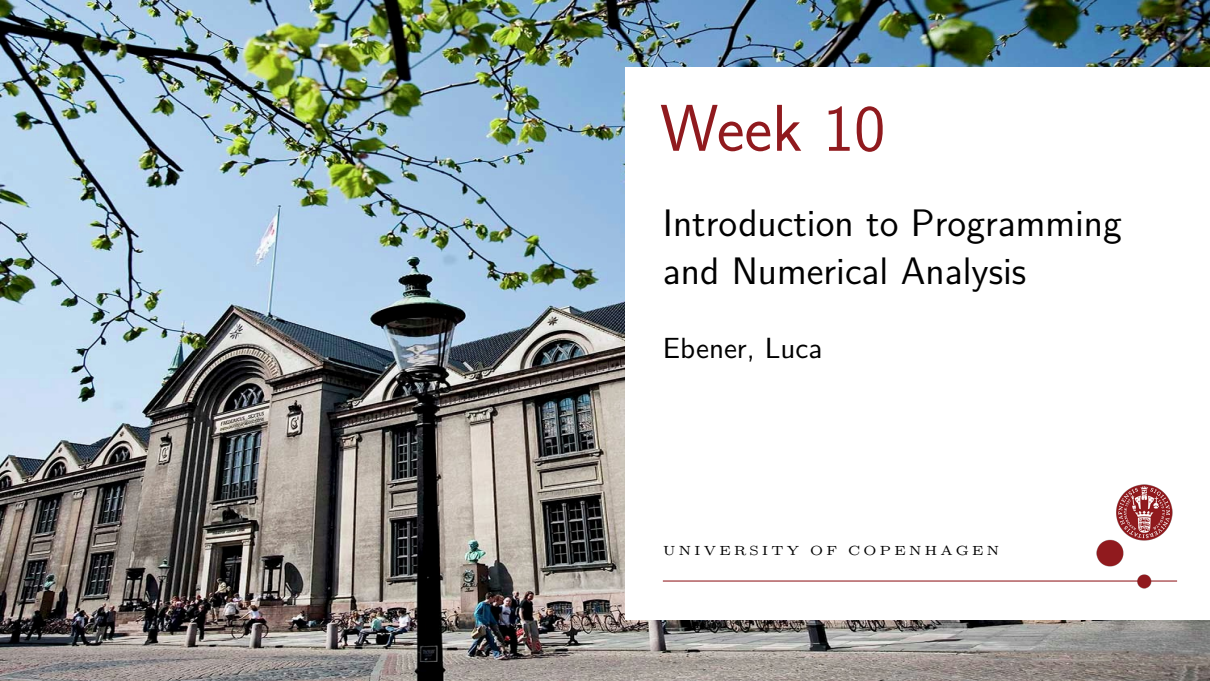

Week 10

Introduction to Programming and Numerical Analysis

Ebener, Luca

UNIVERSITY OF COPENHAGEN

Overview

- Hints and overview of the tasks
- Work on problem set 2
- Next week

Drawing Random Numbers

- Overview:
  - Generating random numbers using NumPy.
  - Setting seed for reproducibility.
  - Repeating random number generation.
- Key Aspects:
  - Seeding the random number generator (`np.random.seed(1986)`).
  - Storing the state of the random number generator (`state = np.random.get_state()`).
  - Resetting the state to reproduce random numbers (`np.random.set_state(state)`).
  - Generating random numbers (`x = np.random.uniform()`).

Expected Value and Variance Estimation (1)

Given:

$$\mathbb{E}[g(x)] \approx \frac{1}{N}\sum_{i=1}^{N} g(x_i)$$

$$\mathbb{VAR}[g(x)] \approx \frac{1}{N}\sum_{i=1}^{N}\left(g(x_i) - \frac{1}{N}\sum_{i=1}^{N} g(x_i)\right)^2$$

where $x_i \sim \mathcal{N}(0, \sigma)$ and

$$g(x, \omega) = \begin{cases} x & \text{if } x \in [-\omega, \omega] \\ -\omega & \text{if } x < -\omega \\ \omega & \text{if } x > \omega \end{cases}$$

Expected Value and Variance Estimation (2)

- Objective: Estimate the expected value and the variance of $g(x)$ using Monte Carlo simulation.
- Steps:
  1. Generate random samples x_i from $\mathcal{N}(0, \sigma)$ (`np.random.normal(loc=0, scale=sigma, size=N)`).
  2. Evaluate $g(x_i)$ for each sample.
  3. Calculate the approximate expected value and variance using the given formulas (`np.mean(g(x, omega))`, `np.var(g(x-mean, omega))`).

