Outline

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:** Python
- **Demo:** numpy
- **In-class activity:** Image Processing

Outline

Why polynomials?

$$a_3x^3 + a_2x^2 + a_1x + a_0$$

How do we write a general degree n polynomial?

$$\sum_{i=0}^{n} a_i x^i.$$

Why polynomials and not something else?

- We can add, multiply, maybe divide (grade school, computer HW)
- More complicated functions (e^x, $\sin x$, $\sqrt{x}$) have to be *built* from those parts $\to$ at least approximately

Why polynomials? (II)

- Easy to work with as a building block.
  General recipe for numerics: Model *observation* with a polynomial, perform operation on that, evaluate. (e.g. calculus, root finding)

Reconstructing a Function From Derivatives

Given $f(x_0), f'(x_0), f''(x_0), \ldots$ can we reconstruct a polynomial f?

$$f(x) = ??? + ???x + ???x^2 + \cdots$$

For simplicity, let us consider $x_0 = 0$ and a degree 4 polynomial f

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

Note that $a_0 = f(0)$. Now, take a derivative

$$f'(x) = a_1 + 2a_2 x + 3a_3 x^2 + 4a_4 x^3$$

we see that $a_1 = f'(0)$, differntiating again

$$f''(x) = 2a_2 + 3 \cdot 2a_3 x + 4 \cdot 3a_4 x^2$$

yields $a_2 = f''(0)/2$, and finally

$$f'''(x) = 3 \cdot 2a_3 + 4 \cdot 3 \cdot 2a_4 x$$

yields $a_3 = f'''(0)/3!$. In general, $a_i = f^{(i)}(0)/i!$.

Reconstructing a Function From Derivatives

Found: Taylor series approximation.

$$f(0+x) \approx f(0) + f'(0)x + \frac{f''(0)}{2}x^2 + \cdots$$

The general Taylor expansion with center $x_0 = 0$ is

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!} x^i$$

Demo: Polynomial Approximation with Derivatives (click to visit) (Part I)

Shifting the Expansion Center

Can you do this at points other than the origin?

In this case, 0 is the center of the series expansion. We can obtain a Taylor expansion of f centered at x_0 by defining a shifted function

$$g(x) = f(x + x_0)$$

The Taylor expansion g with center 0 is as before

$$g(x) = \sum_{i=0}^{\infty} \frac{g^{(i)}(0)}{i!} x^i$$

This expansion of g yields a Taylor expansion of f with center x_0

$$f(x) = g(x - x_0) = \sum_{i=0}^{\infty} \frac{g^{(i)}(0)}{i!} (x - x_0)^i$$

Shifting the Expansion Center (II)

It suffices to note that $f^{(i)}(x_0) = g^{(i)}(0)$ to get the general form

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i \qquad \text{or} \qquad f(x_0 + h) = \sum_{i=0}^{\infty} \frac{f^{(i)}(x_0)}{i!} h^i$$

Errors in Taylor Approximation (I)

Can't sum infinitely many terms. Have to truncate. How big of an error does this cause?

Demo: Polynomial Approximation with Derivatives (click to visit) (Part II)

Suspicion, as $h \to 0$, we have

$$\underbrace{\left| f(x_0 + h) - \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!} h^i \right|}_{\text{Taylor error for degree } n} \leqslant C \cdot h^{n+1}$$

or

$$\underbrace{\left| f(x_0 + h) - \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!} h^i \right|}_{\text{Taylor error for degree } n} = O(h^{n+1})$$

As we will see, there is a satisfactory bound on C for most functions f.

Making Predictions with Taylor Truncation Error

Suppose you expand $\sqrt{x-10}$ in a Taylor polynomial of degree 3 about the center $x_0 = 12$. For $h_1 = 0.5$, you find that the Taylor truncation error is about 10^{-4}.

What is the Taylor truncation error for $h_2 = 0.25$?

$\text{Error}(h) = O(h^{n+1})$, where $n = 3$, i.e.

$$\text{Error}(h_1) \approx C \cdot h_1^4$$

$$\text{Error}(h_2) \approx C \cdot h_2^4$$

While not knowing C or lower order terms, we can use the ratio of h_2/h_1

$$\text{Error}(h_2) \approx C \cdot h_2^4 = C \cdot h_1^4 \left(\frac{h_2}{h_1}\right)^4 \approx \text{Error}(h_1) \cdot \left(\frac{h_2}{h_1}\right)^4$$

Can make prediction of the error for one h if we know another.

Demo: Polynomial Approximation with Derivatives (click to visit) (Part III)

Taylor Remainders: the Full Truth

Let $f : \mathbb{R} \to \mathbb{R}$ be $(n+1)$-times differentiable on the interval (x_0, x) with $f^{(n)}$ continuous on $[x_0, x]$. Then there exists a $\xi \in (x_0, x)$ so that

$$f(x_0+h) - \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!} h^i = \underbrace{\frac{f^{(n+1)}(\xi)}{(n+1)!}}_{\text{“}C\text{”}} \cdot (\xi - x_0)^{n+1}$$

and since $|\xi - x_0| \leqslant h$

$$\left| f(x_0+h) - \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!} h^i \right| \leqslant \underbrace{\frac{\left|f^{(n+1)}(\xi)\right|}{(n+1)!}}_{\text{“}C\text{”}} \cdot h^{n+1}.$$

Intuition for Taylor Remainder Theorem

Given the value of a function and its derivative $f(x_0), f'(x_0)$, prove the Taylor error bound.

