Homework 4

DATA604 Simulation and Modeling

Daniel Dittenhafer

March 22, 2016

1

In this problem, you will implement and investigate a series of variance reduction procedures for Monte Carlo method by estimating the expected value of a cost function c(x) which depends on a D-dimensional random variable x.

The cost function is:

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

where

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

Goal: estimate E[c(x)] - the expected value of c(x) - using Monte Carlo methods and see how it compares to the real value, which you are able to find by hand.

```
# First define the cost function as an R function
costFx <- function(x)
{
  b <- exp(-0.5 * t(x) * x)
  D <- length(x)
  res <- (1 / ((2 * pi)^(D/2))) * b
  return (res)
}
```

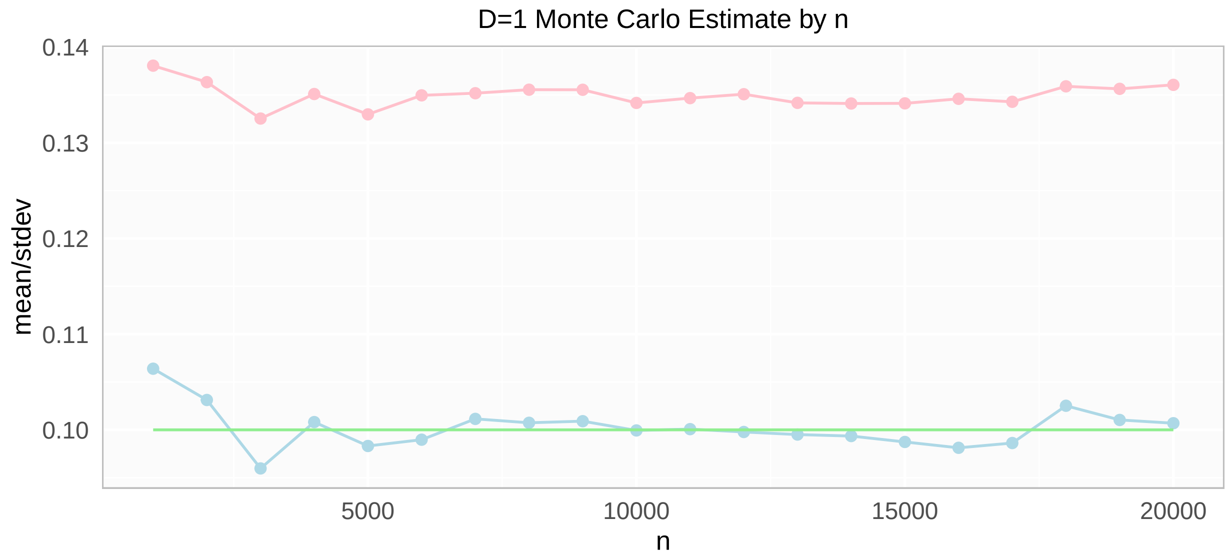
a) Crude Monte Carlo

```
crudeMC <- function(n, min, max, d = 1)
{
  # Need a loop in here
  theta.hat <- matrix(nrow=d, ncol=n)
  for(i in 1:n)
  {
    x <- runif(d, min, max)
    theta.hat[,i] <- costFx(x)
    #gXbar <- (1/n) * costFx(x)
    #print(((max-min) * gXbar))
    #theta.hat[,i] <- t((max-min) * gXbar)
  }

  return (theta.hat)
}

#ret <- crudeMC(10, -5, 5, 2)
#ret
#mean(ret)
```

```
crudeMc.Loop <- function(d, verbose=FALSE)
{
  crudeMc.result <- data.frame(mean=c(), stdev=c(), n=c())
  for(n in seq(1000, 20000, by=1000))
  {
    res <- crudeMC(n=n, min=-5, max=5, d=d)
    if(verbose)
    {
      #print("Data")
      #print(res)
      #print("Mean")
      #print(mean(res))
      print(dim(res))
    }

    crudeMc.result <- rbind(crudeMc.result, data.frame(mean=mean(colSums(res)), stdev=sd(colSums(res)), n=n))
  }

  crudeMc.result$EcActual <- (1/10)^d
  crudeMc.result$CoefVari <- crudeMc.result$stdev / crudeMc.result$mean

  return (crudeMc.result)
}
```

