Lecture 3: Matrices, inverses, LU decomposition

Tuesday, September 1, 2015 9:30 AM

Admin: Homework 1 out

Reading: Meyer, Ch. 3 (skip 3.8)
[or Strang, Ch. 1, 2.1, 2.2]

Outline:

Matrices
Complexity of solving linear equations
Sparse matrices
Example: Medical imaging
LU decomposition of a matrix

MATRICES AND MATRIX MULTIPLICATION

matrix, dimensions, transpose:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \qquad A^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

a 2×3 real matrix (2 = # rows, 3 = # columns)

3×2 matrix

(i,j) entries

$A_{1,1} = 1$, $A_{1,2} = 2$, $A_{2,3} = 6$, ... $\quad (A^T)_{i,j} = A_{j,i}$

sum of two matrices with the same dimensions:

$$A + \begin{pmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} = \begin{pmatrix} 8 & 10 & 12 \\ 14 & 16 & 18 \end{pmatrix}$$

scalar multiple of a matrix:

$$2A = A \cdot 2 = \begin{pmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \end{pmatrix}, \quad 0 \cdot A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Matrix multiplication:

1. **Matrix-vector multiplication:**

$$\begin{pmatrix} 1 \\ \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ \end{pmatrix} \quad (1 \cdot 1 + 2 \cdot 2 + 3 \cdot 3)$$

1. Matrix-vector multiplication:

$$A \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 1\cdot 1 + 2\cdot 2 + 3\cdot 3 \\ 4\cdot 1 + 5\cdot 2 + 6\cdot 3 \end{pmatrix}$$

2×3 $\quad$ 3×1 $\quad$ 2×1

$$= \begin{pmatrix} 14 \\ 32 \end{pmatrix}$$

$$A \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1\cdot 1 + 2\cdot 0 + 3\cdot 0 \\ 4\cdot 1 + 5\cdot 0 + 6\cdot 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 4 \end{pmatrix}$$ first column

$$A \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \end{pmatrix} \qquad A \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix}$$ 3rd column

↑ picks out 2nd column

$\Rightarrow$ By linearity,

$$A \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = A \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + A \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix} + A \begin{pmatrix} 0 \\ 0 \\ 3 \end{pmatrix}$$

$$= \begin{pmatrix} 1 \\ 4 \end{pmatrix} + 2 \cdot \begin{pmatrix} 2 \\ 5 \end{pmatrix} + 3 \cdot \begin{pmatrix} 3 \\ 6 \end{pmatrix}$$

$$= \begin{pmatrix} 14 \\ 32 \end{pmatrix} \checkmark$$

Interpretation:

$m \times n$ matrix $\longleftrightarrow$ function n-dimensional vectors to m-dim vectors

$$1^{st} \text{ column of } A = A \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$2^{nd} \text{ column of } A = A \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

"linearity":

$$A \begin{pmatrix} 2 \\ 3 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = 2 \cdot (1^{st} \text{column}) + 3 \cdot (2^{nd} \text{ column})$$

Example:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \quad \vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad \vec{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

$$A\vec{x} = \begin{pmatrix} x_1 + 2x_2 + 3x_3 \\ 4x_1 + 5x_2 + 6x_3 \end{pmatrix}$$

$\Rightarrow A\vec{x} = \vec{b}$ is equivalent to

$$\begin{cases} x_1 + 2x_2 + 3x_3 = b_1 \\ \end{cases}$$

$$\begin{cases} x_1 + 2x_2 + 3x_3 = b_1 \\ 4x_1 + 5x_2 + 6x_3 = b_2 \end{cases}$$

2. Matrix-matrix multiplication:

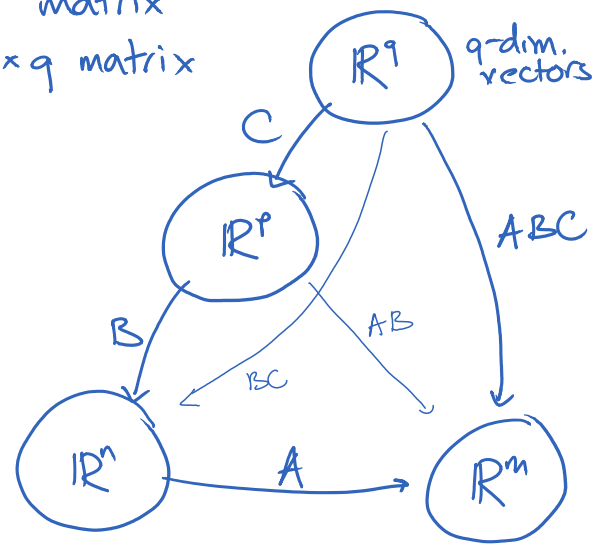
A : $m \times n$ matrix

B : $n \times p$ matrix

AB : $m \times p$ matrix — first apply B, then A (right to left)

C : $p \times q$ matrix

ABC : $m \times q$ matrix



$\Rightarrow (AB)C = A(BC)$ "associativity"

Easy rule: across and down

$$\underset{2\times3}{A} \cdot \underset{3\times2}{A^T} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

$$= \begin{pmatrix} 1\cdot1+2\cdot2+3\cdot3 & 1\cdot4+2\cdot5+3\cdot6 \\ 4\cdot1+5\cdot2+6\cdot3 & 4\cdot4+5\cdot5+6\cdot6 \end{pmatrix}$$

$$= \begin{pmatrix} 14 & 32 \\ 32 & 77 \end{pmatrix}$$

$$A^TA = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} = \begin{pmatrix} 1\cdot1+4\cdot4 & 1\cdot2+4\cdot5 & 1\cdot3+4\cdot6 \\ 2\cdot1+5\cdot4 & 2\cdot2+5\cdot5 & \ddots \end{pmatrix}$$

$$= \begin{pmatrix} 17 & 22 & 27 \\ 22 & 29 & 36 \\ 27 & 36 & 45 \end{pmatrix}$$

$$= \begin{pmatrix} 17 & 22 & 27 \\ 22 & 29 & 36 \\ 27 & 36 & 45 \end{pmatrix}$$

Formally $\underset{m\times n\ n\times p}{(AB)_{i,j}} = \sum_{k=1}^{n} A_{i,k} B_{k,j}$

