BOB, Optimization, Gradient Descent, Error, and Uncertainty

C. Alex Simpkins

November 16, 2006

1 General Homework Description

Now that you can load data into matlab, visualize the basic information, and create simple mathematical models, you need to be able to evaluate those models. In lecture we've discussed the concept of error analysis and uncertainty. We've also discussed optimization within the context of minimizing cost. Those two concepts will be the focus of this assignment.

You will perform nonlinear function minimization on data to determine the best behavioral model for a new life form.

You will then answer a few questions regarding your results and how they tie into the four steps of modeling as laid out in this course.

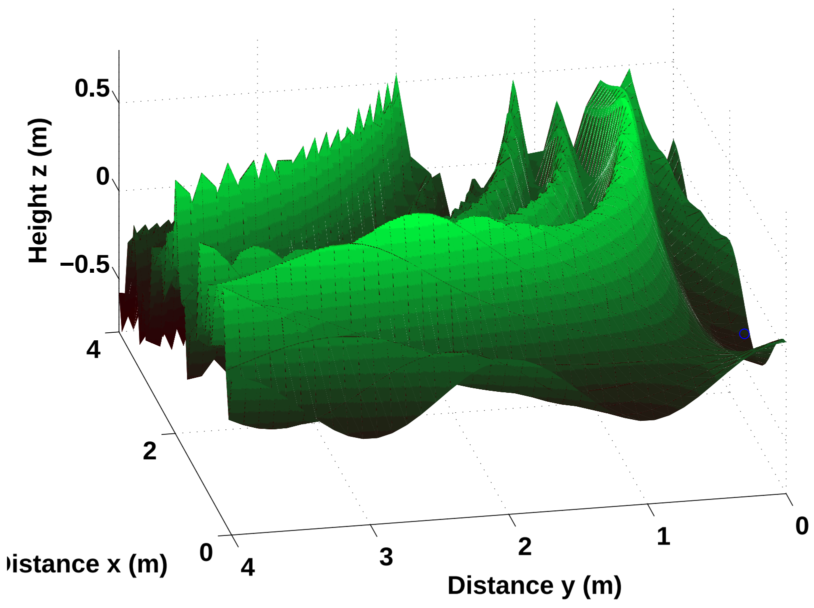
Please read this entire document before proceeding. This is a long document because the programming steps have been extensively documented and most of them given to you directly. So if you follow along with matlab and this pdf open simultaneously, you should hopefully find the programming part fairly straightforward. The questions are designed to help you think about the material and come up with meaningful answers.

2 Background Description

Optimality can be the factor which governs many behaviors of creatures with even primitive minds. In this case you and your team have discovered a new species of life. It has not been classified yet (it is difficult to determine if it is an insect or mammal, possessing characteristics of both, thus it defies current classifications). The creature has been named B.O.B. (this stands for Brilliantly Optimal Brain, since almost ALL its currently observed behaviors have been found to be optimal in some sense). BOB is easy to experiment with, since he (or she) loves food (especially pizza), is completely complacent and friendly with experimenters. As opposed to rats and monkeys often used in cognitive experiments, BOB smells pleasant.

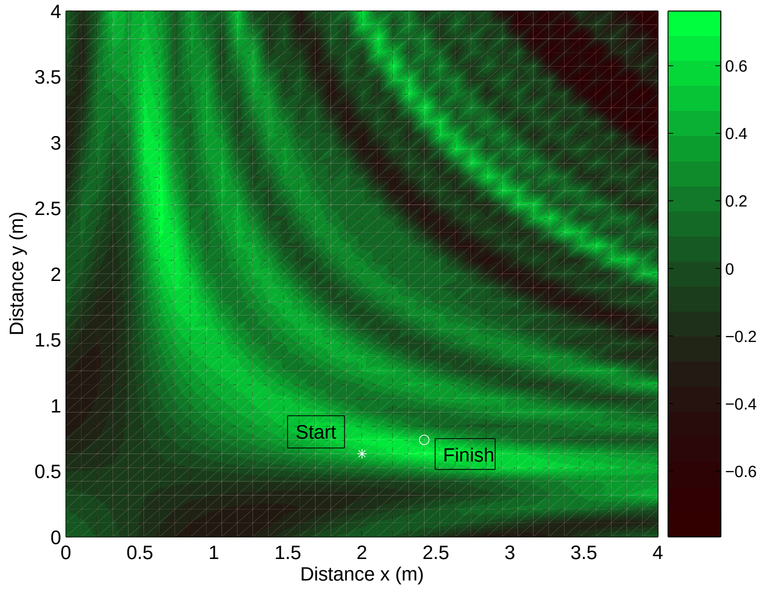
3 The problem

We know BOB is optimal (or we assume this is likely given our prior knowledge of BOB from experiments) in his behavior. The question is what is BOB optimizing? You have devised and executed a simple experiment with the following 3d terrain map which was created for BOB in the new rapid prototyping machine lab in the cognitive science department.



