Question 1:

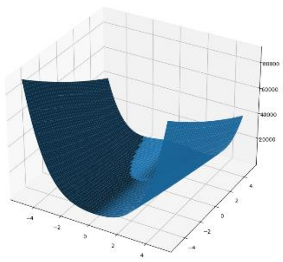
(a) Given the initial iteration point x(0), iterate with Eq.(1) repeatedly until a point with derivative 0 is reached or the maximum number of iterations is reached.

(b)

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


def func(a, b):
    return (b - a**2)**2 * 100 + (1 - a)**2


x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = func(X, Y)
fig = plt.figure(figsize=(10, 10))
# ax = mplot3d.Axes3D(fig)
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z)
plt.show()
```



$$f(x,y) = 100(y-x^2)^2 + (1-x)^2$$
$$= 100(y^2+x^4-2x^2y) + x^2+1-2x$$
$$= 100y^2 + 100x^4 - 200x^2y + x^2+1-2x$$

$$\therefore \text{grad} \begin{cases} \dfrac{\partial f}{\partial x} = 400x^3 - 400xy + 2x - 2 \\ \dfrac{\partial f}{\partial y} = 200y - 200x^2 \end{cases}$$

$$\frac{\partial^2 f}{\partial x^2} = 1200x^2 - 400y + 2$$

$$\frac{\partial^2 f}{\partial y^2} = 200$$

$$\frac{\partial^2 f}{\partial x \partial y} = -400x = \frac{\partial^2 f}{\partial y \partial x}$$

$$\therefore \text{Hessian} \begin{bmatrix} 1200x^2 - 400y + 2 & -400x \\ -400x & \cancel{400x}\ 200 \end{bmatrix}$$

(c)

```python
# c
x0 = np.array([[-1.2], [1]])
count = 0
while True:
    hess = np.array([[1200 * x0[0][0]**2 - 400 * x0[1][0] + 2, -400 * x0[0][0]],
                     [-400 * x0[0][0], 200]])
    hess = np.array(hess, dtype='float')
    hess_inv = np.linalg.inv(hess)
    grad = np.array([[x0[0][0]**3 * 400 - 400 * x0[0][0] * x0[1][0] + 2 * x0[0][0] - 2],
                     [200 * x0[1][0] - 200 * x0[0][0]**2]])
    if np.linalg.norm(grad, 2) <= 0.000001:
        break
    x0 = x0 - np.dot(hess_inv, grad)
    count += 1
    print(count, ':', x0.T[0])
```

