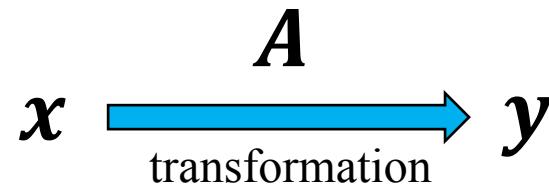


Solving Linear System of Equations

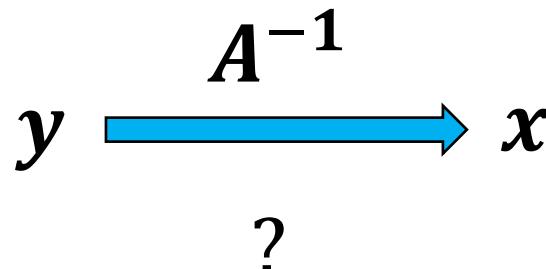
The “Undo” button for Linear Operations

Matrix-vector multiplication: given the data \mathbf{x} and the operator \mathbf{A} , we can find \mathbf{y} such that

$$\mathbf{y} = \mathbf{A} \mathbf{x}$$



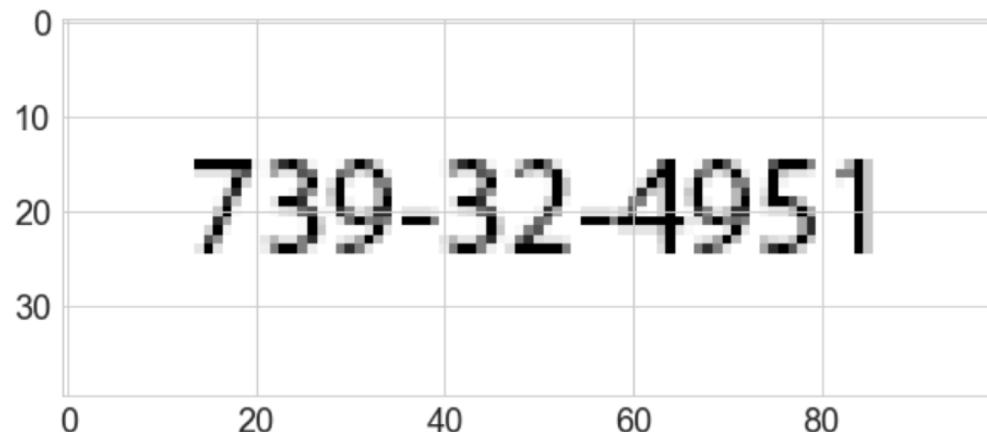
What if we know \mathbf{y} but not \mathbf{x} ? How can we “undo” the transformation?



Solve $\mathbf{A} \mathbf{x} = \mathbf{y}$ for \mathbf{x}

Image Blurring Example

Demo “Image Blur Inverse”

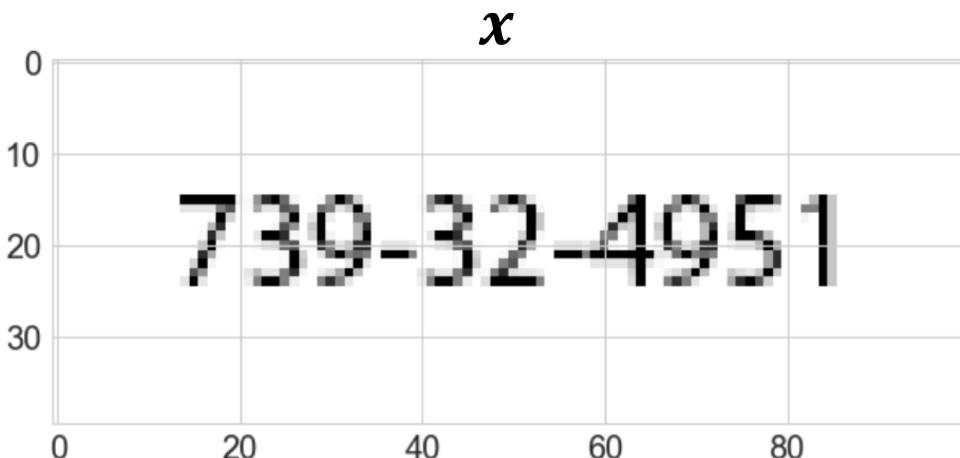


- Image is stored as a 2D array of real numbers between 0 and 1
(0 represents a white pixel, 1 represents a black pixel)
- **$xmat$** contains the 2D data (the image) with dimensions 100x40
- Flatten the 2D array as a 1D array
- \mathbf{x} contains the 1D data with dimension 4000,
- Apply blurring operation to data \mathbf{x} , i.e.

$$\mathbf{y} = \mathbf{A} \mathbf{x}$$

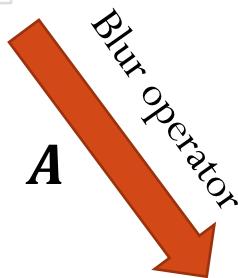
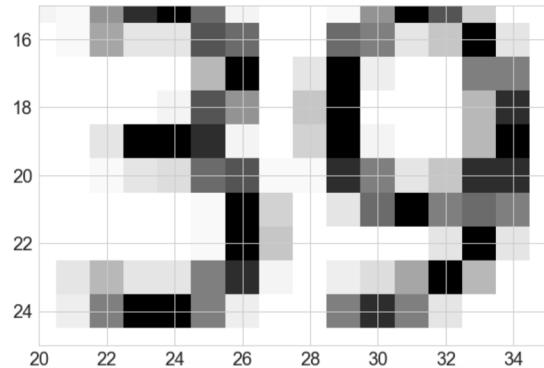
where \mathbf{A} is the blur operator and \mathbf{y} is the blurred image

Blur operator



```
plt.imshow(xmat);
plt.axis([20,35,25,15])
```

[20, 35, 25, 15]

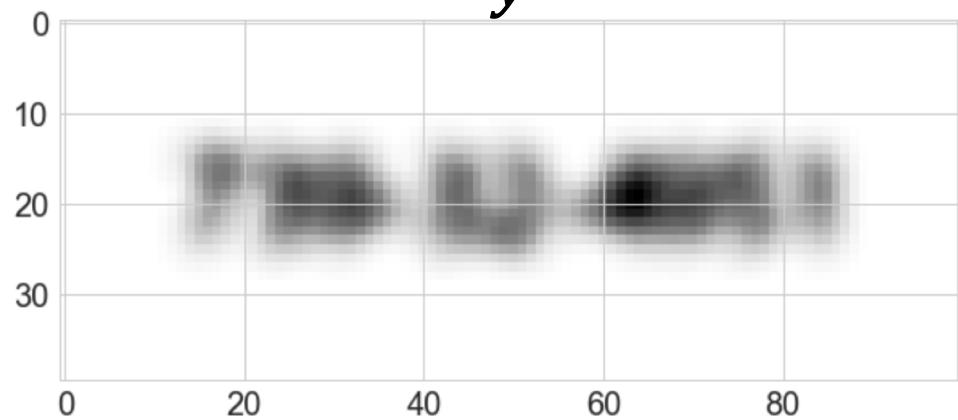


blurred image (4000,)

