Root Finding and Coordinate Descent

Yuki Kitamura

2024-07-16

This is a project focuses on root finding and coordinate descent algorithm. In numerical analysis, a root-finding algorithm is an algorithm for finding zeros, also called "roots", of continuous functions. A zero of a function f, from the real numbers to real numbers or from the complex numbers to the complex numbers, is a number x such that f(x) = 0. I explored four root findings methods: fixed point iteration, Newton Raphson, Secant Method, and Bisection Method.

Coordinate descent is an optimization algorithm. The algorithm attempts to find a local minimum of a function. We perform a search in one direction to find the value that minimizes the function in that direction while the other values are held constant. Once the value for that direction is updated, The code perform the same operation for the other coordinate directions. This repeats until it has been updated for all coordinate directions, at which point the cycle repeats. I used the Golden Section Search Method to do the minimization for the coordinate descent.

Root Finding with Fixed Point Iteration

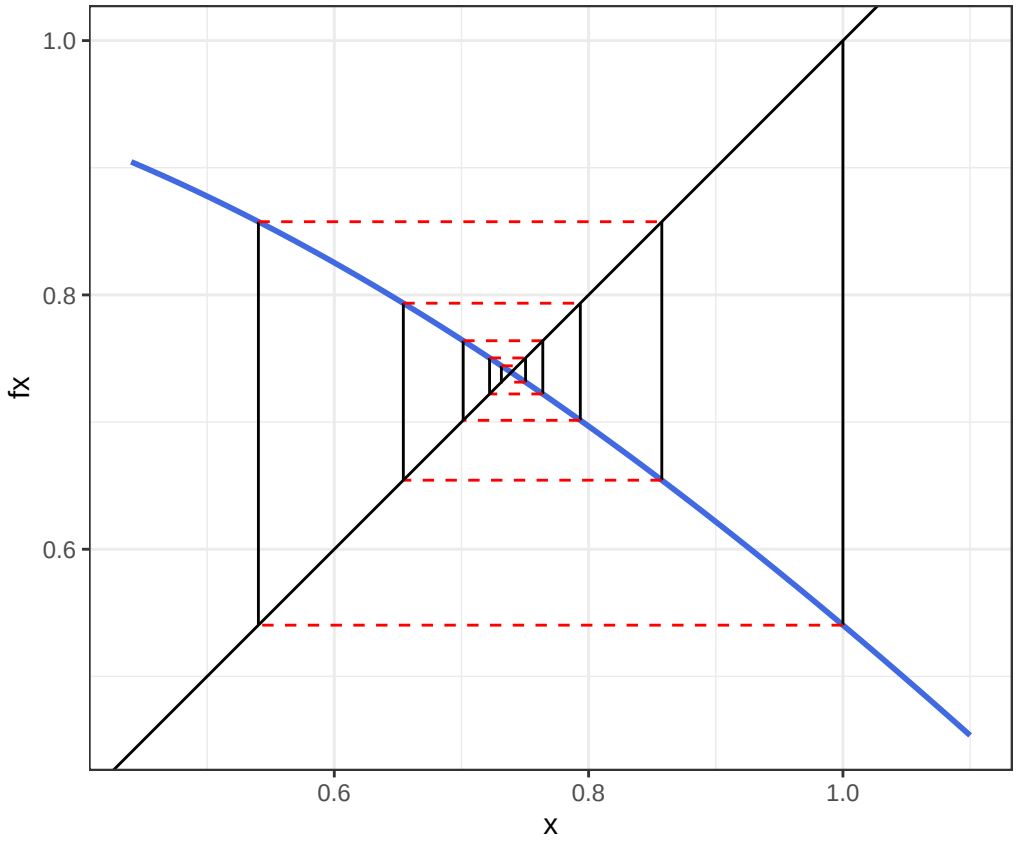
```
library(ggplot2)
fixedpoint_show <- function(ftn, x0, iter = 5){
  # applies fixed-point method to find x such that ftn(x) = x
  # ftn is a user-defined function

  # df_points_1 and df_points_2 are used to track each update
  # it will be used to plot the line segments showing each update
  # each line segment connects the points (x1, y1) to (x2, y2)
  df_points_1 <- data.frame(
    x1 = numeric(0),
    y1 = numeric(0),
    x2 = numeric(0),
    y2 = numeric(0))
  df_points_2 <- df_points_1

  xnew <- x0
  cat("Starting value is:", xnew, "\n")

  # iterate the fixed point algorithm
  for (i in 1:iter) {
    xold <- xnew
    xnew <- ftn(xold)
    cat("Next value of x is:", xnew, "\n")
    # vertical line segments, where x1 = x2
    df_points_1[i, ] <- c(x1 = xold, y1 = xold, x2 = xold, y2 = xnew)
```

```
    # horizontal line segments, where y1 = y2
    df_points_2[i, ] <- c(x1 = xold, y1 = xnew, x2 = xnew, y2 = xnew)
  }

  # use ggplot to plot the function and the segments for each iteration
  # determine the limits to use for the plot
  # start is the min of these values. we subtract .1 to provide a small margin
  plot_start <- min(df_points_1$x1, df_points_1$x2, x0) - 0.1
  # end is the max of these values
  plot_end <- max(df_points_1$x1, df_points_1$x2, x0) + 0.1

  # calculate the value of the funtion fx for all x
  x <- seq(plot_start, plot_end, length.out = 200)
  fx <- rep(NA, length(x))
  for (i in seq_along(x)) {
    fx[i] <- ftn(x[i])
  }
  function_data <- data.frame(x, fx) # data frame containing the function values

  p <- ggplot(function_data, aes(x = x, y = fx)) +
    geom_line(color = "royalblue", linewidth = 1) +  # plot the function
    geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2),
                 data = df_points_1, color = "black", lty = 1) +
    geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2),
                 data = df_points_2, color = "red", lty = 2) +
    geom_abline(intercept = 0, slope = 1) + # plot the line y = x
    coord_equal() + theme_bw()

  print(p) # produce the plot
  xnew # value that gets returned
}
```

Find the value of x when f(x) = cos(x) using x0 = 1

```
f <- function(x) cos(x)
fixedpoint_show(f, 1, iter = 10)
```

```
## Starting value is: 1
## Next value of x is: 0.5403023
## Next value of x is: 0.8575532
## Next value of x is: 0.6542898
## Next value of x is: 0.7934804
## Next value of x is: 0.7013688
## Next value of x is: 0.7639597
## Next value of x is: 0.7221024
## Next value of x is: 0.7504178
## Next value of x is: 0.731404
## Next value of x is: 0.7442374
```

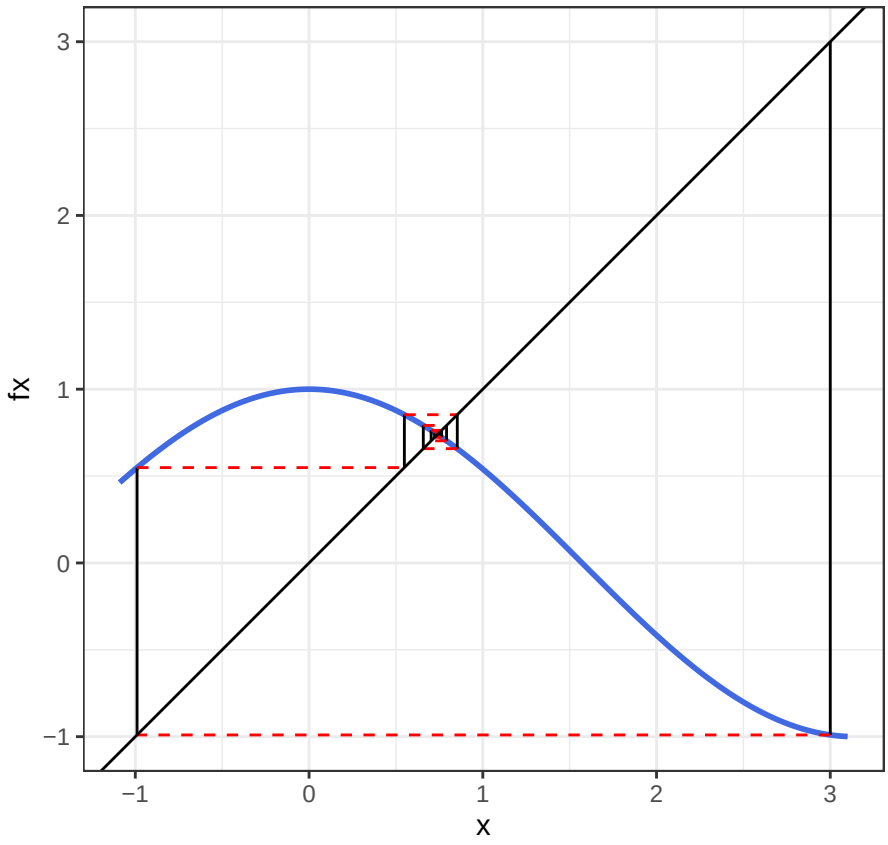



```
## [1] 0.7442374
```

Find the value of x when f(x) = cos(x) using x0 = 3

```
fixedpoint_show(f, 3, iter= 10)
```

```
## Starting value is: 3
## Next value of x is: -0.9899925
## Next value of x is: 0.5486961
## Next value of x is: 0.8532053
## Next value of x is: 0.6575717
## Next value of x is: 0.7914787
## Next value of x is: 0.7027941
## Next value of x is: 0.7630392
## Next value of x is: 0.7227389
## Next value of x is: 0.7499969
## Next value of x is: 0.731691
```

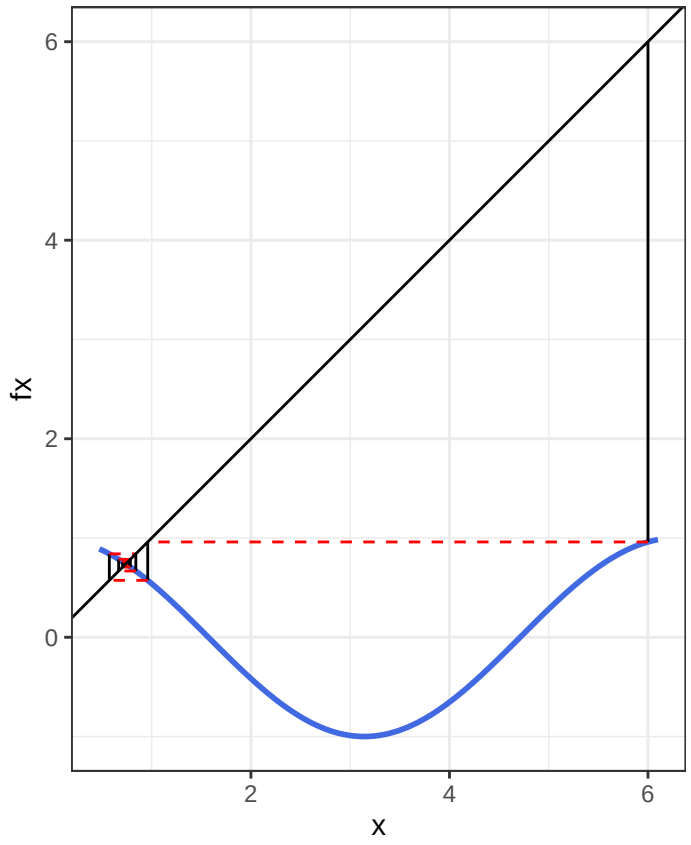



```
## [1] 0.731691
```

Find the value of x when f(x) = cos(x) using x0 = 6

```
fixedpoint_show(f, 6, iter= 10)
```

```
## Starting value is: 6
## Next value of x is: 0.9601703
## Next value of x is: 0.5733805
## Next value of x is: 0.840072
## Next value of x is: 0.6674092
## Next value of x is: 0.7854279
## Next value of x is: 0.7070858
## Next value of x is: 0.7602582
## Next value of x is: 0.7246581
## Next value of x is: 0.7487261
## Next value of x is: 0.7325566
```

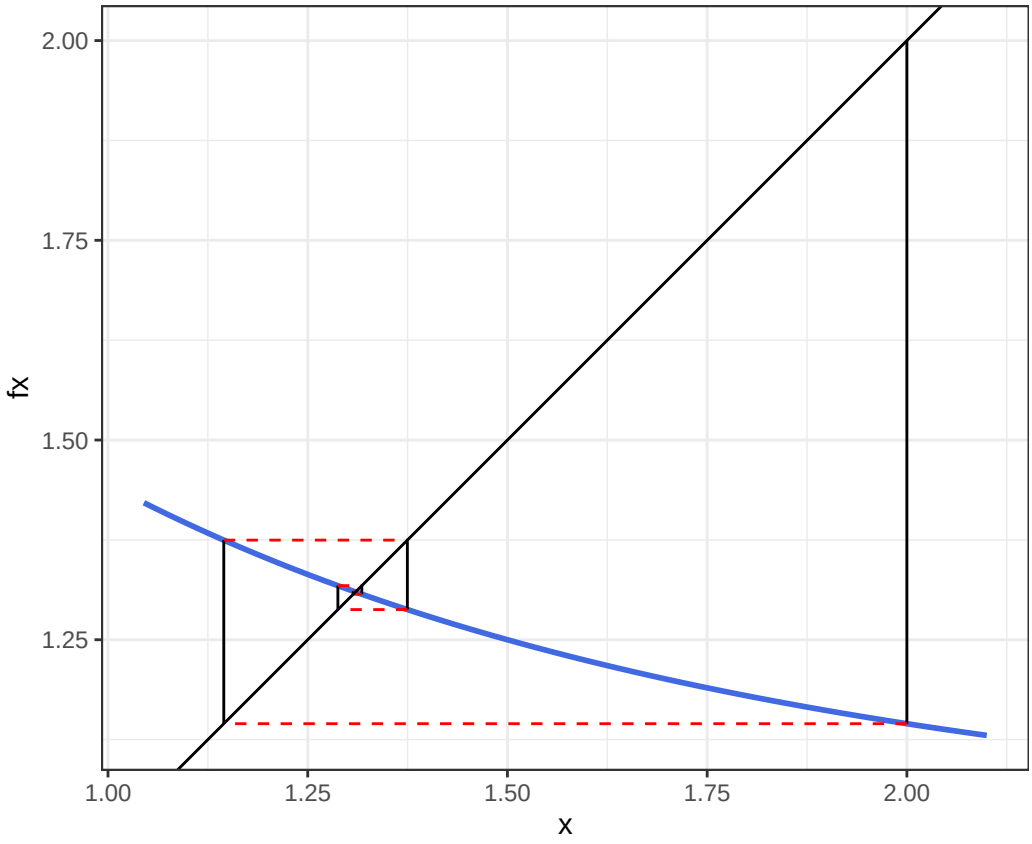



```
## [1] 0.7325566
```

Find the value of x when f(x) = exp(exp(-x)) using x0 = 2

```
f_b <- function(x) {exp(exp(-x))}
fixedpoint_show(f_b, 2, iter = 10)
```

```
## Starting value is: 2
## Next value of x is: 1.144921
## Next value of x is: 1.374719
## Next value of x is: 1.287768
## Next value of x is: 1.317697
## Next value of x is: 1.307022
## Next value of x is: 1.310783
## Next value of x is: 1.309452
## Next value of x is: 1.309922
## Next value of x is: 1.309756
## Next value of x is: 1.309815
```

