Applied Mathematics 205

Advanced Scientific Computing: Numerical Methods

Lecturer: Chris H. Rycroft

Logistics

Lectures: Tuesday/Thursday, 10 AM–11:30 AM
60 Oxford Street, Room 330

Email: `chr@seas.harvard.edu`

Course website (available by Thursday):
`http://iacs-courses.seas.harvard.edu/course/am205`

We will use Piazza for questions and discussion
`http://www.piazza.com`
(see AM205 website for link to Piazza page)

Logistics

My office: Pierce Hall, Room 305

Office hours: Thursday, 1:30 PM–3 PM (starting this week)

TFs: Kevin Chen (yufengchen@seas.harvard.edu)
Dustin Tran (dtran@g.harvard.edu)

(Office hours will be coordinated soon)

Prerequisites

- Calculus
- Linear algebra
- Course will touch on PDEs, but no detailed knowledge required (*i.e.* you don't need to have taken a PDE class)
- Some programming experience

Programming languages

Python will be used for the in-class demonstrations. Why Python?

- Freely available, widely used, and versatile
- Interpreted language, good for small tasks without the need for compilation
- Good linear algebra support via NumPy and SciPy extensions[1]
- Good visualization support via Matplotlib[2]

[1] http://www.numpy.org and http://www.scipy.org/

[2] http://www.matplotlib.org

Programming languages

There are many other languages that are widely used for scientific computing:

Interpreted languages: MATLAB, Julia, Perl, GNU Octave

Compiled languages: Fortran, C/C++

You can complete the assignments in any language of your choice, as long as it is easy for the teaching staff to run your code—for languages not listed here, please check with the teaching staff.

The teaching staff are familiar with different languages:

Chris: C/C++, Perl, Python, Fortran
Kevin: MATLAB, Python
Dustin: Python, R, MATLAB, (Julia)

Programming languages: assigment 0

Assignment 0 will be posted on the course website by tomorrow.

Assignment 0 provides some problems to indicate the expected level of programming familiarity for the outset of the course.

Asssignment 0 is not assessed, but it should either:

- confirm that you are already sufficiently familiar with Python (or MATLAB, C++, *etc.*)
- indicate that you need to get some programming assistance

Also, contact me or TFs regarding programming questions (Piazza is useful for these types of questions).

Syllabus (part 1)

0. **Overview of Scientific Computing**
1. **Data Fitting**
   - 1.1 Polynomial interpolation
   - 1.2 Linear least squares fitting
   - 1.3 Nonlinear least squares
2. **Numerical Linear Algebra**
   - 2.1 LU and Cholesky factorizations
   - 2.2 QR factorization, singular value decomposition
3. **Numerical Calculus and Differential Equations**
   - 3.1 Numerical differentiation, numerical integration
   - 3.2 ODEs, forward/backward Euler, Runge–Kutta schemes
   - 3.3 Lax equivalence theorem, stability regions for ODE solvers
   - 3.4 Boundary value problems, PDEs, finite difference method

Syllabus

4. **Nonlinear Equations and Optimization**
   - 4.1 Root finding, univariate and multivariate cases
   - 4.2 Necessary conditions for optimality
   - 4.3 Survey of optimization algorithms
5. **Eigenvalue problems**
   - 5.1 QR algorithm
   - 5.2 Power method, inverse iteration
   - 5.3 Lanczos algorithm, Arnoldi algorithm

(Similar to previous years by David Knezevic and Ethimios Kaxiras. Small adjustments based on feedback: reduce numerical linear algebra section, increase optimization section.)

Assessment

- 60% – Five homework assignments with equal weighting
- 10% – One take-home midterm exam
- 30% – Final project

Assessment: homework

The focus of the homework assignments will be on the mathematical theory, but will involve significant programming.

Homework will be due on Fridays – submit a written report and source code via the dropbox on the course iSite (linked from main website).

Late homework will only be accepted if there are extenuating circumstances that will be evaluated on a case-by-case basis.

Assignments will be written in LaTeX, which is an excellent platform for writing scientific documents and equations. I encourage you to try and use LaTeX for your assignments.

Assessment: code for homework

Code should be written clearly and commented thoroughly.

In-class examples will try to adhere to this standard.

The TFs should be able to easily run your code and reproduce your figures.

Homework assignments: collaboration policy

Discussion and the exchange of ideas are essential to doing academic work. For assignments in this course, you are encouraged to consult with your classmates as you work on problem sets. However, after discussion with peers, make sure that you can work through the problem sets yourself and ensure that any answers you submit for evaluation are written in your own words.

In addition, you must cite any books, articles, websites, lectures, etc that have helped you with your work using appropriate citation practices. Similarly, you must list the names of students with whom you have collaborated on problem sets.

