Numerical Analysis / Scientific Computing

CS450

Andreas Kloeckner

Spring 2019

Outline

What's the point of this class?

'*Scientific Computing*' describes a family of approaches to obtain approximate solutions to problems *once they've been stated mathematically*.

Name some applications:

- 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

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?

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

Define *norm*.

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

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

Norms: Examples

Examples of norms?

The so-called *p-norms*:

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

$p = 1, 2, \infty$ 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

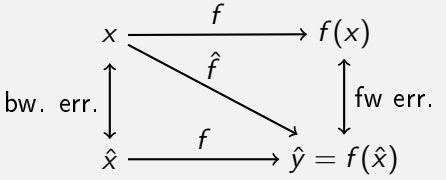
Suppose *want* to compute $y = f(x)$, but *approximate* $\hat{y} = \hat{f}(x)$.

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 $f(\hat{x}) = \hat{y}$.
- $\Delta x = \hat{x} - x$.



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:

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

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

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.

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

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 $b = (1.1010)_2 \cdot 2^0$.

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

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

Solving a Linear System

Given:

- $m \times n$ matrix A
- m-vector $\mathbf{b}$

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

Want: n-vector $\mathbf{x}$ so that

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

- Linear combination of columns of A to yield $\mathbf{b}$.
- 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

$$\|A\mathbf{x}\| \leqslant \|A\| \, \|\mathbf{x}\| \, .$$

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

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

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

Matrix Norm Properties

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

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

(not very difficult to show)
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?) (Can help to remember 1- and ∞-norm.)

Demo: Matrix norms
In-class activity: Matrix norms

Properties of Matrix Norms

Matrix norms inherit the vector norm properties:

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

But also some more properties that stem from our definition:

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

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

Conditioning

Now, let's study conditioning of solving a linear system $A\mathbf{x} = \mathbf{b}$.

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

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

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

Conditioning of Linear Systems: Observations

Showed $\kappa(\text{Solve } A\mathbf{x} = \mathbf{b}) \leq \left\|A^{-1}\right\| \|A\|$.
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:

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

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.

Demo: Condition number visualized
In-class activity: Matrix Conditioning
Demo: Conditioning of 2x2 Matrices

Residual Vector

What is the residual vector of solving the linear system

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

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

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

Residual and Error: Relationship

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

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

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

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

- 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.

$$A\mathbf{x} = \mathbf{b} \quad \rightarrow \quad A\widehat{\mathbf{x}} = \widehat{\mathbf{b}}.$$

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

$$A\mathbf{x} = \mathbf{b} \quad \rightarrow \quad \hat{A}\widehat{\mathbf{x}} = \mathbf{b}.$$

What can we say about the error now?

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

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

And we get

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

Changing Condition Numbers

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

Diagonal scaling is a simple strategy that sometimes helps.

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

What is this called as a general concept?

Preconditioning

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

In-class activity: Matrix Conditioning II

Solving Systems: Triangular matrices

Solve

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

- 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?

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

- 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

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

should be

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

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?

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

This is called LU factorization (or LU decomposition).

Does this help solve $A\mathbf{x} = \mathbf{b}$?

$$\begin{aligned} A\mathbf{x} &= \mathbf{b} \\ L\underbrace{U\mathbf{x}}_{\mathbf{y}} &= \mathbf{b} \\ L\mathbf{y} &= \mathbf{b} \quad \leftarrow \text{ solvable by fwd. subst.} \\ U\mathbf{x} &= \mathbf{y} \quad \leftarrow \text{ solvable by bwd. subst.} \end{aligned}$$

Now know $\mathbf{x}$ that solves $A\mathbf{x} = \mathbf{b}$.

LU: Failure Cases?

Is LU/Gaussian Elimination bulletproof?

No, very much not:

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

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

$$\begin{pmatrix} 1 & \\ \ell_{21} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} \\ & u_{22} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 1 \end{pmatrix} \quad \rightarrow \quad u_{11} = 0$$

$$\underbrace{u_{11} \cdot \ell_{21}}_{0} + 1 \cdot 0 = 2$$

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*.

Fixing nonexistence of LU

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

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

P is called a *permutation matrix*.

Q: What's P^{-1}?

What does this process look like then?

$P_1 A$	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

What about the L in LU?

Sort out what LU with pivoting looks like.

Have: $M_3 P_3 M_2 P_2 M_1 P_1 A = U$

Define: $L_3 = M_3$
Define: $L_2 = P_3 M_2 P_3^{-1}$
Define: $L_1 = P_3 P_2 M_1 P_2^{-1} P_3^{-1}$

Then

$$\begin{aligned} & L_3 L_2 L_1 P_3 P_2 P_1 \\ = \; & M_3 (P_3 M_2 P_3^{-1})(P_3 P_2 M_1 P_2^{-1} P_3^{-1}) P_3 P_2 P_1 \\ = \; & M_3 P_3 M_2 P_2 M_1 P_1 \quad (!) \end{aligned}$$

Summing up:

$$P_3 P_2 P_1 A = L_1^{-1} L_2^{-1} L_3^{-1} U$$

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

In-class activity: Pivoting and Cost

More cost concerns

What's the cost of solving $A\mathbf{x} = \mathbf{b}$?

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

What's the cost of solving $A\mathbf{x} = \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$?

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.

Getting that constant to be small is surprisingly hard, even for something deceptively simple such as matrix-matrix multiplication.

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

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

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*.

Changing matrices

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

In general: not able to do much but recompute.

But: Special cases allow something to be done.

