Lecture 5: Math Review I

Administrative

HW0 due Wednesday 1/29
(1 week from yesterday)

HW1 out yesterday,
due Wednesday 2/5
(3 weeks from yesterday)

Floating Point Arithmetic

This Lecture and Next: Math

Two goals for the next two classes:

- Math with computers ≠ Math
- Practical math you need to know but may not have been taught

This Lecture and Next: Goal

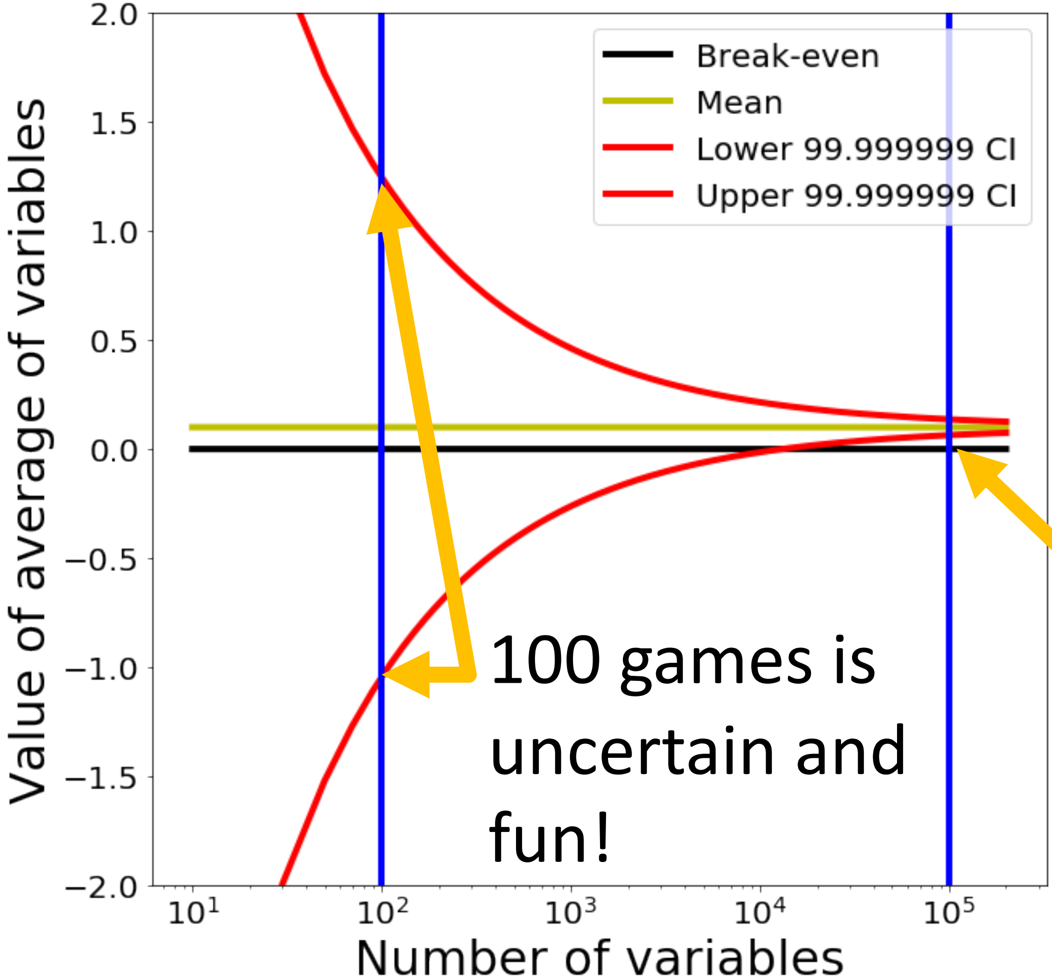
- Not a “Linear algebra in two lectures” – that’s impossible.
- *Some of this you should know!*
- Aimed at reviving your knowledge and plugging any gaps
- Aimed at giving you intuitions

Adding Numbers

- **1 + 1 = ?**
- Suppose x_i is normally distributed with mean μ and standard deviation σ for $1 \leq i \leq N$
- **How is the average, or $\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i$, distributed (qualitatively), in terms of variance?**
- *The Free Drinks in Vegas Theorem*: $\hat{\mu}$ has mean μ and standard deviation $\frac{\sigma}{\sqrt{N}}$.

Free Drinks in Vegas

Each game/variable has mean \$0.10, std \$2

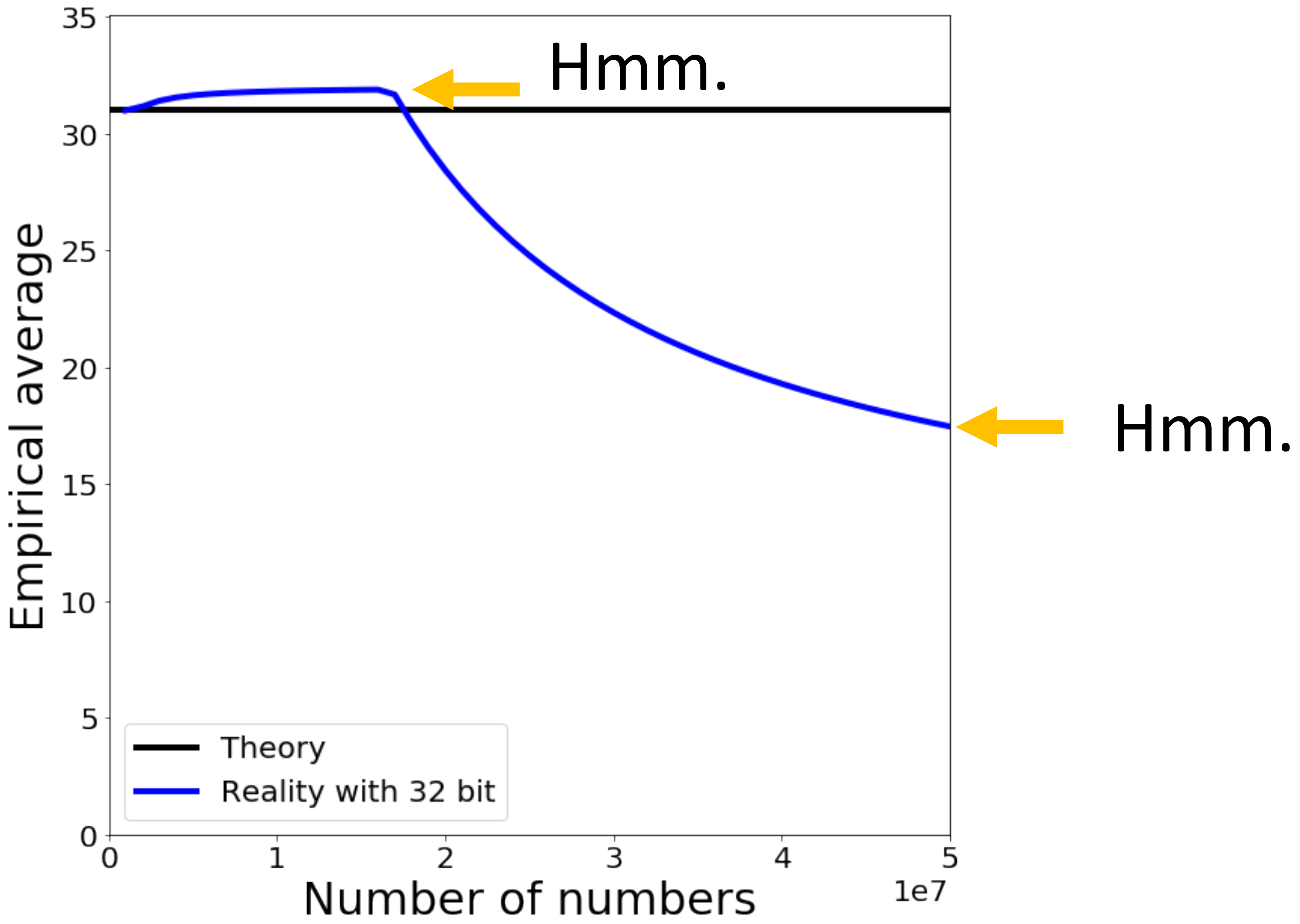


100K games is guaranteed profit: 99.999999% lowest value is \$0.064. \$0.01 for drinks \$0.054 for profits

Let’s make it big

- **What should happen qualitatively?**
- Theory says that the average is distributed with mean 31 and standard deviation $\frac{1}{\sqrt{50M}} \approx (10^{-5})$
- **What will happen?**
- Reality: 17.47

Trying it out



What is a number?

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	0	1	1	1	0	0	1	185

128 + 32 + 16 + 8 + 1 = 185

Adding two numbers

2^8		2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
		1	0	1	1	1	0	0	1	185
	+	0	1	1	0	1	0	0	1	105
1		0	0	1	0	0	0	1	0	34

Carry Flag

Result

"Integers" on a computer are integers ***modulo*** 2^k

Some Gotchas

32 + (3 / 4) x 40 = 32

32 + (3 x 40) / 4 = 62

Why?

<u>Underflow</u>	<u>No Underflow</u>
32 + (3 / 4) x 40 =	32 + (3 x 40) / 4 =
32 + 0 x 40 =	32 + 120 / 4 =
32 + 0 =	32 + 30 =
32	62

Ok – you have to multiply before dividing

Some Gotchas

Should be:
9x4=36

math	32 + (9 x 40) / 10 =	68
uint8	32 + (9 x 40) / 10 =	42

<u>Overflow</u>

32 + 9 x 40 / 10 =
32 + 104 / 10 =
32 + 10 =
42

Why 104?

9 x 40 = 360
360 % 256 = 104

What is a number?

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	0	1	1	1	0	0	1	185

How can we do fractions?

2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	
1	0	1	1	1	0	0	1	45.25

45 0.25

Fixed-Point Arithmetic

2^5	2^4	2^3	2^2	2^1	2^0	●	2^{-1}	2^{-2}	
1	0	1	1	1	0		0	1	45.25

What's the largest number we can represent?

63.75 – **Why?**

How precisely can we measure at 63?

0.25

How precisely can we measure at 0?

0.25

Fine for many purposes but for science, seems silly

Floating Point Numbers

Sign (S)	Exponent (E)	Fraction (F)
1	0 1 1 1	0 0 1
1	**7**	**1**
-1	**$2^{7-7} = 2^0 = 1$**	**1+1/8 = 1.125**

$$\left(-1^S\right)\left(2^{E+bias}\right)\left(1+\frac{F}{2^3}\right)$$

Bias: allows exponent to be negative (bias = -127 for float32)
Note: fraction = significant = mantissa;
exponents of all ones or all zeros are special numbers

Floating Point Numbers

Sign	Exponent
1	0 1 1 1
-1	7-7=0 *(-bias)*

Fraction

0/8	0 0 0	$-2^0 \times 1.00 = -1$
1/8	0 0 1	$-2^0 \times 1.125 = -1.125$
2/8	0 1 0	$-2^0 \times 1.25 = -1.25$
	...	
6/8	1 1 0	$-2^0 \times 1.75 = -1.75$
7/8	1 1 1	$-2^0 \times 1.875 = -1.875$

Floating Point Numbers

Sign	Exponent		Fraction	
1	1 0 0 1	0/8	0 0 0	$-2^2 \times 1.00 = -4$
-1	9-7=2 *(-bias)*	1/8	0 0 1	$-2^2 \times 1.125 = -4.5$
		2/8	0 1 0	$-2^2 \times 1.25 = -5$
			…	
		6/8	1 1 0	$-2^2 \times 1.75 = -7$
		7/8	1 1 1	$-2^2 \times 1.875 = -7.5$

Floating Point Numbers

Sign	Exponent	Fraction	
1	0 1 1 1	0 0 0	$-2^0 \times 1.00 = -1$
		0 0 1	$-2^0 \times 1.125 = -1.125$
1	1 0 0 1	0 0 0	$-2^2 \times 1.00 = -4$
		0 0 1	$-2^2 \times 1.125 = -4.5$

Gap between numbers is *relative*, not absolute

Adding Floating Point Numbers

	Sign	Exponent	Fraction	
	1	0 1 1 0	0 0 0	$-2^{-1} \times 1.00 = -0.5$
+	1	1 0 0 1	0 0 0	$-2^{2} \times 1.00 = -4$
	1	1 0 0 1	0 0 1	$-2^{2} \times 1.125 = -4.5$

