Econ 204B: Section 1

Ryan Sherrard

University of California, Santa Barbara

11 January 2019

Preliminaries

Contact:

- Website: rsherrardecon.com
- OH: TBD

Problem Sets:

- They will be due periodically throughout the quarter (I expect 5-6)
- .Tex solutions; include attach code (Use Verbatim or Listings package for simplicity)

Text Books / References (not required):

- Recursive Methods in Economic Dynamics (Stokey, Lucas, & Prescott)
- Recursive Macroeconomic Theory, 3rd Ed. (Ljungqvist & Sargent)
- Quantitative Economics (Sargent & Stachurski)

Overview of Econ 204B

- Welcome back! I hope everyone's break was refreshing and devoid of economics
- In this class we will expand on some things we saw last quarter
  - e.g. more general framework (more than 2 periods for an agent)
  - e.g. add in uncertainty (idiosyncratic, aggregate)
- We will see some new ways of framing these problems (recursively) and new ways of solving them
- In doing so, we will find that analytic solutions can be difficult to find (or impossible); we will learn how to take these models to the computer

The new method / technique / tool that we will employ is **dynamic programming**.

- Instead of thinking about a bunch of optimization problems every period, we'll think about making decisions in a single period assuming we'll make the right decision in the future

- Suppose we wanted to get from SB to NYC (via car) by minimizing distance travelled; further, assume that we knew the optimal route passed through Omaha
  - we can break up the problem by finding the optimal route to Omaha, and then we would optimize from Omaha to NYC (Bellman's principle of optimality)

- Instead of solving for a sequence of savings / consumption decisions (for each possible state) we can solve for a *policy function* that tells us what to do each period given some state

- The beginning of this class is meant to be a "data bootcamp:" we want to be able to know what is going on in an economy
- How does this tie in with theory? We want a systematic way of describing the economy that isn't simply a statistical description
- We will write down models that formalize the interactions of economic agents and use data to illuminate the important mechanisms driving results
- There are many ways of attacking this; to give a couple examples . . .
  - calibrate the model to be consistent with certain moments, then see if other moments match up
  - introduce other features that might explain any irregularities

Three Types of Economies

1. Arrow-Debreu Equilibria: agents trade contingent claims to fund each other in any state of the world over time (called Arrow-Debreu securities)
2. Sequential Equilibria: agents make savings and consumption decisions *each period* given the state observed today
3. Recursive Equilibria: "simplifies" the sequential problem down to a decision today (given that we will optimize in the future)

What is interesting is that, under certain circumstances, the sequential and recursive economies have the same solution (AD economies can be thought of as a particular type of sequential problem).

Sequential vs Recursive Problem

The sequential problem is what we are used to solving from Econ 204A:

- Infinitely-lived households
- No aggregate uncertainty
- Make a sequence of savings decisions

$$\max_{\{c_t\}_0^\infty} \sum_{t=0}^{\infty} \beta^t u(c_t) \qquad s.t. \qquad \sum_{t=0}^{\infty} c_t \leq \sum_{t=0}^{\infty} e_t$$

With uncertainty, we would need to solve for such a sequence for every possible sequence of states. The recursive problem transforms our equilibrium objects to function space and simplifies the problem

$$V(a) = \max_c \{u(c) + \beta \mathbb{E}(V(a'))\} \qquad s.t. \qquad c + a' = (1 + r)a + e$$

$\Longrightarrow$ Solve for a value function and decision rule (policy function) once and apply it to all periods

- There are some mathematical preliminaries that we'll have to hammer out to truly understand what's going on with this recursive formulation
- We'll get there in due time, but we'll also see that the recursive formulation makes it easy to take these problems to the computer
- There are many different programming languages that we might want to use: **Python**, Julia, R, Matlab, Fortran, C++, etc.
- The language doesn't really matter, good coding knows no one language; there are a lot of similarities between these options
- Use what you want, but I'll be focusing on Python because I know it and I think it'll be useful for you

