# Linear Algebra



Paul Rad, Ph.D.

Associate Professor
Cyber Analytics and AI
Information Systems and Cyber Security
College of Business School
210.872.7259

# Outline

1. Definitions-Scalars, vectors and matrices
2. Vector and matrix calculations
3. Identity, inverse matrices & determinants
4. Eigenvectors & dot products

# Why use Linear Algebra in Machine Learning

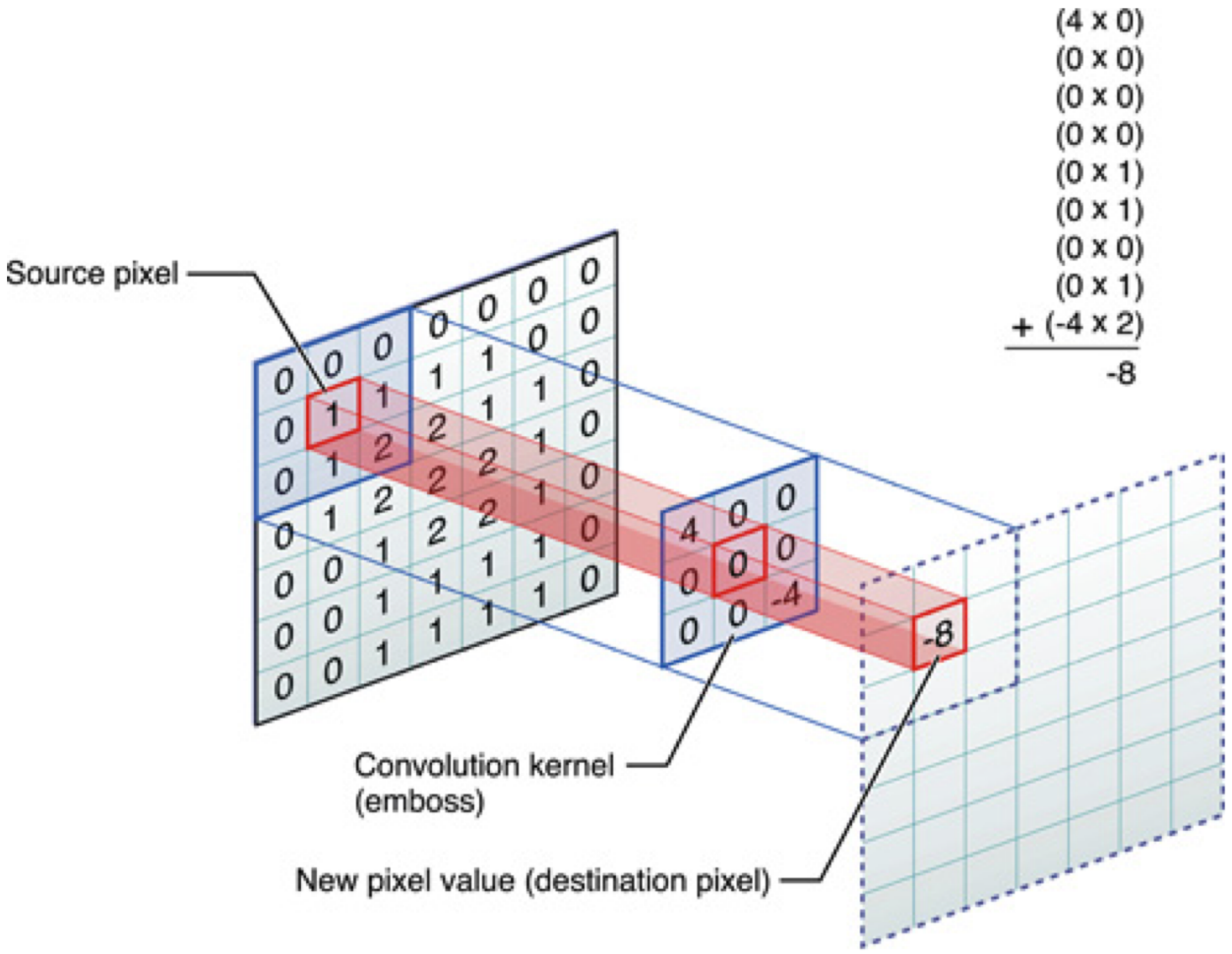
- Images and documents are literally matrices filled with numbers
- Coordinates can be used to perform geometrical transformations on contents

