1 Introduction to Scientific Computing
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, ...
    - → Differential equations (ordinary and partial)
  - Machine learning
    - Statistical models, with unknown parameters
    - → Optimization
• Image and Audio processing
  ○ Enlargement/Filtering
  ○ → Interpolation
• Lots more.
What do we study, and how?

- Problems with real numbers (i.e. ‘continuous’ problems)
  - As opposed to discrete problems, which we won’t study that much.
  - Including the actually non-trivial question of: How do we put a real number into a computer? (and with what restrictions?)

- What mathematical ideas do we use?
  - I.e. what should the underlying mathematical representation be?
  - Given that representation, what solutions are even reachable?

- How good of an answer can we expect to our problem?
  - Since we can’t even represent numbers exactly, our 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?)

• How do these numerical methods get implemented?
  • 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

bit.ly/cs450-s16

- 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
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.
1.1 Errors, Conditioning, Accuracy, Stability
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 does, 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 \(\pi\)? (By rounding/truncating.)
Measuring Error

- How do we measure error?
  
  **Idea:** Consider all error as being *added onto* the result.

  \[
  \text{Absolute error} = \text{approx value} - \text{true value}
  \]

  \[
  \text{Relative error} = \frac{\text{Absolute error}}{\text{True value}}
  \]

**Problem:** True value not known

- Estimate
- ‘How big at worst?’ → Establish **Upper Bounds**
Recap: Norms

- **What’s a norm?**
  - \( f(x) : \mathbb{R}^n \rightarrow \mathbb{R}_0^+ \), returns a ‘magnitude’ of the input vector
  - In symbols: Often written \( \| x \| \).

- **Define norm.**

A function \( \| x \| : \mathbb{R}^n \rightarrow \mathbb{R}_0^+ \) is called a norm if and only if

1. \( \| x \| > 0 \Leftrightarrow x \neq 0 \).
2. \( \| \gamma x \| = |\gamma|\| x \| \) for all scalars \( \gamma \).
3. Obeys triangle inequality \( \| x + y \| \leq \| x \| + \| y \| \)
• Examples of norms?

The so-called $p$-norms:

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

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

• 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 $x \in \mathbb{R}^n$

$$
\alpha \|x\| \leq \|x\|^* \leq \beta \|x\|.
$$
So: No, doesn’t matter *that much*. Will start mattering more for so-called *matrix norms*—see later.
Demo: Vector norms
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:

  Magnitude of error ≠ ||true value|| − ||approximate value||  

Wrong!

Attempt 2:

  Magnitude of error = ||true value − approximate value||
Forward/Backward Error

- Suppose we’re intending to compute \( y = f(x) \), but actually obtain \( \hat{y} = \hat{f}(x) \).

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

As a figure:
\[ x - f \rightarrow y = f(x) \]

backward error  \hspace{2cm} forward error

\[ \hat{x} - f \rightarrow \hat{y} \]
Backward Error: Example

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

  - Forward:
    \[
    |\Delta y| = |1.4 - 1.41421...| \approx 0.0142...
    \]
    Relative forward error:
    \[
    \frac{|\Delta y|}{|y|} = \frac{0.0142...}{1.41421...} \approx 0.01.
    \]
    About 1 percent, or ‘two accurate digits’.

  - Backward:
    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.

- What do you observe about the relative errors?
  - In this case: Got smaller, i.e. variation *damped out*.
  - Typically: Not that lucky: Input error *amplified*.
In-class activity: Forward/Backward Error
Sensitivity and Conditioning

- What can we say about amplification of error?

Define **condition number** as smallest number \( \kappa \) so that

\[
|\text{rel. fwd. err.}| \leq \kappa \cdot |\text{rel. bwd. err.}|
\]

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 is called well-conditioned or insensitive
- ...large: the problem is called 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.

  \[ \kappa = \frac{|\Delta y|/|y|}{|\Delta x|/|x|} \]

  Forward error:

  \[ \Delta y = f(x + \Delta x) = f'(x)\Delta x \]

  Condition number:

  \[ \kappa \geq \frac{|\Delta y|/|y|}{|\Delta x|/|x|} = \frac{|f'(x)||\Delta x|/|f(x)|}{|\Delta x|/|x|} = \frac{|xf'(x)|}{|f(x)|}. \]
**Demo:** Conditioning of evaluating \( \tan() \)

**In-class activity:** Conditioning Theory
Stability and Accuracy

- When is a method **accurate**?
  
  Closeness of method output to true answer for unperturbed input.

- When is a method **stable**?
  
  If the method’s sensitivity to variation in the input is no (or not much) greater than that of the problem itself. (but see Note 2 below)

**Note:** Necessarily includes insensitivity to variation in intermediate results.
Note 2: The book uses a stronger notion: “A method is stable if the result it produces is the exact solution to a nearby problem.” This is commonly called ‘backward stability’ and is a stricter requirement than the notion of stability above, because it does not apply unless the algorithm is accurate. We will use ‘stability’ to mean ‘backward stability’ to avoid confusion unless otherwise specified.

- How can I produce inaccurate results?
  - Apply an inaccurate method
  - Apply an unstable algorithm to a well-conditioned problem
1.2 Floating Point
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 \( \geq 0 \)
- 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 \(\geq 0\) and 32 bits for exponents \(< 0\). What numbers can we represent?

\[
\begin{array}{cccc}
2^{31} & \ldots & 2^0 & 2^{-1} \ldots 2^{-32}
\end{array}
\]

**Smallest:** \(2^{-32} \approx 10^{-10}\)

**Largest:** \(2^{31} + \ldots + 2^{-32} \approx 10^9\)

- 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

  **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.
In-class activity: Floating Point
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 → all zeros after the implicit leading one
- Exponent as small as possible: \(-7\)

So:

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

Unfortunately: *wrong*. 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 exponent 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.

**Note:** Why would you want to know 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’.

- 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**. (An earlier version of the notes defined UFL to be a subnormal. To match the book, we will no longer use that definition.)
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**.
Running out of digits

- Suppose you store $\pi$ in a floating point number. What do you get?
  - An approximation, necessarily—because $\pi$ is irrational, therefore not a fraction, therefore not an exactly representable FP number.
  - The result gets rounded.

How is rounding performed?

\[
\begin{align*}
\text{(1.110101011)}_2 \\
\text{representable}
\end{align*}
\]

- “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? ("Nearest" really isn't well-defined here)

\[ 0.5 = (0.1)_2 \]

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

\[ 0.5 \rightarrow 0 \]
\[ 1.5 \rightarrow 2 \]
Demo: Density of Floating Point Numbers
Demo: Floating Point vs. Program Logic
Floating Point and Rounding Error

What is the relative error produced by working with floating point numbers?

- What is smallest floating point 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 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. $(\varepsilon_{\text{mach}} = 0.5 \text{ULP})$

- 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.)

- 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.

\[
\begin{align*}
a &= 1.101 \cdot 2^1 \\
b &= 0.01001 \cdot 2^1 \\
a + b &\approx 1.111 \cdot 2^1
\end{align*}
\]
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 $b = (1.1010)_2 \cdot 2^0$.

\[
\begin{align*}
a &= 1.1011 \cdot 2^1 \\
b &= 1.1010 \cdot 2^1 \\
a - b &\approx 0.0001???? \cdot 2^1
\end{align*}
\]

or, once we normalize,

\[
1.???? \cdot 2^{-3}.
\]

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

This phenomenon is called **Catastrophic Cancellation**.
Demo: Catastrophic Cancellation
2 Systems of Linear Equations
2.1 Theory: Conditioning
**Solving a Linear System**

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

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

**Want:** $n$-vector $x$ so that

$$Ax = b.$$ 

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

Unique solution exists iff $A$ is **nonsingular**.

**Next:** Want to talk about conditioning. Need a new tool.
Matrix Norms

- What norms would we apply to matrices?
  
  - Easy answer: ‘Flatten’ matrix as vector, use vector norm. Bad, not very meaningful.
  
  - Instead: Choose norms for matrices to interact meaningfully with an ‘associated’ vector norm $\| \cdot \|$ so that $\|A\|$ obeys

$$\|Ax\| \leq \|A\| \|x\|.$$ 

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

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

This is called the **matrix norm**.
• Logically, for each vector norm, we get a different matrix norm, so that, e.g. for the vector 2-norm $\|x\|_2$ we get a matrix 2-norm $\|A\|_2$, and for the vector $\infty$-norm $\|x\|_\infty$ we get a matrix $\infty$-norm $\|A\|_\infty$.

• Two factoids:

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

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

(not very difficult to show)

And the matrix 2-norm? Actually fairly difficult to evaluate. See later.

• Matrix and vector norms agree for $n \times 1$ matrices. (why?) (works well to help remember the two factoids)
Demo: Matrix norms
In-class activity: Matrix norms
Properties of Matrix Norms

Matrix norms inherit the vector norm properties:

1. $\|A\| > 0 \iff A \neq 0$.

2. $\|\gamma A\| = |\gamma|\|A\|$ for all scalars $\gamma$.

3. Obeys triangle inequality $\|A + B\| \leq \|A\| + \|B\|$

• But also some more properties that stem from our definition:

  1. $\|Ax\| \leq \|A\|\|x\|$

  2. $\|AB\| \leq \|A\|\|B\|$ (easy consequence)

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

- Now, let’s study conditioning of solving a linear system

$$Ax = b.$$ 

**Input:** $b$ with error $\Delta b$,

**Output:** $x$ with error $\Delta x$.

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

$$\frac{\text{rel err. in output}}{\text{rel err. in input}} = \frac{\|\Delta x\|/\|x\|}{\|\Delta b\|/\|b\|} = \frac{\|\Delta x\|/\|b\|}{\|\Delta b\|/\|x\|}$$

$$= \frac{\|A^{-1}\Delta b\|/\|Ax\|}{\|\Delta b\|/\|x\|}$$

$$\leq \|A^{-1}\|/\|A\| \frac{\|\Delta b\|/\|x\|}{\|\Delta b\|/\|x\|}$$

$$= \|A^{-1}\|/\|A\|. $$
So we’ve 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 $A$:

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

- cond is relative to a given norm. So, to be precise, use $\text{cond}_2$ or $\text{cond}_\infty$.

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

**In-class activity:** Matrix Conditioning

**Demo:** Conditioning of $2 \times 2$ Matrices
Residual and Error

- What is the residual vector of solving the linear system

\[ b = Ax \]

It’s the thing that’s ‘left over’. Suppose our approximate solution is \( \hat{x} \). Then the residual vector is

\[ r = b - A\hat{x}. \]

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

\[
\| \Delta x \| = \| x - \hat{x} \| \\
= \| A^{-1}(b - A\hat{x}) \| \\
= \| A^{-1}r \|
\]
Divide both sides by $\|\hat{x}\|$: 

$$ \frac{\|\Delta x\|}{\|\hat{x}\|} = \frac{\|A^{-1}r\|}{\|\hat{x}\|} \leq \frac{\|A^{-1}\|\|r\|}{\|\hat{x}\|} = \text{cond}(A) \frac{\|r\|}{\|A\|\|\hat{x}\|}. $$

What does this mean?

