Exercise class 2

(week 9)

Introduction to Programming and Numerical Analysis

Class 4 and 8

Rosa Haslund Meyer
Spring 2024

KØBENHAVNS UNIVERSITET

Numerical optimization

My thoughts on the problem sets + tips

Exercises: Problem set 1

Constrained optimization

Exercises: Problem set 1 (continued)

I won’t go through **printing** and **plotting**, as this is the “basics” of the tasks and problems. However, if you have any questions or uncertainties along the way, feel free to ask!

Plan for today's exercise class

- 15.15-15.30: recap – numerical optimization and my thoughts on problem sets + tips
- 15.30-16.00: exercises – problem set 1
- 16.00-16.15: Break
- 16.15-16.20: recap – constrained optimization
- 16.20-16.55: exercises – problem set 1
- 16.55-17.00: Follow-up and questions

Optimization

Objective: **find** the x that **minimizes/maximizes** $f(x)$

In economics we're used to solve optimization problems analytically using first and second order conditions.

Works different on computers: we can only evaluate one point at a time – how do we find the minimum?

→ numerical optimization: use iterative methods to approximate the minimum or maximum of a function.

FYI: in computer/data science it's convention to minimize – if we want to maximize, we'll simply minimize the negative of the given function.

Optimizing using grid search

One way of optimizing is just to try a lot of different x-values and pick the one that minimizes (gives the smallest values of) $f(x)$:

- Divide the range of possible values for x into a grid
- Evaluate the function at each grid point
- Choose the point with the lowest or highest function value

Pros:

- Provides a rough ”sketch” of what the function looks like
- Robust against non-global minima (relatively)

Cons

- Does not check x-values from outside the grid
- Computationally expensive – especially for high-dimensional problems
- Only ever as precises as the given grid – grid with 1000 points more precise than grid of 100 points – but slower

Optimizing using solvers

A solver is some algorithm that searches for the minimum value of x by trying different values and updating guesses based on the evaluation of $f(x)$.

The algorithm "choses" which values of x to try – except for the initial guess which must always be provided by the user.

Pros:

- Solvers are often more efficient for continuous optimization problems – faster and less computationally intensive than grid search (Newton's Method is a solver)
- More precise solution (generally)

Cons:

- The solution may depend on the chosen start value
- May not converge to a solution… initial guess provided to the solver is too far from the optimal solution, numerical precision issues (dealing with functions that involve very small or large numbers)

My thoughts on problem sets

The problem sets will be more challenging than DataCamp but give you a solid foundation for solving assignments and teach you everything you should know for the exam!

If you don't make it through all problems during the exercise classes, I suggest you finish them at home.

It's OK to look at previous lectures or problem sets. Coding is rarely done from scratch.

OK to take a peak look at the solutions... the only one you're cheating is yourself: make sure you can solve the problem yourself as well!

Don't rush through the exercises – focus on understanding the code to make a good foundation.

My thoughts on problem sets

Always try to understand problems conceptually before you begin programming: what are you trying to achieve, which steps is likely required? etc. …

When trying to understand code or finding the root to some error, always use print-statements to check if everything acts like you expect them to. Experimentation in general is key!

The extra problems are great for preparing you for future projects, as they often requires you to do a bit more ”thinking” yourself.

If you get an error message or forgot some syntax, try to Google before asking. **When it comes to programming, Google (and StackOverflow) can solve a lot of minor errors and syntax errors!**

Tips!

Task 1:

Hints:

- Later you will have troubles with your code running because you didn't specify the values of alpha and beta. (`alpha = 0.5, beta = 1`)

… If you take a sneak peek at the solution (A1.py) you'll see that alpha and beta has a default parameter value. If you didn't write these default parameter values, it's not at all wrong!

I'll admit, it's a bit weird where the values for alpha and beta came from (as they're not mentioned in the problem), but just think of them as some values that alpha and beta happens to take in this problem.

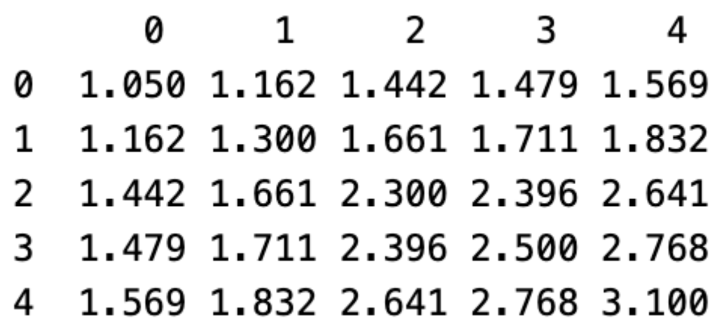
As well as setting them as default parameter values, you could have just as easily set them as global variables or even as local variables within the function (just does not make that much sense).

(Refer to the notebook for problem set 1 I post on GitHub and read comments for clarification.)

Tips!

