Sekhar_Mekala_HW4

Sekhar Mekala

Saturday, October 15, 2016

Problem 1

Given that

$$c(x) = \frac{1}{(2\pi)^{D/2}} e^{-0.5x^T.x}$$

where $x_i \sim U(-5, 5)$ for $i = 1...D$

Analytically, the $E[c(x)]$ can be computed as $\frac{1}{10^D}$, where D = number of dimensions.

Let us create an R function that accepts a matrix as input and computes the $c(x)$. The matrix is organized as D X N, where D is the number of rows in the matrix, and represents the number of dimensions, while N is the number of columns in the matrix, and represents samples in each dimension.

```
#In order to avoid conflict with c() function,
#we will write another function called c1()

c1 <- function(X)
  {
  #X = Matrix with random vectors
  D <- nrow(X)
  #N <- ncol(X)

  m <- (1/((2*pi)^(D/2)))*exp(-0.5*diag(t(X)%*%(X)))

  return(list(mean=mean(m),std_dev=sd(m)))
  }
```

Problem 1 - a. Crude Monte Carlo

The following R code will implement the crude Monte Carlo method to estimate the $E[c(x)]$

```
library(ggplot2)
library(knitr)

#Set the seed
set.seed(1234)

#Set the dimension to 1
D <- 1

avg <- vector(length=10)
std_dev <- vector(length=10)
coeff_of_var <- vector(length=10)
j <- 1
```

```r
for(i in seq(from=1000,to=10000,by=1000))
  {

  x <- replicate(100,c1(matrix(runif(i*D,min=-5,max=5),nrow=D,byrow=TRUE)))
  #c1(matrix(runif(i*D,min=-5,max=5),nrow=1,byrow=TRUE))
  avg[j] <- mean(unlist(x[1,]))
  std_dev[j] <- sd(unlist(x[1,]))
  coeff_of_var[j] <- std_dev[j]/avg[j]
  j <- j+1
  }

df <- data.frame(sample_size=as.factor(seq(from=1000,to=10000,by=1000)),
                 average=avg,std_dev=std_dev,coeff_of_var=coeff_of_var)

ggplot(df,aes(sample_size,average, label=round(average,6)))+
  geom_point(size=2,color="blue")+
  geom_hline(aes(yintercept=0.1),color="red",size=0.5)+
  labs(title="Fig 1: Avg. values for various sample sizes\n computed with 100 estimates,
       for each sample size and D=1",x="Sample size",y="Average value")+
  geom_text(hjust = -0.15, nudge_x = 0.055)+
  #geom_text(data = NULL,x=1000,y=0.09,label ="Analytical E[c(x)]")
  annotate("text",
           label = "Analytical E[c(x)]", x = 2, y = 0.09999, size = 1.5, colour = "red")


ggplot(df,aes(sample_size,std_dev, label=round(std_dev,6)))+
  geom_point(size=2,color="blue")+
  labs(title="Fig 2: Std. Dev of 100 estimates of average values\n
       for each sample size and D=1",x="Sample size",y="Std. Dev")+
  geom_text(hjust = 0, nudge_x = 0.05)


kable(df)
```

Fig 1: Avg. values for various sample sizes

computed with 100 estimates, for each sample size and D=1

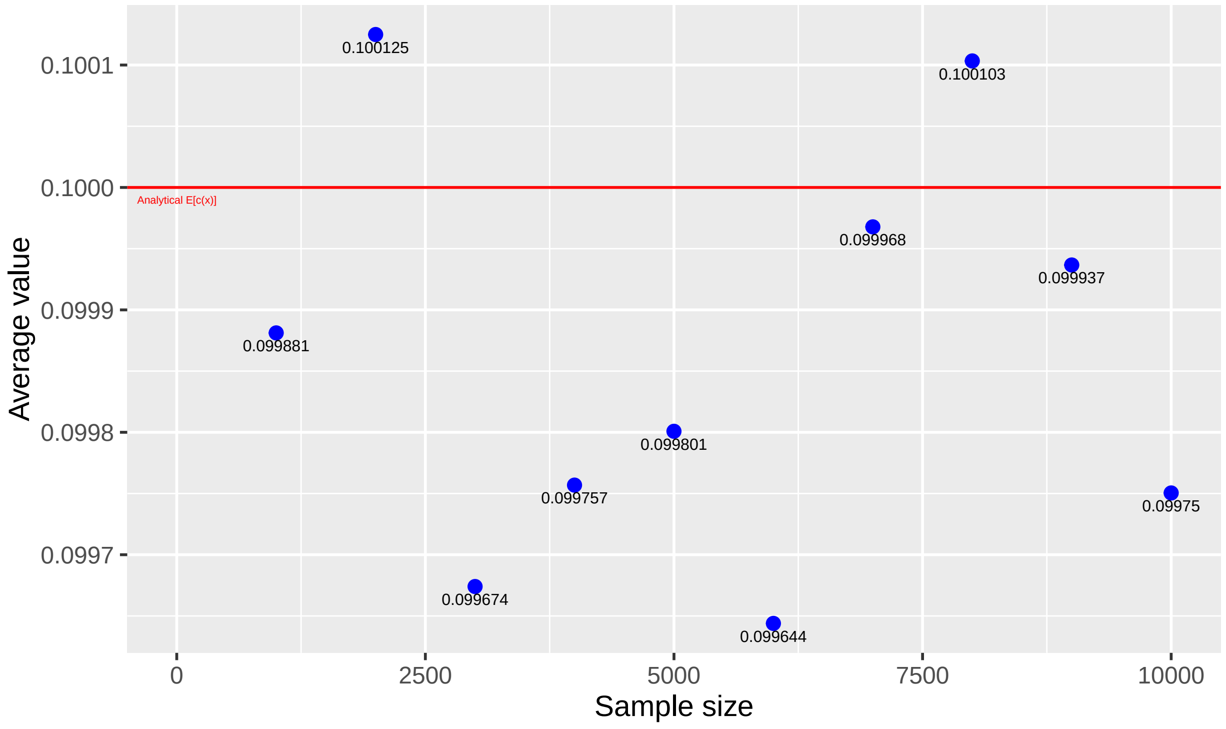
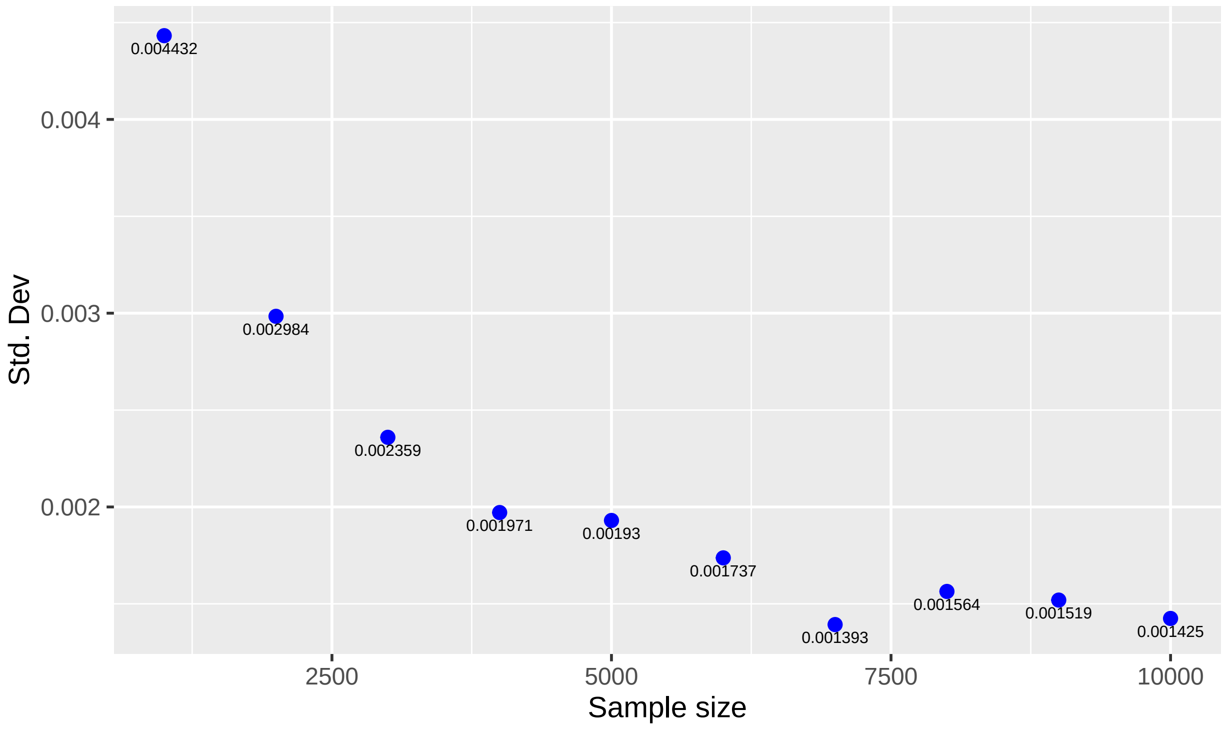


Fig 2: Std. Dev of 100 estimates of average values

for each sample size and D=1



sample_size	average	std_dev	coeff_of_var
1000	0.0998812	0.0044323	0.0443755
2000	0.1001249	0.0029839	0.0298017
3000	0.0996740	0.0023594	0.0236708
4000	0.0997568	0.0019711	0.0197595
5000	0.0998008	0.0019302	0.0193407
6000	0.0996439	0.0017373	0.0174347
7000	0.0999678	0.0013930	0.0139347
8000	0.1001033	0.0015639	0.0156233
9000	0.0999367	0.0015194	0.0152031
10000	0.0997504	0.0014246	0.0142813

From Fig-1, the estimated average value is almost 0.1 for a sample size of 7000. But usually, as the sample size increases the estimate should reach the analytical value. The value estimated for sample sizes of greater than 7000 is not as accurate as the value obtained with a sample size of 7000. This might be due to pure randomness, and is driven by the usage of the seed value (1234). The red line in figure-1 shows the analytical value (0.1).

From Fig-2, we can observe that as the sample size increases, the standard deviation of the estimate decreases.

Let us repeat the same simulation for 2 dimensional data.

```r
#Set the dimension to 2

set.seed(1234)

D <- 2

avg <- vector(length=10)
std_dev <- vector(length=10)
coeff_of_var <- vector(length=10)
j <- 1
for(i in seq(from=1000,to=10000,by=1000))
  {

  x <- replicate(100,c1(matrix(runif(i*D,min=-5,max=5),nrow=D,byrow=TRUE)))
  #c1(matrix(runif(i*D,min=-5,max=5),nrow=1,byrow=TRUE))
  avg[j] <- mean(unlist(x[1,]))
  std_dev[j] <- sd(unlist(x[1,]))
  coeff_of_var[j] <- std_dev[j]/avg[j]
  j <- j+1
  }

#library(ggplot2)
#library(knitr)
df_1 <- data.frame(
  sample_size=as.factor(seq(from=1000,to=10000,by=1000)),
  average=avg,std_dev=std_dev,
  coeff_of_var=coeff_of_var)

ggplot(df_1,aes(sample_size,average, label=round(average,6)))+
  geom_point(size=2,color="blue")+
  geom_hline(aes(yintercept=0.01),color="red",size=0.5)+
  labs(title="Fig 3: Avg. values for various sample sizes\n
       computed with 100 estimates, for each sample size, and D=2",
       x="Sample size",y="Average value")+
        geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)+
  annotate("text", label = "Analytical E[c(x)]", x = 5, y = 0.009998, size = 1.5, colour = "red")

ggplot(df_1,aes(sample_size,std_dev, label=round(std_dev,6)))+
  geom_point(size=2,color="blue")+
  labs(title="Fig 4: Std. Dev of 100 estimates of average values\n
       for each sample size, and D=2",x="Sample size",y="Std. Dev")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)


kable(df_1)
```

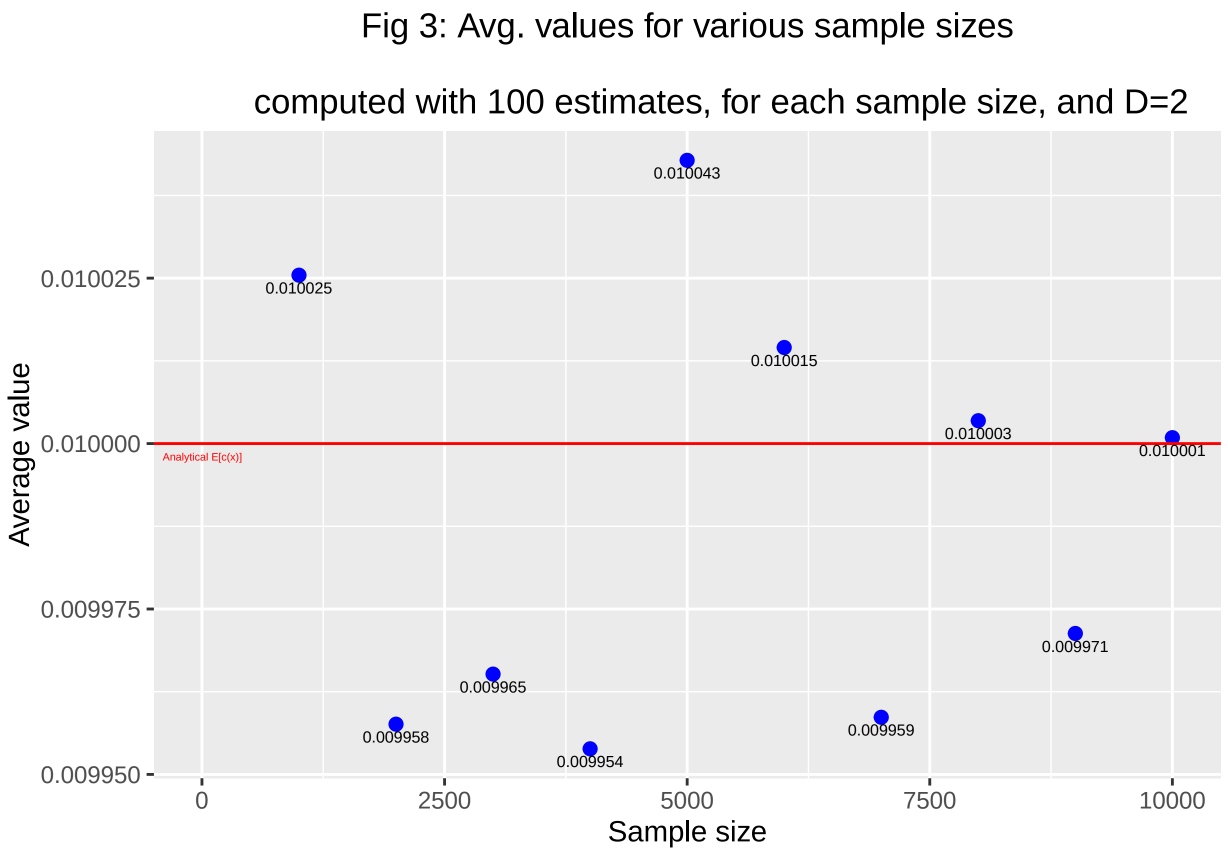
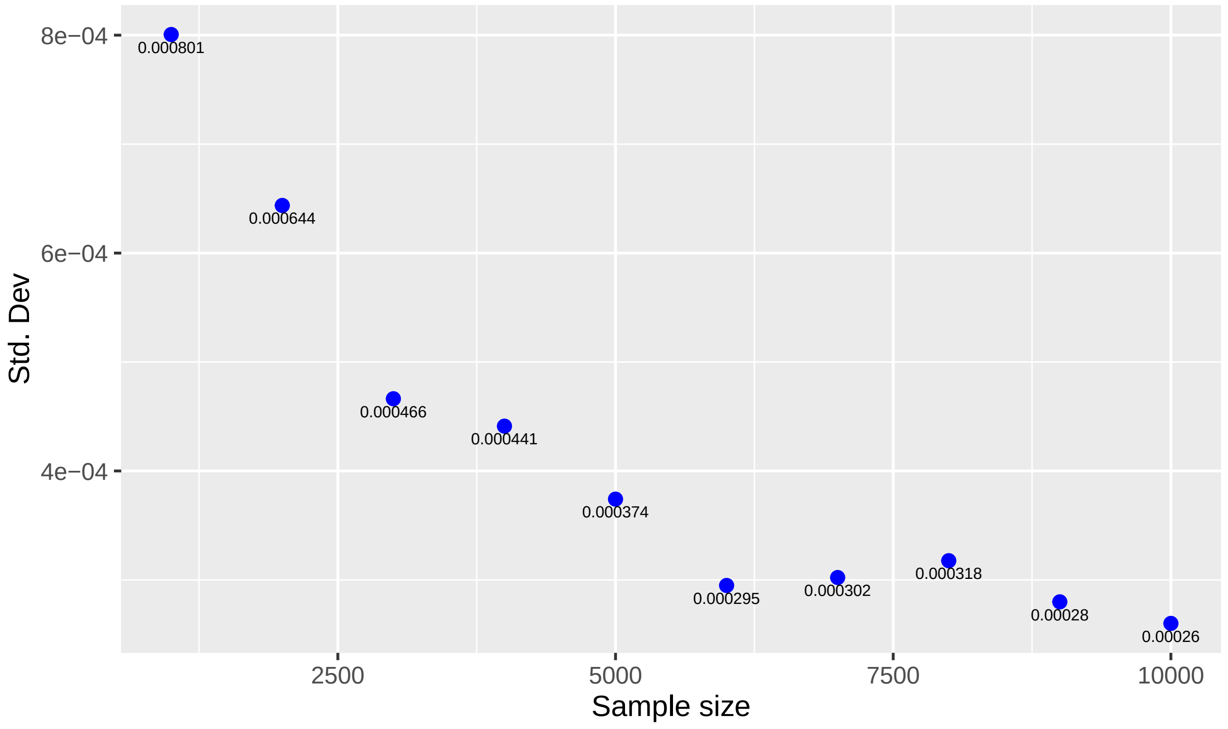


