Numerical Linear Algebra – Matrix interpretation

CS370 Lecture 31 – March 29, 2017

Cost of Factorization

$$
\begin{array}{l}
\text{For } k = 1, .., n \\
\quad \text{For } i = k+1, ..., n \\
\quad \quad mult := a_{ik}/a_{kk} \\
\quad \quad a_{ik} := mult \\
\quad \quad \text{For } j = k+1, ..., n \\
\quad \quad \quad a_{ij} := a_{ij} - mult * a_{kj} \\
\quad \quad \text{EndFor} \\
\quad \text{EndFor} \\
\text{EndFor}
\end{array}
$$

2 FLOPs (1 subtraction, 1 multiply) in the innermost loop.

Summing over all the loops we get:

$$\sum_{k=1}^{n} \sum_{i=k+1}^{n} \sum_{j=k+1}^{n} 2 = \frac{2n^3}{3} + O(n^2)$$

The above requires using the following sum identities…

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

and

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Cost of Triangular solve

Show that the *total* FLOP count of backward substitution is: $n^2 + O(n)$ FLOPs.

$$
\begin{array}{l}
\text{For } i = n, \ldots, 1 \\
\qquad x_i := z_i \\
\qquad \text{For } j = i+1, \ldots, n \\
\qquad\qquad x_i := x_i - u_{ij} * x_j \\
\qquad \text{EndFor} \\
\qquad x_i := x_i / u_{ii} \\
\text{EndFor}
\end{array}
$$

We previously used these identities:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

and

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Costs for Solving Linear Systems

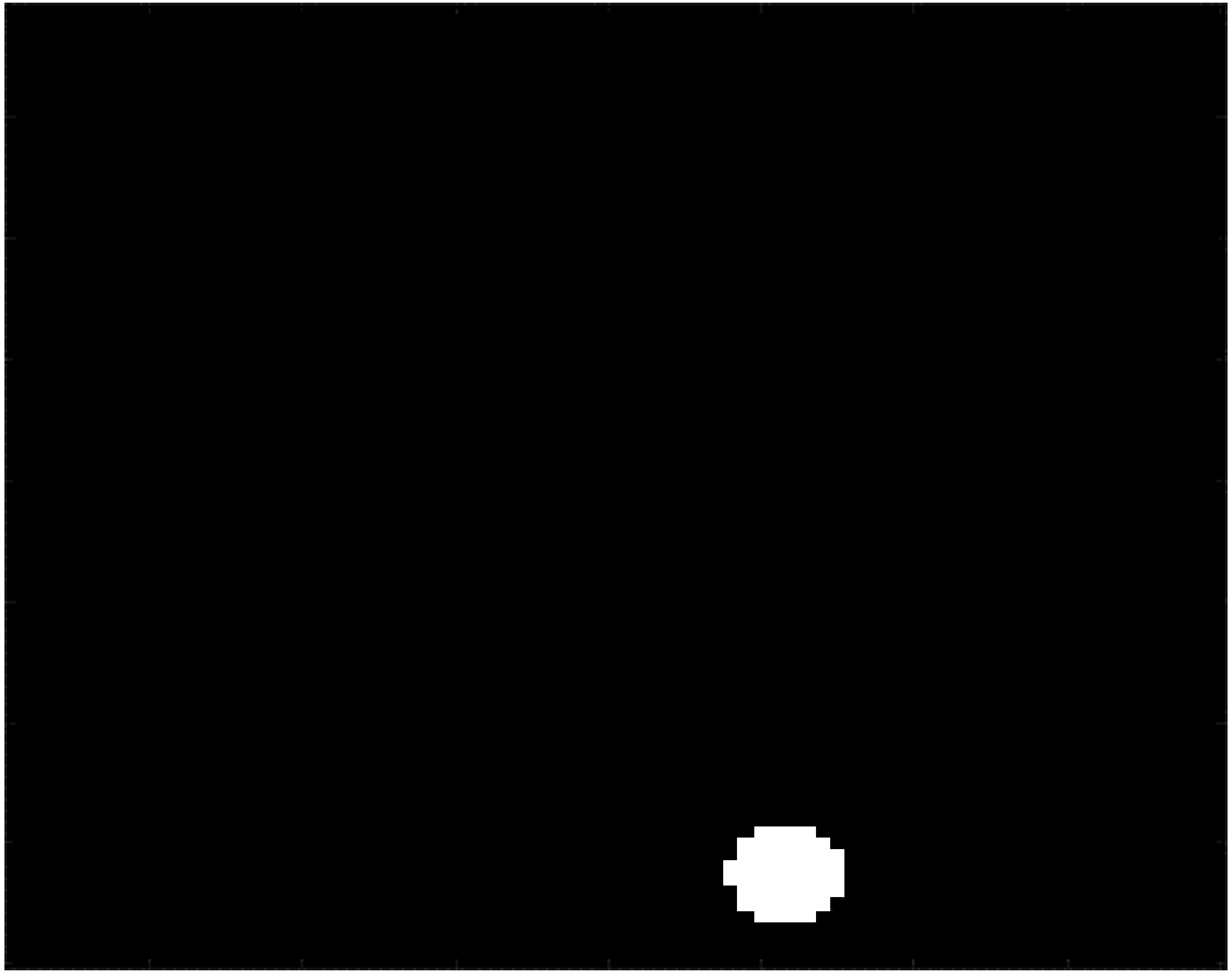
LU factorization costs approximately $\boxed{\frac{2n^3}{3} + O(n^2)}$ FLOPs.

