# Numerical Analysis / Scientific Computing

CS450 / MATH 450 / ECE 491 / CSE 401

Andreas Kloeckner

Fall 2023

# Outline

# What's the point of this class?

'*Scientific Computing*' describes a family of approaches to obtain approximate solutions to problems *once they've been stated mathematically*.

Name some applications:

$\partial_x$        Fluids

                    optimization $\in$

                        trajectories

                            ML

            simulation            solve nonlinear PDEs

## What's the point of this class?

'*Scientific Computing*' describes a family of approaches to obtain approximate solutions to problems *once they've been stated mathematically*.
Name some applications:

> ▶ Engineering simulation
>   > ▶ E.g. Drag from flow over airplane wings, behavior of photonic devices, radar scattering, . . .
>   > ▶ → Differential equations (ordinary and partial)
> ▶ Machine learning
>   > ▶ Statistical models, with unknown parameters
>   > ▶ → Optimization
> ▶ Image and Audio processing
>   > ▶ Enlargement/Filtering
>   > ▶ → Interpolation
> ▶ Lots more.

## What do we study, and how?

Problems with real numbers (i.e. *continuous* problems)

$$\pi$$

What's the general approach?
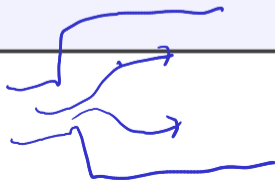
repr
$$\exists \quad uni$$

# What do we study, and how?

$\pm x . 7^{y}$

Problems with real numbers (i.e. *continuous* problems)

> ▶ As opposed to *discrete* problems.
> ▶ Including: How can we put a real number into a computer? (and with what restrictions?)

What's the general approach?

> ▶ Pick a *representation* (e.g.: a polynomial)
> ▶ Existence/uniqueness?

$u(x_1) \to \mathbb{R}^2$
$u: \mathbb{R}^2 \to \mathbb{R}^2$

# What makes for *good* numerics?

How good of an answer can we expect to our problem? $\pi$

How *fast* can we expect the computation to complete?

# What makes for *good* numerics?

↳ repr / model
↳ alg
↳ impl.

How good of an answer can we expect to our problem?

> ▶ Can't even represent numbers exactly. ←
> ▶ Answers will always be *approximate*. ←
> ▶ So, it's natural to ask *how far off the mark* we really are.

*How fast* can we expect the computation to complete?

> ▶ A.k.a. what algorithms do we use? ←
> ▶ What is the cost of those algorithms?
> ▶ Are they efficient?
>    (I.e. do they make good use of available machine time?)

# Implementation concerns

How do numerical methods *get implemented*?

# Implementation concerns

How do numerical methods *get implemented*?

*numpy*

- ▶ Like anything in computing: A layer cake of *abstractions* ("careful lies")
- ▶ What tools/languages are available?
- ▶ Are the methods easy to implement? ⬅
- ▶ If not, how do we make use of existing tools? ⬅
- ▶ How robust is our implementation? (e.g. for error cases)

# Class web page

https://bit.ly/cs450-f23

- ▶ Assignments
  - ▶ HW1 (soon!)
  - ▶ Pre-lecture quizzes
  - ▶ In-lecture interactive content (bring computer or phone if possible)
- ▶ Textbook
- ▶ Exams
- ▶ Class outline (with links to notes/demos/activities/quizzes)
- ▶ Discussion forum
- ▶ Policies
- ▶ Video

# Programming Language: Python/numpy

- ▶ Reasonably readable
- ▶ Reasonably beginner-friendly
- ▶ Mainstream (top 5 in 'TIOBE Index')
- ▶ Free, open-source
- ▶ Great tools and libraries (not just) for scientific computing
- ▶ Python 2/3? 3!
- ▶ `numpy`: Provides an array datatype
  Will use this and `matplotlib` all the time.
- ▶ See class web page for learning materials

**Demo:** Sum the squares of the integers from 0 to 100. First without numpy, then with numpy.

# Supplementary Material

- Numpy (from the SciPy Lectures)
- 100 Numpy Exercises
- Dive into Python3

# Sources for these Notes

- M.T. Heath, Scientific Computing: An Introductory Survey, Revised Second Edition. Society for Industrial and Applied Mathematics, Philadelphia, PA. 2018.
- CS 450 Notes by Edgar Solomonik
- Various bits of prior material by Luke Olson

# Open Source <3

These notes (and the accompanying demos) are open-source!

Bug reports and pull requests welcome:
https://github.com/inducer/numerics-notes

Copyright (C) 2020 Andreas Kloeckner

# What problems *can* we study in the first place?

To be able to compute a solution (through a process that introduces errors), the problem...

$$\exists !$$

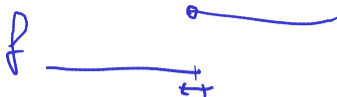If it satisfies these criteria, the problem is called *well-posed*. Otherwise, *ill-posed*.

# What problems *can* we study in the first place?

To be able to compute a solution (through a process that introduces errors), the problem...

> ▶ Needs to *have* a solution
> ▶ That solution should be *unique*
> ▶ And *depend continuously* on the inputs

If it satisfies these criteria, the problem is called *well-posed*. Otherwise, *ill-posed*.

## Dependency on Inputs



We excluded discontinuous problems–because we don't stand much chance
for those.

...what if the problem's input dependency is just *close to discontinuous*?

sensitive

sensitivity of problem: conditioning
sensitivity of comp , stability

## Dependency on Inputs

We excluded discontinuous problems–because we don't stand much chance for those.
. . . what if the problem's input dependency is just *close to discontinuous*?

> ▶ We call those problems *sensitive* to their input data.
>   Such problems are obviously trickier to deal with than
>   non-sensitive ones.
>
> ▶ Ideally, the computational method will not *amplify* the
>   sensitivity