Applied Mathematics 205

Advanced Scientific Computing:
Numerical Methods

Lecturer: Chris H. Rycroft
Logistics

Lectures: Tuesday/Thursday, 10 AM–11:30 AM
Maxwell–Dworkin, Room G115

Email: chr@seas.harvard.edu

Course website:
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, 2 PM–4 PM (starting Sep 7)

TFs:   Gary Pui-Tung Choi (pchoi@g.harvard.edu)
        Yuexia Luna Lin (y_lin@g.harvard.edu)
        Shruti Mishra (shrutimishra@g.harvard.edu)
Office hour schedule will be posted 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 course)
- 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](http://www.numpy.org) and [http://www.scipy.org/](http://www.scipy.org/)

\(^2\) [http://www.matplotlib.org](http://www.matplotlib.org)
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.

Note that MATLAB shares many similarities with Python, and many numerical functions have identical names (e.g. linspace, eig), making it easy to follow the in-class examples.
Assignment 0 is posted on the course website.

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

Assignment 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).
The TFs will organize a tutorial on Python on Wednesday Sep 6th in the afternoon.

It will cover the basics of Python programming, particularly aimed at those who have some limited experience coding in other languages.
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
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 Chris Rycroft, David Knezevic, and Efthimios Kaxiras, with minor adjustments. Notes from 2013–2016 still online. Videos from 2013 still online.)
Assessment

- 60% – Five homework assignments with equal weighting
- 10% – One take-home midterm exam
- 30% – Final project
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 Harvard Canvas page (linked from main website). The written report should be individually submitted as either a PDF or Word document.

Late homework will be evaluated on a case-by-case basis.

Deadlines are Sep 22, Oct 6, Oct 20, Nov 3, Dec 1.
Assignments will be written in \LaTeX{}, which is an excellent platform for writing scientific documents and equations. While completely optional, we encourage you to try and use \LaTeX{} for your assignments. The teaching staff are all happy to talk further.

\LaTeX{} is free and available on all major computing platforms. Using \LaTeX{} involves writing a file in a simple markup language, which is then compiled into a PDF or PostScript file. See the excellent *Not So Short Introduction to \LaTeX{}* for more information.

The original \LaTeX{} files for the homework assignments will be posted to the website for reference.

---

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.
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 discussions with peers, make sure that you can work through the problem yourself and ensure that any answers you submit for evaluation are the result of your own efforts.

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 for Thursday November 9th
- Take-home exam with 48 hours to complete
- No discussion or collaboration permitted

Note that November 10th is Veterans Day, which is a university holiday, but only for staff. Special dispensations will be made for any students who prefer not to take the exam on this date.
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.\(^5\)

- Use concepts/methods related to the course to solve a problem of interest to your group.
- Project proposal in the form of an oral meeting with the teaching staff due by 6 PM on Thursday November 16th.
- Project due at 5 PM on Wednesday Dec 18th.
- Submit a report and associated code.
- Possible option to present a poster at the CS Fall poster session.

\(^5\)Single person projects may be allowed with instructor permission.
Textbooks

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

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 increased 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”

NA is the study of these algorithms, 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 focus 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 is now crucial in molecular biology, e.g. protein folding (cnx.org)

Or statistical analysis of gene expression

(Markus Ringner, Nature Biotechnology, 2008)
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)
Scientific Computing: Geophysics

In geophysics we only have data on the earth's surface.

Computational simulations allow us to test models of the interior.

(www.tacc.utexas.edu)
What is Scientific Computing?

NA and SC have been important subjects for centuries, even though the names we use today are relatively recent.

