

Problem 1

(a)

The model parameter vector $\mathbf{m} = [x_s, y_s, z_s, P]^T$. The forward model is nonlinear, since the partial derivatives $\frac{\partial G}{\partial m_i}$ are not constant.

(b)

For least square problem, we introduce the objective function:

$$\phi = (\mathbf{G}(\mathbf{m}) - \mathbf{d})^T (\mathbf{G}(\mathbf{m}) - \mathbf{d}) = \sum_{i=1}^n \left(\frac{Pz_s}{[(x_s - x_i)^2 + (y_s - y_i)^2 + z_s^2]^{3/2}} - d_i \right)^2$$

define :

$$R_i = (x_s - x_i)^2 + (y_s - y_i)^2 + z_s^2$$

$$lx_i = x_i - x_s$$

$$ly_i = y_i - y_s$$

$$A_i = \frac{Pz_s}{[(x_s - x_i)^2 + (y_s - y_i)^2 + z_s^2]^{3/2}} - d_i$$

We write the Jacobian matrix:

$$\mathbf{J} = \begin{bmatrix} \frac{3Pz_x lx_1}{2R_1^{5/2}} & \frac{3Pz_x ly_1}{2R_1^{5/2}} & \frac{PR_1 - 3Pz_s^2}{2R_1^{5/2}} & \frac{z_s}{2R_1^{3/2}} \\ \frac{3Pz_x lx_2}{2R_2^{5/2}} & \frac{3Pz_x ly_2}{2R_2^{5/2}} & \frac{PR_2 - 3Pz_s^2}{2R_2^{5/2}} & \frac{z_s}{2R_2^{3/2}} \\ \dots & \dots & \dots & \dots \\ \frac{3Pz_x lx_n}{2R_n^{5/2}} & \frac{3Pz_x ly_n}{2R_n^{5/2}} & \frac{PR_n - 3Pz_s^2}{2R_n^{5/2}} & \frac{z_s}{2R_n^{3/2}} \end{bmatrix}$$

$$\nabla_{\mathbf{m}} \phi = (\mathbf{G}(\mathbf{m}) - \mathbf{d})^T \mathbf{J} = \left[\sum_{i=1}^n \left(\frac{Pz_s}{R_i^{3/2}} - d_i \right) \left(\frac{3Pz_x lx_i}{R_i^{5/2}} \right), \sum_{i=1}^n \left(\frac{Pz_s}{R_i^{3/2}} - d_i \right) \left(\frac{3Pz_x ly_i}{R_i^{5/2}} \right), \sum_{i=1}^n \left(\frac{Pz_s}{R_i^{3/2}} - d_i \right) \left(\frac{PR_i - 3Pz_s^2}{R_i^{5/2}} \right), \sum_{i=1}^n \left(\frac{Pz_s}{R_i^{3/2}} - d_i \right) \left(\frac{z_s}{R_i^{3/2}} \right) \right]^T$$

$$\mathbf{H}(\phi) = \nabla_{\mathbf{m}} (\nabla_{\mathbf{m}} \phi) = \nabla (\mathbf{J}^T (\mathbf{G}(\mathbf{m}) - \mathbf{d})) = \mathbf{J}^T \mathbf{J} + (\mathbf{G}(\mathbf{m}) - \mathbf{d})^T \mathbf{Q}$$

$$\mathbf{H}_{approximate} = \mathbf{J}^T \mathbf{J}$$

$$(\mathbf{G}(\mathbf{m}) - \mathbf{d})^T \nabla \mathbf{J} = \sum_{i=1}^n \frac{A_i}{R_i^{7/2}} \begin{bmatrix} 15Pz_s lx_i^2 - 3Pz_s R_i & 15Pz_s lx_i ly_i & 3Plx_i R_i - 15Pz_s^2 lx_i & 3z_s lx_i R_i \\ sym & 15Pz_s ly_i^2 - 3Pz_s R_i & 3Plly_i R_i - 15Pz_s^2 ly_i & 3z_s ly_i R_i \\ & & 15Pz_s^3 - 9Pz_s R_i & R_i^2 - 9z_s^2 R_i \\ & & & 0 \end{bmatrix}$$

