Macroeconomics 3

TA session 1

Python primer

Prof.: Marco Maffezzoli TA: Andrea Pasqualini

A.y. 2017-2018

PhD in Economics and Finance

Bocconi University, Milan

Local VS Global Solutions to Macro Models

- In Macro, need to solve models
  - What happens if I change that? **IRF**
  - What is the contribution of this one shock? **FEVD**
  - What is the prediction of my model? **Simulation**
  - What do my model + data say about parameters? **Estimation**
- Loosely speaking, two classes of models
  - "Regular" DSGE (local solutions feasible)
  - "Non-regular" DSGE (global solutions necessary)
    - Potentially binding constraints
    - Discrete choice models
    - Heterogeneous agents
    - ...
- **These TA sessions**
  - Useful numerical techniques in Economics
  - Basics on global solution methods for Macro models

Plan of TA sessions

- Python primer
- Value Function Iteration & co. (deterministic case)
- Value Function Iteration & co. (stochastic case w/MC)
- Partial replication of Huggett (1993)
- Partial replication of Aiyagari (1994)
- Binning
- (If time permits) Other uses of Python in Econ

Outline of today

- Python
  - What
  - Why
  - How
- Dive-in
- IDEs and workflow

Python

What it is

- An interpreted programming language
- Dynamically, kind-of-strongly typed
- A small set of core functions and data type definitions
- A "platform" for modules

What it is **not**

- An efficient language for numerical calculus (e.g., Fortran)
- A matrix-oriented program (e.g., Matlab)
- A statistics-oriented program (e.g., R)
- A point-and-click software package (e.g., Stata, EViews)
- A comprehensive suite of programs for *[your application here]*

Basics

```
In [1]: 2 ** 3     # exponentiation
Out[1]: 8

In [2]: 3 / 2      # division
Out[2]: 1.5

In [3]: 3 // 2     # floor division
Out[3]: 1

In [4]: 3 % 2      # mod operator
Out[4]: 1
```

Basics

```
In [1]: 2 ** 3     # exponentiation
Out[1]: 8

In [2]: 3 / 2      # division
Out[2]: 1.5

In [3]: 3 // 2     # floor division
Out[3]: 1

In [4]: 3 % 2      # mod operator
Out[4]: 1
```

Wow. Such math, much macro!

Python — yeah, but why??

- Open source (**free** as in "free beer" and "free speech")
- Truly cross-platform (any Linux user here?)
- Enthusiast community (see StackOverflow)
- Flexibility
  - Can use multiple modules in a lean way
  - Example: can scrape the web and feed some number-crunching algorithm, all in one place
- Code readability and program usability
  - Syntax clearer than Matlab's
  - Positional arguments **and** keyword arguments
  - Clear distinction between interactive and non-interactive use
- Accessible option to get hands on with a "real" programming language (e.g., C, C++, Java, etc.)

Python for numerical calculus

- Also called “Scientific Development Stack”
  - `Numpy`: arrays and array operations
  - `Scipy`: numerical recipes
  - `pandas`: data structures and analysis
  - `Sympy`: symbolic mathematics
  - `Matplotlib`: plotting Matlab-like
  - `seaborn`: data visualization
  - [Pro] `Numba`: C interface and multicore processing
  - [Pro] `NumbaPro`: multicore GPU processing
- Some non-numerical extras
  - `BeautifulSoup`: HTML parsing and web scraping
  - `Selenium`: web browser automation and web scraping
  - `slate` and `PDFMiner`: PDF parsing and content extraction

