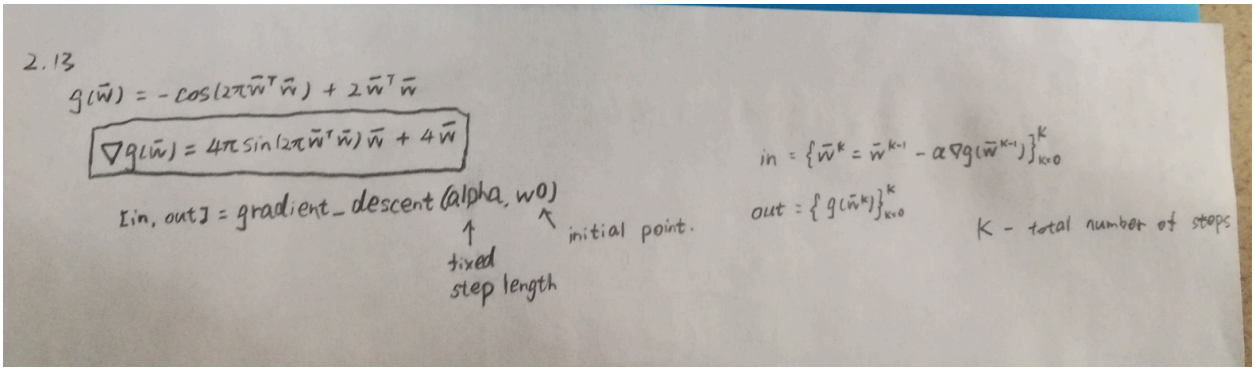
HW2

2.13

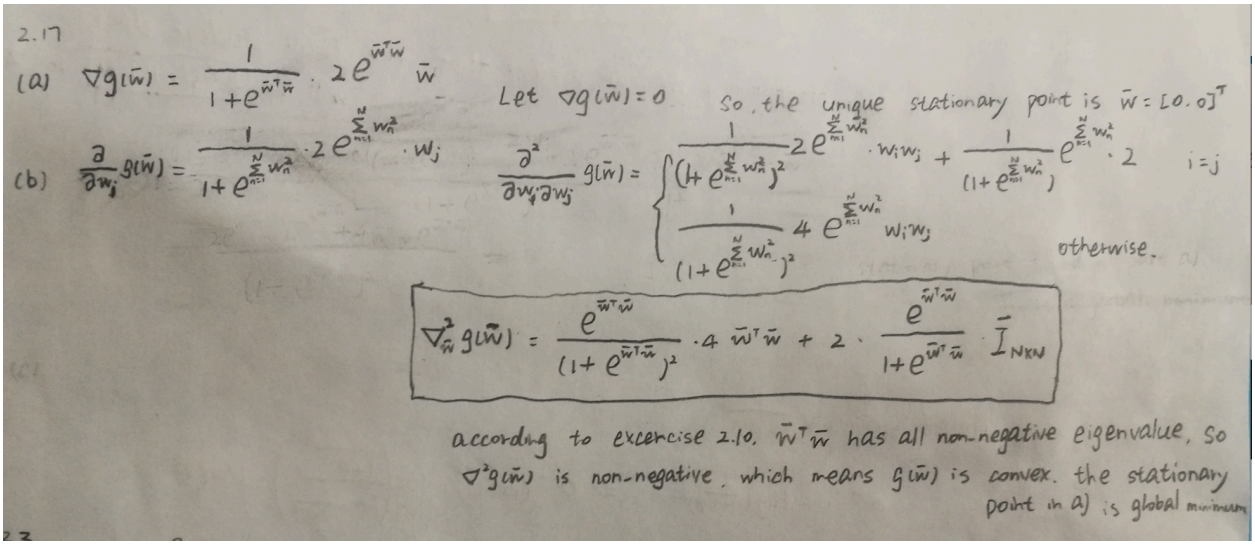


code:

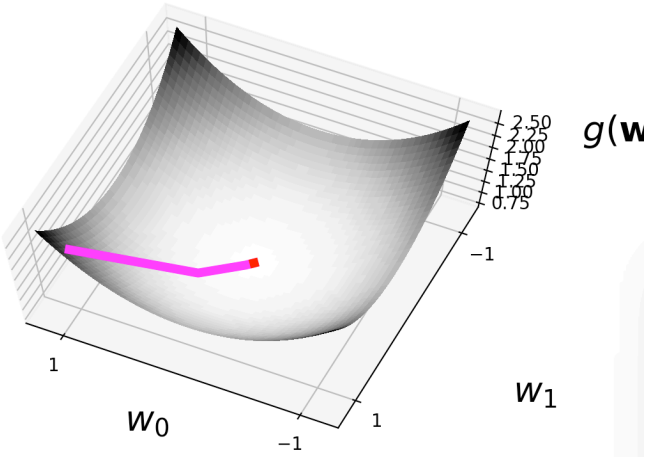
```
    max_its = 50
    while linalg.norm(grad) > 10**(-5) and iter <= max_its:
        # take gradient step
        grad = 2*w + 4*pi*sin(2*pi*dot(w.T,w))*w    # 2.13
```

Whole program please see file two_d_grad_wrapper_hw.py

2.17

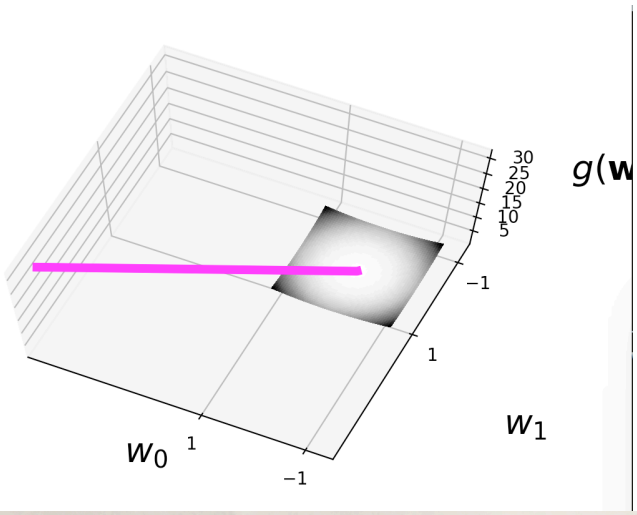


(c)




```
###### ML Algorithm functions ######
def gradient_descent(w0):
    w = w0
    g_path = []
    w_path = []
    w_path.append(w)
    #g_path.append(-cos(2*pi*dot(w.T,w)) + 2*dot(w.T,w))       # 2.13
    g_path.append(np.log(1 + np.e ** dot(w.T,w)))
    # start gradient descent loop
    grad_1 = 1
    grad_2 = 1
    iter = 1
    max_its = 10
    while iter <= max_its:
        # take gradient step
        #grad = 2*w + 4*pi*sin(2*pi*dot(w.T,w))*w   # 2.13
        I = np.eye(2)
        grad_1 = (1 / (1 + np.e ** dot(w.T,w))) * 2 * (np.e ** (dot(w.T,w))) * w
        grad_2 = 4*(np.e**dot(w.T,w))*dot(w,w.T) / (1 + np.e**dot(w.T,w))**2 + (2 * np.e**dot(w.T,w)) / (1 + np.e**dot(w.T,w)) * I
        w = w - dot(linalg.inv(grad_2),grad_1)
        print w
        # update path containers
        w_path.append(w)
        #g_path.append(-cos(2*pi*dot(w.T,w)) + 2*dot(w.T,w))    # 2.13
        g_path.append(np.log(1 + np.e ** dot(w.T,w)))           # 2.17
        iter+= 1
    g_path = asarray(g_path)
    g_path.shape = (iter,1)
    w_path = asarray(w_path)
    w_path.shape = (iter,2)
```

(d)



(d). $g(\bar{w}) = \log(1 + e^{\bar{w}^T\bar{w}})$ $\quad \bar{w} = \begin{bmatrix} 4 \\ 4 \end{bmatrix} \Rightarrow g(\bar{w}) \approx \bar{w}^T\bar{w}$

$h(\bar{w}) = g(\bar{w}^0) + \nabla g(\bar{w}^0)^T(\bar{w} - \bar{w}^0) + \frac{1}{2}(\bar{w} - \bar{w}^0)^T \nabla^2 g(\bar{w}^0)(\bar{w} - \bar{w}^0)$

$= \bar{w}^{0T}\bar{w}^0 + 2\bar{w}^{0T}(\bar{w} - \bar{w}^0) + \frac{1}{2}(\bar{w} - \bar{w}^0)^T \cdot 2(\bar{w} - \bar{w}_0)$

