Homework #4

J. Hamski

10/13/2016

```
library(ggplot2)
library(dplyr)
library(knitr)
library(grid)
library(gridExtra)
library(dplyr)
```

```
set.seed(1272)
```

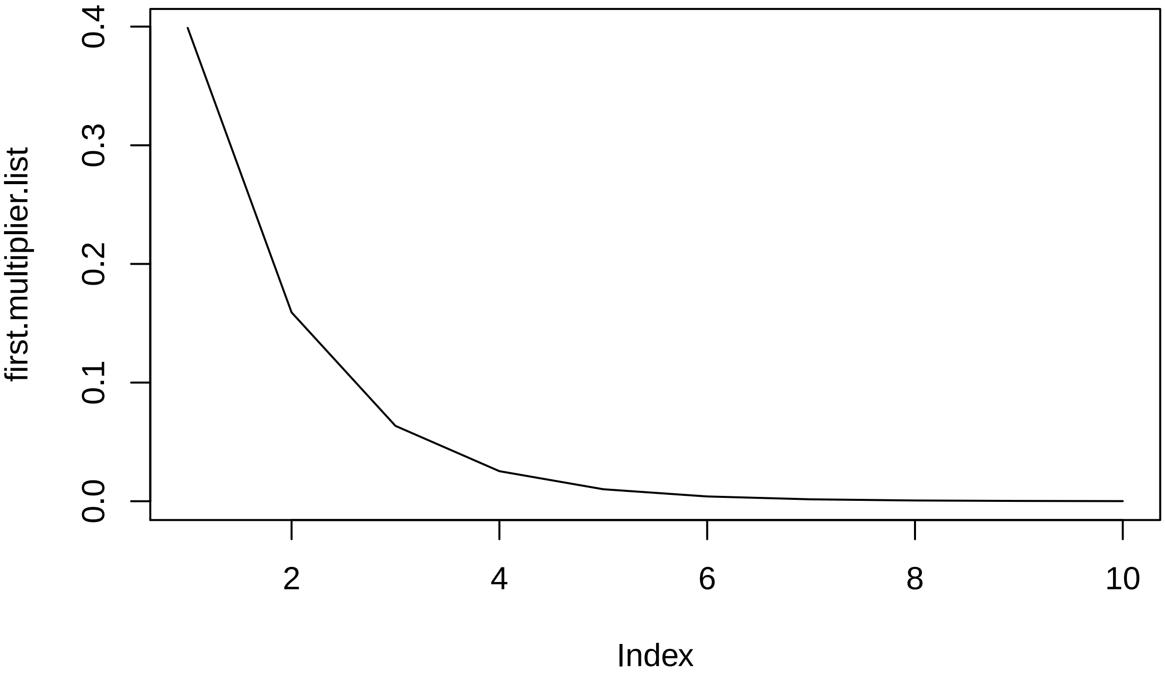
Variance reduction procedures for Monte Carlo methods

Preliminary Setup

The multiplier decreases as D increases.

```
first.multiplier <- function(D){
  return(1/((2*pi)^(D/2)))
}
```

```
D.seq <- seq(1, from=1, to=10)

first.multiplier.list <- sapply(D.seq, FUN=first.multiplier)

plot(first.multiplier.list,  type = "l")
```



Testing the exponent function.

```
exponent.function <- function(x){
  return((-1/2)*(t(x)%*%x))
}
n <- 10
D <- 2
x <- matrix(runif(D*n, min = -5, max = 5), ncol=D)

exponent.function(x)
```

```
##            [,1]       [,2]
## [1,] -62.170918  -1.373319
## [2,]  -1.373319 -30.872275
```

These don't really need to be broken out into different functions, but I found it helpful to try different inputs and see how the components of the function behave.

a) Crude Monte Carlo

First, create a function to evaluate c(x) at N samples in D dimensions (i.e., a DxN matrix).

```
cost.function.crude <- function(x, D){
  first.multiplier.sim <- first.multiplier(D)
  exponent.sim <- exponent.function(x)
  return(first.multiplier.sim * exp(exponent.sim))
}
```

```
n <- 1
D <- 2
x <- matrix(runif(D*n, min = -5, max = 5), nrow=n, ncol=D)
```

```
cost.function.crude.apply <- function(x, D){
  return(mean(apply(x, 1, FUN = cost.function.crude, D=D)))
}
cost.function.crude.apply(x, D)
```

```
## [1] 9.249131e-08
```

This appears to work (the mean is always close to 1/10). Now to move on to the crude Monte Carlo simulation of the cost function for n sizes from 1000 to 10,000 by increments of 1000.

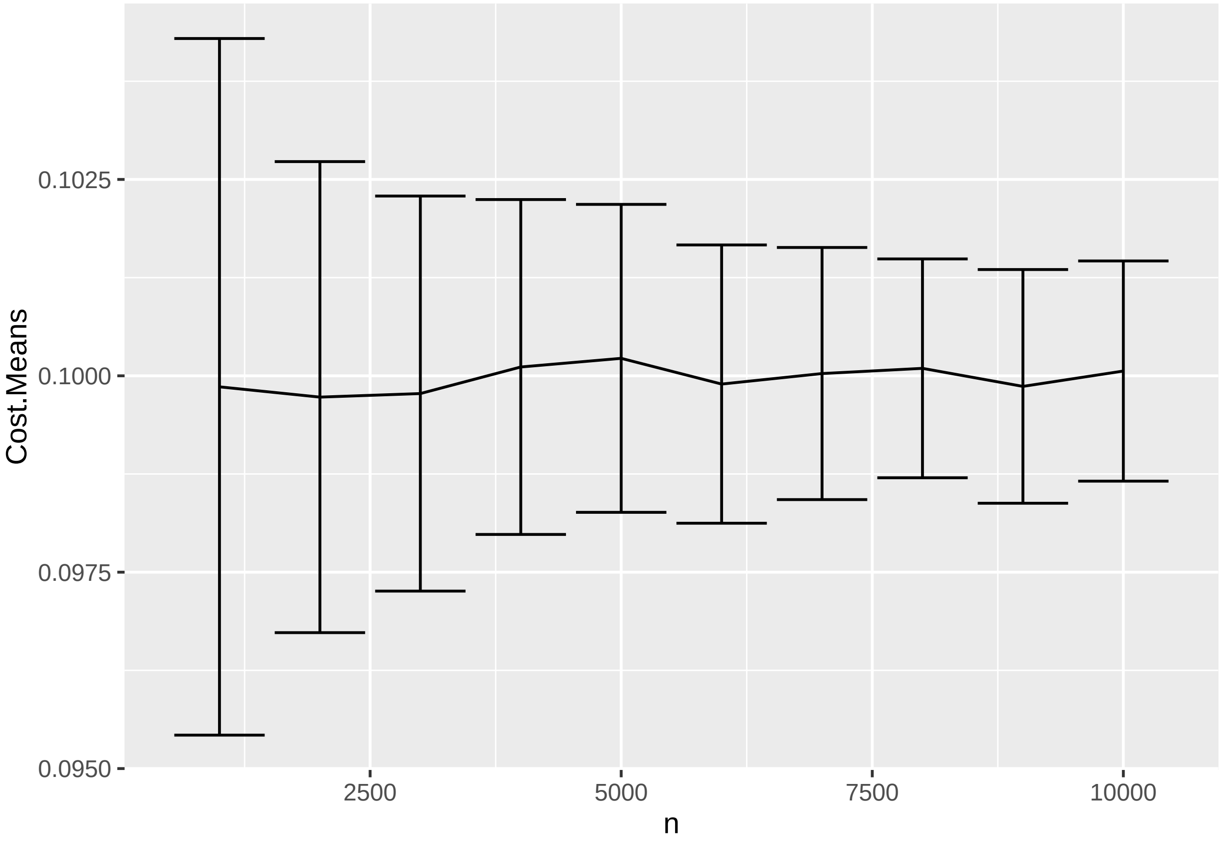
```
n.increments <- seq(from = 1000, to = 10000, by = 1000)

generate.x.list <- function(n, D){
  x <- matrix(runif(D*n, min = -5, max = 5), nrow=n, ncol=D)
  return(x)
}

x.list <- lapply(n.increments, FUN = generate.x.list, D=D)
```