In the code below, we call the crude Monte Carlo loop function, show the top entries and visualize the result for D=1. The blue line represents the mean value, pink is the standard deviation, and the green line is the analytical value for $E[c(x)] = (1/10)^D$.

```
crudeMc.D1 <- crudeMc.Loop(d=1)
```

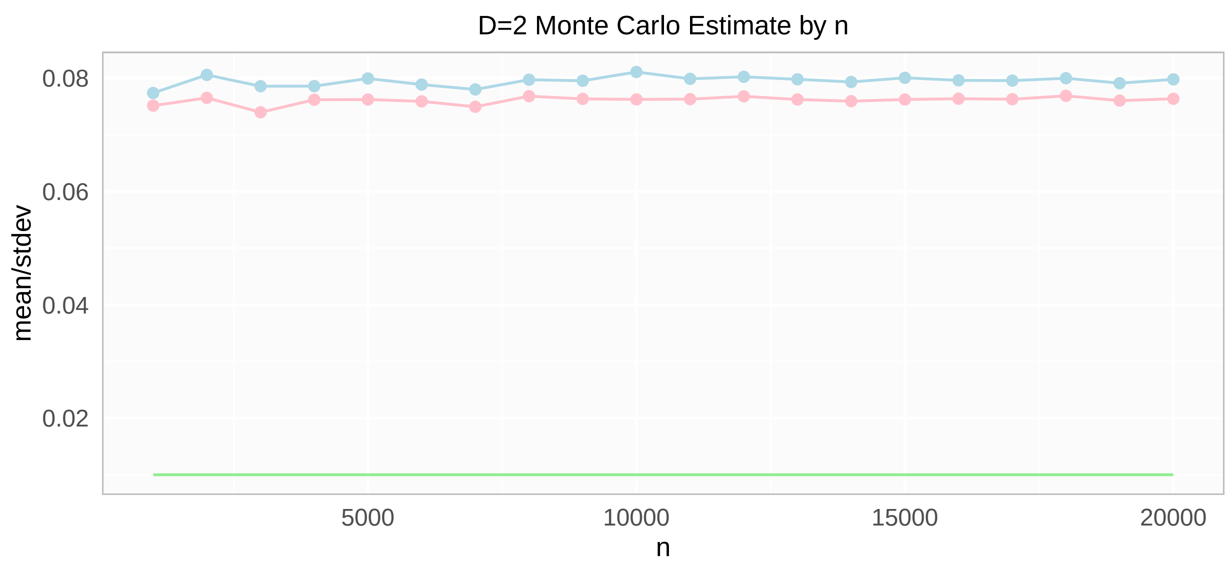


mean	stdev	n	EcActual	CoefVari
0.1063893	0.1380583	1000	0.1	1.297670
0.1031207	0.1363369	2000	0.1	1.322110
0.0959678	0.1325320	3000	0.1	1.381005
0.1008113	0.1350975	4000	0.1	1.340103
0.0983136	0.1329688	5000	0.1	1.352496
0.0989702	0.1349571	6000	0.1	1.363614
0.1011450	0.1351806	7000	0.1	1.336502
0.1007386	0.1355480	8000	0.1	1.345541
0.1009048	0.1355401	9000	0.1	1.343247

mean	stdev	n	EcActual	CoefVari
0.0999374	0.1341640	10000	0.1	1.342481
0.1000755	0.1346675	11000	0.1	1.345659
0.0997761	0.1350776	12000	0.1	1.353808
0.0995073	0.1341704	13000	0.1	1.348347
0.0993517	0.1341042	14000	0.1	1.349792
0.0987340	0.1341199	15000	0.1	1.358397
0.0981232	0.1345956	16000	0.1	1.371700
0.0986219	0.1342821	17000	0.1	1.361585
0.1025172	0.1359001	18000	0.1	1.325632
0.1010334	0.1356360	19000	0.1	1.342488
0.1006939	0.1360526	20000	0.1	1.351150

