STAT 812: Computational Statistics

Rejection Sampling

Longhai Li

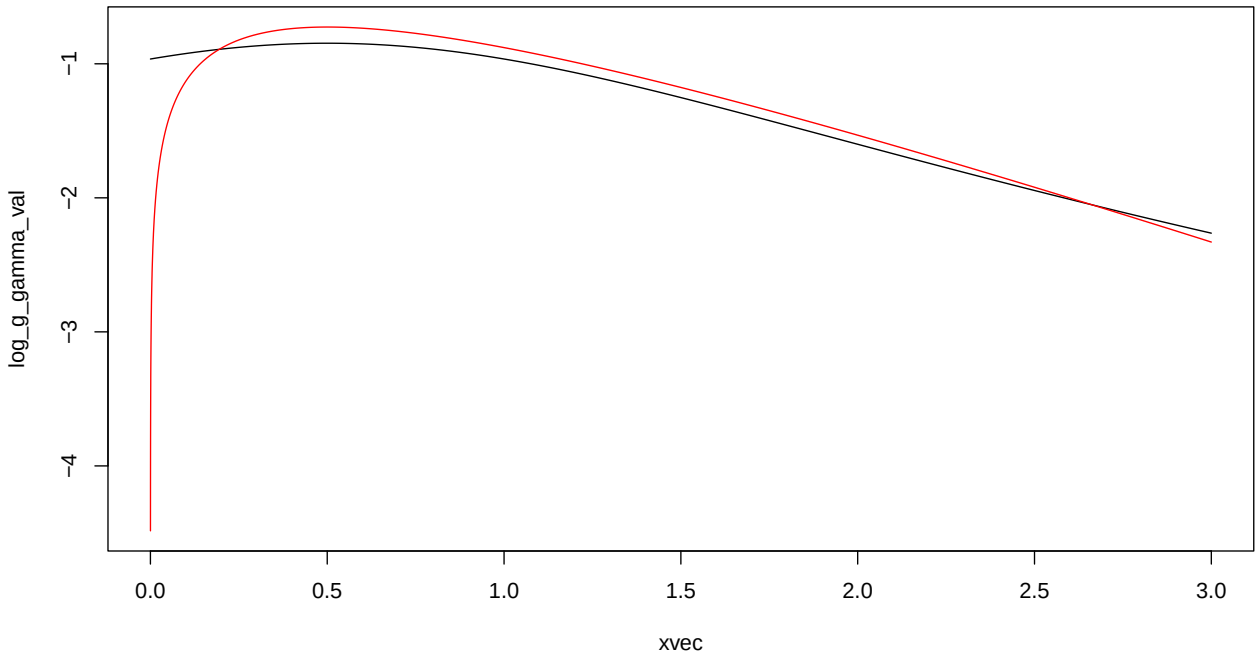
2024-09-17

Contents

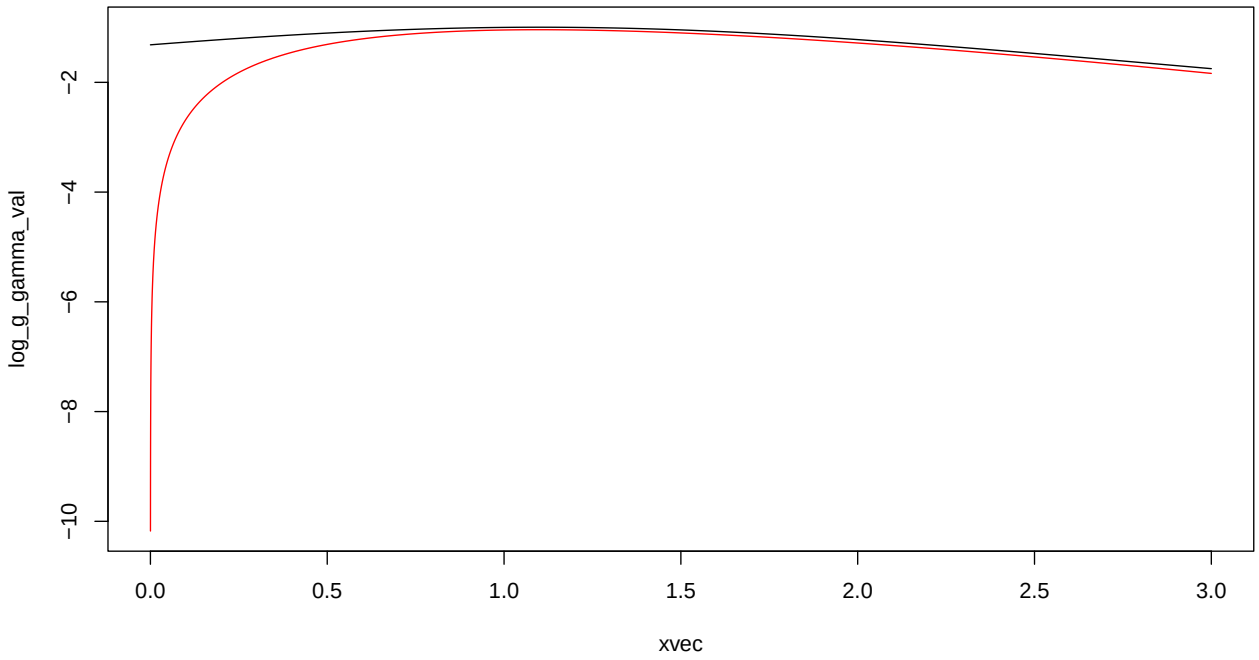
1 Rejection Sampling for Gamma Distribution