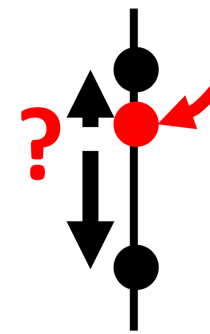
Actual implementation is complex

Adding Floating Point Numbers

	Sign	Exponent	Fraction	
	1	0 1 0 0	0 0 0	$-2^{-3} \times 1.00 = -0.125$
+	1	1 0 0 1	0 0 0	$-2^{2} \times 1.00 = -4$

$-2^{2} \times 1.03125 = -4.125$

1	1 0 0 1	0 0 0	$-2^{2} \times 1.00 = -4$
1	1 0 0 1	0 0 1	$-2^{2} \times 1.125 = -4.5$

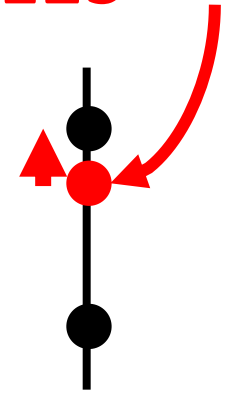


Adding Floating Point Numbers

	Sign	Exponent	Fraction	
	1	0 1 0 0	0 0 0	$-2^{-3} \times 1.00 = -0.125$
+	1	1 0 0 1	0 0 0	$-2^{2} \times 1.00 = -4$

$-2^{2} \times 1.03125 = -4.125$

1	1 0 0 1	0 0 0	$-2^{2} \times 1.00 = -4$



For a and b, these can happen

a + b = a   **a+b-a ≠ b**

Real Floating Point Numbers

IEEE 754 Single Precision / Single / float32

8 bits

$2^{127} \approx 10^{38}$

23 bits

$\approx$ 7 decimal digits

IEEE 754 Double Precision / Double / float64

11 bits

$2^{1023} \approx 10^{308}$

52 bits

$\approx$ 15 decimal digits

Real Floating Point Numbers

IEEE 754 Half Precision / Half / float16

5 bits

$2^{32} \approx 10^{9}$

10 bits

$\approx$ 3 decimal digits

Brain Floating Point / bfloat16

8 bits

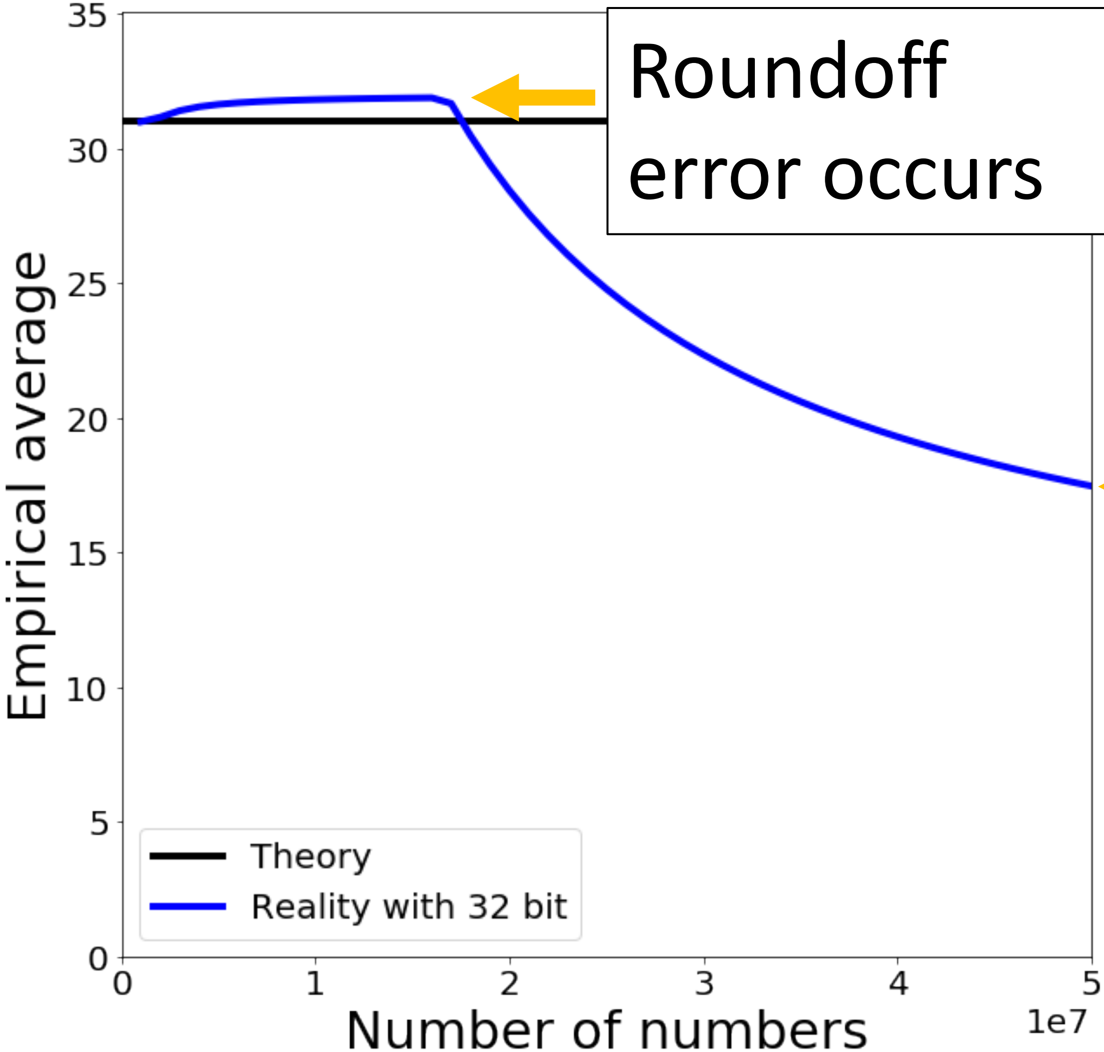
$2^{127} \approx 10^{38}$

7 bits

$\approx$ 2 decimal digits

Same range as FP32, but reduced precision

Trying it out



Roundoff error occurs

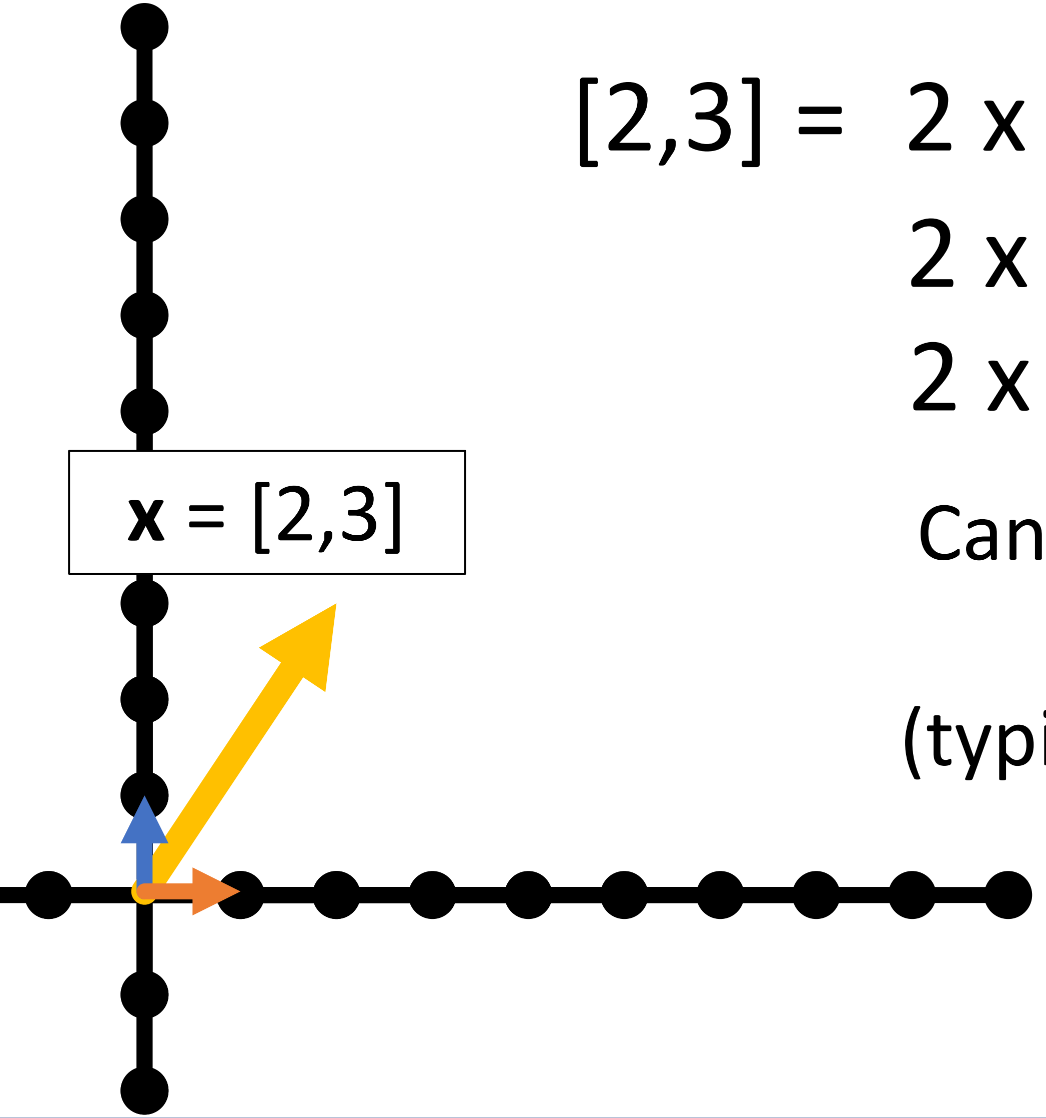
a+b=a -> numerator is stuck, denominator isn’t

Things to Remember

- Computer numbers aren’t math numbers
- Overflow, accidental zeros, roundoff error, and basic equalities are almost certainly incorrect for some values
- Floating point defaults and numpy try to protect you.
- Generally safe to use a double and use built-in-functions in numpy (not necessarily others!)
- Spooky behavior = look for numerical issues

Vectors

Vectors



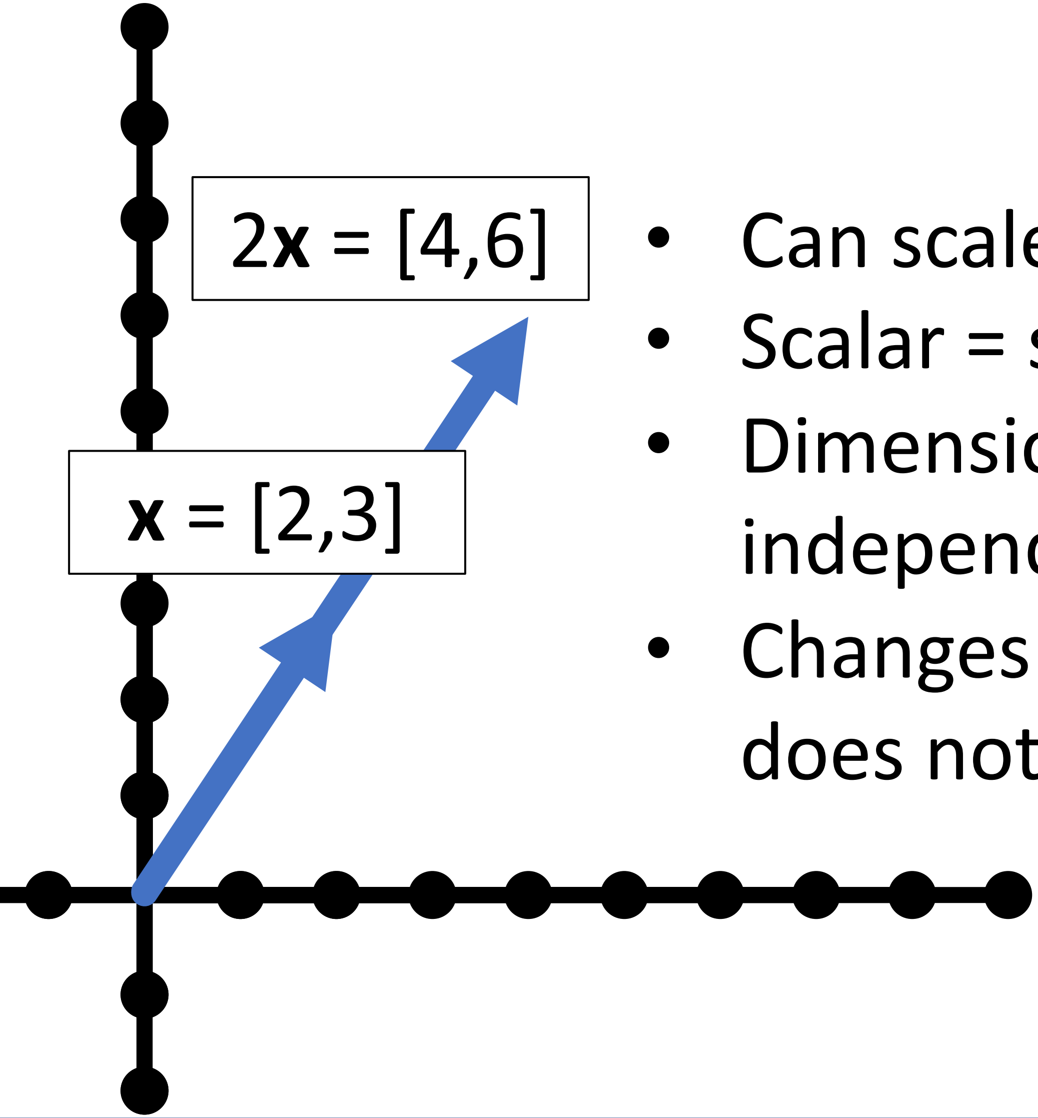
$$[2,3] = 2 \times [1,0] + 3 \times [0,1]$$