# Image

$$A_{n\times m} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$$

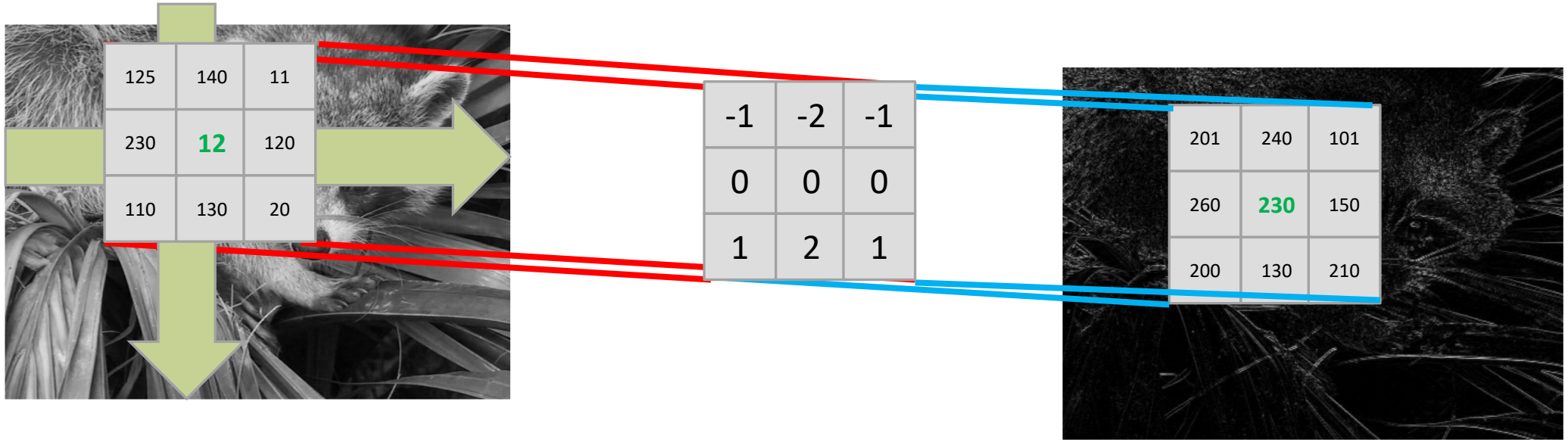
**Pixel intensity value**

# Convolution



# Convolution

**A convolution is an integral (discrete signals :Matrix Dot Product) that expresses the amount of overlap of one function as it is shifted over another function**




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

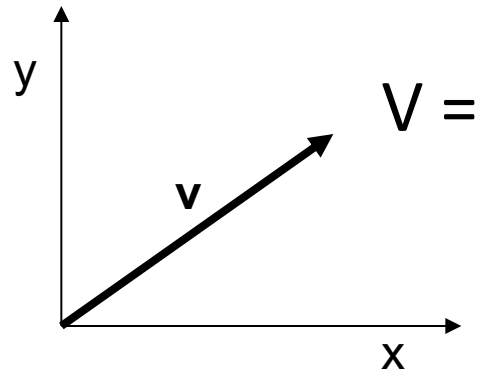
# What is a Vector ?

Think of a vector as a *directed line segment in N-dimensions*! (has “length” and “direction”)

$$\vec{v} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Basic idea: convert geometry in higher dimensions into algebra!

- Once you define a “nice” *basis* along each dimension: x-, y-, z-axis …
- Vector becomes a 1 x N matrix!
- $\mathbf{v} = [a \;\; b \;\; c]^T$
- Geometry starts to become linear algebra on vectors like **v**!

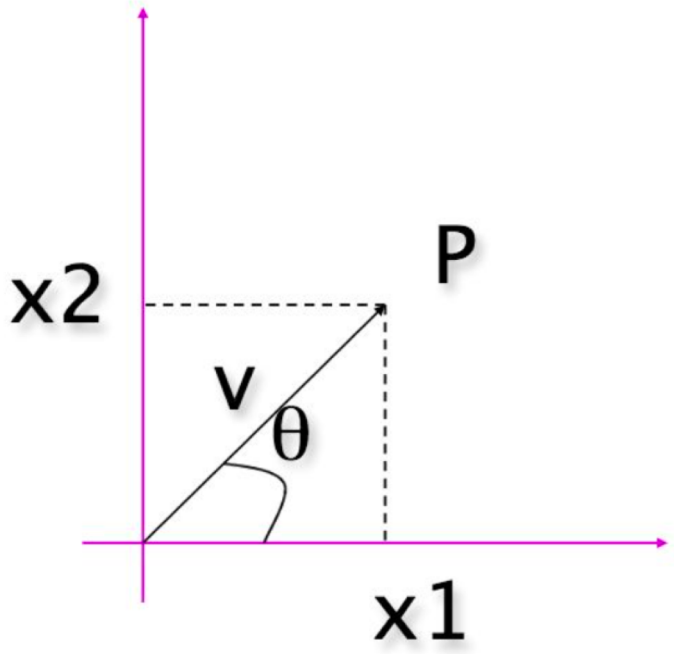



V = (x, y)

# Review Vector

Magnitude: $\| \mathbf{v} \| = \sqrt{x_1^2 + x_2^2}$

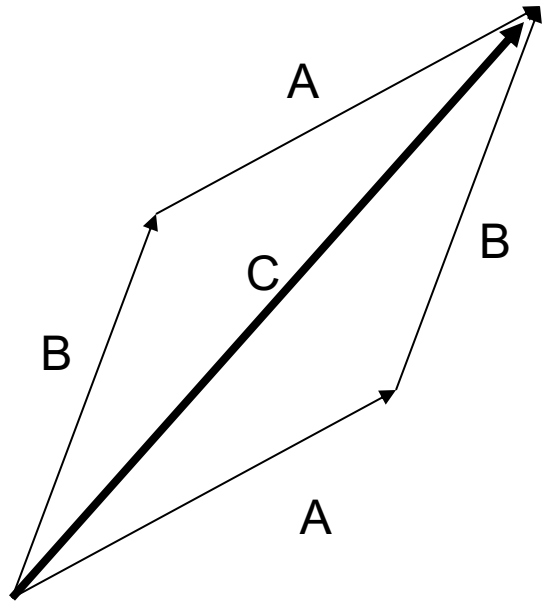
Orientation: $\theta = \tan^{-1}\left(\frac{x_2}{x_1}\right)$




