Stats 102A - Homework 5 - Output File

Daren Sathasivam

Academic Integrity Statement

"By including this statement, I, Daren Sathasivam, declare that all of the work in this assignment is my own original work. At no time did I look at the code of other students nor did I search for code solutions online. I understand that plagiarism on any single part of this assignment will result in a 0 for the entire assignment and that I will be referred to the dean of students."

1. An IEEE 754 Mini-Floating Point number [22 points, 2pts each part]

In class, I demonstrated the use of a mini-floating point number system using 8 bits. In my class demo, I used 1 bit for the sign, 3 bits for the exponent, and 4 bits for the mantissa. For this problem, imagine a different mini-floating point system with 10 bits - 1 bit for the sign, 4 bits for the exponent, and 5 bits for the mantissa.

```
0 0000 00000 # would now represent the decimal value 0
```

Answer the following questions under this new system.

a. What is the bias that would be used for the exponent? What is the largest positive exponent (that is not used to represent infinity)? What is the most negative exponent (that will not lead to a denormalized state)?

With 4 bits for the exponent, the bias used will be 7($2^{4-1} - 1 = 2^3 - 1 = 8 - 1 = 7$). The largest positive exponent is 7 because of "1110" which is 14($2^3 + 2^2 + 2^1 + 0 * 2^0 = 8 + 4 + 2 = 14$) - bias(7) = 7. The most negative exponent is -6 since "0001" is 1 - bias(7) = -6.

b. How would the value 5.5 be stored in this system?

5 = 101; 0.5 = 1
Regular Binary: 101.1 –> $(1.011)_2 * 2^2$
Exponent bit: $bias + p = 7 + 2 = 9$ –> 9 = 1001
Mantissa bit: 011000
ANS: 0 1001 01100

c. What value would the following bit sequence represent `0 0111 00000`?

Mantissa bit: 00000
Exponent bit: 0111 = 7 –> $p = 7 - bias = 0$
Regular binary: $(1.0000)_2 * 2^0$
$(-1)^s * (1 + m) * 2^e = (-1)^0 * (1 + 0) * 2^0$
ANS: 1

d. What value would the following bit sequence represent `0 0111 00001`? (Express as a fraction and also in decimal.)

Mantissa bit: $00001 = 2^{-5} = \frac{1}{32}$
Exponent bit: $0111 = 7 -> p = 7 - bias = 0$
Regular binary: $(1.0001)_2 * 2^0$
$(-1)^s * (1+m) * 2^e = (-1)^0 * (1 + \frac{1}{32}) * 2^0$
ANS: $\frac{33}{32} = 1.03125$

e. What is the smallest positive normalized value that can be expressed? (Fill in the bits. Express as a fraction and also in decimal.)

`0 0001 00000`
Mantissa bit: 00000
Exponent bit: $0001 = 1 -> p = 1 - bias = -6$
Regular binary: $(1.0000)_2 * 2^{-6}$
$(-1)^s * (1+m) * 2^e = (-1)^0 * (1+0) * 2^{-6} = 2^{-6}$
ANS: $\frac{1}{64} = 0.015625$

f. What is the smallest positive (denormalized) non-zero value that can be expressed? (Fill in the bits. Express as a fraction and also in decimal.)

`0 0000 00001`
Mantissa bit: $00001 = 2^{-5}$
Exponent bit(bias is 6): $0000 = 0 -> p = 0 - bias = -6$
Regular binary: $(0.0000)_2 * 2^{-6}$
$(-1)^s * (1+m) * 2^e = (-1)^0 * (0 + \frac{1}{32}) * 2^{-6} = 2^{-11}$
ANS: $\frac{1}{2048} = 0.00048828125$

g. What is the largest denormalized value that can be expressed? (Fill in the bits. Express as a fraction and also in decimal.)

`0 0000 11111`
Mantissa bit: $11111 = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-5} = \frac{31}{32}$
Exponent bit: $0000 = 0 -> p = 0 - bias = -6$
Regular binary: $(0.11111)_2 * 2^{-6}$
$(-1)^s * (1+m) * 2^e = (-1)^0 * (0 + 0.96875) * 2^{-6} = \frac{31}{32} * \frac{1}{64}$
ANS: $\frac{31}{2048} = 0.01513671875$

h. What is the largest finite value that can be expressed with this system? (Fill in the bits. Express as a fraction and also in decimal.)

`0 1110 11111`
Mantissa bit: $11111 = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-5} = \frac{31}{32}$
Exponent bit: $1110 = 14 -> p = 14 - bias = 7$
Regular binary: $(1.11111)_2 * 2^7$
$(-1)^s * (1+m) * 2^e = (-1)^0 * (1 + 0.96875) * 2^7 = 252$
ANS: 252

i. With our 10 bit floating-point system, what is the smallest value you can add to 1 so that the sum will be different from 1? In other words, what is the machine epsilon of this system?

`0 0111 00001`

The smallest value that can be added to this 10-bit system is the machine epsilon value of $2^{-5} * 2^0$. The machine epsilon of this system is $\frac{1}{32}$. The base number is $1 = 2^0$.

j. What is the smallest value you can add to the number 2 so that the sum will be different from 2? (Express as a fraction)

The smallest value that can be added to 2 in this 10-bit system is $2^{-5} * 2^1 = \frac{1}{16}$. The base number is now $2 = 2^1$.

k. What is the smallest value you can add to the number 4 so that the sum will be different from 4? (Express as a fraction)

The smallest value that can be added to 4 in this 10-bit system is $2^{-5} * 2^2 = \frac{1}{8}$. The base number is now $4 = 2^2$

2. Root Finding with Fixed Point Iteration [12 points, 2 points each part]

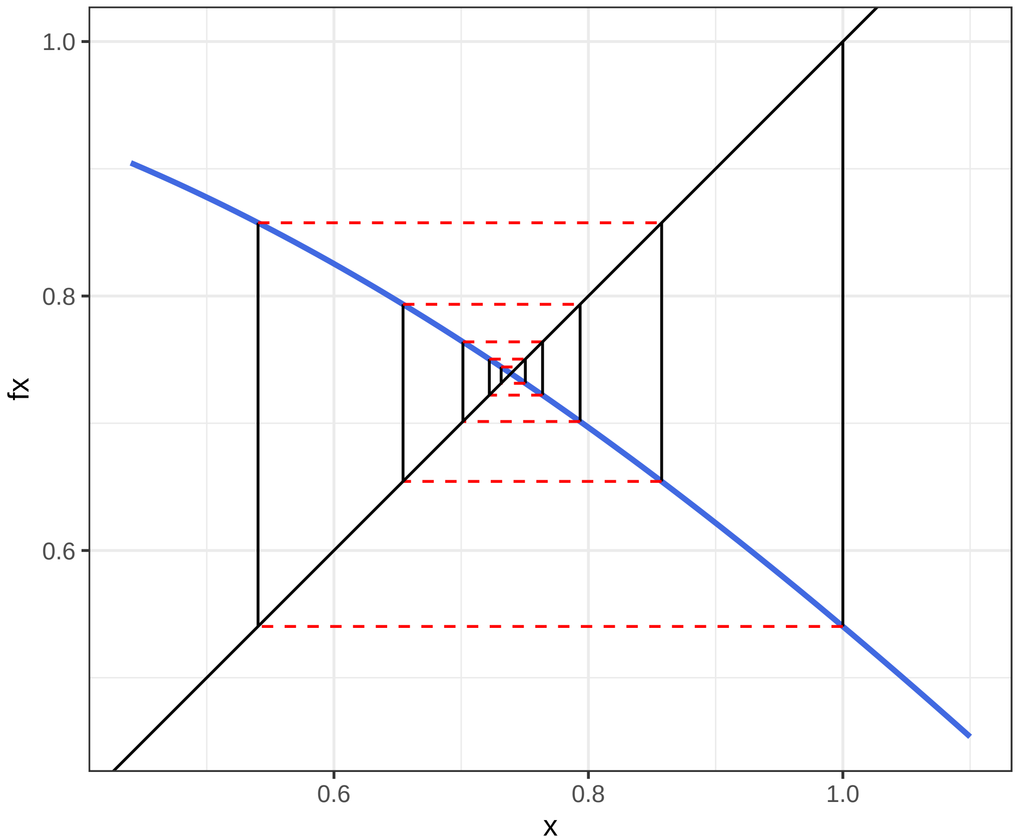
```r
library(ggplot2)
fixedpoint_show <- function(ftn, x0, iter = 5){
  # applies fixed-point method to find x such that ftn(x) = x
  # ftn is a user-defined function

  # 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
```

```r
  # 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
}

## Part a, x0 = 1
# f <- function(x) cos(x)
# fixedpoint_show(f, 1, iter= 10)
```

Do part (a) using x0 = 1

```r
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
```

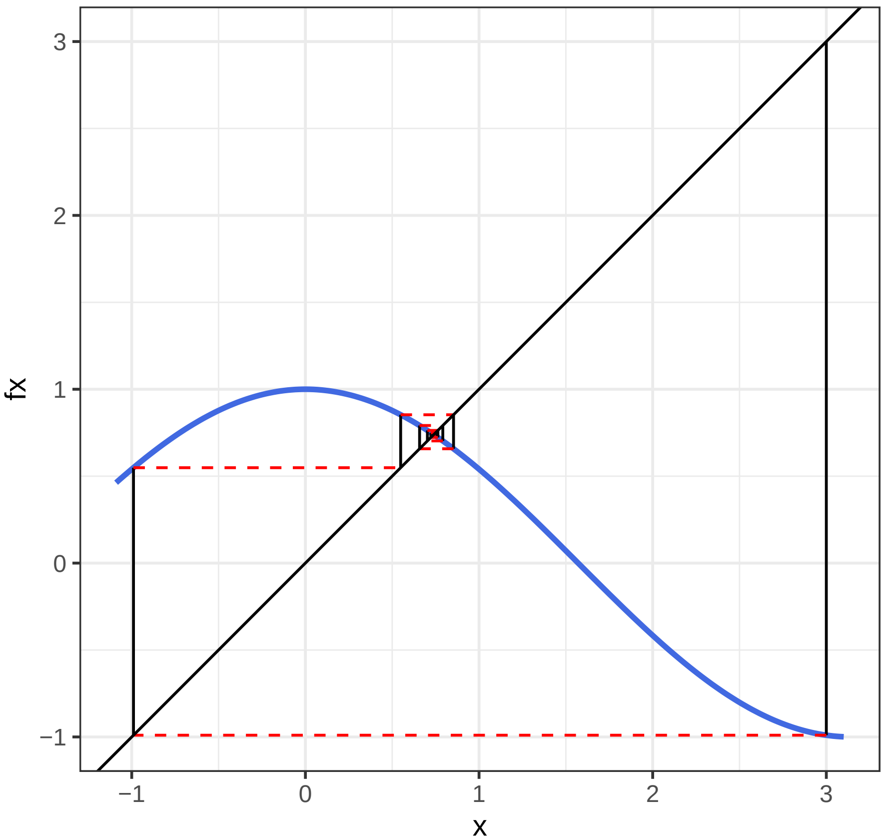



```
## [1] 0.7442374
```

Do part (a) using x0 = 3

```r
f <- function(x) cos(x)
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
```

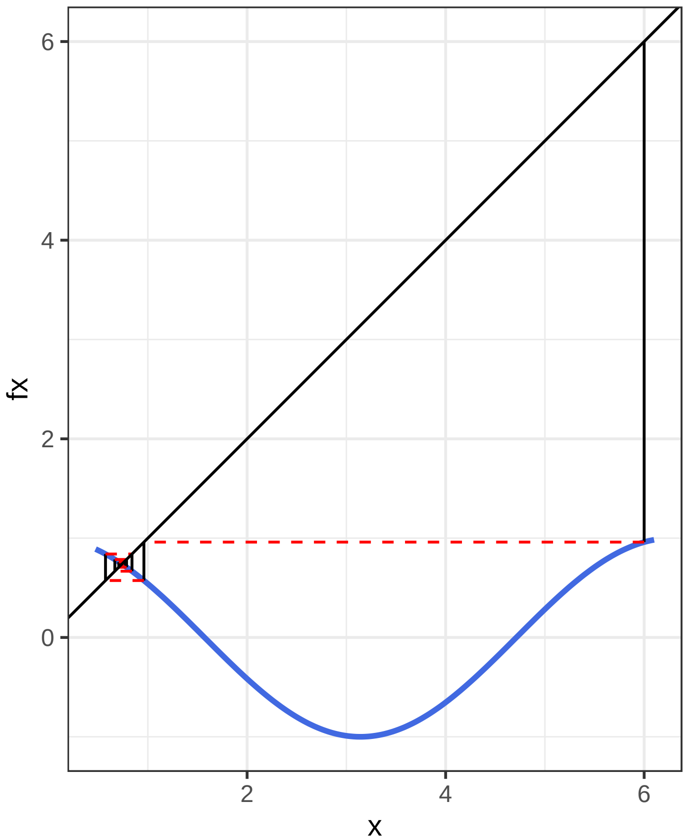



```
## [1] 0.731691
```

Do part (a) using x0 = 6

```
f <- function(x) cos(x)
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
```

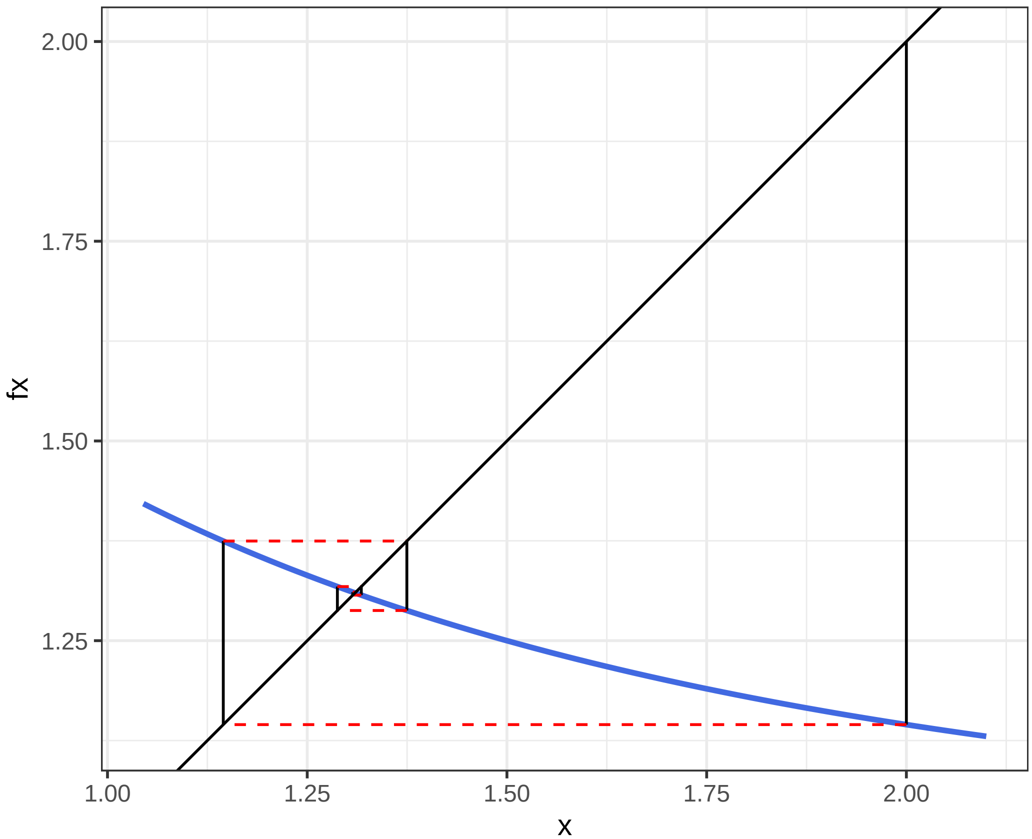



```
## [1] 0.7325566
```

Do part (b) using x0 = 2

```r
g <- function(x) exp(exp(-x))
fixedpoint_show(g, 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
```

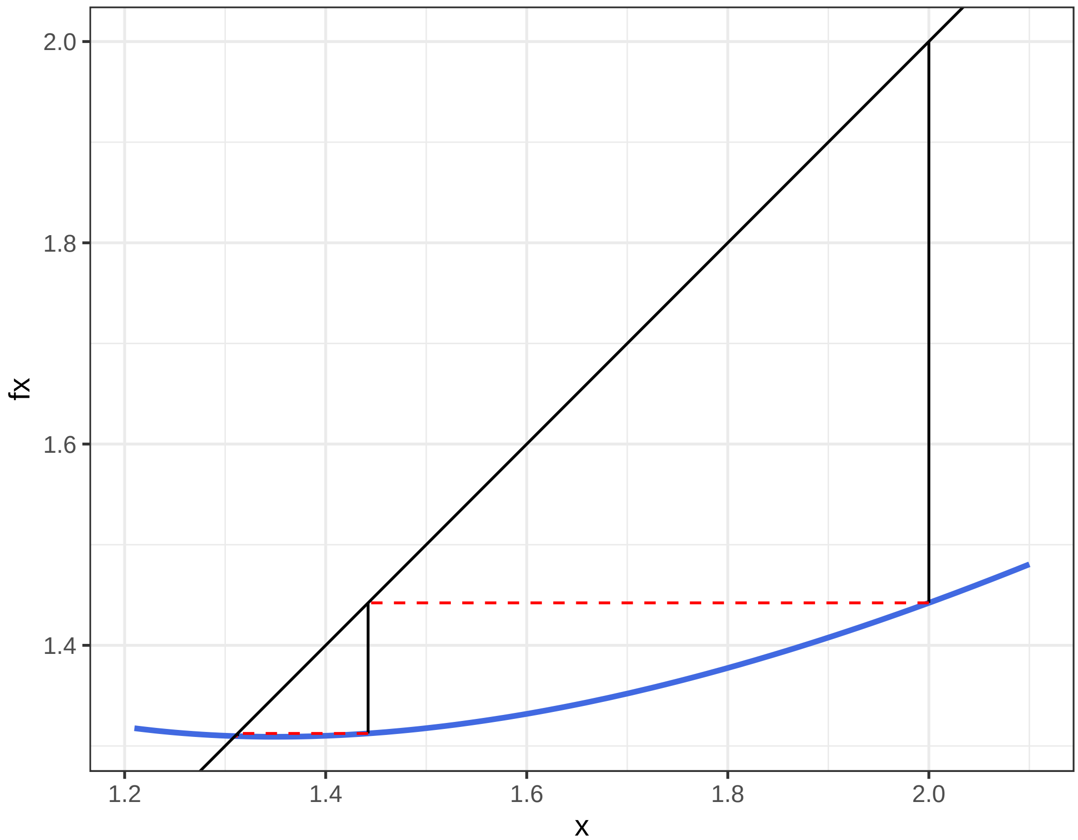



```
## [1] 1.309815
```

Do part (c) using x0 = 2

```r
h <- function(x) x - log(x) + exp(-x)
fixedpoint_show(h, 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
```

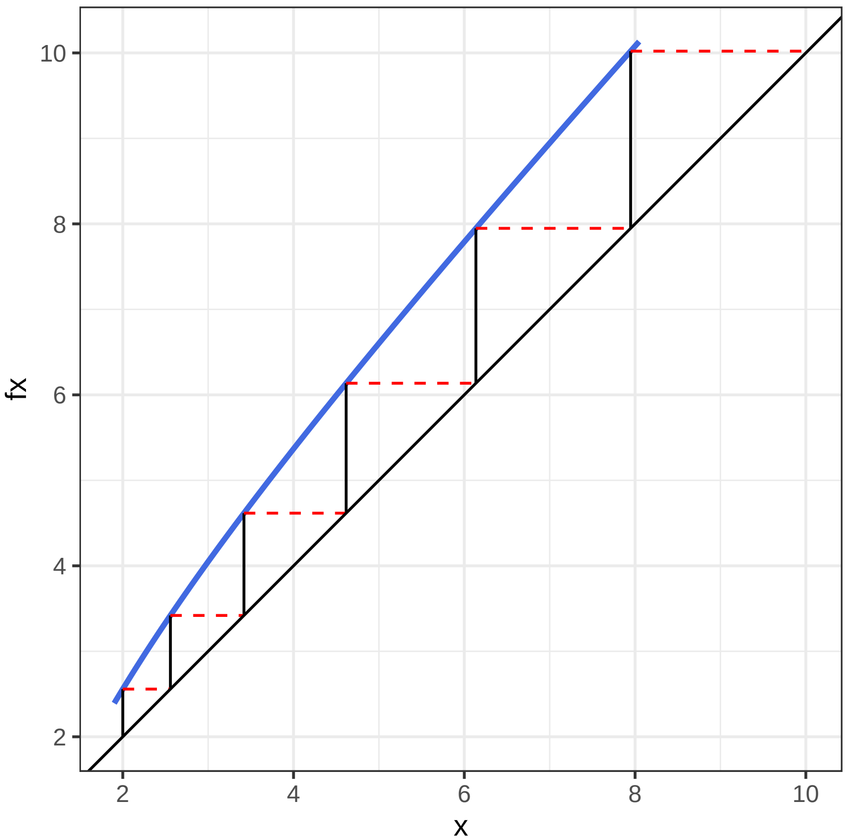



```
## [1] 1.3098
```

Do part (d) using x0 = 2, no more than 6 iterations

```r
k <- function(x) x + log(x) - exp(-x)
fixedpoint_show(k, 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
```

3. Root Finding with Newton Raphson [22 points, 10 points for completing the code. 1 pts each graph]

- NOTES: **Newton-Raphson**

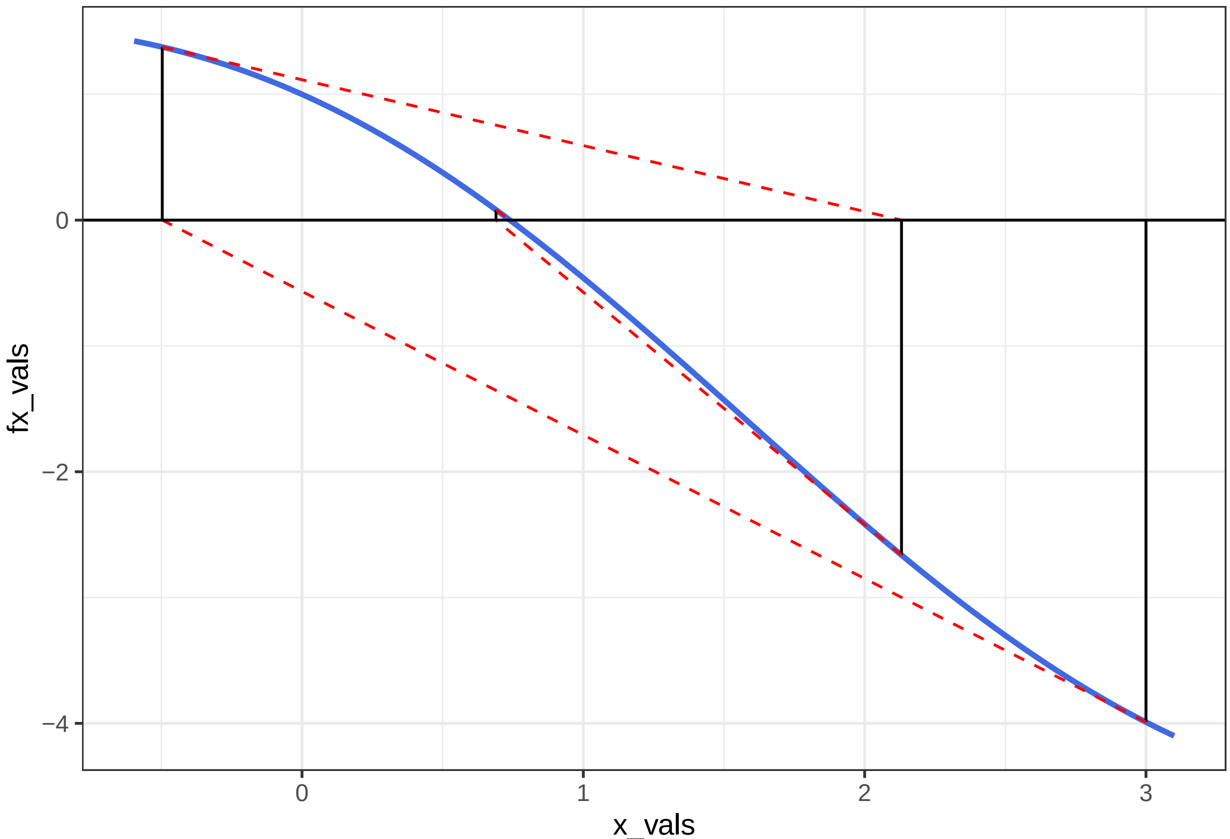
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

```
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
  # From the textbook
  tol <- 1e-9
  max.iter = 100
  cat("Starting value is:", xnew, "\n")
```

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

    # the line segments. You will need to replace the NAs with the appropriate values
    df_points_1[i,] <- c(x1 = xold, y1 = f_xold[1], x2 = xold, y2 = 0) # vertical segment
    yend <- f_xold[1] + f_xold[2] * (xnew - xold) # Tangent intersect
    df_points_2[i,] <- c(x1 = xold, y1 = f_xold[1], x2 = xnew, y2 = yend) # tangent segment
    if (abs(f_xold[1]) < tol) {
      cat("Algrotihm converged at iteration: ", i, "\n")
      break
    }
  }

  # Plot range
  plot_start <- min(df_points_1$x1, df_points_1$x2, x0) - 0.1
  plot_end <- max(df_points_1$x1, df_points_1$x2, x0) + 0.1

  # Calculate value of the function fx for all x
  x_vals <- seq(plot_start, plot_end, length.out = 200)
  fx_vals <- sapply(x_vals, function(x) ftn(x)[1])

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

  print(p)
  # Output depends on success of algorithm
  if (abs(f_xold[1]) > tol) {
    cat("Algorithm failed to converge within given iterations!\n")
  }
  return(xnew) # value that gets returned
}

