Stochastic Simulation HW1

Zihao Wang

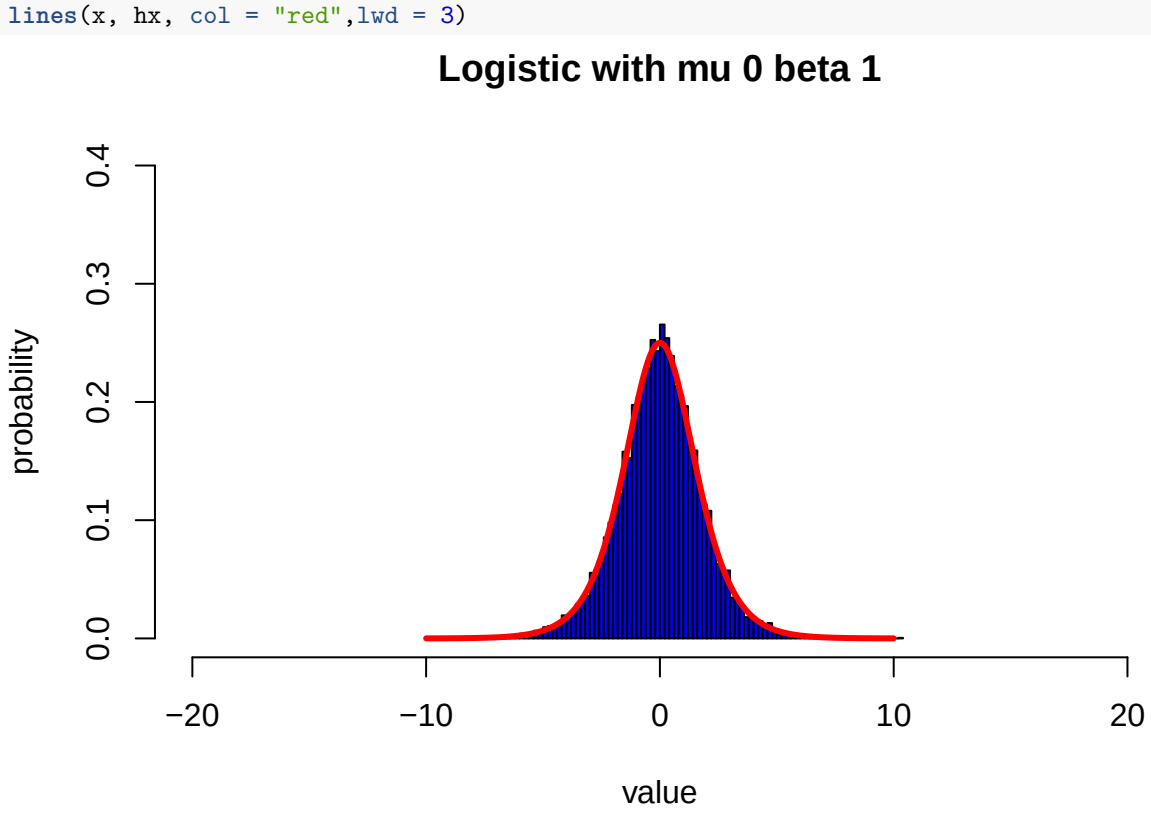
4/13/2019

Note: part of EX 1.5, EX 1.7, EX 2.3 in separate writeup # 1.1

Logistic distribution

It is easy to have $F^{-1}(U) = -\beta log(\frac{1}{U} - 1) + \mu$

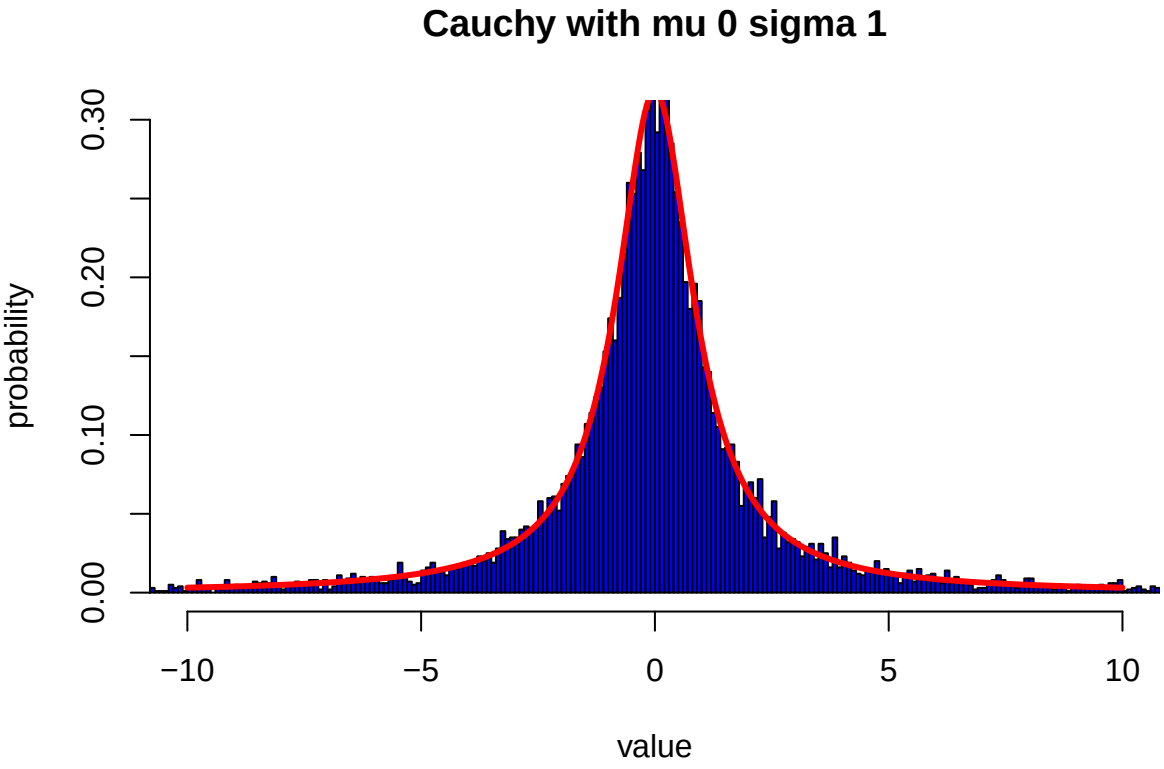
```
set.seed(12345)
N = 10000
## Cauchy
mu = 0
beta = 1
x = seq(-10, 10, length.out = 1000)
hx = dlogis(x, location = mu, scale = beta)

U = runif(N,0,1)
y = -beta*log((1/U) -1) + mu

#y = rcauchy(N, location = mu, scale = beta)
hist(y, freq = F, xlim = c(-20,20),ylim = c(0,0.4), breaks = 100,
     col = "blue", xlab = "value", ylab = "probability", main = paste0("Logistic with mu ", mu, " beta ", 
lines(x, hx, col = "red",lwd = 3)
```



Cauchy distribution

It is easy to see $F^{-1}(U) = \sigma tan(\pi(y - \frac{1}{2})) + \mu$

```
N = 10000
## Cauchy
mu = 0
s = 1
x = seq(-10, 10, length.out = 1000)
hx = dcauchy(x, location = mu, scale = s)

U = runif(N,0,1)
y = s* tan(pi*(y-0.5)) + mu

hist(y, freq = F, xlim = c(-10,10),ylim = c(0,0.3), breaks = 1000000,
     col = "blue", xlab = "value", ylab = "probability", main = paste0("Cauchy with mu ", mu, " sigma ", s
lines(x, hx, col = "red",lwd = 3)
```




```
rm(list = ls())
```

1.2

Proof:
First, it is easy to see that $F^{-1}(U) \leq t$ iff $F(t) \geq U$.
(=>) If not, then $F(t) < U$, contradictory with the definition of $F^{-}(U)$.
(<=) This is obvious by definition of of $F^{-}(U)$.
Then, we have $P(F^{-}(U) \leq t) = P(F(t) \geq U) = F(t)$. Thus $F^{-}(U)$ distributed like X.

1.3

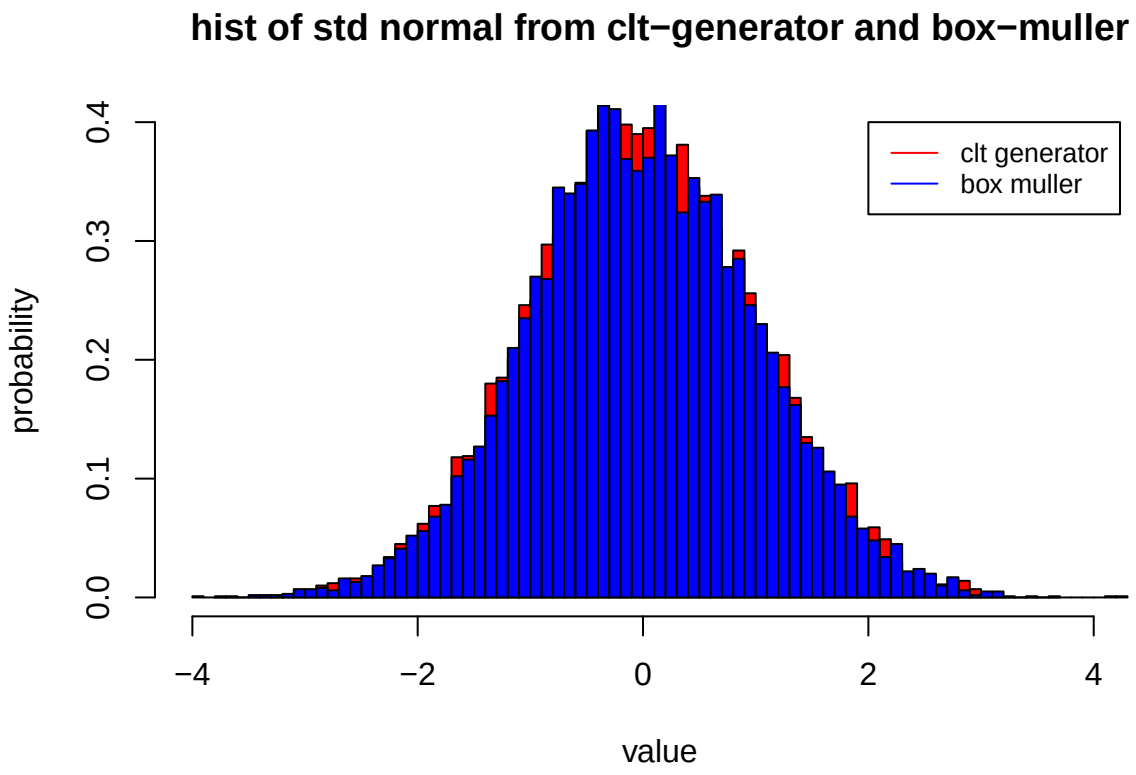
a)

$E(X) = \sum_{i=1}^{12} E(U_i) = 0$;