V = (x1, x2)

# Vector Addition: A+B

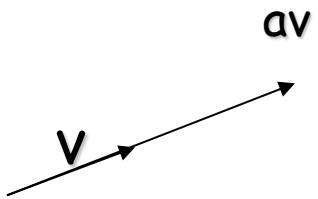
$$\mathbf{A+B} = (x_1, x_2) + (y_1, y_2) = (x_1 + y_1, x_2 + y_2)$$




**A+B = C**
**(use the head-to-tail method to combine vectors)**

# Scalar Product: $a$v

$$a\mathbf{v} = a(x_1, x_2) = (ax_1, ax_2)$$




Change only the length (“scaling”), but keep *direction fixed*.

# Vectors: Dot Product

$$A \cdot B = A^T B = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} d \\ e \\ f \end{bmatrix} = ad + be + cf$$

Think of the dot product as a matrix multiplication

$$\|A\|^2 = A^T A = aa + bb + cc$$

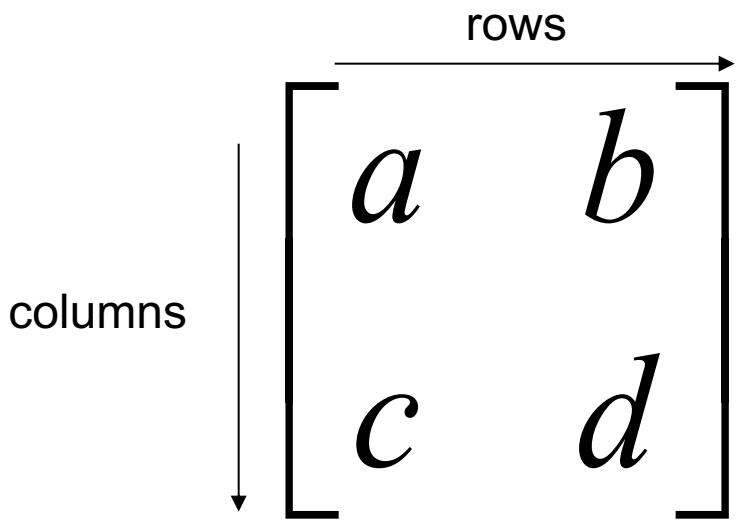
The magnitude is the dot product of a vector with itself

$$A \cdot B = \|A\| \, \|B\| \cos(\theta)$$

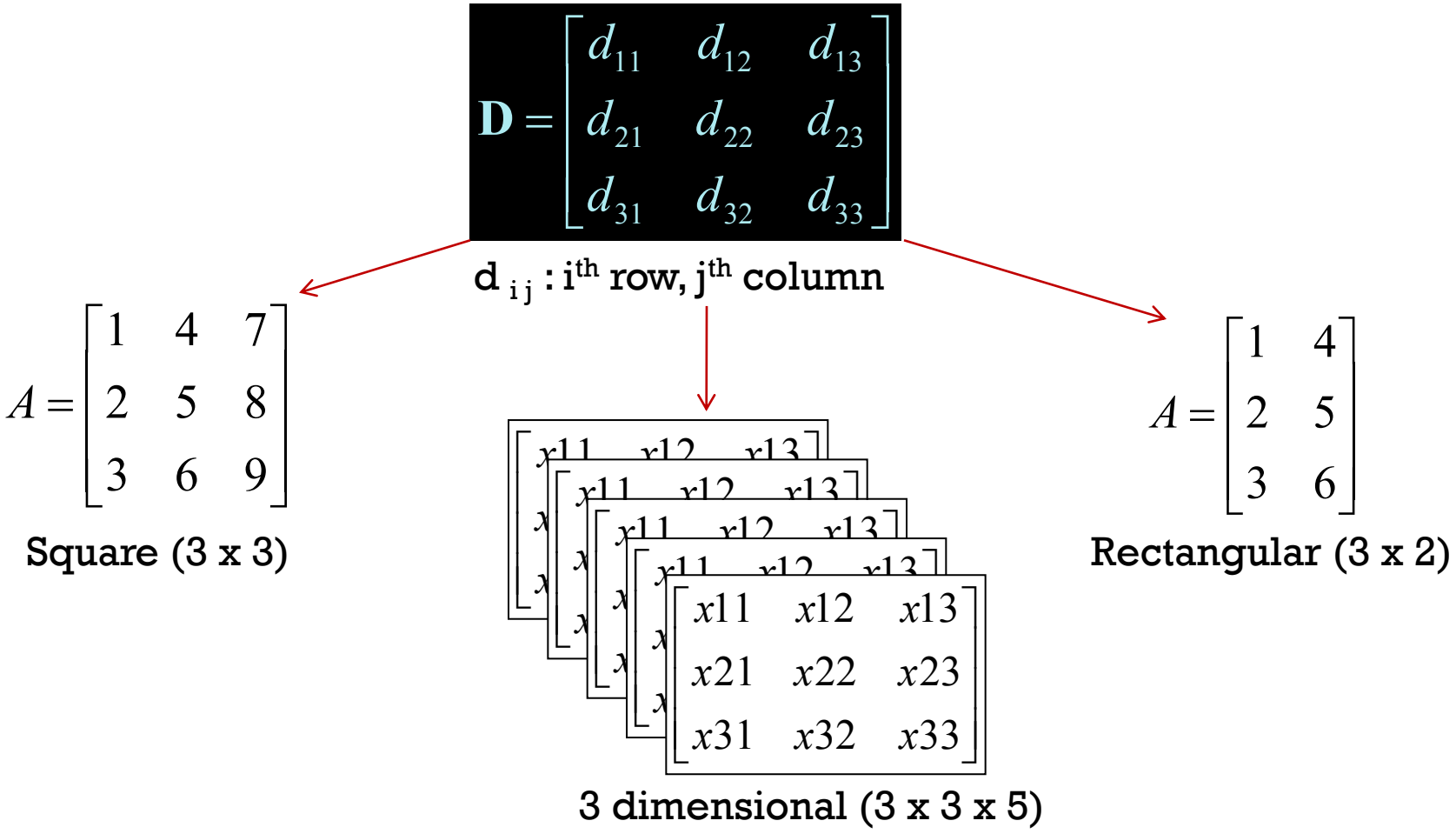
The dot product is also related to the angle between the two vectors

