Scientific Computation

Spring 2019

Lecture 1

Imperial College
London

Instructor

Prasun Ray

Teaching Fellow

Department of Mathematics

p.ray@imperial.ac.uk

Huxley 6M20

Office hours Mondays 10-11am, location TBC

Tuesdays 5-6pm, MLC

(First office hour on Tuesday, 15/1)

Weekly schedule

- **Lectures:**
  Monday, 9-10am, Huxley 311
  Tuesday, 1-2pm, Huxley 311

- **Labs:**
  Tuesday, 4-5pm, MLC (Huxley 414)
  or
  Wednesday, 10-11am, MLC

- **Only need to attend *one* lab session**

Assessment (tentative)

3 Programming assignments
HW1: Assigned 28/1, due 6/2 (**20%**)
HW2: Assigned 18/2, due 1/3 (**40%**)
HW3: Assigned 7/3, due 21/3 (**40%**)

Submitting HW1 **commits you to the course**

Mastery material (M4/M5): Assignment will be provided last week of term and will be due 4 weeks later, counts **"20%"**

Online material

- Main resource is course webpage:

  http://m345sc.bitbucket.io/

- Copies of lecture slides and example codes will be provided

  All course material will be available on course bitbucket page (more on this later):

  https://bitbucket.org/m345sc/sc2019

Scientific computation

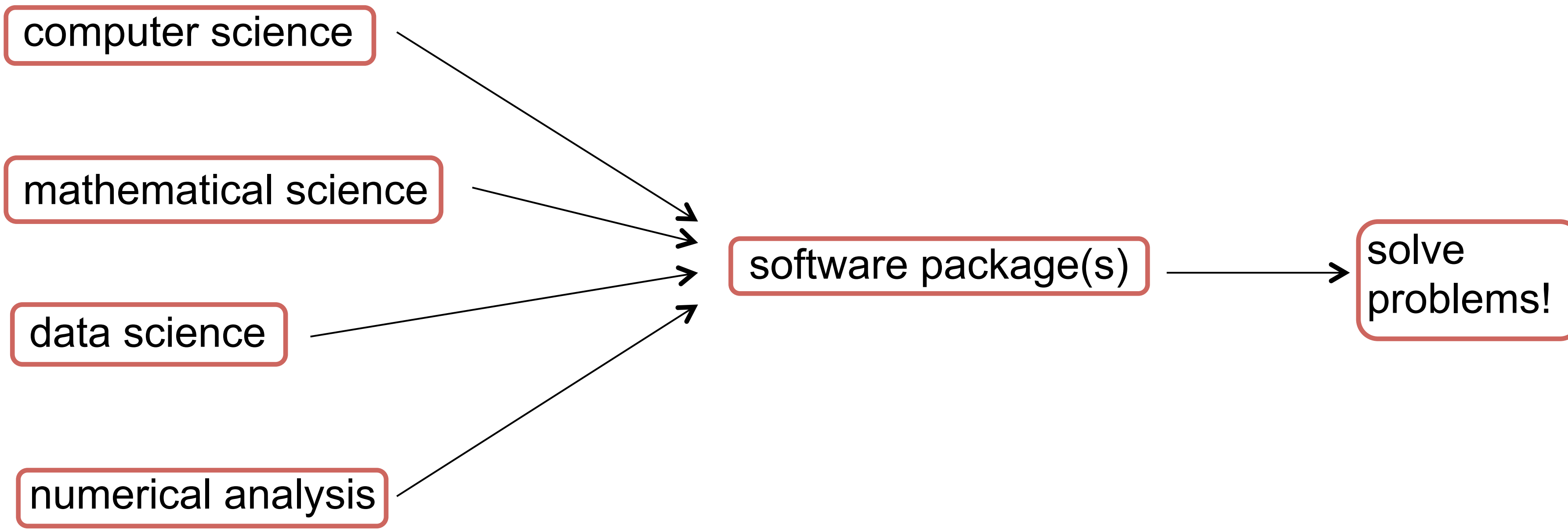
- **What is "*Scientific computation*"?**