Note: Matrix-vector multiplication is a special case

$$\underset{2\times 2}{\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}} \underset{2\times 1}{\begin{pmatrix} 5 \\ 6 \end{pmatrix}} = \underset{2\times 1}{\begin{pmatrix} 17 \\ 39 \end{pmatrix}}$$

Why is this the right rule?

$$\underset{A}{\begin{pmatrix} a & b \\ c & d \end{pmatrix}} \underset{B}{\begin{pmatrix} e & f \\ g & h \end{pmatrix}} = \begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix}$$

$(AB)\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ = first column of AB

$= A$(first column of B)

$= A\begin{pmatrix} e \\ g \end{pmatrix}$

$=$ (first col. of A)$\cdot e +$ (2^{nd} col. of A)$\cdot g$ ✓

Observation 1: In general, $AB \neq BA$.

"matrix multiplication is not commutative"

Observation 2: $(AB)^T = B^T A^T$

Proof:

$$\left((AB)^T\right)_{i,j} = (AB)_{j,i}$$

$$= \sum_k A_{jk} B_{ki}$$

$$= \sum_k (B^T)_{ik} (A^T)_{kj}$$

$$= (B^T A^T)_{i,j}$$ ✓ □

Corollary: AA^T is symmetric

(equals its own transpose).

$(AA^T)^T = A^{TT}A^T = AA^T$ ✓

Examples: Diagonal matrices, block matrices, Matlab

A. Diagonal matrices

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}\begin{pmatrix} 4 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 6 \end{pmatrix} = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 18 \end{pmatrix}$$

product of diagonal, square matrices is diagonal

$$= \begin{pmatrix} 4 & & \\ & 5 & \\ & & 6 \end{pmatrix}\begin{pmatrix} 1 & & \\ & 2 & \\ & & 3 \end{pmatrix}$$ —and they commute!

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} a & 2b & 3c \\ d & 2e & 3f \\ g & 2h & 3i \end{pmatrix}$$

scaling the columns (since the diagonal matrix acts first)

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = \begin{pmatrix} a & b & c \\ 2d & 2e & 2f \\ 3g & 3h & 3i \end{pmatrix}$$

scaling the rows (since the diagonal matrix here acts second)

B. Identity matrix

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

$I\vec{v} = \vec{v}$ for any vector $\vec{v}$ of appropriate dimensions

$IA = A$ " matrix A "

$AI = A$

C. Permutation matrices

Definition: An $n \times n$ (square) matrix is a permutation matrix if every row and every column has exactly one 1 in it, and every

other matrix entry is 0.

Examples:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$ (crossed out)

Why? Multiplying a vector by a permutation matrix permutes its entries, eg.,

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} c \\ a \\ b \end{pmatrix} \checkmark$$

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = \begin{pmatrix} g & h & i \\ a & b & c \\ d & e & f \end{pmatrix} \checkmark$$

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} b & c & a \\ e & f & d \\ h & i & g \end{pmatrix} \checkmark$$

easy proof: follow $e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, e_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, e_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$

Example:

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

What is P^{-1}?

Answer:

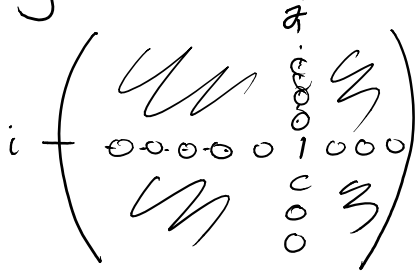
P sends $1 \to 4$, $2 \to 1$, $3 \to 2$, $4 \to 3$. So P^{-1} sends $1 \to 2$, $2 \to 3$, $3 \to 4$, $4 \to 1$.

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

P^{-1} is just the transpose P^T

(Why? Say that P has a 1 at position (i,j). It sends vector i ...

Why: say that P has a 1 at position (i,j).



It sends position j to position i. P^{-1} has to do the opposite.

C. Block matrices

$$A = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

What is AB?

Answer: Of course, you can multiply it out...

Or, break A and B into pieces:

$$A = \begin{pmatrix} I & C \\ C & I \end{pmatrix} \qquad B = \begin{pmatrix} C & I \\ I & C \end{pmatrix} \qquad I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad C = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

and multiply in blocks (since the block dimensions match):

$$AB = \begin{pmatrix} IC+CI & I^2+C^2 \\ C^2+I^2 & CI+IC \end{pmatrix}$$

$$= \begin{pmatrix} 2C & I+C^2 \\ I+C^2 & 2C \end{pmatrix} \qquad C^2 = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix} = 2C$$

$$= \begin{pmatrix} 2 & 2 & 3 & 2 \\ 2 & 2 & 2 & 3 \\ 3 & 2 & 2 & 2 \\ 2 & 3 & 2 & 2 \end{pmatrix}_{4\times 4} \checkmark$$

D. Matlab:

```
A=[1 0 1 1;
   0 1 1 1;
   1 1 1 0;
   1 1 0 1]
A'      <- transpose
A*A'    <- matrix product
b=[1; 2; 3; 4];
```

```
b = [1 2 3 4]';
A * b
A^2      ← matrix power

I = eye(2)
C = ones(2,2)
A = [I C; C I]   ← block matrices are okay
B = [C I; I C]
A * B
```

Aside:

