Assignment4.revision.R

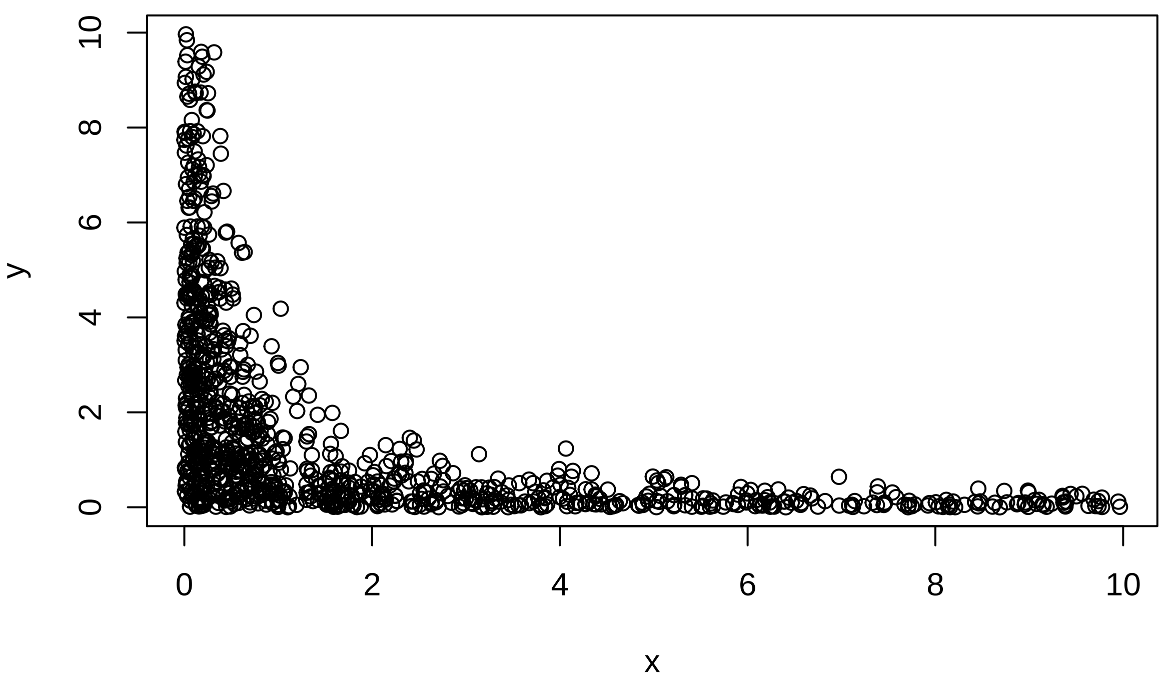
EvergreenFu

Sat Jun 25 19:00:57 2016

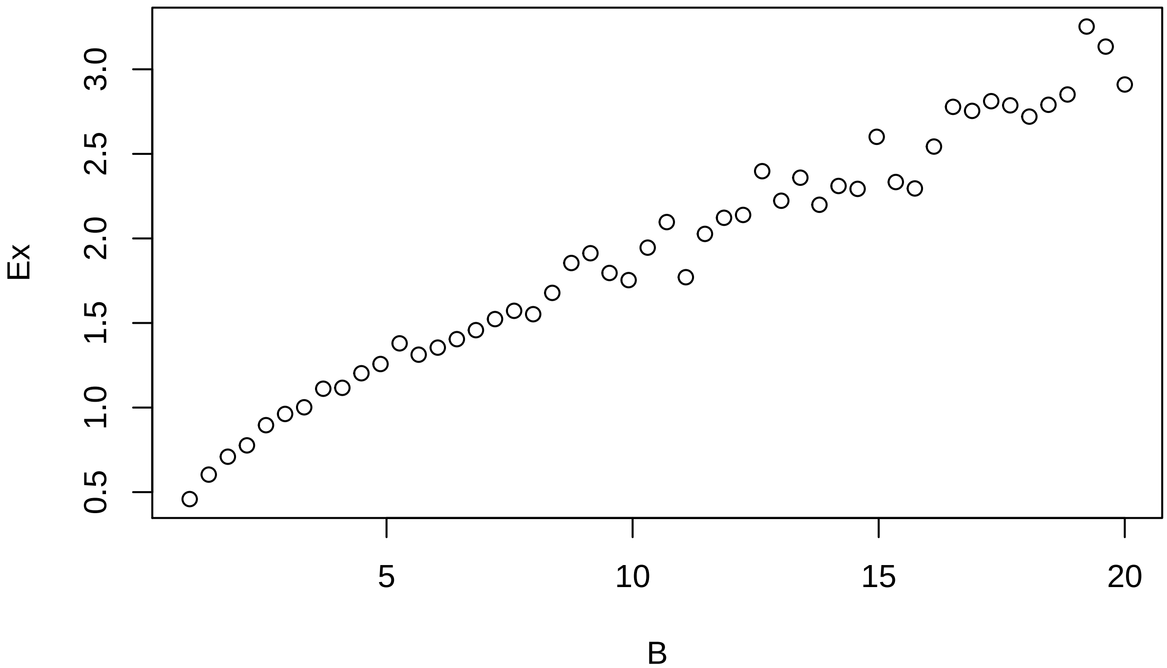
```r
####################################################
## dependencies
library(scatterplot3d)
```

```
## Warning: package 'scatterplot3d' was built under R version 3.2.5
```

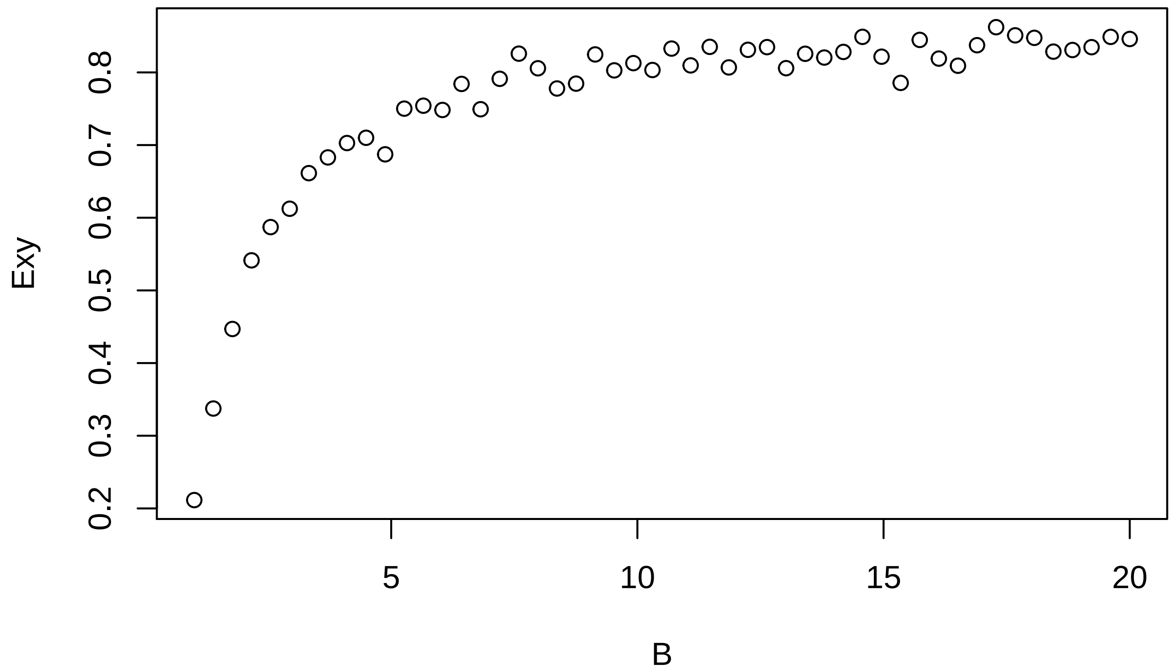
```r
source("/Users/EvergreenFu/Desktop/STAT/hw/3/Assignment3functions.R")
####################################################
## 1.Gibbs Sampling for biexponential distribution
gibbs_biexp <- function(niter,x0,y0,M){
  x = rep(x0,niter);y = rep(y0,niter)
  rexpM <- function(lambda,M){
    repeat{
      x = rexp(1,lambda)
      if(x < M) return(x)
    }
  }
  for(i in 2:niter){
    x[i] = rexpM(y[i-1],M)
    y[i] = rexpM(x[i],M)
  }
  result = cbind.data.frame(x,y)
}
#  assume B = 10, simulate x,y
plot(gibbs_biexp(niter = 1000,1,1,10))
```



```r
#  Ex, Exy as a function of B
B = seq(1,20,length = 50);
f11 <- function(M)
  mean(gibbs_biexp(niter = 3000,1,1,M)[,1])
f12 <- function(M) {
  A = gibbs_biexp(niter = 3000,1,1,M)
  mean(A[,1]*A[,2])
}
Ex = sapply(B, f11)
Exy = sapply(B, f12)
plot(B,Ex)
```




```r
plot(B,Exy)
```

