

1. **Problem 1:** The log-loss function is defined as :

$$L = \sum_{i=1}^N \log(1 + e^{-l_i \langle x_i, \theta \rangle}) \quad (1)$$

We are required to compute L for all data-points provided in our training dataset. The matlab code to calculate the log-loss function is attached.

```

1 function L = calculate_log_loss(pts, labels, theta)
2 % CALCULATELOGLOSS - Function to calculate the log-likelihood of the data
3 % s described by pts, labels and theta. pts are the different values of x
4 % and y from the training samples. labels denote the direction where the
5 % data would be classified into.
6 L = 0;
7 [N m] = size(pts); %Number of items in the training set
8 for icnt = 1: N
9     L = L + log(1 + exp(-labels(icnt)*(pts(icnt,:) * theta)));
10 end

```

Listing 1: Matlab script for the Problem 1

We use the load command to populate the variables containing x_i and l_i . These correspond to `pts` and `labels`. We set different values for `theta` as inputs for the function and the outputs we get are approximately close to the expected value.

```

theta = [0;0;1];
l = calculate_log_loss(pts, labels, theta);
l = 81.3262
theta = [1;2;2];
l = calculate_log_loss(pts, labels, theta);
l = 36.4575

```

2. **Problem 2:** We have,

$$f(x, y) = (x - 2y)^2 + (y - 60)^2 \quad (2)$$

$$\frac{\partial f(x, y)}{\partial x} = 2(x - 2y) \quad \text{[Taking partial derivatives wrt x]}$$

$$\frac{\partial f(x, y)}{\partial y} = -4(x - 2y) + 2(y - 60) \quad \text{[Taking partial derivatives wrt y]}$$

This could also be represented by matrix equations as follows:

$$\nabla_x = \begin{bmatrix} 2 & -2 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ and } \nabla_y = \begin{bmatrix} -4 & 10 & -120 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

3. **Problem 3:** The lines of code are enclosed as follows:

```

1 function x_log = grad_desc(x0)
2 % GRAD_DESC -
3 global x_log g_log
4 stop_flag = 0;
5 x=x0;
6 x_log = [];
7 g_log = [];
8
9 [u, v] = meshgrid(1:256, 1:256);
10 z = (u - 2*v).^2 + (v - 60).^2;
11 n = 0.16; %THIS IS THE STEP SIZE

```

```

12   citer = 0;
13   while(~stop_flag)
14       fprintf(1, 'Iter: %d\n', citer);
15       citer=citer+1;
16       %FILL IN CODE FOR COMPUTING THE GRADIENT
17       %g(1,:) stores gradient along the horizontal direction(x) and g(2,:)
18       %stores the gradient along the vertical direction(y)
19       g(1,:) = 2*(x(1) - 2*x(2));
20       g(2,:) = (-4)*(x(1) - 2*x(2)) + 2*(x(2) - 60);
21
22       x_log = [x_log x];
23       g_log=[g_log -g*n];
24       %FILL IN CODE FOR UPDATING X
25       x = x + n*(-1)*g;
26
27       imagesc(z);
28       axis image
29       hold on
30       quiver(x_log(1,:) ', x_log(2,:) ', g_log(1,:) ', g_log(2,:) ', 0, 'g');
31       hold off;
32       f = (x(1)-2*x(2)).^2 + (x(2)-60).^2;
33       fprintf(1, 'f:%e\n', f);
34       if(citer > 500)
35           stop_flag=1;
36       end
37   end

```

Listing 2: Matlab script for the Problem 3

The following set of figures visualize the output of our program and demonstrate how after every iteration, the gradient descent function converges towards the solution.

