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)
}
```

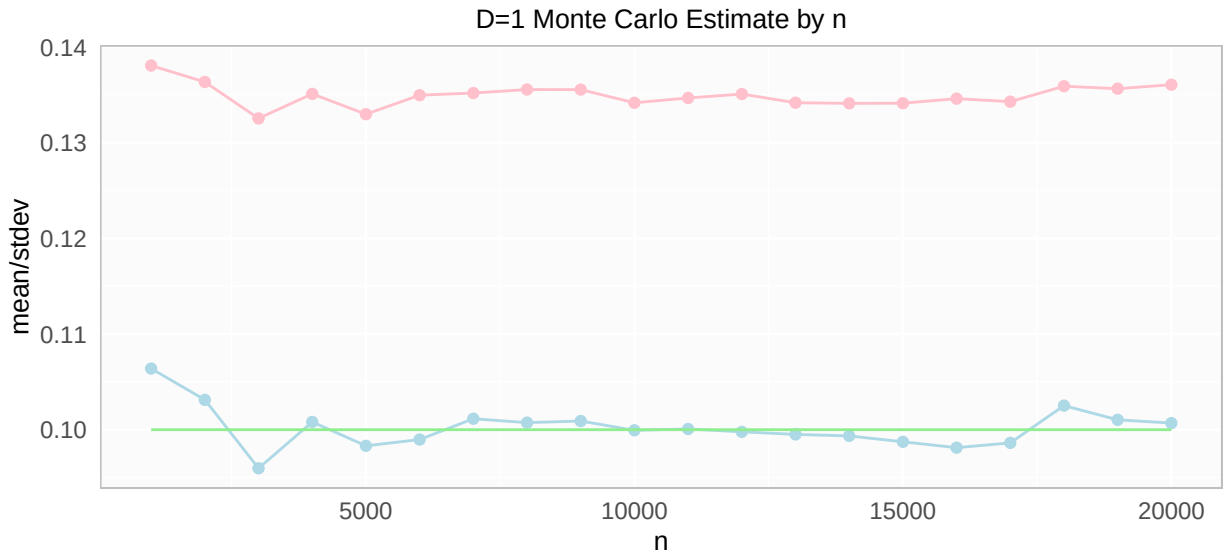
a) Crude Monte Carlo

```
crudeMC <- function(n, min, max, d = 1)
{
  theta.hat <- rep(NA, n)
  for(i in 1:n)
  {
    x <- runif(d, min, max)
    theta.hat[i] <- costFx(x)
  }

  return (theta.hat)
}

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

montecarlo.Loop <- function(d, fun, verbose=FALSE)
{
  crudeMc.result <- data.frame(mean=c(), stdev=c(), n=c())
```

