COMP9417 - Machine Learning Homework 2: Newton's Method and Mean Squared Error of Estimators

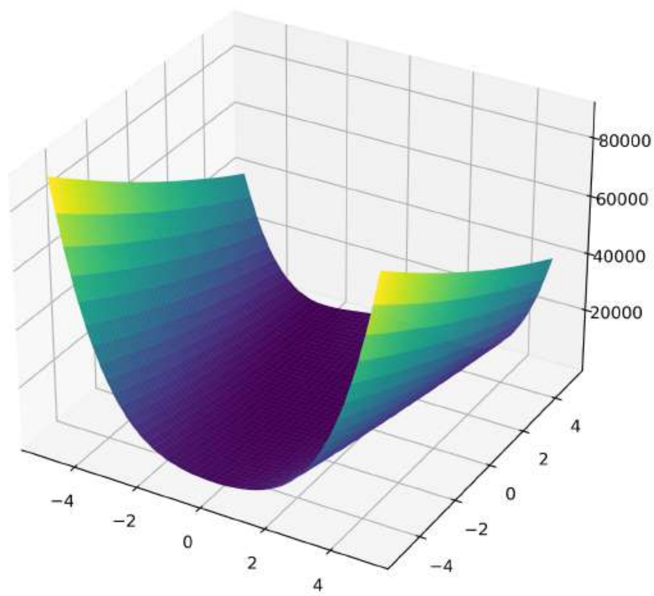
Question 1

a)

When the input x^k is a vector, then the first derivative of x^k is $\bigtriangledown f(x^k)$, and the second derivative of x^k is Hessian matrix which can be calulated as $H(x^k) = \bigtriangledown(\bigtriangledown f(x^k))$. According to the update rule equation 1, we have $x^{k+1} = x^k - \frac{\bigtriangledown f(x^k)}{\bigtriangledown(\bigtriangledown f(x^k))}$ by placing the first and second derivatives in the same position. Therefore, we have $x^{k+1} = x^k - (H(x^k))^{-1}\bigtriangledown f(x^k)$ which is a generalization of equation 1 to functions with vector inputs.

b)

The plot:



The code used to generate the plot:

q1_b.py > ...

```python
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import numpy as np


def func(x, y):
    return 100 * (y - x ** 2) ** 2 + (1 - x) ** 2


# create two one-dimensional grids using linspace
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
# combine the two one-dimensional grids into one two-dimensional grid
X, Y = np.meshgrid(x, y)

Z = func(X, Y)

fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')
plt.savefig('qb.png', dpi=400)
plt.show()
```

The gradient and Hessian of f:

b) $f(x,y) = 100(y-x^2)^2 + (1-x)^2$

$$dx = \frac{\partial f(x,y)}{\partial x} = \frac{\partial}{\partial x}(100(y-x^2)^2) + \frac{\partial}{\partial x}((1-x)^2)$$

$$= -400x(y-x^2) - 2(1-x)$$

$$dy = \frac{\partial f(x,y)}{\partial y} = \frac{\partial}{\partial y}(100(y-x^2)^2) + \frac{\partial}{\partial y}((1-x)^2)$$

$$= 200(y-x^2)$$

$\therefore$ $\cancel{f(x)}$ gradient $= \nabla f(x,y) = (-400x(y-x^2)-2(1-x),\ 200(y-x^2))^T$

$$h00 = \frac{\partial^2 f(x,y)}{\partial x^2} = \frac{\cancel{\partial x}\, dx}{\partial x} = \frac{\partial(-400x(y-x^2))}{\partial x} + \frac{\partial(-2(1-x))}{\partial x}$$

$$= -400(y-3x^2) + 2$$

$$h01 = \frac{\partial^2 f(x,y)}{\partial x \partial y} = \frac{dx}{\partial y} = \frac{\partial(-400x(y-x^2))}{\partial y} + \frac{\partial(-2(1-x))}{\partial y}$$

$$= -400x$$

$$h10 = \frac{\partial^2 f(x,y)}{\partial y \partial x} = \frac{dy}{\partial x} = \frac{\partial(200(y-x^2))}{\partial x}$$

$$= -400x$$

$$h11 = \frac{\partial^2 f(x,y)}{\partial y^2} = \frac{dy}{\partial y} = \frac{\partial(200(y-x^2))}{\partial y}$$

$$= 200$$

$$\therefore \text{hessian} = \begin{bmatrix} -400(y-3x^2)+2 & -400x \\ -400x & 200 \end{bmatrix}$$

c)

The iterations:

```
(base) ➜  hw2 python -u "/Users/yueyifei/Desktop/COMP9417/homework/hw2/q1_c.py"
k = 0, x(0) = [-1.2,1.0]
k = 1, x(1) = [-1.1752808988764043,1.3806741573033703]
k = 2, x(2) = [0.763114871176324,-3.175033854747852]
k = 3, x(3) = [0.7634296788839284,0.5828247754969258]
k = 4, x(4) = [0.999995311085015,0.9440273238533163]
k = 5, x(5) = [0.999995695653691,0.9999913913257614]
k = 6, x(6) = [0.9999999999999999,0.9999999999814724]
```

