An Introduction to Computational Macroeconomics

Lecture 1

John Stachurski

June – July 2022

Introduction

Lectures:

- Wednesday 13:00 – 16:40 (in person, no zoom)
- Location: Seminar Room 515 (Economics building)

Lecturer: John Stachurski

- Email: `john.stachurski@anu.edu.au`
- Office: Daisuke Oyama's office (10th Floor)
- Office hours: Email me (for meeting times, not questions)

Resources

Course homepage:
https://github.com/jstac/tokyo_2022_coursework

- please check regularly

Course notes: Dynamic Programming Volume 1 by John Stachurski and Thomas J. Sargent

- Course notes will change! Print only small sections.
- Please help me fix typos / squash bugs

Programming resources https://lectures.quantecon.org/

Supplementary reading:

- *Abstract Dynamic Programming* by Dimitri Bertsekas, Athena Scientific, 2018
- *Recursive Macroeconomic Theory* by Lars Ljungqvist and Thomas J. Sargent, MIT Press, 2018, chapters 1-7
- *Recursive Methods in Dynamic Economics* by Nancy Stokey and Robert E. Lucas, Harvard University Press, 1989
- *Introduction to Real Analysis* by Robert Bartle and Donald Sherbert, Wiley, 2011
- *Analysis for Applied Mathematics* by Ward Cheney, Springer Science, 2013

Assessment

1. 40% **Programming Assignment** (details TBA)
   - To be submitted as a Jupyter notebook
   - Due date TBA
   - Will include programming and analysis
2. 60% **Final Exam** (20/7/2022 1 pm)
   - Analytical (no programming)

Topics

- Modern scientific computing
- Linear and nonlinear equations
- Markov chains
- Asset pricing
- Optimal stopping problems (job search, options, etc.)
- Markov control problems (savings, investment, inventories)
- Recursive preferences

Motivation

Why do we need computers / computational methods?

Example. A typical problem from undergraduate choice theory:

$$\max_{c_0, c_1} \{u(c_0) + \beta u(c_1)\} \tag{1}$$

subject to

$$0 \leqslant c_0, c_1 \quad \text{and} \quad c_1 \leqslant R(y_0 - c_0) \tag{2}$$

If $u' > 0, u'' < 0$ and $u'(0) = \infty$, then the unique solution obeys

$$u'(c_0) = \beta R u'(R(y_0 - c_0)) \quad \text{and} \quad c_1 = R(y_0 - c_0)$$

In general, undergraduate style optimization problems are **easy**

- All functions are differentiable
- Few choice variables (low dimensional)
- Concave (for max) or convex (for min)
- Interior solutions
- FOCs relatively simple

But grad micro/macro and research problems are harder...

- High dimensions
- Nonsmooth functions
- Discrete choices
- Boundary solutions
- No analytical solution for FOCs
- Neither concave nor convex — local maxima and minima

Most interesting research problems have these features

Example. Simple graduate macroeconomic problem:

Choose consumption at time $t = 0, 1, \dots$ to solve

$$\mathbb{E} \sum_{t=0}^{\infty} \beta^t u(c_t), \tag{3}$$

subject to

$$0 \leqslant a_{t+1} \leqslant R_t a_t + y_t - c_t \quad \text{and} \quad c_t \geqslant 0 \qquad (t \geqslant 0) \tag{4}$$

Even for this simple problem,

- Infinite dimensional because we choose $c_0, c_1, \dots$
- Occasionally binding constraints
- No analytical solution

Tools for this Course

We use two kinds of tools in this course

1. Programming / computation
2. Mathematical analysis

Let's talk a bit about them...

Mathematical Background

We use three branches of mathematics in this course:

1. Linear algebra
2. Real analysis
3. Probability theory

Linear algebra and Analysis

For solving equations and optimization problems

What is/are the solution/solutions to these equations?

1. $x = ax + b$
2. $x = x + 1$
3. $x^2 = 1$

Now let x be $n \times 1$ and A be $n \times n$

When does this **vector equation** have a unique solution ?

$$Ax = b$$

Linear algebra and Analysis

For solving equations and optimization problems

What is/are the solution/solutions to these equations?

