Wiki: https://en.wikipedia.org/wiki/Cholesky_decomposition#Proof_for_positive_semi-definite_matrices

Question: does Cholesky decompostion exists for one general square matrix? What happens when we factorize non-positive definite matrix?

1 (Hermitian) Positive-definite matrix

- Positive-definite matrix: $\boldsymbol{A}$ is one real symmetric (square) maxtrix which satisfies $\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{A}\boldsymbol{x} > \mathbf{0}$ for any non-zero column vector $\boldsymbol{x} \in \mathbb{Z}^{\boldsymbol{n}}$.
  - Postive semi-definite matrix: $\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{A}\boldsymbol{x} \geq \mathbf{0}$
- Hermitian positive-definite matrix: $\boldsymbol{A}$ is one complex **Hermitian** (square) matrix if the scalar $\boldsymbol{x}^{\boldsymbol{H}}\boldsymbol{A}\boldsymbol{x}$ is real and positive for any non-zero column vector $\boldsymbol{x} \in \mathbb{C}^{\boldsymbol{n}}$.
  - Hermitian semi-definite: $\boldsymbol{x}^{\boldsymbol{H}}\boldsymbol{A}\boldsymbol{x} \geq \mathbf{0}$

2 Cholesky decompsition (LL)

The Cholesky decomposition of a Hermitian positive-definite matrix **A** (must one symmetric(square) matrix)is a decomposition of the form

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^{\boldsymbol{H}} \tag{2-1}$$

where $\boldsymbol{L}$ is a lower triangular matrix with real and positive diagonal entries, and $\boldsymbol{L}^{\boldsymbol{H}}$ denotes the conjugate transpose of $\boldsymbol{L}$. Every Hermitian positive-definite matrix (and thus also every real-valued symmetric positive-definite matrix) has a **unique** Cholesky decomposition.

If the matrix $\boldsymbol{A}$ is Hermitian and positive semi-definite, then it still has a decomposition of the form $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^{\boldsymbol{H}}$ if the diagonal entries of $\boldsymbol{L}$ are allowed to be zero.

When **A** has real entries, **L** has real entries as well and the factorization may be written $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^{\boldsymbol{T}}$.

The Cholesky decomposition is unique when **A** is positive definite; there is only one lower triangular matrix **L** with strictly positive diagonal entries such that $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^{\boldsymbol{T}}$. However, the decomposition need not be unique when **A** is positive semidefinite.

The converse holds trivially: if **A** can be written as $\boldsymbol{L}\boldsymbol{L}^{\boldsymbol{T}}$ for some invertible **L**, lower triangular or otherwise, then **A** is Hermitian and positive definite.

2.1 Variant of LDL decomposition

A closely related variant of the classical Cholesky decomposition is the LDL decomposition,

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^{\boldsymbol{H}} \tag{2-2}$$

where **L** is a lower unit triangular (unitriangular) matrix and **D** is a diagonal matrix.

This decomposition is related to the classical Cholesky decomposition, of the form $\boldsymbol{L}\boldsymbol{L}^{\boldsymbol{H}}$, as follows:

$$\begin{aligned} \boldsymbol{A} &= \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^{\boldsymbol{H}} \\ &= \boldsymbol{L}\boldsymbol{D}^{\frac{1}{2}}\left(\boldsymbol{D}^{\frac{1}{2}}\right)^{\boldsymbol{H}}\boldsymbol{L}^{\boldsymbol{H}} \\ &= \boldsymbol{L}\boldsymbol{D}^{\frac{1}{2}}\left(\boldsymbol{L}\boldsymbol{D}^{\frac{1}{2}}\right)^{\boldsymbol{H}} \\ &= \boldsymbol{L}'(\boldsymbol{L}')^{\boldsymbol{H}} \end{aligned} \tag{2-3}$$

The **LDL** variant, if efficiently implemented, requires the same space and computational complexity to construct and use but avoids extracting square roots.[5] Some indefinite matrices for which no Cholesky decomposition exists have an **LDL** decomposition with negative entries in **D**. For these reasons, the LDL decomposition may be preferred. For real matrices, the factorization has the form **A = LDL^T** and is often referred to as **LDLT decomposition** (or LDL^T decomposition). It is closely related to the eigendecomposition of real symmetric matrices, **A=QΛQ^T**.

