1a)

$$RSS(\theta) = (Y - X\theta)^T (Y - X\theta)$$

$$= (Y^T - \theta^T X^T)(Y - X\theta$$

$$= Y^T Y - \theta^T X^T Y - Y^T X\theta + \theta^T X^T X \theta$$

$$\frac{\partial}{\partial \theta}(RSS(\theta)) = -X^T Y - X^T Y + \left(X^T X + (X^T X)^T\right)\theta$$

$$= -2X^T Y + 2X^T X \theta$$

$$= 0$$

$$\theta_{\text{Critical}} = (X^T X)^{-1} X^T Y$$

1 b)

$$\text{Ridge}_\lambda(\theta) = RSS(\theta) + \lambda^2 \theta^T \theta$$

$$\frac{\partial}{\partial \theta}(\text{Ridge}_\lambda \theta) = \frac{\partial}{\partial \theta}(RSS(\theta)) + \lambda^2 \frac{\partial}{\partial \theta}(\theta^T \theta)$$

$$= -2X^T Y + 2X^T X \theta + 2\lambda^2 \theta$$

$$= -2X^T Y + 2(X^T X + \lambda^2 I)\theta$$

$$= 0$$

$$\Rightarrow \theta_{critcal} = (X^T X + \lambda^2 I)^{-1} X^T Y$$

3 a)

$$J(\vec{\theta}, \vec{x}) = \sum_{i=1}^{n} w^{(i)} \left(\vec{\theta}^T \vec{x}^{(i)} - y^{(i)} \right)^2$$

$$= (Y - X\theta)^T W (Y - X\theta)$$

where,

$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}_{n\times 1}, \quad X = \begin{bmatrix} 1 & x^{(1)(1)} & x^{(1)(2)} & \cdots & x^{(1)(d)} \\ 1 & x^{(2)(1)} & x^{(2)(2)} & \cdots & x^{(2)(d)} \\ \vdots & & \ddots & & \vdots \\ 1 & x^{(n)(1)} & x^{(n)(2)} & \cdots & x^{(n)(d)} \end{bmatrix}_{n\times(d+1)} \quad \theta = \begin{bmatrix} \theta^{(0)} \\ \theta^{(1)} \\ \theta^{(2)} \\ \vdots \\ \theta^{(d)} \end{bmatrix}_{(d+1)\times 1} \quad W = \begin{bmatrix} w^{(1)} & & & 0 \\ & w^{(2)} & & \\ & & \ddots & \\ 0 & & & w^{(n)} \end{bmatrix}_{n\times n}$$

$$J(\vec{\theta}, \vec{x}) = (Y^T - \theta^T X^T)(WY - WX\theta)$$

$$= Y^T WY - Y^T WX\theta - \theta^T X^T WY + \theta^T X^T WX\theta$$

(a)

$$\frac{\partial (J(\vec{\theta}, \vec{x}))}{\partial \theta} = 2X^T WX\theta - 2X^T WY$$

$$= 2X^T W(X\theta - Y)$$

(b)

$$\frac{\partial^2 (J(\vec{\theta}, \vec{x}))}{\partial \theta^2} = 2(X^T WX)^T$$

$$= 2X^T W^T X$$

$$= 2X^T WX$$

(c)

$$\theta^{(t+1)} = \theta^{(t)} - 2\eta X^T W \left(X\theta^{(t)} - Y \right)$$

$$= \left(I_n - 2\eta X^T W X \right)\theta^{(t)} + 2\eta X^T W Y$$

(d)

$$\theta^{(t+1)} = \theta^{(t)} - \frac{1}{2}\left(X^T W X \right)^{-1} 2 X^T W \left(X\theta^{(t)} - Y \right)$$

$$= \theta^{(t)} - \theta^{(t)} + \left(X^T W X \right)^{-1} X^T W Y$$

$$= \left(X^T W X \right)^{-1} X^T W Y$$

n

(4)

(1) $f(x) = \beta_0 + \beta_1 \sin(x) + \beta_2 \cos(x)$

$$X = \begin{bmatrix} 1 & \sin(x_1) & \cos(x_1) \\ 1 & \sin(x_2) & \cos(x_2) \\ & \vdots & \\ 1 & \sin(x_n) & \cos(x_n) \end{bmatrix}$$

Sin(x) and cos(x) are linearly independent. There fore, we have design matrix with linearly independent colums. Hence, we can use least-squares method.

