Problem 1 (graded by Kangchen) - 50 points+10 bonus points

(a)10 points

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

(b)10 points

For least square problem, we introduce the objective function:

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

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:

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

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

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

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

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

$$H_{xx} = \sum_i 9P^2 lx_i{}^2 z_s^2 R_i^{-5} + 15lx_i^2 z_s A_i R_i^{-7/2} - 3Pz_s A_i R_i^{-5/2}$$

$$H_{zz} = \sum_i(-6Pz_s^2 R_i^{-5/2} + PR_i^{-3/2})^2 + 15A_i P z_s^3 R_i^{-7/2} - 9A_i P z_s R_i^{-5/2}$$

$$H_{xy} = \sum_i 9P^2 lx_i ly z_i{}_s^2 R_i^{-5} + 15lx_i ly_i z_s A_i R_i^{-7/2}$$

$$H_{xp} = \sum_i 3Plx_i z_s^2 R_i^{-4} + 3lx_i z_s A_i R_i^{-5/2}$$

$$H_{yp} = \sum_i 3Ply_i z_s^2 R_i^{-4} + 3ly_i z_s A_i R_i^{-5/2}$$

$$H_{yy} = \sum_i 9Ply_i{}^2 z_s^2 R_i^{-5} + 15ly_i^2 z_s A_i R_i^{-7/2} - 3Pz_s A_i R_i^{-5/2}$$

$$H_{pp} = \sum_i z_s^2 R_i^{-3}$$

$$H_{xz} = \sum_i 3Plx_i A_i R_i^{-5/2} - 15Plx_i z_s^2 R_i^{-7/2} + 3Plx_i z_s R_i^{-4} - 9P^2 lx_i z_s^3 R_i^{-5}$$

$$H_{yz} = \sum_i 3Ply_i A_i R_i^{-5/2} - 15Ply_i z_s^2 R_i^{-7/2} + 3P^2 ly_i z_s R_i^{-4} - 9Plx_i ly_i z_s R^{-5}$$

$$H_{zp} = \sum_i -3Pz_s^3 R_i^{-4} + P_i z_s R_i^{-3} - 3A_i z_s^2 R_i^{-5/2} + A_i R_i^{-3/2}$$

Note: this is the exact hessian, set $A_i = 0$ will make the approximated Hessian.
The algorithm for finding solution is :

$\boldsymbol{m} = \boldsymbol{m}_0$ (set initial guess)
$\boldsymbol{r} = (G(\boldsymbol{m}) - \boldsymbol{d})$
while $\boldsymbol{r}^{\boldsymbol{T}}\boldsymbol{r}$>errorbound
......compute Hessian $\boldsymbol{H}(\boldsymbol{m})$ and $\boldsymbol{J}^T\boldsymbol{r}$
......$\Delta\boldsymbol{m} = \boldsymbol{H}^{-1}\boldsymbol{J}^T\boldsymbol{r}$
......$\boldsymbol{m} = \boldsymbol{m} - \Delta\boldsymbol{m}$
......$\boldsymbol{r} = (G(\boldsymbol{m}) - \boldsymbol{d})$
end

