1.(a)

CS 229, PS #1

1. (a) From $J(\theta)$, we know the logistic loss for one set of data $y \in \mathbb{R}$, $x \in \mathbb{R}^n$

$$\varphi(y\theta^T x) = \log(1 + e^{-y\theta^T x})$$

First we want to find the ~~gradient~~ derivative of φ

$$\frac{d\varphi(z)}{dz} = \cancel{\frac{1}{1+e^{-z}}} \frac{1}{1+e^{-z}} \cdot e^{-z} \cdot (-1) = -\frac{1}{1+e^{z}}$$

Let sigmoid function $g(x) = \frac{1}{1+e^{-x}}$

$$\frac{\partial \varphi(yx^T\theta)}{\partial \theta_k} = -\frac{1}{1+e^{yx^T\theta}} \times y \times x_k = \frac{-y x_k}{1+e^{yx^T\theta}} = \boxed{-g(yx^T\theta) y x_k}$$

To find hessian, need to calculate second derivative of φ

because: $g'(x) = -(1+e^{-x})^{-2} \cdot e^{-x} \cdot -1 = \frac{e^{-x}}{(1+e^{-x})^2}$

So for any $$\frac{\partial^2 \varphi(yx^T\theta)}{\partial \theta_k \theta_l} = \boxed{y^2 x_k x_l \cdot \frac{e^{-yx^T\theta}}{(1+e^{-yx^T\theta})^2} = h_{kl}}$$ element in hessian

From the hint, we know that $(x_1 z_1 + x_2 z_2 + \cdots + x_n z_n) \cdot (x_1 z_1 + \cdots x_n z_n) = \sum_{i=1}^{n} \sum_{j=1}^{n} z_i x_i z_j x_j$
$= (x^T z)^2 \geq 0$

so $$z^T H z = \sum_{i=1}^{n} \sum_{j=1}^{n} z_i z_j h_{ij} = \sum_{i=1}^{n} \sum_{j=1}^{n} z_i z_j x_i x_j y^2 \cdot \frac{e^{-yx^T\theta}}{(1+e^{-yx^T\theta})^2}$$
$$= y^2 \frac{e^{-yx^T\theta}}{(1+e^{-yx^T\theta})^2} (x^T z)^2 \geq 0$$

There for $H \succeq 0$

1.(b) newton.m

Two sets of theta values are calculated based on different terminate conditions:

% norm(theta-threa_old): terminate when norm of change of theta is below threshold
% -25.5466
% 6.4558
% 5.3584

% thres*J: terminate when the percentage of J's change is below threshold
% -20.0245
% 5.0980
% 4.3148

```
function [theta, ll] = newton(X,y)
    % newton's method
    % rows of X are training samples
    % rows of Y are corresponding -1/1 values

    % output ll: vector of log-likelihood values at each iteration
    % ouptut theta: parameters

    [m,n] = size(X); %99 2
    max_iters = 100000;

    X = [ones(size(X,1),1), X]; % append col of ones for intercept term

    theta = ones(n+1, 1);  % initialize theta
    theta_old = zeros(n+1, 1); % make them very different

    threshold = 1e-5;
    while norm(theta - theta_old) > threshold
        disp(norm(theta - theta_old));
        hessian = hessian_of_empirical_loss(theta, X, y);
        gradient = gradient_of_empirical_loss(theta, X, y);
        theta_old = theta;
        theta = theta - inv(hessian) * gradient;
    end
% ans =
%
%   -25.5466
%     6.4558
%     5.3584
end


function val = J(X, y, theta)
```

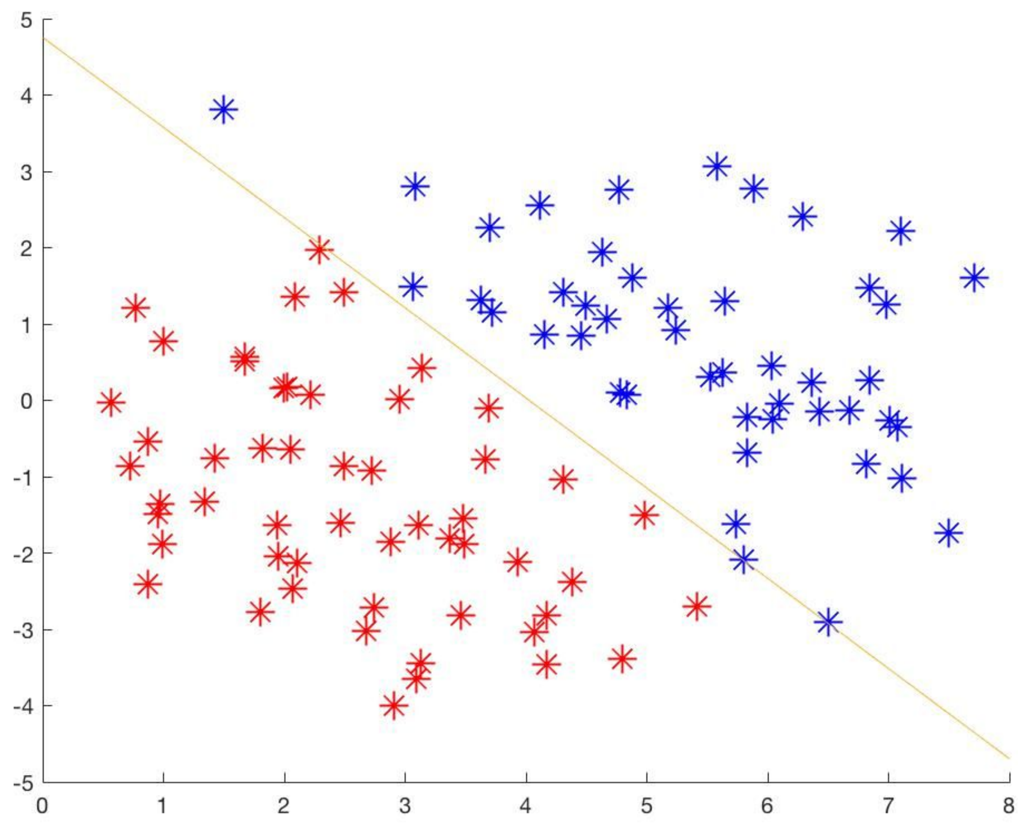
```
% calculate the empirical loss of X, y given theta
    [m, n] = size(X);
    loss = 0;
    for row = 1:m
        loss = loss + log(1+exp(-z(y(row), X(row,:), theta)));
    end
    val = loss/m;
end

function a=sigmoid(z)
    a = 1.0 ./ (1.0+exp(-z));
end

function H=hessian_of_empirical_loss(theta, X, y)
% build the hessian matrix for theta, x, y
    [m, n] = size(X);
    H = zeros(n, n);
    for hessianX = 1:n
        for hessianY = 1:n
            hessian = 0;
            for row = 1:m
                hessian = hessian + y(row)^2 * X(row, hessianX) * X(row, hessianY) *
exp(-z(y(row), X(row,:), theta));
            end
             H(hessianX, hessianY) = hessian / m;
        end
    end

end

function gi=gradient_of_empirical_loss(theta_old, X, y)
% build the gradient vector of theta, x, y
    [m, n] = size(X);
    % zeros(n) will create a square
    gi = zeros(n, 1);
    % fill the gradient one by one
    % n thetas
    for k = 1:n
        gradient = 0;
        for row = 1:m
           gradient = gradient - sigmoid(z(y(row), X(row,:), theta_old)) * y(row) * X(row, k);
        end
        gi(k) = gradient / m;
    end
end

function out=z(y, X_vector, theta_vector)
```

```
    % y is the result at current row
    % X_vector(n+1, 1) are the parameters of current row
    % theta_vector (n+1, 1) is the old theta
    out = y * X_vector * theta_vector;
end

% X = dlmread("logistic_x.txt");
% y = dlmread("logistic_y.txt");
%
% [theta, ll] = newton(X, y);
```

