Numerical Analysis / Scientific Computing

CS450 MATH 450, CSE 401, ECE 491

Andreas Kloeckner

Spring 2026

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:

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, ...
  - $\rightarrow$ Differential equations (ordinary and partial)
- Machine learning
  - Statistical models, with unknown parameters
  - $\rightarrow$ Optimization
- Image and Audio processing
  - Enlargement/Filtering
  - $\rightarrow$ 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

range(0, 100) ↓ [0, 100)

- 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

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

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

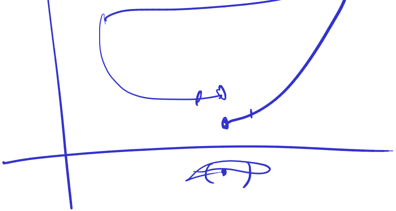
If it satisfies these criteria, the problem is called *well-posed*. Otherwise, *ill-posed*.

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

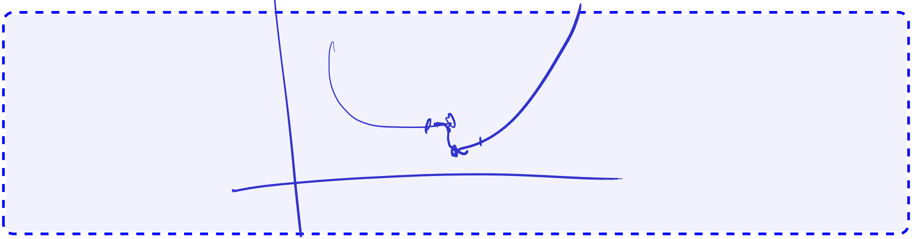
If it satisfies these criteria, the problem is called *well-posed*. Otherwise, *ill-posed*.



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 π? (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?

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

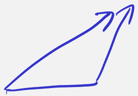
Define *norm*.

$\|\cdot\|$

$\|\vec{x}\| > 0 \iff \vec{x} \neq \vec{0}$

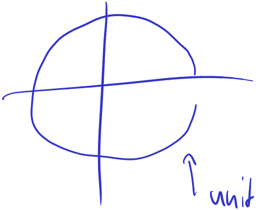
$\gamma \in \mathbb{R}$ $\|\gamma \vec{x}\| = |\gamma| \, \|\vec{x}\|$

$\|\vec{x}+\vec{y}\| \leq \|\vec{x}\| + \|\vec{y}\|$



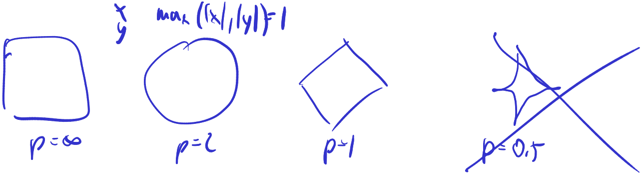
unit balls

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



↑ unit ball for the 2-norm

Norms: Examples



Examples of norms?

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

$p \geq 1$

$p = \infty$ allowed

→ max of abs values

Demo: Vector Norms [cleared]