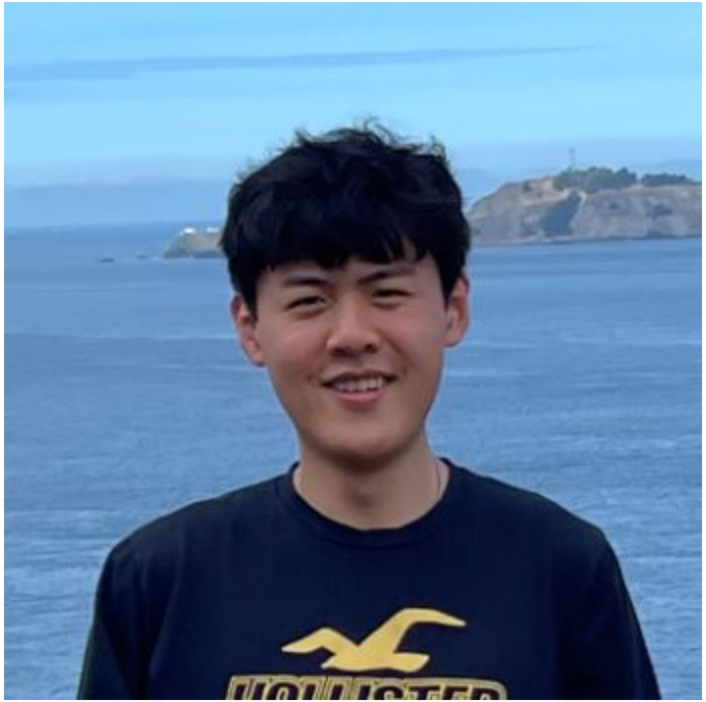
PSET0 + Python & Linear Algebra Review

CS 231A
04/04/2025

A bit about me!

- Coterm CS + Upcoming CS PhD!
- Work in SVL with Professors Fei-Fei Li and Ehsan Adeli
- Research interests in 3D computer vision for medical applications
- How can we extract insights about a 3D world from monocular video?
  - Monocular depth estimation
  - Scene reconstruction
  - Scene understanding
- Office Hours: Thursdays 2-3 PM in CoDa B43 (check Canvas for Zoom)



Outline

- Python Introduction
- Linear Algebra and NumPy
- PSET0

Outline

- **Python Introduction**
- Linear Algebra and NumPy
- PSET0

Python

High-level, interpreted programming language.

Python will be used in all the homeworks and recommended for the project.

We’ll cover some basics today.

Why Python?

- Python is high-level.

JAVA

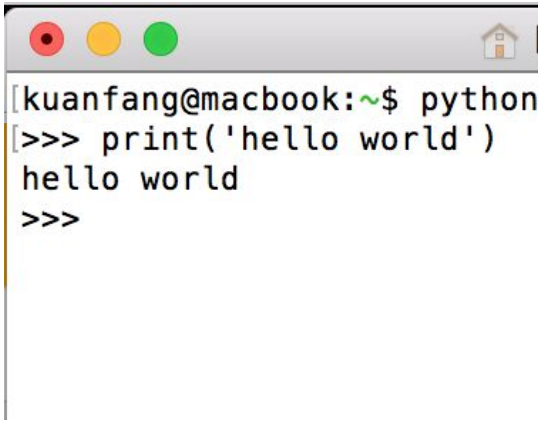
```
public class Main {
  public static void main(String[] args) {
     System.out.println("hello world");
   }
}
```

PYTHON

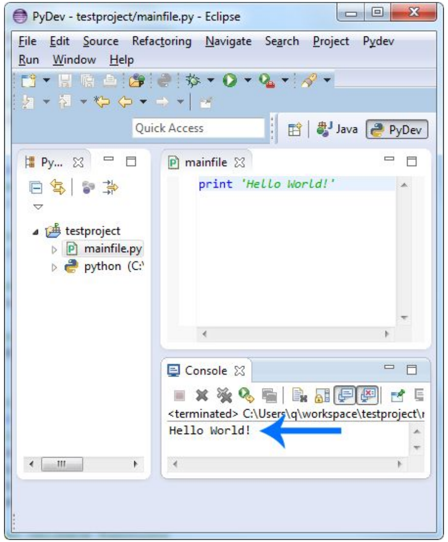
```
print('hello world')
```

Why Python?