For D = 1:

```
D.1 <- 1
crude.cost.sim.1 <- replicate(100, sapply(lapply(n.increments, FUN = generate.x.list, D=D.1), FUN = cos
```

```
crude.cost.sim.1.means <- apply(crude.cost.sim.1, 1, FUN = mean)
crude.cost.sim.1.sd <- apply(crude.cost.sim.1, 1, FUN = sd)
n <- n.increments
crude.cost.sim.1.results <- cbind(n, crude.cost.sim.1.means, crude.cost.sim.1.sd) %>% as.data.frame()
colnames(crude.cost.sim.1.results) <- c("n", "Cost.Means", "Cost.Standard.Deviation")
```

```
limits <- aes(ymax = Cost.Means + Cost.Standard.Deviation, ymin = Cost.Means - Cost.Standard.Deviation)
ggplot(crude.cost.sim.1.results, aes(x=n, y=Cost.Means)) + geom_line() + geom_errorbar(limits)
```



```
crude.cost.sim.1.results <- crude.cost.sim.1.results %>%
  mutate(Cost.Coeff.Varation = Cost.Standard.Deviation / Cost.Means)
```

```
kable(crude.cost.sim.1.results)
```

n	Cost.Means	Cost.Standard.Deviation	Cost.Coeff.Varation
1000	0.0998597	0.0044341	0.0444035
2000	0.0997287	0.0029983	0.0300650
3000	0.0997740	0.0025148	0.0252047
4000	0.1001121	0.0021317	0.0212933

n	Cost.Means	Cost.Standard.Deviation	Cost.Coeff.Varation
5000	0.1002223	0.0019601	0.0195579
6000	0.0998947	0.0017714	0.0177327
7000	0.1000289	0.0016050	0.0160452
8000	0.1000950	0.0013931	0.0139181
9000	0.0998653	0.0014874	0.0148941
10000	0.1000606	0.0014016	0.0140078

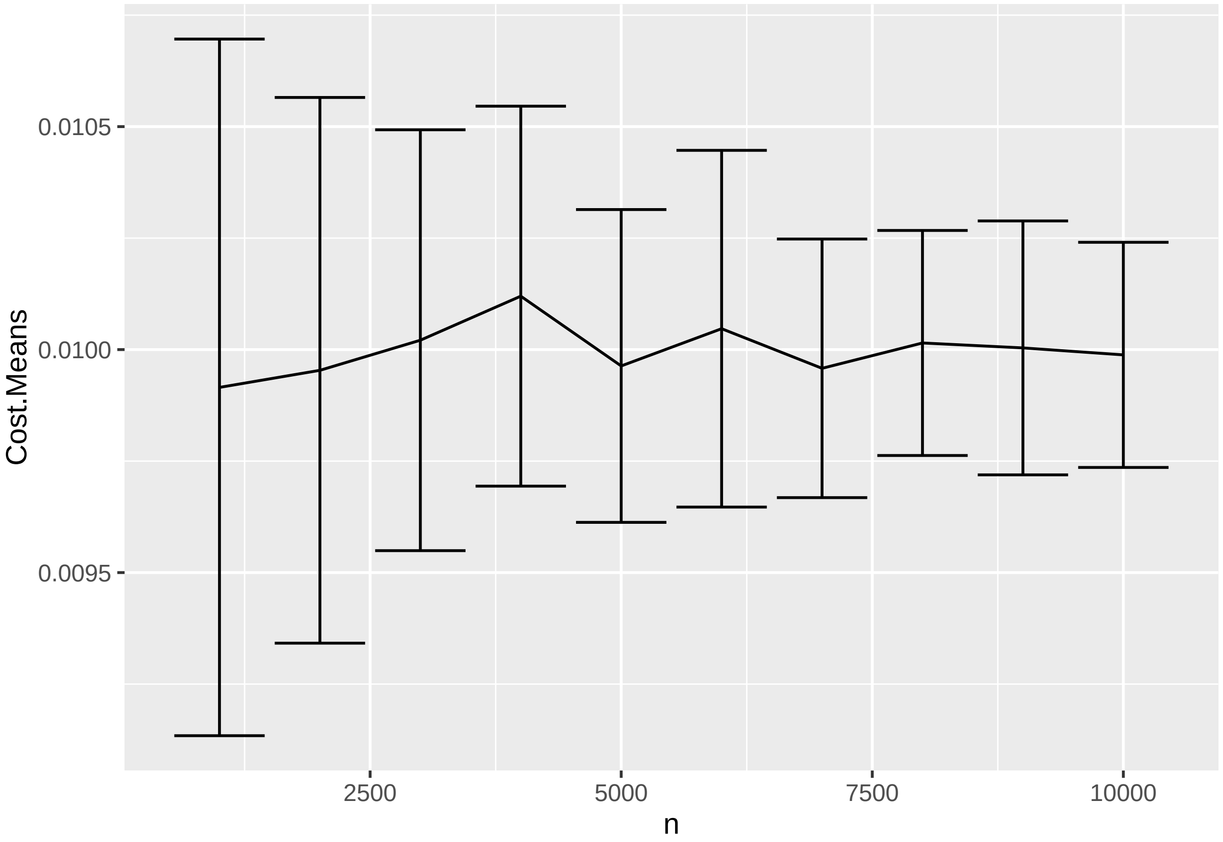
Now for D = 2:

```
D.2 = 2
crude.cost.sim.2 <- replicate(100, sapply(lapply(n.increments, FUN = generate.x.list, D = D.2), FUN = co

(target.D.2 <- (1/10)^2)
```

```
## [1] 0.01
```

```
crude.cost.sim.2.means <- apply(crude.cost.sim.2, 1, FUN = mean)
crude.cost.sim.2.sd <- apply(crude.cost.sim.2, 1, FUN = sd)

crude.cost.sim.2.results <- cbind(n, crude.cost.sim.2.means, crude.cost.sim.2.sd) %>% as.data.frame()
colnames(crude.cost.sim.2.results) <- c("n", "Cost.Means", "Cost.Standard.Deviation")
```

```
limits.2 <- aes(ymax = Cost.Means + Cost.Standard.Deviation, ymin = Cost.Means - Cost.Standard.Deviation
ggplot(crude.cost.sim.2.results, aes(x=n, y=Cost.Means)) + geom_line() + geom_errorbar(limits.2)
```



```r
crude.cost.sim.2.results <- crude.cost.sim.2.results %>%
  mutate(Cost.Coeff.Varation = Cost.Standard.Deviation / Cost.Means)
```

```r
kable(crude.cost.sim.2.results)
```

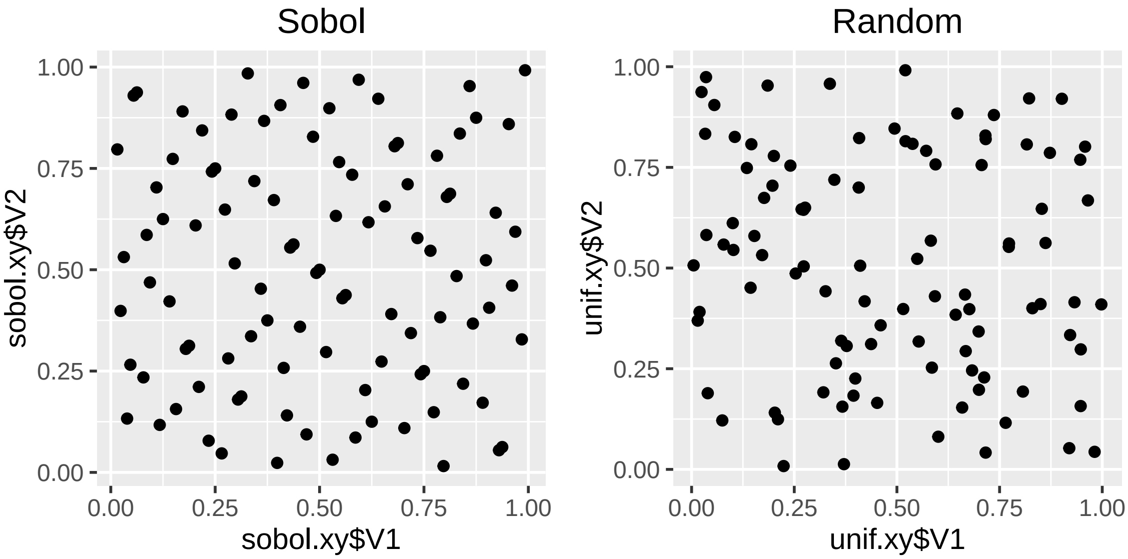
n	Cost.Means	Cost.Standard.Deviation	Cost.Coeff.Varation
1000	0.0099153	0.0007810	0.0787692
2000	0.0099536	0.0006118	0.0614684
3000	0.0100210	0.0004719	0.0470890
4000	0.0101198	0.0004260	0.0420941
5000	0.0099633	0.0003507	0.0351998
6000	0.0100468	0.0004000	0.0398108
7000	0.0099580	0.0002900	0.0291183
8000	0.0100148	0.0002523	0.0251928
9000	0.0100038	0.0002847	0.0284593
10000	0.0099882	0.0002524	0.0252733

b) Quasi-Random Numbers

Here, I use the package ‘randtoolbox’ to generate a Sobol sequence.

```r
library(randtoolbox)
```

```r
n = 100
sobol.xy <- sobol(n, dim = 2) %>% as.data.frame()
sobol <- qplot(sobol.xy$V1, sobol.xy$V2, main="Sobol")

unif.xy <- cbind(runif(n), runif(n)) %>% as.data.frame()
rand <- qplot(unif.xy$V1, unif.xy$V2, main = "Random")

grid.arrange(sobol, rand, nrow=1)
```