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} \qquad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$$\hat{y} = X\beta$$

$$\beta = \left(X^T X\right)^{-1} X^T Y$$

(2)

$$g(x) = \beta_0 + \sin(\beta_1 x) + \cos(\beta_2 x)$$

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}, \quad RSS = \sum_{i=1}^{n} \left(g(x^{(i)}) - y^{(i)} \right)^2$$

$$\beta^{(t+1)} = \beta^{(t)} - \eta \left(\frac{\partial (RSS)}{\partial \beta} \right)_{(t)}$$

learning rate

$$\frac{\partial (RSS)}{\partial \beta} = \begin{bmatrix} \dfrac{\partial (RSS)}{\partial \beta_0} \\[2ex] \dfrac{\partial (RSS)}{\partial \beta_1} \\[2ex] \dfrac{\partial (RSS)}{\partial \beta_2} \end{bmatrix}$$

$$\frac{\partial}{\partial \beta_j}(RSS) = \sum_{i=1}^{n} 2\left(g(x^{(i)}) - y^{(i)}\right) \times \frac{\partial}{\partial \beta_j}\left(g(x^{(i)})\right)$$

$$\frac{\partial}{\partial \beta_0}\left(g(x^{(i)})\right) = 1$$

$$\frac{\partial}{\partial \beta_1}\left(g(x^{(i)})\right) = x^{(i)} \cos(\beta_1 x^{(i)})$$

$$\frac{\partial}{\partial \beta_2}\left(g(x^{(i)})\right) = -x^{(i)} \sin(\beta_2 x^{(i)})$$

Substitute the 3 terms in above equation to get derivatives with β_j $\forall\ j = 0, 1, 2$

Homework2 - Sai Nikhil

February 16, 2022

1 Problem 2

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
x = np.array([1.2, 3.2, 5.1, 3.5, 2.6]).reshape((-1, 1))
y = np.array([7.8, 1.2, 6.4, 2.6, 8.1]).reshape((-1, 1))
```

[3]:
```
def normal_equation(x, y, lmd=0):
    n, d = x.shape[0], x.shape[1]
    ones = np.ones(n).reshape(-1, 1)
    x = np.hstack((ones, x))
    return np.linalg.inv(x.T @ x + (lmd**2) * np.eye(d + 1)) @ x.T @ y
```

[4]:
```
lmd = [0, 1, 10]
theta = []
print("2a)")
for i, l in enumerate(lmd):
    theta.append(normal_equation(x, y, l))
    print(f"Equation for lambda = {l} is y = {theta[i][1][0]} * x +␣
 ↪{theta[i][0][0]}")
```

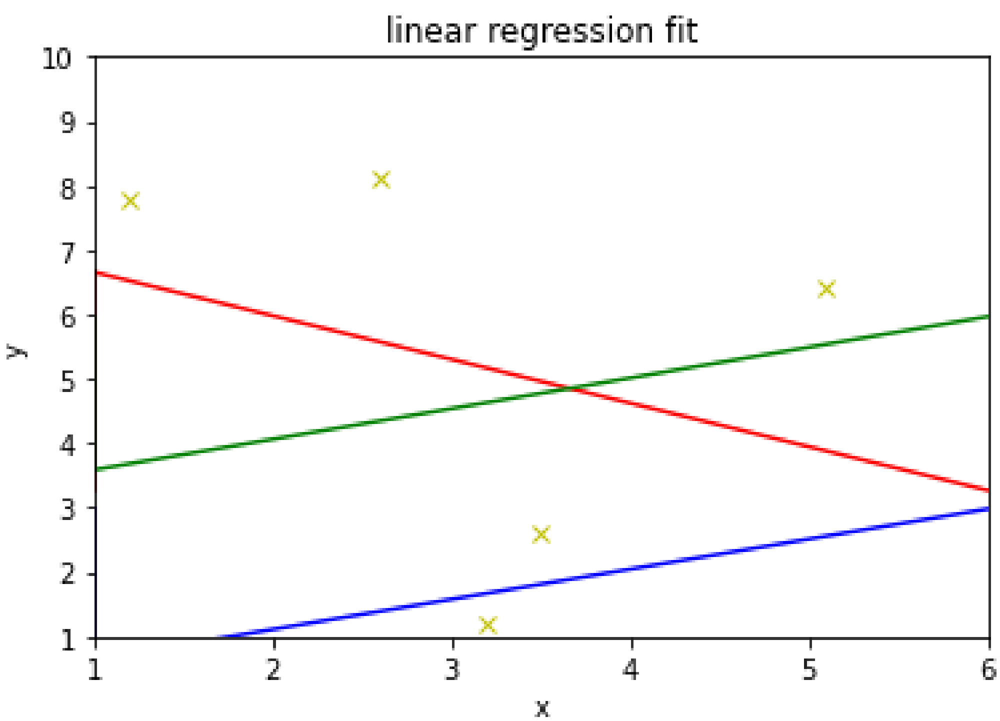
```
2a)
Equation for lambda = 0 is y = -0.6766317887394105 * x + 7.331091180866961
Equation for lambda = 1 is y = 0.47491248541423553 * x + 3.1152275379229852
Equation for lambda = 10 is y = 0.4671668474177275 * x + 0.1791637826693662
```

[5]:
```
x1 = np.linspace(1, 6, 100).reshape((-1, 1))
n, d = x1.shape[0], x1.shape[1]
ones = np.ones(n).reshape(-1, 1)
x1 = np.hstack((ones, x1))
y1 = x1 @ theta[0]

