به نام خدا

دانشگاه صنعتی شریف - دانشکده مهندسی کامپیوتر

آمار و احتمال مهندسی

پاییز 1401

تمرین چهارم عملی

طراح: محمدرضا شاپوری - هیربد بهنام

امیررضا آذری 99101087

موعد تحویل: 5 دی

همفکری در تمامی تمرین‌های درس توصیه می‌شود. در عین حال از شما خواسته می‌شود تا تمام پیاده‌سازی را به تنهایی و بدون مشاهده کد دیگران انجام دهید.

لطفا در فایل ارسالی تمام بلوک‌های کد اجرا شده و شامل نمودارها و خروجی‌های لازم باشند.

سوال اول

در این تمرین قصد داریم ، قضیه حد مرکزی را بر روی مجموعه داده داده شده در R گام به گام شبیه‌سازی کنیم.

ابتدا فایل CSV را در R وارد کنید

In [177]:
```
#Step 1 - Importing Data
#______________________________________________________

#Importing the csv data
data<-read.csv("Clt-data.csv")

#Step 2 - Validate data for correctness
#_____________________________________________________

#Count of Rows and columns
dim(data)

#View top 10 rows of the dataset
head(data,10)
```

9000 · 1

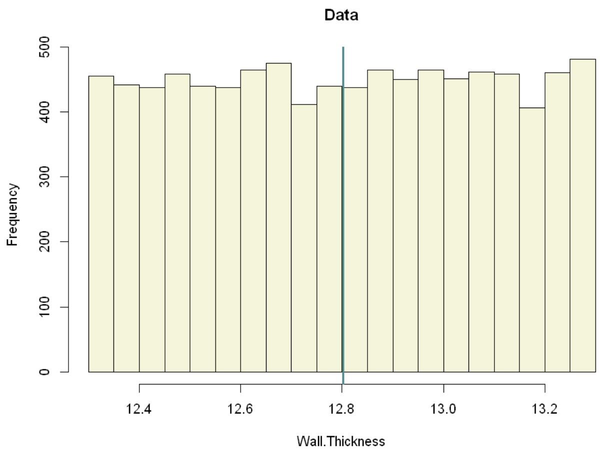
A data.frame: 10 × 1

	Wall.Thickness
	<dbl>
1	12.35487
2	12.61742
3	12.36972
4	13.22335
5	13.15919
6	12.67549
7	12.36131
8	12.44468
9	12.62977
10	12.90381

حال میانگین جمعیت را محاسبه کنید و داده ها را در نمودار رسم کنید:

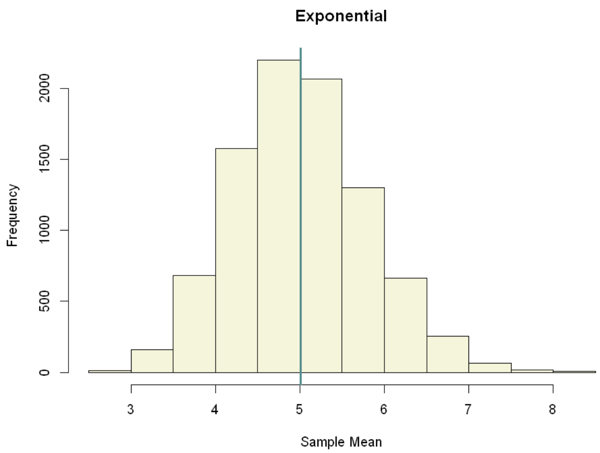
In [178]:
```
#Calculate the population mean
mean(data$Wall.Thickness)

#Plot all the observations in the data
hist(data$Wall.Thickness, xlab='Wall.Thickness', main="Data", col='beige')
abline(v=mean(data$Wall.Thickness), lwd=3, col='darkslategray4')
```