Assessment: midterm exam

- Worth 10% of overall grade
- Scheduled $3/4$ of the way through the semester (exact date TBD)
- Take-home exam with 24 hours to complete
- No discussion or collaboration permitted

Assessment: final project

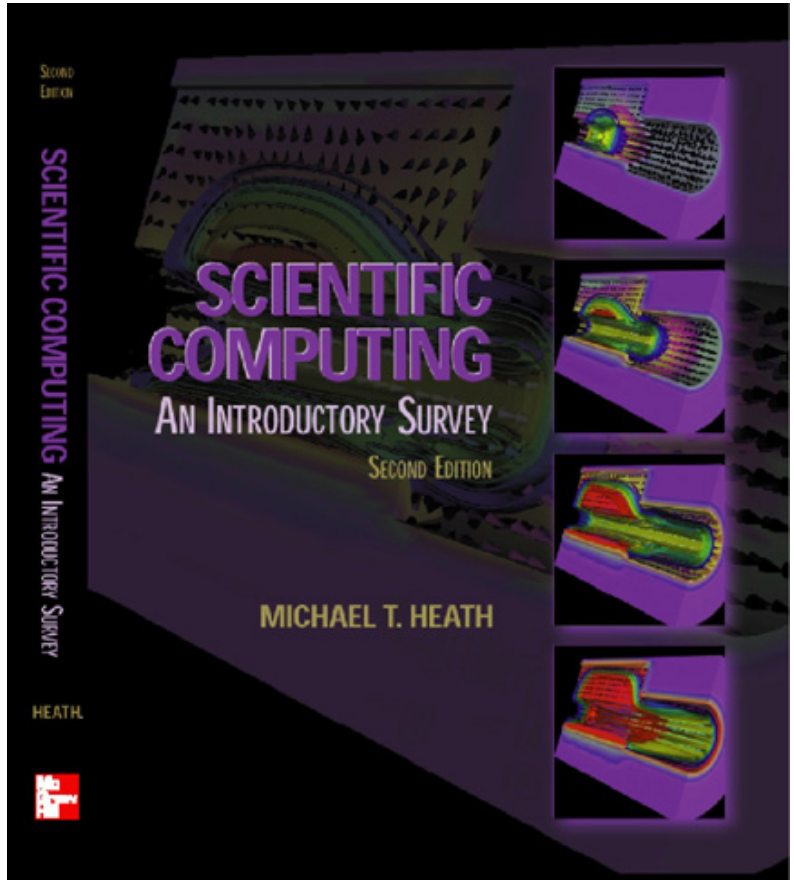
The goal of this course is to get you to be a responsible, productive user of numerical algorithms for real-world applications

The best way to demonstrate this is in your final project, worth 30%, to be completed in a group of two or three students

- Use concepts/methods related to the course to solve a problem of interest to your group
- Teaching staff will be available to discuss your choice of topic and your approach
- Project due at end of semester (exact date TBD)
- Submit a report and associated code

Textbooks

Most relevant textbook is *Scientific Computing: An Introductory Survey* by Michael T. Heath



Textbooks

- A. Greenbaum and T. P. Chartier. *Numerical Methods: Design, Analysis and Computer Implementation of Algorithms.* Princeton University Press, 2012.
- C. Moler. Numerical Computing with MATLAB. SIAM, 2004.
- L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, 1997.
- W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, 2007.
- L. R. Scott. *Numerical Analysis.* Princeton University Press, 2011.
- E. Suli, D. F. Mayers. *An Introduction to Numerical Analysis.* Cambridge University Press, 2003.
- J. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

Applied Mathematics 205

Unit 0: Overview of Scientific Computing

Lecturer: Chris H. Rycroft

Scientific Computing

Computation is now recognized as the "third pillar" of science (along with theory and experiment)

Why?

- Computation allows us to explore theoretical/mathematical models when those models can't be solved analytically
- This is usually the case for real-world problems
- *e.g.* Navier–Stokes equation model fluid flow, but exact solutions only exist in a few simple cases
- Advances in algorithms and hardware over the past 50 years have steadily increase the prominence of scientific computing

What is Scientific Computing?

Scientific computing (SC) is closely related to numerical analysis (NA)

"Numerical analysis is the study of algorithms for the problems of continuous mathematics"
Nick Trefethen, SIAM News, 1992.

