**Advanced Statistical Algorithms MA691**

**Assignment 2**

Name**: Naman Goyal**

Roll No: **180123029**

Ques.**1**

**Code:**

**import numpy as np**

**print("------------------Q1----------------\n")**

**# generating initial normal random sample of size 10 with mean 0, variance 1**

**initSample = []**

**for i in range(10):**

**initSample.append(np.random.normal(0, 1))**

**# resampling with replacement and generating new sample**

**newSample = np.random.choice(initSample, 10)**

**# Regenerating 1000 more samples for new case**

**# now verification**

**# generate 1000 more samples and check the average number of unique elements from the original sample**

**uniqueNo = []**

**for i in range(1000):**

**sample = np.random.choice(initSample, 10)**

**x = np.unique(sample)**

**uniqueNo.append(len(x)/10)**

**print("For Original Sample")**

**print("Original Sample: ", initSample)**

**print("Mean: ", np.mean(initSample))**

**print("Variance: ", np.var(initSample))**

**print("---------------------------------\n")**

**print("For New Sample")**

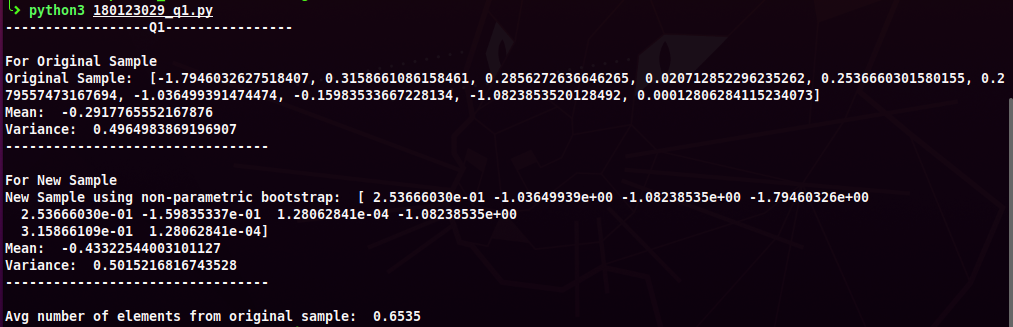
**print("New Sample using non-parametric bootstrap: ", newSample)**

**print("Mean: ", np.mean(newSample))**

**print("Variance: ", np.var(newSample))**

**print("---------------------------------\n")**

**print("Avg number of elements from original sample: ", np.mean(uniqueNo))**

**Output:**

Ques.**2**

**Code:**

**import numpy as np**

**import matplotlib.pyplot as plt**

**print("------------------Q2----------------\n")**

**variance = 1**

**sigma = np.sqrt(variance)**

**epsilon = np.random.normal(0, sigma, 50)**

**n = 50**

**x\_axis = np.linspace(1, n, n)**

**initSample = []**

**initSample.append(0)**

**for i in range(1, n):**

**elem = initSample[i-1]\*0.5 + epsilon[i]**

**initSample.append(elem)**

**plt.plot(x\_axis, initSample)**

**l = 5 # Size of Block**

**# NON-OVERLAPPING**

**numBlocks = *int*(n/l)**

**blocks = np.reshape(initSample, (numBlocks, l))**

**indexes = [x for x in range(numBlocks)]**

**resampleBlock = np.random.choice(indexes, numBlocks)**

**nonOverlap = []**

**for i in resampleBlock:**

**nonOverlap.extend(blocks[i])**

**plt.plot(x\_axis, nonOverlap)**

**# MOVING BLOCK**

**numBlocks = n - l + 1**

**resampleBlock = np.random.choice(indexes, *int*(n/l))**

**movBlock = []**

**for i in resampleBlock:**

**movBlock.extend(initSample[i:i+l])**

**plt.plot(x\_axis, movBlock)**

**# LOCAL BLOCK**

**numBlocks = n - l + 1**

**numBlocksReqd = *int*(n/l)**

**resampleBlock = []**

**delt = 4**

**for i in range(numBlocksReqd):**

**# pick first block randomly**

**if i == 0:**

**resampleBlock.extend(np.random.choice(indexes, 1))**

**else:**

**# new block is selected from vicinity of previous block(+/- delt)**

**lower\_bound = max(0, resampleBlock[i-1] - delt)**

**upper\_bound = min(numBlocks-1, resampleBlock[i-1] + delt)**

**resampleBlock.extend(np.random.choice(indexes[lower\_bound:upper\_bound+1], 1))**

**localBlock = []**

**for i in resampleBlock:**

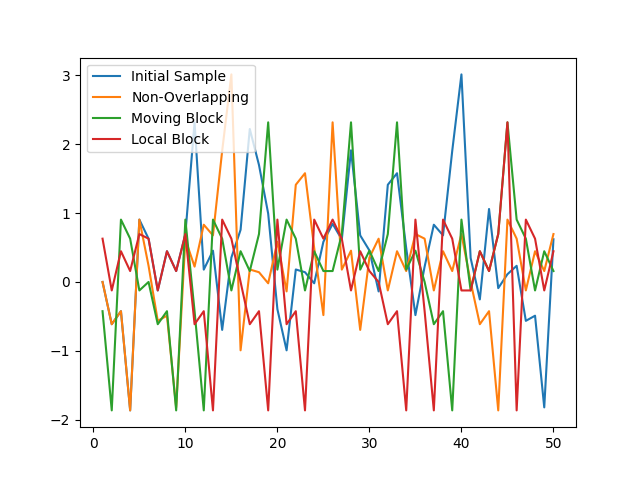
**localBlock.extend(initSample[i:i+l])**