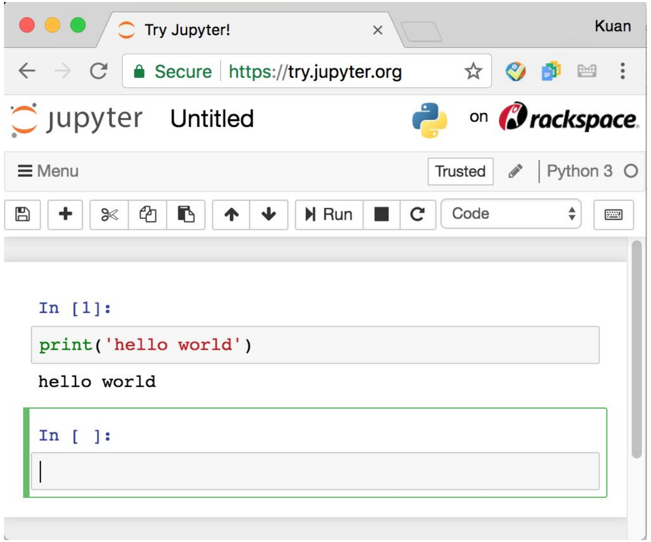
- Python is accessible.



Interpreter/Terminal



IDE



Jupyter Notebook

Why Python?

- Python has many many awesome packages.

How to Set up Python?

1. Follow this guide: https://wiki.python.org/moin/BeginnersGuide/Download
2. Choose your favourite editor or IDE:
   a. Sublime
   b. Vim
   c. VSCode
   d. Spyder
   e. PyCharm
   f. Jupyter Notebook
   g. Colab
   h. ...

Basic Python Review

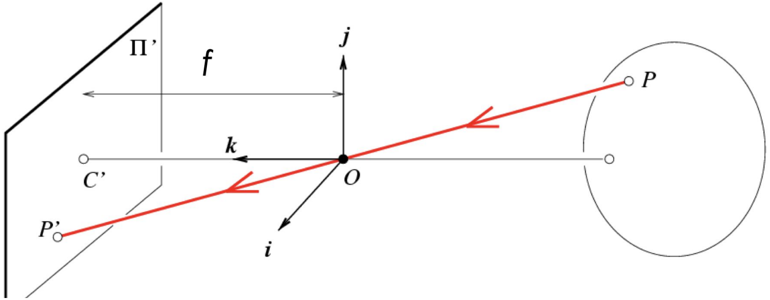
Google Colab Here

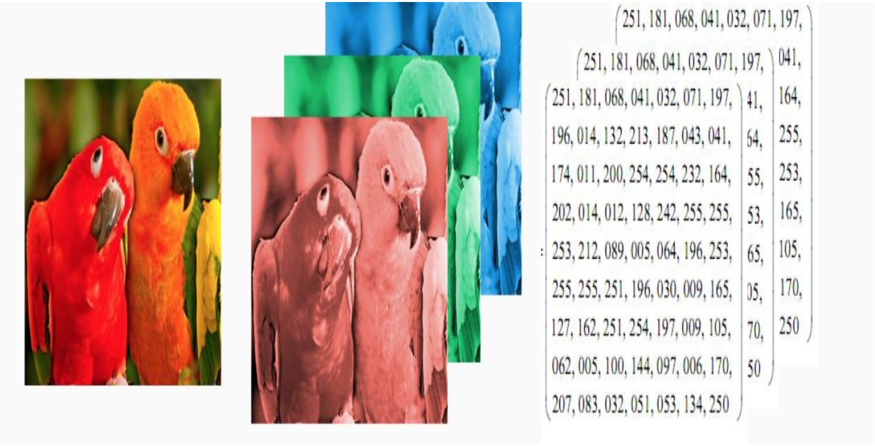
Outline

- Python Introduction
- **Linear Algebra and NumPy**
- PSET0

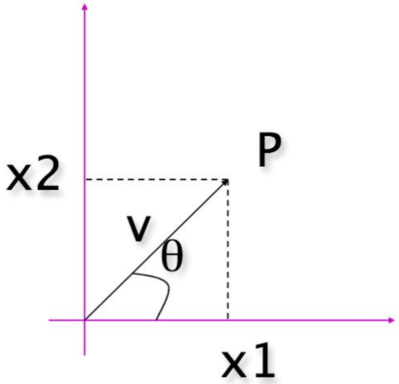
Why use Linear Algebra in Computer Vision?

