Numerical Analysis / Scientific Computing

CS450

Andreas Kloeckner

Spring 2019

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:

- 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)

- 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?

- 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*?

- 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-s19

- Assignments
  - HW0!
  - 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

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*?

- 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?

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

Demo: Truncation vs Rounding

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.

$$\textit{Absolute error} = \text{approx value} \; - \; \text{true value}$$

$$\textit{Relative error} = \frac{\text{Absolute error}}{\text{True value}}$$

Problem: True value not known

- Estimate
- 'How big at worst?' $\rightarrow$ Establish *Upper Bounds*

Recap: Norms

What's a norm?

- $f(\mathrm{x}) : \mathbb{R}^n \to \mathbb{R}_0^+$, returns a 'magnitude' of the input vector
- In symbols: Often written $\|\mathrm{x}\|$.

Define *norm*.

A function $\|\mathrm{x}\| : \mathbb{R}^n \to \mathbb{R}_0^+$ is called a norm if and only if

1. $\|\mathrm{x}\| > 0 \Leftrightarrow \mathrm{x} \neq 0$.
2. $\|\gamma \mathrm{x}\| = |\gamma| \, \|\mathrm{x}\|$ for all scalars γ.
3. Obeys triangle inequality $\|\mathrm{x} + \mathrm{y}\| \leqslant \|\mathrm{x}\| + \|\mathrm{y}\|$

Norms: Examples

Examples of norms?

The so-called *p-norms*:

$$\left\| \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \right\|_p = \sqrt[p]{|x_1|^p + \cdots + |x_n|^p} \quad (p \geqslant 1)$$

$p = 1, 2, \infty$ particularly important

Demo: Vector Norms

Norms: Which one?

Does the choice of norm really matter much?

In finitely many dimensions, all *norms are equivalent*.
I.e. for fixed n and two norms $\|\cdot\|, \|\cdot\|^*$, there exist $\alpha, \beta > 0$ so that for all vectors $\mathbf{x} \in \mathbb{R}^n$

$$\alpha \|\mathbf{x}\| \leqslant \|\mathbf{x}\|^* \leqslant \beta \|\mathbf{x}\| .$$

So: No, doesn't matter *that much*. Will start mattering more for so-called *matrix norms*–see later.

Norms and Errors

If we're computing a vector result, the error is a vector.
That's not a very useful answer to 'how big is the error'.
What can we do?

Apply a norm!

How? *Attempt 1:*

$$\text{Magnitude of error} \neq \|\text{true value}\| - \|\text{approximate value}\|$$

WRONG! (How does it fail?)
Attempt 2:

$$\text{Magnitude of error} = \|\text{true value} - \text{approximate value}\|$$

Forward/Backward Error

Suppose *want* to compute $y = f(x)$, but *approximate* $\hat{y} = \hat{f}(x)$.

What are the forward error and the backward error?

Forward error: $\Delta y = \hat{y} - y$

Backward error: Imagine *all* error came from feeding the wrong input into a fully accurate calculation. Backward error is the difference between true and ‘wrong’ input. I.e.

- Find an $\hat{x}$ so that $f(\hat{x}) = \hat{y}$.
- $\Delta x = \hat{x} - x$.

$$\begin{tikzcd}
x \arrow[r, "f"] \arrow[dr, "\hat{f}"] \arrow[d, leftrightarrow, "\text{bw. err.}"'] & f(x) \arrow[d, leftrightarrow, "\text{fw err.}"] \\
\hat{x} \arrow[r, "f"] & \hat{y} = f(\hat{x})
\end{tikzcd}$$

Forward/Backward Error: Example

Suppose you wanted $y = \sqrt{2}$ and got $\hat{y} = 1.4$.
What's the (magnitude of) the forward error?

$$|\Delta y| = |1.4 - 1.41421\ldots| \approx 0.0142\ldots$$

Relative forward error:

$$\frac{|\Delta y|}{|y|} = \frac{0.0142\ldots}{1.41421\ldots} \approx 0.01.$$

About 1 percent, or *two accurate digits*.

Forward/Backward Error: Example

Suppose you wanted $y = \sqrt{2}$ and got $\hat{y} = 1.4$.
What's the (magnitude of) the backward error?

Need $\hat{x}$ so that $f(\hat{x}) = 1.4$.

$$\sqrt{1.96} = 1.4, \quad \Rightarrow \quad \hat{x} = 1.96.$$

Backward error:

$$|\Delta x| = |1.96 - 2| = 0.04.$$

Relative backward error:

$$\frac{|\Delta x|}{|x|} \approx 0.02.$$

About 2 percent.

Forward/Backward Error: Observations

What do you observe about the relative manitude of the relative errors?

- In this case: Got smaller, i.e. variation *damped out*.
- Typically: Not that lucky: Input error *amplified*.

This amplification factor seems worth studying in more detail.

Sensitivity and Conditioning

What can we say about amplification of error?

Define *condition number* as smallest number κ so that

$$|\text{rel. fwd. err.}| \leqslant \kappa \cdot |\text{rel. bwd. err.}|$$

Or, somewhat sloppily, with x/y notation as in previous example:

$$\text{cond} = \max_x \frac{|\Delta y| \,/\, |y|}{|\Delta x| \,/\, |x|}.$$

(Technically: should use 'supremum'.)

If the condition number is...

- ...small: the problem *well-conditioned* or insensitive
- ...large: the problem *ill-conditioned* or sensitive

Can also talk about condition number for a single input x.

Example: Condition Number of Evaluating a Function

$y = f(x)$. Assume f differentiable.

$$\kappa = \max_x \frac{|\Delta y| / |y|}{|\Delta x| / |x|}$$

Forward error:

$$\Delta y = f(x + \Delta x) - f(x) = f'(x)\Delta x$$

Condition number:

$$\kappa \geqslant \frac{|\Delta y| / |y|}{|\Delta x| / |x|} = \frac{|f'(x)| |\Delta x| / |f(x)|}{|\Delta x| / |x|} = \frac{|xf'(x)|}{|f(x)|}.$$

Demo: Conditioning of Evaluating tan

Stability and Accuracy

Previously: Considered *problems* or *questions*.

Next: Considered *methods*, i.e. computational approaches to find solutions.

When is a method *accurate*?

Closeness of method output to true answer for unperturbed input.

When is a method *stable*?

- "A method is stable if the result it produces is the exact solution to a nearby problem."
- The above is commonly called *backward stability* and is a stricter requirement than just the temptingly simple:

  If the method's sensitivity to variation in the input is no (or not much) greater than that of the problem itself.

Note: Necessarily includes insensitivity to variation in intermediate results

Getting into Trouble with Accuracy and Stability

How can I produce inaccurate results?

- Apply an inaccurate method
- Apply an unstable method to a well-conditioned problem
- Apply any type of method to an ill-conditioned problem

In-Class Activity: Forward/Backward Error

In-class activity: Forward/Backward Error

Wanted: Real Numbers... in a computer

Computers can represent *integers*, using bits:

$$23 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = (10111)_2$$

How would we represent fractions?

Idea: Keep going down past zero exponent:

$$23.625 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$$

So: Could store

- a fixed number of bits with exponents $\geqslant 0$
- a fixed number of bits with exponents < 0

This is called *fixed-point arithmetic*.

Fixed-Point Numbers

Suppose we use units of 64 bits, with 32 bits for exponents $\geqslant 0$ and 32 bits for exponents < 0. What numbers can we represent?

2^{31}	$\cdots$	2^{0}	2^{-1}	$\cdots$	2^{-32}

Smallest: $2^{-32} \approx 10^{-10}$
Largest: $2^{31} + \cdots + 2^{-32} \approx 10^{9}$

How many 'digits' of relative accuracy (think relative rounding error) are available for the smallest vs. the largest number?

For large numbers: about 19
For small numbers: few or none

Idea: Instead of *fixing* the location of the 0 exponent, let it *float*.

Floating Point Numbers

Convert $13 = (1101)_2$ into floating point representation.

$$13 = 2^3 + 2^2 + 2^0 = (1.101)_2 \cdot 2^3$$

What pieces do you need to store an FP number?

Significand: $(1.101)_2$
Exponent: 3

Floating Point: Implementation, Normalization

Previously: Consider *mathematical* view of FP.

Next: Consider *implementation* of FP in hardware.

Do you notice a source of inefficiency in our number representation?

Idea: Notice that the leading digit (in binary) of the significand is always one.

Only store '101'. Final storage format:

Significand: 101 – a fixed number of bits

Exponent: 3 – a (*signed!*) integer allowing a certain range

Exponent is most often stored as a positive 'offset' from a certain negative number. E.g.

$$3 = \underbrace{-1023}_{\text{implicit offset}} + \underbrace{1026}_{\text{stored}}$$

Actually stored: 1026, a positive integer.

Unrepresentable numbers?

Can you think of a somewhat central number that we cannot represent as

$$x = (1.\_\_\_\_\_\_\_\_\_\_\_)_2 \cdot 2^{-p}?$$

Zero. Which is somewhat embarrassing.

Core problem: The implicit 1. It's a great idea, were it not for this issue.

Have to break the pattern. Idea:

- Declare one exponent 'special', and turn off the leading one for that one.
  (say, -1023, a.k.a. stored exponent 0)
- For all larger exponents, the leading one remains in effect.

Bonus Q: With this convention, what is the binary representation of a zero?

Demo: Picking apart a floating point number

Subnormal Numbers

What is the smallest representable number in an FP system with 4 stored bits in the significand and an exponent range of $[-7, 7]$?

First attempt:

- Significand as small as possible $\rightarrow$ all zeros after the implicit leading one
- Exponent as small as possible: -7

So:

$$(1.0000)_2 \cdot 2^{-7}.$$

Unfortunately: wrong.

Subnormal Numbers II

What is the smallest representable number in an FP system with 4 stored bits in the significand and an exponent range of $[-7, 7]$? (Attempt 2)

We can go way smaller by using the special exponent (which turns off the implicit leading one). We'll assume that the special exponent is -8. So: $(0.0001)_2 \cdot 2^{-8}$.
Numbers with the special epxonent are called *subnormal* (or *denormal*) FP numbers. Technically, zero is also a subnormal.

Note: It is thus quite natural to 'park' the special exponent at the low end of the exponent range.

Why learn about subnormals?

Because computing with them is often slow, because it is implemented using 'FP assist', i.e. not in actual hardware. Many C compilers support options to 'flush subnormals to zero'.

Underflow

- FP systems without subnormals will *underflow* (return 0) as soon as the exponent range is exhausted.
- This smallest representable *normal* number is called the *underflow level*, or *UFL*.
- Beyond the underflow level, subnormals provide for *gradual underflow* by 'keeping going' as long as there are bits in the significand, but it is important to note that subnormals don't have as many accurate digits as normal numbers.
- Analogously (but much more simply–no 'supernormals'): the overflow level, *OFL*.

Rounding Modes

How is rounding performed? (Imagine trying to represent π.)

$$(\underbrace{1.1101010}_{\text{representable}}11)_2$$

- "Chop" a.k.a. *round-to-zero*: $(1.1101010)_2$
- *Round-to-nearest*: $(1.1101011)_2$ (most accurate)

What is done in case of a tie? $0.5 = (0.1)_2$ ("Nearest"?)

Up or down? It turns out that picking the same direction every time introduces *bias*. Trick: *round-to-even*.

$$0.5 \to 0, \qquad 1.5 \to 2$$

Demo: Density of Floating Point Numbers

Demo: Floating Point vs Program Logic

Smallest Numbers Above...

- What is smallest FP number > 1? Assume 4 bits in the significand.

$$(1.0001)_2 \cdot 2^0 = x \cdot (1 + 0.0001)_2$$

What's the smallest FP number > 1024 in that same system?

$$(1.0001)_2 \cdot 2^{10} = x \cdot (1 + 0.0001)_2$$

Can we give that number a name?

Unit Roundoff

Unit roundoff or *machine precision* or *machine epsilon* or $\varepsilon_{\text{mach}}$ is the smallest number such that

$$\text{float}(1+\varepsilon) > 1.$$

- Assuming round-to-nearest, in the above system, $\varepsilon_{\text{mach}} = (0.00001)_2$.
- Note the extra zero.
- Another, related, quantity is *ULP*, or *unit in the last place*. $(\varepsilon_{\text{mach}} = 0.5\,\text{ULP})$

FP: Relative Rounding Error

What does this say about the relative error incurred in floating point calculations?

- The factor to get from one FP number to the next larger one is (mostly) independent of magnitude: $1 + \varepsilon_{\text{mach}}$.
- Since we can't represent any results between x and $x \cdot (1 + \varepsilon_{\text{mach}})$, that's really the minimum error incurred.
- In terms of relative error:

$$\left|\frac{\tilde{x} - x}{x}\right| = \left|\frac{x(1 + \varepsilon_{\text{mach}}) - x}{x}\right| = \varepsilon_{\text{mach}}.$$

  At least theoretically, $\varepsilon_{\text{mach}}$ is the maximum relative error in any FP operations. (Practical implementations do fall short of this.)

FP: Machine Epsilon

What's that same number for double-precision floating point? (52 bits in the significand)

$$2^{-53} \approx 10^{-16}$$

Bonus Q: What does $1 + 2^{-53}$ do on your computer? Why?

We can expect FP math to consistently introduce little relative errors of about 10^{-16}.

Working in double precision gives you about 16 (decimal) accurate digits.

Demo: Floating Point and the Harmonic Series

In-Class Activity: Floating Point

In-class activity: Floating Point

Implementing Arithmetic

How is floating point addition implemented?
Consider adding $a = (1.101)_2 \cdot 2^1$ and $b = (1.001)_2 \cdot 2^{-1}$ in a system with three bits in the significand.

Rough algorithm:

1. Bring both numbers onto a common exponent
2. Do grade-school addition from the front, until you run out of digits in your system.
3. Round result.

$$\begin{aligned} a &= 1.\quad 101 \cdot 2^1 \\ b &= 0.\quad 01001 \cdot 2^1 \\ a+b &\approx 1.\quad 111 \cdot 2^1 \end{aligned}$$

Problems with FP Addition

What happens if you subtract two numbers of very similar magnitude?
As an example, consider $a = (1.1011)_2 \cdot 2^0$ and $b = (1.1010)_2 \cdot 2^0$.

$$\begin{aligned} a &= 1. \quad 1011 \cdot 2^1 \\ b &= 1. \quad 1010 \cdot 2^1 \\ a - b &\approx 0. \quad 0001???? \cdot 2^1 \end{aligned}$$

or, once we normalize,

$$1.???? \cdot 2^{-3}.$$

There is no data to indicate what the missing digits should be.
$\rightarrow$ Machine fills them with its ‘best guess’, which is not often good.

This phenomenon is called *Catastrophic Cancellation*.

Demo: Catastrophic Cancellation

Supplementary Material

- Josh Haberman, Floating Point Demystified, Part 1
- David Goldberg, What every computer programmer should know about floating point

Outline

Introduction to Scientific Computing

Systems of Linear Equations
- Theory: Conditioning
- Methods to Solve Systems

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

Solving a Linear System

Given:

- $m \times n$ matrix A
- m-vector b

What are we looking for here, and when are we allowed to ask the question?

Want: n-vector x so that

$$A\mathrm{x} = \mathrm{b}.$$

- Linear combination of columns of A to yield b.
- Restrict to square case ($m = n$) for now.
- Even with that: solution may not exist, or may not be unique.

Unique solution exists iff A is *nonsingular*.

Next: Want to talk about conditioning of this operation. Need to measure distances of matrices.

Matrix Norms

What norms would we apply to matrices?

Easy answer: ‘*Flatten*’ matrix as vector, use vector norm.
Not very meaningful.
Instead: Choose norms for matrices to interact with an ‘associated’ vector norm $\|\cdot\|$ so that $\|A\|$ obeys

$$\|Ax\| \leqslant \|A\| \, \|x\| \, .$$

This can be achieved by choosing, for a given vector norm $\|\cdot\|$,

$$\|A\| := \max_{\|x\|=1} \|Ax\| \, .$$

This is called the matrix norm.
For each vector norm, we get a different matrix norm, e.g. for the vector 2-norm $\|x\|_2$ we get a matrix 2-norm $\|A\|_2$.

Matrix Norm Properties

What is $\|A\|_1$? $\|A\|_\infty$?

$$\begin{aligned}\|A\|_1 &= \max_{\text{col } j} \sum_{\text{row } i} |A_{i,j}|,\\ \|A\|_\infty &= \max_{\text{row } i} \sum_{\text{col } j} |A_{i,j}|.\end{aligned}$$

The matrix 2-norm? Is actually fairly difficult to evaluate. See later.

How do matrix and vector norms relate for $n \times 1$ matrices?

They agree. Why? For $n \times 1$, the vector x in Ax is just a scalar:

$$\max_{\|\mathrm{x}\|=1} \|A\mathrm{x}\| = \max_{x \in \{-1,1\}} \|Ax\| = \|A[:,1]\|$$

This can help to remember 1- and ∞-norm.

Properties of Matrix Norms

Matrix norms inherit the vector norm properties:

- $\|A\| > 0 \Leftrightarrow A \neq 0$.
- $\|\gamma A\| = |\gamma| \, \|A\|$ for all scalars γ.
- Obeys triangle inequality $\|A + B\| \leqslant \|A\| + \|B\|$

But also some more properties that stem from our definition:

- $\|A\mathrm{x}\| \leqslant \|A\| \, \|\mathrm{x}\|$
- $\|AB\| \leqslant \|A\| \, \|B\|$ (easy consequence)

Both of these are called *submultiplicativity* of the matrix norm.

Conditioning

What is the condition number of solving a linear system $A\mathrm{x} = \mathrm{b}$?

Input: b with error $\Delta\mathrm{b}$,
Output: x with error $\Delta\mathrm{x}$.

Observe $A(\mathrm{x} + \Delta\mathrm{x}) = (\mathrm{b} + \Delta\mathrm{b})$, so $A\Delta\mathrm{x} = \Delta\mathrm{b}$.

$$\begin{aligned}
\frac{\text{rel err. in output}}{\text{rel err. in input}} &= \frac{\|\Delta\mathrm{x}\| \,/\, \|\mathrm{x}\|}{\|\Delta\mathrm{b}\| \,/\, \|\mathrm{b}\|} = \frac{\|\Delta\mathrm{x}\|\,\|\mathrm{b}\|}{\|\Delta\mathrm{b}\|\,\|\mathrm{x}\|} \\
&= \frac{\left\|A^{-1}\Delta\mathrm{b}\right\|\,\|A\mathrm{x}\|}{\|\Delta\mathrm{b}\|\,\|\mathrm{x}\|} \\
&\leqslant \left\|A^{-1}\right\|\,\|A\|\,\frac{\|\Delta\mathrm{b}\|\,\|\mathrm{x}\|}{\|\Delta\mathrm{b}\|\,\|\mathrm{x}\|} \\
&= \left\|A^{-1}\right\|\,\|A\|\,.
\end{aligned}$$

Conditioning of Linear Systems: Observations

Showed $\kappa(\text{Solve } A\mathrm{x} = \mathrm{b}) \le \left\|A^{-1}\right\| \|A\|$.
I.e. found an *upper bound* on the condition number. With a little bit of fiddling, it's not too hard to find examples that achieve this bound, i.e. that it is *sharp*.

So we've found the *condition number of linear system solving*, also called the condition number of the matrix A:

$$\operatorname{cond}(A) = \kappa(A) = \|A\| \left\|A^{-1}\right\|.$$

Conditioning of Linear Systems: More properties

- cond is relative to a given norm. So, to be precise, use

$$\text{cond}_2 \quad \text{or} \quad \text{cond}_\infty .$$

- If A^{-1} does not exist: $\text{cond}(A) = \infty$ by convention.

What is $\kappa(A^{-1})$?

$\kappa(A)$

What is the condition number of matrix-vector multiplication?

$\kappa(A)$ because it is equivalent to solving with A^{-1}.