1.(c) plot.m

```
% norm(theta-threa_old)
%   -25.5466
%     6.4558
%     5.3584


% thres*J
% -20.0245
%     5.0980
%     4.3148

X=dlmread('logistic_x.txt');
y=dlmread("logistic_y.txt");

% disp(X);

x1 = X(:,1);
x2 = X(:,2);
% plot(X1, X2);
% scatter(X1, X2);

% norm(theta-threa_old)
% theta = [-25.5466, 6.4558, 5.3584];

% thres*J
theta = [-20.0245, 5.0980, 4.3148];
X = [ones(size(X,1),1), X];

result = X * theta';

% disp(result);
```

```
% Plot first class
scatter(x1(result > 0.5), x2(result > 0.5), 150, 'b', '*');
% Plot second class.
hold on;
scatter(x1(result < 0.5), x2(result < 0.5), 150, 'r', '*');

% a + bx1 + cx2 = 0.5
% so x2 = (0.5 - a - bx1) / c
line_x1 = (0:0.1:8);
line_x2 = (0.5 - theta(1) - theta(2) * line_x1) ./ theta(3);
plot(line_x1, line_x2);
hold off;
```



1_c

2.(a)(b)(c)

2. (a) $P(y;\lambda) = \frac{e^{-\lambda}\lambda^{y}}{y!} = \frac{1}{y!}\cdot\exp(-\lambda)\cdot\exp(y\log\lambda)$

$$= \frac{1}{y!}\cdot\exp(y\log\lambda - \lambda) \quad \text{--- ①}$$

We know exponential famimly has this form

$$P(y;\eta) = b(y)\cdot\exp(\eta\cdot T(y) - a(\eta)) \quad \text{--- ②}$$

For ①, let $\log\lambda = \eta$, $\lambda = \exp(\eta)$

and we know

$$b(y) = \frac{1}{y!},\ \eta = \log\lambda,\ T(y) = y,\ a(\eta) = \exp(\eta)$$

(b) the canonical response function maps η to $E[T(y);\eta]$

$$E[T(y);\eta] = E[y;\eta] = \lambda = \exp(\eta)$$

(c) $\ell(\theta) = \log P(y;\lambda) = \log e^{-\lambda} + \log\lambda^{y} - \log y!$

$$= -\lambda + y\log\lambda - \log y!$$

$$= -\exp(\eta) + y\log\exp(\eta) - \log y!$$

We have the assumption $\eta = \theta^{T}x$

So $\ell(\theta) = -\exp(\theta^{T}x) + y(\theta^{T}x) - \log y!$

Therefore $\frac{\partial\ell}{\partial\theta_i} = -\exp(\theta^{T}x)\cdot x_i + x_i y = x_i(y - \exp(\theta^{T}x))$

$$= x_i(y-\lambda) = x_i(y - h_\theta(x))$$

Therefore the update rule for stochastic G.D is

$$\theta_j \leftarrow \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

3.(a)

3. (a)

$$p(y=1|x) = \frac{p(x|y=1)\cdot\phi}{p(x|y=1)\cdot p(y=1) + p(x|y=-1)\cdot p(y=-1)}$$

$$= \frac{1}{1+\frac{p(x|y=-1)}{p(x|y=1)}\cdot\frac{(1-\phi)}{\phi}}$$

Similarly

$$p(y=-1|x) = \frac{1}{1+\frac{p(x|y=1)}{p(x|y=-1)}\cdot\frac{\phi}{(1-\phi)}}$$

So we can write

$$p(y|x;\phi,\Sigma,\mu_{-1},\mu_1) = \frac{1}{1+\left(\frac{p(x|y=-1)}{p(x|y=1)}\cdot\frac{(1-\phi)}{\phi}\right)^{y}} \quad --- \text{ (A)}$$

Now let $F(A,B) = A^T\Sigma^{-1}B$, then we know $F(x-\mu, x-\mu) = F(x,x) - 2F(x,\mu) + F(\mu,\mu)$

So

$$\frac{p(x|y=-1)}{p(x|y=1)} = \frac{\exp(-\frac{1}{2}F(x-\mu_{-1}))}{\exp(-\frac{1}{2}F(x-\mu_1))} = \exp\left(\frac{1}{2}\left(F(x-\mu_1) - F(x-\mu_{-1})\right)\right)$$

$$= \exp\Big(\frac{1}{2}\cdot\big(\underbrace{2F(x,\mu_{-1}) - 2F(x,\mu_1)}_{\text{calculated from } x,\ \mu_{-1},\ \mu_1,\ \Sigma} + \underbrace{F(\mu_1,\mu_1) - F(\mu_{-1},\mu_{-1})}_{\text{calculated from } \mu_{-1},\ \mu_1,\ \Sigma}\big)\Big)$$

$$= \exp(-(\theta^T x + \theta_0))$$

So (A)

$$= \frac{1}{1+\exp\left(-(\theta^T x+\theta_0)\cdot y + \left(\log\left(\frac{1-\phi}{\phi}\right)\cdot y\right)\right)}$$

$$= \frac{1}{1+\exp(-y(\theta^T x+\theta_0'))}$$

where $\theta_0' = \theta_0 - \log\frac{(1-\phi)}{\phi}$

3.(b)

