Homework 1

Advanced Data Analysis

In [ ]:
```
from __future__ import division
from __future__ import print_function

import numpy as np
import matplotlib

# matplotlib.use('TkAgg')
import matplotlib.pyplot as plt

np.random.seed(0)  # set the random seed for reproducibility

def generate_sample(xmin, xmax, sample_size):
    x = np.linspace(start=xmin, stop=xmax, num=sample_size)
    pix = np.pi * x
    target = np.sin(pix) / pix + 0.1 * x
    noise = 0.05 * np.random.normal(loc=0., scale=1., size=sample_size)
    return x, target + noise


def calc_design_matrix(x, c, h):
    return np.exp(-(x[None] - c[:, None]) ** 2 / (2 * h ** 2))
```

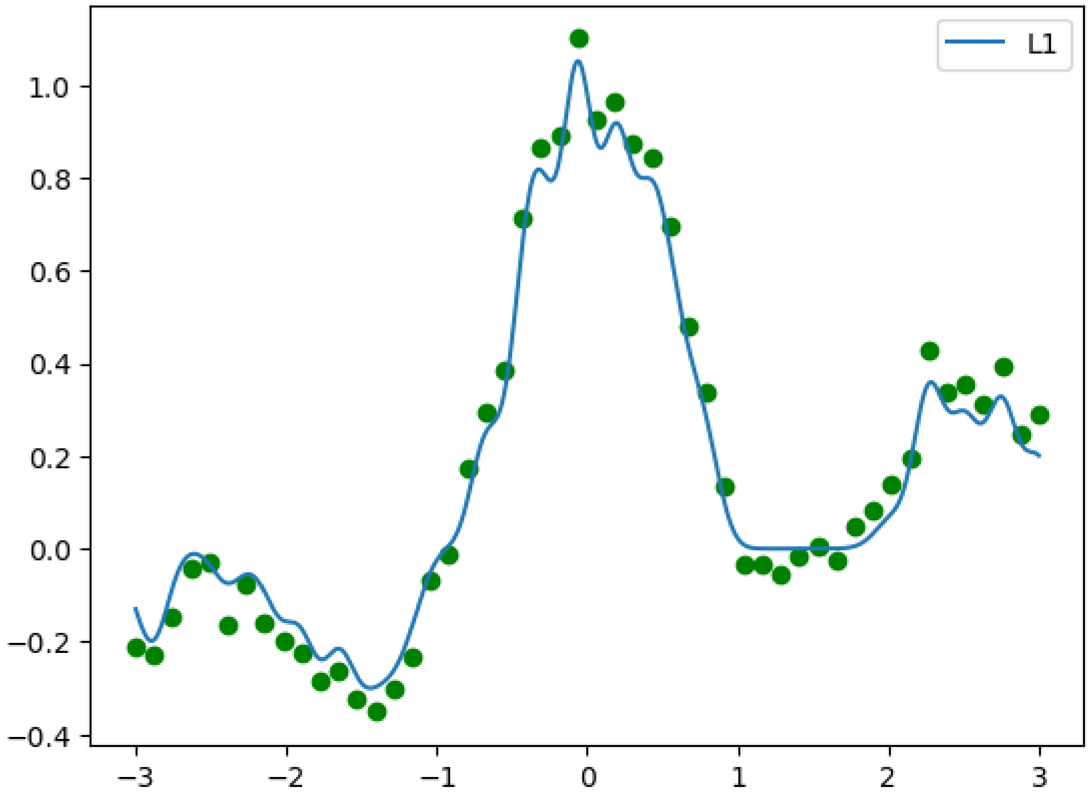
Define L1 CLS

In [ ]:
```
def iteratively_reweighted_shrinkage(sample_size, k, y, l, n_iter=1000):
    # initialize theta using the solution of regularized ridge regression
    theta = theta_prev = np.linalg.solve(k.T.dot(k) + 1e-4 * np.identity(sample_si
    eta = np.Inf # for L1 regularized L2 loss minimization
    for _ in range(n_iter):
        r = np.abs(k.dot(theta_prev) - y)
        W = np.diag(np.where(r > eta, eta / r, 1.))
        # contruct Phi matrix using computed theta
        Phi = np.diag(np.abs(theta))
        # take generalized inverse of phi
        Phi_gi = np.linalg.pinv(Phi)
        # compute new theta
        theta = np.linalg.solve(k.T.dot(W).dot(k) + l * Phi_gi + 0.000001*np.ident
        # check for convergence
        if np.linalg.norm(theta - theta_prev) < 1e-3: break
        theta_prev = theta
    return theta
```

