trf (getrf)

compute an LU factorization of a general $M \times N$ matrix A using partial pivoting with row interchanges

Synopsis

```
template <typename FS>
    int
    trf(GeMatrix<FS> &A, DenseVector<Array<int> > &P);
```

Purpose

computes an LU factorization of a general $M \times N$ matrix A using partial pivoting with row interchanges. The factorization has the form $A = PLU$ where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if $m > n$), and U is upper triangular (upper trapezoidal if $m < n$).

Arguments

A	(input/output) On entry, the $M \times N$ matrix to be factored. On exit, the factors L and U from the factorization $A = PLU$; the unit diagonal elements of L are not stored.
P	(output) The pivot indices; for $1 \leq i \leq \min\{M, N\}$, row i of the matrix was interchanged with row $P(i)$.

Returns

$i = 0$	successful exit
$i > 0$	then $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

tri (getri)

compute the inverse of a matrix using the LU factorization computed by trf

Synopsis

```
template <typename FS>
    int
    tri(GeMatrix<FS> &A, DenseVector<Array<int> > &P);
```

Purpose

tri computes the inverse of a matrix using the LU factorization computed by trf. This function inverts U and then computes A^{-1} by solving the system $A^{-1} * L = U^{-1}$ for A^{-1}.

Arguments

`A`	(input/output) On entry, the factors L and U from the factorization $A = PLU$ as computed by trf. On successful exit, the inverse of the original matrix A.
`P`	(input) The pivot indices from trf; for $1 \leq i \leq N$, row i of the matrix was interchanged with row $P(i)$.

Returns

$i = 0$	successful exit
$i > 0$	then $U(i,i)$ is exactly zero; the matrix is singular and its inverse could not be computed.

trf (gbtrf)

compute an LU factorization of a real $M \times N$ band matrix A using partial pivoting with row interchanges

Synopsis

```
template <typename BS>
    int
    trf(GbMatrix<BS> &A, DenseVector<Array<int> > &P);
```

Purpose

trf computes an LU factorization of a real $M \times N$ band matrix A using partial pivoting with row interchanges. This is the blocked version of the algorithm, calling Level 3 BLAS.

Arguments

`A`	(input/output) On entry, the matrix A in band storage and on exit overwritten with the LU factorization. Matrix `A` is required to have a total of k_l subdiagonals and $k_l + k_u$ superdiagonals where on entry only the elements within the k_l subdiagonals and k_u superdiagonals need to be set (See Further Details).
`P`	(input) The pivot indices; for $1 \leq i \leq N$, row i of the matrix was interchanged with row $P(i)$.

Returns

$i = 0$	successful exit
$i > 0$	then $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Further Details

Assume non-zero elements of A reside within a band of k_l subdiagonals and k_u superdiagonals. Then storing its LU factorization requires a band of k_l subdiagonals and $k_l + k_u$ superdiagonals. Hence, the `GbMatrix` holding the matrix A needs to have k_l subdiagonals and $k_l + k_u$ superdiagonals allocated; but only the elements within its k_l subdiagonals and k_u superdiagonals must be set.

If, for example, A is a tridiagonal matrix (i. e. $k_l = k_u = 1$), then matrix U will have in general two super-diagonals. This requires the `GbMatrix` storing A has allocated an additional superdiagonal (as indicated by '$*$'):

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & * & 0 & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & * & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & * \\ 0 & 0 & a_{4,3} & a_{4,4} & a_{4,5} \\ 0 & 0 & 0 & a_{5,4} & a_{5,5} \end{pmatrix} \quad \rightsquigarrow \quad A_{LU} = \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & 0 & 0 \\ m_{2,1} & u_{2,2} & u_{2,3} & u_{2,4} & 0 \\ 0 & m_{3,2} & u_{3,3} & u_{3,4} & u_{3,5} \\ 0 & 0 & m_{4,3} & u_{4,4} & u_{4,5} \\ 0 & 0 & 0 & m_{5,4} & u_{5,5} \end{pmatrix}.$$

Elements of L are stored (in general rearranged due to pivoting) on the subdiagonal of A_{LU}.

trs (getrs)

solve a system of linear equations $AX = B$ or $A^T X = B$ with a general $N \times N$ matrix A using the LU factorization computed by trf

Synopsis

```
template <typename MA, typename MB>
    int
    trs(Transpose trans, const GeMatrix<MA> &A,
        const DenseVector<Array<int> > &P, GeMatrix<MB> &B);

template <typename MA, typename VB>
    int
    trs(Transpose trans, const GeMatrix<MA> &A,
        const DenseVector<Array<int> > &P,
        DenseVector<VB> &B);
```

Purpose

trs solves a system of linear equations $AX = B$ or $A^T X = B$ with a general $N \times N$ matrix A using the LU factorization computed by trf.