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

(a so-called *rank-one update*)

The *Sherman-Morrison formula* gives us

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

FYI: There is a rank-k analog called the *Sherman-Morrison-Woodbury formula.*

Demo: Sherman-Morrison

Outline

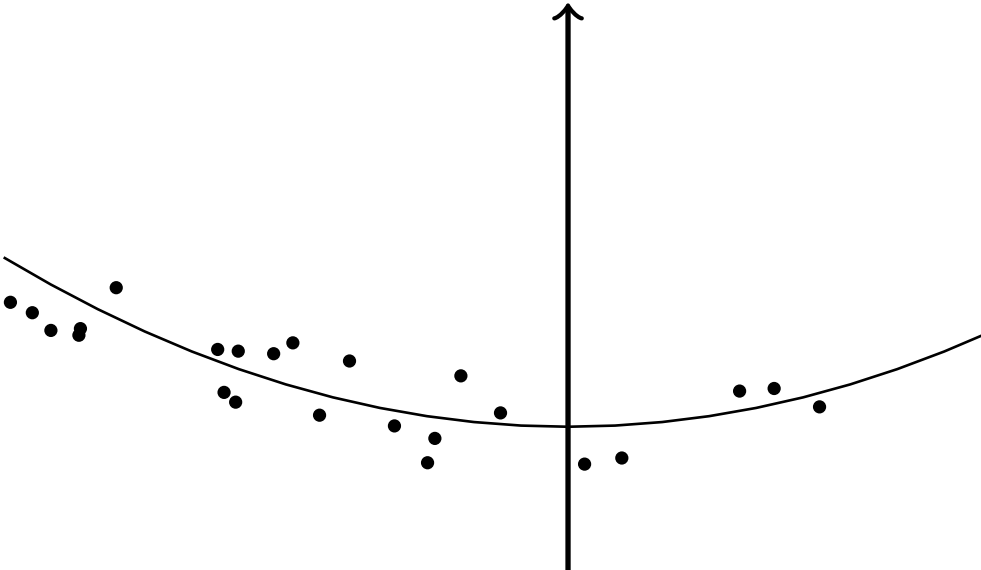
What about non-square systems?

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



Properties of Least-Squares

Consider the least squares problem $A\mathbf{x} \cong \mathbf{b}$ and its associated *objective function*

$$\varphi(x) = \|\mathbf{b} - A\mathbf{x}\|_2^2 .$$

Is there always a solution to a linear least-squares problem

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

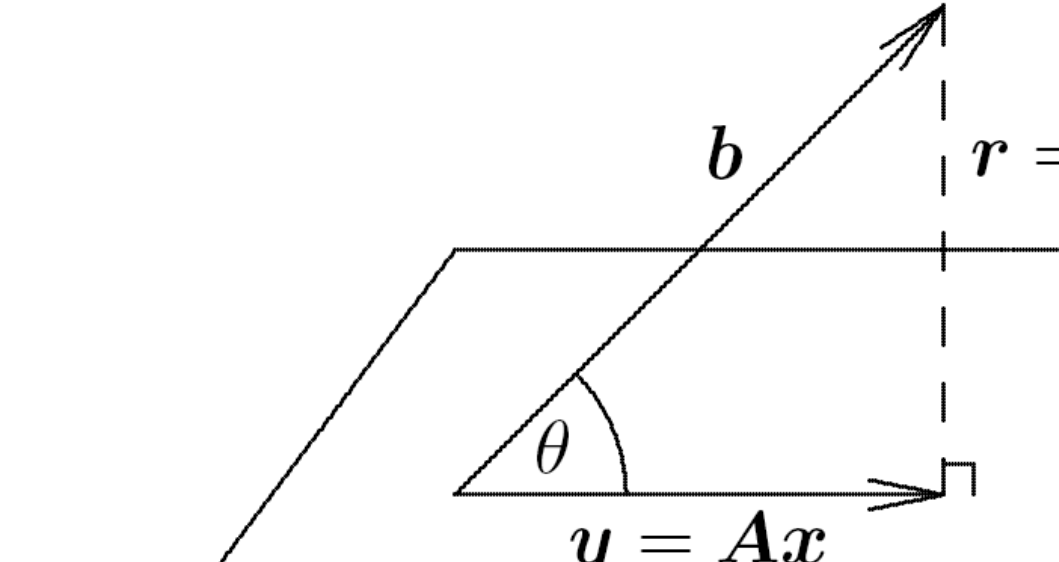
Is it unique?

No, for example if A has a nullspace.

Examine the objective function, find its minimum.

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

Least Squares, Viewed Geometrically



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 $\operatorname{span}\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_\ell\}$?

First define

$$Q = \begin{pmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_\ell \end{pmatrix}.$$

Then

$$QQ^T$$

will project and is obviously symmetric.

Pseudoinverse

What is the *pseudoinverse* of A?

Nonsquare $m \times n$ matrix A has no inverse in usual sense.
If $\text{rank}(A) = n$, *pseudoinverse* is

$$A^+ = (A^T A)^{-1} A^T.$$

(colspan-projector with final A missing)

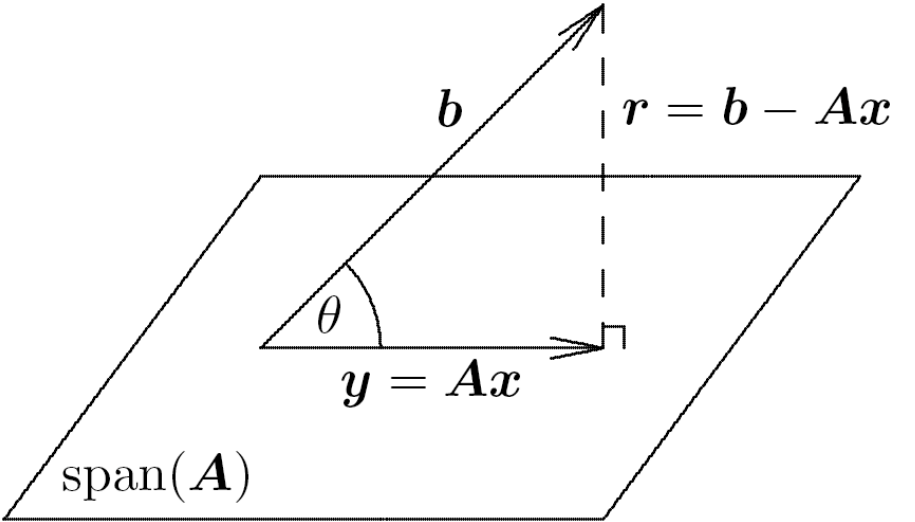
What can we say about the condition number in the case of a tall-and-skinny, full-rank matrix?