$Var(X) = \sum_{i=1}^{12} V(U_i) = 12 \times 1/12 = 1$ (Note: the first $=$ is due to the independence of U_i; variance of uniform distribution found in wikipedia)

b)

```
set.seed(1)
N = 10000
## clt generator
clt <- function(){
  return(sum(runif(12,-0.5,0.5)))
}
x_clt = replicate(N, clt())

## Box-muller
BoxMuller <- function(){
  u1 = runif(1,0,1)
  u2 = runif(1,0,1)
  return(c(sqrt(-2*log(u1))*cos(2*pi*u2),sqrt(-2*log(u1))* sin(2*pi*u2)))
}

x_box = replicate(N/2, BoxMuller())
n_breaks = 100
hist(x_clt, breaks = n_breaks, freq = F, col = "red",
     xlab = "value",ylab = "probability", main = "hist of std normal from clt-generator and box-muller" )
hist(x_box, add = T,breaks = n_breaks,freq = F, col = "blue")
legend(2, 0.4,legend=c("clt generator", "box muller"),
       col=c("red", "blue"), lty=1:1, cex=0.8)
```



look at their behavior at the tail

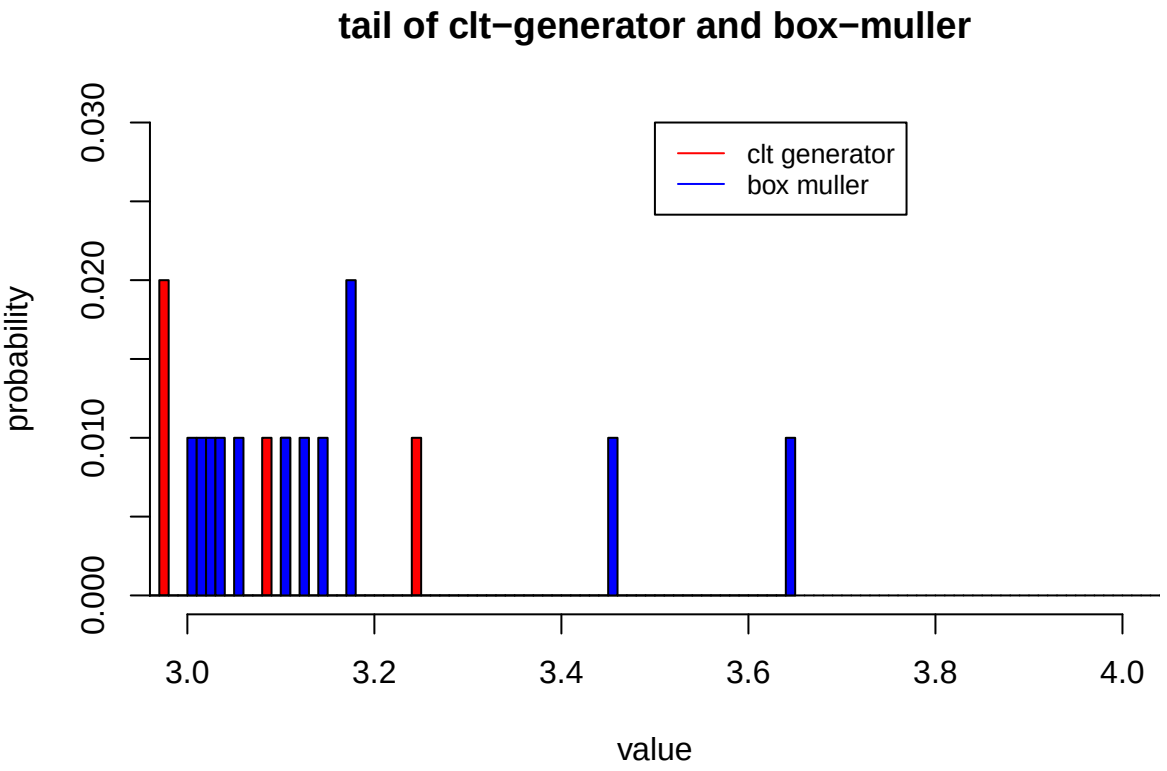
```
print(paste0("# > 3 in clt-generator: ", length(x_clt[x_clt>3])))
```

```
## [1] "# > 3 in clt-generator: 5"
```

```
print(paste0("# > 3 in box-muller   : ", length(x_box[x_box>3])))
```

```
## [1] "# > 3 in box-muller   : 12"
```

```
n_breaks = 1000
hist(x_clt, breaks = n_breaks, freq = F,xlim=c(3,4),ylim = c(0,0.03), col = "red",
     xlab = "value",ylab = "probability", main = "tail of clt-generator and box-muller" )
hist(x_box, add = T,breaks = n_breaks,freq = F,,xlim=c(3,4), col = "blue")
legend(3.5, 0.03,legend=c("clt generator", "box muller"),
       col=c("red", "blue"), lty=1:1, cex=0.8)
```



Comment for (b):

- The two generated distribution are quite close to each other in general.
- Box-Muller has fatter tail.

c)

```
x_norm = rnorm(N, 0, 1)
print(paste0("P(X > 3) = ", pnorm(3,0,1, lower.tail = F), " from pnorm"))
```

```
## [1] "P(X > 3) = 0.00134989803163009 from pnorm"
```

```
print(paste0("P(X > 3) = ", length(x_norm[x_norm > 3])/N, " for X ~ N(0,1)"))
```

```
## [1] "P(X > 3) = 0.0014 for X ~ N(0,1)"
```

```
print(paste0("P(X > 3) = ", length(x_norm[x_norm > 3])/N, " for X ~ box-muller"))
```

```
## [1] "P(X > 3) = 0.0014 for X ~ box-muller"
```

```
print(paste0("P(X > 3) = ", length(x_clt[x_clt > 3])/N, " for X ~ clt-generator"))
```

```
## [1] "P(X > 3) = 5e-04 for X ~ clt-generator"
```

```
rm(list = ls())
```

Comment for (c)

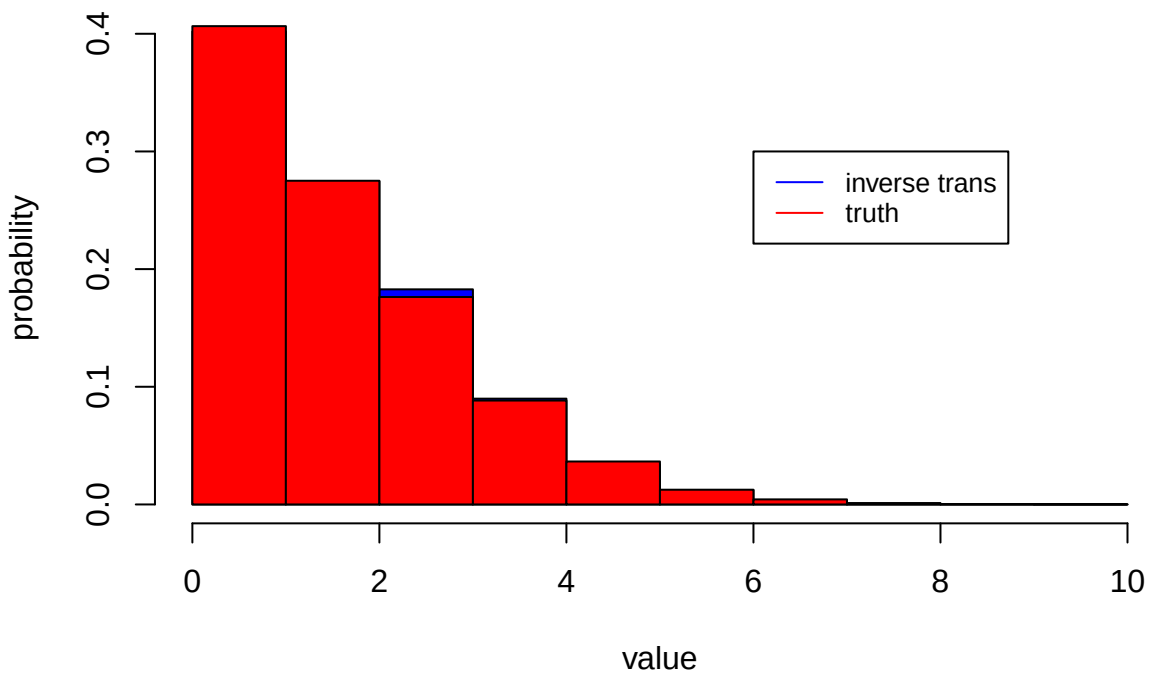
Distribution from Box-Muller fits real normal in the tail pretty well, while distribution from clt generator has thinner (too thin) tail.

1.4 Poisson generator

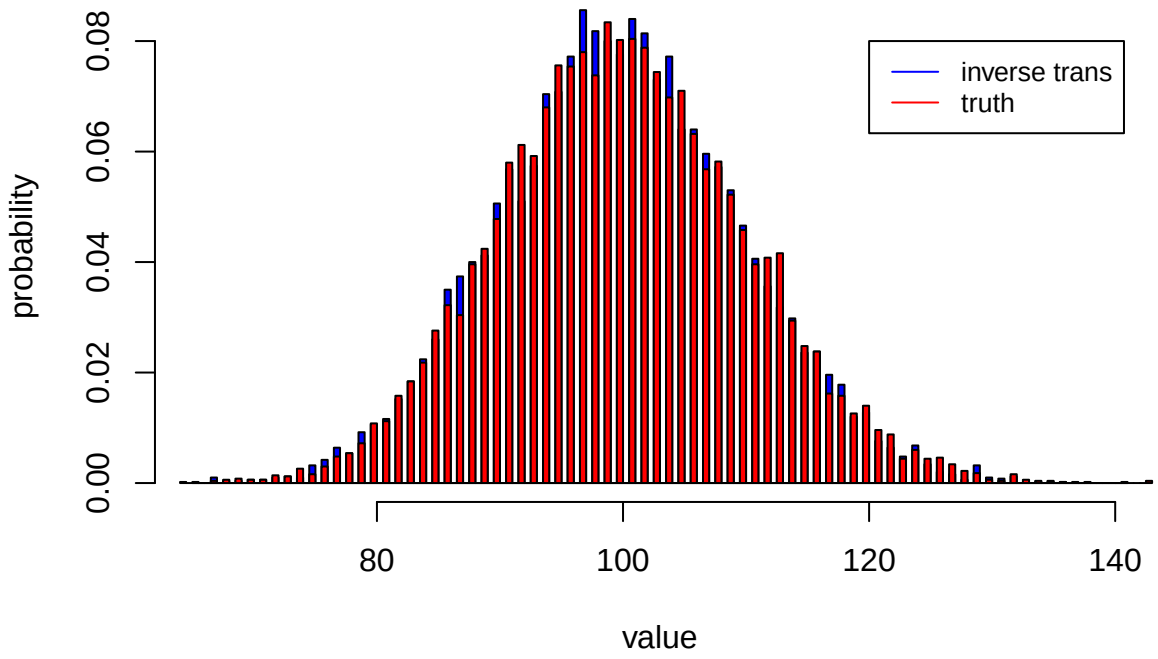
from inverse transformation

```
rm(list = ls())
set.seed(12345)
## first, compute cdf of poisson up to M
pois_util <- function(k, lam){return(exp(-lam)*(lam^k)/factorial(k))}

Finv <- function(u,xs, cum_sum){
  return(min(xs[cum_sum > u]))
}

pois_inv_util <- function(xs, cum_sum){
  return(Finv(runif(1,0,1), xs, cum_sum))
}

pois_inv <- function(N, lam){
  biggest = min(10*lam, 150) ## note factorial(150) can be computed, but factorial(180) is +Inf in R
  xs = seq(0, biggest, 1)
  pois_cumsum = cumsum(pois_util(xs, lam))
  return(replicate(N, pois_inv_util(xs, pois_cumsum)))
}
```

```
N  = 10000
lam = 2
x_poisinv = pois_inv(N, lam)
a = hist(x_poisinv, breaks = max(x_poisinv)+1, col = "blue", freq = F,
     xlab = "value",ylab = "probability", main = paste0("hist of poisson from inverse transform VS truth; l
hist(rpois(N, lam), add = T, breaks = a$breaks, col = "red",freq = F)
legend(6, 0.3,legend=c("inverse trans", "truth"),
       col=c("blue", "red"), lty=1:1, cex=0.8)
```