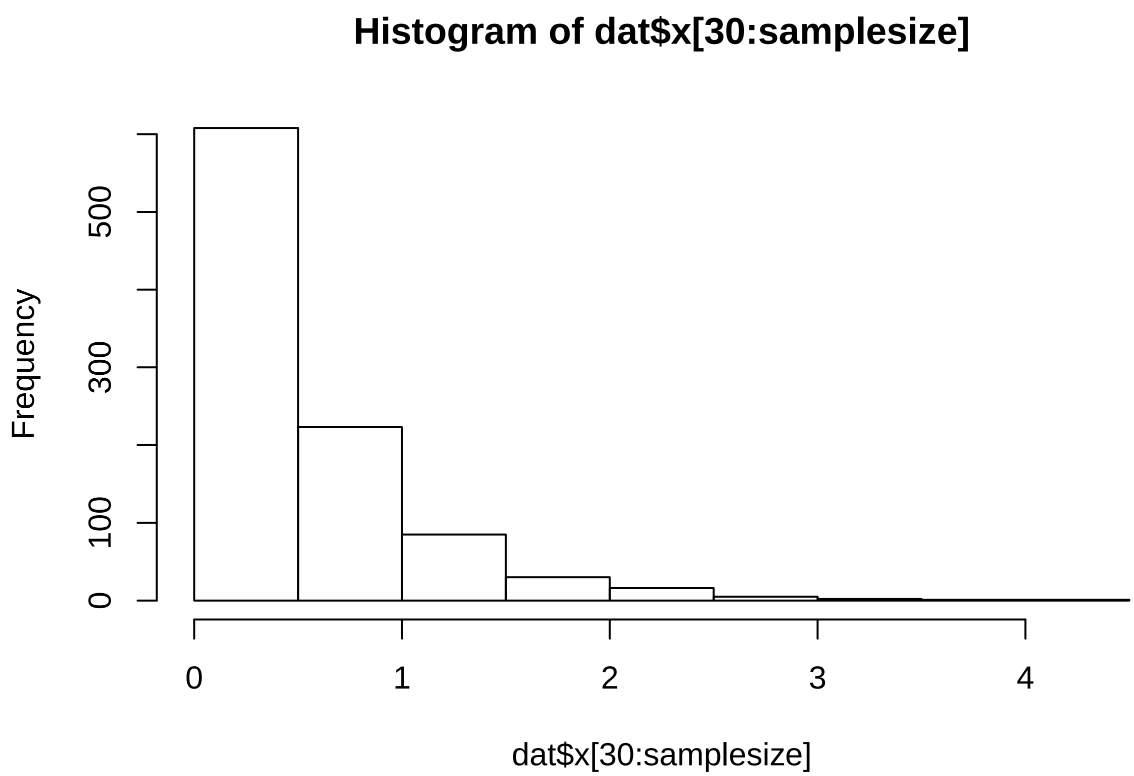



```
#####################################################
## 2.conditional expectation via simulation
#  x_1,2,3 ~ exp dist with mean 1 (satisfying iid).
#  assume y = x_1 + 2*x_2 + 3*x_3, solve E1 = E(y|y>15); E2 = E(y|y<1)
f2 <- function(M,greater = TRUE){
  c = c(1,2,3);
  repeat{
    x = sum(rexp(3,1)*c)
    if(greater){if(x > M) return(x)}
    else if(x < M) return(x)
  }
}
E <- function(niter, bound, greater)
  mean(sapply(rep(bound,niter),f2,greater = greater))
E1 = E(10000,15,TRUE);
E2 = E(10000,1,FALSE);

#####################################################
## 3.
## triple Gibbs sampling algorithm
#  input:   niter
#           FUN.x         f(x|y,z) (similarly, FUN.y, FUN.z)
#           x0, y0, z0    initial value
#           isdiscrete_x  TRUE if x is discrete
#           intervalx     interval type of sample
#           boundx        if discrete, variable interval is specified by this argument
#  output:  P(x|y,z)
triple_simulation <- function(niter, FUN.x, FUN.y = FUN.x, FUN.z = FUN.x, init = c(0.5,0.5,0.5),
                              isdiscrete_x = F, isdiscrete_y = isdiscrete_x, isdiscrete_z = isdiscrete_x
