Homework 12 , minimization

Part A and B and C done.

Just for fun, part A and part C also prints a 3D figure with all the steps taken. This is not part of the exercise, but I used it for debugging and decided to leave it in.

PART A

Output: OutA.txt writes what the output of the minimizations were, including whether or not they were in error. The png files shows the optimization process

In part A I implement Quasi-Newton minimization using numerically calculated derivatives and.

I test it to find maximum of a 1 dimensional gaussian $exp(-(x-2.0)^2)$ (with max x=2.0), minimum of Rosenbrocks 2D valley function (with minimum set to 1,1), and the two-dimensional Himmelblau function (4 possible minima, I find the one I start closest to).

You will see in my out file where I start my guess, and what my precision is.

You can run the program with the flag -v if you want a detailed breakdown

PART B

Output: OutB.txt write the parameters were found. The png files shows the data and the fit.

I do not know if B is counts as done, because I do not get exactly the "true" result, my result is "close" (it is mass being 125.9 GeV/c^2 while it truly was 125.3 GeV/c^2)

I do still like my result, because I tried plotting the data against my fit, and it looks close enough for me to accept it. (Do check out that figure)

Though, Do note that the function to minimize had a lot of local minima around, so I had to do a start guess at m=121 GeV/c^2 to avoid them. This is chosen based on the plot of the data, as it appears to be aroudn there the peak is.

PART C

Output: OutC.txt writes what the output of the minimizations were, including whether or not they were in error. The png files shows the optimization process

In part C I implement the downhill simplex method. I test it on the same examples as in A

In general it takes more steps to find the minima than the newton method, but I can always guarantee that the lowest point is within the tolerance distance of the minimum, something I could not do with the gradient based method, where I only could guarantee that the gradient was below something (technically the gradient based method could find a saddle point, though that would be extremely rare).

This method is, in principle, also more stable; there were some cases where I automatically restart the gradient based method (where finding the gradient fails, or where I get 0-divisions) though these cases should be fairly rare, and all of this is automatically handled by the algorithm behind the scene. So it is not a significant issue

Again, feel free to check out the generated png figures, where I show how the triangular mesh'es of all my vertices.