We express $f(x)$ as

$$f(x) = f(x_0) + \int_{x_0}^{x} f'(w_0)dw_0$$

We then express the integrand $f'(w_0)$ for $w_0 \in [x_0, x]$ using f'' in the same way

$$\begin{aligned} f(x) &= f(x_0) + \int_{x_0}^{x} \left(f'(x_0) + \int_{x_0}^{w_0} f''(w_1)dw_1 \right) dw_0 \\ &= \underbrace{f(x_0) + f'(x_0)(x - x_0)}_{\text{Taylor expansion } t_1(x)} + \int_{x_0}^{x} \left(\int_{x_0}^{w_0} f''(w_1)dw_1 \right) dw_0 \end{aligned}$$

Intuition for Taylor Remainder Theorem (II)

We can bound the error by finding the maximizer $\xi = \operatorname{argmax}_{\xi \in [x_0, x]} |f''(\xi)|$

$$\begin{aligned} |f(x) - t_1(x)| &\le \int_{x_0}^{x} \left(\int_{x_0}^{w_0} |f''(\xi)| dw_1 \right) dw_0 \\ &= \frac{|f''(\xi)|}{2!}(x - x_0)^2 \\ |f(x_0 + h) - t_1(x_0 + h)| &\le \frac{|f''(\xi)|}{2!} h^2 \quad \text{for } x = x_0 + h \end{aligned}$$

In-class activity: Taylor series

Proof of Taylor Remainder Theorem

We can complete the proof by induction of the Taylor degree expansion n

$$f(x) = t_{n-1}(x) + \int_{x_0}^{x} \int_{x_0}^{w_0} \cdots \int_{x_0}^{w_{n-1}} f^{(n)}(w_n) dw_n \cdots dw_0$$

Given the formula above, it suffices to expand

$$f^{(n)}(w_n) = f^{(n)}(x_0) + \int_{x_0}^{w_n} f^{(n+1)}(w_{n+1}) dw_{n+1}$$

Inserting this back in gives us the next inductive hypothesis

$$f(x) = \underbrace{t_{n-1}(x) + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n}_{\text{Taylor expansion } t_n(x)} + \int_{x_0}^{x} \int_{x_0}^{w_0} \cdots \int_{x_0}^{w_n} f^{(n+1)}(w_{n+1}) dw_{n+1} \cdots dw_0$$

Proof of Taylor Remainder Theorem (II)

The bound follows as for $n = 1$, with $\xi = \operatorname{argmax}_{\xi \in [x_0, x]} |f^{(n+1)}(\xi)|$

$$\begin{aligned}
|f(x) - t_n(x)| &\leq \int_{x_0}^{x} \int_{x_0}^{w_0} \cdots \int_{x_0}^{w_n} |f^{(n+1)}(\xi)| dw_{n+1} \cdots dw_0 \\
&= \frac{|f''(\xi)|}{(n+1)!} (x - x_0)^{n+1} \\
|f(x_0 + h) - t_n(x_0 + h)| &\leq \frac{|f''(\xi)|}{(n+1)!} h^{n+1} \quad \text{for } x = x_0 + h
\end{aligned}$$

Using Polynomial Approximation

Suppose we can approximate a function as a polynomial:

$$f(x) \approx a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

How is that useful?
E.g.: What if we want the integral of f?

Easy: Just integrate the polynomial:

$$\begin{aligned}\int_s^t f(x)dx &\approx \int_s^t a_0 + a_1 x + a_2 x^2 + a_3 x^3 dx \\ &= a_0 \int_s^t 1dx + a_1 \int_s^t x \cdot dx + a_2 \int_s^t x^2 dx + a_3 \int_s^t x^3 dx\end{aligned}$$

Even if you had *no idea* how to integrate f using calculus, you can *approximately* integrate f anyway, by taking a bunch of derivatives (and forming a Taylor polynomial).

Demo: Computing Pi with Taylor (click to visit)

Outline

Reconstructing a Function From Point Values

If we know function values at some points $f(x_1), f(x_2), \ldots, f(x_n)$, can we reconstruct the function as a polynomial?

$$f(x) = ??? + ???x + ???x^2 + \cdots$$

In particular, we'd like to obtain $a_1, a_2, \ldots$ to satisfy equations:

$$\begin{array}{rcl} a_0 + a_1 \cdot x_1 + a_2 \cdot x_1^2 + \cdots & = & f(x_1) \\ \vdots & & \vdots \\ a_0 + a_1 \cdot x_n + a_2 \cdot x_n^2 + \cdots & = & f(x_n) \end{array}$$

Q: How many a_i can we (uniquely) determine this way? (Answer: n–same number of unknowns as equations.)