## Part a
# example of how your functions could be written
a <- function(x){
  value <- cos(x) - x    # f(x)
  derivative <- -sin(x) - 1  # f'(x)
  return(c(value, derivative)) # the function returns a vector with two values
}

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

```
## Starting value is: 3
## Iteration:  1 Next x value: -0.4965582
## Iteration:  2 Next x value: 2.131004
```

```
## Iteration:  3 Next x value: 0.6896627
## Iteration:  4 Next x value: 0.739653
## Iteration:  5 Next x value: 0.7390852
## Iteration:  6 Next x value: 0.7390851
## Iteration:  7 Next x value: 0.7390851
## Algrotihm converged at iteration:  7
```



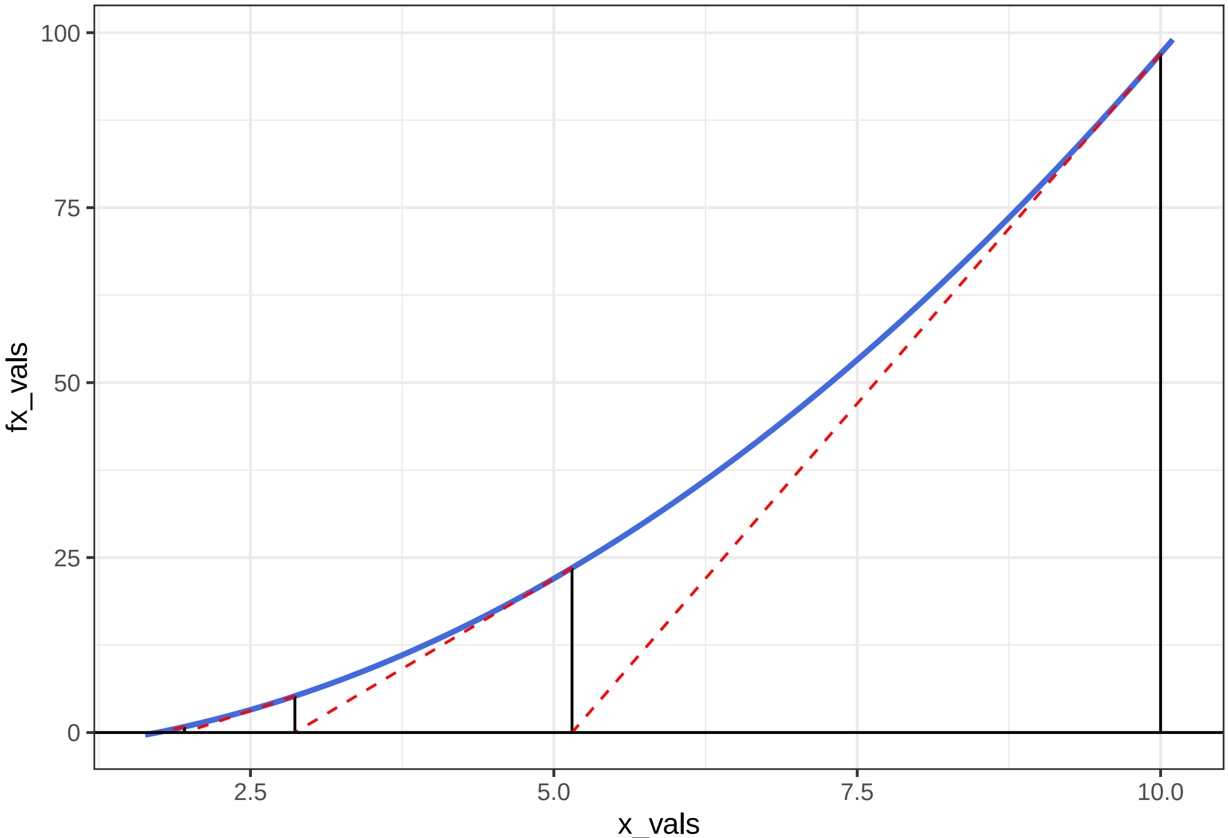

```
## [1] 0.7390851
```

Example from slides($f(x) = x^2 - 3, x_0 = 10$):

```r
ftn <- function(x) {
  fx <- x^2 - 3
  dfx <- 2*x
  return(c(fx, dfx))
}
newtonraphson_show(ftn, 10, 10)
```

```
## Starting value is: 10
## Iteration:  1 Next x value: 5.15
## Iteration:  2 Next x value: 2.866262
## Iteration:  3 Next x value: 1.956461
## Iteration:  4 Next x value: 1.744921
## Iteration:  5 Next x value: 1.732098
## Iteration:  6 Next x value: 1.732051
## Iteration:  7 Next x value: 1.732051
## Iteration:  8 Next x value: 1.732051
```

```
## Algrotihm converged at iteration:  8
```




```
## [1] 1.732051
```

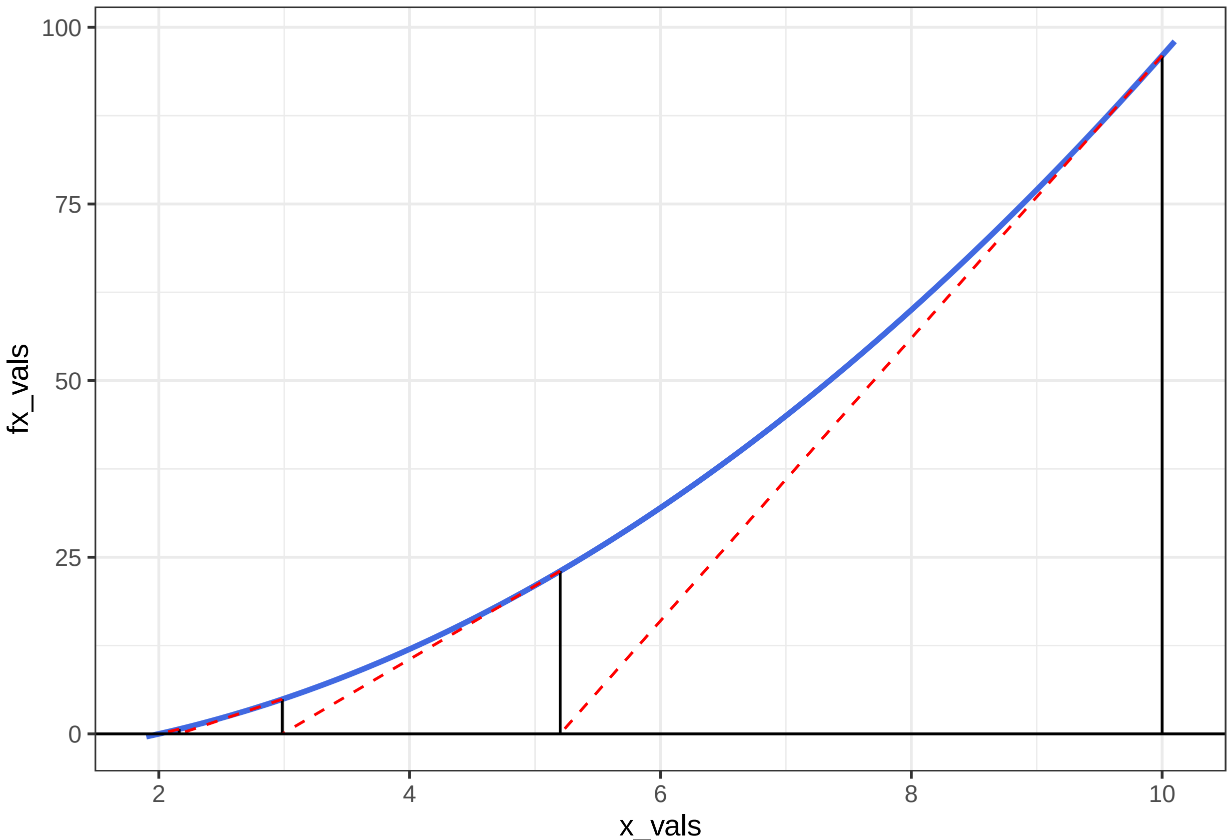
Produce graphs for:

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

```
# --- TO CHECK --- #
# fx <- function(x) {
#   fx <- x^2 - 4
#   fx
# }
# uniroot(fx, c(1.5, 2.5))

ftn <- function(x) {
  fx <- x^2 - 4 # f(x)
  dfx <- 2*x # f'(x)
  c(fx, dfx)
}
newtonraphson_show(ftn, 10, 10)
```

```
## Starting value is: 10
## Iteration:  1 Next x value: 5.2
## Iteration:  2 Next x value: 2.984615
## Iteration:  3 Next x value: 2.162411
## Iteration:  4 Next x value: 2.006099
## Iteration:  5 Next x value: 2.000009
```

```
## Iteration:  6 Next x value: 2
## Iteration:  7 Next x value: 2
## Algrotihm converged at iteration:  7
```

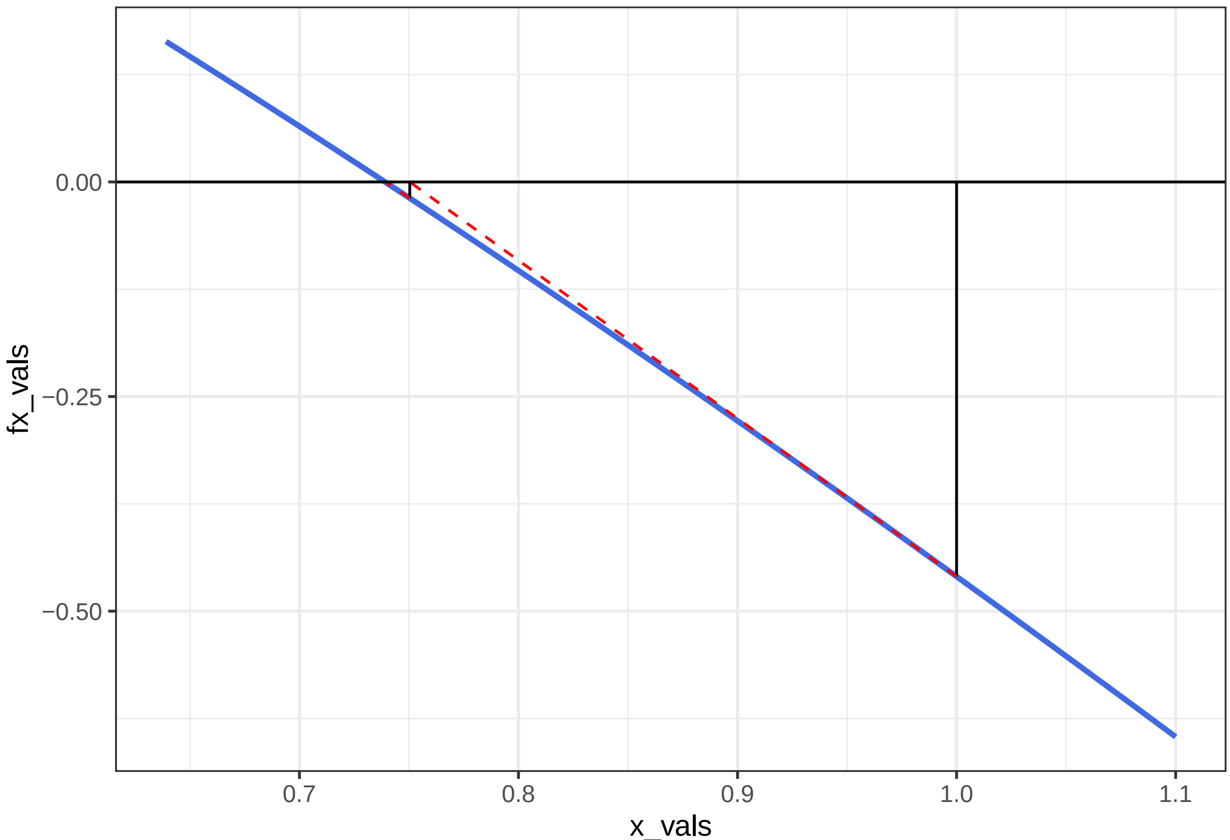



```
## [1] 2
```

part (a) using x0 = 1, 3, 6 Results should be similar to finding fixed point of cos(x)

```r
ftna <- function(x) {
  fx <- cos(x) - x # f(x)
  dfx <- -sin(x) -1 # f'(x)
  c(fx, dfx)
}
newtonraphson_show(ftna, 1, 10)
```

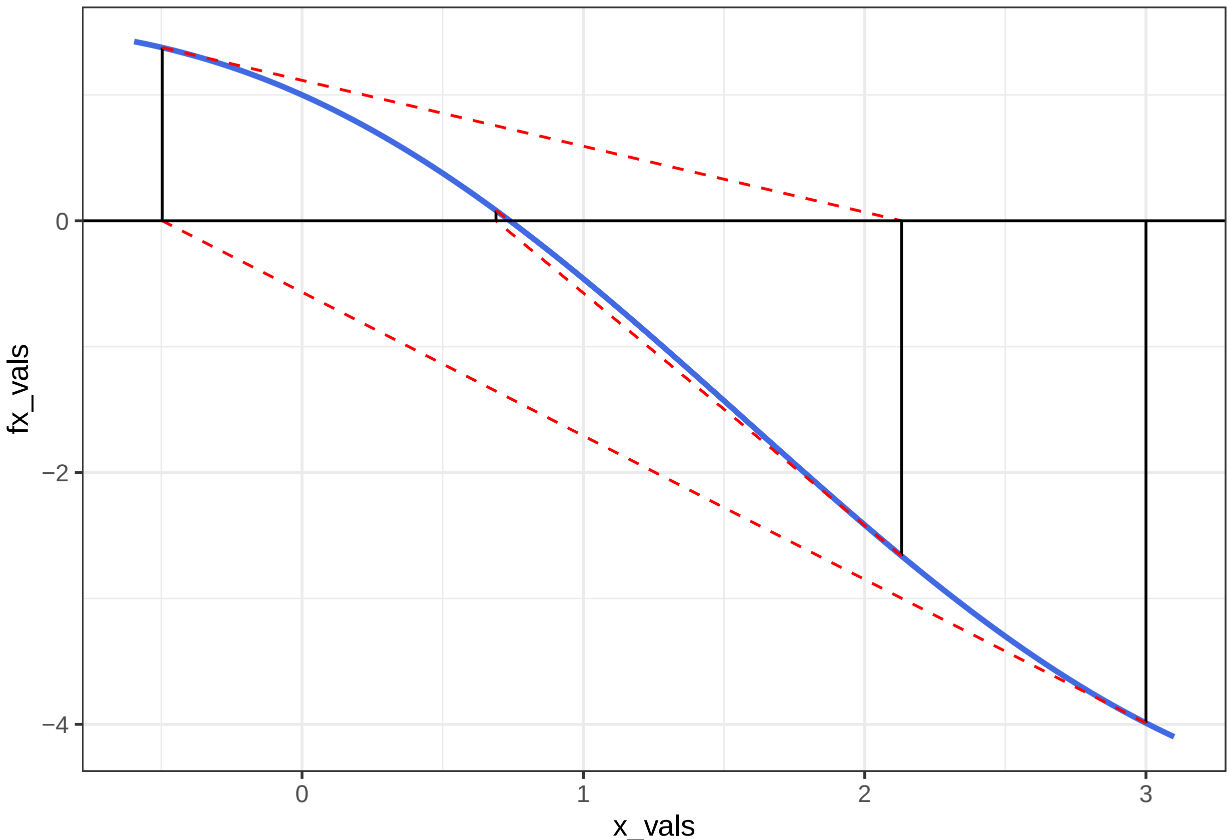
```
## Starting value is: 1
## Iteration:  1 Next x value: 0.7503639
## Iteration:  2 Next x value: 0.7391129
## Iteration:  3 Next x value: 0.7390851
## Iteration:  4 Next x value: 0.7390851
## Algrotihm converged at iteration:  4
```




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

```r
newtonraphson_show(ftna, 3, 10)
```

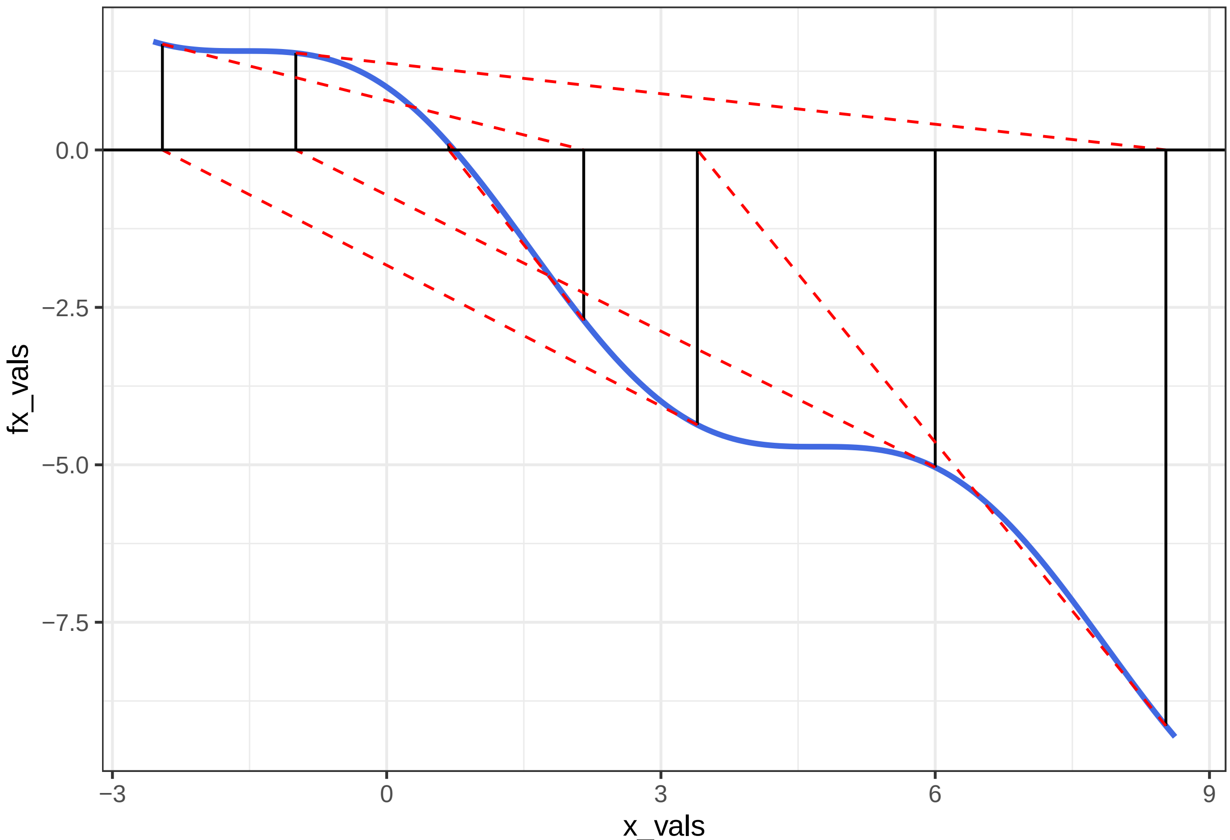
```
## Starting value is: 3
## Iteration:  1 Next x value: -0.4965582
## Iteration:  2 Next x value: 2.131004
## Iteration:  3 Next x value: 0.6896627
## Iteration:  4 Next x value: 0.739653
## Iteration:  5 Next x value: 0.7390852
## Iteration:  6 Next x value: 0.7390851
## Iteration:  7 Next x value: 0.7390851
## Algrotihm converged at iteration:  7
```




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

```r
newtonraphson_show(ftna, 6, 10)
```

```
## Starting value is: 6
## Iteration:  1 Next x value: -0.9940856
## Iteration:  2 Next x value: 8.523426
## Iteration:  3 Next x value: 3.398358
## Iteration:  4 Next x value: -2.45325
## Iteration:  5 Next x value: 2.155349
## Iteration:  6 Next x value: 0.6792118
## Iteration:  7 Next x value: 0.7399276
## Iteration:  8 Next x value: 0.7390853
## Iteration:  9 Next x value: 0.7390851
## Iteration:  10 Next x value: 0.7390851
## Algrotihm converged at iteration:  10
```

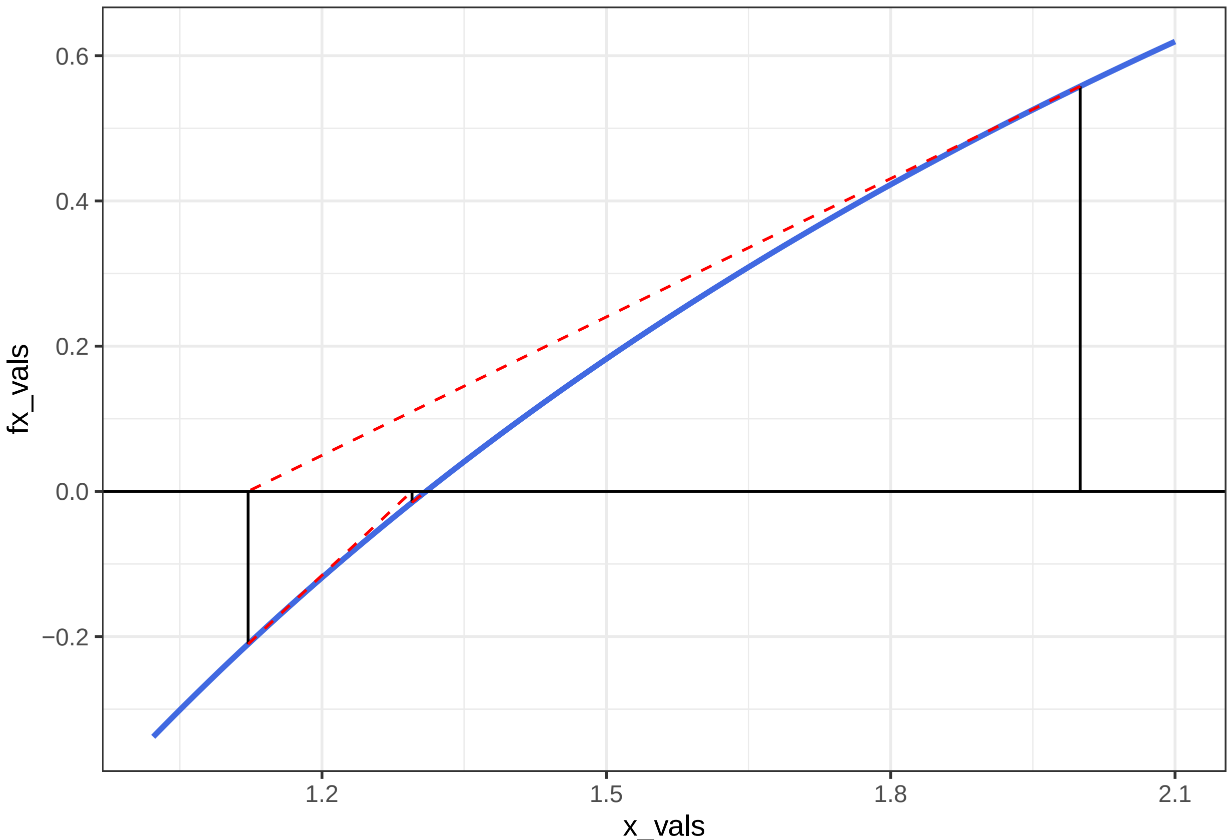



```
## [1] 0.7390851
```

part (b) using x0 = 2 Results should be similar to finding fixed point of exp(exp(-x))

```r
ftnb <- function(x) {
  fx <- log(x) - exp(-x) # f(x)
  dfx <- 1/x + exp(-x) #f'(x)
  c(fx, dfx)
}
newtonraphson_show(ftnb, 2, 10)
```

```
## Starting value is: 2
## Iteration:  1 Next x value: 1.12202
## Iteration:  2 Next x value: 1.294997
## Iteration:  3 Next x value: 1.309709
## Iteration:  4 Next x value: 1.3098
## Iteration:  5 Next x value: 1.3098
## Iteration:  6 Next x value: 1.3098
## Algrotihm converged at iteration:  6
```

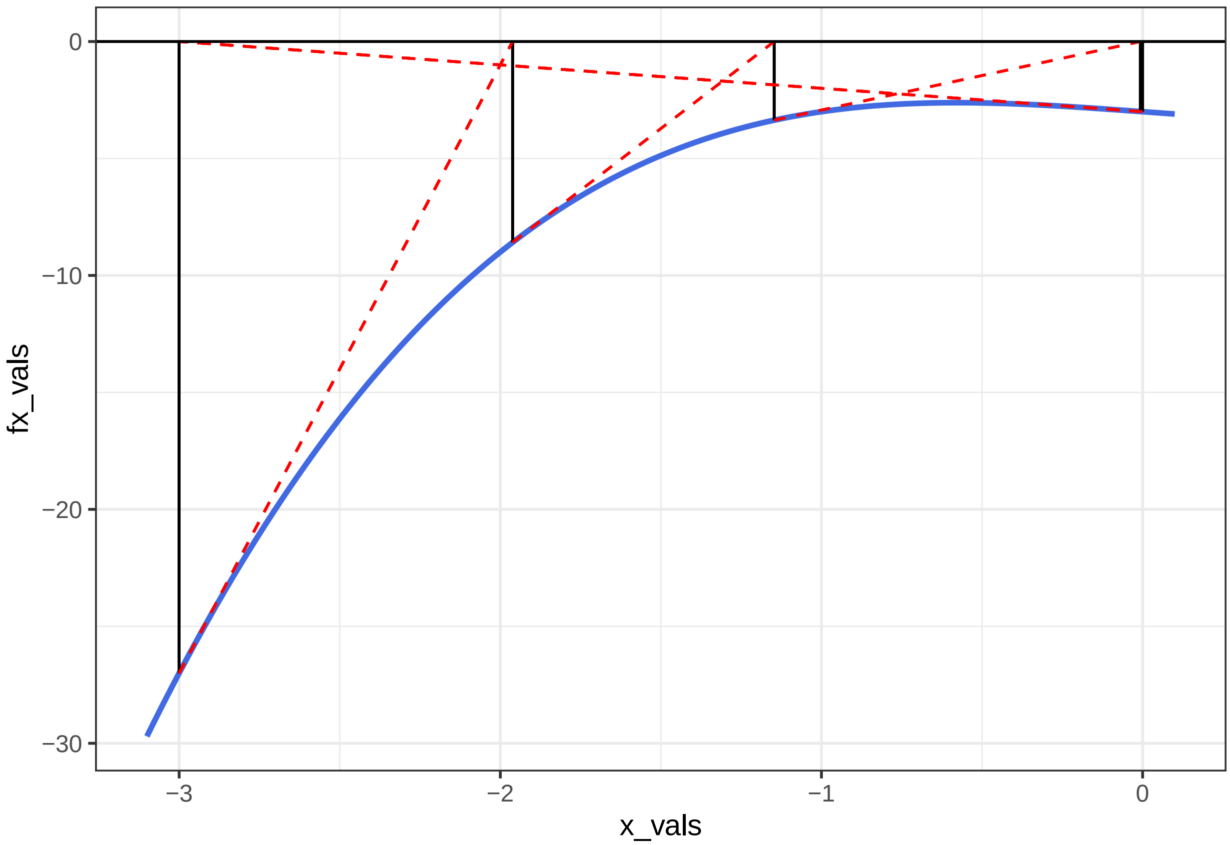



```
## [1] 1.3098
```

part (c) using x0 = 0

```r
ftnc <- function(x) {
  fx <- x^3 - x - 3 # f(x)
  dfx <- 3*x^2 - 1 # f'(x)
  c(fx, dfx)
}
newtonraphson_show(ftnc, 0, 10)
```

```
## Starting value is: 0
## Iteration:  1 Next x value: -3
## Iteration:  2 Next x value: -1.961538
## Iteration:  3 Next x value: -1.147176
## Iteration:  4 Next x value: -0.006579371
## Iteration:  5 Next x value: -3.000389
## Iteration:  6 Next x value: -1.961818
## Iteration:  7 Next x value: -1.14743
## Iteration:  8 Next x value: -0.007256248
## Iteration:  9 Next x value: -3.000473
## Iteration:  10 Next x value: -1.961879
```



