Lec7p1, ORF363/COS323

PRINCETON UNIVERSITY
ORFE
Instructor:
Amir Ali Ahmadi
Fall 2014

TAs: Y. Chen,
G. Hall,
J. Ye

This lecture:

- One-dimensional line search (root finding and minimization)
- Bisection
- Newton's method
- Secant method
- Introduction to rates of convergence

In this lecture, we'll see iterative methods for minimizing a function $f: \mathbb{R} \to \mathbb{R}$.

- A univariate optimization problem has pretty limited use, but:
  - One-dimensional line search will be used as a subroutine in future lectures for multivariate optimization.
  - Some algorithms that we see here (e.g., Newton's method) will directly generalize to several dimensions.
  - Some of the difficulties for multivariate optimization already appear in dimension 1 (e.g., lack of global convergence).

Of course, if the problem is 1-D, you can just plot the function and "look at it" to find the global minimum!

- But what does "plotting" mean? It means evaluating the function at many, many point.
- Ideally, we would like to minimize the number of function evaluations.

We will see three algorithms:

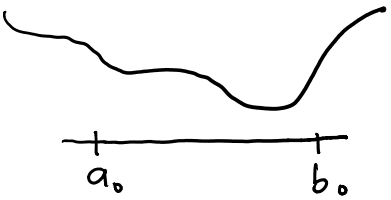
- Bisection (uses f')
- Newton's method (uses f' and f'')
- Secant method (uses f')

Bisection

- Have you ever looked for a book in a library?
- Or searched for a word in the dictionary?
- If you have, then you already know bisection!

Suppose

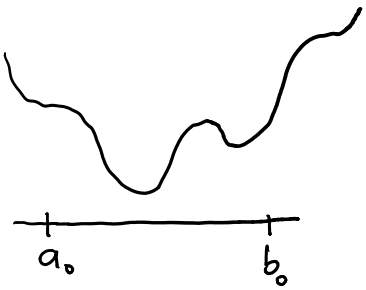
- $f: \mathbb{R} \to \mathbb{R}$ is continuously differentiable.
- We know (a possibly large) interval $[a_0, b_0]$ where its minimum lives.
- f is unimodal on $[a_0, b_0]$; i.e., the derivative changes sign only once. Further, assume there is only one local minimum in the interval.



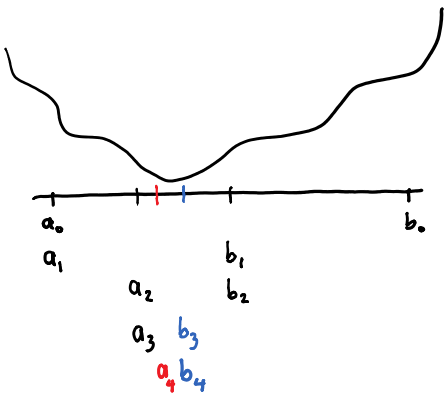
Unimodal

Here is the bisection algorithm:

for $k = 0{:}N$
  if $f'(\frac{x+y}{2}) = 0$
    you are done;
  elseif $f'(\frac{x+y}{2}) > 0$
    $a_{k+1} = a_k$;
    $b_{k+1} = (a_k + b_k)/2$;
  else (if $f'(\frac{x+y}{2})<0$)
    $b_{k+1} = b_k$;
    $a_{k+1} = (a_k + b_k)/2$;
  end
end



Not unimodal



Remarks on bisection:

- In each iteration the length of the interval where the minimum lies is cut in half; i.e.
$$|a_{k+1} - b_{k+1}| = \frac{1}{2}|a_k - b_k|$$
- Suppose we are happy if we isolate the minimum within an interval of length ϵ. How many steps should we take?
$$N = \log_2 \frac{|a_0 - b_0|}{\epsilon}$$
(verify this)
- We can also use bisection for root finding
  - In fact, that's what the algorithm is doing on f'
  - If f' is continuous, then bisection will find (a) root
  - In each step the root is sandwiched between our new endpoints
- Bisection is a safe and robust algorithm
  - But not as fast as the next two algorithms we'll see
  - Since our intervals are halving in every step, bisection has "linear convergence" (notion formalized later)
  - Essentially, in each iteration, we get one more correct significant digit of the root

In many applications, we know a priori an initial interval $[a_0, b_0]$. But what if we don't?

