Problem 1 (graded by Toby) 25 points

(a)

In the absence of testable information, our intuition should tell us that the probabilities $p(A)$ and $p(B)$ are equal. Because the total probability is 1, we find

$$p(A) = p(B) = \frac{1}{2}. \tag{1}$$

(b)

THe general formula for choosing M items among N items is given by the binomial coefficient so that the fraction of choices is given by

$$F(M) = \frac{N!}{M!(N-M)!2^N}. \tag{2}$$

(c)

From high school math, we know that the binomial coefficients can be arranged in Pascale's triangle. It is easy to see that the maximum values in Pascale's triange are exacly the center values. Thus, the M that maximizes $F(M)$ is given by $M = N/2$. Thus, we have

$$p = \frac{M}{N} = \frac{1}{2}. \tag{3}$$

(d)

Realizing that $M = pN$, the approximation can be derived by looking at

$$\begin{aligned}
\log(F(M)) &\approx N\log(N) - M\log(M) - (N-M)\log(N-M) - N\log(2) && \text{(4a)}\\
&= -N\log(2) + N\log(N) - Np\log(N) - Np\log(p) && \text{(4b)}\\
&\quad - N(1-p)\log(1-p) - N(1-p)\log(N) && \text{(4c)}\\
&= N\left[-p\log(p) + (1-p)\log(1-p)\right] - N\log(2) && \text{(4d)}\\
&= NS(p) - N\log(2). && \text{(4e)}
\end{aligned}$$

, where we assumed large $N, M, N-M$ (unsing Stirling's approximation for the factorials). Thus, we can maximize S with respect to p instead of maximizing F with respect to M.

(e)

In order to find the maximum, we need to take the first derivative of $S(p)$ and set it to zero. We have

$$S'(p) = -[1 + \log(p) - \frac{1}{1-p} + \frac{p}{1-p} - \log(1-p)] = -\log\left(\frac{p}{1-p}\right) = 0. \tag{5}$$

THis is equivalent to $1 - p = p$, which yields $p = 1/2$.In order for this to be a maximum, we need that $S'' < 0$. The second derivate is given by

$$S''(p) = -\frac{1-p}{p}\frac{1-p+p}{1-p} = -\frac{1}{p} < 0, \ \forall p > 0. \tag{6}$$

Thus, the maximum entropy solution is given by $p = 1/2$.

