Newton Methods

2026-05-06 · cheerful mango Haubentaucher

A Step in the Right direction.

The Why

In the previous chapter, we characterized numerical methods with conditioning, stability, consistency, and convergence. But our brute-force grid search is fundamentally limited: it explores the solution space blindly, requiring $O(N^d)$ evaluations for N grid points in d dimensions.

The key insight is deceptively simple: instead of asking "what is the value here?" at every grid point, we also ask "which way should I go next?". The function's slope, its derivative, tells us the direction towards the solution. This transforms search fundamentally.

Understanding these methods allows us to:

- Use local information to make sophisticated steps instead of brute-force searches
- Achieve faster and more optimal convergence.

In Machine Learning and Deep Learning, Newton's method simplified to first order is also known as gradient descent. One could argue that the entire field of deep learning is built on the idea of using local gradient information to navigate toward minima in a high-dimensional loss landscape, to train models that are optimized towards specific objectives.

Backpropagation is of course as vital to distribute gradient information through the layers of a neural network, but the core optimization step is still a form of gradient-based navigation.

Hands On Experience

For starters let's get an intuitive feel for the power of using local information. We compare grid search with Newton's method on the same sine function from Chapter 3, but now we're finding where it crosses zero: $\sin(x) = 0$ on the interval $[2.5, 4]$ (the solution is near $\pi \approx 3.14159$).

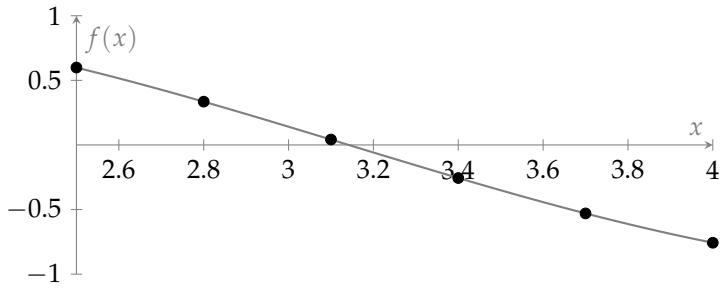

Figure 1: Continuous curve $f(x) = \sin(x)$ on $[2.5, 4]$, and discretized points with step size $h = 0.3$.

GRID SEARCH on $[2.5, 4]$ with step $h = 0.3$ evaluates blindly:

$$
\begin{aligned}
&f(2.5) \approx 0.599 \\
&f(2.8) \approx 0.335 \\
&f(3.1) \approx 0.042 \quad \leftarrow \text{closest to zero} \\
&f(3.4) \approx -0.256 \\
&f(3.7) \approx -0.530 \\
&f(4.0) \approx -0.757
\end{aligned}
$$

We found $x \approx 3.1$ with 6 evaluations, with an error $|3.1 - \pi| \approx 0.042$.

NEWTON'S METHOD has an adaptive approach to search. At every point we evaluate the function and its derivative, and use this local information to make a step towards the solution.

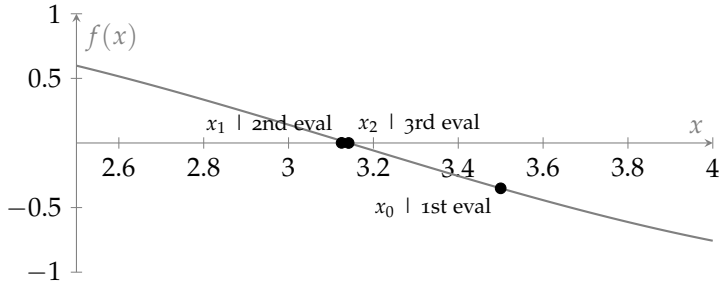

Figure 2: Newton's method iterations: starting from $x_0 = 3.5$, converging rapidly to $\pi \approx 3.14159$.

DERIVATIVE AS SOURCE OF LOCAL INFORMATION $f'(x) = \cos(x)$ gives the slope of the tangent at a given point. There are multiple

WE PAY FOR THIS SOPHISTICATION with more hyperparameters that we need to determine and more complex computations (derivative evaluation and division), but we gain in convergence speed.

ways to use this information to derive the direction and the size of the next step: We for sure want to move in the direction where the function decreases, then we could take a step with fixed size, or we could make the step size proportional to the derivative. Let's just take the next guess where the tangent crosses zero for now - which is the solution to the linear approximation of f at x_n.

