Mathematics for Computer Science
Linear Algebra

Lecture 5: Comparison of algorithms for solving linear systems

Andrei Krokhin

November 2, 2020

Flops

- The amount of work done by an algorithm can be measured in several ways.
- Today we will use the measure often used in numerical algorithms: flops.
- Flop = Floating-point operation, one of the 4 arithmetic operations $(+, -, \times, \div)$ performed on two real numbers in a computer.
- Thus, we will ignore the work done, e.g. moving numbers around.

The currently fastest supercomputer Fugaku, at the RIKEN Center for Computational Science in Kobe, Japan, can achieve the speed of 415.5 petaFLOPS (415.5×10^{15} flops/second)

Contents for today's lecture

Assume we want to solve a linear system $A\mathbf{x} = \mathbf{b}$ where A has size $n \times n$.

Which algorithm should we use:

- Gauss-Jordan elimination or LU-decomposition?
- How do they compare cost-wise (in flops)?
- What are the important differences?

Gauss-Jordan elimination

Input: augmented matrix $[A|\mathbf{b}]$ of size $n \times (n+1)$.

Forward phase (Gaussian elimination):

Step 1. Locate the pivot column – leftmost column that contains a non-zero.

Step 2. Choose a pivot - a non-zero in the pivot column and interchange the first row with another row (if necessary) to move the pivot to the top in this column

Step 3. If a is the pivot, multiply the first row by $1/a$ (to get a leading 1).

Step 4. Add suitable multiples of the first row to the rows below so that all numbers below the leading 1 are 0s.

Step 5. Now separate the top row from the rest ("draw a line below it") and repeat Steps 1–5 for the matrix below the line.

Backward phase (or back substitution):

Step 6. Beginning from the last non-0 row and working upward, add suitable multiples of each row to create 0s above the leading 1s.

Cost of forward phase

Forward phase proceeds in n iterations, each iteration sorting out one column.

Steps 1, 2 and 5 involve no flops, so need to analyse only Steps 3 and 4.

At each step, we will use this notation:

- the symbol $*$ denotes entries that need to be computed at this step
- the symbol $\bullet$ denotes those that don't need to be computed

Forward phase: Column 1, Step 3

Step 3. If a is the pivot, multiply the first row by $1/a$ (to get a leading 1).

Recall: $*$ = need to compute; $\bullet$ = no need to compute

$$\left(\begin{array}{cccccc|c} 1 & * & * & \cdots & * & * & * \\ \bullet & \bullet & \bullet & \cdots & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \cdots & \bullet & \bullet & \bullet \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ \bullet & \bullet & \bullet & \cdots & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \cdots & \bullet & \bullet & \bullet \end{array}\right)$$

Table: Number of flops used

	Column 1	Column 2	$\cdots$	Column n
Step 3	n		$\cdots$	
Step 4			$\cdots$	
Sum			$\cdots$	

Forward phase: Column 1, Step 4

Step 4. Add suitable multiples of the first row to the rows below so that all numbers below the leading 1 are 0s.

$$\left(\begin{array}{c|ccccc} 1 & \bullet & \bullet & \cdots & \bullet & \bullet & \bullet \\ 0 & * & * & \cdots & * & * & * \\ 0 & * & * & \cdots & * & * & * \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & * & * & \cdots & * & * & * \\ 0 & * & * & \cdots & * & * & * \end{array}\right)$$

Need 2 flops to compute each $*$

	Column 1	Column 2	$\cdots$	Column n
Step 3	n		$\cdots$	
Step 4	$2n(n-1)$		$\cdots$	
Sum	$2n^2 - n$		$\cdots$	

Forward phase: Column 2, Step 3

Step 3. If a is the pivot, multiply the first row by $1/a$ (to get a leading 1).

$$\left(\begin{array}{c|cccccc} 1 & \bullet & \bullet & \cdots & \bullet & \bullet & \bullet \\ \hline 0 & 1 & * & \cdots & * & * & * \\ 0 & \bullet & \bullet & \cdots & \bullet & \bullet & \bullet \\ & & & & & & \\ 0 & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & \bullet & \bullet & \cdots & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \cdots & \bullet & \bullet & \bullet \end{array}\right)$$

	Column 1	Column 2	$\cdots$	Column n
Step 3	n	$n-1$	$\cdots$	
Step 4	$2n(n-1)$		$\cdots$	
Sum	$2n^2-n$		$\cdots$	

Forward phase: Column 2, Step 4

$$\left(\begin{array}{c|ccccc|c} 1 & \bullet & \bullet & \cdots & \bullet & \bullet & \bullet \\ \hline 0 & 1 & \bullet & \cdots & \bullet & \bullet & \bullet \\ 0 & 0 & * & \cdots & * & * & * \\ 0 & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & * & \cdots & * & * & * \\ 0 & 0 & * & \cdots & * & * & * \end{array}\right)$$

	Column 1	Column 2	$\cdots$	Column n
Step 3	n	$n-1$	$\cdots$	
Step 4	$2n(n-1)$	$2(n-1)(n-2)$	$\cdots$	
Sum	$2n^2-n$	$2(n-1)^2-(n-1)$	$\cdots$	

Forward phase: Column n, Steps 3 and 4

$$\left(\begin{array}{cccccc|c} 1 & \bullet & \bullet & \cdots & \bullet & \bullet & \bullet \\ 0 & 1 & \bullet & \cdots & \bullet & \bullet & \bullet \\ 0 & 0 & 1 & \cdots & \bullet & \bullet & \bullet \\ & & & & & & \\ 0 & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & \bullet & \bullet \\ \hline 0 & 0 & 0 & \cdots & 0 & 1 & * \end{array}\right)$$

