

Numerical Analysis / Scientific Computing  
CS450

Andreas Kloeckner, Xiaoyu Wei

Fall 2021

Scientific Computing Area

# Outline

Introduction to Scientific Computing

Notes

Notes (unfilled, with empty boxes)

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

## 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:

- Stock market → predictions      'Simulations'
- {
  - engineering models
  - differential equations (ODE/PDE)
- Marketing/Adv. → ML models
- Protein engineering →
- signal processing

# What do we study, and how?

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

- As opposed to discrete
- Putting real numbers in a computer

What's the general approach?

- Pick a representation
- Existence/uniqueness

# What makes for good numerics?

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

- Real numbers don't exist in computers
- Answers be approximate
- Put upper bound on the error

How fast can we expect the computation to complete?

- What algorithms do we use?
- What's the cost?
- How do we model the cost?
- Are they efficient? (on a machine)

$$\begin{matrix} n & & n \\ \boxed{A} & \cdot & \boxed{x} \\ n & & \end{matrix} = Ax$$

mult:  $n^2 \rightarrow 5n^2$   
 $\hookrightarrow O(n^2)$

$$A = a b^T = a (b^T x) = (a b^T) x$$

## Implementation concerns

How do numerical methods *get implemented*?

- interfaces / abstractions (BLAS / numpy)
- What is available?
- Is a method easy to implement?
- How do we build it on existing machinery?
- Does the impl. that influence the error?

# Class web page

<https://bit.ly/cs450-f21>

- ▶ Assignments
  - ▶ HW1!
  - ▶ Pre-lecture quizzes
  - ▶ In-lecture interactive content (bring computer or phone if possible)
- ▶ Textbook
- ▶ Exams
- ▶ Class outline (with links to notes/demos/activities/quizzes)
- ▶ Virtual Machine Image
- ▶ Piazza
- ▶ Policies
- ▶ Video
- ▶ Inclusivity Statement

## 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

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...

- Have a soln
- Unique soln
- Output depends cont. on 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*?

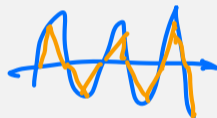
Large error.



# Approximation

When does approximation happen?

- rounding error
- over/under flow.
- truncation error



Demo: Truncation vs Rounding

## Example: Surface Area of the Earth

Compute the surface area of the earth.

What parts of your computation are approximate?

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

# Measuring Error

How do we measure error?

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

$$\text{Abs. error} = \text{approx value} - \underline{\text{true value}}$$

$$\text{rel. error} = \frac{\text{observer}}{|\text{true value}|}$$

Estimate

→ proving upper bounds

"a priori"

"a posteriori"



## Recap: Norms

What's a norm?

$f: \mathbb{R}^n \rightarrow \mathbb{R}_0^+$  returns a "magnitude" of the vector  $\vec{x}$

Define *norm*.