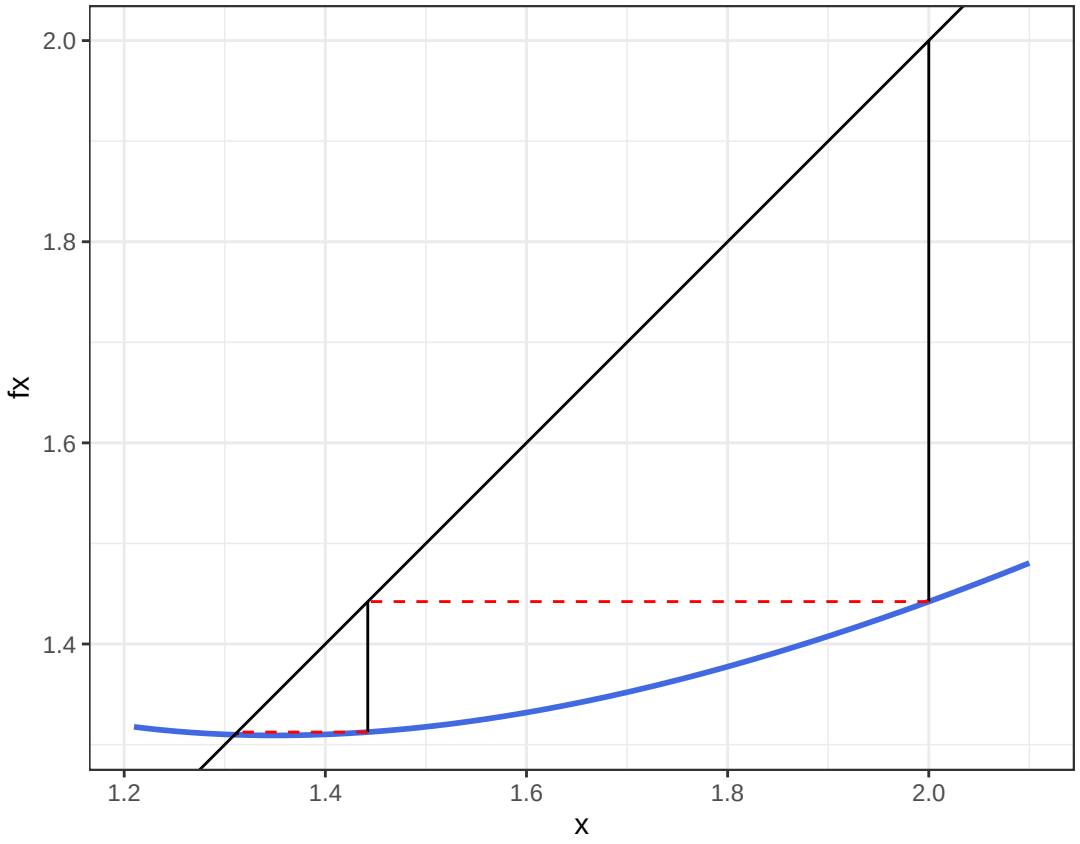



```
## [1] 1.309815
```

Find the value of x when f(x) = x - log(x) + exp(-x) using x0 = 2

```
f_c <- function(x) {x - log(x) + exp(-x)}
fixedpoint_show(f_c, 2, iter = 10)
```

```
## Starting value is: 2
## Next value of x is: 1.442188
## Next value of x is: 1.312437
## Next value of x is: 1.309715
## Next value of x is: 1.309802
## Next value of x is: 1.309799
## Next value of x is: 1.3098
## Next value of x is: 1.3098
## Next value of x is: 1.3098
## Next value of x is: 1.3098
## Next value of x is: 1.3098
```

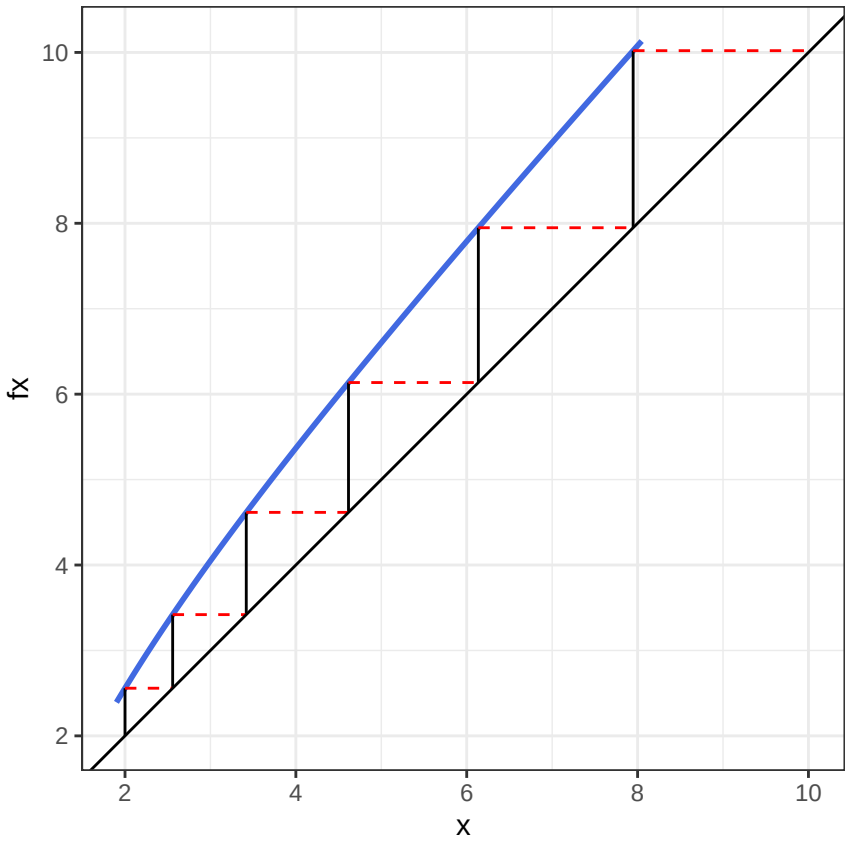



```
## [1] 1.3098
```

Find the value of x when f(x) = x + log(x) - exp(-x) using x0 = 2, with 6 iterations

```
f_d <- function(x) {x + log(x) - exp(-x)}
fixedpoint_show(f_d, 2, iter = 6)
```

```
## Starting value is: 2
## Next value of x is: 2.557812
## Next value of x is: 3.41949
## Next value of x is: 4.616252
## Next value of x is: 6.135946
## Next value of x is: 7.947946
## Next value of x is: 10.02051
```




```
## [1] 10.02051
```

Root Finding with Newton Raphson

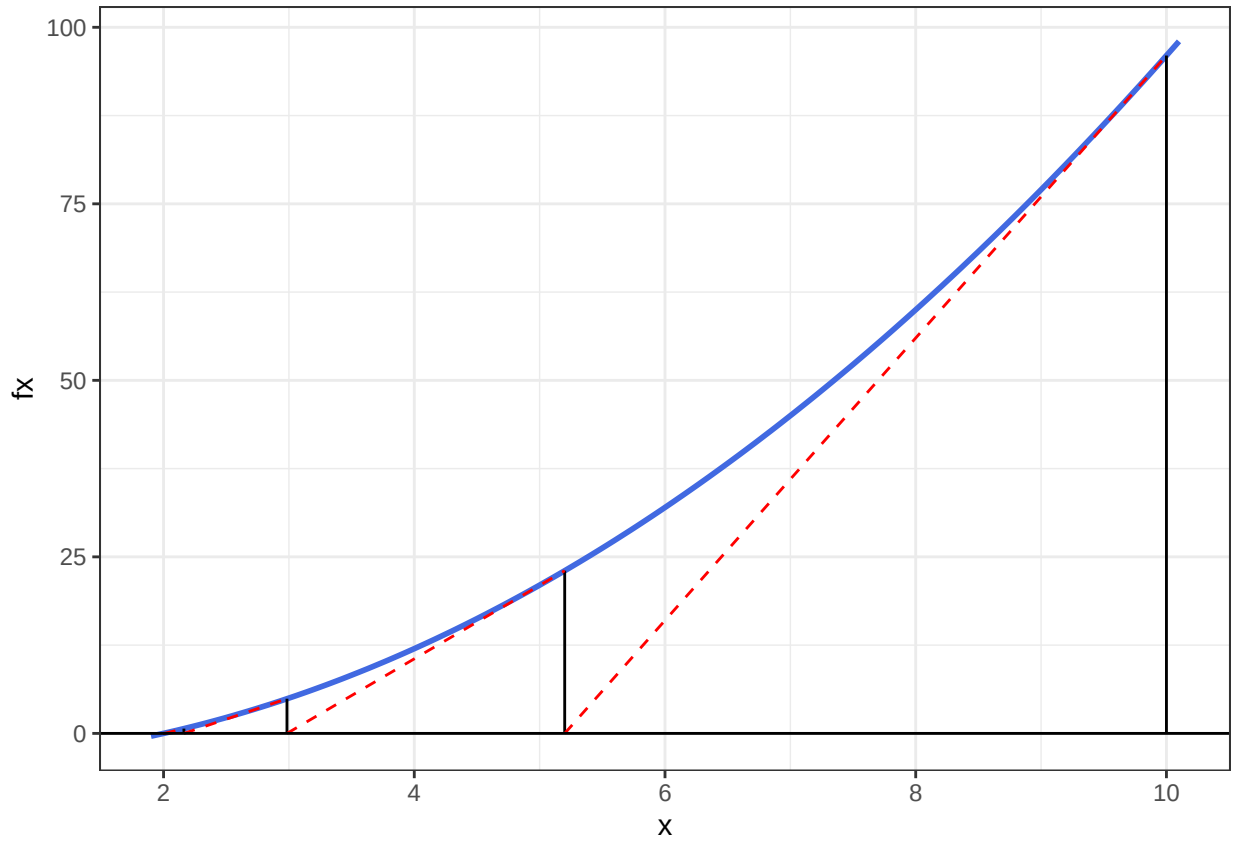
```
newtonraphson_show <- function(ftn, x0, iter = 5) {
  # applies Newton-Raphson to find x such that ftn(x)[1] == 0
  # ftn is a function of x. it returns two values, f(x) and f'(x)
  # x0 is the starting point

  # df_points_1 and df_points_2 are used to track each update
    df_points_1 <- data.frame(
    x1 = numeric(0),
    y1 = numeric(0),
    x2 = numeric(0),
    y2 = numeric(0))
  df_points_2 <- df_points_1
  xnew <- x0
  cat("Starting value is:", xnew, "\n")

  # the algorithm
  for(i in 1:iter){
    xold <- xnew
    f_xold <- ftn(xold)
    xnew <- xold - f_xold[1]/f_xold[2]
    cat("Next x value:", xnew, "\n")

    # the line segments
    df_points_1[i,] <- c(x1 = xold, y1 = 0, x2 = xold, y2 = f_xold[1]) # vertical segment
    df_points_2[i,] <- c(x1 = xnew, y1 = 0, x2 = xold, y2 = f_xold[1]) # tangent segment
  }

  plot_start <- min(df_points_1$x1, df_points_1$x2, x0) - 0.1 # Find min x value
  plot_end <- max(df_points_1$x1, df_points_1$x2, x0) + 0.1 # Find max x value

  # calculate the value of the funtion fx for all x
  x <- seq(plot_start, plot_end, length.out = 200)
  fx <- rep(NA, length(x))
  for (i in seq_along(x)) {
    fx[i] <- ftn(x[i])[1]
  }
  function_data <- data.frame(x, fx) # data frame containing the function values

  p <- ggplot(function_data, aes(x = x, y = fx)) +
    geom_line(color = "royalblue", linewidth = 1) +  # plot the function
    geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2),
                 data = df_points_1, color = "black", lty = 1) +
    geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2),
                 data = df_points_2, color = "red", lty = 2) +
    geom_abline(intercept = 0, slope = 0) + # plot the line y = 0
    theme_bw()

  print(p) # produce the plot
  xnew # value that gets returned
}
```

Produce graphs for:

The function f(x) = x^2 - 4 using x0 = 10

```
f_z <- function(x){
  value <- x^2 - 4   # f(x)
  derivative <- 2*x  # f'(x)
  c(value, derivative) # the function returns a vector with two values
}
newtonraphson_show(f_z, 10, iter = 8)
```

```
## Starting value is: 10
## Next x value: 5.2
## Next x value: 2.984615
## Next x value: 2.162411
## Next x value: 2.006099
## Next x value: 2.000009
## Next x value: 2
## Next x value: 2
## Next x value: 2
```



