Macro II

Professor Griffy

UAlbany

Spring 2022

Welcome to Macro II

- My name: Ben Griffy.
- Email: bgriffy@albany.edu.
- Office Hours: TTh 10:30-11:30am in-person (for now).
- Teaching Assistant:
  - George Siton
  - Email: gsiton@albany.edu
  - Office Hours WTh 1-3:00pm online.

Course Overview

- What is this course about?
  1. Learning about key questions addressed by macroeconomists.
  2. Developing advanced tools to address those questions.
  3. Seeing how others have worked to address those questions.
- I will emphasize quantitative (computational) work.
- I see this as a complement to Adrian's more analytical course.

Course Requirements

- Materials:
  1. We will mostly rely on Ljundqvist and Sargent.
  2. Feel free to get an older version (problems might differ between versions).
  3. There are some suggested textbooks as well as free resources in the syllabus.
- Tools:
  1. One (or two, or three, or ...) programming language.
  2. My suggestion: Matlab, Julia, (Fortran or C++, or maybe Python) for dynamic programming/numerical optimization.
  3. Python and Julia are free and we have a campus Matlab license.
  4. We probably won't do any data analysis, so don't worry about R/Stata.
  5. LaTeXfor homework and projects.
- Please note: best resource for class materials is my website: https://www.bengriffy.com

Grading

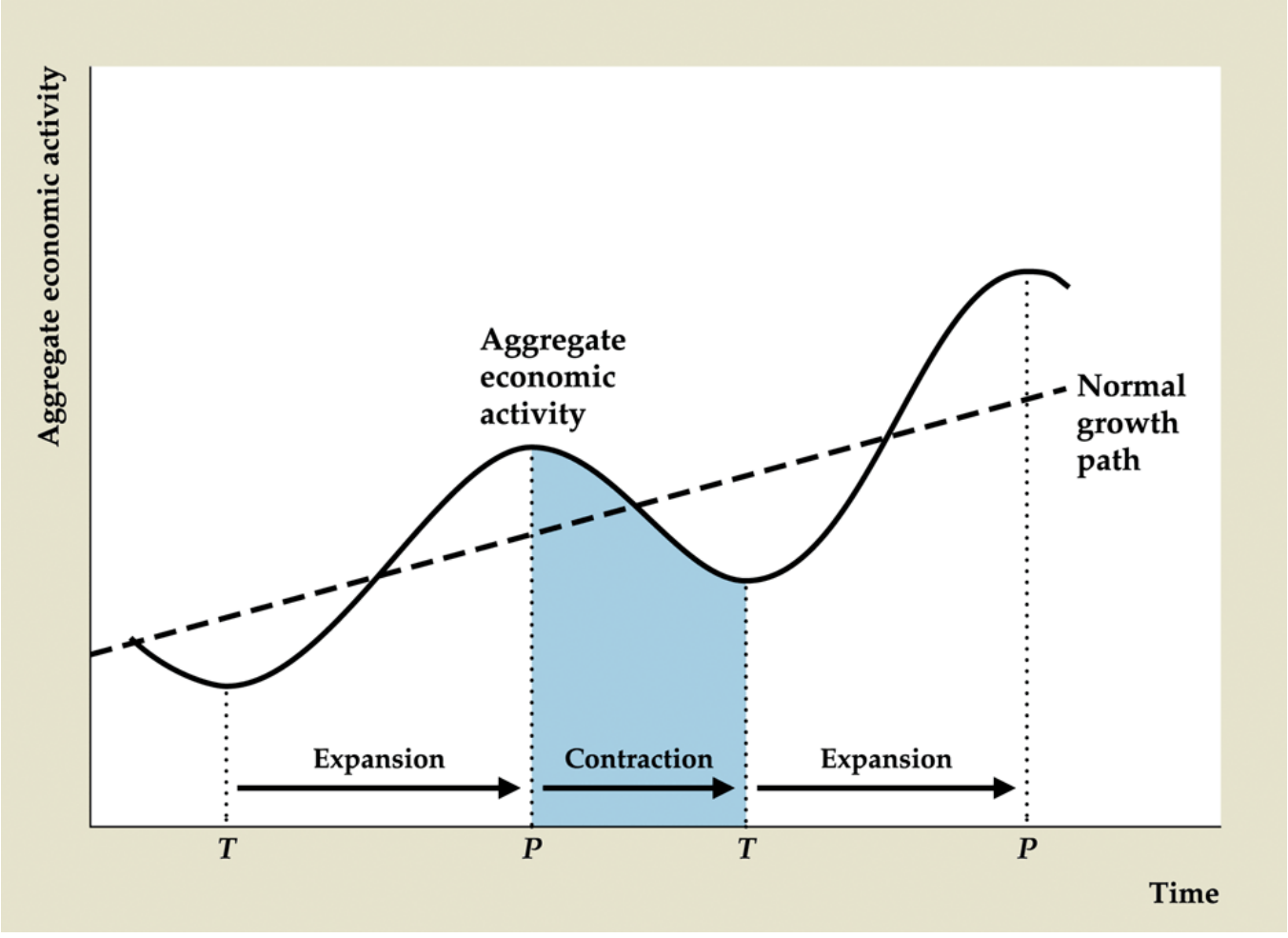
- Semi-weekly homework: 30%. I will (intend to) give you a homework assignment at least every other week.
- I am going to require that the homework be turned in through the campus cluster.
- Midterm (30%):
- Final (40%):
- My philosophical view of first-year grades: the comps are your real "grade" in the course, so I am unlikely to fail anyone.
- I just want to see that you understand the material and are putting in the effort required to pass comps.

What do Macroeconomists Study?

- Macroeconomics:
  - what are the aggregate consequences of individual decisions for

    1. growth;
    2. business cycles.
- this is not exhaustive: labor markets, finacial markets, international trade, etc.
- To do this, we specify models with
  1. individuals and firms with assumptions about preferences and technology;
  2. an equilibrium that describes how individual actions will aggregate.
- In this course, we will focus on using models to understand business cycles.
- Today: start by exploring the data.

What is a business cycle?



- What do you notice?

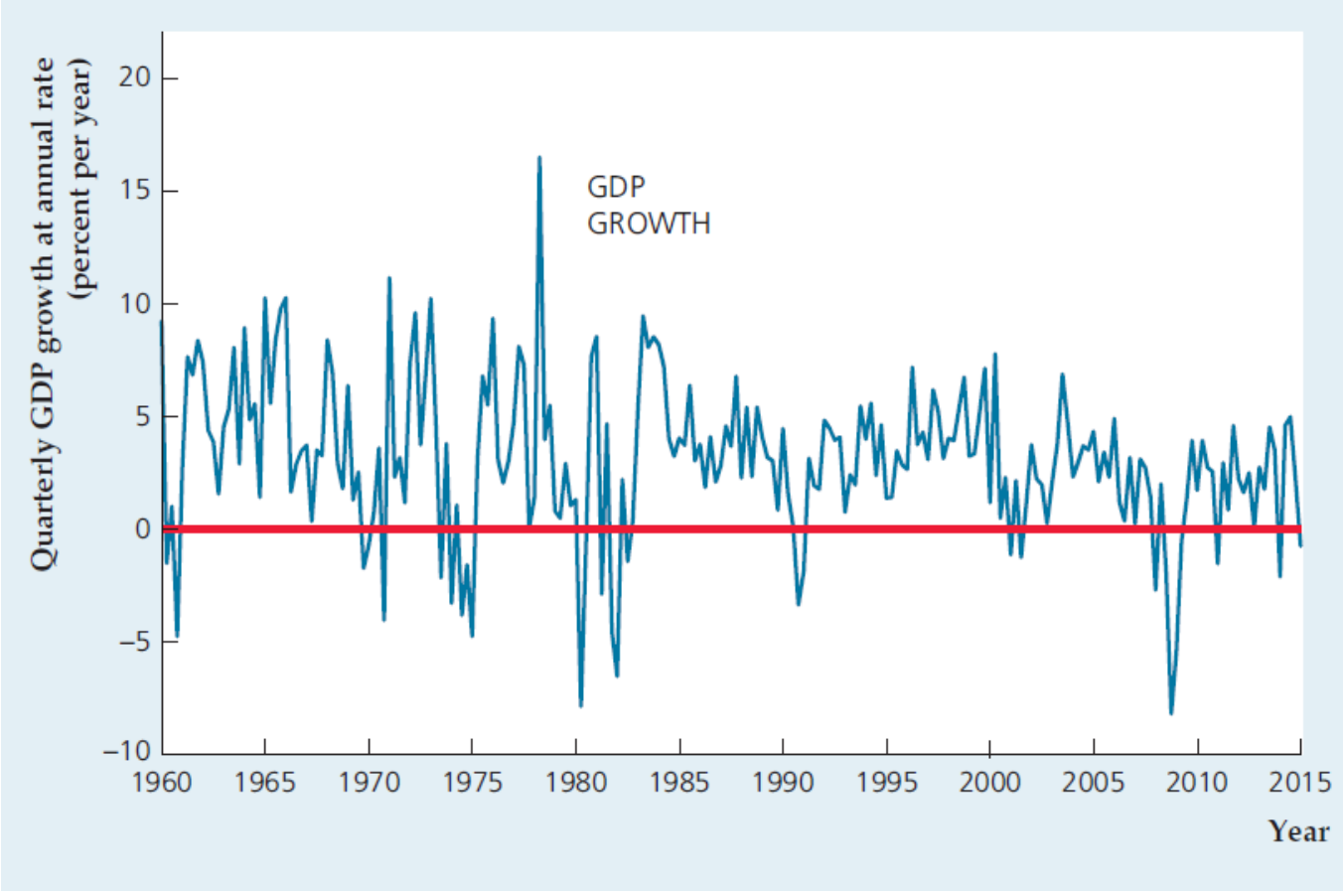
Empirical Regularities (aka Stylized Facts)