3(b). Let $p(y;\phi) = \phi^{(\frac{1+y}{2})}\cdot(1-\phi)^{(\frac{1-y}{2})}$, since we're considering $|\Sigma| = \sigma^2$

Then $\ell(\phi,\mu_{-1},\mu_1,\Sigma) = \log\prod_{i=1}^{m} p(x^{(i)}|y^{(i)};\mu_{-1};\mu_1,\Sigma)\,p(y^{(i)};\phi)$

$$= \sum_{i=1}^{m}\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\cdot\exp\left(-\frac{1}{2}\frac{(x-\mu)}{\sigma^2}\right)\cdot\phi^{(\frac{1+y}{2})}\cdot(1-\phi)^{(\frac{1-y}{2})}\right)$$

$$\frac{\partial\ell}{\partial\phi} = \sum_{i=1}^{m}\left(\frac{1+y}{2}\cdot\frac{1}{\phi} - \frac{1-y}{2}\cdot\frac{1}{1-\phi}\right) \quad \text{— ①}$$

Let $a = \sum_{i=1}^{m} 1\{y^{(i)}=1\}$ --- ☆

then ① =

$$\frac{1}{\phi}\left(a\cdot\frac{1+1}{2} + (m-a)\cdot 0\right) - \frac{1}{1-\phi}\left(a\cdot\frac{1-1}{2} + (m-a)\cdot\frac{1+1}{2}\right)$$

Let $\boxed{\frac{\partial\ell}{\partial\phi}=0}$, then we have

$$\frac{a}{\phi} = \frac{m-a}{1-\phi} \Rightarrow \boxed{\phi = \frac{a}{m} = \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)}=1\}}$$

to calculate $\frac{\partial\ell}{\partial\mu}$, write ℓ as follows:

$$\ell = \sum_{i=1}^{m}\log\frac{1}{\sqrt{2\pi\sigma^2}}\cdot\exp\left(-\frac{1}{2}\left(x_i - \frac{(1-y)}{2}\mu_{-1} - \frac{(1+y)}{2}\mu_1\right)\right)^2\cdot\Sigma^{-1} \quad (\Sigma^{-1}\in\mathbb{R})$$

$$\frac{\partial\ell}{\partial\mu_1} = \sum_{i=1}^{m} -2\cdot\frac{1}{2}\left(x_i - \frac{(1-y)}{2}\mu_{-1} - \frac{(1+y)}{2}\mu_1\right)\cdot\left(-\frac{y+1}{2}\right)$$

let $\boxed{\frac{\partial\ell}{\partial\mu_1}=0}$ then $\sum_{i=1}^{m} x\cdot(y+1) - \underbrace{\sum_{i=1}^{m}\mu_{-1}\cdot\frac{(1-y^2)}{2}}_{=0} - \sum_{i=1}^{m}\mu_1\frac{(1+y)^2}{2} = 0$

$2\cdot\sum_{i=1}^{m} 1\{y^{(i)}=1\}\cdot x_i = \frac{1}{2}\cdot 4a\mu_1$ → a defined in ☆

$\boxed{\mu_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)}=1\}x_i}{\sum_{i=1}^{m} 1\{y^{(i)}=1\}}}$

We can prove μ_{-1} simililarly

3(b). continue.

$$\ell = \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi\Sigma}} \cdot \exp\left(-\frac{1}{2}(x-\mu_y)^2 \cdot \Sigma^{-1}\right)$$

$$\frac{\partial \ell}{\partial \Sigma} = m \cdot \sqrt{2\pi\Sigma} \cdot \frac{1}{\sqrt{2\pi}} \cdot \left(-\frac{1}{2}\right) \cdot \Sigma^{-\frac{3}{2}} + \frac{1}{2}\sum_{i=1}^{m}(x^{(i)}-\mu_y^{(i)})^2 \cdot \Sigma^{-2}$$

$$= -\frac{m}{2}\Sigma^{-1} + \frac{1}{2}\sum_{i=1}^{m}(x^{(i)}-\mu_y^{(i)})^2 \cdot \Sigma^{-2}$$

Let $\boxed{\frac{\partial \ell}{\partial \Sigma} = 0}$, then $\frac{m}{2}\Sigma^{-1} = \frac{1}{2}\sum_{i=1}^{m}(x^{(i)}-\mu_y^{(i)})^2 \cdot \Sigma^{-2}$

$$\boxed{\Sigma = \frac{1}{m} \cdot \sum_{i=1}^{m}(x^{(i)}-\mu_y^{(i)})^2}$$

4.(a)

4.(a) we want to find the relationship between $\nabla f(x)$ and $\nabla g(x)$, $\nabla^2 f(x)$ and $\nabla g^2(x)$

Let $f(x) = \theta X$, then we know $\nabla f(x) = \theta$

$$g(x) = f(AX) = \theta \cdot \begin{bmatrix} x_1 a_{11} + x_2 a_{12} \cdots x_n a_{1n} \\ x_1 a_{21} + x_2 a_{22} \cdots x_n a_{2n} \\ \vdots \\ x_1 a_{n1} + x_2 a_{n2} \quad x_n a_{nn} \end{bmatrix}$$

$$\text{so } \nabla g(x) = \begin{pmatrix} \theta_1 a_{11} + \theta_2 a_{21} + \cdots \theta_n a_{n1} \\ \theta_1 a_{12} + \theta_2 a_{22} + \cdots \theta_n a_{n2} \\ \vdots \\ \theta_1 a_{1n} + \theta_2 a_{2n} + \cdots \theta_n a_{nn} \end{pmatrix} = A^T \nabla f(x)$$

$$\boxed{\nabla g(x) = A^T \nabla f(x)} \quad ①$$

Now let $f(x) = x^T \cdot B \cdot x$, from quadratic function, we know

$$\nabla^2 f(x) = 2B$$

$$g(x) = f(AX) = (AX)^T \cdot B(AX)$$

$$= x^T (A^T \cdot B \cdot A) X$$

$$\text{therefor } \nabla^2 g(x) = 2 \cdot A^T \cdot B \cdot A = A^T \cdot \nabla^2 f(x) \cdot A$$

$$\boxed{\nabla^2 g(x) = A^T \cdot \nabla^2 f(x) \cdot A} \quad ②$$

with ① and ②, For induction, assume $x^{(i)} = Az^{(i)}$

$$z^{(i+1)} = z^{(i)} - \frac{\nabla g(z)}{\nabla^2 g(z)}$$

$$Az^{(i+1)} = Az^{(i)} - A \cdot \frac{A^T \nabla f(x)}{A^T \nabla^2 f(x) \cdot A}$$

$$= x^{(i)} - \frac{\nabla f(x)}{\nabla^2 f(x)}$$

$$= x^{(i+1)}$$

4.(b)

