Computational Methods in Economics

Lecture 1: Introduction

Alex Schmitt

ifo Institute, Center for Energy, Climate and Exhaustible Resources

ifo Institute

Personal Information

- Alex Schmitt, PhD
- Economist at the ifo Institute, Center for Energy, Climate and Exhaustible Resources
- Web: `http://www.cesifo-group.de/ifoHome/CESifo-Group/ifo/ifo-Mitarbeiter/cvifo-schmitt_a.html`
- Email: schmitt@ifo.de
- Phone: 089 9224 1408
- Office hours
  - When? On appointment - please send an email!
  - Where? ifo Institute, Poschingerstr. 5 - please ask for me at the reception!
  - Via Email: 24/7

ifo Institute

Outline

ifo Institute

About the Course: Logistics and Objectives

ifo Institute

Course Logistics

- 6 ECTS course (2h + 2h per week)
- Lecture: Wednesday, 16-18 c.t., M 101
- Tutorial (Christina Littlejohn): Friday 10-12 c.t., M 101
- Examination
  - Oral exam ($\sim$ 20 min)
  - Date (tentative): February 7

ifo Institute

Schedule: Part 1

Week	*Wednesday 16-18 c.t.*	*Friday 10-12 c.t.*
Oct 16-20	L: Introduction	T: Installing Python & Version Control
Oct 23-27	T: Intro to Python I	T: Intro to Python II
Oct 30-Nov 3	NO LECTURE (*Holiday*)	T: Intro to Python III
Nov 6-10	L: Systems of Linear Equations	T: Intro to Python IV
Nov 13-17	L: Root Finding	T: Problem Set 1
Nov 20-24	L: Numerical Optimization	T: Problem Set 2
Nov 27-Dec 1	L: Function Approximation	T: Problem Set 3
Dec 4-8	L: Numerical Integration	T: Problem Set 4

ifo Institute

Schedule: Part 2

Week	*Wednesday 16-18 c.t.*	*Friday 10-12 c.t.*
December 11-15	L: Dynamic Programming I	T: Problem Set 5
December 18-22	L: Dynamic Programming II	T: Problem Set 6
January 8-12	L: Projection Methods	T: Problem Set 7
January 15-19	L: Applied Example 1	T: Problem Set 8
January 22-26	L: Applied Example 2	T: Review Session
Jan 29-Feb 2	L: Applied Example 3	T: Review Session
February 5-9	EXAM	-

ifo Institute

Material

- The slides and notebooks used in the lecture will be uploaded on LSF.
- You can also download them using Git (more on that in the first tutorial session!)
- The mandatory textbook for this class is Mario J. Miranda and Paul L. Fackler (2004), "Applied Computational Economics and Finance", MIT Press.
- Other textbooks that I will refer to as background reading (not required):
  - Kenneth Judd (1998), "Numerical Methods in Economics", MIT Press.
  - John Stachurski (2009), "Economic Dynamics - Theory and Computation", MIT Press.
- Some sections will also have papers, as either required or recommended reading (compare syllabus).

ifo Institute

Study Tips

- Please attend the lectures regularly – the notebooks are no substitute for the lecture!
- Complement the notebooks by additional information provided in the lecture and the required readings
- In case you cannot come, get the notes from a classmate
- You will only understand the material once you have practiced solving models; try to attempt the problem sets before attending the tutorial!
- A balanced combination of lonely struggle and teamwork is good, but whatever works

ifo Institute

Course Objectives

- The goal of this course is to provide an introduction to computational tools for conducting numerical analysis of economic models; hence,
  - to provide you with a basic understanding of some fundamental techniques and algorithms used in numerical analysis and economic modeling
  - to give you "hands-on" experience on how to implement these techniques in Python
- "Computers and mathematics are like beer and potato chips: two fine tastes that are best enjoyed together" (J. Stachurski)
- That said, it is important to note that
  - this is neither a math class nor a CS course!
  - the balance between hands-on coding and mathematical background may at times be tilted in the direction of the former!

ifo Institute

Programming

- Consider this course an **opportunity** to get familiar with programming in general and a specific language in particular.
  - Indispensable skill in the context of economic modeling
  - Programming has become increasingly important in all fields of economics
  - Great skill to have for the job market
  - It is fun!

ifo Institute

What is "Computational Economics" and why do we need it?

ifo Institute

Computation in Economics

- When do we use computers in economic analysis?
- Econometrics
  - "black box" applications (e.g. STATA)
  - what happens in the box are implementations of numerical algorithms discussed in this course
- Illustration of theoretical "pen-and-paper" results
  - usually, the main result of a "deductive theory" paper is a general statement or closed-form expressions (*theorems*)
  - numerical analysis is used to illustrate this result for a given set of parameters
  - check for quantitative importance of (specific instances of) a result (e.g. the relative sizes of opposing effects)

ifo Institute

Computation in Economics (cont.)

