Math 128A, Summer 2019
Lecture 26, 8/6/2019

1 Pivoting

In real arithmetic, performing Gaussian elimination on something like

$$\begin{bmatrix} 1 & 2 \\ 3000 & 4 \end{bmatrix}$$

is perfectly fine. However, in computer arithmetic, subtracting a large number by a very small number is bad, because any error grows. Hence we want to perform **pivoting** to move rows and columns around via 'swaps'.
We want to do something like:

$$A = \begin{bmatrix} 1 & 2 \\ 3000 & 4 \end{bmatrix} \mapsto PA = \begin{bmatrix} 3000 & 4 \\ 1 & 2 \end{bmatrix},$$

where $P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. Notice $P^{-1} = P$ and $P^2 = I$. Let's check if the general case for $P^{1,i}$ is a symmetric matrix:
To swap i, j rows,

$$\begin{aligned} P^{ij} &= I - e_i e_i^T - e_j e_j^T + e_i e_j^T + e_j e_i^T \\ &= I - (e_i - e_j)(e_i - e_j)^T \end{aligned}$$

Strain reminds us to keep the multipliers ≤ 1 to not lose information we may want to keep. To achieve this, consider Gaussian Elimination with Partial Pivoting (GEPP):

Definition: Partial Pivoting -

The simplest strategy, called **partial pioting**, is to select an element in the same column that is below the diagonal and **has the largest absolute value**. Specifically, we determine the smallest $p \geq k$ with:

$$|a_{pk}^{(k)}| = \max_{k \leq i \leq n} |a_{ik}^{(k)}|$$

If $|a_{11}| = \max_{1 \leq i \leq n} |a_{i1}|$ then $m_{i1} \leq 1$ for all i.

We construct:

$$M_{n-1} P_{n-1} \cdots M_3 P_3 M_2 P_2 M_1 P_1 A = U = \begin{bmatrix} u_{11} & \cdots & u_{1n} \\ & \ddots & \vdots \\ 0 & \cdots & u_{nn} \end{bmatrix}$$

where M_i are unit lower-triangular, and P_i performs row swaps, so U is nonsingular if and only if A is nonsingular.
We claim that if all these P above are absent, then

$$M_{n-1} \cdots M_3 M_2 M_1 = \begin{bmatrix} 1 & & & 0 \\ -m_{21} & \ddots & & 0 \\ \vdots & & \ddots & 0 \\ m_{n1} & \cdots & & m_{n,n-1} \end{bmatrix}$$

If there are no Ps, then $A = LU$, where L is lower-triangular, and U is upper-triangular.
We check if $PA = LU$, then does this mean $M_1P_1 = P_1M_1$? Recall that m_i below is a vector.

$$\begin{aligned}
M_1 &= I - m_1e_1^T, \qquad e_1^Tm_1 = 0\\
P_1 &= I - (e_1 - e_j)(e_1 - e_j)^T\\
M_1P_1 &= \left(I - m_1e_1^T\right)\left((e_1 - e_j)(e_1 - e_j)^T\right)\\
&= I - (e_1 - e_j)(e_1 - e_j)^T - m_1\underbrace{\left[e_1^T(e_1 - e_j)\right]}_{1}(e_1 - e_j)^T - m_1e_1^T\\
&= I - (e_1 - e_j)(e_1 - e_j)^T + e_1e_j^T\\
P_1M_1 &= \left(I - (e_1 - e_j)(e_1 - e_j)^T\right)\left(I - m_1e_1^T\right)\\
&= I - m_1e_1^T - (e_1 - e_j)(e_1 - e_j)^T + (e_1 - e_j)(e_1 - e_j)^Tm_1e_1^T\\
&= I - (e_1 - e_j)(e_1 - e_j)^T - m_1e_1^T + (e_1 - e_j)e_j^Tm_1e_1^T
\end{aligned}$$

And we notice

$$m_1 + (e_1 - e_j)e_j^Tm_1$$

returns a vector like:

$$\begin{bmatrix} m_{j1}\\ m_{21}\\ \vdots\\ 0\\ \vdots\\ m_{n1}\end{bmatrix},$$

where we replace the first entry by a possibly nonzero entry and insert a zero into that slot.
Hence we conclude that

$$P_1M_1 = M_1P_1$$

with m rearranged. Precisely, we write:

$$\begin{aligned}
U &= M_{n-1}P_{n-1}\cdots P_1A\\
&= \tilde{M}_{n-1}\tilde{M}_{n-2}\cdots\tilde{M}_1P_{n-1}\cdots P_1A\\
PA &= LU
\end{aligned}$$

2 Algorithm Without Pivoting

First we look at an algorithm without pivoting to see what we want to change.

```
for k = 1 : n-1 % k is column and row index
  for i = k+1 : n % row i
    % compute a_ik and check it is zero (or how close, for
    debugging)
    a_ik = a_ik / a_kk
    for j = k+1 : n
    % perform rank-1 update to bottom-right sub-matrix
      a_ij = a_ij - a_ik * a_kj;
    endfor
  endfor
endfor
```