1. $x = ax + b$
2. $x = x + 1$
3. $x^2 = 1$

Now let x be $n \times 1$ and A be $n \times n$

When does this **vector equation** have a unique solution ?

$$Ax = b$$

When does this vector equation in $\mathbb{R}^n$ have a unique solution?

$$x = Ax + b$$

When is the solution given by

$$x = (I - A)^{-1} b?$$

When does the **method of successive approximations** converge?

1. pick any $x_0 \in \mathbb{R}^n$
2. set $x_{n+1} = Ax_n + b$ for $n = 0, 1, \ldots$

When does this vector equation in $\mathbb{R}^n$ have a unique solution?

$$x = Ax + b$$

When is the solution given by

$$x = (I - A)^{-1} b?$$

When does the **method of successive approximations** converge?

1. pick any $x_0 \in \mathbb{R}^n$
2. set $x_{n+1} = Ax_n + b$ for $n = 0, 1, \ldots$

When does this vector equation in $\mathbb{R}^n$ have a unique solution?

$$x = Ax + b$$

When is the solution given by

$$x = (I - A)^{-1} b?$$

When does the **method of successive approximations** converge?

1. pick any $x_0 \in \mathbb{R}^n$
2. set $x_{n+1} = Ax_n + b$ for $n = 0, 1, \ldots$

Now let's make it a bit harder:

$$x = ((Ax)^{1/\gamma} + b)^{\gamma}, \qquad \gamma > 0$$

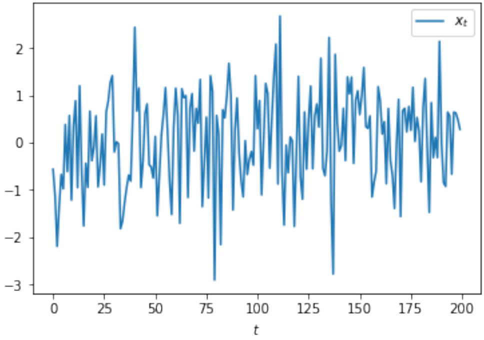
When does this have a solution?

Is it unique?

How would we compute it?

Probability

This sequence is IID



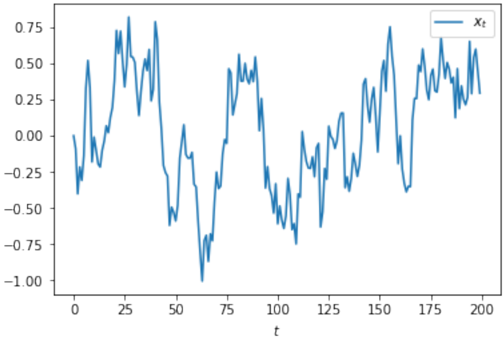
Does it follow that, for some function h, we have

$$\frac{1}{n}\sum_{t=1}^{n} h(X_t) \to \mathbb{E}h(X_t) \ \ ? \tag{5}$$

If so this is good:

- Right hand side is something we want to compute
- Left hand side is simulated from the model
- Convergence means we can use Monte Carlo...

This sequence is **not** IID



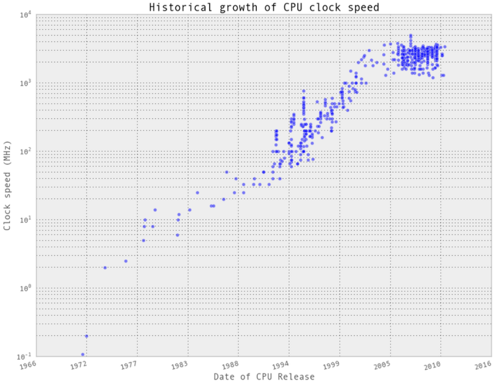
Is it possible that, for some function h, we have

$$\frac{1}{n}\sum_{t=1}^{n} h(X_t) \to \mathbb{E}h(X_t) \quad ? \tag{6}$$

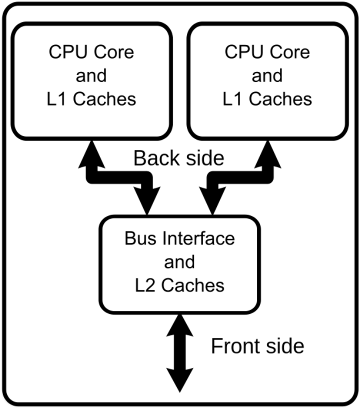
What conditions do we require?