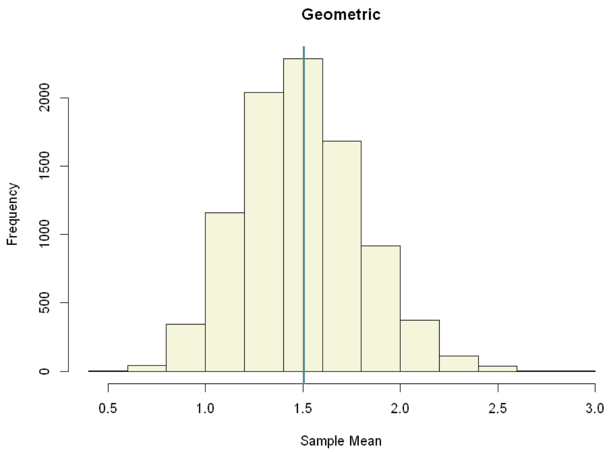
12.8020492455356

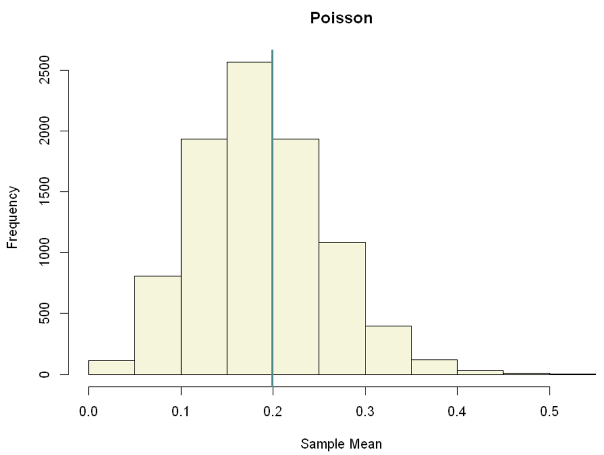


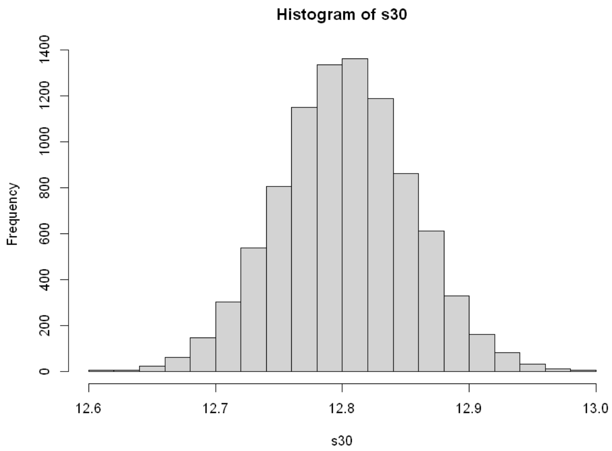
حال سمپل هایی (حداقل ۳ عدد) با سایز بزرگ تر (حداقل ۳۰) گرفته و میانگین آن را برای تعداد دفعات بالا(n=9000) محاسبه کنید و شکل نمودار را با نمودار اولیه مقایسه کنید

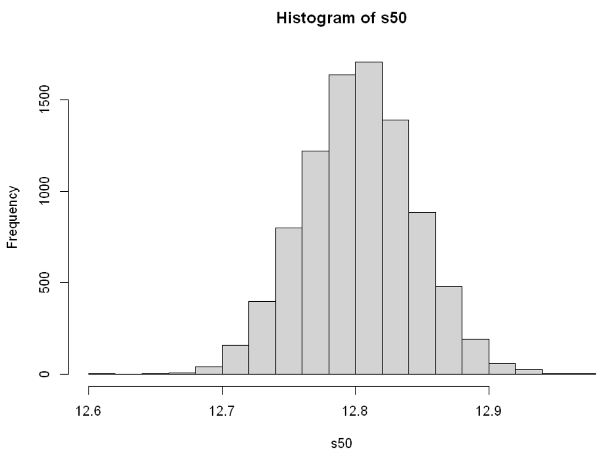
In [179]:
```
clt1 <- NULL
clt2 <- NULL
clt3 <- NULL
n <- 40
lambda <- 0.2
for (i in 1:9000) {
  clt1 <- c(clt1, mean(rexp(n, lambda)))
  clt2 <- c(clt2, mean(rgeom(n, 0.4)))
  clt3 <- c(clt3, mean(rpois(n, lambda)))
}

hist(clt1, xlab='Sample Mean', main="Exponential", col='beige')
abline(v=mean(clt1), lwd=3, col='darkslategray4')

hist(clt2, xlab='Sample Mean', main="Geometric", col='beige')
abline(v=mean(clt2), lwd=3, col='darkslategray4')

hist(clt3, xlab='Sample Mean', main="Poisson", col='beige')
abline(v=mean(clt3), lwd=3, col='darkslategray4')


# Or:

s30 <- c()
s50 <- c()
s500 <- c()
n = 9000
for ( i in 1:n){
  s30[i] = mean(sample(data$Wall.Thickness,30, replace = TRUE))
  s50[i] = mean(sample(data$Wall.Thickness,50, replace = TRUE))
  s500[i] = mean(sample(data$Wall.Thickness,500, replace = TRUE))
  }
hist(s30)
hist(s50)
hist(s500)
```