Starting at $x_0 = 3.5$:

$$\begin{aligned} x_1 &= x_0 - \frac{\sin(x_0)}{\cos(x_0)} \\ &= 3.5 - \frac{-0.351}{-0.936} \\ &= 3.5 - 0.375 \\ &= 3.125 \\ x_2 &= 3.125 - \frac{\sin(3.125)}{\cos(3.125)} \\ &= 3.125 - \frac{0.0008}{-0.9999} \\ &\approx 3.14159 \\ x_3 &\approx 3.14159 \\ \text{and } \sin(3.14159) &\approx 0 \end{aligned}$$

After just 2 iterations, to be fair this means 4 function/derivative evaluations, we have $x \approx 3.14159$ with error $< 10^{-5}$. The derivative told us where to look way more efficiently than we have been used to.

The Learning Objectives of this chapter aim at providing you with the abilities to:

- Derive and apply Newton-Raphson iteration for root finding in 1D
- Derive and understnad the Taylor expansion and its role in Newton's method
- Analyze convergence rates and failure modes for Newton's method
- Understand and apply the Secant method when derivatives are unavailable

"Newton-Rhapson" because Isaac Newton came up with the general idea, while Joseph Raphson simplified the approach into a practical iterative method.

Newton-Raphson and Taylor Expansion

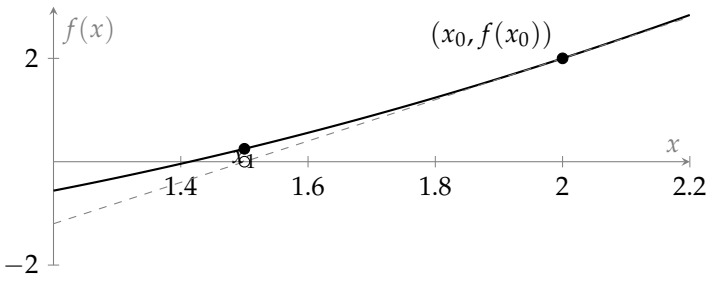

Figure 3: Newton's method geometric intuition: For $f(x) = x^2 - 2$, the tangent line at $(x_0, f(x_0))$ intersects the x-axis at x_1, our next approximation.

Above you can see the geometric intuition behind Newton's method for a single iteration. Let's formalize the approach of finding x^* such that $f(x^*) = 0$.

The linear approximation is effectively given as:

$$f(x) \approx f'(x_n) \cdot (x - x_n) \quad \text{for } x \text{ close to } x_n + f(x_n) \tag{1}$$

Note: This linear approximation is the first-order Taylor expansion of f around x_n. We'll return to Taylor expansions later when we study accuracy and approximation formally.

Finding the next guess: We want to find where this linear approximation crosses zero, because this is our best guess for where the actual function crosses zero. Setting the approximation to zero:

$$\begin{aligned} 0 &= f(x_n) + f'(x_n) \cdot (x_{n+1} - x_n) \\ f'(x_n) \cdot (x_{n+1} - x_n) &= -f(x_n) \\ x_{n+1} - x_n &= -\frac{f(x_n)}{f'(x_n)} \end{aligned}$$

This gives us the Newton-Raphson iteration formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2}$$

Algorithm 1: Newton-Raphson Method

Require: Initial guess x_0, function f, derivative f', tolerance ε

```
x ← x_0
while |f(x)| > ε do
    Compute derivative: d ← f'(x)
    if |d| < ε_machine then
        error "Derivative too small"
    end if
    Update: x ← x − f(x)/d
end whilereturn x
```

THIS ERROR occurs when the derivative $f'(x_k)$ approaches zero, which would cause division by zero or numerical instability in Newton's method.

Taylor Expansion

SO FAR WE DID JUST TRUST the linear approximation at x_n:

$$f(x) \approx f'(x_n)(x - x_n) + f(x_n), \tag{3}$$

to quickly find the next guess x_{n+1} and finally converge to the root, but is that trust well placed?

The answer lies in the Taylor expansion, a fundamental tool that tells us how well polynomials approximate smooth functions at a given point.

Let's derive it from first principles to understand why the linear approximation is a good choice for Newton's method, and how we can systematically improve it if needed. To illustrate let's approximate the sine function step by step.