The experiment consists of placing food on the terrain at a particular location (2.4, 0.7, 0.5), and placing BOB on the terrain map at approximately the same altitude (height) but a different location. BOB excitedly advances to the food. You record BOB's movements through motion capture (CALIT2 just purchased a $400k motion capture system that records movements with over 20 cameras which identify and track infrared dots in 3D space, then output position coordinates to you as a matrix of numbers at a sample rate of

5kHz) .



4 Modeling the movement

4.1 Description

The shape of the movement of BOB depends highly on what is being optimized. If he is optimizing time to get to food, no matter what the effort to get there, you will see one kind of function fit more precisely. If he is optimizing energy expenditure to get to the food (and it takes much more energy for him to change his altitude - i.e. climb), his movement will have a very different shape, because he will go to great lengths to avoid having to climb the mountain.

4.2 Load the data - 20 points

Download **hw4.zip** and load it into the matlab workspace. You will have one three dimensional variable, `BOBposition` and another group which consist of x, y points and heights H representing a terrain map. The data variable called `BOBposition` has rows corresponding to observations and columns to variables (x, y, z). The terrain map variables x, y, and H contains the x, y, and H coordinates of the terrain map BOB is navigating. Take a moment to familiarize yourself with the variables, their sizes, etc.

4.3 Plot the raw data - 20 points

Create a plot in 3d of the data on top of the terrainmap. You may use any colormap you wish, or use contours so that it is easier to print this assignment in black and white. Merely make your choice so the information is clearly conveyed.

To plot the terrain data, use the `surf` command in matlab. It takes as input the locations of the x coordinates, y coordinates and height values at each coordinate. The tricky part is that you need to generate an x-y mesh of points. Why? Because you have a height value at every x and y point, so you need an x for every y, and a y for every x. The actual implementation of this mesh operation is simple in matlab:

```
[X,Y] = meshgrid(x,y);
```

Go to the command line, and now that you have the variables, look at their size (you can use the `whos` command to list all your variables, or you can use `size(X)` and `size(Y)`) Now you can use X and Y when you apply the surf command:

```
%Plot the terrain in 3d as a surface...

surf(X, Y, H);

% I recommend this smoothing since that is a closer model to the true terrain...

shading interp;
```

Now plot the start and finish points on the same plot. They are given in the description above. Use a star, circle, or other shape (built into matlab) to make the points more visible. Also use an appropriate color to provide the best contrast possible.

You can plot a single point in 3d much the same as you would in 2d by using the following command:

```
plot3(xpoint, ypoint, zpoint, 'r*');
```

4.4 Set up the Problem and Perform the Minimization - 20 points

4.4.1 Description

You have identified two curves to fit the data with, depending on how BOB moved. He either moves straight toward the food, regardless of the cost of climbing the mountain, or goes around the incline in a very nonlinear (in x and y) curve. The two candidate models for his movement data are:

Number one:

$$x = a * cos(\theta) * t + b \tag{1}$$

$$y = c * sin(\theta) * t + d \tag{2}$$

$$z = f(x(t), y(t)) \tag{3}$$

With θ b, c, d and a constants, and z merely a function of x and y, determined by the terrain, changing as rapidly as we like (i.e. z changes not constrained except by terrain).

Number two:

$$x = a + b * cos(t * \pi * c) + d * t + exp(-f * t) \tag{4}$$

$$y = f + g * sin(t * \pi * h) + k * t + exp(-j * t) \tag{5}$$

$$z = m + n * sin(t * \pi * p) + w * t + exp(-s * t) \tag{6}$$

$$dz_i = |z_{i-1} - f(x(t_i), y(t_i))| < \epsilon \tag{7}$$

With $\sum dz$ minimized (i.e. we change z as little as possible to avoid climbing the mountain at all). Epsilon is an arbitrary constant number that is small.