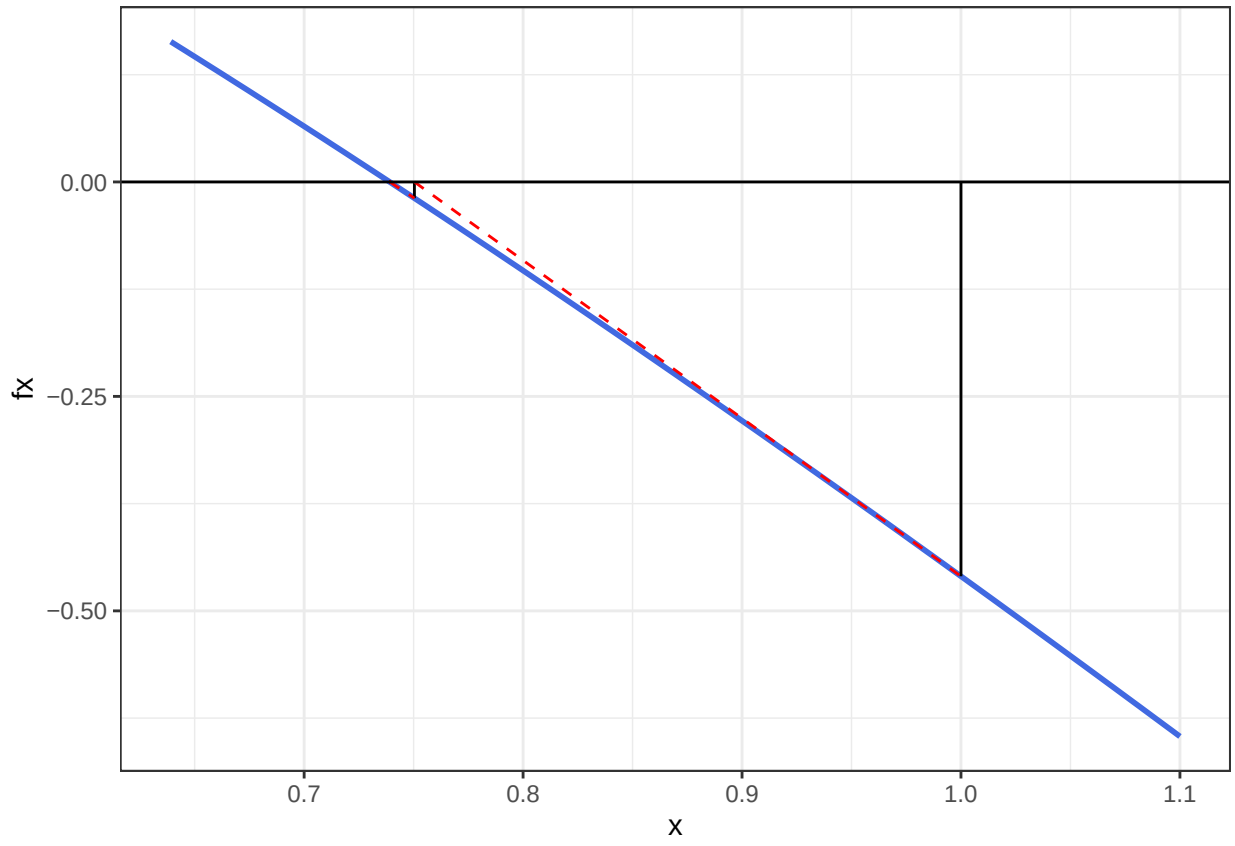

```
## [1] 2
```

The function f(x) = cos(x) - x using x0 = 1, 3, 6

Results should be similar to finding fixed point of cos(x)

```
f_a <- function(x){
  value <- cos(x) - x   # f(x)
  derivative <- -sin(x) - 1 # f'(x)
  c(value, derivative) # the function returns a vector with two values
}
newtonraphson_show(f_a, 1, iter = 8)
```

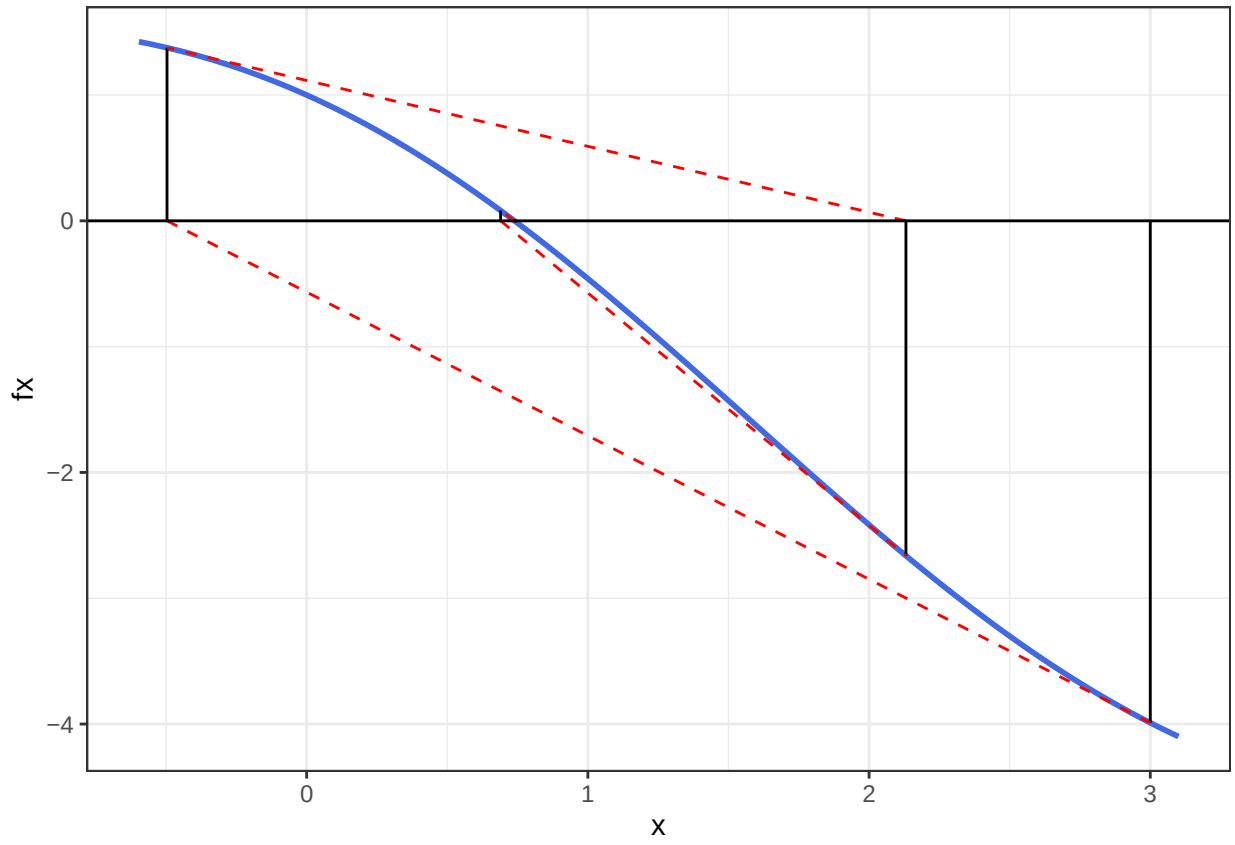
```
## Starting value is: 1
## Next x value: 0.7503639
## Next x value: 0.7391129
## Next x value: 0.7390851
## Next x value: 0.7390851
## Next x value: 0.7390851
## Next x value: 0.7390851
## Next x value: 0.7390851
## Next x value: 0.7390851
```




```
## [1] 0.7390851
```

```
newtonraphson_show(f_a, 3, iter = 8)
```

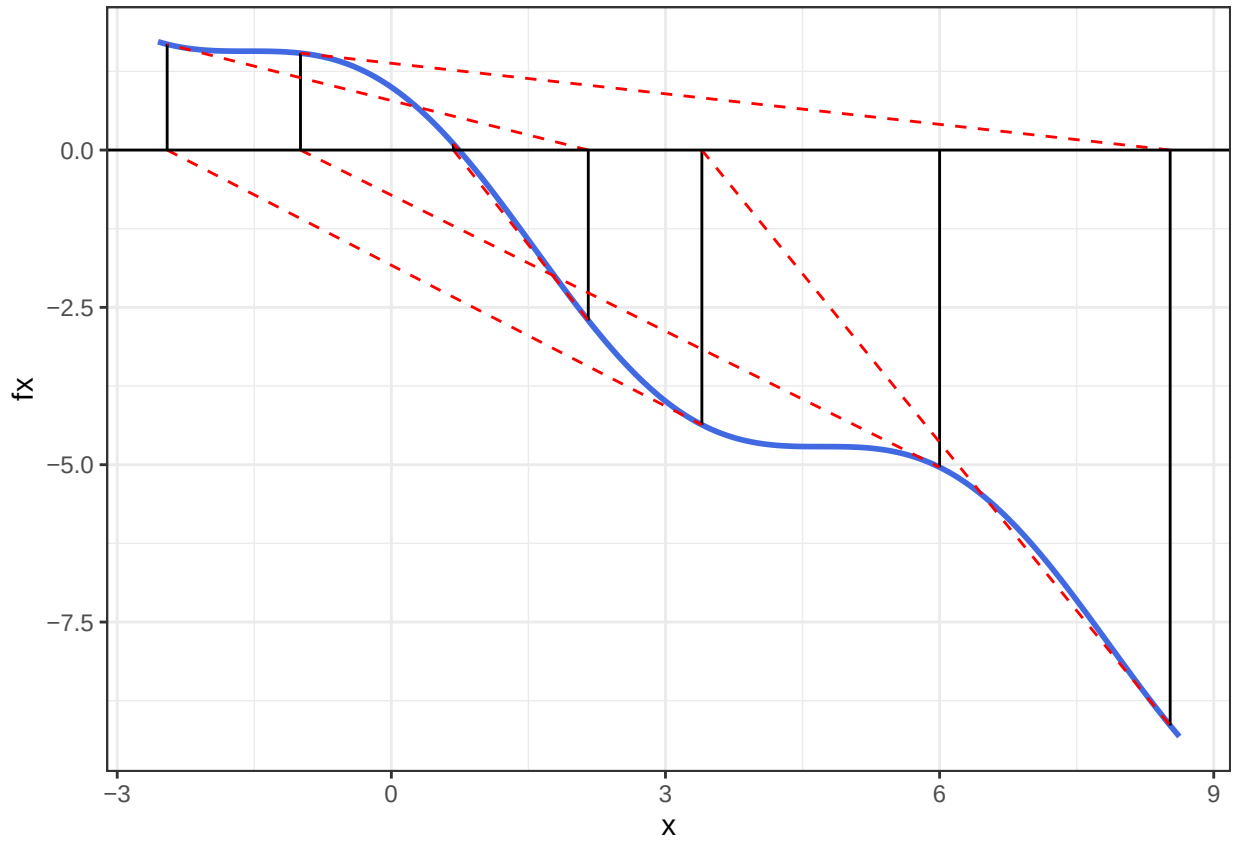
```
## Starting value is: 3
## Next x value: -0.4965582
## Next x value: 2.131004
## Next x value: 0.6896627
## Next x value: 0.739653
## Next x value: 0.7390852
## Next x value: 0.7390851
## Next x value: 0.7390851
## Next x value: 0.7390851
```




```
## [1] 0.7390851
```

```
newtonraphson_show(f_a, 6, iter = 8)
```

```
## Starting value is: 6
## Next x value: -0.9940856
## Next x value: 8.523426
## Next x value: 3.398358
## Next x value: -2.45325
## Next x value: 2.155349
## Next x value: 0.6792118
## Next x value: 0.7399276
## Next x value: 0.7390853
```



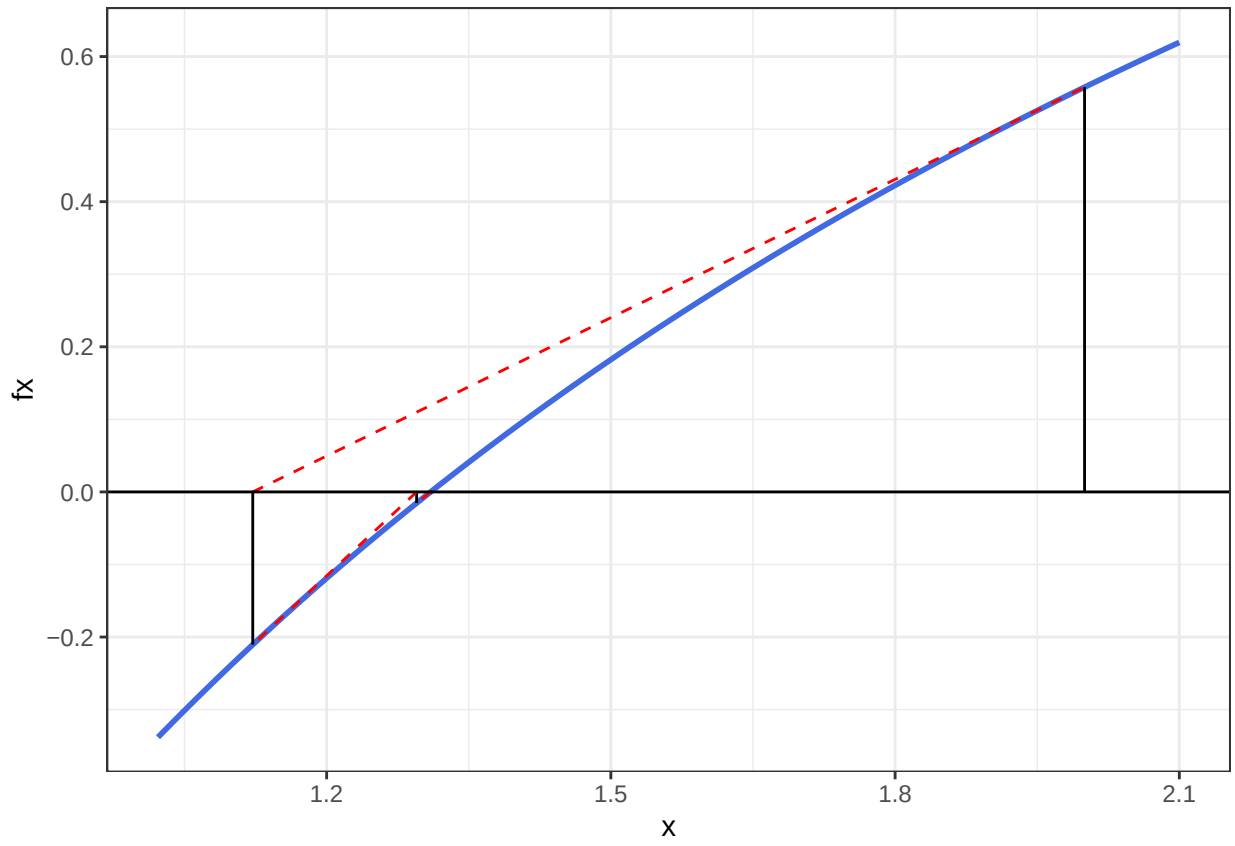

```
## [1] 0.7390853
```

The function f(x) = log(x) - exp(-x) using x0 = 2

Results should be similar to finding fixed point of exp(exp(-x))

```
f_b <- function(x){
  value <- log(x) - exp(-x)    # f(x)
  derivative <- (1 / x) + exp(-x) # f'(x)
  c(value, derivative) # the function returns a vector with two values
}
newtonraphson_show(f_b, 2, iter = 8)
```

```
## Starting value is: 2
## Next x value: 1.12202
## Next x value: 1.294997
## Next x value: 1.309709
## Next x value: 1.3098
## Next x value: 1.3098
## Next x value: 1.3098
## Next x value: 1.3098
## Next x value: 1.3098
```