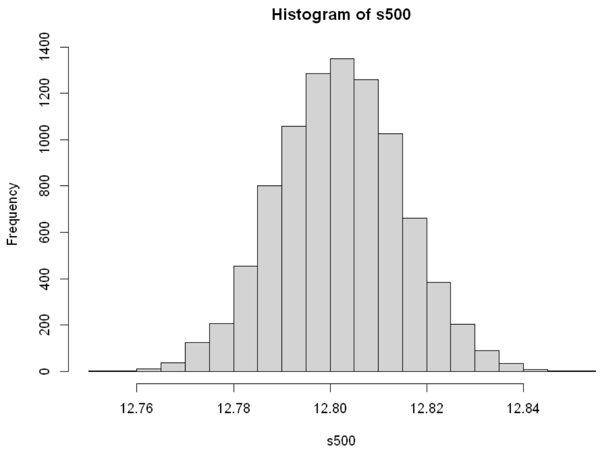



Geometric
Frequency
0
500
1000
1500
2000
0.5
1.0
1.5
2.0
2.5
3.0
Sample Mean


Poisson
Frequency
0
500
1000
1500
2000
2500
0.0
0.1
0.2
0.3
0.4
0.5
Sample Mean


Histogram of s30
Frequency
0
200
400
600
800
1000
1200
1400
12.6
12.7
12.8
12.9
13.0
s30


Histogram of s50
Frequency
0
500
1000
1500
12.6
12.7
12.8
12.9
s50



سوال دوم

در این سوال می‌خواهیم که تنها به کمک یک عدد رندوم ساز یکنوا(uniform) صحیح، اعداد رندومی که از یک توزیع ریاضی پیروی می‌کنند را بسازیم.
یک عدد رندوم ساز یکنوا صحیح، تابعی است که خروجی آن یک عدد صحیح در بازه‌ی $[0, n)$ است که احتمال انتخاب شدن هر عدد در آن برابر است.
دقت کنید که در این سوال حق استفاده از توابعی همچون qnorm/rexp/runif و غیره را ندارید!
برای شروع، این موضوع را در نظر داشته باشید که الگوریتم‌های تولید اعداد تصادفی [(PRNG) (https://en.wikipedia.org/wiki/Pseudorandom_number_generator)](https://en.wikipedia.org/wiki/Pseudorandom_number_generator) که در زبان‌های برنامه نویسی استفاده می‌شوند صرفا بیت‌های رندوم تولید می‌کنند. این بدین معنا است که به عنوان مثال الگورتمی به عنوان خروجی 32بیت به ما می‌دهد که هر بیت به احتمال نیم برابر 1 است و به احتمال نیم برابر 0 است. حال دقت کنید که می‌توان این 32 بیت را به عنوان یک عدد صحیح در بازه‌ی $[0, 2^{32})$ در نظر گرفت. این موضوع نشان می‌دهد که پایه‌ای ترین ابزاری که برای ساخت اعداد تصادفی داریم عملا یک int می‌سازد که بین 0 و 2^{32} است و احتمال آمدن هر عدد برابر است (پس توزیع آن uniform است). به عنوان مثال Python و R از الگوریتم [Mersenne Twister (https://en.wikipedia.org/wiki/Mersenne_Twister)](https://en.wikipedia.org/wiki/Mersenne_Twister) برای ساخت اعداد تصادفی int استفاده می‌کند.
می‌دانیم که برای استفاده از inverse transform sampling - که در ساختن اعداد تصادفی با توزیع‌های خاص کاربرد دارد - باید در ابتدا متغیر تصادفی داشته باشیم که از توزیع یونیفرم بین 0 تا 1 پیروی کند. به عبارتی دیگر باید تابعی داشته باشیم که یک عدد رندوم با توزیع یکنوا بین 0 و 1 بسازد. روشی که امروزه برای ساخت اعداد اعشاری بین 0 و 1 با توزیع یکنوا استفاده می‌شود بدین صورت است که اگر به عنوان مثال تابعی داشته باشیم که یک عدد رندوم صحیح با توزیع یکنوا $[0, n)$ به ما بدهد، کافی است که آنرا صدا بزنیم و خروجی آنرا تقسیم بر n کنیم. بدین صورت یک عدد تصادفی اعشاری در بازه‌ی $[0, 1)$ داریم.
برای شروع این سوال، به کمک تابع sample در r، تابعی بنویسید که یک عدد تصادفی با توزیع یونیفرم در بازه‌ی $[0, 1)$ بدهد. پیشنهاد می‌کنم که بازه‌ی خروجی اعداد sample را $[0, 2^{23})$ قرار دهید. (چرایی این عدد را در درس ساختار و زبان کامپیوتر پیدا کنید!)

