ECE 50024: Homework 1

Manish Kumar Krishne Gowda, 0033682812

(Spring 2023)

Exercise 1: Histogram and Cross-Validation

(a)

```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

x = np.linspace(-3,3,1000)
fx = stats.norm.pdf(x, 0, 1)
plt.plot(x,fx,markersize=12)
plt.savefig("1_a")
plt.show()
```

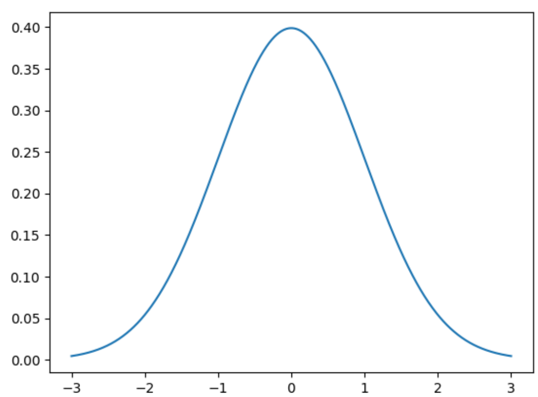


Figure 1: Exercise 1(a)

(b)

```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

x = np.linspace(-3,3,1000)
x_samp = np.random.normal(0,1,1000)  #Use numpy.random.normal to draw 1000 random
                                      samples from N (0, 1)
plt.hist(x_samp,bins=4,density=True); #Histogram plots using matplotlib.pyplot.
                                      hist, with the number of bins m set to 4
mu, sigma = stats.norm.fit(x_samp) #Use scipy.stats.norm.fit to estimate the mean
                                      and standard deviation of your data
mean_n_sigma_string = str(mu) + " & " + str(sigma)
print("mean and sd values of fit data are : " + mean_n_sigma_string + "
                                      respectively" ) #Reporting the estimated
                                      values
```

```
fx = stats.norm.pdf(x, mu, sigma)  #Plot the fitted gaussian curve on top of the
                                           histogram plots using scipy.stats.norm.
                                           pdf.
plt.plot(x,fx,markersize=12)
plt.savefig("1_b_i")
plt.show()

plt.hist(x_samp,bins=1000,density=True); #repeating the same for m = 1000
plt.plot(x,fx,markersize=12)
plt.savefig("1_b_ii")
plt.show()
```

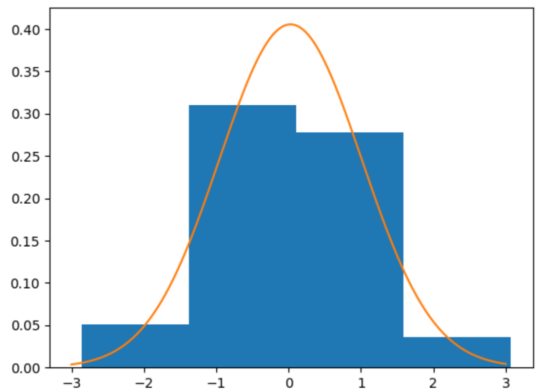


Figure 2: Exercise 1(b), m=4

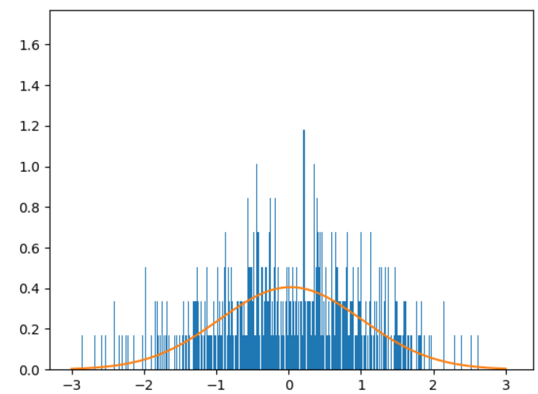


Figure 3: Exercise 1(b), m=1000

Observation :

1. Mean and Standard Deviation values of fit data are : 0.057575305878533226 & 1.0267647881000108 respectively
2. Higher the bin width, higher will be the frequency of the occurrence of that element, as more data values from the data set will now fall into this wider width rectangle