A screen shot of code for c):

```python
q1_c.py > i
import numpy as np


def grad(x, y):
    dx = -400 * x * (y - x ** 2) - 2 * (1 - x)
    dy = 200 * (y - x ** 2)
    return np.transpose(np.array([dx, dy]))

def hess(x, y):
    h00 = -400 * (y - 3 * x ** 2) + 2
    h01 = -400 * x
    h10 = -400 * x
    h11 = 200
    return np.array([[h00, h01], [h10, h11]])

x_list = []
x0 = np.transpose(np.array([-1.2, 1]))
x_list.append(x0)

x = x0
k = 1

while True:
    if np.linalg.norm(grad(x[0], x[1]), ord=2) <= 10 ** -6:
        break
    new_x = x - np.matmul(np.linalg.inv(hess(x[0], x[1])), grad(x[0], x[1]))
    x = new_x
    x_list.append(new_x)
    k = k + 1

for i in range(0, len(x_list)):
    x = x_list[i]
    print("k = {}, x({}) = [{},{}]".format(i, i, x[0], x[1]))
```

Question 2

a)

The coordinate level GD updates:

a) $L(\beta_0, \beta) = \frac{1}{2}\|\beta\|_2^2 + \frac{\lambda}{n}\sum_{i=1}^{n}\left[y_i \ln\left(\frac{1}{\sigma(\beta_0+\beta^T x_i)}\right) + (1-y_i)\ln\left(\frac{1}{1-\sigma(\beta_0+\beta^T x_i)}\right)\right]$

let $z_i = \beta_0 + \beta^T x_i$,

Then we have.

$$L(\beta_0, \beta) = \frac{1}{2}\|\beta\|_2^2 - \frac{\lambda}{n}\sum_{i=1}^{n}\left[y_i \ln\sigma(z_i) + (1-y_i)\ln(1-\sigma(z_i))\right]$$

Therefore, when $j \neq 0$,

$$\frac{\partial L(\beta_0, \beta)}{\partial \beta_j} = \beta_j - \frac{\lambda}{n}\sum_{i=1}^{n}\left[\frac{y_i}{\sigma(z)} - \frac{1-y_i}{1-\sigma(z)}\right]\sigma(z_i)(1-\sigma(z_i))$$

$$= \beta_j - \frac{\lambda}{n}\sum_{i=1}^{n}\left[y_i - \sigma(z_i)\right]$$

$$= \beta_j - \frac{\lambda}{n}\sum_{i=1}^{n}\left[y_i - \frac{1}{1+e^{-\beta_0-\beta^T x_i}}\right]x_{ij}$$

$$\therefore \frac{\partial L(\beta_0, \beta)}{\partial \beta_j} = \beta_j + \frac{\lambda}{n}\sum_{i=1}^{n}\left[\frac{1}{1+e^{-\beta_0-\beta^T x_i}} - y_i\right]x_{ij}$$

and when $j = 0$,

$$\frac{\partial L(\beta_0, \beta)}{\partial \beta_0} = -\frac{\lambda}{n}\sum_{i=1}^{n}\left[y_i - \sigma(z_1)\right]$$

$$= \frac{\lambda}{n}\sum_{i=1}^{n}\left[\frac{1}{1+e^{-\beta_0-\beta^T x_i}} - y_i\right]$$

$$\therefore \frac{\partial L(\beta_0, \beta)}{\partial \beta_0} = \frac{\lambda}{n}\sum_{i=1}^{n}\left[\frac{1}{1+e^{-\beta_0-\beta^T x_i}} - y_i\right]$$

b)

The vectorized GD updates:

b) Since $L(\beta_0,\beta) = \frac{1}{2}||\beta||_2^2 - \frac{\lambda}{n}\sum_{i=1}^{n}[y_i \ln\sigma(z_i) + (1-y_i)\ln(1-\sigma(z_i))]$

$z_i = \beta_0 + \beta^T x_i$

Then, $\frac{\partial L(\beta_0,\beta)}{\partial \beta} = \beta - \frac{\lambda}{n}\sum_{i=1}^{n}[y_i - \sigma(z_i)]$

$$= \beta + \frac{\lambda}{n}\sum_{i=1}^{n}\left[\frac{1}{1+e^{-\beta_0-\beta^T x_i}} - y_i\right] x_i^T$$

$$= \beta + X^T\left(\frac{1}{1+e^{-\beta_0-\beta^T X}} - y\right)$$

$$\frac{\partial L(\beta_0,\beta)}{\partial \beta_0} = \frac{\lambda}{n}\sum_{i=1}^{n}\left[\frac{1}{1+e^{-\beta_0-\beta^T x_i}} - y_i\right] \quad \text{(proved in a1)}$$

Therefore, $\cancel{\left[\frac{\partial L(\beta_0,\beta)}{\partial \beta_0}, \frac{\partial L(\beta_0,\beta)}{\partial \beta}^T\right]^T}$

$$\gamma^{(k)} = \gamma^{(k-1)} - \alpha\left[\frac{\partial L(\beta_0,\beta)}{\partial \beta_0}, \left(\frac{\partial L(\beta_0,\beta)}{\partial \beta}\right)^T\right]^T$$

c)