In [180]:
```
funct <- function(x) sample(0:2^23, 1) / 2^23
# funct()        example to show how this function works
# funct()
```

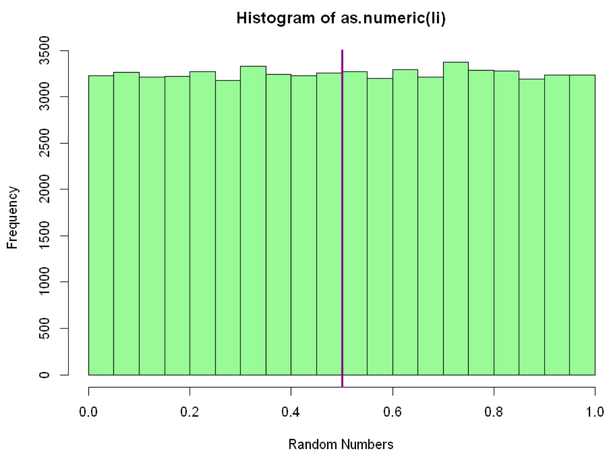
تابعی که نوشتید را به تعداد زیادی صدا بزنید و histogram اعداد برگردانده شده را رسم کنید و نشان دهید که اعداد برگردانده شده، از توزیع یونیفرم بین ۰ و ۱ پیروی می‌کنند.

In [181]:
```
li = list()
for (i in 1:65000) {
  li <- c(li, funct())
}
hist(as.numeric(li), xlab = "Random Numbers", col = 'palegreen')
abline(v=mean(as.numeric(li)), lwd=3, col='darkmagenta')
mean(as.numeric(li))
sd(as.numeric(li))
var(as.numeric(li))     # ~ Uniform(0, 1) , μ = 0.5, var = 0.08
```