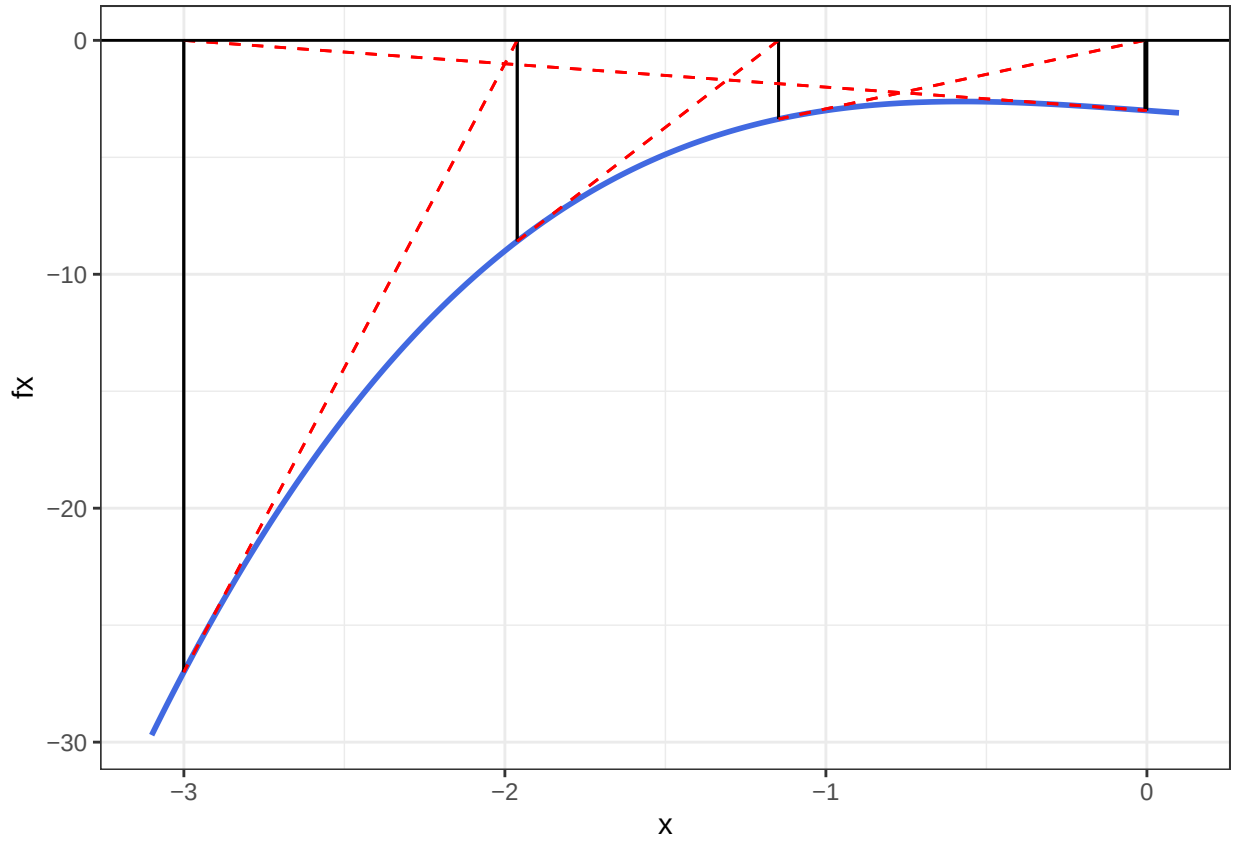



```
## [1] 1.3098
```

The function f(x) = x^3 - x - 3 using x0 = 0

```
f_3c <- function(x){
  value <- x^3 - x - 3 # f(x)
  derivative <- 3*x^2 - 1 # f'(x)
  c(value, derivative) # the function returns a vector with two values
}
newtonraphson_show(f_3c, 0, iter = 8)
```

```
## Starting value is: 0
## Next x value: -3
## Next x value: -1.961538
## Next x value: -1.147176
## Next x value: -0.006579371
## Next x value: -3.000389
## Next x value: -1.961818
## Next x value: -1.14743
## Next x value: -0.007256248
```

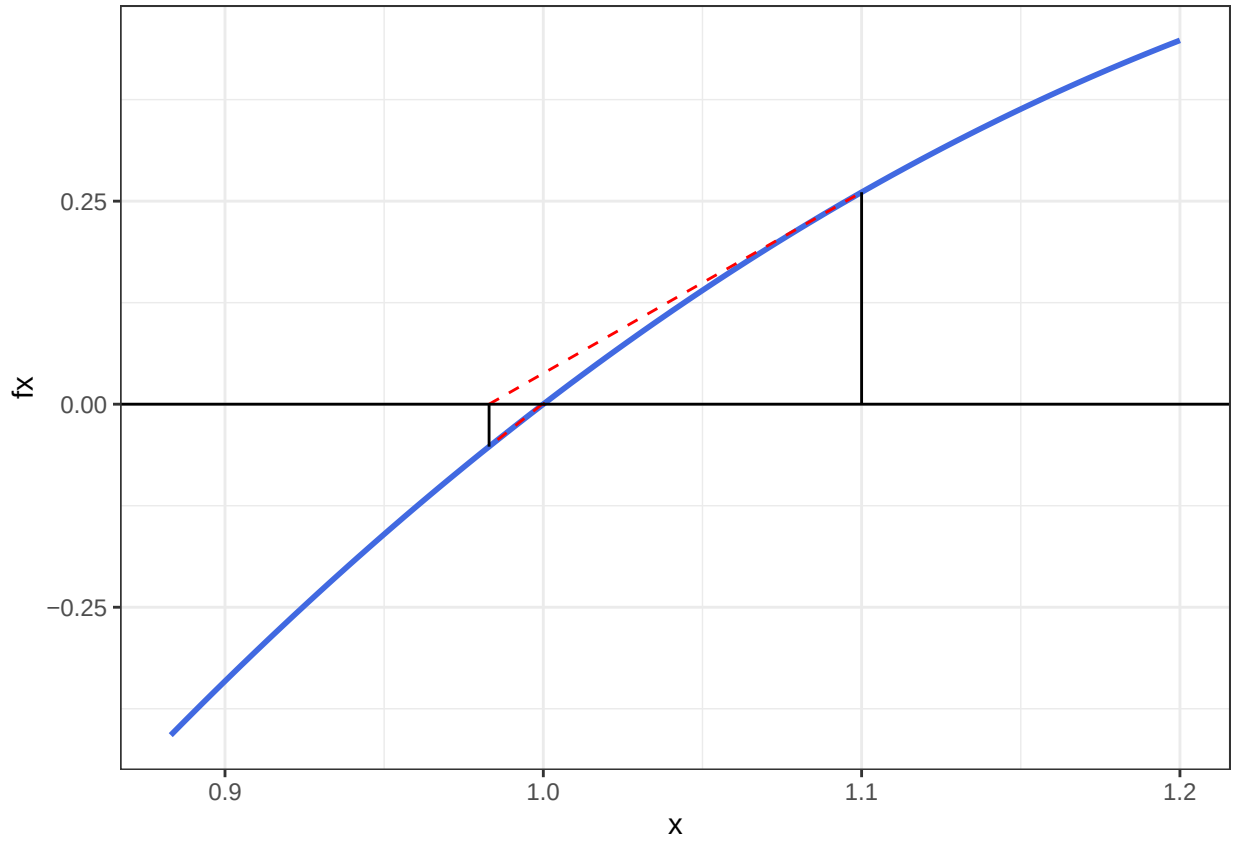



```
## [1] -0.007256248
```

The function f(x) = x^3 - 7x^2 + 14x - 8 using x0 = 1.1, 1.3, 1.4, 1.5, 1.6, 1.7

```
f_d <- function(x){
  value <- x^3 - 7*x^2 + 14*x - 8   # f(x)
  derivative <- 3*x^2 - 14*x + 14 # f'(x)
  c(value, derivative) # the function returns a vector with two values
}
newtonraphson_show(f_d, 1.1, iter = 8)
```

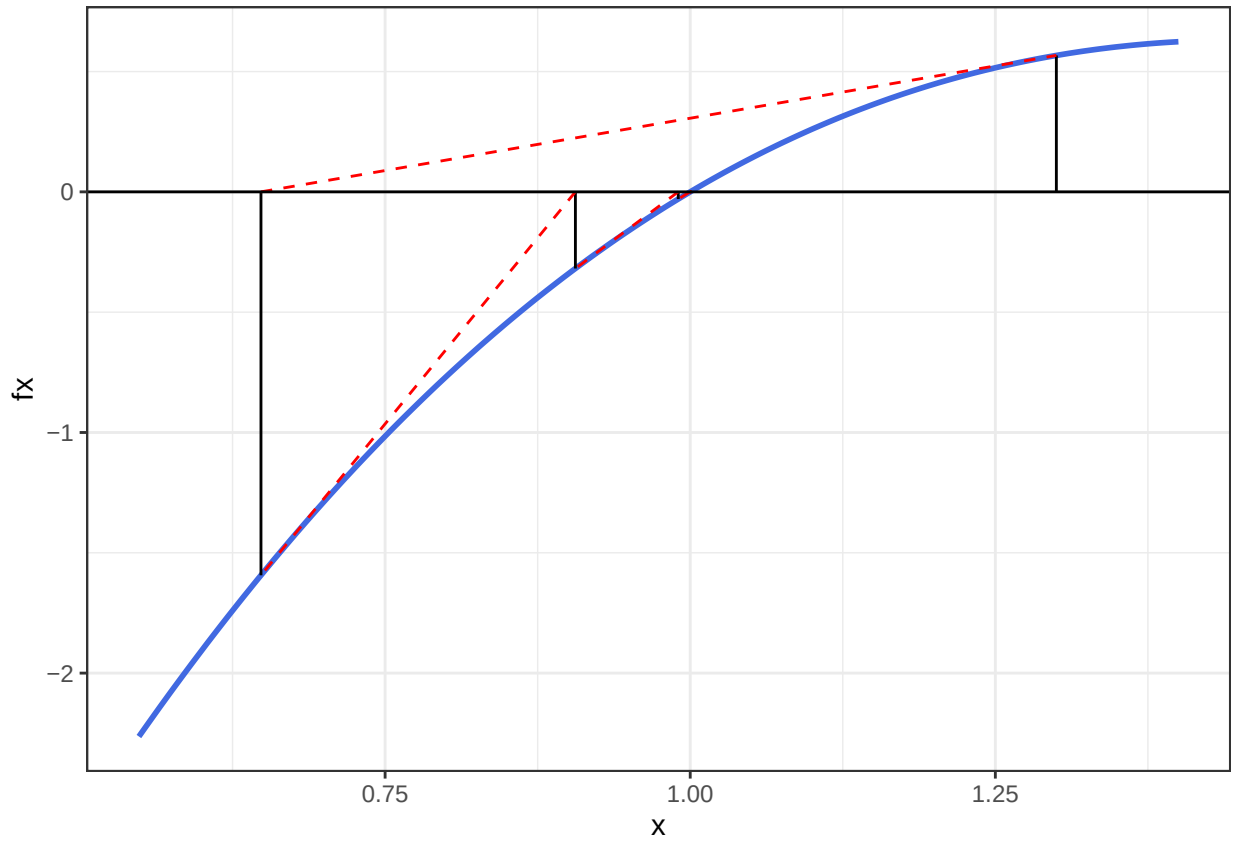
```
## Starting value is: 1.1
## Next x value: 0.9829596
## Next x value: 0.9996266
## Next x value: 0.9999998
## Next x value: 1
## Next x value: 1
## Next x value: 1
## Next x value: 1
## Next x value: 1
```




```
## [1] 1
```

```
newtonraphson_show(f_d, 1.3, iter = 8)
```

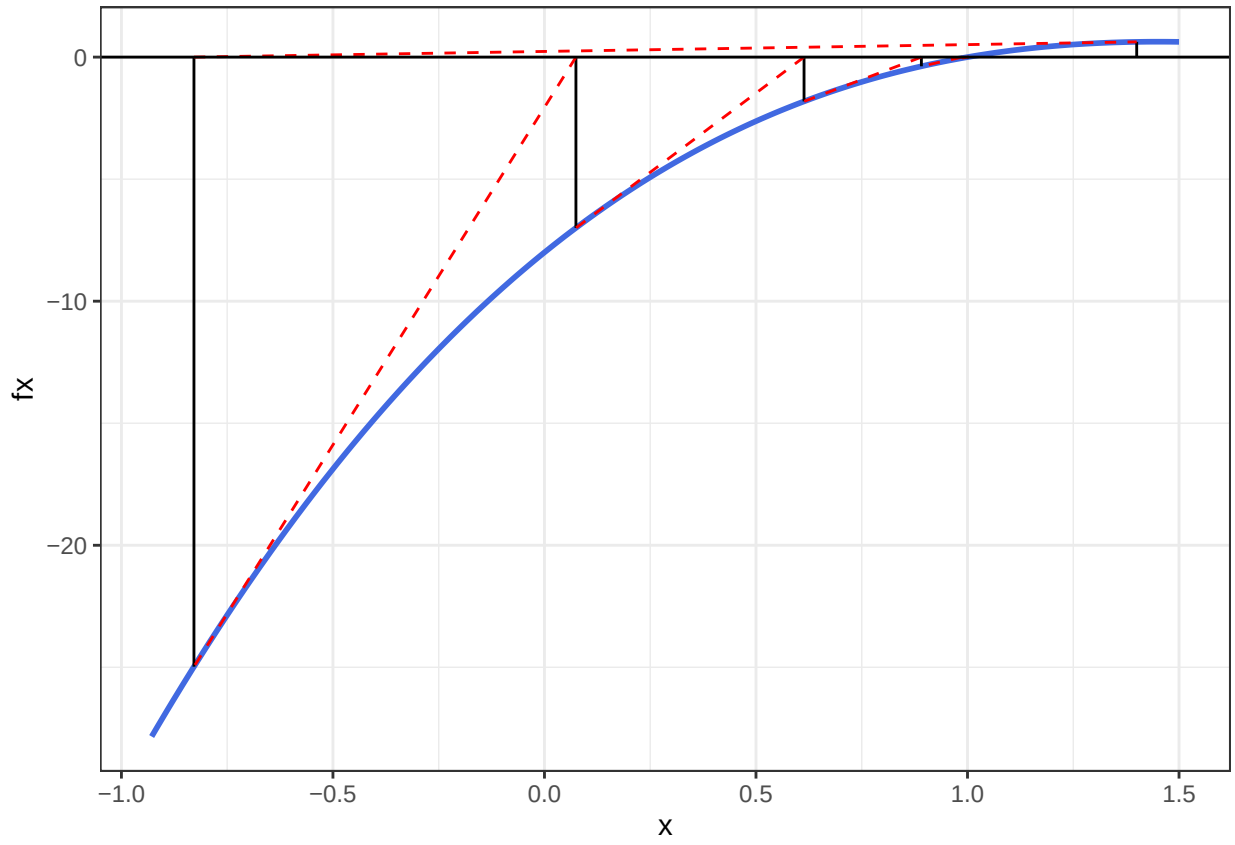
```
## Starting value is: 1.3
## Next x value: 0.6482759
## Next x value: 0.9059224
## Next x value: 0.9901916
## Next x value: 0.9998744
## Next x value: 1
## Next x value: 1
## Next x value: 1
## Next x value: 1
```




```
## [1] 1
```

```
newtonraphson_show(f_d, 1.4, iter = 8)
```

