the original matrix system. Occasionally they may be ill-conditioned, if the positive definite matrix of the original system is ill-conditioned. In practice such a situation is very rare.

Notes

Matrix b is expected to be in the format produced by the allocation functions `fsquare`, `trngl_fmatrix` or `fsym`. If the matrix is allocated by the function `trngl_fmatrix` the lower triangle must be specified.

Rectangular $m \times n$ matrix z is expected to be in the format produced by the allocation function `fmatrix`. The number of rows in the matrix must not be less than m.

Test

The test programs are contained in `rsgt.c, rsgabt.c` files. The example matrices are contained in `rgensym.xpd` file.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 11, pp.99-110, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

5.2.2 REBAKB Eigenvectors of Original Matrix System

Name

srebakb — Forms generalized eigenvectors, single precision
drebakb — Forms generalized eigenvectors, double precision

Synopsis

```
#include <ceispack.h>

void srebakb (n, b, dl, m, z)
int n;
float **b, *dl;
int m;
float **z;

void drebakb (n, b, dl, m, z)
int n;
double **b, *dl;
int m;
double **z;
```

n	— order of matrix b and number of columns in eigenvector matrix
b	— input Cholesky factor
dl	— length n input array containing diagonals of Cholesky factor
m	— number of eigenvectors to be back transformed
z	— input eigenvector/output transposed (row) generalized eigenvectors matrix

Description

Function rebakb computes the eigenvectors of the generalized symmetric eigensystem:

$$BAx = \lambda x,$$

with the positive definite matrix B, by back transforming the eigenvectors of the reduced symmetric matrix determined by function `reduc2`.

If the above generalized symmetric eigensystem has been reduced to the standard symmetric eigenproblem:

$$Gz = \lambda z,$$

by the transformation $G = L^T AL$, where L is the Cholesky factor of the symmetric positive definite matrix B, then the function rebakb back transforms the eigenvectors of the reduced eigensystem to those of the original system. The eigenvectors of the original system are obtained by calculating the product

$$x = Lz$$

If the input eigenvectors are normalized having their Euclidean norms equal to 1, then the back transformed eigenvectors would satisfy the generalized normalization:

$$x^T B^{-1} x = 1$$

In its strict lower triangle, the input matrix b contains information about the transformation (namely, the Cholesky factor of matrix B) used in the reduction by the function `reduc2`.

The input vector dl contains the diagonals of the Cholesky factor.

The input matrix z contains the m transposed (row) eigenvectors to be back transformed.

Having returned from the function, matrix z contains the transformed (transposed) orthogonal eigenvectors in its first m rows.

Performance

The computed eigenvectors are close to the exact eigenvectors of a system which is small perturbation of the original matrix system. Occasionally they may be ill-conditioned, if the positive definite matrix of original system is ill-conditioned. In practice such a situation is very rare.

Notes

Matrix b is expected to be in the format produced by the allocation functions `fsquare`, `trngl_fmatrix` or `fsym`. If the matrix is allocated by the function `trngl_fmatrix` the lower triangle must be specified.

Rectangular $m \times n$ matrix z is expected to be in the format produced by the allocation function `fmatrix`. The number of rows in the matrix must not be less than m.

Test

The test program is contained in `rsgbat.c` file. The example matrices are contained in `rgensym.xpd` file.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 11, pp.99-110, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

Chapter 6

Symmetric Eigenproblem

6.1 Reduction to Symmetric Tridiagonal Form

6.1.1 TRED1 Reduction by Orthogonal Transformations

Name

stred1 — Reduction to symmetric tridiagonal form, single precision
dtred1 — Reduction to symmetric tridiagonal form, double precision

Synopsis

```
#include <ceispack.h>
void stred1 (n, a, d, e, e2)
int n;
float **a, *d, *e, *e2;

void dtred1 (n, a, d, e, e2)
int n;
double **a, *d, *e, *e2;
```

n	— order of input Hermitian/output symmetric tridiagonal matrices
a	— symmetric input matrix (lower triangle)/transformation output matrix
d	— output vector of length n containing diagonal elements of output matrix
e	— output vector of length n containing subdiagonal elements of output matrix
e2	— length n output array of squares of subdiagonal elements of output matrix

Description

Function tred1 reduces a real symmetric matrix to a symmetric tridiagonal form using orthogonal similarity transformations.

The function performs a sequence of $(n-2)$ Householder reflections:

$$P_i = I - u_i u_i^T/h, \text{ where } h = u_i^T u_i/2$$

The sequence is applied to the input matrix row-wise, starting with the last row and continuing from bottom to top:

$$A_i = P_i A_{i-1} P_i, i \in [1, n-2]$$

Post-multiplications annihilates the elements in the $(n-i)^{\text{th}}$ row (rows are numbered starting with 0 from top to bottom) to the left from the principal subdiagonal. Pre-multiplications would annihilate the $(n-i)^{\text{th}}$ column (columns are numbered in similar manner) elements, but they are not actually applied.

The calculations proceed as follows. At first, the elements in the $(n-i)^{\text{th}}$ row to the left from the principal subdiagonal are scaled to avoid possible underflow that would result in destroying the orthogonality of the transformation. The sum of the squares S_i of the scaled elements is taken as the square of the i^{th} subdiagonal element of the symmetric tridiagonal form. A square root $S_i^{1/2}$ with the sign set opposite to a transformed subdiagonal element of the original matrix a is then taken as the value of this subdiagonal element.

At last, the i^{th} reflection eliminates these elements in the $(n-i)^{\text{th}}$ row. The symmetric elements in the corresponding column of the matrix would also be eliminated if the pre-multiplication were applied.

Function tred1 references only the *lower* triangle of the input matrix and the diagonal and subdiagonal elements of the resulted symmetric tridiagonal matrix. The transformed diagonal and subdiagonal elements are stored in the vectors d and e, therefore preserving the full upper triangle of the matrix a.

The information about the Householder transformations used in the reduction process is stored in the *strict lower* triangle of the original matrix a. The rest of the information is represented by the elements of the output vector e.

Having returned from the function, the output vectors d and e contain respectively the diagonal and subdiagonal elements of resulted symmetric tridiagonal matrix. The subdiagonal elements has been stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array has been set to 0.

The array $e2$ contains the squares of the corresponding subdiagonal elements of resulted symmetric tridiagonal matrix in its last $(n-1)$ locations. The entry `e2[0]` has been set to 0.

Performance

The whole reduction (taking into account the symmetry property) is required $\frac{2}{3}n^3$ multiplication and n square roots.

The reduction process is numerically stable because it is based on orthogonal transformations. The resultant symmetric tridiagonal matrix is orthogonally similar to a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of the original matrix.

Notes

Matrix a is expected to be in the format produced by the allocation functions `fsquare`, `trngl_fmatrix` or `fsym`. If the matrix is allocated by the function `trngl_fmatrix` the *lower* triangle must be specified.

Arrays e and $e2$ may coincide if the squares are not required.

Test

The test program is contained in `rsmt.c` file. The example matrices are contained in `rsymmet1.xpd, rsymmet2.xpd, rsymmet3.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

R. S. Martin, C. Reinsch, J. H. Wilkinson, Numerical Mathematics, Vol. 11, pp.181-195, 1968.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

6.1.2 TRED2 Reduction and Accumulation of Transformations

Name

stred2 — Reduction and accumulation of the transformations, single precision
dtred2 — Reduction and accumulation of the transformations, double precision

Synopsis

```
#include <ceispack.h>
void stred2 (n, a, d, e, z, rv)
int n;
float **a, *d, *e, **z;
float *rv;

void dtred2 (n, a, d, e, z, rv)
int n;
double **a, *d, *e, **z;
double *rv;
```

n	— order of input Hermitian/output symmetric tridiagonal matrices
a	— symmetric input matrix (lower triangle)
d	— output vector of length n containing diagonal elements of output matrix
e	— output vector of length n containing subdiagonal elements of output matrix
z	— square output matrix containing transposed transformations
rv	— temporary storage array of length n

Description

Function tred2 reduces a real symmetric matrix to a symmetric tridiagonal matrix using orthogonal similarity transformations.

At first, the lower triangle of matrix a is copied into the lower triangle of output matrix z. The transformations are applied to the matrix z.

The function performs a sequence of $(n-2)$ Householder reflections:

$$P_i = I - u_i u_i^T/h, \text{ where } h = u_i^T u_i/2$$

The sequence is applied to the matrix row-wise, starting with the last row and continues from bottom to top:

$$A_i = P_i A_{i-1} P_i, i \in [1, n-2]$$

Post-multiplications eliminates the elements in the $(n-i)^{\text{th}}$ row (rows are numbered starting with 0 from top to bottom) to the left from the principal subdiagonal. Pre-multiplications would annihilate the $(n-i)^{\text{th}}$ column (columns are numbered in similar manner) elements, but they are not actually applied.

The calculations proceed as follows. The elements in the $(n-i)^{\mathrm{th}}$ row to the left from the principal subdiagonal are scaled to avoid possible underflow that would result in destroying the orthogonality of the transformation. The sum of the squares S_i of the scaled elements is taken as the square of the i^{th} subdiagonal element of the symmetric tridiagonal form. A square root $S_i^{1/2}$ with the sign set opposite to a transformed subdiagonal element of the original matrix a is taken then as the value of this subdiagonal element.

At last, the i^{th} reflection eliminates these elements in the $(n-i)^{\mathrm{th}}$ row. The symmetric elements in the corresponding column of the matrix would also be eliminated if the pre-multiplication were applied.

Function tred2 references only the *lower* triangle of the input matrix and the diagonal and subdiagonal elements of the resulted symmetric tridiagonal matrix.

The transformations used in the reduction process are accumulated transposed in the output matrix z by forming the product of $(n-2)$ Householder reflections.

Having returned from the function, the output vectors d and e contain respectively the diagonal and subdiagonal elements of resulted symmetric tridiagonal matrix. The subdiagonal elements has been stored in tha last $(n-1)$ locations of the vector e. The first entry of the array has been set to 0.

Performance

The whole reduction (taking into account the symmetry property) irequires $\frac{2}{3}n^3$ multiplications and n square roots.

The reduction process is numerically stable because it is based on orthogonal transformations. The resultant symmetric tridiagonal matrix is orthogonally similar to a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of the original matrix.

Notes

Matrix a is expected to be in the format produced by the allocation functions `fsquare`, `trngl_fmatrix` or `fsym`. If the matrix is allocated by the function `trngl_fmatrix` the *lower* triangle must be specified. If the matrix a is allocated by the function `fsquare`, it may coincide with the square array z.

Square array z is expected to be in the format produced by the allocation function `fsquare`.

The output matrix a is unchanged, unless it coincide with the matrix z.

Test

The test programs are contained in `rstest.c, rsgt.c, rsgabt.c, rsgbat.c` files. The example matrices are contained in `rgensym.xpd, rsymmet1.xpd, rsymmet2.xpd, rsymmet3.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

R. S. Martin, C. Reinsch, J. H. Wilkinson, Numerical Mathematics, Vol. 11, pp.181-195, 1968.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

6.2 Eigenvectors

6.2.1 TRBAK1 Eigenvectors of Original Matrix

Name

strbak1 — Forms eigenvectors of symmetric matrix, single precision
dtrbak1 — Forms eigenvectors of symmetric matrix, double precision

Synopsis

```
#include <ceispack.h>
void strbak1 (n, a, e, m, z)
int n;
float **a, *e;
int m;
float **z;

void dtrbak1 (n, a, e, m, z)
int n;
double **a, *e;
int m;
double **z;
```

n — order of input matrices, number of columns in eigenvector matrix
a — orthogonal transformation input matrix
e — length n input array of subdiagonals of reduced symmetric tridiagonal matrix
m — number of eigenvectors to be back transformed
z — input/output matrix containing m transposed (row) eigenvectors

Description

Function trbak1 computes the eigenvectors of a real symmetric matrix by back transforming the eigenvectors of the corresponding symmetric tridiagonal matrix determined by the function `tred1`.

The function calculates the matrix product PZ, where P is the product of $(n-2)$ Householder reflections, accumulated in the reduction of a real symmetric matrix A to a symmetric tridiagonal form T:

$$T = PAP$$

by the function `tred1`.

The input matrix a contains information about orthogonal transformation used in the reduction by the function `tred1` in its strict lower triangle.

The input vector e contains the subdiagonal elements of the derived symmetric tridiagonal matrix in its *last* $(n-1)$ locations. The entry `e[0]` is arbitrary.

Input matrix z contains the m transposed (row) eigenvectors to be back transformed.

Having returned from the function, matrix z contains transformed eigenvectors of the symmetric matrix. They are stored row-wise in place of the eigenvectors of derived symmetric tridiagonal matrix. Since the transformations used by the function trbak1 are orthogonal, the Euclidean norm of the eigenvectors is kept unchanged.

Performance

The computed eigenvectors are close to the exact eigenvectors of a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of the original matrix.

Notes

Matrix a is expected to be in the format produced by the allocation functions `fsquare`, `trngl_fmatrix` or `fsym`. If the matrix is allocated by the function `trngl_fmatrix` the *lower* triangle must be specified.

Rectangular $m \times n$ array z is expected to be in the format produced by the allocation function `fmatrix`. The number of rows in the array must not be less than m.

Test

The test program is contained in `rsmt.c` file. The example matrices are contained in `rsymmet1.xpd, rsymmet2.xpd, rsymmet3.xpd` files.

References

R. S. Martin, C. Reinsch, J. H. Wilkinson, Numerical Mathematics, Vol. 11, pp.181-195, 1968.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

Chapter 7

Symmetric Band Eigenproblem

7.1 Reduction to Symmetric Tridiagonal Form

7.1.1 BANDR Reduction and Accumulation of Transformations

Name

sbandr — Reduction and accumulation of transformations, single precision
dbandr — Reduction and accumulation of transformations, double precision

Synopsis

```
#include <ceispack.h>
void sbandr (n, mb, a, d, e, e2, matz, z)
int n, mb;
float **a, *d, *e, *e2;
int matz;
float **z;

void dbandr (n, mb, a, d, e, e2, matz, z)
int n, mb;
double **a, *d, *e, *e2;
int matz;
double **z;
```

n — order of input band symmetric/output symmetric tridiagonal matrices
n — order of input/output matrices
mb — number of subdiagonals in input matrix
a — band symmetric input matrix
d — output vector of length n containing diagonal elements of output matrix
e — output vector of length n containing subdiagonal elements of output matrix
e2 — length n output array of squares of subdiagonal elements of output matrix
matz — input flag which causes accumulation of transformations
z — output square matrix containing transposed transformations

Description

Function bandr reduces a real symmetric band matrix to a symmetric tridiagonal form using and optionally accumulating the orthogonal similarity transformations.

The original matrix A is reduced to a symmetric tridiagonal form T by a sequence of fast Givens rotations:

$$T = Q^T A Q$$

where Q is the product of elementary plane rotation matrices.

The reduction process begins with the first column of the original matrix. The non-zero matrix elements in the column below the main subdiagonal are eliminated from bottom to top, starting with the element in last non-zero subdiagonal and the process continues successively on the next columns.

Each rotation annihilating an element inside the band simultaneously introduces a non-zero element outside the band. This element is eliminated by a sequence of $\frac{n-i-k}{m_b}$ additional rotations, while the $a_{i+k,i}$ matrix element is eliminated.

The process of reduction is illustrated for a matrix with $m_b = 4$. Only the full lower part of band takes part in the computations:

$$\begin{array}{ccccccc}
\times & & & & & & \\
\times & \times & & & & & \\
(2) & \times & \times & & & & \\
(1) & (4) & \times & \times & & & \\
 & (3) & (6) & \times & \times & & \\
 & & (5) & (8) & \times & \times & \\
 & & & (7) & (10) & \times & \times \\
 & & & & \cdots & \cdots & \cdots
\end{array}$$

The off-band non-zero elements that appear in the reduction are not shown.

If specified by the flag *matz*, the transformations used in the reduction are accumulated in the output array *z*.

On input, matrix *a* contains symmetric band matrix to be reduced.

The input parameter *mb* specifies the number of non-zero subdiagonals below the principal diagonal of the original matrix. The total number of diagonals in the original matrix is equal to $(2m_b + 1)$.

The input flag *matz* specifies whether the transformations should accumulated or not. It should be set to any non-zero value if the transformation matrix is to be accumulated, and to 0 otherwise.

Having returned from the function, matrix *a* has been modified. Its principal diagonal and the first subdiagonal contain a copy of the reduced symmetric tridiagonal matrix.

The output vectors *d* and *e* contain respectively the diagonal and subdiagonal elements of resultant symmetric tridiagonal matrix. The subdiagonal elements has been stored in the *last* $(n - 1)$ locations of the vector *e*. The first entry of the array has been set to 0.

The array *e*2 contains the squares of the corresponding subdiagonal elements of resulted symmetric tridiagonal matrix in its last $(n - 1)$ locations. The entry `e2[0]` has been set to 0.

If specified by the input flag *matx* matrix *z* contains the orthogonal transformations matrix accumulated in the reduction process.

Performance

The number of rotations required to reduce the band-width from m_b to $(m_b - 1)$ is about $\frac{n^2}{2m_b}$ and this involves about $4(m_b + 1)n^2/m$ multiplications.
The reduced matrix is orthogonally similar to a matrix that is close to the original matrix.

Notes

Matrix *a* is expected to be in the format produced by the allocation functions `trngl_band_fmatrix` or `fsymband`. If the matrix is allocated by the function `trngl_band_fmatrix` the *lower* part of band must be specified.

Array *z* is expected to be in the format produced by the allocation function `fsquare`.

If `matz` has been set to 0 then the matrix *z* is not referenced by the function.

Test

The test program is contained in `rsbt.c` file. The example matrices are contained in `bsm01.xpd, bsm02.xpd` and `bsm03.xpd` files.

References

H. R. Schwarz, Communications of the ACM, Vol. 6, pp.315-316, 1963.

H. R. Schwarz, Numerical Mathematics, Vol. 12, pp.231-241, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

W. M. Gentleman, Journal Inst. Maths. Applic., Vol. 12, pp.329-336, 1973.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

7.1.2 BQR Eigenvalue of Smallest Magnitude, QR Algorithm

Name

sbqr — Forms eigenvalue of smallest magnitude, single precision
dbqr — Forms eigenvalue of smallest magnitude, double precision

Synopsis

```
#include <ceispack.h>
void sbqr (n, mb, a, t, r, ierr, rv)
int n, mb;
float **a, *t, *r;
int *ierr;
float *rv;

void dbqr (n, mb, a, t, r, ierr, rv)
int n, mb;
double **a, *t, *r;
int *ierr;
double *rv;
```

n — order of input matrix
mb — number of sub and super diagonals in input matrix
a — band symmetric input matrix
t — address of input constant containing shift value/output eigenvalue
r — address of input zero constant/output tolerance value
ierr — address of output variable containing error completion code
rv — temporary storage array of length at least $2(m_b + 1)^2 + 4(m_b + 1) - 3$

Diagnostics

Function bqr returns the output parameter `*ierr` set to 0 for normal return or to n if the eigenvalue has not been determined after `MAXITER` iterations.

Description

Function bqr computes the smallest magnitude eigenvalue of a real band symmetric matrix using the QR algorithm with shifts of origin.

The function can be applied to find the eigenvalue of smallest magnitude of the matrix $A + tI$, where t is a constant value and I is the identity matrix. Therefore the eigenvalue of A is found which is nearest to the constant t.

The QR algorithm iterates a sequence of band symmetric matrices, which are orthogonally similar to the original matrix with a shifted origin. The sequence converges to a matrix having last row and column of zero. Matrix bandwidth is not widened during the iterations.

The shift of origin is postponed until the sum of the absolute values of the off-diagonal elements of the last row (and column) of the original matrix becomes less than the one quarter of the tolerance quantity r. Such a strategy is applied to ensure the computation of the required nearest to zero eigenvalue and keep the rate of convergence of the iteration process as fast as possible.

On return, the tolerance value r is reset to the maximum of the sum of the absolute values of the off-diagonal elements of the current last row and the current value of r. If the matrix has elements of widely varying magnitudes, it is recommended to place the largest entries in the upper left corner of the matrix, thus allowing the criterion of shifting to be more effective.

The shift of origin is taken as the eigenvalue of the lowermost 2×2 principal minor, closer to the second diagonal element of the minor. When the last row (and the last column) becomes negligible, then the sum of the origin shifts is added to the input value of t and the function returns this number as the desired eigenvalue of the matrix $A + tI$. The matrix is deflated by purging (zeroing) its last row.

Having returned from the function, the matrix a contains the transformed band symmetric matrix. The matrix $A' + t'I$, derived from the output parameters, is similar to the input $A + tI$ to within rounding errors. Its last row and column are null if the output flag `*ierr` is zero. The variable `*t` contains the computed eigenvalue of $A + tI$ if `*ierr` is zero. The output value of `*r` contains the maximum of its input value and the norm of the last row of the input matrix a.

Function bqr can be invoked subsequently to find the next eigenvalue nearest to the previous one. In such a case the output values of variables `*r, *t` and deflated matrix a should be passed to the function. The order of matrix n should be decreased by 1, neglecting the situation when it becomes even smaller than the bandwidth m_b of the original matrix. The bandwidth value mb parameter should not be modified on subsequent calls.

To find the eigenvectors corresponding to the computed eigenvalues, the function bandv should be invoked. The original matrix a and the eigenvalue in t should be preserved for later use by the function bandv.

The input parameter mb specifies the number of non-zero subdiagonals below the principal diagonal of the original matrix. The total number of diagonals in the original matrix is equal to $(2m_b + 1)$.

Performance

Appropriate shift of the origin makes the rate of convergence cubic in almost all the cases.

The computed eigenvalue is exact for a perturbed original matrix. The value of the perturbation is proportional to the machine precision times the norm of the original matrix. The algorithm is not guaranteed a high relative accuracy for the small eigenvalues of a band symmetric matrix, although if the row norms of the matrix decrease from top to bottom of the matrix, then such a small eigenvalues would be computed having low relative errors.

Notes

Matrix *a* is expected to be in the format produced by the allocation functions `trngl_band_fmatrix` or `fbandsym`. If the first function is being used, the *lower* part of band must be specified.

The input matrix is modified by the function bqr.

Function bqr references only the non-zero lower part of the band and only the lower non-zero part of half band width of the matrix need be supplied. Other elements of the matrix are not referenced. On a subsequent call, its output contents from the previous call should be passed to the function.

The variable `*r` should be set to zero on the first call and as its output value from the previous call on a subsequent call.

Test

The test program is contained in `bqrt.c` file. The example matrices are contained in `bsm01.xpd, bsm02.xpd` and `bsm03.xpd` files.

References

R. S. Martin, J. H. Wilkinson, C. Reinsch, Numerical Mathematics, Vol̇16, pp.85-93, 1970.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

7.2 Eigenvectors

7.2.1 BANDV Eigenvectors Corresponding to Specified Eigenvalues

Name

sbandv — Forms eigenvectors or solves linear equations, single precision
dbandv — Forms eigenvectors or solves linear equations, double precision

Synopsis

```
#include <ceispack.h>
int sbandv (n, mb, a, e21, m, w, z, index, rv, rv6)
int n, mb, m;
float e21, **a, *w, **z;
int *index;
float *rv, *rv6;

int dbandv (n, mb, a, e21, m, w, z, index, rv, rv6)
int n, mb, m;
double e21, **a, *w, **z;
int *index;
double *rv, *rv6;
```

n — order of input matrix and number of columns in matrix z
mb — number of sub/super diagonals in input matrix
a — band symmetric/non-symmetric input matrix
e21 — input flag specifying ordering of eigenvalues/symmetry of coefficient matrix
m — input variable which specifies number of eigenvalues/linear systems
w — input array containing m eigenvalues/constant values
z — input row constant vector/output row solution vector/eigenvector matrix
index — address of output variable containing error completion code
rv — temporary storage array of length $n(2m_b + 1)$
rv6 — temporary storage array of length n

Diagnostics

The function returns 0 for normal return or 1 if an error exit has been made. The output error flag `*index` is set to:

- 0 for normal return.
- $-i$ if the eigenvector corresponding to the i^{th} eigenvalue failed to converge, or if the i^{th} system of linear equations is close to be singular.

Description

Function bandv computes the eigenvectors of a real band symmetric matrix which correspond to specified the eigenvalues, using the inverse iteration technique.

The function can also be used to solve systems of linear equations

$$(A - w_i I)x_i = z_i \text{ where } i \in [0, m-1]$$

with a band symmetric or non-symmetric coefficient matrix. Matrix I is the identity matrix, and w_r are constant values.

When function bandv is applied to find the eigenvectors of a real band symmetric matrix, the computation proceed as follows. At first, the original matrix is modified by shifting its diagonal elements by the negative of the i^{th} eigenvalue w_i. Next, the LU decomposition of the modified matrix is calculated by Gaussian elimination with partial pivoting. The upper triangular matrix U of the decomposition is saved in the array *rv*. Starting from an initial vector, the approximate eigenvector is obtained by the back substitution process. Its norm is compared with the matrix norm to test if the growth of the norm is large enough to terminate process and accept the vector to be an approximate eigenvector. If the vector is accepted, its l_2 norm is set equal to 1. Up to n orthogonal initial vectors are attempted in order to obtain an appropriate norm growth. If no vector is accepted, an error flag is set and bandv continues to iterate the next eigenvector. The components of unaccepted eigenvector are set to zero. If such a situation occurs more than once, the output error flag `*ierr` would contain the last index of the eigenvalue for which the computation of the eigenvector fails.

The eigenvectors corresponding to separated eigenvalues would be computed orthogonal, while eigenvectors corresponding to a close or multiple eigenvalues may not be satisfactory orthogonal. To ensure the orthogonal set of the eigenvectors, each approximate vector corresponding to an eigenvalue of a group of close eigenvalues is orthogonalized with respect to earlier computed eigenvectors of the group. The test for norm growth is performed after the orthogonalization. Eigenvalues identical within the machine precision are perturbed by multiplies of the relative machine precision in order to obtain linearly independent eigenvectors. The perturbed values are not stored to the eigenvalue array w.

When the function bandv is applied to find a solution of a linear systems with a band symmetric or non-symmetric coefficient matrix, the above steps are performed except for the vector orthogonalization procedure and growth test. Having returned from bandv, the determinant

$$\det(A - w_{m-1}I)$$

can be calculated within sign precision, as the product of the first n elements of vector rv.

The input constant *mb* contains the number m_b of diagonals below or above the main diagonal of the input band matrix.

The input flag $e21$ specifies the ordering of the eigenvalues and should be set to:

0 if the eigenvalues are stored in ascending order.

2 if the eigenvalues are stored in descending order.

If the function is being applied to solve systems of linear equations, $e21$ should be set to:

1 if the coefficient matrix is symmetric.

-1 if the coefficient matrix is non-symmetric.

The input vector w contains m eigenvalues of the original matrix stored in ascending or descending order. In general, the absolute error in the eigenvalues should be on order of the product of the relative machine precision times the norm of the original matrix. Some matrices require more precision in the eigenvalues, sometimes as small as the relative machine precision times the eigenvalue of the smallest magnitude.

If the function is applied to solve systems of linear equations the input array w contains m constant values. Usually they are zeros.

If the function is being applied to solve systems of linear equations, then the input matrix z contains m transposed (row) constant vectors.

Having returned from the function, matrix z contains the associated set of transposed (row) orthogonal eigenvectors. Any vector which failed to converge is set to zero. If the function is applied to solve systems of linear equations, z contains the associated set of transposed (row) solution vectors.

Performance

The rate of convergence of the inverse iterations is linear.

The computed eigenvectors (and solution vectors) are close to the exact ones of a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of the original matrix.

Notes

Band symmetric matrix a is expected to be in the format produced by the allocation functions `trngl_band_fmatrix` or `fbandsym`. If the first function is being used, the *lower* part of band must be specified. Only the lower part (half band width) of the input matrix need be supplied.

Band non-symmetric matrix a is expected to be in the format produced by the allocation functions `band_fmatrix` or `fband`. The number of sub- and super-diagonals must be the same.

In both the cases the total number of diagonals in the input matrix is equal to $(2m_b + 1)$.

General $m \times n$ matrix z is expected to be in the format produced by the allocation function `fmatrix`.

The input matrix a and vector w are preserved by the function bandv.

Test

The test program for eigenvectors computation is contained in `bqrt.c` file. The example matrices are contained in `bsm01.xpd, bsm02.xpd, bsm03.xpd` files.

The test programs for solving of linear systems is contained in `bandvt.c` file. The example linear systems are contained in `bls01.xpd, bls02.xpd, bls03.xpd` files.

References

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

Chapter 8

Real Special Tridiagonal Eigenproblem

8.1 Reduction to Symmetric Tridiagonal Form

8.1.1 FIGI Reduction by Diagonal Transformations

Name

sfigi — Reduction to symmetric tridiagonal form, single precision
dfigi — Reduction to symmetric tridiagonal form, double precision

Synopsis

```
#include <ceispack.h>
void sfigi (n, a, d, e, e2, ierr)
int n;
float **a, *d, *e, *e2;
int *ierr;

void dfigi (n, a, d, e, e2, ierr)
int n;
double **a, *d, *e, *e2;
int *ierr;
```

n	— order of input special/output symmetric tridiagonal matrix
a	— special tridiagonal input matrix
d	— output vector of length n containing diagonal elements of output matrix
e	— output vector of length n containing subdiagonal elements of output matrix
e2	— length n output array of squares of subdiagonal elements of output matrix
ierr	— output variable address containing error completion code

Diagnostics

The output error flag *ierr is set to:

- 0 for normal return.
- $(n+i)$ if a product $a_{i,i-1}a_{i-1,i}$ is negative.
- $-(3n+i)$ if a product $a_{i,i-1}a_{i-1,i}$ is zero with one factor non-zero. In such a case the eigenvectors of reduced symmetric tridiagonal matrix are not simply related to the eigenvectors of original matrix and should not be computed.

Description

Function figi reduces a special tridiagonal matrix to symmetric tridiagonal form using diagonal similarity transformations.

If the original tridiagonal matrix has its elements such that the product of the corresponding off-diagonal entries is positive:

$$a_{i,i-1}a_{i-1,i} > 0, \text{ where } i \in [1, n-1]$$

and this product is equal to zero if and only if both the multipliers are zero, then the matrix can be reduced to symmetric tridiagonal form by the non-singular diagonal similarity transformation:

$$T = D^{-1}AD$$

The diagonal elements of the derived symmetric tridiagonal matrix are:

$$t_{ii} = a_{ii}, \text{ where } i \in [0, n-1]$$

and its subdiagonal elements are:

$$t_{i,i-1} = \sqrt{a_{i,i-1}a_{i-1,i}}, \text{ where } i \in [1, n-1]$$

The diagonal transformation is defined by the following formuli when both subdiagonal and superdiagonal entries are not zero:

$$d_{00} = 1 \text{ and } d_{ii} = d_{i-1,i-1}\sqrt{a_{i-1,i}/a_{i,i-1}}$$

In the case where both elements $a_{i-1,i}$ and $a_{i,i-1}$ are equal to zero, the corresponding element of the diagonal transformation matrix is:

$$d_{ii} = 1$$

If the product of a pair of corresponding off-diagonal elements is zero, while one of the multipliers is not equal to zero, then the resultant symmetric tridiagonal matrix still has the same eigenvalues as the original matrix, but the eigenvectors of the transformed matrix cannot be obtained easily by back transforming those of corresponding symmetric tridiagonal matrix.

