IS604_charleyferrari_hw4_fin

Charley Ferrari

October 16, 2015

Question 1

Variance reduction procedures

```
#Libraries

library(ggplot2)
library(abind)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
##
## The following objects are masked from 'package:stats':
##
##     filter, lag
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(bootstrap)
library(boot)
```

a. Crude Monte Carlo

For sample sizes 1000 to 10000 (in increments of 1000), obtain 100 estimates for E[c(x)] when D = 1, using crude Monte Carlo sampling.

First, lets create a function cofx, that will perform our function. It's meant to take in a D-dimensional vector, and return:

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

```
cofx <- function(x,D){
  theta.hat <- (1/((2*pi)^(D/2)))*exp((-1/2)*(t(x) %*% x))
  return(theta.hat)
}
```

For crude monte carlo, I'm going to create a function "crudefunction" that creates a $D \times n$ matrix, applies cofx to each column n times. n in this case is meant to refer to the sample sizes 1000 to 10000 in increments of 1000.

```
crudefunction <- function(n,D){

  m <- D*n
  x <- matrix(runif(m, min=-5, max=5),nrow=n)

  return(mean(apply(x,1,cofx,D=D)))

}
```

Now, we're going to perform our simulation. For our group of sample sizes, I'll perform 100 pulls to get a mean and standard deviation.

```
crudecreate <- function(D){

  n <- seq(1000,10000,by=1000)
  means <- c()
  sds <- c()
  coefvars <- c()

  for(i in n){
    thetasample <- replicate(100,crudefunction(i,D))
    means <- c(means, mean(thetasample))
    sds <- c(sds, sd(thetasample))
    coefvars <- c(coefvars, mean(thetasample)/sd(thetasample))
  }

  table <- data.frame(n=n, means=means, sds=sds, coefvars=coefvars, D=D,
                      method="Crude")

  return(table)

}

table <- crudecreate(1)
```

Lets repeat this analysis for D=2

```
table <- rbind(table,crudecreate(2))
```

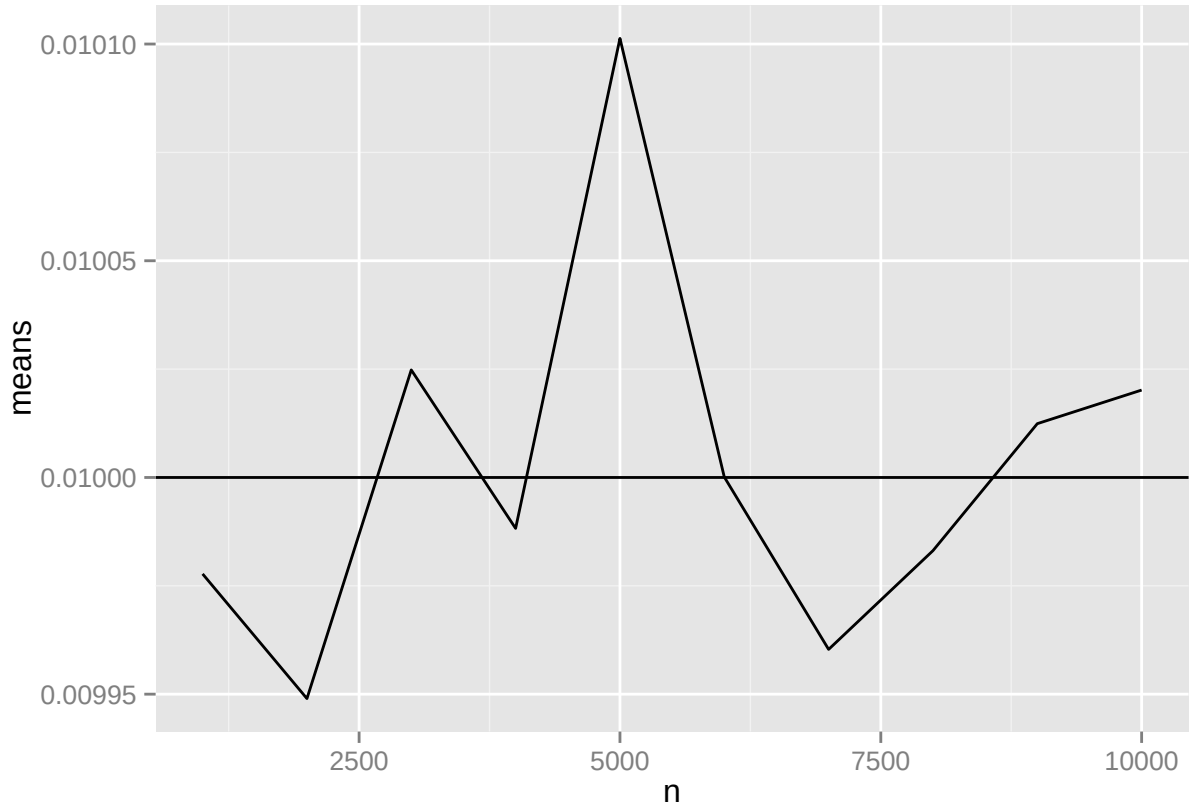
(I'm bringing in this table from a previously saved csv file to speed up the RMD)

```
setwd("E:/Downloads/Courses/CUNY/SPS/Git/IS 604 Simulation and Modeling techniques/Homework 4")

table <- read.csv("variancetable.csv")

ggplot(data=filter(table,method=="Crude",D==1),
       aes(x=n,y=means)) +
  geom_line() +
  geom_hline(yintercept=(1/10))
```

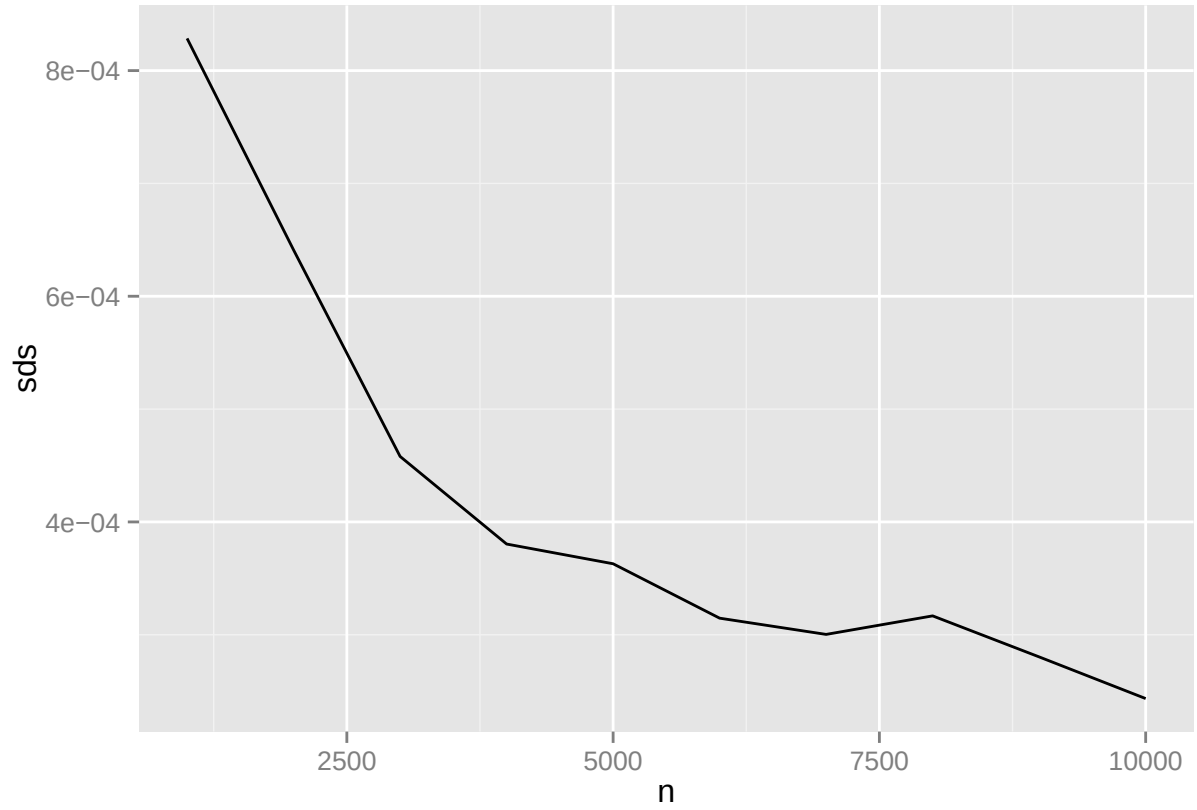

```
ggplot(data=filter(table,method=="Crude",D==1),
       aes(x=n,y=sds,coolor=as.factor(D))) +
  geom_line()
```

For D == 1, you can see the standard deviation fall steadily as my sample size increases, while the mean seems to fluctuate slightly closer to the true mean.

```
ggplot(data=filter(table,method=="Crude",D==2),
       aes(x=n,y=means)) +
  geom_line() +
  geom_hline(yintercept=(1/10)^2)
```




```
ggplot(data=filter(table,method=="Crude",D==2),
       aes(x=n,y=sds,coolor=as.factor(D))) +
  geom_line()
```



We get similar results for D==2, with less of an indication of the fluctuation about the mean decreasing in the graph.

b. Quasi-Random Numbers

To create the same pull from a sobol distribution, I'll need a few helper functions. Below I defined:

binvec: returns a binary representation of a number

mcalc: returns a vector m, which is used as an input in the generator matrices

vcalc: returns a 3 dimensional vector, with dimensions r,r,d. r is the number of binary digits required to print out the length of the sample n, and d is the number of components of the sobol distribution.

scramblecalc: returns a matrix that is used to scramble the sobol numbers, according to the Uniform Linear Scrambling algorithm.

```
binvec <- function(n){
  return(rev(as.numeric(intToBits(n))[1:(floor(log(n)/log(2))+1)]))
}

mcalc <- function(k,r){

  polys <- c(1,3,7,11,13,19,25,37,59,47,61,55,41,67,97,91,109,103,
             115,131,193,137,145,143,241,157,185,167,229,171,213,191,
             253,203,211,239,247,285,369,299,425,301,361,333,357,351,
             501,355,397,391,451,463,487)
```

```
  ivals <- t(matrix(c(1,1,1,1,1,1,1,1,
                      1,3,5,15,17,51,85,255,
                      1,1,7,11,13,61,67,79,
                      1,3,7,5,7,43,49,147,
                      1,1,5,3,15,51,125,141,
                      1,3,1,1,9,59,25,89,
                      1,1,3,7,31,47,109,173,
                      1,3,3,9,9,57,43,43,
                      1,3,7,13,3,35,89,9,
                      1,1,5,11,27,53,69,25,
                      1,3,5,1,15,19,113,115,
                      1,1,7,3,29,51,47,97,
                      1,3,7,7,21,61,55,19,
                      1,1,1,9,23,39,97,97,
                      1,3,3,5,19,33,3,197,
                      1,1,3,13,11,7,37,101,
                      1,1,7,13,25,5,83,255,
                      1,3,5,11,7,11,103,29,
                      1,1,1,3,13,39,27,203,
                      1,3,1,15,17,63,13,65),nrow=8))

  m <- ivals[k,]
  ppn <- polys[k]
  c <- tail(binvec(ppn),-1)
  deg <- length(c)
  for(i in 8:r){
    s <- 0
    for(j in 1:deg){
      s <- bitwXor(s,(2^j)*c[j]*m[i-j])
    }
    m <- c(m,bitwXor(s,m[i-deg]))
  }

  return(m[1:r])

}

vcalc <- function(k,r){
  V <- diag(r)
  if(k>=2){
    for(i in 2:k){
      m <- mcalc(i,r)
      vk <- c()
      for(j in 1:r){
        h <- binvec(m[j])
        h <- c(rep(0,j-length(h)), h, rep(0,r-j))
        vk <- cbind(vk,as.matrix(h,nrow=r))
      }
      V <- abind(V,vk,along=3)
    }
  }
  return(array(V,dim=c(r,r,k)))
}
```

```
scramblevcalc <- function(r){

  scrambleapply <- function(rownum,r){
    samp <- sample(c(1,0),rownum-1,replace=TRUE)
    return(c(samp,1,rep(0,r-rownum)))
  }

  t(sapply(1:r,scrambleapply, r=r))

}
```

The sobol function uses these helper functions to generate a sobol sequence. It takes the number of dimensions (or components), n, and a bool of whether or not to scramble the sequence.

```
sobol <- function(d,n,scramble=FALSE){

  b <- 2
  N <- 0:(n-1)
  r <- floor(log(n-1)/log(b))+1
  bb <- 1/b^(1:r)

  bitcreate <- function(N,r){
    return(as.numeric(intToBits(N))[1:r])
  }

  a <- t(apply(as.matrix(N,ncol=1),1,bitcreate, r=r))

  p <- matrix(nrow=n,ncol=d)

  V <- vcalc(d, r)

  for(i in 1:d){
    Vtemp <- V[,,i]
    atv <- (a%*%t(Vtemp))%%b
    if(scramble){
      scramblevec <- scramblevcalc(r)
      atv <- (atv%*%t(scramblevec))%%b
    }

    p[,i] <- bb %*% t(atv)
  }

  return(p)

}
```

Lets use a similar method as before to generate 100 samples each of n=1000 - 10000

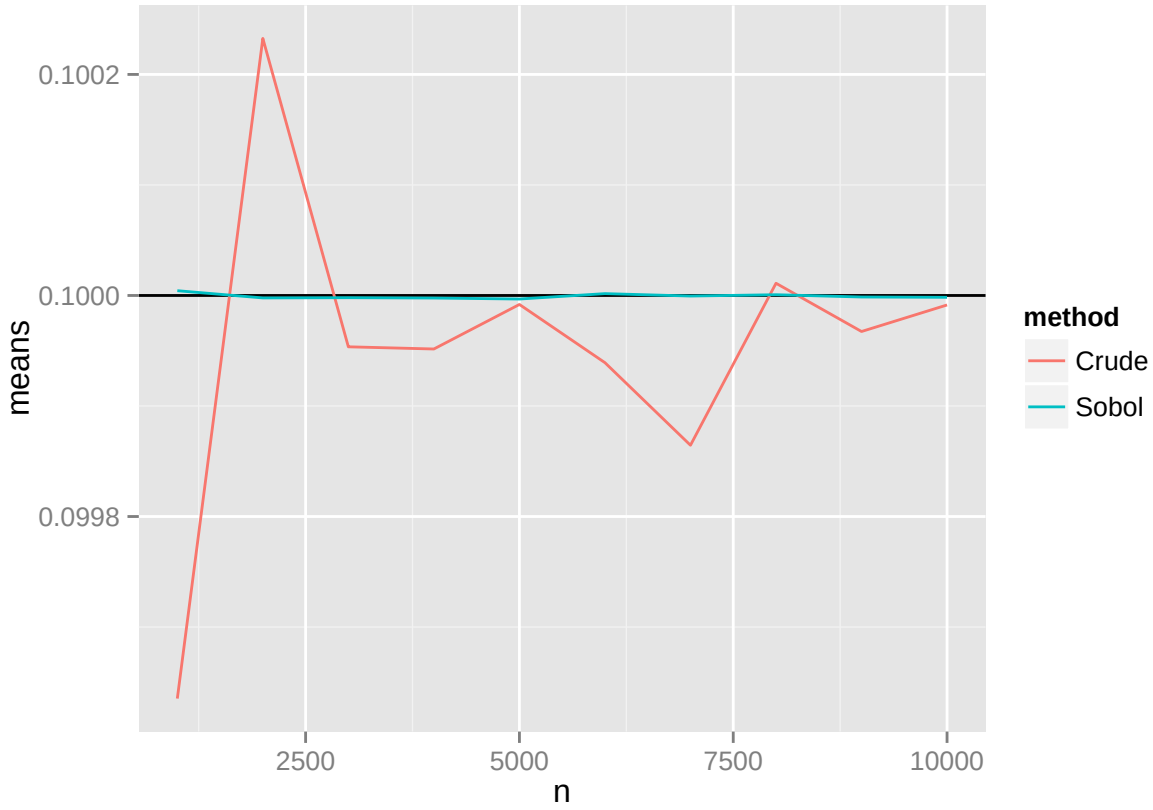
```
sobolfunction <- function(n,D){

  x <- sobol(D,n,scramble=TRUE)
  x <- (x-0.5)*10
```

```
  return(mean(apply(x,1,cofx,D=D)))

}

sobolcreate <- function(D){

  n <- seq(1000,10000,by=1000)
  means <- c()
  sds <- c()
  coefvars <- c()

  for(i in n){
    thetasample <- replicate(100,sobolfunction(i,D))
    means <- c(means, mean(thetasample))
    sds <- c(sds, sd(thetasample))
    coefvars <- c(coefvars, mean(thetasample)/sd(thetasample))
  }

  table <- data.frame(n=n, means=means, sds=sds, coefvars=coefvars, D=D,
                      method="Sobol")

  return(table)

}

#table <- rbind(table,sobolcreate(1))
```

