Newton's Method

The Problem: Root Finding

Objective

Given a differentiable function $f(x)$, our goal is to find a value x^* such that:

$$f(x^*) = 0$$

- This value x^* is called a "root" or "zero" of the function.
- Examples:
  - Finding $\sqrt{2}$ is the same as finding the root of $f(x) = x^2 - 2$.
  - Solving complex equations that don't have a simple analytical solution.

The Core Idea: Tangent Lines

The strategy of Newton's method is to iteratively improve an initial guess.

1. **Start** with an initial guess, x_0.
2. **Approximate** the function $f(x)$ at x_0 with its tangent line.
3. **Find the root** of this tangent line. This root is our new, better guess, x_1.
4. **Repeat** the process from x_1 to get x_2, and so on.

Intuition

The tangent line is a good local approximation of the function. The root of the tangent should be close to the root of the function.

The Derivation

1. **Tangent line** at a point $(x_k, f(x_k))$:

$$y = f(x_k) + f'(x_k)(x - x_k)$$

2. **Find the root** of this line by setting $y = 0$ and $x = x_{k+1}$:

$$0 = f(x_k) + f'(x_k)(x_{k+1} - x_k)$$

3. **Solve for** x_{k+1} (assuming $f'(x_k) \neq 0$):

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

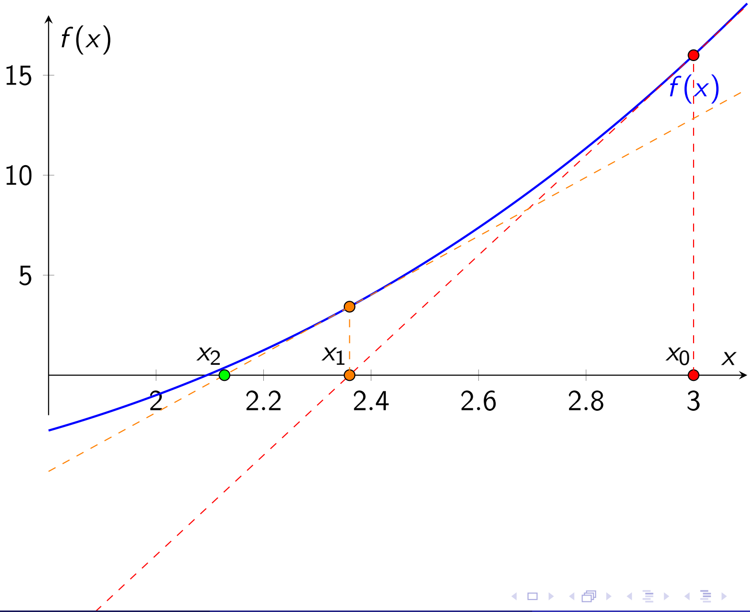
$$x_{k+1} - x_k = -\frac{f(x_k)}{f'(x_k)}$$

Newton's Method Recurrence

This gives us the iterative formula:

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

Visualizing the Method (Root Finding)



The Problem: Minimization

Objective

Given a twice-differentiable function $f(x)$, our goal is to find a value x^* that is a local minimum.

$$\min_x f(x)$$

- At a local minimum (or maximum), the slope of the function is zero.
- This is a fundamental problem in optimization, science, and engineering.

The Connection: Roots of the Derivative

First-Order Optimality Condition

A necessary condition for a point x^* to be a local minimum of a smooth function $f(x)$ is that the gradient (derivative in 1D) is zero:

$$f'(x^*) = 0$$

Connection!

Finding a minimum of $f(x)$ is the **same problem** as finding a root of its derivative, $f'(x)$!

- We can simply apply Newton's method to a new function, $g(x) = f'(x)$.

The Derivation (for Minimization)

1. **Goal:** Find x such that $g(x) = 0$, where $g(x) = f'(x)$.
2. **Apply Newton's** formula to $g(x)$:

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

3. **Substitute** $g(x) = f'(x)$ and $g'(x) = f''(x)$:

Newton's Method for Optimization

This gives the optimization recurrence relation:

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

Newton: quadratic approximation

Let us understand the meaning of the previous update rule.
Using Taylor expansion up to order 2 we have