- To find a bracket containing minimizer, it suffices to find 3 points $a < c < b$ such that $f(c) < f(a)$ and $f(c) < f(b)$

Pick $x_0 < x_1 < x_2$. ① If $f(x_1) < f(x_0)$, $f(x_1) < f(x_2)$ → done.

– ② If $f(x_0) > f(x_1) > f(x_2)$

Pick $x_3 > x_2$ (e.g., s.t. $(x_3 - x_2) = 2(x_2 - x_1)$)

If $f(x_2) < f(x_3)$ → done (bracket is $[x_1, x_3]$)

Else, continue (until function increases.)

– ③ If $f(x_0) < f(x_1) < f(x_2)$ → similar.

A common use of bisection in optimization

Consider an optimization problem:

$$\begin{aligned} &\min. f(x) \\ &\text{s.t.} \quad g_i(x) \leq 0 \end{aligned} \qquad (f, g_i: \mathbb{R}^n \to \mathbb{R})$$

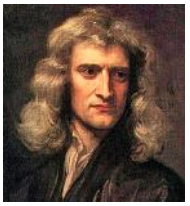
- Suppose we have a black box that can test for feasibility - it tells us whether the set $\{x | \ g_i(x) \leq 0\}$ is empty or not.
- How can use the black box to solve our optimization problem?
- Note that our problem is equivalent to the following:

$$\begin{aligned} &\min. \gamma \\ &\text{s.t.} \quad g_i(x) \leq 0, f(x) \leq \gamma \end{aligned}$$

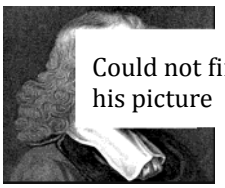
- So we can do bisection on γ. For each fixed γ, call the feasibility black box on the set $\{g_i(x) \leq 0, f(x) \leq \gamma\}$.
  - If feasible, decrease γ.
  - If infeasible, increase γ.

- If we know an interval of length M where f_* lies, we can get f_* with accuracy ϵ if we call our feasibility black box $\log_2 M/\epsilon$ times.
- Hence (in many cases) an efficient algorithm for feasibility testing directly gives an efficient algorithm for optimization.

Newton's method



Newton
(1642-1727)



Raphson
(1648-1715?)

Suppose now that we have access to f' and f''. Newton's method (aka the Newton-Raphson method) for minimization is based on the following iterations:

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

We present two different motivations for deriving this iterative algorithm:

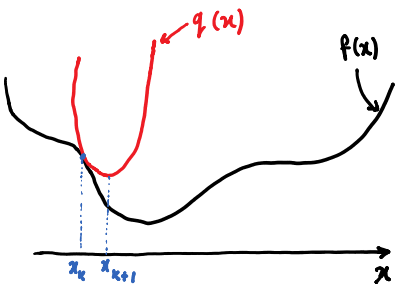
- Local quadratic approximation of f
- Solving the equation $f'(x) = 0$ by the so-called "method of tangents"

Local quadratic approximation

Idea: Let's approximate f locally by a simpler function that we know how to minimize; say, a quadratic function q. If f is quadratic itself, then our approximation is globally correct and we terminate in one step. If it isn't, we move to the minimum of q and re-approximate f again at that point with a new quadratic.