Task 2:

You need to print a table looking like this:



```
       0     1     2     3     4
0  1.050 1.162 1.442 1.479 1.569
1  1.162 1.300 1.661 1.711 1.832
2  1.442 1.661 2.300 2.396 2.641
3  1.479 1.711 2.396 2.500 2.768
4  1.569 1.832 2.641 2.768 3.100
```

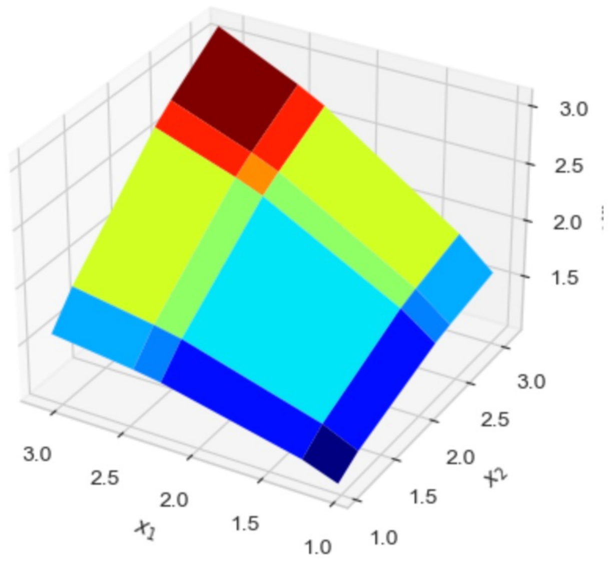
Hints:

- Create the side numbers using enumerate()
- f-strings is a great tool for printing and formatting your results. The basic syntax for printing floats is: f'{value:width.digitsf}, where width is the total width (characters) and digitsf is digits after the decimal point (if you're feeling overwhelmed, forget about this for now).

Tips!

Task 3:

You need to reproduce a figure looking like this:



Hints:

- You can look at the lecture notebook 1-plot, specifically cell 52: plot the utility function.
- Before task 3 you import a lot of plot modules – you might experience an error with the line: `plt.style.use('seaborn-whitegrid')` – to see which styles is available to you, write: `print(plt.style.available)` and chose which ever style you like.

Tips!

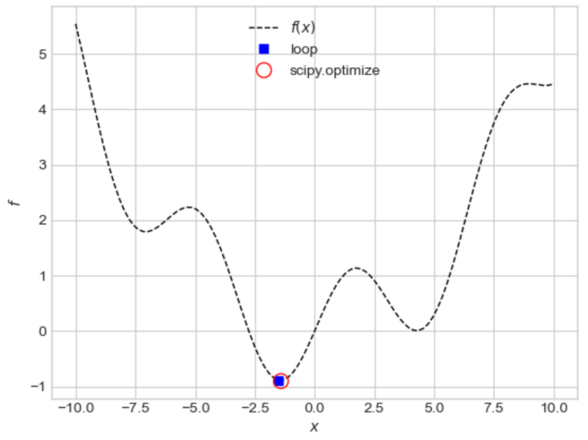
Task 4:

Hints:

- Numpy has the function np.sin()
- For Scipy optimize, I recommend you look at previous lecture notebooks (for example "1-optimize" or refer to the documentation (i.e. google)
- I recommend you use the "Nelder-Mead" method for the solver
- The figure you need to plot should look something like this:
  (i. e. you should plot the function and the two points you found during optimization)

```
Best with the loop method is: -0.88366802 at x = -1.51515152
Best with scipy.optimize is:  -0.88786283 at x = -1.42756250

<matplotlib.legend.Legend at 0x157b4f550>
```



Tips!

Task 5:

Hints:

- How does multiple good impact for loops?
- You should get:

```
x = [2.14 1.07 0.71 0.36 0.43 ]
utility = 0.758
E = 10.00 <= I = 10.00
expenditure shares = [0.21 0.21 0.21 0.14 0.21 ]
```

Task 6:

Hints:

- Google the documentation for the itertools product-function
- Generate all possible combinations of x1, x2, x3, x4, x5
- Use only one for loop – what should it iterate through?

Time for exercises

Problem set 1:

- Task 1-3: functions
- Problem 4: optimization
- Problem 5-8: utility maximization problem

If this is your very first-time programming or working Python and your feeling overwhelmed, I can recommend you take it slow and try and understand each line of code before moving on.

At **16:15**, I'll do a short recap on **constrained optimization**.

Constrained optimization

Constrained optimization work much in the same way: both grid search and solvers can be used.

OR: The objective function can be adjusted by adding a "penalty" if the constraint is violated and use unconstrained optimization:

- Pro: Helps "guiding" the solver if it end up outside the given constraint
- Con: May introduce new local minima

Unconstrained optimization simply means that the choice variable can take on any value – there are no restrictions. Constrained means that the choice variable can only take on certain values within a larger range.

Solvers - optimization

Which solver to use for optimizing? Depends on the problem, you'll grow stronger in which to choose along the way. Some examples of methods from SciPy are:

Unconstrained optimization:

- `bfgs` (fast, especially if provided with gradient/hessian)
- `nelder-mead` (robust, but computationally slow)

Constrained optimization:

- `flsqp` (fast, especially if provided with gradient/hessian)
- `SLSQP` – multi-dimensional constrained solver (finds optimal solution almost instantly for a small amount if input features – used in lecture notebook "1 – optimize"!)
- … Or you can use unconstrained optimization + penalty of constraint if violated

Next time...

Video lectures:

- Random numbers basics
- Random numbers example
- Random numbers advanced (+) (optional)

Exercises – Problem set 2. Finding the Walras equilibrium in a multi-agent economy:

- Drawing random numbers
- Monte Carlo integration
- User-defined modules
- Saving and loading
- … and everything you've learnt up until now!

The inaugural problem is due 24th march!

- Make sure you understand the concepts covered these previous weeks!
- Will be presented in a **physical lecture March 11th** (Monday)

Questions and/or comments?