$$2 \times \rightarrow + 3 \times \uparrow$$

$$2 \times e_1 + 3 \times e_2$$

Can be arbitrary # of dimensions (typically denoted R^n)

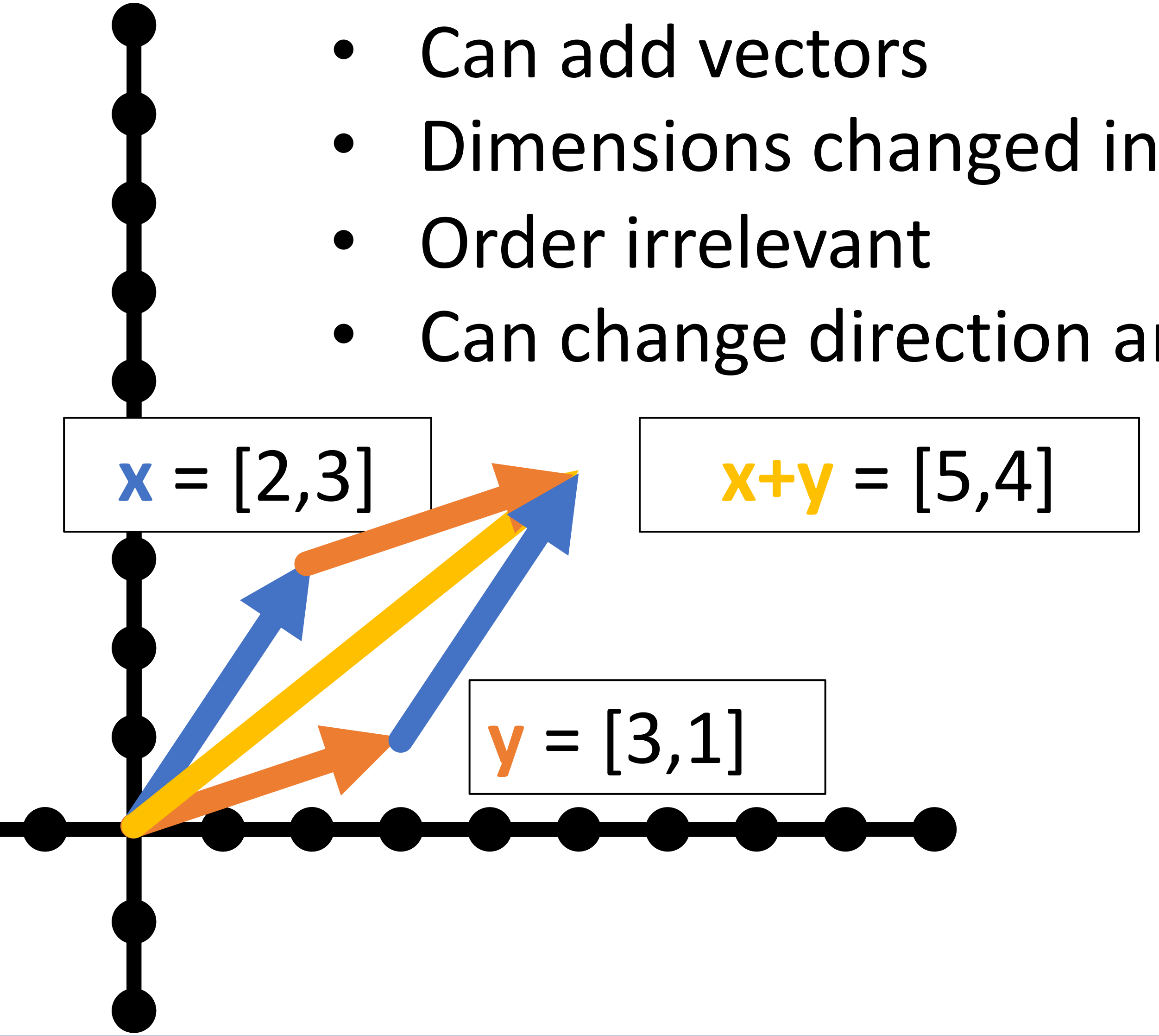
Scaling Vectors



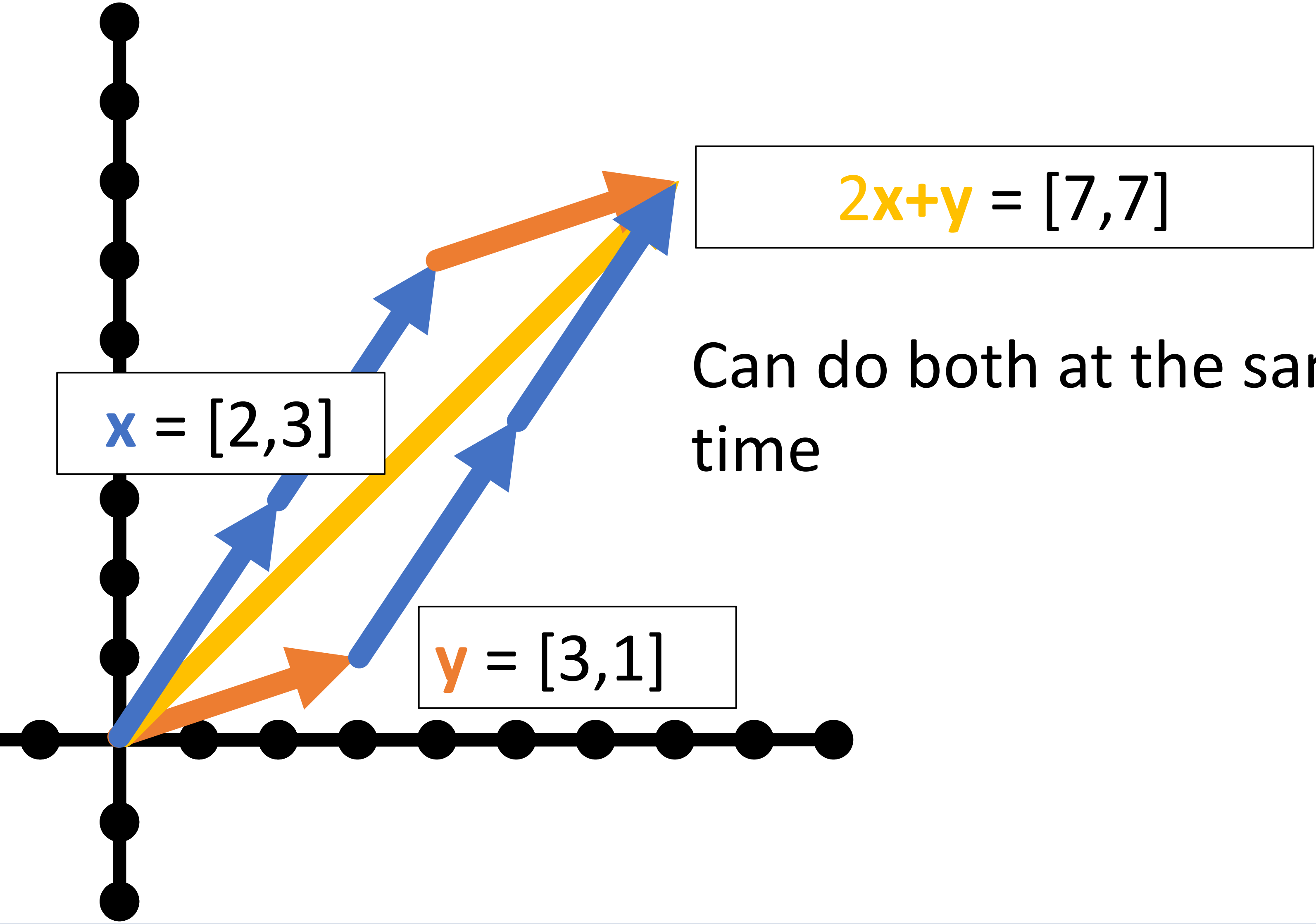
- Can scale vector by a *scalar*
- Scalar = single number
- Dimensions changed independently
- Changes *magnitude / length*, does not change *direction*.

Adding Vectors

- Can add vectors
- Dimensions changed independently
- Order irrelevant
- Can change direction and magnitude



Scaling and Adding



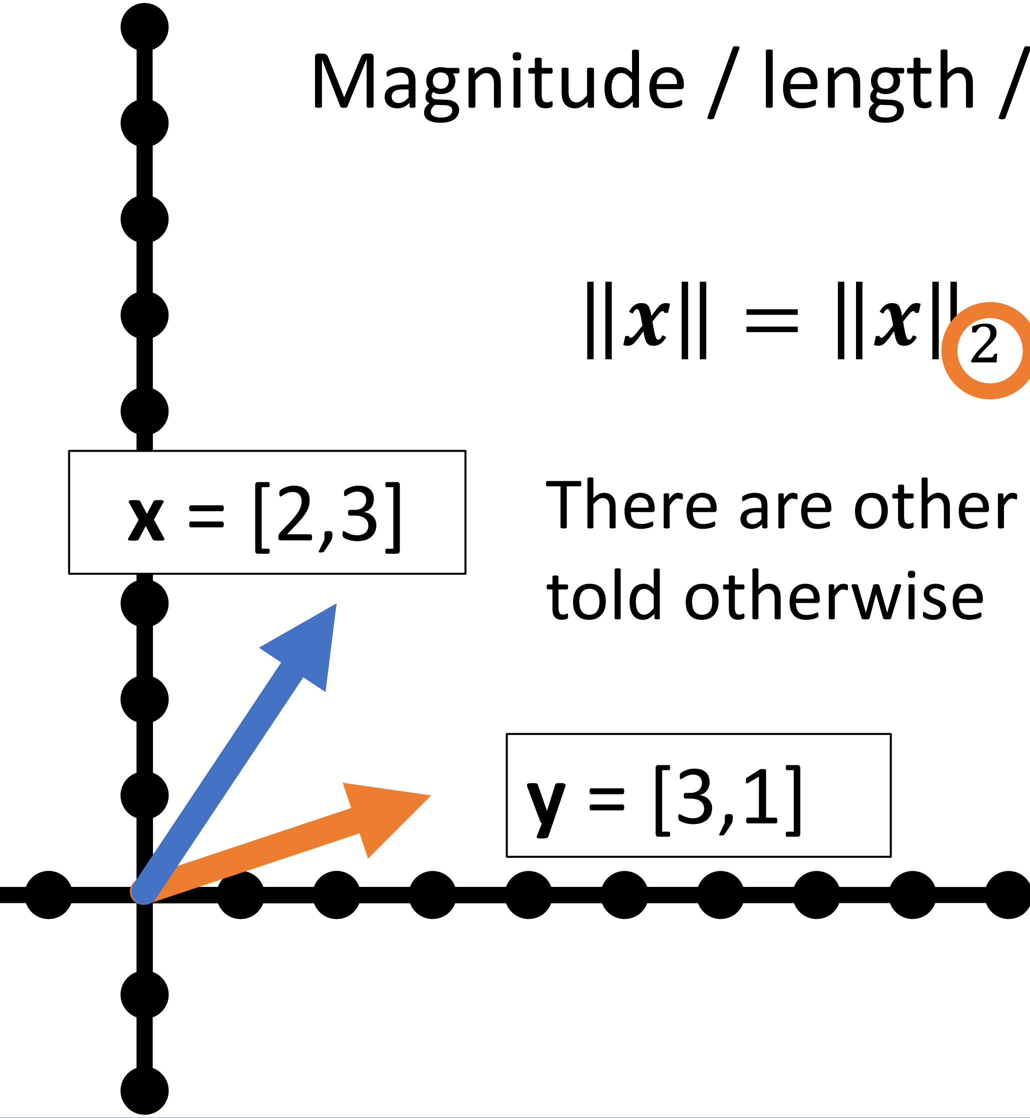
Can do both at the same time

Measuring Length

Magnitude / length / (L2) norm of vector

$$\|\boldsymbol{x}\| = \|\boldsymbol{x}\|_2 = \left(\sum_i^n x_i^2\right)^{1/2}$$