Problem 2 (graded by Dunzhu) 20 points

(a) - 10 points

The code is shown below. Note that Gauss-Newton is just a special case of Levenberg-Marquardt when $\lambda = 0$.

```
function hw6p2()
xi=[0 10 15 6 -7 3]';
yi=[0 0 6 13 10 7]';
zi=[0 0 0 0 0 0]';
ti=[322.418 321.031 321.228 323.093 324.415 322.706]';
M0=[300 10 -10 10 5]';

lambda=zeros(1000,1);
[M1,err1]=do_one_ti(xi,yi,zi,ti, M0,lambda);
lambda(1:3)=10;
[M2,err2]=do_one_ti(xi,yi,zi,ti, M0,lambda);

% the error contour is ploted assuming ys,zs,v is at the best guess
Mbest = M1(:,end);
perturb_ts = linspace(-30,30,100);
perturb_xs = linspace(-30,30,200);
for k=1:length(perturb_ts)
    for j=1:length(perturb_xs)
        Mnew=Mbest;
        Mnew(1) = Mnew(1) + perturb_ts(k);
        Mnew(2) = Mnew(2) + perturb_xs(j);
        F(k,j) = 0.5*norm(predict(xi,yi,zi,Mnew)-ti)^2;
    end
end

% be careful about x and y axis
contour(Mbest(1) + perturb_ts, Mbest(2)+perturb_xs, F');
hold on;
plot(M1(1,:), M1(2,:),'ro-','linewidth',2,'markersize',7);
plot(M2(1,:), M2(2,:),'bo-','linewidth',2,'markersize',7);
axis equal
legend({'','GN','LM'})
hold off;
print -depsc LM.eps




function [Msave,err]=do_one_ti(xi,yi,zi,ti, M0, lambda)

M=M0;
Msave=M;
for step = 1:10
    grad=zeros(5,1);
```

```
46      hess=zeros(5,5);      % exact hessian
47      hess_app=zeros(5,5); % approximate hessian
48      for i=1:length(xi)
49          [tmp_grad, tmp_hess_app]=get_grad_hessian(xi(i),yi(i),zi(i),ti(i),M)
                ;
50          grad = grad + tmp_grad;
51          hess_app = hess_app + tmp_hess_app;
52      end
53      err(step) = norm(predict(xi,yi,zi,M)-ti)/sqrt(length(xi));
54      M = M - inv(hess_app + lambda(step)*diag(diag(hess_app)) ) *grad;
55
56      Msave=[Msave,M];
57  end
58
59  function [grad,hess_approx] = get_grad_hessian(xi,yi,zi,ti,M)
60  ts=M(1); xs=M(2); ys=M(3); zs=M(4); v=M(5);
61  R=sqrt((xs-xi)^2 + (ys-yi)^2 + (zs-zi)^2);
62  nx=(xs-xi)/R;
63  ny=(ys-yi)/R;
64  nz=(zs-zi)/R;
65  e=(ts+R/v-ti);
66
67  grad = e*[1 nx/v ny/v nz/v -R/v^2]';
68  sen = [1 nx/v ny/v nz/v -R/v^2]';
69  hess_approx = sen*sen'; % approximate hess
70
71  function r=predict(xi,yi,zi,M)
72  ts=M(1); xs=M(2); ys=M(3); zs=M(4); v=M(5);
73  r=xi*0;
74  for i=1:length(xi)
75      R=sqrt((xs-xi(i))^2 + (ys-yi(i))^2 + (zs-zi(i))^2);
76      r(i)=(ts+R/v);
77  end
```

(b) - 10 points

LM method should converge to the same value as GN for this problem. The contour of misfit function is ploted assuming y_s, z_s, v is the same as the best model. Note that

- λ control the length of updating step for LM method. First 3 step ($\lambda = 10$), the updating is smaller compaired with GN method.
- updating becomes smaller and smaller as the gradient goes to 0
- the initial updating for LM is not along the gradient direction even when λ is large. If we use $H + \lambda I$ instead, when λ is large, updating of the model will along the gradient direction

Problem 3 (graded by Stephen) 20 points

(a) - 5 points

For the case of a linear regression model, we know that our G matrices always have the form:

