Assignment I - CompStat2023

Name of the Team

Team Member A, Team Member B, Team Member C

Contents

I was about to mara 5

1 Set up

```
library(dplyr)
library(ggplot2)
# ...
```

2 Exercise 1

A potential solution might be the following... I know that $\mathbb{E}[X] = \int x f_X(x) dx$ and that $Var(X) \geq 0$.

2.1 Point a

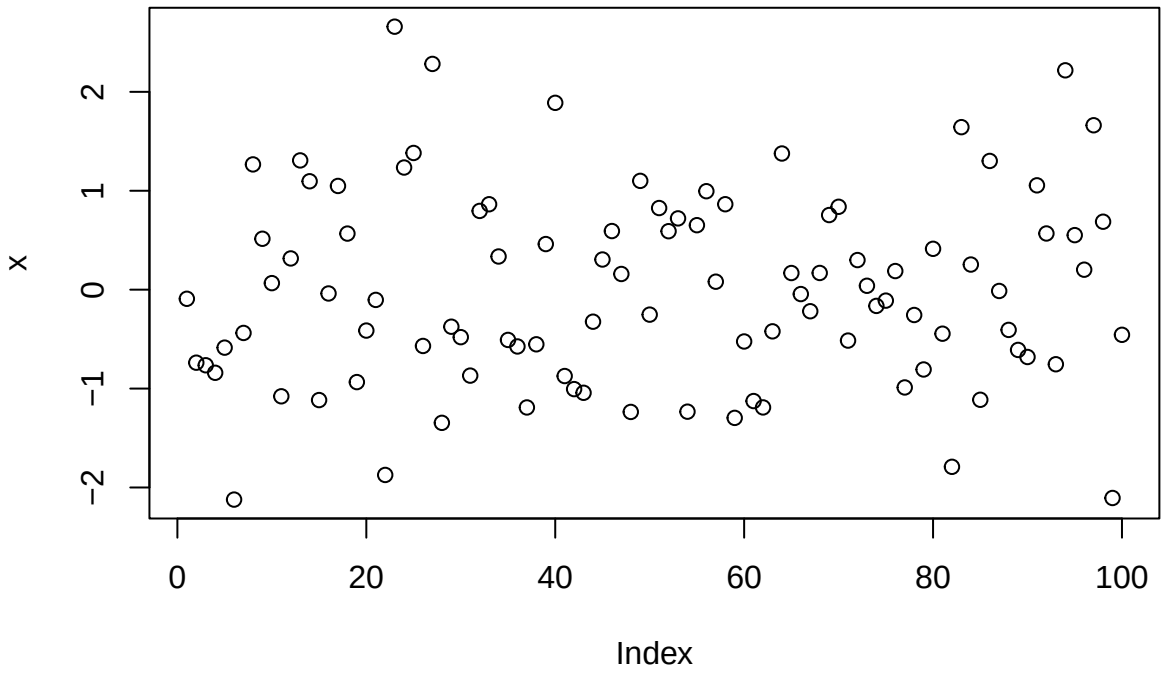
Therefore, we can say that

$$Var(X) = \int (x - \mu)^2 f_X(x) dx.$$

2.2 Point b

We can implement also some code:

```
x <- rnorm(100)
plot(x)
```



Comment of my results

3 Exercise 2

The other solutions

4 Exercise 4: MC integration

4.1 Point a

Consider a standard Normal random variable X:

$$X \sim \mathcal{N}(\mu = 0, \sigma^2 = 1) \tag{1}$$

4.2

If we want to calculate $\mathbb{P}(X > 20)$ we could try to integrate the PDF of X in this way:

$$\mathbb{P}(X > 20) = 1 - \int_{-\infty}^{20} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx = \int_{20}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx \tag{2}$$

We immediately see that that function is not so easy to be integrated because that is the son of the Gaussian function e^{x^2}. But if we try to estimate this with the Monte Carlo method but although this is extremely powerful and flexible, there are some situations in which it can fail.

The reason why the Monte Carlo crude method can fail in the case of a standard normal distributed random variable X, in particular in the estimation of the quantity $\mathbb{P}(X > 20)$ is the structure of this distribution. The standard normal is continuous long-tailed distribution (extended to infinity and decreasing very slowly), and the probability of getting a value (in this case, 20) very far from the mean (0 for definition) is extremely low, but not exactly zero. This means that to obtain an accurate estimate of the probability that this random variable is greater than 20, this method requires a large number of random numbers, which would require a significant amount of computation time and the approximation error could be very large.

Additionally, the Monte Carlo method requires the functions to be evaluated to be integrated or summed over the entire sampled space. The probability density function of a standard normal distribution does not have an analytical solution and therefore integration would require the use of numerical techniques that would further increase computational complexity.