3. Note that the above figure is normalised to enable the pdf and histogram being compared

(c)

```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

n = 1000
x = np.linspace(-3,3,n)
x_samp = np.random.normal(0,1,n)
mu, sigma = stats.norm.fit(x_samp)
fx = stats.norm.pdf(x, mu, sigma)
m = np.arange(1,201) #m the number of bins, for m = 1, 2, ..., 200
J = np.zeros((200))
max_value = np.max(x_samp)
min_value = np.min(x_samp)
for i in range(0,200):
  hist,bins = np.histogram(x_samp,bins=m[i])
  h = (max_value-min_value)/m[i]
  J[i] = 2/((n-1)*h)-((n+1)/((n-1)*h))*np.sum((hist/n)**2)
plt.plot(m,J);
plt.savefig("1_c_i")
plt.show()

min_J_value = np.min(J)
m_star = 1+np.where(J==min_J_value)[0][0]
print("m* that minimizes the risk value : " + str(m_star) )
plt.hist(x_samp,bins=m_star,density=True); #plot the histogram of your data with
                                           that m
plt.plot(x,fx,markersize=12) #Plot the Gaussian curve fitted to your data on top
                                           of your histogram
plt.savefig("1_c_iii")
plt.show()
```

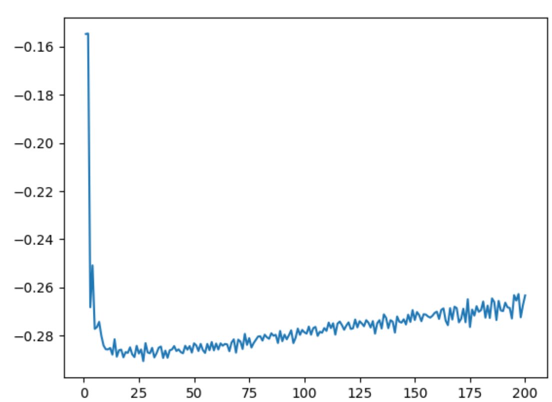


Figure 4: Exercise 1(c), m vs J

Observed m that minimizes the risk value : 27

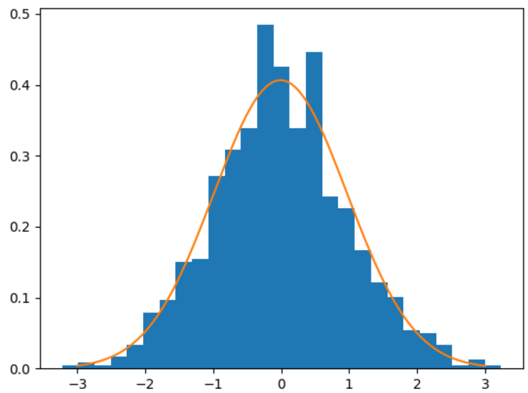
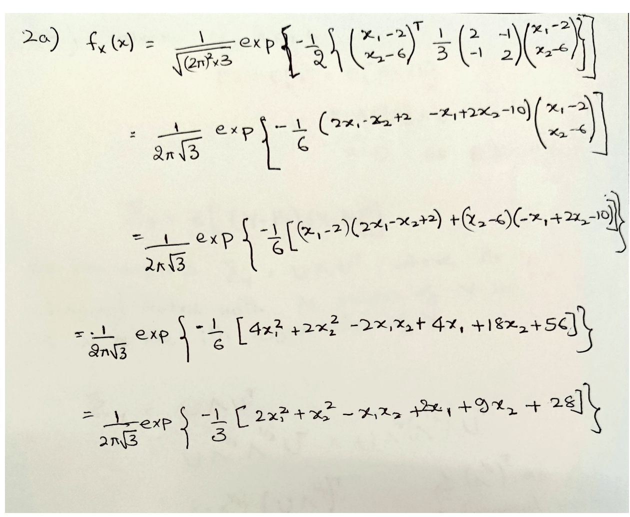


Figure 5: Exercise 1(c), Gaussian curve fitted to histogram

Figure. To insert a figure, you may use the command `includegraphics`. When inserting a figure, make sure that the resolution is high enough and the font of is readable. The general rule of thumb is that if you can read the figure at 100% view (i.e., no zoom in), the figure is typically okay. When generating plots, please save them as `.pdf` or `.eps` because these are vectorized graphics files. We recommend you bold the curves so that they are more visible. Mark your axes and legends clearly. We reserve the right to not giving points to plots that are not readable.