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

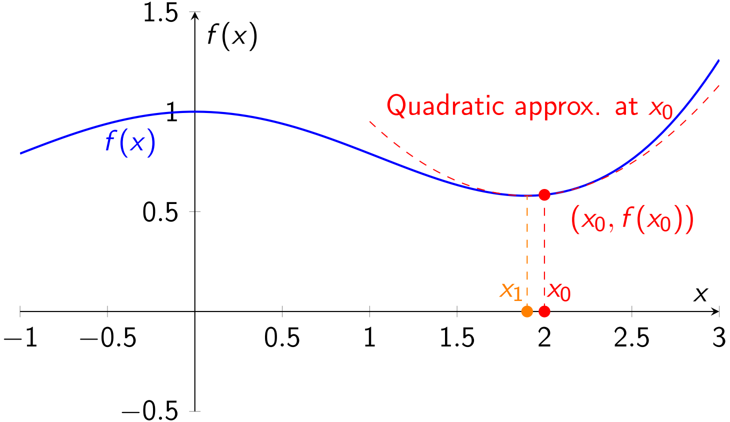
This is equivalent to fitting a quadratic function (a parabola) to $f(x)$ at x_k.
Now let us find the minimun of the parabola *i.e.*

$$q'(x) = 0 \Rightarrow q'(x) = f'(x_k) + f''(x_k)(x - x_k) = 0$$

Now setting $q'(x_{k+1}) = 0$ we have

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

Visualizing the Method (Minimization)



Summary and Caveats

Summary

- **Root Finding:** $x_{k+1} = x_k - f(x_k)/f'(x_k)$
- **Minimization:** $x_{k+1} = x_k - f'(x_k)/f''(x_k)$
- The method is powerful and converges very quickly (quadratically) when near a solution.

Caveats

Newton's method is not guaranteed to work:

- It requires calculating derivatives (f', and f'' for optimization), which can be difficult.
- It can be very sensitive to the initial guess x_0. A bad guess can send the iterations far away from the solution.
- It will fail if $f'(x_k) \approx 0$ (for root-finding) or $f''(x_k) \approx 0$ (for minimization), as this causes division by zero.

The n-Dimensional Problem I

Objective

Given a twice-differentiable scalar field $f : \mathbb{R}^n \rightarrow \mathbb{R}$, our goal is to find a vector $\boldsymbol{x}^* \in \mathbb{R}^n$ that is a local minimum.

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x})$$

The n-Dimensional Problem II

We must generalize our derivative concepts:

- **1D Derivative $f'(x)$:** becomes the **Gradient** (a vector).

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} \partial f / \partial x_1 \\ \vdots \\ \partial f / \partial x_n \end{pmatrix}$$

- **2nd Derivative $f''(x)$:** becomes the **Hessian** (an $n \times n$ matrix).

$$D^2 f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

The Connection: Root of the Gradient

The analogy to the 1D case is direct:

First-Order Optimality Condition

A necessary condition for a point $\boldsymbol{x}^*$ to be a local minimum of a smooth function $f(\boldsymbol{x})$ is that the gradient is zero:

$$\nabla f(\boldsymbol{x}^*) = 0$$

(where 0 is the $n \times 1$ zero vector)

Idea

Finding a minimum of $f(\boldsymbol{x})$ is the same problem as finding a root of its **gradient vector**, $\nabla f(\boldsymbol{x})$!

- We can apply Newton's method to the vector function $\boldsymbol{g}(\boldsymbol{x}) = \nabla f(\boldsymbol{x})$.

Deriving the n-D Method

1. **Goal:** Find $\boldsymbol{x}$ such that $\boldsymbol{g}(\boldsymbol{x}) = 0$, where $\boldsymbol{g}(\boldsymbol{x}) = \nabla f(\boldsymbol{x})$.

Deriving the n-D Method

1. **Goal:** Find $\boldsymbol{x}$ such that $\boldsymbol{g}(\boldsymbol{x}) = 0$, where $\boldsymbol{g}(\boldsymbol{x}) = \nabla f(\boldsymbol{x})$.
2. **Recall 1D Root-Finder:** $x_{k+1} = x_k - [g'(x_k)]^{-1} g(x_k)$

Deriving the n-D Method

1. **Goal:** Find $\boldsymbol{x}$ such that $\boldsymbol{g}(\boldsymbol{x}) = 0$, where $\boldsymbol{g}(\boldsymbol{x}) = \nabla f(\boldsymbol{x})$.
2. **Recall 1D Root-Finder:** $x_{k+1} = x_k - [g'(x_k)]^{-1} g(x_k)$
3. **Generalize to n-D:**
   - $g(x_k)$ becomes the vector $\boldsymbol{g}(\boldsymbol{x}_k) = \nabla f(\boldsymbol{x}_k)$.
   - $g'(x_k)$ (derivative of g) becomes the **Jacobian Matrix** of $\boldsymbol{g}$, which is $J_{\boldsymbol{g}}(\boldsymbol{x}_k)$.
   - The Jacobian of the gradient ∇f is the **Hessian** $D^2 f(\boldsymbol{x}_k)$.
   - $[g'(x_k)]^{-1}$ becomes the **Inverse Hessian** $[D^2 f(\boldsymbol{x}_k)]^{-1}$.