- As you’ve seen in lecture, using linear algebra is necessary to represent many quantities, e.g. 3D points on a scene, 2D points on an image.
- Transformations of 3D points with 2D points can be represented as matrices.
- Images are literally matrices filled with numbers (as you will see in PSET0).





Vector Review



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

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

If $\|\mathbf{v}\| = 1$, $\mathbf{v}$ Is a UNIT vector

$$\frac{\mathbf{v}}{\|\mathbf{v}\|} = \left(\frac{x_1}{\|\mathbf{v}\|}, \frac{x_2}{\|\mathbf{v}\|} \right)$$ Is a unit vector

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

Vector Review

$$\mathbf{v} + \mathbf{w} = (x_1, x_2) + (y_1, y_2) = (x_1 + y_1, x_2 + y_2)$$

$$\mathbf{v} - \mathbf{w} = (x_1, x_2) - (y_1, y_2) = (x_1 - y_1, x_2 - y_2)$$

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

Matrix Review

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

Pixel's intensity value

Sum: $C_{n\times m} = A_{n\times m} + B_{n\times m}$ $\qquad c_{ij} = a_{ij} + b_{ij}$

A and B must have the same dimensions!

Example: $\begin{bmatrix} 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 6 & 2 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 8 & 7 \\ 4 & 6 \end{bmatrix}$

Matrices and Vectors in Python (NumPy)

```
import numpy as np
```

An optimized, well-maintained scientific computing package for Python.

As time goes on, you'll learn to appreciate NumPy more and more.

np.ndarray: Matrices and Vectors in Python

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```python
import numpy as np

M = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

v = np.array([[1],
              [2],
              [3]])
```

np.ndarray: Matrices and Vectors in Python

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```python
print(M.shape)  # (3, 3)
print(v.shape)  # (3, 1)
```

np.ndarray: Matrices and Vectors in Python

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```python
print(v + v)
```

```
[[2]
 [4]
 [6]]
```

```python
print(3 * v)
```

```
[[3]
 [6]
 [9]]
```

Other Ways to Create Matrices and Vectors

NumPy provides many convenience functions for creating matrices/vectors.

```
a = np.zeros((2,2))  # Create an array of all zeros
print a              # Prints "[[ 0.  0.]
                     #          [ 0.  0.]]"

b = np.ones((1,2))   # Create an array of all ones
print b              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7) # Create a constant array
print c               # Prints "[[ 7.  7.]
                      #          [ 7.  7.]]"

d = np.eye(2)        # Create a 2x2 identity matrix
print d              # Prints "[[ 1.  0.]
                     #          [ 0.  1.]]"

e = np.random.random((2,2)) # Create an array filled with random values
print e                     # Might print "[[ 0.91940167  0.08143941]
                            #               [ 0.68744134  0.87236687]]"
```

Matrix Indexing

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

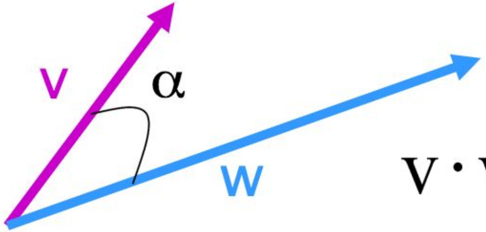
```python
print(M)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```python
print(M[:2, 1:3])
```