Fig 4: Std. Dev of 100 estimates of average values

for each sample size, and D=2



sample_size	average	std_dev	coeff_of_var
1000	0.0100254	0.0008006	0.0798580
2000	0.0099576	0.0006437	0.0646431
3000	0.0099652	0.0004663	0.0467906
4000	0.0099539	0.0004411	0.0443181
5000	0.0100428	0.0003741	0.0372537
6000	0.0100145	0.0002949	0.0294474
7000	0.0099586	0.0003022	0.0303441
8000	0.0100035	0.0003176	0.0317526
9000	0.0099713	0.0002799	0.0280663
10000	0.0100009	0.0002601	0.0260048

From Fig 3, we can observe that as the dimension increases, the accuracy of the estimates increased. At the sample size of 10000, the estimated value is almost equal to the analytical value. Fig 4 shows that the standard deviation is also minimum for a sample size of 10000. The std. deviation obtained for D=2 is less than the std. deviation obtained for D=1 simulation.

Problem 1 - b. Quasi-Random Numbers

We will use the GSL package to generate the quasi random numbers (Sobol variates). Let us generate 1000 standard normal numbers and sobol random numbers, and plot the obtained data:

```r
#Reference:
#http://www.theresearchkitchen.com/archives/700

library(gsl)

q <- qrng_alloc(type="sobol", 1)
rs <- qrng_get(q,1000)
#rs <- (rs * 10 - 5)
par(mfrow=c(2,1))

plot(rnorm(1000),main="Fig 5a: Random numbers from Std. Normal distribution",ylab="Value")

plot(rs, pch=20, main="Fig 5b: Sobol random numbers",
 ylab="Value", xlab="")
```

Fig 5a: Random numbers from Std. Normal distribution

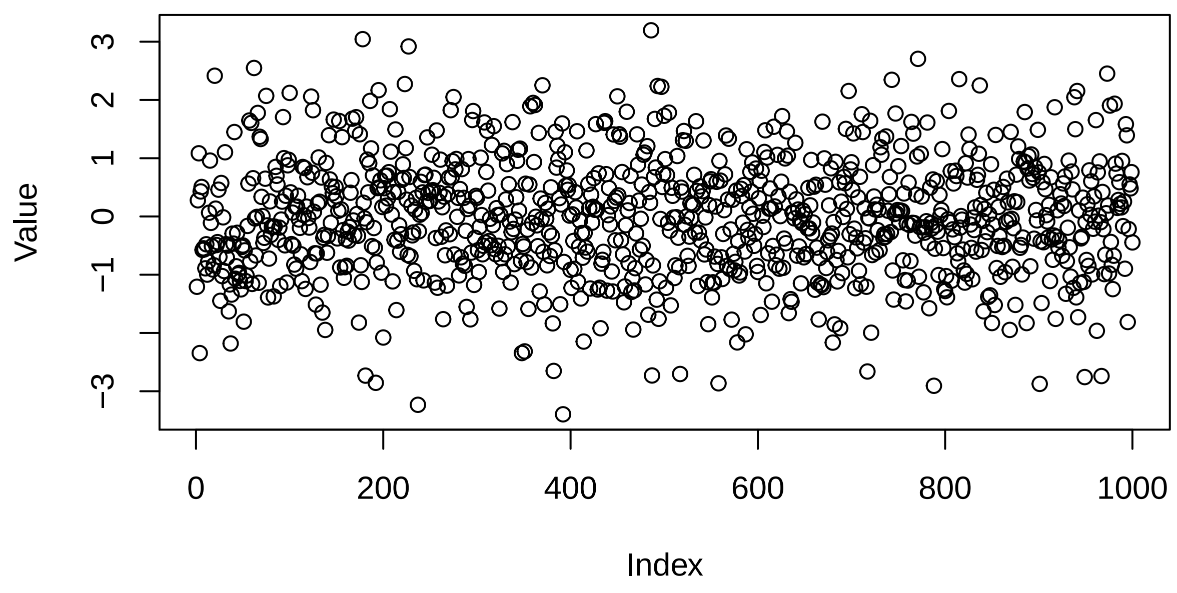
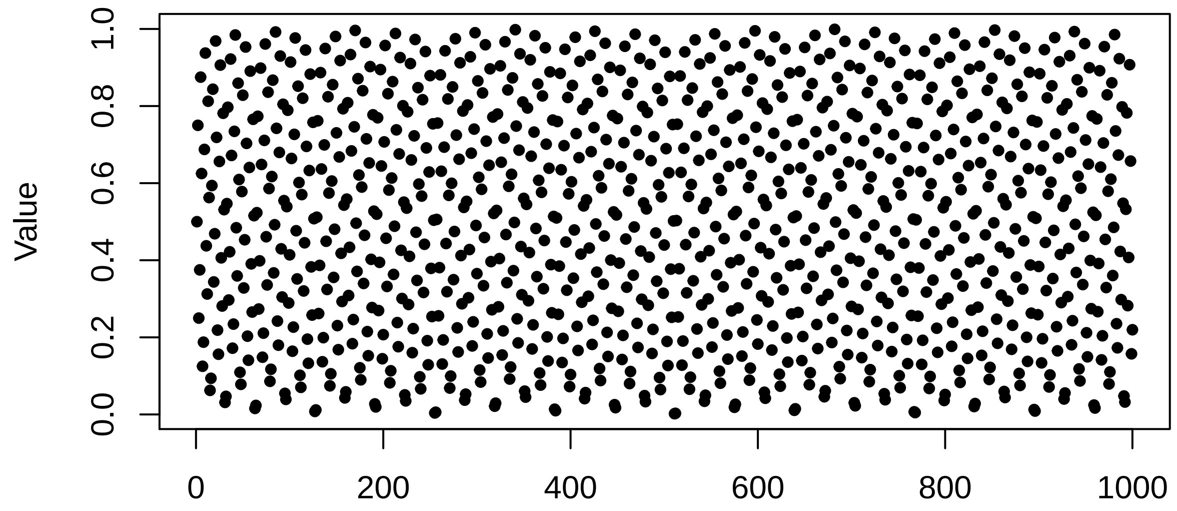


Fig 5b: Sobol random numbers



```
par(mfrow=c(1,1))
```

The above display clearly shows that the sobol random numbers have a specific pattern, while the random numbers from normal distribution have no pattern.

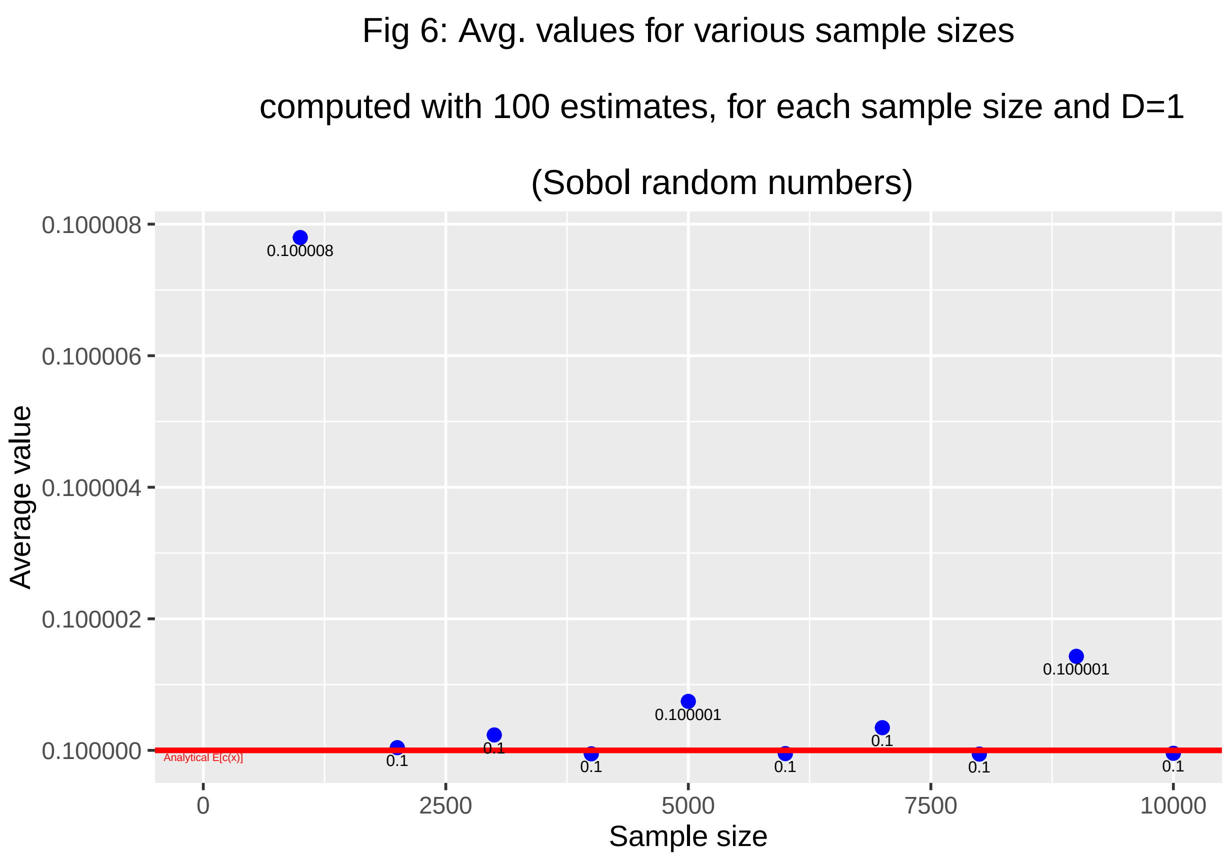
Now we will use the following transformation of sobol numbers to get the sobol random numbers in the range of [-5,5]. This transformation is obtained using the inverse transform method.

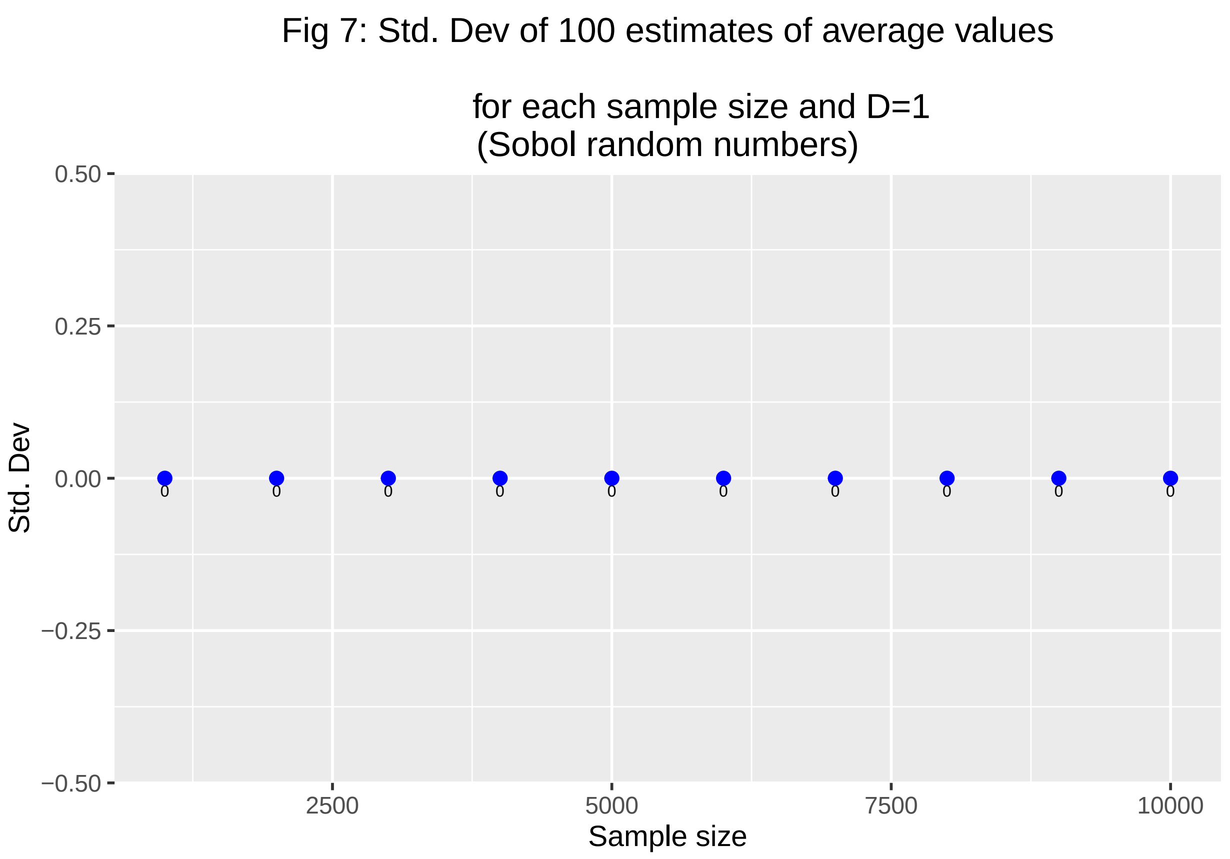
$$S = 10.R - 5$$

Where S = Sobol random number in the range of [-5,5], R = Sobol random number in the range of [0,1).