Scientific computation

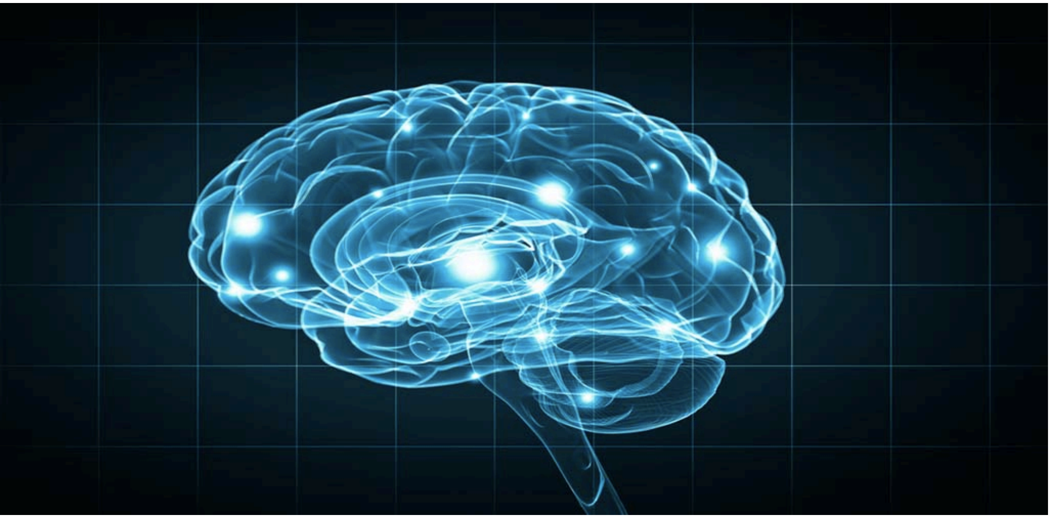
- **What is "*Scientific computation*"?**
  - **No single, standard definition (that I'm aware of!)**

Scientific computation

- **What is "*Scientific computation*"?**
  - **No single, standard definition (that I'm aware of!)**



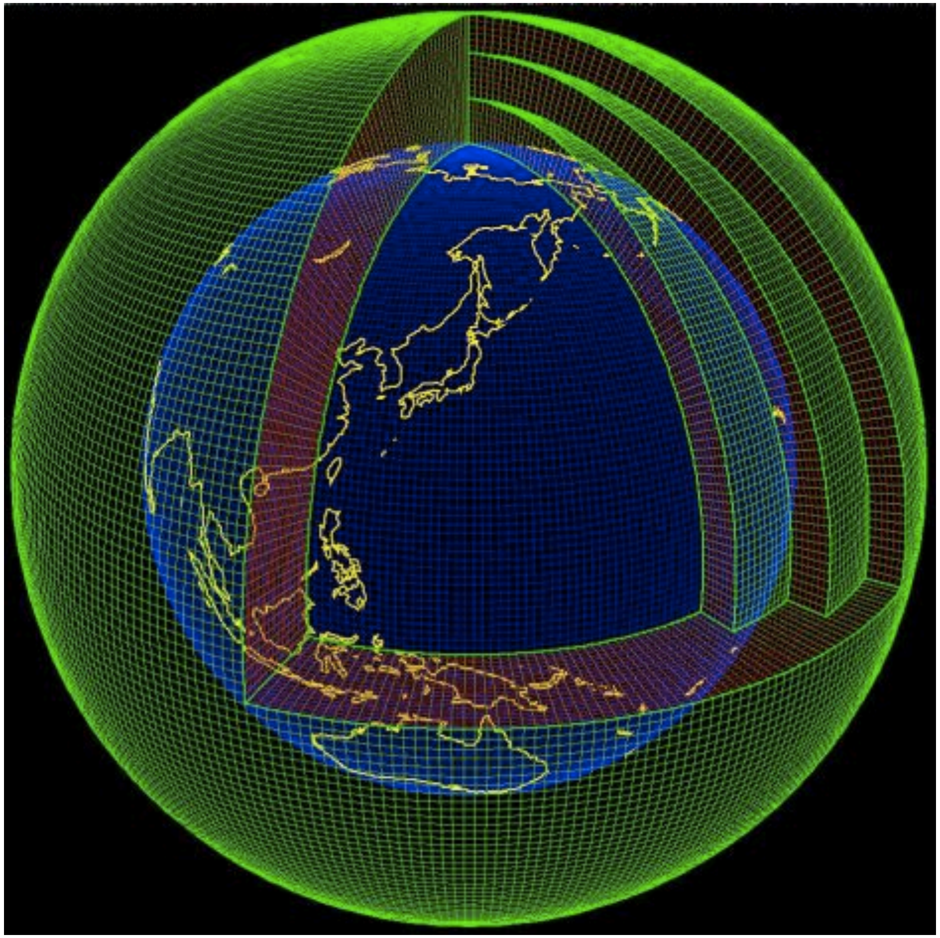
Scientific computation



D.S. Bassett, How You Think: Structural Network Mechanisms of Human Brain Function

