5C – Optimization: 1D Optimization

Derek Nowrouzezahrai
derek@cim.mcgill.ca

More Non-linear Problems

There are many categories of non-linear problems

- *root finding* problems were the first such category we studied:

$$\textbf{solve for } \mathbf{x} \textbf{ in } f(\mathbf{x}) = \mathbf{y} \textbf{ or equivalently in } f(\mathbf{x}) = 0$$

- recall – linear system solves are a special case with $\mathbf{Ax} - \mathbf{b} = 0$

Two more very important non-linear problems are

- unconstrained optimization, and
- constrained optimization

Unconstrained Optimization

We express the unconstrained optimization problem as:

solve for $\mathbf{x}$ that minimizes $f(\mathbf{x})$

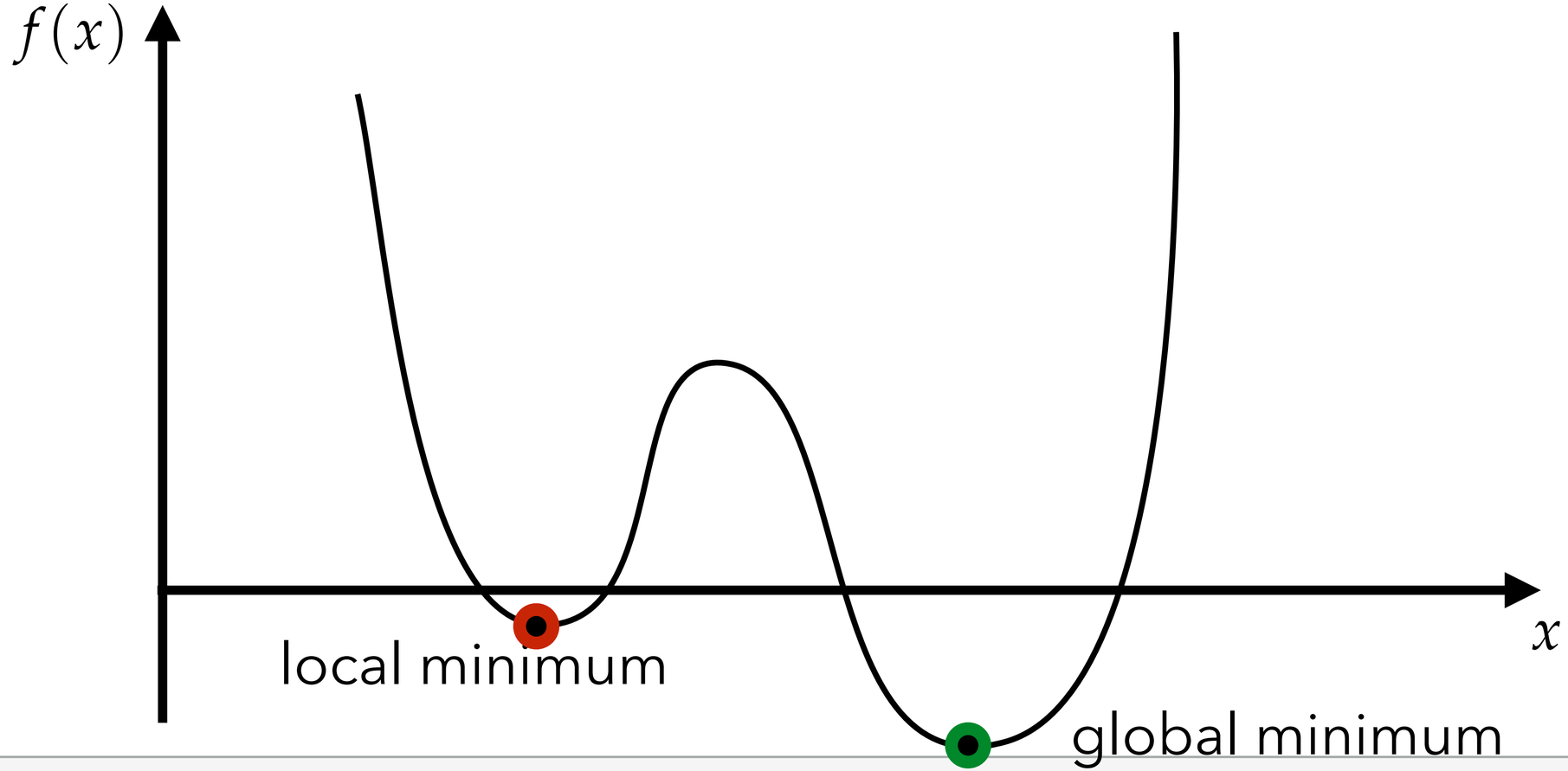
(or equivalently, solve for $\mathbf{x}$ that maximizes $-f(\mathbf{x})$)

- while similar to root finding, optimization is generally more challenging as we don't know *what* the minimum value is
- this form of optimization does not impose any additional constraints on $\mathbf{x}$, which is why we call is *unconstrained*

Unconstrained Optimization

solve for x that minimizes $f(\mathbf{x})$

- we don't know *what* the minimum value is!



Constrained Optimization

Constrained optimization problems allow for further restrictions on the solution $\mathbf{x}$, and are expressed as:

solve for $\mathbf{x}$ that minimizes $f(\mathbf{x})$

(or equivalently, solve for $\mathbf{x}$ that maximizes $-f(\mathbf{x})$)

subject to the (optional) *equality* constraints $\mathbf{g}(\mathbf{x}) = 0$

and (also optional) *inequality* constraints $\mathbf{h}(\mathbf{x}) \leq 0$

- an important special-case of unconstrained optimization is when f, $\mathbf{g}$, and $\mathbf{h}$ are all linear maps – ***linear programming***

We will focus exclusively on <u>unconstrained optimization</u>

Starting simple, again…

Let's again start by considering functions that map from 1D to 1D

$$f : \mathbb{R} \rightarrow \mathbb{R}$$

and only seek solutions to the *unconstrained optimization problem* (simply referred to as "*optimization*", here on out)

$$x_* = \underset{x}{\operatorname{argmin}}\, f(x)$$

Root Finding vs. Unconstrained Optimization

While distinct, root finding and unconstrained optimization problems can be related to each other

- in fact, every unconstrained optimization problem can be expressed as a root finding problem, *and*
- every root finding problem can be expressed as a dual unconstrained optimization problem

Root Finding vs. Unconstrained Optimization

While distinct, root finding and unconstrained optimization problems can be related to each other

- in fact, every unconstrained optimization problem can be expressed as a root finding problem
  - we've already seen this in the context of linear systems:

    every least-squares minimization problem $\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2$

    can be expressed as a root finding problem $\mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{A}^T\mathbf{b} = 0$

    (and vice-versa)

Root Finding vs. Unconstrained Optimization

While distinct, root finding and unconstrained optimization problems can be related to each other

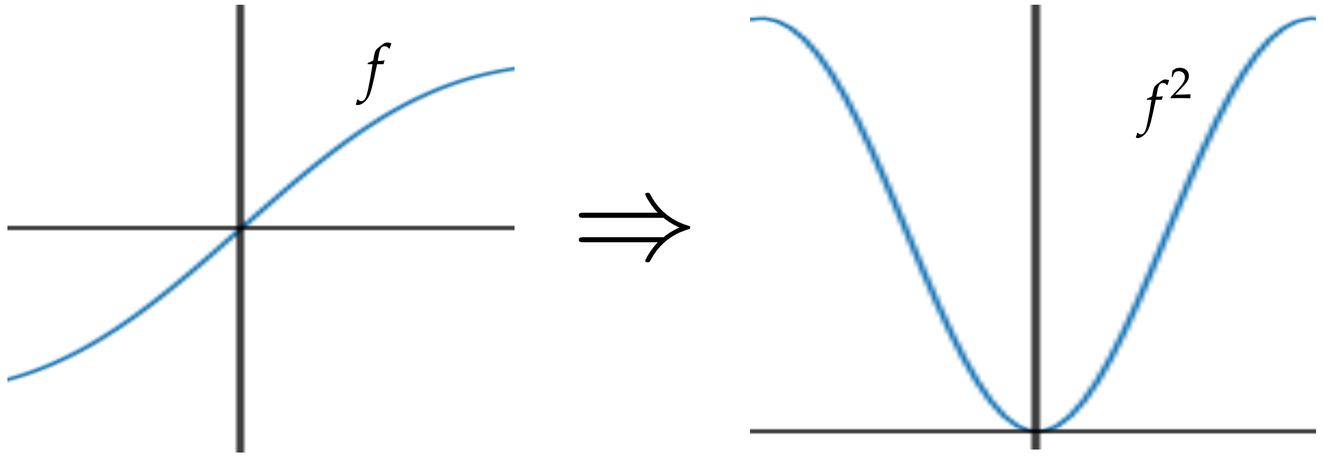
- in fact, every unconstrained optimization problem can be expressed as a root finding problem
  - and, for non-linear systems, an optimization $x_* = \underset{x}{\operatorname{argmin}} f(x)$ is equivalent to a root finding problem $f'(x_*) = 0$
    - we've seen this (albeit, for vector-valued input) in linear least squares:

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 \Leftrightarrow \frac{\partial}{\partial \mathbf{x}} \left[\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\mathbf{b}^T\mathbf{A}\mathbf{x} + \|\mathbf{b}\|_2^2\right] = 0$$

Root Finding vs. Unconstrained Optimization

While distinct, root finding and unconstrained optimization problems can be related to each other

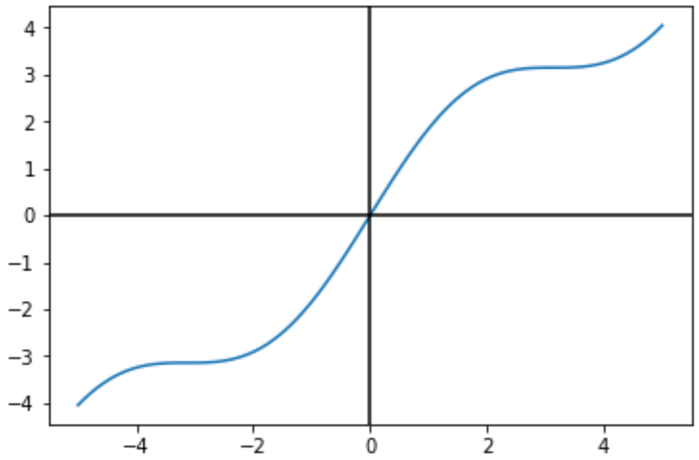
- every root finding problem can be expressed as a dual unconstrained optimization problem
  - finding the root x^* s.t. $f(x^\star) = 0$ is equivalent to minimizing $f^2(x^\star)$



