1 a)

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

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

For Critical point,

$$\frac{\partial}{\partial \theta}(RSS(\theta)) = 0$$

$$\Rightarrow -2X^T Y + 2X^T X\theta = 0$$

$$\Rightarrow \theta = (X^T X)^{-1} X^T Y$$

1 b)

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

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

$$= Y^T Y - 2\theta^T X^T Y + \theta^T \left(X^T X + \lambda^2 I_n\right)\theta$$

For Critical point,

$$\frac{\partial}{\partial \theta}\left(\text{Ridge}_{\lambda}(\theta)\right) = 0$$

$$\Rightarrow -2X^T Y + 2\left(X^T X + \lambda^2 I_n\right)\theta = 0$$

$$\theta = \left(X^T X + \lambda^2 I_n\right)^{-1} X^T Y$$

Homework2 - Problem2 (Sakshi)

February 16, 2022

1 Problem 2

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[3]:
```
x = np.array([1.2, 3.2, 5.1, 3.5, 2.6])
y = np.array([7.8, 1.2, 6.4, 2.6, 8.1])
```

[5]:
```
# convert to column vectors
x = x.reshape((-1, 1))
y = y.reshape((-1, 1))
```

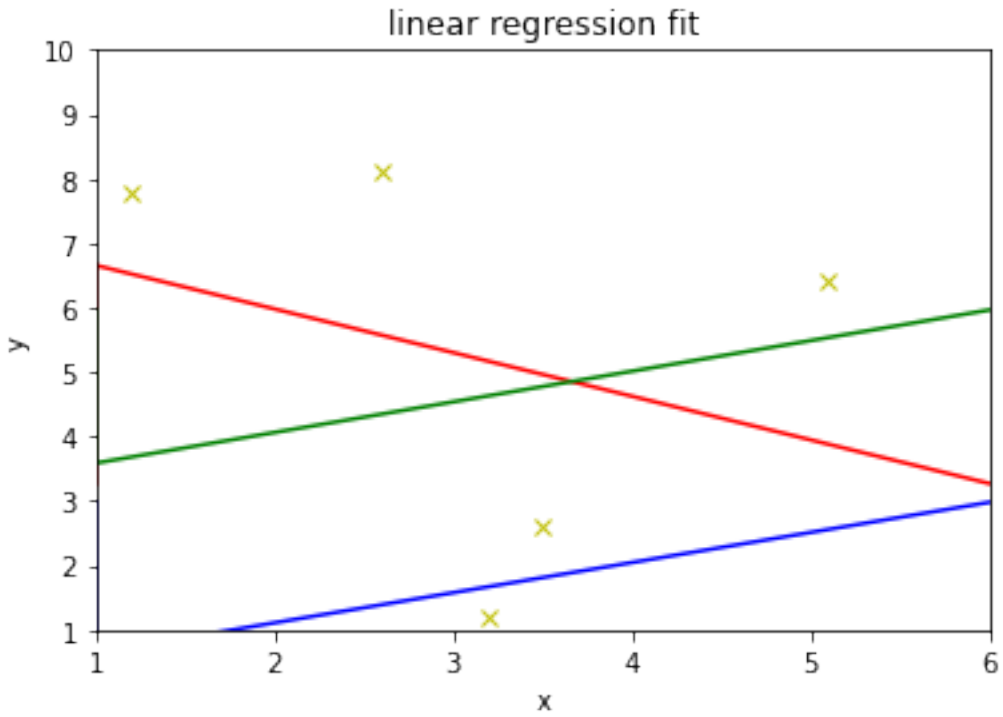
[6]:
```
def least_squares(x, y, reg=0):
    n, d = x.shape[0], x.shape[1]
    # append x to a column of ones. bias trick
    x = np.hstack((np.ones(n).reshape(-1, 1), x))
    solution = np.linalg.inv(x.T @ x + (reg**2) * np.eye(d + 1)) @ x.T @ y
    return solution
```

1.1 Part 2a)

[10]:
```
regularization_constants = [0, 1, 10]
solutions = []
for i, reg in enumerate(regularization_constants):
    solution = least_squares(x, y, reg)
    print(f"Equation of line when regularization = {reg} is y =␣
 ↪{solution[1][0]} * x + {solution[0][0]}")
    solutions.append(solution)
```

```
Equation of line when regularization = 0 is y = -0.6766317887394105 * x +
7.331091180866961
Equation of line when regularization = 1 is y = 0.47491248541423553 * x +
3.1152275379229852
Equation of line when regularization = 10 is y = 0.4671668474177275 * x +
0.1791637826693662
```

