Math 660

Sep 2023

Name: Krisha Shah

1. Write a function to evaluate ab given a and b. The function should have a default value of 3 for b.

```
power_of_ab <- function(a, b = 3)
  {
  a^b
  }
power_of_ab(2)
```

```
## [1] 8
```

```
power_of_ab(2,4)
```

```
## [1] 16
```

Comment: When we pass 2 in the arguments list, it takes b as default 3. However in the second case, it takes b to be 4.

2. Write a function that accepts two arguments, a number and a vector, and returns TRUE if the number is inside the vector.

```
check_number_in_vector <- function(num, vec)
{
  if (num %in% vec)
    return(TRUE)
  else
    return(FALSE)
}

vector1 <- c(1,2,3,4,5,6,7)
check_number_in_vector(8, vector1)
```

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

3. Write a function that, given a number and a vector, will return the number of times the number occurs in the vector.

```r
check_number_in_vector_count <- function(num, vec)
{
  count <- 0
  for (i in vec)
  if (i==num)
    count <- count+1
  else
    count <- count
  return(count)
}

vector1 <- c(1,2,3,7,5,6,7)
result <- check_number_in_vector_count(7, vector1)
print(result)
```

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

Comment: We run a for loop to check if any of the values in the vector match with num. If they match we increase the count by 1 each time and finally return count.

4. Use the mtcars dataset for this question. Use one of the apply functions to find the mean of every column in mtcars.

```r
lapply(mtcars, mean)
```

```
## $mpg
## [1] 20.09062
##
## $cyl
## [1] 6.1875
##
## $disp
## [1] 230.7219
##
## $hp
## [1] 146.6875
##
## $drat
## [1] 3.596563
##
## $wt
## [1] 3.21725
##
## $qsec
## [1] 17.84875
##
## $vs
## [1] 0.4375
##
## $am
```

```
## [1] 0.40625
##
## $gear
## [1] 3.6875
##
## $carb
## [1] 2.8125
```

5. Write a function in R that takes as input a non-negative integer n and returns Fn. Use a for loop in the function. Then use the function to find F20 and F80.

```r
fibonacci <- function(n)
  {
  if (n < 0)
    {
    return(NULL)
    }
  else if (n == 0)
    {
    return(0)
    }
  else if (n == 1)
    {
    return(1)
    }
  else
    {
    fn <- numeric(n+1)
    fn[1] <- 0
    fn[2] <- 1

    for (i in 3:(n+1))
    {
      fn[i] <- fn[i-1] + fn[i-2]
    }

    return(fn[n+1])
    }
  }

cat("Fibonacci of 20 =",fibonacci(20),"\n")
```

```
## Fibonacci of 20 = 6765
```

```r
cat("Fibonacci of 80 =",fibonacci(80),"\n")
```

```
## Fibonacci of 80 = 2.341673e+16
```

6. Write an R function to compute fn(x) based on Taylor's expansion for any real number x and positive integer n. Use a while loop to do this. The function should return a vector with two numbers, the value of fn(x) and the approximation error. Run your function for x = 3, n = 10 and x = 0.44, n = 3.

```
taylor_expansion <- function(x, n)
{
  i <- 2
  term <- 1+x

  while (i < n+1)
  {
    term <- term + (x^i /factorial(i))
    i <- i + 1
  }
  approximation_error <- exp(x) - term
  result <- c(term, approximation_error)
  return(result)
}
cat("For x = 3 and n = 10:\n")
```

```
## For x = 3 and n = 10:
```

```
cat(taylor_expansion(3,10),"\n")
```

```
## 20.07967 0.005871745
```

```
cat("\nFor x = 0.44 and n = 3:\n")
```

```
##
## For x = 0.44 and n = 3:
```

```
cat(taylor_expansion(0.44,3),"\n")
```

```
## 1.550997 0.001709885
```

7. Write two R functions to compute the factorial of any user-supplied positive integer n. The first function should calculate (and return) n! using a for loop. The second function calculates and returns n! recursively.

```
factorial_loop <- function(n)
{
  result <- 1
  if (n < 0)
  {
    return("NULL")
  }
  else if (n == 0)
```

```r
  {
    return(1)
  }
  else
  {
    for (i in 1:n)
      {
      result <- result * i
      }
    return(result)
  }
}

#n <- readline()
factorial_loop(5)
```

```
## [1] 120
```

```r
factorial_recursion <- function(n)
{
  if (n < 0)
  {
    return("NULL")
  }
  else if (n == 0)
  {
    return(1)
  }
  else
  {
    return(n * factorial_recursion(n - 1))
  }
}

#n <- as.integer(readline())
factorial_recursion(5)
```

```
## [1] 120
```

