



Convex Optimization for Machine Learning

with Mathematica Applications

CHAPTER 2

Single-Variable Optimization Without Constraints

M. M. Hammad

Department of Mathematics
Faculty of Science
Damanhour University
Egypt

Chapter 2

Single-Variable Optimization Without Constraints

2.1 Single-variable Functions (Results From Calculus)

Definition (derivative of a function): The derivative of a function f at a number a , denoted by $f'(a)$, is

$$f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h} = \lim_{h \rightarrow 0} \frac{f(x) - f(a)}{x - a} \quad (2.1)$$

if this limit exists.

Hence, the tangent line to $f(x)$ at $(a, f(a))$ is the line through $(a, f(a))$ whose slope is equal to $f'(a)$, see [figure 2.1](#).

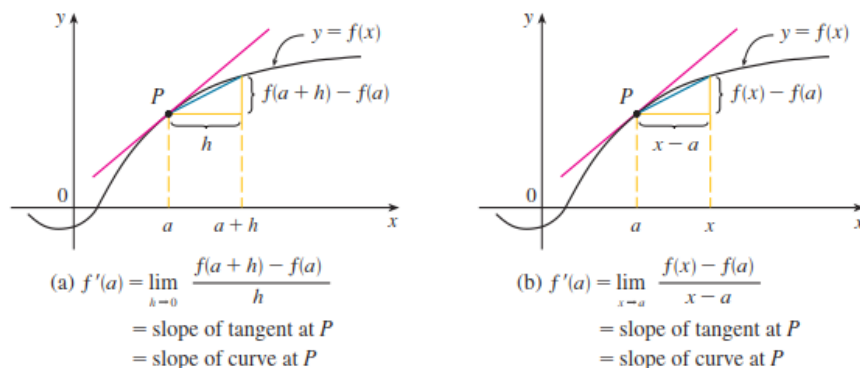


Figure 2.1. Geometric interpretation of the derivative.

Definitions (global and local minima) :

- A function f has a global maximum at c if $f(c) \geq f(x)$ for all x in D , where D is the domain of f . Similarly, f has a global minimum at c if $f(c) \leq f(x)$ for all x in D .
- The maximum and minimum values of f are called the extreme values of f .
- A function f has a local maximum at c if $f(c) \geq f(x)$ for all x in some open interval containing c .
- Similarly, f has a local minimum at c if $f(c) \leq f(x)$ when x is near c .

[Figure 2.2](#) shows the graph of a function f with global maximum at d and global minimum at a . Note that $(d, f(d))$ is the highest point on the graph and $(a, f(a))$ is the lowest point. If we consider only values of x near b [for instance, if we restrict our attention to the interval (a, c)], then $f(b)$ is the largest of those values of $f(x)$ and is called a local maximum value of f .

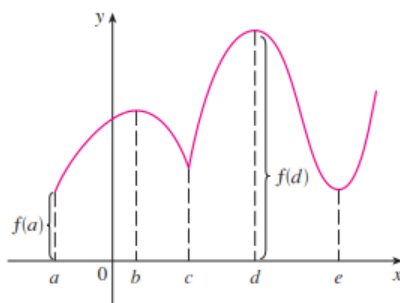


Figure 2.2. Minimum value $f(a)$, maximum value $f(d)$.

Definition (increasing and decreasing functions): A function f is called increasing on an interval I if

$$f(x_1) < f(x_2) \text{ whenever } x_1 < x_2 \text{ in } I$$

It is called decreasing on I if

$$f(x_1) > f(x_2) \text{ whenever } x_1 < x_2 \text{ in } I$$

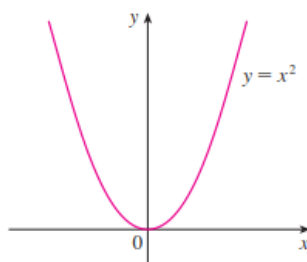


Figure 2.3. The function is decreasing on the interval $[-\infty, 0]$ and increasing on the interval $[0, \infty]$.

Increasing/Decreasing Test

(a) If $f'(x) > 0$ on an interval, then f is increasing on that interval.

(b) If $f'(x) < 0$ on an interval, then f is decreasing on that interval.

Proof:

Let x_1 and x_2 be any two numbers in the interval with $x_1 < x_2$. According to the definition of an increasing function, we have to show that $f(x_1) < f(x_2)$. Because we are given that $f'(x) > 0$, we know that f is differentiable on $[x_1, x_2]$. So, by the Mean Value Theorem there is a number c between x_1 and x_2 such that

$$f'(c) = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

Now $f'(c) > 0$ by assumption and because $x_2 - x_1 > 0$.

Thus, $f(x_1) < f(x_2)$. This shows that f is increasing.

Part (b) is proved similarly.

■

Definition (critical point): A critical point of a function f is a point in the domain of f such that $f'(c) = 0$ or $f'(c)$ does not exist.

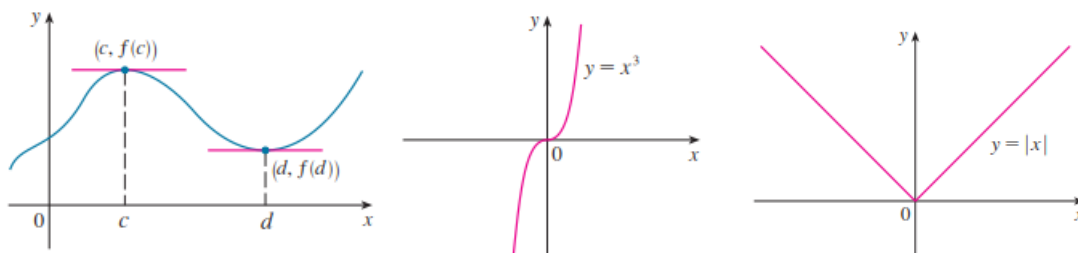


Figure 2.4. Critical points.

The First Derivative Test

Suppose that c is a critical number of a continuous function f .

- (a) If f' changes from positive to negative at c , then f has a local maximum at c .
- (b) If f' changes from negative to positive at c , then f has a local minimum at c .
- (c) If f' does not change sign at c (that is, f is positive on both sides of c or negative on both sides), then f has no local maximum or minimum at c .

See figure 2.3.

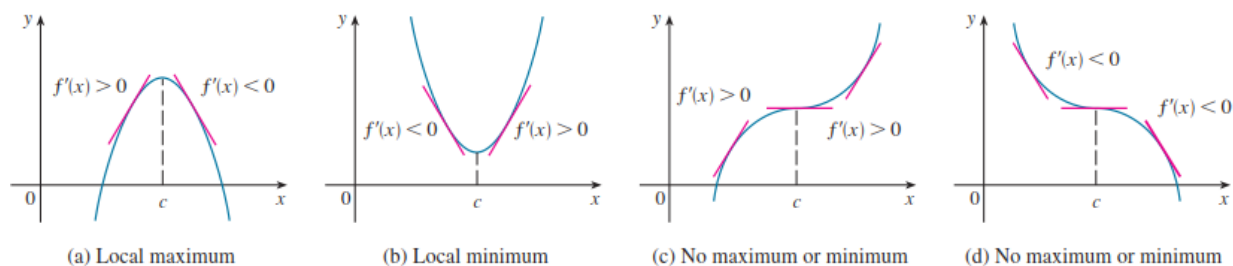


Figure 2.3. Local maximum and minimum.

Definition (concave upward and downward): A function (or its graph) is called concave upward on an interval I if f' is an increasing function on I . It is called concave downward on I if f' is decreasing on I .

Notice in Figure 2.4 that the slopes of the tangent lines increase from left to right on the interval (a, b) , so f is increasing and f is concave upward on (a, b) . [It can be proved that this is equivalent to saying that the graph of f lies above all of its tangent lines on (a, b) .] Similarly, the slopes of the tangent lines decrease from left to right on (b, c) , so f' is decreasing and f is concave downward on (b, c) .