**plt.plot(x\_axis, localBlock)**

**plt.legend(["Initial Sample", "Non-Overlapping", "Moving Block", "Local Block"])**

**plt.show()**

**Output:**

****

Ques.**3**

**Code:**

**import numpy as np**

**import math**

**import matplotlib.pyplot as plt**

**import pandas as pd**

**from numpy import loadtxt**

**from sklearn.linear\_model import LinearRegression**

**from sklearn.preprocessing import PolynomialFeatures**

**from sklearn.pipeline import Pipeline**

**from sklearn.model\_selection import cross\_val\_score**

**from sklearn.metrics import mean\_squared\_error as mse**

**print("------------------Q3----------------\n")**

**x = [(i+1) for i in range(36)]**

**y = loadtxt("FRWRD.txt", *delimiter*="\n", *unpack*=False)**

**x = np.array(x)**

**y = np.array(y)**

**N = 36**

***def* polyRegressionModel(*degree*, *k\_fold*):**

**poly\_features = PolynomialFeatures(*degree*=degree)**

**X\_poly = poly\_features.fit\_transform(x.reshape(-1, 1))**

**poly = LinearRegression()**

**return np.mean(cross\_val\_score(poly, X\_poly, y.reshape(-1, 1), *cv*=k\_fold, *scoring*='neg\_mean\_squared\_error'))**

**err\_Poly3 = polyRegressionModel(3, 10)**

**print("Error in Polynomial 3")**

**print(err\_Poly3)**

**err\_Poly6 = polyRegressionModel(6, 10)**

**print("Error in Polynomial 6")**

**print(err\_Poly6)**

**err\_Poly8 = polyRegressionModel(8, 10)**

**print("Error in Polynomial 8")**

**print(err\_Poly8)**

**lm = LinearRegression()**

**lm.fit(x.reshape(-1, 1), y.reshape(-1, 1))**

**plt.scatter(x, y, *s*=15, *label*='Original Pts')**

**print("Calc for degree 3")**

**Inp3 = [('polynomial', PolynomialFeatures(*degree*=3)),('modal', LinearRegression())]**

**pipe3 = Pipeline(Inp3)**

**pipe3.fit(x.reshape(-1, 1), y.reshape(-1, 1))**

**predPoly3 = pipe3.predict(x.reshape(-1, 1))**

**sortZip3 = sorted(zip(x, predPoly3))**

**xPolyfor3, predPoly3 = zip(\*sortZip3)**

**plt.plot(xPolyfor3, predPoly3, *label*='Polynomial Reg for degree -- 3')**

**print("Calc for degree 6")**

**Inp6 = [('polynomial', PolynomialFeatures(*degree*=6)),('modal', LinearRegression())]**

**pipe6 = Pipeline(Inp6)**

**pipe6.fit(x.reshape(-1, 1), y.reshape(-1, 1))**

**predPoly6 = pipe6.predict(x.reshape(-1, 1))**

**sortZip6 = sorted(zip(x, predPoly6))**

**xPolyfor6, predPoly6 = zip(\*sortZip6)**

**plt.plot(xPolyfor6, predPoly6, *label*='Polynomial Reg for degree -- 6')**

**print("Calc for degree 8")**

**Inp8 = [('polynomial', PolynomialFeatures(*degree*=8)),('modal', LinearRegression())]**

**pipe8 = Pipeline(Inp8)**

**pipe8.fit(x.reshape(-1, 1), y.reshape(-1, 1))**

**predPoly8 = pipe8.predict(x.reshape(-1, 1))**

**sortZip8 = sorted(zip(x, predPoly8))**

**xPolyfor8, predPoly8 = zip(\*sortZip8)**

**plt.plot(xPolyfor8, predPoly8, *label*='Polynomial Reg for degree -- 8')**

**plt.legend()**

**plt.show()**

* Text is been loaded from FRWRD.txt file

2.351

2.387

2.4020000000000001

2.4119999999999999

2.4239999999999999

2.4239999999999999

2.4369999999999998

2.5670000000000002

2.6840000000000002

2.6989999999999998

2.5640000000000001

2.4409999999999998

2.3090000000000002

2.2759999999999998

2.2679999999999998

2.2709999999999999

2.2730000000000001

2.2730000000000001

2.2949999999999999

2.4180000000000001

2.5489999999999999

2.569

2.4740000000000002

2.3740000000000001

2.2570000000000001

2.2450000000000001

2.25

2.2549999999999999

2.2589999999999999

2.2679999999999998

2.2959999999999998

2.4239999999999999

2.5710000000000002

2.5950000000000002

2.48

2.3799999999999999

**Output:**

**Terminal**



**Plot**