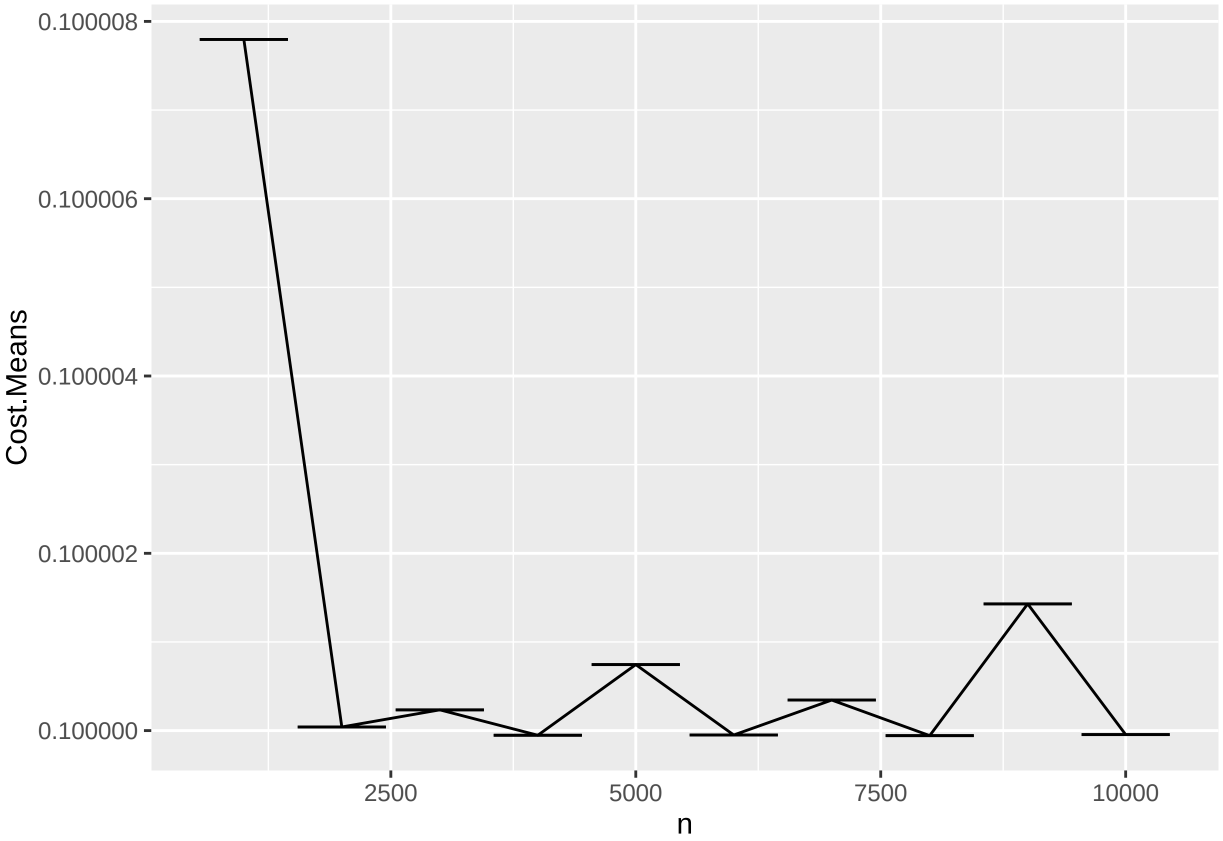
The random numbers (uniform distribution) form clusters where there are several random variables with close values and "dead space" where there are no random variables with the values.

The difference between the two is their discrepency. The Sobol numbers are low discrepency but are "quasi-random" - the enforcement of even distribution means that the values are not i.i.d. and therefore not truly random. The uniform distribution random numbers have the potential to be high discrepancy.

For D = 1:

```r
generate.x.list.Sobol <- function(n, D){
  sobol.seq <- (sobol(n, dim = D) * 10) - 5
  x <- as.matrix(sobol.seq)
  return(x)
}
```

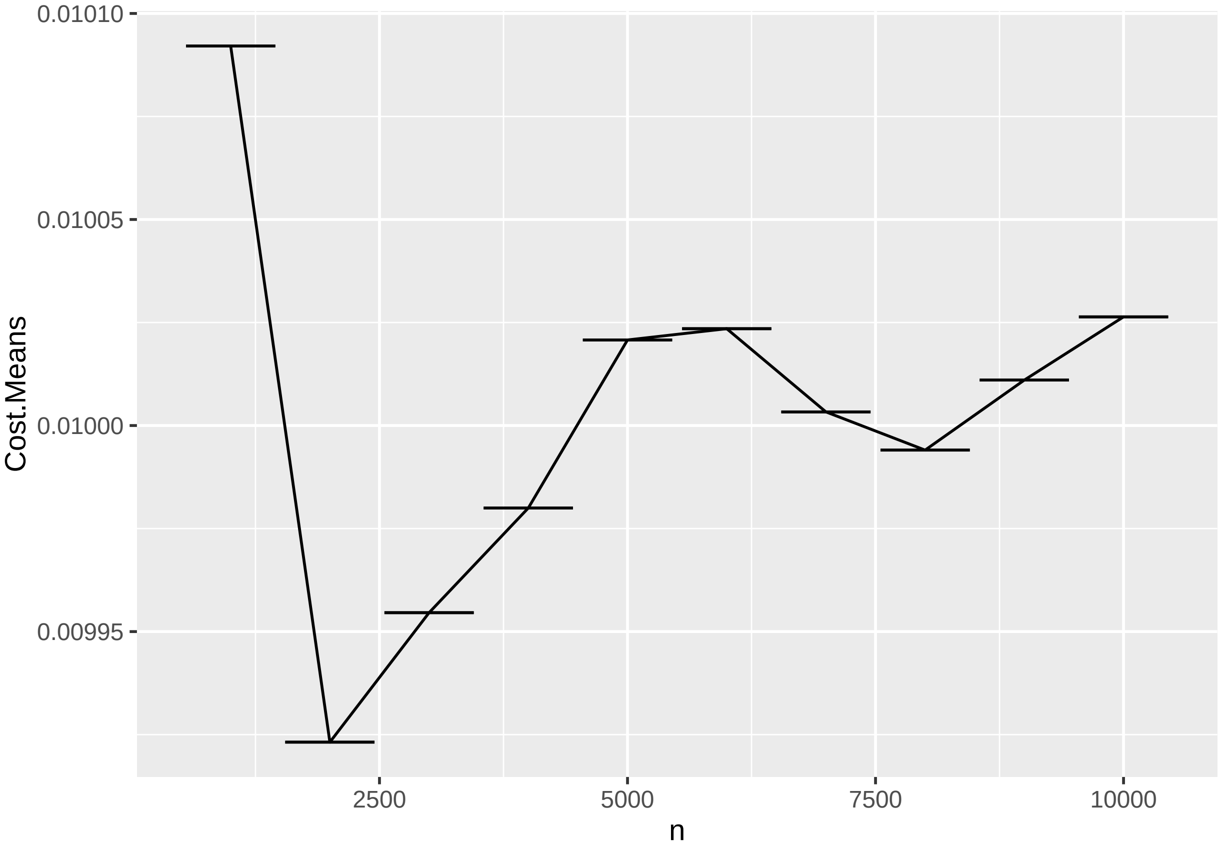
```r
D.1 <- 1
sobol.cost.sim.1 <- replicate(100, sapply(lapply(n.increments, FUN = generate.x.list.Sobol, D=D.1), FUN
```



n	Cost.Means	Cost.Standard.Deviation	Cost.Coeff.Varation
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

n	Cost.Means	Cost.Standard.Deviation	Cost.Coeff.Varation
9000	0.1000014	0	0
10000	0.1000000	0	0