Visualize L1 Regression

In [ ]:
```
# create sample
sample_size = 50
xmin, xmax = -3, 3
x, y = generate_sample(xmin=xmin, xmax=xmax, sample_size=sample_size)

# calculate design matrix
```

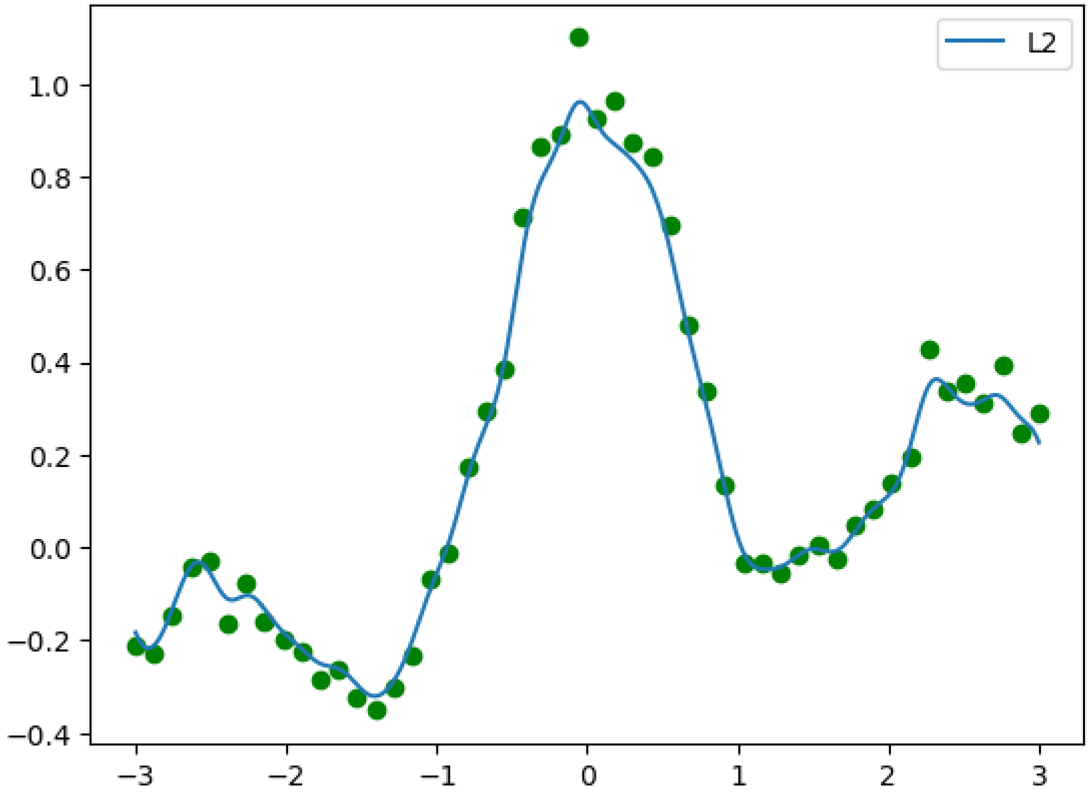
```python
h = 0.1
k = calc_design_matrix(x, x, h)

# solve the L1 least square problem
l = 0.1
theta1 = iteratively_reweighted_shrinkage(sample_size,k,y,l)

# solve the L2 least square problem
l = 0.3
theta2 = np.linalg.solve(
    k.T.dot(k) + l * np.identity(len(k)),
    k.T.dot(y[:, None]))

# create data to visualize the L2 prediction
X = np.linspace(start=xmin, stop=xmax, num=5000)
K = calc_design_matrix(x, X, h)
prediction1 = K.dot(theta1)
prediction2 = K.dot(theta2)

# visualization
plt.clf()
plt.scatter(x, y, c='green', marker='o')
plt.plot(X, prediction1,label="L1")
plt.legend()
plt.show()
```