- **Example 1: Brain dynamics**
- **Graph partitioning**
- **Biochemistry**
- **Nonlinear “differential” equations**

Scientific computation



https://www.jma.go.jp/jma/jma-eng/jma-center/nwp

- **Example 2: Numerical weather prediction**
- **Partial differential equations**
- **Data assimilation**
- **Large sparse linear systems**

Scientific computation

- **Example 3: Self-driving car**
- **Rapid image processing (images are matrices)**
- **Numerical linear algebra**
- **Machine learning**
- **Mathematical optimization**

Tentative syllabus

Lectures 1-4: Searching and sorting (plus a little DNA)

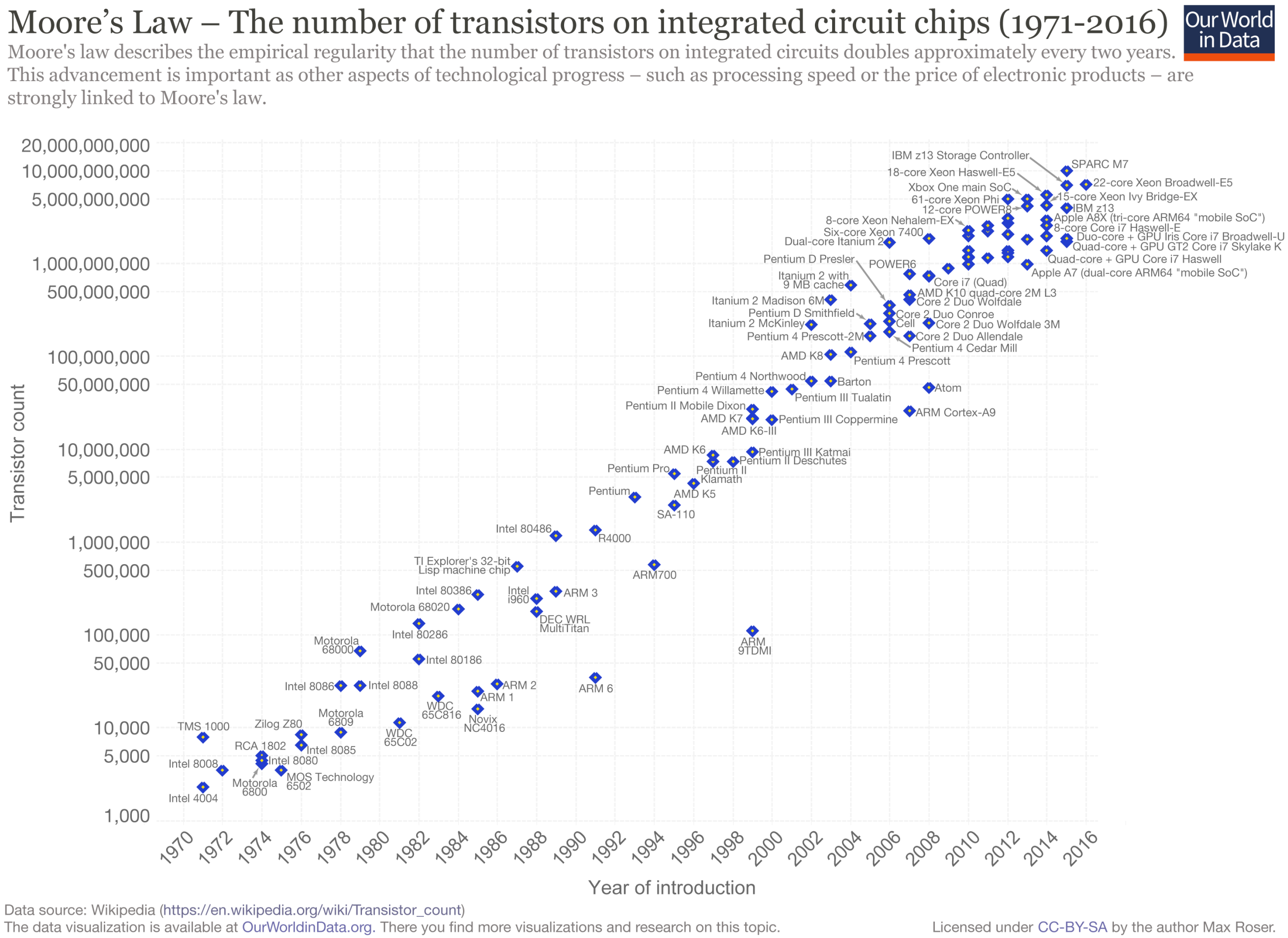
Lectures 5-8: Complex networks, graph search

