# Homework 1

Maggie Isaacson

October 31, 2024

## Problem 1

My github repository is found here.

## Problem 2

### Comparison

For this simple function, the solutions and times are found in table 1 below.

When using the same number of pulls of the MC as points in the quadrature methods, the quadrature methods perform better. However, it would be very easy to increase the accuracy of the Monte Carlo method by upping the number of pulls without increasing the time very much. For more difficult functions with more inputs, the quadrature methods become more costly in terms of time as the function needs to be computed multiple times.

## Problem 3

### Comparison

The solutions and times are found in table 2 below.

While all 4 methods found the correct solution, conjugate descent proved to be the fastest in terms of time while Newton-Raphson required the smallest number of iterations. However, in calculating each of these, I was able to create a function to find the gradient and the hessian without needing to use any of the derivation methods. If there wasn't an analytical solution for the derivatives, it is likely that either BFGS or Newton-Raphson would be faster, as the number of numerical derivatives would be much smaller.

## Problem 4

Using the fmincon function for 3 agents and 3 goods is fairly successful, although some values equaling zero can cause problems. Most of the time, it is able to find a solution. However, with 10 agents and 10 goods, using the same function, is far more sensitive. With deeply heterogeneous endowments, the algorithm can rarely find its way to the appropriate equal allocation (in the simplest case with equal weights and parameters).

| Method | Solution | Time |
|---|---|---|
| Midpoint | -18.2095251513542 | 0.0057 |
| Trapezoid | -18.2094487366169 | 0.0022 |
| Simpson's Rule | -18.2095253999815 | 0.0029 |
| Monte Carlo | -17.6166773287535 | 0.2523 |

Table 1: Caption

| Method | Solution | Iterations | Time |
|---|---|---|---|
| Steepest Descent | (1,1) | 50000 | 0.0799 |
| Conjugate Descent | (1,1) | 2917 | 0.0056 |
| Newton-Raphson | (1,1) | 6 | 0.1766 |
| BFGS | (1,1) | 35 | 0.0082 |

Table 2: Caption

# Problem 6

## 6.1

The recursive social planner's problem for this set-up is as follows

$$V(k,\tau,i_{-1},z) = \max_{i,c,k',g,\ell} log[c] + 0.2log(g) - \frac{\ell^2}{2} + \beta \sum_{T} \pi(\tau'|\tau) \sum_{Z} \pi(z'|z) V(k',\tau',i,z')$$

$$s.t.$$

$$k' = 0.9k + [1 - 0.05(\frac{i}{i_{-1}} - 1)^2]i$$

$$c + i + g = e^z k^{0.33} l^{0.67}$$

## 6.3

After successfully completing the steady state calculations, I attempted to complete the value function iteration section. I did not finish- my loop for the VFI finds reasonable labor and capital. However, I could not find a way to aggregate over the probabilities of tau and z without causing the loop to run even more slowly than it already does. After working with it for awhile, I decided to work on a stripped down version to get the capital-investment connection and the value function iteration concepts correctly. The deterministic, exogenous labor version is also in the repository. I understand the basic premise of value function iteration, but executing it efficiently and correctly on problems that are more complicated than the Neoclassical Growth Model is difficult. Thus, I'm working through the simplified problem using code from a classmate that has more coding experience. I'm having some issues with the investment policy function, but overall, the code runs nicely. I hope to start adding the uncertainty soon.

I am struggling to get the linear interpolation functions to work correctly, as well as which MATLAB functions are best to use in the loop of the problem. The theory of how to complete VFI is not the problem so much as the underlying MATLAB coding is the problem.