On return from the function the output vectors d and e contain the diagonal and subdiagonal elements of the resultant symmetric tridiagonal matrix. The subdiagonal elements has been stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array has been set to 0.

The array $e2$ contains the squares of the corresponding subdiagonal elements of the resultant symmetric tridiagonal matrix in its last $(n-1)$ locations. The entry `e2[0]` has been set to 0.

Performance

The reduced matrix is similar to a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Matrix a is in the format produced by the allocation functions `band_fmatrix, fband` or `f3diag`.

Arrays e and $e2$ may coincide if the squares are not required.

The input matrix a is preserved by the function.

Test

The test program is contained in `bakvect.c` file. The example matrices are contained in `trispc1.xpd, trispc2.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

8.1.2 FIGI2 Reduction and Accumulation of Transformations

Name

sfigi2 — Reduction and accumulation of transformations, single precision
dfigi2 — Reduction and accumulation of transformations, double precision

Synopsis

```
#include <ceispack.h>
void sfigi2 (n, a, d, e, z, ierr)
int n;
float **a, *d, *e, **z;
int *ierr;

void dfigi2 (n, a, d, e, z, ierr)
int n;
double **a, *d, *e, **z;
int *ierr;
```

n — order of input/output matrices
a — special tridiagonal input matrix
d — output vector of length n containing diagonal elements of output matrix
e — output vector of length n containing subdiagonal elements of output matrix
z — square output matrix containing transposed transformations
ierr — output variable address containing error completion code

Diagnostics

The output error flag `*ierr` is set to:

- 0 for normal return.
- $(n + i)$ if a product $a_{i,i-1}a_{i-1,i}$ is negative.
- $(2n + i)$ if a product $a_{i,i-1}a_{i-1,i}$ is zero with one factor non-zero.

Description

Function figi2 reduces a special tridiagonal matrix to symmetric tridiagonal form by using and accumulating diagonal similarity transformations.

If the original tridiagonal matrix has its elements such that the product of the corresponding off-diagonal entries is positive:

$$a_{i,i-1}a_{i-1,i} > 0, \text{ where } i \in [1, n-1]$$

and this product is equal to zero if and only if both the multipliers are zero, then the matrix can be reduced to symmetric tridiagonal form by the non-singular diagonal similarity transformation:

$$T = D^{-1}AD$$

The diagonal elements of the derived symmetric tridiagonal matrix are:

$$t_{ii} = a_{ii}, \text{ where } i \in [0, n-1]$$

and its subdiagonal elements are:

$$t_{i,i-1} = \sqrt{a_{i,i-1}a_{i-1,i}}, \text{ where } i \in [1, n-1]$$

The diagonal transformation is defined by the following formuli when both subdiagonal and superdiagonal entries are not zero:

$$d_{00} = 1 \text{ and } d_{ii} = d_{i-1,i-1}\sqrt{a_{i-1,i}/a_{i,i-1}}$$

In the case where both the elements $a_{i-1,i}$ and $a_{i,i-1}$ are equal to zero, the corresponding element of diagonal transformation matrix is:

$$d_{ii} = 1$$

If the product of a pair of corresponding off-diagonal elements is zero, while one of the multipliers is not equal to zero, then the resultant symmetric tridiagonal matrix still has the same eigenvalues as the original matrix, but the eigenvectors of the transformed matrix cannot be obtained easily by back transforming those of corresponding symmetric tridiagonal matrix.

The diagonal similarity transformations used in the reduction are accumulated in the diagonal entries of the output matrix z for later use in eigenvector computations.

On return from the function the output vectors d and e contain respectively the diagonal and subdiagonal elements of the resulted symmetric tridiagonal matrix. The subdiagonal elements has been stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array has been set to 0.

The array $e2$ contains the squares of the corresponding subdiagonal elements of the resultant symmetric tridiagonal matrix in its last $(n-1)$ locations. The entry `e2[0]` has been set to 0.

Matrix z contains the diagonal transformations.

Performance

The reduced matrix is similar to a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Matrix a is in the format produced by the allocation functions `band_fmatrix, fband` or `f3diag`.

Arrays e and $e2$ may coincide if the squares are not required.

Matrix z is in the format produced by the allocation function `fsquare`.

The input matrix a is preserved by the function.

Test

The test program is contained in `rtt.c` file. The example matrices are contained in `trispc1.xpd, trispc2.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

8.2 Eigenvectors

8.2.1 BAKVEC Eigenvectors of Original Matrix

Name

sbakvec — Forms eigenvectors of original matrix, single precision
dbakvec — Forms eigenvectors of original matrix, double precision

Synopsis

```
#include <ceispack.h>
void sbakvec (n, a, e, m, z, ierr)
int n;
float **a, *e;
int m;
float **z;
int *ierr;

void dbakvec (n, a, e, m, z, ierr)
int n;
double **a, *e;
int m;
double **z;
int *ierr;
```

n — order of input/output matrices
a — special tridiagonal input matrix
e — length n input array of subdiagonals of reduced symmetric tridiagonal matrix
m — number of eigenvectors to be back transformed
z — input/output matrix containing m transposed (row) eigenvectors
ierr — output variable address containing error completion code

Diagnostics

The output error flag `*ierr` is set to:

- 0 for normal return.

- $(2n + i)$ if e_i is zero when matrix elements $a_{i,i-1}$ or $a_{i-1,i}$ is not zero. In such a case the symmetric matrix is not similar to the original matrix, and the eigenvectors cannot be found by the function bakvec.

Description

Function bakvec computes the eigenvectors of the non-symmetric tridiagonal matrix by back transforming the eigenvectors of the corresponding symmetric matrix determined by the function `figi`.

If a non-symmetric tridiagonal matrix A has been transformed to a symmetric tridiagonal matrix T using the diagonal similarity transformation D:

$$T = D^{-1}AD,$$

where the diagonal transformation matrix D has been defined by the function `figi`, then the function bakvec transforms the eigenvectors of the transformed matrix T into those of the original matrix by computing the product:

$$X_i = DZ_i, \text{ where } i \in [0, m-1].$$

The back transformed eigenvectors overwrite the eigenvectors of the symmetric tridiagonal matrix.

The input vector e contains the subdiagonal elements of the derived symmetric tridiagonal matrix in its *last* $(n-1)$ locations. The entry `e[0]` is arbitrary.

The input matrix z contains the eigenvectors to be back transformed in its first m rows.

On return from the function the matrix z contains the transformed eigenvectors in its first m rows.

Performance

The computed eigenvectors are close to the exact eigenvectors of a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Matrix a is in the format produced by the allocation functions `band_fmatrix, fband` or `f3diag`.

The rectangular $m \times n$ matrix z is in the format produced by the allocation function `fmatrix`. The number of rows in the matrix must not be less than m.

The matrix a is preserved by the function.

The function bakvec modifies the vector e.

Test

The test program is contained in `bakvect.c` file. The example matrices are contained in `trispc1.xpd, trispc2.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

Chapter 9

Symmetric Tridiagonal Eigenproblem

9.1 Sturm Sequence

9.1.1 NEIGN3 Number of Eigenvalues in an Interval

Name

sneign3 — Determines number of eigenvalues that lie in a specified interval, single precision
dneign3 — Determines number of eigenvalues that lie in a specified interval, double precision

Synopsis

```
#include <ceispack.h>
int sneign3 (n, d, e, e2, lb, ub)
int n;
float *d, *e, *e2, lb, ub;

int dneign3 (n, d, e, e2, lb, ub)
int n;
double *d, *e, *e2, lb, ub;
```

n — order of symmetric tridiagonal input matrix
d — input vector of length n containing diagonal elements of input matrix
e — input vector of length n containing subdiagonal elements of input matrix
e2 — length n input array of squares of subdiagonal elements of input matrix
lb — input constant specifying lower bound of interval
lu — input constant specifying upper bound of interval

Diagnostics

The function returns number of eigenvalues that lie in the specified interval $[l_b, u_b)$. If input values $l_b \geq u_b$ then the function returns 0.

Description

Function neign3 estimates the number of eigenvalues which belong to the specified interval $[l_b, u_b)$.

The calculations proceed as follows. First the function determines negligible subdiagonal elements of the original matrix and sets to zero the corresponding entries of array $e2$, the original matrix is split into the direct sum of submatrices.

The Sturm sequence gives the number or eigenvalues of a symmetric tridiagonal matrix that are less than the specified value x:

$$\begin{cases} q_0(x) = c_0 - x \\ q_i(x) = (c_i - x) - b_i^2/q_{i-1}(x),\ i \in [1, n-1] \end{cases}$$

equal to the number of negative values in the sequence of n polynomials $q_i(x)$. Above factors c_i and b_i are the diagonals and subdiagonals of the matrix respectively . In the computation process any value of the polynomial $q_{i-1}(x)$ may be zero. If this situation occurs zeros may be replaced by the value of the relative machine precision.

At the next step, the function calculates the Sturm sequence at both the interval boundaries and estimates the number of eigenvalues within the interval as a difference between the values of Sturm sequences at the upper and lower bounds u_b and l_b.

The input constants *lb* and *ub* determine the interval containing the eigenvalues the number of which is to be determined.

The input vectors d and e contain diagonal and subdiagonal elements of the original matrix. The subdiagonal elements should be stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array is arbitrary.

The input vector $e2$ contains the squares of the subdiagonal elements of the original matrix in its *last* $(n-1)$ locations. The entry `e2[0]` is arbitrary. On return from the function the entries of vector $e2$ corresponding to the negligible subdiagonal elements of input matrix are set to zero. The entry `e2[0]` is also set to 0.

Performance

The computed number of eigenvalues is exact within the machine precision.

Notes

The input vectors of diagonal and subdiagonal elements d and e of the original matrix are preserved by the function neign3.

The function neign3 is the subset of the C function `tsturm`.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

9.2 Eigenvalues and Eigenvectors

9.2.1 TQL1 Eigenvalues, QL Algorithm

Name

stql1 — Forms eigenvalues by QL algorithm, single precision
dtql1 — Forms eigenvalues by QL algorithm, double precision

Synopsis

```
#include <ceispack.h>
int stql1 (n, d, e, index)
int n;
float *d, *e;
int *index;

int dtql1 (n, d, e, index)
int n;
double *d, *e;
int *index;
```

n — order of symmetric tridiagonal input matrix
d — length n input vector of diagonal elements/output n ascending eigenvalues
e — length n input vector containing subdiagonal elements of input matrix
index — output variable address containing error completion code

Diagnostics

Function returns 0 for normal return or 1 if an error exit has been made. The output variable `*index` is set to:

- 0 for normal return.
- i if the i^{th} eigenvalue has not been determined after `MAXITER` iterations. The eigenvalues corresponding to indices $j \in [0, i-1]$ would be correct and ordered, but they may not be the smallest eigenvalues.

Description

Function tql1 computes all the eigenvalues of a real symmetric tridiagonal matrix using the QL algorithm with shift of origin.

The QL algorithm iterates the sequence of symmetric tridiagonal matrices that are orthogonally similar to the original matrix. The sequence converges to a diagonal matrix. A shift of origin is applied to improve the rate of convergence.

Before the next iteration the currently iterated submatrix is tested for possible splitting to submatrices. If the matrix is split only the uppermost submatrix takes part in the next iterations. The shift of origin on each iteration is taken to be the eigenvalue of the uppermost 2×2 principal minor, closer to the upper diagonal element of this minor. When the uppermost 1×1 principal minor has split from the rest of the matrix, its value is taken as an eigenvalue of the original matrix and the iterations continue on the remaining submatrix until the whole matrix finally split on minors of order 1. The tolerance of the test for splitting is commensurate to the relative machine precision.

The input vectors d and e contain diagonal and subdiagonal elements of the original matrix. The subdiagonal elements should be stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array is arbitrary.

The eigenvalues are ordered in ascending order as they are computed, and stored in the output vector d.

Performance

Appropriate shift of the origin makes the convergence of the algorithm global. The rate of convergence is cubic in almost all the cases.

The computed eigenvalues are close to those of the exact matrix. Also they are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is proportional to the machine precision times the norm of the original matrix at least in a case when the successive row norms are increasing from top to bottom of the original matrix.

Notes

If the original matrix has its row norms widely varying and not strictly decreasing downward, it is recommended to use implicit QL algorithm for eigenvalue computation or to permute the original matrix to obtain the successive row norms increasing from top to bottom of the matrix.

Test

The test program is contained in `tql1t.c` file. The example matrices are contained in `trisym1.xpd, trisym2.xpd, trisym3.xpd` and `trisym4.xpd` files.

References

H. Boulder, R. S. Martin, C. Reinsch, J. H. Wilkinson, Numerical Mathematics, Vol. 11, pp.293-306, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

9.2.2 IMTQL1 Eigenvalues, Implicit QL Algorithm

Name

simtql1 — Forms eigenvalues by implicit QL algorithm, single precision
dimtql1 — Forms eigenvalues by implicit QL algorithm, double precision

Synopsis

```
#include <ceispack.h>
int simtql1 (n, d, e, index)
int n;
float *d, *e;
int *index;

int dimtql1 (n, d, e, index)
int n;
double *d, *e;
int *index;
```

n	— order of symmetric tridiagonal input matrix
d	— length n input vector of diagonal elements/output n ascending eigenvalues
e	— length n input vector containing subdiagonal elements of input matrix
index	— output variable address containing error completion code

Diagnostics

Function returns 0 for normal return or 1 if an error exit has been made. The output variable `*index` is set to:

- 0 for normal return.
- i if the i^{th} eigenvalue has not been determined after `MAXITER` iterations. The eigenvalues corresponding to indices $j \in [0, i-1]$ would be correct and ordered, but they may not be the smallest eigenvalues.

Description

Function imtql1 computes all the eigenvalues of a real symmetric tridiagonal matrix using the QL algorithm with implicit shifts.

The implicit QL algorithm iterates a sequence of symmetric tridiagonal matrices that are orthogonally similar to the original matrix. The sequence converges to a diagonal matrix. A shift of origin is applied *implicitly* to improve the rate of convergence.

Before the next iteration the currently iterated submatrix is tested for possible splitting to submatrices. If the matrix split only the uppermost submatrix takes part in the next iterations. The shift of origin on each iteration is taken to be the eigenvalue of the uppermost 2×2 principal minor, closer to the upper diagonal element of this minor. When the uppermost 1×1 principal minor has split from the rest of the matrix, its value is taken as an eigenvalue of the original matrix and iterations continue on the remaining submatrix until the whole matrix finally split on minors of order 1. The tolerance of the test for splitting is commensurate to the relative machine precision.

The input vectors d and e contain diagonal and subdiagonal elements of the original matrix. The subdiagonal elements should be stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array is arbitrary.

The eigenvalues are ordered in ascending order as the are computed, and stored in the output vector d.

Performance

Appropriate shift of the origin makes the convergence of the algorithm global. The rate of convergence is cubic in almost all the cases.

The computed eigenvalues are close to those of the exact matrix. Also they are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is proportional to the machine precision times the norm of the original matrix.

As opposed to the QL algorithm with the explicit shifts, the implicit modification is not sensitive to the matrix structure with its row norms widely varying and not strictly decreasing downward.

Test

The test program is contained in `imtql1t.c` file. The example matrices are contained in `trisym1.xpd, trisym2.xpd, trisym3.xpd` and `trisym4.xpd` files.

References

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 12, pp.377-383, 1968.

A. Dubrulle, Numerical Mathematics, Vol. 15, p.450, 1970.
J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

9.2.3 IMTQLV Eigenvalues, Implicit QL Algorithm

Name

simtqlv — Forms eigenvalues and corresponding submatrix indices, single precision
dimtqlv — Forms eigenvalues and corresponding submatrix indices, double precision

Synopsis

```
#include <ceispack.h>
int simtqlv (n, d, e, e2, w, ind, index, rv)
int n;
float *d, *e, *e2, *w;
int *ind;
int *index;
float *rv;

int dimtqlv (n, d, e, e2, w, ind, index, rv)
int n;
double *d, *e, *e2, *w;
int *ind;
int *index;
double *rv;
```

n — order of symmetric tridiagonal input matrix

d — input vector of length n containing diagonal elements of input matrix

e — input vector of length n containing subdiagonal elements of input matrix

e2 — length n input array of squares of subdiagonal elements of input matrix

w — length n output array containing n eigenvalues in ascending order

ind — length n output array containing n submatrix indices of eigenvalues

index — output variable address containing error completion code

rv — temporary storage array of length n

Diagnostics

Function returns 0 for normal return or 1 if an error exit has been made. The output variable `*index` is set to:

- 0 for normal return.
- i if the i^{th} eigenvalue has not been determined after `MAXITER` iterations. The eigenvalues corresponding to indices $j \in [0, i-1]$ would be correct and ordered, but they may not be the smallest eigenvalues.

Description

Function imtqlv computes all the eigenvalues of a real symmetric tridiagonal matrix using the QL algorithm with implicit shifts and associates with the computed eigenvalues their corresponding submatrix indices.

The implicit QL algorithm iterates a sequence of symmetric tridiagonal matrices that are orthogonally similar to the original matrix. The sequence converges to a diagonal matrix. A shift of origin is applied *implicitly* to improve the rate of convergence.

Before the next iteration the currently iterated submatrix is tested for possible splitting to submatrices. If the matrix is split only the uppermost submatrix takes part in the next iterations. The shift of origin on each iteration is taken to be the eigenvalue of the uppermost 2×2 principal minor, closer to the upper diagonal element of this minor. When the uppermost 1×1 principal minor has split from the rest of the matrix, its value is taken as an eigenvalue of the original matrix and iterations continue on the remaining submatrix until the whole matrix finally split on minors of order 1. The tolerance of the test for splitting is commensurate to the relative machine precision.

The input vectors d and e contain diagonal and subdiagonal elements of the original matrix. The subdiagonal elements should be stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array is arbitrary.

The input array $e2$ contains the squares of the subdiagonal elements of the original matrix in its *last* $(n-1)$ locations. The entry `e2[0]` is arbitrary. On return from the function the entries of vector $e2$, corresponding to negligible subdiagonal elements of the original matrix are set to zero. The element `e2[0]` is also set to zero.

The eigenvalues are ordered in ascending order as the are computed, and stored in the vector w.

The output vector ind contains the submatrix indices associated with the corresponding eigenvalues in w array. An entry of array ind is set to 1 for eigenvalues belonging to the first submatrix from the top, 2 for those belonging to the second submatrix, etc.

Performance

Appropriate shift of the origin makes the convergence of the algorithm global. The rate of convergence is cubic in almost all the cases.

The computed eigenvalues are close to those of the exact matrix. Also they are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is proportional to the machine precision times the norm of the original matrix.

As opposed to the QL algorithm with the explicit shifts, the implicit modification is not sensitive to the matrix structure with its row norms widely varying and not strictly decreasing downward.

Notes

The input vectors of diagonal and subdiagonal elements d and e of the original matrix are preserved by the function imtqlv.

Test

The test programs are contained in `bakvect.c, rsmt.c, imtql1t.c, imtqlvt.c, tql1t.c, tqlratt.c` files. The example matrices are contained in `trisym1.xpd, trisym2.xpd, trisym3.xpd` and `trisym4.xpd` files.

References

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 12, pp.377-383, 1968.

A. Dubrulle, Numerical Mathematics, Vol. 15, p.450, 1970.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

9.2.4 TQLRAT Eigenvalues, Rational QL Algorithm

Name

stqlrat — Forms eigenvalues by rational QL algorithm, single precision
dtqlrat — Forms eigenvalues by rational QL algorithm, double precision

Synopsis

```
#include <ceispack.h>
int stqlrat (n, d, e, e2, index)
int n;
float *d, *e, *e2;
int *index;

int dtqlrat (n, d, e, e2, index)
int n;
double *d, *e, *e2;
int *index;
```

n	— order of symmetric tridiagonal input matrix
d	— length n input vector of diagonal elements/output n ascending eigenvalues
e	— length n input vector containing subdiagonal elements of input matrix
e2	— length n input array of squares of subdiagonal elements of input matrix
index	— output variable address containing error completion code

Diagnostics

Function returns 0 for normal return or 1 if an error exit has been made. The output variable `*index` is set to:

- 0 for normal return.
- i if the i^{th} eigenvalue has not been determined after `MAXITER` iterations. The eigenvalues corresponding to indices $j \in [0, i-1]$ would be correct and ordered, but they may not be the smallest eigenvalues.

Description

Function tqlrat computes all the eigenvalues of a real symmetric tridiagonal matrix using the rational QL algorithm with shift of origin.

The rational QL algorithm iterates a sequence of symmetric tridiagonal matrices that are orthogonally similar to the original matrix. The sequence converges to a diagonal matrix. A shift of origin is applied to improve the rate of convergence.

Before the next iteration the currently iterated submatrix is tested for possible splitting to submatrices. If the matrix is split only the uppermost submatrix takes part in the next iterations. The shift of origin on each iteration is taken to be the eigenvalue of the uppermost 2×2 principal minor, closer to the upper diagonal element of this minor. When the uppermost 1×1 principal minor has split from the rest of the matrix, its value is taken as an eigenvalue of the original matrix and the iterations continue on the remaining submatrix until the whole matrix finally split on minors of order 1. The tolerance of the test for splitting is commensurate to the relative machine precision.

The input vectors d and e contain diagonal and subdiagonal elements of the original matrix. The subdiagonal elements should be stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array is arbitrary.

On input the array $e2$ contains the squares of the subdiagonal elements of the original matrix in its *last* $(n-1)$ locations. The entry `e2[0]` is arbitrary.

On return from the function the elements of vector $e2$, corresponding to negligible subdiagonal elements of the original matrix are set to zero. The element `e2[0]` is also set to 0.

The eigenvalues are computed sequentially in ascending order and stored in the output vector d.

Performance

Appropriate shift of the origin makes the convergence of the algorithm global. The rate of convergence is cubic in almost all the cases.

The computed eigenvalues are close to those of the exact matrix. Also they are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is proportional to the machine precision times the norm of the original matrix at least in a case when the successive row norms are increase from top to bottom of the original matrix.

Notes

If the original matrix has its row norms widely varying and not strictly decreasing downward, it is recommended to use implicit QL algorithm for eigenvalue computation or to permute the original matrix to obtain the successive row norms increasing from top to bottom of the matrix.

Test

The test program is contained in `tqlratt.c` file. The example matrices are contained in `trisym1.xpd, trisym2.xpd, trisym3.xpd` and `trisym4.xpd` files.

References

C. Reinsch, Mathematical Computations, Vol. 25, pp.591-597, 1971.

C. Reinsch, Communications of the ACM, Algorithm 464, Vol. 16, p.683, 1973.

H. Boulder, R. S. Martin, C. Reinsch, J. H. Wilkinson, Numerical Mathematics, Vol. 11, pp.293-306, 1968.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

9.2.5 BISECT Eigenvalues within Specified Interval

Name

sbisect — Forms eigenvalues which lie in a specified interval, single precision
dbisect — Forms eigenvalues which lie in a specified interval, double precision

Synopsis

```
#include <ceispack.h>
void sbisect (n, eps1, d, e, e2, lb, ub, mm, m, w, ind, ierr, rv4, rv5)
int n;
float *eps1, *d, *e, *e2, *w, lb, ub;
int mm, *m;
int *ind, *ierr;
float *rv4, *rv5;

void dbisect (n, eps1, d, e, e2, lb, ub, mm, m, w, ind, ierr, rv4, rv5)
int n;
double *eps1, *d, *e, *e2, *w, lb, ub;
int mm, *m;
int *ind, *ierr;
double *rv4, *rv5;
```

n — order of input symmetric tridiagonal matrix
eps1 — input/output variable address containing absolute error tolerance
d — input vector of length n containing diagonal elements of input matrix
e — input vector of length n containing subdiagonal elements of input matrix
e2 — length n input array of squares of subdiagonal elements of input matrix
lb — input constant specifying lower bound of interval
lu — input constant specifying upper bound of interval
mm — input variable, estimate of number eigenvalues in interval
m — output variable address containing number of eigenvalues actually found
w — length mm output array containing m eigenvalues
ind — length mm output array containing m submatrix indices of eigenvalues
ierr — output variable address containing error completion code
rv4 — temporary storage array of length n
rv5 — temporary storage array of length n

Diagnostics

The output error flag `*ierr` is set to:

- 0 for normal return.

- $(3n+1)$ if the actual number of eigenvalues which lie in a specified interval m is greater than the specified number mm. In such a case no eigenvalues have been computed.

Description

Function bisect computes the eigenvalues of a real symmetric tridiagonal matrix which belong to a specified interval using the bisection technique applied to the Sturm sequence and associates the computed eigenvalues with their corresponding submatrix indices.

For more details about the bisection algorithm see the description of the function `tsturm`. The function bisect is the subset of the C function `tsturm`.

The input variable mm should be set to an expected upper bound for the number of eigenvalues that lie in the interval. If more than `mm` eigenvalues lie in the interval, an error is returned indicating no eigenvalues found.

The variable `*eps1` on input contains an absolute error tolerance for the computation of the eigenvalues. If the input value ε_1 is non-positive, it is reset for each submatrix to a default value, namely to the negative of the product of the relative machine precision and the larger magnitude Gershgorin bound of the submatrix. The variable is unchanged on output if it has not been reset to its *last* default value.

The input vectors d and e contain diagonal and subdiagonal elements of the original matrix. The subdiagonal elements should be stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array is arbitrary.

The input vector $e2$ contains the squares of the subdiagonal elements of the original matrix in its *last* $(n-1)$ locations. The entry `e2[0]` is arbitrary. On return from the function the entries of vector $e2$ corresponding to the negligible subdiagonal elements of input matrix are set to zero. The entry `e2[0]` is also set to zero.

The input constants lb and ub determine the interval containing required eigenvalues. If $lb \geq ub$ then no eigenvalues are found.

On return from the function the variable `*m` contains the actual number of eigenvalues found to lie in the interval $[l_b, u_b)$.

The output array w contains m computed eigenvalues. They are arranged in ascending order.

The output vector ind contains the submatrix indices associated with the corresponding eigenvalues in w array. An entry of array ind is set to 1 for eigenvalues belonging to the first submatrix from the top, 2 for those belonging to the second submatrix, etc.

Performance

The rate of convergence of bisection method is commensurate to the value of the absolute error tolerance for computed eigenvalues.

The computed eigenvalues are close to those of the exact matrix. Also they are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

The theoretical upper bound for the errors in computed eigenvalues is defined by the following formula:

$$\delta = 0.5\varepsilon_1 + 7\varepsilon \max(|x_{\min}|, |x_{\max}|)$$

where ε is the relative machine precision, $x_{\min}$ and $x_{\max}$ are lower and upper margins for Gershgorin interval.

Notes

The input vectors of diagonal and subdiagonal elements d and e of the original matrix are preserved by the function bisect.

Test

The test program is contained in `bisectt.c` file. The test matrices are contained in `trisym1.xpd, trisym2.xpd, trisym3.xpd` and `trisym4.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

Barth, R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 9, pp.386-393, 1967.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

9.2.6 TRIDIB Eigenvalues within Specified Boundary Indices

Name

stridib — Forms eigenvalues within a specified boundary indices, single precision
dtridib — Forms eigenvalues within a specified boundary indices, double precision

Synopsis

```
#include <ceispack.h>
void stridib (n, eps1, d, e, e2, lb, ub, m11, m, w, ind, ierr, rv4, rv5)
int n;
float *eps1, *d, *e, *e2, *lb, *ub, *w;
int m11, m;
int *ind, *ierr;
float *rv4, *rv5;

void dtridib (n, eps1, d, e, e2, lb, ub, m11, m, w, ind, ierr, rv4, rv5)
int n;
double *eps1, *d, *e, *e2, *lb, *ub, *w;
int m11, m;
int *ind, *ierr;
double *rv4, *rv5;
```

n — order of input symmetric tridiagonal matrix
eps1 — input/output variable address containing absolute error tolerance
d — input vector of length n containing diagonal elements of input matrix
e — input vector of length n containing subdiagonal elements of input matrix
e2 — length n input array of squares of subdiagonal elements of input matrix
lb — output variable address containing lower bound of exact interval
lu — output variable address containing upper bound of exact interval
m11 — input variable containing lower boundary index for required eigenvalues
m — input variable containing number of required eigenvalues
w — length m output array containing m eigenvalues
ind — length m output array containing m submatrix indices of eigenvalues
ierr — output variable address containing error completion code
rv4 — temporary storage array of length n
rv5 — temporary storage array of length n

Diagnostics

The output error flag `*ierr` is set to:

- 0 for normal return.
- $(3n + 1)$ if multiple eigenvalue associated with index $m11$ makes unique selection impossible,

- $(3n+2)$ if multiple eigenvalue associated with index $m22 = m11 + m - 1$ makes unique selection impossible.

Description

Function tridib computes the eigenvectors of a real symmetric tridiagonal matrix using the bisection technique as described for the function `tsturm`. The only difference from the `tsturm` function is that tridib first establishes the exact half-open interval $[l_b, u_b)$ containing the required eigenvalues:

$$\lambda_i, \text{where } i \in [m_{11}, m_{22}]$$

The eigenvalue upper boundary index m_{22} is equal to $m_{11} + m - 1$.

First, tridib evaluates the Gershgorin bounds containing all the eigenvalues of the matrix. At the next stage tridib calculates the Gershgorin bounds using the bisection technique and refines the exact interval to contain the required eigenvalues.

The variable `*eps1` contains an absolute error tolerance for the computation of the eigenvalues. If the input value ε_1 is non-positive, it is reset for each submatrix to a default value, namely the negative of the product of the relative machine precision and the larger in magnitude Gershgorin bound of the submatrix. On output the variable is unchanged if it has not been reset to its *last* default value.

The input vectors d and e contain diagonal and subdiagonal elements of the original matrix. The subdiagonal elements should be stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array is arbitrary.

The input vector $e2$ contains the squares of the subdiagonal elements of the original matrix in its *last* $(n-1)$ locations. The entry `e2[0]` is arbitrary. On return from the function the entries of vector $e2$ corresponding to the negligible subdiagonal elements of input matrix are set to zero. The entry `e2[0]` is also set to zero.

The output variables `*lb, *ub` define the interval exactly containing the required eigenvalues.

The output array w contains the computed eigenvalues within indices $i \in [m_{11}, m_{22}]$ in its first m positions. They are arranged in ascending order.

The output vector *ind* contains the submatrix indices associated with the corresponding eigenvalues in w array. An entry of array *ind* is set to 1 for eigenvalues belonging to the first submatrix from the top, 2 for those belonging to the second submatrix, etc.

Performance

The rate of convergence of bisection method is commensurate to the value of the absolute error tolerance for computed eigenvalues.

The computed eigenvalues are close to those of the exact matrix. Also they are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Theoretical upper bound for the errors in computed eigenvalues is defined by the following formula:

$$\delta = 0.5\varepsilon_1 + 7\varepsilon \max(|x_{\min}|, |x_{\max}|)$$

where ε is the relative machine precision, $x_{\min}$ and $x_{\max}$ are lower and upper margins for Gershgorin interval.