```
[[2 3]
 [5 6]]
```

Dot Product



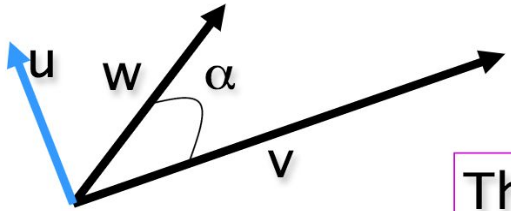
$$v \cdot w = (x_1, x_2) \cdot (y_1, y_2) = x_1 y_1 + x_2 y_2$$

The inner product is a SCALAR!

$$v \cdot w = (x_1, x_2) \cdot (y_1, y_2) = \| v \| \cdot \| w \| \cos\alpha$$

$$\text{if} \quad v \perp w, \quad v \cdot w = ? \;= 0$$

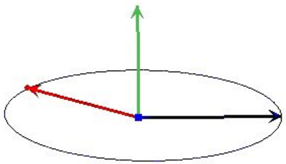
Cross Product



$$u = v \times w$$

The cross product is a **VECTOR!**

$$\text{Magnitude:} \|u\| = \|v \times w\| = \|v\| \|w\| \sin \alpha$$



Orientation:

$$u \perp v \Rightarrow u \cdot v = (v \times w) \cdot v = 0$$

$$u \perp w \Rightarrow u \cdot w = (v \times w) \cdot w = 0$$

$$\text{if} \quad \mathrm{v} // \mathrm{w}\,? \qquad \rightarrow \mathrm{u} = 0$$

Cross Product

$$\mathbf{i} = (1,0,0) \qquad \|\mathbf{i}\| = 1 \qquad \mathbf{i} = \mathbf{j} \times \mathbf{k}$$

$$\mathbf{j} = (0,1,0) \qquad \|\mathbf{j}\| = 1 \qquad \mathbf{j} = \mathbf{k} \times \mathbf{i}$$

$$\mathbf{k} = (0,0,1) \qquad \|\mathbf{k}\| = 1 \qquad \mathbf{k} = \mathbf{i} \times \mathbf{j}$$

$$\mathbf{u} = \mathbf{v} \times \mathbf{w} = (x_1, x_2, x_3) \times (y_1, y_2, y_3)$$

$$= (x_2 y_3 - x_3 y_2)\mathbf{i} + (x_3 y_1 - x_1 y_3)\mathbf{j} + (x_1 y_2 - x_2 y_1)\mathbf{k}$$

Matrix Multiplication

$$A_{n\times m} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \mathbf{a}_i \qquad B_{m\times p} = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mp} \end{bmatrix}$$

$$\mathbf{b}_j$$

Product:

$$C_{n\times p} = A_{n\times \boxed{m}} B_{\boxed{m}\times p}$$

$$c_{ij} = \mathbf{a}_i \cdot \mathbf{b}_j = \sum_{k=1}^{m} a_{ik} b_{kj}$$

A and B must have compatible dimensions!

$$A_{n\times n} B_{n\times n} \neq B_{n\times n} A_{n\times n}$$

Basic Operations - Dot Multiplication

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```python
print(M.dot(v))
```

```
[[14]
 [32]
 [50]]
```

Matrix multiplication in NumPy can be defined as the dot product between a matrix and a matrix/vector.

Basic Operations - Element-wise Multiplication

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
print(np.multiply(M, v))
```

```
[[ 1  2  3]
 [ 8 10 12]
 [21 24 27]]
```

```
print(np.multiply(v, v))
```

```
[[1]
 [4]
 [9]]
```

Broadcasting

- Between numpy arrays of different shapes
- Easy and more concise implementations
- Faster with parallel numerical computations

Broadcasting

https://numpy.org/doc/1.20/user/theory.broadcasting.html

```
>>> from numpy import array
>>> a = array([1.0, 2.0, 3.0])
>>> b = array([2.0, 2.0, 2.0])
>>> a * b
array([ 2.,  4.,  6.])
```