Want to approximate f at a point x_k with a quadratic function:

$$q(x) = ax^2 + bx + c$$



We want to match: $q(x_k) = f(x_k)$ ①
$q'(x_k) = f'(x_k)$ ②
$q''(x_k) = f''(x_k)$ ③

③ $\Rightarrow$ $a = \frac{f''(x_k)}{2}$

② $\Rightarrow$ $f''(x_k)x_k + b = f'(x_k) \Rightarrow b = f'(x_k) - f''(x_k)x_k$

① $\Rightarrow$ $\frac{f''(x_k)}{2}x_k^2 + f'(x_k)x_k - f''(x_k)x_k^2 + c = f(x_k)$

$\Rightarrow$ $c = f(x_k) - f'(x_k)x_k + \frac{1}{2}f''(x_k)x_k^2$

$$q(x) = ax^2 + bx + c$$

- Let's plug in the values of a, b, c that we found:

$$q(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(x_k)(x - x_k)^2$$

- Do you recognize this? This is just the second order Taylor expansion of f around x_k. Not surprising.

- Assuming $a > 0$, the minimum of the quadratic is finite and is achived at $-\frac{b}{2a}$. This minimum is the new point we want to jump to:

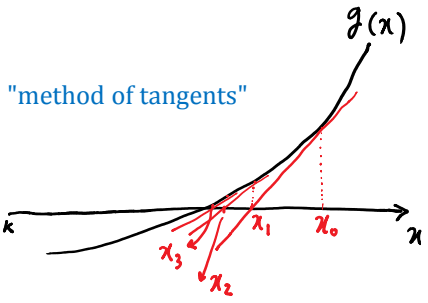
$$x_{k+1} = -\frac{b}{2a} = -\frac{f'(x_k) - f''(x_k)x_k}{f''(x_k)} = x_k - \frac{f'(x_k)}{f''(x_k)}.$$

This is exactly what the Newton method told us to do.

Solving $f'(x) = 0$

- If you saw Newton's method in high school, it was probably to solve equations (find roots), not to minimize functions.
- But it is the same thing; we are in effect finding the roots of f' hoping that they are local minima of f. Let $g(x) = f'(x)$,

At x_k, we approximate g with a line; the zero of the line is our next point. Let's find the equation of the line:

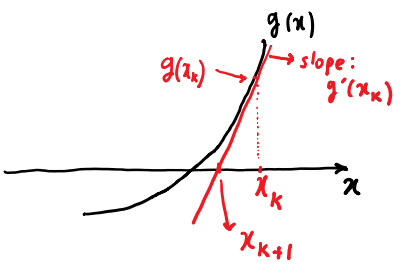


$y = mx + b \rightarrow g(x_k) = \overbrace{g'(x_k)}^{m} x_k + b \Rightarrow b = g(x_k) - g'(x_k)x_k$

$0 = g'(x_k)\,x_{k+1} + (g(x_k) - g'(x_k)x_k)$

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

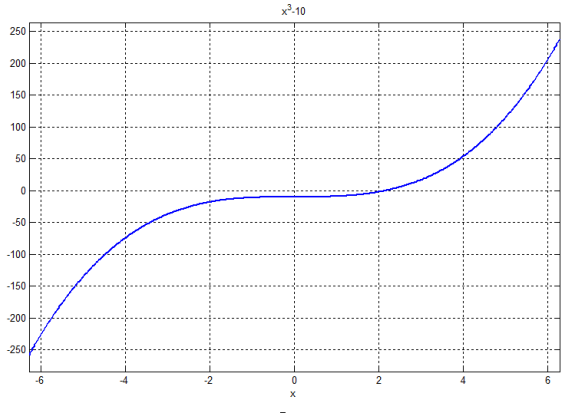
$\therefore\ x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$



This is exactly what the Newton method told us to do.

Good things can happen with Newton's method

Let's see how the Newton and the bisection method compare on a toy problem: Find a root of $g(x) = x^3 - 10$.



Answer is obviously:
>> 10^(1/3)

ans =

2.154434690031884

- We run Newton with $x_0 = 12$, bisection wih $[a_0, b_0] = [0,12]$.

