HW2

Liu Cao 3072379

Exercise 3.1 Fitting a regression line to the student debt data

Linear regression model:

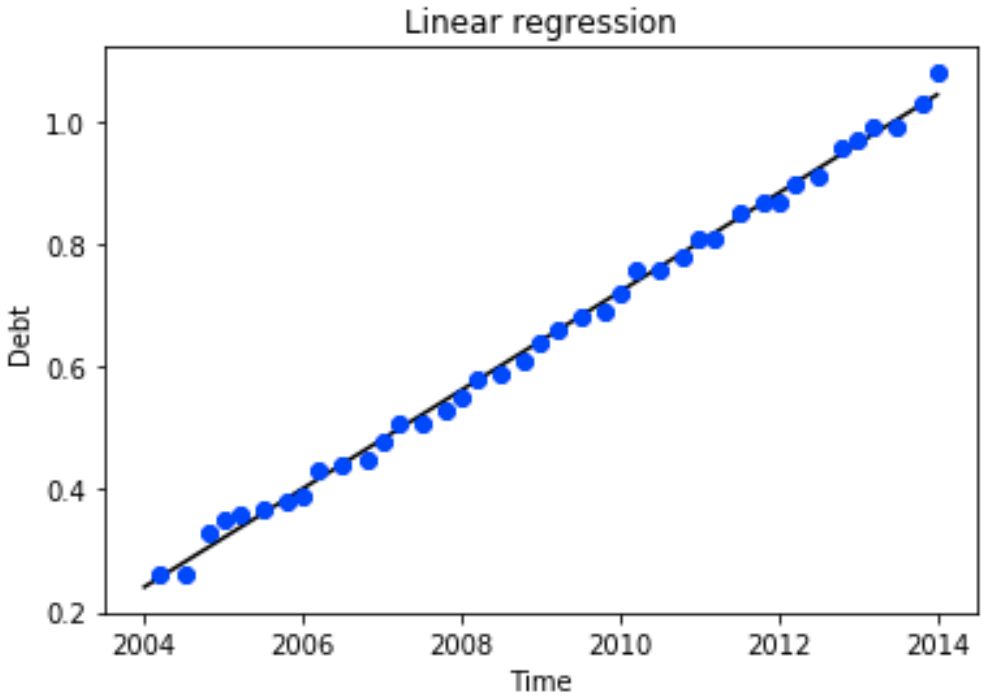


Fig.1 Linear regression from 2004 to 2014

Where w is 8.03244175e-02 and b is -1.60729045e+02.