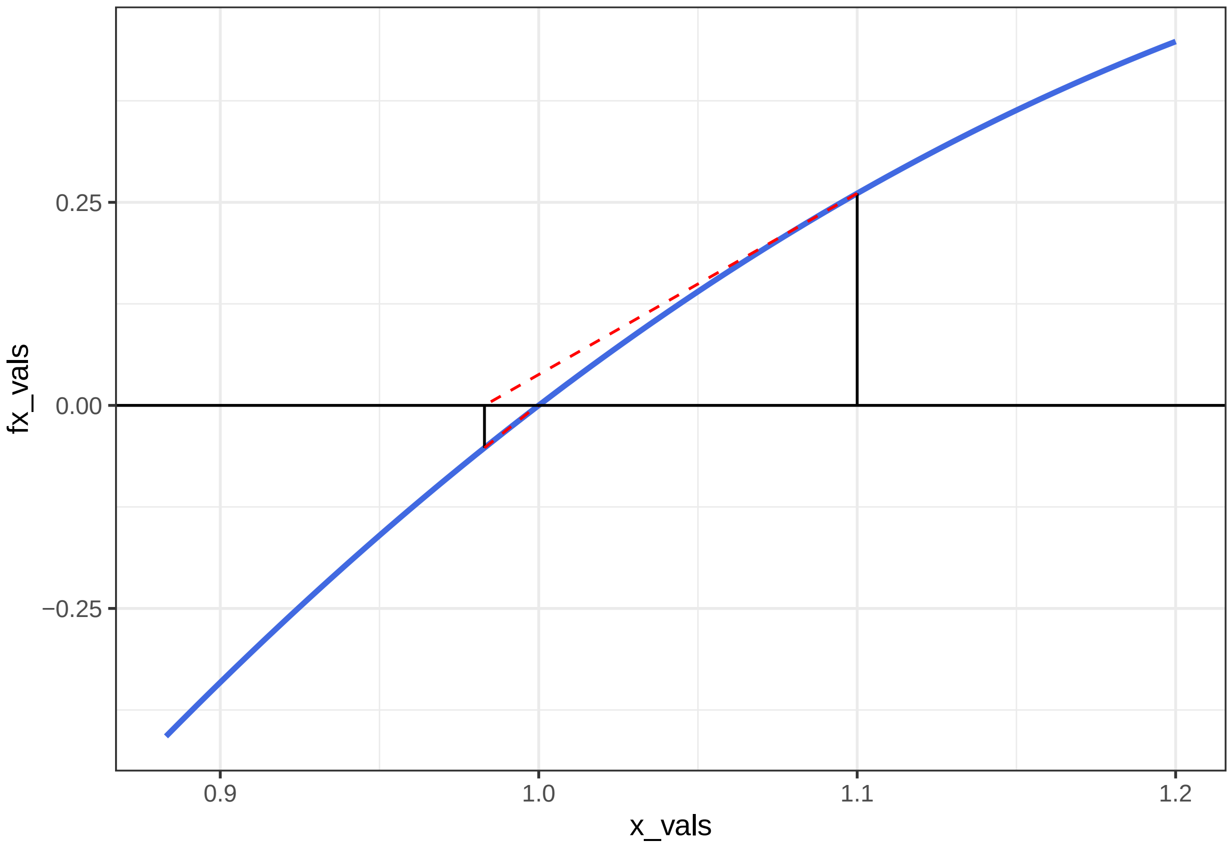

```
## Algorithm failed to converge within given iterations!
```

```
## [1] -1.961879
```

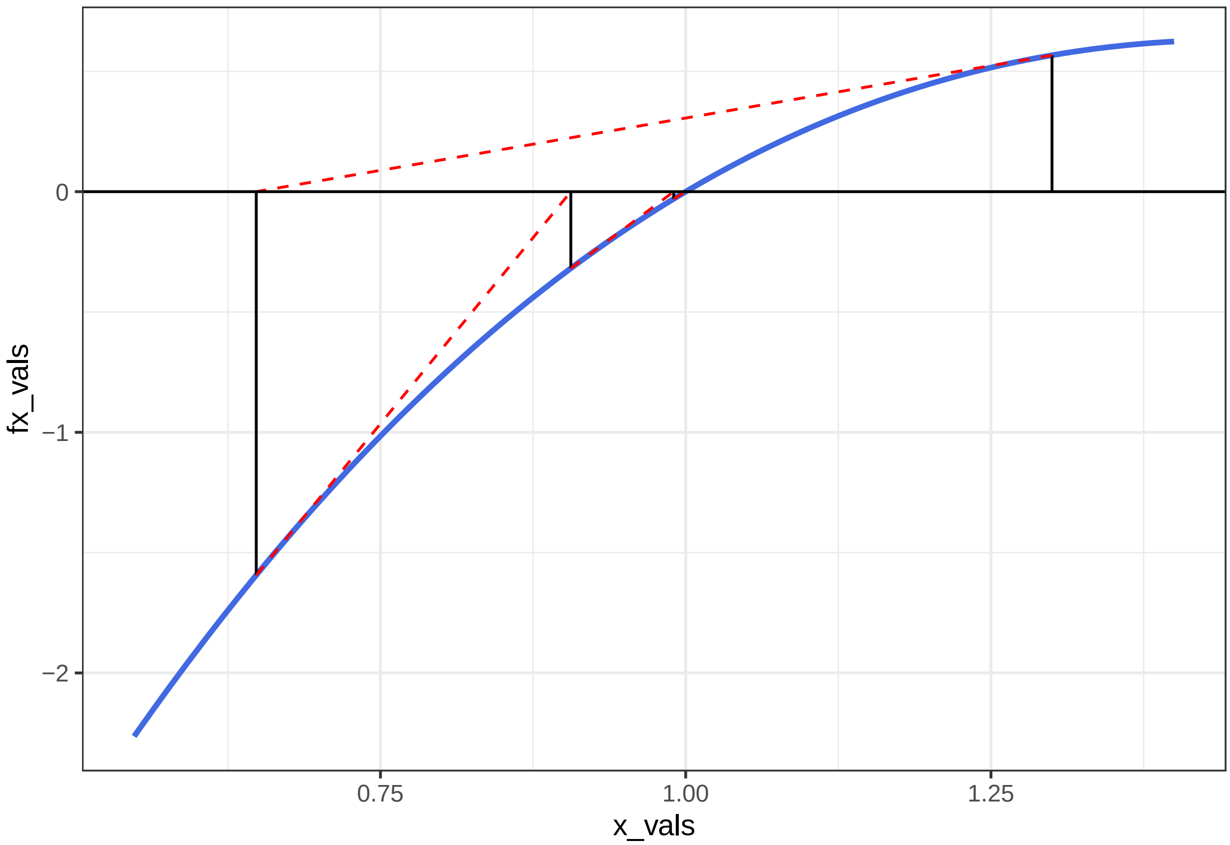
Part (d) using x0 = 1.1, 1.3, 1.4, 1.5, 1.6, 1.7 (should be simple. just repeat the command several times)

```
ftnd <- function(x) {
  fx <- x^3 - 7*x^2 + 14*x -8 # f(x)
  dfx <- 3*x^2-14*x+14 # f'(x)
  c(fx, dfx)
}
d_vals <- c(1.1, 1.3, 1.4, 1.5, 1.6, 1.7)
for (i in d_vals) {
  cat("\nx0 = ", i, "\n")
  newtonraphson_show(ftnd, i, iter = 10)
}
```

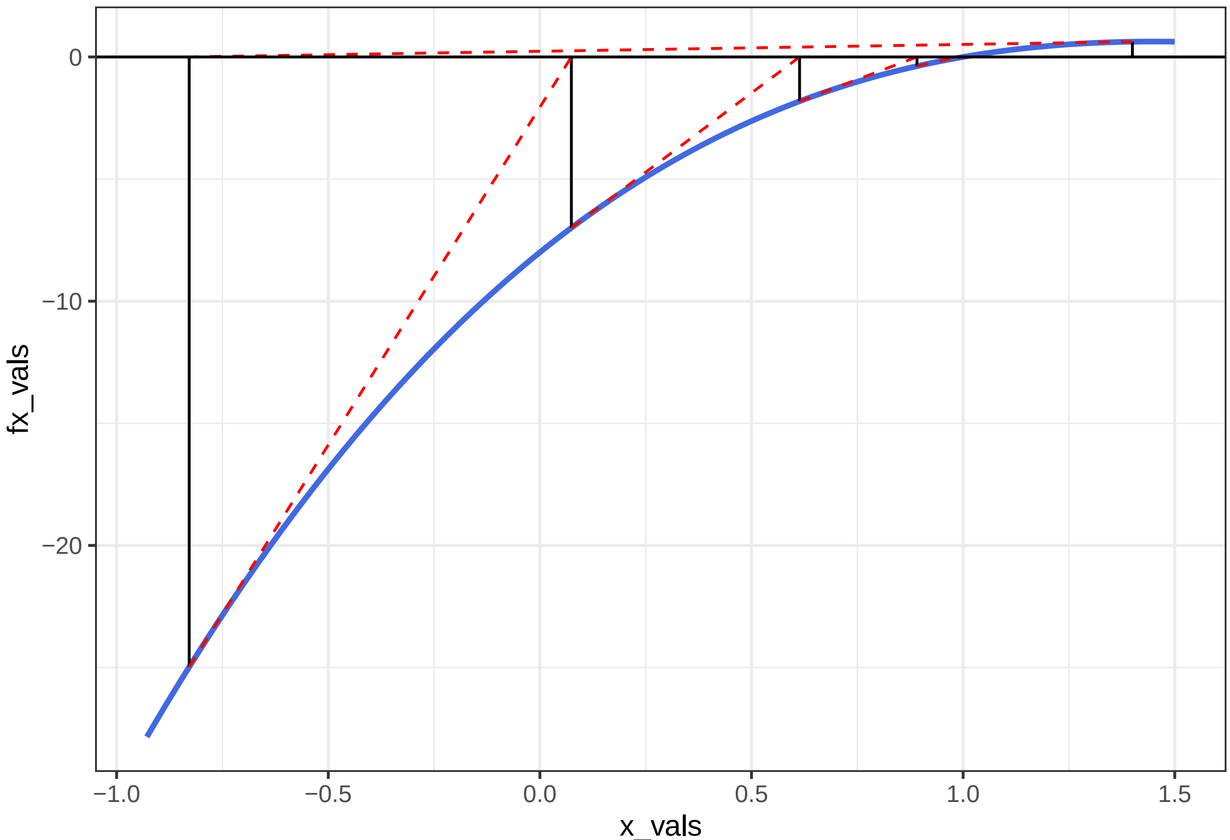
```
##
## x0 =  1.1
## Starting value is: 1.1
## Iteration:  1 Next x value: 0.9829596
## Iteration:  2 Next x value: 0.9996266
## Iteration:  3 Next x value: 0.9999998
## Iteration:  4 Next x value: 1
## Iteration:  5 Next x value: 1
## Algrotihm converged at iteration:  5
```



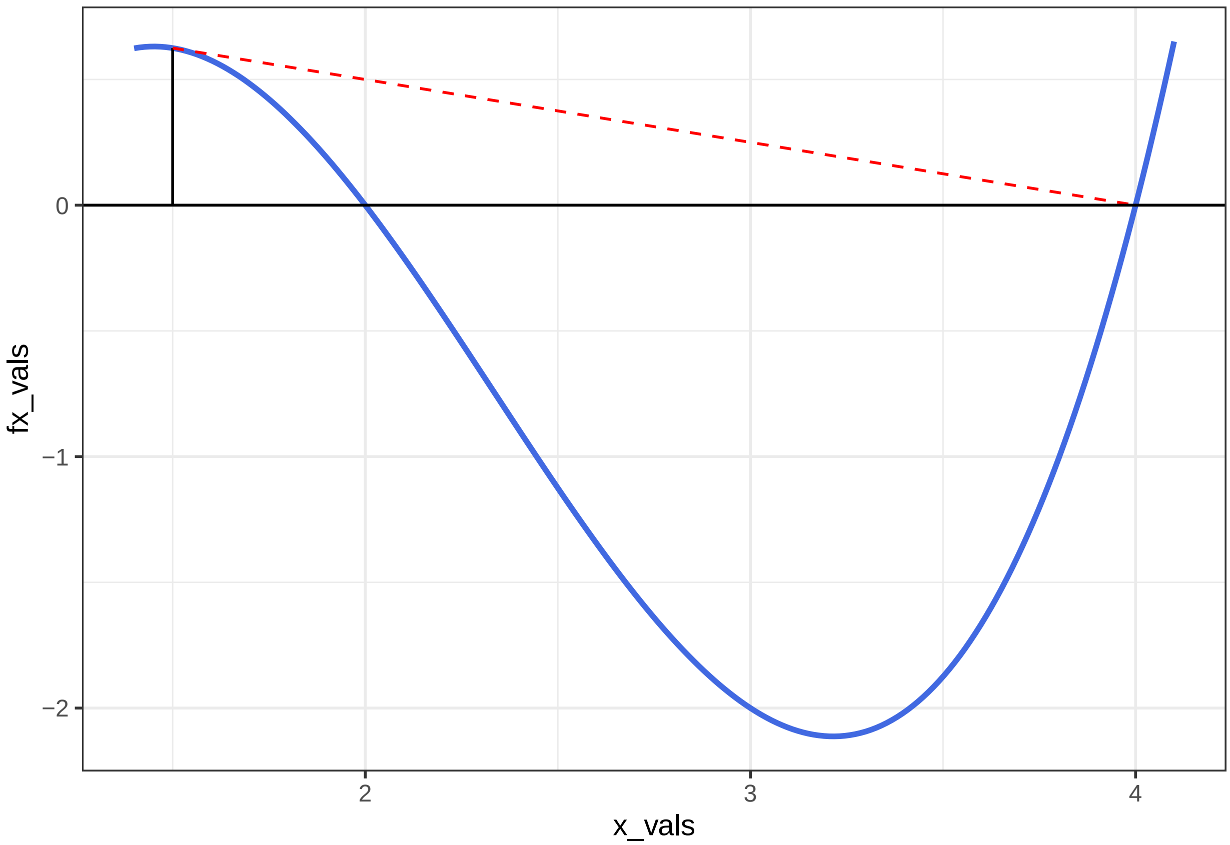

```
##
## x0 =  1.3
## Starting value is: 1.3
## Iteration:  1 Next x value: 0.6482759
## Iteration:  2 Next x value: 0.9059224
## Iteration:  3 Next x value: 0.9901916
## Iteration:  4 Next x value: 0.9998744
## Iteration:  5 Next x value: 1
## Iteration:  6 Next x value: 1
## Iteration:  7 Next x value: 1
## Algrotihm converged at iteration:  7
```



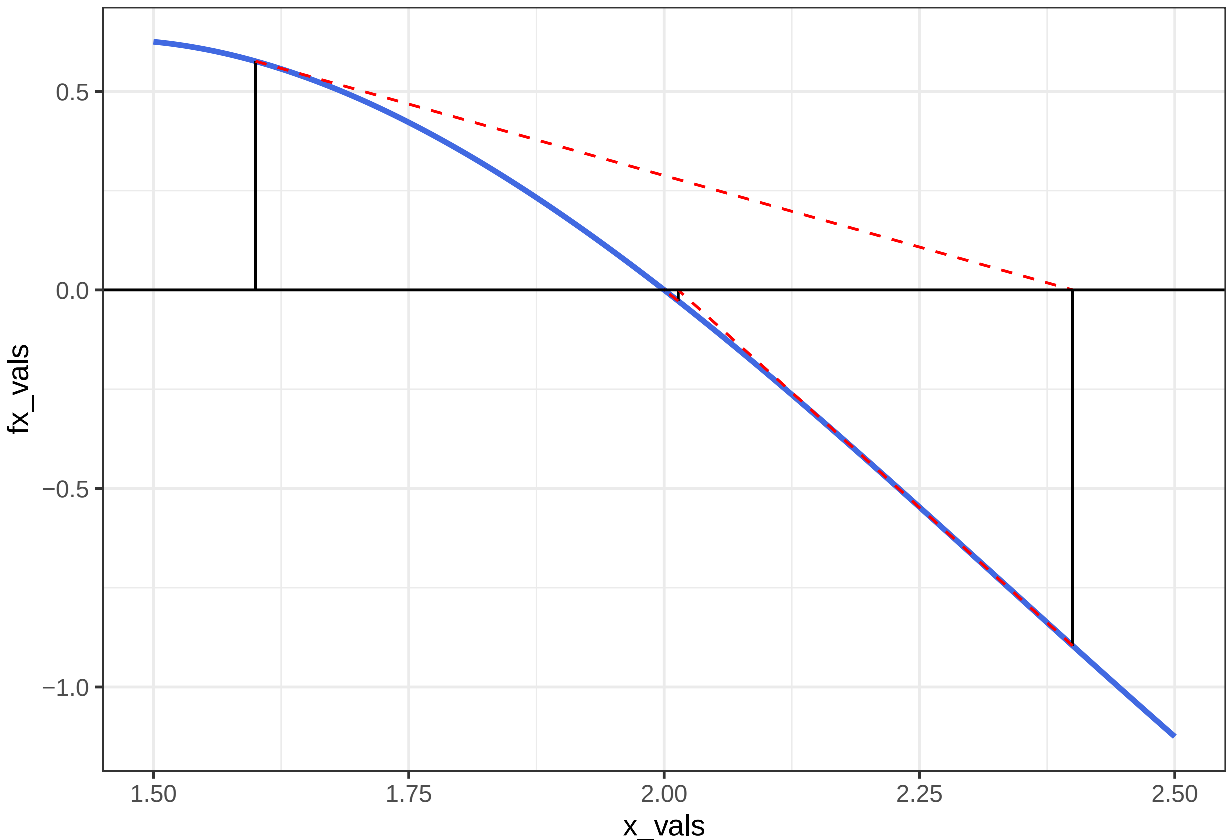

```
##
## x0 =  1.4
## Starting value is: 1.4
## Iteration:  1 Next x value: -0.8285714
## Iteration:  2 Next x value: 0.07435419
## Iteration:  3 Next x value: 0.6136214
## Iteration:  4 Next x value: 0.891034
## Iteration:  5 Next x value: 0.9871826
## Iteration:  6 Next x value: 0.9997869
## Iteration:  7 Next x value: 0.9999999
## Iteration:  8 Next x value: 1
## Iteration:  9 Next x value: 1
## Algrotihm converged at iteration:  9
```



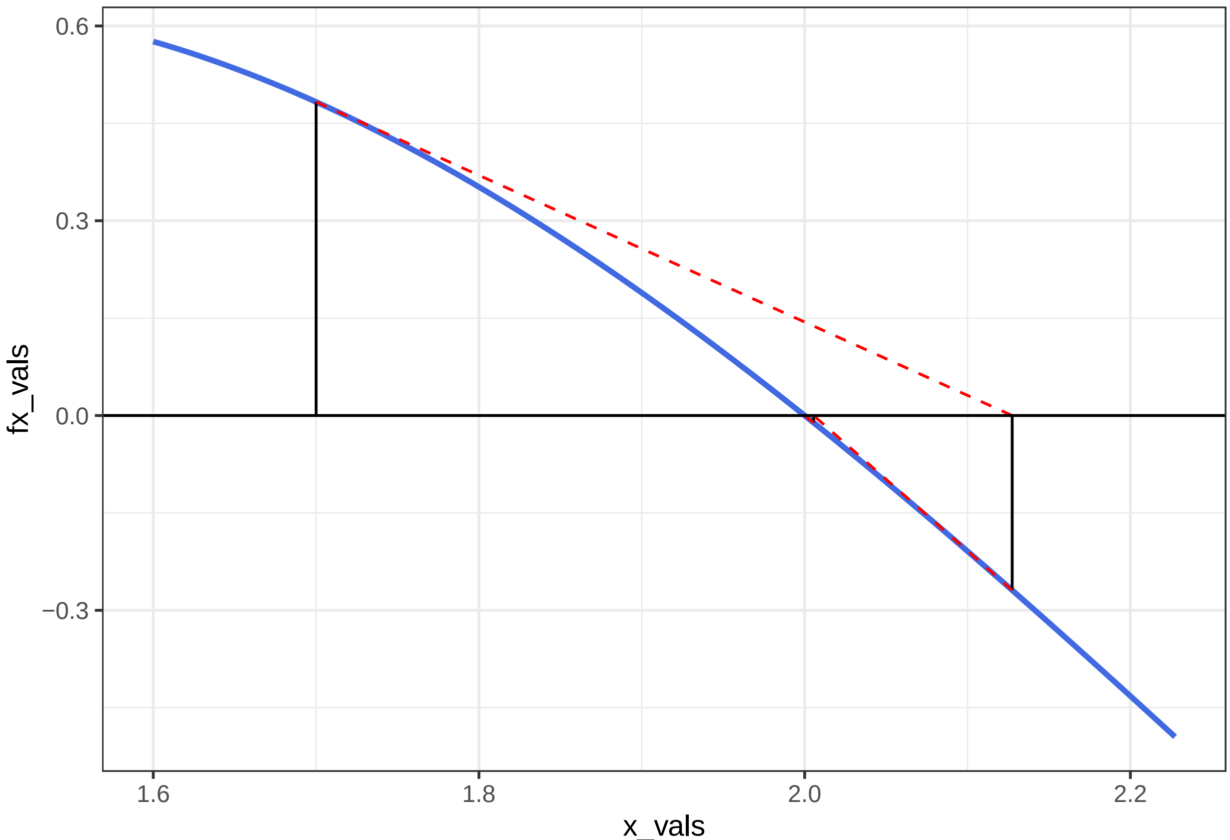

```
##
## x0 =  1.5
## Starting value is: 1.5
## Iteration:  1 Next x value: 4
## Iteration:  2 Next x value: 4
## Algrotihm converged at iteration:  2
```




```
##
## x0 =  1.6
## Starting value is: 1.6
## Iteration:  1 Next x value: 2.4
## Iteration:  2 Next x value: 2.013793
## Iteration:  3 Next x value: 2.000091
## Iteration:  4 Next x value: 2
## Iteration:  5 Next x value: 2
## Iteration:  6 Next x value: 2
## Algrotihm converged at iteration:  6
```




```
##
## x0 =  1.7
## Starting value is: 1.7
## Iteration:  1 Next x value: 2.127434
## Iteration:  2 Next x value: 2.005485
## Iteration:  3 Next x value: 2.000015
## Iteration:  4 Next x value: 2
## Iteration:  5 Next x value: 2
## Algrotihm converged at iteration:  5
```



4. Root Finding with Secant Method [24 points- 20 points for completing the code. 1 pts each graph]

- NOTES: Secant Method:

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

  - Stop when $|f(x_n)| \leq \epsilon$, where $\epsilon > 0$ is the pre-sepcified tolerance level.