- $\text{rel err} \leq \text{cond} \cdot \text{rel resid}$
- 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.

\[ Ax = b \quad \rightarrow \quad A\hat{x} = \hat{b}. \]

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

\[ Ax = b \quad \rightarrow \quad \hat{A}\hat{x} = b. \]

What can we say about the error now?

Consider

\[
\Delta x = \hat{x} - x = A^{-1}(A\hat{x} - b) = -A^{-1}\Delta A\hat{x}.
\]

Thus

\[
\|\Delta x\| \leq \|A^{-1}\| \|\Delta A\| \|\hat{x}\|.
\]

And we get

\[
\frac{\|\Delta x\|}{\|\hat{x}\|} \leq \text{cond}(A) \frac{\|\Delta A\|}{\|A\|}.
\]
Changing Condition Numbers

- Once we have a matrix $A$ in a linear system $Ax = b$, are we stuck with its condition number? Or could we improve it?

  **Diagonal scaling** is a simple strategy that sometimes helps.

  - Row-wise:
    
    $$DAx = Db$$

  - Column-wise:
    
    $$AD\hat{x} = b$$

    Different $\hat{x}$: Recover $x = D\hat{x}$.

- What is this called as a general concept?
Preconditioning

- ‘Left’ preconditioning
  \[ MAx = Mb \]

- ‘Right’ preconditioning:
  \[ A M\hat{x} = b \]

Different \( \hat{x} \): Recover \( x = M\hat{x} \).

Right preconditioning is often far more effective than left.
In-class activity: Matrix Conditioning II
2.2 Methods to Solve Systems
Solving Systems

- Want methods/algorithms to solve linear systems. Starting small, a kind of system that’s easy to solve has a … matrix.

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
More General Matrices

- 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?

\[
\begin{pmatrix}
1 & 0 \\
0 & 1 \\
-\frac{1}{2} & 1 \\
0 & 0
\end{pmatrix}
\begin{pmatrix}
* & * & * & * & * \\
* & * & * & * & * \\
* & * & * & * & * \\
* & * & * & * & *
\end{pmatrix}
\]

- 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

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

should be

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

\[
A = M_1^{-1} M_2^{-1} \cdots M_{\ell-1}^{-1} M_{\ell}^{-1} U = L U.
\]

This is called LU factorization (or LU decomposition).

- Does this help solve \( Ax = b \)?

\[
\begin{align*}
Ax &= b \\
L \begin{bmatrix} y \\
x \end{bmatrix} &= b \\
Ly &= b \quad \leftarrow \text{solvable by fwd. subst.} \\
Ux &= y \quad \leftarrow \text{solvable by bwd. subst.}
\end{align*}
\]

Now know \( x \) that solves \( Ax = b \).
**Demo:** LU factorization
In-class activity: LU Factorization
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?

\[
\begin{pmatrix}
  0 & 1 \\
  2 & 1
  \end{pmatrix} = \begin{pmatrix}
  1 & \ell_{21} & 1
  \end{pmatrix} \begin{pmatrix}
  u_{11} & u_{12} \\
  u_{21} & u_{22}
  \end{pmatrix}
\]

So

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

\[
\underbrace{u_{11} \cdot \ell_{21}}_{0} + 1 \cdot 0 = 2
\]
It turns out to be that $A$ doesn’t have an LU factorization.

- 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?

\[
\begin{pmatrix}
1 & & & \\
 & 1 & & \\
 & & 1 & \\
& & & 1 \\
\end{pmatrix}
\begin{pmatrix}
B & B & B & B \\
C & C & C & C \\
D & D & D & D \\
\end{pmatrix}
= \begin{pmatrix}
C & C & C & C \\
B & B & B & B \\
D & D & D & D \\
\end{pmatrix}.
\]

\(P\) is called a permutation matrix.

Q: What’s \(P^{-1}\)?

- What does this process look like then?
\[ P_1 A \ \text{Pivot first column} \]
\[ M_1 P_1 A \ \text{Eliminate first column} \]
\[ P_2 M_1 P_1 A \ \text{Pivot second column} \]
\[ M_2 P_2 M_1 P_1 A \ \text{Eliminate second column} \]
\[ P_3 M_2 P_2 M_1 P_1 A \ \text{Pivot third column} \]
\[ M_3 P_3 M_2 P_2 M_1 P_1 A \ \text{Eliminate third column} \]

Or

\[ A = P_1 M_1^{-1} P_2 M_2^{-1} P_3 M_3^{-1} U. \]

Unfortunately, \( P \)'s and \( M \)'s don't commute, so it's not obvious how to get a lower-triangular \( L \).
Demo: LU with Pivoting (Part I)
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

$$L_3 L_2 L_1 P_3 P_2 P_1$$
$$= M_3 (P_3 M_2 P_3^{-1}) (P_3 P_2 M_1 P_2^{-1} P_3^{-1}) P_3 P_2 P_1$$
$$= M_3 P_3 M_2 P_2 M_1 P_1$$ (1)
Summing up:

\[ P_3P_2P_1A = L_1^{-1}L_2^{-1}L_3^{-1}U. \]

\[ PA = LU. \]

Best miracle of all: \( L_1, \ldots, L_3 \) are still lower triangular!

**Q:** Outline the solve process with pivoted LU.
Demo: LU with partial pivoting (Part II)
Computational Cost

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

  $O(n^3)$

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

  Recall

  \[ M_3P_3M_2P_2M_1P_1A = U \ldots \]

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

  Fortunately not: Multiplications with permutation 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 $Ax = b$?
  LU: $O(n^3)$
  FW/BW Subst: $2 \times O(n^2) = O(n^2)$

- What’s the cost of solving $Ax = b_1, b_2, ..., b_n$?
  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** \( z = \alpha x + y \)  
vector-vector operations  
\( O(n) \)  
?axpy

**Level 2** \( z = Ax + y \)  
matrix-vector operations  
\( O(n^2) \)  
?gemv

**Level 3** \( C = AB + \beta C \)  
matrix-matrix operations  
\( O(n^3) \)  
?gemm, ?trsm

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

- banded
- sparse (see hw0)
**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.

\[
\hat{A} = A + uv^T
\]

(a so-called rank-one update)

The Sherman-Morrison formula gives us

\[
(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}.
\]

FYI: There is a rank-\(k\) analog called the Sherman-Morrison-Woodbury formula.
Demo: Sherman-Morrison
3 Linear Least Squares

3.1 Introduction
What about non-square systems?

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

- Specifically, any hope that we will solve those exactly?

  Not really: more equations than unknowns.
Example: Data Fitting

Have data: \((x_i, y_i)\) and model:

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

- Find data that best fit model:

\[
\alpha + \beta x_1 + \gamma x_1^2 = y_1
\]
\[ \alpha + \beta x_n + \gamma x_n^2 = y_n \]

Not going to happen for \( n > 3 \). Instead:

- minimize residual
- minimize 2-norm of residual

\[ \rightarrow \text{‘Least Squares’} \]

\[ |\alpha + \beta x_1 + \gamma x_1^2 - y_1|^2 + \cdots + |\alpha + \beta x_n + \gamma x_n^2 - y_n|^2 \rightarrow \min! \]

This is called \textit{linear least squares} specifically because the coefficients \( x \) enter linearly into the residual.

Easy to see: writable as

\[ \| A x - b \|^2 \rightarrow \min! \]
Matrices like $A$ are called **Vandermonde matrices**.

- Easy to generalize to higher polynomial degrees.

\[ A = \begin{pmatrix} 1 & x_1 & x_1^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{pmatrix}, \quad x = \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix}, \quad b = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \]

\[ \| A x - b \|_2^2 \rightarrow \min! \]

\[ A x \approx b \]

**Q:** Give an example of a nonlinear least squares problem.

\[ | \exp(\alpha) + \beta x_1 + \gamma x_1^2 - y_1 |^2 + \ldots + | \exp(\alpha) + \beta x_n + \gamma x_n^2 - y_n |^2 \rightarrow \min! \]
But that would be easy to remedy: Do linear least squares with \( \exp(\alpha) \) as the unknown. More difficult:

\[
|\alpha + \exp(\beta x_1 + \gamma x_1^2) - y_1|^2 + \cdots +
|\alpha + \exp(\beta x_n + \gamma x_n^2) - y_n|^2 \rightarrow \min!
\]
Demo: Interactive Polynomial Fitting
Properties of Least-Squares
Consider the least squares problem $Ax \simeq b$ and its associated objective function

$$\varphi(x) = \|b - Ax\|_2^2.$$ 

- Is there always a solution to a linear least-squares problem
  
  Yes. $\varphi \geq 0$, $\varphi \to \infty$ as $\|x\| \to \infty$, $\varphi$ continuous $\Rightarrow$ has a minimum.

- Is it unique?
  
  No, for example if $A$ has a nullspace.

- Examine the objective function, find its minimum.
  
  $$\varphi(x) = (b - Ax)^T(b - Ax)$$
  $$= b^Tb - 2x^TA^Tb + x^TA^TAx$$
  $$\nabla \varphi(x) = -2A^Tb + 2A^TAx$$
\[ \nabla \varphi(x) = 0 \text{ yields } A^T A x = A^T b \]

These are called the normal equations.
In-class activity: Least Squares
Demo: Polynomial fit using the normal equations

• What’s the shape of $A^TA$?
  Always square.

Demo: Issues with the normal equations
Least Squares, Viewed Geometrically

- Why is \( r \perp \text{span}(A) \) a good thing to require?

Because then the distance between \( y = Ax \) and \( b \) is minimal.

Q: Why?
Because of Pythagoras’s theorem—another \( y \) would mean additional distance traveled in \( \text{span}(A) \).
• Phrase that as an equation.

\[ \text{span}(A) \perp b - Ax \]
\[ A^Tb - A^T Ax = 0 \]

Congratulations: Just rediscovered the normal equations.

• Write that with an orthogonal projection matrix \( P \).

\[ Ax = Pb. \]
About Orthogonal Projectors

- What is a projector?
  
  A matrix satisfying
  
  \[ P^2 = P. \]

- What is an orthogonal projector?
  
  A symmetric projector.

- How do I make one projecting onto \( \text{span}\{q_1, q_2, \ldots, q_\ell\} \)?
  
  First define
  
  \[ Q = \begin{pmatrix} q_1 & q_2 & \cdots & q_\ell \end{pmatrix}. \]
Then

\[ Q Q^T \]

will project and is obviously symmetric.

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

\[
P^2 = A(A^T A)^{-1} A^T A (A^T A)^{-1} A^T = A(A^T A)^{-1} A^T = P.
\]

Symmetry: also yes.

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

**Conclusion:** \( P \) is the projector from the previous slide!
What assumptions do we need to define the $P$ from the last question?

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

- What is the pseudoinverse of $A$?

  Nonsquare $m \times n$ matrix $A$ has no inverse in usual sense. If 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?

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

  If not full rank, $\text{cond}(A) = \infty$ by convention.
What does all this have to do with solving least squares problems?

\[ x = A^+ b \] solves \( Ax \approx b \).
3.2 Sensitivity and Conditioning
Sensitivity and Conditioning of Least Squares

- How sensitive is it?