4. (b) For gradient decent, we don't need $\nabla^2 f(x)$

$$x^{(i+1)} = x^{(i)} - \alpha \nabla f(x)$$

so $z^{(i+1)} = z^{(i)} - \alpha \cdot A^T \nabla f(x)$

$$Az^{(i+1)} = Az^{(i)} - \alpha \, A \cdot A^T \nabla f(x)$$

Note $A \cdot A^T \neq I$, so for G.D, it's NOT invariant to linear representation

5.(a).i

5.(a)(i) let $X = \begin{pmatrix} x_{11}, x_{12} \cdots x_{1n} \\ x_{21}, x_{22} \cdots x_{2n} \\ \vdots \\ x_{m1}, x_{m2} \cdots x_{mn} \end{pmatrix}$ $\quad \theta = (\theta_1, \theta_2 \cdots \theta_n)^T$

$W = (W_1, W_2 \cdots W_m)$

$Y = (Y_1, Y_2 \cdots Y_n)$

Then $J(\theta) = \frac{1}{2} \sum_{i=1}^{m} w^{(i)} (\theta^T x^{(i)} - y^{(i)})^2$

$$= \frac{1}{2} W_1 \left((\theta_1 x_{11} + \theta_2 x_{12} + \cdots \theta_n x_{1n}) - Y_1\right)^2 + \frac{1}{2} W_2 \left((\theta_2 x_{21} + \theta_2 x_{22} + \cdots \theta_n x_{2n}) - Y_2\right)^2 + \cdots + \frac{1}{2} W_m \left((\theta_1 x_{m1} + \theta_2 x_{n2} + \cdots \theta_n x_{mn}) - Y_m\right)^2$$

$$= \begin{bmatrix} \sum_{i=1}^{n} \theta_i x_{1i} - Y_1, \\ \sum_{i=1}^{n} \theta_i x_{2i} - Y_2, \\ \vdots \\ \sum_{i=1}^{n} \theta_i x_{mi} - Y_m \end{bmatrix}^T \cdot \begin{bmatrix} \frac{W_1}{2} & & & 0 \\ & \frac{W_2}{2} & & \vdots \\ & & \ddots & \\ 0 & & & \frac{W_m}{2} \end{bmatrix} \begin{bmatrix} \sum_{i=1}^{n} \theta_i x_{1i} - Y_1, \\ \sum_{i=1}^{n} \theta_i x_{2i} - Y_2, \\ \vdots \\ \sum_{i=1}^{n} \theta_i x_{mi} - Y_m \end{bmatrix}$$

$$= (X\theta - \vec{y})^T \cdot W \cdot (X\theta - \vec{y})$$

where $W = \begin{bmatrix} \frac{W_1}{2} & 0 & \cdots & 0 \\ 0 & \frac{W_2}{2} & & \vdots \\ 0 & & \ddots & \\ 0 & \cdots & & \frac{W_m}{2} \end{bmatrix}$

5.(a).ii

(a)(ii)

5. similar to the class notes, use the following matrix rules

$$\nabla_{A^T} \operatorname{tr} ABA^TC = B^TA^TC + BA^TC \quad ①$$

$$\nabla_A \operatorname{tr} AB = B^T, \quad \nabla_{A^T} \operatorname{tr} AB = B \quad ②$$

$$W = W^T \quad ③$$

$$\nabla_\theta J(\theta) = \nabla_\theta (X\theta - y)^T W (X\theta - y)$$

$$= \nabla_\theta (\theta^T X^T W - y^T W)(X\theta - y)$$

$$(y^T W X \theta)^T = \theta^T X^T W y$$

$$= \nabla_\theta (\theta^T X^T W X \theta - y^T W X \theta - \theta^T X^T W y + y^T W y)$$

$$= \nabla_\theta (\operatorname{tr} \theta^T X^T W X \theta - 2 \operatorname{tr} \theta^T X^T W y)$$

use ①, let $A^T = \theta$, $B = X^T W X$, $C = I$

use ②, let $A^T = \theta$, $B = X^T W y$

$$= X^T W X \theta + X^T W X \theta - 2X^T W y$$

$$= 2X^T W X \theta - 2X^T W y$$

Let $\nabla_\theta J(\theta) = 0$

we have

$$2X^T W X \theta = X^T W y$$

$$\theta = (X^T W X)^{-1} \cdot X^T W y$$

5.(a).iii

5.(a)(iii)

$$\ell(\theta) = \log \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta)$$

$$= \sum_{i=1}^{m} \left(\log \frac{1}{\sqrt{2\pi}\theta} - \frac{1}{\theta^2} \cdot \frac{1}{2} (y^{(i)} - \theta^T x^{(i)})^2 \right)$$

$$= \underbrace{C}_{\text{constant}} - \underbrace{\frac{1}{2} \sum_{i=1}^{m} \frac{1}{\theta^2} (\theta^T x^{(i)} - y^{(i)})^2}_{\star}$$

Note ☆ is the same as weighted cost function $J(\theta)$

since ☆ is ~~minimize~~ negative, we know maximize $\ell(\theta) \Leftrightarrow$ minimize ☆

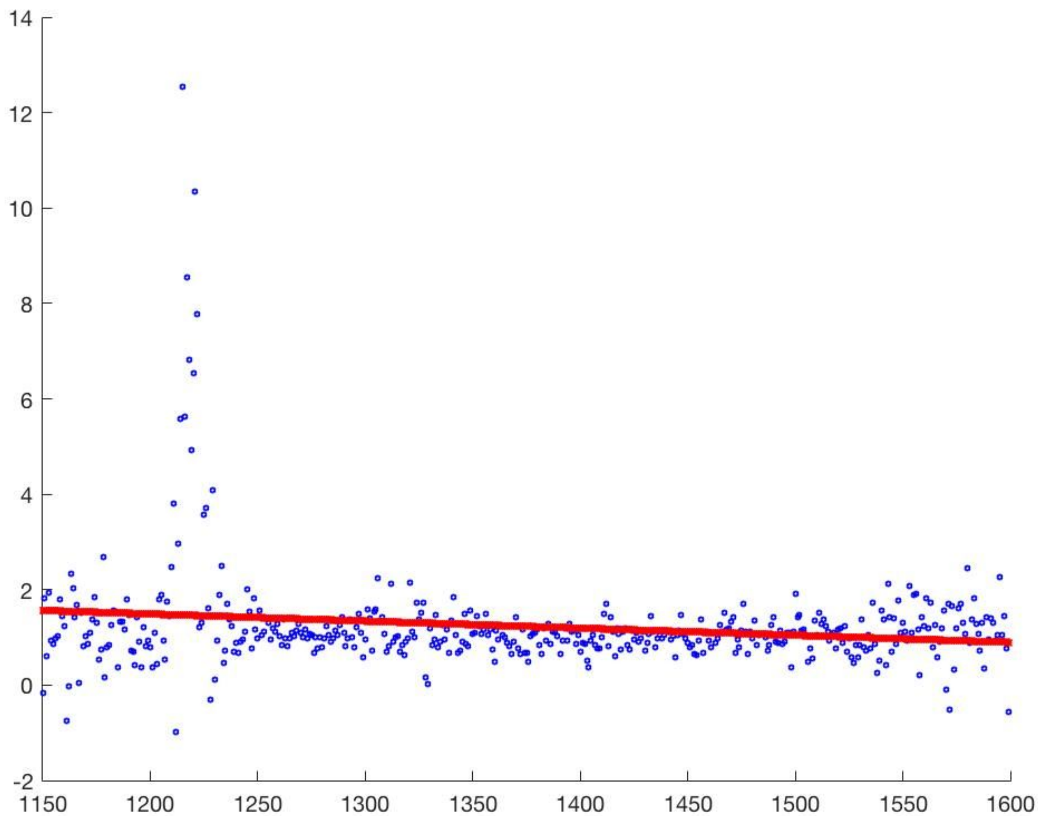
which is the same as minimize $J(\theta)$