Matrix multiplication in practice: Strassen's algorithm

http://en.wikipedia.org/wiki/Strassen_algorithm

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Compute

$$M_1 = (A_{11}+A_{22})(B_{11}+B_{22}), \quad M_5 = (A_{11}+A_{12})B_{22}$$
$$M_2 = (A_{21}+A_{22})B_{11} \quad M_6 = (A_{21}-A_{11})(B_{11}+B_{12})$$
$$M_3 = A_{11}(B_{12}-B_{22}) \quad M_7 = (A_{12}-A_{22})(B_{21}+B_{22})$$
$$M_4 = A_{22}(B_{21}-B_{11})$$

using 7 matrix multiplications (instead of the 8 obvious ones $A_{11}B_{11}+A_{12}B_{21}, \ldots, A_{21}B_{12}+A_{22}B_{22}$).

$$\Rightarrow AB = \begin{pmatrix} M_1+M_4-M_5+M_7 & M_3+M_5 \\ M_2+M_4 & M_1-M_2+M_3+M_6 \end{pmatrix}$$

$$(A_{21}B_{11}+\cancel{A_{22}B_{11}})+(A_{22}B_{21}-\cancel{A_{22}B_{11}}) \checkmark$$

Asymptotic complexity $O(n^{\log_2 7 = 2.81\ldots})$ for $n \times n$ matrices.

Back to linear equations: Solve $A\vec{x} = \vec{b}$ for $\vec{x}$

Trivia: What is the fastest way of solving $Ax = b$?

In practice: Matlab (not Mathematica) or roll-your-own

In theory: for a general $n \times n$ matrix A (ie., n equations, n unknowns)

the same as matrix multiplication

http://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations#Matrix_algebra

$O(n^3)$ naïve algorithm $(AB)_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$

$O(n^{2.376})$ [Coppersmith-Winograd '90]

$O(n^{2.373})$ [V. Williams 2012]

meaning $c \cdot n^{2.373}$ for some constant c

- unfortunately, the constant is too large for practical use!

Conjecture: Perhaps the exponent is ultimately 2??

Sparse matrices

Recall: Last time we solved numerically the differential equation

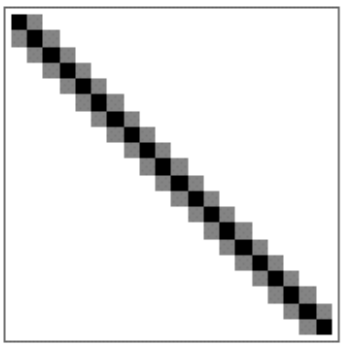
$$\begin{cases} f''(t) = \sqrt{t} \\ f(0) = f(1) = 0 \end{cases}$$

by discretizing the interval $[0,1]$.

$x_j = f(\frac{j}{n})$

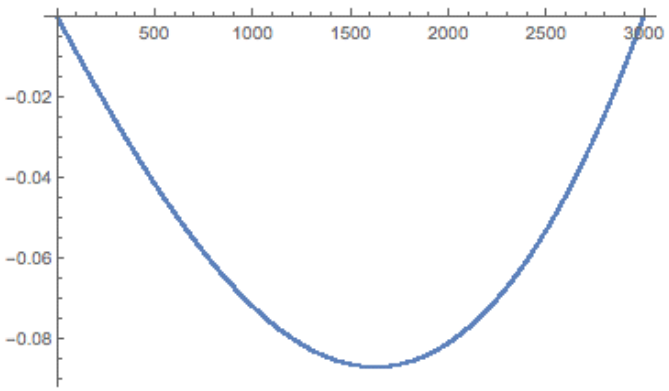
$$x_{j+1} - 2x_j + x_{j-1} = \frac{1}{n^2}\sqrt{\frac{j}{n}}$$

```
n = 3000;

b = Table[1/n^2 Sqrt[j/n], {j, 1, n - 1}] // N;

A = SparseArray[Join[
    Table[{j, j} → -2, {j, 1, n - 1}],
    Table[{j, j + 1} → 1, {j, 1, n - 2}],
    Table[{j + 1, j} → 1, {j, 1, n - 2}]
   ]];

ArrayPlot[A[[ ;; 20, ;; 20]]]
```



```
Timing[
 x = LinearSolve[A, b];
]
{0.002488, Null}

Afull = Normal[A];   ← converts A to a dense matrix
Timing[
 x = LinearSolve[Afull, b];
]
{2.21352, Null}

ListPlot[x]
```

sparse matrix solving 1000x faster!!!



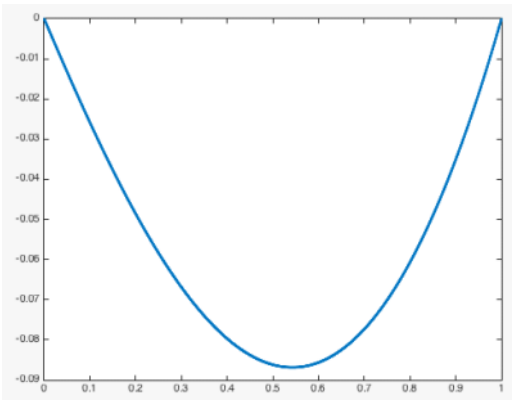
Matlab commands

```
>> n = 3000;

b = 1/n^2.5 * sqrt(1:n-1)';

A = spdiags(ones(n-2,1),-1,n-1,n-1);
A = A + A' - 2*spdiags(ones(n-1,1),0,n-1,n-1);
full(A(1:9,1:9))

size(A)

ans =

    -2     1     0     0     0     0     0     0     0
     1    -2     1     0     0     0     0     0     0
     0     1    -2     1     0     0     0     0     0
     0     0     1    -2     1     0     0     0     0
     0     0     0     1    -2     1     0     0     0
     0     0     0     0     1    -2     1     0     0
     0     0     0     0     0     1    -2     1     0
     0     0     0     0     0     0     1    -2     1
     0     0     0     0     0     0     0     1    -2


ans =

        2999        2999
```

```
>> tic
x = A \ b;
toc
Elapsed time is 0.010280 seconds.
>> Afull = full(A);
tic
x = Afull \ b;
toc
Elapsed time is 0.999726 seconds.
```

converts A to a dense matrix

```
>> plot(1/n*(1:n-1),x,'.');
axis([0 1 -.1 0]);
```