The second equation group is nonlinear in the parameters, and the first can be linearized ($sin(\theta)$ and $cos(\theta)$ are essentially 'constants.' If confusing, think of this as 5 times 1 = 5. The sine of 90 degrees is always 1, so we can define some new constants $r = sin(\theta)$ since θ is constant and so on).

4.4.2 Fit the first model to the data using gradient descent - 10 points

Fit the first (linear) model to the data using the gradient descent algorithm. We need to set up the required parameters and run the code.

Start by beginning a new m-file in matlab now. Call it something meaningful such as `BOBModel.m`

You need to either have the data variables in memory (ie BOB's movement data `BOBposition`), or paste it into your matlab gradient descent code.

You also need the start and end positions for BOB (where he is started, and where his food is):

```
start = [2, .6316,.5163];
```

```
location = [2.421, .7368, .5338];
```

The time steps are 10 seconds for this data (you've subsampled, since BOB is somewhat slow, and you just want to get a sense of the model set first).

```
 % time steps are 10 seconds...
```

```
dt = 10;
```

In other words, each data point is sampled every 10 seconds (you could think about this question but are not required to write about it - what would a sample every 10 seconds be as a sample rate be in Hertz?).

We'll also need to know how many points we have, which is accomplished with the following...

```
%number of data points...
```

```
n = length(BOBposition(:,1));
```

And now let's realize we're going to fit the x, y, and z components as parametric equations as a function of time. This way we'll see how x, y, and z change as the evolution of BOB's movement proceeds. The other advantage of creating the time array as follows is if we in the future have more points, we can just change the data, and n is recalculated, as the time array is, and very little effort is required on our parts...

```
%create the time array...
```

```
t = 0:(n*dt/(n-1)):(n*dt);
```

Now we need to do the final setup for our specific problem, which is to set up the A and b matrices exactly like a least squares fit, with time as the independent (x) variable, and the first row of the data as the dependent variable (we'll need 3 equations, one for x, one for y, one for z, so we'll do this one at a time).

```
%set up the A matrix and b vector of our Ax=b equation...
```

```
A = [ones(size(BOBposition(:,1))) t'];
```

```
b = BOBposition(:,1);
```

We'll do a linear fit, just like least squares, but using gradient descent. But before we can, for a gradient descent algorithm to work on an over-determined system (ie more data points than unknown parameters) we need to transform the system into an equivalent one which is not over-constrained (a perfectly constrained system)...

```
%deal with the overdetermined aspect of this system...
```

```
%it is overconstrained since we have more data points than parameters...
```

```
%we will convert to a properly constrained system by converting to the
```

```
%equivalent A'A*x = A'*b system...solve that for x and it is the same

%solution as for Ax=b, but works with gradient descent!

b = A'*b;

A = A'*A;
```

Now we can take the rest of the gradient descent code right from class...

```
%now we can do gradient descent...

clear res_save x_save x;

epsilon = 1e-6; %tolerance for the change in the cost

itmax = 1000; %max possible iterations

x=zeros(size(b)); %initial guess for our parameters

%figure(1);

for iter=1:itmax %keep iterating until we hit a criterion or max iterations

 r=(b-A*x); %compute the residual difference between approx and system

 res=r'*r; %dot product of the residual components to get one number

 res_save(iter)=res; %save this for later plots

 x_save(:,iter)=x; %save our estimate for later plots

 if(res<epsilon | iter==itmax),break; end %stop iterating if these conditions
are met

 alpha=res/(r'*A*r); %compute the magnitude of the jump to the next point

 x=x+alpha*r; %increment our approximation to x

end

%-------end of gradient descent code for fitting x--------

%----now you need to do the same thing for y, just change the data input to
b------
```

4.4.3 BONUS - 10 points

Compute a least squares linear fit, and compare the parameters (in a table made in word or a similar program). Why are they so similar (briefly)?

4.4.4 Fit the second model to the data using fminsearch - 10 points

Fit the second model using the matlab `fminsearch()` function. Here's how...

Create another m-file, called `bobnonlinearfunction.m` . This will contain our function for matlab's `fminsearch` function to use for optimization.