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

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

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

Sensitivity and Conditioning of Least Squares



How sensitive is it?

Ideas for Solving Least Squares

Tell me about the *augmented system method*.

Idea: Solve for residual *and* solution.

$$\begin{aligned} \mathbf{r} - A\mathbf{x} &= \mathbf{b} \\ A^T\mathbf{r} &= \mathbf{0} \end{aligned}$$

Bad:

- Not spd
- Still poorly conditioned
- $4\times$ the storage, including two copies of A.

Least-squares by Transformation

Want a matrix Q so that

$$QA\mathbf{x} \cong Q\mathbf{b}$$

has the same solution as

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

I.e. want

$$\|Q(A\mathbf{x} - \mathbf{b})\|_2 = \|A\mathbf{x} - \mathbf{b}\|_2 .$$

What type of matrix does that? Any invertible one?

No. (Think diagonal matrix with non-const diagonal entries.)

But orthogonal matrices do.

Orthogonal Matrices

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

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

Are orthogonal projectors orthogonal?

Nope, not in general.

Now what about that norm property?

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

Simpler Problems: Triangular

Would we win anything from transforming a least-squares system to upper triangular form?

$$\begin{pmatrix} R \\ 0 \end{pmatrix} \mathbf{x} \cong \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}$$

If so, how would we minimize the residual norm?

$$\|\mathbf{r}\|_2^2 = \|\mathbf{b}_1 - R\mathbf{x}\|_2^2 + \|\mathbf{b}_2\|_2^2 .$$

R is invertible, so we can find $\mathbf{x}$ so that $\|\mathbf{b}_1 - R\mathbf{x}\|_2^2 = 0$, i.e.

$$\|\mathbf{r}\|_2^2 = \|\mathbf{b}_2\|_2^2 .$$

So the goal becomes: Find orthogonal matrix Q so that $Q^T A = R$, i.e. $A = QR$. This will be called a *QR factorization*.

Computing QR

- Gram-Schmidt
- Householder Reflectors
- Givens Rotations

Latter two similar to LU:

- Successively zero out below-diagonal part
- But: using orthogonal matrices

Demo: Gram-Schmidt–The Movie
Demo: Gram-Schmidt and Modified Gram-Schmidt
Demo: Keeping track of coefficients in Gram-Schmidt
In-class activity: QR

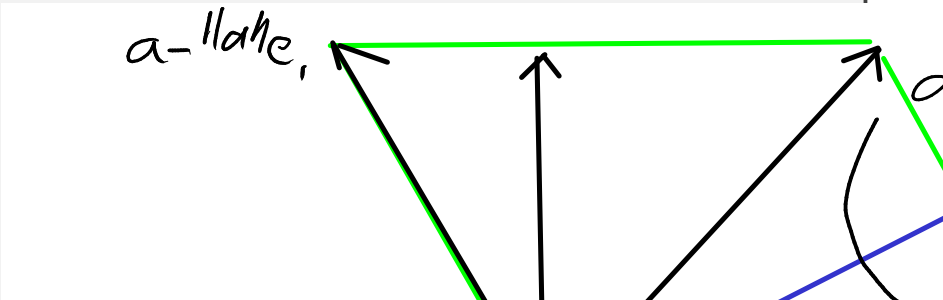
Householder Transformations

Find an *orthogonal* matrix Q that zeroes out the lower part of a column vector $\mathbf{a}$.

Not too many things can be done with an orthogonal matrix:

- Rotate
- Reflect

Reflection turns out to to do the job:



Givens Rotations

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

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

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?

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

(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 $\rightarrow$ 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*.
- $\Sigma = \mathrm{diag}(\sigma_i)$ 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?

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

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

- *Numerical rank*:

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

- *Low-rank Approximation*

Theorem
(Eckart-Young) If $k < r = \text{rank}(A)$ *and*

$$\sum_{i=1}^{k}$$

Comparing the Methods

Multiplications 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

Eigenvalue Problems: Setup/Math Recap

A is an $n \times n$ matrix.

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

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

- 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?

Linear Algebra approach:

$$\begin{aligned} & A\mathbf{x} = \lambda\mathbf{x} \\ \Leftrightarrow \quad & (A - \lambda I)\mathbf{x} = 0 \\ \Leftrightarrow \quad & A - \lambda I \text{singular} \\ \Leftrightarrow \quad & \det(A - \lambda I) = 0 \end{aligned}$$

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

Q: Does that help algorithmically?
A: Abel showed that for $n \geqslant 5$ there is no general formula for the roots of the polynomial. (i.e. no analog to the quadratic formula for $n = 5$)

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

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

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

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

$\Rightarrow x + y = x \Rightarrow y = 0$. 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 $(\mathbf{x}_n)_{i=1}^n$.

In that case, let

$$X = \begin{pmatrix} z| & & | \\ \mathbf{x}_1 & \cdots & \mathbf{x}_n \\ z| & & | \end{pmatrix},$$

and observe

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

where D is a diagonal matrix with the eigenvalues.

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?)

Sensitivity

Assume A not defective. Suppose $X^{-1}AX = D$. Perturb

$$A \to A + E$$

What happens to the eigenvalues?

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

Observations:

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

Suppose $\mathbf{v}$ is a perturbed eigenvector.

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

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

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

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

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

$$A^{-1}\mathbf{x} = \lambda^{-1}\mathbf{x}$$

Power. $A \to A^k$

$$A^k\mathbf{x} = \lambda^k\mathbf{x}$$

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

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

\

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

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

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

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*.

Also, if we knew how to compute this, how would it help us find eigenvalues?

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

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

In matrix form

$$\begin{pmatrix} \lambda & * & * & * & * \\ 0 & & & & \\ 0 & & & & \\ 0 & & & & \\ 0 & & & & \end{pmatrix}$$

Power Iteration

What are the eigenvalues of A^{1000}?