2.2 Applications

The Cholesky decomposition is mainly used for the numerical solution of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$. If **A** is symmetric and positive definite, then we can solve $\boldsymbol{x}$ by first computing the Cholesky decomposition $\boldsymbol{A} = \boldsymbol{LL^T}$, then solving $\boldsymbol{Ly} = \boldsymbol{b}$ for **y** by forward substitution, and finally solving $\boldsymbol{L^T x} = \boldsymbol{y}$ for **x** by back substitution.

An alternative way to eliminate taking square roots in the $\boldsymbol{LL^T}$ decomposition is to compute the Cholesky decomposition $\boldsymbol{A} = \boldsymbol{LDL^T}$, then solving $\boldsymbol{Ly} = \boldsymbol{b}$ for **y**, and finally solving $\boldsymbol{DL^T} = \boldsymbol{y}$.

For linear systems that can be put into **symmetric** form, the Cholesky decomposition (or its LDL variant) is the method of choice, for superior efficiency and numerical stability. Compared to the LU decomposition, it is roughly twice as efficient.

2.2.1 Linear least squares

Systems of the form **Ax** = **b** with **A** symmetric and positive definite arise quite often in applications. For instance, the normal equations in linear least squares problems are of this form. It may also happen that matrix **A** comes from an energy functional which must be positive from physical considerations; this happens frequently in the numerical solution of partial differential equations.

2.2.2 Non-linear optimization

2.2.3 Monte Carlo simulation

2.2.4 Kalman filters

2.2.5 Matrix inversion

The explicit inverse of a **Hermitian matrix** can be computed via Cholesky decomposition, in a manner similar to solving linear systems, using n^3 operations ($\frac{1}{2}n^3$ multiplications).[5] The entire inversion can even be efficiently performed **in-place**.

A non-Hermitian matrix **B** can also be inverted using the following identity, where **BB*** will always be Hermitian:

$$\boldsymbol{B^{-1}} = \boldsymbol{B^H}(\boldsymbol{BB^H})^{-1}\ \boldsymbol{or} \\ \boldsymbol{B^{-1}} = (\boldsymbol{B^H B})^{-1}\boldsymbol{B} \tag{2-4}$$

2.3 Computation

There are various methods for calculating the Cholesky decomposition. The computational complexity of commonly used algorithms is *O*(*n*^{3}) in general. The algorithms described below all involve about *n*^{3}/3 FLOPs, where *n* is the size of the matrix **A**. Hence, they are half the cost of the LU decomposition, which uses 2*n*^{3}/3 FLOPs (see Trefethen and Bau 1997).

Which of the algorithms below is faster depends on the details of the implementation. Generally, the first algorithm will be slightly slower because it accesses the data in a less regular manner.

2.3.1 The Cholesky algorithm

The **Cholesky algorithm**, used to calculate the decomposition matrix L, is a modified version of Gaussian elimination.

The recursive algorithm starts with $i := 1$ and

$$A^{(1)} := A$$

At step i, the matrix $A^{(i)}$ has the following form:

$$\boldsymbol{A}^{(i)} := \begin{pmatrix} \boldsymbol{I}_{i-1} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{a}_{i,i} & \boldsymbol{b}_i^H \\ \boldsymbol{0} & \boldsymbol{b}_i & \boldsymbol{B}^{(i)} \end{pmatrix} \tag{2-5}$$