Python Indroduction

- Python is an open-source, high-level, interpreted, object oriented, procedural programming language
- Though slower than lower-level languages like C++ or Fortran, it's a very active language useful for many situations
  - useful for us: data manipulation, statistics, numerical optimization
- That it's interpreted essentially means writing (and reading) code is much easier and intuitive
  - modules like "Numba" have been written to make Python very fast (comparable to compiled languages)
- Python allows object-oriented programming, which can be very useful / intuitive

Installing Python

- "Hard" method: download directly from Python website
- Easy method: get from a distribution (e.g. Anaconda)
- Anaconda is a Python distribution that comes packaged with a lot of nice tools and modules (e.g. Numpy, Scipy, and Matplotlib)
- Once downloaded, you might notice that it comes with an Integraded Development Environment (IDE): Spyder (or Jupyter Notebook if you like being a bad person)
- There are many IDEs like Spyder which essentially streamline the process of writing, running, and viewing the output of your code
  - use whatever floats your boat

Installing Other Packages

- At some point you might need to install a module that you want to use
- I recommend the use of the "preferred installer program," pip (it comes with Anaconda)
- You can pip install directly from the terminal:

```
pip install pkgname
```

- Next, let's jump right into using Python

Crash Course: Modules

- To get you started, I'll give you some lines of code to build off of

- At the top of your program, you'll need to import your modules

```
import datetime
import matplotlib.pyplot as plt
import numpy as np
import pandas_datareader.data as wb
```

1) **datetime**: Provides routines for manipulating dates and times in a simple way (dates are surprisingly tricky)
2) **matplotlib.pyplot**: Provides easy and intuitive commands to make high quality figures; pyplot provides an environment similar to MatLab
3) **numpy**: Essential package for numerical and scientific computing
4) **pandas datareader** Provides easy to use data access.

- You are first asked to get some data from FRED; this can be done a few different ways
  - download it from the website and read it in with a function
  - download it directly from the website

```
data = np.genfromtxt('filename.csv', delimiter=',')

start = datetime.datetime(1947,1,1)
end   = datetime.datetime(2016,12,31)

rGDP = np.ravel( wb.DataReader('GDPC1','fred',start,end)
                                                .as_matrix() )
```

- You are also asked to work with the data; e.g. calculating the percentage growth of real GDP