- To guide our models, we look at "empirical regularities" or "stylized facts."
- These are patterns we observe in the data that
  1. motivate the construction of our models and/or
  2. that we seek to understand.
- For example, Kydland and Prescott (1982): "The equilibrium growth model is modified and used to explain the
  1. cyclical variances of a set of economic time series,
  2. the covariances between real output and the other series,
  3. and the autocovariance of output."

Empirical Regularities and Business Cycles

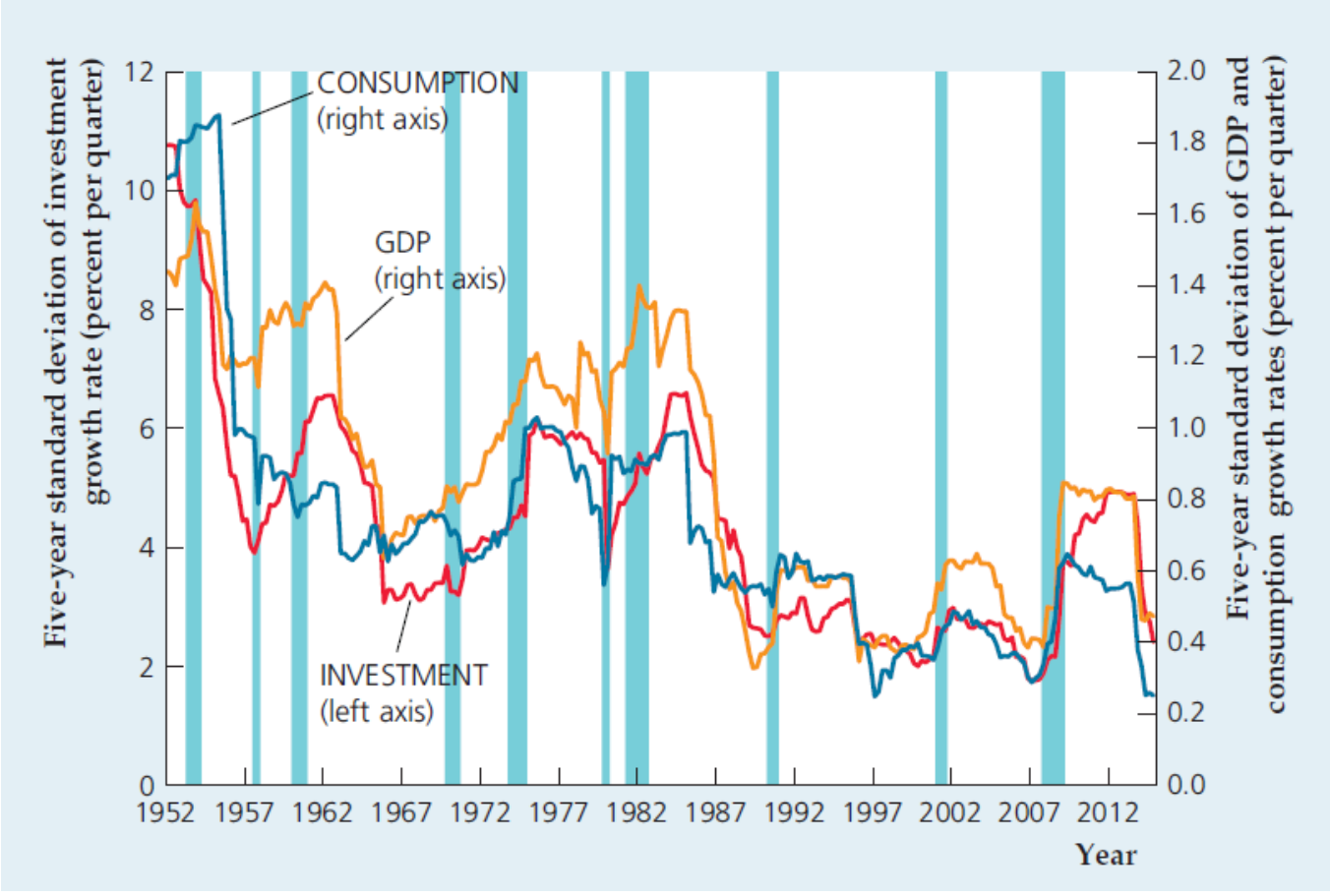
- Business cycles are not identical, but do have commonality.
- We want our models to capture what is common
- then perhaps use them to explain what differs.
- Cyclicality: when GDP goes up, does a variable move **up** (procyclical) or **down** (countercyclical)?
- Some procyclical variables:
  - production, consumption, investment, and employment.
  - inflation, real wages, and money growth.
- First group generally a function of GDP, should always move with GDP.
- Second group, predictions less clear.
- A macro model might seek to explain the movement of this second group.
- Let's look at the time series.

GDP Growth 1960-2015



- What are some patterns you notice?

GDP and Components Volatility



- How does volatility between these series compare?