# What is a Matrix?

A matrix is a set of elements, organized into rows and columns



# Matrices


$\mathbf{D} = \begin{bmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \end{bmatrix}$
d i j : i^th row, j^th column
$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$
Square (3 x 3)
$A = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$
Rectangular (3 x 2)
$\begin{bmatrix} x11 & x12 & x13 \\ x21 & x22 & x23 \\ x31 & x32 & x33 \end{bmatrix}$
3 dimensional (3 x 3 x 5)

# Basic Matrix Operations

Addition, Subtraction, Multiplication: creating new matrices (or functions)

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

**Just add elements**

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

**Just subtract elements**

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

**Multiply each row by each column**

## Addition

- Commutative: **A**+**B**=**B**+**A**
- Associative: (**A**+**B**)+**C**=**A**+(**B**+**C**)

# Example

$$\mathbf{A}-\mathbf{B}=\begin{bmatrix}2 & 4\\ 5 & 3\end{bmatrix}-\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}=\begin{bmatrix}2 & 4\\ 5 & 3\end{bmatrix}+\begin{bmatrix}-1 & -2\\ -3 & -4\end{bmatrix}=\begin{bmatrix}1 & 2\\ 2 & -1\end{bmatrix}$$

# Matrix Times Matrix

$$\mathbf{L} = \mathbf{M} \cdot \mathbf{N}$$

$$\begin{bmatrix} l_{11} & l_{12} & l_{13} \\ l_{21} & l_{22} & l_{23} \\ l_{31} & l_{32} & l_{33} \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \cdot \begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{bmatrix}$$

$$l_{12} = m_{11}n_{12} + m_{12}n_{22} + m_{13}n_{32}$$

# Transposition

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 5 & 4 & 1 \\ 6 & 7 & 4 \end{bmatrix}$$

$$\mathbf{A}^T = \begin{bmatrix} 1 & 5 & 6 \\ 2 & 4 & 7 \\ 3 & 1 & 4 \end{bmatrix}$$

# Multiplication

Is AB = BA? Maybe, but maybe not!

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} e & f \\ g & h \end{bmatrix}=\begin{bmatrix} ae+bg & \dots \\ \dots & \dots \end{bmatrix} \quad \begin{bmatrix} e & f \\ g & h \end{bmatrix}\begin{bmatrix} a & b \\ c & d \end{bmatrix}=\begin{bmatrix} ea+fc & \dots \\ \dots & \dots \end{bmatrix}$$

Matrix multiplication AB: apply transformation B first, and then again transform using A!

Heads up: **multiplication is NOT commutative!**

**Note**: If A and B both represent either pure "*rotation*" or "*scaling*" they can be interchanged (i.e. AB = BA)

# Matrix multiplication

Matrix multiplication is NOT commutative i.e the order matters!

- $AB \neq BA$

Matrix multiplication IS associative

- $A(BC)=(AB)C$

Matrix multiplication IS distributive

- $A(B+C)=AB+AC$
- $(A+B)C=AC+BC$

# Matrix operating on vectors

Matrix is like a function that transforms the vectors on a plane

Matrix operating on a general point => transforms x- and y-components

*System of linear equations*: matrix is just the bunch of coeffs !