Visualize L2 Regression

In [ ]:

```python
plt.clf()
plt.scatter(x, y, c='green', marker='o')
plt.plot(X, prediction2, label="L2")
plt.legend()
plt.show()
```


L2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−3
−2
−1
0
1
2
3

Advanced Data Analysis
Homework Week 3

Aswin Vijay

May 4, 2023

Homework 3

Given the linear model $f_\theta(x) = \sum_{j=1}^{b} \theta_j \phi_j(x)$, where $\{\phi_j(x)\}_{j=1}^{b}$ are the basis functions, the weighted least squares objective is given by,

$$\hat{\theta} = \min_\theta \frac{1}{2} \sum_{i=1}^{n} \tilde{w}_i \Bigg(f_\theta(x_i) - y_i \Bigg)^2$$

To derive the analytical solution we rewrite the above objective using the weighting matrix $\tilde{W} = diag(\tilde{w}_1, \dots, \tilde{w}_n)$ in matrix format as shown below.

$$\hat{\theta} = \min_\theta \Bigg[\frac{1}{2} (\Phi\theta - \mathbf{y})^T \tilde{W} (\Phi\theta - \mathbf{y}) \Bigg]$$

where Φ is the design matrix and $\mathbf{y} = (y_1, \dots, y_n)^T$. Expanding the brackets and then taking the derivative w.r.t θ we get,

$$\begin{aligned}
\hat{\theta} &= \min_\theta \Bigg[\frac{1}{2} (\Phi\theta - \mathbf{y})^T \tilde{W} (\Phi\theta - \mathbf{y}) \Bigg] \\
&= \min_\theta \Bigg[\frac{1}{2} (\theta^T \Phi^T - \mathbf{y}^T) \tilde{W} (\Phi\theta - \mathbf{y}) \Bigg] \\
&= \min_\theta \Bigg[\frac{1}{2} (\theta^T \Phi^T - \mathbf{y}^T) \tilde{W} (\Phi\theta - \mathbf{y}) \Bigg] \\
&= \min_\theta \frac{1}{2} \Bigg[\theta^T \Phi^T \tilde{W} \Phi\theta - \theta^T \Phi^T \tilde{W} \mathbf{y} - \mathbf{y}^T \tilde{W} \Phi\theta + \mathbf{y}^T \tilde{W} \mathbf{y} \Bigg] \quad \textit{Taking derivative w.r.t } \theta \\
&= \Bigg[\Phi^T \tilde{W} \Phi\theta - \Phi^T \tilde{W} \mathbf{y} \Bigg] = 0
\end{aligned}$$

Thus we get the required result,

$$\hat{\theta} = (\Phi^T \tilde{W} \Phi)^{-1} \Phi^T \tilde{W} \mathbf{y}$$

Homework 3

Advanced Data Analysis

In [ ]:
```
import numpy as np
import matplotlib
#matplotlib.use('TkAgg')
import matplotlib.pyplot as plt

np.random.seed(1)

def generate_sample(x_min=-3., x_max=3., sample_size=10):
    x = np.linspace(x_min, x_max, num=sample_size)
    y = x + np.random.normal(loc=0., scale=.2, size=sample_size)
    y[-1] = y[-2] = y[1] = -4 # outliers
    return x, y

def build_design_matrix(x):
    phi = np.empty(x.shape + (2,))
    phi[:, 0] = 1.
    phi[:, 1] = x
    return phi
```