Cyclicality Summary

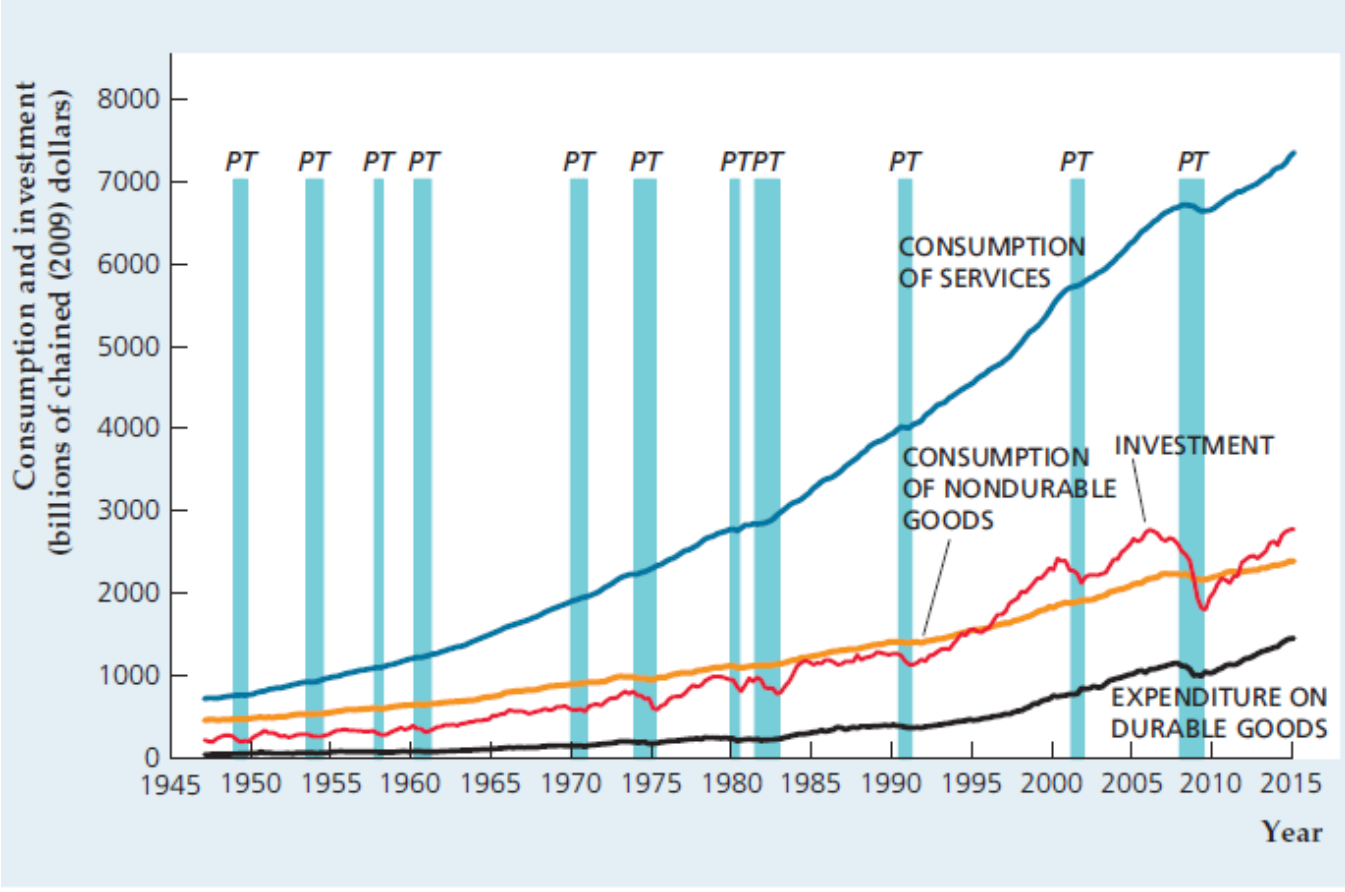
SUMMARY 10

The Cyclical Behavior of Key Macroeconomic Variables (The Business Cycle Facts)

Variable	Direction	Timing
Production		
Industrial production	Procyclical	Coincident
Durable goods industries are more volatile than nondurable goods and services		
Expenditure		
Consumption	Procyclical	Coincident
Business fixed investment	Procyclical	Coincident
Residential investment	Procyclical	Leading
Inventory investment	Procyclical	Leading
Government purchases	Procyclical	—[a]
Investment is more volatile than consumption		
Labor Market Variables		
Employment	Procyclical	Coincident
Unemployment	Countercyclical	Unclassified[b]
Average labor productivity	Procyclical	Leading[a]
Real wage	Procyclical	—[a]
Money Supply and Inflation		
Money supply	Procyclical	Leading
Inflation	Procyclical	Lagging
Financial Variables		
Stock prices	Procyclical	Leading
Nominal interest rates	Procyclical	Lagging
Real interest rates	Acyclical	—[a]

- Want our business cycle models to be consistent with all of these regularities.
- Often unrealistic, depends on question.

More NIPA Components

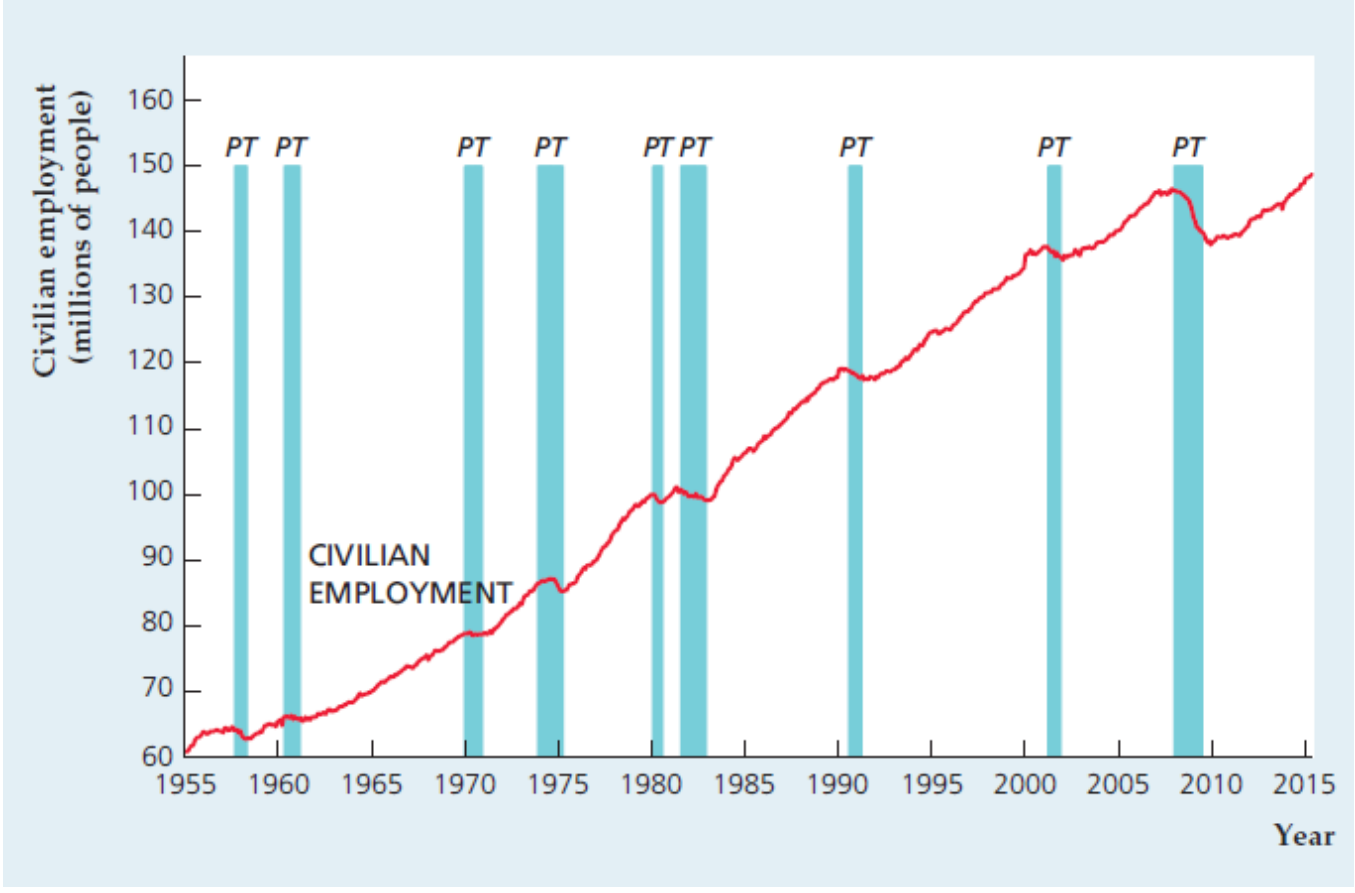


- What do you notice?