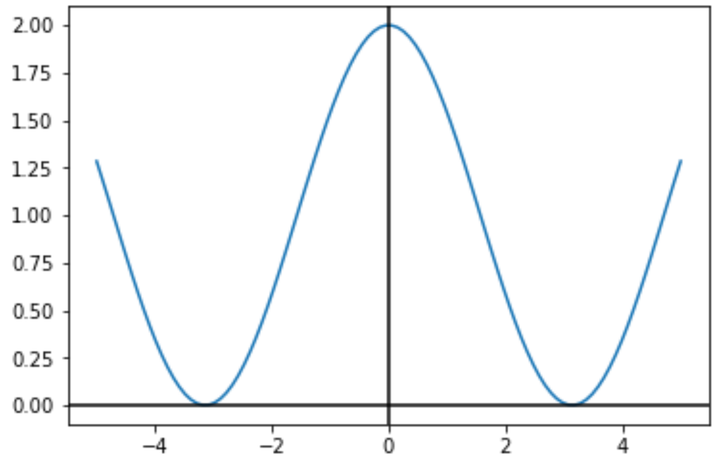
Root Finding vs. Unconstrained Optimization

These present necessary, but not sufficient, conditions for mapping between optimization and root finding:

- not every root of f' corresponds to a minimum/maximum of f



$$\Rightarrow$$

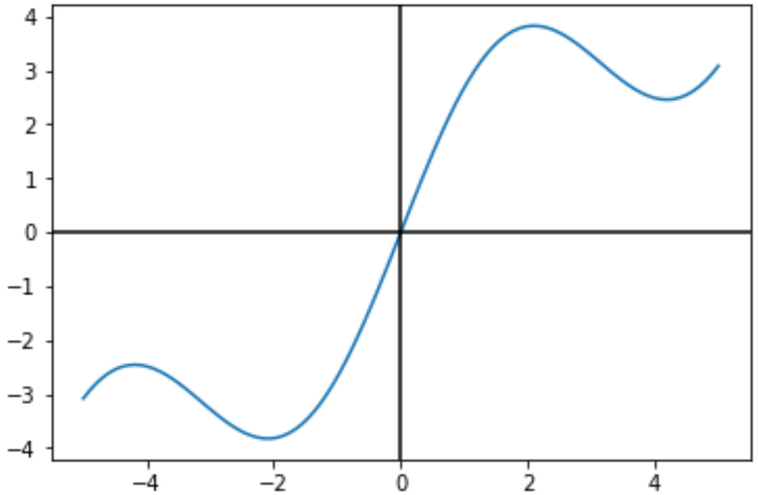


$$f(x) = \sin(x) + x \qquad f'(x) = \cos(x) + 1$$

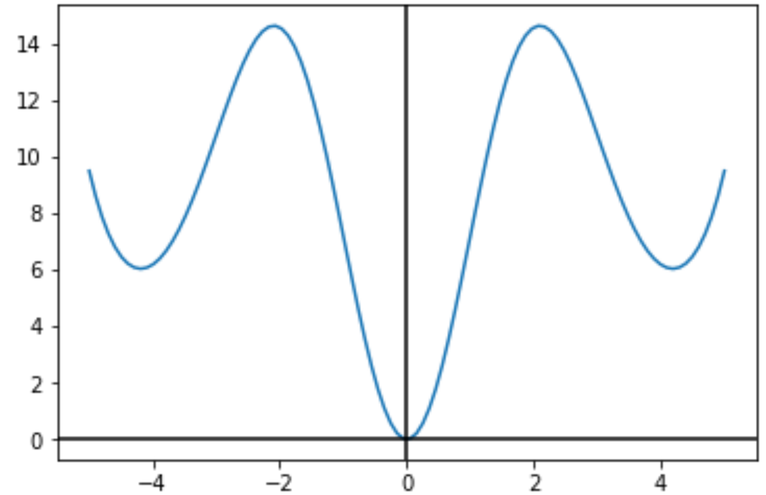
Root-finding vs. Unconstrained Optimization

These present necessary, but not sufficient, conditions for mapping between optimization and root-finding:

- not every minimum of f^2 corresponds to a root of f



$$\Rightarrow$$



$$f(x) = 2\sin(x) + x \qquad f^2(x) = (2\sin(x) + x)^2$$

1D Optimization – Unimodal Functions

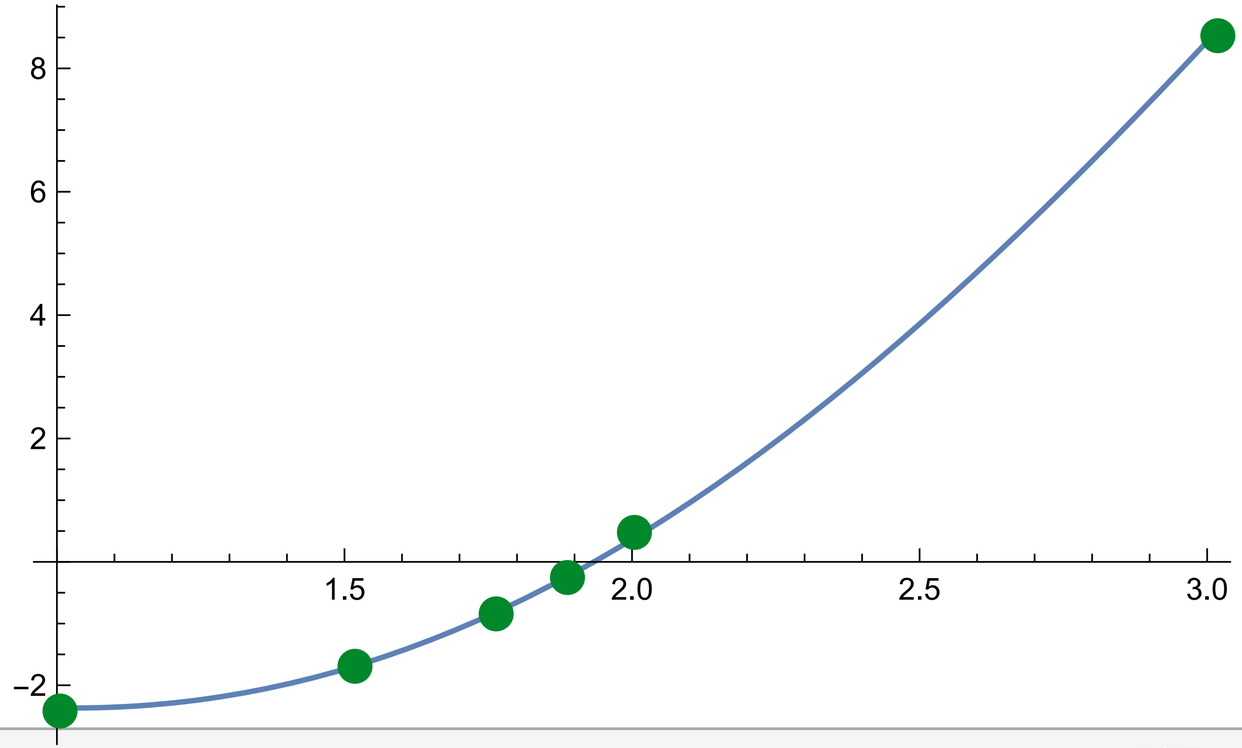
Extending Bisection to Optimization

Recall the (slow) bisection method for root finding

- makes minimal assumptions on the function (continuity)
- requires no gradient information

It's almost always useful to have at least one such simple – albeit inefficient – strategy in our toolbox

- can we extend the approach of bisection to minimization?