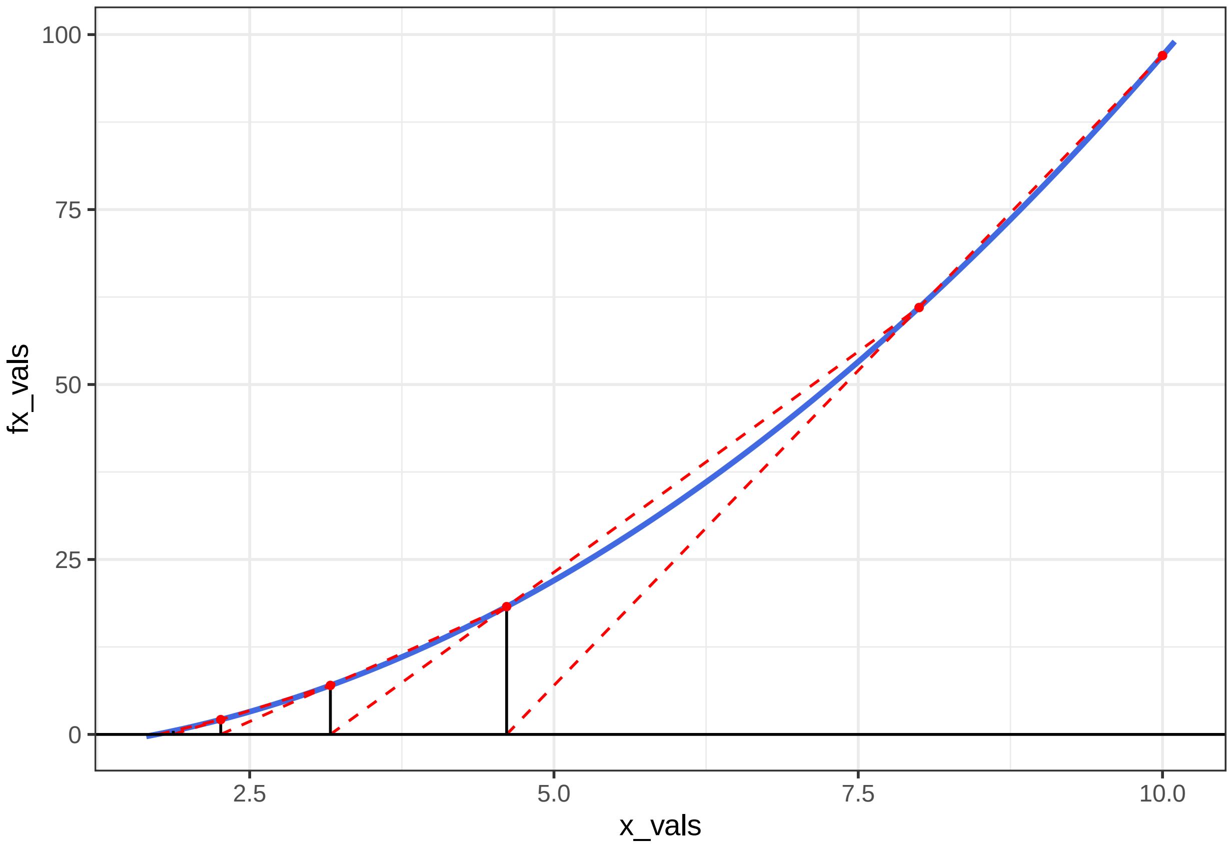
```
secant_show <- function(ftn, x0, x1, iter = 5) {
  # Initializers
  df_points <- data.frame(
    x1 = numeric(0),
    y1 = numeric(0),
    x2 = numeric(0),
    y2 = numeric(0),
    type = character(0))
  xolder <- x0
  xold <- x1
  xnew <- numeric(0)
  tol <- 1e-9
  cat("Starting values are:", " x0 = ", xolder, ", x1 = ", xold, "\n")

  # the algorithm
  for(i in 1:iter){
    f_xolder <- ftn(xolder)
    f_xold <- ftn(xold)
    xnew <- xold - f_xold * (xold - xolder) / (f_xold - f_xolder)
```

```r
    cat("Iteration: ", i, " Next x value:", xnew, "\n")

    # the line segments. You will need to replace the NAs with the appropriate         values
    xleft <- min(xolder, xold, xnew)
    xright <- max(xolder, xold, xnew)
    f_xleft <- c(0, f_xolder, f_xold)[which(xleft == c(xnew, xolder, xold))] # Get f(x) of the leftside
    f_xright <- c(0, f_xolder, f_xold)[which(xright == c(xnew, xolder, xold))] # Get f(x) of the rights
    # Secant line:
    df_points <- rbind(df_points, data.frame(x1 = xleft, y1 = f_xleft, x2 = xright, y2 = f_xright, type
    # Vertical line:
    df_points <- rbind(df_points, data.frame(x1 = xnew, y1 = 0, x2 = xnew, y2 = ftn(xnew), type = "verti
    # Check tolerance like textbook ex
    if (abs(f_xold) < tol) {
      cat("Algrotihm converged at iteration: ", i, "\n")
      break
    }
    # Update values
    xolder <- xold
    xold <- xnew
  }

  # Plot range
  plot_start <- min(df_points$x1, df_points$x2, x0) - 0.1
  plot_end <- max(df_points$x1, df_points$x2, x0) + 0.1

  # Calculate value of the function fx for all x
  x_vals <- seq(plot_start, plot_end, length.out = 200)
  fx_vals <- sapply(x_vals, function(x) ftn(x))

  p <- ggplot() +
    geom_line(aes(x = x_vals, y = fx_vals), color = "royalblue", linewidth = 1) +  # plot the function
    geom_segment(data = df_points[df_points$type == "secant", ], aes(x = x1, y = y1, xend = x2, yend = y
    geom_segment(data = df_points[df_points$type == "vertical", ], aes(x = x1, y = y1, xend = x2, yend =
    geom_abline(intercept = 0, slope = 0) + # plot the line y = 0
    geom_point(data = df_points[df_points$type == "secant", ], aes(x = x2, y = y2), color = "red", size
    theme_bw()

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

Example from lecture $x^2 - 3, x_0 = 10, x_1 = 8$

```r
lec_ex <- function(x) x^2 - 3
secant_show(lec_ex, 10, 8, iter = 5)
```

```
## Starting values are:  x0 =  10 , x1 =  8
## Iteration:  1  Next x value: 4.611111
## Iteration:  2  Next x value: 3.162996
## Iteration:  3  Next x value: 2.261986
## Iteration:  4  Next x value: 1.871832
## Iteration:  5  Next x value: 1.74997
```



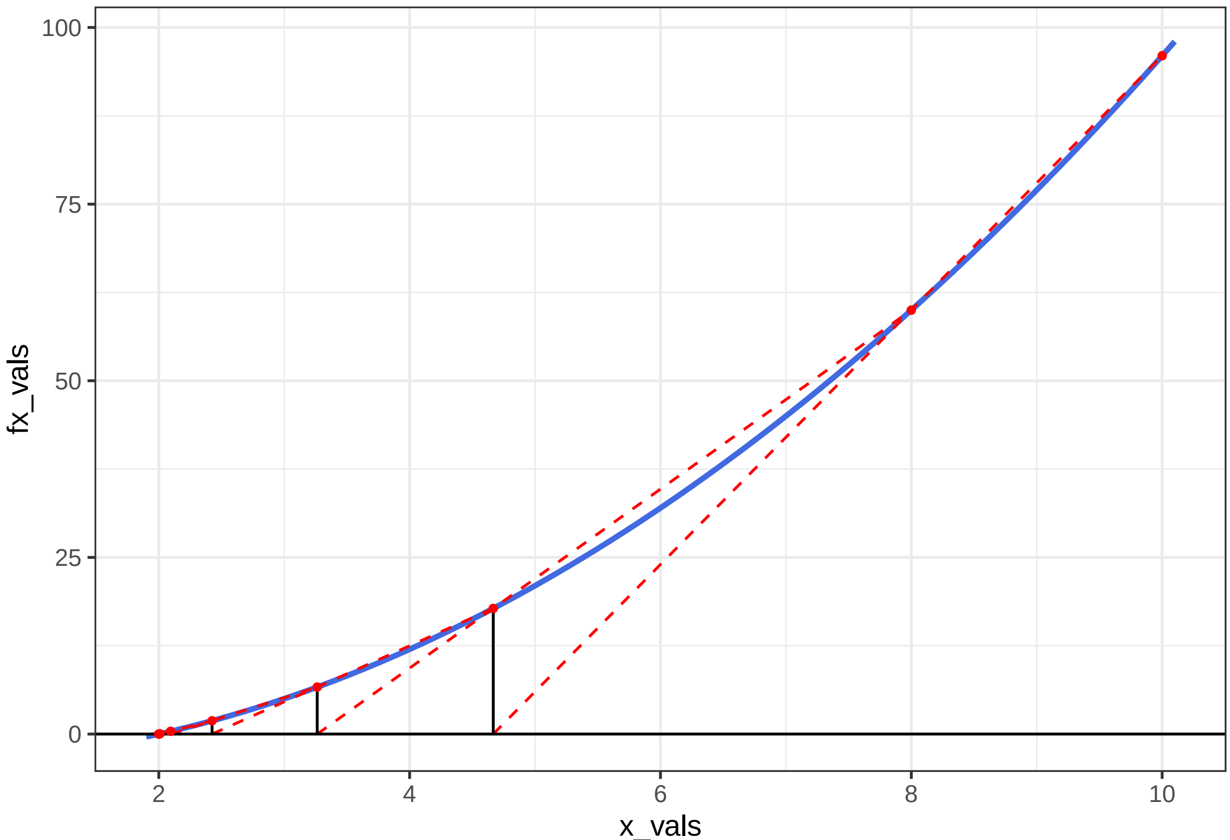

```
## [1] 1.74997
```

Produce graphs for:

The function $f(x) = x^2 - 4$ using x0 = 10, and x1 = 8

```r
ftna <- function (x) x^2 - 4
secant_show(ftna, 10, 8, iter = 10)
```

```
## Starting values are:  x0 =  10 , x1 =  8
## Iteration:  1  Next x value: 4.666667
## Iteration:  2  Next x value: 3.263158
## Iteration:  3  Next x value: 2.424779
## Iteration:  4  Next x value: 2.094333
## Iteration:  5  Next x value: 2.008867
## Iteration:  6  Next x value: 2.000204
## Iteration:  7  Next x value: 2
## Iteration:  8  Next x value: 2
## Iteration:  9  Next x value: 2
## Algrotihm converged at iteration:  9
```

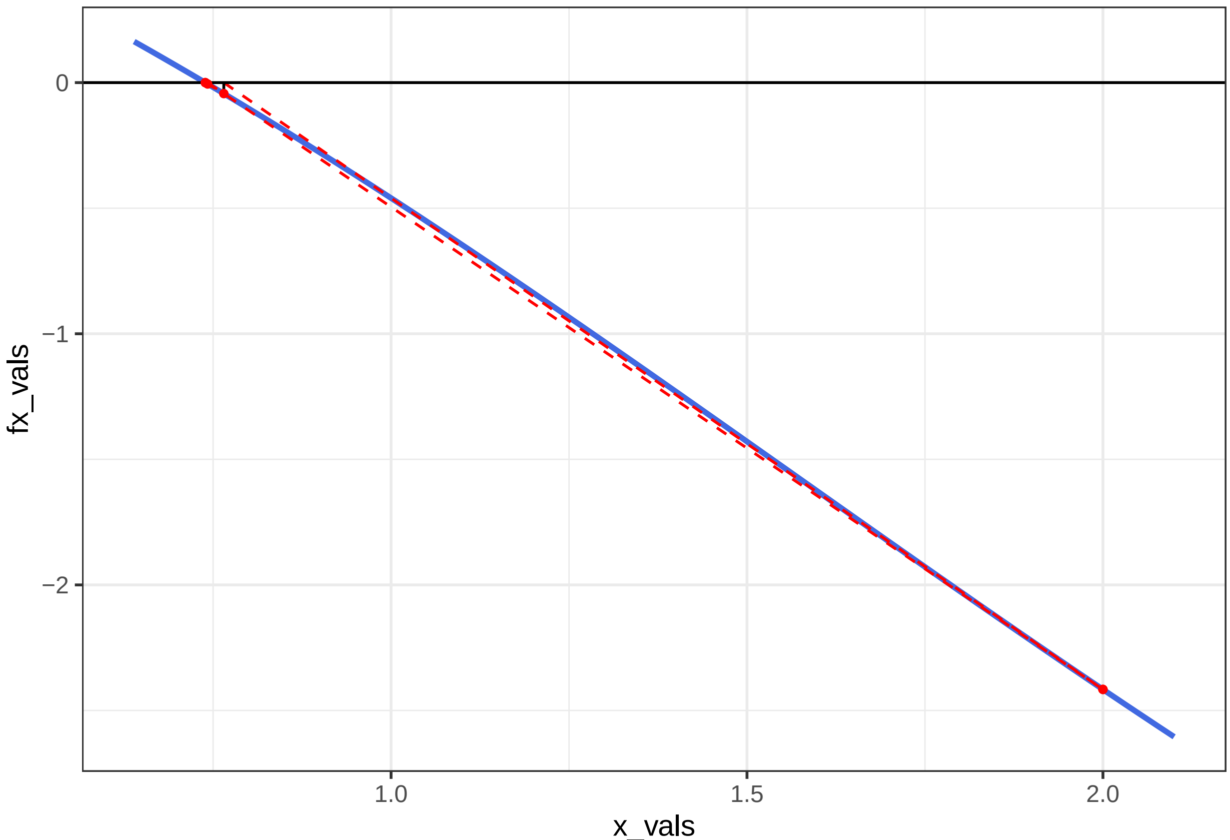



```
## [1] 2
```

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

```
ftnb <- function(x) cos(x) - x
secant_show(ftnb, 1, 2, iter = 10)
```

```
## Starting values are:  x0 =  1 , x1 =  2
## Iteration:  1  Next x value: 0.7650347
## Iteration:  2  Next x value: 0.7422994
## Iteration:  3  Next x value: 0.7391033
## Iteration:  4  Next x value: 0.7390851
## Iteration:  5  Next x value: 0.7390851
## Iteration:  6  Next x value: 0.7390851
## Algrotihm converged at iteration:  6
```

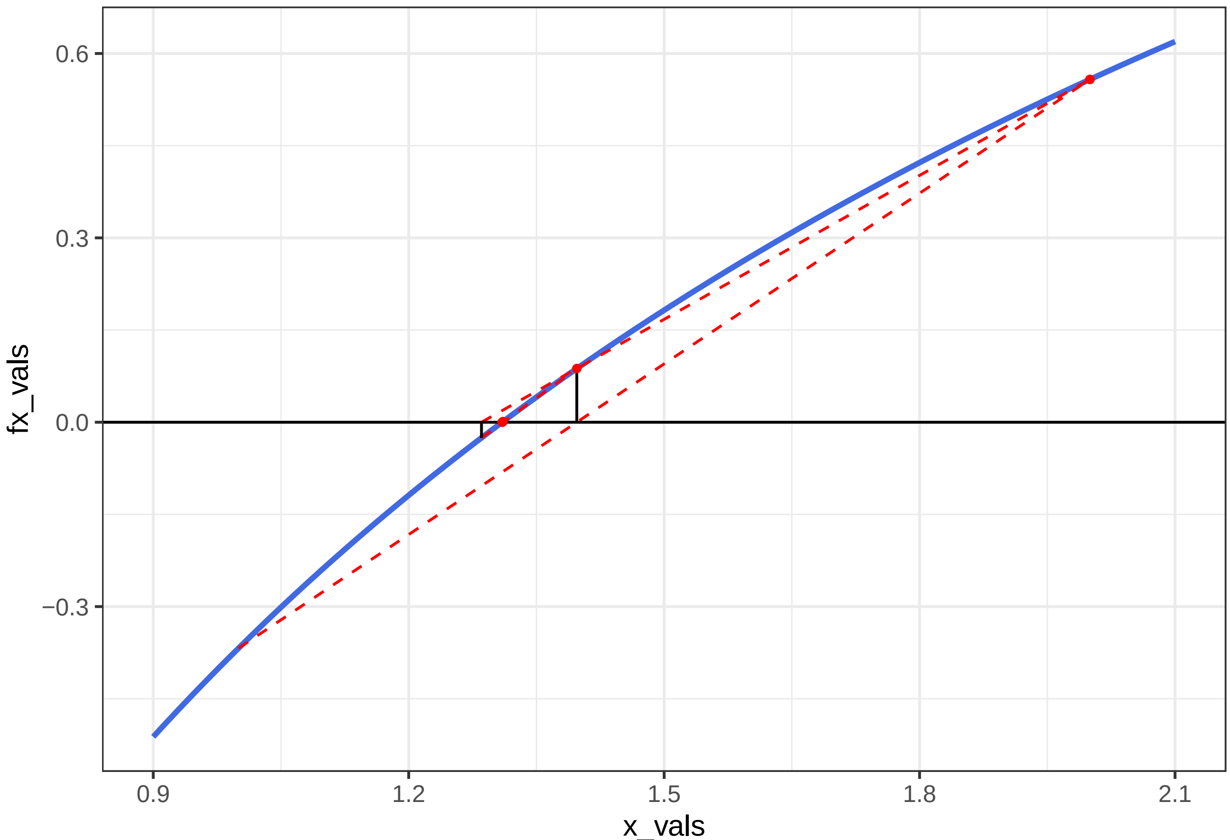



```
## [1] 0.7390851
```

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

```
ftnc <- function(x) log(x) - exp(-x)
secant_show(ftnc, 1, 2, iter = 10)
```

```
## Starting values are:  x0 =  1 , x1 =  2
## Iteration:  1  Next x value: 1.39741
## Iteration:  2  Next x value: 1.285476
## Iteration:  3  Next x value: 1.310677
## Iteration:  4  Next x value: 1.309808
## Iteration:  5  Next x value: 1.3098
## Iteration:  6  Next x value: 1.3098
## Iteration:  7  Next x value: 1.3098
## Algrotihm converged at iteration:  7
```

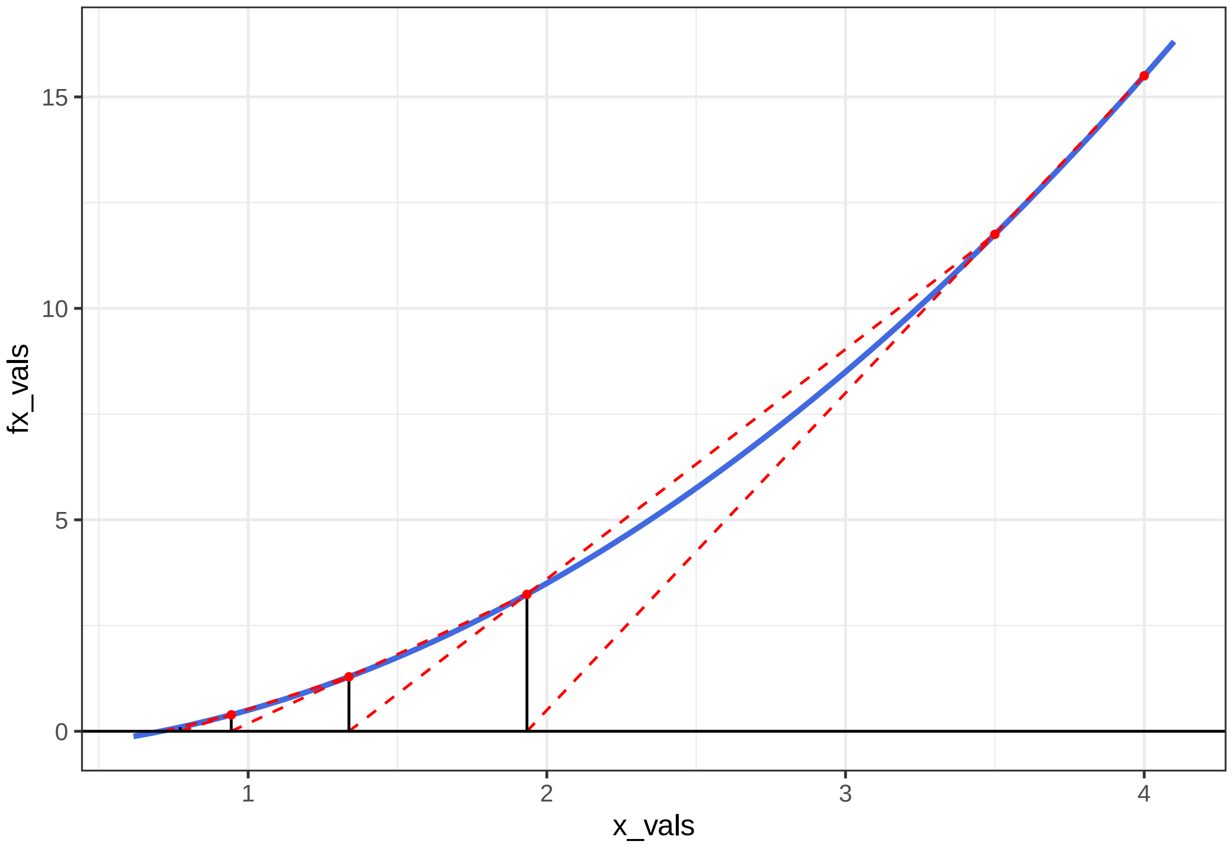



```
## [1] 1.3098
```

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

```r
ftnd <- function(x) x^2 - 0.5
secant_show(ftnd, 4, 3.5, iter = 5)
```

```
## Starting values are:  x0 =  4 , x1 =  3.5
## Iteration:  1  Next x value: 1.933333
## Iteration:  2  Next x value: 1.337423
## Iteration:  3  Next x value: 0.9434163
## Iteration:  4  Next x value: 0.7724116
## Iteration:  5  Next x value: 0.7161008
```




```
## [1] 0.7161008
```

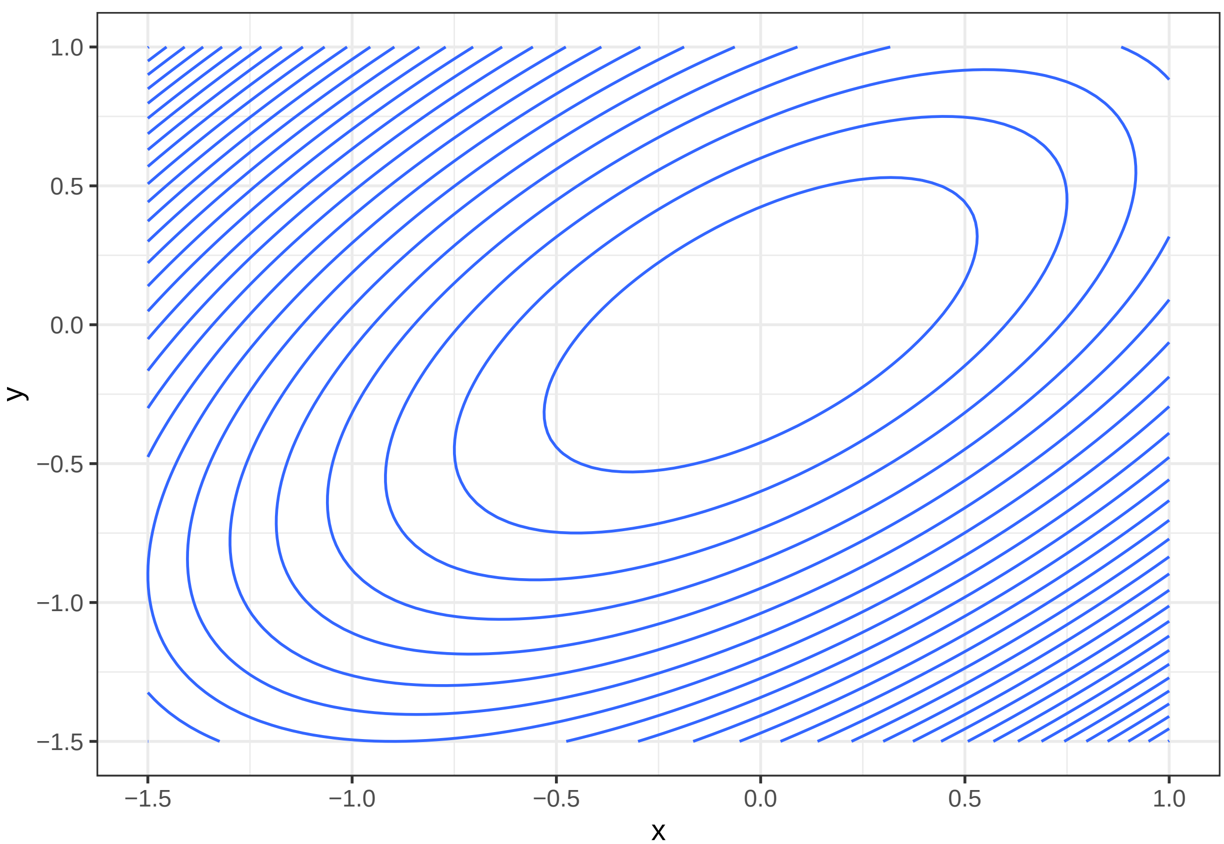
5. Coordinate Descent Algorithm for Optimization [20 points]

```r
##### 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)
```

```r
        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
}
```

```r
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
contour_df <- data.frame(
  x = rep(x, each = 100),
  y = rep(y, 100),
  z = outer(x, y, g)[1:100^2]
  )
ggplot(contour_df, aes(x = x, y = y, z = z)) +
  geom_contour(binwidth = 0.9) +
  theme_bw()
```

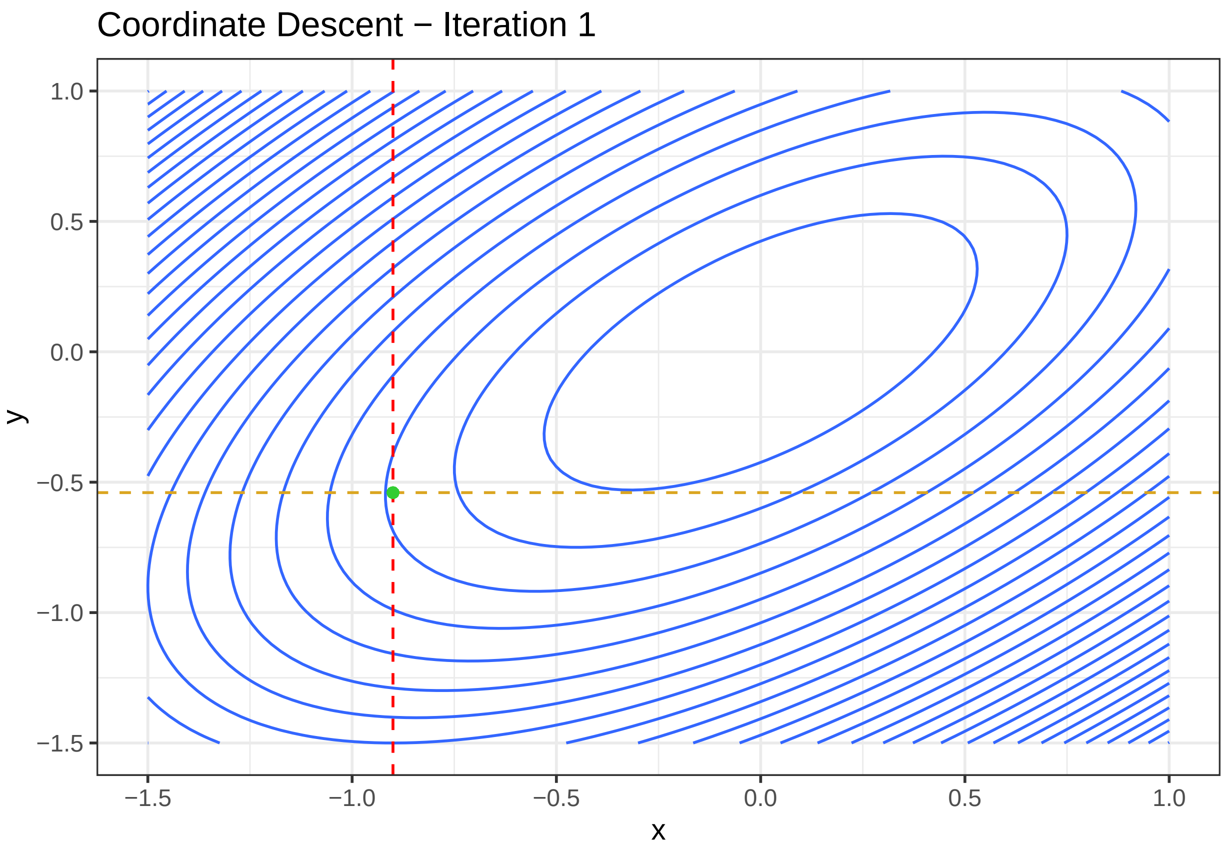