It's also easy to see that $\boxed{\frac{1}{\sigma^{(i)2}} = w^{(i)}}$

5.(b).i

Snippet from weighted_linear_regression.m

```
% 5.b.i
% do_unweighted_linear_regression();


% for 5.b.i
function do_unweighted_linear_regression()
    [lambdas, train_qso, test_qso] = reload_data();
    X = lambdas;
    y = train_qso(2, :)';

    theta = unweighted_linear_regression(X, y);
    scatter(X, y, 5, 'b');
    hold on;
    scatter(X, theta(2, 1) * X + theta(1, 1), 5, 'r', '*');
    hold off;
end

function theta = unweighted_linear_regression(X,y)
    % X: mXn training examples
    % Y: mX1 results

    % ouptut theta: close form solution of unweight linear regression

    [m,n] = size(X);

    X = [ones(size(X,1),1), X]; % append col of ones for intercept term
    % theta = inv(X' * X) * X' * y;
    theta = (X' * X) \ X' * y;
end
```



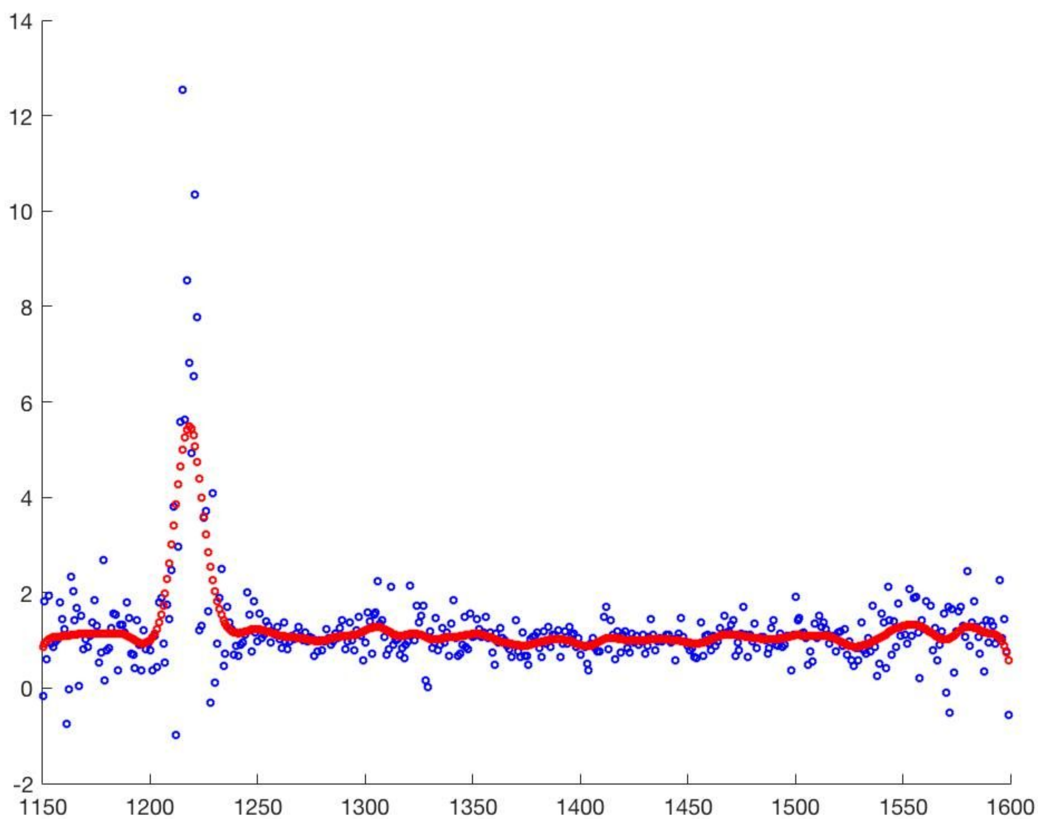
5_b_i

5.(b).ii

```
% 5.b.ii
% tau_vector = [5];
% do_weighted_linear_regression(tau_vector);

% for 5.b.ii
function do_weighted_linear_regression(tau)
    [lambdas, train_qso, test_qso] = reload_data();
```

```
    X = lambdas;
    y = train_qso(2, :)';

    scatter(X, y, 10, 'b');

    colors = ['r' 'g' 'c' 'y' 'm' 'c'];
    hold on;
    [m,~] = size(X);
    j = 1;
    for taui = tau
        weighted_results = zeros(m, 1);
        for i = 1:m
            % for wieghted linear regression, each query value(lambdas(i,1)) needs
            % to be passed to algorithm
            theta = weighted_linear_regression(X, y, taui, lambdas(i, 1));
            weighted_results(i, 1) = theta(2, 1) * X(i, 1) + theta(1, 1);
        end
        scatter(X, weighted_results, 10, colors(j));
        j = j+1;
    end
    hold off;
end

function theta = weighted_linear_regression(X,y,t,queryX)
    % X: mXn training examples
    % Y: mX1 results
    % t weight parameter
    % queryX: 1X1 the X value this algorithm is going to be run on

    % ouptut theta: close form solution of unweight linear regression

    [m, n] = size(X);

    X = [ones(size(X,1),1), X]; % append col of ones for intercept term

    W = weight_matrix(t, X, queryX);

%     theta = inv(X' * W * X) * X' * W * y;
    theta = (X' * W * X) \ X' * W * y;

end

function W = weight_matrix(t, X, queryX)
    % build a weight matrix from t and X
    % t: weight parameter
    % X: mXn - here X is 2X1
    % queryX: X value to run on
```

```
    [m, ~] = size(X);
    W = zeros(m,m);
    for i = 1:m
        % what if X has more than 2 columns?
        W(i,i) = exp(-(queryX - X(i, 2)) ^ 2 / (2 * t ^ 2));
    end
end
```