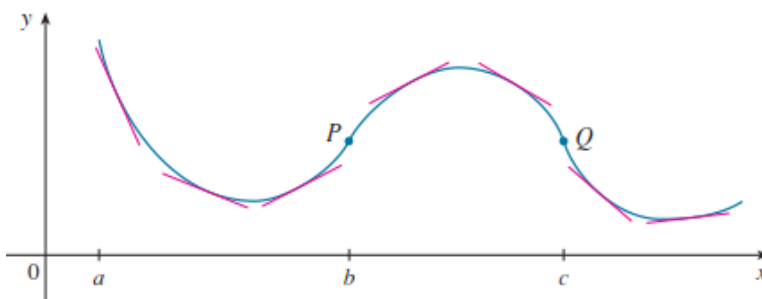


Figure 2.4. Concave upward and downward of a function.

Because $f'' = (f')'$, we know that if $f''(x)$ is positive, then f' is an increasing function and so f is concave upward. Similarly, if f'' is negative, then f' is decreasing and f is concave downward. Thus, we have the following test for concavity.

Concavity Test

- (a) If $f'' > 0$ for all x in I , then the graph of f is concave upward on I .
- (b) If $f'' < 0$ for all x in I , then the graph of f is concave downward on I .

The Second Derivative Test

Suppose f is continuous near c .

- (a) If $f'(c) = 0$ and $f''(c) > 0$, then f has a local minimum at c .
- (b) If $f'(c) = 0$ and $f''(c) < 0$, then f has a local maximum at c .

Example 2.1

Maximize: $z = xe^{-x^2}$.

Solution

Here

$$f'(x) = e^{-x^2} - 2x^2 e^{-x^2} = e^{-x^2}(1 - 2x^2)$$

which is defined for all x and which vanishes only at $x = \pm \frac{1}{\sqrt{2}}$. Since x is unrestricted, the values of the objective function at the stationary points (critical points),

$$f\left(\pm \frac{1}{\sqrt{2}}\right) = \pm \frac{1}{\sqrt{2}} e^{-\frac{1}{2}} = \pm 0.429$$

must be compared to the limiting values of $f(x)$ as $x \rightarrow \pm\infty$, which are both 0 in this case. Recording these results,

x	$x \rightarrow -\infty$	$-\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$	$x \rightarrow +\infty$
$f(x)$	0	-0.429	0.429	0

we see that a global maximum exists at $x^* = 1/\sqrt{2}$ and is $f(x^*) = 0.429$.

2.2 The optimization problem

We begin this section with optimization algorithms for single-variable and unconstrained functions. Since single-variable functions involve only one variable, the optimization procedures are simple and easier to understand. Moreover, these algorithms are repeatedly used as a subtask of many multi-variable optimization methods. Therefore, a clear understanding of these algorithms will help readers learn complex algorithms discussed in subsequent chapters.

The algorithms described in this chapter can be used to solve minimization problems of the following type:

$$\text{Minimize } f(x), \quad (2.2)$$

where $f(x)$ is the objective function and x is a real variable. The purpose of an optimization algorithm is to find a solution x , for which the function $f(x)$ is minimum. Two distinct types of algorithms are presented in this chapter: Direct search methods use only objective function values to locate the minimum point, and gradient-based methods use the first and/or the second-order derivatives of the objective function to locate the minimum point.

Even though the optimization methods described here are for minimization problems, they can also be used to solve a maximization problem by adopting the following procedure. In this case, an equivalent dual problem ($-f(x)$) is formulated and minimized. Hence, the algorithms described in this chapter can be directly used to solve a maximization problem. The approach is to use an iterative search algorithm, also called a line-search method. There are many of these algorithms, for examples

- **Bracketing Methods**
 - Exhaustive Search Method
 - Bounding Phase Method
- **Region-Elimination Methods**
 - Interval Halving Method
 - Fibonacci Search Method
 - Golden Section Search Method
- **Point-Estimation Method**
 - Successive Quadratic Estimation Method
- **Gradient-based Methods**
 - Newton-Raphson Method
 - Bisection Method
 - Secant Method
 - Cubic Search Method
- **Root-finding Using Optimization Techniques**

In an iterative algorithm, we start with an initial candidate solution $x^{(0)}$ and generate a sequence of iterates $x^{(1)}, x^{(2)}, \dots$. For each iteration $k = 0, 1, 2, \dots$, the next point $x^{(k+1)}$ depends on $x^{(k)}$ and the objective function f . The algorithm may use only the value of f at specific points, or perhaps its first derivative f' , or even its second derivative f'' . We will focus on the following algorithms:

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

2.3 Optimization Algorithms

Golden Section Search

The search methods we discuss in this section allow us to determine the minimizer of an objective function $f: \mathbb{R} \rightarrow \mathbb{R}$ over a closed interval, say $[a_0, b_0]$. The only property that we assume of the objective function f is that it is unimodal, which means that f has only one local minimizer.

The methods we discuss are based on evaluating the objective function at different points in the interval $[a_0, b_0]$. We choose these points in such a way that an approximation to the minimizer of f may be achieved in as few evaluations as possible. Our goal is to narrow the range progressively until the minimizer is "boxed in" with sufficient accuracy.

Consider a unimodal function f of one variable and the interval $[a_0, b_0]$. If we evaluate f at only one intermediate point of the interval, we cannot narrow the range within which we know the minimizer is located. We have to evaluate f at two intermediate points, as illustrated in figure 2.5. We choose the intermediate points in such a way that the reduction in the range is symmetric, in the sense that

$$a_1 - a_0 = b_0 - b_1 = \rho(b_0 - a_0), \quad (2.3)$$

where

$$\rho < \frac{1}{2}. \quad (2.4)$$

We then evaluate f at the intermediate points. If $f(a_1) < f(b_1)$, then the minimizer must lie in the range $[a_0, b_1]$. If, on the other hand, $f(a_1) \geq f(b_1)$, then the minimizer is located in the range $[a_1, b_0]$ (see figure 2.6).

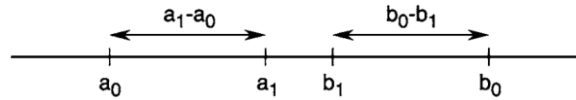


Figure 2.5. Evaluating the objective function at two intermediate points.

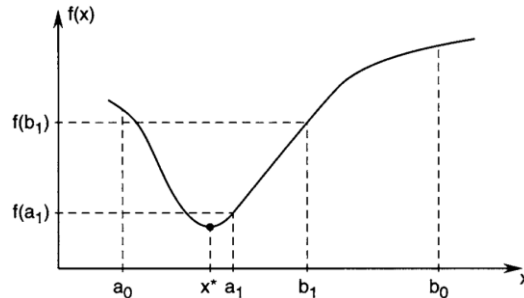


Figure 2.6. The case where $f(a_1) < f(b_1)$; the minimizer $x^* \in [a_0, b_1]$.

Starting with the reduced range of uncertainty, we can repeat the process and similarly find two new points, say a_2 and b_2 , using the same value of $\rho < \frac{1}{2}$ as before. However, we would like to minimize the number of objective function

$$\rho(b_1 - a_0) = b_1 - b_2. \quad (2.5)$$

$$\rho(b_1 - a_0) = b_1 - b_2. \quad (2.5)$$

Because $b_1 - a_0 = 1 - \rho$ and $b_1 - b_2 = 1 - 2\rho$, we have

$$\rho(1 - \rho) = 1 - 2\rho. \quad (2.6)$$

We write the quadratic equation above as

$$\rho^2 - 3\rho + 1 = 0. \quad (2.7)$$

The solutions are

$$\rho_1 = \frac{3 + \sqrt{5}}{2}, \quad \rho_2 = \frac{3 - \sqrt{5}}{2}. \quad (2.8)$$

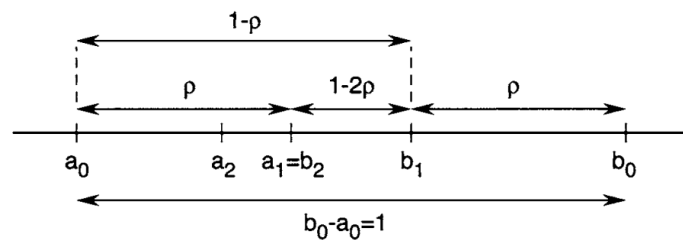


Figure 2.7. Finding value of ρ resulting in only one new evaluation of f .