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

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

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

Or

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

Idea: Use this as a computational procedure to find $\mathbf{x}_1$.
Called *Power Iteration*.

Power Iteration: Issues?

What could go wrong with Power Iteration?

- Starting vector has no component along $\mathbf{x}_1$
  Not a problem in practice: Rounding will introduce one.
- Overflow in computing λ_1^{1000}
  $\rightarrow$ *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:

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

- $= \lambda$ if $\mathbf{x}$ 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 $\mathbf{v}_1^{(k)}$ is the kth estimate of the eigenvector $\mathbf{x}_1$, and

$$e_k = \left\| \mathbf{x}_1 - \mathbf{v}_1^{(k)} \right\|.$$

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*.

$$\mathbf{x}_{k+1} := (A - \sigma)^{-1}\mathbf{x}_k$$

- 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 $\sigma_k = \mathbf{x}_k^T A \mathbf{x}_k / \mathbf{x}_k^T \mathbf{x}_k$ to be the Rayleigh quotient for $\mathbf{x}_k$.

$$\mathbf{x}_{k+1} := (A - \sigma_k I)^{-1}\mathbf{x}_k$$

Demo: Power Iteration and its Variants

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

$$\begin{pmatrix} \lambda_1 & * \\ & B \end{pmatrix},$$

   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 $\mathbf{x}_1$

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. $X_{k+1} = A Q_k$

Good: X_k converge to X with eigenvectors in columns
Bad:

- Slow/linear convergence
- Expensive iteration

Tracing through:

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

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?

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

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*

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

- 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.

Overall procedure:

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

Krylov space methods: Intro

What subspaces can we use to look for eigenvectors?

QR:

$$\text{span}\{A^\ell \mathbf{y}_1, A^\ell \mathbf{y}_2, \ldots, A^\ell \mathbf{y}_k\}$$

Krylov:

$$\text{span}\{\underbrace{\mathbf{x}}_{\mathbf{x}_0}, A\mathbf{x}, \ldots, \underbrace{A^{k-1}\mathbf{x}}_{\mathbf{x}_{k-1}}\}$$

Define matrix

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

Then

$$AK = \begin{pmatrix} z| & & |\\ \end{pmatrix}$$

Conditioning in Krylov Space Methods/Arnoldi

Iteration

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

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

Realizing that

- C_n is upper Hessenberg

Krylov: What about eigenvalues?

How can we use Arnoldi/Lancos to compute eigenvalues?

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

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

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

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)

Outline

Solving Nonlinear Equations

What is the goal here?

Solve $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ 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 $J_{\mathbf{f}}$, works only locally)
- *Contraction mapping theorem*
  A function $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^n$ is called *contractive* if there exists a $0 < \gamma < 1$ so that

$$\|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{y})\| \leqslant \gamma \|\mathbf{x} - \mathbf{y}\| .$$

  A fixed point of $\mathbf{g}$ is a point where $\mathbf{g}(\mathbf{x}) = \mathbf{x}$.

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

Sensitivity and Multiplicity

What is the sensitivity/conditioning of root finding?

cond (root finding) = cond (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?

The inverse function is steep near one, so conditioning is poor

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

Let $\mathbf{e}_k = \widehat{\mathbf{u}}_k - \mathbf{u}$ be the error in the kth iterate $\widehat{\mathbf{u}}_k$.

An iterative method *converges with rate* r if

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

$r = 1$ 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:

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

  (and $C < 1$ matters!)

- Quadratic convergence

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

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

Convergence Rates: Understanding the Definition

Contrast the following 'alternate' definitions of convergence rate:

$$(1) \quad \frac{\|\mathbf{e}_{k+1}\|}{\|\mathbf{e}_k\|^r} \leqslant C \begin{cases} > 0, \\ < \infty. \end{cases}$$

$$(2) \quad 0 < C_{\text{low}} \leqslant \frac{\|\mathbf{e}_{k+1}\|}{\|\mathbf{e}_k\|^r} \leqslant C_{\text{high}}$$

$$(3) \quad \lim_{k\to\infty} \frac{\|\mathbf{e}_{k+1}\|}{\|\mathbf{e}_k\|^r} = C \begin{cases} > 0, \\ < \infty. \end{cases}$$

- (1) Is true for $r = 1$ even if the process converges faster
- (2) Makes strong promises about *pre-asymptotic behavior* (and only weak promises about asymptotic behavior)
- (3) is actually the most informative about what happens 'eventually' (i.e. asymptotically), and it does not restrict pre-asymptotic beahvior.

Stopping Criteria

Here are some ideas for stopping criteria:

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

Comment on them. Are any of them 'foolproof'?

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:

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

Mean value theorem says: There is a θ_k between $\mathbf{x}_k$ and $\mathbf{x}^*$ so that

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

So:

Newton's Method

Derive Newton's method.

Idea: Approximate f at current iterate 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:

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

So

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

Then let's look at

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

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

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)

'Trusting' Newton and Secant

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

- Fix a region where they're 'trustworthy'.
- Limit step size to that region.

Such a method is called a *trust region method*.

Fixed Point Iteration

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

When does this converge?

Converges (locally) if $\rho(J_{\mathbf{g}}(\mathbf{x}^*)) < 1$, where the *Jacobian matrix*

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

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

Newton's method

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

Approximate by linear:

$$\mathbf{f}(\mathbf{x}+\mathbf{s}) = \mathbf{f}(\mathbf{x}) + J_{\mathbf{f}}(\mathbf{x})\mathbf{s}$$

Set to 0:

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

So

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

Downsides:

- Still only locally convergent

Secant in n dimensions?

What would the secant method look like in n dimensions?

Need an 'approximate Jacobian' satisfying

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

Suppose we have *already taken* a step to $\mathbf{x}_{k+1}$. 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

Optimization

State the problem.