Furthermore, the Crude Monte Carlo method uses a uniform distribution to generate random samples, which means that the samples are generated evenly spaced in the specified interval. However, the standard normal distribution has a particular shape this particular shape of the distribution makes it difficult to generate random samples sufficiently far from the mean to estimate the probability of very rare events.

4.3 Point b

Considering the change of variable $Y = \frac{1}{X}$ the integral will become:

$$\mathbb{P}(X > 20) = \mathbb{P}(Y < 20) = \int_0^{1/20} \frac{1}{\sqrt{2\pi} * y^2} e^{-\frac{(1/y)^2}{2}} dy \tag{3}$$

because $f_Y(y) = f_X(g(y))|\frac{\delta g(y)}{\delta y}|$ where:

- $g(y) = \frac{1}{y}$
- $\frac{\delta g(y)}{\delta y} = -\frac{1}{y^2}$
- $f_X(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$

As the first one, this integral has not have an analytic solution so we use the Monte Carlo method to estimate. The idea is to generate N random numbers from a uniform distribution between 0 and 1/20 that we obtain from the change of variable.

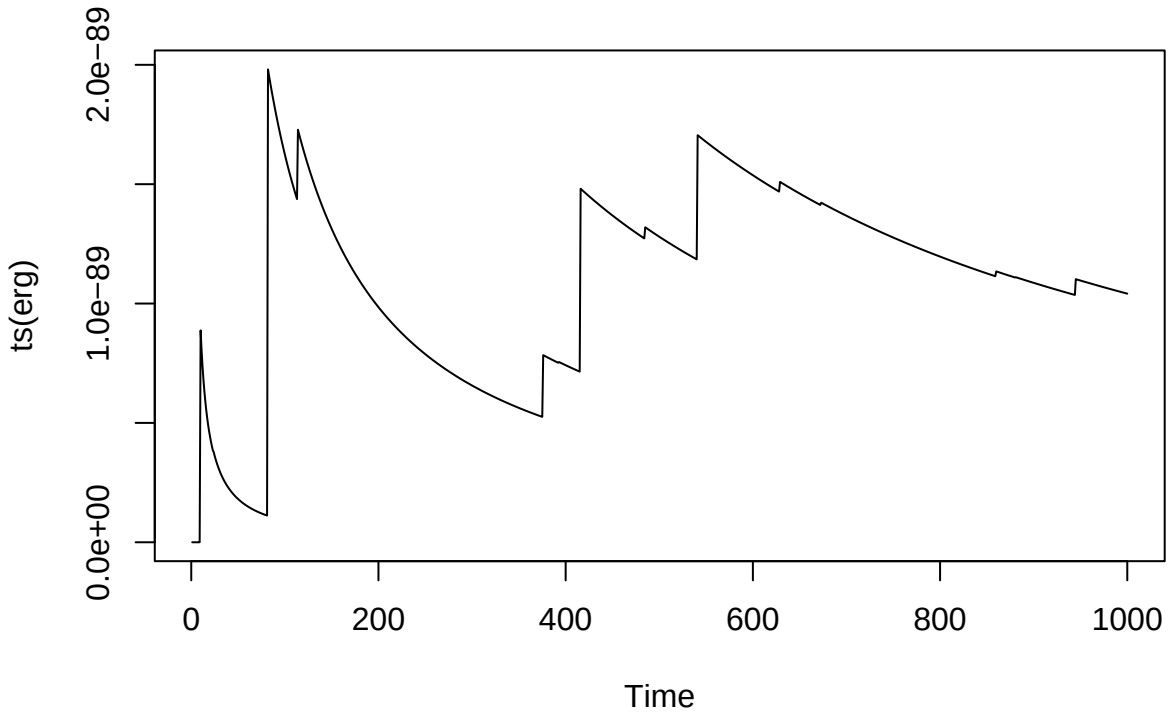
$$\mathbb{P}(X > 20) \approx \frac{0.05 - 0}{N} \sum_{i=1}^{N} \frac{1}{\sqrt{2\pi} y_i^2} \cdot e^{-\frac{(1/y_i)^2}{2}} \tag{4}$$

```
rm(list = ls())
set.seed(1234)

f = function(x) 1/(sqrt(2*pi)*x^2)*exp(-(1/x)^2/2)
a = integrate(f,0,1/20)
n = 100

mc = function(n, a=0, b=1/20){

  if(a>b){stop("reconsider your interval")}

  x = runif(n,a,b)

  mean(f(x)) * (b-a)

}


mc(1000)
```

```
## [1] 3.258382e-89
```

```
ergodic = function(n_max, a=0, b=1/20){
```