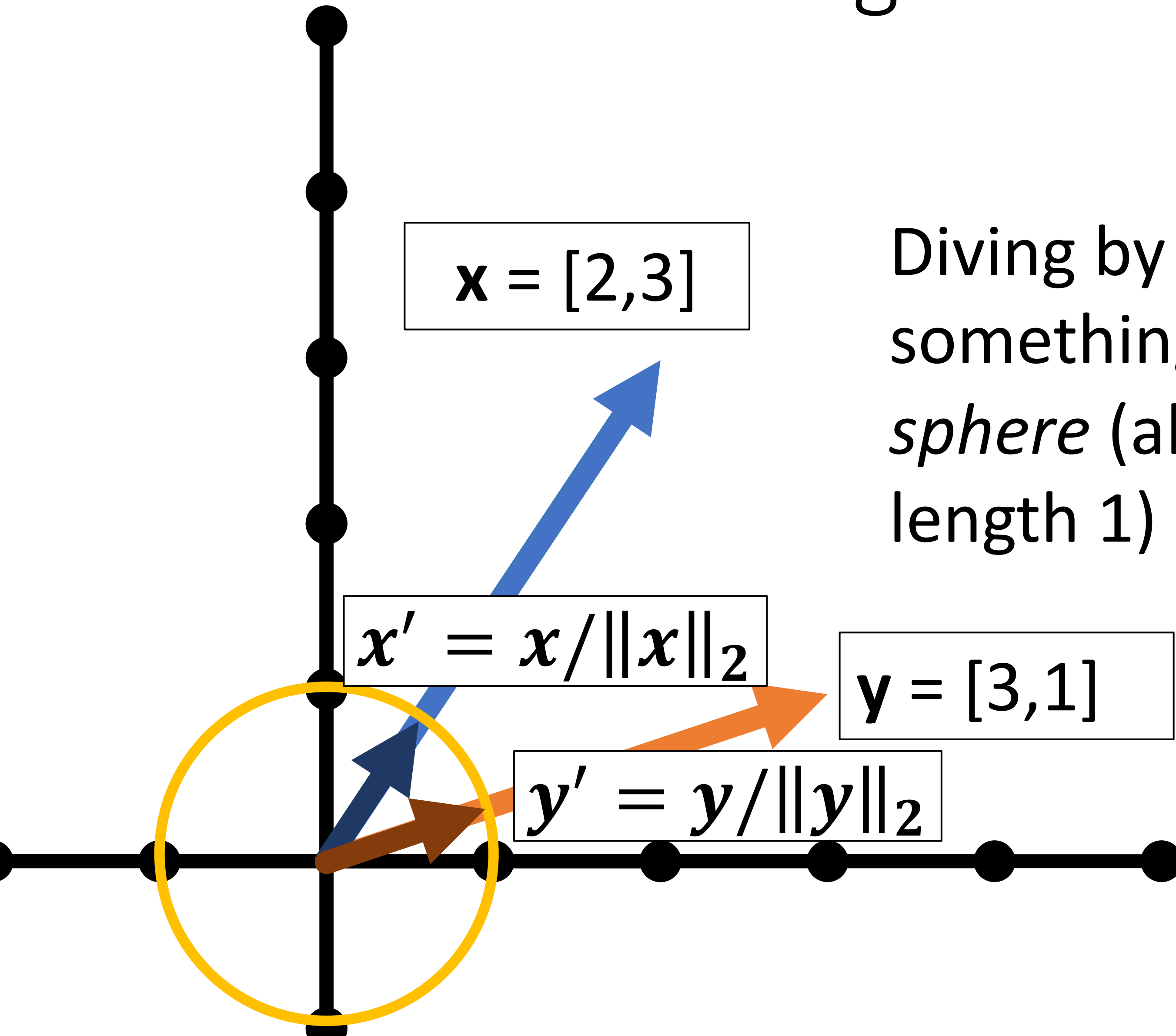
There are other norms; assume L2 unless told otherwise



$$\|\boldsymbol{x}\|_2 = \sqrt{13}$$
$$\|\boldsymbol{y}\|_2 = \sqrt{10}$$

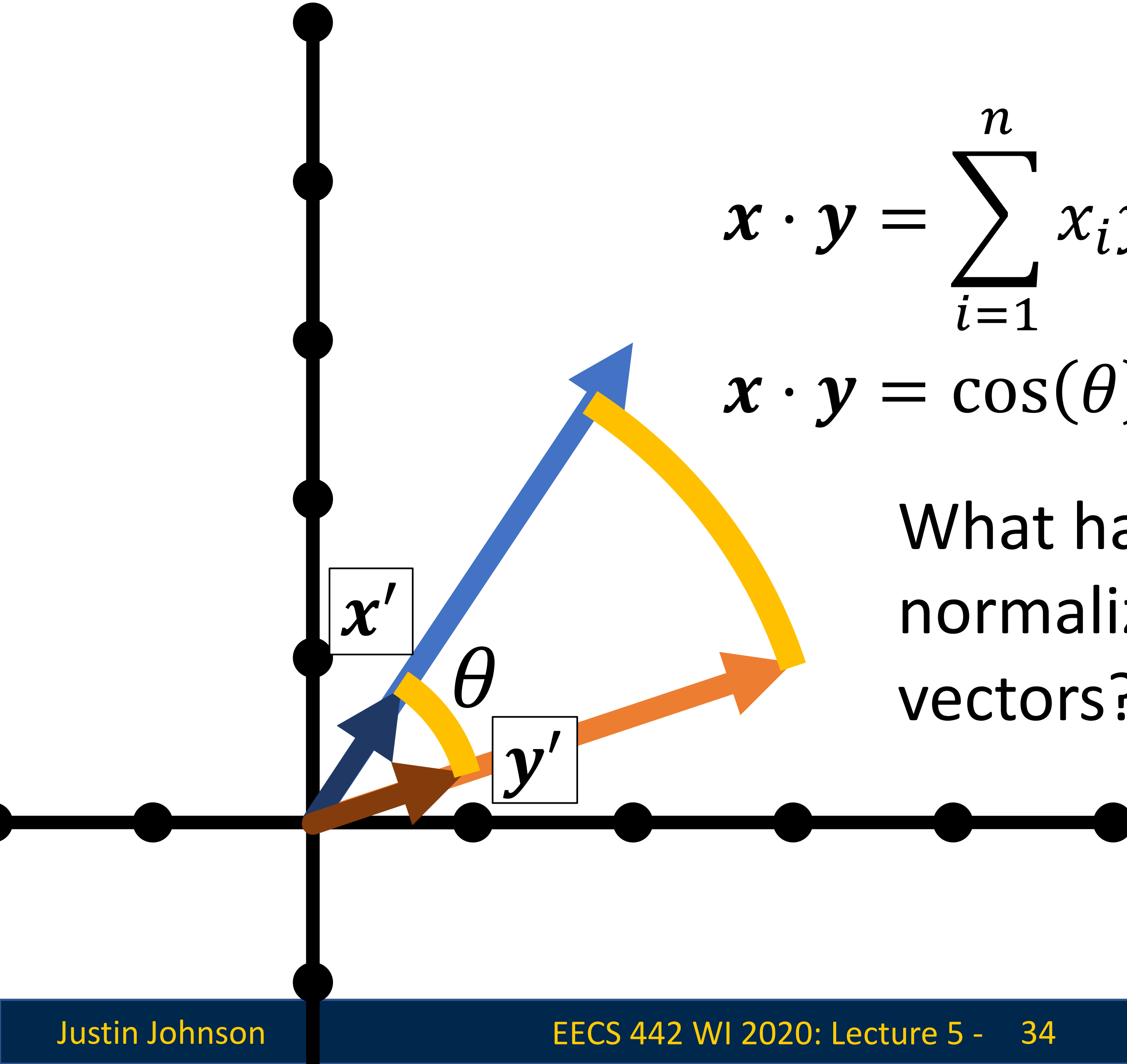
Why?

Normalizing a Vector



Diving by norm gives something on the *unit sphere* (all vectors with length 1)

Dot Products

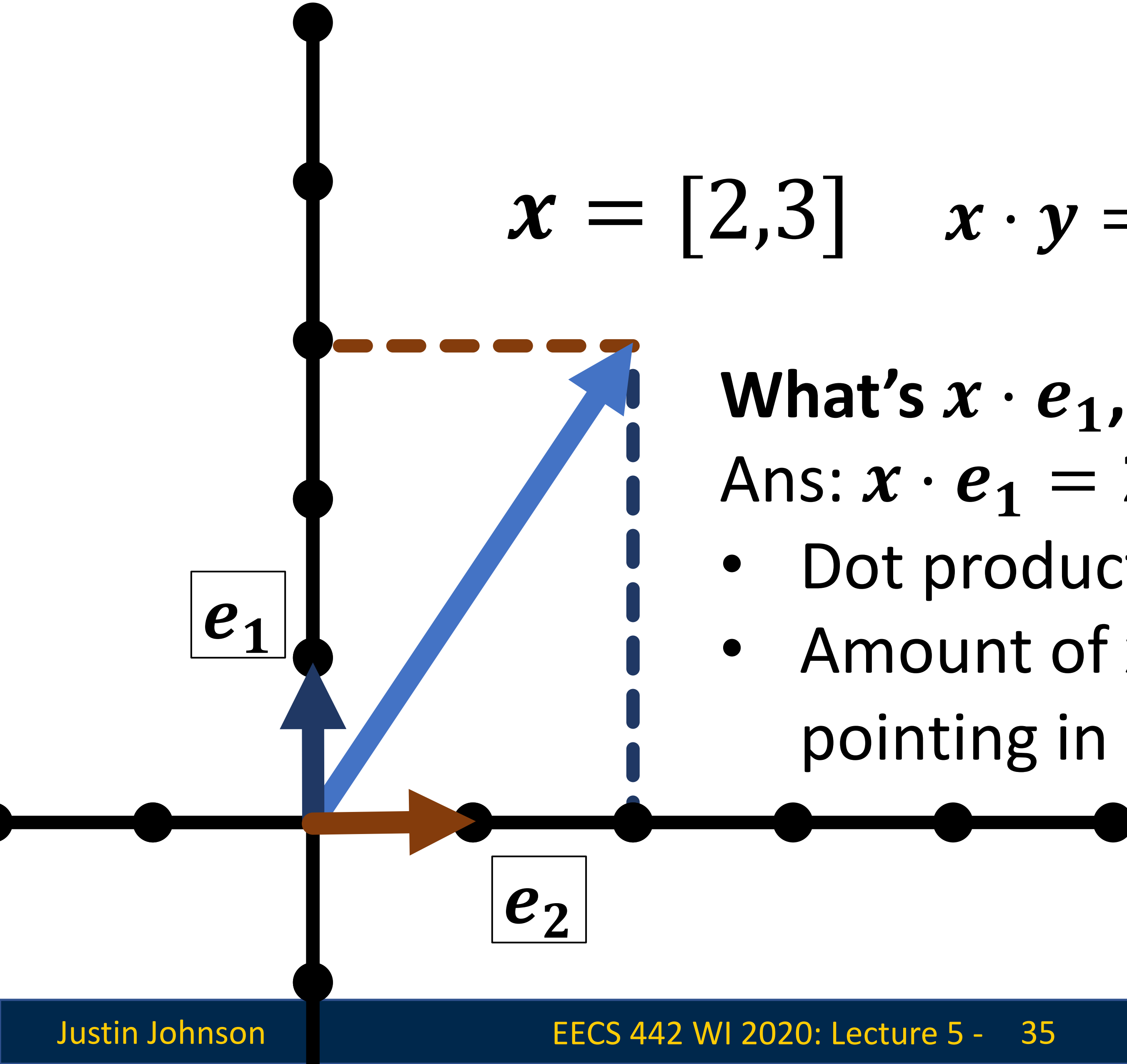


$$\boldsymbol{x} \cdot \boldsymbol{y} = \sum_{i=1}^{n} x_i y_i = \boldsymbol{x}^T \boldsymbol{y}$$

$$\boldsymbol{x} \cdot \boldsymbol{y} = \cos(\theta) \|\boldsymbol{x}\| \, \|\boldsymbol{y}\|$$

What happens with normalized / unit vectors?