Programming Background — Hardware

CPU frequency (clock speed) growth is slowing

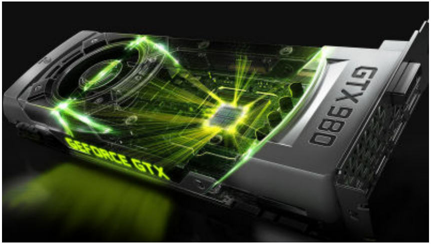


Chip makers have responded by developing multi-core processors



Source: Wikipedia

and GPUs...



Issues

- Exploiting multiple cores / threads is nontrivial
- Sometimes we need to redesign algorithms
- Sometimes we can use tools that automate exploitation of multiple cores

Programming

The assignment will require programming

Acceptable languages

- Python
- Julia

We will use a mix

- Mainly Python in class
- Julia used in the textbook

Programming Background

A common classification:

- **low** level languages (assembly, C, Fortran)
- **high** level languages (Python, Ruby, Haskell)

Low level languages give us fine grained control

Example. $1 + 1$ in assembly

```
pushq   %rbp
movq    %rsp, %rbp
movl    $1, -12(%rbp)
movl    $1, -8(%rbp)
movl    -12(%rbp), %edx
movl    -8(%rbp), %eax
addl    %edx, %eax
movl    %eax, -4(%rbp)
movl    -4(%rbp), %eax
popq    %rbp
```

High level languages give us abstraction, automation, etc.

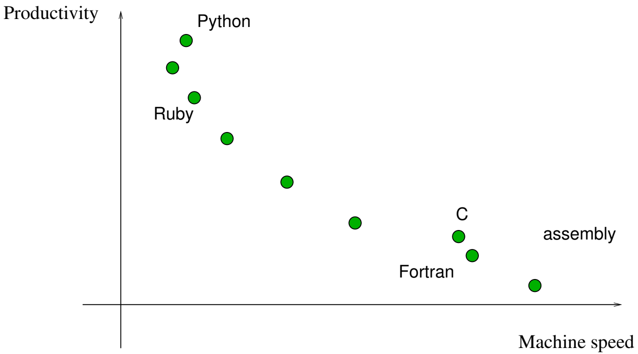
Example. Reading from a file in Python

```
data_file = open("data.txt")
for line in data_file:
    print(line.capitalize())
data_file.close()
```

Jane Street on readability:

> *There is no faster way for a trading firm to destroy itself than to deploy a piece of trading software that makes a bad decision over and over in a tight loop.*
>
> *Part of Jane Street's reaction to these technological risks was to put a very strong focus on building software that was easily understood—software that was readable.*
>
> – Yaron Minsky, Jane Street

Trade-Offs

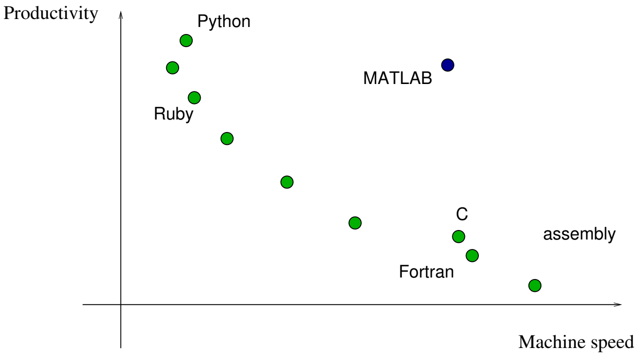

Productivity
Python
Ruby
C
assembly
Fortran
Machine speed

But what about scientific computing?