Arguments

`trans`	(input) Specifies the form of the system of equations: `trans = NoTrans` $AX = B$ (No transpose) `trans = Trans` $A^T X = B$ (Transpose) `trans = ConjTrans` $A^H X = B$ (Conjugate transpose = Transpose)
`A`	(input) The factors L and U from the factorization $A = PLU$ as computed by trf.
`P`	(input) The pivot indices from trf; for $1 \leq i \leq N$, row i of the matrix was interchanged with row $P(i)$.
`B`	(input/output) On entry, the right hand side matrix B. On exit, the solution matrix X.

Returns

$i = 0$	successful exit

trs (gbtrs)

solve a system of linear equations $AX = B$ or $A^T X = B$ with a general band matrix A using the LU factorization computed by trf

Synopsis

```
template <typename MA, typename MB>
    int
    trs(Transpose trans, const GbMatrix<MA> &LU,
        const DenseVector<Array<int> > &P, GeMatrix<MB> &B);

template <typename MA, typename VB>
    int
    trs(Transpose trans, const GbMatrix<MA> &LU,
        const DenseVector<Array<int> > &P,
        DenseVector<VB> &B);
```

Purpose

trs solves a system of linear equations $AX = B$ or $A^T X = B$ with a general band matrix A using the LU factorization computed by trf.

Arguments

`trans`	(input) Specifies the form of the system of equations: `trans = NoTrans` $AX = B$ (No transpose) `trans = Trans` $A^T X = B$ (Transpose) `trans = ConjTrans` $A^H X = B$ (Conjugate transpose = Transpose)
`A`	(input) Details of the LU factorization of the band matrix A, as computed by trf. U is stored as an upper triangular band matrix in the diagonal and the $k_l + k_u$ superdiagonals of A; the multipliers (i. e. the elements of L rearranged due to pivoting) used during the factorization are stored in the k_l subdiagonals of A.
`P`	(input) The pivot indices from trf; for $1 \leq i \leq N$, row i of the matrix was interchanged with row $P(i)$.
`B`	(input/output) On entry, the right hand side matrix B. On exit, the solution matrix X.

Returns

$i = 0$	successful exit

sv (gesv)

compute the solution to a real system of linear equations $AX = B$

Synopsis

```
template <typename MA, typename MB>
    int
    sv(GeMatrix<MA> &A, DenseVector<Array<int> > &P, GeMatrix<MB> &B);

template <typename MA, typename VB>
    int
    sv(GeMatrix<MA> &A, DenseVector<Array<int> > &P, DenseVector<VB> &B);
```

Purpose

computes the solution to a real or complex system of linear equations $AX = B$, where A is an $N \times N$ matrix and X and B are $N \times R$ matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor A as

$$A = PLU,$$

where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. The factored form of A is then used to solve the system of equations $AX = B$.

Arguments

`A`	(input/output) On entry, the $N \times N$ coefficient matrix A. On exit, the factors L and U from the factorization $A = PLU$; the unit diagonal elements of L are not stored.
`P`	(output) The pivot indices; for $1 \le i \le \min\{M, N\}$, row i of the matrix was interchanged with row $P(i)$.
`B`	(input/output) On entry, the $N \times R$ matrix of right hand side matrix B. On exit, if function returned 0, the $N \times R$ solution matrix X.

