Introduction

Numerical Methods

Cezar Santos

FGV/EPGE

Some Info

- **This course:** Provides techniques for the analysis and evaluation of dynamic economic models, e.g. DSGE, firm and industry models
- The main emphasis is on learning the computational tools and their implementation
- **Target group:** Students who intend to do quantitative research in macro, applied micro or structural econometrics
- Requires you to use standard computer programming languages (such as Matlab, C or Fortran)
- **Background:** Dynamic programming / Recursive macro

Why Study Numerical Methods?

- "Paper-and-pencil" has its limits
- Often closed form solutions to models are hard or impossible to obtain
- Increasing computational power allows to study complex models
- Techniques also useful for fields that require structural modeling (and estimation), e.g. applied microeconomics
- Macro:
  - Dynamic general equilibrium models: Keystone of modern macroeconomics
  - Provides powerful tool for evaluating macroeconomic theories, macroeconomic policy, welfare
  - "Quantitative": Provides ways in which to evaluate the empirical plausibility of the theory
  - Need to have the tools to analyze and compute dynamic economic models
- Drawbacks: Only approximate solution, no theorems/proofs, plenty of room for human error ...

What About Knowledge of Computers?

- In order to implement the techniques we're gonna cover, you need to learn (at least) one programming language
- Interpreted vs compiled languages
  - Interpreted: Matlab, Mathematica, Python, etc.
  - Compiled: Fortran, C, etc.
- Comparison:
  - Compiled is much faster and easier to spot errors before the code is run
    - Check out this comparison: **Table 1**
  - Interpreted is much more flexible
  - Compiled is much more verbose
- Basic principle: minimize ***total*** time

Getting Things Ready on YOUR Computer

- Interpreted languages:
  - Usually not much to do here; just install them
  - Note: Octave is a software that is essentially compatible with Matlab. Plus, it's free
- Compiled languages:
  - Usually easier to set up on Unix based machines (Mac or Linux)
  - You first need a compiler in order to run things
  - Intel Fortran is fast (and expensive)
  - GNU Fortran (gfortran) is free
  - Both Intel and the GNU project (gcc) have C compilers
  - Most economists prefer Fortran
  - Good code editor
    - Mac: gedit, Xcode
    - Windows: gedit, Notepad++
  - IDE vs command line
    - Open source IDE: Code::Blocks

Some Helpful Material

- We're not gonna follow any book in particular, but you may find the following references helpful:
  - Adda and Cooper, Dynamic Economics (**Amazon link**)
  - Judd, Numerical Methods in Economics (**Amazon link**)
  - Marimon and Scott, Computational Methods for the Study of Dynamic Economies (**Amazon link**)
  - Sauer, Numerical Analysis (**Amazon link**)
- Programming languages:
  - McConnell, Code Complete (**Amazon link**)
  - Gilat, Matlab: An Introduction (**Amazon link**)
  - Ellis, Philips and Lahey, Fortran 90 (**Amazon link**)
  - Fortran tutorials: **Link here** or **Grey's slides**

Some Helpful Material (cont.)

- Some economists with helpful stuff online:
  - Grey Gordon: **Link**
  - Karen Kopecky: **Link**
  - Makoto Nakajima: **Link**
- Tony Smith's tips: **Link**
- Alan Miller's Fortran software: **Link**

Our Running Example for a Good Chunk of the Course

- Standard RBC model
- Representative agent with preferences

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

- Representative firm with CRS technology subjective to multiplicative TFP shocks: $Y_t = z_t F(K_t, L_t)$
- No government
- Competitive equilibrium
- By the welfare theorems, we'll work with the planner's problem

Our Example: Recursive Formulation

- Since there's no utility for leisure, let's forget about it and define $f(k) = F(k,1)$
- The (basic) problem we'll be trying to solve can be written as:

$$V(k,z) = \max_{c,k'} u(c) + E\beta V(k',z')$$

s.t.

$$c + k' = zf(k) + (1-\delta)k$$

- Question: how to solve this problem?

Different Methods

- We'll discuss several different methods to solve the problem above (and, obviously, a lot more)
- Each method may be more suited for certain types of problems
- Some methods:
  - Value function iteration
  - Projection methods (spectral, finite elements)
  - Perturbation methods (e.g. linearization)
- We'll also need to learn several helpful things:
  - root-finding, interpolation, discretization of stochastic processes, etc.
- We'll also discuss some techniques for some potentially more complicated models:
  - Computing stationary distributions, idiosyncratic and aggregate risk, OLG, calibration, simulation-based estimation, etc.