Triangular solves cost $n^2 + O(n)$ FLOPs *each*, so total is $\boxed{2n^2 + O(n).}$

Factorization cost dominates when n is large, and scales worse.

Given an existing factorization, solving for new RHS's is cheaper: $O(n^2)$.

Cost Example: Simple 2D smoke simulation



Compare speed between:

(1) Doing full LU factorization and forward/backward solve each step.

(2) Factoring the matrix once at the start, and reusing the L & U factors.

As expected from the FLOP counts, (2) is dramatically faster.

Justifying the Factorization View of G.E.

We viewed row-swapping as a matrix; do the same for row-subtraction!

Zeroing a (sub-diagonal) entry of a column by row subtraction can be written as applying a specific matrix M such that

$$MA^{old} = A^{new}$$

where

- A^{old} is the original matrix.
- A^{new} is the matrix *after* subtracting the specific row.

Row Subtraction via Matrices

e.g. The operation…

(2nd row) := (2nd row) $-\frac{a_{2,1}}{a_{1,1}}$ (1st row)

can be written as a matrix:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{a_{2,1}}{a_{1,1}} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

M is the identity matrix, but with a zero entry replaced by the (negative of the) necessary multiplicative factor.

Row Subtraction via Matrices

Example:

$$\begin{array}{ccccc} M & & A^{old} & & A^{new} \\ \begin{bmatrix} 1 & 0 & 0 \\ -3/2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & & \begin{bmatrix} 2 & 3 & 4 \\ 3 & 4 & -1 \\ -2 & 2 & 1 \end{bmatrix} & = & \begin{bmatrix} 2 & 3 & 4 \\ 0 & -1/2 & -7 \\ -2 & 2 & 1 \end{bmatrix} \end{array}$$

So, the whole process of factorization can be viewed as a sequence of matrix (left-)multiplications applied to A.

The matrix left at the end is U, so e.g., in 3x3 case, we have shown:

$$M^{(3)}M^{(2)}M^{(1)}A = U$$

Row Subtraction via Matrices

The matrix left at the end is U, so e.g., in 3x3 case, we have shown:

$$M^{(3)}M^{(2)}M^{(1)}A = U$$

Therefore $A = \left(M^{(3)}M^{(2)}M^{(1)}\right)^{-1}U = \underbrace{\left(M^{(1)}\right)^{-1}\left(M^{(2)}\right)^{-1}\left(M^{(3)}\right)^{-1}}_{L}U$.

Define $L = \left(M^{(1)}\right)^{-1}\left(M^{(2)}\right)^{-1}\left(M^{(3)}\right)^{-1}$ and we have our factorization!

But what is $\left(M^{(k)}\right)^{-1}$?

Inverse of $M^{(i)}$

The inverse of this simple matrix form is the same matrix, but with the off-diagonal entry ***negated***.

e.g.

$$\begin{bmatrix} 1 & 0 & 0 \\ -3/2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 3/2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

You can easily verify that:

$$\begin{bmatrix} 1 & 0 & 0 \\ -3/2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 3/2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This is why we stored $a_{i,k}/a_{k,k}$ (not $-a_{i,k}/a_{k,k}$) during row subtraction.

Summary: Row Subtraction via Matrices

The effect of row reduction is to left-multiply A by a series of matrices $M^{(k)}$ that comprise L^{-1} to get U, recording the entries of L as we go.

$$M^{(3)}M^{(2)}M^{(1)}A = L^{-1}A = U$$

Interleaving permutation matrices $P^{(k)}$ before each $M^{(k)}$ similarly leads to the $PA = LU$ factorization (but it's a bit trickier to see this.)

*Note: The course notes combine multiple row subtraction operations for a given column into a single matrix operation M; the net result is the same.

Costs: Solving $Ax = b$ by Matrix Inversion.

An obvious alternative for solving $Ax = b$ is:

1. Invert A to get A^{-1}.
2. Multiply $A^{-1}b$ to get x.

One can show that the above is actually ***more*** expensive (in FLOPs) than using our "factor and triangular solve" strategy.

It also generally incurs more F.P. error.

Most numerical algorithms avoid ever computing A^{-1}.

Summary: Gaussian elimination

Linear systems like $Ax = b$ can be efficiently solved by:

1. LU factorization of A, followed by…
2. Forward/backward substitution.

Adding row pivoting avoids div-by-zero and reduces floating point error.

Total cost is $\frac{2\mathrm{n}^3}{3} + \mathrm{O}(n^2) = O(n^3)$ FLOPs, dominated by the factorization.

If RHS changes, the factorization can be reused at lower cost, $O(n^2)$.

Final topic: Norms and Conditioning

+

Norms and Conditioning

Norms are measurements of “size”/magnitude for vectors or matrices.