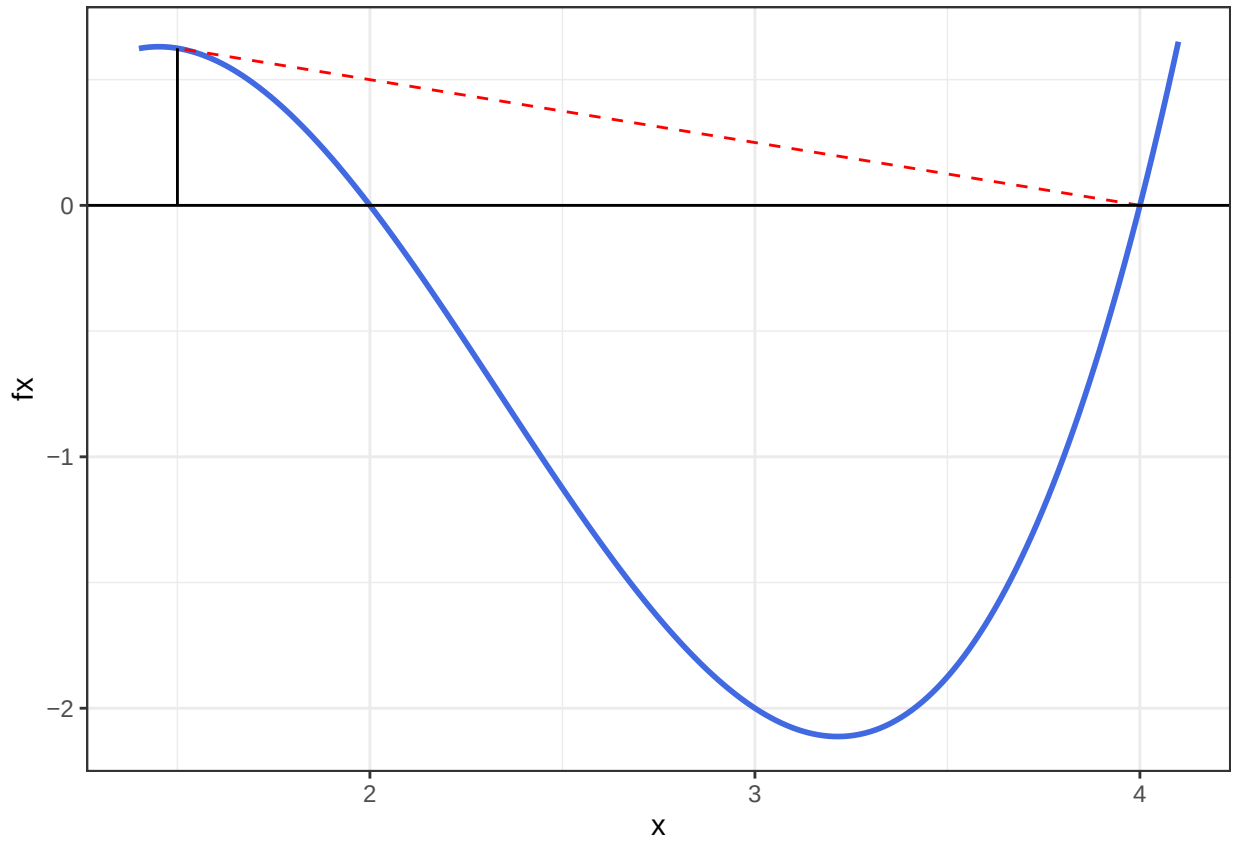
```
## Starting value is: 1.4
## Next x value: -0.8285714
## Next x value: 0.07435419
## Next x value: 0.6136214
## Next x value: 0.891034
## Next x value: 0.9871826
## Next x value: 0.9997869
## Next x value: 0.9999999
## Next x value: 1
```




```
## [1] 1
```

```
newtonraphson_show(f_d, 1.5, iter = 8)
```

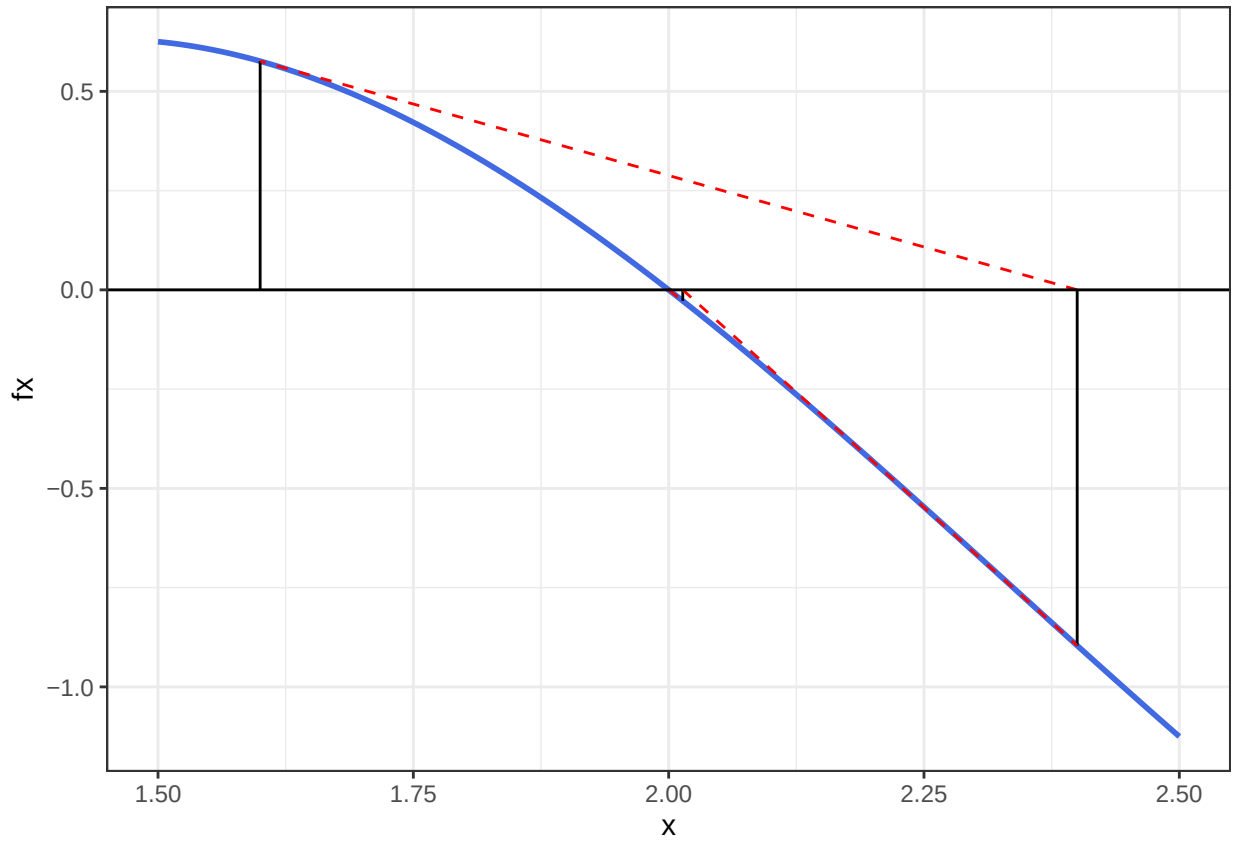
```
## Starting value is: 1.5
## Next x value: 4
## Next x value: 4
## Next x value: 4
## Next x value: 4
## Next x value: 4
## Next x value: 4
## Next x value: 4
## Next x value: 4
```




```
## [1] 4
```

```
newtonraphson_show(f_d, 1.6, iter = 8)
```

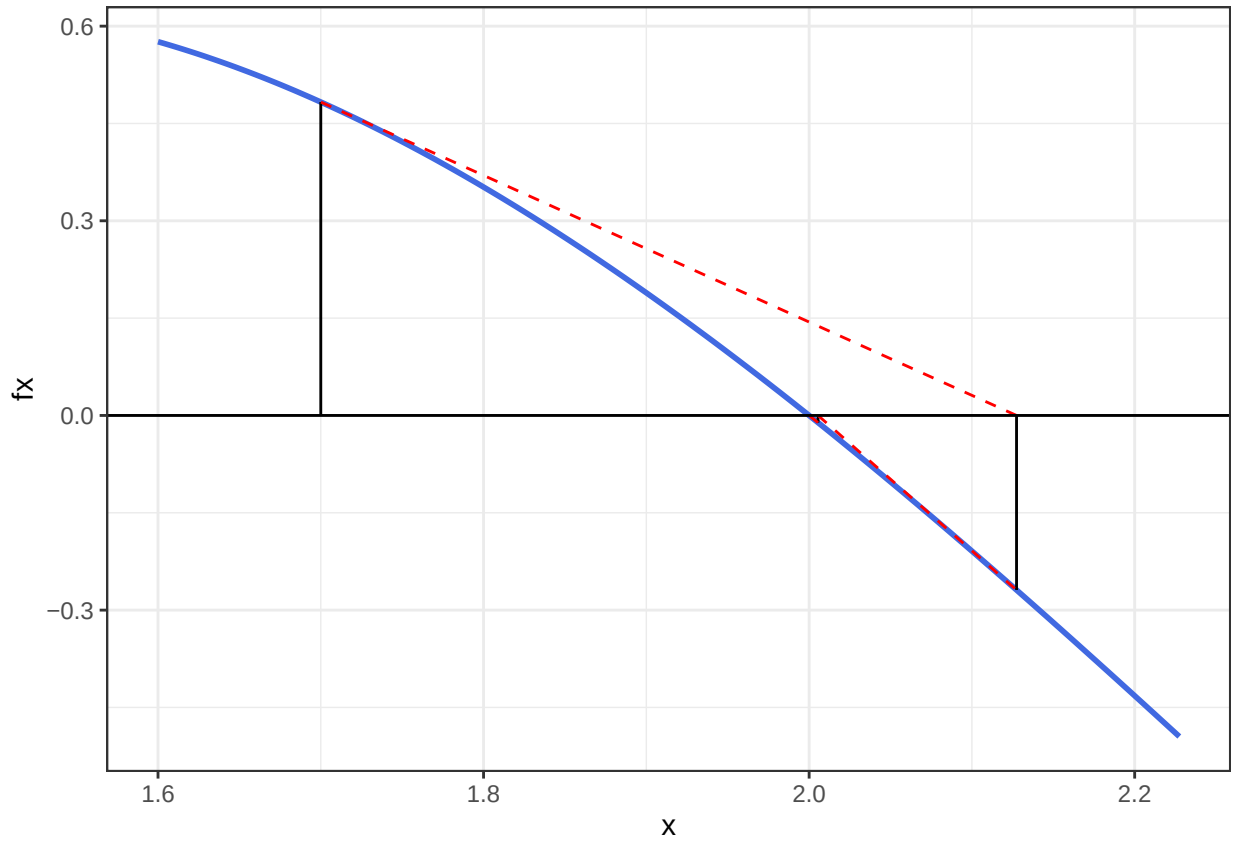
```
## Starting value is: 1.6
## Next x value: 2.4
## Next x value: 2.013793
## Next x value: 2.000091
## Next x value: 2
## Next x value: 2
## Next x value: 2
## Next x value: 2
## Next x value: 2
```




```
## [1] 2
```

```
newtonraphson_show(f_d, 1.7, iter = 8)
```

```
## Starting value is: 1.7
## Next x value: 2.127434
## Next x value: 2.005485
## Next x value: 2.000015
## Next x value: 2
## Next x value: 2
## Next x value: 2
## Next x value: 2
## Next x value: 2
```




```
## [1] 2
```

Root Finding with Secant Method

```
secant_show <- function(ftn, x0, x1, iter = 5) {
 # df_points_1 and df_points_2 are used to track each update
  df_points_1 <- data.frame(
    x1 = numeric(0),
    y1 = numeric(0),
    x2 = numeric(0),
```