For D = 2:



n	Cost.Means	Cost.Standard.Deviation	Cost.Coeff.Varation
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

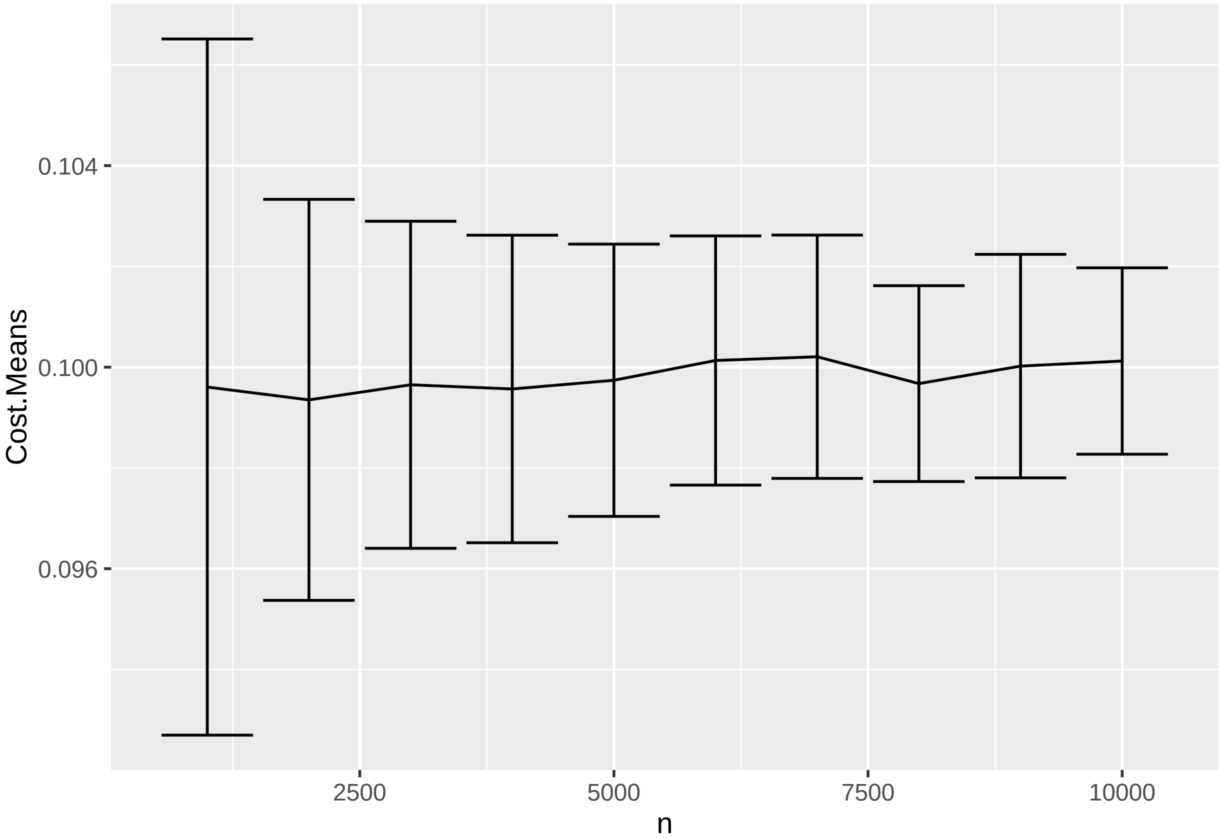
c) Antithetic Variates

```
generate.x.list.antithetic <- function(n, D){
  x <- matrix(runif(D*n, min = 0, max = 1), nrow=n, ncol=D)
  for(i in 1:nrow(x)){
    ifelse(i %% 2 == 0, x[i,] <- 1 - x[i-1,], next)
  }
  x <- (x * 10) - 5
  return(x)
}
```

```
generate.x.list.antithetic(4, 3)
```

```
##            [,1]       [,2]       [,3]
## [1,] -2.1348339  2.4321640 -0.4028483
## [2,]  2.1348339 -2.4321640  0.4028483
## [3,] -0.7463661 -0.5485907 -3.5443062
## [4,]  0.7463661  0.5485907  3.5443062
```

```
D.1 <- 1
antithetic.cost.sim.1 <- replicate(100, sapply(lapply(n.increments, FUN = generate.x.list.antithetic, D
```

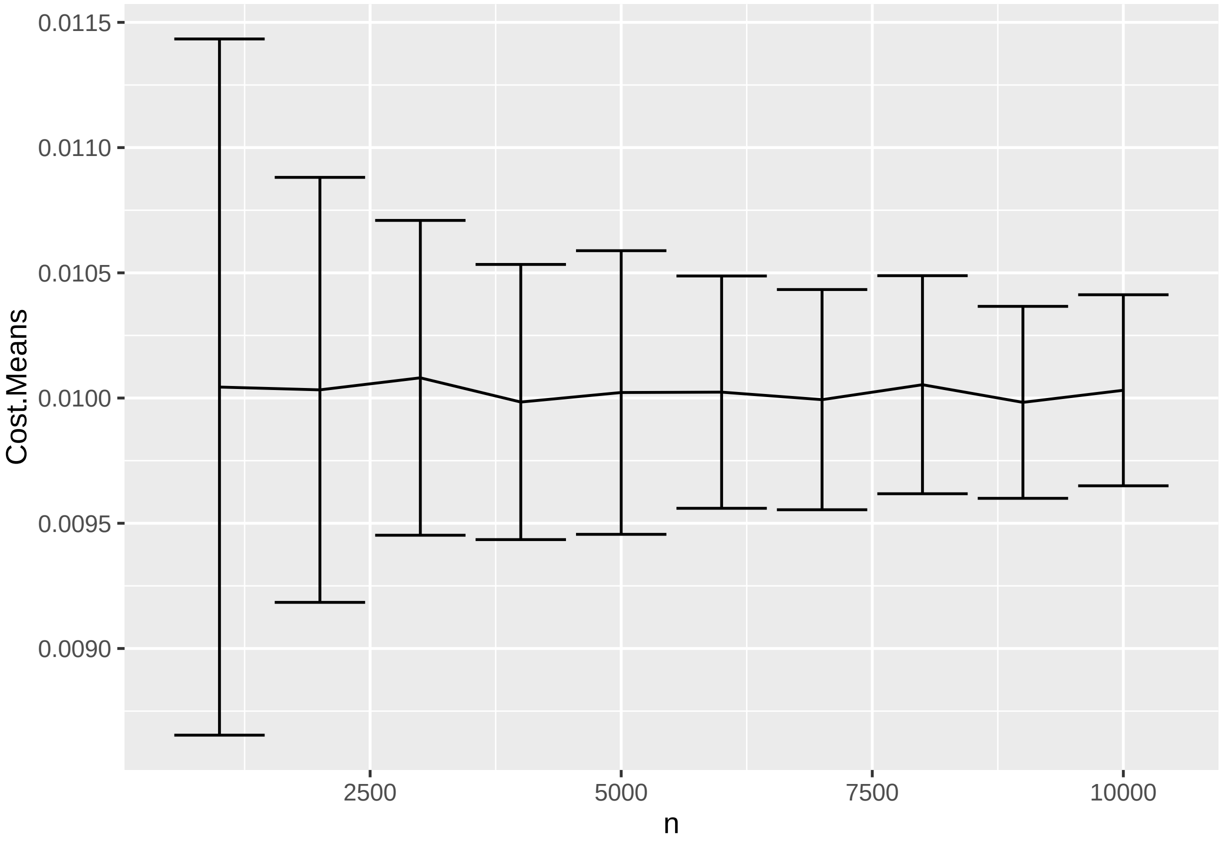


n	Cost.Means	Cost.Standard.Deviation	Cost.Coeff.Varation
1000	0.0996053	0.0069090	0.0693635
2000	0.0993512	0.0039797	0.0400569
3000	0.0996503	0.0032459	0.0325728
4000	0.0995676	0.0030517	0.0306496
5000	0.0997402	0.0027021	0.0270917
6000	0.1001326	0.0024729	0.0246960

n	Cost.Means	Cost.Standard.Deviation	Cost.Coeff.Varation
7000	0.1002068	0.0024146	0.0240964
8000	0.0996741	0.0019436	0.0194991
9000	0.1000214	0.0022179	0.0221741
10000	0.1001223	0.0018497	0.0184745