(c)10 points

```
function [ Grad, Hess] = compute_gradient_approx_hess( x,y,M,residue)

xs = M(1);
ys = M(2);
zs = M(3);
p = M(4);

R = ((x - xs).^2 + (y - ys).^2 + zs^2);

dx = x-xs;
dy = y-ys;

Jacob(:,1) = (3.*p.*zs.*(dx))./((R).^(5/2));
Jacob(:,2) = (3.*p.*zs.*(dy))./((R).^(5/2));
Jacob(:,3) = p./(R).^(3/2) - (3*p.*zs.^2)./(R).^(5/2);
Jacob(:,4) = zs./(R).^(3/2);

Grad = (residue')* Jacob;
%this is the apprximated Hessian;


%dHess = (G(m)-d)^T*Q
dHess = zeros(4);
dHess(1,2) = residue'*((15*p*zs*(dx).*(dy))./(4*(R).^(7/2)));
dHess(1,3) = residue'*((3*p.*(dx))./(2*(R).^(5/2)) - (15*p*zs.^2.*(dx))./(2*(R).^(7/2)));
dHess(1,4) = residue'*((3*zs.*(dx))./(2*(R).^(5/2)));
dHess(2,3) = residue'*( (3*p.*(dy))./(2*(R).^(5/2)) - (15*p*zs.^2.*(dy))./(2*(R).^(7/2)));
dHess(2,4) = residue'*((3*zs.*(dy))./(2*(R).^(5/2)));
dHess(3,4) = residue'*(1./(R).^(3/2) - (3*zs^2)./(R).^(5/2));

dHess = (dHess + dHess');

dHess(1,1) = residue'*((15*p*zs.*(dx).^2)./(4*(R).^(7/2)) - (3*p.*zs)./(R).^(5/2));
dHess(2,2) = residue'*((15*p*zs.*(dy).^2)./(4*(R).^(7/2)) - (3*p.*zs)./(R).^(5/2));
dHess(3,3) = residue'*((15*p*zs^3)./(R).^(7/2) - (9*p*zs)./(R).^(5/2));
dHess(4,4) = 0;



Hess = (Jacob')*Jacob;
%only add dHess if using exact Hessian
Hess = Hess+dHess;

Hess = 0.5*(Hess + Hess');

end
```

```

function [M]=nonlinear_solver(x,y,d,Minit)

%x = [0 11 15 6 -7 3]';
%y = [0 0 6 13 10 -7]';
%d = [0.103 0.162 0.065 0.036 0.025 0.169]'+error;
M=Minit;
%M=[3 -7 10 20]';

%lambda = 1e-5;
for ii = 1:1:1000

r=compute_residue(x,y,M,d);

%disp(norm(r));

[Grad,Hess]=compute_gradient_approx_hess(x,y,M,r);

%deltaM = (Hess+lambda*eye(4))\Grad';
deltaM= (Hess)\Grad';

M=M-deltaM;

if (norm(r)<1e-7)
    break;
end
end
```

