HW_4_STA_445_S24

April Meadows

2024-03-07

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

```
library("tidyverse")
```

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.

```
# defines a function that prints out the value of b * a!
prob1a.fun <- function(a,b){
  return(b*factorial(a))
}
```

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

```
# tests prob1a with integers
prob1a.fun(5,4)
```

```
## [1] 480
```

```
# The result is 480, which is the correct answer
```

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

```
# defines a function called prob1c.fun with two outcomes
prob1c.fun <- function(a,b){
  return(ifelse(a>b, b*factorial(a),b-a))
  # returns b*a! if a>b or b-a if 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?

```
# tests prob1c.fun's two possible outcomes
prob1c.fun(5,4) # a > b
```

```
## [1] 480
```

```
prob1c.fun(4,5) # b >= a
```

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

```
# the results are 480 for the first test and 1 for the second test
# both of the results are correct
```

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.

```r
# defines duniform function for arbitrary values of x
duniform.fun <- function(x,a,b){
  if(x<a | b<x){return(0)}  # returns 0 if x<a or b<x
  else {return(1/(b-a))}    # returns 1/(b-a) if a<=x<=b
}

# tests duniform function for x<a
  # returns 0 since 1<2 (x<a)
duniform.fun(1,2,3)
```

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

```r
# tests duniform function for a<=x<=b
  # returns 1/(3-1) since 1<=2<=3 (a<=x<=b)
duniform.fun(2,1,3)
```

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

```r
# tests duniform function for b<x
  # returns 0 since 2<4 (b<x)
duniform.fun(4,3,2)
```

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

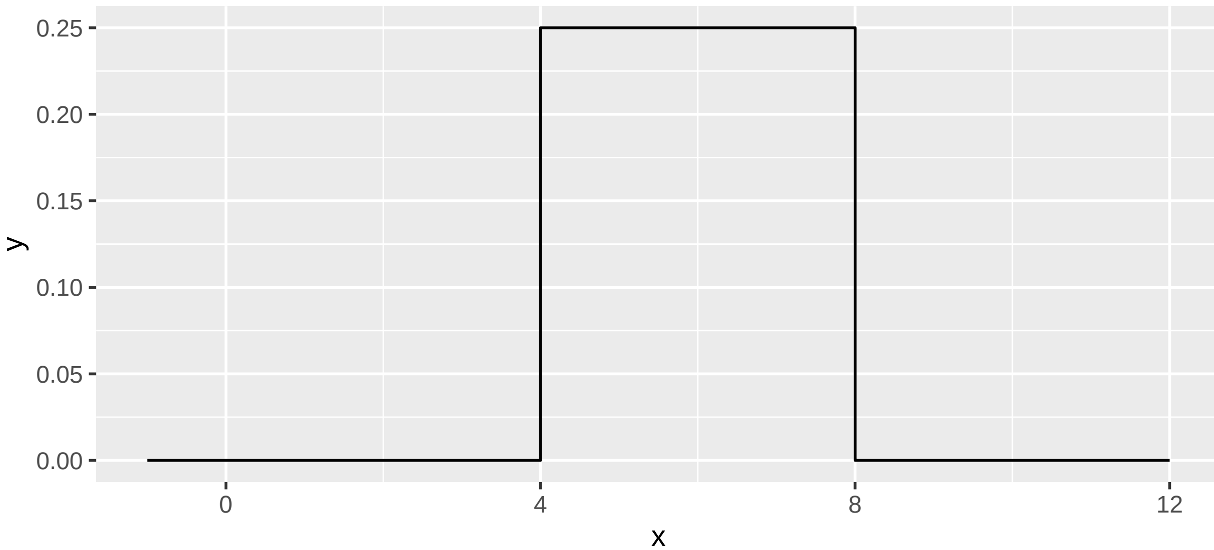
b. Next we force our function to work correctly for a vector of `x` values. Modify your function in part (a) so that the core logic is inside a `for` statement and the loop moves through each element of `x` in succession.