Note: $A = \text{sparse}([i_1, i_2, i_3], [j_1, j_2, j_3], [s_1, s_2, s_3])$
creates a sparse matrix with $a_{i_1 j_1} = s_1$, $a_{i_2 j_2} = s_2$, $a_{i_3 j_3} = s_3$.

```
n = 100;
A1 = sparse(1:n-1, 1:n-1, -2);
A2 = sparse(2:n-1, 1:n-2, 1, n-1,n-1);
A3 = sparse(1:n-2, 2:n-1, 1, n-1,n-1);
A = A1+A2+ A3;
A
spy(A, 'o', 1);
```

range $[1,2,3,\ldots,n-1]$

rows, columns

creates $(n-1)\times(n-1)$ sparse matrix with -2 on diagonal

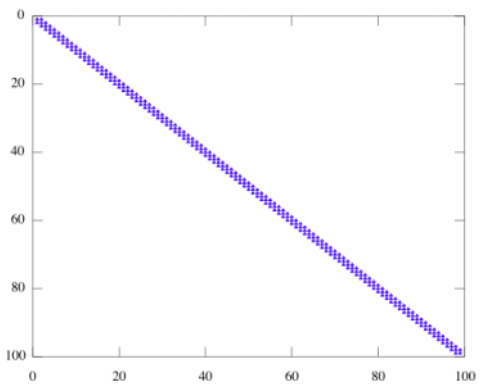
A2 has $+1$'s just below the diagonal

A3 has $+1$'s just above the diagonal

make it an $(n-1)\times(n-1)$ matrix instead of $(n-2)\times(n-1)$

tridiagonal

"spy" command to visualize the matrix



to create this tridiagonal matrix, you can equivalently use:

```
e = ones(n,1);
A = spdiags([e, -2*e, e], -1:1, n, n);
```

transpose to get column vector

```
b = n^(-2.5) * (1:n-1)'.^ (1/2);
```

.^ gives element-wise exponentiation

```
y = A\b;
```

solve for y

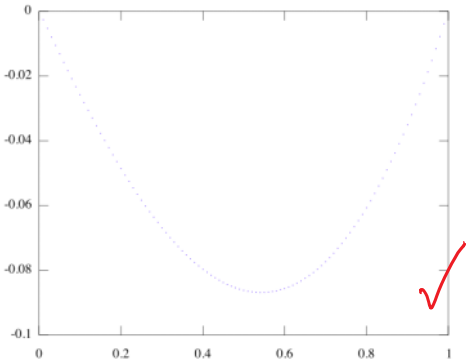
```
b = n^(-2.5) * (1:n-1)'.^(1/2);
y = A\b;
plot((1:n-1)/100, y, '.');
```

Solve for y!

.^ gives element-wise exponentiation

plot marker

x-coordinates y-coords



Today: How to solve the SAME system of linear equations repeatedly (with different right-hand sides)

Method 1: Gauss-Jordan in parallel

Example: (from last time)

Let $A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}$.

Solve $A\vec{x} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, $A\vec{y} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$, $A\vec{z} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$

Answer: In condensed form,

$$\left(\begin{array}{ccc|ccc} 2 & -1 & 0 & 1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 & 1 \end{array}\right) \xrightarrow{-\frac{1}{2}} \left(\begin{array}{ccc|ccc} 2 & -1 & 0 & 1 & 0 & 0 \\ 0 & \frac{3}{2} & -1 & \frac{1}{2} & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 & 1 \end{array}\right) \xrightarrow{-\frac{2}{3}}$$

r.h.s. for 1st system; 2nd system; 3rd system

$$\longrightarrow \left(\begin{array}{ccc|ccc} 2 & -1 & 0 & 1 & 0 & 0 \\ 0 & \frac{3}{2} & -1 & \frac{1}{2} & 1 & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{2}{3} & 1 \end{array}\right) \uparrow 3$$

$$\longrightarrow \left(\begin{array}{ccc|ccc} 2 & -1 & 0 & 1 & 0 & 0 \\ 0 & \frac{3}{2} & 0 & \frac{3}{2} & 3 & 3 \\ 0 & 0 & 1 & 1 & 2 & 3 \end{array}\right) \uparrow \frac{2}{3}$$

divide

$$\left(\begin{array}{ccc|ccc} 0 & 0 & 1 & 1 & 2 & 3 \end{array}\right)$$

$$\longrightarrow \left(\begin{array}{ccc|ccc} 2 & 0 & 0 & 2 & 2 & 2 \\ 0 & 1 & 0 & 1 & 2 & 2 \\ 0 & 0 & 1 & 1 & 2 & 3 \end{array}\right) \quad \text{2 divide by 2}$$

$$\longrightarrow \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 2 & 2 \\ 0 & 0 & 1 & 1 & 2 & 3 \end{array}\right) \quad \begin{matrix} \vec{x} & \vec{y} & \vec{z} \end{matrix}$$

First row: $1 \cdot x_1 + 0 \cdot x_2 + 0 \cdot x_3 = 1$

$y_1 = 1, \; z_1 = 1$

2nd row: $0 \cdot x_1 + 1 \cdot x_2 + 0 \cdot x_3 = 1$

$y_2 = 2, \; z_2 = 2$

3rd row: $x_3 = 1, \; y_3 = 2, \; z_3 = 3$

$$\Rightarrow \boxed{\vec{x} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \; \vec{y} = \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix}, \; z = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}}$$

Alternatively, in terms of matrices

$A\vec{x}$ = first column on right has been transformed to

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{I} \vec{x} = \text{first column on right} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\Rightarrow \vec{x} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \checkmark$$

Alternatively, in terms of matrices, the 3 equations

$$A\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \; A\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \; A\begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

are equivalent to the single equation

$$\underset{3\times 3}{A}\underbrace{\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix}}_{X} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{B}$$

and we need to solve for the matrix X in

$$AX = B.$$

Gauss-Jordan row elimination simplified this to

$$\begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} \end{pmatrix}$$

Gauss-Jordan row elimination simplified this to

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} X = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \end{pmatrix}$$

$$\Rightarrow X = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \end{pmatrix} \checkmark$$

Method 2: Matrix inversion

• Definition • Existence • Properties

Definition: If A is an $n \times n$ square matrix, then A^{-1} is the matrix satisfying

$$A \cdot A^{-1} = I,$$

if such a matrix exists.