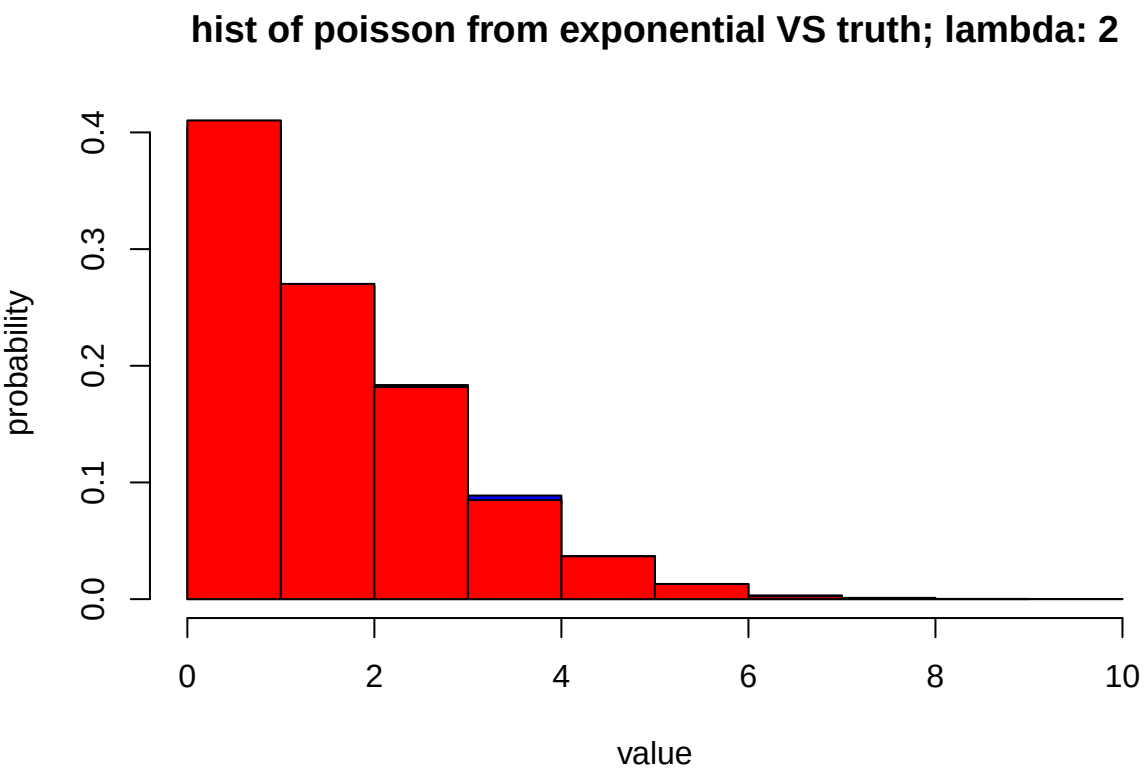

```
lam = 100
x_poisinv = pois_inv(N, lam)
hist(x_poisinv, breaks = max(x_poisinv), col = "blue", freq = F,
     xlab = "value",ylab = "probability", main = paste0("hist of poisson from inverse transform VS truth; l
hist(rpois(N, lam), add = T, breaks = max(x_poisinv), col = "red",freq = F)
legend(120, 0.08,legend=c("inverse trans", "truth"),
       col=c("blue", "red"), lty=1:1, cex=0.8)
```



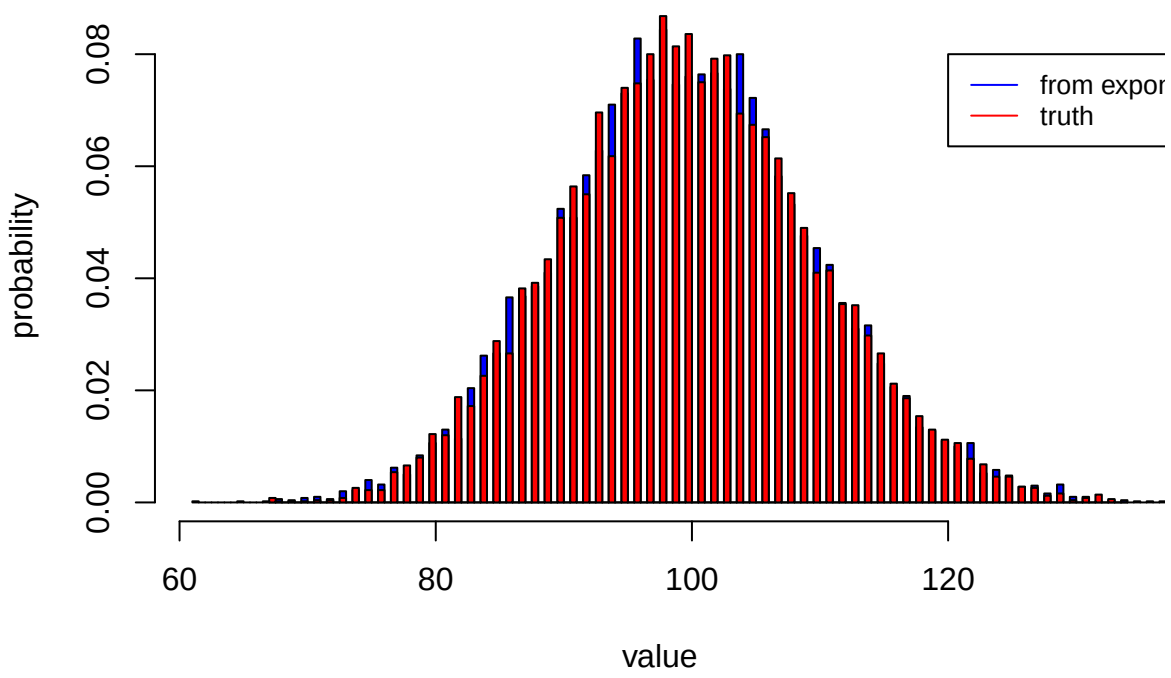
from exponential

```
pois_exp_util <- function(lam){
  m = max(2*lam, 100) ## well, only make sure that they work for our two cases
  xs = rexp(m, lam)
  xs_sum = cumsum(xs)
  return(length(xs_sum[xs_sum < 1]))
}
pois_exp <- function(N, lam){
  return(replicate(N, pois_exp_util(lam)))
}
```

```
lam = 2
x_poisexp = pois_exp(N, lam)

hist(x_poisexp, breaks = max(x_poisexp), col = "blue", freq = F,
     xlab = "value",ylab = "probability", main = paste0("hist of poisson from exponential VS truth; lambda
hist(rpois(N, lam), add = T, breaks = max(x_poisexp), col = "red",freq = F)
legend(120, 0.08,legend=c("from exponential", "truth"),
       col=c("blue", "red"), lty=1:1, cex=0.8)
```




```
lam = 100
x_poisexp = pois_exp(N, lam)

hist(x_poisexp, breaks = max(x_poisexp), col = "blue", freq = F,
     xlab = "value",ylab = "probability", main = paste0("hist of poisson from exponential VS truth; lambda
hist(rpois(N, lam), add = T, breaks = max(x_poisexp), col = "red",freq = F)
legend(120, 0.08,legend=c("from exponential", "truth"),
       col=c("blue", "red"), lty=1:1, cex=0.8)
```



Comment:

They are quite similar to `rpois` results, as expected.

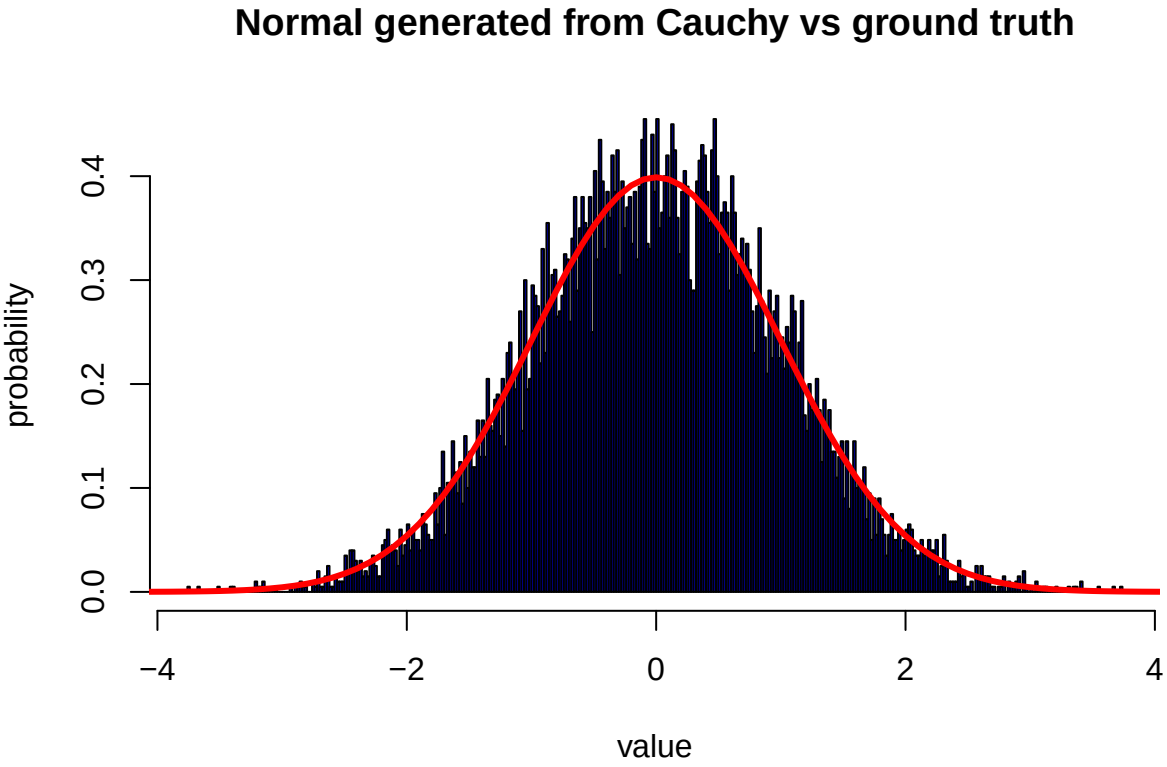
1.6

Use Cauchy to generate normal

I use Cauchy with the the same mean as the normal, and $\beta = \sigma$. $M = \frac{4}{\sqrt{\pi}}$. This parameterization may not yield optimal M.

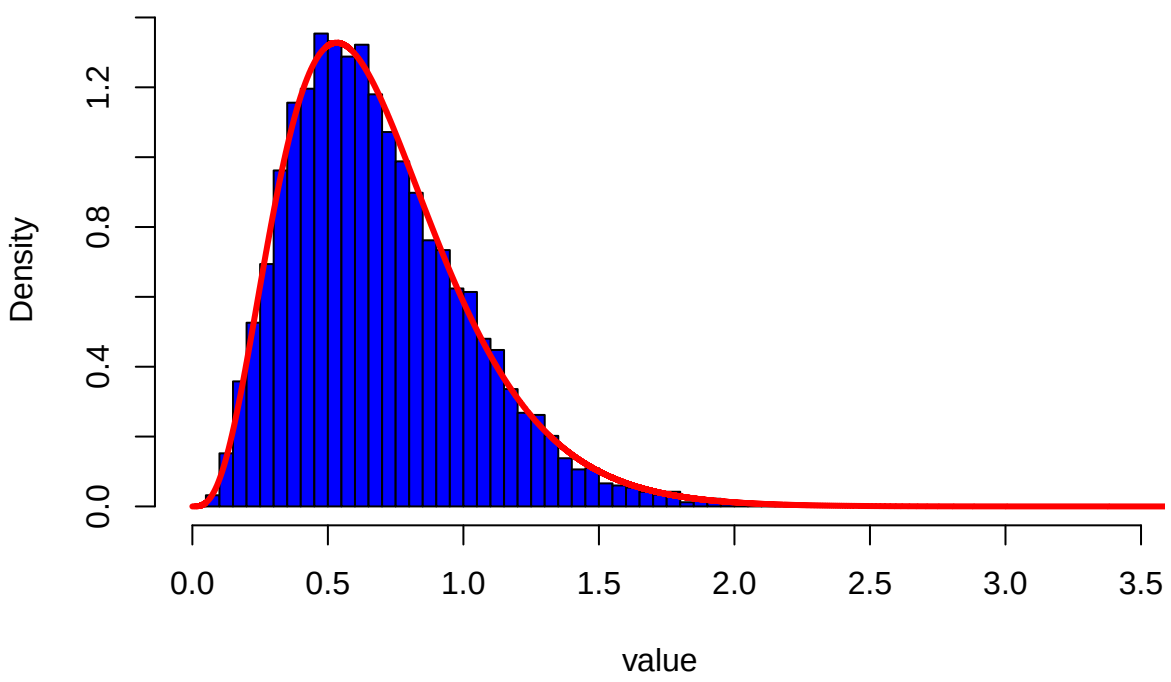
```
rm(list = ls())
set.seed(123)
N = 10000
mu = 0
s = 1

xs = seq(mu - 5*s^2, mu + 5*s^2, 0.1)
hs = dnorm(xs, mu, s)

Cauchy2Normal <- function(mu, s){
  beta = s ## beta is the scale of cauchy
  lam = mu ## lam is mean of cauchy
  M = 4/sqrt(pi)
  accept = FALSE
  while(!accept){
    y = rcauchy(1, lam, beta)
    f = dnorm(y,mu,s)
    g = dcauchy(y,lam,beta)
```

```
    accept = rbinom(1,1,f/(g*M))
  }
  return(y)
}

x_c2n = replicate(N, Cauchy2Normal(mu, s))
hist(x_c2n, freq = F, breaks = 500, col = "blue", xlab = "value", ylab = "probability", main = "Normal gen
lines(xs, hs, col = "red", lwd = 3)
```



