1 February 11th, 2021

1.1 LU Decomposition Continued

Recall that:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \quad L = L = \begin{bmatrix} \ell_{11} & & & \\ \ell_{21} & \ell_{22} & & \\ \vdots & \vdots & \ddots & \\ \ell_{n1} & \ell_{n2} & & \ell_{nn} \end{bmatrix} \quad L^T = \begin{bmatrix} \ell_{11} & \ell_{21} & \dots & \ell_{n_1} \\ & \ell_{22} & \dots & \ell_{n_2} \\ & & \ddots & \vdots \\ & & & \ell_{nn} \end{bmatrix}$$

Observing the first column of LL^T, we have:

- $a_{11} = \ell_{11}^2 \implies \ell_{11} = \sqrt{a_{11}}$
- $a_{21} = \ell_{21}\ell_{11} \implies \ell_{21} = a_{21}/\ell_{11}$
- $a_{n1} = \ell_{n1}\ell_{11} \implies \ell_{n1} = a_{n1}/\ell_{11}$

Remark 1.1 — Note that we only need to compare the lower triangular part of LL^T, since it is symmetric.

For the second column, note that we have:

$$a_{k2} = \ell_{k1}\ell_{21} + \ell_{k2}\ell_{22}$$

As such, we have:

- $a_{22} = \ell_{21}^2 + \ell_{22}^2 \implies \ell_{22} = (a_{22} - \ell_{21}^2)^{\frac{1}{2}}$
- $\ell_{k2} = (a_{k2} - \ell_{k1}\ell_{21})\ell_{22}$ for $k = 3, \dots, n$.

We can continue this process, and for the k-th column, we have:

$$a_{kk} = \ell_{k1}^2 + \ell_{k2}^2 + \ldots + \ell_{kk}^2 \implies \ell_{kk} = (a_{kk} - \sum_{i=1}^{k-1} \ell_{ki}^2)^{\frac{1}{2}}$$

and:

$$a_{ik} = \sum_{j=1}^{k} \ell_{ij}\ell_{kj} \implies \ell_{kj} = a_{ik} - (\sum_{j=1}^{k-1} \ell_{ij}\ell_{kj})/\ell_{kk}$$

for $i = k+1, \dots, n$. Thus, we have Algorithm 1.

However, Algorithm 1 is not memory efficient. To create a memory efficient version, we should use the lower triangular part of L to overwrite that of A. This gives us Algorithm 2.

Remark 1.2 — The only difference between Algorithm 1 and 2 is that we change ℓ to a.

Using BLAS, we have Algorithm 3.

Algorithm 1 Cholesky Decomposition Version 1

for $k = 1 : n$ **do**
  $\ell_{kk} = (a_{kk} - \sum_{i=1}^{k-1} \ell_{ki}^2)^{1/2}$
  for $i = k + 1 : n$ **do**
    $\ell_{ik} = (a_{ik} - \sum_{j=1}^{k-1} \ell_{ij}\ell_{kj})/\ell_{kk}$
  end for
end for

Algorithm 2 Cholesky Decomposition Version 2 - Memory Efficient

for $k = 1 : n$ **do**
  $a_{kk} = (a_{kk} - \sum_{i=1}^{k-1} a_{ki}^2)^{1/2}$
  for $i = k + 1 : n$ **do**
    $a_{ik} = (a_{ik} - \sum_{j=1}^{k-1} a_{ij}a_{kj})/a_{kk}$
  end for
end for

Algorithm 3 Cholesky Decomposition Version 3 - BLAS

for $k = 1 : n$ **do**
  $A(k, k) = A(k, k) - \text{norm}(A(k, 1 : k - 1), 2)^2$
  $A(k, k) = (A(k, k))^{1/2}$
  $A(k + 1 : n, k) = A(k + 1 : n, k) - A(k + 1 : n, 1 : k - 1)(A(k, 1 : k - 1))^T$
  $A(k + 1 : n, k) = A(k + 1 : n, k)/A(k, k)$
end for

Theorem 1.3

Cholesky decomposition has a complexity of:

$$\frac{1}{3}n^3 + O(n^2)$$

which is only half of LU for general A. In addition, the memory required is also halved.