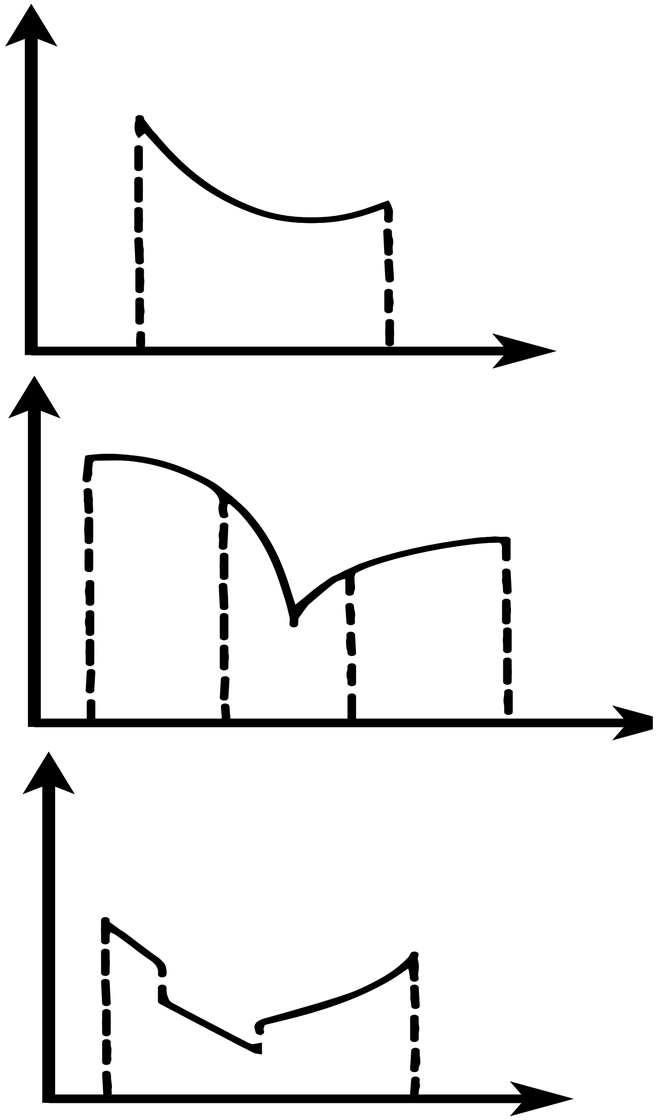
Extending Bisection to Optimization

The IVT does not apply naturally to extrema of a function

- need a setting where a concept equivalent to the IVT can apply

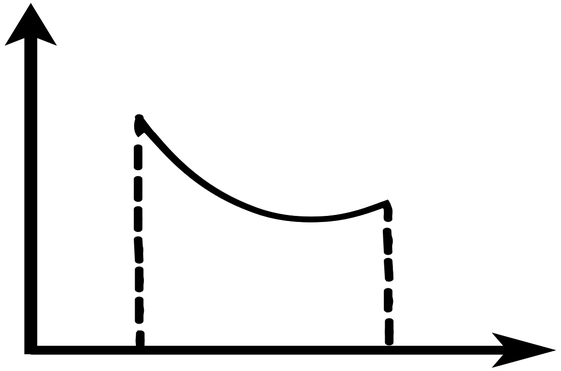
Unimodal Functions

- $f : \mathbb{R} \to \mathbb{R}$ is *unimodal* if there is an $x^* \in [a, b]$ an such that $f(x)$ is non-increasing for $x \in [a, x^*]$ and non-decreasing for $x \in [x^*, b]$
  - convex/concave functions are unimodal
  - don't need to be differentiable (e.g., $f(x) = |x|$)



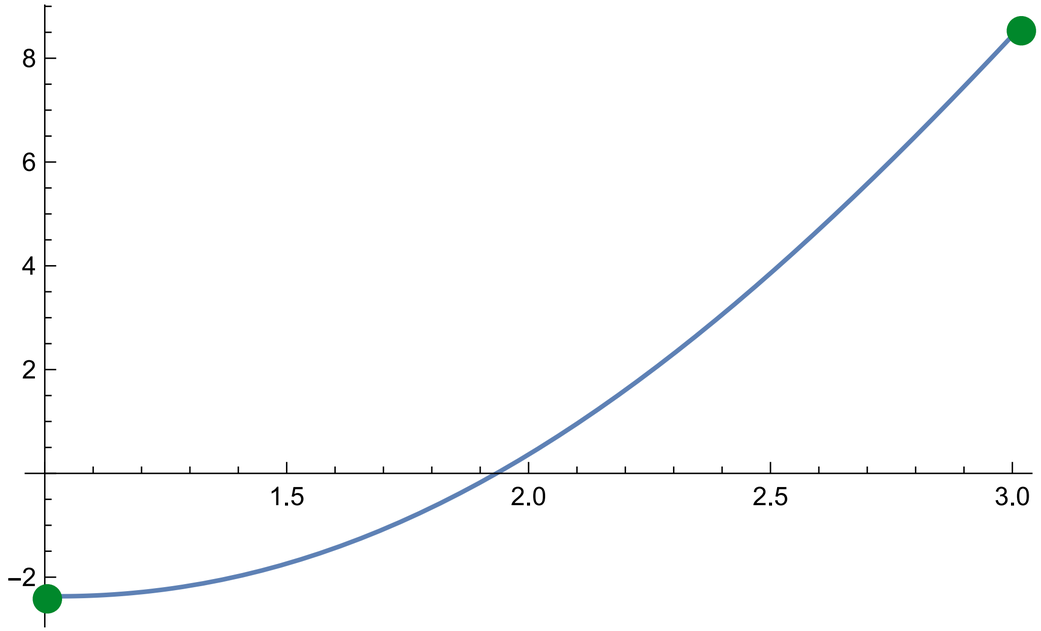
Extending Bisection to Optimization

Unimodular functions are an appealing place to start, as they have a single extrema – once you've found *a* local extrema, you've also found *the* global extremum



- can bisection find the extremum of unimodal functions?

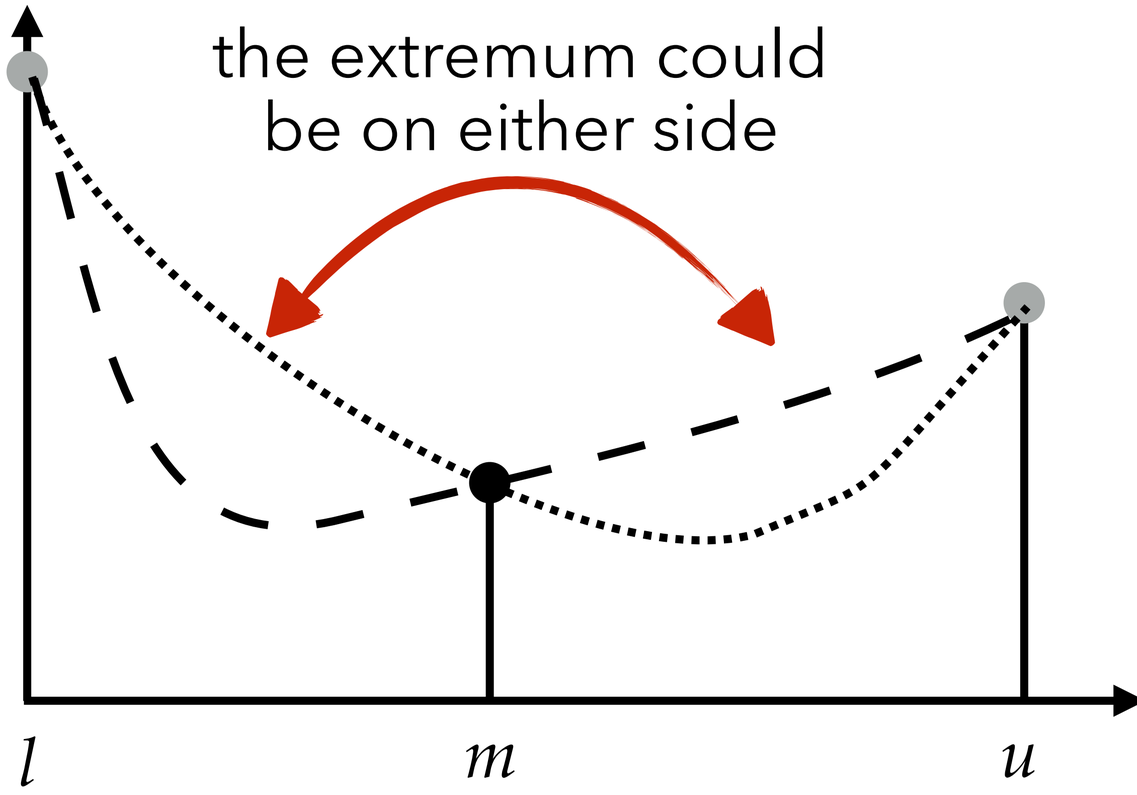
When root finding with bisection, two points are sufficient to bracket a single root



Bracketing an Extremum

Bracketing an extremum is more complicated

- consider an initial bracket for an extremum $[l, u]$, with $l < u$
- let's add the midpoint m and try to reason about which sub-bracket contains the extremum...



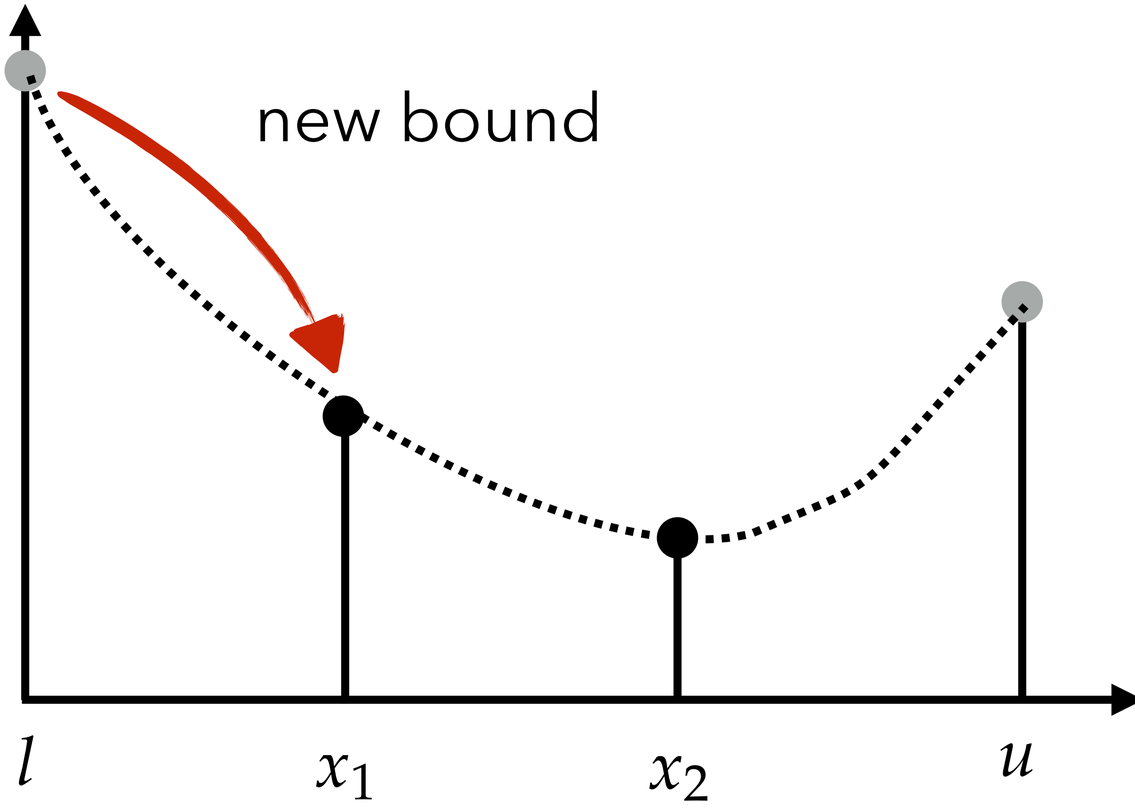
Bracketing an Extremum

Given an initial bracket $[l, u]$

- we sample f at **two** points x_1 and x_2, such that $l < x_1 < x_2 < u$

How to update the bracket?

