Applied Machine Learning for Educational Data Science

Warm-up II | Optimization

true

05/26/2021

Contents

[Updated: Thu, Jun 03, 2021 - 19:11:47]

This is a crash course on Optimization for students in social science. This document does not replace a course on Calculus or Optimization, and only serves the purpose of developing some vocabulary you will likely encounter in any Machine Learning course or textbook. If you had taken Calculus before, it will refresh your memory.

Functions

A function is a rule to assign an input number from a defined domain to an output number in another defined domain. A domain is a set of all possible input and output values. For instance,

$$f : \mathbb{R} \to \mathbb{R},$$

indicates that f takes values from the real numbers as an input and then produces an output in the real numbers. When the set of domain is not explicitly defined, we typically assume that domain of inputs is the largest set of possible numbers the rule can apply.

For instance, the following is a function:

$$f(x) = 5x^2 - 2x - 12$$

Below are the output of this function for a series of input values.

x	$f(x)$
-3	39
-2	12
-1	-5
0	-12
1	-9
2	4
3	27

Constant functions

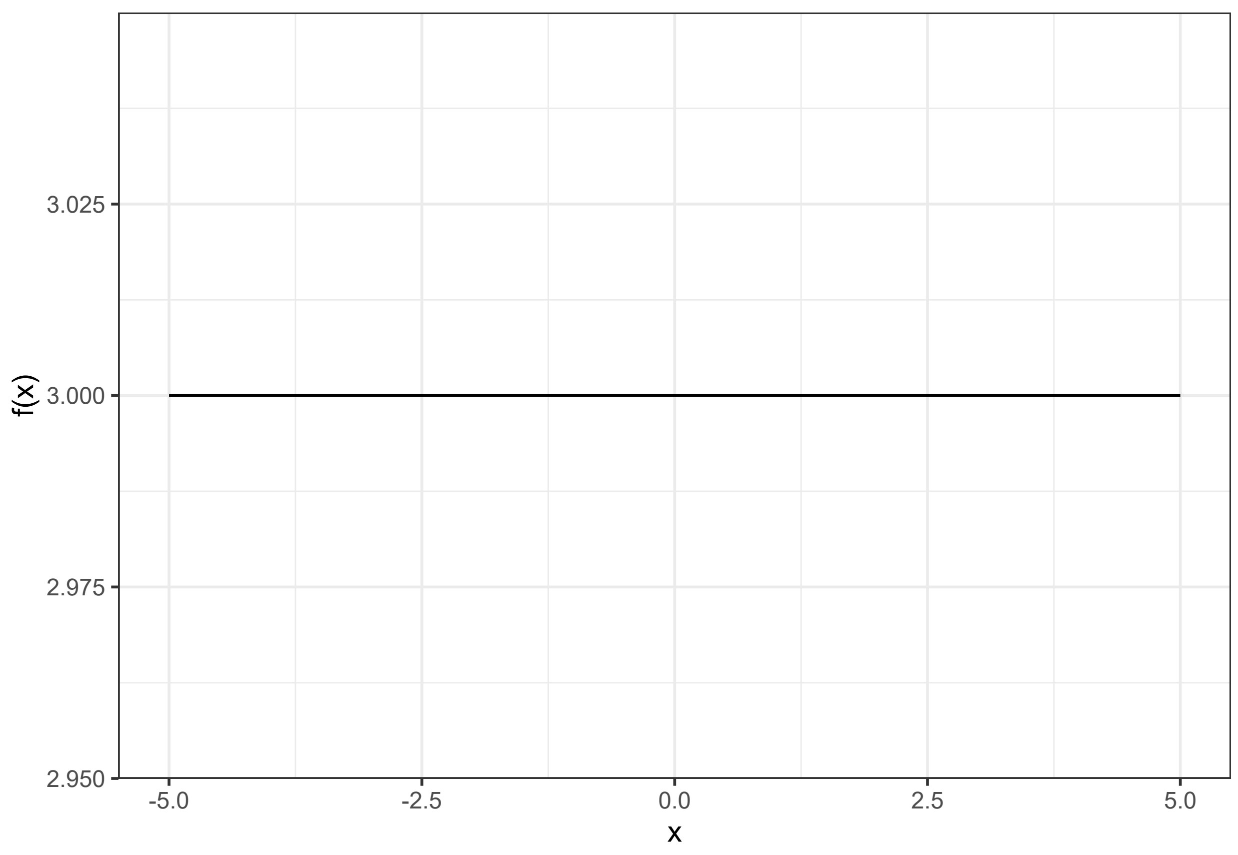
Some functions are constant. No matter what the input is they always produce the same output.

$$f(x) = a_0$$

,

where a_0 is a constant number.

```r
f <- function(x) {3}

ggplot() +
  geom_function(fun = f) +
  xlim(-5,5)+
  xlab('x')+
  ylab('f(x)')+
  theme_bw()
```

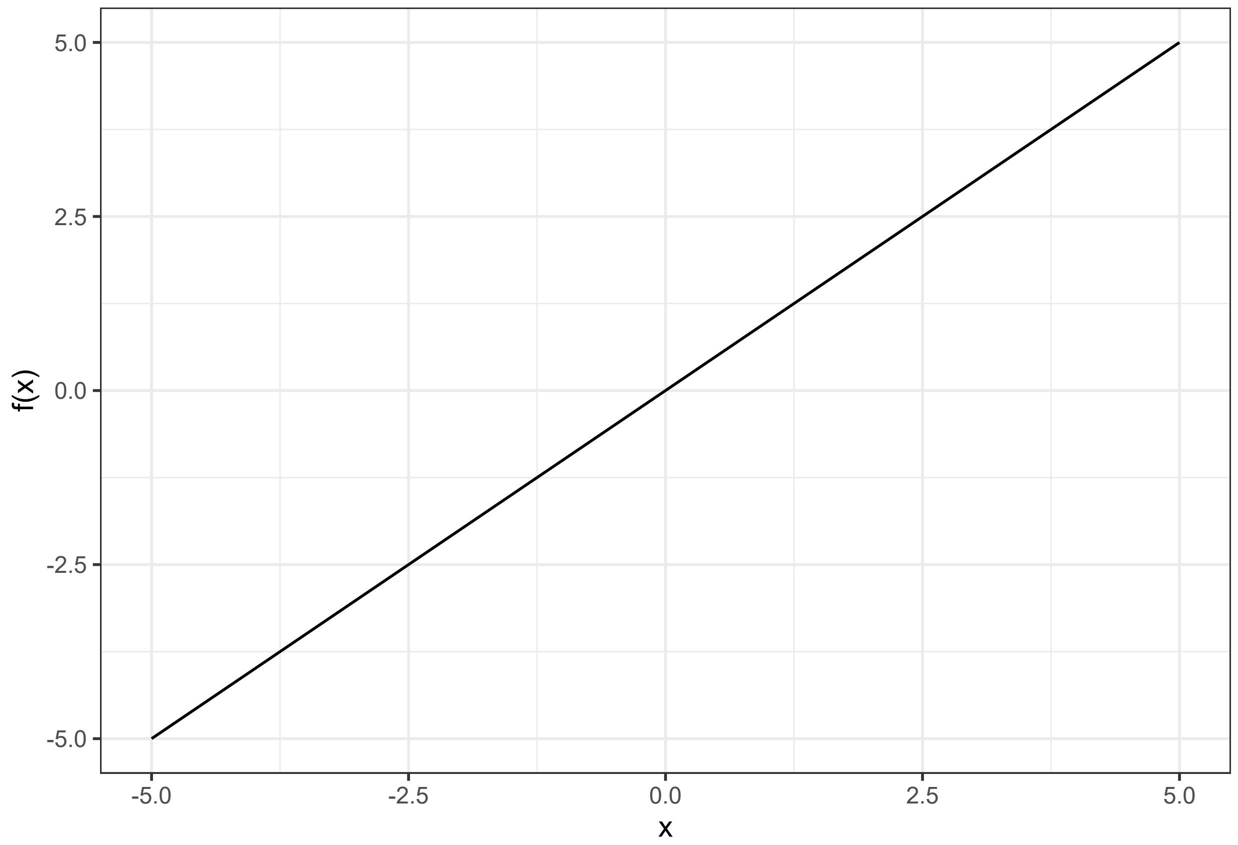


Identity functions

Some functions always produces the output identical to the input.

$$f(x) = x$$

```
f <- function(x) {x}

ggplot() +
  geom_function(fun = f) +
  xlim(-5,5)+
  xlab('x')+
  ylab('f(x)')+
  theme_bw()
```



Inverse functions

Some functions are inverse of other functions. Consider the following function:

$$f(x) = 3x + 5$$

For an input value of 2, this function returns a value of 11. If we want to find an inverse of this function, we should come up with something that takes the input value of 11 and returns a value of 2. Following is the inverse of this function:

$$f^{-1}(x) = \frac{x - 5}{3}$$

Polynomials

Polynomial functions are in the form of

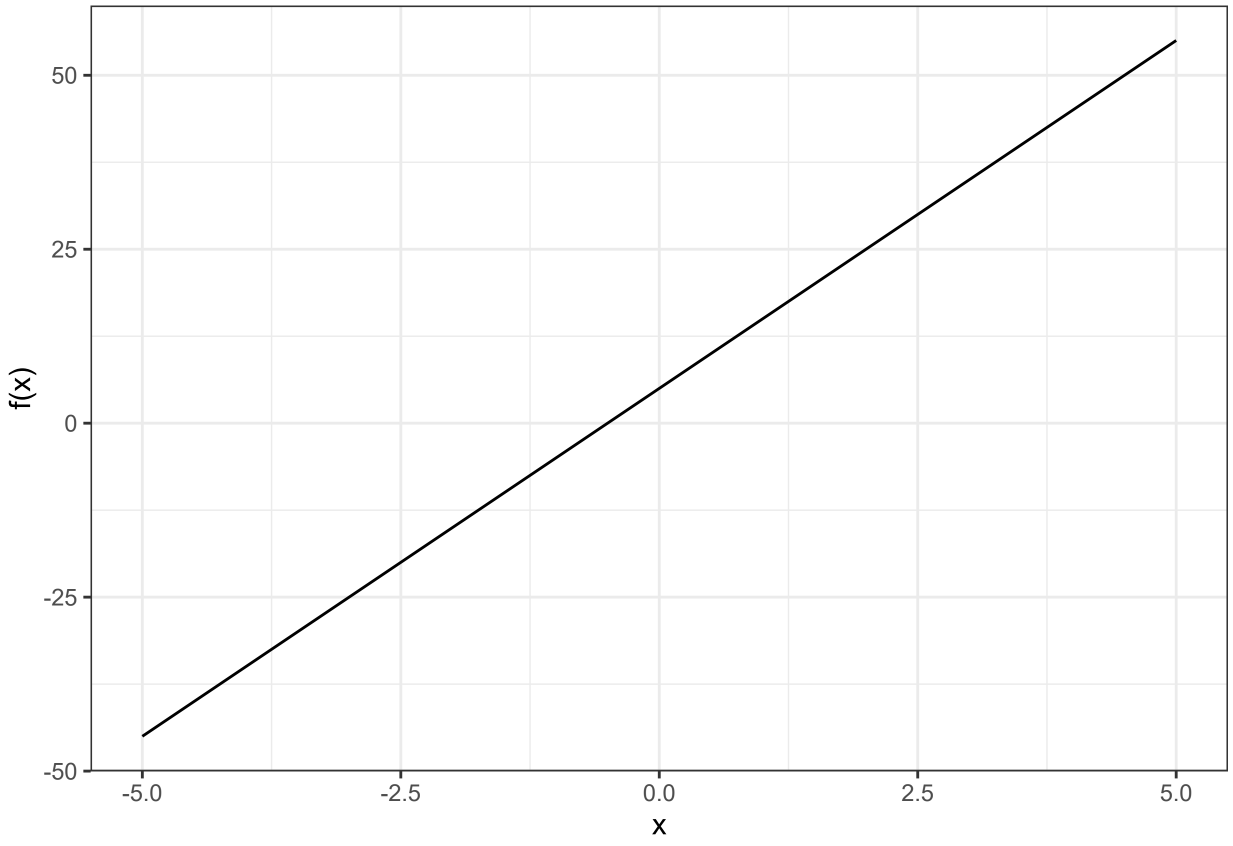
$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_n x^n,$$

where a_0, a_1, a_2, a_3,…, a_n are coefficients, some of which may be zero. The largest power of x in the function is called the order of the polynomial.

Below is an example for a first-order polynomial function which is a straight line.

$$f(x) = 10x + 5$$

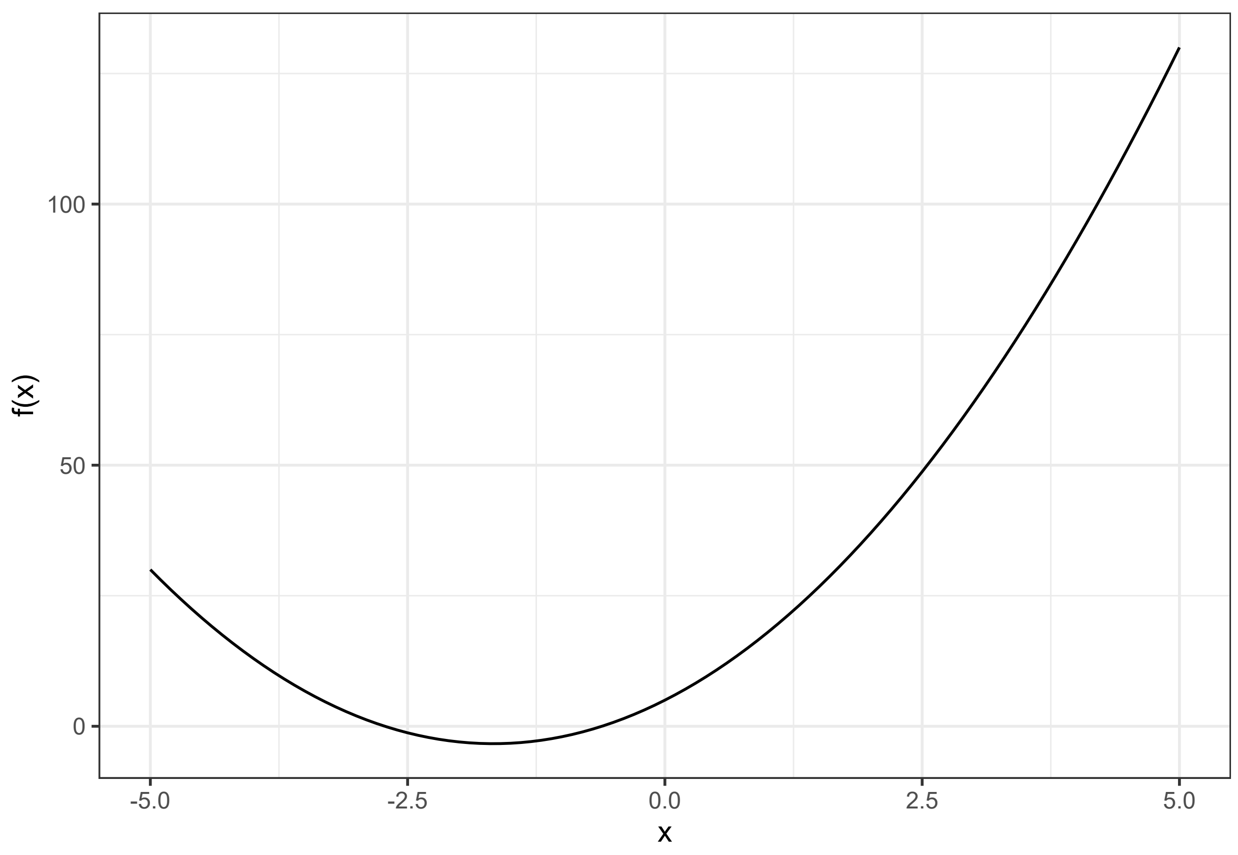
```r
f <- function(x) {10*x + 5}

ggplot() +
  geom_function(fun = f) +
  xlim(-5,5)+
  xlab('x')+
  ylab('f(x)')+
  theme_bw()
```