```
  for(n in seq(1000, 20000, by=1000))
  {
    res <- fun(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 <- montecarlo.Loop(d=1, fun=crudeMC)
```

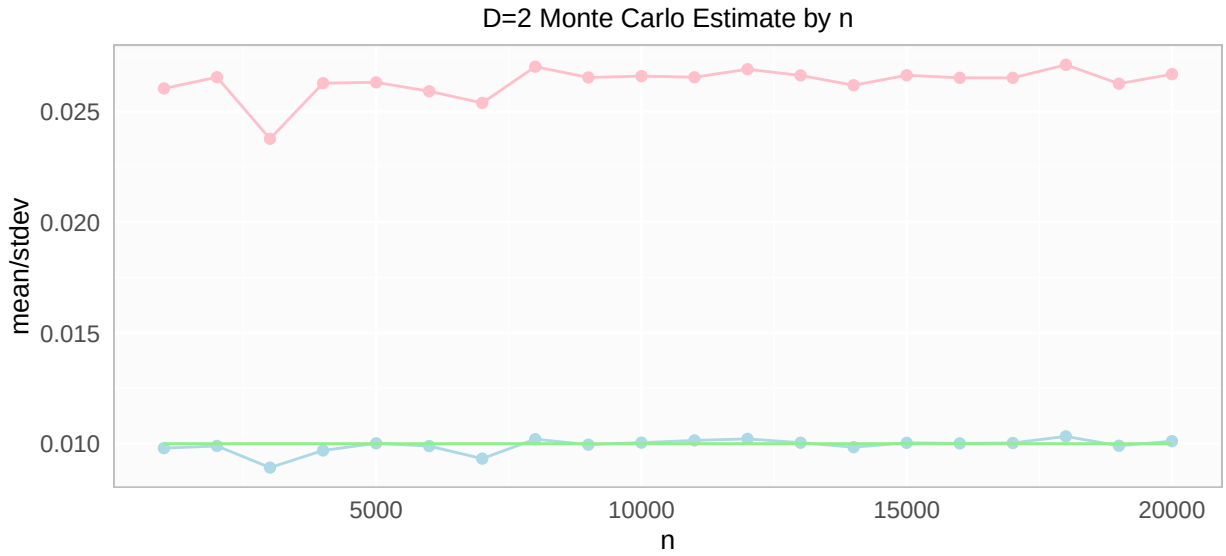


mean	stdev	n	EcActual	CoefVari
0.1063893	0.1380583	1000	0.1	1.297670
0.1031207	0.1363369	2000	0.1	1.322110
0.0959678	0.1325320	3000	0.1	1.381005
0.1008113	0.1350975	4000	0.1	1.340103
0.0983136	0.1329688	5000	0.1	1.352496
0.0989702	0.1349571	6000	0.1	1.363614
0.1011450	0.1351806	7000	0.1	1.336502
0.1007386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

mean	stdev	n	EcActual	CoefVari
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 <- montecarlo.Loop(d=2, fun=crudeMC, 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

b) Quasi-Random Numbers

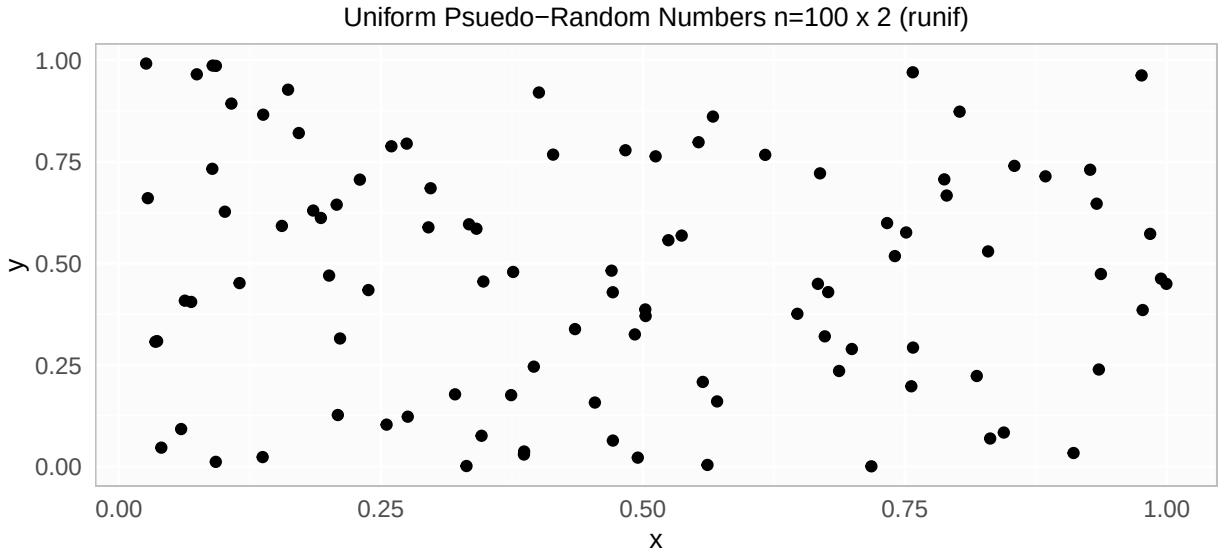
First, we compare the typical uniform random numbers from R's `runif` function to Sobol quasi-random numbers from `randtoolbox::sobol` function. 100 pairs of numbers are drawn from both generators and visualized below.

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
```

