

Numerical Analysis / Scientific Computing  
CS450 MATH450, CSE401, ECE491

Andreas Kloeckner

Spring 2026

# Outline

## Introduction to Scientific Computing

Notes

Notes (unfilled, with empty boxes)

Notes (source code on Github)

About the Class

Errors, Conditioning, Accuracy, Stability

Floating Point

Systems of Linear Equations

Linear Least Squares

Eigenvalue Problems

Nonlinear Equations

Optimization

Interpolation

Numerical Integration and Differentiation

Initial Value Problems for ODEs

Boundary Value Problems for ODEs

Partial Differential Equations and Sparse Linear Algebra

Fast Fourier Transform

Additional Topics

Wrap-up

## 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:

# 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)

What's the general approach?

# What do we study, and how?

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?

## What makes for *good* numerics?

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

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

# What makes for *good* numerics?

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

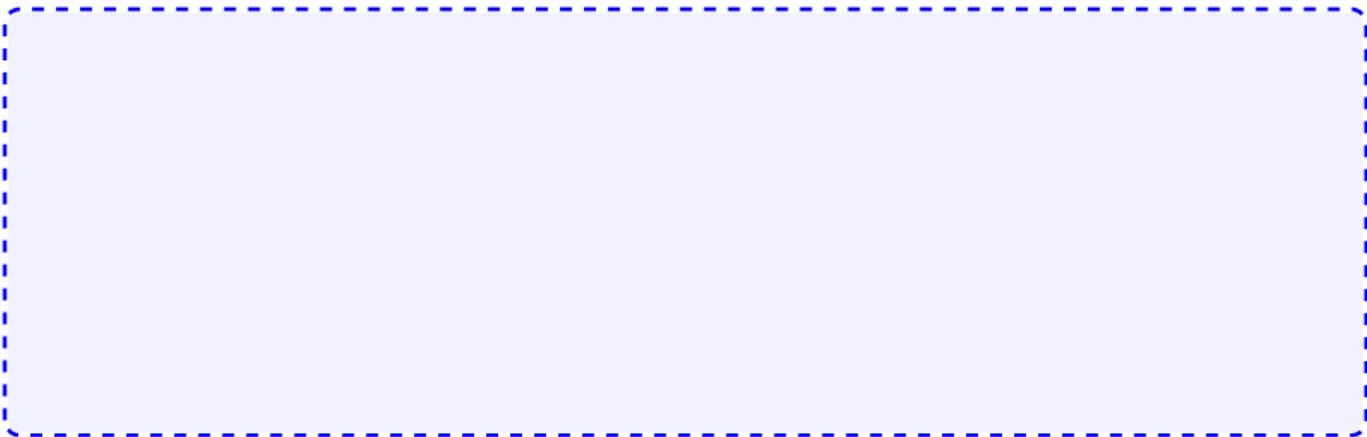
- ▶ Can't even represent numbers exactly.
- ▶ Answers will always be *approximate*
- ▶ So, it's natural to ask *how far off the mark* we really are.

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*?

- ▶ 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-s26>

- ▶ 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

range (0, 100)  
↓  
[0, 100)

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

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

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

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

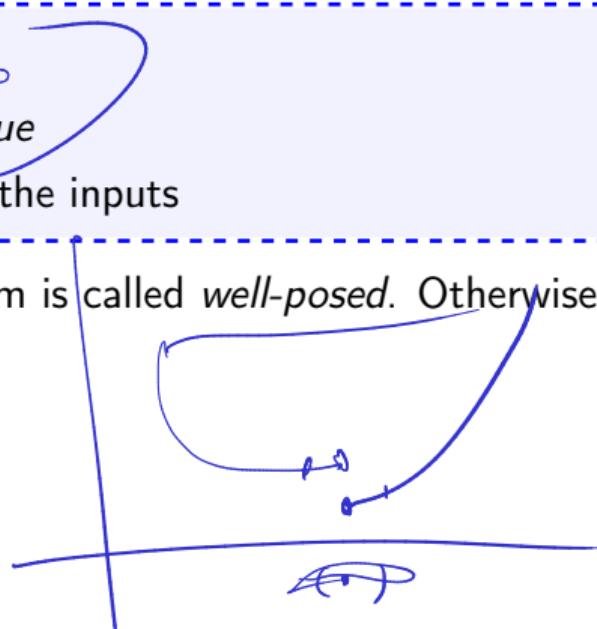
If it satisfies these criteria, the problem is called *well-posed*. Otherwise, *ill-posed*.