Use Gamma(4,7) to generate Gamma(4.3, 6.2) I emperically find M.

```
G2G <- function(M,af,bf, ag, bg){
  accept = FALSE
  while(!accept){
    y = rgamma(1, ag, bg)
    f = dgamma(y,af,bf)
    g = dgamma(y, ag, bg)
    accept = rbinom(1, 1, f/(M*g))
  }
  return(y)
}

xs = seq(0,4, 0.0001)
gs = dgamma(xs,4, 7)
fs = dgamma(xs, 4.3, 6.2)
M = max(fs[fs>0]/gs[gs > 0]) ## use the M determined empirically
x_g2g = replicate(N, G2G(M+1, 4.3, 6.2, 4, 7))
hist(x_g2g, breaks = 100, col = "blue", freq = F, xlab = "value", main = "Gamma(4.3,6.2) generated from Ga
lines(xs, fs, col = "red", lwd = 3)
```




```
rm(list = ls())
```

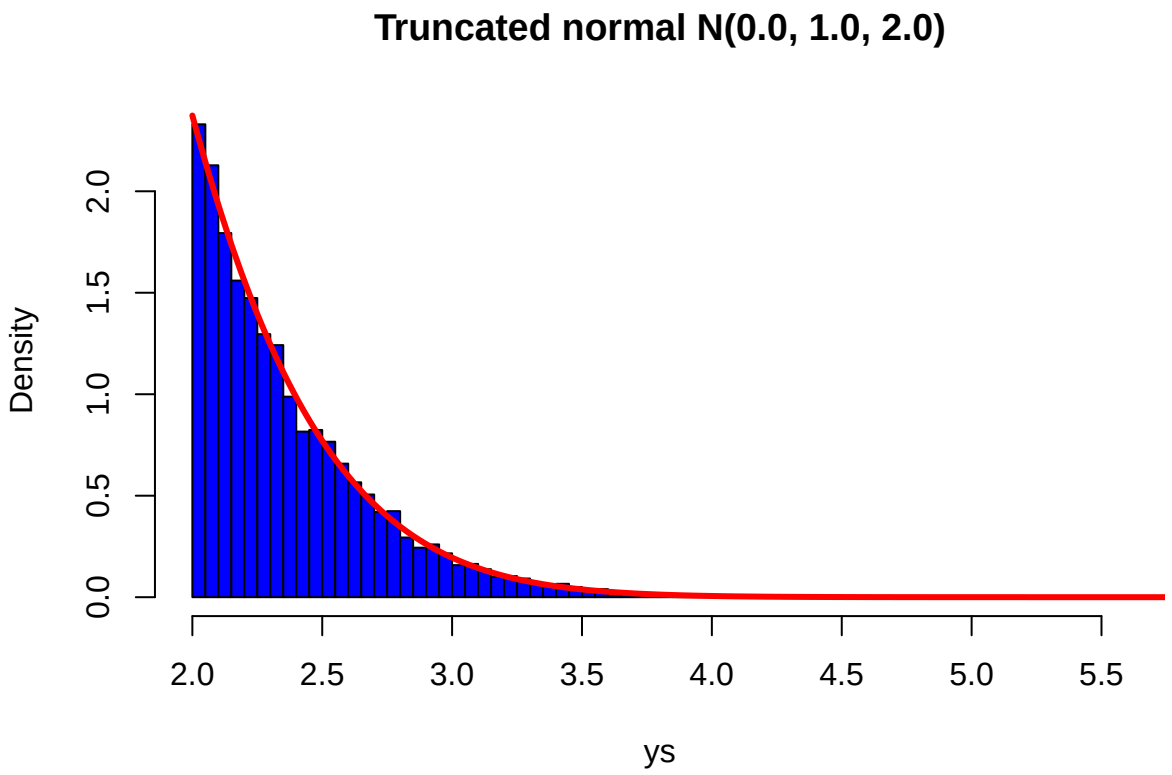
1.7

The proofs are in the writeup.

a)

```
rm(list = ls())
set.seed(12345)
trunc_normal_util <- function(lam, s, a){
  accept = F
  while(!accept){
    y = rnorm(1,lam, s)
    if(y >= a) accept = T
  }
  return(y)
}

trunc_normal <- function(N, lam, s, a){
  return(replicate(N, trunc_normal_util(lam, s, a)))
}

trunc_normal_eval <- function(N ,lam, s, a){
  ys = trunc_normal(N, lam, s, a)
  hist(ys, freq = F, breaks = 100, col = "blue", main = sprintf("Truncated normal N(%0.1f, %0.1f, %0.1f)",
  library(truncnorm)
  xs = seq(a, max(10, 10*a), 0.01)
  fs = dtruncnorm(xs, a, Inf, lam, s)
```

```
  lines(xs, fs, col = "red", lwd = 3)
}


N = 10000
lam = 0
s = 1

#trunc_normal_eval(N, lam, s, 0)
trunc_normal_eval(N, lam, s, 2)
```




```
#trunc_normal_eval(N, lam, s, 3) ## this takes a while
```

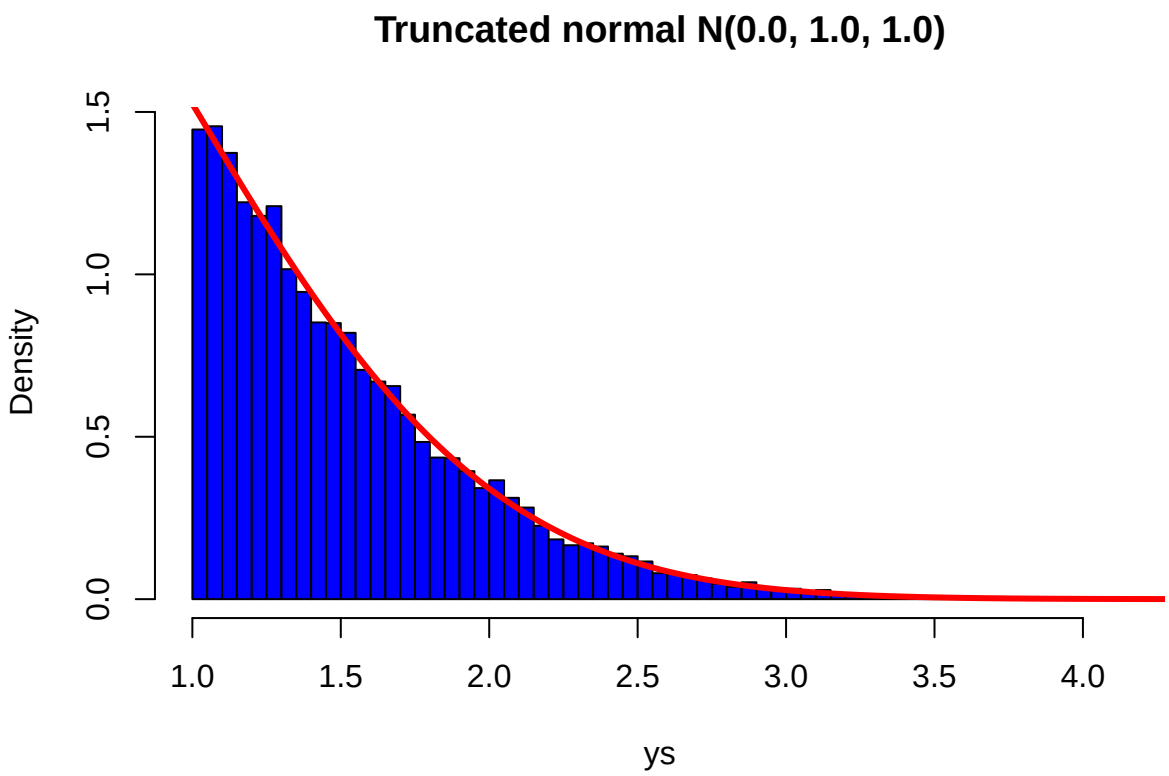
Comment:

It takes much longer when a gets big. This is because $E(T) = 1/\Phi(\frac{\mu-a}{\sigma})$, with T the number of trials for one acceptance. As a gets large, $\Phi(\frac{\mu-a}{\sigma}$ becomes very very small. Therefore, this algorithm is not very efficient.

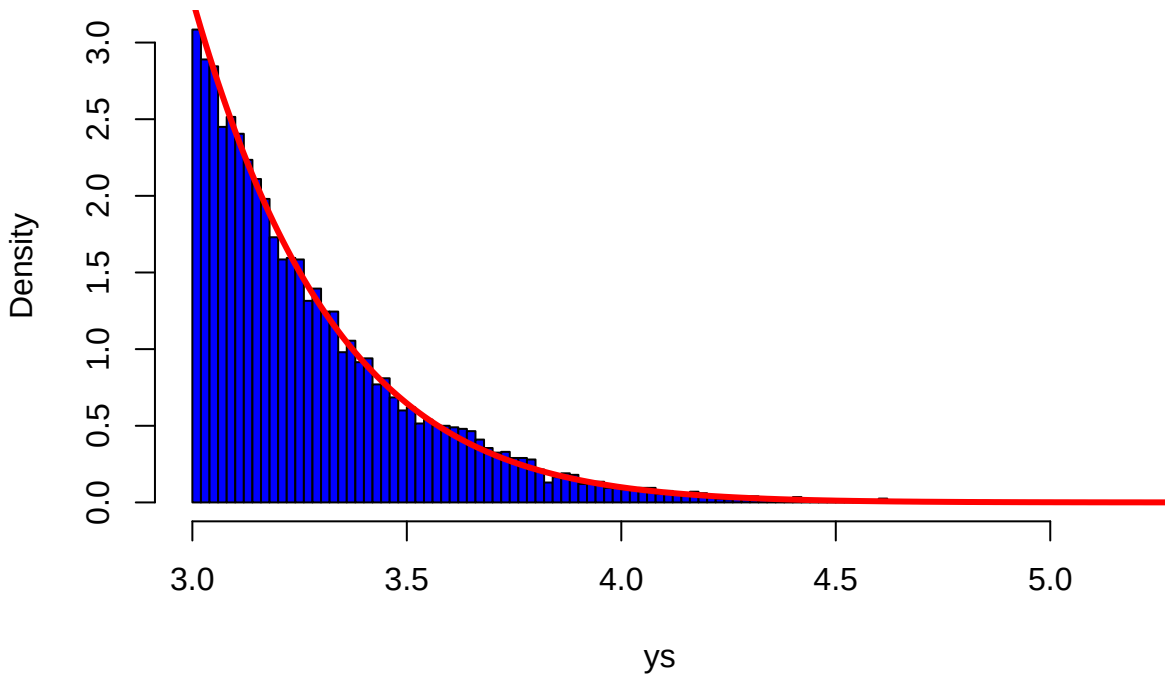
b)

```
set.seed(12345)
trunc_normal_util <- function(a){
  M_tilde = exp(-0.5*a^2) ##
  accept = F
  while(!accept){
    y = rnorm(1,a, 1)
    if(y < a){
      prob = 0
    }
```

