Python Introduction & Linear Algebra Review

Kuan Fang

CS 231A

1/12/2018

Outline

- Python Introduction
- Linear Algebra Review + NumPy

Python Review

Python

High-level, interpreted programming language

You should already be proficient in programming

- Being proficient with Python is a plus, but not strictly necessary

We'll cover some basics today

Why Python?

- Python is high-level.

JAVA

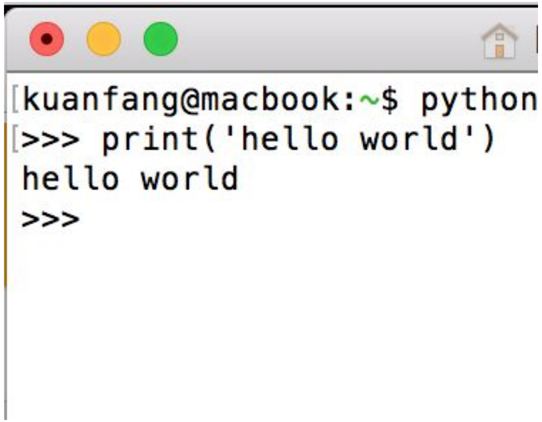
```java
public class Main {
  public static void main(String[] args) {
     System.out.println("hello world");
   }
}
```

PYTHON

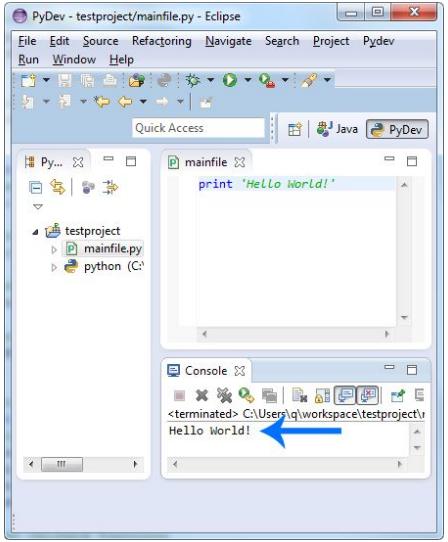
```python
print(‘hello world’)
```

Why Python?

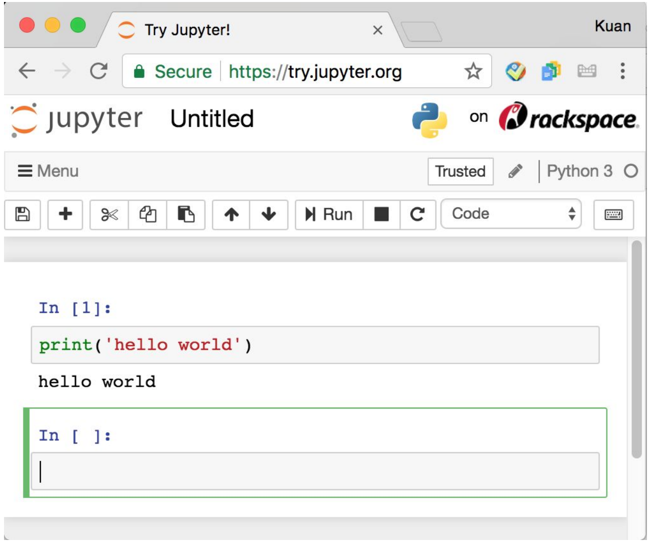
- Python is accessible.



Interpreter/Terminal



IDE



Jupyter Notebook

Why Python?

- Python has many many awesome packages.

How to Set up Python?

1. Find a computer:
   a. Your Linux/Mac/Windows/... machines.
   b. Use Stanford Corn machines: https://web.stanford.edu/group/farmshare/cgi-bin/wiki/index.php/Main_Page
   c. iMac computers in Stanford Libraries.
2. Follow this guide: https://wiki.python.org/moin/BeginnersGuide/Download
3. Choose your favourite editor or IDE:
   a. Sublime
   b. Notepad++
   c. Vim
   d. Spyder
   e. PyCharm
   f. Eclipse
   g. Jupyter Notebook
   h. ...