Notes

The input vectors of diagonal and subdiagonal elements d and e of the original matrix are preserved by the function tridib.

Test

The test program is contained in `tridibt.c` file. The example matrices are contained in `trisym1.xpd, trisym2.xpd, trisym3.xpd` and `trisym4.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

Barth, R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 9, pp.386-393, 1967.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

9.2.7 RATQR Smallest or Largest Eigenvalue, Rational QR Algorithm

Name

sratqr — Finds algebraically smallest or largest eigenvalue, single precision
dratqr — Finds algebraically smallest or largest eigenvalue, double precision

Synopsis

```
#include <ceispack.h>
void sratqr (n, eps1, d, e, e2, m, w, ind, bd, type, idef, ierr)
int n, m;
float *eps1;
float *d, *e, *e2, *w, *bd;
int *ind, type, idef, *ierr;

void dratqr (n, eps1, d, e, e2, m, w, ind, bd, type, idef, ierr)
int n, m;
double *eps1;
double *d, *e, *e2, *w, *bd;
int *ind, type, idef, *ierr;
```

n — order of symmetric tridiagonal input matrix
eps1 — input/output variable address containing absolute error tolerance
d — input vector of length n containing diagonal elements of input matrix
e — input vector of length n containing subdiagonal elements of input matrix
e2 — length n input array of squares of subdiagonal elements of input matrix
m — input variable containing number of eigenvalues required
w — length n output array containing m eigenvalues in ascending/descending order
ind — length n output array containing m submatrix indices of eigenvalues
bd — length n output array containing refined bounds for theoretical errors
type — input flag which causes computation of smallest or largest eigenvalues
idef — input flag specifying whether input matrix is positive definite or not
ierr — output variable address containing error completion code

Diagnostics

The output error flag `*ierr` is set to

- 0 for normal return.
- $(6n + 1)$ if *idef* is set to 1 and *type* to 1, and the matrix is found to be not positive definite; or if *idef* is set to -1 and *type* to 0, and the matrix is found to be not negative definite. No eigenvalues have been computed.

- $(5n + k)$ if successive iterate to the k^{th} eigenvalue are not strictly monotone increasing. The sum of all the shifts applied up to the moment is taken as an eigenvalue and the function continues with the next eigenvalue. If this error occurs more than once, the value of k refers the last failure.

Description

Function ratqr determines the algebraically smallest or largest eigenvalues of a symmetric tridiagonal matrix by the rational QR algorithm using Newton method for shift computations.

The rational QR algorithm iterates a sequence of symmetric tridiagonal matrices that are orthogonally similar to the original matrix with shifted origin. The sequence converges to a matrix having diagonal elements set to zero. The rate of convergence is improved using the shift of origin.

At first, the diagonals of the original matrix are copied into vector w. If the input flag *type* indicates that the largest eigenvalues are to be found, the signs of the elements of the vector w are inverted as well as the sign of an internal variable where the input flag `idef` has been stored to indicate that the matrix definiteness is inverted. This is done because the algorithm is arranged to find the algebraically smallest eigenvalues. In this case, on output the signs of computed eigenvalues will be back reversed.

At the next step the subdiagonal entries are verified for negligibility. The tolerance for the test is proportional to the machine precision. If an element is found to be negligible, the original matrix is split to a direct sum of submatrices and the square of corresponding subdiagonal element is set to zero in the vector $e2$.

The internal representation of the input matrix is shifted if the matrix is negative semi-definite or if it is positive definite and the Gershgorin lower bound for the smallest eigenvalue is greater than zero. The shifted matrix has a Gershgorin lower bound equal to zero.

The QR iterations begin with the originally shifted origin. At each iteration the function performs QR decomposition, calculates the Newton shift and again performs a QR step with the computed shift of the origin. This process is repeated until one of the diagonal elements of iterated matrix becomes negligible. When a diagonal element becomes negligible, the sum of shifts applied is taken as an eigenvalue, the original matrix is deflated by deleting the row and column that contain this negligible element and the process described above is again applied to the deflated matrix.

The input value of `*eps1` specifies a theoretical absolute error tolerance for computed eigenvalues. It may be set to a non-positive value. If at any iteration is becomes smaller than the machine precision times the sum of shifts it is reset to this product. On output it contains the maximum of its input value and the product of the relative machine precision and the maximum magnitude of an eigenvalue iterate. The theoretical absolute error in the i^{th} eigenvalue is usually not greater than the output value ε_1 of the variable `*eps1` times i.

The input flag `idef` should be set to 1 if the input matrix is known to be positive definite, to -1 if the input matrix is known to be negative definite, and to 0 otherwise.

The input flag `type` should be set to non-zero if the smallest eigenvalues are to be computed, and to 0 if the largest eigenvalues are required.

The input vectors d and e contain diagonal and subdiagonal elements of the original matrix. The subdiagonal elements should be stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array is arbitrary.

The input array $e2$ contains the squares of the subdiagonal elements of the original matrix in its *last* $(n-1)$ locations. The entry `e2[0]` is arbitrary. On return from the function the entries of vector $e2$, corresponding to the negligible subdiagonal elements of original matrix are set to zero. The element `e2[0]` is also set to 0.

On return from the function the m eigenvalues are stored in the first locations of the w array in ascending order if the smallest eigenvalues have been found or in descending order if the largest eigenvalues have been found. The original matrix is unchanged.

The output vector bd contains refined bounds for the theoretical errors in the corresponding eigenvalues stored in array w. These bounds are usually more precise than the tolerance specified by the output value ε_1.

The output vector ind contains in its first m positions the submatrix indices associated with the corresponding eigenvalues in w array. An entry of array ind is set to 1 for eigenvalues belonging to the first submatrix from the top, 2 for those belonging to the second submatrix, etc.

Performance

The Newton scheme for shift computation makes the convergence of the algorithm global and stable. The rate of convergence is of second order, therefore the algorithm is best applied when a relatively high accuracy of the eigenvalues is required and they are not multiple or close to each other. The Bisection method can significantly reduce time for the computation of close eigenvalues. The computed eigenvalues are close to those of the exact matrix. They are also exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is proportional to the relative machine precision.

Notes

Array *bd* may coincide with array *e*2. If it is distinct from the array *e*2, then on output the element `e2[0]` is set to 0 if the smallest eigenvalues have been found or to 2 if the largest eigenvalues have been found.
The input arrays *d* and *e* are preserved by the function ratqr.

Test

The test program is contained in `ratqrt.c` file. The example matrices are contained in `trisym1.xpd, trisym.xpd, trisym3.xpd` files.

References

C. Reinsch, F. L. Bauer, Numerical Mathematics, Vol. 11, pp.264-272, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

9.2.8 TINVIT Eigenvectors Corresponding to Specified Eigenvalues

Name

stinvit — Forms eigenvectors corresponding to specified eigenvalues, single precision
dtinvit — Forms eigenvectors corresponding to specified eigenvalues, double precision

Synopsis

```
#include <ceispack.h>
int stinvit (n, d, e, e2, m, w, ind, z, index, rv1, rv2, rv3, rv4, rv6)
int n, m;
float *d, *e, *e2, *w, **z;
int *ind, *index;
float *rv1, *rv2, *rv3, *rv4, *rv6;

int dtinvit (n, d, e, e2, m, w, ind, z, index, rv1, rv2, rv3, rv4, rv6)
int n, m;
double *d, *e, *e2, *w, **z;
int *ind, *index;
double *rv1, *rv2, *rv3, *rv4, *rv6;
```

n — order of symmetric tridiagonal input matrix
d — input vector of length n containing diagonal elements of input matrix
e — input vector of length n containing subdiagonal elements of input matrix
e2 — length n input array of squares of subdiagonal elements of input matrix
m — input variable containing number of eigenvectors required
w — length m input array containing eigenvalues in ascending/descending order
ind — length m input array containing submatrix indices of eigenvalues
z — output matrix containing m transposed (row) orthonormal eigenvectors
index — output variable address containing error completion code
rv1 — temporary storage array of length n
rv2 — temporary storage array of length n
rv3 — temporary storage array of length n
rv4 — temporary storage array of length n
rv6 — temporary storage array of length n

Diagnostics

The function returns 0 for normal return or 1 if an error exit has been made. The output variable `*index` is set to:

- 0 for normal return.

- -k if the eigenvector corresponding to the k^{th} eigenvalue has not converge after 5 iterations. If this failure occurs more than once then `*index` contains the negative of the *last* index. The rows of matrix z corresponding to such eigenvalues contain zero vectors.

Description

Function tinvit computes the eigenvectors of a real symmetric tridiagonal matrix corresponding to the eigenvalues stored in the first m locations of the input array w, using an inverse iteration technique.

For more details about the inverse iteration algorithm see the description of the function `tsturm`. The function tinvit is the subset of the C function `tsturm`.

The input vectors d and e contain diagonal and subdiagonal elements of the original matrix. The subdiagonal elements should be stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array is arbitrary.

The input vector $e2$ contains the squares of the corresponding subdiagonal elements of the original matrix, where the negligible subdiagonal elements have been replaced with zeros. An element $t_{i,i-1}$ is considered negligible if it is not larger than the product of the relative machine precision and the sum of the magnitudes of consecutive diagonal elements $t_{ii}t_{i-1,i-1}$. The element `e2[0]` must contain 0 if the eigenvalues are stored in ascending order, or 2 if they are stored in descending order. If the functions *bisect*, *tridib*, or *imtqlv* have been used to find the eigenvalues, the output array $e2$ is set to exactly what is required by the function tinvit.

The input vector w contains m eigenvalues stores in ascending or descending order.

The input vector *ind* contains the submatrix indices associated with the corresponding eigenvalues in w array. An entry of array *ind* is set to 1 for eigenvalues belonging to the first submatrix from the top, 2 for those belonging to the second submatrix, etc.

On return from the function matrix z contains m transposed (row) orthonormal eigenvectors. Eigenvectors which failed to converge are set to zero.

Performance

The rate of convergence of the inverse iterations is linear. In most cases the appropriate accuracy of an eigenvector is achived in about 1–2 iterations.

The computed eigenvectors are close to exact eigenvectors of a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

The $m \times n$ matrix z is expected to be in the format produced by the allocation function `fmatrix`. The number of rows of the matrix may be greater than m.

The input arrays *d, e* $e2$, *w, ind* are preserved by the function tinvit.

Test

The test programs are contained in `bakvect.c, imtql1t.c, imtqlvt.c, ratqrt.c, tql1t.c, tqlratt.c, bisectt.c, tridibt.c` files. The example matrices are contained in `trispc1.xpd, trispc2.xpd` and `trisym1.xpd, trisym2.xpd, trisym3.xpd, trisym4.xpd` files.

References

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

9.2.9 TQL2 Eigenvalues and Eigenvectors, QL Algorithm

Name

stql2 — Forms eigenvalues and corresponding eigenvectors, single precision
dtql2 — Forms eigenvalues and corresponding eigenvectors, double precision

Synopsis

```
#include <ceispack.h>
int stql2 (n, d, e, z, index)
int n;
float *d, *e, **z;
int *index;

int dtql2 (n, d, e, z, index)
int n;
double *d, *e, **z;
int *index;
```

n	— order of symmetric tridiagonal input /output eigenvector matrices
d	— length n input vector of diagonal elements/output n ascending eigenvalues
e	— length n input vector containing subdiagonal elements of input matrix
z	— output matrix containing n transposed (row) orthonormal eigenvectors
index	— output variable address containing error completion code

Diagnostics

The function returns 0 for normal return or 1 if an error exit has been made. The output variable `*index` is set to:

- 0 for normal return.
- i if the i^{th} eigenvalue has not been determined after `MAXITER` iterations. The eigenvalues corresponding to indices $j \in [0, i-1]$ would be correct but unordered. The eigenvectors corresponding to them are correct and stored in the first k rows of matrix z.

Description

Function tql2 computes all the eigenvalues and corresponding eigenvectors of a real symmetric tri-diagonal matrix using the QL algorithm with shift of origin.

The QL algorithm iterates the sequence of symmetric tridiagonal matrices that are orthogonally similar to the original matrix. The sequence converges to a diagonal matrix. A shift of origin is applied to improve the rate of convergence.

Before the next iteration the currently iterated submatrix is tested for possible splitting to submatrices. If the matrix is split only the uppermost submatrix takes part in the next iterations. The shift of origin on each iteration is taken to be the eigenvalue of the uppermost 2×2 principal minor, closer to the upper diagonal element of this minor. When the uppermost 1×1 principal minor has split from the rest of the matrix, its value is taken as an eigenvalue of the original matrix and the iterations continue on the remaining submatrix until the whole matrix finally split on minors of order 1. The tolerance of the test for splitting is commensurate to the relative machine precision.

The information about orthogonal similarity transformation used in the QL decomposition is stored into matrix z, thus forming the orthogonal eigenvectors.

The input vectors d and e contain diagonal and subdiagonal elements of the original matrix. The subdiagonal elements should be stored in the *last* $(n - 1)$ locations of the vector e. The first entry of the array is arbitrary.

On input matrix z should contain the transformation matrix produced in the reduction to symmetric tridiagonal form by either the functions `tred2`, `htridi` or `htrid3`, if they have been performed. If the eigenvectors of the tridiagonal matrix are desired, matrix z should be set to the identity matrix.

On return from the function matrix z contains transposed (row) orthonormal eigenvectors of the original matrix.

When all the eigenvalues and eigenvectors are computed the eigenvalues are ordered in ascending order, and stored in the output vector d. The eigenvectors are interchanged respectively.

Performance

Appropriate shift of the origin makes the convergence of the algorithm global. The rate of convergence is cubic in almost all the cases.

The computed eigenvalues are close to those of the exact matrix. Also they are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is proportional to the machine precision times the norm of the original matrix at least in a case when the successive row norms are increase from top to bottom of the original matrix.

The computed eigenvectors are close to exact eigenvectors of such a perturbed matrix.

Notes

Array z is in the format produced by the allocation function `fsquare`.

If the original matrix has its row norms widely varying and not strictly decreasing downward, it is recommended to use implicit QL algorithm for eigenvalue computation or to permute the original matrix to obtain the successive row norms increasing from top to bottom of the matrix.

Test

The test programs are contained in `cht.c, ch3t.c, rsgt.c, rsgabt.c, rgbat.c, tql2t.c, rsbt.c, rstest.c` files. The example matrices are contained in `chermiti0.xpd, chermiti1.xpd, chermit30.xpd, chermit31.xpd, rgensym.xpd, rsymmet1.xpd, rsymmet2.xpd, rsymmet3.xpd` and `trisym1.xpd, trisym2.xpd, trisym3.xpd, trisym4.xpd` files.

References

H. Boulder, R. S. Martin, C. Reinsch, J. H. Wilkinson, Numerical Mathematics, Vol. 11, pp.293-306, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

9.2.10 IMTQL2 Eigenvalues and Eigenvectors, Implicit QL Algorithm

Name

simtql2 — Forms eigenvalues and corresponding eigenvectors, single precision
dimtql2 — Forms eigenvalues and corresponding eigenvectors, double precision

Synopsis

```
#include <ceispack.h>
int simtql2 (n, d, e, z, index)
int n;
float *d, *e, **z;
int *index;

int dimtql2 (n, d, e, z, index)
int n;
double *d, *e, **z;
int *index;
```

n	— order of symmetric tridiagonal input /output eigenvector matrices
d	— length n input vector of diagonal elements/output n ascending eigenvalues
e	— length n input vector containing subdiagonal elements of input matrix
z	— output matrix containing n transposed (row) orthonormal eigenvectors
index	— output variable address containing error completion code

Diagnostics

The function returns 0 for normal return or 1 if an error exit has been made. The output variable `*index` is set to:

- 0 for normal return.
- i if the i^{th} eigenvalue has not been determined after `MAXITER` iterations. The eigenvalues corresponding to indices $j \in [0, i-1]$ would be correct but unordered. The eigenvectors corresponding to them are correct and stored in the first k rows of matrix z.

Description

Function imtql2 computes all the eigenvalues and corresponding eigenvectors of a real symmetric tridiagonal matrix using the QL algorithm with implicit shifts of origin.

The implicit QL algorithm iterates a sequence of symmetric tridiagonal matrices that are orthogonally similar to the original matrix. The sequence converges to a diagonal matrix.

A shift of origin is applied *implicitly* to improve the rate of convergence.

Before the next iteration the currently iterated submatrix is tested for possible splitting to submatrices. If the matrix is split only the uppermost submatrix takes part in the next iterations. The shift of origin on each iteration is taken to be the eigenvalue of the uppermost 2×2 principal minor, closer to the upper diagonal element of this minor. When the uppermost 1×1 principal minor has split from the rest of the matrix, its value is taken as an eigenvalue of the original matrix and iterations continue on the remaining submatrix until the whole matrix finally split on minors of order 1. The tolerance of the test for splitting is commensurate to the relative machine precision.

The information about orthogonal similarity transformation used in the QL decomposition is stored into matrix z, thus forming the orthogonal eigenvectors.

The input vectors d and e contain diagonal and subdiagonal elements of the original matrix. The subdiagonal elements should be stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array is arbitrary.

On input matrix z should contain the transformation matrix produced in the reduction to symmetric tridiagonal form by either the functions `tred2`, `htridi` or `htrid3`, if they have been performed. If the eigenvectors of the tridiagonal matrix are desired, matrix z should be set to the identity matrix.

On return from the function matrix z contains transposed (row) orthonormal eigenvectors of the original matrix.

When all the eigenvalues and eigenvectors are computed the eigenvalues are ordered in ascending order, and stored in the output vector d. The eigenvectors are interchanged respectively.

Performance

Appropriate shift of the origin makes the convergence of the algorithm global. The rate of convergence is cubic in almost all the cases.

The computed eigenvalues are close to those of the exact matrix. Also they are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is proportional to the machine precision times the norm of the original matrix.

The computed eigenvectors are close to exact eigenvectors of such a perturbed matrix.

As opposed to the QL algorithm with the explicit shifts, the implicit modification is not sensitive to the matrix structure with its row norms widely varying and not strictly decreasing downward.

Notes

Array z is in the format produced by the allocation function `fsquare`.

Test

The test programs are contained in `rtt.c, rstt.c, imtql2t.c` files. The example matrices are contained in `trisym1.xpd, trisym2.xpd, trisym3.xpd` and `trisym4.xpd` files.

References

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 12, pp.377-383, 1968.

A. Dubrulle, Numerical Mathematics, Vol. 15, p.450, 1970.
J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

9.2.11 TSTURM Eigenvalues and Eigenvectors, Bisection Technique

Name

ststurm — Forms some eigenvalues and corresponding eigenvectors, single precision
dtsturm — Forms some eigenvalues and corresponding eigenvectors, double precision

Synopsis

```
#include <ceispack.h>
void ststurm (n,eps1,d,e,e2,lb,ub,mm,m,w,z,ierr,rv1,rv2,rv3,rv4,rv5,rv6)
int n;
float *eps1, *d, *e, *e2, lb, ub;
int mm, *m;
float *w, **z;
int *ierr;
float *rv1, *rv2, *rv3, *rv4, *rv5, *rv6;

void dtsturm (n,eps1,d,e,e2,lb,ub,mm,m,w,z,ierr,rv1,rv2,rv3,rv4,rv5,rv6)
int n;
double *eps1, *d, *e, *e2, lb, ub;
int mm, *m;
double *w, **z;
int *ierr;
double *rv1, *rv2, *rv3, *rv4, *rv5, *rv6;
```

n	— order of symmetric tridiagonal input matrix
eps1	— input/output variable address containing eigenvalue absolute error tolerance
d	— input vector of length n containing diagonal elements of input matrix
e	— input vector of length n containing subdiagonal elements of input matrix
e2	— length n input array of squares of subdiagonal elements of input matrix
lb	— input constant specifying lower bound of interval
lu	— input constant specifying upper bound of interval
mm	— input variable, estimate number of eigenvalues in interval
m	— output variable address containing number of eigenvalues actually found
w	— length mm output array containing m eigenvalues
z	— output array containing m transposed (row) eigenvectors
ierr	— output variable address containing error completion code
rv1	— temporary storage array of length n
rv2	— temporary storage array of length n
rv3	— temporary storage array of length n
rv4	— temporary storage array of length n
rv5	— temporary storage array of length n
rv6	— temporary storage array of length n

Diagnostics

The output error flag `*ierr` is set to:

- 0 for normal return.
- $(3n + 1)$ if the actual number of eigenvalues m exceeds the specified number mm. In this case no eigenvalues and eigenvectors have been computed. The output variable `*m` contains the actual number of the eigenvalues lie on the interval specified.
- $(4n + i)$ if the eigenvector corresponding to the i^{th} eigenvalue has not converged in 5 iterations. The eigenvalues and eigenvectors should be correct for indices $j \in [0, i - 1]$.

Description

Function tsturm computes the eigenvalues of a real symmetric tridiagonal matrix which belong to the specified interval, using the bisection technique applied to the Sturm sequence. Eigenvectors corresponding to obtained eigenvalues are computed using an inverse iteration method.

First the function tsturm determines negligible subdiagonal elements of the original matrix and set to zero the corresponding entries of array $e2$, therefore splitting the original matrix into the direct sum of submatrices.

At the next step, the function calculates the Sturm sequence at both the interval boundaries and estimates the number of eigenvalues within the interval as a difference between the values of Sturm sequences at the upper and lower bounds u_b and l_b.

The function then performs the computation of the eigenvalues of a the submatrices. The half-open input interval $[l_b, u_b)$ is refined by the Gershgorin interval containing all the eigenvalues. The subintervals comprising the eigenvalues are squeezed by the bisection process until the bounds of the subinterval become close enough each other. The measure of the closeness is chosen as:

$$x_0 - x_u \leq \varepsilon_1 + 2\varepsilon(|x_0| + |x_u|),$$

where x_0, x_u are current upper and lower bounds of the subinterval, ε is the relative machine precision. If the input value of eigenvalue absolute error tolerance ε_1 is non-positive it is reset to the negative of the relative precision times the l_1 norm of the current submatrix:

$$\varepsilon'_1 = -\varepsilon \|T_c\|_1$$

When all the eigenvalues of current submatrix have been evaluated, the corresponding eigenvectors are computed. The LU decomposition of the submatrix is performed by Gaussian elimination with partial pivoting. The information about the decomposition is stored into working arrays, thus avoiding repetitive decomposition for the next eigenvector iterations of current submatrix. Starting from an initial vector, the approximate eigenvector is obtained by the back substitution process. Up to five successive iterations for the different initial vectors are tried in order to obtain an acceptable growth of the eigenvector iterate norm. The accepted vector is normalized so that its Euclidean norm is made 1.

The eigenvectors which correspond to separated eigenvalues should be orthogonal, while those corresponding to the group of close eigenvalues or multiple ones may be weakly orthogonal. To provide orthogonal eigenvectors, each eigenvector iterate is orthogonalized with respect to the eigenvectors of the group of close eigenvalues. The multiple eigenvalues are perturbed by the multiple of the related machine precision to obtain linearly independent eigenvectors. The output values of these eigenvalues is stored unperturbed.

The process described above is applied for all the submatrices from top to bottom of the original matrix, until all the eigenvalues and corresponding eigenvectors on the interval are computed.

The input variable *mm* should be set to an expected upper bound for the number of eigenvalues lie in the interval. If more than *mm* eigenvalues are determined to belong in the interval, an error return is made with no eigenvalues found.

The variable `*eps1` on input contains an absolute error tolerance for the computation of the eigenvalues. If the input value ε_1 is non-positive, it is reset for each submatrix to a default value, namely to the negative of the product of the relative machine precision and the larger magnitude Gershgorin bound of the submatrix. The variable is unchanged on output if it has not been reset to its *last* default value.

The input vectors d and e contain diagonal and subdiagonal elements of the original matrix. The subdiagonal elements should be stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array is arbitrary.

The input vector $e2$ contains the squares of the subdiagonal elements of the original matrix in its *last* $(n-1)$ locations. The entry `e2[0]` is arbitrary. On return from the function the entries of vector $e2$ corresponding to the negligible subdiagonal elements of input matrix are set to zero. The entry `e2[0]` is also set to zero.

The input constants lb and ub determine the interval containing required eigenvalues. If $lb \geq ub$ then no eigenvalues would be found.

On return from the function the variable `*m` contains the actual number of eigenvalues found to lie in the interval $[l_b, u_b)$.

The output array w contains m computed eigenvalues. They are arranged in ascending order.

The output vector *ind* contains the submatrix indices associated with the corresponding eigenvalues in w array. An entry of array *ind* is set to 1 for eigenvalues belonging to the first submatrix from the top, 2 for those belonging to the second submatrix, etc.

On return from the function matrix z contains m transposed (row) eigenvectors, associated with the corresponding eigenvalues. The eigenvectors are orthogonal and normalized.

Performance

The rate of convergence of bisection method is commensurate to the value of the absolute error tolerance for computed eigenvalues.

The computed eigenvalues are close to those of the exact matrix. Also they are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Theoretical upper bound for the errors in computed eigenvalues is defined by the following formula:

$$\delta = 0.5\varepsilon_1 + 7\varepsilon \max(|x_{\min}|, |x_{\max}|)$$

where ε is the relative machine precision, $x_{\min}$ and $x_{\max}$ are lower and upper margins for Gershgorin interval.

The rate of convergence of the inverse iterations is linear. In the most cases the appropriate accuracy of an eigenvector is achieved in about 1–2 iterations.

The computed eigenvectors are exact for a matrix which is a small perturbation of the original matrix. They are orthogonal within a high precision, even if the matrix has a multiple or a group of close eigenvalues.

Notes

The rectangular $mm \times n$ array z is in the format produced by the allocation function `fmatrix`. The number of rows in the array may be greater than mm.

The input vectors of diagonal and subdiagonal elements d and e of the original matrix are preserved by the function tsturm.

Test

The test program is contained in `tsturmt.c` file. The example matrices are contained in `trisym1.xpd, trisym2.xpd, trisym3.xpd` and `trisym4.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

Chapter 10

Complex General Eigenproblem

10.1 Matrix Balancing

10.1.1 CBAL Balancing

Name

ccbal — Balances complex matrix, single precision
zcbal — Balances complex matrix, double precision

Synopsis

```
#include <ceispack.h>
void ccbal (n, a, low, igh, scale)
int n;
fcomplex **a;
int *low, *igh;
float *scale;

void zcbal (n, a, low, igh, scale)
int n;
dcomplex **a;
int *low, *igh;
double *scale;
```

n — order of input/output matrices
a — unbalanced input matrix /balanced output matrix
low — output variable address containing lower index of balanced submatrix
igh — output address variable containing higher index of balanced submatrix
scale — length n output array containing scaling factors and permutation indices

Description

Function `cbal` balances a complex square general matrix and extracts the eigenvalues of the original matrix when they can be found exactly.

The reduction is performed as follows. First the matrix is rendered to block-triangular form (if it is possible):

$$PAP = \begin{pmatrix} R & \vdots & W & \vdots & X \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & B & \vdots & Y \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & 0 & \vdots & T \end{pmatrix},$$

by applying permutation similarity transformations P. The matrix B is of order $(igh - low + 1)$ and located in rows and columns from low to igh. Matrix B has the property such that if matrix B_0 is equal to matrix B with its diagonal elements set to zero, then the matrix B_0 does not have any row or column of zero 1-norm. The matrices R and T are upper triangular, other matrices of respective dimensions are rectangular. The diagonal elements of matrices R and T are the eigenvectors of the original matrix.

Two boundary cases arise. The first one occurs when matrix B is of order 0. In such a case the function sets the output variables low and igh both to 0. The second case occurs when matrix B is of order n and all other submatrices are empty. In this case the product of permutations $P = I$ and the function sets $low = 0$ and $igh = n - 1$.

At the next step the elements of matrix B are transformed by the diagonal similarity transformation:

$$B_f = D^{-1}BD,$$

such that the 1-norms of rows and corresponding columns in the matrix B_f are nearly equal. The entries of diagonal matrix D (that are scaling factors) are chosen be integer powers of the base machine arithmetic, so that the transformation does not produce any rounding errors. The whole matrix has the form:

$$D^{-1}PAPD = \begin{pmatrix} R & \vdots & WD & \vdots & X \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & D^{-1}BD & \vdots & D^{-1}Y \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & 0 & \vdots & T \end{pmatrix},$$

The 1-norm of the original matrix also is reduced by the diagonal transformation when 1-norms of columns and corresponding rows of the original matrix are significantly different. Generally reducing the 1-norm of a matrix improves the accuracy in determining eigenvalues and eigenvectors.

The similarity transformations applied keep the eigenvalues of the balanced matrix equal to those of the original matrix.

On return from the function array *scale* contains the information about the permutations and diagonal transformations used by the function cbal and should be interpreted as follows:

1. The rows and columns j and $scale_j$ have been interchanged for indices $j \in [0, low-1]$ and $j \in [igh+1, n-1]$. The interchanges are performed in indices first from $(n-1)$ to $(igh+1)$, then from 0 to $(low-1)$.

2. The elements from low to igh of the diagonal matrix D are stored in the locations $scale_j$, where $j \in [low, igh]$.

Performance

Since the non-unitary entries in the diagonal matrix D are integer powers of the base machine arithmetic, the function cbal produces no rounding errors.

Notes

Array a is in the format produced by the allocation function `fsquare`.

The function `cbabk2` should be performed after the computation of eigenvectors of balanced matrix in order to reconstruct the eigenvectors of the original (unbalanced) matrix.

Test

The test programs are contained in `cgt.c, combakt.c, cortbt.c` files. The example matrices are contained in `cmatrix1.xpd, cmatrix2.xpd, cmatrix3.xpd, cmatrix4.xpd, cmatrix5.c` files.

References

B. N. Parlett, C. Reinsch, Numerical Mathematics, Vol. 13, pp.293-304, 1969.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

10.2 Reduction to Hessenberg form

10.2.1 COMHES Reduction by Elementary Transformations

Name

ccomhes — Reduction by stabilized elementary transformations, single precision
zcomhes — Reduction by stabilized elementary transformations, double precision

Synopsis

```
#include <ceispack.h>
void ccomhes (n, low, igh, a, iperm)
int n, low, igh;
fcomplex **a;
int *iperm;

void zcomhes (n, low, igh, a, iperm)
int n, low, igh;
dcomplex **a;
int *iperm;
```

n — order of input/output matrices
low — input constant set to lower index of balanced submatrix
igh — input constant set to higher index of balanced submatrix
a — input (balanced)/output upper Hessenberg and transformations matrix
iperm — output integer array of length at least *igh* containing permutation indices

Description

Function comhes reduces a submatrix located in the rows and columns from *low* to *igh* of given complex general matrix to upper Hessenberg form using and accumulating elementary similarity transformations stabilized by permutation transformations.

The above submatrix is derived usually from balancing the original general matrix performed by the function `cbal`, that reduces the original matrix into the block-triangular form:

$$\left(\begin{array}{ccccc} R & \vdots & W & \vdots & X \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & B & \vdots & Y \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & 0 & \vdots & T \end{array}\right),$$

where matrix B is of order $(igh - low + 1)$ and located in rows and columns from *low* to *igh*, matrices R and T are upper triangular, other matrices are rectangular of the respective dimensions.