```
>>> from numpy import array
>>> a = array([1.0,2.0,3.0])
>>> b = 2.0
>>> a * b
array([ 2.,  4.,  6.])
```

Broadcasting

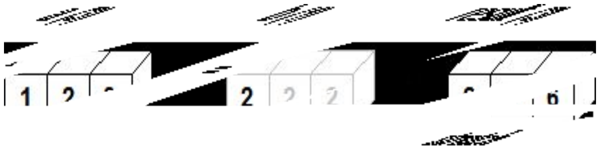
The broadcasting rule

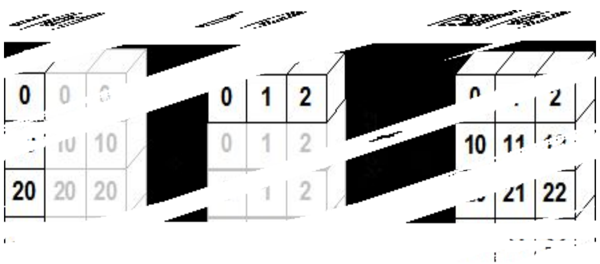
Image	(3d array)	256 x	256 x	3
Scale	(1d array)			3
Result	(3d array)	256 x	256 x	3

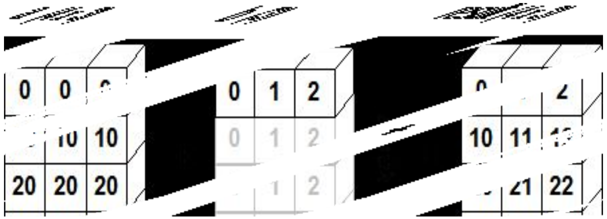
A	(4d array)	8 x	1 x	6 x	1
B	(3d array)		7 x	1 x	5
Result	(4d array)	8 x	7 x	6 x	5

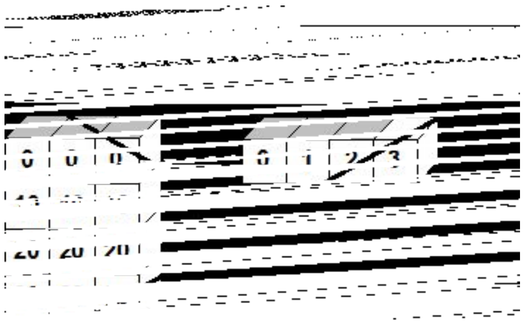
Broadcasting

The broadcasting rule



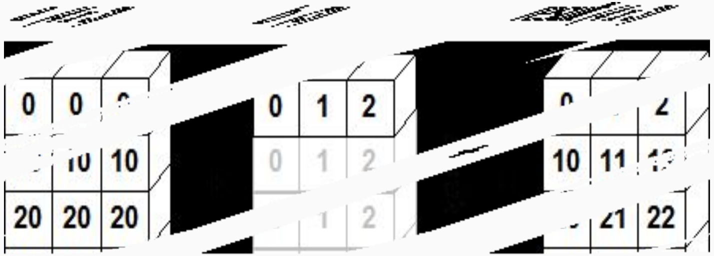






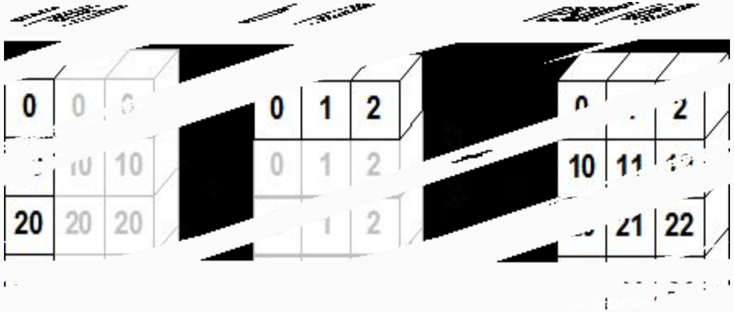
Broadcasting

```
>>> from numpy import array
>>> a = array([[ 0.0,  0.0,  0.0],
...            [10.0, 10.0, 10.0],
...            [20.0, 20.0, 20.0],
...            [30.0, 30.0, 30.0]])
>>> b = array([1.0, 2.0, 3.0])
>>> a + b
array([[  1.,   2.,   3.],
       [ 11.,  12.,  13.],
       [ 21.,  22.,  23.],
       [ 31.,  32.,  33.]])
```



Broadcasting