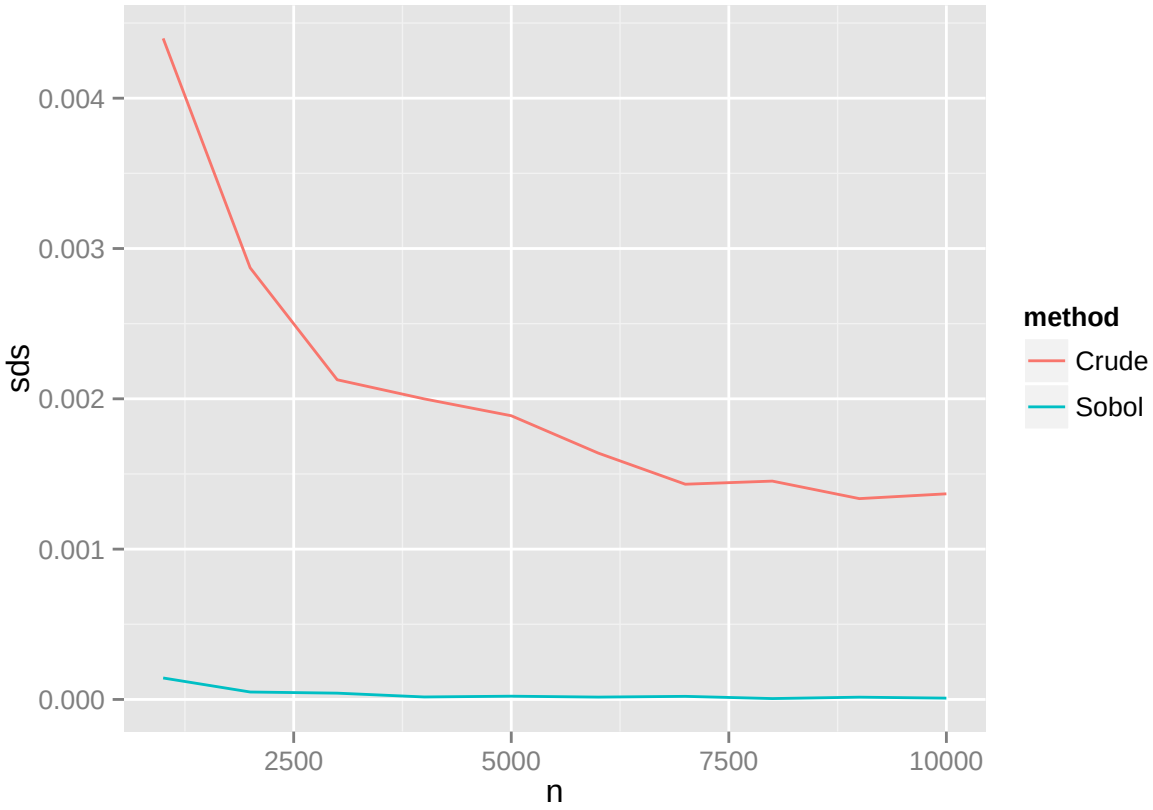
And lets do this for D=2

```
table <- rbind(table,sobolcreate(2))
```

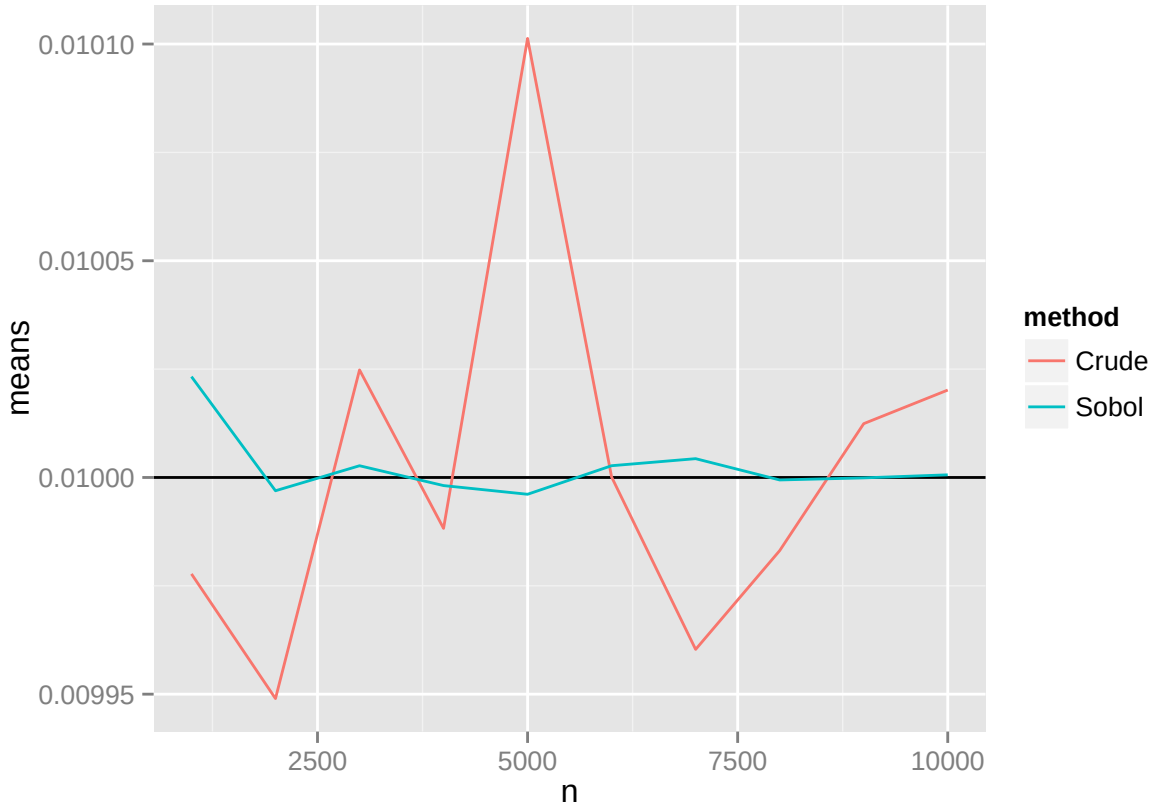
```
ggplot(filter(table,method %in% c("Crude","Sobol"),D==1),
       aes(x=n,y=means,color=method)) +
  geom_hline(yintercept=(1/10)) +
  geom_line()
```



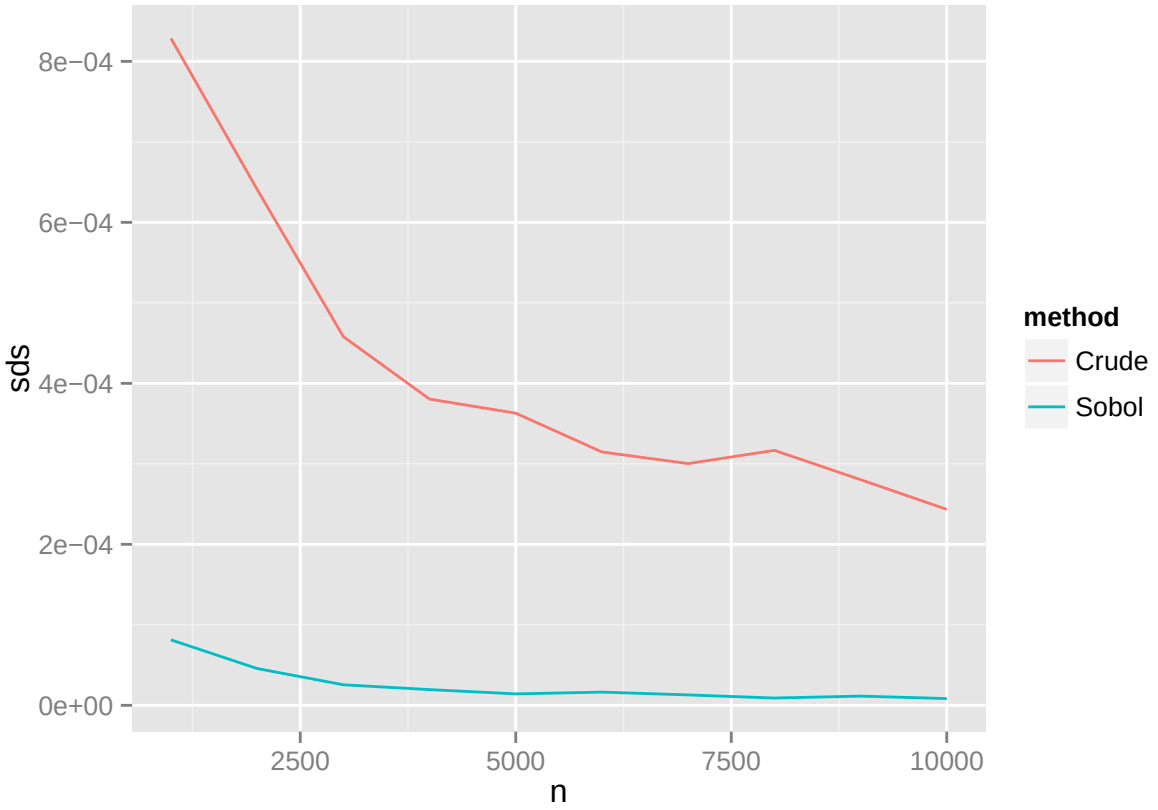

```
ggplot(filter(table,method %in% c("Crude","Sobol"),D==1),
       aes(x=n,y=sds,color=method)) + geom_line()
```




```
ggplot(filter(table,method %in% c("Crude","Sobol"),D==2),
       aes(x=n,y=means,color=method)) +
  geom_hline(yintercept=(1/10)^2) +
  geom_line()
```




```
ggplot(filter(table,method %in% c("Crude","Sobol"),D==2),
       aes(x=n,y=sds,color=method)) + geom_line()
```



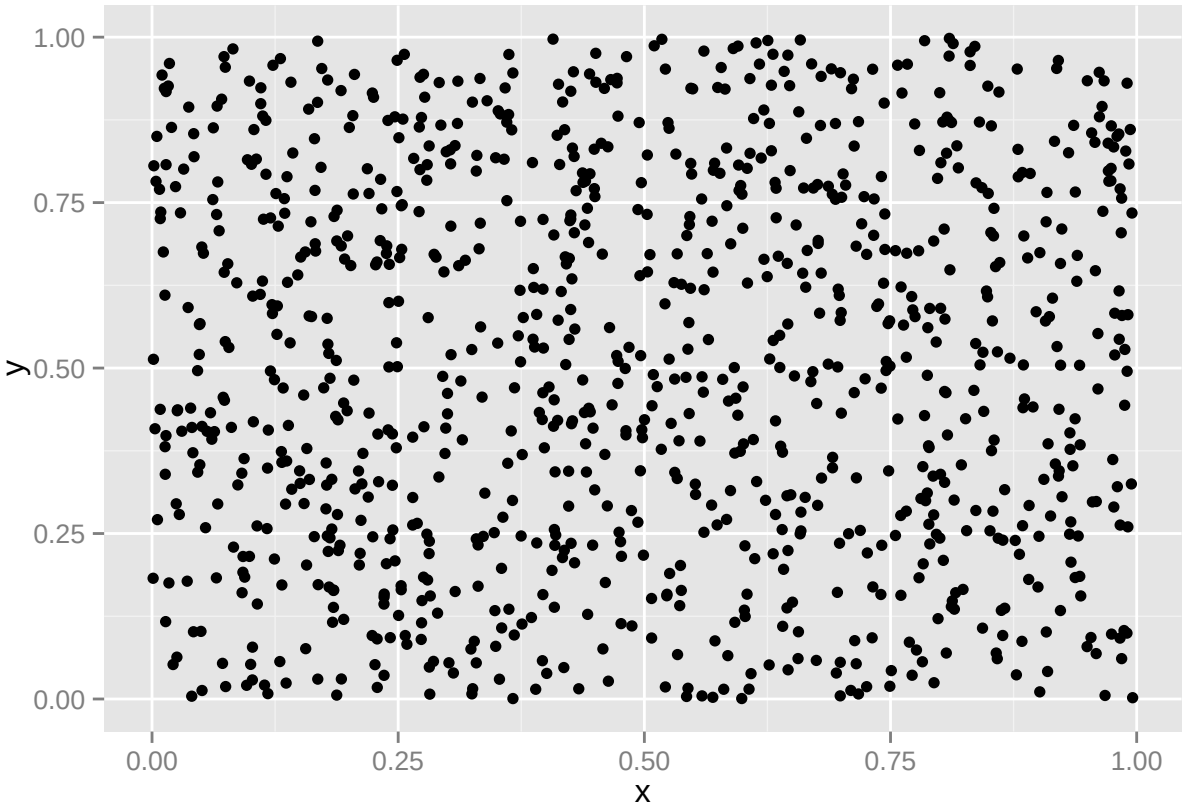
These graphs clearly show that we ended up with a better estimate of the mean using Sobol numbers, and the variance was lower.