Demo: Condition number visualized
Demo: Conditioning of 2x2 Matrices

Residual Vector

What is the residual vector of solving the linear system

$$\mathrm{b} = A\mathrm{x}?$$

It's the thing that's 'left over'. Suppose our approximate solution is $\widehat{\mathrm{x}}$. Then the residual vector is

$$\mathrm{r} = \mathrm{b} - A\widehat{\mathrm{x}}.$$

Residual and Error: Relationship

How do the (norms of the) residual vector r and the error $\Delta x = x - \widehat{x}$ relate to one another?

$$\begin{aligned} \|\Delta x\| &= \|x - \widehat{x}\| \\ &= \left\|A^{-1}(b - A\widehat{x})\right\| \\ &= \left\|A^{-1}r\right\| \end{aligned}$$

Divide both sides by $\|\widehat{x}\|$:

$$\frac{\|\Delta x\|}{\|\widehat{x}\|} = \frac{\left\|A^{-1}r\right\|}{\|\widehat{x}\|} \leqslant \frac{\left\|A^{-1}\right\| \|r\|}{\|\widehat{x}\|} = \operatorname{cond}(A)\frac{\|r\|}{\|A\| \|\widehat{x}\|}.$$

- rel err $\leqslant$ cond $\cdot$ rel resid
- Given small (rel.) residual, (rel.) error is only (guaranteed to be) small if the condition number is also small.

Changing the Matrix

So far, all our discussion was based on changing the right-hand side, i.e.

$$A\mathrm{x} = \mathrm{b} \quad \to \quad A\widehat{\mathrm{x}} = \widehat{\mathrm{b}}.$$

The matrix consists of FP numbers, too–it, too, is approximate. I.e.

$$A\mathrm{x} = \mathrm{b} \quad \to \quad \hat{A}\widehat{\mathrm{x}} = \mathrm{b}.$$

What can we say about the error now?

Consider $\Delta\mathrm{x} = \widehat{\mathrm{x}} - \mathrm{x} = A^{-1}(A\widehat{\mathrm{x}} - \mathrm{b}) = -A^{-1}\Delta A\widehat{\mathrm{x}}$. Thus

$$\|\Delta\mathrm{x}\| \leqslant \left\|A^{-1}\right\| \|\Delta A\| \|\widehat{\mathrm{x}}\|.$$

And we get

$$\frac{\|\Delta\mathrm{x}\|}{\|\widehat{\mathrm{x}}\|} \leqslant \mathrm{cond}(A)\frac{\|\Delta A\|}{\|A\|}.$$

Changing Condition Numbers

Once we have a matrix A in a linear system $A\mathrm{x} = \mathrm{b}$, are we stuck with its condition number? Or could we improve it?

Diagonal scaling is a simple strategy that sometimes helps.

- Row-wise: $DA\mathrm{x} = D\mathrm{b}$
- Column-wise: $AD\widehat{\mathrm{x}} = \mathrm{b}$
  Different $\widehat{\mathrm{x}}$: Recover $\mathrm{x} = D\widehat{\mathrm{x}}$.

What is this called as a general concept?

Preconditioning

- Left' preconditioning: $MA\mathrm{x} = M\mathrm{b}$
- Right preconditioning: $AM\widehat{\mathrm{x}} = \mathrm{b}$
  Different $\widehat{\mathrm{x}}$: Recover $\mathrm{x} = M\widehat{\mathrm{x}}$.

In-Class Activity: Matrix Norms and Conditioning

In-class activity: Matrix Norms and Conditioning

Solving Systems: Triangular matrices

Solve

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}.$$

- Rewrite as individual equations.
- This process is called back-substitution.
- The analogous process for lower triangular matrices is called forward substitution.

Demo: Coding back-substitution

What about non-triangular matrices?

Can do *Gaussian Elimination*, just like in linear algebra class.

Gaussian Elimination

Demo: Vanilla Gaussian Elimination

What do we get by doing Gaussian Elimination?

Row Echelon Form.

How is that different from being upper triangular?

Zeros allowed on and above the diagonal.

What if we do not just eliminate downward but also upward?

That's called *Gauss-Jordan elimination*. Turns out to be computationally inefficient. We won't look at it.

Elimination Matrices

What does this matrix do?

$$\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}$$

- Add $(-1/2)\times$ the first row to the third row.
- One elementary step in Gaussian elimination
- Matrices like this are called *Elimination Matrices*

About Elimination Matrices

Are elimination matrices invertible?

Sure! Inverse of

$$\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}$$

should be

$$\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ +\frac{1}{2} & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}.$$

More on Elimination Matrices

Demo: Elimination matrices I

Idea: With enough elimination matrices, we should be able to get a matrix into row echelon form.

$$M_\ell M_{\ell-1}\cdots M_2 M_1 A = \langle \text{Row Echelon Form } U \text{ of } A\rangle.$$

So what do we get from many combined elimination matrices like that?

(a lower triangular matrix)

Demo: Elimination Matrices II

Summary on Elimination Matrices

- El.matrices with off-diagonal entries in a single column just "merge" when multiplied by one another.
- El.matrices with off-diagonal entries in different columns merge when we multiply (left-column) * (right-column) but not the other way around.
- Inverse: Flip sign below diagonal

LU Factorization

Can build a *factorization* from elimination matrices. How?

$$A = \underbrace{M_1^{-1} M_2^{-1} \overset{-1}{\cdots} M_{\ell-1}^{-1} M_\ell^{-1}}_{\text{lower } \triangle \text{ mat } L} U = LU.$$

This is called LU factorization (or LU decomposition).

Solving $Ax = b$

Does LU help solve $Ax = b$?

$$\begin{aligned} Ax &= b \\ L\underbrace{Ux}_{y} &= b \\ Ly &= b && \leftarrow && \text{solvable by fwd. subst.} \\ Ux &= y && \leftarrow && \text{solvable by bwd. subst.} \end{aligned}$$

Now know x that solves $Ax = b$.

Demo: LU factorization

LU: Failure Cases?

Is LU/Gaussian Elimination bulletproof?

No, very much not:

$$A = \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix}.$$

Q: Is this a problem with the process or with the entire *idea* of LU?

$$\begin{array}{cc} & \begin{bmatrix} u_{11} & u_{12} \\ & u_{22} \end{bmatrix} \\ \begin{bmatrix} 1 & \\ \ell_{21} & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix} \end{array} \quad \rightarrow \quad \begin{array}{l} u_{11} = 0 \\ \underbrace{u_{11} \cdot \ell_{21}}_{0} + 1 \cdot 0 = 2 \end{array}$$

It turns out to be that A doesn't *have* an LU factorization.

Saving the LU Factorization

What can be done to get something *like* an LU factorization?

Idea: In Gaussian elimination: simply swap rows, equivalent linear system.

Approach:

- Good Idea: Swap rows if there's a zero in the way
- Even better Idea: Find the largest entry (by absolute value), swap it to the top row.

The entry we divide by is called the *pivot*.
Swapping rows to get a bigger pivot is called *(partial) pivoting*.

Recap: Permuation Matrices

How do we capture 'row switches' in a factorization?

$$\underbrace{\begin{bmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{bmatrix}}_{P} \begin{bmatrix} A & A & A & A \\ B & B & B & B \\ C & C & C & C \\ D & D & D & D \end{bmatrix} = \begin{bmatrix} A & A & A & A \\ C & C & C & C \\ B & B & B & B \\ D & D & D & D \end{bmatrix}.$$

P is called a *permutation matrix*.
Q: What's P^{-1}?

Fixing nonexistence of LU

What does LU with permutations process look like?

$P_1 A$	Pivot first column
$M_1 P_1 A$	Eliminate first column
$P_2 M_1 P_1 A$	Pivot second column
$M_2 P_2 M_1 P_1 A$	Eliminate second column
$P_3 M_2 P_2 M_1 P_1 A$	Pivot third column
$M_3 P_3 M_2 P_2 M_1 P_1 A$	Eliminate third column

Or

$$A = P_1 M_1^{-1} P_2 M_2^{-1} P_3 M_3^{-1} U.$$

Unfortunately, P's and M's don't commute, so it's not obvious how to get a lower-triangular L.

Demo: LU with Partial Pivoting (Part I)

What about the L in LU?

Sort out what LU with pivoting looks like. Have: $M_3P_3M_2P_2M_1P_1A = U$.

Define: $L_3 := M_3$
Define $L_2 := P_3M_2P_3^{-1}$
Define $L_1 := P_3P_2M_1P_2^{-1}P_3^{-1}$

$$\begin{aligned}&(L_3L_2L_1)(P_3P_2P_1)\\ =&M_3(P_3M_2P_3^{-1})(P_3P_2M_1P_2^{-1}P_3^{-1})P_3P_2P_1\\ =&M_3P_3M_2P_2M_1P_1 \quad (!)\end{aligned}$$

$$\underbrace{P_3P_2P_1}_{P}A = \underbrace{L_1^{-1}L_2^{-1}L_3^{-1}}_{L}U.$$

$L_1, \ldots, L_3$ are still lower triangular!

Q: Outline the solve process with pivoted LU.

Demo: LU with Partial Pivoting (Part II)

Computational Cost

What is the computational cost of multiplying two $n \times n$ matrices?

$O(n^3)$

What is the computational cost of carrying out LU factorization on an $n \times n$ matrix?

Recall

$$M_3 P_3 M_2 P_2 M_1 P_1 A = U \ldots$$

so $O(n^4)$?!!!

Fortunately not: Multiplications with permuation matrices and elimination matrices only cost $O(n^2)$.

So overall cost of LU is just $O(n^3)$.

Demo: Complexity of Mat-Mat multiplication and LU

More cost concerns

What's the cost of solving $Ax = b$?

LU: $O(n^3)$
FW/BW Subst: $2 \times O(n^2) = O(n^2)$

What's the cost of solving $Ax = b_1, b_2, \ldots, b_n$?

LU: $O(n^3)$
FW/BW Subst: $2n \times O(n^2) = O(n^3)$

What's the cost of finding A^{-1}?

Same as solving

$$AX = I,$$

so still $O(n^3)$.

Cost: Worrying about the Constant, BLAS

$O(n^3)$ really means

$$\alpha \cdot n^3 + \beta \cdot n^2 + \gamma \cdot n + \delta.$$

All the non-leading and constants terms swept under the rug. But: at least the leading constant ultimately matters.

Shrinking the constant: surprisingly hard (even for 'just' matmul)

Idea: Rely on library implementation: *BLAS* (Fortran)

Level 1	$z = \alpha x + y$	vector-vector operations $O(n)$ `?axpy`
Level 2	$z = Ax + y$	matrix-vector operations $O(n^2)$ `?gemv`
Level 3	$C = AB + \beta C$	matrix-matrix operations $O(n^3)$ `?gemm`, `?trsm`

Show (using perf): `numpy` matmul calls BLAS `dgemm`

LAPACK

LAPACK: Implements ‘higher-end’ things (such as LU) using BLAS
Special matrix formats can also help save const significantly, e.g.

- banded
- sparse
- symmetric
- triangular

Sample routine names:

- `dgesvd`, `zgesdd`
- `dgetrf`, `dgetrs`

LU on Blocks: The Schur Complement

Given a matrix

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix},$$

can we do 'block LU' to get a *block triangular matrix*?

Multiply the top row by $-CA^{-1}$, add to second row, gives:

$$\begin{bmatrix} A & B \\ 0 & D - CA^{-1}B \end{bmatrix}.$$

$D - CA^{-1}B$ is called the Schur complement. Block pivoting is also possible if needed.

LU: Special cases

What happens if we feed a non-invertible matrix to LU?

$$PA = LU$$

(invertible, not invertible) (Why?)

What happens if we feed LU an $m \times n$ non-square matrices?

Think carefully about sizes of factors and columns/rows that do/don't matter. Two cases:

- $m > n$ (tall&skinny): $L : m \times n$, $U : n \times n$
- $m < n$ (short&fat): $L : m \times m$, $U : m \times n$

This is called reduced LU factorization.

Round-off Error in LU

Consider factorization of $\begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix}$ where $\epsilon < \epsilon_{\text{mach}}$:

- Without pivoting: $L = \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix}$, $U = \begin{bmatrix} \epsilon & 1 \\ 0 & 1 - 1/\epsilon \end{bmatrix}$
- Rounding: $\text{fl}(U)) = \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix}$
- This leads to $L\,\text{fl}(U)) = \begin{bmatrix} \epsilon & 1 \\ 1 & 0 \end{bmatrix}$, a backward error of $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$

Permuting the rows of A in partial pivoting gives $PA = \begin{bmatrix} 1 & 1 \\ \epsilon & 1 \end{bmatrix}$

- We now compute $L = \begin{bmatrix} 1 & 0 \\ \epsilon & 1 \end{bmatrix}$, $U = \begin{bmatrix} 1 & 1 \\ 0 & 1 - \epsilon \end{bmatrix}$, so $\text{fl}(U) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$
- This leads to $L\,\text{fl}(U) = \begin{bmatrix} 1 & 1 \\ \epsilon & 1 + \epsilon \end{bmatrix}$, a backward error of $\begin{bmatrix} 0 & 0 \\ 0 & \epsilon \end{bmatrix}$.

Changing matrices

Seen: LU cheap to re-solve if RHS changes. (Able to keep the expensive bit, the LU factorization) What if the *matrix* changes?

Special cases allow something to be done (a so-called *rank-one update*):

$$\hat{A} = A + \mathbf{u}\mathbf{v}^T$$

The Sherman-Morrison formula gives us

$$(A + \mathbf{u}\mathbf{v}^T)^{-1} = A^{-1} - \frac{A^{-1}\mathbf{u}\mathbf{v}^T A^{-1}}{1 + \mathbf{v}^T A^{-1}\mathbf{u}}.$$

Proof: Multiply the above by $\hat{A}$ get the identity.
FYI: There is a rank-k analog called the Sherman-Morrison-Woodbury formula.

Demo: Sherman-Morrison

In-Class Activity: LU

In-class activity: LU and Cost

Outline

Introduction to Scientific Computing

Systems of Linear Equations

Linear Least Squares
- Introduction
- Sensitivity and Conditioning
- Solving 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

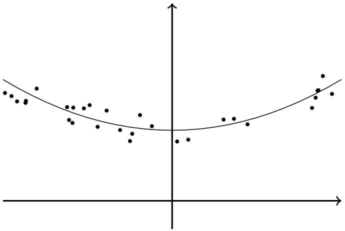
What about non-square systems?

Specifically, what about linear systems with ‘tall and skinny’ matrices? (A: $m \times n$ with $m > n$) (aka *overdetermined* linear systems)

Specifically, any hope that we will solve those exactly?

Not really: more equations than unknowns.

Example: Data Fitting



Have data: (x_i, y_i) and model:

$$y(x) = \alpha + \beta x + \gamma x^2$$

Find data that (best) fit model!

Data Fitting Continued

$$\begin{aligned} \alpha + \beta x_1 + \gamma x_1^2 &= y_1 \\ &\vdots \\ \alpha + \beta x_n + \gamma x_n^2 &= y_n \end{aligned}$$

Not going to happen for $n > 3$. Instead:

$$\begin{aligned} &\left|\alpha + \beta x_1 + \gamma x_1^2 - y_1\right|^2 \\ &\qquad + \cdots + \\ &\left|\alpha + \beta x_n + \gamma x_n^2 - y_n\right|^2 \quad \to \quad \text{min!} \end{aligned}$$

$\to$ *Least Squares*
This is called *linear least squares* specifically because the coefficients x enter linearly into the residual.

Rewriting Data Fitting

Rewrite in matrix form.

$$\|A\mathrm{x} - \mathrm{b}\|_2^2 \to \min!$$

with

$$A = \begin{bmatrix} 1 & x_1 & x_1^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix}, \quad \mathrm{x} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix}, \quad \mathrm{b} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

- Matrices like A are called Vandermonde matrices.
- Easy to generalize to higher polynomial degrees.

Least Squares: The Problem In Matrix Form

$$\|Ax - b\|_2^2 \to \min!$$

is cumbersome to write.
Invent new notation, defined to be equivalent:

$$Ax \cong b$$

NOTE:

- Data Fitting is *one example* where LSQ problems arise.
- Many other application lead to $Ax \cong b$, with different matrices.

Data Fitting: Nonlinearity

Give an example of a nonlinear data fitting problem.

$$\begin{aligned} &\left|\exp(\alpha) + \beta x_1 + \gamma x_1^2 - y_1\right|^2 \\ &\qquad + \cdots + \\ &\left|\exp(\alpha) + \beta x_n + \gamma x_n^2 - y_n\right|^2 \quad \to \quad \text{min!} \end{aligned}$$

But that would be easy to remedy: Do linear least squares with $\exp(\alpha)$ as the unknown. More difficult:

$$\begin{aligned} &\left|\alpha + \exp(\beta x_1 + \gamma x_1^2) - y_1\right|^2 \\ &\qquad + \cdots + \\ &\left|\alpha + \exp(\beta x_n + \gamma x_n^2) - y_n\right|^2 \quad \to \quad \text{min!} \end{aligned}$$

Demo: Interactive Polynomial Fit

Properties of Least-Squares

Consider LSQ problem $A\mathrm{x} \cong \mathrm{b}$ and its associated *objective function* $\varphi(x) = \|\mathrm{b} - A\mathrm{x}\|_2^2$. Does this always have a solution?

Yes. $\varphi \geqslant 0$, $\varphi \to \infty$ as $\|\mathrm{x}\| \to \infty$, φ continuous $\Rightarrow$ has a minimum.

Is it always unique?

No, for example if A has a nullspace.

Examine the objective function, find its minimum.

$$\begin{aligned} \varphi(\mathrm{x}) &= (\mathrm{b} - A\mathrm{x})^T(\mathrm{b} - A\mathrm{x}) \\ & \mathrm{b}^T\mathrm{b} - 2\mathrm{x}^T A^T \mathrm{b} + \mathrm{x}^T A^T A \mathrm{x} \\ \nabla\varphi(\mathrm{x}) &= -2A^T\mathrm{b} + 2A^T A\mathrm{x} \end{aligned}$$

$\nabla\varphi(\mathrm{x}) = 0$ yields $A^T A\mathrm{x} = A^T\mathrm{b}$. Called the *normal equations*.

Least squares: Demos

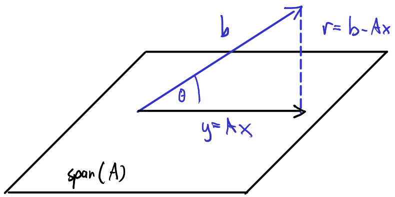
Demo: Polynomial fitting with the normal equations

What's the shape of $A^T A$?

Always square.

Demo: Issues with the normal equations

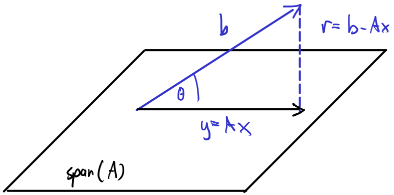
Least Squares, Viewed Geometrically



Why is r $\perp$ span(A) a good thing to require?

Because then the distance between y $=$ Ax and b is minimal.
Q: Why?
Because of Pythagoras's theorem–another y would mean additional distance traveled in span(A).

Least Squares, Viewed Geometrically (II)



Phrase the Pythagoras observation as an equation.

$$\begin{aligned} \operatorname{span}(A) &\perp b - Ax \\ A^T b - A^T A x &= 0 \end{aligned}$$

Congratulations: Just rediscovered the normal equations.

Write that with an orthogonal projection matrix P.

$Ax = Pb$.

About Orthogonal Projectors

What is a *projector*?

A matrix satisfying

$$P^2 = P.$$

What is an *orthogonal projector*?

A symmetric projector.

How do I make one projecting onto $\operatorname{span}\{q_1, q_2, \ldots, q_\ell\}$ for orthogonal q_i?

First define

$$Q = \begin{bmatrix} q_1 & q_2 & \cdots & q_\ell \end{bmatrix}.$$

Then

$$QQ^T$$

will project and is obviously symmetric.

Least Squares and Orthogonal Projection

Check that $P = A(A^TA)^{-1}A^T$ is an orthogonal projector onto colspan(A).

$$P^2 = A(A^TA)^{-1}A^TA(A^TA)^{-1}A^T = A(A^TA)^{-1}A^T = P.$$

Symmetry: also yes.

Onto colspan(A): Last matrix is $A \to$ result of $P\mathbf{x}$ must be in colspan(A).

Conclusion: P is the projector from the previous slide!

What assumptions do we need to define the P from the last question?

A^TA has full rank (i.e. is invertible).

Pseudoinverse

What is the pseudoinverse of A?

Nonsquare $m \times n$ matrix A has no inverse in usual sense.
If $\operatorname{rank}(A) = n$, pseudoinverse is $A^{+} = (A^T A)^{-1} A^T$. (colspan-projector with final A missing)

What can we say about the condition number in the case of a tall-and-skinny, full-rank matrix?

$$\operatorname{cond}_2(A) = \|A\|_2 \left\|A^{+}\right\|_2$$

If not full rank, $\operatorname{cond}(A) = \infty$ by convention.