                              boundx = NULL, boundy = boundx, boundz = boundx,
                              intervalx = NULL, intervaly = intervalx, intervalz = intervalx
                              ){
```

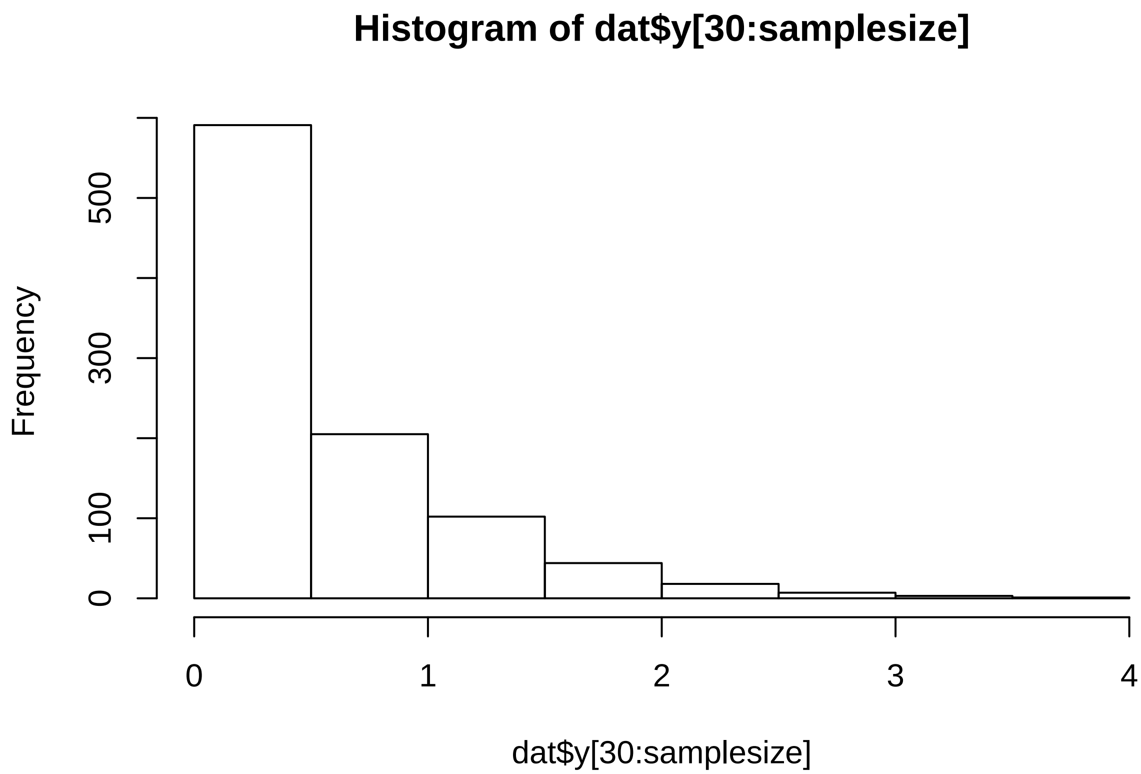
```
  x = rep(init[1],niter);y = rep(init[2],niter);z = rep(init[3],niter)
  # to make things neater:
  sample <- function(FUN,y,z,interval,isdiscrete = FALSE, bound = NULL){
    if(!isdiscrete)
      rcont(1,function(x){FUN(x,y,z)},interval = interval, method = "acceptance")
    else
      rdisc(1,function(x){FUN(x,y,z)},bound = bound, interval = interval, c = c,method = "acceptance")
  }
  # problem here -- cannot preprocess...
  # preprocess c to accelerate sampling process
  # optimization <- function(FUN,y,z,isdiscrete,k = NULL) {
  #   result = {
  #     if(!isdiscrete)
  #       optimize(f = {if(interval == "01") function(x) FUN(x,y,z)
  #                      else if(interval == "real_positive") function(x) FUN(x,y,z)/dexp(x)
  #                      else if(interval == "real") function(x) FUN(x,y,z)/dnorm(x)
  #                     },
  #                maximum = T,
  #                interval = {if(interval == "01")c(0,1)
  #                            