The vectorized dampened Newton updates:

c) Since $\dfrac{\partial L(\beta_0,\beta)}{\partial \beta_j} = \beta_j + \dfrac{\lambda}{n}\sum_{i=1}^{n}\left[\dfrac{1}{1+e^{-\beta_0-\beta^T x_i}} - y_i\right]x_{ij}$,

let $x_i = [1, x_i]$

then the element $h_{j,k}$ in Hessian matrix is:

$$h_{j,k} = \frac{\partial L(\beta_0,\beta)}{\partial\beta_j\,\partial\beta_k} = \frac{\lambda}{n}\sum_{i=1}^{n}\left[\frac{x_{ik}\,x_{ij}\cdot e^{-\beta_0-\beta^T x_i}}{(1+e^{-\beta_0-\beta^T x_i})^2}\right] \quad (j\neq k)$$

and

$$h_{j,j} = \frac{\partial L(\beta_0,\beta)}{\partial\beta_j\,\partial\beta_j} = 1 + \frac{\lambda}{n}\sum_{i=1}^{n}\left[\frac{x_{ij}\,x_{ij}\cdot e^{-\beta_0-\beta^T x_i}}{(1+e^{-\beta_0-\beta^T x_i})^2}\right] \quad \cancel{(j=k)}$$

(elements on diagonal)

~~Then we have~~.

Therefore, $H = I + X^T S X$

where $I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$

$$S = \frac{\lambda}{n}\begin{bmatrix} \sigma(z_1)(1-\sigma(z_1)) & \cdots & & 0 \\ 0 \;\cdots & \sigma(z_2)(1-\sigma(z_2)) & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma(z_n)(1-\sigma(z_n)) \end{bmatrix}$$

$$\therefore\ \gamma^{(k)} = \gamma^{(k-1)} - \partial \times H^{-1}\left[\frac{\partial L}{\partial \beta_0}, \left(\frac{\partial L}{\partial \beta}\right)^T\right]^T$$

d)

A print out of the rows requested:

```
(base) → hw2 python -u "/Users/yueyifei/Desktop/COMP9417/homework/hw2/q2_def.py"
The first row of X_train:
[-0.93555843  0.67519298  1.3849985 ]
The last row of X_train:
[-1.13301479 -1.09458877  0.96702449]

The first row of X_test:
[-0.29382524  1.36005105  0.26306826]
The last row of X_test:
[-0.29382524 -1.05390413 -1.34833155]

The first row of Y_train:
0
The last row of Y_train:
1

The first row of Y_test:
0
The last row of Y_test:
1
```

A screen shot of code for d):

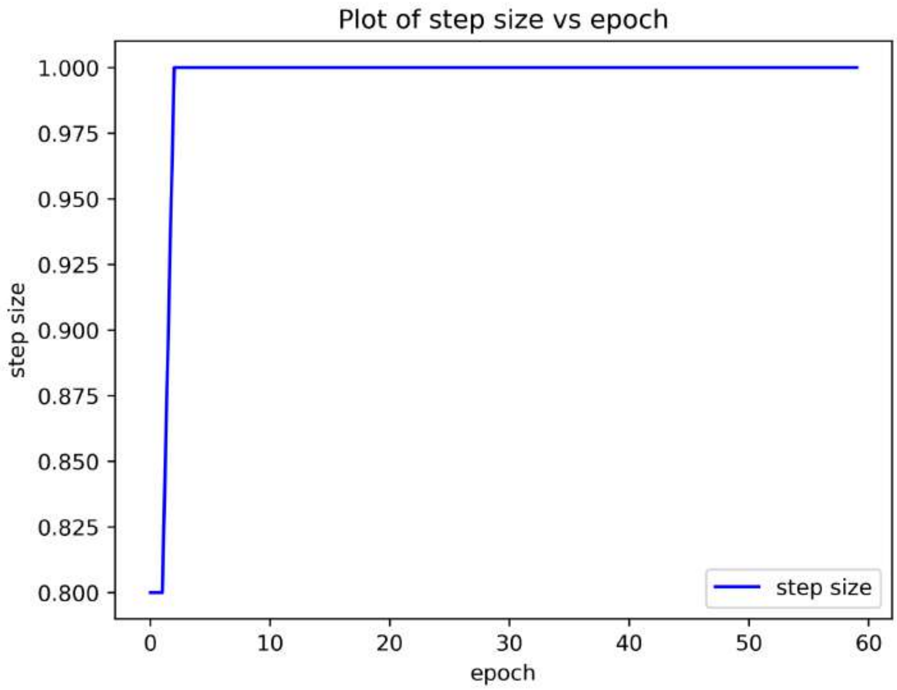
q2_def.py > ...