Lectures 9-12: Data analysis and optimization

Lectures 13-16: Numerical solution of differential equations

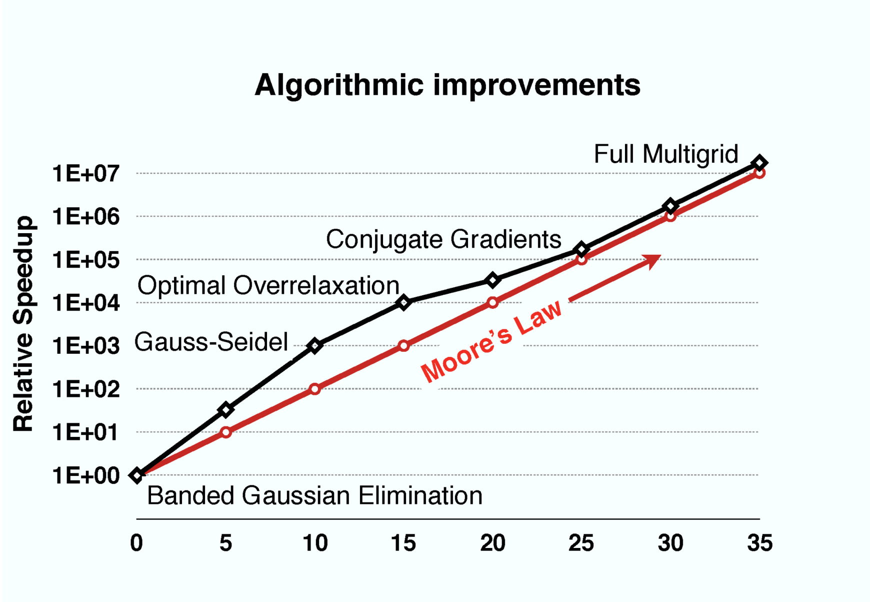
Lecture 17-20: Possible topics: data analysis with Pandas, compiling Python with numba, parallel computing with Python

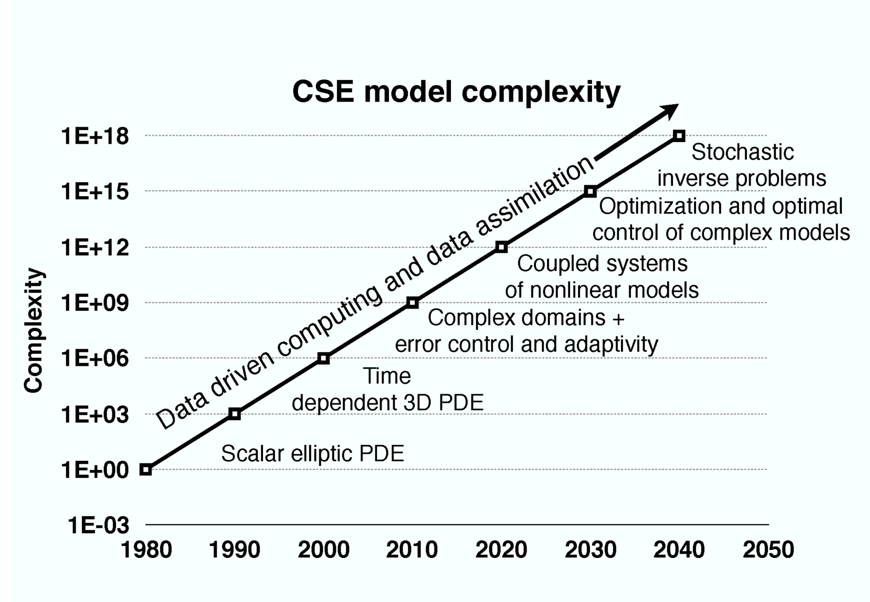
Moore’s law



- **Number of transistors on chip doubles every 2 years**
- **Enormous amount of computational power at our fingertips**

Algorithms and hardware





- **Algorithmic efficiency (in some cases) has improved in parallel with computational power**
- **With this improvement in efficiency and power has come an increase in the complexity of problems being investigated**

SIAM Rev (2001)

Python

- This course could be called *Scientific computation with Python*
- Is Python a “good” language for scientific computing?

Python

- This course could be called *Scientific computation with Python*

- Is Python a "good" language for scientific computing?

- Advantages:
  - Easy to learn, free
  - Many powerful tools for scientific computing are available (e.g. numpy, scipy, scikit-learn)
  - Also a general-purpose language – well-designed for general software development

Python

- This course could be called *Scientific computation with Python*

- Is Python a “good” language for scientific computing?