Reconstructing a Function From Point Values (II)

The equations for these n-unkowns are

$$\begin{array}{rcl} a_0 + a_1 \cdot x_1 + a_2 \cdot x_1^2 + \cdots + a_{n-1} \cdot x_1^{n-1} & = & f(x_1) \\ \vdots & & \vdots \\ a_0 + a_1 \cdot x_n + a_2 \cdot x_n^2 + \cdots + a_{n-1} \cdot x_n^{n-1} & = & f(x_n) \end{array}$$

We can rewrite them in matrix form

$$\underbrace{\begin{pmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ 1 & x_2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^{n-1} \end{pmatrix}}_{V} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}}_{\boldsymbol{a}} = \underbrace{\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix}}_{\boldsymbol{f}}.$$

Reconstructing a Function From Point Values (III)

Thus, to find $(a_0, \ldots, a_n)$ we need to *solve a linear system*. This gives us a degree $n-1$ polynomial interpolant satisfying $\tilde{f}(x_i) = f(x_i)$ of the form

$$\tilde{f}(x) = a_0 + a_1 \cdot x_n + a_2 \cdot x_n^2 + \cdots + a_{n-1} \cdot x_n^{n-1}$$

Vandermonde Linear Systems

Polynomial interpolation is a critical component in many numerical models.

V is called the Vandermonde matrix.

Main lesson:

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

This is called interpolation. $x_0, \ldots, x_{n-1}$ are called the nodes.

The problem also has surprisingly 'interesting' numerical properties.

In other words, while solving this linear system may *look* bulletproof, it can break in a number of 'interesting' ways. We will study these in detail later.

Demo: Polynomial Approximation with Point Values (click to visit)

Error in Interpolation

How did the interpolation error behave in the demo?

To fix notation: f is the function we're interpolating. $\tilde{f}$ is the interpolant that obeys $\tilde{f}(x_i) = f(x_i)$ for $x_i = x_1 < \ldots < x_n$. $h = x_n - x_1$ is the interval length.

For $x \in [x_1, x_n]$:

$$\left| f(x) - \tilde{f}(x) \right| = O(h^n)$$

Can predict errors with this just like for Taylor.

What is the error *at* the interpolation nodes?

Zero–we're matching the function exactly there.

Care to make an unfounded prediction? What will you call it?

Error in Interpolation (II)

It looks like approximating by an $(n-1)$th degree polynomial somewhat generally results in an $O(h^n)$ error. This is called convergence of order n or nth order convergence.

Proof Intuition for Interpolation Error Bound

Let us consider an interpolant $\tilde{f}$ based on $n = 2$ points so

$$\tilde{f}(x_1) = f(x_1) \quad \text{and} \quad \tilde{f}(x_2) = f(x_2).$$

The interpolation error is $O((x_2 - x_1)^2)$ for any $x \in [x_1, x_2]$, why?

Let us define the error function to be

$$E(x) = f(x) - \tilde{f}(x) \quad \text{so we have} \quad E(x_1) = 0 \quad \text{and} \quad E(x_2) = 0.$$

Now note that

$$\int_{x_1}^{x_2} E'(x)dx = E(x_2) - E(x_1) = 0.$$

This implies $E'(x)$ cannot be strictly positive or strictly negative in $[x_1, x_2]$, so

$$\exists_{z \in [x_1, x_2]} E'(z) = 0$$

Proof Intuition for Interpolation Error Bound (II)

Having found $z \in (x_1, x_2)$ such that $E'(z) = 0$, we can proceed analogously to the Taylor remainder theorem proof

$$\begin{aligned} E(x) &= E(x_1) + \int_{x_1}^{x} E'(w_0)dw_0 \\ &= E(x_1) + \int_{x_1}^{x} \left(E'(z) + \int_{z}^{w_0} E''(w_1)dw_1\right)dw_0 \\ &= \int_{x_1}^{x} \left(\int_{z}^{w_0} f''(w_1)dw_1\right)dw_0. \end{aligned}$$

Defining as $x_2 - x_1 = h$ and assuming $x \in [x_1, x_2]$, we again look for a maximizer $\xi = \text{argmax}_{\xi \in [x_1, x_2]} |f''(\xi)|$

$$\begin{aligned} |E(x)| \leq \left| \int_{x_1}^{x} \left(\int_{z}^{w_0} f''(\xi)dw_1\right)dw_0 \right| &= \frac{|f''(\xi)|}{2!} \cdot |z - x| \cdot |x_1 - x| \\ &\leq \frac{|f''(\xi)|}{2!} h^2. \end{aligned}$$

Proof of Interpolation Error Bound

We can use induction on n to show that if $E(x) = f(x) - \tilde{f}(x)$ has n zeros $x_1, \dots, x_n$ and $\tilde{f}$ is a degree n polynomial, then there exist $y_1, \dots, y_n$ such that

$$E(x) = \int_{x_1}^{x} \int_{y_1}^{w_0} \cdots \int_{y_n}^{w_{n-1}} f^{(n+1)}(w_n) dw_n \cdots dw_0 \qquad (1)$$

As before we start by writing

$$E(x) = E(x_1) + \int_{x_1}^{x} E'(w_0) dw_0 \qquad (2)$$

Now note that for each of $n-1$ consecutive pairs x_i, x_{i+1} we have

$$\int_{x_i}^{x_{i+1}} E'(x) dx = E(x_2) - E(x_1) = 0$$

Proof of Interpolation Error Bound (II)

and so there are $z_i \in (x_i, x_{i+1})$ such that $E'(z_i) = 0$. By inductive hypothesis

$$E'(w_0) = \int_{z_1}^{w_0} \int_{y_2}^{w_1} \cdots \int_{y_n}^{w_{n-1}} f^{(n+1)}(w_n) dw_n \cdots dw_1 \tag{3}$$

Substituting (3) into (2), we obtain (1) with $y_1 = z_1$

Making Use of Interpolants

Suppose we can approximate a function as a polynomial:

$$f(x) \approx a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

How is that useful? E.g. what if we want the integral of f?