Define

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

then

\[ \frac{\|\Delta x\|_2}{\|x\|_2} \leq \text{cond}(A) \frac{1}{\cos(\theta)} \cdot \frac{\|\Delta b\|_2}{\|b\|_2}. \]
• What values of $\theta$ are bad?

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

• Any comments regarding dependencies?

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

• What about changes in the matrix?

$$\frac{\|\Delta \mathbf{x}\|_2}{\|\mathbf{x}\|_2} \leq [\text{cond}(A)^2 \tan(\theta) + \text{cond}(A)] \cdot \frac{\|\Delta A\|_2}{\|A\|_2}.$$
Two behaviors:

- If $\tan(\theta) \approx 0$, condition number is $\text{cond}(A)$.
- Otherwise, $\text{cond}(A)^2$.
3.3 Solving Least Squares
Ideas for Solving Least Squares

- Tell me about the augmented system method.

**Idea:** Solve for residual and solution.

\[ r - Ax = b \]
\[ A^T r = 0 \]

Bad:

- Not spd
- Still poorly conditioned
- 4× the storage, including two copies of A.
Least-squares by Transformation

- Want a matrix $Q$ so that

\[ QAx \cong Qb \]

has the same solution as

\[ Ax \cong b. \]

I.e. want

\[ \|Q(Ax - b)\|_2 = \|Ax - b\|_2. \]

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 $Q^TQ = I$ and $QQ^T = I$.

- Are orthogonal projectors orthogonal?
  
  Nope, not in general.

- Now what about that norm property?

  $$\|Q\mathbf{v}\|_2^2 = (Q\mathbf{v})^T(Q\mathbf{v}) = \mathbf{v}^TQ^TQ\mathbf{v} = \mathbf{v}^T\mathbf{v} = \|\mathbf{v}\|_2^2.$$
Simpler Problems: Triangular

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

\[
\begin{pmatrix}
R \\
0
\end{pmatrix}x \cong \begin{pmatrix}
b_1 \\
b_2
\end{pmatrix}
\]

- If so, how would we minimize the residual norm?

\[\|r\|_2^2 = \|b_1 - Rx\|_2^2 + \|b_2\|_2^2.\]

\(R\) is invertible, so we can find \(x\) so that \(\|b_1 - Rx\|_2^2 = 0\), i.e.

\[\|r\|_2^2 = \|b_2\|_2^2.\]
So the goal becomes: Find orthogonal matrix $Q$ so that $Q^TA = R$, 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 $a$.

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

- Rotate
- Reflect

Reflection turns out to do the job:
Justify the orthogonality in the picture:

\[(a - \|a\|_2 e_1) \cdot (a + \|a\|_2 e_1) = \|a\|_2^2 - \|a\|_2^2\]

Let’s call \(v = a - \|a\|_2 e_1\). How do we reflect about the plane orthogonal
to \( \mathbf{v} \)? Project-and-keep-going:

\[
H := I - 2 \frac{\mathbf{v} \mathbf{v}^T}{\mathbf{v}^T \mathbf{v}}.
\]

This is called a **Householder reflector**.

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

\[
Ha = e_1.
\]

Remarks:

- **Q:** What if we want to zero out only the \( i + 1 \)th through \( n \)th entry?
  **A:** Use \( e_i \) above.

- A product \( H_n \cdots H_1 A = R \) of Householders makes it easy (and quite efficient!) to build a QR factorization.

- It turns out \( \mathbf{v}' = \mathbf{a} + \|\mathbf{a}\|_2 e_1 \) works out, too—just pick whichever one causes less cancellation.
- $H$ is symmetric
- $H$ is orthogonal
Demo: 3x3 Householder QR
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 solve for \( c \) and \( s \).

**Downside?** Produces only one zero at a time.
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?

\[
A P = QR
\]

Also used as the basis for rank-revealing QR.
Rank-Deficient Matrices and Least-Squares

- What happens with Least Squares for rank-deficient matrices?

\[ Ax \approx b \]

- 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. \( x + n \) for \( n \in N(A) \) has the same residual.
- **In other words:** Have more freedom

**Or:** Can demand another condition, for example:

- Minimize \( \| b - Ax \|_2^2 \), and
- minimize \( \| x \|_2^2 \), simultaneously.

Unfortunately, QR does not help much with that → 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.

- $\Sigma = \text{diag}(\sigma_i)$ is $m \times n$ and non-negative
  Typically $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. 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
- \text{Nullspace } N(A) = \text{span}(\{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 1.** *(Eckart-Young)* If $k < r = \text{rank}(A)$ and

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

then

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

• The minimum norm solution to $Ax \cong b$:

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

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

$$\Sigma y = U^T b$$
Then define

$$\Sigma^+ = \begin{pmatrix} \sigma_1^+ \\ \vdots \\ \sigma_n^+ \end{pmatrix}$$

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

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

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

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

solves the minimum-norm least-squares problem.

- Define $A^+ = V \Sigma^+ U^T$ and call it the pseudoinverse. Coincides with prior definition in case of full rank.
In-class activity: SVDs
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
4 Eigenvalue Problems
Eigenvalue Problems: Setup/Math Recap

$A$ is an $n \times n$ matrix.

- $x \neq 0$ is called an eigenvector of $A$ if there exists a $\lambda$ so that
  \[ Ax = \lambda x. \]

- In that case, $\lambda$ 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:

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

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

**Q:** Does that help algorithmically?

**A:** Abel showed that for \( n \geq 5 \) there is no general formula for the roots of the polynomial. (i.e. no analog to the quadratic formula for \( n = 5 \))
Algorithmically, that means we will need to approximate. So far (e.g. for LU and QR), if it had not been for FP error, we would have obtained exact answers. For eigenvalue problems, that is no longer true—we can only hope for an approximate answer.
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: $AM \geq GM$.
If $AM > GM$, the matrix is called **defective**.
An Example

- Give characteristic polynomial, eigenvalues, eigenvectors of

\[
\begin{pmatrix}
1 & 1 \\
1 & 1
\end{pmatrix}.
\]

CP: \((\lambda - 1)^2\)

Eigenvalues: 1 (with multiplicity 2)

Eigenvectors:

\[
\begin{pmatrix}
1 & 1 \\
1 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y
\end{pmatrix} =
\begin{pmatrix}
x \\
y
\end{pmatrix}
\]

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

In that case, let

\[
X = \begin{pmatrix} x_1 & \cdots & x_n \end{pmatrix},
\]

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?)
4.1 Sensitivity
**Sensitivity**

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

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.

$$(D + F)\mathbf{v} = \mu\mathbf{v}$$

$$\Leftrightarrow F\mathbf{v} = (\mu I - D)\mathbf{v}$$

$$\Leftrightarrow (\mu I - D)^{-1}F\mathbf{v} = \mathbf{v} \quad \text{(when is that invertible?)}$$
\[ \Rightarrow \| v \| \leq \| (\mu I - D)^{-1} \| F \| \| v \| \]
\[ \Rightarrow \| (\mu I - D)^{-1} \|^{-1} \leq \| F \| \]

Now
\[ \| (\mu I - D)^{-1} \|^{-1} = |\mu - \lambda_k| \]

where \( \lambda_k \) is the closest eigenvalue of \( D \) to \( \mu \).

\[ |\mu - \lambda_k| = \| (\mu I - D)^{-1} \|^{-1} \leq \| F \| = \| X^{-1} E X \| \leq \text{cond}(X) \| E \|. \]

- This is ‘bad’ if \( X \) is ill-conditioned, i.e. if the eigenvectors are nearly linearly dependent.
- If \( X \) is orthogonal (e.g. for symmetric \( A \)), then eigenvalues are always well-conditioned.
- This bound is in terms of all eigenvalues, so may overestimate for each individual eigenvalue.
In-class activity: Eigenvalues
4.2 Properties and Transformations

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

- **Shift.** $A \rightarrow A - \sigma I$
  
  $$(A - \sigma I)\mathbf{x} = (\lambda - \sigma)\mathbf{x}$$

- **Inversion.** $A \rightarrow A^{-1}$
  
  $$A^{-1}\mathbf{x} = \lambda^{-1}\mathbf{x}$$

- **Power.** $A \rightarrow A^k$
  
  $$A^k\mathbf{x} = \lambda^k\mathbf{x}$$

- **Polynomial** $A \rightarrow aA^2 + bA + cI$
  
  $$(aA^2 + bA + cI)\mathbf{x} = (a\lambda^2 + b\lambda + c)\mathbf{x}$$
- **Similarity** $T^{-1}A\ T$ with $T$ invertible
  
  Let $y := T^{-1}x$. Then
  
  $$T^{-1}A\ Ty = \lambda y$$
**Schur form**

- Show: Every matrix is orthonormally similar to an upper triangular matrix, i.e.

\[ A = Q U Q^T. \]

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

\[
\begin{align*}
A: & \ V \rightarrow V \\
V^\perp & \rightarrow V \oplus V^\perp
\end{align*}
\]
In matrix form

\[
A = \begin{pmatrix}
\vdots & \\
v & \text{Basis of } V^\perp
\end{pmatrix}
\begin{pmatrix}
\lambda & * & * & * \\
0 & * & * & * \\
\vdots & * & * & * \\
0 & * & * & *
\end{pmatrix}Q_1^T.
\]

Repeat \( n \) times: Matrix becomes triangular:

\[
Q_n \cdots Q_1 U Q_1^T \cdots Q_n^T.
\]

For complex \( \lambda \): Either complex matrices or \( 2 \times 2 \) blocks on diag.

This is useful because now the eigenvalues are on the diagonal.
4.3 Computing Eigenvalues
Power Iteration

- What are the eigenvalues of $A^{1000}$?

Assume $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$ with eigenvectors $x_1, \ldots, x_n$. Further assume $\|x_i\| = 1$.