We will start by defining some norms.

Then, we will use these norms to explore **conditioning** of matrices.

Conditioning describes how the output of a function/operation/matrix changes due to changes in input.

Vector Norms

There are many reasonable norms for a vector $\boldsymbol{x} = [x_1, x_2, \ldots x_n]^T$.

Common choices are:

1-norm (or taxicab/manhattan norm): $||\boldsymbol{x}||_1 = \sum_{i=1}^{n} |x_i|$

2-norm (or Euclidean norm): $||\boldsymbol{x}||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$

∞-norm (or max norm): $||\boldsymbol{x}||_\infty = \max_i |x_i|$

$\boldsymbol{p}$-norms

These are collectively called p-norms, for $p = 1, 2, \infty$ and can be written:

$$||\boldsymbol{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}$$

(Only holds *in the limit* for ∞ case).

Matlab has a **norm()** command that implements these vector norms.

Some Properties of Norms

If the norm is zero, then the vector must be the zero-vector.

$$||\boldsymbol{x}|| = 0 \ \rightarrow \ x_i = 0 \ \forall i.$$

The norm of a scaled vector must satisfy

$$||\alpha\boldsymbol{x}|| = |\alpha| \cdot ||\boldsymbol{x}|| \quad \text{for scalar } \alpha.$$

The triangle inequality holds:

$$||\boldsymbol{x} + \boldsymbol{y}|| \leq ||\boldsymbol{x}|| + ||\boldsymbol{y}||$$

Defining Norms for Matrices

Matrix norms are often defined/“induced” as follows, *using* p-norms of vectors:

$$||A|| = \max_{||x|| \neq 0} \frac{||A\boldsymbol{x}||}{||\boldsymbol{x}||}$$

A bit tricky… clearly we can’t try out all possible $\boldsymbol{x}$ to determine this!

But, there are simpler equivalent definitions in some cases:

$$||A||_1 = \max_j \sum_{i=1}^{n} |A_{ij}|$$

(max absolute column sum)

$$||A||_\infty = \max_i \sum_{j=1}^{n} |A_{ij}|$$

(max absolute row sum)

Matrix 2-norm (or *spectral* norm)

Using the vector 2-norm, we get the matrix 2-norm, or spectral norm.

$$||A||_2 \ = \max_{||x|| \neq 0} \frac{||A\boldsymbol{x}||_2}{||\boldsymbol{x}||_2},$$

The matrix's 2-norm relates to the eigenvalues.

Specifically, if λ_i are the eigenvalues of $A^T A$, then

$$||A||_2 = \max_i \sqrt{|\lambda_i|}$$

Some Matrix Norm Properties

$$||A|| = 0 \;\leftrightarrow\; A_{ij} = 0 \;\forall i, j.$$

$$||\alpha A|| = |\alpha| \cdot ||A|| \text{ for scalar } \alpha$$

$$||A + B|| \leq ||A|| + ||B||$$

$$||A\boldsymbol{x}|| \leq ||A|| \cdot ||\boldsymbol{x}||$$

$$||AB|| \leq ||A|| \cdot ||B||$$

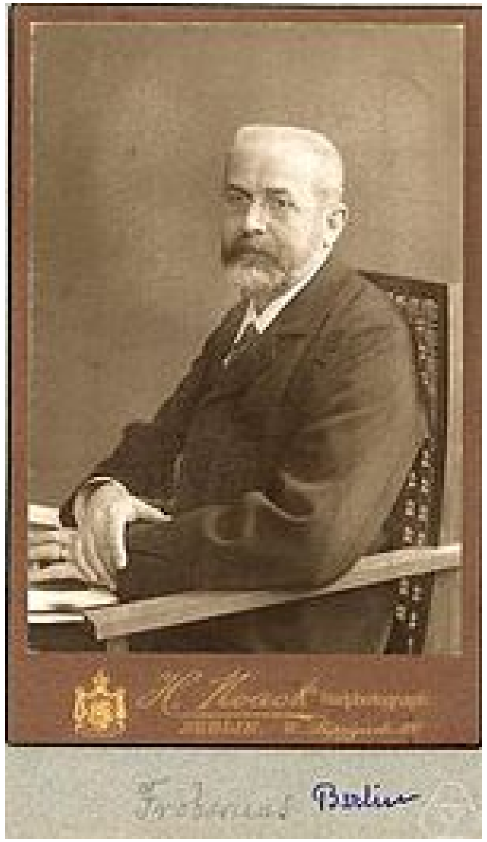
$$||I|| = 1$$

Bonus: Frobenius Norm

Another fun matrix norm that is not "induced" by a standard vector norm is the *Frobenius* norm.

$$||A||_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij}^2}$$

(Like a "2D" version of the 2-norm for vectors.)



Dr. Frobenius

Next Time: Conditioning of Linear Systems

Conditioning describes how the output of a function/operation/matrix changes due to changes in input.

Conditioning is indicative of how difficult a problem is to solve, *independent* of the algorithm / numerical method used.

Norms are a useful tool to help characterize the conditioning of linear systems.