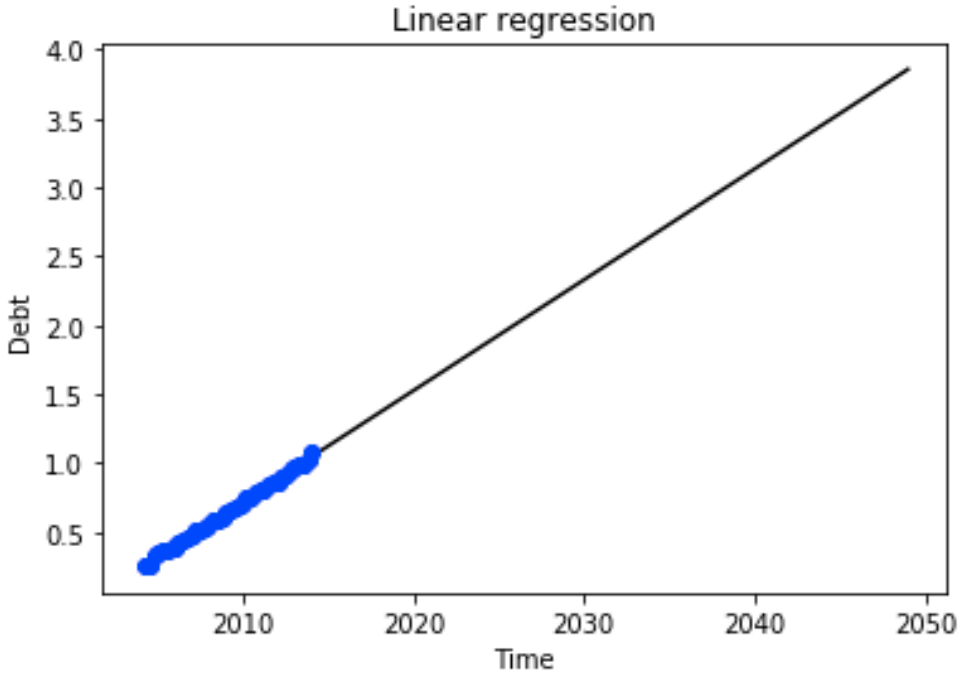


Fig.2 Linear regression from 2004 to 2050

If this linear trend continues, the student loan debt will be **3.93601056** in 2050.

Complete Code

```
import csv
from pylab import *

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

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('Time')
    plt.title('Linear regression')
```

```
    g = weight[1] * it + weight[0]
    plt.plot(it, g, 'k')
    plt.plot(X_temp, Y, 'bo')
    plt.show()
    print (weight[1] * 2050 + weight[0])
main()
```

Exercise 3.3 The least square cost for linear regression is convex

a) $\because g(\tilde{w}) = \sum_{p=1}^{P} (\tilde{X}_p^T \tilde{w} - y_p)^2$

$= \sum_{p=1}^{P} [(\tilde{X}_p^T \tilde{w})^T (\tilde{X}_p^T \tilde{w}) - 2y_p (\tilde{X}_p^T \tilde{w}) + y_p^2]$

$= \sum_{p=1}^{P} [\tilde{w}^T (\tilde{X}_p \tilde{X}_p^T) \tilde{w} - 2y_p (\tilde{X}_p^T \tilde{w}) + y_p^2]$

$= \frac{1}{2} \tilde{w}^T (\sum_{p=1}^{P} 2\tilde{X}_p \tilde{X}_p^T) \tilde{w} - (\sum_{p=1}^{P} 2y_p \tilde{X}_p^T) \tilde{w} + \sum_{p=1}^{P} y_p^2$

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

where $\bar{Q} = \sum_{p=1}^{P} 2\tilde{X}_p \tilde{X}_p^T$ $\quad \bar{r}^T = -\sum_{p=1}^{P} 2y_p \tilde{X}_p^T$ $\quad d = \sum_{p=1}^{P} y_p^2$

b) Suppose if $\tilde{X}_p \tilde{X}_p^T$ has negative eigenvalue a which makes the Eigenvector $\bar{v}$:

$\tilde{X}_p \tilde{X}_p^T \bar{v} = a\bar{v}$

$\therefore \bar{v}^T \tilde{X}_p \tilde{X}_p^T \bar{v} = \bar{v}^T a \bar{v}$

$\because \bar{v}^T a \bar{v} = a \bar{v}^T \bar{v} = a \|\bar{v}\|^2 < 0$ ($\bar{v}$ is non-zero Eigenvector $\bar{v}$)

$\bar{v}^T \tilde{X}_p \tilde{X}_p^T \bar{v} = (\tilde{X}_p^T \bar{v})^T \cdot (\tilde{X}_p^T \bar{v}) = (\tilde{X}_p^T \bar{v})^2 >= 0$

$\therefore \tilde{X}_p \tilde{X}_p^T \bar{v} \neq a\bar{v}$

$\therefore \tilde{X}_p \tilde{X}_p^T$ must have all non-negative eigenvalues.