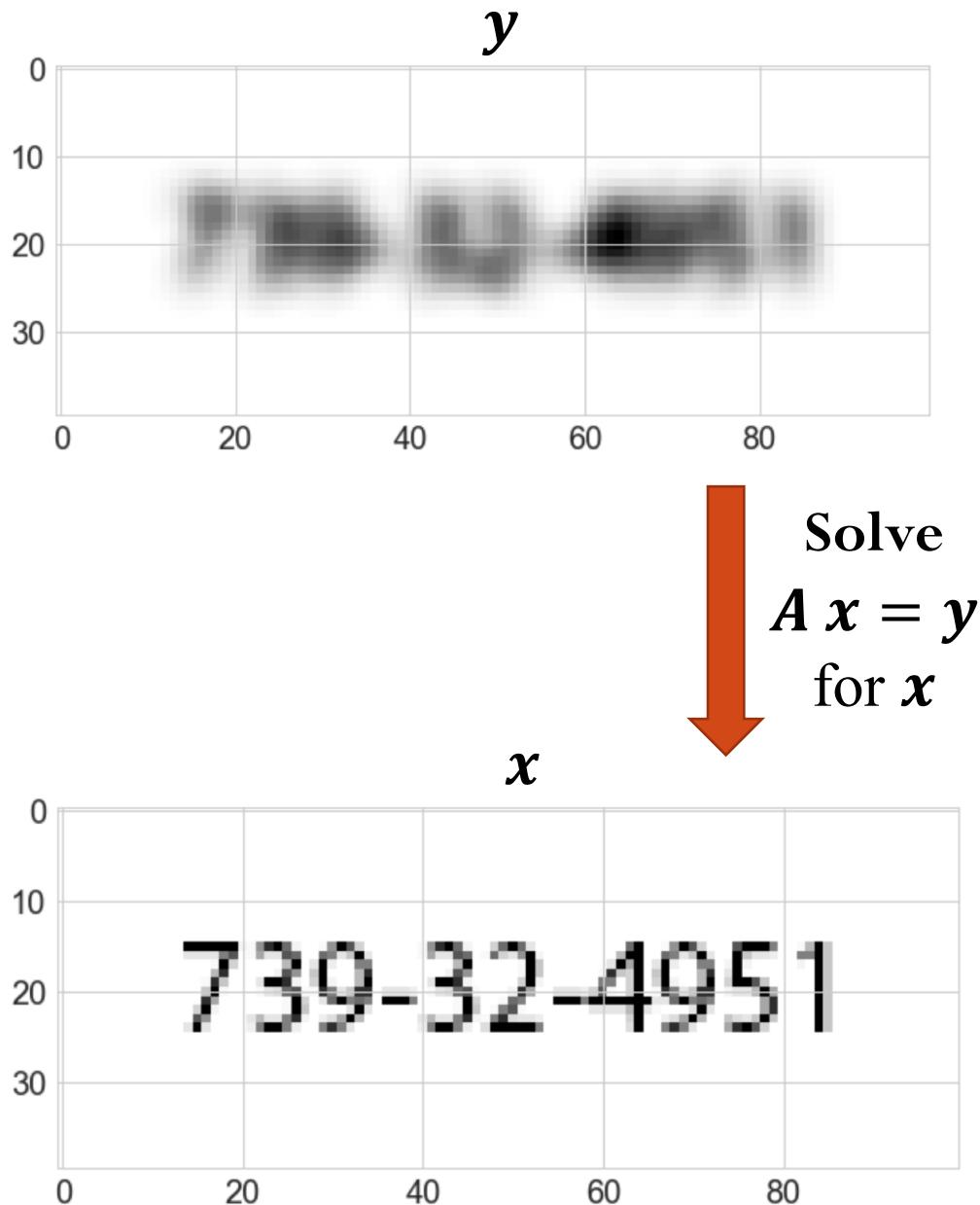
Blur operator (4000,4000)

"original" image (4000,)

$$y = A x$$



”Undo” Blur to recover original image



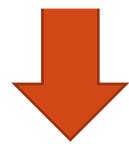
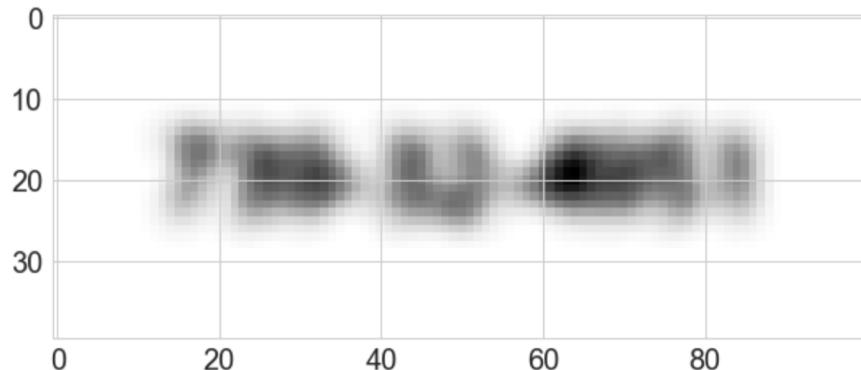
Assumptions:

1. we know the blur operator A
2. the data set y does not have any noise (“clean data”)

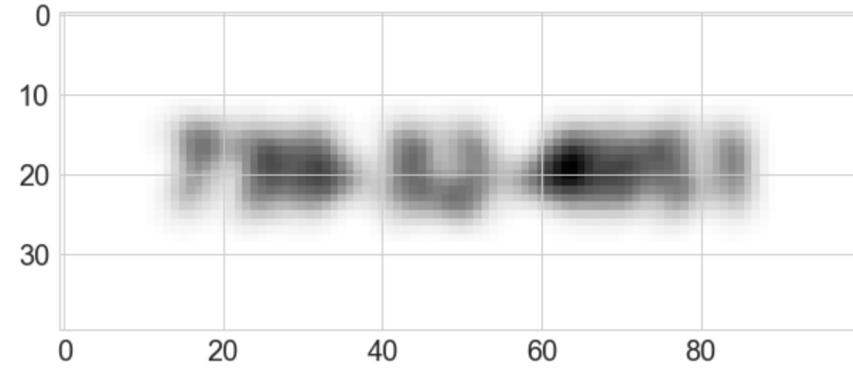
What happens if we add some noise to y ?