Below is an example for a second-order polynomial function.

$$f(x) = 3x^2 + 10x + 5$$

```r
f <- function(x) {3*x^2 + 10*x + 5}

ggplot() +
  geom_function(fun = f) +
  xlim(-5,5)+
  xlab('x')+
  ylab('f(x)')+
  theme_bw()
```



Exponential functions

Exponential functions are in the form of

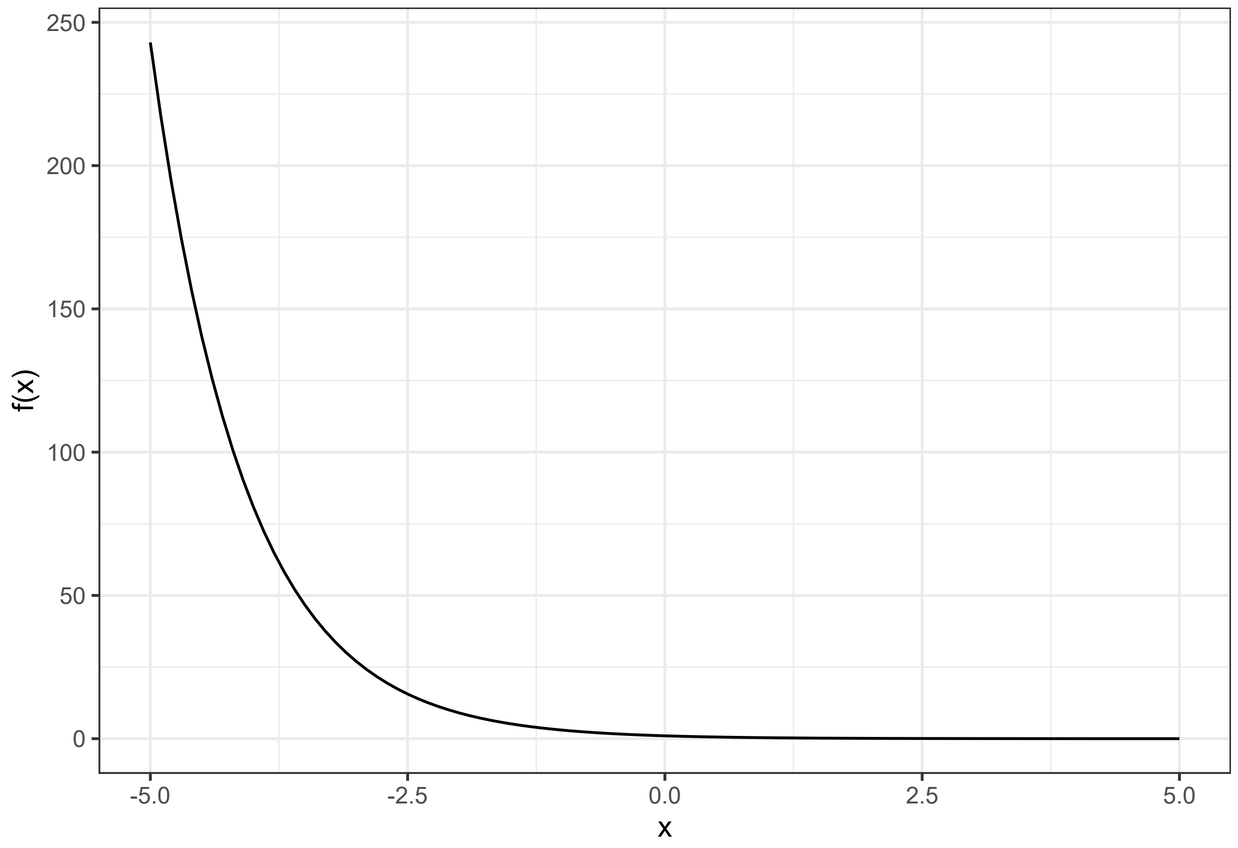
$$f(x) = a_0 a_1^x,$$

where a_0 and a_1 are coefficients.

Below is an example for an exponential.

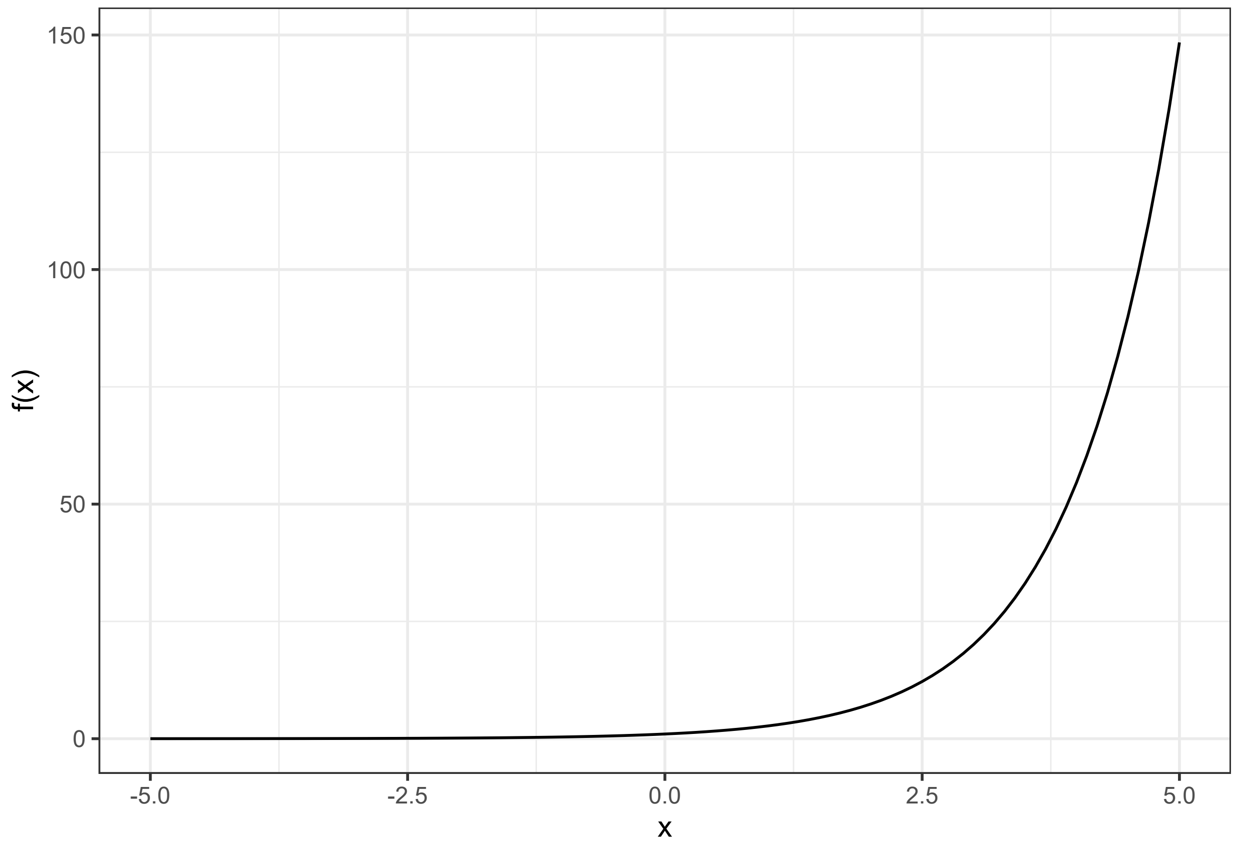
$$f(x) = (\frac{1}{3})^x$$

```r
f <- function(x) {(1/3)^x}

ggplot() +
  geom_function(fun = f) +
  xlim(-5,5)+
  xlab('x')+
  ylab('f(x)')+
  theme_bw()
```



A common exponential function uses e as a base. e is a mathematical constant just like π and equals to 2.718282… In R, you can use `exp(x)` to specify e^x. Below is the graph for e^2

```r
f <- function(x) {exp(x)}

ggplot() +
  geom_function(fun = f) +
  xlim(-5,5)+
  xlab('x')+
  ylab('f(x)')+
  theme_bw()
```



Logarithmic functions

Logarithmic functions are the inverse of exponential functions. The expression

$$x = b^y$$

can be written as

$$y = log_b(x)$$

.b is called the base. In R, you can use `log()` function to take logarithms.

See the following examples:

- $log_{10}(100) = 2$ because $10^2 = 100$.

```
log(100,base=10)
```

```
[1] 2
```

- $log_2(16) = 4$ because $2^4 = 16$.

```
log(16,base=2)
```

```
[1] 4
```

The most commonly used base is the base $e \approx 2.718$ and it is called natural logarithm when the base is e.

- $log_e(10) \approx 2.3026$ because $e^{2.3026} \approx 10$.

```
log(10)

  # when you don't specify a base, log() function uses e as base by default
```

```
[1] 2.302585
```

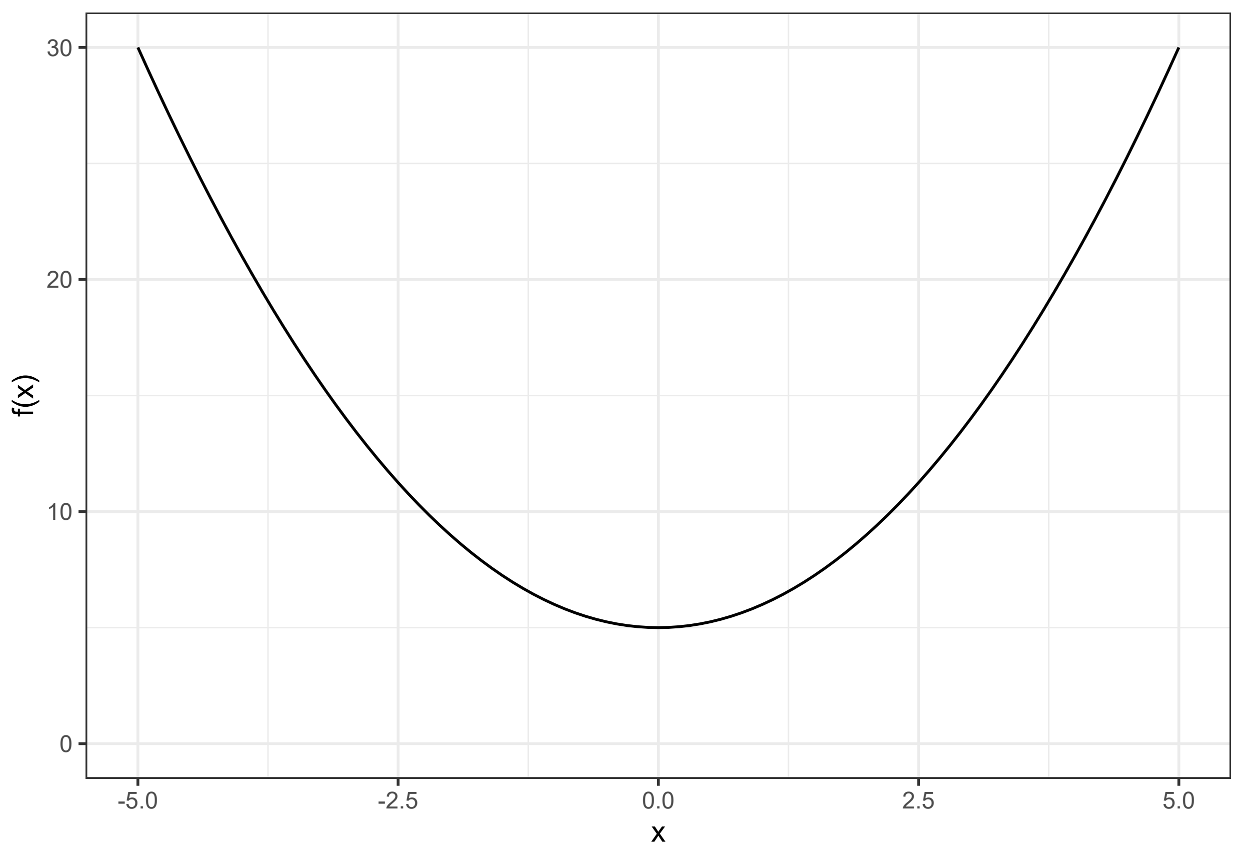
Bounded functions

Some functions are bounded such that the output values are restricted within a range. Consider the following function.

$$f(x) = x^2 + 5$$

This function can produce output values only within the the range of $[5, \infty]$

```
f <- function(x) {x^2 + 5}

ggplot() +
  geom_function(fun = f) +
  xlim(-5,5)+
  ylim(0,30)+
  xlab('x')+
  ylab('f(x)')+
  theme_bw()
```

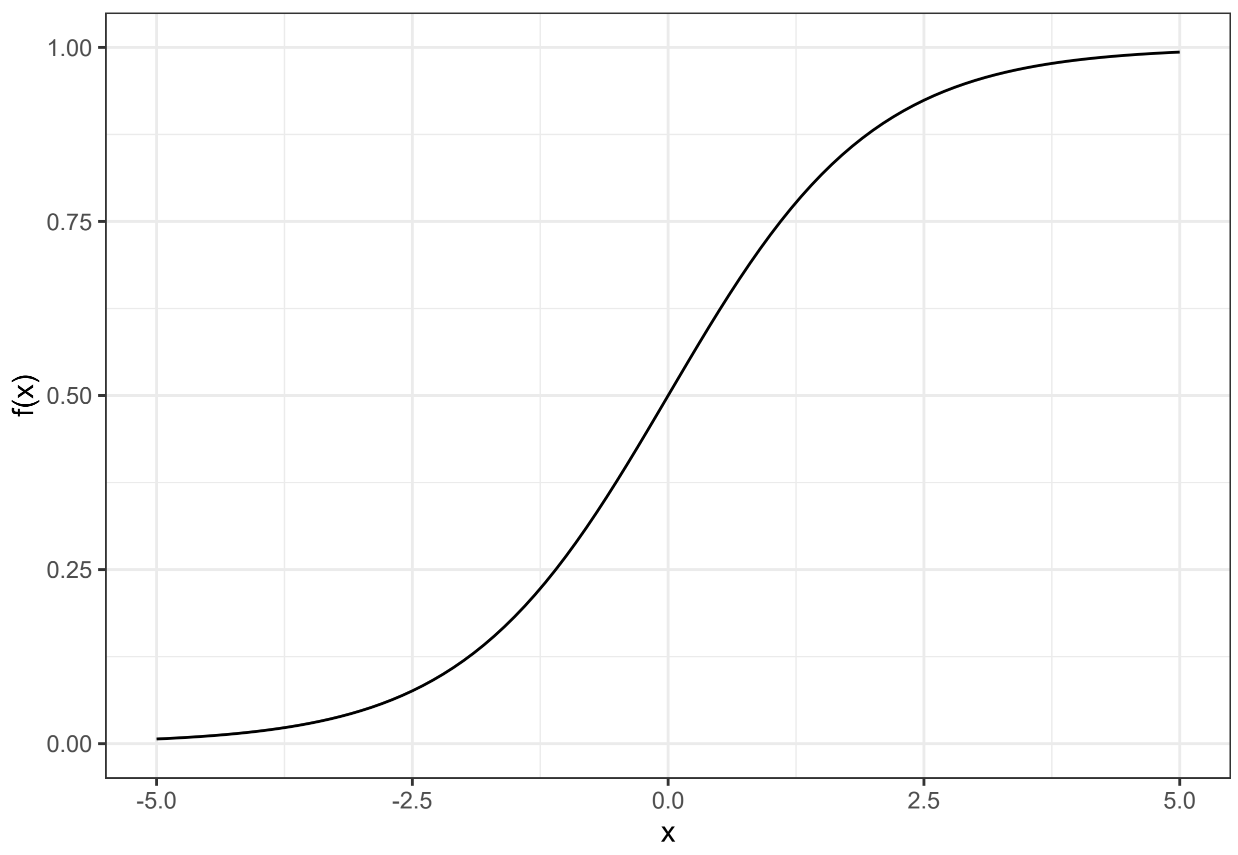


An important bounded function is the sigmoid function (a.k.a., logistic function).

$$f(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$$

The sigmoid function is always bounded between 0 and 1, and produces an S-shaped curve.

