Numerical Linear Algebra – Pivoting and FLOP counting

CS370 Lecture 30 – March 27, 2017

Gaussian Elimination: $Ax = b$

(Some) numerical algorithms use Gaussian elimination, too. But it is interpreted differently…

Our view will be the following:

1. **Factor** matrix A into $A = LU$, where L and U are *triangular*.
2. **Solve** $Lz = b$ for intermediate vector z.
3. **Solve** $Ux = z$ for x.

(Later: We may also want to reorder (*permute*) the equations, which leads to the factorization $PA = LU$.)

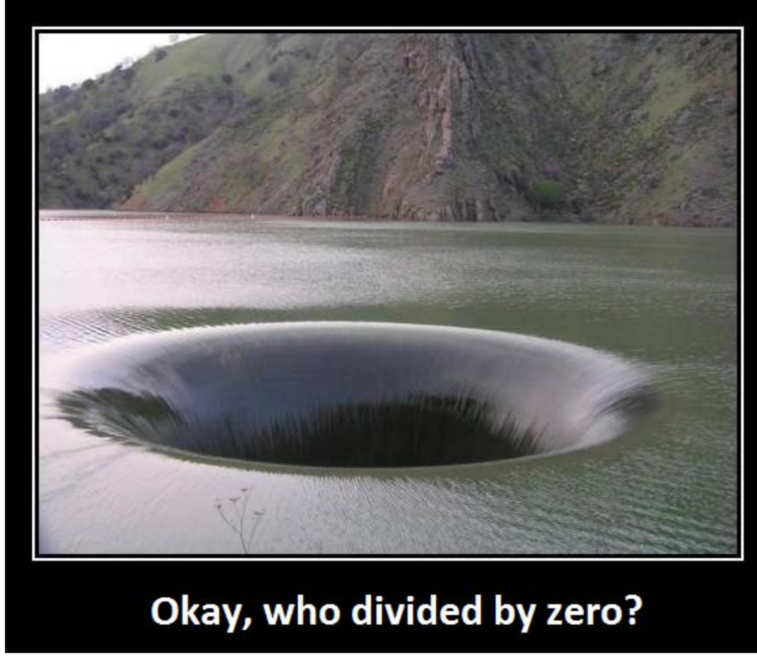
Numerical Problems – Factorization

During factoring, what if a diagonal entry, $a_{k,k}$ is zero? Or close to zero?

For $k = 1, .., n$
    For $i = k + 1, ..., n$
        $mult := a_{ik}/a_{kk}$
        $a_{ik} := mult$
        For $j = k + 1, ..., n$
            $a_{ij} := a_{ij} - mult * a_{kj}$
        EndFor
    EndFor
EndFor

Divide by (exactly or nearly) zero?

Numerical Problems



A division by zero is obviously bad news.

What about nearly zero?

This will make the multiplicative factor, $a_{i,k}/a_{k,k}$, **large** in magnitude.

A large factor can cause large floating point error during subtraction and magnify existing floating point error.

Leads to numerical instability!

Solution: Row/Partial Pivoting

In earlier classes, you likely swapped rows if a diagonal entry was zero.

This is called “row pivoting” or “partial pivoting”.

For numerical algorithms, one difference: we will **always** swap, to minimize floating point error incurred.

Strategy: Find the row with the *largest magnitude entry* in the current column beneath the current row, and swap those rows if larger than the current entry.

Row/Partial Pivoting

Strategy: Find the row with the *largest magnitude entry* in the current column beneath the current row, and swap those rows if larger than the current entry.

e.g. $\begin{bmatrix} 0 & 2 & 3 \\ 1 & 4 & 2 \\ -3 & 2 & -1 \end{bmatrix}$

We immediately swap the first and third rows, since $|-3| > |1| > 0$.

Pivoting with a Permutation Matrix

How can our factorization view account for row swaps?

Find a modified factorization of A such that $PA = LU$ where P is a ***permutation matrix***.

A permutation matrix P is a matrix whose effect is to swap rows of the matrix it is applied to (i.e., multiplied with).

Permutation Matrix

P is simply a permuted (row-swapped) version of identity matrix, I.

e.g. to swap rows 2 and 3 of a 3x3 matrix, swap rows 2 and 3 of I.

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Multiplying with a matrix performs the same swap:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 3 & 1 \\ 1 & 2 & 2 \\ 5 & 4 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 \\ 5 & 4 & -1 \\ 1 & 2 & 2 \end{bmatrix}$$

Solving with the new factorization

Given the factorization $PA = LU$, multiplying $Ax = b$ by P shows that $PAx = Pb$.

Using our factorization to replace PA, we have $LUx = Pb$.

Solve by first permuting entries of b according to P.
Then forward and backward substitution lets us find x as usual.

How do we determine P during factorization?

Finding the permutation matrix, P

Start with P set to be an $n \times n$ identity matrix, I.

Whenever we swap a pair of rows during LU factorization, also swap the corresponding rows of P (*including* the already stored factors).

The final *P* will be the desired permutation matrix.

Example with pivoting

Let’s try an example!

$$\begin{bmatrix} 1 & 4 & 5 \\ -2 & 3 & 3 \\ 3 & 0 & 6 \end{bmatrix} x = \begin{bmatrix} 4 \\ 1 \\ -3 \end{bmatrix}$$

Summary:
Gaussian Elimination with Row Pivoting

1. Perform LU factorization on A to find $P, L,$ and U.
2. Solve $Lz = Pb$ for z. (Forward Solve.)
3. Solve $Ux = z$ for x. (Backward Solve.)

Matlab's built-in LU factorization is the command:
[L,U,P] = lu(A);

Costs of Gaussian Elimination

We want to know the (asymptotic) cost to solve a system of size $n \times n$.

We will measure cost in total *FLOPs: FLoating point OPerations.*
Approximate as the number of: *adds+subtracts+multiplies+divides*.

A true operation count would be hardware-dependent.

- e.g. Fused-multiply-add (FMA) may be a single operation.

(Careful: FLOPS is also floating-point-operations-*per-second*.)

Cost of Factorization

Summing over all the loops we get:

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

For $k = 1, .., n$
    For $i = k + 1, ..., n$
        $mult := a_{ik}/a_{kk}$
        $a_{ik} := mult$
        For $j = k + 1, ..., n$
            $a_{ij} := a_{ij} - mult * a_{kj}$
        EndFor
    EndFor
EndFor

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

The above requires using the following sum identities…

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

and

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