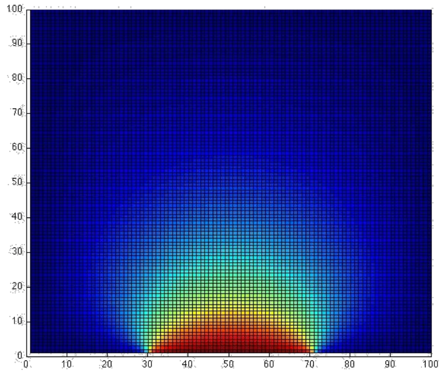

Lecture 04:
LU Factorization in Computation 2

CS 111: Intro to Computational Science

Spring 2023

Ziad Matni, Ph.D.

Dept. of Computer Science, UCSB

Administrative

- New homework!

Developing the Algorithm 1

$$A = \begin{array}{c} r_0 \\ r_1 \\ r_2 \end{array} \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \\ -1 & 1 & 2 \end{bmatrix}$$

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 – look for the pivot from 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. 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]**
4. Put **m** in a **NEW 3x3** matrix (what will be the **L** matrix) in the **[1,0]** position

Ra = Ra – m.Rb

Developing the Algorithm 2

$$A = \begin{array}{c} r_0 \\ r_1 \\ r_2 \end{array} \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \\ -1 & 1 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ -1 & 1 & 2 \end{bmatrix} \quad m = 1$$

Algorithm:

5. Shift over to $\mathbf{r_2}$ **vs** $\mathbf{r_0}$ & repeat steps 2, 3, 4…
   a) Multiplier: **m** = A[2,0]/A[0,0] = -1/1 = -1
   b) Row reduction: $r_2 = r_2 - \mathbf{m}r_0 = r_2 + r_0$
   c) Transformation: r_2 changes from [-1, 1, 2] to **[0, 3, 5]**
   d) Update the new **L** matrix

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 $\mathbf{r_2}$ **vs** $\mathbf{r_1}$
   a) Multiplier: **m** = A[2,1]/A[1,1] = 3/-1 = -3
   b) Row reduction: $r_2 = r_2 - \mathbf{m}r_1 = r_2 + 3r_1$
   c) Transformation: r_2 changes from [0, 3, 5] to **[0, 0, -1]**
   d) Update the new **L** matrix

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 = the transformed matrix A**).

8. Finish constructing matrix **L** from the multipliers (have to make **L**'s diagonals all 1s):

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

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
```

Avoiding Problems in LU Factorization

- Take, for example, this 3x3 A = LU :

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

- Note that $a_{11} = l_{11}.u_{11}$
- If it's ever the case that $a_{11} = 0$, then ***either*** l_{11} ***or*** u_{11} has to be 0
  - Not good!!! It means either L or U are *singular (non-invertible)* *(why??)*
  - Solution: *re-arrange the rows of matrix A* *(is that ok to do?)*
  - This is called **pivoting**

When a Pivot Goes Bunk… (Bad)

- If a **pivot = 0**… it's a problem…
- Because **m** cannot be computed! (divide-by-zero error)

- Solution: re-order the rows in the matrix & try again
  - Done algorithmically by using a **permutation matrix** (i.e. **P.A**)

- Let's apply all of this knowledge to a Python function!

Demonstration with a Python-ized solution!

Using LU Factorization to Solve **Ax = b** Problems

- Assume you know the factors of **A** = **LU**

**from a computational resources sense*

- Since triangular matrices are cheaper* to do calculations with, can we use **L** and **U** in calculations to find **x**?
  - Explanation:

$$A\mathbf{x} = b \;\Rightarrow\; L(U\mathbf{x}) = b$$

So if we call $U\mathbf{x}$ = **y**, so that $L\mathbf{y} = b$,

then we can solve for **y** first.

Then we can extract the **U** from **y** and we're left with **x**

Why Can’t We Use Other Techniques?

- LU factorization to solve for **Ax = b** *guarantees* the smallest errors (residuals) in computation
  - It’s still computationally expensive though – compared to some other methods (not learned yet…)
  - But that’s the trade-off: **Accuracy vs. Expense**

- Using other computational techniques, like inverting matrices can introduce more errors
  - Especially if pivots are *very small, but non-zero, numbers*
    - *What’s the danger there?*
  - More on round-off errors and the like later…

```
   Ax = b
➔ x = A^-1 b
Not a good technique!
```

Recall: Calculating Error Factors

- **Residual**
  - The difference between your *expected* and *calculated* results
  - In an **Ax** = **b** scenario, that would be **b – Ax**
    - Ideally, this difference should be **zero**
    - ***When is it not zero??***
  - Note: this calculation yields a matrix (a vector in this case)

Metrics to Calculate Error Factors

- **Residual** *vs.* **Norm** of the Residual *vs.* Relative **Norm** of the Residual
  - The norm of a matrix is *a* measure of its **magnitude**
  - The most *common* way to calculate the **norm** is to use the Euclidean approach

Your TO DOs!

- New homework (due next week Monday)
- Lab tomorrow!

</LECTURE>