Let us repeat the same simulation we performed in "Problem 1 - Crude Monte Carlo", using sobol random numbers:

```r
set.seed(1234)

#Set the dimension to 1
D <- 1

avg <- vector(length=10)
std_dev <- vector(length=10)
coeff_of_var <- vector(length=10)
j <- 1

sobol_rand_fun <- function(D,i)
  {
    q <- qrng_alloc(type="sobol", D)
    rs <- qrng_get(q,i)
    s <- (rs * 10 - 5)
    return(s)
  }

for(i in seq(from=1000,to=10000,by=1000))
  {


  #x <- replicate(100,c1(t(sobol_rand_fun(D,i))))
  x <- replicate(100,expr={sb <- sobol_rand_fun(D,i)
                           c1(t(sb))})


  avg[j] <- mean(unlist(x[1,]))
  std_dev[j] <- sd(unlist(x[1,]))
  coeff_of_var[j] <- std_dev[j]/avg[j]
  j <- j+1

  }

sobol_df <- data.frame(sample_size=
                         as.factor(seq(from=1000,to=10000,by=1000)),
                       average=avg,std_dev=std_dev,
                       coeff_of_var=coeff_of_var)

ggplot(sobol_df,aes(sample_size,average, label=round(average,6)))+
  geom_point(size=2,color="blue")+
  geom_hline(aes(yintercept=0.1),color="red",size=1)+
  labs(title="Fig 6: Avg. values for various sample sizes\n
       computed with 100 estimates, for each sample size and D=1\n
       (Sobol random numbers)",
       x="Sample size",y="Average value")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)+
  annotate("text", label = "Analytical E[c(x)]", x = 1.5, y = 0.0999999, size = 1.5, colour = "red")

ggplot(sobol_df,aes(sample_size,std_dev, label=round(std_dev,6)))+
  geom_point(size=2,color="blue")+
  labs(title="Fig 7: Std. Dev of 100 estimates of average values\n for each sample size and D=1\n(Sobol
        geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)


kable(sobol_df)
```





sample_size	average	std_dev	coeff_of_var
1000	0.1000078	0	0
2000	0.1000000	0	0
3000	0.1000002	0	0
4000	0.0999999	0	0
5000	0.1000007	0	0
6000	0.1000000	0	0
7000	0.1000003	0	0
8000	0.0999999	0	0
9000	0.1000014	0	0
10000	0.1000000	0	0

We can observe that irrespective of the sample size, the estimated value is almost equal to the analytical value of 0.1. The standard deviation is also zero. Hence sobol random numbers have most accurately estimated the average value, with almost 0 standard deviation.

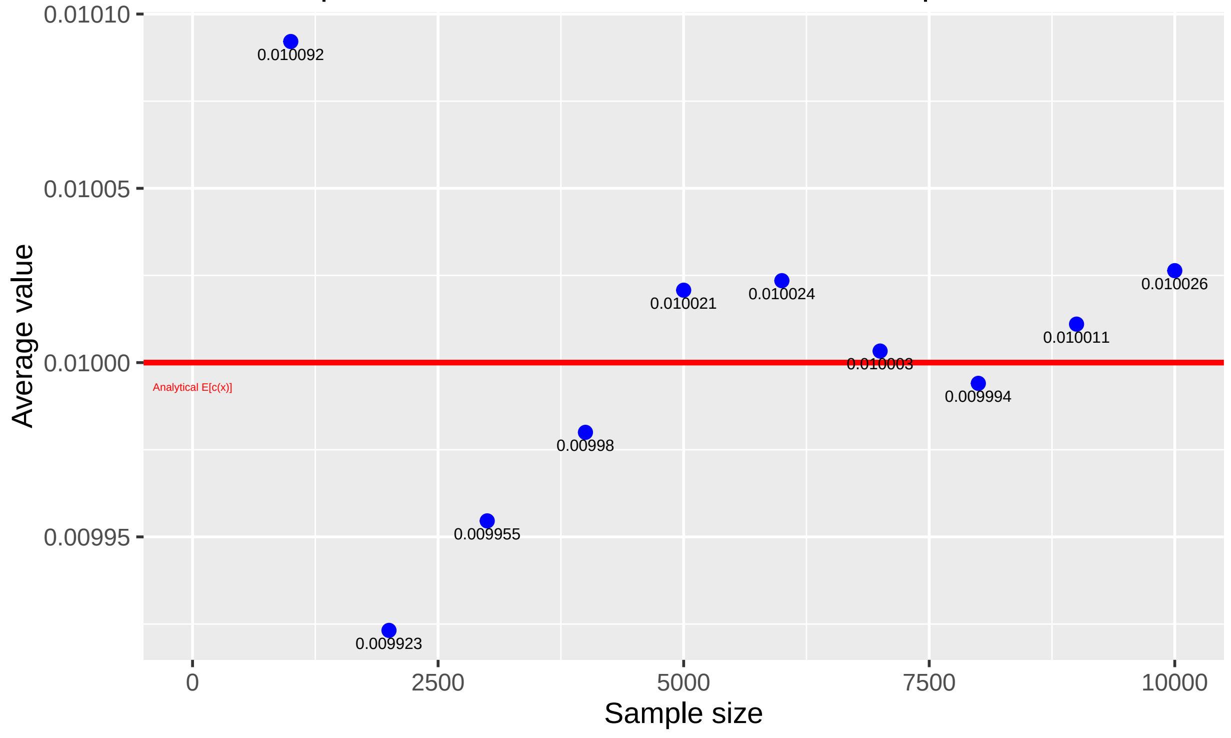
Let us repeat the same simulation for 2 dimensional data.

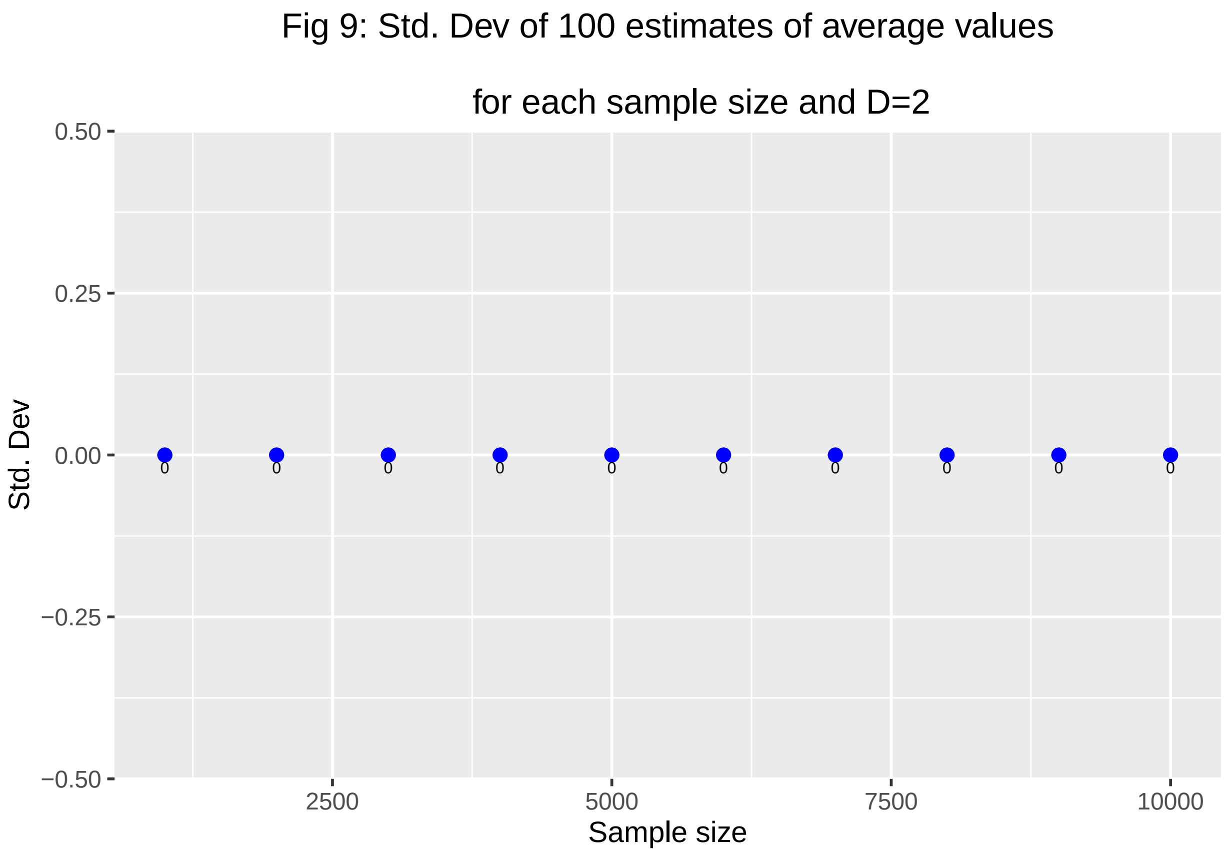
```
#Set the seed

set.seed(1234)

#Set the dimension to 2
D <- 2
#temp_avg <- vector(length=100)
```

```
#temp_sd <- vector(length=100)
avg <- vector(length=10)
std_dev <- vector(length=10)
coeff_of_var <- vector(length=10)
j <- 1

for(i in seq(from=1000,to=10000,by=1000))
  {

  x <- replicate(100,expr={sb <- sobol_rand_fun(D,i)
                           c1(t(sb))})

  avg[j] <- mean(unlist(x[1,]))
  std_dev[j] <- sd(unlist(x[1,]))


  coeff_of_var[j] <- std_dev[j]/avg[j]
  j <- j+1
  }

sobol_df_1 <- data.frame(sample_size=as.factor(seq(from=1000,to=10000,by=1000)),
                         average=avg,std_dev=std_dev,coeff_of_var=coeff_of_var)

ggplot(sobol_df_1,aes(sample_size,average, label=round(average,6)))+
  geom_point(size=2,color="blue")+
  geom_hline(aes(yintercept=0.01),color="red",size=1)+
  labs(title="Fig 8: Avg. values for various sample sizes\n
       computed with 100 estimates, for each sample size and D=2",x="Sample size",
       y="Average value")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)+
  annotate("text", label = "Analytical E[c(x)]", x = 1.5, y = 0.009993,
           size = 1.5, colour = "red")

ggplot(sobol_df_1,aes(sample_size,std_dev, label=round(std_dev,6)))+
  geom_point(size=2,color="blue")+
  labs(title="Fig 9: Std. Dev of 100 estimates of average values\n
       for each sample size and D=2",x="Sample size",y="Std. Dev")+
        geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)



kable(sobol_df_1)
```

Fig 8: Avg. values for various sample sizes

computed with 100 estimates, for each sample size and D=2





sample_size	average	std_dev	coeff_of_var
1000	0.0100921	0	0
2000	0.0099232	0	0
3000	0.0099546	0	0
4000	0.0099800	0	0
5000	0.0100208	0	0
6000	0.0100235	0	0
7000	0.0100033	0	0
8000	0.0099940	0	0
9000	0.0100110	0	0
10000	0.0100264	0	0

For D=2 simulation also, the estimated average value is almost equal to the analytical value of 0.01. The standard deviation is also 0.

So we can conclude that sobol random numbers give almost zero variance for the estimated value for $E[c(x)]$

Problem 1 - c. Antithetic variates

Let us repeat the same simulation we performed in "Problem 1 - Crude Monte Carlo", using antithetic random numbers. We will create a function "antithetic_rand_fun" to generate a sample of random numbers in (-5,5) interval. This function will take 2 inputs: Sample size and the required dimension for the sample. Only half of the required random numbers are generated using runif() function, and the remaining are obtained by subtracting the generated random numbers from 1, and transforming the obtained values to the scale (-5,5) using inverse transform method.:

```r
set.seed(1234)

#Set the dimension to 1
D <- 1

avg <- vector(length=10)
std_dev <- vector(length=10)
coeff_of_var <- vector(length=10)

j <- 1

antithetic_rand_fun <- function(i,D)
  {

  m <- matrix(runif((i/2)*D),nrow=D,byrow=TRUE)

  r <- cbind(m,1-m)

  #Use inverse transform method
  r <- 10*r-5

  }

for(i in seq(from=1000,to=10000,by=1000))
  {


  #x <- replicate(100,c1(antithetic_rand_fun(i,D)))
  x <- replicate(100,expr={
                            at <- antithetic_rand_fun(i,D)
                            c1(at)
                          })

  avg[j] <- mean(unlist(x[1,]))
  std_dev[j] <- sd(unlist(x[1,]))
  coeff_of_var[j] <- std_dev[j]/avg[j]
  j <- j+1
  }

antithetic_df <- data.frame(sample_size=as.factor(seq(from=1000,to=10000,by=1000)),
                            average=avg,std_dev=std_dev,
                            coeff_of_var=coeff_of_var)

ggplot(antithetic_df,aes(sample_size,average,
                         label=round(average,6)))+
  geom_point(size=2,color="blue")+
  geom_hline(aes(yintercept=0.1),color="red",size=0.5)+
  labs(title="Fig 9: Avg. values for various sample sizes\n
       computed with 100 estimates, for each sample size and D=1\n
       (Antithetic random numbers)",
       x="Sample size",
       y="Average value")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)
```

```
ggplot(antithetic_df,aes(sample_size,std_dev, label=round(std_dev,6)))+
  geom_point(size=2,color="blue")+
  labs(title="Fig 10: Std. Dev of 100 estimates of average values\n
       for each sample size and D=1\n
       (Antithetic random numbers)",
       x="Sample size",
       y="Std. Dev")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)


kable(antithetic_df)
```

Fig 9: Avg. values for various sample sizes
computed with 100 estimates, for each sample size and D=1
(Antithetic random numbers)