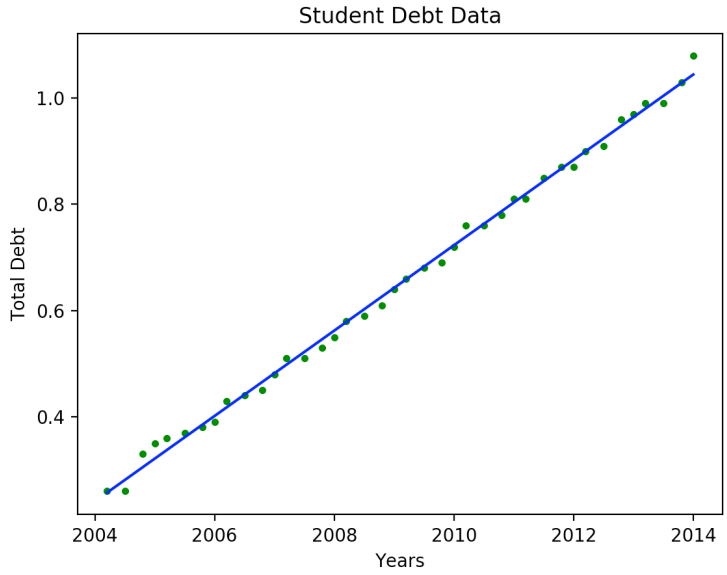
$= \bar{w}^{0T}\bar{w}^0 + 2\bar{w}^{0T}\bar{w} - 2\bar{w}^{0T}\bar{w}^0 + \bar{w}^T\bar{w} - 2\bar{w}^{0T}\bar{w} + \bar{w}^{0T}\bar{w}^0$

$= \bar{w}^T\bar{w}$ $\quad$ Let $\nabla h(\bar{w}) = 0$ $\; 2\bar{w} = 0$ $\quad \boxed{\bar{w} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}}$ which is stationary point.

∴ Because initialization is further away, $g(\bar{w}) = \log(e^{\bar{w}^T\bar{w}}) = \bar{w}^T\bar{w}$, the second Taylor $g(w)$ is the minimum of $g(\bar{w})$ itself.

For (c) and (d), please see the file NewTon_Method_2_17.py in zip.

