732A90: Lab 1

David Björelind, davbj395

10/22/2020

1. Be careful when comparing

Good!

```
## [1] "Subtraction is wrong"
```

```
## [1] "Subtraction is correct"
```

The first expression tells us "Subtraction is wrong" and the second expression "Subtraction is correct". The first get it incorrect, because 1/3-1/4 cannot be represented in an exact way in binary. 1-1/2 can be represented correctly, which is why it gets the calculations correct.

Improvements: use the comparing statement 'all.equal()' instead of using '==' will help this.

```
options(digits=20)
1/3-1/4
```

```
## [1] 0.08333333333333331483
```

```
1/12
```

```
## [1] 0.08333333333333328707
```

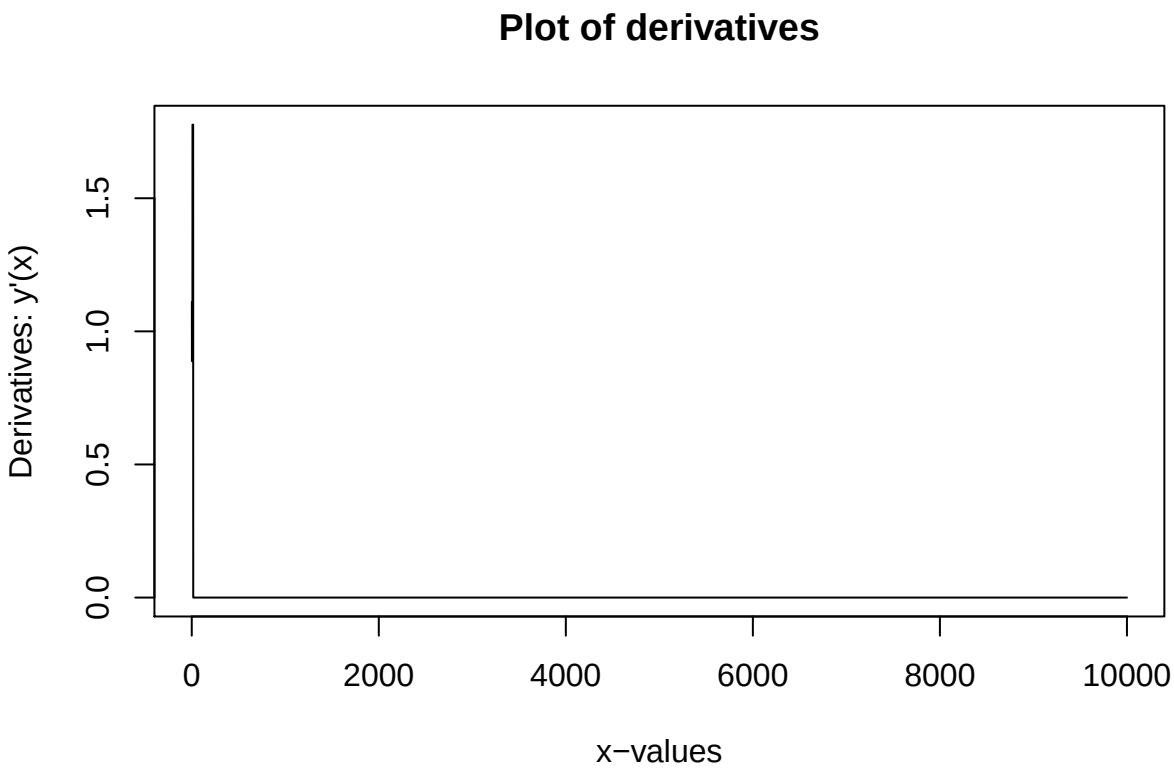
```
x1<- 1/3
x2<- 1/4

# First expression again
if ( all.equal(x1-x2,1/12) ) {
print ("Subtraction is correct")
} else {
print ("Subtraction is wrong")
}
```

```
## [1] "Subtraction is correct"
```

Now it gets it correct!

2. Derivative

Good!



The results are very surprising! The expected answer would be **1** for each value from 1-10000, f'(x) = 1. This is not observed however. The values quickly becomes 0, and this can be explained by underflow. **f(x+e)-f(x)** will produce a very small number. When **x** gets larger, the misrepresentation of **f(x+e)-f(x)** will grow and at some point it will be **0**. Some experimentation:

```
(10+epsilon)-10
```

```
## [1] 1.7763568394002504647e-15
```

```
(15+epsilon)-15
```