Use $x = \alpha x_1 + \beta x_2$.

$$y = A^{1000}(\alpha x_1 + \beta x_2) = \alpha \lambda_1^{1000} x_1 + \beta \lambda_2^{1000} x_2$$

Or

$$\frac{y}{\lambda_1^{1000}} = \alpha x_1 + \beta \left( \frac{\lambda_2}{\lambda_1} \right)^{1000} x_2.$$ 

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

- What could go wrong with Power Iteration?
  - Starting vector has no component along $x_1$
    - Not a problem in practice: Rounding will introduce one.
  - Overflow in computing $\lambda_1^{1000}$
    - $\rightarrow$ 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:

\[
\frac{x^T A x}{x^T x}
\]

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

This is called the **Rayleigh quotient**.
Convergence of Power Iteration

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

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

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 \(\sigma\), with
    \[
    \mathbf{e}_{k+1} \approx \frac{|\lambda_{\text{closest}} - \sigma|}{|\lambda_{\text{second-closest}} - \sigma|} \mathbf{e}_k.
    \]

- 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 $X_0 \in \mathbb{R}^{n \times p}$ ($p \leq n$) with (arbitrary) iteration vectors in columns

2. $X_{k+1} = AX_k$

Problems:

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

Fix: orthogonalize!
Orthogonal Iteration

1. Start with $X_0 \in \mathbb{R}^{n \times p}$ ($p \leq n$) 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:

\[
\begin{align*}
Q_0R_0 &= X_0 \\
X_1 &= A \ Q_0 \\
Q_1R_1 &= X_1 = A \ Q_0 \quad \Rightarrow \quad Q_1R_1Q_0^T = A \\
X_2 &= A \ Q_1 \\
Q_2R_2 &= X_2 = A \ Q_1 \quad \Rightarrow \quad Q_2R_2Q_1^T = A
\end{align*}
\]

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

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

**Observation 1:** Once \( Q_{n+1} \approx Q_n \), we also have \( Q_2R_2Q_2^T \approx A \).

**Observation 2:** \( \hat{X}_2 := Q_2^T A \ Q_2 \approx R_2 \).

**Idea:** Use magnitude of below-diag part of \( \hat{X}_2 \) for convergence check.

**Q:** Can we compute \( \hat{X}_k \) directly?
Demo: Orthogonal Iteration
In-class activity: Eigenvalue Iterations
QR Iteration/QR Algorithm

Orthogonal iteration:
\[ X_0 = A \]
\[ Q_k R_k = X_k \]
\[ X_{k+1} = A Q_k \]

QR iteration:
\[ \tilde{X}_0 = A \]
\[ \tilde{Q}_k \tilde{R}_k = \tilde{X}_k \]
\[ \tilde{X}_{k+1} = \tilde{R}_k \tilde{Q}_k \]

Tracing through reveals:
- \[ \hat{X}_k = \tilde{X}_{k+1} \]
- \[ Q_0 = \tilde{Q}_0 \]
- \[ Q_1 = \tilde{Q}_0 \tilde{Q}_1 \]
- \[ Q_k = \tilde{Q}_0 \tilde{Q}_1 \ldots \tilde{Q}_k \]

Orthogonal iteration showed: \( \hat{X}_k = \tilde{X}_{k+1} \) converge. Also:
\[ \tilde{X}_{k+1} = \tilde{R}_k \tilde{Q}_k = \tilde{Q}_k^T \bar{X}_k \tilde{Q}_k, \]
so the \( \tilde{X}_k \) are all similar \( \rightarrow \) all have the same eigenvalues.
\( \rightarrow \) QR iteration produces Schur form.
QR Iteration: Incorporating a Shift

- How can we accelerate convergence of QR iteration using shifts?

  \[
  \begin{align*}
  \bar{X}_0 &= A \\
  \bar{Q}_k \bar{R}_k &= \bar{X}_k - \sigma_k l \\
  \bar{X}_{k+1} &= \bar{R}_k \bar{Q}_k + \sigma_k l
  \end{align*}
  \]

  Still a similarity transform:

  \[
  \bar{X}_{k+1} = \bar{R}_k \bar{Q}_k + \sigma_k l = [Q_k^T \bar{X}_k - Q_k^T \sigma_k] \bar{Q}_k + \sigma_k l
  \]

  **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 \times 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

For symmetric matrices:

- Use Householders to attain tridiagonal form
- Use QR iteration with Givens to attain diagonal form
Demo: Householder Similarity Transforms
4.4 Krylov Space Methods
Krylov space methods: Intro

What subspaces can we use to look for eigenvectors?

QR:

$$\text{span}\{A^\ell y_1, A^\ell y_2, ..., A^\ell y_k\}$$

Krylov:

$$\text{span}\left\{ x_0, Ax, ..., \underbrace{A^{k-1}x}_{x_{k-1}} \right\}$$

Define matrix

$$K_k := \begin{pmatrix} x_0 & \cdots & x_{k-1} \end{pmatrix} \quad (n \times k)$$
Then

\[ AK_n = \left( \begin{array}{ccc} x_1 & \cdots & x_n \\ \vdots & \ddots & \vdots \\ \vdots & \vdots & x_n \end{array} \right) = K_n \left( \begin{array}{ccc} e_2 & \cdots & e_n \\ \vdots & \ddots & \vdots \\ \vdots & \vdots & e_n \end{array} \right) K_n^{-1} x_n \]

\[ C_n \]

Realize:

- \[ K_n^{-1} A K_n = C_n \]
- \[ C_n \] is upper Hessenberg
- So Krylov is ‘just’ another way to get a matrix into upper Hessenberg form. But: built incrementally!
Conditioning in Krylov Space Methods/Arnoldi Iteration

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

$(x_i)$ converge rapidly to eigenvector for largest eigenvalue
$\to K_k$ 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

$$Q_n^T A Q_n = R_n K_n^{-1} A K_n R_n^{-1}.$$

Realizing that

- $C_n$ is upper Hessenberg
- (Left/right) Multiplication by triangulars preserves upper Hessenberg
we find that $Q_n^T A Q_n$ is also upper Hessenberg:

$$Q_n^T A_n Q_n = H.$$ 

Also readable as $A Q_n = Q_n H$, which, read column-by-column, is:

$$A q_k = h_{1k} q_1 + \ldots h_{k+1,k} q_{k+1}$$

We find: $h_{jk} = q_j^T A q_k$.

**Important consequence:** Can compute

- $q_{k+1}$ from $q_1, \ldots, q_k$
- $(k+1)$st column of $H$

analogously to Gram-Schmidt QR!

This is called **Arnoldi iteration**.
For symmetric matrices: **Lanczos iteration.**
Demo: Arnoldi Iteration (Part I)
Krylov: What about eigenvalues?

- How can we use Arnoldi/Lanczos 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)

**In-class activity:** Krylov space methods
5 Nonlinear Equations
Solving Nonlinear Equations

What is the goal here?

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

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

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

*Demo*: Intersection of quadratics
Showing Existence

- How can we show existence of a root?

  - Intermediate value theorem  
    (uses continuity, 1D only)

  - Inverse function theorem  
    (relies on invertible Jacobian $J_f$, works only locally)

  - Contraction mapping theorem  
    A function $g: \mathbb{R}^n \to \mathbb{R}^n$ is called contractive if there exists a $0 < \gamma < 1$ so that
    $$\|g(x) - g(y)\| \leq \gamma \|x - y\|.$$  
    A fixed point of $g$ is a point where $g(x) = x$.

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

**Q:** What can we say about uniqueness in general?

**A:** Usually, the solution is not unique.
Sensitivity and Multiplicity

- What is the sensitivity/conditioning of root finding?
  \[
  \text{cond(root finding)} = \text{cond(evaluation of the inverse function)}
  \]

- What are multiple roots?
  \[
  f(x) = 0 \\
  f'(x) = 0 \\
  
  \vdots \\
  f^{(m-1)}(x) = 0
  \]

  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.
5.1 Iterative Procedures

- What is linear convergence? quadratic convergence?

Let $e_k = \hat{u}_k - u$ be the error in the $k$th iterate $\hat{u}_k$.

An iterative method converges with rate $r$ if

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

$r = 1$ 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:
    \[ \|e_{k+1}\| \leq C \|e_k\| \]
    (and \( C < 1 \) matters!)
  - Quadratic convergence
    \[ \|e_{k+1}\| \leq C \|e_k\|^2 \]
    (Only starts making sense once \( \|e_k\| \) is small. \( C \) doesn’t matter much.)
Contrast the following ‘alternate’ definitions of convergence rate:

\[ (1) \quad \frac{\|e_{k+1}\|}{\|e_k\|} \leq C \begin{cases} >0, \\ <\infty. \end{cases} \]

\[ (2) \quad 0 < C_{\text{low}} \leq \frac{\|e_{k+1}\|}{\|e_k\|} \leq C_{\text{high}} \]

\[ (3) \quad \lim_{k \to \infty} \frac{\|e_{k+1}\|}{\|e_k\|} = C \begin{cases} >0, \\ <\infty. \end{cases} \]

- (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 behavior.
Stopping Criteria

Here are some ideas for stopping criteria:

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

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.
5.2 Methods in One Dimension
Bisection Method

**Demo:** Bisection Method

- What’s the rate of convergence? What’s the constant? Linear with constant $1/2$. 
**Fixed Point Iteration**

\[
\begin{align*}
  x_0 &= \langle \text{starting guess} \rangle \\
  x_{k+1} &= g(x_k)
\end{align*}
\]

**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 \( \theta_k \) between \( x_k \) and \( x^* \) so that

\[
g(x_k) - g(x^*) = g'(\theta_k)(x_k - x^*) = g'(\theta_k)e_k.
\]
So:

\[ e_{k+1} = g'(\theta_k)e_k \]

and if \( \|g'\| \leq C < 1 \) near \( x^* \), then we have linear convergence with constant \( C \).

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

By Taylor:

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

So we have *quadratic convergence* in this case!

We would obviously like a systematic way of finding \( g \) that produces quadratic convergence.
Newton’s Method

- Derive Newton’s method.

**Idea:** Approximate \( f \) at 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 \( h \):

\[
f(x_k) + f'(x_k)h = 0 \iff h = -\frac{f(x_k)}{f'(x_k)}.
\]

So

\[
x_0 = \langle \text{starting guess} \rangle
\]

\[
x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = g(x_k)
\]

Then let’s look at

\[
g'(x) = \frac{f(x)f''(x)}{f'(x)^2}
\]
So if $f(x^*) = 0$ and $f'(x^*) \neq 0$, we have $g'(x^*) = 0$, i.e. quadratic convergence toward single roots.

**Drawbacks** of Newton:

- Convergence argument only good *locally*
  Will see: convergence only local (near root)
- Have to have derivative!
Demo: Newton’s method
Demo: Convergence of Newton’s Method
**Secant Method**

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

  Approximate

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

  So

  \[
  x_0 = \langle \text{starting guess} \rangle \\
  x_{k+1} = x_k - \frac{f(x_k)}{f(x_k) - f(x_{k-1})}. \\
  \]

  Rate of convergence (not shown) is \( \frac{1 + \sqrt{5}}{2} \approx 1.618. \)

**Drawbacks of Secant:**

- Convergence argument only good *locally*
  Will see: convergence only local (near root)
• Slower convergence
• Need two starting guesses
Demo: Secant Method
Demo: Convergence of the Secant Method
In-class activity: Nonlinear equations in 1D
‘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.
5.3 Methods in $n$ Dimensions ("Systems of Equations")
Fixed Point Iteration

\[ x_0 = \text{(starting guess)} \]
\[ x_{k+1} = g(x_k) \]

- When does this converge?

Converges (locally) if \( \rho(J_g(x^*)) < 1 \), where the Jacobian matrix

\[
J_g(x^*) = \begin{pmatrix}
\partial_{x_1}g_1 & \cdots & \partial_{x_n}g_1 \\
\vdots & \ddots & \vdots \\
\partial_{x_1}g_n & \cdots & \partial_{x_n}g_n
\end{pmatrix}.
\]

Similarly: If \( J_g(x^*) = 0 \), we have at least quadratic convergence.
Newton’s method

- What does Newton’s method look like in \( n \) dimensions?

Approximate by linear:

\[
f(x + s) = f(x) + J_f(x)s
\]

Set to 0:

\[
J_f(x)s = -f(x) \quad \Rightarrow \quad s = - (J_f(x))^{-1}f(x)
\]

So

\[
x_0 = \text{(starting guess)}
\]

\[
x_{k+1} = x_k - (J_f(x_k))^{-1}f(x_k)
\]

Downsides:

- Still only locally convergent
- Computing and inverting \( J_f \) is expensive.
Demo: Newton’s method in $n$ dimensions
Secant in $n$ dimensions?

