Last time:

- Fixed point convergence
  \[ \Rightarrow \text{if } |f'(s)| > 1 \text{ at fixed point } s \text{, then } x_{n+1} = f(x_n) \text{ does not converge, } s \text{ is unstable} \]

- 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 $\varepsilon_{\text{mach}}$ in MATLAB.

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

\[ \text{Ex: } a + b = \text{round} (a + b) \]

\[ \therefore \quad (a + b) \{1 + \varepsilon\} \]

This is a statement of relative accuracy:

\[ \varepsilon = \left| \frac{\text{round} (a + b) - (a + b)}{(a + b)} \right| = |\varepsilon| \leq \varepsilon \]

\[ \text{Ex: } \left| \text{round} (a + b) - (a + b) \right| = |a + b| |\varepsilon| \leq (|a| + |b|) \varepsilon \]

\[ \text{absolute accuracy} \quad \rightarrow \quad \leq 2 \cdot \max (|a|, |b|) \varepsilon. \]
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 \( A\hat{x} = \hat{b} \)
  - minimize \( \| A\hat{x} - \hat{b} \| \) (least squares)

- **Factorizations**
  - \( A = LU \)
  - \( A = U S V^T \)
  - \( A = QR \)

- **Eigen computations**
  - Find all \( \lambda_i, v_i \) s.t. \( Av_i = \lambda_i v_i \).
Efficient methods for doing all these calculations are the building blocks of almost all scientific computing.

Tell Levi Strauss anecdote.

Ignore things like $AB$, $Ax$, $V^TV$ for now; these are computations that merely need to be optimized, a CS endeavor.

**Note:** Computational cost: $A \sim n \times n$, $V, \tilde{V} \sim n \times 1$

- $V^TV \sim O(n)$ flops
- $Ax \sim O(n^2)$ flops
- $ATA \sim O(n^3)$ flops

Consequence for larger computers:

A compute with twice the speed/storage can only compute $ATA$ with an $\sqrt{2}n$ in the same time as original machine.

First problem to tackle: solve $Ax = b$

using Gaussian elimination
Recall: Let \( A \) be a \( 2 \times 2 \) matrix
\[
\begin{pmatrix}
\begin{array}{cc}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{array}
\end{pmatrix}
\]

\( A \mathbf{x} = \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \) can be solved using row reduction:

\[
\begin{pmatrix}
\begin{array}{cc|c}
a_{11} & a_{12} & b_1 \\
a_{21} & a_{22} & b_2
\end{array}
\end{pmatrix} \rightarrow \begin{pmatrix}
\begin{array}{cc|c}
a_{11} & a_{12} & b_1 \\
0 & a_{22} - \frac{a_{12} a_{21}}{a_{11}} & b_2 - \frac{a_{12} b_1}{a_{11}}
\end{array}
\end{pmatrix}
\]

\( x_2 \) can be computed as
\[
x_2 = b_2 - \frac{a_{12} b_1}{a_{11}} \frac{a_{22} - \frac{a_{12} a_{21}}{a_{11}}}{a_{22} - \frac{a_{12} a_{21}}{a_{11}}}
\]

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

This algorithm is very easy to break.

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

In order to systematically solve \( A \mathbf{x} = \mathbf{b} \) using Gaussian elimination, one must use pivoting.

\( \text{Ex: } A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \) must be (row) pivoted to \( \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \).
On every step, it must be ensured that the top element is non-zero.

How expensive is Gaussian elimination?
Let's count operations: (treat +, -, ÷, × as having the same cost)

To put $A$ in echelon form: \[
\begin{pmatrix}
  a_{11} & a_{12} & \cdots & a_{1n} \\
  0 & a_{22} & \cdots & a_{2n} \\
  & & & \ddots \\
  & & & 0 & a_{nn}
\end{pmatrix}
\]

Loop over columns $j = 1, \ldots, n-1$
Loop over rows $i = j+1, \ldots, n$

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

2. compute row $i$ $-$ $\frac{a_{ij}}{a_{jj}}$ row $j$ \hspace{1cm} (2(n-j) flops)

3. compute $b_i$ $-$ $\frac{a_{ij}}{a_{jj}}$ $b_j$ \hspace{1cm} (2 flop)

Total \hspace{1cm} (for each $j$) \hspace{1cm} $2(n-j) + 3$ flops

Now compute the total:
\[
\sum_{j=1}^{n-1} \sum_{i=j+1}^{n} \left( 2(n-j) + 3 \right) = \sum_{j=1}^{n} (n-j)(2(n-j) + 3)
\]
\[
\approx 2 \sum_{j=1}^{n-1} (n-j)^2 \approx O(n^3) \text{ flops. (of course, the more careful derivation)}
\]