[13]:
```
# defining input space
x_new = np.linspace(1, 6, 100).reshape((-1, 1))
n, d = x_new.shape[0], x_new.shape[1]
# append x to a column of ones. bias trick
x_new = np.hstack((np.ones(n).reshape(-1, 1), x_new))

y1 = x_new @ solutions[0]
y2 = x_new @ solutions[1]
y3 = x_new @ solutions[2]

# 3 new plots one each for different regularization
plt.plot(x_new, y1, '-r')
plt.plot(x_new, y2, '-g')
plt.plot(x_new, y3, '-b')

# actual x and y scatter plot
plt.plot(x, y, 'yx')

# defining space for x and y
plt.axis([1, 6, 1, 10])

plt.xlabel('x')
plt.ylabel('y')
plt.title('linear regression fit')
plt.show()
```



1.2 2b

The above plot says that when regularization constant is 0, it is overfitting the data and is giving weight to outliers. This can be seen above in red line. It also says that when regularization constant is 1, the line fits the data just right. This can be seen above in green line. When regularization constant is 10, there is too much of regularization. This can be seen above in the blue line.

[ ]:

3a) Given,

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

$$= \underbrace{(Y - X\theta)^{\top}}_{1 \times n} \underbrace{W}_{n \times n} \underbrace{(Y - X\theta)}_{n \times 1}$$

$n \times (d+1)$

$(d+1) \times 1$

Here,

$$W = \begin{bmatrix} w^{(1)} & & & 0 \\ & w^{(2)} & & \\ & & \ddots & \\ 0 & & & w^{(n)} \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta^{(0)} \\ \theta^{(1)} \\ \vdots \\ \theta^{(d)} \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & x^{(1)(1)} & \cdots & x^{(1)(d)} \\ 1 & x^{(2)(1)} & \cdots & x^{(2)(d)} \\ \vdots & & & \vdots \\ 1 & x^{(n)(1)} & \cdots & x^{(n)(d)} \end{bmatrix} \qquad Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

$$J(\vec{\theta};\vec{x}) = Y^T W Y - 2\theta^T X^T W Y + \theta^T X^T W X \theta$$

$\Rightarrow$

(a) $$\frac{\partial}{\partial \theta}\left(J(\vec{\theta};\vec{x})\right) = 2X^T W X\,\theta - 2X^T W Y$$

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

(b) $$\text{Hessian } (H) = \frac{\partial^2}{\partial \theta^2}\left(J(\vec{\theta};\vec{x})\right)$$

$$= 2X^T W X$$

(c) Gradient descent

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

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

(d) Newton's method

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1} \times \left(\frac{\partial}{\partial \theta} \left(J(\vec{\theta}; \vec{x}) \right) \right)^{(t)}$$

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

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

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

closed form solution

(4)

1)

Given,

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

Input data matrix for "n" points,

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

These two columns are linearly independent as,

$$a\sin(x_i) + b\cos(x_i) = 0$$

$$\Leftrightarrow \sqrt{a^2+b^2}\left(\frac{a}{\sqrt{a^2+b^2}}\sin(x_i) + \frac{b}{\sqrt{a^2+b^2}}\cos(x_i)\right) = 0$$

$\frac{a}{\sqrt{a^2+b^2}} = \cos(\alpha)$, $\frac{b}{\sqrt{a^2+b^2}} = \sin(\alpha)$

$$\Leftrightarrow \sqrt{a^2+b^2}\,\sin(x_i+\alpha) = 0$$

$$\Leftrightarrow a = b = 0$$

$\therefore$ We can use least squares method

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

Using least squares,

we get,

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

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

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

$$\frac{\partial (RSS)}{\partial \beta} = \begin{bmatrix} \frac{\partial (RSS)}{\partial \beta_0} \\ \frac{\partial}{\partial \beta_1}(RSS) \\ \frac{\partial}{\partial \beta_2}(RSS) \end{bmatrix} \quad \text{—(i)}$$

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

(ii)

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

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

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

Substituting (iii), (iv) & (v) in (ii) and re-writing (i) gives,

$$\frac{\partial (RSS)}{\partial \beta} = \begin{bmatrix} \sum_{i=1}^{n} 2\left(g(x^{(i)}) - y^{(i)}\right) \times 1 \\ \sum_{i=1}^{n} 2\left(g(x^{(i)}) - y^{(i)}\right) \times \left(x^{(i)} \cos\left(\beta_1 x^{(i)}\right)\right) \\ \sum_{i=1}^{n} 2\left(g(x^{(i)}) - y^{(i)}\right) \times \left(-x^{(i)} \sin\left(\beta_2 x^{(i)}\right)\right) \end{bmatrix}$$

