Homework 4

Max Wagner

March 15, 2016

Before anything else, I'll write out a function that performs the cost function, and returns.

```
cost <- function(x, D) {
  cx <- (1 / ((2*pi) ^ (D/2))) * exp((-.5) * (t(x) %*% x))
  return (cx)
}
```

a. Crude Monte Carlo

Again, I'll start with a function that calculates this so I don't have to. It makes a matrix with the D x n dimensions and uses the `cost` function on the columns.

```
crudemonte <- function(n, D) {
  dims <- n * D
  nums <- runif(dims, -5, 5)
  mtx <- matrix(nums, nrow = n)
  return(mean(apply(mtx, 1, cost, D = D)))
}
```

And now a function to create a table...

```
crudemontetable <- function(D) {
  samps <- seq(1000, 10000, by = 1000) # says 100 times, this only gives 10
  means <- c() # make some empty things to store in later
  stds <- c()
  cvs <- c()

  for (samp in samps) { # loop to get all the samples
    cx <- replicate(100, crudemonte(samp, D))
    means <- c(means, mean(cx))
    stds <- c(stds, sd(cx))
    cvs <- c(cvs, mean(cx)/sd(cx))
  }
  return (data.frame(samples = samps, means = means, std.dev = stds, coef.vars = cvs, D = D, method = "
}
```

And finally lets simulate it. I'll save this to a csv so it doesn't take years to run everytime. Doing this for D = 1 and D = 2

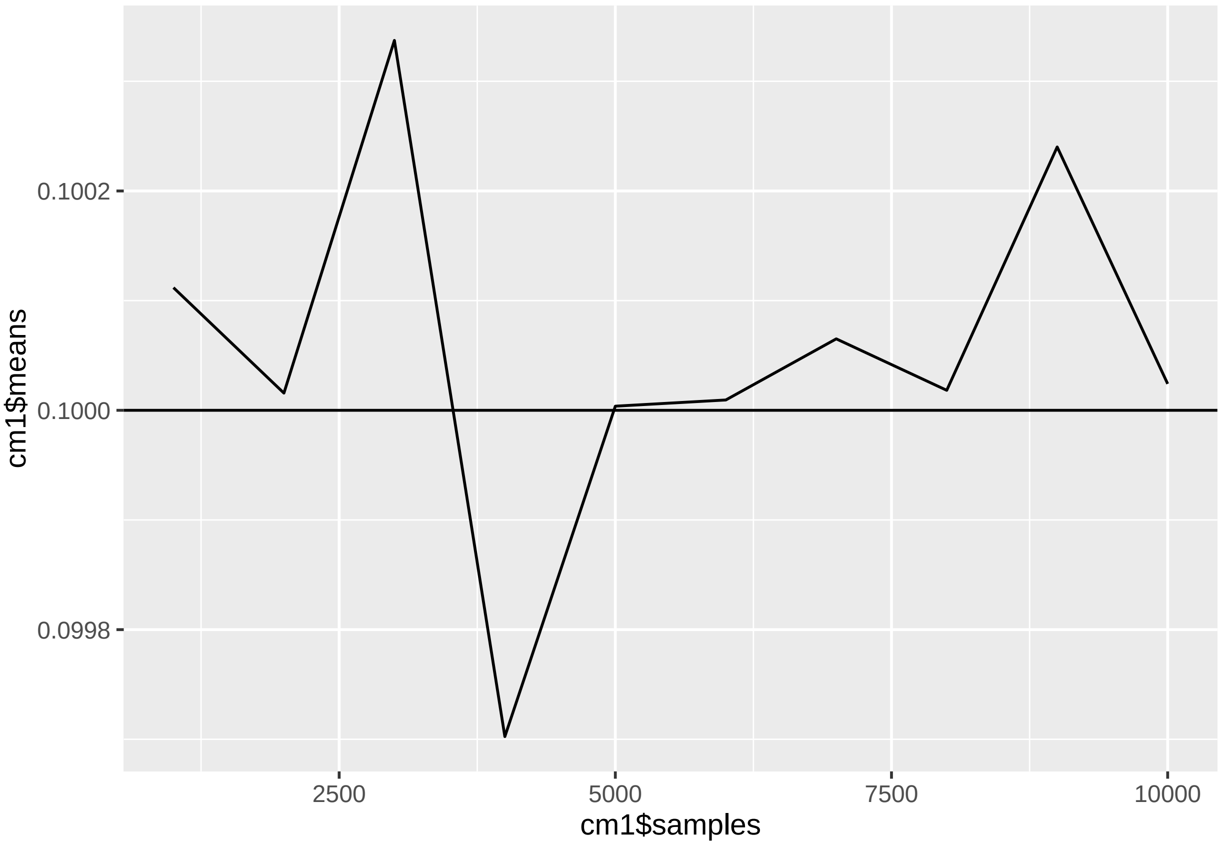
```
write.csv(crudemontetable(1), file = "crudemonte.csv")
write.csv(crudemontetable(2), file = "crudemonte2.csv")
```

```r
cm1 <- read.csv("crudemonte.csv")
cm2 <- read.csv("crudemonte2.csv")
cm1;cm2
```

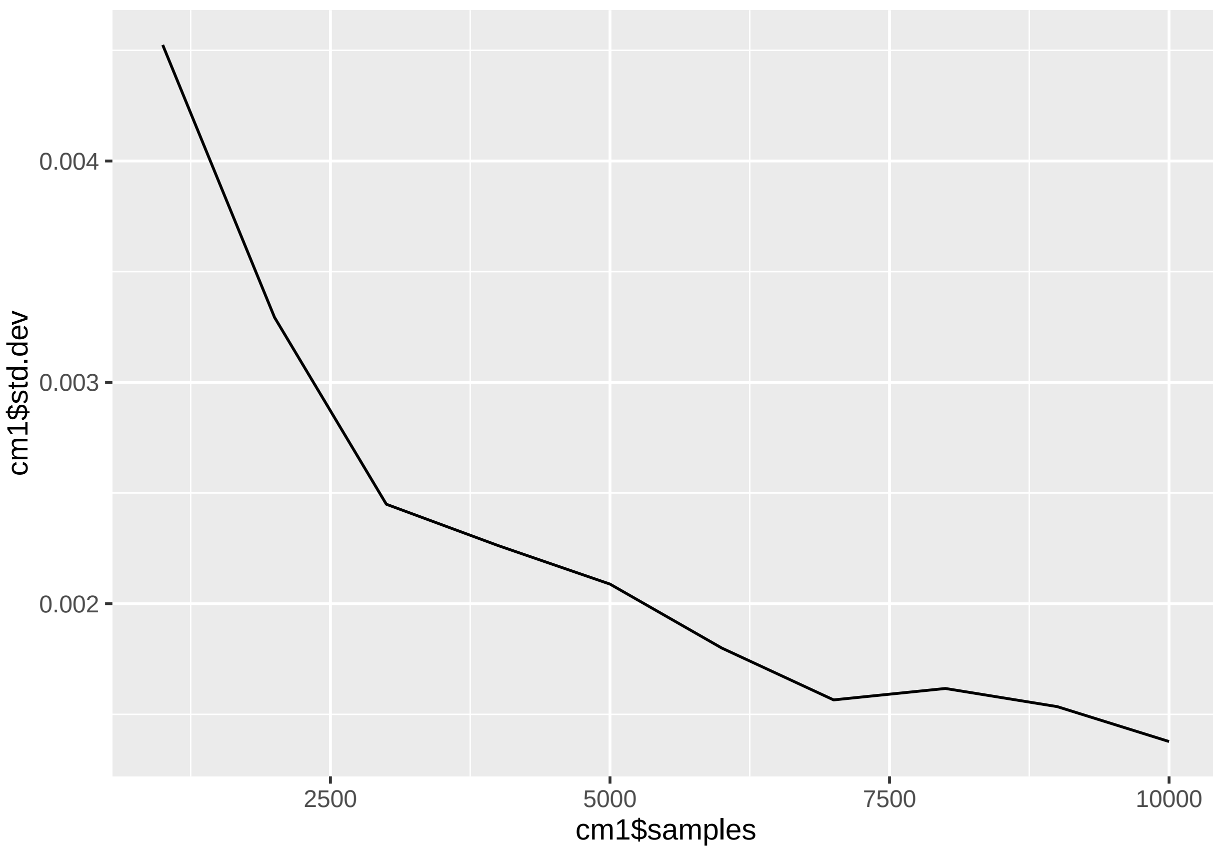
```
##     X samples      means     std.dev coef.vars D      method
## 1   1    1000 0.10011186 0.004524262  22.12778 1 Crude Monte
## 2   2    2000 0.10001567 0.003293458  30.36798 1 Crude Monte
## 3   3    3000 0.10033727 0.002449112  40.96884 1 Crude Monte
## 4   4    4000 0.09970246 0.002262407  44.06920 1 Crude Monte
## 5   5    5000 0.10000376 0.002088350  47.88650 1 Crude Monte
## 6   6    6000 0.10000941 0.001799438  55.57815 1 Crude Monte
## 7   7    7000 0.10006514 0.001565177  63.93216 1 Crude Monte
## 8   8    8000 0.10001823 0.001616733  61.86440 1 Crude Monte
## 9   9    9000 0.10024011 0.001534813  65.31097 1 Crude Monte
## 10 10   10000 0.10002412 0.001377129  72.63234 1 Crude Monte

##     X samples       means      std.dev coef.vars D      method
## 1   1    1000 0.010037234 0.0009122067  11.00324 2 Crude Monte
## 2   2    2000 0.010018370 0.0005626787  17.80478 2 Crude Monte
## 3   3    3000 0.009996352 0.0004954536  20.17616 2 Crude Monte
## 4   4    4000 0.010014905 0.0003650038  27.43781 2 Crude Monte
## 5   5    5000 0.009994480 0.0004130666  24.19580 2 Crude Monte
## 6   6    6000 0.010018711 0.0002969464  33.73913 2 Crude Monte
## 7   7    7000 0.009974660 0.0003286802  30.34762 2 Crude Monte
## 8   8    8000 0.009978739 0.0002646449  37.70615 2 Crude Monte
## 9   9    9000 0.010024854 0.0002908782  34.46410 2 Crude Monte
## 10 10   10000 0.009989636 0.0002429210  41.12298 2 Crude Monte
```

And now for some graphs about the means and sds for both D = 1 and D = 2. In this case both graphs for the means and both graphs for the sd's were fairly similar with no large differences.