Returns

$i = 0$	successful exit
$i > 0$	then $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

sv (gbsv)

compute the solution to a real system of linear equations $AX = B$, where A is a band matrix of order N with k_l subdiagonals and k_u superdiagonals, and X and B are $N \times R$ matrices

Synopsis

```
template <typename MA, typename MB>
    int
    sv(GbMatrix<MA> &A, DenseVector<Array<int> > &P, GeMatrix<MB> &B);

template <typename MA, typename VB>
    int
    sv(GbMatrix<MA> &A, DenseVector<Array<int> > &P, DenseVector<VB> &B);
```

Purpose

sv computes the solution to a real system of linear equations $AX = B$, where A is a band matrix of order N with k_l subdiagonals and k_u superdiagonals, and X and B are $N \times R$ matrices. The LU decomposition with partial pivoting and row interchanges is used to factor A as $A = LU$, where L is a product of permutation and unit lower triangular matrices with k_l subdiagonals, and U is upper triangular with $k_l + k_u$ superdiagonals. The factored form of A is then used to solve the system of equations $AX = B$.

Arguments

`A`	(input/output) On entry, the matrix A in band storage and on exit overwritten with the LU factorization. Matrix `A` is required to have a total of k_l subdiagonals and $k_l + k_u$ superdiagonals where on entry only the elements within the k_l subdiagonals and k_u superdiagonals need to be set (See Further Details).
`P`	(input) The pivot indices; for $1 \le i \le N$, row i of the matrix was interchanged with row $P(i)$.
`B`	(input/output) On entry, the right hand side matrix B. On exit, the solution matrix X.

Returns

$i = 0$	successful exit

$i > 0$	then $U(i,i)$ is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Further Details

Assume non-zero elements of A reside within a band of k_l subdiagonals and k_u superdiagonals. Then storing its LU factorization requires a band of k_l subdiagonals and $k_l + k_u$ superdiagonals. Hence, the `GbMatrix` holding the matrix A needs to have k_l subdiagonals and $k_l + k_u$ superdiagonals allocated; but only the elements within its k_l subdiagonals and k_u superdiagonals must be set.

If, for example, A is a tridiagonal matrix (i. e. $k_l = k_u = 1$), then matrix U will have in general two super-diagonals. This requires the `GbMatrix` storing A has allocated an additional superdiagonal (as indicated by '$*$'):

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & * & 0 & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & * & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & * \\ 0 & 0 & a_{4,3} & a_{4,4} & a_{4,5} \\ 0 & 0 & 0 & a_{5,4} & a_{5,5} \end{pmatrix} \rightsquigarrow A_{LU} = \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & 0 & 0 \\ m_{2,1} & u_{2,2} & u_{2,3} & u_{2,4} & 0 \\ 0 & m_{3,2} & u_{3,3} & u_{3,4} & u_{3,5} \\ 0 & 0 & m_{4,3} & u_{4,4} & u_{4,5} \\ 0 & 0 & 0 & m_{5,4} & u_{5,5} \end{pmatrix}.$$

Elements of L are stored (in general rearranged due to pivoting) on the subdiagonal of A_{LU}.

trs (trtrs)

solve a triangular system of the form $AX = B$ or $A^T X = B$

Synopsis

```
template <typename MA, typename MB>
    int
    trs(Transpose trans, const TrMatrix<MA> &A, GeMatrix<MB> &B);

template <typename MA, typename VB>
    int
    trs(Transpose trans, const TrMatrix<MA> &A, DenseVector<VB> &B);
```

Purpose

trs solves a triangular system of the form $AX = B$ or $A^T X = B$, where A is a triangular matrix of order N, and B is an $N \times R$ matrix. A check is made to verify that A is nonsingular.

Arguments

`trans`	(input) Specifies the form of the system of equations: `trans = NoTrans` $AX = B$ (No transpose) `trans = Trans` $A^T X = B$ (Transpose) `trans = ConjTrans` $A^H X = B$ (Conjugate transpose = Transpose)
`A`	(input) The (unit or non-unit) triangular matrix A.
`B`	(input/output) On entry, the $N \times R$ matrix of right hand side matrix B. On exit, if function returned 0, the $N \times R$ solution matrix X.