x2 = np.linspace(1, 6, 100).reshape((-1, 1))
n, d = x2.shape[0], x2.shape[1]
ones = np.ones(n).reshape(-1, 1)
x2 = np.hstack((ones, x2))
y2 = x2 @ theta[1]
```

```
x3 = np.linspace(1, 6, 100).reshape((-1, 1))
n, d = x3.shape[0], x3.shape[1]
ones = np.ones(n).reshape(-1, 1)
x3 = np.hstack((ones, x3))
y3 = x3 @ theta[2]


plt.plot(x1, y1, '-r')
plt.plot(x2, y2, '-g')
plt.plot(x3, y3, '-b')
plt.plot(x, y, 'yx')
plt.axis([1, 6, 1, 10])
plt.xlabel('x')
plt.ylabel('y')
plt.title('linear regression fit')
plt.show()
```



1.1 2b

From the above plots, it is clear that when $\lambda = 0$ (or no regularization), it is overfitting the outliers (Red line). When $\lambda = 1$, it is just fitting the data (Green line) and not sensitive to outliers. When $\lambda = 10$, there is too much of regularization. Hence, it is underfitting the data (Blue line).

2 Problem 4

[6]:
```
def get_gradient(beta, g, x, y):
    del_g = np.array([x * np.cos(beta[1][0] * x), -x * np.sin(beta[2][0] * x)]).
↪reshape((10, 2))
    n = x.shape[0]
    del_g = np.hstack((del_g, np.ones(n).reshape(-1, 1)))
    return np.sum(2 * (g(beta, x) - y) * del_g, axis=0).reshape((-1, 1))
```

[7]:
```
def g(beta, x):
    return beta[0][0] + np.sin(beta[1][0] * x) + np.cos(beta[2][0] * x)
```

[8]:
```
def RSS(beta, g, x, y):
    return np.sum((g(beta, x) - y)**2)
```

[9]:
```
def gradient_descent(g, x, y, lr=0.01, iterations=100, threshold=0.001):
    beta = np.array([np.random.rand(), np.random.rand(), np.random.rand()]).
↪reshape((-1, 1))
    beta_prev = beta

    losses = []
    for i in range(iterations):
        gradient = get_gradient(beta_prev, g, x, y)
        beta = beta_prev - lr * gradient
        if np.all(np.abs(beta - beta_prev) < threshold):
            print(f"Convergence found at iteration {i}.")
            break
        beta_prev = beta
        losses.append(RSS(beta, g, x, y))

    return losses, beta
```

[10]:
```
x = np.array([0, 2, 4, 6, 8, 10, 12, 14, 16, 18])
y = np.array([2.85, 1.5, 0.49, 1.57, 1.9, 0.6, 0.38, 2.33, 1.65, 0.3])
x = x.reshape((-1, 1))
y = y.reshape((-1, 1))
```

[11]:
```
losses, beta = gradient_descent(g, x, y, lr=0.0001, iterations=50, threshold=0.
↪00001)
```

[12]:
```
plt.plot(losses, 'rx')
plt.xlabel('iterations')
plt.ylabel('losses')
plt.title('iterations vs loss plot')
plt.show()
```


iterations vs loss plot
losses
11.4
11.2
11.0
10.8
10.6
10.4
10.2
0
10
20
30
40
50
iterations

```
[ ]:
```