Interactive Histogram

To create an interactive histogram with fitting normal distribution:

1. Import necessary libraries:
   - `matplotlib.pyplot` for plotting.
   - `ipywidgets` for interactivity.
   - `scipy.stats.norm` for the normal distribution.
2. Define a function (`fitting_normal`) to:
   2.1 Generate a normal distribution based on given parameters (μ and σ).
   2.2 Generate x-values within a certain range.
   2.3 Plot the normal distribution and a histogram of the provided data.
3. Set true and initial guess parameters for mean (μ) and standard deviation (σ).
4. Generate random data from a normal distribution with true parameters.
5. Use `widgets.interact` to create sliders for adjusting the guess values of μ and σ.

Importing from Modules and Defining Functions

- Importing from Modules:
  - Modules are files containing Python code that can be reused in other Python scripts.
  - Importing syntax: `import module_name`.
  - Import all functions from a given module: `from module_name import *`

Working with Git: Forking, Cloning, and Syncing

1. Forking a Repository:
   - Go to the GitHub repository you want to fork.
   - Click on the "Fork" button in the upper right corner.
   - This creates a copy of the repository in your own GitHub account.
2. Cloning to VS Code: Follow the guide on the course webpage.
3. Adding the Original Repository as a Remote:
   - In VS Code, open the command pallette (`Ctrl+Shift+p+'>Git: Add remote...'`).
   - Paste in the url of the master repository+Enter
   - Type any name (e.g. 'Upstream')+Enter
   - Now, changes from the original repository can be loaded by `Ctrl+Shift+p+'>Git: Pull From...'`, then choosing 'Upstream'.

Exchange Economy

Consider an **exchange economy** with the following characteristics:

1. 2 goods, (x_1, x_2), N consumers indexed by $j \in \{1, 2, \ldots, N\}$
2. Preferences are Cobb-Douglas with truncated normally **heterogenous** coefficients:

$$\begin{aligned} u^j(x_1, x_2) &= x_1^{\alpha_j} x_2^{1-\alpha_j} \\ \tilde{\alpha}_j &\sim \mathcal{N}(\mu, \sigma) \\ \alpha_j &= \mathsf{max}(\underline{\mu}, \mathsf{min}(\overline{\mu}, \tilde{\alpha}_j)) \end{aligned}$$

3. Endowments are **heterogenous** and given by:

$$\begin{aligned} \boldsymbol{e}^j &= (e_1^j, e_2^j) \\ e_i^j &\sim f, f(x, \beta_i) = 1/\beta_i \exp(-x/\beta) \end{aligned}$$

Implementation Hints

To implement the exchange economy, consider the following steps:

1. Generate truncated normally distributed coefficients $\tilde{\alpha}_j$ for each consumer.
2. Sample endowment values e_i^j from the given distribution $f(x, \beta_i)$ for each consumer.
3. Implement demand function from analytic solution and compare to aggregated endowments to get excess demand.
4. Minimize excess demand by adapting prices.

Understanding Pickle

Pickle is a Python module used for serializing and deserializing Python objects. It allows objects to be converted into a byte stream for storage or transmission, and later reconstructed into their original form.

- **Serialization:** Pickle converts Python objects into a byte stream, preserving their structure and data.
- **Deserialization:** Pickle reconstructs Python objects from the byte stream, restoring their original state.
- **Usage:**
  - Import the `pickle` module in Python.
  - Serialize objects using `pickle.dump()` method.
  - Deserialize objects using `pickle.load()` method.

Multiple Goods

Consider an exchange economy with the following characteristics:

- M goods, $(x_1, x_2, \ldots, x_M)$, N consumers indexed by $j \in \{1, 2, \ldots, N\}$
- Preferences are Cobb-Douglas with truncated normally **heterogenous** coefficients:

$$u^j(x_1, x_2, \ldots, x_M) = x_1^{\alpha_j^1} \cdot x_2^{\alpha_j^2} \cdots x_M^{\alpha_j^M}$$
$$\alpha_j = [\alpha_j^1, \alpha_j^2, \ldots, \alpha_j^M]$$
$$\log(\alpha_j) \sim \mathcal{N}(0, \Sigma)$$

  where Σ is a valid covariance matrix.
- Follow the same steps as before but in matrix algebra to calculate the demand and excess demand for all goods at the same time and adapt all prices at once. Make sure that alphas add up to one.

Next week

- Inaugural Project
  - Hand-in: 24.03.
  - Peer-feedback: 31.03.
- Demonstrating Git/Github use