1.1 Envolop Function

```
# Note: this is only a toy example demonstrating how to program rejection sampling
# this is not a good sampling scheme for gamma distribution.
# Do not use it for serious applications that demand high efficiency.


#log of a function which is always above the Gamma density function
# alpha must be > 2
log_g_gamma <- function(x, alpha)
{
   (alpha-1) * (log(alpha-1) - 1) - log( 1 + (x-(alpha-1))^2 / (2*alpha-1) )
}

## look at the approximating functionn
xvec <- seq (0, 3, by = 0.0001)
alpha <- 1.5
log_g_gamma_val <- log_g_gamma(xvec, alpha = alpha)
log_gamma_val <- dgamma (xvec, shape = alpha, log =T)
ylim <- range (log_g_gamma_val, log_gamma_val, finite = T)
plot (xvec, log_g_gamma_val, col = "black", type = "l", ylim = ylim )
points (xvec,log_gamma_val, col = "red",type = "l")
```

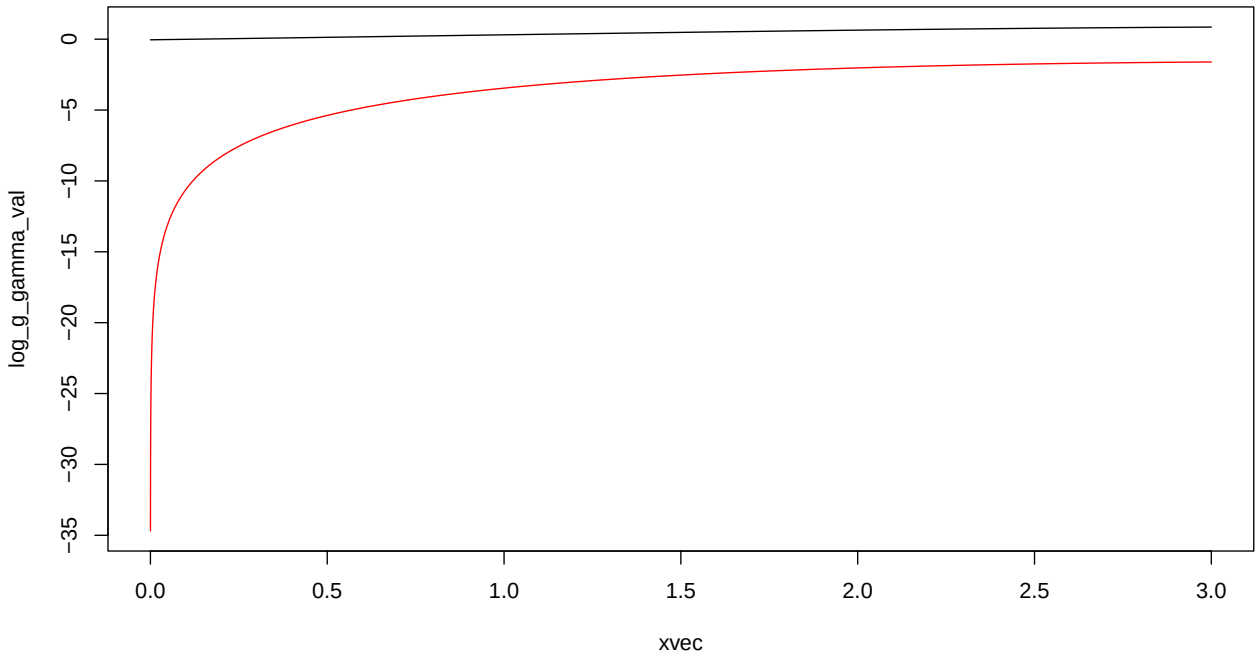



```
## look at the approximating functionn
xvec <- seq (0, 3, by = 0.0001)
alpha <- 2.1
log_g_gamma_val <- log_g_gamma(xvec, alpha = alpha)
log_gamma_val <- dgamma (xvec, shape = alpha, log =T)
ylim <- range (log_g_gamma_val, log_gamma_val, finite = T)
plot (xvec, log_g_gamma_val, col = "black", type = "l", ylim = ylim )
points (xvec,log_gamma_val, col = "red",type = "l")
```




```
## look at the approximating functionn
xvec <- seq (0, 3, by = 0.0001)
alpha <- 4.5
```

```r
log_g_gamma_val <- log_g_gamma(xvec, alpha = alpha)
log_gamma_val <- dgamma (xvec, shape = alpha, log =T)
ylim <- range (log_g_gamma_val, log_gamma_val, finite = T)
plot (xvec, log_g_gamma_val, col = "black", type = "l", ylim = ylim )
points (xvec,log_gamma_val, col = "red",type = "l")
```