Variable

```python
a = 6
b = 7.0
c = a + b
print(c)
```

```
13.0
```

```python
string_var = 'Hello World!'
print(string_var)
```

```
Hello World!
```

Comment

```
# This line is comment.
a = 5  # After the number sign it is also comment.
a = a + 1

"""
Some times we also use three double quotation marks
for a large piece of comments.

This is usually used at the beginning of a file, a
class, or a function.
"""
```

List

```
list_var = []
print(list_var)        # []
print(len(list_var))   # 0
```

```
list_var.append(1)
list_var.append(42)
print(list_var)        # [1, 42]
print(len(list_var))   # 2
```

List

```python
list_var = [0, 1, 2, 3, 4]
print(list_var)          # [0, 1, 2, 3, 4]
```

```python
list_var = range(5)
print(list_var)          # [0, 1, 2, 3, 4]
```

List

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

print(list_var)  # [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

List Indexing

```python
list_var = [0, 1, 2, 3, 4]
print(list_var[0])     # 0
print(list_var[0:2])  # [0, 1]
print(list_var[1:3])  # [1, 2]
```

```python
list_var = [0, 1, 2, 3, 4]
print(list_var[2:])      # [2, 3, 4]
print(list_var[:2])      # [0, 1]
print(list_var[0:4:2])  # [0, 2]
print(list_var[-1])      # 4
```

List Indexing

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

print(list_var[2][0:2])  # [7, 8]
```

Dictionary (Similar to Map in Java/C++)

```
dict_var = {}
print(dict_var)  # {}
```

```
dict_var['a'] = 'hello'
dict_var['b'] = 'world'
print(dict_var)  # {'a': 'hello', 'b': 'world'}
```

Dictionary

```
dict_var = {'a': 'hello', 'b': 'world'}
print(dict_var)  # {'a': 'hello', 'b': 'world'}
```

Dictionary Indexing

```python
dict_var = {'a': 'hello', 'b': 'world'}
print(dict_var['a'])  # hello
print(dict_var['b'])  # world
```

```python
dict_var = {'a': 'hello', 'b': 'world'}
print(dict_var.keys())    # ['a', 'b']
print(dict_var.values())  # ['hello', 'world']
```

Control Flow

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

Control Flow

```python
i = 11

if i < 10:
    print('small')
elif i < 100:
    print('medium')
else:
    print('large')
```

```
medium
```

List Comprehension

```
list_var = [i * i for i in range(5)]
print(list_var)
```

```
[0, 1, 4, 9, 16]
```

```
list_var = [i * i for i in range(5) if i % 2 == 0]
print(list_var)
```

```
[0, 4, 16]
```

Function

```
def add_numbers(a, b):
    return a + b

result = add_numbers(3, 4)
print(result)  # 7
```

Linear Algebra Review + Numpy

Why use Linear Algebra in Computer Vision?

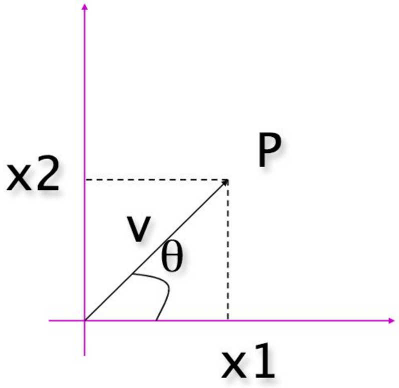
As you’ve seen in lecture, it’s useful to represent many quantities, e.g. 3D points on a scene, 2D points on an image.

Transformations of 3D points with 2D points can be represented as matrices.

Images are literally matrices filled with numbers (as you will see in HW0).

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

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

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

*Courtesy of last year’s slides.

Vector Review

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

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

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

*Courtesy of last year’s slides.

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

A and B must have the same dimensions!

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

*Courtesy of last year's slides.

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.

Years later I'm **still** learning new things about it!

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

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

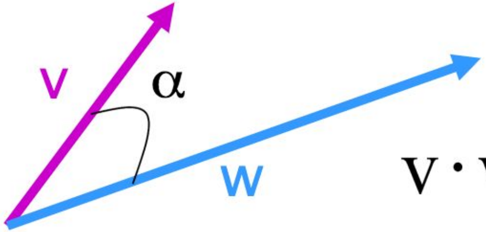
```python
print(M)
```

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

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

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

Dot Product



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

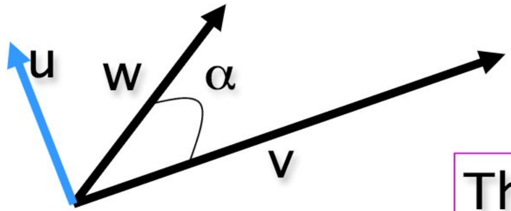
The inner product is a SCALAR!

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

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

*Courtesy of last year’s slides.

Cross Product



$$u = v \times w$$

The cross product is a VECTOR!

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

Orientation:

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

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

if $\quad v // w\,? \qquad \rightarrow u = 0$

*Courtesy of last year's slides.

Cross Product

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

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

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

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

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

*Courtesy of last year’s slides.

Matrix Multiplication

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

$$B_{m\times p} = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mp} \end{bmatrix}$$

$$\mathbf{b}_j$$

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!

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

*Courtesy of last year’s slides.

Basic Operations - Dot Multiplication

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