$$\begin{aligned} H_{xx} &= \sum_i 9P^2 lx_i^2 z_s^2 R_i^{-5} + 15lx_i^2 z_s A_i R_i^{-7/2} - 3Pz_s A_i R_i^{-5/2} & H_{yy} &= \sum_i 9Plly_i^2 z_s^2 R_i^{-5} + 15ly_i^2 z_s A_i R_i^{-7/2} - 3Pz_s A_i R_i^{-5/2} \\ H_{zz} &= \sum_i (-6Pz_s^2 R_i^{-5/2} + PR_i^{-3/2})^2 + 15A_i Pz_s^3 R_i^{-7/2} - 9A_i Pz_s R_i^{-5/2} & H_{pp} &= \sum_i z_s^2 R_i^{-3} \\ H_{xy} &= \sum_i 9P^2 lx_i ly_i z_s^2 R_i^{-5} + 15lx_i ly_i z_s A_i R_i^{-7/2} & H_{xz} &= \sum_i 3Plx_i A_i R_i^{-5/2} - 15Plx_i z_s^2 R_i^{-7/2} + 3Plx_i z_s R_i^{-4} - 9P^2 lx_i z_s^3 R_i^{-5} \\ H_{xp} &= \sum_i 3Plx_i z_s^2 R_i^{-4} + 3lx_i z_s A_i R_i^{-5/2} & H_{yz} &= \sum_i 3Plly_i A_i R_i^{-5/2} - 15Plly_i z_s^2 R_i^{-7/2} + 3P^2 ly_i z_s R_i^{-4} - 9Plly_i ly_i z_s R_i^{-5} \\ H_{yp} &= \sum_i 3Plly_i z_s^2 R_i^{-4} + 3ly_i z_s A_i R_i^{-5/2} & H_{zp} &= \sum_i -3Pz_s^3 R_i^{-4} + P_i z_s R_i^{-3} - 3A_i z_s^2 R_i^{-5/2} + A_i R_i^{-3/2} \end{aligned}$$

The algorithm for finding solution is :

$\mathbf{m} = \mathbf{m}_0$ (set initial guess)

```

 $\mathbf{r} = (G(\mathbf{m}) - \mathbf{d})$ 
while  $\mathbf{r}^T \mathbf{r} > \text{errorbound}$ 
.....compute Hessian  $\mathbf{H}(\mathbf{m})$  and  $\mathbf{J}^T \mathbf{r}$ 
..... $\Delta \mathbf{m} = \mathbf{H}^{-1} \mathbf{J}^T \mathbf{r}$ 
..... $\mathbf{m} = \mathbf{m} - \Delta \mathbf{m}$ 
..... $\mathbf{r} = (G(\mathbf{m}) - \mathbf{d})$ 
end

```

(c)

```

1 function [ Grad, Hess] = compute_gradient_approx_hess( x,y,M,residue)
2
3 xs = M(1);
4 ys = M(2);
5 zs = M(3);
6 p = M(4);
7
8 R = ((x - xs).^2 + (y - ys).^2 + zs^2);
9
10 dx = x-xs;
11 dy = y-ys;
12
13 Jacob(:,1) = (3.*p.*zs.*(dx))./((R).^(5/2));
14 Jacob(:,2) = (3.*p.*zs.*(dy))./((R).^(5/2));
15 Jacob(:,3) = p./((R).^(3/2)) - (3*p.*zs.^2)./(R).^(5/2);
16 Jacob(:,4) = zs./((R).^(3/2));
17
18 Grad = (residue')* Jacob;
19 %this is the approximated Hessian;
20
21
22 %dHess = (G(m)-d)^T*Q
23 dHess = zeros(4);
24 dHess(1,2) = residue'*((15*p*zs*(dx).*(dy))./(4*(R).^(7/2)));
25 dHess(1,3) = residue'*((3*p.*(dx))./(2*(R).^(5/2)) - (15*p*zs.^2.*(dx))./(2*(R).^(7/2)));
26 dHess(1,4) = residue'*((3*zs.*(dx))./(2*(R).^(5/2)));
27 dHess(2,3) = residue'*((3*p.*(dy))./(2*(R).^(5/2)) - (15*p*zs.^2.*(dy))./(2*(R).^(7/2)));
28 dHess(2,4) = residue'*((3*zs.*(dy))./(2*(R).^(5/2)));
29 dHess(3,4) = residue'*(1./((R).^(3/2)) - (3*zs.^2)./(R).^(5/2));
30
31 dHess = (dHess + dHess');
32
33 dHess(1,1) = residue'*((15*p*zs.*(dx).^2)./(4*(R).^(7/2)) - (3*p.*zs)./(R).^(5/2));
34 dHess(2,2) = residue'*((15*p*zs.*(dy).^2)./(4*(R).^(7/2)) - (3*p.*zs)./(R).^(5/2));
35 dHess(3,3) = residue'*((15*p*zs.^3)./(R).^(7/2) - (9*p*zs)./(R).^(5/2));
36 dHess(4,4) = 0;
37
38
39
40 Hess = (Jacob')*Jacob;
41 %only add dHess if using exact Hessian
42 Hess = Hess+dHess;
43
44 Hess = 0.5*(Hess + Hess');
45
46 end