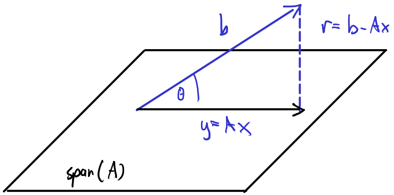
What does all this have to do with solving least squares problems?

$\mathbf{x} = A^{+}\mathbf{b}$ solves $A\mathbf{x} \cong \mathbf{b}$.

In-Class Activity: Least Squares

In-class activity: Least Squares

Sensitivity and Conditioning of Least Squares



Define

$$\cos(\theta) = \frac{\|Ax\|_2}{\|b\|_2},$$

then

$$\frac{\|\Delta x\|_2}{\|x\|_2} \leqslant \operatorname{cond}(A)\frac{1}{\cos(\theta)} \cdot \frac{\|\Delta b\|_2}{\|b\|_2}.$$

What values of θ are bad?

$b \perp \operatorname{colspan}(A)$, i.e. $\theta \approx \pi/2$.

Sensitivity and Conditioning of Least Squares (II)

Any comments regarding dependencies?

Unlike for $A\mathrm{x} = \mathrm{b}$, the sensitivity of least squares solution depends on both A and b.

What about changes in the matrix?

$$\frac{\|\Delta \mathrm{x}\|_2}{\|\mathrm{x}\|_2} \leqslant [\mathrm{cond}(A)^2 \tan(\theta) + \mathrm{cond}(A)] \cdot \frac{\|\Delta A\|_2}{\|A\|_2}.$$

Two behaviors:

- If $\tan(\theta) \approx 0$, condition number is $\mathrm{cond}(A)$.
- Otherwise, $\mathrm{cond}(A)^2$.

Recap: Orthogonal Matrices

What's an *orthogonal (=orthonormal) matrix*?

One that satisfies $Q^T Q = I$ and $QQ^T = I$.

How do orthogonal matrices interact with the 2-norm?

$$\|Q\mathrm{v}\|_2^2 = (Q\mathrm{v})^T(Q\mathrm{v}) = \mathrm{v}^T Q^T Q\mathrm{v} = \mathrm{v}^T\mathrm{v} = \|\mathrm{v}\|_2^2 .$$

Transforming Least Squares to Upper Triangular

Suppose we have $A = QR$, with Q square and orthogonal, and R upper triangular. This is called a QR factorization.
How do we transform the least squares problem $A\mathrm{x} \cong \mathrm{b}$ to one with an upper triangular matrix?

$$\begin{aligned} & \|A\mathrm{x} - \mathrm{b}\|_2 \\ = & \left\| Q^T (QR\mathrm{x} - \mathrm{b}) \right\|_2 \\ = & \left\| R\mathrm{x} - Q^T\mathrm{b} \right\|_2 \end{aligned}$$

Simpler Problems: Triangular

What do we win from transforming a least-squares system to upper triangular form?

$$\begin{bmatrix} R_{\text{top}} \end{bmatrix} \mathbf{x} \cong \begin{bmatrix} (Q^T\mathbf{b})_{\text{top}} \\ (Q^T\mathbf{b})_{\text{bottom}} \end{bmatrix}$$

How would we minimize the residual norm?

For the residual vector r, we find

$$\|\mathbf{r}\|_2^2 = \left\|(Q^T\mathbf{b})_{\text{top}} - R_{\text{top}}\mathbf{x}\right\|_2^2 + \left\|(Q^T\mathbf{b})_{\text{bottom}}\right\|_2^2 .$$

R is invertible, so we can find x to zero out the first term, leaving

$$\|\mathbf{r}\|_2^2 = \left\|(Q^T\mathbf{b})_{\text{bottom}}\right\|_2^2 .$$

Computing QR

- Gram-Schmidt
- Householder Reflectors
- Givens Rotations

Demo: Gram-Schmidt–The Movie
Demo: Gram-Schmidt and Modified Gram-Schmidt
Demo: Keeping track of coefficients in Gram-Schmidt
Seen: Even modified Gram-Schmidt still unsatisfactory in finite precision arithmetic because of roundoff.

NOTE: Textbook makes further modification to 'modified' Gram-Schmidt:

- Orthogonalize *subsequent* rather than *preceding* vectors.
- Numerically: no difference, but sometimes algorithmically helpful.

Economical/Reduced QR

Is QR with square Q for $A \in \mathbb{R}^{m \times n}$ with $m > n$ efficient?

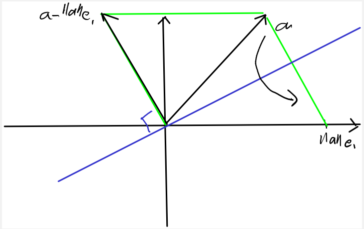
No. Can obtain economical or reduced QR with $Q \in \mathbb{R}^{m \times n}$ and $R \in \mathbb{R}^{n \times n}$. Least squares solution process works unmodified with the economical form, though the equivalence proof relies on the 'full' form.

In-Class Activity: QR

In-class activity: QR

Householder Transformations

Find an *orthogonal* matrix Q to zero out the lower part of a vector a.



Orthogonality in figure: $(\mathrm{a} - \|\mathrm{a}\|_2 \mathrm{e}_1) \cdot (\mathrm{a} + \|\mathrm{a}\|_2 \mathrm{e}_1) = \|\mathrm{a}\|_2^2 - \|\mathrm{a}\|_2^2$.

Let's call $\mathrm{v} = \mathrm{a} - \|\mathrm{a}\|_2 \mathrm{e}_1$. How do we reflect about the plane orthogonal to v? Project-and-keep-going:

$$H := I - 2\frac{\mathrm{v}\mathrm{v}^T}{\mathrm{v}^T\mathrm{v}}.$$

This is called a Householder reflector.

Householder Reflectors: Properties

Seen from picture (and easy to see with algebra):

$$H\mathrm{a} = \pm \|\mathrm{a}\|_2 \mathrm{e}_1.$$

Remarks:

- Q: What if we want to zero out only the $i+1$th through nth entry?
  A: Use e_i above.
- A product $H_n \cdots H_1 A = R$ of Householders makes it easy (and quite efficient!) to build a QR factorization.
- It turns out $\mathrm{v}' = \mathrm{a} + \|\mathrm{a}\|_2 \mathrm{e}_1$ works out, too–just pick whichever one causes less cancellation.
- H is symmetric
- H is orthogonal

Demo: 3x3 Householder demo

Givens Rotations

If reflections work, can we make rotations work, too?

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sqrt{a_1^2 + a_2^2} \\ 0 \end{bmatrix}.$$

Not hard to sovle for c and s.

Downside? Produces only one zero at a time.

Demo: 3x3 Givens demo

Rank-Deficient Matrices and QR

What happens with QR for rank-deficient matrices?

$$A = Q \begin{bmatrix} * & * & * \\ & \text{(small)} & * \\ & & * \end{bmatrix}$$

(where $*$ represents a generic non-zero)

Practically, it makes sense to ask for all these 'small' columns to be gathered near the 'right' of R $\rightarrow$ Column pivoting.

Q: What does the resulting factorization look like?

$$AP = QR$$

Also used as the basis for *rank-revealing QR*.

Rank-Deficient Matrices and Least-Squares

What happens with Least Squares for rank-deficient matrices?

$$A\mathbf{x} \cong \mathbf{b}$$

- QR still finds a solution with minimal residual
- By QR it's easy to see that least squares with a short-and-fat matrix is equivalent to a rank-deficient one.
- But: No longer unique. $\mathbf{x} + \mathbf{n}$ for $\mathbf{n} \in N(A)$ has the same residual.
- In other words: Have more freedom
  Or: Can demand another condition, for example:
  - Minimize $\|\mathbf{b} - A\mathbf{x}\|_2^2$, *and*
  - minimize $\|\mathbf{x}\|_2^2$, simultaneously.
    Unfortunately, QR does not help much with that $\to$ Need better tool.

Singular Value Decomposition (SVD)

What is the *Singular Value Decomposition* of an $m \times n$ matrix?

$$A = U\Sigma V^T,$$

with

- U is $m \times m$ and orthogonal
  Columns called the *left singular vectors*.
- $\Sigma = \operatorname{diag}(\sigma_i)$ is $m \times n$ and non-negative
  Typically $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_n \geqslant 0$.
  Called the *singular values*.
- V is $n \times n$ and orthogonal
  Columns called the *right singular vectors*.

Existence: Not yet, later.

SVD: What's this thing good for? (I)

- $\|A\|_2 = \sigma_1$
- $\mathrm{cond}_2(A) = \sigma_1/\sigma_n$
- Nullspace $N(A) = \mathrm{span}(\{\mathrm{v}_i : \sigma_i = 0\})$.
- $\mathrm{rank}(A) = \#\{i : \sigma_i \neq 0\}$

  Computing rank in the presence of round-off error is not laughably non-robust. More robust:
- *Numerical rank*:

$$\mathrm{rank}_\varepsilon = \#\{i : \sigma_i > \varepsilon\}$$

SVD: What's this thing good for? (II)

- *Low-rank Approximation*

Theorem (Eckart-Young-Mirsky)

If $k < r = \operatorname{rank}(A)$ *and*

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T,$$

then

$$\min_{\operatorname{rank}(B)=k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}.$$

Demo: Image compression

SVD: What's this thing good for? (III)

- The minimum norm solution to $A\mathrm{x} \cong \mathrm{b}$:

$$\begin{aligned} U\Sigma V^T\mathrm{x} &\cong \mathrm{b} \\ \Leftrightarrow \Sigma \underbrace{V^T\mathrm{x}}_{\mathrm{y}} &\cong U^T\mathrm{b} \\ \Leftrightarrow \Sigma\mathrm{y} &\cong U^T\mathrm{b} \end{aligned}$$

Then define

$$\Sigma^+ = \mathrm{diag}(\sigma_1^+, \ldots, \sigma_n^+),$$

where Σ^+ is $n \times m$ if A is $m \times n$, and

$$\sigma_i^+ = \begin{cases} 1/\sigma_i & \sigma_i \neq 0, \\ 0 & \sigma_i = 0. \end{cases}$$

SVD: Minimum-Norm, Pseudoinverse

$y = \Sigma^+ U^T b$ is the minimum norm-solution to $\Sigma y \cong U^T b$.
Observe $\|x\|_2 = \|y\|_2$.

$$x = V\Sigma^+ U^T b$$

solves the minimum-norm least-squares problem.

Define $A^+ = V\Sigma^+ U^T$ and call it the pseudoinverse of A.
Coincides with prior definition in case of full rank.

In-Class Activity: Householder, Givens, SVD

In-class activity: Householder, Givens, SVD

Comparing the Methods

Methods to solve least squares with A an $m \times n$ matrix:

- Form: $A^T A$: $n^2 m/2$
  Solve with $A^T A$: $n^3/6$
- Solve with Householder: $mn^2 - n^3/3$
- If $m \approx n$, about the same
- If $m \gg n$: Householder QR requires about twice as much work as normal equations
- SVD: $mn^2 + n^3$ (with a large constant)

Demo: Relative cost of matrix factorizations

Outline

Introduction to Scientific Computing

Systems of Linear Equations

Linear Least Squares

Eigenvalue Problems
- Properties and Transformations
- Sensitivity
- Computing Eigenvalues
- Krylov Space Methods

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

Eigenvalue Problems: Setup/Math Recap

A is an $n \times n$ matrix.

- $\mathrm{x} \neq 0$ is called an *eigenvector* of A if there exists a λ so that

$$A\mathrm{x} = \lambda \mathrm{x}.$$

- In that case, λ is called an *eigenvalue*.
- The set of all eigenvalues $\lambda(A)$ is called the *spectrum*.
- The *spectral radius* is the magnitude of the biggest eigenvalue:

$$\rho(A) = \max \{|\lambda| : \lambda(A)\}$$

Finding Eigenvalues

How do you find eigenvalues?

$$Ax = \lambda x \Leftrightarrow (A - \lambda I)x = 0$$
$$\Leftrightarrow A - \lambda I \text{ singular} \Leftrightarrow \det(A - \lambda I) = 0$$

$\det(A - \lambda I)$ is called the *characteristic polynomial*, which has degree n, and therefore n (potentially complex) roots.

Does that help algorithmically? Abel-Ruffini theorem: for $n \geqslant 5$ is no general formula for roots of polynomial. IOW: no.

- For LU and QR, we obtain *exact* answers (except rounding).
- For eigenvalue problems: not possible—must *approximate*.

Demo: Rounding in characteristic polynomial using SymPy

Multiplicity

What is the *multiplicity* of an eigenvalue?

Actually, there are two notions called multiplicity:

- *Algebraic Multiplicity*: multiplicity of the root of the characteristic polynomial
- *Geometric Multiplicity*: #of lin. indep. eigenvectors

In general: AM $\geqslant$ GM.
If AM $>$ GM, the matrix is called *defective*.

An Example

Give characteristic polynomial, eigenvalues, eigenvectors of

$$\begin{bmatrix} 1 & 1 \\ & 1 \end{bmatrix}.$$

CP: $(\lambda - 1)^2$
Eigenvalues: 1 (with multiplicity 2)
Eigenvectors:

$$\begin{bmatrix} 1 & 1 \\ & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

$\Rightarrow x + y = x \Rightarrow y = 0$. So only a 1D space of eigenvectors.

Diagonalizability

When is a matrix called *diagonalizable*?

If it is not defective, i.e. if it has a n linear independent eigenvectors (i.e. a full basis of them). Call those $(\mathrm{x}_n)_{i=1}^n$.

In that case, let

$$X = \begin{bmatrix} | & & | \\ \mathrm{x}_1 & \cdots & \mathrm{x}_n \\ | & & | \end{bmatrix},$$

and observe $AX = XD$ or

$$A = XDX^{-1},$$

where D is a diagonal matrix with the eigenvalues.

Similar Matrices

Related definition: Two matrices A and B are called similar if there exists an invertible matrix X so that $A = XBX^{-1}$.

In that sense: "Diagonalizable" = "Similar to a diagonal matrix".

Observe: Similar A and B have same eigenvalues. (Why?)

Suppose $A\mathrm{x} = \lambda\mathrm{x}$. We have $B = X^{-1}AX$. Let $\mathrm{w} = X^{-1}\mathrm{v}$. Then

$$B\mathrm{w} = X^{-1}A\mathrm{v} = \lambda\mathrm{w}.$$

Eigenvalue Transformations (I)

What do the following transformations of the eigenvalue problem $A\mathrm{x} = \lambda\mathrm{x}$ do?

Shift. $A \to A - \sigma I$

$$\boxed{(A - \sigma I)\mathrm{x} = (\lambda - \sigma)\mathrm{x}}$$

Inversion. $A \to A^{-1}$

$$\boxed{A^{-1}\mathrm{x} = \lambda^{-1}\mathrm{x}}$$

Power. $A \to A^k$

$$\boxed{A^k\mathrm{x} = \lambda^k\mathrm{x}}$$

Eigenvalue Transformations (II)

Polynomial $A \to aA^2 + bA + cI$

$$(aA^2 + bA + cI)\mathrm{x} = (a\lambda^2 + b\lambda + c)\mathrm{x}$$

Similarity $T^{-1}AT$ with T invertible

Let $\mathrm{y} := T^{-1}\mathrm{x}$. Then

$$T^{-1}AT\mathrm{y} = \lambda\mathrm{y}$$

Sensitivity (I)

Assume A not defective. Suppose $X^{-1}AX = D$. Perturb $A \to A + E$. What happens to the eigenvalues?

$$X^{-1}(A+E)X = D + F$$

- $A+E$ and $D+F$ have same eigenvalues
- $D+F$ is not necessarily diagonal

Suppose v is a perturbed eigenvector.

$$\begin{aligned}
& (D+F)\mathrm{v} = \mu \mathrm{v} \\
\Leftrightarrow\quad & F\mathrm{v} = (\mu I - D)\mathrm{v} \\
\Leftrightarrow\quad & (\mu I - D)^{-1} F\mathrm{v} = \mathrm{v} \qquad \text{(when is that invertible?)} \\
\Rightarrow\quad & \|\mathrm{v}\| \leqslant \left\|(\mu I - D)^{-1}\right\| \|F\| \, \|\mathrm{v}\| \\
\Rightarrow\quad & \left\|(\mu I - D)^{-1}\right\|^{-1} \leqslant \|F\|
\end{aligned}$$

Sensitivity (II)

$X^{-1}(A+E)X = D + F$. Have $\left\|(\mu I - D)^{-1}\right\|^{-1} \leqslant \|F\|$.

$$\left\|(\mu I - D)^{-1}\right\|^{-1} = |\mu - \lambda_k|$$

where λ_k is the closest eigenvalue of D to μ.

$$\left|\mu - \lambda_k\right| = \left\|(\mu I - D)^{-1}\right\|^{-1} \leqslant \|F\| = \left\|X^{-1}EX\right\| \leqslant \operatorname{cond}(X)\,\|E\|\,.$$

- This is 'bad' if X is ill-conditioned, i.e. if the eigenvectors are nearly linearly dependent.
- If X is orthogonal (e.g. for symmetric A), then eigenvalues are always well-conditioned.
- This bound is in terms of *all* eigenvalues, so may overestimate for each individual eigenvalue.

Demo: Bauer-Fike Eigenvalue Sensitivity Bound

Power Iteration

What are the eigenvalues of A^{1000}?

Assume $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$ with eigenvectors $\mathrm{x}_1, \ldots, \mathrm{x}_n$. Further assume $\|\mathrm{x}_i\| = 1$.

Use $\mathrm{x} = \alpha \mathrm{x}_1 + \beta \mathrm{x}_2$.

$$\mathrm{y} = A^{1000}(\alpha \mathrm{x}_1 + \beta \mathrm{x}_2) = \alpha \lambda_1^{1000} \mathrm{x}_1 + \beta \lambda_2^{1000} \mathrm{x}_2$$

Or

$$\frac{\mathrm{y}}{\lambda_1^{1000}} = \alpha \mathrm{x}_1 + \beta \underbrace{\left(\underbrace{\frac{\lambda_2}{\lambda_1}}_{<1} \right)^{1000}}_{\ll 1} \mathrm{x}_2.$$

Idea: Use this as a computational procedure to find x_1.
Called Power Iteration.

Power Iteration: Issues?

What could go wrong with Power Iteration?

- Starting vector has no component along x_1
  Not a problem in practice: Rounding will introduce one.
- Overflow in computing λ_1^{1000}
  $\to$ *Normalized Power Iteration*
- $|\lambda_1| = |\lambda_2|$
  Real problem.

What about Eigenvalues?

Power Iteration generates eigenvectors. What if we would like to know eigenvalues?

Estimate them:

$$\frac{x^T A x}{x^T x}$$

- $= \lambda$ if x is an eigenvector w/ eigenvalue λ
- Otherwise, an estimate of a ‘nearby’ eigenvalue

This is called the *Rayleigh quotient*.

Convergence of Power Iteration

What can you say about the convergence of the power method?
Say $v_1^{(k)}$ is the kth estimate of the eigenvector x_1, and

$$e_k = \left\| x_1 - v_1^{(k)} \right\|.$$

Easy to see:

$$e_{k+1} \approx \frac{|\lambda_2|}{|\lambda_1|} e_k.$$

We will later learn that this is *linear convergence*. It's quite slow.
What does a shift do to this situation?

$$e_{k+1} \approx \frac{|\lambda_2 - \sigma|}{|\lambda_1 - \sigma|} e_k.$$

Picking $\sigma \approx \lambda_1$ does not help...
Idea: Invert *and* shift to bring $|\lambda_1 - \sigma|$ into numerator.

Rayleigh Quotient Iteration

Describe *inverse iteration*.

$$x_{k+1} := (A - \sigma)^{-1} x_k$$

- Implemented by storing/solving with LU factorization
- Converges to eigenvector for eigenvalue closest to σ, with

$$e_{k+1} \approx \frac{|\lambda_{\text{closest}} - \sigma|}{|\lambda_{\text{second-closest}} - \sigma|} e_k.$$

Describe *Rayleigh Quotient Iteration*.

Compute $\sigma_k = x_k^T A x_k / x_k^T x_k$ to be the Rayleigh quotient for x_k.

$$x_{k+1} := (A - \sigma_k I)^{-1} x_k$$

In-Class Activity: Eigenvalues

In-class activity: Eigenvalues

Schur form

Show: Every matrix is orthonormally similar to an upper triangular matrix, i.e. $A = QUQ^T$. This is called the Schur form or Schur factorization.