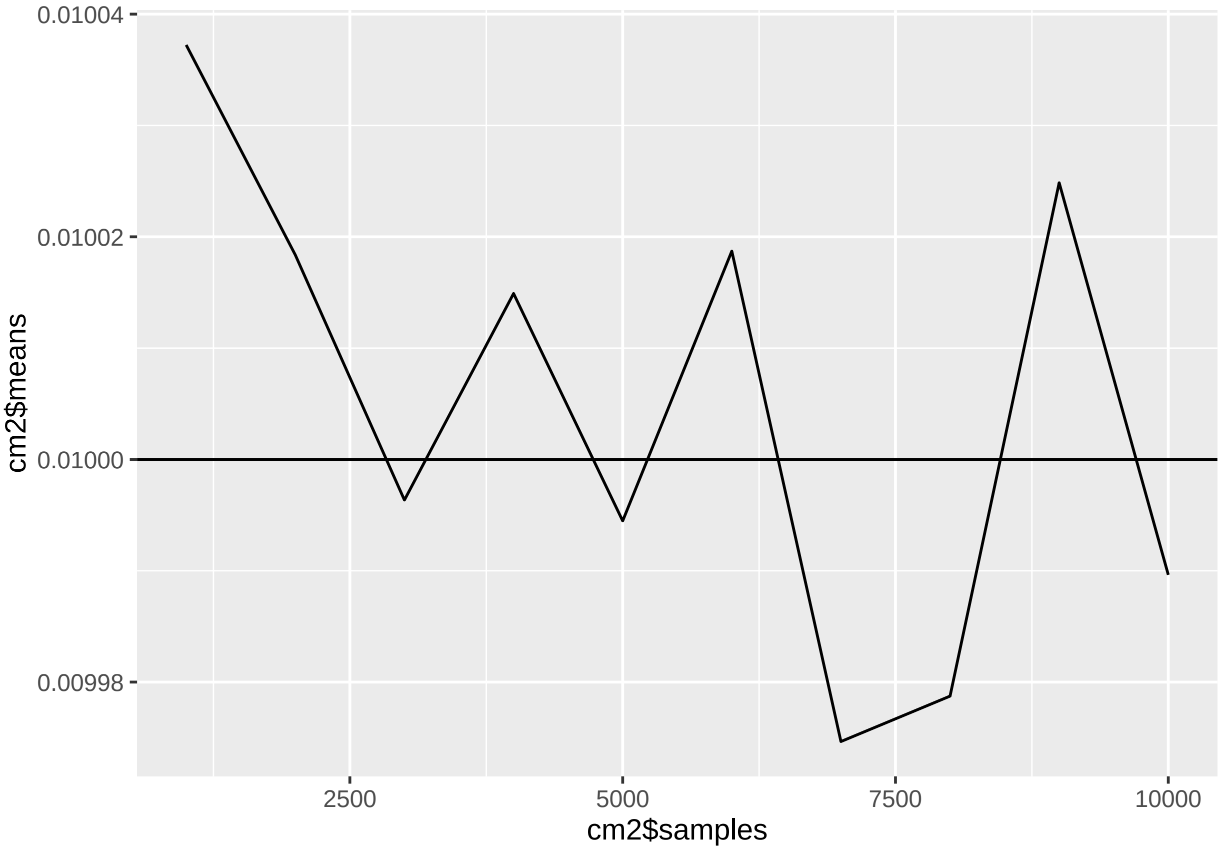
```r
library(ggplot2)
qplot(cm1$samples,cm1$means, geom = "line") + geom_hline(yintercept = (1/10))
```



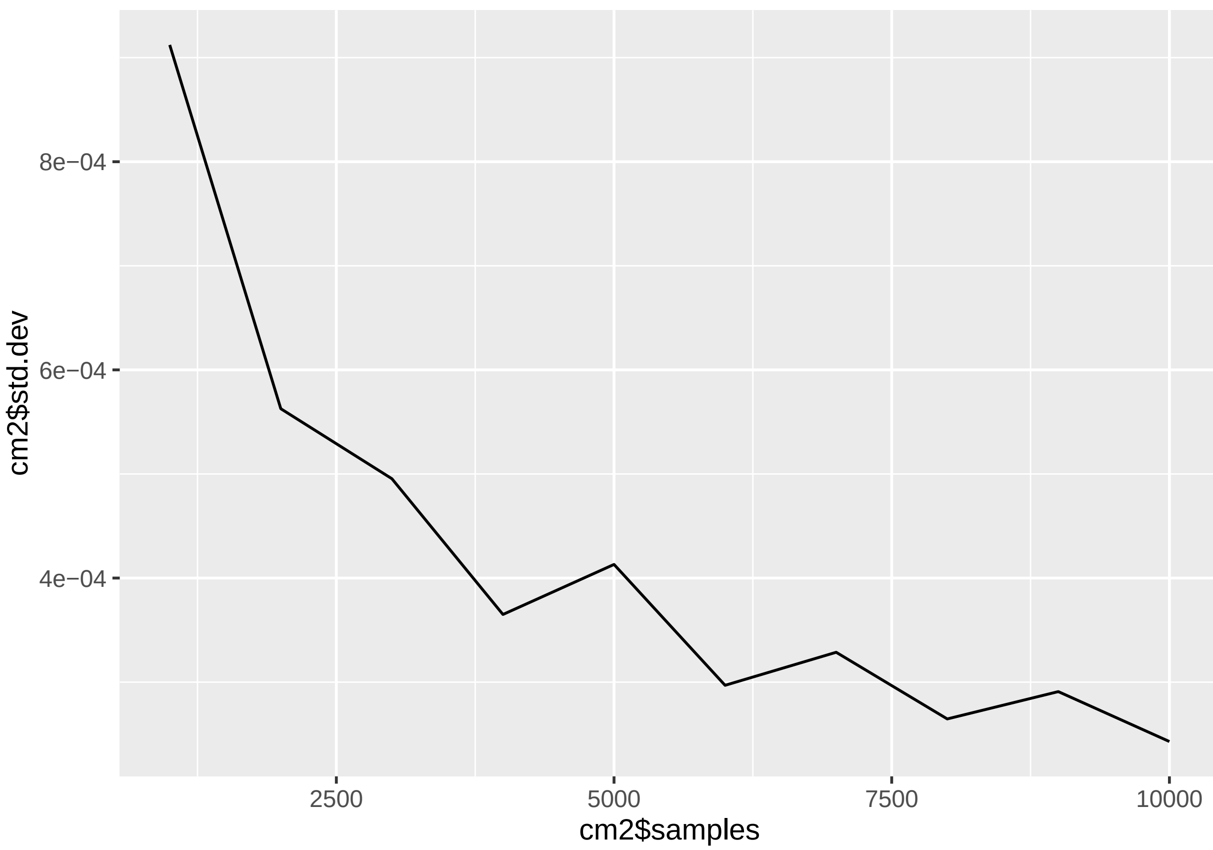

```
qplot(cm1$samples,cm1$std.dev, geom = "line")
```




```
qplot(cm2$samples,cm2$means, geom = "line") + geom_hline(yintercept = (1/10)^2)
```




```
qplot(cm2$samples,cm2$std.dev, geom = "line")
```



b. Quasi Random Numbers

I have an idea how the Sobol numbers work, but I have zero idea of how to code it, looking forward to seeing the answers for this week.

c. Antithetic Variates

The first function generates samples, sort of ripped from the montecarlo section, but with a few extra steps for splitting into fx1 and fx2.

```
anti <- function(n, D) {
  dims <- n * D
  nums <- runif(dims/2)
  mtx1 <- matrix(nums, nrow = dims/2)
  mtx1 <- (mtx1 - .5) * 10
  mtx2 <- 1 - mtx1
  fx1 <- apply(mtx1, 1, cost, D = D)
  fx2 <- apply(mtx2, 1, cost, D = D)
  return((fx1 + fx2) / 2)
}
```

The second function creates a table with the info for the simulation and plots.

```
antitable <- function(D) {
  samps <- seq(1000, 10000, by = 1000) # says 100 times, this only gives 10
  means <- c() # make some empty things to store in later
  stds <- c()
  cvs <- c()

  for (samp in samps) { # loop to get all the samples
    cx <- replicate(100, anti(samp, D))
    means <- c(means, mean(cx))
    stds <- c(stds, sd(cx))
    cvs <- c(cvs, mean(cx)/sd(cx))
  }
  return (data.frame(samples = samps, means = means, std.dev = stds, coef.vars = cvs, D = D, method = "A
}
```

Again, I'll write these to csv's to reduce load time later on.

```
write.csv(antitable(1), file = "anti.csv")
write.csv(antitable(2), file = "anti2.csv")
```