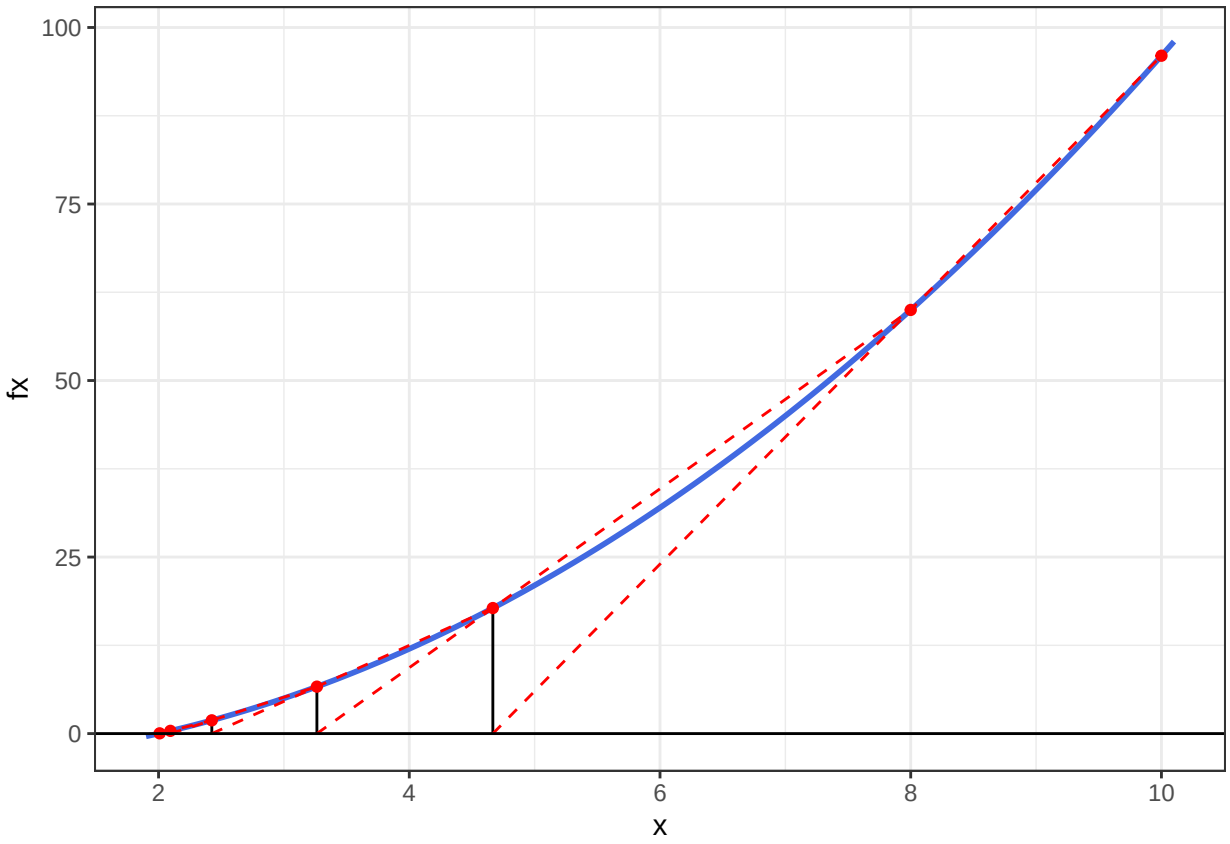
```
  y2 = numeric(0))
df_points_2 <- df_points_1
df_points_3 <- data.frame(
  x1 = numeric(0),
  y1 = numeric(0)
)

xnew <- x1
xnext <- x0
cat("Starting x0 and x1 value are:", xnext, xnew, "\n")
# the algorithm
for(i in 1:iter){
  xold <- xnext
  f_xold <- ftn(xold)
  xnext <- xnew
  f_xnext <- ftn(xnext)
  slope <- (f_xnext - f_xold) / (xnext - xold)
  xnew <- xold - (f_xold * 1 / slope)
  f_xnew <- ftn(xnew)
  cat("Next x value:", xnew, "\n")

  # the line segments
  df_points_1[i,] <- c(x1 = xnew, y1 = 0, x2 = xnew, y2 = f_xnew) # vertical segment
  if((xnew > xold & xnew < xnext) | (xnew < xold & xnew > xnext)) {
    df_points_2[i,] <- c(x1 = xnext, y1 = f_xnext, x2 = xold, y2 = f_xold)
  } else {
    df_points_2[i,] <- c(x1 = xnew, y1 = 0, x2 = xold, y2 = f_xold) # tangent segment
  }
  df_points_3[i,] <- c(x1 = xold, y1 = f_xold) # dot point
}

plot_start <- min(df_points_1$x1, df_points_1$x2, x0, x1) - 0.1
# end is the max of these values
plot_end <- max(df_points_1$x1, df_points_1$x2, x0, x1) + 0.1

# calculate the value of the funtion fx for all x
x <- seq(plot_start, plot_end, length.out = 200)
fx <- rep(NA, length(x))
for (i in seq_along(x)) {
  fx[i] <- ftn(x[i])
}
function_data <- data.frame(x, fx) # data frame containing the function values

p <- ggplot(function_data, aes(x = x, y = fx)) +
  geom_line(color = "royalblue", linewidth = 1) +  # plot the function
  geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2),
               data = df_points_1, color = "black", lty = 1) +
  geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2),
               data = df_points_2, color = "red", lty = 2) +
  geom_point(aes(x = x1, y = y1), data = df_points_3, color = "red") +
  geom_abline(intercept = 0, slope = 0) + # plot the line y = 0
  theme_bw()
```

```
  print(p) # produce the plot

  xnew # value that gets returned
}
```

Produce graphs for: ### The function $f(x) = x^2 - 4$ using x0 = 10, and x1 = 8

```
f_a <- function(x) {x^2 - 4}
secant_show(f_a, 10, 8, 7)
```

```
## Starting x0 and x1 value are: 10 8
## Next x value: 4.666667
## Next x value: 3.263158
## Next x value: 2.424779
## Next x value: 2.094333
## Next x value: 2.008867
## Next x value: 2.000204
## Next x value: 2
```

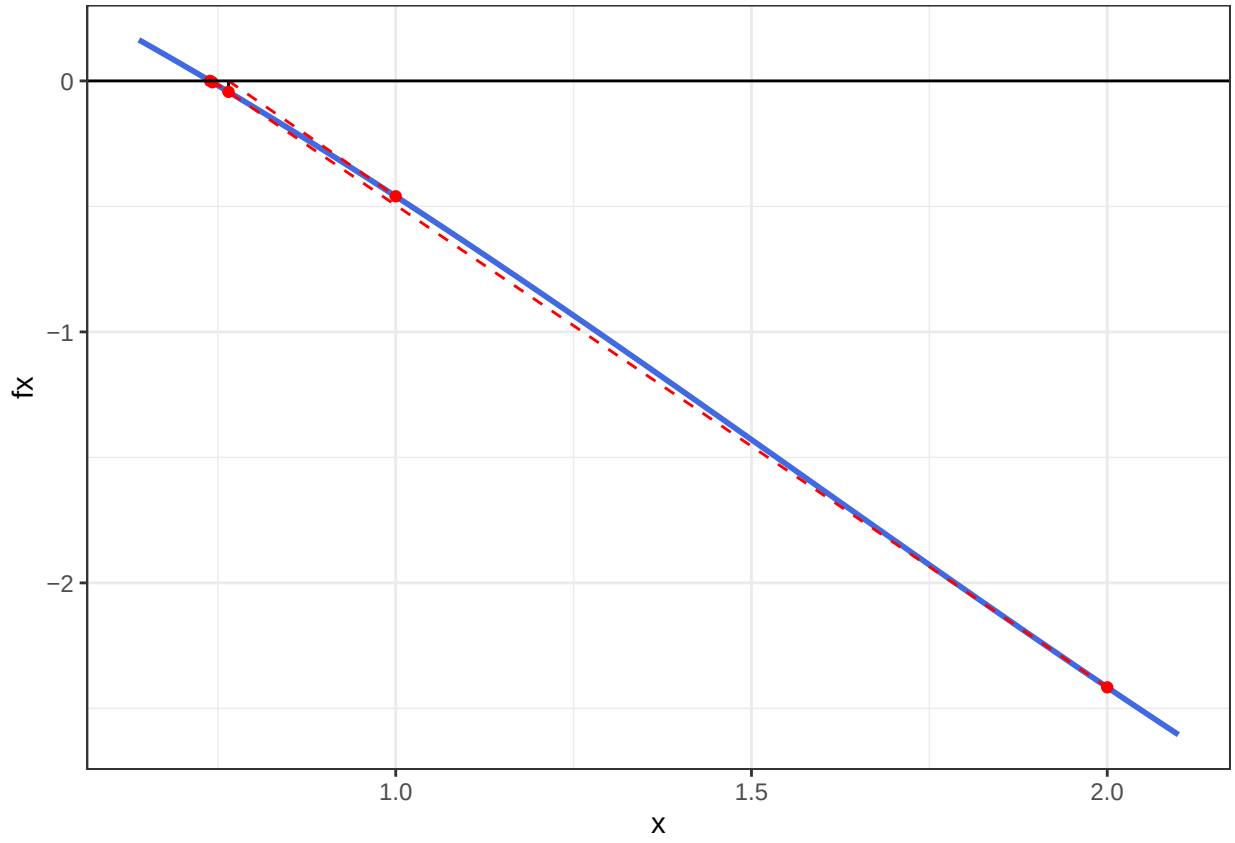



```
## [1] 2
```

$f(x) = cos(x) - x$ **using** $x_0 = 1$ **and** $x_1 = 2$**.**

```
f_b <- function(x) {cos(x) - x}
secant_show(f_b, 1, 2)
```

```
## Starting x0 and x1 value are: 1 2
## Next x value: 0.7650347
## Next x value: 0.7422994
## Next x value: 0.7391033
## Next x value: 0.7390851
## Next x value: 0.7390851
```

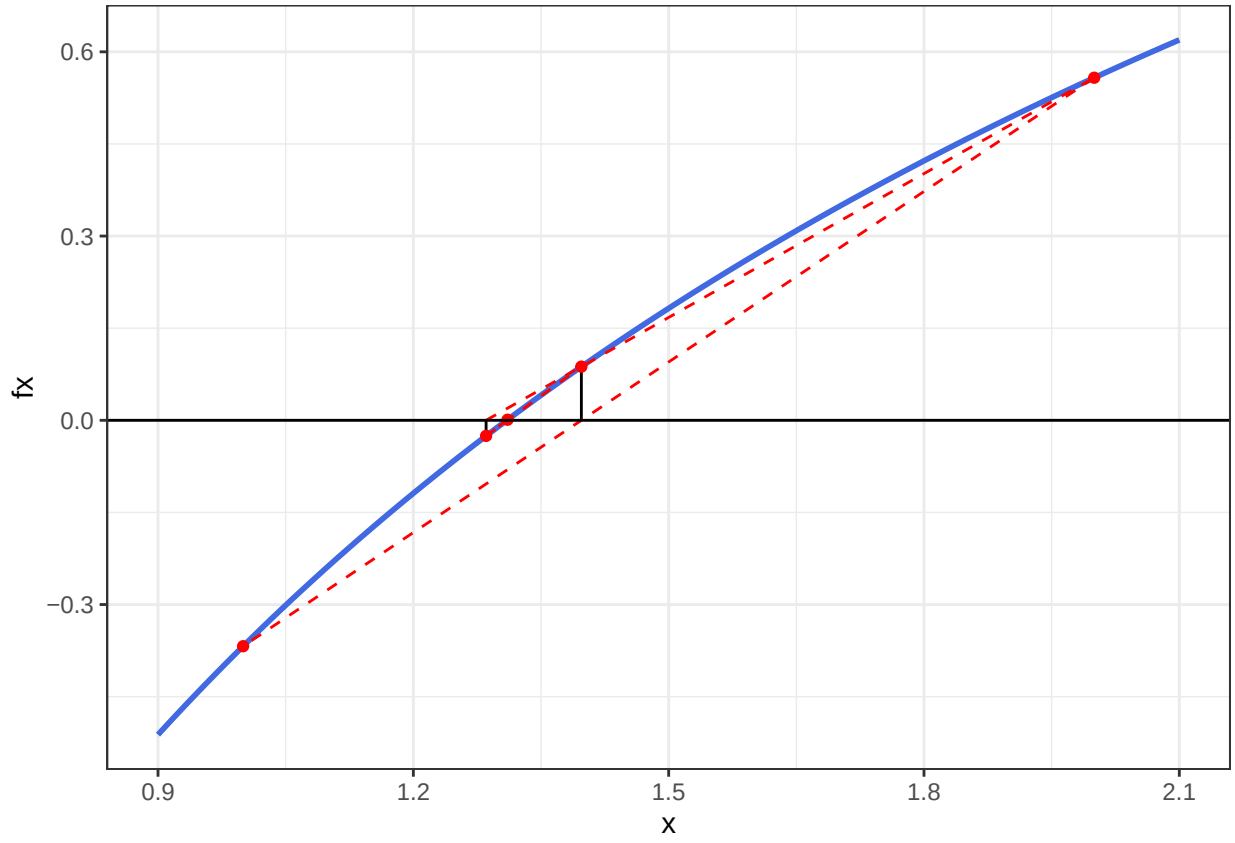



```
## [1] 0.7390851
```

$f(x) = log(x) - exp(-x)$ **using** $x_0 = 1$ **and** $x_1 = 2$**.**

```
f_c <- function(x) {log(x) - exp(-x)}
secant_show(f_c, 1, 2)
```

```
## Starting x0 and x1 value are: 1 2
## Next x value: 1.39741
## Next x value: 1.285476
## Next x value: 1.310677
## Next x value: 1.309808
## Next x value: 1.3098
```

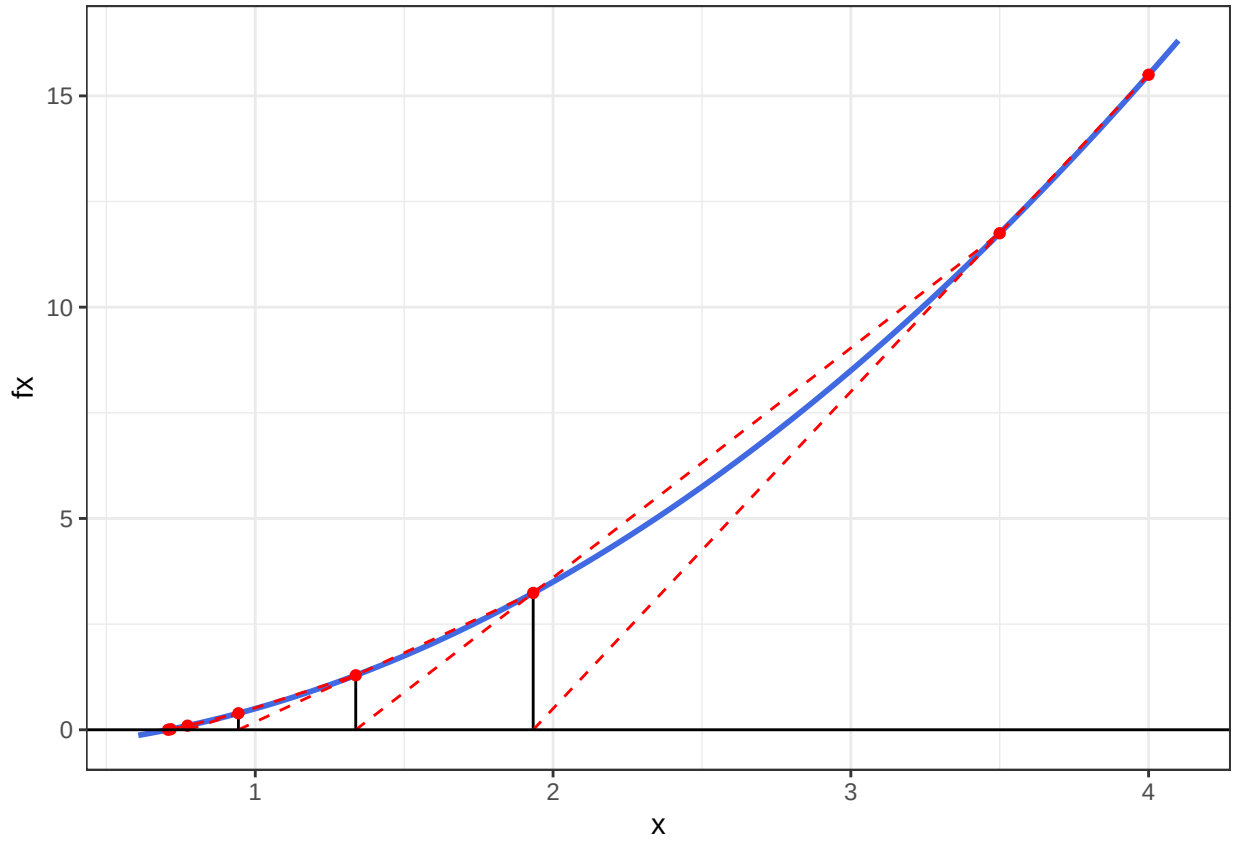



```
## [1] 1.3098
```

Find the root of $x^2 - 0.5$ using $x_0 = 4$ and $x_1 = 3.5$.

```
f_d <- function(x) {x^2 - 0.5}
secant_show(f_d, 4, 3.5, 8)
```

```
## Starting x0 and x1 value are: 4 3.5
## Next x value: 1.933333
## Next x value: 1.337423
## Next x value: 0.9434163
## Next x value: 0.7724116
## Next x value: 0.7161008
## Next x value: 0.7075014
## Next x value: 0.7071093
## Next x value: 0.7071068
```