Look at Code for further analysis

Homework2 - Problem4 (Sakshi)

February 16, 2022

1 Problem 4

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

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

[3]:
```
# calculate reduced sum of squares loss
def RSS(g, x, y, beta):
    return np.sum((g(x, beta) - y)**2)
```

[4]:
```
# calculate gradient
def get_gradient(g, x, y, beta):
    del_g = np.hstack((np.array([x * np.cos(beta[1][0] * x), -x * np.
 ↪sin(beta[2][0] * x)]).reshape((10, 2)), np.ones(x.shape[0]).reshape(-1, 1)))
    gradient = np.sum(2 * (g(x, beta) - y) * del_g, axis=0).reshape((-1, 1))
    return gradient
```

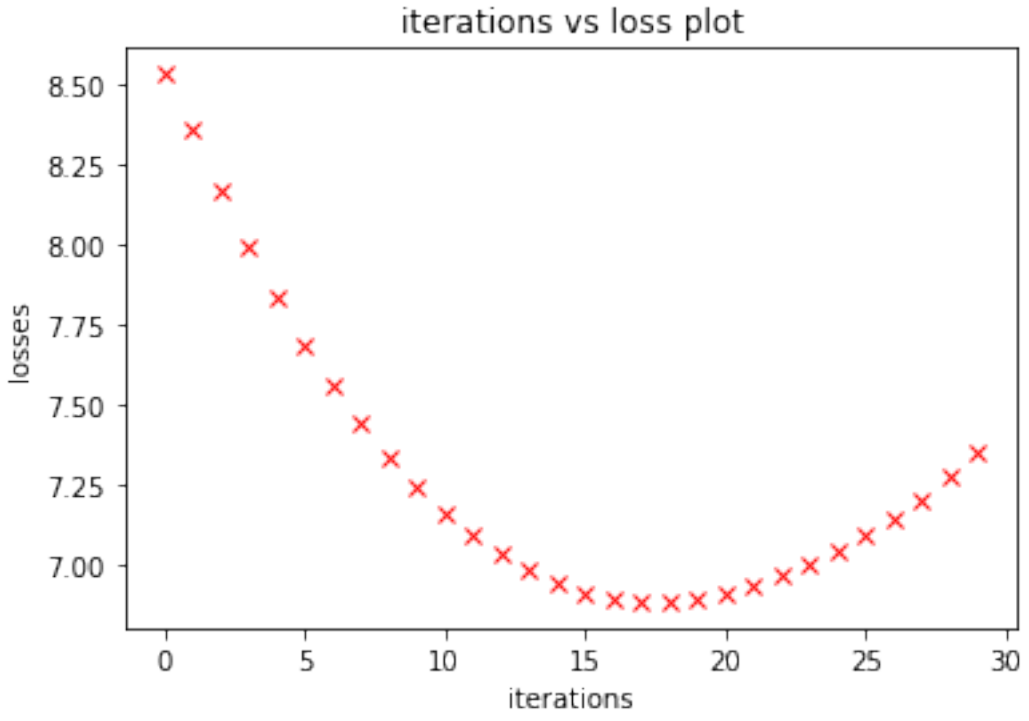
[5]:
```
# implementation of batch gradient descent
def gradient_descent(g, x, y, learning_rate=0.01, iterations=100,␣
 ↪error_threshold=0.001):
    # random parameter initialization
    param = np.array([np.random.rand(), np.random.rand(), np.random.rand()]).
 ↪reshape((-1, 1))
    prev = param

    losses = []
    for i in range(iterations):
        gradient = get_gradient(g, x, y, prev)
        param = prev - learning_rate * gradient
        if np.all(np.abs(param - prev) < error_threshold):
            print(f"Convergence found at iteration {i}.")
            break
        prev = param
        # collect loss w.r.t. number of iterations
```

```
        losses.append(RSS(g, x, y, param))

    return losses, param
```

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

[7]:
```
losses, param = gradient_descent(g, x, y, learning_rate=0.001, iterations=30,␣
 ↪error_threshold=0.0001)
```

[8]:
```
plt.plot(losses, 'rx')
plt.xlabel('iterations')
plt.ylabel('losses')
plt.title('iterations vs loss plot')
plt.show()
```



[ ]:
```

```