If we now define the matrix $\boldsymbol{L}_i$ by

$$\boldsymbol{L}_i := \begin{pmatrix} \boldsymbol{I}_{i-1} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \sqrt{\boldsymbol{a}_{i,i}} & \boldsymbol{0} \\ \boldsymbol{0} & \frac{\boldsymbol{1}}{\sqrt{\boldsymbol{a}_{i,i}}}\boldsymbol{b}_i & \boldsymbol{I}_{n-i} \end{pmatrix} \tag{2-6}$$

Then we can write **A**$^{(i)}$ as

$$\boldsymbol{A}^{(i)} = \boldsymbol{L}_i \boldsymbol{A}^{(i+1)} \boldsymbol{L}_i^H \tag{2-7}$$

where

$$\boldsymbol{A}^{(i+1)} := \begin{pmatrix} \boldsymbol{I}_{i-1} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \left(\boldsymbol{B}^{(i)} - \frac{1}{a_{i,i}}\boldsymbol{b}_i\boldsymbol{b}_i^H\right) \end{pmatrix} \tag{2-8}$$

Note that $\boldsymbol{b}_i\boldsymbol{b}_i^H$ is an outer product, therefore this algorithm is called the outer product version in (Golub & Van Loan).

We repeat this for I from 1 to n. After n steps, we get $\boldsymbol{A}^{(n+1)} = \boldsymbol{I}$. Hence, the lower triangular matrix L we are looking for is calculated as

$$\boldsymbol{L} := \boldsymbol{L}_1\boldsymbol{L}_2 \dots \boldsymbol{L}_n \tag{2-9}$$

Since

$$\begin{aligned} \boldsymbol{A} = \boldsymbol{A}^{(1)} &= \boldsymbol{L}_1\boldsymbol{A}^{(2)}\boldsymbol{L}_1^H \\ &= \boldsymbol{L}_2\boldsymbol{L}_1\boldsymbol{A}^{(3)}\boldsymbol{L}_1^H\boldsymbol{L}_2^H \\ &= \boldsymbol{L}_1\boldsymbol{L}_2 \dots \boldsymbol{L}_n\boldsymbol{A}^{(n+1)}\boldsymbol{L}_n^H \dots \boldsymbol{L}_2^H\boldsymbol{L}_1^H \\ &= \boldsymbol{LIL}^H \\ &= \boldsymbol{LL}^H \end{aligned} \tag{2-10}$$

2.3.1.1 HW/SW implementaiton

For the computation in (2-5) to (2-9):

$$\boldsymbol{L}_i \coloneqq \begin{pmatrix} \boldsymbol{I}_{i-1} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \sqrt{\boldsymbol{a}_{i,i}} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \frac{\boldsymbol{1}}{\sqrt{\boldsymbol{a}_{i,i}}}\boldsymbol{L}_{i-1}(\boldsymbol{i}+\boldsymbol{1},\boldsymbol{i}) & \boldsymbol{1} & \cdots & \boldsymbol{0} \\ \cdots & \cdots & \cdots. & \cdots & \cdots \\ \boldsymbol{0} & \frac{\boldsymbol{1}}{\sqrt{\boldsymbol{a}_{i,i}}}\boldsymbol{L}_{i-1}(\boldsymbol{n},\boldsymbol{i}) & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{pmatrix} \tag{2-11}$$

$$\boldsymbol{L}_{i+1} \coloneqq \begin{pmatrix} \boldsymbol{I}_{i-1} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \sqrt{\boldsymbol{a}_{i+1,i+1}} & \cdots & \boldsymbol{0} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \boldsymbol{0} & \boldsymbol{0} & \frac{\boldsymbol{1}}{\sqrt{\boldsymbol{a}_{i+1,i+1}}}\boldsymbol{b}_{i+1} & \boldsymbol{0} & \boldsymbol{1} \end{pmatrix}$$