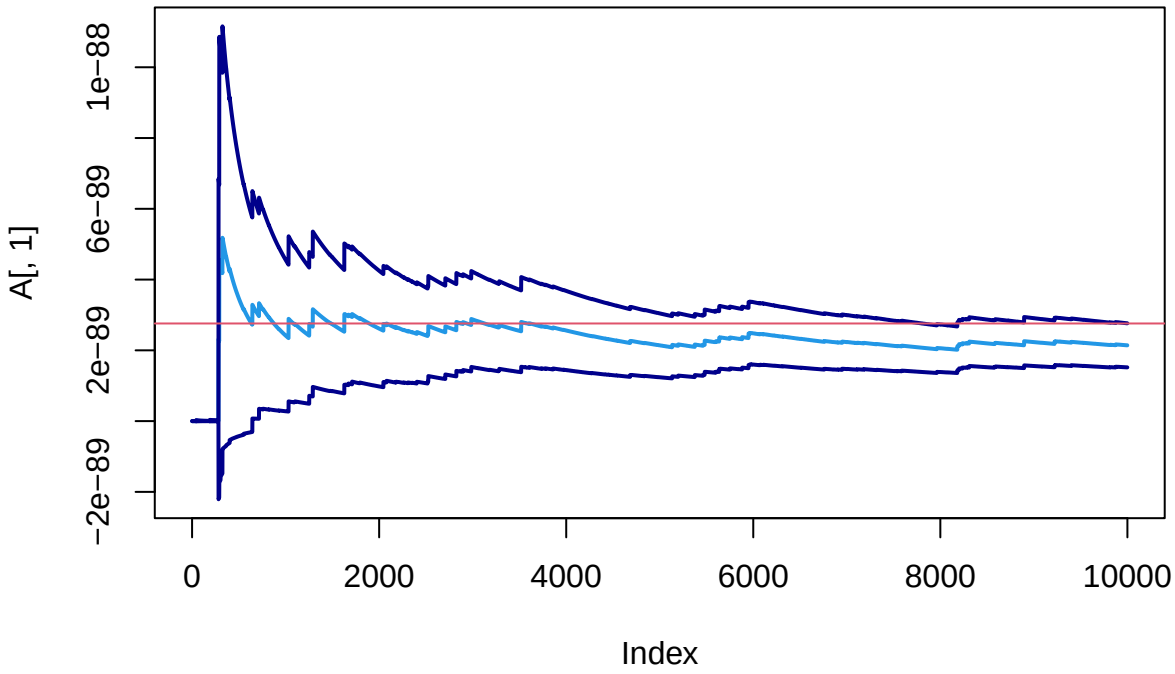
```
  x = runif(n_max,a,b)

  (b-a) * cumsum(f(x))/c(1:n_max)#seq(1,n_max,by=1)
  #dplyr::cummean()
}

erg = ergodic(1000)
plot(ts(erg))
```



##Point c Now we try to construct a more efficient estimator using antithetic variables it will be more efficient because it reduces the variance of the estimator because we are using 2 unbiased estimators identically distributed but negatively correlated and for definition the variance decreases.