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

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

- $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ and $\mathbf{h}(\mathbf{x}) \leqslant \mathbf{0}$ are called *constraints*.
  They define the set of *feasible points* $\mathbf{x} \in S \subseteq \mathbb{R}^n$.
- If $\mathbf{g}$ or $\mathbf{h}$ are present, this is *constrained optimization*.
  Otherwise *unconstrained optimization*.
- If $\mathbf{f}$, $\mathbf{g}$, $\mathbf{h}$ are *linear*, this is called *linear programming*.
  Otherwise *nonlinear programming*.
- Q: What if we are looking for a *maximizer*?
- Examples:
  - What is the fastest/cheapest/shortest... way to do...?

Existence/Uniqueness

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

- Terminology: *global minimum///local minimum*
- If $f : S \to \mathbb{R}$ is continuous on a closed and bounded $S \subseteq \mathbb{R}^n$, then a minimum exists.
- $f : S \to \mathbb{R}$ is called *coercive* on $S \subseteq \mathbb{R}^n$ (which must be unbounded) if

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

  If f is coercive, a global minimum exists (but is possibly non-unique).
- $S \subseteq \mathbb{R}^n$ is called *convex* if for all $\mathbf{x}, \mathbf{y} \in S$ and all $0 \leqslant \alpha \leqslant 1$

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

- $f : S \to \mathbb{R}$ is called *convex* on $S \subseteq \mathbb{R}^n$ if for $\setminus$ $\mathbf{x}, \mathbf{y} \in S$ and all

Optimality Conditions

If we have found a candidate $\mathbf{x}^*$ for a minimum, how do we know it actually is one?

To make this doable, assume f is smooth–i.e. has as many derivatives as needed.

- 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: $\nabla f(\mathbf{x}^*) = 0$ (i.e. $\mathbf{x}^*$ is an extremal point)
  - Sufficient: $\nabla f(\mathbf{x}^*) = 0$ and $H_f(\mathbf{x}^*)$ positive definite (implies x^* is a local minimum)

where the *Hessian*

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

Sensitivity and Conditioning

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

- In one dimension: Suppose we still have

$$|f(\tilde{x}) - f(x^*)| < \text{tol}$$

(where x^* is the true minimum) Apply Taylor's theorem to get an idea:

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

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

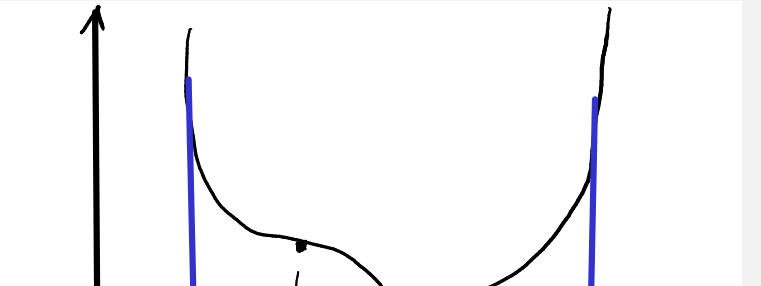
Golden Section Search

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 $x_1 < x_2$

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

Suppose we have an interval with f unimodal:



Newton's Method

Reuse the Taylor approximation idea, but for optimization.

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

Remark: Line (i.e. order 1 Taylor) wouldn't suffice–no minimum.

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

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

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)

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

Newton's method (n D)

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

Build a Taylor approximation:

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

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

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

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

Drawbacks: (from demo)

- Need second (!) derivatives
  (addressed by Conjugate Gradients, later in the class)

Nonlinear Least Squares/Gauss-Newton

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

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

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:

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

For brevity: $J := J_{\mathbf{r}}(\mathbf{x})$. Can show similarly:

Constrained Optimization: Problem Setup

Want $\mathbf{x}^*$ so that

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

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

Unconstrained necessary condition:

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

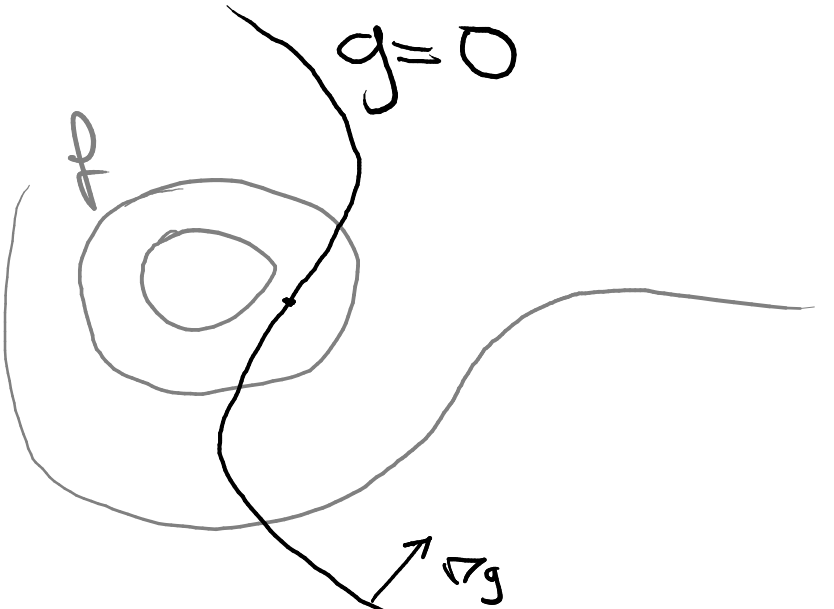
Problem: That alone is not helpful. 'Downhill' direction has to be *feasible*. So just this doesn't help.

$\mathbf{s}$ is a *feasible direction* at $\mathbf{x}$ if

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

Necessary for minimum:

Lagrange Multipliers



Inequality-Constrained Optimization

Want $\mathbf{x}^*$ so that

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

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

Define Lagrangian:

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

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

$$h_i(\mathbf{x}^*)\lambda_{2,i} = 0.$$

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

Outline

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

Does this problem have a unique answer?

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

Making the Interpolation Problem Unique

How can we cut down the set of possible answers to *exactly one*?{}

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:

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

Want unique answer: Pick $N_{\text{func}} = N \to V$ square.{}

V is called the *(generalized) Vandermonde matrix*.