For D = 2:

```
D.2 = 2
antithetic.cost.sim.2 <- replicate(100, sapply(lapply(n.increments, FUN = generate.x.list.antithetic, D
```



n	Cost.Means	Cost.Standard.Deviation	Cost.Coeff.Varation
1000	0.0100438	0.0013899	0.1383868
2000	0.0100327	0.0008485	0.0845693
3000	0.0100807	0.0006285	0.0623505
4000	0.0099841	0.0005494	0.0550251
5000	0.0100221	0.0005662	0.0564996
6000	0.0100236	0.0004638	0.0462707
7000	0.0099936	0.0004396	0.0439848
8000	0.0100532	0.0004353	0.0433040
9000	0.0099828	0.0003833	0.0383915
10000	0.0100310	0.0003814	0.0380200

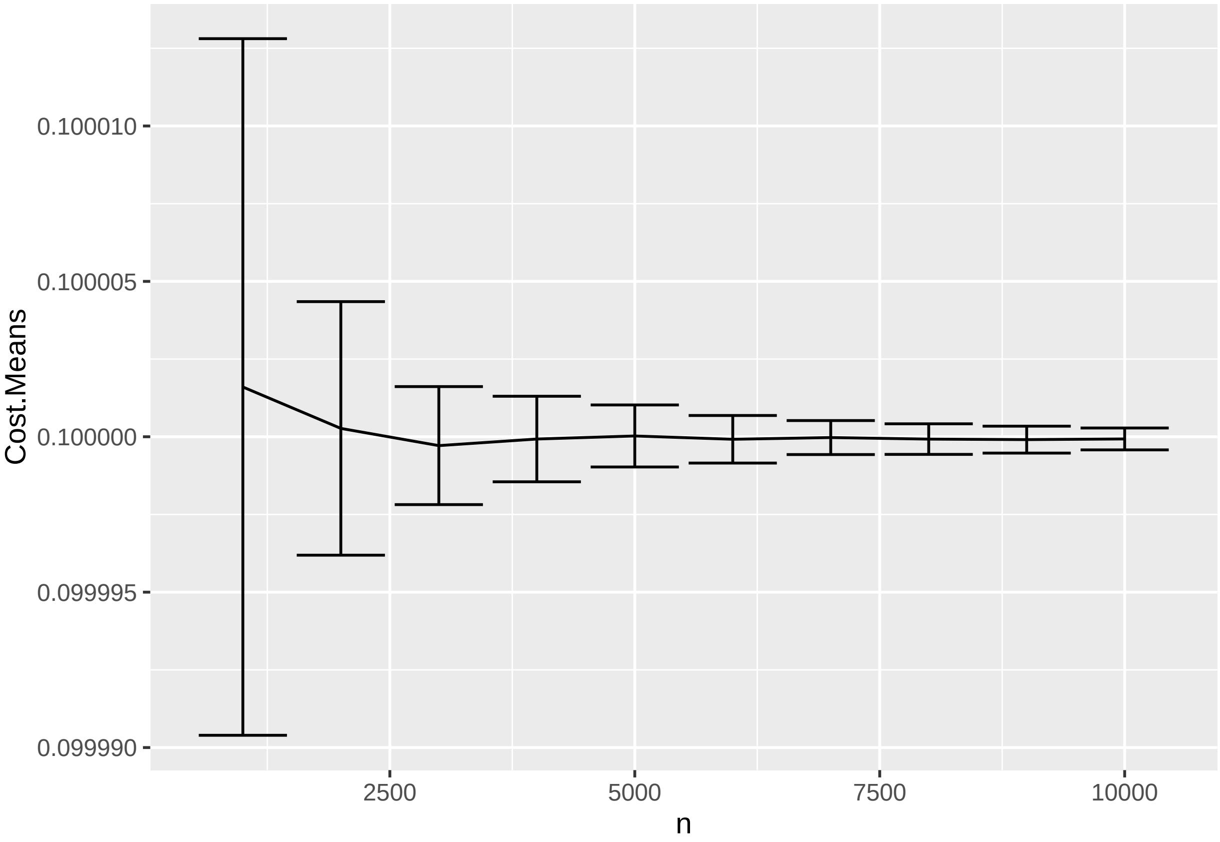
d) Latin Hypercube Sampling

```
library(lhs)
```

```
## Warning: package 'lhs' was built under R version 3.2.5
```

For D = 1:

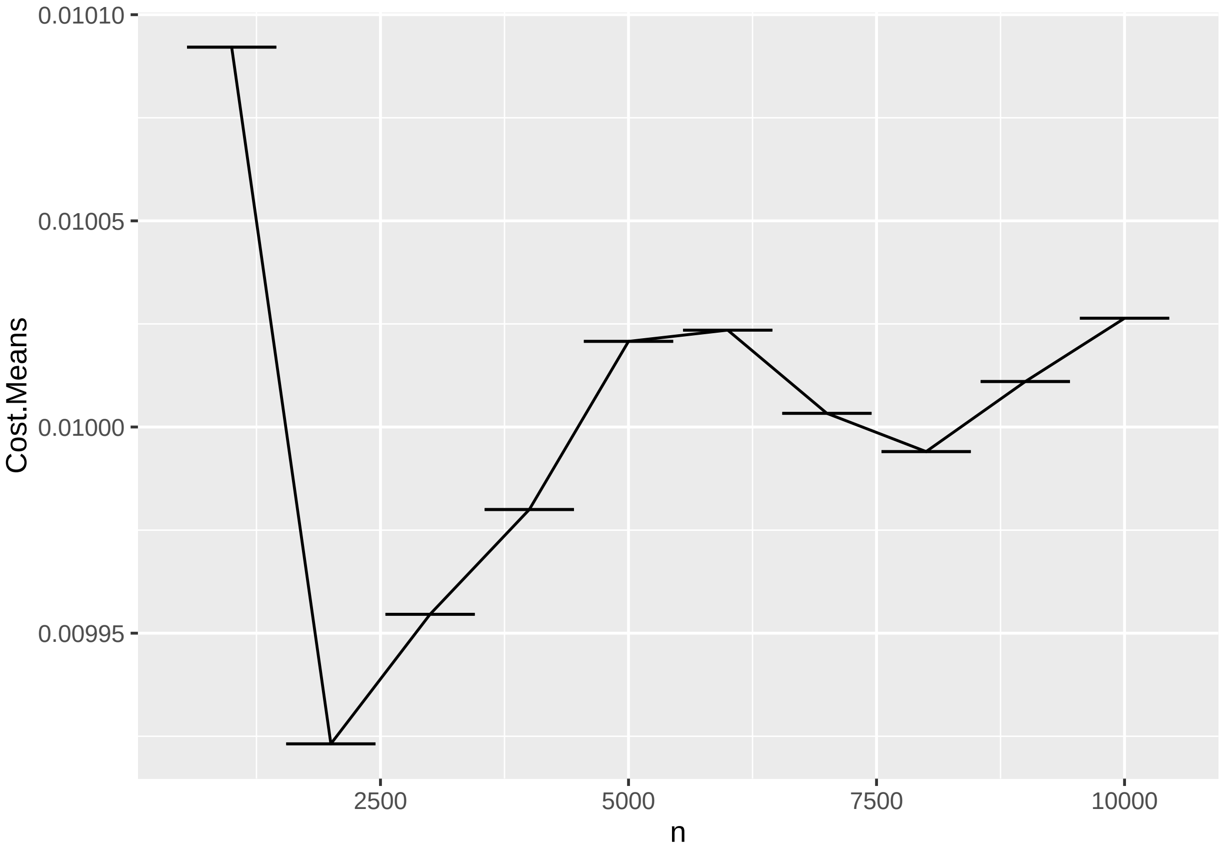
```
generate.x.list.latin <- function(n, D){
  x <- as.matrix((randomLHS(n,D) * 10) - 5 )
  return(x)
}
```



n	Cost.Means	Cost.Standard.Deviation	Cost.Coeff.Varation
1000	0.1000016	1.12e-05	0.0001121
2000	0.1000003	4.10e-06	0.0000408
3000	0.0999997	1.90e-06	0.0000190
4000	0.0999999	1.40e-06	0.0000138
5000	0.1000000	1.00e-06	0.0000100
6000	0.0999999	8.00e-07	0.0000077
7000	0.1000000	5.00e-07	0.0000055
8000	0.0999999	5.00e-07	0.0000049
9000	0.0999999	4.00e-07	0.0000043
10000	0.0999999	4.00e-07	0.0000035