```



```

1
2 function [M]=nonlinear_solver(x,y,d,Minit)
3
4 %x = [0 11 15 6 -7 3]';
5 %y = [0 0 6 13 10 -7]';
6 %d = [0.103 0.162 0.065 0.036 0.025 0.169]'+error;
7 M=Minit;
8 %M=[3 -7 10 20]';
9
10 %lambda = 1e-5;
11 for ii = 1:1:1000
12
13 r=compute_residue(x,y,M,d);
14
15 %disp(norm(r));
16
17 [Grad,Hess]=compute_gradient_approx_hess(x,y,M,r);
18
19 %deltaM = (Hess+lambda*eye(4))\Grad';
20 deltaM = (Hess)\Grad';
21
22 M=M-deltaM;
23
24 if (norm(r)<1e-7)
25 break;
26 end
27 end
28 end

```

```

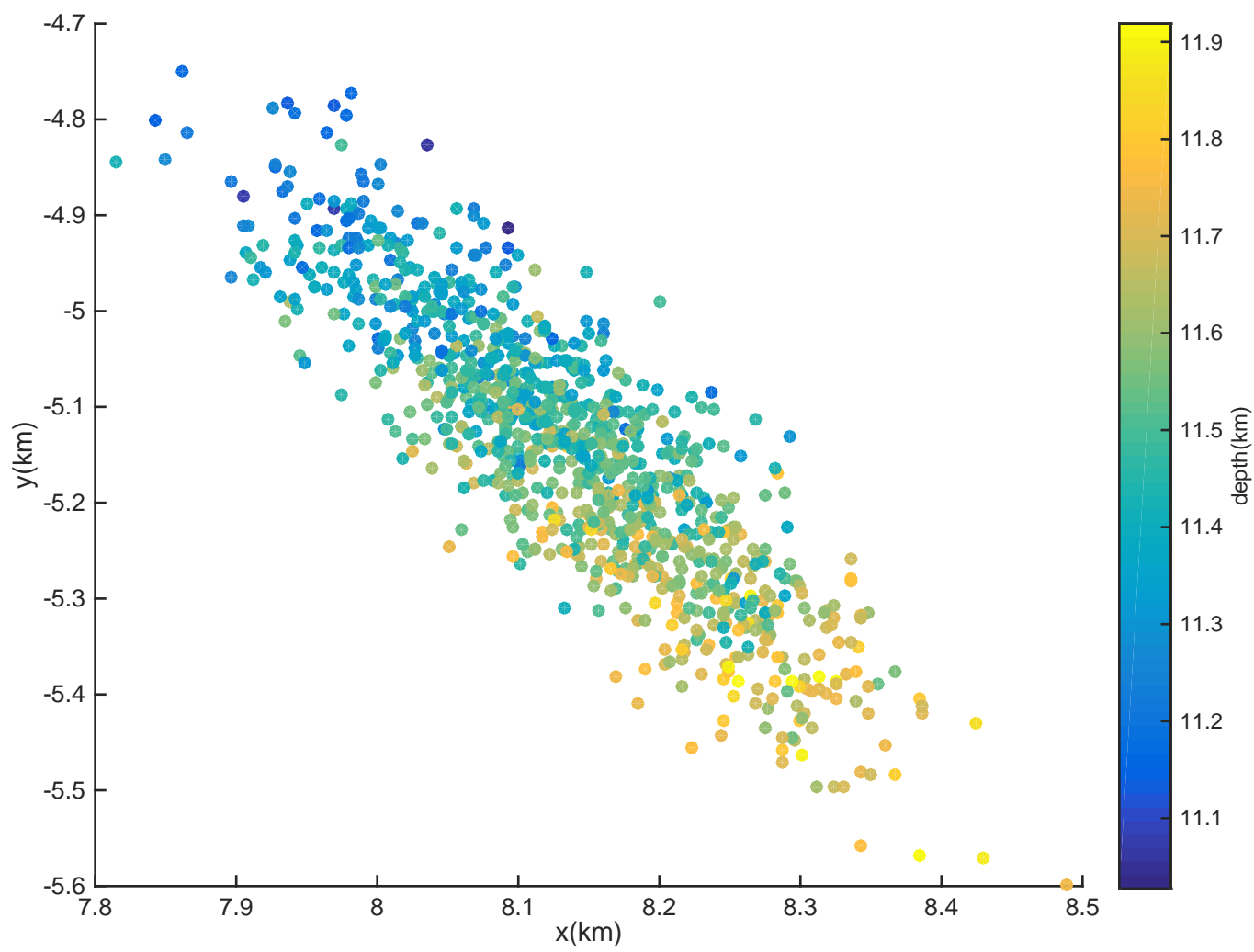
1  %%%problem 1d
2  x = [0 11 15 6 -7 3]';
3  y = [0 0 6 13 10 -7]';
4  d = [0.103 0.162 0.065 0.036 0.025 0.169]';
5  M0 = [8 -5 10 30]'; %initial guess
6  Ms = nonlinear_solver(x,y,d,M0);
7
8
9
10
11 %%%problem 1e
12 Mrec = zeros(4,1000);
13
14 for it = 1:1:1000
15     derror = d + 0.001*randn(6,1);
16     Merror = nonlinear_solver(x,y,derror,M0);
17     Mrec(:,it) = Merror;
18 end
19
20 scatter(Mrec(1,:),Mrec(2,:),30, Mrec(3,:), 'fill ');
21 stdx=std(Mrec(1,:));
22 stdy=std(Mrec(2,:));
23 stdz=std(Mrec(3,:));
24 stdp=std(Mrec(4,:));
25 c = colorbar;
26 ylabel(c,'depth(km)');
27 xlabel('x(km)');
28 ylabel('y(km)');
29
30 print('measurements_error.pdf','-dpdf');