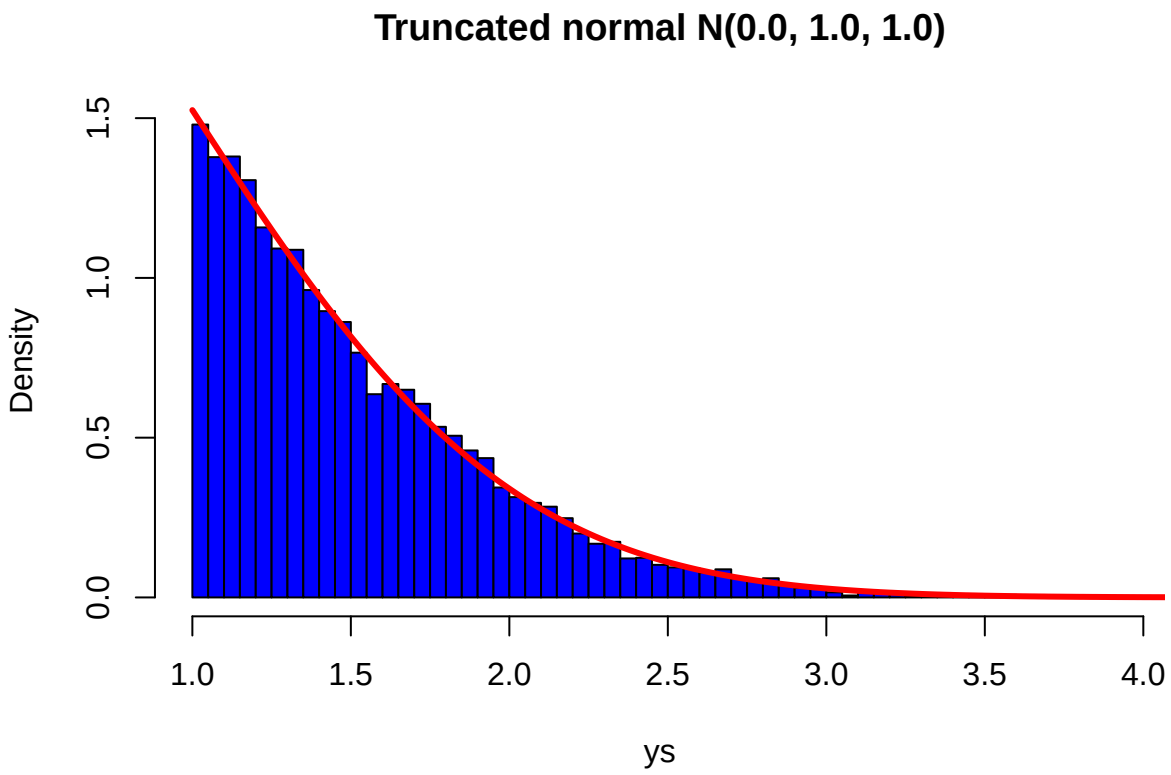
```
    else{
      prob = dnorm(y,0,1)/(M_tilde*dnorm(y,a,1))
    }
    accept = rbinom(1,1, prob)
  }
  return(y)
}

trunc_normal <- function(N,a){
  return(replicate(N, trunc_normal_util(a)))
}

trunc_normal_eval <- function(N ,lam, s, a){
  ys = trunc_normal(N, a)
  hist(ys, freq = F, breaks = 100, col = "blue", main = sprintf("Truncated normal N(%0.1f, %0.1f, %0.1f)",
  library(truncnorm)
  xs = seq(a, max(10, 10*a), 0.01)
  fs = dtruncnorm(xs, a, Inf, lam, s)
  lines(xs, fs, col = "red", lwd = 3)
}


N = 10000
lam = 0
s = 1
#trunc_normal_eval(N, lam, s, 0)
trunc_normal_eval(N, lam, s, 1)
```



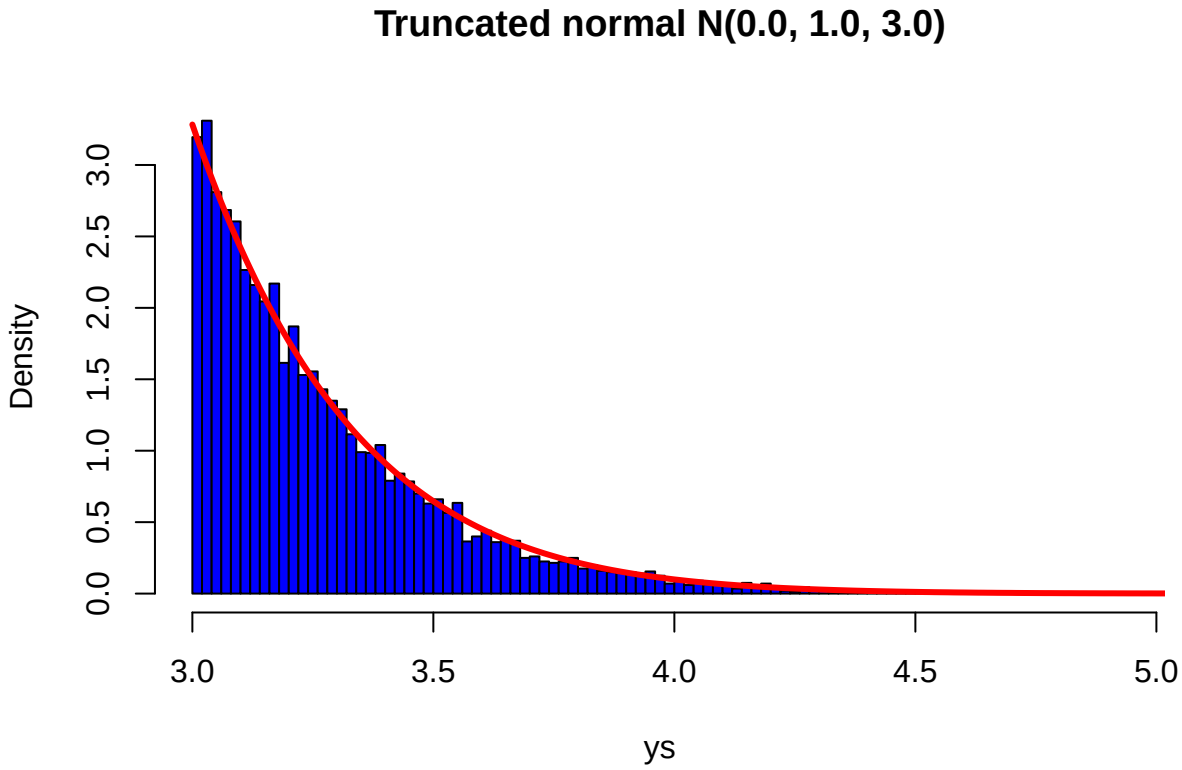

```
trunc_normal_eval(N, lam, s, 3) ## this is much faster than a
```




```r
set.seed(12345)
trunc_normal_util <- function(a){
  M_tilde = exp(-0.5*a^2)/(sqrt(2*pi)*a)
  accept = F
  while(!accept){
    y0 = rexp(1,a)
    y = y0+a
    prob = dnorm(y,0,1)/(M_tilde*dexp(y0,a))
    accept = rbinom(1,1, prob)
  }
  return(y)
}

trunc_normal <- function(N,a){
  return(replicate(N, trunc_normal_util(a)))
}

trunc_normal_eval <- function(N ,lam, s, a){
  ys = trunc_normal(N, a)
  hist(ys, freq = F, breaks = 100, col = "blue", main = sprintf("Truncated normal N(%0.1f, %0.1f, %0.1f)",
  library(truncnorm)
  xs = seq(a, max(10, 10*a), 0.01)
  fs = dtruncnorm(xs, a, Inf, lam, s)
  lines(xs, fs, col = "red", lwd = 3)
}

N = 10000
lam = 0
s = 1
#trunc_normal_eval(N, lam, s, 0)
trunc_normal_eval(N, lam, s, 1)
```




```
trunc_normal_eval(N, lam, s, 3) ## also very fast
```



Comment:

- All methods fit the distribution pretty well.
- Methods in b and c are much faster when `a` is large, as expected from my writeup.

- In my writeup, I show method c is faster than b when a is large.

2.1

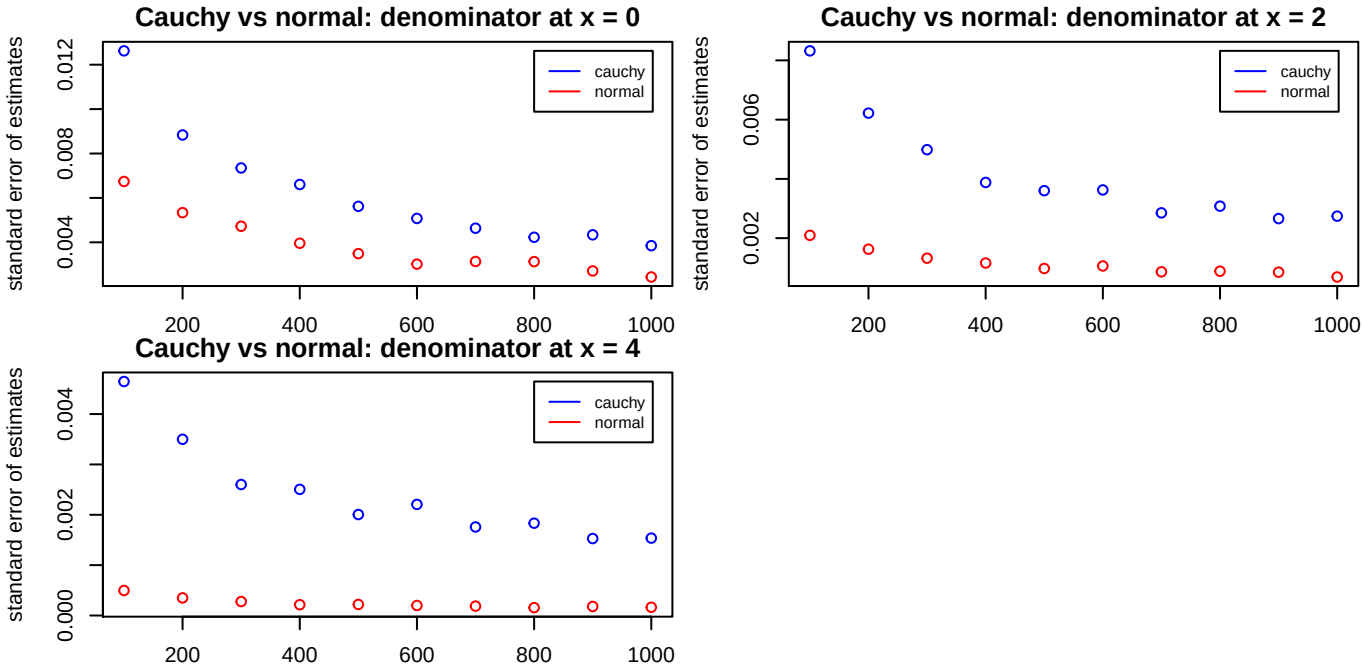
Use Cauchy

```
rm(list = ls())
set.seed(12)
denom_cauchy <- function(N, x){
  h2 <- function(x, theta){
    return(pi* exp(-(x-theta)^2))
  }
  t = rcauchy(N,0,1)
  we = h2(x,t)
  return(mean(we))
}

numer_cauchy <- function(N, x){
  h1 <- function(x, theta){
    return(theta*pi* exp(-(x-theta)^2))
  }
  t = rcauchy(N,0,1)
  we = h1(x,t)
  return(mean(we))
}

denom_normal <- function(N, x){
  h2 <- function(x, theta){
    return(sqrt(2*pi)* exp(-0.5*(x-theta)^2)/(1+theta^2))
  }
  t = rnorm(N,x,1)
  we = h2(x,t)
  return(mean(we))
}

numer_normal <- function(N, x){
  h1 <- function(x, theta){
    return(theta*sqrt(2*pi)* exp(-0.5*(x-theta)^2)/(1+theta^2))
  }
  t = rnorm(N,x,1)
  we = h1(x,t)
  return(mean(we))
}
```

