Computational Statistics Lab1

Yuki Washio and Nicolas Taba

02/11/2020

Question 1: Be Careful when comparing

We are asked to evaluate the following code snippet

```
x1 <- 1/3 ; x2 <- 1/4
if ( x1 - x2 == 1/12){
  print("substraction is correct")
}else{
  print("substraction is wrong")
}
```

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

And this code snippet

```
x1 <- 1 ; x2 <- 1/2
if ( x1 - x2 == 1/2){
  print("substraction is correct")
}else{
  print("substraction is wrong")
}
```

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

Some explanation about the reason why this is. It probably has to do with how the 1/3 value is stored in a finite manner and that means that it's not really equal

One possibility to improve the first code is doing the following:

```
x1 <- 1 ; x2 <- 1/2
if (isTRUE(all.equal( x1 - x2, 1/2))){
  print("substraction is correct")
}else{
  print("substraction is wrong")
}
```

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

This works because the **all.equal** method tests for "near equality" between the two quantities instead of exact results.

Question 2: Derivative

Here, we evaluate the derivative of $f(x) = x$ by writing out own R function.

```
epsilon = 10^(-15)

derivative <- function(x){
  stopifnot(is.numeric(x))
  dx <- ((x+epsilon) - x)/(epsilon)
  return(dx)
}
```

We now evaluate for $x = 1$ and $x = 100000$

```
x1 = 1
x2 = 100000
derivative(x1)
```

```
## [1] 1.110223
```

```
derivative(x2)
```

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

The true value of the derivative is 1. For $x = 1$, ((I think it's the same mechanism than for x = 100000, but it's less clear to me exactly why since x is small to start with this time around. . . .)) For $x = 100000$, we have the difference in the nominator that is evaluated to be zero and the division by ϵ, which is small and doesn't get stored properly.

Question 3: Variance

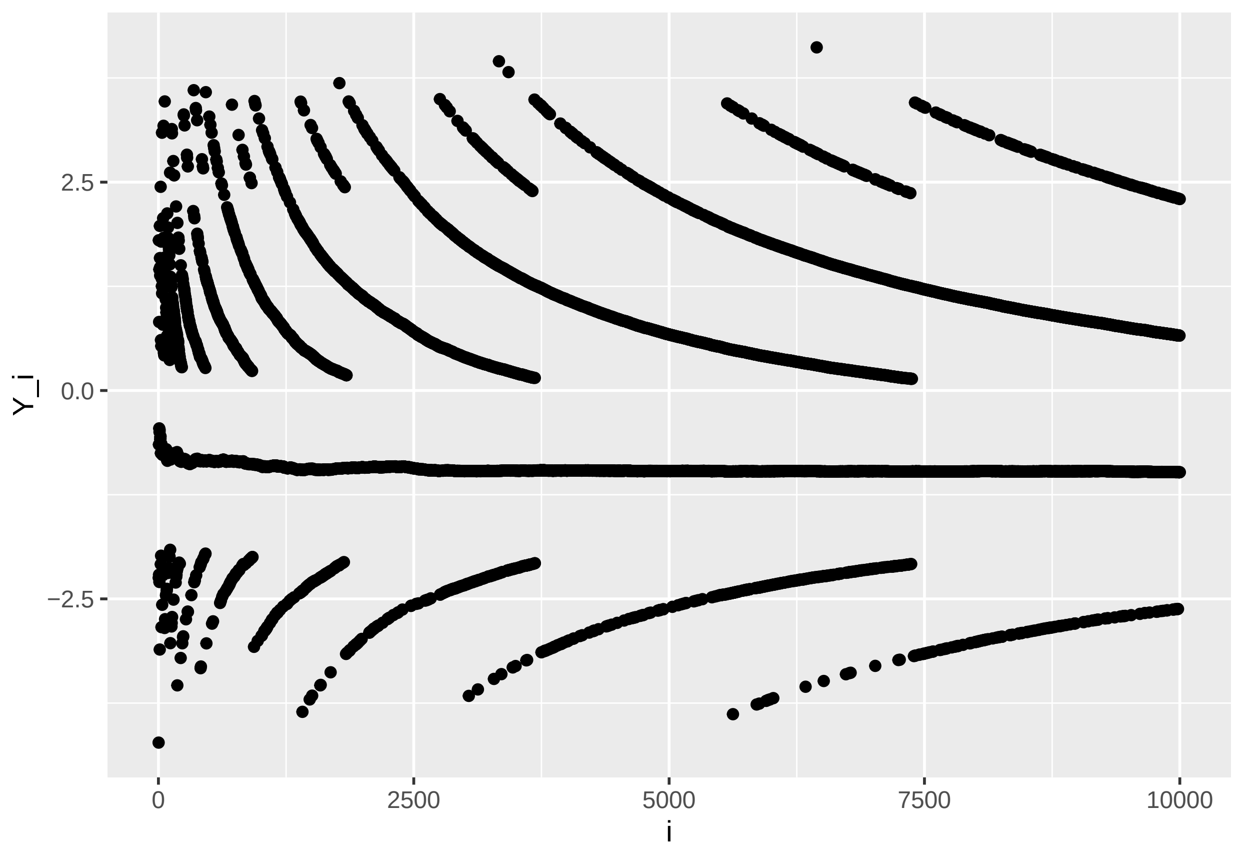
This time, we are asked to write a **myvar** function to calculate the variance based on a vector given by the exercise. We then generate a vector with 10000 random numbers with mean 10^8 and variance 1. We then compare our function to the in-built function **var** in R. We plot the dependence of the difference with respect to the length of the vector.

```
myvar <- function(x){
  stopifnot(is.vector(x),
            is.numeric(x))
  n <- length(x)
  variance <- (1/(n-1))*(sum(x^2) - (1/n)*((sum(x))^2))
  return(variance)
}

vect_x <- rnorm(10000, mean = 10^8, sd = 1)

Y <- c()
for (i in 1:length(vect_x)){
  Y[i] <- myvar(vect_x[1:i]) - var(vect_x[1:i])
}
```