Because we require that $\rho < \frac{1}{2}$, we take

$$\rho = \frac{3 - \sqrt{5}}{2} \approx 0.382. \quad (2.9)$$

Observe that

$$1 - \rho = \frac{\sqrt{5} - 1}{2} \quad (2.10)$$

and

$$\frac{\rho}{1-\rho} = \frac{3-\sqrt{5}}{\sqrt{5}-1} = \frac{\sqrt{5}-1}{2} = \frac{1-\rho}{1}, \quad (2.11)$$

that is,

$$\frac{\rho}{1-\rho} = \frac{1-\rho}{1}. \quad (2.12)$$

Thus, dividing a range in the ratio of ρ to $1 - \rho$ has the effect that the ratio of the shorter segment to the longer equals the ratio of the longer to the sum of the two. This rule was referred to as the golden section.

Using the golden section rule means that at every stage of the uncertainty range reduction (except the first), the objective function f need only be evaluated at one new point. The uncertainty range is reduced by the ratio $1 - \rho \approx 0.61803$ at every stage. Hence, N steps of reduction using the golden section method reduces the range by the factor

$$(1 - \rho)^N \approx (0.61803)^N. \quad (2.13)$$

Example 2.2

Suppose that we wish to use the golden section search method to find the value of x that minimizes

$$f(x) = x^4 - 14x^3 + 60x^2 - 70x$$

in the interval $[0,2]$. We wish to locate this value of x to within a range of 0.3.

Solution

After N stages the range $[0,2]$ is reduced by $(0.61803)^N$. So, we choose N so that

$$(0.61803)^N \leq 0.3/2.$$

Four stages of reduction will do; that is, $N = 4$.

Iteration 1.

We evaluate f at two intermediate points a_1 and b_1 . We have

$$\begin{aligned} a_1 &= a_0 + \rho(b_0 - a_0) = 0.7639, \\ b_1 &= b_0 - \rho(b_0 - a_0) \\ &= a_0 + b_0 - a_0 - \rho(b_0 - a_0) \\ &= a_0 + (1 - \rho)(b_0 - a_0) = 1.236, \end{aligned}$$

where $\rho = (3 - \sqrt{5})/2$. We compute $b_0 - b_1 = \rho(b_0 - a_0)$

$$\begin{aligned} f(a_1) &= -24.36, \\ f(b_1) &= -18.96. \end{aligned}$$

Thus, $f(a_1) < f(b_1)$, so the uncertainty interval is reduced to

$$[a_0, b_1] = [0, 1.236].$$

Iteration 2.

We choose b_2 to coincide with a_1 , and so f need only be evaluated at one new point,

$$a_2 = a_0 + \rho(b_1 - a_0) = 0.4721.$$

We have

$$\begin{aligned} f(a_2) &= -21.10, \\ f(b_2) &= f(a_1) = -24.36. \end{aligned}$$

Now, $f(b_2) < f(a_2)$, so the uncertainty interval is reduced to

$$[a_2, b_1] = [0.4721, 1.236].$$

Iteration 3.

We set $a_3 = b_2$ and compute b_3 :

$$b_3 = a_2 + (1 - \rho)(b_1 - a_2) = 0.9443.$$

We have

$$\begin{aligned} f(a_3) &= f(b_2) = -24.36, \\ f(b_3) &= -23.59. \end{aligned}$$

So $f(b_3) > f(a_3)$. Hence, the uncertainty interval is further reduced to

$$[a_2, b_3] = [0.4721, 0.9443].$$

Iteration 4.

We set $b_4 = a_3$ and

$$a_4 = a_2 + \rho(b_3 - a_2) = 0.6525.$$

We have

$$\begin{aligned} f(a_4) &= -23.84, \\ f(b_4) &= f(a_3) = -24.36. \end{aligned}$$

Hence, $f(a_4) > f(b_4)$. Thus, the value of x that minimizes f is located in the interval

$$[a_4, b_3] = [0.6525, 0.9443].$$

Note that $b_3 - a_4 = 0.292 < 0.3$.

Bisection Method

Again, we consider finding the minimizer of an objective function $f: \mathbb{R} \rightarrow \mathbb{R}$ over an interval $[a_0, b_0]$. As before, we assume that the objective function f is unimodal. Further, suppose that f is continuously differentiable and that we can use values of the derivative f' as a basis for reducing the uncertainty interval.

The bisection method is a simple algorithm for successively reducing the uncertainty interval based on evaluations of the derivative. To begin, let $x^{(0)} = (a_0 + b_0)/2$ be the midpoint of the initial uncertainty interval. Next, evaluate $f'(x^{(0)})$. If $f'(x^{(0)}) > 0$, then we deduce that the minimizer lies to the left of $x^{(0)}$. In other words, we reduce the

uncertainty interval to $[a_0, x^{(0)}]$. On the other hand, if $f'(x^{(0)}) < 0$, then we deduce that the minimizer lies to the right of $x^{(0)}$. In this case, we reduce the uncertainty interval to $[x^{(0)}, b_0]$. Finally, if $f'(x^{(0)}) = 0$, then we declare $x^{(0)}$ to be the minimizer and terminate our search.

With the new uncertainty interval computed, we repeat the process iteratively. At each iteration k , we compute the midpoint of the uncertainty interval. Call this point $x^{(k)}$.

Depending on the sign of $f'(x^{(k)})$ (assuming that it is nonzero), we reduce the uncertainty interval to the left or right of $x^{(k)}$. If at any iteration k we find that $f'(x^{(k)}) = 0$, then we declare $x^{(k)}$ to be the minimizer and terminate our search.

Two salient features distinguish the bisection method from the golden section. First, instead of using values of f , the bisection methods uses values of f' . Second, at each iteration, the length of the uncertainty interval is reduced by a factor of $1/2$. Hence, after N steps, the range is reduced by a factor of $(1/2)^N$. This factor is smaller than in the golden section.

Example 2.3

Use the bisection method to find the minimum of $f(x) = (x - 1)^4 + e^x$ on the interval $x \in [0, 3]$ for seven iterations.

Solution

You may find it useful to follow along in the following table.

k	a_k	b_k	$a_k + b_k/2$	$d_k = f'(a_k + b_k/2)$
1	0	3	1.5	4.98
2	0	1.5	0.75	2.05
3	0	0.75	0.375	0.478
4	0	0.375	0.1875	-0.939
5	0.1875	0.375	0.28125	-0.160
6	0.28125	0.375	0.328125	0.175
7	0.28125	0.328125	0.3046875	0.0116

At the end of the seventh iteration, our best guess of the optimum as the midpoint of the interval at this point: $x^* \approx 0.3047$. The true minimum point occurs at $x^* = 0.3031$; if we had chosen a smaller tolerance ϵ , the algorithm would have narrowed the interval further, with both ends converging towards this value.

Optimization means finding a maximum or minimum. In mathematical terms, optimization means finding where the derivative is zero. One can compare the goal of optimization in equation $\frac{df(x)}{dx} = 0$ with the goal of root-finding in equation $f(x) = 0$. The two equations are essentially the same, both setting a function equal to zero. This motivates the idea that we can use our existing single-equation root-finding tools to optimize nonlinear equations. If the function is simple, we can perform the differentiation by hand and obtain the functional form of the derivative. At that point we can apply any single-equation root-finding technique, such as the bisection method or the Newton-Raphson method, without modification, using as an input $f'(x)$ rather than $f(x)$.