5_b_ii

5.(b).iii

5.(b) iii

From plot 5_b_3, we know that the bigger τ is, the more each weight assigned is close to one, the less effect the weight would take. so when τ is small, the output is more align with curved ~~training~~ data,

when τ is bigger, the output tends to become a straight line (like unweighted,

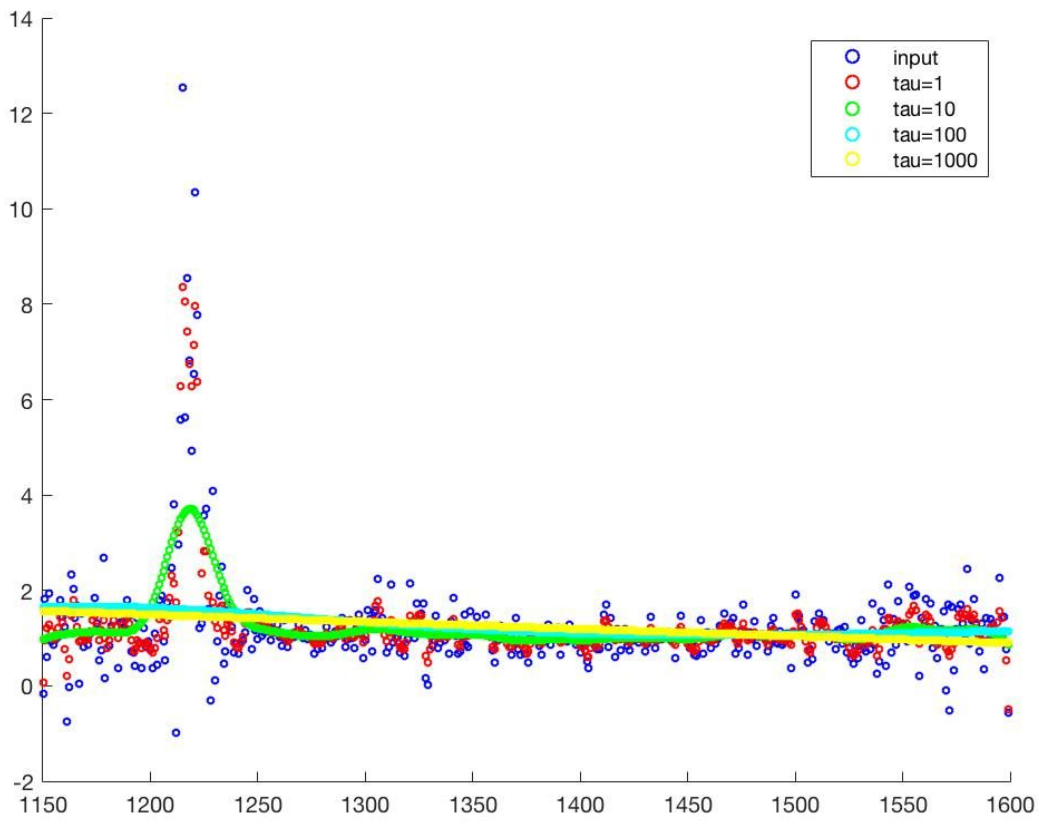
5.(c) please refer to the code file unweight_linear_regression.m

for 5.(c).ii, the average training error is 769.7348

for 5.(c).iii, the average training error is 30.3842

```
% 5.b.iii
%tau_vector = [1, 10, 100, 1000];
%do_weighted_linear_regression(tau_vector);


function do_weighted_linear_regression(tau)
    [lambdas, train_qso, test_qso] = reload_data();
    X = lambdas;
    y = train_qso(2, :)';

    scatter(X, y, 10, 'b');

    colors = ['r' 'g' 'c' 'y' 'm' 'c'];
    hold on;
    [m,~] = size(X);
    j = 1;
    for taui = tau
        weighted_results = zeros(m, 1);
        for i = 1:m
            % for wieghted linear regression, each query value(lambdas(i,1)) needs
            % to be passed to algorithm
            theta = weighted_linear_regression(X, y, taui, lambdas(i, 1));
            weighted_results(i, 1) = theta(2, 1) * X(i, 1) + theta(1, 1);
        end
        scatter(X, weighted_results, 10, colors(j));
        j = j+1;
    end
    hold off;
end

function theta = weighted_linear_regression(X,y,t,queryX)
    % X: mXn training examples
    % Y: mX1 results
    % t weight parameter
    % queryX: 1X1 the X value this algorithm is going to be run on

    % ouptut theta: close form solution of unweight linear regression

    [m, n] = size(X);

    X = [ones(size(X,1),1), X]; % append col of ones for intercept term

    W = weight_matrix(t, X, queryX);

%     theta = inv(X' * W * X) * X' * W * y;
    theta = (X' * W * X) \ X' * W * y;
```

```
end

function W = weight_matrix(t, X, queryX)
    % build a weight matrix from t and X
    % t: weight parameter
    % X: mXn - here X is 2X1
    % queryX: X value to run on
    [m, ~] = size(X);
    W = zeros(m,m);
    for i = 1:m
        % what if X has more than 2 columns?
        W(i,i) = exp(-(queryX - X(i, 2)) ^ 2 / (2 * t ^ 2));
    end
end
```



5_b_iii

5.(c).i

5.(b) iii

From plot 5_b_3, we know that the bigger τ is, the more each weight assigned is close to one, the less effect the weight would take. so when τ is small, the output is more align with curved ~~training~~ data, when τ is bigger, the output tends to become a straight line (like unweighted.