- Recall that in mathematics, a closed-form expression is a mathematical expression that can be evaluated in a finite number of operations (algebra and calculus)
- In economics, a model can be solved analytically if a solution can be derived from a closed-form expression
- Many models (in particular dynamic ones) and hence many interesting problems lack a *closed-form/analytical solution*
- Therefore, computers have become an indispensable tool for conducting quantitative research in economics

ifo Institute

Example

- An agent maximizes her utility over two consumption goods. She has an initial endowment x for good 1. Moreover, she can convert good 1 in good 2, using a decreasing-returns-to-scale technology. Formally,

$$u(c_1, c_2) = \frac{c_1^{1-\nu}}{1-\nu} + \frac{c_2^{1-\nu}}{1-\nu}.$$

s.t.

$$c_2 = (x - c_1)^{\alpha}.$$

- Taking first-order conditions and rearranging, we can find an optimality condition, that is, an expression in c_1:

$$(x - c_1)^{\alpha\nu-\alpha+1} = \alpha c_1^{\nu}.$$

ifo Institute

Example (cont.)

- In the special case $\nu = 1$, it is easy to verify that

$$c_1^* = (1 + \alpha)^{-1} x.$$

- In the general case $\nu \neq 1$, we are not able to find a closed-form solution for c_1
- This indicates a *trade-off*: do we rely exclusively to analytical methods at the cost of limiting ourselves to special (and potentially not that relevant) cases? Or do we embrace numerical methods to be able to deal with more general problems?

ifo Institute

Computational Economics - Defined

- What is *computational economics/computational methods/numerical methods*?
- Wikipedia: "Computational economics is a research discipline at the interface of computer science, economics, and management science. This subject encompasses computational modeling of economic systems, whether agent-based, *general-equilibrium, macroeconomic, or rational-expectations*, computational econometrics and statistics, computational finance, computational tools for the design of automated internet markets, [...]. Some of these areas are unique to computational economics, while others extend traditional areas of economics by solving problems that are difficult to study without the use of computers and associated numerical methods."
- More narrow: "Computational economics uses computer-based economic modeling for the solution of analytically and statistically formulated economic problems."

ifo Institute

Example II

- As the prime example for computer-based economic modeling, consider the stochastic neoclassical growth (Ramsey) model
- Workhorse model in macroeconomics, in particular when analyzing business cycles
- A representative consumer has lifetime utility

$$\sum_{t=0}^{\infty} \beta^t u(c_t, h_t),$$

where c denotes consumption, h labor supply and

$$u(c, h) = \frac{c^{1-\nu}}{1-\nu} - B\frac{h^{1+\eta}}{1+\eta}.$$

- β denotes the discount factor.

ifo Institute

Example II (cont.)

- Output of the single consumption good is given by

$$y_t = f(k_t, h_t) = z_t k_t^{\alpha} h_t^{1-\alpha},$$

  where k_t denotes the capital stock and z_t is an exogenous parameter capturing shifts in productivity
- The economy's resource constraint reads:

$$k_{t+1} + c_t = f(k_t, h_t) + (1-\delta)k_t = z_t k_t^{\alpha} h_t^{1-\alpha} + (1-\delta)k_t$$

  where δ denotes the rate of depreciation
- Assume that productivity shocks follow a stochastic process:

$$\ln(z_t) = \rho \ln(z_{t-1}) + \epsilon_t, \quad \epsilon_t \sim N(0, \sigma^2)$$

  and $\rho < 1$

ifo Institute

Example II (cont.)

- The social planner's problem reads:

$$\max_{\{c_t, k_{t+1}, h_t\}} \mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t u(c_t, h_t)\right]$$

s.t.

$$k_{t+1} + c_t = z_t k_t^{\alpha} h_t^{1-\alpha} + (1-\delta)k_t$$

and k_0 given.

- A solution to this problem consists of a set of policy functions (decision rules) $c_t = \tilde{c}(k_t, z_t)$, $k_{t+1} = \tilde{k}(k_t, z_t)$ and $h_t = \tilde{h}(k_t, z_t)$.

ifo Institute

Example II (cont.)