The reduction is performed as follows. At first the element having maximum absolute value is searched in i^{th} column below the principal diagonal and moved to the $(i+1, i)$ location by the permutation transformations to stabilize the reduction process. Then the elements of the i^{th} column below the first subdiagonal are annihilated by the elementary row transformations.

Column transformations are applied to both the permutation and elementary transformations to accomplish the similarity transformation. The information about the permutation transformations is stored in the vector *iperm* while the multipliers that define elementary transformations are stored in place of the eliminated elements of the matrix.

The steps described above are performed successively on columns from *low* to $(igh-2)$.

On return from the function the submatrix of the input matrix a has been reduced to upper Hessenberg form. The information about the permutation transformations is stored into locations from *low* to *igh* of vector *iperm*, other entries of the array are not used. The multipliers that have been used in the elementary transformations are stored into the *strict lower* triangle below the upper Hessenberg submatrix in matrix a.

Performance

The reduction process is numerically stable. The computed eigenvalues and eigenvectors will be exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

This function is generally faster than the function `corth`, since it involves fewer number of arithmetic operations. The disadvantage of the function `comhes` is the fact that the transformations used are not unitary and possibly may increase the norm of the reduced matrix, therefore affecting on the precision of the computed eigenvalues and eigenvectors. In practice such a matrices rarely arise.

Notes

Array a is in the format produced by the allocation function `csquare`.

The function references only the entries in $[low+1, igh-1]$ locations of the array *iperm*.

If the function `cbal` has not been used to balance the original matrix, the input parameters *low* and *igh* must be set to 0 and $(n-1)$ respectively.

Test

The test program is contained in `combakt.c` file. The example matrices are contained in `cmatrix1.xpd, cmatrix2.xpd, cmatrix3.xpd, cmatrix4.xpd, cmatrix5.xpd` files.

References

J. H. Wilkinson, The Algebraic Eigenvalue Problem, Published by: Clarendon Press, Oxford, 1965.

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 12, pp.349-368, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

10.2.2 CORTH Reduction by Unitary Transformations

Name

ccorth — Reduction by unitary transformations, single precision
zcorth — Reduction by unitary transformations, double precision

Synopsis

```
#include <ceispack.h>
void ccorth (n, low, igh, a, ort, cv)
int n, low, igh;
fcomplex **a, *ort, *cv;

void zcorth (n, low, igh, a, ort, cv)
int n, low, igh;
dcomplex **a, *ort, *cv;
```

n	— order of input/output matrix
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
a	— input (balanced)/output upper Hessenberg and transformations matrix
ort	— length n output array containing additional information about transformations
cv	— temporary storage array of length n

Description

Function corth reduces a submatrix located in the rows and columns from *low* to *igh* of given complex general matrix to upper Hessenberg form using and accumulating unitary similarity transformations.

The above submatrix is derived usually from balancing the original general matrix performed by the function `cbal`, that reduces the original matrix into the block-triangular form:

$$\begin{pmatrix} R & \vdots & W & \vdots & X \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & B & \vdots & Y \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \vdots & 0 & \vdots & T \end{pmatrix},$$

where matrix B is of order $(igh - low + 1)$ and located in rows and columns from *low* to *igh*, matrices R and T are upper triangular, other matrices are rectangular of the respective dimensions.

The reduction is performed as follows. At first the elements in i^{th} column below the principal diagonal are scaled to avoid possible underflow that would result in destroying the orthogonality of the transformation.

At the next phase the element in $(i+1, i)$ location is expanded by adding the square root of sum of squared magnitudes S_i of scaled elements. Then the elements of the i^{th} column below the principal diagonal form a vector w, that defines the Householder reflection:

$$Q = I - ww^*/h, \text{ where } h = w^*w/2,$$

I is the identity matrix, matrix Q is unitary and Hermitian. The transformation QFQ eliminates the elements in i^{th} column below the first subdiagonal.

As described above, Hessenberg reflections are applied successively to eliminate elements below the subdiagonal in columns from *low* to *igh* of the input matrix.

The product of Householder reflections is accumulated in the *strict lower* triangle below the upper Hessenberg matrix in the array *a* and in the vector *ort*.

On return from the function the array *a* contains the upper Hessenberg matrix. Information about the orthogonal transformations used in the reduction is stored in the *strict lower* triangle below the Hessenberg matrix.

The output vector *ort* contains the rest of information about the transformations.

Performance

The reduction process is numerically stable. The computed eigenvalues and eigenvectors would be exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Array *a* is in the format produced by the allocation function `csquare`.

Array *ort* should be allocated as a complex array of length at least *igh*. The function references only the entries in $[low+1, igh]$ locations of the array *ort*.

If the function `cbal` has not been used to balance the original matrix, the input parameters *low* and *igh* must be set to 0 and $(n-1)$ respectively.

Test

The test programs are contained in `cgt.c, cortbt.c` files. The example matrices are contained in `cmatrix1.xpd, cmatrix2.xpd, cmatrix3.xpd, cmatrix4.xpd, cmatrix5.c` files.

References

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 12, pp.349-368, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

10.3 Eigenvalues and Eigenvectors

10.3.1 COMLR Eigenvalues, LR Algorithm

Name

ccomlr — Forms eigenvalues by *LR* algorithm, single precision
zcomlr — Forms eigenvalues by *LR* algorithm, double precision

Synopsis

```
#include <ceispack.h>
void ccomlr (n, low, igh, h, w, ierr)
int n, low, igh;
fcomplex **h, *w;
int *ierr;

void zcomlr (n, low, igh, h, w, ierr)
int n, low, igh;
dcomplex **h, *w;
int *ierr;
```

n	— order of input matrix
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
h	— input matrix in upper Hessenberg form
w	— length n output array containing eigenvalues
ierr	— output variable address containing error completion code

Diagnostics

On output the variable `*ierr` is set to 0 for normal return. If the limit of `MAXITER*n` iterations is exceeded while the i^{th} eigenvalue is being computed, the error flag `*ierr` is set to $(i+1)$. Eigenvalues corresponding to indices $j \in [*ierr, n-1]$ would be correct.

Description

Function comlr finds the eigenvalues of a complex upper Hessenberg matrix by the modified *LR* method. This algorithm iterates a sequence of upper Hessenberg matrices that are similar to the original upper Hessenberg matrix. The sequence of upper Hessenberg matrices converges to an upper triangular form. The permutation similarity transformations are used, if necessary on each iteration to ensure stability of the iteration process.

The elementary similarity transformation are stored in the strict lower triangle of the matrix h together with those were used to reduce the general matrix to upper Hessenberg form.

A shift of origin before the next iteration generally improves the rate of convergence. The current upper Hessenberg form is tested before the next iteration for possible splitting to submatrices. The tolerance for the test is proportional to relative machine precision. If the matrix splits, the next iterations continue with the lower submatrix.

The value of the origin shift is chosen to be equal to the eigenvalue of the 2×2 lowermost principal minor of the currently iterating submatrix, which is closer to the second diagonal element of this minor. When the lowermost 2×2 minor of the currently iterating submatrix has finally split into two 1×1 minors, the element of lower minor is taken to be an eigenvalue of the original matrix. Iteration proceeds until the whole matrix has finally split into minors of first order.

If an eigenvalue has not converged within 10 consecutive iterations an additional shift of origin is applied to improve convergence.

The comlr function iterates the submatrix which is located in the rows and columns *low* through *igh* (see the description of the `cbal` function for more details about the variables *low* and *igh*) and defines the diagonal elements of the submatrices located in rows and columns 0 through *low* and $(igh + 1)$ through $(n - 1)$ as the eigenvalues of original matrix. These eigenvalues are computed exactly. If the `cbal` function has not been used the input variables *low* and *igh* must be set to 0 and $(n - 1)$ respectively.

On return from the function the array w contains eigenvalues of the original matrix.

The eigenvalues of a complex general matrix can also be found if the function `comhes` has been used to reduce the original general matrix into upper Hessenberg form. Generally it is recommended to use the function `cbal` to balance the original general matrix before the reduction to upper Hessenberg form.

Performance

It is expected that the LR algorithm will converge in most the cases. Appropriate shift of origin makes the convergence to be fast, while the permutation similarity transformations and additional shifts provide the stability of convergence in almost all the cases.

The computed eigenvalues are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

The accuracy of computed eigenvalues may decrease by the LR iteration process, which in turn may increase the norm of a triangularized matrix. Because of a possible loss of accuracy it is recommended to use the QR algorithm for determining the eigenvalues, in spite of the fact that the QR algorithm requires a larger number of arithmetic operations and therefore is slower than the LR one.

Notes

Matrix h is in the format produced by the allocation function `csquare`.

On input the lower triangle below the main subdiagonal of the input matrix h contains multipliers which were used in the reduction of a general matrix to upper Hessenberg form by the function `comhes`, if it has been used.

Matrix h is modified by the function comlr.

Test

The test programs are contained in `combakt.c, comlrt.c` files. The example matrices are contained in `cmatrix1.xpd, cmatrix2.xpd, cmatrix3.xpd, cmatrix4.xpd, cmatrix5.c` and `chessen1.xpd, chessen2.xpd, chessen3.xpd` files.

References

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 12, pp.369-376, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

10.3.2 COMQR Eigenvalues, QR Algorithm

Name

ccomqr — Forms eigenvalues by QR algorithm, single precision
zcomqr — Forms eigenvalues by QR algorithm, double precision

Synopsis

```
#include <ceispack.h>
void ccomqr (n, low, igh, h, w, ierr)
int n, low, igh;
fcomplex **h, *w;
int *ierr;

void zcomqr (n, low, igh, h, w, ierr)
int n, low, igh;
dcomplex **h, *w;
int *ierr;
```

n	— order of input matrix
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
h	— input matrix in upper Hessenberg form
w	— length n output array containing eigenvalues
ierr	— output variable address containing error completion code

Diagnostics

On output the variable `*ierr` is set to 0 for normal return. If the limit of `MAXITER*n` iterations is exceeded while the i^{th} eigenvalue is being computed, the error flag `*ierr` is set to $(i + 1)$. Eigenvalues corresponding to indices $j \in [*ierr, n - 1]$ would be correct.

Description

Function comqr computes all the eigenvalues of a complex matrix in upper Hessenberg form by the QR algorithm. This algorithm iterates a sequence of upper Hessenberg matrices that are unitary similar to the original matrix. The sequence of upper Hessenberg matrices converges to an upper triangular form.

The unitary similarity transformations are stored in the strict lower triangle of the matrix h together with those that were used to reduce a general matrix to upper Hessenberg form.

A shift of the origin before the next iteration generally improves the rate of convergence. The current upper Hessenberg form is tested before the next iteration for possible splitting to submatrices. The tolerance for the test is proportional to relative machine precision. If the matrix splits, the next iterations continue with the lower submatrix.

The value of the origin shift is chosen to be equal to the eigenvalue of the 2×2 lowermost principal minor of the currently iterating submatrix, which is closer to the second diagonal element of this minor. When the lowermost 2×2 minor of the currently iterating submatrix has finally split into two 1×1 minors, the element of the lower minor is taken to be an eigenvalue of the original matrix. Iteration proceeds until the whole matrix has finally split into minors of first order.

If an eigenvalue has not converged within 10 consecutive iterations an additional shift of origin is applied to improve convergence.

The comqr function iterates the submatrix which is located in rows and columns *low* through *igh* (see the description of the `cbal` function for more details about the variables *low* and *igh*) and defines the diagonal elements of the submatrices located in rows and columns 0 through *low* and $(igh + 1)$ through $(n - 1)$ as the eigenvalues of original matrix. These eigenvalues are computed exact. If the `cbal` function has not been used the input variables *low* and *igh* must be set to 0 and $(n - 1)$ respectively.

The subdiagonal elements are transformed real by the diagonal similarity transformation and kept real during the whole computation process.

On return from the function the array w contains the eigenvalues of the original matrix.

The eigenvalues of a complex general matrix can also be found if the function `corth` has been used to reduce the original general matrix into upper Hessenberg form. Generally it is recommended to use the function `cbal` to balance the original general matrix before the reduction to upper Hessenberg form.

Performance

The QR algorithm converges in most cases. Appropriate shift of the origin makes the convergence to be very fast.

The computed eigenvalues are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Matrix h is in the format produced by the allocation function `csquare`.

On input the lower triangle below the first subdiagonal of the input matrix h contains the information about the unitary transformations used in the reduction of a general matrix to upper Hessenberg form by the function `corth` if it has been used.

Matrix h is modified by the function comqr.

Test

The test programs are contained in `cortbt.c, comqrt.c` files. The example matrices are contained in `cmatrix1.xpd, cmatrix2.xpd, cmatrix3.xpd, cmatrix4.xpd, cmatrix5.c` and `chessen1.xpd, chessen2.xpd, chessen3.xpd` files.

References

J. G. F. Francis, Computer Journal, Vol. 4, pp.332-345, 1962.

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 12, pp.369-376, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

10.3.3 COMLR2 Eigenvalues and Eigenvectors, LR Algorithm

Name

ccomlr2 — Forms eigenvalues and eigenvectors, single precision
zcomlr2 — Forms eigenvalues and eigenvectors, double precision

Synopsis

```
#include <ceispack.h>
void ccomlr2 (n, low, igh, iperm, h, w, z, ierr)
int n, low, igh;
int *iperm;
fcomplex **h, *w, **z;
int *ierr;

void zcomlr2 (n, low, igh, iperm, h, w, z, ierr)
int n, low, igh;
int *iperm;
dcomplex **h, *w, **z;
int *ierr;
```

n	— order of input/output matrices
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
iperm	— input array of length n containing permutation indices
h	— input matrix in upper Hessenberg form
w	— length n output array containing eigenvalues
z	— square output matrix containing n column eigenvectors
ierr	— output variable address containing error completion code

Diagnostics

On output the variable `*ierr` is set to 0 for normal return. If the limit of `MAXITER*n` iterations is exceeded while the i^{th} eigenvalue is being computed, the error flag `*ierr` is set to $(i+1)$. Eigenvalues corresponding to indices $j \in [*ierr, n-1]$ would be correct. None of the eigenvectors have been determined yet.

Description

Function comlr2 finds the eigenvalues and corresponding eigenvectors of a complex upper Hessenberg matrix by the modified *LR* method. This algorithm iterates a sequence of upper Hessenberg matrices that are similar to the original upper Hessenberg matrix. The sequence of upper Hessenberg matrices converges to an upper triangular form. The permutation similarity transformations are used, if necessary on each iteration to ensure stability of the iteration process.

The elementary similarity transformation are stored in the strict lower triangle of the matrix *h* together with those used to reduce general matrix to upper Hessenberg form.

A shift of origin before the next iteration generally improves the rate of convergence. The current upper Hessenberg form is tested before the next iteration for possible splitting to submatrices. The tolerance for the test is proportional to relative machine precision. If the matrix splits, the next iterations continue with the lower submatrix.

The value of the origin shift is chosen to be equal to the eigenvalue of the 2×2 lowermost principal minor of the currently iterating submatrix, which is closer to the second diagonal element of this minor. When the lowermost 2×2 minor of the currently iterating submatrix has finally split into two 1×1 minors, the element of lower minor is taken to be an eigenvalue of the original matrix. Iteration proceeds until the whole matrix has finally split into minors of first order.

If an eigenvalue has not converged within 10 consecutive iterations an additional shift of origin is applied to improve convergence.

The comlr2 function iterates the submatrix which is located in rows and columns *low* through *igh* (see the description of the `cbal` function for more details about the variables *low* and *igh*) and defines the diagonal elements of the submatrices located in rows and columns 0 through *low* and $(igh + 1)$ through $(n - 1)$ as the eigenvalues of original matrix. These eigenvalues are computed exact. If the `cbal` function has not been used the input variables *low* and *igh* must be set to 0 and $(n - 1)$ respectively.

The eigenvectors of the finally converged triangular matrix are computed by the back substitution process and multiplied by the transformation matrix to be back transformed into the eigenvectors of the original matrix.

The eigenvalues and eigenvectors of a real general matrix can also be found if the function `comhes` is used to reduce the original general matrix into upper Hessenberg form and accumulate the similarity transformations in the strict lower triangle of array *h*.

Generally it is recommended to use the function `cbal` to balance the original general matrix before the reduction to upper Hessenberg form. Note that the function `cbabk2` must follow comlr2 if `cbal` has been used.

On return from the function the array *w* contains eigenvalues of the original matrix. The columns of matrix *z* contains the corresponding eigenvectors. They are unnormalized. The real part of the element `h[0][0]` contains the l_1 norm of the triangularized matrix.

Performance

The *LR* algorithm converges in most the cases. Appropriate shifts of origin make the convergence fast, while the permutation similarity transformations and additional shifts provide the stability of convergence in almost all the cases.

The computed eigenvalues and eigenvectors are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

The accuracy of computed eigenvalues and especially eigenvectors may decrease with the *LR* iteration process, which may increase the norm of a triangularized matrix. The l_1 norm of converged triangular matrix is stored in the real part of the `h[0][0]` entry of output array `h` to control the accuracy obtained.

Notes

Matrices h and z are in the format produced by the allocation function `csquare`.

On input the lower triangle below the main subdiagonal of the input matrix h contains multipliers which were used in the reduction of a general matrix to upper Hessenberg form by the function `comhes`, if it has been used. If the eigenvalues and eigenvectors of the Hessenberg matrix are desired, the strict lower triangle of the input matrix should be set to $(0 + i0)$.

Matrix h is modified by the function comlr2.

Input vector *iperm* contains the information about the permutation similarity transformations used in the reduction by the function `comhes`, if it has been used. Only the entries *low* through *igh* are referenced. If the eigenvectors of the Hessenberg matrix are desired these entries must be set to 0.

Test

The test program is contained in `comlr2t.c` file. The example matrices are contained in `chessen1.xpd, chessen2.xpd, chessen3.xpd` files.

References

G. Peters, J. H. Wilkinson, Numerical Mathematics, Vol. 16, pp.181-204, 1970.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

10.3.4 COMQR2 Eigenvalues and Eigenvectors, QR Algorithm

Name

ccomqr2 — Forms eigenvalues and eigenvectors, single precision
zcomqr2 — Forms eigenvalues and eigenvectors, double precision

Synopsis

```
#include <ceispack.h>
void ccomqr2 (n, low, igh, ort, h, w, z, ierr)
int n, low, igh;
fcomplex *ort, **h, *w, **z;
int *ierr;

void zcomqr2 (n, low, igh, ort, h, w, z, ierr)
int n, low, igh;
dcomplex *ort, **h, *w, **z;
int *ierr;
```

n	— order of input/output matrices
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
ort	— input array of length n containing information about reduction to Hessenberg form
h	— input matrix in upper Hessenberg form
w	— length n output array containing eigenvalues
z	— output eigenvector matrix
ierr	— output variable address containing error completion code

Diagnostics

On output the variable `*ierr` is set to 0 for normal return. If the limit of `MAXITER*n` iterations is exceeded while the i^{th} eigenvalue is being computed, the error flag `*ierr` is set to $(i + 1)$. Eigenvalues corresponding to indices $j \in [*ierr, n - 1]$ would be correct. None of the eigenvectors have yet been determined.

Description

Function comqr2 computes all the eigenvalues and corresponding eigenvectors of a complex matrix in upper Hessenberg form by the QR algorithm. This algorithm iterates a sequence of upper Hessenberg matrices that are unitary similar to the original matrix. The sequence of upper Hessenberg matrices converges to an upper triangular form.

The unitary similarity transformations are stored in the strict lower triangle of the matrix h together with those were used to reduce a general matrix to upper Hessenberg form.

A shift of origin before the next iteration generally improves the rate of convergence. The current upper Hessenberg form is tested before the next iteration for possible splitting to submatrices. The tolerance for the test is proportional to relative machine precision. If the matrix splits, the next iterations continue with the lower submatrix.

The value of the origin shift is chosen to be equal to the eigenvalue of the 2×2 lowermost principal minor of the currently iterating submatrix, which is closer to the second diagonal element of this minor. When the lowermost 2×2 minor of the currently iterating submatrix has finally split into two 1×1 minors, the element of lower minor is taken to be an eigenvalue of the original matrix. Iteration proceeds until the whole matrix has finally split into minors of first order.

If an eigenvalue has not converged within 10 consecutive iterations the additional shift of origin is applied to improve convergence.

The comqr2 function iterates the submatrix which is located in the rows and columns *low* through *igh* (see the description of the `cbal` function for more details about the variables *low* and *igh*) and defines the diagonal elements of the submatrices located in rows and columns 0 through *low* and $(igh + 1)$ through $(n - 1)$ as the eigenvalues of original matrix. These eigenvalues are computed exactly. If the `cbal` function has not been used the input variables *low* and *igh* must be set to 0 and $(n - 1)$ respectively.

The subdiagonal elements are transformed real by the diagonal similarity transformation and kept real during the whole computation process.

The eigenvectors of the final converged triangular matrix are computed by back substitution and multiplied by the transformation matrix to be back transformed into the eigenvectors of the original matrix.

The eigenvalues and eigenvectors of a real general matrix can also be found if the function `corth` has been used to reduce the original general matrix into upper Hessenberg form and accumulate the similarity transformations in the strict lower triangle of array h.

Generally it is recommended to use the function `cbal` to balance the original general matrix before the reduction to upper Hessenberg form. Note that the function `cbabk2` must follow `comqr2` if `cbal` has been used.

On return from the function the array w contains eigenvalues of the original matrix. The columns of matrix z contains the corresponding eigenvectors. They are unnormalized.

Performance

The QR algorithm converges in most the cases. Appropriate shifts of origin makes the convergence to be very fast.

The computed eigenvalues and eigenvectors are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Matrices h and z are in the format produced by the allocation function `csquare`.

On input the lower triangle below the first subdiagonal of the input matrix h contains the information about the unitary transformations used in the reduction of a general matrix to upper Hessenberg form by the function `corth` if it has been used. If the eigenvectors of the Hessenberg matrix are desired, the strict lower triangle of the input matrix can be arbitrary.

Matrix h is modified by the function comqr2.

Input vector *ort* contains the rest of information about the unitary transformations used in the reduction by the function `corth`, if it has been used. Only the entries *low* through *igh* are referenced. If the eigenvectors of the Hessenberg matrix are desired these entries must be set to $(0 + i0)$.

Test

The test program is contained in `comqr2t.c` file. The example matrices are contained in `chessen1.xpd, chessen2.xpd, chessen3.xpd` files.

References

J. G. F. Francis, Computer Journal, Vol. 4, pp.332-345, 1962.

G. Peters, J. H. Wilkinson, Numerical Mathematics, Vol. 16, pp.181-204, 1970.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

10.3.5 CINVIT Eigenvectors Coresponding to Specified Eigenvalues

Name

ccinvit — Finds eigenvectors corresponding to specified eigenvalues, single precision
zcinvit — Finds eigenvectors corresponding to specified eigenvalues, double precision

Synopsis

```
#include <ceispack.h>
void ccinvit (n, a, w, select, mm, m, mz, z, ierr, cm, cv)
int n;
fcomplex **a, *w, **z;
int *select, mm, *m, mz, *ierr;
fcomplex **cm, *cv;

void zcinvit (n, a, w, select, mm, m, mz, z, ierr, cm, cv)
int n;
dcomplex **a, *w, **z;
int *select, mm, *m, mz, *ierr;
dcomplex **cm, *cv;
```

n — order of input matrix a and number of rows in eigenvector matrix
a — input matrix in upper Hessenberg form
w — length n input array containing n eigenvalues
select — length n input array which specifies required eigenvectors
mm — input constant specifying upper bound for number of required eigenvectors
m — output variable address containing number of eigenvectors actually computed
mz — number of columns in eigenvector matrix z
z — rectangular output matrix containing column eigenvectors
ierr — output variable address containing error completion code
cm — order n temporary storage square matrix
cv — temporary storage array of length n

Diagnostics

The function returns 0 for normal return or 1 if an error exit has been made. The output error flag `*ierr` is set to:

- 0 for normal return.
- $-(2n+1)$ if more than mm eigenvectors have been specified. The output variable `*m` is set to mm and specifies the number of eigenvectors that have been found.
- $-k$ if the iteration corresponding to the k^{th} eigenvalue failed.
- $-(n+k)$ if both the error situations occurred.

Description

Function cinvit computes the eigenvectors of a complex matrix in upper Hessenberg form, which correspond to specified eigenvalues, using an inverse iteration technique. Computed eigenvectors are normalized so that the component of largest magnitude is set equal to 1.

The function cinvit solves a linear system

$$Uy = y_0$$

in order to obtain the eigenvector corresponding to i^{th} eigenvalue, where matrix U is the upper triangular multiplier in the LU decomposition of the matrix

$$F = B - \mu_i I, \text{ where}$$

matrix B is the leading order p submatrix of the input upper Hessenberg matrix, μ_i is the approximate eigenvalue of the matrix A. Starting from an initial vector, the approximate eigenvector is obtained by the back substitution process. The solution vector y is accepted as the eigenvector of the matrix F if its norm is larger than the norm of the initial vector y_0:

$$\sqrt{p}\,\varepsilon \|F\|_\infty \|y\|_\infty > \frac{\|y_0\|_\infty}{10}$$

If the acceptance test failed, up to p orthogonal initial vectors are tried successively to obtain an appropriate norm growth. The accepted vector is transformed to the eigenvector of the original matrix by adding $(n - p)$ zeros after its last component.

If all the initial vectors have not produced an acceptable eigenvector, the function cinvit set the error flag *ierr* to $-i$, where i is the index of corresponding eigenvalue, terminates iterations for this eigenvector and continues the computation process for the next eigenvalue. Components of the unaccepted eigenvector are set to zero. If such a situation occurs more than once, the output error flag would contain the last index of the eigenvalue for which the computation of an eigenvector failed.

The real parts of multiple eigenvalue or a group of close eigenvalues are perturbed by adding small multiples of $\varepsilon\|B\|$ in order to obtain linearly independent eigenvectors.

The input array w contains the eigenvectors of the input upper Hessenberg matrix. They are stored unordered except that eigenvalues of any submatrix of input upper Hessenberg matrix must have indices in the array w between the boundary indices of this submatrix. If the `comlr` or `comqr` functions have been used to find eigenvalues of an upper Hessenberg matrix, their output arrays of eigenvectors are arranged in the required order.

The entries of input vector *select* specify the eigenvectors to be found. The entry corresponding to the i^{th} eigenvalue must be set to a non-zero value if the i^{th} eigenvector is required or set to 0 otherwise.

On return from the function the real parts of multiple eigenvalues or a group of close eigenvalues may have been perturbed a little in order to compute linearly independent eigenvectors.

The output variable m is set to the number of eigenvectors actually found.

The columns of output the matrix z contain the eigenvectors. The eigenvectors are normalized so that the component of largest magnitude is 1. Any vector which does not satisfy the acceptance test is set to zero.

Performance

The rate of convergence of the inverse iterations is linear. The computed eigenvectors are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Matrices a and cm are in the format produced by the allocation function `csquare`.

The matrix z of dimension $n \times mz$ is in the format produced by the allocation function `cmatrix`.

The input matrix a and array *select* are preserved by the function cinvit.

Test

The test programs are contained in `combakt.c, cortbt.c, comlrt.c, comqrt.c` files. The example matrices are contained in `cmatrix1.xpd, cmatrix2.xpd, cmatrix3.xpd, cmatrix4.xpd, cmatrix5.xpd` and `chessen1.xpd, chessen2.xpd, chessen3.xpd` files.

References

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. N. Parlett, The Symmetric Eigenvalue Problem, Published by: Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632, 1980.

10.3.6 COMBAK Eigenvectors of Original Matrix

Name

ccombak — Forms eigenvectors of original matrix, single precision
zcombak — Forms eigenvectors of original matrix, double precision

Synopsis

```
#include <ceispack.h>
void ccombak (n, low, igh, a, iperm, m, z)
int n, low, igh;
fcomplex **a;
int *iperm, m;
fcomplex **z;

void zcombak (n, low, igh, a, iperm, m, z)
int n, low, igh;
dcomplex **a;
int *iperm, m;
dcomplex **z;
```

n	— order of input matrix *a* and number of rows in eigenvector array
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
a	— input upper Hessenberg and transformations matrix
iperm	— length *n* input integer array containing permutation indices
m	— number of columns in eigenvector matrix
z	— input/output column eigenvector matrix

Description

Function combak forms the eigenvectors of a complex general matrix by back transforming the eigenvectors of the corresponding upper Hessenberg matrix determined by the function `comhes`.

If submatrix B, stored in the input array a has been reduced to upper Hessenberg form by permutation and stabilized elementary similarity transformations:

$$H = S^{-1}BS,$$

where S is the product of the permutation matrices and elementary transformations, that have been stored in array *iperm* and in the *strict lower* triangle below upper Hessenberg matrix, then the function combak computes for each eigenvector x_i of the upper Hessenberg matrix the product:

$$z_i = Sx_i, \text{ where } i \in [0, m-1]$$

thus forming the eigenvectors of the original matrix B. The transformed eigenvectors are stored column-wise in the matrix z, overwriting the input eigenvectors of the upper Hessenberg matrix.

Performance

The computed eigenvectors are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Array a is in the format produced by the allocation function `csquare`.

The rectangular $n \times m$ matrix z is in the format produced by the allocation function `cmatrix`.

The function references only the entries in $[low + 1, igh - 1]$ locations of the array *iperm*.

If the function `cbal` has not been used to balance the original matrix, the input parameters *low* and *igh* must be set to 0 and $(n - 1)$ respectively.

Test

The test program is contained in `combakt.c` file. The example matrices are contained in `cmatrix1.xpd, cmatrix2.xpd, cmatrix3.xpd, cmatrix4.xpd, cmatrix5.xpd` files.

References

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 12, pp.349-368, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

10.3.7 CORTB Eigenvectors of Original Matrix

Name

ccortb — Forms eigenvectors of original matrix, single precision
zcortb — Forms eigenvectors of original matrix, double precision

Synopsis

```
#include <ceispack.h>
void ccortb (n, low, igh, a, ort, m, z, cv)
int n, low, igh;
fcomplex **a, *ort;
int m;
fcomplex **z, *cv;

void zcortb (n, low, igh, a, ort, m, z, cv)
int n, low, igh;
dcomplex **a, *ort;
int m;
dcomplex **z, *cv;
```

n	— order of input array a and number of rows in eigenvector array
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
a	— input upper Hessenberg and transformations matrix
ort	— input array containing rest of information about transformations
m	— number of columns in eigenvector array
z	— input/output eigenvector array
cv	— temporary storage array of length n

Description

Function cortb forms the eigenvectors of a complex general matrix by back transforming the eigenvectors of the corresponding upper Hessenberg matrix determined by the function `corth`.

If submatrix B, stored in the input array a has been reduced to upper Hessenberg form by unitary similarity transformations:

$$H = Q^T BQ,$$

where Q is the product of the unitary transformations, that has been stored in array *ort* and in the *strict lower* triangle below upper Hessenberg matrix, then the function cortb computes for each eigenvector x_i of the upper Hessenberg matrix the product:

$$z_i = Qx_i, \text{ where } i \in [0, m-1]$$

thus forming the eigenvectors of the original matrix B. The transformed eigenvectors are stored column-wise in the matrix z, overwriting the input eigenvectors of the upper Hessenberg matrix.