```

(d)

$$[x_s, y_s, z_s, P]^T = [8.226, -5.307, 11.577, 30.781]^T$$

(e)

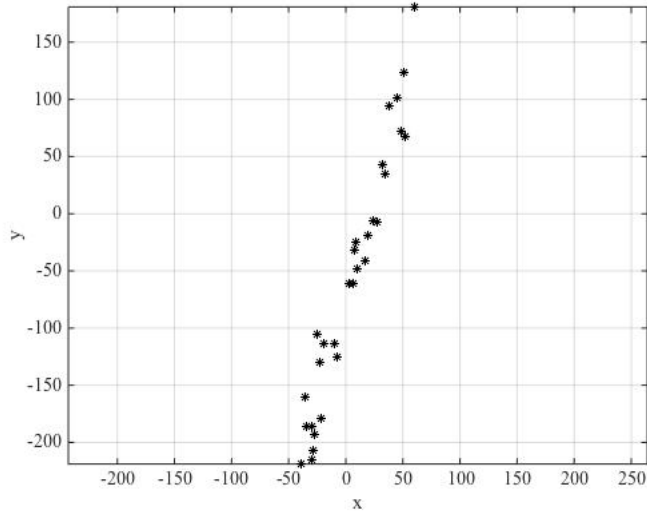


The standard deviations are $\sigma_{x_s} = 0.099670$, $\sigma_{y_s} = 0.137469$, $\sigma_{z_s} = 0.149719$, $\sigma_p = 0.691071$

There is a strong trade off relation between x_s, y_s, z_s . x_s, y_s are negatively related. x_s, z_s are positively related. z_s, y_s are negatively related. Note that we use z_s as depth value. It is always non-negative.

Problem 2 (graded by Yiran) - 50 points

(a) 4 points



(b) 6 points

m_1 is the intercept with the y axis. From the plot, we estimate that it should be bounded by $[-150, 0]$.

m_2 is the slope of the line, we also estimate from the plot that it should be bounded by $[1, 10]$.

As suggested in the problem, the arrays are better no larger than a few megabytes (1 double = 8 bytes) to avoid “out of memory” error. A 1000 by 1000 double-type matrix is 8 megabytes. Therefore, we can choose the discretization as $m1 = [-150 : 0.1 : 0]$, and $m2 = [1 : 0.01 : 10]$, so that the matrices of size $\text{length}(m1)$ by $\text{length}(m2)$ (e.g. the error matrix plotted in (d)), will be in appropriate size.