Easy: Just integrate the interpolant:

$$\begin{aligned}\int_s^t f(x)dx &\approx \int_s^t a_0 + a_1 x + a_2 x^2 + a_3 x^3 dx \\ &= a_0 \int_s^t 1dx + a_1 \int_s^t x \cdot dx + a_2 \int_s^t x^2 dx + a_3 \int_s^t x^3 dx\end{aligned}$$

Even if you had *no idea* how to integrate f using calculus, you can *approximately* integrate f anyway, by taking a bunch of function values (and forming an interpolant).

Demo: Computing Pi with Interpolation (click to visit)

More General Functions

Is this technique limited to the monomials $\{1, x, x^2, x^3, \dots\}$?

No, not at all. Works for any set of functions $\{\varphi_1, \dots, \varphi_n\}$ for which the generalized Vandermonde matrix

$$\begin{pmatrix} \varphi_1(x_1) & \varphi_2(x_1) & \cdots & \varphi_n(x_1) \\ \varphi_1(x_2) & \varphi_2(x_2) & \cdots & \varphi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_1(x_n) & \varphi_2(x_n) & \cdots & \varphi_n(x_n) \end{pmatrix}$$

is invertible.

Interpolation with General Sets of Functions

For a general set of functions $\{\varphi_1, \ldots, \varphi_n\}$, solve the linear system with the generalized Vandermonde matrix for the coefficients $(a_1, \ldots, a_n)$:

$$\underbrace{\begin{pmatrix} \varphi_1(x_1) & \varphi_2(x_1) & \cdots & \varphi_n(x_1) \\ \varphi_1(x_2) & \varphi_2(x_2) & \cdots & \varphi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_1(x_n) & \varphi_2(x_n) & \cdots & \varphi_n(x_n) \end{pmatrix}}_{V} \underbrace{\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}}_{\boldsymbol{a}} = \underbrace{\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix}}_{\boldsymbol{f}}.$$

Given those coefficients, what is the interpolant $\tilde{f}$ satisfying $\tilde{f}(x_i) = f(x_i)$?

$$\tilde{f}(x) = \sum_{i=1}^{n} a_i \varphi_i(x).$$

In-class activity: Interpolation

Outline

Randomness: Why?

What types of problems can we solve with the help of random numbers?

We can compute (potentially) *complicated averages*.

- Where does 'the average' web surfer end up? (PageRank)
- How much is my stock portfolio/option going to be worth?
- How will my robot behave if there is measurement error?

Random Variables

What is a random variable?

A random variable X is a function that depends on 'the (random) state of the world'.

Example: X could be

- 'how much rain tomorrow?', or
- 'will my buttered bread land face-down?'

Idea: Since I don't know the entire state of the world (i.e. all the influencing factors), I can't know the value of X.

$\rightarrow$ Next best thing: Say something about the *average* case.

To do that, I need to know how likely each individual value of X is. I need a probability distribution.

Probability Distributions

What kinds of probability distributions are there?

- discrete distribution:

Event	$X = x_1$	$X = x_2$	$\cdots$	$X = x_n$
Probability	p_1	p_2	$\cdots$	p_n

  Need: $p_i \geq 0$ for the word ‘probability’ to make sense.
- continuous distribution:
  - Values are arbitrary real numbers
  - Each individual value has *zero probability*
  - *But:* Ranges ‘value in range/interval $[a, b]$’ has non-zero probability $\int_a^b p(x)dx$ where p is the distribution function. (Sometimes also called the probability density)

  Need: $p(x) \geq 0$ for ‘probability’ to make sense.

Demo: Plotting Distributions with Histograms (click to visit)

Expected Values/Averages: What?

Define the 'expected value' of a random variable.

For a discrete random variable X:

$$E[f(X)] = \sum_{i=1}^{n} p_i f(x_i)$$

For a continuous random variable:

$$E[f(X)] = \int_{\mathbb{R}} f(x) \cdot p(x) dx$$

Define variance of a random variable.

$$\sigma^2[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2.$$

'Average squared distance from the average'

Expected Value: Example I

What is the expected snowfall (e.g. per day) in Champaign?

$$E[\text{Snow}] = \int_0^\infty s \cdot p(s) ds$$

where

- Snow is a random variable distributed according to $p(s)$,
- s is an integration variable–its name doesn't matter,
- $p(s)$ is the distribution (or density) function.

Note that we're integrating over all possible values of Snow.