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from helper import sigmoid, loss
import matplotlib.pyplot as plt

# Question d of Part 2
data = pd.read_csv('songs.csv')

# (I)
remove_features = ['Artist Name', 'Track Name', 'key', 'mode', 'time_signature', 'instrumentalness']
data = data.drop(remove_features, axis=1)

# (II)
data = data[(data['Class'] == 5) | (data['Class'] == 9)]
data['Class'].replace([5, 9], [1, 0], inplace=True)
data = data.reset_index(drop=True)

# (III)
data = data.dropna()

# (IV)
X_train, X_test, Y_train, Y_test = train_test_split(data.iloc[:, 0:-1], data.iloc[:, -1], test_size=0.3,
                                                    random_state=23)
# (V)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

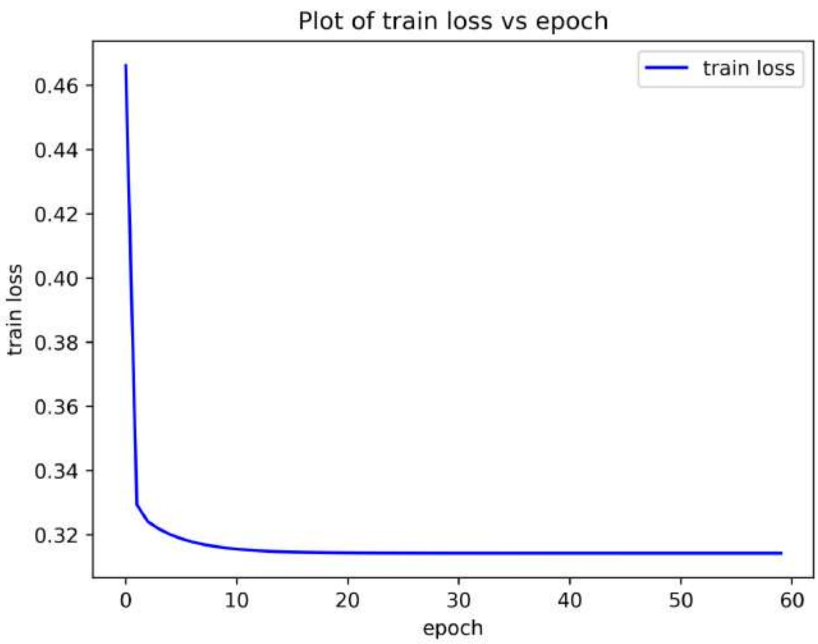
```
# (VI)
Y_train = Y_train.to_numpy()
Y_test = Y_test.to_numpy()

print('The first row of X_train:')
print(X_train[0][:3])
print('The last row of X_train:')
print(X_train[-1][:3])
print()
print('The first row of X_test:')
print(X_test[0][:3])
print('The last row of X_test:')
print(X_test[-1][:3])
print()
print('The first row of Y_train:')
print(Y_train[0])
print('The last row of Y_train:')
print(Y_train[-1])
print()
print('The first row of Y_test:')
print(Y_test[0])
print('The last row of Y_test:')
print(Y_test[-1])
print()
```

e)

The plot of step size vs epoch:



The plot of train loss vs epoch:



Final achieved train and test losses:

```
Final loss achieved by GD algorithm on the train data
0.3142968229209097
Final loss achieved by GD algorithm on the test data
0.31721131314546464
```

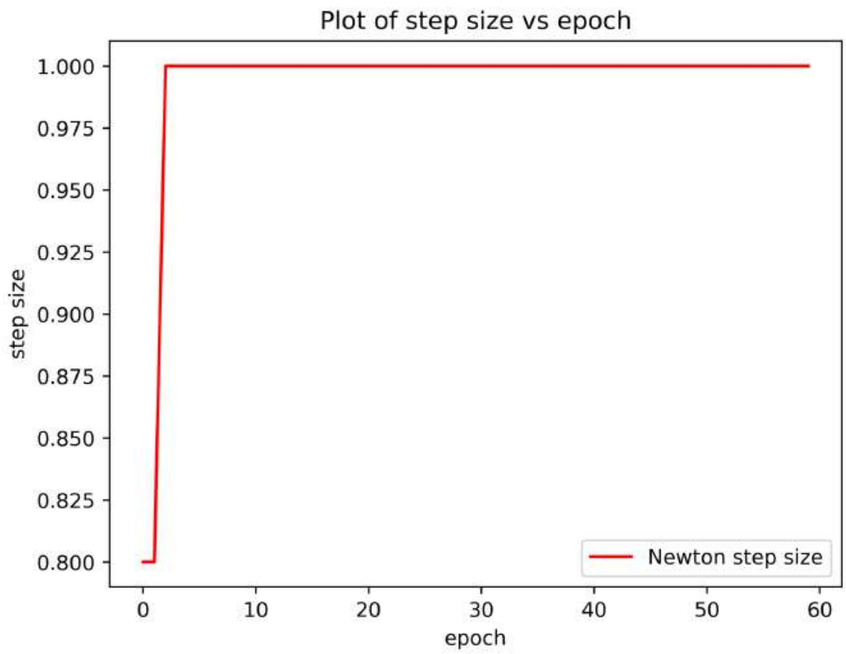
A screen shot of code for e):

```python
# Question e of Part 2

epochs = 60
lam = 0.5
beta_0 = 0
beta = np.ones(len(data.columns) - 1)
gamma_0 = np.insert(beta, 0, beta_0)
a = 0.5
b = 0.8


def grad(gamma, X, y, lam):
    z = np.dot(X, gamma[1:]) + gamma[0]
    grad_beta = gamma[1:] + (lam / len(X)) * np.dot((sigmoid(z) - y), X)
    grad_beta0 = (lam / len(X)) * np.sum((sigmoid(z) - y))
    grad_gamma = np.insert(grad_beta, 0, grad_beta0)
    return grad_gamma


def get_gd_gammas_alphas(gamma_0, X, y, epochs, lam):
    alphas = []
    gammas = []
    gamma = gamma_0
    alphas.append(1)
    gammas.append(gamma)
    # Backtracking line search
    for k in range(epochs):
        alpha = 1
        if loss(gamma - alpha * grad(gamma, X, y, lam), X, y, lam) > loss(gamma, X, y, lam) - a * alpha * np.linalg.norm(grad(gamma, X, y, lam), ord=2)**2:
            alpha = alpha * b
        alphas.append(alpha)
        gamma_updated = gamma - alpha * grad(gamma, X, y, lam)
        gamma = gamma_updated
        gammas.append(gamma_updated)
    return gammas, alphas
```