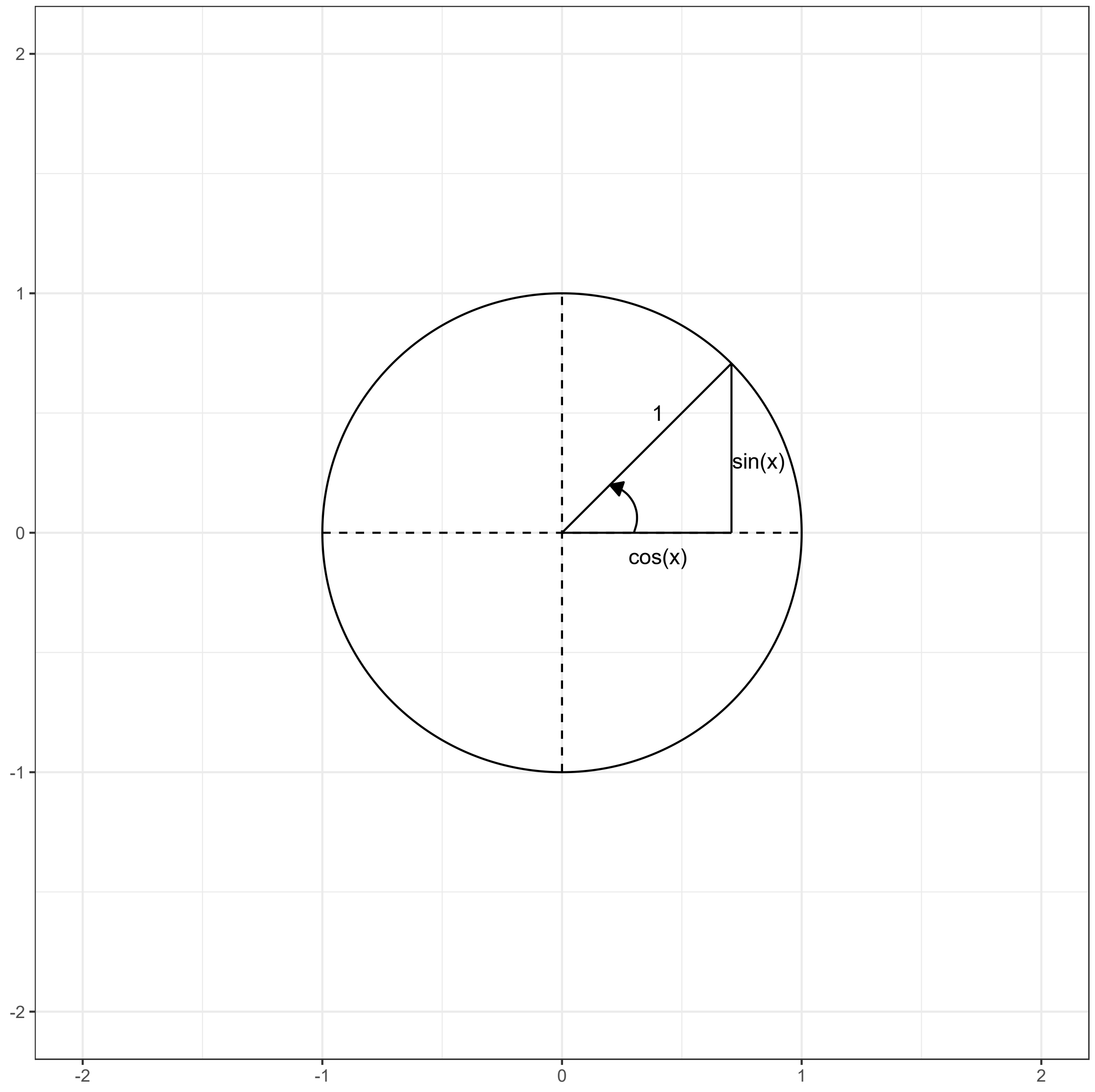
```
f <- function(x) {1/(1+exp(-x))}

ggplot() +
  geom_function(fun = f) +
  xlim(-5,5)+
  ylim(0,1)+
  xlab('x')+
  ylab('f(x)')+
  theme_bw()
```



Trigonometric functions

Trigonometric functions are defined using a *unit circle*, a circle centered at the origin and with a radius of 1. An angle always creates a triangle inside the circle. The cosine function takes the magnitude of the angle as an input and returns the signed length of the side of the triangle adjacent to the angle while the sine function returns the signed length of the opposite side. The tangent functions returns the ratio of the signed length of the opposite side to the signed length of the adjacent side.

```r
require(ggforce)

ggplot() +
  geom_circle(aes(x0=0,y0=0,r=1))+
  geom_segment(aes(x=0,y=-1,xend=0,yend=1),lty=2)+
  geom_segment(aes(x=-1,y=0,xend=1,yend=0),lty=2)+
  geom_segment(aes(x=0,y=0,xend=sqrt(.5),yend=sqrt(.5)))+
  geom_segment(aes(x=sqrt(.5),y=0,xend=sqrt(.5),yend=sqrt(.5)))+
  geom_segment(aes(x=0,y=0,xend=sqrt(.5),yend=0))+
  geom_curve(aes(x = 0.3, y = 0, xend = .2, yend = 0.2),
             color = "black",
             arrow = arrow(type = "closed",length=unit(0.1, "inches")))+
  xlim(-2,2)+
  ylim(-2,2)+
  xlab('')+
  ylab('')+
  theme_bw() +
  annotate('text',x=.4,y=-.1,label='cos(x)')+
  annotate('text',x=.82,y=.3,label='sin(x)')+
  annotate('text',x=.4,y=.5,label='1')
```



In R, you can use `cos()`, `sin()`, and `tan()` commands to calculate the value of cosine, sine, and tangent for a given angle. Note that an angle can be specified in terms of a degrees or radians, and these functions take radians as input. Below is simple formula to convert degrees to radians.

For instance, below code calculates the sine, cosine, and tangent for a 45 degree angle.

```
# pi = 180 degrees

cos(pi/4)
```

```
[1] 0.7071068
```

```
sin(pi/4)
```

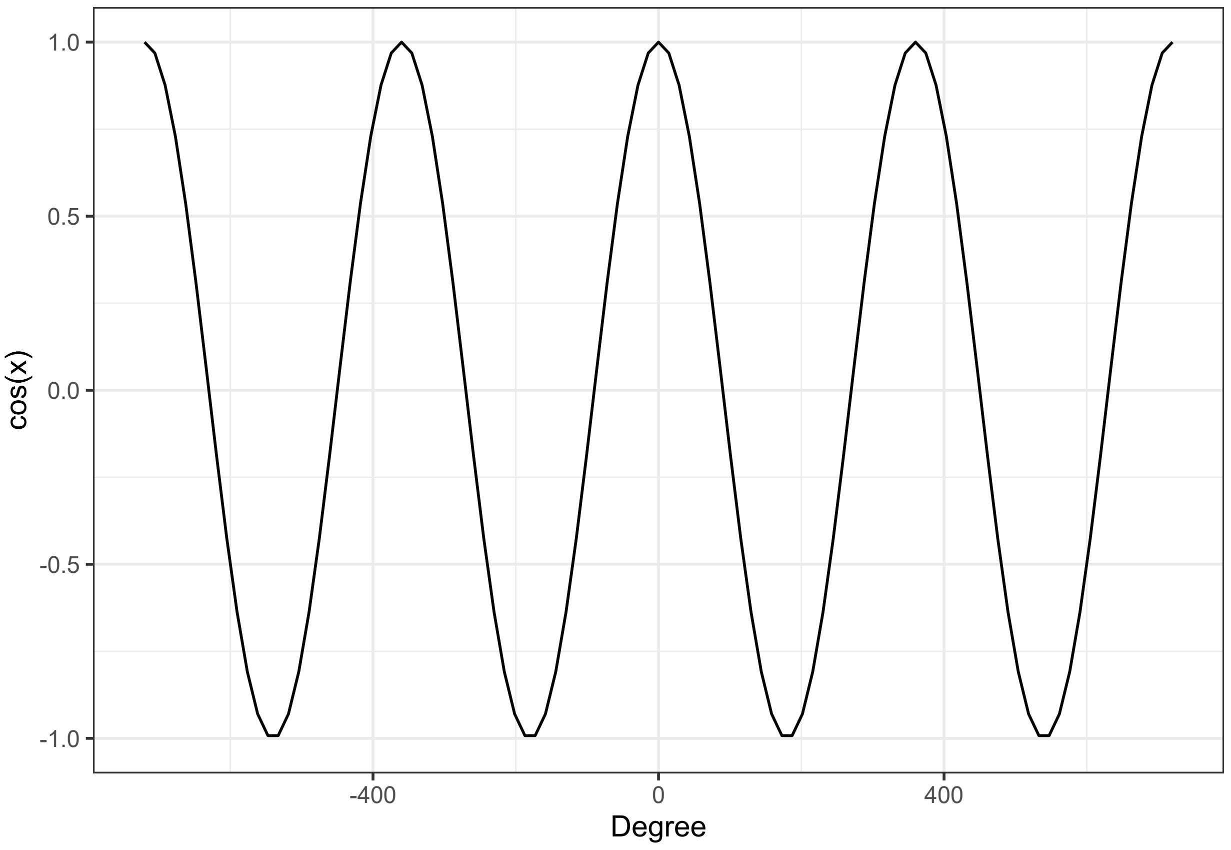
```
[1] 0.7071068
```

```
tan(pi/4)
```

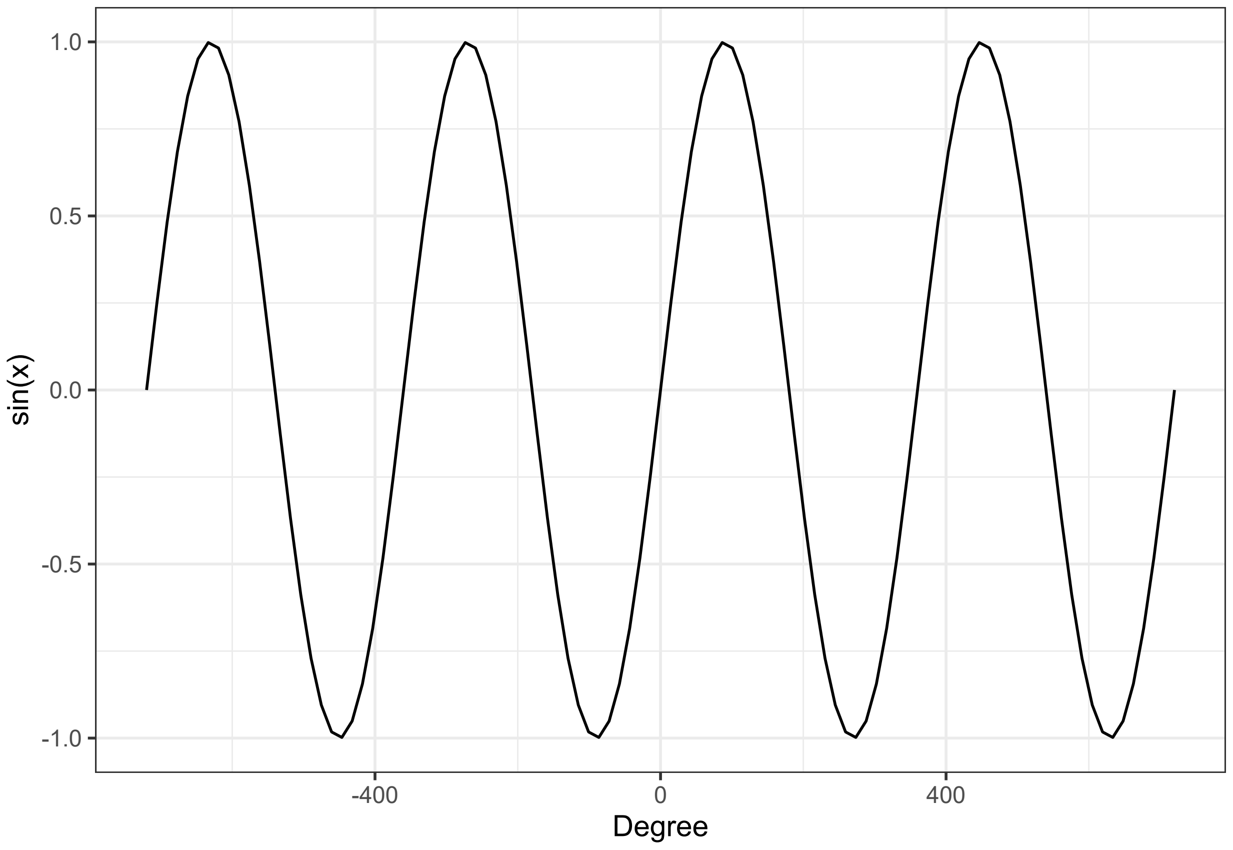
```
[1] 1
```

The cosine and sine functions are always return values between -1 and 1, while tangent can take values from - ∞ to ∞.