Tool: Law of Large Numbers

Terminology:

- *Sample*: A sample $s_1, \ldots, s_N$ of a discrete random variable X (with potential values $x_1, \ldots, x_n$) selects each s_i such that $s_i = x_j$ with probability $p(x_j)$.

In words:

- As the number of samples $N \to \infty$, the average of samples converges to the expected value with probability 1.

What can samples tell us about the distribution?

$$P\left[\lim_{N\to\infty} \frac{1}{N}\left(\sum_{i=1}^{N} s_i\right) = E[X]\right] = 1.$$

Or for an expected value,

$$E[X] \approx \frac{1}{N}\left(\sum_{i=1}^{N} s_i\right)$$

Sampling: Approximating Expected Values

Integrals and sums in expected values are often challenging to evaluate.

How can we approximate an expected value?
Idea: Draw random samples. Make sure they are distributed according to $p(x)$.

1. *Draw N samples* s_i distributed according to $p(x)$.
2. Approximate

$$E[f(X)] \approx \frac{1}{N} \sum_{i=1}^{N} f(s_i).$$

What is a Monte Carlo (MC) method?

Monte Carlo methods are algorithms that compute approximations of desired quantities or phenomena based on randomized sampling.

Expected Values with Hard-to-Sample Distributions

Computing the sample mean requires samples from the distribution $p(x)$ of the random variable X. What if such samples aren't available?

Find a different distribution $\tilde{p}(x)$ that we *can* sample from. Then:

$$
\begin{aligned}
E[X] &= \int_{\mathbb{R}} x \cdot p(x) dx = \int_{\mathbb{R}} x \frac{p(x)}{\tilde{p}(x)} \cdot \tilde{p}(x) dx \\
&= \int_{\mathbb{R}} \tilde{x} \frac{p(\tilde{x})}{\tilde{p}(\tilde{x})} \cdot \tilde{p}(\tilde{x}) dx = E\left[\tilde{X} \cdot \frac{p(\tilde{X})}{\tilde{p}(\tilde{X})}\right].
\end{aligned}
$$

The purpose of this exercise is that we can now apply the sample mean to the last expected value, since $\tilde{p}$ is easy to sample from.

Note 1: The random variable $\tilde{X}$ is distributed according to $\tilde{p}$.
Note 2: Must ensure $\tilde{p}(x) \neq 0$ wherever $p(x) \neq 0$.

(Discrete case goes analogously.)

Switching Distributions for Sampling

Found:

$$E[X] = E\left[\tilde{X} \cdot \frac{p(\tilde{X})}{\tilde{p}(\tilde{X})}\right]$$

Why is this useful for sampling?

Starting point: X is hard to sample from, $\tilde{X}$ is easy to sample from (think uniform). Both have known distribution functions $p(x)$ and $\tilde{p}(x)$. Knowing $p(x)$ does **not** imply that its easy to sample from X.

Then we can approximate $E[X]$ by sampling $\tilde{s}_i$ from $\tilde{X}$:

$$E[X] \approx \frac{1}{N} \sum_{i=1}^{N} \tilde{s}_i \cdot \frac{p(\tilde{s}_i)}{\tilde{p}(\tilde{s}_i)}.$$

In-class activity: Monte-Carlo Methods

Sampling for Integrals

The machinery we have developed can actually be used to approximate arbitrary integrals. How?

We just have to rewrite them as an expected value with respect to some distribution function $\tilde{p}$:

$$\begin{aligned}\int_\Omega g(x)dx &= \int_\Omega \frac{g(x)}{\tilde{p}(x)}\tilde{p}(x)dx \\ &= E\left[\frac{g(\tilde{X})}{\tilde{p}(\tilde{X})}\right] \approx \frac{1}{N}\sum_{i=1}^{N}\frac{g(\tilde{s}_i)}{\tilde{p}(\tilde{s}_i)}.\end{aligned}$$

Note that we have switched to the random variable $\tilde{X}$ under the expected value to indicate that it is distributed according to $\tilde{p}$. We also require that the samples $\tilde{s}_i$ are distributed according to $\tilde{p}$.

We are free to choose $\tilde{p}$ as we like, as long as $\tilde{p}(x) \neq 0$ on Ω. (It is obviously often also convenient if $\tilde{p}$ is easy to sample. That's why uniform distributions are often used.)

Expected Value: Example II

What is the average snowfall in Illinois?

$$E[\text{Snow}] = \int_{\mathbb{R}} \int_{\mathbb{R}} \text{Snow}(x, y) \cdot p(x, y) dx dy?$$

What's $p(x, y)$? What's $\text{Snow}(x, y)$?

- Snow(x,y): expected snowfall at longitude x and latitude y in (say) a day.
- $p(x, y)$: Probability that a point with longitude x and latitude y is in Illinois.

Expected Value: Example II (II)

Can check whether (x, y) is in Illinois, e.g. as

$$q(x,y) = \begin{cases} 1 & \text{if } (x,y) \text{ is in Illinois,} \\ 0 & \text{if it's not.} \end{cases}$$

But: Need p to be a probability density, i.e.

$$\int_{\mathbb{R}} \int_{\mathbb{R}} p(x,y) dx dy = 1.$$

Can ensure that by finding a scaling factor C so that $p(x) = C \cdot q(x)$.