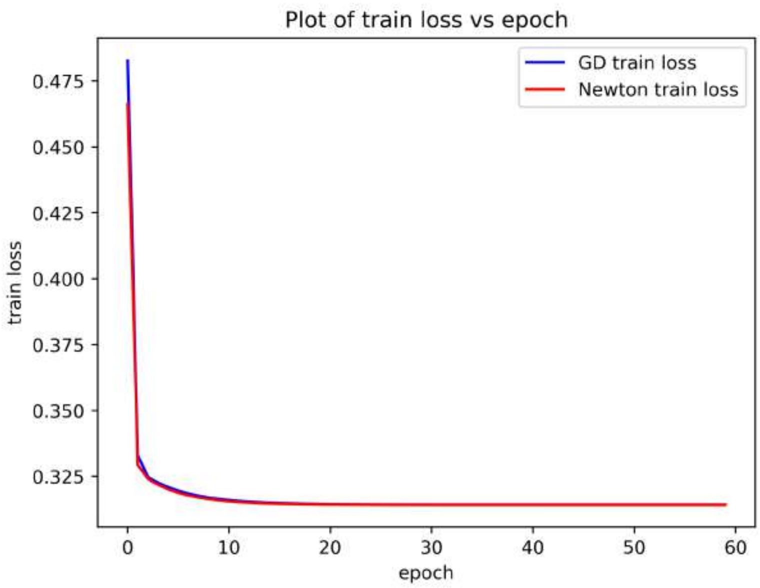
```python

def get_gd_losses(gamma_0, X, y, epochs, lam):
    losses = []
    gammas = get_gd_gammas_alphas(gamma_0, X, y, epochs, lam)[0]
    for gamma in gammas:
        gamma_loss = loss(gamma, X, y, lam)
        losses.append(gamma_loss)
    return losses


alphas = get_gd_gammas_alphas(gamma_0, X_train, Y_train, epochs, lam)[1]
losses_train = get_gd_losses(gamma_0, X_train, Y_train, epochs, lam)
losses_test = get_gd_losses(gamma_0, X_test, Y_test, epochs, lam)
print('Final loss achieved by GD algorithm on the train data')
print(losses_train[-1])
print('Final loss achieved by GD algorithm on the test data')
print(losses_test[-1])

plt.plot(range(epochs), alphas[1:], color='blue', label='step size')
plt.xlabel('epoch')
plt.legend()  # creates legend in top right corner of plot
plt.ylabel('step size')
plt.title('Plot of step size vs epoch')
plt.savefig('qe_step_sizes.png', dpi=400)
plt.show()

plt.plot(range(epochs), losses_train[1:], color='blue', label='train loss')
plt.xlabel('epoch')
plt.legend()  # creates legend in top right corner of plot
plt.ylabel('train loss')
plt.title('Plot of train loss vs epoch')
plt.savefig('qe_train_loss.png', dpi=400)
plt.show()

```

f)

The plot of step size vs epoch:



The plot of train losses (of both GD and Newton) vs epoch:



Final achieved train and test losses:

```
Final loss achieved by Newton algorithm on the train data
0.31429685408159996
Final loss achieved by Newton algorithm on the test data
0.3172113358850347
```

A screen shot of code for f):

```python
epochs = 60
lam = 0.5
beta_0 = 0
beta = np.ones(len(data.columns) - 1)
gamma_0 = np.insert(beta, 0, beta_0)
a = 0.5
b = 0.8


def grad(gamma, X, y, lam):
    z = np.dot(X, gamma[1:]) + gamma[0]
    grad_beta = gamma[1:] + (lam / len(X)) * np.dot((sigmoid(z) - y), X)
    grad_beta0 = (lam / len(X)) * np.sum((sigmoid(z) - y))
    grad_gamma = np.insert(grad_beta, 0, grad_beta0)
    return grad_gamma

```

```python
# Question f of Part 2

def hess(gamma, X, lam):
    # H = I + X.T * S * X
    hess_matrix = np.zeros((11, 11))
    identity_matrix = np.identity(11)
    z = np.dot(X, gamma[1:]) + gamma[0]
    diagnose_matrix = np.diag((lam / len(X)) * sigmoid(z) * (1 - sigmoid(z)))
    col = np.ones((len(X), 1))
    X = np.column_stack((col, X))
    hess_matrix = identity_matrix + X.T @ diagnose_matrix @ X
    return hess_matrix


def get_dn_gammas_alphas(gamma_0, X, y, epochs, lam):
    alphas = []
    gammas = []
    gamma = gamma_0
    alphas.append(1)
    gammas.append(gamma)
    # Backtracking line search
    for k in range(epochs):
        alpha = 1
        if loss(gamma - alpha * grad(gamma, X, y, lam), X, y, lam) > loss(gamma, X, y, lam) - a * alpha * np.linalg.norm(grad(gamma, X, y, lam), ord=2)**2:
            alpha = alpha * b
        alphas.append(alpha)
        gamma_updated = gamma - alpha * np.matmul(np.linalg.inv(hess(gamma, X, lam)), grad(gamma, X, y, lam))
        gamma = gamma_updated
        gammas.append(gamma_updated)
    return gammas, alphas