Labor Markets

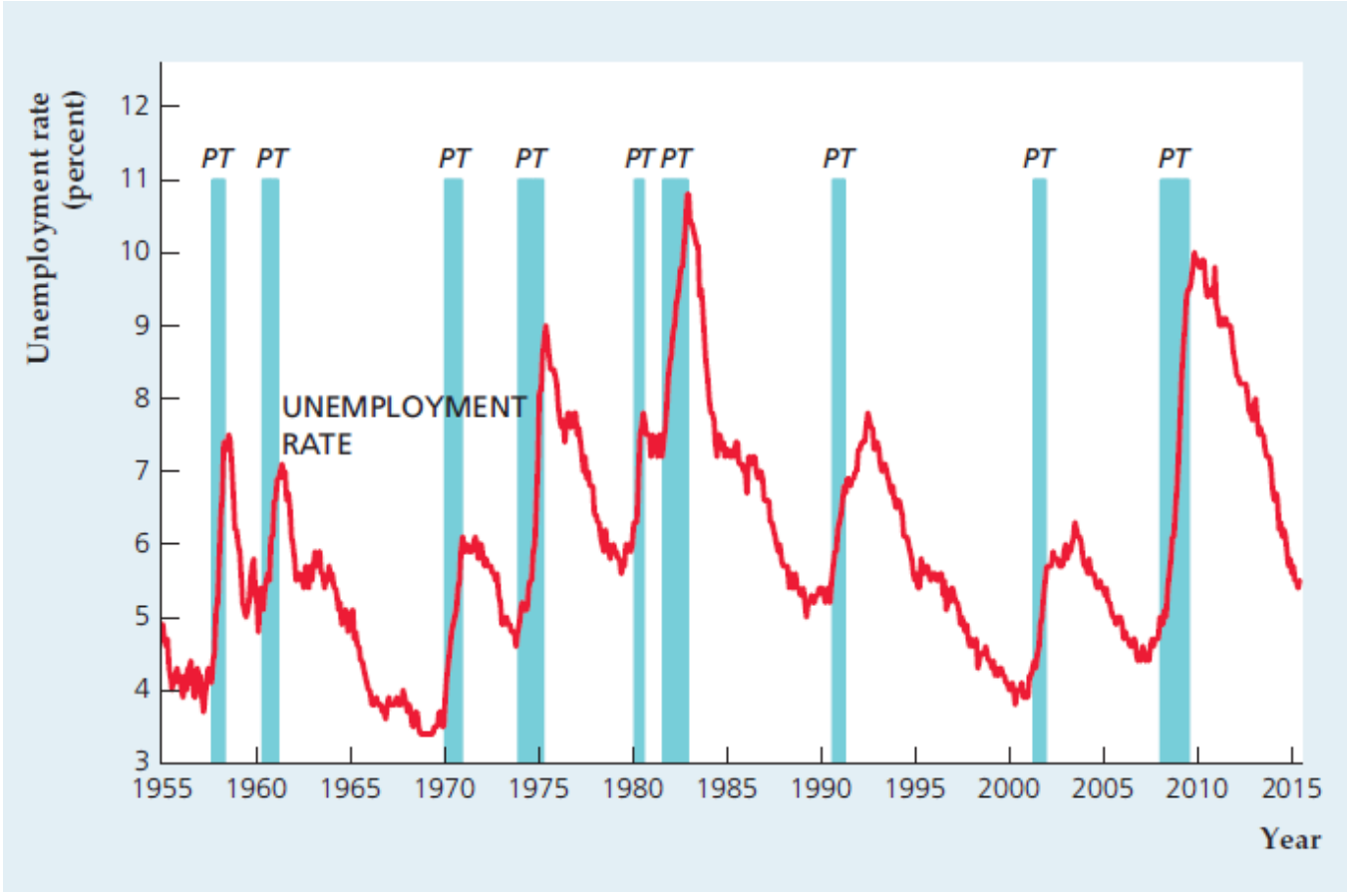
- In addition to NIPA components (cons., inv., gov't spend, net exports), labor markets important.
- Unemployment: consistent feature of recessions.
- More labor market indicators:
  - Job-Finding Rate: probability that someone searching (generally unemployed) for a job finds one.
  - Job Separation Rate: probability that someone currently employed loses their job.
- Nuance: unemployment and employment are stocks;
- job-find and job sep. are flows.
- To understand fluctuations in unemployment rate, we need to understand fluctuations in flows!

Employment



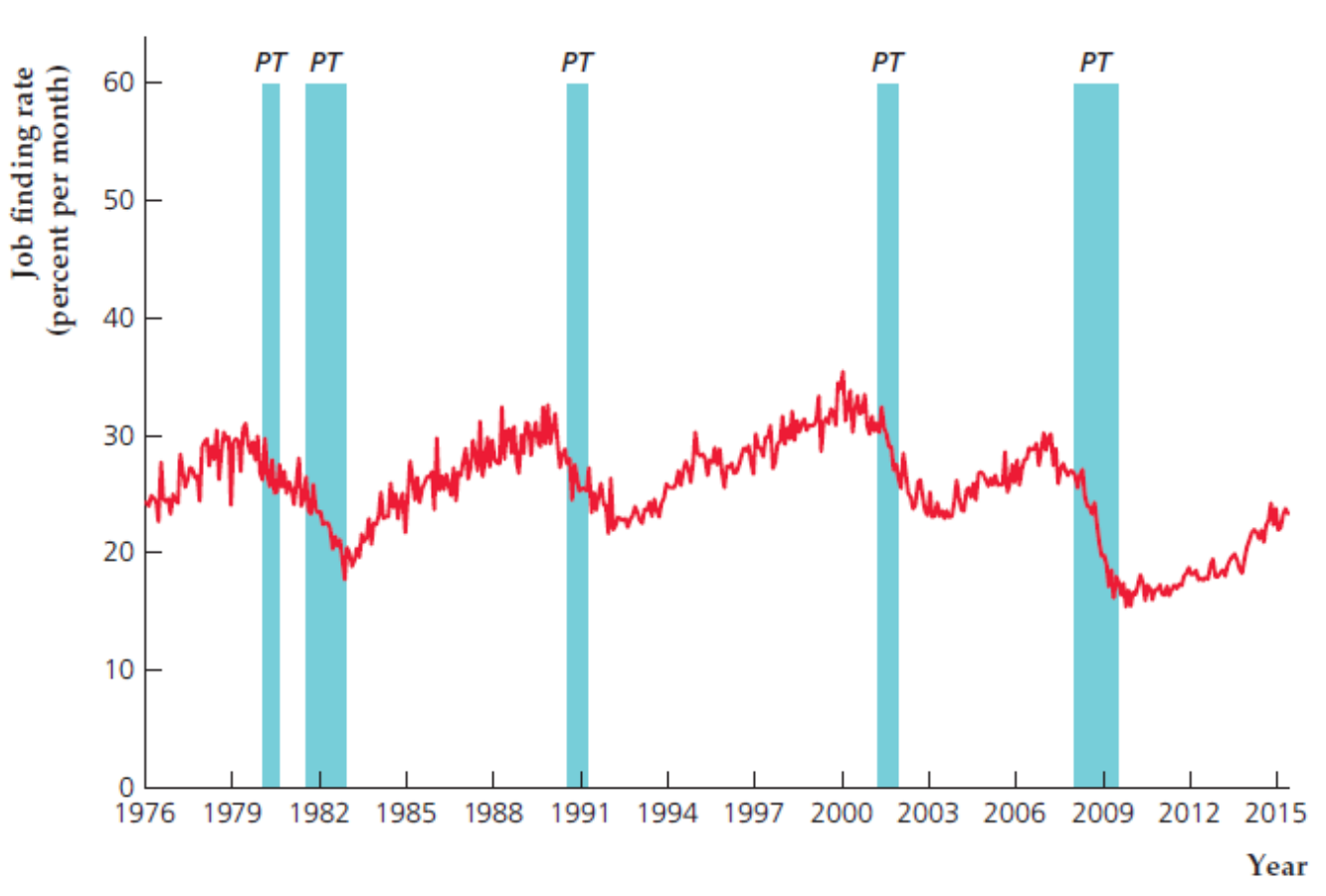
- What is important to note about this figure? (hint: look left)
- Why does the denominator matter?