4. **Problem 4:** In order to determine the gradient of L from (1), we are required to find its partial derivatives wrt θ .

$$\frac{\partial L}{\partial \theta} = \begin{bmatrix} \frac{\partial L}{\partial \theta_1} \\ \frac{\partial L}{\partial \theta_2} \\ \frac{\partial L}{\partial \theta_3} \end{bmatrix} \quad (3)$$

Partially differentiating wrt $\theta_1, \theta_2, \theta_3$, we get:

$$\begin{aligned} \frac{\partial L}{\partial \theta_1} &= \sum_{i=1}^N \frac{1}{1 + e^{-l_i \langle x_{i,1}, \theta_1 \rangle}} (-l_i x_{i,1}) e^{-l_i \langle x_{i,1}, \theta_1 \rangle} = \sum_{i=1}^N \frac{-l_i x_{i,1} e^{-l_i \langle x_{i,1}, \theta_1 \rangle}}{1 + e^{-l_i \langle x_{i,1}, \theta_1 \rangle}} \\ \frac{\partial L}{\partial \theta_2} &= \sum_{i=1}^N \frac{1}{1 + e^{-l_i \langle x_{i,2}, \theta_2 \rangle}} (-l_i x_{i,2}) e^{-l_i \langle x_{i,2}, \theta_2 \rangle} = \sum_{i=1}^N \frac{-l_i x_{i,2} e^{-l_i \langle x_{i,2}, \theta_2 \rangle}}{1 + e^{-l_i \langle x_{i,2}, \theta_2 \rangle}} \\ \frac{\partial L}{\partial \theta_3} &= \sum_{i=1}^N \frac{1}{1 + e^{-l_i \langle x_{i,3}, \theta_3 \rangle}} (-l_i x_{i,3}) e^{-l_i \langle x_{i,3}, \theta_3 \rangle} = \sum_{i=1}^N \frac{-l_i x_{i,3} e^{-l_i \langle x_{i,3}, \theta_3 \rangle}}{1 + e^{-l_i \langle x_{i,3}, \theta_3 \rangle}} \end{aligned}$$

5. **Problem 5:** The following code is used to calculate the gradient of log loss function as specified in Problem 1. A vector corresponding to the number of elements in the vector θ is returned.

```

1   function g=calculate_gradient_log_loss(pts, labels, theta)
2   % CALCULATE_GRADIENT_LOGLOSS - for each line parameters in theta, we would
3   % need to calculate the gradient.
4   % Number of items in training dataset
5   N = length(pts);
6   % number of line parameters
7   nt = length(theta);
8   % Initializing gradients for each of the line parameters.
9   g = zeros(nt, 1);