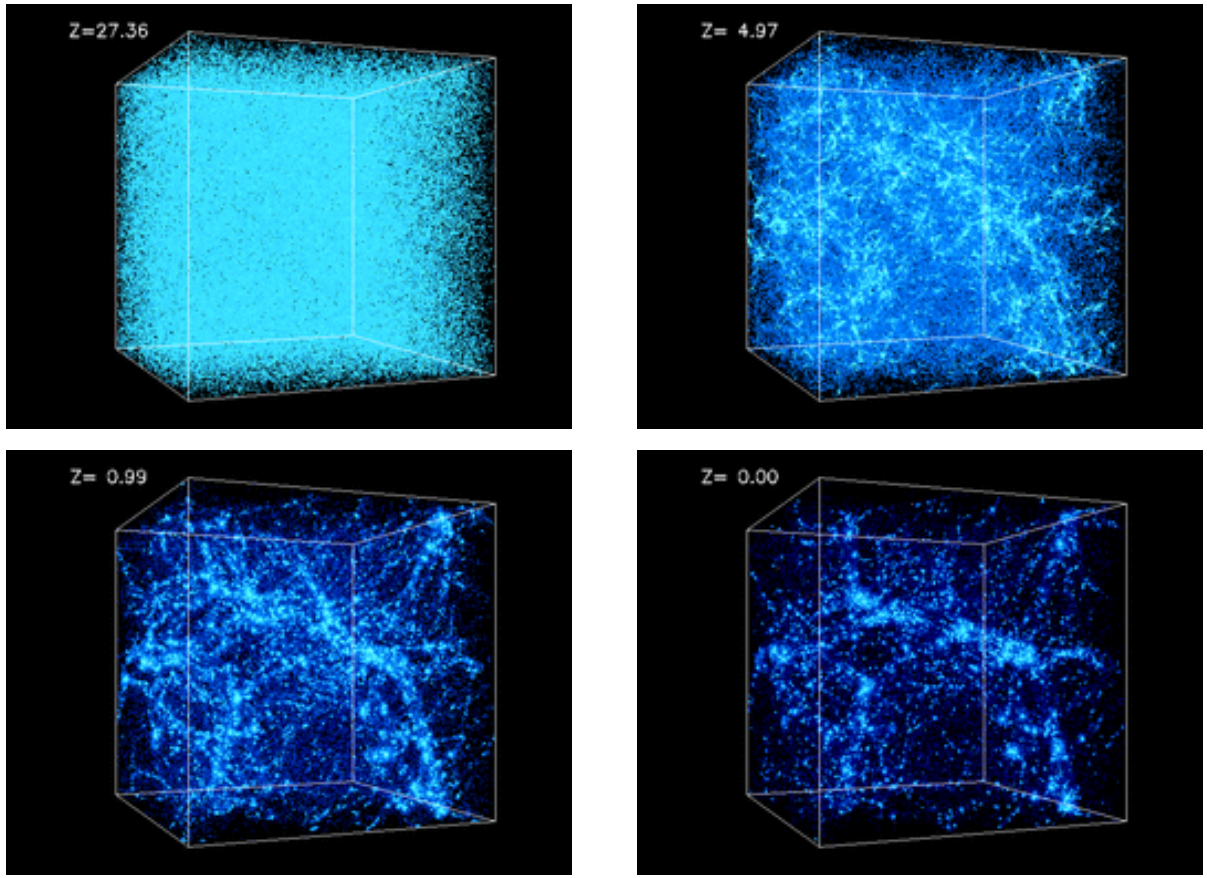
NA is the study of these algoritms, while SC emphasizes their application to practical problems

Continuous mathematics: algorithms involving real (or complex) numbers, as opposed to integers

NA/SC are quite distinct from Computer Science, which usually focuss on discrete mathematics (*e.g.* graph theory or cryptography)

Scientific Computing: Cosmology

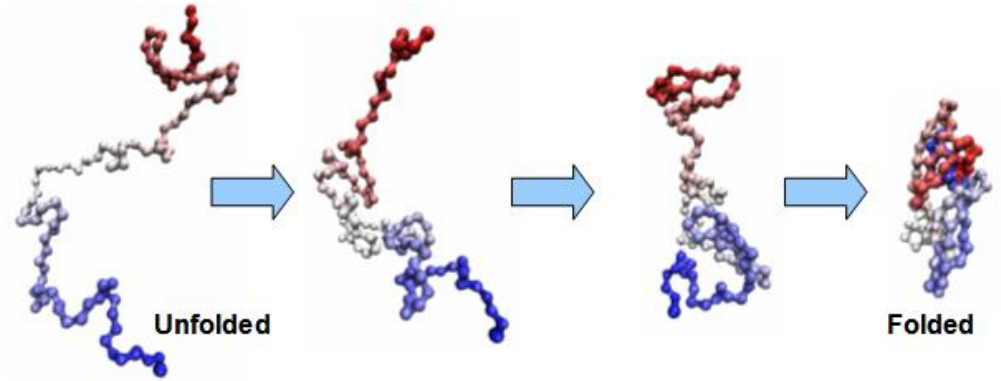
Cosmological simulations allow researchers to test theories of galaxy formation