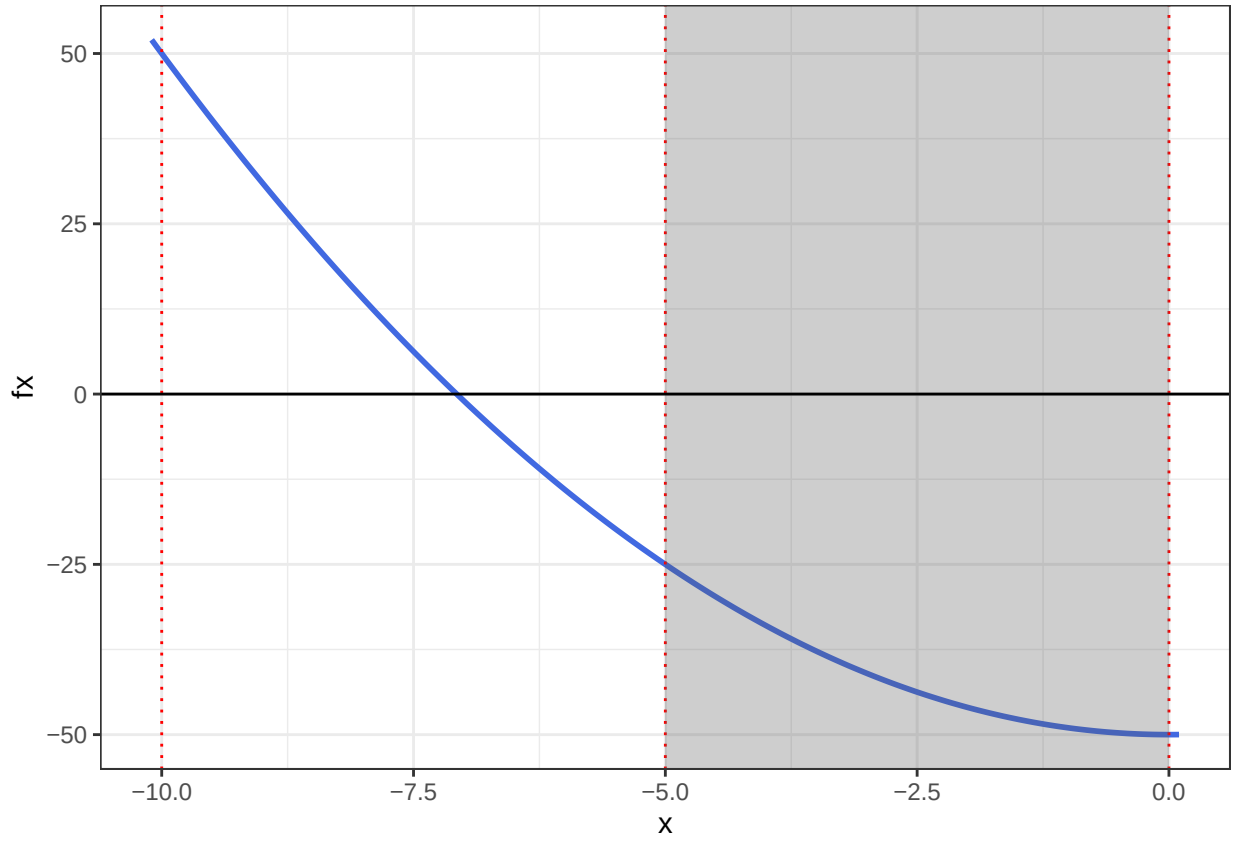

```
## [1] 0.7071068
```

Bisection Search Graph

```r
bisection_show <- function(ftn, x_l, x_r, tol = 1e-8, iter) {
  points_df <- data.frame(left = numeric(0), middle = numeric(0),
                          right = numeric(0), fill = numeric(0))
  f_l <- ftn(x_l)
  f_r <- ftn(x_r)
  for (i in 1:iter) {
    x_m <- (x_l + x_r) / 2
    f_m <- ftn(x_m)
    if (identical(all.equal(f_m, 0), TRUE)) {
      break
    }
    points_df[i, ] <- c(left = x_l, middle = x_m, right = x_r, fill <- NA)
    if (f_l * f_m < 0) {
      x_f <- x_r
      x_r <- x_m
    } else {
      x_f <- x_l
      x_l <- x_m
    }
    points_df[i, 4] <- x_f
  }

  plot_start <- min(points_df$left, points_df$right) - 0.1
  plot_end <- max(points_df$left, points_df$right) + 0.1

  # calculate the value of the funtion fx for all x
  x <- seq(plot_start, plot_end, length.out = 200)
  fx <- rep(NA, length(x))
  for (i in seq_along(x)) {
    fx[i] <- ftn(x[i])
  }

  function_data <- data.frame(x, fx) # data frame containing the function values

  p <- ggplot(function_data, aes(x = x, y = fx)) +
    geom_line(color = "royalblue", linewidth = 1) +  # plot the function
    geom_vline(aes(xintercept = left), data = points_df, color = "red", lty = 3) +
    geom_vline(aes(xintercept = middle), data = points_df, color = "red", lty = 3) +
    geom_vline(aes(xintercept = right), data = points_df, color = "red", lty = 3) +
    geom_rect(inherit.aes = FALSE, aes(xmin = middle, xmax = fill, ymin = -Inf, ymax = Inf),
              data = points_df, alpha = 0.3) +
    geom_abline(intercept = 0, slope = 0) + # plot the line y = 0
    theme_bw()

  print(p) # produce the plot
  x_m
}

f <- function(x) {
  x^2 - 50
}
```

1 iteration

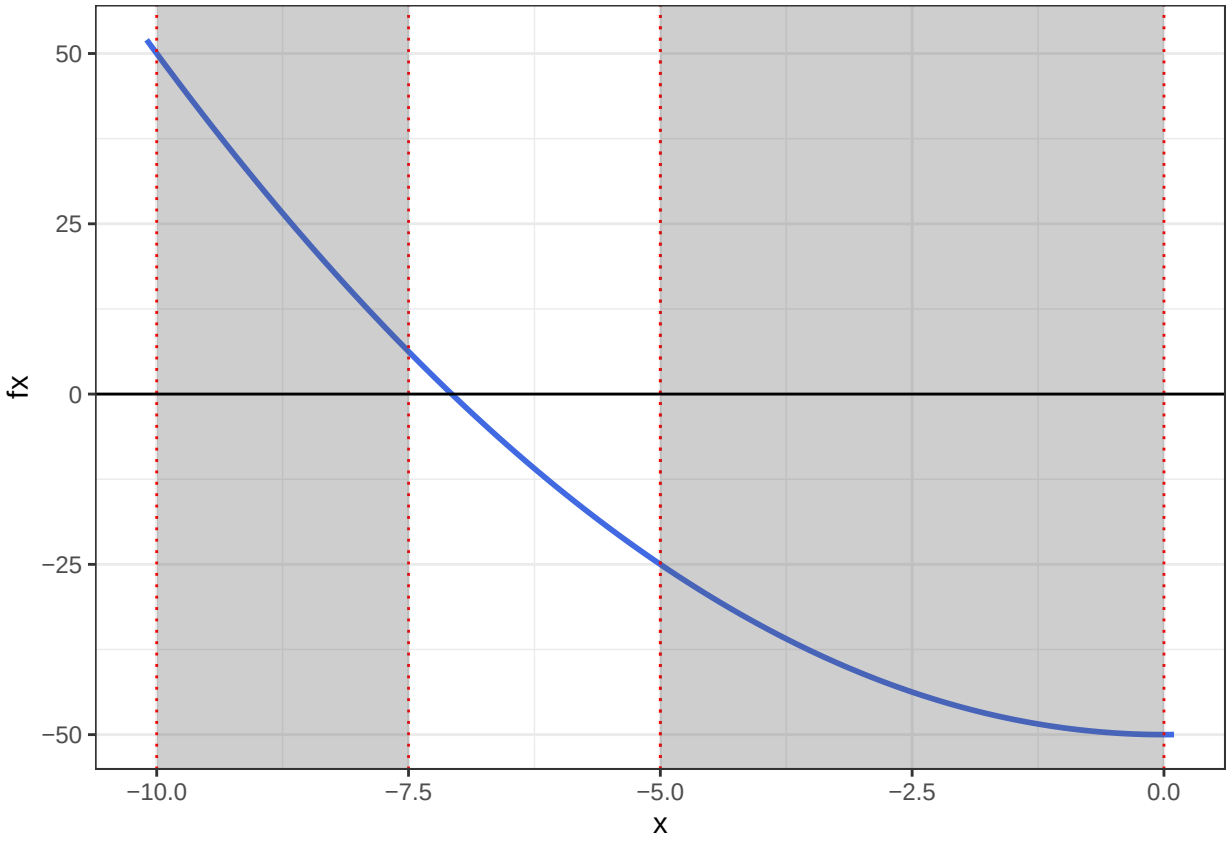
```
bisection_show(f, -10, 0, iter = 1)
```




```
## [1] -5
```

2 iterations

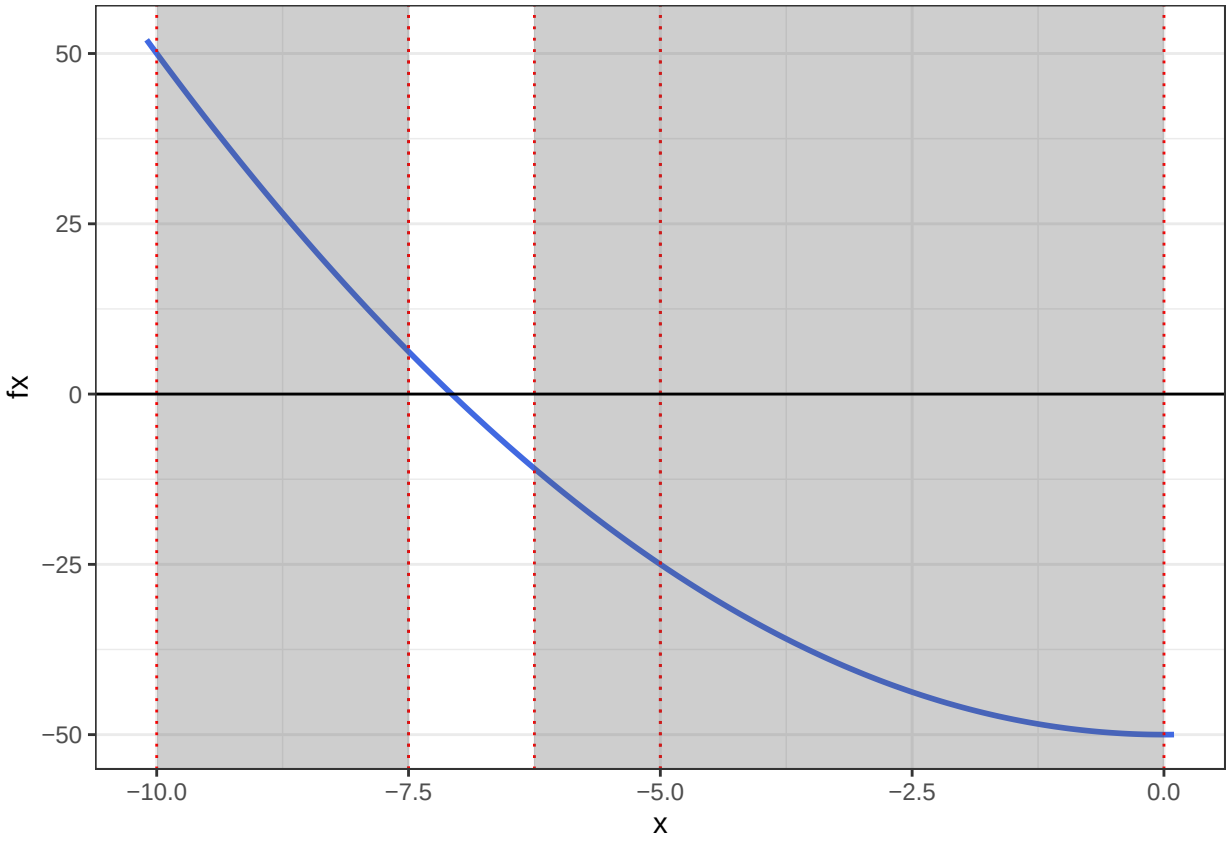
```
bisection_show(f, -10, 0, iter = 2)
```




```
## [1] -7.5
```