Convergence analysis for denominator

```
## convergence and 3-digits accuracy for Denominator estimated with Cauchy
n_exper = 100
xs = c(0,2,4)
par(mfrow = c(3, 1))
par(mar=c(2,6,2,6))
par(fig=c(0,7,6,10)/10)
for(x in xs){
```

```r
  Ns = seq(100,1000,100)
  sde1 = c()
  sde2 = c()
  for(N in Ns){
    exper1 = replicate(n_exper, denom_cauchy(N,x))
    sde1 = c(sde1, sd(exper1)/sqrt(n_exper))
    exper2 = replicate(n_exper, denom_normal(N,x))
    sde2 = c(sde2, sd(exper2)/sqrt(n_exper))
  }
  #dev.new(width=5, height=4, unit="in")
  plot(Ns, sde1, col = 'blue', ylim = c(min(sde1, sde2), max(sde1,sde2)),main = paste0("Cauchy vs normal: 
  points(Ns, sde2, col = "red")
  legend(x = 800, y =  max(sde1,sde2), legend = c("cauchy", "normal"), col = c("blue", "red"), lty = 1:1, 
}
```



estimate for Denominator

I need to choose n so that $\frac{2\hat{\sigma}}{\sqrt{n}} < 0.001$. $n = 1000000$ is big enough as previous convergence analysis shows that standard error is much smaller than 0.005 at this n.

```r
cat(sprintf("From Cauchy:\n"))
```

```
## From Cauchy:
```

```r
for(x in xs){
  est = denom_cauchy(1e6,x)
  cat(sprintf("Estimate for Denominator is %0.3f for x = %d\n", round(est,3), x))
}
```

```
## Estimate for Denominator is 1.343 for x = 0
## Estimate for Denominator is 0.440 for x = 2
## Estimate for Denominator is 0.113 for x = 4
```

```r
cat(sprintf("From Normal:\n"))
```

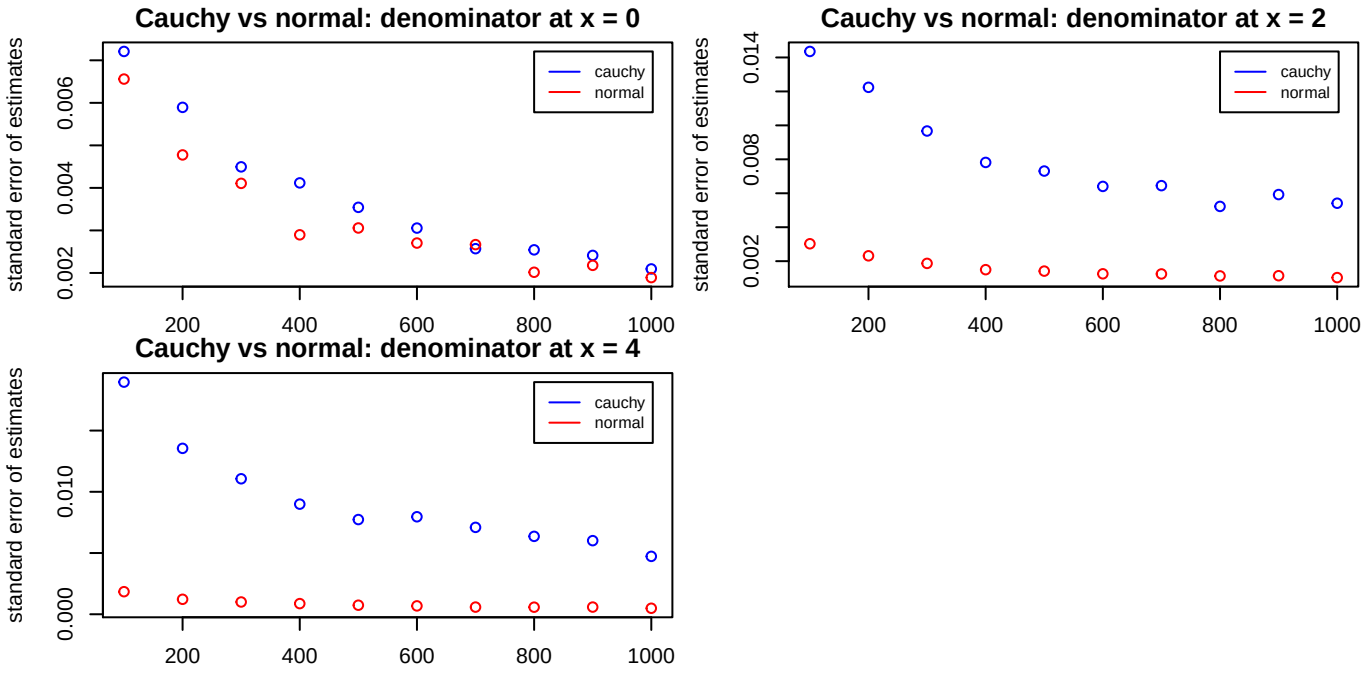
```
## From Normal:
```

```
for(x in xs){
  est = denom_normal(1e6,x)
  cat(sprintf("Estimate for Denominator is %0.3f for x = %d\n", round(est,3), x))
}
```

```
## Estimate for Denominator is 1.343 for x = 0
## Estimate for Denominator is 0.441 for x = 2
## Estimate for Denominator is 0.114 for x = 4
```

Convergence analysis for numerator

```
## convergence and 3-digits accuracy for Denominator estimated with Cauchy
n_exper = 100
xs = c(0,2,4)
par(mfrow = c(3, 1))
par(mar=c(2,6,2,6))
par(fig=c(0,7,6,10)/10)
for(x in xs){
  Ns = seq(100,1000,100)
  sde1 = c()
  sde2 = c()
  for(N in Ns){
    exper1 = replicate(n_exper, numer_cauchy(N,x))
    sde1 = c(sde1, sd(exper1)/sqrt(n_exper))
    exper2 = replicate(n_exper, numer_normal(N,x))
    sde2 = c(sde2, sd(exper2)/sqrt(n_exper))
  }
  plot(Ns, sde1, col = 'blue', ylim = c(min(sde1, sde2), max(sde1,sde2)) ,main = paste0("Cauchy vs normal:
  points(Ns, sde2, col = "red")
  legend(x = 800, y =  max(sde1,sde2), legend = c("cauchy", "normal"), col = c("blue", "red"), lty = 1:1,
}
```



estimate for numerator

I need to choose n so that $\frac{2\hat{\sigma}}{\sqrt{n}} < 0.001$. $n = 1000000$ is big enough as previous convergence analysis shows that standard error is much smaller than 0.005 at this n.

```
cat(sprintf("From Cauchy:\n"))
```

```
## From Cauchy:
```

```
for(x in xs){
  est = numer_cauchy(1e6,x)
  cat(sprintf("Estimate for Denominator is %0.3f for x = %d\n", round(est,3), x))
}
```

```
## Estimate for Denominator is -0.000 for x = 0
## Estimate for Denominator is 0.698 for x = 2
## Estimate for Denominator is 0.427 for x = 4
```

```
cat(sprintf("From Normal:\n"))
```

```
## From Normal:
```

```
for(x in xs){
  est = numer_normal(1e6,x)
  cat(sprintf("Estimate for Denominator is %0.3f for x = %d\n", round(est,3), x))
}
```

```
## Estimate for Denominator is 0.001 for x = 0
## Estimate for Denominator is 0.699 for x = 2
## Estimate for Denominator is 0.426 for x = 4
```

Comparison

- I use Cauchy(0,1) and Normal(x,1) to compute the integrals.
- Normal gets smaller variance for the same value of N and x, for larger x in particular.

2.2

a)

- First, I want to sample from posterior. Using conclusion from Ex 1.5, we need to find $\hat{M}$ s.t. $\frac{P(x|\theta)P(\theta)}{g(\theta)} < \hat{M}$. Then we accept sample from $g(\theta)$ with probability $\frac{P(x|\theta)P(\theta)}{g(\theta)\hat{M}}$ .We can use candidate $g(\theta) = p(\theta)$. Then we can choose $\hat{M} = \frac{1}{\sqrt{2\pi}}$.
- Then, I use the posterior mean as estimator

```
set.seed(12345)
pos_cauchy <- function(x){
  M = 1/sqrt(2*pi)
  accept = F
  while(!accept){
    y = rcauchy(1,0,1)
    prob = dnorm(x,y,1)/M
    accept = rbinom(1,1,prob)
  }
  return(y)
```

```
}
N = 10000
xs = c(0,2,4)
for(x in xs){
  cat(sprintf("Estimate for posterior mean: %f with x = %d\n", mean(replicate(N, pos_cauchy(x))), x))
}
```

```
## Estimate for posterior mean: -0.000567 with x = 0
## Estimate for posterior mean: 1.287146 with x = 2
## Estimate for posterior mean: 3.426939 with x = 4
```

b)

```
BayesEst_cauchy_same <- function(N, x){
  h2 <- function(x, theta){
    return(pi* exp(-(x-theta)^2))
  }
  h1 <- function(x, theta){
    return(theta*pi* exp(-(x-theta)^2))
  }
  t = rcauchy(N,0,1)
  we1 = h1(x,t)
  we2 = h2(x,t)
  return(mean(we1)/mean(we2))
}

BayesEst_cauchy_diff <- function(N,x){
  return(numer_cauchy(N,x)/denom_cauchy(N,x))
}

N = 10000
xs = c(0,2,4)
for(x in xs){
  cat(sprintf("Estimate with same variables: %f with x = %d\n", BayesEst_cauchy_same(N,x), x))
}
```

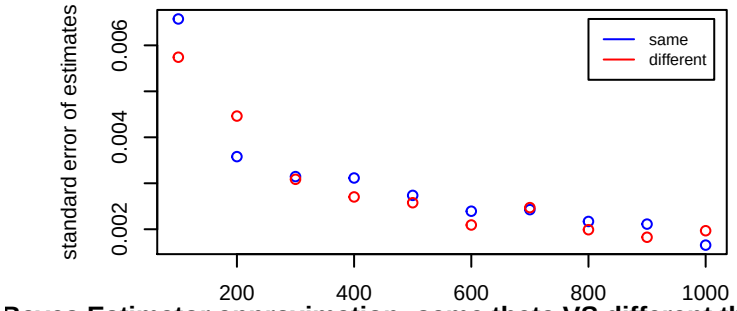
```
## Estimate with same variables: 0.004696 with x = 0
## Estimate with same variables: 1.579141 with x = 2
## Estimate with same variables: 3.746364 with x = 4
```

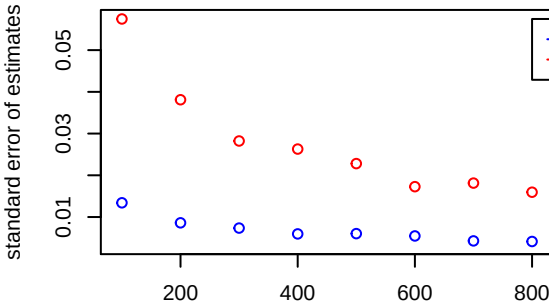
```
N = 10000
xs = c(0,2,4)
for(x in xs){
  cat(sprintf("Estimate with diff variables: %f with x = %d\n", BayesEst_cauchy_diff(N,x), x))
}
```