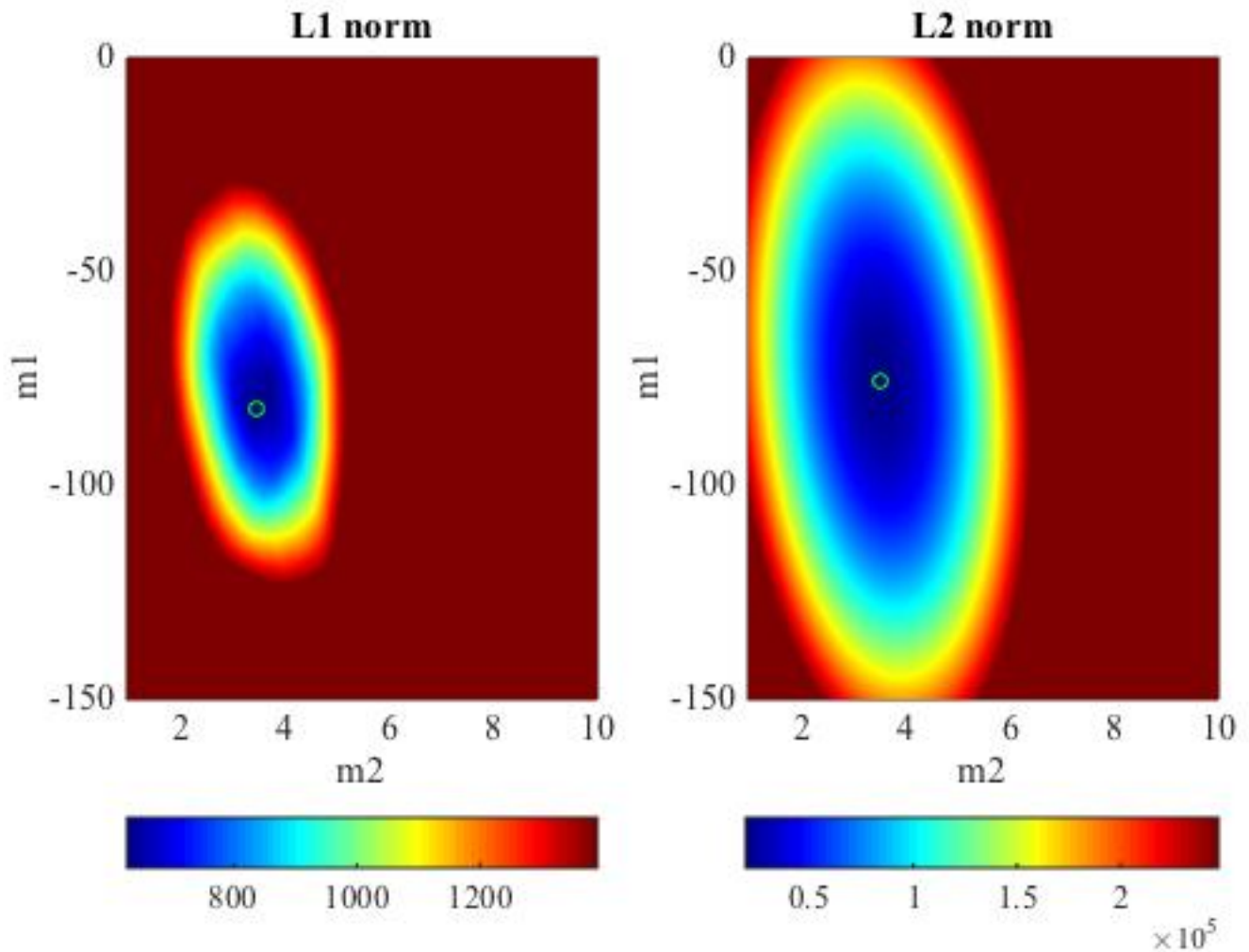
We can always shrink our model space and do a finer search as following steps.

(c) 6 points

(See attached MATLAB code.)

(d) 10 points

(See attached MATLAB code.)

**(e) 4 points**

The solutions with lowest misfit are

L2 norm: (-75.5, 3.53)

L1 norm: (-81.9, 3.48)

(f) 10 points

(See attached MATLAB code.)