	Column 1	Column 2	$\cdots$	Column n
Step 3	n	$n-1$	$\cdots$	1
Step 4	$2n(n-1)$	$2(n-1)(n-2)$	$\cdots$	0
Sum	$2n^2-n$	$2(n-1)^2-(n-1)$	$\cdots$	1

Forward phase: Summing up

	Column 1	Column 2	$\cdots$	Column n
Step 3	n	$n-1$	$\cdots$	1
Step 4	$2n(n-1)$	$2(n-1)(n-2)$	$\cdots$	0
Sum	$2n^2-n$	$2(n-1)^2-(n-1)$	$\cdots$	1

$$[2n^2-n]+[2(n-1)^2-(n-1)]+[2(n-2)^2-(n-2)]+\ldots+[2(1)-1]=$$

$$2(n^2+(n-1)^2+(n-2)^2+\ldots+1)-(n+(n-1)+(n-2)+\ldots+1)=$$

$$2\frac{n(n+1)(2n+1)}{6}-\frac{n(n+1)}{2}=\frac{2}{3}n^3+\frac{1}{2}n^2-\frac{1}{6}n.$$

You proved in the practical that the blue expressions are equal. One can easily prove by induction that the red expressions are also equal.

For large n, the first term dominates the other, so the number of flops is $\approx \frac{2}{3}n^3$.

Backward phase: Columns n and $n-1$

Step 6. Beginning from the last non-0 row and working upward, add suitable multiples of each row to create 0s above the leading 1s.

$$\left(\begin{array}{ccccccc|c} 1 & \bullet & \bullet & \cdots & \bullet & \bullet & 0 & * \\ 0 & 1 & \bullet & \cdots & \bullet & \bullet & 0 & * \\ 0 & 0 & 1 & \cdots & \bullet & \bullet & 0 & * \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & \bullet & 0 & * \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 & * \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \bullet \end{array}\right) \qquad \left(\begin{array}{ccccccc|c} 1 & \bullet & \bullet & \cdots & \bullet & 0 & 0 & * \\ 0 & 1 & \bullet & \cdots & \bullet & 0 & 0 & * \\ 0 & 0 & 1 & \cdots & \bullet & 0 & 0 & * \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 & * \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \bullet \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \bullet \end{array}\right)$$

Need 2 flops to compute each $*$

The pattern is now clear: the total number of flops is

$$2(n-1) + 2(n-2) + \ldots + 2(n-n) = 2\frac{n(n-1)}{2} = n^2 - n \approx n^2.$$

Cost comparison table

Approximate cost for an $n \times n$ matrix A with large n	
Algorithm	Cost in flops
Gauss-Jordan (forward phase)	$\approx \frac{2}{3}n^3$
Gauss-Jordan (backward phase)	$\approx n^2$
Computing LU-decomposition of A	$\approx \frac{2}{3}n^3$
Forward substitution (to solve $L\mathbf{y} = \mathbf{b}$)	$\approx n^2$
Backward substitution (to solve $U\mathbf{x} = \mathbf{y}$)	$\approx n^2$
Finding A^{-1} by inversion algorithm	$\approx 2n^3$
Computing $A^{-1}\mathbf{b}$	$\approx 2n^3$

Comments on the table

- Computing LU-decomposition involves the same process as G-J forward phase: Gaussian elimination
  - except on $n \times n$ matrix A, rather than on the augmented matrix of size $n \times (n+1)$, but the cost is practically the same
- Backward substitution (to solve $U\mathbf{x} = \mathbf{y}$) is essentially the same process as G-J backward phase
- Cost-wise, forward substitution (to solve $L\mathbf{y} = \mathbf{b}$) is the same as backward
- The estimate for finding A^{-1} by inversion algorithm - in the practical

- The LU method and G-J elimination have rouhgly the same cost in flops.
- Do you see any advantage(s) of using LU?

Solving matrix equations $AX = B$

Assume that matrices A, X and B have sizes $n \times n$, $n \times k$ and $n \times k$, respectively:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1k} \\ x_{21} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nk} \end{pmatrix} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1k} \\ b_{21} & b_{22} & \cdots & b_{2k} \\ \vdots & \vdots & & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nk} \end{pmatrix}$$

- Think of X as consisting of its columns: k matrices $\mathbf{x}_1, \ldots, \mathbf{x}_k$ of size $n \times 1$.
- Think of B as consisting of its columns: k matrices $\mathbf{b}_1, \ldots, \mathbf{b}_k$ of size $n \times 1$.
- Observe: Solving $AX = B$ is the same as solving $A\mathbf{x}_1 = \mathbf{b}_1, \ldots, A\mathbf{x}_k = \mathbf{b}_k$.

- Once we know an LU-decomposition of A, we can use it to solve each system $A\mathbf{x}_i = \mathbf{b}_i$ by substitution. LU-decomposition is reusable!
- If we use Gauss-Jordan elimination, we would have to run it from scratch for each system.

Inverting matrices from LU-decomposition

Observe: inverting A is the same as solving the matrix equation $AX = I$.

As discussed above, solving this equation

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

is the same as solving

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{21} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ and } \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_{12} \\ x_{22} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

If we already know an LU-decomposition $A = LU$, we can easily solve systems such as above by using only forward/backward substitution.

What we learnt today

LU-decomposition vs. Guass-Jordan elimination:

- The two methods have the same cost in flops ($\approx \frac{2}{3}n^3$ for $n \times n$ matrices)
  - Neither has a cost advantage if we need to solve a single system
- LU is reusable: much better when solving many systems with the same A
  - Many applications in Scientific Computing are exactly like that
- LU-decomposition can be used to invert matrices

Next time:

- Determinants