Lets look at the Sobol distribution compared to the random, when scatter- plotted in two dimensions:

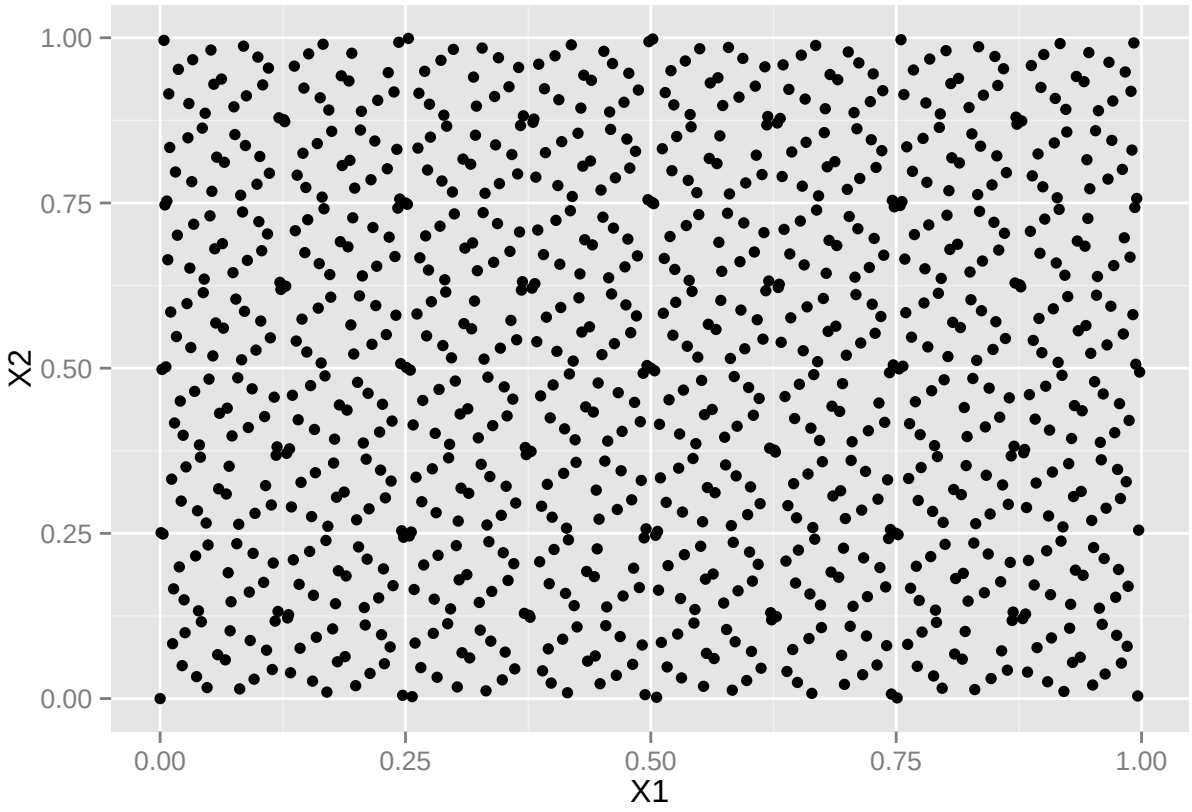
```
sobolfunction(1000,2)
```

```
## [1] 0.009995432
```

```
soboldata <- data.frame(sobol(2,1000))

runifdata <- data.frame(x=runif(1000),y=runif(1000))

ggplot(runifdata,aes(x=x,y=y))+geom_point()
```




```
ggplot(soboldata,aes(x=X1,y=X2))+geom_point()
```



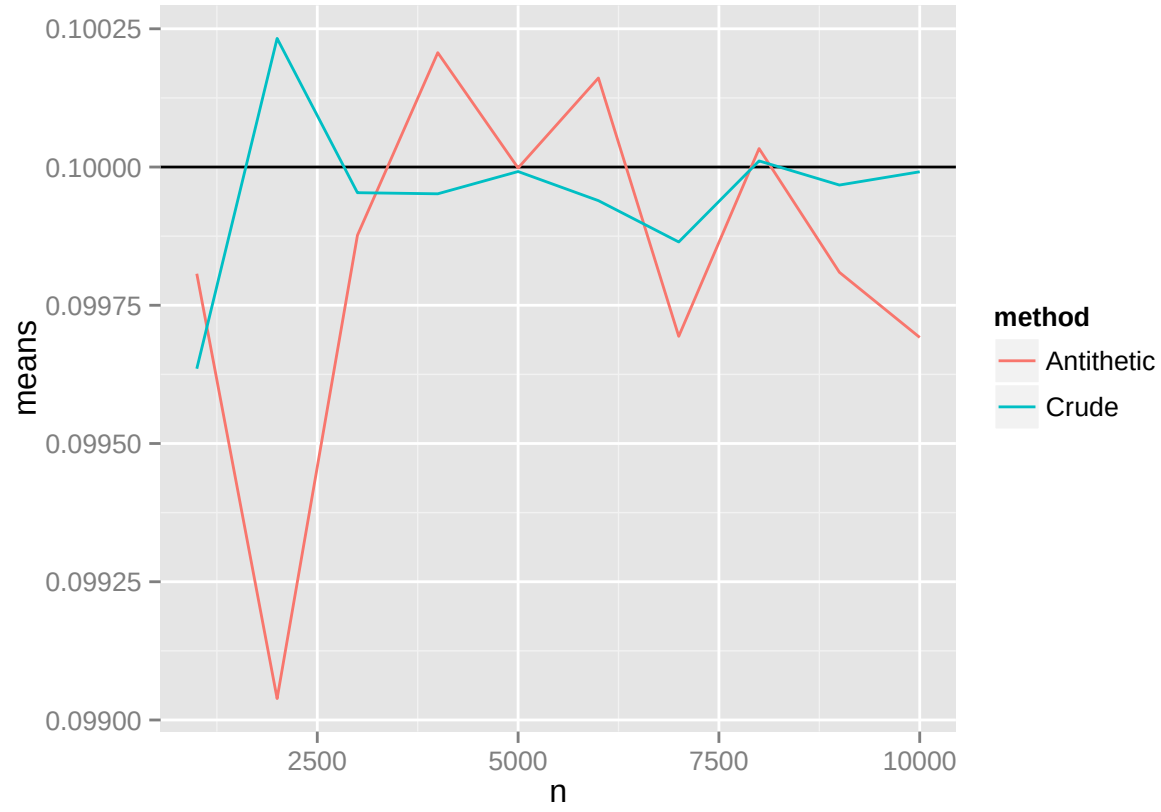
c. Antithetic Variables

Reducing variance using antithetic variables. Similar in method to part a, I'm dividing my sample in half, and then creating another half by subtracting the first half from 1.

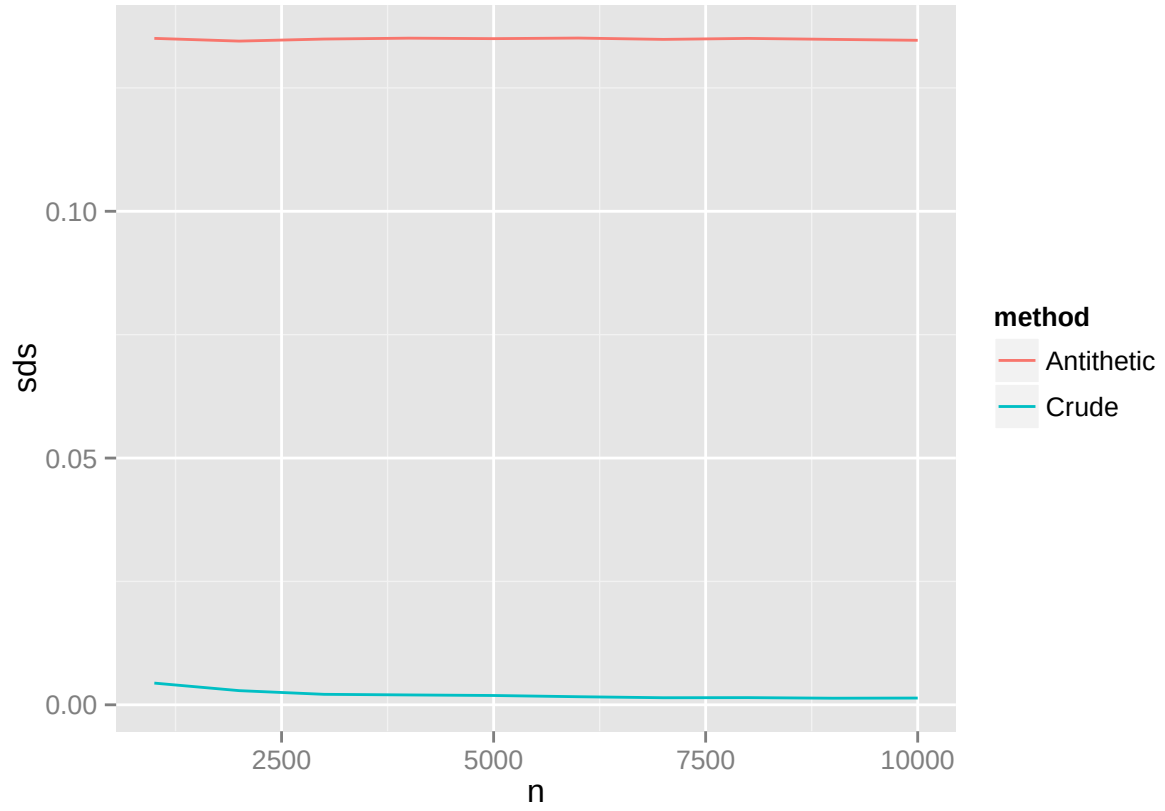
```
atfunction <- function(n,D){

  m <- D*n
  xhalf1 <- matrix(runif(m/2),nrow=n/2)
  xhalf2 <- 1-xhalf1
  xhalf1 <- (xhalf1-0.5)*10
  xhalf2 <- (xhalf2-0.5)*10

  fx1 <- apply(xhalf1,1,cofx,D=D)
  fx2 <- apply(xhalf2,1,cofx,D=D)
  fx <- (fx1+fx2)/2

  return(fx)

}

atcreate <- function(D){

  n <- seq(1000,10000,by=1000)
  means <- c()
```

```
  sds <- c()
  coefvars <- c()

  for(i in n){
    thetasample <- replicate(100,atfunction(i,D))
    means <- c(means, mean(thetasample))
    sds <- c(sds, sd(thetasample))
    coefvars <- c(coefvars, mean(thetasample)/sd(thetasample))
  }

  table <- data.frame(n=n, means=means, sds=sds, coefvars=coefvars, D=D,
                      method="Antithetic")

  return(table)

}

table <- rbind(table,atcreate(1))

table <- rbind(table,atcreate(2))
```

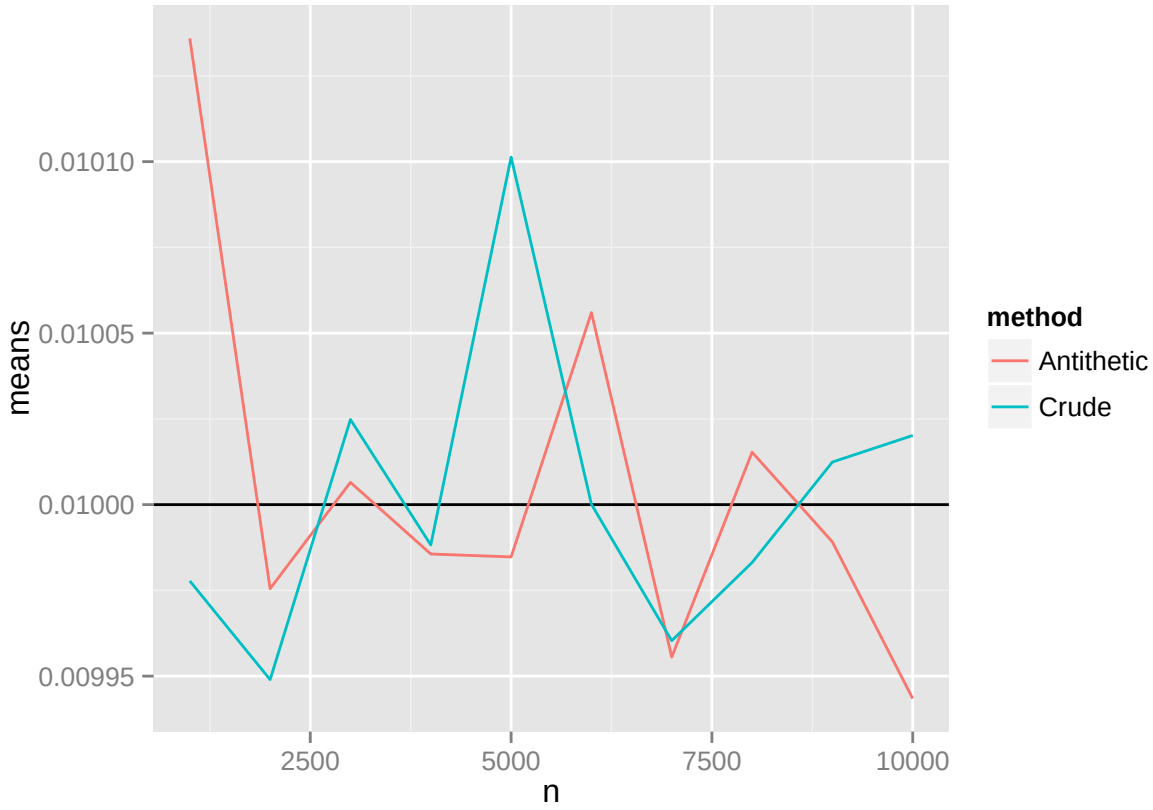
```
ggplot(filter(table,method %in% c("Crude","Antithetic"),D==1),
       aes(x=n,y=means,color=method)) +
  geom_hline(yintercept=(1/10)) +
  geom_line()
```



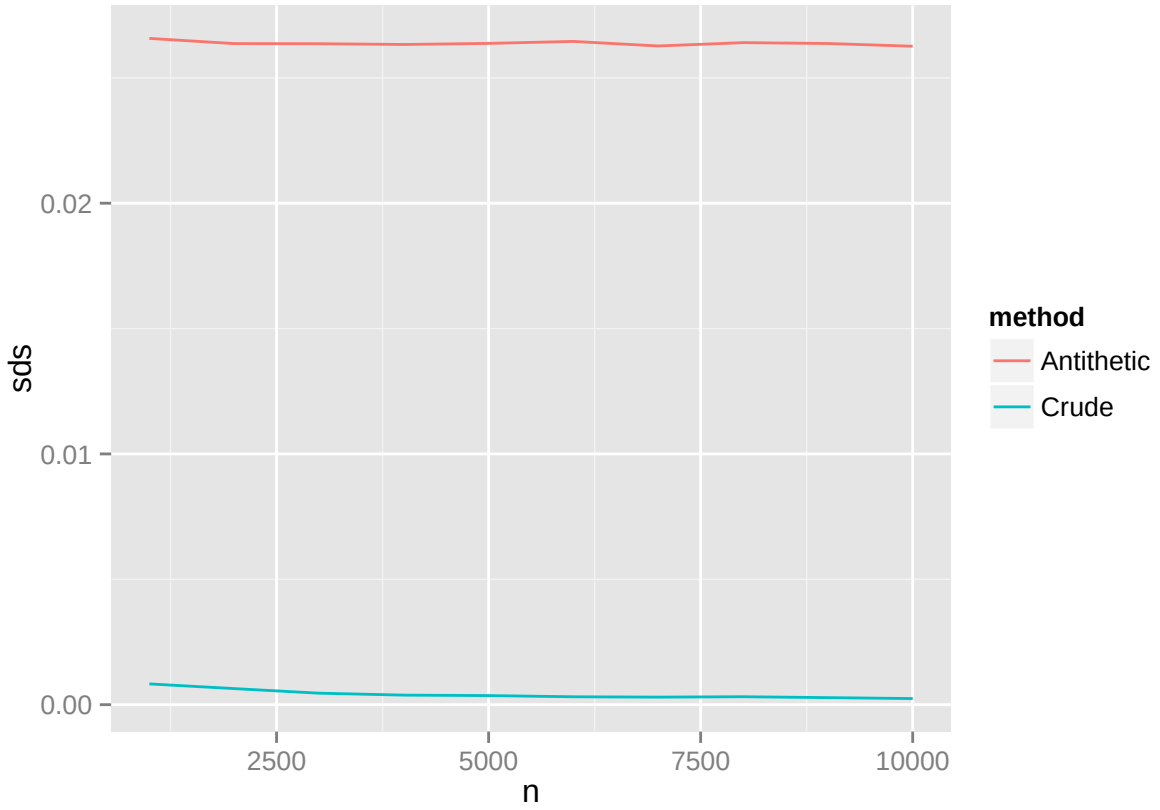
```
ggplot(filter(table,method %in% c("Crude","Antithetic"),D==1),
       aes(x=n,y=sds,color=method)) + geom_line()
```




```
ggplot(filter(table,method %in% c("Crude","Antithetic"),D==2),
       aes(x=n,y=means,color=method)) +
  geom_hline(yintercept=(1/10)^2) +
  geom_line()
```




```
ggplot(filter(table,method %in% c("Crude","Antithetic"),D==2),
       aes(x=n,y=sds,color=method)) + geom_line()
```



It seems the standard deviations of our 100 pulls actually increased in this case, which would seem strange for a variance reduction technique.