$\because \bar{Q} = \sum_{p=1}^{P} 2\tilde{X}_p \tilde{X}_p^T$, which is sum of $\tilde{X}_p \tilde{X}_p^T$

$\therefore Q$ has all nonnegtive eigenvalues.

c) $\because \nabla g(\tilde{w}) = \frac{1}{2} (\bar{Q} + \bar{Q}^T) \tilde{w} + \bar{r}$

$\nabla^2 g(\tilde{w}) = \frac{1}{2} (\bar{Q} + \bar{Q}^T)$

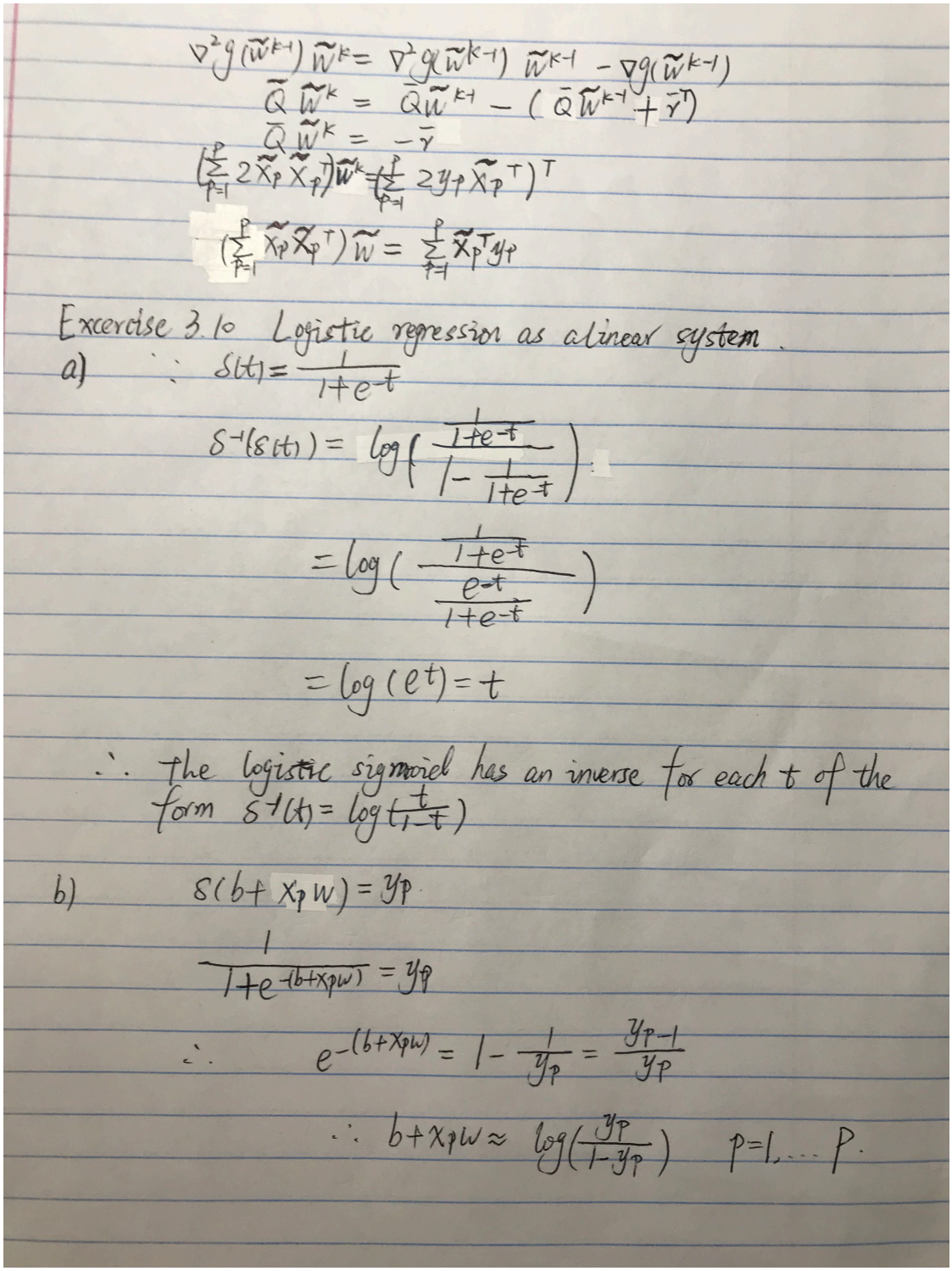
$\because \bar{Q} = \sum_{p=1}^{P} 2\tilde{X}_p \tilde{X}_p^T$

$\therefore \bar{Q}$ is a symmetric matrix.