Assume A non-defective for now. Suppose $A\mathrm{v} = \lambda \mathrm{v}$ ($\mathrm{v} \neq 0$). Let $V = \mathrm{span}\{\mathrm{v}\}$. Then

$$\begin{array}{rccc} A: & V & \to & V \\ & V^\perp & \to & V \oplus V^\perp \end{array}$$

$$A = \underbrace{\begin{bmatrix} z & \\ \mathrm{v} & \text{Basis of } V^\perp \\ z & \end{bmatrix}}_{Q_1} \begin{bmatrix} \lambda & * & * & * & * \\ 0 & * & * & * & * \\ \vdots & * & * & * & * \\ 0 & * & * & * & * \end{bmatrix} Q_1^T.$$

Repeat n times *to* triangular: $Q_n \cdots Q_1 U Q_1^T \cdots Q_n^T$.

Schur Form: Comments, Eigenvalues, Eigenvectors

$A = QUQ^T$. For complex λ:

- Either complex matrices, or
- 2×2 blocks on diag.

If we had a Schur form of A, how can we find the eigenvalues?

The eigenvalues (of U and A!) are on the diagonal of U.

And the eigenvectors?

It suffices to find eigenvectors of U: can convert to eigenvectors of A by similarity transform. Suppose λ is an eigenvalue.

$$U - \lambda I = \begin{bmatrix} U_{11} & \mathrm{u} & U_{13} \\ 0 & 0 & \mathrm{v}^T \\ 0 & 0 & U_{31} \end{bmatrix}$$

Then $[U_{11}^{-1}\mathrm{u}; -1; 0]^T$ is an eigenvector.

Computing Multiple Eigenvalues

All Power Iteration Methods compute one eigenvalue at a time.
What if I want *all* eigenvalues?

Two ideas:

1. *Deflation*: similarity transform to
$$\begin{bmatrix} \lambda_1 & * \\ & B \end{bmatrix},$$
i.e. one step in Schur form. Now find eigenvalues of B.
2. Iterate with multiple vectors simultaneously.

Simultaneous Iteration

What happens if we carry out power iteration on multiple vectors simultaneously?

Simultaneous Iteration:

1. Start with $X_0 \in \mathbb{R}^{n \times p}$ ($p \leqslant n$) with (arbitrary) iteration vectors in columns
2. $X_{k+1} = AX_k$

Problems:

- Needs rescaling
- X increasingly ill-conditioned: all columns go towards x_1

Fix: orthogonalize!

Orthogonal Iteration

Orthogonal Iteration:

1. Start with $X_0 \in \mathbb{R}^{n\times p}$ $(p \leqslant n)$ with (arbitrary) iteration vectors in columns
2. $Q_k R_k = X_k$ (reduced)
3. $X_{k+1} = AQ_k$

Good: X_k converge to X with eigenvectors in columns

Bad:

- Slow/linear convergence
- Expensive iteration

Toward the QR Algorithm

$$\begin{aligned}
Q_0 R_0 &= X_0 \\
X_1 &= A Q_0 \\
Q_1 R_1 &= X_1 &&= A Q_0 &&\Rightarrow && Q_1 R_1 Q_0^T = A \\
X_2 &= A Q_1 \\
Q_2 R_2 &= X_2 &&= A Q_1 &&\Rightarrow && Q_2 R_2 Q_1^T = A
\end{aligned}$$

Once the Q_k converge ($Q_{n+1} \approx Q_n$), we have a Schur factorization!

Problem: $Q_{n+1} \approx Q_n$ works poorly as a convergence test.

Observation 1: Once $Q_{n+1} \approx Q_n$, we also have $Q_2 R_2 Q_2^T \approx A$.
Observation 2: $\hat{X}_2 := Q_2^T A Q_2 \approx R_2$.

Idea: Use magnitude of below-diag part of $\hat{X}_2$ for convergence check.
Q: Can we compute $\hat{X}_k$ directly?

Demo: Orthogonal Iteration

QR Iteration/QR Algorithm

Orthogonal iteration:	QR iteration:
$X_0 = A$	$\bar{X}_0 = A$
$Q_k R_k = X_k$	$\bar{Q}_k \bar{R}_k = \bar{X}_k$
$X_{k+1} = A Q_k$	$\bar{X}_{k+1} = \bar{R}_k \bar{Q}_k$

Tracing through reveals:

- $\hat{X}_k = \bar{X}_{k+1}$
- $Q_0 = \bar{Q}_0$
  $Q_1 = \bar{Q}_0 \bar{Q}_1$
  $Q_k = \bar{Q}_0 \bar{Q}_1 \cdots \bar{Q}_k$

Orthogonal iteration showed: $\hat{X}_k = \bar{X}_{k+1}$ converge. Also:

$$\bar{X}_{k+1} = \bar{R}_k \bar{Q}_k = \bar{Q}_k^T \bar{X}_k \bar{Q}_k,$$

so the $\bar{X}_k$ are all similar $\rightarrow$ all have the same eigenvalues.
$\rightarrow$ QR iteration produces Schur form.

QR Iteration: Incorporating a Shift

How can we accelerate convergence of QR iteration using shifts?

$$
\begin{aligned}
\bar{X}_0 &= A\\
\bar{Q}_k\bar{R}_k &= \bar{X}_k - \sigma_k I\\
\bar{X}_{k+1} &= \bar{R}_k\bar{Q}_k + \sigma_k I
\end{aligned}
$$

Still a similarity transform:

$$\bar{X}_{k+1} = \bar{R}_k\bar{Q}_k + \sigma_k I = [\overline{Q}_k^T\bar{X}_k - \bar{Q}_k^T\sigma_k]\bar{Q}_k + \sigma_k I$$

Q: How should the shifts be chosen?

- Ideally: Close to existing eigenvalue
- Heuristics:
  - Lower right entry of $\bar{X}_k$
  - Eigenvalues of lower right 2×2 of $\bar{X}_k$

QR Iteration: Computational Expense

A full QR factorization at each iteration costs $O(n^3)$–can we make that cheaper?

Idea: *Hessenberg form*

$$A = Q \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ & * & * & * \\ & & * & * \end{bmatrix} Q^T$$

- Attainable by *similarity transforms* (!) HAH^T with Householders that start 1 entry lower than ‘usual’
- QR factorization of Hessenberg matrices can be achieved in $O(n^2)$ time using Givens rotations.

Demo: Householder Similarity Transforms

QR/Hessenberg: Overall procedure

Overall procedure:

1. Reduce matrix to Hessenberg form
2. Apply QR iteration using Givens QR to obtain Schur form

For symmetric matrices:

- Use Householders to attain tridiagonal form
- Use QR iteration with Givens to attain diagonal form

Krylov space methods: Intro

What subspaces can we use to look for eigenvectors?

QR:

$$\operatorname{span}\{A^\ell \mathrm{y}_1, A^\ell \mathrm{y}_2, \dots, A^\ell \mathrm{y}_k\}$$

Krylov space:

$$\operatorname{span}\{\underbrace{\mathrm{x}}_{\mathrm{x}_0}, A\mathrm{x}, \dots, \underbrace{A^{k-1}\mathrm{x}}_{\mathrm{x}_{k-1}}\}$$

Define:

$$K_k := \begin{bmatrix} | & & | \\ \mathrm{x}_0 & \cdots & \mathrm{x}_{k-1} \\ | & & | \end{bmatrix}. \qquad (n \times k)$$

Krylov for Matrix Factorization

What matrix factorization is obtained through Krylov space methods?

$$AK_n = \begin{bmatrix} | & & | \\ \mathrm{x}_1 & \cdots & \mathrm{x}_n \\ | & & | \end{bmatrix} = K_n \underbrace{\begin{bmatrix} | & & | & | \\ \mathrm{e}_2 & \cdots & \mathrm{e}_n & K_n^{-1}\mathrm{x}_n \\ | & & | & | \end{bmatrix}}_{C_n}.$$

- $K_n^{-1}AK_n = C_n$
- C_n is upper Hessenberg
- So Krylov is ‘just’ another way to get a matrix into upper Hessenberg form. But: built incrementally!

Conditioning in Krylov Space Methods/Arnoldi Iteration (I)

What is a problem with Krylov space methods? How can we fix it?

(x_i) converge rapidly to eigenvector for largest eigenvalue
$\rightarrow K_k$ become ill-conditioned

Idea: Orthogonalize! (at end... for now)

$$Q_n R_n = K_n \quad \Rightarrow \quad Q_n = K_n R_n^{-1}$$

Then

$$Q_n^T A Q_n = R_n \underbrace{K_n^{-1} A K_n}_{C_n} R_n^{-1}.$$

Conditioning in Krylov Space Methods/Arnoldi Iteration (II)

- C_n is upper Hessenberg
- (Left/right) mul. by triangulars preserves upper Hessenberg

We find that $Q_n^T A Q_n$ is also upper Hessenberg: $Q_n^T A_n Q_n = H$.
Also readable as $AQ_n = Q_n H$, which, read column-by-column, is:

$$A\mathrm{q}_k = h_{1k}\mathrm{q}_1 + \dots h_{k+1,k}\mathrm{q}_{k+1}$$

We find: $h_{jk} = \mathrm{q}_j^T A \mathrm{q}_k$.

Important consequence: Can compute

- q_{k+1} from $\mathrm{q}_1, \dots, \mathrm{q}_k$
- $(k+1)$st column of H

analogously to Gram-Schmidt QR!

This is called Arnoldi iteration. For symmetric: Lanczos iteration.

Krylov: What about eigenvalues?

How can we use Arnoldi/Lanczos to compute eigenvalues?

$$Q = \begin{bmatrix} Q_k & U_k \end{bmatrix}$$

Green: known (i.e. already computed), red: not yet computed.

$$H = Q^T A Q = \begin{bmatrix} Q_k \\ U_k \end{bmatrix} A \begin{bmatrix} Q_k & U_k \end{bmatrix} = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix}$$

Use eigenvalues of top-left matrix as approximate eigenvalues.
(still need to be computed, using QR it.)

Those are called Ritz values.

Demo: Arnoldi Iteration (Part 2)

Computing the SVD (Kiddy Version)

How can I compute an SVD of a matrix A?

1. Compute the eigenvalues and eigenvectors of $A^T A$.

$$A^T A \mathrm{v}_1 = \lambda_1 \mathrm{v}_1 \quad \cdots \qquad A^T A \mathrm{v}_n = \lambda_n \mathrm{v}_n$$

2. Matrix V from the vectors v_i. ($A^T A$ symm.: V orth.)
3. Make a diagonal matrix Σ from the square roots of the eigenvalues: $\Sigma = \operatorname{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n})$
4. Find U from $A = U\Sigma V^T \quad \Leftrightarrow \quad U\Sigma = AV$.
   For square matrices: $U = AV\Sigma^{-1}$.
   Observe U is orthogonal: (Use: $V^T A^T A V = \Sigma^2$)

$$U^T U = \Sigma^{-1} \underbrace{V^T A^T A V}_{\Sigma^2} \Sigma^{-1} = \Sigma^{-1}\Sigma^2\Sigma^{-1} = I.$$

Demo: Computing the SVD

Outline

Introduction to Scientific Computing

Systems of Linear Equations

Linear Least Squares

Eigenvalue Problems

Nonlinear Equations
- Introduction
- Iterative Procedures
- Methods in One Dimension
- Methods in n Dimensions (“Systems of 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

Solving Nonlinear Equations

What is the goal here?

Solve $\mathbf{f}(\mathbf{x}) = 0$ for $f : \mathbb{R}^n \to \mathbb{R}^n$.

If looking for solution to $\widetilde{\mathbf{f}}(\mathbf{x}) = \mathbf{y}$, simply consider $\mathbf{f}(\mathbf{x}) = \tilde{f}(\mathbf{x}) - \mathbf{y}$.

Intuition: Each of the n equations describes a surface. Looking for intersections.

Demo: Three quadratic functions

Showing Existence

How can we show existence of a root?

- Intermediate value theorem (uses continuity, 1D only)
- Inverse function theorem (relies on invertible Jacobian J_{f})
  Get *local* invertibility, i.e. $f(\mathrm{x}) = \mathrm{y}$ solvable
- Contraction mapping theorem
  A function $\mathrm{g} : \mathbb{R}^n \to \mathbb{R}^n$ is called *contractive* if there exists a $0 < \gamma < 1$ so that $\|\mathrm{g}(\mathrm{x}) - \mathrm{g}(\mathrm{y})\| \leqslant \gamma \|\mathrm{x} - \mathrm{y}\|$. A fixed point of g is a point where $\mathrm{g}(\mathrm{x}) = \mathrm{x}$.

  Then: On a closed set $S \subseteq \mathbb{R}^n$ with $\mathrm{g}(S) \subseteq S$ there exists a unique fixed point.
  Example: (real-world) map

In general, *no uniquness* results available.

Sensitivity and Multiplicity

What is the sensitivity/conditioning of root finding?

$$\operatorname{cond}(\text{root finding}) = \operatorname{cond}(\text{evaluation of the inverse function})$$

What are multiple roots?

$$\begin{aligned} f(x) &= 0 \\ f'(x) &= 0 \\ &\vdots \\ f^{(m-1)}(x) &= 0 \end{aligned}$$

This would be a root of multiplicity m.

How do multiple roots interact with conditioning?

In-Class Activity: Krylov and Nonlinear Equations

In-class activity: Krylov and Nonlinear Equations

Rates of Convergence

What is *linear convergence*? *quadratic convergence*?

Let $\mathrm{e}_k = \widehat{\mathrm{u}}_k - \mathrm{u}$ be the error in the kth iterate $\widehat{\mathrm{u}}_k$. Assume $\mathrm{e}_k \to 0$ as $k \to \infty$.

An iterative method *converges with rate r* if

$$\lim_{k\to\infty} \frac{\|\mathrm{e}_{k+1}\|}{\|\mathrm{e}_k\|^r} = C \begin{cases} > 0, \\ < \infty. \end{cases}$$

- $r = 1$ is called *linear convergence*.
- $r > 1$ is called *superlinear convergence*.
- $r = 2$ is called *quadratic convergence*.

Examples:

- Power iteration is linearly convergent.
- Rayleigh quotient iteration is quadratically convergent

About Convergence Rates

Demo: Rates of Convergence

Characterize linear, quadratic convergence in terms of the 'number of accurate digits'.

- Linear convergence gains a constant number of digits each step:

  $$\|\mathrm{e}_{k+1}\| \leqslant C \|\mathrm{e}_k\|$$

  (and $C < 1$ matters!)
- Quadratic convergence

  $$\|\mathrm{e}_{k+1}\| \leqslant C \|\mathrm{e}_k\|^2$$

  (Only starts making sense once $\|\mathrm{e}_k\|$ is small. C doesn't matter much.)

Stopping Criteria

Comment on the 'foolproof-ness' of these stopping criteria:

1. $|f(x)| < \varepsilon$ ('residual is small')
2. $\|x_{k+1} - x_k\| < \varepsilon$
3. $\|x_{k+1} - x_k\| / \|x_k\| < \varepsilon$

1. Can trigger far away from a root in the case of multiple roots (or a 'flat' f)
2. Allows different 'relative accuracy' in the root depending on its magnitude.
3. Enforces a relative accuracy in the root, but *does not* actually check that the function value is small.
   So if convergence 'stalls' away from a root, this may trigger without being anywhere near the desired solution.

Lesson: No stopping criterion is bulletproof. The 'right' one almost always depends on the application.

Bisection Method

Demo: Bisection Method

What's the rate of convergence? What's the constant?

Linear with constant $1/2$.

Fixed Point Iteration

$$\begin{aligned} x_0 &= \langle\text{starting guess}\rangle \\ x_{k+1} &= g(x_k) \end{aligned}$$

Demo: Fixed point iteration

When does fixed point iteration converge? Assume g is smooth.

Let x^* be the fixed point with $x^* = g(x^*)$.
If $|g'(x^*)| < 1$ at the fixed point, FPI converges.

Error:

$$e_{k+1} = x_{k+1} - x^* = g(x_k) - g(x^*)$$

[cont'd.]

Fixed Point Iteration: Convergence cont'd.

Error in FPI: $e_{k+1} = x_{k+1} - x^* = g(x_k) - g(x^*)$

Mean value theorem says: There is a θ_k between x_k and x^* so that

$$g(x_k) - g(x^*) = g'(\theta_k)(x_k - x^*) = g'(\theta_k)e_k.$$

So: $e_{k+1} = g'(\theta_k)e_k$ and if $\|g'\| \leqslant C < 1$ near x^*, then we have linear convergence with constant C.

Q: What if $g'(x^*) = 0$?

By Taylor:

$$g(x_k) - g(x^*) = g''(\xi_k)(x_k - x^*)^2/2$$

So we have *quadratic convergence* in this case!

We would obviously like a systematic way of finding g that produces quadratic convergence.

Newton's Method

Derive Newton's method.

Idea: Approximate f at x_k using Taylor: $f(x_k+h) \approx f(x_k) + f'(x_k)h$
Now find root of this linear approximation in terms of h:

$$f(x_k) + f'(x_k)h = 0 \quad \Leftrightarrow \quad h = -\frac{f(x_k)}{f'(x_k)}.$$

$$\begin{aligned} x_0 &= \langle\text{starting guess}\rangle \\ x_{k+1} &= x_k - \frac{f(x_k)}{f'(x_k)} = g(x_k) \end{aligned}$$

Convergence and Properties of Newton

What's the rate of convergence of Newton's method?

$$g'(x) = \frac{f(x)f''(x)}{f'(x)^2}$$

So if $f(x^*) = 0$ and $f'(x^*) \neq 0$, we have $g'(x^*) = 0$, i.e. quadratic convergence toward single roots.

Drawbacks of Newton?

- Convergence argument only good *locally*
  Will see: convergence only local (near root)
- Have to have derivative!

Demo: Newton's method

Demo: Convergence of Newton's Method

Secant Method

What would Newton without the use of the derivative look like?

Approximate

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}.$$

So

$$\begin{aligned} x_0 &= \langle \text{starting guess} \rangle \\ x_{k+1} &= x_k - \frac{f(x_k)}{\frac{f(x_k)-f(x_{k-1})}{x_k - x_{k-1}}}. \end{aligned}$$

Convergence of Properties of Secant

Rate of convergence (not shown) is $\left(1+\sqrt{5}\right)/2 \approx 1.618$.

Drawbacks of Secant?

- Convergence argument only good *locally*
  Will see: convergence only local (near root)
- Slower convergence
- Need two starting guesses

Demo: Secant Method
Demo: Convergence of the Secant Method

Secant (and similar methods) are called Quasi-Newton Methods.

Root Finding with Interpolants

Secant method uses a linear interpolant based on points $f(x_k)$, $f(x_{k-1})$, could use more points and higher-order interpolant:

- Can fit polynomial to (subset of) $(x_0, f(x_0)), \ldots, (x_k, f(x_k))$
- Look for a root of that
- Fit a quadratic to the last three: Muller's method
  - Also finds complex roots
  - Convergence rate $r \approx 1.84$

What about existence of roots in that case?

- Inverse quadratic interpolation
  - Interpolate quadratic polynomial q so that $q(f(x_i)) = x_i$ for $i \in \{k, k-1, k-2\}$.
  - Approximate root by evaluating $x_{k+1} = q(0)$.

Achieving Global Convergence

The linear approximations in Newton and Secant are only good locally. How could we use that?

- Hybrid methods: bisection + Newton
  - Stop if Newton leaves bracket
- Fix a region where they're 'trustworthy' (trust region methods)
- Limit step size

In-Class Activity: Nonlinear Equations

In-class activity: Nonlinear Equations

Fixed Point Iteration

$$\begin{aligned} \mathrm{x}_0 &= \langle\text{starting guess}\rangle \\ \mathrm{x}_{k+1} &= \mathrm{g}(\mathrm{x}_k) \end{aligned}$$

When does this converge?

Converges (locally) if $\|J_{\mathrm{g}}(\mathrm{x}^*)\| < 1$ in some norm, where the *Jacobian matrix*

$$J_{\mathrm{g}}(\mathrm{x}^*) = \begin{bmatrix} \partial_{x_1} g_1 & \cdots & \partial_{x_n} g_1 \\ \vdots & & \\ \partial_{x_1} g_n & \cdots & \partial_{x_n} g_n \end{bmatrix}.$$

Similarly: If $J_{\mathrm{g}}(\mathrm{x}^*) = 0$, we have at least quadratic convergence.

