A5

Maharaj Teertha Deb, 40227747

2024-03-14

Problem 1:

Refer to the car dealer promotion example in this section. Calculate the first 20 updates of the interest rate `i`, starting with `i=0.006`. Repeat with a starting value of `i=0.005`, and with a starting value of `i=0.004`.Based on these observations, what is the true value of `i` (up to 5-digit accuracy)?

Solution:

```r
# Function to update interest rate
update_interest_rate <- function(initial_i) {
  i <- initial_i
  updated_i <- numeric(20)
  for (j in 1:20) {
    i <- (1 - (1 + i)^-20) / 19;
    updated_i[j] <- i
  }
  return(updated_i)
}

# Calculate updated interest rates for initial values 0.006, 0.005, and 0.004
initial_i_values <- c(0.006, 0.005, 0.004)
all_updated_i_values <- lapply(initial_i_values, update_interest_rate)

# Display the updated interest rates
all_updated_i_values
```

```
## [[1]]
##  [1] 0.005934815 0.005874258 0.005817928 0.005765464 0.005716546 0.005670886
##  [7] 0.005628225 0.005588329 0.005550987 0.005516007 0.005483215 0.005452453
## [13] 0.005423575 0.005396449 0.005370954 0.005346979 0.005324421 0.005303187
## [19] 0.005283189 0.005264348
##
## [[2]]
##  [1] 0.004996689 0.004993551 0.004990575 0.004987754 0.004985079 0.004982543
##  [7] 0.004980137 0.004977856 0.004975693 0.004973642 0.004971696 0.004969850
## [13] 0.004968099 0.004966439 0.004964863 0.004963369 0.004961951 0.004960606
## [19] 0.004959330 0.004958120
##
## [[3]]
```

```
##  [1] 0.004038755 0.004076253 0.004112508 0.004147532 0.004181343 0.004213959
##  [7] 0.004245401 0.004275691 0.004304851 0.004332907 0.004359884 0.004385809
## [13] 0.004410709 0.004434612 0.004457546 0.004479540 0.004500622 0.004520821
## [19] 0.004540165 0.004558684
```

Problem 2:

Use a fixed-point iteration to determine the solution (in [0,1])of the equation x = cos(x). Use a starting value of 0.5. How many iterations does it take before you have an answer which is accurate in the first 2 digits? ...in the first 3 digits?...in the first 4 digits? What happens if you change the starting value to 0.7?...to 0.0?

Solution:

```
x <- 0.5
count <- 0
while(abs(x - cos(x)) > 0.01){
  x <- cos(x)
  count <- count + 1
}
count
```

```
## [1] 10
```

```
x <- 0.5
count <- 0
while(abs(x - cos(x)) > 0.001){
  x <- cos(x)
  count <- count + 1
}
count
```

```
## [1] 15
```

```
x <- 0.5
count <- 0
while(abs(x - cos(x)) > 0.0001){
  x <- cos(x)
  count <- count + 1
}
count
```

```
## [1] 21
```

```
x <- 0.7
count <- 0
while(abs(x - cos(x))>0.0001){
  x <- cos(x)
  count <- count + 1
}
count
```

```
## [1] 17
```

```r
x <- 0.0
count <- 0
while(abs(x - cos(x)) > 0.0001){
  x <- cos(x)
  count <- count + 1
}
count
```

```
## [1] 23
```

Problem 3:

6. Repeat the previous question, but using the equation $x = 1.5\cos(x)$. (The solution is near $x = 0.9148565$.) Does the fixed-point iteration converge? If not, modify the equation so that $x = \frac{\cos(x)}{30} + \frac{44x}{45}$. Does the iteration converge now?

Solution:

```r
# Function to perform fixed-point iteration for equation x = 1.5*cos(x)
fixed_point_iteration_cos_mod <- function(initial_guess, equation) {
  x <- initial_guess
  tol <- 1e-6  # Tolerance for convergence

  count <- 0  # Counter for iterations

  # Fixed-point iteration loop
  while (TRUE) {
    count <- count + 1
    new_x <- equation(x)
    if (abs(new_x - x) < tol) {
      break
    }
    x <- new_x
    # Break if iteration count exceeds a threshold
    if (count > 1000) {
      return(list(solution = NA, iterations = NA))
    }
  }

  return(list(solution = x, iterations = count))
}

# Define the equation x = 1.5*cos(x)
equation_mod <- function(x) {
  return(1.5 * cos(x))
}

# Repeat the previous question
print("Repeat the previous question:")
```

```
## [1] "Repeat the previous question:"
```

```r
# Perform fixed-point iteration for equation x = 1.5*cos(x) with a starting value of 0.5
result_mod <- fixed_point_iteration_cos_mod(0.5, equation_mod)
print("Fixed-point iteration for x = 1.5*cos(x):")
```

```
## [1] "Fixed-point iteration for x = 1.5*cos(x):"
```

```r
print(result_mod)
```

```
## $solution
## [1] NA
##
## $iterations
## [1] NA
```

```r
# Check if fixed-point iteration was successful
if (!is.na(result_mod$solution)) {
  print("Fixed-point iteration succeeded.")
} else {
  print("Fixed-point iteration failed. Modifying the equation...")

  # Modify the equation x = cos(x)/30 + 44*x/45
  equation_mod_modified <- function(x) {
    return(cos(x)/30 + 44*x/45)
  }

  # Perform fixed-point iteration for modified equation
  result_mod_modified <- fixed_point_iteration_cos_mod(0.5, equation_mod_modified)
  print("Fixed-point iteration for modified equation:")
  print(result_mod_modified)

  # Check if fixed-point iteration was successful after modification
  if (!is.na(result_mod_modified$solution)) {
    print("Fixed-point iteration succeeded after modification.")
  } else {
    print("Fixed-point iteration failed even after modification.")
  }
}
```

```
## [1] "Fixed-point iteration failed. Modifying the equation..."
## [1] "Fixed-point iteration for modified equation:"
## $solution
## [1] 0.914836
##
## $iterations
## [1] 202
##
## [1] "Fixed-point iteration succeeded after modification."
```