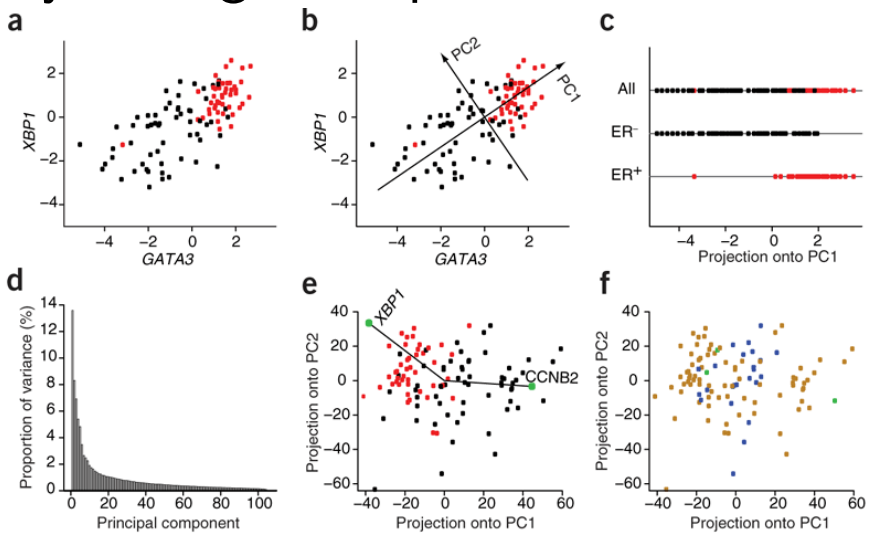
(cosmicweb.uchicago.edu)

Scientific Computing: Biology

Scientific computing is now crucial in molecular biology, e.g. protein folding (cnx.org)



Or statistical analysis of gene expression

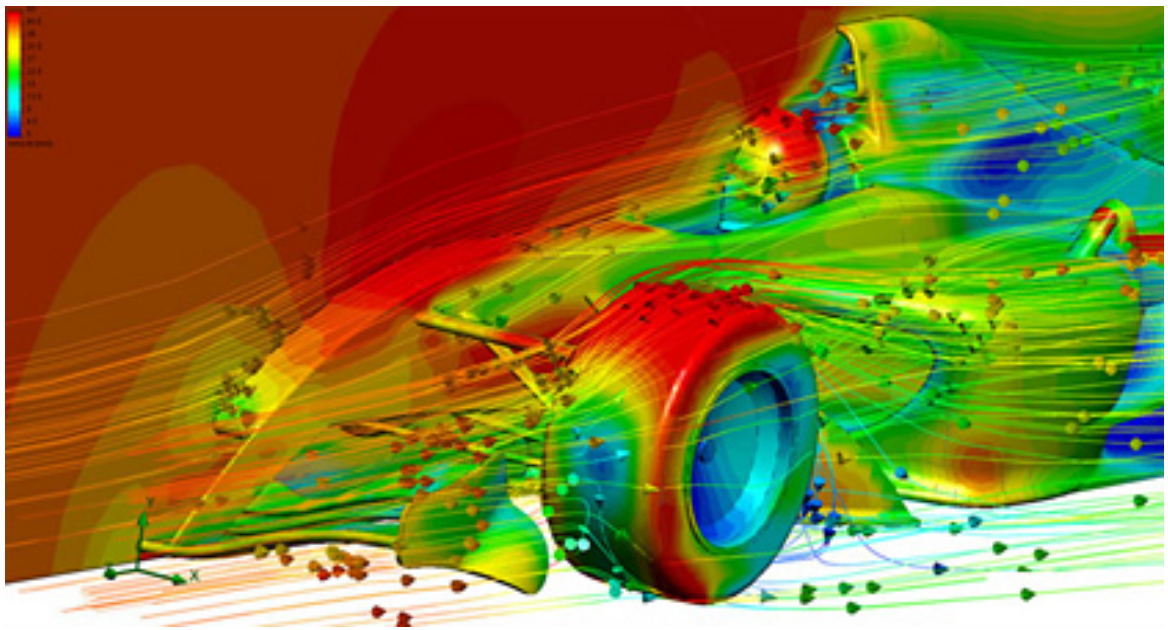


(Nature Biotechnology, 2008)