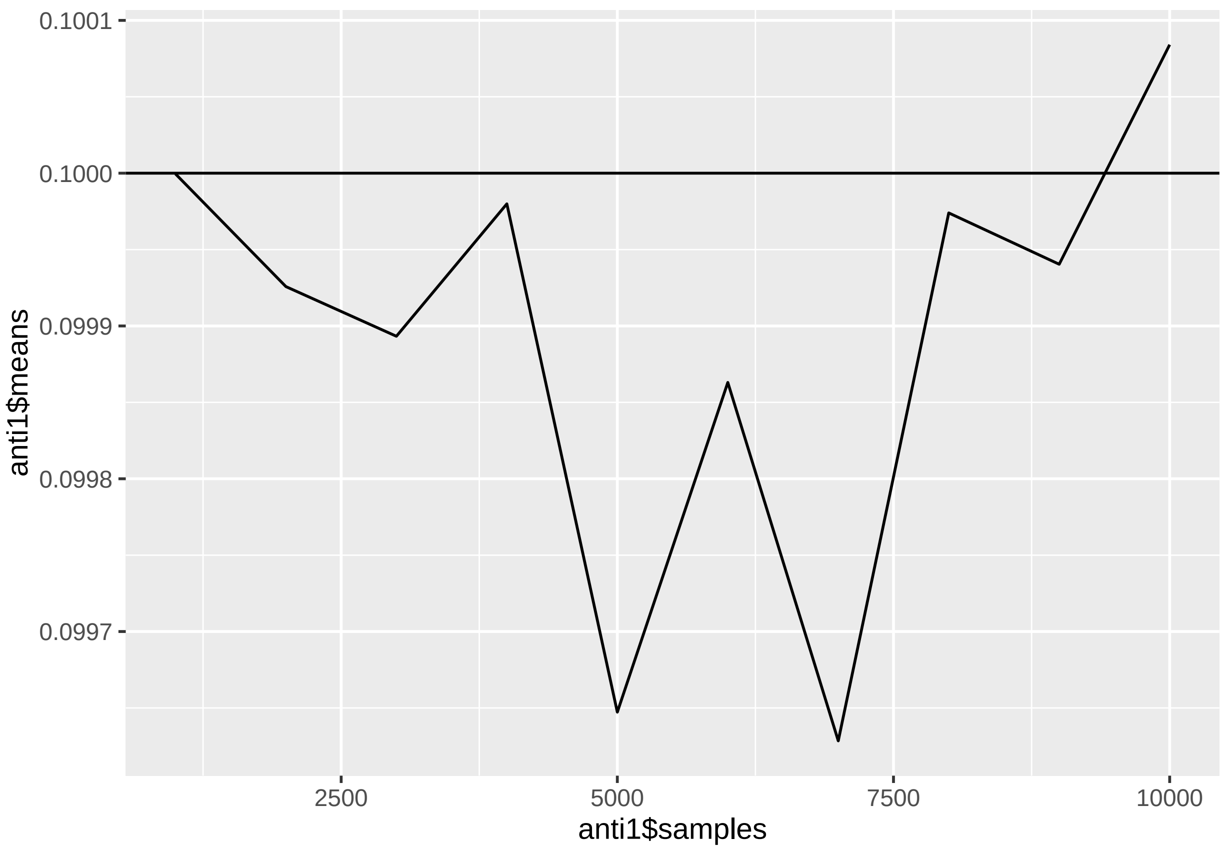
```
anti1 <- read.csv("anti.csv")
anti2 <- read.csv("anti2.csv")
anti1;anti2
```

```
##     X samples      means   std.dev coef.vars D     method
## 1   1    1000 0.09999955 0.1226618 0.8152462 1 Antithetic
## 2   2    2000 0.09992577 0.1227464 0.8140830 1 Antithetic
## 3   3    3000 0.09989326 0.1229162 0.8126940 1 Antithetic
## 4   4    4000 0.09997993 0.1228833 0.8136167 1 Antithetic
## 5   5    5000 0.09964725 0.1228202 0.8113261 1 Antithetic
## 6   6    6000 0.09986304 0.1227227 0.8137291 1 Antithetic
## 7   7    7000 0.09962844 0.1227891 0.8113787 1 Antithetic
## 8   8    8000 0.09997400 0.1228458 0.8138173 1 Antithetic
## 9   9    9000 0.09994039 0.1228720 0.8133703 1 Antithetic
## 10 10   10000 0.10008410 0.1228967 0.8143760 1 Antithetic
```

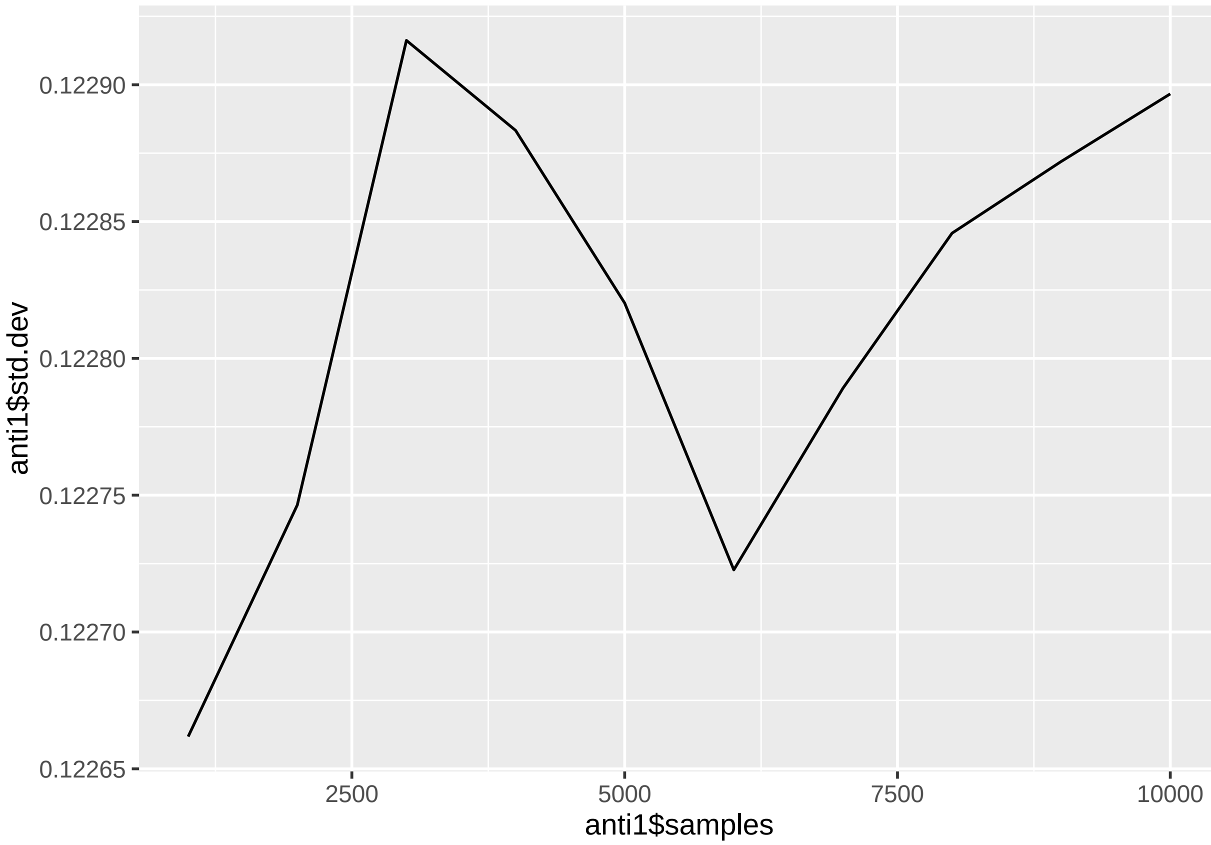
```
##     X samples      means    std.dev coef.vars D     method
## 1   1    1000 0.03984957 0.04901223 0.8130536 2 Antithetic
## 2   2    2000 0.03993102 0.04907693 0.8136414 2 Antithetic
## 3   3    3000 0.03975714 0.04902112 0.8110207 2 Antithetic
## 4   4    4000 0.03993977 0.04897787 0.8154656 2 Antithetic
## 5   5    5000 0.03996042 0.04907472 0.8142771 2 Antithetic
## 6   6    6000 0.03991874 0.04902924 0.8141824 2 Antithetic
## 7   7    7000 0.03993163 0.04902775 0.8144699 2 Antithetic
## 8   8    8000 0.03993414 0.04901663 0.8147058 2 Antithetic
## 9   9    9000 0.03988777 0.04903078 0.8135251 2 Antithetic
## 10 10   10000 0.03982451 0.04900028 0.8127406 2 Antithetic
```

And now for the graphs of the means and sds. The D = 2 graph for means has something wrong with it, as the estimation for the mean is far too high. Not sure why. The std's are higher than in the monte carlo method above.

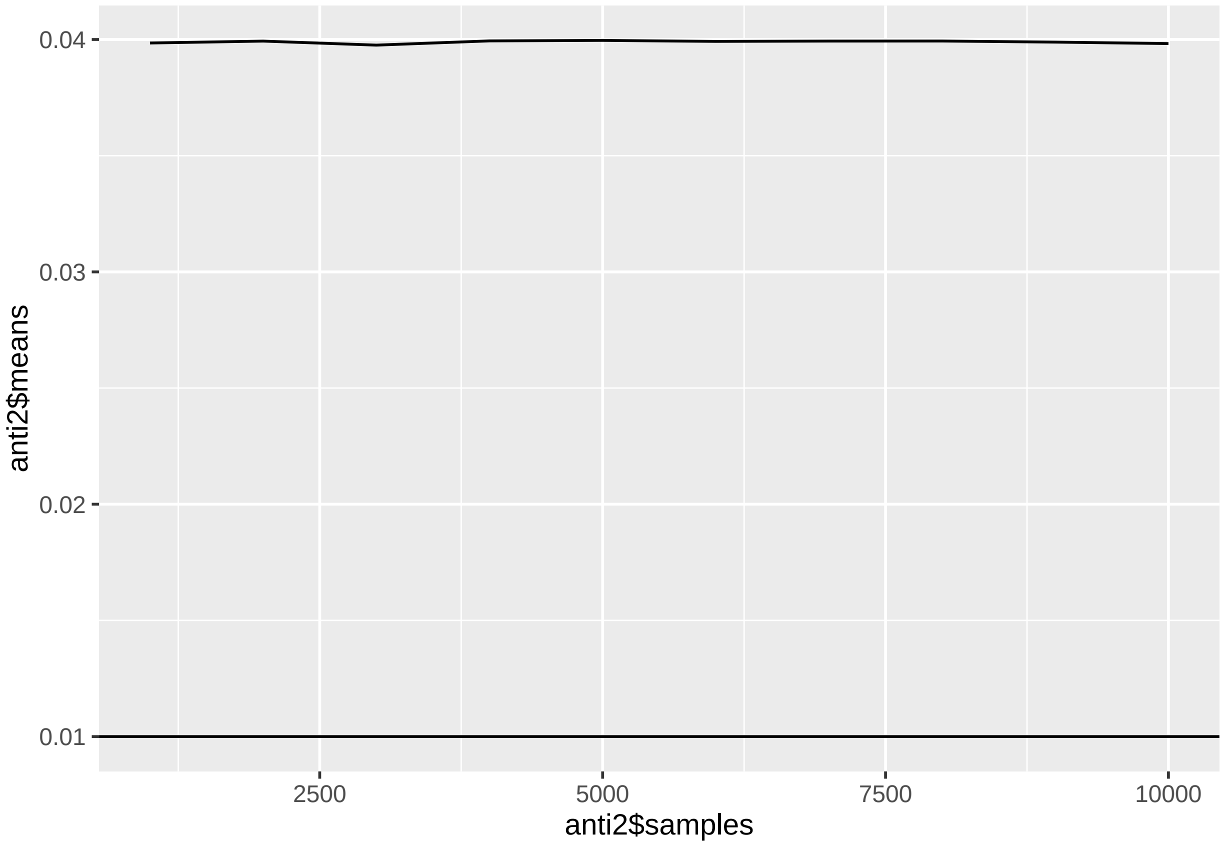
```r
qplot(anti1$samples,anti1$means, geom = "line")  + geom_hline(yintercept = (1/10))
```