```r
# write your code here
x_i <- -1.5
y_i <- -1.5
```

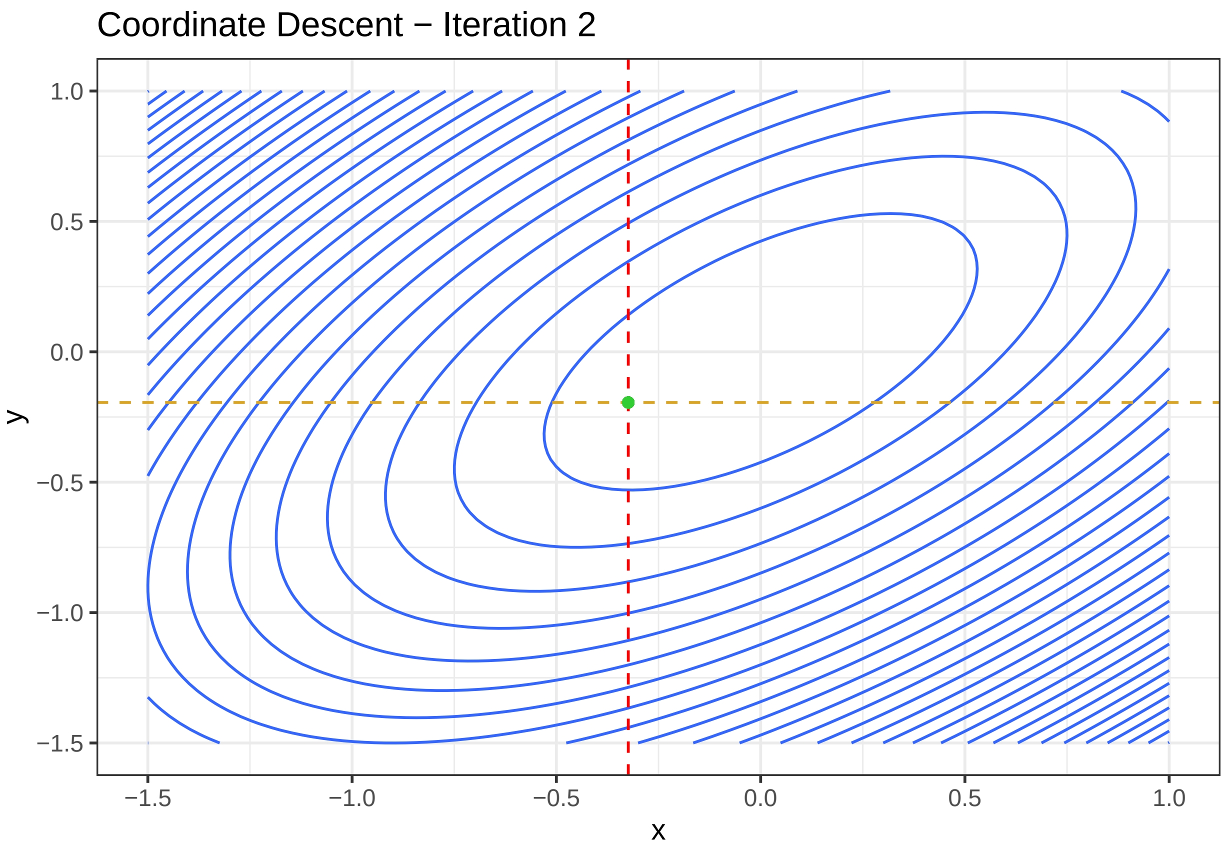
Graph for starting point x = -1.5, and y = -1.5.

```r
tolerance <- 1e-5
for (i in 1:15) {
  # Update x while holding y constant
  f_x <- function(x) g(x, y_i)
  xnew <- golden(f_x, -1.5, 1.5)
  # Update y while holding x constant
  f_y <- function(y) g(xnew, y)
  ynew <- golden(f_y, -1.5, 1.5)
  cat(sprintf("Iteration %d: x = %.5f, y = %.5f\n", i, xnew, ynew))
  # Plot the segments
  p <- ggplot(contour_df, aes(x = x, y = y, z = z)) +
    ggtitle(sprintf("Coordinate Descent - Iteration %d", i)) +
    theme(plot.title = element_text(hjust = 0.5)) +
    geom_contour(binwidth = 0.9) +
    geom_vline(xintercept = xnew, lty = 2, color = "red") +
    geom_hline(yintercept = ynew, lty = 2, color = "goldenrod") +
    geom_point(x = xnew, y = ynew, color = "limegreen") +
    theme_bw()
  print(p)
  # Check convergence
  if (i > 1 && abs(xnew - x_i) < tolerance) {
    cat("Converges!\n")
    break
  }
  # Update initial values
  x_i <- xnew
  y_i <- ynew
}
```

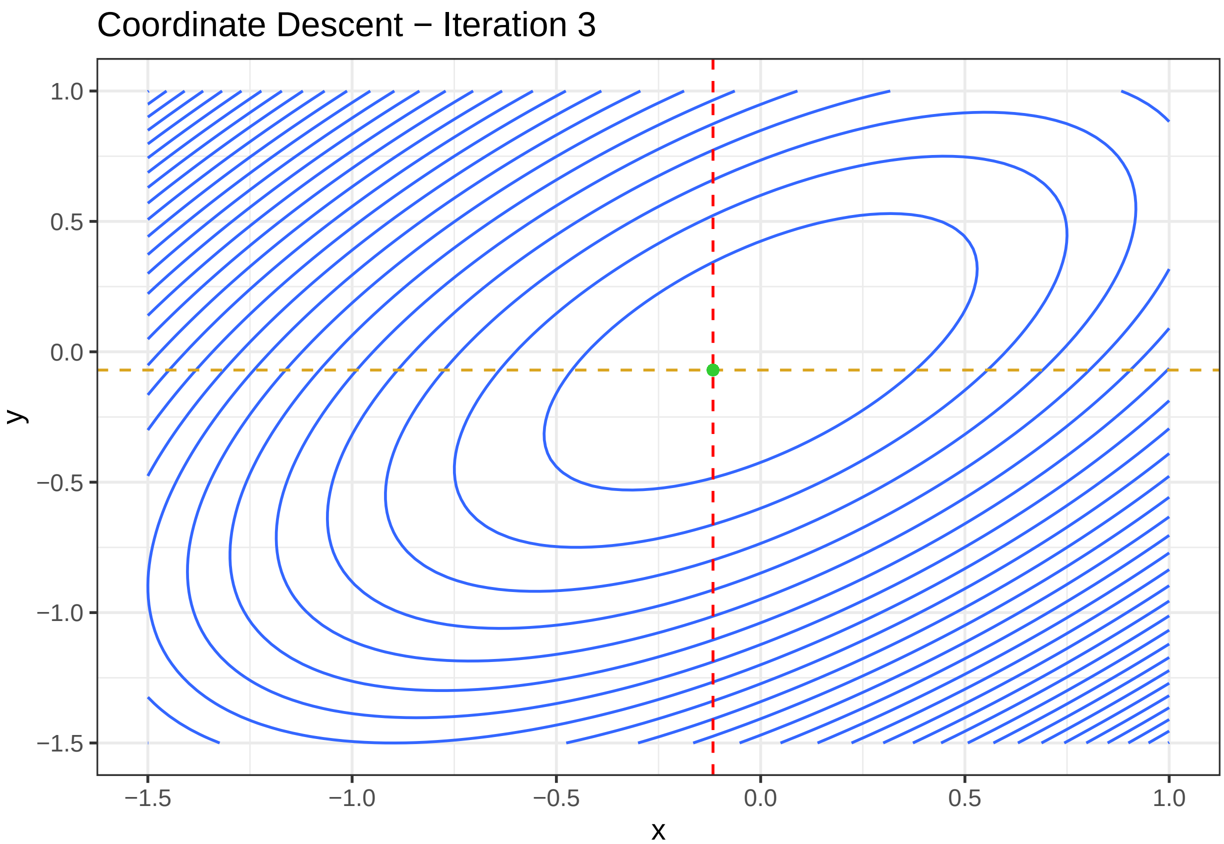
```
## Iteration 1: x = -0.90000, y = -0.54000
```



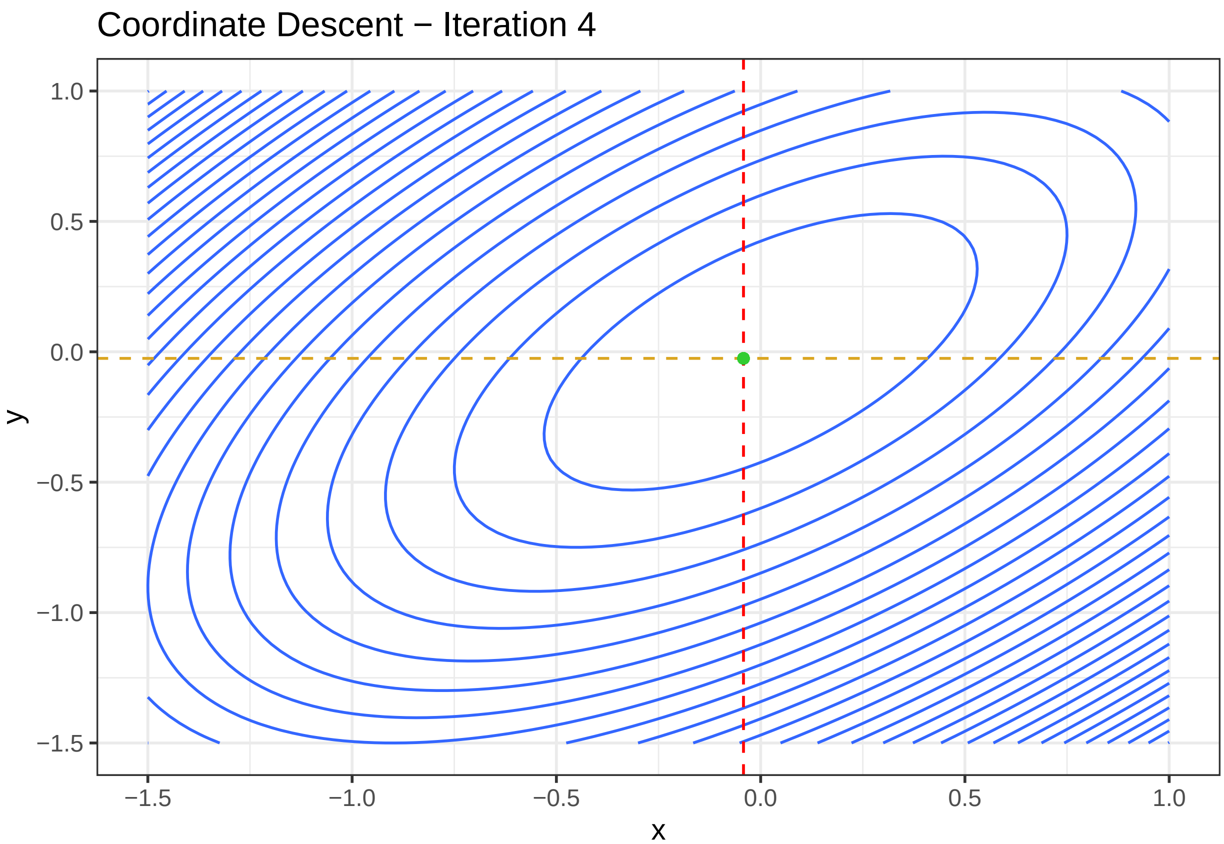

```
## Iteration 2: x = -0.32400, y = -0.19440
```



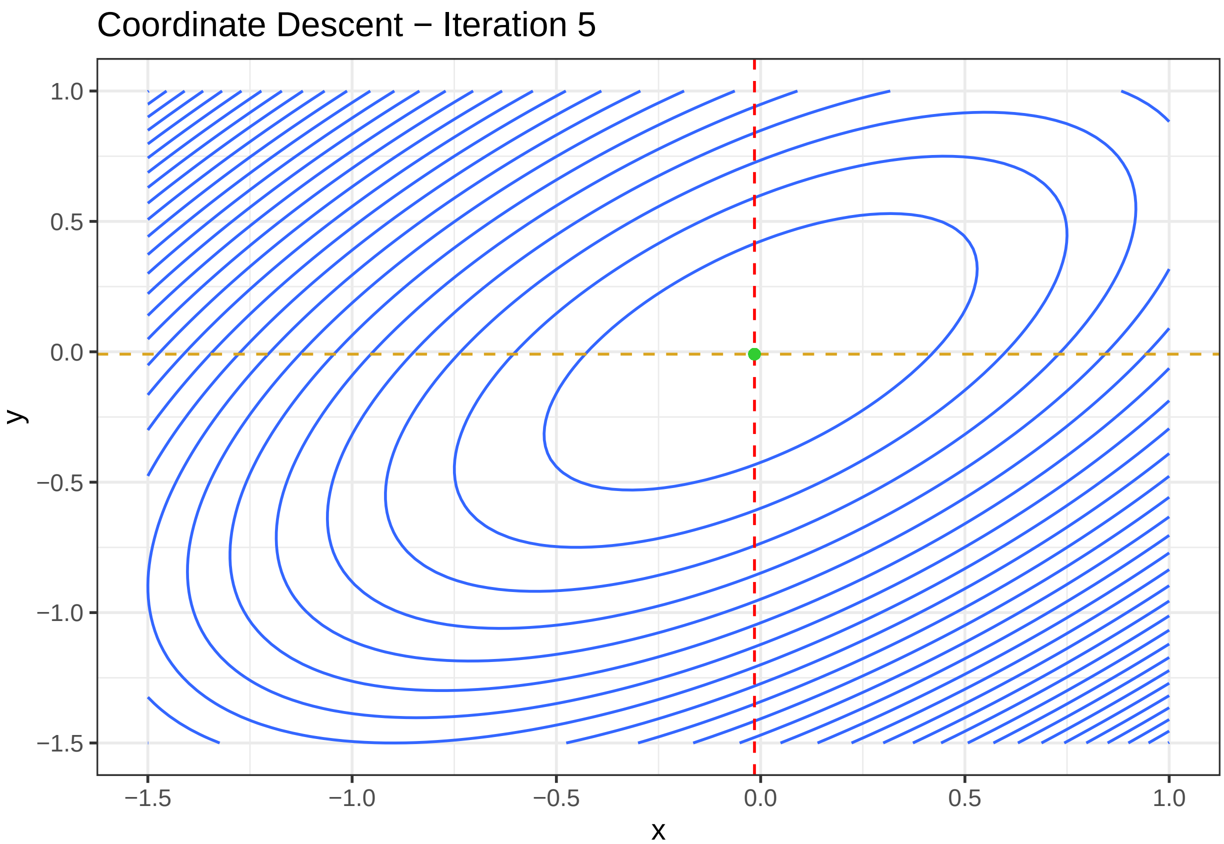

```
## Iteration 3: x = -0.11664, y = -0.06998
```



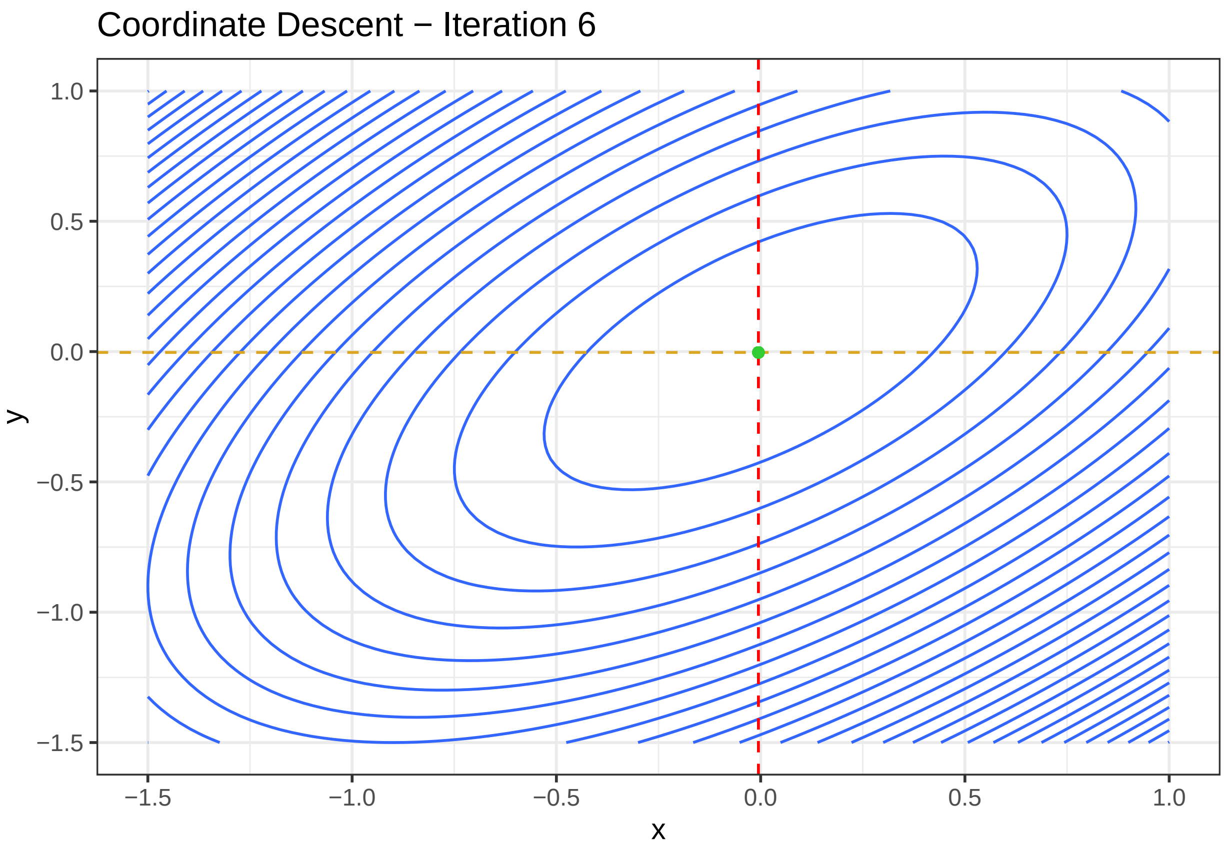

```
## Iteration 4: x = -0.04199, y = -0.02520
```



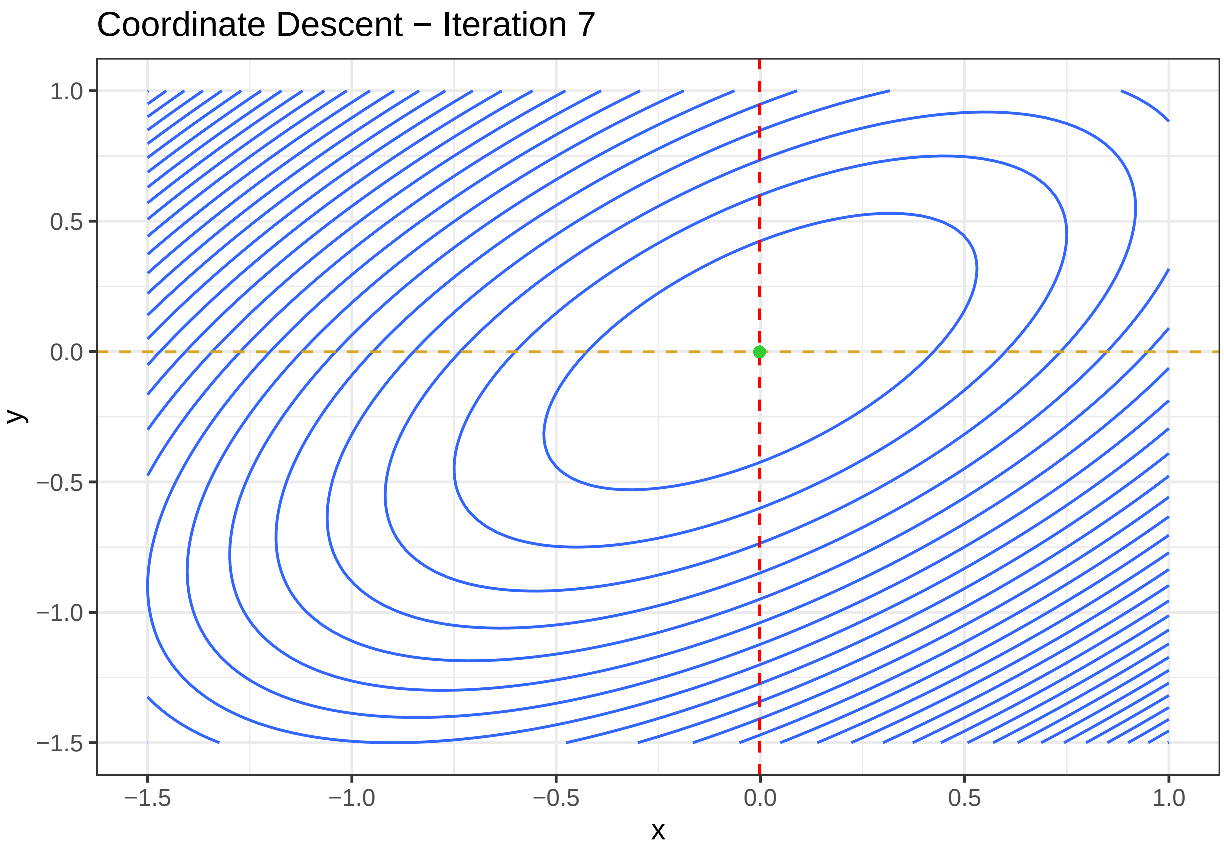

```
## Iteration 5: x = -0.01512, y = -0.00907
```



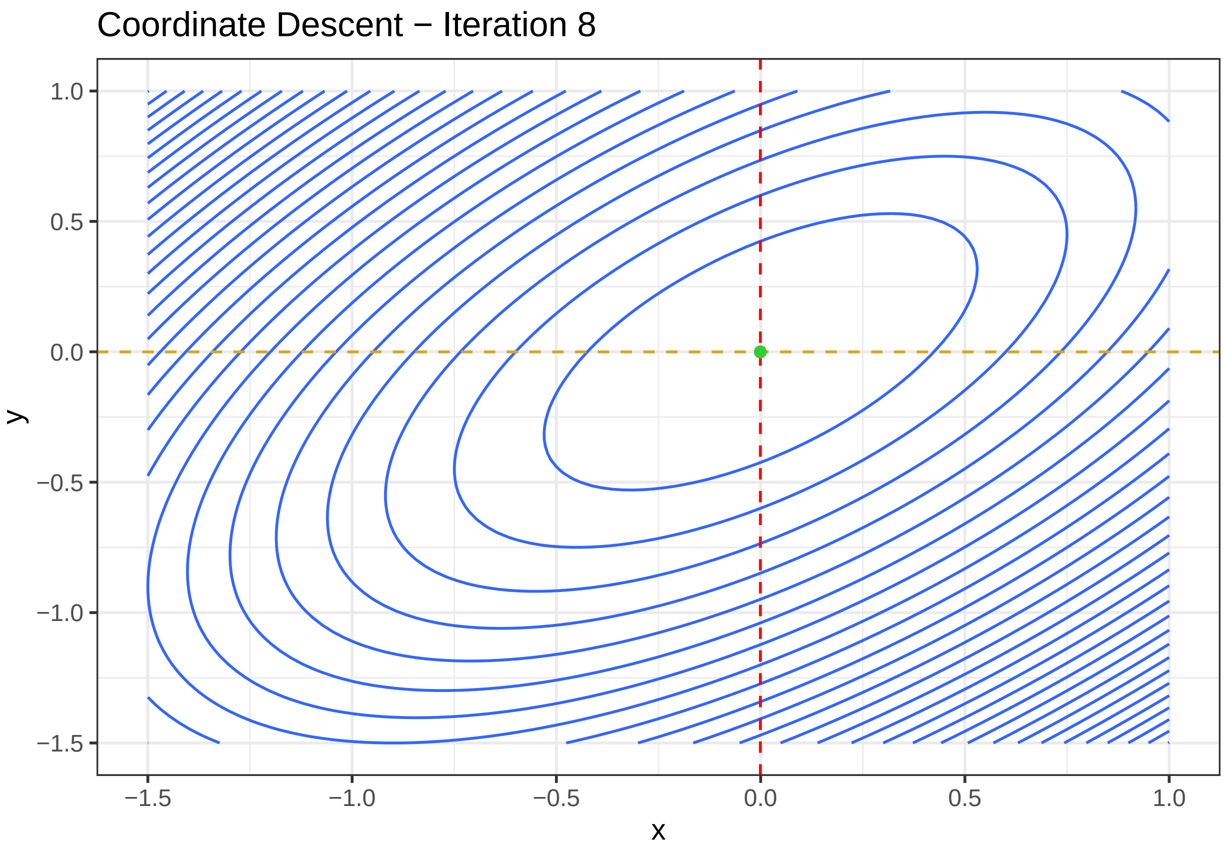