```
1 : [-1.1752809   1.38067416]
2 : [ 0.76311487 -3.17503385]
3 : [0.76342968 0.58282478]
4 : [0.99999531 0.94402732]
5 : [0.9999957  0.99999139]
6 : [1. 1.]
```

Question 2:

$$\frac{d\ln x}{dx} = \frac{1}{x} \qquad \frac{d\ln(1-x)}{dx} = \frac{1}{x-1} \qquad \frac{d(\sigma(x))}{dx} = \sigma(x)(1-\sigma(x))$$

when β_0, $x_0 = 1$ $\Rightarrow \beta_0 + \beta^T x_i = \beta'^T x_i$, $\beta_0, \cdots \beta_p$ in β'

so: $$\frac{dL(\beta_0,\beta)}{d\beta_p} = \beta_p + \frac{\lambda}{n}\sum_{i=1}^{n}\left[y_i \cdot \frac{d}{d\beta_p}\ln\left(\frac{1}{\sigma(\beta'^T x_i)}\right) + (1-y_i)\frac{d}{d\beta_p}\ln\left(\frac{1}{1-\sigma(\beta'^T x_i)}\right)\right]$$

$$= \beta_p - \frac{\lambda}{n}\sum_{i=1}^{n}\left(y_i \cdot \frac{1}{\sigma(\beta'^T x_i)} \cdot \frac{d(\sigma(\beta'^T x_i))}{d\beta_p} + (1-y_i)\frac{1}{\sigma(\beta'^T x_i)-1} \cdot \frac{d(\sigma(\beta'^T x_i))}{d\beta_p}\right)$$

$$= \beta_p^{\sigma} - \frac{\lambda}{n}\sum_{i=1}^{n}\left(y_i \frac{1}{\sigma(\beta'^T x_i)} - (1-y_i)\frac{1}{\sigma(\beta'^T x_i)}\right) \cdot \sigma(\beta'^T x_i)(1-\sigma(\beta'^T x_i)) \cdot X_{ip}$$

$$= \beta_p - \frac{\lambda}{n}\sum_{i=1}^{n}\left(y_i(1-\sigma(\beta'^T x_i)) - (1-y_i)\sigma(\beta'^T x_i)\right) X_{ip}$$

$$= \beta_p - \frac{\lambda}{n}\sum_{i=1}^{n}\left(\sigma(\beta'^T x_i) - y_i\right) X_{ip}$$

$$= \beta_p - \frac{\lambda}{n}\sum_{i=1}^{n}\left(\sigma(\beta_0+\beta^T x_i) - y_i\right) X_{ip}$$

$$\frac{d\,\sigma(\beta'^T x_i)}{d\beta_p} = \frac{-1}{(1+e^{-\beta'^T x_i})^2} \cdot (-e^{-\beta'^T x_i}) \cdot \frac{d(\beta'^T x_i)}{d(\beta_p)}$$

$$= \sigma(\beta'^T x_i)(1-\sigma(\beta'^T x_i)) \cdot X_{ip}$$

so: $$\beta_0^{(k)} = \beta_0^{(k-1)} - \alpha \times \left[\beta_0 - \frac{\lambda}{n}\sum_{i=1}^{n}(\sigma(\beta_0+\beta^T x_i) - y_i)\right]$$

$$\beta_p^{(k)} = \beta_p^{(k-1)} - \alpha \times \left[\beta_p - \frac{\lambda}{n}\sum_{i=1}^{n}(\sigma(\beta_0+\beta^T x_i) - y_i) \cdot X_{ip}\right]$$

(b) from (a) $\beta_p^{(k)} = \beta_p^{(k-1)} - \alpha \times \left[\beta_p^{(k-1)} - \frac{\lambda}{n}\sum_{i=1}^{n}(\sigma(\beta_0+\beta^T x_i) - y_i) \cdot X_i\right]$

$y = [\beta_0, \beta^T]^T$ $\quad$ β_0 can has $x_0 = 1$ (assume) $\quad$ $\beta^{(k)} = \beta^{(k-1)} - \alpha[\beta^{(k-1)} + \frac{\lambda}{n} \cdot X.T \cdot (\sigma(x \cdot \beta^{(k-1)} - y)]$

$\Rightarrow y^{(k)} = y^{(k-1)} - \alpha \times [y^{(k-1)} + \frac{\lambda}{n} \cdot X.T \cdot (\sigma(x \cdot y^{(k-1)}) - y)]$

$y^{(k)}[0] = y^{(k)}[0] + \alpha \cdot y^{(k-1)}[0]$

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

Disregard the regularization term,

$$L(\beta^k) + L'(\beta^k)(\beta-\beta^k) + \frac{1}{2}L''(\beta^k)(\beta-\beta^k)^2 = 0$$

$$\Rightarrow \beta^{k+1} = \beta^k - \frac{L'(\beta^k)}{L''(\beta^k)} = \beta^k - H_k^{-1}\left\{\beta^k - \frac{\lambda}{n}\sum_{i=1}^{n}(\sigma(\beta_0+\beta^T x_i) - y_i) \cdot X_i\right\}$$

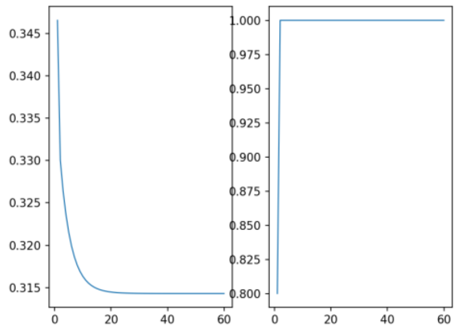
$H_K = \frac{\lambda}{n} \cdot (X^T \cdot \mathrm{diag}(\sigma(\beta_0+\beta^T x_i)) \cdot \mathrm{diag}(1-\sigma(\beta_0+\beta^T x_i)) \cdot X + I'$

($I' = I$ but $I[0][0] = 0$)

(d)