For D = 2:

```
D.2 = 2
latin.cost.sim.2 <- replicate(100, sapply(lapply(n.increments, FUN = generate.x.list.latin, D = D.2), FU
```



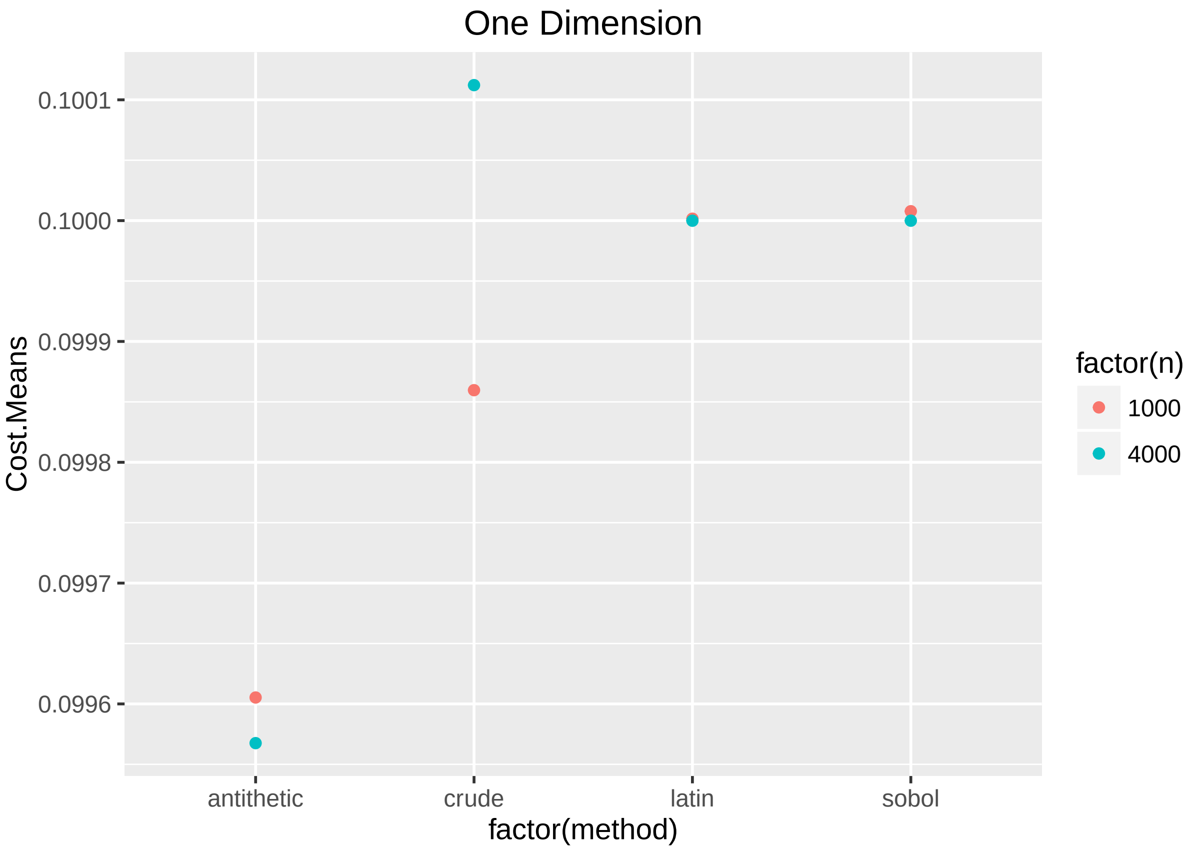
n	Cost.Means	Cost.Standard.Deviation	Cost.Coeff.Varation
1000	0.0100921	0	0
2000	0.0099232	0	0
3000	0.0099546	0	0
4000	0.0099800	0	0
5000	0.0100208	0	0
6000	0.0100235	0	0
7000	0.0100033	0	0
8000	0.0099940	0	0
9000	0.0100110	0	0
10000	0.0100264	0	0
##e) Imp	ortance Sampl	ing	

f) Summary

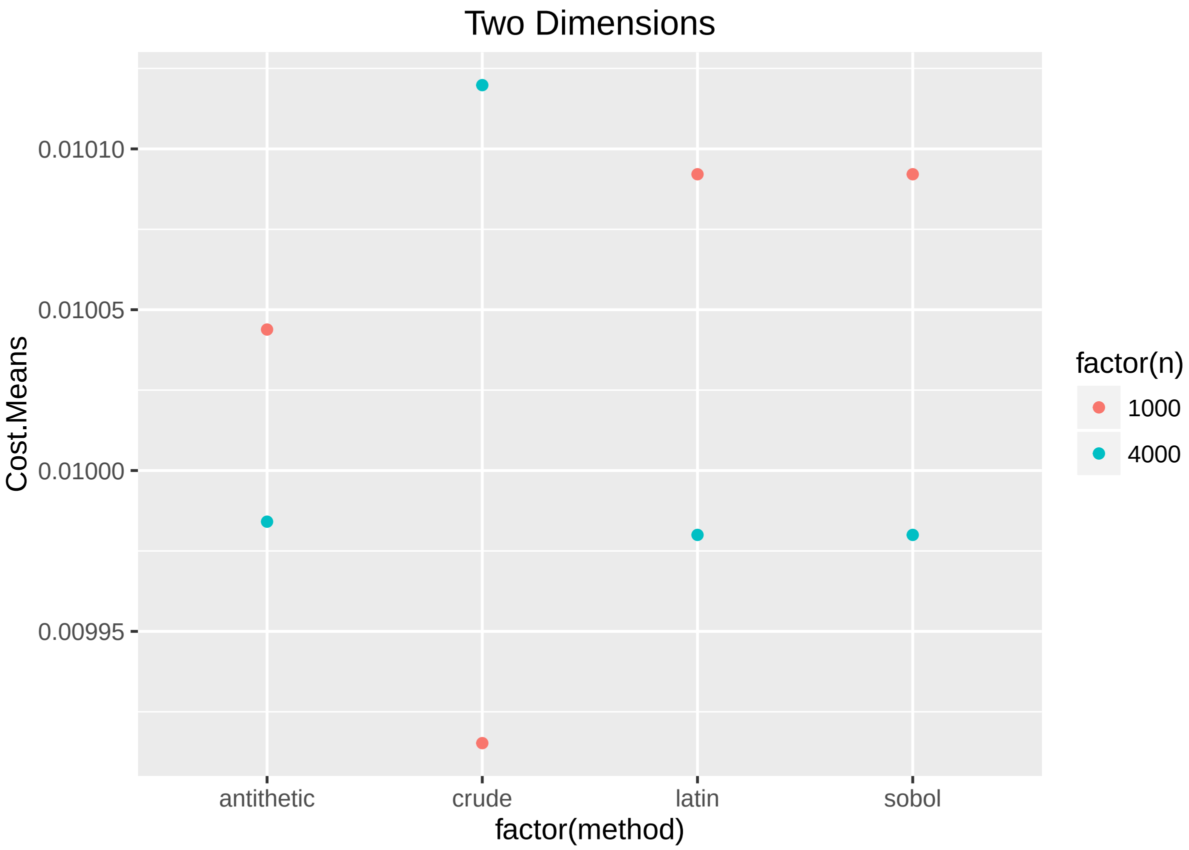
```
summary.cost <- rbind(crude.cost.sim.1.results[c(1,4),], crude.cost.sim.2.results[c(1,4),],
                      sobol.cost.sim.1.results[c(1,4),], sobol.cost.sim.2.results[c(1,4),],
                      antithetic.cost.sim.1.results[c(1,4),], antithetic.cost.sim.2.results[c(1,4),],
                      latin.cost.sim.1.results[c(1,4),], latin.cost.sim.2.results[c(1,4),])
dimensions <- rep(c(1,1,2,2), 4)
method <- c(rep("crude",4), rep("sobol", 4), rep("antithetic", 4), rep("latin",4))

summary.cost <- cbind(method, dimensions, summary.cost)
summary.cost$upper <- summary.cost$Cost.Means + summary.cost$Cost.Standard.Deviation
summary.cost$lower <- summary.cost$Cost.Means - summary.cost$Cost.Standard.Deviation

kable(summary.cost)
```