$\therefore \nabla^2 g(\tilde{w}) = Q$ $\therefore \nabla^2 g(\tilde{w})$ has all nonnegtive eigenvalues.

$\therefore g(\tilde{w})$ is a convex function.

d) $\because$ According to Newton step.



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

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

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

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

$$\left(\sum_{p=1}^{P} \tilde{X}_p \tilde{X}_p^T\right)\tilde{w} = \sum_{p=1}^{P} \tilde{X}_p^T y_p$$

Excercise 3.10 Logistic regression as a linear system.

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

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

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

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

$\therefore$ the logistic sigmoid has an inverse for each t of the form $\sigma^{-1}(t) = \log\left(\frac{t}{1-t}\right)$

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

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

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

$$\therefore\ b + x_p w \approx \log\left(\frac{y_p}{1-y_p}\right) \quad p = 1, \ldots P.$$

Exercise 3.10

c)

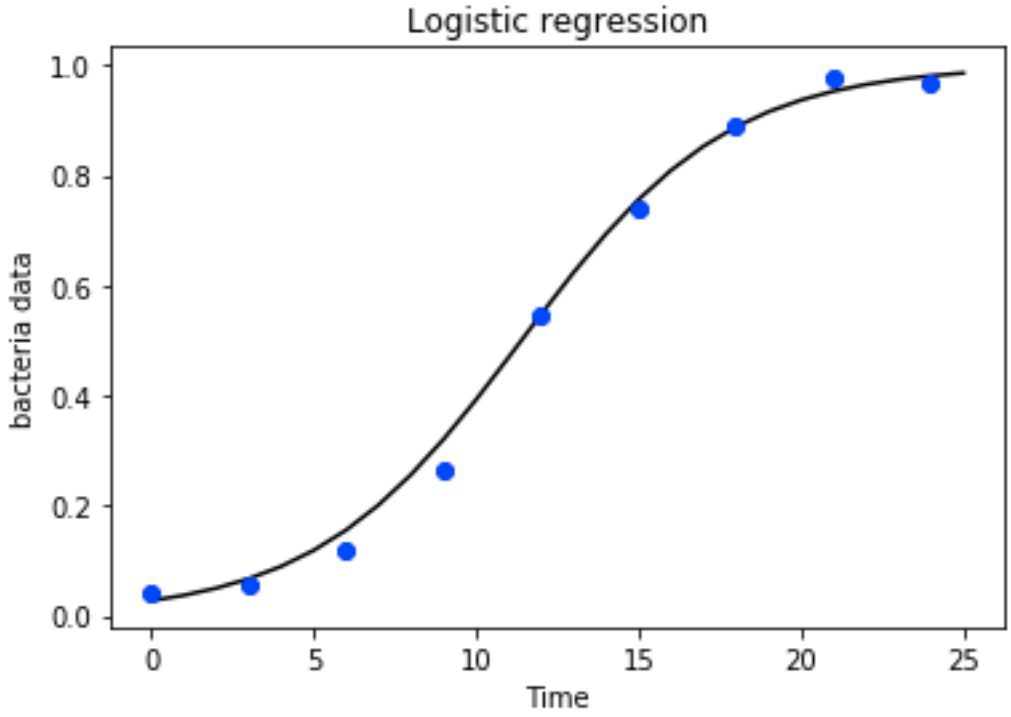


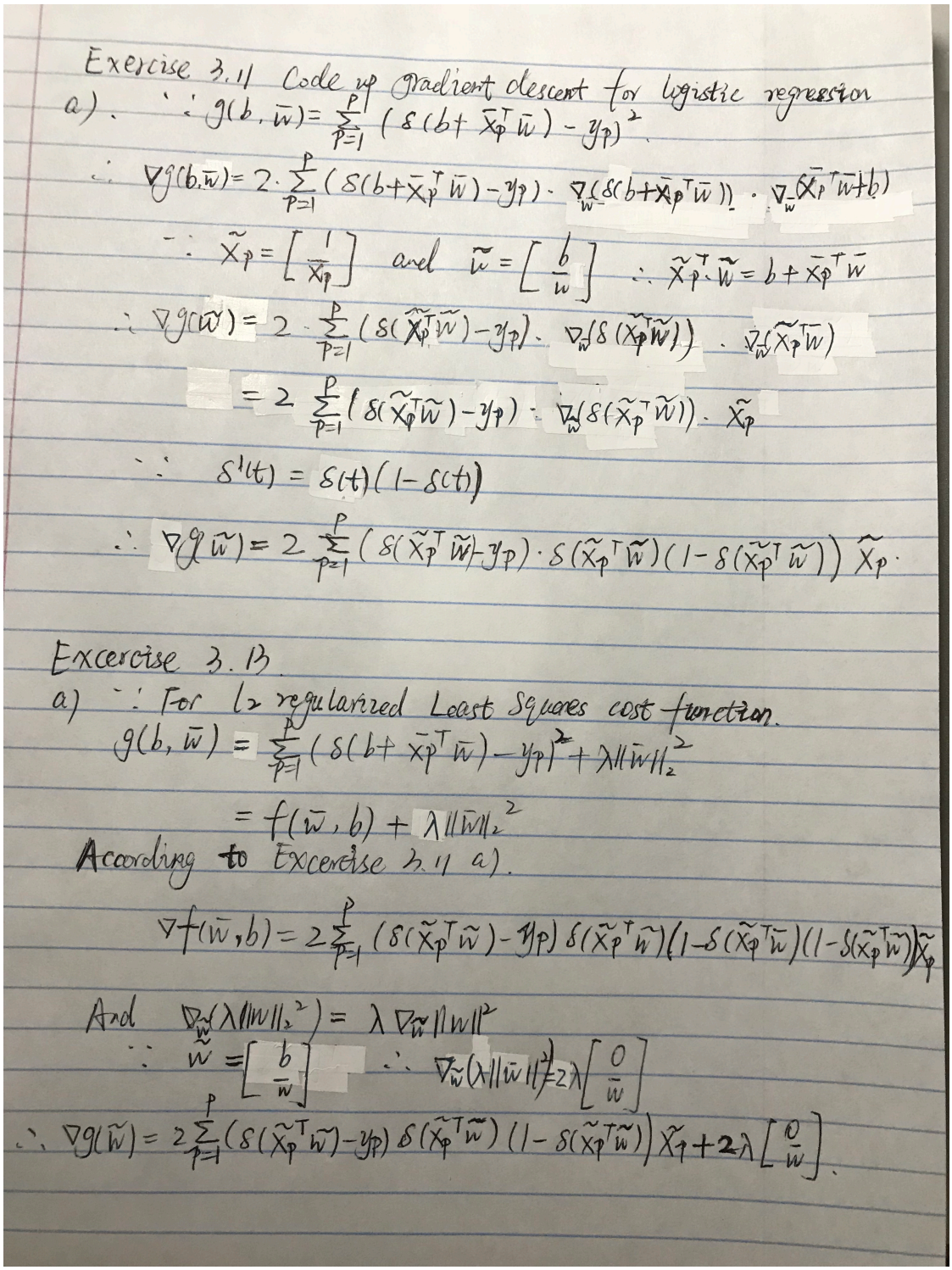
Fig.3 bacteria from 0 to 24 hours

Where w is 0.31386916, and b is -3.57527391.

Complete code

```
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, 'bo')
    plt.show()
main()
```



Exercise 3.11 Code up gradient descent for logistic regression

b)