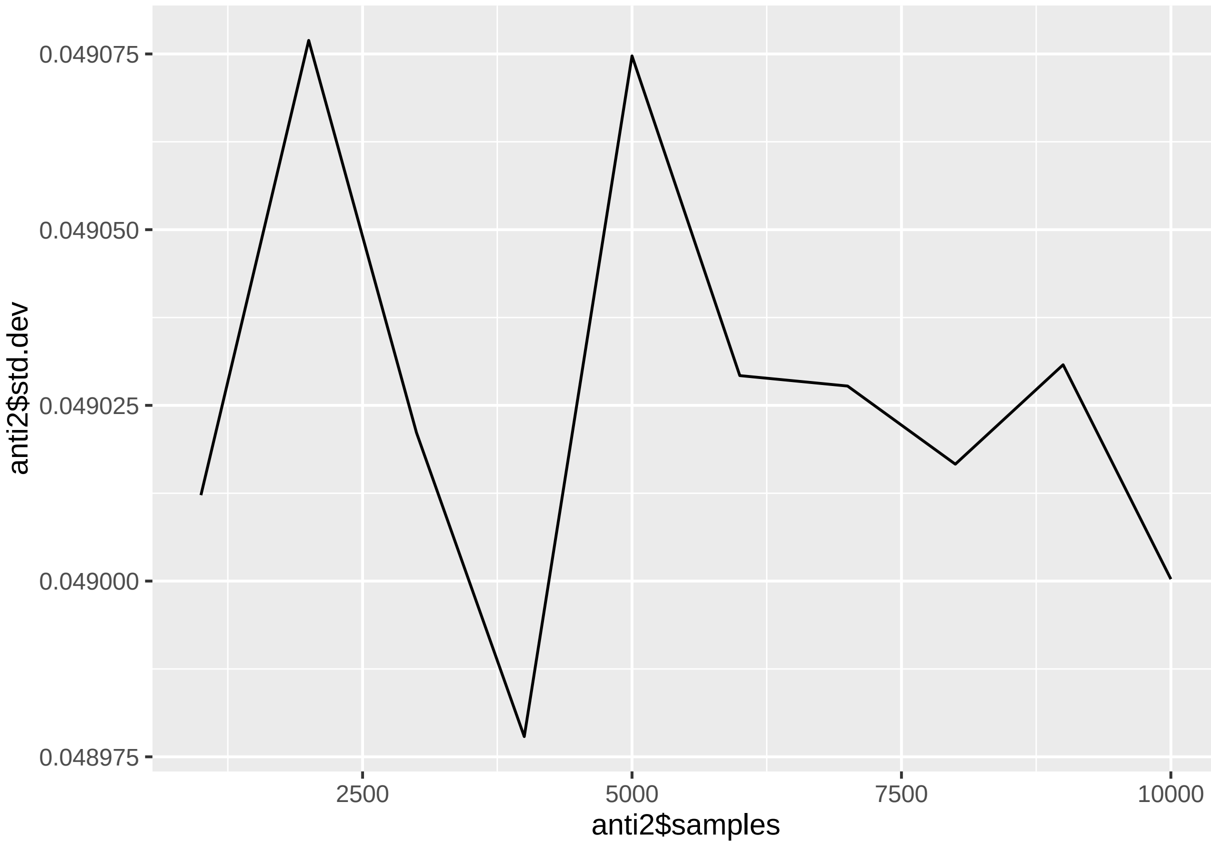

```r
qplot(anti1$samples,anti1$std.dev, geom = "line")
```




```
qplot(anti2$samples,anti2$means, geom = "line")  + geom_hline(yintercept = (1/10)^2)
```




```
qplot(anti2$samples,anti2$std.dev, geom = "line")
```



d. Latin Hypercubing

The functions work similarily again, but with a added k value.

```
latin <- function(n, D, k) {
  dims <- n * D / k
  mtx <- matrix(runif(dims), nrow = n/k)
  p <- replicate(D, sample(1:(n/k)))
  mtx <- (p + 1 - mtx) / (n/k)
  mtx <- (v-.5) * 10
  y <- mean(apply(v, 1, cost, D = D))
  return(mean(replicate(k, y)))
}

latintable <- function(D) {
  samps <- seq(1000, 10000, by = 1000) # says 100 times, this only gives 10
  means <- c() # make some empty things to store in later
  stds <- c()
  cvs <- c()

  for (samp in samps) { # loop to get all the samples
    cx <- replicate(100, anti(samp, D))
    means <- c(means, mean(cx))
    stds <- c(stds, sd(cx))
    cvs <- c(cvs, mean(cx)/sd(cx))
```

```r
  }
  return (data.frame(samples = samps, means = means, std.dev = stds, coef.vars = cvs, D = D, method = "I
}
```

Again, I'll write these to csv's to reduce load time later on.

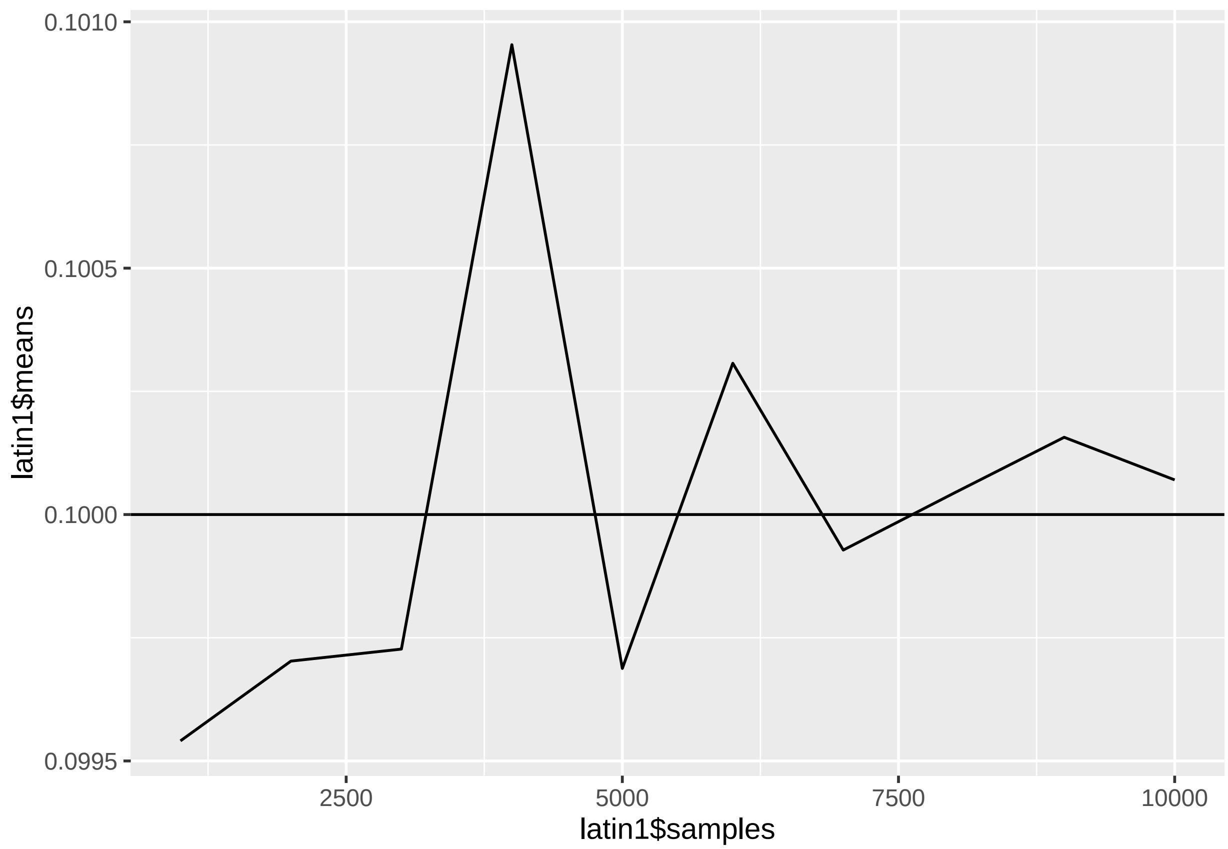
```r
write.csv(latintable(1), file = "latin.csv")
write.csv(latintable(2), file = "latin2.csv")
```

```r
latin1 <- read.csv("latin.csv")
latin2 <- read.csv("latin2.csv")
latin1;latin2
```