Examples:

$$I^{-1} = I$$

$$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & -2 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}^{-1} = \text{does not exist!}$$

Exercise: Prove that for a general 2×2 matrix

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

then A^{-1} exists $\Leftrightarrow ad - bc \neq 0$.

If $ad - bc \neq 0$,

$$A^{-1} = \frac{1}{ad-bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

Application: Solving linear systems of equations

To solve $A\vec{x} = \vec{b}$:

① Compute A^{-1} — $O(n^3)$ time

② Let $\vec{x} = A^{-1}\vec{b}$. — $O(n^2)$ time

$\Rightarrow A\vec{x} = AA^{-1}b$

$= Ib$

$= b$ ✓

↘ After computing A^{-1} once, solving new equations $A\vec{y} = \vec{c}$, $A\vec{z} = \vec{d}$ very fast!

$= b$ ✓

How to compute A^{-1}? We already did!

Use Gauss-Jordan to solve matrix equation

$$AX = I$$

e.g.,

$$\left(\begin{array}{ccc|ccc} 2 & -1 & 0 & 1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 & 1 \end{array}\right) \longrightarrow \cdots \longrightarrow \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 2 & 2 \\ 0 & 0 & 1 & 1 & 2 & 3 \end{array}\right)$$

A $\qquad$ A^{-1}

Properties of the matrix inverse:

- $AA^{-1} = I = A^{-1}A$

  Equivalently, $(A^{-1})^{-1} = A$

- If it exists, then A^{-1} is unique.

  Proof:

  Say X and Y are both inverses of A.

  $$X = XI = X(AY)$$
  $$= (XA)Y$$
  $$= IY$$
  $$= Y \quad \checkmark$$

- If A and B are invertible, so is AB:

  $$(AB)^{-1} = B^{-1}A^{-1}$$

  Since $(B^{-1}A^{-1})AB = B^{-1}IB = I$ ✓

  If A, B, C are invertible, so is ABC:

  $$(ABC)^{-1} = C^{-1}B^{-1}A^{-1}.$$

- Not all matrices are invertible!

But: Matrix inversion is not the answer we want.

In practice, we almost never invert matrices!

① Too slow. ② Might not even exist.

③ Numerically unstable

Example:

```
>> A = [.01 0; 0 1];
inv(A)

ans =

   100     0
     0     1
```

```
>> A = [.02 0; 0 1];
inv(A)

ans =

    50     0
     0     1
```

④ Can take **sparse** matrices to **dense** matrices
(⇒ uses too much memory!)

Example:

```
>> n = 10;
e = ones(n-1,1);
A = -2*eye(n) + diag(ones(n-1,1), -1) + diag(ones(n-1,1), 1)

A =

    -2     1     0     0     0     0     0     0     0     0
     1    -2     1     0     0     0     0     0     0     0
     0     1    -2     1     0     0     0     0     0     0
     0     0     1    -2     1     0     0     0     0     0
     0     0     0     1    -2     1     0     0     0     0
     0     0     0     0     1    -2     1     0     0     0
     0     0     0     0     0     1    -2     1     0     0
     0     0     0     0     0     0     1    -2     1     0
     0     0     0     0     0     0     0     1    -2     1
     0     0     0     0     0     0     0     0     1    -2

>> inv(A)

ans =

   -0.9091   -0.8182   -0.7273   -0.6364   -0.5455   -0.4545   -0.3636   -0.2727   -0.1818   -0.0909
   -0.8182   -1.6364   -1.4545   -1.2727   -1.0909   -0.9091   -0.7273   -0.5455   -0.3636   -0.1818
   -0.7273   -1.4545   -2.1818   -1.9091   -1.6364   -1.3636   -1.0909   -0.8182   -0.5455   -0.2727
   -0.6364   -1.2727   -1.9091   -2.5455   -2.1818   -1.8182   -1.4545   -1.0909   -0.7273   -0.3636
   -0.5455   -1.0909   -1.6364   -2.1818   -2.7273   -2.2727   -1.8182   -1.3636   -0.9091   -0.4545
   -0.4545   -0.9091   -1.3636   -1.8182   -2.2727   -2.7273   -2.1818   -1.6364   -1.0909   -0.5455
   -0.3636   -0.7273   -1.0909   -1.4545   -1.8182   -2.1818   -2.5455   -1.9091   -1.2727   -0.6364
   -0.2727   -0.5455   -0.8182   -1.0909   -1.3636   -1.6364   -1.9091   -2.1818   -1.4545   -0.7273
   -0.1818   -0.3636   -0.5455   -0.7273   -0.9091   -1.0909   -1.2727   -1.4545   -1.6364   -0.8182
   -0.0909   -0.1818   -0.2727   -0.3636   -0.4545   -0.5455   -0.6364   -0.7273   -0.8182   -0.9091
```

Dense versus sparse matrices

```
>> n = 10;
A = zeros(n,n);
for i = 1:9
  A(i,i+1) = i;
end
A
```

```
A =

     0     1     0     0     0     0     0     0     0     0
     0     0     2     0     0     0     0     0     0     0
     0     0     0     3     0     0     0     0     0     0
     0     0     0     0     4     0     0     0     0     0
     0     0     0     0     0     5     0     0     0     0
     0     0     0     0     0     0     6     0     0     0
     0     0     0     0     0     0     0     7     0     0
     0     0     0     0     0     0     0     0     8     0
     0     0     0     0     0     0     0     0     0     9
     0     0     0     0     0     0     0     0     0     0
```

```
>> A = sparse(n,n);
for i = 1:9
  A(i,i+1) = i;
end
A
```

```
A =

   (1,2)        1
   (2,3)        2
   (3,4)        3
   (4,5)        4
   (5,6)        5
   (6,7)        6
   (7,8)        7
   (8,9)        8
   (9,10)       9
```

instead of storing all matrix entries, just keep the nonzero ones
- uses less memory
- allows for **much** faster computations

