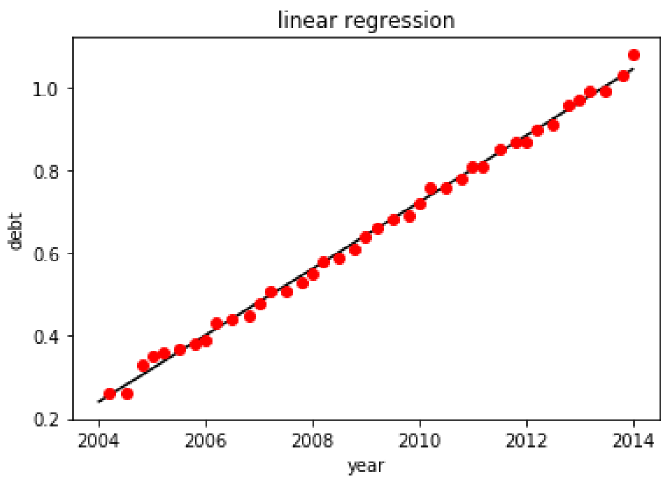
HW2 Huaiyu Wang

3.1



Output: [3.93601056]

w= 8.03244175e-02

b=-1.60729045e+02

In 2050, the student loan debt will be **3.93601056**

Code here:

```
import csv
from pylab import *


def main():
    rfile = 'student_debt.csv'
    csvfile = open(rfile, 'rt')
    data = csv.reader(csvfile, delimiter = ',')
    X = []
    Y = []
    for i, row in enumerate(data):
            X.append(float(row[0]))
            Y.append(float(row[1]))
    X = array(X)
    X_temp = X[:]
    Y = array(Y)
    one = ones((len(X)))
    X = row_stack((one,X))
    X = X.T
    Y = reshape(Y, (len(Y),1))
    weight = linearfit(X,Y)
    print (weight)
```

```
    it = np.arange(2004, 2015, 1)
    plt.ylabel('debt')
    plt.xlabel('year')
    plt.title('linear regression')
    g = weight[1] * it + weight[0]
    plt.plot(it, g, 'k')
    plt.plot(X_temp, Y, 'ro')
    plt.show()
    plt.close()
    print (weight[1] * 2050 + weight[0])

def linearfit(X, Y):
    W =dot(inv(dot(X.T, X)),dot(X.T,Y))
    return W

main()
```

HW2 Huaiyu Wang

3.3 (a) $g(\bar{w}) = \sum_{p=1}^{P} (\bar{x}_p^T \bar{w} - y_p)^2$

$$= \sum_{p=1}^{P} \left[(\bar{x}_p^T \bar{w})^T (\bar{x}_p^T \bar{w}) - 2y_p \bar{x}_p^T \bar{w} + y_p^2 \right]$$

$$= \sum_{p=1}^{P} \left(\bar{w}^T \bar{x}_p \bar{x}_p^T \bar{w} - 2y_p \bar{x}_p^T \bar{w} + y_p^2 \right)$$

$$= \frac{1}{2} \bar{w}^T \underbrace{\left(\sum_{p=1}^{P} 2 \bar{x}_p \bar{x}_p^T \right)}_{\bar{Q}} \bar{w} - \underbrace{\left(\sum_{p=1}^{P} 2y_p \bar{x}_p^T \right)}_{\bar{r}^T} \bar{w} + \underbrace{\sum_{p=1}^{P} y_p^2}_{d}$$

$$= \frac{1}{2} \bar{w}^T \bar{Q} \bar{w} + \bar{r}^T \bar{w} + d$$

(b) for any $\bar{x}_p \bar{x}_p^T = A$, has eigenvalue a with eigenvector $\bar{v}$.

$$\bar{x}_p \bar{x}_p^T \bar{v} = a\bar{v}$$

$$\bar{v}^T a \bar{v} = \bar{v}^T \bar{x}_p \bar{x}_p^T \bar{v} = (\bar{x}_p^T \bar{v})^T \cdot (\bar{x}_p^T \bar{v}) = |\bar{x}_p^T \bar{v}|^2 \geq 0.$$

$$a \bar{v}^T \bar{v} = a \|\bar{v}\|^2 \geq 0 \quad \therefore a \geq 0$$

$\because Q = \sum_{p=1}^{P} 2 \bar{x}_p \bar{x}_p^T$

$\therefore Q$ has all ~~none~~ nonnegative eigenvalues.

(c) $\nabla g(\bar{w}) = \frac{1}{2}(Q + Q^T) \bar{w} + \bar{r}$ $\quad Q = \sum_{p=1}^{P} 2 \bar{x}_p \bar{x}_p^T$ is symmetric

$\therefore Q = Q^T$

$\nabla g(\bar{w}) = Q\bar{w} + \bar{r}$

$\nabla^2 g(\bar{w}) = Q$ has all ~~none~~ nonnegative eigenvalues.

$\therefore g(\bar{w})$ is convex.