```
>>> from numpy import array, newaxis
>>> a = array([0.0, 10.0, 20.0, 30.0])
>>> b = array([1.0, 2.0, 3.0])
>>> a[:,newaxis] + b
array([[  1.,   2.,   3.],
       [ 11.,  12.,  13.],
       [ 21.,  22.,  23.],
       [ 31.,  32.,  33.]])
```



Broadcasting

Match the dimension of two arrays

Add new axis: np.newaxis or None

Examples: arr[None], arr[:, None], arr[:, :, None, :],

arr[..., None], arr[..., None, :]

Repeat an array: np.repeat(), np.tile()

Norm

- Informally a measure of the "length" of a vector
- More formally, any measure f(x) that is:
    - Non-negative: f(x) >= 0 for all x.
    - Definite: f(x) = 0 iif and only if x = 0
    - Homogeneous: f(tx) = |t|f(x)
    - Triangle inequality: f(x + y) <= f(x) + f(y)
- There are also norms for matrices like the Frobenius norm:

$$\|A\|_F \equiv \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$$

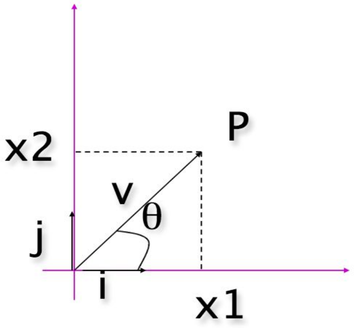
$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$$

Orthonormal Basis

= <u>Ortho</u>gonal and <u>Normal</u>ized Basis



$$\mathbf{i} = (1,0) \qquad \|\mathbf{i}\| = 1$$

$$\mathbf{j} = (0,1) \qquad \|\mathbf{j}\| = 1$$

$$\mathbf{i} \cdot \mathbf{j} = 0$$

$$\mathbf{v} = (x_1, x_2) \qquad \mathbf{v} = \mathrm{x}_1\mathbf{i} + \mathrm{x}_2\mathbf{j}$$

$$\mathbf{v} \cdot \mathbf{i} = ? \;= (\mathrm{x}_1\mathbf{i} + \mathrm{x}_2\mathbf{j}) \cdot \mathbf{i} = \mathrm{x}_1 1 + \mathrm{x}_2 0 = \mathrm{x}_1$$

$$\mathbf{v} \cdot \mathbf{j} = (\mathrm{x}_1\mathbf{i} + \mathrm{x}_2\mathbf{j}) \cdot \mathbf{j} = \mathrm{x}_1.0 + \mathrm{x}_2.1 = \mathrm{x}_2$$

Transpose

Definition:

$$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}^T = \begin{bmatrix} 6 & 1 \\ 4 & -9 \\ 24 & 8 \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$$

$$\mathbf{C}_{m \times n} = \mathbf{A}^T_{n \times m}$$

$$c_{ij} = a_{ji}$$

Identities:

$$(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$$

$$(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$$

If $\mathbf{A} = \mathbf{A}^T$, then $\mathbf{A}$ is *symmetric*

Basic Operations - Transpose

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
print(M.T)
```

```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

```
print(v.T)
```

```
[[1 2 3]]
```

```
print(M.T.shape)
print(v.T.shape)
```

```
(3, 3)
(1, 3)
```

Matrix Determinant

Useful value computed from the elements of a *square* matrix **A**

$$\det \begin{bmatrix} a_{11} \end{bmatrix} = a_{11}$$

$$\det \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

$$\det \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32}$$

$$- a_{13}a_{22}a_{31} - a_{23}a_{32}a_{11} - a_{33}a_{12}a_{21}$$

Matrix Inverse

Does not exist for all matrices, necessary (but not sufficient) that the matrix is square

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$$

$$\mathbf{A}^{-1} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}^{-1} = \frac{1}{\det \mathbf{A}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}, \det \mathbf{A} \neq 0$$

If $\det \mathbf{A} = 0$, $\mathbf{A}$ does not have an inverse.

Basic Operations - Determinant and Inverse

$$M = \begin{bmatrix} 3 & 0 & 2 \\ 2 & 0 & -2 \\ 0 & 1 & 1 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
print(np.linalg.inv(M))
```

```
[[ 0.2   0.2   0.  ]
 [-0.2   0.3   1.  ]
 [ 0.2  -0.3  -0.  ]]
```

```
print(np.linalg.det(M))
```

```
10.0
```

Matrix Eigenvalues and Eigenvectors

A eigenvalue λ and eigenvector $\mathbf{u}$ satisfies

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

where $\mathbf{A}$ is a square matrix.

- Multiplying $\mathbf{u}$ by $\mathbf{A}$ scales $\mathbf{u}$ by λ

Matrix Eigenvalues and Eigenvectors

Rearranging the previous equation gives the system

$$\mathbf{Au} - \lambda\mathbf{u} = (\mathbf{A} - \lambda\mathbf{I})\mathbf{u} = 0$$

which has a solution if and only if $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$.