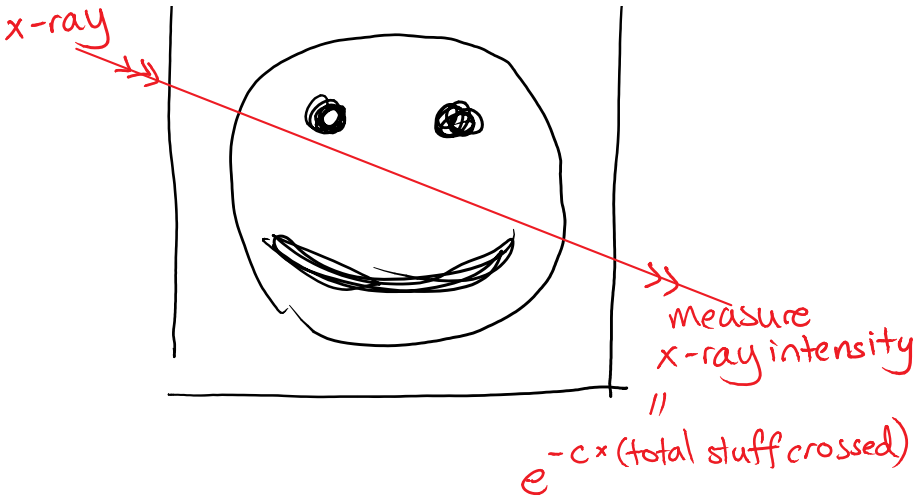
```
>> full(A)

ans =

     0     1     0     0     0     0     0     0     0     0
     0     0     2     0     0     0     0     0     0     0
     0     0     0     3     0     0     0     0     0     0
     0     0     0     0     4     0     0     0     0     0
     0     0     0     0     0     5     0     0     0     0
     0     0     0     0     0     0     6     0     0     0
     0     0     0     0     0     0     0     7     0     0
     0     0     0     0     0     0     0     0     8     0
     0     0     0     0     0     0     0     0     0     9
     0     0     0     0     0     0     0     0     0     0
```

EXAMPLE: MEDICAL IMAGING

(@USC: Justin Haldar, Krishna Nayak, ...)

Problem: Given scanning data,
Output reconstructed image

Example: "Computed tomography" (CT) scan:



Simplification:

- Discretize to an $n \times n$ grid (n^2 variables)
- For each (discrete) line, you are given $\exp(-c \times \text{sum of pixels along line})$.

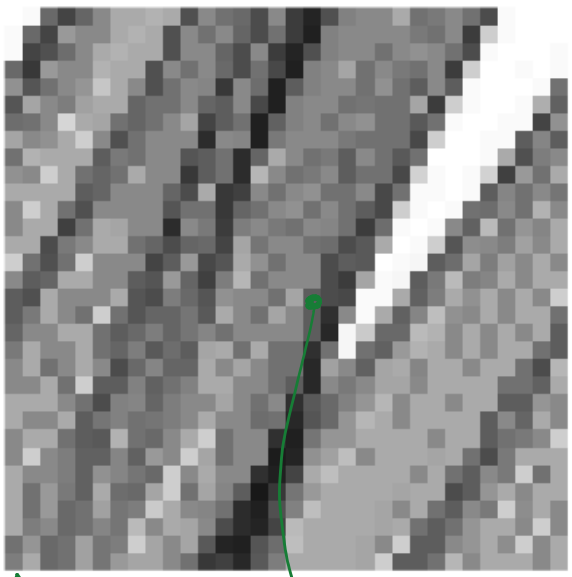
$\therefore$ Each discrete line gives one equation:

$$\sum_{\text{points } p \text{ in the line}} (\text{pixel value at } p) = \text{observed sum}$$

Goal: Solve for all the pixel values.

In Matlab:

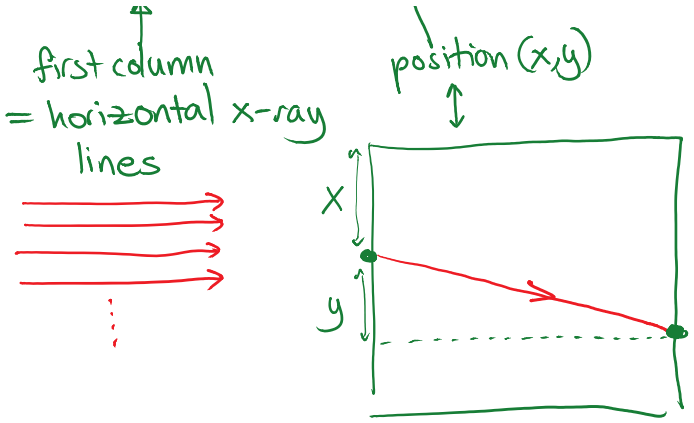
```
octave:1> load('xraydata.mat');
octave:2> size(xraydata)
ans =

   32   32

octave:3> imshow(xraydata);
```



How is this data given?

first column

position (x,y)




```
% output the (x,y) coordinates of a line that starts at
%  (x=0,y=ystart) and ends at (x=n-1,y=ystart+ychange mod n),
% wrapping around vertically
function coords = lineCoordinates(ystart, ychange, n)
  slope = ychange / (n-1);
  coords = [0:n-1 ; ystart + round(slope * (0:n-1)) ]';
  coords(:,2) = mod(coords(:,2), n);  % wrap coordinates around mod n
end
```

```
octave:6> coords = lineCoordinates(4,3,6)
coords =
   x  y
   0  4
   1  5
   2  5
   3  0
   4  0
   5  1
```

```
n = size(xraydata)(1)    % okay in Octave, but not Matlab
logxraydata = -log(xraydata);
b = reshape(logxraydata, n^2);  % change the n x n matrix into an n^2-long vector
```