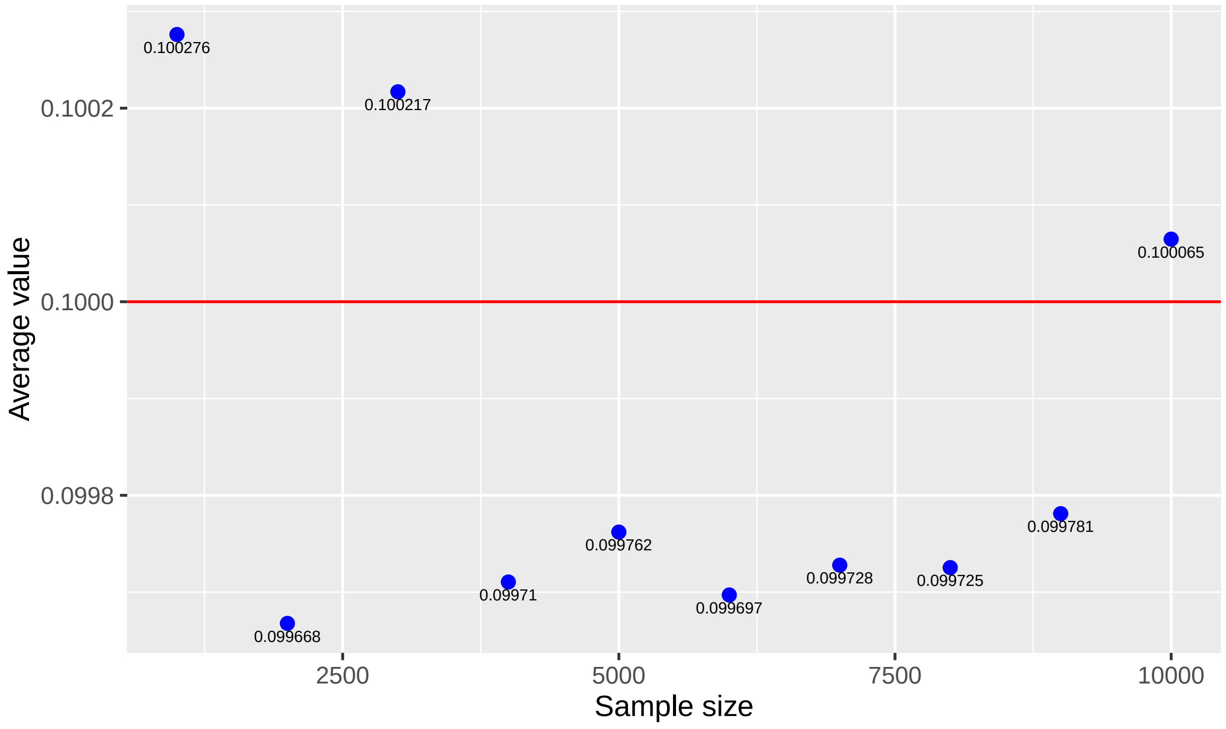
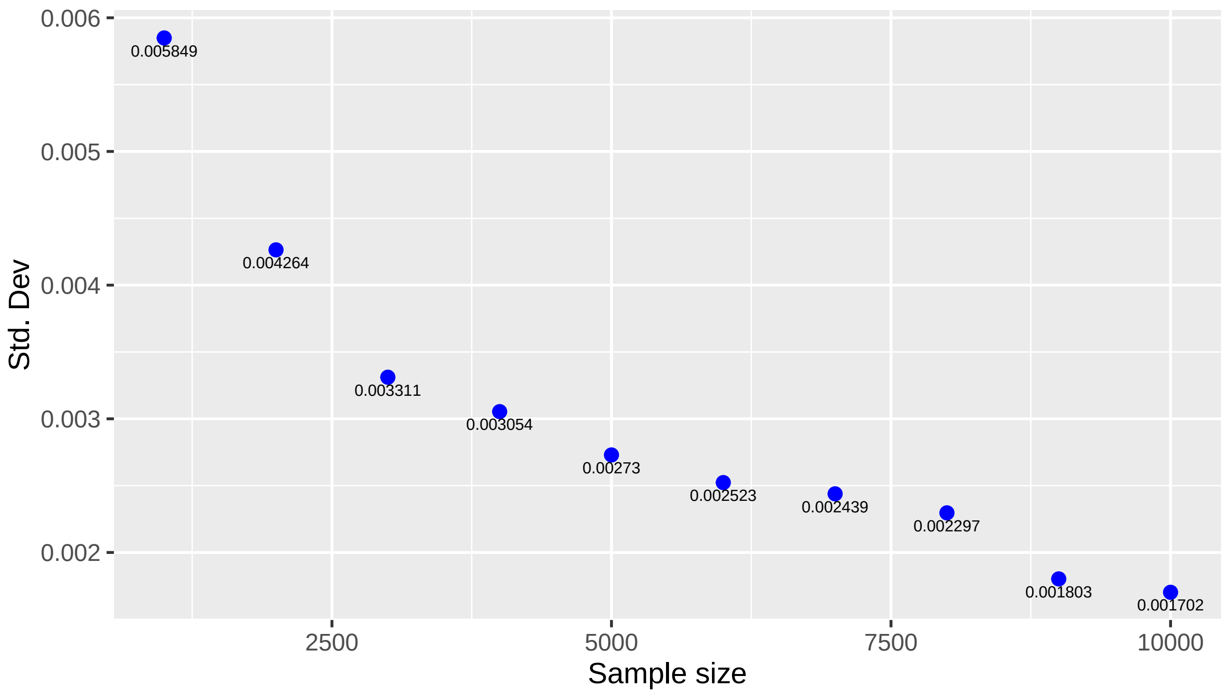


Fig 10: Std. Dev of 100 estimates of average values

for each sample size and D=1
(Antithetic random numbers)

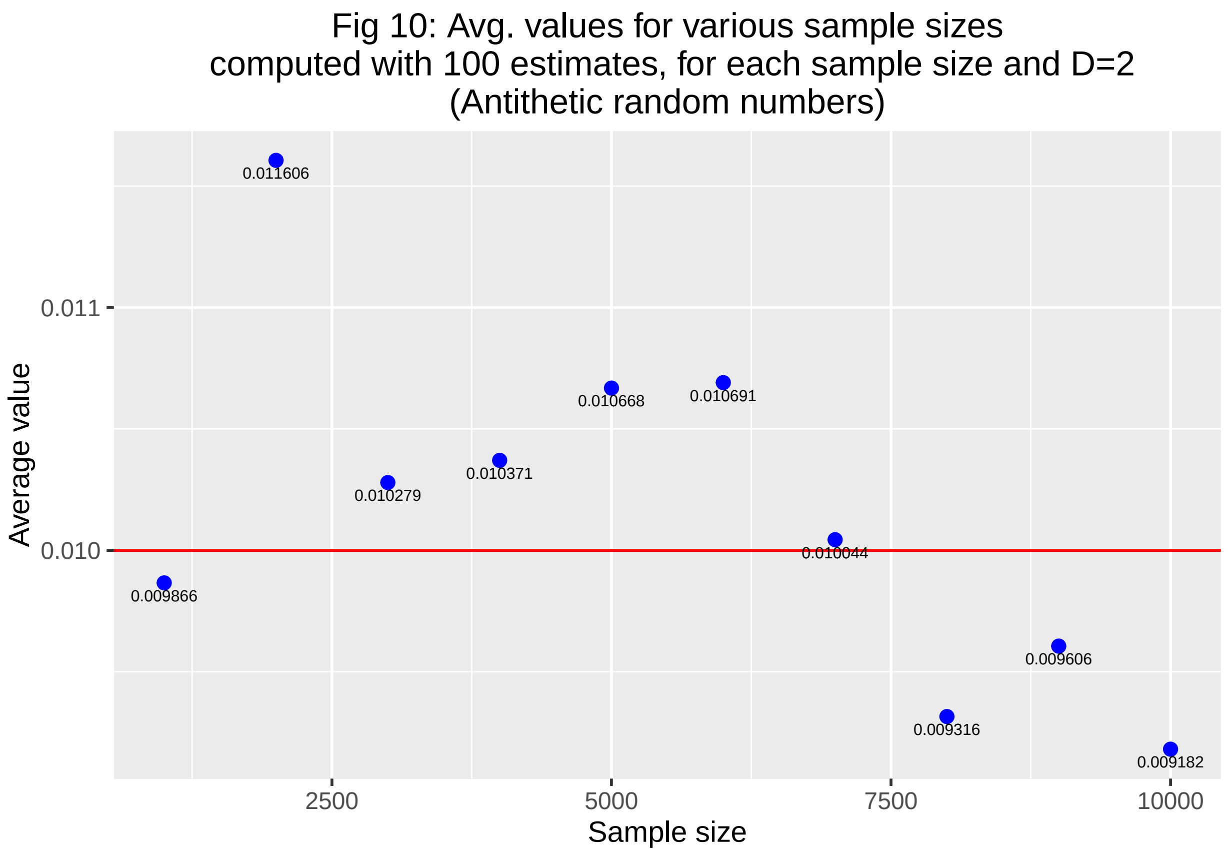


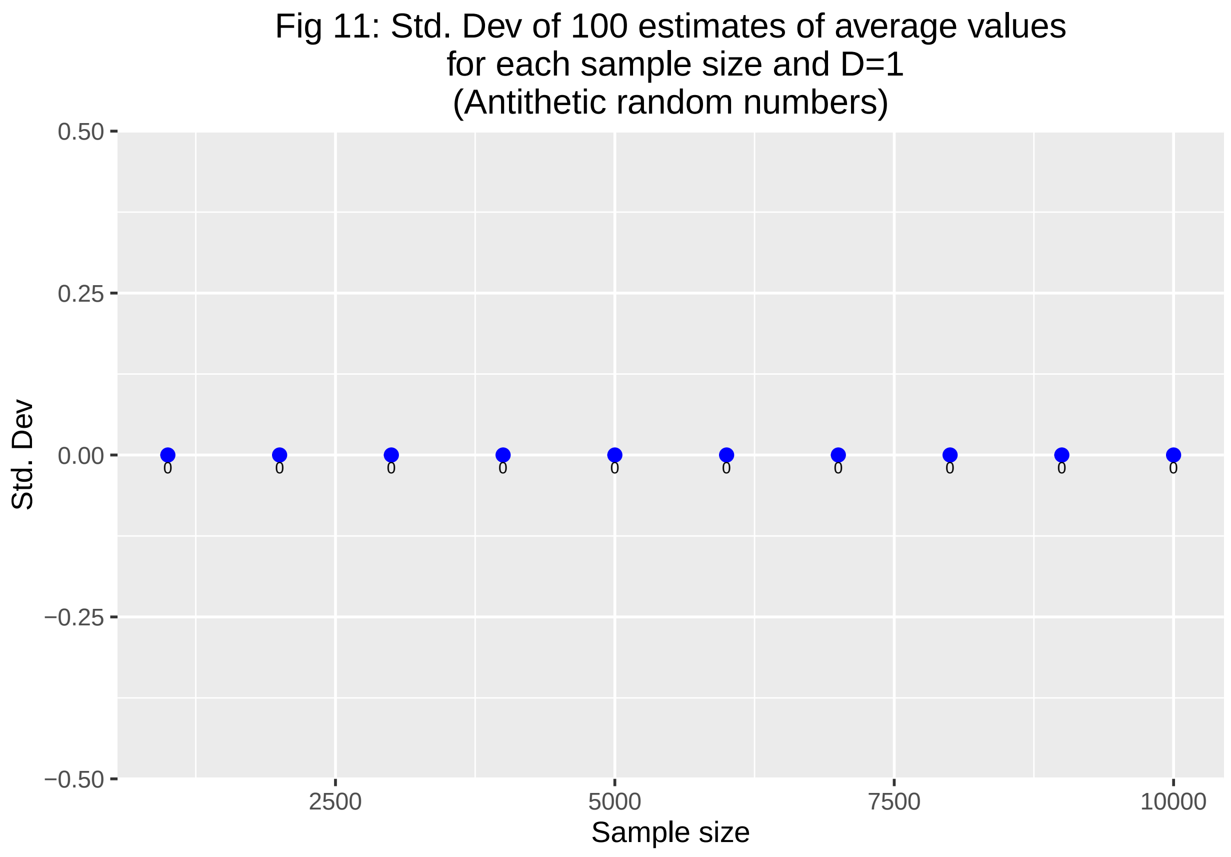
sample_size	average	std_dev	coeff_of_var
1000	0.1002761	0.0058490	0.0583293
2000	0.0996676	0.0042639	0.0427816
3000	0.1002169	0.0033111	0.0330390
4000	0.0997104	0.0030538	0.0306271
5000	0.0997620	0.0027297	0.0273617
6000	0.0996970	0.0025229	0.0253058
7000	0.0997278	0.0024389	0.0244554
8000	0.0997253	0.0022966	0.0230290
9000	0.0997810	0.0018025	0.0180647
10000	0.1000646	0.0017023	0.0170120

We can observe that as the sample size increases, the estimated average value is almost equal to the analytical value of 0.1. The standard deviation is also approaching zero as the sample size increases.

Let us repeat the same simulation for 2 dimensional data.

```r
set.seed(1234)

#Set the dimension to 2
D <- 2

avg <- vector(length=10)
std_dev <- vector(length=10)
coeff_of_var <- vector(length=10)

j <- 1

for(i in seq(from=1000,to=10000,by=1000))
  {

  m <- matrix(runif((i/2)*D),nrow=D,byrow=TRUE)
  #matrix(runif(i*D,min=-5,max=5),nrow=D,byrow=TRUE)
  r <- cbind(m,1-m)

  r <- 10*r-5

  x <- replicate(100,c1(r))
    avg[j] <- mean(unlist(x[1,]))
  std_dev[j] <- sd(unlist(x[1,]))
  coeff_of_var[j] <- std_dev[j]/avg[j]
  j <- j+1
  }

antithetic_df_1 <- data.frame(sample_size=as.factor
                              (seq(from=1000,to=10000,by=1000)),
                              average=avg,std_dev=std_dev,
                              coeff_of_var=coeff_of_var)

ggplot(antithetic_df_1,aes(sample_size,average, label=round(average,6)))+
  geom_point(size=2,color="blue")+
  geom_hline(aes(yintercept=0.01),color="red",size=0.5)+
  labs(title="Fig 10: Avg. values for various sample sizes\n
       computed with 100 estimates, for each sample size and D=2\n
       (Antithetic random numbers)",
       x="Sample size",
       y="Average value")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)

ggplot(antithetic_df_1,aes(sample_size,std_dev, label=round(std_dev,6)))+
  geom_point(size=2,color="blue")+
  labs(title="Fig 11: Std. Dev of 100 estimates of average values\n
       for each sample size and D=1\n
       (Antithetic random numbers)",
       x="Sample size",y="Std. Dev")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)


kable(antithetic_df_1)

write.csv(antithetic_df_1,"antithetic_df_1.csv")
```





sample_size	average	std_dev	coeff_of_var
1000	0.0098662	0	0
2000	0.0116059	0	0
3000	0.0102793	0	0
4000	0.0103708	0	0
5000	0.0106683	0	0
6000	0.0106909	0	0
7000	0.0100440	0	0
8000	0.0093160	0	0
9000	0.0096059	0	0
10000	0.0091816	0	0

For D=2 also, as the sample size increases, the estimated value approaches the analytical value of 0.01. But after the sample size of 7000, the estimated values deviated slightly from the analytical value. This could be because of pure randomness or due to the usage of a seed value. The standard deviation is almost zero irrespective of the sample sizes. The std. deviation using the antithetic numbers is almost 0, this is due to the fact of using the correlated values.

Problem 1 - d. Latin Hypercube sampling

Let us repeat the same simulation we performed in "Problem 1 - Crude Monte Carlo", using Latin Hypercube sampling. In this method, we will divide the sample intervals into 4 intervals, as given below:

$$Interval - 1 : [-5, -2.5)$$

$$Interval - 2 : [-2.5, 0)$$
$$Interval - 3 : [0, 2.5)$$
$$Interval - 4 : [2.5, 5]$$

Getting estimates of E[c(x)], for D=1

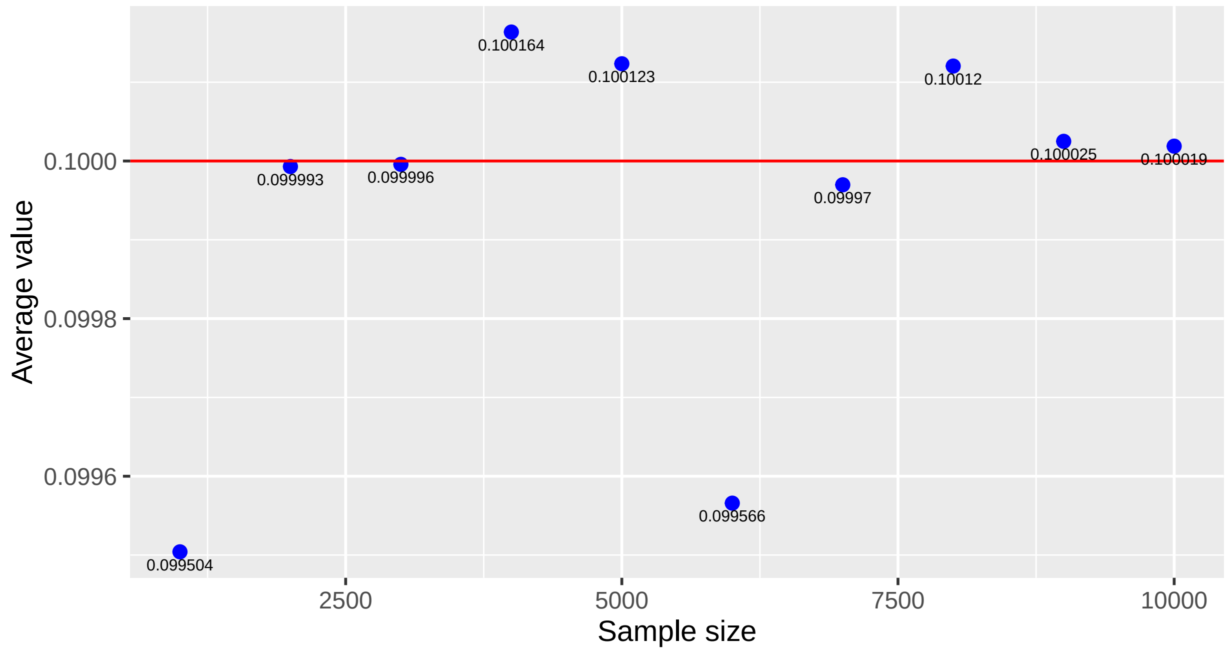
```
set.seed(1234)

#Set the dimension to 1
D <- 1

avg <- vector(length=10)
std_dev <- vector(length=10)
coeff_of_var <- vector(length=10)
a <- vector(length=4)
j <- 1
temp_avg <- vector(length=100)

for(i in seq(from=1000,to=10000,by=1000))
  {
      for(k in 1:100)
        {

           x <- c1(matrix(runif(min=-5,max=-2.5,i/4),nrow=D,byrow=TRUE))

          a[1] <- unlist(x[1])

          x <- c1(matrix(runif(min=-2.5,max=0,i/4),nrow=D,byrow=TRUE))

          a[2] <- unlist(x[1])

          x <- c1(matrix(runif(min=0,max=2.5,i/4),nrow=D,byrow=TRUE))

          a[3] <- unlist(x[1])

          x <- c1(matrix(runif(min=2.5,max=5,i/4),nrow=D,byrow=TRUE))

          a[4] <- unlist(x[1])

          temp_avg[k] <- sum(a)/(4*D)

          }
      avg[j] <- mean(temp_avg)
      std_dev[j] <- sd(temp_avg)
      coeff_of_var[j] <- std_dev[j]/avg[j]
      j <- j+1
  }


stratified_df <- data.frame(sample_size=as.factor(
  seq(from=1000,to=10000,by=1000)),
  average=avg,std_dev=std_dev,
  coeff_of_var=coeff_of_var)
```