- The eigenvalues are the roots of this determinant which is polynomial in λ.
- Substitute the resulting eigenvalues back into $\mathbf{Au} = \lambda\mathbf{u}$ and solve to obtain the corresponding eigenvector.

Basic Operations - Eigenvalues, Eigenvectors

$$M = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

```
eigvals, eigvecs = np.linalg.eig(M)
```

```
print(eigvals)
```

```
[-1. -2.]
```

```
print(eigvecs)
```

```
[[ 0.70710678 -0.4472136 ]
 [-0.70710678  0.89442719]]
```

NOTE: Please read the NumPy docs on this function before using it, lots more information about multiplicity of eigenvalues and etc there.

Facts of Eigenvalues

Q1: If an nxn square matrix is real, does it always have n eigenvalues?

Q2: If an nxn real square matrix is symmetric, are all its eigenvalues real?

Q3: For any mxn matrix A, does AA^T have to be symmetric?

Facts of Eigenvalues

Q1: If an nxn square matrix is real, does it always have n eigenvalues?

Yes. But they may not all be real.

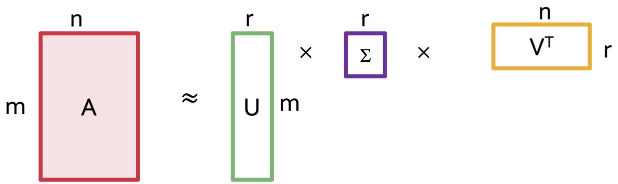
Q2: If an nxn real square matrix is symmetric, are all its eigenvalues real?

Yes.

Q3: For any mxn matrix A, does AA^T have to be symmetric?

Yes.

Singular Value Decomposition



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

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

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

Singular Value Decomposition

- SVD is ALWAYS possible on a real matrix A
- ***U***, Σ, ***V*** unique
- ***U*** and ***V*** are column orthonormal
  - $U^T U = I$, $V^T V = I$
  - Columns are orthogonal unit vectors
- Σ: Diagonal
  - Entries are nonzero and sorted in descending order

Singular Value Decomposition

$$M = \begin{bmatrix} 3 & 0 & 2 \\ 2 & 0 & -2 \\ 0 & 1 & 1 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
U, S, V_transpose = np.linalg.svd(M)
```

```
print(U)
```

```
[[-0.95123459  0.23048583 -0.20500982]
 [-0.28736244 -0.90373717  0.31730421]
 [-0.11214087  0.36074286  0.92589903]]
```