Define Huber and Tukey reweighted iterative regularized regression

In [ ]:
```
def iterative_reweighted_least_squares_huber(x, y, eta=1., n_iter=1000):
    phi = build_design_matrix(x)
    # initialize theta using the solution of regularized ridge regression
    theta = theta_prev = np.linalg.solve(
    phi.T.dot(phi) + 1e-4 * np.identity(phi.shape[1]), phi.T.dot(y))
    for _ in range(n_iter):
        r = np.abs(phi.dot(theta_prev) - y)
        w = np.diag(np.where(r > eta, eta / r, 1.))
        phit_w_phi = phi.T.dot(w).dot(phi)
        phit_w_y = phi.T.dot(w).dot(y)
        theta = np.linalg.solve(phit_w_phi, phit_w_y)
        if np.linalg.norm(theta - theta_prev) < 1e-3: break
        theta_prev = theta
    return theta

def iterative_reweighted_least_squares_tukey(x, y, eta=1., n_iter=1000):
    phi = build_design_matrix(x)
    # initialize theta using the solution of regularized ridge regression
    theta = theta_prev = np.linalg.solve(
    phi.T.dot(phi) + 1e-4 * np.identity(phi.shape[1]), phi.T.dot(y))
    for _ in range(n_iter):
        r = np.abs(phi.dot(theta_prev) - y)
        w = np.diag(np.where(r > eta, 0, (1-(r/eta)**2)**2))
        phit_w_phi = phi.T.dot(w).dot(phi)
        phit_w_y = phi.T.dot(w).dot(y)
        theta = np.linalg.solve(phit_w_phi, phit_w_y)
        if np.linalg.norm(theta - theta_prev) < 1e-3: break
        theta_prev = theta
    return theta
```

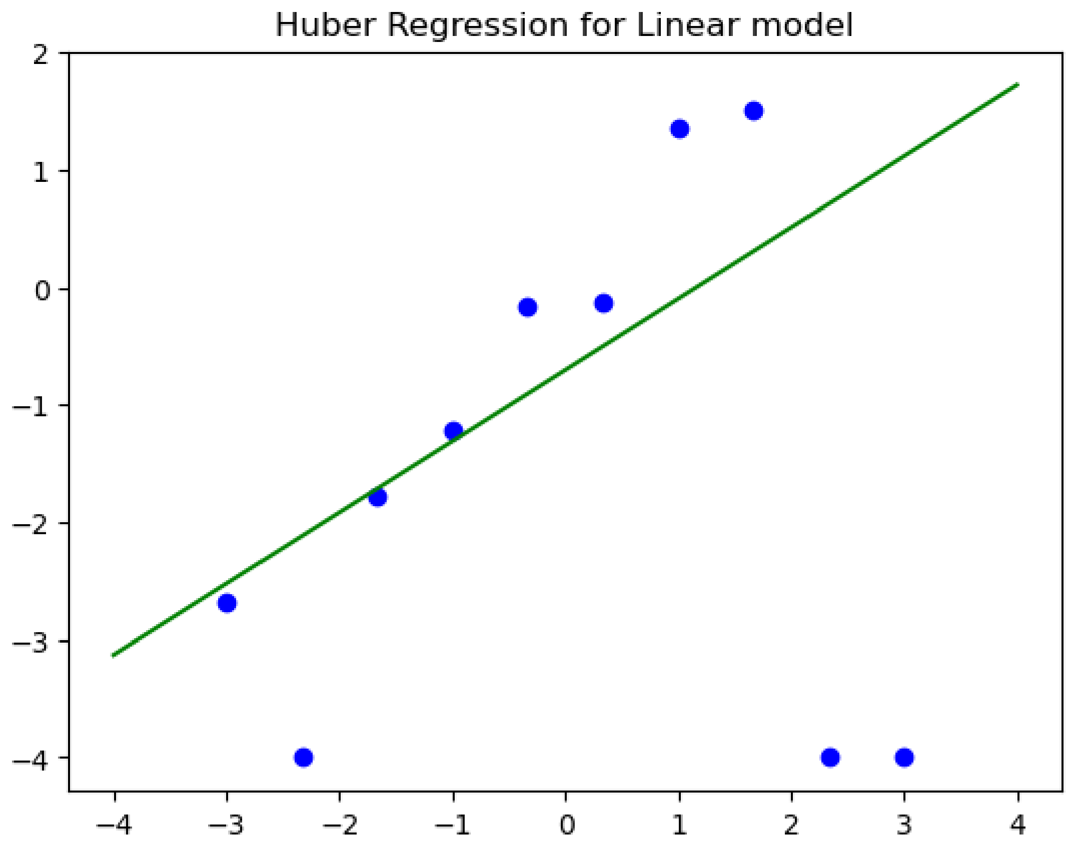
```python
def predict(x, theta):
    phi = build_design_matrix(x)
    return phi.dot(theta)
```