- For a specific setting where
  - the model is deterministic ($z_t = 1$);
  - utility is logarithmic in consumption:

$$u(c, h) = \log(c) - B\frac{h^{1+\eta}}{1+\eta}.$$

  - there is full depreciation ($\delta = 1$),

  we can find analytical solutions for the policy functions, specifically

$$k_{t+1} = \alpha\beta f(k_t, h_t), \;\; c_t = (1 - \alpha\beta) f(k_t, h_t)$$

  and

$$h_t = \left[B\frac{1-\alpha}{1-\alpha\beta}\right]^{\frac{1}{1-\eta}}.$$

- In its general form, this model cannot be solved analytically. Instead, quantitative macroeconomics in the past 35 years has made extensive use of computational methods.

ifo Institute

Deductive Theory vs. Computational Methods

- Compare Judd (1997)
- What is a *theory*?
  - collection of concepts, definitions and assumptions
  - a *theoretical analysis* determines the implications of a theory
- *Deductive* (conventional) theory:
  - "proving theorems" (or lemmas, or propositions)
  - "pen-and-paper results"
  - can determine the "qualitative structure of a model"
  - can rarely answer all important questions; if the theorist is unable to prove general results, she needs to turn to special cases which are tractable

ifo Institute

Deductive Theory vs. Computational Methods (cont.)

- Drawbacks of deductive theory:
  - special cases may sacrifice model elements of first-order importance
  - nonquantitative, i.e. no indication about the relevance of a result
- Deductive theory is one approach to theoretical analysis, but is not a synonym.
  - distinction between "mathematical economics and economic theory"
- Computation as another way to conduct theoretical analysis
- Hamming's motto: *The Goal of Computing is Insight, Not Numbers*.

ifo Institute

Arguments against Computational Methods

- Computer programs as a "black box"
  - Easy remedy: better exposition
  - Einstein: a model should be "as simple as possible, but not simpler"
- Numerical analysis contains approximation errors
  - "in economic theory, [...] the issue is not whether we use approximation methods, but where in our analysis we make approximations, what kind of approximation errors we tolerate and which ones we avoid, and how we interpret the inevitable approximation errors." (Judd 1997, p. 918)
  - simple cases in deductive analysis also make approximation errors
  - of course, even very complex computer models are simplifications of the real world
- Solving a model numerically also considers a special case, e.g. a given calibration
  - importance of sensitivity analysis/robustness checks
  - computation allows the modeler to consider a (finite) set of specific instances of a theory, not just one special case

ifo Institute

To Summarize

- In a nutshell: "[S]ince any theoretical analysis is really an approximation of the economic reality we are trying to model, we have to ask which is more important: the development of theories simple enough for theorem-proving, or examining more reasonable theories using less precise numerical methods." (Judd 1997, p. 909)
- At the end of the day, computational methods are another useful instrument in your economist's "toolbox"
- As with any tool, they will not be applicable to *every* problem, but may be the best (and sometimes the only) way forward for *some* problems
- *If all you have is a hammer, everything looks like a nail.*

ifo Institute

Rules of Thumb for Doing Good Computational Work in Economics (via Tony Smith)

1. Start with the simplest possible model, preferably one with an analytical solution.
2. Add features incrementally.
3. Never add another feature until you are confident of your current results.
4. Use the simplest possible methods. Use one-dimensional algorithms as much as possible.
5. Accuracy is more important than speed or elegance.
6. Use methods that are as transparent as possible (i.e., methods for which the computer code reflects as closely as possible the economic structure of the problem).
7. When you learn (or develop) a new method, test it on the simplest possible problem, preferably one with an analytical solution.

ifo Institute

Rules of Thumb for Doing Good Computational Work in Economics (cont.)

8. Dan Bernhardt's rule: If you have n errors in a piece of code and you remove one, you still have n errors. Scrutinize your results, even if they look right. Look for anomalies. *Assume your code is wrong until proven otherwise.*
9. Graph, graph, graph. Two-dimensional graphs are more informative than three-dimensional graphs.
10. Be able to replicate all of your intermediate and final results instantly. Save exact copies of the code used for each run, together with inputs and outputs (*version control!*).
11. Watch the computations as they proceed.
12. Look for hidden structure. Always compute (and print out) a few more numbers than you need.

ifo Institute

Rules of Thumb for Doing Good Computational Work in Economics (cont.)