- What would the secant method look like in $n$ dimensions?

Need an ‘approximate Jacobian’ satisfying

$$\tilde{J}(x_{k+1} - x_k) = f(x_{k+1}) - f(x_k).$$

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

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

One choice: Broyden’s method (minimizes change to $\tilde{J}$)
6 Optimization
Optimization

- State the problem.

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

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

- $g(x) = 0$ and $h(x) \leq 0$ are called constraints.
  They define the set of feasible points $x \in S \subseteq \mathbb{R}^n$.

- If $g$ or $h$ are present, this is constrained optimization.
  Otherwise unconstrained optimization.

- If $f$, $g$, $h$ are linear, this is called linear programming.
  Otherwise nonlinear programming.

- **Q:** What if we are looking for a maximizer?
• Examples:
  o What is the fastest/cheapest/shortest... way to do...?
    **Q:** What about multiple objectives?
    **A:** Make up your mind–decide on one (or build a combined objective). Then we’ll talk.

  o Solve a system of equations ‘as well as you can’ (if no exact solution exists)–similar to what least squares does for linear systems:
    \[
    \min \| F(x) \| \n    \]
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
    \[
    \lim_{\|x\| \to \infty} f(x) = +\infty
    \]
    If $f$ is coercive, a global minimum exists (but is possibly non-unique).
  - $S \subseteq \mathbb{R}^n$ is called convex if for all $x, y \in S$ and all $0 \leq \alpha \leq 1$
    \[
    \alpha x + (1 - \alpha)y \in S.
    \]
• The function $f: S \to \mathbb{R}$ is called **convex** on $S \subseteq \mathbb{R}^n$ if for $x, y \in S$ and all $0 \leq \alpha \leq 1$

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

With ‘<’: **strictly convex**.

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

• If $f$ is convex, then $f$ is continuous at interior points. (Why? What would happen if it had a jump?)

• If $f$ is convex, a local minimum is a **global minimum**.

• If $f$ is strictly convex, a local minimum is a **unique global minimum**.
Optimality Conditions

- If we have found a candidate $x^*$ for a minimum, how do we know it actually is one? 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: $\nabla f(x^*) = 0$ (i.e. $x^*$ is an extremal point)
    - Sufficient: $\nabla f(x^*) = 0$ and $H_f(x^*)$ positive definite (implies $x^*$ is a local minimum)
where the Hessian

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

Q: Come up with a hypothetical approach for finding minima.
A: Solve \( \nabla f = 0 \).

Q: Is the Hessian symmetric?
A: Yes. (Schwarz’s theorem)

Q: How can we practically test for positive definiteness?
A: Attempt a Cholesky factorization. If it succeeds, the matrix is PD.
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:
    \[
    f(x^* + h) = f(x^*) + f'(x^*)h + f''(x^*)\frac{h^2}{2} + O(h^3)
    \]
    Solve for \(h\):
    \[
    |\tilde{x} - x^*| \leq \sqrt{2 \text{tol}/f''(x^*)}
    \]
    **In other words:** Can expect *half as many digits* in \(\tilde{x}\) as in \(f(\tilde{x})\). This is important to keep in mind when setting tolerances.
It’s only possible to do better when derivatives are explicitly known and convergence is not based on function values alone. (then: can solve $\nabla f = 0$)

- In $n$ dimensions: Assume $\|s\| = 1$.

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

Yields:

$$|h|^2 \leq \frac{2 \text{tol}}{\lambda_{\min}(H_f(x^*))}.$$  

**In other words:** Conditioning of $H_f$ determines sensitivity of $x^*$. 
6.1 Methods for unconstrained opt. in one dimension
Golden Section Search

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

\[ f \text{ is called unimodal if for all } x_1 < x_2 \]

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

- \[ x^* < x_1 \Rightarrow f(x_1) < f(x_2) \]

Suppose we have an interval with \( f \) unimodal:
Would like to maintain unimodality.

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

Remaining question: Where to put $x_1, x_2$?

- Want symmetry:
  
  $x_1 = a + (1 - \tau)(b - a)$
  $x_2 = a + \tau(b - a)$

- Want to reuse function evaluations: $\tau^2 = 1 - \tau$
  Find: $\tau = \left(\sqrt{5} - 1\right)/2$. Also known as the ‘golden section’.

- Hence golden section search.

Linearly convergent. Can we do better?
Demo: Golden Section Search Proportions
Newton’s Method

- Reuse the Taylor approximation idea, but for optimization.

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

**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).
**Demo:** Newton’s method in 1D

**In-class activity:** Optimization Methods
6.2 Methods for unconstrained opt. in $n$ dimensions
**Steepest Descent**

- Given a scalar function $f: \mathbb{R}^n \to \mathbb{R}$ at a point $x$, 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. $x_0 = \langle$ some starting guess $\rangle$
2. $s_k = -\nabla f(x_k)$
3. Use line search to choose $\alpha_k$ to minimize $f(x_k + \alpha_k s_k)$
4. $x_{k+1} = x_k + \alpha_k s_k$
5. Go to 2.

**Observation:** (from demo)

- Linear convergence
Demo: Steepest Descent
Newton’s method (nD)

- 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 \( \nabla \hat{f}(\mathbf{s}) = 0 \) 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)

- local convergence
- Works poorly when $H_f$ is nearly indefinite
Demo: Newton’s method in $n$ dimensions

Demo: Nelder-Mead Method
6.3 Nonlinear Least Squares
Nonlinear Least Squares/Gauss-Newton

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

$$f(x) = \| r(x) \|_2, \quad r(x) = y - f(x)$$

Define ‘helper function’

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

and minimize that instead.

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

or, in matrix form:

$$\nabla \varphi = J_r(x)^T r(x).$$
For brevity: \( J := J_r(x) \). Can show similarly:

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

Newton step \( s \) can be found by solving

\[
H_\varphi(x)s = -\nabla \varphi
\]

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

**Gauss–Newton method:** Find step \( s \) by

\[
J^T Js = -\nabla \varphi = -J^T r(x)
\]

Does that remind you of the *normal equations*?

\[
Js \approx -r(x)
\]
Solve that using our existing methods for least-squares problems.

**Observations:** (from demo)

- Newton on its own is only locally convergent
- Gauss-Newton is clearly similar
- It’s worse because the step is only approximate
  → Much depends on the starting guess.

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

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

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

What Levenberg-Marquardt does is generically called ‘Regularization’: Make \(H\) more positive definite.
Demo: Gauss-Newton
6.4 Constrained Optimization
Constrained Optimization: Problem Setup

Want $x^*$ so that

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

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

Unconstrained necessary condition:

$$\nabla f(x) = 0$$

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

$s$ is a feasible direction at $x$ if

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

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

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

- At boundary: $g(x) = 0$. Need:
  $$-\nabla f(x) \in \text{rowspan}(J_g)$$
  a.k.a. "all descent directions would cause a change (→violation) of the constraints."

Q: Why ‘rowspan’? Think about shape of $J_g$.

$$\Leftrightarrow -\nabla f(x) = J_g^T \lambda \text{ for some } \lambda.$$
Lagrange Multipliers

- \[ g = 0 \]
- \[ \nabla g \]

- Seen: Need

\[ -\nabla f(x) = J_g^T \lambda \]

at the (constrained) optimum.

**Idea:** Turn constrained optimization problem for \( x \) into an *unconstrained* optimization problem for \( (x, \lambda) \). How?
Need a new function $\mathcal{L}(x, \lambda)$ to minimize:

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

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

$$0 = \nabla \mathcal{L} = \begin{pmatrix} \nabla_x \mathcal{L} \\ \nabla_\lambda \mathcal{L} \end{pmatrix} = \begin{pmatrix} \nabla f + J_g(x)^T \lambda \\ g(x) \end{pmatrix}.$$

Convenient: This matches our necessary condition!

So we could use any unconstrained method to minimized $\mathcal{L}$.

**For example:** Using Newton to minimize $\mathcal{L}$ is called *Sequential Quadratic Programming*. (‘SQP’)
Demo: Sequential Quadratic Programming
Inequality-Constrained Optimization

Want \( x^* \) so that

\[
f(x^*) = \min_x f(x) \quad \text{subject to} \quad g(x) = 0 \quad \text{and} \quad h(x) \leq 0
\]

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

Define Lagrangian:

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

- Some inequality constraints may not be “active” (active \( \iff h_i(x^*) = 0 \iff \) at ‘boundary’ of ineq. constraint) (Equality constraints are always ‘active’)
- If \( h_i \) inactive \( (h_i(x^*) < 0) \), must force \( \lambda_2,i = 0 \).
Otherwise: Behavior of $h$ could change location of minimum of $\mathcal{L}$. Use complementarity condition

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

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

$$\nabla_x \mathcal{L}(x^*, \lambda_1^*, \lambda_2^*) = 0$$

$$g(x^*) = 0$$

$$h(x^*) \leq 0$$

$$\lambda_2 \geq 0$$

$$h(x^*) \cdot \lambda_2 = 0$$

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

**Computational approach:** Solve (*) equations by Newton.
7 Interpolation
Interpolation: Setup

Given: \((x_i)_i^{N}, (y_i)_i^{N}\)

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:
Why is this important?

It brings all of calculus within range of numerical operations.

- Why?
  Because calculus works on functions.

- How?
  1. Interpolate (go from discrete to continuous)
2. Apply calculus
3. Re-discretize (evaluate at points)
Making the Interpolation Problem Unique

- 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 \( \varphi_i \).

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

  Interpolation becomes solving the linear system:

  \[
  y_i = f(x_i) = \sum_{j=0}^{N_{\text{func}}} \alpha_j \varphi_j(x_i) \quad \leftrightarrow \quad V\mathbf{\alpha} = \mathbf{y}.
  \]

  Want unique answer: Pick \( N_{\text{func}} = N \rightarrow V \) square.

  \( V \) is called the (generalized) Vandermonde matrix.
Main lesson:

\[ V(\text{coefficients}) = (\text{values at nodes}). \]
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 $1, x, x^2, x^3, x^4, ...$
- 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

● Why not equispaced?
  **Demo:** Choice of Nodes for Polynomial Interpolation
Lagrange Interpolation

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

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

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

$$
\varphi_1(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} \\
\varphi_2(x) = \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} \\
\varphi_3(x) = \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}
$$

Numerator: Ensures $\varphi_i$ 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 coefficient 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!

\[
\begin{align*}
\mathbf{f} \cdot \mathbf{g} & = \sum_{i=1}^{n} f_ig_i = \langle \mathbf{f}, \mathbf{g} \rangle \\
\langle \mathbf{f}, \mathbf{g} \rangle & = \int_{-1}^{1} f(x)g(x)dx
\end{align*}
\]

Orthogonal then just means \( \langle \mathbf{f}, \mathbf{g} \rangle = 0 \).