0.500574492423351

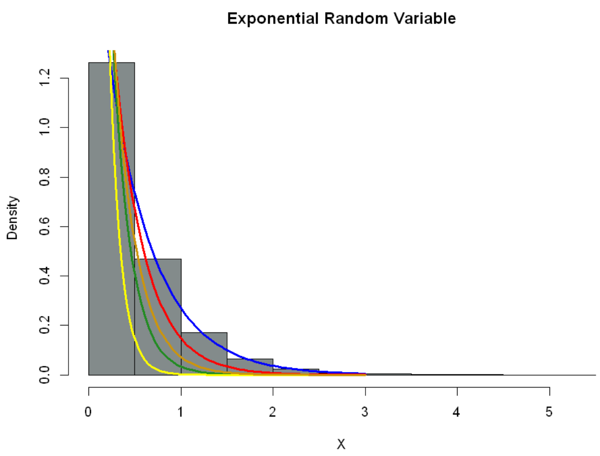
0.28828755124637

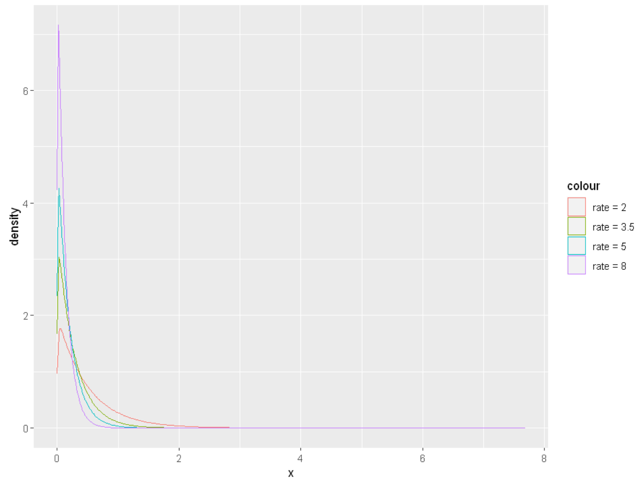
0.0831097122036282



حال می‌خواهیم که تابعی پیاده سازی کنیم که یک عدد تصادفی با توزیع نمایی را تولید کند. این کار را به کمک inverse transform sampling انجام دهید.
سپس با رسم نمودار به ازای لانداهای مختلف و دلخواه، عملکرد تابع خودتان را با تابع rexp مقایسه کنید.
پیشنهاد: می‌توانید از geom_density استفاده کنید.

In [182]:
```
# inverse transform sampling
funct2 <- function(y) -log(1-as.numeric(y)) / 2
X <- funct2(li)
# plot
hist(X, freq=F, xlab='X', main='Exponential Random Variable', col='azure4')
curve(dexp(x, rate=2) , 0, 3, lwd=3, xlab = "", ylab = "", add = T, col='blue')
curve(dexp(x, rate=3) , 0, 3, lwd=3, xlab = "", ylab = "", add = T, col='red')
curve(dexp(x, rate=5) , 0, 3, lwd=3, xlab = "", ylab = "", add = T, col='forestgreen')
curve(dexp(x, rate=8) , 0, 3, lwd=3, xlab = "", ylab = "", add = T, col='yellow')
curve(dexp(x, rate=4) , 0, 3, lwd=3, xlab = "", ylab = "", add = T, col='darkgoldenrod3')

# Another way:
library(ggplot2)
range <- 1:1000000
df <- data.frame(range, rexp(range, 2))
df2 <- data.frame(range, rexp(range, 3.5))
df3 <- data.frame(range, rexp(range, 5))
df4 <- data.frame(range, rexp(range, 8))

df5 <- data.frame(df, df2, df3, df4)

options(repr.plot.width = 8, repr.plot.height = 6)
ggplot(df5) +
  geom_density(aes(rexp.range..2., color = 'rate = 2')) +
  geom_density(aes(rexp.range..3.5., color = 'rate = 3.5')) +
  geom_density(aes(rexp.range..5., color = 'rate = 5')) +
  geom_density(aes(rexp.range..8., color = 'rate = 8')) +
  xlab("x")
```





در قسمت بعدی سوال می‌خواهیم تابعی بنویسیم که اعداد تصادفی با توزیع نرمال را تولید کنید. در ابتدا CDF توزیع نرمال را پیدا می‌کنیم:

$$\frac{1}{2}\left[1+\operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$$

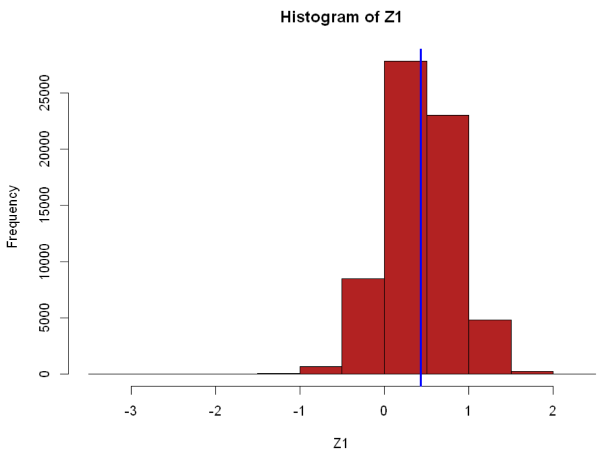
که در اینجا erf به صورت زیر تعریف شده است:

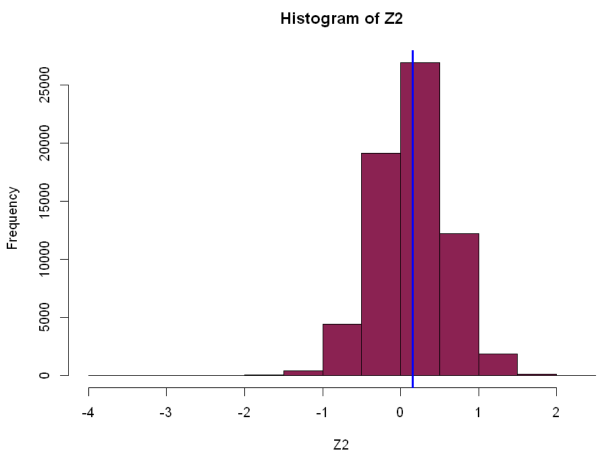
$$\operatorname{erf} z = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2}\, dt$$

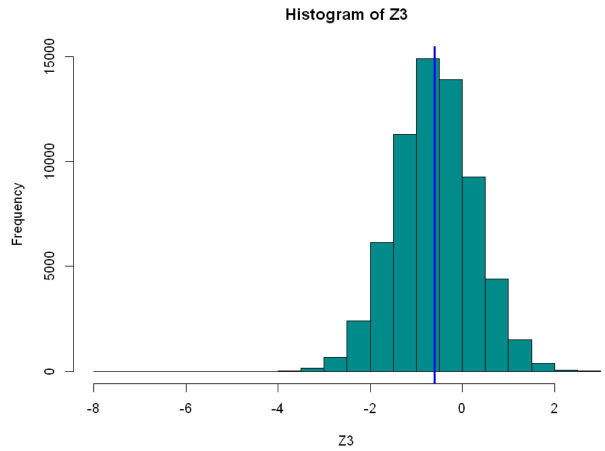
حساب کردن این تابع کار راحتی نیست چه برسد به حساب کردن معکوس آن! اما تقریب‌هایی برای آن وجود دارند که می‌توان به کمک آنها erf^{-1} را با دقت نسبتا خوبی حساب کرد. یکی از این توابع در بلاک زیر آمده است. [منبع] (https://people.maths.ox.ac.uk/gilesm/codes/erfinv/gems.pdf)
به کمک این تابع داده شده، تابعی بنویسید که بتوان به کمک آن اعداد تصادفی تولید کرد که از توزیع نرمال با انحراف معیار سیگما و میانگین مو پیروی می‌کند.

In [183]:
```
erfinv <- function(x) {
    w <- -log((1 - x) * (1 + x))
    if (any(w < 5.000000)) {
        w <- w - 2.500000
        p <- 2.81022636e-08
        p <- 3.43273939e-07 + p * w
        p <- -3.5233877e-06 + p * w
        p <- -4.39150654e-06 + p * w
        p <- 0.00021858087 + p * w
        p <- -0.00125372503 + p * w
        p <- -0.00417768164 + p * w
        p <- 0.246640727 + p * w
        p <- 1.50140941 + p * w
    } else {
        w <- sqrt(w) - 3.000000
        p <- -0.000200214257
        p <- 0.000100950558 + p * w
        p <- 0.00134934322 + p * w
        p <- -0.00367342844 + p * w
        p <- 0.00573950773 + p * w
        p <- -0.0076224613 + p * w
        p <- 0.00943887047 + p * w
        p <- 1.00167406 + p * w
        p <- 2.83297682 + p * w
    }
    return(p * x)
}
```

In [184]:
```
norm_random <- function(mio, sigma, x) mio + sigma*sqrt(2)*erfinv(x)

Z1 <- norm_random(0.43, 0.4, 2*as.numeric(li) - 1)
Z2 <- norm_random(0.15, 0.45, 2*as.numeric(li) - 1)
Z3 <- norm_random(-0.6, 0.85, 2*as.numeric(li) - 1)
hist(Z1, col='firebrick')
abline(v=mean(Z1), lwd=3, col='blue')
hist(Z2, col='violetred4')
abline(v=mean(Z2), lwd=3, col='blue')
hist(Z3, col='cyan4')
abline(v=mean(Z3), lwd=3, col='blue')
```