Main lesson:

Modes and Nodes (aka Functions and Points)

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

Ideas for 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)

But first: Why *not* monomials on equispaced points?

Demo: Monomial 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

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

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!

How can those be orthogonal? Just need something like a dot product!

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

Another family of orthogonal polynomials: Chebyshev

Three equivalent definitions:

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

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

(Like for Legendre, you won't exactly get the standard normalization if you do this.)

- $T_k(x) = \cos(k \cos^{-1}(x))$
- $T_k(x) = 2xT_k(x) - T_{k-1}(x)$

Demo: Chebyshev Interpolation (Part 1)

What is the Vandermonde matrix for Chebyshev polynomials?{}

> - Need to know the nodes to answer that
> - The answer would be particularly simple if the nodes were $\cos(*)$.

Chebyshev nodes

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

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

The 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)

Summary?

- Chebyshev interpolation is fast and works extremely well

Error Result

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

Why does Chebyshev-like 'bunching' work?

- 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.

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 $h = x_n - x_1$.

Going piecewise: Simplest Case

Construct a piecewise linear interpolant at four points.

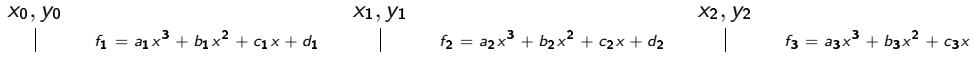
x_0, y_0	x_1, y_1	x_2, y_2	x_3, y_3
$f_1 = a_1 x + b_1$	$f_2 = a_2 x + b_2$	$f_3 = a_3 x + b_3$	
2 unk.	2 unk.	2 unk.	
$f_1(x_0) = y_0$	$f_2(x_1) = y_1$	$f_3(x_2) = y_2$	
$f_1(x_1) = y_1$	$f_2(x_2) = y_2$	$f_3(x_3) = y_3$	
2 eqn.	2 eqn.	2 eqn.	

Why three intervals?

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

Piecewise Cubic ('Splines')

Now do the same accounting for piecewise cubic at four points:



4 unknowns 4 unknowns 4 unknowns

$f_1(x_0) = y_0 \quad f_2(x_1) = y_1 \quad f_3(x_2) = y_2$

$f_1(x_1) = y_1 \quad f_2(x_2) = y_2 \quad f_3(x_3) = y_3$

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

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

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

Not enough: need yet more conditions.

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

Now: have a square system.{}

Observation: Number of conditions:

$$2N_{\text{intervals}} + 2N_{\text{middle nodes}} + 2$$

where

Outline

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*.

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

Examples and Exactness

To what polynomial degree are the following rules exact?

Midpoint rule
$(b-a)f\left(\frac{a+b}{2}\right)$

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?

Notation: $\|f\|_\infty = \max_{x\in[a,b]} |f(x)|$

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

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

$$\begin{aligned} & \left| \sum_i \omega_i f(x_i) - \sum_i \omega_i \hat{f}(x_i) \right| \\ = & \left| \sum_i \omega_i e(x_i) \right| \leqslant \sum_i |\omega_i e(x_i)| \\ \leqslant & \left(\sum_i |\omega_i| \right) \|e\|_\infty \end{aligned}$$

About Newton-Cotes

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

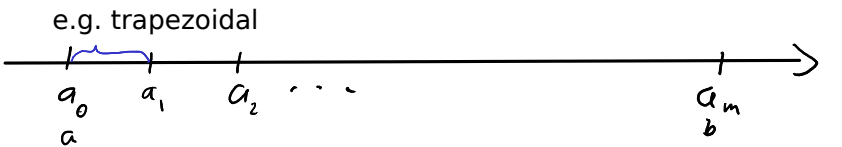
Demo: Newton-Cotes weight finder (again, with many nodes)
In fact, Newton-Cotes must have at least one negative weight as soon as $n \geqslant 11$.

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.

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.



What can we say about the error in this case?

Recall: error for a 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.

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

So far: nodes chosen from outside.
Can we gain something if we let the quadrature rule choose the weights, 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

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

If you *absolutely* have to take num. derivatives, what could you do?

- Compute interpolation coefficients, differentiate basis
- '/Finite Differences/'

Idea: Start from definition of derivative.

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

Called a *forward difference*.

Q: What would a backward difference look like?

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:

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 $\alpha + \beta = 1$.

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

Demo: Richardson with Finite Differences

Outline

What can we solve?

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

Some Applications

ODEs	IVPs
▶ 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 $\mathbf{y} : [0, T] \to \mathbb{R}^n$ so that

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

  or
- $\mathbf{f}(t, \mathbf{y}, \mathbf{y}', \mathbf{y}'', \ldots, \mathbf{y}^{(k)}) = \mathbf{0}$ (*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?)

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

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:

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

Because:

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

Properties of ODEs

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.

What is a *linear* ODE?

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

What is a *linear and homogeneous* ODE?

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

What is a *constant-coefficient* ODE?

Existence and Uniqueness

Consider the perturbed problem

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

Then if $\mathbf{f}$ is *Lipschitz continuous* (has 'bounded slope'), i.e.

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

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

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

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

An ODE is *asymptotically stable* if and only if

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

Example I: Scalar, Constant-Coefficient

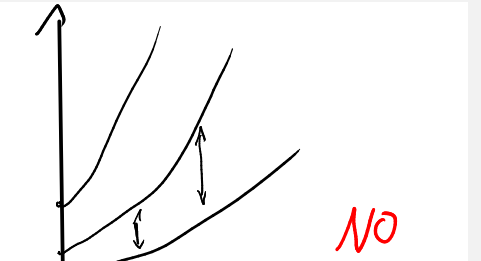
$$\begin{cases} y'(t) = \lambda y \\ y(0) = y_0 \end{cases} \quad \text{where} \quad \lambda = a + ib$$

Solution?

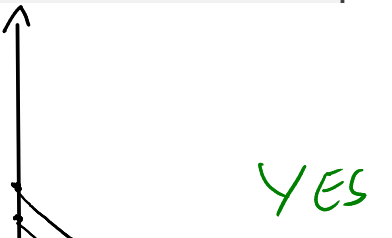
$$y(t) = y_0 e^{\lambda t} = y_0(e^{at} \cdot e^{ibt})$$

When is this stable?