Returns

$i = 0$	successful exit
$i > 0$	then $A(i, i)$ is zero, indicating that the matrix is singular and the solutions X have not been computed.

qrf (geqrf)

compute a QR factorization of a real $M \times N$ matrix A

Synopsis

```
template <typename MA, typename VT>
    int
    qrf(GeMatrix<MA> &A, DenseVector<VT> &tau);
```

Purpose

qrf computes a QR factorization of a real $M \times N$ matrix A:

$$A = QR.$$

Arguments

`A`	(input/output) On entry, the $M \times N$ matrix A. On exit, the elements on and above the diagonal of the array contain the $\min\{M, N\} \times N$ upper trapezoidal matrix R (R is upper triangular if $m \geq n$); the elements below the diagonal, with the array `tau`, represent the orthogonal matrix Q as a product of $\min\{m, n\}$ elementary reflectors (see Further Details).
`tau`	(output) The scalar factors τ of the elementary reflectors (see Further Details).

Returns

$i = 0$	successful exit

Further Details

The matrix Q is represented as a product of elementary reflectors

$$Q = H_1 H_2 \cdots H_k, \text{ where } k = \min\{m, n\}.$$

Each H_i has the form

$$H_i = I - \tau * v * v'$$

where τ is a real scalar, and v is a real vector with $v_1 = \ldots v_{i-1} = 0$ and $v_i = 1$; $(v_{i+1}, \ldots, v_m)$ is stored on exit in `A(i+1,i), .., A(m,i)` and τ in `tau(i)`.

orgqr (orgqr)

generate an $M \times N$ real matrix Q with orthonormal columns

Synopsis

```
template <typename MA, typename VT>
    int
    orgqr(GeMatrix<MA> &A, const DenseVector<VT> &tau);
```

Purpose

orgqr generates an $M \times N$ real matrix Q with orthonormal columns, which is defined as the first N columns of a product of k elementary reflectors of order M

$$Q = H_1 H_2 \cdot \cdots \cdot H_k$$

as returned by qrf.

Arguments

`A`	(input/output) On entry, the i-th column must contain the vector which defines the elementary reflector H_i, for $i = 1, 2, \ldots, k$, as returned by qrf in the first k columns of its matrix argument A. On exit, the $M \times N$ matrix Q.
`tau`	(input) `TAU(i)` must contain the scalar factor of the elementary reflector H_i, as returned by qrf.

Returns

$i = 0$	successful exit

ormqr (ormqr)

Synopsis

```
template <typename MA, typename VT, typename MC>
    int
    ormqr(BlasSide side, Transpose trans,
          const GeMatrix<MA> &A, const DenseVector<VT> &tau,
          GeMatrix<MC> &C);
```

Purpose

ormqr overwrites the general real $M \times N$ matrix C as follows:

$$C \leftarrow \begin{cases} QC & \text{if } \texttt{side=Left} \text{ and } \texttt{trans=NoTrans} \\ Q^TC & \text{if } \texttt{side=Left} \text{ and } \texttt{trans=Trans} \\ CQ & \text{if } \texttt{side=Right} \text{ and } \texttt{trans=NoTrans} \\ CQ^T & \text{if } \texttt{side=Right} \text{ and } \texttt{trans=Trans} \end{cases}$$

where Q is a real orthogonal matrix defined as the product of k elementary reflectors

$$Q = H_1 H_2 \cdot \cdots \cdot H_k$$

as returned by qrf. Q is of order M if `side=Left` and of order N if `side=Right`.

Arguments

`side`	(input)	
	`side = Left`	apply Q or Q^T from left
	`side = Right`	apply Q or Q^T from right
`trans`	(input)	
	`trans=NoTrans`	No transpose, apply Q
	`trans=Trans`	Transpose, apply Q^T
`A`	(input) On entry, the i-th column must contain the vector which defines the elementary reflector H_i, for $i = 1, 2, \ldots, k$, as returned by qrf in the first k columns of its matrix argument A. A is modified by the routine but restored on exit.	
`tau`	(input) `TAU(i)` must contain the scalar factor of the elementary reflector H_i, as returned by qrf.	
`C`	(input/output) On entry, the $M \times N$ matrix C. On exit, C is overwritten by QC or Q^TC or CQ^T or CQ.	