Aside: Changing a matrix to a vector using reshape()

```
octave:8> reshape([1 2; 3 4], 4)
ans =

   1
   3
   2
   4
```

```
% takes x and y both from 0 to n-1
% returns an integer j from 0 to n^2-1
function j = xytoj(x,y, n)
  j = x + n * y;
end

% simple test code:
n = 2;
for y = 0:n-1
  for x = 0:n-1
    [x, y, xytoj(x,y,n)]
  end
end
```

→

```
ans =
   0   0   0

ans =
   1   0   1

ans =
   0   1   2

ans =
   1   1   3
```

(this gives the same index ordering that the reshape() function uses)

Next, we need to set up the equations. Each equation, corresponding to a line, will be one row of a sparse matrix A.

```
m = n^2;                % m = number of equations, in n^2 unknowns
A = sparse(m, n^2);     % initialize all-zeros sparse matrix
eqn = 1;
for ystart = 0:n-1
  for ychange = 0:n-1
    % for each line, we'll have one equation
    line = lineCoordinates(ystart, ychange, n);
    for i = 1:n
      j = 1 + xytoj(line(i,1), line(i,2), n); % +1 so index goes from 1 to n^2
      A(eqn, j) = 1;  % the coefficient is +1 on every pixel in the line
    end
    eqn += 1;   % move to the next equation (row of A)
  end;
end;
```

Let's solve these equations!

```
x = A\b;
recoveredimage = reshape(x, n, n);
imshow(recoveredimage);
```

That didn't work.

If we run the same code with n=2, we can see the problem:

```
octave:26> full(A)
ans =

   1   1   0   0
   1   0   0   1
   0   0   1   1
   0   1   1   0
```

← These $4 = n^2$ equations are not independent:

row 4 = row 1 + row 3 − row 2

Therefore we have **fewer** independent equations than unknowns, and there are **infinitely many** solutions. Matlab found one solution, not the one we wanted.

To fix this, we need to add more equations. Now if you look in the upper-left corner of xraydata, the pixel is pure white. This means that the total mass of bone crossed by the first horizontal x-ray is 0. There is no bone in the first row. We can add equations for this:

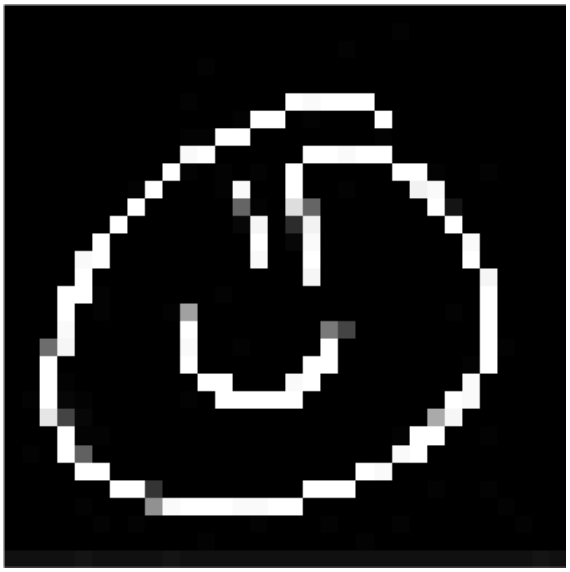
```
for k = 1:n
  A(eqn,k) = 1;
  eqn += 1;
end
b = [b ; zeros(n,1)];
```

Why couldn't Matlab already see this? It doesn't know that bone densities must be ≥ 0, so there could be negative values cancelling out positive values along the first row.

Solving again:

```
x = A\b;
x = x / max(x);
recoveredimage = reshape(x, n, n);
imshow(recoveredimage);
```

← normalize here because we don't know the attenuation factor of bone



Perfect!

Note that sparse matrix routines are faster than using dense matrices:

```
octave:42> t = cputime;
octave:43> x = A\b;
octave:44> elapsedtime = cputime - t
elapsedtime =  0.66671 seconds
octave:45>
octave:45> Afull = full(A);
octave:46> t = cputime;
octave:47> x = Afull\b;
octave:48> elapsedtime = cputime - t
elapsedtime =  1.3295 seconds
```

← convert to dense representation

One moral: Setting up the equations can be painful! Matlab is fast at solving them — but doesn't warn you if there are infinitely many solutions.

Some obvious research problems:

1. How to solve the equations faster?
   (Eg., what if we want to reconstruct live video? .....)
2. How can we collect the data more quickly?

2. How can we collect the data more quickly?
Fewer equations $\longleftrightarrow$ fewer x-rays
$\longleftrightarrow$ faster, less radiation exposure

So can we use fewer equations?
No.
Then the solution will not be unique.
But maybe that is okay ..... !!!
"Compressed sensing"
Candes, Caltech; Romberg, Caltech; Tao, UCLA 2006 ...

LU DECOMPOSITION OF A MATRIX

Problem: How can we solve the equations faster?
Given a matrix A,
and vectors b, b', b'', ...,
Goal: Solve quickly

$$Ax = b$$
$$Ax' = b'$$
$$Ax'' = b''$$
$$\vdots$$

same matrix A; different vectors b

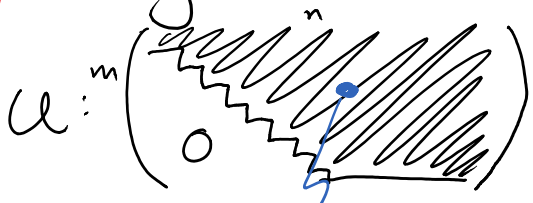
First idea: Precompute A^{-1} (if it exists).
Then just multiply
$x = A^{-1}b$, $x' = A^{-1}b'$, $x'' = A^{-1}b''$, ...
But there's a better way...