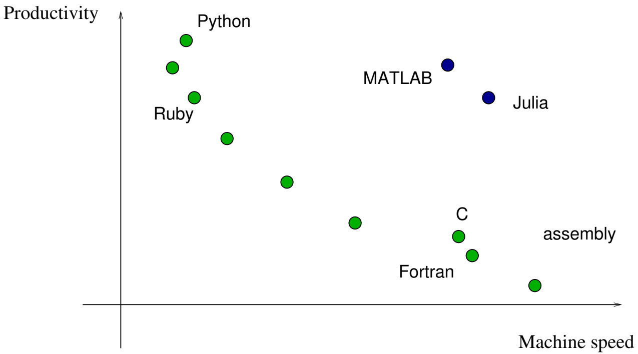
Requirements

- Productive — easy to read, write, debug, explore
- Fast computations

Trade-Offs

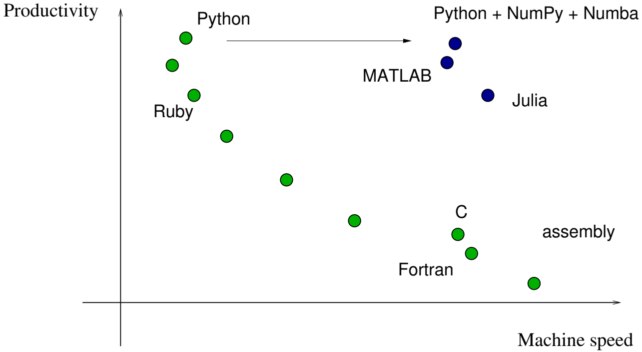

Productivity
Python
MATLAB
Ruby
C
assembly
Fortran
Machine speed

Trade-Offs



Productivity
Python
MATLAB
Julia
Ruby
C
assembly
Fortran
Machine speed

Trade-Offs


Productivity
Python
Python + NumPy + Numba
MATLAB
Julia
Ruby
C
assembly
Fortran
Machine speed

Key Takeaways

- Don't write in C / C++ / Fortran, no matter what your professor says
- JIT compilation is changing scientific computing
- Same with parallelization
- New algorithms, new techniques — and opportunities

Demo: Fast Computing with Python

Let's quickly see what a difference computing platforms make

1. Download the notebook from `https://notes.quantecon.org/submission/622ed4daf57192000f918c61`
2. Run on `colab.research.google.com`

Notes:

- You need a Google account
- Runs faster if you have Colab Pro

Need for Analysis

The demonstration showed the power of modern hardware/software

But **can faster computers always save us**?

If so, do we really need to care about clever maths/algorithms?

Below we demonstrate that

- Fast computers are not enough
- Clever algorithms and analysis are vital!

The demonstration concerns **brute force** maximization

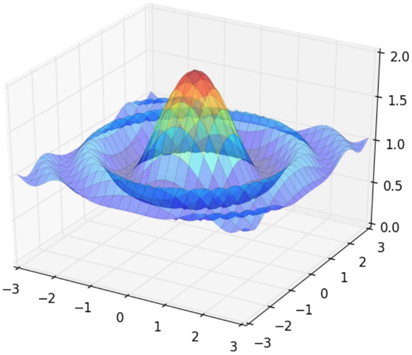


Figure: The function to maximize

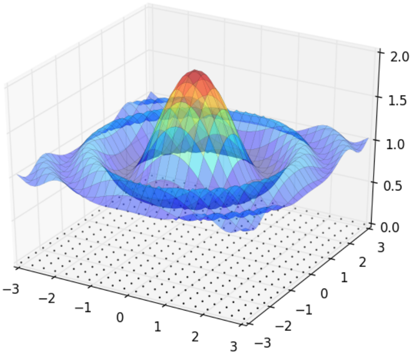


Figure: Grid of points to evaluate the function at

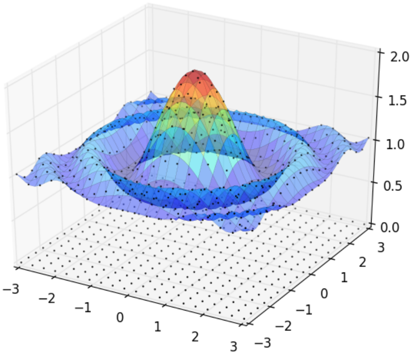


Figure: Evaluations

Grid size $= 20 \times 20 = 400$

Outcomes

- Number of function evaluations $= 400$
- Time taken $=$ almost zero
- Maximal value recorded $= 1.951$
- True maximum $= 2$