SMOOTH: A function is smooth if it has derivatives of all orders, and is thus differentiable

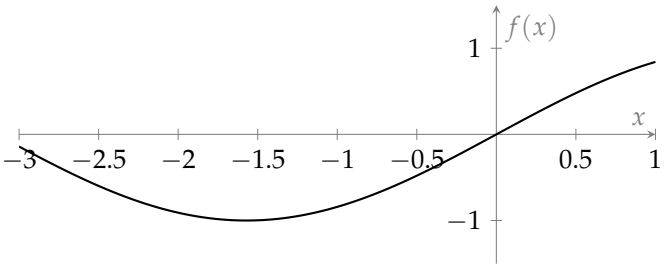

Figure 4: Continuous curve $f(x) = \sin(x)$ and its discretized version (black dots) on $[-3, 1]$ with step size $h = 1$.

0TH ORDER: MATCH THE FUNCTION VALUE. We want a polynomial $P(x)$ that approximates $f(x)$ near x_n. Start with matching at the point itself:

$$P(x_n) = f(x_n) \tag{4}$$

This comes very close to what grid search does: it only looks at the function value at discrete points, without any information about how the function behaves between those points.

1ST ORDER: MATCH THE FIRST DERIVATIVE. Near x_n, the function's slope matters. We want $P'(x_n) = f'(x_n)$.

Adding a linear term:

$$P(x) = f(x_n) + f'(x_n)(x - x_n) \tag{5}$$

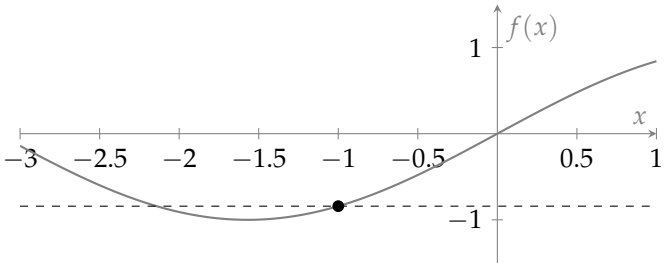

Figure 5: Continuous curve $f(x) = \sin(x)$ with constant approximation $P(x) = f(-1)$ (black dashed line) anchored at $x = -1$ (black dot).

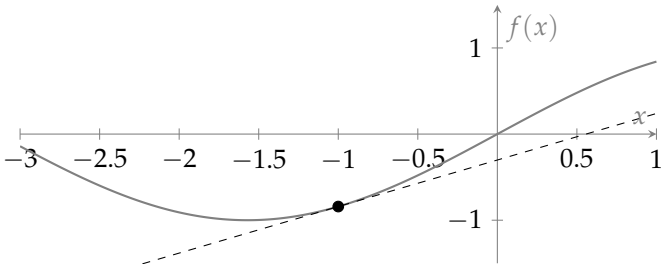

Figure 6: Continuous curve $f(x) = \sin(x)$ with linear approximation $P(x) = f(-1) + f'(-1)(x+1)$ (black dashed tangent line) anchored at $x = -1$ (black dot).

2ND ORDER: MATCH THE SECOND DERIVATIVE. The curvature captures how the slope changes. We want $P''(x_n) = f''(x_n)$. Adding a quadratic term:

$$P(x) = f(x_n) + f'(x_n)(x - x_n) + \frac{f''(x_n)}{2}(x - x_n)^2 \tag{6}$$

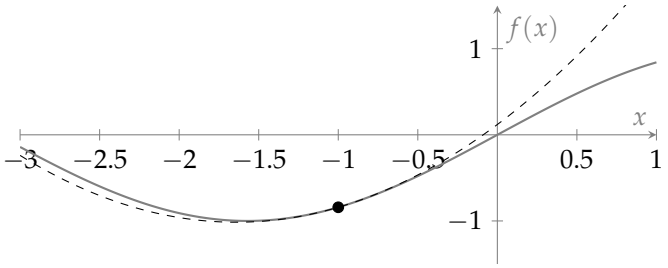

Figure 7: Continuous curve $f(x) = \sin(x)$ with quadratic approximation $P(x) = f(-1) + f'(-1)(x+1) + \frac{f''(-1)}{2}(x+1)^2$ (black dashed parabola) anchored at $x = -1$ (black dot).