1.2 Rejection Sampling Function

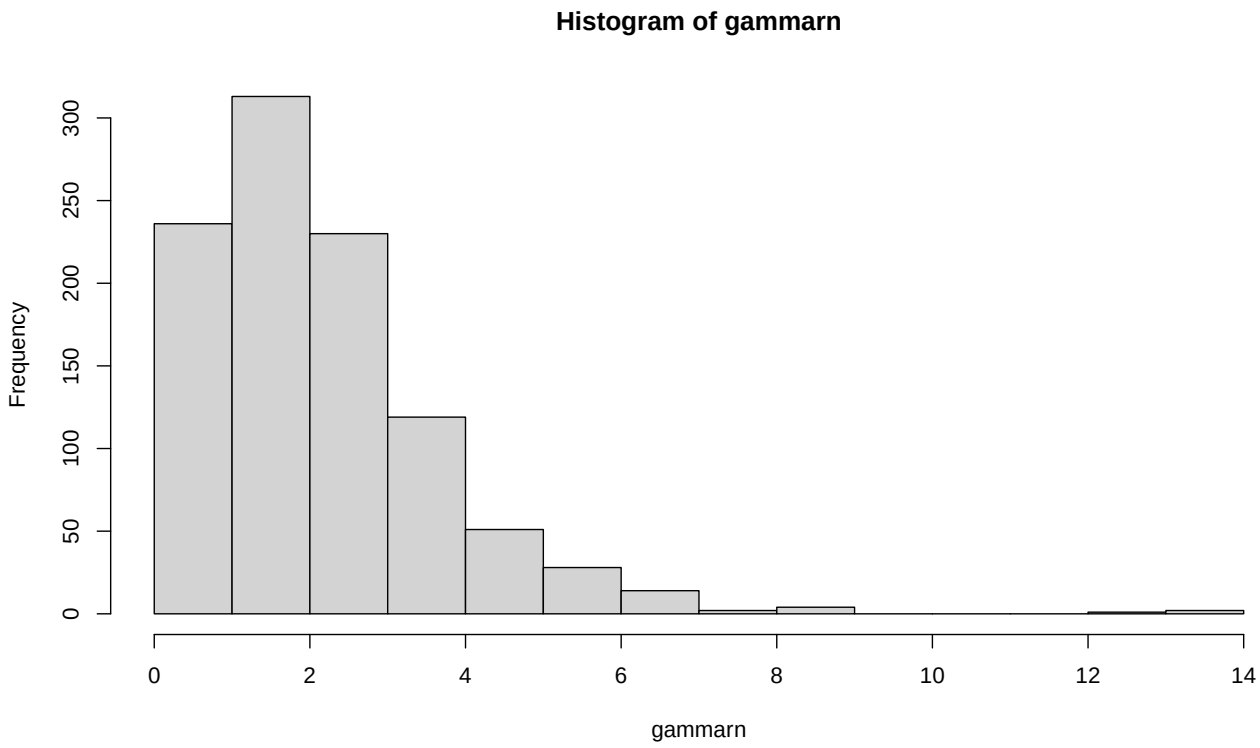
```r
#sampling from Gamma distribution with rejection sampling
sample_gamma_rej <- function(n,alpha)
{  sample_gamma <- rep(0,n)
   no.draw <- 0
   for(i in 1:n)
   {   rejected <- TRUE

       while(rejected)
       {  sample_gamma[i] <- rcauchy(1) * sqrt(2*alpha-1) + (alpha -1)
          no.draw <- no.draw + 1
          U <- runif(1)
          rejected <- (log(U) > dgamma(sample_gamma[i],shape=alpha,log=TRUE) -
                               log_g_gamma(sample_gamma[i],alpha) )
       }
   }
   attr(sample_gamma, "accept.rate") <- n/no.draw
   sample_gamma
}
```