The least square solution is: (-75.4631, 3.5301)

(g) 4 points

From (d), we infer that the model parameters are negatively correlated, and also the error is underestimated (much larger than 0.1). It is not given in the standard result. The standard result gives

the standard deviations of each parameter, which are the diagonal terms of the model covariance matrix. It will underestimate the range of possible model parameters, if the model parameters are not independent, or the data error is underestimated. Therefore, the extra information given in (d) is also very important.

(h) 6 points

(This is an open question.)

The L1 and L2 methods (“grid search”) are more straightforward in showing the error distribution, thus the probability of the model parameters over the “full” model space.

The least-square solution (“direct inversion”) is fast, and gives the exact solution that minimizes L2 norm error. We can also infer the covariances of the model parameters from the model covariance matrix. However, it is not as straightforward, and sometimes gives an illusion that the results have small errors/standard deviations. In this small size problem, I would prefer the “grid search” method.

As the errors get bigger, the L1 norm method can be better, because it would be less likely to be affected by the outliers. Moreover, if there are several solutions that can minimize the error equally well, we can see it in the error map produced by the direct search method, and can choose one solution based on some priori information. Therefore, I will prefer L1 norm.

```

1 function hw2_2
2 set(0,'defaulttextfontname','times','defaulttextfontsize',14);
3 set(0,'defaultaxesfontname','times','defaultaxesfontsize',14);
4
5 % load data
6 load gel18_hw2.mat
7
8 % plot data
9 figure(1)
10 plot(x,y,'k*');
11 grid on; axis equal;
12 xlabel('x'); ylabel('y');
13
14 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
15 % grid search
16 m1 = -150:0.1:0;
17 m2 = 1:0.01:10;
18
19 % L1 norm
20 [m1_l1,m2_l1,err_l1,err_all_l1] = grid_search(x,y,m1,m2,1);
21
22 % L2 norm
23 [m1_l2,m2_l2,err_l2,err_all_l2] = grid_search(x,y,m1,m2,2);
24
25 % plotting
26 figure(2)
27 colormap(jet);
28 subplot(121)
29 pcolor(m2,m1,err_all_l1); % error map
30 hold on;
31 plot(m2_l1,m1_l1,'go'); % optimum solution
32 shading flat;
33 caxis(crange(err_all_l1));
34 colorbar('horiz');
35 xlabel('m2'); ylabel('m1'); title('L1 norm');
36 hold off;
37
38 subplot(122)
39 pcolor(m2,m1,err_all_l2); % error map
40 hold on;
41 plot(m2_l2,m1_l2,'go'); % optimum solution
42 shading flat;
43 caxis(crange(err_all_l2));
44 colorbar('horiz');
45 xlabel('m2'); ylabel('m1'); title('L2 norm');
46 hold off;
47
48 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
49 % least square
50 [m1_ls,m2_ls] = least_square(x,y);
51
52 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
53 disp('L1 norm          L2 norm          LS');
54 disp('m1');
55 disp([m1_l1    m1_l2    m1_ls]);
56 disp('m2');
57 disp([m2_l1    m2_l2    m2_ls]);
58
59 end
60
61 % LEAST SQUARE SOLUTION

```

```

62 function [m1, m2] = least_square(xdata, ydata)
63 G = [ones(length(xdata),1) xdata(:)];
64 tmp = (G'*G)^(-1) * G' * ydata;
65 m1 = tmp(1);
66 m2 = tmp(2);
67 end
68
69 % GRID SEARCH
70 function [m1_best, m2_best, err_best, err] = grid_search(xdata, ydata, m1, m2, flag)
71 % error over the model space
72 err = zeros(length(m1), length(m2));
73 for i = 1:length(m1)
74     for j = 1:length(m2)
75         err(i, j) = misfit(xdata, ydata, m1(i), m2(j), flag);
76     end
77 end
78
79 % find the optimum solution
80 [err_best, id] = min(err(:));
81 [I, J] = ind2sub(size(err), id);
82 m1_best = m1(I);
83 m2_best = m2(J);
84 end
85
86 % CALCULATE THE MISFIT
87 function err = misfit(xdata, ydata, m1, m2, flag)
88 ypred = m1 + m2 * xdata;
89 if flag == 1 % L1 norm
90     err = sum(abs(ypred-ydata));
91 elseif flag == 2 % L2 norm
92     err = sum((ypred-ydata).^2);
93 end
94 end
95
96 % caxis for error plot
97 function vec = crange(err)
98 minval = min(err(:));
99 maxval = minval + 0.15 * (max(err(:)) - minval);
100 vec = [minval maxval];
101 end

```