If we either cannot or will not differentiate the function analytically, we can still use the framework of the bisection method or the Newton-Raphson method where we use the finite difference formula to provide the first derivative of the function. If we are using a technique like the bisection method, that is all we require as input. If we are using a technique like the Newton-Raphson method, which requires derivatives of the function, then we shall require the second derivative as well. The finite difference formulae for the second derivative is simple.

Later in this chapter, we provide subroutines for using the bisection and Newton-Raphson method for one-dimensional optimization. The only change in the bisection code necessary to convert it from a root-finding routine to an optimization routine is that, where we previously evaluated the function at the brackets, we now evaluate the first

derivative of the function at the brackets using a finite difference formula. The only changes in the Newton-Raphson with Numerical derivatives method to convert it from a root-finding routine to an optimization routine are that, (i) where we previously evaluated the function, we now evaluate the first derivative of the function using a finite difference formula and (ii) where we previously evaluated the first derivative of the function, we now evaluate the second derivative of the function using a finite difference formula.

Newton Method

Suppose again that we are confronted with the problem of minimizing a function f of a single real variable x . We assume now that at each measurement point $x^{(k)}$ we can determine $f(x^{(k)})$, $f'(x^{(k)})$, and $f''(x^{(k)})$.

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \dots \quad (2.14a)$$

$$x+h = x^{(k+1)}, \quad x = x^{(k)} \quad \text{and} \quad h = x^{(k+1)} - x^{(k)} \quad (2.14b)$$

$$f(x^{(k+1)}) = f(x^{(k)}) + f'(x^{(k)})(x^{(k+1)} - x^{(k)}) + \frac{1}{2}f''(x^{(k)})(x^{(k+1)} - x^{(k)})^2 + \dots \quad (2.14c)$$

Instead of minimizing f , we minimize its approximation. The first-order necessary condition for a minimizer yields

$$f'(x^{(k+1)}) \approx f'(x^{(k)}) + f''(x^{(k)})(x^{(k+1)} - x^{(k)}) = 0 \quad (2.15)$$

we obtain

$$x^{(k+1)} = x^{(k)} - \frac{f'(x^{(k)})}{f''(x^{(k)})}. \quad (2.16)$$

Example 2.4

Using Newton's method, find the minimizer of

$$f(x) = \frac{1}{2}x^2 - \sin x.$$

Solution

Suppose that the initial value is $x^{(0)} = 0.5$, and that the required accuracy is $\varepsilon = 10^{-5}$, in the sense that we stop when $|x^{(k+1)} - x^{(k)}| < \varepsilon$.

We compute

$$f'(x) = x - \cos x, \quad f''(x) = 1 + \sin x.$$

Hence,

$$\begin{aligned} x^{(1)} &= 0.5 - \frac{0.5 - \cos 0.5}{1 + \sin 0.5} \\ &= 0.5 - \frac{-0.3775}{1.479} \\ &= 0.7552. \end{aligned}$$

Proceeding in a similar manner, we obtain

$$\begin{aligned} x^{(2)} &= x^{(1)} - \frac{f'(x^{(1)})}{f''(x^{(1)})} = x^{(1)} - \frac{0.02710}{1.685} = 0.7391, \\ x^{(3)} &= x^{(2)} - \frac{f'(x^{(2)})}{f''(x^{(2)})} = x^{(2)} - \frac{9.461 \times 10^{-5}}{1.673} = 0.7390, \\ x^{(4)} &= x^{(3)} - \frac{f'(x^{(3)})}{f''(x^{(3)})} = x^{(3)} - \frac{1.17 \times 10^{-9}}{1.673} = 0.7390. \end{aligned}$$

Note that $|x^{(4)} - x^{(3)}| < \varepsilon = 10^{-5}$. Furthermore, $f'(x^{(4)}) = -8.6 \times 10^{-6} \approx 0$. Observe that $f''(x^{(4)}) = 1.673 > 0$, so we can assume that $x^* \approx x^{(4)}$ is a strict minimizer.

Newton's method works well if $f''(x) > 0$ everywhere (see [figure 2.8](#)). However, if $f''(x) < 0$ for some x , Newton's method may fail to converge to the minimizer (see [figure 2.9](#)).

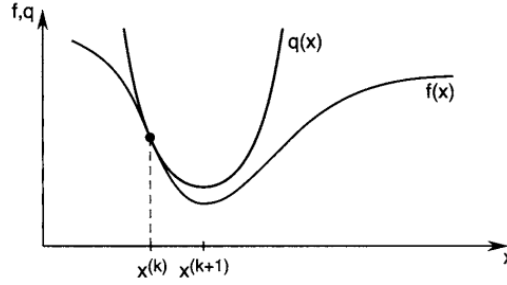


Figure 2.8. Newton's algorithm with $f''(x) > 0$, where
 $q(x^{(k+1)}) = f(x^{(k)}) + f'(x^{(k)})(x^{(k+1)} - x^{(k)}) + \frac{1}{2}f''(x^{(k)})(x^{(k+1)} - x^{(k)})^2$.

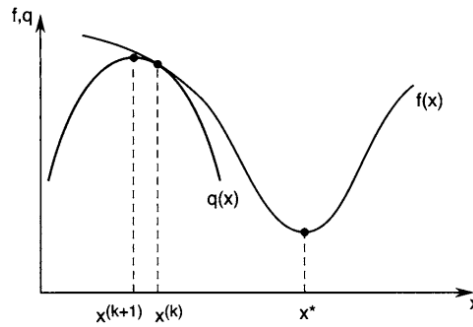


Figure 2.9. Newton's algorithm with $f''(x) < 0$.

Newton's method can also be viewed as a way to drive the first derivative of f to zero. Indeed, if we set $g(x) = f'(x)$, then we obtain a formula for iterative solution of the equation $g(x) = 0$:

$$x^{(k+1)} = x^{(k)} - \frac{g(x^{(k)})}{g'(x^{(k)})}. \quad (2.17)$$

In other words, we can use Newton's method for zero finding.

Example 2.5

We apply Newton's method to improve a first approximation, $x^{(0)} = 12$, to the root of the equation
 $g(x) = x^3 - 12.2x^2 + 7.45x + 42 = 0$.

Solution

We have $g'(x) = 3x^2 - 24.4x + 7.45$.

Performing two iterations yields

$$\begin{aligned} x^{(1)} &= 12 - \frac{102.6}{146.65} = 11.33, \\ x^{(2)} &= 11.33 - \frac{14.73}{116.11} = 11.21. \end{aligned}$$

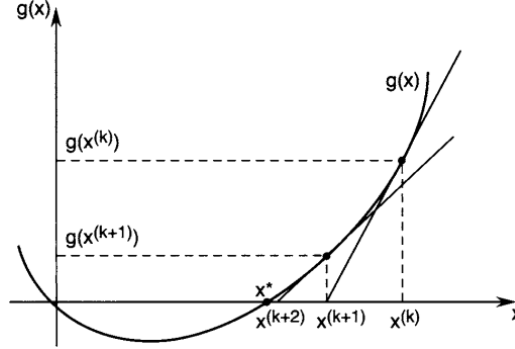


Figure 2.10. Newton's method of tangents.

Newton's method for solving equations of the form $g(x) = 0$ is also referred to as Newton's method of tangents. This name is easily justified if we look at a geometric interpretation of the method when applied to the solution of the equation $g(x) = 0$ (see Figure 2.10).