1.3 Test 1

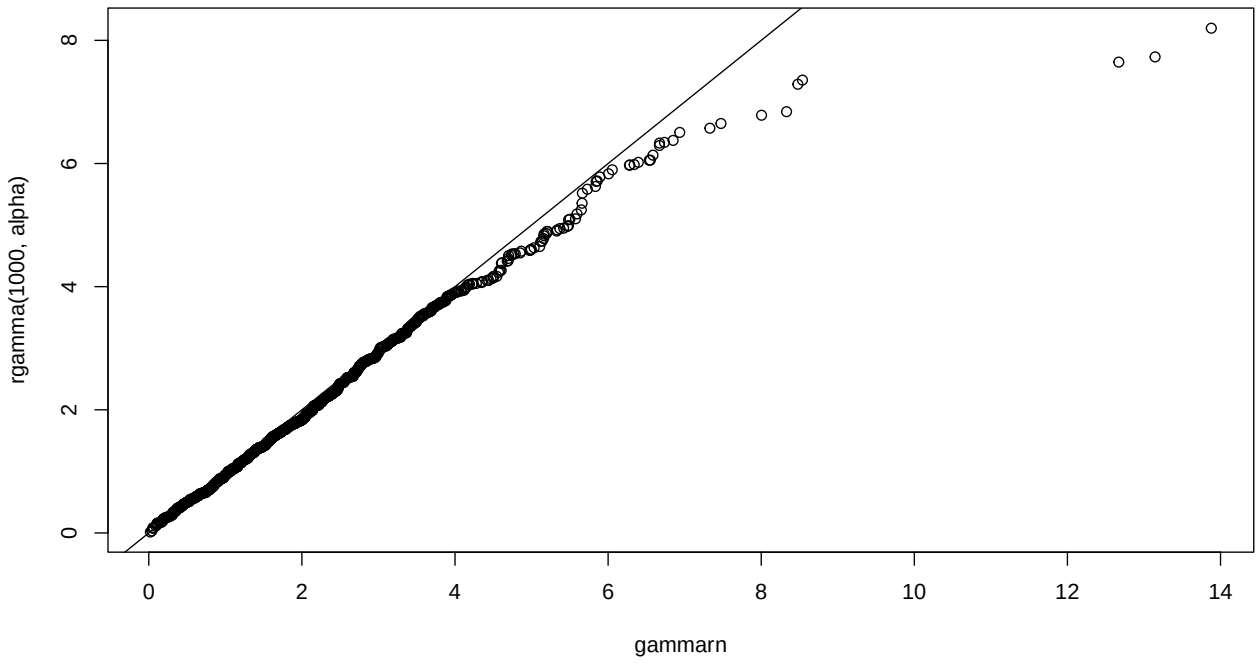
```r
alpha <- 2.1
gammarn <- sample_gamma_rej (1000,alpha); attr (gammarn, "accept.rate")
```

```
## [1] 0.4770992
```

```
hist (gammarn)
```




```
qqplot(gammarn, rgamma (1000, alpha))
abline (a = 0, b=1)
```