Unemployment



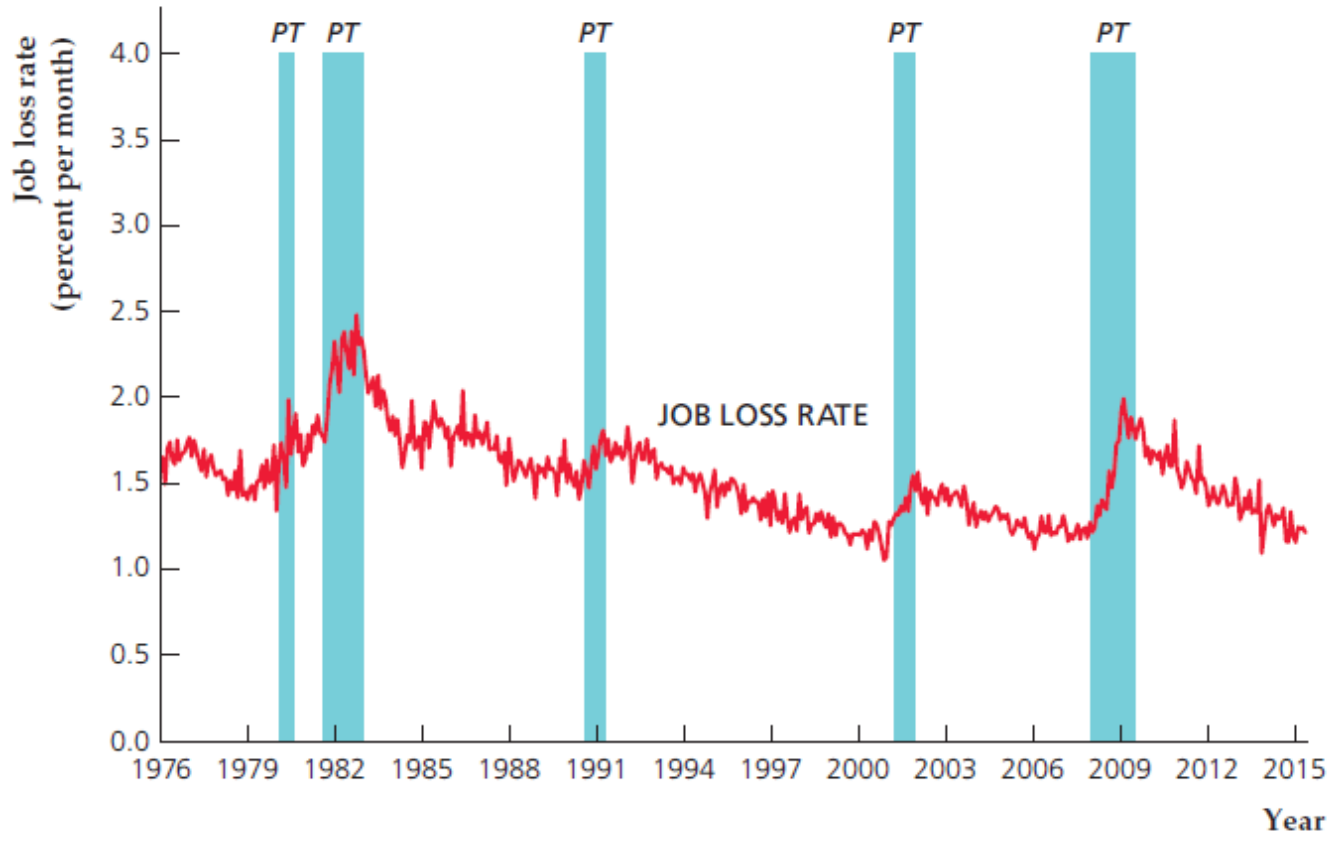
- What do you notice?

Job Finding Rate



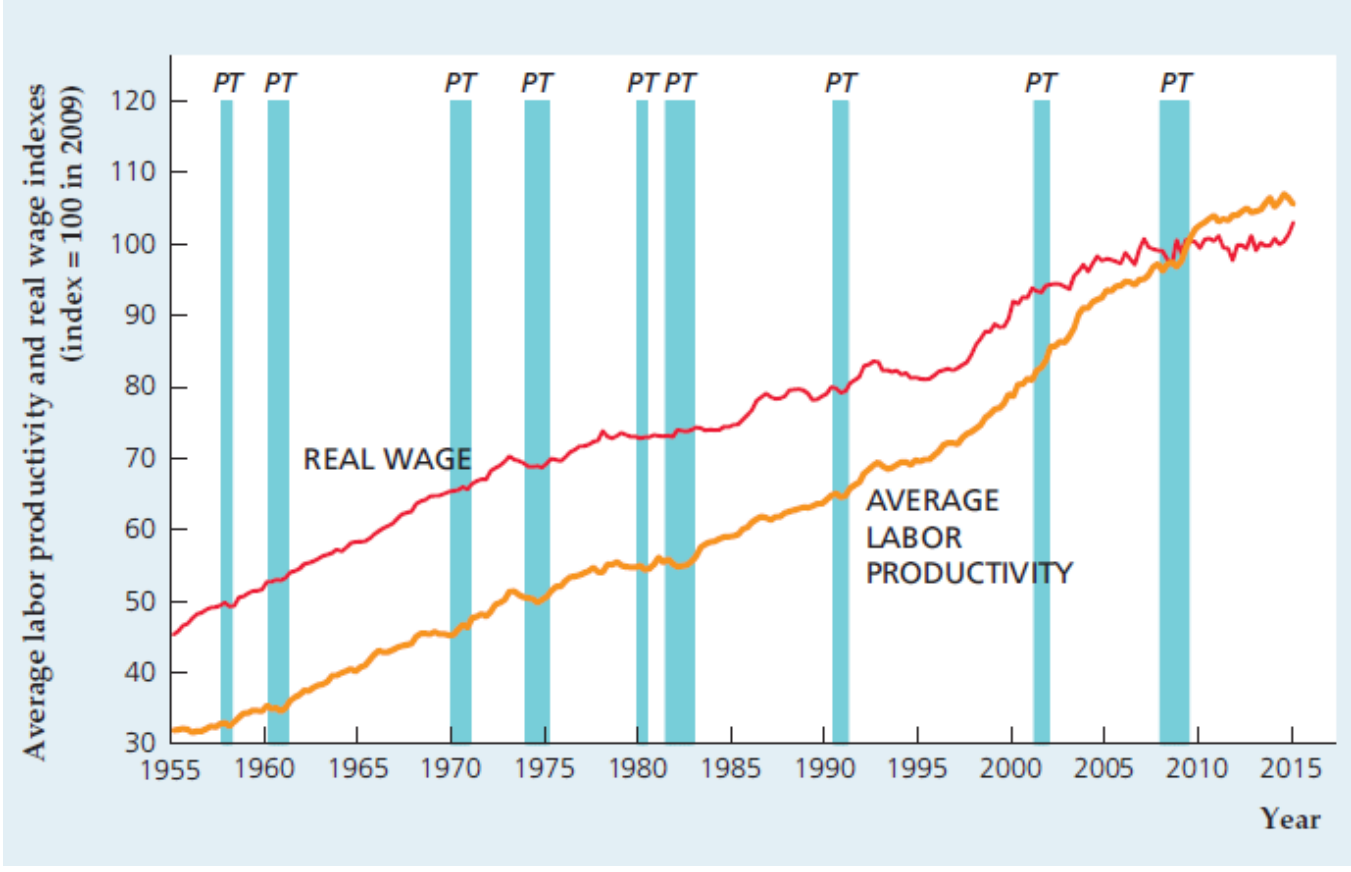
- How does this coincide with what we saw for unemployment?
- Does this contribute to the cyclicality of unemployment?

Job Sepration Rate



- How does this coincide with what we saw for unemployment?
- What would you say is driving unemployment?