Quick & dirty starter: Data Types

- Built-ins: `list`, `tuple`, `dict`, `str`, `int`, `float`
  - Strings (`str`): `greeting = 'ciao'`
  - Integers (`int`): `a = 1`
  - Floating point (`float`): `b = 1.`
  - Lists (`list`): `c = [a, b]`
  - Tuples (`tuple`): `d = (a, b)`
  - Dictionaries (`dict`): `e = {'a': 1, 'b': 1.}`
- Numpy: `array`
  - 2-by-2 matrix: `I = numpy.array([[1., 0.], [0., 1.]])`
  - column vector: `x = numpy.array([[3], [4]], dtype=float)`
  - 1-dimensional vector: `y = numpy.array([2., 3.])`
- pandas: `DataFrame`
  - `df = pandas.DataFrame(numpy.array([[1., 0.], [1., 2.]]))`

Quick & dirty starter: Operators

Operation	Operator
Sum	`+, -`
Multiplication	`*`
Division	`/`
Floor division	`//`
Exponentiation	`**`
Matrix creation	`A = numpy.array([[1, 2], [3, 4]])`
Matrix transpose	`A.T`
Matrix product	`A @ x`
Matrix inverse	`scipy.linalg.inv(A)`
Matrix determinant	`scipy.linalg.det(A)`
Diagonal extraction	`scipy.linalg.diag(A)`
Sum along the rows	`numpy.sum(A, axis=0)`

Quick & dirty starter: More operators

Relation	Operator
Greater than	`>`
Smaller than	`<`
Greater than or equal to	`>=`
Smaller than or equal to	`<=`
Equal (numeric)	`==`
Equal (boolean)	`is`
Not equal (numeric)	`!=`
Not equal (boolean)	`is not`
And	`and`
Or	`or`
Not	`not`

Quick & dirty starter: Control flow

Conditional statements

```
if x > 10:
    # do this thing
elif x = 10:
    # do this other thing
else:
    # screw it
```

Repeat with stopping rule

```
while crit > eps:
    # do this
    # include new value for ’crit’
```

Iterate over a range of values

```
for i in range(N):
    # something to do with ’i’
```

- Code blocks are delimited with indentation (4 spaces, no tabs)!
- Code block starts with a colon in the non-indented line
- Keep your code clean and organized, i.e., **pretty**

Quick & dirty starter: Functions

- Simple “wrappers”

```python
def printZip(a, b):
    for ai, bi in zip(a, b):
        print(ai, bi, sep=': ')
```

- Recursive definitions

```python
def fibonacci(n):
    """ Returns the n-th element of the Fibonacci sequence. """
    if n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return Fibonacci(n-1) + Fibonacci(n-2)
```

Quick & dirty starter: Classes (objects)

```
import numpy
from numpy.random import normal as randn

class AR1process:

    def __init__(alpha=0, rho, sigma=1):
        self.alpha = alpha
        self.rho = rho
        self.sigma = sigma

    def simulate(T, x0):
        x = numpy.zeros((T, 1))
        x[0] = x0
        for t in range(1, T):
            x[t] = alpha + rho * x[t-1] + randn(scale=sigma)
        return x
```

Quick & dirty starter: Exceptions

```python
def fibonacci(n):
    """ Returns the n-th element of the Fibonacci sequence. """
    if not isinstance(n, int):
        raise TypeError('Input must be an integer.')
    if n <= 0:
        raise ValueError('Input must be a positive integer.')
    # ... ... ...
```

```python
try:
    # try this
except ValueError:
    # do this if ValueError (avoid
      catch-all 'except' clauses)
else:
    # run this if 'try' gave no
     error
```

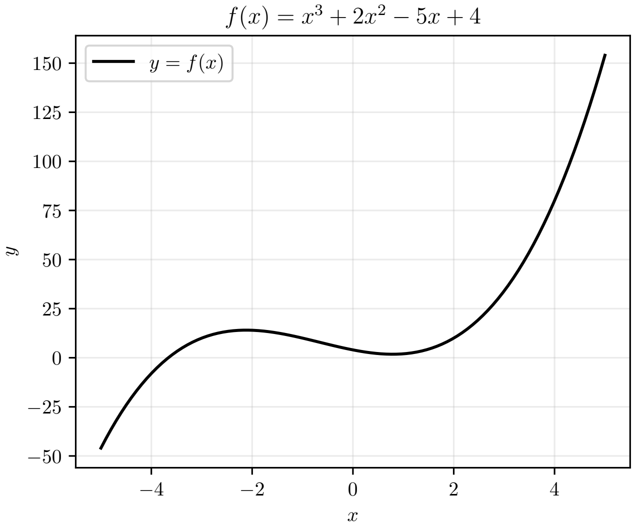
```python
try:
    # try this
except ValueError:
    # do this if ValueError (avoid
      catch-all 'except' clauses)
finally:
    # run this regardless of the
     above
```