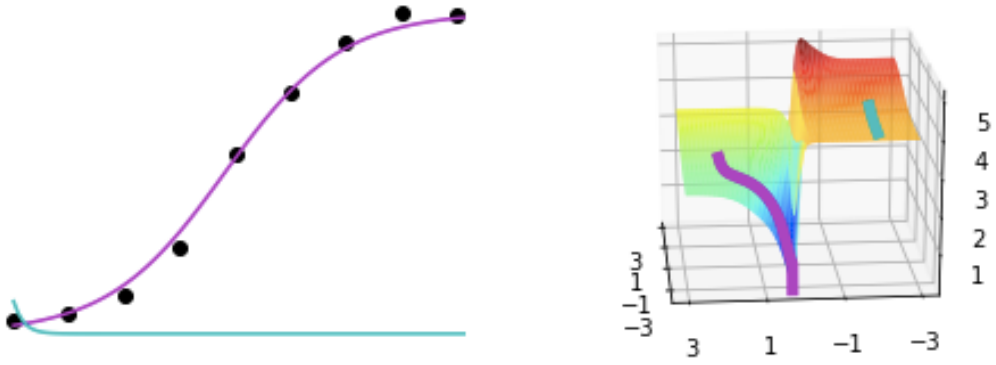


Fig.4 Sigmoidal fits and surface of cost function

Complete Code

```
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
import csv
import math

# load the data
def load_data(csvname):
    data = np.matrix(np.genfromtxt(csvname, delimiter=','))
    x = np.asarray(data[:,0])
    temp = np.ones((np.size(x),1))
    X = np.concatenate((temp,x),1)
    y = np.asarray(data[:,1])
    y = y/y.max()
    return X,y

def product(x):
    d = 1/(1 + my_exp(-x))
    return d

# run gradient descent
def gradient_descent(X,y,w0):
```

```
    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

    # start gradient descent loop
    max_its = 5000
    alpha = 10**(-2)
    for k in range(max_its):
        # YOUR CODE GOES HERE - compute gradient
        grad=0
        for n in range(0,np.shape(X)[0]):
            grad=grad+2*(product(np.dot(X[n].T,w))-y[n])*product(np.dot(X[n].T,w))*(1-
product(np.dot(X[n].T,w)))*X[n]
        grad=np.asarray(grad)
        grad.shape=(2,1)
        # 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

# calculate the cost value for a given input weight w
def compute_cost(w):
        temp = 1/(1 + my_exp(-np.dot(X,w))) - y
        temp = np.dot(temp.T,temp)
        return temp

# avoid overflow when using exp - just cutoff after arguments get too large/small
def my_exp(u):
        s = np.argwhere(u > 100)
        t = np.argwhere(u < -100)
        u[s] = 0
        u[t] = 0
        u = np.exp(u)
        u[t] = 1
        return u

# used by plot_logistic_surface to make objective surface of logistic regression cost function
def add_layer(a,b,c):
        a.shape = (2,1)
        b.shape = (1,1)
        z = my_exp(-np.dot(c,a))
        z = 1/(1 + z) - b
        z = z**2
        return z

# plot fit to data and corresponding gradient descent path onto the logistic regression objective surface
def show_fit(w_path,ax,col):
        # plot solution of gradient descent fit to original data
        s = np.linspace(0,25,100)
```

```
        t = 1/(1 + my_exp(-(w_path[-1,0] + w_path[-1,1]*s)))
        ax.plot(s,t,color = col)

# plot gradient descent paths on cost surface
def show_paths(w_path,cost_path,ax,col):
        # plot grad descent path onto surface
        ax.plot(w_path[:,0],w_path[:,1],cost_path[:,0],color = col,linewidth = 5)      # add a little to
output path so its visible on top of the surface plot

# plot logistic regression surface
def plot_surface(ax):
        # plot logistic regression surface
        r = np.linspace(-3,3,150)
        s,t = np.meshgrid(r, r)
        s = np.reshape(s,(np.size(s),1))
        t = np.reshape(t,(np.size(t),1))
        h = np.concatenate((s,t),1)

        # build 3d surface
        surf = np.zeros((np.size(s),1))
        max_its = np.size(y)
        for i in range(0,max_its):
                surf = surf + add_layer(X[i,:],y[i],h)

        s = np.reshape(s,(int(math.sqrt(np.size(s))),int(math.sqrt(np.size(s)))))
        t = np.reshape(t,(int(math.sqrt(np.size(t))),int(math.sqrt(np.size(t)))))
        surf = np.reshape(surf,(int(math.sqrt(np.size(surf))),int(math.sqrt(np.size(surf)))))

        # plot 3d surface
        ax.plot_surface(s,t,surf,cmap = 'jet')
        ax.azim = 175
        ax.elev = 20
```

```
# plot points
def plot_points(X,y,ax):
    ax.plot(X[:,1],y,'ko')

# load dataset
X,y = load_data('bacteria_data.csv') # load in data



# initialize figure, plot data, and dress up panels with axes labels etc.,
fig = plt.figure(facecolor = 'white',figsize = (8,3))
ax1 = fig.add_subplot(121)
ax1.set_xlim(min(X[:,1])-0.5, max(X[:,1])+0.5)
ax1.set_ylim(min(y)-0.1,max(y)+0.1)
ax1.axis('off')

ax2 = fig.add_subplot(122, projection='3d')
ax2.xaxis.set_rotate_label(False)
ax2.yaxis.set_rotate_label(False)
ax2.zaxis.set_rotate_label(False)
ax2.get_xaxis().set_ticks([-3,-1,1,3])
ax2.get_yaxis().set_ticks([-3,-1,1,3])
# ax2.axis('off')

### run gradient descent with first initial point
w0 = np.array([0,2])
w0.shape = (2,1)
w_path, cost_path = gradient_descent(X,y,w0)

# plot points
plot_points(X,y,ax1)

# plot fit to data and path on objective surface
show_fit(w_path,ax1,'m')
```

```
show_paths(w_path,cost_path,ax2,'m')

### run gradient descent with first initial point
w0 = np.array([0,-2])
w0.shape = (2,1)
w_path, cost_path = gradient_descent(X,y,w0)

# plot fit to data and path on objective surface
show_fit(w_path,ax1,'c')
show_paths(w_path,cost_path,ax2,'c')
plot_surface(ax2)
plt.show()
```