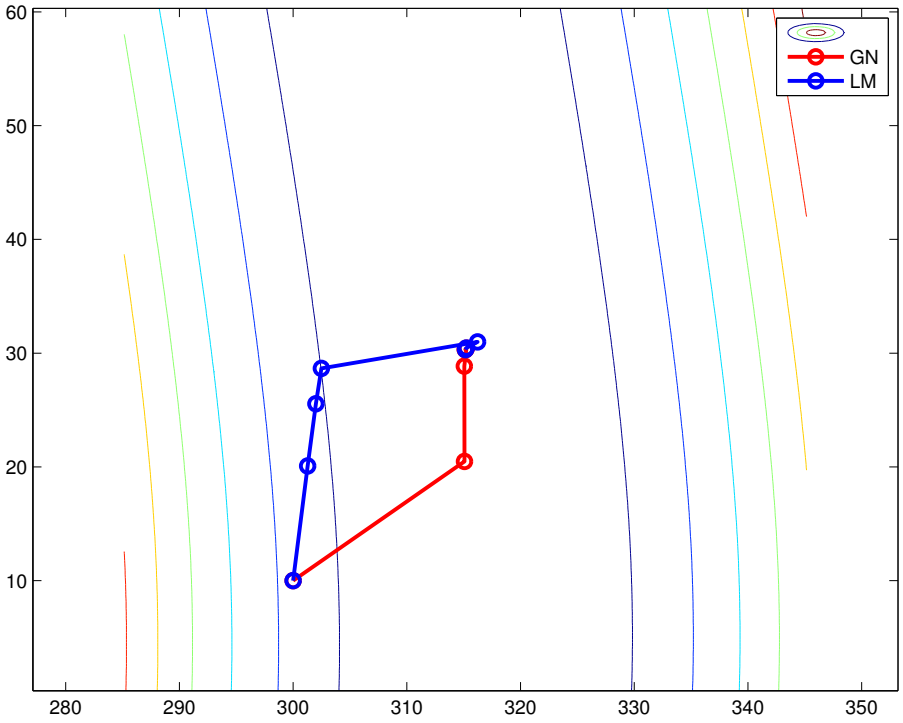

Figure 1: Problem 2

$$G = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \end{pmatrix}$$

Our G matrices only know about where we took the measurements (x-values); they don't know anything about the corresponding y-values at these points. Thus there are infinitely many datasets that can correspond to these G matrices. The points DO NOT need to be on a line, or even nearly linear for that matter, for we can still find a "best fit linear model" for any arbitrary dataset. The only requirement is that the points be at x=1, -3, 4, 5 for G_1, x=-0.1, 0.3, -0.4, 0.5 for G_2, and x= 101, 97, 104, 105 for G_3.

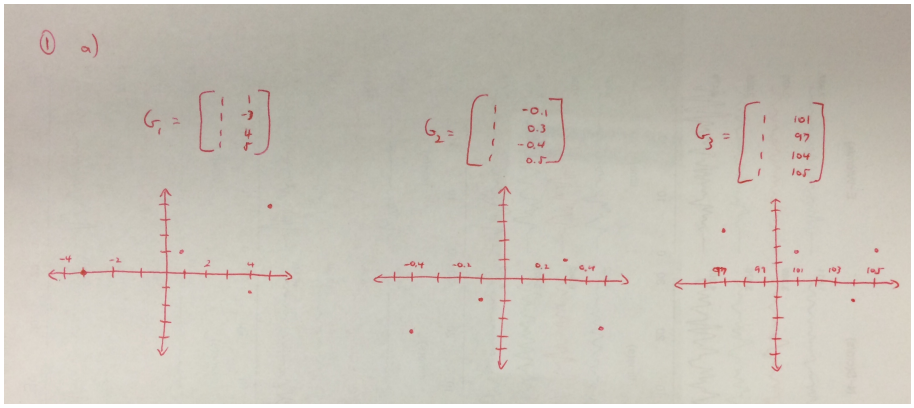

Figure 2: Example Sketch for datasets that fit with each G matrix.

(b) - 5 points

We plug in G_1, G_2, and G_3 into Matlab and simply use the svd function to get the singular value decompostions. See attached matlab code after part c. The outputs are:

$$U_1 = \begin{pmatrix} 0.1572 & 0.4897 & -0.5885 & -0.6239 \\ -0.3916 & 0.8240 & 0.2055 & 0.3543 \\ 0.5688 & 0.2390 & 0.7102 & -0.3390 \\ 0.7060 & 0.1554 & -0.3272 & 0.6086 \end{pmatrix}$$

$$S_1 = \begin{pmatrix} 7.2125 & 0 \\ 0 & 1.7262 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$V_1 = \begin{pmatrix} 0.1442 & 0.9895 \\ 0.9895 & -0.1442 \end{pmatrix}$$

$$U_2 = \begin{pmatrix} -0.4924 & 0.2653 & -0.8186 & -0.1304 \\ -0.5093 & -0.3073 & 0.3239 & -0.7357 \\ -0.4797 & 0.6948 & 0.4738 & 0.2504 \\ -0.5178 & -0.5936 & 0.0210 & 0.6157 \end{pmatrix}$$

$$S_2 = \begin{pmatrix} 2.0064 & 0 \\ 0 & 0.6960 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$V_2 = \begin{pmatrix} -0.9964 & 0.0850 \\ -0.0850 & -0.9964 \end{pmatrix}$$

$$U_3 = \begin{pmatrix} 0.4961 & 0.1357 & -0.5885 & -0.6239 \\ 0.4764 & 0.7780 & 0.2055 & 0.3543 \\ 0.5108 & -0.3460 & 0.7102 & -0.3390 \\ 0.5157 & -0.5066 & -0.3272 & 0.6086 \end{pmatrix}$$

$$S_3 = \begin{pmatrix} 203.6050 & 0 \\ 0 & 0.0611 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$V_3 = \begin{pmatrix} 0.0098 & 1.0000 \\ 1.0000 & -0.0098 \end{pmatrix}$$