In the code below, we call the crude Monte Carlo loop function, show the top entries and visualize the result for D=2.

```
crudeMc.D2 <- crudeMc.Loop(d=2, verbose=FALSE)
```



mean	stdev	n	EcActual	CoefVari
0.0773534	0.0751242	1000	0.01	0.9711811
0.0805473	0.0764963	2000	0.01	0.9497066
0.0785478	0.0739540	3000	0.01	0.9415152
0.0785672	0.0761640	4000	0.01	0.9694130
0.0799151	0.0762010	5000	0.01	0.9535247
0.0788421	0.0758748	6000	0.01	0.9623644
0.0779757	0.0749225	7000	0.01	0.9608438
0.0796979	0.0767931	8000	0.01	0.9635518
0.0795152	0.0763194	9000	0.01	0.9598089
0.0810627	0.0762206	10000	0.01	0.9402672
0.0798560	0.0762696	11000	0.01	0.9550891
0.0802091	0.0767647	12000	0.01	0.9570569
0.0797709	0.0762072	13000	0.01	0.9553259
0.0792998	0.0759092	14000	0.01	0.9572437
0.0800337	0.0762069	15000	0.01	0.9521841
0.0795884	0.0763470	16000	0.01	0.9592729
0.0795370	0.0762558	17000	0.01	0.9587454
0.0799604	0.0768504	18000	0.01	0.9611049
0.0790827	0.0760083	19000	0.01	0.9611245
0.0797646	0.0763336	20000	0.01	0.9569865

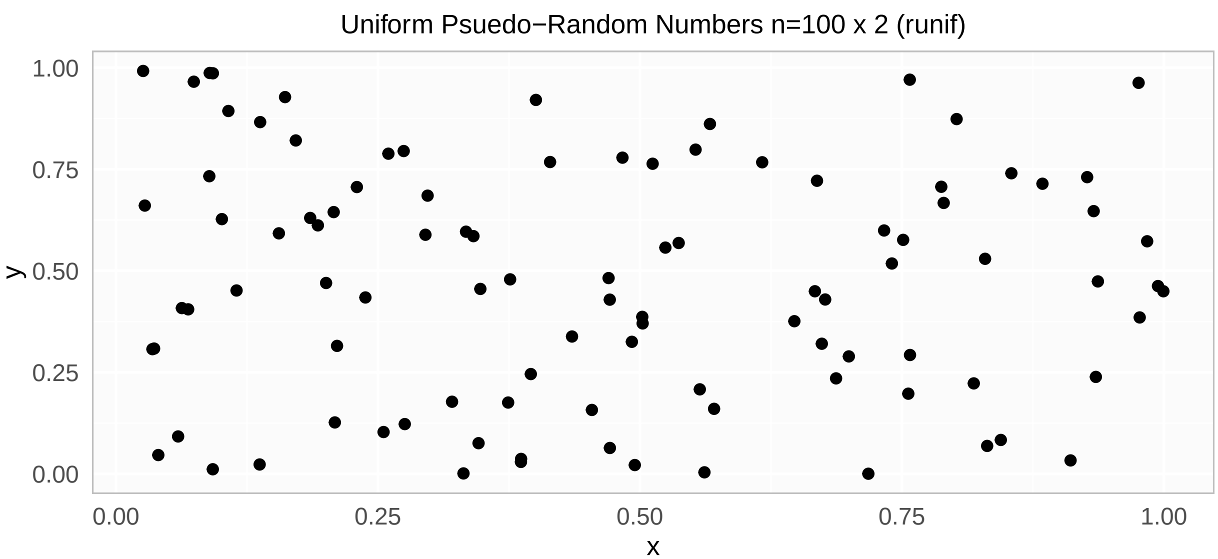
Quasi-Random Numbers

First, we compare the typical uniform random numbers from R's `runif` function to Sobol quasi-random numbers from `randtoolbox::sobol` function. 100 pairs of numbers are drawn from both generators and visualized below.