WHY DIVIDE BY 2? When we differentiate $(x - x_n)^2$ twice, we get 2. So if we want $P''(x_n) = f''(x_n)$, we need to divide by 2 to cancel this factor.

NTH ORDER: GENERAL PATTERN.

THE TAYLOR EXPANSION of a function f around a point x_n is given by:

$$f(x) = f(x_n) + f'(x_n)(x - x_n) + \frac{f''(x_n)}{2!}(x - x_n)^2 + \frac{f'''(x_n)}{3!}(x - x_n)^3 + \cdots$$

Or more compactly:

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_n)}{k!}(x - x_n)^k \tag{7}$$

AROUND A POINT. Visually we can see that the approximations are anchored on a specific point x_n. The Taylor expansion is a local approximation.

WHY LINEAR IS ENOUGH FOR NEWTON?

We want to solve $f(x^*) = 0$, not compute $f(x)$ as best as we can. Improving an approximation of course comes with a cost: We need to compute higher derivatives and evaluate more complex polynomials. In numerical computations we always need to decide where to cut off the Taylor series. We do so at some finite order, and the linear term is the first non-constant term that gives us directional information.

DERIVING THE CONVERGENCE RATE. Recall from Chapter 3, that a method is convergent if our approximation reaches a specific stable limit. For Newton's method finding a root x^* where $f(x^*) = 0$, we want:

$$|x_n - x^*| \to 0 \quad \text{as } n \to \infty \tag{8}$$

Thanks to the Taylor expansion, we can quantify how fast the error decreases.

Let $e_n = x_n - x^*$ be the error at iteration n. Applying a 2nd Order Taylor expansion to f around the true root x^*:

$$f(x_n) = f(x^*) + f'(x^*)(x_n - x^*) + \frac{f''(\xi)}{2}(x_n - x^*)^2, \tag{9}$$

where ξ is some point between x_n and x^*. Since $f(x^*) = 0$, this simplifies to:

$$f(x_n) = f'(x^*) \cdot e_n + \frac{f''(\xi)}{2} e_n^2 \tag{10}$$

Where $e_n = x_n - x^*$ is the error at iteration n.

NOW APPLY NEWTON'S FORMULA:

$$\begin{aligned} e_{n+1} &= x_{n+1} - x^* \\ &= x_n - \frac{f(x_n)}{f'(x_n)} - x^* \\ &= (x_n - x^*) - \frac{f(x_n)}{f'(x_n)} \\ &= e_n - \frac{f(x_n)}{f'(x_n)} \end{aligned}$$

and substitute the simplified Taylor expansion for $f(x_n)$:

$$e_{n+1} = e_n - \frac{f'(x^*) \cdot e_n + \frac{f''(\xi)}{2} e_n^2}{f'(x_n)}. \tag{11}$$

FOR POINTS CLOSE TO x^*, we can make the assumption that $f'(x_n) \approx f'(x^*)$,

$$e_{n+1} \approx e_n - \frac{f'(x^*) \cdot e_n + \frac{f''(\xi)}{2} e_n^2}{f'(x^*)} = e_n - e_n - \frac{f''(\xi)}{2f'(x^*)} e_n^2 \tag{12}$$

NOTICE THE ASSUMPTION PLAY OUT: $10^{-1} \to 10^{-2} \to 10^{-3} \to 10^{-6} \to 10^{-12}$. Once we're close enough and our assumptions hold (after iteration 2), the error squares perfectly, demonstrating quadratic convergence.

which makes the first order error term cancel out, leaving us with:

$$e_{n+1} \approx -\frac{f''(\xi)}{2f'(x^*)} e_n^2 \tag{13}$$

In more abstract terms, we can write this as:

$$|e_{n+1}| \leq C \cdot |e_n|^2 \tag{14}$$

Which means that the error at the next iteration is approximately proportional to the square of the current error, where C is a constant that depends on the function's curvature and slope at the root.

This is quadratic convergence.

Let's come back to our introductory example of computing $\sqrt{2}$ with Newton starting from $x_0 = 2$:

n	x_n	$\lvert e_n \rvert$ **(error)**
0	2.000000000	5.86×10^{-1}
1	1.500000000	8.58×10^{-2}
2	1.416666667	2.45×10^{-3}
3	1.414215686	2.13×10^{-6}
4	1.414213562	4.5×10^{-12}

Table 1: Newton's method for computing $\sqrt{2}$. See how the error approximately squares each iteration.