(d) $\nabla^2 g(\bar{w}^{k-1})\,\bar{w}^k = \nabla^2 g(\bar{w}^{k-1})\,\bar{w}^{k-1} - \nabla g(\bar{w}^{k-1})$

$$\bar{Q}\bar{w}^k = \bar{Q}\bar{w}^{k-1} - (\bar{Q}\bar{w}^{k-1} + \bar{r}^T)$$

$$\bar{Q}\bar{w}^k = -\bar{r}$$

$$\left(\sum_{p=1}^{P} 2\bar{x}_p \bar{x}_p^T\right)\bar{w}^k = \left(\sum_{p=1}^{P} 2 y_p x_p^T\right)^T$$

$$\left(\sum_{p=1}^{P} \bar{x}_p x_p^T\right)\bar{w} = \sum_{p=1}^{P} \bar{x}_p y_p$$

3.10 (a) $\sigma(t) = \frac{1}{1+e^{-t}}$

$$\sigma^{-1}(\sigma(t)) = \log\left(\frac{\frac{1}{1+e^{-t}}}{1-\frac{1}{1+e^{-t}}}\right) = \log\left(\frac{\frac{1}{1+e^{-t}}}{\frac{e^{-t}}{1+e^{-t}}}\right)$$

$$= \log(e^{t}) = t$$

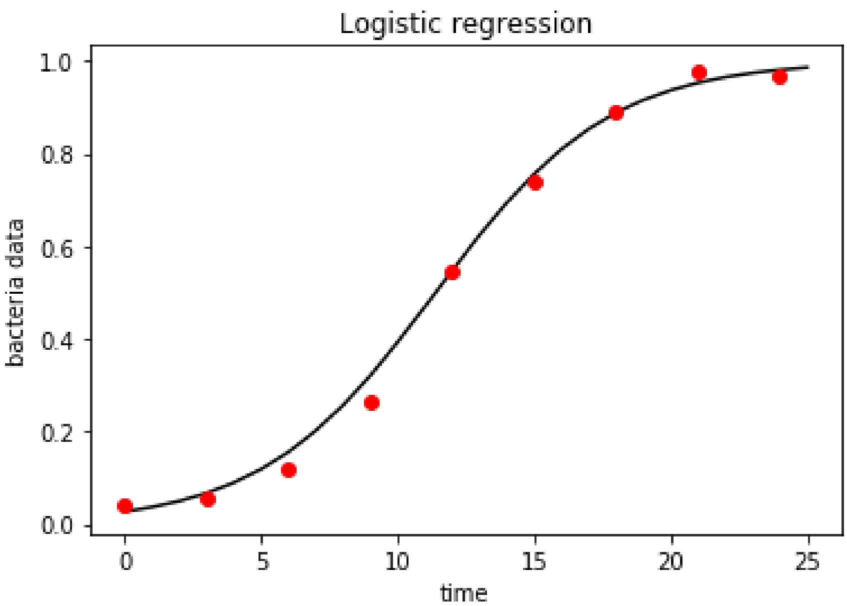
(b) $\sigma(b + x_p w) = y_p$

$$\frac{1}{1+e^{-(b+x_p w)}} = y_p$$

$$e^{-(b+x_p w)} = \frac{y_p - 1}{y_p}$$

$$b + x_p w = \cancel{\log\frac{y_p}{y_p}}\ \log\left(\frac{y_p}{1-y_p}\right) \quad p=1,\dots,P$$

(c)



Code here：

```python
import csv
from pylab import *

def linearfit(X, Y):
    W =dot(inv(dot(X.T, X)),dot(X.T,Y))
    return W

def sigmoid(x):
    return 1 / (1 + exp(-x))

def main():
    rfile = 'bacteria_data.csv'
    csvfile = open(rfile, 'rt')
    data = csv.reader(csvfile, delimiter = ',')
    X = []
    Y = []
    for i, row in enumerate(data):
        X.append(float(row[0]))
        Y.append(float(row[1]))
    X = array(X)
    Y = array(Y)
    X1 = X[:]
    Y1 = Y[:]
    one = ones(len(X))
    X = column_stack((one,X))
    Y = Y.T
    Y = log(Y/(1-Y))
    w = linearfit(X,Y)

    it = np.arange(0,26,1)
    plt.ylabel('bacteria data')
    plt.xlabel('time')
    plt.title('Logistic regression')
    g = sigmoid(w[1] * it + w[0])
    plt.plot(it, g, 'k')
    plt.plot(X1, Y1, 'ro')
    plt.show()
    plt.close()
main()
```

3.11 (a) $\sigma(t)' = \sigma(t)(1-\sigma(t))$

$$\tilde{x}_p = \begin{pmatrix} 1 \\ \bar{x}_p \end{pmatrix} \qquad \tilde{w} = \begin{pmatrix} b \\ \bar{w} \end{pmatrix} \qquad \tilde{x}_p^T \cdot \tilde{w} = b + \bar{x}_p^T \bar{w}$$