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



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



Example II: Constant-Coefficient System

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

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

How do we find a solution?

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

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

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

When is this stable?

When $\operatorname{Re}\lambda_i \leqslant 0$ for all eigenvalues λ_i.

Euler's Method

Discretize the IVP

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

- Discrete times: $t_1, t_2, \ldots$, with $t_{i+1} = t_i + h$
- Discrete function values: $\mathbf{y}_k \approx \mathbf{y}(t_k)$.

Idea: Rewrite the IVP in integral form:

$$\mathbf{y}(t) = \mathbf{y}_0 + \int_{t_0}^{t} \mathbf{f}(\mathbf{y}(\tau))\mathrm{d}\tau,$$

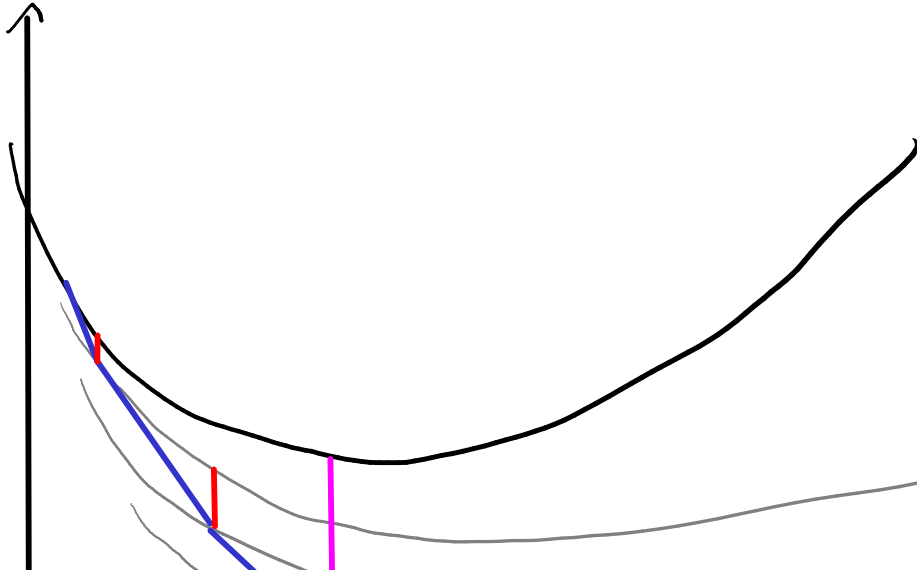
then throw a simple quadrature rule at that. With the rectangle rule, we obtain *Euler's method*.

- Using 'left rectangle rule':

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h\mathbf{f}(\mathbf{y}_k)$$

Time advancement can be computed simply by {evaluating the

Global and Local Error



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...

$\ell_k = O(h^{p+1})$

Q: This 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

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, Nonlinear ODEs

What about stability for systems, i.e.

$$\mathbf{y}'(t) = A\mathbf{y}(t)?$$

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.

What about stability for nonlinear systems, i.e.

$$\mathbf{y}'(t) = \mathbf{f}(\mathbf{y}(t))?$$

Consider perturbation $\mathbf{e}(t) = \mathbf{y}(t) - \widehat{\mathbf{y}}(t)$. Linearize:

$$\mathbf{e}'(t) = \mathbf{f}(\mathbf{y}(t)) - \mathbf{f}(\widehat{\mathbf{y}}(t)) \approx J_{\mathbf{f}}(\mathbf{y}(t))\mathbf{e}(t)$$

I.e. can (at least locally) apply analysis for linear systems to the

Stability for Backward Euler

Find out when backward Euler is stable when applied to

$$y'(t) = \lambda y(t).$$

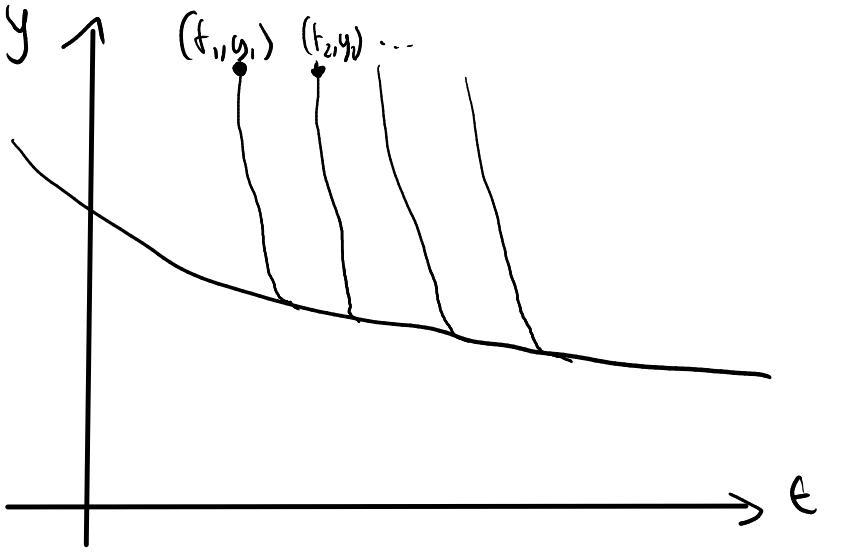
$$\begin{aligned} y_k &= y_{k-1} + h\lambda y_k \\ y_k(1 - h\lambda) &= y_{k-1} \\ y_k &= \frac{1}{1 - h\lambda} y_{k-1} \\ &= \left(\frac{1}{1 - h\lambda}\right)^k y_0. \end{aligned}$$

So: stable $\Leftrightarrow |1 - h\lambda| \geqslant 1$.

Strangely, backward Euler can be stable even when the ODE is *unstable*.
(i.e. for $\operatorname{Re}\lambda > 0$) But it won't (can't!) be accurate.

Demo: Stiffness

'Stiff' ODEs



- Stiff problems have *multiple time scales*.