”Undo” Blur to recover original image

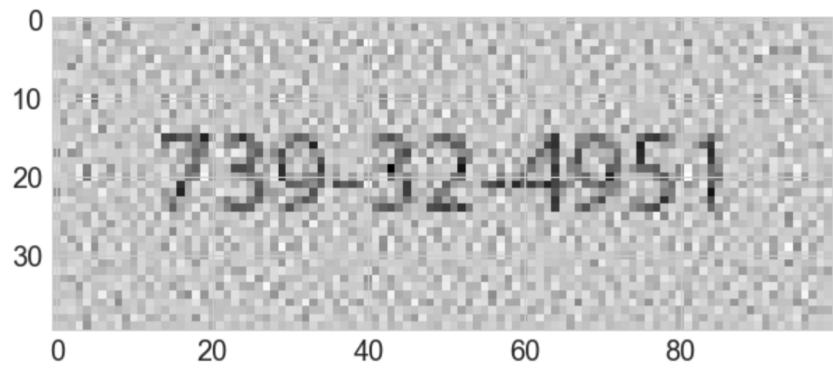
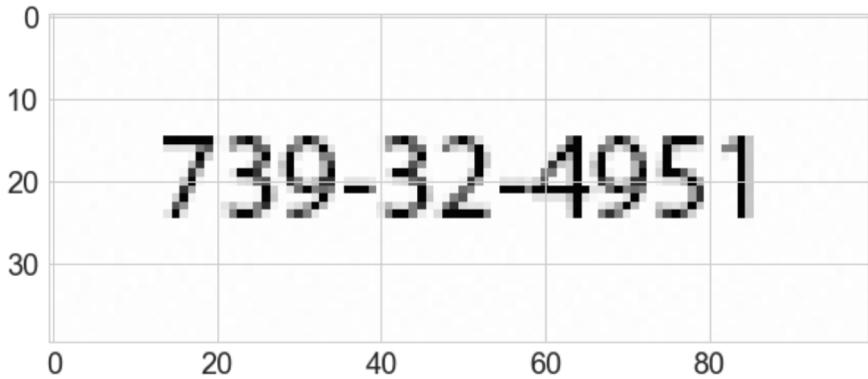
$$\mathbf{y} + a * 10^{-6} \ (a \in (0,1))$$



$$\mathbf{y} + a * 10^{-4} \ (a \in (0,1))$$



Solve $\mathbf{A} \mathbf{x} = \mathbf{y}$ for \mathbf{x}



How much noise can we add and still be able to recover meaningful information from the original image? At which point this inverse transformation fails?
We will talk about sensitivity of the “undo” operation later.

Linear System of Equations

How do we actually solve $\mathbf{A} \mathbf{x} = \mathbf{b}$?

We can start with an “easier” system of equations...

Let’s consider triangular matrices (lower and upper):

$$\begin{pmatrix} L_{11} & 0 & \dots & 0 \\ L_{21} & L_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{n1} & L_{n2} & \dots & L_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$\begin{pmatrix} U_{11} & U_{12} & \dots & U_{1n} \\ 0 & U_{22} & \dots & U_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & U_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Example: Forward-substitution for lower triangular systems

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 3 & 2 & 0 & 0 \\ 1 & 2 & 6 & 0 \\ 1 & 3 & 4 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 6 \\ 4 \end{pmatrix}$$

Triangular Matrices

$$\begin{pmatrix} U_{11} & U_{12} & \dots & U_{1n} \\ 0 & U_{22} & \dots & U_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & U_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Recall that we can also write $\mathbf{U} \mathbf{x} = \mathbf{b}$ as a linear combination of the columns of \mathbf{U}

Triangular Matrices

Forward-substitution for lower-triangular systems:

$$\begin{pmatrix} L_{11} & 0 & \dots & 0 \\ L_{21} & L_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{n1} & L_{n2} & \dots & L_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$x_1 = b_1 / L_{11} \quad x_i = \frac{b_i - \sum_{j=1}^{i-1} L_{ij} x_j}{L_{ii}}, \quad i = 2, 3, \dots, n$$

Example:

$$\begin{pmatrix} 2 & 8 & 4 & 2 \\ 0 & 4 & 4 & 3 \\ 0 & 0 & 6 & 2 \\ 0 & 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 4 \\ 1 \end{pmatrix}$$

$$x_4 = \frac{1}{2}$$

$$x_3 = \frac{4 - 2\frac{1}{2}}{6} = \frac{1}{2}$$

$$x_2 = \frac{4 - 4\frac{1}{2} - 3\frac{1}{2}}{4} = \frac{1/2}{4} = \frac{1}{8}$$

$$x_1 = \frac{2 - 8\frac{1}{8} - 4\frac{1}{2} - 2\frac{1}{2}}{2} = \frac{-2}{2} = -1$$

Cost of solving triangular systems

$$x_n = b_n/U_{nn} \quad x_i = \frac{b_i - \sum_{j=i+1}^n U_{ij}x_j}{U_{ii}}, \quad i = n-1, n-2, \dots, 1$$

n divisions

$n(n-1)/2$ subtractions/additions

$n(n-1)/2$ multiplications



Computational complexity is $O(n^2)$

$$x_1 = b_1/L_{11} \quad x_i = \frac{b_i - \sum_{j=1}^{i-1} L_{ij}x_j}{L_{ii}}, \quad i = 2, 3, \dots, n$$

n divisions

$n(n-1)/2$ subtractions/additions

$n(n-1)/2$ multiplications



Computational complexity is $O(n^2)$

Demo “Coding Back-substitution”

Linear System of Equations

How do we solve $\mathbf{A} \mathbf{x} = \mathbf{b}$ when \mathbf{A} is a non-triangular matrix?

We can perform LU factorization: given a $n \times n$ matrix \mathbf{A} , obtain lower triangular matrix \mathbf{L} and upper triangular matrix \mathbf{U} such that

$$\mathbf{A} = \mathbf{L}\mathbf{U}$$

where we set the diagonal entries of \mathbf{L} to be equal to 1.

$$\begin{pmatrix} 1 & 0 & \dots & 0 \\ L_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{n1} & L_{n2} & \dots & 1 \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & \dots & U_{1n} \\ 0 & U_{22} & \dots & U_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & U_{nn} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{pmatrix}$$

LU Factorization

$$\begin{pmatrix} 1 & 0 & \dots & 0 \\ L_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{n1} & L_{n2} & \dots & 1 \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & \dots & U_{1n} \\ 0 & U_{22} & \dots & U_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & U_{nn} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{pmatrix}$$

Assuming the LU factorization is known, we can solve the general system

Iclicker question

Assume the $\mathbf{A} = \mathbf{LU}$ factorization is known, yielding:

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.5 & 1 & 1 & 0 \\ 0.5 & 0.5 & 0.5 & 1 \end{pmatrix} \quad \mathbf{U} = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 0 & -2 & 1 & 2.5 \\ 0 & 0 & 3 & -1 \\ 0 & 0 & 0 & 0.75 \end{pmatrix}$$

Determine the solution \mathbf{x} that satisfies $\mathbf{Ax} = \mathbf{b}$, when $\mathbf{b} = \begin{pmatrix} 2 \\ 2 \\ 1 \\ 4 \end{pmatrix}$

$$\mathbf{L} \underbrace{\mathbf{Ux}}_{\mathbf{y}} = \mathbf{b}$$

First, solve the lower-triangular system $\mathbf{L} \mathbf{y} = \mathbf{b}$ for the variable \mathbf{y}

Then, solve the upper-triangular system $\mathbf{U} \mathbf{x} = \mathbf{y}$ for the variable \mathbf{x}

2x2 LU Factorization (simple example)

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ L_{21} & 1 \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}$$

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} U_{11} & U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + U_{22} \end{pmatrix} \rightarrow U_{11} = A_{21}/U_{11}$$

2) $L_{21} = A_{21}/U_{11}$

3) $U_{22} = A_{22} - L_{21}U_{12}$

Seems quite simple! Can we generalize this for a $n \times n$ matrix A ?

Iclicker question

Which of the following statements about the $A = LU$ factorization are true?

- 1) It can happen that A is invertible, but U is not.
- 2) U has a diagonal that is full of ones.
- 3) A factorization $A = LU$ does not always exist.
- 4) L has a diagonal that is full of ones.

- A) 1 and 2
- B) 1 and 3
- C) 2 and 3
- D) 2 and 4
- E) 3 and 4

Iclicker question

Computing the Lower-Triangular Factor in LU

1 point

Consider the matrix

$$A = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix}$$

and its corresponding LU factorization ($A = LU$), where the lower and upper triangular matrices given respectively by

$$L = \begin{bmatrix} 1 & 0 \\ l_{21} & 1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}.$$

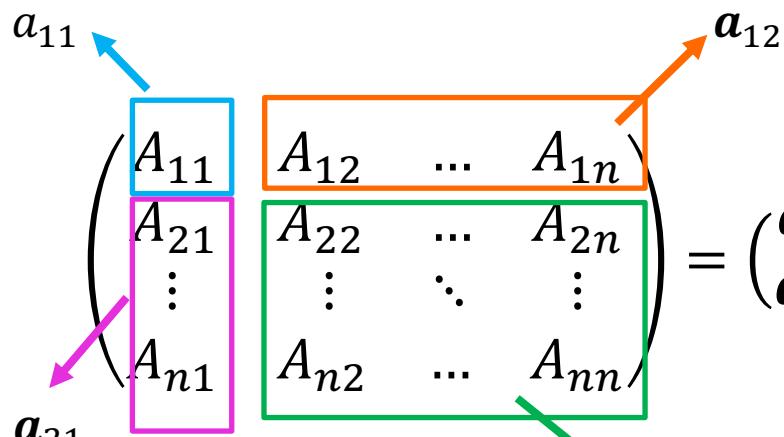
$l_{21} =$

$u_{11} =$

$u_{12} =$

$u_{22} =$

LU Factorization

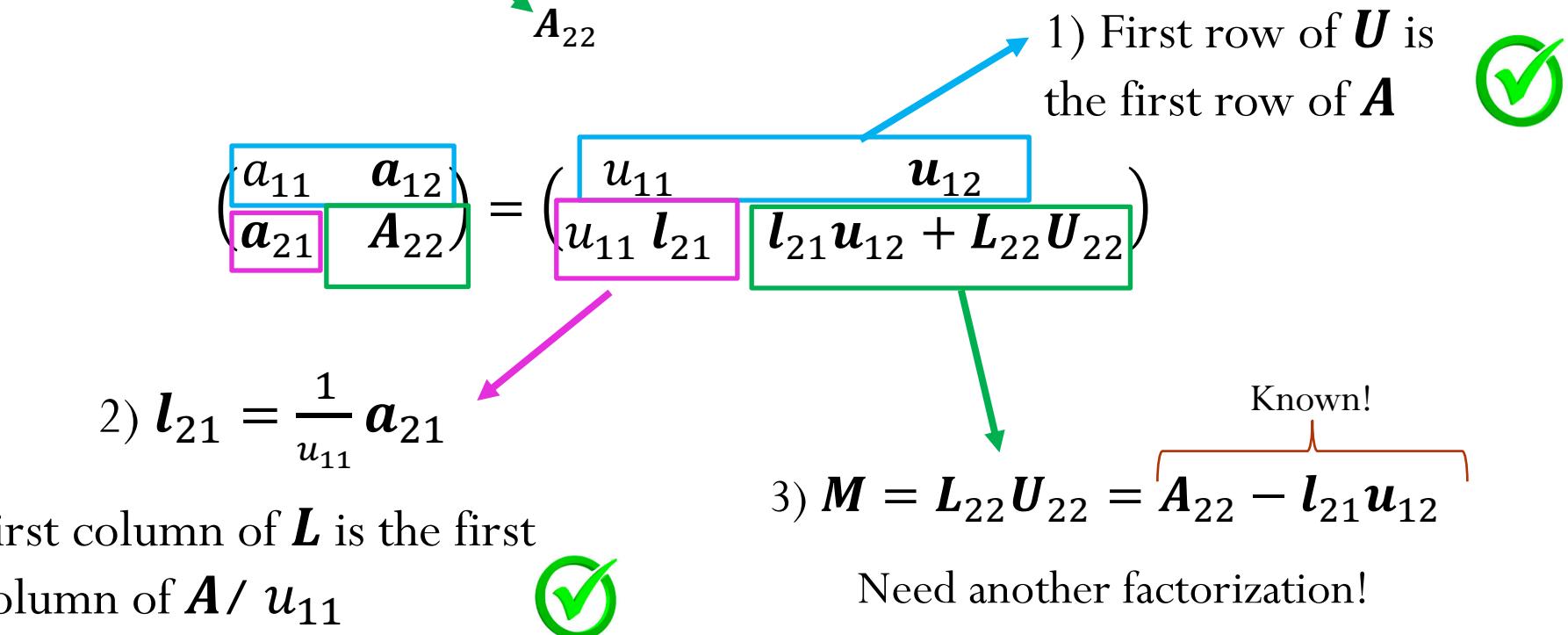


a_{11} : scalar

\mathbf{a}_{12} : row vector ($1 \times (n - 1)$)

\mathbf{a}_{21} : column vector ($(n - 1) \times 1$)

A_{22} : matrix ($(n - 1) \times (n - 1)$)



First column of \mathbf{L} is the first column of \mathbf{A} / u_{11}



Need another factorization!