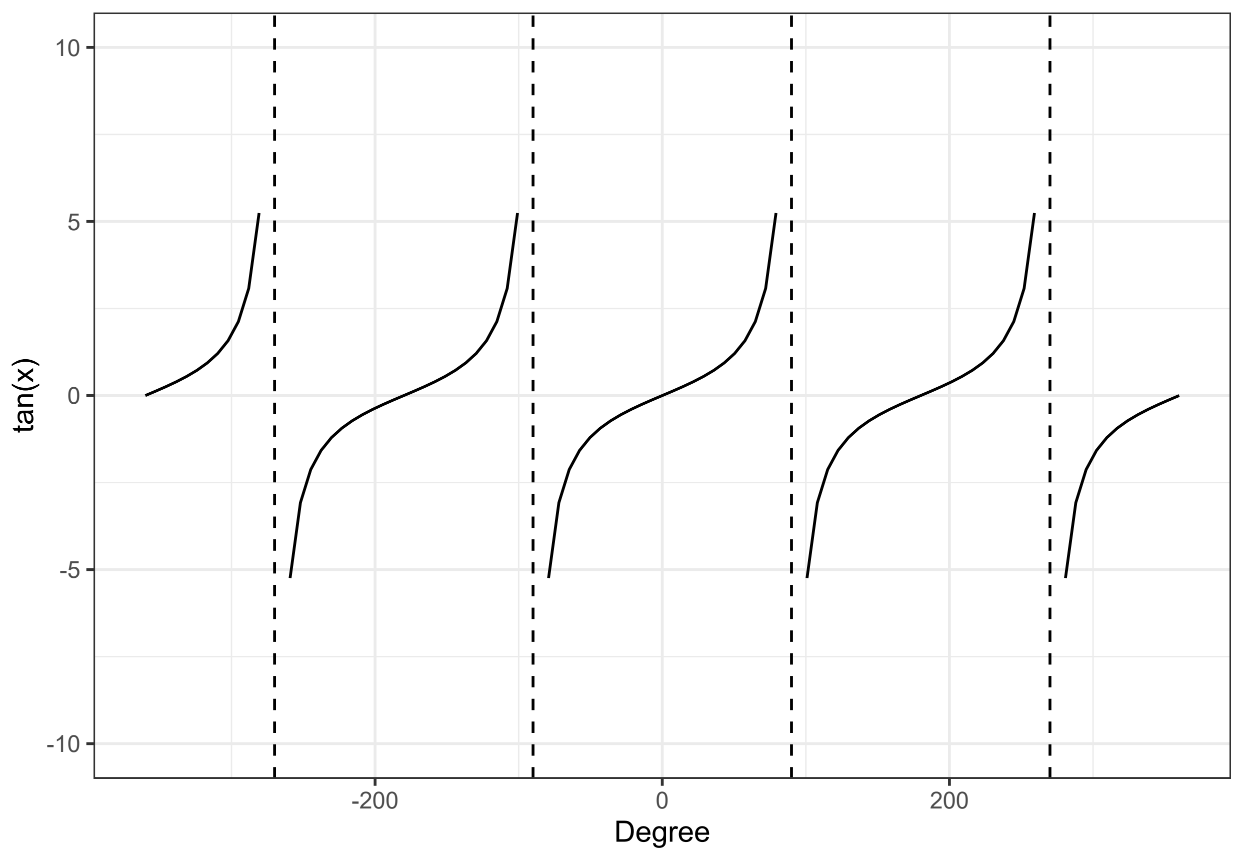
```
f <- function(x) {cos(pi*(x/180))}

ggplot() +
  geom_function(fun = f) +
  xlim(-720,720)+
  ylim(-1,1)+
  xlab('Degree')+
  ylab('cos(x)')+
  theme_bw()
```



```r
f <- function(x) {sin(pi*(x/180))}

ggplot() +
  geom_function(fun = f) +
  xlim(-720,720)+
  ylim(-1,1)+
  xlab('Degree')+
  ylab('sin(x)')+
  theme_bw()
```



```r
f <- function(x) {tan(pi*(x/180))}

ggplot() +
  geom_function(fun = f) +
  geom_vline(xintercept = -270,lty=2)+
  geom_vline(xintercept = -90,lty=2)+
  geom_vline(xintercept = 90,lty=2)+
  geom_vline(xintercept = 270,lty=2)+
  xlim(-360,360)+
  ylim(-10,10)+
  xlab('Degree')+
  ylab('tan(x)')+
  theme_bw()
```



Multivariate Functions

Functions don’t have to have a single input variable. Some functions may have multiple inputs. For instance, below is an example of a function with two inputs.

$$f(x, y) = x^2 + 2y - 5$$

Below this function looks like in 3D. x-axis and y-axis represent the input values and z-axis represent the output value based on the rule above. Note that most functions we will deal in machine learning are high dimensional with many variable inputs.

```r
require(plotly)

grid     <- expand.grid(x=seq(-20,20,.2),y=seq(-20,20,.2))
grid$z <- grid[,1] ^2 + 2*grid[,2] - 5

plot_ly(grid,
        x = ~x,
        y = ~y,
        z = ~z) %>%
  layout(
    xaxis = list(range = c(-40, 40)),
    yaxis = list(range = c(-40, 40)))
```

WebGL is not supported by your browser - visit https://get.webgl.org for more info

The Derivative of a Function

First, let's define the *difference quotient.* The difference quotient is defined as the change in a function's output value divided by the change in the function's input value. Suppose we have a $f(x)$ and find the output value for two input values, x_1 and x_2.

$$y_1 = f(x_1)$$
$$y_2 = f(x_2)$$

Then, the difference quotient is

$$\frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x} = \frac{f(x_2) - f(x_1)}{x_2 - x_1},$$

where Δ means *change.*

Let's see an example. Consider the following function:

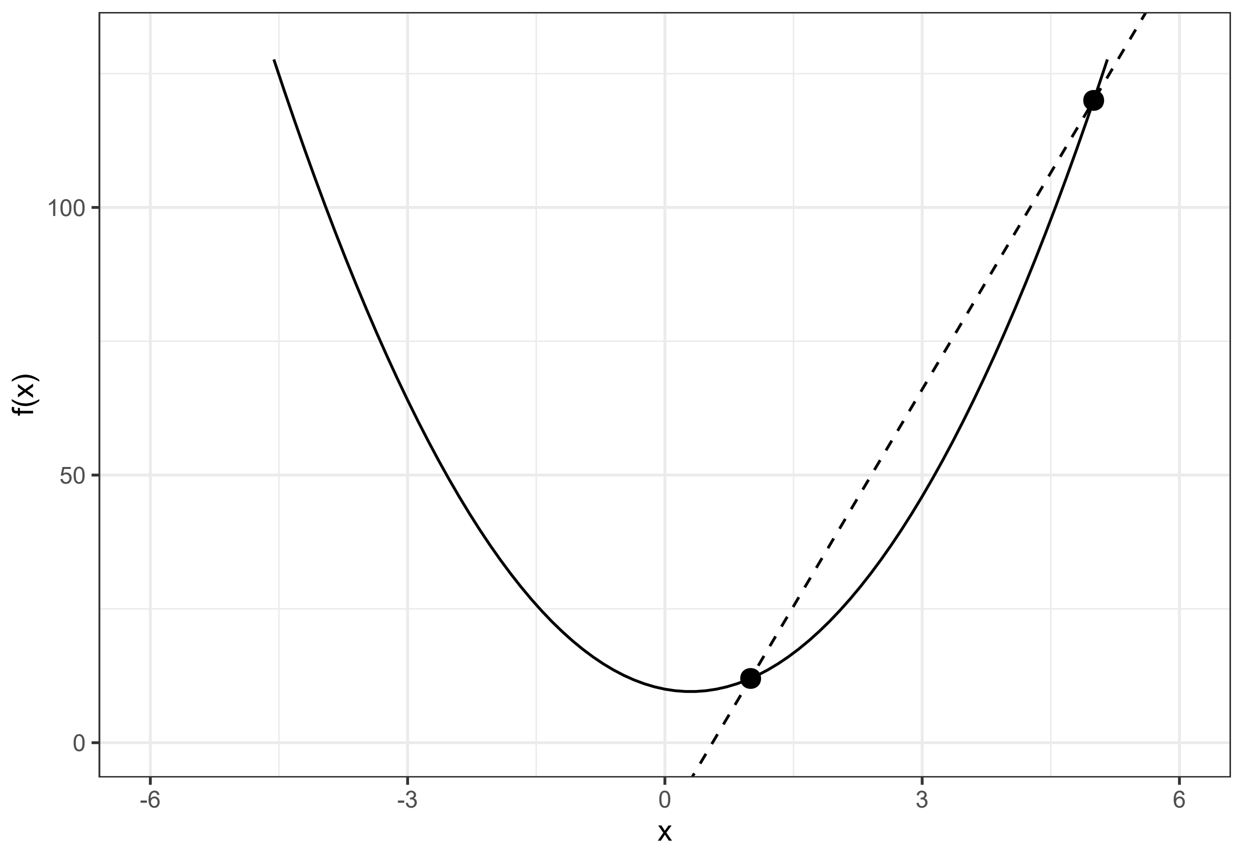
$$f(x) = y = 5x^2 - 3 * x + 10.$$

If we compute the output value for $x_1 = 1$ and $x_2 = 5$, the difference quotient is

$$\frac{\Delta y}{\Delta x} = \frac{f(5) - f(1)}{5-1} == \frac{120-12}{5-1} = 27.$$

If you look at this visually, the difference quotient gives the slope of the line that connects the two points (1,12) and (5,120). This line is called the secant line.

```r
f <- function(x) {5*x^2 -3*x + 10}

ggplot() +
  geom_function(fun = f) +
  geom_point(aes(x=1,y=12),size = 3) +
  geom_point(aes(x=5,y=120),size=3) +
  geom_abline(slope=27,intercept = -15,lty=2)+
  xlim(-6,6)+
  ylim(0,130)+
  xlab('x')+
  ylab('f(x)')+
  theme_bw()
```

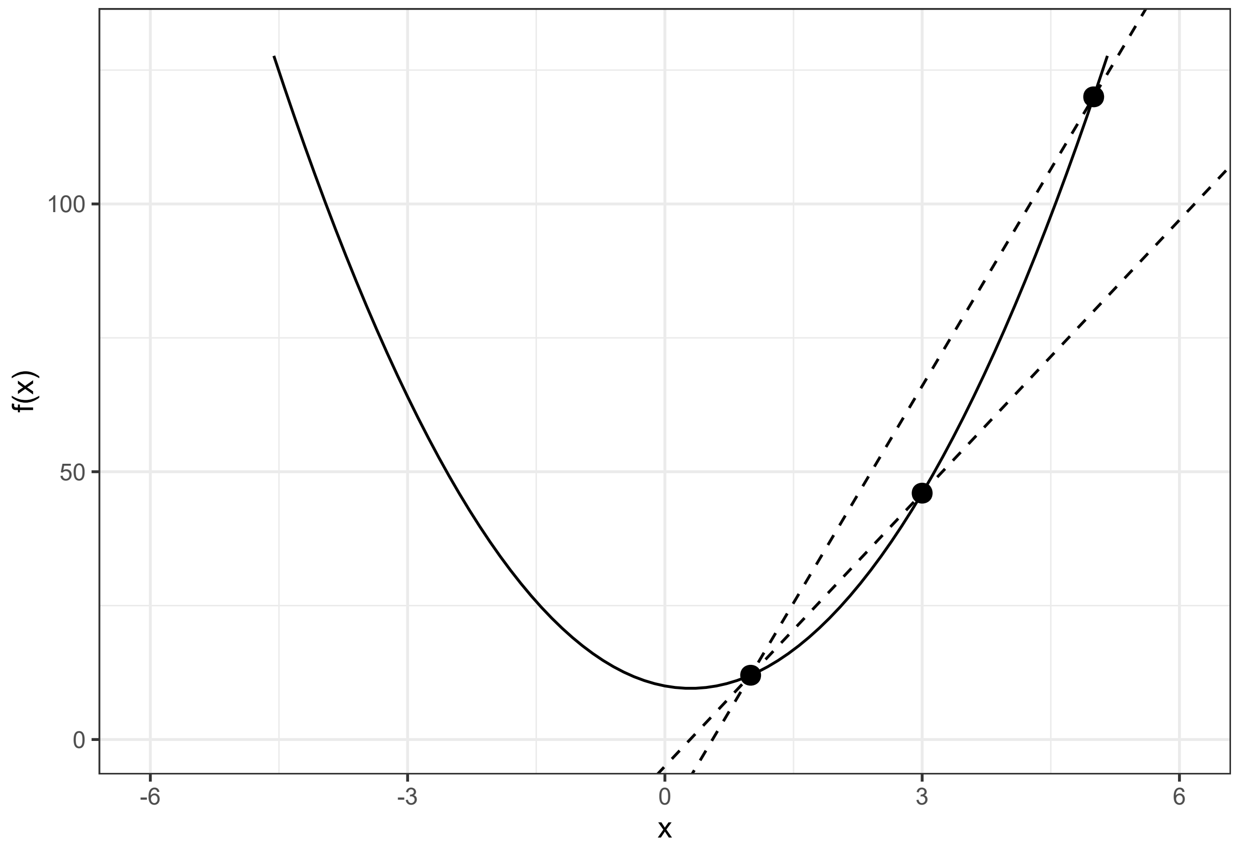


Now, consider that the change in x is smaller. Instead of moving x from 1 to 5, let's move x from 1 to 3.

$$\frac{\Delta y}{\Delta x} = \frac{f(3) - f(1)}{3-1} == \frac{46-12}{3-1} = 17.$$

The slope of the secant line, the line that connects (1,12) to (3,40), is now 17.

```r
f <- function(x) {5*x^2 -3*x + 10}

ggplot() +
  geom_function(fun = f) +
  geom_point(aes(x=1,y=12),size=3) +
  geom_point(aes(x=5,y=120),size=3) +
  geom_abline(slope=27,intercept = -15,lty=2)+
  geom_point(aes(x=3,y=46),size=3) +
  geom_abline(slope=17,intercept = -5,lty=2)+
  xlim(-6,6)+
  ylim(0,130)+
  xlab('x')+
  ylab('f(x)')+
  theme_bw()
```

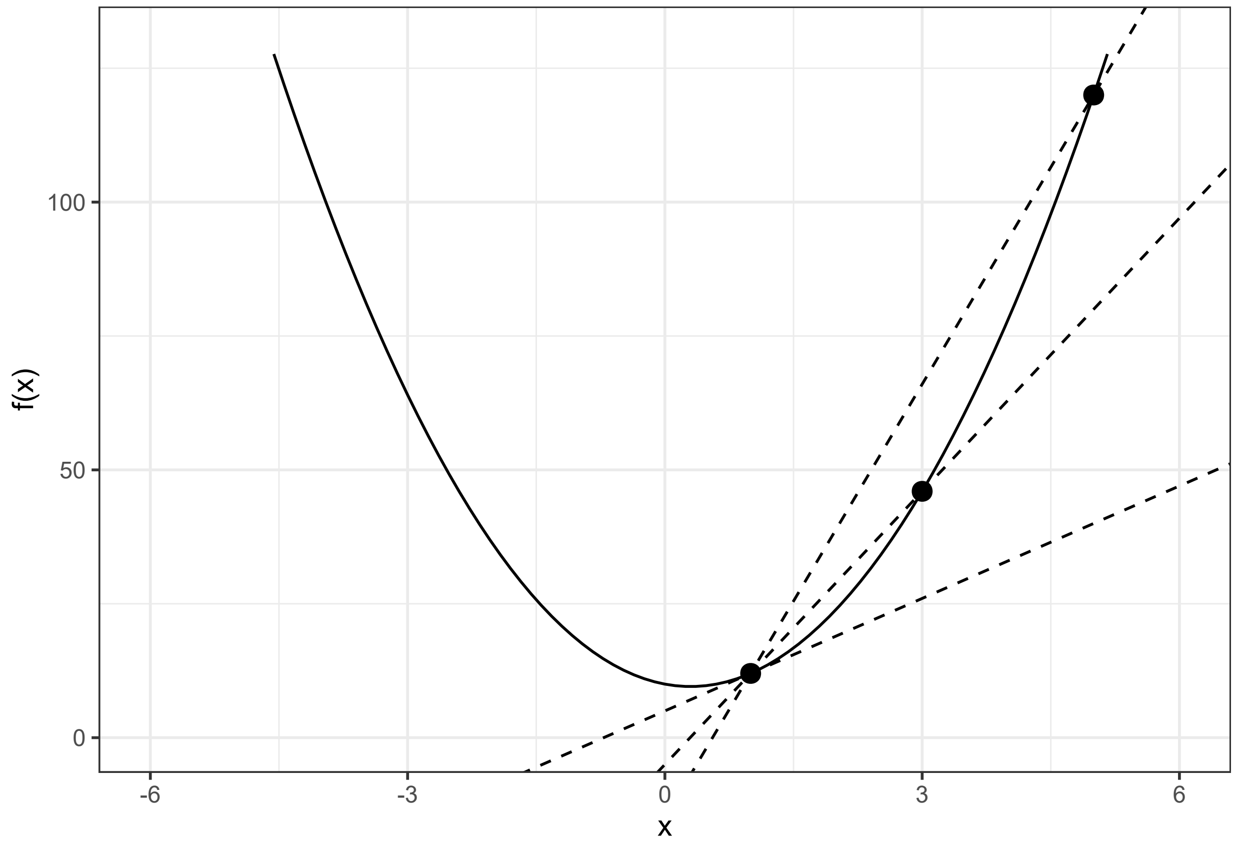


Now, consider that the change in x is such a tiny number, something not even noticeable such as 0.00001. So, let's move x from 1 to 1.0001. The slope becomes 7.00005. In this case, secant line becomes a line that barely touches the function at $x = 1$.

$$\frac{\Delta y}{\Delta x} = \frac{f(1.00001) - f(1)}{1.00001 - 1} = 7.00005.$$

```r
f <- function(x) {5*x^2 -3*x + 10}
```

```r
ggplot() +
  geom_function(fun = f) +
  geom_point(aes(x=1,y=12),size=3) +
  geom_point(aes(x=5,y=120),size=3) +
  geom_abline(slope=27,intercept = -15,lty=2)+
  geom_point(aes(x=3,y=46),size=3) +
  geom_abline(slope=17,intercept = -5,lty=2)+
  geom_abline(slope=7,intercept = 5,lty=2)+
  xlim(-6,6)+
  ylim(0,130)+
  xlab('x')+
  ylab('f(x)')+
  theme_bw()
```



First-order derivative

The first-order derivative of a function is the limit of the difference quotient as the change in x approximates to zero. In other words, when the change in x becomes infinitely small, such a tiny number nobody has any idea what it is, the secant line becomes a line that barely touches to the function at x_1 and called *tangent line*. The first-order derivative of a function is something that tells us about the slope of this tangent line.