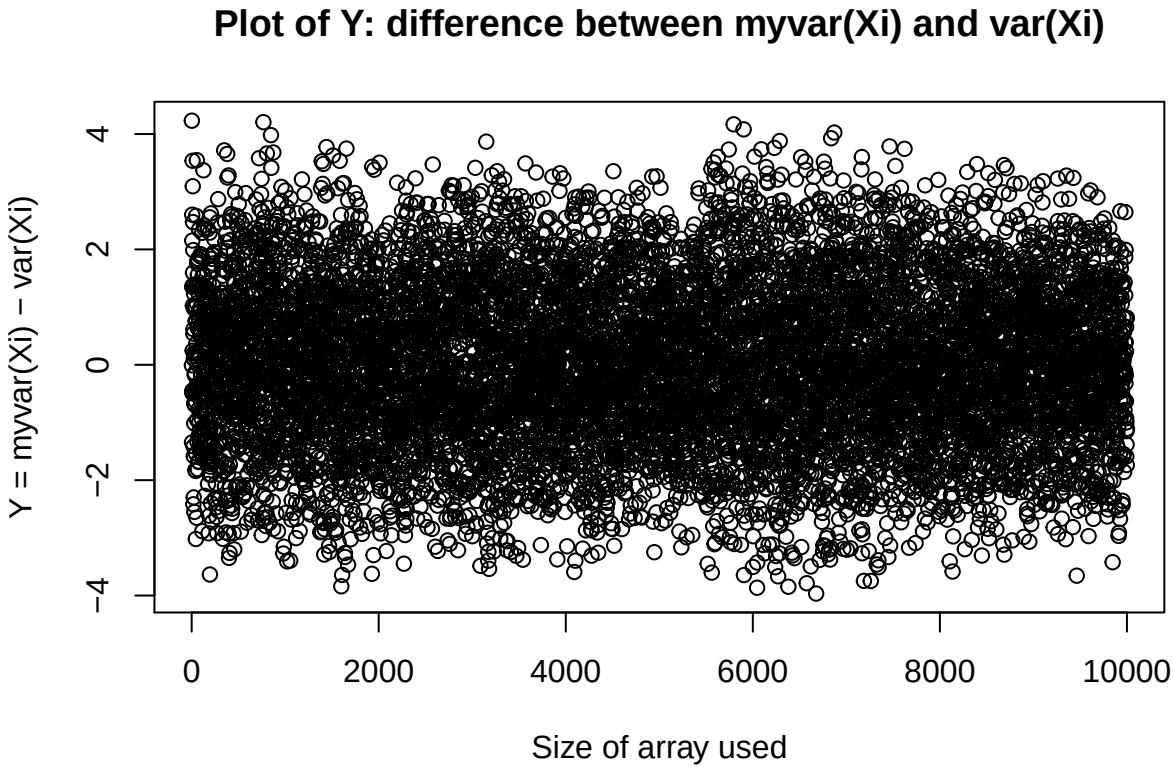
```
## [1] 1.7763568394002504647e-15
```

```
(20+epsilon)-20
```