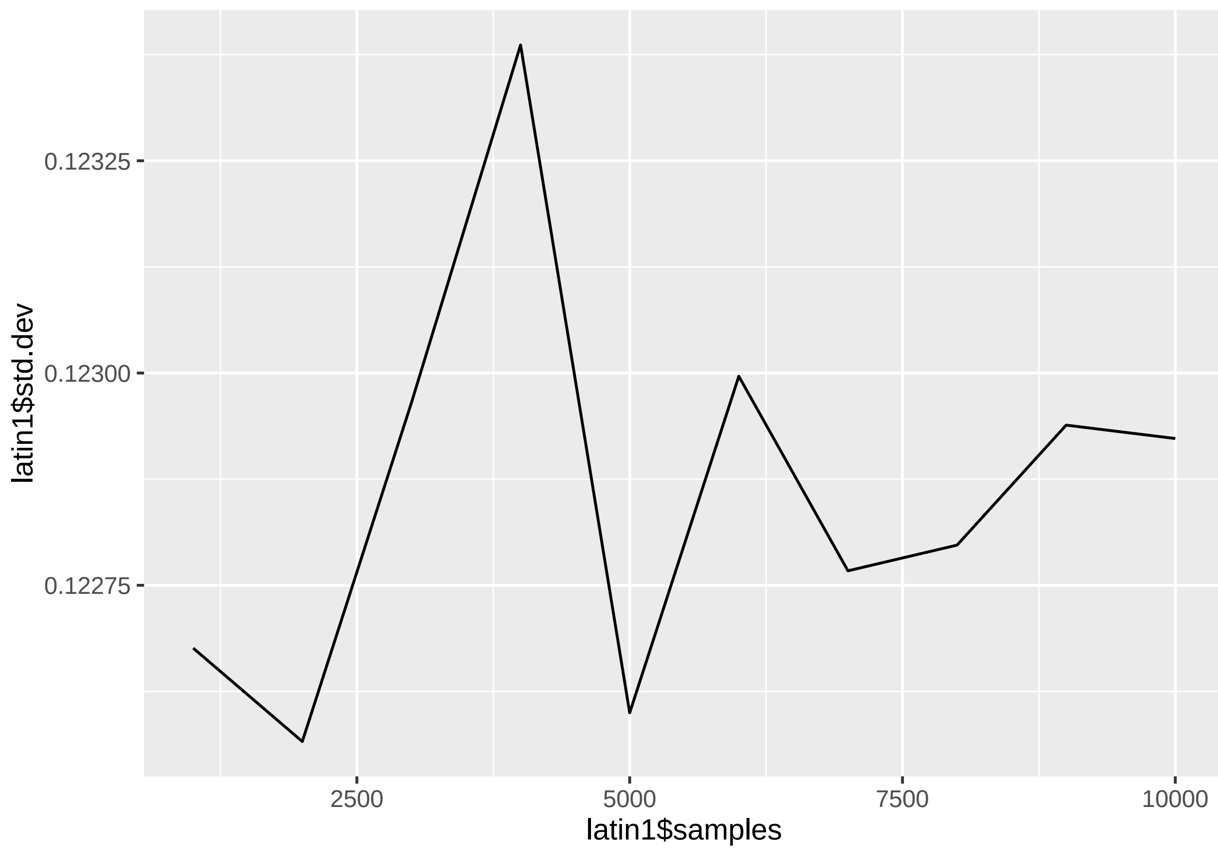
```
##     X samples      means   std.dev coef.vars D method
## 1   1    1000 0.09954073 0.1226760 0.8114117 1  Latin
## 2   2    2000 0.09970262 0.1225660 0.8134608 1  Latin
## 3   3    3000 0.09972695 0.1229649 0.8110194 1  Latin
## 4   4    4000 0.10095354 0.1233864 0.8181900 1  Latin
## 5   5    5000 0.09968765 0.1225998 0.8131144 1  Latin
## 6   6    6000 0.10030706 0.1229962 0.8155299 1  Latin
## 7   7    7000 0.09992794 0.1227670 0.8139639 1  Latin
## 8   8    8000 0.10004326 0.1227973 0.8147024 1  Latin
## 9   9    9000 0.10015682 0.1229386 0.8146895 1  Latin
## 10 10   10000 0.10007032 0.1229229 0.8140904 1  Latin

##     X samples      means    std.dev coef.vars D method
## 1   1    1000 0.03964922 0.04886153 0.8114610 2  Latin
## 2   2    2000 0.04006758 0.04911557 0.8157816 2  Latin
## 3   3    3000 0.03984855 0.04900489 0.8131546 2  Latin
## 4   4    4000 0.03993616 0.04904312 0.8143070 2  Latin
## 5   5    5000 0.03983604 0.04895641 0.8137044 2  Latin
## 6   6    6000 0.04005779 0.04906298 0.8164565 2  Latin
## 7   7    7000 0.03985544 0.04897184 0.8138440 2  Latin
## 8   8    8000 0.03995447 0.04902154 0.8150391 2  Latin
## 9   9    9000 0.03978116 0.04896475 0.8124449 2  Latin
## 10 10   10000 0.03979956 0.04895703 0.8129488 2  Latin
```

And now for graphs. From looking at the graphs, the d = 2 mean suffers from the same problem as the antithetic one did. Not sure where it's coming from. But again the std's are higher than the monte carlo method.

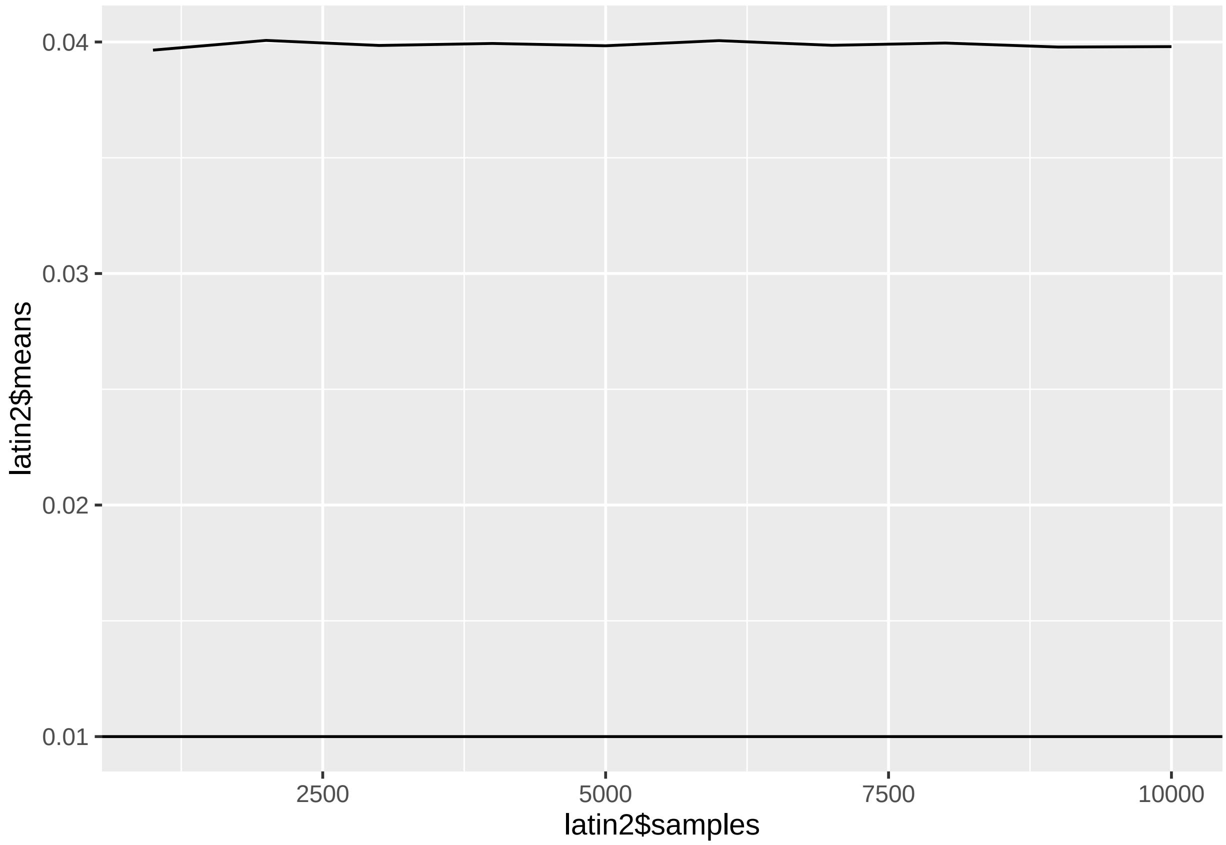
```
qplot(latin1$samples,latin1$means, geom = "line")  + geom_hline(yintercept = (1/10))
```



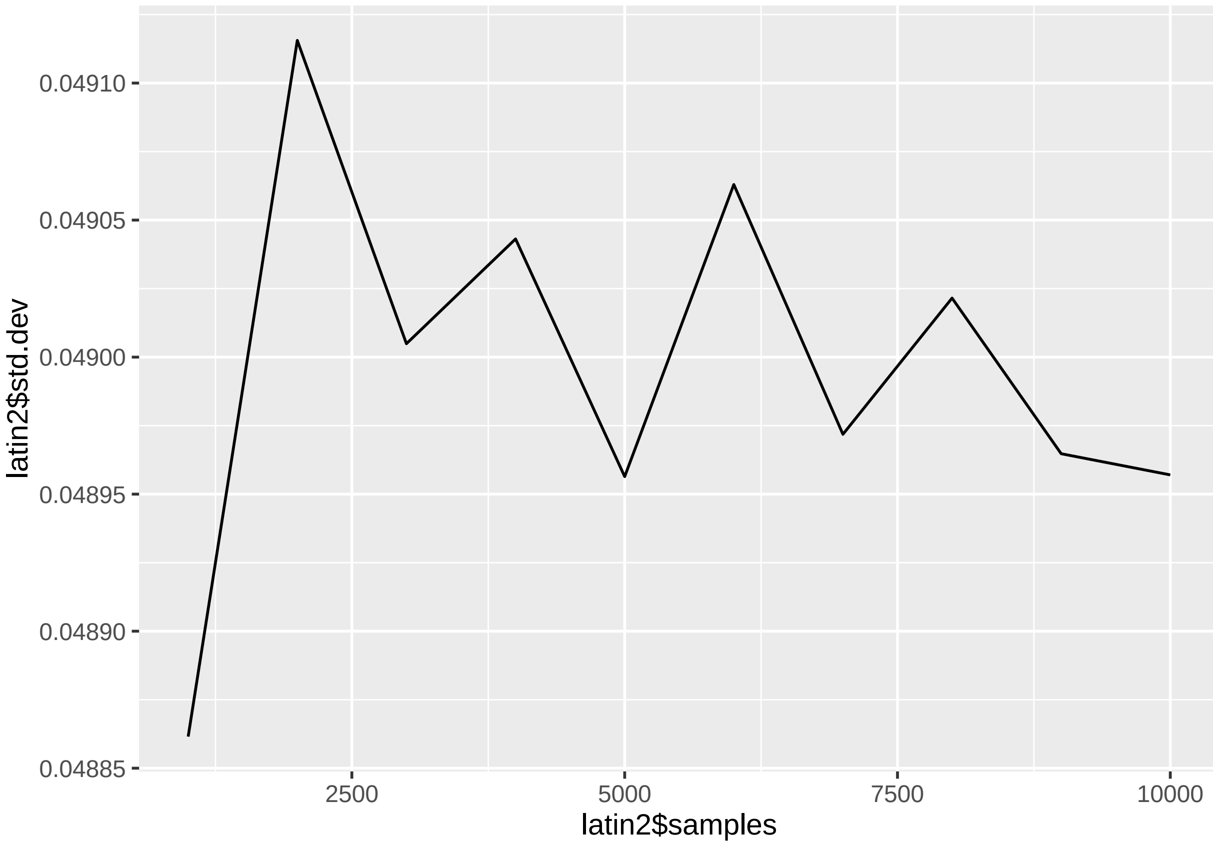

```
qplot(latin1$samples,latin1$std.dev, geom = "line")
```




```
qplot(latin2$samples,latin2$means, geom = "line")  + geom_hline(yintercept = (1/10)^2)
```




```
qplot(latin2$samples,latin2$std.dev, geom = "line")
```



e. Importance Sampling

Same problem as Sobol, I feel like I understand the method, but just can't get it into R.