Problem 4:

A twin prime is a pair of primes (x, y), such that y = x + 2. Construct a list of all twin primes less than 1000. The result should be stored in a list of numeric vectors called `twin_primes`, whose elements are the twin prime pairs. Print the length of the list `twin_primes` and print the 10th and the 15th elements of the list, i.e., `twin_primes[[10]]` and `twin_primes[[15]]`.

Solution:

```r
# Function to check if a number is prime
is_prime <- function(n) {
  if (n <= 1) {
    return(FALSE)
  }
  if (n <= 3) {
    return(TRUE)
  }
  if (n %% 2 == 0 || n %% 3 == 0) {
    return(FALSE)
  }
  i <- 5
  while (i * i <= n) {
    if (n %% i == 0 || n %% (i + 2) == 0) {
      return(FALSE)
    }
    i <- i + 6
  }
  return(TRUE)
}

# Construct a list of all twin primes less than 1000
twin_primes <- list()

for (i in 2 : 1000) {
  if (is_prime(i) && is_prime(i + 2)) {
    twin_primes[[length(twin_primes) + 1]] <- c(i, i + 2)
  }
}

# Print the length of the list twin_primes
cat("Length of twin_primes list:", length(twin_primes), "\n")
```

```
## Length of twin_primes list: 35
```

```r
# Print the 10th element of the list
cat("10th element of twin_primes:", twin_primes[[10]], "\n")
```

```
## 10th element of twin_primes: 107 109
```

```r
# Print the 15th element of the list
cat("15th element of twin_primes:", twin_primes[[15]], "\n")
```

```
## 15th element of twin_primes: 197 199
```

Prolem 5:

A bank offers a guaranteed investment certificate (GIC) which pays an annual interest rate of 4% (compounded annually) if the term is 3 years or less, or 5% if the term is more than 3 years.

Write a function which takes the initial investment amount, P, and the number of interest periods (i.e., years) as arguments and which returns the amount of interest earned over the term of the GIC. That is, return I, where $I = P((1+i)^n - 1)$. Call the desired function `GIC`. It should be of the form `GIC(P, n)`, where P is the initial investment amount and n is the number of years.

Print the output of `GIC(1000, 2)` and `GIC(1000, 20)` to show the corresponding amount of interest earned.

The function `GIC` should return an error (using the `stop()` command) in the following cases: - P is negative - n is negative or is not an integer

Solution:

```r
# Function to calculate the amount of interest earned on a GIC
GIC <- function(P, n) {
  # Check for invalid input
  P <- as.integer(P)
  n <- as.integer(n)

  if (P < 0 ) {
    stop("Invalid input. Please provide a non-negative initial investment amount
         and a non-negative integer number of years.")
  } else if ( n < 0){
      stop("Invalid input. Please provide a non-negative integer number of years.")
  }

  # Calculate the interest rate based on the number of years
  if (n <= 3) {
    i <- 0.04  # 4% interest for 3 years or less
  } else {
    i <- 0.05  # 5% interest for more than 3 years
  }

  # Calculate the amount of interest earned
  I <- P * ((1 + i)^n - 1)
  return(I)
}

# Test the function with examples provided
tryCatch({
  interest_2_years <- GIC(1000, 2)
  print(paste("Interest earned over 2 years:", interest_2_years))
```

```r
}, error = function(e) {
  print("ERROR!")
  print(e$message)
})
```

```
## [1] "Interest earned over 2 years: 81.6000000000001"
```

```r
tryCatch({
  interest_20_years <- GIC(1000, 20)
  print(paste("Interest earned over 20 years:", interest_20_years))
}, error = function(e) {
  print("ERROR!")
  print(e$message)
})
```

```
## [1] "Interest earned over 20 years: 1653.29770514442"
```