Online Lecture Notes

Prof. Boris Houska

April 21, 2022

1 Gauss Elimination

The main steps of Gauss elimination for solving the linear equations systems,

$$Ax = b,$$

are

1. Permute the rows of A such that $a_{1,1} \neq 0$.
2. Next, set $q_j = -\frac{a_{j,1}}{a_{1,1}}$ and introduce the Frobenius matrix

$$G_1 = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ q_2 & 1 & 0 & \dots & 0 \\ q_3 & 0 & 1 & \dots & 0 \\ \vdots & & & \ddots & \\ q_n & 0 & 0 & \dots & 1 \end{pmatrix}$$

   which can be stored by working out the factors q_j for $j \in \{2, 3, \dots, n\}$.

This is useful because the matrix $C_1 = G_1 A$ is given by

$$C_1 = G_1 A = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ q_2 & 1 & 0 & \dots & 0 \\ q_3 & 0 & 1 & \dots & 0 \\ \vdots & & & \ddots & \\ q_n & 0 & 0 & \dots & 1 \end{pmatrix} \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \dots & a_{3,n} \\ \vdots & & & \ddots & \\ a_{n,1} & a_{n,2} & a_{n,3} & \dots & a_{n,n} \end{pmatrix}$$

$$= \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ q_2 a_{1,1} + a_{2,1} & q_2 a_{1,2} + a_{2,2} & q_2 a_{1,3} + a_{2,3} & \dots & q_2 a_{1,n} + a_{2,n} \\ q_3 a_{1,1} + a_{3,1} & q_3 a_{1,2} + a_{3,2} & q_3 a_{1,3} + a_{3,3} & \dots & q_3 a_{1,n} + a_{3,n} \\ \vdots & & & \ddots & \\ q_n a_{1,1} + a_{n,1} & q_n a_{1,2} + a_{n,2} & q_3 a_{1,3} + a_{n,3} & \dots & q_3 a_{1,n} + a_{n,n} \end{pmatrix}$$

$$= \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ 0 & q_2 a_{1,2} + a_{2,2} & q_2 a_{1,3} + a_{2,3} & \dots & q_2 a_{1,n} + a_{2,n} \\ 0 & q_3 a_{1,2} + a_{3,2} & q_3 a_{1,3} + a_{3,3} & \dots & q_3 a_{1,n} + a_{3,n} \\ \vdots & & & \ddots & \\ 0 & q_n a_{1,2} + a_{n,2} & q_3 a_{1,3} + a_{n,3} & \dots & q_3 a_{1,n} + a_{n,n} \end{pmatrix}$$

where the last equation follows since we have defined the factors q_j such that $q_j a_{1,1} + a_{j,1} = 0$.

3. The key observation of Gauss is that we can keep on doing this until the matrix becomes upper triangular. For this keep on permuting the rows such that we never divided by zero. For instance if $q_2 a_{1,2} + a_{2,2} \neq 0$ we can directly continue by introducing next Frobenius matrix of the form

$$G_2 = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & q_{2,3} & 1 & \dots & 0 \\ \vdots & \vdots & & \ddots & \\ 0 & q_{2,n} & 0 & \dots & 1 \end{pmatrix}$$

such that the matrix $C_2 = G_2 C_1$ has the structure

$$C_2 = \begin{pmatrix} \star & \star & \star & \dots & \star \\ 0 & \star & \star & \dots & \star \\ 0 & 0 & \star & \dots & \star \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & \star & \dots & \star \end{pmatrix}$$

where the $\star$s are placeholders for non-zero coefficients.

4. If we keep on doing this (to make it easier without permutation), we obtain an upper triangular matrix of the form

$$R = G_{n-1}G_{n-2}G_{n-3}\dots G_2G_1A$$

This means that the corresponding linear equations system

$$Rx = G_{n-1}G_{n-2}G_{n-3}\dots G_2G_1b$$

is equivalent to our original equations system $Ax = b$. Here, we use that the Frobenius matrix G_i are invertible, since these matrices are lower triangular and have ones on the diagonal (this means that we can invert by "forward substitution"). For instance

$$G_2^{-1} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & q_{2,3} & 1 & \dots & 0 \\ \vdots & \vdots & & \ddots & \\ 0 & q_{2,n} & 0 & \dots & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & -q_{2,3} & 1 & \dots & 0 \\ \vdots & \vdots & & \ddots & \\ 0 & -q_{2,n} & 0 & \dots & 1 \end{pmatrix}$$

5. Also observe that the product of lower triangular matrices remains lower triangular. This means that

$$L = [G_{n-1}G_{n-2}G_{n-3}\dots G_2G_1]^{-1} = G_1^{-1}G_2^{-1}\dots G_{n-3}^{-1}G_{n-2}^{-1}G_{n-1}^{-1}$$

is a lower triangular matrix. Notice that this definition is such that

$$LR = A\,,$$

where L is a lower triangular matrix with ones on the diagonal and R an upper triangular matrix.