```
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))
```

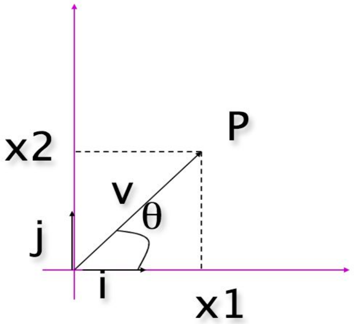
```
[[ 1  2  3]
 [ 8 10 12]
 [21 24 27]]
```

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

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

Orthonormal Basis

= Orthogonal and Normalized Basis



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

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

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

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

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

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

*Courtesy of last year’s slides.

Transpose

Definition:

$$\mathbf{C}_{m\times n} = \mathbf{A}_{n\times m}^{T}$$
$$c_{ij} = a_{ji}$$

Identities:

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

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

*Courtesy of last year's slides.

Basic Operations - Transpose

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

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

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

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

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

```python
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**

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

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

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

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

*Courtesy of last year's slides.

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.

*Courtesy of last year's slides.

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

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

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

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

*Courtesy of last year's slides.

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.

*Courtesy of last year's slides.

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]]
```

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

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

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

*Courtesy of last year's slides.

Singular Value Decomposition

Suppose we know the singular values of **A** and we know r are non zero

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq \sigma_{r+1} = \ldots = \sigma_p = 0$$

- Rank(**A**) = r.
- Null(**A**) = span$\{\mathbf{v_{r+1}},\ldots,\mathbf{v_n}\}$
- Range(**A**)=span$\{\mathbf{u_1},\ldots,\mathbf{u_r}\}$

$||\boldsymbol{A}||_F^2 = \sigma_1^2 + \sigma_2^2 + \ldots + \sigma_p^2$ $\qquad ||A||_2 = \sigma_1$

Numerical rank: If k singular values of A are larger than a given number ε. Then the ε rank of A is k.

Distance of a matrix of rank n from being a matrix of rank $k = \sigma_{k+1}$

*Courtesy of last year's slides.

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 <u>common notation</u>.

More Information

Python Documentation: https://docs.python.org/2/index.html

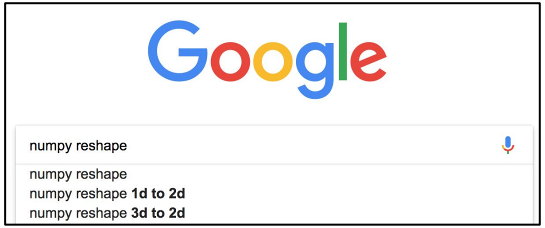
NumPy Documentation: https://docs.scipy.org/doc/numpy-1.13.0/user/index.html

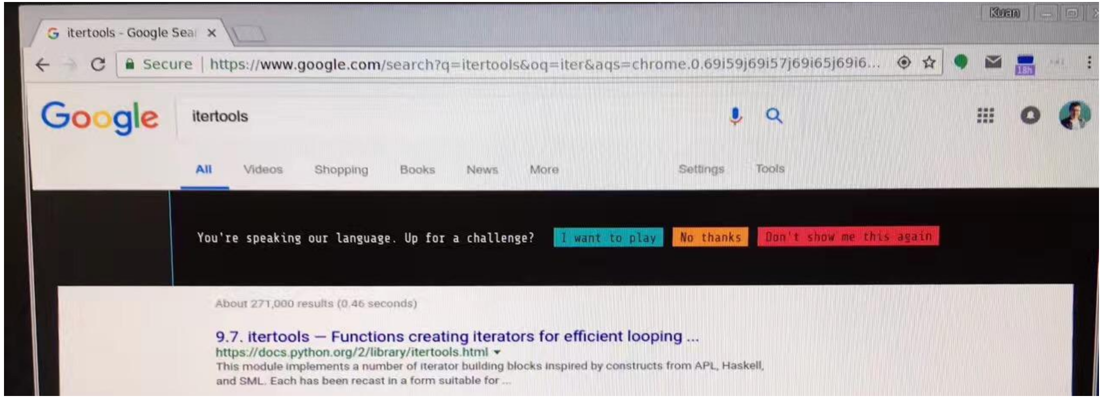
The Matrix Cookbook: https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

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

Office hours!

The rest of the internet!





Thanks!

Questions