The first-order derivative of a function is denoted as $\frac{dy}{dx}$ and more formally defined as the following:

$$\frac{df(x)}{dx} = \frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{f(x_1 + \Delta x) - f(x_1)}{\Delta x}$$

Most of the time you can numerically approximate the first-order derivative of a function by selecting such a small number for Δx (e.g., .00001) as we did above, and it works no matter how complex the function is. You can also analytically find the first-order derivative of a function but it may become a tedious job as the function becomes more complex and you have to be familiar with a number of derivative rules and most of the time be creative when finding a solution. In some cases, there may not even be an easy analytic solution and numerical approximation is the easiest solution.

You can get help from several tools while trying to find the first-order derivative of a function. There are online tools (e.g., [https://www.symbolab.com/solver/derivative-calculator]) you can use. You simply write the function and it returns the first-order derivative. In R, you can use the `calculus` package to find the derivatives of a function. For instance, see below for the example above.

```r
require(calculus)

derivative(f = '5*x^2 -3*x + 10',
           var = 'x',
           order = 1)

  # var = 'x', indicates that we are taking the derivative with respect to variable x
  # order = 1, indicates that we are asking for the first derivative
```

```
[1] "5 * (2 * x) - 3"
```

This indicates that the first-order derivative of the function

$$f(x) = 5x^2 - 3 * x + 10$$

is

$$f'(x) = 10x - 3.$$

Note that the first derivative of $f(x)$ is also denoted as $f'(x)$.

$f'(x = x_1)$ gives you the slope of the tangent line that touches to $f(x)$ at $x = x_1$. The slope of the tangent line at x=1 for this function can be directly obtained using the first derivative of the function.

$$f'(x) = 10 * x - 3$$

$$f'(1) = 10 * 1 - 3 = 7$$

It is very close to what we computed above using a tiny change in x above, $\Delta x = 0.00001$. Analytical solution is the accurate one, but numerical solution typically approximates enough for most applications.

You can also use the same `derivative()` function from the `calculus` package to get the numerical approximation for the first-order derivative of any function.

```r
derivative(f = '5*x^2 -3*x + 10',
           var = c(x=1),
           order = 1)

  # var = c(x=1), indicates that we are not interested in symbolic derivative
    # function and want to obtain a numerical solution of the
    # derivative function at x=1
```

```
[1] 7
```

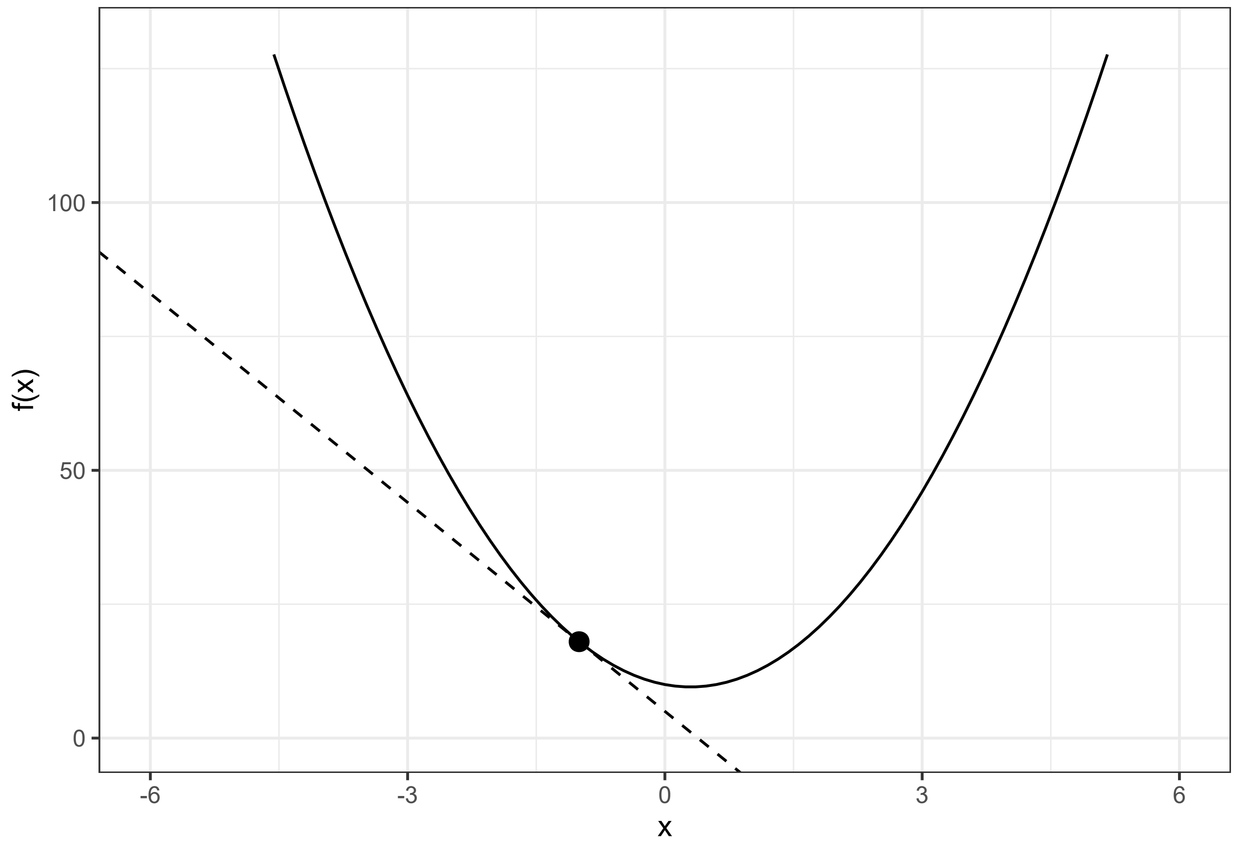
Let's find the first-order derivative at $x = -1$.

```r
derivative(f = '5*x^2 -3*x + 10',
           var = c(x=-1),
           order = 1)
```

```
[1] -13
```

This indicates that the value of the first-order derivative at $x = -1$ is -13. Note that this is a negative number, so we understand that the slope of the tangent line that touches to this function at $x = -1$ is negative.

```r
f <- function(x) {5*x^2 -3*x + 10}

ggplot() +
  geom_function(fun = f) +
  geom_point(aes(x=-1,y=18),size=3) +
  geom_abline(slope= - 13,intercept = 5,lty=2)+
  xlim(-6,6)+
  ylim(0,130)+
  xlab('x')+
  ylab('f(x)')+
  theme_bw()
```



Second-order and higher-order derivatives

As derivatives are also functions, we can always take a derivative of a derivative. If we take the derivative of a first-order derivative function, then it becomes the second-order derivative of the original function. For instance, see below for the first-order and second-order derivatives of the original function above.

```
derivative(f = '5*x^2 -3*x + 10',
           var = 'x',
           order = 1)

derivative(f = '5*x^2 -3*x + 10',
           var = 'x',
           order = 2)
```

```
[1] "5 * (2 * x) - 3"
[1] "5 * 2"
```

$$f(x) = y = 5x^2 - 3 * x + 10.$$

$$f'(x) = 10x - 3.$$

$$f''(x) = 10.$$

Below is an example of another function and its derivatives up to the fifth order.

```
derivative(f = '5*x^4 + 3*x^2 + 4*x + 5',
           var = 'x',
           order = 1)

derivative(f = '5*x^4 + 3*x^2 + 4*x + 5',
           var = 'x',
           order = 2)

derivative(f = '5*x^4 + 3*x^2 + 4*x + 5',
           var = 'x',
           order = 3)

derivative(f = '5*x^4 + 3*x^2 + 4*x + 5',
           var = 'x',
           order = 4)

derivative(f = '5*x^4 + 3*x^2 + 4*x + 5',
           var = 'x',
           order = 5)
```

```
[1] "5 * (4 * x^3) + 3 * (2 * x) + 4"
[1] "5 * (4 * (3 * x^2)) + 3 * 2"
[1] "5 * (4 * (3 * (2 * x)))"
[1] "5 * (4 * (3 * 2))"
[1] "0"
```

$$f(x) = 5x^4 - 3x^2 + 4x + 5$$

$$f'(x) = 20x^3 - 6 * x + 4$$

$$f''(x) = 60x^2 - 6$$

$$f'''(x) = 120x$$

$$f''''(x) = 120$$

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

Partial Derivatives

A partial derivative is a derivative of a multivariate function with respective to a particular input variable while treating all other inputs as constant. The examples you read until this point considered a function with a single variable. Most of the time, we have to deal with functions with multiple variables, and they are sometimes highly complex. Consider the following function with two variables:

$$f(x, y) = -\sqrt{(x-1)^2 + y^2}$$

We can take the first-order derivative of this function with respect to x while treating y as a constant, or we can take the first-order derivative of this function with respect to y while treating x as a constant. Without knowing anything about the rules of taking derivatives, you can find them using the `derivative()` function from the `calculus` package.

```
derivative(f = '-sqrt((x-1)^2 + y^2)',
           var = 'x',
           order = 1)
```

```
[1] "-(0.5 * (2 * (x - 1) * ((x - 1)^2 + y^2)^-0.5))"
```

```
derivative(f = 'sin(x^2/2 - y^2/4)*cos(2*x - exp(y))',
           var = 'y',
           order = 1)
```

```
[1] "sin(x^2/2 - y^2/4) * (sin(2 * x - exp(y)) * exp(y)) - cos(x^2/2 - y^2/4) * (2 * y/4) * cos(2 * x -
```

Below are the partial derivatives of this function with respect to x and y.

$$\frac{\partial f(x,y)}{\partial x} = \frac{1-x}{\sqrt{(x-1)^2 + y^2}}$$

$$\frac{\partial f(x,y)}{\partial y} = \frac{-y}{\sqrt{(x-1)^2 + y^2}}$$

Note that we use ∂ symbol instead of d to differentiate that these are partial derivatives.

Gradient

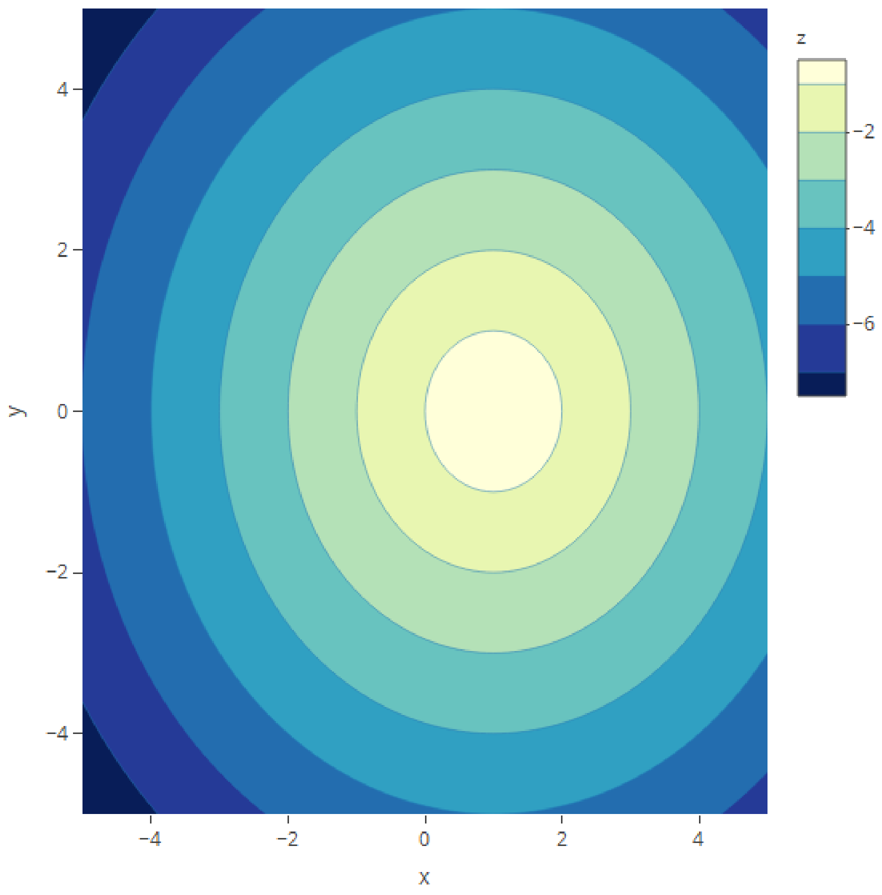
The first-order derivative of a single variable function indicates the slope of a tangent line that touches to that function at a particular point. Then, how do we interpret the partial derivatives? It is best to visualize this to get a glimpse of it. Below are how this function looks like in 3D. I also added a counterplot to represent the same shape in 2D.

```r
grid     <- expand.grid(x=seq(-5,5,.1),y=seq(-5,5,.1))
grid$z   <- -((grid[,1] - 1)^2 + grid[,2]^2)^.5

plot_ly(grid,
        x = ~x,
        y = ~y,
        z = ~z,
        marker = list(color = ~z,
                      colorscale = 'YlGnBu',
                      showscale = TRUE)) %>%
  add_markers() %>%
  layout(scene = list(xaxis=list(range = c(-5,5)),
                      yaxis=list(range = c(-5,5))))
```

WebGL is not supported by your browser - visit https://get.webgl.org for more info

```r
grid     <- expand.grid(x=seq(-5,5,.1),y=seq(-5,5,.1))
grid$z   <- -((grid[,1] - 1)^2 + grid[,2]^2)^.5
```

```r
plot_ly(grid,
        x = ~x,
        y = ~y,
        z = ~z,
        type='contour',
        colors = 'YlGnBu',
        showscale=TRUE,
        reversescale =T)
```



