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)
}
```

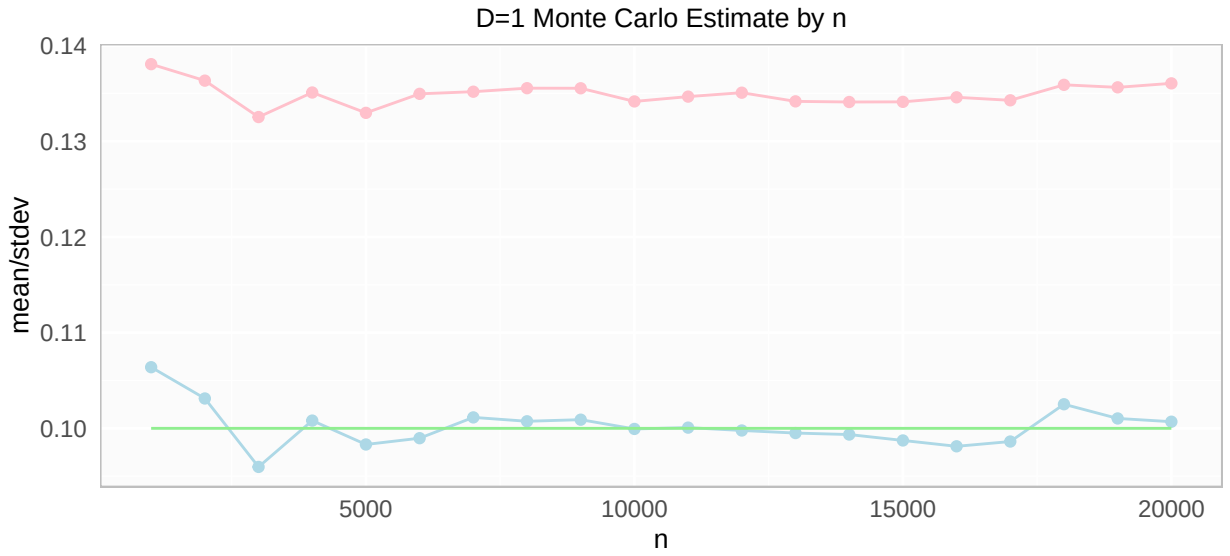
a) Crude Monte Carlo

```
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(res), stdev=sd(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)
```

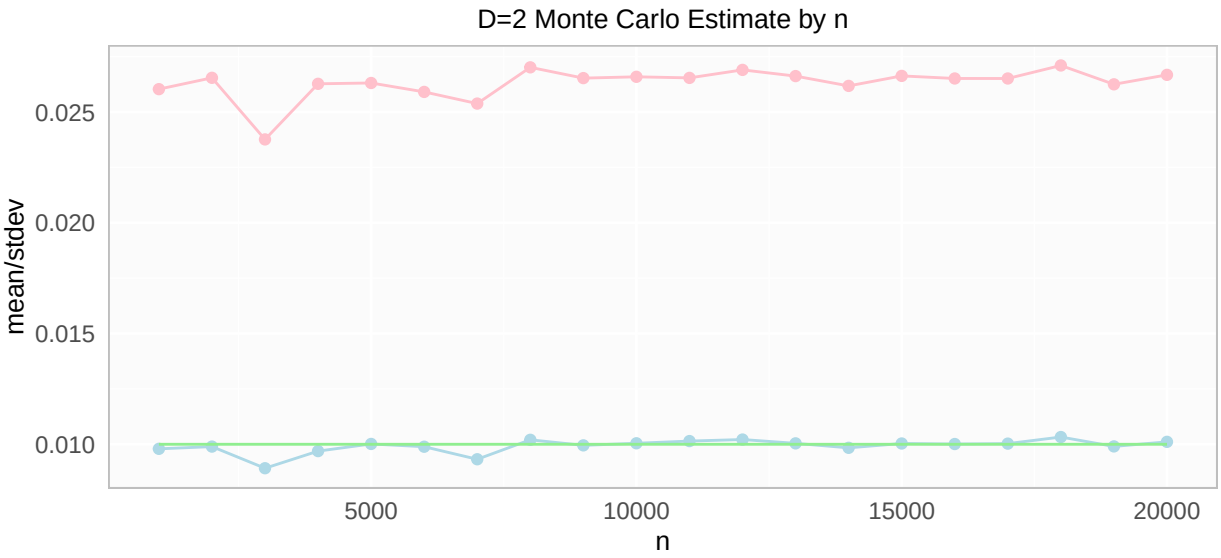


mean	stdev	n	EcActual	CoefVari
0.1063893	0.1380583	1000	0.1	1.297670
0.1031207	0.1363369	2000	0.1	1.322110
0.0959678	0.1325320	3000	0.1	1.381005
0.1008113	0.1350975	4000	0.1	1.340103
0.0983136	0.1329688	5000	0.1	1.352496
0.0989702	0.1349571	6000	0.1	1.363614
0.1011450	0.1351806	7000	0.1	1.336502
0.1007386	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
0.0097945	0.0260313	1000	0.01	2.657744
0.0098958	0.0265426	2000	0.01	2.682196
0.0089194	0.0237615	3000	0.01	2.664035