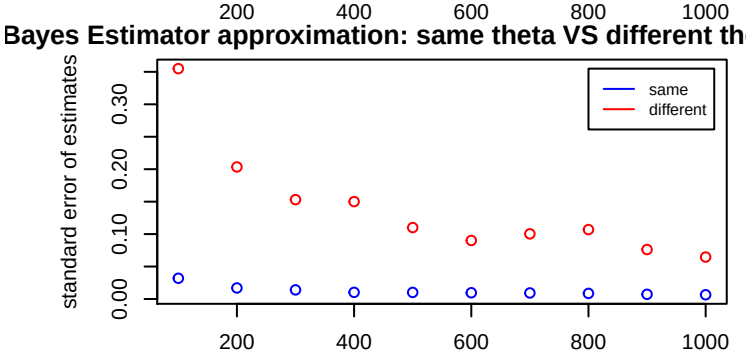
```
## Estimate with diff variables: 0.010053 with x = 0
## Estimate with diff variables: 1.666567 with x = 2
## Estimate with diff variables: 3.948564 with x = 4
```

```
## convergence and 3-digits accuracy for Denominator estimated with Cauchy
n_exper = 100
xs = c(0,2,4)
par(mfrow = c(3, 1))
par(mar=c(2,6,2,6))
par(fig=c(0,7,6,10)/10)
for(x in xs){
```

```
  Ns = seq(100,1000,100)
  sde1 = c()
  sde2 = c()
  for(N in Ns){
    exper1 = replicate(n_exper, BayesEst_cauchy_same(N,x))
    sde1 = c(sde1, sd(exper1)/sqrt(n_exper))
    exper2 = replicate(n_exper, BayesEst_cauchy_diff(N,x))
    sde2 = c(sde2, sd(exper2)/sqrt(n_exper))
  }
  plot(Ns, sde1, col = 'blue', ylim = c(min(sde1, sde2), max(sde1,sde2)) ,main = paste0("Bayes Estimator a
  points(Ns, sde2, col = "red")
  legend(x = 800, y =  max(sde1,sde2), legend = c("same", "different"), col = c("blue", "red"), lty = 1:1,
}
```







Comment: * when x is larger, using the same variable consistently have smaller standard variance.

c) repoeat for normal

Estimate by sample from posterior

Similarly, note the pdf for two normals cancel out (because of the symmetry between x and theta). Thus, we can choose $\hat{M} = 1/\pi$

```
set.seed(123)
pos_normal <- function(x){
  M = 1/pi
  #M = pi
  accept = F
  while(!accept){
    y = rnorm(1,x,1)
    #prob = dcauchy(y,0,1)*dnorm(x,y,1)/(M*dnorm(y,x,1))
    prob = dcauchy(y,0,1)/M
    accept = rbinom(1,1,prob)
  }
  return(y)
```

```
}
N = 10000
xs = c(0,2,4)
for(x in xs){
  cat(sprintf("Estimate for posterior mean: %f with x = %d\n", mean(replicate(N, pos_normal(x))), x))
}
```

```
## Estimate for posterior mean: -0.011135 with x = 0
## Estimate for posterior mean: 1.282614 with x = 2
## Estimate for posterior mean: 3.425233 with x = 4
```

Estimate by estimating the two integrals

```
BayesEst_normal_same <- function(N, x){
  h2 <- function(x, theta){
    return(sqrt(2*pi)* exp(-0.5*(x-theta)^2)/(1+theta^2))
  }
  h1 <- function(x, theta){
    return(theta*sqrt(2*pi)* exp(-0.5*(x-theta)^2)/(1+theta^2))
  }
  t = rnorm(N,x,1)
  we1 = h1(x,t)
  we2 = h2(x,t)
  return(mean(we1)/mean(we2))
}

BayesEst_normal_diff <- function(N, x){
  return(numer_normal(N,x)/denom_normal(N,x))
}



N = 10000
xs = c(0,2,4)
for(x in xs){
  cat(sprintf("Estimate with same variables: %f with x = %d\n", BayesEst_normal_same(N,x), x))
}
```

```
## Estimate with same variables: 0.001010 with x = 0
## Estimate with same variables: 1.580572 with x = 2
## Estimate with same variables: 3.736317 with x = 4
```

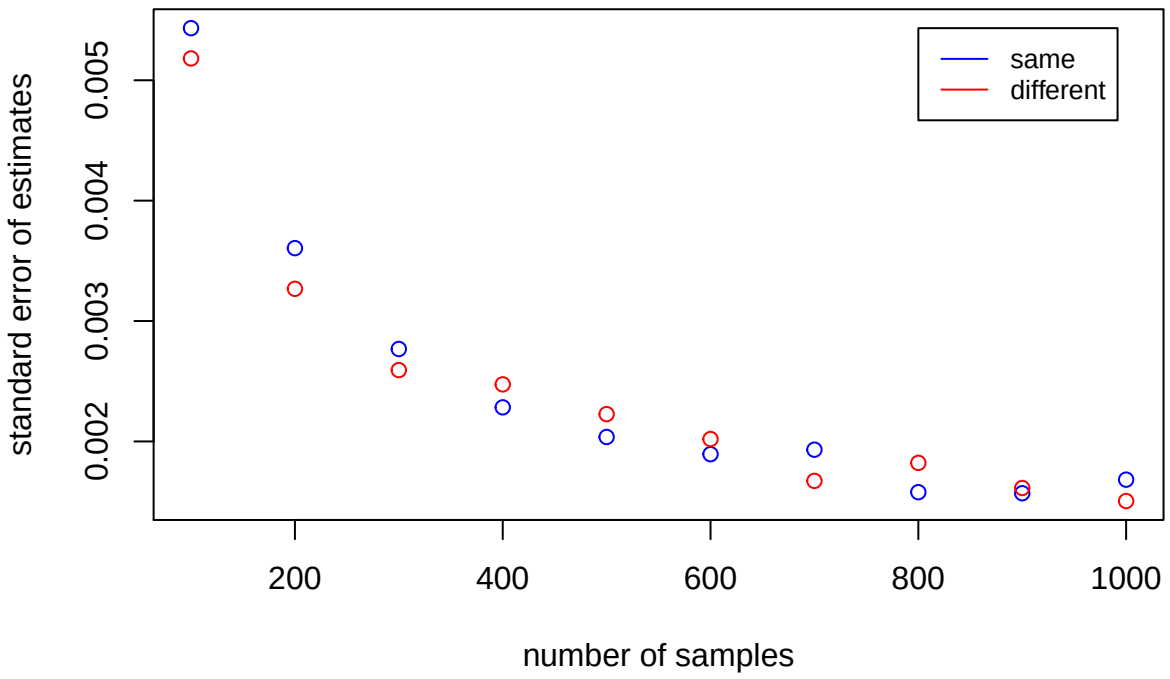
```
N = 10000
xs = c(0,2,4)
for(x in xs){
  cat(sprintf("Estimate with diff variables: %f with x = %d\n", BayesEst_normal_diff(N,x), x))
}
```

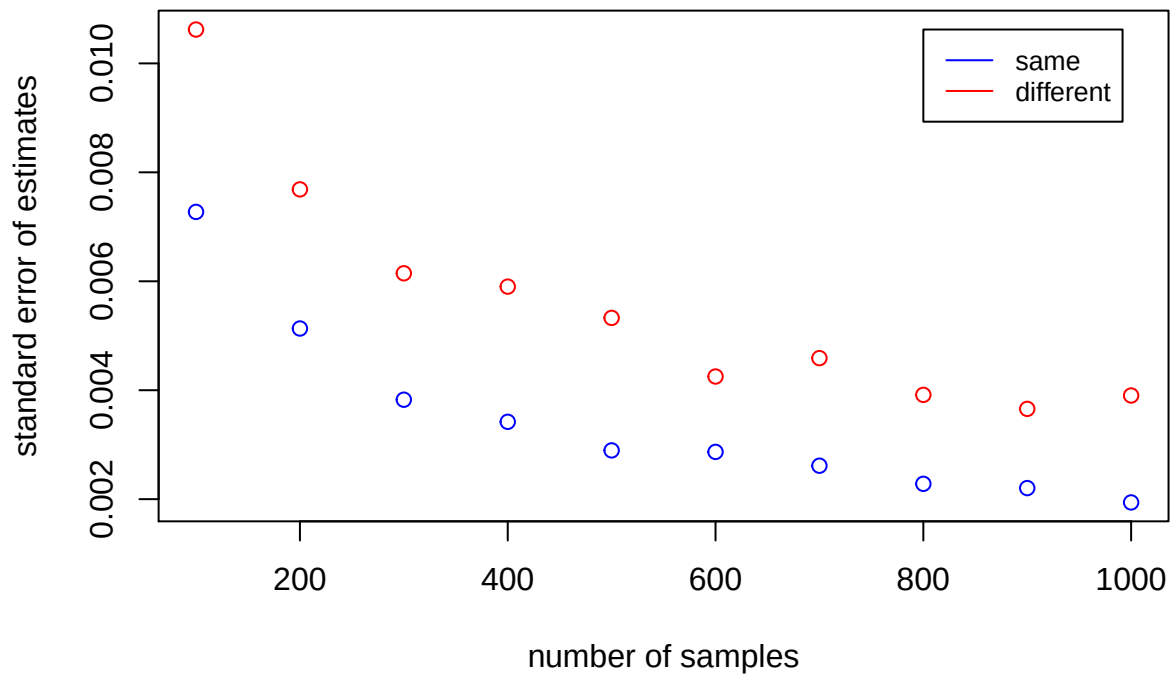
```
## Estimate with diff variables: 0.002545 with x = 0
## Estimate with diff variables: 1.590851 with x = 2
## Estimate with diff variables: 3.741536 with x = 4
```

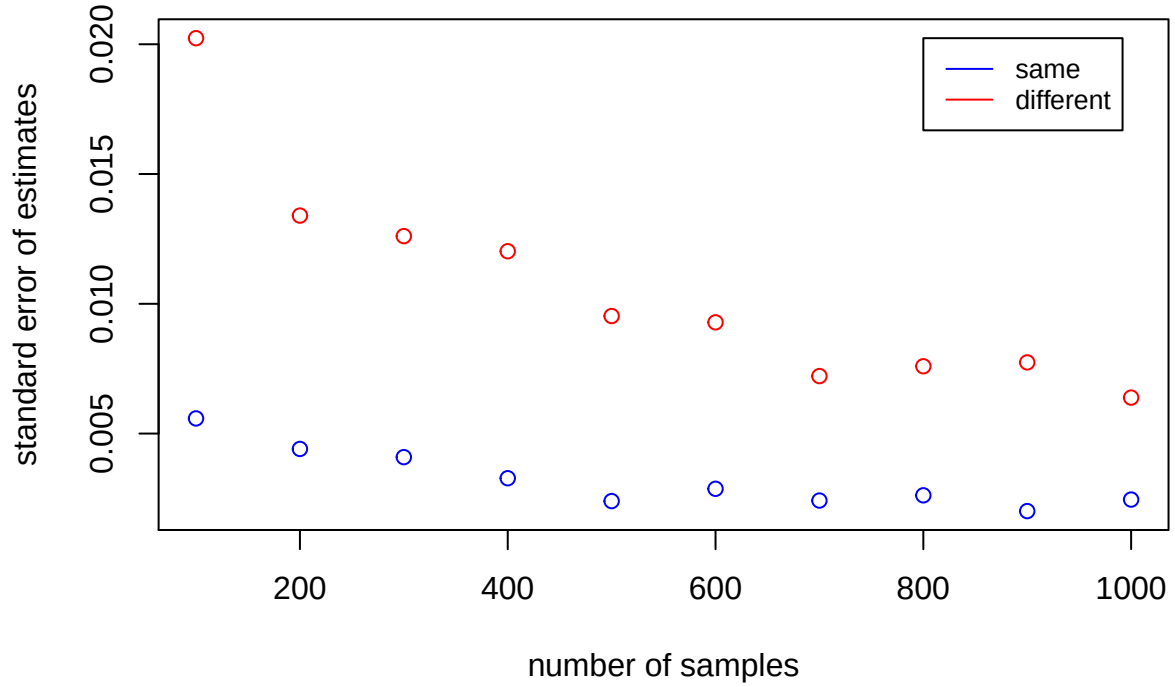
```
## convergence and 3-digits accuracy for Denominator estimated with Cauchy
n_exper = 100
xs = c(0,2,4)
for(x in xs){
  Ns = seq(100,1000,100)
  sde1 = c()
```

```
  sde2 = c()
  for(N in Ns){
    exper1 = replicate(n_exper, BayesEst_normal_same(N,x))
    sde1 = c(sde1, sd(exper1)/sqrt(n_exper))
    exper2 = replicate(n_exper, BayesEst_normal_diff(N,x))
    sde2 = c(sde2, sd(exper2)/sqrt(n_exper))
  }
  plot(Ns, sde1, col = 'blue', ylim = c(min(sde1, sde2), max(sde1,sde2)) ,main = paste0("Bayes Estimator a
  points(Ns, sde2, col = "red")
  legend(x = 800, y =  max(sde1,sde2), legend = c("same", "different"), col = c("blue", "red"), lty = 1:1,
}
```