```

```python

def get_dn_losses(gamma_0, X, y, epochs, lam):
    losses = []
    gammas = get_dn_gammas_alphas(gamma_0, X, y, epochs, lam)[0]
    for gamma in gammas:
        gamma_loss = loss(gamma, X, y, lam)
        losses.append(gamma_loss)
    return losses


alphas = get_dn_gammas_alphas(gamma_0, X_train, Y_train, epochs, lam)[1]
losses_train_dn = get_dn_losses(gamma_0, X_train, Y_train, epochs, lam)
losses_test_dn = get_dn_losses(gamma_0, X_test, Y_test, epochs, lam)
print('Final loss achieved by Newton algorithm on the train data')
print(losses_train_dn[-1])
print('Final loss achieved by Newton algorithm on the test data')
print(losses_test_dn[-1])

plt.plot(range(epochs), alphas[1:], color='red', label='Newton step size')
plt.xlabel('epoch')
plt.legend()  # creates legend in top right corner of plot
plt.ylabel('step size')
plt.title('Plot of step size vs epoch')
plt.savefig('qf_step_sizes.png', dpi=400)
plt.show()

plt.plot(range(epochs), losses_train_dn[1:], color='blue', label='GD train loss')
plt.plot(range(epochs), losses_train[1:], color='red', label='Newton train loss')
plt.xlabel('epoch')
plt.legend()  # creates legend in top right corner of plot
plt.ylabel('train loss')
plt.title('Plot of train loss vs epoch')
plt.savefig('qf_train_loss.png', dpi=400)
plt.show()

```

g)

Because Newton's method requires to calculate the Hessian matrix, but the time complexity of calculating Hessian matrix is higher. It is often more difficult or intractable to derive the second derivative, which requires a lot of computation. If n values are required to compute the first derivative, then n^2 are required for the second derivative. Therefore, GD and its variants are much more popular than Newton's method for solving machine learning problems.

Question 3

a)

a)

For $\hat{\mu}_{MLE}$

since $\hat{\mu}_{MLE} = \bar{X}$,

$$\begin{aligned}
bias(\hat{\mu}_{MLE}) &= bias(\bar{X}) \\
&= E\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) - \mu \\
&= \frac{1}{n}\sum_{i=1}^{n} E(X_i) - \mu \\
&= \frac{1}{n}\sum_{i=1}^{n} \mu - \mu \\
&= \mu - \mu \\
&= 0
\end{aligned}$$

~~For~~

say $\bar{X}$ is an unbiased estimator for μ

$$\begin{aligned}
var(\hat{\mu}_{MLE}) &= Var(\bar{X}) \\
&= var\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) \\
&= \frac{1}{n^2}\sum_{i=1}^{n} var(X_i) \\
&= \frac{1}{n}
\end{aligned}$$

For $\hat{\sigma}^2_{MLE}$

since $\hat{\sigma}^2_{MLE} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2$

$$\begin{aligned}
bias(\hat{\sigma}^2_{MLE}) &= bias\left(\frac{1}{n}\sum_{i=1}^{n}(X_i-\bar{X})^2\right) \\
&= E\left(\frac{1}{n}\sum_{i=1}^{n}(X_i-\bar{X})^2\right) - \sigma^2 \\
&= E\left(\frac{1}{n}\sum_{i=1}^{n}(X_i-\mu+\mu-\bar{X})^2\right) - \sigma^2 \\
&= \frac{1}{n}E\left(\sum_{i=1}^{n}\left((X_i-\mu)^2 - 2(X_i-\mu)(\bar{X}-\mu) + (\bar{X}-\mu)^2\right)\right) - \sigma^2 \\
&= \frac{1}{n}E\left(\sum_{i=1}^{n}(X_i-\mu)^2 - 2n(\bar{X}-\mu)(\bar{X}-\mu) + n(\bar{X}-\mu)^2\right) - \sigma^2 \\
&= \frac{1}{n}E\left(\sum_{i=1}^{n}(X_i-\mu)^2 - n(\bar{X}-\mu)^2\right) - \sigma^2 \\
&= \frac{1}{n}\left(\sum_{i=1}^{n}E((X_i-\mu)^2) - nE((\bar{X}-\mu))^2\right) - \sigma^2 \\
&= \frac{1}{n}(nVar(X) - nVar(\bar{X})) - \sigma^2 \\
&= Var(X) - Var(\bar{X}) - \sigma^2 \\
&= \sigma^2 - \frac{\sigma^2}{n} - \sigma^2 \\
&= -\frac{\sigma^2}{n}
\end{aligned}$$