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

x’ = ax + by

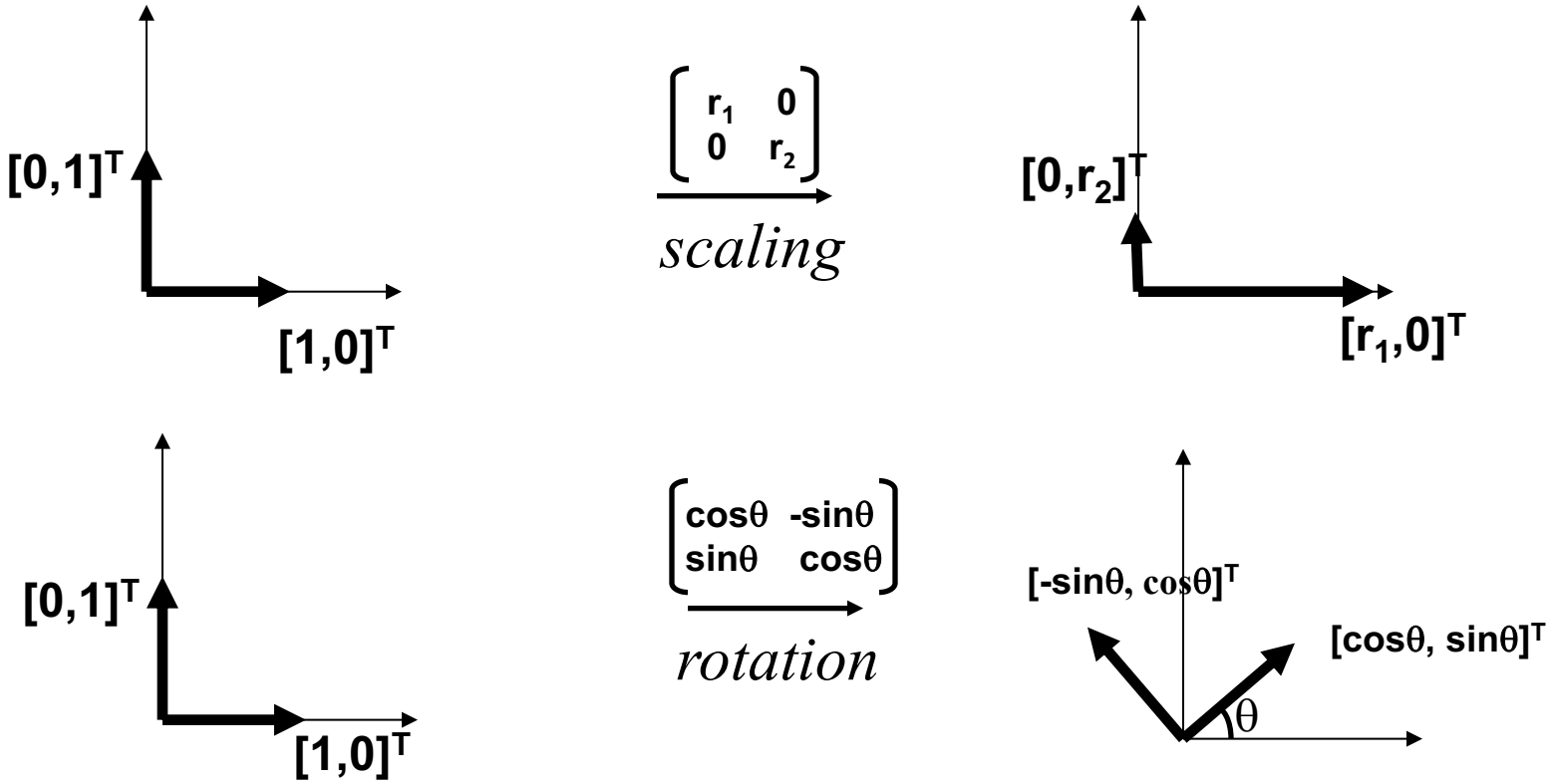
y’ = cx + dy

# Matrices: Scaling, Rotation, Identity

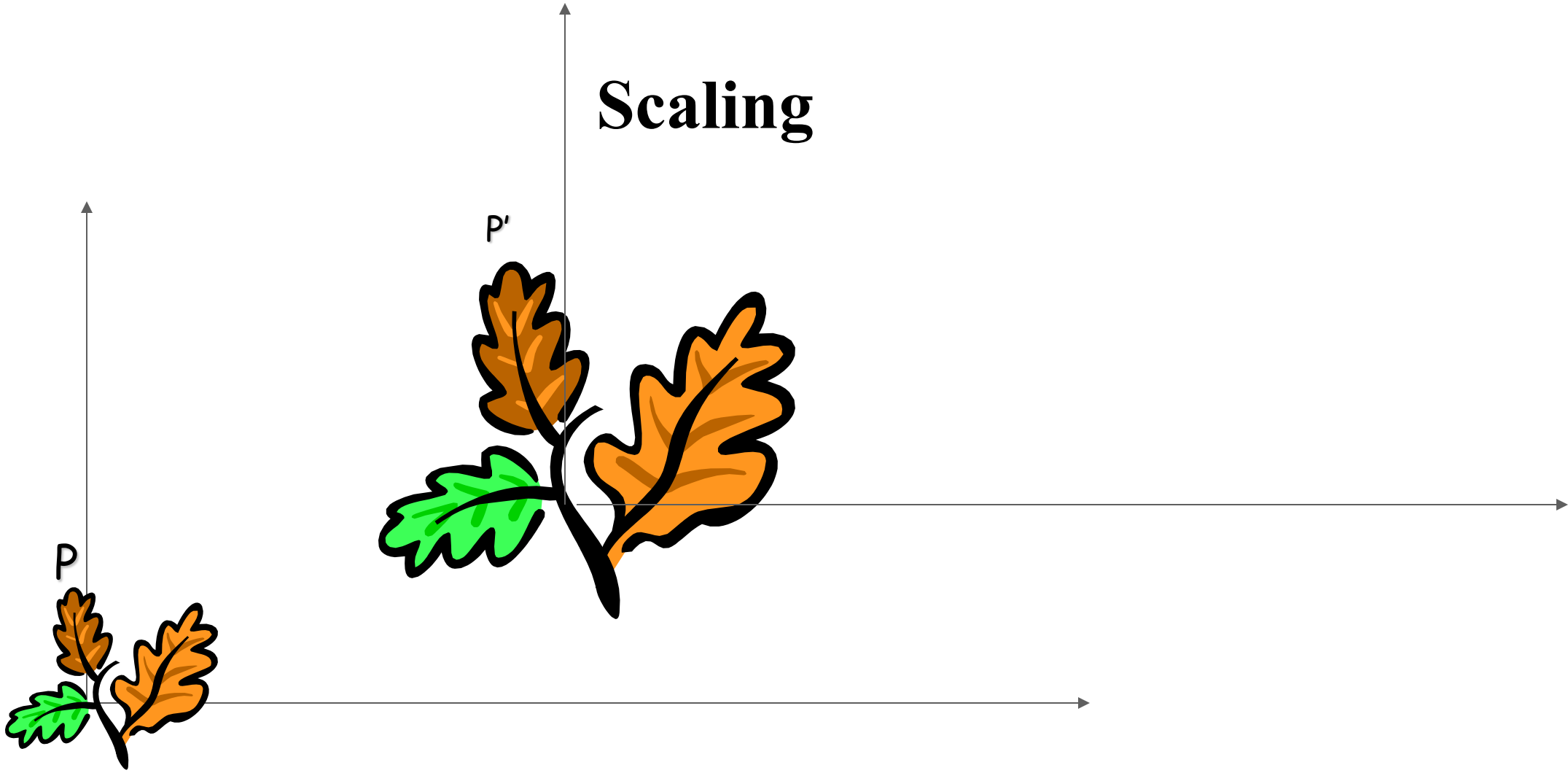
Pure scaling, no rotation => "**diagonal** matrix" (note: x-, y-axes could be scaled differently!)