- Find
$$C = \frac{1}{\int_{\mathbb{R}} \int_{\mathbb{R}} p(x,y) dx dy}$$
by using sampling to approximate the integral. (How?)

Expected Value: Example II (III)

- And once more to compute

$$E[\text{Snow}] = E\left[\text{Snow}\,\frac{p(x,y)}{p_{\mathsf{uniform}(x,y)}}\right]$$

where we have chosen $p_{\mathsf{uniform}(x,y)}$ to be an (easy-to-sample) uniform distribution on a rectangle containing the state area.

Example: Computing a 2D Integral using Monte Carlo

Lets consider integrating $f(x,y)$ on domain $\Omega \subset [0,L]^2$

$$G = \int\int_{\Omega} f(x,y)dxdy = \int_0^L \int_0^L f(x,y)\mathbf{1}_{\Omega}(x,y)dxdy,$$

where $\mathbf{1}_{\Omega}(x,y) = 1$ if $(x,y) \in \Omega$ and $\mathbf{1}_{\Omega}(x,y) = 0$ if $(x,y) \notin \Omega$.

If $|\Omega|$ is the area of domain Ω, then $p(x,y) = \frac{1}{|\Omega|}\mathbf{1}_{\Omega}(x,y)$ can be interpreted as a probability distribution (the normalization ensures its integral over the whole domain is equal to 1).
We can express G as an expected value of a random variable Z (which takes on 2D coordinate values),

$$G = |\Omega|E[f(Z)] = |\Omega| \int_0^L \int_0^L f(x,y)p(x,y)dxdy.$$

Example: Computing a 2D Integral using Monte Carlo (II)

We can approximate G based on samples of uniform random variable X: $(x_1, y_1), \ldots, (x_N, y_N) \in [0,1]^2$ with distribution $\tilde{p}(x,y) = \frac{1}{L^2}$, using only the function $\mathbf{1}_\Omega(x,y)$,

$$
\begin{aligned}
G &= |\Omega| E\left[f(X) \frac{p(X)}{\frac{1}{L^2}} \right] = |\Omega| L^2 E[f(X)p(X)] \\
&\approx \frac{|\Omega| L^2}{N} \sum_{i=1}^{N} f(x_i, y_i) p(x_i, y_i) \\
&= \frac{L^2}{N} \sum_{i=1}^{N} f(x_i, y_i) \mathbf{1}_\Omega(x_i, y_i).
\end{aligned}
$$

Demo: Computing Pi using Sampling (click to visit)
Demo: Errors in Sampling (click to visit)

Sampling: Error

The Central Limit Theorem states that with

$$S_N := X_1 + X_2 + \cdots + X_n$$

for the (X_i) independent and identically distributed according to random variable X with variance σ^2, we have that

$$\frac{S_N - NE[X]}{\sqrt{\sigma^2 N}} \to \mathcal{N}(0,1),$$

i.e. that term approaches the normal distribution. As we increase N, σ^2 stays fixed, so the asymptotic behavior of the error is

$$\left|\frac{1}{N}S_N - E[X]\right| = O\left(\frac{1}{\sqrt{N}}\right).$$

Proof Intuition for Central Limit Theorem

The Central Limit Theorem uses the fact that given N identically distribution samples of random variable X with variance $\sigma^2[X]$, the average of the samples will have variance $\sigma^2[X]/N$. Since $\sigma^2[X] = E[(E[X] - X)^2]$ is the expected square of the deviation, it tells us how far away the average of the samples is expected to be from the real mean. Why is this the case?

Represent each sample $s_1, \ldots, s_N$ by random variable $X_1, \ldots X_N$ (each one is identical to X, but all are independent).
Assume that

$$E[X] = E[X_i] = 0.$$

By considering the variance of $S_N = \sum_{i=1}^{N} X_i/N$, we can gauge the expectation of how far their average is expected to deviate from $E[X] = 0$. That variance is given by

$$\sigma^2[S_N] = E[S_N^2] = E\left[\left(\sum_{i=1}^{N} X_i/N\right)^2\right] = \frac{1}{N^2} E\left[\left(\sum_{i=1}^{N} X_i\right)^2\right].$$

Proof Intuition for Central Limit Theorem (II)

We can separate that into parts

$$\sigma^2[S_N] = \frac{1}{N^2}\left(\sum_{i=1}^{N} E[X_i^2] + \sum_{i=1}^{N}\sum_{j=1, j\neq i}^{N} E[X_i X_j]\right).$$

Since X_i and X_j are independent, we have

$$\begin{aligned}
E[X_i X_j] &= \int\int x_i x_j p(x_i) p(x_j) dx_i dx_j \\
&= \left(\int x_i p(x_i) dx_i\right)\left(\int x_j p(x_j) dx_j\right) \\
&= E[X_i]E[X_j] = 0 \cdot 0 = 0,
\end{aligned}$$

therefore, the previous expression simplifes to

$$\sigma^2[S_N] = \frac{1}{N^2}\sum_{i=1}^{N} E[X_i^2] = \frac{1}{N}E[X^2] = \frac{\sigma^2[X]}{N}.$$