**Q:** How can we find an orthogonal basis?
Apply Gram-Schmidt to the monomials. Obtained **Legendre polynomials**.

**Demo:** Orthogonal polynomials

- But how can I practically compute the Legendre polynomials? → DLMF, Chapter on orthogonal polynomials

Main lessons:

- There exist three-term recurrences. Easy to apply if you know the first two.
- There is a whole zoo of polynomials there, depending on the weight function $w$ in the (generalized) inner product:

$$\langle f, g \rangle = \int w(x)f(x)g(x)dx.$$  

Some sets of orthogonal polynomials live on intervals other than $(-1, 1)$. 
Another family of orthogonal polynomials: Chebyshev

Three equivalent definitions:

- Result of Gram-Schmidt with weight $\frac{1}{\sqrt{1-x^2}}$
- What is that weight?
  - $\frac{1}{(\text{Half circle})}$, 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.)
- $T_k(x) = \cos(k \cos^{-1}(x))$
- $T_k(x) = 2x T_k(x) - T_{k-1}(x)$

**Demo:** Chebyshev interpolation part I

- 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 $\cos(*)$. 
- So why not \( \cos(\text{equispaced}) \)?

Might get

\[ x_i = \cos \left( \frac{i \pi}{k} \right) \quad (i = 0, 1, \ldots, k) \]

It turns out that these are just the **extrema** (minima/maxima) of \( T_k \).

In this case:

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

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

Might also consider 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 II

- Summary?

  - Chebyshev interpolation is fast and works extremely well
  - Scroll through ATAP
• In 1D, they’re a very good answer to the interpolation question
• But sometimes a piecewise approximation (with a specifiable level of smoothness) is more suited to the application
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 \( x_1 < \cdots < x_n \).

- \(|f^{(n)}(x)| \leq M\) for \( x \in [x_1, x_n] \),
- Set the interval length \( h = x_n - x_1 \).

Then \( |x - x_i| \leq h \).

Altogether—there is a constant \( C \) independent of \( h \) so that:

\[
\max_x |f(x) - p_{n-1}(x)| \leq CMh^n.
\]

For the grid spacing \( h \to 0 \), we have

\[
E(h) = O(h^n).
\]

This is called convergence of order \( n \).
Demo: Interpolation Error
**Going piecewise: Simplest Case**

Construct a piecewise linear interpolant at four points.

<table>
<thead>
<tr>
<th>$x_0, y_0$</th>
<th>$x_1, y_1$</th>
<th>$x_2, y_2$</th>
<th>$x_3, y_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1 = a_1x + b_1$</td>
<td>$f_2 = a_2x + b_2$</td>
<td>$f_3 = a_3x + b_3$</td>
<td></td>
</tr>
<tr>
<td>2 unk.</td>
<td>2 unk.</td>
<td>2 unk.</td>
<td></td>
</tr>
<tr>
<td>$f_1(x_0) = y_0$</td>
<td>$f_2(x_1) = y_1$</td>
<td>$f_3(x_2) = y_2$</td>
<td></td>
</tr>
<tr>
<td>2 eqn.</td>
<td>2 eqn.</td>
<td>2 eqn.</td>
<td></td>
</tr>
<tr>
<td>$f_1(x_1) = y_1$</td>
<td>$f_2(x_2) = y_2$</td>
<td>$f_3(x_3) = y_3$</td>
<td></td>
</tr>
<tr>
<td>2 eqn.</td>
<td>2 eqn.</td>
<td>2 eqn.</td>
<td></td>
</tr>
</tbody>
</table>

- Why three intervals?
  General situation → 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:

\[
\begin{align*}
&x_0, y_0 & x_1, y_1 & x_2, y_2 & x_3, y_3 \\
&f_1 = a_1x^3 + b_1x^2 + c_1x + d_1 & f_2 = a_2x^3 + b_2x^2 + c_2x + d_2 & f_3 = a_3x^3 + b_3x^2 + c_3x + d_3
\end{align*}
\]

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

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

\[
\begin{align*}
&f_1'(x_1) = f_2'(x_1) & f_2'(x_2) = f_3'(x_2) \\
&f_1''(x_1) = f_2''(x_1) & f_2''(x_2) = f_3''(x_2)
\end{align*}
\]

Not enough: need yet more conditions.

\[
\begin{align*}
&f_1'''(x_0) = 0 & f_3'''(x_3) = 0
\end{align*}
\]

Now: have a square system.
**Observation:** Number of conditions:

\[ 2N_{\text{intervals}} + 2N_{\text{middle nodes}} + 2 \]

where

\[ N_{\text{intervals}} - 1 = N_{\text{middle nodes}} \]

so

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

which is exactly the number of unknown coefficients.

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

Can also come up with a basis of spline functions (with the chosen smoothness conditions). ‘B-Splines’
8 Numerical Integration and Differentiation

8.1 Numerical Integration
Numerical Integration: About the Problem

- What is numerical integration? (Or ‘quadrature’?)

  Given $a$, $b$, $f$, compute
  \[ \int_{a}^{b} f(x) \, dx. \]

- 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.

\[
\left| \int_a^b f(x)\,dx - \int_a^b \hat{f}(x)\,dx \right| \\
= \left| \int_a^b e(x)\,dx \right| \leq \int_a^b |e(x)|\,dx \leq (b - a) \max_{x \in [a,b]} |e(x)|.
\]
8.1.1 Quadrature Methods
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.

\[
\int_a^b f(x) \, dx \approx \sum_{i=1}^{n} \omega_i f(x_i)
\]

Then: nodes \( (x_i) \) and weights \( (\omega_i) \) 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

\[
\int_a^b f(x) \, dx \approx \sum_i f(x_i) \int_a^b \ell_i(x) \, dx.
\]

- With polynomials and (often) equispaced nodes, this is called **Newton-Cotes quadrature**.

- With Chebyshev nodes and Chebyshev weights, this is called **Clenshaw-Curtis quadrature**.

**Q:** How do the weights actually get computed?

- Integrating an actual Lagrange polynomial: too much work.

- Instead: Solve linear system.
Know: This quadrature should at least integrate monomials correctly.

\[
b - a = \int_a^b 1 \, dx = \omega_1 \cdot 1 + \cdots + \omega_n \cdot 1
\]

\[
\vdots
\]

\[
\frac{1}{k+1}(b^{k+1} - a^{k+1}) = \int_a^b x^k \, dx = \omega_1 \cdot x_1^k + \cdots + \omega_n \cdot x_n^k
\]

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

This is called the method of undetermined coefficients.
Demo: Newton-Cotes weight finder
Examples and Exactness

- To what polynomial degree are the following rules exact?

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

  **Trapezoidal rule**
  \[\frac{b-a}{2}(f(a) + f(b))\]

  **Simpson’s rule**
  \[\frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)\]
Answers:

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

**Idea:** Could use difference between trapezoidal and midpoint rule as an error estimate.
8.1.2 Accuracy and Stability
Interpolatory Quadrature: Accuracy

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

Recall
\[
\sum_i \omega_i f(x_i) = \int_a^b p_{n-1}(x) \, dx.
\]

What can you say about the accuracy of the method?

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

\[
\left| \int_a^b f(x) \, dx - \int_a^b p_{n-1}(x) \, dx \right| \\
\leq \int_a^b |f(x) - p_{n-1}(x)| \, dx \\
\leq (b - a) \|f - p_{n-1}\|_\infty \\
\leq C(b - a) h^n \|f^{(n)}\|_\infty \\
\leq C h^{n+1} \|f^{(n)}\|_\infty
\]

(using interpolation error)
<table>
<thead>
<tr>
<th></th>
<th>( n )</th>
<th>( \text{Deg. (reg.)} )</th>
<th>( \text{Ex.Int.Deg. (w/odd)} )</th>
<th>( \text{Intp.Ord.} )</th>
<th>( \text{Quad.Ord. (regular)} )</th>
<th>( \text{Quad.Ord. (w/odd)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Midp.</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Trapz.</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Simps.</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>unnamed</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

- \( n \): number of points
- “Deg. (reg.)”: Degree of polynomial used in interpolation to build the quadrature rule. \( (=n - 1) \)
- “Ex.Int.Deg.”: Polynomials of up to (and including) this degree \( actually \) get integrated exactly. (including the odd-order bump) \( \left( =\begin{cases} n-1 & \text{even} \\ n & \text{odd} \end{cases} \right) \)
- “Intp.Ord.”: Order of Accuracy of Interpolation: \( O(h^n) \)
• “Quad. Ord. (regular)”: Order of accuracy for quadrature predicted by the error result above: $O(h^{n+1})$

• “Quad. Ord. (w/odd)”: Actual order of accuracy for quadrature given ‘bonus’ degrees for rules with odd point count

\[
\left\{
\begin{array}{ll}
O(h^{n+1}) & \text{even} \\
O(h^{n+2}) & \text{odd}
\end{array}
\right.
\]

Observation: Quadrature gets (at least) ‘one order higher’ than interpolation—even more for odd-order rules. (i.e. more accurate)
Demo: Accuracy of Newton-Cotes
Interpolatory Quadrature: Stability

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

Recall

\[ \sum_i \omega_i f(x_i) = \int_a^b p_n(x) \, dx \]

What can you say about the stability of this method?

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

\[
\left| \sum_i \omega_i f(x_i) - \sum_i \omega_i \hat{f}(x_i) \right| \\
= \left| \sum_i \omega_i e(x_i) \right| \leq \sum_i |\omega_i e(x_i)| \\
\leq \left( \sum_i |\omega_i| \right) \| e \|_\infty
\]
Q: So, what quadrature weights make for bad stability bounds?
A: Quadratures with large negative weights. (Recall: $\sum_i \omega_i$ is fixed.)
About Newton-Cotes

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

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

More drawbacks:

- All the fun of high-order interpolation with monomials and equi-spaced 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.
8.1.3 Composite Quadrature

- High-order polynomial interpolation requires a high degree of smoothness of the function.

**Idea:** Stitch together multiple lower-order quadrature rules to alleviate smoothness requirement.

- e.g. trapezoidal

What can we say about the error in this case?

**Recall:** error for a one panel of length $h$: $\left| \int f - p_{n-1} \right| \leq C \cdot h^{n+1} \| f^{(n)} \|_{\infty}$
\[ \leq C \| f^{(n)} \|_\infty \sum_{j=1}^{m} (a_{j+1} - a_j)^{n+1} \]

\[ = C \| f^{(n)} \|_\infty (a_{j+1} - a_j)^n \sum_{j=1}^{m} (a_{j+1} - a_j) \]

\[ = C \| f^{(n)} \|_\infty h^n (b - a), \]

where \( h \) is now the length of a single panel.

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

**Idea:** If we can estimate errors on each subinterval, we can shrink (e.g. by splitting in half) only those contributing the most to the error. (‘adaptive quadrature’, → hw)
8.1.4 Gaussian Quadrature

- 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
8.2 Numerical Differentiation
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:

\[ f(x + h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2} + \ldots \]
Plug in:

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

→ first order accurate.

Similarly:

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

('Centered differences')

Can also take higher order derivatives:

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

Can find these by trying to match Taylor terms.
Alternative: Use linear algebra with interpolate-then-differentiate to find FD formulas.
Demo: Finite Differences vs Noise
Demo: Floating point vs Finite Differences
8.3 Richardson Extrapolation

- If we have two estimates of something, can we get a third that’s more accurate? Suppose we have an approximation

\[ F = \tilde{F}(h) + O(h^p) \]

and we know \( \tilde{F}(h_1) \) and \( \tilde{F}(h_2) \).