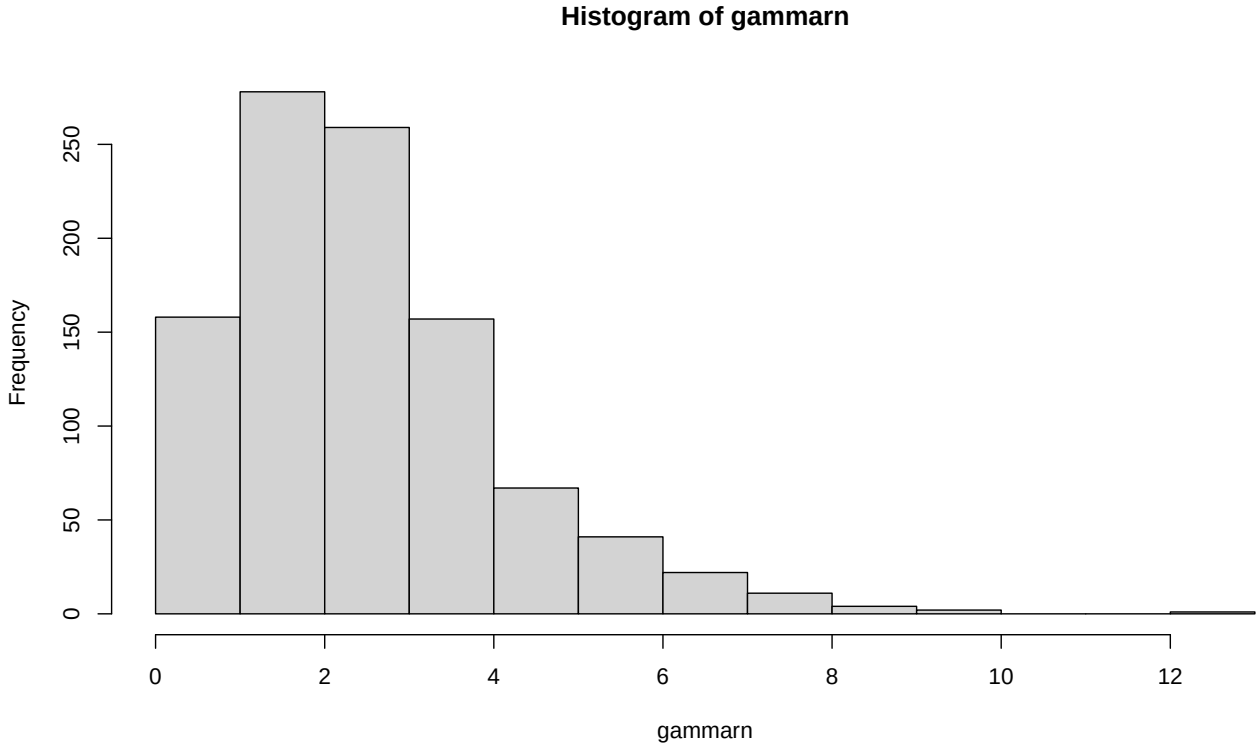


1.4 Test 2

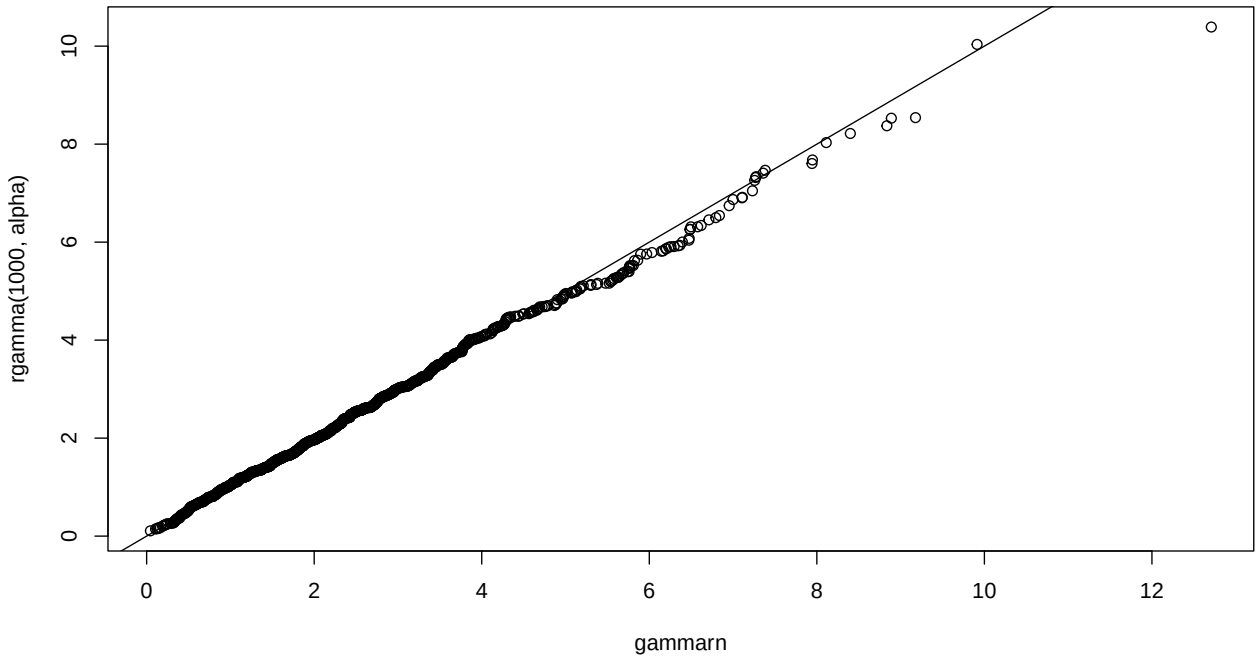
```
alpha <- 2.5
gammarn <- sample_gamma_rej (1000,alpha); attr (gammarn, "accept.rate")
```

```
## [1] 0.3783579
```

```
hist (gammarn)
```




```
qqplot(gammarn, rgamma (1000, alpha))
abline (a = 0, b=1)
```