8. Write a function that accepts a vector of numbers (of length >= 4) and returns a vector of moving averages.

```r
average_of_vector <- function(vec)
{
  len <- length(vec)
  if(len<4)
    return("Length less than 4 is invalid")
  else
    avg_vec <- numeric(len-3)
    for(i in 1:(len-3))
```

```
    {
      avg_vec[i] <- sum(vec[i:(i+3)])/4
    }
    return(avg_vec)
}

y <- c(1, 1, 2, 5, 8, 3, 4, -4, 3, 7, 2, 2, -2, 1)
average_of_vector(y)
```

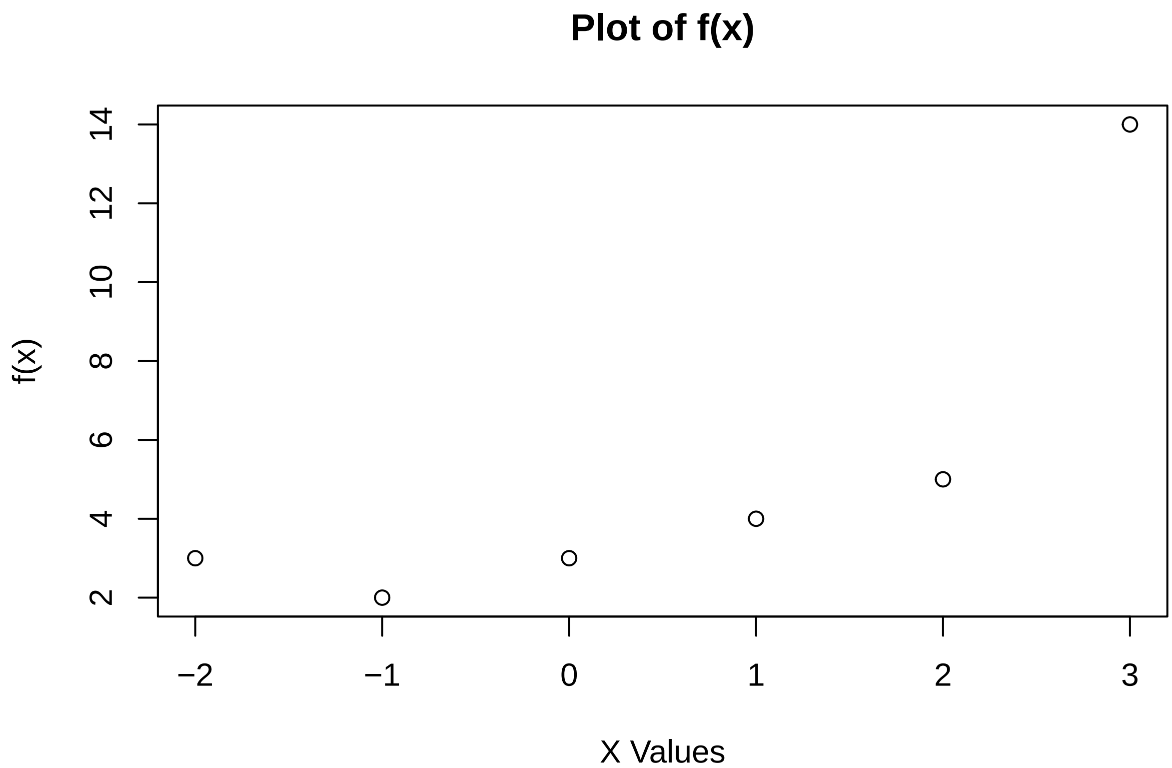
```
##  [1] 2.25 4.00 4.50 5.00 2.75 1.50 2.50 2.00 3.50 2.25 0.75
```

9. Write an R function that accepts a vector of numbers and returns the values of the function f(x) evaluated using that vector of numbers. Use your R function to make a plot of f(x)

```
fn_x <- function(vec3)
{
  result_vec <- numeric(length(vec3))
  for (i in 1:length(vec3))
  {
    x <- vec3[i]
    if(x<0)
      result_vec[i] <- (x^2 + (2*x) + 3)
    else if(x<2)
      result_vec[i] <- x+3
    else
      result_vec[i] <- (x^2 + (4*x) - 7)
  }
  return(result_vec)
}

x <- seq(-2, 3)
fn_x_evaluate <- fn_x(x)

plot(x, fn_x_evaluate, xlab ="X Values", ylab="f(x)", main="Plot of f(x)")
```



10. Generate a dataset using the code below and then follow the example in Lecture set 8 to create an objective function based on the generated dataset and obtain estimates for a and b. Select your own starting values.

```
set.seed(20330)
mydata <- rgamma(100, shape=4, scale=5)
```

```
make.NegLogLik <- function(data, fixed=c(FALSE, FALSE)){
  params <- fixed
  function(param.values){
    params[!fixed] <- param.values
    mu <- params[1]
    sigma <- params[2]
    estimate <- (mu-1)*sum(log(mydata))-length(mydata)*log(gamma(mu))-length(mydata)*mu*log(sigma)-sum(r
  }
}
my.negloglik <- make.NegLogLik(mydata)
optim(c(mu=4,sigma=5), my.negloglik)$par
```

```
##        mu     sigma
## 171.61448  37.44566
```