Dot Products



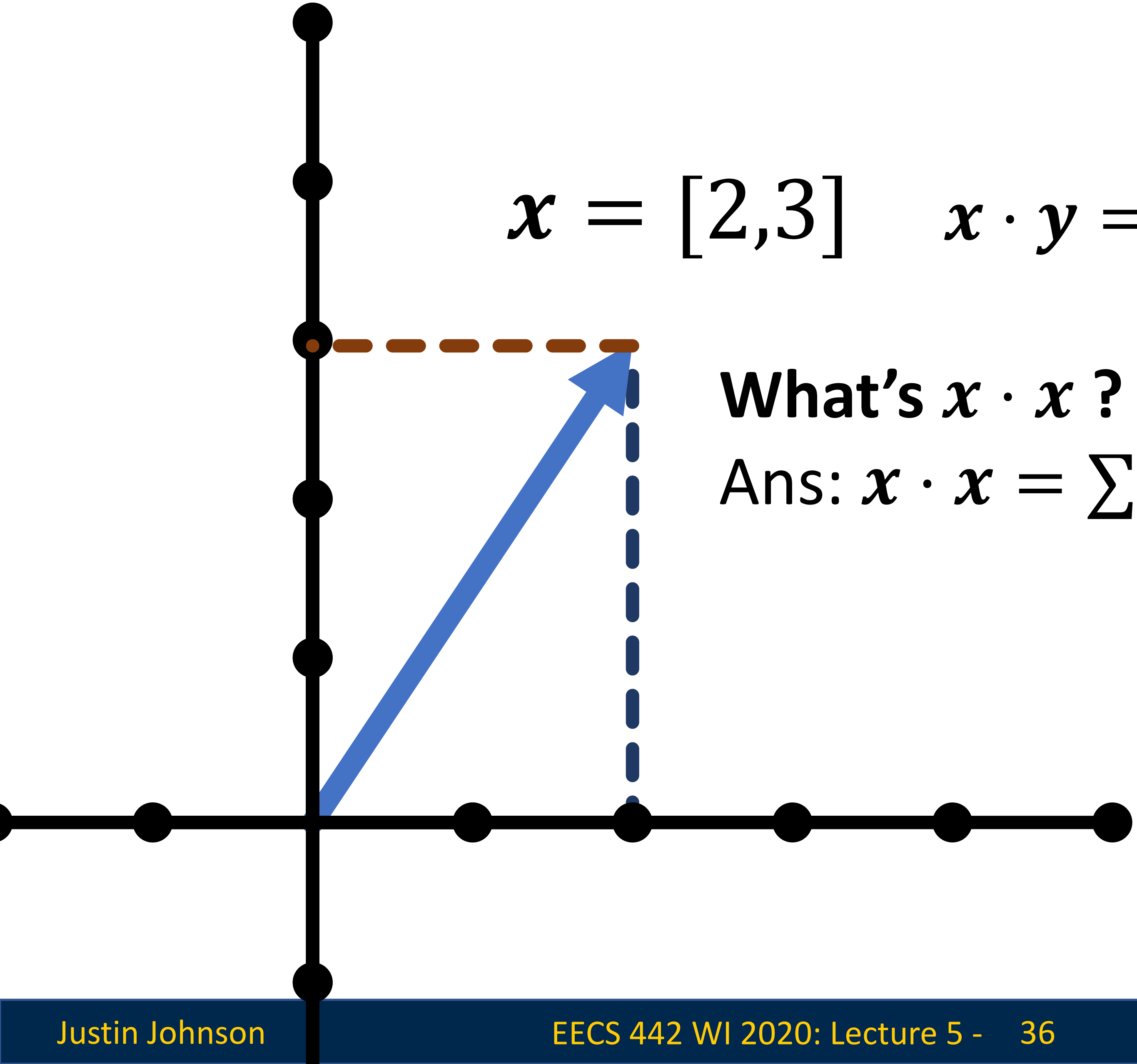
$$\boldsymbol{x} = [2,3] \qquad \boldsymbol{x} \cdot \boldsymbol{y} = \sum_{i}^{n} x_i y_i$$

What's $\boldsymbol{x} \cdot \boldsymbol{e_1}, \boldsymbol{x} \cdot \boldsymbol{e_2}$?

Ans: $\boldsymbol{x} \cdot \boldsymbol{e_1} = 2$; $\boldsymbol{x} \cdot \boldsymbol{e_2} = 3$

- Dot product is projection
- Amount of **x** that's also pointing in direction of **y**

Dot Products

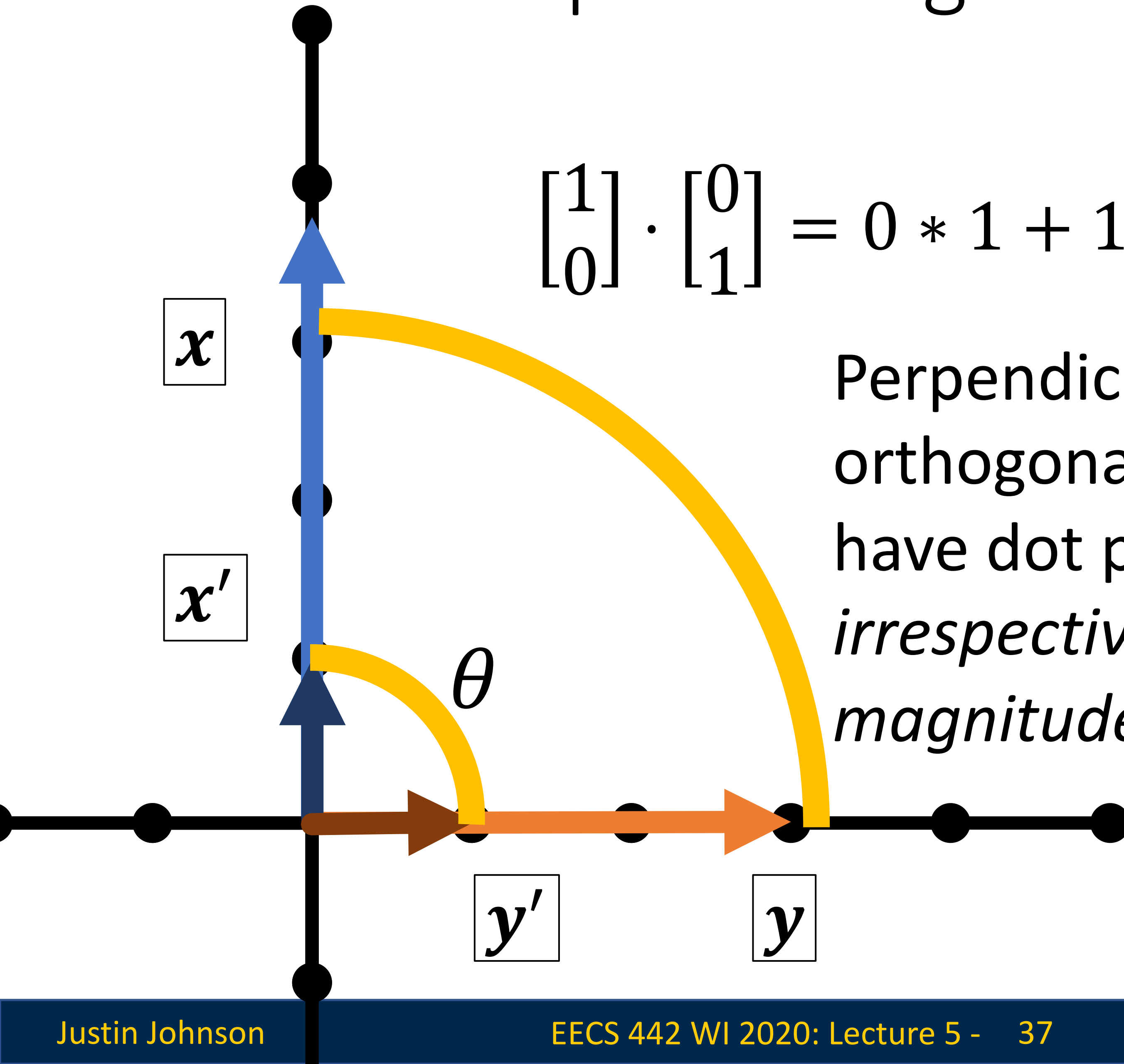


$$\boldsymbol{x} = [2,3] \qquad \boldsymbol{x} \cdot \boldsymbol{y} = \sum_{i}^{n} x_i y_i$$

What's $\boldsymbol{x} \cdot \boldsymbol{x}$?

Ans: $\boldsymbol{x} \cdot \boldsymbol{x} = \sum x_i x_i = \|\boldsymbol{x}\|_2^2$