Exercise 2: Gaussian Whitening

(a)

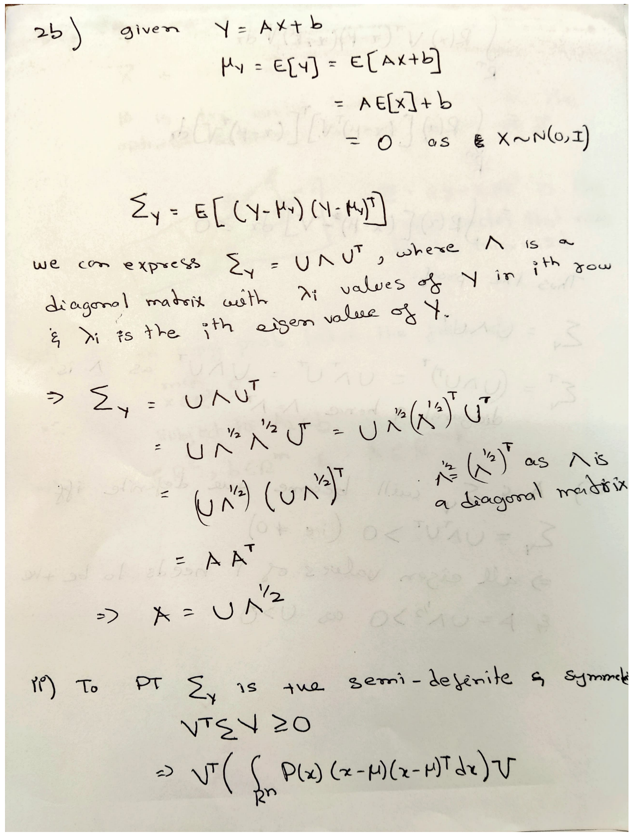


2a)
$$f_x(x) = \frac{1}{\sqrt{(2\pi)^2 \times 3}} \exp\left[-\frac{1}{2}\left\{\begin{pmatrix} x_1-2 \\ x_2-6 \end{pmatrix}^T \frac{1}{3}\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}\begin{pmatrix} x_1-2 \\ x_2-6 \end{pmatrix}\right\}\right]$$
$$= \frac{1}{2\pi\sqrt{3}} \exp\left[-\frac{1}{6}\begin{pmatrix} 2x_1-x_2+2 & -x_1+2x_2-10 \end{pmatrix}\begin{pmatrix} x_1-2 \\ x_2-6 \end{pmatrix}\right]$$
$$= \frac{1}{2\pi\sqrt{3}} \exp\left\{-\frac{1}{6}\left[(x_1-2)(2x_1-x_2+2) + (x_2-6)(-x_1+2x_2-10)\right]\right\}$$
$$= \frac{1}{2\pi\sqrt{3}} \exp\left\{-\frac{1}{6}\left[4x_1^2 + 2x_2^2 - 2x_1x_2 + 4x_1 + 18x_2 + 56\right]\right\}$$
$$= \frac{1}{2\pi\sqrt{3}} \exp\left\{-\frac{1}{3}\left[2x_1^2 + x_2^2 - x_1x_2 + 2x_1 + 9x_2 + 28\right]\right\}$$

Figure 6: Exercise 2(a), Gaussian Expression Simplification

(ii)

```
import numpy as np
import numpy.matlib as npm
```


2b) given $Y = AX + b$

$\mu_Y = E[Y] = E[AX+b]$

$= AE[X] + b$

$= 0$ as $X \sim N(0, I)$

$\Sigma_Y = E[(Y-\mu_Y)(Y-\mu_Y)^T]$

we can express $\Sigma_Y = U \Lambda U^T$, where Λ is a diagonal matrix with λ_i values of Y in i^{th} row & λ_i is the i^{th} eigen value of Y.