```
## Iteration 6: x = -0.00544, y = -0.00327
```



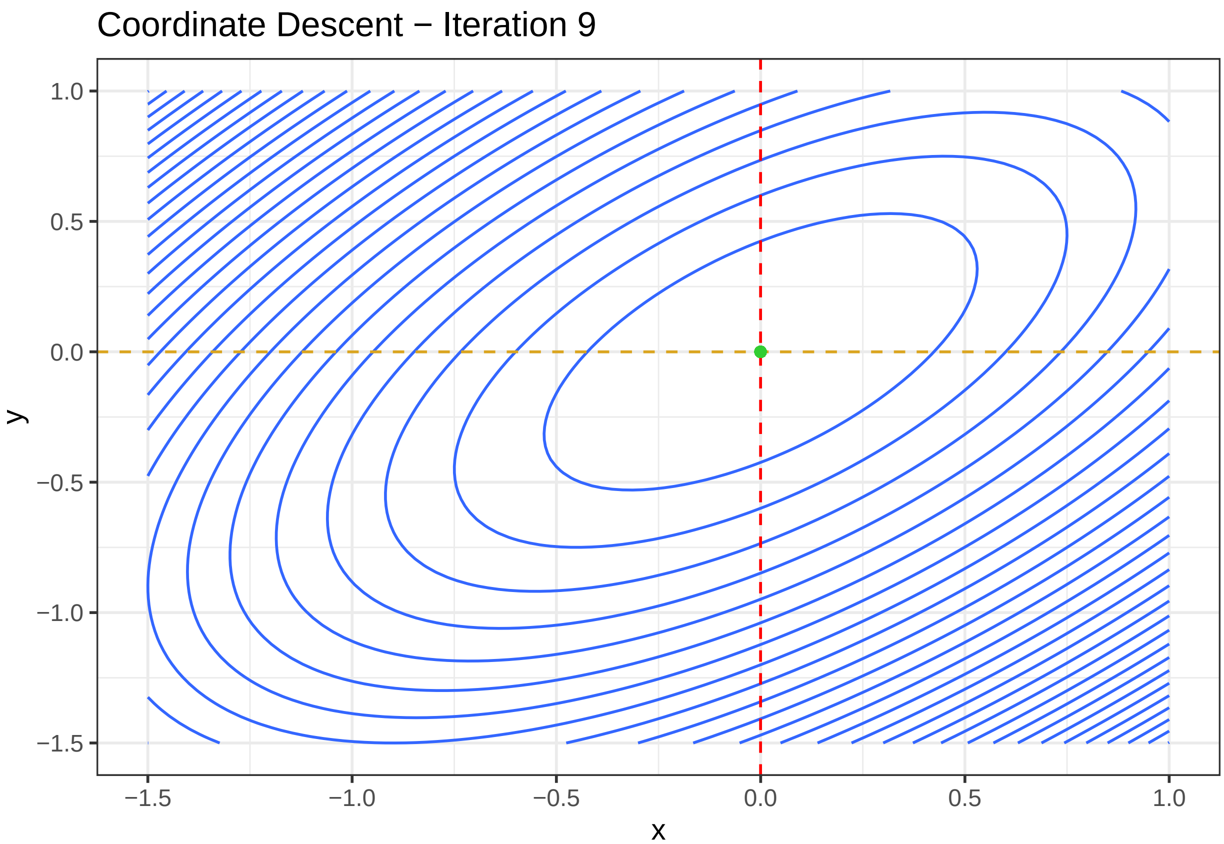

```
## Iteration 7: x = -0.00196, y = -0.00117
```



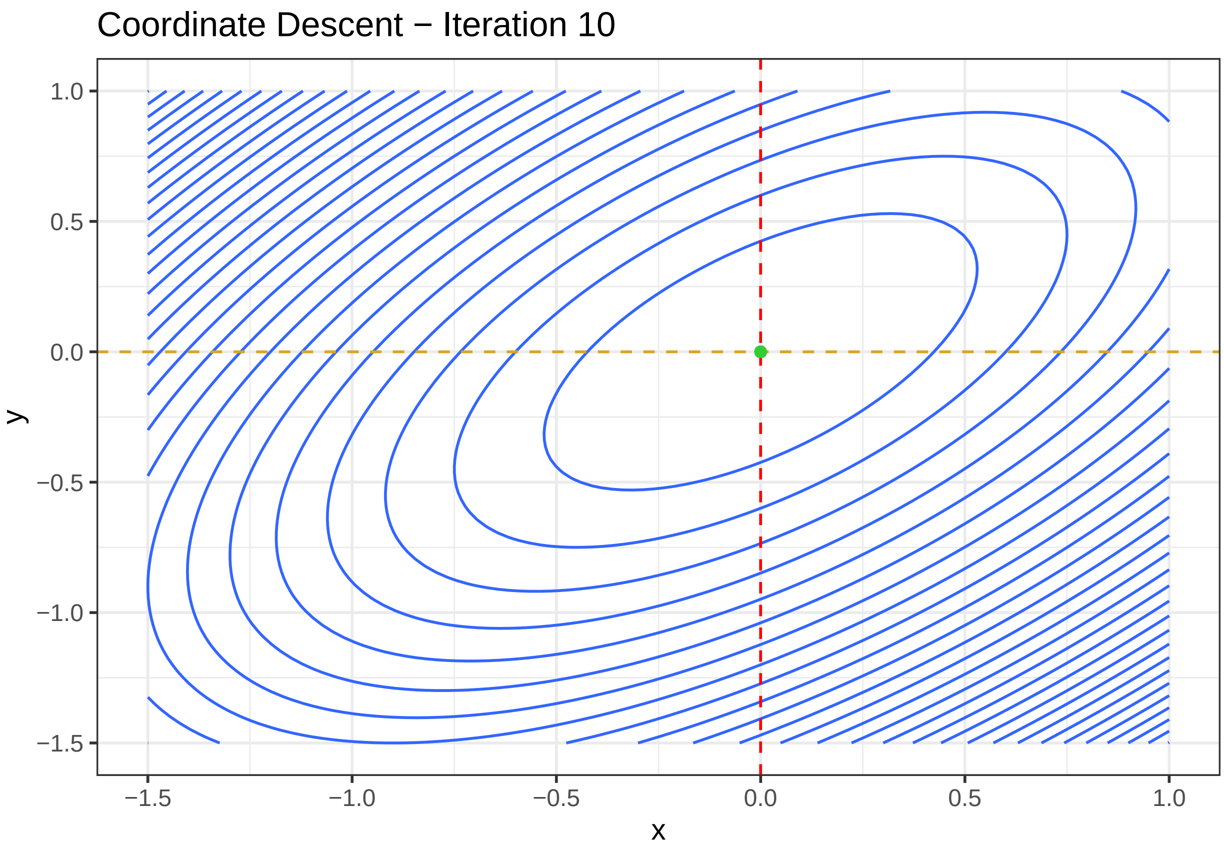

```
## Iteration 8: x = -0.00070, y = -0.00042
```



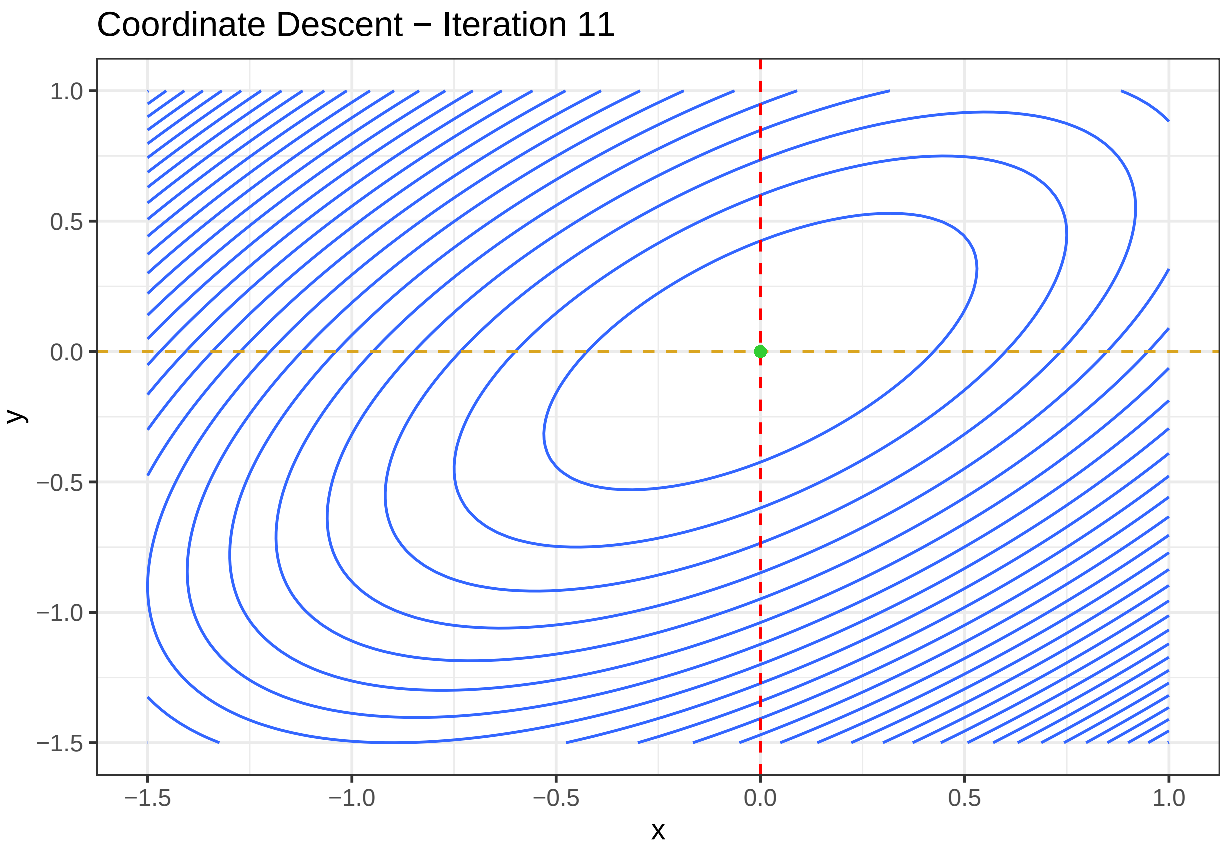

```
## Iteration 9: x = -0.00025, y = -0.00015
```



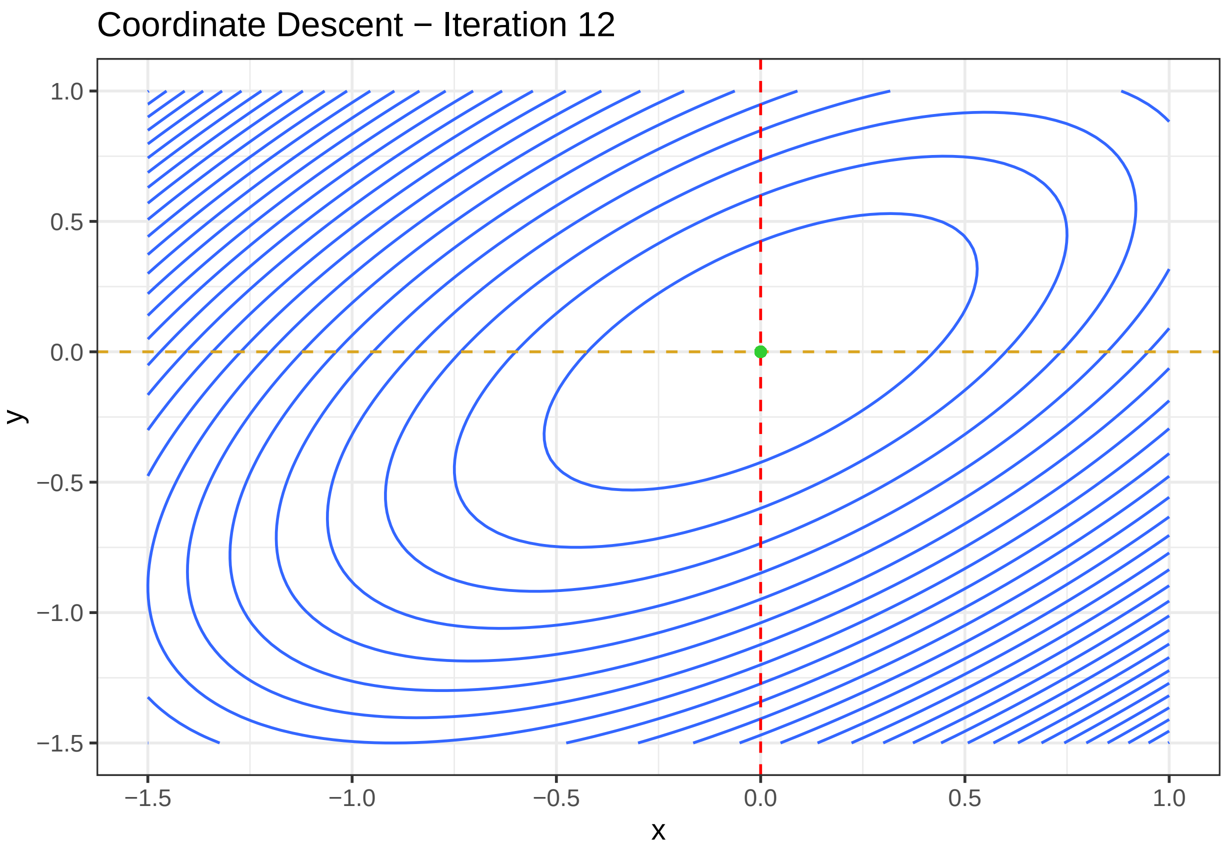

```
## Iteration 10: x = -0.00009, y = -0.00005
```



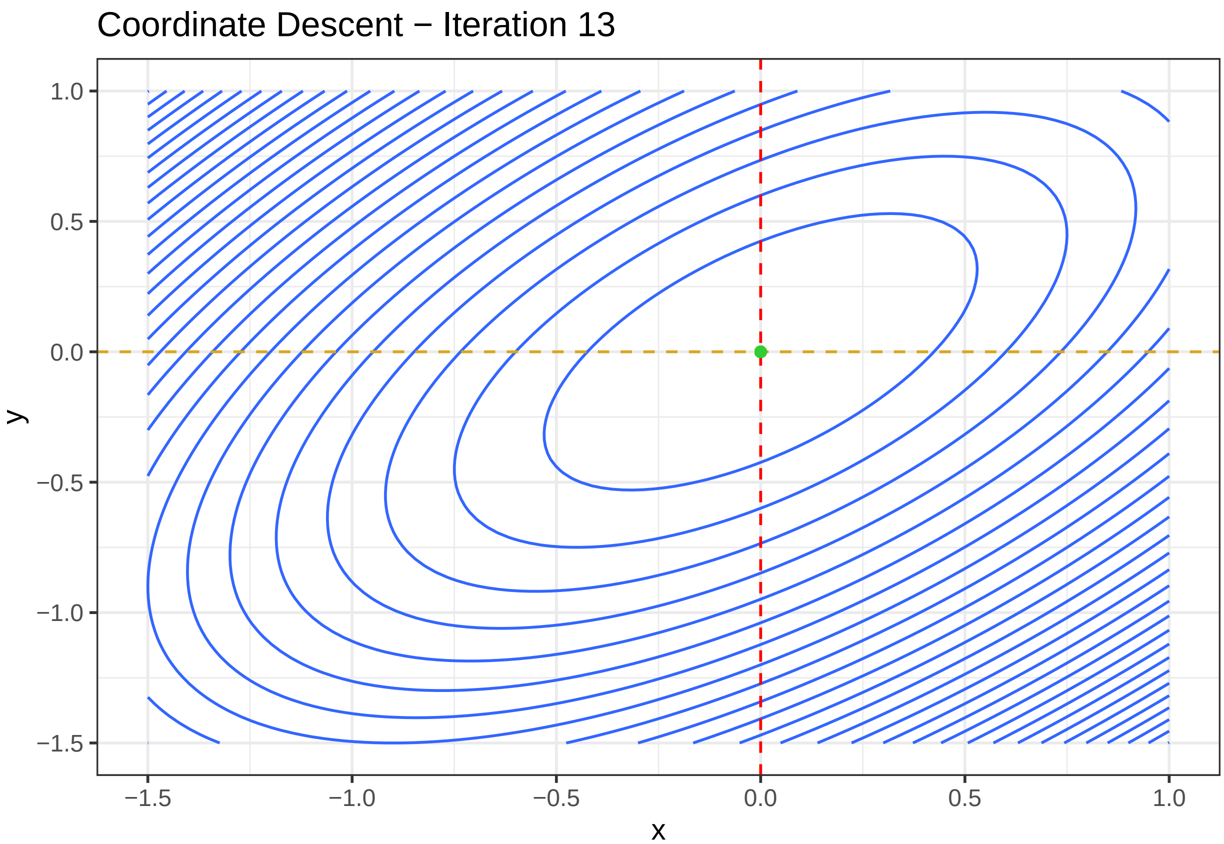

```
## Iteration 11: x = -0.00003, y = -0.00002
```




```
## Iteration 12: x = -0.00001, y = -0.00001
```




```
## Iteration 13: x = -0.00000, y = -0.00000
```



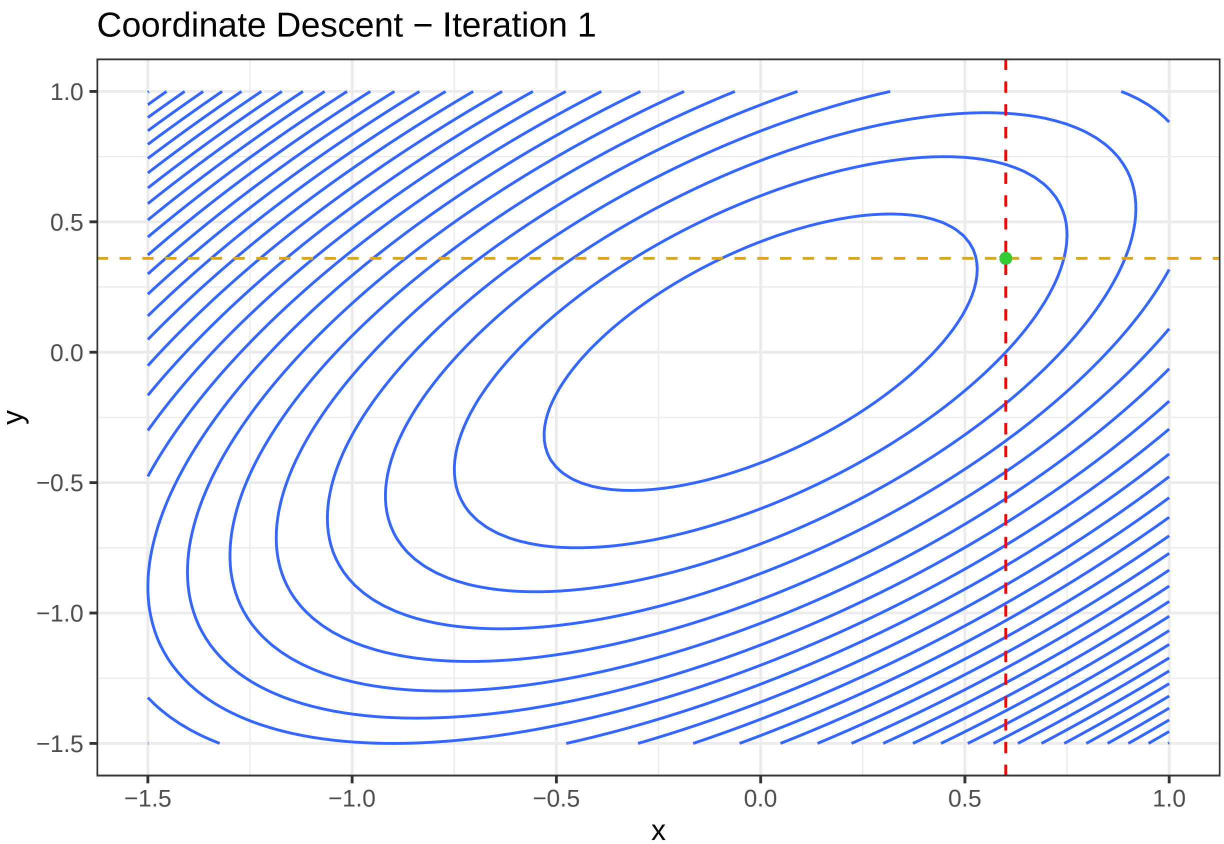

```
## Converges!
```

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

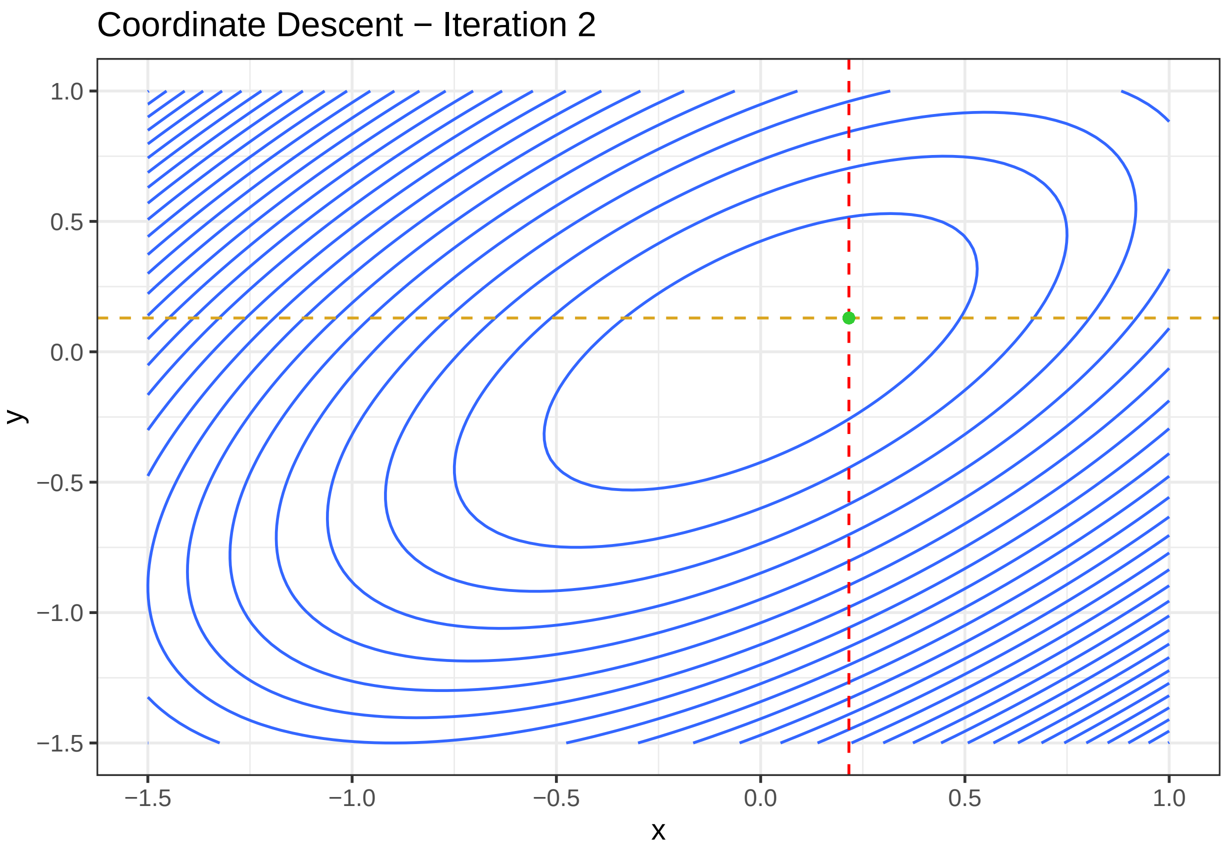
```
x_i <- -1.5
y_i <- 1
for (i in 1:15) {
  # Update x while holding y constant
  f_x <- function(x) g(x, y_i)
  xnew <- golden(f_x, -1.5, 1.5)
  # Update y while holding x constant
  f_y <- function(y) g(xnew, y)
  ynew <- golden(f_y, -1.5, 1.5)
  cat(sprintf("Iteration %d: x = %.5f, y = %.5f\n", i, xnew, ynew))
  # Plot the segments
  p <- ggplot(contour_df, aes(x = x, y = y, z = z)) +
    ggtitle(sprintf("Coordinate Descent - Iteration %d", i)) +
    theme(plot.title = element_text(hjust = 0.5)) +
    geom_contour(binwidth = 0.9) +
    geom_vline(xintercept = xnew, lty = 2, color = "red") +
    geom_hline(yintercept = ynew, lty = 2, color = "goldenrod") +
    geom_point(x = xnew, y = ynew, color = "limegreen") +
    theme_bw()
  print(p)
  # Check convergence
  if (i > 1 && abs(xnew - x_i) < tolerance) {
    cat("Converges!\n")
```