~~For~~ $Var(\hat{\sigma}^2_{MLE}) = Var\left(\frac{1}{n}\sum_{i=1}^{n}(X_i-\bar{X})^2\right)$

since

$$var\left(\frac{1}{\sigma^2}\sum_{i=1}^{n}(X_i-\bar{X})^2\right) = 2(n-1)$$

$$Var\left(\sum_{i=1}^{n}(X_i-\bar{X})^2\right) = 2\sigma^4(n-1)$$

$$Var\left(\frac{1}{n}\sum_{i=1}^{n}(X_i-\bar{X})^2\right) = \frac{2\sigma^4(n-1)}{n^2}$$

so $var(\hat{\sigma}^2_{MLE}) = \frac{2\sigma^4(n-1)}{n^2}$

b)

The bias and variance of the new estimator:

b) $bias(\sigma^2) = bias(\frac{1}{n-1}\sum_{i=1}^{n}(X_i-\bar{X})^2)$

$= E(\frac{1}{n-1}\sum_{i=1}^{n}(X_i-\bar{X})^2) - \sigma^2$

$\cancel{= \frac{1}{n-1}E(\sum}$

$= \frac{1}{n-1}E(\sum_{i=1}^{n}(X_i-\mu)^2 - n(\bar{X}-\mu)) - \sigma^2$ (as we proved in a))

$= \frac{1}{n-1}(n\,Var(X) - n\,Var(\bar{X})) - \sigma^2$

$= \frac{n\,Var(X)}{n-1} - \frac{n\,Var(\bar{X})}{n-1} - \sigma^2$

$= \frac{n\sigma^2}{n-1} - \frac{\sigma^2}{n-1} - \sigma^2$

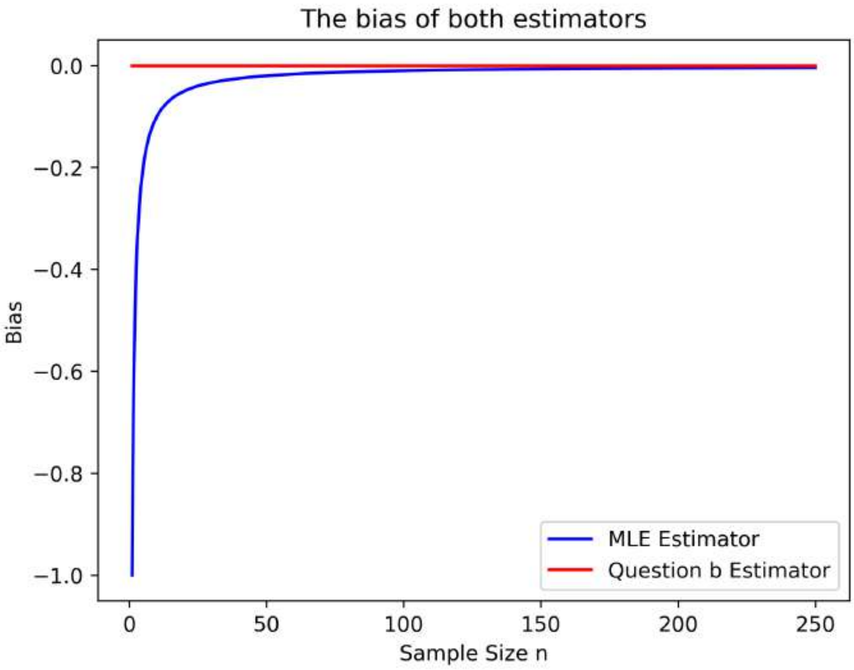
$= 0$

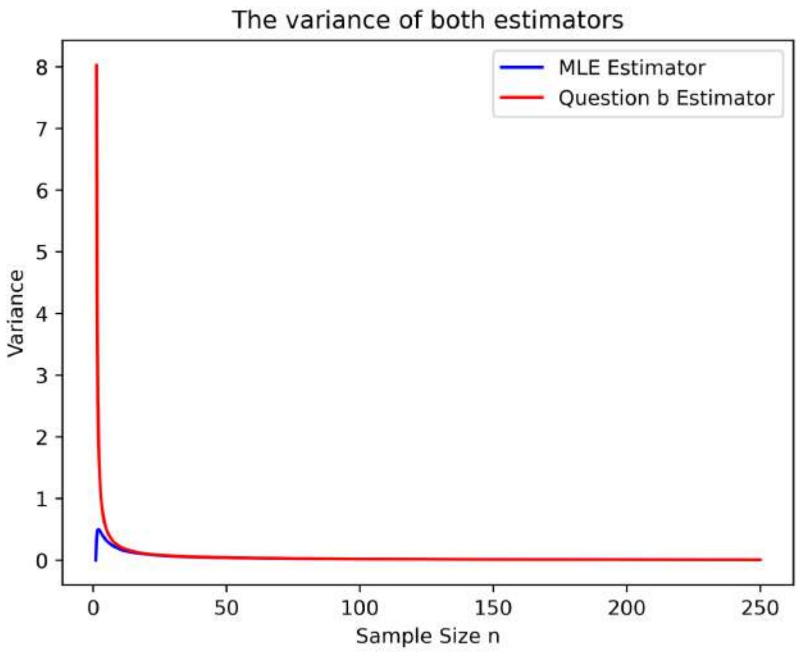
$Var(\sigma^2) = var(\frac{1}{n-1}\sum_{i=1}^{n}(X_i-\bar{X})^2)$

since $Var(\frac{1}{\sigma^2}\sum_{i=1}^{n}(X_i-\bar{X})^2) = 2(n-1)$

$Var(\sum_{i=1}^{n}(X_i-\bar{X})^2) = 2\sigma^4(n-1)$

$Var(\frac{1}{n-1}\sum_{i=1}^{n}(X_i-\bar{X})^2) = \frac{2\sigma^4}{n-1}$


The bias of both estimators
Bias
Sample Size n
MLE Estimator
Question b Estimator
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0
50
100
150
200
250