0.0096924	0.0262718	4000	0.01	2.710546
0.0100151	0.0263094	5000	0.01	2.626973
0.0098905	0.0259061	6000	0.01	2.619294
0.0093241	0.0253798	7000	0.01	2.721957
0.0102021	0.0270152	8000	0.01	2.648000
0.0099517	0.0265291	9000	0.01	2.665797
0.0100456	0.0265880	10000	0.01	2.646730
0.0101451	0.0265396	11000	0.01	2.615998
0.0102142	0.0269004	12000	0.01	2.633622
0.0100423	0.0266203	13000	0.01	2.650813
0.0098407	0.0261778	14000	0.01	2.660149
0.0100364	0.0266291	15000	0.01	2.653255
0.0100113	0.0265128	16000	0.01	2.648279
0.0100301	0.0265124	17000	0.01	2.643296
0.0103272	0.0270998	18000	0.01	2.624112
0.0099044	0.0262491	19000	0.01	2.650248
0.0101104	0.0266759	20000	0.01	2.638474

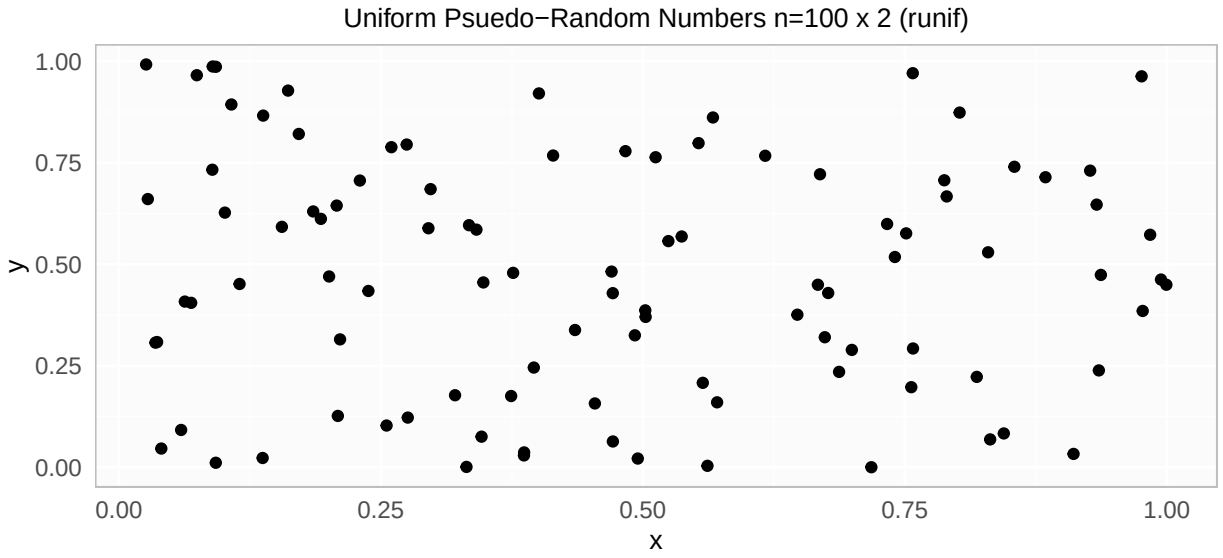
Quasi-Random Numbers

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.

