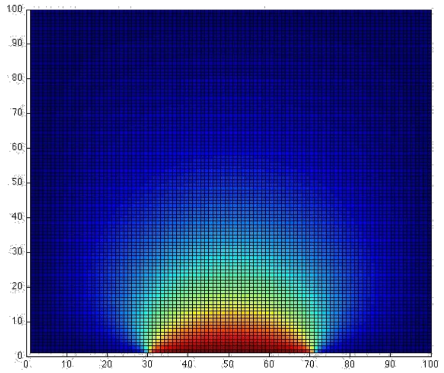

Lecture 03:

# Computation of LU Factorization

CS 111: Intro to Computational Science

Spring 2023

Ziad Matni, Ph.D.

Dept. of Computer Science, UCSB

# Administrative

- Current homework assignment due tonight (by 11:59 pm)
  - Any issues?
- New homework will be uploaded on Canvas later today
- Don't forget: Quiz 1 on Wednesday!
  - Please be here a little early so we can start on time at 9:30 am

# Use of Python

- We will be using the **numpy** module extensively
  - Numerical computing with arrays and matrices
  - https://docs.scipy.org/doc/numpy/reference/
  - http://www.numpy.org/

- Also, read more about **scipy** and **matplotlib**:
  - **scipy**: More advanced numerical computing, including sparse matrices
  - https://docs.scipy.org/doc/scipy/reference/
  - **matplotlib**: Plotting and visualization
  - https://matplotlib.org/contents.html

```python
# These are the standard imports for CS 111.

import os
import time
import math
import numpy as np
import numpy.linalg as npla
import scipy
from scipy import sparse
from scipy import linalg
import scipy.sparse.linalg as spla
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import axes3d
%matplotlib tk
```

The ones highlighted in red are the core ones you need for every exercise in this class!

# Let's Do this In Python!

We turn to a demonstration using

***Jupyter Notebook***

You will get the entire transcript posted on our class' Main Website

# Upper and Lower Triangle Matrices

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & & \vdots \\ 0 & 0 & u_{33} & & \vdots \\ \vdots & & & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & u_{nn} \end{bmatrix} \qquad L = \begin{bmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & & \vdots \\ l_{31} & l_{32} & l_{33} & & \vdots \\ \vdots & & & \ddots & 0 \\ l_{n1} & \cdots & \cdots & \cdots & l_{nn} \end{bmatrix}$$

- A matrix **U** is an **upper triangular matrix** if its nonzero elements are found ***only*** in the upper triangle of the matrix, including the main diagonal; that is: $u_{ij} = 0$ **if i > j**

- A matrix **L** is an **lower triangular matrix** if its nonzero elements are found ***only*** in the lower triangle of the matrix, including the main diagonal; that is: $l_{ij} = 0$ **if i < j**
  - Special case: when the diagonal is all 1s, we call that a ***unit lower triangle matrix***

It is often useful to factor a matrix A as a multiplication of a lower- with an upper-triangle matrix.
i.e. **A = L.U**

Sometimes, we **have to** move A's rows around before we are able to do a proper factorization, this is the same as first multiplying by the appropriate permutation matrix.
i.e. **P.A = L.U**

# Matrix Factorization using Gaussian Elimination

- Recall the **Ax** = **b** example where $\mathbf{A} = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$ $\begin{matrix} a00 & a01 \\ a10 & a11 \end{matrix}$

- Note what happens when we try to do Gaussian elimination on A:
  - In order to make element $a_{10}$ become zero, then I need to make: $r_1 = r_1 + ½r_0$
  - So, using the general rule: $\mathbf{r_1 = r_1 – C.r_0}$ it means: **C = –0.5**

- Note that I can write **A** as a factor of 2 other matrices:

$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -0.5 & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ 0 & 1.5 \end{bmatrix}$$

*Note: This matrix has **1** in the diagonals and the bottom part of it has the coefficient **C (-0.5)***

- Important Rule: I can ONLY do this if **A** is invertible! (already proven…)

# Matrix Factorization using Gaussian Elimination

- Note that I can write **A** as a factor of 2 other matrices:

$$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -0.5 & 1 \end{bmatrix} \begin{bmatrix} \boxed{2} & -1 \\ 0 & \boxed{1.5} \end{bmatrix}$$