- if $f(x_1) \geq f(x_2)$, then $f(x) \geq f(x_1)$ for all $x \in [l, x_1]$ and we can update our new bracket as $[x_1, u]$



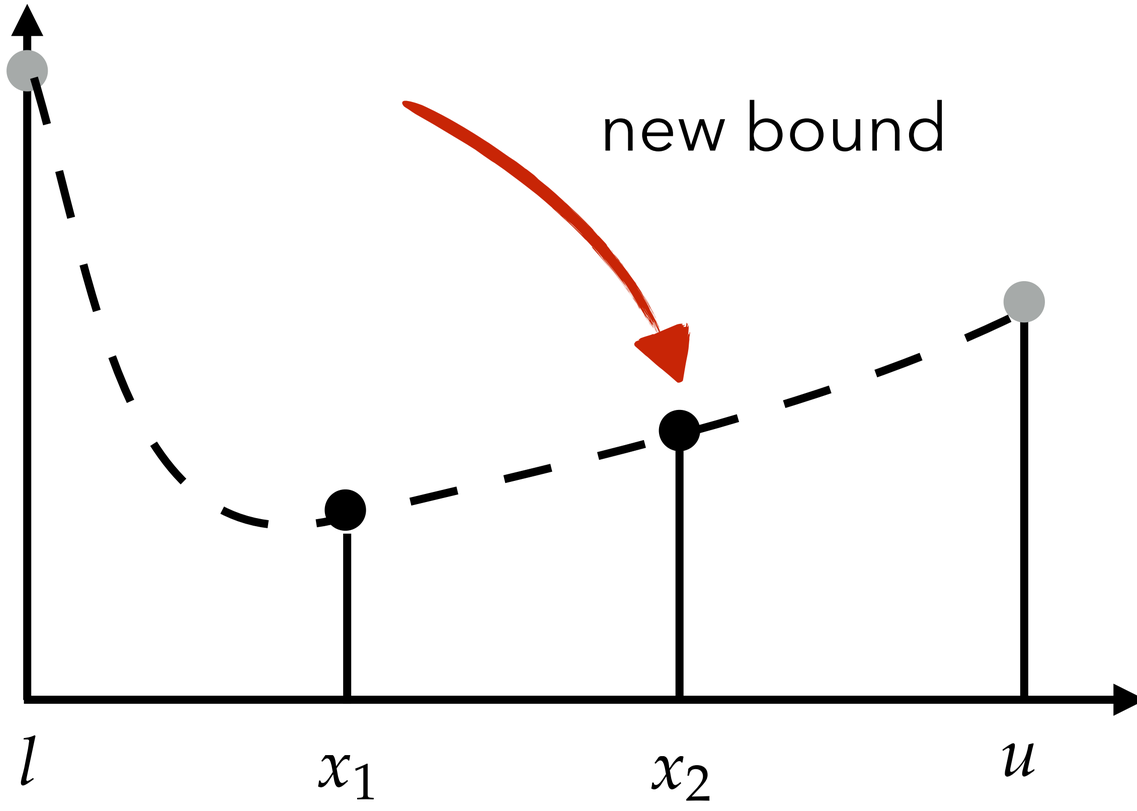
Bracketing an Extremum

Given an initial bracket $[l, u]$

- we sample f at **two** points x_1 and x_2, such that $l < x_1 < x_2 < u$

How to update the bracket?

- similarly, if $f(x_2) \geq f(x_1)$, we update the bracket as $[l, x_2]$

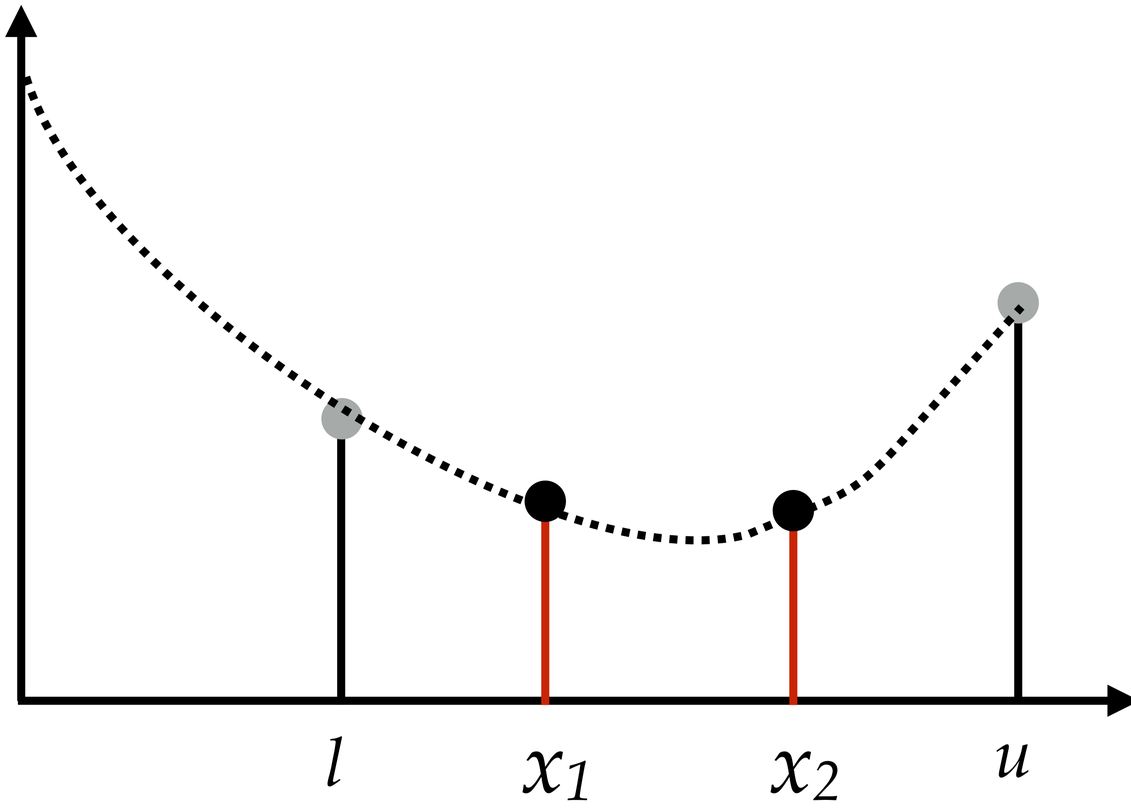


Bracketing an Extremum

One remaining questions is: how do we *choose* points x_1 and x_2?

<u>Strategy #1</u>

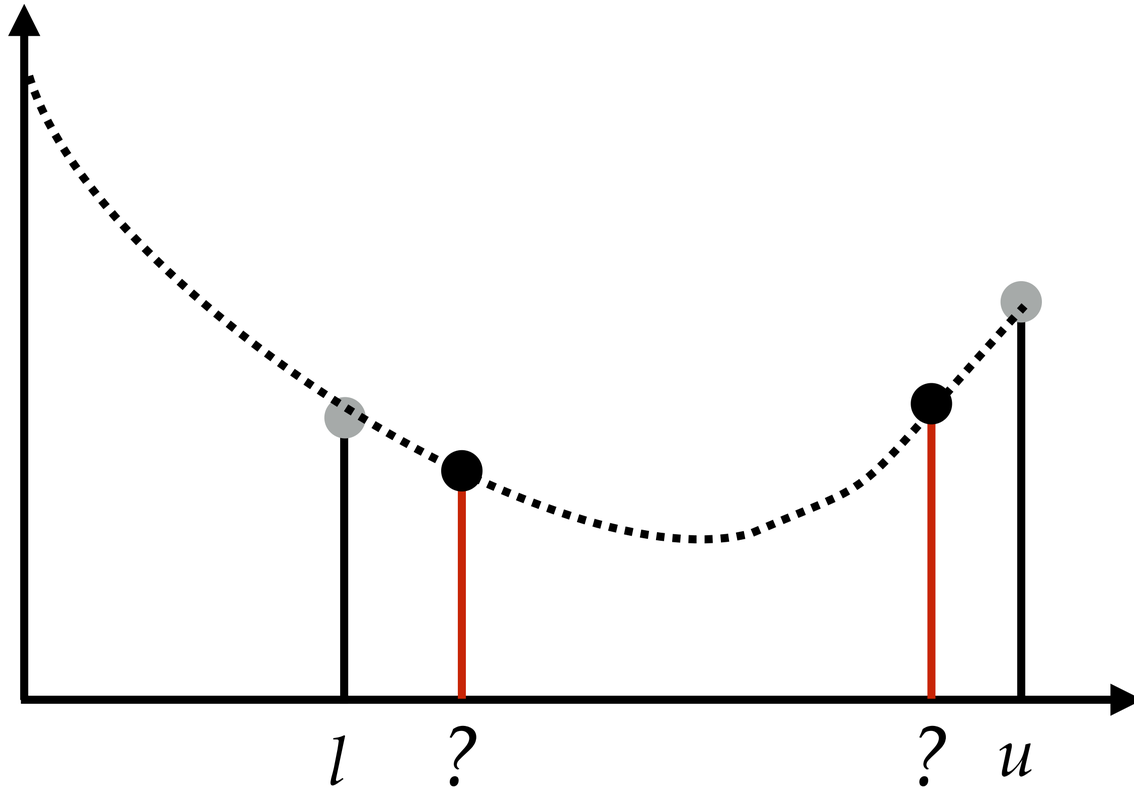
- equally-spaced between l and u
  - divides the bracket into thirds
  - after update, only 2/3 remain
- requires **two new** function evaluations, per iteration



Bracketing an Extremum

One remaining questions is: how do we *choose* points x_1 and x_2?