If we draw a tangent to $g(x)$ at the given point $x^{(k)}$, then the tangent line intersects the x -axis at the point $x^{(k+1)}$ which we expect to be closer to the root x^* of $g(x) = 0$. Note that the slope of $g(x)$ at $x^{(k)}$ is

$$g'(x^{(k)}) = \frac{g(x^{(k)})}{x^{(k)} - x^{(k+1)}}. \quad (2.18)$$

Hence,

$$x^{(k+1)} = x^{(k)} - \frac{g(x^{(k)})}{g'(x^{(k)})}. \quad (2.19)$$

Secant Method

Newton's method for minimizing f uses second derivatives of f :

$$x^{(k+1)} = x^{(k)} - \frac{f'(x^{(k)})}{f''(x^{(k)})}. \quad (2.20)$$

If the second derivative is not available, we may attempt to approximate it using first derivative information. In particular, we may approximate $f''(x^{(k)})$ above with

$$\frac{f'(x^{(k)}) - f'(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}. \quad (2.21)$$

Using the foregoing approximation of the second derivative, we obtain the algorithm

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - x^{(k-1)}}{f'(x^{(k)}) - f'(x^{(k-1)})} f'(x^{(k)}), \quad (2.22)$$

called the secant method. Note that the algorithm requires two initial points to start it, which we denote $x^{(-1)}$ and $x^{(0)}$. The secant algorithm can be represented in the following equivalent form:

$$x^{(k+1)} = \frac{f'(x^{(k)})x^{(k-1)} - f'(x^{(k-1)})x^{(k)}}{f'(x^{(k)}) - f'(x^{(k-1)})}. \quad (2.23)$$

Observe that, like Newton's method, the secant method does not directly involve values of $f(x^{(k)})$. Instead, it tries to drive the derivative f' to zero. In fact, as we did for Newton's method, we can interpret the secant method as an algorithm for solving equations of the form $g(x) = 0$. Specifically, the secant algorithm for finding a root of the equation $g(x) = 0$ takes the form

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - x^{(k-1)}}{g(x^{(k)}) - g(x^{(k-1)})} g(x^{(k)}), \quad (2.24)$$

or, equivalently,

$$x^{(k+1)} = \frac{g(x^{(k)})x^{(k-1)} - g(x^{(k-1)})x^{(k)}}{g(x^{(k)}) - g(x^{(k-1)})}. \quad (2.25)$$

The secant method for root finding is illustrated in Figure 2.11. Unlike Newton's method, which uses the slope of g to determine the next point, the secant method uses the "secant" between the $(k-1)$ th and k th points to determine the $(k+1)$ th point.

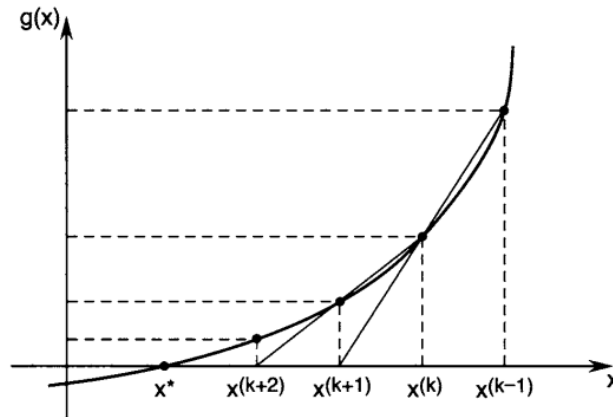


Figure 2.11. Secant method for root finding.

Example 2.6

We apply the secant method to find the root of the equation

$$g(x) = x^3 - 12.2x^2 + 7.45x + 42 = 0.$$

Solution

We perform two iterations, with starting points $x^{(-1)} = 13$ and $x^{(0)} = 12$. We obtain

$$x^{(1)} = 11.40,$$

$$x^{(2)} = 11.25.$$

2.4 Mathematica Built in Functions

<code>FindMinimum[f, x]</code>	searches for a local minimum in f , starting from an automatically selected point.
<code>FindMinimum[f, {x, x₀}]</code>	searches for a local minimum in f , starting from the point $x=x_0$.
<code>FindMinimum[f, {{x, x₀}, {y, y₀}, ...}]</code>	searches for a local minimum in a function of several variables.
<code>FindMinimum[{f, cons}, {{x, x₀}, {y, y₀}, ...}]</code>	searches for a local minimum subject to the constraints $cons$.
<code>FindMinimum[{f, cons}, {x, y, ...}]</code>	starts from a point within the region defined by the constraints.

Details and Options

- `FindMinimum` returns a list of the form $\{f_{\min}, \{x \rightarrow x_{\min}\}\}$, where f_{\min} is the minimum value of f found, and x_{\min} is the value of x for which it is found.
- If the starting point for a variable is given as a list, the values of the variable are taken to be lists with the same dimensions.
- The constraints $cons$ can contain equations, inequalities or logical combinations of these.
- The constraints $cons$ can be any logical combination of:

<code>lhs==rhs</code>	equations
<code>lhs>rhs</code> or <code>lhs>=rhs</code>	Inequalities
<code>{x, y, ...} ∈ reg</code>	region specification

- `FindMinimum[f, {x, x0, x1}]` searches for a local minimum in f using x_0 and x_1 as the first two values of x , avoiding the use of derivatives.
- `FindMinimum[f, {x, x0, xmin, xmax}]` searches for a local minimum, stopping the search if x ever gets outside the range x_{\min} to x_{\max} .
- Except when f and $cons$ are both linear, the results found by `FindMinimum` may correspond only to local, but not global, minima.
- By default, all variables are assumed to be real.
- For linear f and $cons$, $x \in \text{Integers}$ can be used to specify that a variable can take on only integer values.
- The following options can be given:

<code>AccuracyGoal</code>	Automatic	the accuracy sought
<code>EvaluationMonitor</code>	None	expression to evaluate whenever f is evaluated
<code>Gradient</code>	Automatic	the list of gradient components for f
<code>MaxIterations</code>	Automatic	maximum number of iterations to use
<code>Method</code>	Automatic	method to use
<code>PrecisionGoal</code>	Automatic	the precision sought
<code>StepMonitor</code>	None	expression to evaluate whenever a step is taken
<code>WorkingPrecision</code>	MachinePrecision	the precision used in internal computations

- The settings for `AccuracyGoal` and `PrecisionGoal` specify the number of digits to seek in both the value of the position of the minimum, and the value of the function at the minimum.
- `FindMinimum` continues until either of the goals specified by `AccuracyGoal` or `PrecisionGoal` is achieved.
- Possible settings for `Method` include "ConjugateGradient", "PrincipalAxis", "LevenbergMarquardt", "Newton", "QuasiNewton", "InteriorPoint", and "LinearProgramming", with the default being Automatic.
- The `FindMinimum` function in the Wolfram Language has five essentially different ways of choosing this model, controlled by the method option.

"Newton"	use the exact Hessian or a finite difference approximation if the symbolic derivative cannot be computed
"QuasiNewton"	use the quasi-Newton BFGS approximation to the Hessian built up by updates based on past steps
"LevenbergMarquardt"	a Gauss–Newton method for least-squares problems; the Hessian is approximated by $J^T J$, where J is the Jacobian of the residual function
"ConjugateGradient"	a nonlinear version of the conjugate gradient method for solving linear systems; a model Hessian is never formed explicitly
"PrincipalAxis"	works without using any derivatives, not even the gradient, by keeping values from past steps; it requires two starting conditions in each variable

- With Method->Automatic, the Wolfram Language uses the quasi-Newton method unless the problem is structurally a sum of squares, in which case the Levenberg–Marquardt variant of the Gauss–Newton method is used. When given two starting conditions in each variable, the principal axis method is used.