Now, suppose that we dropped you with a helicopter on this surface and your coordinates are $x = -2$ and $y = 2$. Your goal is to hike to the top. The partial derivatives would help you to find your direction to the top in your hike. The partial derivatives with respect to x and with respect y would be equal to 0.832 and -0.554, respectively.

$$\frac{\partial f(x,y)}{\partial x} = \frac{1-(-2)}{\sqrt{(-2-1)^2+2^2}} = 0.8320503$$

$$\frac{\partial f(x,y)}{\partial y} = \frac{-2}{\sqrt{(-2-1)^2+2^2}} = -0.5547002$$

Or, you can find these values using the `gradient()` function from the `calculus` package.

```r
gradient(f = '-sqrt((x-1)^2 + y^2)',
           var = c(x=-2,y=2))
```

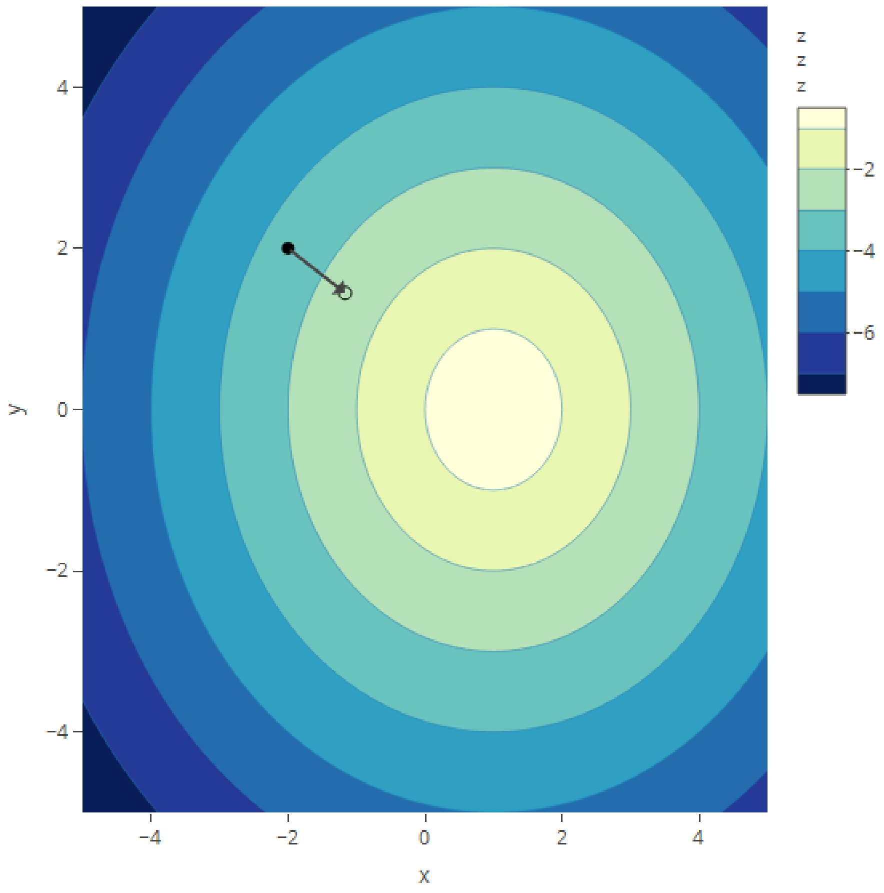
```
[1]  0.8320503 -0.5547002
```

If we add the gradient values with respect to x and with respect to y to the original coordinates of x and y, then the new coordinate gives the direction with the largest slope at (x, y) towards the maximum. In the graph below, the filled dot is (-2,2) and open circle is (-2 + 0.832, 2 - 0.554). The direction from the filled dot to the open circle gives the largest slope towards the maximum (steepest ascent). Or, if you want to reach to the maximum, it shows you the best next step.

The dimensions of gradient depends on the number of inputs in a function. Suppose f is a function with n inputs, then the gradient of this function will be a column vector with length n.

$$\Delta f = f'(x_1, x_2, ..., x_n) = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ ... \\ \frac{\partial y}{\partial x_n} \end{bmatrix}$$

```r
a <- -2
b <- 2

gr <- gradient(f = '-sqrt((x-1)^2 + y^2)',
               var = c(x=a,y=b))


grid     <- expand.grid(x=seq(-5,5,.1),y=seq(-5,5,.1))
grid$z   <- -((grid[,1] - 1)^2 + grid[,2]^2)^.5

plot_ly(grid,
        x = ~x,
        y = ~y,
        z = ~z,
        type='contour',
        colors = 'YlGnBu',
        showscale=TRUE,
        reversescale =T) %>%
  add_trace(x = a,
            y = b,
            type = 'scatter',
            mode = 'markers',
            marker = list(color = 'black',size=8),
            showlegend=FALSE,
            showlegend = FALSE) %>%
  add_trace(x = a+gr[1],
            y = b+gr[2],
            type = 'scatter',
            mode = 'markers',
            symbols = c('x'),
            marker = list(color = 'black',size=8,symbol = 'circle-open'),
            showlegend = FALSE) %>%
 add_annotations(ax = a,
                 ay = b,
                 x = a + gr[1],
                 y = b + gr[2],
                 axref = 'x',
                 ayref = 'y',
                 xref = 'x',
                 yref = 'y',
                 text = '',
                 showlegend=FALSE)
```



Note. We here assume that we are searching for the maximum. If we are searching for a minimum, then the open circle would be (-2 - 0.832, 2 + 0.554) and provide the direction with the largest slope towards the minimum (steepest descent).

Hessian

If we take the first-order partial derivatives from a multivariate function and take derivative again with respect to each variable, then we obtain the second-order partial derivatives. The Hessian matrix is a way of organizing the second-order partial derivatives. For instance, consider the function used in the previous section.

$$f(x,y) = -\sqrt{(x-1)^2 + y^2}$$

We found its first-order partial derivative with respect to x as the following..

$$\frac{\partial f(x,y)}{\partial x} = \frac{1-x}{\sqrt{(x-1)^2+y^2}}$$

$\frac{\partial f(x,y)}{\partial x}$ is a function itself, so we can again take its derivatives with respect to x and y.

$$\frac{\partial f^2(x,y)}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial f(x,y)}{\partial x}\right)$$
$$\frac{\partial f^2(x,y)}{\partial x \partial y} = \frac{\partial}{\partial y}\left(\frac{\partial f(x,y)}{\partial x}\right)$$

```
fprime.x <- '(1-x)/sqrt((x-1)^2 + y^2)'

derivative(f = fprime.x,
           var = 'x',
           order = 1)
```

```
[1] "-(1/sqrt((x - 1)^2 + y^2) + (1 - x) * (0.5 * (2 * (x - 1) * ((x - 1)^2 + y^2)^-0.5))/sqrt((x - 1)^
```

```
derivative(f = fprime.x,
           var = 'y',
           order = 1)
```

```
[1] "-((1 - x) * (0.5 * (2 * y * ((x - 1)^2 + y^2)^-0.5))/sqrt((x - 1)^2 + y^2)^2)"
```

This gets pretty ugly! I simplified it for you.

$$\frac{\partial f^2(x,y)}{\partial x^2} = -\frac{y^2}{\left((x-1)^2+y^2\right)^{\frac{3}{2}}}$$

$$\frac{\partial f^2(x,y)}{\partial x \partial y} = \frac{(x-1)\,y}{\left(y^2+(x-1)^2\right)^{\frac{3}{2}}}$$

We are not done yet! These are the derivatives of the first-order partial derivative with respect to x. We can do the same thing for the first-order partial derivative with respect to y.

$$\frac{\partial f^2(x,y)}{\partial y^2} = \frac{\partial}{\partial y}\left(\frac{\partial f(x,y)}{\partial y}\right)$$

$$\frac{\partial f^2(x,y)}{\partial y \partial x} = \frac{\partial}{\partial x}\left(\frac{\partial f(x,y)}{\partial y}\right)$$

```
fprime.y <- '(-y)/sqrt((x-1)^2 + y^2)'

derivative(f = fprime.y,
           var = 'x',
           order = 1)
```

```
[1] "-((-y) * (0.5 * (2 * (x - 1) * ((x - 1)^2 + y^2)^-0.5))/sqrt((x - 1)^2 + y^2)^2)"
```

```
derivative(f = fprime.x,
           var = 'y',
           order = 1)
```

```
[1] "-((1 - x) * (0.5 * (2 * y * ((x - 1)^2 + y^2)^-0.5))/sqrt((x - 1)^2 + y^2)^2)"
```

They are simplified for you.

$$\frac{\partial f^2(x,y)}{\partial y^2} = -\frac{(x-1)^2}{\left(y^2+(x-1)^2\right)^{\frac{3}{2}}}$$

$$\frac{\partial f^2(x,y)}{\partial y \partial x} = \frac{y\,(x-1)}{\left((x-1)^2+y^2\right)^{\frac{3}{2}}}$$

We can organize these second-order derivatives in a 2 x 2 Hessian matrix. Since our function has only two input variables, the Hessian matrix will be a 2 x 2 matrix. For a function with n input variables, the Hessian matrix would be an n by n matrix.

$$\mathbf{H} = \begin{bmatrix} \frac{\partial f^2(x,y)}{\partial x^2} & \frac{\partial f^2(x,y)}{\partial x \partial y} \\ \frac{\partial f^2(x,y)}{\partial y \partial x} & \frac{\partial f^2(x,y)}{\partial y^2} \end{bmatrix}$$

In the previous section, we calculated the gradient vector for $(x, y) = (-2, 2)$. Now, let's compute the Hessian matrix for the same $(x, y) = (-2, 2)$ coordinate.

$$\frac{\partial f^2(x,y)}{\partial x^2} = -\frac{2^2}{\left((-2-1)^2+2^2\right)^{\frac{3}{2}}} = -0.08533849$$

$$\frac{\partial f^2(x,y)}{\partial x \partial y} = \frac{(-2-1)\,2}{\left(2^2+(-2-1)^2\right)^{\frac{3}{2}}} = -0.1280077$$

$$\frac{\partial f^2(x,y)}{\partial y^2} = -\frac{(-2-1)^2}{\left(2^2+(-2-1)^2\right)^{\frac{3}{2}}} = -0.1920116$$

$$\frac{\partial f^2(x,y)}{\partial y \partial x} = \frac{2\,(-2-1)}{\left((-2-1)^2+2^2\right)^{\frac{3}{2}}} = -0.1280077$$

$$\mathbf{H} = \begin{bmatrix} -0.085 & -0.128 \\ -0.128 & 0.192 \end{bmatrix}$$

Luckily, we don't have to do this tedious work by ourself. We can use the `hessian()` function from the calculus package to compute the hessian matrix for any given function.

```r
hessian(f = '-sqrt((x-1)^2 + y^2)',
           var = c(x=-2,y=2))
```

```
             [,1]       [,2]
[1,] -0.08533849 -0.1280077
[2,] -0.12800774 -0.1920116
```

Optimization

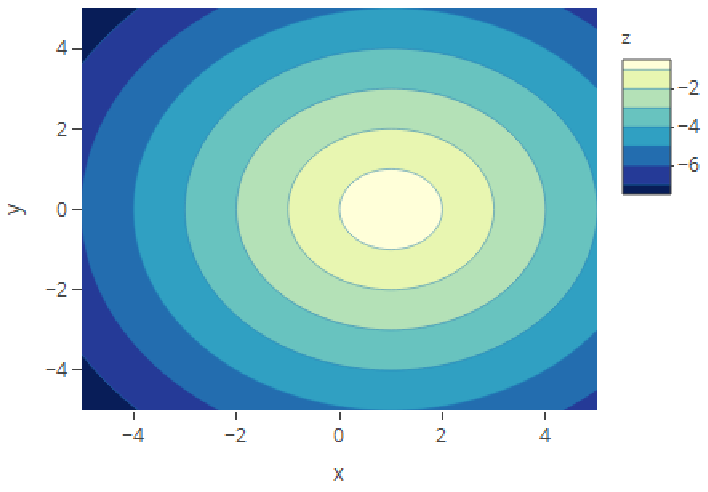
The optimization is finding a minimum or maximum of a function. The concept of gradient and Hessian can be used to find a minimum or maximum of a given function although they have their limitations. Here, we will focus on two methods, steepest ascent and Newton's algorithm, to give a sense of how optimization process looks like.

Steepest Ascent (or Descent)

Let's consider the following function and its first-order derivatives (gradient vector). We would like to know the coordinates of the maximum. At which specific coordinates, does this function reach to a maximum value? We can have a guess that it is somewhere between 0 and 2 in terms of the x-axis and somewhere between -1 and 1 in terms of the y-axis by looking at the plot. We can actually mathematically show that it is exactly $(x, y) = (1, 0)$, but we pretend that we do not know that.

$$f(x, y) = -\sqrt{(x-1)^2 + y^2}$$

```r
grid     <- expand.grid(x=seq(-5,5,.1),y=seq(-5,5,.1))
grid$z   <- -((grid[,1] - 1)^2 + grid[,2]^2)^.5

plot_ly(grid,
        x = ~x,
        y = ~y,
        z = ~z,
        type='contour',
        colors = 'YlGnBu',
        showscale=TRUE,
        reversescale =T)
```



Each optimization process starts with a wild guess about where the minimum or maximum is. Depending on the first guess, we can evaluate the gradient to update our guess. In a more formal way, we can summarize the steepest ascent for this function as the following.

Step 0: Wild estimate about where the maximum is.

$$(x_0, y_0)$$

Step 1: Update your first estimate.

$$(x_1, y_1) = (x_0, y_0) + \alpha f'(x_0, y_0)$$

StepY 2: Continue updating your estimate until it doesn't significantly improve anymore.

$$(x_{n+1}, y_{n+1}) = (x_n, y_n) + \alpha f'(x_n, y_n)$$

$f'(x_n, y_n)$ is the gradient vector at the n^th step and α is known as the step size. While gradient indicates which direction to go for the next step, α indicates how big of a step to take in that direction. In machine learning literature, they call α the *learning rate*.