3.1



3.1. $g(\tilde{w}) = \sum_{p=1}^{P} (\tilde{x}_p^T \tilde{w} - y_p)^2$

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

Let $\nabla g(\tilde{w}) = 0$

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

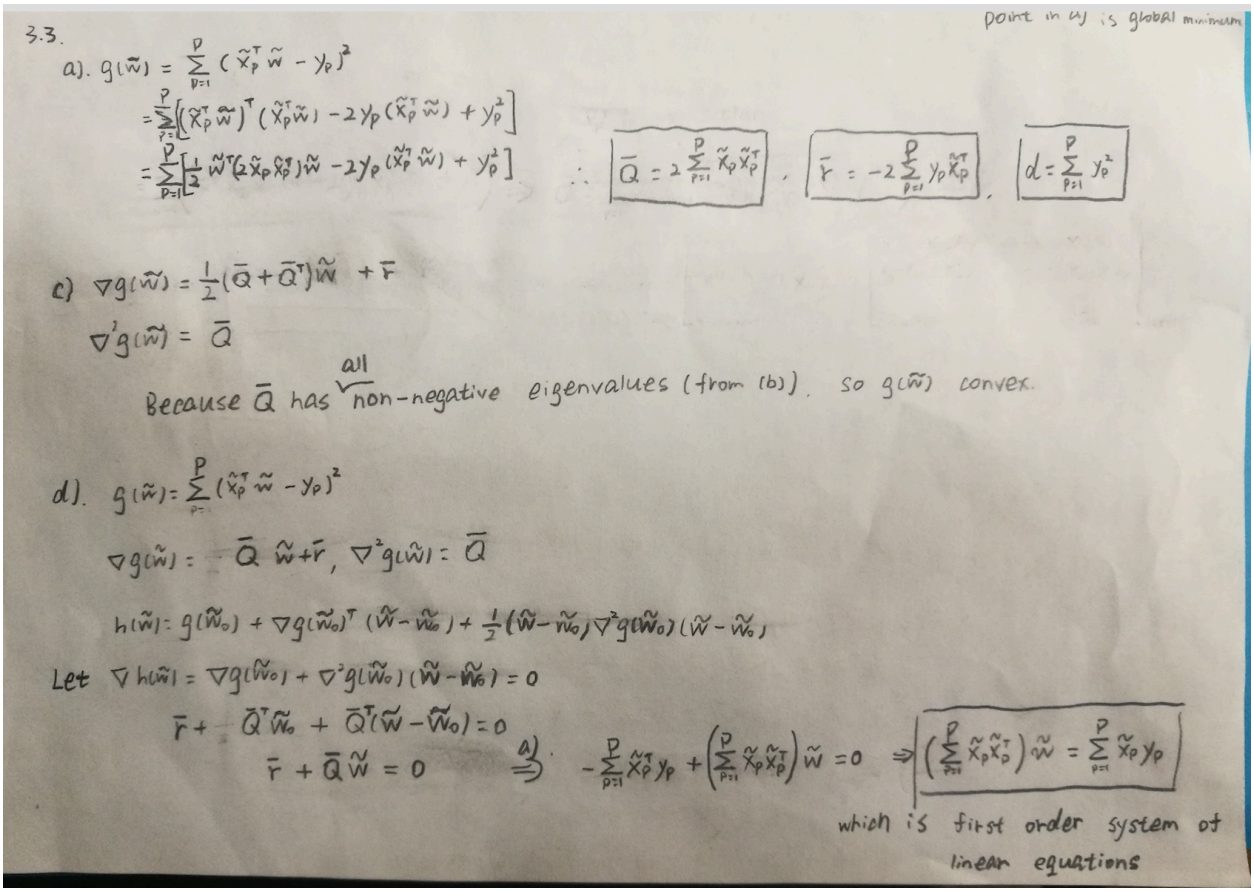
$\tilde{w}^* = \left(\sum_{p=1}^{P} \tilde{x}_p \tilde{x}_p^T\right)^{-1} \sum_{p=1}^{P} \tilde{x}_p y_p$

Through programming, we can predict that the total student debt will be 3.936011966 in 2050

```python
def read_data(filename):
    f = open(filename)
    csv_f = csv.reader(f)
    x = []
    y = []
    for row in csv_f:
        temp = array([1, float(row[0])])
        temp.shape = (2, 1)
        x.append(temp)
        y.append(float(row[1]))
    return x, y

def basic_linear_regression(x,y):
    A = array([0.,0.,0.,0.])
    A.shape = (2,2)
    for i in x:
        A += dot(i,i.T)
    A_inv = pinv(A)

    b = array([0.,0.])
    b.shape = (2,1)
    for i in range(len(x)):
        b += y[i] * x[i]
    w = dot(A_inv,b)
    return w
def main():
    x, y = read_data('student_debt.csv')                # plot objective function
    w = basic_linear_regression(x,y)
    _x = []
    _y = []
    _b = w[0][0]
    _w = w[1][0]
    for i in range(len(x)):
        _x.append(x[i][1])
        temp = x[i][1] * _w + _b
        _y.append(temp)
    print(w)
    predict = _b + _w * 2050
    print(predict)

    plt.plot(_x,y,'g.')
    plt.plot(_x,_y,'b')
    plt.title('Student Debt Data')
    plt.xlabel('Years')
    plt.ylabel('Total Debt')
    show()
main()
```

For 3.1, please see the file exercise_regression_line_3_1.py in zip.

3.3



3.3.

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

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

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

$\therefore \boxed{\bar{Q} = 2\sum_{p=1}^{P} \tilde{x}_p \tilde{x}_p^T}, \quad \boxed{\bar{r} = -2\sum_{p=1}^{P} y_p \tilde{x}_p^T}, \quad \boxed{d = \sum_{p=1}^{P} y_p^2}$

point in a) is global minimum

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

$\nabla^2 g(\tilde{w}) = \bar{Q}$

Because $\bar{Q}$ has all non-negative eigenvalues (from (b)), so $g(\tilde{w})$ convex.

d). $g(\tilde{w}) = \sum_{p=1}^{P} (\tilde{x}_p^T \tilde{w} - y_p)^2$

$\nabla g(\tilde{w}) = \bar{Q}\tilde{w} + \bar{r}, \quad \nabla^2 g(\tilde{w}) = \bar{Q}$