Newton's method can fail in several ways, which we will endure for now, and for which we will find solutions later on. As a brain teaser, it is a nice exercise to visualize the failure modes in this plot:

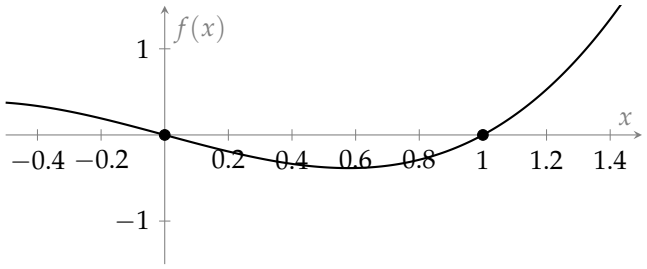

Figure 8: $f(x) = x^3 - x$ has multiple roots. Newton's method can fail in several ways: zero derivative (if $f'(x_n) = 0$, the tangent is horizontal and doesn't cross the axis), oscillation (especially for symmetrical functions Newton can cycle between points), and ambiguous starting point selection (x_0, heavily influences which root is found).

Now something we can not endure, is if we can't compute $f'(x)$.

This can happen because:

- The derivative is expensive to compute.
- The function is given as a black box (e.g., a simulation).
- The function isn't differentiable everywhere.

Secant Method

Here comes a clever way to approximate the derivative using only function evaluations, eliminating the need for an explicit derivative.

A POOR MAN'S DERIVATIVE. Approximate the derivative using two recent points:

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

Visually this is the slope of the secant line connecting the points $(x_{n-1}, f(x_{n-1}))$ and $(x_n, f(x_n))$ on the graph of f:

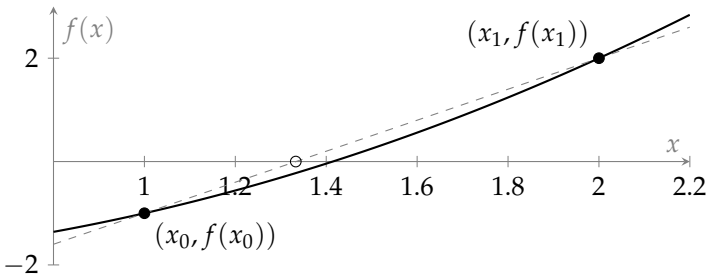

Figure 9: Secant method: the line through $(x_0, f(x_0))$ and $(x_1, f(x_1))$ gives the next approximation x_2.

NOTE, that this also means that we need two initial guesses x_0 and x_1 to start the iteration, instead of just one for Newton's method.

Substituting into Newton's formula, gives us the iterative Secant method:

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \tag{15}$$

AND HERE'S THE ALGORITHMIC VERSION FORMULATED AS A LOOP:

Algorithm 2: Secant Method

Require: Initial guesses x_0, x_1, function f, tolerance ε
1: **while** $|f(x_1)| > \varepsilon$ **do**
2: Compute secant slope: $s \leftarrow (f(x_1) - f(x_0))/(x_1 - x_0)$
3: Store previous: $x_{\text{old}} \leftarrow x_1$
4: Update: $x_1 \leftarrow x_1 - f(x_1)/s$
5: $x_0 \leftarrow x_{\text{old}}$
6: **end whilereturn** x_1

Examples & Exercises

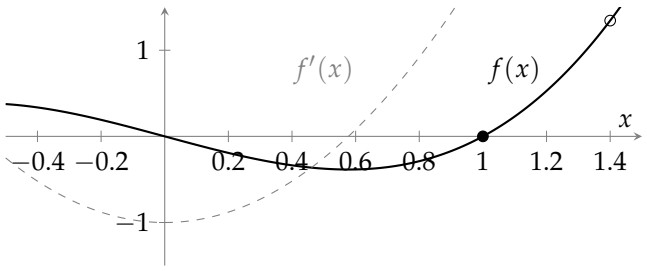

Figure 10: $f(x) = x^3 - x$ has multiple roots. We assume the root at $x = 1$ is the target. The gray dashed line shows the derivative $f'(x) = 3x^2 - 1$.