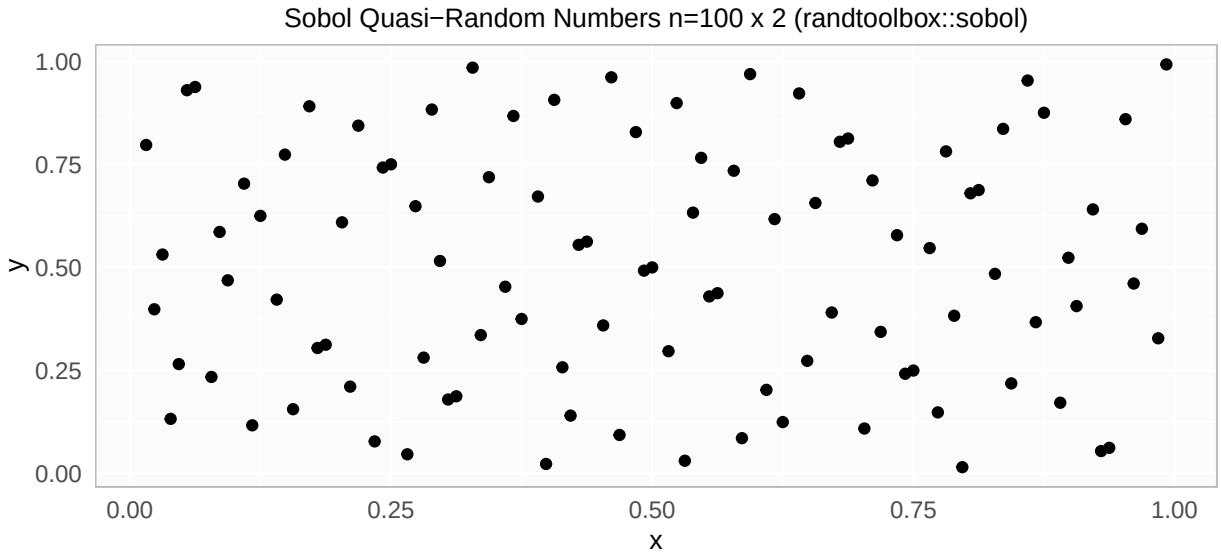


Sobol Random Numbers

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

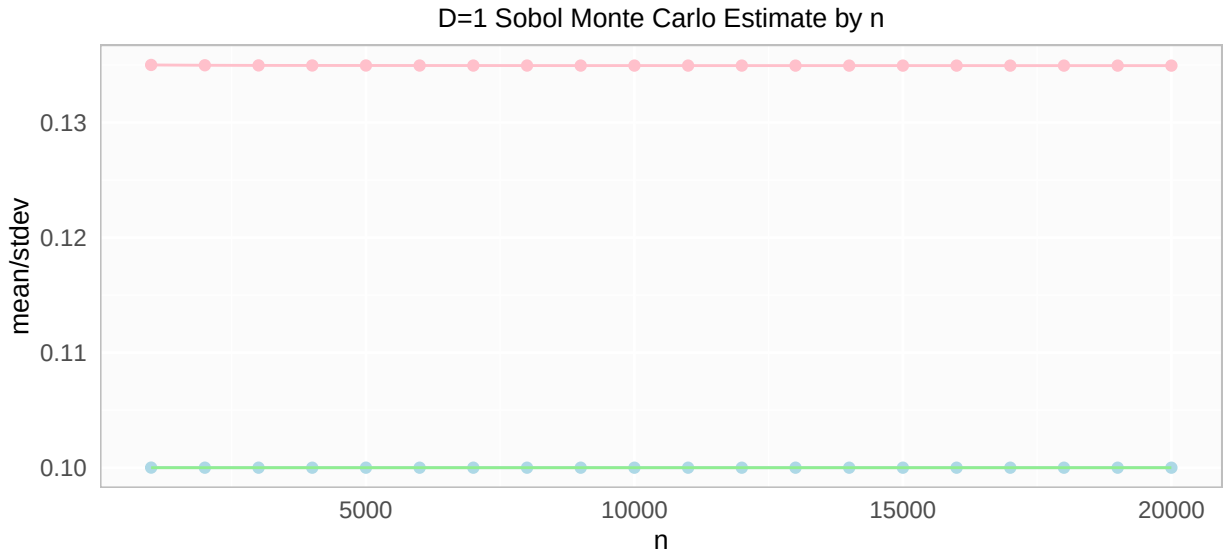
First, `sobol` based helper functions are defined using the same structure as the prior `runif` based functions. A couple of changes were needed:

- `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)
```

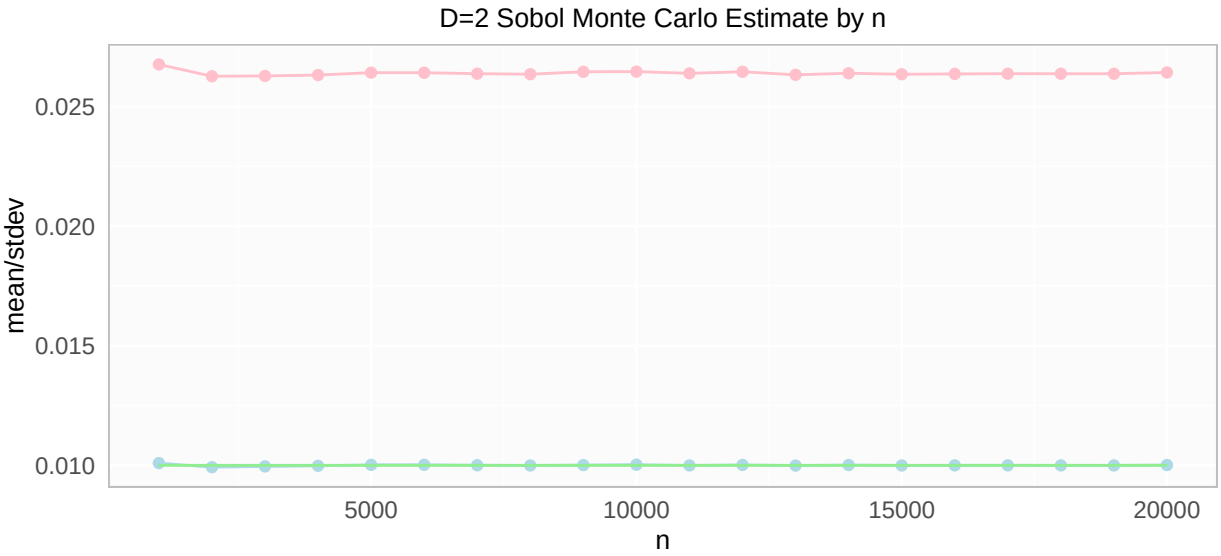
In the code below, we call the Sobol Monte Carlo loop function, show the top entries and visualize the result for D=1. Again, the blue line represents the mean value, pink is the standard deviation, and the green line is the analytical value for $E[c(x)] = (1/10)^D$.