Productivity vs. Real Wage

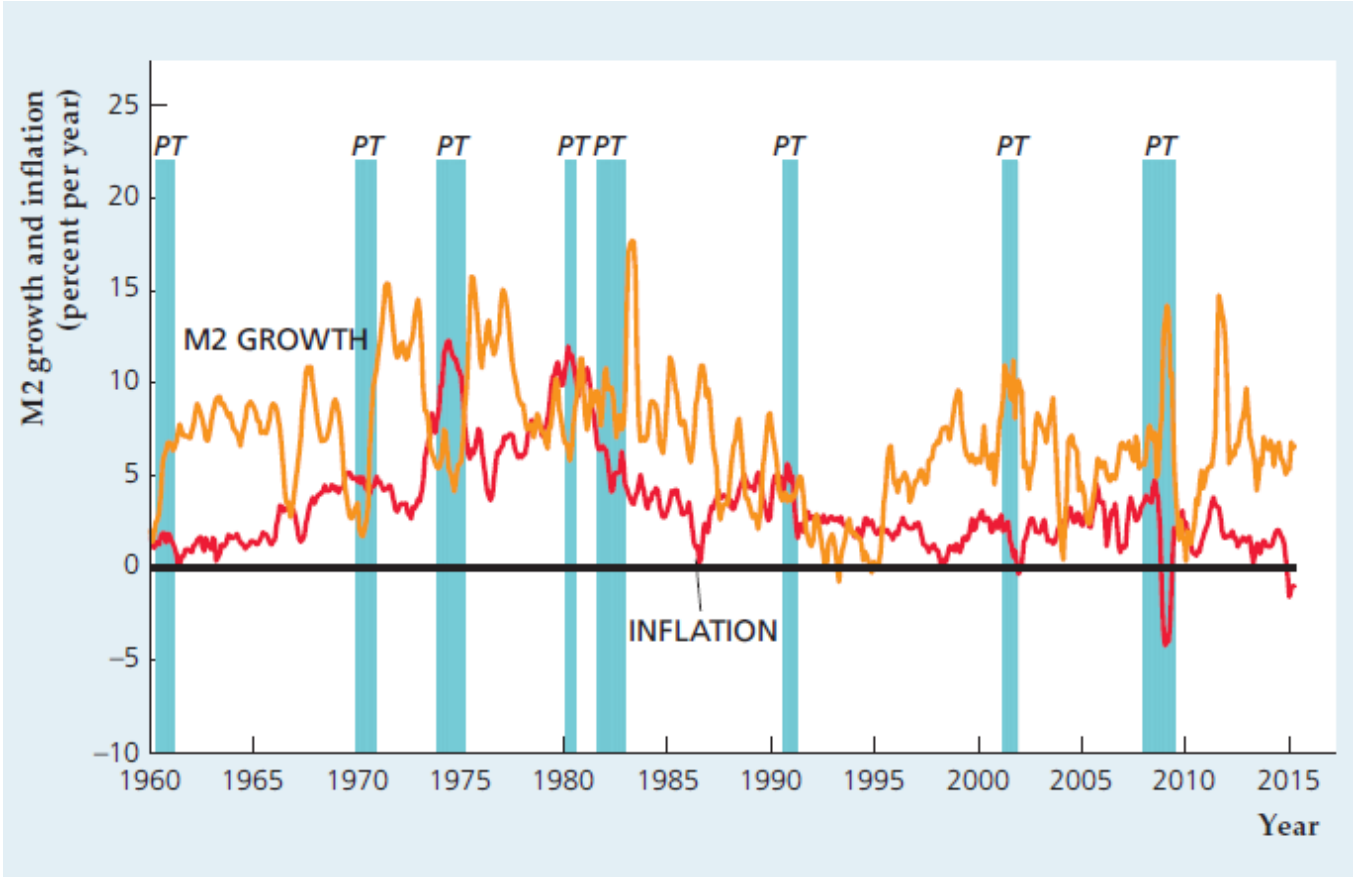


- What do you notice?

Questions

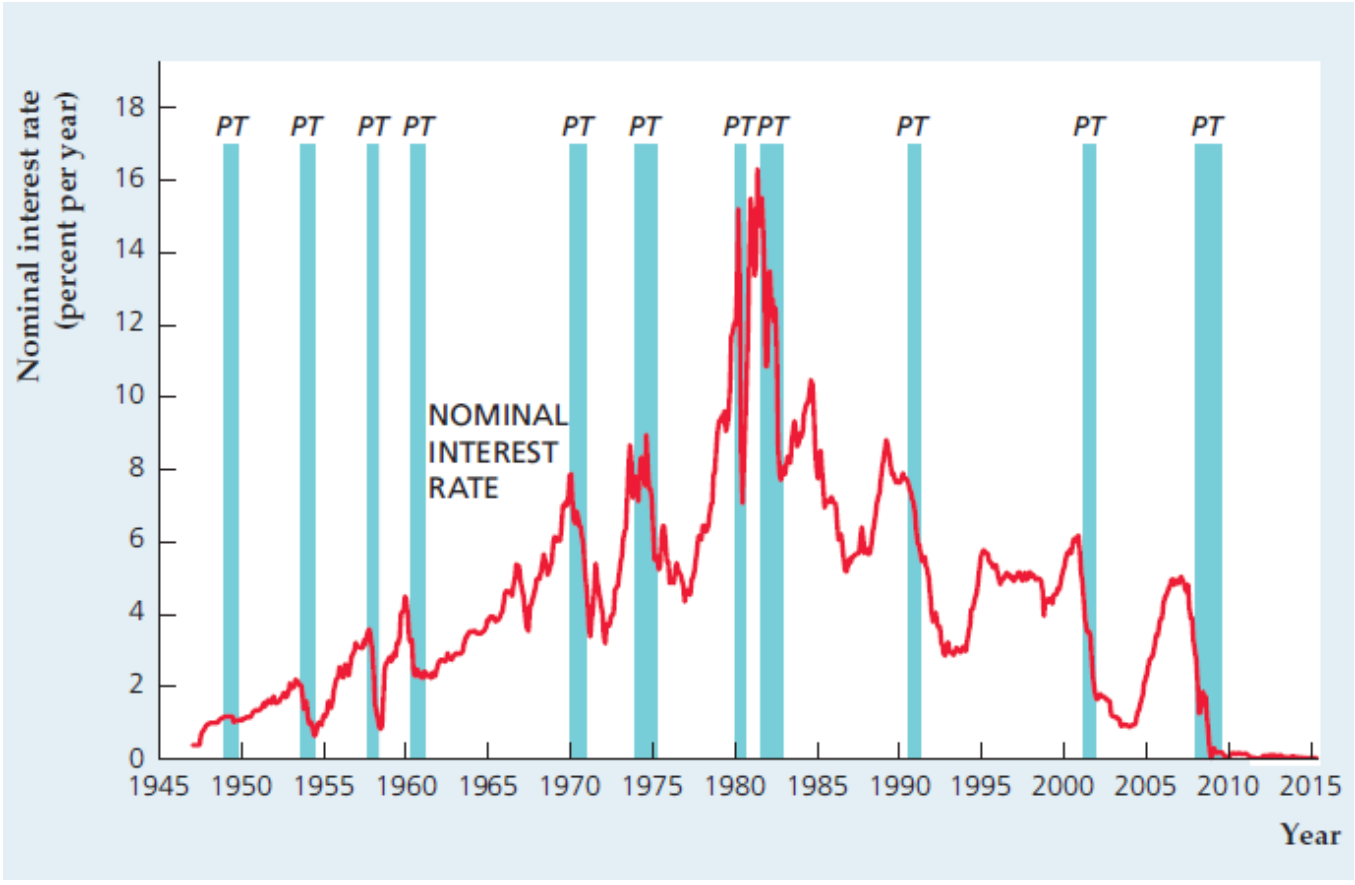
- Perfect competition: real wage $=$ marginal productivity.
- Marginal productivity hard to measure $\approx$ average productivity.
- Do they move together over the business cycle?
- What could this mean?

Money Growth and Inflation



- What do you notice?

Nominal Interest Rates



- How does this differ?