else if(interval == "real_positive")c(0,100)
  #                            else if(interval == "real_positive")c(-100,100)
  #                           }
  #                )$objective
  #     else{
  #       if (interval == "finite"){max(sapply(0:k,FUN))}
  #       else if (interval == "positive"){max(sapply(k:100,FUN)/dexp(1:100))}
  #       else if (interval == "integer"){max(sapply(-100:1000,FUN)/dnorm(-100:100))}
  #     }
  #   }
  # }
  # cx = optimization(FUN.x,y,z,isdiscrete_x,boundx)
  # cy = optimization(FUN.y,z,x,isdiscrete_y,boundy)
  # cz = optimization(FUN.z,x,y,isdiscrete_z,boundz)
  #iteration:
  for(i in 2:niter){
    #here we use a function in assignment 3 to sample using acceptance-rejection method
    x[i] = sample(FUN.x,y[i-1],z[i-1],intervalx, isdiscrete_x, bound = z[i-1])
    y[i] = sample(FUN.y,z[i-1],x[i],intervaly, isdiscrete_y, bound = boundy)
    z[i] = sample(FUN.z,x[i],y[i],intervalz, isdiscrete_z, bound = boundz)
    }
  cbind.data.frame(x = x, y = y, z = z)
}

## apply algorithm, in this case x,y,z are symmetric
#  when f(x,y,z) = exp(-x-y-z-x*y-y*z-x*z), f(x|y,z) is
FUN <- function(x,y,z){
  exp(-x-y-z-x*y-y*z-x*z)*(y+z+1)/(1-exp(-y*z-y-z))