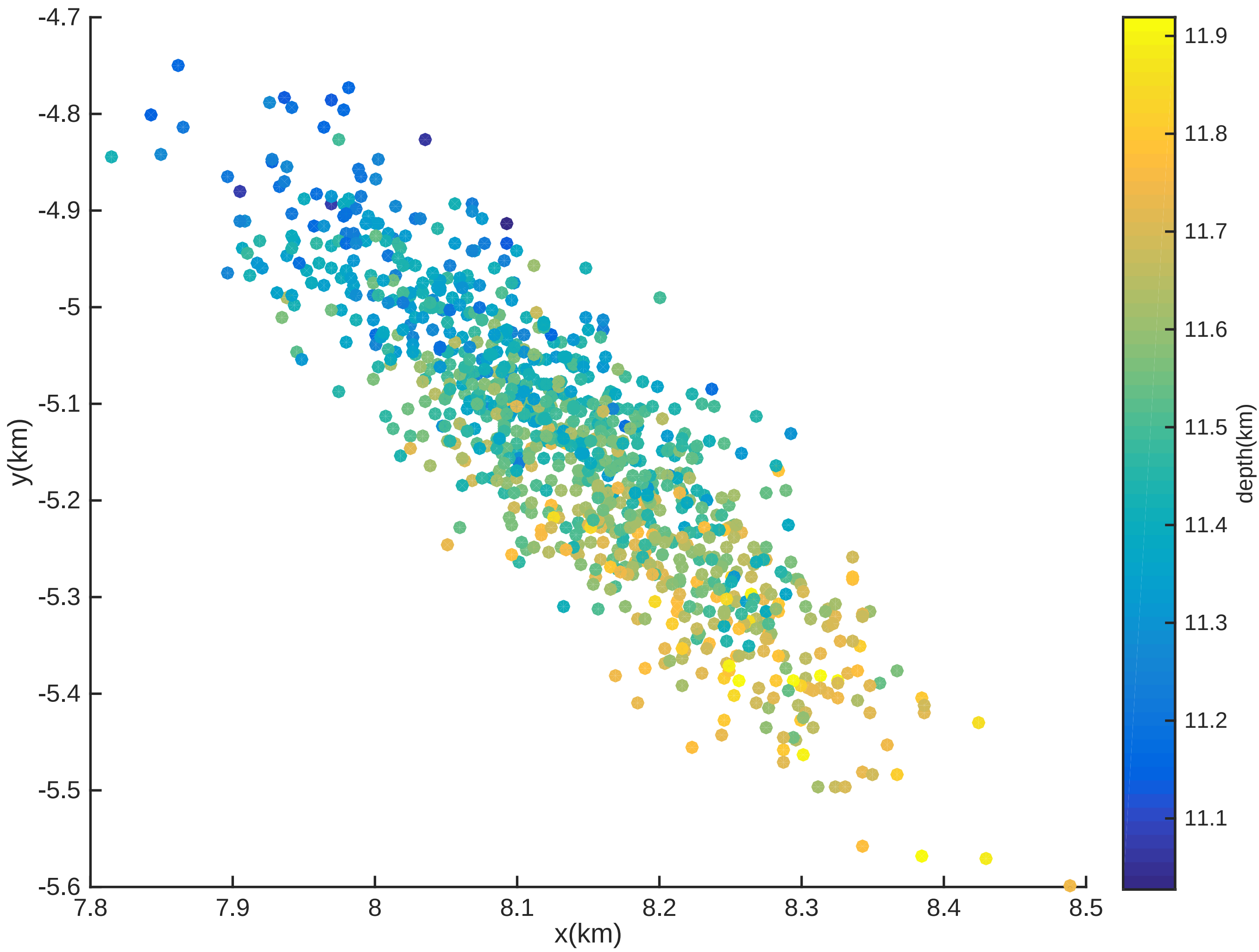
```
end
```

```
%%%problem 1d
x = [0 11 15 6 -7 3]';
y = [0 0 6 13 10 -7]';
d = [0.103 0.162  0.065  0.036 0.025  0.169]';
M0 = [8 -5 10 30]';  %initial guess
Ms = nonlinear_solver(x,y,d,M0);




%%%problem 1e
Mrec = zeros(4,1000);

for it = 1:1:1000
    derror = d + 0.001*randn(6,1);
    Merror = nonlinear_solver(x,y,derror,M0);
    Mrec(:,it) = Merror;
end

scatter(Mrec(1,:),Mrec(2,:),30, Mrec(3,:),'fill');
stdx=std(Mrec(1,:));
stdy=std(Mrec(2,:));
stdz=std(Mrec(3,:));
stdp=std(Mrec(4,:));
c = colorbar;
ylabel(c,'depth(km)');
xlabel('x(km)');
ylabel('y(km)');

print('measurements_error.pdf','-dpdf');
```

(d)10 points

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

(e)10 points


-4.7
-4.8
-4.9
-5
-5.1
-5.2
-5.3
-5.4
-5.5
-5.6
y(km)
7.8
7.9
8
8.1
8.2
8.3
8.4
8.5
x(km)
11.9
11.8
11.7
11.6
11.5
11.4
11.3
11.2
11.1
depth(km)