Visualize

In [ ]:
```python
def visualize(x, y, theta, x_min=-4., x_max=4., title='plot'):
    X = np.linspace(x_min, x_max, 1000)
    Y = predict(X, theta)
    plt.clf()
    plt.plot(X, Y, color='green')
    plt.scatter(x, y, c='blue', marker='o')
    plt.title(title)
    plt.show()

x, y = generate_sample()
theta1 = iterative_reweighted_least_squares_huber(x, y, eta=1.)
theta2 = iterative_reweighted_least_squares_tukey(x, y, eta=1.)
```

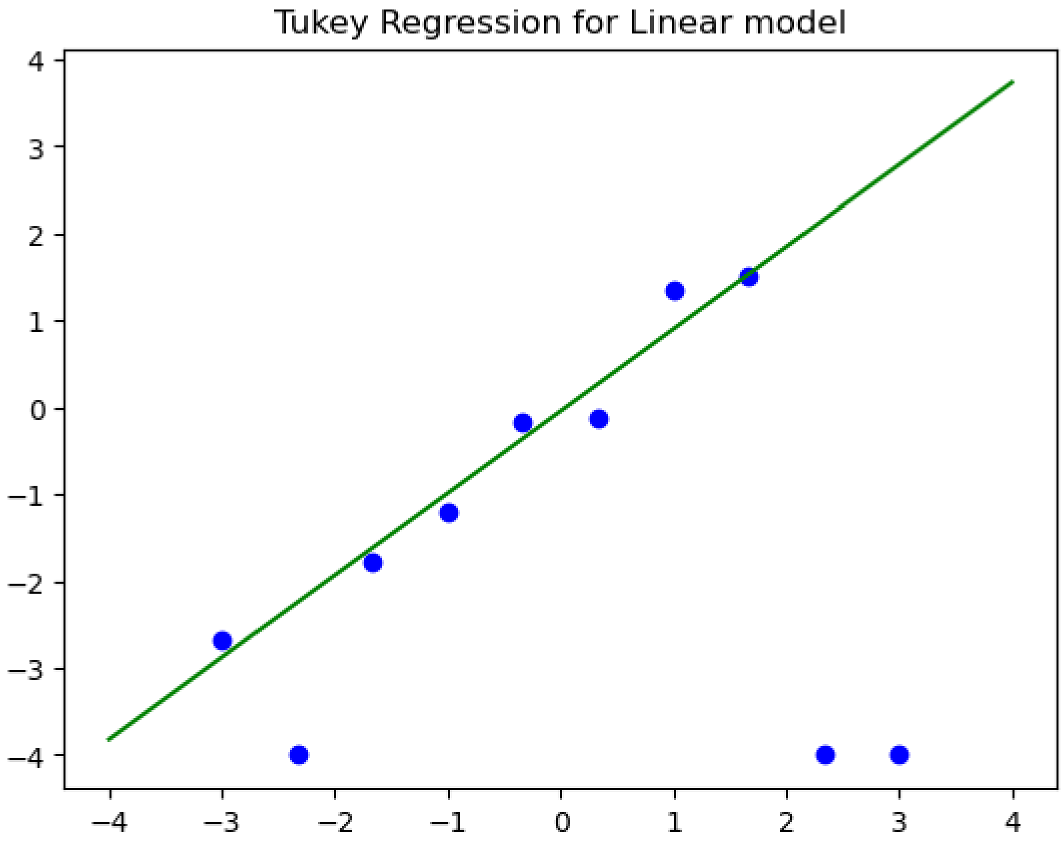
Visualize Huber

In [ ]:
```python
visualize(x, y, theta1, title = "Huber Regression for Linear model")
```



Visualize Tukey

In [ ]:
```python
visualize(x, y, theta2, title = "Tukey Regression for Linear model")
```


Tukey Regression for Linear model
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3
4