Not bad and we can easily do better

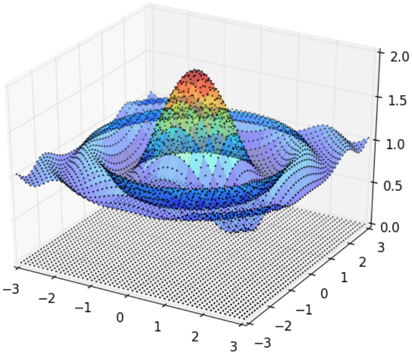


Figure: $50^2 = 2500$ evaluations

- Number of function evaluations $= 50^2$
- Time taken $= 400\ \mu$s
- Maximal value recorded $= 1.992$
- True maximum $= 2$

So why even study optimization?

The problem is mainly with larger numbers of choice variables

- 3 vars: $\max_{x_1,x_2,x_3} f(x_1, x_2, x_3)$
- 4 vars: $\max_{x_1,x_2,x_3,x_4} f(x_1, x_2, x_3, x_4)$
- $\ldots$

If we have 50 grid points per variable and

- 2 variables then evaluations $= 50^2 = 2500$
- 3 variables then evaluations $= 50^3 = 125,000$
- 4 variables then evaluations $= 50^4 = 6,250,000$
- 5 variables then evaluations $= 50^5 = 312,500,000$
- $\ldots$

Example. Recent study: Optimal placement of drinks across vending machines in Tokyo

Approximate dimensions of problem:

- Number of choices for each variable $= 2$
- Number of choice variables $= 1000$

Hence number of possibilities $= 2^{1000}$

How big is that?

```
In [10]: 2**1000
Out[10]:
1071508607186267320948425049060001810561404811705
5336074437503883703510511249361224931983788156958
5812759467291755314682518714528569231404359845775
7469857480393456777482423098542107460506237114187
7954182153046474983581941267398767559165543946077
0629145711964776865421676604298316526243868372056
68069376
```

Let's say my machine can evaluate about 1 billion possibilities per second

How long would that take?

```
In [16]: (2**1000 / 10**9) / 31556926  # In years
Out[16]:
3395478403651443492780079558636357072806789899995
899349462539661933596146571733926965255861364854
060286985707326991591901311029244639453805988092
045933072657455119924381235072941549332310199388
301571394569707026437986448403352049168514244509
939816790601568621661265174170019913588941596
```

What about high performance computing?

- more powerful hardware
- faster CPUs
- GPUs
- vector processors
- cloud computing
- massively parallel supercomputers
- ...

Let's say speed up is 10^{12} (wildly optimistic)

```
In [19]: (2**1000 / 10**(9 + 12)) / 31556926
Out[19]:
3395478403651443492780079558636357072806789899958
9934946253966193359614657173392696525586136485406
0286985707326991591901311029244639453805988092045
9330726574551199243812350729415493323101993883015
7139456970702643798644840335204916851424450993981
6790601568621661265174170019
```

For comparison:

```
In [20]: 5 * 10**9 # Expected lifespan of sun
Out[20]: 5000000000
```

Summary

Software, platforms and hardware **do** matter

- Fast machine code
- Compiler optimization tricks
- Parallelization (CPUs, GPUs)

But algorithms matter even more

- Clever ideas reduce curse of dimensionality
- Mathematical analysis is needed to find and study algorithms

Getting Started with Python

Option 1: Install locally

1. Go to anaconda.com
2. Download Anaconda Python
3. Install
4. Start **Jupyter notebook**

Option 2:

1. Get a Google account (if necessary)
2. Go to https://colab.research.google.com

Getting Started with Python

Get notebooks from

https://github.com/jstac/tokyo_2022_coursework/tree/main/lecture_1

Steps:

1. Download the notebooks to your machine
   - One by one (raw), zip file, clone
2. Go to Jupyter Notebook landing page (dashboard)
3. Click on **Upload**
4. (or in Colab, use `File -> Upload notebook`)

Homework

1. Review what we have covered!
2. Optionally, start to read Chapter 1 of Dynamic Programming