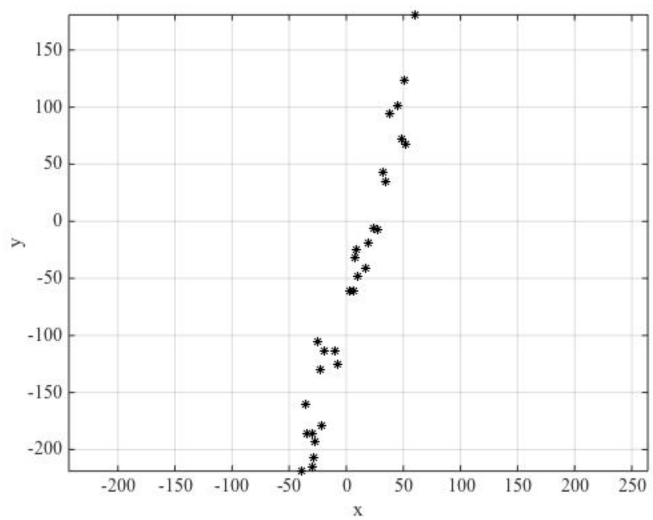
The standard deviations are $\sigma_{x_s} = 0.099670, \sigma_{y_s} = 0.137469, \sigma_{z_s} = 0.149719, \sigma_p = 0.691071$

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.

Bonus points: The exact Hessian is shown in 1(c) .m file.(10 points)