Exercise 3.13 Code up gradient for l_2 regularized logistic regression

b)

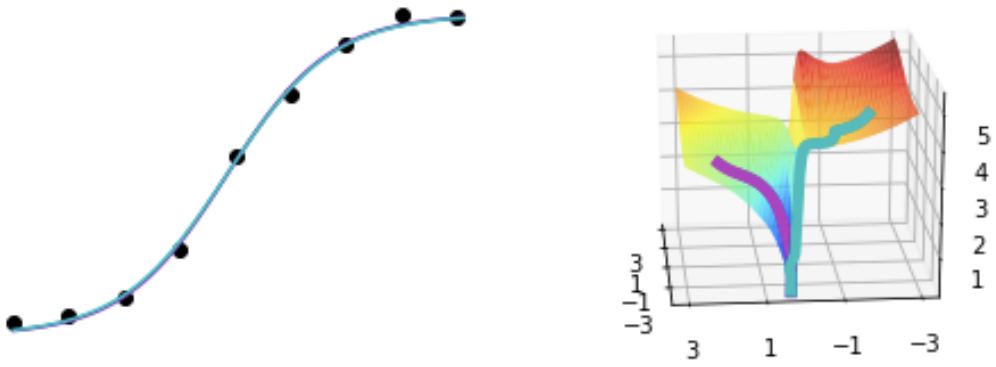


Fig.5 Sigmoidal fits and their surface of cost function

Complete Code

```
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
import csv
import math
```

```
# load the data
def load_data(csvname):
    data = np.matrix(np.genfromtxt(csvname, delimiter=','))
    x = np.asarray(data[:,0])
    temp = np.ones((np.size(x),1))
    X = np.concatenate((temp,x),1)
    y = np.asarray(data[:,1])
    y = y/y.max()
    return X,y


def product(x):
    d = 1/(1 + my_exp(-x))
    return d


# run gradient descent
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 n in range(0,np.shape(X)[0]):
            grad=grad+2*(product(np.dot(X[n].T,w))-y[n])*product(np.dot(X[n].T,w))*(1-
```

```
product(np.dot(X[n].T,w)))*X[n]
            grad=np.asarray(grad)
            grad.shape=(2,1)
            w1[1]=w[1]
            grad=grad+0.1*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

# calculate the cost value for a given input weight w
def compute_cost(w):
    temp = 1/(1 + my_exp(-np.dot(X,w))) - y
    temp = np.dot(temp.T,temp)
    return temp

# avoid overflow when using exp - just cutoff after arguments get too large/small
def my_exp(u):
    s = np.argwhere(u > 100)
    t = np.argwhere(u < -100)
```

```
        u[s] = 0
        u[t] = 0
        u = np.exp(u)
        u[t] = 1
        return u

# used by plot_logistic_surface to make objective surface of logistic regression cost function
def add_layer(a,b,c):
        a.shape = (2,1)
        b.shape = (1,1)
        z = my_exp(-np.dot(c,a))
        z = 1/(1 + z) - b
        z = z**2
        return z

# plot fit to data and corresponding gradient descent path onto the logistic regression objective surface
def show_fit(w_path,ax,col):
        # plot solution of gradient descent fit to original data
        s = np.linspace(0,25,100)
        t = 1/(1 + my_exp(-(w_path[-1,0] + w_path[-1,1]*s)))
        ax.plot(s,t,color = col)

# plot gradient descent paths on cost surface
def show_paths(w_path,cost_path,ax,col):
        # plot grad descent path onto surface