Example

$$M = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 1 & 2 & 3 & 3 \\ 1 & 2 & 6 & 2 \\ 1 & 3 & 4 & 2 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \end{pmatrix}$$

$$U = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$M = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 1 & -2 & 1 & 2.5 \\ 1 & -2 & 4 & 1.5 \\ 1 & -1 & 2 & 1.5 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 \end{pmatrix}$$

$$U = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 0 & -2 & 1 & 2.5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$M = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 1 & -2 & 1 & 2.5 \\ 1 & -2 & 3 & -1 \\ 1 & -1 & 1.5 & 0.25 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.5 & 1 & 1 & 0 \\ 0.5 & 0.5 & 0.5 & 0 \end{pmatrix}$$

$$U = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 0 & -2 & 1 & 2.5 \\ 0 & 0 & 3 & -1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$M = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 1 & -2 & 1 & 2.5 \\ 1 & -2 & 3 & -1 \\ 1 & -1 & 1.5 & 0.75 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.5 & 1 & 1 & 0 \\ 0.5 & 0.5 & 0.5 & 1 \end{pmatrix}$$

$$U = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 0 & -2 & 1 & 2.5 \\ 0 & 0 & 3 & -1 \\ 0 & 0 & 0 & 0.75 \end{pmatrix}$$

LU Algorithm

```
A = np.array([[2.0,8,4,1],[1,2,3,3],[1,2,6,2],[1,3,4,2]])
```

```
## Algorithm 1
## Factorization using the block-format,
## creating new matrices L and U
## and not modifying A
print("LU factorization using Algorithm 1")
L = np.zeros((n,n))
U = np.zeros((n,n))
M = A.copy()
for i in range(n):
    U[i,i:] = M[i,i:]
    L[i:,i] = M[i:,i]/U[i,i]
    M[i+1:,i+1:] -= np.outer(L[i+1:,i],U[i,i+1:])
```

```
## Algorithm 2
## Factorization using the block-format
## Matrices L and U are stored in the input matrix
## that could be a copy of A or A itself
print("LU factorization using Algorithm 2")
M = A.copy()
for i in range(n):
    M[i+1:,i] = M[i+1:,i]/M[i,i]
    M[i+1:,i+1:] -= np.outer(M[i+1:,i],M[i,i+1:])
```

LU Algorithm

```
A = np.array([[2.0,8,4,1],[1,2,3,3],[1,2,6,2],[1,3,4,2]])
```

```
## Algorithm 2
## Factorization using the block-format
## Matrices L and U are stored in the input matrix
## that could be a copy of A or A itself
print("LU factorization using Algorithm 2")
M = A.copy()
for i in range(n):
    M[i+1:,i] = M[i+1:,i]/M[i,i]
    M[i+1:,i+1:] -= np.outer(M[i+1:,i],M[i,i+1:])
    For j = i + 1:n - 1
         $M[j,i] = M[j,i]/M[i,i]$ 
```

For $j = i + 1:n - 1$
For $k = i + 1:n - 1$
 $M[j,k] -= M[j,i] * M[i,k]$

```
M = A.copy()
for i in range(n-1):
    for j in range(i+1,n):
        M[j,i] = M[j,i]/M[i,i]
        for k in range(i+1,n):
            M[j,k] -= M[j,i]*M[i,k]
```

Cost of LU factorization

Side note:

$$\sum_{i=1}^m i = \frac{1}{2}m(m + 1)$$

$$\sum_{i=1}^m i^2 = \frac{1}{6}m(m + 1)(2m + 1)$$

```
## Algorithm 1
## Factorization using the block-format,
## creating new matrices L and U
## and not modifying A
print("LU factorization using Algorithm 1")
L = np.zeros((n,n))
U = np.zeros((n,n))
M = A.copy()
for i in range(n):
    U[i,i:] = M[i,i:]
    L[i:,i] = M[i:,i]/U[i,i]
    M[i+1:,i+1:] -= np.outer(L[i+1:,i],U[i,i+1:])
```

Number of divisions: $(n - 1) + (n - 2) + \dots + 1 = n(n - 1)/2$

Number of multiplications $(n - 1)^2 + (n - 2)^2 + \dots + (1)^2 = \frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6}$

Number of subtractions: $(n - 1)^2 + (n - 2)^2 + \dots + (1)^2 = \frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6}$



Computational complexity is $O(n^3)$

Iclicker question

Which of the following statements are true about the LU factorization of an $n \times n$ matrix A , assuming LU factorization of A exists and not considering any row/column interchanges?

Select all that apply:

- 1) $A = LU$.
- 2) LU factorization is exactly performing Gaussian elimination.
- 3) We can solve for $LUX = b$ instead of solving $Ax = b$ to obtain x .
- 4) L is a lower triangular matrix, and is exactly the lower part of A but with unit diagonal.
- 5) U is an upper triangular matrix, and is exactly the upper part of A (including diagonal).

- A) 1,2,3
- B) 1,2,3,5
- C) 1,3
- D) 1,2,3,4,5
- E) 4,5

Solving linear systems

In general, we can solve a linear system of equations following the steps:

- 1) Factorize the matrix \mathbf{A} : $\mathbf{A} = \mathbf{L}\mathbf{U}$ (complexity $O(n^3)$)
- 2) Solve $\mathbf{L} \mathbf{y} = \mathbf{b}$ (complexity $O(n^2)$)
- 3) Solve $\mathbf{U} \mathbf{x} = \mathbf{y}$ (complexity $O(n^2)$)

But why should we decouple the factorization from the actual solve?
(Remember from Linear Algebra, Gaussian Elimination does not
decouple these two steps...)

Iclicker question

Let's assume that when solving the system of equations $\mathbf{K} \mathbf{U} = \mathbf{F}$, we observe the following:

- When the matrix \mathbf{K} has dimensions (100,100), computing the LU factorization takes about 1 second and each solve (forward + backward substitution) takes about 0.01 seconds.

Estimate the total time it will take to find the response \mathbf{U} corresponding to 10 different vectors \mathbf{F} when the matrix \mathbf{K} has dimensions (1000,1000)?

- A) ~ 10 seconds
- B) $\sim 10^2$ seconds
- C) $\sim 10^3$ seconds
- D) $\sim 10^4$ seconds
- E) $\sim 10^5$ seconds

What can go wrong with the previous algorithm?