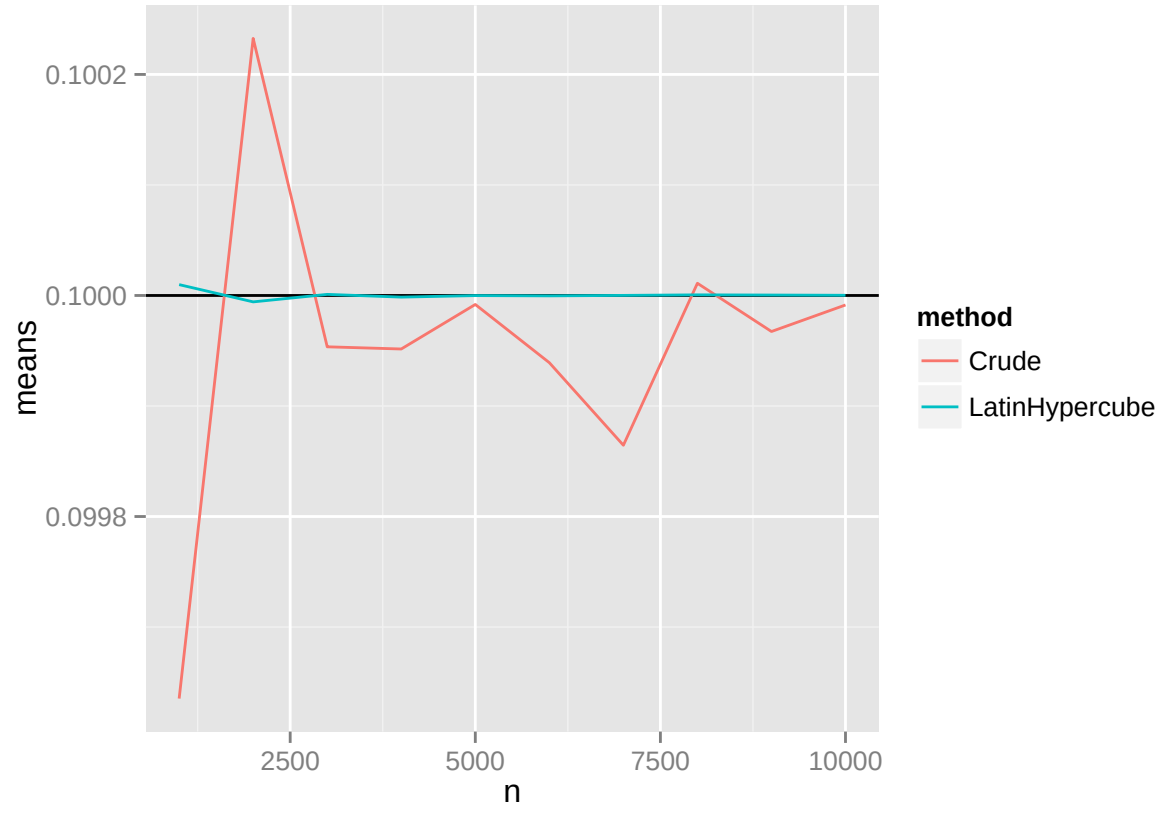
This might be due to the fact that the normal function is not monotonic. The Kroese handbook mentions this is a precondition to the antithetic method being successful.

d. Latin Hypercube Sampling

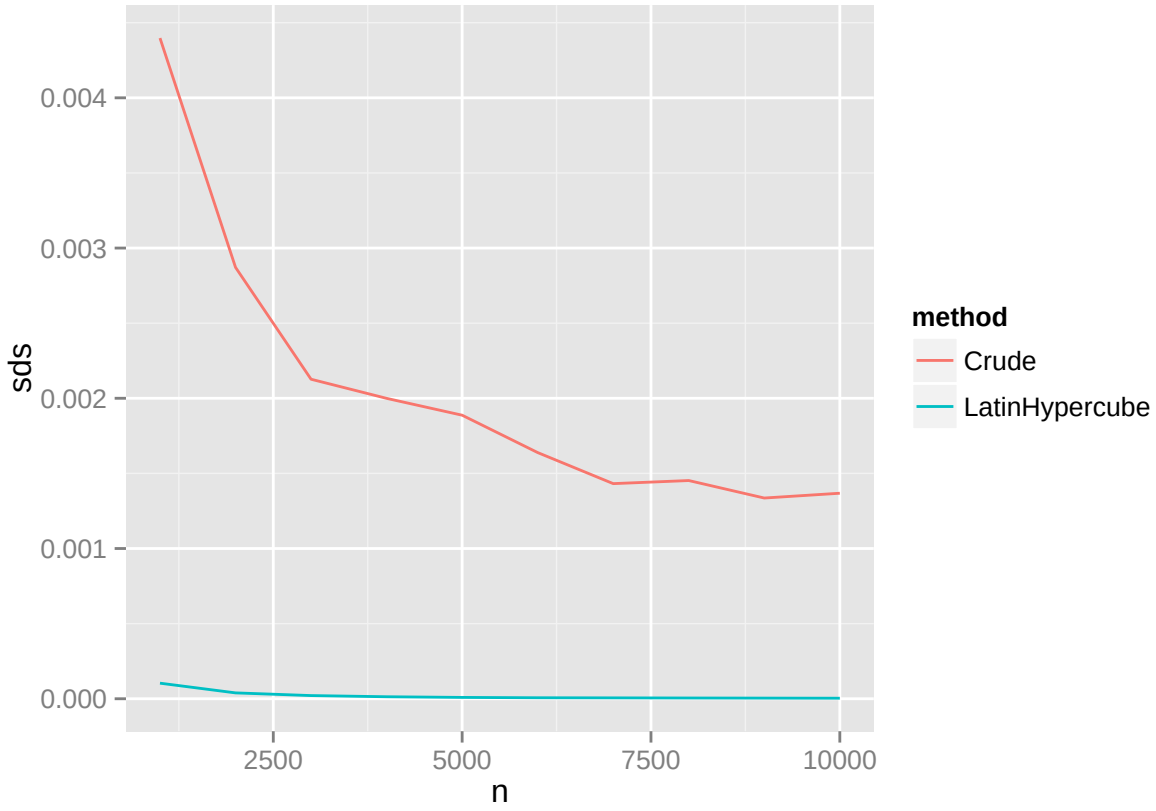
```
lhfunction <- function(n,D,k){

  m <- D*n/k

  return(mean(replicate(k,{

    x  <- matrix(runif(m),nrow=n/k)
    p <- replicate(D,sample(1:(n/k)))
    V <- (p+1-x)/(n/k)
    V <- (V-0.5)*10
    Y <- mean(apply(V,1,cofx,D=D))

  })))

}

lhcreate <- function(D,k){
```

```
  n <- seq(1000,10000,by=1000)
  means <- c()
  sds <- c()
  coefvars <- c()

  for(i in n){
    thetasample <- replicate(100,lhfunction(i,D,k))
    means <- c(means, mean(thetasample))
    sds <- c(sds, sd(thetasample))
    coefvars <- c(coefvars, mean(thetasample)/sd(thetasample))
  }

  table <- data.frame(n=n, means=means, sds=sds, coefvars=coefvars, D=D,
                      method="LatinHypercube")

  return(table)

}

table <- rbind(table,lhcreate(1,10),lhcreate(2,10))
```

```
ggplot(filter(table,method %in% c("Crude","LatinHypercube"),D==1),
       aes(x=n,y=means,color=method)) +
  geom_hline(yintercept=(1/10)) +
  geom_line()
```

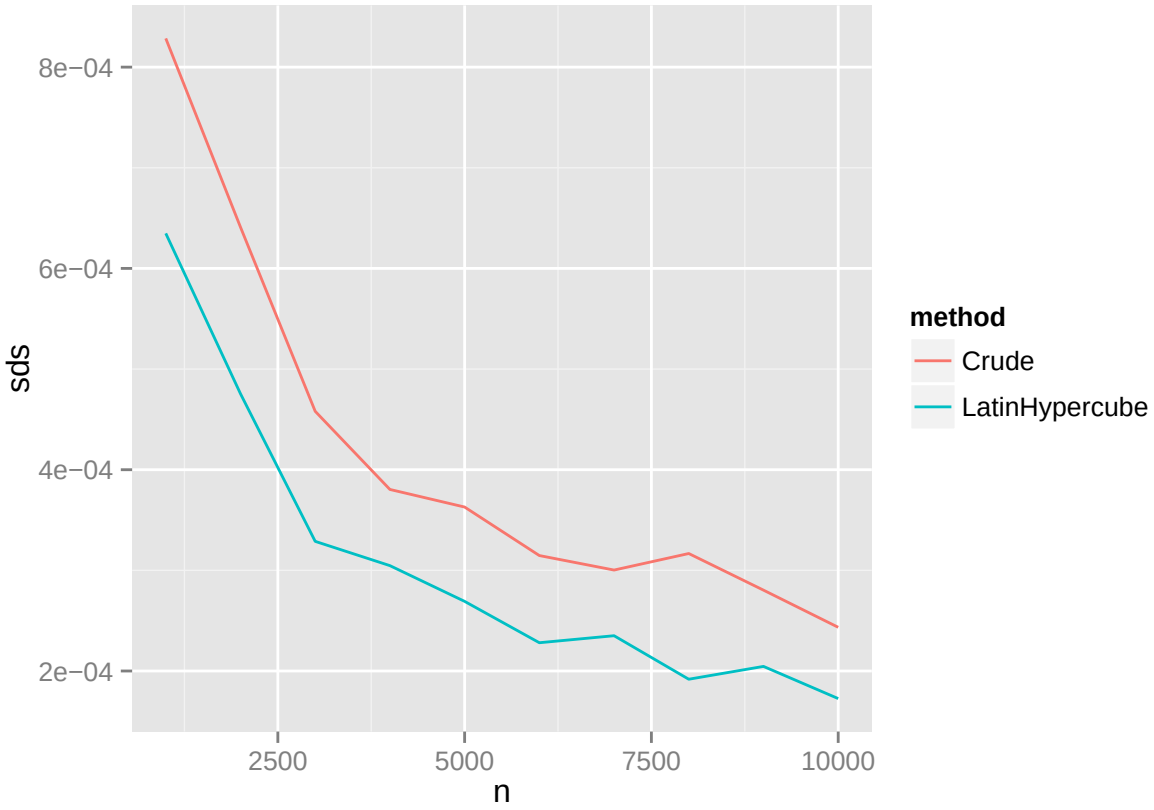


```
ggplot(filter(table,method %in% c("Crude","LatinHypercube"),D==1),
       aes(x=n,y=sds,color=method)) + geom_line()
```




```
ggplot(filter(table,method %in% c("Crude","LatinHypercube"),D==2),
       aes(x=n,y=means,color=method)) +
  geom_hline(yintercept=(1/10)^2) +
  geom_line()
```


```
ggplot(filter(table,method %in% c("Crude","LatinHypercube"),D==2),
       aes(x=n,y=sds,color=method)) + geom_line()
```



The Latin Hypercube method also seems to have reduced the variance, and gave us a better estimate of the mean.

e. Importance Sampling To pick a function here, I wanted to use an upside-down parabola that crossed the x-axis at -5 and 5, which would be $\frac{-3(x-5)(x+5)}{500}$

This would be difficult to solve for an inverse transform, so lets sample from it using acceptance/rejection.

The max of my parabola is at 0.15, and cofx(0,1) (the normal distribution's maximum) is 0.3989. So, I can safely say:

$$\frac{f(x)}{g(x)} \leq \frac{0.15}{0.1} \leq 1.5$$

So, to sample from f(x), I'll generate y from g(x), generate a random u from Uniform(0,1), and accept y if $U < \frac{f(y)}{cg(y)}$

The sampling would work like this:

$$\frac{f(x)}{cg(x)} = \frac{-3(x-5)(x+5)}{500} \times \frac{10}{1.5} = \frac{(x-5)(x+5)}{25}$$

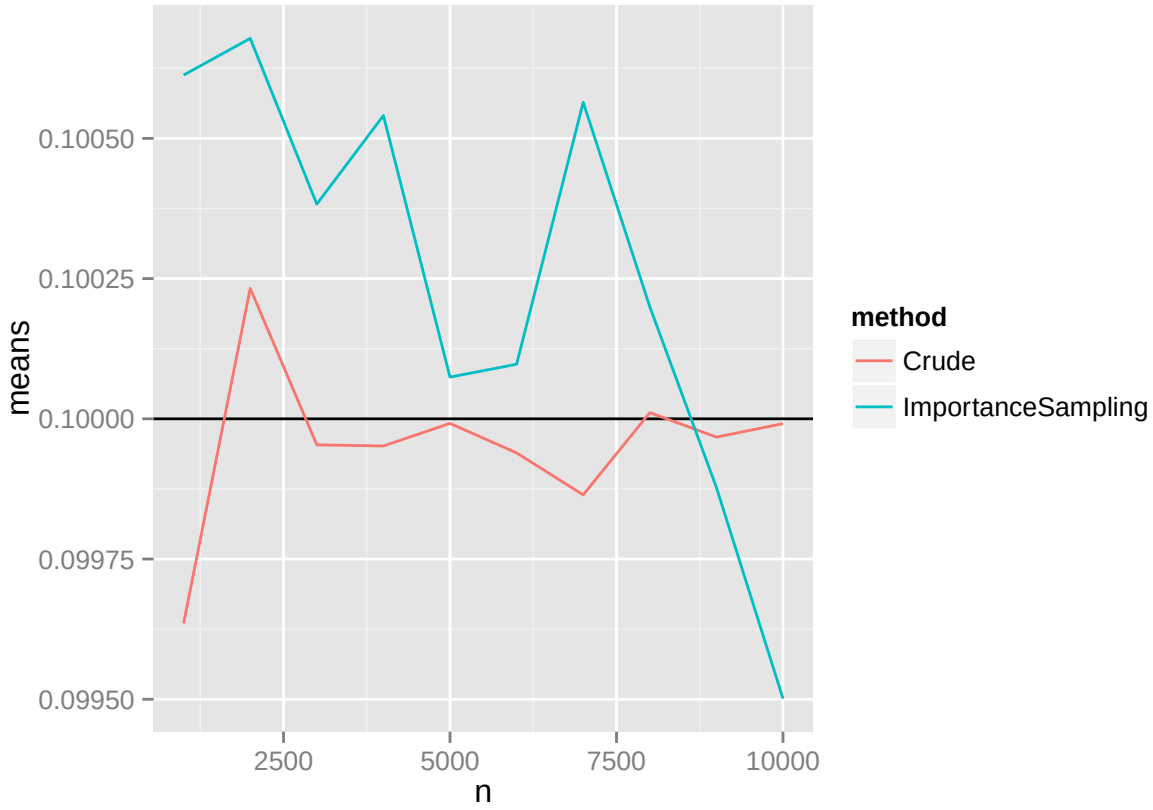
```
rcharleyparabola <- function(n){
  n <- 1000
  k <- 0
  j <- 0
```

```
  y <- numeric(n)

  while(k < n){
    u <- runif(1)
    j <- j+1
    x <- runif(1,min=-5,max=5)
    if(-(x-5)*(x+5)/25 > u){
      k <- k+1
      y[k] <- x
    }
  }

  return(y)

}
```