- Advantages:
  - Easy to learn, free
  - Many powerful tools for scientific computing are available (e.g. numpy, scipy, scikit-learn)
  - Also a general-purpose language – well-designed for general software development

- Disadvantages:
  - Not specifically designed for scientific computing (cf. Matlab)
  - Python is an *interpreted* language – will be slower than compiled languages (c, Fortran) in many cases (e.g. loops)

It’s a good place to start!

Python

- This class assumes either:
    - Familiarity with M1C-level python:
        - Basic datatypes, loops, functions, numpy arrays, plotting with matplotlib
    - *Or* good programming experience and a willingness to independently learn the basics of Python during the first ~2 weeks of the class

Python

- **This class assumes either:**
  - **Familiarity with M1C-level python:**
    - **Basic datatypes, loops, functions, numpy arrays, plotting with matplotlib**
  - ***Or* good programming experience and a willingness to independently learn the basics of Python during the first ~2 weeks of the class**

- **The course webpage has introductory videos, exercises and references:**
  - https://m345sc.bitbucket.io/python.html
  - https://m345sc.bitbucket.io/reading.html
  - **M1C notes are also a good reference**

- **Look through the exercises this week!**

Python

- **The class uses Python 3**

- **I will be using Python3.6 – this is also installed on MLC machines, there is an "anaconda navigator" link on the desktop**

- **You are encouraged to work on your own computers**
  - **This is not required! Everything you need is on the MLC computers**
  - **Straightforward to get needed packages via** anaconda **or** canopy
    - **E.g.** https://www.anaconda.com/download
    - **These packages also provide their own code editors (e.g. Spyder w/ anaconda). I will use Atom, but you can use whatever you like**

- **If using MLC machines, you should save your work within your home directory on the network (h://) drive**

Searching

- **The two foundational problems in computer science are searching and sorting**

- **Problem statement:** ***Given a sorted list, find a location of a specified item within the list. Return "not found" if item is not contained within list***

- **Examples:**
    - **Find name in directory**
    - **Find book (by call number) in library**
    - **Find id number in database**
    - **Find "31" in the array below**

2	6	8	23	24	31	32	53	56

Searching

- **Linear (naive) search: Step through list one element at a time, terminate search if/when item is found**

```
def linsearch(L,x):
    """find location of x in L, if x is not in L, return -1
    """

    for i,y in enumerate(L):
        if y==x: return i

    return -1
```

- **What is the cost?**

  - **On average,** N/2
    - N = len(L)

  - **Can we do better?**

Searching

- **To do better, we should take advantage of the list being sorted**

- **Linear search discards one element at a time**
  - **Can we discard more?**

- **We can discard half if we first compare with the median element**
  - **If** x < L_median**: no need to search in “right” half of list**
  - **or if** x > L_median**: no need to search in “left” half**
  - **Can then run linear search on appropriate half**

Searching

- **To do better, we should take advantage of the list being sorted**

- **Linear search discards one element at a time**
  - **Can we discard more?**

- **We can discard half if we first compare with the median element**
  - **If** x < L_median**: no need to search in "right" half of list**
  - **or if** x > L_median**: no need to search in "left" half**
  - **Can then run linear search on appropriate half**

- **But why run linear search? Can again discard half (of what's left)**
  - **This is the idea underlying *binary search***

Binary search

Task: find 31 **in list below**

2	6	8	23	24	31	32	53	56

31 > 24: **discard left 'half':**

31	32	53	56

31 < 53: **discard right half**

31	32

31 < 32: **discard right half**

Imperial College
London

Binary search

Python implementation: keep track of start **and** end **indices of list as it is truncated**

```
#Set initial start and end indices for full list
   istart = 0
   iend = len(L)-1

   #Contract "active" portion of list
   while istart<=iend:

       imid = int(0.5*(istart+iend))

       if x==L[imid]:
           return imid
       elif x < L[imid]:
           iend = imid-1
       else:
           istart = imid+1

   return -1
```

Binary search

- **Correctness: basic idea – if target is within original array, it will be found in sub-array after contraction (convince yourself that a proof of correctness follows)**

- **Speed: Is binary search faster than linear search?**

Binary search

Speed: Is binary search faster than linear search?

Consider modified form of binary search where exactly half of array is discarded each iteration

- **If we start with** N=16**, then in the worst case, there are iterations for arrays with size** 16,8,4,2,1
- **Each iteration requires 5-6 operations (why?)**
- **Worst case:** $6(\log_2 N+1) + 3$ **operations**
- **Generally, we are interested in cases where** N **is large and say that the cost is** $O(\log_2 N)$