The variance of both estimators
Variance
Sample Size n
MLE Estimator
Question b Estimator
0
1
2
3
4
5
6
7
8
0
50
100
150
200
250

A screen shot of code for b):

q3_bc.py > MSE_BE

```python
import numpy as np
import matplotlib.pyplot as plt

# Question b of Part 3
sigma = 1

bias_MLE = lambda n: (-1 * sigma ** 2) / n
var_MLE = lambda n: (2 * sigma ** 4 * (n - 1)) / n ** 2

bias_BE = lambda n: n * 0
var_BE = lambda n: (2 * sigma ** 4) / (n - 1)

nrange = np.linspace(1, 250, 1000)

# Plot the bias of both estimators
plt.plot(nrange, bias_MLE(nrange), label="MLE Estimator", color='blue')
plt.plot(nrange, bias_BE(nrange), label="Question b Estimator", color="red")

plt.xlabel('Sample Size n')
plt.ylabel('Bias')
plt.title('The bias of both estimators')
plt.legend()
plt.savefig('q3b_bias.png', dpi=400)
plt.show()

# the variance of both estimators
plt.plot(nrange, var_MLE(nrange), label="MLE Estimator", color='blue')
plt.plot(nrange, var_BE(nrange), label="Question b Estimator", color="red")

plt.xlabel('Sample Size n')
plt.ylabel('Variance')
plt.title('The variance of both estimators')
plt.legend()
plt.savefig('q3b_var.png', dpi=400)
plt.show()
```

c)

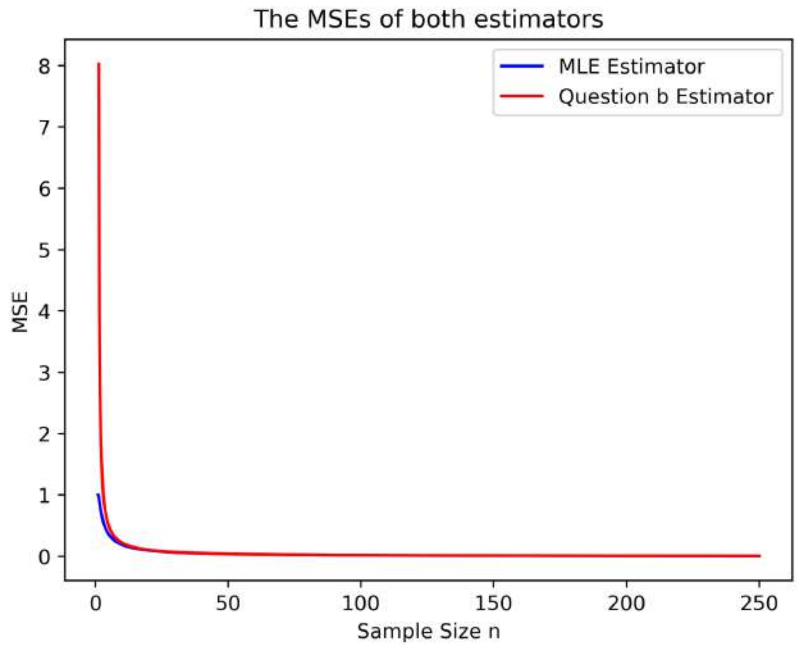
c) MLE: $MSE(\hat{\sigma}^2) = \left(-\frac{\sigma^2}{n}\right)^2 + \frac{2\sigma^4(n-1)}{n^2}$

$$= \cancel{\frac{2n\sigma^4}{n^2}} \ \frac{\sigma^4(2n-1)}{n^2}$$

New Estimator:

$$MSE(\hat{\sigma}^2) = 0^2 + \frac{2\sigma^4}{n-1}$$

$$= \frac{2\sigma^4}{n-1}$$



A screen shot of code for c):

```
# Question c of Part 3
MSE_MLE = lambda n: bias_MLE(n) ** 2 + var_MLE(n)
MSE_BE = lambda n: bias_BE(n) ** 2 + var_BE(n)

# the MSEs of both estimators
plt.plot(nrange, MSE_MLE(nrange), label="MLE Estimator", color='blue')
plt.plot(nrange, MSE_BE(nrange), label="Question b Estimator", color="red")

plt.xlabel('Sample Size n')
plt.ylabel('MSE')
plt.title('The MSEs of both estimators')
plt.legend()
plt.savefig('q3c.png', dpi=400)
plt.show()

```

Discussion:

I think MLE is better. With the increase of the size of samples, the gap between the two estimators will be smaller and smaller, but when the size of samples is small, the MSE of MLE is smaller, so MLE is better to some extent.