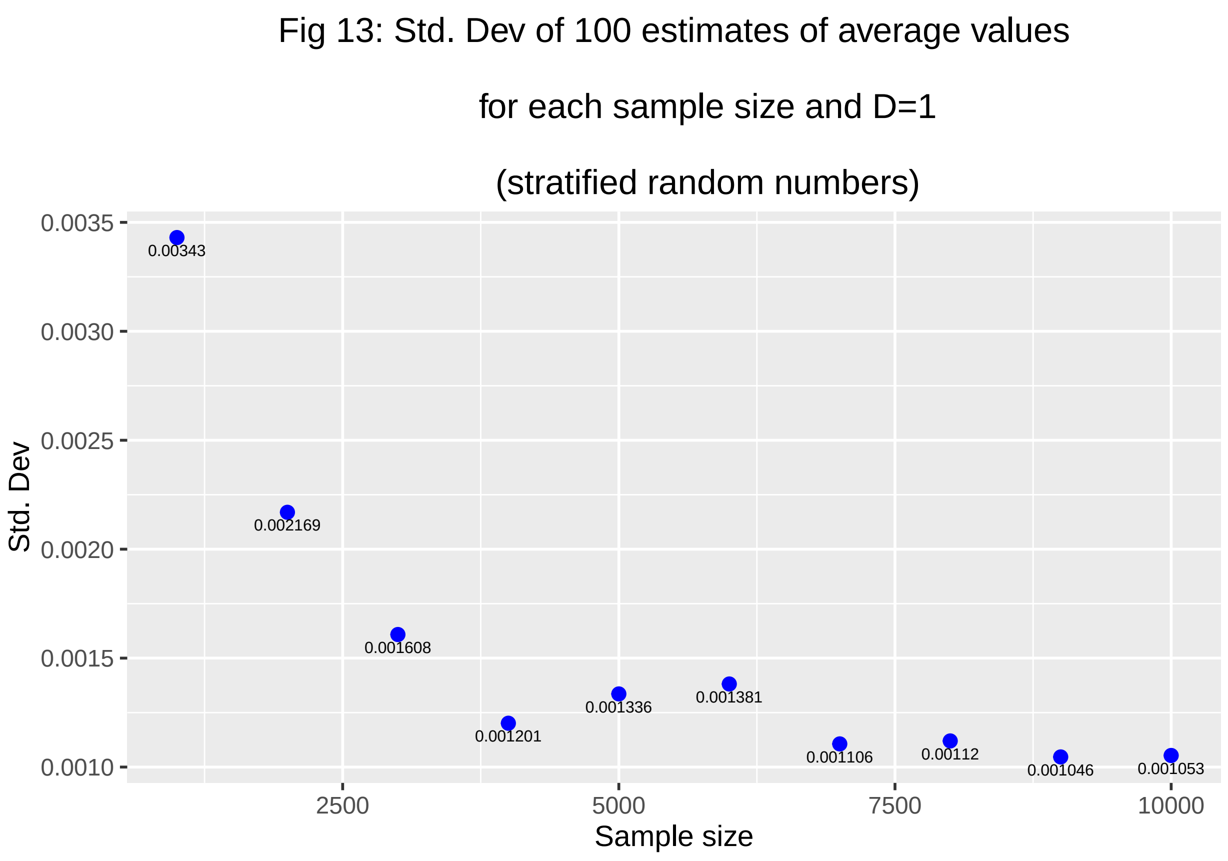
```r
ggplot(stratified_df,aes(sample_size,average, label=round(average,6)))+
  geom_point(size=2,color="blue")+
  geom_hline(aes(yintercept=0.1),color="red",size=.5)+
  labs(title="Fig 12: Avg. values for various sample sizes\n
       computed with 100 estimates, for each sample size and D=1\n
       (Stratified random numbers)",
       x="Sample size",
       y="Average value")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)

ggplot(stratified_df,aes(sample_size,std_dev, label=round(std_dev,6)))+
  geom_point(size=2,color="blue")+
  labs(title="Fig 13: Std. Dev of 100 estimates of average values\n
       for each sample size and D=1\n
       (stratified random numbers)",
       x="Sample size",y="Std. Dev")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)


kable(stratified_df)
```

Fig 12: Avg. values for various sample sizes

computed with 100 estimates, for each sample size and D=1

(Stratified random numbers)





sample_size	average	std_dev	coeff_of_var
1000	0.0995041	0.0034302	0.0344732
2000	0.0999930	0.0021693	0.0216946
3000	0.0999958	0.0016084	0.0160843
4000	0.1001636	0.0012011	0.0119917
5000	0.1001234	0.0013357	0.0133407
6000	0.0995658	0.0013813	0.0138731
7000	0.0999698	0.0011063	0.0110664
8000	0.1001205	0.0011199	0.0111860
9000	0.1000250	0.0010465	0.0104623
10000	0.1000188	0.0010532	0.0105296

The estimated average is almost equal to 0.1, and the accuracy of the estimate is maximum at 10000 sample size. The standard deviation of the estimation also decreases with the sample size. Also the stratified sample method is a bit faster than the other methods discussed above (since I obtained the output pretty fast with this method).

Let us repeat the same simulation with stratified sample method for D=2.

```r
set.seed(1234)

#Set the dimension to 2
D <- 2

avg <- vector(length=10)
```

```r
std_dev <- vector(length=10)
coeff_of_var <- vector(length=10)
a <- vector(length=4)
j <- 1
temp_avg <- vector(length=100)

for(i in seq(from=1000,to=10000,by=1000))
  {
      for(k in 1:100)
        {

           x <- c1(matrix(runif(i*D/4,min=-5,max=-2.5),nrow=D,byrow=TRUE))

          a[1] <- unlist(x[1])

          x <- c1(matrix(runif(i*D/4,min=-2.5,max=0),nrow=D,byrow=TRUE))

          a[2] <- unlist(x[1])

          x <- c1(matrix(runif(i*D/4,min=0,max=2.5),nrow=D,byrow=TRUE))

          a[3] <- unlist(x[1])

          x <- c1(matrix(runif(i*D/4,min=2.5,max=5),nrow=D,byrow=TRUE))

          a[4] <- unlist(x[1])

          temp_avg[k] <- sum(a)/(4*D)

          }
      avg[j] <- mean(temp_avg)
      std_dev[j] <- sd(temp_avg)
      coeff_of_var[j] <- std_dev[j]/avg[j]
      j <- j+1
  }


stratified_df_1 <- data.frame(sample_size=as.factor(
  seq(from=1000,to=10000,by=1000)),
  average=avg,std_dev=std_dev,
  coeff_of_var=coeff_of_var)

ggplot(stratified_df_1,aes(sample_size,average, label=round(average,6)))+
  geom_point(size=2,color="blue")+
  geom_hline(aes(yintercept=0.01),color="red",size=.5)+
  labs(title="Fig 14: Avg. values for various sample sizes\n
       computed with 100 estimates, for each sample size and D=2\n
       (Stratified random numbers)",
       x="Sample size",y="Average value")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)

ggplot(stratified_df_1,aes(sample_size,std_dev, label=round(std_dev,6)))+
  geom_point(size=2,color="blue")+
  labs(title="Fig 15: Std. Dev of 100 estimates of average values\n
       for each sample size and D=1\n
       (stratified random numbers)",
       x="Sample size",y="Std. Dev")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)


kable(stratified_df_1)
```

Fig 14: Avg. values for various sample sizes

computed with 100 estimates, for each sample size and D=2

(Stratified random numbers)

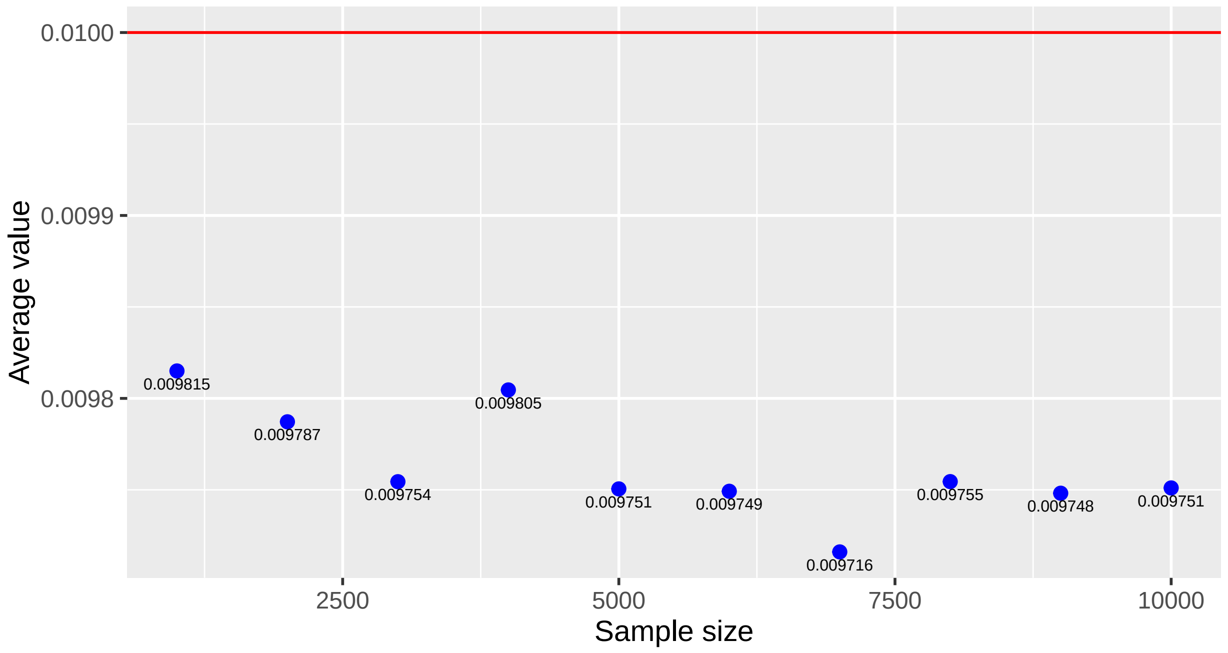
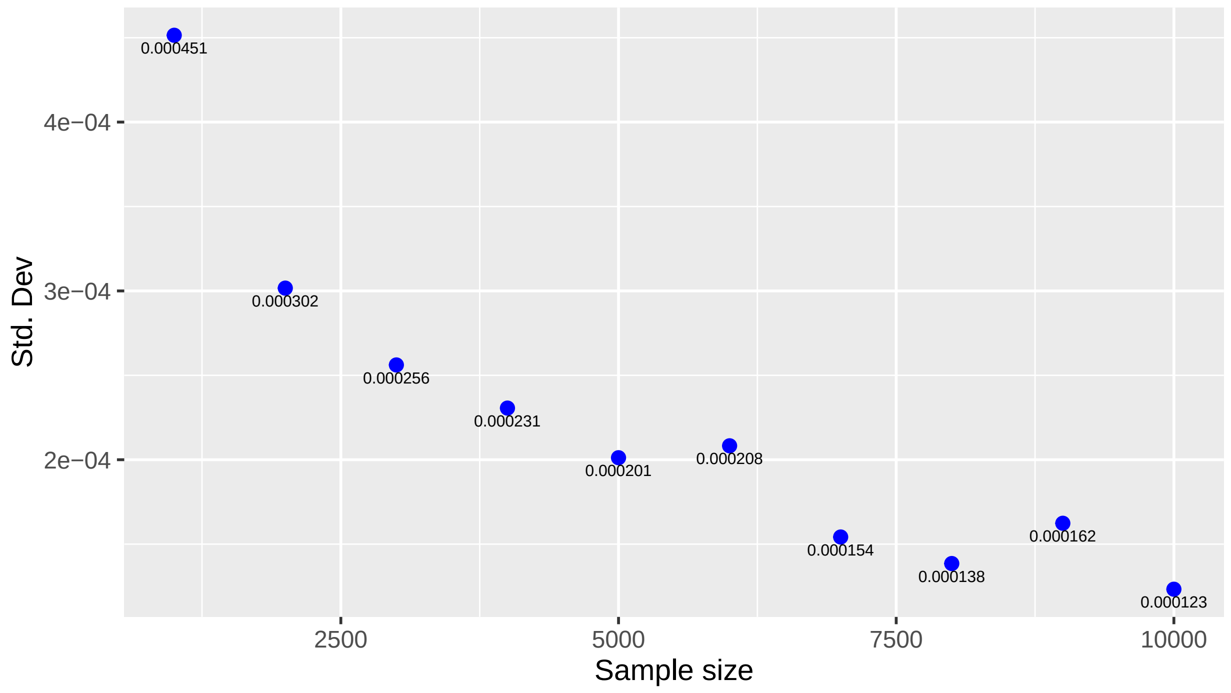


Fig 15: Std. Dev of 100 estimates of average values

for each sample size and D=1
(stratified random numbers)



sample_size	average	std_dev	coeff_of_var
1000	0.0098150	0.0004514	0.0459946
2000	0.0097872	0.0003016	0.0308207
3000	0.0097545	0.0002562	0.0262599
4000	0.0098046	0.0002306	0.0235174
5000	0.0097505	0.0002012	0.0206338
6000	0.0097492	0.0002083	0.0213612
7000	0.0097161	0.0001542	0.0158725
8000	0.0097545	0.0001385	0.0141984
9000	0.0097482	0.0001624	0.0166548
10000	0.0097511	0.0001233	0.0126439

The accuracy of the estimate is not good for D=2 (using stratified sampling). But the std. deviation of the estimate decreases as the sample size increases.

e. Importance sampling

The optimal importance sampling density function, $q(x)$ is obtained as:

$$q(x) = c(x).p(x)/E[c(x)]$$

In general this optimal function is difficult to obtain, since we do not know the unknown E[c(x)]. But in our problem, we know that

$$E[c(x)] = (1/10)^D$$

Also the $p(x)$ is a uniform density function between the interval [-5,5]. Therefore the probability density function, $p(x)$ is defined as 1/10. If we have D=2, then we have two independent samples from uniform distribution, and their joint probability density function will be $(1/10)^2$. Hence the joint probability density function for D dimensions would be $(1/10)^D$.

