Untitled

Maharaj Teertha Deb, 40227747

2024-03-21

Problem 1 :

The variable old Fib2 isn't strictly necessary. Rewrite the Fibonacci while() loop with the update of Fib1 based just on the current values of Fib1 and Fib2.

Solution:

```
Fib1 <- 0
Fib2 <- 1
Fibonacci <- c(Fib1, Fib2)

while (Fib2 < 300) {
  nextFib <- Fib1 + Fib2
  Fibonacci <- c(Fibonacci, nextFib)
  Fib1 <- Fib2
  Fib2 <- nextFib
}

print(Fibonacci)
```

```
##  [1]   0   1   1   2   3   5   8  13  21  34  55  89 144 233 377
```

Problem 2:

In fact, Fib1 and Fib2 aren't necessary either. Rewrite the Fibonacci while() loop without using any variables except Fibonacci.

Solution:

```
Fibonacci <- c(0, 1)

while (Fibonacci[length(Fibonacci)] + Fibonacci[length(Fibonacci) - 1] < 300) {
  nextFib <- Fibonacci[length(Fibonacci)] + Fibonacci[length(Fibonacci) - 1]
  Fibonacci <- c(Fibonacci, nextFib)
}
print(Fibonacci)
```

```
##  [1]   0   1   1   2   3   5   8  13  21  34  55  89 144 233
```

Problem 3:

Determine the number of Fibonacci numbers less than 1000000

Solution:

```
Fibonacci <- c(1, 1, 1)

while (max(Fibonacci) < 1000000){
  Fibonacci <- c(Fibonacci, Fibonacci[length(Fibonacci)]+
  Fibonacci[length(Fibonacci)-1])
}

sum(Fibonacci < 1000000)
```

```
## [1] 31
```

Problem 4:

Recall the car dealer interest rate example in Example 4.4. Use a `while ()` loop to iteratively calculate the interest rate i which satisfies the fixed-point equation

$$i = \frac{1 - (1+i)^{-20}}{19}$$

Use a starting guess of $i = 0.006$. Stop the calculation when two successive values of the interest rate are less than 0.000001 apart. What happens when you try other starting guesses? Referring to the previous exercise, modify your code so that it also computes the number of iterations required to get two successive values of the interest rate that are less than 0.000001 apart.

Solution:

```
# Function to calculate the interest rate using the fixed-point equation
calculate_interest <- function(guess, tolerance = 0.000001) {
  iter <- 0
  prev_guess <- guess + 2 * tolerance # To ensure the loop starts

  while (abs(prev_guess - guess) >= tolerance) {
    prev_guess <- guess
    guess <- (1 - (1 + guess)^(-20)) / 19
    iter <- iter + 1
  }

  return(list(interest_rate = guess, iterations = iter))
```

```
}

# Initial guess for interest rate
initial_guess <- 0.006

# Calculate interest rate and number of iterations
result <- calculate_interest(initial_guess)

# Print the result
cat("Interest rate:", result$interest_rate, "\n")
```

```
## Interest rate: 0.004954139
```

```
cat("Number of iterations:", result$iterations, "\n")
```

```
## Number of iterations: 74
```

Problem 5:

Referring to the previous exercise, modify your code so that it also computes the number of iterations required to get two successive values of the interest rate that are less than 0.000001 apart.

Solution:

```
# Function to calculate the interest rate using the fixed-point equation
calculate_interest <- function(guess, tolerance = 0.000001) {
  iter <- 0
  prev_guess <- guess + 2 * tolerance # To ensure the loop starts

  while (abs(prev_guess - guess) >= tolerance) {
    prev_guess <- guess
    guess <- (1 - (1 + guess)^(-20)) / 19
    iter <- iter + 1
  }

  return(list(interest_rate = guess, iterations = iter))
}

# Initial guess for interest rate
initial_guess <- 0.006

# Calculate interest rate and number of iterations
result <- calculate_interest(initial_guess)

# Print the result
cat("Interest rate:", result$interest_rate, "\n")
```

```
## Interest rate: 0.004954139
```

```r
cat("Number of iterations:", result$iterations, "\n")
```

```
## Number of iterations: 74
```

Problem 6:

Find a minimizer of the function $f(x) = (x-3)^4 + 7(x-2)^2 + x$.