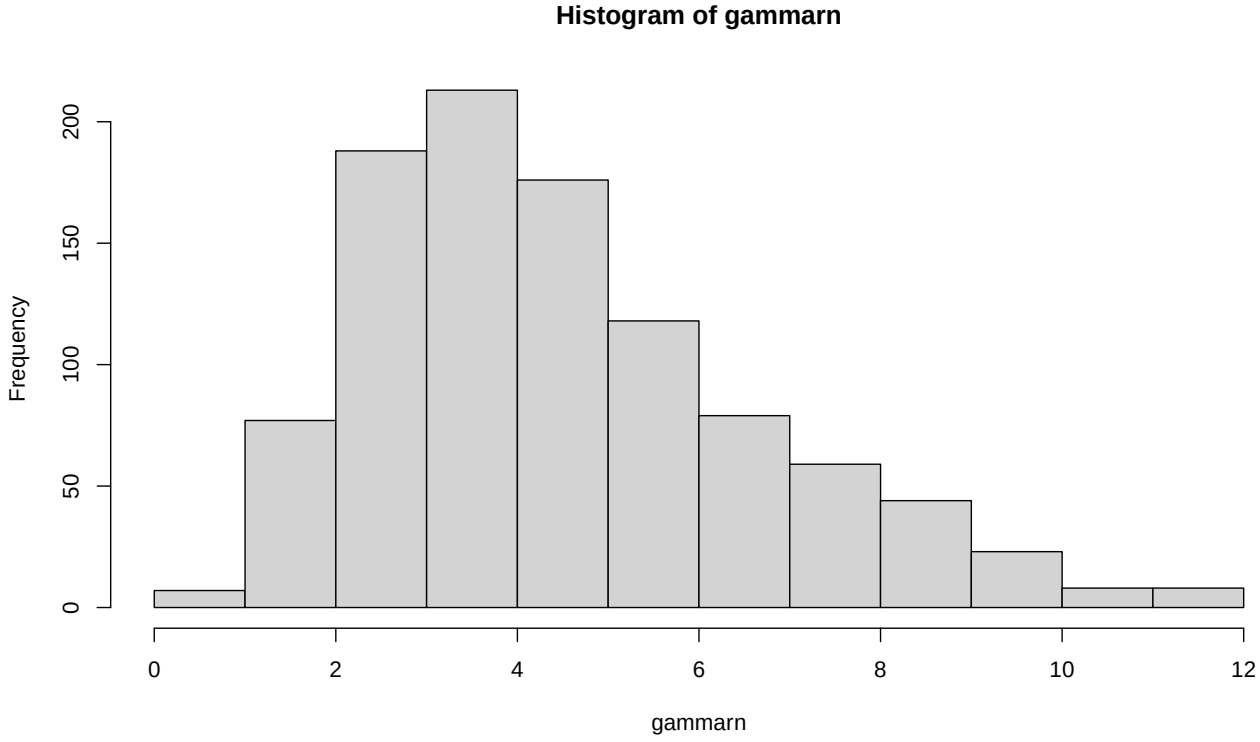


1.5 Test 3

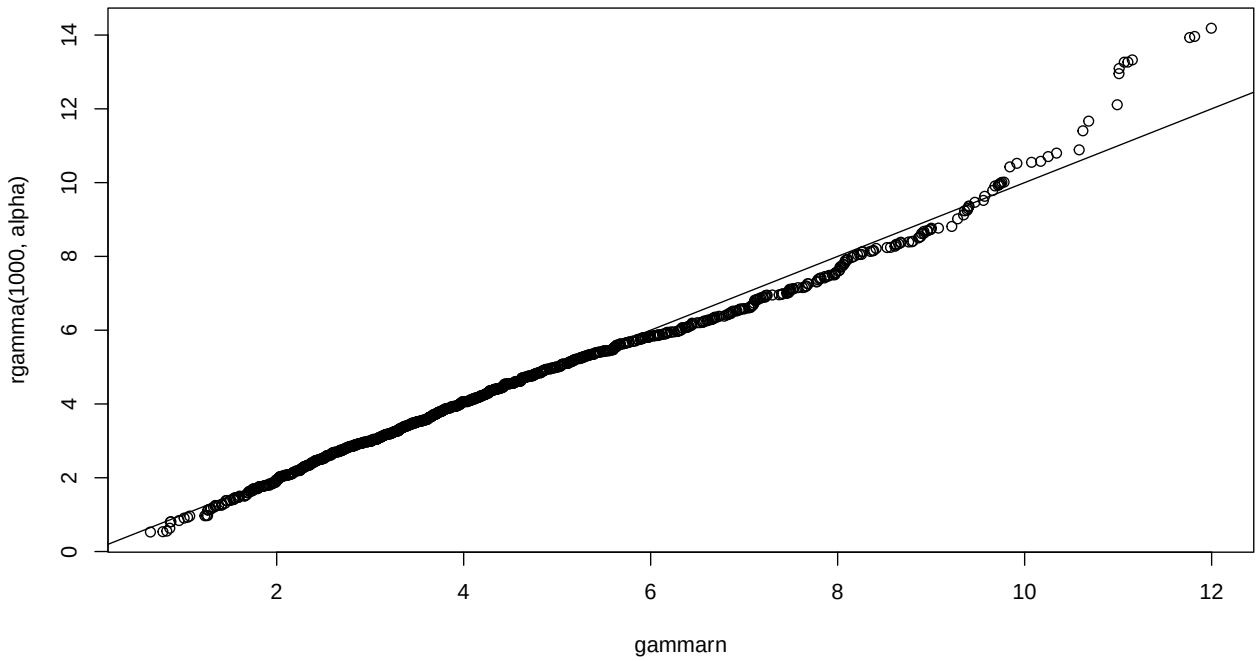
```
alpha <- 4.5
gammarn <- sample_gamma_rej (1000,alpha); attr (gammarn, "accept.rate")
```

```
## [1] 0.04639295
```

```
hist (gammarn)
```



```
qqplot(gammarn, rgamma (1000, alpha))
abline (a = 0, b=1)
```



We see that when alpha is larger than 2, the overall acceptance rate is very low

If you requires highly efficient gamma random numbers generators, read a paper in Computational Statistics and Data Analysis (2007) (http://www.sciencedirect.com/science/article/pii/S0167947306003616)

2 Adaptive Rejection Sampling for Truncated Normal

```
library (ars)

## a direct rejection sampling for truncated normal
sample_tnorm_drs <- function (n, lb = -Inf, ub = Inf)
{
    x <- rep (0, n)
    for (i in 1:n)
    {
        rej <- TRUE
        while (rej)
        {
            x[i] <- rnorm (1)
            if (x[i] >= lb & x[i] <= ub) rej <- FALSE
        }
    }
    x
}

## sample from truncated normal using ars package
sample_tnorm_ars <- function (n, lb, ub)
{
    logf <- function (x) dnorm (x, log = TRUE) ## define log density
```