Performance

The computed eigenvectors are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Array a is in the format produced by the allocation function `csquare`.

The rectangular $n \times m$ matrix z is in the format produced by the allocation function `cmatrix`.

Array ort should be allocated as a complex array of length at least igh.

If the function `cbal` has not been used to balance the original matrix, the input parameters low and igh must be set to 0 and $(n-1)$ respectively.

Test

The test program is contained in `cortbt.c` file. The example matrices are contained in `cmatrix1.xpd, cmatrix2.xpd, cmatrix3.xpd, cmatrix4.xpd, cmatrix5.xpd` files.

References

R. S. Martin, J. H. Wilkinson, Numerical Mathematics, Vol. 12, pp.349-368, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

10.3.8 CBABK2 Eigenvectors of Non-balanced Matrix

Name

ccbabk2 — Forms eigenvectors of non-balanced matrix, single precision
zcbabk2 — Forms eigenvectors of non-balanced matrix, double precision

Synopsis

```
#include <ceispack.h>
void ccbabk2 (n, low, igh, scale, m, z)
int n, low, igh;
float *scale;
int m;
fcomplex **z;

void zcbabk2 (n, low, igh, scale, m, z)
int n, low, igh;
double *scale;
int m;
dcomplex **z;
```

n — number of rows in eigenvector matrix
low — input constant set to lower index of balanced submatrix
igh — input constant set to higher index of balanced submatrix
scale — length n input array containing scaling factors and permutation indices
m — number of columns in eigenvector matrix
z — input/output column eigenvector matrix

Description

Function cbabk2 forms the eigenvectors of a complex general matrix by back transforming the corresponding eigenvectors of a balanced matrix that has been determined by the function `cbal`.

The function `cbal` transforms the original matrix A into the balanced block-triangular matrix A_b by applying the similarity transformations:

$$A_b = D^{-1} P^T A P D,$$

where matrix P is the product of permutation transformations. Matrix D is the diagonal matrix of scaling factors, such that its locations within the indices $j \in [low, igh]$ are integer powers of the base machine arithmetic and all other locations are 1.

Function cbabk2 back transforms the eigenvectors of balanced matrix to the eigenvectors of the original matrix by forming the product:

$$z_i = PDx_i,$$

where x_i is the i^{th} eigenvector of balanced matrix, z_i is the corresponding eigenvector of the original matrix.

The back transformed eigenvectors are stored column-wise in the matrix z, overwriting the input eigenvectors of the balanced matrix.

Performance

Since the non-unitary entries in the diagonal matrix D are integer powers of the base machine arithmetic, the function cbabk2 produces no rounding errors.

Notes

The rectangular $n \times m$ matrix z is in the format produced by the allocation function `cmatrix`.

Test

The test programs are contained in `cgt.c, combakt.c, cortbt.c` files. The example matrices are contained in `cmatrix1.xpd, cmatrix2.xpd, cmatrix3.xpd, cmatrix4.xpd, cmatrix5.c` files.

References

B. N. Parlett, C. Reinsch, Numerical Mathematics, Vol. 13, pp.293-304, 1969.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

10.3.9 CBABKL Left-hand Eigenvectors of Non-balanced Matrix

Name

ccbabkl — Forms left-hand eigenvectors of non-balanced matrix, single precision
zcbabkl — Forms left-hand eigenvectors of non-balanced matrix, double precision

Synopsis

```
#include <ceispack.h>
void ccbabkl (n, low, igh, scale, m, z)
int n, low, igh;
float *scale;
int m;
fcomplex **z;

void zcbabkl (n, low, igh, scale, m, z)
int n, low, igh;
double *scale;
int m;
dcomplex **z;
```

n	— number of rows in left-hand eigenvector matrix
low	— input constant set to lower index of balanced submatrix
igh	— input constant set to higher index of balanced submatrix
scale	— length n input array containing scaling factors and permutation indices
m	— number of columns in left-hand eigenvector matrix
z	— input/output column left-hand eigenvector matrix

Description

Function cbabkl forms the left-hand eigenvectors of a complex general matrix by back transforming the corresponding left-hand eigenvectors of the balanced matrix determined by the function `cbal`.

The function `cbal` transforms the original matrix A into the balanced block-triangular matrix A_b by applying the similarity transformations:

$$A_b = D^{-1}P^TAPD,$$

where matrix P is the product of permutation transformations. Matrix D is the diagonal matrix of scaling factors, such that its locations within the indices $j \in [low, igh]$ are integer powers of the base machine arithmetic and all other locations are 1.

Function cbabkl back transforms the left-hand eigenvectors of balanced matrix to the left-hand eigenvectors of the original matrix by forming the product:

$$z_i = PD^{-1}x_i,$$

where x_i is the i^{th} left-hand eigenvector of balanced matrix, z_i is the corresponding left-hand eigenvector of the original matrix.

The back transformed left-hand eigenvectors are stored column-wise in the matrix z, overwriting the input left-hand eigenvectors of the balanced matrix.

Performance

Since the non-unitary entries in the diagonal matrix D are integer powers of the base the machine arithmetic, the function cbabkl produces no rounding errors.

Notes

The rectangular $n \times m$ matrix z is in the format produced by the allocation function `cmatrix`.

Function cbabkl is a modification of the C function `cbabk2`.

References

B. N. Parlett, C. Reinsch, Numerical Mathematics, Vol. 13, pp.293-304, 1969.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

Chapter 11

Hermitian Eigenproblem

11.1 Reduction to Symmetric Tridiagonal Form

11.1.1 HTRID3 Reduction by Unitary Transformations

Name

chtrid3 — Reduction and accumulation of transformations, single precision
zhtrid3 — Reduction and accumulation of transformations, double precision

Synopsis

```
#include <ceispack.h>
void chtrid3 (n, a, d, e, e2, tau)
int n;
float **a;
float *d, *e, *e2, **tau;

void zhtrid3 (n, a, t, e2, tau)
int n;
double **a;
double *d, *e, *e2, **tau;
```

n	— order of Hermitian input matrix/symmetric tridiagonal output matrix
a	— input Hermitian (lower triangle)/output transformations matrix
d	— output vector of length n containing diagonal elements of output matrix
e	— output vector of length n containing subdiagonal elements of output matrix
e2	— length n output array of squares of subdiagonal elements of output matrix
tau	— output $2 \times n$ diagonal transformation array

Description

Function htrid3 reduces a complex Hermitian matrix, stored as a single real array, to a real symmetric tridiagonal matrix by unitary similarity transformations.

The function performs a sequence of $(n-2)$ Householder reflections:

$$P_i = I - u_i u_i^*/h, \text{ where } h = u_i^* u_i/2$$

The sequence is applied to the input matrix row-wise, starting with the last row and continuing from bottom to top:

$$A_i = P_i A_{i-1} P_i, i \in [1, n-2]$$

Post-multiplications annihilates the elements in $(n-i)^{\text{th}}$ row (rows are numbered starting with 0 from top to bottom) to the left from the principal subdiagonal. Pre-multiplications would annihilate $(n-i)^{\text{th}}$ column (columns are numbered in similar manner) elements, but they do not actually applied.

The calculations proceed as follows. At first, the elements in $(n-i)^{\text{th}}$ row to the left from the principal subdiagonal are scaled to avoid possible underflow that would result in destroying the orthogonality of the transformation. The sum of squares S_i of scaled elements is taken as the square of the i^{th} subdiagonal element of the symmetric tridiagonal form. A square root $S_i^{1/2}$ with the sign set opposite to a transformed subdiagonal element of the original matrix a is taken then as the value of this subdiagonal element.

The last i^{th} reflection eliminates these elements in the $(n-i)^{\text{th}}$ row. The symmetric elements in the corresponding column of the matrix would also be eliminated if the pre-multiplication were applied.

The transformed subdiagonal element is then further transformed by diagonal unitary similarity transformation to be exactly real. The diagonal transformations are accumulated in the output array *tau*.

The information about Householder transformations used in the reduction process is accumulated in the matrix a.

Function htrid3 references only the diagonal and subdiagonal elements of the resultant symmetric tridiagonal matrix. Transformed diagonal and subdiagonal elements are stored into output vectors d and e, therefore preserving the information about the transformations used in the reduction process.

The input matrix a contains the lower triangle of the complex Hermitian matrix. The real parts of the matrix elements are stored in the *full lower* triangle of the array a. The imaginary parts are stored in the *transposed* positions of the *strict upper* triangle of the array a. There is no storage required for zero imaginary parts of the diagonal elements of the Hermitian matrix.

Suppose that a Hermitian matrix of order 4 is to be reduced by the function htrid3. Then the input array a should be set as follows:

$$\begin{pmatrix} \mathrm{Re}(a_{00}) & \mathrm{Im}(a_{10}) & \mathrm{Im}(a_{20}) & \mathrm{Im}(a_{30}) \\ \mathrm{Re}(a_{10}) & \mathrm{Re}(a_{11}) & \mathrm{Im}(a_{21}) & \mathrm{Im}(a_{31}) \\ \mathrm{Re}(a_{20}) & \mathrm{Re}(a_{21}) & \mathrm{Re}(a_{22}) & \mathrm{Im}(a_{32}) \\ \mathrm{Re}(a_{30}) & \mathrm{Re}(a_{31}) & \mathrm{Re}(a_{32}) & \mathrm{Re}(a_{33}) \end{pmatrix}$$

On return from the function the output vectors d and e contain respectively the diagonal and subdiagonal elements of resulted symmetric tridiagonal matrix. The subdiagonal elements has been stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array has been set to 0.

The array $e2$ contains the squares of the corresponding subdiagonal elements of resulted symmetric tridiagonal matrix in its last $(n-1)$ locations. The entry `e2[0]` has been set to 0.

Array *tau* contains the information about the diagonal transformations used in the reduction process.

Performance

The reduction process is numerically stable since it is based on unitary transformations. The resultant symmetric tridiagonal matrix is unitarily similar to a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Array a is in the format produced by the allocation function `fsquare`.

Arrays e and $e2$ may coincide if the squares are not required.

The rectangular $2 \times n$ array *tau* is in the format produced by the allocation function `fmatrix`.

Test

The test program is contained in `ch3t.c` file. The example matrices are contained in `hermit30.xpd, hermit31.xpd` files.

References

D. J. Mueller, Numerical Mathematics, Vol. 8, pp.72-92, 1968.

R. S. Martin, C. Reinsch, J. H. Wilkinson, Numerical Mathematics, Vol. 11, pp.181-195, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

11.1.2 HTRIDI Reduction by Unitary Transformations

Name

chtridi — Reduction and accumulation of transformations, single precision
zhtridi — Reduction and accumulation of transformations, double precision

Synopsis

```
#include <ceispack.h>
void chtridi (n, a, d, e, e2, tau)
int n;
fcomplex **a;
float *d, *e, *e2, **tau;

void zhtridi (n, a, d, e, e2, tau)
int n;
dcomplex **a;
double *d, *e, *e2, **tau;
```

n	— order of Hermitian input matrix/symmetric tridiagonal output matrix
a	— input Hermitian (lower triangle)/output transformation matrix
d	— output vector of length n containing diagonal elements of output matrix
e	— output vector of length n containing subdiagonal elements of output matrix
e2	— length n output array of squares of subdiagonal elements of output matrix
tau	— output $2 \times n$ diagonal transformation array

Description

Function htridi reduces a complex Hermitian matrix to a real symmetric tridiagonal form by unitary similarity transformations.

The function performs a sequence of $(n-2)$ Householder reflections:

$$P_i = I - u_i u_i^* / h, \text{ where } h = u_i^* u_i / 2$$

The sequence is applied to the input matrix row-wise, starting with the last row and continuing from bottom to top:

$$A_i = P_i A_{i-1} P_i, i \in [1, n-2]$$

Post-multiplications annihilates the elements in $(n-i)^{\text{th}}$ row (rows are numbered starting with 0 from top to bottom) to the left from the principal subdiagonal. Pre-multiplications would annihilate $(n-i)^{\text{th}}$ column (columns are numbered in similar manner) elements, but they do not actually applied.

The calculations proceed as follows. At first, the elements in $(n-i)^{\text{th}}$ row to the left from the principal subdiagonal are scaled to avoid possible underflow that would result in destroying the orthogonality of the transformation. The sum of squares S_i of scaled elements is taken as the square of the i^{th} subdiagonal element of the symmetric tridiagonal form. A square root $S_i^{1/2}$ with the sign set opposite to a transformed subdiagonal element of the original matrix a is taken then as the value of this subdiagonal element.

The last i^{th} reflection eliminates these elements in the $(n-i)^{\text{th}}$ row. The symmetric elements in the corresponding column of the matrix would also be eliminated if the pre-multiplication were applied.

The transformed subdiagonal element is then further transformed by diagonal unitary similarity transformation to be exactly real. The diagonal transformations are accumulated in the output array *tau*.

The information about Householder transformations used in the reduction process is stored in the *strict lower* triangle and *imaginary* parts of diagonal elements of the original matrix *a*.

Function htridi references only the *strict lower* triangle and *real* parts of diagonal elements of the input matrix. The transformed diagonal and subdiagonal elements are stored in the vectors *d* and *e*, therefore preserving the *strict upper* triangle and *real* parts of diagonal elements of the matrix *a*.

On return from the function the output vectors *d* and *e* contain respectively the diagonal and subdiagonal elements of resulted symmetric tridiagonal matrix. The subdiagonal elements has been stored in the *last* $(n-1)$ locations of the vector *e*. The first entry of the array has been set to 0.

The array *e*2 contains the squares of the corresponding subdiagonal elements of the resultant symmetric tridiagonal matrix in its last $(n-1)$ locations. The entry `e2[0]` has been set to 0.

Array *tau* contains the information about the diagonal transformations used in the reduction process.

Performance

The reduction process is numerically stable since it is based on unitary transformations. The resultant symmetric tridiagonal matrix is unitarily similar to a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Matrix a is in the format produced by the allocation functions `csquare`, `trngl_cmatrix` or `chermit`. If the matrix is allocated by the function `trngl_cmatrix` the *lower* triangle must be specified.
Arrays e and $e2$ may coincide if the squares are not required.

The rectangular $2 \times n$ array *tau* is in the format produced by the allocation function `fmatrix`.

Test

The test program is contained in `cht.c` file. The example matrices are contained in `hermiti0.xpd, hermiti1.xpd` files.

References

D. J. Mueller, Numerical Mathematics, Vol. 8, pp.72-92, 1968.

R. S. Martin, C. Reinsch, J. H. Wilkinson, Numerical Mathematics, Vol. 11, pp.181-195, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

11.2 Eigenvectors

11.2.1 HTRIB3 Eigenvectors of Original Matrix

Name

chtrib3 — Forms eigenvalues of Hermitian matrix, single precision
zhtrib3 — Forms eigenvalues of Hermitian matrix, double precision

Synopsis

```
#include <ceispack.h>
void chtrib3 (n, a, tau, m, zr, zi)
int n;
float **a, **tau;
int m;
float **zr, **zi;

void zhtrib3 (n, a, tau, m, zr, zi)
int n;
double **a, **tau;
int m;
double **zr, **zi;
```

n	— order of input matrix, number of columns in eigenvector matrices
a	— input unitary transformation matrix
tau	— input $2 \times n$ diagonal transformation array
m	— number of eigenvectors to be back transformed
zr	— input eigenvectors/output real parts of transformed eigenvector (row) matrix
zi	— output imaginary parts of transformed eigenvector (row) matrix

Description

Function htrib3 forms the eigenvectors of a complex Hermitian matrix by back transforming the eigenvectors of the corresponding real symmetric tridiagonal matrix determined by the function `htrid3`.

The function calculates the matrix product PDZ, where matrix P is the product of $(n-2)$ Householder reflections and matrix D is unitary diagonal matrix, that were accumulated in the reduction of a complex Hermitian matrix A to a symmetric tridiagonal form T:

$$T = D^*Q^*AQD$$

by the function `htrid3`.

The last entry of all the eigenvectors computed by the function htrib3 is rendered real. Since the transformations used by the function htrib3 are unitary, the Euclidean norm of the eigenvectors is kept unchanged.

The input matrix a contains information about the unitary transformations used in the reduction by the function `htrid3`.

Input array tau contains the diagonal similarity transformations.

Input matrix zr contains in its first m rows the transposed (row) eigenvectors of reduced real symmetric tridiagonal matrix to be back transformed.

On return from the function the matrix zr contains the real parts of the transformed eigenvectors (those of the original Hermitian matrix) in its first m rows.

Matrix zi contains the imaginary parts of the transformed eigenvectors of the original Hermitian matrix in its first m rows.

Performance

The computed eigenvectors are close to the exact eigenvectors of a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Array a is in the format produced by the allocation functions `fsquare`.

The rectangular $m \times n$ matrices zr, zi are in the format produced by the allocation function `fmatrix`. The number of rows in these arrays must not be less than m.

The rectangular $2 \times n$ array tau is in the format produced by the allocation function `fmatrix`.

Test

The test program is contained in `ch3t.c` file. The example matrices are contained in `hermit30.xpd, hermit31.xpd` files.

References

D. J. Mueller, Numerical Mathematics, Vol. 8, pp.72-92, 1968.

R. S. Martin, C. Reinsch, J. H. Wilkinson, Numerical Mathematics, Vol. 11, pp.181-195, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

11.2.2 HTRIBK Eigenvectors of Original Matrix

Name

chtribk — Forms eigenvalues of Hermitian matrix, single precision
zhtribk — Forms eigenvalues of Hermitian matrix, double precision

Synopsis

```
#include <ceispack.h>
void chtribk (n, a, tau, m, zr, zi)
int n;
fcomplex **a;
float **tau;
int m;
float **zr, **zi;

void zhtribk (n, a, tau, m, zr, zi)
int n;
dcomplex **a;
double **tau;
int m;
double **zr, **zi;
```

n — order of input matrix, number of columns in eigenvector matrices
a — input unitary transformation matrix
tau — input $2 \times n$ diagonal transformation array
m — number of eigenvectors to be back transformed
zr — input eigenvectors/output real parts of transformed eigenvector (row) matrix
zi — output imaginary parts of transformed eigenvector (row) matrix

Description

Function htribk computes the eigenvectors of a complex Hermitian matrix by back transforming the eigenvectors of the corresponding real symmetric tridiagonal matrix determined by the function `htridi`.

The function calculates the matrix product PDZ, where matrix P is the product of $(n-2)$ Householder reflections and matrix D is unitary diagonal matrix, that were accumulated in the reduction of a complex Hermitian matrix A to a symmetric tridiagonal form T:

$$T = D^{*}Q^{*}AQD$$

by the function `htridi`.

The last entry of all the eigenvectors computed by the function htribk is rendered real. Since the transformations used by the function htribk are unitary, the Euclidean norm of the eigenvectors is kept unchanged.

The input matrix *a* contains information about the unitary transformations used in the reduction by the function `htridi` in its full lower triangle except for the real parts of diagonal elements.

Input array *tau* contains the diagonal similarity transformations.

Input matrix *zr* contains in its first *m* rows the transposed (row) eigenvectors of reduced real symmetric tridiagonal matrix to be back transformed.

On return from the function the matrix *zr* contains the real parts of the transformed eigenvectors (those of the original Hermitian matrix) in its first *m* rows.

Matrix *zi* contains the imaginary parts of the transformed eigenvectors of the original Hermitian matrix in its first *m* rows.

Performance

The computed eigenvectors are close to the exact eigenvectors of a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Matrix *a* is in the format produced by the allocation functions `csquare`, `trngl_cmatrix` or `chermit`. If the matrix is allocated by the function `trngl_cmatrix` the *lower* triangle must be specified.

The rectangular $m \times n$ matrices *zr*, *zi* are in the format produced by the allocation function `fmatrix`. The number of rows in these arrays must not be less than *m*.

The rectangular $2 \times n$ array *tau* is in the format produced by the allocation function `fmatrix`.

Test

The test program is contained in `cht.c` file. The example matrices are contained in `hermiti0.xpd, hermiti1.xpd` files.

References

D. J. Mueller, Numerical Mathematics, Vol. 8, pp.72-92, 1968.

R. S. Martin, C. Reinsch, J. H. Wilkinson, Numerical Mathematics, Vol. 11, pp.181-195, 1968.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

Chapter 12

Matrix Decomposions

12.1 Matrix Equations

12.1.1 HQRDC Quasi-Triangular Decomposition

Name

shqrdc — Quasi-triangular decomposition, real matrix, single precision
dhqrdc — Quasi-triangular decomposition, real matrix, double precision

Synopsis

```
#include <ceispack.h>
void shqrdc (n, h, w, z, rv, ierr)
int n;
float **h, **z;
fcomplex *w;
float *rv;
int *ierr;

void dhqrdc (n, h, w, z, rv, ierr)
int n;
double **h, **z;
dcomplex *w;
double *rv;
int *ierr;
```

n	— order of input/output matrices
h	— input upper Hessenberg/output quasi-triangular matrix
w	— length n output array containing eigenvalues
z	— input transformations (identity)/output orthogonal transformation matrix
rv	— temporary storage array of length n
ierr	— output variable address containing error completion code

Diagnostics

On output the variable `*ierr` is set to zero for normal return. If the limit of `MAXITER*n` iterations is exceeded while the i^{th} eigenvalue is being computed, the error flag `*ierr` is set to $(i+1)$. Eigenvalues corresponding to indices $k \in [*ierr, n-1]$ would be correct.

Description

Function hqrdc reduces the original matrix H in upper Hessenberg form to the upper quasi-triangular matrix, such that its principal diagonal contains blocks of first and second order only, using the QR algorithm:

$$Q^T HQ = R,$$

where matrix Q is orthogonal, matrix R is upper quasi-triangular. The diagonal blocks of first order are the real eigenvalues of the original matrix, while the diagonal blocks of second order correspond to the pairs of complex conjugate eigenvalues of the original matrix.

For more details on the QR algorithm see the description of the functions `hqr` and `hqr2`.

If the decomposition of a Hessenberg matrix is required, the input matrix z must be set to the identity matrix. The decomposition of a general matrix can also be obtained if the general matrix has been reduced by orthogonal transformations to upper Hessenberg form using the function `orthes` and the transformations have been accumulated in the matrix z by the function `ortran`. The original general matrix should not be balanced. The input parameters *low, igh* to the functions `orthes` and `ortran` should be set to 0 and $(n-1)$ respectively.

On return from the function the orthogonal transformation matrix Q is stored in the array z, matrix R is stored in place of the input matrix in the array h.

The function hqrdc could be applied to find the solution of a matrix equation. Let matrices A, B and C be real $n \times n$, $m \times m$ and $n \times m$ matrices respectively and $n \times m$ matrix X is to be found which satisfies the matrix equation:

$$AX + XB = C$$

If the matrix coefficients A^T and B have been reduced to upper quasi-triangular form:

$$A^T = UL^T U^T \text{ and } B = VRV^T,$$

the matrix equation can be rewritten in simpler form having lower L and upper R quasi-triangular matrix coefficients:

$$LY + YR = F,$$

where $Y = U^T XV$ and $F = U^T CV$. The columns of the unknown matrix Y can be found from the equation by the forward substitution method, and the solution matrix X can be computed by the following formula:

$$X = UYV^T$$

The forward substitution process needs be modified to take into account the diagonal blocks of order 2 in the quasi-triangular coefficients.

Performance

The QR algorithm converges in most the cases. Appropriate shifts of origin make convergence very fast.

The computed eigenvalues are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Matrices h and z are in the format produced by the allocation function `fsquare`.

Test

The test program is contained in `hqrdct.c` file. The example matrices are contained in `rmatrix1.xpd, rmatrix2.xpd, rmatrix3.xpd` and `rhessen1.xpd, rhessen2.xpd, rhessen3.xpd, rhessen4.xpd, rhessen5.xpd, rhessen6.xpd` files.

References

J. G. F. Francis, Computer Journal, Vol. 4, pp.332-345, 1962.

G .Peters, J. H. Wilkinson, Numerical Mathematics, Vol. 16, pp.181-204, 1971.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

H. R. Bartels, G. W. Stuart, Communication of the ACM, vol. 15, pp.820-826, 1972.

A. N. Beavers, E. D. Denman, SIAM Journal of Applied Mathematics, vol. 29, pp.416-421, 1975.

P. A. Van Dooren, SIAM Journal on Scientific Statistical Computations, vol. 2, pp.121-135, 1981.

G. H. Golub, S. Nash, C. A. Van Loan, IEEE Transactions on Automatic Control, vol. 24, pp.909-913, 1979.

P. Harlander, Information Science, vol. 4, pp.35-40, 1972.

W. D. Hoskins, D. S. Meek, D. J. Walton, BIT, vol. 17, pp.184-190, 1977.

G. Kitagava, International Journal on Control, vol. 25, pp.745-753, 1977.

D. L. Kleinman, IEEE Transactions on Automatic Control, vol. 13, pp.114-115, 1968.

A. J. Laub, IEEE Transactions on Automatic Control, vol. 24, pp.913-921, 1979.

T. Pappas, A. J. Laub, N. R. Sandell Jr., IEEE Transactions on Automatic Control, vol. 25, pp.631-641, 1980.

D. Rothschild, A. Jameson, International Journal on Control, vol. 11, pp.181-198, 1970.

N. R. Sandell Jr., IEEE Transactions on Automatic Control, vol. 19, pp.254-255, 1974.

J. Snyders, M. Zakai, SIAM Journal of Applied Mathematics, vol. 18, pp.704-714, 1970.

12.2 Singular Value Decomposition

12.2.1 Real Matrix SVD

Name

ssvd — Singular value decomposition, real matrix, single precision
dsvd — Singular value decomposition, real matrix, double precision

Synopsis

```
#include <ceispack.h>
void ssvd (m, n, a, s, e, matu, u, matv, v, ierr, rv)
int m, n;
float **a, *s, *e;
int matu;
float **u;
int matv;
float **v;
int *ierr;
float *rv;

void dsvd (m, n, a, s, e, matu, u, matv, v, ierr, rv)
int m, n;
double **a, *s, *e;
int matu;
double **u;
int matv;
double **v;
int *ierr;
double *rv;
```

m — number of rows in the input/output matrices a and u
n — number of columns in the input/output matrices a and u and order of array v
a — rectangular input matrix
s — length $min(m+1,n)$ output array containing $\min(m,n)$ singular values
e — temporary storage array of length n
matu — flag which causes computation of matrix u
u — rectangular output matrix containing n column left singular vectors
matv — flag which causes computation of matrix v
v — output square matrix containing n column right singular vectors
ierr — output variable address containing error completion code
rv — temporary storage array of length n

Diagnostics

Function returns

- 0 for normal return.
- 1 if the k^{th} singular value has not been determined after `MAXITER` iterations. Output flag `*index` has been set to k. Singular values corresponding to indices $i \in [k+1, n-1]$ would be correct. The columns of arrays u and v, corresponding to indices of correct singular values would be also correct.

Description

Function svd determines the singular value decomposition of real $m \times n$ matrix A:

$$A = U\Sigma V^T \text{ where } \Sigma \text{ is diagonal}$$

$$\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n),$$

and its elements are the singular values of matrix A. The columns of matrices U and V contain respectively left and right singular vectors.

The computations proceed as follows. At first, the original matrix is reduced to a bidiagonal form by two sequences of Householder transformations:

$$J = P^T AQ,$$

where P and Q are the products of Householder transformations. Bidiagonal matrix J has the same singular values as the original matrix A:

$$J = G\Sigma H^T,$$

so that the left and right transformation matrices in the decomposition of matrix A can be defined as follows:

$$U = PG \text{ and } V = QH$$

The singular values of the matrix J can be found as positive square roots of the eigenvalues of the product $J^T J$.

At the next step the modified QR algorithm is applied to the bidiagonal form J in order to diagonalize symmetric tridiagonal matrix $J^T J$.

This algorithm iterates a sequence of symmetric tridiagonal matrices which are orthogonally similar to the original symmetric tridiagonal matrix $J^T J$. This sequence converges to a diagonal matrix. A shift of origin generally improves the rate of convergence.

The current symmetric tridiagonal matrix is verified before the next iteration for possible splitting to submatrices. If the matrix splits, the subsequent iterations continue with the lowermost submatrix. The tolerance in the test for splitting is commensurate to relative machine precision.

The left and right hand transformations are accumulated in the arrays u and v if the input parameters *matu, matv* are set to any non-zero value. When they are set to 0 only the singular values are computed.

The value of the origin shift at the current iteration is defined as the eigenvalue of the lowermost 2×2 principal minor. When the lowermost 1×1 minor splits from the rest of the matrix, its eigenvalue is considered to be an eigenvalue of the original matrix $J^T J$. The iterations proceed with the remaining submatrix until the matrix has completely split into blocks of first order, or equivalently, when it has become a diagonal matrix.

On return from the function vector s contains in its first $\min(m, n)$ locations the singular values of the input matrix. They are *unordered.* In a case $n > m + 1$ the extra $(m + 1)^{\text{th}}$ element is set to 0.

If specified by the input flag *matu*, matrix u contains column left singular vectors. It may coincide with the input array a.

If specified by the input flag *matv*, matrix v contains column right singular vectors. The matrix may coincide with the a matrix if $m \geq n$.

Performance

Appropriate shift of the origin makes the convergence of the algorithm global. The rate of convergence is cubic in almost all the cases.

The computed singular values and singular vectors are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Matrices a, u are in the format produced by the allocation function `fmatrix`. Matrix v is expected to be in the format produced by the allocation function `fsquare`.
Matrix a is modified by the function `ssvd`.

Test

The test program is contained in `svdt.c` file, the example matrices are contained in `rsvd1.xpd, rsvd2.xpd` files.

References

G. Golub, C. Reinsch, Numerical Mathematics, Vol. 14, pp.403-420, 1970.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

12.2.2 Complex Matrix SVD

Name

csvd — Singular value decomposition, complex matrix, single precision
zsvd — Singular value decomposition, complex matrix, double precision

Synopsis

```
#include <ceispack.h>
void csvd (m, n, a, s, e, matu, u, matv, v, ierr, cv)
int m, n;
fcomplex **a, *s, *e;
int matu;
fcomplex **u;
int matv;
fcomplex **v;
int *ierr;
fcomplex *cv;

void zsvd (m, n, a, s, e, matu, u, matv, v, ierr, cv)
int m, n;
dcomplex **a, *s, *e;
int matu;
dcomplex **u;
int matv;
dcomplex **v;
int *ierr;
dcomplex *cv;
```

m	— number of rows in input/output matrices a and u
n	— number of columns in input/output matrices a and u and order of matrix v
a	— rectangular input matrix
s	— length $\min(m+1, n)$ output array containing $\min(m, n)$ singular values
e	— temporary storage array of length n
matu	— flag which causes computation of matrix u
u	— rectangular output matrix containing n column left singular vectors
matv	— flag which causes computation of matrix v
v	— square output matrix containing n column right singular vectors
ierr	— output variable address containing error completion code
cv	— temporary storage array of length n

Diagnostics

Function returns

- 0 for normal return.

- 1 if the k^{th} singular value has not been determined after `MAXITER` iterations. Output flag `*index` has been set to k. Singular values corresponding to indices $i \in [k+1, n-1]$ would be correct. The columns of arrays u and v, corresponding to indices of correct singular values would be correct.