}
#note: FUN is complicated, and iteration in R is slow, so we use small sample size to speed-up
samplesize = 1000
dat = triple_simulation(samplesize,FUN,intervalx = "real_positive")
#template of sample
dat[1:10,]
```

```
##              x          y         z
## 1  0.50000000 0.50000000 0.5000000
## 2  0.11541316 0.94231377 0.1603458
## 3  0.66622060 1.35702006 0.5220106
## 4  0.17711491 0.55365934 1.8083679
## 5  1.43065650 0.19735587 0.6672028
## 6  0.38461662 0.38151946 0.6741417
## 7  0.26392056 1.07666201 0.7995798
## 8  0.23974651 0.09905332 1.2949737
## 9  0.09242397 0.25800592 1.5874469
## 10 0.47635450 0.01107742 0.5706523
```

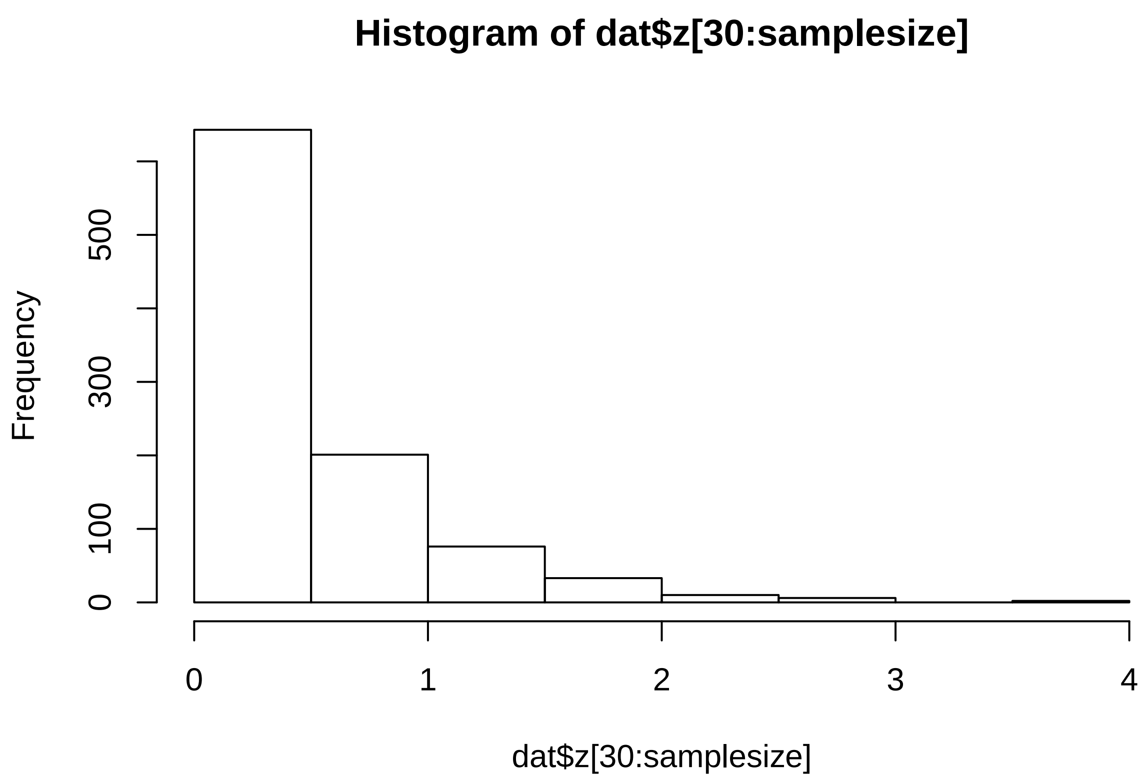
```r
hist(dat$x[30:samplesize])
```



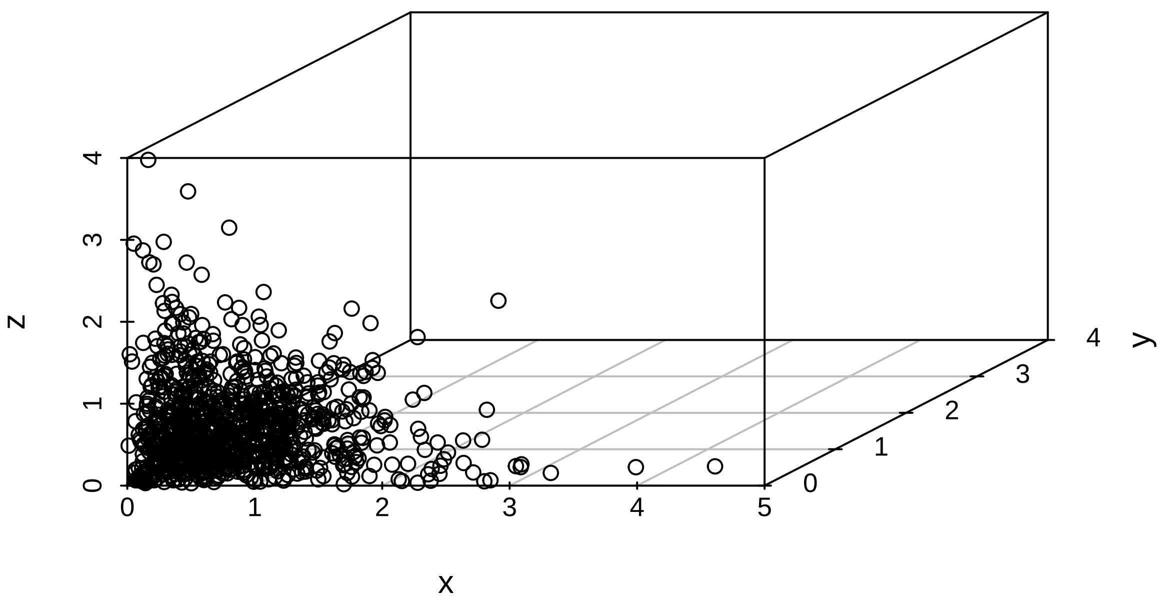

```r
hist(dat$y[30:samplesize])
```




```r
hist(dat$z[30:samplesize])
```



```r
scatterplot3d(dat)
```




```r
#Exyz
mean(apply(dat,1,function(x)exp(sum(log(x)))))
```

```
## [1] 0.08568615
```

```r
####################################################
## 4.f(x,y,z) amalgam of discrete-continuous distribution
#  re-write algorithm slightly since interval is a variable (marked as #*)
triple_simulation_s <- function(niter, FUN.x, FUN.y = FUN.x, FUN.z = FUN.x, init = c(2,2,2),