Solution:

```r
# Define the objective function
f <- function(x) {
  return((x - 3)^4 + 7 * (x - 2)^2 + x)
}

# Use optim() function to minimize the objective function
result <- optim(par = 0, fn = f, method = "Brent", lower = -10, upper = 10)

# Print the minimizer
cat("Minimizer of the function:", result$par, "\n")
```

```
## Minimizer of the function: 2.121972
```

Problem 7:

How many zeros does the function $f(x) = \frac{5x-3}{x-1}$ have? What are they? Describe the behavior of Newton's method applied to this function if the initial guess is:

(a) 0.5

(b) 0.75

(c) 0.2

(d) 1.25

Solution:

```r
# Define the function
f <- function(x) {
  return ((5*x - 3)/(x - 1))
}

# Define the derivative of the function
f_prime <- function(x) {
  return ((5*(x - 1) - (5*x - 3))/((x - 1)^2))
```

```r
}

# Implement Newton's method
newton_method <- function(guess) {
  tol <- 0.0001  # tolerance for convergence
  max_iter <- 100  # maximum number of iterations

  x <- guess
  iter <- 0

  while(iter < max_iter) {
    x_new <- x - f(x)/f_prime(x)

    # Check if the function value is within tolerance
    if (!is.na(f(x_new)) && abs(f(x_new)) < tol) {
      return(list(zero = x_new, iterations = iter + 1))
    }

    x <- x_new
    iter <- iter + 1
  }

  # Return NaN if no convergence within max_iter
  return(list(zero = NaN, iterations = max_iter))
}

# Define initial guesses
initial_guesses <- c(0.5, 0.75, 0.2, 1.25)

# Apply Newton's method with different initial guesses
results <- lapply(initial_guesses, newton_method)

# Print results
for (i in 1:length(results)) {
  cat("Initial guess:", initial_guesses[i], "\n")
  cat("Zero found:", results[[i]]$zero, "\n")
  cat("Number of iterations:", results[[i]]$iterations, "\n\n")
}
```

```
## Initial guess: 0.5
## Zero found: 0.6000061
## Number of iterations: 3
##
## Initial guess: 0.75
## Zero found: 0.6000001
## Number of iterations: 4
##
## Initial guess: 0.2
## Zero found: NaN
## Number of iterations: 100
##
## Initial guess: 1.25
## Zero found: NaN
```

```
## Number of iterations: 100
```

Problem 8:

How many zeros does the function $f(x) = (x^2 - 6x + 9)e^{-x}$ have? What are they? Describe the behavior of Newton's method applied to this function if the initial guess is: (a) $x_0 = 3$ (b) $x_0 = 3.2$ (c) $x_0 = 2.99$ (d) $x_0 = 3.01$

Solution:

```r
# Define the function f(x)
f <- function(x) {
  return((x^2 - 6*x + 9) * exp(-x))
}

# Define the derivative of the function f'(x)
f_der <- function(x) {
  return((2*x - 6) * exp(-x) + (x^2 - 6*x + 9) * (-exp(-x)))
}

# Define initial guesses
initial_guesses <- c(3, 3.2, 2.99, 3.01)

# Define tolerance to stop the loop
tolerance <- 1e-3

# Run the loop for each initial guess
for (guess in initial_guesses) {

  # Set the initial guess
  x <- guess

  # Run Newton's method
  while (TRUE) {
    # Calculate next value of x
    x_next <- x - (f(x) / f_der(x))

    # Check if derivative of function is zero
    if (is.na(x_next)) {
      cat("Derivative of function is zero, cannot be evaluated further.\n")
      break
    }

    # Calculate error
    error <- abs(x_next - x)

    # Check if error is less than tolerance
    if (error < tolerance) {
      cat("With initial guess of", guess, "Root is", x_next, "\n")
      break
    } else {
```

```
      # Update the parameter
      x <- x_next
    }
  }
}
```

```
## Derivative of function is zero, cannot be evaluated further.
## With initial guess of 3.2 Root is 3.000635
## With initial guess of 2.99 Root is 2.999369
## With initial guess of 3.01 Root is 3.000619
```

Problem 9:

We could implement the Sieve of Eratosthenes using a while () loop. (c) Modify this function using break to take advantage of the above result.