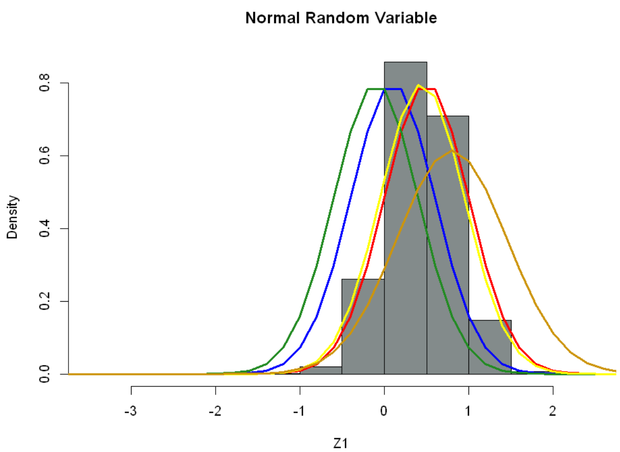


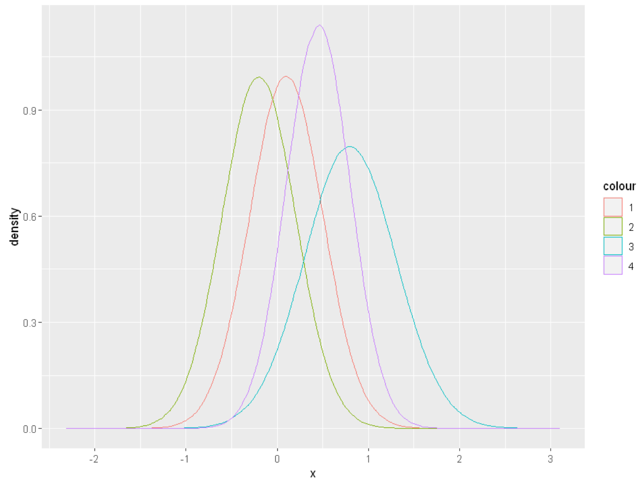




به کمک نمودار و مقایسه با rnorm به ازای ورودی‌های مختلف نشان دهید که تابع شما درست کار می‌کند.

In [185]:
```
hist(Z1, freq=F, xlab='Z1', main='Normal Random Variable', col='azure4')
curve(dnorm(x, 0.1, 0.5) , -10, 10, lwd=3, xlab = "", ylab = "", add = T, col='blue')
curve(dnorm(x, 0.5, 0.5) , -10, 10, lwd=3, xlab = "", ylab = "", add = T, col='red')
curve(dnorm(x, -0.1, .5) , -10, 10, lwd=3, xlab = "", ylab = "", add = T, col='forestgreen')
curve(dnorm(x, 0.45, 0.5) , -10, 10, lwd=3, xlab = "", ylab = "", add = T, col='yellow')
curve(dnorm(x, 0.8, .65) , -10, 10, lwd=3, xlab = "", ylab = "", add = T, col='darkgoldenrod3')

# Another way:
library(ggplot2)
range <- 1:1000000
df <- data.frame(range, rnorm(range, 0.1, 0.4))
df2 <- data.frame(range, rnorm(range, -0.2, 0.4))
df3 <- data.frame(range, rnorm(range, 0.8, 0.5))
df4 <- data.frame(range, rnorm(range, 0.45, 0.35))

df5 <- data.frame(df, df2, df3, df4)

options(repr.plot.width = 8, repr.plot.height = 6)
ggplot(df5) +
  geom_density(aes(rnorm.range..0.1..0.4., color = '1')) +
  geom_density(aes(rnorm.range...0.2..0.4., color = '2')) +
  geom_density(aes(rnorm.range..0.8..0.5., color = '3')) +
  geom_density(aes(rnorm.range..0.45..0.35., color = '4')) +
  xlab("x")
```