Better: There exists a norm $\|\cdot\|_A$ such that $\rho(A) \le \|A\|_A < \rho(A) + \epsilon$, so $\rho(A) < 1$ suffices.

Newton's Method

What does Newton's method look like in n dimensions?

Approximate by linear: $\mathrm{f}(\mathrm{x}+\mathrm{s}) = \mathrm{f}(\mathrm{x}) + J_{\mathrm{f}}(\mathrm{x})\mathrm{s}$.

Set to 0:

$$J_{\mathrm{f}}(\mathrm{x})\mathrm{s} = -\mathrm{f}(\mathrm{x}) \quad \Rightarrow \quad \mathrm{s} = -(J_{\mathrm{f}}(\mathrm{x}))^{-1}\mathrm{f}(\mathrm{x}).$$

$$\begin{aligned} \mathrm{x}_0 &= \langle\text{starting guess}\rangle \\ \mathrm{x}_{k+1} &= \mathrm{x}_k - (J_{\mathrm{f}}(\mathrm{x}_k))^{-1}\mathrm{f}(\mathrm{x}_k) \end{aligned}$$

Downsides of n-dim. Newton?

- Still only locally convergent
- Computing and inverting J_{f} is expensive.

Secant in n dimensions?

What would the secant method look like in n dimensions?

Need an 'approximate Jacobian' satisfying

$$\tilde{J}(\mathrm{x}_{k+1} - \mathrm{x}_k) = \mathrm{f}(\mathrm{x}_{k+1}) - \mathrm{f}(\mathrm{x}_k).$$

Suppose we have *already taken* a step to x_{k+1}. Could we 'reverse engineer' $\tilde{J}$ from that equation?

- No: n^2 unknowns in $\tilde{J}$, but only n equations
- Can only hope to 'update' $\tilde{J}$ with information from current guess.

One choice: *Broyden's method* (minimizes change to $\tilde{J}$)

Outline

Introduction to Scientific Computing

Systems of Linear Equations

Linear Least Squares

Eigenvalue Problems

Nonlinear Equations

Optimization
- Introduction
- Methods for unconstrained opt. in one dimension
- Methods for unconstrained opt. in n dimensions
- Nonlinear Least Squares
- Constrained 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

Optimization: Problem Statement

Have: Objective function $f : \mathbb{R}^n \to \mathbb{R}$
Want: Minimizer $\mathrm{x}^* \in \mathbb{R}^n$ so that

$$f(\mathrm{x}^*) = \min_{\mathrm{x}} f(\mathrm{x}) \quad \text{subject to} \quad \mathrm{g}(\mathrm{x}) = 0 \quad \text{and} \quad \mathrm{h}(\mathrm{x}) \leqslant 0.$$

- $\mathrm{g}(\mathrm{x}) = 0$ and $\mathrm{h}(\mathrm{x}) \leqslant 0$ are called constraints.
  They define the set of feasible points $\mathrm{x} \in S \subseteq \mathbb{R}^n$.
- If g or h are present, this is constrained optimization.
  Otherwise unconstrained optimization.
- If f, g, h are *linear*, this is called linear programming.
  Otherwise nonlinear programming.

Optimization: Observations

Q: What if we are looking for a *maximizer* not a minimizer?

Give some examples:

- What is the fastest/cheapest/shortest... way to do...?
- Solve a system of equations 'as well as you can' (if no exact solution exists)–similar to what least squares does for linear systems:

$$\min \|F(x)\|$$

What about multiple objectives?

- In general: Look up Pareto optimality.
- For 450: Make up your mind–decide on one (or build a combined objective). Then we'll talk.

Existence/Uniqueness

Terminology: global minimum / local minimum

Under what conditions on f can we say something about existence/uniqueness?

If $f : S \to \mathbb{R}$ is continuous on a closed and bounded set $S \subseteq \mathbb{R}^n$, then

a minimum exists.

$f : S \to \mathbb{R}$ is called *coercive* on $S \subseteq \mathbb{R}^n$ (which must be unbounded) if

$$\lim_{\|x\| \to \infty} f(x) = +\infty$$

If f is coercive, ...

a global minimum exists (but is possibly non-unique).

Convexity

$S \subseteq \mathbb{R}^n$ is called convex if for all $x, y \in S$ and all $0 \leqslant \alpha \leqslant 1$

$$\alpha x + (1 - \alpha)y \in S.$$

$f : S \to \mathbb{R}$ is called convex on $S \subseteq \mathbb{R}^n$ if for $\setminus$ $x, y \in S$ and all $0 \leqslant \alpha \leqslant 1$

$$f(\alpha x + (1 - \alpha)y \in S) \leqslant \alpha f(x) + (1 - \alpha) f(y).$$

With '$<$': *strictly convex.*

Q: Give an example of a convex, but not strictly convex function.

Convexity: Consequences

If f is convex, ...

- then f is continuous at interior points.
  (Why? What would happen if it had a jump?)
- a local minimum is a *global* minimum.

If f is strictly convex, ...

- a local minimum is a *unique global* minimum.

Optimality Conditions

If we have found a candidate x^* for a minimum, how do we know it actually is one? Assume f is smooth, i.e. has all needed derivatives.

- In one dimension:
  - Necessary: $f'(x^*) = 0$ (i.e. x^* is an extremal point)
  - Sufficient: $f'(x^*) = 0$ and $f''(x^*) > 0$ (implies x^* is a local minimum)
- In n dimensions:
  - Necessary: $\nabla f(\mathrm{x}^*) = 0$ (i.e. x^* is an extremal point)
  - Sufficient: $\nabla f(\mathrm{x}^*) = 0$ and $H_f(\mathrm{x}^*)$ positive definite (implies x^* is a local minimum)

where the Hessian

$$H_f(\mathrm{x}^*) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2}{\partial x_n^2} \end{bmatrix} f(\mathrm{x}^*).$$

Optimization: Observations

Q: Come up with a hypothetical approach for finding minima.

A: Solve $\nabla f = 0$.

Q: Is the Hessian symmetric?

A: Yes. (Schwarz's theorem)

Q: How can we practically test for positive definiteness?

A: Attempt a Cholesky factorization. If it succeeds, the matrix is PD.

In-Class Activity: Optimization Theory

In-class activity: Optimization Theory

Sensitivity and Conditioning (1D)

How does optimization react to a slight perturbation of the minimum?

Suppose we still have $|f(\tilde{x}) - f(x^*)| < \text{tol}$ (where x^* is true min.). Apply Taylor's theorem:

$$f(x^* + h) = f(x^*) + \underbrace{f'(x^*)}_{0} h + f''(x^*)\frac{h^2}{2} + O(h^3)$$

Solve for h: $|\tilde{x} - x^*| \leqslant \sqrt{2\,\text{tol}\,/f''(x^*)}$.
In other words: Can expect *half as many digits* in $\tilde{x}$ as in $f(\tilde{x})$.
This is important to keep in mind when setting tolerances.

It's only possible to do better when derivatives are explicitly known and convergence is not based on function values alone. (then: can solve $\nabla f = 0$)

Sensitivity and Conditioning (nD)

How does optimization react to a slight perturbation of the minimum?

Assume $\|\mathrm{s}\| = 1$.

$$f(\mathrm{x}^* + h\mathrm{s}) = f(\mathrm{x}^*) + h\underbrace{\nabla f(\mathrm{x}^*)^T}_{0}\mathrm{s} + \frac{h^2}{2}\mathrm{s}^T H_f(\mathrm{x}^*)\mathrm{s} + O(h^3)$$

Yields:

$$|h|^2 \leqslant \frac{2\,\mathrm{tol}}{\lambda_{\min}(H_f(\mathrm{x}^*))}.$$

In other words: Conditioning of H_f determines sensitivity of x^*.

Unimodality

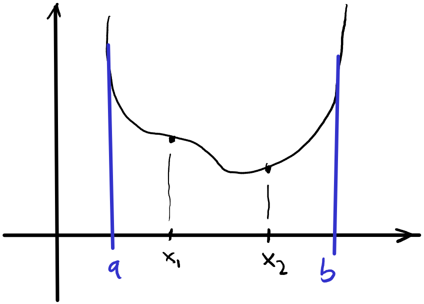
Would like a method like bisection, but for optimization.
In general: No invariant that can be preserved.
Need *extra assumption.*

f is called unimodal if for all $x_1 < x_2$

- $x_2 < x^* \Rightarrow f(x_1) > f(x_2)$
- $x^* < x_1 \Rightarrow f(x_1) < f(x_2)$

Golden Section Search

Suppose we have an interval with f unimodal:



Would like to maintain unimodality.

1. Pick x_1, x_2
2. If $f(x_1) > f(x_2)$, reduce to (x_1, b)
3. If $f(x_1) \leqslant f(x_2)$, reduce to (a, x_2)

Golden Section Search: Efficiency

Where to put x_1, x_2?

- Want symmetry:
  $x_1 = a + (1-\tau)(b-a)$
  $x_2 = a + \tau(b-a)$
- Want to reuse function evaluations: $\tau^2 = 1 - \tau$
  Find: $\tau = \left(\sqrt{5}-1\right)/2$. Also known as the *golden section*.
- Hence golden section search.

Convergence rate?

Linearly convergent. Can we do better?

Demo: Golden Section Proportions

Newton's Method

Reuse the Taylor approximation idea, but for optimization.

$$f(x+h) \approx f(x) + f'(x)h + f''(x)\frac{h^2}{2} =: \hat{f}(h)$$

Solve

$$0 = \hat{f}'(h) = f'(x) + f''(x)h :$$

$h = -f'(x)/f''(x)$.

1. $x_0 = \langle \text{some starting guess} \rangle$
2. $x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$

Q: Notice something? Identical to Newton for solving $f'(x) = 0$. Because of that: locally quadratically convergent.

Good idea: Combine slow-and-safe (bracketing) strategy with fast-and-risky (Newton).

In-Class Activity: Optimization Methods

In-class activity: Optimization Methods

Steepest Descent

Given a scalar function $f : \mathbb{R}^n \to \mathbb{R}$ at a point $\mathbf{x}$, which way is down?

Direction of steepest descent: $-\nabla f$

Q: How far along the gradient should we go?

Unclear–do a line search. For example using Golden Section Search.

1. $\mathbf{x}_0 = \langle\text{some starting guess}\rangle$
2. $\mathbf{s}_k = -\nabla f(\mathbf{x}_k)$
3. Use line search to choose α_k to minimize $f(\mathbf{x}_k + \alpha_k \mathbf{s}_k)$
4. $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k$
5. Go to 2.

Observation: (from demo)

- Linear convergence

Demo: Steepest Descent

Steepest Descent: Convergence

Consider quadratic model problem:

$$f(\mathrm{x}) = \frac{1}{2}\mathrm{x}^T A\mathrm{x} + \mathrm{c}^T\mathrm{x}$$

where A is SPD. (A good model of f near a minimum.)

Define error $\mathrm{e}_k = \mathrm{x}_k - \mathrm{x}^*$. Then

$$||\mathrm{e}_{k+1}||_A = \sqrt{\mathrm{e}_{k+1}^T A\mathrm{e}_{k+1}} = \frac{\sigma_{\max}(A) - \sigma_{\min}(A)}{\sigma_{\max}(A) + \sigma_{\min}(A)}||\mathrm{e}_k||_A$$

$\to$ confirms linear convergence.

Convergence constant related to conditioning:

$$\frac{\sigma_{\max}(A) - \sigma_{\min}(A)}{\sigma_{\max}(A) + \sigma_{\min}(A)} = \frac{\kappa(A) - 1}{\kappa(A) + 1}.$$

Hacking Steepest Descent for Better Convergence

Extrapolation methods: Look back a step, maintain '*momentum*'.

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta_k (x_k - x_{k-1})$$

Heavy ball method: constant $\alpha_k = \alpha$ and $\beta_k = \beta$. Gives:

$$||e_{k+1}||_A = \frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} ||e_k||_A$$

Conjugate gradient method:

$$(\alpha_k, \beta_k) = \text{argmin}_{\alpha_k, \beta_k} \left[f\left(x_k - \alpha_k \nabla f(x_k) + \beta_k (x_k - x_{k-1}) \right) \right]$$

- Will see in more detail later (for solving linear systems)
- Provably optimal first-order method for the quadratic model problem
- Turns out to be closely related to Lanczos (A-orthogonal search directions)

Nelder-Mead Method

Idea:

Form a n-point polytope in n-dimensional space and adjust worst point (highest function value) by moving it along a line passing through the centroid of the remaining points.

Demo: Nelder-Mead Method

Newton's method (n D)

What does Newton's method look like in n dimensions?

Build a Taylor approximation:

$$f(\mathrm{x}+\mathrm{s}) \approx f(\mathrm{x}) + \nabla f(\mathrm{x})^T \mathrm{s} + \frac{1}{2}\mathrm{s}^T H_f(\mathrm{x})\mathrm{s} =: \hat{f}(\mathrm{s})$$

Then solve $\nabla \hat{f}(\mathrm{s}) = 0$ for s to find

$$H_f(\mathrm{x})\mathrm{s} = -\nabla f(\mathrm{x}).$$

1. $x_0 = \langle$some starting guess$\rangle$
2. Solve $H_f(\mathrm{x}_k)\mathrm{s}_k = -\nabla f(\mathrm{x}_k)$ for s_k
3. $\mathrm{x}_{k+1} = \mathrm{x}_k + \mathrm{s}_k$

Newton's method (n D): Observations

Drawbacks?

- Need second (!) derivatives
  (addressed by Conjugate Gradients, later in the class)
- local convergence
- Works poorly when H_f is nearly indefinite

Demo: Newton's method in n dimensions

Quasi-Newton Methods

Secant/Broyden-type ideas carry over to optimization. How?

Come up with a way to update to update the approximate Hessian.

$$\mathrm{x}_{k+1} = \mathrm{x}_k - \alpha_k B_k^{-1} \nabla f(\mathrm{x}_k)$$

- α_k: a line search/damping parameter.

BFGS: Secant-type method, similar to Broyden:

$$\mathrm{B}_{k+1} = \mathrm{B}_k + \frac{\mathrm{y}_k \mathrm{y}_k^T}{\mathrm{y}_k^T \mathrm{s}_k} - \frac{\mathrm{B}_k \mathrm{s}_k \mathrm{s}_k^T \mathrm{B}_k}{\mathrm{s}_k^T \mathrm{B}_k \mathrm{s}_k}$$

where

- $\mathrm{s}_k = \mathrm{x}_{k+1} - \mathrm{x}_k$
- $\mathrm{y}_\mathrm{k} = \nabla f(\mathrm{x}_{k+1}) - \nabla f(\mathrm{x}_k)$

Nonlinear Least Squares: Setup

What if the f to be minimized is actually a 2-norm?

$$f(\mathbf{x}) = \|\mathbf{r}(\mathbf{x})\|_2, \qquad \mathbf{r}(\mathbf{x}) = \mathbf{y} - \mathbf{a}(\mathbf{x})$$

Define ‘helper function’

$$\varphi(\mathbf{x}) = \frac{1}{2}\mathbf{r}(\mathbf{x})^T\mathbf{r}(\mathbf{x}) = \frac{1}{2}f^2(\mathbf{x})$$

and minimize that instead.

$$\frac{\partial}{\partial x_i}\varphi = \frac{1}{2}\sum_{j=1}^{n}\frac{\partial}{\partial x_i}[r_j(\mathbf{x})^2] = \sum_j \left(\frac{\partial}{\partial x_i} r_j\right) r_j,$$

or, in matrix form:

$$\nabla\varphi = J_{\mathbf{r}}(\mathbf{x})^T\mathbf{r}(\mathbf{x}).$$

Gauss-Newton

For brevity: $J := J_{\mathrm{r}}(\mathrm{x})$.

Can show similarly:

$$H_\varphi(\mathrm{x}) = J^T J + \sum_i r_i H_{r_i}(\mathrm{x}).$$

Newton step s can be found by solving $H_\varphi(\mathrm{x})\mathrm{s} = -\nabla\varphi$.

Observation: $\sum_i r_i H_{r_i}(\mathrm{x})$ is inconvenient to compute *and* unlikely to be large (since it's multiplied by components of the residual, which is supposed to be small) $\to$ forget about it.

Gauss-Newton method: Find step s by $J^T J\mathrm{s} = -\nabla\varphi = -J^T\mathrm{r}(\mathrm{x})$. Does that remind you of the *normal equations*? $J\mathrm{s} \cong -\mathrm{r}(\mathrm{x})$. Solve that using our existing methods for least-squares problems.

Gauss-Newton: Observations?

Demo: Gauss-Newton

Observations?

- Newton on its own is still only locally convergent
- Gauss-Newton is clearly similar
- It's worse because the step is only approximate
  $\rightarrow$ Much depends on the starting guess.

Levenberg-Marquardt

If Gauss-Newton on its own is poorly, conditioned, can try Levenberg-Marquardt:

$$(J_r(x_k)^T J_r(x_k) + \mu_k I)s_k = -J_r(x_k)^T r(x_k)$$

for a 'carefully chosen' μ_k. This makes the system matrix 'more invertible' but also less accurate/faithful to the problem. Can also be translated into a least squares problem (see book).

What Levenberg-Marquardt does is generically called regularization: Make H more positive definite.
Easy to rewrite to least-squares problem:

$$\begin{bmatrix} J_r(x_k) \\ \sqrt{\mu_k} \end{bmatrix} s_k \cong \begin{bmatrix} -r(x_k) \\ 0 \end{bmatrix}.$$

Constrained Optimization: Problem Setup

Want x^* so that

$$f(\mathrm{x}^*) = \min_{\mathrm{x}} f(\mathrm{x}) \quad \text{subject to} \quad \mathrm{g}(\mathrm{x}) = 0$$

No inequality constraints just yet. This is *equality-constrained optimization*. Develop a necessary condition for a minimum.

Unconstrained necessary condition:

$$\nabla f(\mathrm{x}) = 0$$

Problem: That alone is not helpful. 'Downhill' direction has to be *feasible*. So just this doesn't help.

s is a feasible direction at x if

$$\mathrm{x} + \alpha \mathrm{s} \quad \text{feasible for } \alpha \in [0, r] \quad (\text{for some } r)$$

Constrained Optimization: Necessary Condition

- $\nabla f(\mathrm{x}) \cdot \mathrm{s} \geqslant 0$ ("uphill that way") for any feasible direction s.

  If not at boundary of feasible set:
  s and $-s$ are feasible directions
  $\Rightarrow \nabla f(\mathrm{x}) = 0$
  $\Rightarrow$ Only the boundary of the feasible set is different from the unconstrained case (i.e. interesting)

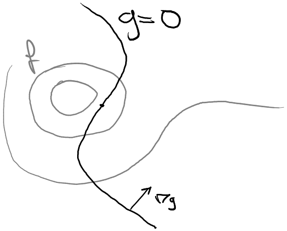
- At boundary: $g(x) = 0$. Need:

  $$-\nabla f(\mathrm{x}) \in \mathrm{rowspan}(J_{\mathrm{g}})$$

  a.k.a. "all descent directions would cause a change ($\rightarrow$violation) of the constraints."
  Q: Why 'rowspan'? Think about shape of J_{g}.

  $$\Leftrightarrow -\nabla f(\mathrm{x}) = J_{\mathrm{g}}^T \lambda \quad \text{for some } \lambda.$$

Lagrange Multipliers



Seen: Need $-\nabla f(\mathrm{x}) = J_{\mathrm{g}}^T \lambda$ at the (constrained) optimum.

Idea: Turn constrained optimization problem for x into an *unconstrained* optimization problem for (x, λ). How?

Need a new function $\mathcal{L}(\mathrm{x}, \lambda)$ to minimize:

$$\mathcal{L}(\mathrm{x}, \lambda) := f(\mathrm{x}) + \lambda^T \mathrm{g}(\mathrm{x}).$$

Lagrange Multipliers: Development

$$\mathcal{L}(\mathrm{x}, \lambda) := f(\mathrm{x}) + \lambda^T \mathrm{g}(\mathrm{x}).$$

Then: $\nabla \mathcal{L} = 0$ at unconstrained minimum, i.e.

$$0 = \nabla \mathcal{L} = \begin{bmatrix} \nabla_{\mathrm{x}} \mathcal{L} \\ \nabla_{\lambda} \mathcal{L} \end{bmatrix} = \begin{bmatrix} \nabla f + J_{\mathrm{g}}(\mathrm{x})^T \lambda \\ \mathrm{g}(\mathrm{x}) \end{bmatrix}.$$

Convenient: This matches our necessary condition!