- what if evaluating f is expensive?
  - can we place x_1 and x_2 so that, at each iteration, we can reuse one of these (previous) points



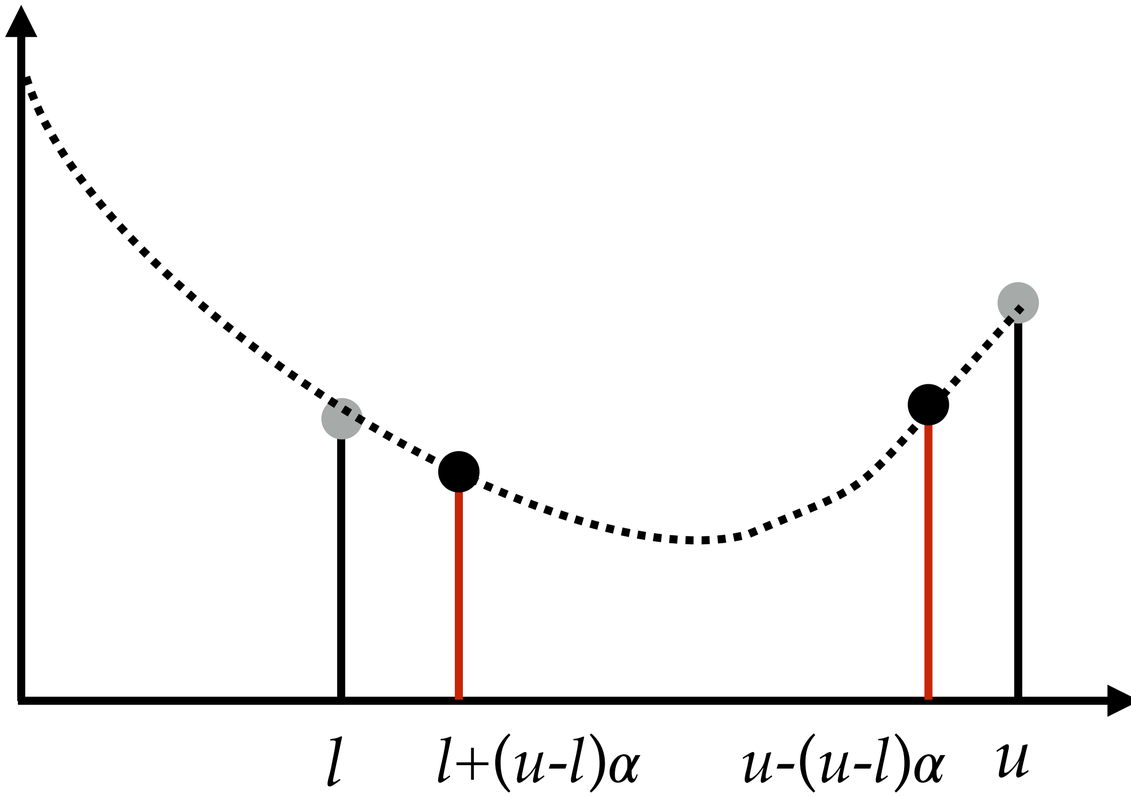
Strategy #2

- without any knowledge of f, we can choose x_1 and x_2:
  - symmetrically, and
  - in a manner that allows us to reuse $f(x_1)$ or $f(x_2)$ in the next iteration

Bracketing an Extremum

Strategy #2

- without any knowledge of f, we can choose x_1 and x_2:
  - symmetrically, and
    - place x_1 as far from l, as x_2 is from u
  - in a manner that allows us to reuse $f(x_1)$ or $f(x_2)$ in the next iteration

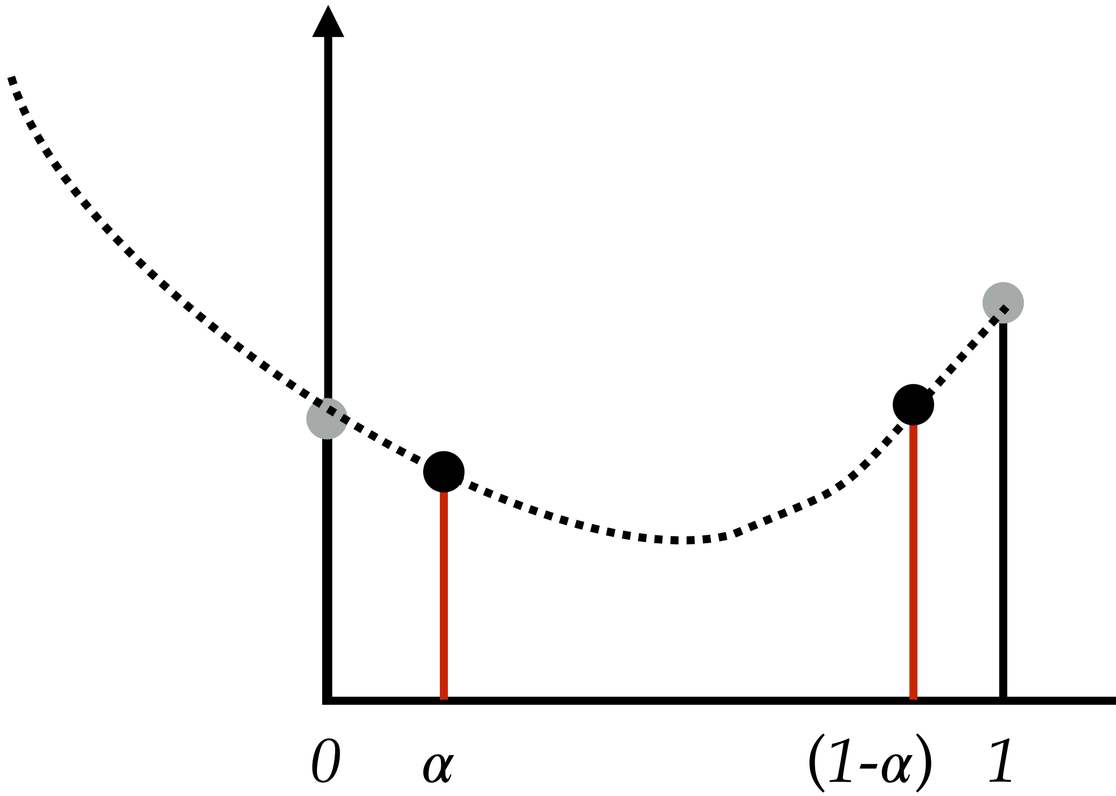


$\alpha \in [0, ½]$

Bracketing an Extremum

Strategy #2

- without any knowledge of f, we can choose x_1 and x_2:
  - symmetrically, and
    - place x_1 as far from l, as x_2 is from u
  - in a manner that allows us to reuse $f(x_1)$ or $f(x_2)$ in the next iteration

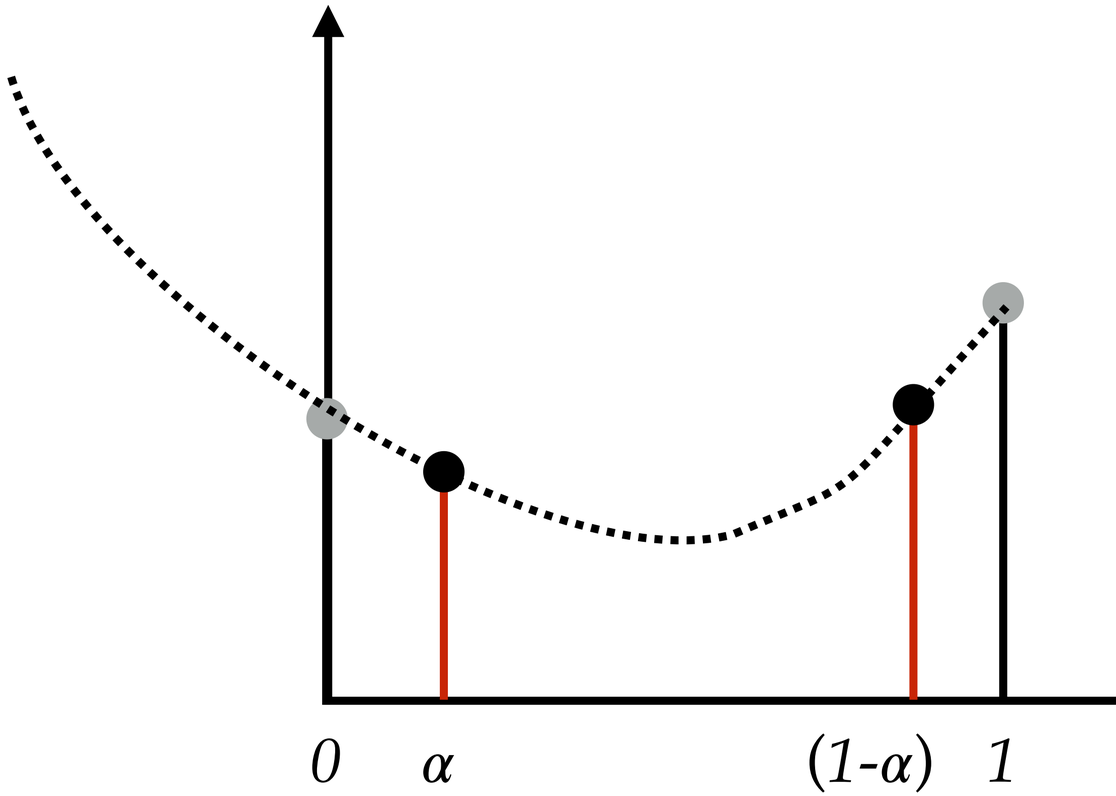


WLOG, we can assume $l = 0$ and $u = 1$, and extend our results to arbitrary $[l, u]$ by translating and scaling relative ratios

Bracketing an Extremum

Strategy #2

- without any knowledge of f, we can choose x_1 and x_2:
  - symmetrically, and
    - place x_1 as far from l, as x_2 is from u
  - in a manner that allows us to reuse $f(x_1)$ or $f(x_2)$ in the next iteration



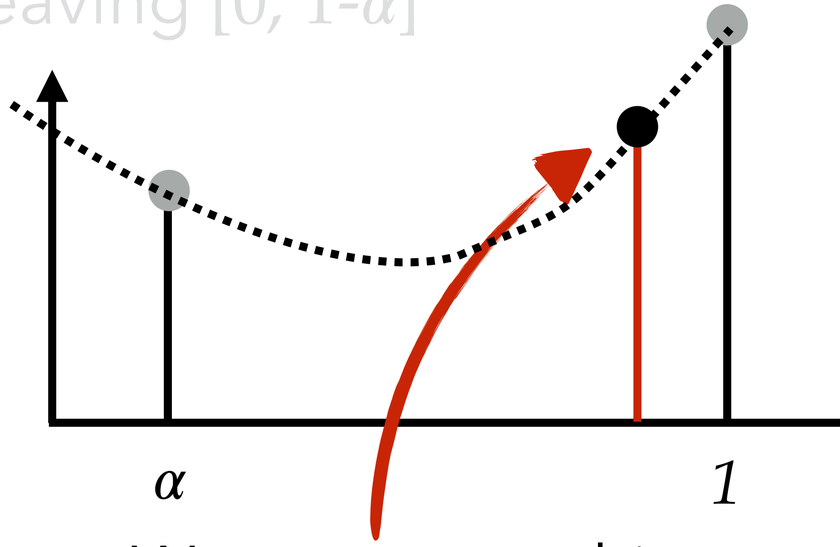
How to pick α so that $f(\alpha)$ or $f(1\text{-}\alpha)$ can be reused in the next iteration?