Uniform Random Numbers The following code segment uses `runif` to generate $m = 100$ random numbers and plots them.

```r
m <- 100
unifRn <- as.data.frame(matrix(runif(m * 2), ncol=2))
colnames(unifRn) <- c("x", "y")
head(unifRn)
```

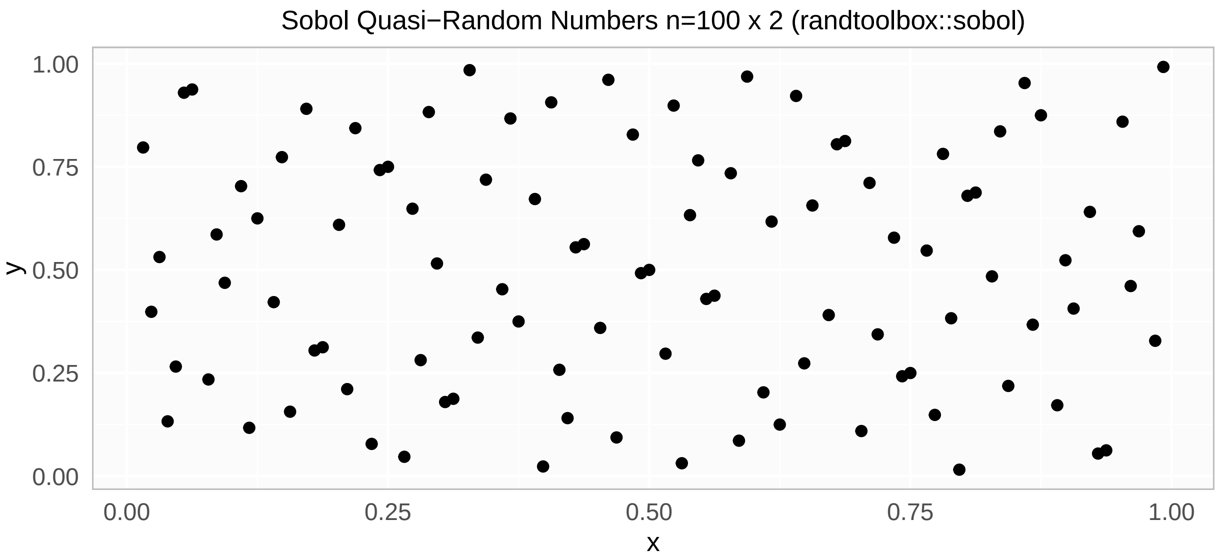
```
##            x         y
## 1 0.97692386 0.3851265
## 2 0.29529562 0.5887170
## 3 0.48334489 0.7785136
## 4 0.97589246 0.9627245
## 5 0.08905365 0.7328018
## 6 0.10104166 0.6271944
```



Sobol Random Numbers The following code segment uses `sobol` to generate $m = 100$ random numbers and plots them.

```r
sobolRn <- as.data.frame(sobol(m, d=2))
colnames(sobolRn) <- c("x", "y")
head(sobolRn)
```

```
##       x     y
## 1 0.500 0.500
## 2 0.750 0.250
## 3 0.250 0.750
## 4 0.375 0.375
## 5 0.875 0.875
## 6 0.625 0.125
```



At $m = 100$, the differences are less obvious, but there is some discernable pattern to the Sobol numbers which is more apparent at great m. As such, the prefix "quasi" seems appropriate. The definition of "quasi" is "seemingly, apparently but not really". These might appear to be random numbers at first glace, but there is quite a pattern, the lattice, as more and more are generated.

