UNIVERSITY OF COPENHAGEN

Fourth exercise class

Class 5

Introduction to numerical programming and analysis

Asker Nygaard Christensen

Spring 2021

Plan

1. 2.1-2.2 shown in class
2. Problem 2.3-2.5
3. Problem 2.6
4. Problem 3
5. Next time and the inaugural Project

2.1-2.2 shown in class

Notes for problem 2.1

This problem set is centered around lecture 4 and random numbers.
If you want to see my take on the problem sets, they are here.
I'll quickly show you problem 2.1-2.2 in class, since they are not that important.

Problem 2.1:
You need the *np.random.get_state()* and *np.random.set_state()* explained in lecture 4 section 3.2

Notes for problem 2.2

Use numpy, it has functions for calculating means and variance.
Think in steps:

- Create an x-array that contains elements pulled from a normal distribution
- Create an array that contains the elements of the output of the g-function. You have two options: either create a g-function that uses arrays as input and output and call it on the entire x-array, or one that takes floats as input/outputs and then call it on each element of x (bit-wise))
- Use numpy functions to calculate the statistics. (You can also write the equations to calculate them 'manually' if you wish, but *np.mean()* and *np.var()* are easier)

Another thing: I don't know why the mean is subtracted when calculating the variance in the solution, but it does not change the answer

Problem 2.3-2.5

Problem 2.3

The first missing line is that you need to import the function *norm* from the *scipy.stats*-module. (documentation here) normally I use np.random.normal, but scipy.stats, as you can see in the documentation, also has a bunch of distributions. And some different methods (e.g. *.ppf()*)
For the next two missing lines, look at lecture 4 section 3.4, point: b. vector of x values
For widgets look at section 5, case 1, in lecture 4. Then notice that the *fitting_normal()*-function in the PS, can be called in a similar fashion as the *interactive_figure()*-function in the lecture note.

Problem 2.4-2.5

2.4

You can generate any data you want to, it does not need to be the same as Jeppe (it would be impossible to guess), the my_np_data just needs to be a numpy array (although lists actually also work).
For saving files using *pickle* and *numpy* look at section 8.1 and 8.2 in lecture 4

2.5

After defining the new function, it is possible you have to restart your kernel in jupyter lab for it to work, if you have not run the:
%load_ext autoreload
%autoreload 2
-cell, in the beginning of the notebook.

PS 2.3-2.5 DYI

We'll meet back at 16:15 after the break

If you don't mind me visiting you randomly, make sure to open a video meeting, then I'll move between channels when there are no questions.

Write me if you want me to open a group channel, then ones I have made so far are based on the sign-up sheet for the inaugural project (BTW make sure you fill that out). At 16:15 we'll meet back here, and go through the Github exercise together.

Problem 2.6

Problem 2.6, Github exercise

We go through this together. Last time I made a mistake when showing you how forking works, I apologise.

Normal cloning: Follow the guide

For forking: Go to the repository, press the 'fork'-button. Now a new repository will show up which is saved under your Github account, you then clone this repository (see guide from section 2).

Then you need to tell VS code, that this repository should accept changes from the master repository. This is done by typing:

Ctrl+Shift+p+'>Git: Add remote...'

Then pasting in the url of the master repository+Enter.

Then typing any name is valid (e.g. 'Upstream')+ Enter.

Now to load master changes, you type Ctrl+Shift+p+'>Git: Pull From...', then choosing the 'Upstream'.

See here for a explanation of why.

Problem 3

Problem 3

- $f\left(x,\beta_i\right)=\frac{1}{\beta_i}\exp\left(\frac{-x}{\beta_i}\right)$ refers to the PDF of the exponential distribution. So you'll need *np.random.exponential(beta1,size=N)* (and also for *beta2*).
- When restricting bounds on α you can draw inspiration from problem 2.1
- For the demand function you are allowed to use the the Cobb-Douglas rule (demand for good 1 is $\alpha_1 \cdot \frac{Income}{p_1}$).
- You can normalise p_2 to numeraire ($p_2 = 1$), as it is only the relative price that matters.
- Use Walras' law so you only have to clear the market for one good. The *find_equilibrium*-function, which is hinted only needs a few changes to apply to this problem.

Next time and the inaugural Project

Inaugural Project

Programming can be frustrating. Remember to take breaks when encountering frustrating bugs. Fresh eyes are much better a spotting bugs, this applies both to asking a friend, but also for taking breaks. You'll have time to do it in class next week under my supervision
Don't forget to organise your group and register in group excel file (In MS Teams: UCPH_Lectures - Introduction to Programming → 'General'-channel → Files → Groups.xlsx).

Sources of inspiration for the inaugural project:

The deadline for the inaugural problem is 19th of March.

1. It's okay to hand-in something where the final results wrong, but you've been unable to fix it. Just make sure to document it, and I'll help you. You'll have time to revise for the exam.
2. PS1 and PS2 contains the tools you need for the project, understanding and using them is crucial.
3. Lecture 3 and 4 also covers the important topics, drawing inspiration and re-applying code from them is allowed.
4. Last year's inaugural project is very similar to this year's. There is no official solution guide, but you can see the repositories of former groups. This group had a particularly elegant solution.
   I think Jeppe might show an official solution on Monday.