Monte Carlo Methods: The Good and the Bad

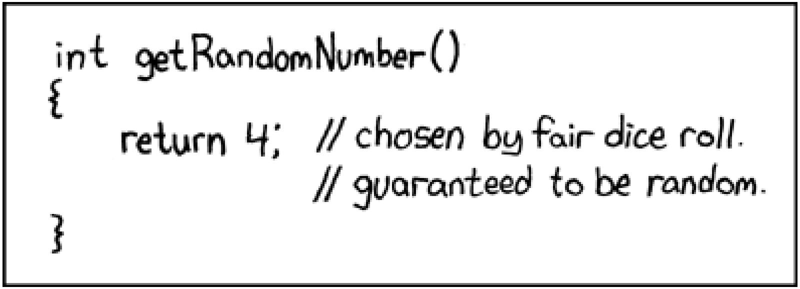
What are some *advantages of MC methods?*

- Computes integrals when nothing else will
- Convergence does not depend on dimensionality
- Still applies when deterministic modeling fails

What are some *disadvantages* of MC methods?

- Convergence is very slow $\left(O\left(1/\sqrt{n}\right)\right)$
- Outcome is non-deterministic

Computers and Random Numbers



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

[from xkcd]

How can a computer make random numbers?

It kind of can't. Computers are predictable. Random numbers aren't supposed to be.

Option 1: Stick a *source of actual randomness* into the computer.

Computers and Random Numbers (II)

- Don't have to look very far: Arrival times of network packets, mouse movements, ... are all sort of random.
- `xxd /dev/random`
- `xxd /dev/urandom`
  Difference?
- $\sim$40 bucks will buy you one: e.g. Altus Metrum ChaosKey

For generating large samples, these methods are clearly too expensive.

Random Numbers: What do we want?

What properties can ‘random numbers’ have?

- Have a specific distribution
  (e.g. ‘uniform’–each value in given interval is equally likely)
- Real-valued/integer-valued
- Repeatable (i.e. you may *ask* to exactly reproduce a sequence)
- Unpredictable
  - V1: ‘I have no idea what it’s going to do next.’
  - V2: No amount of engineering effort can get me the next number.
- Uncorrelated with later parts of the sequence
  (Weaker: Doesn’t repeat after a short time)
- Usable on parallel computers

What's a Pseudorandom Number?

Actual randomness seems like a lot of work. How about 'pseudo-random numbers?'

Idea: Maintain some 'state'. Every time someone asks for a number:

$$\text{random\_number}, \text{new\_state} = f(\text{state})$$

Satisfy:

- Distribution
- 'I have no idea what it's going to do next.'
- Repeatable (just save the state)
- Typically *not* easy to use on parallel computers

Demo: Playing around with Random Number Generators (click to visit)

Some Pseudorandom Number Generators

Lots of variants of this idea:

- LC: 'Linear congruential' generators
- MT: 'Mersenne twister'
- almost all randonumber generators you're likely to find are based on these–Python's `random` module, `numpy.random`, C's `rand()`, C's `rand48()`.

Counter-Based Random Number Generation (CBRNG)

What's a CBRNG?

Idea: Cryptography has *way* stronger requirements than RNGs.
And the output *must* 'look random'.

(Advanced Encryption Standard) AES algorithm:
128 encrypted bits $= \mathrm{AES}\,(\text{128-bit plaintext}, \text{128 bit key})$

We can treat the encrypted bits as random:
128 random bits $= \mathrm{AES}\,(\text{128-bit counter}, \text{arbitrary 128 bit key})$

- Just use $1, 2, 3, 4, 5, \ldots.$ as the counter.
- *No* quality requirements on counter or key to obtain high-quality random numbers
- *Very* easy to use on parallel computers
- Often accelerated by hardware, faster than the competition

Demo: Counter-Based Random Number Generation (click to visit)

Outline

Error in Numerical Methods

Every result we compute in Numerical Methods is inaccurate. What is our model of that error?

$$\text{Approximate Result} = \text{True Value} + \text{Error}.$$

$$\tilde{x} = x_0 + \Delta x.$$

Suppose the true answer to a given problem is x_0, and the computed answer is $\tilde{x}$. What is the absolute error?

$|x_0 - \tilde{x}|$.

Relative Error

What is the relative error?

$$\frac{|x_0 - \tilde{x}|}{|x_0|}$$

Why introduce relative error?

Because absolute error can be misleading, depending on the magnitude of x_0. Take an absolute error of 0.1 as an example.

- If $x_0 = 10^5$, then $\tilde{x} = 10^5 + 0.1$ is a fairly accurate result.
- If $x_0 = 10^{-5}$, then $\tilde{x} = 10^{-5} + 0.1$ is a completely inaccurate result.

Relative Error (II)

Relative error is independent of magnitude.

What is meant by 'the result has 5 accurate digits'?

Say we compute an answer that gets printed as

$$3.1415777777.$$

The closer we get to the correct answer, the more of the leading digits will be right:

$$3.1415777777.$$

This result has 5 accurate digits. Consider another result:

$$123,477.7777$$

This has four accurate digits. To determine the number of accurate digits, start counting from the front (most-significant) non-zero digit.

Relative Error (III)

Observation: ‘Accurate digits’ is a measure of relative error.