```
sobolMc.D1 <- montecarlo.Loop(d=1, fun=sobolMC)
```



mean	stdev	n	EcActual	CoefVari
0.1000078	0.1350043	1000	0.1	1.349938
0.1000000	0.1349762	2000	0.1	1.349762
0.1000002	0.1349645	3000	0.1	1.349642
0.0999999	0.1349594	4000	0.1	1.349595
0.1000007	0.1349560	5000	0.1	1.349550
0.1000000	0.1349538	6000	0.1	1.349538
0.1000003	0.1349523	7000	0.1	1.349519
0.0999999	0.1349510	8000	0.1	1.349511
0.1000014	0.1349487	9000	0.1	1.349468
0.1000000	0.1349493	10000	0.1	1.349493
0.1000000	0.1349484	11000	0.1	1.349484
0.0999999	0.1349482	12000	0.1	1.349482
0.1000004	0.1349475	13000	0.1	1.349470
0.0999999	0.1349474	14000	0.1	1.349474
0.1000001	0.1349471	15000	0.1	1.349470
0.0999999	0.1349468	16000	0.1	1.349468
0.1000005	0.1349460	17000	0.1	1.349453
0.1000000	0.1349463	18000	0.1	1.349463
0.1000000	0.1349460	19000	0.1	1.349460
0.0999999	0.1349459	20000	0.1	1.349460

```
sobolMc.D2 <- montecarlo.Loop(d=2, fun=sobolMC)
```



mean	stdev	n	EcActual	CoefVari
0.0100921	0.0267602	1000	0.01	2.651591
0.0099232	0.0262662	2000	0.01	2.646959
0.0099546	0.0262777	3000	0.01	2.639755
0.0099800	0.0263159	4000	0.01	2.636868
0.0100208	0.0264157	5000	0.01	2.636098
0.0100235	0.0264129	6000	0.01	2.635097
0.0100033	0.0263734	7000	0.01	2.636471
0.0099940	0.0263523	8000	0.01	2.636799
0.0100110	0.0264525	9000	0.01	2.642330
0.0100264	0.0264598	10000	0.01	2.639022
0.0099956	0.0263874	11000	0.01	2.639900
0.0100174	0.0264517	12000	0.01	2.640572
0.0099898	0.0263261	13000	0.01	2.635296
0.0100134	0.0263908	14000	0.01	2.635543
0.0099946	0.0263486	15000	0.01	2.636278
0.0099994	0.0263634	16000	0.01	2.636496
0.0100000	0.0263737	17000	0.01	2.637361
0.0099976	0.0263709	18000	0.01	2.637718
0.0099966	0.0263707	19000	0.01	2.637966
0.0100131	0.0264233	20000	0.01	2.638879

Compare Pure Uniform and Sobol

From what I see, the average of the sobol-based approach is closer to the expected value vs the more pure uniform approach. On the otherhand, the standard deviations are generally similar, though the sobol stdevs tend to drive to ≈ 0.13494 as whereas the uniform approach ranges 0.134 to 0.136 (a broader range). In other words, the sobol approach seems to result is a reduced variance of the variance.

