Lecture 6 Numerical Analysis 02/07/18

Last time:

- Fixed point convergence

$\Rightarrow$ if $|f'(\xi)| > 1$ at fixed point ξ, then $x_{n+1} = f(x_n)$ does not converge, ξ is unstable

- IEEE floating point arithmetic

- floating point numbers are stored using bits : 1 byte = 8 bits

DOUBLE PRECISION = 8 bytes
= 64 bits

- Machine precision is the distance between 1.0 and the next floating point number on your machine.

- DEMO to calculate ϵ_{MACH} in MATLAB.

- IEEE rules make sure that arithmetic is done to precision at least that of machine precision:

Ex: $\underbrace{a \oplus b}_{\text{addition in floating point}} = \text{round}(a+b)$

$= \underbrace{(a+b)}_{\text{true exact result}}(1+\delta)$, $|\delta| < \epsilon$.

This is a statement of relative accuracy:

$$\Rightarrow \left| \frac{\text{round}(a+b) - (a+b)}{(a+b)} \right| = |\delta| < \epsilon$$

Ex: $|\text{round}(a+b) - (a+b)| = |a+b||\delta|$

$\leq (|a|+|b|)\epsilon$

absolute accuracy $\longrightarrow$ $\leq 2 \cdot \max(|a|,|b|)\,\epsilon$.

For rules on IEEE accuracy for other operations, GOOGLE it.

Next: Numerical Linear Algebra

Basically, the only math your computer can do is linear algebra. There are very few instances when non-linear problems are solved without some element of linear algebra.

What are the standard tools needed in linear algebra?

- Products:
  - vector-vector (inner product)
  - matrix-vector
  - matrix-matrix
- Solutions:
  - solve $A\vec{x} = \vec{b}$
  - minimize $\| A\vec{x} - \vec{b} \|$ (least squares)
- Factorizations
  - $A = LU$
  - $A = USV^T$
  - $A = QR$
- Eigen computations
  - Find all $\lambda_i, \vec{v}_i$ s.t. $A\vec{v}_i = \lambda_i \vec{v}_i$.

Efficient methods for doing all these calculations are the building blocks of almost all scientific computing.

Tell Levi Strauss anecdote.

Ignore things like AB, $A\vec{x}$, $\vec{u}^T\vec{v}$ for now, these are computations that merely need to be optimized, a CS endeavor.

Note: Computational cost: $A \sim n \times n$, $\vec{u}, \vec{v} \sim n \times 1$

$$\vec{u}^T\vec{v} \sim \mathcal{O}(n) \text{ flops}$$
$$A\vec{x} \sim \mathcal{O}(n^2) \text{ flops}$$
$$A^TA \sim \mathcal{O}(n^3) \text{ flops}$$

Consequence for larger computers:

A computer with twice the speed/storage can only compute A^TA with $A \sim \sqrt[3]{2}n$ in the same time as original machine.

First problem to tackle: solve $A\vec{x} = \vec{b}$ using Gaussian elimination

Recall: Let $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$

$A\vec{x} = \vec{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$ can be solved using row reduction:

$$\left(\begin{array}{cc|c} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \end{array}\right) \sim \left(\begin{array}{cc|c} a_{11} & a_{12} & b_1 \\ 0 & a_{22} - \frac{a_{12}a_{21}}{a_{11}} & b_2 - \frac{a_{21}b_1}{a_{11}} \end{array}\right)$$

x_2 can be computed as $x_2 = \dfrac{b_2 - \frac{a_{21}b_1}{a_{11}}}{a_{22} - \frac{a_{12}a_{21}}{a_{11}}}$

Then $x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2)$ (this is called <u>back-substitution</u>)

This algorithm is <u>very</u> easy to break.

Ex: $a_{11} = 0$ or $a_{22} - \frac{a_{12}a_{21}}{a_{11}} = 0$.

In order to systematically solve $A\vec{x} = \vec{b}$ using Gaussian elimination, one must use <u>pivoting</u>.

<u>Ex:</u> $A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ must be (row) pivoted to $\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$.

On every step, it must be ensured that the <u>pivot element</u> is non-zero.

How expensive is Gaussian elimination?
Lets count operations: (treat $+, -, \div, \times$ as having the same cost)

To put A in echelon form: $\begin{pmatrix} a_{11} & \cdot & \cdots & \vdots \\ & a_{22} & & \vdots \\ 0 & & \ddots & \\ & & & a_{nn} \end{pmatrix}$

Loop over columns $j = 1, .., n-1$

Loop over rows $i = j+1, \dots, n$

① compute $\frac{a_{ij}}{a_{jj}}$ (1 flop)

② compute row $i - \frac{a_{ij}}{a_{jj}}$ row j $\left(2(n-j) \text{ flops}\right)$

③ compute $b_i - \frac{a_{ij}}{a_{jj}} b_j$ (2 flop)

<u>Total</u> (for each j)
$2(n-j) + 3$ flops

Now compute the total:

$$\sum_{j=1}^{n-1} \sum_{i=j+1}^{n} \left(2(n-j)+3\right) = \sum_{j=1}^{n-1} (n-j)\left(2(n-j)+3\right)$$

$$\approx 2\sum_{j=1}^{n-1} (n-j)^2 \approx \mathcal{O}(n^3) \text{ flops.}$$

(of course there are more careful derivations)