f. Summary

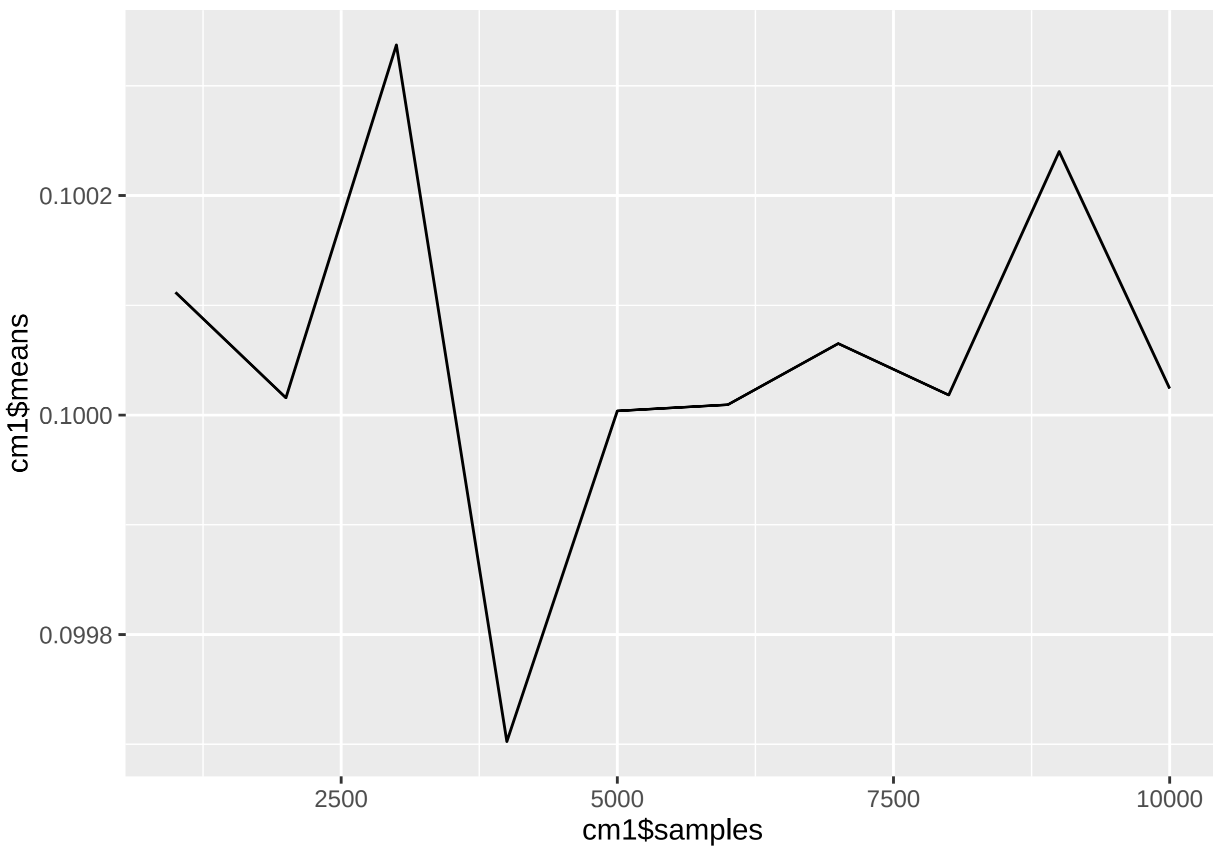
I'll include only the methods that I managed to do. For the 2 other methods I managed the sd was higher than in the original monte carlo. This seems blatantly wrong to me, but I'm unsure. I completely struggled with the coding for this week's assignment. I feel like I understand the ideas, but just couldn't translate it into R very well.

```
table <- rbind(cm1,cm2,anti1,anti2,latin1,latin2)
table
```

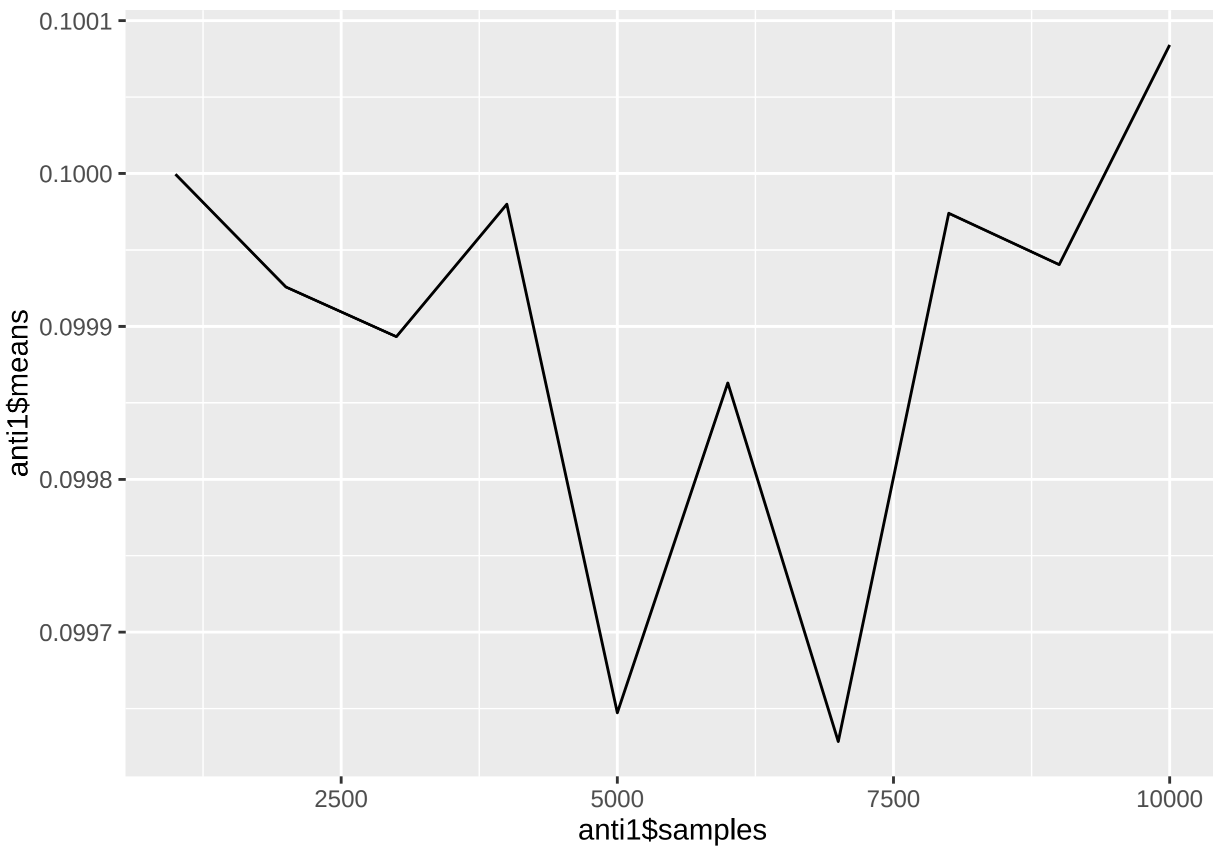
```
##      X samples       means      std.dev  coef.vars D       method
## 1    1    1000 0.100111856 0.0045242615 22.1277783 1 Crude Monte
## 2    2    2000 0.100015670 0.0032934576 30.3679844 1 Crude Monte
## 3    3    3000 0.100337275 0.0024491120 40.9688381 1 Crude Monte
## 4    4    4000 0.099702458 0.0022624066 44.0692049 1 Crude Monte
## 5    5    5000 0.100003762 0.0020883497 47.8865016 1 Crude Monte
## 6    6    6000 0.100009405 0.0017994376 55.5781456 1 Crude Monte
## 7    7    7000 0.100065144 0.0015651769 63.9321635 1 Crude Monte
## 8    8    8000 0.100018231 0.0016167332 61.8643994 1 Crude Monte
## 9    9    9000 0.100240112 0.0015348128 65.3109687 1 Crude Monte
```

```
## 10 10   10000 0.100024117 0.0013771291 72.6323434 1 Crude Monte
## 11  1    1000 0.010037234 0.0009122067 11.0032447 2 Crude Monte
## 12  2    2000 0.010018370 0.0005626787 17.8047804 2 Crude Monte
## 13  3    3000 0.009996352 0.0004954536 20.1761608 2 Crude Monte
## 14  4    4000 0.010014905 0.0003650038 27.4378063 2 Crude Monte
## 15  5    5000 0.009994480 0.0004130666 24.1958045 2 Crude Monte
## 16  6    6000 0.010018711 0.0002969464 33.7391275 2 Crude Monte
## 17  7    7000 0.009974660 0.0003286802 30.3476191 2 Crude Monte
## 18  8    8000 0.009978739 0.0002646449 37.7061514 2 Crude Monte
## 19  9    9000 0.010024854 0.0002908782 34.4641010 2 Crude Monte
## 20 10   10000 0.009989636 0.0002429210 41.1229768 2 Crude Monte
## 21  1    1000 0.099999545 0.1226617757  0.8152462 1  Antithetic
## 22  2    2000 0.099925771 0.1227464100  0.8140830 1  Antithetic
## 23  3    3000 0.099893261 0.1229162040  0.8126940 1  Antithetic
## 24  4    4000 0.099979929 0.1228833270  0.8136167 1  Antithetic
## 25  5    5000 0.099647246 0.1228202140  0.8113261 1  Antithetic
## 26  6    6000 0.099863036 0.1227227060  0.8137291 1  Antithetic
## 27  7    7000 0.099628445 0.1227890882  0.8113787 1  Antithetic
## 28  8    8000 0.099973999 0.1228457563  0.8138173 1  Antithetic
## 29  9    9000 0.099940394 0.1228719526  0.8133703 1  Antithetic
## 30 10   10000 0.100084102 0.1228966692  0.8143760 1  Antithetic
## 31  1    1000 0.039849572 0.0490122305  0.8130536 2  Antithetic
## 32  2    2000 0.039931017 0.0490769252  0.8136414 2  Antithetic
## 33  3    3000 0.039757143 0.0490211184  0.8110207 2  Antithetic
## 34  4    4000 0.039939772 0.0489778744  0.8154656 2  Antithetic
## 35  5    5000 0.039960422 0.0490747216  0.8142771 2  Antithetic
## 36  6    6000 0.039918743 0.0490292360  0.8141824 2  Antithetic
## 37  7    7000 0.039931626 0.0490277508  0.8144699 2  Antithetic
## 38  8    8000 0.039934137 0.0490166330  0.8147058 2  Antithetic
## 39  9    9000 0.039887769 0.0490307774  0.8135251 2  Antithetic
## 40 10   10000 0.039824513 0.0490002754  0.8127406 2  Antithetic
## 41  1    1000 0.099540733 0.1226759899  0.8114117 1       Latin
## 42  2    2000 0.099702616 0.1225659695  0.8134608 1       Latin
## 43  3    3000 0.099726948 0.1229649424  0.8110194 1       Latin
## 44  4    4000 0.100953542 0.1233864228  0.8181900 1       Latin
## 45  5    5000 0.099687645 0.1225997734  0.8131144 1       Latin
## 46  6    6000 0.100307062 0.1229961734  0.8155299 1       Latin
## 47  7    7000 0.099927941 0.1227670483  0.8139639 1       Latin
## 48  8    8000 0.100043264 0.1227973160  0.8147024 1       Latin
## 49  9    9000 0.100156816 0.1229386436  0.8146895 1       Latin
## 50 10   10000 0.100070322 0.1229228676  0.8140904 1       Latin
## 51  1    1000 0.039649225 0.0488615308  0.8114610 2       Latin
## 52  2    2000 0.040067575 0.0491155663  0.8157816 2       Latin
## 53  3    3000 0.039848555 0.0490048912  0.8131546 2       Latin
## 54  4    4000 0.039936161 0.0490431248  0.8143070 2       Latin
## 55  5    5000 0.039836044 0.0489564060  0.8137044 2       Latin
## 56  6    6000 0.040057791 0.0490629817  0.8164565 2       Latin
## 57  7    7000 0.039855436 0.0489718392  0.8138440 2       Latin
## 58  8    8000 0.039954475 0.0490215449  0.8150391 2       Latin
## 59  9    9000 0.039781163 0.0489647546  0.8124449 2       Latin
## 60 10   10000 0.039799562 0.0489570321  0.8129488 2       Latin
```