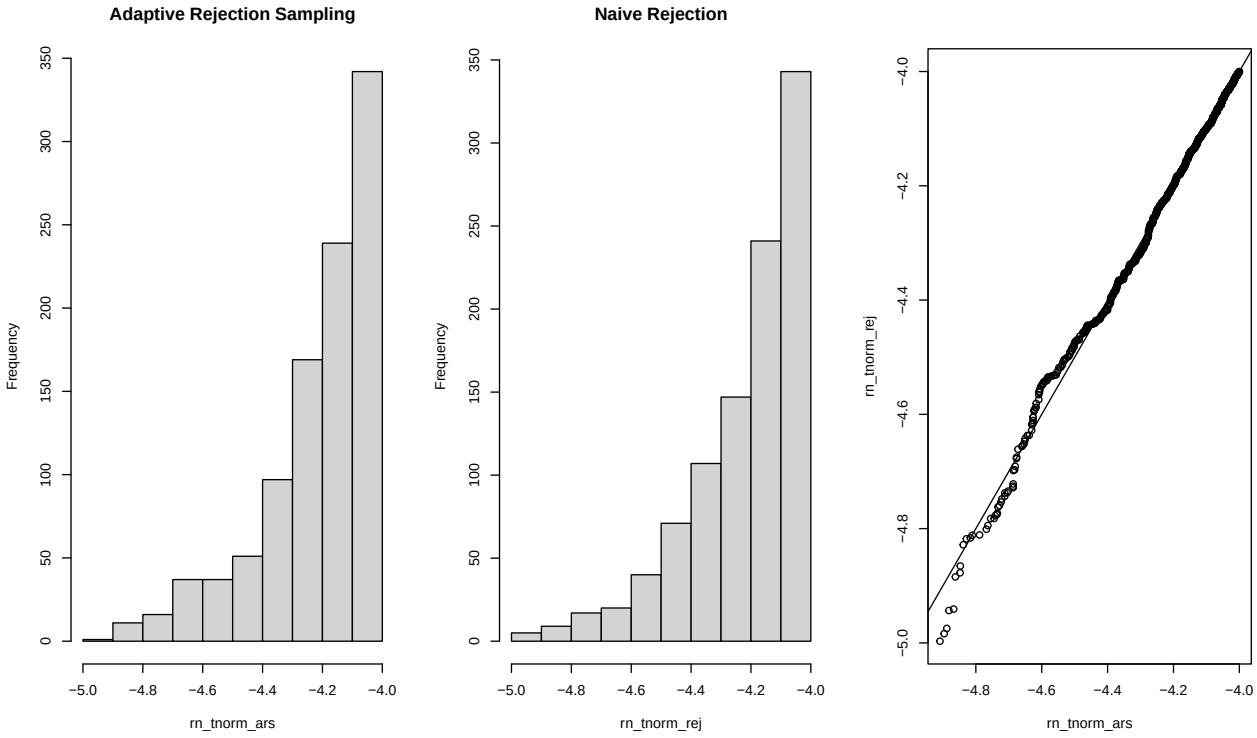
```
    fprima <- function (x) -x ## define derivative of log density

    ars (n, f = logf, fprima = fprima,
         x = c(lb, (lb + ub )/2, ub), # starting points
         lb = TRUE, ub = TRUE, xlb = lb, xub = ub) # boundary of log density
}
n <- 1000
system.time (
    rn_tnorm_ars <- sample_tnorm_ars (n, -5, -4)
)
```

```
##    user  system elapsed
##   0.010   0.000   0.009
```

```
system.time(
 rn_tnorm_rej <- sample_tnorm_drs (n, -5, -4)
)
```

```
##    user  system elapsed
##  56.407   6.447  65.318
```

```
par (mfrow = c(1,3))
hist (rn_tnorm_ars, main = "Adaptive Rejection Sampling")
hist (rn_tnorm_rej, main = "Naive Rejection")
qqplot (rn_tnorm_ars, rn_tnorm_rej)
abline (a = 0, b = 1)
```




```
# Draw truncated normal sample on the far tail
n <- 1000
system.time (
    rn_tnorm_ars2 <- sample_tnorm_ars (n, -50, -40)
)
```

```
##    user  system elapsed
##   0.016   0.000   0.027
```

```
system.time (
    rn_tnorm_ars3 <- sample_tnorm_ars (n, 100, 110)
)
```

```
##    user  system elapsed
##   0.008   0.001   0.008
```

```
par (mfrow = c(1,2))
hist (rn_tnorm_ars2, main = "Adaptive Rejection Sampling")
hist (rn_tnorm_ars3, main = "Adaptive Rejection Sampling")
```