Consider $A = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}$. We have:

- Step 1: $\begin{bmatrix} \sqrt{2} & & \\ \frac{1}{\sqrt{2}} & 2 & \\ 0 & 1 & 2 \end{bmatrix}$
- Step 2: $\begin{bmatrix} \sqrt{2} & & \\ \frac{1}{\sqrt{2}} & \sqrt{\frac{3}{2}} & \\ 0 & \sqrt{\frac{2}{3}} & 2 \end{bmatrix}$

- Step 3: $\begin{bmatrix} \sqrt{2} & & \\ \frac{1}{\sqrt{2}} & \sqrt{\frac{3}{2}} & \\ 0 & 1 & \sqrt{\frac{4}{3}} \end{bmatrix}$

This gives us:

$$L = \begin{bmatrix} \sqrt{2} & & \\ \frac{1}{\sqrt{2}} & \sqrt{\frac{3}{2}} & \\ 0 & 1 & \sqrt{\frac{4}{3}} \end{bmatrix}$$

Remark 1.4 — The SPD-ness of A guarantees that $a_{kk} - \sum_{i=1}^{k-1} \ell_{ki}^2$ is positive.

1.2 LU on Banded Matrices

Definition 1.5. A matrix is **banded** if there exists an integer $p > 0$ such that:

$$a_{ij} \neq 0 \implies |i - j| \leq p$$

Example 1.6

. For $p = 1$, the matrix is called a **tri-diagonal matrix**:

$$A = \begin{bmatrix} \times & \times & & & & \\ \times & \times & \ddots & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & \times \\ & & & & \times & \times \end{bmatrix} \in \mathbb{R}^{n \times n}$$

If we apply LU decomposition to a tri-diagonal matrix, we have:

- Step 1:

$$A(2,1) = A(2,1)/A(1,1)$$
$$A(2,2) = A(2,2) - A(2,1)A(1,2)$$

- Step k:

$$A(k+1,k) = A(k+1,k)/A(kk)$$
$$A(k+1,k+1) = A(k+1,k+1) - A(k+1,k)A(k,k+1)$$

This leads to Algorithm 4

Remark 1.7 — Note that the resulting matrix is also tri-diagonal.

Algorithm 4 LU Decomposition for Tri-Diagonal Matrix

1: **for** $k = 1 : n-1$ **do**
2: $A(k+1,k) = A(k+1,k)/A(k,k)$
3: $A(k+1,k+1) = A(k+1,k+1) - A(k+1,k)A(k,k+1)$
4: **end for**

Theorem 1.8
The computation complexity for Algorithm 4 is $\sum_{k=1}^{n-1} 3 = 3(n-1) \sim O(n)$.

Theorem 1.9
In general, for a banded matrix with band width p, the computation cost is $O(np^2)$.

Example 1.10
The discrete Laplacian matrix in 1-D is a banded matrix with $p = 1$ (tri-diagonal).

Example 1.11
The discrete Laplacian in 2-D on a grid is:

$$A_1 \otimes I + I \otimes A_2 \in \mathbb{R}^{N \times N},$$

where $N = n^2$, $A_1, A_2 \in \mathbb{R}^{n \times n}$ are the 1-D discrete Laplacian matrix for each dimension, it is a banded matrix with $p = n$.

Remark 1.12 — If we applied GE to example 1.11, it would be $O(Np^2) = O(N^2)$, which is significantly lower than $O(N^3)$.

Our goal is to improve this cost to $O(N)$.

1.3 LU on Spare Matrix

Definition 1.13. A **sparse matrix** is a matrix with many zero entries.

Remark 1.14 — Banded matrices with small bandwith can be considered to be sparse.

Sparse matrices arise from many numerical solutions of PDE, as differential operators are local, making many of the entries zero. Sometimes, we can exploit the structure of sparse matrix to speed up LU decomposition. However, LU does not always work well.

Example 1.15

Consider $\begin{bmatrix} 5 & 4 & 3 & 2 \\ 4 & 4 & & \\ 3 & & 3 & \\ 2 & & & 2 \end{bmatrix}$. Note that this is sparse. However, when we divide the first row, we will modify the lower sub matrix, making the final result a dense matrix. Thus it would be the same as applying LU to a general dense matrix, as the sparse structure is destroyed.

Now, we should ask, can we keep as many zeros as possible in LU? The answer is yes, by re-ordering both the equations and the unknowns, i.e. permute the matrix.

Example 1.16

If we consider the matrix in 1.15, we could permute it so we get:

$$\begin{bmatrix} 2 & & & 2 \\ & 3 & & 3 \\ & & 4 & 4 \\ 2 & 3 & 4 & 5 \end{bmatrix}$$

If we perform LU on this matrix, the sparse structure is preserved.

Now we must ask, how should we re-order to allow LU to produce the minimum number of **fill-in's**? This is a very difficult topic to answer, and was an active research topic. To answer this, we use graph theory. This is beyond the scope of this course.