Then

$$\boldsymbol{L}_i\boldsymbol{L}_{i+1} = \begin{bmatrix} \boldsymbol{I}_{i-1} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \sqrt{\boldsymbol{a}_{i,i}} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \frac{\boldsymbol{1}}{\sqrt{\boldsymbol{a}_{i,i}}}\boldsymbol{L}_{i-1}(\boldsymbol{i}+\boldsymbol{1},\boldsymbol{i}) & \sqrt{\boldsymbol{a}_{i+1,i+1}} & \cdots & \boldsymbol{0} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \boldsymbol{0} & \frac{\boldsymbol{1}}{\sqrt{\boldsymbol{a}_{i,i}}}\boldsymbol{L}_{i-1}(\boldsymbol{n},\boldsymbol{i}) & \frac{\boldsymbol{1}}{\sqrt{\boldsymbol{a}_{i+1,i+1}}}\boldsymbol{b}_{i+1} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \tag{2-12}$$

So one can use the same place of $\boldsymbol{A}$ to calculate the $\boldsymbol{L}$.

below is one HW/SW efficient implemenation:

- Initial: $k = 1, \boldsymbol{L}_{\boldsymbol{k}} := \boldsymbol{A}$
- Step k:
  - $\sqrt{\boldsymbol{a}_{i,i}} \Rightarrow L_{k,k} = \sqrt{L_{k,k}}$
  - $\frac{\boldsymbol{1}}{\sqrt{\boldsymbol{a}_{i,i}}}\boldsymbol{b}_i \Rightarrow For\ i = k+1:n, L_{i,k} = L_{i,k}/L_{k,k}$
  - $\boldsymbol{B}^{(i)} - \frac{1}{a_{i,i}}\boldsymbol{b}_i\boldsymbol{b}_i^H \Rightarrow$ <u>**and as it's a Hermitian/Symmetric matrix, only need calculate and store the lower-left part**</u>

```
▪ For j = k + 1: n  % columns
      for i =j:n % rows
          $L_{i,j} = L_{i,j} - L_{i,k}L_{j,k}^*$
```

- After n steps, the lower-left of L is the LL decompostion

```
▪ For i = 1: n
      for j =i+1:n
          $L_{i,j} = 0$
```

```
%% Cholesky Algorithm
L = A;
for k=1:n
  L(k,k) = sqrt(L(k,k));
  for i=(k+1):n
    if (L(i,k)~=0)
      L(i,k) = L(i,k)/L(k,k);
    end
  end
  for j=(k+1):n % columns
    for i=j:n % rows
      L(i,j) = L(i,j)-L(i,k)*L(j,k)';
    end
  end
end
for i=1:n
  for j=i+1:n
    L(i,j) = 0;
  end
end
```
```
lseif strcmpi(CholAlg, 'CholBan')
```

2.3.2 The Cholesky–Banachiewicz and Cholesky–Crout algorithms

If we write out the equation $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^{\boldsymbol{H}}$

$$\begin{aligned}\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^{\boldsymbol{H}} &= \begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} L_{11}^* & L_{21}^* & L_{31}^* \\ 0 & L_{22}^* & L_{32}^* \\ 0 & 0 & L_{33}^* \end{bmatrix} \\ &= \begin{bmatrix} L_{11}L_{11}^* & L_{11}L_{21}^* & L_{11}L_{31}^* \\ L_{21}L_{11}^* & L_{21}L_{21}^* + L_{22}L_{22}^* & L_{21}L_{31}^* + L_{22}L_{32}^* \\ L_{31}L_{11}^* & L_{31}L_{21}^* + L_{32}L_{22}^* & L_{31}L_{31}^* + L_{32}L_{32}^* + L_{33}L_{33}^* \end{bmatrix} \\ &\underset{\text{Hermitian}}{=} \begin{bmatrix} L_{11}^2 & L_{11}L_{21}^* & L_{11}L_{31}^* \\ L_{21}L_{11} & L_{21}L_{21}^* + L_{22}^2 & L_{21}L_{31}^* + L_{22}L_{32}^* \\ L_{31}L_{11} & L_{31}L_{21}^* + L_{32}L_{22} & L_{31}L_{31}^* + L_{32}L_{32}^* + L_{33}^2 \end{bmatrix}\end{aligned} \tag{2-13}$$

we obtain the following formula for the entries of $\boldsymbol{L}$:

$$\begin{aligned} L_{j,j} &= \sqrt{A_{j,j} - \sum_{k=1}^{k=j-1} L_{j,k}L_{j,k}^*} \\ L_{i,j} &= \frac{1}{L_{jj}}\left(A_{i,j} - \sum_{k=1}^{k=j-1} L_{i,k}L_{j,k}^*\right) \text{ for i>j} \end{aligned} \tag{2-14}$$