                                isdiscrete_x = F, isdiscrete_y = isdiscrete_x, isdiscrete_z = isdiscret
                                boundx = NULL, boundy = boundx, boundz = boundx,
                                intervalx = NULL, intervaly = intervalx, intervalz = intervalx
                                )
{
  x = rep(init[1],niter);y = rep(init[2],niter);z = rep(init[3],niter)
  # to make things neater:
  sample <- function(FUN,y,z,interval, isdiscrete = FALSE, bound = NULL){
    if(!isdiscrete)
      rcont(1,function(x){FUN(x,y,z)}, interval = interval, method = "acceptance")
    else
      rdisc(1,function(x){FUN(x,y,z)}, bound = bound, interval = interval, method = "acceptance")
  }
  #iteration:
  for(i in 2:niter){
    #here we use a function in assignment 3 to sample using acceptance-rejection method
    x[i] = sample(FUN.x,y[i-1],z[i-1],intervalx, isdiscrete_x, bound = z[i-1]) #*
    y[i] = sample(FUN.y,z[i-1],x[i],intervaly, isdiscrete_y, bound = boundy)
    z[i] = sample(FUN.z,x[i],y[i],intervalz, isdiscrete_z, bound = x[i]) #*
  }
  cbind.data.frame(x = x, y = y, z = z)
}
```

```r