Pure rotation, no stretching => "**orthogonal** matrix" **O**

**Identity** ("do nothing") matrix = unit scaling, no rotation!



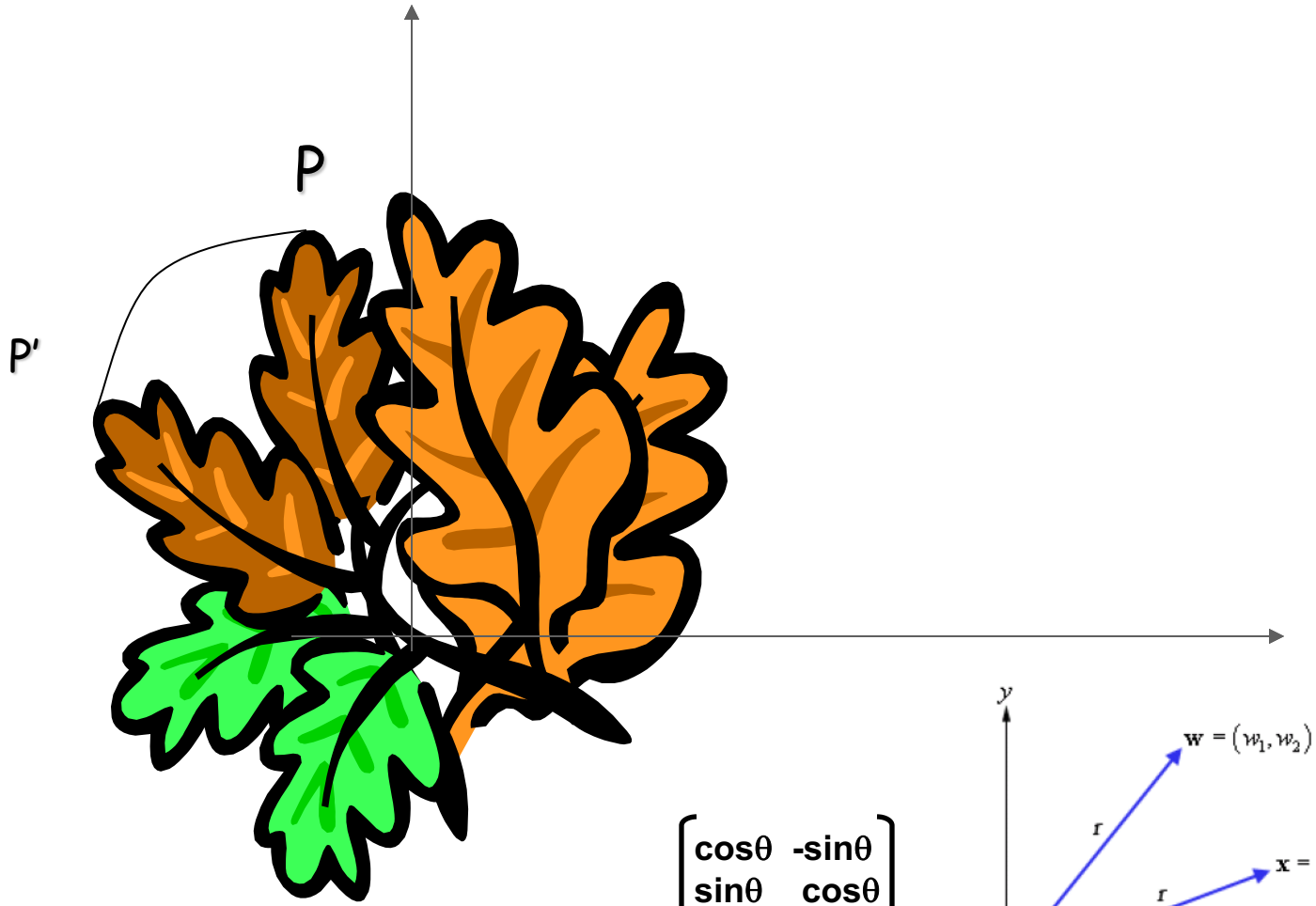
# Scaling




$$\begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix}$$

a.k.a: dilation (r >1),
contraction (r <1)

# Rotation




$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

# Inverse of a Matrix

Identity matrix:
**AI = A**

Inverse exists only for square matrices that are non-singular

- Maps N-d space to another N-d space bijectively

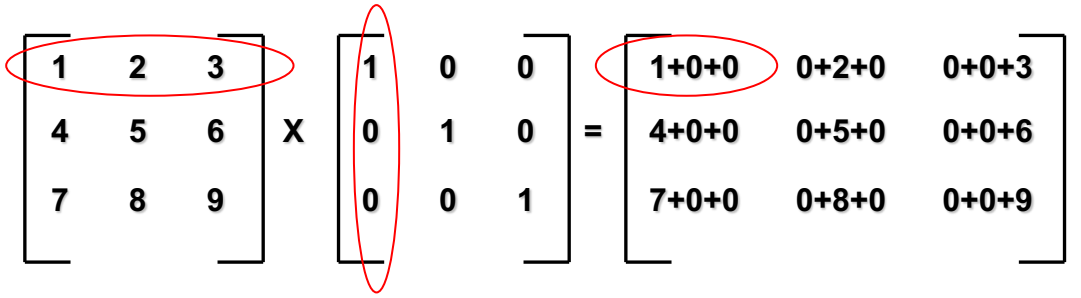
Some matrices have an inverse, such that:
$\mathbf{AA^{-1} = I}$

Inversion is tricky:
$\mathbf{(ABC)^{-1} = C^{-1}B^{-1}A^{-1}}$

Derived from non-commutativity property

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

# Example



# Vector Products

**Two vectors:** $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$

**Inner product $X^TY$ is a scalar**

**(1xn) (nx1)**

**Inner product** = scalar

$$\mathbf{x}^T\mathbf{y} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = x_1y_1 + x_2y_2 + x_3y_3 = \sum_{i=1}^{3} x_i y_i$$

**Outer product** = matrix

$$xy^T = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} = \begin{bmatrix} x_1y_1 & x_1y_2 & x_1y_3 \\ x_2y_1 & x_2y_2 & x_2y_3 \\ x_3y_1 & x_3y_2 & x_3y_3 \end{bmatrix}$$

**Outer product $XY^T$ is a matrix**

**(nx1) (1xn)**

# Determinant of a Matrix

Used for inversion

If det(A) = 0, then A has no inverse

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad \det(A) = ad - bc$$

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

http://www.euclideanspace.com/maths/algebra/matrix/functions/inverse/threeD/index.htm