For real symmetric matrix, the following formula applies:

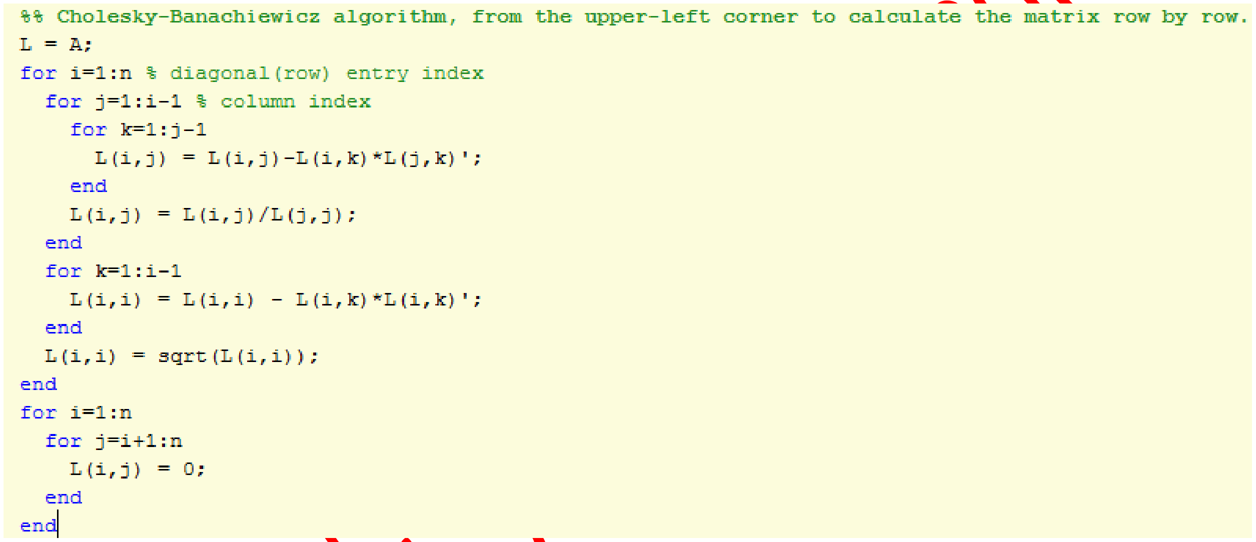
(2-15)

$$L_{i,j} = \frac{1}{L_{jj}}\left(A_{i,j} - \sum_{k=1}^{k=j-1} L_{i,k}L_{j,k}\right) \text{ for i>j}$$

The expression under the square root is always positive if $\boldsymbol{A}$ is real and positive-definite.