Bisection

Newton

k	x_k	a_k	$\frac{a_k+b_k}{2}$	b_k
1.000000000000000	12.000000000000000	0	6.000000000000000	12.000000000000000
2.000000000000000	8.023148148148149	0	3.000000000000000	6.000000000000000
3.000000000000000	5.400548660419450	0	1.500000000000000	3.000000000000000
4.000000000000000	3.714654390828676	1.500000000000000	2.250000000000000	3.000000000000000
5.000000000000000	2.718005659267038	1.500000000000000	1.875000000000000	2.250000000000000
6.000000000000000	2.263213061967483	1.875000000000000	2.062500000000000	2.250000000000000
7.000000000000000	2.159579216386407	2.062500000000000	2.156250000000000	2.250000000000000
8.000000000000000	2.154446935535738	2.062500000000000	2.109375000000000	2.156250000000000
9.000000000000000	2.154434690101485	2.109375000000000	2.132812500000000	2.156250000000000
10.000000000000000	2.154434690031884	2.132812500000000	2.144531250000000	2.156250000000000
11.000000000000000	2.154434690031884	2.144531250000000	2.150390625000000	2.156250000000000

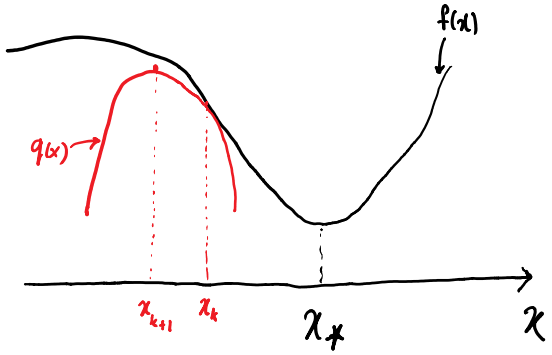
Circled in red: correct significant digits

- The convergence of Newton's method is much faster than bisection
- Number of correct digits *doubles* in each iteration (when the iterates are close enough to the root)
- This is an implication of "*quadratic convergence*"
  - We'll see more about this in upcoming lectures

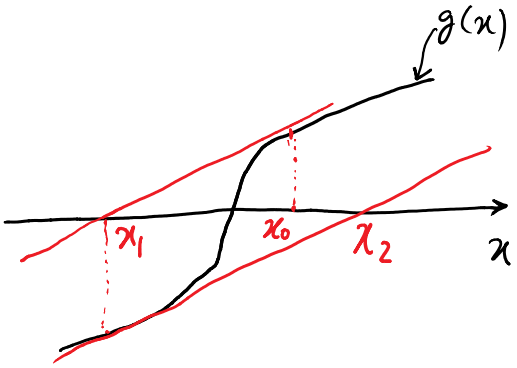
Bad things can happen with Newton's method

(Remark on notation: when you see f, the goal is to minimze; when you see g, the goal is to find a root.)

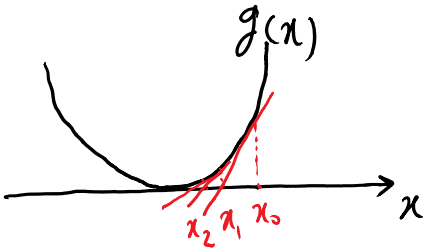
- Convergence is sensitive to our starting point



Moved in the wrong direction.



Iterations diverge.

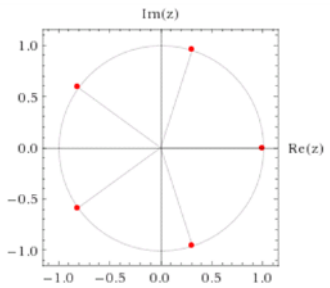


Convergence can get slow if $g'(x_*) = 0$.