Sobol Monte Carlo First, `sobol` based helper functions are defined using the same structure as the prior `runif` based functions. A couple of changes were needed:

- `sobol` returns a matrix. This is converted to a vector for use in the cost function.
- `sobol` has an init parameter, but we don't want to re-initialize every call, so this is bubbled up to the loop function to allow it to drive the re-init.

```
sobolMC <- function(n, d = 1, init=TRUE)
{
  # Need a loop in here
  theta.hat <- matrix(nrow=d, ncol=n)
  for(i in 1:n)
  {
    x <- as.vector(sobol(n=1, d=d, init=init))
    theta.hat[,i] <- costFx(x)
    init <- FALSE # turn off the init for i > 1 iterations
  }

  return (theta.hat)
}

sobolMC(n=10, d=2)
```

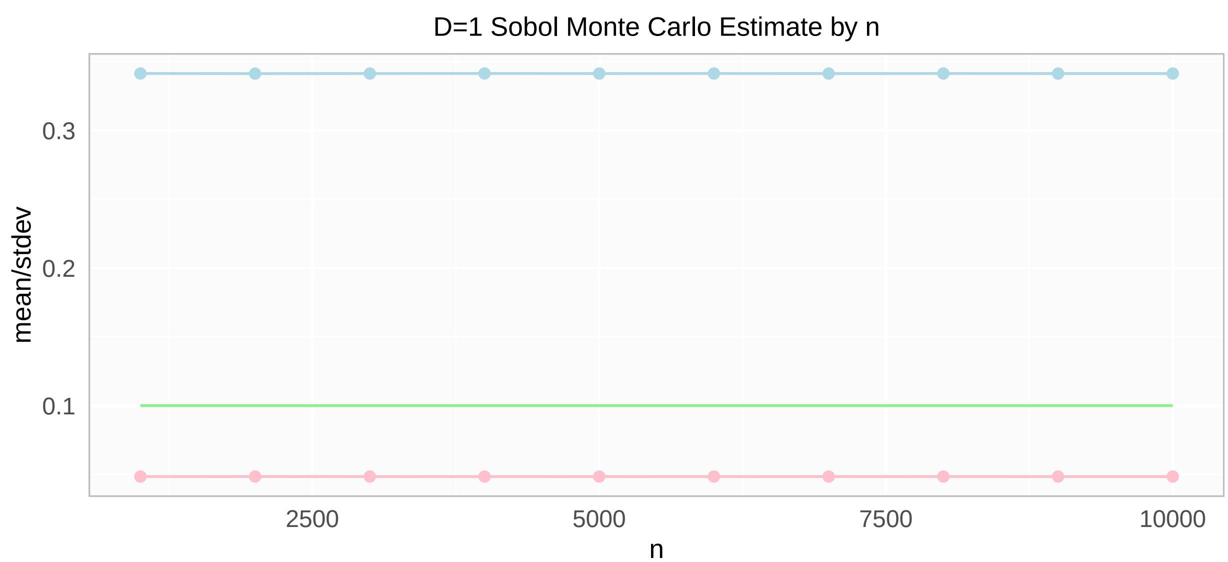
```
##            [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.1404537 0.1201365 0.1542583 0.1483487 0.1085342 0.1309173 0.1579164
## [2,] 0.1404537 0.1542583 0.1201365 0.1483487 0.1085342 0.1579164 0.1309173
##            [,8]      [,9]     [,10]
## [1,] 0.1563817 0.1256563 0.1025577
## [2,] 0.1515704 0.1144113 0.1588444
```

```
sobolMc.Loop <- function(d, verbose=FALSE)
{
  initSobol <- TRUE
  sobolMC.result <- data.frame(mean=c(), stdev=c(), n=c())
```

```r
  for(n in seq(1000, 10000, by=1000))
  {
    res <- sobolMC(n=n, d=d, initSobol)
    if(verbose)
    {
      print("Data")
      print(res)
      print("Mean")
      print(mean(res))
    }

    sobolMC.result <- rbind(sobolMC.result, data.frame(mean=mean(res), stdev=sd(res), n=n))
    initSobol <- FALSE # don't re-initialize sobol every time.
  }

  sobolMC.result$EcActual <- (1/10)^d
  sobolMC.result$CoefVari <- sobolMC.result$stdev / sobolMC.result$mean

  return (sobolMC.result)
}
```