Demo “Little c”

$$M = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 1 & 4 & 3 & 3 \\ 1 & 2 & 6 & 2 \\ 1 & 3 & 4 & 2 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \\ -0.5 & 0 & 0 & 0 \end{pmatrix}$$

$$U = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$l_{21}u_{12} = \begin{pmatrix} 4 & 2 & 0.5 \\ 4 & 2 & 0.5 \\ 4 & 2 & 0.5 \end{pmatrix}$$

$$M - l_{21}u_{12} = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 1 & 0 & 1 & 2.5 \\ 1 & -2 & 4 & 1.5 \\ 1 & -1 & 2 & 1.5 \end{pmatrix}$$

The next update for the lower triangular matrix will result in a division by zero! LU factorization fails.

What can we do to get something like an LU factorization?

Pivoting

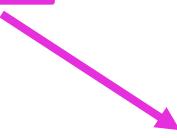
Approach:

1. Swap rows if there is a zero entry in the diagonal
2. Even better idea: Find the largest entry (by absolute value) and swap it to the top row.

The entry we divide by is called the pivot.

Swapping rows to get a bigger pivot is called (partial) pivoting.

$$\begin{pmatrix} a_{11} & \mathbf{a}_{12} \\ \boxed{a_{21}} & A_{22} \end{pmatrix} = \begin{pmatrix} u_{11} & \mathbf{u}_{12} \\ u_{11} l_{21} & l_{21} \mathbf{u}_{12} + L_{22} U_{22} \end{pmatrix}$$



Find the largest entry (in magnitude)

LU Factorization with Partial Pivoting

Example

$$A = M = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 2 \\ 1 & 3 & 4 & 2 \end{pmatrix} \quad L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \end{pmatrix} \quad U = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$M = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 1 & -2 & 1 & 2.5 \\ 1 & -2 & 1 & 1.5 \\ 1 & -1 & 2 & 1.5 \end{pmatrix} \quad L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 \end{pmatrix} \quad U = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 0 & -2 & 1 & 2.5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$M = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 1 & -2 & 1 & 2.5 \\ 1 & -2 & 0 & -1 \\ 1 & -1 & 1.5 & 0.25 \end{pmatrix} \quad L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 \\ 0.5 & 1.0 & 0 & 0 \end{pmatrix} \quad U = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 0 & -2 & 1 & 2.5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$M = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 1 & -2 & 1 & 2.5 \\ 1 & -1 & 1.5 & 0.25 \\ 1 & -2 & 0 & -1 \end{pmatrix} \quad L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.5 & 0.5 & 1.0 & 0 \\ 0.5 & 1.0 & 0 & 1.0 \end{pmatrix} \quad U = \begin{pmatrix} 2 & 8 & 4 & 1 \\ 0 & -2 & 1 & 2.5 \\ 0 & 0 & 1.5 & 0.25 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

Demo “Pivoting example”

$$A = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix}$$

$$\bar{A} = PA = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \\ 0.25 & 0 & 0 & 0 \\ 0.75 & 0 & 0 & 0 \end{pmatrix} \quad U = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad l_{21}u_{12} = \begin{pmatrix} 3.5 & 4.5 & 2.5 \\ 1.75 & 2.25 & 1.25 \\ 5.25 & 6.75 & 3.75 \end{pmatrix}$$

$$\bar{A} - l_{21}u_{12} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 4 & -0.5 & -1.5 & -1.5 \\ 2 & -0.75 & -1.25 & -1.25 \\ 6 & 1.75 & 2.25 & 4.25 \end{pmatrix}$$

Demo “Pivoting example”

$$\bar{A} = \bar{A} - l_{21}u_{12} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 4 & -0.5 & -1.5 & -1.5 \\ 2 & -0.75 & -1.25 & -1.25 \\ 6 & 1.75 & 2.25 & 4.25 \end{pmatrix}$$

$$\bar{A} = P\bar{A} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 8 & 7 & 9 & 5 \\ 6 & 1.75 & 2.25 & 4.25 \\ 2 & -0.75 & -1.25 & -1.25 \\ 4 & -0.5 & -1.5 & -1.5 \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 6 & 1.75 & 2.25 & 4.25 \\ 2 & -0.75 & -1.25 & -1.25 \\ 4 & -0.5 & -1.5 & -1.5 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.75 & 1 & 0 & 0 \\ 0.25 & -0.428 & 0 & 0 \\ 0.5 & -0.285 & 0 & 0 \end{pmatrix} \quad U = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 0 & 1.75 & 2.25 & 4.25 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad l_{21}u_{12} = \begin{pmatrix} -0.963 & -1.819 \\ -0.6412 & -1.2112 \end{pmatrix}$$

$$\bar{A} = \bar{A} - l_{21}u_{12} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 6 & 1.75 & 2.25 & 4.25 \\ 2 & -0.75 & -0.287 & 0.569 \\ 4 & -0.5 & -0.8587 & -0.2887 \end{pmatrix}$$

Demo “Pivoting example”

$$\bar{A} = \bar{A} - l_{21}u_{12} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 6 & 1.75 & 2.25 & 4.25 \\ 2 & -0.75 & -0.287 & 0.569 \\ 4 & -0.5 & -0.8587 & -0.2887 \end{pmatrix}$$

$$\bar{A} = P\bar{A} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 8 & 7 & 9 & 5 \\ 6 & 1.75 & 2.25 & 4.25 \\ 2 & -0.75 & -0.287 & 0.569 \\ 4 & -0.5 & -0.8587 & -0.2887 \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 6 & 1.75 & 2.25 & 4.25 \\ 4 & -0.5 & -0.8587 & -0.287 \\ 2 & -0.75 & 0.569 & 0.2887 \end{pmatrix}$$

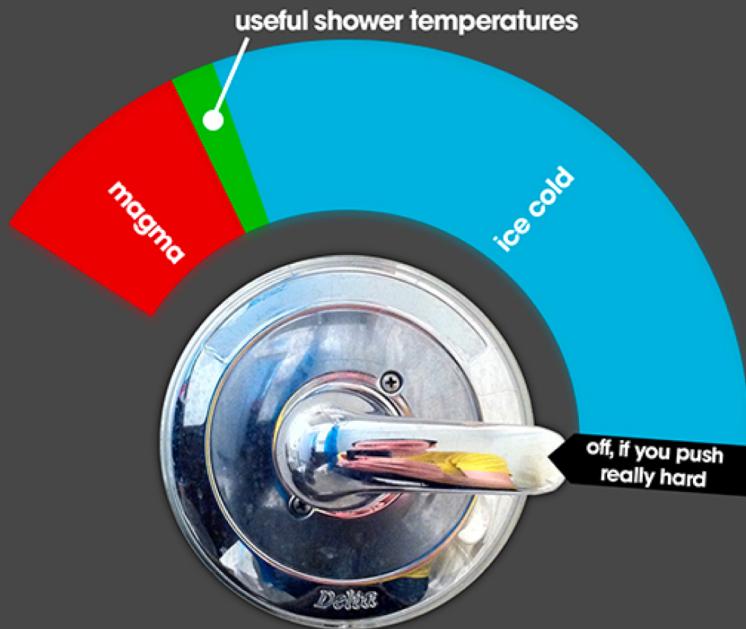
$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.75 & 1 & 0 & 0 \\ 0.5 & -0.285 & 1 & 0 \\ 0.25 & -0.428 & 0.334 & 0 \end{pmatrix} \quad U = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 0 & 1.75 & 2.25 & 4.25 \\ 0 & 0 & -0.86 & -0.29 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.75 & 1 & 0 & 0 \\ 0.5 & -0.285 & 1 & 0 \\ 0.25 & -0.428 & 0.334 & 1 \end{pmatrix} \quad U = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 0 & 1.75 & 2.25 & 4.25 \\ 0 & 0 & -0.86 & -0.29 \\ 0 & 0 & 0 & 0.67 \end{pmatrix}$$