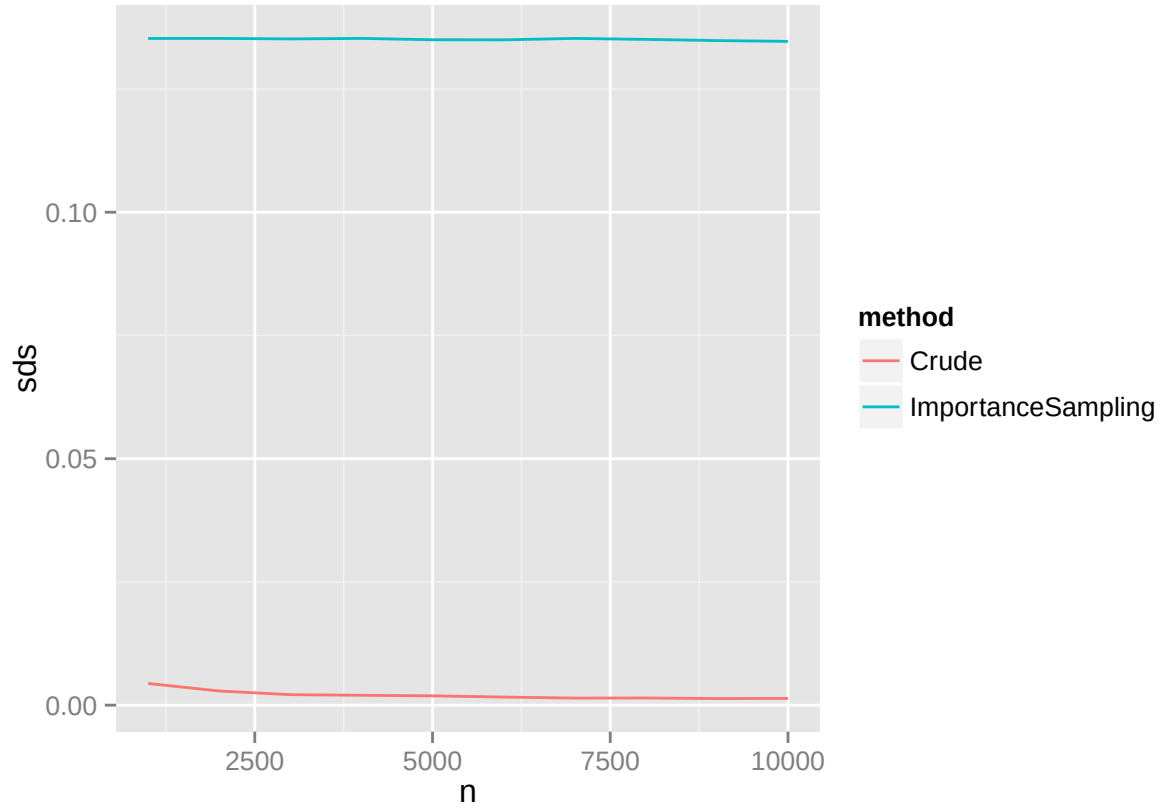
So lets perform our importance sampling:

```
isfunction <- function(n,D){
  m <- n*D
  x <- matrix(rcharleyparabola(m),nrow=n)
  fg <- apply(x,1,cofx,D=D) / (-3*(x-5)*(x+5)/500)

  return(fg)
}

iscreate <- function(D){

  n <- seq(1000,10000,by=1000)
  means <- c()
  sds <- c()
  coefvars <- c()

  for(i in n){
    thetasample <- replicate(100,atfunction(i,D))
    means <- c(means, mean(thetasample))
    sds <- c(sds, sd(thetasample))
    coefvars <- c(coefvars, mean(thetasample)/sd(thetasample))
  }

  table <- data.frame(n=n, means=means, sds=sds, coefvars=coefvars, D=D,
                      method="ImportanceSampling")

  return(table)

}

table <- rbind(table,iscreate(1),iscreate(2))
```

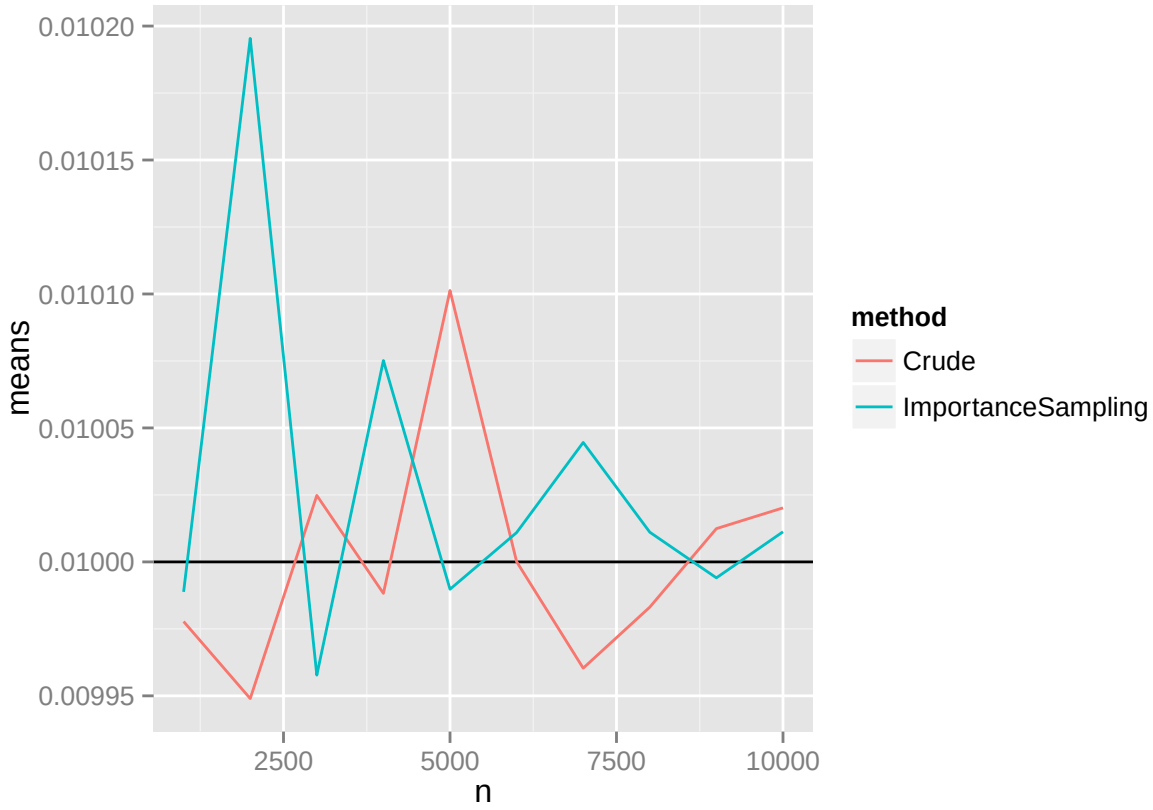
```
ggplot(filter(table,method %in% c("Crude","ImportanceSampling"),D==1),
       aes(x=n,y=means,color=method)) +
  geom_hline(yintercept=(1/10)) +
  geom_line()
```



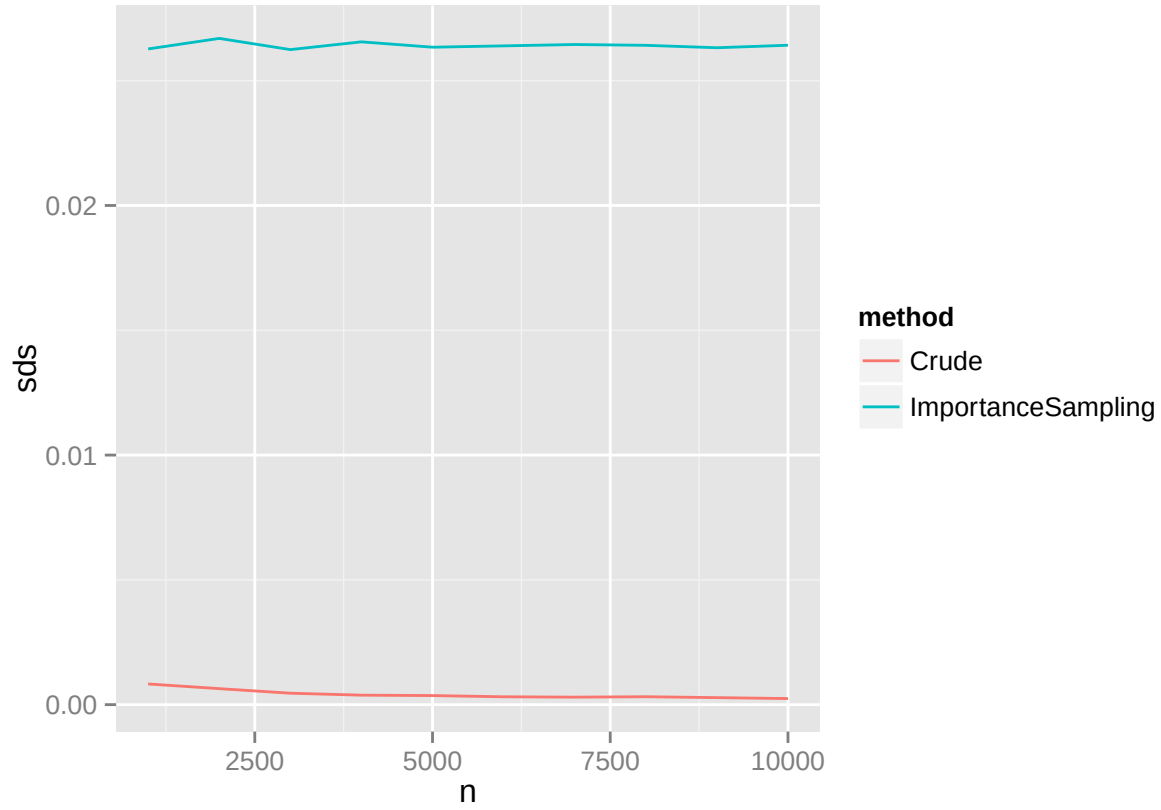

```
ggplot(filter(table,method %in% c("Crude","ImportanceSampling"),D==1),
       aes(x=n,y=sds,color=method)) + geom_line()
```




```
ggplot(filter(table,method %in% c("Crude","ImportanceSampling"),D==2),
       aes(x=n,y=means,color=method)) +
  geom_hline(yintercept=(1/10)^2) +
  geom_line()
```




```
ggplot(filter(table,method %in% c("Crude","ImportanceSampling"),D==2),
       aes(x=n,y=sds,color=method)) + geom_line()
```



Importance Sampling seemed to have not worked in the case I used. Looking online, I found a few interesting takes on how to pick an f(x) to use as your distribution. A few common themes I found was to pick an f(x) greater than the source distribution, and to have it be a similar shape, which is the direction I chose when using this parabola. The slides, and a few other sources I found, seemed better geared towards estimating a certain portion of a distribution. In the normal's case, the tails. There, the goal seemed to choose your f(x) to oversample from the tails, or from the area of interest. If I were looking at a certain portion of my distribution, I'm sure I would be able to find an interesting way to use importance sampling to reduce my variance, or find out what's happening in tail events.

f. Summary

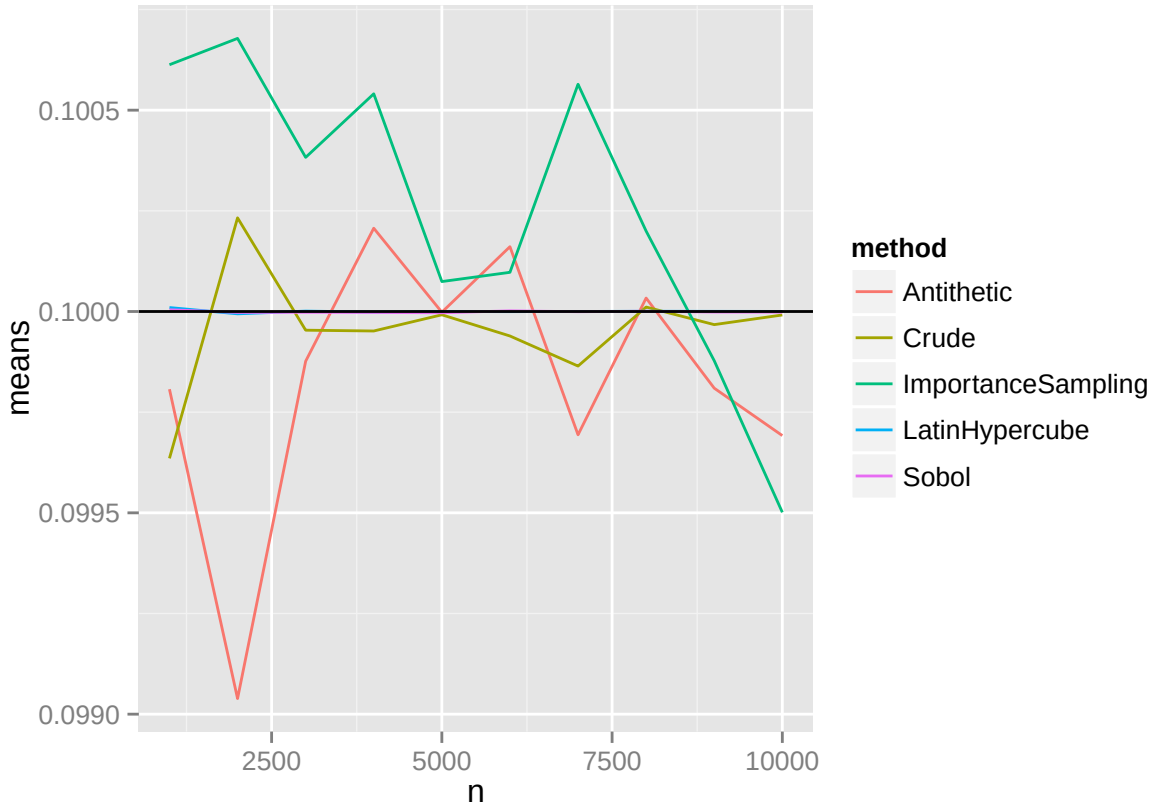
```
table
```

```
##       X     n       means         sds     coefvars D            method
## 1     1  1000 0.099635264 0.004397766 2.265588e+01 1             Crude
## 2     2  2000 0.100232606 0.002871483 3.490622e+01 1             Crude
## 3     3  3000 0.099953542 0.002126760 4.699804e+01 1             Crude
## 4     4  4000 0.099951575 0.001999533 4.998747e+01 1             Crude
## 5     5  5000 0.099991865 0.001887441 5.297749e+01 1             Crude
## 6     6  6000 0.099939194 0.001638776 6.098406e+01 1             Crude
## 7     7  7000 0.099864479 0.001431882 6.974352e+01 1             Crude
## 8     8  8000 0.100011060 0.001452489 6.885496e+01 1             Crude
## 9     9  9000 0.099967395 0.001336098 7.482039e+01 1             Crude
## 10   10 10000 0.099991417 0.001368016 7.309231e+01 1             Crude
```