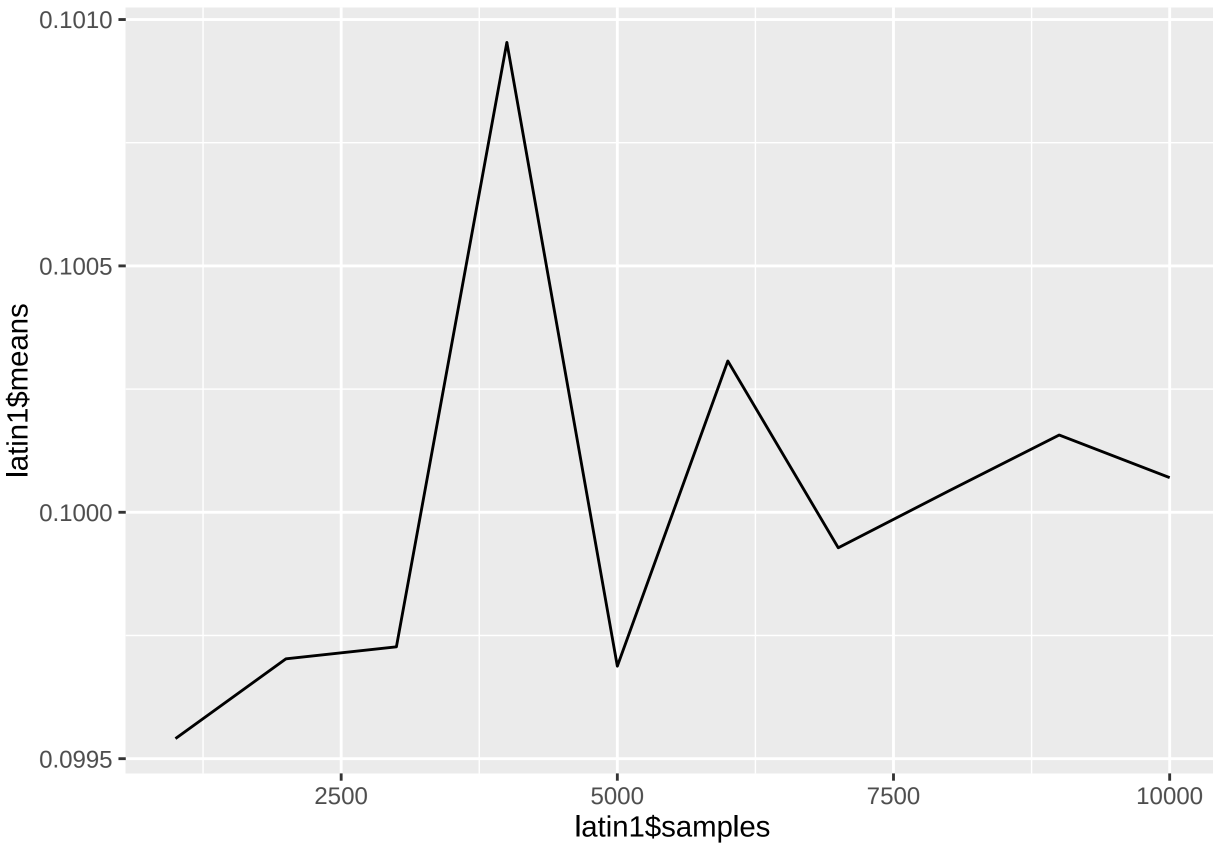
```r
qplot(cm1$samples,cm1$means, geom = "line")
```



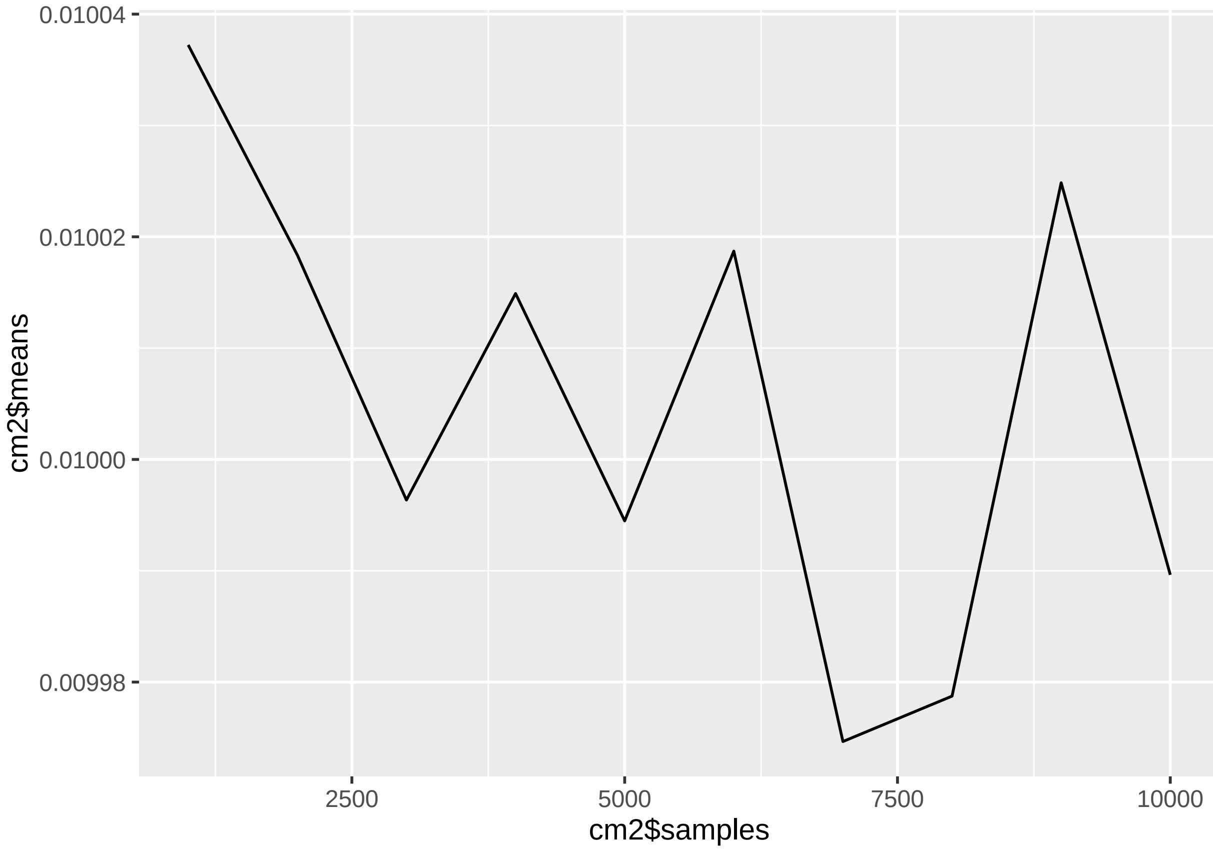

```r
qplot(anti1$samples,anti1$means, geom = "line")
```



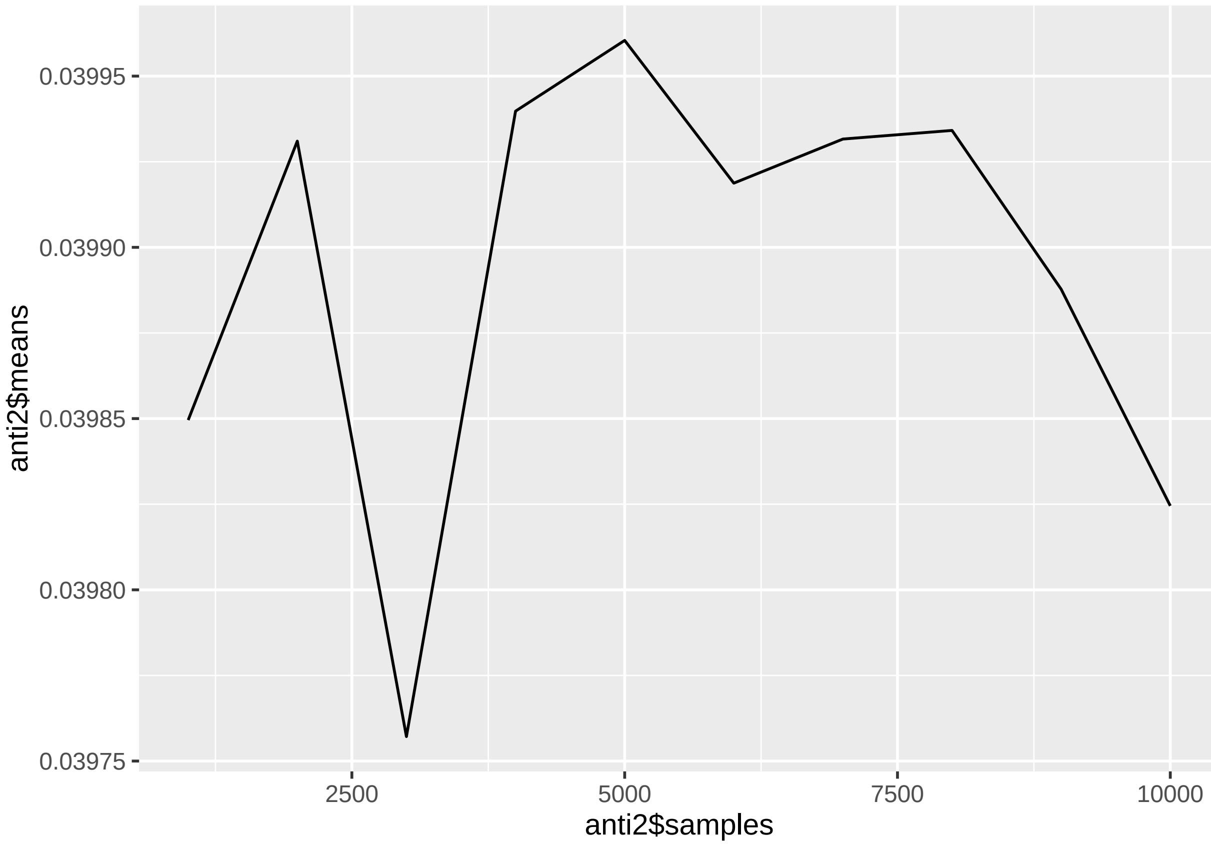

```
qplot(latin1$samples,latin1$means, geom = "line")
```