Quick & dirty starter: example

We want to get the plot of some function $f(x)$ with $x \in X$.

```
import numpy, matplotlib
import matplotlib.pyplot as plt
matplotlib.rc('text', usetex=True)
f = lambda x: a * x**3 + b * x**2 + c * x + d
a, b, c, d = 1, 2, -5, 4
x = numpy.linspace(-5, 5, num=100)
fig, ax = plt.subplots(figsize=[5, 4])
ax.plot(x, f(x), linewidth=1.5, color='black', label='$y=f(x)$')
ax.grid(alpha=0.25); ax.legend()
ax.set_xlabel('$x$'); ax.set_ylabel('$y$')
ax.set_title('$f(x) = x^3 + 2 x^2 - 5 x + 4$')
fig.savefig('./plot_f.pdf')
```

Quick & dirty starter: example



IDEs and workflow

A simple text editor would suffice, but...

- We want to have nice syntax highlighting
- We want to manage multiple `.py` files at once
- We want hints/reminders about functions we use
- We want to see plots on the fly
- We want to test interactively (often)
- We want a state-of-the-art debugger (more often)
- We want to do all the above in one place

We can choose:

- Most basic: `Sublime Text` / `Atom`
- Most straightforward: `Spyder`
- Most complete: `PyCharm`

IDEs and workflow

A simple text editor would suffice, but...

- We want to have nice syntax highlighting
- We want to manage multiple `.py` files at once
- We want hints/reminders about functions we use
- We want to see plots on the fly
- We want to test interactively (often)
- We want a state-of-the-art debugger (more often)
- We want to do all the above in one place

We can choose:

- Most basic: `Sublime Text` / `Atom`
- Most straightforward: `Spyder`
- Most complete: `PyCharm`

IDEs and workflow

A simple text editor would suffice, but...

- We want to have nice syntax highlighting
- We want to manage multiple `.py` files at once
- We want hints/reminders about functions we use
- We want to see plots on the fly
- We want to test interactively (often)
- We want a state-of-the-art debugger (more often)
- We want to do all the above in one place

We can choose:

- Most basic: `Sublime Text` / `Atom`
- Most straightforward: `Spyder`
- Most complete: `PyCharm`

IDEs and workflow

A simple text editor would suffice, but...

- We want to have nice syntax highlighting
- We want to manage multiple `.py` files at once
- We want hints/reminders about functions we use
- We want to see plots on the fly
- We want to test interactively (often)
- We want a state-of-the-art debugger (more often)
- We want to do all the above in one place

We can choose:

- Most basic: `Sublime Text` / `Atom`
- Most straightforward: `Spyder`
- Most complete: `PyCharm`

Best practices

- Interactive VS non-interactive use
  - Matlab works exclusively in interactive mode (console is crucial)
  - Python works faster in non-interactive mode (no console at all)
  - Test interactively, run in non-interactive mode
  - Use the debugger (a programmer's creature comfort)
- Functions, functions and functions again (even "wrappers")
- Always ask: "Can I refactor this into a `Class`?"
- **Google is your best friend**

Warm up for next time

Recall Pavoni's material?

$$V(x) = \max_{y \in D(x)} u(y, x) + \beta V(x')$$

- Maths: this defines a contraction mapping over a compact set. There exists a unique fixed point $V(\cdot)$
- The fixed point can be obtained by some iterative (numerical) algorithm
- This is going to be our workhorse for the remaining TA sessions