Let's implement this algorithm for our function. For instance, suppose our initial estimate about the location of the maximum point is $(x, y) = c(-2, 2)$. Below is an R script using a `while` loop to iterate this procedure until one of the two conditions is met. Either the change in function value is negative (so the function doesn't improve) or the number of iterations is less than 500. I used an α value of 0.1 at each step.

```r
my.f                <- function(x,y){
 -sqrt((x-1)^2 + y^2)
}

f                   <- '-sqrt((x-1)^2 + y^2)'

iter                <- data.frame(x=NA,y=NA,z=NA)
iter[1,c('x','y')] <- c(-2,2)
iter[1,]$z          <- my.f(x=iter[1,1],y=iter[1,2])
```

```r
change              <- 1
i                   <- 1

while((change > 0) & (i < 500)){

  der               <- gradient(f = f,
                                var = c(x=iter[i,1],y=iter[i,2]))
  i                 <- i + 1

  iter[i,1:2]       <- iter[i-1,1:2] + 0.1*der
  iter[i,]$z        <- my.f(x=iter[i,1],y=iter[i,2])

  change            <- iter[i,]$z - iter[i-1,]$z

  #cat('Iteration :', i, 'Value :', round(iter[i,]$z,4),'\n')
}

iter
```

```
              x           y            z
1  -2.000000000 2.000000000 -3.605551275
2  -1.916794971 1.944529980 -3.505551275
3  -1.833589941 1.889059961 -3.405551275
4  -1.750384912 1.833589941 -3.305551275
5  -1.667179882 1.778119922 -3.205551275
6  -1.583974853 1.722649902 -3.105551275
7  -1.500769823 1.667179882 -3.005551275
8  -1.417564794 1.611709863 -2.905551275
9  -1.334359765 1.556239843 -2.805551275
10 -1.251154735 1.500769823 -2.705551275
11 -1.167949706 1.445299804 -2.605551275
12 -1.084744676 1.389829784 -2.505551275
13 -1.001539647 1.334359765 -2.405551275
14 -0.918334617 1.278889745 -2.305551275
15 -0.835129588 1.223419725 -2.205551275
16 -0.751924558 1.167949706 -2.105551275
17 -0.668719529 1.112479686 -2.005551275
18 -0.585514500 1.057009666 -1.905551275
19 -0.502309470 1.001539647 -1.805551275
20 -0.419104441 0.946069627 -1.705551275
21 -0.335899411 0.890599608 -1.605551275
22 -0.252694382 0.835129588 -1.505551275
23 -0.169489352 0.779659568 -1.405551275
24 -0.086284323 0.724189549 -1.305551275
25 -0.003079294 0.668719529 -1.205551275
26  0.080125736 0.613249509 -1.105551275
27  0.163330765 0.557779490 -1.005551275
28  0.246535795 0.502309470 -0.905551275
29  0.329740824 0.446839451 -0.805551275
30  0.412945854 0.391369431 -0.705551275
31  0.496150883 0.335899411 -0.605551275
32  0.579355912 0.280429392 -0.505551275
33  0.662560942 0.224959372 -0.405551275
34  0.745765971 0.169489352 -0.305551275
```

```
35  0.828971001  0.114019333 -0.205551275
36  0.912176030  0.058549313 -0.105551275
37  0.995381060  0.003079294 -0.005551275
38  1.078586089 -0.052390726 -0.094448725
```

As you see, we start from the initial estimate of (-2,2), and the steepest ascent algorithm took us to a location of approximately 1 and 0. In this case, it is stuck between two data points, (0.995,.003) and (1.079, -0.052), and starts going back and forth between these two data points without finding a final solution. Let's see how this journey to the top visually looks like.

If we start from another location, it will still find the way to the top. Below is an example when the starting position is $(x, y) = (4, 3)$.

For this example, it is straightforward to go to the top for the steepest ascent algorithm because the surface is not that complex and there is a single peak.

Another example of the Steepest Ascent

Now, let's see how the steepest ascent behaves for a more complicated surface. Consider the following function and its 3D representation. As you see, this function has a more complex surface. Instead of an obvious single peak, there were three of them. The highest peak (global maximum) was around the coordinates of $(x, y, z) = (2.03, 1.40, 1.00)$. The second highest peak (local maximum) was around the coordinates of $(x, y, z) = (3.00, 1.01, 0.89)$, and the third highest peak (local maximum) was around the coordinates of $(x, y, z) = (0.34, 1.43, 0.41)$.

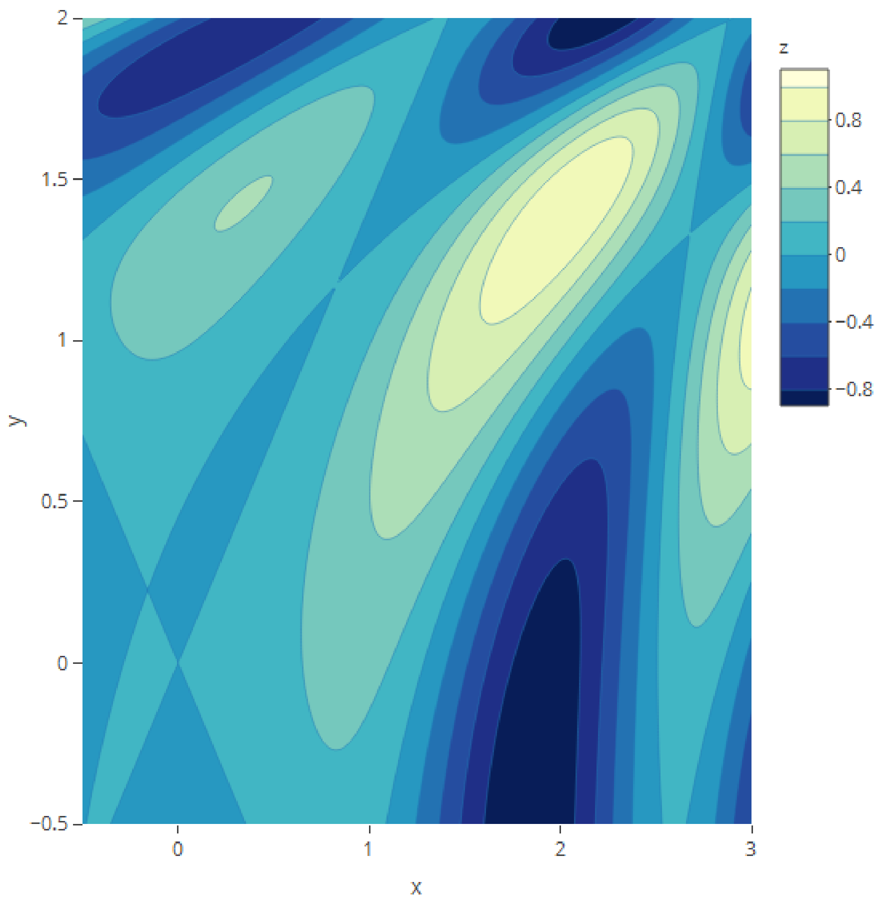
The code below implements the steepest ascent algorithm using the starting values of $(x, y) = (0.1, 0.3)$. After 134 iterations, the algorithm successfully finds the global maximum. As you will see, the path is not very straightforward, but the algorithm successfully finds the maximum point.

$$f(x, y) = sin(\frac{x^2}{2} - \frac{y^2}{4}) \times cos(2x - e^y)$$

```r
grid     <- expand.grid(x=seq(-.5,3,.01),y=seq(-.5,2,.01))
grid$z   <- sin(grid[,1]^2/2 - grid[,2]^2/4)*cos(2*grid[,1] - exp(grid[,2]))

plot_ly(grid,
        x = ~x,
        y = ~y,
        z = ~z,
        marker = list(color = ~z,
                      colorscale = 'YlGnBu',
                      showscale = FALSE)) %>%
  add_markers() %>%
  layout(scene = list(xaxis=list(range = c(3,-.5)),
                      yaxis=list(range = c(2,-.5))))
```

WebGL is not supported by your browser - visit https://get.webgl.org for more info

```
plot_ly(grid,
        x = ~x,
        y = ~y,
        z = ~z,
        type='contour',
        colors = 'YlGnBu',
        showscale=TRUE,
        reversescale =T)
```



```r
my.f                  <- function(x,y){
 sin(x^2/2 - y^2/4)*cos(2*x - exp(y))
}

f                     <- 'sin(x^2/2 - y^2/4)*cos(2*x - exp(y))'

iter                  <- data.frame(x=NA,y=NA,z=NA)
iter[1,c('x','y')] <- c(0.1,0.3)
iter[1,]$z            <- my.f(x=iter[1,1],y=iter[1,2])


change                <- 1
i                     <- 1

while((change > 0) & (i < 500)){

  der                 <- gradient(f = f,
                                  var = c(x=iter[i,1],y=iter[i,2]))
  i                   <- i + 1

  iter[i,1:2]         <- iter[i-1,1:2] + 0.1*der
  iter[i,]$z          <- my.f(x=iter[i,1],y=iter[i,2])

  change              <- iter[i,]$z - iter[i-1,]$z
  #cat('Iteration :', i, 'Value :', round(iter[i,]$z,4),'\n')
}

iter
```

```
           x          y            z
1   0.1000000 0.30000000 -0.0071504205
2   0.1008912 0.29602763 -0.0069813460
3   0.1020187 0.29194132 -0.0067977154
4   0.1034053 0.28773472 -0.0065968698
5   0.1050763 0.28340116 -0.0063756112
6   0.1070600 0.27893368 -0.0061300776
7   0.1093883 0.27432503 -0.0058555844
8   0.1120970 0.26956758 -0.0055464208
9   0.1152266 0.26465337 -0.0051955882
10  0.1188231 0.25957407 -0.0047944609
11  0.1229389 0.25432090 -0.0043323444
12  0.1276340 0.24888465 -0.0037958946
13  0.1329773 0.24325564 -0.0031683534
14  0.1390484 0.23742368 -0.0024285336
15  0.1459393 0.23137804 -0.0015494662
16  0.1537570 0.22510747 -0.0004965909
17  0.1626265 0.21860025  0.0007746738
18  0.1726937 0.21184429  0.0023221945
19  0.1841297 0.20482739  0.0042215033
20  0.1971347 0.19753767  0.0065716441
21  0.2119437 0.18996440  0.0095028879
22  0.2288305 0.18209930  0.0131867005
23  0.2481124 0.17393880  0.0178481404
24  0.2701521 0.16548759  0.0237802259
25  0.2953554 0.15676428  0.0313582972
26  0.3241596 0.14781027  0.0410492716
27  0.3570064 0.13870292  0.0534050090
28  0.3942899 0.12957472  0.0690203203
29  0.4362666 0.12063880  0.0884268234
30  0.4829214 0.11221948  0.1118938916
31  0.5337922 0.10478034  0.1391400984
32  0.5877966 0.09893417  0.1690491491
33  0.6431526 0.09541006  0.1996126398
34  0.6975181 0.09495643  0.2283448756
35  0.7484210 0.09818703  0.2531358538
36  0.7938628 0.10542572  0.2730422868
37  0.8328014 0.11663056  0.2884357842
38  0.8652565 0.13143840  0.3004837631
39  0.8920441 0.14929465  0.3104729053
40  0.9143642 0.16959458  0.3194073566
41  0.9334547 0.19178414  0.3279282823
42  0.9503955 0.21540901  0.3363969055
43  0.9660403 0.24012515  0.3450046742
44  0.9810237 0.26568858  0.3538551580
45  0.9958002 0.29193711  0.3630122202
46  1.0106884 0.31877134  0.3725247450
47  1.0259118 0.34613811  0.3824382444
```

48	1.0416297	0.37401718	0.3928000166
49	1.0579605	0.40241117	0.4036614191
50	1.0749980	0.43133813	0.4150789479
51	1.0928223	0.46082616	0.4271148454
52	1.1115073	0.49090949	0.4398374997
53	1.1311254	0.52162552	0.4533216861
54	1.1517500	0.55301243	0.4676485993
55	1.1734573	0.58510701	0.4829055548
56	1.1963267	0.61794243	0.4991851800
57	1.2204405	0.65154579	0.5165838411
58	1.2458823	0.68593507	0.5351989725
59	1.2727353	0.72111559	0.5551248792
60	1.3010781	0.75707548	0.5764464948
61	1.3309802	0.79378038	0.5992305253
62	1.3624948	0.83116728	0.6235134460
63	1.3956495	0.86913761	0.6492860416
64	1.4304347	0.90755028	0.6764747143
65	1.4667896	0.94621528	0.7049207571
66	1.5045863	0.98488928	0.7343602923
67	1.5436142	1.02327483	0.7644094691
68	1.5835664	1.06102504	0.7945613095
69	1.6240339	1.09775519	0.8242012626
70	1.6645099	1.13306188	0.8526467339
71	1.7044101	1.16654851	0.8792105744
72	1.7431088	1.19785437	0.9032802740
73	1.7799881	1.22668286	0.9243962788
74	1.8144933	1.25282418	0.9423090654
75	1.8461814	1.27616884	0.9569988933
76	1.8747561	1.29670962	0.9686538400
77	1.9000796	1.31453271	0.9776153357
78	1.9221643	1.32980031	0.9843091842
79	1.9411477	1.34272861	0.9891805364
80	1.9572583	1.35356521	0.9926451205
81	1.9707810	1.36256891	0.9950607697
82	1.9820254	1.36999383	0.9967168636
83	1.9913019	1.37607831	0.9978362518
84	1.9989051	1.38103825	0.9985840146
85	2.0051032	1.38506402	0.9990787084
86	2.0101337	1.38832001	0.9994033960
87	2.0142020	1.39094581	0.9996151315
88	2.0174826	1.39305844	0.9997524891
89	2.0201218	1.39475498	0.9998412211
90	2.0222409	1.39611531	0.9998983471
91	2.0239400	1.39720472	0.9999350249
92	2.0253006	1.39807631	0.9999585224
93	2.0263891	1.39877308	0.9999735496
94	2.0272593	1.39932975	0.9999831465
95	2.0279544	1.39977425	0.9999892684
96	2.0285094	1.40012906	0.9999931702
97	2.0289525	1.40041217	0.9999956552
98	2.0293060	1.40063802	0.9999972370
99	2.0295880	1.40081815	0.9999982434
100	2.0298129	1.40096180	0.9999988834
101	2.0299923	1.40107633	0.9999992904

```
102 2.0301353 1.40116764  0.9999995491
103 2.0302493 1.40124044  0.9999997135
104 2.0303402 1.40129846  0.9999998180
105 2.0304126 1.40134471  0.9999998844
106 2.0304704 1.40138158  0.9999999266
107 2.0305164 1.40141096  0.9999999533
108 2.0305531 1.40143438  0.9999999704
109 2.0305823 1.40145305  0.9999999812
110 2.0306056 1.40146792  0.9999999880
111 2.0306242 1.40147978  0.9999999924
112 2.0306390 1.40148922  0.9999999952
113 2.0306508 1.40149675  0.9999999969
114 2.0306602 1.40150276  0.9999999981
115 2.0306677 1.40150754  0.9999999988
116 2.0306736 1.40151135  0.9999999992
117 2.0306784 1.40151438  0.9999999995
118 2.0306822 1.40151681  0.9999999997
119 2.0306852 1.40151873  0.9999999998
120 2.0306876 1.40152028  0.9999999999
121 2.0306896 1.40152149  0.9999999999
122 2.0306911 1.40152248  0.9999999999
123 2.0306923 1.40152324  1.0000000000
124 2.0306933 1.40152389  1.0000000000
125 2.0306941 1.40152435  1.0000000000
126 2.0306946 1.40152478  1.0000000000
127 2.0306952 1.40152504  1.0000000000
128 2.0306955 1.40152536  1.0000000000
129 2.0306959 1.40152547  1.0000000000
130 2.0306961 1.40152575  1.0000000000
131 2.0306964 1.40152572  1.0000000000
132 2.0306964 1.40152602  1.0000000000
133 2.0306967 1.40152584  1.0000000000
134 2.0306966 1.40152625  1.0000000000
```

Limitations of steepest ascent . . . .