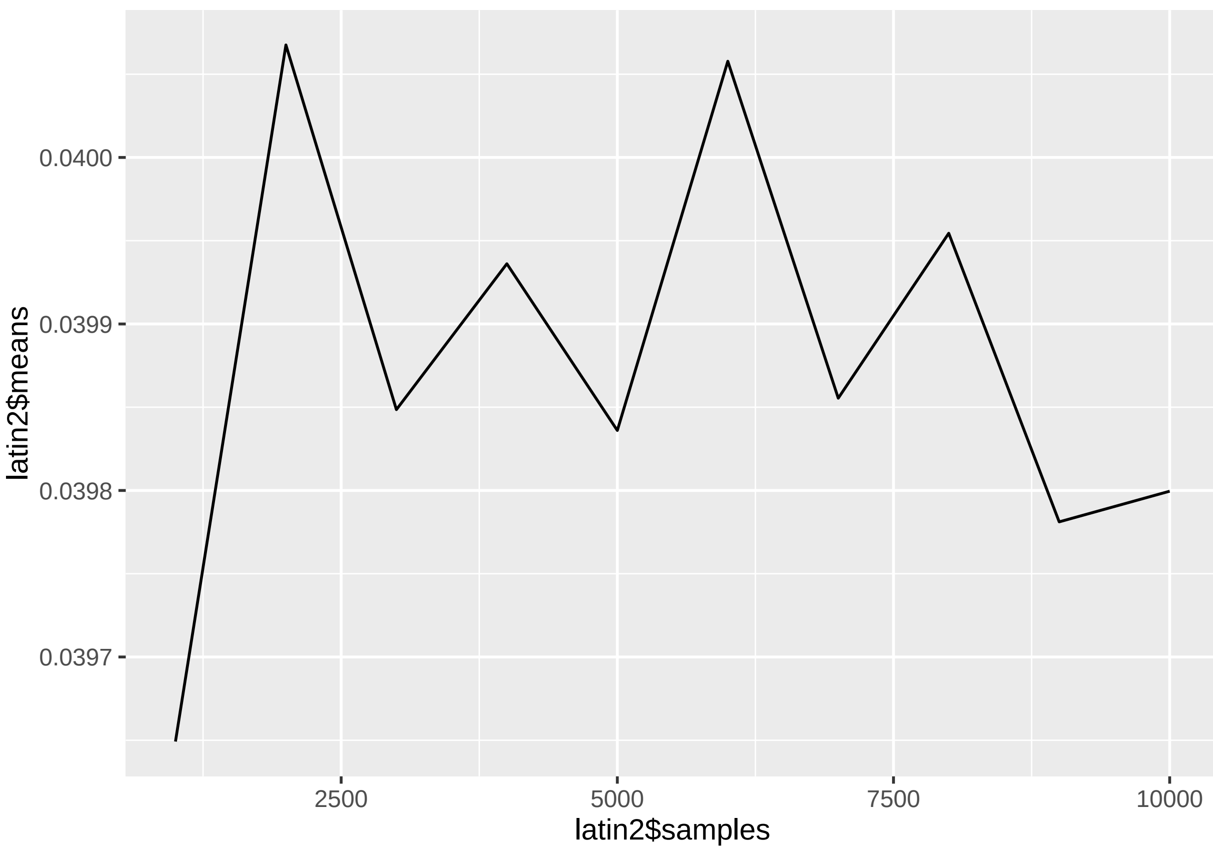

```
qplot(cm2$samples,cm2$means, geom = "line")
```




```
qplot(anti2$samples,anti2$means, geom = "line")
```



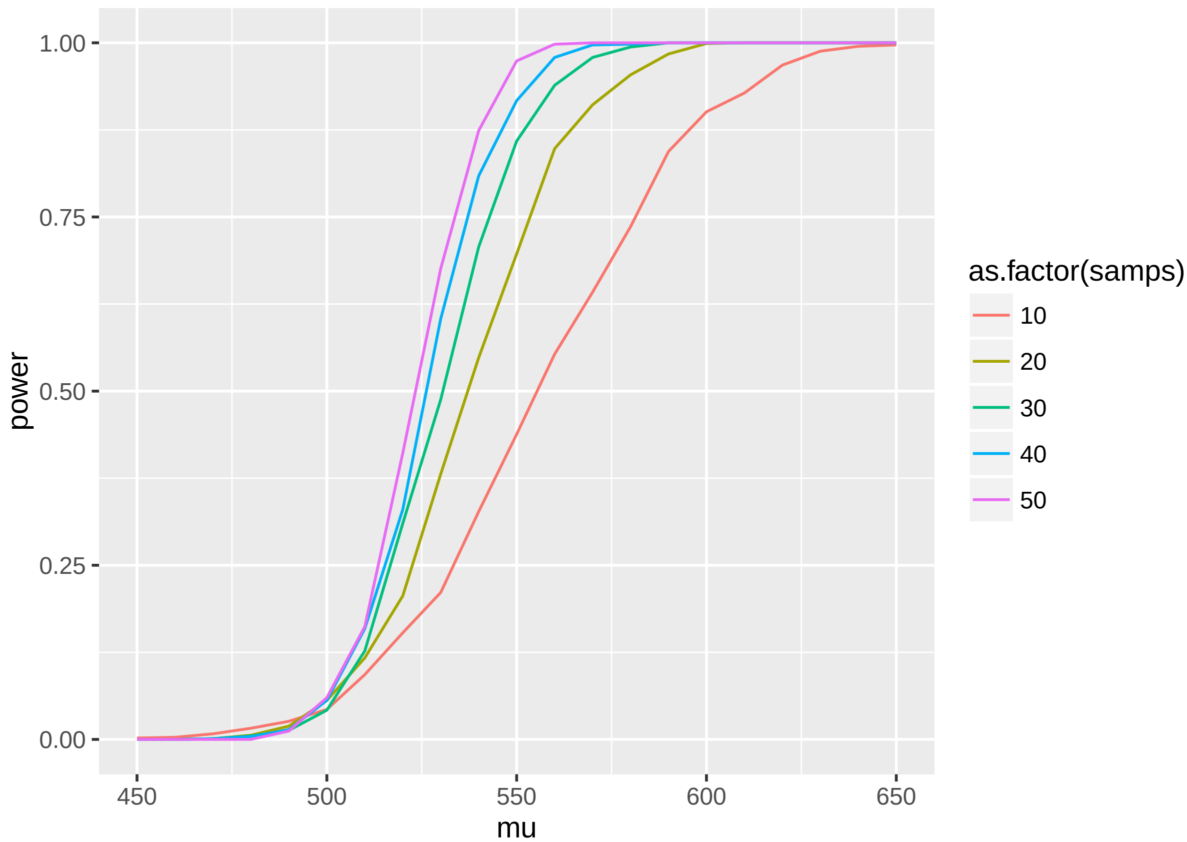

```
qplot(latin2$samples,latin2$means, geom = "line")
```



6.3

```
samps <- c(10,20,30,40,50)
### this section taken from the book with some edits
m <- 1000
mu0 <- 500
sigma <- 100
mu <- c(seq(450,650,10))
M <- length(mu)
power <- c()
for(samp in samps) {
  for(i in 1:M) {
    mu1 <- mu[i]
    pvalues <- replicate(m,expr={
      x <- rnorm(samp, mean=mu1, sd=sigma)
      ttest <- t.test(x, alternative="greater", mu=mu0)
      ttest$p.value
    })
    power <- c(power,mean(pvalues<=0.05))
  }
}
powers <- expand.grid(mu, samps)
powers$power <- power
```

```r
colnames(powers) <- c("mu", "samps", "power")
ggplot(powers, aes(x = mu, y = power, color = as.factor(samps))) + geom_line()
```



From the graph we can see that all the lines are fairly similar until a mu value of 500, and all the lines plateau at power = 1. It makes sense for the higher samples to have a steeper slope because of lower error.

6.4

With unknown parameters we can assume normallity and se of $\frac{\sigma}{\sqrt{n}}$. With a random sample...

```r
rand <- rlnorm(100)
mean <- mean(rand)
sd <- sd(rand)
se <- sd/sqrt(length(rand))
ci <- c(mean - se, mean + se);ci
```

```
## [1] 1.374632 1.697125
```

7.1

```r
library(bootstrap)
x <- cor(law$LSAT, law$GPA)
```

```r
rows <- nrow(law)
jack <- numeric(rows)

for(row in 1:rows) {
  jack[row] <- cor(law$LSAT[-row],law$GPA[-row])
}

se <- sd(jack)
bias <- (rows - 1) * (mean(jack) - x)
se;bias
```

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

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

7.4

```r
library(boot)
x <- length(aircondit$hours) / sum(aircondit$hours)
n <- nrow(aircondit)
y <- numeric(50)

for(i in 1:50) {
  samp <- sample(aircondit$hours, n, replace = TRUE)
  y[i] <- length(samp) / sum(samp)
}

mean(y);x
```

```
## [1] 0.01074782
```

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