$\Rightarrow \Sigma_Y = U \Lambda U^T$

$= U \Lambda^{1/2} \Lambda^{1/2} U^T = U \Lambda^{1/2} (\Lambda^{1/2})^T U^T$

$= (U \Lambda^{1/2})(U \Lambda^{1/2})^T$ $\quad \Lambda^{1/2} = (\Lambda^{1/2})^T$ as Λ is a diagonal matrix

$= A A^T$

$\Rightarrow A = U \Lambda^{1/2}$

ii) To PT Σ_Y is +ve semi-definite & symme

$V^T \Sigma V \geq 0$

$\Rightarrow V^T \left(\int_{\mathbb{R}^n} P(x)(x-\mu)(x-\mu)^T dx \right) V$

Figure 7: Exercise 2(b), Eigen Decomposition

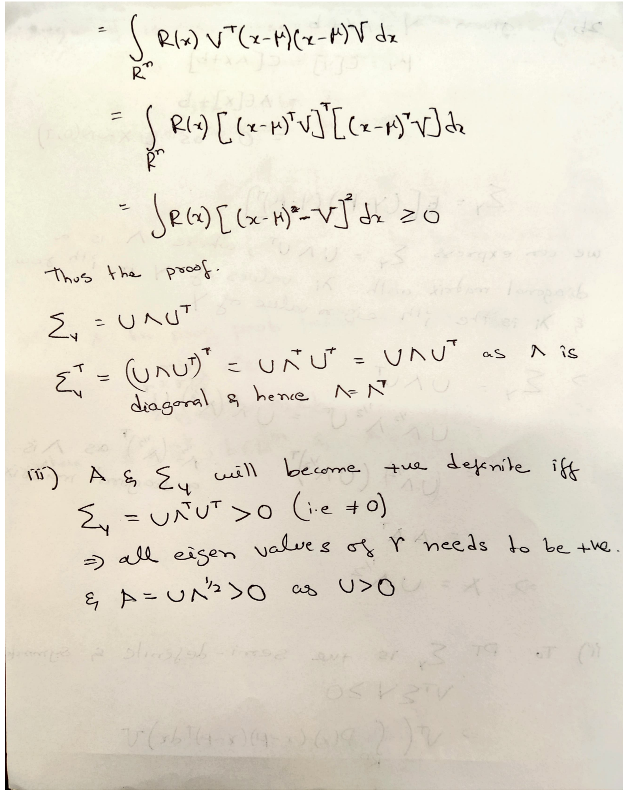
```
import scipy.stats as stats
from scipy.linalg import fractional_matrix_power
import matplotlib.pyplot as plt


X = stats.multivariate_normal.rvs([2,6],[[2,1],[1,2]],1000)
x1 = np.arange(-1, 5, 0.01)
x2 = np.arange(0, 10, 0.01)
X1, X2 = np.meshgrid(x1,x2)
Xpos = np.empty(X1.shape + (2,))
Xpos[:,:,0] = X1
Xpos[:,:,1] = X2
F = stats.multivariate_normal.pdf(Xpos,[2,6],[[2,1],[1,2]])
plt.scatter(X[:,0],X[:,1])
plt.contour(x1,x2,F)
plt.savefig("2_a_ii")
plt.show()
```

(c)

(i)

```
import numpy as np
import numpy.matlib as npm
import scipy.stats as stats
from scipy.linalg import fractional_matrix_power
import matplotlib.pyplot as plt
```



$$= \int_{R^n} R(x)\, V^T (x-\mu)(x-\mu)^T V \, dx$$

$$= \int_{R^n} R(x) \left[(x-\mu)^T V \right]^T \left[(x-\mu)^T V \right] dx$$

$$= \int R(x) \left[(x-\mu)^T V \right]^2 dx \geq 0$$

Thus the proof.