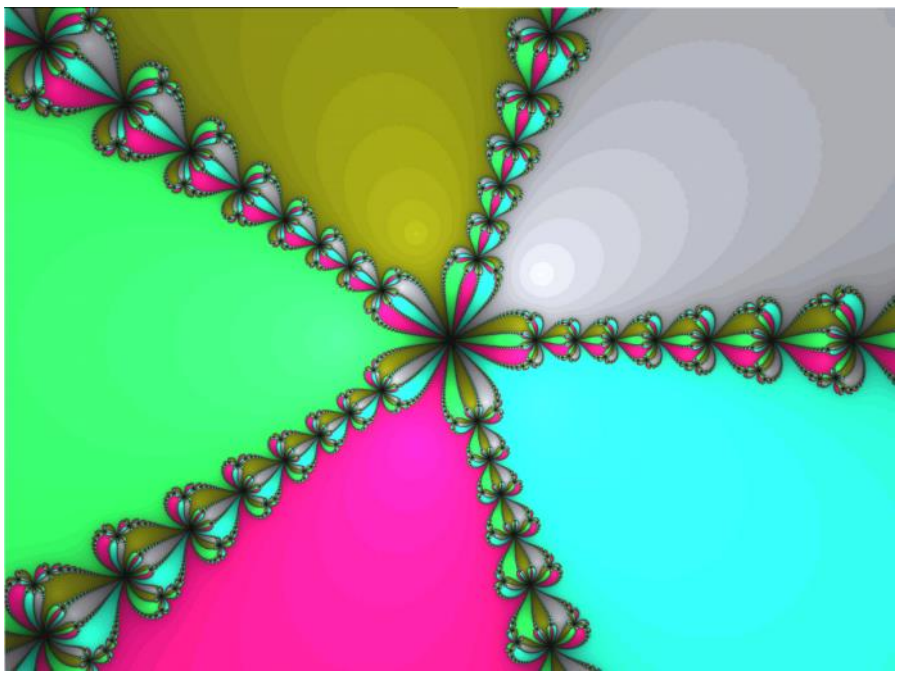
Bad things can happen with Newton's method

Convergence of Newton's method can be extremely sensitive on initial conditions. In the example below, we are trying to find the roots of a simple polynomial defined on the complex plane. There are arbitrarily close initial conditions whose Newton iterations converge to completely different roots.

$g(z) = z^5 - 1$



Roots of g



"Newton's fractal"

Image credit: N. Buroojy

Each point z of the complex plane is colored with one of five colors, depending on which root of the function g the Newton iterations converge to if we start them off from the initial point z.

The secant method

Recall the Newton iterations for minimizing a function:

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

The secant method is a very simple modification of the Newton's method, where we assume that we don't have access to f'', and instead approximate it with f' using finite differencing:

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

After substitution, we get:

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

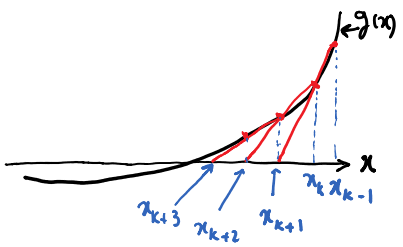
Or, equivalently:

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

- Note: we need two points to initialize this algorithm.
- Similarly, we can write a secant algorithm for root finding: $g(x) = 0$.
  - Simply replace f' with g.

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{g(x_k) - g(x_{k-1})} g(x_k) \qquad x_{k+1} = \frac{g(x_k)x_{k-1} - g(x_{k-1})x_k}{g(x_k) - g(x_{k-1})}$$

Geometric interpretation:



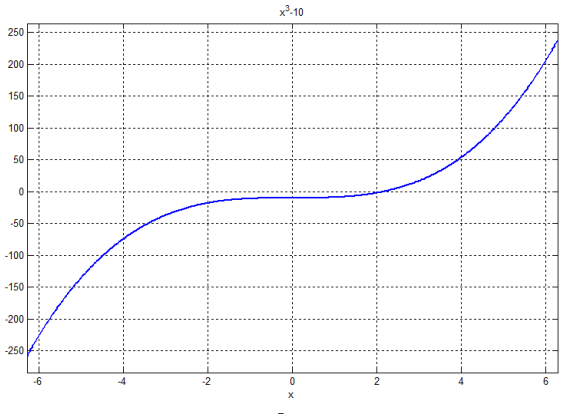
The secant method in action

Let's go back to our toy example:
Find a root of $g(x) = x^3 - 10$.