It must begin with the following, since it is a function, and these are the parameters we will be using...

```
function err = bobnonlinearfunction(const,t,x,handle)
```

The function we will fit is the following...

```
% the function is x = a + b*sin(c*t) +t*d + exp(-f*t)
```

where a, b, c, d, and f are constants to be determined by our algorithm

```
xp= ones(length(t),1)*const(1) + const(2)*sin(const(3)*t') + t'*const(4)+exp(-const(5)*t');
```

The function will be returning the error between model and data, defined just as the last homework...

```
err = norm(xp-x);
```

and that is the end of the function. Now create another m-file which will be your main nonlinear BOBfit function. Give it a new name such as `NLBOBFit.m` or whatever you deem appropriate.

It will start just like the linear fit gradient descent code:

You need to either have the data variables in memory (ie BOB's movement data `BOBposition`), or paste it into your matlab fminsearch NLBOBFit.m code.

You also need the start and end positions for BOB (where he is started, and where his food is):

```
location = [2, .6316,.5163];
```

```
start = [2.421, .7368, .5338];
```

The time steps are 10 seconds for this data (you've subsampled, since BOB is somewhat slow, and you just want to get a sense of the model set first).

```
 % time steps are 10 seconds...
```

```
dt = 10;
```

In other words, each data point is sampled every 10 seconds (you could think about this question but are not required to write about it - what would a sample every 10 seconds be as a sample rate be in Hertz?).

We'll also need to know how many points we have, which is accomplished with the following...

```
%number of data points...

n = length(BOBposition(:,1));
```

And now let's realize we're going to fit the x, y, and z components as parametric equations as a function of time. This way we'll see how x, y, and z change as the evolution of BOB's movement proceeds. The other advantage of creating the time array as follows is if we in the future have more points, we can just change the data, and n is recalculated, as the time array is, and very little effort is required on our parts...

```
%create the time array...

t = 0:(n*dt/(n-1)):(n*dt);
```

Now this code is a local minima, and since we are no longer dealing with a quadratic cost, we may not find the global minimum. That may be why the fit comes out not so great at times. There may also be terms which are neglected or chosen incorrectly, and significant noise in the measurements. This is only a suggestion for an equation. It works fine for a rough fit to learn what you need, so we should be able to answer the questions with the following equation fit.

Here is a moderately good initial guess for the parameters to be, for the xdata...

```
%initialguess = [ 1.0230 -0.4819 -0.4554 -0.0178 -0.0122 -5.3567]';
```

and for the y data...

```
initialguess = [ -0.2205 -0.1363 -0.3798 0.0023 -0.0000 -8.5763]';
```

Now we're going to give matlab's function the above initial guess, then some options, which we will store in a variable. Finally we will make the function call and pass in the name of the function which has our equation.

```
options = optimset('TolX', 1e-6,'TolFun', 1e-6 ); % termination tolerance on
x, and the error
```

see the matlab help for details about the outputs, but the most important thing is the estimated_C term is the array of parameters we are finding. You will also need the error term. That is for you to find which of these returns the error. See the matlab help on fminsearch.

```
[estimated_C, n,m, output ]= fminsearch(@(abcdf) bobnonlinearfunction( abcdf,
t, y ), initialguess, options)
```

After that, it's handy to immediately plot the results vs. our actual data to view the fit visually...

```
clear xp

xp=estimated_C(1) + estimated_C(2)*sin(estimated_C(3)*t) + estimated_C(4)*t +
exp(-estimated_C(5)*t);

plot(xp)

 hold on; plot(BOBposition(:,2), 'o')

legend('Function fit', 'Actual data samples')

% End of fminsearch code.  Now you just need to do this for BOBposition(:,1)
and BOBposition(:,3)
```

4.5 Question - 20 points

Compare your two models by computing a norm-based error, the same way as the last homework with a slight adjustment, since now we want to see how BOB's x and y coordinates of his output are fitted with the model. We just add the error from the x part of the fit to the error from the y part of the fit.

```
E = norm(ydata - ymodel) + norm(xdata - xmodel);
```

Where all the above variables are nx1 vectors.

Do this for the linear model, and record the values given to you by the fminsearch function, it is computing a normed based error already, and there's no sense repeating it. You just need to add the error for the x data and y data

Which model has more error as compared to the recorded data, given this definition of error (compare the numerical results of the above calculations)?

4.6 Question - 20 points

What is BOB more likely to be optimizing (energy or time to target)? Why do you think this? Support your decision with your model results and error analysis (compare what the cost of his actions would be for the two different given cost concepts - i.e. consider which equation fits the data better, and what cost function that equation minimizes - time or energy, though you don't have either cost function as an explicit equation).