$\Sigma_Y = U \Lambda U^T$

$\Sigma_Y^T = (U \Lambda U^T)^T = U \Lambda^T U^T = U \Lambda U^T$ as Λ is diagonal & hence $\Lambda = \Lambda^T$

iii) A & Σ_Y will become +ve definite iff

$\Sigma_Y = U \Lambda^T U^T > 0$ (i.e $\neq 0$)

$\Rightarrow$ all eigen values of Y needs to be +ve.

& $A = U \Lambda^{1/2} > 0$ as $U > 0$

Figure 8: Exercise 2(b), Eigen Decomposition2

```
X = np.random.multivariate_normal([0,0],[[1,0],[0,1]],5000)
x1 = np.arange(-2.5, 2.5, 0.01)
x2 = np.arange(-2.5, 2.5, 0.01)
X1, X2 = np.meshgrid(x1,x2)
Xpos = np.empty(X1.shape + (2,))
Xpos[:,:,0] = X1
Xpos[:,:,1] = X2
F = stats.multivariate_normal.pdf(Xpos,[0,0],[[1,0],[0,1]])
plt.scatter(X[:,0],X[:,1])
plt.contour(x1,x2,F)
plt.savefig("2_c_i")
plt.show()
```

(ii)

```
import numpy as np
import numpy.matlib as npm
import scipy.stats as stats
from scipy.linalg import fractional_matrix_power
import matplotlib.pyplot as plt


X = np.random.multivariate_normal([0,0],[[1,0],[0,1]],5000)
mu = np.array([2,6])
Sigma = np.array([[2,1],[1,2]])
S, U = np.linalg.eig(Sigma)  #Sigma = USU.T
```

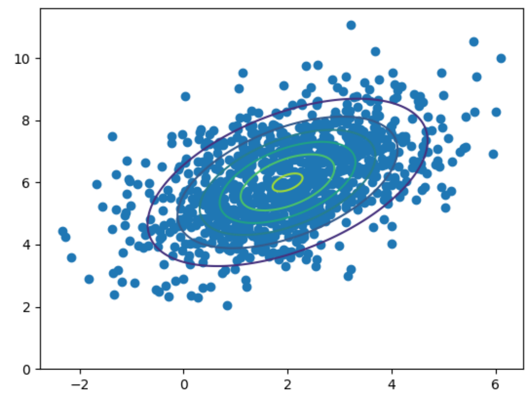


Figure 9: Contour of fX(x).

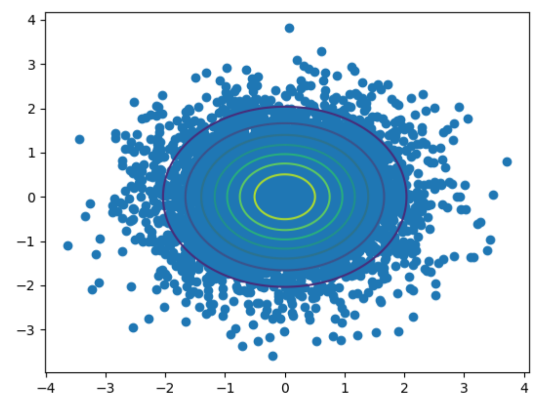


Figure 10: Scatter plot of normal distribution

```
s = np.diag(S)
A = np.matmul(U,fractional_matrix_power(s,0.5))
print(np.matmul(A,A.T))  #This should be equal to sigma

Sigma_half = fractional_matrix_power(Sigma,0.5)
Y = np.dot(Sigma_half , X.T) + npm.repmat(mu,5000,1).T
x1 = np.arange(-1, 5, 0.01)
x2 = np.arange(0, 10, 0.01)
X1, X2 = np.meshgrid(x1,x2)
Xpos = np.empty(X1.shape + (2,))
Xpos[:,:,0] = X1
Xpos[:,:,1] = X2
F = stats.multivariate_normal.pdf(Xpos,[2,6],[[2,1],[1,2]])
plt.scatter(Y.T[:,0],Y.T[:,1])
plt.contour(x1,x2,F)
plt.savefig("2_c_iii")
plt.show()
```