5.(c) please refer to the code file unweight_linear_regression.m

for 5.(c). ii, the average training error is 769.7348

for 5.(c).iii, the average training error is 30.3842

```
% 5.c.i
% [lambdas, train_qso, test_qso] = reload_data();
% smoothed_train_qso = smooth_data(train_qso, lambdas);
% smoothed_test_qso = smooth_data(test_qso, lambdas);


function smoothed_data = smooth_data(unsmoothed_data, lambdas)
% smooth the data using tau=5
% unsmoothed_data: a X 450
% lambdas: 1 X 450
    [a, m] = size(unsmoothed_data);
    smoothed_data = zeros(a, m);
    % apply the smooth to each row from the second row
    for i = 1:a
        y = unsmoothed_data(i, :)';
        result = create_weighted_results(lambdas, y, 5, lambdas);
        smoothed_data(i,:)=result';
    end
end

function weighted_results = create_weighted_results(X, y, tau, queries)
% map y to a weighted results, smooth the data
    [m,~] = size(X);
    weighted_results = zeros(m, 1);
    for i = 1:m
        % for wieghted linear regression, each query value(queries(i,1)) needs
        % to be passed to algorithm
        theta = weighted_linear_regression(X, y, tau, queries(i, 1));
        weighted_results(i, 1) = theta(2, 1) * X(i, 1) + theta(1, 1);
    end
end

function theta = weighted_linear_regression(X,y,t,queryX)
    % X: mXn training examples
    % Y: mX1 results
    % t weight parameter
    % queryX: 1X1 the X value this algorithm is going to be run on

    % ouptut theta: close form solution of unweight linear regression

    [m, n] = size(X);

    X = [ones(size(X,1),1), X]; % append col of ones for intercept term

    W = weight_matrix(t, X, queryX);
```

```
%     theta = inv(X' * W * X) * X' * W * y;
    theta = (X' * W * X) \ X' * W * y;

end

function W = weight_matrix(t, X, queryX)
    % build a weight matrix from t and X
    % t: weight parameter
    % X: mXn - here X is 2X1
    % queryX: X value to run on
    [m, ~] = size(X);
    W = zeros(m,m);
    for i = 1:m
        % what if X has more than 2 columns?
        W(i,i) = exp(-(queryX - X(i, 2)) ^ 2 / (2 * t ^ 2));
    end
end
```

5.c.ii

```
% 5.c.ii
%[training_error, estimated_fleft] = estimate_f_left(smoothed_train_qso);
% average training_error: 769.7348



function [training_error, estimated_fleft] = estimate_f_left(input_data)
   % calculate a matrix of m * 50 for estimated fleft values
   % the width is 50 because left only ranges from 1150 to 1199
   % input_data: m X n matrix, feed in training data and testing data

   [m, n] = size(input_data);
   estimated_fleft = zeros(m, n);

   for j = 1:m
      % furthest distance from current vector
      h_value = h(input_data, j);
      for i = 1:n
          % find 3 closest rows
          indices = neighb(input_data, j, 3);
          upper = 0;
          lower = 0;
          for index = indices
```

```
            distance = d(input_data(index,:), input_data(j,:), 1300);
            fleft = input_data(j, i);
            upper = upper + ker(distance/h_value) * fleft;
            lower = lower + ker(distance/h_value);
         end
         estimated_fleft(j, i) = upper/lower;
      end
   end

   training_error = zeros(m, 1);
   for i = 1:m
      % cost is fleft, calculate the entire row
      training_error(i,1) = d(input_data(i,:), estimated_fleft(i,:), 1150);
   end
end


function result = d(f1_vector, f2_vector, starting_phi_index)
% calculte the d between two vectors
% calculate from starting_phi_index for f(right) calculations
% both vectors are supposed to be 1*450
    % our index starts from 1150
    offset_index = starting_phi_index - 1150 + 1;
    difference_sqr_vector = (f1_vector - f2_vector) .^2;
    result = sum(difference_sqr_vector(1,offset_index:end));
end

function result = ker(t)
    result = max(1-t, 0);
end

function indices = neighb(X, row_index, k)
% find k row indices from X that are closest to
% X(row_index,:), closest defined in d
% X: m X n matrix
    target_row = X(row_index,:);
    [m,n] = size(X);
    distance_column = zeros(m,1);
    for i = 1:m
        distance_column(i, 1) = d(target_row, X(i,:), 1300);
    end
    [~, original_positions] = sort(distance_column);
    % staring from 2, the first is row_index itself
    indices = original_positions(2:2+k-1, 1);
end

function result = h(X, row_index)
```

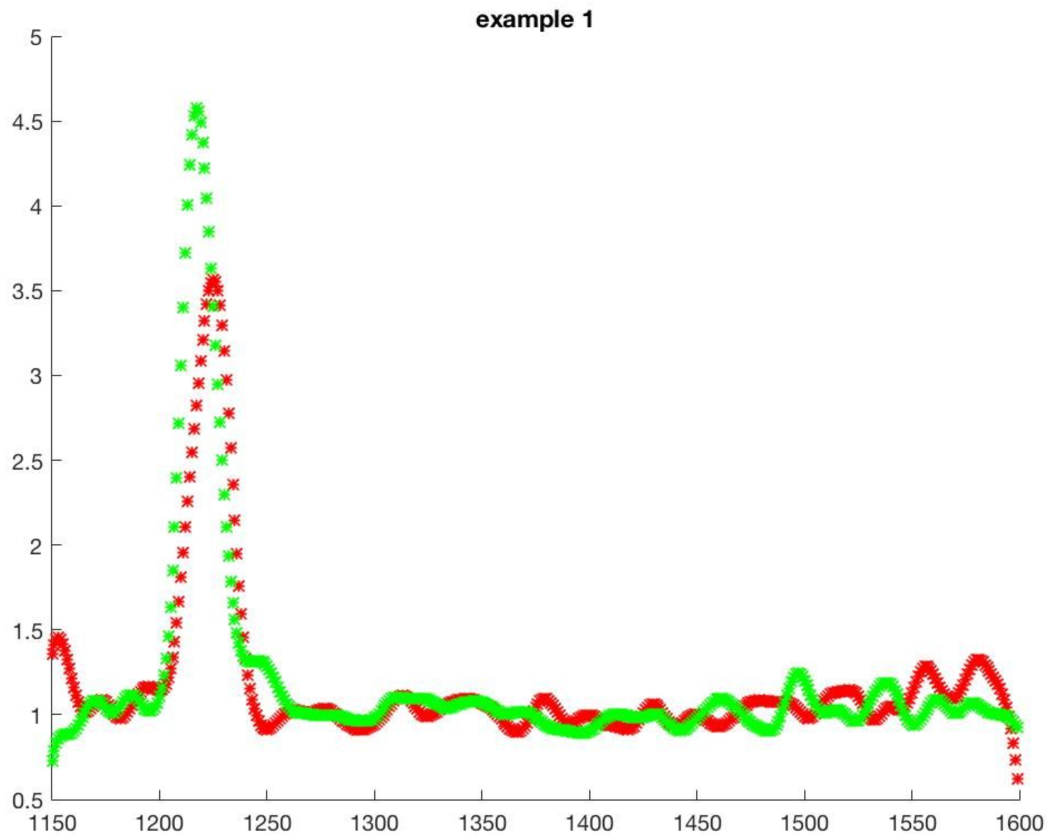
```
% calculate the max distance from X(row_index:)
% distance is defined in d and fright
% X: input matrix mXn, n=450 here
    result = 0;
    [row_count, ~] = size(X);
    for i=1:row_count
        distance = d(X(row_index,:), X(i,:), 1300);
        if distance > result
            result = distance;
        end
    end
end
```

