Next: Numerical Linear Algebra

Basically, the only math your computer can do is linear algebra. Then 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**:  
  - \( A = LU \)  
  - \( A = USV^T \)  
  - \( A = QR \)
- **Eigen computation**:  
  - Find all \( \lambda_i, \hat{v}_i \) s.t. \( A\hat{v}_i = \lambda_i\hat{v}_i \).
Efficient methods for doing all these calculations are the building blocks of almost all scientific computing.

Tell Levi Strauss anecdote.

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

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{u}^T\hat{v}$</td>
<td>$O(n)$ flops</td>
</tr>
<tr>
<td>$A\hat{x}$</td>
<td>$O(n^2)$ flops</td>
</tr>
<tr>
<td>$A^TA$</td>
<td>$O(n^3)$ flops</td>
</tr>
</tbody>
</table>

Consequence for larger computers:

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

First problem to tackle: solve $A\hat{x} = \hat{b}$

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:

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

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

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.

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 pivoting.

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 first element 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: \[
\begin{pmatrix}
 a_{11} & \cdots & a_{1n} \\
 0 & \ddots & \vdots \\
 0 & \cdots & a_{nn}
\end{pmatrix}
\]

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

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

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

Now compute the total:
\[
\sum_{j=1}^{n-1} \sum_{i=j+1}^{n} (2(n-j) + 3) = \sum_{j=1}^{n-1} (n-j)(2(n-j) + 3)
\]
\[
\approx 2 \sum_{j=1}^{n-1} (n-j)^2 \approx O(n^3) \text{ flops. (Of course then an more careful derivation)}
\]
Another way to think of

Gaussian Elimination: LV factorization:

- Each row operation corresponds to multiplication by a lower triangular matrix.

Ex:

\[
A = \begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 & 0 & 0 \\
-4 & 1 & 0 \\
-7 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{pmatrix}
= \begin{pmatrix}
1 & 2 & 3 \\
0 & -3 & -6 \\
0 & -6 & -21
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & -2 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 2 & 3 \\
0 & -3 & -6 \\
0 & -6 & -21
\end{pmatrix}
= \begin{pmatrix}
1 & 2 & 3 \\
0 & -3 & -6 \\
0 & 0 & -9
\end{pmatrix}
\]

So \( L_2 L_1 A = U \)

\[
A = (L_2 L_1)^{-1} U
= L_1^{-1} L_2^{-1} U
\]

These "undo" the row operations in \( L_1, L_2 \).

\[
L_1^{-1} = \begin{pmatrix}
1 & 0 & 0 \\
-4 & 1 & 0 \\
-7 & 0 & 1
\end{pmatrix}
\]

\[
L_2^{-1} = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & -2 & 1
\end{pmatrix}
\]

\[
L_1 L_2 = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & -2 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 & 0 \\
-4 & 1 & 0 \\
-7 & 0 & 1
\end{pmatrix}
= \begin{pmatrix}
1 & 0 & 0 \\
4 & 1 & 0 \\
7 & 0 & 1
\end{pmatrix}
\]
Gaussian Elimination & LU factorization are equivalent.

Easy to alter GE code to save LU (exercise):

Then to solve $A\hat{x} = \hat{b}$, \((LU\hat{x}) = \hat{b}\)

1. Solve $L\hat{y} = \hat{b} \quad \text{Forward substitution}$
2. Solve $U\hat{x} = \hat{y} \quad \text{Backward substitution}$

Operation count? (For the solve)

\[
\begin{pmatrix}
lu & lu & \ldots & lu \\
lu & lu & \ldots & \ldots \\
\vdots & \vdots & \ddots & \vdots \\
lu & \ldots & \ldots & l_{nn}
\end{pmatrix}
\hat{x} = \hat{b}
\]

\[
y_i = b_i / l_{ii} \quad \text{are all 1's in our case}
\]

\[
y_i = (b_i - l_{i1}y_1) / l_{ii}
\]

\[
do j = 1, n \quad \text{not included in op. count.}
\]

\[
y_i = y_i - l_{ij} \cdot y_j
\]

End do

End do

\[
\text{Operations:} \quad \sum_{c=1}^{n} \sum_{j=1}^{c-1} 2 = 2 \sum_{c=1}^{n} (c-1)
\]

\[
= 2 \sum_{c=1}^{n} c - 2 \cdot \frac{n(n-1)}{2} = n^2 + \Theta(n)
\]

This is fast