CSE 417
Binary Search (pt 3)
Reminders

> HW1 is due today
  – added a clarification on 1d

> HW2 will be posted shortly
  – coding HW: fitting an ML model using ternary search
  – start early! (might require some trial & error)
Outline for Today

- Binary search over the reals
- Ternary search over the reals
- Applications to ML
Binary search over the reals

> Above we considered \( f : \mathbb{Z} \rightarrow \mathbb{R} \). What about \( f : \mathbb{R} \rightarrow \mathbb{R} \)?

> Standard problem in numerical analysis...

> But instead of inverting \( f \), they want to find the zeros of \( f \)

  > these are equivalent problems:
  > 1. to find \( t \) s.t. \( f(t) = x \), just define \( g(t) = f(t) - x \)
  > 2. \( g(t) = 0 \) means \( f(t) = x \)
Binary search over the reals

> New problem: when do we stop?
  – before, if f(t) < x, we can try f(t+1), f(t+2), ...
  – now, we might need to try f(t+0.00000000001)

**Input:** A monotonically increasing function f : R \(\rightarrow\) R, a range \([a, b]\), a number \(x\) in R, and an error tolerance \(\varepsilon\) in R

**Output:** number \(u\) in \([a, b]\) such that:
  – \(f(t) \leq x\) for all \(t\) in \([a, u]\)
  – \(x < f(t)\) for all \(t\) in \([u + \varepsilon, b]\)
Binary search over the reals

```c
float u = a, v = b;

// Invariant: (f(s) <= x for s <= u) and (x < f(s) for s >= v)
while (v - u > eps) {
    float t = (u + v) / 2;
    if (f(t) <= x)
        u = m;
    else
        v = m;
}
return u;
```

How to choose $\varepsilon$  \textit{(out of scope)}

$\varepsilon$ is small

If $f$ is differentiable, then $f(t + s) \approx f(t) + f'(t) s$

so $f(t+s) - f(t) \approx f'(t) s$

If $f'(t) \leq U$, for some $U$, and we want $|f(t+\varepsilon) - f(t)| < \delta$ then we can take $\varepsilon = \delta / U$

(this works if $f$ is not differentiable provided that it is Lipschitz continuous)
Find minimum of a convex function

> Rather than find the zeros of function, we might want to *minimize* it

> This is always reasonable if \( f \) is convex

   - definition: \( f((x_1 + x_2) / 2) \leq (f(x_1) + f(x_2)) / 2 \) for all \( x_1 \) & \( x_2 \)

Picture from en.wikipedia.org/wiki/Convex_function
Fact: differentiable function $f$ is convex iff its derivative $f'$ is *monotonically increasing*.

In particular, its minimum occurs where $f'(t) = 0$.

So we can minimize $f$ by finding a zero of $f'$.

- hence, we can minimize a convex function using binary search
- assuming that we can compute the derivative $f'$
  - (actually, weaker notions of derivatives would work here)
  - (e.g., a convex function is left-differentiable)
Example: minimize smoothed rank

> Minimize $f(t) = \log \det(X + t Y)$, where $X$ and $Y$ are matrices
  – this is $\mathbb{R} \rightarrow \mathbb{R}$ even though it operates on matrices (tables)
    > would not expect an analytical solution
  – it is also convex! (not easy to prove, but true)
  – would still need to compute $f'$ though (yuck!)

> $\log \det(A + \varepsilon I)$ is called the smoothed rank of the matrix
  – minimizing actual matrix rank is hard
  – this gives an efficiently computable approximation
  – applications to, e.g., low-dimension approximation of high-dimensional noisy data
Find minimum of a convex function

> That said, binary search is not the fastest algorithm
> The fastest algorithm is Newton’s method
  – in fact, Newton’s method is exponentially faster than binary search

> Cannot overstate the importance of an exponential speedup
  – that said, if binary search takes $O(\log n)$ iterations, then Newton’s method takes $O(\log \log n)$ iterations
  – in practice, the difference is not always important
  – importance is when $n$ is exponentially large
    > then $\log n$ is polynomial, but $\log \log n$ is actually small
Find minimum of a convex function

> Newton’s method requires another derivative
  – so we would need the function to be twice differentiable
  – if our function is multi-variate, then the second derivative is a matrix
    > see the HW for an example where this would arise
  – sometimes that is too much to ask

> It is actually possible to find the minimum with *no derivatives*...
Outline for Today