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

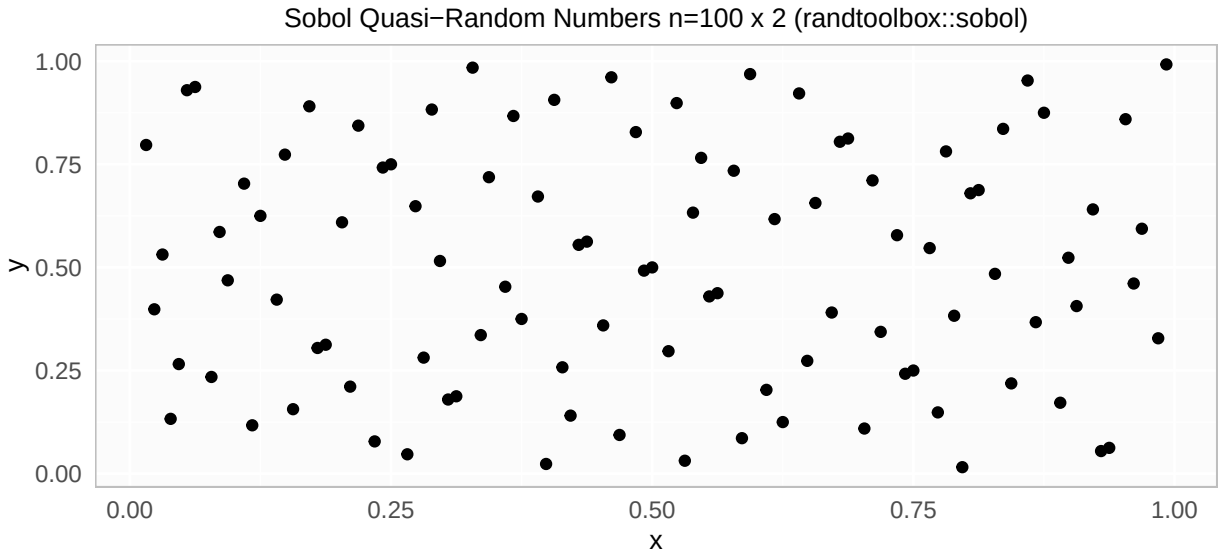
```
##            x         y
## 1 0.97692386 0.3851265
## 2 0.29529562 0.5887170
## 3 0.48334489 0.7785136
## 4 0.97589246 0.9627245
## 5 0.08905365 0.7328018
## 6 0.10104166 0.6271944
```



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

```
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:

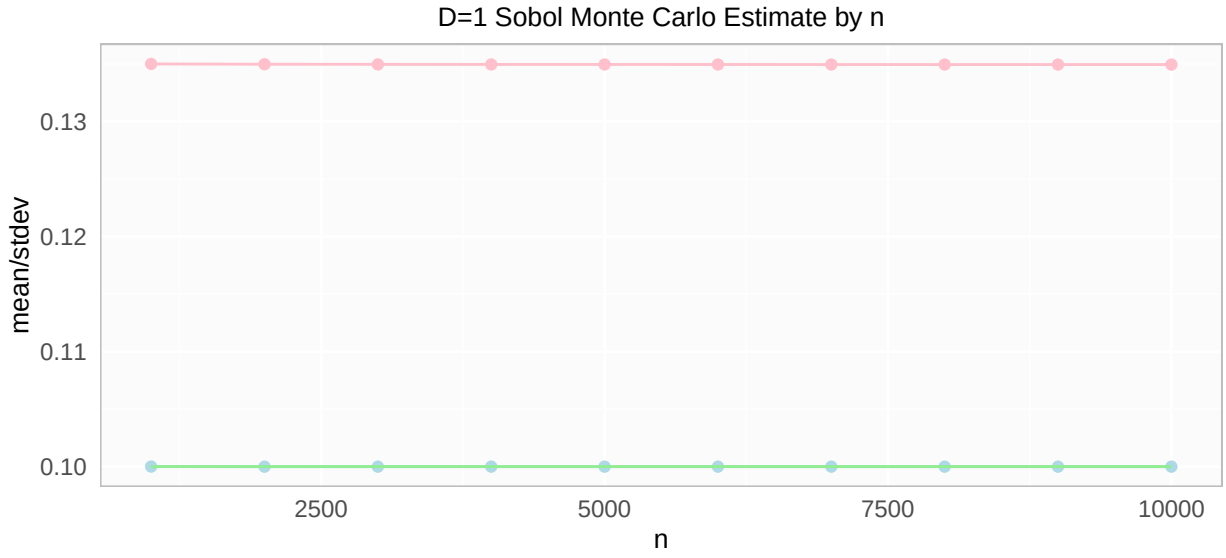
- `sobol` returns a matrix. This is converted to a vector for use in the cost function.
- `sobol` has an init parameter, but we don't want to re-initialize every call, so this is bubbled up to the loop function to allow it to drive the re-init.