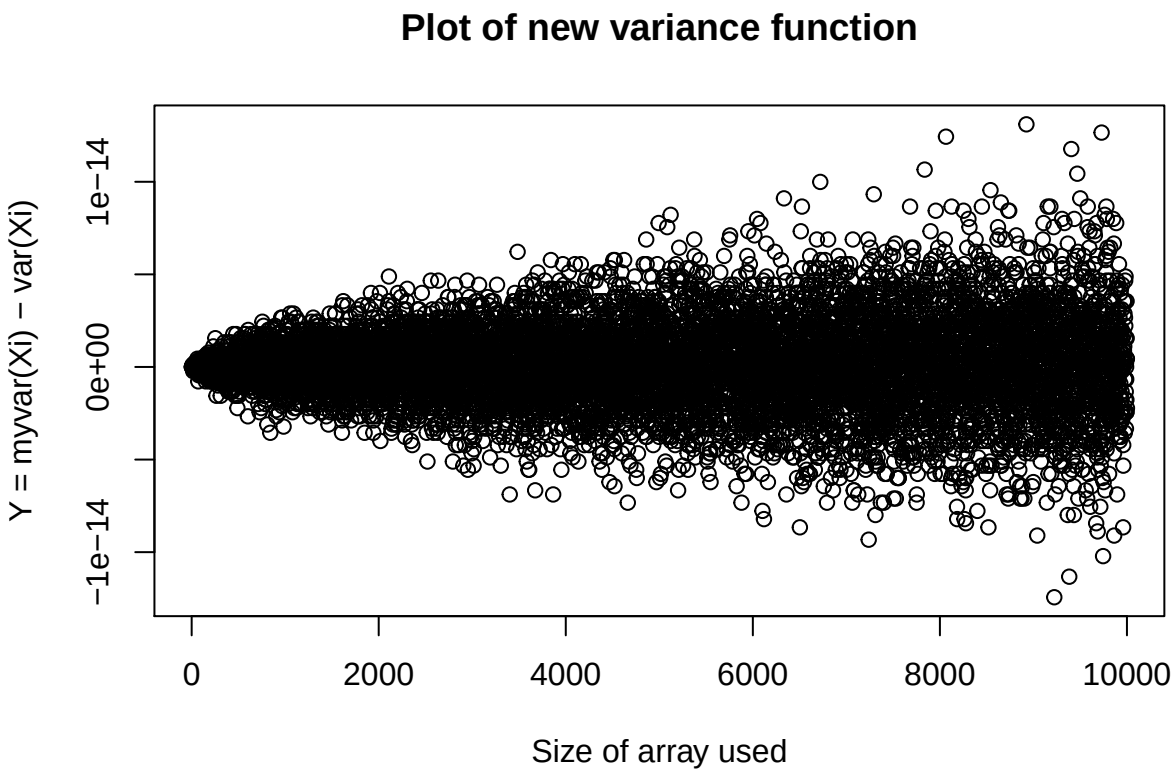
```
## [1] 0
```

3. Variance

The reason why myvar() is not working is not complete. Think on how this large numbers are added (and the sign of such). The argumentation for 3.3 needs fixing.



From the plot we can see that **myvar** differs from **var** +- 4 through the entire array. This difference is quite big since the real variance is **1**. We get the same error size, no matter how many numbers are used for calculation. My function works well on samples of smaller size. Adding many large numbers, which is done here, becomes a problem due to overflow. The middle steps in **myvar**-function becomes misrepresented and therefore making the var-value wrong.

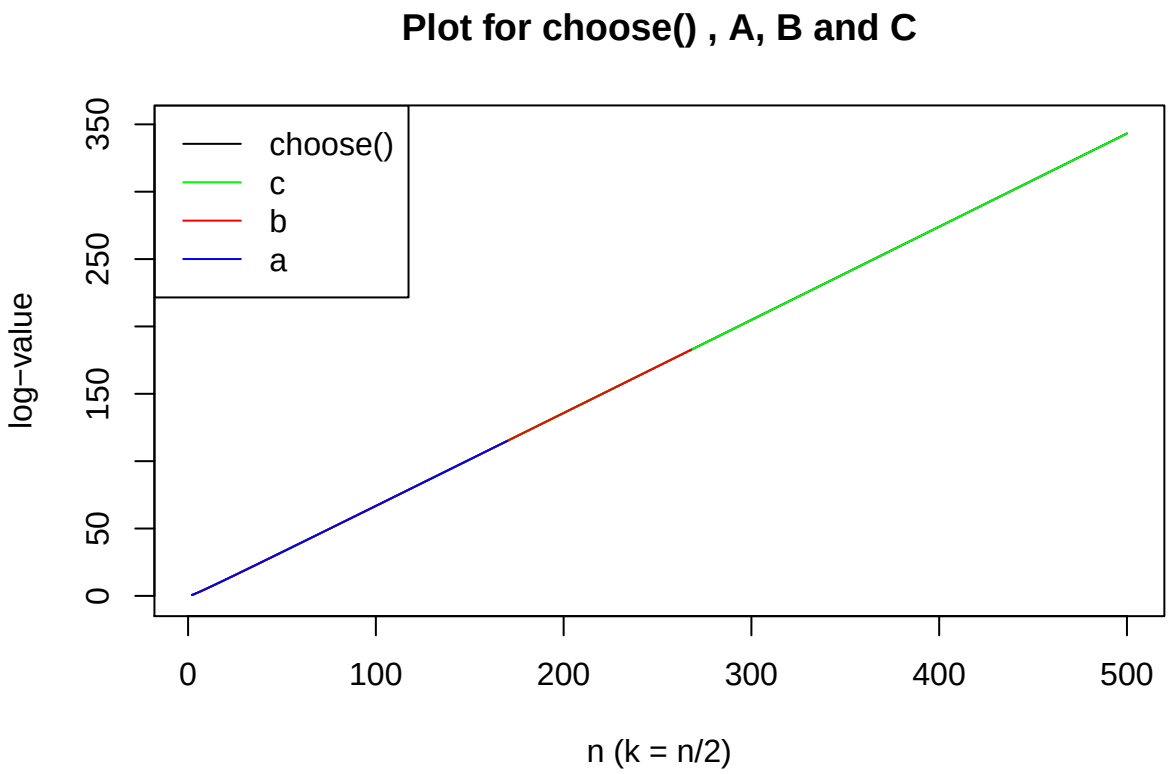


Here, we see a much small spread from the rear **var** function! The difference is that is intermittent steps is divided by **n**. Because of this, the numbers does not get too big. Thus, preventing overflow. However, we still get some overflow, especially for larger array sizes, but not in the same magnitude as before.

Explain why you can see in the plot the overflow.
Overall explanation is vague, be more concise.
This question needs fixing.