- Binary search over the reals
- Ternary search over the reals
- Applications to ML
Ternary Search

**Input:** a *unimodal* function \( f : \mathbb{R} \rightarrow \mathbb{R} \), a range \([a, b]\), and an error tolerance \( \varepsilon \) in \( \mathbb{R} \)

> **unimodal** means it decreases *monotonically* and then increases
> slightly larger class than convex
>  – functions can have cusps (see picture)

Picture from abcalculus.wikispaces.com/When+does+a+derivative+NOT+exist%3F
Ternary Search

Input: a unimodal function $f : \mathbb{R} \rightarrow \mathbb{R}$, a range $[a, b]$, and an error tolerance $\varepsilon$ in $\mathbb{R}$

Output: number $t$ in $[a, b]$ such that the minimum lies in $[t, t + \varepsilon]$
**Ternary Search**

Let \( x_1 = a + \frac{b - a}{3} \)
Let \( x_2 = a + 2 \frac{b - a}{3} \)

**Idea:** evaluate \( f(x_1) \) and \( f(x_2) \)

> if \( f(x_1) < f(x_2) \), then the minimum cannot be in \( [x_2, b] \)
  - function decreases then increases
  - if the minimum were in \( [x_2, b] \), it would have decreased from \( x_1 \) to \( x_2 \)
Let \( x_1 = a + (b - a) / 3 \)
Let \( x_2 = a + 2 (b - a) / 3 \)

**Idea:** evaluate \( f(x_1) \) and \( f(x_2) \)

- if \( f(x_1) < f(x_2) \), then the minimum cannot be in \([x_2, b]\)
  - function decreases then increases
  - if the minimum were in \([x_2, b]\), would have decreased from \(x_1\) to \(x_2\)

- if \( f(x_1) > f(x_2) \), then the minimum cannot be in \([a, x_1]\)
  - if the minimum were in \([a, x_1]\), would have increased from \(x_1\) to \(x_2\)
Ternary Search

> Each iteration removes 1/3 of the search space
  – if \( f(x_1) < f(x_2) \), we eliminate \([x_2, b]\)
  – if \( f(x_1) > f(x_2) \), we eliminate \([a, x_1]\)

> Reduced to \((b - a) \cdot (2/3)^k\) after \(k\) iterations
> Done after \(k = \log_{3/2} \left(\frac{b - a}{\varepsilon}\right)\) iterations

> Takes \(O(\log \left(\frac{b - a}{\varepsilon}\right))\) time if evaluating \(f\) takes \(O(1)\)
  – in general, factor of \(O(\log \left(\frac{b - a}{\varepsilon}\right))\) increase in time
    versus time to evaluate \(f\)
“Binary Search” Most Broadly

> Binary and ternary search reduce search space by constant factor (1/2 or 2/3) on each iteration

> Any algorithm with that properly will take $O(\log(b - a))$ iterations to reduce to interval of length 1
  - even a reduction to 99/100 of the size still works
  - since $(b - a) (99/100)^k = 1$ is true when
    $$k = \log_{100/99} (b - a) = \log(b - a) / \log(100/99) = O(\log(b - a))$$
This is a general algorithm design approach: try to reduce size of search space by a constant fraction
  – some people consider this a type of “divide & conquer” algorithm
  – I think of it as a separate group, but whatever

Other examples of algorithms within this paradigm

Worth trying out when you’re looking for algorithms
“Binary Search” Most Broadly

> For me, the presence of monotonicity & unimodality are usually the best clues to look for
  – these only work in a subset of the cases
  – but they are the easiest to spot and arise most often

> In particular, key idea: consider computing the inverse instead
  – e.g., rather than trying to compute profit $\rightarrow$ hemming cost,
    compute hemming cost $\rightarrow$ profit
  – if it is monotonic, then you can invert with binary search
Outline for Today

- Binary search over the reals
- Ternary search over the reals
- Applications to ML
Model Fitting

- **Problem**: want to find the best description of the data

- Choose a model that you think should look like the data
  - describes the general shape

- Model usually has parameters
  - parameters give the exact function

- **Sub-problem**: find the choice of parameters that make the model best fit the data
Example: best fit line

- You think the data should be \( \sim \)linear
  - not exactly linear due to measurement error etc.