## apply algorithm
FUN <- function(x,y,n, alpha = 2,beta = 3, lambda = 4)
  choose(n,x)*y^(x+alpha-1)*(1-y)^(n-x+beta-1)*exp(-lambda)*lambda^n/factorial(n)
  ##suspitious typo here at (1-y)^(n*x...), should be (1-y)^(n-x...)
FUN.x <- function(x,y,n)
  choose(n,x)*y^(x)*(1-y)^(n-x)
FUN.y <- function(y,n,x,alpha = 2,beta = 3)
  y^(x+alpha-1)*(1-y)^(n-x+beta-1)/beta(x+alpha,n-x+beta)
FUN.z <- function(n,x,y,lambda = 4)
  1/factorial(n-x)*(lambda*(1-y))^(n-x)*exp(y-lambda)
source("/Users/EvergreenFu/Desktop/STAT/hw/3/Assignment3functions.R")
dat = triple_simulation_s(500,FUN.x,FUN.y,FUN.z,init = c(2,0.9,5),
                          isdiscrete_x = T,isdiscrete_y = F,isdiscrete_z = T,
                          intervalx = "finite", intervaly = "01", intervalz = "positive"
)
#template of sample
dat[1:10,]
```

```
##    x          y  z
## 1  2 0.90000000  5
## 2  5 0.48176123  6
## 3  3 0.27897849 10
## 4  1 0.22288819 12
## 5  3 0.29475288  8
## 6  2 0.53048378  5
## 7  2 0.10671014 11
## 8  0 0.29622861  5
## 9  0 0.09367639  7
## 10 3 0.53868560  9
```

```r
## Ex, Ey, En
mean(dat$x)
```

```
## [1] 3.72
```

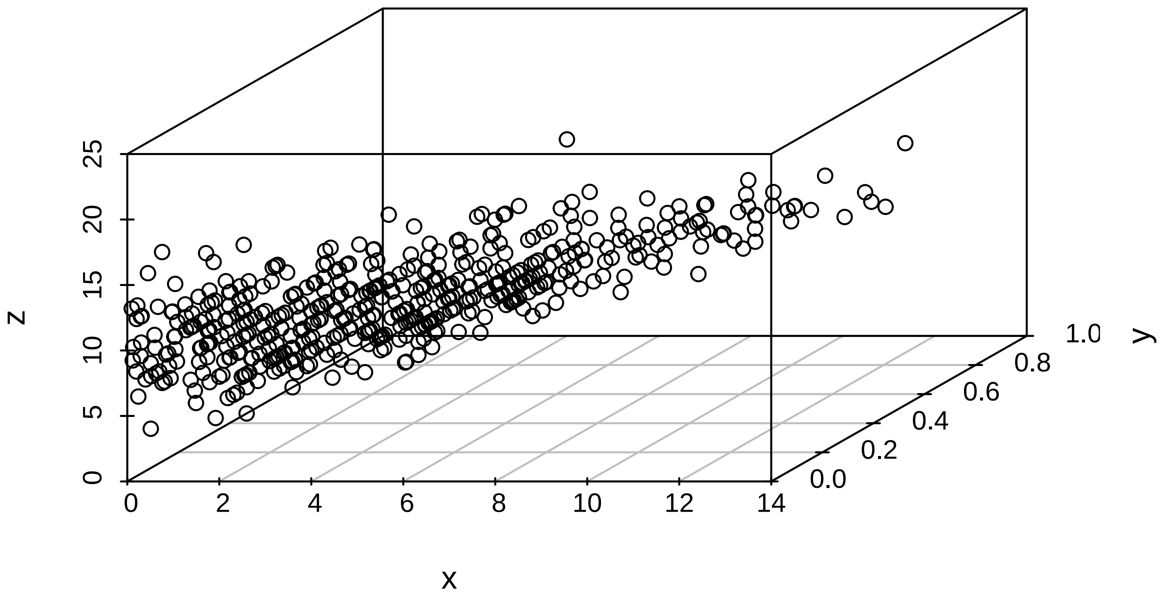
```r
mean(dat$y)
```

```
## [1] 0.396873
```