```
# Define a function to help us convert a 0-1 RV to a x-y RV
rndRange <- function(x, min, max)
{
  r <- max - min
  p <- x * r
  new <- min + p
  return(new)
}

# Sobol Monte Carlo Inner function
sobolMC <- function(n, min, max, d = 1, init=TRUE)
{
  # Need a loop in here
  theta.hat <- rep(NA, n)
  for(i in 1:n)
  {
    x <- as.vector(sobol(n=1, d=d, init=init))
    x <- rndRange(x, min, max)
    theta.hat[i] <- costFx(x)
    init <- FALSE # turn off the init for i > 1 iterations
  }

  return (theta.hat)
}


#sobolMC(n=10, -5, 5, d=2)
```

```
sobolMc.Loop <- function(d, verbose=FALSE)
{
  initSobol <- TRUE
  sobolMC.result <- data.frame(mean=c(), stdev=c(), n=c())
  for(n in seq(1000, 10000, by=1000))
  {
    res <- sobolMC(n=n, -5, 5, 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 $E[c(x)] = (1/10)^D$.

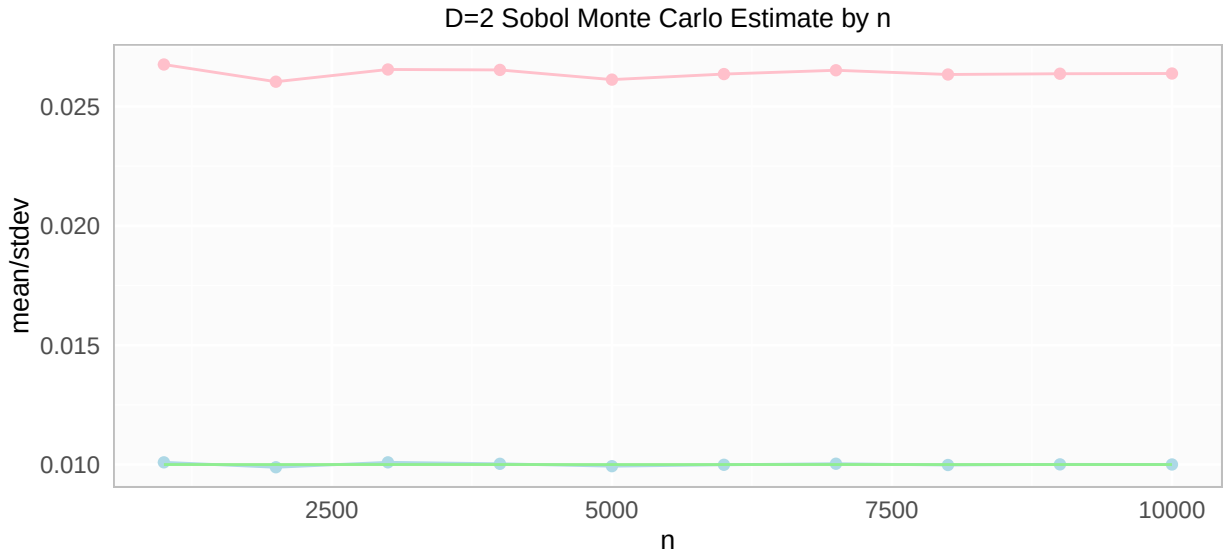
```
sobolMc.D1 <- sobolMc.Loop(d=1)
```



mean	stdev	n	EcActual	CoefVari
0.1000078	0.1350043	1000	0.1	1.349938
0.0999965	0.1349784	2000	0.1	1.349832
0.0999997	0.1349655	3000	0.1	1.349660
0.1000000	0.1349594	4000	0.1	1.349595
0.1000003	0.1349562	5000	0.1	1.349558
0.1000002	0.1349538	6000	0.1	1.349535
0.0999995	0.1349521	7000	0.1	1.349528

mean	stdev	n	EcActual	CoefVari
0.0999999	0.1349510	8000	0.1	1.349511
0.1000005	0.1349499	9000	0.1	1.349492
0.0999997	0.1349495	10000	0.1	1.349500

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



mean	stdev	n	EcActual	CoefVari
0.0100921	0.0267602	1000	0.01	2.651591
0.0098858	0.0260396	2000	0.01	2.634030
0.0100924	0.0265517	3000	0.01	2.630851
0.0100307	0.0265334	4000	0.01	2.645220
0.0099311	0.0261273	5000	0.01	2.630847
0.0099892	0.0263593	6000	0.01	2.638772
0.0100342	0.0265188	7000	0.01	2.642845
0.0099801	0.0263416	8000	0.01	2.639410
0.0100070	0.0263735	9000	0.01	2.635521
0.0100029	0.0263827	10000	0.01	2.637506