```
comparedMc.D1 <- data.frame(crudeMc.D1$mean,
                            sobolMc.D1$mean,
                            meanDiff=crudeMc.D1$mean - sobolMc.D1$mean,
                            crudeMc.D1$stdev,
                            sobolMc.D1$stdev,
                            stdevDiff=crudeMc.D1$stdev - sobolMc.D1$stdev,
                            n=crudeMc.D1$n)
kable(comparedMc.D1)
```

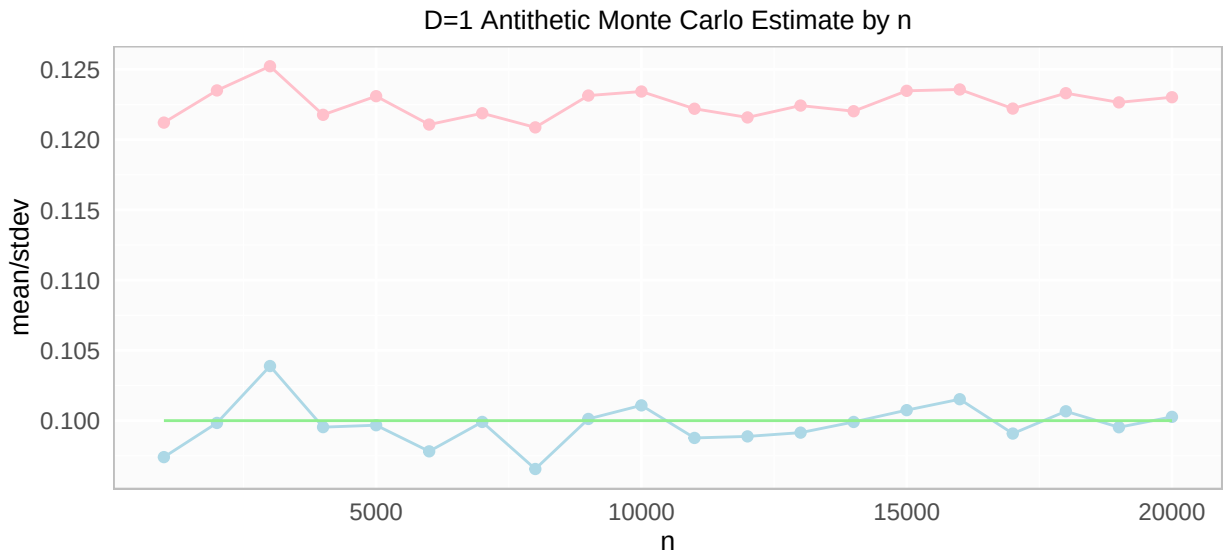
crudeMc.D1.mean	sobolMc.D1.mean	meanDiff	crudeMc.D1.stdev	sobolMc.D1.stdev	stdevDiff	n
0.1063893	0.1000078	0.0063815	0.1380583	0.1350043	0.0030539	1000
0.1031207	0.1000000	0.0031207	0.1363369	0.1349762	0.0013606	2000
0.0959678	0.1000002	-0.0040325	0.1325320	0.1349645	-0.0024325	3000
0.1008113	0.0999999	0.0008113	0.1350975	0.1349594	0.0001381	4000
0.0983136	0.1000007	-0.0016871	0.1329688	0.1349560	-0.0019872	5000
0.0989702	0.1000000	-0.0010298	0.1349571	0.1349538	0.0000033	6000
0.1011450	0.1000003	0.0011447	0.1351806	0.1349523	0.0002282	7000
0.1007386	0.0999999	0.0007387	0.1355480	0.1349510	0.0005970	8000
0.1009048	0.1000014	0.0009034	0.1355401	0.1349487	0.0005914	9000
0.0999374	0.1000000	-0.0000626	0.1341640	0.1349493	-0.0007853	10000
0.1000755	0.1000000	0.0000755	0.1346675	0.1349484	-0.0002809	11000
0.0997761	0.0999999	-0.0002239	0.1350776	0.1349482	0.0001294	12000
0.0995073	0.1000004	-0.0004931	0.1341704	0.1349475	-0.0007771	13000
0.0993517	0.0999999	-0.0006482	0.1341042	0.1349474	-0.0008432	14000
0.0987340	0.1000001	-0.0012661	0.1341199	0.1349471	-0.0008272	15000
0.0981232	0.0999999	-0.0018768	0.1345956	0.1349468	-0.0003512	16000
0.0986219	0.1000005	-0.0013786	0.1342821	0.1349460	-0.0006639	17000
0.1025172	0.1000000	0.0025173	0.1359001	0.1349463	0.0009538	18000
0.1010334	0.1000000	0.0010334	0.1356360	0.1349460	0.0006901	19000
0.1006939	0.0999999	0.0006939	0.1360526	0.1349459	0.0011067	20000

c) Antithetic Variates

```
antitheticMC <- function(n, min, max, d = 1)
{
  theta.hat <- rep(NA, n)
  for(i in 1:n)
  {
    x <- runif(d, min, max)
    x2 <- 1 - x

    theta.hat[i] <- (costFx(x) + costFx(x2) ) / 2
  }

  return (theta.hat)
}

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

First we do the D=1 scenario:

```
antitheticMc.D1 <- montecarlo.Loop(d=1, fun=antitheticMC)
```