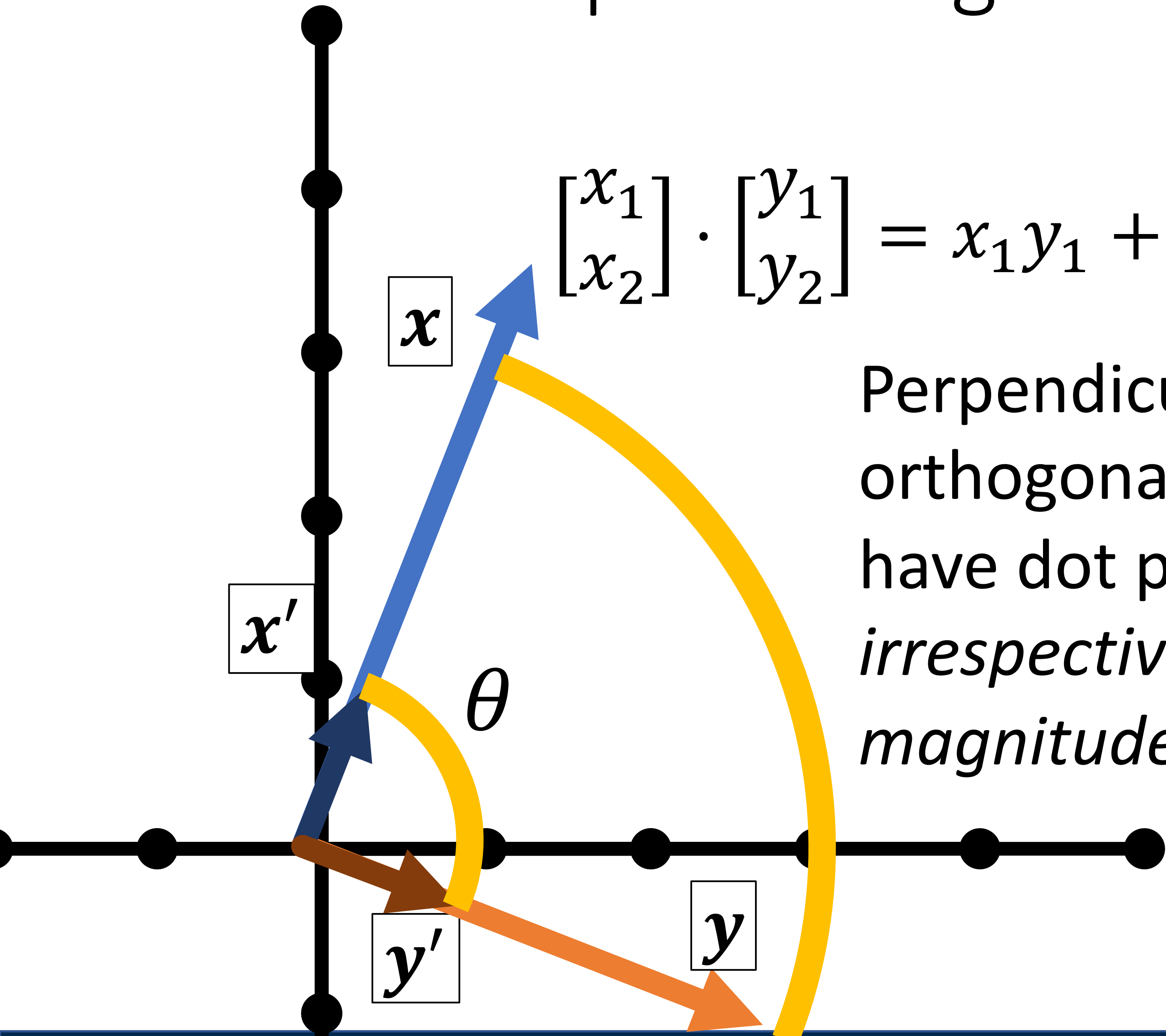
Special Angles



$$\begin{bmatrix}1\\0\end{bmatrix} \cdot \begin{bmatrix}0\\1\end{bmatrix} = 0 * 1 + 1 * 0 = 0$$

Perpendicular / orthogonal vectors have dot product 0 *irrespective of their magnitude*

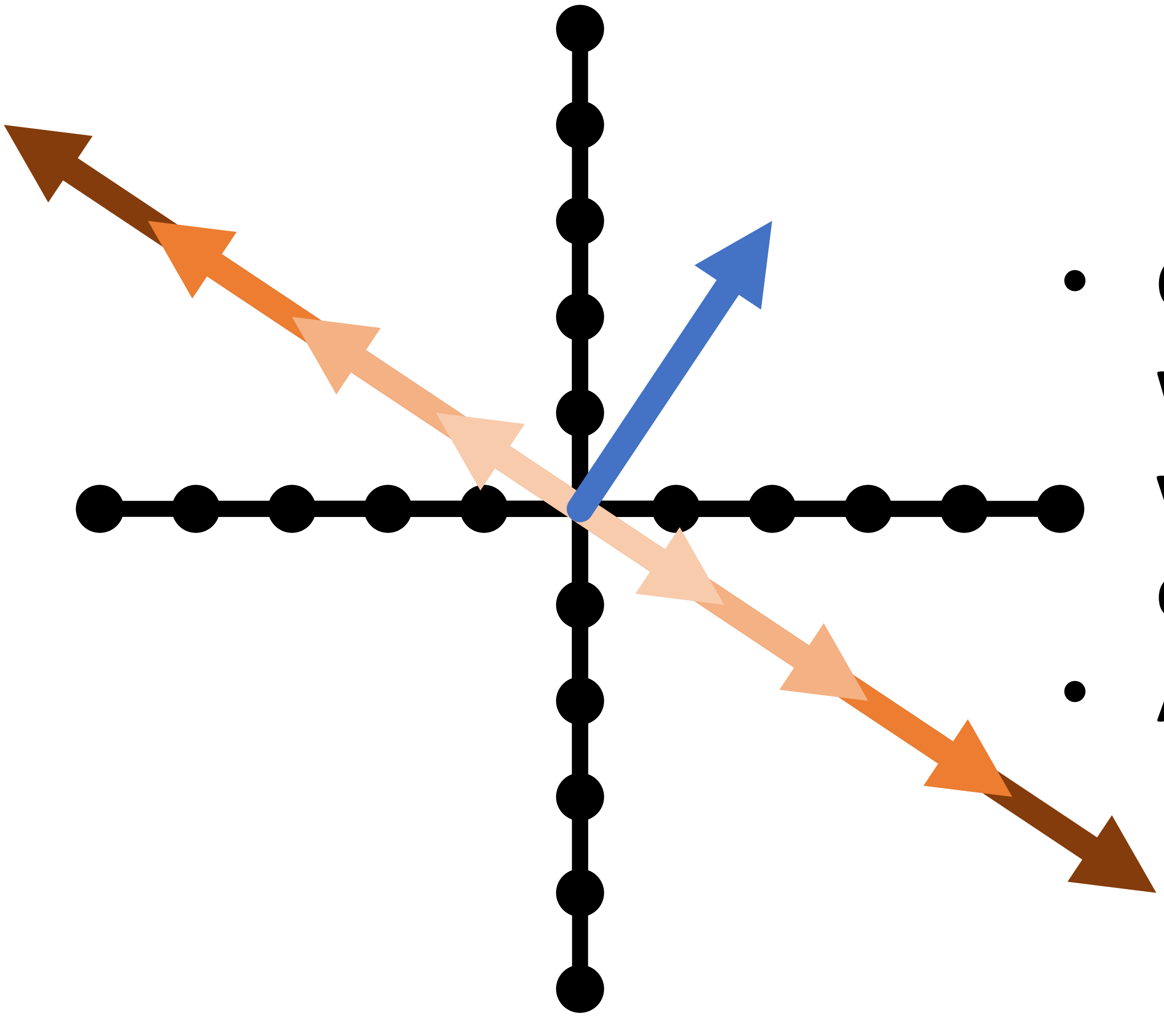
Special Angles



$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = x_1 y_1 + x_2 y_2 = 0$$

Perpendicular / orthogonal vectors have dot product 0 *irrespective of their magnitude*

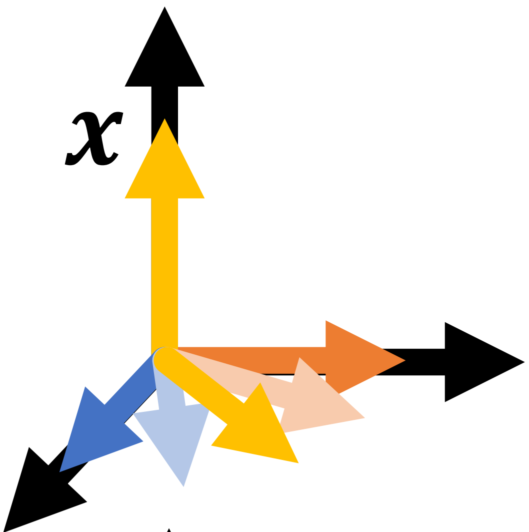
Orthogonal Vectors



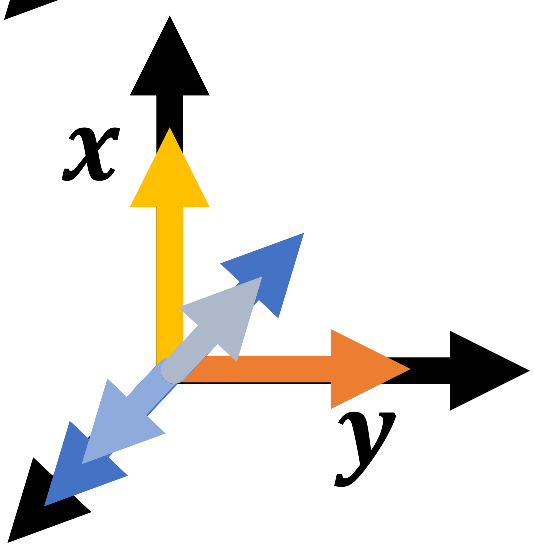
$\boldsymbol{x} = [2,3]$

- **Geometrically, what's the set of vectors that are orthogonal to x?**
- A line [3,-2]

Orthogonal Vectors

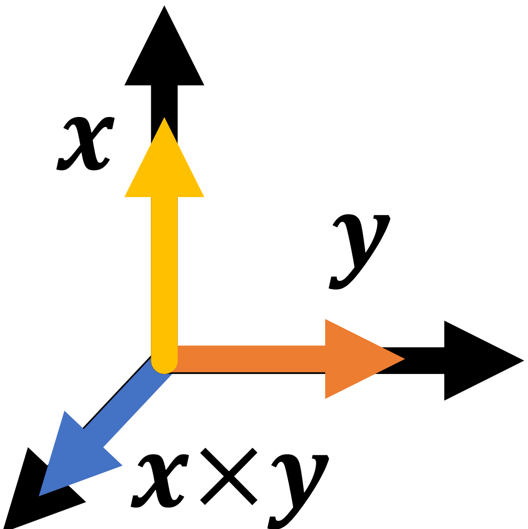


- **What's the set of vectors that are orthogonal to x = [5,0,0]?**
- A plane/2D space of vectors/any vector $[0, a, b]$



- **What's the set of vectors that are orthogonal to x <u>and</u> y = [0,5,0]?**
- A line/1D space of vectors/any vector $[0,0, b]$
- Ambiguity in *sign and magnitude*

Cross Product



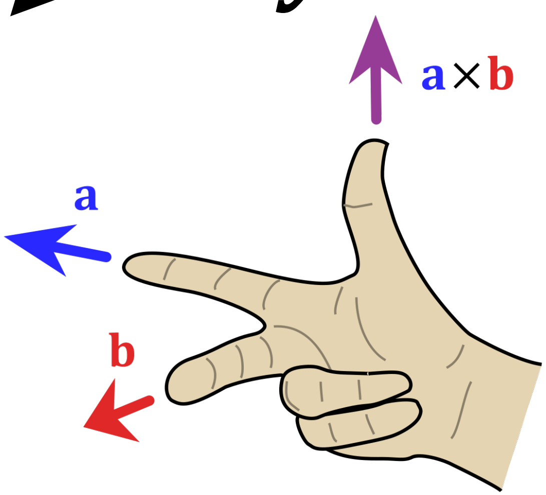


Image credit: Wikipedia.org

- Set $\{\boldsymbol{z}: \boldsymbol{z} \cdot \boldsymbol{x} = 0, \boldsymbol{z} \cdot \boldsymbol{y} = 0\}$ has an ambiguity in sign and magnitude
- Cross product $\boldsymbol{x} \times \boldsymbol{y}$ is: (1) orthogonal to x, y (2) has sign given by right hand rule and (3) has magnitude given by area of parallelogram of **x** and **y**
- **Important**: if x and y are the same direction or either is **0**, then $\boldsymbol{x} \times \boldsymbol{y} = \boldsymbol{0}$
- Only in 3D!
- (See wedge product for D != 3)

Operations You Should Know

- Scale (vector, scalar → vector)
- Add (vector, vector → vector)
- Magnitude (vector → **scalar)**
- Dot product (vector, vector → **scalar)**
    - **Dot products are projection / angles**
- Cross product (vector, vector → **vector)**
    - Vectors facing same direction have cross product **0**
- You can **never** mix vectors of different sizes

Matrices

Matrices

Horizontally concatenate n, m-dim column vectors and you get a mxn matrix A (here 2x3)

$$\boldsymbol{A} = [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n] = \begin{bmatrix} \boxed{\begin{matrix} v_{1_1} \\ v_{1_2} \end{matrix}} & \boxed{\begin{matrix} v_{2_1} \\ v_{2_2} \end{matrix}} & \boxed{\begin{matrix} v_{3_1} \\ v_{3_2} \end{matrix}} \end{bmatrix}$$

a	(scalar) lowercase undecorated	**a**	(vector) lowercase **bold or arrow**	**A**	(matrix) uppercase **bold**

Matrices

Horizontally concatenate n, m-dim column vectors and you get a mxn matrix A (here 2x3)

$$\boldsymbol{A} = [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n] = \begin{bmatrix} \boxed{\begin{matrix} v_{1_1} \\ v_{1_2} \end{matrix}} & \boxed{\begin{matrix} v_{2_1} \\ v_{2_2} \end{matrix}} & \boxed{\begin{matrix} v_{3_1} \\ v_{3_2} \end{matrix}} \end{bmatrix}$$