Description

Function csvd determines the singular value decomposition of complex $m \times n$ matrix A:

$$A = U \Sigma V^* \text{ where } \Sigma \text{ is diagonal}$$

$$\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n),$$

and its elements are the singular values of matrix A. Matrix V^* is the Hermitian conjugate of the matrix V. Columns of matrices U and V contain respectively left and right singular vectors.

The computations proceed as follows. At first, the original matrix is reduced to a bidiagonal form by two sequences of unitary Householder transformations:

$$J = P^* A Q,$$

where P and Q are the products of Householder transformations. Bidiagonal matrix J has the same singular values as the original matrix A:

$$J = G \Sigma H^T,$$

so that the left and right transformation matrices in the decomposition of matrix A can be defined as follows:

$$U = PG \text{ and } V = QH$$

The singular values of the matrix J can be found as positive square roots of the eigenvalues of the product $J^T J$.

Finally the diagonal and bidiagonal locations of the bidiagonal matrix J are rendered real by two diagonal transformations.

At the next step the modified QR algorithm is applied to the real bidiagonal form J in order to diagonalize symmetric tridiagonal matrix $J^T J$.

This algorithm iterates a sequence of symmetric tridiagonal matrices which are orthogonally similar to the original symmetric tridiagonal matrix $J^T J$. This sequence converges to a diagonal matrix. A shift of origin generally improves the rate of convergence.

The current symmetric tridiagonal matrix is verified before the next iteration for possible splitting to submatrices. If the matrix splits, the subsequent iterations continue with the lowermost submatrix. The tolerance in the test for splitting is commensurate to relative machine precision.

The left and right hand transformations are accumulated in the arrays u and v if the input parameters *matu, matv* are set to any non-zero value. When they are set to 0 only the singular values are computed.

The value of the origin shift at the current iteration is defined as the eigenvalue of the lowermost 2×2 principal minor. When the lowermost 1×1 minor splits from the rest of the matrix, its eigenvalue is considered to be an eigenvalue of the original matrix $J^T J$. The iterations proceed with the remaining submatrix until the matrix has completely split into blocks of first order, or equivalently, when it has become a diagonal matrix.

On return from the function vector s contains $\min(m, n)$ singular values. They are *unordered* and stored in real parts of corresponding entries. Their correponding immaginary parets are set to zero. In a case $n > m + 1$ the extra $(m + 1)^{\text{th}}$ element is set to 0.

If specified by the input flag *matu*, matrix u contains column left singular vectors. It may coincide with the input array a.

If specified by the input flag *matv*, matrix v contains column right singular vectors. The matrix may coincide with the a matrix if $m \geq n$.

Performance

Appropriate shift of the origin makes the convergence of the algorithm global. The rate of convergence is cubic in almost all the cases.

The computed singular values and singular vectors are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Matrices a, u are in the format produced by the allocation function `cmatrix`. Matrix v is in the format produced by the allocation function `csquare`.

Matrix a is modified by the function `csvd`.

Test

The test program is contained in `csvdt.c` file. The example matrices are contained in `csvd1.xpd, csvd2.xpd` files.

References

G. Golub, C. Reinsch, Numerical Mathematics, Vol. 14, pp.403-420, 1970.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

12.2.3 MINFIT Linear Least Square Problem Solution

Name

sminfit — Linear least squares problem solution of minimal norm, single precision
dminfit — Linear least squares problem solution of minimal norm, double precision

Synopsis

```
#include <ceispack.h>
void sminfit (m, n, a, s, e, ip, b, ierr, rv)
int m, n;
float **a, *s, *e;
int ip;
float **b, *rv;
int *ierr;

void dminfit (m, n, a, s, e, ip, b, ierr, rv)
int m, n;
double **a, *s, *e;
int ip;
double **b, *rv;
int *ierr;
```

m	— number of rows in input matrices a and b
n	— number of columns in matrix a and row dimension of a if $n > m$
a	— rectangular input matrix/output matrix of right singular vectors
s	— output array containing n singular values
e	— temporary storage array of length n
ip	— number of columns in input matrix b
b	— input column constant vectors matrix/output matrix product U^TB
ierr	— output variable address containing error completion code
rv	— temporary storage array of length n

Diagnostics

Function returns

- 0 for normal return.
- 1 if the k^{th} singular value has not been determined after `MAXITER` iterations. Output flag `*index` has been set to k. Singular values corresponding to indices $i \in [k+1, n-1]$ would be correct. The columns of V corresponding to indices of correct singular values and the rows of UB^T corresponding to this indices would also be correct.

Description

Function minfit computes the singular value decomposition:

$$A = U\Sigma V^T$$

of a real $m \times n$ matrix A, forming the product $C = U^T B$ instead of the matrix U to determine further the minimal norm solution of the linear system:

$$AX = B,$$

where B is the matrix of constant column right-hand vectors. The matrix of right-hand transformations is accumulated in place of the input matrix A. The matrix of left-hand transformations is accumulated as the product $U^T B$ in place of the input constant matrix.

On return from the function the matrix a has been overwritten by the orthogonal $n \times n$ matrix of right-hand transformations in the decomposition. The vector w contains the singular values. They are unordered. The input matrix b has been overwritten by the matrix product $U^T B$.

For more details on the algorithm for singular value decomposition see the description of the `svd` function.

The function minfit can be applied to find a minimal norm solution of a linear least square problem. Vector x is the least squares solution of the system:

$$Ax = b$$

if it minimizes the Euclidean norm of the residual vector

$$\|b - Ax\|_2 = \min$$

When the coefficient matrix A of the linear system is not of full rank then the system does not have a unique least square solution. To obtain a unique solution x is usuallly chosen from all x as the one which has the minimal Euclidean norm

$$\|x\|_2 = \min$$

Defining the threshold θ as described in the next section **Application of SVD and MINFIT functions** and applying the singular value decomposition of the coefficient matrix A the minimal norm least square solution can be computed as:

$$x = V\Sigma^\dagger U^T b$$

where the diagonal matrix $\Sigma^\dagger$ is defined having its entries:

$$\sigma_i^\dagger = \begin{cases} \frac{1}{\sigma_i} & \text{when } \sigma_i > \theta \\ 0 & \text{otherwise} \end{cases}$$

The value of the threshold θ affects the solution of the system. If it is increased in such a way that some of the singular values become negligible then the norm of residual vector $\|b - Ax\|$ would increase and the norm of solution vector $\|x\|$ would decrease.

The minfit function allows a user to solve simultaneous linear systems with *ip* right-hand side vectors. They should be stored in the first *ip* columns of input matrix *b*.

Performance

Appropriate shift of the origin makes the convergence of the algorithm global. The rate of convergence is cubic in almost all the cases.

The computed singular values and singular vectors are exact for a matrix which is a small perturbation of the original matrix. The upper bound for the norm of the perturbation matrix is commensurate to the relative machine precision times the norm of original matrix.

Notes

Matrices *a* and *b* are in the format produced by the allocation function `fmatrix`.

In a case when the original matrix has its number of columns *n* greater than its row dimension *m*, than the matrix must be allocated as square matrix of order *n* by the allocation function `fsquare`. It is required for the correct accumulation of the transformation matrix V, which is stored in place of the input matrix A.

The input parameter `ip` can be zero. In such a case the matrix *b* is not referenced.

Test

The test program is contained in `minfitt.c` file. The example matrices are contained in `minfit1.xpd, minfit2.xpd` files.

References

G. Golub, C. Reinsch, Numerical Mathematics, Vol. 14, pp.403-420, 1970.

J. H. Wilkinson, C. Reinsch, Handbook for Automatic Computation, vol. II – Linear Algebra, Published by: Clarendon Press, Oxford, 1971.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

12.2.4 Application of SVD and MINFIT functions

Rank Estimation

The function ssvd can be applied for determining the numerical or effective rank r of a rectangular matrix. Suppose that matrix A is $m \times n$ matrix and $m > n$. Then the matrix is called full *column* rank if its singular values all satisfy the inequalities:

$$\sigma_i > \theta, \text{where } i \in [0, n-1]$$

If only some of them are greater than the threshold θ then the effective rank r of the matrix can be obtained from the following inequalities:

$$\begin{cases} \sigma_i > \theta & \text{where } i \in [0, r-1] \\ \sigma_i \leq \theta & \text{where } i \in [r, n-1] \end{cases}$$

The value of the threshold θ is usually close to zero and commensurate to the error level within the matrix elements are known. There are two situations appear choosing the threshold.

The first one is when the entries of input matrix are known to be experimental data, so that the matrix can be written as the sum of two matrices:

$$A + \delta A,$$

where A is unknown exact matrix, the matrix δA is a perturbation matrix. The value of threshold θ can be defined as the l_1 norm of the perturbation matrix:

$$\theta \geq \|\delta A\|_1 = \max_{\forall i,j}(\delta a_{ij})$$

The second situation is when the error level of the entries is determined only by the roundoff errors. In such a case the appropriate value for θ would be:

$$\theta \geq \sqrt{mn}\, \varepsilon \max_i \sigma_i,$$

where ε is the relative machine precision.

The Pseudo-Inverse of a Rectangular Matrix

Another application for the ssvd function is determination of the pseudo-inverse of an input $m \times n$ matrix. An $n \times m$ matrix Y is called the pseudo-inverse of A if it has the following properties:

1. $(AY)^T = AY$ and $(YA)^T = YA$
2. $YAY = Y$ and $AYA = A$

3. $\|(AYA - A)\| < \theta$
4. $\|Y\| < \frac{1}{\theta}$

The norms used above are Euclidean matrix norms. Denote the pseudo-inverse matrix as $A^{\dagger}$ and apply the singular value decomposition of A, then the matrix $A^{\dagger}$ can be obtained from the following formula:

$$A^{\dagger} = V\Sigma^{\dagger}U^{T},$$

where the diagonal matrix $\Sigma^{\dagger}$ is defined to have its entries as follows:

$$\sigma_i^{\dagger} = \begin{cases} \frac{1}{\sigma_i} & \text{when } \sigma_i > \theta \\ 0 & \text{otherwise} \end{cases}$$

If the threshold θ is chosen to be smaller than all the non-zero singular values of matrix A then matrix $A^{\dagger}$ is called Moore-Penrose inverse. If matrix A is constrained to be square and non-singular then the pseudo-inverse matrix coincides with the inverse A^{-1} of the original matrix.

Homogeneous Linear Equations

If matrix A is of rank $r < n$, then the non-trivial solution of the homogeneous linear system:

$$Az_i = 0, \text{ where } i \in [r, n-1]$$

can be found as the columns of right-hand singular vector matrix V in the singular value decomposition of the matrix A, that correspond to the singular values of A which are smaller then the threshold θ. It follows from the equation:

$$AV_i = \sigma_i U_i,$$

and when $\sigma_i < \theta$ then the norm $\|AV_i\| < \theta$. A linear combination of such a vectors with the coefficients constrained to have the sum of squares $\sum \alpha_i^2 \leq 1$ would also be a solution of the homogeneous linear system within the chosen threshold .

Both the functions `minfit` and `ssvd` could be applied to find the solution of a homogeneous linear system. Using the `ssvd` function it is not required to accumulate matrix U of left-hand singular vectors and using the `minfit` function one should pass the input parameter *ip* which has been set to zero to the function.

Chapter 13

Performance

13.1 Accuracy Performance Indexes

13.1.1 Real Generalized Eigenproblem

Name

spignr — Estimates accuracy performance index, single precision
dpignr — Estimates accuracy performance index, double precision

Synopsis

```
#include <ceispack.h>
float spignr (n, a, b, alfa, beta, z, s, rv, cv)
int n;
float **a, **b, **z;
fcomplex *alfa;
float *beta;
float *s, *rv;
fcomplex *cv;

double dpignr (n, a, b, alfa, beta, z, s, rv, cv)
int n;
double **a, **b, **z;
dcomplex *alfa;
double *beta;
double *s, *rv;
dcomplex *cv;
```

n	— order of input matrices
a	— first input matrix
b	— second input matrix
alfa	— length n input array containing n numerators of eigenvalues
beta	— length n input array containing n denominators of eigenvalues
z	— square input array containing n transposed (row) eigenvectors
s	— temporary storage/output performance indices array of length n
rv	— temporary storage array of length n
cv	— temporary storage array of length n

Description

Function pignr computes an accuracy performance index to verify that the generalized eigenproblem solution contains accurate/reliable data. The index is computed by the following formula:

$$\mu = \max_{i\in[0,n-1]} \left(\frac{\|\beta_i A z_i - \alpha_i B z_i\|_2}{10\,n\,\varepsilon\ (|\beta_i|\,\|A\|_2 + |\alpha_i|\,\|B\|_2)\ \|z_i\|_2} \right),$$

where ε is the relative machine precision, $\|\cdot\|_2$ denotes Frobenius norm. The performance index is returned as the function value and is to be interpreted as follows:

- $0 < \mu < 1$ implies that the expected accuracy has been achieved within set tolerance.
- $1 < \mu < 100$ suggests that one should be careful about the results.
- $\mu > 100$ data is unreliable and should not be used.

Notes

Arrays *a, b* and *z* are in the format produced by the allocation function `fsquare`.

The eigenvectors should be stored in the matrix *z* in the manner specified by the function `qzvec`.

On return from the function vector *s* contains the array of estimated performance indices for each eigenpair.

References

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

13.1.2 Real General Eigenproblem

Name

spistd — Estimates accuracy performance index, single precision
dpistd — Estimates accuracy performance index, double precision

Synopsis

```
#include <ceispack.h>
float spistd (n, m, a, w, z, s, rv, cv, zv)
int n, m;
float **a, **z;
fcomplex *w;
float *s, *rv;
fcomplex *cv, *zv;

double dpistd (n, m, a, w, z, s, rv, cv, zv)
int n, m;
double **a, **z;
dcomplex *w;
double *s, *rv;
dcomplex *cv, *zv;
```

n — order of input matrix, number of rows in eigenvector matrix
m — number of eigenvalues and eigenvectors, number of columns in eigenvector matrix
a — input general/upper Hessenberg matrix
w — input array containing m eigenvalues
z — input matrix containing m column eigenvectors
s — temporary storage/output performance indices array of length n
rv — temporary storage array of length n
cv — temporary storage array of length n
zv — temporary storage array of length n

Description

Function pistd computes an accuracy performance index to verify thata real general (with a full or upper Hessenberg matrix) eigenproblem solution contains accurate/reliable data. The index is computed by the following formula:

$$\mu = \max_{i \in [0, m-1]} \left(\frac{\|Az_i - \lambda_i z_i\|_2}{10\, n\, \varepsilon\, \|A\|_2\, \|z_i\|_2} \right),$$

where ε is the relative machine precision, $\|\cdot\|_2$ denotes Frobenius norm. The performance index is returned as the function value and is to be interpreted as follows:

- $0 < \mu < 1$ implies that the expected accuracy has been achieved within set tolerance.
- $1 < \mu < 100$ suggests that one should be careful about the results.
- $\mu > 100$ data is unreliable and should not be used.

Notes

Array a is in the format produced by the allocation function `fsquare`. Array z is in the format produced by the allocation function `fmatrix`.

The eigenvectors should be stored in the m columns of the array z in a manner similiar to that of the functions `hqr2` or `invit`.

On return from the function vector s contains the array of estimated performance indices for each eigenpair.

References

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

13.1.3 Symmetric Generalized Eigenproblem

Name

spigsm — Estimates accuracy performance index, single precision
dpigsm — Estimates accuracy performance index, double precision

Synopsis

```
#include <ceispack.h>
float spigsm (n, a, b, w, z, s, rv)
int n;
float **a, **b, *w, **z;
float *s, *rv;

double dpigsm (n, a, b, w, z, s, rv)
int n;
double **a, **b, *w, **z;
double *s, *rv;
```

n — order of input matrices
a — symmetric input matrix
b — symmetric positive definite input matrix
w — length n input array containing n eigenvalues
z — square input array containing n transposed (row) eigenvectors
s — temporary storage/output performance indices array of length n
rv — temporary storage array of length n

Description

Function pigsm computes an accuracy performance index to verify that a symmetric generalized eigenproblem solution contains accurate/reliable data. The eigenproblem is defined as:

$$Ax = \lambda Bx,$$

where matrix A is symmetric, matrix B is symmetric and positive definite. The index is computed by the following formula:

$$\mu = \max_{i\in[0,n-1]} \left(\frac{\|Az_i - \lambda_i B z_i\|_2}{10\, n\, \varepsilon\ (\|A\|_2 + \|B\|_2)\ \|z_i\|_2} \right),$$

where ε is the relative machine precision, $\|\cdot\|_2$ denotes Frobenius norm. The performance index is returned as the function value and is to be interpreted as follows:

- $0 < \mu < 1$ implies that the expected accuracy has been achieved within set tolerance.
- $1 < \mu < 100$ suggests that one should be careful about the results.
- $\mu > 100$ data is unreliable and should not be used.

Notes

Arrays *a, b* and z are expected to be in the format produced by the allocation function `fsquare`.

Having returned from the function vector s contains the array of estimated performance indices for each eigenpair.

References

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

Name

spisab — Estimates accuracy performance index, single precision
dpisab — Estimates accuracy performance index, double precision

Synopsis

```
#include <ceispack.h>
float spisab (n, a, b, w, z, mcase, fm1, s, rv)
int n, mcase;
float **a, **b, *w, **z;
float **fm1, *s, *rv;

double dpisab (n, a, b, w, z, mcase, fm1, s, rv)
int n, mcase;
double **a, **b, *w, **z;
double **fm1, *s, *rv;
```

n	— order of input matrices
a	— symmetric input matrix
b	— symmetric positive definite input matrix
w	— length n input array containing n eigenvalues
z	— input square array containing n transposed (row) eigenvectors
mcase	— input flag which specifies ordering of matrices in the eigensystem equation
fm1	— temporary storage array of order n
s	— temporary storage/output performance indices array of length n
rv	— temporary storage array of length n

Description

Function pisab computes an accuracy performance index to verify that the symmetric generalized eigenproblem solution contains accurate/reliable data. The eigenproblem is defined as either:

$$ABx = \lambda x,$$

or

$$BAx = \lambda x,$$

where matrix A is symmetric, matrix B is symmetric and positive definite. The index is computed by the following formula:

$$\mu = \max_{i \in [0, n-1]} \left(\frac{\|Cz_i - \lambda_i z_i\|_2}{10\, n\, \varepsilon \, \left(\|A\|_2 + \|B\|_2\right) \, \|z_i\|_2} \right),$$

where matrix C is either $C = AB$ or $C = BA$, ε is the relative machine precision. The performance index is returned as the function value and is to be interpreted as follows:

- $0 < \mu < 1$ implies that the expected accuracy has been achieved within set tolerance.
- $1 < \mu < 100$ suggests that one should be careful about the results.
- $\mu > 100$ data is unreliable and should not be used.

Notes

The ordering of the matrices in the eigensystem equation is defined by the input flag *mcase*. There are two possible cases exists:

- *mcase* is set to 0. Then the product AB is used in the equation for the computation of the performance index.
- *mcase* is set to any non-zero value. Then the product BA is used.

Arrays *a, b* and z are in the format produced by the allocation function `fsquare`.

On return from the function vector s contains the array of estimated performance indices for each eigenpair.

References

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

13.1.4 Symmetric Eigenproblem

Name

spisym — Estimates accuracy performance index, single precision
dpisym — Estimates accuracy performance index, double precision

Synopsis

```
#include <ceispack.h>

float spisym (n, m, a, w, z, s, rv)
int n, m;
float **a, *w, **z;
float *s, *rv;

double dpisym (n, m, a, w, z, s, rv)
int n, m;
double **a, *w, **z;
double *s, *rv;
```

n — order of input matrix, number of columns in eigenvector matrix
m — number of eigenvalues and eigenvectors, number of rows in eigenvector matrix
a — symmetric input matrix
w — input array containing m eigenvalues
z — input matrix containing m transposed (row) eigenvectors
s — temporary storage/output performance indices array of length n
rv — temporary storage array of length n

Description

Function pisym computes an accuracy performance index to verify that the symmetric eigenproblem solution contains accurate/reliable data. The index is computed by the following formula:

$$\mu = \max_{i\in[0,m-1]} \left(\frac{\|Az_i - \lambda_i z_i\|_2}{10\, n\, \varepsilon\, \|A\|_2\, \|z_i\|_2} \right),$$

where ε is the relative machine precision, $\|\cdot\|_2$ denotes Frobenius norm. The performance index is returned as the function value and is to be interpreted as follows:

- $0 < \mu < 1$ implies that the expected accuracy has been achieved within set tolerance.
- $1 < \mu < 100$ suggests that one should be careful about the results.
- $\mu > 100$ data is unreliable and should not be used.

Notes

Array a is expected to be in the format produced by the allocation functions `fsym` or `trngl_fmatrix`. If the first function is being used, the *lower* triangle must be specified.

Array z is in the format produced by the allocation function `fmatrix`.

On return from the function vector s contains the array of estimated performance indices for each eigenpair.

References

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

13.1.5 Symmetric Band Eigenproblem

Name

spibds — Estimates accuracy performance index, single precision
dpibds — Estimates accuracy performance index, double precision

Synopsis

```
#include <ceispack.h>
float spibds (n, mb, a, w, m, z, s, rv)
int n, mb, m;
float **a, *w, **z;
float *s, *rv;

double dpibds (n, mb, a, w, m, z, s, rv)
int n, mb, m;
double **a, *w, **z;
double *s, *rv;
```

n — order of input matrix
mb — number of subdiagonals in input matrix
a — band symmetric input matrix
w — input array containing m eigenvalues
m — number of eigenvalues and eigenvectors
z — input matrix containing m transposed (row) eigenvectors
s — temporary storage/output performance indices array of length n
rv — temporary storage array of length n

Description

Function pibds computes an accuracy performance index to verify that the band symmetric eigenproblem solution contains accurate/reliable data. The index is computed by the following formula:

$$\mu = \max_{i\in[0,m-1]} \left(\frac{\|Az_i - \lambda_i z_i\|_2}{10\, n\, \varepsilon\, \|A\|_2\, \|z_i\|_2} \right),$$

where ε is the relative machine precision, $\|\cdot\|_2$ denotes Frobenius norm. The performance index is returned as the function value and is to be interpreted as follows:

- $0 < \mu < 1$ implies that the expected accuracy has been achieved within set tolerance.
- $1 < \mu < 100$ suggests that one should be careful about the results.
- $\mu > 100$ data is unreliable and should not be used.

Notes

Array t is expected to be in the format produced by the allocation functions `trngl_band_fmatrix` or `fsymband`. If the first function is being used, the *lower* part of band must be specified.

Array z is in the format produced by the allocation function `fmatrix`.

On return from the function vector s contains the array of estimated performance indices for each eigenpair.

References

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

13.1.6 Real Special Tridiagonal Eigenproblem

Name

spitrd — Estimates accuracy performance index, single precision
dpitrd — Estimates accuracy performance index, double precision

Synopsis

```
#include <ceispack.h>
float spitrd (n, m, a, w, z, s, rv)
int n, m;
float **a, *w, **z;
float *s, *rv;

double dpitrd (n, m, a, w, z, s, rv)
int n, m;
double **a, *w, **z;
double *s, *rv;
```

n — order of input matrix, number of columns in eigenvector matrix
m — number of eigenvalues and eigenvectors, number of rows in eigenvector matrix
a — tridiagonal input matrix
w — input array containing m eigenvalues
z — input matrix containing m transposed (row) eigenvectors
s — temporary storage/output performance indices array of length n
rv — temporary storage array of length n

Description

Function pitrd computes an accuracy performance index to verify that the special tridiagonal eigenproblem solution contains accurate/reliable data. The index is computed by the following formula:

$$\mu = \max_{i\in[0,m-1]} \left(\frac{\|Az_i - \lambda_i z_i\|_2}{10\, n\, \varepsilon\, \|A\|_2\, \|z_i\|_2} \right),$$

where ε is the relative machine precision, $\|\cdot\|_2$ denotes Frobenius norm. The performance index is returned as the function value and is to be interpreted as follows:

- $0 < \mu < 1$ implies that the expected accuracy has been achieved within set tolerance.
- $1 < \mu < 100$ suggests that one should be careful about the results.
- $\mu > 100$ data is unreliable and should not be used.

Notes

Array a is in the format produced by the allocation functions `band_fmatrix, fband` or `f3diag`.

Array z is in the format produced by the allocation function `fmatrix`.

On return from the function vector s contains the array of estimated performance indices for each eigenpair.

References

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

13.1.7 Symmetric Tridiagonal Eigenproblem

Name

spi3ds — Estimates accuracy performance index, single precision
dpi3ds — Estimates accuracy performance index, double precision

Synopsis

```
#include <ceispack.h>
float spi3ds (n, m, d, e, w, z, s, rv)
int n, m;
float *d, *e, *w, **z;
float *s, *rv;

double dpi3ds (n, m, d, e, w, z, s, rv)
int n, m;
double *d, *e, *w, **z;
double *s, *rv;
```

n — order of symmetric tridiagonal input matrix
m — number of eigenvalues and eigenvectors
d — input vector of length n containing diagonal elements of input matrix
e — input vector of length n containing subdiagonal elements of input matrix
w — input array containing m eigenvalues
z — input matrix containing m transposed (row) eigenvectors
s — temporary storage array of length n
rv — temporary storage array of length n

Description

Function pi3ds computes an accuracy performance index to verify that the symmetric tridiagonal eigenproblem solution contains accurate/reliable data. The index is computed by the following formula:

$$\mu = \max_{i \in [0,m-1]} \left(\frac{\|T z_i - \lambda_i z_i\|_2}{10\, n\, \varepsilon\, \|T\|_2\, \|z_i\|_2} \right),$$

where ε is the relative machine precision, $\|T\|_2$ denotes Frobenius norm of the input matrix. The performance index is returned as the function value and is to be interpreted as follows:

- $0 < \mu < 1$ implies that the expected accuracy has been achieved within set tolerance.
- $1 < \mu < 100$ suggests that one should be careful about the results.
- $\mu > 100$ data is unreliable and should not be used.

The input vectors d and e contain diagonal and subdiagonal elements of the original matrix. The subdiagonal elements should be stored in the *last* $(n-1)$ locations of the vector e. The first entry of the array is arbitrary.

Notes

The rectangular $m \times n$ array z is in the format produced by the allocation function `fmatrix`.

References

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

13.1.8 Complex General Eigenproblem

Name

scpisd — Estimates accuracy performance index, single precision
dzpisd — Estimates accuracy performance index, double precision

Synopsis

```
#include <ceispack.h>
float scpisd (n, m, a, w, z, s, cv)
int n, m;
fcomplex **a, *w, **z;
float *s;
fcomplex *cv;

double dzpisd (n, m, a, w, z, s, cv)
int n, m;
dcomplex **a, *w, **z;
double *s;
dcomplex *cv;
```

n — order of input matrix, number of rows in eigenvector matrix
m — number of eigenvalues and eigenvectors, number of columns in eigenvector matrix
a — input general/upper Hessenberg matrix
w — input array containing m eigenvalues
z — input matrix containing m column eigenvectors
s — temporary storage/output performance indices array of length n
cv — temporary storage array of length n

Description

Function cpisd computes an accuracy performance index to verify that the complex general (with a full or upper Hessenberg matrix) eigenproblem solution contains accurate/reliable data. The index is computed by the following formula:

$$\mu = \max_{i \in [0,m-1]} \left(\frac{\|Az_i - \lambda_i z_i\|_2}{10\, n\, \varepsilon\, \|A\|_2\, \|z_i\|_2} \right),$$

where ε is the relative machine precision, $\|\cdot\|_2$ denotes Frobenius norm. The performance index is returned as the function value and is to be interpreted as follows:

- $0 < \mu < 1$ implies that the expected accuracy has been achieved within set tolerance.
- $1 < \mu < 100$ suggests that one should be careful about the results.
- $\mu > 100$ data is unreliable and should not be used.

Notes

Array a is in the format produced by the allocation function `csquare`. Array z is in the format produced by the allocation function `cmatrix`.

On return from the function vector s contains the array of estimated performance indices for each eigenpair.

References

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

13.1.9 Hermitian Eigenproblem

Name

scpihm — Estimates accuracy performance index, single precision
dzpihm — Estimates accuracy performance index, double precision

Synopsis

```
#include <ceispack.h>
float scpihm (n, m, a, w, zr, zi, s, cv)
int n, m;
fcomplex **a;
float *w, **zr, **zi;
float *s;
fcomplex *cv;

double dzpihm (n, m, a, w, zr, zi, s, cv)
int n, m;
dcomplex **a;
double *w, **zr, **zi;
double *s;
dcomplex *cv;
```

n — order of input matrix
m — number of eigenvalues and eigenvectors
a — Hermitian input matrix
w — input array containing m eigenvalues
zr — input matrix containing real parts of m transposed (row) eigenvectors
zi — input matrix containing imaginary parts of m transposed (row) eigenvectors
s — temporary storage/output performance indices array of length n
cv — temporary storage array of length n

Description

Function cpihm computes an accuracy performance index to verify that the Hermitian eigenproblem solution contains accurate/reliable data. The index is computed by the following formula:

$$\mu = \max_{i\in[0,m-1]} \left(\frac{\|Az_i - \lambda_i z_i\|_2}{10\, n\, \varepsilon\, \|A\|_2\, \|z_i\|_2} \right),$$

where ε is the relative machine precision, $\|\cdot\|_2$ denotes Frobenius norm. The performance index is returned as the function value and is to be interpreted as follows:

- $0 < \mu < 1$ implies that the expected accuracy has been achieved within set tolerance.

- $1 < \mu < 100$ suggests that one should be careful about the results.
- $\mu > 100$ data is unreliable and should not be used.

Notes

Array a is in the format produced by the allocation functions `trngl_cmatrix`, `chermit` or `csquare`. If the first function is being used, the lower array must be specified.

The m-row, n-columns arrays zr, zi are in the format produced by the allocation function `fmatrix`.

On return from the function vector s contains the array of estimated performance indices for each eigenpair.

References

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

13.1.10 Real SVD

Name

spisvd — Estimates accuracy performance index, single precision
dpisvd — Estimates accuracy performance index, double precision

Synopsis

```
#include <ceispack.h>
float spisvd (m, n, a, s, u, v, rv1, rv2)
int m, n;
float **a, **u, **v;
float *s;
float *rv1, *rv2;

double dpisvd (m, n, a, s, u, v, rv1, rv2)
int m, n;
double **a, **u, **v;
double *s;
double *rv1, *rv2;
```

m	— number of rows in matrices a and u
n	— number of columns in matrices a and u, order of matrix v
a	— rectangular input matrix
s	— input array containing $\min(m, n)$ singular values
u	— rectangular input matrix containing n column left singular vectors
v	— square input matrix containing n column right singular vectors
rv1	— temporary storage/output performance indices array of length n
rv2	— temporary storage array of length n

Description

Function pisvd computes an accuracy performance index to verify that the real singular value decomposition:

$$A = U \Sigma V^T$$

of the input matrix A is performed reliable/accurate. The index is computed by the following formula:

$$\mu = \max_{i \in [0, n-1]} \left(\frac{\|Av_i - \sigma_i u_i\|_2 + \left\|A^T u_i - \sigma_i v_i\right\|_2}{10 \max(m, n)\, \varepsilon\, \|A\|_2 \, (\|u_i\|_2 + \|v_i\|_2)} \right),$$

where ε is the relative machine precision, $\|A\|_2$ denotes matrix spectral norm, i. e. the maximum singular value of the matrix, σ_i are the singular values of matrix A.