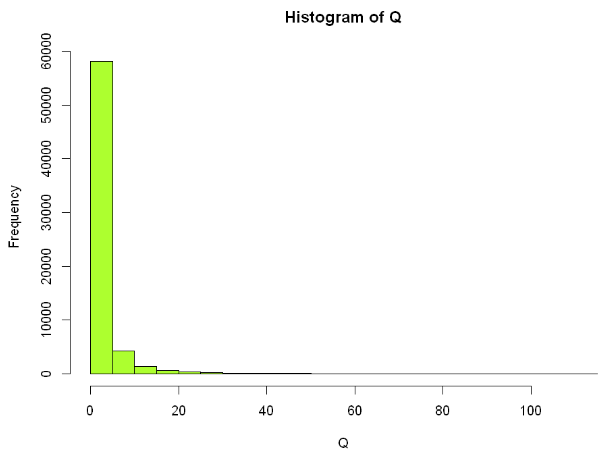


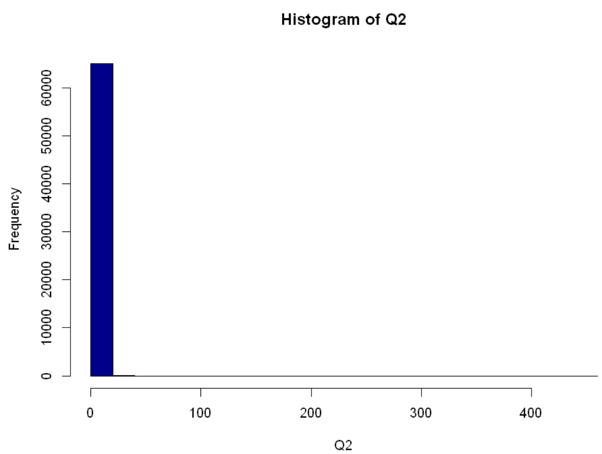
در قسمت آخر این سوال می‌خواهیم همین کارها را بر روی یک توزیع دیگر انجام دهیم. PDF این توزیع به صورت زیر است:

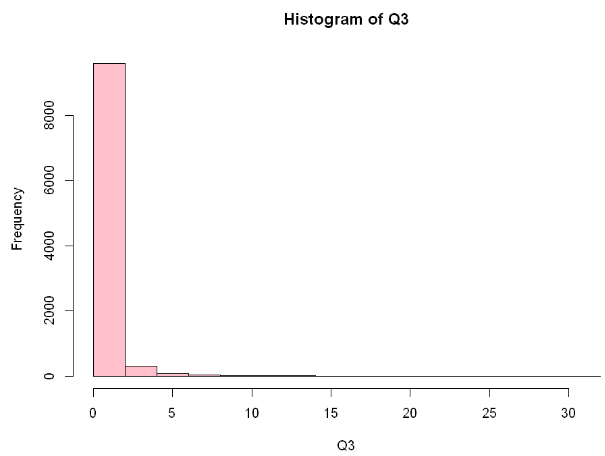
$$\frac{1}{2\sqrt{x}}\exp(-\sqrt{x})$$

در ابتدا CDF این تابع را بدست آورید و سپس تابعی بنویسد که اعداد تصادفی با این توزیع تولید کند. در نهایت، نمودار PDF این تابع را رسم کنید. نمودار PDFی که در صورت سوال به شما داده شده است را با نموداری که رسم کرده‌اید مقایسه کنید.

In [186]:
```
# finding CDF from integral of PDF from -∞ to x
# --> 1 - e^(-sqrt(x))
# finding function from inverse of CDF :
# (1-e^(-sqrt(x)))' = ln^2(1-x)
# So:
funct3 <- function(x) (log(1-x))^2
# funct3(0.7)    # Example
Q <- funct3(as.numeric(li))
hist(Q, col='greenyellow')

orig_funct <- function(x) (1/(2*sqrt(x)))*exp(-sqrt(x))
Q2 <- orig_funct(as.numeric(li))
hist(Q2, col='darkblue')
Q3 <- orig_funct(runif(10000))
hist(Q3, col='pink')
```






Histogram of Q3
Frequency
0
2000
4000
6000
8000
0
5
10
15
20
25
30
Q3