Returns

$i = 0$	successful exit

ls (gels)

solve overdetermined or underdetermined real linear systems involving an $M \times N$ matrix A, or its transpose, using a QR or LQ factorization of A

Synopsis

```
template <typename MA, typename MB>
    int
    ls(Transpose trans, GeMatrix<MA> &A, GeMatrix<MB> &B);
```

Purpose

ls solves overdetermined or underdetermined real linear systems involving an $M \times N$ matrix A, or its transpose, using a QR or LQ factorization of A. It is assumed that A has full rank. The following options are provided:

1. If `trans = NoTrans` and $M \geq N$: find the least squares solution of an overdetermined system, i. e. , solve the least squares problem

$$||B - AX|| \rightarrow \min.$$

2. If `trans = NoTrans` and $M < N$: find the minimum norm solution of an underdetermined system $AX = B$.

3. If `trans = Trans` and $M \geq N$: find the minimum norm solution of an undetermined system $A^T X = B$.

4. If `trans = Trans` and $M < N$: find the least squares solution of an overdetermined system, i. e. , solve the least squares problem

$$||B - A^T * X|| \rightarrow \min.$$

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the $M \times R$ right hand side matrix B and the $N \times R$ solution matrix X.

Arguments

`trans`	(input)	
	`trans = NoTrans`	the linear system involves A;
	`trans = Trans`	the linear system involves A^T.
`A`	(input/output) On entry, the $M \times N$ matrix A. On exit, if $M \geq N$, A is overwritten by details of its QR factorization as returned by qrf; if $M < N$, A is overwritten by details of its LQ factorization as returned by lqf.	

`B` (input/output)
On entry, the matrix B of right hand side vectors, stored column-wise; B is $M \times R$ if `trans=NoTrans`, or $N \times R$ if `trans=Trans`.
On exit, B is overwritten by the solution vectors, stored column-wise:

1. if `trans=NoTrans` and $M \geq N$, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements $N+1$ to M in that column;
2. if `trans=NoTrans` and $M < N$, rows 1 to N of B contain the minimum norm solution vectors;
3. if `trans=Trans` and $M \geq N$, rows 1 to M of B contain the minimum norm solution vectors;
4. if `trans=Trans` and $M < N$, rows 1 to M of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements $M+1$ to N in that column.

Returns

$i = 0$ successful exit

lss (gelss)

compute the minimum norm solution to a real linear least squares problem using a singular value decomposition (SVD)

Synopsis

```
template <typename MA, typename MB>
    int
    lss(GeMatrix<MA> &A, GeMatrix<MB> &B);
```

Purpose

lss computes the minimum norm solution to a real linear least squares problem:

$$||b - Ax||_2 \to \min$$

using the singular value decomposition (SVD) of A. A is an $M \times N$ matrix which may be rank-deficient.

Several right hand side vectors b and solution vectors x can be handled in a single call; they are stored as the columns of the $M \times R$ right hand side matrix B and the $N \times R$ solution matrix X.

The effective rank of A is determined by treating as zero those singular values which are less than RCOND times the largest singular value.

Arguments

`A`	(input/output) On entry, the $M \times N$ matrix A. On exit, the first $\min\{m, n\}$ rows of A are overwritten with its right singular vectors, stored row-wise.
`B`	(input/output) On entry, the $M \times R$ right hand side matrix B. On exit, B is overwritten by the $N \times R$ solution matrix X. If $M \geq N$ and $\operatorname{rank}(A) = N$, the residual sum-of-squares for the solution in the i-th column is given by the sum of squares of elements $B_{N+1,i}, \ldots, B_{M,i}$.

Returns

$i = 0$	successful exit
$i > 0$	the algorithm for computing the SVD failed to converge; more precisely, i off-diagonal elements of an intermediate bi-diagonal form did not converge to zero.

To-do

1. Provide a version of lss for a single right-hand side, i. e. handle the case where B would be a $M \times 1$ matrix (as was done for sv).

2. In this form the wrapper for `gelss` suppresses some of the output computed by its underlying LAPACK routine (e. g. the singular values).

ev (geev,real)

compute for an $N \times N$ real non-symmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors

Synopsis

```
template <typename MA, typename WR, typename WI, typename VL, typename VR>
    int
    ev(bool leftEV, bool rightEV,
       GeMatrix<MA> &A, DenseVector<WR> &wr, DenseVector<WI> &wi,
       GeMatrix<VL> &vl, GeMatrix<VR> &vr);
```

Purpose

ev computes for an $N \times N$ real non-symmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors. The right eigenvector v_j of A satisfies

$$Av_j = \lambda_j v_j$$

where λ_j is its eigenvalue.