Scientific Computing: Computational Fluid Dynamics

Wind-tunnel studies are being replaced and/or complemented by CFD simulations

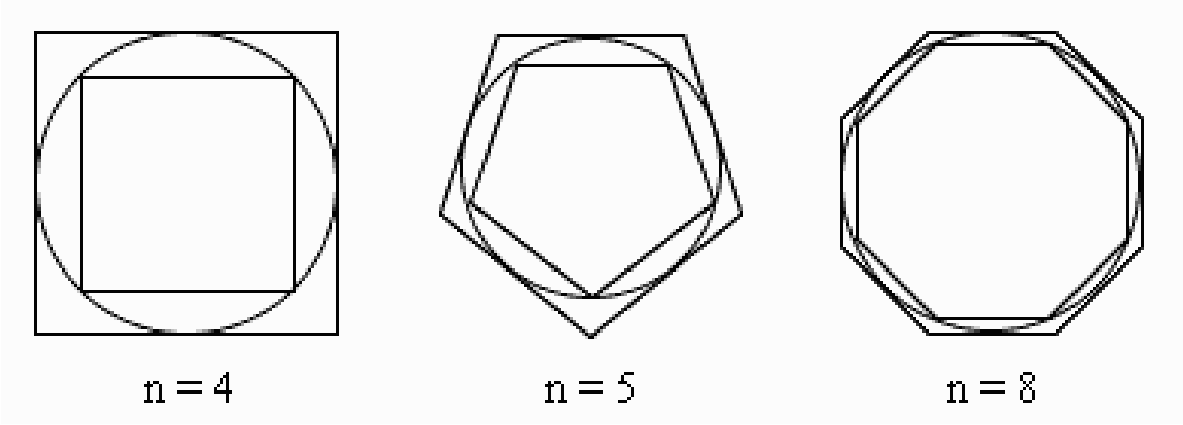
- Faster/easier/cheaper to tweak a computational design than a physical model
- Can visualize the entire flow-field to inform designers



(www.mentor.com)

What is Scientific Computing?

Archimedes' (287–212 BC) approximation of π used a recursion relation for the area of a polygon



Archimedes calculated that $3\frac{10}{71} < \pi < 3\frac{1}{7}$, an interval of 0.00201

What is Scientific Computing?

Key numerical analysis ideas captured by Archimedes:

- Approximate an infinite/continuous process (area integration) by a finite/discrete process (polygon perimeter)
- Error estimate ($3\frac{10}{71} < \pi < 3\frac{1}{7}$) is just as important as the approximation itself

What is Scientific Computing?

We will encounter algorithms from many great mathematicians: Newton, Gauss, Euler, Lagrange, Fourier, Legendre, Chebyshev, ...

They were practitioners of scientific computing (using "hand calculations"), *e.g.* for astronomy, optics, mechanics, ...

Very interested in accurate and efficient methods since hand calculations are so laborious.

Calculating π more accurately

Jame Gregory (1638–1675) discovers the arctangent series

$$\tan^{-1} x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots.$$

Putting $x = 1$ gives

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots,$$

but this is formula converges very slowly.

Formula of John Machin (1680–1752)

If $\tan\alpha = 1/5$, then

$$\tan 2\alpha = \frac{2\tan\alpha}{1-\tan^2\alpha} = \frac{5}{12} \implies \tan 4\alpha = \frac{2\tan 2\alpha}{1-\tan^2 2\alpha} = \frac{120}{119}$$

This very close to one, and hence

$$\tan\left(4\alpha - \frac{\pi}{4}\right) = \frac{\tan 4\alpha - 1}{1+\tan 4\alpha} = \frac{1}{239}$$

Taking the arctangent of both sides gives the Machin formula

$$\frac{\pi}{4} = 4\tan^{-1}\frac{1}{5} - \tan^{-1}\frac{1}{239},$$

which gives much faster convergence.

The arctangent digit hunters

1706 John Machin, *100 digits*
1719 Thomas de Lagny, *112 digits*
1739 Matsunaga Ryohitsu, *50 digits*
1794 Georgvon Vega, *140 digits*
1844 Zacharias Dase, *200 digits*
1847 Thomas Clausen, *248 digits*
1853 William Rutherford, *440 digits*
1876 William Shanks, *707 digits*

A short poem to Shanks

Seven hundred seven
Shanks did state
Digits of π he would calculate
And none can deny
It was a good try
But he erred in five twenty eight!

Scientific Computing vs. Numerical Analysis

S.C. and N.A. are closely related, each field informs the other

Emphasis of AM205 is Scientific Computing

> We focus on knowledge required for you to be a responsible user of numerical methods for practical problems

Sources of Error in Scientific Computing

There are several sources of error in solving real-world Scientific Computing problems

Some are beyond our control, e.g. uncertainty in modeling parameters or initial conditions

Some are introduced by our numerical approximations:

- Truncation/discretization: We need to make approximations in order to compute (finite differences, truncate infinite series...)
- Rounding: Computers work with *finite precision arithmetic*, which introduces rounding error

Sources of Error in Scientific Computing

It is crucial to understand and control the error introduced by numerical approximation, otherwise our results might be garbage

This is a major part of Scientific Computing, called error analysis

Error analysis became crucial with advent of modern computers: larger scale problems $\Longrightarrow$ more accumulation of numerical error

Most people are more familiar with rounding error, but discretization error is usually far more important in practice