- case 1: the left subinterval $[0, \alpha]$ is discarded, leaving a new interval $[\alpha, 1]$
- case 2: the right subinterval $[1\text{-}\alpha, 1]$ is discarded, leaving $[0, 1\text{-}\alpha]$

Bracketing an Extremum

How to pick α so that $f(\alpha)$ or $f(1\text{-}\alpha)$ can be reused in the next iteration?

- case 1: the left subinterval $[0, \alpha]$ is discarded, leaving a new interval $[\alpha, 1]$
- case 2: the right subinterval $[1\text{-}\alpha, 1]$ is discarded, leaving $[0, 1\text{-}\alpha]$



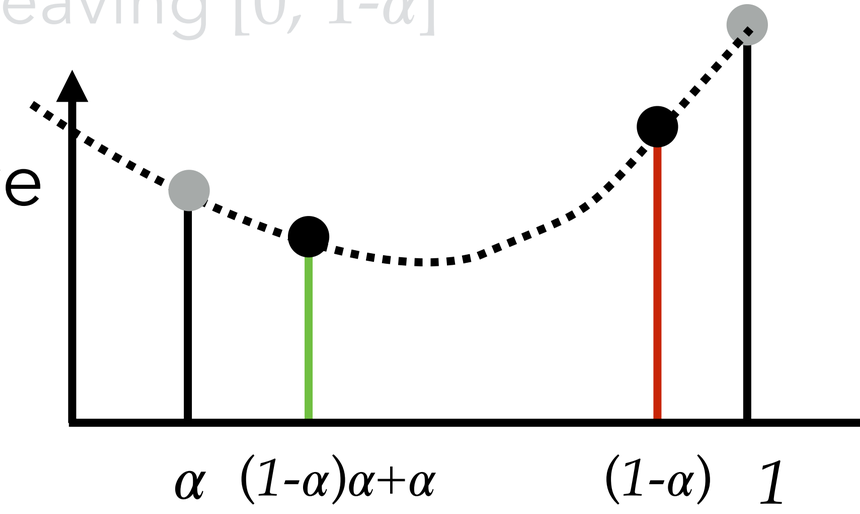
Want to reuse this point's function evaluation

Bracketing an Extremum

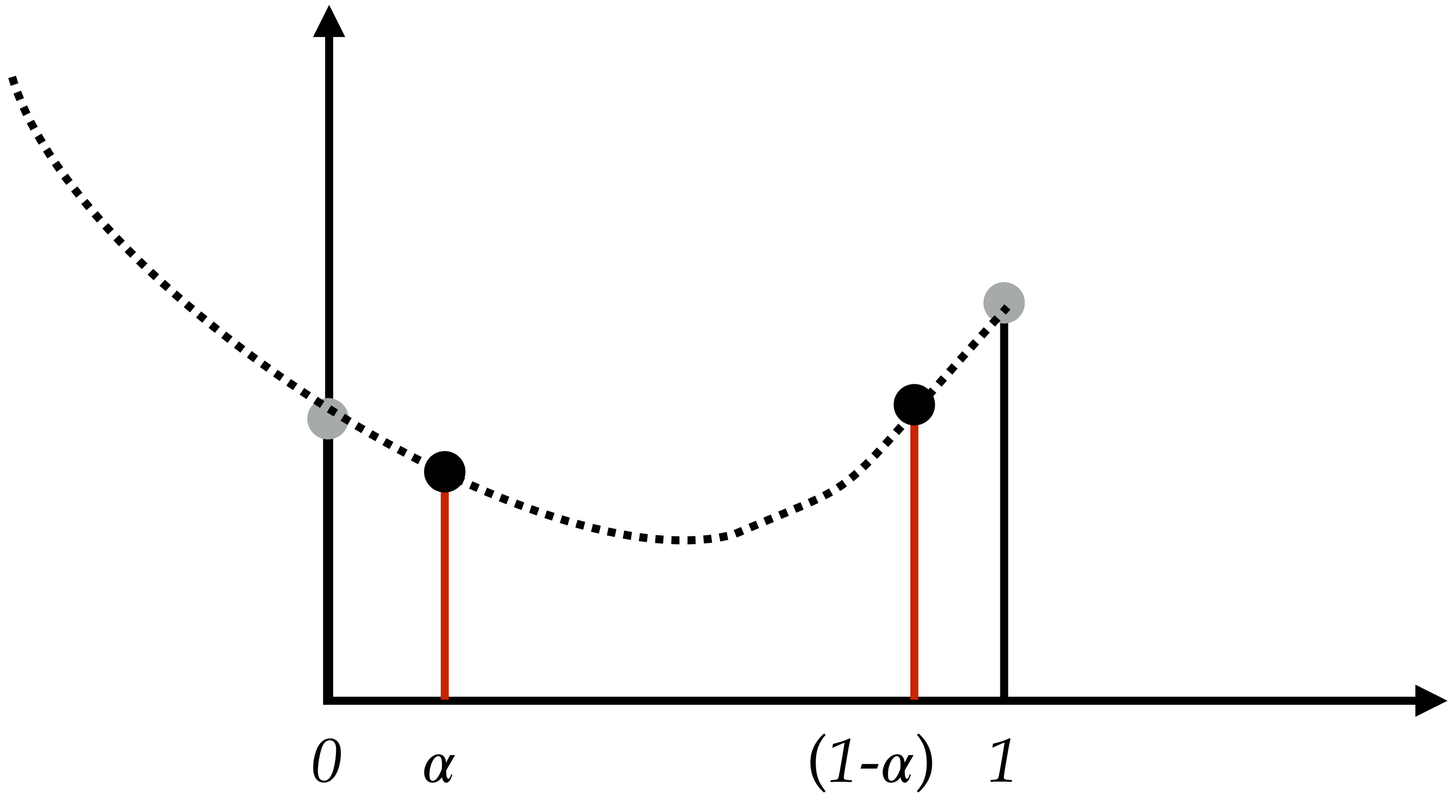
How to pick α so that $f(\alpha)$ or $f(1\text{-}\alpha)$ can be reused in the next iteration?

- case 1: the left subinterval $[0, \alpha]$ is discarded, leaving a new interval $[\alpha, 1]$
- case 2: the right subinterval $[1\text{-}\alpha, 1]$ is discarded, leaving $[0, 1\text{-}\alpha]$

To reuse $f(1\text{-}\alpha)$ we can set $1\text{-}\alpha = (1\text{-}\alpha)\alpha + \alpha$ and solve for $\alpha = (3 - \sqrt{5})/2$ and $1\text{-}\alpha = (\sqrt{5} - 1)/2$



Bracketing an Extremum



How to pick α so that $f(\alpha)$ or $f(1\text{-}\alpha)$ can be reused in the next iteration?

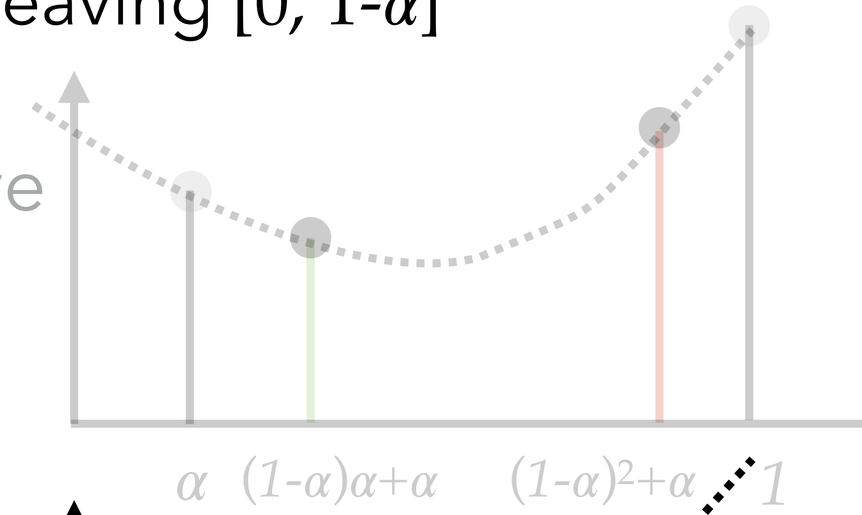
- case 1: the left subinterval $[0, \alpha]$ is discarded, leaving a new interval $[\alpha, 1]$
- case 2: the right subinterval $[1\text{-}\alpha, 1]$ is discarded, leaving $[0, 1\text{-}\alpha]$

Bracketing an Extremum

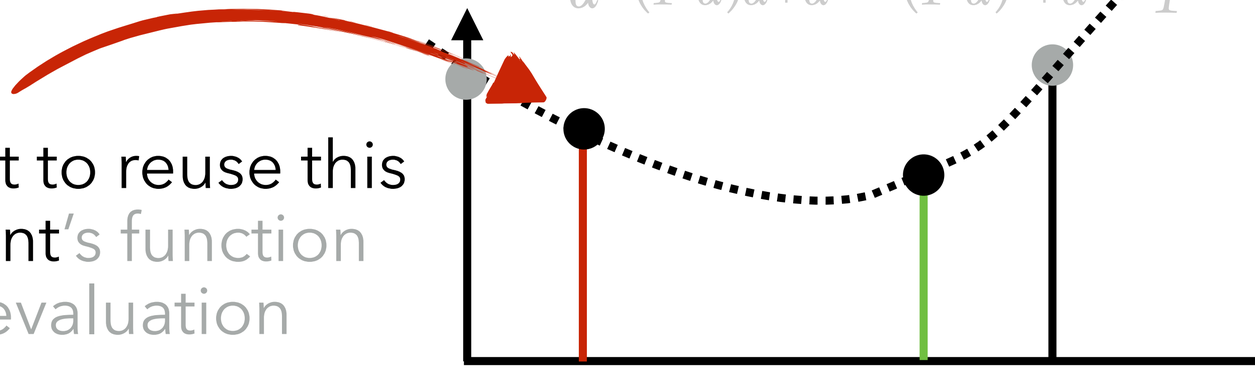
How to pick α so that $f(\alpha)$ or $f(1\text{-}\alpha)$ can be reused in the next iteration?