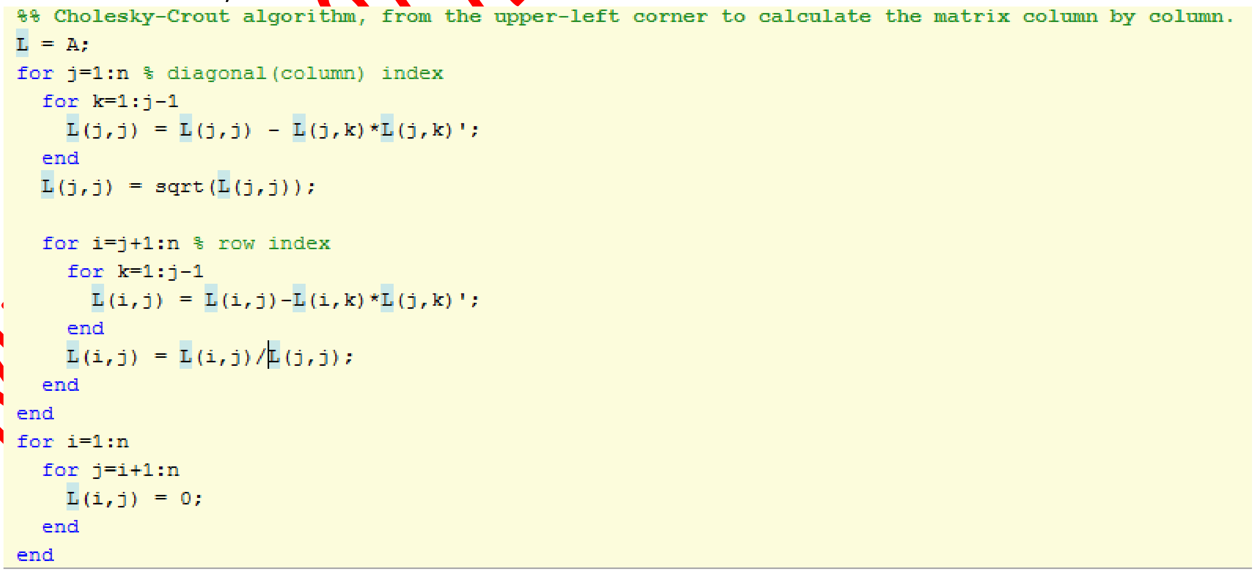
So we can compute the (*i*, *j*) entry if we know the entries to the left and above. The computation is usually arranged in either of the following orders.

- The **Cholesky–Banachiewicz algorithm** starts from the upper left corner of the matrix *L* and proceeds to calculate the matrix row by row.



```
%% Cholesky-Banachiewicz algorithm, from the upper-left corner to calculate the matrix row by row.
L = A;
for i=1:n % diagonal(row) entry index
  for j=1:i-1 % column index
    for k=1:j-1
      L(i,j) = L(i,j)-L(i,k)*L(j,k)';
    end
    L(i,j) = L(i,j)/L(j,j);
  end
  for k=1:i-1
    L(i,i) = L(i,i) - L(i,k)*L(i,k)';
  end
  L(i,i) = sqrt(L(i,i));
end
for i=1:n
  for j=i+1:n
    L(i,j) = 0;
  end
end
```

- The **Cholesky–Crout algorithm** starts from the upper left corner of the matrix *L* and proceeds to calculate the matrix column by column.



```
%% Cholesky-Crout algorithm, from the upper-left corner to calculate the matrix column by column.
L = A;
for j=1:n % diagonal(column) index
  for k=1:j-1
    L(j,j) = L(j,j) - L(j,k)*L(j,k)';
  end
  L(j,j) = sqrt(L(j,j));

  for i=j+1:n % row index
    for k=1:j-1
      L(i,j) = L(i,j)-L(i,k)*L(j,k)';
    end
    L(i,j) = L(i,j)/L(j,j);
  end
end
for i=1:n
  for j=i+1:n
    L(i,j) = 0;
  end
end
```

Either pattern of access allows the entire computation to be performed in-place if desired.

2.3.3 Stability of the computation

Suppose that we want to solve a well-conditioned system of linear equations. **If the LU decomposition is used, then the algorithm is unstable unless we use some sort of pivoting strategy**. In the latter case, the error depends on the so-called growth factor of the matrix, which is usually (but not always) small.