Therefore,

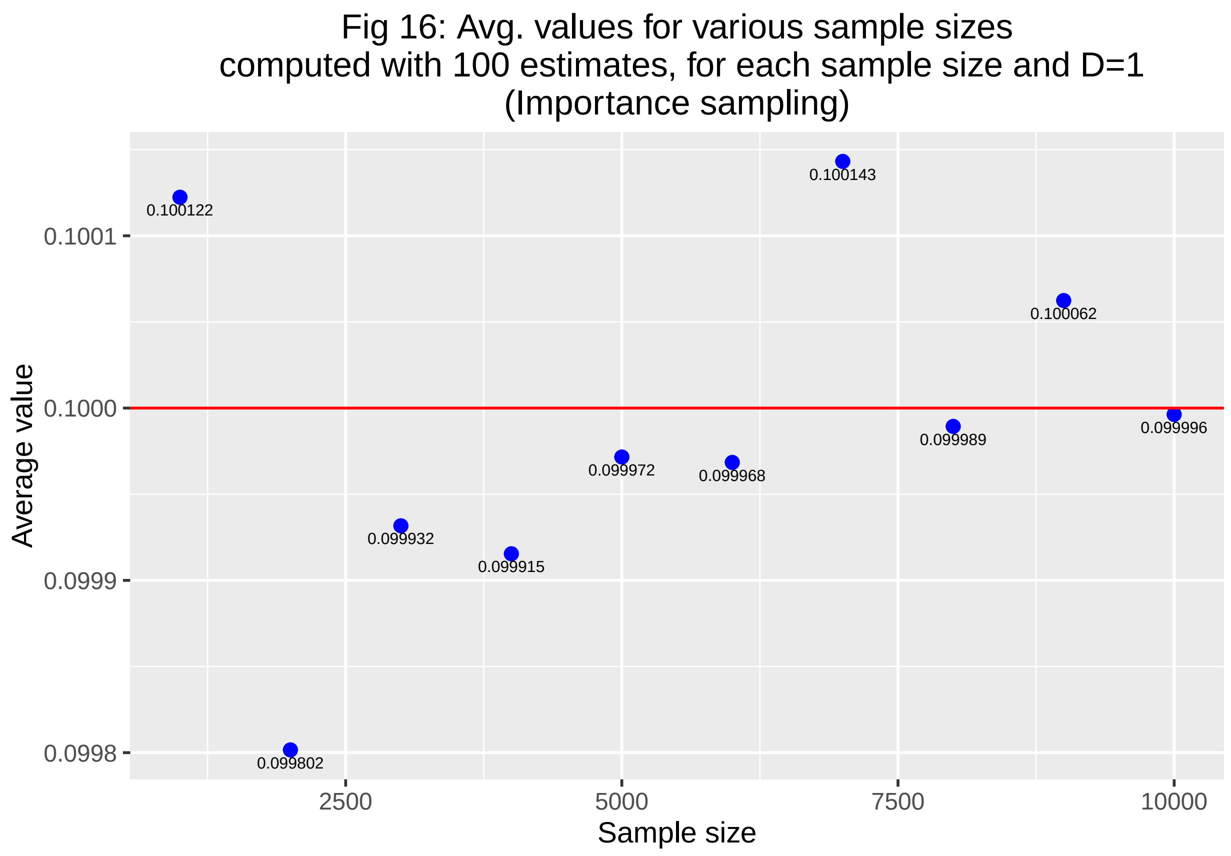
$$q(x) = c(x).p(x)/E[c(x)] = c(x).(1/10)^D/(1/10)^D = c(x)$$

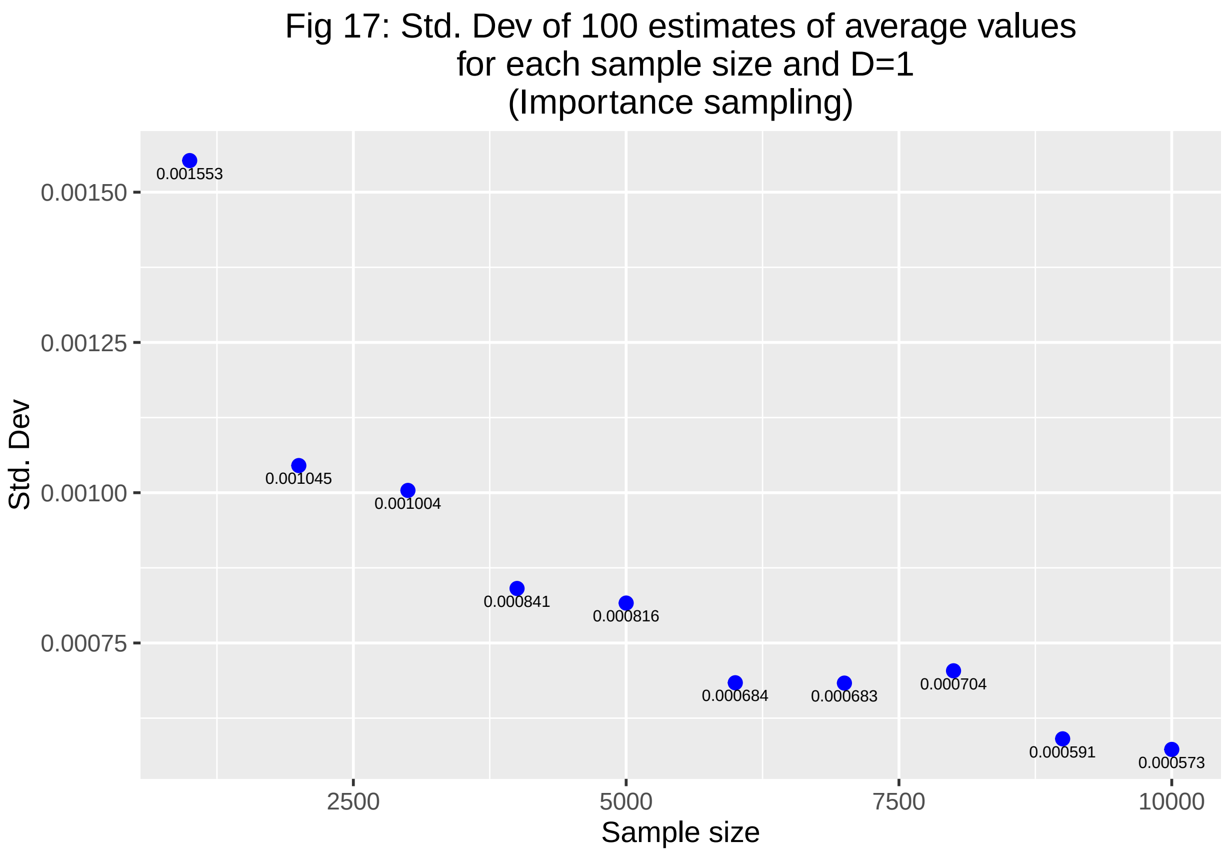
We know that x belongs to the uniform density function between [-5,5]. Therefore E[c(x)] between the interval [-5,5] can be obtained by using importance function, $g(x)$ as a normal distribution with mean of 0 and std. dev. of 1.67 (which is obtained by 5/3, since 99.7% of all the values of a std. distribution should lie within 3 std. deviations from the mean).

The following R code will get the estimate of E[c(x)] using importance sampling, for D=1 dimension:

```r
f <- function(X)
  {
  #X = Matrix with random vectors
  D <- nrow(X)
  #N <- ncol(X)

  m <- (1/((2*pi)^(D/2)))*exp(-0.5*diag(t(X)%*%(X)))

  return(m)
  }


w <- function(x) dunif(x, min=-5,max=5)/dnorm(x, mean=0, sd=1.67)

D <- 1
s <- 0
temp_stg <- vector(length=100)
avg <- vector(length=10)
std_dev <- vector(length=10)

for(j in 1:10)
{
    s <- s+1000

    for(i in 1:100)
    {
      X=rnorm(s*D,mean=0,sd=1.67)
      Y=w(X)*f(matrix(X,nrow=D,byrow=TRUE))
      temp_avg[i] <- mean(Y)/D
    }

    avg[j] <- mean(temp_avg)
    std_dev[j] <- sd(temp_avg)

}

importance_df <- data.frame(sample_size=as.factor(
  seq(from=1000,to=10000,by=1000)),
  average=avg,std_dev=std_dev,
  coeff_of_var=std_dev/avg)
```

```r
ggplot(importance_df,aes(sample_size,average, label=round(average,6)))+
  geom_point(size=2,color="blue")+
  geom_hline(aes(yintercept=0.1),color="red",size=.5)+
  labs(title="Fig 16: Avg. values for various sample sizes\n
       computed with 100 estimates, for each sample size and D=1\n
       (Importance sampling)",
       x="Sample size",
       y="Average value")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)

ggplot(importance_df,aes(sample_size,std_dev, label=round(std_dev,6)))+
  geom_point(size=2,color="blue")+
  labs(title="Fig 17: Std. Dev of 100 estimates of average values\n
       for each sample size and D=1\n(Importance sampling)",
       x="Sample size",y="Std. Dev")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)


kable(importance_df)
```





sample_size	average	std_dev	coeff_of_var
1000	0.1001224	0.0015526	0.0155066
2000	0.0998016	0.0010452	0.0104725
3000	0.0999317	0.0010038	0.0100449
4000	0.0999154	0.0008406	0.0084135
5000	0.0999716	0.0008164	0.0081665
6000	0.0999685	0.0006838	0.0068403
7000	0.1001432	0.0006832	0.0068226
8000	0.0999893	0.0007037	0.0070377
9000	0.1000624	0.0005905	0.0059016
10000	0.0999963	0.0005729	0.0057295

The above figures show that as the sample size increases, the estimated value's accuracy increases, and the std. deviation of the estimate also decreases as the sample size increases.

Let us repeat the same simulation for D=2.

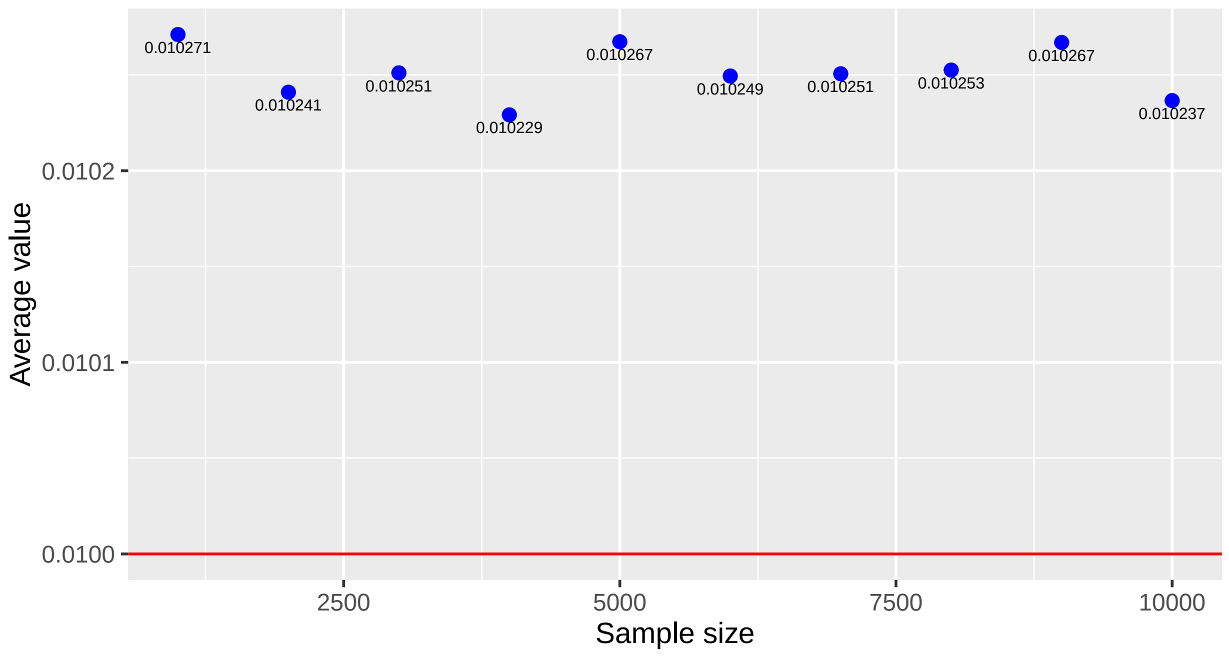
```
D <- 2
s <- 0
temp_stg <- vector(length=100)
avg <- vector(length=10)
std_dev <- vector(length=10)

for(j in 1:10)
{
```

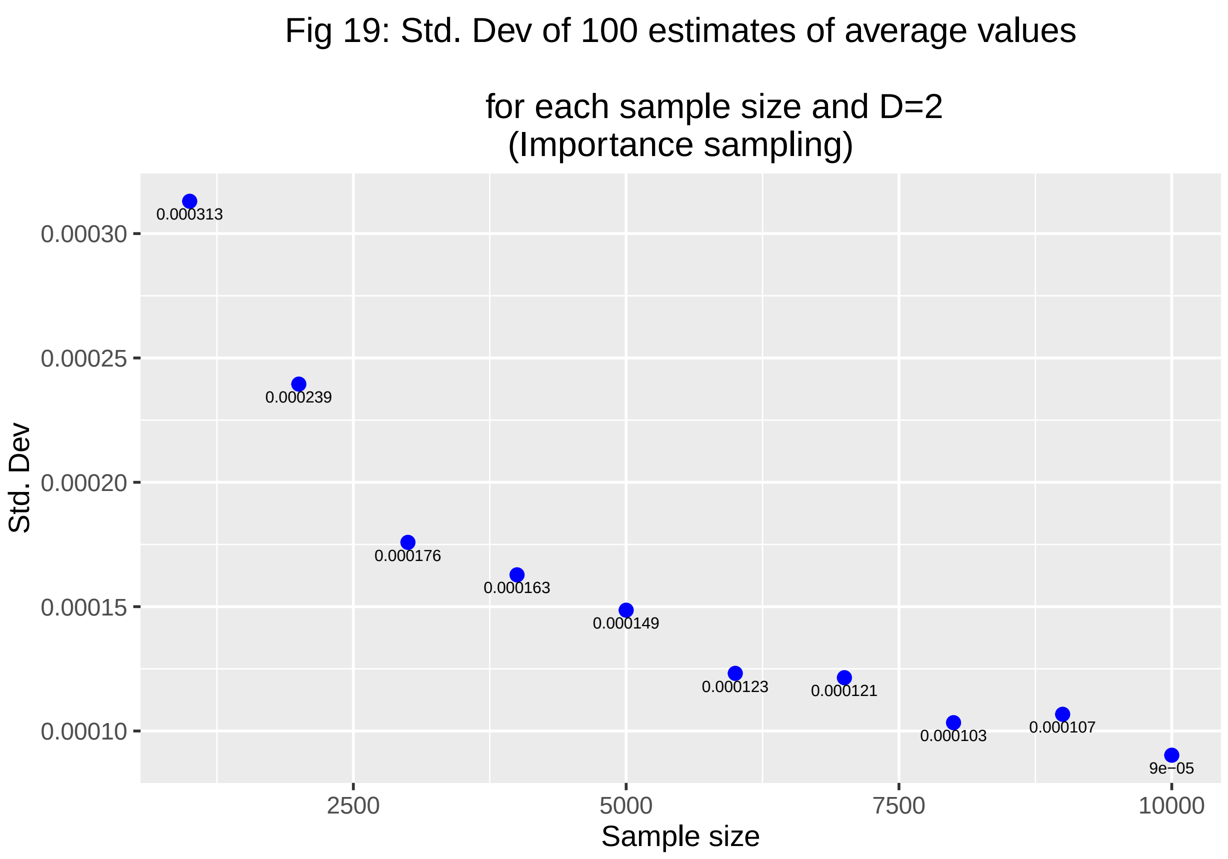
```r
    s <- s+1000

    for(i in 1:100)
    {
      X=rnorm(s*D,mean=0,sd=1.67)
      Y=w(X)*f(matrix(X,nrow=D,byrow=TRUE))
      temp_avg[i] <- mean(Y)/D
    }

    avg[j] <- mean(temp_avg)
    std_dev[j] <- sd(temp_avg)

}

importance_df_1 <- data.frame(sample_size=as.factor
                              (seq(from=1000,to=10000,by=1000)),
                              average=avg,std_dev=std_dev,
                              coeff_of_var=std_dev/avg)


ggplot(importance_df_1,aes(sample_size,average, label=round(average,6)))+
  geom_point(size=2,color="blue")+
  geom_hline(aes(yintercept=0.01),color="red",size=.5)+
  labs(title="Fig 18: Avg. values for various sample sizes\n
       computed with 100 estimates, for each sample size and D=2\n
       (Importance sampling)",
       x="Sample size",y="Average value")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)

ggplot(importance_df_1,aes(sample_size,std_dev, label=round(std_dev,6)))+
  geom_point(size=2,color="blue")+
  labs(title="Fig 19: Std. Dev of 100 estimates of average values\n
       for each sample size and D=2\n
       (Importance sampling)",
       x="Sample size",y="Std. Dev")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)


kable(importance_df_1)
```