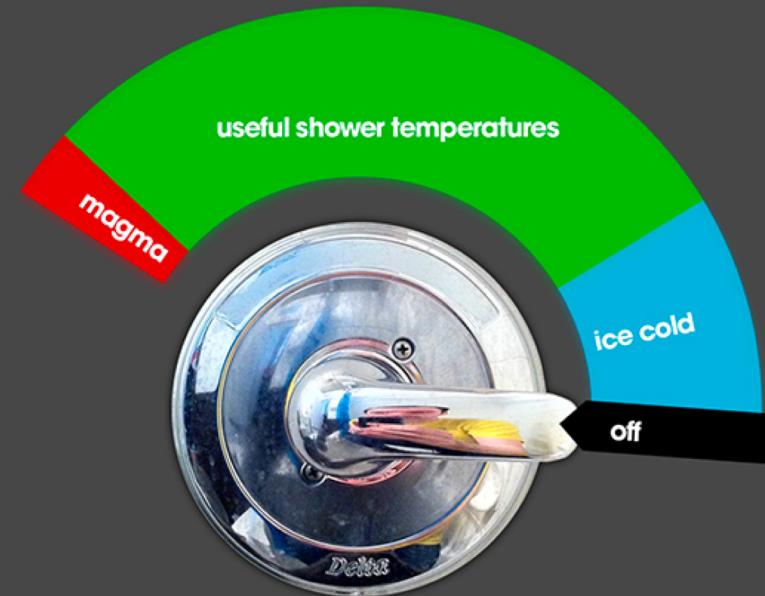
$$P = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

the shower faucet

how they are:



how they should be:



WHAT IT LOOKS LIKE



WHAT IT FEELS LIKE



Numerical experiments

Input has uncertainties:

- Errors due to representation with finite precision
- Error in the sampling

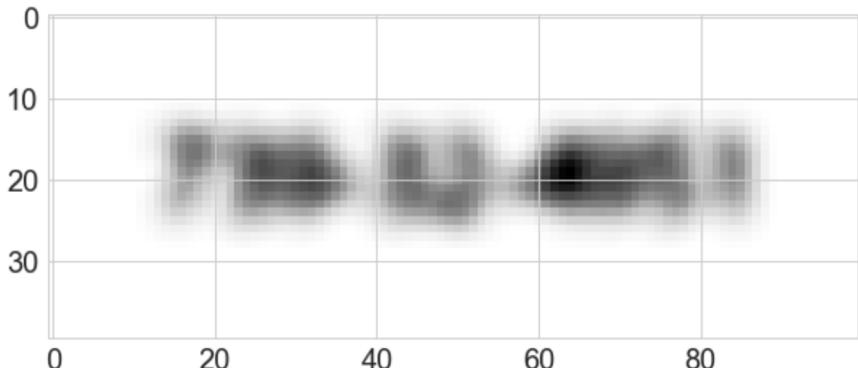
Once you select your numerical method , how much error should you expect to see in your **output**?

Is your method sensitive to errors (perturbation) in the input?

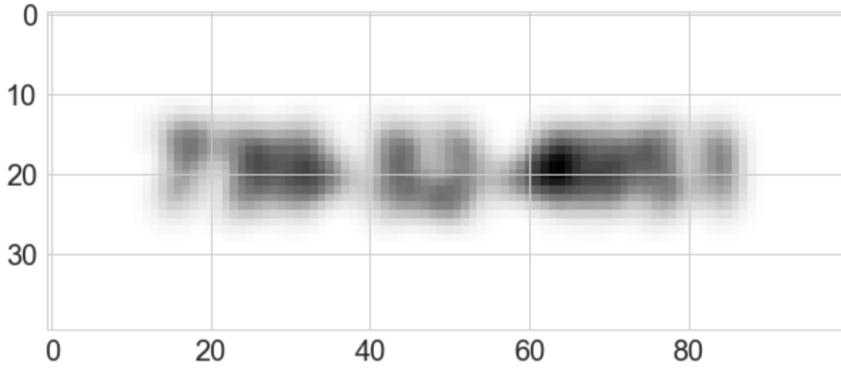
Demo “HilbertMatrix-ConditionNumber”

Demo “image-blur-inverse-v3”

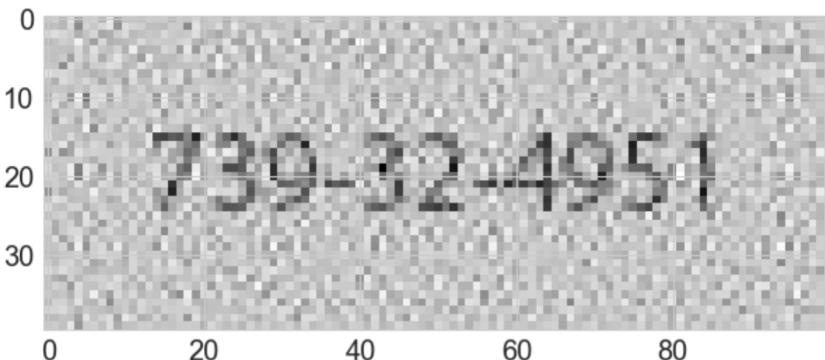
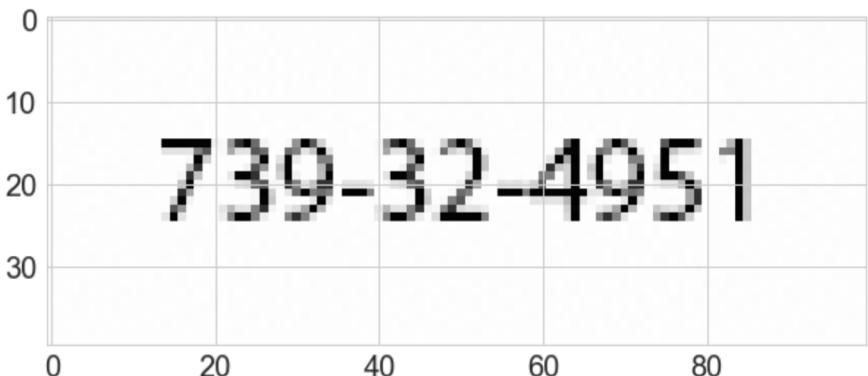
$$\mathbf{b} + a * 10^{-6} \ (a \in (0,1))$$



$$\mathbf{b} + a * 10^{-4} \ (a \in (0,1))$$



Solve $\mathbf{A} \mathbf{x} = \mathbf{b}$ for \mathbf{x}



Is your method sensitive to errors (perturbation) in the input?