Grab one more term of the Taylor series:

\[ F = \tilde{F}(h) + a \, h^p + O(h^q) \]

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 $\alpha + \beta = 1$.

$$\alpha (h_1^p - h_2^p) + 1 h_2^p = 0$$

$$\alpha = \frac{-h_2^p}{h_1^p - h_2^p}$$

Important observation: Never needed to know $a$.

**Idea:** Can repeat this for even higher accuracy.

e.g. 1st 2nd 3rd 4th order accurate
Carrying out this process for quadrature is called Romberg integration.
Demo: Richardson with Finite Differences
Initial Value Problems for ODEs
What can we solve?

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

Systems with derivatives

- ODEs ($\partial$ in 1D)
- PDEs ($\partial$ in $n$D) (11)

Initial Value Prob. (9)  Boundary Value Prob. (10)
## Some Applications

<table>
<thead>
<tr>
<th>ODEs</th>
<th>IVPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Population dynamics</td>
<td>• bridge load</td>
</tr>
<tr>
<td>[ y_1' = y_1(\alpha_1 - \beta_1 y_2) ] (prey)</td>
<td>• pollutant concentration</td>
</tr>
<tr>
<td>[ y_2' = y_2(-\alpha_2 + \beta_2 y_1) ] (predator)</td>
<td>(steady state)</td>
</tr>
<tr>
<td>• chemical reactions</td>
<td>• temperature</td>
</tr>
<tr>
<td>• equations of motion</td>
<td>(steady state)</td>
</tr>
</tbody>
</table>

- Population dynamics
- chemical reactions
- equations of motion
**Initial Value Problems: Problem Statement**

- Want: Function \( y: [0, T] \to \mathbb{R}^n \) so that
  
  - \( y^{(k)}(t) = f(t, y, y', y'', ..., y^{(k-1)}) \) (explicit)

  or

  - \( f(t, y, y', y'', ..., y^{(k)}) = 0 \) (implicit)

  are called explicit/implicit \( k \)th-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?)

  \[ y(0) = g_0, \quad y'(0) = g_1, ... \quad y^{(k-1)}(0) = g_{k-1}. \]

  Boundary Value Problems (BVPs) trade some derivatives for conditions at the ‘other end’.
Reducing ODEs to First-Order Form

- A $k$th 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:
  \[
  y_0'(t) = 1, \quad y_0(0) = 0.
  \]
  \( \rightarrow \) Without loss of generality: Get rid of explicit \( t \) dependency.

- What is a linear ODE?
  \[
  f(t, x) = A(t)x + b
  \]

- What is a linear and homogeneous ODE?
  \[
  f(t, x) = A(t)x
  \]

- What is a constant-coefficient ODE?
  \[
  f(t, x) = Ax
  \]
9.1 Existence, Uniqueness, Conditioning
Existence and Uniqueness

Consider the perturbed problem

\[
\begin{align*}
\begin{cases}
  y'(t) &= f(y) \\
  y(t_0) &= y_0
\end{cases} & \quad \begin{cases}
  \hat{y}'(t) &= f(\hat{y}) \\
  \hat{y}(t_0) &= \hat{y}_0
\end{cases}
\end{align*}
\]

Then if \( f \) is Lipschitz continuous (has ‘bounded slope’), i.e.

\[
\|f(y) - f(\hat{y})\| \leq L \|y - \hat{y}\|
\]

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

- there exists a solution \( y \) in a neighborhood of \( t_0 \), and...
  
  \[
  \|y(t) - \hat{y}(t)\| \leq e^{L(t-t_0)} \|y_0 - \hat{y}_0\|
  \]

- 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 continuously dependent on the initial condition, i.e.
  
  For all $\varepsilon > 0$ there exists a $\delta > 0$ so that
  
  $$
  \|\hat{y}_0 - y_0\| < \delta \quad \Rightarrow \quad \|\hat{y}(t) - y(t)\| < \varepsilon \quad \text{for all } t \geq t_0.
  $$

- An ODE is **asymptotically stable** if and only if
  
  $$
  \|\hat{y}(t) - y(t)\| \to 0 \quad (t \to \infty).
  $$
Example I: Scalar, Constant-Coefficient

\[
\begin{aligned}
\begin{cases}
y'(t) = \lambda y \\
y(0) = y_0
\end{cases}
\quad \text{where} \quad \lambda = a + i b
\end{aligned}
\]

- Solution?

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

- When is this stable?
  
  When \( a = \text{Re} \lambda > 0 \): \hspace{1cm} \text{When } a = \text{Re} \lambda \leq 0 :
Example II: Constant-Coefficient System

\[
\begin{align*}
    \begin{cases}
        y'(t) &= Ay(t) \\
        y(t_0) &= y_0
    \end{cases}
\end{align*}
\]

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

- How do we find a solution?

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

\[
    w'(t) = V^{-1}y'(t) = V^{-1}Ay(t) = V^{-1}AVw(t) = Dw(t).
\]

Now: \( n \) decoupled IVPs (with \( w_0 = V^{-1}y_0 \)) \( \rightarrow \) Solve as in scalar case. Find \( y(t) = Vw(t) \).

- When is this stable?
  When \( \text{Re} \lambda_i \leq 0 \) for all eigenvalues \( \lambda_i \).
9.2 Numerical Methods (I)
Euler’s Method

- Discretize the IVP

\[
\begin{align*}
y'(t) &= f(y) \\
y(t_0) &= y_0
\end{align*}
\]

- Discrete times: \( t_1, t_2, \ldots \), with \( t_{i+1} = t_i + h \)

- Discrete function values: \( y_k \approx y(t_k) \).

**Idea:** Rewrite the IVP in integral form:

\[
y(t) = y_0 + \int_{t_0}^{t} f(y(\tau))d\tau,
\]

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

- Using ‘left rectangle rule’:

\[
y_{k+1} = y_k + hf(y_k)
\]
Time advancement can be computed simply by evaluating the \textit{RHS formula}. A method like that is called \textit{explicit}. This particular method is called `Forward Euler’.

- Using ‘right rectangle rule’:

\[
y_{k+1} = y_k + hf(y_{k+1})
\]

Time advancement must be computed by \textit{solving a system of equations}. A method like that is called \textit{implicit}. This particular method is called `Backward Euler’.
Demo: Forward Euler stability
9.3 Accuracy and Stability
Global and Local Error

Let $u_k(t)$ be the function that solves the ODE with the initial condition $u_k(t_k) = y_k$.

- Define the local error at step $k$ as...
  \[ \ell_k = y_k - u_{k-1}(t_k) \]
- Define the global error at step $k$ as...
  \[ g_k = y(t_k) - y_k \]
About Local and Global Error

- Is global error $= \sum$ local errors?

No.
Consider an analogy with interest rates—at any given moment, you receive 5% interest ($\sim$ incur 5% error) on your current balance. But your current balance includes prior interest (error from prior steps), which yields more interest (in turn contributes to the error).

This contribution to the error is called propagated error.

The local error is much easier to estimate $\rightarrow$ will focus on that.

- A time integrator is said to be accurate of order $p$ if...
  $\ell_k = O(h^{p+1})$
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

$$\frac{1}{h} \cdot O(h^{p+1}) = O(h^p).$$

(Note that this ignores ‘accrual’ of propagated error.)
Stability of a Method

- Find out when forward Euler is stable when applied to

\[
y'(t) = \lambda y(t).
\]

\[
y_k = y_{k-1} + h\lambda y_{k-1}
\]

\[
= (1 + h\lambda)y_{k-1}
\]
So: stable $\Leftrightarrow |1 + h\lambda| \leq 1$.

$|1 + h\lambda|$ 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.
  \[ y'(t) = A y(t)? \]

  1. Diagonalize system as before
  2. Notice that same \( V \) also diagonalizes the time stepper
  3. apply scalar analysis to components.

→ Stable if \( |1 + h \lambda_i| \leq 1 \) for all eigenvalues \( \lambda_i \).

- What about stability for nonlinear systems, i.e.
  \[ y'(t) = f(y(t))? \]

Consider perturbation \( e(t) = y(t) - \hat{y}(t) \). Linearize:

\[ e'(t) = f(y(t)) - f(\hat{y}(t)) \approx J_f(y(t))e(t) \]
I.e. can (at least locally) apply analysis for linear systems to the nonlinear case.
Stability for Backward Euler

- Find out when backward Euler is stable when applied to

\[ y'(t) = \lambda y(t). \]

\[
\begin{align*}
  y_k &= y_{k-1} + h\lambda y_k \\
  y_k(1 - h\lambda) &= y_{k-1} \\
  y_k &= \frac{1}{1 - h\lambda} y_{k-1} \\
  &= \left( \frac{1}{1 - h\lambda} \right)^k y_0.
\end{align*}
\]

So: stable \( \iff |1 - h\lambda| \geq 1. \)

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

- For explicit methods, main concern in time step selection is stability (but also accuracy).
- For implicit methods, main concern in time step selection is accuracy.
Demo: Backward Euler stability
9.4 Stiffness
Demo: Stiffness
‘Stiff’ ODEs

- Stiff problems have multiple time scales.
  (In the example above: Fast decay, slow evolution.)
- In the case of a stable ODE system

\[ y'(t) = f(y(t)), \]

stiffness can arise if \( J_f \) has eigenvalues of very different magnitude.
• Why not just ‘small’ or ‘large’ magnitude? Because the discrepancy between time scales is the root of the problem. If all time scales are similar, then time integration must simply ‘deal with’ that one time scale. If there are two, then some (usually the fast ones) may be considered uninteresting.

• What is the problem with applying explicit methods to stiff problems? Fastest time scale governs time step → tiny time step → inefficient.

• Phrase this as a conflict between accuracy and stability.
  • Accuracy (here: capturing the slow time scale) could be achieved with large time steps.
  • Stability (in explicit methods) demands a small time step.

• Can an implicit method take arbitrarily large time steps?
In terms of stability: sure.
In terms of accuracy: no.
9.5 Numerical Methods (II)
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’:

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

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:

|   | \(a_{11}\) & \(\cdots\) & \(a_{1s}\) |
|---|---|---|
|\(c_1\) | \(\vdots\) & \(\vdots\) & \(\vdots\) |
|\(c_s\) | \(a_{s1}\) & \(\cdots\) & \(a_{ss}\) |
|   | \(b_1\) & \(\cdots\) & \(b_s\) |

- When is an RK method explicit?
If the diagonal entries in the Butcher tableau and everything above it are zero.

- When is it implicit? (Otherwise)

- When is it diagonally implicit? (And what does that mean?) If the everything above the diagonal entries in the Butcher tableau is zero. This means that one can solve for one stage value at a time (and not multiple).