- case 1: the left subinterval $[0, \alpha]$ is discarded, leaving a new interval $[\alpha, 1]$
- case 2: the right subinterval $[1\text{-}\alpha, 1]$ is discarded, leaving $[0, 1\text{-}\alpha]$

To reuse $f(1\text{-}\alpha)$ we can set $1\text{-}\alpha = (1\text{-}\alpha)\alpha + \alpha$ and solve for $\alpha = (3 - \sqrt{5})/2$ and $1\text{-}\alpha = (\sqrt{5} - 1)/2$



Want to reuse this point's function evaluation

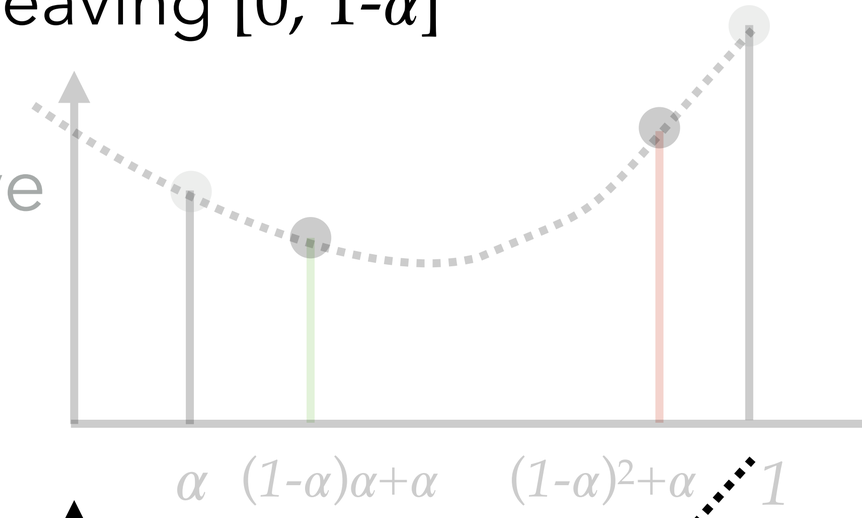


Bracketing an Extremum

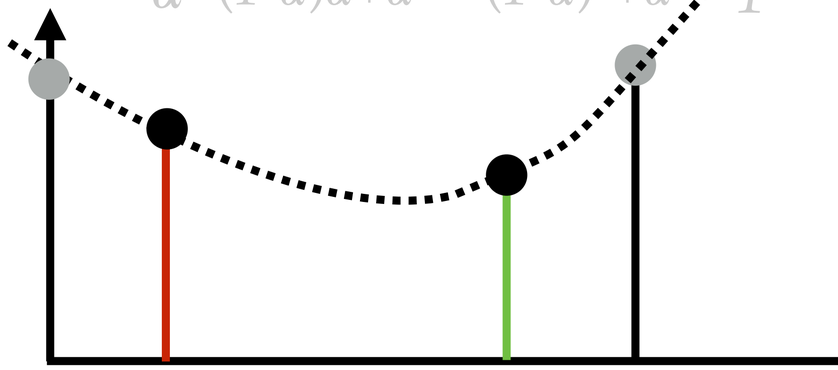
How to pick α so that $f(\alpha)$ or $f(1\text{-}\alpha)$ can be reused in the next iteration?

- case 1: the left subinterval $[0, \alpha]$ is discarded, leaving a new interval $[\alpha, 1]$
- case 2: the right subinterval $[1\text{-}\alpha, 1]$ is discarded, leaving $[0, 1\text{-}\alpha]$

To reuse $f(1\text{-}\alpha)$ we can set $1\text{-}\alpha = (1\text{-}\alpha)\alpha + \alpha$ and solve for $\alpha = (3 - \sqrt{5})/2$ and $1\text{-}\alpha = (\sqrt{5} - 1)/2$



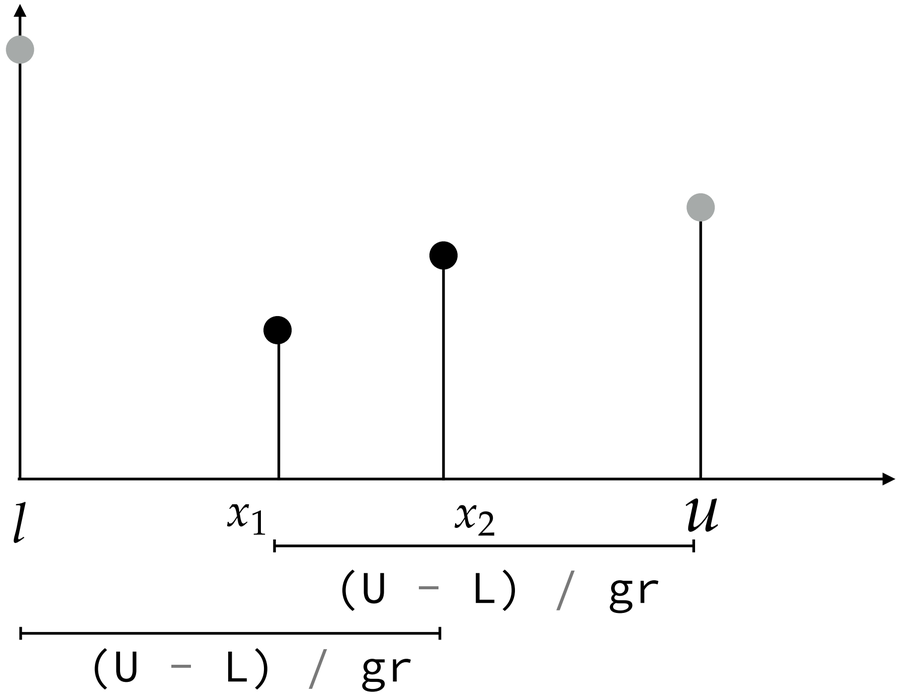
To reuse $f(\alpha)$ we can set $\alpha = (1\text{-}\alpha)^2$ and solve for $\alpha = (3 - \sqrt{5})/2$ and $1\text{-}\alpha = (\sqrt{5} - 1)/2$



Golden Section Search

Golden section search is a subdivision algorithm for minimizing unimodular functions, using one new evaluation per iteration

- any idea why it's called that?




```
gr = (math.sqrt(5) + 1) / 2
def gss(f, L, U, tol = 1e-5):
  while True:
    x1 = U - (U - L) / gr
    x2 = L + (U - L) / gr
    if abs(U - L) < tol:
      break
    if f(x1) < f(x2):
      U = x2
    else:
      L = x1
  return (L + U) / 2
```

Golden Section Search – Summary

Golden section search is

+ a simple approach for minimization/maximization
+ does not require gradient information
+/– only applies to unimodular functions
– linear convergence rate
– hard to scale to higher-dimensional problems

1D Optimization – Higher-order Methods

Optimization with Gradient Information

Recall

- Newton's root finding method leverages explicit gradient information to accelerate root-finding
- the secant method approximates the gradient numerically
- extrema x^* of f must satisfy $\nabla f(x^*) = 0$ (a root-finding problem)

As such, we can imagine extending these two higher-order methods to finding an extremum of a function $f : \mathbb{R} \to \mathbb{R}$

Newton's Method for Optimization

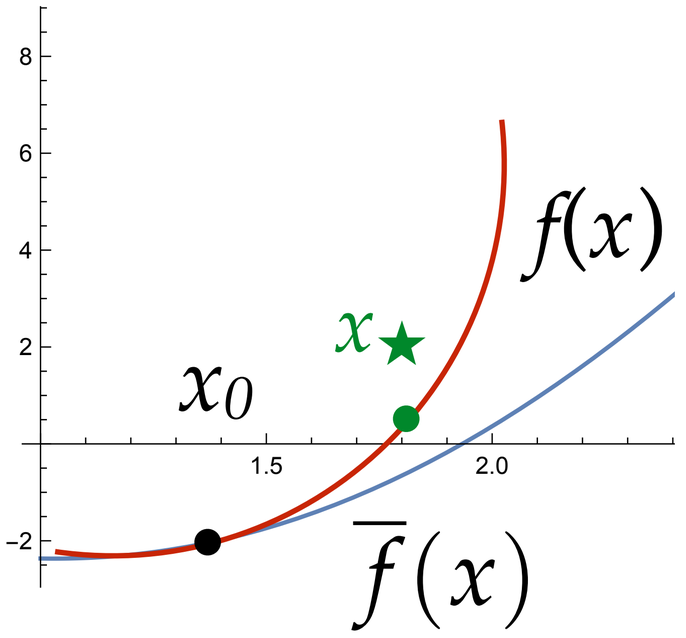
Starting from a 2^{nd}-order Taylor approximation of a (twice differentiable) function f

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + f''(x_0)(x - x_0)^2 \Big/ 2 = \overline{f}(x)$$

one iterative strategy seeks stationary points* of f as the extremum of the *quadratic approximation* $\overline{f}$, a parabola that is:

- tangent to f at $x = x_0$ and,
- with vertex $x_\star = x_0 - f'(x_0)/f''(x_0)$

Why do we need 2^{nd}-order accuracy, here?



