Getting to the bottom, fast

Prof. Ramin Zabih

http://cs100r.cs.cornell.edu

Cornell University
Computer Science

Administrivia

- Assignment 4 is due on Friday
- Quiz 6 will be Thursday Oct 25
- Prelim schedule
  - In class exams, 30 minutes
  - P2: Thursday Nov 1
  - P3: Thursday Nov 29 (last lecture…)
- Please start thinking about your final project!
  - Due sometime during finals week (date TBA)

2nd-to-last Convex Hull Fact

- Convex hull is closely linked to sorting
- You can sort the numbers x_i by computing the convex hull of the points (x_i, x_i^2)
  - This implies that a fast method for convex hull gives you a fast method for sorting
  - Fact: you can't do comparison-based sort in linear time
    - Proof is sometimes taught in CS280
  - Consequence: you can't do convex hull in linear time either!

Why is an error function hard?

- An error function where we can get stuck if we roll downhill is a hard one
  - Where we get stuck depends on where we start (i.e., initial guess/conditions)
  - An error function is hard if the area “above it” has a certain shape
    - Nooks and crannies
    - In other words, non-CONVEX!
  - Non-convex error functions are hard to minimize

More general problem

- Suppose we have a convex function $f(x)$
  - 1D, for simplicity
- How do we find the minimum x^*?

$$x^* = \arg\min_x f(x)$$

- This is an extremely important task
  - For example, for various physics problems
- We showed you a dumb way to do this
  - Hillclimbing: start somewhere, change your guess a little, see if it improves
- Can we do something smarter?

Interval strategy

- Many algorithms "bracket" the minimum
  - Maintain an interval $[x_{\text{low}}, x_{\text{high}}]$ such that the minimum x^* lies somewhere inside
    - We will call such an interval *valid*
  - Shrink the interval steadily
- To start off, we need an initial interval
  - For simplicity, we will pretend the minimum happens at some positive value (i.e., $x^* > 0$)
  - So our initial interval is [0, ???]
- Need an initial $x_{\text{high}} > x^*$, i.e. to the right of the minimum. How can we find one?

Finding derivatives

- Suppose we can tell something about the slope of f
  - In other words, the derivative f'
- Suppose we just know the **sign** of f'
  - I.e., we know f is increasing/decreasing
  - We know: $f'(x_{\text{low}}) \leq 0 \quad f'(x_{\text{high}}) \geq 0$
  - Derivative sign tells us if we are to the left or right of the answer
- Approach (standard CS "hallucination")
  - Pretend we know the sign of f'
  - Figure out how to compute the sign of f'

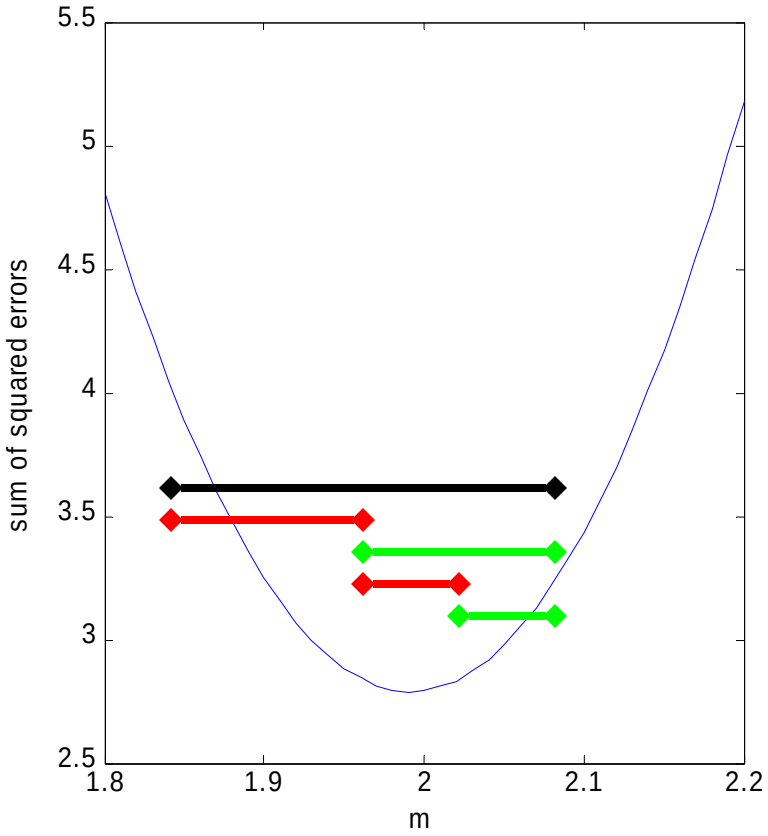
Computing the initial interval

- We can find an initial choice of x_{high} by moving right until the derivative has the correct sign
  - I.e., f is increasing: $f' \geq 0$
  - As we will see, the best way to do this is by repeated doubling
    - Try 1, then 2, then 4, then 8, etc.
    - This will get us a valid x_{high} fast, though it might be very large
      - For instance, if $x^* = 1025$, $x_{\text{high}} = 2048$
    - This will turn out not to matter!

Reducing the interval

- We have a valid interval $[x_{\text{low}}, x_{\text{high}}]$
  - How do we create a smaller valid interval?
  - Consider the midpoint: $x_{\text{mid}} = \frac{x_{\text{low}} + x_{\text{high}}}{2}$
  - The two intervals $[x_{\text{low}}, x_{\text{mid}}]$ and $[x_{\text{mid}}, x_{\text{high}}]$ are half the size
  - One of them is valid!
    - Can you prove this?

Binary search in action



Binary search speed

- The basic operation is very fast
  - Each iteration halves the size of the interval!
    - This is why it's OK to start with a big interval
- We need to evaluate the function at a relatively small number of places
  - Reducing the number of function evaluations is the name of the game
  - We also need to evaluate the derivative
    - Which is a bit more work, though not hard

Evaluating the derivative

- To compute the derivative f' we recall:

$$f'(x) \equiv \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- We can approximate this by using a suitably small value of h
  - Details in CS322/CS421

More binary search

- This technique is quite general, and useful for lots of problems besides minimization
  - We'll need it for images when we look into recognizing colored objects
- Here is a nice (non-image) example
  - Suppose you have many versions of a file, that lots of people are editing
    - Think of a big business document (or program)
  - You'd like to know who introduced some text
    - Often, you're looking for a bug…

Binary search example

- We want the earliest document where a certain change was made
- If we are going to make many queries like this, it is worth spending some effort at the beginning to make the later ones fast
- This is a incredibly common technique
  - Pay up front, reap the rewards later
- We will simply sort the documents by date
  - Sorting, in CS, is often the answer

Using binary search

- With the documents in order, we can find the earliest one with the text we want via binary search
  - The last document has the new text, the first document does not: interval = [first, last]
  - What about the middle document?
  - If it has the new text, we can look in the interval [first, middle]
  - If it doesn't have the new text, we can look in the interval [middle+1, last]