4. Binomial Coefficient

After experimenting some with different values I reach the following conclusions. What happens when we move from A to C is than less and less expressions are being calculated via the **prod** function. A: Cannot handle **k=0**. Answer should be **1** but returns **Inf**. This is because **prod(1:0) = 0**. All: When **k=n** the result should be **1**, but all the functions return **Inf**. This occurs because they all contain **prod(1:(n-k))** which becomes **0** when **k=n**.



Note: I did log of all values to make then more manageable and that choose() is plotted behind C in the graph.

From the plot we can see that both **A** and **C** overflows, why? t different values. At around **n=200**, A starts to overflow and at **n=300** B starts to overflow. **C** start to overflow at the same time as **choose()** does. This can be explained by the terms used in calculation. The term **prod(1:n)** makes **A** overflow first and **prod((k+1):n)** makes **B** overflow. None of these term occurs in **C**, it avoids multiplying the largest numbers. Thus, preventing overflow.

All code for this report

```
knitr::opts_chunk$set(echo = TRUE, warning=FALSE, message=FALSE)
# Include packages here


x1<- 1/3
x2<- 1/4

# First expression
if ( x1-x2 == 1/12 ) {
print ("Subtraction is correct")
} else {
print ("Subtraction is wrong")
}
```

```r
# Second expression
x1 <- 1
x2 <- 1/2
if ( x1-x2 == 1/2 ) {
print ("Subtraction is correct")
} else {
print ("Subtraction is wrong")
}
options(digits=20)
1/3-1/4
1/12

x1<- 1/3
x2<- 1/4

# First expression again
if ( all.equal(x1-x2,1/12) ) {
print ("Subtraction is correct")
} else {
print ("Subtraction is wrong")
}
derivative = function(x, func, epsilon){
  return((func(x+epsilon)-func(x))/epsilon)
}
f = function(x){
  return(x)
}

epsilon = 10^-15

# Evaluating derivative
ders = c()
for (i in 1:10000){
  temp = derivative(i, f, epsilon)
  ders = c(ders, temp)
}

plot(x=1:10000, y=ders, type = 'l', main="Plot of derivatives", xlab="x-values", ylab="Derivatives: y'(
(10+epsilon)-10
(15+epsilon)-15
(20+epsilon)-20
# Variance function
myvar = function(x){
  n = length(x)
  ave = sum(x)/n
  temp = sum(x^2-ave^2)
  return(temp/(n-1))
}
# Generating numbers
xx = rnorm(n=10000, mean=10^8, sd=sqrt(1))

y = c()
normvar = c()
```

```
for (i in 2:10000){
  my_var = myvar(xx[1:i])
  vardif = my_var - var(xx[1:i])
  y = c(y, vardif)
  normvar = c(normvar, var(xx[1:i]))
}
# Plottning the results
plot(x=2:10000, y=y, main="Plot of Y: difference between myvar(Xi) and var(Xi)", xlab = "Size of array u
# Making a new var-function
myvar_new = function(x){
  mean = mean(x)
  n = length(x)
  var = 0
  for (i in 1:n){
    var = var + ((x[i]-mean)^2)/(n-1)
  }
  return(var)
}
# Generating numbers
xx = rnorm(n=10000, mean=10^8, sd=sqrt(1))

y = c()
normvar = c()
for (i in 2:10000){
  my_var = myvar_new(xx[1:i])
  vardif = my_var - var(xx[1:i])
  y = c(y, vardif)
}
# Plottning the results
plot(x=2:10000, y=y, main="Plot of new variance function", xlab = "Size of array used", ylab = "Y = myva
# Writing all the functions
aa = function(n, k){
  return(prod(1:n) / (prod(1:k) * prod(1:(n-k))))
}
bb = function(n, k){
  return(prod((k+1):n) / (prod(1:(n-k))))
}
cc = function(n, k){
  return(prod(((k+1):n) / (1:(n-k))))
}
# Playing around with large values for n and k first

a = c()
b = c()
c = c()
choose = c()
for (n in 2:500){
  k = floor(n/2)
  #k = n - 15
  a = c(a, aa(n, k))
  b = c(b, bb(n, k))
  c = c(c, cc(n, k))
  choose = c(choose, choose(n, k))
```

```
}
#log
a = log(a)
b = log(b)
c = log (c)
choose = log(choose)

plot(x=2:500, y=choose, main="Plot for choose() , A, B and C", col="black", type='l', ylim=c(-1, 350), 
lines(x=2:500, y=c, col="green", type='l')
lines(x=2:500, y=b, col="red", type='l')
lines(x=2:500, y=a, col="blue", type='l')
legend("topleft", legend=c("choose()", "c", "b", "a"), lwd=c(1,1,1,1),col=c("black", "green", "red", "bl
```