6. The final step of the Gauss-elimination is to implement

$$x = R^{-1}\left[L^{-1}b\right]$$

by one forward substitution for solving the lower triangular system and then one backward recursion for solving the upper triangular system (see our previous lecture notes).

Summary: By using Gauss elimination we can explicitly construct a lower triangular matrix L with ones on the diagonal and an upper triangular matrix R such that

$$A = LR\,.$$

This formula is called the "LR-decomposition" of the matrix A. The computational complexity for computing L and R is of order $O(n^3)$, since the complexity of working out the coefficients of $C_1 = G_1A$ is of order $O((n-1)^2)$, the complexity for working out $C_2 = G_2C_1$ is $O((n-2)^2)$, and so on. Thus, the final computation complexity is of order

$$O(1^2 + 2^2 + 3^2 + \dots (n-1)^2) = O\left(\frac{1}{6}n^3\right)\,.$$

1.1 Properties of LR decompositions

In this lecture, we briefly discuss two properties of the LR decomposition. As mentioned before $A = LR$ is called the LR decomposition of A. In the following, we will see that this decomposition is unique. Moreover, we will discuss how to compute the determinant of A.

1. In order to prove that the LR decomposition of a matrix A is unique, we assume that we have two decompositions of the form

   $$A = L_1 R_1 \qquad \text{and} \qquad A = L_2 R_2$$

   with L_1 and L_2 being lower triangular with ones on the diagonal and R_1 and R_2 being upper triangular. This implies that

   $$L_1 R_1 = A = L_2 R_2$$

   which implies that

   $$R_1 R_2^{-1} = L_1^{-1} L_2 \ .$$

   Now, we see that the matrix $L_1^{-1} L_2$ is lower triangular and has ones on the diagonal while $R_1 R_2^{-1}$ is an upper triangular matrix. But these matrices can only be equal if they are both equal to unit matrix. Thus, we find that

   $$R_1 R_2^{-1} = I \qquad \text{and} \qquad L_1^{-1} L_2 = I$$

   which yields $R_1 = R_2$ and $L_1 = L_2$. Thus, the LR decomposition of a matrix A is unique.

2. In some applications it is necessary to compute the determinant of a given matrix A. One way to compute this determinant is by using the formula

   $$\det(A) = \det(LR) = \det(L)\det(R) = \det(R) = \prod_{i=1}^{n} R_{i,i} \ .$$

   This is very elegant formula for computing the determinant in practice. Although in many applications in which R is positive, it is better to work out the logarithm of the determinant

   $$\log(\det(A)) = \sum_{i=1}^{n} \log(R_{i,i}) \ .$$

   This formula is better conditioned in case A is postive definite.

1.2 Practical Applications of LR Decomposition

In practice we often don't want to solve only one linear equation, but a sequence of linear equations. If the matrix A in all of these equations is the same, this means that we want to solve the equations

$$Ax_1 = b_1 \quad , \quad Ax_2 = b_2 \quad , \quad Ax_3 = b_3 \quad , \ldots , \quad Ax_N = b_N$$

with incoming residual vectors $b_1, b_2, b_3 \ldots, b_N \in \mathbb{R}^n$. We want to compute the solutions $x_1, x_2, x_3, \ldots, x_N \in \mathbb{R}^n$ efficiently. Here, we assume that $A \in \mathbb{R}^{n \times n}$ is a given invertible matrix.

There is two ways to implement this. One "good way" and one "bad way". Let us start with the "bad implementation".

1.2.1 How to not implement LR decomposition

- Implement a function `solve(A,b)` by using LR decomposition:
  - `function solve(A,b)`
  - `(L,R) = lrdecomp(A)`
  - `return backwardSubstitute(R,forwardSubstitute(L,b))`
  - end
- Use the function solve and call it in a loop
  - `for i=1:N`
  - `x[i] = solve(A,b[i])`
  - `return x`

Notice that this implementation is **not efficient**, since the complexity of this code would be of order

$$O(Nn^3),$$

since the complexity of the LR decomposition is of order $O(n^3)$ and we do this N times.

1.2.2 How can we fix this?

The main observation here is that we need to pre-compute L and R as these matrices only depend on A. This only needs to be done once!

- `(L,R) = lrdecomp(A)`
- `for i=1:N`
- `x[i] = backwardSubstitute(R,forwardSubstitute(L,b[i])`
- `end`
- `return x`

The advantage of this implementation is that the overall complexity of this algorithm is given by

$$O(n^3 + Nn^2)$$

Notice that there is no product of N and n^3 in this complexity estimate!