- Model: \( y \sim A \times x + B \)
  - parameter A: slope
  - parameter B: y-intercept

Picture from en.wikipedia.org/wiki/Linear_regression
Q: How do you find the best model parameters
A: Choose the setting that minimizes / maximizes some function
  – for minimization, often called a “loss function”

Take an ML course for details on ways choose these
  – maximum likelihood
  – maximum a posteriori
  – regularization
  – etc.
Example: best fit line

> Model: \( y \sim A \times x + B \)

> Classical linear regression:
  Find A and B to minimize sum of squared errors:

\[
\sum_i (y_i - (A \times x_i + B))^2
\]
Model Fitting

- There is a formula in this case, but not in general...
- Q: How do we fit the model?

- Note that the loss function is usually convex
  - counter-example: deep learning
- A: Newton’s method
Minimizing multivariate convex function

- In contrast to earlier, this is never a function of one argument
  - even for linear regression, we have two parameters: A and B

- Newton’s method generalizes well to multivariate functions
- However, it is not always used due to complexity
  - derivative of a multivariate function is a vector (the gradient)
  - second derivative is a matrix (the Hessian)
  - finding this matrix is work, both theoretical and computational
    - seen ML papers whose full contribution is giving formulas for the entries of the Hessian matrix of a model
Minimizing *multivariate* convex function

- In contrast to our earlier, this is never a function of one argument
  - even for linear regression, we have two parameters: A and B

- Common alternative is gradient descent...
**Gradient Descent (~out of scope)**

> Gradient = direction of steepest ascent
> -Gradient = direction of steepest decline

> Minimize loss function by stepping along -gradient
  – re-compute gradient after each step
  – decrease step size on each iteration
  – see numerical analysis or ML class for more...

Picture from en.wikipedia.org/Gradient_descent
Gradient Descent (~out of scope)

> Gradient descent is widely used

> Newton’s method may be faster
  – fewer iterations but more work per iteration

> also function must be twice differentiable...
  – not all loss functions even differentiable once!
    > see HW2 for an example
Coordinate Descent

> Only try stepping along coordinate axes

> Only trying to change one parameter at a time
  > loss function with all parameters but one fixed
  > convex function of one parameter
    > still convex since the loss function is convex
  > can minimize it using ternary search
Coordinate Descent for Model Fitting

start with an initial model
  – maybe all zeros or random

repeat until it “stops changing much”:
  for each parameter of the model:
    choose new value for the parameter that minimizes the loss function with all other parameters fixed
Coordinate Descent for Linear Regression

Either changing A or B
Minimizing function of the form:

\[
 f_B(A) := \sum_i (y_i - (A x_i + B))^2
\]

\[
 g_A(B) := \sum_i (y_i - (A x_i + B))^2
\]

B is fixed in first definition
A is fixed in second one
> **Warning**: coordinate descent does not work for all functions

- function can appear minimum along both axes BUT not be a true minimum
- e.g., point (-2, -2) in picture

Picture from en.wikipedia.org/wiki/Coordinate_descent
Coordinate Descent

> Works for functions of the form $f(x_1, .., x_n) + g(x_1, ..., x_n)$ where
>   - $f$ is convex and differentiable
>   - $g$ is separable: $g(x_1, ..., x_n) := \sum_i g_i(x_i)$ with each $g_i$ convex

> In particular, it works for this function:

$$\sum_i \left( y_i - (A x_i + B) \right)^2 + |A| + |B|$$

> We will use this in HW2
>   - (separable part is an “L1 regularization term”... they have a tendency to make many parameters zero)
Fit a model to describe NFL teams

Each NFL game as a sequence of “drives”
  - drive is a series of consecutive plays with one team’s offense against the other team’s defense
  - drive starts somewhere on the field and ends somewhere else
    > the team on offense wants it to end in the end zone (6 points)
    > team on defense wants it to end on the other side (-2 points)
  - consider expected points for drives start & end points
    > average points scored by teams starting (ending) at that position
> More drives than games. More data = more predictive

> Fit a model to explain change in expected points on each drive

> Two parameters per team $t$: $A_t$ for offense, $B_t$ for defense
  
  – also a constant term $C$

$$
\sum_{\text{drive } i} (y_i - (A_{\text{offense } i} - B_{\text{defense } i} + C))^2 + \sum_{\text{team } t} |A_t| + |B_t|$$
Web page to test out the model (if you want):
-  simulate a game OR
-  compute win probability (\(\Rightarrow\) point spread)