Discretization Error vs. Rounding Error

Consider finite difference approximation to $f'(x)$:

$$f_{\text{diff}}(x; h) \equiv \frac{f(x+h) - f(x)}{h}$$

From Taylor series:

$$f(x+h) = f(x) + hf'(x) + f''(\theta)h^2/2, \text{ where } \theta \in [x, x+h]$$

we see that

$$f_{\text{diff}}(x; h) = \frac{f(x+h) - f(x)}{h} = f'(x) + f''(\theta)h/2$$

Suppose $|f''(\theta)| \leq M$, then bound on discretization error is

$$|f'(x) - f_{\text{diff}}(x; h)| \leq Mh/2$$

Discretization Error vs. Rounding Error

But we can't compute $f_{\text{diff}}(x; h)$ in exact arithmetic

Let $\tilde{f}_{\text{diff}}(x; h)$ denote finite precision approximation of $f_{\text{diff}}(x; h)$

Numerator of $\tilde{f}_{\text{diff}}$ introduces rounding error $\lesssim \epsilon f(x)$
(on modern computers $\epsilon \approx 10^{-16}$, will discuss this shortly)

Hence we have the rounding error

$$\begin{aligned} |f_{\text{diff}}(x; h) - \tilde{f}_{\text{diff}}(x; h)| &\lesssim \left| \frac{f(x+h) - f(x)}{h} - \frac{f(x+h) - f(x) + \epsilon f(x)}{h} \right| \\ &= \epsilon |f(x)|/h \end{aligned}$$

Discretization Error vs. Rounding Error

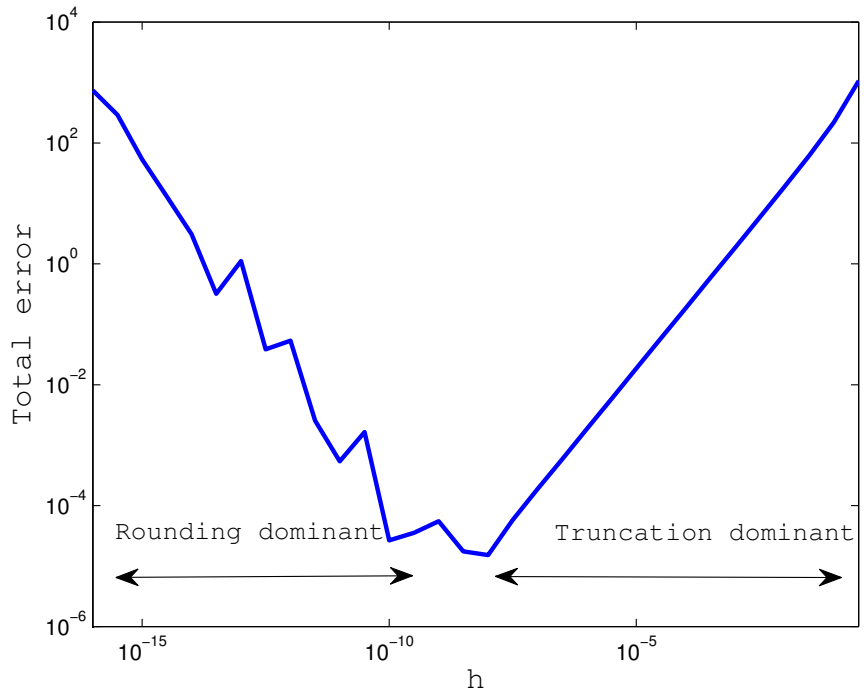
We can then use the triangle inequality ($|a+b| \leq |a| + |b|$) to bound the total error (discretization and rounding)

$$\begin{aligned} |f'(x) - \tilde{f}_{\text{diff}}(x;h)| &= |f'(x) - f_{\text{diff}}(x;h) + f_{\text{diff}}(x;h) - \tilde{f}_{\text{diff}}(x;h)| \\ &\leq |f'(x) - f_{\text{diff}}(x;h)| + |f_{\text{diff}}(x;h) - \tilde{f}_{\text{diff}}(x;h)| \\ &\leq Mh/2 + \epsilon|f(x)|/h \end{aligned}$$

Since ϵ is so small, we expect discretization error to dominate until h gets sufficiently small

Discretization Error vs. Rounding Error

For example, consider $f(x) = \exp(5x)$, f.d. error at $x = 1$ as function of h:



Exercise: Use calculus to find local minimum of error bound as a function of h to see why minimum occurs at $h \approx 10^{-8}$