Now, suppose that the **Cholesky decomposition** is applicable. As mentioned above**, the algorithm will be twice as fast**. Furthermore, **no pivoting is necessary and the error will always be small**. Specifically, if we want to solve $\boldsymbol{Ax} = \boldsymbol{b}$, and $\boldsymbol{y}$ denotes the computed solution, then y solves the disturbed system $(\boldsymbol{A} + \boldsymbol{E})\boldsymbol{y} = \boldsymbol{b}$ where

$$\|\boldsymbol{E}\|_2 \leq c_n \varepsilon \|\boldsymbol{A}\|_2 \tag{2-16}$$

Here, $\|*\|_2$ is the matrix 2-norm, c_n is a small constant depending on n, and ε denotes the unit round-off.

- Matrix 2-norm

$$\|\boldsymbol{A}\|_2 = \sqrt{\lambda_{max}(\boldsymbol{A}^{\boldsymbol{H}}\boldsymbol{A})} = \sigma_{max}(\boldsymbol{A}) \tag{2-17}$$

  - Whcih means it is the largest **singular value** of $\boldsymbol{A}$ i.e. the square root of the largest **eigenvalue** of the **postive-semidefinite** matrix $\boldsymbol{A}^{\boldsymbol{H}}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^{\boldsymbol{H}}$
- Unit round-off
  - Which is the fix-point round off unit and depends on the wordlength

One concern with the Cholesky decomposition to be aware of is the use of square roots. If the matrix being factorized is positive definite as required, the numbers under the square roots are always positive *in exact arithmetic*. **Unfortunately, the numbers can become negative because of round-off errors, in which case the algorithm cannot continue. However, this can only happen if the matrix is very ill-conditioned**. **One way to address this is to add a diagonal correction matrix** to the matrix being decomposed in an attempt to promote the positive-definiteness.[7] While this might lessen the accuracy of the decomposition, **it can be very favorable for other reasons; for example, when performing Newton's method in optimization, adding a diagonal matrix can improve stability when far from the optimum.**

2.3.4 LDL decomposition

An alternative form, eliminating the need to take square roots, is the symmetric indefinite factorization

$$\begin{aligned}\boldsymbol{A} = \boldsymbol{LL^H} &= \begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{bmatrix}\begin{bmatrix} L_{11}^* & L_{21}^* & L_{31}^* \\ 0 & L_{22}^* & L_{32}^* \\ 0 & 0 & L_{33}^* \end{bmatrix} \\ &= \begin{bmatrix} L_{11}L_{11}^* & L_{11}L_{21}^* & L_{11}L_{31}^* \\ L_{21}L_{11}^* & L_{21}L_{21}^* + L_{22}L_{22}^* & L_{21}L_{31}^* + L_{22}L_{32}^* \\ L_{31}L_{11}^* & L_{31}L_{21}^* + L_{32}L_{22}^* & L_{31}L_{31}^* + L_{32}L_{32}^* + L_{33}L_{33}^* \end{bmatrix} \\ &\overline{\overline{\text{Hermitian}}} \begin{bmatrix} L_{11}^2 & L_{11}L_{21}^* & L_{11}L_{31}^* \\ L_{21}L_{11} & L_{21}L_{21}^* + L_{22}^2 & L_{21}L_{31}^* + L_{22}L_{32}^* \\ L_{31}L_{11} & L_{31}L_{21}^* + L_{32}L_{22} & L_{31}L_{31}^* + L_{32}L_{32}^* + L_{33}^2 \end{bmatrix}\end{aligned} \tag{2-18}$$

If **A** is real, the following recursive relations apply for the entries of **D** and **L**

$$D_j = A_{jj} - \sum_{k=1}^{j-1} L_{jk}^2 D_k$$

$$L_{ij} = \frac{1}{D_j}\left(A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk} D_k\right), \qquad \text{for } i > j.$$

For complex Hermitian matrix **A**, the following formula applies:

$$D_j = A_{jj} - \sum_{k=1}^{j-1} L_{jk} L_{jk}^* D_k$$

$$L_{ij} = \frac{1}{D_j}\left(A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk}^* D_k\right), \qquad \text{for } i > j.$$

Again, the pattern of access allows the entire computation to be performed in-place if desired.

2.3.5 Block variant

2.3.6 Updating the decomposition