```
print(S)
```

```
[ 3.72021075  2.87893436  0.93368567]
```

```
print(V_transpose)
```

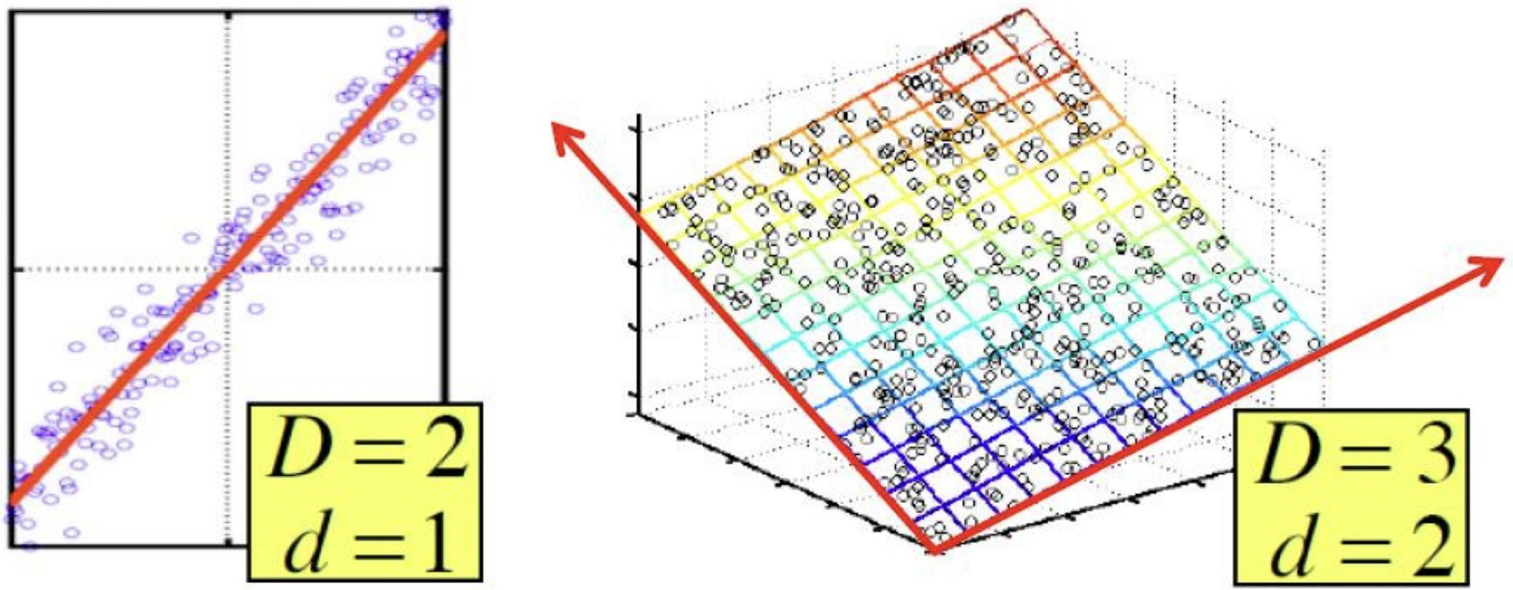
```
[[-0.9215684  -0.03014369 -0.38704398]
 [-0.38764928  0.1253043   0.91325071]
 [ 0.02096953  0.99166032 -0.12716166]]
```

Recall SVD is the factorization of a matrix into the product of 3 matrices, and is formulated like so:

$$M = U\Sigma V^T$$

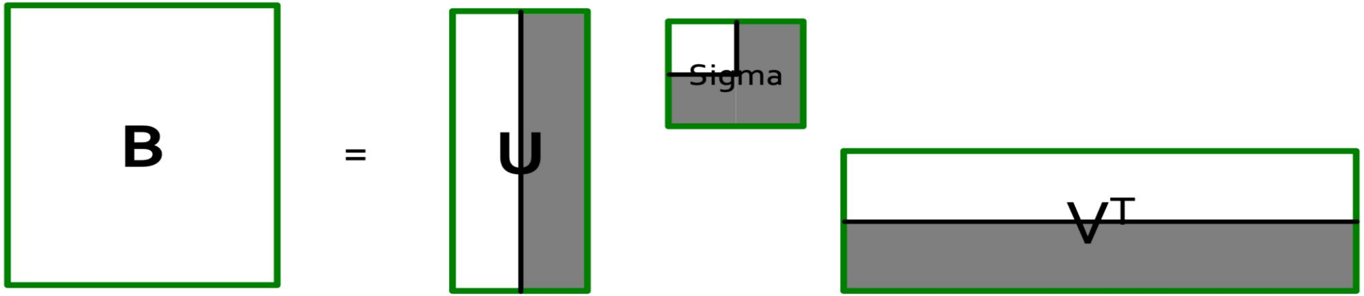
Caution: The notation of SVD in NumPy is slightly different. Here V is actually V^T in the common notation.

Dimensionality Reduction



A fun fact

SVD provides the best rank-k approximation of A (in terms of reconstruction error)!



In other words: **B** is a solution to $\min_B ||A - B||_F$

Note: A nice proof of this can be found in Section 11.3.4 of Jure Leskovec's MMDS!

More Information

Python Documentation:https://docs.python.org/3/

NumPy Documentation: https://numpy.org/doc/2.2/reference/index.html

CS 229 Linear Algebra Tutorial: : https://cs229.stanford.edu/section/cs229-linalg.pdf

CS231N Python Tutorial: http://cs231n.github.io/python-numpy-tutorial/

Office hours! Ed! The Internet!

PSET0 Overview

Announcements

- Lecture on Mon and Wed virtual – TAs in person!
- PSET 1 Released!
- Project partner searching thread on Ed

Thanks!

Questions?