3 iterations

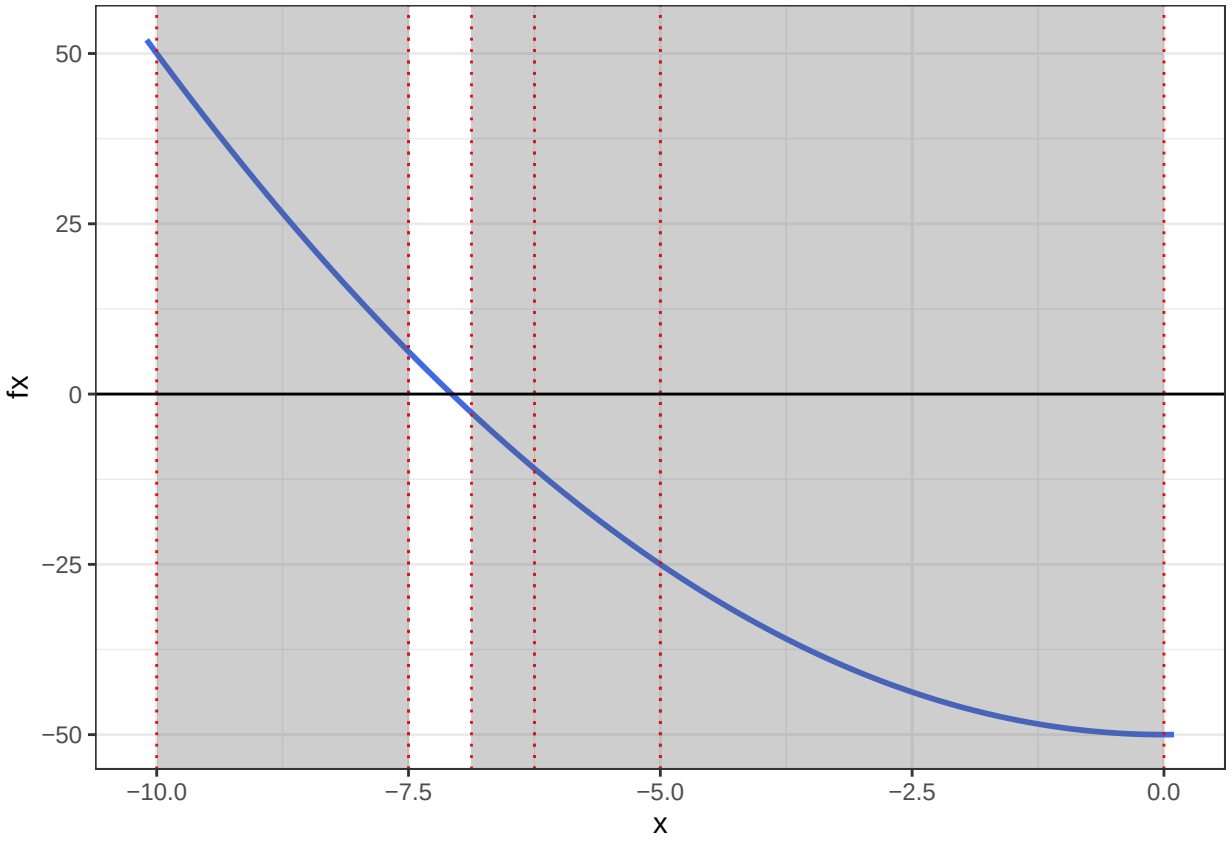
```
bisection_show(f, -10, 0, iter = 3)
```




```
## [1] -6.25
```

4 iterations

```
bisection_show(f, -10, 0, iter = 4)
```




```
## [1] -6.875
```

Coordinate Descent Algorithm for Optimization

```
##### A modifcation of code provided by Eric Cai
golden = function(f, lower, upper, tolerance = 1e-5)
{
   golden.ratio = 2/(sqrt(5) + 1)

   ## Use the golden ratio to find the initial test points
   x1 <- lower + golden.ratio * (upper - lower)
   x2 <- upper - golden.ratio * (upper - lower)

   ## the arrangement of points is:
   ## lower ----- x2 --- x1 ----- upper

   ### Evaluate the function at the test points
   f1 <- f(x1)
   f2 <- f(x2)

   while (abs(upper - lower) > tolerance) {
        if (f2 > f1) {
        # the minimum is to the right of x2
        lower <- x2  # x2 becomes the new lower bound
        x2 <- x1     # x1 becomes the new x2
        f2 <- f1     # f(x1) now becomes f(x2)
        x1 <- lower + golden.ratio * (upper - lower)
        f1 <- f(x1)  # calculate new x1 and f(x1)
        } else {
        # then the minimum is to the left of x1
        upper <- x1  # x1 becomes the new upper bound
        x1 <- x2     # x2 becomes the new x1
        f1 <- f2
        x2 <- upper - golden.ratio * (upper - lower)
        f2 <- f(x2)  # calculate new x2 and f(x2)
        }
    }
    (lower + upper)/2 # the returned value is the midpoint of the bounds
}
```

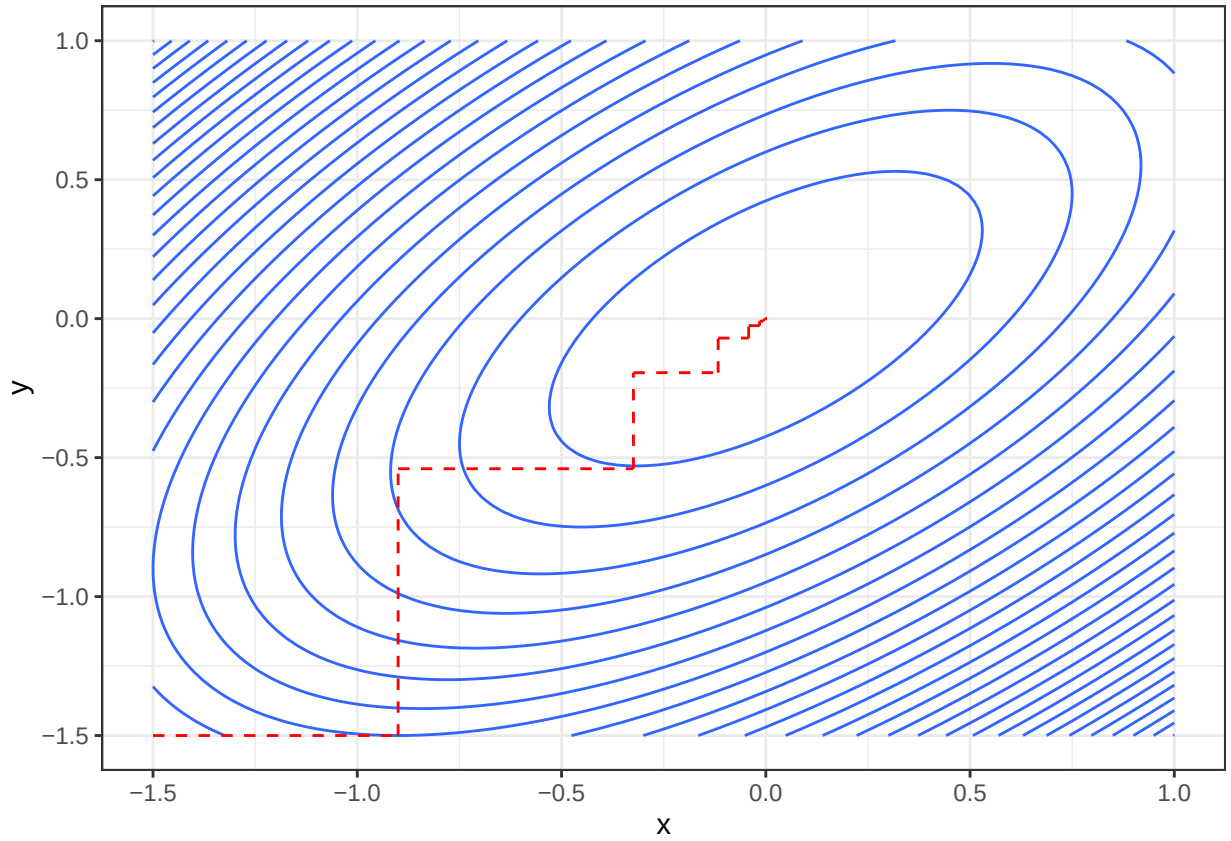
```
g <- function(x,y) {
    5 * x ^ 2 - 6 * x * y + 5 * y ^ 2
}
x <- seq(-1.5, 1, len = 100)
y <- seq(-1.5, 1, len = 100)
```

```r
# Data frane to store coordinates
contour_df <- data.frame(
  x = rep(x, each = 100),
  y = rep(y, 100),
  z = outer(x, y, g)[1:100^2]
  )

# Setting the graph
p <- ggplot(contour_df, aes(x = x, y = y, z = z)) +
  geom_contour(binwidth = 0.9) +
  theme_bw()

# Coordinate descent function
coordinate_descent <- function(x_i, y_i) {
  cat("Start point is: ", x_i, y_i, "\n") #Prints the start point
  points_df <- data.frame(x0 = numeric(0), x1 = numeric(0), y0 = numeric(0),
                          y1 = numeric(0), z = numeric(0))
  xold <- x_i # assigning the initial x value
  yold <- y_i # assigning the initial y value
  iter <- 1
  while(iter <= 15) {
    f_x <- function(x) { 5 * x ^ 2 - 6 * x * yold + 5 * yold ^ 2} #function to calculate new x value
    xnew <- golden(f_x, -1.5, 1.5)
    f_y <- function(y) { 5 * xnew ^ 2 - 6 * xnew * y + 5 * y ^ 2} #function to calculate new y value
    ynew <- golden(f_y, -1.5, 1.5)
    points_df[iter, ] <- c(x0 = xold, x1 = xnew, y0 = yold, y1 = ynew,
                           z = outer(xold,  yold, g)) # initial and new x and y values are recorded to
    cat("Next values of x and y: ", xnew, ynew, "\n") # prints the next x and y values
    iter <- iter + 1
    if(abs(xnew - xold) < 1e-5){ # the algorithm stops if the difference between new x and old x values
      break
    }
    xold <- xnew
    yold <- ynew
  }
  p <- p + geom_segment(aes(x = x0, xend = x1, y = y0, yend = y0),
                    data = points_df, color = "red", lty = 2) +
    geom_segment(aes(x = x1, xend = x1, y = y0, yend = y1),
                 data = points_df, color = "red", lty = 2)
  print(p)
  cat("Minimum value is at: ", xnew, ynew)
}
```

Graph for starting point x = -1.5, and y = -1.5.

```
coordinate_descent(-1.5, -1.5)
```

```
## Start point is:  -1.5 -1.5
## Next values of x and y:  -0.9000005 -0.5399991
## Next values of x and y:  -0.3239996 -0.1943994
## Next values of x and y:  -0.1166404 -0.06998455
## Next values of x and y:  -0.04199138 -0.02519619
## Next values of x and y:  -0.01511759 -0.009071722
## Next values of x and y:  -0.005444077 -0.003266769
## Next values of x and y:  -0.001961105 -0.001174652
## Next values of x and y:  -0.0007038706 -0.0004213549
## Next values of x and y:  -0.0002519686 -0.0001515037
## Next values of x and y:  -8.941291e-05 -5.364775e-05
## Next values of x and y:  -3.315613e-05 -1.788258e-05
## Next values of x and y:  -1.105204e-05 -6.830539e-06
## Next values of x and y:  -4.221505e-06 -4.221505e-06
```

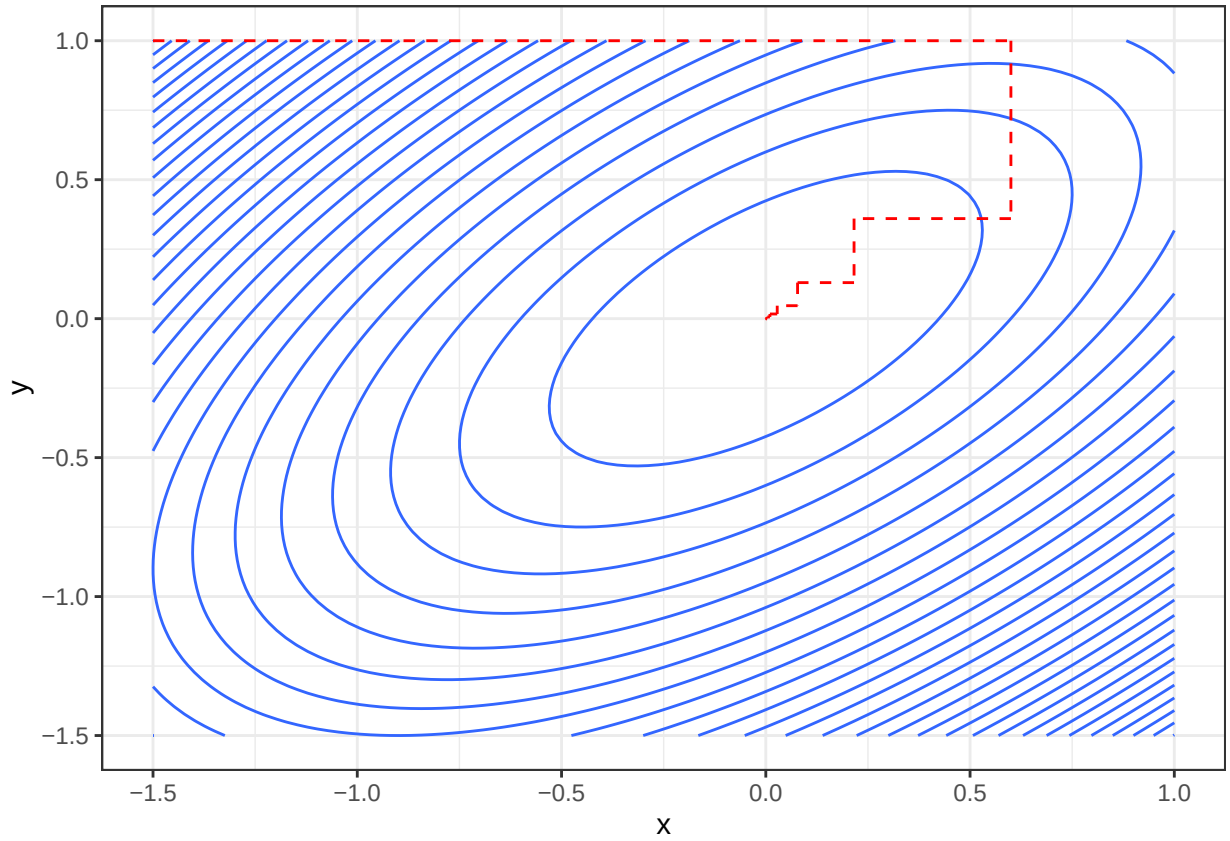



```
## Minimum value is at:  -4.221505e-06 -4.221505e-06
```

Graph for starting point x = -1.5, and y = 1.

```
coordinate_descent(-1.5, 1)
```

```
## Start point is:  -1.5 1
## Next values of x and y:  0.6000003 0.3600014
## Next values of x and y:  0.2160011 0.1296002
## Next values of x and y:  0.07775905 0.04665583
## Next values of x and y:  0.02799318 0.01679518
## Next values of x and y:  0.01007699 0.00604587
## Next values of x and y:  0.003627645 0.002177308
## Next values of x and y:  0.001305664 0.0007822315
## Next values of x and y:  0.0004681721 0.0002809033
## Next values of x and y:  0.0001693863 0.000100465
## Next values of x and y:  6.209076e-05 3.576517e-05
## Next values of x and y:  2.210409e-05 1.366108e-05
## Next values of x and y:  6.830539e-06 4.221505e-06
## Next values of x and y:  4.221505e-06 4.221505e-06
```




```
## Minimum value is at:  4.221505e-06 4.221505e-06
```