1 February 4th, 2021

1.1 Efficient Gaussian Elimination

Last time, we introduced a memory efficient way to perform Gaussian Elimination by reusing the inputs. However, this implementation is **not** the most time efficient. To do this, we need to consider the computer architecture. Recall that computers have a few different layers of memory:

- CPU register
- Cache
- RAM
- Disk

The lower the level, the larger the memory but communicating is slower. As such, to improve the efficiency, we should minimize the communication between the different levels of memory.

> **Remark 1.1 —** This principle of memory management should always be considered when performing matrix multiplication.

Note that we can reformulate the previous algorithms into matrix operations. First we perform a vector scaling, then we perform a rank 1 update, and then a matrix vector product. Since these are very basic operations, there are many very efficient implementations of these **Basic Linear Algebra Subroutines (BLAS)**. There are a few different levels of these BLAS.

BLAS1 : Vector operations, e.g.

$$\beta = x^T y, y = \beta x + y, \quad \text{where } x, y \in \mathbb{R}^n, !\beta \in \mathbb{R}$$

BLAS2 : Matrix-Vector operations

$$y = Ax + y, \quad \text{where } A \in \mathbb{R}^{m \times n},\ x, y \text{ are vectors}$$

BLAS3 : Matrix-Matrix operations

$$C = AB + C, \quad \text{where } A, B, C \text{ are matricies}$$

These BLAS are implemented to be optimized on the given computer architecture.

> **Remark 1.2 —** Since BLAS3 can be constructed using BLAS2, etc. in terms of the efficiency, we have:
>
> $$\text{BLAS3} > \text{BLAS2} > \text{BLAS1}$$
>
> where efficiency is the time needed

Remark 1.3 — BLAS will come with the CPU.

AMD Core Math Library (ACML)

Intel Math Kernel Lib (MKL)

Replacing the algorithm with BLAS, we have:

Algorithm 1 Gaussian Elimination with BLAS

for $k = 1 : n-1$ **do**
  $A(k+1:n,k) = A(k+1:n,k)/A(k,k)$
  $A(k+1:n,k+1:n) = A(k+1:n,k+1:n) - A(k+1:n,k)A(k,k+1:n)$
  $b(k+1:n,1) = b(k+1:n,1) - b(k.1)A(k+1:n,k)$
end for
for $k = n : -1 : 1$ **do**
  $b(k) = b(k) - A(k,k+1:n)b(k+1:n)$
  $b(k) = b(k)/A(k,k)$
end for

1.2 Complexity of Gaussian Elimination

For this,we want to see how many scalar operations are needed. Counting the cost, we have:

$$\sum_{k=1}^{n-1}\left[(n-k)+2(n-k)^2+2(n-k)\right]+\sum_{k=1}^{n}\left[2(n-k)+1\right]$$

$$\begin{aligned}
&\sum_{k=1}^{n-1}\left[(n-k)+2(n-k)^2+2(n-k)\right]+\sum_{k=1}^{n}\left[2(n-k)+1\right]\\
&=3\sum_{k=1}^{n-1}(n-k)+2\sum_{k=1}^{n-1}(n-k)^2+2\sum_{k=1}^{n-1}k+\sum_{k=1}^{n}1\\
&=\frac{2}{3}n^3+\frac{3}{2}n^2-\frac{1}{3}n\\
&=\frac{2}{3}n^3+O(n^2)\\
&=O(n^3).
\end{aligned}$$

Theorem 1.4
The computational complexity of Gaussian Elimination is $O(n^3)$.

1.3 LU Decomposition view of Gaussian Elimination

Remember we can express Gaussian Elimination as:

$$L_{n-1}\dots L_2L_1Ax = L_{n-1}\dots L_1b$$

Note that $L_{n-1}\dots L_2L_1A = A^{(n)}$ which is upper triangular. Let us denote this by:

$$U = \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ & u_{22} & \dots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{bmatrix} \in \mathbb{R}^{n\times n}$$

Note that since each L_i is a Gaussian transformation which is invertable, we have:

$$A = L_1^{-1}L_2^{-1}\dots L_{n-1}^{-1}U$$

Theorem 1.5

$$L_k^{-1} = I + \ell_k e_k^T$$

Proof.

$$\begin{aligned}&(I - \ell_k e_k^T)(I + \ell_k e_k^T)\\ &= I - \ell_k e_k^T + \ell_k e_k^T - \ell_k e_k^T \ell_k e_k^T\\ &= I.\end{aligned}$$

□

Theorem 1.6

$$L_1^{-1}L_2^{-1}\dots L_{n-1}^{-1} = I + \sum_{i=1}^{n-1} \ell_i e_i^T$$

which is lower triangular.

Proof. Direct computation. □

Theorem 1.7

Gaussian Elimination gives:

$$A = LU$$

where L is unit lower triangular, and U is upper triangular. This is called the **LU Decomposition**.

Remark 1.8 — Note that in the memory efficient implementation of GE, we are computing L and U and it is stored in the original matrix.

With this, we have:

$$Ax = b \iff LUx = b \iff \begin{cases} Ly = b \\ Ux = y \end{cases}$$

Theorem 1.9

The computation cost is:
$$\begin{cases} \text{LU decomposition} & \frac{2}{3}n^3 + O(n^2) \\ \text{Forward Substitution} & O(n^2) \\ \text{Backward Substitution} & O(n^2) \end{cases}$$

Remark 1.10 — If we use the same coefficient matrix, we only need to compute the decomposition once.

This LU decomposition view of Gaussian Elimination has a number of advantages:

1. If we want to solve $Ax_i = b_i$ for $i = 1, \dots m$, then we only need to do LU decomposition once
2. LU decomposition can be used for other matrix computation tasks. e.g.
   (a) $\det(A) = \det(LU) = \det(L)\det(U) = u_{11}u_{22}\dots u_{nn}$
   (b) $A^{-1} = U^{-1}L^{-1}$