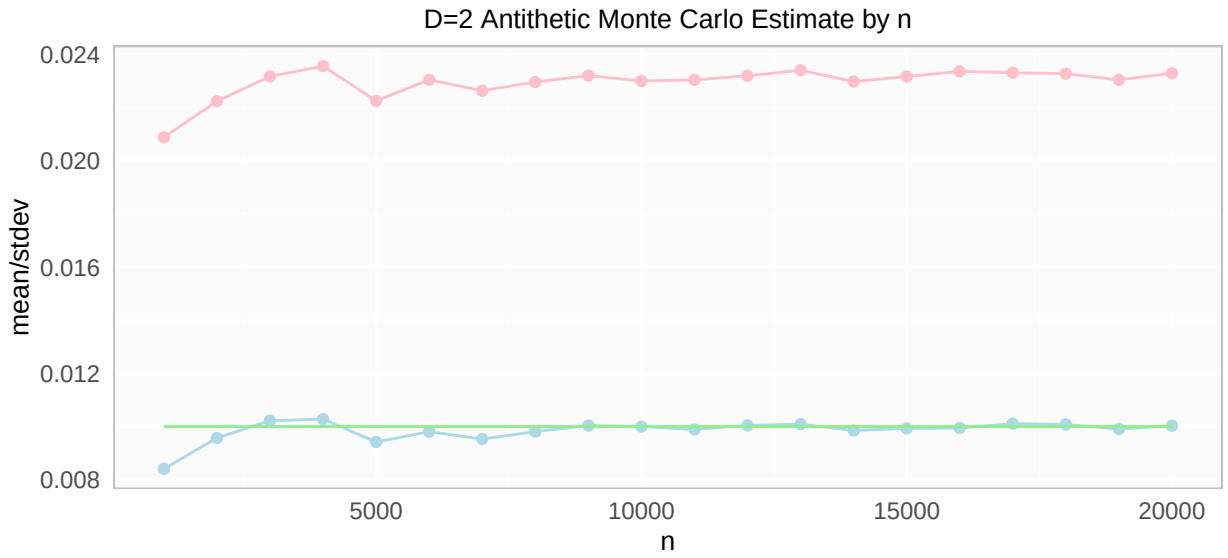


mean	stdev	n	EcActual	CoefVari
0.0974034	0.1212140	1000	0.1	1.244454
0.0998374	0.1235045	2000	0.1	1.237057
0.1038854	0.1252266	3000	0.1	1.205431
0.0995391	0.1217680	4000	0.1	1.223318
0.0996740	0.1230949	5000	0.1	1.234975
0.0978149	0.1210755	6000	0.1	1.237803
0.0999068	0.1218799	7000	0.1	1.219936
0.0965583	0.1208741	8000	0.1	1.251825
0.1001205	0.1231400	9000	0.1	1.229917
0.1010866	0.1234271	10000	0.1	1.221003
0.0987690	0.1222010	11000	0.1	1.237240
0.0988780	0.1215830	12000	0.1	1.229628
0.0991463	0.1224262	13000	0.1	1.234803
0.0999053	0.1220252	14000	0.1	1.221409

mean	stdev	n	EcActual	CoefVari
0.1007474	0.1234773	15000	0.1	1.225613
0.1015242	0.1235729	16000	0.1	1.217176
0.0990793	0.1222117	17000	0.1	1.233474
0.1006663	0.1233080	18000	0.1	1.224919
0.0995291	0.1226538	19000	0.1	1.232342
0.1002695	0.1230203	20000	0.1	1.226897

Next we do the D=2 scenario with the antithetic function:

```
antitheticMc.D2 <- montecarlo.Loop(d=2, fun=antitheticMC)
```



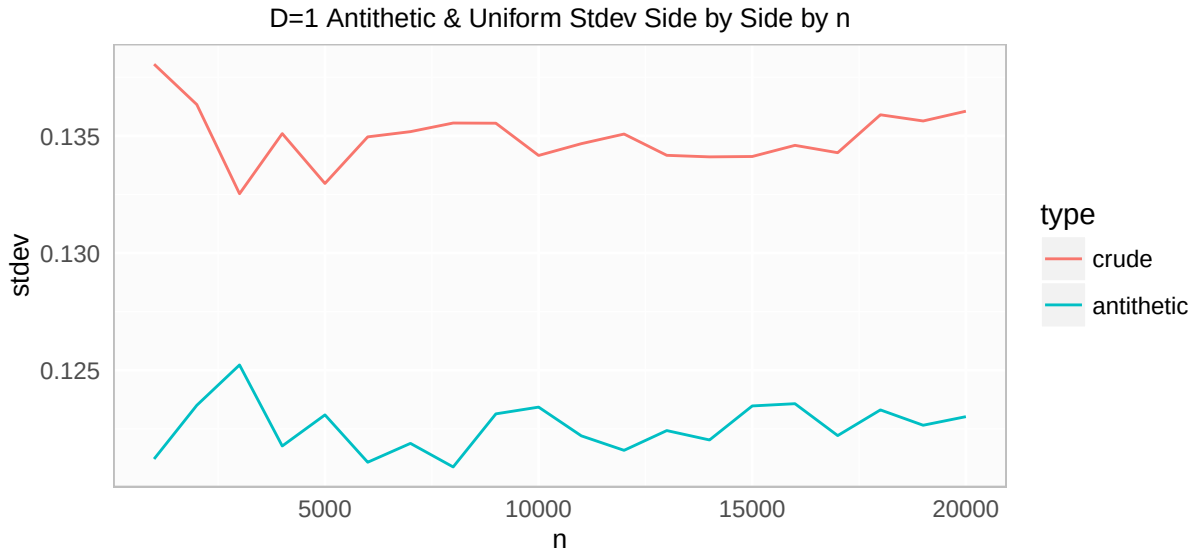
mean	stdev	n	EcActual	CoefVari
0.0084097	0.0208781	1000	0.01	2.482626
0.0095689	0.0222353	2000	0.01	2.323706
0.0102194	0.0231648	3000	0.01	2.266744
0.0102795	0.0235568	4000	0.01	2.291628
0.0094190	0.0222485	5000	0.01	2.362100
0.0098075	0.0230423	6000	0.01	2.349455
0.0095280	0.0226285	7000	0.01	2.374948
0.0098109	0.0229586	8000	0.01	2.340118
0.0100363	0.0232003	9000	0.01	2.311628
0.0099968	0.0229959	10000	0.01	2.300319
0.0098925	0.0230386	11000	0.01	2.328901
0.0100434	0.0231907	12000	0.01	2.309060
0.0100898	0.0234002	13000	0.01	2.319198
0.0098533	0.0229743	14000	0.01	2.331623
0.0099288	0.0231615	15000	0.01	2.332765
0.0099434	0.0233584	16000	0.01	2.349136
0.0101092	0.0233057	17000	0.01	2.305399
0.0100780	0.0232712	18000	0.01	2.309097
0.0099062	0.0230379	19000	0.01	2.325599
0.0100304	0.0232857	20000	0.01	2.321502