	method	dimensions	n	Cost.Means	Cost.Standard.Deviation	Cost.Coeff.Varation	upper	lowe
1	crude	1	1000	0.0998597	0.0044341	0.0444035	0.1042938	0.095425
4	crude	1	4000	0.1001121	0.0021317	0.0212933	0.1022439	0.097980
11	crude	2	1000	0.0099153	0.0007810	0.0787692	0.0106963	0.009134
41	crude	2	4000	0.0101198	0.0004260	0.0420941	0.0105458	0.009693
12	sobol	1	1000	0.1000078	0.0000000	0.0000000	0.1000078	0.100007
42	sobol	1	4000	0.0999999	0.0000000	0.0000000	0.0999999	0.099999
13	sobol	2	1000	0.0100921	0.0000000	0.0000000	0.0100921	0.010092
43	sobol	2	4000	0.0099800	0.0000000	0.0000000	0.0099800	0.009980
14	antithetic	1	1000	0.0996053	0.0069090	0.0693635	0.1065143	0.092696
44	antithetic	1	4000	0.0995676	0.0030517	0.0306496	0.1026193	0.096515
15	antithetic	2	1000	0.0100438	0.0013899	0.1383868	0.0114338	0.008653
45	antithetic	2	4000	0.0099841	0.0005494	0.0550251	0.0105335	0.009434
16	latin	1	1000	0.1000016	0.0000112	0.0001121	0.1000128	0.099990
46	latin	1	4000	0.0999999	0.0000014	0.0000138	0.1000013	0.099998
17	latin	2	1000	0.0100921	0.0000000	0.0000000	0.0100921	0.010092
47	latin	2	4000	0.0099800	0.0000000	0.0000000	0.0099800	0.009980

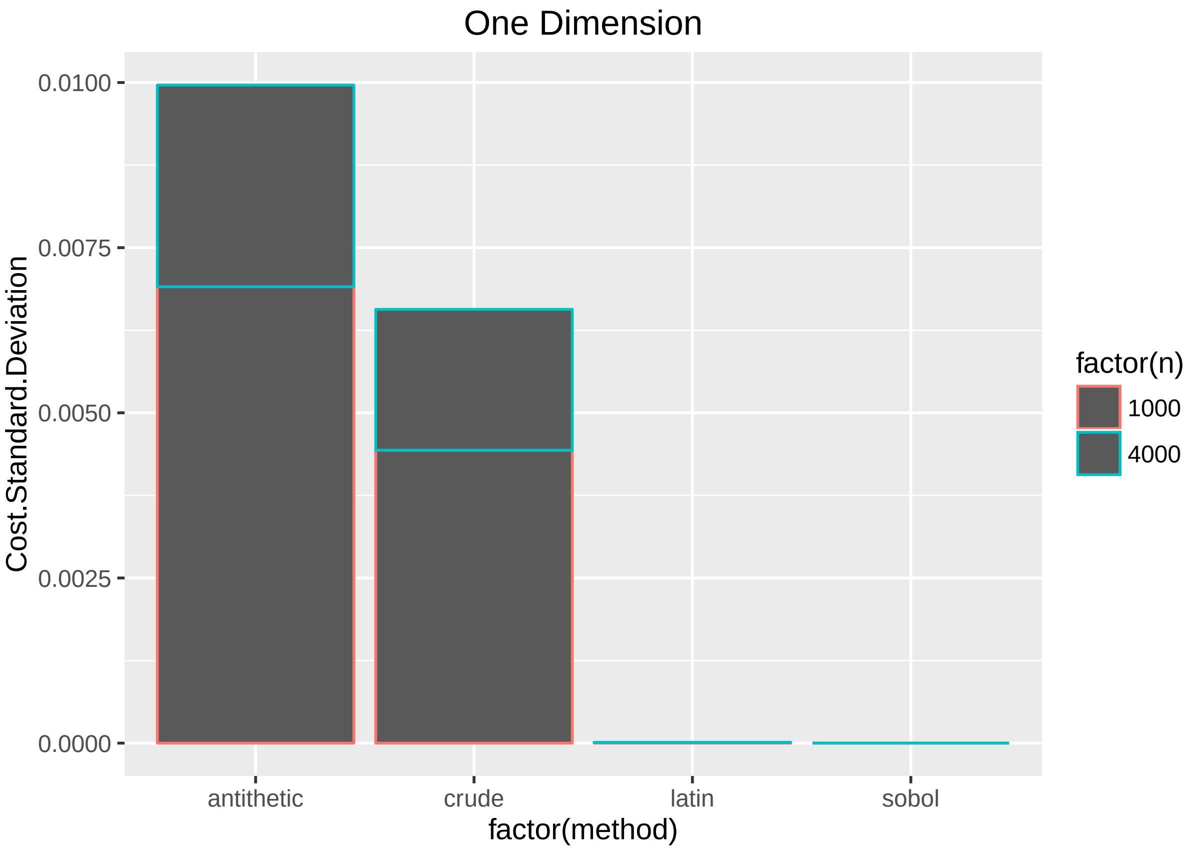
```
ggplot(summary.cost %>% filter(dimensions==1), aes(factor(method), Cost.Means, color = factor(n))) +
  geom_point() + ggtitle("One Dimension")
```



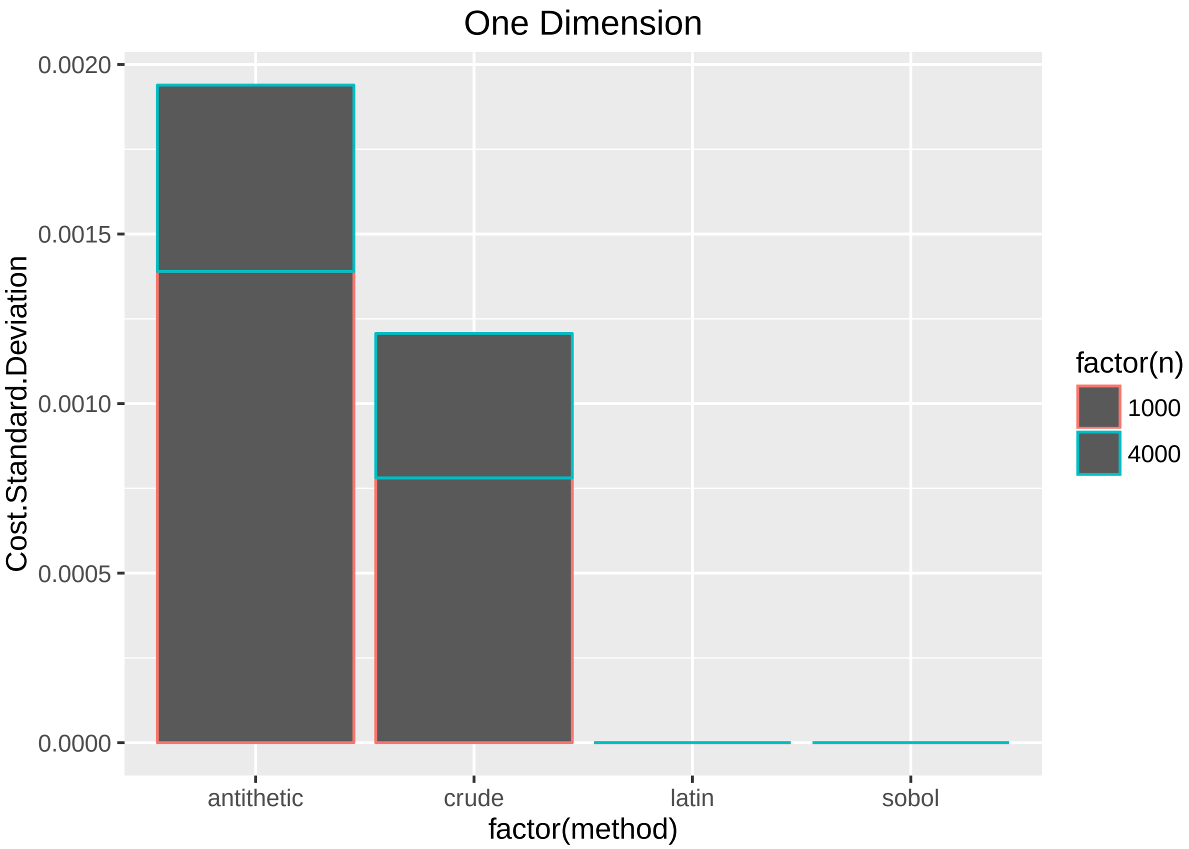
```
ggplot(summary.cost %>% filter(dimensions==2), aes(factor(method), Cost.Means, color = factor(n))) +
  geom_point() + ggtitle("Two Dimensions")
```



```
ggplot(summary.cost %>% filter(dimensions==1), aes(factor(method), Cost.Standard.Deviation, color = fac
  geom_bar(stat = "identity") + ggtitle("One Dimension")
```



```
ggplot(summary.cost %>% filter(dimensions==2), aes(factor(method), Cost.Standard.Deviation, color = fac
  geom_bar(stat = "identity") + ggtitle("One Dimension")
```