One of the earliest examples: calculation of $\pi$. Early values:

- Babylonians: $3\frac{1}{8}$
- Egyptians: $4\left(\frac{8}{9}\right)^2 \approx 3.16049$
- Quote from the Old Testament: “And he made the molten sea of ten cubits from brim to brim, round in compass, and the height thereof was five cubits; and a line of thirty cubits did compass it round about” – 1 Kings 7:23. Implies $\pi \approx 3$. 
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 $\pi$ more accurately

Jame Gregory (1638–1675) discovers the arctangent series

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

Putting $x = 1$ gives

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

but this is formula converges very slowly.
If \( \tan \alpha = \frac{1}{5} \), then

\[
\tan 2\alpha = \frac{2 \tan \alpha}{1 - \tan^2 \alpha} = \frac{5}{12} \quad \implies \quad \tan 4\alpha = \frac{2 \tan 2\alpha}{1 - \tan^2 2\alpha} = \frac{120}{119}.
\]

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  Georg von 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\textsuperscript{6}

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

\textsuperscript{6}If you would like more poems and facts about $\pi$, see slides from \textit{The Wonder of Pi}, a public lecture Chris gave at Amherst Town Library on 3/14/16.
Scientific Computing vs. Numerical Analysis

SC and NA 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 the advent of modern computers: larger scale problems $\implies$ 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

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

\[
|f_{\text{diff}}(x; h) - \tilde{f}_{\text{diff}}(x; h)| \lesssim \left| \frac{f(x + h) - f(x)}{h} - \frac{f(x + h) - f(x) + \epsilon f(x)}{h} \right| \\
\leq \epsilon |f(x)| / h
\]
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)

\[
|f'(x) - \tilde{f}_{\text{diff}}(x; h)| = |f'(x) - f_{\text{diff}}(x; h) + f_{\text{diff}}(x; h) - \tilde{f}_{\text{diff}}(x; h)| \\
\leq |f'(x) - f_{\text{diff}}(x; h)| + |f_{\text{diff}}(x; h) - \tilde{f}_{\text{diff}}(x; h)| \\
\leq \frac{Mh}{2} + \epsilon|f(x)|/h
\]

Since $\epsilon$ is so small, here 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 the denominator in the error bound

A more common situation (that we’ll see in Unit 1, for example) is that the error plateaus at around $\epsilon$ due to rounding error
Discretization Error vs. Rounding Error

Error plateau:

The graph illustrates the relationship between discretization error and rounding error. It shows that the error decreases as the number of iterations increases, approaching a convergence plateau at a certain error level. The error is characterized by a truncation dominant phase where the error decreases significantly initially, followed by a convergence plateau at a specific error level, $\epsilon$. This indicates that the rounding error becomes dominant in the later stages of the process.
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\(^7\):

\[
\text{Absolute Error} \equiv \text{true value} - \text{approximate value}
\]

Generally more interesting to consider Relative Error:

\[
\text{Relative Error} \equiv \frac{\text{Absolute Error}}{\text{true value}}
\]

Relative error takes the scaling of the problem into account

\(^7\)We generally don't know the true value, we often have to use a surrogate for the true value, e.g. an accurate approximation using a different method
For our finite difference example, plotting relative error just rescales the error values.
Sidenote: Convergence plots

We have shown several plots of error as a function of a discretization parameter.

In general, these plots are very important in scientific computing to demonstrate that a numerical method is behaving as expected.

To display convergence data in a clear way, it is important to use appropriate axes for our plots.
Sidenote: Convergence plots

Most often we will encounter algebraic convergence, where error decreases as $\alpha h^\beta$ 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 $\beta$.

Hence a good way to deduce the algebraic convergence rate is by comparing error to $\alpha h^\beta$ 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 $\log y = \log \alpha - \beta N$.

Hence 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 is sensitivity inherent in $f$, independent of any approximation (though approximation $\hat{f} \approx f$ can exacerbate sensitivity)
Sensitivity and Conditioning

For a sensitive problem, small input perturbation $\implies$ large output perturbation

Can be made quantitative with concept of condition number

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

Condition number $\gg 1$ $\iff$ small perturbations are amplified
$\iff$ ill-conditioned problem

---

\(^8\)Here we introduce the relative condition number, generally more informative than the absolute condition number.
Sensitivity and Conditioning

Condition number can be analyzed for all sorts of different problem types (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 notes: 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