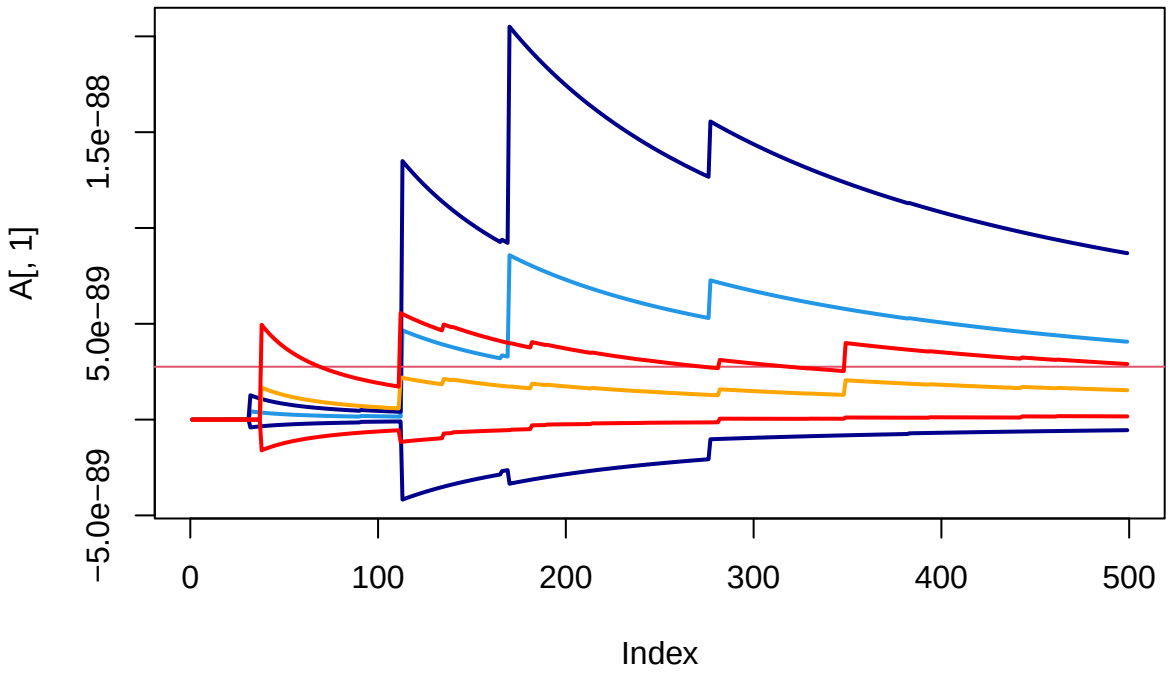
```
cumsd_f = function(x){

  sapply(2:length(x), function(r) sd(f(x[1:r])))

}


ergodic2 = function(n_max, a=0, b=1/20){

  x = runif(n_max,a,b)

  thetahat = (b-a) * cumsum(f(x))/c(1:n_max)#seq(1,n_max,by=1)
  #dplyr::cummean()

  sdthetahat = (b-a)/sqrt(2:n_max) * cumsd_f(x)

  return(cbind(thetah = thetahat[-1], sd = sdthetahat))
}
```

```r
plot_ergodic = function(output,theta){

  A = output
  x_top =    max(A[,1]+ 1.96 * A[,2])
  x_bottom = min(A[,1]- 1.96 * A[,2])

  plot(A[,1],type="l",col=4,lwd=2,ylim = c(x_bottom,x_top))
  lines(A[,1] + 1.96 * A[,2],
        type="l",col="darkblue",lwd=2)
  lines(A[,1] - 1.96 * A[,2],
        type="l",col="darkblue",lwd=2)

  abline(h=theta,col=2)

}


output = ergodic2(10000)
plot_ergodic(output,theta = a[1])
```




```r
MC_ergodic = function(n_max, a=0, b=1/20){

  x = runif(n_max,a,b)

  thetahat = (b-a) * cumsum(f(x))/c(1:n_max)#seq(1,n_max,by=1)
  #dplyr::cummean()
  sdthetahat = (b-a)/sqrt(2:n_max) * cumsd_f(x)

  return(cbind(thetah = thetahat[-1], sd = sdthetahat))
}
```

```r
cumsd_AV = function(x,xprime){
  sapply(2:length(x), function(r)
    sd( (f(x[1:r])+f(xprime[1:r])) /2 ) )
}

AV_ergodic = function(n_max, a=0, b=1/20){

  x        = runif(n_max,a,b)
  xprime   = a+b-x
  h = (f(x) + f(xprime))/2
  thetahat = (b-a) * cumsum(h)/c(1:n_max)#seq(1,n_max,by=1)
  #dplyr::cummean()

  sdthetahat = (b-a)/sqrt(2:n_max) * cumsd_AV(x,xprime)

  # same as
  # sapply(2:n_max, function(q) sd(f(x[1:q])+f(xprime[1:q])))/2

  return(cbind(thetah = thetahat[-1], sd = sdthetahat))
}


a = integrate(f,0,1/20)
MC_res = MC_ergodic(500)
plot_ergodic(MC_res,a[1])


AV_res = AV_ergodic(n_max = 500)
A = AV_ergodic(n_max = 500)
lines(A[,1],type="l",lwd=2,col="orange")
lines(A[,1] + 1.96 * A[,2],
      type="l",col="red",lwd=2)
lines(A[,1] - 1.96 * A[,2],
      type="l",col="red",lwd=2)
```



We obtain a more efficient point and interval estimation with a smaller standard deviation and nearest estimation of the integral to the *real* value given by the `integrate` function of `R`

```
MC_res[499,]
```

```
##       thetah           sd
## 4.065196e-89 2.356068e-89
```

```
AV_res[499,]
```

```
##       thetah           sd
## 3.674989e-89 1.546452e-89
```

```
a[1]
```

```
## $value
## [1] 2.759158e-89
```

##Point d Finally, we can compare our results to the output of `R` function that should give the probability that we are looking for, `pnorm` but it gives the asymptotic result 0. Our method gives a more accurate result.

```
a[1]
```

```
## $value
## [1] 2.759158e-89
```

```
1-pnorm(20,0,1)
```

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