Discretization Error vs. Rounding Error

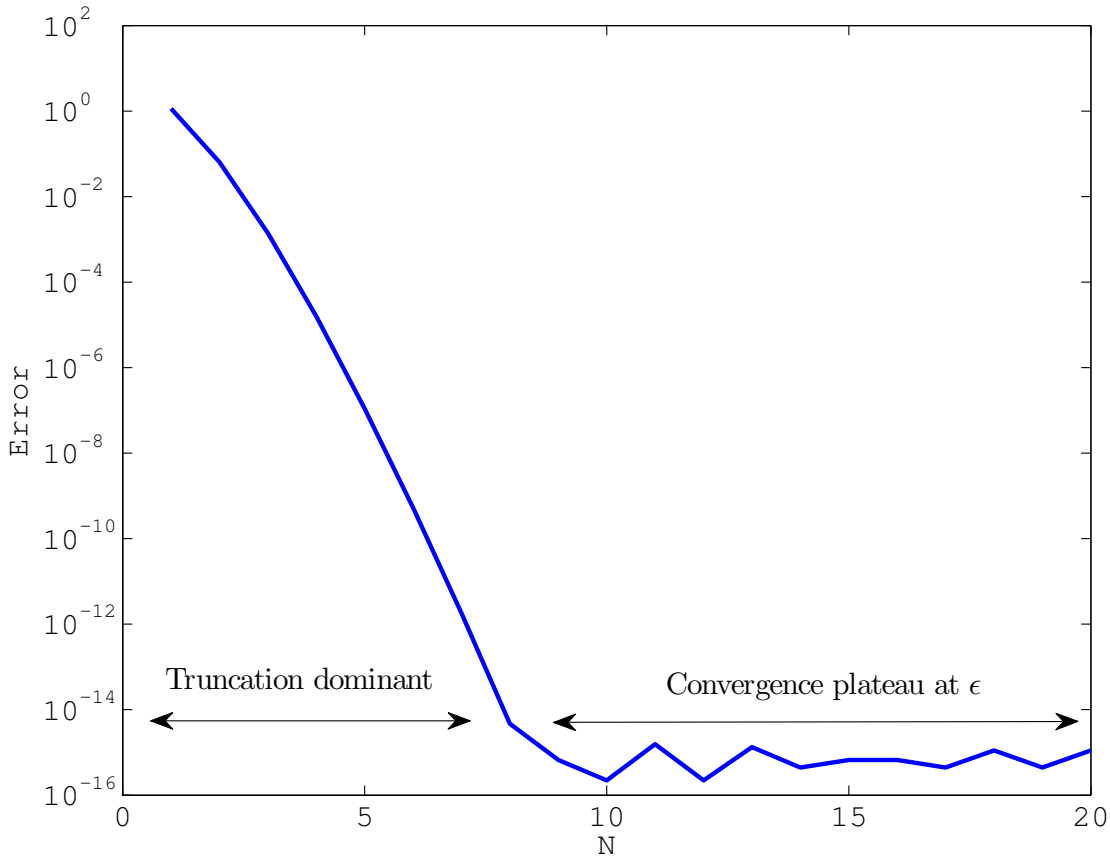
Note that this in this finite difference example, we observe error growth due to rounding as $h \to 0$

This is a nasty situation, due to factor of h on denominator in the error bound

A more common situation (that we'll see in Unit I, for example) is that the error plateaus at around ϵ due to rounding error

Discretization Error vs. Rounding Error

Error plateau:



Absolute vs. Relative Error

Recall our bound $|f'(x) - \tilde{f}_{\text{diff}}(x; h)| \leq Mh/2 + \epsilon|f(x)|/h$

This is a bound on Absolute Error:

Absolute Error $\equiv$ true value - approximate value

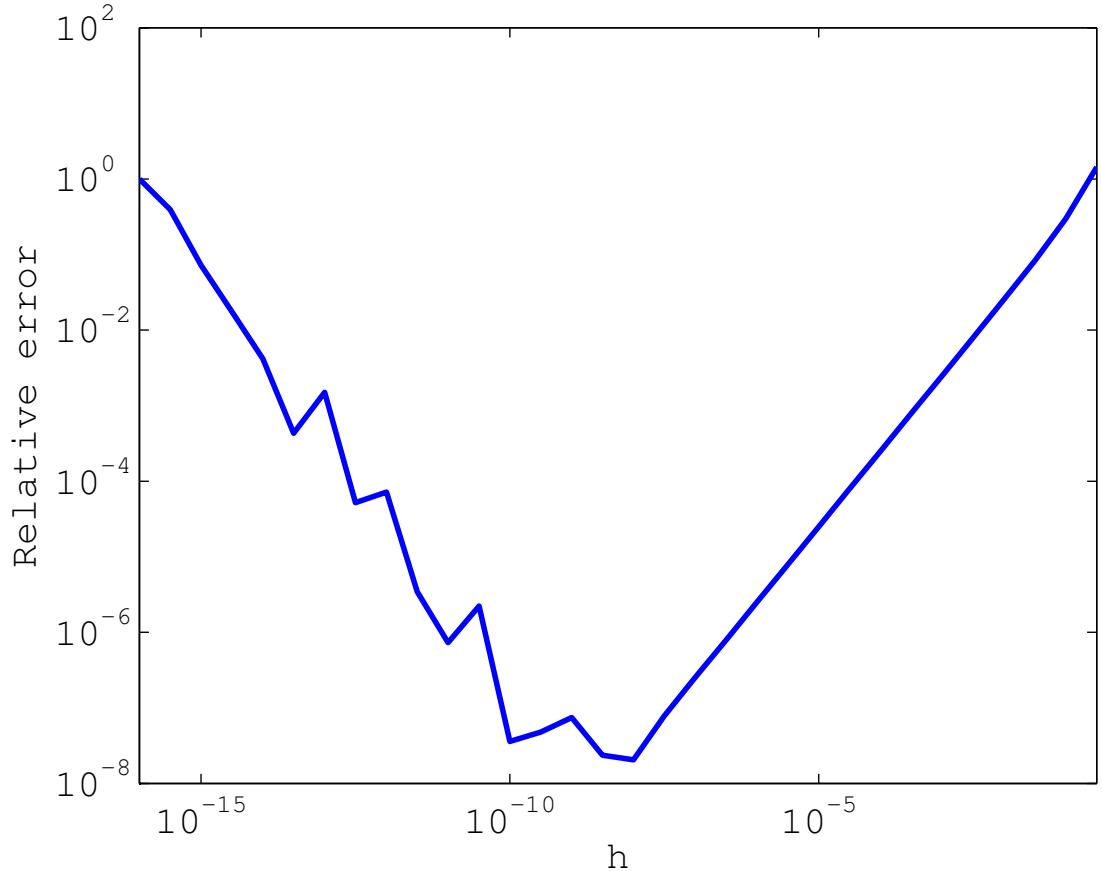
Generally more interesting to consider Relative Error:

Relative Error $\equiv$ Absolute Error / true value

Relative error takes the scaling of the problem into account

Absolute vs. Relative Error

For our finite difference example, plotting relative error just rescales the error values



Sidenote: Convergence plots

Most often we will encounter algebraic convergence, where error decreases as αh^β for some $\alpha, \beta \in \mathbb{R}$.

Algebraic convergence: If $y = \alpha h^\beta$, then

$$\log(y) = \log \alpha + \beta \log h.$$

Plotting algebraic convergence on log–log axes asymptotically

yields a straight line with gradient β.

Hence a good way to deduce the algebraic convergence rate is by comparing error to αh^β on log–log axes.

Sidenote: Convergence plots

Sometimes we will encounter exponential convergence, where error decays as $\alpha e^{-\beta N}$ as $N \to \infty$.

If $y = \alpha e^{-\beta N}$ then $\log y = \log \alpha - \beta N$.

Hene for exponential convergence, better to use semilog-y axes (like the previous "error plateau" plot.

Numerical sensitivity

In practical problems we will always have input perturbations (modeling uncertainty, rounding error)

Let $y = f(x)$, and denote perturbed input $\hat{x} = x + \Delta x$

Also, denote perturbed output by $\hat{y} = f(\hat{x})$, and $\hat{y} = y + \Delta y$

The function f is sensitive to input perturbations if $\Delta y \gg \Delta x$

This sensitivity is a property of f (i.e. not related to a numerical approximation of f)

Sensitivity and Conditioning

Hence for a sensitive problem: small input perturbation $\Longrightarrow$ large output perturbation

Can be made quantitive with concept of condition number[1]

$$\text{Condition number} \equiv \frac{|\Delta y/y|}{|\Delta x/x|}$$

$$\begin{aligned} \text{Condition number} \gg 1 \quad &\Longleftrightarrow \quad \text{small perturbations are amplified} \\ &\Longleftrightarrow \quad \text{ill-conditioned problem} \end{aligned}$$

[1] Here we introduce the relative condition number, generally more informative than the absolute condition number

Sensitivity and Conditioning

Condition number can be analyzed for different types of problem (independent of algorithm used to solve the problem), e.g.

- Function evaluation, $y = f(x)$
- Matrix multiplication, $Ax = b$ (solve for b given x)
- Matrix equation, $Ax = b$ (solve for x given b)

See Lecture: Numerical conditioning examples

Stability of an Algorithm

In practice, we solve problems by applying a numerical method to a mathematical problem, e.g. apply Gaussian elimination to $Ax = b$

To obtain an accurate answer, we need to apply a stable numerical method to a well-conditioned mathematical problem

Question: What do we mean by a stable numerical method?

Answer: Roughly speaking, the numerical method doesn't accumulate error (e.g. rounding error) and produce "garbage"

We will make this definition more precise shortly... but first, we discuss rounding error and finite-precision arithmetic