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':

$$\begin{aligned} r_1 &= f(t_k + c_1 h, y_k + (a_{11} \cdot r_1 + \cdots + a_{1s} \cdot r_s)h) \\ &\vdots \quad\quad \vdots \\ r_s &= f(t_k + c_s h, y_k + (a_{s1} \cdot r_1 + \cdots + a_{ss} \cdot r_s)h) \end{aligned}$$

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*:

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array}$$

When is an RK method explicit?

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):

$$y_{k+1} = \sum_{i=1}^{M} \alpha_i y_{k+1-i} + h \sum_{i=1}^{N} \beta_i f(y_{k+1-i})$$

(one of these $\rightarrow$ hw) 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.

Demo: Stability regions

Outline

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:

$$\mathbf{y}'(x) = \mathbf{f}(\mathbf{y}(x)) \quad \mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$$

BCs:

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = \mathbf{0} \quad \mathbf{g} : \mathbb{R}^{2n} \to \mathbb{R}^n$$

(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

$$B_a\mathbf{y}(a) + B_b\mathbf{y}(b) = \mathbf{c}$$

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.

$$(*) \begin{cases} \mathbf{y}'(x) = A(x)\mathbf{y}(x) + \mathbf{b}(x) \\ B_a\mathbf{y}(a) + B_b\mathbf{y}(b) = \mathbf{c} \end{cases}$$

To solve that, consider *homogeneous IVP*

$$\mathbf{y}_i'(x) = A(x)\mathbf{y}_i(x)$$

with initial condition

$$\mathbf{y}_i(a) = \mathbf{e}_i.$$

Note: $\mathbf{y} \neq \mathbf{y_i}$. $\mathbf{e}_i$ is the ith unit vector. With that, build *fundamental solution matrix*

$$Y(x) = \begin{pmatrix} z| & & | \\ \mathbf{y}_1 & \cdots & \mathbf{y}_n \\ | & & | \end{pmatrix}$$

Let

$$Q := B_a Y(a) + B_b Y(b)$$

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→Use Newton.

Collocation Method

Consider

$$(*) \begin{cases} y'(x) = f(y(x), \\ g(y(a), y(b)) = 0. \end{cases}$$

(Scalar for simplicity–vector generalization is straightforward.)

What can we do?

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.

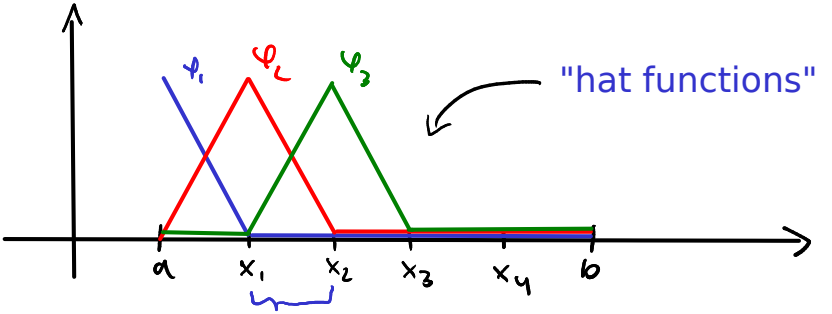
2. Idea: Pick n points where we would like $(*)$ to be satisfied.
   $\rightarrow$ Get a big (non-)linear system

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.



What's the problem with that?

Outline

Remark: Both PDEs and Sparse 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

We would love to see you there! :)

Solving Sparse Linear Systems

Solving $A\mathbf{x} = \mathbf{b}$ 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.

When do these methods converge?

$$\begin{aligned} A\mathbf{x} &= \mathbf{b} \\ M\mathbf{x} &= N\mathbf{x} + \mathbf{b} \\ M\mathbf{x}_{k+1} &= N\mathbf{x}_k + \mathbf{b} \\ \mathbf{x}_{k+1} &= M^{-1}(N\mathbf{x}_k + \mathbf{b}) \end{aligned}$$

- These methods are called *stationary* because they do the same thing in every iteration.
- They carry out fixed point iteration.
  $\rightarrow$ Converge if contractive, i.e. $\rho(M^{-1}N) < 1$.

Conjugate Gradient Method

Assume A is symmetric positive definite.{}

Idea: View solving $A\mathbf{x} = \mathbf{b}$ as an optimization problem.

$$\text{Minimize} \quad \varphi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b} \quad \Leftrightarrow \quad \text{Solve} \quad A\mathbf{x} = \mathbf{b}.$$

Observe $-\nabla\varphi(\mathbf{x}) = \mathbf{b} - A\mathbf{x} = \mathbf{r}$ (residual).

Use an iterative procedure ($\mathbf{s}_k$ is the search direction):

$$\begin{aligned} \mathbf{x}_0 &= \langle \text{starting vector} \rangle \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{s}_k, \end{aligned}$$

What should we choose for α_k (assuming we know $\mathbf{s}_k$)?

$$\begin{aligned} 0 &\stackrel{!}{=} \frac{\partial}{\partial\alpha}\varphi(\mathbf{x}_k + \alpha_k\mathbf{s}_k) \\ &= \nabla\varphi(\mathbf{x}_{k+1}) \cdot \mathbf{s}_k = \mathbf{r}_{k+1} \cdot \mathbf{s}_k. \end{aligned}$$

Introduction

Notation:

$$\frac{\partial}{\partial x} u \quad = \quad \partial_x u \quad = \quad u_x.$$

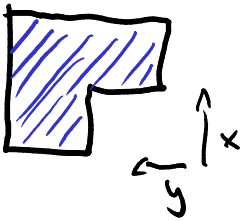
A *PDE* (*partial differential equation*) is an equation with multiple partial derivatives:

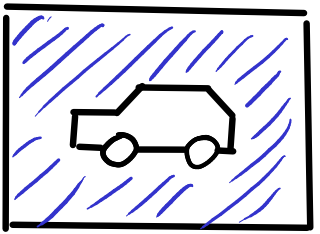
$$u_{xx} + u_{yy} = 0$$

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.





Outline