```
    break
  }
  # Update initial values
  x_i <- xnew
  y_i <- ynew
}
```

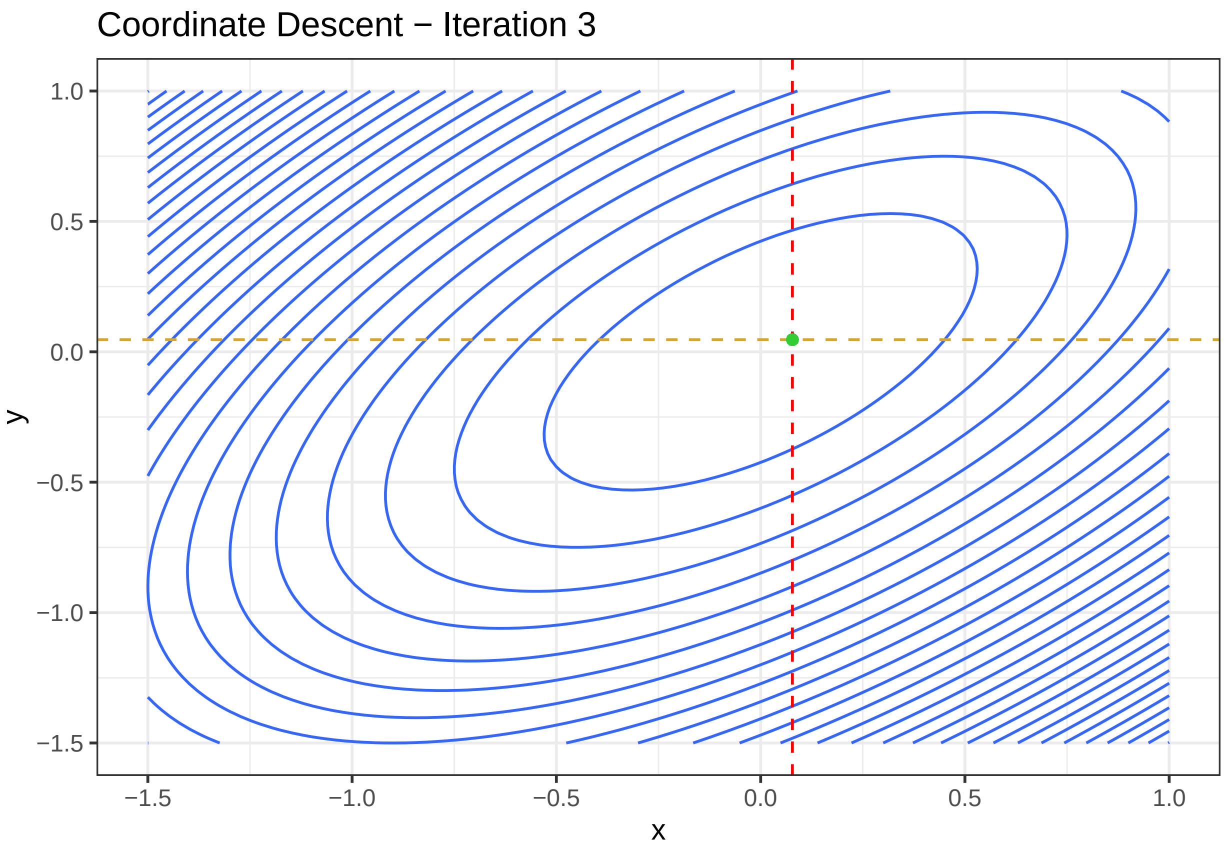
```
## Iteration 1: x = 0.60000, y = 0.36000
```



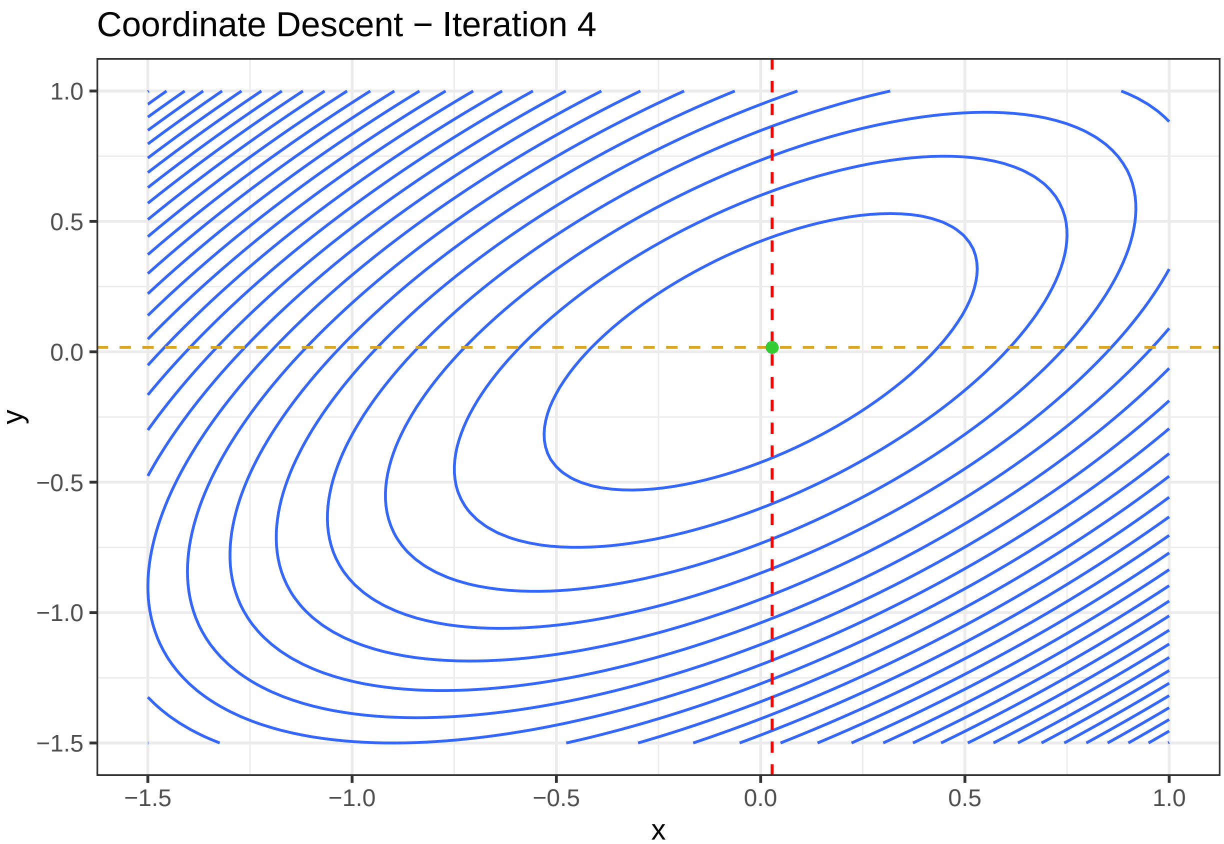

```
## Iteration 2: x = 0.21600, y = 0.12960
```



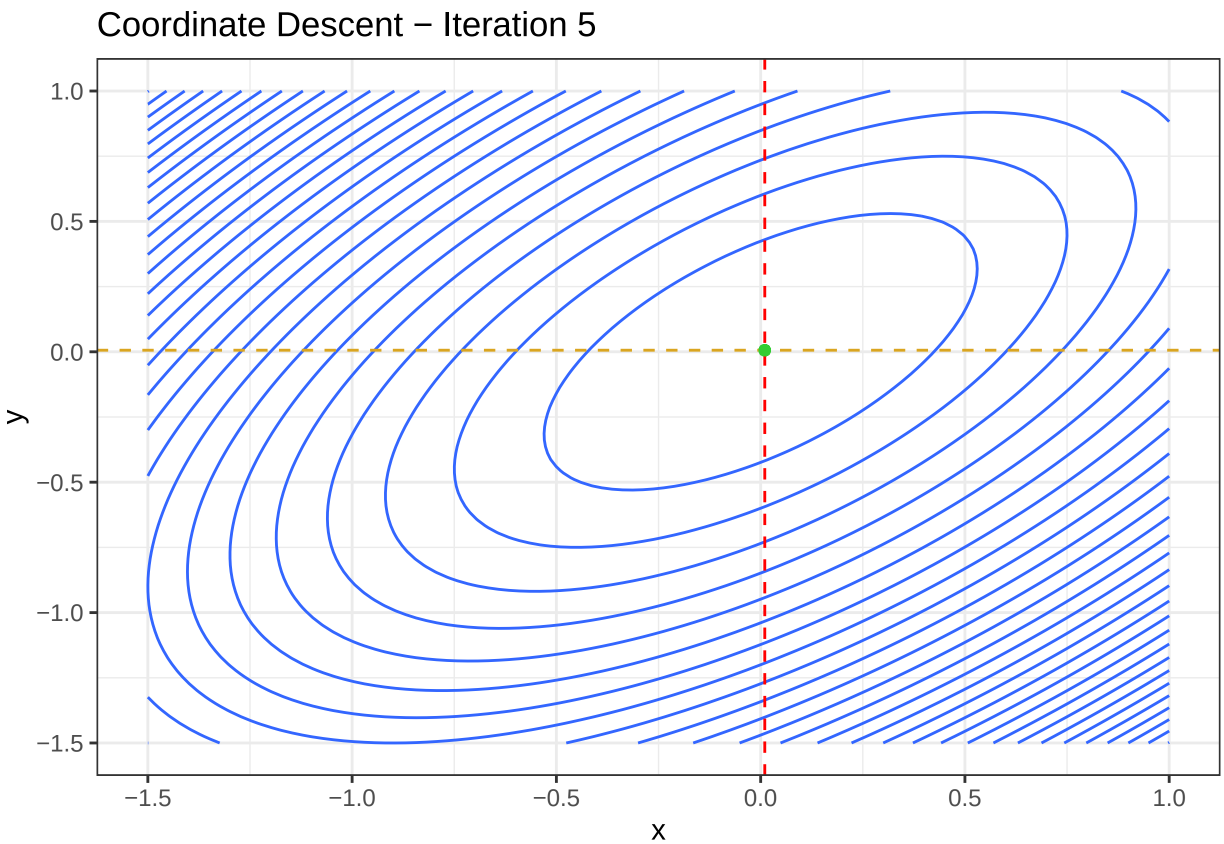

```
## Iteration 3: x = 0.07776, y = 0.04666
```



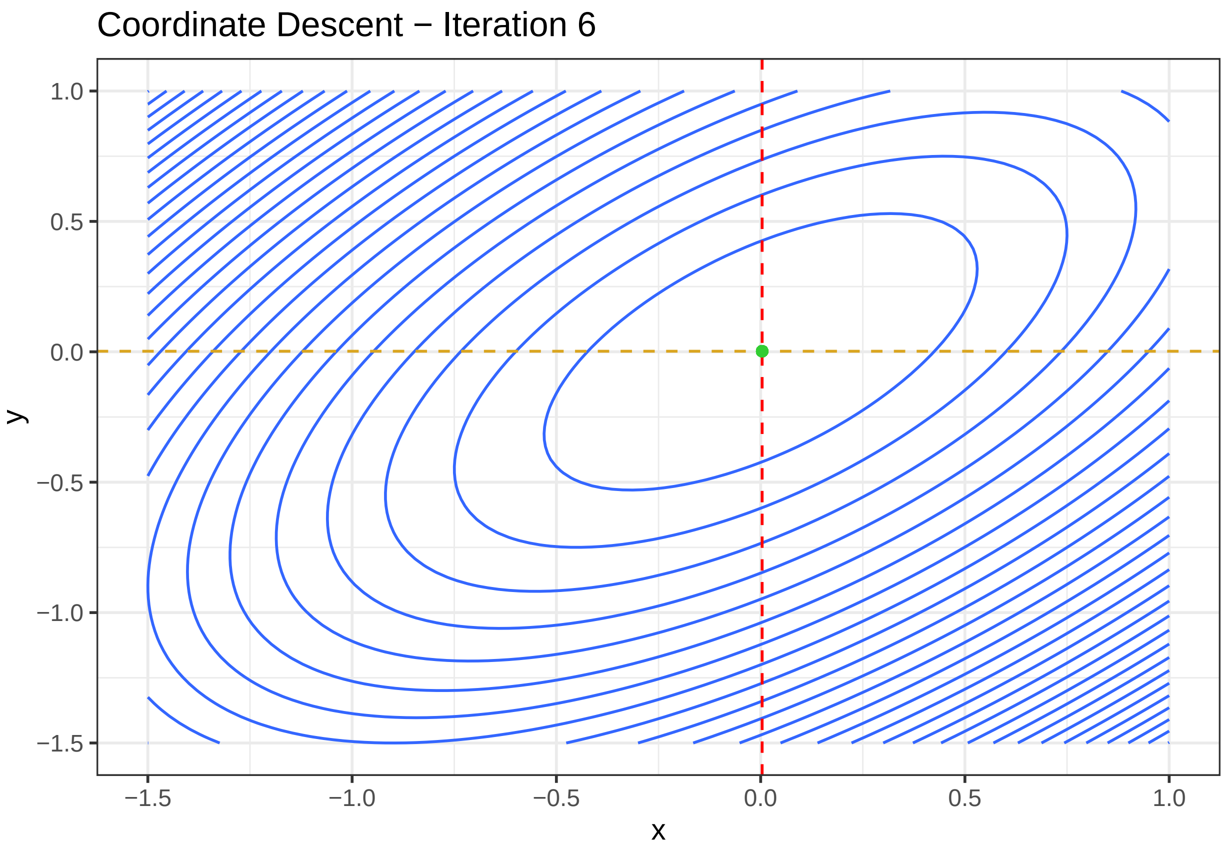

```
## Iteration 4: x = 0.02799, y = 0.01680
```



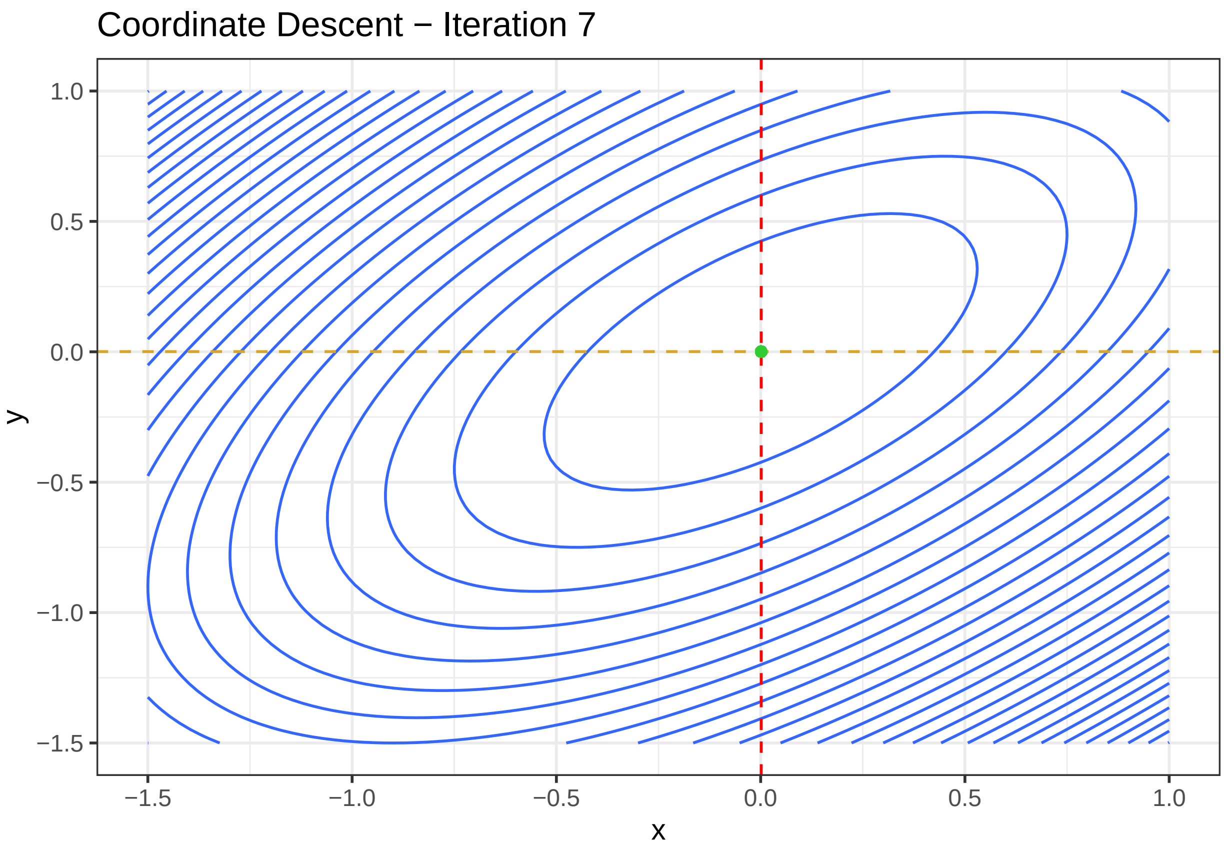

```
## Iteration 5: x = 0.01008, y = 0.00605
```



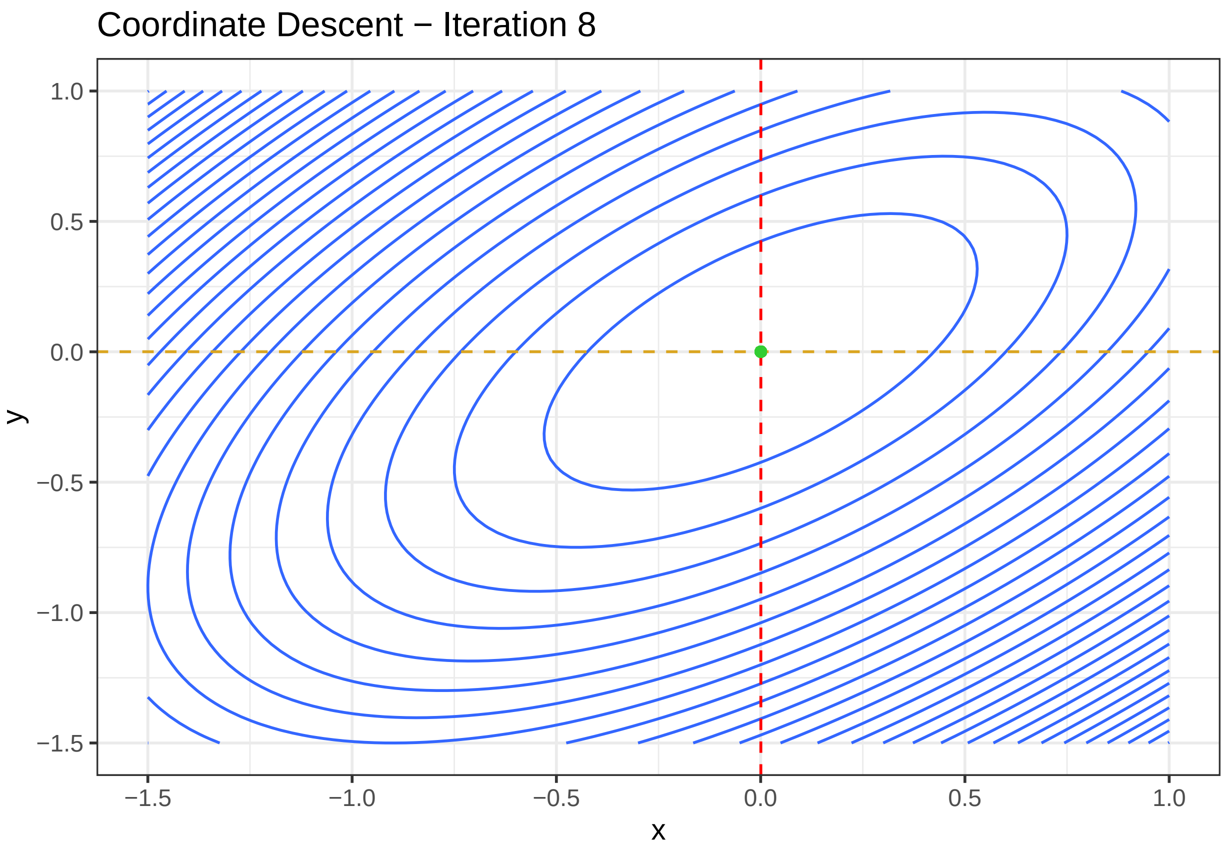

```
## Iteration 6: x = 0.00363, y = 0.00218
```



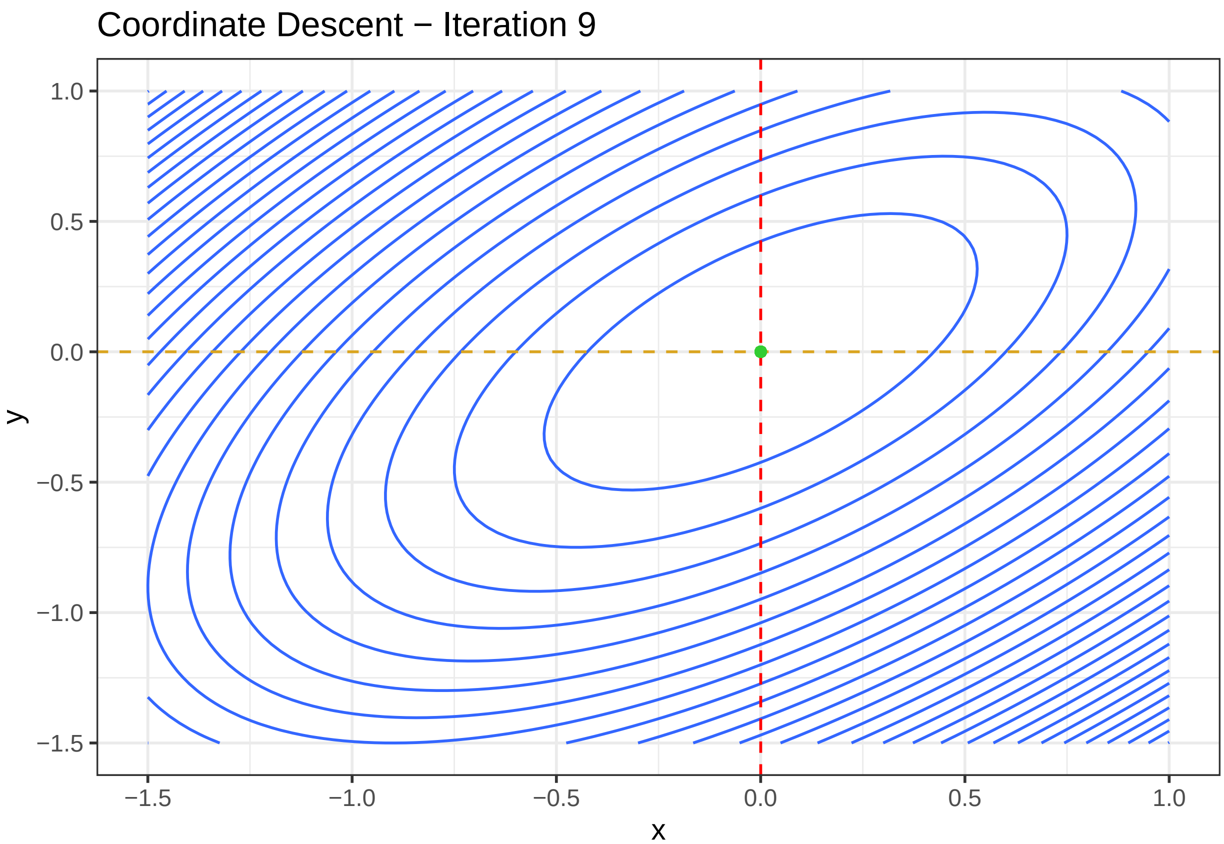

```
## Iteration 7: x = 0.00131, y = 0.00078
```



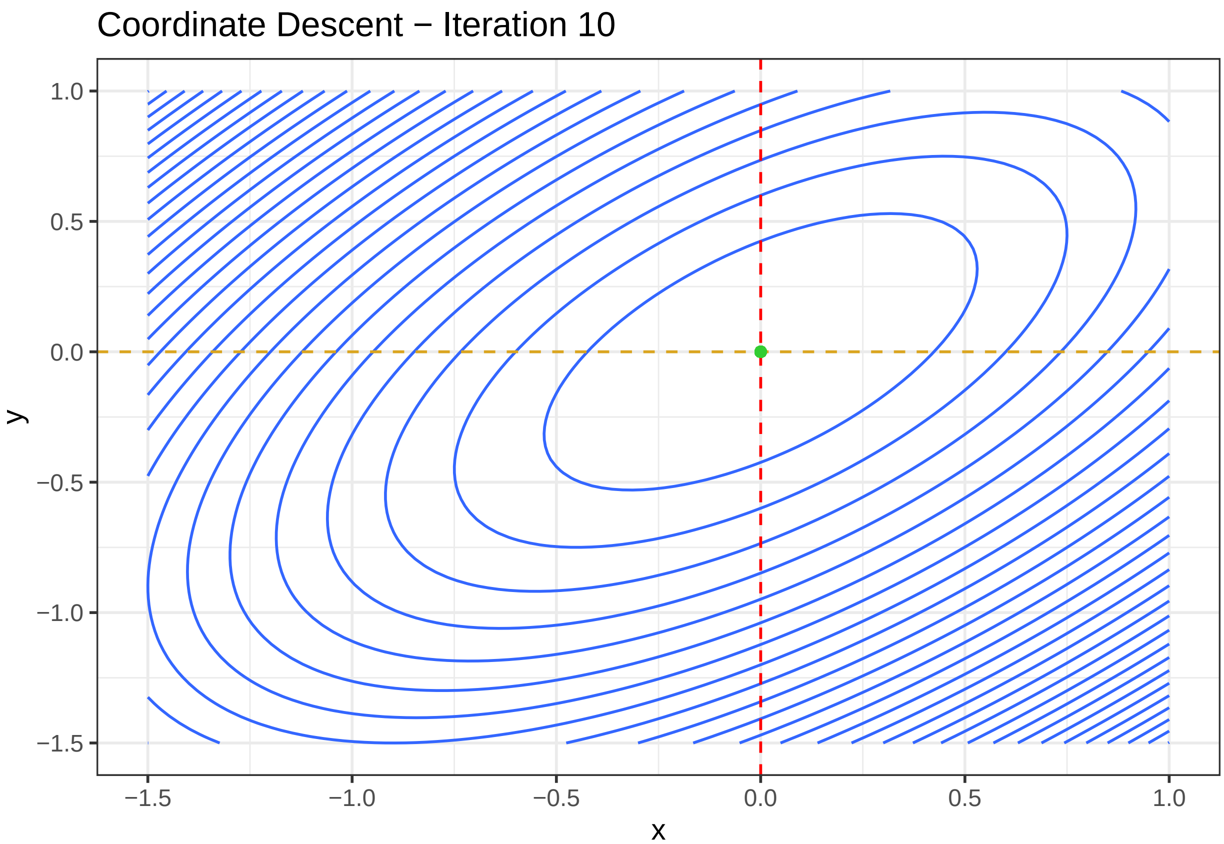