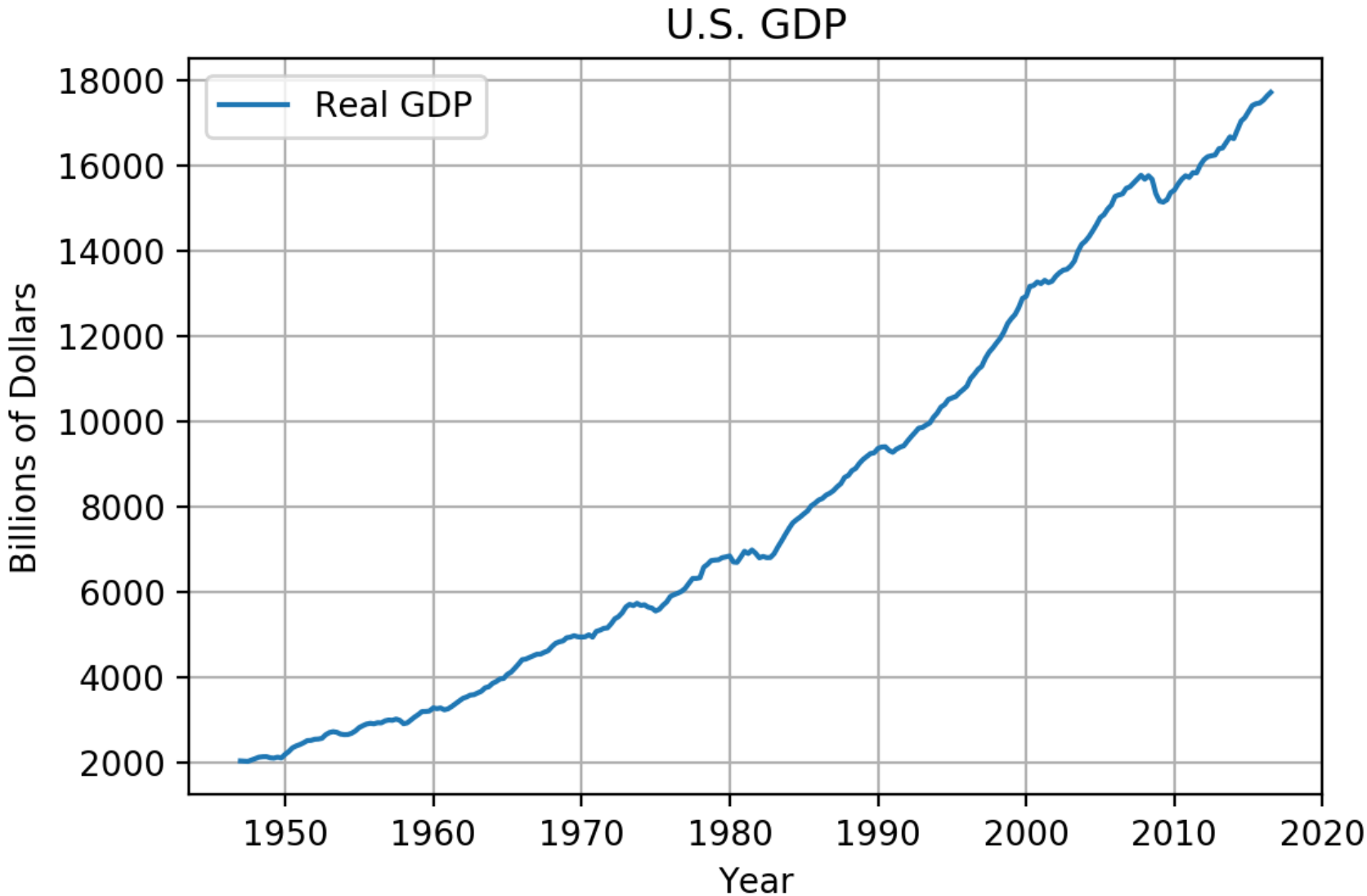
```
year1 = np.linspace(1947.0,2016.5,279)

rGDP_qpc  = np.zeros(len(year1))

for i in range(len(rGDP_qpc)-1):
    rGDP_qpc[i] = (rGDP[i+1]-rGDP[i])/rGDP[i]
```