Example 2.7

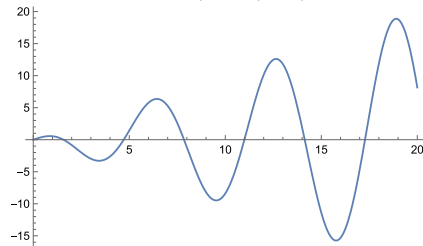
Input : (* With different starting points, you may get different local minima *)

: FindMinimum[x Cos[x], {x, 2}]

Output : {-3.28837, {x -> 3.42562}}

Input : Plot[x Cos[x], {x, 0, 20}]

Output



Input : FindMinimum[x Cos[x], {x, 5}]

Output : {-3.28837, {x -> 3.42562}}

Input : FindMinimum[x Cos[x], {x, 10}]

Output : {-9.47729, {x -> 9.52933}}

Input :

f=2 x^2-3 x+5;

Plot[f,{x,-10,10}]

FindMinimum[f,x,Method->"ConjugateGradient"]

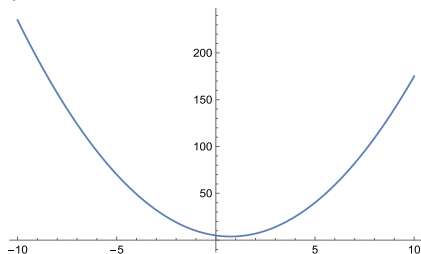
FindMinimum[f,x,Method->"PrincipalAxis"]

FindMinimum[f,x,Method->"Newton"]

FindMinimum[f,x,Method->"QuasiNewton"]

FindMinimum[f,x,Method->"LevenbergMarquardt"]

Output :



Output : {3.875, {x -> 0.75}}

Output : {3.875, {x -> 0.75}}

Output : {3.875, {x -> 0.75}}

Output : {3.875, {x -> 0.75}}

FindMinimum: The objective function for the method LevenbergMarquardt must be in a least-squares form: $\text{Sum}[f[i][x]^2, \{i, 1, n\}]$ or $\text{Sum}[w[i] f[i][x]^2, \{i, 1, n\}]$ with positive $w[i]$.

Output :FindMinimum[f, x, Method -> "LevenbergMarquardt"]

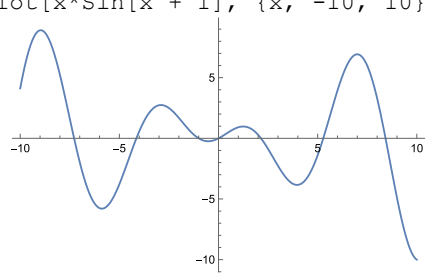
FindMinimumPlot[f, {x, x _{st} }]	plots the steps and the points at which the function f and any of its derivatives are evaluated in FindMinimum[f, {x, x _{st} }], superimposed on a plot of f versus x.
FindMinimumPlot[f, {{x, x _{st} }, {y, y _{st} }}]	plots the steps and the points at which the bivariate function f and any of its derivatives are evaluated, superimposed on a contour plot of f as a function of x and y.
FindMinimumPlot[f, range, property]	returns the specified property.

The FindMinimumPlot command is defined in the Optimization`UnconstrainedProblems` package loaded automatically by this notebook. It runs FindMinimum, keeps track of the function and gradient evaluations and steps taken during the search (using the EvaluationMonitor and StepMonitor options), and shows them superimposed on a plot of the function.

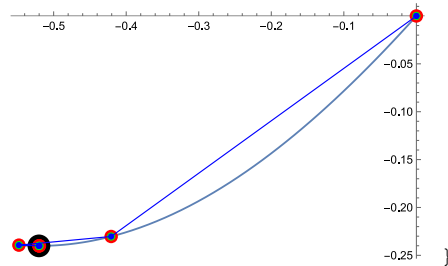
- Steps and evaluation points are color coded for easy detection as follows:
 - Steps are shown with blue lines and blue points.
 - Function evaluations are shown with green points.
 - Gradient evaluations are shown with red points.
 - Hessian evaluations are shown with cyan points.
 - Residual function evaluations are shown with yellow points.
 - Jacobian evaluations are shown with purple points.
 - The search termination is shown with a large black point.
- FindMinimumPlot and FindRootPlot return a list containing {result,summary,plot}, where:
 - result is the result of FindMinimum or FindRoot.
 - summary is a list of rules showing the number of steps and evaluations of the function and its derivatives.
 - plot is the graphics object shown.
- From the plot, it is clear that FindMinimum has found a local minimum point.
- With the setting PlotLegends->Automatic, FindMinimumPlot shows a legend for the evaluation points.

Example 2.8

```

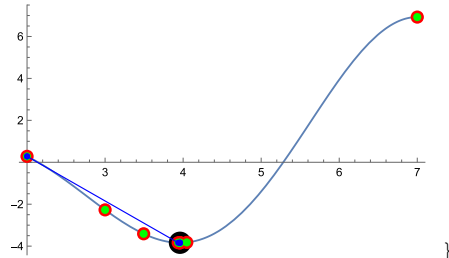
Input      : (*This loads a package that contains some utility functions.*)
            << Optimization`UnconstrainedProblems`
Input      : (*This shows a plot of the function x*Sin[x+1]. *)
            Plot[x*Sin[x + 1], {x, -10, 10}]
Output     
Input      :
            :
Input      : (* This shows the steps taken by FindMinimum for the function x*
            Sin[x+1] starting at x = 0. *)
            FindMinimumPlot[x*Sin[x + 1], {x, 0}]
Output     : {{-0.240125, {x -> -0.520269}}, {"Steps" -> 5, "Function" -> 6,
            "Gradient" -> 6},

```

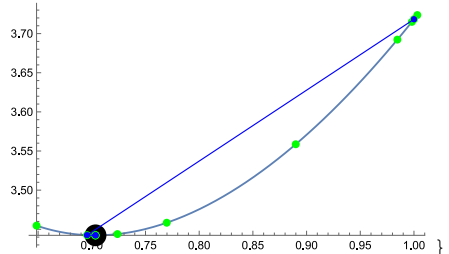
Input : (*This shows the steps taken by FindMinimum for the function $x \sin[x+1]$ starting at $x = 2$.)
FindMinimumPlot[$x \sin[x+1]$, { x , 2}]

Output : {{-3.83922, { $x \rightarrow 3.95976$ }}, {"Steps" $\rightarrow 4$, "Function" $\rightarrow 9$, "Gradient" $\rightarrow 9$ },



Input : (*Show the steps and function evaluations used in finding a local minimum of the function $\exp(x) + 1/x^*$)
FindMinimumPlot[$\exp[x] + 1/x$, { x , 1, 1.1}]

Output : {{3.44228, { $x \rightarrow 0.703467$ }}, {"Steps" $\rightarrow 6$, "Function" $\rightarrow 14$ },



Minimize[f, x]	minimizes f exactly with respect to x .
Minimize[f, {x, y, ...}]	minimizes f exactly with respect to x, y, \dots .
Minimize[{f, cons}, {x, y, ...}]	minimizes f exactly subject to the constraints cons .
Minimize[... , x \in rdom]	constrains x to be in the region or domain rdom .
Minimize[... , dom]	constrains variables to the domain dom , typically Reals or Integers.

Details and Options

- Minimize is also known as infimum.
- Minimize finds the global minimum of f subject to the constraints given.
- Minimize returns a list of the form $\{f_{\min}, \{x \rightarrow x_{\min}, y \rightarrow y_{\min}, \dots\}\}$.
- If f and cons are linear or polynomial, Minimize will always find a global minimum.
- The constraints cons can be any logical combination of:

lhs==rhs	equations
lhs>rhs, lhs \geq rhs, lhs<rhs, lhs \leq rhs	inequalities (LessEqual,...)
lhs>rhs, lhs \geq rhs, lhs<rhs, lhs \leq rhs	vector inequalities (VectorLessEqual,...)
Exists[...], ForAll[...]	quantified conditions
{x, y, ...} \in rdom	region or domain specification

- By default, all variables are assumed to be real.