Fig 18: Avg. values for various sample sizes

computed with 100 estimates, for each sample size and D=2

(Importance sampling)





sample_size	average	std_dev	coeff_of_var
1000	0.0102711	0.0003130	0.0304736
2000	0.0102410	0.0002395	0.0233847
3000	0.0102510	0.0001758	0.0171532
4000	0.0102292	0.0001628	0.0159149
5000	0.0102673	0.0001486	0.0144703
6000	0.0102494	0.0001232	0.0120182
7000	0.0102506	0.0001214	0.0118468
8000	0.0102525	0.0001034	0.0100806
9000	0.0102669	0.0001067	0.0103947
10000	0.0102366	0.0000903	0.0088184

The estimate obtained for D=2, using importance sampling is not as accurate as the other methods. But the std. deviation of the estimate has decreased with the sample size, and is almost zero at 10000 sample size.

Summary

Let us analyze all the methods together:

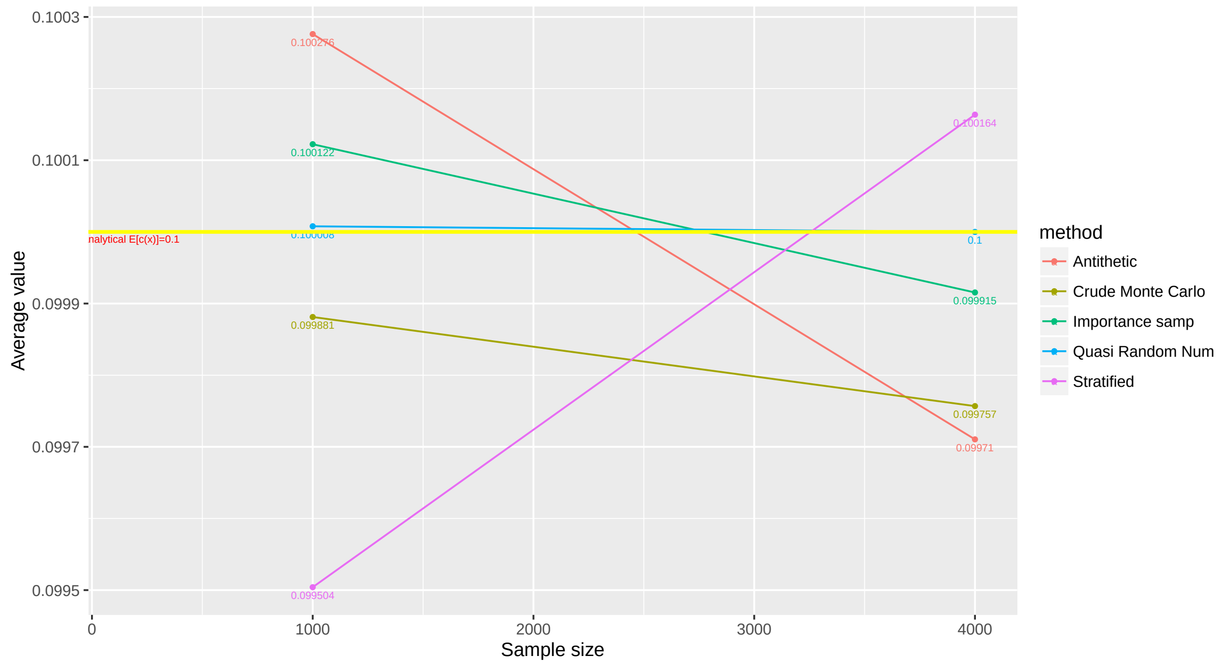
Analyzing the estimates for sample sizes of 1000 and 4000, for D=1:

```
df$method <- "Crude Monte Carlo"
sobol_df$method <- "Quasi Random Num"
antithetic_df$method <- "Antithetic"
```

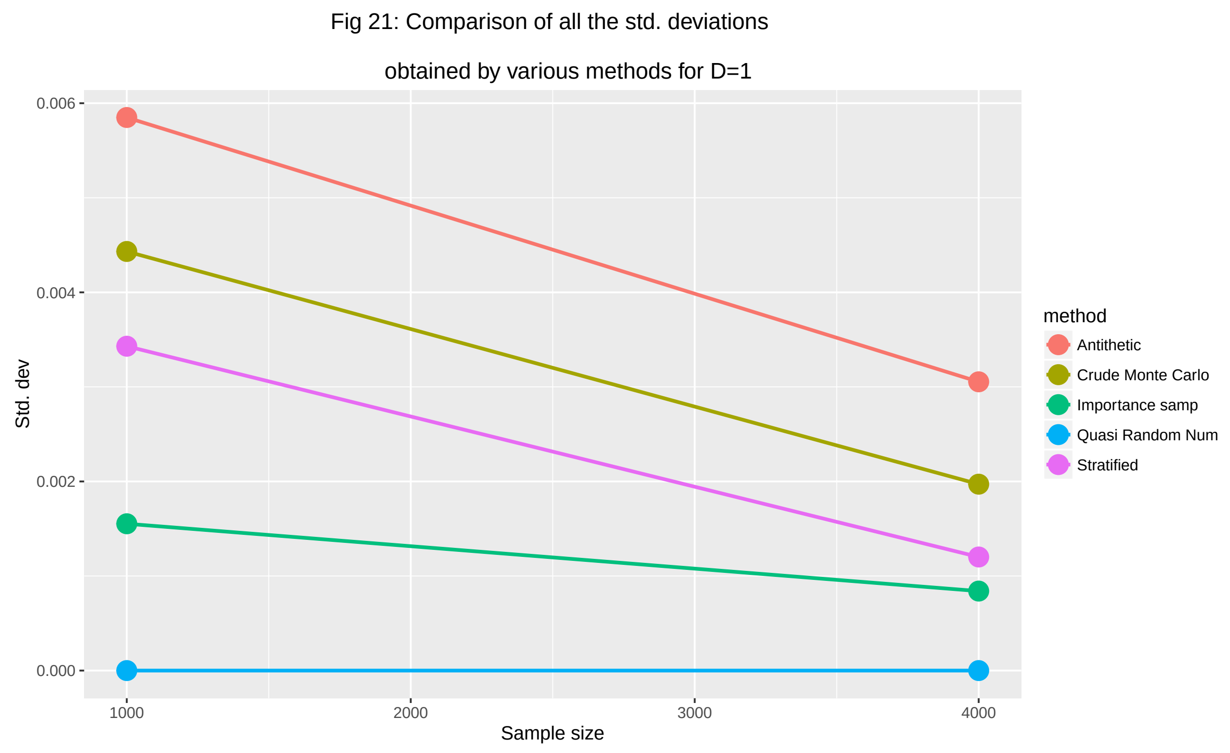
```
stratified_df$method <- "Stratified"
importance_df$method <- "Importance samp"

summary_df<-
rbind(
df[c(1,4),],
sobol_df[c(1,4),],
antithetic_df[c(1,4),],
stratified_df[c(1,4),],
importance_df[c(1,4),]
)

summary_df_avg <- summary_df[,c(1,2,5)]

ggplot(data=summary_df_avg,aes(x=sample_size,y=average,
                              color=method,group=method,label=round(average,6)))+
geom_point(size=1)+
geom_line(size=.5)+
labs(title="Fig 20: Comparison of all the average values \n
     obtained by various methods for D=1",
     x="Sample size",y="Average value")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)+
    geom_hline(aes(yintercept=0.1),color="yellow",size=1)+
      annotate("text", label = "Analytical E[c(x)]=0.1",
               x = 175, y = 0.09999, size = 2, colour = "red")
```

Fig 20: Comparison of all the average values

obtained by various methods for D=1



```
summary_df_sd <- summary_df[,c(1,3,5)]

ggplot(data=summary_df_sd,aes(x=sample_size,y=std_dev,color=method,group=method))+
geom_point(size=5)+
geom_line(size=1)+
labs(title="Fig 21: Comparison of all the std. deviations \n
     obtained by various methods for D=1",
     x="Sample size",y="Std. dev")
```



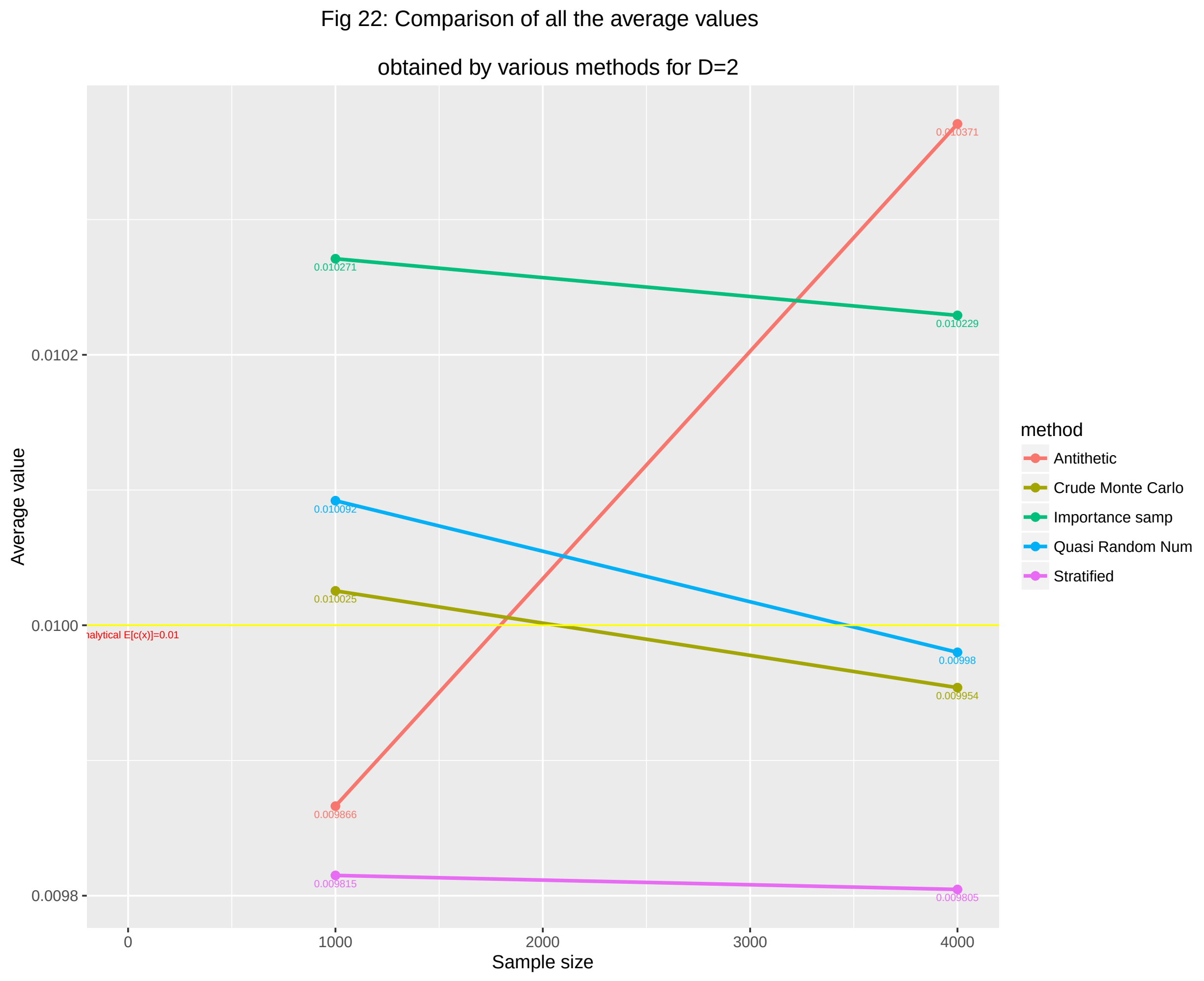
The above figure shows that the estimated value of E[c(x)] is almost equal to the analytical value of 0.1 for all the methods. However the Quasi-Random numbers have obtained the best accuracy. At the sample size of 10000, the estimate obtained using Quasi Random numbers is exactly equal to 0.1.

The std. deviation of the estimate decreases as the sample size increases, but the std.deviation is almost zero for the estimate obtained using Quasi-Random numbers.