Deriving the n-D Method

1. **Goal:** Find $\boldsymbol{x}$ such that $\boldsymbol{g}(\boldsymbol{x}) = 0$, where $\boldsymbol{g}(\boldsymbol{x}) = \nabla f(\boldsymbol{x})$.
2. **Recall 1D Root-Finder:** $x_{k+1} = x_k - [g'(x_k)]^{-1} g(x_k)$
3. **Generalize to n-D:**
   - $g(x_k)$ becomes the vector $\boldsymbol{g}(\boldsymbol{x}_k) = \nabla f(\boldsymbol{x}_k)$.
   - $g'(x_k)$ (derivative of g) becomes the **Jacobian Matrix** of $\boldsymbol{g}$, which is $J_{\boldsymbol{g}}(\boldsymbol{x}_k)$.
   - The Jacobian of the gradient ∇f is the **Hessian** $D^2 f(\boldsymbol{x}_k)$.
   - $[g'(x_k)]^{-1}$ becomes the **Inverse Hessian** $[D^2 f(\boldsymbol{x}_k)]^{-1}$.
4. **Substitute:**

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - [J_{\boldsymbol{g}}(\boldsymbol{x}_k)]^{-1} \boldsymbol{g}(\boldsymbol{x}_k)$$

Newton's Method for n-D Optimization

Substituting the gradient and Hessian gives the final recurrence:

Newton's Method for Optimization (n-D)

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [D^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k)$$

- $\mathbf{x}_k$ is the current $n \times 1$ position vector.
- $\nabla f(\mathbf{x}_k)$ is the $n \times 1$ gradient vector.
- $D^2 f(\mathbf{x}_k)$ is the $n \times n$ Hessian matrix.
- $[D^2 f(\mathbf{x}_k)]^{-1}$ is the $n \times n$ inverse Hessian.

In Practice

We don't compute the inverse. We solve the linear system for the step $\Delta \mathbf{x}_k$:

$$[D^2 f(\mathbf{x}_k)] \Delta \mathbf{x}_k = -\nabla f(\mathbf{x}_k)$$

Then update: $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta \mathbf{x}_k$

The n-D Intuition: Quadratic Model

The intuition from 1D (fitting a parabola) holds perfectly.

- In n-D, we fit a quadratic model (a paraboloid) given by the 2nd-order Taylor expansion around $\boldsymbol{x}_k$:
  $q(\boldsymbol{x}) \approx f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{x} - \boldsymbol{x}_k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k)^T D^2 f(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k)$
- Newton's method simply **jumps to the exact minimum of this quadratic model**.
- Finding the minimum of $q(\boldsymbol{x})$ means finding where $\nabla q(\boldsymbol{x}) = 0$.
- $\nabla q(\boldsymbol{x}) = \nabla f(\boldsymbol{x}_k) + D^2 f(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k)$
- Setting $\nabla q(\boldsymbol{x}_{k+1}) = 0$ gives the exact same formula as before.

New Challenges in n-Dimensions

The 1D caveats (sensitivity to $\boldsymbol{x}_0$, etc.) still apply, but n-D adds new computational challenges:

Caveats for n-Dimensions

- **Cost of Derivatives:**
  - The gradient $\nabla f(\boldsymbol{x})$ has n components.
  - The Hessian $D^2 f(\boldsymbol{x})$ is an $n \times n$ matrix with $O(n^2)$ components to compute. This can be very expensive.
- **Cost of the Step (The Bottleneck):**
  - We must solve the $n \times n$ linear system: $[D^2 f(\boldsymbol{x}_k)]\Delta \boldsymbol{x}_k = -\nabla f(\boldsymbol{x}_k)$
  - Using standard methods (e.g., LU decomposition), this costs $O(n^3)$ operations. This is infeasible for large n.
- **Hessian Properties:**
  - The Hessian must be invertible.
  - For minimization, $D^2 f(\boldsymbol{x}_k)$ must be **positive definite**. If not, the step may point to a maximum or saddle point.

Summary: 1D vs n-D Minimization

The analogy is the key:

Concept	1D Version	n-D Generalization
Problem	$\min_x f(x)$	$\min_{\boldsymbol{x}} f(\boldsymbol{x})$
Condition	$f'(x) = 0$	$\nabla f(\boldsymbol{x}) = 0$
Model	Parabola	Paraboloid
2nd Derivative	$f''(x)$ (scalar)	$D^2 f(\boldsymbol{x})$ (matrix)
Update Step	$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$	$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - [D^2 f(\boldsymbol{x}_k)]^{-1} \nabla f(\boldsymbol{x}_k)$
Cost of Step	$O(1)$ (division)	$O(n^3)$ (solve system)

Next Steps

The $O(n^3)$ cost and the need for the full Hessian motivates **Quasi-Newton Methods** (like BFGS), which build an *approximation* of the Hessian using only gradient information.