```
## Iteration 8: x = 0.00047, y = 0.00028
```



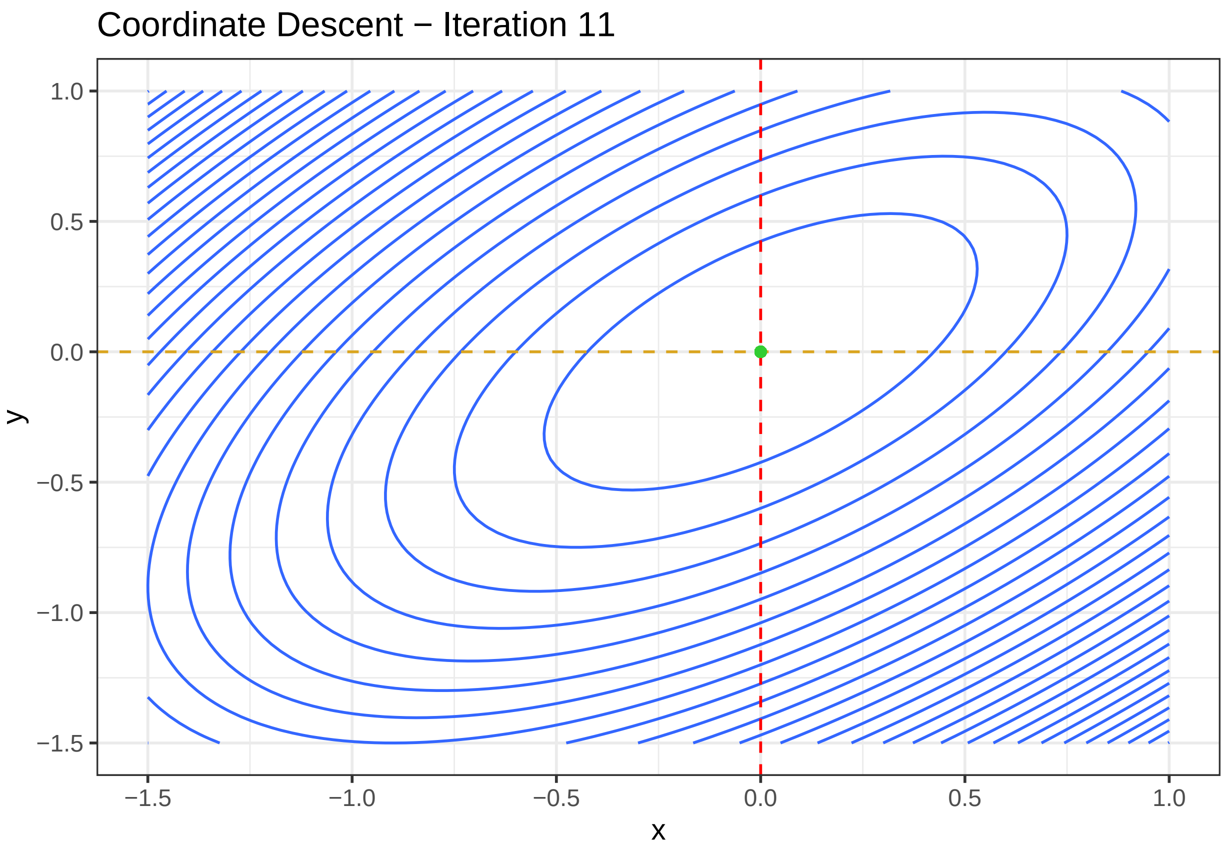

```
## Iteration 9: x = 0.00017, y = 0.00010
```



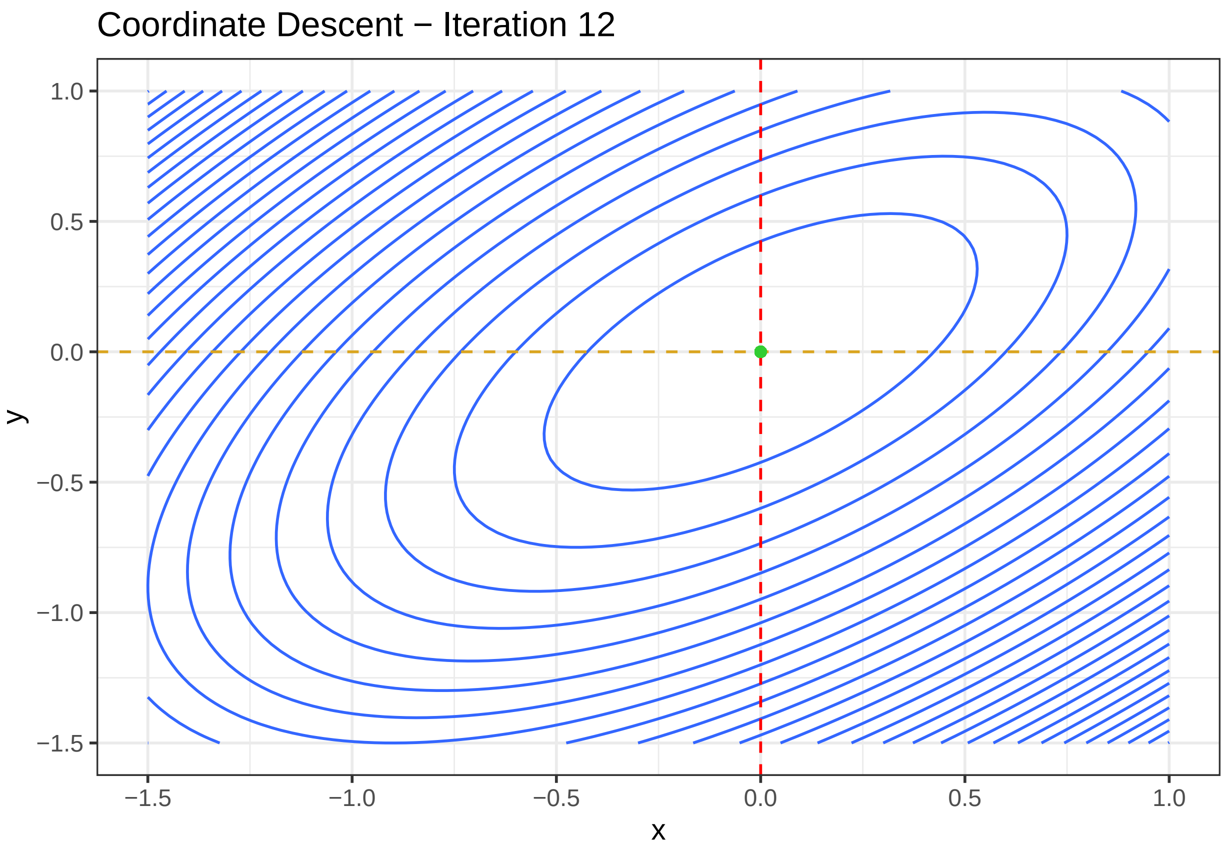

```
## Iteration 10: x = 0.00006, y = 0.00004
```



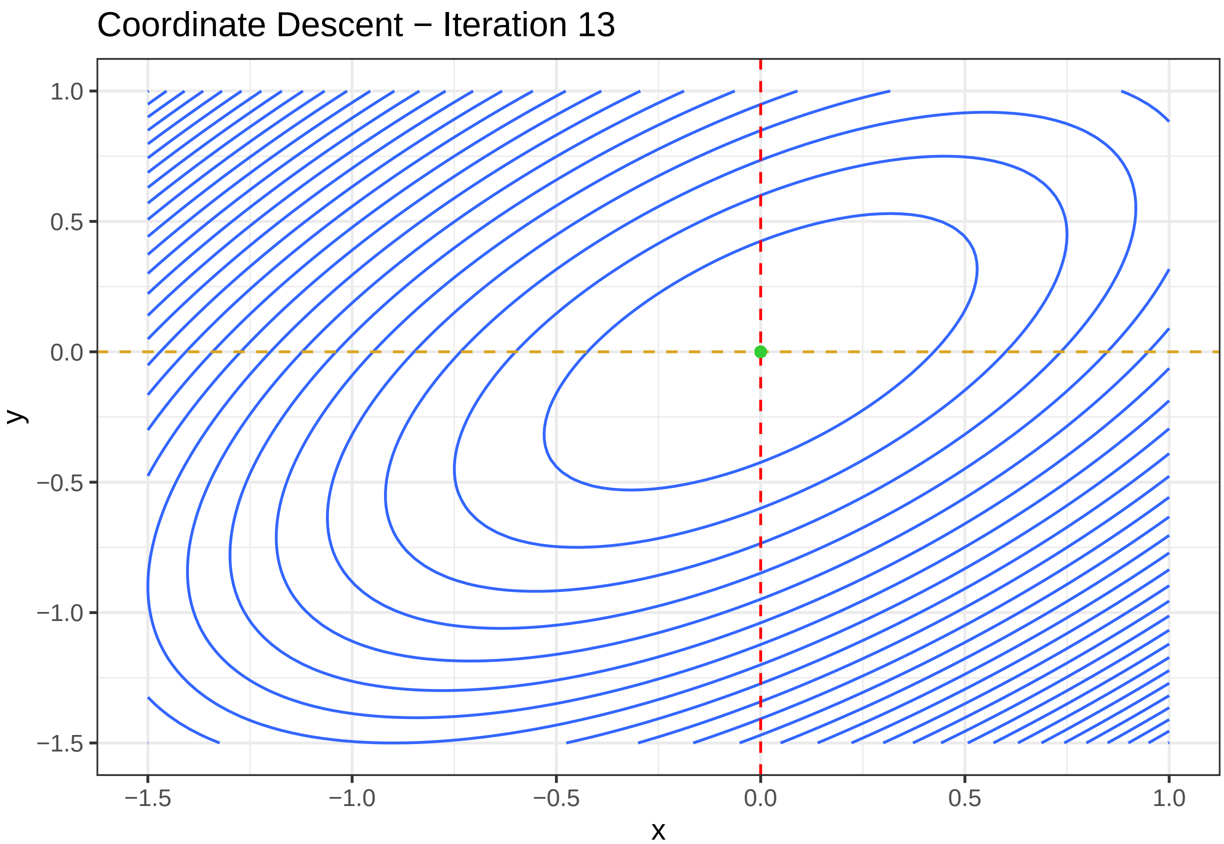

```
## Iteration 11: x = 0.00002, y = 0.00001
```




```
## Iteration 12: x = 0.00001, y = 0.00000
```




```
## Iteration 13: x = 0.00000, y = 0.00000
```




```
## Converges!
```

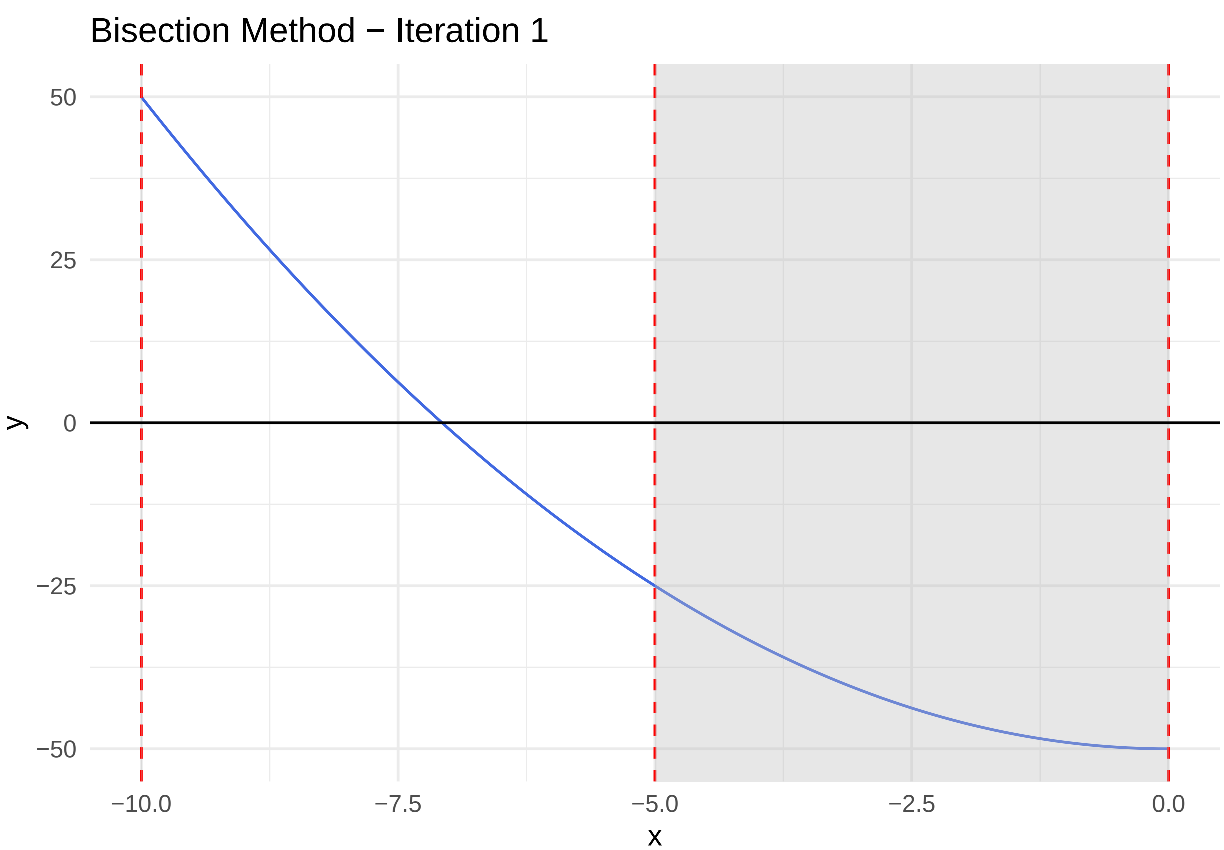
6. Extra Credit: Bisection Search Graph [up to 10 points]

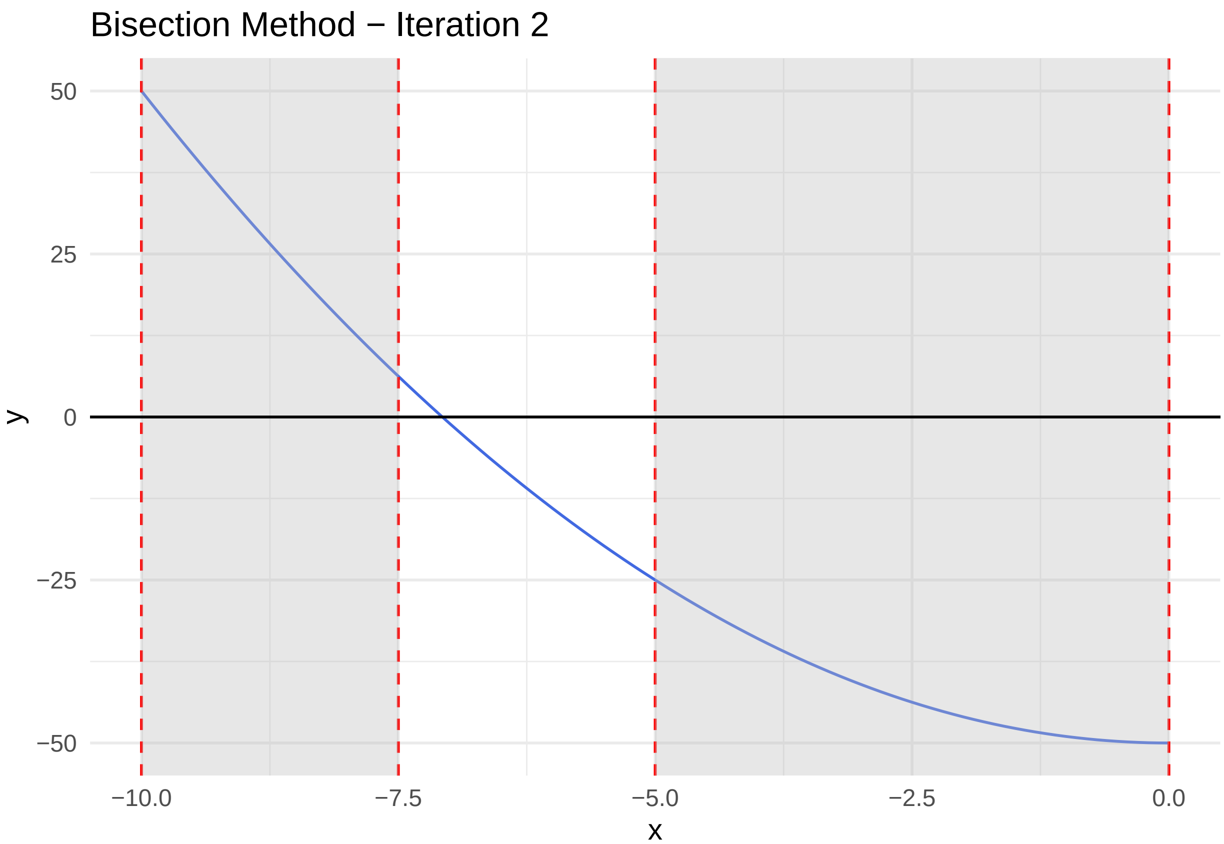
NOTES: $$$$

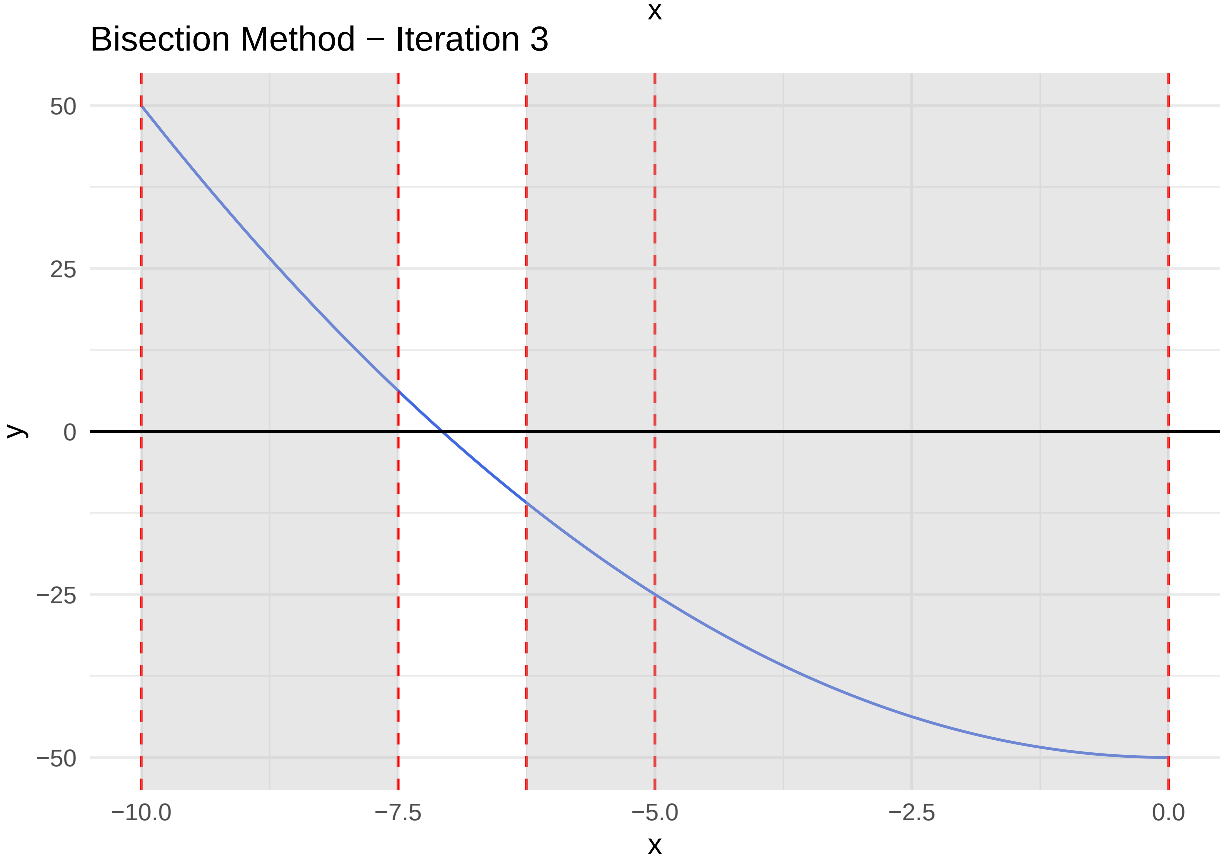
```r
bisection_show <- function(ftn, xl, xr, iter = 5) {
  tol <- 1e-9
  # For graph starting xl/xr lines
  xs <- c(xl, xr)
  xm <- numeric(0)
  global_xl <- xl
  global_xr <- xr
  cat("Starting values are, xleft: ", xl, "xright: ", xr, "\n")

  for (i in 1:iter) {
    xm <- (xl + xr) / 2
    if (ftn(xm) == 0 || (xr - xl) / 2 < tol) {
      break
    }

    if (ftn(xl) * ftn(xm) < 0) {
      xr <- xm
    } else {
      xl <- xm
    }
    # Update bounds
    xs <- c(xs, xl, xr)
```

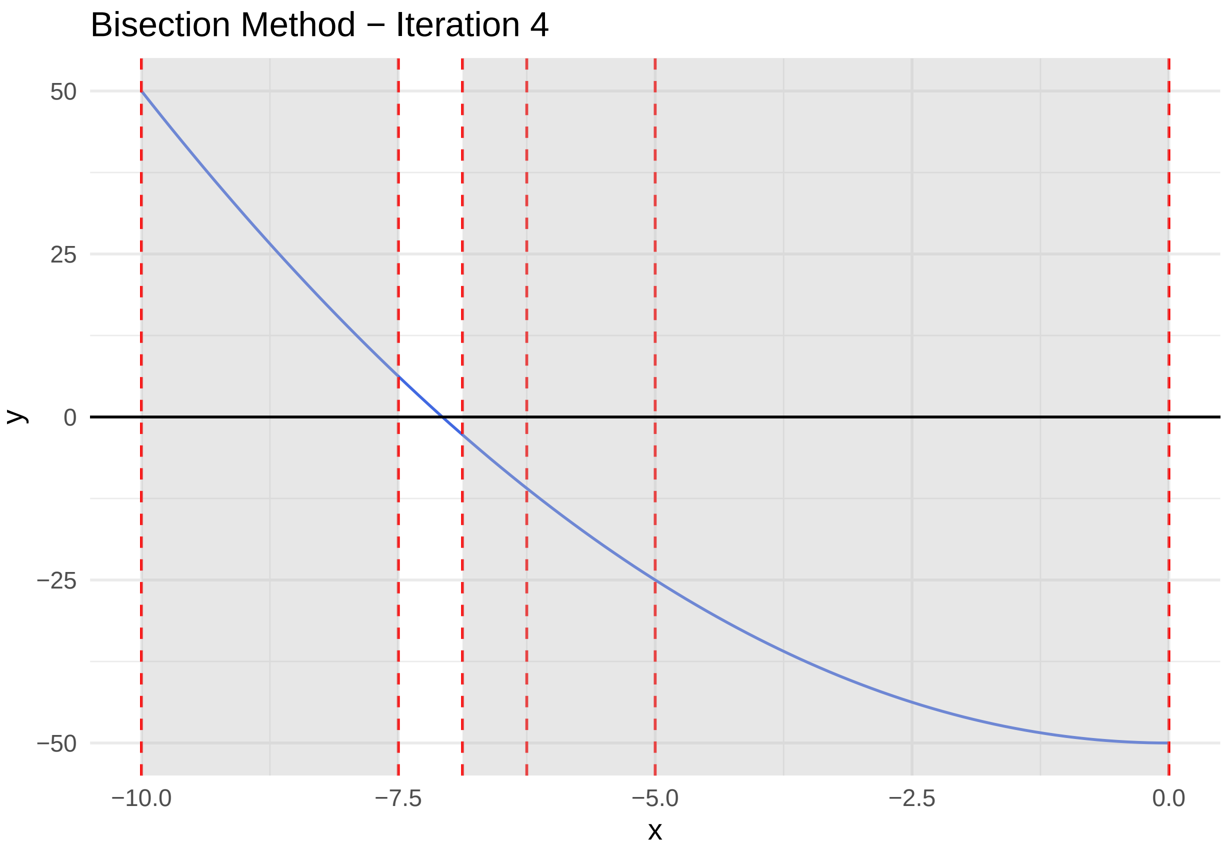
```
    p <- ggplot(data = data.frame(x = c(global_xl, global_xr)), aes(x)) +
      stat_function(fun = ftn, geom = "line", color = "royalblue") +
      geom_vline(xintercept = xs, linetype = 2, color = "red") +
      geom_rect(aes(xmin = global_xl, xmax = xl, ymin = -Inf, ymax = Inf), fill = "gray", alpha = 0.2) +
      geom_rect(aes(xmin = xr, xmax = global_xr, ymin = -Inf, ymax = Inf), fill = "gray", alpha = 0.2) +
      ggtitle(sprintf("Bisection Method - Iteration %d", i)) +
      geom_abline(intercept = 0, slope = 0) +
      # geom_vline(xintercept = 0) +
      theme_minimal()

    print(p)
  }

  xnew <- (xl + xr) / 2
  return(xnew)  # Return the midpoint of the final interval
}
# ex <- function(x) x^2 - 9
# bisection_show(ex, 0, 10, iter = 4)

f <- function(x) x^2 - 50
bisection_show(f, -10, 0, iter = 4)
```

```
## Starting values are, xleft:  -10 xright:  0
```




Bisection Method − Iteration 2
50
25
0
−25
−50
y
−10.0
−7.5
−5.0
−2.5
0.0
x


Bisection Method − Iteration 3
50
25
0
−25
−50
y
−10.0
−7.5
−5.0
−2.5
0.0
x




```
## [1] -7.1875
```