```
first row X_train:[-0.93555843  0.67519298  1.3849985 ]
last row X_train:[-1.13301479 -1.09458877  0.96702449]
first row X_test:[-0.29382524  1.36005105  0.26306826]
last row X_test:[-0.29382524 -1.05390413 -1.34833155]
first row y_train:0
last row y_train:1
first row y_test:0
last row y_test:1
```

(e)




```
Train lost:0.3142968225795515
Test lost:0.31751540540611334
```

```python
# e
def sigmoid(x):
    return 1 / (1 + np.exp(-x))


def loss_function(data_x, label, weights, lam):
    norm_beta_sq = np.linalg.norm(weights[1:], ord=2) ** 2
    z = np.dot(data_x, weights)
    sig_z = sigmoid(z)
    return lam * log_loss(label, sig_z, normalize=True) + 0.5 * norm_beta_sq


init_alpha = 1
a = 0.5
b = 0.8
n = X_train.shape[1]
beta1 = np.zeros((n + 1, 1))
```

```python
la = 0.5
epoch = 60
loss_list1 = []
step_size_list1 = []
X_train = np.insert(X_train, 0, 1, axis=1)
Y_train = Y_train.reshape(2720, 1)
for i in range(epoch):
    loss_num = loss_function(X_train, Y_train, beta1, la)
    grad = beta1 + la * X_train.T @ (sigmoid(X_train @ beta1) - Y_train) / X_train.shape[0]
    grad[0] = grad[0] - beta1[0]
    alpha = init_alpha
```

```python
    while True:
        x1 = loss_function(X_train, Y_train, beta1, la)
        x2 = loss_function(X_train, Y_train, beta1 - alpha * grad, la)
        y = a * alpha * (np.linalg.norm(grad, ord=2) ** 2)
        if x1 - x2 < y:
            alpha = alpha * b
        else:
            break
    beta1 = beta1 - alpha * grad
    loss_list1.append(loss_num)
    step_size_list1.append(alpha)
X_test = np.insert(X_test, 0, 1, axis=1)
Y_test = Y_test.reshape(1166, 1)
train_loss1 = loss_function(X_train, Y_train, beta1, la)
test_loss1 = loss_function(X_test, Y_test, beta1, la)
print(f"Train lost:{train_loss1}")
print(f"Test lost:{test_loss1}")
```

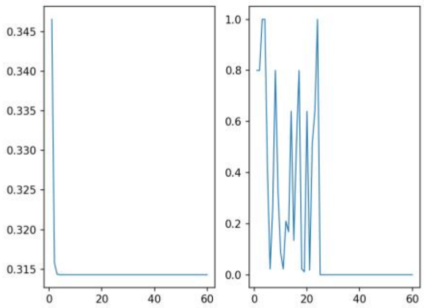
```python
dot = np.linspace(1, 60, num=60)
plt.subplot(1, 2, 1)
plt.plot(dot, loss_list1, linewidth=1)
plt.subplot(1, 2, 2)
plt.plot(dot, step_size_list1, linewidth=1)
plt.show()
```

(f)

