Newton Raphson

The first problem I explored was a simple implementation of the Newton-Raphson method. I used C code for the algorithm and tested its convergence using different input and different polynomial functions.

I used the Numerical Recipes in C book to implement this approach, however, I remember that it did not converge correctly if you shifted the polynomial. Either I transcribed a bug, which is possible (but I double and triple checked my code), or their code is not very robust. I would not recommend using this approach unless you desire an exercise in reverse engineering code and then adding comments. It does provide a good starting point, but it does not provide much detail. Also, classes and libraries developed for this very problem are likely to have been more rigorously tested and thus useful.

Simulated Annealing

The second problem I explored was a simulated annealing problem. Again, I used C code in combination with OpenGL to display my results. Once more, I used the Numerical Recipes in C textbook to begin. I would not recommend using C and OpenGL for this. Processing.org's Processing application would make visual verification much easier -- possibly slower though.

I *would recommend the text book* for its description of the process. Most importantly, it gives a good idea of how to combine the standard annealing with "objectives." (Note: this is not done in code, but rather the description of the energy function) Also, the description is concise with a good allusion to the annealing of crystals.

The one drawback to using the code in the text is that it is not well commented. It was difficult to decipher the progression of the algorithm through the code.

Example application of SA:
The book uses the example of the Traveling Salesman Problem. It also introduces an objective on top of the basic energy function:
There is a continuous constraint path, may be linear or nonlinear, to avoid (or hit as much as possible) based on its effect with respect to the energy function.

$$E = \sum_{i=1}^{N} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}$$

The energy function to minimize without additional constraints.

$$E = \sum_{i=1}^{N} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} + \lambda_1 + \lambda_2 + \ldots + \lambda_n$$

The energy function to minimize with regard to *n* additional constraints.

x and y are Euclidean coordinates. Other λ objectives can be added by introducing new metrics associated with each coordinate or as some function of the euclidean coordinates. For instance, in the example in the book, each coordinate was assigned a 1 or -1 depending on if it was east or west of a "river." The λ objective reduced E or raised E based on this value.