Answer is obviously:
>> 10^(1/3)

ans =

2.154434690031884



- How does the secant method compare with Newton and bisection?
- We run Newton with $x_0 = 12$, bisection wih $[a_0, b_0] = [0,12]$, secant with $(x_0, x_1) = (12,11)$.

Circled in red: correct significant digits

Bisection

Newton

k	x_k	a_k	$\frac{a_k+b_k}{2}$	b_k
1.000000000000000	12.000000000000000	0	6.000000000000000	12.000000000000000
2.000000000000000	8.023148148148149	0	3.000000000000000	6.000000000000000
3.000000000000000	5.400548660419450	0	1.500000000000000	3.000000000000000
4.000000000000000	3.714654390828676	1.500000000000000	2.250000000000000	3.000000000000000
5.000000000000000	2.718005659267038	1.500000000000000	1.875000000000000	2.250000000000000
6.000000000000000	2.263213061967483	1.875000000000000	2.062500000000000	2.250000000000000
7.000000000000000	2.159579216386407	2.062500000000000	2.156250000000000	2.250000000000000
8.000000000000000	2.154446935535738	2.062500000000000	2.109375000000000	2.156250000000000
9.000000000000000	2.154434690101485	2.109375000000000	2.132812500000000	2.156250000000000
10.000000000000000	2.154434690031884	2.132812500000000	2.144531250000000	2.156250000000000
11.000000000000000	2.154434690031884	2.144531250000000	2.150390625000000	2.156250000000000

Secant

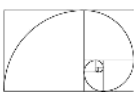
x_k
12.000000000000000
11.000000000000000
7.672544080604534
6.001247182309382
4.538549117833383
3.542882710716309
2.842693152077779
2.420226190958084
2.219611807907510
2.161719894878337
2.154650233385267
2.154435417201451

Secant with more iterations

x_k
12.000000000000000
11.000000000000000
7.672544080604534
6.001247182309382
4.538549117833383
3.542882710716309
2.842693152077779
2.420226190958084
2.219611807907510
2.161719894878337
2.154650233385267
2.154435417201451
2.154434690104630
2.154434690031884

# of correct digits:

1
1
2
4
6
10
16



Let's finish by mentioning that designing algorithms for root finding (in many dimensions) is a central area of research in computational mathematics. Many fundamental results of the area have only appeared in the past century.

Here is a summary of the state of affairs for those who are interested. We'll prove some of these statements later in the course. Some others are quite involved and well beyond the scope of this class.

Finding *real* roots in many dimensions

- Set of linear equations? Can be done efficiently. [poly-time]
- Set of quadratic equations? Can be done in finite time, but no efficient algorithm known (and unlikely to exist). [NP-hard]
- A single degree-4 equation? Same as above (why?). [NP-hard]

Finding *integer* roots in many dimensions

- Set of linear equations? Can be done efficiently. [poly-time]
- Set of quadratic equations? Not possible in finite time! [Undecidable]
- A single degree-4 equation? Same as above (why?).
- Google, e.g., Hilbert's 10th problem.

Finding *rational* roots in many dimensions

- Set of linear equations? Can be done efficiently. [poly-time]
- Set of quadratic equations? We actually currently don't know if it can be done in finite time!

What about solving systems of inequalities?

- Can only get harder
- But interestingly, the linear case over the reals can still be done efficiently (this is called linear programming!)
- Finding an integer solution to a system of linear inequalities is however NP-hard.

Notes:

- Chapter 7 of [CZ13] also covers root finding and line search in one dimension.

References:

- [CZ13] E.K.P. Chong and S.H. Zak. An Introduction to Optimization. Fourth edition. Wiley, 2013.

- [Tit13] A.L. Tits. Lecture notes on optimal control. University of Maryland, 2013.