The performance index is returned as the function value and is to be interpreted as follows:

- $0 < \mu < 1$ implies that the expected accuracy has been achieved within set tolerance.
- $1 < \mu < 100$ suggests that one should be careful about the results.
- $\mu > 100$ data is unreliable and should not be used.

Notes

Arrays *a, u* are in the format produced by the allocation function `fmatrix`. Array v is in the format produced by the allocation function `fsquare`.

For more details on the allocation see the description of function `svd`.

On return from the function vector *rv1* contains the array of estimated performance indices for each singular triplet.

References

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

13.1.11 Complex SVD

Name

scpisv — Estimates accuracy performance index, single precision
dzpisv — Estimates accuracy performance index, double precision

Synopsis

```
#include <ceispack.h>
float scpisv (m, n, a, s, u, v, sv, cv)
int m, n;
fcomplex **a, **u, **v;
fcomplex *s, *cv;
float *sv;

double dzpisv (m, n, a, s, u, v, sv, cv)
int m, n;
dcomplex **a, **u, **v;
dcomplex *s, *cv;
double *sv;
```

m — number of rows in matrices a and u
n — number of columns in matrices a and u, order of matrix v
a — rectangular input matrix
s — input array containing $\min(m, n)$ singular values
u — rectangular input matrix containing n column left singular vectors
v — square input matrix containing n column right singular vectors
sv — temporary storage/output performance indices array of length n
cv — temporary storage array of length n

Description

Function cpisv computes the accuracy performance index to verify that the real singular value decomposition:

$$A = U\Sigma V^*$$

of the input matrix A is performed reliable/accurate. The index is computed by the following formula:

$$\mu = \max_{i\in[0,n-1]} \left(\frac{\|Av_i - \sigma_i u_i\|_2 + \|A^* u_i - \sigma_i v_i\|_2}{10 \max(m,n)\, \varepsilon\, \|A\|_2 \,(\|u_i\|_2 + \|v_i\|_2)} \right),$$

where A^* is the Hermitian conjugate of matrix A, ε is the relative machine precision, $\|A\|_2$ denotes matrix spectral norm, i. e. the maximum singular value of the matrix, σ_i are the singular values of matrix A.

The performance index is returned as the function value and is to be interpreted as follows:

- $0 < \mu < 1$ implies that the expected accuracy has been achieved within set tolerance.
- $1 < \mu < 100$ suggests that one should be careful about the results.
- $\mu > 100$ data is unreliable and should not be used.

Notes

Arrays *a, u* are in the format produced by the allocation function `cmatrix`. Array *v* is in the format produced by the allocation function `csquare`.

For more details on the allocation see the description of function `csvd`.

On return from the function vector *sv* contains the array of estimated performance indices for each singular triplet.

References

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

13.1.12 Linear Least Squares Problem

Name

spilsq — Estimates accuracy performance index, single precision
dpilsq — Estimates accuracy performance index, double precision

Synopsis

```
#include <ceispack.h>
float spilsq (n, a, w, v, rv)
int n;
float **a, *w, **v;
float *rv;

double dpilsq (n, a, w, v, rv)
int n;
double **a, *w, **v;
double *rv;
```

n	— number of columns in input matrix and order of array v
a	— rectangular input matrix
w	— input array containing n singular values
v	— input matrix containing n column right singular vectors
rv	— temporary storage array of length n

Description

Function pistd computes the performance index to verify the accuracy of the function `minfit` which solves homogeneous system of linear equations:

$$Ax = 0$$

The solution of the system are the right singular vectors of matrix A, which are corresponds to the zero singular vectors of matrix A. For more details see the description of the function `minfit`.

The index is computed by the following formula:

$$\mu = \max_{i \in \{j:\, \sigma_j = 0\}} \left(\frac{\|Av_i\|_2}{n\,\varepsilon\,\|A\|_2\,\|v_i\|_2} \right),$$

where ε is the relative machine precision, $\|A\|_2$ denotes matrix spectral norm, i. e. the maximum singular value of the matrix, σ_i are the singular values of matrix A.

The performance index is returned as the function value and is to be interpreted as follows:

- $0 < \mu < 1$ implies that the expected accuracy has been achieved within set tolerance.
- $1 < \mu < 100$ suggests that one should be careful about the results.
- $\mu > 100$ data is unreliable and should not be used.

Notes

Array a is in the format produced by the allocation function `fmatrix`.

Array v is in the format produced by the allocation function `fsquare`.

For more details on the allocation see the description of function `minfit`.

References

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

13.1.13 Band Linear Systems

Name

spibnd — Estimates accuracy performance index, single precision
dpibnd — Estimates accuracy performance index, double precision

Synopsis

```
#include <ceispack.h>
float spibnd (n, mb, a, e21, w, m, x, b, s, rv)
int n, mb, m;
float e21;
float **a, *w, **x, **b;
float *s, *rv;

double dpibnd (n, mb, a, e21, w, m, x, b, s, rv)
int n, mb, m;
double e21;
double **a, *w, **x, **b;
double *s, *rv;
```

n — order of input matrix
mb — number of sub/super diagonals in input matrix
a — input symmetric/non-symmetric band matrix
e21 — input flag which specifies whether input matrix is symmetric or not
w — input array containing m constant values
m — number of right-hand side vectors
x — input matrix containing m transposed (row) solution vectors
b — input matrix containing m transposed (row) right-hand vectors
s — temporary storage/output performance indices array of length n
rv — temporary storage array of length n

Description

Function pibnd computes an accuracy performance index to verify that the band symmetric/non-symmetric linear system solution contains accurate/reliable data. The simultaneous linear system is defined as:

$$(A - \lambda I)X = B$$

Where $\lambda_i,\ i \in [0, m-1]$ are the constant values.

The index is computed by the following formula:

$$\mu = \max_{i\in[0,m-1]} \left(\frac{\|(A - w_i I)x_i - b_i\|_2}{n\,\varepsilon\,\|A - w_i I\|_2\,\|b_i\|_2} \right),$$

where ε is the relative machine precision, $\|\cdot\|_2$ denotes Frobenius norm. The performance index is returned as the function value and is to be interpreted as follows:

- $0 < \mu < 1$ implies that the expected accuracy has been achieved within set tolerance.
- $1 < \mu < 100$ suggests that one should be careful about the results.
- $\mu > 100$ data is unreliable and should not be used.

Notes

Array a is in the format produced either by the allocation functions `trngl_band_fmatrix` or `fsymband` if the input matrix is symmetric. If the first function has been used, the *lower* matrix must be specified. If the original matrix is not symmetric then the `band_fmatrix` or `fband` functions should be used.

The input parameter *e21* specifies whether the input matrix symmetric or not. It should be set to 1 if matrix is symmetric or to -1 if not. Note that non-symmetric matrix should have an equal number of sub/super diagonals.

Arrays x and b are in the format produced by the allocation function `fmatrix`.

The constant values $\lambda_i,\ i \in [0, m-1]$ are stored in the array w.

On return from the function vector s contains the array of estimated performance indices for each solution vector.

References

B. T. Smith, et. all, Matrix Eigensystem Routines – EISPACK Guide, Lecture Notes in Computer Science, vol. 6, Published by: Springer-Verlag, New York, 1976.

B. S. Garbow, et. all, Matrix Eigensystem Routines – EISPACK Guide Extension, Lecture Notes in Computer Science, vol. 51, Published by: Springer-Verlag, New York, 1977.

Chapter 14

Basic Vector Functions

14.1 Real Vector Operations

14.1.1 Fill a Vector with a Constant

Name

sfill — Fill a vector with a scalar, single precision
dfill — Fill a vector with a scalar, double precision

Synopsis

```
#include <ceispack.h>
void sfill (n, sa, sx, incx)
int n;
float sa;
float *sx;
int incx;

void dfill (n, sa, sx, incx)
int n;
double sa;
double *sx;
int incx;
```

n	— number of elements in output vector
sa	— constant value
sx	— output vector
incx	— index increment in output vector

Diagnostics

None.

Description

This function fills elements of the output vector with the constant:

$$x_i = a, \quad \text{where } i \in [0, n-1]$$

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 1 floating point store operation per output vector element.

14.1.2 Copy a Vector to a Vector

Name

scopy — Copy elements of a vector to another vector, single precision
dcopy — Copy elements of a vector to another vector, double precision

Synopsis

```
#include <ceispack.h>
void scopy (n, sx, incx, sy, incy)
int n;
float *sx;
int incx;
float *sy;
int incy;

void dcopy (n, sx, incx, sy, incy)
int n;
double *sx;
int incx;
double *sy;
int incy;
```

n	— number of elements in the input and output vectors
sx	— input vector
incx	— index increment of input vector
sy	— output vector
incy	— index increment of output vector

Diagnostics

None.

Description

This function copies elements of the input vector `sx` to the output vector `sy`:

$$y_i = x_i, \quad \text{where } i \in [0, n-1]$$

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 1 floating point load/store operation per vector element.

14.1.3 Interchange Elements of Two Vectors

Name

sswap — Interchange elements of two vectors, single precision
dswap — Interchange elements of two vectors, double precision

Synopsis

```
#include <ceispack.h>
void sswap (n, sx, incx, sy, incy)
int n;
float *sx;
int incx;
float *sy;
int incy;

void dswap (n, sx, incx, sy, incy)
int n;
double *sx;
int incx;
double *sy;
int incy;
```

n	— number of elements in the input/output vectors
sx	— first input/output vector
incx	— index increment of first vector
sy	— second input/output vector
incy	— index increment of second vector

Diagnostics

None.

Description

This function interchanges the elements of two vectors:

$$x_i \leftrightarrow y_i, \quad \text{where } i \in [0, n-1]$$

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 3 floating point load/store operations per vector element.

14.1.4 Index of the Maximum Absolute Value of a Vector Elements

Name

isamax — Finds the index of maximum absolute value of vector elements, single precision
idamax — Finds the index of maximum absolute value of vector elements, double precision

Synopsis

```
#include <ceispack.h>
int isamax (n, sx, incx)
int n;
float *sx;
int incx;

int idamax (n, sx, incx)
int n;
double *sx;
int incx;
```

n	— number of elements in input vector
sx	— input vector
incx	— index increment of input vector

Diagnostics

If $n < 1$ then no operations are performed and the function returns `IDXE`.

Description

This function returns index l of the element with maximum absolute value. If the vector has several elements with equal maximum absolute value, the function returns index of the first such element.

$$l = \min\left\{i : |x_i| = \max\right\}, \quad \forall\, i \in [0, n-1]$$

Performance

This function takes 1 absolute value and executes 1 floating point comparison per each value tested.

14.1.5 Maximum Absolute Value of a Vector Elements

Name

samax — Find the vector element with the maximum absolute value, single precision
damax — Find the vector element with the maximum absolute value, double precision

Synopsis

```
#include <ceispack.h>
float samax (n, sx, incx)
int n;
float *sx;
int incx;

double damax (n, sx, incx)
int n;
double *sx;
int incx;
```

n	— number of elements in input vector
sx	— input vector
incx	— index increment of input vector

Diagnostics

If $n < 1$ then no operations are performed and the function returns `ZERO`.

Description

This function determines the maximum of absolute values of all the input vector elements and returns this maximum as function value:

$$a = \max(|x_i|), \quad \text{where } i \in [0, n-1]$$

Performance

This function takes 1 absolute value and executes 1 floating point comparison per each value tested.

14.1.6 Sum of the Absolute Values of a Vector Elements

Name

sasum — Sum of absolute values of vector elements, single precision
dasum — Sum of absolute values of vector elements, double precision

Synopsis

```
#include <ceispack.h>
float sasum (n, sx, incx)
int n;
float *sx;
int incx;

double dasum (n, sx, incx)
int n;
double *sx;
int incx;
```

n	— number of elements in input vector
sx	— input vector
incx	— index increment of input vector

Diagnostics

If $n < 1$ then no operations are performed and the function returns `ZERO`.

Description

This function returns sum of absolute values of elements of the input vector:

$$s = \sum_{i=0}^{n-1} |x_i|$$

Performance

This function executes $(n - 1)$ floating point additions and takes n absolute values.

14.1.7 Add a Constant to a Vector

Name

sshift2 — Add a scalar to a vector, single precision
dshift2 — Add a scalar to a vector, double precision

Synopsis

```
#include <ceispack.h>
void sshift2 (n, sa, sx, incx, sy, incy)
int n;
float sa;
float *sx;
int incx;
float *sy;
int incy;

void dshift2 (n, sa, sx, incx, sy, incy)
int n;
double sa;
double *sx;
int incx;
double *sy;
int incy;


#define sshift(n, sa, sx, incx) \
        sshift2(n, sa, sx, incx, sx, incx)


#define dshift(n, sa, sx, incx) \
        dshift2(n, sa, sx, incx, sx, incx)
```

n — number of elements in input/output vector
sa — real constant
sx — input vector
incx — index increment of input vector
sy — output vector
incy — index increment of output vector

Diagnostics

None.

Description

This function adds the constant `sa` to the elements of the input vector `sx` and store result to the output vector `sy`:

$$y_i = a + x_i, \quad \text{where } i \in [0, n-1]$$

Macro `sshift` stores the result in the input vector, i.e. performs the operations in-place.

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 1 floating point addition per output vector element.

14.1.8 Add a Vector to a Vector

Name

svadd2 — Add two vectors, single precision
dvadd2 — Add two vectors, double precision

Synopsis

```
#include <ceispack.h>
void svadd2 (n, sx, incx, sy, incy, sz, incz)
int n;
float *sx;
int incx;
float *sy;
int incy;
float *sz;
int incz;

void dvadd2 (n, sx, incx, sy, incy, sz, incz)
int n;
double *sx;
int incx;
double *sy;
int incy;
double *sz;
int incz;

#define svadd(n, sx, incx, sy, incy) \
        svadd2(n, sx, incx, sy, incy, sy, incy)

#define dvadd(n, sx, incx, sy, incy) \
        dvadd2(n, sx, incx, sy, incy, sy, incy)
```

n	— number of elements in the input/output vectors
sx	— first input vector
incx	— index increment of first vector
sy	— second input vector
incy	— index increment of second vector
sz	— output vector
incz	— index increment of output vector

Diagnostics

None.

Description

This function adds two vectors `sx` and `sy` element by element, and stores the result to the output vector `sz`:

$$z_i = x_i + y_i, \quad \text{where } i \in [0, n-1]$$

Macro `svadd` stores the result in the second input vector, i.e. performs the operations in-place.

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 1 floating point addition per output vector element.

14.1.9 Multiply a Vector by a Constant

Name

sscal2 — Scale a vector by a scalar, single precision
dscal2 — Scale a vector by a scalar, double precision

Synopsis

```
#include <ceispack.h>
void sscal2 (n, sa, sx, incx, sy, incy)
int n;
float sa;
float *sx;
int incx;
float *sy;
int incy;

void dscal2 (n, sa, sx, incx, sy, incy)
int n;
double sa;
double *sx;
int incx;
double *sy;
int incy;


#define sscal(n, sa, sx, incx) \
        sscal2(n, sa, sx, incx, sx, incx)


#define dscal(n, sa, sx, incx) \
        dscal2(n, sa, sx, incx, sx, incx)
```

n	— number of elements in the input/output vectors
sa	— constant, scale factor
sx	— input vector
incx	— index increment of input vector
sy	— output vector
incy	— index increment of output vector

Diagnostics

None.

Description

This function scales the input vector `sx` by the constant `sa` and stores the result in the output vector `sy`:

$$y_i = ax_i, \quad \text{where } i \in [0, n-1]$$

Macro `sscal` stores the result in the input vector, i.e. performs the operations in-place.

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 1 floating point multiplication per output vector element.

14.1.10 Multiply a Vector by a Constant Add a Vector

Name

saxpy2 — Multiplies a vector by a scalar, add a vector, single precision
daxpy2 — Multiplies a vector by a scalar, add a vector, double precision

Synopsis

```
#include <ceispack.h>
void saxpy2 (n, sa, sx, incx, sy, incy, sz, incz)
int n;
float sa;
float *sx;
int incx;
float *sy;
int incy;
float *sz;
int incz;

void daxpy2 (n, sa, sx, incx, sy, incy, sz, incz)
int n;
double sa;
double *sx;
int incx;
double *sy;
int incy;
double *sz;
int incz;

#define saxpy(n, sa, sx, incx, sy, incy) \
        saxpy2(n, sa, sx, incx, sy, incy, sy, incy)

#define daxpy(n, sa, sx, incx, sy, incy) \
        daxpy2(n, sa, sx, incx, sy, incy, sy, incy)
```

n — number of elements in the input/output vectors
sa — constant
sx — first input vector
incx — index increment of first vector
sy — second input vector
incy — index increment of second vector
sz — output vector
incz — index increment of output vector

Diagnostics

None.

Description

This function multiplies the input vector `sx` by the constant `sa` and adds the result to another vector. The resultant vector is stored in the output vector `sz`.

$$z_i = ax_i + y_i, \quad \text{where } i \in [0, n-1]$$

Macro `saxpy` overwrites the second input vector with the output values, i.e. the operation is performed in-place.

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 1 floating point multiplication and 1 floating point addition per output vector element.

14.1.11 Multiply a Vector by a Vector

Name

svmul2 — Multiply two vectors, single precision
dvmul2 — Multiply two vectors, double precision

Synopsis

```
#include <ceispack.h>
void svmul2 (n, sx, incx, sy, incy, sz, incz)
int n;
float *sx;
int incx;
float *sy;
int incy;
float *sz;
int incz;

void dvmul2 (n, sx, incx, sy, incy, sz, incz)
int n;
double *sx;
int incx;
double *sy;
int incy;
double *sz;
int incz;

#define svmul(n, sx, incx, sy, incy) \
        svmul2(n, sx, incx, sy, incy, sy, incy)

#define dvmul(n, sx, incx, sy, incy) \
        dvmul2(n, sx, incx, sy, incy, sy, incy)
```

n	— number of elements in the input/output vectors
sx	— first input vector
incx	— index increment of first vector
sy	— second input vector
incy	— index increment of second vector
sz	— output vector
incz	— index increment of output vector

Diagnostics

None.

Description

This function multiplies two vectors `sx` and `sy` element by element, and stores the result in the output vector `sz`:

$$z_i = x_i y_i, \quad \text{where } i \in [0, n-1]$$

Macro `svmul` stores the result in the second input vector, i.e. performs the operations in-place.

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

The function executes 1 floating point multiplication per output vector element.

14.1.12 Dot Product

Name

sdot — Dot product of two vectors, single precision
ddot — Dot product of two vectors, double precision

Synopsis

```
#include <ceispack.h>
float sdot (n, sx, incx, sy, incy)
int n;
float *sx;
int incx;
float *sy;
int incy;

double ddot (n, sx, incx, sy, incy)
int n;
double *sx;
int incx;
double *sy;
int incy;
```

n	— number of elements in the input vectors
sx	— first input vector
incx	— index increment of first vector
sy	— second input vector
incy	— index increment of second vector

Diagnostics

If $n < 1$ then no operations are performed and the function returns `ZERO`.

Description

This function forms the dot product of two input vectors and returns it as the function value:

$$d = \sum_{i=0}^{n-1} x_i y_i$$

Performance

This function executes n floating point multiplications and $(n-1)$ floating point additions.

14.1.13 Euclidean Norm of a Vector

Name

snrm2, snrm2p — Euclidean norm of a vector, single precision
dnrm2, dnrm2p — Euclidean norm of a vector, double precision

Synopsis

```
#include <ceispack.h>
float snrm2 (n, sx, incx)
int n;
float *sx;
int incx;

float snrm2p (n, sx, incx)
int n;
float *sx;
int incx;

double dnrm2 (n, sx, incx)
int n;
double *sx;
int incx;

double dnrm2p (n, sx, incx)
int n;
double *sx;
int incx;
```

n	— number of elements in input vector
sx	— input vector
incx	— index increment of input vector

Diagnostics

If $n < 1$ then no operations are performed and the functions returns `ZERO`.

Description

This function determines the Euclidean or l_2 norm of the input vector and returns it as the function value:

$$\|x\|_2 = \left(\sum_{i=0}^{n-1} {x_i}^2 \right)^{1/2}$$

Notes

The snrm2p function and its double precision counterpart are generally slower than the snrm2 function because they perform scailing of vector elements to avoid possible overflow/underfow and obtain more precise result.

Performance

The snrm2 function executes n floating point multiplications and $(n - 1)$ additions.

The snrm2p function executes up to $2n$ floating point multiplications in order to obtain more precise result.

14.1.14 Construct Givens Plane Rotation

Name

srotg — Construct Givens plane rotation, single precision
drotg — Construct Givens plane rotation, double precision

Synopsis

```
#include <ceispack.h>
void srotg (sa, sb, c, s)
float *sa, *sb;
float *c, *s;

void drotg (sa, sb, c, s)
double *sa, *sb;
double *c, *s;
```

sa — address of input a/output r variable
sb — address of input b/output z variable
c — address of output variable containing the cosine of rotation
s — address of output variable containing the sine of rotation

Diagnostics

None.

Description

Given the values a and b this function estimates rotation matrix elements c and s. At first, the function computes two parameters:

$$\varrho = \begin{cases} \frac{a}{|a|} & \text{when } |a| > |b| \\ \frac{b}{|b|} & \text{when } |a| \le |b| \end{cases} \qquad \text{and} \qquad r = \varrho\sqrt{a^2 + b^2}$$

Subsequently the matrix elements are computed by the following formuli:

$$c = \begin{cases} 1 & \text{when } r = 0 \\ \frac{a}{r} & \text{otherwise} \end{cases} \qquad \text{and} \qquad s = \begin{cases} 0 & \text{when } r = 0 \\ \frac{b}{r} & \text{otherwise} \end{cases}$$

Therefore computed values c, s and r satisfy the equation:

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}$$

where s and c are respectively the sine and cosine of the rotation angle.

The sign constant $\varrho = \pm 1$ does not affect the rotations themselves, but is necessary when the compact storage scheme using only one memory location is requested for the elements of the rotation angle, allowing sine and cosine construction to be stable. This function additionally computes one more parameter z for the above purpose:

$$z = \begin{cases} s & \text{if } |a| > |b| \\ 1 & \text{if } c = 0 \\ c^{-1} & \text{if } |a| \le |b| \end{cases}$$

The sine and cosine can be constructed now by using the following formuli:

$$c = \begin{cases} \sqrt{1-z^2} & \text{if } |z| < 1 \\ 0 & \text{if } z = 1 \\ z^{-1} & \text{if } |z| > 1 \end{cases} \qquad \text{and} \qquad s = \begin{cases} z & \text{if } |z| < 1 \\ 1 & \text{if } z = 1 \\ \sqrt{1-c^2} & \text{if } |z| > 1 \end{cases}$$

Notes

Values a and b are passed to the function by reference.

On return from the function, the memory locations `*sa`, `*sb`, `*s` and `*c` would contain values r, z, s and c, respectively.

14.1.15 Apply Givens Plane Rotation

Name

srot2 — Apply Givens plane rotation, single precision
drot2 — Apply Givens plane rotation, double precision

Synopsis

```
#include <ceispack.h>
void srot2 (n, sx, incx, sy, incy, sw, sz, c, s)
int n;
float *sx;
int incx;
float *sy;
int incy;
float *sw, *sz,
float c, s;

void drot2 (n, sx, incx, sy, incy, sw, sz, c, s)
int n;
double *sx;
int incx;
double *sy;
int incy;
double *sw, *sz,
double c, s;

#define srot(n, sx, incx, sy, incy, c, s) \
        srot2(n, sx, incx, sy, incy, sx, sy, c, s)

#define drot(n, sx, incx, sy, incy, c, s) \
        drot2(n, sx, incx, sy, incy, sx, sy, c, s)
```

n	— number of elements in the input/output vectors
sx	— first input vector
incx	— index increment of first input/output vector
sy	— second input vector
incy	— index increment of second input/output vector
sw	— first output vector
sz	— second output vector
c	— the cosine of rotation angle
s	— the sine of rotation angle

Diagnostics

None.

Description

This function applies Givens plane rotation to the input vectors `sx` and `sy` and stores the result in the output vectors `sw` and `sz`:

$$\begin{pmatrix} w_i \\ z_i \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}, \quad \text{where } i \in [0, n-1]$$

Macro `srot` stores the result in the input vectors, i.e. performs the operations in-place.

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 2 floating point multiplications and 1 floating point addition per output vector element.

14.1.16 Machine Constants

Name

smach — Compute machine dependent parameters, single precision
dmach — Compute machine dependent parameters, double precision

Synopsis

```
#include <ceispack.h>
float smach (job)
int job;

double dmach (job)
int job;
```

job — specifies the constant to be computed

Diagnostics

None.

Description

The `smach` function computes one of three machine dependent constants and returns it as function value.

Assuming a computer has the following parameters of floating point arithmetic:

r the base of arithmetic
t the number of base r digits
s the smallest possible exponent
l the largest possible exponent

then the function computes either:

$\texttt{eps} = r^{1-t}$
$\texttt{tiny} = 100\, r^{-s+t}$
$\texttt{huge} = 0.01\, b^{l-t}$

The `eps` value is such that $1.0 + \texttt{eps} > 1.0$.

The value to be computed is specifed by the input parameter `job`:

if $\texttt{job} = 1$ the `eps` constant is to be computed
if $\texttt{job} = 2$ the `tiny` constant is to be computed
if $\texttt{job} = 3$ the `huge` constant is to be computed

If `job` has been set to any other value than 2 or 3 the `eps` constant would be computed.

14.2 Complex Vector Operations

14.2.1 Fill a Vector with a Constant

Name

cfill — Fill a complex vector with a complex scalar, single precision
zfill — Fill a complex vector with a complex scalar, double precision

Synopsis

```
#include <ceispack.h>
void cfill (n, ca, cx, incx)
int n;
fcomplex ca;
fcomplex *cx;
int incx;

void zfill (n, ca, cx, incx)
int n;
dcomplex ca;
dcomplex *cx;
int incx;
```

n	— number of elements in output vector
ca	— complex constant
cx	— output complex vector
incx	— index increment in output vector

Diagnostics

None.

Description

This function fills elements of the output vector with the complex constant:

$$x_i = a, \quad \text{where } i \in [0, n-1]$$

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 2 floating point store operations per output vector element.

14.2.2 Copy a Vector to a Vector

Name

ccopy — Copy elements of a complex vector to another complex vector, single precision
zcopy — Copy elements of a complex vector to another complex vector, double precision

Synopsis

```
#include <ceispack.h>
void ccopy (n, cx, incx, cy, incy)
int n;
fcomplex *cx;
int incx;
fcomplex *cy;
int incy;

void zcopy (n, cx, incx, cy, incy)
int n;
dcomplex *cx;
int incx;
dcomplex *cy;
int incy;
```

n	— number of elements in the input and output vectors
cx	— input complex vector
incx	— index increment of input vector
cy	— output complex vector
incy	— index increment of output vector

Diagnostics

None.

Description

This function copies elements of the input vector `cx` to the output vector `cy`:

$$y_i = x_i, \quad \text{where } i \in [0, n-1]$$

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 2 floating point load/store operations per vector element.

14.2.3 Interchange Elements of Two Vectors

Name

cswap — Interchange elements of two complex vectors, single precision
zswap — Interchange elements of two complex vectors, double precision

Synopsis

```
#include <ceispack.h>
void cswap (n, cx, incx, cy, incy)
int n;
fcomplex *cx;
int incx;
fcomplex *cy;
int incy;

void zswap (n, cx, incx, cy, incy)
int n;
dcomplex *cx;
int incx;
dcomplex *cy;
int incy;
```

n	— number of elements in the input/output vectors
cx	— input/output complex vector
incx	— index increment of first vector
cy	— input/output complex vector
incy	— index increment of second vector

Diagnostics

None.

Description

This function interchanges the elements of two complex vectors:

$$x_i \leftrightarrow y_i, \quad \text{where } i \in [0, n-1]$$

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 6 floating point load/store operations per vector element.

14.2.4 Index of the Maximum Absolute Value of a Vector Elements

Name

icamax — Finds the index of maximum absolute value of vector elements, single precision
izamax — Finds the index of maximum absolute value of vector elements, double precision

Synopsis

```
#include <ceispack.h>
int icamax (n, cx, incx)
int n;
fcomplex *cx;
int incx;

int izamax (n, cx, incx)
int n;
dcomplex *cx;
int incx;
```

n	— number of elements in input vector
cx	— input vector
incx	— index increment of input vector

Diagnostics

If $n < 1$ then no operations are performed and the function returns `IDXE`.

Description

This function returns index l of an element of the input vector which has the maximum sum of absolute values of real and imaginary parts. If the vector has several elements with the same maximum, then the function returns the index of the first such element it finds.

$$l = \min\left\{i : |Re(x_i)| + |Im(x_i)| = \max\right\}, \quad \forall\ i \in [0, n-1]$$

Performance

This function executes 1 floating point addition, takes 2 absolute values and executes 1 floating point comparison per each complex value tested.

14.2.5 Maximum Absolute Value of a Vector Elements

Name

scamax — Find the vector element with the maximum sum of absolute values of real and imaginary parts, single precision
dzamax — Find the vector element with the maximum sum of absolute values of real and imaginary parts, double precision

Synopsis

```
#include <ceispack.h>
float scamax (n, cx, incx)
int n;
fcomplex *cx;
int incx;

double dzamax (n, cx, incx)
int n;
dcomplex *cx;
int incx;
```

n	— number of elements in input vector
cx	— input vector
incx	— index increment of input vector

Diagnostics

If $n < 1$ then no operations are performed and the function returns `ZERO`.

Description

This function determines the maximum sum of absolute values of real and imaginary parts of all the input vector elements and returns this maximum as function value:

$$a = \max\left(|Re(x_i)| + |Im(x_i)|\right), \quad \text{where } i \in [0, n-1]$$

Performance

This function takes 2 absolute value, executes 1 floating point addition and 1 floating point comparison per each complex value tested.

14.2.6 Sum of the Absolute Values of a Vector Elements

Name

scasum — Sum of absolute values of real and imaginary parts of complex vector elements, single precision
dzasum — Sum of absolute values of real and imaginary parts of complex vector elements, double precision

Synopsis

```
#include <ceispack.h>
float scasum (n, cx, incx)
int n;
fcomplex *cx;
int incx;

double dzasum (n, cx, incx)
int n;
dcomplex *cx;
int incx;
```

n	—	number of elements in input vector
cx	—	input vector
incx	—	index increment of input vector

Diagnostics

If $n < 1$ then no operations are performed and the function returns `ZERO`.

Description

This function returns the sum of absolute values of real and imaginary parts of elements of the input complex vector:

$$s = \sum_{i=0}^{n-1} (|Re(x_i)| + |Im(x_i)|)$$

Performance

This function executes $(2n - 1)$ floating point additions and takes $2n$ absolute values.