'es Estimator approximation with Normal: same theta VS different theta







2.3

a)

(show in writep)

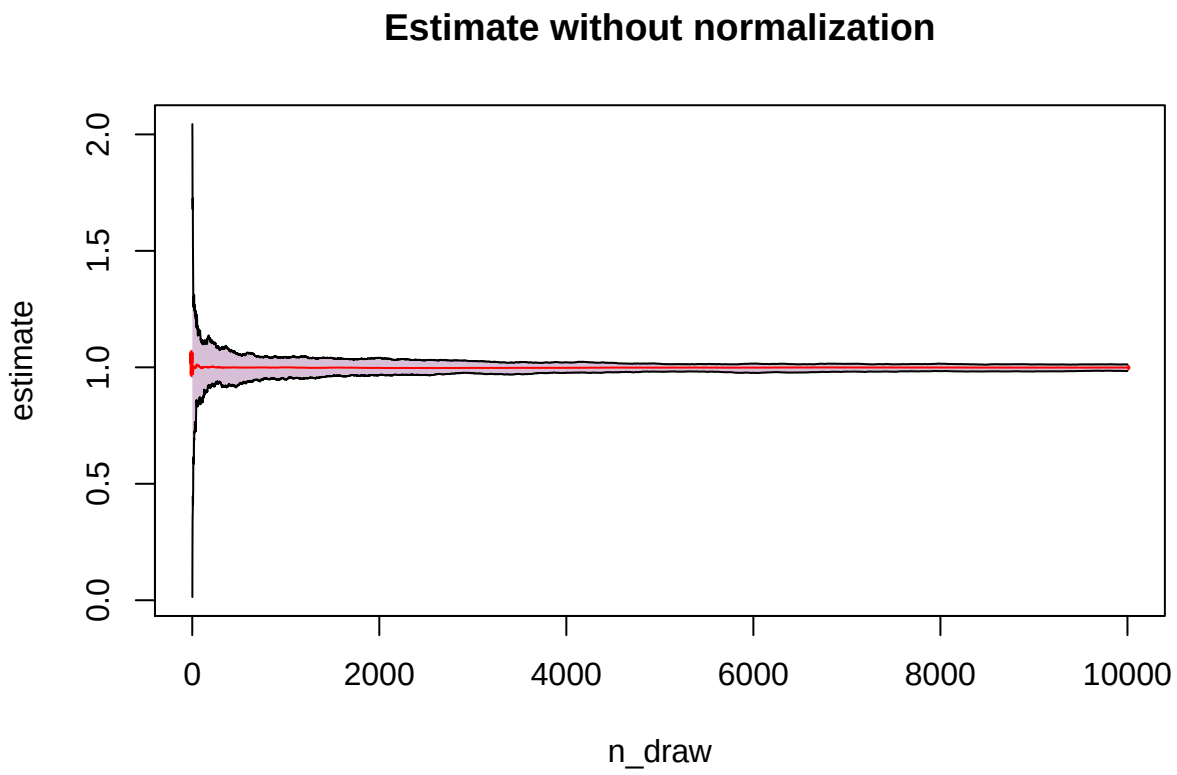
b)

we = f/g

```
f <- function(x){return(exp(x - exp(x)))}
x = matrix(rnorm(100*10^4), ncol = 100)
we = f(x)/dnorm(x)
```

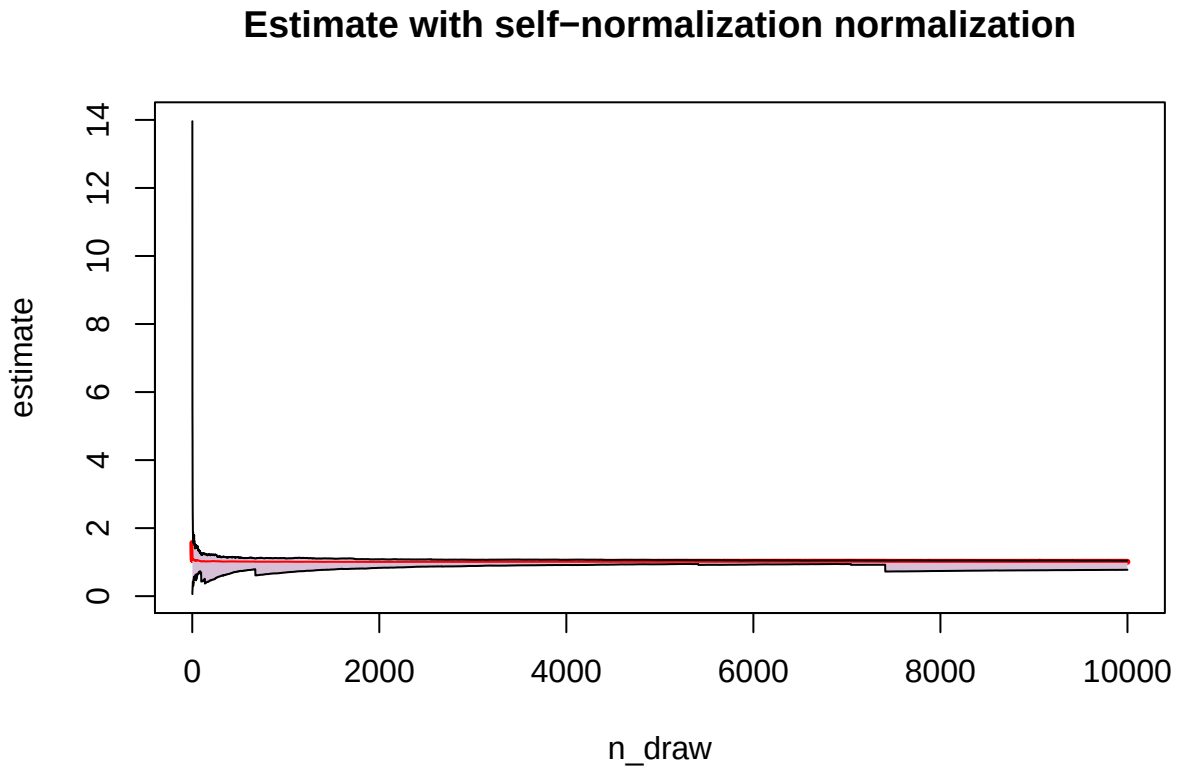
c)

```
ts = 1:10^4
unnorm =  apply(we*exp(x), 2, cumsum)/(1:10^4)

unnorm_mean = apply(unnorm, 1, mean)
unnorm_max = apply(unnorm, 1, max)
unnorm_min = apply(unnorm, 1, min)

plot(ts, unnorm_mean, col = "red", ylim = range(unnorm), xlab = "n_draw", ylab = "estimate", main = "Estim
polygon(c(ts,rev(ts)),c(unnorm_min, rev(unnorm_max)),col="thistle",border=NA)
lines(ts, unnorm_mean, type = "l", col = "red")
lines(ts, unnorm_max)
lines(ts, unnorm_min)
```




```
ts = 1:10^4
renorm = apply(we*exp(x), 2, cumsum)/apply(we, 2, cumsum)

renorm_mean = apply(renorm, 1, mean)
renorm_max = apply(renorm, 1, max)
renorm_min = apply(renorm, 1, min)

plot(ts, renorm_mean, col = "red", ylim = range(renorm), xlab = "n_draw", ylab = "estimate", main = "Estim
```

```
polygon(c(ts,rev(ts)),c(renorm_min, rev(renorm_max)),col="thistle",border=NA)
lines(ts, renorm_mean, type = "l", col = "red")
lines(ts, renorm_max)
lines(ts, renorm_min)
```



2.4

a)

```
rm(list = ls())
set.seed(12345)
## since it is symmetric, I only need to compute integral starting from 0; then multiply by 2
end = 100
space = 1e-5
xs = seq(0,50,space)
myfunc <- function(x){return(abs(x)*dt(x, df = 3))}
est_rec = 2*sum(myfunc(xs))*space
cat(sprintf("estimate from rectangle method:%f\n", est_rec))
```

```
## estimate from rectangle method:1.101336
```

b)

```
n_sample = 1e4
n_exper = 100
x = matrix(rt(n = n_sample*n_exper, df = 3), ncol = n_exper)
we = dt(x, df = 3)/dt(x, df = 3)
```

```r
ests = colSums(we*abs(x))/n_sample
cat(sprintf("mean of estimator: %f\n", mean(ests)))
```

```
## mean of estimator: 1.101142
```

```r
cat(sprintf("variance of estimator: %f\n", var(ests)))
```

```
## variance of estimator: 0.000149
```

c)

```r
n_sample = 1e4
n_exper = 100
x = matrix(rt(n = n_sample*n_exper, df = 1), ncol = n_exper)
we = dt(x, df = 3)/dt(x, df = 1)
ests = colSums(we*abs(x))/n_sample
cat(sprintf("mean of estimator: %f\n", mean(ests)))
```

```
## mean of estimator: 1.101306
```

```r
cat(sprintf("variance of estimator: %f\n", var(ests)))
```

```
## variance of estimator: 0.000046
```

d)

```r
n_sample = 1e4
n_exper = 100
x = matrix(rnorm(n = n_sample*n_exper), ncol = n_exper)
we = dt(x, df = 3)/dnorm(x)
ests = colSums(we*abs(x))/n_sample
cat(sprintf("mean of estimator: %f\n", mean(ests)))
```

```
## mean of estimator: 1.069457
```

```r
cat(sprintf("variance of estimator: %f\n", var(ests)))
```

```
## variance of estimator: 0.521204
```

Comment:

IS refering to f1 is most accurate, followed by f3, and followed by normal.

2.5

a)

```r
rm(list = ls())
set.seed(12345)
n_sample = 1e4
N = 100
decide_in <- function(xyz){
  return(xyz[1]^(1/2) + xyz[2]^(1/3) + xyz[3]^(1/4) < 1)
```

```
}

est_rec <- function(n_sample){
  draws = matrix(runif(3*n_sample,0,1), ncol = 3)
  result = apply(draws, 1, decide_in)
  est = sum(result)/n_sample
  return(est)
}

est_rec_exper = replicate(N, est_rec(n_sample))
cat(sprintf("mean of estimatee: %f\n", mean(est_rec_exper)))
```

```
## mean of estimatee: 0.000775
```

```
cat(sprintf("var of estimatee: %12e\n", round(var(est_rec_exper),12)))
```

```
## var of estimatee: 6.896500e-08
```

b)

```
rej_sample <- function(n_sample){
  draws = matrix(runif(3*n_sample,0,1), ncol = 3)
  result = apply(draws, 1, decide_in)
  X1 = draws[result == T,1]
  X2 = draws[result == T,2]
  X3 = draws[result == T,3]
  return(list(n_acc = length(X1),X1=X1,X2=X2,X3=X3))
}
n_sample = 1e6
out = rej_sample(n_sample)
cat(sprintf("Number of accepted samples: %d\n", mean(out$n_acc)))
```

```
## Number of accepted samples: 774
```

```
cat(sprintf("Estimated Mean of X1: %5e\n", mean(out$X1)))
```

```
## Estimated Mean of X1: 5.512897e-02
```

```
cat(sprintf("Estimated Mean of X2: %5e\n", mean(out$X2)))
```

```
## Estimated Mean of X2: 4.139326e-02
```

```
cat(sprintf("Estimated Mean of X3: %5e\n", mean(out$X3)))
```

```
## Estimated Mean of X3: 5.250921e-02
```

c)

```
my_Aarea = mean(est_rec_exper)
weight <- function(xyz,A_area= my_Aarea){
  if(decide_in(xyz)){
    we = (1/A_area) * exp(sum(log(dexp(xyz, rate = 0.1))))
    return(we)
  }
  return(0)
}
```

```r
n_sample = 1e6
draws = matrix(rexp(3*n_sample,rate = 0.1), ncol = 3)
we = apply(draws,1,weight)
est = mean(we * draws[,1])
cat(sprintf("estimated value: %12e\n", est))
```

```
## estimated value: 0.000000e+00
```