(iii) Clearly, theoretical findings from part(b) is matching with that of programming solution

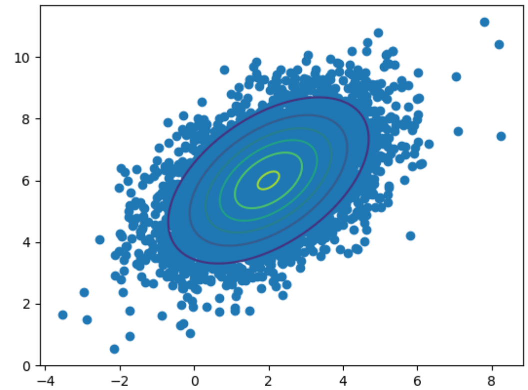
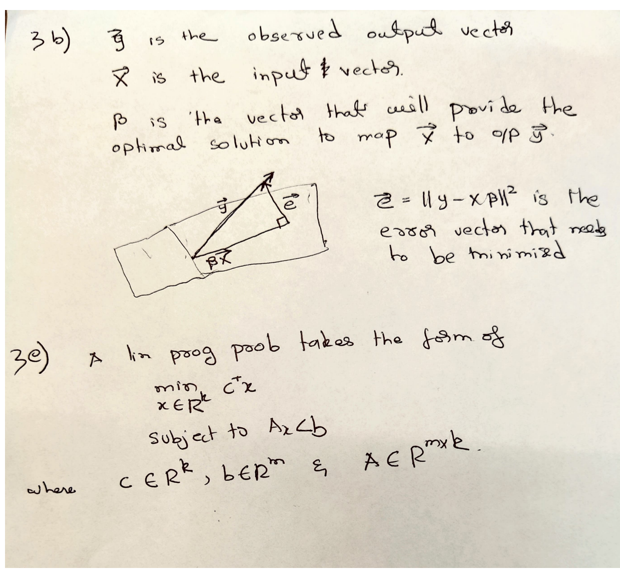


Figure 11: Scatter plot of transformed distribution

(eigen values in both cases are (3,1))

Exercise 3: Linear Regression



3 b) $\vec{y}$ is the observed output vector

$\vec{X}$ is the input ~~t~~ vector.

β is the vector that will provide the optimal solution to map $\vec{X}$ to o/p $\vec{y}$.

$\vec{y}$ $\vec{e}$ βX

$\vec{e} = \|y - X\beta\|^2$ is the error vector that needs to be minimized

3e) A lin prog prob takes the form of

$$\min_{x \in R^k} c^T x$$

subject to $Ax < b$

where $c \in R^k$, $b \in R^m$ & $A \in R^{m \times k}$.

Figure 12: 3(b) and (e)

(a), (c), (d) and (f)

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.special import eval_legendre
from scipy.optimize import linprog

x = np.linspace(-1,1,50)
beta = np.array([-0.001, 0.01, 0.55, 1.5, 1.2])
```

```
y = beta[0]*eval_legendre(0,x) + beta[1]*eval_legendre(1,x) + beta[2]*eval_legendre(2,x) + \
  beta[3]*eval_legendre(3,x) + beta[4]*eval_legendre(4,x) + np.random.normal(0, 0.2, 50)
X = np.column_stack((eval_legendre(0,x), eval_legendre(1,x), eval_legendre(2,x), \
  eval_legendre(3,x), eval_legendre(4,x)))
beta_hat = np.linalg.lstsq(X, y, rcond=None)[0]
print(beta_hat)
yhat = beta_hat[0]*eval_legendre(0,x) + beta_hat[1]*eval_legendre(1,x) + beta_hat[2]*eval_legendre(2,x) + \
    beta_hat[3]*eval_legendre(3,x) + beta_hat[4]*eval_legendre(4,x)

