Aim: To implement

(a) Matrix operations (using vectorization),

(b) transformation using python and

(c) SVD using Python.

import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array

print("type: %s" %type(a))            # Prints "<class 'numpy.ndarray'>"

print("shape: %s" %a.shape)            # Prints "(3,)"

print(a[0], a[1], a[2])   # Prints "1 2 3"

a[0] = 5                  # Change an element of the array

print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array

print("\n shape of b:",b.shape)                     # Prints "(2, 3)"

print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"

a = np.zeros((2,2))   # Create an array of all zeros

print("All zeros matrix:\n  %s" %a)              # Prints "[[ 0.  0.]

                      #          [ 0.  0.]]"

b = np.ones((1,2))    # Create an array of all ones

print("\nAll ones matrix:\n  %s" %b)              # Prints "[[ 1.  1.]]"

d = np.eye(2)         # Create a 2x2 identity matrix

print("\n identity matrix: \n%s"%d)              # Prints "[[ 1.  0.]

                      #          [ 0.  1.]]"

e = np.random.random((2,2))  # Create an array filled with random values

print("\n random matrix: \n%s"%e)

output:

type: <class 'numpy.ndarray'>

shape: 3

1 2 3

[5 2 3]

shape of b: (2, 3)

1 2 4

All zeros matrix:

[[0. 0.]

[0. 0.]]

All ones matrix:

[[1. 1.]]

identity matrix:

[[1. 0.]

[0. 1.]]

random matrix:

[[0.69537762 0.23928727]

[0.45105123 0.27090952]]

#vectorized sum

print("Vectorized sum example\n")

x = np.array([[1,2],[3,4]])

print("x:\n %s" %x)

print("sum: %s"%np.sum(x))  # Compute sum of all elements; prints "10"

print("sum axis = 0: %s" %np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"

print(" sum axis = 1: %s" %np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]"

#matrix dot product

a = np.arange(10000)

b = np.arange(10000)

dp = np.dot(a,b)

print("Dot product: %s\n" %dp)

#outer product

op = np.outer(a,b)

print("\n Outer product: %s\n" %op)

#elementwise product

ep = np.multiply(a, b)

print("\n Element Wise product: %s \n" %ep)

Out put:

Vectorized sum example

x:

[[1 2]

[3 4]]

sum: 10

sum axis = 0: [4 6]

sum axis = 1: [3 7]

Dot product: 333283335000

Outer product: [[ 0 0 0 ... 0 0 0]

[ 0 1 2 ... 9997 9998 9999]

[ 0 2 4 ... 19994 19996 19998]

...

[ 0 9997 19994 ... 99940009 99950006 99960003]

[ 0 9998 19996 ... 99950006 99960004 99970002]

[ 0 9999 19998 ... 99960003 99970002 99980001]]

Element Wise product: [ 0 1 4 ... 99940009 99960004 99980001]

import numpy as np

x = np.array([[1,2], [3,4]])

print("Original x: \n%s " %x)    # Prints "[[1 2]

            #          [3 4]]"

print("\nTranspose of x: \n%s" %x.T)  # Prints "[[1 3]

            #          [2 4]]"

Out put:

Original x:

[[1 2]

[3 4]]

Transpose of x:

[[1 3]

[2 4]]

# Singular-value decomposition

from numpy import array

from scipy.linalg import svd

# define a matrix

A = array([[1, 2], [3, 4], [5, 6]])

print("A: \n%s" %A)

# SVD

U, s, VT = svd(A)

print("\nU: \n%s" %U)

print("\ns: \n %s" %s)

print("\nV^T: \n %s" %VT)

output:

A:

[[1 2]

[3 4]

[5 6]]

U:

[[-0.2298477 0.88346102 0.40824829]

[-0.52474482 0.24078249 -0.81649658]

[-0.81964194 -0.40189603 0.40824829]]

s:

[9.52551809 0.51430058]

V^T:

[[-0.61962948 -0.78489445]

[-0.78489445 0.61962948]]

**Definition: Inner and Outer Product If u and v are column vectors with the same size, then u T v is the inner product of u and v; if u and v are column vectors of any size, then uvT is the outer product of u and v.**

**1.Data visualisation using matplotlib**

# initializing the data

x = [10, 20, 30, 40]

y = [20, 25, 35, 55]

# plotting the data

plt.plot(x, y)

plt.show()

**# Adding title to the plot**

plt.title("Linear graph")

plt.show()

plt.title("Linear graph", fontsize=25, color="green")

plt.show()

**Adding X-label, Y-Label**

**X label and the Y label are the titles given to X-axis and Y-axis respectively**

# Adding label on the y-axis

plt.ylabel('Y-Axis')