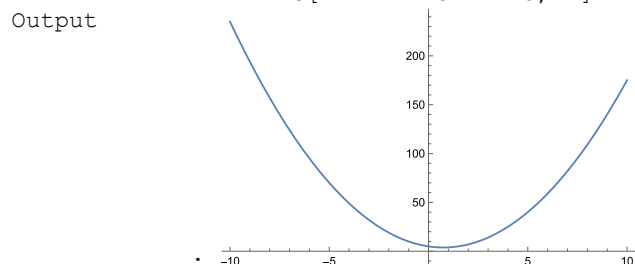
- `Minimize` will return exact results if given exact input. With approximate input, it automatically calls `NMinimize`.
- `Minimize` will return the following forms:

$\{f_{\min}, \{x \rightarrow x_{\min}, \dots\}\}$	finite minimum
$\{\infty, \{x \rightarrow \text{Indeterminate}, \dots\}\}$	infeasible, i.e. the constraint set is empty
$\{-\infty, \{x \rightarrow x_{\min}, \dots\}\}$	unbounded, i.e. the values of f can be arbitrarily small

- Even if the same minimum is achieved at several points, only one is returned.
- `Minimize[f, x, WorkingPrecision->n]` uses n digits of precision while computing a result.

Example 2.9

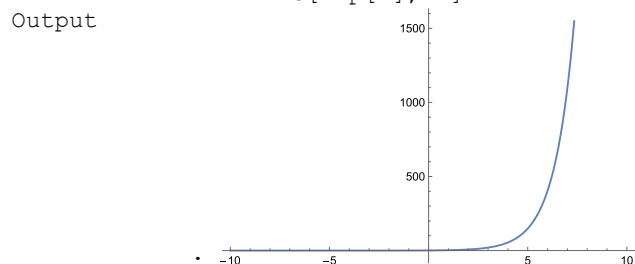
Input : `Plot[2 x^2 - 3 x + 5, {x, -10, 10}]`
`Minimize[2 x^2 - 3 x + 5, x]`



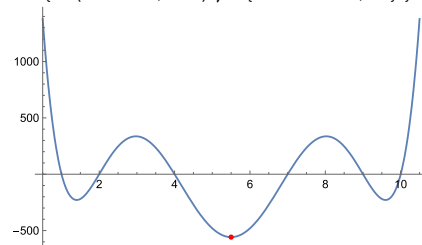
Input : `Minimize[a x^2 + b x + c, x]`
Output :
$$\left\{ \begin{array}{ll} \frac{c}{4a} & (b == 0 \wedge a == 0) \vee (b == 0 \wedge a > 0) \\ \frac{-b^2 + 4ac}{4a} & (b > 0 \wedge a > 0) \vee (b < 0 \wedge a > 0), \\ -\infty & \text{True} \end{array} \right.$$

$$\{x \rightarrow \begin{cases} -\frac{b}{2a} & (b > 0 \wedge a > 0) \vee (b < 0 \wedge a > 0) \\ 0 & (b == 0 \wedge a == 0) \vee (b == 0 \wedge a > 0) \\ \text{Indeterminate} & \text{True} \end{cases} \}$$

Input : `Plot[Exp[x], {x, -10, 10}]`
`Minimize[Exp[x], x]`



Input : `{0, {x -> -∞}}`
: `f = Expand[(x - 1) (x - 2) (x - 4) (x - 7) (x - 9) (x - 10)];`
`Minimize[f, x]`
`Plot[f, {x, 0.5, 10.5},`
`Epilog -> {PointSize[Medium], Red, Point[{x /. %[[2]], %[[1]]}]}`
Output : `{-(35721/64), {x -> 11/2}}`

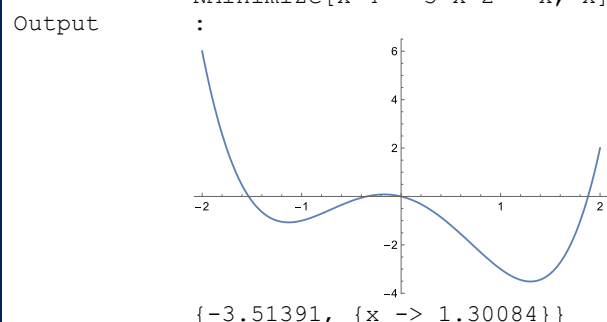


<code>NMinimize[f, x]</code>	minimizes f numerically with respect to x .
<code>NMinimize[f, {x, y, ...}]</code>	minimizes f numerically with respect to x, y, \dots .
<code>NMinimize[{f, cons}, {x, y, ...}]</code>	minimizes f numerically subject to the constraints cons .
<code>NMinimize[..., x ∈ rdom]</code>	constrains x to be in the region or domain rdom .

- `NMinimize` always attempts to find a global minimum of f subject to the constraints given.
- `NMinimize` returns a list of the form $\{f_{\min}, \{x \rightarrow x_{\min}, y \rightarrow y_{\min}, \dots\}\}$.
- If f and cons are linear or convex, the result given by `NMinimize` will be the global minimum, over both real and integer values; otherwise, the result may sometimes only be a local minimum.
- If `NMinimize` determines that the constraints cannot be satisfied, it returns $\{\text{Infinity}, \{x \rightarrow \text{Indeterminate}, \dots\}\}$.

Example 2.10

Input : `Plot[x^4 - 3 x^2 - x, {x, -2, 2}]`
`NMinimize[x^4 - 3 x^2 - x, x]`



<code>FindRoot[f, {x, x₀}]</code>	searches for a numerical root of f , starting from the point $x=x_0$.
<code>FindRoot[lhs==rhs, {x, x₀}]</code>	searches for a numerical solution to the equation $\text{lhs}=\text{rhs}$.
<code>FindRoot[{f₁, f₂, ...}, {{x, x₀}, {y, y₀}, ...}]</code>	searches for a simultaneous numerical root of all the f_i .
<code>FindRoot[{eqn₁, eqn₂, ...}, {{x, x₀}, {y, y₀}, ...}]</code>	searches for a numerical solution to the simultaneous equations eqn_i .

Details and Options

- If the starting point for a variable is given as a list, the values of the variable are taken to be lists with the same dimensions.
- `FindRoot` returns a list of replacements for x, y, \dots , in the same form as obtained from `Solve`.
- `FindRoot[lhs==rhs, {x, x0, x1}]` searches for a solution using x_0 and x_1 as the first two values of x , avoiding the use of derivatives.
- If you specify only one starting value of x , `FindRoot` searches for a solution using Newton methods. If you specify two starting values, `FindRoot` uses a variant of the secant method.
- If all equations and starting values are real, then `FindRoot` will search only for real roots. If any are complex, it will also search for complex roots.
- `FindRoot` continues until either of the goals specified by `AccuracyGoal` or `PrecisionGoal` is achieved.

Example 2.11

Input : `FindRoot[Sin[x] + Exp[x], {x, 0}]`
Output : `{x -> -0.588533}`
Input : `FindRoot[Cos[x] == x, {x, 0}]`
Output : `{x -> 0.739085}`

2.5 Resource Functions

BisectionMethodFindRoot: Determine the root of an equation using the bisection method

<code>ResourceFunction["BisectionMethodFindRoot"] [f, {x, xa, xb}, tol, n]</code>	searches for a numerical root of f between the points x_a and x_b using tol digits and up to n steps.
<code>ResourceFunction["BisectionMethodFindRoot"] [lhs==rhs, {x, xa, xb}, tol, n]</code>	searches for a numerical solution to the equation $lhs==rhs$.
<code>ResourceFunction["BisectionMethodFindRoot"] [f, {x, xa, xb}, tol, n, property]</code>	returns a property of the search for the root of f .

Details and Options

- BisectionMethodFindRoot supports two options for property:

"Solution"	return the root of f
"Steps"	return a table of steps taken to reach the root

- "PropertyAssociation" can be used to return an Association of the properties.
- BisectionMethodFindRoot terminates when the result is correct to the requested tolerance or the maximum number of steps has been taken, whichever comes first.