Some equations

- Velocity equation:

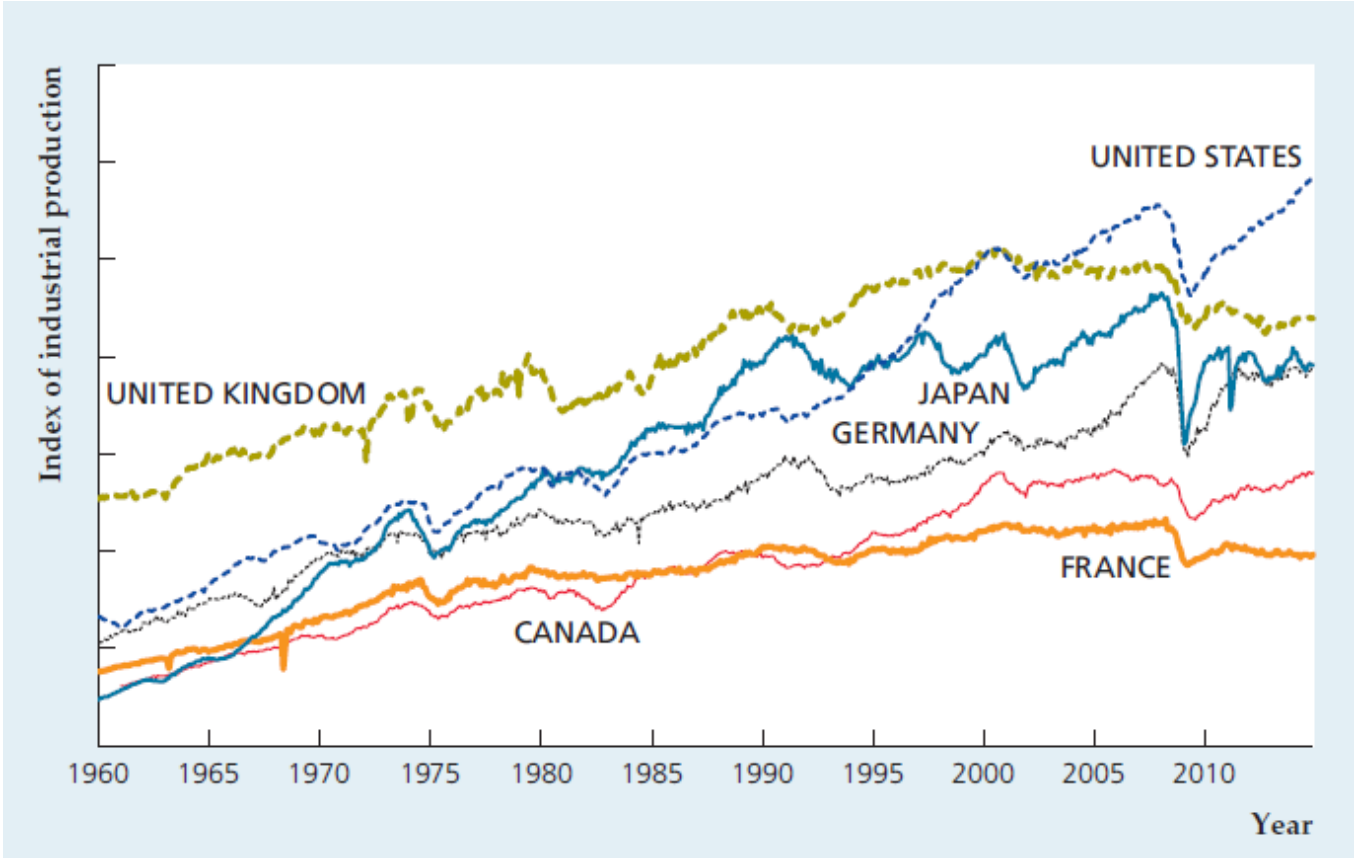
$$\%\Delta M + \%\Delta V = \%\Delta P + \%\Delta Y \tag{1}$$

- What's going on?
- Fisher Equation:

$$i \approx r + \pi \tag{2}$$

- What does this mean for the real interest rate?

Industrial Production across Countries



- What changes do you notice?

Empirical Regularities

- These are just some things that we observe in the data.
- I should emphasize that macro
  - is more broad than just business cycles or growth
- Macroeconomists study inequality, labor markets, technology, money, wealth, assets, etc.
- Business cycles are a convenient jumping off point.

Programming Basics

- Modern macro: most models not solvable by hand:
  "The era of closed-form solutions for their own sake should be over. Newer generations get similar intuitions from computer-generated examples than from functional expressions,"
  -Jose-Victor Rios-Rull (JME, 2008).
- But we're rarely taught anything about programming.

Matlab

- Focus on Matlab (because it's most widely used), but should apply for nearly any language.
- Variables:
  - store numbers, vectors (1 x n or n x 1), arrays or matrices (m x n)
  - can perform operations on these variables (addition, subtraction, multiplication)
- Functions:
  - take variables as inputs
  - return something useful
- Indexing:
  - way to access a specific column or row.
  - i.e., say we have a variable X. X(1,2) accesses row 1, column 2 (x and y are flipped for indexing).
  - X(:,2) accesses all rows of column 2.

Simple Example: Regression

```
>> X =  [1 1 4; 1 8  9; 1 3 7]

X =

     1     1     4
     1     8     9
     1     3     7

>> y = [4; 8; 2]

y =

     4
     8
     2
```

- Specified X and y matrices in Matlab. Now regress:

```
>> bbeta = inv((X'*X))*(X'*y)

bbeta =

   10.0000
    2.0000
   -2.0000
```

- i.e., intercept is 10 and coefficients are 2 and -2.

Where programming gets tricky

- When we're working by hand, we solve for a functional expression, $y = f(x)$.
- Generally cannot do this for dynamic programming.
- Instead, approximate function $y_i \approx \hat{f}(x_i)$.
- i.e., find a function $\hat{f}(x)$ that approximates $f(x)$ and evaluate it at a pre-specified set of points, $x_i, i = 1, ..., N$.
- An example: what is $f(k) = k^{\alpha}$ for $k = 1, ..., 10$?

Production function

```
>> aalpha = 0.5

aalpha =

    0.5000
```

Figure: α parameter

```
>> k = linspace(1,10,10)

k =

     1     2     3     4     5     6     7     8     9    10
```

Figure: Grid of k values

- Need α, grid of k values, and function.

```
>> prodFun = @(x)x.^(aalpha)

prodFun =

  function handle with value:

    @(x)x.^(aalpha)
```

Figure: Prod. function k^{α}

```
>> prodFun(k)

ans =

    1.0000    1.4142    1.7321    2.0000    2.2361    2.4495    2.6458    2.8284    3.0000    3.1623
```

Figure: Output at k grid

- Note: specified prod function for x, not k.
- Important: already specified k as grid.

Last example

- That wasn't so bad. But what about a more dynamic problem?
- Consider this problem:
- An agent needs to split consumption over the life-cycle (2 periods).

$$\max_{a} u(c_1) + \beta u(c_2) \tag{3}$$

$$\text{s.t.} \quad c_1 + a \leq m \tag{4}$$

$$c_2 \leq (1 + r)a \tag{5}$$

- We can solve this by hand for any value of m.
- How would we solve on computer for any m? We need
  - $u(x)$, β, r
  - m grid, a grid.
- What we will do: set up a c_1 and c_2 grid.
- These depend on values of m and decisions of a.

Preliminaries

```
>> r = 0.04

r =

    0.0400
```

Figure: r parameter

```
>> bbeta = 1/(1 + r)

bbeta =

    0.9615
```

Figure: β value

- Need α, grid of k values, and function.

```
>> mGrid = linspace(1,10,10)

mGrid =

     1     2     3     4     5     6     7     8     9    10
```

Figure: m grid

```
>> aGrid = linspace(1,10,10)

aGrid =

     1     2     3     4     5     6     7     8     9    10
```

Figure: Output at a grid

- Note: specified prod function for x, not k.
- Important: already specified k as grid.