3 Algorithm WITH Pivoting

First let's be smart and not actually perform row swaps but rather store an array with which row swaps to perform.

```
p = [1:n]
for k = 1:n+1
  [amax, imax ] = max( abs( a(k+1:n , k ) ) )
  imax = imax + k % fix the index of destination of swap
  iswap = p(k)
  p(k) = imax
  p(imax) = iswap

  for i = k+1 : n
    a_{p(i), k} = a_{p(i), k} / a_{p(k), k}
    for j = k+1:n
      a_{p(i), j} = a_{p(i),j} - a_{p(i),k} * a_{p(k),j}
    endfor
  endfor
endfor
```

Fortran stores values by columns, but some programming languages stores values by rows. It's beneficial to know how your data is stored in your programming language to control your memory storage and access.
Notice that Complete Pivoting (row and column swaps) is very expensive. We also have Rook pivoting (row and column swaps but not all), or Randomized Rook pivoting (for which Professor Gu is an authority).
After the break we'll look at diagonally dominant matrices (where pivoting won't be necessary), and later we'll look at positive definite matrices.

Break time.

4 Special Matrices: (1) Diagonally Dominant, (2) Symmetric Positive Definite

We consider two classes of matrices for which Gaussian elimination can be performed effectively without row interchanges (pivoting is not only not necessary but **possibly harmful**). The first is **diagonally dominant** matrices, and the second is **positive-definite** matrices. For symmetric positive-definite matrices, we take Cholesky decomposition.

4.1 Diagonally Dominant

Definition: Diagonally Dominant -

We say an $n \times n$ matrix A is **strictly** diagonally dominant when

$$|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ij}|, \quad \forall i = 1 : n$$

.

Take for example,

$$\begin{bmatrix} 10 & 1 & -2 \\ -3 & 42 & -4 \\ 50 & -6 & 630 \end{bmatrix}$$

Consider the solution to $Ax = b$ where:

$$x_i = b_i - \sum_{j \neq i} \overbrace{\left(\frac{a_{ij}}{a_{ii}}\right)}^{<1} x_j$$

involves an effective 'average' of x_j, and hence **fixed point iteration will converge**.

Theorem 4.1. If A is Diagonally Dominant, then Gaussian elimination **without** pivoting will succeed.

Proof. According to Strain, the fundamental rule in linear algebra is to induct on the dimension. In the 1×1 case, we have $ax = b$, with $a > 0$, and hence $L = 1$ and $U = a$, as desired.

For fun, consider the 2×2 case. For $a > |b|$ and $d > |c|$, check:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \to \begin{bmatrix} a & b \\ 0 & b - \frac{1}{a}cb \end{bmatrix}$$

and we check that $a \neq 0$. We already have $a > |b|$, so we consider:

$$d - \frac{b}{a}c > |c| - |c| = 0$$

Now we check the 3×3 case before declaring victory.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \to \begin{bmatrix} a & b & c \\ 0 & e - \frac{db}{a} & f - \frac{dc}{a} \\ 0 & h - \frac{gb}{a} & i - \frac{gc}{a} \end{bmatrix}$$

We want to check that the bottom-right 2×2 sub-matrix is diagonally dominant. To do this, we have $e > |f| + |d|$ and $a > |b| + |c|$. We check:

$$e - \frac{db}{a} > f - \frac{dc}{a}$$

by seeing:

$$\begin{aligned} e &> f - \frac{d}{a}c + \frac{d}{a}b \\ &= f + d\left(\frac{b}{a} + \frac{c}{a}\right) \leq f + d \end{aligned}$$

□

4.2 Symmetric Positive Definite Matrices

Last week, we discussed this type of matrix. (1) We have $A = A^T$ and $x^T A x > 0$ unless $x = 0$. This requires us to test all possible x (which we sometimes can algebraically). This type is so important that we simply write:

$$A > 0,$$

which is **not** the same as saying all entries are positive. For example, take:

$$\begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0.$$

On the other hand, take:

$$\begin{bmatrix} x_1 x_2 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} (2x_1 - x_2) \\ (-x_1 + 3x_2) \end{bmatrix} = 2x_1^2 - 2x_1x_2 + 3x_2^2 \geq x_1^2 + 2x_2^2 > 0$$

unless $x = 0$ because $2ab \leq a^2 + b^2$ from the trivial inequality on $(a + b)^2$.
(2) Another test for symmetric positive definite matrices ($A > 0$) is that if $\det(A_{kk}) > 0$ for all leading principle submatrices.
(3) Recall that the other test for $A > 0$ is that all the eigenvalues are strictly greater than 0. That is,

$$\lambda_j(A) > 0, \quad \forall j$$

Review Recall some classes of matrices:

Symmetric: $A = A^T$ (a symmetric matrix always has real eigenvalues.)
Normal (unitarily diagonalizable): $AA^T = A^T A$
Orthogonal: $A^T A = I = AA^T$
Projection: $A^2 = A$.

Now we want to show that a symmetric positive definite matrix (SPD) implies that no pivoting is necessary. Take the 2×2 case.

Lecture ends here.

Next time we'll look at Cholesky Factorizations.