HW_4_STA_445_S24

Matthew Hashim

2024-03-19

```
library(tidyverse)
```

Conjunction Junction! This assignment is all about functions!(Chapter 10)

Problem 1 (A Warmup Problem)

a. Create a function with two inputs, a and b, that returns the output $b \times a!$. Name the function prob1a.fun.

```
prob1a.fun <- function(a, b) {
  return (b*a)
}
```

b. We will test the function. Run prob1a.fun(5, 4). Did you get the correct result?

```
prob1a.fun(5, 4)
```

```
## [1] 20
```

c. Create a function with two inputs, a and b, that returns the output $b \times a!$ if $a > b$ and returns $b - a$ if $b \geq a$. Name the function prob1c.fun

```
prob1c.fun <- function(a, b) {
  ifelse( a > b, return(b*a), return(b-a))
}
```

d. We will test the function. Run prob1c.fun(5, 4). Did you get the correct result? Run prob1c.fun(4, 5). Did you get the correct result?

```
prob1c.fun(5, 4)
```

```
## [1] 20
```

```
prob1c.fun(4, 5)
```

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

Problem 2 (Writing Functions for Computational Efficency)

Write a function that calculates the density function of a Uniform continuous variable on the interval (a, b). The function is defined as

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

We want to write a function `duniform(x, a, b)` that takes an arbitrary value of `x` and parameters a and b and return the appropriate height of the density function. For various values of `x`, `a`, and `b`, demonstrate that your function returns the correct density value.

a. Write your function without regard for it working with vectors of data. Demonstrate that it works by calling the function three times, once where $x < a$, once where $a < x < b$, and finally once where $b < x$. Make sure to show the output for the three tests.

```
duniform <- function(x, a, b) {
  ifelse( x < a, return(0), ifelse( x <= b, return(1/(b-a)), return(0)))
}
duniform(1, 2, 4)
```

```
## [1] 0
```

```
duniform(3, 2, 4)
```

```
## [1] 0.5
```

```
duniform(5, 2, 4)
```

```
## [1] 0
```

b. Next we force our function to work correctly for a vector of `x` values. Modify your function in part (a) so that the core logic is inside a `for` statement and the loop moves through each element of `x` in succession.

```
duniform <- function(x, a, b) {
  temp <- numeric(length(x))
  for(i in 0:length(x))
  {
    ifelse( x[i] < a, temp[i] <- 0,
            ifelse( x[i] <= b, temp[i] <- 1/(b-a), temp[i] <- 0))
  }
  return(temp)
}
```

Verify that your function works correctly by running the following code:

```
        data.frame( x=seq(-1, 12, by=.001) ) %>%
          mutate( y = duniform(x, 4, 8) ) %>%
          ggplot( aes(x=x, y=y) ) +
          geom_step()
```

c. Install the R package `microbenchmark`. We will use this to discover the average duration your function takes.

```
microbenchmark::microbenchmark( duniform( seq(-4,12,by=.0001), 4, 8), times=100)
```

The median run time was 585 ms

This will call the input R expression 100 times and report summary statistics on how long it took for the code to run. In particular, look at the median time for evaluation.

d. Instead of using a `for` loop, it might have been easier to use an `ifelse()` command. Rewrite your function to avoid the `for` loop and just use an `ifelse()` command. Verify that your function works correctly by producing a plot, and also run the `microbenchmark()`. Which version of your function was easier to write? Which ran faster?

```
duniform <- function(x, a, b){
  return(ifelse(x>=a & x<=b, 1/(b-a), 0))
}
```

```
microbenchmark::microbenchmark( duniform( seq(-4,12,by=.0001), 4, 8), times=100)
```

```
## Unit: milliseconds
##                                    expr    min      lq     mean  median     uq
##  duniform(seq(-4, 12, by = 1e-04), 4, 8) 4.6101 5.99175 7.999439 6.79405 10.163
##     max neval
##  18.524   100
```

The second function was much faster and easier to write after figuring out how to.

Problem 3 (Modify your Uniform Function)

I very often want to provide default values to a parameter that I pass to a function. For example, it is so common for me to use the `pnorm()` and `qnorm()` functions on the standard normal, that R will automatically use `mean=0` and `sd=1` parameters unless you tell R otherwise. To get that behavior, we just set the default parameter values in the definition. When the function is called, the user specified value is used, but if none is specified, the defaults are used. Look at the help page for the functions `dunif()`, and notice that there are a number of default parameters. For your `duniform()` function provide default values of `0` and `1` for `a` and `b`. Demonstrate that your function is appropriately using the given default values by producing a plot by running the code chunk below.

```
duniform <- function(x, a = 0, b = 1){
  return(ifelse(x>=a & x<=b, 1/(b-a), 0))
}
```

```
        data.frame( x=seq(-1, 2, by=.001) ) %>%
          mutate( y = duniform(x) ) %>%
          ggplot( aes(x=x, y=y) ) +
          geom_step()
```

```r
# I found a faster version of the dunif. This one averages 4ms.
# The ifelse version averages 8ms
duniform <- function(x, a = 0, b = 1) {
  temp <- x>=a & x<=b
  temp <- temp/(b-a)
  return(temp)
}
```

Problem 4

Note: We will use this function when we create a package. Don't forget where you save this assignment.

In this example, we'll write a function that will output a vector of the first n terms in the child's game *Fizz Buzz*. The input of the function is a positive integer n and the output will be a vector of characters. The goal is to count as high as you can, but for any number evenly divisible by 3, substitute "Fizz" and any number evenly divisible by 5, substitute "Buzz", and if it is divisible by both, substitute "Fizz Buzz". So the sequence will look like 1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, ...

a. Write the function and name it FizzBuzz.

```r
FizzBuzz <- function(n) {
  output = NULL
  for(i in 1:n)
  {
    if(i %% 3 == 0 & i %% 5 == 0)
    {
      output[i] <- "Fizz Buzz"
    }
    else if(i %% 3 == 0)
    {
      output[i] <- "Fizz"
    }
    else if ( i %% 5 == 0)
    {
      output[i] <- "Buzz"
    }
    else
    {
      output[i] <- i
    }
  }
  return(output)
}
```

b. Test the function by running FizzBuzz(50). Did it work?

```r
FizzBuzz(50)
```

```
##  [1] "1"         "2"         "Fizz"      "4"         "Buzz"      "Fizz"
##  [7] "7"         "8"         "Fizz"      "Buzz"      "11"        "Fizz"
## [13] "13"        "14"        "Fizz Buzz" "16"        "17"        "Fizz"
## [19] "19"        "Buzz"      "Fizz"      "22"        "23"        "Fizz"
```

```
## [25] "Buzz"      "26"        "Fizz"      "28"        "29"        "Fizz Buzz"
## [31] "31"        "32"        "Fizz"      "34"        "Buzz"      "Fizz"
## [37] "37"        "38"        "Fizz"      "Buzz"      "41"        "Fizz"
## [43] "43"        "44"        "Fizz Buzz" "46"        "47"        "Fizz"
## [49] "49"        "Buzz"
```