Newton's Method for Optimization

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + f''(x_0)(x - x_0)^2 \Big/ 2 = \overline{f}(x)$$

Each iteration estimates x_{k+1} of a stationary point of f as the vertex of the quadratic approximation $\overline{f}$ centered at the previous stationary point estimate x_k

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

which we can obtain by solving for x in $d\overline{f}/dx = 0$

- recall: extrema satisfy the root finding problem $\nabla f(x^*) = 0$

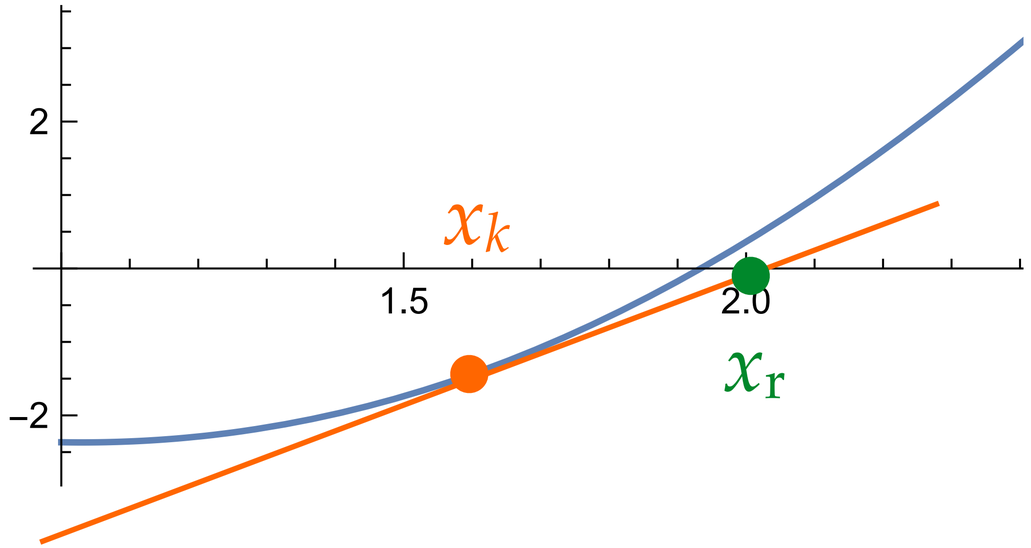
From Newton to Secant for Optimization

If evaluating the 1^{st}- and 2^{nd}-derivatives is not possible (or too expensive), we can approximate them numerically

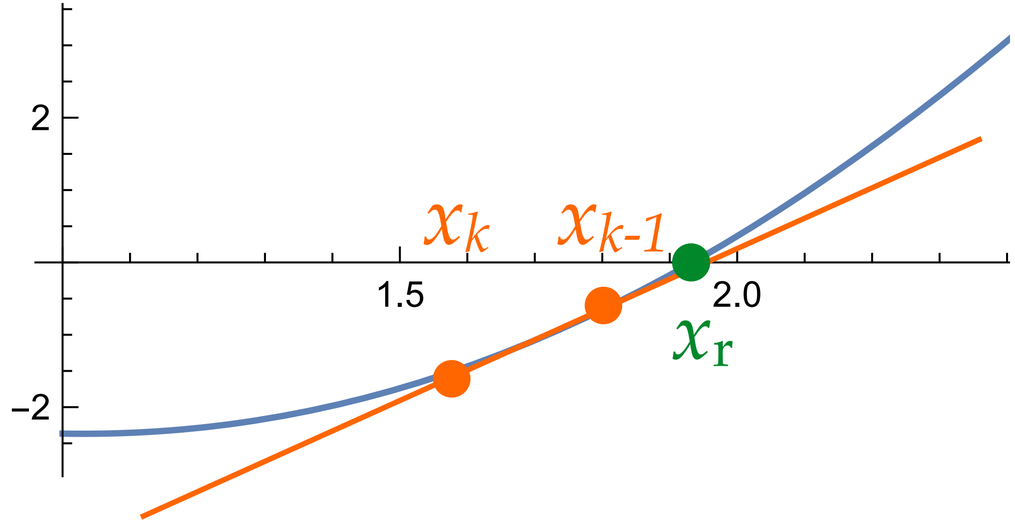
- the Newton optimization method finds stationary points of the second-order approximation $\overline{f}(x)$ of $f(x)$
- we can apply the secant method to numerically approximate the 2^{nd}-derivative in the Newton update $x_{k+1} = x_k - f'(x_k)/f''(x_k)$
  - still requires evaluating $f'(x)$
  - rare to have access to f', but not f'', in a 1D setting
  - can we compute *everything* numerically?

From Newton to Secant for Optimization

Newton root finding uses a line **tangent** to f at the current iterate x_k, using exact 1^{st}-derivatives f' at x_k



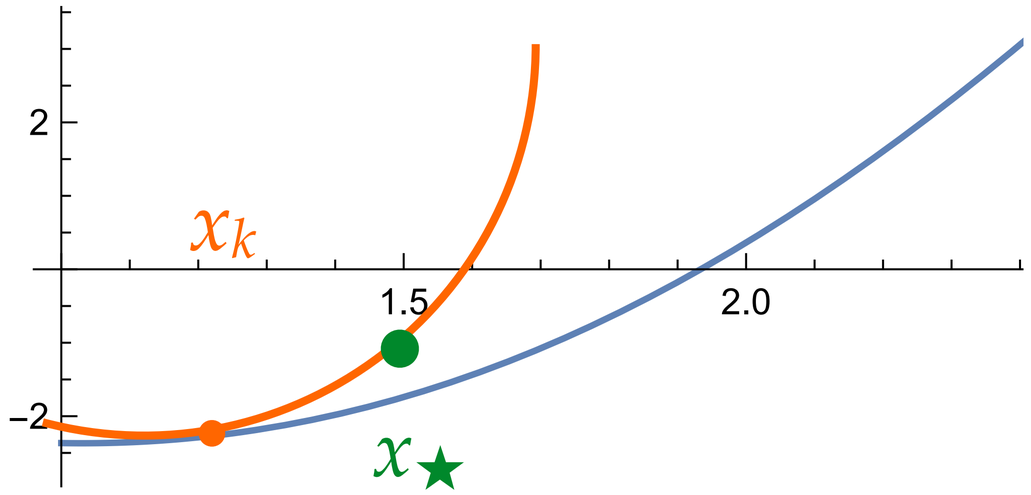
Secant root finding fits a line through the current and previous iterates' function evaluations, $f(x_k)$ and $f(x_{k-1})$



- both methods approximate a root estimate as the root x_r of the line

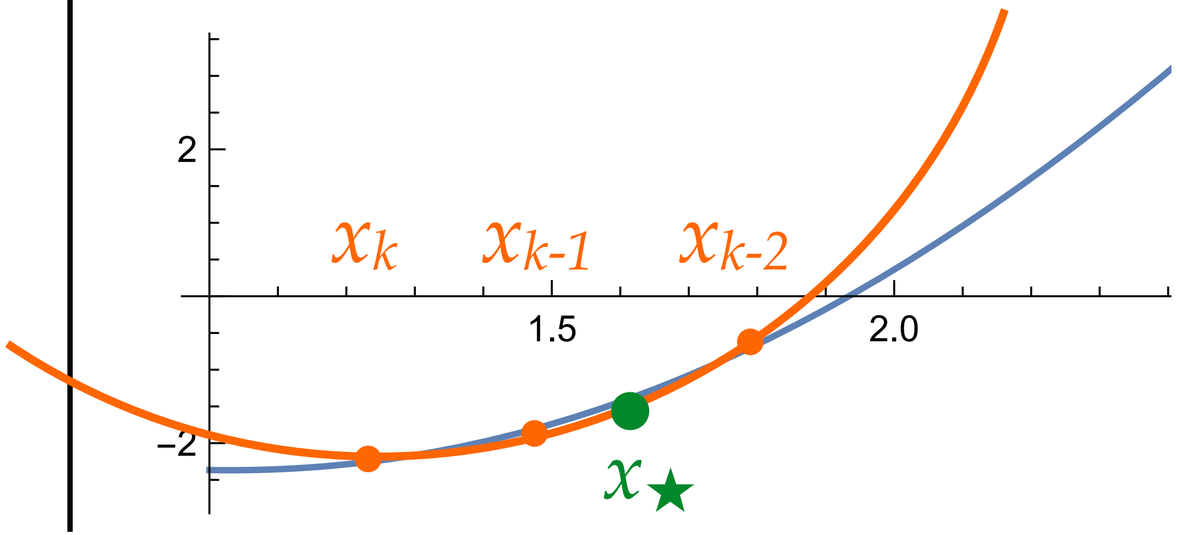
From Newton to Secant for Optimization

Newton *optimization* uses a parabola **tangent** to f at the current iterate x_k, using exact 1^{st}- and 2^{nd}-derivatives at x_k



Secant *optimization* fits a parabola through the previous iterates

- how many points do we need?



- both approximate a stationary point as the vertex $x_\star$ of the parabola

1D Optimization – Convergence

The $f'(x_r) = 0$ condition is *necessary*, but not *sufficient*, to determine whether we have converged to an extremum

In the 1D setting, you need to additionally evaluate the 2^{nd}-derivative at your stationary point estimate x_r

- if it's strictly greater than zero: minimum
- if it's strictly less than zero: maximum
- if it's equal to zero: inflection point

1D Optimization – Summary

- Golden section search is a bisection-like algorithm for finding the (global) optimum of a unimodular function
- Newton's method can be extended to optimization, leveraging the fact that extrema satisfy a dual root-finding problem
  - evaluates 1^{st}- and 2^{nd}-derivatives to form a 2^{nd}-order expansion that is then used to estimate a stationary point, as $x_{k+1} = x_k - f'(x_k)/f''(x_k)$
  - can also numerically estimate the 2^{nd}-derivative evaluation, here
- Can also extend the secant method to fit a parabola to previous iterates and solve for its vertex as an estimate of the stationary point
  - also known as *successive parabolic* interpolation