# Adding label on the x-axis

plt.xlabel('X-Axis')

plt.show()

**#Setting the limit of y-axis**

plt.ylim(0, 80)

# setting the labels of x-axis

plt.xticks(x, labels=["one", "two", "three", "four"])

plt.show()

**# Adding legends**

plt.legend(["GFG"])

plt.show()

**Add Axes to a Figure**

Import libraries

import matplotlib

import numpy

# Create figure() objects

# This acts as a container

# for the different plots

fig = matplotlib.pyplot.figure()

# Generate line graph

x = numpy.arange(0, 1.414\*2, 0.05)

y1 = numpy.sin(x)

y2 = numpy.cos(x)

# Creating two axes

# add\_axes([xmin,ymin,dx,dy])

axes1 = fig.add\_axes([0, 0, 1, 1])

axes1.plot(x, y1)

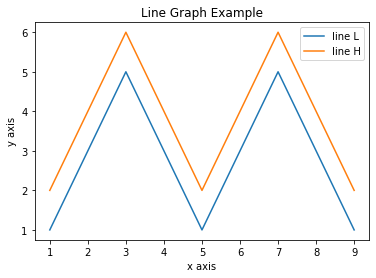
axes2 = fig.add\_axes([0, 1, 1, 1])

axes2.plot(x, y2)

# Show plot

plt.show()

Line Plots



import matplotlib.pyplot as plt

x  = [1, 2, 3, 4, 5, 6, 7, 8, 9]

y1 = [1, 3, 5, 3, 1, 3, 5, 3, 1]

y2 = [2, 4, 6, 4, 2, 4, 6, 4, 2]

plt.plot(x, y1, label="line L")

plt.plot(x, y2, label="line H")

plt.plot()

plt.xlabel("x axis")

plt.ylabel("y axis")

plt.title("Line Graph Example")

plt.legend()

plt.show()

Bar Plots

import matplotlib.pyplot as plt

# Look at index 4 and 6, which demonstrate overlapping cases.

x1 = [1, 3, 4, 5, 6, 7, 9]

y1 = [4, 7, 2, 4, 7, 8, 3]

x2 = [2, 4, 6, 8, 10]

y2 = [5, 6, 2, 6, 2]

# Colors: https://matplotlib.org/api/colors\_api.html

plt.bar(x1, y1, label="Blue Bar", color='b')

plt.bar(x2, y2, label="Green Bar", color='g')

plt.plot()

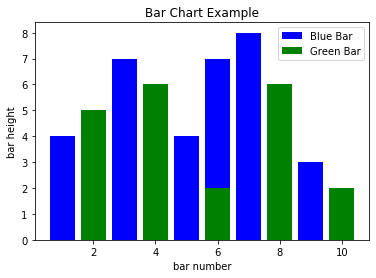
plt.xlabel("bar number")

plt.ylabel("bar height")

plt.title("Bar Chart Example")

plt.legend()

plt.show()



**Histograms**

import matplotlib.pyplot as plt

import numpy as np

# Use numpy to generate a bunch of random data in a bell curve around 5.

n = 5 + np.random.randn(1000)

m = [m for m in range(len(n))]

plt.bar(m, n)

plt.title("Raw Data")

plt.show()

plt.hist(n, bins=20)

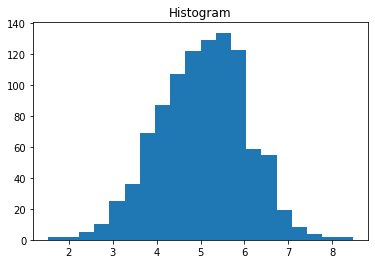
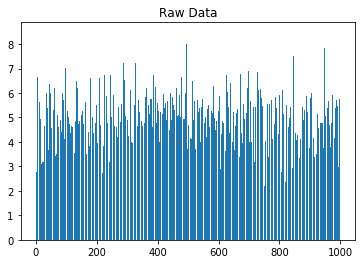
plt.title("Histogram")

plt.show()

plt.hist(n, cumulative=True, bins=20)

plt.title("Cumulative Histogram")

plt.show()



Scatter Plots

import matplotlib.pyplot as plt

x1 = [2, 3, 4]

y1 = [5, 5, 5]

x2 = [1, 2, 3, 4, 5]

y2 = [2, 3, 2, 3, 4]

y3 = [6, 8, 7, 8, 7]

# Markers: https://matplotlib.org/api/markers\_api.html

plt.scatter(x1, y1)

plt.scatter(x2, y2, marker='v', color='r')

plt.scatter(x2, y3, marker='^', color='m')

plt.title('Scatter Plot Example')

plt.show()