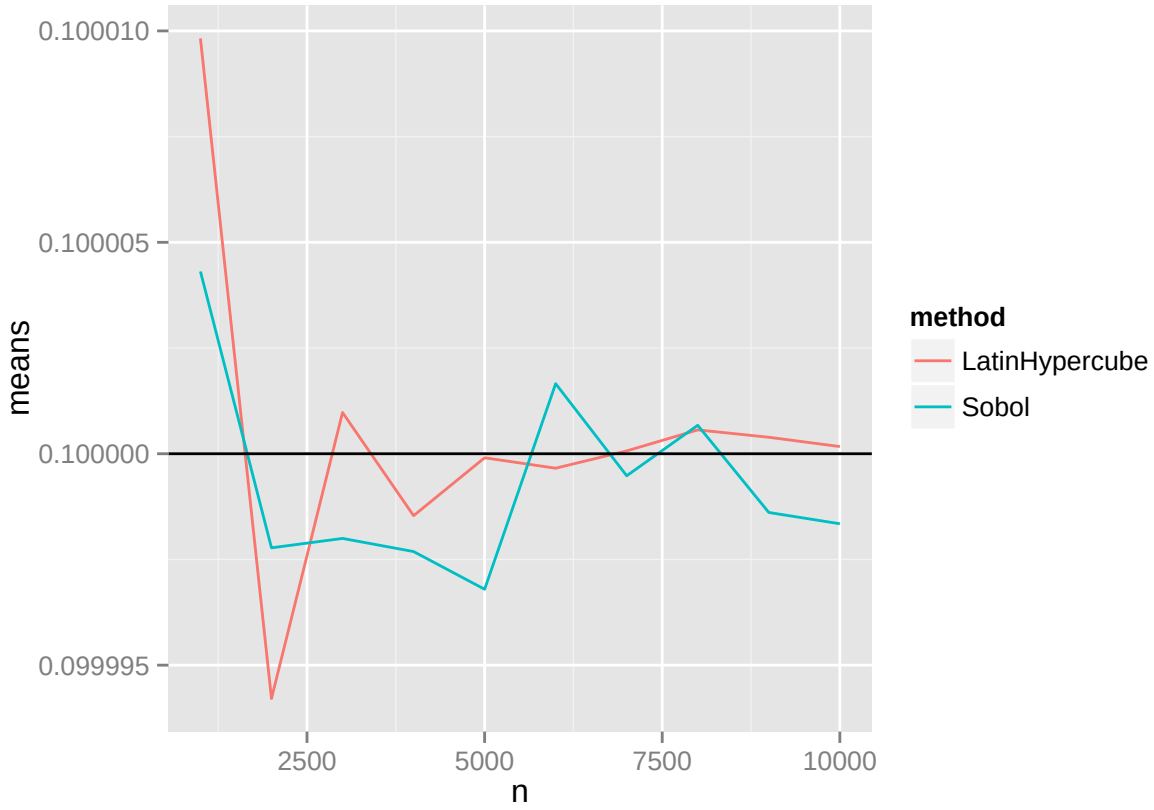
```
## 11   11  1000 0.009977747 0.000828539 1.204258e+01 2             Crude
## 12   12  2000 0.009948986 0.000640998 1.552109e+01 2             Crude
## 13   13  3000 0.010024813 0.000458036 2.188649e+01 2             Crude
## 14   14  4000 0.009988249 0.000380402 2.625709e+01 2             Crude
## 15   15  5000 0.010101302 0.000362914 2.783389e+01 2             Crude
## 16   16  6000 0.010000109 0.000314704 3.177629e+01 2             Crude
## 17   17  7000 0.009960331 0.000300298 3.316816e+01 2             Crude
## 18   18  8000 0.009983117 0.000316730 3.151936e+01 2             Crude
## 19   19  9000 0.010012414 0.000280354 3.571352e+01 2             Crude
## 20   20 10000 0.010020163 0.000243436 4.116146e+01 2             Crude
## 21   21  1000 0.100004311 0.000142771 7.004525e+02 1             Sobol
## 22   22  2000 0.099997775 0.000049500 2.020665e+03 1             Sobol
## 23   23  3000 0.099997997 0.000041900 2.386884e+03 1             Sobol
## 24   24  4000 0.099997687 0.000016400 6.107121e+03 1             Sobol
## 25   25  5000 0.099996795 0.000021600 4.627258e+03 1             Sobol
## 26   26  6000 0.100001657 0.000015500 6.465671e+03 1             Sobol
## 27   27  7000 0.099999479 0.000020500 4.878611e+03 1             Sobol
## 28   28  8000 0.100000673 0.000005820 1.717088e+04 1             Sobol
## 29   29  9000 0.099998611 0.000015100 6.611622e+03 1             Sobol
## 30   30 10000 0.099998344 0.000008640 1.157017e+04 1             Sobol
## 31   31  1000 0.010023266 0.000081300 1.233378e+02 2             Sobol
## 32   32  2000 0.009996934 0.000045700 2.189181e+02 2             Sobol
## 33   33  3000 0.010002702 0.000025500 3.927803e+02 2             Sobol
## 34   34  4000 0.009998124 0.000019500 5.122297e+02 2             Sobol
## 35   35  5000 0.009996124 0.000014200 7.027491e+02 2             Sobol
## 36   36  6000 0.010002708 0.000016400 6.084194e+02 2             Sobol
## 37   37  7000 0.010004332 0.000012900 7.764598e+02 2             Sobol
## 38   38  8000 0.009999427 0.000009020 1.108112e+03 2             Sobol
## 39   39  9000 0.009999902 0.000011400 8.797519e+02 2             Sobol
## 40   40 10000 0.010000608 0.000008370 1.194431e+03 2             Sobol
## 41   41  1000 0.099807012 0.135036719 7.391102e-01 1        Antithetic
## 42   42  2000 0.099038729 0.134485622 7.364262e-01 1        Antithetic
## 43   43  3000 0.099876522 0.134899370 7.403780e-01 1        Antithetic
## 44   44  4000 0.100206797 0.135078138 7.418432e-01 1        Antithetic
## 45   45  5000 0.099998219 0.134983193 7.408198e-01 1        Antithetic
## 46   46  6000 0.100160958 0.135102499 7.413701e-01 1        Antithetic
## 47   47  7000 0.099693817 0.134830332 7.394020e-01 1        Antithetic
## 48   48  8000 0.100033388 0.135032824 7.408079e-01 1        Antithetic
## 49   49  9000 0.099809595 0.134829242 7.402667e-01 1        Antithetic
## 50   50 10000 0.099691844 0.134647159 7.403932e-01 1        Antithetic
## 51   51  1000 0.010135927 0.026573116 3.814354e-01 2        Antithetic
## 52   52  2000 0.009975495 0.026366456 3.783404e-01 2        Antithetic
## 53   53  3000 0.010006497 0.026361678 3.795850e-01 2        Antithetic
## 54   54  4000 0.009985600 0.026330874 3.792354e-01 2        Antithetic
## 55   55  5000 0.009984758 0.026373852 3.785855e-01 2        Antithetic
## 56   56  6000 0.010055984 0.026452578 3.801514e-01 2        Antithetic
## 57   57  7000 0.009955548 0.026271310 3.789513e-01 2        Antithetic
## 58   58  8000 0.010015293 0.026403800 3.793126e-01 2        Antithetic
## 59   59  9000 0.009989131 0.026370245 3.788031e-01 2        Antithetic
## 60   60 10000 0.009943457 0.026260634 3.786450e-01 2        Antithetic
## 61   61  1000 0.100009823 0.000103688 9.645252e+02 1    LatinHypercube
## 62   62  2000 0.099994210 0.000039100 2.560340e+03 1    LatinHypercube
## 63   63  3000 0.100000976 0.000021200 4.723296e+03 1    LatinHypercube
## 64   64  4000 0.099998535 0.000013400 7.442787e+03 1    LatinHypercube
```

```
## 65   65  5000 0.099999905 0.000008890 1.125056e+04 1     LatinHypercube
## 66   66  6000 0.099999658 0.000006620 1.509549e+04 1     LatinHypercube
## 67   67  7000 0.100000067 0.000005850 1.710342e+04 1     LatinHypercube
## 68   68  8000 0.100000564 0.000005000 1.998476e+04 1     LatinHypercube
## 69   69  9000 0.100000390 0.000004250 2.352161e+04 1     LatinHypercube
## 70   70 10000 0.100000170 0.000003600 2.780011e+04 1     LatinHypercube
## 71   71  1000 0.010024462 0.000634826 1.579089e+01 2     LatinHypercube
## 72   72  2000 0.009996355 0.000475467 2.102430e+01 2     LatinHypercube
## 73   73  3000 0.009995389 0.000328816 3.039811e+01 2     LatinHypercube
## 74   74  4000 0.009988403 0.000304687 3.278253e+01 2     LatinHypercube
## 75   75  5000 0.009999932 0.000269193 3.714782e+01 2     LatinHypercube
## 76   76  6000 0.010008599 0.000228104 4.387733e+01 2     LatinHypercube
## 77   77  7000 0.010035040 0.000235057 4.269201e+01 2     LatinHypercube
## 78   78  8000 0.009999550 0.000191830 5.212722e+01 2     LatinHypercube
## 79   79  9000 0.010003123 0.000204571 4.889813e+01 2     LatinHypercube
## 80   80 10000 0.010017636 0.000172617 5.803402e+01 2     LatinHypercube
## 81   81  1000 0.100612862 0.135254260 7.438794e-01 1 ImportanceSampling
## 82   82  2000 0.100678232 0.135263781 7.443104e-01 1 ImportanceSampling
## 83   83  3000 0.100382787 0.135163235 7.426782e-01 1 ImportanceSampling
## 84   84  4000 0.100540632 0.135263713 7.432935e-01 1 ImportanceSampling
## 85   85  5000 0.100074375 0.134985531 7.413711e-01 1 ImportanceSampling
## 86   86  6000 0.100097305 0.134968815 7.416328e-01 1 ImportanceSampling
## 87   87  7000 0.100564306 0.135264388 7.434648e-01 1 ImportanceSampling
## 88   88  8000 0.100199740 0.135064158 7.418677e-01 1 ImportanceSampling
## 89   89  9000 0.099877840 0.134824855 7.407969e-01 1 ImportanceSampling
## 90   90 10000 0.099501225 0.134660455 7.389046e-01 1 ImportanceSampling
## 91   91  1000 0.009988807 0.026272198 3.802045e-01 2 ImportanceSampling
## 92   92  2000 0.010195393 0.026694636 3.819267e-01 2 ImportanceSampling
## 93   93  3000 0.009957735 0.026245194 3.794118e-01 2 ImportanceSampling
## 94   94  4000 0.010075155 0.026556616 3.793840e-01 2 ImportanceSampling
## 95   95  5000 0.009989793 0.026342264 3.792306e-01 2 ImportanceSampling
## 96   96  6000 0.010010938 0.026395625 3.792651e-01 2 ImportanceSampling
## 97   97  7000 0.010044576 0.026449498 3.797643e-01 2 ImportanceSampling
## 98   98  8000 0.010011081 0.026417604 3.789549e-01 2 ImportanceSampling
## 99   99  9000 0.009994043 0.026320616 3.797040e-01 2 ImportanceSampling
## 100 100 10000 0.010011238 0.026420750 3.789157e-01 2 ImportanceSampling
```

```
ggplot(filter(table,D==1),aes(x=n,y=means,color=method)) + geom_line() +
  geom_hline(yintercept=(1/10))
```

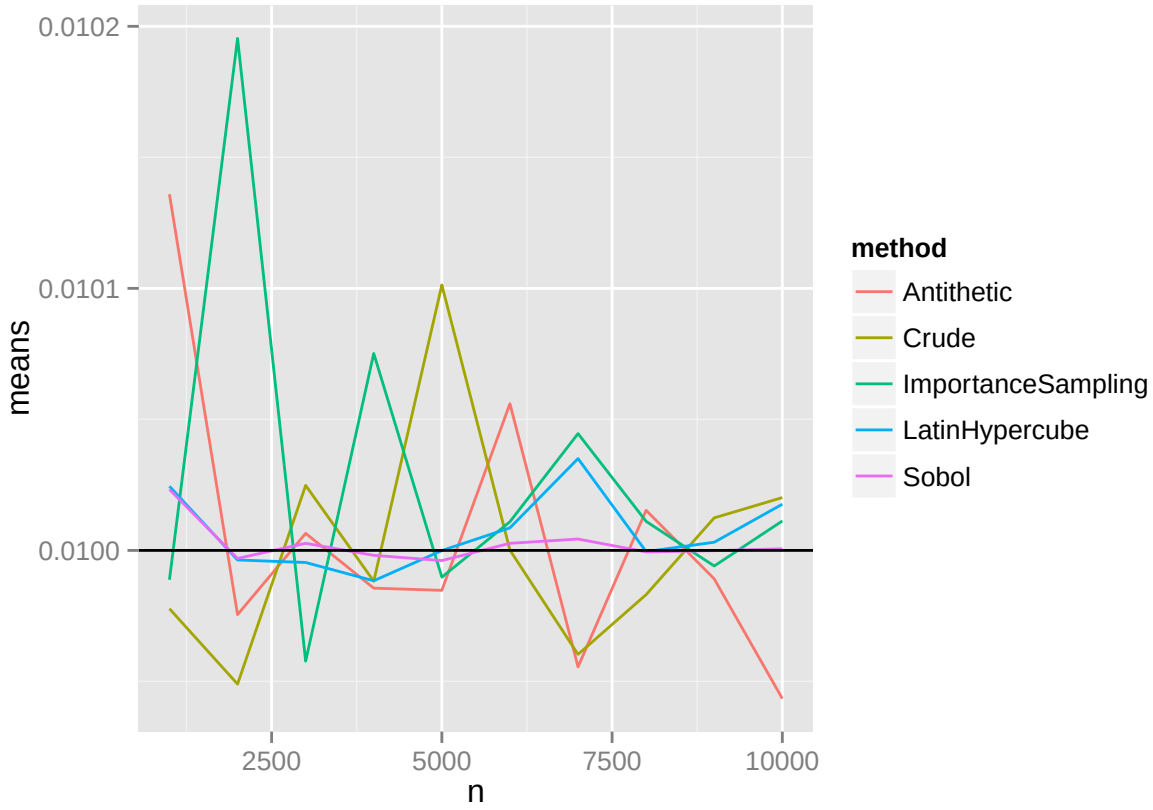



```
ggplot(filter(table,method %in% c("Sobol", "LatinHypercube"),D==1),
       aes(x=n,y=means,color=method)) + geom_line() +
  geom_hline(yintercept=(1/10))
```