        ax.plot(w_path[:,0],w_path[:,1],cost_path[:,0],color = col,linewidth = 5)      # add a little to output path so its visible on top of the surface plot

# plot logistic regression surface
def plot_surface(ax,lam):
        # plot logistic regression surface
        r = np.linspace(-3,3,150)
```

```
        s,t = np.meshgrid(r, r)
        s = np.reshape(s,(np.size(s),1))
        t = np.reshape(t,(np.size(t),1))
        h = np.concatenate((s,t),1)

        # build 3d surface
        surf = np.zeros((np.size(s),1))
        max_its = np.size(y)
        for i in range(0,max_its):
            surf = surf + add_layer(X[i,:],y[i],h)
        surf = surf + lam*t**2

        s = np.reshape(s,(int(math.sqrt(np.size(s))),int(math.sqrt(np.size(s)))))
        t = np.reshape(t,(int(math.sqrt(np.size(t))),int(math.sqrt(np.size(t)))))
        surf = np.reshape(surf,(int(math.sqrt(np.size(surf))),int(math.sqrt(np.size(surf)))))

        # plot 3d surface
        ax.plot_surface(s,t,surf,cmap = 'jet')
        ax.azim = 175
        ax.elev = 20

# plot points
def plot_points(X,y,ax):
        ax.plot(X[:,1],y,'ko')

# load dataset
X,y = load_data('bacteria_data.csv') # load in data

# initialize figure, plot data, and dress up panels with axes labels etc.,
fig = plt.figure(facecolor = 'white',figsize = (8,3))
ax1 = fig.add_subplot(121)
ax1.set_xlim(min(X[:,1])-0.5, max(X[:,1])+0.5)
ax1.set_ylim(min(y)-0.1,max(y)+0.1)
```

```
ax1.axis('off')

ax2 = fig.add_subplot(122, projection='3d')
ax2.xaxis.set_rotate_label(False)
ax2.yaxis.set_rotate_label(False)
ax2.zaxis.set_rotate_label(False)
ax2.get_xaxis().set_ticks([-3,-1,1,3])
ax2.get_yaxis().set_ticks([-3,-1,1,3])
# ax2.axis('off')

# define regularizer parameter
lam = 10**-1

### run gradient descent with first initial point
w0 = np.array([0,2])
w0.shape = (2,1)
w_path, cost_path = gradient_descent(X,y,w0,lam)
# plot points
plot_points(X,y,ax1)
# plot fit to data and path on objective surface
show_fit(w_path,ax1,'m')
show_paths(w_path,cost_path,ax2,'m')

### run gradient descent with first initial point
w0 = np.array([0,-2])
w0.shape = (2,1)
w_path, cost_path = gradient_descent(X,y,w0,lam)

# plot fit to data and path on objective surface
show_fit(w_path,ax1,'c')
show_paths(w_path,cost_path,ax2,'c')
plot_surface(ax2,lam)
plt.show()
```