*Note: This matrix is a unit lower-triangle-type (L)*
***The numbers in the lower-part are the linear coefficients***

*Note: This matrix is an upper-triangle type (U)*
***The diagonals are the pivots***

# Matrix Factorization: A = LU

- Since it's a logical, algorithmic way to do this factorization, that means that there is a *computational way* of doing this…
- ***Step 1***: Use Gaussian Elimination to transform **A** into a **U** via *row reduction process*
- ***Step 2***: The **L** matrix is populated by various *coefficients* from the transformation process

- Seems simple enough…
- BUT, it isn't always… lots of "what-ifs"…

# Matrix Factorization: A = LU

- Example 1: Using pivoting (classical, manual)
- Example 2: Without pivoting (for easy computations)
- Example 3: When no-pivoting doesn't work… (better computations)

**Demonstrations**

# Developing the Algorithm 1

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \\ -1 & 1 & 2 \end{bmatrix}$$

Matrix A has to be a SQUARE matrix
and invertible for this to work
A[0,0] is called the pivot of row 0 ($r_0$)

*Algorithm --- GOAL: transform A into an upper-triangle matrix using Gaussian Elimination:*

1. Start with $r_1$ vs $r_0$
2. Find the multiplier needed for row reduction of $r_1$ by $r_0$
    a) **m** = A[1,0]/A[0,0] = 1/1 = 1
3. Put multiplier in the space where 0 will be: A[1,0]
4. Change the rest of the row based on **m**
    a) $r_1 = r_1 - \mathbf{m}r_0 = r_1 - r_0$
    b) So $r_1$ changes from [1, 1, 1] to [0, -1, -2]

© 2023, Matni

# Developing the Algorithm 2

$$\rightarrow \quad \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ -1 & 1 & 2 \end{bmatrix} \qquad m = 1$$

*__Algorithm:__*

5. Shift over to $r_2$ vs $r_0$ & repeat steps 2, 3, 4…
   a) **m** = A[2,0]/A[0,0] = -1/1 = -1
   b) $r_2 = r_2 - \mathbf{m}r_0 = r_2 + r_0$
   c) So $r_2$ changes from [-1, 1, 2] to [0, 3, 5]

# Developing the Algorithm 3

$$\rightarrow \quad \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ 0 & 3 & 5 \end{bmatrix} \begin{array}{l} \\ m = 1 \\ m = -1 \end{array}$$

*__Algorithm:__*

6. We're done with comparisons with $r_0$, let's move on to $r_1$…

   Do $r_2$ vs $r_1$

   a) **m** = A[2,1]/A[1,1] = 3/-1 = -3

   b) $r_2 = r_2 - \mathbf{m}r_1 = r_2 + 3r_1$

   c) So $r_2$ changes from [0, 3, 5] to [0, 0, -1]

# Developing the Algorithm 4

$$\rightarrow \quad \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ 0 & 0 & -1 \end{bmatrix} \quad \begin{matrix} \\ m = 1 \\ m = -1, \; -3 \end{matrix}$$

*Algorithm:*

7. You now have an upper triangle matrix (**U**).
   You can construct matrix **L** from the multipliers (& make **L**'s diagonals all 1s):

$$A = L.U = \begin{bmatrix} \mathbf{1} & 0 & 0 \\ 1 & \mathbf{1} & 0 \\ -1 & -3 & \mathbf{1} \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ 0 & 0 & -1 \end{bmatrix} \xrightarrow[\textit{Original A}]{\text{Check it?}} \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \\ -1 & 1 & 2 \end{bmatrix} ✔$$

# Let’s Go to Python!

- And try to develop this into a **function**!

```
Algorithm: Given a matrix A of size nxn

For p in [0, n):
    pivot = A[p ,p]
    if pivot == 0 then quit
    For row in (p+1, n]:
        m = A[row, p] / pivot
        A[row, p] = m
        # The rest of the column values in
        # that row of A get modified by m
        A[row, p+1 thru end] -= m * A[p, p+1 thu end]
    Separate L and U in the resulting A
    Return L and U
```

# Your TO DOs!

- Turn in current homework on Gradescope
- Start with new homework (due next week Monday)

# </LECTURE>