13. Get good initial conditions.
14. Avoid black boxes. Understanding how the algorithm works is critical to interpreting the results.
15. Remember that programming is a creative activity analogous to writing a sonnet or composing a sonata (a static, visual representation of a process). Craft your programs. Strive for efficiency and elegance in your computer code. Develop a style. Practice structured programming, i.e., write code that reflects the structure of the algorithm.
16. Don't program when you are tired. Don't program too quickly.

ifo Institute

Python: An Overview

ifo Institute

Choice of Programming Language

- In this course, we're going to use exclusively **Python** in the lectures and tutorials; in other words, examples, exercises, solutions etc. will be provided only in Python.
- You are required to work on the exam in Python as well!

ifo Institute

What is Python?

Python is a widely used ***high-level programming language*** *for* ***general-purpose*** *programming, created by Guido van Rossum and first released in 1991. An* ***interpreted language****, Python has a design philosophy which emphasizes* ***code readability*** *(notably using whitespace indentation to delimit code blocks rather than curly braces or keywords), and a syntax which allows programmers to* ***express concepts in fewer lines of code*** *than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale.*

`https://en.wikipedia.org/wiki/Python_(programming_language)`

ifo Institute

Some characteristics

- High-level:
  - emphasis on readibility
  - easy to use and easy to learn

    "I have this hope that there is a better way. Higher-level tools that actually let you see the structure of the software more clearly will be of tremendous value." (Guido van Rossum)

- Interpreted / dynamically typed:
  - no explicit compilation step (as e.g. in C++ or Fortran)
  - inefficient memory access
  - As a result, running an application in Python is (typically) slower than in lower-level, compiled languages

ifo Institute

Speed vs. Access

Given this inherent inefficiency, why would we even think about using Python? Well, it comes down to this: Dynamic typing makes Python easier to use than C. It's extremely flexible and forgiving, this flexibility leads to efficient use of development time, and on those occasions that you really need the optimization of C or Fortran, Python offers easy hooks into compiled libraries. It's why Python use within many scientific communities has been continually growing. With all that put together, ***Python ends up being an extremely efficient language for the overall task of doing science with code****.*

`https://jakevdp.github.io/blog/2014/05/09/why-python-is-slow/`

- Keep in mind: programming time is the sum of *coding* time and *running* time!

ifo Institute

Open Source

- Python is free and open-source, with a large collection of standard and external software libraries
- Large community of users and developers ("Pythonistas"); many applications have been implemented in packages which are easy to downloaded and import
- Extensive documentation
  - Standard Library: `https://docs.python.org/3/library/`
  - Google is your friend;)!
- Fun fact: Python is not named after the animal, but after *Monty Python's Flying Circus*.

ifo Institute

General-purpose Language

In some sense, Python is a "jack of all trades, and master of *some*". Examples for applications include:

- "Scientific Computing" (`Numpy/Scipy`, `Matplotlib`: matrix operations, optimization, root finding, plotting, etc.) $\Leftrightarrow$ substitute for MATLAB
- Working with data (`Pandas`, `Statsmodel`) $\Leftrightarrow$ substitute for R, Stata
- Parsing text files
- Web scraping (`Beautiful Soup`)
- Machine Learning (`Scikit-Learn`)
- Games
- ...

ifo Institute

Why Should You Learn Python?

- Easiest answer: because it is required for this course :)
- General answer: "trifecta of greatness"
  - ease of use
  - open source with a large dev community
  - general purpose
- If you have no or very little experience with programming and scripting languages:
  - programming skills are highly useful and valuable both on the academic and non-academic job market
  - due to its emphasis on readability and simple syntax, Python is (most likely) the easiest way to get into programming while still being a powerful language

ifo Institute

Why Should You Learn Python (cont.)?

- Why should you learn Python *in addition* or *instead of* another programming language?
- General point 1: having more tools at your disposal is never a bad idea!
  - *If all you have is a hammer, everything looks like a nail.*
- General point 2: there is no such thing as the "best programming language"
  - what the "right" language is depends very much on the task(s) at hand
  - highly subjective

*To be honest, I really hate those types of questions: "Which * is the best?" (* insert "programming language, text editor, IDE, operating system, computer manufacturer" here). This is really a nonsense question and discussion. Sometimes it can be fun and entertaining though, but I recommend saving this question for our occasional after-work beer or coffee with friends and colleagues.*

https://sebastianraschka.com/blog/2015/why-python.html

ifo Institute

Why Should You Learn Python (cont.)?

