Homework Set: Line of Best Fit

1. Review how we constructed the linear program when the error measure was: $E(m,b) = \sum_{i=1}^{n} |y_i - mx_i - b|$. Our set of basic variables was:

$$\mathbf{x} = [\ m,\ b,\ \epsilon_1, \ldots, \epsilon_n,\ a_1,\ \ldots,\ a_n\ ]^T$$

   (a) Rewrite the linear programming problem so that we can remove the set of ϵ's, and the new set of basic variables will be:

$$\mathbf{x} = [\ m,\ b,\ a_1,\ \ldots,\ a_n\ ]^T$$

   Your answer will not require A_{eq} and b_{eq}.

   (b) Change our Matlab code appropriately- This will be a large decrease in the size of our matrix A, so we'll use this version.
   (NOTE: You can double check your answer by trying out both programs on the "toy" data we gave in class).

2. Review the dimensional analysis we did on the length of the pendulum rod to the time it took to complete one "swing" (period). We said that, given data, we could verify our equation. Let us do that now, with the following experimental data:

Swings	Length	Time
133	19	120
118	25	120
80	31.7	100
89	45	120
62	60	100
77	60	120
63	110	120
43	126	100
49	150	120
36	199	100
31	247	100
18	264	60
19	268.5	60

   Some notes before we begin:

   (a) We will need to get *time per period* from this data set.

   (b) Time was measured on a stopwatch, so we will assume that the error we get is all from measuring time. Why do we mention this (was there some assumption we made in setting up all the error measures)?

   (c) You will need to linearize our model before finding the line of best fit. That is, we want to experimentally determine the scaling exponent. You may assume θ plays no role in this model.

   (d) Use all three lines of best fit- Which one seems to work the best?

3. How resistant to change is the slope under each method? Here's a numerical experiment to check. We will select **one** of the y's, and move it up and down (using the value of Δy). We will then check the slope using each of the three methods, and plot the result. In the following code, we make 40 changes to y_1:

```
x=[10 20 30 40 50 60 70 80];
y=[25 70 380 550 610 1220 830 1450];
n=length(x);
DeltaY=linspace(-800,800,40);

for i=1:40
y(1)=25+DeltaY(i);  %Try changing this line-
[xopt2,fopt]=unifbestline(x,y);
A=linreg(x,y)
[xopt1,fopt2]=onenormline(x,y);
%The following are 1-,2- then infinity norms:
Slopes(i,:)=[xopt1(1), A.m, xopt2(1)];
end
plot(Slopes)  %You'll see 3 curves
```

Your code will be different depending on the function names you used. The code as written will move the first data point up and down. Try changing other data points as well! For example, to see what happens if you change the third data point, replace the line following `for i=1:40` to:

```
y(3)=380+DeltaY(i);
```

What conclusions can you draw about the *stability* of the slope computation regarding the position of an "outlier" (the data point that is changing)?

4. In this experiment, we want to see how resistant the slope is to *noise*. One way to quantify this is to see what kind of variation there is in the slope- At the end, we'll compute the standard deviation of our slope computations- Small variation is what we're looking for:

```
%Noise Resistance Example:

x=linspace(-5,5,50); %30 data points
y=3*x-2;  %Perfect Line
n=length(x);
AmtNoise=linspace(0.1,10,40);

for i=1:40
ynoise=y+AmtNoise(i)*randn(size(y));
```

```
[xopt2,fopt]=unifbestline(x,ynoise);
A=linreg2(x,ynoise)
[xopt1,fopt2]=onenormline(x,ynoise);
%The following are 1-,2- then infinity norms:
Slopes(i,:)=[xopt1(1), A.m, xopt2(1)];
end
mean(Slopes)
std(Slopes)
```

5. The final example is very similar to the last one- I'm only changing how the noise is defined- In this case, which is best?

```
%Noise Resistance Example:
x=linspace(-5,5,50); %30 data points
y=3*x-2;  %Perfect Line
n=length(x);
AmtNoise=linspace(0.1,10,40);

for i=1:40
ynoise=y+AmtNoise(i)*(rand(size(y))-0.5);
[xopt2,fopt]=unifbestline(x,ynoise);
A=linreg2(x,ynoise)
[xopt1,fopt2]=onenormline(x,ynoise);
%The following are 1-,2- then infinity norms:
Slopes(i,:)=[xopt1(1), A.m, xopt2(1)];
end
mean(Slopes)
std(Slopes)
```

6. What kinds of conclusions can we draw from these experiments? In what situations would one choice of error be better than the others?

A note about *noise*: Matlab gives two random number generators- `rand` and `randn`. The difference is that `rand` draws randomly from the interval $[0, 1]$, with a *uniform distribution*. The other generator, `randn`, draws numbers from a *normal distribution* with mean zero and standard deviation of 1.