Watch out: In math, it's common to treat D-dim vector as a Dx1 matrix (column vector);
In numpy these are different things

Matrices

Transpose: flip rows / columns

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix}^T = \begin{bmatrix} a & b & c \end{bmatrix} \quad (3x1)^T = 1x3$$

Vertically concatenate m, n-dim row vectors and you get a mxn matrix A (here 2x3)

$$A = \begin{bmatrix} \boldsymbol{u}_1^T \\ \vdots \\ \boldsymbol{u}_n^T \end{bmatrix} = \begin{bmatrix} \boxed{u_{1_1} \quad u_{1_2} \quad u_{1_3}} \\ \boxed{u_{2_1} \quad u_{2_2} \quad u_{2_3}} \end{bmatrix}$$

Matrix-vector Product

$$\boldsymbol{y}_{2x1} = \boldsymbol{A}_{2x3}\boldsymbol{x}_{3x1}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{v_1} & \boldsymbol{v_2} & \boldsymbol{v_3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\boldsymbol{y} = x_1\boldsymbol{v_1} + x_2\boldsymbol{v_2} + x_3\boldsymbol{v_3}$$

*Linear combination of columns of **A***

Matrix-vector Product

$$\boldsymbol{y}_{2x1} = \boldsymbol{A}_{2x3}\boldsymbol{x}_{3x1}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{array}{c} \boxed{3} \\ \begin{bmatrix} \quad \boldsymbol{u}_{\boldsymbol{1}}^{\boldsymbol{T}} \quad \\ \quad \boldsymbol{u}_{\boldsymbol{2}}^{\boldsymbol{T}} \quad \end{bmatrix} \end{array} \;\; \boldsymbol{x} \; \boxed{3}$$

$$y_1 = \boldsymbol{u}_{\boldsymbol{1}}^{\boldsymbol{T}}\boldsymbol{x} \qquad y_2 = \boldsymbol{u}_{\boldsymbol{2}}^{\boldsymbol{T}}\boldsymbol{x}$$

*Dot product between rows of **A** and **x***

Matrix Multiplication

Generally: $\boldsymbol{A}_{mn}$ *and* $\boldsymbol{B}_{np}$ *yield product* $(\boldsymbol{AB})_{mp}$

$$\boldsymbol{AB} = \begin{bmatrix} - & \boldsymbol{a}_1^T & - \\ & \vdots & \\ - & \boldsymbol{a}_m^T & - \end{bmatrix} \begin{bmatrix} | & & | \\ \boldsymbol{b}_1 & \cdots & \boldsymbol{b}_p \\ | & & | \end{bmatrix}$$

Yes – in **A**, I'm referring to the rows, and in **B**, I'm referring to the columns

Matrix Multiplication

Generally: $\boldsymbol{A}_{mn}$ *and* $\boldsymbol{B}_{np}$ *yield product* $(\boldsymbol{AB})_{mp}$

$$\boldsymbol{AB}_{ij} = \boldsymbol{a}_i^T \boldsymbol{b}_j$$

$$\begin{bmatrix} | & & | \\ \boldsymbol{b}_1 & \cdots & \boldsymbol{b}_p \\ | & & | \end{bmatrix}$$

$$\boldsymbol{AB} = \begin{bmatrix} - & \boldsymbol{a}_1^T & - \\ & \vdots & \\ - & \boldsymbol{a}_m^T & - \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_1^T \boldsymbol{b}_1 & \cdots & \boldsymbol{a}_1^T \boldsymbol{b}_p \\ \vdots & \ddots & \vdots \\ \boldsymbol{a}_m^T \boldsymbol{b}_1 & \cdots & \boldsymbol{a}_m^T \boldsymbol{b}_p \end{bmatrix}$$

Matrix Multiplication

- Dimensions must match
- Dimensions must match
- Dimensions must match
- (Yes, it's associative): ABx = (A)(Bx) = (AB)x
- (*No it's not commutative*): ABx ≠ (BA)x ≠ (BxA)

Operations they don't teach

You Probably Saw Matrix Addition

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$$

What is this? FYI: e is a scalar

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + e = \qquad \begin{bmatrix} a+e & b+e \\ c+e & d+e \end{bmatrix}$$

Broadcasting

If you want to be pedantic and proper, you expand e by multiplying a matrix of 1s (denoted **1**)

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + e = \begin{bmatrix} a & b \\ c & d \end{bmatrix} + \mathbf{1}_{2x2}e$$

$$= \begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & e \\ e & e \end{bmatrix}$$

Many smart matrix libraries do this automatically. This is the source of many bugs.

Broadcasting Example

Given: a nx2 matrix **P** and a 2D column vector **v**, Want: nx2 difference matrix **D**

$$\boldsymbol{P} = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} \quad \boldsymbol{v} = \begin{bmatrix} a \\ b \end{bmatrix} \quad \boldsymbol{D} = \begin{bmatrix} x_1 - a & y_1 - b \\ \vdots & \vdots \\ x_n - a & y_n - b \end{bmatrix}$$

$$\boldsymbol{P} - \boldsymbol{v}^T = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} - \begin{matrix} [a & b] \\ & \vdots \\ [a & b] \end{matrix}$$

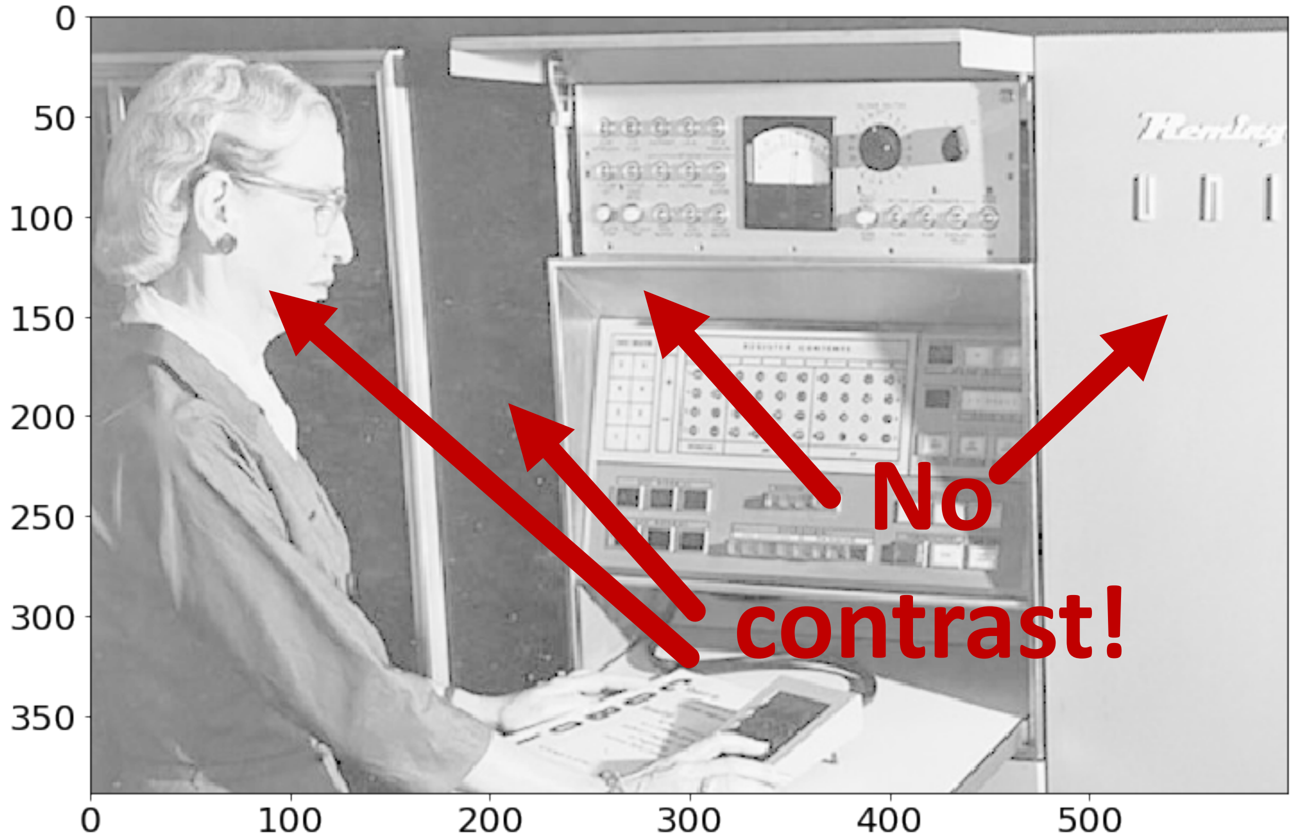
Blue stuff is assumed / broadcast

Two uses for Matrices

1. Storing things in a rectangular array (images, maps)
    - *Typical operations*: element-wise operations, convolution (which we'll cover next)
    - *Atypical operations*: almost anything you learned in a math linear algebra class
2. A linear operator that maps vectors to another space (**Ax**)
    - *Typical/Atypical:* reverse of above

Images as Matrices

Suppose someone hands you this matrix.
What's wrong with it?