5.c.iii

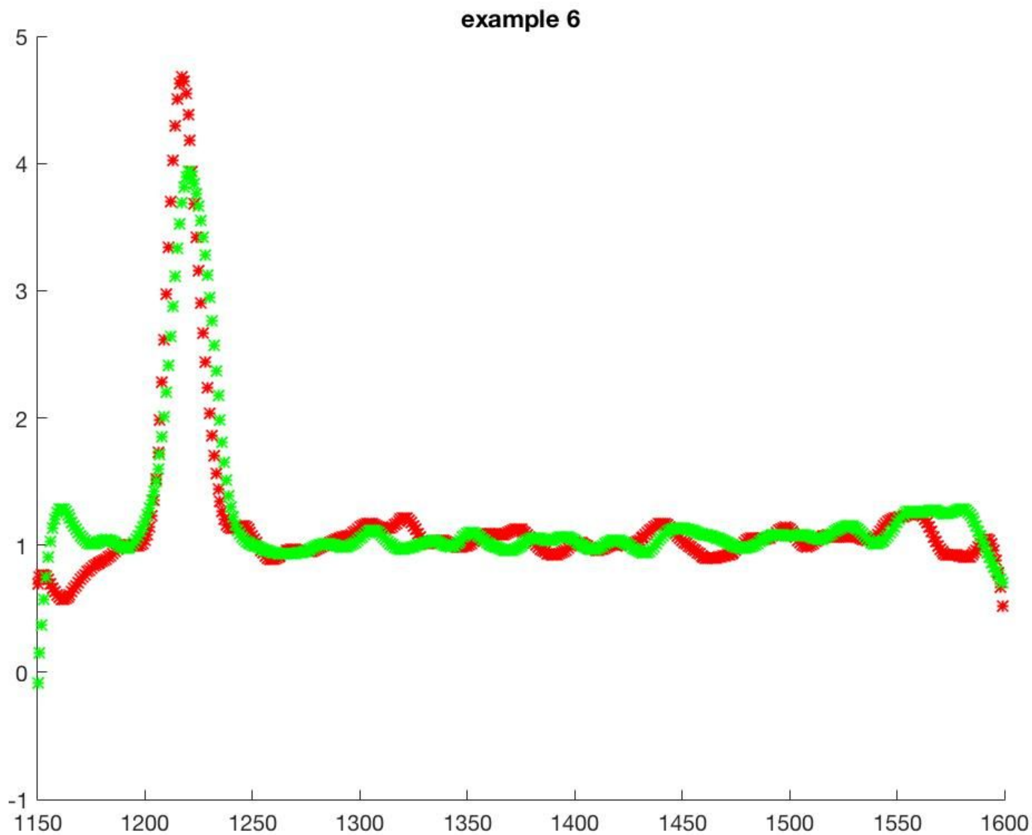
```
% 5.c.iii
% [training_error_test, estimated_fleft_test] =
estimate_f_left_with_test_data(smoothed_train_qso, smoothed_test_qso);
% average trainering_error: 30.3842

% test_output_example_1 = estimated_fleft_test(1,:);
% test_input_example_1 = smoothed_test_qso(1,:);
%
% hold on;
% title("example 1");
% scatter(lambdas, test_output_example_1, 20, 'r', '*');
% scatter(lambdas, test_input_example_1, 20, 'g', '*');
% hold off;
%
% test_output_example_6 = estimated_fleft_test(6,:);
% test_input_example_6 = smoothed_test_qso(6,:);
%
% hold on;
% title("example 6");
% scatter(lambdas, test_output_example_6, 20, 'r', '*');
% scatter(lambdas, test_input_example_6, 20, 'g', '*');
% hold off;


function [training_error, estimated_fleft] =
estimate_f_left_with_test_data(smoothed_train_qso, smoothed_test_qso)
   % calculate a matrix of m * 50 for estimated fleft values
   % the width is 50 because left only ranges from 1150 to 1199
```

```
  % input_data: m X n matrix, feed in training data and testing data

  [m, n] = size(smoothed_test_qso);
  estimated_fleft = zeros(m, n);

  for j = 1:m
    current_vector = smoothed_test_qso(j,:);
    % furthest distance from current vector
    h_value = h_test(smoothed_train_qso, current_vector);
    for i = 1:n
       % find 3 closest rows
       indices = neighb_test(smoothed_train_qso, current_vector, 3);
       upper = 0;
       lower = 0;
       for index = indices
          distance = d(smoothed_train_qso(index,:), current_vector, 1300);
          fleft = smoothed_train_qso(j, i);
          upper = upper + ker(distance/h_value) * fleft;
          lower = lower + ker(distance/h_value);
       end
       estimated_fleft(j, i) = upper/lower;
    end
  end

  training_error = zeros(m, 1);
  for i = 1:m
     % cost is fleft, calculate the entire row
     training_error(i,1) = d(smoothed_test_qso(i,:), estimated_fleft(i,:), 1150);
  end
end

function result = h_test(X, vector)
% calculate the max distance from vector
% distance is defined in d and fright
% X: input matrix mXn, n=450 here
   result = 0;
   [row_count, ~] = size(X);
   for i=1:row_count
      distance = d(vector, X(i,:), 1300);
      if distance > result
         result = distance;
      end
   end
end

function indices = neighb_test(X, vector, k)
% find k row indices from X that are closest to
```

```
% vector, closest defined in d
% X: m X n matrix
    [m,~] = size(X);
    distance_column = zeros(m,1);
    for i = 1:m
        distance_column(i, 1) = d(vector, X(i,:), 1300);
    end
    [~, original_positions] = sort(distance_column);
    indices = original_positions(1:k, 1);
end



function result = d(f1_vector, f2_vector, starting_phi_index)
% calculte the d between two vectors
% calculate from starting_phi_index for f(right) calculations
% both vectors are supposed to be 1*450
    % our index starts from 1150
    offset_index = starting_phi_index - 1150 + 1;
    difference_sqr_vector = (f1_vector - f2_vector) .^2;
    result = sum(difference_sqr_vector(1,offset_index:end));
end

function result = ker(t)
    result = max(1-t, 0);
end
```



5_c_iii_example1



5_c_iii_example6