(c) - 5 points

We can calculate $G^T G$ for each of the matrices by hand. The results are:

$$G_1^T G_1 = \begin{pmatrix} 4 & 7 \\ 7 & 51 \end{pmatrix}$$

$$G_2^T G_2 = \begin{pmatrix} 4.00 & 0.30 \\ 0.30 & 0.51 \end{pmatrix}$$

$$G_3^T G_3 = \begin{pmatrix} 4 & 407 \\ 407 & 41451 \end{pmatrix}$$

We can compare these to the orthogonal eigendecomposition discussed in class. Here is the form of the orthogonal eigendecomposition:

$$G^T G = V \Sigma^T U^T U \Sigma V^T \tag{7}$$

From this we know that $U^T U$ should multiply out to the identity. We can check this within Matlab for U_1, U_2, and U_3 and see that it checks out. We also want to show that the columns of the V matrix do indeed contain the eigenvectors of $G^T G$ and that the corresponding eigenvalues are the squares of the singular values.

If we use the eig() function in Matlab we can find the eigenvectors and eigenvalues for each $G^T G$. (see attached Matlab code) We compare the first output to the vectors that make up V in each case and see that they match (great!). Then we compare the eigenvalue output to the singular values by taking the square root of the eigenvalues to get the singular values. They also match in each case (fantastic!).

```
EIGVECTORS1 =

   -0.9895    0.1442
    0.1442    0.9895


EIG1 =

    2.9796         0
         0   52.0204


SV1 =

    1.7262         0
         0    7.2125


EIGVECTORS2 =

    0.0850   -0.9964
   -0.9964   -0.0850


EIG2 =

    0.4844         0
         0    4.0256


SV2 =

    0.6960         0
         0    2.0064


EIGVECTORS3 =
```

```
   -1.0000    0.0098
    0.0098    1.0000


EIG3 =

   1.0e+04 *

    0.0000         0
         0    4.1455


SV3 =

    0.0611         0
         0  203.6050
```

(d) - 5 points

We can see that the singular values vary for each of the G matrices from part a. Both singular values for G_1 and G_2 are close to each other. In each case they are only separated by 1 order of magnitude. This is due to the fact that the measurement points (x-values) were separated by the roughly the same order of magnitude as their values. For example, for G_1 the measurement points are on the order of 1 and are separated by just 2 or 3 each. For G_2 the measurement points are on the order of 0.1 and are separated by 0.2 or 0.3 each.

However, the singular values for G_3 show a much greater disparity (4 orders of magnitude). This is due to the fact that the measurement points are on the order of 100 but are only separated by 2 or 3 each (2 orders of magnitude difference). This can lead to difficulties when trying to use G_3 to find a best fitting model because the singular values are so far apart. It would be helpful in this situation to have measured points further apart along the x-axis. A cluster of measurements very near to each other relative to the overall data space can lead to a great disparity in singular values. We can think about this in the extreme case that we measure the same point (x-value) over and over. Due to measurement error, this will return different values. However, the G matrix will have all the same values in the second column. When we calculate G^TG we will see that it's columns are not linearly independent. This means that G^TG is not invertible and will have one singular value that is 0.

In our problem, $G_3^TG_3's$ columns are still linearly independent, but they are much closer to linear dependence than either of the first two cases. Thus, it gives a singular value that is much closer to 0 than the first two cases.

The V matrices are made up of two orthogonal vectors (the eigenvectors in the model space in this case) that make the rotated axes of our new reference frame. Thus, those eigenvectors associated with large singular values are stably determined (well constrained) and those associated with small singular values are not. If we look at the case for G_3 we see that the first singular value is large and the second is not. The eigenvector associated with the small value is:

$$\begin{pmatrix} 1.0000 \\ -0.0098 \end{pmatrix} \tag{8}$$

(This is not really complete...)