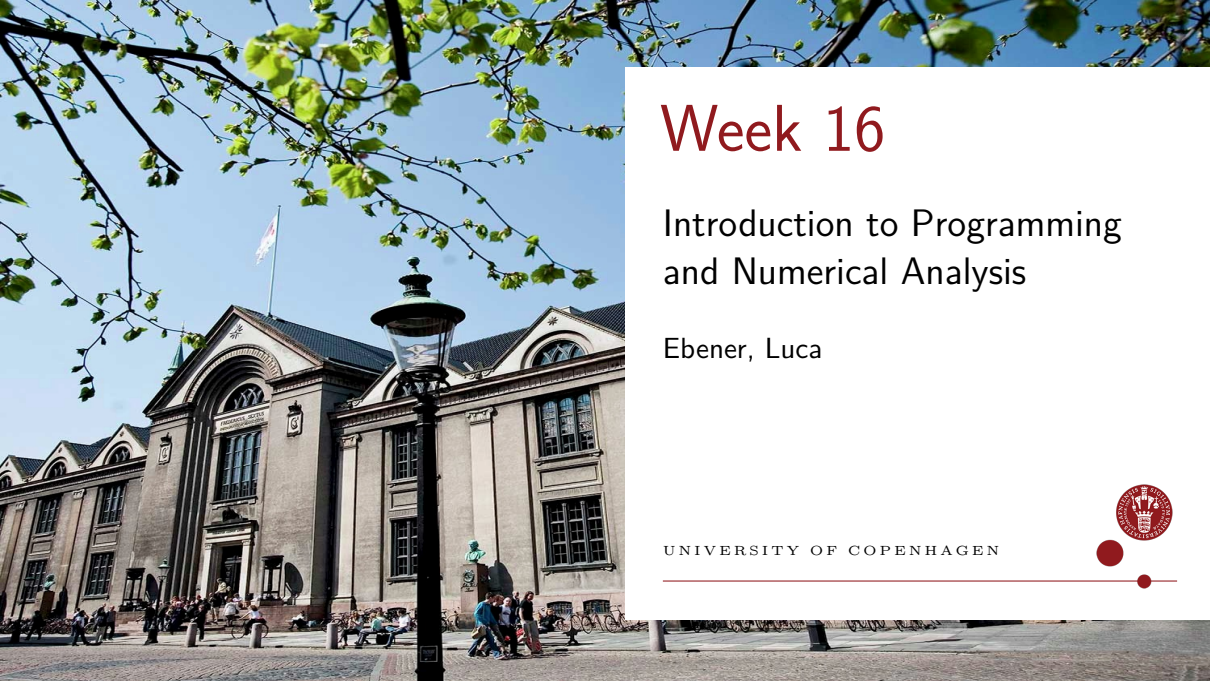

Week 16

Introduction to Programming and Numerical Analysis

Ebener, Luca

UNIVERSITY OF COPENHAGEN

Overview

- Linear equations
- Non-linear equations
- Symbolic math
- Work on problem set 6

Linear equations

A lot of economic models can be expressed as systems of equations

$$Ax = b \Leftrightarrow \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

We have options with varying levels of efficiency to solve them.

- Inverting matrices - may be costly
- Gauss-Jordan elimination
- Gauss-Seidel - faster but may not converge
- LU-factorization - very fast due to lack of matrix inversion

Non-linear equations

We are often interested in the roots of non-linear equation systems. Examples for root-finding algorithms include

- Bisection - no gradient, but my be slow
- Newton or Halley - use gradient to update guesses efficiently
- Brent

Symbolic Python

Symbolic Python (SymPy) can perform mathematical operations and solve equations analytically. It operates much like what we are familiar with from micro and macro courses.

- It is more familiar when we are accustomed to solving equations analytically.
- It can sometimes find all solutions to problems with multiple solutions.
- It can provide exact solutions, rather than numerical approximations.

However, there are limitations:

- It can only perform tasks that could also be done with pen and paper.
- It does not enhance understanding of numerical methods.
- Many models lack analytical solutions.

Problem set I

1, A You want to find the determinant of the inverse of the dot-product of two matrices. For dot-product, you can use `np.dot()` but also the `@`-operator.

1, B It should maybe say using `scipy.linalg`. Note that each equation can be solved separately.

2 For the `gauss_jordan()` function to work, you need to add e as a 4th column in F. `np.column_stack((F,e))` is the easiest in this case.

Problem set II

3

- Remember to initiate unknown variables in SymPy using `sm.symbols()`.
- Also, notice that you need to use the SymPy sine-function (`sm.sin()`).
- For the remaining SymPy operators you need, I'd suggest using Google. Notice that their tutorial/documentation loads SymPy as: `from sympy import *`
- You can also refer to this week's notebook, which utilizes all the relevant SymPy functions.

Problem set III

Solow

A: Use the answer from above with solving equations symbolically. Notice that the default return of the solver is a list, even when there is only one solution, so you need to extract the first element of this list to get pretty printing.

B: `sm.lambdify((args),f)` is like `lambda args: f`, for symbolic arguments. In this setting, you can use your answer to A as `f`.

C: There are multiple ways of using `root_scalar`. 'Brentq'-method, which requires bounds (called brackets for `root_scalar`), 'bisect' is also possible with the same needs. 'Newton' method doesn't need bounds but does need a first derivative and many more.

D: Using CES-production function is (relatively) easy because we're using Python instead of maths.