```

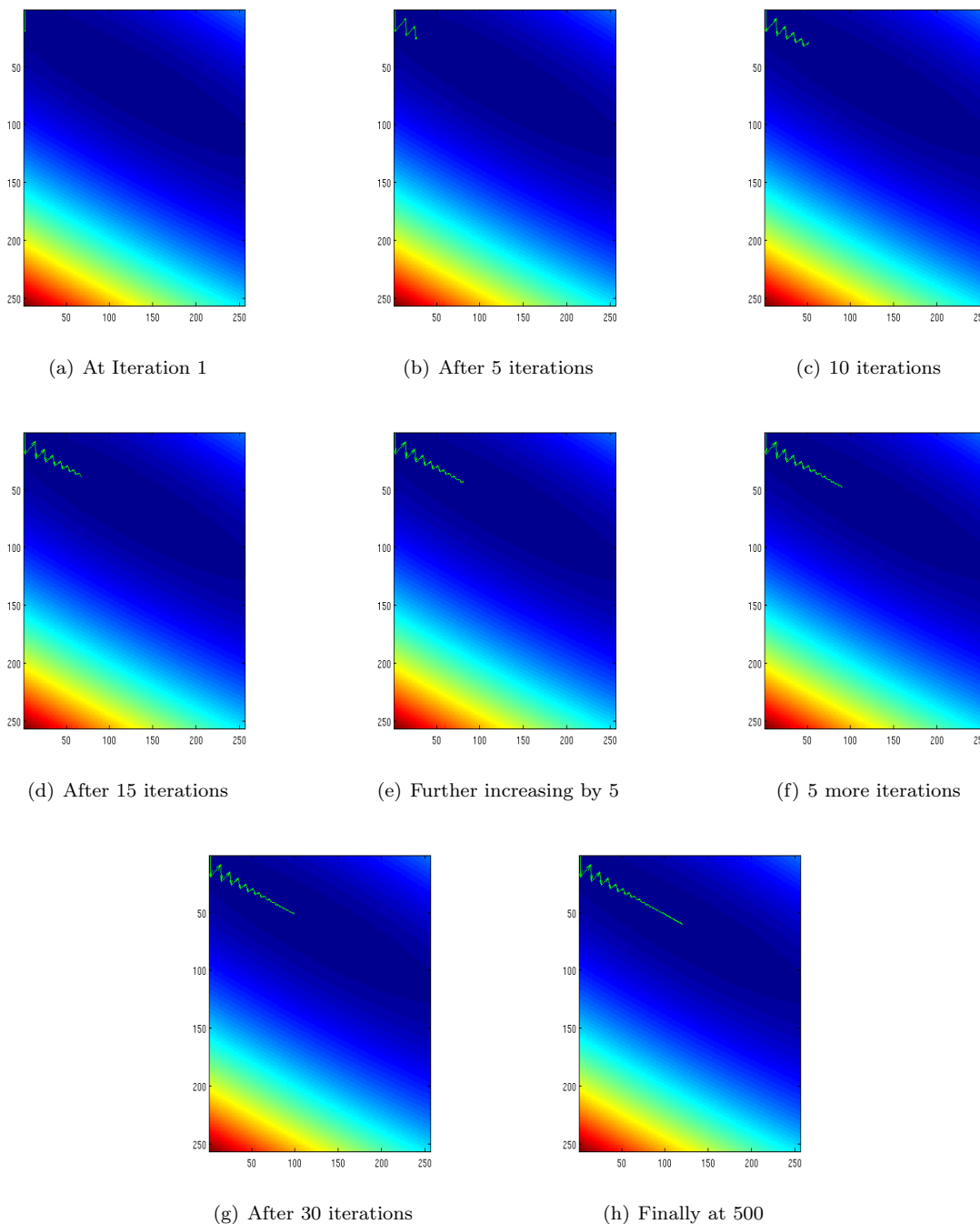


Figure 1: Outputs captured at various steps of execution of the Gradient Descent script

```

10
11 for icnt =1: N
12     for jcnt = 1:nt
13         c = -labels(icnt)*pts(icnt ,:)* theta;
14         y = -labels(icnt)*pts(icnt , jcnt);
15         g(jcnt) = g(jcnt) + y*exp(c)/(1+exp(c));
16     end
17 end
18 %Insert code to compute the gradient of the log loss
    
```

Listing 3: Matlab script for Calculating the Gradient of Log loss function given in Problem 1

Following are the results for using line parameters $\theta = [1, 1, 1]$:

```
>> theta = [1;1;1];
>> x = calculate_gradient_log_loss(pts,labels,theta)
x =
    -12.3636
     -9.2956
     11.2743
```

The complete code with the block to update the line parameters to calculate the value of the log-loss function for the desired number of iterations:

```
1 function logistic_regression(pts, labels)
2 % LOGISTIC REGRESSION -
3 %
4 theta0 = [1;0;5];
5 theta=theta0;
6 step = 0.0125;
7 max_num_iter=50;
8 stop_flag = 0;
9 citer=1;
10 pos_inds = find(labels >0);
11 neg_inds = find(labels <0);
12 while (~stop_flag)
13     plot(pts(pos_inds,1), pts(pos_inds,2), 'bx');
14     hold on;
15     plot(pts(neg_inds,1), pts(neg_inds,2), 'ro');
16     v=axis;
17     line_pt1 = (theta(3)-theta(1)*v(1))/theta(2);
18     line_pt2 = (theta(3)-theta(1)*v(2))/theta(2);
19     plot([v(1) v(2)], [line_pt1 line_pt2]);
20     hold off;
21     axis(v);
22     title(num2str(citer))
23     drawnow
24     pause(0.1)
25     g=calculate_gradient_log_loss(pts, labels, theta);
26     L=calculate_log_loss(pts, labels, theta);
27     fprintf(1, 'L:%e\n', L);
28     %INSERT CODE TO UPDATE THETA
29     theta = theta + step*(-1)*g;
30
31     if(citer >=max_num_iter)
32         stop_flag=1;
33     end
34     citer=citer+1;
35 end
```

Listing 4: Matlab script visualizing logistic regression

The results depict that after every iteration, we converge to a value close to $1.6e1$ as stated in the problem.

```
logistic_regression(pts, labels)
L:2.041555e+02
L:1.231060e+02
L:7.420995e+01
L:5.357203e+01
L:4.309031e+01
L:3.671882e+01
...
...
```

```
L:1.552061e+01  
L:1.550827e+01  
>>
```