- Python vs. MATLAB
  - everything that you can do in (basic) MATLAB (scientific computing, graphics) you can also do in Python (using packages like Numpy, Scipy, Matplotlib), with little adjustment cost
  - not everything that you can do in Python you can do in MATLAB (e.g. data analysis)
  - Data Science/Big Data Analysis: Python and R seem to be the most popular languages by a mile
  - No general speed advantages: `http://julialang.org/benchmarks/`
  - "Knockout blow": MATLAB is rather pricey, Python is open source
  - More arguments
  - if you're an experienced MATLAB user, the cost of learning Python is small!
- Main advantages of MATLAB:
  - Quality of (some) algorithms and routines
  - Slightly easier syntax than Python/Numpy
  - "Networking" aspect: (still) predominant language in computational economics

ifo Institute

Why Should You Learn Python (cont.)?

- Python vs. Octave ($\approx$ MATLAB for free)
  - everything that you can do in (basic) Octave you can also do in Python (using packages like Numpy, Scipy, Matplotlib), with little adjustment cost
  - not everything that you can do in Python you can do in Octave (e.g. data analysis)
  - (Much) smaller community, limited set of toolboxes (hence you might have to write more yourself); less mature IDE
  - No general speed advantages (quite the opposite!)
  - Go for the more powerful alternative!!!

ifo Institute

Why Should You Learn Python (cont.)?

- Python vs. R
  - Disclaimer: I have no experience with R, so all of the following is based on Google and hearsay :)
  - R has a more narrow application spectrum: data science/data analysis $\Rightarrow$ not general purpose
  - if you don't know either, Python seems to have considerably lower learning cost
  - if you're an experienced R user, you may not have a strong incentive to learn Python: in the realm of data science/data analysis, both are very close substitutes
  - however, the cost of learning Python are most likely even lower than for a beginner:

> *Just think about, you learned how to swing a hammer to drive the nail in, how hard can it possibly be to pick up a hammer from a different manufacturer?*

`https://sebastianraschka.com/blog/2015/why-python.html`

ifo Institute

Some Other Alternatives

- "Classics": Fortran, C, C++
- Excel
- Algebraic Modeling Languages: GAMS, AMPL
  - proprietary, but there are trial versions
  - much less general purpose: primary use is numerical optimization
  - less intuitive syntax, hence higher cost of learning
- **Julia**

ifo Institute

Installing Python

- Different Python distributions
- We use the **Anaconda** distribution:
  - download for free at `https://www.continuum.io/downloads`
  - easy to install, manage and update
  - contains all relevant packages (Numpy, Scipy, Pandas, etc.)
- Important commands: `conda list`, `conda update` *`package name`*
- Check `https://conda.io/docs/using/index.html` for more information.

ifo Institute

Python 2 or Python 3?

- *Python 2.x is legacy, Python 3.x is the present and future of the language*
- For us, it doesn't matter too much which one we use; since Python 3.6 is the "current" version, we go this way
- Python 2.7 is no longer actively developed, but still widely used; some packages are written exclusively for Python 2
- There are some differences with respect to the syntax; see for example here or here
- When writing new code, it is relatively easy to write it in a way so that it runs under both Python 2 and Python 3
- With Anaconda, it is fairly straightforward to have an environment for each of them, and switch between them as need be

ifo Institute

How to Use Python

Multiple possibilities:

- Command line (Ipython)
- Scripts ⇒ make sure to get a "good" text editor (e.g. Sublime Text 2/3, Notepad++, etc.)!
- IDLE (Integrated Development and Learning Environment) ⇒ simple GUI (outdated?)
- **Jupyter notebooks**
  - **Ju**lia, **Pyt**hon and **R**
  - Great tool for writing and documenting code, as well as for teaching
- ... and more!

ifo Institute

(Some) Python Resources

- (Imperfect) Substitutes for this course: free online courses on MOOC sites like Coursera or Udacity. Examples that I can recommend:
  - `https://www.coursera.org/specializations/python`
  - `https://www.udacity.com/course/intro-to-data-analysis--ud170`
- Lots of examples for using Python in scientific computing, with a focus on *economics* application:
  `http://quant-econ.net/py/about_py.html`
- `http://www.pythonforbeginners.com/`
- Great blog on using Python in general and for machine learning in particular: `https://sebastianraschka.com/blog/index.html`
- ... and many more!!!

ifo Institute