Consumption Grids

```
>> c1Grid = repmat(mGrid.',1,10) - repmat(aGrid,10,1)

c1Grid =

     0    -1    -2    -3    -4    -5    -6    -7    -8    -9
     1     0    -1    -2    -3    -4    -5    -6    -7    -8
     2     1     0    -1    -2    -3    -4    -5    -6    -7
     3     2     1     0    -1    -2    -3    -4    -5    -6
     4     3     2     1     0    -1    -2    -3    -4    -5
     5     4     3     2     1     0    -1    -2    -3    -4
     6     5     4     3     2     1     0    -1    -2    -3
     7     6     5     4     3     2     1     0    -1    -2
     8     7     6     5     4     3     2     1     0    -1
     9     8     7     6     5     4     3     2     1     0
```

- m is in rows, 1 to 10. a choice in columns, 1 to 10.

```
>> c2Grid = (1 + r).*repmat(aGrid,10,1)

c2Grid =

    1.0400    2.0800    3.1200    4.1600    5.2000    6.2400    7.2800    8.3200    9.3600   10.4000
    1.0400    2.0800    3.1200    4.1600    5.2000    6.2400    7.2800    8.3200    9.3600   10.4000
    1.0400    2.0800    3.1200    4.1600    5.2000    6.2400    7.2800    8.3200    9.3600   10.4000
    1.0400    2.0800    3.1200    4.1600    5.2000    6.2400    7.2800    8.3200    9.3600   10.4000
    1.0400    2.0800    3.1200    4.1600    5.2000    6.2400    7.2800    8.3200    9.3600   10.4000
    1.0400    2.0800    3.1200    4.1600    5.2000    6.2400    7.2800    8.3200    9.3600   10.4000
    1.0400    2.0800    3.1200    4.1600    5.2000    6.2400    7.2800    8.3200    9.3600   10.4000
    1.0400    2.0800    3.1200    4.1600    5.2000    6.2400    7.2800    8.3200    9.3600   10.4000
    1.0400    2.0800    3.1200    4.1600    5.2000    6.2400    7.2800    8.3200    9.3600   10.4000
    1.0400    2.0800    3.1200    4.1600    5.2000    6.2400    7.2800    8.3200    9.3600   10.4000
```

- Make matrices conformable, i.e., size of c1Grid

Value Function

```
>> ValFun = uFun(c1Grid) + bbeta.*uFun(c2Grid)

ValFun =

     -Inf + 0.0000i   0.7042 + 3.1416i   1.7872 + 3.1416i   2.4693 + 3.1416i   2.9715 + 3.1416i   3.3700 + 3.1416i   3.7005 + 3.1416i   3.9831 + 3.1416i   4.2299 + 3.1416i   4.4490 + 3.1416i
   0.0377 + 0.0000i     -Inf + 0.0000i   1.0941 + 3.1416i   2.0638 + 3.1416i   2.6839 + 3.1416i   3.1469 + 3.1416i   3.5182 + 3.1416i   3.8289 + 3.1416i   4.0963 + 3.1416i   4.3312 + 3.1416i
   0.7309 + 0.0000i   0.7042 + 0.0000i     -Inf + 0.0000i   1.3707 + 3.1416i   2.2784 + 3.1416i   2.8592 + 3.1416i   3.2951 + 3.1416i   3.6466 + 3.1416i   3.9422 + 3.1416i   4.1976 + 3.1416i
   1.1363 + 0.0000i   1.3973 + 0.0000i   1.0941 + 0.0000i     -Inf + 0.0000i   1.5852 + 3.1416i   2.4537 + 3.1416i   3.0074 + 3.1416i   3.4235 + 3.1416i   3.7599 + 3.1416i   4.0435 + 3.1416i
   1.4240 + 0.0000i   1.8028 + 0.0000i   1.7872 + 0.0000i   1.3707 + 0.0000i     -Inf + 0.0000i   1.7606 + 3.1416i   2.6019 + 3.1416i   3.1358 + 3.1416i   3.5367 + 3.1416i   3.8612 + 3.1416i
   1.6472 + 0.0000i   2.0905 + 0.0000i   2.1927 + 0.0000i   2.0638 + 0.0000i   1.5852 + 0.0000i     -Inf + 0.0000i   1.9088 + 3.1416i   2.7303 + 3.1416i   3.2490 + 3.1416i   3.6380 + 3.1416i
   1.8295 + 0.0000i   2.3136 + 0.0000i   2.4804 + 0.0000i   2.4693 + 0.0000i   2.2784 + 0.0000i   1.7606 + 0.0000i     -Inf + 0.0000i   2.0372 + 3.1416i   2.8436 + 3.1416i   3.3503 + 3.1416i
   1.9836 + 0.0000i   2.4960 + 0.0000i   2.7035 + 0.0000i   2.7570 + 0.0000i   2.6839 + 0.0000i   2.4537 + 0.0000i   1.9088 + 0.0000i     -Inf + 0.0000i   2.1504 + 3.1416i   2.9449 + 3.1416i
   2.1172 + 0.0000i   2.6501 + 0.0000i   2.8858 + 0.0000i   2.9801 + 0.0000i   2.9715 + 0.0000i   2.8592 + 0.0000i   2.6019 + 0.0000i   2.0372 + 0.0000i     -Inf + 0.0000i   2.2517 + 3.1416i
   2.2349 + 0.0000i   2.7836 + 0.0000i   3.0400 + 0.0000i   3.1624 + 0.0000i   3.1947 + 0.0000i   3.1469 + 0.0000i   3.0074 + 0.0000i   2.7303 + 0.0000i   2.1504 + 0.0000i     -Inf + 0.0000i
```

- lifetime val $= u(c1Grid) + \beta \times u(c2Grid)$

```
>> ValFun(c1Grid<=0) = -inf

ValFun =

     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf
   0.0377     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf
   0.7309   0.7042     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf
   1.1363   1.3973   1.0941     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf
   1.4240   1.8028   1.7872   1.3707     -Inf     -Inf     -Inf     -Inf     -Inf     -Inf
   1.6472   2.0905   2.1927   2.0638   1.5852     -Inf     -Inf     -Inf     -Inf     -Inf
   1.8295   2.3136   2.4804   2.4693   2.2784   1.7606     -Inf     -Inf     -Inf     -Inf
   1.9836   2.4960   2.7035   2.7570   2.6839   2.4537   1.9088     -Inf     -Inf     -Inf
   2.1172   2.6501   2.8858   2.9801   2.9715   2.8592   2.6019   2.0372     -Inf     -Inf
   2.2349   2.7836   3.0400   3.1624   3.1947   3.1469   3.0074   2.7303   2.1504     -Inf
```

- Can't eat less than nothing

Maximize

```
>> ValFun(c1Grid<=0) = -inf

ValFun =

      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
    0.0377      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
    0.7309    0.7042      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
    1.1363    1.3973    1.0941      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
    1.4240    1.8028    1.7872    1.3707      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
    1.6472    2.0905    2.1927    2.0638    1.5852      -Inf      -Inf      -Inf      -Inf      -Inf
    1.8295    2.3136    2.4804    2.4693    2.2784    1.7606      -Inf      -Inf      -Inf      -Inf
    1.9836    2.4960    2.7035    2.7570    2.6839    2.4537    1.9088      -Inf      -Inf      -Inf
    2.1172    2.6501    2.8858    2.9801    2.9715    2.8592    2.6019    2.0372      -Inf      -Inf
    2.2349    2.7836    3.0400    3.1624    3.1947    3.1469    3.0074    2.7303    2.1504      -Inf
```

- Now, best savings decisions for each level of money?

```
>> [maxValFun,polInd] = nanmax(ValFun,[],2)
```

```
>> maxValFun'

ans =

      -Inf    0.0377    0.7309    1.3973    1.8028    2.1927    2.4804    2.7570    2.9801    3.1947

>> polInd'

ans =

     1     1     1     2     2     3     3     4     4     5
```

- We picked the best decision for each initial value!

Next Time

- In class: Outline models in modern macro.
- You guys:
  - Install the appropriate programming languages.
  - Look at my handout (online) about accessing the cluster.
  - Solve 2-period model by hand, then solve on computer for $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$ where $\sigma = 2$.
- Due next Thursday (2/3).