The left eigenvector u_j of A satisfies

$$u_j^H A = \lambda_j u_j^H$$

where u_j^H denotes the conjugate transpose of u_j.

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Arguments

`leftEV`	(input) specifies whether left eigenvectors of A are computed.
`rightEV`	(input) specifies whether right eigenvectors of A are computed.
`A`	(input/output) On entry, the $N \times N$ matrix A. On exit, A has been overwritten.
`wr,wi`	(output) `wr` and `wi` contain the real and imaginary parts, respectively, of the computed eigenvalues. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.
`vl`	(output)

If `leftEV=true`, the left eigenvectors u_j are stored one after another in the columns of `vl`, in the same order as their eigenvalues. If `leftEV=false`, then `vl` is not referenced.
If the j-th eigenvalue is real, then u_j is stored in the j-th column of `vl`. If the j-th and $(j+1)$-th eigenvalues form a complex conjugate pair, then

$$u_j = \texttt{vl(\_,j) + i * vl(\_,j+1)}$$

and

$$u_{j+1} = \texttt{vl(\_,j) - i * vl(\_,j+1)}$$

`vr` (output)
If `rightEV=true`, the left eigenvectors u_j are stored one after another in the columns of `vr`, in the same order as their eigenvalues. If `rightEV=false`, then `vr` is not referenced.
If the j-th eigenvalue is real, then u_j is stored in the j-th column of `vr`. If the j-th and $(j+1)$-th eigenvalues form a complex conjugate pair, then

$$u_j = \texttt{vr(\_,j) + i * vr(\_,j+1)}$$

and

$$u_{j+1} = \texttt{vr(\_,j) - i * vr(\_,j+1)}$$

Returns

$i = 0$ successful exit
$i > 0$ the QR algorithm failed to compute all the eigenvalues, and no eigenvectors have been computed; elements $i+1$ to N of `wr` and `wi` contain eigenvalues which have converged.

ev (geev,complex)

compute for an $N \times N$ complex non-symmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors

Synopsis

```
template <typename MA, typename W, typename VL, typename VR>
    int
    ev(bool leftEv, bool rightEv,
       GeMatrix<MA> &A, DenseVector<W> &w, GeMatrix<VL> &vl, GeMatrix<VR> &vr);
```

Purpose

ev computes for an $N \times N$ complex non-symmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors. The right eigenvector v_j of A satisfies

$$Av_j = \lambda_j v_j$$

where λ_j is its eigenvalue.

The left eigenvector u_j of A satisfies

$$u_j^H A = \lambda_j u_j^H$$

where u_j^H denotes the conjugate transpose of u_j.

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

Arguments

`leftEV`	(input) specifies whether left eigenvectors of A are computed.
`rightEV`	(input) specifies whether right eigenvectors of A are computed.
`A`	(input/output) On entry, the $N \times N$ matrix A. On exit, A has been overwritten.
`w`	(output) contains the computed eigenvalues.
`vl`	(output) If `leftEV=true`, the left eigenvectors u_j are stored one after another in the columns of `vl`, in the same order as their eigenvalues. If `leftEV=false`, then `vl` is not referenced.
`vr`	(output) If `rightEV=true`, the left eigenvectors u_j are stored one after another in the columns of `vr`, in the same order as their eigenvalues. If `rightEV=false`, then `vr` is not referenced.

Returns

$i = 0$	successful exit
$i > 0$	the QR algorithm failed to compute all the eigenvalues, and no eigenvectors have been computed; elements $i + 1$ to N of `wr` and `wi` contain eigenvalues which have converged.

ev (syev)

compute all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A

Synopsis

```
template <typename MA, typename VW>
   int
   ev(bool compEV, SyMatrix<MA> &A, DenseVector<VW> &w);
```

Purpose

ev computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix A.