Example 2.12

Input

: ResourceFunction["BisectionMethodFindRoot"][x - Sqrt[30], {x, 5, 6}, 5, 100]

Output

: {x -> 5.4772}

Input

: ResourceFunction["BisectionMethodFindRoot"][Cos[x] == x, {x, 0, 1}, 3, 100, "Steps"]

Output

:

"steps"	"a"	"f[a]"	"b"	"f[b]"
1	1.00	-0.459698	0	1.
2	1.00	-0.459698	0.500	0.377583
3	0.750	-0.0183111	0.500	0.377583
4	0.750	-0.0183111	0.625	0.185963
5	0.750	-0.0183111	0.688	0.0853349
6	0.750	-0.0183111	0.719	0.0338794
7	0.750	-0.0183111	0.734	0.00787473
8	0.742	-0.00519571	0.734	0.00787473
9	0.742	-0.00519571	0.738	0.00134515
10	0.740	-0.00192387	0.738	0.00134515
11	0.739	0.	0.739	0.

NewtonsMethodFindRoot: Determine the root of an equation using Newton's method

<code>ResourceFunction["NewtonsMethodFindRoot"] [f, {x, x₀}, tol]</code>	searches for a numerical root of f starting at x_0 with digits equal to tol .
<code>ResourceFunction["NewtonsMethodFindRoot"] [lhs==rhs, {x, x₀}, tol]</code>	searches for a numerical solution to the equation $lhs==rhs$.
<code>ResourceFunction["NewtonsMethodFindRoot"] [f, {x, x₀}, tol, property]</code>	returns a property of the search for the root of f .

Example 2.13

Input	<code>ResourceFunction["NewtonsMethodFindRoot"][x - Sqrt[30], {x, 5}, 10]</code>
Output	<code>{x -> 5.477225575}</code>

Input : ResourceFunction["NewtonMethodFindRoot"][Cos[x] == x, {x, 1}, 8, "Steps"]

Output :

step	x	residual	derivative
0	1.	0.4596977	-1.84147098
1	0.750364	0.0189231	-1.68190495
2	0.739113	0.0000465	-1.67363254
3	0.739085	0.*10^-8	

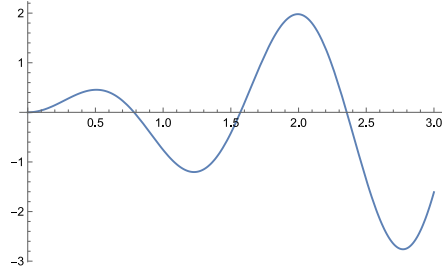
NewtonMethod: Approximate the root of a function using Newton's method

ResourceFunction["NewtonMethod"][f, {x, x₀}, n] returns the root approximation obtained by applying Newton's method at most n times to a differentiable function f with starting value x₀.

Example 2.14

Input : Plot[x Sin[4 x], {x, 0, 3}]

Output :



Input : ResourceFunction["NewtonMethod"][x Sin[4 x], {x, 1.5}]

Output : {x -> 1.5708}

NewtonMethodPlot: Plot the function together with a graphical display of the Newton iterations approximating its root

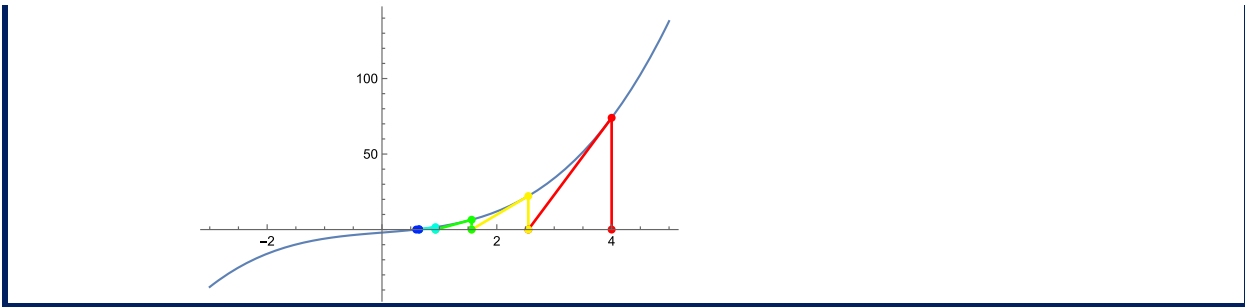
ResourceFunction["NewtonMethodPlot"][f, {x, x_{min}, x_{max}}, pt] returns a plot of f from x=x_{min} to x=x_{max}, together with illustrations representing the iterations of Newton's root-finding method, starting at x=pt.

Example 2.15

Input : ResourceFunction["NewtonMethodPlot"][x^3 - 2 + 3 x, {x, -3, 5}, 4]

Output :

4.
2.54902
1.56161
0.932191
0.645648
0.597163



SecantMethodFindRoot: Determine the root of an equation using the secant method

ResourceFunction["SecantMethodFindRoot"] [f,{x,x ₀ ,x ₁ },prec]	searches for a numerical root of f starting at x ₀ with digits equal to prec.
ResourceFunction["SecantMethodFindRoot"] [lhs==rhs,{x,x ₀ ,x ₁ },prec]	searches for a numerical solution to the equation lhs==rhs.
ResourceFunction["SecantMethodFindRoot"] [f,{x,x ₀ ,x ₁ },prec,property]	returns a property of the search for the root of f.

Example 2.16

Input	: ResourceFunction["SecantMethodFindRoot"][x^2 - 30, {x, 5, 6}, 10]
Output	: {x -> 5.477225575}

NumericalMethodFindRoot: Find the root of an equation or number using a specified numerical method

ResourceFunction["NumericalMethodFindRoot"] [f,x,method]	searches for a numerical root of f as a function of x, using the specified method.
ResourceFunction["NumericalMethodFindRoot"] [f,{x,x ₀ }, method]	searches for a numerical root of f, starting from the point x= x ₀ .
ResourceFunction["NumericalMethodFindRoot"] [f,{x, x ₀ },method,property]	returns the specified property for the numerical search.

- ResourceFunction["NumericalMethodFindRoot"] supports "Bisection", "Newton" and "Secant" methods.

Example 2.17

```

Input      : ResourceFunction["NumericalMethodFindRoot"][x Cos[x], x, "Newton"]
Output     : {x -> -7.85398}
Input      : ResourceFunction["NumericalMethodFindRoot"][ x Cos[x], {x, 2},
"Newton"]
Output     : {x -> 1.5708}
Input      : grid = ResourceFunction["NumericalMethodFindRoot"][x Cos[x], {x, 2},
"Newton", "Steps"]
Output     :

```

"steps"	"x"	"f[x]"
1	2.	-0.832294
2	1.62757	-0.0923469
3	1.57265	-0.00291944
4	1.5708	-3.43469*10^-6
5	1.5708	-4.78107*10^-12
6	1.5708	9.61835*10^-17
7	1.5708	9.61835*10^-17

FindRootPlot: Visualize the function evaluations done by FindRoot

ResourceFunction["FindRootPlot"]
[f, {x, x_{st}}]

plots the steps and the points at which the function f and any of its derivatives are evaluated in `FindRoot[f, {x, xst}]`, superimposed on a plot of f versus x .

ResourceFunction["FindRootPlot"]
[{f₁, f₂}, {{x, x_{st}}, {y, y_{st}}}]

plots the steps and the points at which the pair of functions and their derivatives are evaluated, superimposed on a contour plot of the merit function.

ResourceFunction["FindRootPlot"]
[f, range, property]

returns the specified property.

Example 2.18

Input : ResourceFunction["FindRootPlot"][Cos[x], {x, 3, 6}, PlotRange -> All]
Output :