```python
beta2 = np.zeros((n + 1, 1))
loss_list2 = []
step_size_list2 = []
for i in range(epoch):
    loss_num = loss_function(X_train, Y_train, beta2, la)
    grad = beta2 + la * X_train.T @ (sigmoid(X_train @ beta2) - Y_train) / X_train.shape[0]
    grad[0] = grad[0] - beta2[0]
    h = sigmoid(X_train @ beta2)
    m = X_train.shape[0]
    l2 = np.eye(n+1)
    l2[0][0] = 0
    order = la / m * ((X_train.T @ np.diag(h.reshape(m)))@np.diag((1-h).reshape(m)))@X_train + l2
    alpha = init_alpha
```

```python
    while True:
        x1 = loss_function(X_train, Y_train, beta2, la)
        x2 = loss_function(X_train, Y_train, beta2 - alpha * grad, la)
        y = a * alpha * (np.linalg.norm(grad, ord=2) ** 2)
        if x1 - x2 < y:
            alpha = alpha * b
        else:
            break
    beta2 = beta2 - alpha * np.dot(np.linalg.inv(order), grad)
    loss_list2.append(loss_num)
    step_size_list2.append(alpha)
```

```python
train_loss2 = loss_function(X_train, Y_train, beta2, la)
test_loss2 = loss_function(X_test, Y_test, beta2, la)
print(f"Train lost:{train_loss2}")
print(f"Test lost:{test_loss2}")
dot = np.linspace(1, 60, num=60)
plt.subplot(1, 2, 1)
plt.plot(dot, loss_list2, linewidth=1)
plt.subplot(1, 2, 2)
plt.plot(dot, step_size_list2, linewidth=1)
plt.show()
plt.plot(dot, loss_list1, linewidth=1, color='orange', label='GD')
plt.plot(dot, loss_list2, linewidth=1, color='green', label='NT')
plt.show()
```



```
Train lost:0.31429681501971396
Test lost:0.31751782119901467
```

(g) The gradient descent method only requires solving the gradient, while methods such as Newton's method require Hessian matrices or calculating analytic solutions, etc. Since machine learning often requires the use of a large sample size, the gradient descent method takes much less time per iteration.

Question3:

(a)

(a) $\text{bias}(\hat{\mu}_{MLE}) = \text{bias}(\bar{X}) = E\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) - \mu$

$= \frac{1}{n}\sum_{i=1}^{n} E(X_i) - \mu$

$= \frac{1}{n}\sum_{i=1}^{n} \mu - \mu = \mu - \mu = 0$ ✓

$\text{bias}(\sigma^2_{MLE}) = E(\sigma^2_{MLE}) - \sigma^2$

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

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

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

$= \sigma^2 - \sigma_{\bar{X}}^2 - \sigma^2$

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

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

$\text{Var}(\hat{\mu}_{MLE}) = \text{Var}(\bar{X})$

$= \text{Var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right)$

$= \frac{1}{n^2}\sum_{i=1}^{n}\text{Var}(X_i) = \frac{1}{n}\sigma^2$

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

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

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

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

(b)

(b) $\text{bias}(\sigma^2_{new}) = E(\sigma^2_{new}) - \sigma^2$

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

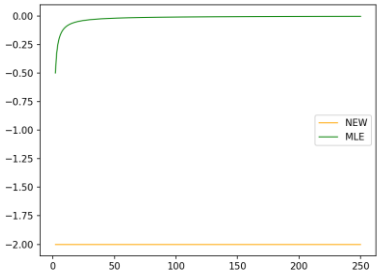
$= \frac{2n-2n^2}{n(n-1)}\sigma^2$

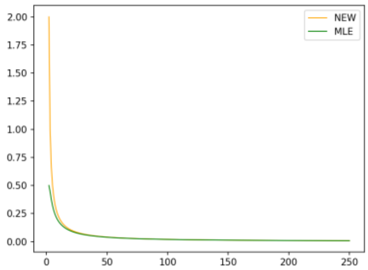
$\text{Var}(\sigma^2_{new}) = \text{Var}\left(\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2\right)$

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

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

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





(c) The calculated MSE curve shows that the MLE estimator is better.
When the sample size n increases, the MSE of the two estimators decreases first and then stabilizes at a certain value.