# Coding Exercise #1

# numpy

The key to NumPy is the ndarray object, an *n*-dimensional array of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size. Modifying the size means creating a new array.
- NumPy arrays must be of the same data type, but this can include Python objects.
- More efficient mathematical operations than built-in sequence types.

# Numpy datatypes

To begin, NumPy supports a wider variety of data types than are built-in to the Python language by default. They are defined by the numpy.dtype class and include:

- intc (same as a C integer) and intp (used for indexing)
- int8, int16, int32, int64
- uint8, uint16, uint32, uint64
- float16, float32, float64
- complex64, complex128
- bool_, int_, float_, complex_ are shorthand for defaults.

These can be used as functions to cast literals or sequence types, as well as arguments to numpy functions that accept the dtype keyword argument.

# Numpy arrays

There are a couple of mechanisms for creating arrays in NumPy:

- Conversion from other Python structures (e.g., lists, tuples).
- Built-in NumPy array creation (e.g., arange, ones, zeros, etc.).
- Reading arrays from disk, either from standard or custom formats (e.g. reading in from a CSV file).
- and others …

# Numpy arrays

In general, any numerical data that is stored in an array-like container can be converted to an ndarray through use of the array() function. The most obvious examples are sequence types like lists and tuples.

```
>>> x = np.array([2,3,1,0])
>>> x = np.array([2, 3, 1, 0])
>>> x = np.array([[1,2.0],[0,0],(1+1j,3.)])
>>> x = np.array([[ 1.+0.j, 2.+0.j], [ 0.+0.j, 0.+0.j], [ 1.+1.j, 3.+0.j]])
```

# Numpy arrays

There are a couple of built-in NumPy functions which will create arrays from scratch.