and now we plot the dependence of Y_i on i.

```
data_Y <- data.frame(values = Y, len = 1:length(Y))
plot_dependence <- ggplot(data = data_Y)+
  geom_point(aes(x = len, y = values))+ xlab("i")+ylab("Y_i")

print(plot_dependence)
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



If myvar was a good estimate and close to the var function in R, we would expect to see all values of their difference to be equal to 0. This graph shows that the values are concentrated around -1. No clue how to explain this behavior. . . .

We can improve the calculation of the variance by remembering that the sample variance is

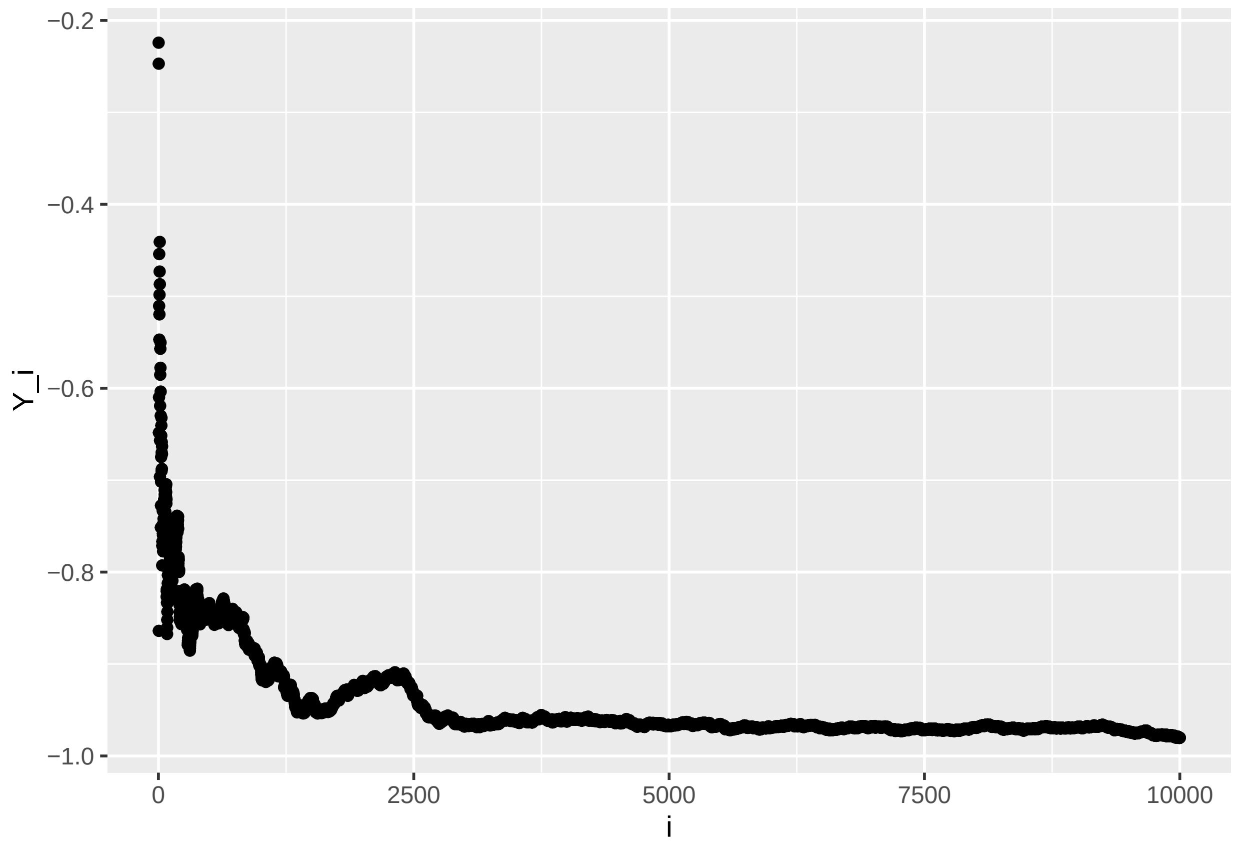
$$V[X] = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

. This is a good estimate of the variance of a population if the sample is large enough (which is our case). We can thus improve our variance calculation in the following way:

```r
myvar_improved <- function(x){
  stopifnot(is.vector(x),
            is.numeric(x))
  mu <- mean(x)
  n <- length(x)
  variance_improved <- (sum(x - mu)^2)/(n-1)
  return(variance_improved)
}

Y_improved <- c()
for (i in 1:length(vect_x)){
  Y_improved[i] <- myvar_improved(vect_x[1:i]) - var(vect_x[1:i])
}

data_Y_improved <- data.frame(values = Y_improved, len = 1:length(Y_improved))
```

```r
plot_dependence_improved <- ggplot(data = data_Y_improved)+
  geom_point(aes(x = len, y = values))+ xlab("i")+ylab("Y_i")

print(plot_dependence_improved)
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



((Not certain why we're left with all the garbage there. The sample variance SHOULD be a good estimate of the variance))

Question 4: Binomial coefficient

((For A, I think the n-k term might lead to some issues if k is much smaller than n. Also if n-k = 0, we get an issue with the division)) ((Not sure what other mechanism there is. The question seems to be hinting at more than just the one)).

```r
# Formulae A
binom_a <- function(n,k){
  stopifnot(is.numeric(n), is.numeric(k))
  coeff_a <- prod(1:n) / (prod(1:k) * prod(1:(n-k)))
  return(coeff_a)
}
# Formulae B
binom_b <- function(n,k){
  stopifnot(is.numeric(n), is.numeric(k))
  coeff_b <- prod((k+1):n) / prod(1:(n-k))
  return(coeff_b)
}
# Formulae C
binom_c <- function(n,k){
  stopifnot(is.numeric(n), is.numeric(k))
  coeff_c <- prod(((k+1):n) / (1:(n-k)))
  return(coeff_c)
}


# What a surprise!! R doesn't like vectors that have a length of 10^12
n <- c(2, 10, 10^2, 10^4)
k_rand <- sample(1:2, 4, replace = TRUE)
k1 <- n - k_rand
k2 <- c(0, 1, 2, 4)

for(i in 1:4){
  small_a <- c()
  small_b <- c()
  small_c <- c()

  large_a <- c()
  large_b <- c()
  large_c <- c()
  # small difference between n and k
  small_a[i] <- binom_a(n[i], k1[i])
  small_b[i] <- binom_b(n[i], k1[i])
  small_c[i] <- binom_c(n[i], k1[i])

  #big difference between n and k
  large_a[i] <- binom_a(n[i], k2[i])
  large_b[i] <- binom_b(n[i], k2[i])
  large_c[i] <- binom_c(n[i], k2[i])
}

print(small_a)
```

```
## [1]  NA  NA  NA NaN
```

```r
print(small_b)
```

```
## [1]       NA       NA       NA 49995000
```

```r
print(small_c)
```

```
## [1]       NA       NA       NA 49995000
```

```r
print(large_a)
```

```
## [1]  NA  NA  NA NaN
```

```r
print(large_b)
```

```
## [1]  NA  NA  NA NaN
```

```r
print(large_c)
```

```
## [1]           NA           NA           NA 4.164167e+14
```