So we could use any unconstrained method to minimized $\mathcal{L}$.
For example: Using Newton to minimize $\mathcal{L}$ is called *Sequential Quadratic Programming*. ('SQP')

Demo: Sequential Quadratic Programming

Inequality-Constrained Optimization

Want x^* so that

$$f(x^*) = \min_x f(x) \quad \text{subject to} \quad g(x) = 0 \quad \text{and} \quad h(x) \leqslant 0$$

This is *inequality-constrained optimization*. Develop a necessary condition for a minimum.

Define Lagrangian:

$$\mathcal{L}(x, \lambda_1, \lambda_2) := f(x) + \lambda_1^T g(x) + \lambda_2^T h(x)$$

- Some inequality constrains may not be "active"
  (active $\Leftrightarrow h_i(x^*) = 0 \Leftrightarrow$at 'boundary' of ineq. constraint)
  (Equality constrains are always 'active')
- If h_i inactive ($h_i(x^*) < 0$), must force $\lambda_{2,i} = 0$.
  Otherwise: Behavior of h could change location of minimum of $\mathcal{L}$. Use complementarity condition $h_i(x^*)\lambda_{2,i} = 0$.

Inequality-Constrained Optimization (cont'd)

Develop a set of necessary conditions for a minimum.

Assuming J_{g} and $J_{\mathrm{h,active}}$ have full rank, this set of conditions is *necessary*:

$$\begin{aligned}
(*)\quad \nabla_x \mathcal{L}(x^*, \lambda_1^*, \lambda_2^*) &= 0\\
(*)\quad g(x^*) &= 0\\
h(x^*) &\leqslant 0\\
\lambda_2 &\geqslant 0\\
(*)\quad h(x^*)\cdot\lambda_2 &= 0
\end{aligned}$$

These are called the Karush-Kuhn-Tucker ('KKT') conditions.

Computational approach: Solve $(*)$ equations by Newton.

Outline

Introduction to Scientific Computing

Systems of Linear Equations

Linear Least Squares

Eigenvalue Problems

Nonlinear Equations

Optimization

Interpolation
- Introduction
- Methods
- Error Estimation
- Piecewise interpolation, Splines

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

Interpolation: Setup

Given: $(x_i)_{i=1}^N$, $(y_i)_{i=1}^N$
Wanted: Function f so that $f(x_i) = y_i$

How is this not the same as function fitting? (from least squares)

It's very similar–but the key difference is that we are asking for *exact equality*, not just minimization of a residual norm.
$\rightarrow$ Better error control, error not dominated by residual

Idea: There is an *underlying function* that we are approximating from the known point values.

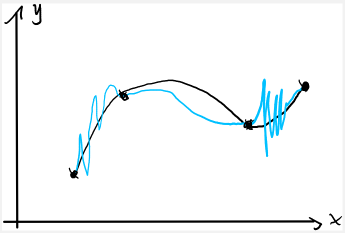
Error here: Distance from that underlying function

Interpolation: Setup (II)

Given: $(x_i)_{i=1}^N$, $(y_i)_{i=1}^N$
Wanted: Function f so that $f(x_i) = y_i$

Does this problem have a unique answer?

No–there are infinitely many functions that satisfy the problem as stated:



Interpolation: Importance

Why is interpolation important?

It brings all of calculus within range of numerical operations.

- Why?
  Because calculus works on functions.
- How?
  1. Interpolate (go from discrete to continuous)
  2. Apply calculus
  3. Re-discretize (evaluate at points)

Making the Interpolation Problem Unique

Limit the set of functions to a linear combination from an *interpolation basis* φ_i.

$$f(x) = \sum_{j=0}^{N_{\text{func}}} \alpha_j \varphi_j(x)$$

Interpolation becomes solving the linear system:

$$y_i = f(x_i) = \sum_{j=0}^{N_{\text{func}}} \alpha_j \underbrace{\varphi_j(x_i)}_{V_{ij}} \qquad \leftrightarrow \qquad V\alpha = \mathrm{y}.$$

Want unique answer: Pick $N_{\text{func}} = N \to V$ square.
V is called the *(generalized) Vandermonde matrix*.

$$V\,(\text{coefficients}) = (\text{values at nodes})\,.$$

Existence/Sensitivity

Solution to the interpolation problem: Existence? Uniqueness?

Equivalent to existence/uniqueness of the linear system

Sensitivity?

- Shallow answer: Simply consider the condition number of the linear system
- $\|\text{coefficients}\|$ does not suffice as measure of stability.
  $f(x)$ can be evaluated in many places. (f is interpolant.)
- Want: $\max_{x\in[a,b]} |f(x)| \le \Lambda \|\mathrm{y}\|_\infty$
- Λ: Lebesgue constant
- Λ depends on n and $\{x_i\}_i$
  - Technically also depends on $\{\phi_i\}_i$
  - But: same for all polynomial bases

Modes and Nodes (aka Functions and Points)

Both function basis and point set are under our control. What do we pick?

Ideas for basis functions:

- Monomials $1, x, x^2, x^3, x^4, \ldots$
- Functions that make $V = I \rightarrow$ 'Lagrange basis'
- Functions that make V triangular $\rightarrow$ 'Newton basis'
- *Splines* (piecewise polynomials)
- *Orthogonal polynomials*
- Sines and cosines
- 'Bumps' (*'Radial Basis Functions'*)

Ideas for points:

- Equispaced
- *'Edge-Clustered'* (so-called Chebyshev/Gauss/... nodes)

Specific issues:

- Why *not* monomials on equispaced points?
  Demo: Monomial interpolation
- Why not equispaced?
  Demo: Choice of Nodes for Polynomial Interpolation

Lagrange Interpolation

Find a basis so that $V = I$, i.e.

$$\varphi_j(x_i) = \begin{cases} 1 & i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Start with simple example. Three nodes: x_1, x_2, x_3

$$\begin{aligned} \varphi_1(x) &= \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} \\ \varphi_2(x) &= \frac{(x - x_1)\qquad(x - x_3)}{(x_2 - x_1)\qquad(x_2 - x_3)} \\ \varphi_3(x) &= \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} \end{aligned}$$

Numerator: Ensures φ_i zero at other nodes.
Denominator: Ensures $\varphi_i(x_i) = 1$.

Lagrange Polynomials: General Form

$$\varphi_j(x) = \frac{\prod_{k=1,k\neq j}^{m}(x - x_k)}{\prod_{k=1,k\neq j}^{m}(x_j - x_k)}$$

Newton Interpolation

Find a basis so that V is triangular.

Easier to build than Lagrange, but: coefficient finding costs $O(n^2)$.

$$\varphi_j(x) = \prod_{k=1}^{j-1} (x - x_k).$$

(At least) two possibilities for coefficent finding:

1. Set up V, run forward substitution.
2. "Divided Differences" (see, e.g. Wikipedia)

Why not Lagrange/Newton?

Cheap to form, expensive to evaluate, expensive to do calculus on.

Better conditioning: Orthogonal polynomials

What caused monomials to have a terribly conditioned Vandermonde?

Being close to linearly dependent.

What's a way to make sure two vectors are *not* like that?

Orthogonality

But polynomials are functions!

$$
\begin{aligned}
\mathsf{f}\cdot\mathsf{g} &= \sum_{i=1}^{n} f_i g_i \quad = \langle \mathsf{f},\mathsf{g}\rangle \\
\langle f,g\rangle &= \int_{-1}^{1} f(x)g(x)\mathrm{d}x
\end{aligned}
$$

Need an inner product. Orthogonal then just means $\langle f,g\rangle = 0$.

Constructing Orthogonal Polynomials

How can we find an orthogonal basis?

Apply Gram-Schmidt to the monomials.

Demo: Orthogonal Polynomials — Obtained: Legendre polynomials.
But how can I practically compute the Legendre polynomials?

$\rightarrow$ DLMF: Chapter on orthogonal polynomials

- There exist three-term recurrences, e.g.: $T_{n+1} = 2xT_n - T_{n+1}$
- There is a whole zoo of polynomials there, depending on the weight function w in the (generalized) inner product:

$$\langle f, g \rangle = \int w(x) f(x) g(x) \mathrm{d}x.$$

Some sets of orth. polys. live on intervals $\neq (-1, 1)$.

Chebyshev Polynomials: Definitions

Three equivalent definitions:

- Result of Gram-Schmidt with weight $1/\sqrt{1-x^2}$. What is that weight?

  $$\boxed{1/\,\text{(Half circle), i.e. } x^2+y^2=1, \text{ with } y=\sqrt{1-x^2}}$$

  (Like for Legendre, you won't exactly get the standard normalization if you do this.)
- $T_k(x) = \cos(k\cos^{-1}(x))$
- $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ plus $T_0 = 1$, $T_0 = x$

Demo: Chebyshev Interpolation (Part 1)

Chebyshev Interpolation

What is the Vandermonde matrix for Chebyshev polynomials?

- Need to know the nodes to answer that
- The answer would be very simple if the nodes were $\cos(*)$.
- So why not $\cos(\text{equispaced})$?

Might get

$$x_i = \cos\left(\frac{i}{k}\pi\right) \qquad (i = 0, 1, \ldots, k)$$

These are just the *extrema* (minima/maxima) of T_k.

$$V_{ij} = \cos\left(j \cos^{-1}\left(\cos\left(\frac{i}{k}\pi\right)\right)\right) = \cos\left(j\frac{i}{k}\pi\right).$$

This is called the *Discrete Cosine Transform* and a matvec with this matrix (and its inverse!) can be implemented in $O(N \log N)$ time (similar to the *Fast Fourier Transform*→Chapter 12).

Chebyshev Nodes

Might also consider roots (instead of extrema) of T_k:

$$x_i = \cos\left(\frac{2i-1}{2k}\pi\right) \quad (i = 1\dots,k).$$

Vandermonde for these (with T_k) can be applied in $O(N \log N)$ time, too.
It turns out that we were still looking for a good set of interpolation nodes.
We came up with the criterion that the nodes should bunch towards the ends. Do these do that?

Yes.

Demo: Chebyshev Interpolation (Part 2)

Chebyshev Interpolation: Summary

- Chebyshev interpolation is fast and works extremely well
- `http://www.chebfun.org/` and: ATAP
- In 1D, they're a very good answer to the interpolation question
- But sometimes a piecewise approximation (with a specifiable level of smoothness) is more suited to the application

In-Class Activity: Interpolation

In-class activity: Interpolation

Interpolation Error

If f is n times continuously differentiable on a closed interval I and $p_{n-1}(x)$ is a polynomial of degree at most n that interpolates f at n distinct points $\{x_i\}$ $(i = 1, ..., n)$ in that interval, then for each x in the interval there exists ξ in that interval such that

$$f(x) - p_{n-1}(x) = \frac{f^{(n)}(\xi)}{n!}(x - x_1)(x - x_2)\cdots(x - x_n).$$

Set the error term to be $R(x) := f(x) - p_{n-1}(x)$ and set up an auxiliary function:

$$Y(t) = R(t) - \frac{R(x)}{W(x)}W(t) \quad \text{where} \quad W(t) = \prod_{i=1}^{n}(t - x_i).$$

Note also the introduction of t as an additional variable, independent of the point x where we hope to prove the identity.

Interpolation Error: Proof cont'd

$$Y(t) = R(t) - \frac{R(x)}{W(x)}W(t) \quad \text{where} \quad W(t) = \prod_{i=1}^{n}(t - x_i)$$

- Since x_i are roots of $R(t)$ and $W(t)$, we have $Y(x) = Y(x_i) = 0$, which means Y has at least $n+1$ roots.
- From Rolle's theorem, $Y'(t)$ has at least n roots, then $Y^{(n)}$ has at least one root ξ, where $\xi \in I$.
- Since $p_{n-1}(x)$ is a polynomial of degree at most $n-1$, $R^{(n)}(t) = f^{(n)}(t)$. Thus

$$Y^{(n)}(t) = f^{(n)}(t) - \frac{R(x)}{W(x)}n!.$$

- Plugging $Y^{(n)}(\xi) = 0$ into the above yields the result.

Error Result: Connection to Chebyshev

What is the connection between the error result and Chebyshev interpolation?

- The error bound suggests choosing the interpolation nodes such that the product $|\prod_{i=1}^{n}(x - x_i)|$, is as small as possible. The Chebyshev nodes achieve this.
- Error is zero at the nodes
- If nodes scoot closer together near the interval ends, then

$$(x - x_1)(x - x_2)\cdots(x - x_n)$$

clamps down the (otherwise quickly-growing) error there.

Error Result: Simplified From

Boil the error result down to a simpler form.

Assume $x_1 < \cdots < x_n$.

- $\left|f^{(n)}(x)\right| \leqslant M$ for $x \in [x_1, x_n]$,
- Set the interval length $h = x_n - x_1$.
  Then $|x - x_i| \leqslant h$.

Altogether–there is a constant C independent of h so that:

$$\max_x |f(x) - p_{n-1}(x)| \leqslant CMh^n.$$

For the grid spacing $h \to 0$, we have

$$E(h) = O(h^n).$$

This is called *convergence of order n*.

Demo: Interpolation Error

Going piecewise: Simplest Case

Construct a piecweise linear interpolant at four points.

x_0, y_0		x_1, y_1		x_2, y_2		x_3, y_3
|	$f_1 = a_1 x + b_1$	|	$f_2 = a_2 x + b_2$	|	$f_3 = a_3 x + b_3$	|
|	2 unk.	|	2 unk.	|	2 unk.	|
|	$f_1(x_0) = y_0$	|	$f_2(x_1) = y_1$	|	$f_3(x_2) = y_2$	|
|	$f_1(x_1) = y_1$	|	$f_2(x_2) = y_2$	|	$f_3(x_3) = y_3$	|
|	2 eqn.	|	2 eqn.	|	2 eqn.	|

Why three intervals?

General situation $\rightarrow$ two end intervals and one middle interval. Can just add more middle intervals if needed.

Piecewise Cubic ('Splines')

x_0, y_0		x_1, y_1		x_2, y_2		x_3, y_3
	f_1		f_2		f_3	
	$a_1x^3 + b_1x^2 + c_1x + d_1$		$a_2x^3 + b_2x^2 + c_2x + d_2$		$a_3x^3 + b_3x^2 + c_3x + d_3$	

4 unknowns	4 unknowns	4 unknowns
$f_1(x_0) = y_0$	$f_2(x_1) = y_1$	$f_3(x_2) = y_2$
$f_1(x_1) = y_1$	$f_2(x_2) = y_2$	$f_3(x_3) = y_3$

Not enough: need more conditions. Ask for more smoothness.

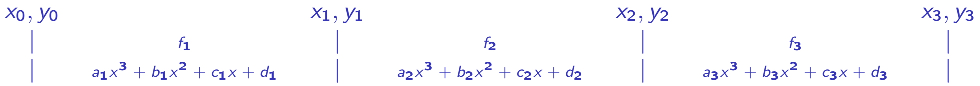
$$f_1'(x_1) = f_2'(x_1) \qquad f_2'(x_2) = f_3'(x_2)$$
$$f_1''(x_1) = f_2''(x_1) \qquad f_2''(x_2) = f_3''(x_2)$$

Not enough: need yet more conditions.

$$f_1''(x_0) = 0 \qquad f_3''(x_3) = 0$$

Now: have a square system.

Piecewise Cubic ('Splines'): Accounting


x_0, y_0
x_1, y_1
x_2, y_2
x_3, y_3
f_1
$a_1x^3 + b_1x^2 + c_1x + d_1$
f_2
$a_2x^3 + b_2x^2 + c_2x + d_2$
f_3
$a_3x^3 + b_3x^2 + c_3x + d_3$

Number of conditions: $2N_{\text{intervals}} + 2N_{\text{middle nodes}} + 2$ where

$$N_{\text{intervals}} - 1 = N_{\text{middle nodes}}$$

so

$$2N_{\text{intervals}} + 2(N_{\text{intervals}} - 1) + 2 = 4N_{\text{intervals}},$$

which is exactly the number of unknown coefficients.

These conditions are fairly arbitrary: Can choose different ones basically at will. The above choice: *'natural spline'*.

Can also come up with a basis of spline functions (with the chosen smoothness conditions). These are called B-Splines.

Outline

Introduction to Scientific Computing

Systems of Linear Equations

Linear Least Squares

Eigenvalue Problems

Nonlinear Equations

Optimization

Interpolation

Numerical Integration and Differentiation
- Numerical Integration
- Quadrature Methods
- Accuracy and Stability
- Gaussian Quadrature
- Composite Quadrature
- Numerical Differentiation
- Richardson Extrapolation

Initial Value Problems for ODEs

Boundary Value Problems for ODEs

Partial Differential Equations and Sparse Linear Algebra

Fast Fourier Transform

Additional Topics

Numerical Integration: About the Problem

What is numerical integration? (Or quadrature?)

Given a, b, f, compute

$$\int_a^b f(x)\mathrm{d}x.$$

What about existence and uniqueness?

- Answer exists e.g. if f is *integrable* in the Riemann or Lebesgue senses.
- Answer is unique if f is e.g. piecewise continuous and bounded. (this also implies existence)

Conditioning

Derive the (absolute) condition number for numerical integration.

Let $\hat{f}(x) := f(x) + e(x)$, where $e(x)$ is a perturbation.

$$\begin{aligned} & \left| \int_a^b f(x)\mathrm{d}x - \int_a^b \hat{f}(x)\mathrm{d}x \right| \\ = & \left| \int_a^b e(x)\mathrm{d}x \right| \leqslant \int_a^b |e(x)|\,\mathrm{d}x \leqslant (b-a) \max_{x\in[a,b]} |e(x)|\,. \end{aligned}$$

Interpolatory Quadrature

Design a quadrature method based on interpolation.

Idea: The result ought to be a linear (Q: why linear?) combination of a few function values.

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n} \omega_i f(x_i)$$

Then: *nodes* (x_i) and *weights* (ω_i) together make a quadrature rule.

Idea: Any interpolation method (nodes+basis) gives rise to an *interpolatory quadrature method*.

Interpolatory Quadrature: Examples

Example: Fix (x_i). Then

$$f(x) \approx \sum_i f(x_i)\ell_i(x),$$

where $\ell_i(x)$ is the Lagrange polynomial for the node x_i. Then

$$\int_a^b f(x)\mathrm{d}x \approx \sum_i f(x_i) \underbrace{\int_a^b \ell_i(x)\mathrm{d}x}_{\omega_i}.$$

- With polynomials and (often) equispaced nodes, this is called Newton-Cotes quadrature.
- With Chebyshev nodes and Chebyshev weights, this is called Clenshaw-Curtis quadrature.

Interpolatory Quadrature: Computing Weights

How do the weights in interpolatory quadrature get computed?

Done by solving linear system.
Know: This quadrature should at least integrate monomials exactly.

$$\begin{aligned} b-a &= \int_a^b 1 \mathrm{d}x = \omega_1 \cdot 1 + \cdots + \omega_n \cdot 1 \\ &\vdots \\ \frac{1}{k+1}(b^{k+1}-a^{k+1}) &= \int_a^b x^k \mathrm{d}x = \omega_1 \cdot x_1^k + \cdots + \omega_n \cdot x_n^k \end{aligned}$$

Write down n equations for n unknowns, solve linear system, done.

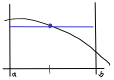
This is called the *method of undetermined coefficients*.

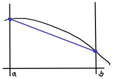
Demo: Newton-Cotes weight finder

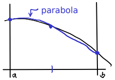
Examples and Exactness

To what polynomial degree are the following rules exact?

Midpoint rule	$(b-a)f\left(\frac{a+b}{2}\right)$
Trapezoidal rule	$\frac{b-a}{2}(f(a)+f(b))$
Simpson's rule	$\frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right)$







Answers:

- Midpoint: technically 0 (constants), actually 1 (linears)
- Trapezoidal: 1 (linears)
- Simpson's: technically 2 (parabolas), actually 3 (cubics)

Idea: Could use difference between trapezoidal and midpoint rule as an error estimate.