Code can be found at https://github.com/Quillstacks/lecturecode_numericalmethods.git.

Newton by hand. Find the root of $f(x) = x^3 - x = 0$ solving Newton's method geometrically, and then by hand. The roots are at $x = -1, 0, 1$. Let's find the root at $x = 1$ starting from $x_0 = 1.4$. The 1st order Taylor expansion around x_n gives us:

$$f(x) \approx f(x_n) + f'(x_n)(x - x_n),$$
$$f(x_n) \approx f(x) - f'(x_n)(x - x_n)$$

We have $f(x) = x^3 - x = 0$, with $f'(x) = 3x^2 - 1$, so we can rearrange to find the next guess, starting from $x_0 = 1.4$:

$$\begin{aligned}
x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)} \\
&= x_0 - \frac{x_0^3 - x_0}{3x_0^2 - 1} \\
&= 1.4 - \frac{2.744 - 1.4}{5.88 - 1} \\
&= 1.4 - \frac{1.344}{4.88} \approx 1.127 \\
x_2 &= 1.127 - \frac{1.127^3 - 1.127}{3(1.127)^2 - 1} \\
&\approx 1.127 - \frac{0.432}{2.808} \\
&\approx 0.976 \\
x_3 &= 0.976 - \frac{0.976^3 - 0.976}{3(0.976)^2 - 1} \\
&\approx 0.976 - \frac{-0.072}{1.857} \\
&\approx 1.014
\end{aligned}$$

The root at $x = 1$ is found quickly.

Secant by hand. Apply the Secant method to the same problem with $x_0 = 1.2$, $x_1 = 1.4$. Again first geometrically, then by hand.

$$
\begin{aligned}
x_2 &= x_1 - f(x_1) \cdot \frac{x_1 - x_0}{f(x_1) - f(x_0)} \\
&= 1.4 - (0.744) \cdot \frac{1.4 - 1.2}{0.744 - 0.128} \\
&= 1.4 - 0.744 \cdot \frac{0.2}{0.616} \\
&\approx 1.4 - 0.241 \\
&\approx 1.159 \\
x_3 &= 1.159 - f(1.159) \cdot \frac{1.159 - 1.4}{f(1.159) - f(1.4)} \\
&\approx 1.159 - (0.283) \cdot \frac{-0.241}{0.283 - 0.744} \\
&\approx 1.159 - 0.283 \cdot \frac{-0.241}{-0.461} \\
&\approx 1.159 - 0.283 \cdot 0.522 \\
&\approx 1.159 - 0.148 \\
&\approx 1.011
\end{aligned}
$$

Geometrical Failure Search. Find starting points x_0 for different failure modes: One where Newton's method fails due to zero derivative, one where it converges to a non-target root, and one where it oscillates between points.

Enter the machine. Let's have a look on the convergence and convergence rates of Grid-Search, Newton and Secant method in Python. Then continue to experiment with differentstarting points, and observe the failure modes in practice. Try to first find the starting points geometrically, then try finding them analytically, before finally moving over to verifying in code.

Self-Reflection and Recap

Self-Reflection Questions to guide your understanding:

- What is the main advantage of using Newton's method over Grid-Search?
- Why is a linear approximation sufficient for finding roots? What does the Taylor expansion tell us about this choice?
- Why is quadratic convergence so powerful? How many iterations would Newton need to go from error 10^{-1} to error 10^{-16}?

- In what situations would you prefer Secant over Newton? Why not in others?

Recap of Key Concepts:

- Newton-Raphson gives (limited) convergence guarantees near simple roots.
- Taylor expansion provides a systematic way to approximate functions locally.
- You saw how Taylor expansion justifies the linear approximation in Newton's method and explains its quadratic convergence and allows to estimate the error in each iteration.
- In situations where the derivative is difficult to compute, the Secant method approximates the derivative from two points and converges.

Teaser. How can we make sure that we find the global solution, and not just a local one?

Local methods find local solutions. Newton converges to whatever optimum is nearby, it is also prone to oscillate between points, or diverge if the derivative is zero or near zero. In the next chapter, we will reformulate the root finding into an optimization problem. And from there we'll explore how to go on and handle global optimization when the landscape has many valleys or is otherwise pathologically difficult.