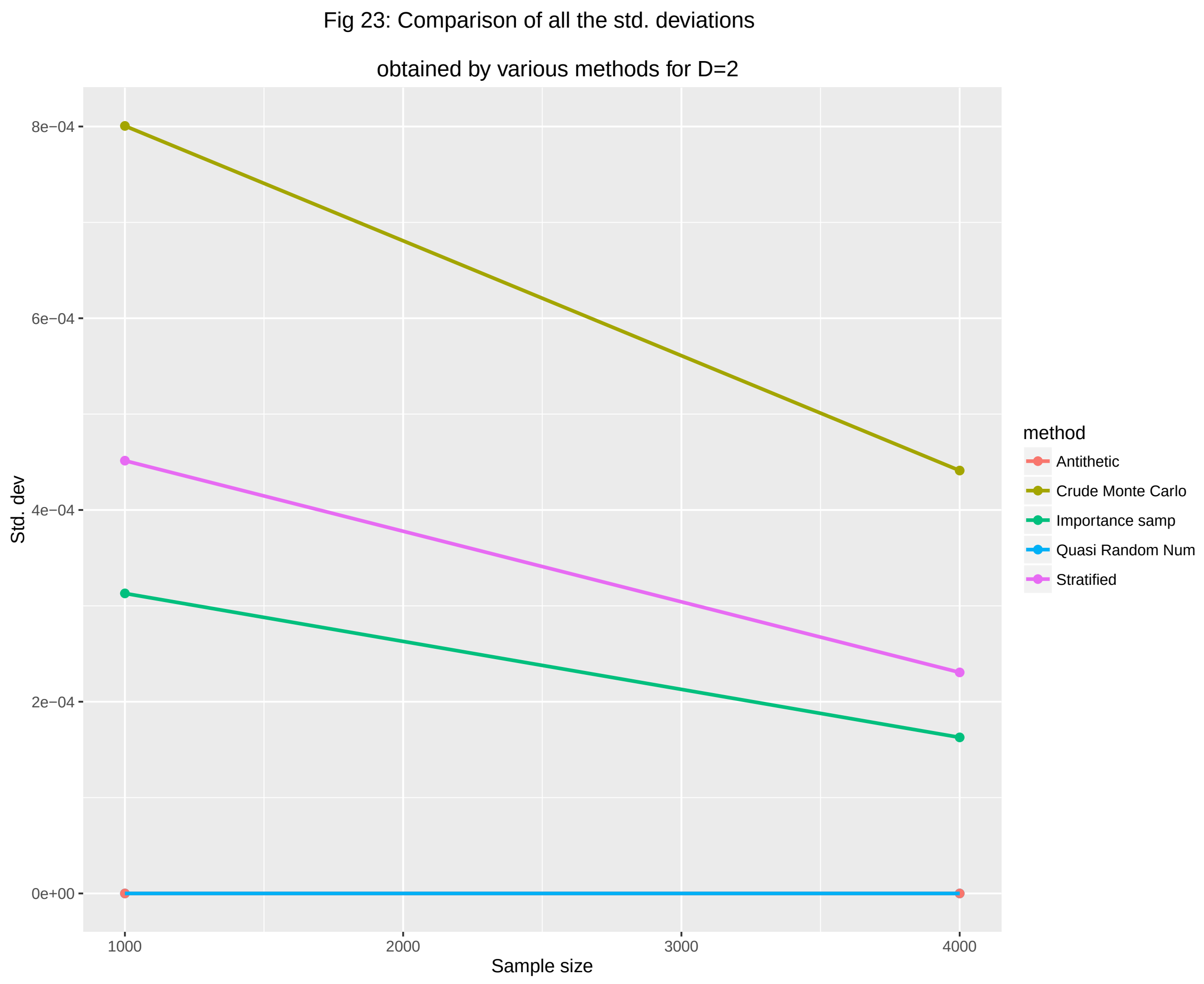
Analyzing the estimates for sample sizes of 1000 and 4000, for D=2:

```
df_1$method <- "Crude Monte Carlo"
sobol_df_1$method <- "Quasi Random Num"
antithetic_df_1$method <- "Antithetic"
stratified_df_1$method <- "Stratified"
importance_df_1$method <- "Importance samp"

summary_df_1 <-
rbind(
df_1[c(1,4),],
sobol_df_1[c(1,4),],
antithetic_df_1[c(1,4),],
stratified_df_1[c(1,4),],
importance_df_1[c(1,4),]
)
```

```r
summary_df_1_avg <- summary_df_1[,c(1,2,5)]


ggplot(data=summary_df_1_avg,aes(x=sample_size,
                                 y=average,color=method,
                                 group=method,
                                 label=round(average,6)))+
geom_point(size=2)+
geom_line(size=1)+
labs(title="Fig 22: Comparison of all the average values \n
     obtained by various methods for D=2",
     x="Sample size",
     y="Average value")+
      geom_text(hjust = 0.5, vjust=1.6,nudge_x = 0.05,size=2)+
    geom_hline(aes(yintercept=0.01),color="yellow",size=.5)+
      annotate("text", label = "Analytical E[c(x)]=0.01",
               x = .75, y = 0.009993, size = 2, colour = "red")
```



```
summary_df_1_sd <- summary_df_1[,c(1,3,5)]

ggplot(data=summary_df_1_sd,aes(x=sample_size,y=std_dev,color=method,group=method))+
geom_point(size=2)+
geom_line(size=1)+
labs(title="Fig 23: Comparison of all the std. deviations \n
     obtained by various methods for D=2",
     x="Sample size",y="Std. dev")
```

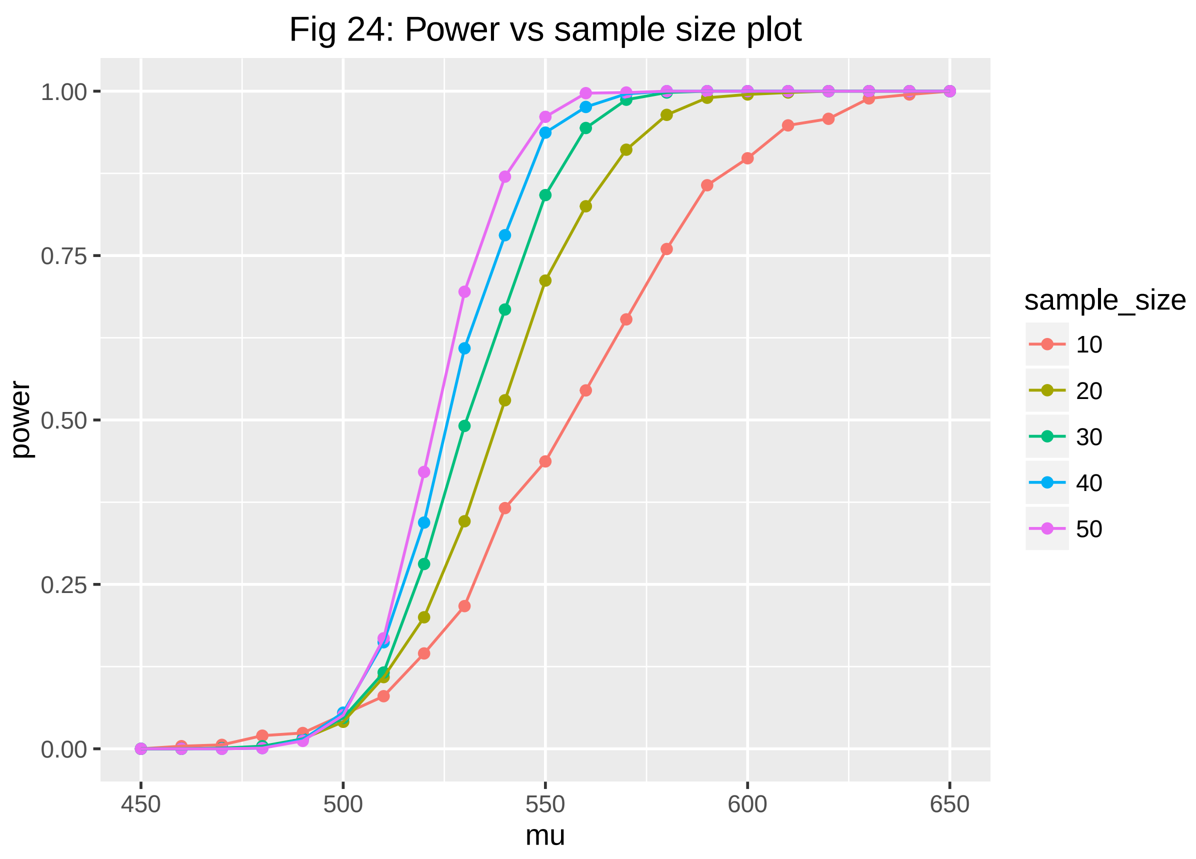


For D=2 also, the estimate obtained using Quasi Random numbers is the best, followed by crude random numbers. The std. deviation of the estimate is zero for Quasi random numbers method and antithetic method. Since the estimate obtained by quasi random numbers is almost equal to the analytical value, and also the std. error of the quasi random numbers estimate is minimum, it is suggested to use Quasi random numbers to estimate any integral or expected value.

SCR 6.3

The following R code will compute the power of the test for sample sizes of 10,20,30,40,50, and plots the graph between power and various true mean values between 450 and 650.

```r
#Sample size
#Define the sample sizes
n <- seq(from=10,to=50,by=10)

#Number of sample values
m <- 1000

#Assumed mean
mu0 <- 500

#Assumed std. dev
sigma <- 100

df <- data.frame()

#True mean values considered
mu <- c(seq(450,650,10))

#Define the required vectors
M <- length(mu)
power <- numeric(M)

#Determine the power of the test for various sample sizes, and at
#different true mean values

for(j in 1:length(n))
{

    for(i in 1:M)
    {

      mu1 <- mu[i]

      pvalues <- replicate(m, expr={
        x <- rnorm(n[j],mean=mu1,sd=sigma)
        ttest <- t.test(x,
                        alternative = "greater",mu=mu0)
        ttest$p.value})
      power[i] <- mean(pvalues<=0.05)

    }
    df <- rbind(df,data.frame(sample_size=n[j],mu=mu,power=power))
}


#Plot the graph
library(ggplot2)
df$sample_size <- as.factor(df$sample_size)


ggplot(df,aes(x=mu,y=power,color=sample_size))+
  geom_point()+
  geom_path()+
  labs(title="Fig 24: Power vs sample size plot")
```



The plot clearly shows that the power of the test increases more rapidly as the sample size increases.

SCR 6.4

As per central limit theorem, irrespective of the population disrtibution, if a random sets of samples (of size 30 or more) are drawn from the population then the means of the sample will have a normal distribution, with almost the same mean as the population mean, and the std. dev of the mean estimated will be std. dev of the means divided by the square root of the sample size (if the population std. dev is known, then it will be std. dev of the population divided by the square root of the sample size). So the 95% confidence interval for the mean estimate can be obtained as:

$$\pm 1.96 s/\sqrt{n}$$

where s = std deviation(or std. error), and n = sample size. -1.96 and +1.96 are the z-scores at 0.025 and 0.975 p-values.

In the following code we will assume the mean value of the log-normal distribution as 10 with standard deviation as 1.

```
x <- vector(length=1000)
for (i in 1:1000)
{
  x[i] <- mean(log(rlnorm(1000, meanlog = 10, sdlog = 1)))
}
```

The mean value estimated using a sample size of 1000 is 9.9992059 and the 95% confidence interval is: [9.9972571,10.0011546]

SCR 7.1

We have to estimate the correlation between LSAT and GPA using Jackknife method. The std. error of this estimate must also be obtained.

```r
#Reading the data
LSAT <- c(576,635,558,578,666,580,555,661,651,605,653,575,545,572,594)
GPA <- c(339,330,281,303,344,307,300,343,336,313,312,274,276,288,296)

n <- length(LSAT)
jack <- vector(length=n)

for(i in 1:n)
{
  jack[i] <- cor(LSAT[-i],GPA[-i])
}
jack_corr <- mean(jack)
jack_bias <- (n-1) * (jack_corr - cor(LSAT,GPA))
jack_se <- sqrt((n-1) * mean((jack - mean(jack))^2))
```

The estimated correlation is 0.7759121, with a bias of -0.0064736 and standard error of 0.1425186

SCR 7.4

```r
y <- c(3,5,7,18,43,85,91,98,100,130,230,487)
mean(y)
```

```
## [1] 108.0833
```

Let us use the Maximum Likelihood estimator for the mean(β) of the exponential distribution. We can obtain this by getting the average of the hours between failures and getting the reciprocal. But we will use the MLE (Maximum Likelihood Estimator) to get the value of β. Once the mean of the exponential distribution is obtained, we can get the mean of the failures, λ as $\frac{1}{\beta}$.

$$L(\beta) = \prod_{i=1}^{12} \frac{1}{\beta} e^{\frac{-y_i}{\beta}}$$

Where $y = \{3, 5, 7, 18, 43, 85, 91, 98, 100, 130, 230, 487\}$ and y_i represents each of the i^{th} element of y.

We have to find for what value of β is the $L(\beta)$ is maximized (for what value of β do we get the observed data with maximum probability).

Applying natural log on both sides, will give us:

$$ln(L(\beta)) = -12ln(\beta) + \sum_{i=1}^{12}(-y_i/\beta)$$

Differentiating with respect to β, and equating the differentiated outcome to 0, will get β as:

$$\beta = 0$$
$$\beta = \frac{3+5+7+18+43+85+91+98+100+130+230+487}{12} = 108.0833$$

Since the β cannot be 0 for exponential distribution, we consider β as 108.0833. At this value of β, the function $ln(L(\beta))$ is maximum, since the second derivative of $ln(L(\beta))$ evaluates to a value of less than zero for $\beta = 108.0833$.

The reciprocal of β, the λ will be the average number of failures. Hence $\lambda = 1/108.0833 = 0.00925212$

Hence we will have 0.00925212 failures per hour (the hazard rate).

We will use the following R code to compute the std. dev and bias of our estimation:

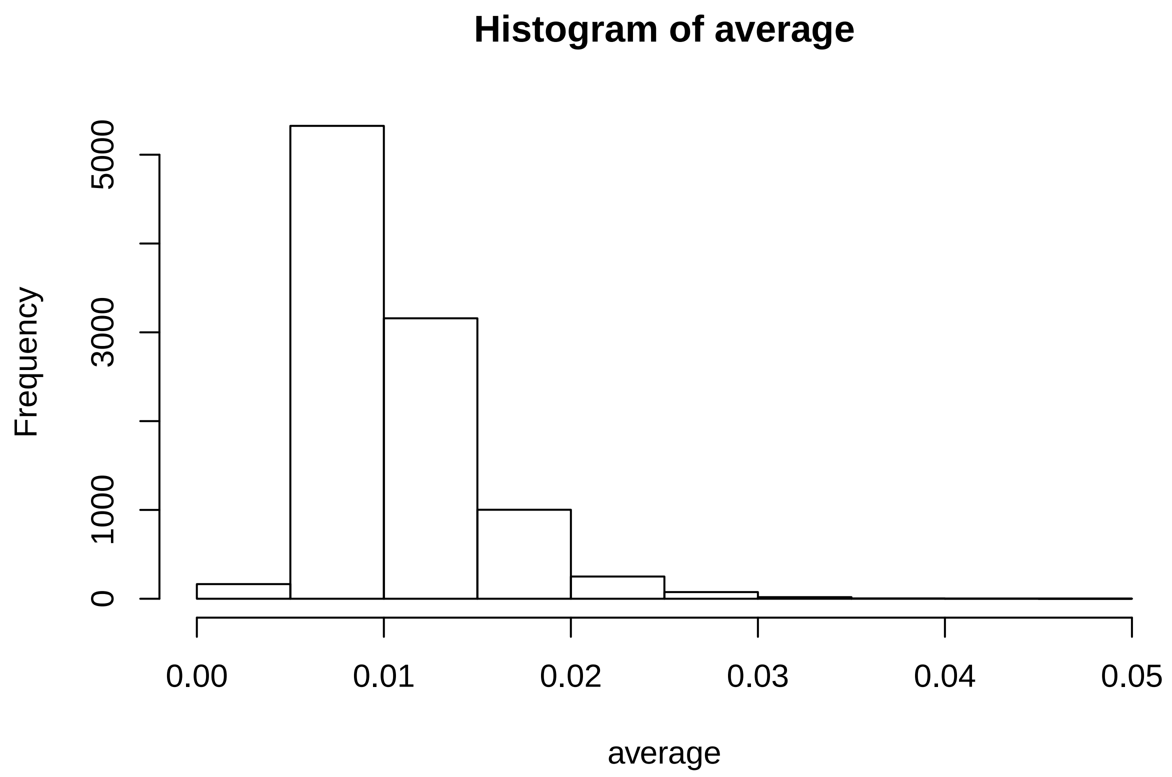
```r
##R code to compute the std. dev and bias.
#Read the data
y <- c(3,5,7,18,43,85,91,98,100,130,230,487)

#sample size
n <- length(y)

#Number of iterations
B <- 10000

#Declare a vector to contain the avg. failures
average <- numeric(200)

for(b in 1:B)
{
  i <- sample(1:n,size=n,replace=TRUE)
  #You must reciprocate the mean to get the avg. failures number
  average[b] <- 1/mean(y[i])

}

#Get the avg. number of failures
mean(average)
```

```
## [1] 0.01055697
```

```r
#Get the bias
mean(average) - (1/mean(y))
```

```
## [1] 0.001304848
```

```r
#Plot the histogram of average failures/hour
hist(average)
```



```r
#Std. dev
sd(average)
```

```
## [1] 0.004264762
```

Hence the estimated average via bootstrap process is: 0.010557 failures/hour, with a bias of 0.0013048 and std. error of the estimate is: 0.0042648