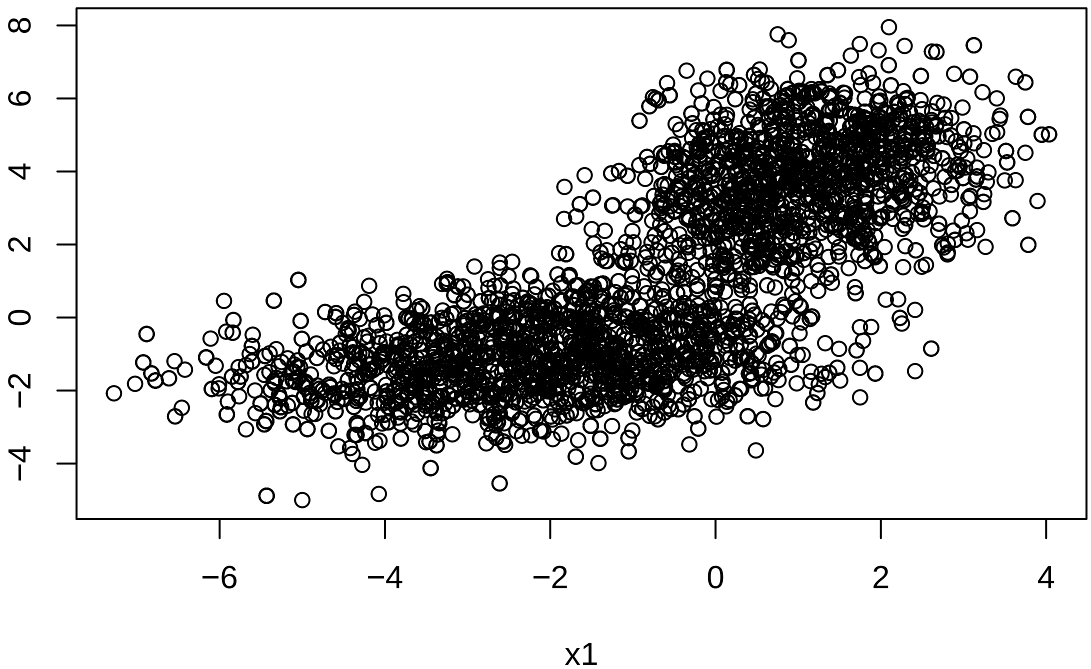
```r
mean(dat$z)
```

```
## [1] 9.55
```

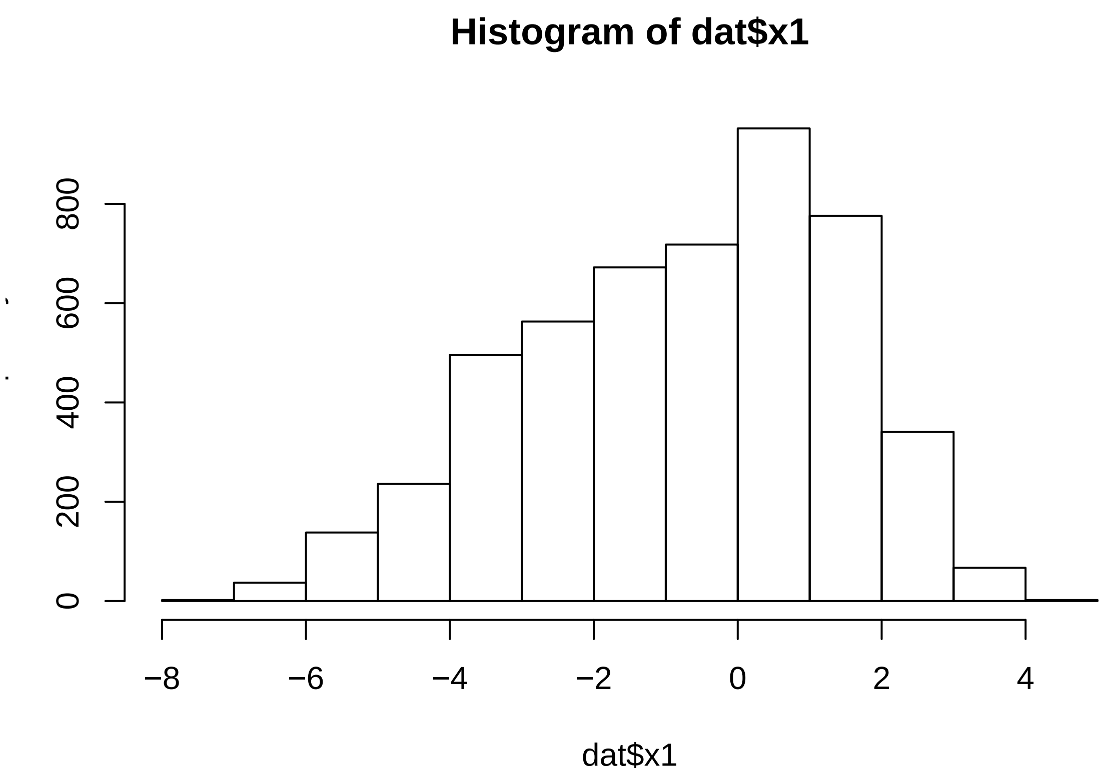
```r
scatterplot3d(dat)
```



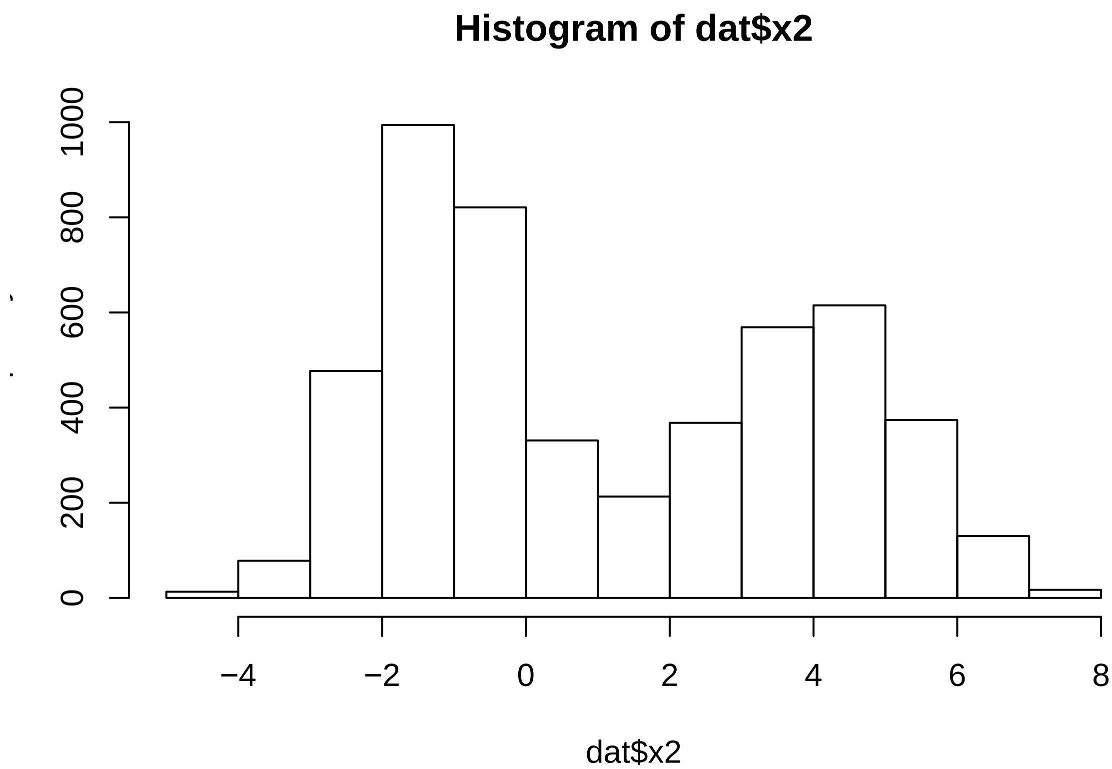

```
## note: experiment shows that this distribution does not depend on initial value

# ####################################################
## 5.Metropolis-Hasting Algorithm for 2-dim-GMM
#  algorithm: gmm
#  method:    Metropolis-Hasting
#  datatype:  x       data.frame(names = c("x1","x2"))
#             mu      list(names = c("a","b"))
#             sigma   list(names = c("a","b"))
#             sigma0  matrix
library(mvtnorm)
gmm_MH <- function(n, mu ,sigma, sigma0 = diag(2,2)){
  x = data.frame(x1=rep(-5,n),x2=rep(-5,n))
  for(i in 2:n){
    x[i,] = x[i-1,] + rmvnorm(1,c(0,0),sigma0)
    if(runif(1) > (dmvnorm(x[i,],mu[[1]],sigma[[1]])+dmvnorm(x[i,],mu[[2]],sigma[[2]]))/(dmvnorm(x[i-1,
      x[i,] = x[i-1,]
    }
  }
  names(x)=c("x1",'x2')
  x
}
mu = list(a=c(1,4),b=c(-2,-1))
sigma = list(a=matrix(c(1,0.3,0.3,2),2),b=matrix(c(3,0.4,0.4,1),2))
dat = gmm_MH(5000,mu,sigma)
plot(dat)
```




```r
hist(dat$x1)
```




```r
hist(dat$x2)
```


Histogram of dat$x2
0
200
400
600
800
1000
−4
−2
0
2
4
6
8
dat$x2