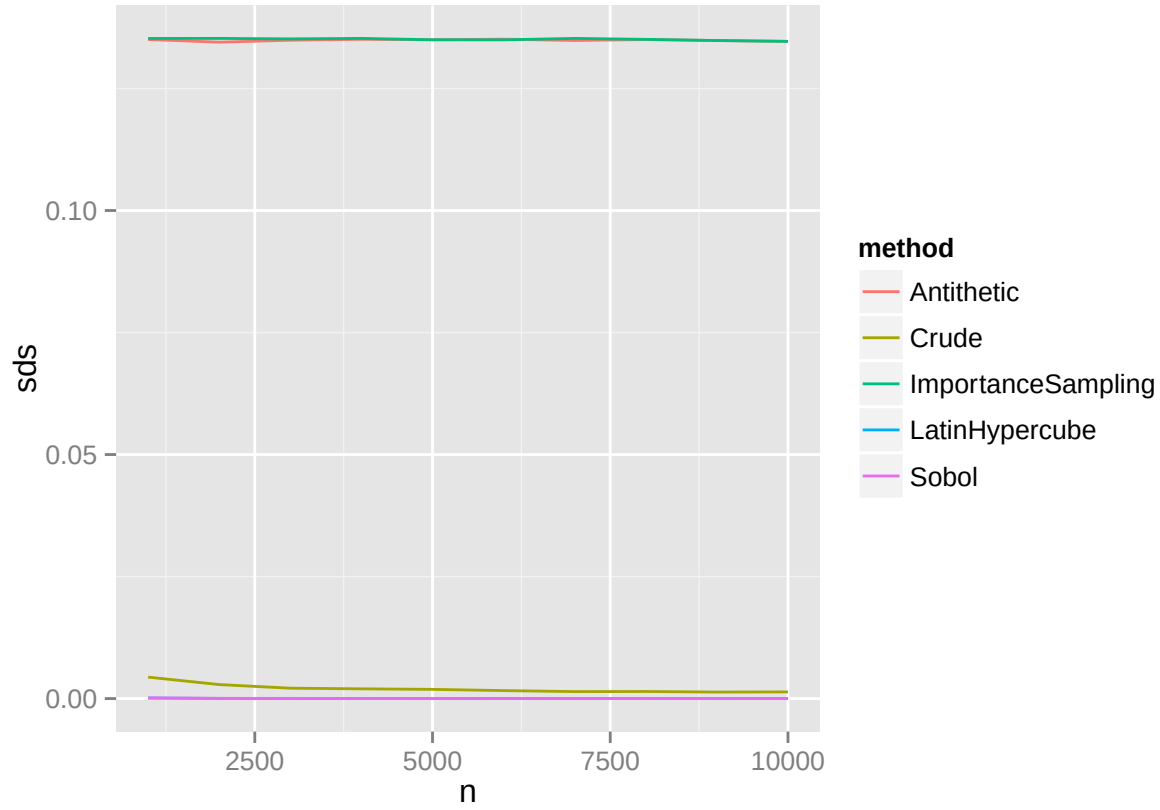
For D==1, It seems Sobol and Latin Hypercube were the best at guessing the mean. The Latin Hypercube method seemed to work better for higher samples, while the Sobol method seemed to perform steadily throughout.

```
ggplot(filter(table,D==2),aes(x=n,y=means,color=method)) + geom_line() +
  geom_hline(yintercept=(1/10)^2)
```

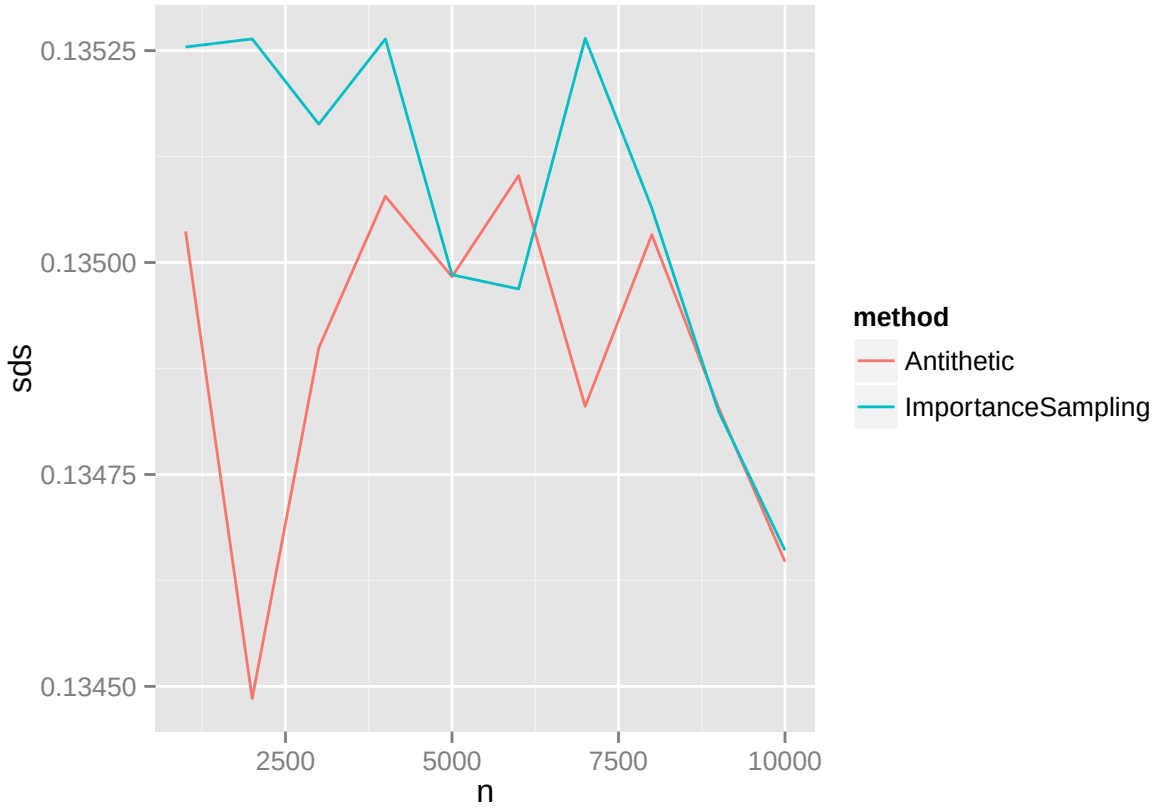


Oddly enough, for D==2, The Sobol and Latin Hypercube method seemed to perform less well versus other methods. They were still the best at making the prediction however.

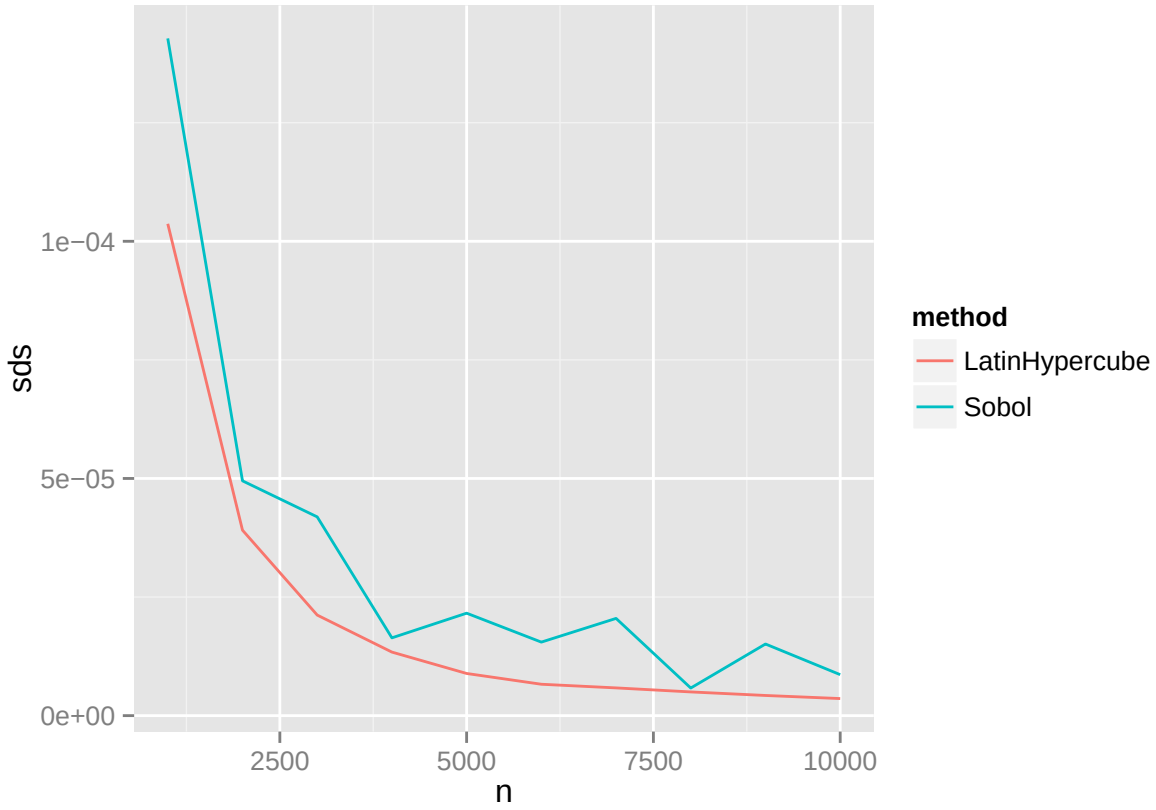
```
ggplot(filter(table,D==1),aes(x=n,y=sds,color=method)) + geom_line()
```



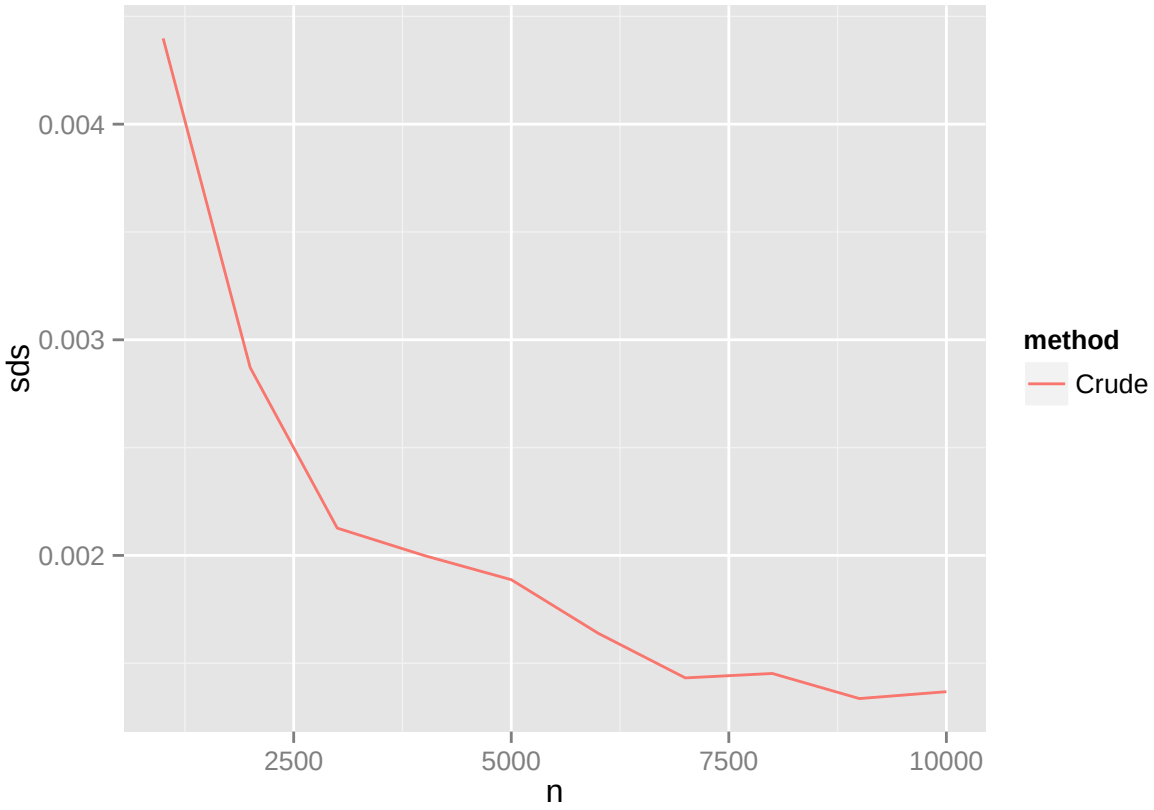

```
ggplot(filter(table,method %in% c("Antithetic","ImportanceSampling"),
              D==1),aes(x=n,y=sds,color=method)) + geom_line()
```




```
ggplot(filter(table,method %in% c("Sobol", "LatinHypercube"),D==1),
       aes(x=n,y=sds,color=method)) + geom_line()
```

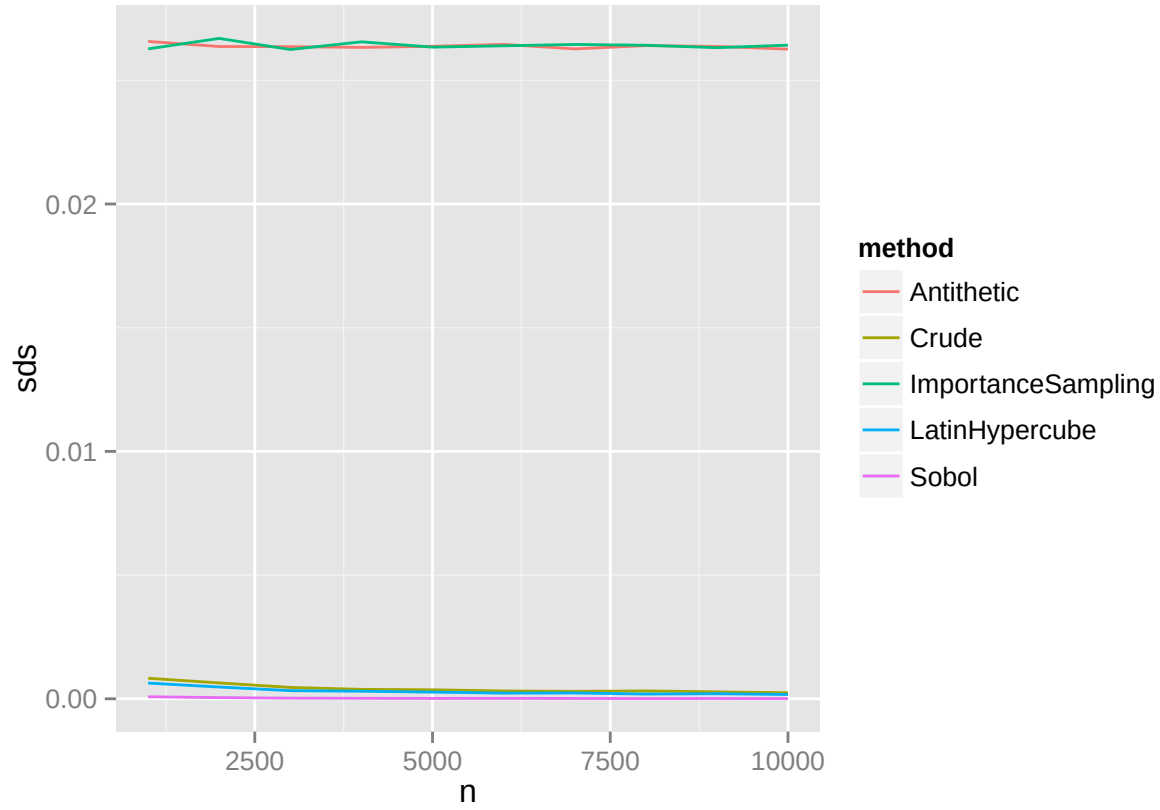



```
ggplot(filter(table,method %in% c("Crude"),D==1),
       aes(x=n,y=sds,color=method)) + geom_line()
```

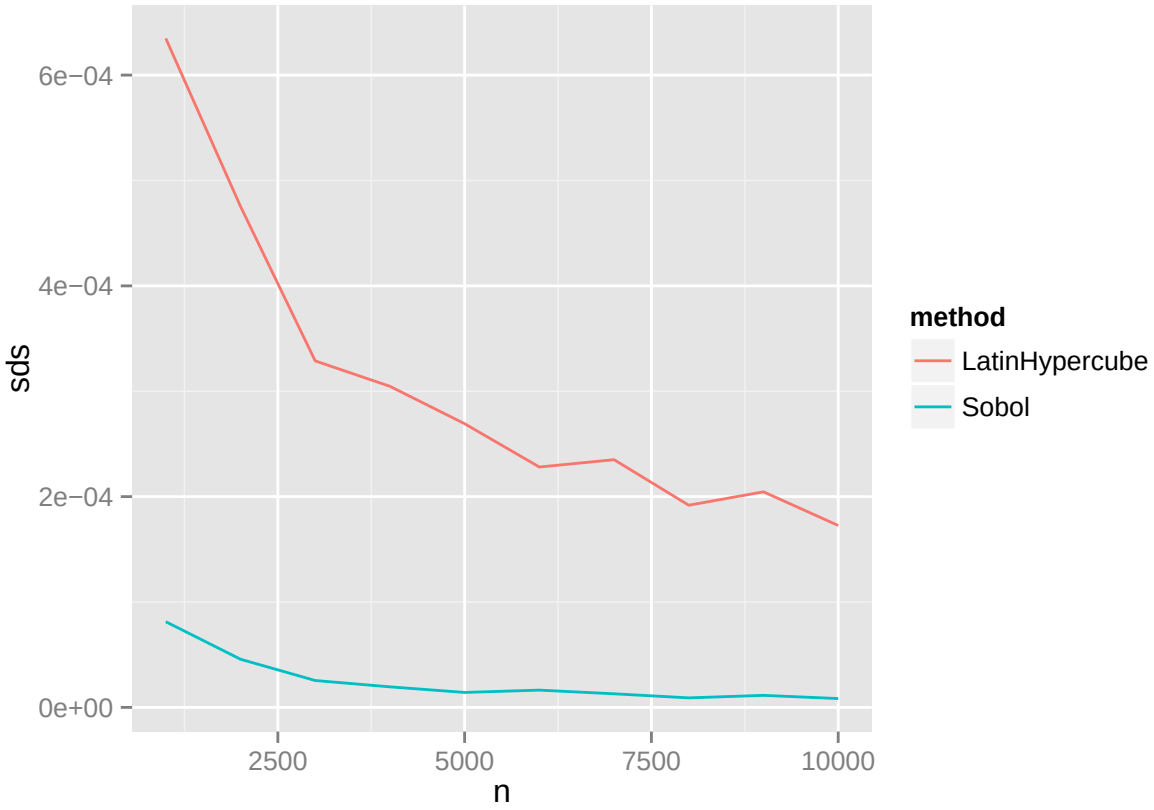


For my standard deviations, Sobol and Latin Hypercube are once again the clear winners, while Antithetic and Importance Sampling methods were significantly higher than the crude.