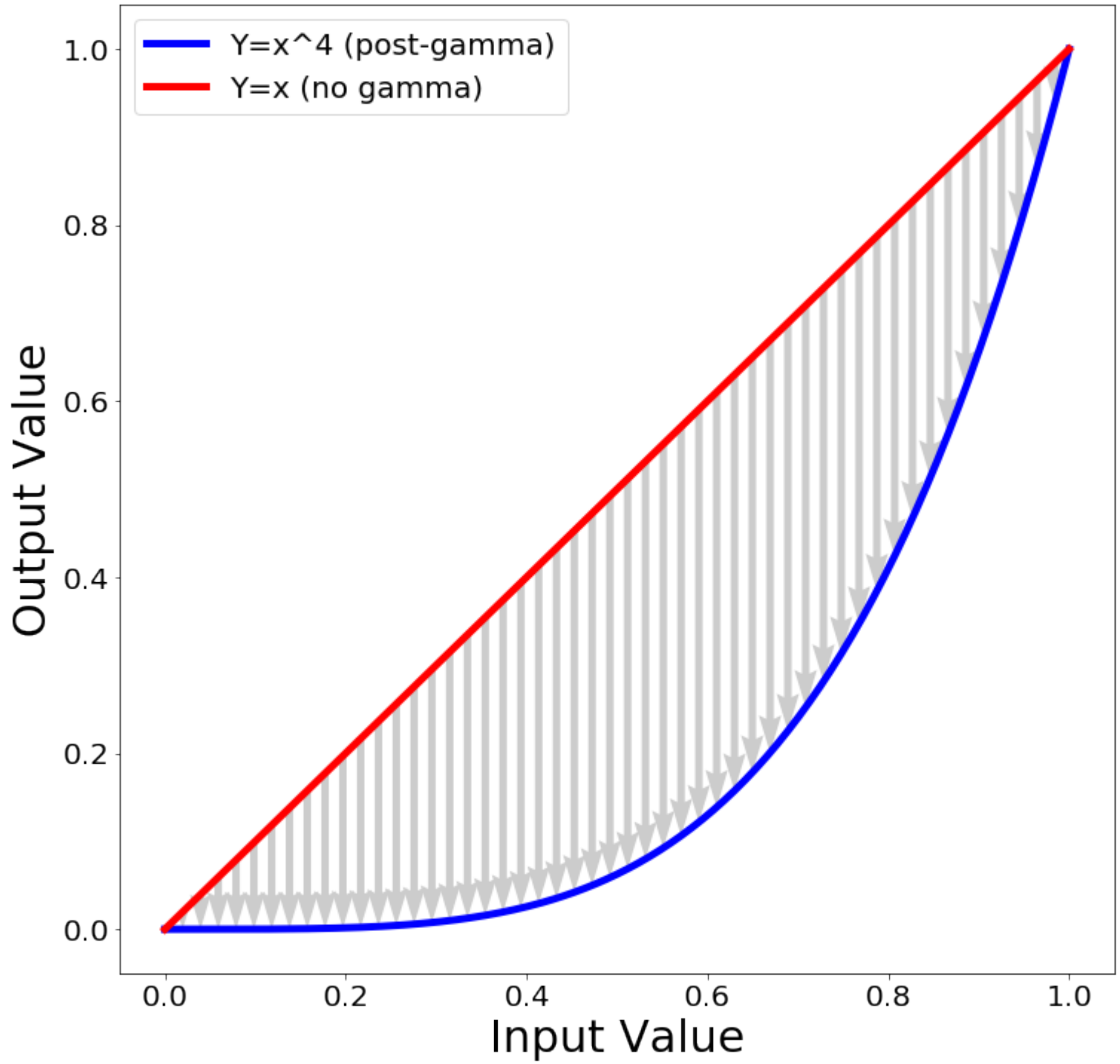


Contrast: Gamma Curve

Typical way to change the contrast is to apply a nonlinear correction

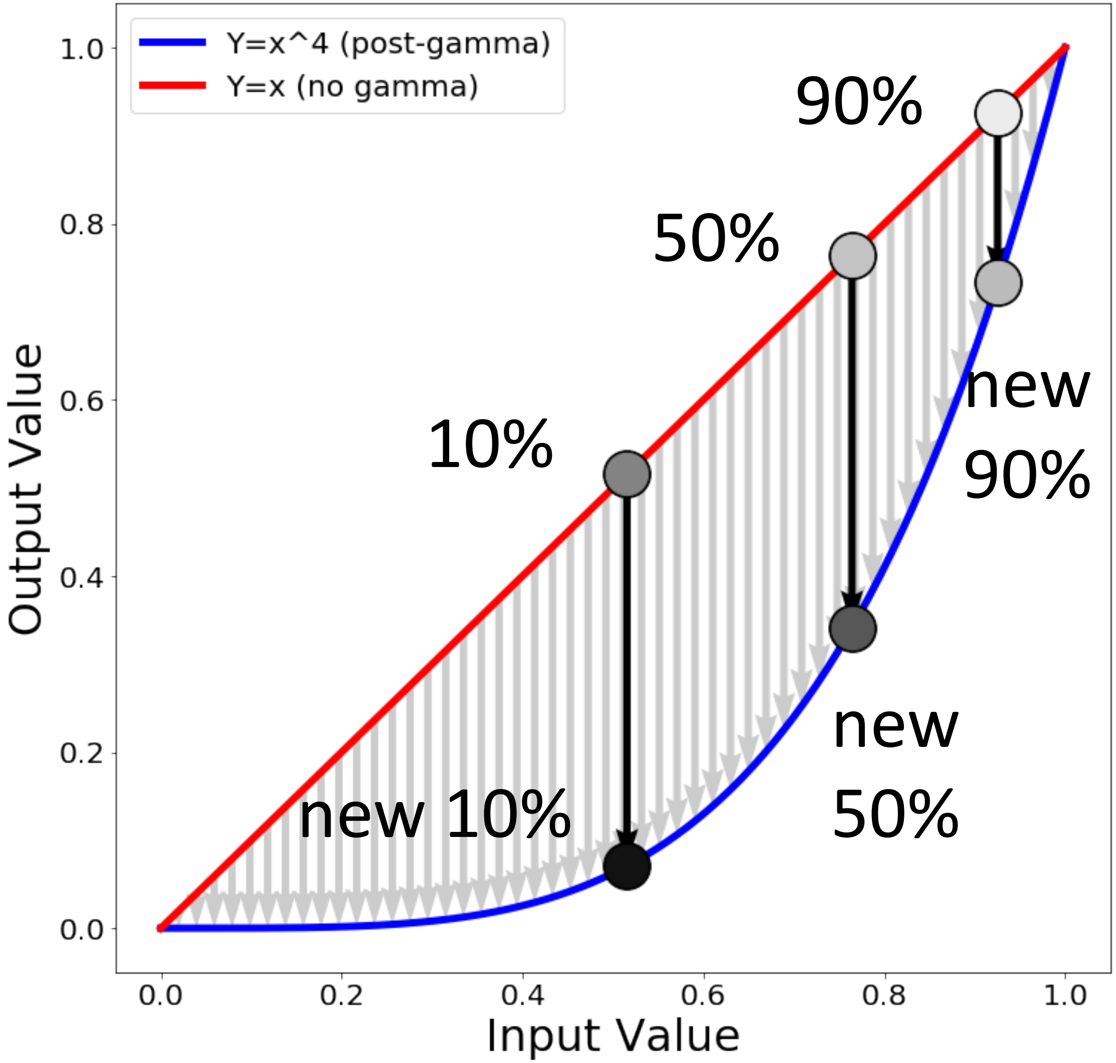
$$\text{pixelvalue}^{\gamma}$$

The quantity γ controls how much contrast gets added

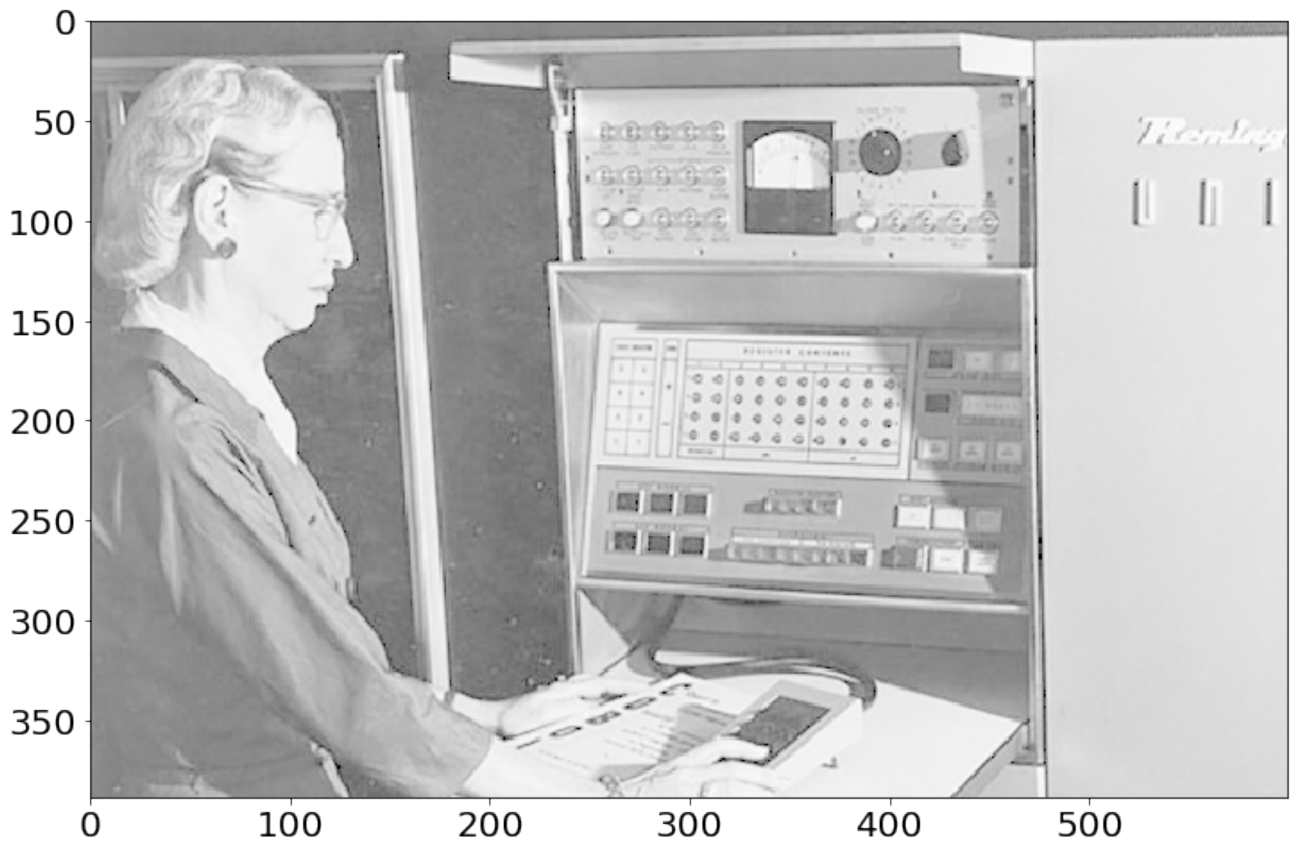


Contrast: Gamma Curve

Now the darkest regions (10th pctile) are **much** darker than the moderately dark regions (50th pctile).

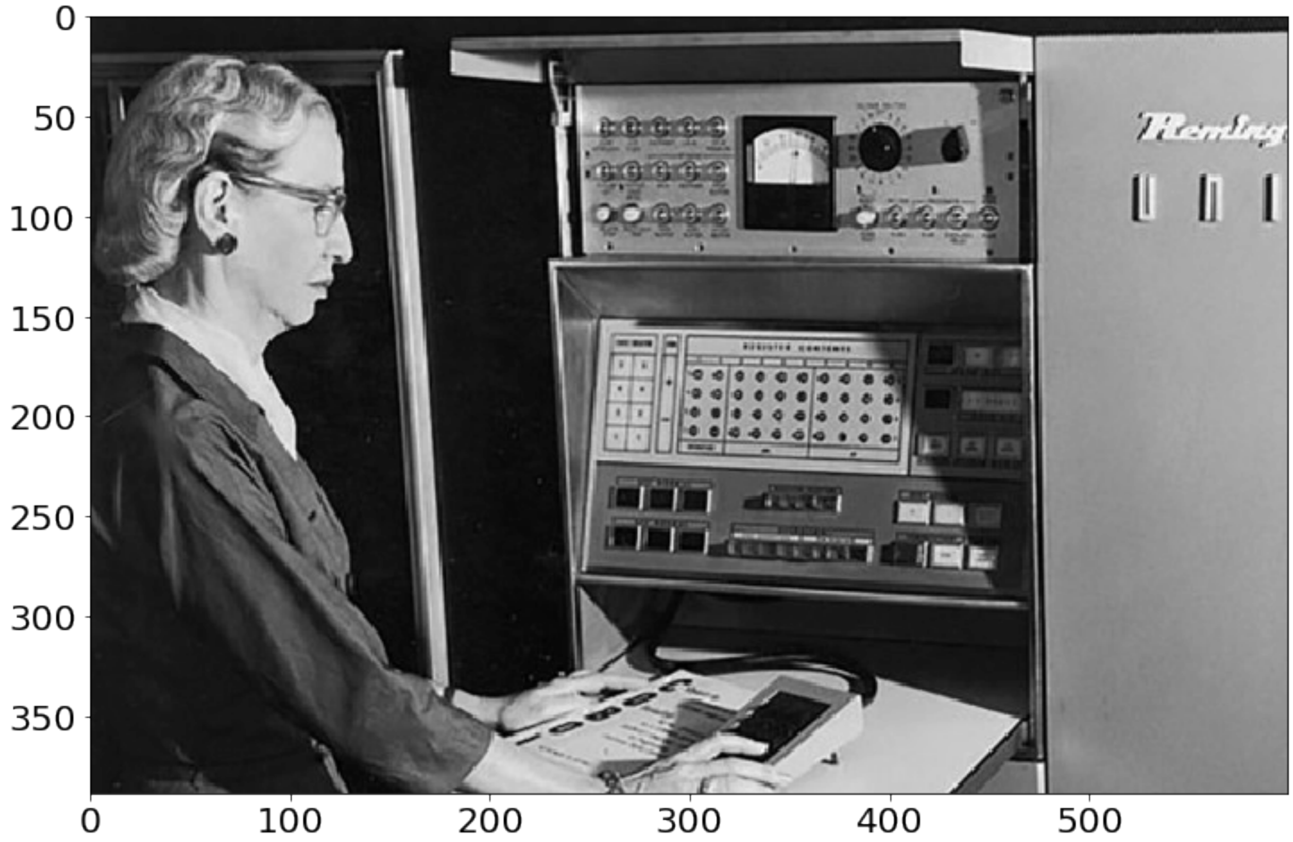


Contrast: Gamma Correction



Contrast: Gamma Correction

Phew! Much Better.



Implementation

Python+Numpy (right way):

```
imNew = im**4
```

Python+Numpy (slow way – **why?**):

```
imNew = np.zeros(im.shape)
for y in range(im.shape[0]):
  for x in range(im.shape[1]):
    imNew[y,x] = im[y,x]**expFactor
```

Next Time:
Vectorization, Tensors, Linear Algebra