14.2.7 Add a Constant to a Vector

Name

cshift2 — Add a complex scalar to a complex vector, single precision
zshift2 — Add a complex scalar to a complex vector, double precision

Synopsis

```
#include <ceispack.h>
void cshift (n, ca, cx, incx, cy, incy)
int n;
fcomplex ca;
fcomplex *cx;
int incx;
fcomplex *cy;
int incy;

void zshift (n, ca, cx, incx, cy, incy)
int n;
dcomplex ca;
dcomplex *cx;
int incx;
dcomplex *cy;
int incy;


#define cshift(n, ca, cx, incx) \
        cshift2(n, ca, cx, incx, cx, incx)


#define zshift(n, ca, cx, incx) \
        zshift2(n, ca, cx, incx, cx, incx)
```

n	— number of elements in input/output vector
ca	— complex constant
cx	— input complex vector
incx	— index increment of input vector
cy	— output complex vector
incy	— index increment of output vector

Diagnostics

None.

Description

This function adds the complex constant `ca` to the elements of the complex input vector `cx` and stores the result in the output complex vector `cy`:

$$y_i = a + x_i, \quad \text{where } i \in [0, n-1]$$

Macro `cshift` stores the result in the input vector, i.e. performs the operations in-place.

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 2 floating point additions per output vector element.

14.2.8 Add a Vector to a Vector

Name

cvadd2 — Add two complex vectors, single precision
zvadd2 — Add two complex vectors, double precision

Synopsis

```
#include <ceispack.h>
void cvadd2 (n, cx, incx, cy, incy, cz, incz)
int n;
fcomplex *cx;
int incx;
fcomplex *cy;
int incy;
fcomplex *cz;
int incz;

void zvadd2 (n, cx, incx, cy, incy, cz, incz)
int n;
dcomplex *cx;
int incx;
dcomplex *cy;
int incy;
dcomplex *cz;
int incz;

#define cvadd(n, cx, incx, cy, incy) \
        cvadd2(n, cx, incx, cy, incy, cy, incy)

#define zvadd(n, cx, incx, cy, incy) \
        zvadd2(n, cx, incx, cy, incy, cy, incy)
```

n	— number of elements in the input/output vectors
cx	— input complex vector
incx	— index increment of input vector
cy	— input complex vector
incy	— index increment of input vector
cz	— output complex vector
incz	— index increment of output vector

Diagnostics

None.

Description

This function adds two complex vectors `cx` and `cy` element by element, and stores the result in the complex output vector `cz`:

$$z_i = x_i + y_i, \quad \text{where } i \in [0, n-1]$$

Macro `cvadd` stores the result in the second input vector, i.e. performs the operations in-place.

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 2 floating point additions per output vector element.

14.2.9 Multiply a Vector by a Real Constant

Name

csscal — Scale a complex vector by a real scalar, single precision
zdscal — Scale a complex vector by a real scalar, double precision

Synopsis

```
#include <ceispack.h>
void csscal2 (n, sa, cx, incx)
int n;
float sa;
fcomplex *cx;
int incx;
fcomplex *cy;
int incy;

void zdscal2 (n, sa, cx, incx)
int n;
double sa;
dcomplex *cx;
int incx;
dcomplex *cy;
int incy;


#define csscal(n, sa, cx, incx) \
        csscal2(n, sa, cx, incx, cx, incx)


#define zsscal(n, ca, cx, incx) \
        zsscal2(n, ca, cx, incx, cx, incx)
```

n — number of elements in the input/output vectors
sa — real constant, scale factor
cx — input complex vector
incx — index increment of input vector
cy — output complex vector
incy — index increment of output vector

Diagnostics

None.

Description

This function scales the input vector `cx` by a real constant `sa` and stores the result to the output vector `cy`:

$$\begin{cases} Re(y_i) = aRe(x_i) \\ \\ Im(y_i) = aIm(x_i) \end{cases}, \quad \text{where } i \in [0, n-1]$$

Macro `csscal` stores the result in the input vector, i.e. performs the operations in-place.

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 2 floating point multiplications per output vector element.

14.2.10 Multiply a Vector by a Constant

Name

cscal2 — Scale a complex vector by a complex scalar, single precision
zscal2 — Scale a complex vector by a complex scalar, double precision

Synopsis

```
#include <ceispack.h>
void cscal2 (n, ca, cx, incx, cy, incy)
int n;
fcomplex ca;
fcomplex *cx;
int incx;
fcomplex *cy;
int incy;

void zscal2 (n, ca, cx, incx, cy, incy)
int n;
dcomplex ca;
dcomplex *cx;
int incx;
dcomplex *cy;
int incy;


#define cscal(n, ca, cx, incx) \
        cscal2(n, ca, cx, incx, cx, incx)


#define zscal(n, ca, cx, incx) \
        zscal2(n, ca, cx, incx, cx, incx)
```

n	— number of elements in the input/output vectors
ca	— complex constant/scale factor
cx	— input complex vector
incx	— index increment of input vector
cy	— output complex vector
incy	— index increment of output vector

Diagnostics

None.

Description

This function scales a complex input vector `cx` by a complex constant `ca` and stores the result in the output complex vector `cy`:

$$y_i = ax_i, \quad \text{where } i \in [0, n-1]$$

Macro `cscal` stores the result in the input vector, i.e. performs the operations in-place.

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 4 floating point multiplications and 2 floating point additions per output vector element.

14.2.11 Multiply a Vector by a Constant Add a Vector

Name

caxpy2 — Multiply a complex vector by a complex scalar, add a complex vector, single precision
zaxpy2 — Multiply a complex vector by a complex scalar, add a complex vector, double precision

Synopsis

```
#include <ceispack.h>
void caxpy2 (n, ca, cx, incx, cy, incy, cz, incz)
int n;
fcomplex ca;
fcomplex *cx;
int incx;
fcomplex *cy;
int incy;
fcomplex *cz;
int incz;

void zaxpy2 (n, ca, cx, incx, cy, incy, cz, incz)
int n;
dcomplex ca;
dcomplex *cx;
int incx;
dcomplex *cy;
int incy;
dcomplex *cz;
int incz;

#define caxpy(n, ca, cx, incx, cy, incy) \
        caxpy2(n, ca, cx, incx, cy, incy, cy, incy)

#define zaxpy(n, ca, cx, incx, cy, incy) \
        zaxpy2(n, ca, cx, incx, cy, incy, cy, incy)
```

n — number of elements in the input/output vectors
ca — complex constant/scale factor
cx — input complex vector
incx — index increment of input vector
cy — input complex vector
incy — index increment of input vector
cz — output complex vector
incz — index increment of output vector

Diagnostics

None.

Description

This function multiplies the complex input vector `cx` by the complex constant `ca`, and adds the result to another complex vector. The result vector is stored in the output vector `cz`.

$$z_i = ax_i + y_i, \quad \text{where } i \in [0, n-1]$$

Macro `caxpy` overwrites the second input vector with the output values, i.e. the operation is performed in-place.

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 4 floating point multiplications and 4 floating point additions per output vector element.

14.2.12 Multiply a Vector Conjugate by a Constant Add a Vector

Name

caxcpy2 — Multiply a complex conjugated vector by a complex scalar, add a complex vector, single precision
zaxcpy2 — Multiply a complex conjugated vector by a complex scalar, add a complex vector, double precision

Synopsis

```
#include <ceispack.h>
void caxcpy2 (n, ca, cx, incx, cy, incy, cz, incz)
int n;
fcomplex ca;
fcomplex *cx;
int incx;
fcomplex *cy;
int incy;
fcomplex *cz;
int incz;

void zaxcpy2 (n, ca, cx, incx, cy, incy, cz, incz)
int n;
dcomplex ca;
dcomplex *cx;
int incx;
dcomplex *cy;
int incy;
dcomplex *cz;
int incz;
```

```
#define caxcpy(n, ca, cx, incx, cy, incy) \
        caxcpy2(n, ca, cx, incx, cy, incy, cy, incy)
```

```
#define zaxcpy(n, ca, cx, incx, cy, incy) \
        zaxcpy2(n, ca, cx, incx, cy, incy, cy, incy)
```

n	— number of elements in input/output vectors
ca	— complex constant/scale factor
cx	— input complex vector
incx	— index increment of input vector
cy	— input complex vector
incy	— index increment of input vector
cz	— output complex vector
incz	— index increment of output vector

Diagnostics

None.

Description

This function multiplies a conjugate of input vector `cx` by a complex constant `ca`, and adds another vector `cy`. The result vector is stored in the complex output vector `cz`..

$$z_i = a\bar{x}_i + y_i, \quad \text{where } i \in [0, n-1]$$

Macro `caxcpy` puts the output result in the second input vector, i.e. performs the operation in-place.

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

This function executes 4 floating point multiplications and 4 floating point additions per output vector element.

14.2.13 Multiply a Vector by a Vector

Name

cvmul2 — Multiply two complex vectors, single precision
zvmul2 — Multiply two complex vectors, double precision

Synopsis

```
#include <ceispack.h>
void cvmul2 (n, cx, incx, cy, incy, cz, incz)
int n;
fcomplex *cx;
int incx;
fcomplex *cy;
int incy;
fcomplex *cz;
int incz;

void zvmul2 (n, cx, incx, cy, incy, cz, incz)
int n;
dcomplex *cx;
int incx;
dcomplex *cy;
int incy;
dcomplex *cz;
int incz;

#define cvmul(n, cx, incx, cy, incy) \
        cvmul2(n, cx, incx, cy, incy, cy, incy)

#define zvmul(n, cx, incx, cy, incy) \
        zvmul2(n, cx, incx, cy, incy, cy, incy)
```

n	— number of elements in the input/output vectors
cx	— input complex vector
incx	— index increment of input vector
cy	— input complex vector
incy	— index increment of input vector
cz	— output complex vector
incz	— index increment of output vector

Diagnostics

None.

Description

This function multiplies two complex vectors `cx` and `cy` element by element, and stores the result in the complex output vector `cz`:

$$z_i = x_i y_i, \quad \text{where } i \in [0, n-1]$$

Macro `cvmul` stores the result in the second input vector, i.e. performs the operations in-place.

Notes

If $n < 1$ then the function returns immediately with no operations performed.

Performance

The function executes 4 floating point multiplications and 2 floating point additions per output vector element.

14.2.14 Dot Product

Name

cdotu — Dot product of two complex vectors, single precision
zdotu — Dot product of two complex vectors, double precision

Synopsis

```
#include <ceispack.h>
fcomplex cdotu (n, cx, incx, cy, incy)
int n;
fcomplex *cx;
int incx;
fcomplex *cy;
int incy;

dcomplex zdotu (n, cx, incx, cy, incy)
int n;
dcomplex *cx;
int incx;
dcomplex *cy;
int incy;
```

n	— number of elements in the input vectors
cx	— input complex vector
incx	— index increment of input vector
cy	— input complex vector
incy	— index increment of input vector

Diagnostics

If $n < 1$ then no operations are performed and the function returns complex `ZERO`.

Description

This function forms the complex dot product of two complex vectors and returns it as the function value:

$$d = \sum_{i=0}^{n-1} x_i y_i$$

Performance

This function executes $4n$ floating point multiplications and $(2n - 2)$ floating point additions.

14.2.15 Dot Product with Conjugation

Name

cdotc — Dot product of two complex vectors, conjugating first vector, single precision
zdotc — Dot product of two complex vectors, conjugating first vector, double precision

Synopsis

```
#include <ceispack.h>
fcomplex cdotc (n, cx, incx, cy, incy)
int n;
fcomplex *cx;
int incx;
fcomplex *cy;
int incy;

dcomplex zdotc (n, cx, incx, cy, incy)
int n;
dcomplex *cx;
int incx;
dcomplex *cy;
int incy;
```

n	— number of elements in the input vectors
cx	— input complex vector
incx	— index increment of input vector
cy	— input complex vector
incy	— index increment of input vector

Diagnostics

If $n < 1$ then no operations are performed and the function returns complex `ZERO`.

Description

This function forms the complex dot product of two complex input vectors, conjugating the first vector. The function returns the dot product as the function value.

$$d = \sum_{i=0}^{n-1} \bar{x}_i y_i$$

Performance

This function executes $4n$ floating point multiplications and $(2n-2)$ floating point additions.

14.2.16 Unitary Norm of a Vector

Name

scnrm2, scnrm2p — Unitary norm of a complex vector, single precision
dznrm2, dznrm2p — Unitary norm of a complex vector, double precision

Synopsis

```
#include <ceispack.h>
float scnrm2 (n, cx, incx)
int n;
fcomplex *cx;
int incx;

float scnrm2p (n, cx, incx)
int n;
fcomplex *cx;
int incx;

double dznrm2 (n, cx, incx)
int n;
dcomplex *cx;
int incx;

double dznrm2p (n, cx, incx)
int n;
dcomplex *cx;
int incx;
```

n	— number of elements in input vector
cx	— input vector
incx	— index increment of input vector

Diagnostics

If $n < 1$ then no operations are performed and the functions returns `ZERO`.

Description

This function determines the unitary norm of the input vector and returns it as the function value:

$$\|x\|_2 = \left(\sum_{i=0}^{n-1} \left(Re(x_i)^2 + Im(x_i)^2 \right) \right)^{1/2}$$

Notes

The scnrm2p function and its double precision counterpart are generally slower than the scnrm2 function because they perform scaling of the vector elements to avoid possible overflow/underflow and obtain a more precise result.

Performance

The scnrm2 function executes $2n$ floating point multiplications and $(2n - 1)$ additions.

The scnrm2p function executes up to $4n$ floating point multiplications in order to obtain a more precise result.

14.2.17 Construct Givens Plane Rotation

Name

crotg — Construct Givens plane rotation, single precision
zrotg — Construct Givens plane rotation, double precision

Synopsis

```
#include <ceispack.h>
void crotg (ca, cb, c, s)
fcomplex *ca, *cb;
float *c;
fcomplex *s;

void zrotg (ca, cb, c, s)
dcomplex *ca, *cb;
double *c;
dcomplex *s;
```

ca — address of first input a/output αr variable
cb — address of second input variable b
c — address of output variable containing first parameter of rotation
s — address of output variable containing second parameter of rotation

Description

Given the values a and b this function estimates rotation matrix elements c and s. At first, the function verifies if $|a|$ is not zero and computes two parameters:

$$\alpha = \frac{a}{|a|} \qquad \text{and} \qquad r = \sqrt{|a|^2 + |b|^2}$$

Subsequently the matrix elements are computed by the following expressions:

$$c = \begin{cases} 0 & \text{when } |a| = 0 \\ \frac{|a|}{r} & \text{otherwise} \end{cases} \qquad \text{and} \qquad s = \begin{cases} 1 & \text{when } |a| = 0 \\ \frac{\alpha \bar{b}}{r} & \text{otherwise} \end{cases}$$

Therefore computed values c, s and r satisfy the equation:

$$\begin{pmatrix} c & s \\ -\bar{s} & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \alpha r \\ 0 \end{pmatrix}$$

Notes

Values a and b are passed to the function by reference.

On return from the function, the memory locations `*s` and `*c` contain the values s and c respectively.

Variable `*sa` has been set to αr if $a \neq 0$, otherwise it is set to the value of input variable `*sb`.

14.2.18 Apply Givens Plane Rotation

Name

csrot2 — Apply Givens plane rotation, single precision
zdrot2 — Apply Givens plane rotation, double precision

Synopsis

```
#include <ceispack.h>
void csrot2 (n, cx, incx, cy, incy, cw, cz, c, s)
int n;
fcomplex *cx;
int incx;
fcomplex *cy;
int incy;
fcomplex *cw, *cz;
float c, s;

void csrot2 (n, cx, incx, cy, incy, cw, cz, c, s)
int n;
dcomplex *cx;
int incx;
dcomplex *cy;
int incy;
dcomplex *cw, *cz;
double c, s;

#define csrot(n, cx, incx, cy, incy, c, s) \
        csrot2(n, cx, incx, cy, incy, cx, cy, c, s)

#define zdrot(n, cx, incx, cy, incy, c, s) \
        zdrot2(n, cx, incx, cy, incy, cx, cy, c, s)
```

n	— number of elements in the input/output vectors
cx	— first input complex vector
incx	— index increment of first input/output vector
cy	— second input complex vector
incy	— index increment of second input/output vector
cw	— first output complex vector
cz	— second output complex vector
c	— the cosine of rotation angle
s	— the sine of rotation angle

Diagnostics

None.

Description

This function applies Givens plane rotations to the complex input vectors `cx` and `cy` and stores the result in the complex output vectors `cw` and `cz`:

$$\left\{ \begin{array}{l} Re \begin{pmatrix} w_i \\ z_i \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} Re \begin{pmatrix} x_i \\ y_i \end{pmatrix} \\ \\ Im \begin{pmatrix} w_i \\ z_i \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} Im \begin{pmatrix} x_i \\ y_i \end{pmatrix} \end{array} \right. , \quad \text{where } i \in [0, n-1]$$

Macro `csrot` stores the result in the input vectors, i.e. performs the operations in-place.

Notes

If $n < 1$ then the function returns immediately with no operation performed.

Performance

This function executes 4 floating point multiplications and 2 floating point additions per output vector element.

14.2.19 Machine Constants

Name

cmach — Compute machine dependent parameters, single precision
zmach — Compute machine dependent parameters, double precision

Synopsis

```
#include <ceispack.h>
float cmach (job)
int job;

double zmach (job)
int job;
```

job — specifies the constant to be computed

Diagnostics

None.

Description

The `cmach` function computes one of three machine dependent constants and returns it as function value.

Assuming a computer has the following parameters of floating point arithmetic:

r	the base of arithmetic
t	the number of base r digits
s	the smallest possible exponent
l	the largest possible exponent

then the function computes either:

$\texttt{eps} = r^{1-t}$
$\texttt{tiny} = 100\, r^{-s+t}$
$\texttt{huge} = 0.01\, b^{l-t}$

The `eps` value is such that $1.0 + \texttt{eps} > 1.0$.

The value to be computed is specifed by the input parameter `job`:

if	$\texttt{job} = 1$	the `eps` constant is to be computed
if	$\texttt{job} = 2$	the `tiny` constant is to be computed
if	$\texttt{job} = 3$	the `huge` constant is to be computed

If `job` has been set to any other value than 2 or 3 the `eps` constant would be computed.

Chapter 15

Matrix Allocation

15.1 Real Matrices and Vectors

15.1.1 General Matrices and Vectors

Name

fmatrix — Allocate real general matrix, single precision
dmatrix — Allocate real general matrix, double precision

Synopsis

```
#include <cblas.h>

float **fmatrix (m, n)
int m;
int n;

double **dmatrix (m, n)
int m;
int n;

#define   fsquare(n)   fmatrix (n, n)
#define   dsquare(n)   dmatrix (n, n)
#define   fvector(n)   (float *) fmatrix (1, n)
#define   dvector(n)   (double *) dmatrix (1, n)
```

m — number of rows in matrix
n — number of columns in matrix/order of matrix/length of vector

Diagnostics

The `fmatrix` function returns a pointer to the allocated matrix if it is successful, or `NULL` if the allocation failed.

Description

The `fmatrix` function allocates storage for a real rectangular m-rows, n-columns matrix and returns a pointer to the allocated matrix.

Matrix elements are stored in memory contiguously, row-wise.

Macro `fsquare` allocates storage for a square matrix of order n, and returns a pointer to the allocated matrix.

Macro `fvector` allocates storage for a real vector of length n, and returns a pointer to the allocated vector.

15.1.2 Symmetric and Triangular Matrices

Name

trngl_fmatrix — Allocate real triangular matrix, single precision
trngl_dmatrix — Allocate real triangular matrix, double precision

Synopsis

```
#include <cblas.h>

float **trngl_fmatrix (uplo, diag, n)
char uplo;
char diag;
int n;

double **trngl_dmatrix (uplo, diag, n)
char uplo;
char diag;
int n;

#define   fsym(n)   trngl_fmatrix ('l', 'n', n)
#define   dsym(n)   trngl_dmatrix ('l', 'n', n)
```

uplo — specifies lower/upper triangular matrix
diag — specifies allocation of principal diagonal
n — order of the matrix

Diagnostics

The `trngl_fmatrix` function returns a pointer to the allocated matrix if it is successful, or `NULL` if the allocation failed.

Description

The `trngl_fmatrix` function allocates storage for the lower or upper part of a real symmetric (triangular) matrix of order n, and returns the pointer to the allocated matrix.

Matrix elements are stored in memory contiguosly row-wise.

Only the entries in the upper or lower triangular part of the matrix are allocated and can be referenced only for:

Type of triangle	matrix entries a_{ij}
lower	$i \geq j$
strict lower	$i > j$
upper	$i \leq j$
strict upper	$i < j$

Other locations in the matrix do not exist and must not be referenced.

Macro `fsym` allocates storage for the lower part of a symmetric or triangular non-unit matrix of order n, and returns a pointer to the allocated matrix.

Notes

If the input flag `uplo` is set to `'U'` or `'u'` then the upper triangle part is to be allocated, otherwise the lower part would be allocated.

If the input flag `diag` is set to `'U'` or `'u'` then the principal diagonal of the matrix is *not* allocated.

15.1.3 Band Matrix

Name

band_fmatrix — Allocate real band matrix, single precision
band_dmatrix — Allocate real band matrix, double precision

Synopsis

```
#include <cblas.h>

float **band_fmatrix (m, n, nl, nu)
int m, n;
int nl, nu;

double **band_dmatrix (m, n, nl, nu)
int m, n;
int nl, nu;

#define   fband(n, nl, nu)   band_fmatrix (n, n, nl, nu)
#define   dband(n, nl, nu)   band_dmatrix (n, n, nl, nu)
#define   f3diag(n)   band_fmatrix (n, n, 1, 1)
#define   d3diag(n)   band_dmatrix (n, n, 1, 1)
```

m	— number of rows in the matrix
n	— number of columns/order of the matrix
nl	— number of subdiagonals in the matrix
nu	— number of superdiagonals in the matrix

Diagnostics

The `band_fmatrix` function returns a pointer to the allocated matrix if it is successful, or `NULL` if the allocation failed.

Description

The `band_fmatrix` function allocates storage for a real m-rows, n-columns band matrix and returns a pointer to the allocated matrix.

Macro `fband` allocates storage for a band matrix of order n.

The elements of the allocated matrix are stored in memory contiguously row-wise. Only band entries are allocated and can be referenced only for:

$$a_{ij} : |i - j| \leq n_u, \; i < j$$

or

$$a_{ij} : |i-j| \leq n_l,\ i \geq j$$

Other locations in the matrix do not exist and must not be referenced.

Macro `f3diag` allocates storage for a real tridiagonal matrix of order n, and returns a pointer to the allocated matrix. Matrix elements are stored in memory contiguously row-wise. Only the entries in upper, principal and lower diagonals are allocated and can be referenced only for:

$$a_{ij} :\ |i-j| \leq 1$$

As in above routines, other locations in the matrix do not exist and must not be referenced.

15.1.4 Band Symmetric and Triangular Matrices

Name

trngl_band_fmatrix — Allocate real band symmetric matrix, single precision
trngl_band_dmatrix — Allocate real band symmetric matrix, double precision

Synopsis

```
#include <cblas.h>

float **trngl_band_fmatrix (uplo, diag, n, k)
char uplo, diag;
int n;
int k;

double **trngl_band_dmatrix (uplo, diag, n, k)
char uplo, diag;
int n;
int k;

#define   fsymband(n, k)   trngl_band_fmatrix ('l', 'n', n, k)
#define   dsymband(n, k)   trngl_band_dmatrix ('l', 'n', n, k)
#define   fsym3diag(n)   trngl_band_fmatrix ('l', 'n', n, 1)
#define   dsym3diag(n)   trngl_band_dmatrix ('l', 'n', n, 1)
```

uplo	— specifies lower/upper part of the band
diag	— specifies allocation of principal diagonal
n	— order of matrix
k	— number of sub-/super-diagonals

Diagnostics

The `trngl_band_fmatrix` function returns a pointer to the allocated matrix if it is successful, or `NULL` if the allocation failed.

Description

The `trngl_band_fmatrix` function allocates storage for a real band symmetric matrix of order n, and returns a pointer to the allocated matrix.

Macro `fsymband` allocates storage for the lower part of a band symmetric matrix of order n.

Matrix elements are stored in memory contiguously row-wise.

Only the entries in upper or lower band part of the matrix are allocated and can be referenced only for:

in a case of the upper band part of the matrix:

$$a_{ij} : \quad |i-j| \leq k, \ i \leq j$$

in a case of the lower band part of the matrix:

$$a_{ij} : \quad |i-j| \leq k, \ i \geq j$$

Other locations in the matrix do not exist and must not be referenced.

If the principal diagonal is not allocated, then in the case the the upper part of the matrix the index i must satisfy $i < j$. In the case for the lower part of the matrix the index i must satisfy $i > j$.
Macro `fsym3diag` allocates storage for the lower part of a real symmetric tridiagonal matrix of order n, and returns a pointer to the allocated matrix.

Only the entries in principal diagonal a_{ii}, and in the first subdiagonal $a_{i,i-1}$ are allocated and can be referenced.

Notes

If the input flag `uplo` is set to `'U'` or `'u'` then the upper band part of the matrix is to be allocated, otherwise the lower band part would be allocated.

If the input flag `diag` is set to `'U'` or `'u'` then the principal diagonal of the matrix is *not* allocated.

15.2 Complex Matrices and Vectors

15.2.1 General Matrices and Vectors

Name

cmatrix — Allocate complex general matrix, single precision
zmatrix — Allocate complex general matrix, double precision

Synopsis

```
#include <ceispack.h>
fcomplex **cmatrix (m, n)
int m;
int n;

dcomplex **zmatrix (m, n)
int m;
int n;

#define   csquare(n)   cmatrix (n, n)
#define   zsquare(n)   zmatrix (n, n)
#define   cvector(n)   (fcomplex *) cmatrix (1, n)
#define   zvector(n)   (dcomplex *) zmatrix (1, n)
```

m	— number of rows in matrix
n	— number of columns in matrix/order of matrix/length of vector

Diagnostics

The `cmatrix` function returns a pointer to the allocated matrix if it is successful, or `NULL` if the allocation failed.

Description

The `cmatrix` function allocates storage for a complex rectangular m-rows, n-columns matrix and returns a pointer to the allocated matrix.

Matrix elements are stored in memory contiguously, row-wise, with their real and imaginary part interleaved.

Macro `csquare` allocates storage for a square matrix of order n, and returns the pointer to the allocated matrix.

Macro `cvector` allocates storage for a complex vector of length n, and returns the pointer to the allocated vector.

15.2.2 Hermitian and Triangular Matrices

Name

trngl_cmatrix — Allocate complex triangular matrix, single precision
trngl_zmatrix — Allocate complex triangular matrix, double precision

Synopsis

```
#include <ceispack.h>
fcomplex **trngl_cmatrix (uplo, diag, n)
char uplo;
char diag;
int n;

dcomplex **trngl_zmatrix (uplo, diag, n)
char uplo;
char diag;
int n;

#define   chermit(n)   trngl_cmatrix ('l', 'n', n)
#define   zhermit(n)   trngl_zmatrix ('l', 'n', n)
```

uplo	— specifies lower/upper triangular matrix
diag	— specifies allocation of principal diagonal
n	— order of the matrix

Diagnostics

The `trngl_cmatrix` function returns a pointer to the allocated matrix if it is successful, or `NULL` if the allocation failed.

Description

The `trngl_cmatrix` function allocates storage for lower or upper part of a complex triangular (or symmetric, or Hermitian) matrix of order n, and returns a pointer to the allocated matrix.

Matrix elements are stored in memory row-wise, with their real and imaginary part interleaved.

Only the entries in the upper or lower triangular part of the matrix are allocated and can be referenced only for:

Type of triangle	matrix entries a_{ij}
lower	$i \geq j$
strict lower	$i > j$
upper	$i \leq j$
strict upper	$i < j$

Other locations in the matrix do not exist and must not be referenced.

Macro `chermit` allocates storage the for lower part of a Hermitian (symmetric or triangular) non-unit matrix of order n, and returns a pointer to the allocated matrix.

Notes

If the input flag `uplo` is set to `'U'` or `'u'` then the upper triangle part is to be allocated, otherwise the lower part would be allocated.

If the input flag `diag` is set to `'U'` or `'u'` then the principal diagonal of the matrix is *not* allocated.

15.2.3 Band Matrix

Name

band_cmatrix — Allocate complex band matrix, single precision
band_zmatrix — Allocate complex band matrix, double precision

Synopsis

```
#include <ceispack.h>
fcomplex **band_cmatrix (m, n, nl, nu)
int m, n;
int nl, nu;

dcomplex **band_zmatrix (m, n, nl, nu)
int m, n;
int nl, nu;
```

m	— number of rows in the matrix
n	— number of columns in the matrix
nl	— number of subdiagonals in the matrix
nu	— number of superdiagonals in the matrix

Diagnostics

The `band_cmatrix` function returns a pointer to the allocated matrix if it is successful, or `NULL` if the allocation failed.

Description

The function `band_cmatrix` allocates storage for a complex m-rows, n-columns band matrix and returns a pointer to the allocated matrix.

Matrix elements are stored in memory contiguously, row-wise, with their real and imaginary parts interleaved.

Only band entries are allocated and can be referenced only for:

$$a_{ij} : |i - j| \leq n_u,\ i < j$$

or

$$a_{ij} : |i - j| \leq n_l,\ i \geq j$$

Other locations in the matrix do not exist and must not be referenced.

15.2.4 Band Hermitian and Triangular Matrix

Name

trngl_band_cmatrix — Allocate band Hermitian matrix, single precision
trngl_band_zmatrix — Allocate band Hermitian matrix, double precision

Synopsis

```
#include <ceispack.h>
fcomplex **trngl_band_cmatrix (uplo, diag, n, k)
char uplo, diag;
int n;
int k;

dcomplex **trngl_band_zmatrix (uplo, diag, n, k)
char uplo, diag;
int n;
int k;
```

uplo	— specifies lower/upper part of the band
diag	— specifies allocation of principal diagonal
n	— order of matrix
k	— number of sub-/super-diagonals

Diagnostics

The `trngl_band_cmatrix` function returns a pointer to the allocated matrix if it is successful, or `NULL` if the allocation failed.

Description

The `trngl_band_cmatrix` function allocates storage for a band Hermitian matrix of order n, and returns a pointer to the allocated matrix.

Matrix elements are stored in memory contiguously, row-wise, with their real and imaginary part interleaved.

Only the entries in the upper or lower band part of the matrix are allocated and can be referenced only for:

in a case of the upper band part of the matrix:

$$a_{ij} : \quad |i-j| \leq k, \; i \leq j$$

in a case of the lower band part of the matrix:

$$a_{ij}: \quad |i-j| \leq k,\ i \geq j$$

Other locations in the matrix do not exist and must not be referenced.

If the principal diagonal is not allocated, then in the case of the upper part of the matrix the index i must satisfy $i < j$. In the case of the lower part of the matrix the index i must satisfy $i > j$.

Notes

If the input flag `uplo` is set to `'U'` or `'u'` then the upper band part of the matrix is to be allocated, otherwise the lower band part would be allocated.

If the input flag `diag` is set to `'U'` or `'u'` then the principal diagonal of the matrix is *not* allocated.

Index