Arguments

`compEV`	(input) specifies whether eigenvectors of A are computed.
`A`	(input/output) On entry, the symmetric matrix A. On successful exit and if `compEV=true`, then the underlying full storage scheme of A contains the orthonormal eigenvectors of the matrix A. If `compEV=false`, then on exit the referenced triangle of the underlying full storage scheme is destroyed.
`w`	(output) On successful exit, the eigenvalues in ascending order.

Returns

$i = 0$	successful exit
$i > 0$	the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

ev (sbev)

compute all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A

Synopsis

```
template <typename MA, typename VW, typename MZ>
    int
    ev(bool compEV, SbMatrix<MA> &A, DenseVector<VW> &w, GeMatrix<MZ> &Z);
```

Purpose

ev computes all the eigenvalues and, optionally, eigenvectors of a real symmetric band matrix A.

Arguments

`compEV`	(input) specifies whether eigenvectors of A are computed.
`A`	(input/output) On entry, the symmetric band matrix A. On exit, A is overwritten by values generated during the reduction to tridiagonal form.
`w`	(output) On successful exit, the eigenvalues in ascending order.
`Z`	If `compEV=true`, then on successful exit, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with $w(i)$. If `compEV=false`, then Z is not referenced.

Returns

$i = 0$	successful exit
$i > 0$	the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

ev (spev)

compute all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix in packed storage

Synopsis

```
template <typename MA, typename VW, typename MZ>
    int
    ev(bool compEV, SpMatrix<MA> &A, DenseVector<VW> &w, GeMatrix<MZ> &Z);
```

Purpose

ev computes all the eigenvalues and, optionally, eigenvectors of a real symmetric matrix in packed storage.

Arguments

`compEV`	(input) specifies whether eigenvectors of A are computed.
`A`	(input/output) On entry, the symmetric matrix A in packed storage format. On exit, A is overwritten by values generated during the reduction to tridiagonal form.
`w`	(output) On successful exit, the eigenvalues in ascending order.
`Z`	If `compEV=true`, then on successful exit, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with $w(i)$. If `compEV=false`, then Z is not referenced.

Returns

$i = 0$	successful exit
$i > 0$	the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

ev (heev)

compute all eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A

Synopsis

```
template <typename MA, typename VW>
    int
    ev(bool compEV, HeMatrix<MA> &A, DenseVector<VW> &w);
```

Purpose

ev computes all eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix A.

Arguments

`compEV`	(input) specifies whether eigenvectors of A are computed.
`A`	(input/output) On entry, the Hermitian matrix A. On successful exit and if `compEV=true`, A contains the orthonormal eigenvectors of the matrix A. If `compEV=false`, then on exit the referenced triangle of the underlying full storage scheme is destroyed.
`w`	(output) On successful exit, the eigenvalues in ascending order.

Returns

$i = 0$	successful exit
$i > 0$	the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

ev (hbev)

compute all the eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A

Synopsis

```
template <typename MA, typename VW, typename MZ>
    int
    ev(bool compEV, HbMatrix<MA> &A, DenseVector<VW> &w, GeMatrix<MZ> &Z);
```

Purpose

ev computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian band matrix A.

Arguments

`compEV`	(input) specifies whether eigenvectors of A are computed.
`A`	(input/output) On entry, the Hermitian band matrix A. On exit, A is overwritten by values generated during the reduction to tridiagonal form.
`w`	(output) On successful exit, the eigenvalues in ascending order.
`Z`	If `compEV=true`, then on successful exit, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with $w(i)$. If `compEV=false`, then Z is not referenced.

Returns

$i = 0$	successful exit
$i > 0$	the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

ev (hpev)

compute all the eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix in packed storage

Synopsis

```
template <typename MA, typename VW, typename MZ>
    int
    ev(bool compEV, HpMatrix<MA> &A, DenseVector<VW> &w, GeMatrix<MZ> &Z);
```

Purpose

ev computes all the eigenvalues and, optionally, eigenvectors of a complex Hermitian matrix in packed storage.

Arguments

`compEV`	(input) specifies whether eigenvectors of A are computed.
`A`	(input/output) On entry, the Hermitian matrix A in packed storage format. On exit, A is overwritten by values generated during the reduction to tridiagonal form.
`w`	(output) On successful exit, the eigenvalues in ascending order.
`Z`	If `compEV=true`, then on successful exit, Z contains the orthonormal eigenvectors of the matrix A, with the i-th column of Z holding the eigenvector associated with $w(i)$. If `compEV=false`, then Z is not referenced.