- zeros(shape) -- creates an array filled with 0 values with the specified shape. The default dtype is float64.

```
>>> np.zeros((2, 3))
array([[ 0., 0., 0.], [ 0., 0., 0.]])
```

- ones(shape) -- creates an array filled with 1 values.
- arange() -- creates arrays with regularly incrementing values.

```
>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.arange(2, 10, dtype=np.float)
array([ 2., 3., 4., 5., 6., 7., 8., 9.])
>>> np.arange(2, 3, 0.1)
array([ 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9])
```

# Numpy arrays

- linspace() -- creates arrays with a specified number of elements, and spaced equally between the specified beginning and end values.

```
>>> np.linspace(1., 4., 6)
array([ 1. , 1.6, 2.2, 2.8, 3.4, 4. ])
```

- random.random(shape) – creates arrays with random floats over the interval [0,1).

```
>>> np.random.random((2,3))
array([[ 0.75688597, 0.41759916, 0.35007419],
       [ 0.77164187, 0.05869089, 0.98792864]])
```

# Numpy arrays

Printing an array can be done with the print statement.

```
>>> import numpy as np
>>> a = np.arange(3)
>>> print a
[0 1 2]
>>> a
array([0, 1, 2])
>>> b = np.arange(9).reshape(3,3)
>>> print b
[[0 1 2]
 [3 4 5]
 [6 7 8]]
>>> c = np.arange(8).reshape(2,2,2)
>>> print c
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
```

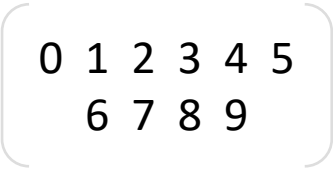
# indexing

Single-dimension indexing is accomplished as usual.

```
>>> x = np.arange(10)
>>> x[2]
2
>>> x[-2]
8
```


Multi-dimensional arrays support multi-dimensional indexing.

```
>>> x.shape = (2,5) # now x is 2-dimensional
>>> x[1,3]
8
>>> x[1,-1]
9
```



# Eigenvalues and Eigenvectors

Eigenvalue problem (one of the most important problems in the linear algebra):

If $A$ is an $n \times n$ matrix, do there exist nonzero vectors $\mathbf{x}$ in $R^n$ such that $A\mathbf{x}$ is a scalar multiple of $\mathbf{x}$?

(The term eigenvalue is from the German word *Eigenwert*, meaning "proper value")

$A$: an $n \times n$ matrix

$\lambda$: a scalar (could be **zero**)

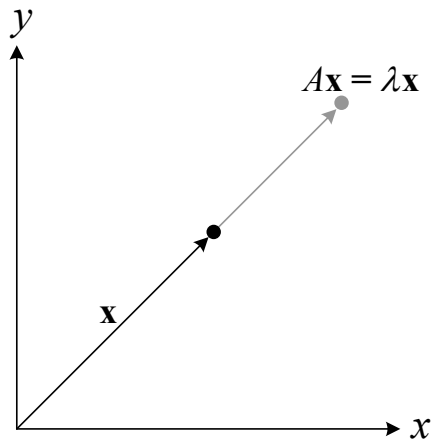
$\mathbf{x}$: a **nonzero** vector in $R^n$

Eigenvalue

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

Eigenvector

※ Geometric Interpretation



# Verifying eigenvalues and eigenvectors

$$A = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix} \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$A\mathbf{x}_1 = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} = 2\begin{bmatrix} 1 \\ 0 \end{bmatrix} = 2\mathbf{x}_1$$

Eigenvalue

Eigenvector

$$A\mathbf{x}_2 = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} = -1\begin{bmatrix} 0 \\ 1 \end{bmatrix} = (-1)\mathbf{x}_2$$

Eigenvalue

Eigenvector

In fact, for each eigenvalue, it has infinitely many eigenvectors. For $\lambda = 2$, $[3\ 0]^T$ or $[5\ 0]^T$ are both corresponding eigenvectors. Moreover, $([3\ 0] + [5\ 0])^T$ is still an eigenvector. The proof is in Thm. 7.1.

# Finding eigenvalues and eigenvectors

$$A = \begin{bmatrix} 2 & -12 \\ 1 & -5 \end{bmatrix}$$

Sol: Characteristic equation:

$$\det(\lambda I - A) = \begin{vmatrix} \lambda - 2 & 12 \\ -1 & \lambda + 5 \end{vmatrix}$$

$$= \lambda^2 + 3\lambda + 2 = (\lambda + 1)(\lambda + 2) = 0$$

$$\Rightarrow \lambda = -1, -2$$

Eigenvalue: $\lambda_1 = -1, \lambda_2 = -2$

# Matrix Determinant

# Singular Value Decomposition

**Singular values**: Non negative square roots of the eigenvalues of $\mathbf{A}^t\mathbf{A}$. Denoted $\sigma_i$, $i=1,\ldots,n$

SVD: If $\mathbf{A}$ is a real $m$ by $n$ matrix then there exist orthogonal matrices $\mathbf{U}$ ($\in \mathbb{R}^{m\times m}$) and $\mathbf{V}$ ($\in \mathbb{R}^{n\times n}$) such that

$$A = U\,\Sigma\,V^{-1} \qquad \mathrm{U}^{-1}\mathrm{AV} = \Sigma = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & . & \\ & & & \sigma_N \end{bmatrix}$$