## 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

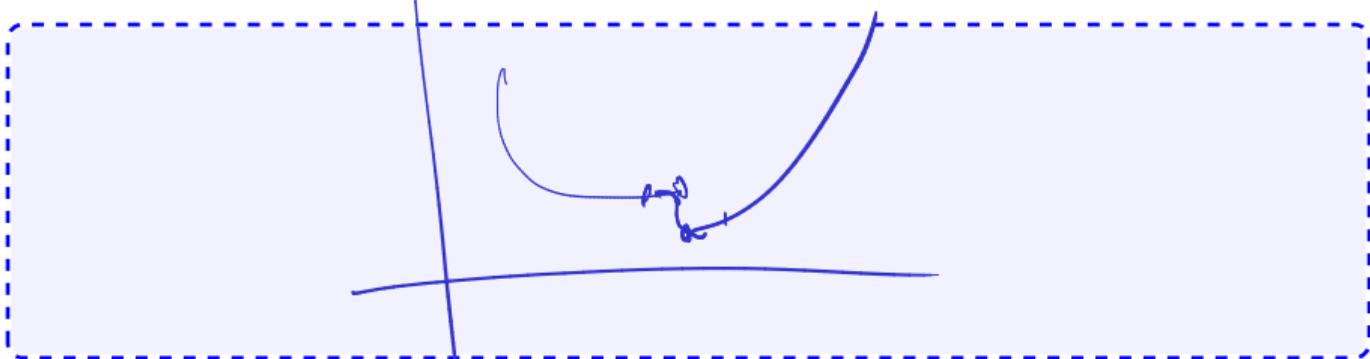
If it satisfies these criteria, the problem is called *well-posed*. Otherwise, *ill-posed*.



## 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*?



## 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

# Approximation

*When* does approximation happen?

Demo: Truncation vs Rounding [cleared]

# Approximation

*When* does approximation happen?

- ▶ Before computation
  - ▶ modeling
  - ▶ measurements of input data
  - ▶ computation of input data
- ▶ During computation
  - ▶ truncation / discretization
  - ▶ rounding

Demo: Truncation vs Rounding [cleared]

## Example: Surface Area of the Earth

Compute the surface area of the earth.

What parts of your computation are approximate?

$$A = 4\pi r^2$$

## Example: Surface Area of the Earth

Compute the surface area of the earth.

What parts of your computation are approximate?

*All of them.*

$$A = 4\pi r^2$$

- ▶ Earth isn't really a sphere
- ▶ What does radius mean if the earth isn't a sphere?
- ▶ How do you compute with  $\pi$ ? (By rounding/truncating.)

# Measuring Error

How do we measure error?

Idea: Consider all error as being *added onto* the result.

Absolute error = approx value - true value

Relative error =  $\frac{\text{Absolute error}}{\text{true value}}$

→ Goal: Estimate these

## Recap: Norms

vector  $\mathbb{R}^n$

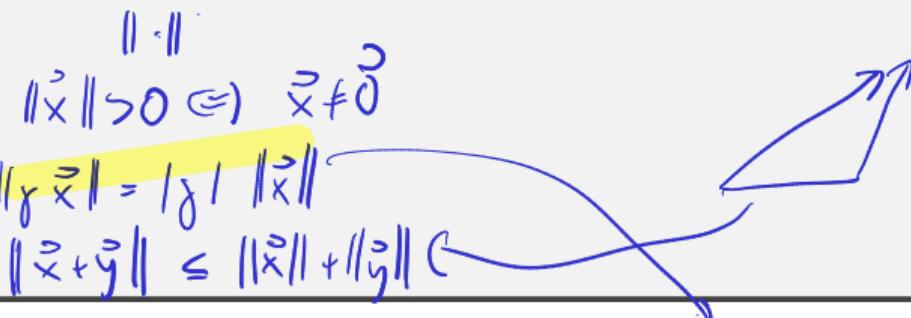
What's a norm?

$$f: \mathbb{R}^n \rightarrow \mathbb{R}_0^+$$

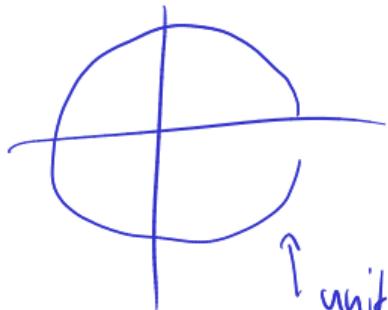
Define *norm*.

$$\begin{aligned} \|\cdot\| \\ \|\vec{x}\| > 0 \Leftrightarrow \vec{x} \neq \vec{0} \\ \forall \gamma \in \mathbb{R} \quad \|\gamma \vec{x}\| = |\gamma| \|\vec{x}\| \\ \|\vec{x} + \vec{y}\| \leq \|\vec{x}\| + \|\vec{y}\| \end{aligned}$$

unit ball



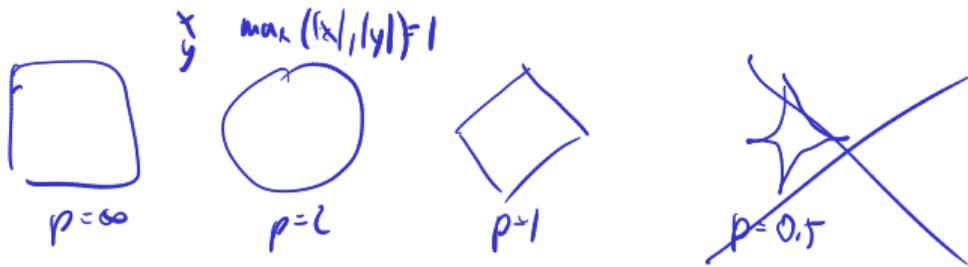
$$\left\| \begin{pmatrix} x \\ y \end{pmatrix} \right\|_2 = \sqrt{x^2 + y^2}$$



unit ball for the 2-norm

## Norms: Examples

Examples of norms?



$$\left\| \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \right\|_p = \sqrt[p]{\sum_{i=1}^n |x_i|^p}$$



$p=\infty$  allowed

Demo: Vector Norms [cleared]

$\rightarrow$  max of abs values