Variance was eliminated with the Latin Hypercube sampling and Sobol sampling due to the repeating nature of the distributions. Antithetic sampling increased variance compared to the crude Monte Carlo method.

6.3

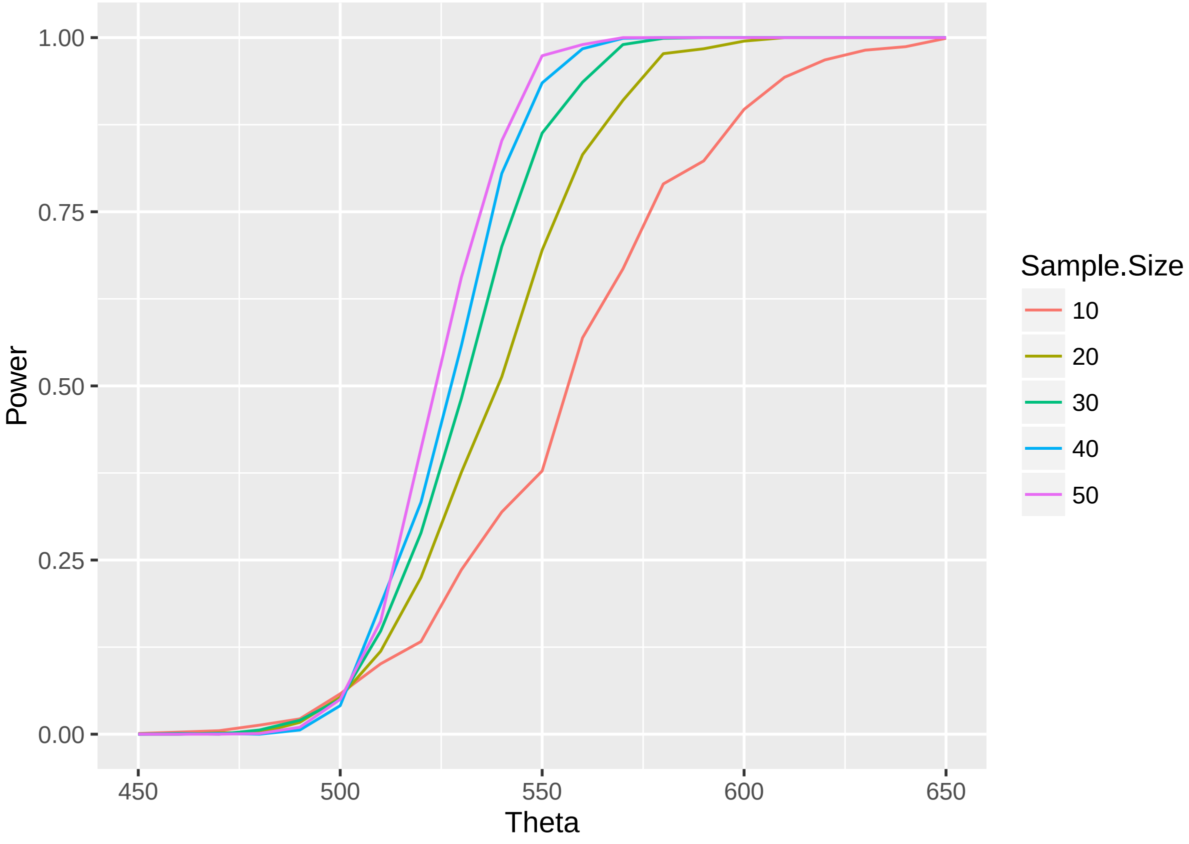
Plot the power curves for the t-test in Example 6.9 for sample sizes 10, 20, 30, 40, and 50, but omit the standard error bars. Plot the curves on the same craph, each in a different color or line type, and include a legend. Comment on the relation between power and sample size.

```
empirical.power <- function(n){
  m = 1000
  mu0 = 500
  sigma = 100
  mu <- c(seq(450, 650, 10))
  M <- length(mu)
  power <- numeric(M)

  for (i in 1:M) {
    mu1 <- mu[i]
    pvalues <- replicate(m, expr = {
      x <- rnorm(n, mean = mu1, sd = sigma)
      ttest <- t.test(x, alternative = "greater",
                      mu = mu0)
      ttest$p.value})
    power[i] <- mean(pvalues <= 0.05)
  }
  return(power)
}
```

```
n.set <- seq(from = 10, to = 50, by = 10)

power.sim <- lapply(n.set, FUN = empirical.power)
```

```
theta <- seq(450, 650, 10)

tidy.results <- NULL

for (i in 1:length(n.set)){

  results.set <- cbind(rep(i*10, times = length(n.set)),
                       theta, unlist(power.sim[i]))
  tidy.results <- rbind(results.set, tidy.results)
}

tidy.results.6.3 <- as.data.frame(tidy.results)
colnames(tidy.results.6.3) <- c("Sample.Size", "Theta", "Power")
tidy.results.6.3$Sample.Size <- as.factor(tidy.results.6.3$Sample.Size)
```

```
ggplot(tidy.results.6.3) +
  geom_line(aes(x = Theta, y = Power, group = Sample.Size, color = Sample.Size))
```



6.4

Suppose that $X_1, ..., X_n$ are a random sample from a lognormal distribution with unknown parameters. Construct a 95% confidence interval for the parameter u. Use a Monte Carlo method to obtain an empirical estimate of the the confidence level.

```
set.seed(4388)
n = 20
alpha = 0.5
reps = 1000

mlg <- runif(1, min = 0, max = 1)
sdlg <- runif(1, min = 0, max = 2)

calcCI <- function(n, alpha){
  x <- rlnorm(n, mlg, sdlg)
  return((n-1) * var(x) / qchisq(alpha, df = n-1))
}

UCL <- replicate(reps, calcCI(n, alpha))

sum(UCL > sdlg^2)
```

```
## [1] 998
```

```
mean(UCL > sdlg^2)
```

```
## [1] 0.998
```

Because the lognormal distribution is very close to the normal distribution, the confidence level is quite high.

7.1

Compute the jackknife estimate of the bias and the standard error of the correlation statistic in Example 7.2.

```
data(law, package = 'bootstrap')
law <- law
```

```
n <- nrow(law)
lsat <- law$LSAT
gpa <- law$GPA

(theta.hat <-cor(lsat, gpa))
```

```
## [1] 0.7763745
```

```
theta.jack <- numeric(n)

for (i in 1:n){
  theta.jack[i] <- cor(lsat[-i], gpa[-i])
}

(bais <- (n-1) * (mean(theta.jack) - theta.hat))
```

```
## [1] -0.006473623
```

```
(se <- sqrt((n-1) * mean((theta.jack - mean(theta.jack))^2)))
```

```
## [1] 0.1425186
```

7.4

Refer to the air-conditioning data set aircondit provided in the boot package. The 12 observations are the times in hours between failures of ac equiment. Assme that the times between failures follow an exponential model. Obtain the MLE of the hazard rate and use bootstrap to estimate the bias and standard error of the estimate.

```
data(aircondit, package = 'boot')
aircondit <- aircondit
```