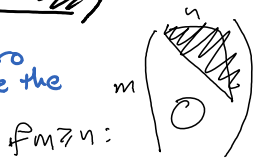
Theorem [Turing 1948]:
Any $m \times n$ matrix A can be factored as

$$\boxed{A = PLU}$$

$$A = PLU$$

where P is an $m \times m$ permutation matrix,
L is an $m \times m$ lower-triangular matrix
U is an $m \times n$ upper-triangular matrix.





Why? We'll see in a moment. First, an application:
Solving faster multiple sets of equations with one A

To solve $Ax = b$

$$PLUx = b,$$

solve $Ly = P^{-1}b$ for y, then $Ux = y$ for x.
For an $n \times n$ matrix, this is fast! $O(n^2)$

by forward substitution
(get y_1, substitute, $y_2, y_3, \ldots$)

by back substitution
(get $x_n, x_{n-1}, x_{n-2}, \ldots$)

∴ Precomputing the LU decomposition of A allows for solving $Ax = b$, $Ax' = b'$, $Ax'' = b''$, ... faster.

(In fact, if A is tridiagonal, L and U will be sparse, while A^{-1} can be dense ⟶ even given A^{-1} for free, it is faster to solve $Ax = b$ using the LU decomposition than to multiply by A^{-1}!)

[Do it in Mathematica.]

Another (less interesting) application: Computing Det(A)

$$\det(A) = \det(P)\cdot\det(L)\cdot\det(U)$$

Theorem [Turing 1948]:

Any $m \times n$ matrix A can be factored as

$$\boxed{A = PLU}$$

where P is an $m \times m$ permutation matrix,
L is an $m \times m$ lower-triangular matrix
U is an $m \times n$ upper-triangular matrix.

constructive

Proof idea: Apply Gaussian elimination to A.

$$A = \begin{pmatrix} 2 & 2 & 2 \\ 4 & 7 & 7 \\ 6 & 18 & 22 \end{pmatrix} \begin{matrix} \\ -2 \\ -3 \end{matrix}$$

$$\rightarrow \underbrace{\begin{pmatrix} 2 & 2 & 2 \\ 0 & 3 & 3 \\ 6 & 18 & 22 \end{pmatrix}}_{A_1} \begin{matrix} \\ \\ -3 \end{matrix} \rightarrow \underbrace{\begin{pmatrix} 2 & 2 & 2 \\ 0 & 3 & 3 \\ 0 & 12 & 18 \end{pmatrix}}_{A_2} \begin{matrix} \\ \\ -4 \end{matrix}$$

$$\rightarrow \underbrace{\begin{pmatrix} 2 & 2 & 2 \\ 0 & 3 & 3 \\ 0 & 0 & 6 \end{pmatrix}}_{A_3} \quad \text{done}$$

Observe:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \end{pmatrix} A_1$$

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} A_1$$

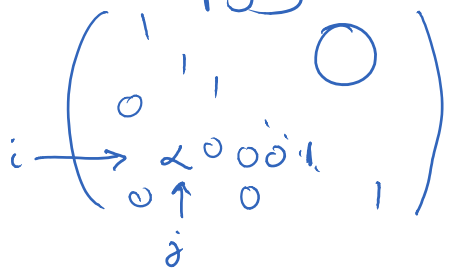
$$A_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix} A_2$$

$$A_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 1 \end{pmatrix} A_3$$

$$\Rightarrow A = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 1 \end{pmatrix}}_{\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{pmatrix} = L}\begin{pmatrix} 2 & 2 & 2 \\ 0 & 3 & 3 \\ 0 & 0 & 6 \end{pmatrix}$$

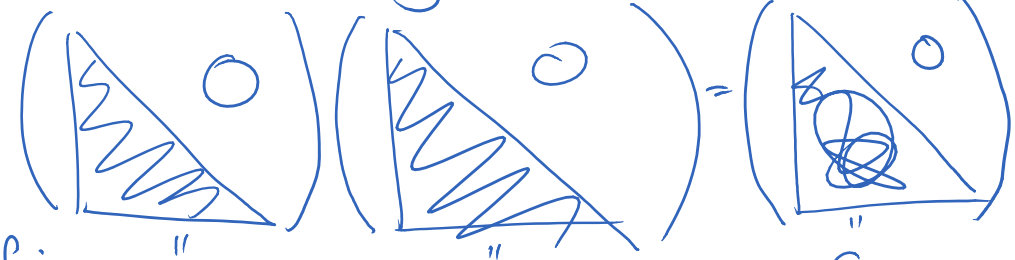
$\parallel$ U ✓

Claim 1: Multiplying on the left by



adds α times row j to row i

Claim 2: The product of two lower-triangular matrices is lower triangular, too.



Proof: $\parallel$ A, $\parallel$ B, $\parallel$ C

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = \sum_{k=j}^{i} a_{ik} b_{kj} = 0 \quad \text{if } i<j \ \square$$

$a_{ik} = 0$ if $i<k$

$b_{kj} = 0$ if $k<j$

Why is P necessary?

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} a & 0 \\ b & c \end{pmatrix}\begin{pmatrix} d & e \\ 0 & f \end{pmatrix}$$

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} a & 0 \\ b & c \end{pmatrix}\begin{pmatrix} d & e \\ 0 & f \end{pmatrix}$$

$$\Rightarrow 0 = a \cdot d$$

$$\Rightarrow \underline{a = 0} \quad \text{or} \quad \underline{d = 0}$$

$$\begin{pmatrix} 0 & 0 \\ b & c \end{pmatrix}\begin{pmatrix} d & e \\ 0 & f \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ u & u \end{pmatrix} \times$$

$$\begin{pmatrix} a & 0 \\ b & c \end{pmatrix}\begin{pmatrix} 0 & e \\ 0 & f \end{pmatrix} = \begin{pmatrix} 0 & u \\ 0 & u \end{pmatrix} \times$$

Problem: You can't pivot on 0.

So switch the two rows with $P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

$$PA = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \checkmark$$

$L \qquad U$