Problem 2 (graded by Yiran) - 50 points

(a) 4 points



(b) 6 points

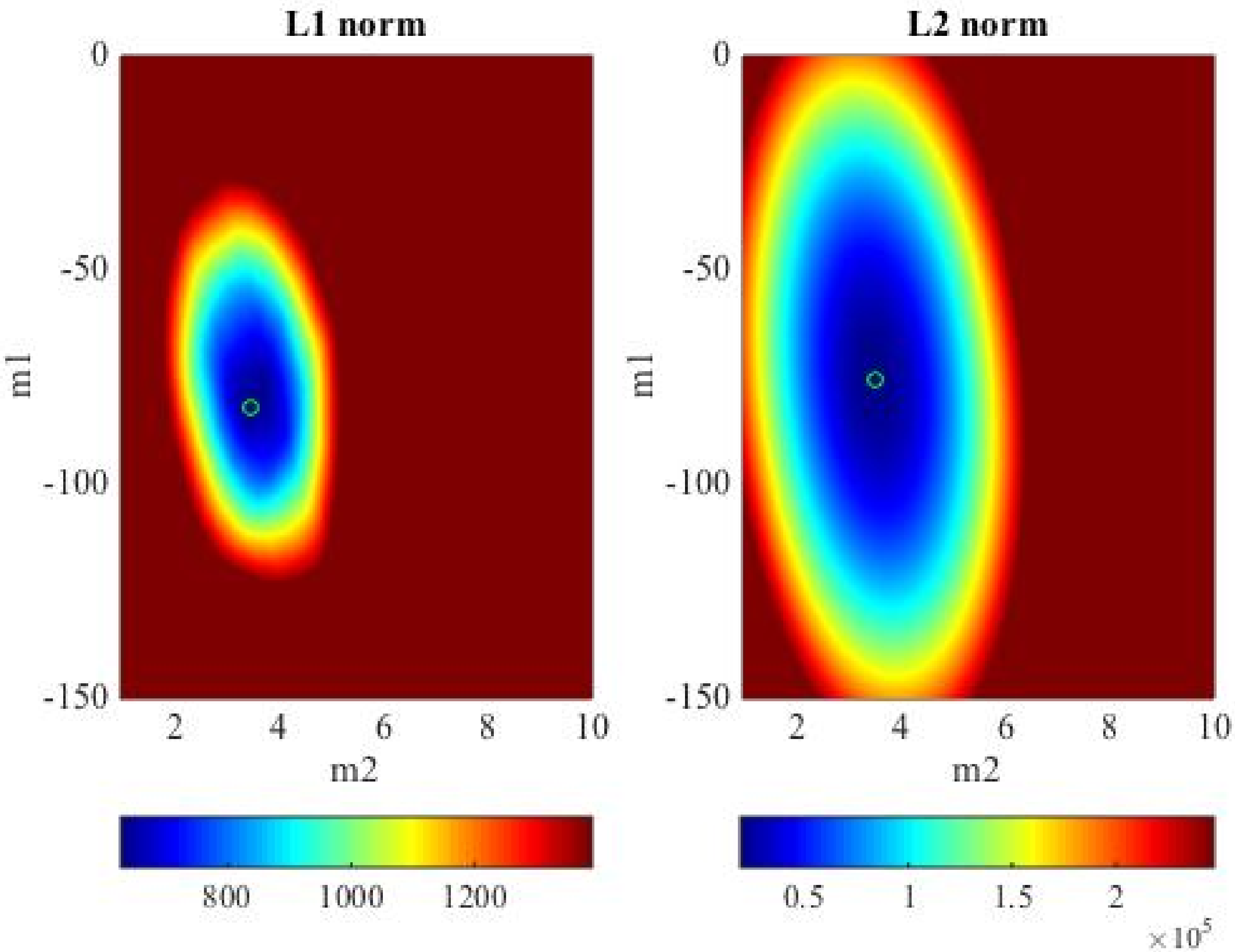
m_1 is the intercept with the y axis. From the plot, we estimate that it should be bounded by $[-150, 0]$.
m_2 is the slope of the line, we also estimate from the plot that it should be bounded by $[1, 10]$.
As suggested in the problem, the arrays are better no larger than a few megabytes (1 double = 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 length(m1) by 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 misifit 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 convariance 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 straighforward 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.

```
function hw2_2
set(0,'defaulttextfontname','times','defaulttextfontsize',14);
set(0,'defaultaxesfontname','times','defaultaxesfontsize',14);

% load data
load ge118_hw2.mat

% plot data
figure(1)
plot(x,y,'k*');
grid on; axis equal;
xlabel('x'); ylabel('y');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% grid search
m1 = -150:0.1:0;
m2 = 1:0.01:10;

% L1 norm
[m1_l1,m2_l1,err_l1,err_all_l1] = grid_search(x,y,m1,m2,1);

% L2 norm
[m1_l2,m2_l2,err_l2,err_all_l2] = grid_search(x,y,m1,m2,2);

% plotting
figure(2)
colormap(jet);
subplot(121)
pcolor(m2,m1,err_all_l1);  % error map
hold on;
plot(m2_l1,m1_l1,'go');    % optimum solution
shading flat;
caxis(crange(err_all_l1));
colorbar('horiz');
xlabel('m2');ylabel('m1');title('L1 norm');
hold off;

subplot(122)
pcolor(m2,m1,err_all_l2);  % error map
hold on;
plot(m2_l2,m1_l2,'go');    % optimum solution
shading flat;
caxis(crange(err_all_l2));
colorbar('horiz');
xlabel('m2');ylabel('m1');title('L2 norm');
hold off;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% least square
[m1_ls,m2_ls] = least_square(x,y);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('L1 norm            L2 norm              LS');
disp('m1');
disp([m1_l1  m1_l2  m1_ls]);
disp('m2');
disp([m2_l1  m2_l2  m2_ls]);

end

% LEAST SQUARE SOLUTION
```

```
function [m1, m2] = least_square(xdata,ydata)
G = [ones(length(xdata),1) xdata(:)];
tmp = (G'*G)^(-1) * G' * ydata;
m1 = tmp(1);
m2 = tmp(2);
end

% GRID SEARCH
function [m1_best,m2_best,err_best,err] = grid_search(xdata,ydata,m1,m2,flag)
% error over the model space
err = zeros(length(m1),length(m2));
for i = 1:length(m1)
    for j = 1:length(m2)
        err(i,j) = misfit(xdata,ydata,m1(i),m2(j),flag);
    end
end

% find the optimum solution
[err_best,id] = min(err(:));
[I,J] = ind2sub(size(err),id);
m1_best = m1(I);
m2_best = m2(J);
end

% CALCULATE THE MISFIT
function err = misfit(xdata,ydata,m1,m2,flag)
ypred = m1 + m2 * xdata;
if flag == 1        % L1 norm
    err = sum(abs(ypred-ydata));
elseif flag == 2    % L2 norm
    err = sum((ypred-ydata).^2);
end
end

% caxis for error plot
function vec = crange(err)
minval = min(err(:));
maxval = minval + 0.15 * (max(err(:)) - minval);
vec = [minval maxval];
end
```