Eleventh exercise class

Class 5

Introduction to numerical programming and analysis

Asker Nygaard Christensen

Spring 2021

UNIVERSITY OF COPENHAGEN

Plan

1. My notes on the model project

2. Notes on problem set 7

My notes on the model project

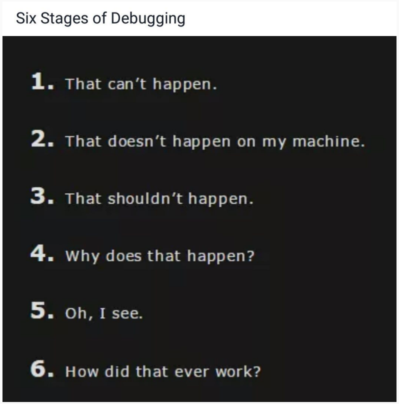
Model project

- The deadline is firm at Friday, the 14th of May. There won't be time for re-submitting, because I have to tell the administration whether you're eligible for the exam.
- You don't have to make a model from scratch, but you do have to be a bit creative. You're also welcome to take inspiration from former groups, since they are all available at Github
- If you already have a complex model, your extension doesn't necessarily have to be revolutionising. It will typically be easy to re-solve the model with a different production or utility function.
- If you're not certain how to solve the model then start out by brute-forcing it: simulating a model will likely be slow, but should give you the right answer.
- Motivation: Two great projects will be a good safety net. They can compensate for a disappointing exam project.

Notes on problem set 7

General

These problems are really complex, and also a step op from last week.
Small mistakes and errors will occur, so have patience!
However they are also very relevant for the exam and could give you the tools you need for the model project.


Six Stages of Debugging

1. That can’t happen.
2. That doesn’t happen on my machine.
3. That shouldn’t happen.
4. Why does that happen?
5. Oh, I see.
6. How did that ever work?

Optimization problem I

- Question a: I would recommend taking a look at the 3D plot and contour plot part of section 2 in lecture 11. And for the contour plot: it's not important to guess/figure out the same levels as the answer, but the way it is done is quite smart.
- Question b: Depending on your version of *scipy*, next question might only work if these function return numpy arrays. And the Hessian needs to be a 2×2-array of the form: *np.array([[f11,f12],[f21,f21]])*.

The mathematical formulations:

$$J\left(f\left(x_1, x_2\right)\right) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

$$H\left(f\left(x_1, x_2\right)\right) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} \end{bmatrix}$$

Optimization problem I -c

- Question c: The programming is almost identical to section 2.1 in lecture 11. But see the documentation for the different optimizers and how to use them.
  Choosing an optimal optimizer is very much problem-dependent. Using analytical gradients typically saves computing power (Fewer iterations and evaluations), but of course calculating them analytically either requires some computation in sympy or by yourself, so you have to take that into account. Unless I'm doing something that's computationally very heavy, my main concern when choosing an optimizer is whether or not I need bounds and constraints. In the introduction to lecture 11 Jeppe has a nice summary of pros and cons on the different scipy optimizers, where he also lists which optimizers handles bounds and constraints.
  In lecture 14 you'll learn how to time the computing time of your functions, allowing you to test yourself, which optimizer is ideal for your specific problem.

Optimization Problem II

- Question A: Can be done mostly by reusing code from Optimization problem I, and then looping trough the different starting points marking the optimization if it is better than the previous best. You'll also need to store all optimizations and results. The answer uses *'BFGS'*, but you can use any method you prefer, which will give you slightly different answers.
- Question B: 3D scatter plots are a lot like plotting surfaces like you did in Optimization problem I, but instead of:
  cs = ax.plot_surface(x1_grid,x2_grid,f_grid,cmap=cm.jet)
  and the lines creating the grids, just write this:
  cs = ax.scatter(xs[:,0],xs[:,1],fs,c=fs)
  (So it's all 1-Dimensional input), the *c* is for scaling the color.
- For question C you're just plotting x0s instead of xs.

Solve the consumer problem with income risk I

- Question A: In this problem you are given the functions for solving the problem, but read them thoroughly as you'll need to be able to use them correctly.
  You also need to create the *v2_interp* argument using the scipy.interpolate function RegularGridInterpolator, by solving period 2 first. You can see an example of this in the last section of lecture 11.
- Question B: You'll need to define a new *v1*-function.
  Interpretation: Higher risk in future periods increases savings.

Solve the consumer problem with income risk II

Depending on the power of your computer, the *solve_period_2()*-function might run a bit slowly, you can turn down precision (replace 200 with 100 for example) if it becomes a problem.
The RegularGridInterpolator works for functions with multiple arguments also. If you are unsure of the syntax for that, I think the best way to learn it is to look at the answer. The x-values are a list of the two arguments as vectors, while the function-values should be in a grid format.
Also, it isn't explicitly noted, but $d_1 = d_2$, as it is durable consumption with no depreciation.
If you're using scipy, you might get the warning: *RuntimeWarning: Values in x were outside bounds during a minimize step, clipping to bounds warnings.warn("Values in x were outside bounds during a "*
This seems to be an error in the scipy package, so don't worry to much about it.
My version of the extra problem (and all the others) is on Github