Solution:

```
sieve_of_eratosthenes <- function(n) {
  primes <- vector(mode = "logical", length = n + 1)
  primes[2:n] <- TRUE

  p <- 2
  while (TRUE) {
    # Mark multiples of p as non-prime
    primes[p * 2:n] <- FALSE

    # Find the next prime number
    for (i in (p + 1):n) {
      if (primes[i]) {
        p <- i
        break  # Move to the next prime
      }
    }

    # If there's no more prime to find, break the loop
    if (p^2 > n) {
      break
    }
  }

  # Return the list of prime numbers
  return(which(primes))
}

print(sieve_of_eratosthenes(3000))
```

```
##   [1]    2    3    5    7   11   13   17   19   23   29   31   37   41   43   47
##  [16]   53   59   61   67   71   73   79   83   89   97  101  103  107  109  113
```

```
##  [31]  127  131  137  139  149  151  157  163  167  173  179  181  191  193  197
##  [46]  199  211  223  227  229  233  239  241  251  257  263  269  271  277  281
##  [61]  283  293  307  311  313  317  331  337  347  349  353  359  367  373  379
##  [76]  383  389  397  401  409  419  421  431  433  439  443  449  457  461  463
##  [91]  467  479  487  491  499  503  509  521  523  541  547  557  563  569  571
## [106]  577  587  593  599  601  607  613  617  619  631  641  643  647  653  659
## [121]  661  673  677  683  691  701  709  719  727  733  739  743  751  757  761
## [136]  769  773  787  797  809  811  821  823  827  829  839  853  857  859  863
## [151]  877  881  883  887  907  911  919  929  937  941  947  953  967  971  977
## [166]  983  991  997 1009 1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069
## [181] 1087 1091 1093 1097 1103 1109 1117 1123 1129 1151 1153 1163 1171 1181 1187
## [196] 1193 1201 1213 1217 1223 1229 1231 1237 1249 1259 1277 1279 1283 1289 1291
## [211] 1297 1301 1303 1307 1319 1321 1327 1361 1367 1373 1381 1399 1409 1423 1427
## [226] 1429 1433 1439 1447 1451 1453 1459 1471 1481 1483 1487 1489 1493 1499 1511
## [241] 1523 1531 1543 1549 1553 1559 1567 1571 1579 1583 1597 1601 1607 1609 1613
## [256] 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697 1699 1709 1721 1723 1733
## [271] 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811 1823 1831 1847 1861 1867
## [286] 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949 1951 1973 1979 1987
## [301] 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063 2069 2081 2083 2087
## [316] 2089 2099 2111 2113 2129 2131 2137 2141 2143 2153 2161 2179 2203 2207 2213
## [331] 2221 2237 2239 2243 2251 2267 2269 2273 2281 2287 2293 2297 2309 2311 2333
## [346] 2339 2341 2347 2351 2357 2371 2377 2381 2383 2389 2393 2399 2411 2417 2423
## [361] 2437 2441 2447 2459 2467 2473 2477 2503 2521 2531 2539 2543 2549 2551 2557
## [376] 2579 2591 2593 2609 2617 2621 2633 2647 2657 2659 2663 2671 2677 2683 2687
## [391] 2689 2693 2699 2707 2711 2713 2719 2729 2731 2741 2749 2753 2767 2777 2789
## [406] 2791 2797 2801 2803 2819 2833 2837 2843 2851 2857 2861 2879 2887 2897 2903
## [421] 2909 2917 2927 2939 2953 2957 2963 2969 2971 2999
```