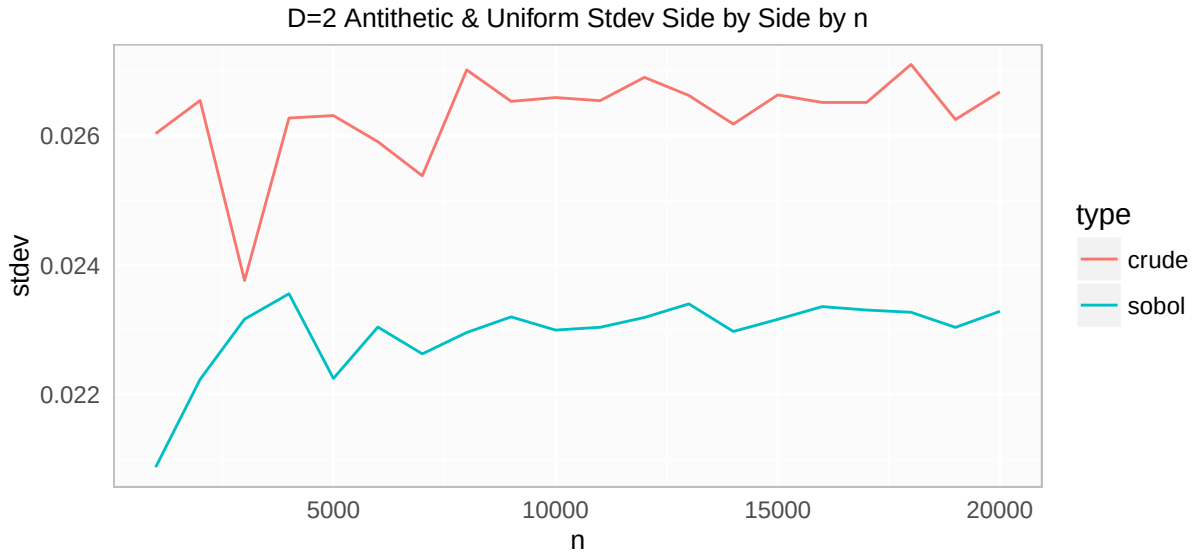
Compare Pure Uniform and Antithetic

The antithetic scenario produces a smaller standard deviation (and therefore variance) in both the D=1 and D=2 cases. The visualizations below illustrate the point. The avarege values are generally the same, centering on the expected value.

```
comparedMc2.D1 <- data.frame(crudeMc.D1$mean,
                             antitheticMc.D1$mean,
                             meanDiff=crudeMc.D1$mean - antitheticMc.D1$mean,
                             crudeMc.D1$stdev,
                             antitheticMc.D1$stdev,
                             stdevDiff=crudeMc.D1$stdev - antitheticMc.D1$stdev,
                             n=crudeMc.D1$n)
kable(comparedMc2.D1)
```

crudeMc.D1.mean	antitheticMc.D1.mean	meanDiff	crudeMc.D1.stdev	antitheticMc.D1.stdev	stdevDiff	n
0.1063893	0.0974034	0.0089859	0.1380583	0.1212140	0.0168442	1000
0.1031207	0.0998374	0.0032833	0.1363369	0.1235045	0.0128324	2000
0.0959678	0.1038854	-0.0079176	0.1325320	0.1252266	0.0073054	3000
0.1008113	0.0995391	0.0012722	0.1350975	0.1217680	0.0133295	4000
0.0983136	0.0996740	-0.0013604	0.1329688	0.1230949	0.0098739	5000
0.0989702	0.0978149	0.0011553	0.1349571	0.1210755	0.0138816	6000
0.1011450	0.0999068	0.0012382	0.1351806	0.1218799	0.0133007	7000
0.1007386	0.0965583	0.0041803	0.1355480	0.1208741	0.0146739	8000
0.1009048	0.1001205	0.0007843	0.1355401	0.1231400	0.0124001	9000
0.0999374	0.1010866	-0.0011492	0.1341640	0.1234271	0.0107369	10000
0.1000755	0.0987690	0.0013065	0.1346675	0.1222010	0.0124665	11000
0.0997761	0.0988780	0.0008981	0.1350776	0.1215830	0.0134946	12000
0.0995073	0.0991463	0.0003610	0.1341704	0.1224262	0.0117442	13000
0.0993517	0.0999053	-0.0005535	0.1341042	0.1220252	0.0120790	14000
0.0987340	0.1007474	-0.0020135	0.1341199	0.1234773	0.0106426	15000
0.0981232	0.1015242	-0.0034011	0.1345956	0.1235729	0.0110227	16000
0.0986219	0.0990793	-0.0004573	0.1342821	0.1222117	0.0120704	17000
0.1025172	0.1006663	0.0018510	0.1359001	0.1233080	0.0125920	18000
0.1010334	0.0995291	0.0015043	0.1356360	0.1226538	0.0129822	19000
0.1006939	0.1002695	0.0004244	0.1360526	0.1230203	0.0130323	20000