- Stuff Heun’s method into a Butcher tableau:
  1. \( \tilde{y}_{k+1} = y_k + hf(y_k) \)
  2. \( y_{k+1} = y_k + \frac{h}{2}(f(y_k) + f(\tilde{y}_{k+1})) \).
What is RK4?
(See Wikipedia page, note similarity to Simpson’s rule.)
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
10 Boundary Value Problems for ODEs
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** \( u'(a) = \nu_a \)
- **or Robin** \( \alpha u(a) + \beta u'(a) = \omega_a \)

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:
\[ y'(x) = f(y(x)) \quad f: \mathbb{R}^n \rightarrow \mathbb{R}^n \]

BCs:
\[ g(y(a), y(b)) = 0 \quad g: \mathbb{R}^{2n} \rightarrow \mathbb{R}^n \]

(Recall the rewriting procedure to first-order for any-order ODEs.)

- Does a first-order, scalar BVP make sense?
  No—need second order (or \( n \geq 2 \)) to allow two boundary conditions.

**Example:** Linear BCs
\[ B_a y(a) + B_b y(b) = c \]

- 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.
10.1 Existence, Uniqueness, Conditioning
Does a solution even exist? How sensitive are they?

General case is harder than root finding, and we couldn’t say much there. → Only consider linear BVP.

\[
(*) \begin{cases}
    y'(x) = A(x)y(x) + b(x) \\
    B_ay(a) + B_by(b) = c
\end{cases}
\]

To solve that, consider homogeneous IVP

\[
y_i'(x) = A(x)y_i(x)
\]

with initial condition

\[
y_i(a) = e_i.
\]

Note: \( y \neq y_i \). \( e_i \) is the \( i \)th unit vector. With that, build fundamental solution matrix

\[
Y(x) = \begin{pmatrix}
    y_1 & \cdots & y_n
\end{pmatrix}
\]
Let

\[ Q := B_a Y(a) + B_b Y(b) \]

Then (*) has a unique solution if and only if \( Q \) is invertible. Solve to find coefficients:

\[ Q\alpha = c \]

Then \( Y(x)\alpha \) solves (*) with \( b(x) = 0 \).

Define \( \Phi(x) := Y(x) Q^{-1} \). So \( \Phi(x)c \) solves (*) with \( b(x) = 0 \).
Define Green’s function

\[ G(x, y) := \begin{cases} 
\Phi(x)B_a\Phi(a)\Phi^{-1}(y) & y \leq x, \\
-\Phi(x)B_b\Phi(b)\Phi^{-1}(y) & y > x. 
\end{cases} \]

Then

\[ y(x) = \Phi(x)c + \int_a^b G(x, y)b(y)dy. \]

**Conditioning:**
Now easy. For perturbed problem with \( b(x) + \Delta b(x) \) and \( c + \Delta c \):

\[ \|\Delta y\|_{\infty} \leq \max(\|\Phi\|_{\infty}, \|G\|_{\infty}) \left( \|\Delta c\|_1 + \int \|\Delta b(y)\|_1dy \right). \]

- Did not prove uniqueness. (But true.)
- Also get continuous dependence on data.
- Can verify that above formula solves (*) by plug’n’chug.
10.2 Numerical Methods
**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

\[
\begin{align*}
    u'(x) &= \frac{u(x + h) - u(x - h)}{2h} + O(h^2) \\
    u''(x) &= \frac{u(x + h) - 2u(x) + u(x - h)}{h^2} + O(h^2)
\end{align*}
\]

**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.

\[ \implies \text{Get a big (non-)linear system} \]

3. Solve that (LU/Newton)\( \implies \text{done.} \)
Galerkin/Finite Element Method

\[ u''(x) = f(x), \quad u(a) = u(b) = 0. \]

**Problem** with collocation: Big dense matrix.
**Idea:** Use piecewise basis. Maybe it’ll be sparse.

- What’s the problem with that?
  - \( u' \) does not exist. (at least at a few points where it’s discontinuous)
  - \( u'' \) really does not exist.
**Idea:** Enforce a ‘weaker’ version of the ODE. Compute ‘moments’:
\[
\int_a^b u''(x) \psi(x) dx = \int_a^b f(x) \psi(x) dx
\]

Require that this holds for some test functions \( \psi \) from some set \( W \). Now possible to get rid of (undefined) second derivative using integration by parts:
\[
\int_a^b u''(x) \psi(x) dx = [u'(x) \psi(x)]_a^b - \int_a^b u'(x) \psi'(x) dx.
\]

Make some choices:
- Solve for \( u \in \text{span}\{\text{hat functions } \varphi_i\} \)
- Choose \( \psi \in W = \text{span}\{\text{hat functions } \varphi_i\} \) with \( \psi(a) = \psi(b) = 0 \).
  \[ \to \text{Kills boundary term } [u'(x) \psi(x)]_a^b. \]

These choices are called the **Galerkin method**. Also works with other bases.
Assemble:

\[
- \int_a^b u'(x) \psi'(x) \, dx = \int_a^b f(x) \psi(x) \, dx \\
- \int_a^b \left[ \sum_{j=1}^{n} \alpha_j \varphi_j'(x) \right] \psi'(x) \, dx = \int_a^b f(x) \psi(x) \, dx \\
- \sum_{j=1}^{n} \alpha_j \int_a^b \varphi_j'(x) \varphi_i'(x) \, dx = \int_a^b f(x) \varphi_i(x) \, dx
\]

\[S \alpha = r.\]

**Now:** Compute \( S \), solve sparse (!) linear system.
11 Partial Differential Equations and Sparse Linear Algebra
Remark: Both PDEs and Sparse Linear Algebra are big topics. Will only scratch the surface here. Want to know more?

- CS555 → Numerical Methods for PDEs
- CS556 → Iterative and Multigrid Methods

We would love to see you there! :)}
11.1 Sparse Linear Algebra
Solving Sparse Linear Systems

Solving $Ax = b$ has been our bread and butter.

Typical approach: Use factorization (like LU or Cholesky)
Why is this problematic?

**Demo:** Sparse Matrix Factorizations and “Fill-In”

**Idea:** Don’t factorize, iterate.
‘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?

\[
\begin{align*}
Ax &= b \\
Mx &= Nx + b \\
Mx_{k+1} &= Nx_k + b \\
x_{k+1} &= M^{-1}(Nx_k + b)
\end{align*}
\]

- These methods are called *stationary* because they do the same thing in every iteration.

- They carry out fixed point iteration.
Converge if contractive, i.e. $\rho(M^{-1}N) < 1$.

- Choose $M$ so that it’s easy to invert.

- What could we choose for $M$ (so that it’s easy to invert)?

<table>
<thead>
<tr>
<th>Name</th>
<th>$M$</th>
<th>$N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jacobi</td>
<td>$D$</td>
<td>$-(L + U)$</td>
</tr>
<tr>
<td>Gauss-Seidel</td>
<td>$D + L$</td>
<td>$-U$</td>
</tr>
<tr>
<td>SOR</td>
<td>$\frac{1}{\omega}D + L$</td>
<td>$(\frac{1}{\omega} - 1)D - U$</td>
</tr>
</tbody>
</table>

where $L$ is the below-diagonal part of $A$, and $U$ the above-diagonal.
Demo: Stationary Methods
Conjugate Gradient Method

Assume $A$ is symmetric positive definite.

**Idea:** View solving $Ax = b$ as an optimization problem.

Minimize $\varphi(x) = \frac{1}{2} x^T Ax - x^T b \iff$ Solve $Ax = b$.

Observe $-\nabla \varphi(x) = b - Ax = r$ (residual).

Use an iterative procedure ($s_k$ is the search direction):

$x_0 = \langle \text{starting vector} \rangle$

$x_{k+1} = x_k + \alpha_k s_k,$

- What should we choose for $\alpha_k$ (assuming we know $s_k$)?

$$0 = \frac{\partial}{\partial \alpha} \varphi(x_k + \alpha_k s_k)$$

$$= \nabla \varphi(x_{k+1}) \cdot s_k = r_{k+1} \cdot s_k.$$
**Learned:** Choose $\alpha$ so that next residual is $\perp$ to current search direction.

\[
\begin{align*}
    r_{k+1} &= r_k + \alpha_k As_k \\
    0 &= s_k^T r_{k+1} = s_k^T r_k + \alpha_k s_k^T As_k
\end{align*}
\]

Solve:

\[
\alpha_k = \frac{s_k^T r_k}{s_k^T As_k} = -\frac{s_k^T A e_k}{s_k^T As_k}, \quad (*)
\]

where $e_k = x_k - x^*$ and $r_k = -A e_k$.

- What should we choose for $s_k$?
  
  **Idea:** $s_k = r_k = -\nabla \varphi(x_k)$, i.e. steepest descent. No—still a bad idea.

  
  $x, y$ are called *A-orthogonal* or *conjugate* if and only if $x^T A y = 0$.

  **Better Idea:** Require $s_i^T A s_j = 0$ if $i \neq j$. 
View error as linear combination of search directions, with some coefficients:

\[ e_0 = x_0 - x^* = \sum_i \delta_i s_i. \]

- We run out of \( A \)-orthogonal directions after \( n \) iterations.
- Is the error going to be zero then? If \( \delta_k = -\alpha_k \), then yes.

\[ s_k^T A e_0 = \sum_i \delta_i s_k^T A s_i = \delta_k s_k^T A s_k. \]

Then

\[ \delta_k = \frac{s_k^T A e_0}{s_k^T A s_k} = \frac{s_k A (e_0 + \sum_{i=1}^{k-1} \alpha_i s_i)}{s_k^T A s_k} = \frac{s_k^T A e_k}{s_k^T A s_k} = -\alpha_k. \]

How do we generate the \( s_k \)?

- Pick a random one to start with. Perhaps \( r_k \)?
• Generate next one by orthogonalizing from Krylov space procedure

\[ z, Az, A^2z \]

Awesome insight: Can use three-term Lanczos iteration to generate those. \( \rightarrow \) cheap!
Demo: Conjugate Gradient Method
11.2 PDEs
Introduction

Notation:

\[ \frac{\partial}{\partial x} u = \partial_x u = u_x. \]

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.
• Sometimes one variable is time-like. What makes a variable time-like?
  ○ Causality
  ○ No geometry

Have:
• PDE
• Boundary conditions
• Initial conditions (in $t$)
Time-dependent PDEs give rise to a **steady-state** PDE:

\[ u_t = f(u_x, u_y, u_{xx}, u_{yy}) \quad \rightarrow \quad 0 = f(u_x, u_y, u_{xx}, u_{yy}) \]

Idea for time-dep problems (‘Method of Lines’):

- Discretize spatial derivatives first
- Obtain large (‘semidiscrete’) system of ODEs
- Use ODE solver from Chapter 9

**Demo:** Time-dependent PDEs

**Notation:** Laplacian (dimension-independent)

\[ \Delta u = \text{div } \text{grad } u = \nabla \cdot (\nabla u) = u_{xx} + u_{yy} \]

Three main types of PDEs:

- **hyperbolic** (wave-like, conserve energy)
  - first-order **conservation laws**: \( u_t + f(u)_x = 0 \)
second-order wave equation: $u_{tt} = \Delta u$

- parabolic (heat-like, dissipate energy)
  - heat equation: $u_t = \Delta u$

- elliptic (steady-state, of heat and wave eq. for example)
  - Laplace equation $\Delta u = 0$
  - Poisson equation $\Delta u = f$

(Pure BVP, similar to 1D BVPs, same methods apply—FD, Galerkin, etc.)