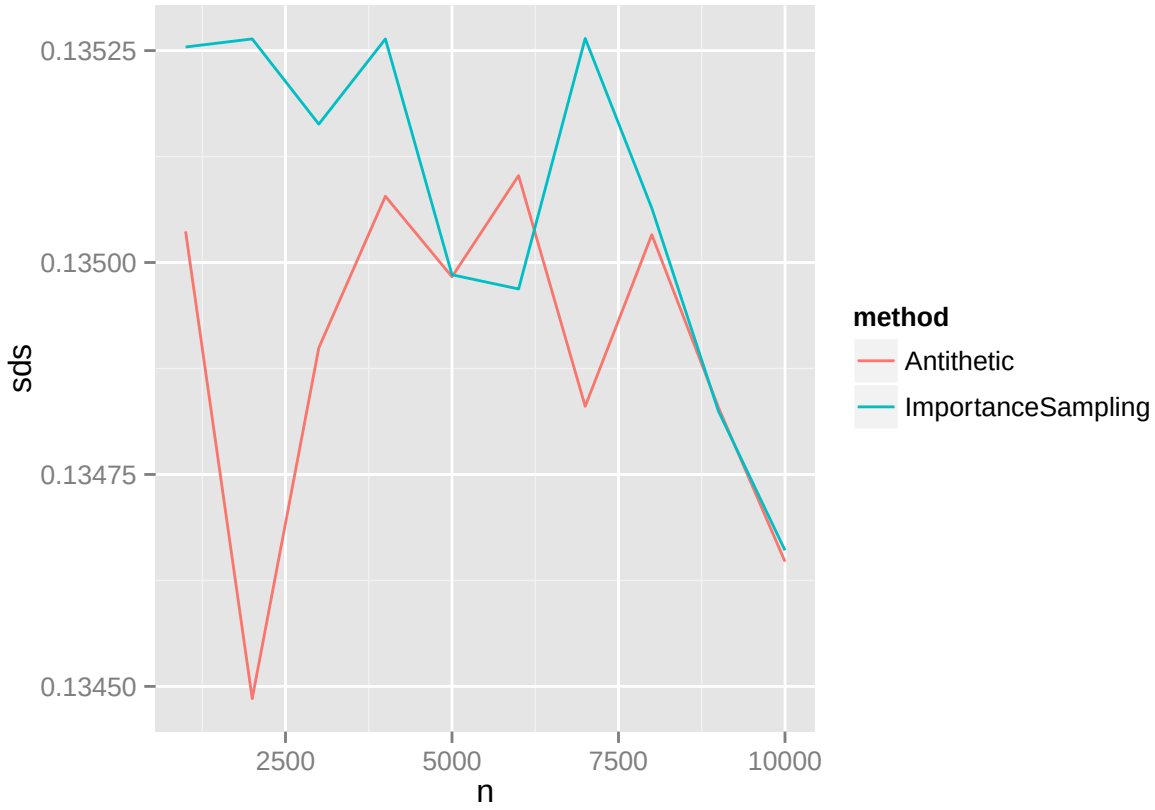
```
ggplot(filter(table,D==2),aes(x=n,y=sds,color=method)) + geom_line()
```



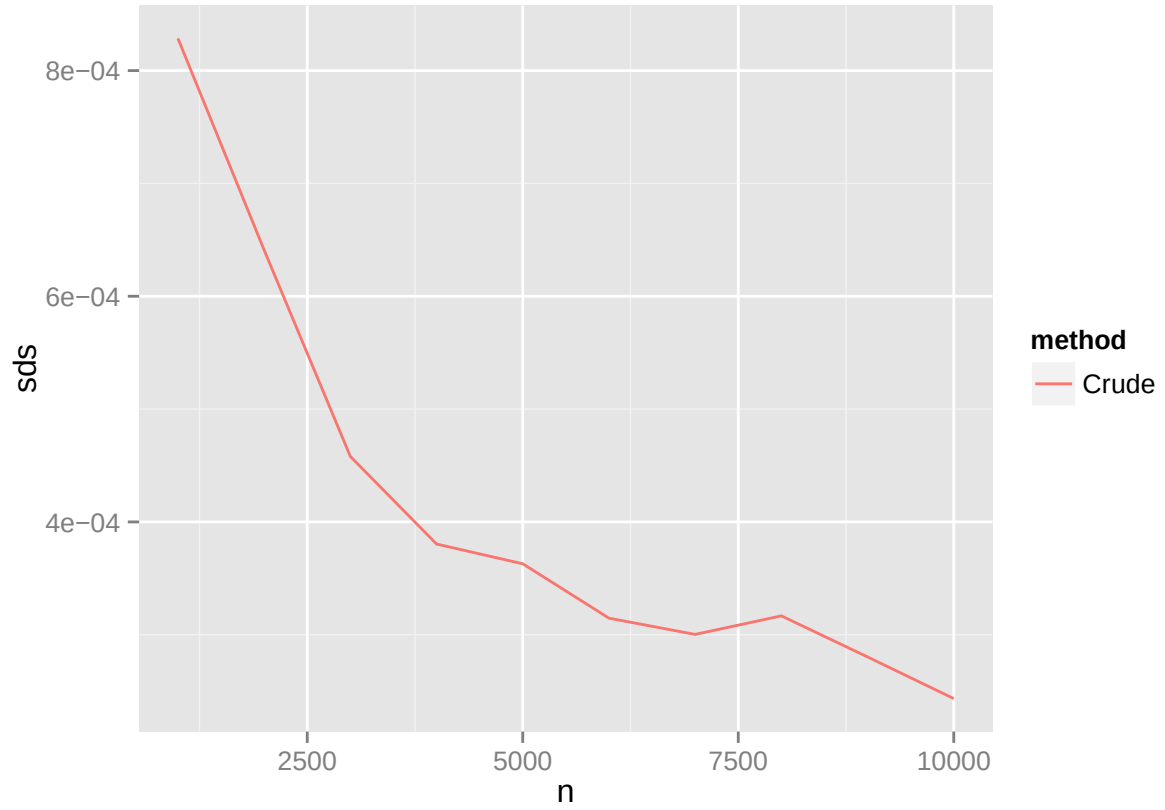

```
ggplot(filter(table,method %in% c("Sobol", "LatinHypercube"),D==2),
       aes(x=n,y=sds,color=method)) + geom_line()
```




```
ggplot(filter(table,method %in% c("Antithetic","ImportanceSampling"),
              D==1),aes(x=n,y=sds,color=method)) + geom_line()
```




```
ggplot(filter(table,method %in% c("Crude"),D==2),
       aes(x=n,y=sds,color=method)) + geom_line()
```



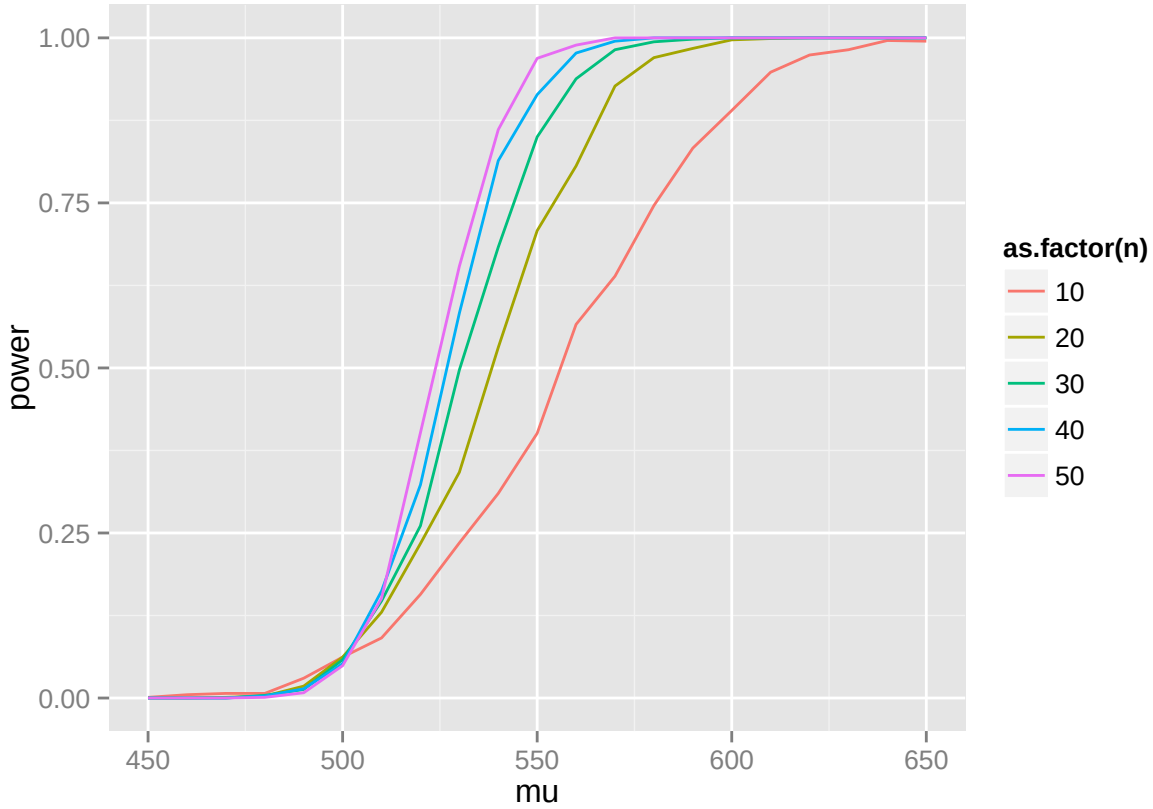
For D==2, the postiions of Sobol and the Latin Hypercube switcch positions. It would appear the Sobol estimation algorithm is better at estimating multi- dimensional data.

SCR Problem 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 graph, each in a different color or different line type, and include a legend. Comment on the relation between power and sample size.

```
n <- c(10,20,30,40,50)
m <- 1000
mu0 <- 500
sigma <- 100
mu <- c(seq(450,650,10))
M <- length(mu)
power <- c()
for(j in n){

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

```
    })
    power <- c(power,mean(pvalues<=0.05))
  }
}

table <- expand.grid(mu,n)
colnames(table) <- c("mu","n")
table$power <- power

power <- data.frame(power)

ggplot(table,aes(x=mu,y=power,color=as.factor(n))) + geom_line()
```



The lines all seem to diverge at around $\mu = 500$, with the power ~0.05. This makes sense, we set our α significance level to 0.05, and the true mean being sampled is 500. Under perfect conditions, we'd expect the function to shoot immediately to 1 once we get past 500. Obviously, we get differing slopes towards 1. As the sample size increases, the function approaches 1 more quickly. This also makes sense, our standard error decreases and n increases, and a low standard error would mean less of a chance of accepting a null hypothesis for a value of μ drastically greater than 500.

SCR Problem 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 μ. Use a Monte Carlo method to obtain an empirical estimate of the confidence level

If we don't know the underlying distribution, then no matter what the distribution is, $\bar{x}$ will be normally distributed, with a standard error $\frac{\sigma}{\sqrt{n}}$

For a two tailed 95% confidence interval, we'd use z=1.96. So:

$$\bar{x} \pm \frac{\sigma}{\sqrt{n}}$$

Here's an example using the default settings on the lognormal distribution (which are set as meanlog and sdlog)

```
x <- rlnorm(1000)

xbar <- mean(x)

sdbar <- sd(x)

se <- sdbar/sqrt(n)

CI <- c(xbar-se,xbar+se)

CI
```

```
##  [1] 0.9588149 1.1372240 1.2162621 1.2633783 1.2955319 2.1770684 1.9986593
##  [8] 1.9196211 1.8725050 1.8403514
```

SCR Problem 7.1

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

```
n <- nrow(law)
theta.hat <- cor(law$LSAT, law$GPA)

theta.jack <- numeric(n)

for(i in 1:n){
  theta.jack[i] <- cor(law$LSAT[-i],law$GPA[-i])
}
se.jack <- sd(theta.jack)

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

se.jack
```

```
## [1] 0.03942659
```

```
bias.jack
```

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

SCR Problem 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 air conditioning equipment. Assume that the times between failures follow an exponential model $Exp(\lambda)$. Obtain the MLE of the hazard rate λ and use bootstrap to estimate the bias and standard error of the estimate.

The maximum likelihood estimator for λ is:

$$L(\theta, x_i) = \prod_{i=1}^{n} f(\theta, x_i) = \prod_{i=1}^{n} \lambda e^{-\lambda x_i}$$

$$L(\lambda, x_i) = \lambda^n e^{-\lambda \sum_{t=1}^{n} x_t}$$

The log-likelihood, taking the log of both sides for easier calculations, is:

$$l(\theta, x_i) = n * ln(\lambda) - \lambda \sum_{j=1}^{n} x_i$$

When trying to find the max of this function in terms of θ, so lets take the derivative $\frac{d}{d\theta}$

$$\frac{d}{d\lambda} l(\lambda, x_i) = \frac{n}{\lambda} - \sum_{j=1}^{n} x_i$$

$$\hat{\lambda} = \frac{n}{\sum_{i=0}^{n} x_i}$$

```
lambda.hat <- length(aircondit$hours)/sum(aircondit$hours)

B <- B <- 200
n <- nrow(aircondit)
R <- numeric(B)

for(b in 1:B){
  acsamp <- sample(aircondit$hours,n,replace=TRUE)
  R[b] <- length(acsamp)/sum(acsamp)

}

theta.boot <- mean(R)

lambda.hat
```

```
## [1] 0.00925212
```

```
theta.boot
```

```
## [1] 0.0109454
```