How much noise can we add to the input data?

How can we define "little" amount of noise? Should be relative with the magnitude of the data.

Sensitivity of Solutions of Linear Systems

Suppose we start with a non-singular system of linear equations $\mathbf{A} \mathbf{x} = \mathbf{b}$.

We change the right-hand side vector \mathbf{b} (input) by a small amount $\Delta\mathbf{b}$.

How much the solution \mathbf{x} (output) changes, i.e., how large is $\Delta\mathbf{x}$?

Sensitivity of Solutions of Linear Systems

We can also add a perturbation to the matrix A (input) by a small amount E , such that

Condition number

The condition number is a measure of sensitivity of solving a linear system of equations to variations in the input.

Iclicker question

Give an example of a matrix that is very well-conditioned (i.e., has a condition number that is good for computation). Select the best possible condition number(s) of a matrix?

- A) $\text{cond}(\mathbf{A}) < 0$
- B) $\text{cond}(\mathbf{A}) = 0$
- C) $0 < \text{cond}(\mathbf{A}) < 1$
- D) $\text{cond}(\mathbf{A}) = 1$
- E) $\text{cond}(\mathbf{A}) = \text{large numbers}$

Condition number

$$\frac{\|\Delta \mathbf{x}\|}{\|\mathbf{x}\|} \leq \text{cond}(\mathbf{A}) \frac{\|\Delta \mathbf{b}\|}{\|\mathbf{b}\|}$$

Small condition numbers mean not a lot of error amplification. Small condition numbers are good!

The identity matrix should be well-conditioned:

$$\|\mathbf{I}\| = \max_{\|\mathbf{x}\|=1} \|\mathbf{I} \mathbf{x}\| = 1$$

It turns out that this is the smallest possible condition number:

$$\text{cond}(\mathbf{A}) = \|\mathbf{A}^{-1}\| \|\mathbf{A}\| \geq \|\mathbf{A}^{-1}\mathbf{A}\| = \|\mathbf{I}\| = 1$$

If \mathbf{A}^{-1} does not exist, then $\text{cond}(\mathbf{A}) = \infty$ (by convention)

Recall Induced Matrix Norms

$$\|A\|_1 = \max_j \sum_{i=1}^n |A_{ij}| \quad \text{Maximum absolute column sum of the matrix } A$$

$$\|A\|_\infty = \max_i \sum_{j=1}^n |A_{ij}| \quad \text{Maximum absolute row sum of the matrix } A$$

$$\|A\|_2 = \max_k \sigma_k$$

σ_k are the singular value of the matrix A

Iclicker question

Condition Number of a Diagonal Matrix

What is the 2-norm-based condition number of the diagonal matrix

$$A = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 13 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}?$$

- A) 1
- B) 50
- C) 100
- D) 200

About condition numbers

1. For any matrix A , $\text{cond}(A) \geq 1$
2. For the identity matrix I , $\text{cond}(I) = 1$
3. For any matrix A and a nonzero scalar γ , $\text{cond}(\gamma A) = \text{cond}(A)$
4. For any diagonal matrix D , $\text{cond}(D) = \frac{\max|d_i|}{\min|d_i|}$
5. The condition number is a measure of how close a matrix is to being singular: a matrix with large condition number is nearly singular, whereas a matrix with a condition number close to 1 is far from being singular
6. The determinant of a matrix is NOT a good indicator is a matrix is near singularity

Iclicker question

Condition Number of a Matrix

What's the 1-norm condition number of the following matrix?

$$\begin{bmatrix} 3 & 1 \\ 0 & 0.5 \end{bmatrix}$$

- A) 3
- B) 1.5
- C) 8
- D) 6.7

“Little c” demo

Discuss what happens when c is “close” to zero

Talk about eigenvalues

Talk about pivoting need when c is almost zero

Iclicker question

The need for pivoting depends on whether the matrix is singular.

- A) True
- B) False

Which of the following statements is correct?

Choice*

- A*) A singular matrix does not have a solution
- B*) A matrix is well conditioned if its condition number is less or equal to 1
- C*) A nonsingular matrix always has a solution
- D*) 1-norm of a matrix is the absolute column sum

Condition Number of Orthogonal Matrices

What is the 2-norm condition number of an orthogonal matrix A ?

$$\text{cond}(A) = \|A^{-1}\|_2 \|A\|_2 = \|A^T\|_2 \|A\|_2 = 1$$

That means orthogonal matrices have optimal conditioning.

They are very well-behaved in computation.

Residual versus error

Our goal is to find the solution \mathbf{x} to the linear system of equations $\mathbf{A} \mathbf{x} = \mathbf{b}$

Let us recall the solution of the perturbed problem

Iclicker question

When solving a system of linear equations via LU with partial pivoting, which of the following is guaranteed to be small?

A) Relative residual: $\frac{\|r\|}{\|A\|\|x\|}$

B) Relative error: $\frac{\|\Delta x\|}{\|x\|}$

C) Neither one of them

D) Both of them

Residual versus error

Let us first obtain the norm of the error:

Residual versus error

Without loss of generality, let us assume that the perturbed solution $\hat{\mathbf{x}}$ satisfies

$$(\mathbf{A} + \mathbf{E}) \hat{\mathbf{x}} = \mathbf{b}$$

Rule of thumb for conditioning

Suppose we want to find the solution \mathbf{x} to the linear system of equations $\mathbf{A} \mathbf{x} = \mathbf{b}$ using LU factorization with partial pivoting and backward/forward substitutions.

Suppose we compute the solution $\hat{\mathbf{x}}$.

If the entries in \mathbf{A} and \mathbf{b} are accurate to S decimal digits,

and $cond(\mathbf{A}) = 10^W$,

then the elements of the solution vector $\hat{\mathbf{x}}$ will be accurate to about

$$S - W$$

decimal digits

Iclicker question

Matrix Conditioning: Accurate digits

1 point

Let's say we want to solve the following linear system:

$$Ax = b$$

Assuming you are working with IEEE double precision floating point numbers, how many digits of accuracy will your answer have if $\kappa(A) = 1000$?

- A) 3
- B) 10
- C) 13
- D) 16
- E) 32