d) Latin Hypercube Sampling

Well, first we define a function for the hypercube V_k:

```
hyperV <- function(p, U, K)
{
  return ((p + 1 - U) / K)
}
```

```
latinHypercubeMC <- function(n, min, max, d = 1, K=5)
{
  theta.hat <- rep(NA, n)
  for(i in 1:n)
  {

    U <- matrix(runif(d*K, 0, 1), nrow=d)
    TT <- matrix(rep(NA, K*d), nrow=d)
    for(j in 1:d)
    {
      TT[j,] <- sample(1:K, size=K)
    }

    Vk <- hyperV(TT, U, K)

    #print(Vk)

    theta.hat[i] <- mean(costFx(Vk))
  }

  return (theta.hat)
}

ret <- latinHypercubeMC(10, -5, 5, 2)
ret
```

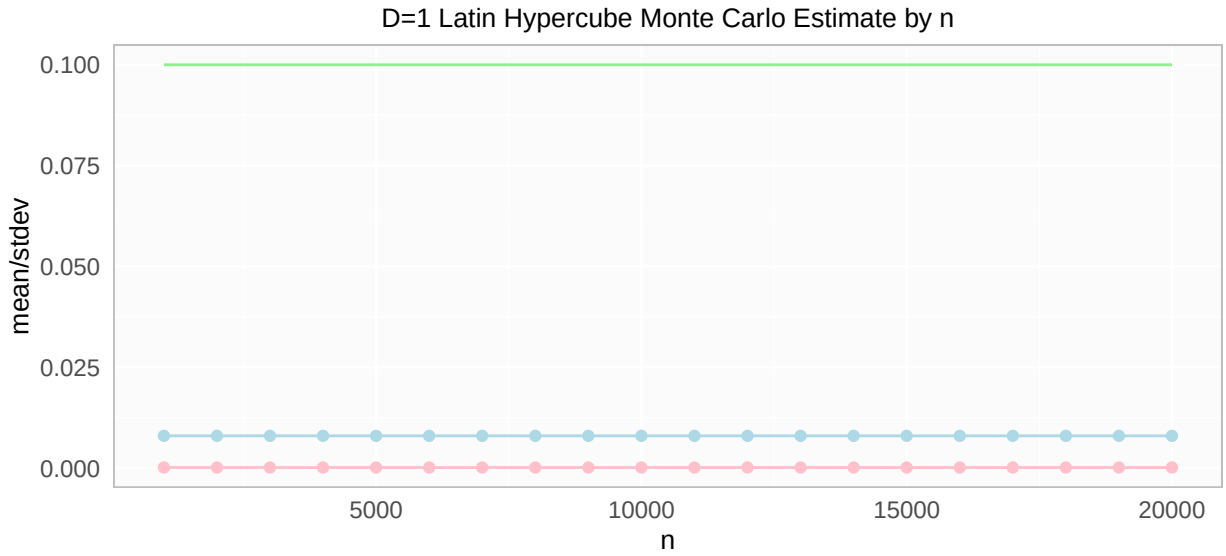
```
##  [1] 6.212177e-05 6.630558e-05 6.300821e-05 6.467318e-05 6.160394e-05
##  [6] 6.405027e-05 6.322628e-05 6.496521e-05 6.073935e-05 6.259709e-05
```

```
mean(ret)
```

```
## [1] 6.332909e-05
```

First we do the D=1 scenario:

```
latinHypercubeMc.D1 <- montecarlo.Loop(d=1, fun=latinHypercubeMC)
```



mean	stdev	n	EcActual	CoefVari
0.0079994	0.0001418	1000	0.1	0.0177313
0.0079932	0.0001419	2000	0.1	0.0177541
0.0079954	0.0001408	3000	0.1	0.0176052
0.0079928	0.0001391	4000	0.1	0.0174001
0.0079914	0.0001387	5000	0.1	0.0173502
0.0079913	0.0001397	6000	0.1	0.0174777
0.0079926	0.0001392	7000	0.1	0.0174135
0.0079939	0.0001409	8000	0.1	0.0176235
0.0079915	0.0001404	9000	0.1	0.0175625
0.0079925	0.0001390	10000	0.1	0.0173947
0.0079927	0.0001370	11000	0.1	0.0171452
0.0079919	0.0001405	12000	0.1	0.0175752
0.0079926	0.0001394	13000	0.1	0.0174439
0.0079917	0.0001396	14000	0.1	0.0174721
0.0079944	0.0001389	15000	0.1	0.0173729
0.0079925	0.0001396	16000	0.1	0.0174677
0.0079924	0.0001401	17000	0.1	0.0175319
0.0079929	0.0001400	18000	0.1	0.0175125
0.0079930	0.0001402	19000	0.1	0.0175427
0.0079913	0.0001389	20000	0.1	0.0173862