Interpolatory Quadrature: Accuracy

Let p_{n-1} be an interpolant of f at nodes $x_1, \ldots, x_n$ (of degree $n-1$)
Recall

$$\sum_i \omega_i f(x_i) = \int_a^b p_{n-1}(x)\mathrm{d}x.$$

What can you say about the accuracy of the method?

$$\begin{aligned}
& & & \left|\int_a^b f(x)\mathrm{d}x - \int_a^b p_{n-1}(x)\mathrm{d}x\right| \\
& & \leqslant & \int_a^b |f(x) - p_{n-1}(x)|\,\mathrm{d}x \\
& & \leqslant & (b-a)\,\|f - p_{n-1}\|_\infty \\
& \text{(using interpolation error)} & \leqslant & C(b-a)h^n \left\|f^{(n)}\right\|_\infty \\
& & \leqslant & Ch^{n+1} \left\|f^{(n)}\right\|_\infty
\end{aligned}$$

Quadrature: Overview of Rules

	n	Deg.	Ex.Int.Deg. (w/odd)	Intp.Ord.	Quad.Ord. (regular)	Quad.Ord. (w/odd)
		$n-1$	$(n-1)+1_{\text{odd}}$	n	$n+1$	$(n+1)+1_{\text{odd}}$
Midp.	1	0	1	1	2	3
Trapz.	2	1	1	2	3	3
Simps.	3	2	3	3	4	5
—	4	3	3	4	5	5

- n: number of points
- "Deg.": Degree of polynomial used in interpolation ($= n-1$)
- "Ex.Int.Deg.": Polynomials of up to (and including) this degree *actually* get integrated exactly. (including the odd-order bump)
- "Intp.Ord.": Order of Accuracy of Interpolation: $O(h^n)$
- "Quad.Ord. (regular)": Order of accuracy for quadrature predicted by the error result above: $O(h^{n+1})$
- "Quad.Ord. (w/odd):" Actual order of accuracy for quadrature given 'bonus' degrees for rules with odd point count

Observation: Quadrature gets (at least) 'one order higher' than interpolation–even more for odd-order rules. (i.e. more accurate)

Interpolatory Quadrature: Stability

Let p_n be an interpolant of f at nodes $x_1, \ldots, x_n$ (of degree $n-1$)
Recall

$$\sum_i \omega_i f(x_i) = \int_a^b p_n(x)\mathrm{d}x$$

What can you say about the stability of this method?

Again consider $\hat{f}(x) = f(x) + e(x)$.

$$\left|\sum_i \omega_i f(x_i) - \sum_i \omega_i \hat{f}(x_i)\right| = \left|\sum_i \omega_i e(x_i)\right| \leqslant \sum_i |\omega_i e(x_i)|$$

$$\leqslant \left(\sum_i |\omega_i|\right) \|e\|_\infty$$

Q: So, what quadrature weights make for bad stability bounds?
A: Quadratures with large negative weights. (Recall: $\sum_i \omega_i$ is fixed.)

About Newton-Cotes

What's not to like about Newton-Cotes quadrature?

Demo: Newton-Cotes weight finder (again, with many nodes)
In fact, Newton-Cotes must have at least one negative weight as soon as $n \geqslant 11$.

More drawbacks:

- All the fun of high-order interpolation with monomials and equispaced nodes (i.e. convergence not guaranteed)
- Weights possibly non-negative ($\rightarrow$stability issues)
- Coefficients determined by (possibly ill-conditioned) Vandermonde matrix
- Thus hard to extend to arbitrary number of points.

Gaussian Quadrature

So far: nodes chosen from outside.
Can we gain something if we let the quadrature rule choose the nodes, too? Hope: More design freedom $\rightarrow$ Exact to higher degree.

Idea: method of undetermined coefficients
But: Resulting system would be nonlinear.

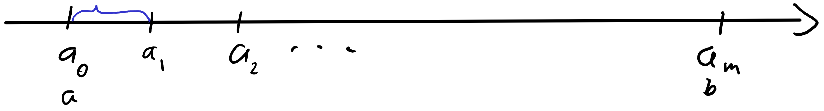
Can use orthogonal polynomials to get a leg up. ($\rightarrow$ hw)
Gaussian quadrature with n points: Exactly integrates polynomials up to degree $2n-1$.

Demo: Gaussian quadrature weight finder

Composite Quadrature

High-order polynomial interpolation requires a high degree of smoothness of the function.
Idea: Stitch together multiple lower-order quadrature rules to alleviate smoothness requirement.

e.g. trapezoidal



Error in Composite Quadrature

What can we say about the error in the case of composite quadrature?

Error for one panel of length h: $\left|\int f - p_{n-1}\right| \leqslant C \cdot h^{n+1} \left\|f^{(n)}\right\|_\infty$

$$\begin{aligned}
&\left|\int_a^b f(x)\mathrm{d}x - \sum_{j=1}^m \sum_{i=1}^n \omega_{j,i} f(x_{j,i})\right| \\
\leqslant\ & C \left\|f^{(n)}\right\|_\infty \sum_{j=1}^m (a_{j+1} - a_j)^{n+1} \\
=\ & C \left\|f^{(n)}\right\|_\infty (a_{j+1} - a_j)^n \sum_{j=1}^m (a_{j+1} - a_j) \\
=\ & C \left\|f^{(n)}\right\|_\infty h^n (b - a),
\end{aligned}$$

where h is now the length of a single panel.

Composite Quadrature: Notes

Observation: Composite quadrature loses an order compared to non-composite.

Idea: If we can estimate errors on each subinterval, we can shrink (e.g. by splitting in half) only those contributing the most to the error. (adaptivity, $\rightarrow$ hw)

Taking Derivatives Numerically

Why *shouldn't* you take derivatives numerically?

- 'Unbounded'
  A function with small $\|f\|_\infty$ can have arbitrarily large $\|f'\|_\infty$
- Amplifies noise
  Imagine a smooth function perturbed by small, high-frequency wiggles
- Subject to cancellation error
- Inherently less accurate than integration
  - Interpolation: h^n
  - Quadrature: h^{n+1}
  - Differentiation: h^{n-1}
    (where n is the number of points)

Demo: Taking Derivatives with Vandermonde Matrices

Finite Differences

Idea: Start from definition of derivative. Called a forward difference.

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Q: What accuracy does this achieve?
Using Taylor:

$$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2} + \cdots$$

Plug in:

$$\frac{f(x) + f'(x)h + f''(x)\frac{h^2}{2} + \cdots - f(x)}{h} = f'(x) + O(h)$$

$\rightarrow$ first order accurate.

More Finite Difference Rules

Similarly:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$$

(Centered differences)

Can also take higher order derivatives:

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2)$$

Can find these by trying to match Taylor terms.
Alternative: Use linear algebra with interpolate-then-differentiate to find FD formulas.

Demo: Finite Differences vs Noise

Demo: Floating point vs Finite Differences

Richardson Extrapolation

If we have two estimates of something, can we get a third that's more accurate? Suppose we have an approximation $F = \tilde{F}(h) + O(h^p)$ and we know $\tilde{F}(h_1)$ *and* $\tilde{F}(h_2)$.

Grab one more term of the Taylor series: $F = \tilde{F}(h) + ah^p + O(h^q)$
Typically: $q = p + 1$ (but not necessarily).

Idea: Construct new approximation with the goal of $O(h^q)$ accuracy:

$$F = \alpha\tilde{F}(h_1) + \beta\tilde{F}(h_2) + O(h^q)$$

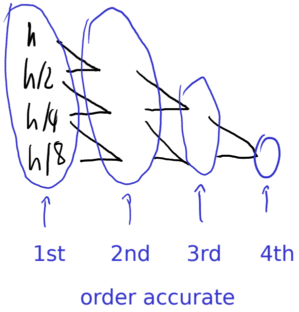
To get this, must have $\alpha a h_1^p + \beta a h_2^p = 0$. Also require $\alpha + \beta = 1$.

$$\begin{aligned} \alpha(h_1^p - h_2^p) + 1h_2^p &= 0 \\ \alpha &= \frac{-h_2^p}{h_1^p - h_2^p} \end{aligned}$$

Richardson Extrapolation: Observations, Romberg Integration

Important observation: Never needed to know a.

Idea: Can repeat this for even higher accuracy.



Carrying out this process for quadrature is called Romberg integration.

Demo: Richardson with Finite Differences

In-Class Activity: Differentiation and Quadrature

In-class activity: Differentiation and Quadrature

Outline

Introduction to Scientific Computing

Systems of Linear Equations

Linear Least Squares

Eigenvalue Problems

Nonlinear Equations

Optimization

Interpolation

Numerical Integration and Differentiation

Initial Value Problems for ODEs
- Existence, Uniqueness, Conditioning
- Numerical Methods (I)
- Accuracy and Stability
- Stiffness
- Numerical Methods (II)

Boundary Value Problems for ODEs

Partial Differential Equations and Sparse Linear Algebra

Fast Fourier Transform

Additional Topics

What can we solve already?

- Linear Systems: yes
- Nonlinear systems: yes
- Systems with derivatives: no

Some Applications

IVPs	BVPs
▶ Population dynamics $y_1' = y_1(\alpha_1 - \beta_1 y_2)$ (prey) $y_2' = y_2(-\alpha_2 + \beta_2 y_1)$ (predator) ▶ chemical reactions ▶ equations of motion	▶ bridge load ▶ pollutant concentration (steady state) ▶ temperature (steady state)

Initial Value Problems: Problem Statement

Want: Function $\mathrm{y} : [0, T] \to \mathbb{R}^n$ so that

- $\mathrm{y}^{(k)}(t) = \mathrm{f}(t, \mathrm{y}, \mathrm{y}', \mathrm{y}'', \dots, \mathrm{y}^{(k-1)})$ (*explicit*)

  or
- $\mathrm{f}(t, \mathrm{y}, \mathrm{y}', \mathrm{y}'', \dots, \mathrm{y}^{(k)}) = 0$ (*implicit*)

are called explicit/implicit *kth-order ordinary differential equations* (*ODEs*).
Give a simple example.

$$y'(t) = \alpha y$$

Not uniquely solvable on its own. What else is needed?

Initial conditions. (Q: How many?)

$$\mathrm{y}(0) = g_0, \quad \mathrm{y}'(0) = g_1, \dots \quad \mathrm{y}^{(k-1)}(0) = g_{k-1}.$$

Boundary Value Problems (BVPs) trade some derivatives for conditions at the ‘other end’.

Reducing ODEs to First-Order Form

A kth order ODE can always be reduced to first order. Do this in this example:

$$y''(t) = f(y)$$

In first-order form:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}'(t) = \begin{bmatrix} y_2(t) \\ f(y_1(t)) \end{bmatrix}$$

Because:

$$y_1''(t) = (y_1'(t))' = y_2'(t) = f(y_1(t)).$$

Properties of ODEs

What is a linear ODE?

$$\mathrm{f}(t, \mathrm{x}) = A(t)\mathrm{x} + \mathrm{b}$$

What is a linear and homogeneous ODE?

$$\mathrm{f}(t, \mathrm{x}) = A(t)\mathrm{x}$$

What is a constant-coefficient ODE?

$$\mathrm{f}(t, \mathrm{x}) = A\mathrm{x}$$

Properties of ODEs (II)

What is an autonomous ODE?

One in which the function f does not depend on time t.
An ODE can made autonomous by introducing an extra variable:

$$y_0'(t) = 1, \qquad y_0(0) = 0.$$

$\rightarrow$ Without loss of generality: Get rid of explicit t dependency.

Existence and Uniqueness

Consider the perturbed problem

$$\begin{cases} y'(t) = f(y) \\ y(t_0) = y_0 \end{cases} \qquad \begin{cases} \widehat{y}'(t) = f(\widehat{y}) \\ \widehat{y}(t_0) = \widehat{y}_0 \end{cases}$$

Then if f is *Lipschitz continuous* (has 'bounded slope'), i.e.

$$\|f(y) - f(\widehat{y})\| \leqslant L \|y - \widehat{y}\|$$

(where L is called the *Lipschitz constant*), then. . .

- there exists a solution y in a neighborhood of t_0, and. . .
- $\|y(t) - \widehat{y}(t)\| \leqslant e^{L(t-t_0)} \|y_0 - \widehat{y}_0\|$

What does this mean for uniqueness?

It *implies* uniqueness. If there were two separate solutions with identical initial values, they are not allowed to be different.

Conditioning

Unfortunate terminology accident: “Stability” in ODE-speak

To adapt to conventional terminology, we will use ‘Stability’ for

- the conditioning of the IVP, *and*
- the stability of the methods we cook up.

Some terminology:

An ODE is stable if and only if...

The solution is continously dependent on the initial condition, i.e.
For all $\varepsilon > 0$ there exists a $\delta > 0$ so that

$$\|\widehat{\mathrm{y}}_0 - \mathrm{y}_0\| < \delta \quad \Rightarrow \quad \|\widehat{\mathrm{y}}(t) - \mathrm{y}(t)\| < \varepsilon \quad \text{for all } t \geqslant t_0.$$

An ODE is asymptotically stable if and only if

$$\|\widehat{\mathrm{y}}(t) - \mathrm{y}(t)\| \to 0 \quad (t \to \infty).$$

Example I: Scalar, Constant-Coefficient

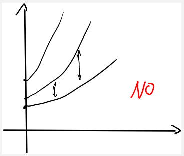
$$\begin{cases} y'(t) = \lambda y \\ y(0) = y_0 \end{cases} \quad \text{where} \quad \lambda = a + ib$$

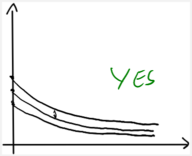
Solution?

$$y(t) = y_0 e^{\lambda t} = y_0(e^{at} \cdot e^{ibt})$$

When is this stable?

When $a = \operatorname{Re}\lambda > 0$: When $a = \operatorname{Re}\lambda \leqslant 0$:





Example II: Constant-Coefficient System

$$\begin{cases} \mathrm{y}'(t) = A\mathrm{y}(t) \\ \mathrm{y}(t_0) = \mathrm{y}_0 \end{cases}$$

Assume $V^{-1}\,\mathrm{A}V = D = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ diagonal.

How do we find a solution?

Define $\mathrm{w}(t) := V^{-1}\mathrm{y}(t)$. Then

$$\mathrm{w}'(t) = V^{-1}\mathrm{y}'(t) = V^{-1}A\mathrm{y}(t) = V^{-1}\,\mathrm{A}V\,\mathrm{w}(t) = D\mathrm{w}(t).$$

Now: *n decoupled* IVPs (with $\mathrm{w}_0 = V^{-1}\mathrm{y}_0$) $\rightarrow$ Solve as in scalar case.
Find $\mathrm{y}(t) = V\mathrm{w}(t)$.

When is this stable?

When Re λ < 0 for all eigenvalues λ

Euler's Method

Discretize the IVP

$$\begin{cases} y'(t) = f(y) \\ y(t_0) = y_0 \end{cases}$$

- Discrete times: $t_1, t_2, \ldots$, with $t_{i+1} = t_i + h$
- Discrete function values: $y_k \approx y(t_k)$.

Idea: Rewrite the IVP in integral form:

$$y(t) = y_0 + \int_{t_0}^{t} f(y(\tau))\mathrm{d}\tau,$$

then throw a simple quadrature rule at that. With the rectangle rule, we obtain Euler's method.

Euler's method: Forward and Backward

$$y(t) = y_0 + \int_{t_0}^{t} f(y(\tau))\mathrm{d}\tau,$$

Use 'left rectangle rule' on integral:

$$y_{k+1} = y_k + hf(y_k)$$

Time advancement requires *evaluating the RHS*. A method like that is called explicit. This method is called Forward Euler.

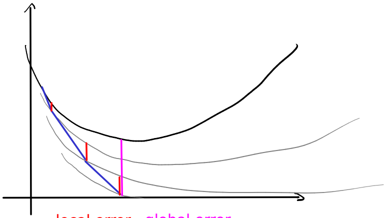
Use 'right rectangle rule' on integral:

$$y_{k+1} = y_k + hf(y_{k+1})$$

Time advancement requires *solving a system of equations*. A method like that is called implicit. This method is called Backward Euler.

Demo: Forward Euler stabilit

Global and Local Error



Let $u_k(t)$ be the function that solves the ODE with the initial condition $u_k(t_k) = y_k$.

Define the local error at step k as...

$$\ell_k = y_k - u_{k-1}(t_k)$$

Define the global error at step k as...

$$g_k = y(t_k) - y_k$$

About Local and Global Error

Is global error $= \sum$local errors?

No.
Consider an analogy with interest rates–at any given moment, you receive 5% interest ($\sim$ incur 5%error) on your current balance.
But your current balance *includes* prior interest (error from prior steps), which yields more interest (in turn contributes to the error).

This contribution to the error is called *propagated error*.
The local error is much easier to estimate $\rightarrow$ will focus on that.

A time integrator is said to be *accurate of order p* if. . .

$$\ell_k = O(h^{p+1})$$

ODE IVP Solvers: Order of Accuracy

A time integrator is said to be *accurate of order p* if $\ell_k = O(h^{p+1})$

This requirement is one order higher than one might expect–why?

A: To get to time 1, at least $1/h$ steps need to be taken, so that the global error is roughly

$$\underbrace{\frac{1}{h}}_{\#\text{steps}} \cdot O(h^{p+1}) = O(h^p).$$

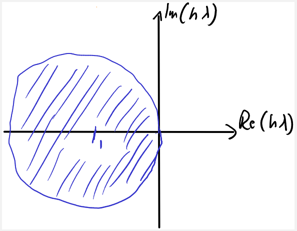
(Note that this ignores 'accrual' of propagated error.)

Stability of a Method

Find out when forward Euler is stable when applied to $y'(t) = \lambda y(t)$.

$$\begin{aligned} y_k &= y_{k-1} + h\lambda y_{k-1} \\ &= (1 + h\lambda) y_{k-1} \\ &= (1 + h\lambda)^k y_0 \end{aligned}$$

So: stable $\Leftrightarrow |1 + h\lambda| \leqslant 1$.
$|1 + h\lambda|$ is also called the amplification factor.
Gives rise to the stability region in the complex plane:



Stability: Systems

What about stability for systems, i.e.

$$\mathrm{y}'(t) = A\mathrm{y}(t)?$$

1. Diagonalize system as before
2. Notice that same V also diagonalizes the time stepper
3. apply scalar analysis to components.

$\rightarrow$ Stable if $|1 + h\lambda_i| \leqslant 1$ for all eigenvalues λ_i.

Stability: Nonlinear ODEs

What about stability for nonlinear systems, i.e.

$$\mathrm{y}'(t) = \mathrm{f}(\mathrm{y}(t))?$$

Consider perturbation $\mathrm{e}(t) = \mathrm{y}(t) - \widehat{\mathrm{y}}(t)$. Linearize:

$$\mathrm{e}'(t) = \mathrm{f}(\mathrm{y}(t)) - \mathrm{f}(\widehat{\mathrm{y}}(t)) \approx J_{\mathrm{f}}(\mathrm{y}(t))\mathrm{e}(t)$$

I.e. can (at least locally) apply analysis for linear systems to the nonlinear case.

Stability for Backward Euler

Find out when backward Euler is stable when applied to $y'(t) = \lambda y(t)$.

$$\begin{aligned} y_k &= y_{k-1} + h\lambda y_k \\ y_k(1 - h\lambda) &= y_{k-1} \\ y_k &= \frac{1}{1-h\lambda} y_{k-1} = \left(\frac{1}{1-h\lambda}\right)^k y_0. \end{aligned}$$

So: stable $\Leftrightarrow |1 - h\lambda| \geqslant 1$.

In particular: stable for any h if $\Re\lambda \leq 0$ ("unconditionally stable").

BE can be stable even when ODE is *unstable*. ($\operatorname{Re}\lambda > 0$). Accuracy?

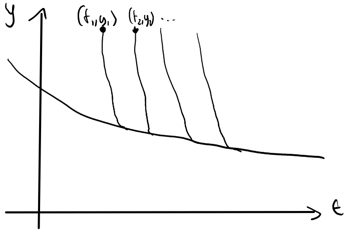
- *Explicit* methods: main concern in choosing h is *stability* (but *also* accuracy).
- *Implicit* methods: main concern in chosing h is *accuracy*.

Demo: Backward Euler stability

Stiff ODEs: Demo

Demo: Stiffness

‘Stiff’ ODEs



- Stiff problems have *multiple time scales*.
  (In the example above: Fast decay, slow evolution.)
- In the case of a stable ODE system

$$\mathrm{y}'(t) = \mathrm{f}(\mathrm{y}(t)),$$

stiffness can arise if J_{f} has eigenvalues of very different magnitude.