```r
# creates a vector of values for x
x <- 1:10

# defines duniform function for a vector x of values
duniform <- function(x,a,b){
  vec <- c()
  for(i in length(x)){
    val = ifelse(x<a | b<x,0,1/(b-a))
  vec <- append(vec, val)}
  return(vec)
}
```

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

```r
# the following code creates a graph of the density function
        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.

```r
# install microbenchmark package: install.packages("microbenchmark")

# saves the microbenchmark of duniform as m to recall the mean later
m <- summary(microbenchmark::microbenchmark( duniform(
  seq(-4,12,by=.0001), 4, 8), times=100))
```

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?

```r
# defines a new duniform function for a vector x of values
duniform2 <- function(x,a,b){
  vec <- c()
  val = ifelse(x<a | b<x, 0, 1/(b-a))
  vec <- append(vec, val)
  return(vec)
}

# saves the microbenchmark on duniform2 as m2 to print out the mean
m2 <- summary(microbenchmark::microbenchmark( duniform2(
  seq(-4,12,by=.0001), 4, 8), times=100))


print(m$mean) # prints out the mean for the first duniform function
```

```
## [1] 8.392456
```

```r
print(m2$mean) # prints out the mean for the second duniform function
```

```
## [1] 7.593669
```

```r
# the new duniform function was easier to write,
#   but it was slower overall as seen from the outputs below
```

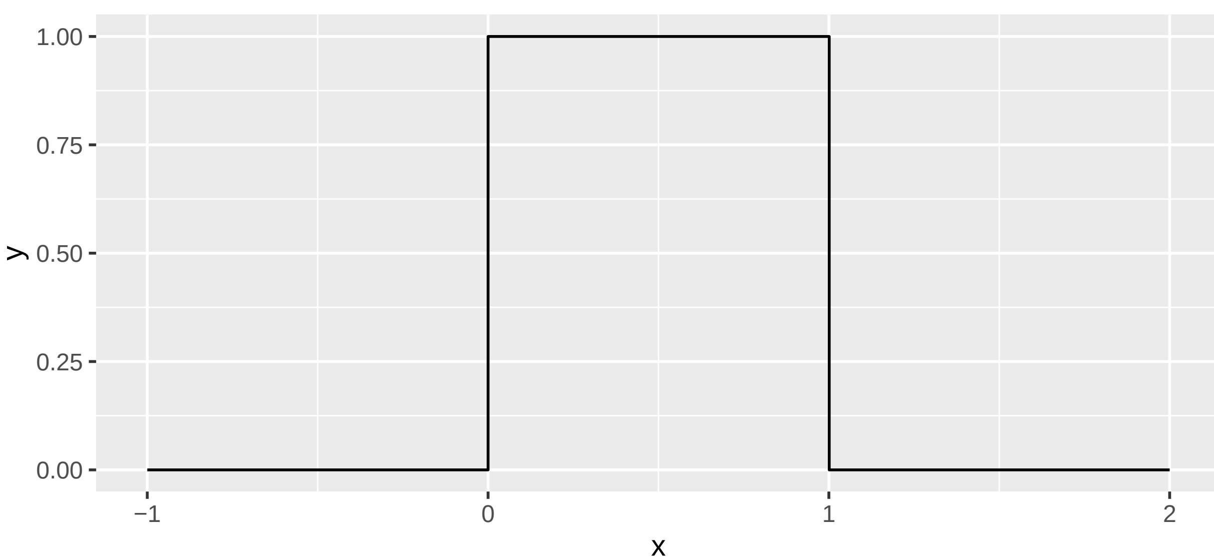
```r
# displays duniform2's density function plot to show they're the same
        data.frame( x=seq(-1, 12, by=.001) ) %>%
          mutate( y = duniform2(x, 4, 8) ) %>%
          ggplot( aes(x=x, y=y) ) +
          geom_step()
```

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.

```r
# defines duniform with default values a=0 and b=1
duniform <- function(x,a=0,b=1){
  vec <- c()
  val = ifelse(x<a | b<x, 0, 1/(b-a))
  vec <- append(vec, val)
  return(vec)
}
```

```r
# displays duniform's density function plot to show the arbitrary values work
        data.frame( x=seq(-1, 2, by=.001) ) %>%
          mutate( y = duniform(x) ) %>%
          ggplot( aes(x=x, y=y) ) +
          geom_step()
```



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
# creates the FizzBuzz function with one numerical input
FizzBuzz <- function(n){
  vect <- c(1:n)
  for(i in vect){
    if(i%%3==0 & i%%5==0){vect[i]="Fizz Buzz"}
    else if(i%%3==0){vect[i]="Fizz"}
    else if(i%%5==0){vect[i]="Buzz"}
  }
  return(vect)
}
```

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

```r
# tests the FizzBuzz function where n=50
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"
```

```r
# the function did work:
    #"Fizz" replaced the numbers divisible by 3
    #"Buzz" replaced the numbers divisible by 5
    #"Fizz Buzz" replaced the numbers divisible by both 3 and 5
```