plt.plot(x,y,'o',markersize=12)
plt.plot(x,yhat, linewidth=8)
plt.savefig("3_a_n_c")
plt.show()
#===========================================================================
idx = [10, 16, 23, 37, 45];
y[idx] = 5;
beta_hat_with_outlier = np.linalg.lstsq(X, y, rcond=None)[0]
yhat_with_outlier = beta_hat_with_outlier[0]*eval_legendre(0,x) + beta_hat_with_outlier[1]*eval_legendre(1,x) + beta_hat_with_outlier[2]*eval_legendre(2,x) \
          + beta_hat_with_outlier[3]*eval_legendre(3,x) + beta_hat_with_outlier[4]*eval_legendre(4,x)

plt.plot(x,y,'o',markersize=12)
plt.plot(x,yhat_with_outlier, 'r', linewidth=8)
plt.savefig("3_d")
plt.show()
#===========================================================================
X_lp = np.column_stack((eval_legendre(0,x), eval_legendre(1,x), eval_legendre(2,x), \
  eval_legendre(3,x), eval_legendre(4,x)))
A = np.vstack((np.hstack((X_lp, -np.eye(50))), np.hstack((-X_lp, -np.eye(50)))))
b = np.hstack((y,-y))
c = np.hstack((np.zeros(5), np.ones(50)))
res = linprog(c, A, b, bounds=(None,None), method="highs")
beta_cap_lp = res.x
yhat_lp = beta_cap_lp[0]*eval_legendre(0,x) + beta_cap_lp[1]*eval_legendre(1,x) + beta_cap_lp[2]*eval_legendre(2,x) + \
    beta_cap_lp[3]*eval_legendre(3,x) + beta_cap_lp[4]*eval_legendre(4,x)
plt.plot(x,y,'o',markersize=12)
plt.plot(x,yhat_lp, 'y', linewidth=8)
plt.savefig("3_f")
plt.show()
```

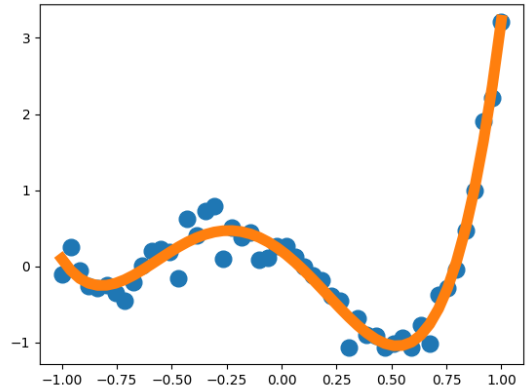


Figure 13: Predicted curve with scattered plot

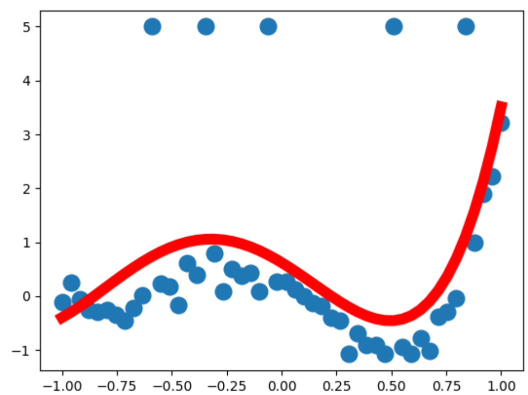


Figure 14: Predicted curve with outliers

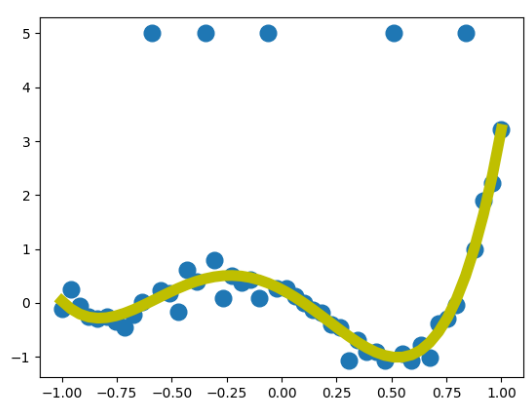


Figure 15: Optimized with Linear programming problem

Learning to Reweight Examples for Robust Deep Learning

Manish Kumar Krishne Gowda, Purdue University, ECE Department, On Campus MS Student[1]