Stiffness: Observations

Why not just ‘small’ or ‘large’ magnitude?

Because the discrepancy between time scales is the root of the problem. If all time scales are similar, then time integration must simply ‘deal with’ that one time scale.
If there are two, then some (usually the fast ones) may be considered uninteresting.

What is the problem with applying explicit methods to stiff problems?

Fastest time scale governs time step $\rightarrow$ tiny time step $\rightarrow$ inefficient.

Stiffness vs. Methods

Phrase this as a conflict between accuracy and stability.

- Accuracy (here: capturing the slow time scale) *could* be achieved with large time steps.
- Stability (in explicit methods) demands a small time step.

Can an implicit method take arbitrarily large time steps?

In terms of stability: sure.
In terms of accuracy: no.

Predictor-Corrector Methods

Idea: Obtain intermediate result, improve it (with same or different method).

For example:

1. *Predict* with forward Euler: $\tilde{y}_{k+1} = y_k + hf(y_k)$
2. *Correct* with the trapezoidal rule: $y_{k+1} = y_k + \frac{h}{2}(f(y_k) + f(\tilde{y}_{k+1}))$.

This is called Heun's method.

Runge-Kutta/‘Single-step’/‘Multi-Stage’ Methods

Idea: Compute intermediate ‘stage values’:

$$\begin{aligned} r_1 &= f(t_k + c_1 h, y_k + (a_{11} \cdot r_1 + \cdots + a_{1s} \cdot r_s)h) \\ \vdots &\quad \vdots \\ r_s &= f(t_k + c_s h, y_k + (a_{s1} \cdot r_1 + \cdots + a_{ss} \cdot r_s)h) \end{aligned}$$

Then compute the new state from those:

$$y_{k+1} = y_k + (b_1 \cdot r_1 + \cdots + b_s \cdot r_s)h$$

Can summarize in a *Butcher tableau*:

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array}$$

Runge-Kutta: Properties

When is an RK method explicit?

If the diagonal entries in the Butcher tableau and everything above it are zero.

When is it implicit?

(Otherwise)

When is it *diagonally implicit*? (And what does that mean?)

If the everything above the diagonal entries in the Butcher tableau is zero.
This means that one can solve for one stage value at a time (and not multiple).

Heun and Butcher

Stuff Heun's method into a Butcher tableau:

1. $\tilde{y}_{k+1} = y_k + hf(y_k)$
2. $y_{k+1} = y_k + \frac{h}{2}(f(y_k) + f(\tilde{y}_{k+1}))$.

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

What is RK4?

(See Wikipedia page, note similarity to Simpson's rule.)

Demo: Dissipation in Runge-Kutta Methods

Multi-step/Single-stage/Adams Methods/Backward Differencing Formulas (BDFs)

Idea: Instead of computing stage values, use *history* (of either values of f or y–or both):

$$y_{k+1} = \sum_{i=1}^{M} \alpha_i y_{k+1-i} + h \sum_{i=1}^{N} \beta_i f(y_{k+1-i})$$

Extensions to implicit possible.
Method relies on existence of history. What if there isn't any? (Such as at the start of time integration?)

These methods are *not self-starting*.
Need another method to produce enough history.

Stability Regions

Why does the idea of stability regions still apply to more complex time integrators (e.g. RK?)

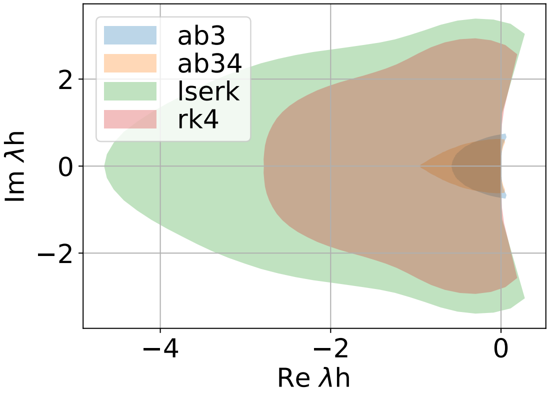
As long as the method doesn't "treat individual vector entries specially", a matrix that diagonalizes the ODE also diagonalizes the time integrator.
$\Rightarrow$ Can consider stability one eigenvalue at a time.

Demo: Stability regions

More Advanced Methods

Discuss:

- What is a good cost metric for time integrators?
- AB3 vs RK4
- Runge-Kutta-Chebyshev
- LSERK and AB34
- IMEX and multi-rate
- Parallel-in-time ("Parareal")



In-Class Activity: Initial Value Problems

In-class activity: Initial Value Problems

Outline

Introduction to Scientific Computing

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
- Existence, Uniqueness, Conditioning
- Numerical Methods

Partial Differential Equations and Sparse Linear Algebra

Fast Fourier Transform

Additional Topics

BVP Problem Setup: Second Order

Example: Second-order linear ODE

$$u''(x) + p(x)u'(x) + q(x)u(x) = r(x)$$

with *boundary conditions ('BCs')* at a:

- *Dirichlet* $u(a) = u_a$
- or *Neumann* $u'(a) = v_a$
- or *Robin* $\alpha u(a) + \beta u'(a) = w_a$

and the same choices for the BC at b.

Note: BVPs in time are rare in applications, hence x (not t) is typically used for the independent variable.

BVP Problem Setup: General Case

ODE:

$$\mathrm{y}'(x) = \mathrm{f}(\mathrm{y}(x)) \quad \mathrm{f} : \mathbb{R}^n \to \mathbb{R}^n$$

BCs:

$$\mathrm{g}(\mathrm{y}(a), \mathrm{y}(b)) = 0 \quad \mathrm{g} : \mathbb{R}^{2n} \to \mathbb{R}^n$$

(Recall the rewriting procedure to first-order for any-order ODEs.)

Does a first-order, scalar BVP make sense?

No–need second order (or $n \geqslant 2$) to allow two boundary conditions.

Example: Linear BCs

$$B_a \mathrm{y}(a) + B_b \mathrm{y}(b) = \mathrm{c}$$

Is this Dirichlet/Neumann/...?

Could be any–we're in the system case, and B_a and B_b are matrices–so conditions could be ony *any* component.

Does a solution even exist? How sensitive are they?

General case is harder than root finding, and we couldn't say much there.
$\rightarrow$ Only consider linear BVP.

$$(*)\begin{cases} \mathrm{y}'(x) = A(x)\mathrm{y}(x) + \mathrm{b}(x) \\ B_a\mathrm{y}(a) + B_b\mathrm{y}(b) = \mathrm{c} \end{cases}$$

To solve that, consider *homogeneous IVP*

$$\mathrm{y}_i'(x) = A(x)\mathrm{y}_i(x)$$

with initial condition

$$\mathrm{y}_i(a) = \mathrm{e}_i.$$

Note: $\mathrm{y} \neq \mathrm{y_i}$. e_i is the *i*th unit vector. With that, build the fundamental solution matrix

$$Y(x) = \begin{bmatrix} | & & | \\ \mathrm{y}_1 & \cdots & \mathrm{y}_n \\ | & & | \end{bmatrix}$$

ODE Systems: Existence

Let

$$Q := B_a Y(a) + B_b Y(b)$$

Then $(*)$ has a unique solution if and only if Q is invertible. Solve to find coefficients:

$$Q\alpha = \mathrm{c}$$

Then $Y(x)\alpha$ solves $(*)$ with $\mathrm{b}(x) = 0$.

Define $\Phi(x) := Y(x)Q^{-1}$. So $\Phi(x)\mathrm{c}$ solves $(*)$ with $\mathrm{b}(x) = 0$.
Define *Green's function*

$$G(x, y) := \begin{cases} \Phi(x)B_a\Phi(a)\Phi^{-1}(y) & y \leqslant x, \\ -\Phi(x)B_b\Phi(b)\Phi^{-1}(y) & y > x. \end{cases}$$

Then

$$\mathrm{y}(x) = \Phi(x)\mathrm{c} + \int_a^b G(x, y)\mathrm{b}(y)\mathrm{d}y.$$

Can verify that this solves $(*)$ by plug'n'chug.

ODE Systems: Conditioning

For perturbed problem with $\mathrm{b}(x) + \Delta\mathrm{b}(x)$ and $\mathrm{c} + \Delta\mathrm{c}$:

$$\|\Delta\mathrm{y}\|_\infty \leqslant \max\left(\|\Phi\|_\infty, \|G\|_\infty\right)\left(\|\Delta\mathrm{c}\|_1 + \int \|\Delta\mathrm{b}(y)\|_1 \,\mathrm{d}y\right).$$

- Did not prove uniqueness. (But true.)
- Also get continuous dependence on data.

Shooting Method

Idea: Want to make use of the fact that we can already solve IVPs.
Problem: Don't know *all* left BCs.

Demo: Shooting method

What about systems?

No problem–cannons are aimed in 2D as well. :)

What are some downsides of this method?

- Can fail
- Can be unstable even if ODE is stable

What's an alternative approach?

Set up a big linear system.

Finite Difference Method

Idea: Replace u' and u'' with finite differences.
For example: second-order centered

$$
\begin{aligned}
u'(x) &= \frac{u(x+h) - u(x-h)}{2h} + O(h^2) \\
u''(x) &= \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + O(h^2)
\end{aligned}
$$

Demo: Finite differences

What happens for a nonlinear ODE?

Get a nonlinear system$\rightarrow$Use Newton.

Demo: Sparse matrices

Collocation Method

$$(*)\begin{cases} y'(x) = f(y(x), \\ g(y(a), y(b)) = 0. \end{cases}$$

1. Pick a basis (for example: Chebyshev polynomials)

   $$\hat{y}(x) = \sum_{i=1}^{n} \alpha_i T_i(x)$$

   Want $\hat{y}$ to be close to solution y. So: plug into $(*)$.

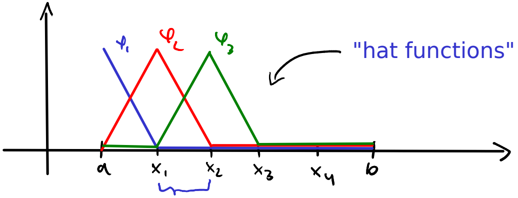
   Problem: $\hat{y}$ won't satisfy the ODE at all points at least.
   We do not have enough unknowns for that.

2. Idea: Pick n points where we would like $(*)$ to be satisfied.
   $\rightarrow$ Get a big (non-)linear system
3. Solve that (LU/Newton)$\rightarrow$ done.

Galerkin/Finite Element Method

$$u''(x) = f(x), \qquad u(a) = u(b) = 0.$$

Problem with collocation: Big dense matrix.
Idea: Use piecewise basis. Maybe it'll be sparse.



"hat functions"

one "finite element"

What's the problem with that?

u' does not exist. (at least at a few points where it's discontinuous)
u'' really does not exist.

Weak solutions/Weighted Residual Method

Idea: Enforce a 'weaker' version of the ODE.

Compute 'moments':

$$\int_a^b u''(x)\psi(x)\mathrm{d}x = \int_a^b f(x)\psi(x)\mathrm{d}x$$

Require that this holds for some *test functions* ψ from some set W. Now possible to get rid of (undefined) second derivative using integration by parts:

$$\int_a^b u''(x)\psi(x)\mathrm{d}x = [u'(x)\psi(x)]_a^b - \int_a^b u'(x)\psi'(x)\mathrm{d}x.$$

- Also called weighted residual methods.
- Can view collocation as a type of WR method with $\psi_j(x) = \delta(x - x_j)$

Galerkin: Choices in Weak Solutions

Make some choices:

- Solve for $u \in \operatorname{span}\{\text{hat functions } \varphi_i\}$
- Choose $\psi \in W = \operatorname{span}\{\text{hat functions } \varphi_i\}$ with $\psi(a) = \psi(b) = 0$.
  $\rightarrow$ Kills boundary term $[u'(x)\psi(x)]_a^b$.

These choices are called the Galerkin method. Also works with other bases.

Discrete Galerkin

Assemble a matrix for the Galerkin method.

$$
\begin{aligned}
-\int_a^b u'(x)\psi'(x)\mathrm{d}x &= \int_a^b f(x)\psi(x)\mathrm{d}x \\
-\int_a^b \left[\sum_{j=1}^{n} \alpha_j \varphi_j'(x)\right] \psi'(x)\mathrm{d}x &= \int_a^b f(x)\psi(x)\mathrm{d}x \\
-\sum_{j=1}^{n} \alpha_j \underbrace{\int_a^b \varphi_j'(x)\varphi_i'(x)\mathrm{d}x}_{S_{ij}} &= \underbrace{\int_a^b f(x)\varphi_i(x)\mathrm{d}x}_{r_i} \\
S\alpha &= \mathrm{r}.
\end{aligned}
$$

Now: Compute S, solve sparse (!) linear system.

Outline

Introduction to Scientific Computing

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
- Sparse Linear Algebra
- PDEs

Fast Fourier Transform

Additional Topics

Advertisement

Remark: Both PDEs and Large Scale Linear Algebra are big topics. Will only scratch the surface here. Want to know more?

- CS555 $\rightarrow$ Numerical Methods for PDEs
- CS556 $\rightarrow$ Iterative and Multigrid Methods
- CS554 $\rightarrow$ Parallel Numerical Algorithms

We would love to see you there! :)

Solving Sparse Linear Systems

Solving $A\mathsf{x} = \mathsf{b}$ has been our bread and butter.

Typical approach: Use factorization (like LU or Cholesky)
Why is this problematic?

Idea: Don't factorize, iterate.
Demo: Sparse Matrix Factorizations and "Fill-In"

‘Stationary’ Iterative Methods

Idea: Invert only part of the matrix in each iteration. Split

$$A = M - N,$$

where M is the part that we are actually inverting. Convergence?

$$\begin{aligned} A\mathsf{x} &= \mathsf{b} \\ M\mathsf{x} &= N\mathsf{x} + \mathsf{b} \\ M\mathsf{x}_{k+1} &= N\mathsf{x}_k + \mathsf{b} \\ \mathsf{x}_{k+1} &= M^{-1}(N\mathsf{x}_k + \mathsf{b}) \end{aligned}$$

- These methods are called *stationary* because they do the same thing in every iteration.
- They carry out fixed point iteration.
  $\to$ Converge if contractive, i.e. $\rho(M^{-1}N) < 1$.
- Choose M so that it's easy to invert.

Choices in Stationary Iterative Methods

What could we choose for M (so that it's easy to invert)?

Name	M	N
Jacobi	D	$-(L+U)$
Gauss-Seidel	$D+L$	$-U$
SOR	$\frac{1}{\omega}D+L$	$\left(\frac{1}{\omega}-1\right)D-U$

where L is the below-diagonal part of A, and U the above-diagonal.

Demo: Stationary Methods

Conjugate Gradient Method

Assume A is symmetric positive definite.

Idea: View solving $A\mathrm{x} = \mathrm{b}$ as an optimization problem.

$$\text{Minimize} \quad \varphi(\mathrm{x}) = \frac{1}{2}\mathrm{x}^T A\mathrm{x} - \mathrm{x}^T\mathrm{b} \quad \Leftrightarrow \quad \text{Solve} \quad A\mathrm{x} = \mathrm{b}.$$

Observe $-\nabla\varphi(\mathrm{x}) = \mathrm{b} - A\mathrm{x} = \mathrm{r}$ (residual).

Use an iterative procedure (s_k is the search direction):

$$\begin{aligned} \mathrm{x}_0 &= \langle \text{starting vector} \rangle \\ \mathrm{x}_{k+1} &= \mathrm{x}_k + \alpha_k \mathrm{s}_k, \end{aligned}$$

CG: Choosing the Step Size

What should we choose for α_k (assuming we know s_k)?

$$\begin{aligned} 0 &\overset{!}{=} \frac{\partial}{\partial\alpha}\varphi(x_k + \alpha_k s_k) \\ &= \nabla\varphi(x_{k+1}) \cdot s_k = r_{k+1} \cdot s_k. \end{aligned}$$

Learned: Choose α so that next residual is $\perp$ to current search direction.

$$\begin{aligned} r_{k+1} &= r_k + \alpha_k A s_k \\ 0 \overset{!}{=} s_k^T r_{k+1} &= s_k^T r_k + \alpha_k s_k^T A s_k \end{aligned}$$

Solve:

$$\alpha_k = \frac{s_k^T r_k}{s_k^T A s_k} = -\frac{s_k^T A e_k}{s_k^T A s_k}, \quad (*)$$

where $e_k = x_k - x^*$ and $r_k = -Ae_k$.

CG: Choosing the Search Direction

What should we choose for s_k?

Idea: $s_k = r_k = -\nabla\varphi(x_k)$, i.e. steepest descent. No–still a bad idea.

x, y are called *A-orthogonal* or *conjugate* if and only if $x^T A y = 0$.

Better Idea: Require $s_i^T A s_j = 0$ if $i \neq j$.

View error as linear combination of search directions, with some (thus far unknown) coefficients:

$$e_0 = x_0 - x^* = \sum_i \delta_i s_i.$$

- We run out of A-orthogonal directions after n iterations.
- Is the error going to be zero then? If $\delta_k = -\alpha_k$, then yes.

CG: Further Development

$$s_k^T A e_0 = \sum_i \delta_i s_k^T A s_i = \delta_k s_k^T A s_k.$$

Then

$$\delta_k = \frac{s_k^T A e_0}{s_k^T A s_k} = \frac{s_k A \left(e_0 + \sum_{i=1}^{k-1} \alpha_i s_i\right)}{s_k^T A s_k} = \frac{s_k^T A e_k}{s_k^T A s_k} = -\alpha_k.$$

How do we generate the s_k?

- Pick a random one to start with. Perhaps r_0?
- Generate next one by orthogonalizing from Krylov space procedure z, Az, A^2z
  Insight: Use three-term Lanczos iteration to generate. $\rightarrow$ cheap!

Demo: Conjugate Gradient Method

Introduction

Notation:

$$\frac{\partial}{\partial x} u \quad = \quad \partial_x u \quad = \quad u_x.$$

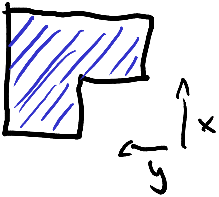
A *PDE* (*partial differential equation*) is an equation with multiple partial derivatives:

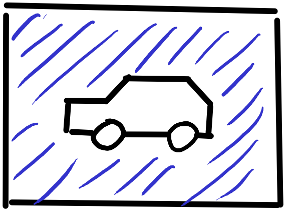
$$u_{xx} + u_{yy} = 0$$

Here: solution is a function $u(x, y)$ of two variables.

Examples: Wave propagation, fluid flow, heat diffusion

- Typical: Solve on domain with complicated geometry.



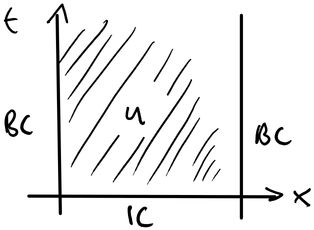


Initial and Boundary Conditions

- Sometimes one variable is time-like.
  What makes a variable time-like?
  - Causality
  - No geometry

Have:

- PDE
- Boundary conditions
- Initial conditions (in t)



Time-Dependent PDEs

Time-dependent PDEs give rise to a *steady-state* PDE:

$$u_t = f(u_x, u_y, u_{xx}, u_{yy}) \quad \to \quad 0 = f(u_x, u_y, u_{xx}, u_{yy})$$

Idea for time-dep problems (Method of Lines):

- Discretize spatial derivatives first
- Obtain large (semidiscrete) system of ODEs
- Use ODE solver from Chapter 9

Demo: Time-dependent PDEs

Notation: Laplacian

Laplacian (dimension-independent)

$$\Delta u = \operatorname{div}\operatorname{grad} u = \nabla \cdot (\nabla u) = u_{xx} + u_{yy}$$

Classifying PDEs

Three main types of PDEs:

- hyperbolic (wave-like, conserve energy)
  - first-order conservation laws: $u_t + f(u)_x = 0$
  - second-order wave equation: $u_{tt} = \Delta u$
- parabolic (heat-like, dissipate energy)
  - heat equation: $u_t = \Delta u$
- elliptic (steady-state, of heat and wave eq. for example)
  - Laplace equation $\Delta u = 0$
  - Poisson equation $\Delta u = f$
    (Pure BVP, similar to 1D BVPs, same methods apply–FD, Galerkin, etc.)

Outline

Introduction to Scientific Computing

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

Outline

Introduction to Scientific Computing

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