Returns

$i = 0$	successful exit
$i > 0$	the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

svd (gesvd)

compute the singular value decomposition (SVD) of a complex or real $M \times N$ matrix A, optionally computing the left and/or right singular vectors

Synopsis

```
template <typename MA, typename VS, typename VU, typename VVT>
    int
    svd(SVectorsJob jobu, SVectorsJob jobvt, GeMatrix<MA> &A,
        DenseVector<VS> &S, GeMatrix<VU> &U, GeMatrix<VVT> &VT);

/* calls: svd(All,All,A,s,U,V) */
template <typename MA, typename VS, typename MU, typename MV>
    int
    svd(GeMatrix<MA> &A, DenseVector<VS> &s, GeMatrix<MU> &U, GeMatrix<MV> &VT);
```

Purpose

svd computes the singular value decomposition (SVD) of a real (or complex) $M \times N$ matrix A, optionally computing the left and/or right singular vectors. The SVD is written

$$A = U\Sigma V^T \quad (\text{or } A = U\Sigma V^H)$$

where Σ is an $M \times N$ matrix which is zero except for its $\min\{m,n\}$ diagonal elements, U is an $M \times N$ orthogonal (or unitary) matrix, and V is an $N \times N$ orthogonal (or unitary) matrix. The diagonal elements of Σ are the singular values of A; they are real and non-negative, and are returned in descending order. The first $\min\{m,n\}$ columns of U and V are the left and right singular vectors of A. **Note** that the routine returns V^T (or V^H), not V.

Arguments

`jobu`	Specifies options for computing all or part of the matrix U:	
	`jobu = All`	all M columns of U are returned in matrix `U`
	`jobu = SmallDim`	the first $\min\{m,n\}$ columns of U (the left singular vectors) are returned in the matrix `U`
	`jobu = Overwrite`	the first $\min\{m,n\}$ columns of U (the left singular vectors) are overwritten on the matrix `A`
	`jobu = None`	no columns of U (no left singular vectors) are computed
`jobvt`	Specifies options for computing all or part of the matrix V^T (or V^H):	

`jobu = All`	all N rows of V^T (or V^H) are returned in matrix `VT`
`jobu = SmallDim`	the first $\min\{m, n\}$ rows of V^T (or V^H) (the right singular vectors) are returned in the matrix `VT`
`jobu = Overwrite`	the first $\min\{m, n\}$ rows of V^T (or V^H) (the right singular vectors) are overwritten on the matrix `A`
`jobu = None`	no rows of V^T (or V^H) (no right singular vectors) are computed

Note: `jobu` and `jobvt` can not both be set to be `Overwrite`.

`A` (input/output)
On entry, the $M \times N$ matrix A.
On exit, if `jobu = Overwrite`, A is overwritten with the first $\min\{m, n\}$ columns of U (the left singular vectors, stored columnwise); if `jobvt = Overwrite`, A is overwritten with the first $\min\{m, n\}$ rows of V^T (or V^H) (the right singular vectors, stored rowwise); if `jobu` $\neq$ `Overwrite` and `jobu` $\neq$ `Overwrite`, the contents of A are destroyed.

`S` (output)
The singular values of A, sorted so that $S(i) \geq S(i+1)$.

`U` (output)
If `jobu = All`, `U` contains the $M \times M$ orthogonal (or unitary) matrix U; if `jobu = SmallDim`, `U` contains the first $\min\{m, n\}$ columns of U (the left singular vectors, stored columnwise); if `jobu = None`, then `U` is not referenced.

`VT` (output)
If `jobvt = All`, `VT` contains the $N \times N$ orthogonal (or unitary) matrix V^T (or V^H); if `jobu = SmallDim`, `VT` contains the first $\min\{m, n\}$ rows of V^T (or V^H) (the right singular vectors, stored rowwise); if `jobu = None`, then `VT` is not referenced.

Returns

$i = 0$	successful exit
$i > 0$	the algorithm failed to converge; i specifies how many superdiagonals of an intermediate bidiagonal form B did not converge to zero.