- slow convergence
- sensitivity to starting values for complex optimization problems

. . .

```r
grid     <- expand.grid(x=seq(-.5,3,.01),y=seq(-.5,2,.01))
grid$z   <- sin(grid[,1]^2/2 - grid[,2]^2/4)*cos(2*grid[,1] - exp(grid[,2]))

my.f                <- function(x,y){
 sin(x^2/2 - y^2/4)*cos(2*x - exp(y))
}

f                   <- 'sin(x^2/2 - y^2/4)*cos(2*x - exp(y))'

iter                <- data.frame(x=NA,y=NA,z=NA)
iter[1,c('x','y')] <- c(0,0.3)
iter[1,]$z          <- my.f(x=iter[1,1],y=iter[1,2])
```

```
change              <- 1
i                   <- 1

while((change > 0) & (i < 500)){

  der               <- gradient(f = f,
                                var = c(x=iter[i,1],y=iter[i,2]))
  i                 <- i + 1

  iter[i,1:2]       <- iter[i-1,1:2] + 0.1*der
  iter[i,]$z        <- my.f(x=iter[i,1],y=iter[i,2])

  change            <- iter[i,]$z - iter[i-1,]$z

  #cat('Iteration :', i, 'Value :', round(iter[i,]$z,4),'\n')
}

iter
```

```
               x         y             z
1    0.000000000 0.3000000 -0.00493033410
2   -0.004390245 0.2996768 -0.00473474333
3   -0.008869820 0.2994760 -0.00453189103
4   -0.013432708 0.2994015 -0.00432189564
5   -0.018071921 0.2994571 -0.00410496685
6   -0.022779525 0.2996466 -0.00388140599
7   -0.027546674 0.2999737 -0.00365160287
8   -0.032363668 0.3004418 -0.00341602880
9   -0.037220040 0.3010547 -0.00317522542
10  -0.042104646 0.3018159 -0.00292978939
11  -0.047005787 0.3027291 -0.00268035297
12  -0.051911341 0.3037979 -0.00242756092
13  -0.056808910 0.3050261 -0.00217204414
14  -0.061685978 0.3064176 -0.00191439077
15  -0.066530074 0.3079767 -0.00165511550
16  -0.071328940 0.3097079 -0.00139462780
17  -0.076070694 0.3116159 -0.00113320006
18  -0.080743989 0.3137061 -0.00087093606
19  -0.085338162 0.3159842 -0.00060774047
20  -0.089843370 0.3184568 -0.00034328942
21  -0.094250705 0.3211310 -0.00007700236
22  -0.098552295 0.3240149  0.00019198543
23  -0.102741384 0.3271175  0.00046485032
24  -0.106812386 0.3304489  0.00074310892
25  -0.110760926 0.3340205  0.00102864938
26  -0.114583852 0.3378450  0.00132376722
27  -0.118279238 0.3419368  0.00163120834
28  -0.121846358 0.3463122  0.00195422176
29  -0.125285661 0.3509892  0.00229662589
30  -0.128598719 0.3559884  0.00266289232
31  -0.131788177 0.3613329  0.00305825232
32  -0.134857694 0.3670487  0.00348883256
```

33	-0.137811874	0.3731652	0.00396182772
34	-0.140656203	0.3797155	0.00448572052
35	-0.143396982	0.3867373	0.00507056211
36	-0.146041259	0.3942732	0.00572833002
37	-0.148596762	0.4023718	0.00647338627
38	-0.151071830	0.4110882	0.00732306554
39	-0.153475341	0.4204857	0.00829843326
40	-0.155816623	0.4306365	0.00942526718
41	-0.158105354	0.4416239	0.01073533387
42	-0.160351414	0.4535437	0.01226805682
43	-0.162564689	0.4665067	0.01407270517
44	-0.164754771	0.4806415	0.01621127535
45	-0.166930508	0.4960975	0.01876229197
46	-0.169099317	0.5130487	0.02182581844
47	-0.171266118	0.5316980	0.02553003195
48	-0.173431691	0.5522815	0.03003975594
49	-0.175590114	0.5750738	0.03556728746
50	-0.177724795	0.6003910	0.04238555428
51	-0.179802298	0.6285929	0.05084275712
52	-0.181762818	0.6600782	0.06137553540
53	-0.183505608	0.6952691	0.07451313751
54	-0.184867214	0.7345742	0.09085627128
55	-0.185590313	0.7783130	0.11099982577
56	-0.185282787	0.8265798	0.13535210652
57	-0.183373253	0.8790248	0.16380648651
58	-0.179085493	0.9345586	0.19530425039
59	-0.171482441	0.9910786	0.22756306141
60	-0.159653115	1.0454843	0.25751289246
61	-0.143071947	1.0943341	0.28267334652
62	-0.121986051	1.1351355	0.30246560397
63	-0.097505812	1.1674229	0.31798871972
64	-0.071222741	1.1926105	0.33066561812
65	-0.044625466	1.2128253	0.34141362294
66	-0.018738692	1.2298660	0.35068425531
67	0.005898914	1.2448683	0.35871879945
68	0.029082751	1.2584368	0.36568636437
69	0.050784983	1.2708727	0.37172681871
70	0.071053886	1.2823351	0.37696134593
71	0.089964150	1.2929245	0.38149571905
72	0.107596523	1.3027170	0.38542223488
73	0.124030654	1.3117776	0.38882133028
74	0.139342891	1.3201645	0.39176302517
75	0.153605706	1.3279304	0.39430821379
76	0.166887592	1.3351233	0.39650981330
77	0.179253126	1.3417871	0.39841377951
78	0.190763079	1.3479622	0.40006000132
79	0.201474558	1.3536856	0.40148308559
80	0.211441164	1.3589911	0.40271304424
81	0.220713156	1.3639101	0.40377589474
82	0.229337620	1.3684713	0.40469418402
83	0.237358633	1.3727012	0.40548744524
84	0.244817430	1.3766244	0.40617259559
85	0.251752564	1.3802634	0.40676428243
86	0.258200061	1.3836391	0.40727518446

87	0.264193570	1.3867707	0.40771627348
88	0.269764502	1.3896761	0.40809704187
89	0.274942174	1.3923718	0.40842570010
90	0.279753931	1.3948731	0.40870934830
91	0.284225274	1.3971940	0.40895412507
92	0.288379976	1.3993477	0.40916533664
93	0.292240189	1.4013462	0.40934756880
94	0.295826557	1.4032009	0.40950478405
95	0.299158303	1.4049221	0.40964040567
96	0.302253332	1.4065194	0.40975739061
97	0.305128311	1.4080019	0.40985829263
98	0.307798757	1.4093778	0.40994531695
99	0.310279106	1.4106547	0.41002036756
100	0.312582796	1.4118399	0.41008508811
101	0.314722323	1.4129399	0.41014089739
102	0.316709314	1.4139608	0.41018901991
103	0.318554581	1.4149084	0.41023051240
104	0.320268179	1.4157880	0.41026628670
105	0.321859457	1.4166043	0.41029712958
106	0.323337106	1.4173621	0.41032371978
107	0.324709205	1.4180654	0.41034664284
108	0.325983263	1.4187182	0.41036640384
109	0.327166260	1.4193242	0.41038343843
110	0.328264683	1.4198866	0.41039812234
111	0.329284558	1.4204087	0.41041077960
112	0.330231485	1.4208933	0.41042168965
113	0.331110669	1.4213431	0.41043109346
114	0.331926942	1.4217607	0.41043919879
115	0.332684797	1.4221482	0.41044618480
116	0.333388404	1.4225080	0.41045220595
117	0.334041639	1.4228419	0.41045739540
118	0.334648102	1.4231519	0.41046186797
119	0.335211134	1.4234396	0.41046572263
120	0.335733842	1.4237066	0.41046904469
121	0.336219109	1.4239545	0.41047190771
122	0.336669613	1.4241847	0.41047437509
123	0.337087841	1.4243983	0.41047650148
124	0.337476103	1.4245965	0.41047833398
125	0.337836543	1.4247806	0.41047991320
126	0.338171153	1.4249514	0.41048127412
127	0.338481782	1.4251100	0.41048244693
128	0.338770148	1.4252572	0.41048345760
129	0.339037844	1.4253939	0.41048432855
130	0.339286351	1.4255207	0.41048507909
131	0.339517044	1.4256385	0.41048572587
132	0.339731199	1.4257478	0.41048628321
133	0.339930000	1.4258492	0.41048676350
134	0.340114548	1.4259434	0.41048717737
135	0.340285864	1.4260308	0.41048753402
136	0.340444896	1.4261120	0.41048784135
137	0.340592525	1.4261873	0.41048810618
138	0.340729568	1.4262572	0.41048833439
139	0.340856783	1.4263221	0.41048853104
140	0.340974876	1.4263823	0.41048870049

141	0.341084500	1.4264383	0.41048884651
142	0.341186263	1.4264902	0.41048897234
143	0.341280727	1.4265384	0.41048908076
144	0.341368417	1.4265831	0.41048917419
145	0.341449818	1.4266246	0.41048925470
146	0.341525380	1.4266632	0.41048932407
147	0.341595523	1.4266990	0.41048938385
148	0.341660635	1.4267322	0.41048943536
149	0.341721077	1.4267630	0.41048947974
150	0.341777184	1.4267916	0.41048951799
151	0.341829267	1.4268182	0.41048955095
152	0.341877614	1.4268428	0.41048957935
153	0.341922493	1.4268657	0.41048960382
154	0.341964153	1.4268870	0.41048962490
155	0.342002824	1.4269067	0.41048964307
156	0.342038722	1.4269250	0.41048965873
157	0.342072045	1.4269420	0.41048967222
158	0.342102978	1.4269578	0.41048968384
159	0.342131691	1.4269724	0.41048969386
160	0.342158345	1.4269860	0.41048970249
161	0.342183087	1.4269986	0.41048970993
162	0.342206055	1.4270103	0.41048971634
163	0.342227375	1.4270212	0.41048972186
164	0.342247165	1.4270313	0.41048972662
165	0.342265536	1.4270407	0.41048973072
166	0.342282589	1.4270494	0.41048973425
167	0.342298419	1.4270574	0.41048973729
168	0.342313113	1.4270649	0.41048973992
169	0.342326753	1.4270719	0.41048974218
170	0.342339415	1.4270784	0.41048974412
171	0.342351168	1.4270843	0.41048974580
172	0.342362078	1.4270899	0.41048974725
173	0.342372206	1.4270951	0.41048974850
174	0.342381607	1.4270999	0.41048974957
175	0.342390334	1.4271043	0.41048975049
176	0.342398434	1.4271084	0.41048975129
177	0.342405954	1.4271123	0.41048975198
178	0.342412934	1.4271158	0.41048975257
179	0.342419414	1.4271191	0.41048975308
180	0.342425428	1.4271222	0.41048975352
181	0.342431011	1.4271251	0.41048975390
182	0.342436194	1.4271277	0.41048975422
183	0.342441005	1.4271302	0.41048975451
184	0.342445471	1.4271324	0.41048975475
185	0.342449616	1.4271345	0.41048975496
186	0.342453464	1.4271365	0.41048975514
187	0.342457036	1.4271383	0.41048975529
188	0.342460352	1.4271400	0.41048975543
189	0.342463429	1.4271416	0.41048975554
190	0.342466286	1.4271431	0.41048975564
191	0.342468939	1.4271444	0.41048975572
192	0.342471400	1.4271457	0.41048975580
193	0.342473686	1.4271468	0.41048975586
194	0.342475807	1.4271479	0.41048975592

195	0.342477776	1.4271489	0.41048975596
196	0.342479604	1.4271498	0.41048975600
197	0.342481301	1.4271507	0.41048975604
198	0.342482876	1.4271515	0.41048975607
199	0.342484338	1.4271523	0.41048975610
200	0.342485695	1.4271529	0.41048975612
201	0.342486955	1.4271536	0.41048975614
202	0.342488124	1.4271542	0.41048975615
203	0.342489210	1.4271547	0.41048975617
204	0.342490217	1.4271553	0.41048975618
205	0.342491153	1.4271557	0.41048975619
206	0.342492021	1.4271562	0.41048975620
207	0.342492827	1.4271566	0.41048975621
208	0.342493575	1.4271570	0.41048975621
209	0.342494270	1.4271573	0.41048975622
210	0.342494914	1.4271576	0.41048975623
211	0.342495513	1.4271580	0.41048975623
212	0.342496068	1.4271582	0.41048975623
213	0.342496584	1.4271585	0.41048975624
214	0.342497063	1.4271587	0.41048975624
215	0.342497507	1.4271590	0.41048975624
216	0.342497919	1.4271592	0.41048975624
217	0.342498302	1.4271594	0.41048975625
218	0.342498658	1.4271596	0.41048975625
219	0.342498987	1.4271597	0.41048975625
220	0.342499294	1.4271599	0.41048975625
221	0.342499578	1.4271600	0.41048975625
222	0.342499842	1.4271602	0.41048975625
223	0.342500087	1.4271603	0.41048975625
224	0.342500314	1.4271604	0.41048975625
225	0.342500525	1.4271605	0.41048975625
226	0.342500721	1.4271606	0.41048975625
227	0.342500903	1.4271607	0.41048975625
228	0.342501072	1.4271608	0.41048975626
229	0.342501228	1.4271609	0.41048975626
230	0.342501374	1.4271609	0.41048975626
231	0.342501509	1.4271610	0.41048975626
232	0.342501634	1.4271611	0.41048975626
233	0.342501751	1.4271611	0.41048975626
234	0.342501859	1.4271612	0.41048975626
235	0.342501959	1.4271612	0.41048975626
236	0.342502052	1.4271613	0.41048975626
237	0.342502138	1.4271613	0.41048975626
238	0.342502219	1.4271614	0.41048975626
239	0.342502293	1.4271614	0.41048975626
240	0.342502362	1.4271614	0.41048975626
241	0.342502426	1.4271615	0.41048975626
242	0.342502486	1.4271615	0.41048975626
243	0.342502541	1.4271615	0.41048975626
244	0.342502592	1.4271616	0.41048975626
245	0.342502640	1.4271616	0.41048975626
246	0.342502684	1.4271616	0.41048975626
247	0.342502725	1.4271616	0.41048975626
248	0.342502763	1.4271617	0.41048975626

```
249  0.342502799 1.4271617  0.41048975626
250  0.342502831 1.4271617  0.41048975626
251  0.342502862 1.4271617  0.41048975626
252  0.342502890 1.4271617  0.41048975626
253  0.342502916 1.4271617  0.41048975626
254  0.342502941 1.4271617  0.41048975626
255  0.342502963 1.4271618  0.41048975626
256  0.342502984 1.4271618  0.41048975626
257  0.342503004 1.4271618  0.41048975626
258  0.342503022 1.4271618  0.41048975626
259  0.342503039 1.4271618  0.41048975626
260  0.342503054 1.4271618  0.41048975626
261  0.342503069 1.4271618  0.41048975626
262  0.342503082 1.4271618  0.41048975626
263  0.342503095 1.4271618  0.41048975626
264  0.342503106 1.4271618  0.41048975626
265  0.342503117 1.4271618  0.41048975626
266  0.342503127 1.4271618  0.41048975626
267  0.342503136 1.4271618  0.41048975626
268  0.342503145 1.4271618  0.41048975626
269  0.342503153 1.4271619  0.41048975626
270  0.342503160 1.4271619  0.41048975626
271  0.342503167 1.4271619  0.41048975626
272  0.342503174 1.4271619  0.41048975626
273  0.342503180 1.4271619  0.41048975626
274  0.342503185 1.4271619  0.41048975626
275  0.342503190 1.4271619  0.41048975626
276  0.342503195 1.4271619  0.41048975626
277  0.342503199 1.4271619  0.41048975626
278  0.342503203 1.4271619  0.41048975626
279  0.342503207 1.4271619  0.41048975626
280  0.342503211 1.4271619  0.41048975626
281  0.342503214 1.4271619  0.41048975626
282  0.342503217 1.4271619  0.41048975626
283  0.342503220 1.4271619  0.41048975626
284  0.342503222 1.4271619  0.41048975626
285  0.342503225 1.4271619  0.41048975626
```

Newton Raphson algorithm

Other optimization algorithms

- R function `optimize()`
- Nelder-Mead
- Quasi-newton
- conjugate gradient
- simulated annealing
- genetic algorithm