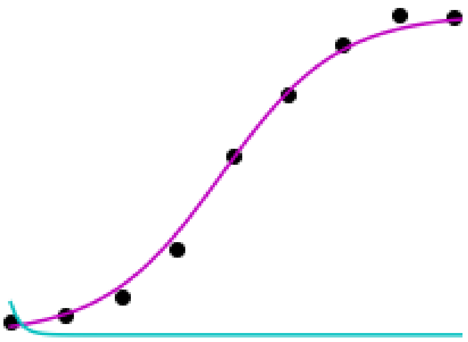
$$\nabla g(b, \bar{w}) = 2\sum_{p=1}^{P} \left(\sigma(b + \bar{x}_p^T \bar{w}) - y_p\right) \cdot \nabla_{\tilde{w}}\left(\sigma(b + \bar{x}_p^T \bar{w})\right) \cdot \nabla_{\tilde{w}}(\bar{x}_p^T \bar{w} + b)$$

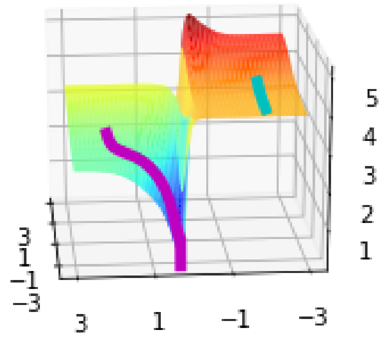
$$\nabla g(\tilde{w}) = 2\sum_{p=1}^{P} \left(\sigma(\tilde{x}_p^T \tilde{w}) - y_p\right) \cdot \nabla_{\tilde{w}}\, \sigma(\tilde{x}_p^T \tilde{w}) \cdot \nabla_{\tilde{w}}(\tilde{x}_p^T \tilde{w})$$

$$= 2\sum_{p=1}^{P} \left(\sigma(\tilde{x}_p^T \tilde{w}) - y_p\right) \cdot \nabla_{\tilde{w}}\left(\sigma(\tilde{x}_p^T \tilde{w})\right) \cdot \tilde{x}_p$$

$$= 2\sum_{p=1}^{P} \left(\sigma(\tilde{x}_p^T \tilde{w}) - y_p\right) \cdot \sigma(\tilde{x}_p^T \tilde{w})\left(1 - \sigma(\tilde{x}_p^T \tilde{w})\right) \tilde{x}_p$$

(b)






```
# YOUR CODE GOES HERE - compute gradient
        grad=0
        for p in range(0,np.shape(X)[0]):
            grad+=2*(s(np.dot(X[p].T,w))-y[p])*s(np.dot(X[p].T,w))*(1-s(np.dot(X[p].T,w)))*X[p]
        grad=np.asarray(grad)
        grad.shape=(2,1)
```

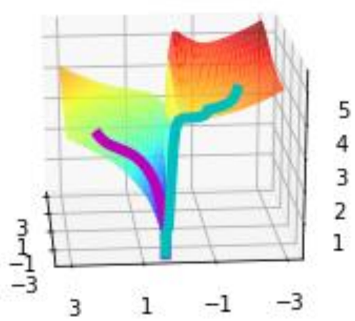
3.13 (a) $g(b,\tilde{w}) = \sum_{p=1}^{P} \left(\sigma(b+\tilde{x}_p^T\tilde{w}) - y_p\right)^2 + \lambda \|w\|_2^2$

$$\nabla_{\tilde{w}}\left(\lambda\|\tilde{w}\|^2\right) = 2\lambda\begin{pmatrix}0\\ \tilde{w}\end{pmatrix}$$

Substituting 3.11(a).

$$\nabla g(\tilde{w}) = 2\sum_{p=1}^{P}\left(\sigma(\tilde{x}_p^T\tilde{w}) - y_p\right)\sigma(\tilde{x}_p^T\tilde{w})\left(1-\sigma(\tilde{x}_p^T\tilde{w})\right)\tilde{x}_p + 2\lambda\begin{pmatrix}0\\ \tilde{w}\end{pmatrix}$$

(b)



Code here：

```
def gradient_descent(X,y,w0,lam):
    w_path = []                    # container for weights learned at each iteration
    cost_path  =  []                   #  container  for  associated  objective  values  at  each 
iteration
    w_path.append(w0)
    cost = compute_cost(w0)
    cost_path.append(cost)
    w= w0
    w1= np.array([0,0])
    w1.shape = (2,1)
    # start gradient descent loop
    max_its =20000
    alpha = 10**(-2)
    for k in range(max_its):
        # YOUR CODE GOES HERE - compute gradient
        grad=0
        for p in range(0,np.shape(X)[0]):
            grad+=2*(s(np.dot(X[p].T,w))-y[p])*s(np.dot(X[p].T,w))*(1-
s(np.dot(X[p].T,w)))*X[p]
        grad=np.asarray(grad)
        grad.shape=(2,1)
        w1[1]=w[1]
        grad=grad+0.05*w1

        # take gradient step
        w = w - alpha*grad

        # update path containers
        w_path.append(w)
        cost = compute_cost(w)
        cost_path.append(cost)

    # reshape containers for use in plotting in 3d
    w_path = np.asarray(w_path)
    w_path.shape = (np.shape(w_path)[0],2)

    cost_path = np.asarray(cost_path)
    cost_path.shape = (np.size(cost_path),1)

    return w_path,cost_path
```