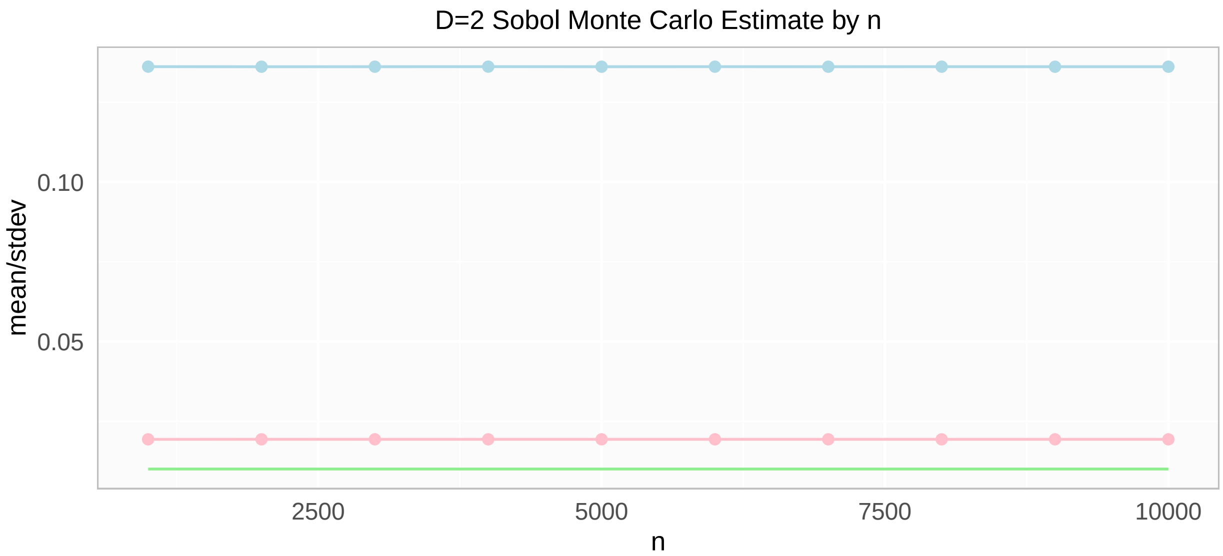
In the code below, we call the Sobol Monte Carlo loop function, show the top entries and visualize the result for D=1. Again, the blue line represents the mean value, pink is the standard deviation, and the green line is the analytical value for $E[c(x)] = (1/10)^D$.

```r
sobolMc.D1 <- sobolMc.Loop(d=1)
```



mean	stdev	n	EcActual	CoefVari
0.3413791	0.0484139	1000	0.1	0.1418186
0.3412909	0.0484595	2000	0.1	0.1419889
0.3413458	0.0484297	3000	0.1	0.1418786
0.3413900	0.0484068	4000	0.1	0.1417932
0.3413302	0.0484301	5000	0.1	0.1418863
0.3413725	0.0484124	6000	0.1	0.1418168
0.3413094	0.0484400	7000	0.1	0.1419239
0.3413702	0.0484120	8000	0.1	0.1418168
0.3413274	0.0484293	9000	0.1	0.1418851
0.3413343	0.0484275	10000	0.1	0.1418770

```r
sobolMc.D2 <- sobolMc.Loop(d=2) #data.frame(mean=c(), stdev=c(), EcActual=c(), n=c()) #
```



mean	stdev	n	EcActual	CoefVari
0.1361935	0.0193106	1000	0.01	0.1417877
0.1361683	0.0193212	2000	0.01	0.1418922
0.1361750	0.0193181	3000	0.01	0.1418626
0.1361905	0.0193129	4000	0.01	0.1418079
0.1361731	0.0193197	5000	0.01	0.1418764
0.1361810	0.0193168	6000	0.01	0.1418467
0.1361718	0.0193191	7000	0.01	0.1418730
0.1361815	0.0193149	8000	0.01	0.1418324
0.1361727	0.0193188	9000	0.01	0.1418701
0.1361776	0.0193173	10000	0.01	0.1418540