$h(\tilde{w}) = g(\tilde{w}_0) + \nabla g(\tilde{w}_0)^T (\tilde{w} - \tilde{w}_0) + \frac{1}{2}(\tilde{w} - \tilde{w}_0)^T \nabla^2 g(\tilde{w}_0)(\tilde{w} - \tilde{w}_0)$

Let $\nabla h(\tilde{w}) = \nabla g(\tilde{w}_0) + \nabla^2 g(\tilde{w}_0)(\tilde{w} - \tilde{w}_0) = 0$

$\bar{r} + \bar{Q}^T \tilde{w}_0 + \bar{Q}^T(\tilde{w} - \tilde{w}_0) = 0$

$\bar{r} + \bar{Q}\tilde{w} = 0 \overset{a)}{\Rightarrow} -\sum_{p=1}^{P} \tilde{x}_p^T y_p + \left(\sum_{p=1}^{P} \tilde{x}_p \tilde{x}_p^T\right)\tilde{w} = 0 \Rightarrow \boxed{\left(\sum_{p=1}^{P} \tilde{x}_p \tilde{x}_p^T\right)\tilde{w} = \sum_{p=1}^{P} \tilde{x}_p y_p}$

which is first order system of linear equations

3.10

3.10

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

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

Therefore, that indeed $\sigma^{-1}(\sigma(t)) = t$ for all such values of t.

(b).

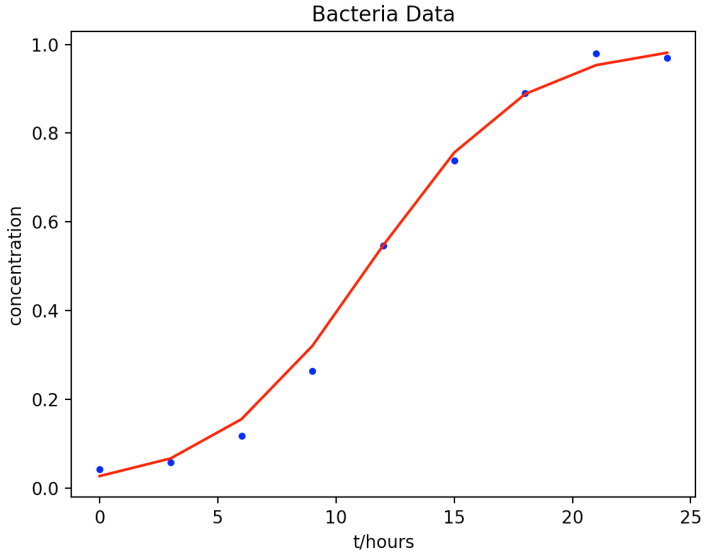
for 3.24 $\sigma(b+x_p w) \approx y_p$, $p = 1, \ldots, P$.

then, we could obtain

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

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

(c)



```
def read_data(filename):
    f = open(filename)
    csv_f = csv.reader(f)
    x = []
    y = []
    for row in csv_f:
        temp = array([1, float(row[0])])
        temp.shape = (2, 1)
        x.append(temp)
        y.append(float(row[1]))
    print(y)
    return x, y

def basic_linear_regression(x,y):
    A = array([0.,0.,0.,0.])
    A.shape = (2,2)
    for i in x:
        A += dot(i,i.T)
    A_inv = pinv(A)

    b = array([0.,0.])
    b.shape = (2,1)
    for i in range(len(x)):
        b += y[i] * x[i]
    w = dot(A_inv,b)
    return w

def main():
    x, y = read_data('bacteria_data.csv')                # plot objective function
    y_p = []
    for i in y:
        temp = np.log((i) / (1 - i))
        y_p.append(temp)
    w = basic_linear_regression(x,y_p)
    _x = []
    _y = []
    _b = w[0][0]
    _w = w[1][0]
    print(w)
    for i in range(len(x)):
        _x.append(x[i][1])
        temp = np.e**(_b + x[i][1] * _w) / (1 + np.e**(_b + x[i][1] * _w))
        _y.append(temp)
    # print(w)
    # predict = _b + _w * 2050
    # print(predict)

    plt.plot(_x,y,'b.')
    plt.plot(_x,_y,'r')
    plt.title('Bacteria Data')
    plt.xlabel('t/hours')
    plt.ylabel('concentration')
    show()
main()
```

Please see file Logistic_regression_3_10.py in zip