‘$\tilde{x}$ has n accurate digits’ is roughly equivalent to having a relative error of 10^{-n}. Generally, we can show

$$\frac{|\tilde{x} - x_0|}{|x_0|} < 10^{-n+1}.$$

Measuring Error

Why is $|\tilde{x}| - |x_0|$ a bad measure of the error?

Because it would claim that $\tilde{x} = -5$ and $x_0 = 5$ have error 0.

If $\widetilde{\boldsymbol{x}}$ and $\boldsymbol{x}_0$ are vectors, how do we measure the error?

Using something called a vector norm. Will introduce those soon.
Basic idea: Use norm in place of absolute value. Symbol: $\|\boldsymbol{x}\|$. E.g. for relative error:

$$\frac{\|\widetilde{\boldsymbol{x}} - \boldsymbol{x}_0\|}{\|\boldsymbol{x}_0\|}.$$

Sources of Error

What are the main sources of error in numerical computation?

- Truncation error:
  (E.g. Taylor series truncation, finite-size models, finite polynomial degrees)
- Rounding error
  (Numbers only represented with up to ˜15 accurate digits.)

Digits and Rounding

Establish a relationship between '*accurate digits*' and rounding error.

Suppose a result gets rounded to 4 digits:

$$3.1415926 \quad \to \quad 3.142.$$

Since computers always work with finitely many digits, they must do something similar. By doing so, we've introduced an error–'rounding error'.

$$|3.1415926 - 3.142| = 0.0005074$$

Rounding to 4 digits leaves 4 accurate digits–a relative error of about 10^{-4}.

Computers round *every* result–so they *constantly* introduce relative error.
(Will look at how in a second.)

Condition Numbers

Methods f take input x and produce output $y = f(x)$.
Input has (relative) error $|\Delta x| \,/\, |x|$.
Output has (relative) error $|\Delta y| \,/\, |y|$.
Q: Did the method make the relative error bigger? If so, by how much?

The condition number provides the answer to that question.
It is simply the smallest number κ across all inputs x so that

$$\text{Rel error in output} \leqslant \kappa \cdot \text{Rel error in input},$$

or, in symbols,

$$\kappa = \max_x \frac{\text{Rel error in output } f(x)}{\text{Rel error in input } x} = \max_x \frac{\frac{|f(x) - f(x+\Delta x)|}{|f(x)|}}{\frac{|\Delta x|}{|x|}}.$$

nth-Order Accuracy

Often, *truncation error* is controlled by a parameter h.

Examples:

- distance from expansion center in Taylor expansions
- length of the interval in interpolation

A numerical method is called 'nth-order accurate' if its truncation error $E(h)$ obeys

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

Outline

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, e.g. 23.625?

Idea: Keep going down past zero exponent:

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

So: Could store

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

This is called fixed-point arithmetic.

Fixed-Point Numbers

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

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

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

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

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

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

Floating Point numbers

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

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

What pieces do you need to store an FP number?

Significand: $(1.101)_2$
Exponent: 3

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

Only store '101'. Final storage format:

Floating Point numbers (II)

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 (click to visit)

Subnormal Numbers

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

First attempt:

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

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

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:

$$(0.0001)_2 \cdot 2^{-7}$$

Numbers with the special exponent are called subnormal (or denormal) FP numbers. Technically, zero is also a subnormal.

Subnormal Numbers (II)

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

Subnormal Numbers (II)

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.
- 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.

Summary: Translating Floating Point Numbers

To summarize: To translate a *stored* (double precision) floating point value consisting of the stored $\mathrm{fraction}$ (a 52-bit integer) and the stored exponent value e_{stored} the into its (mathematical) value, follow the following method:

$$\mathrm{value} = \begin{cases} (1 + \mathrm{fraction} \cdot 2^{-52}) \cdot 2^{-1023 + e_{\mathrm{stored}}} & e_{\mathrm{stored}} \neq 0, \\ (\mathrm{fraction} \cdot 2^{-52}) \cdot 2^{-1022} & e_{\mathrm{stored}} = 0. \end{cases}$$

Demo: Density of Floating Point Numbers (click to visit)
Demo: Floating Point vs Program Logic (click to visit)

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 stored 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?

Floating Point and Rounding Error (II)

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

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

Ignoring possible subtleties about rounding, in the above system, $\varepsilon_{\text{mach}} = (0.0001)_2$. Another related quantity is ULP, or unit in the last place.

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

  $$x \quad \text{and} \quad x \cdot (1 + \varepsilon_{\text{mach}}),$$

  that's really the minimum error incurred.

Floating Point and Rounding Error (III)

- 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)

$$2^{-52} \approx 10^{-16}$$

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

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

Implementing Arithmetic

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

Rough algorithm:

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

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

Demo: Floating point and the Harmonic Series (click to visit)

Problems with FP Addition

What happens if you subtract two numbers of very similar magnitude?
As an example, consider $a = (1.1011)_2 \cdot 2^1$ and $b = (1.1010)_2 \cdot 2^1$.

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

or, once we normalize,

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

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

This phenomenon is called Catastrophic Cancellation.

Demo: Catastrophic Cancellation (click to visit)
In-class activity: Floating Point 2