The corresponding visualization for the above program :

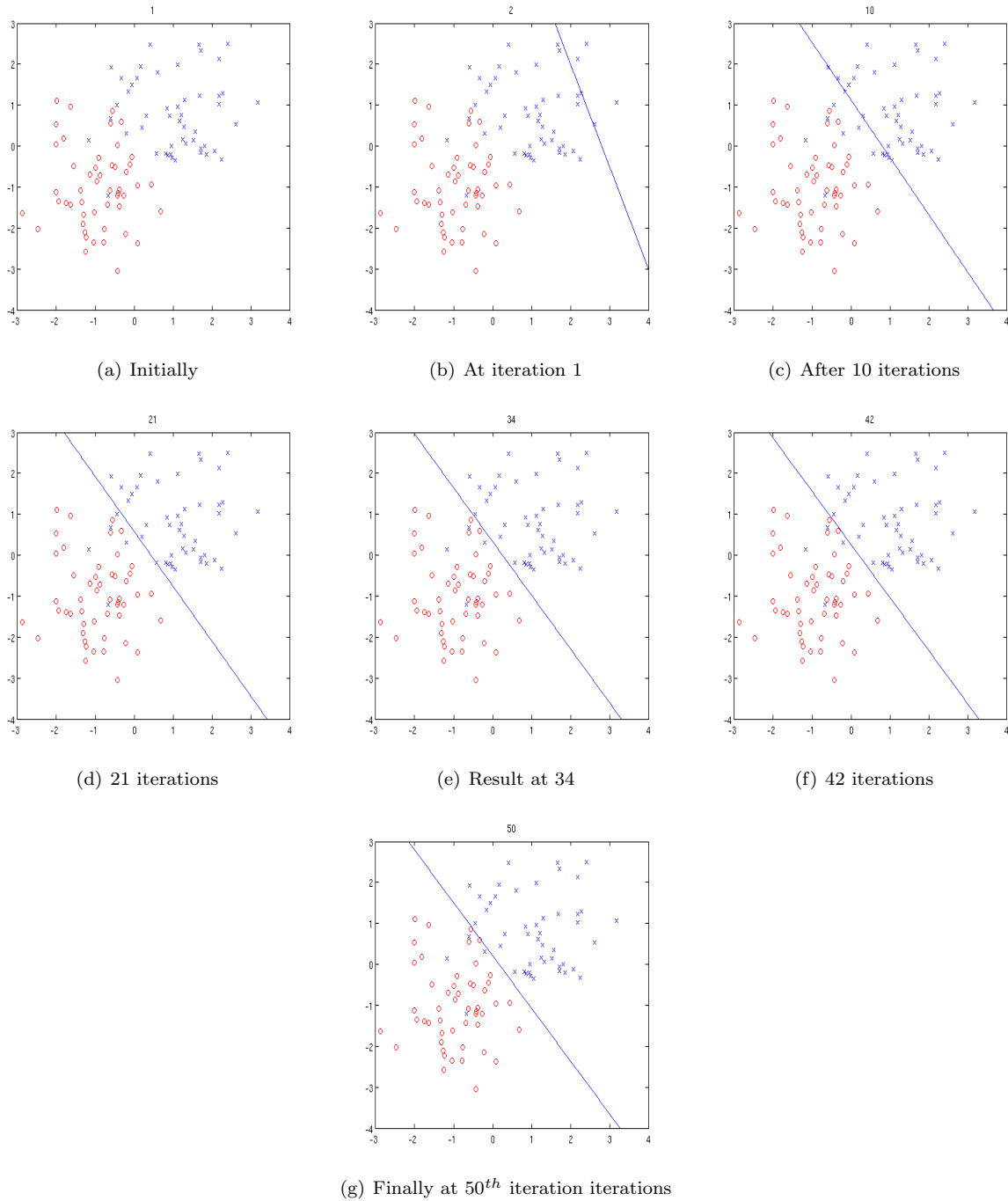


Figure 2: Output captured in various steps of execution of the Logistic Regression script