Warm up for next time

```
import numpy

x = numpy.linspace(-5, 5, num=100)
a, b, c = -1, 5, 0
f = lambda x: a * x**2 + b * x + c

i_max = numpy.argmax(f(x))  # this an index for the grid!!
x_max = x[i_max]            # point we're interested in
f_max = numpy.max(f(x))     # gives 'f' at x_max

print(i_max)  # 74 (i.e., the 75th point on grid)
print(x_max)  # 2.4747474747...
print(f_max)  # 6.2493623099...
```

Exercises

- Relax: no problem sets in this course, but...
- Don't relax too much: better to keep up with the TA sessions (you'll thank me later)
- Have a look at the folder `exercises_ta1`: see you next week and try your best to do those exercises
- For any question: send me an email or pass by anytime

Appendix: A Bit of Computer Science

- A processor can crunch only zeros and ones
- Any integer in base 10 has a unique binary *representation* (e.g., $2 \to 10, 4 \to 100, 8 \to 1000, 16 \to 10000, \ldots$)
- Integers? No problem... if "small"!
- Any non-integer is **represented** as a *floating-point* number (the decimal point *floats* with the exponent):

$$1.2345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10}_{\text{base}}{}^{\overbrace{-4}^{\text{exponent}}}$$

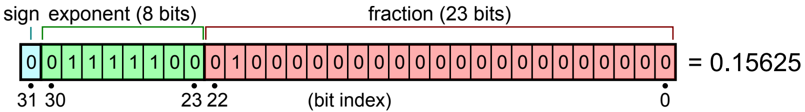
  - Numbers with "small" amount of decimals? Sure
  - Numbers with too many digits? ...Damn!

Appendix: A Bit of Computer Science

Problems arise depending on number of significant digits:

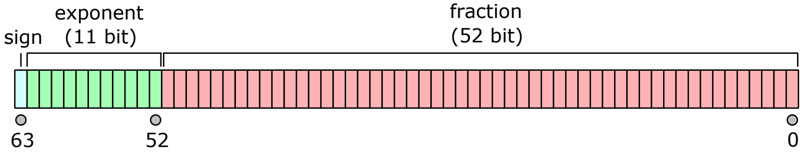
- *Single-precision* floating-point numbers (`numpy.float32`)

  [https://en.wikipedia.org/wiki/Single-precision_floating-point_format]



- *Double-precision* floating-point numbers (`float` and `numpy.float64`)

  [https://en.wikipedia.org/wiki/Double-precision_floating-point_format]



Appendix: A Bit of Computer Science

- Try these

```python
import math
math.sqrt(3) ** 2
math.sin(math.pi)
math.cos(math.pi/2)
7/3 - 4/3 - 1  # machine epsilon for numpy.float64 numbers
```

- These are operations affected by *rounding-off errors*
- Rounding-off errors are everywhere (if you use non-integers)
- We will see some tricks to reduce their impact (but this course is not about those tricks). For example:
  - Never invert a matrix with `scipy.linalg.inv()`
  - Get as many zeros around as possible (e.g., QR decomposition)
  - Use sparse matrices where applicable/reasonable

Appendix: Useful References

- Numerical Methods
  - Burden and Faires (2016): https://www.cengage.com/c/numerical-analysis-10e-burden
- Documentation and User Manuals
  - Numpy: https://docs.scipy.org/doc/numpy-1.14.0/reference/
  - Scipy: https://docs.scipy.org/doc/scipy-1.0.0/reference/
  - Matplotlib: https://matplotlib.org/contents.html
- Your question, already answered
  - https://stackoverflow.com/questions/tagged/numpy
- Python for Macroeconomics (and other people too!)
  - QuantEcon: https://lectures.quantecon.org/py/
  - Data Science: https://github.com/jakevdp/PythonDataScienceHandbook
- Let your PC do the heavy lifting for you, whatever that is
  - https://automatetheboringstuff.com/