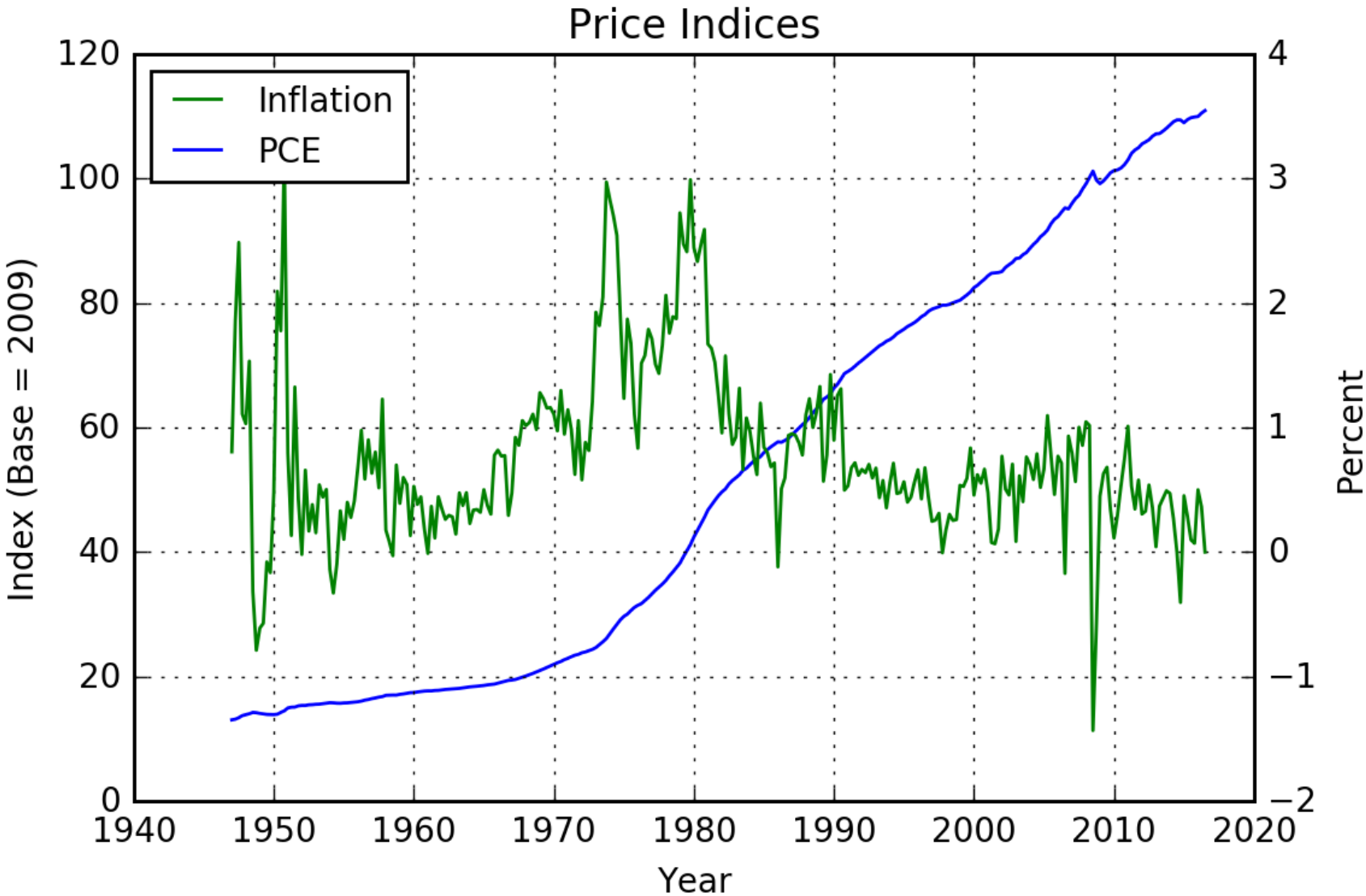
- Now, what about graphing? I recommend the use of Matplotlib (presented next) or GGPlot2

```
rGDP = rGDP[:-1]
plt.figure()
plt.plot(year1,rGDP,label = 'Real GDP')
plt.grid()
plt.xlabel('Year')
plt.ylabel('Billions of Dollars')
plt.legend(loc='best',prop={'size':10})
plt.title('U.S. GDP')
plt.savefig('GDP.png',dpi=200, bbox_inches='tight')
```


U.S. GDP
Real GDP
Billions of Dollars
18000
16000
14000
12000
10000
8000
6000
4000
2000
1950
1960
1970
1980
1990
2000
2010
2020
Year

- What about multiple axes? Here it'll be recommended to use an object oriented approach ...

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.plot(year1,PCE,'b-',linewidth=1,label='PCE')
ax2 = ax.twinx()
ax2.plot(year1,pi_PCE,'g-',linewidth=1,label='Inflation')
ax2.plot(2009,0,'b-',linewidth=1,label='PCE')
ax.grid()
ax.set_xlabel('Year')
ax.set_ylabel('Index (Base = 2009)')
ax.set_title('Price Indices')
ax2.legend(loc='upper left',prop={'size':10})
plt.savefig('PCE-pi.png',dpi=200, bbox_inches='tight')
```


Price Indices
Inflation
PCE
Index (Base = 2009)
Percent
Year
120
100
80
60
40
20
0
4
3
2
1
0
−1
−2
1940
1950
1960
1970
1980
1990
2000
2010
2020