lab 1

2025-10-05

1 Install package

```
options(repos = c(CRAN = "https://cloud.r-project.org/"))
install.packages(c("corrplot"))
library(corrplot)
```

```
## corrplot 0.95 loaded
```

2 Correlation coefficient

2.1 Correlation coefficient visualization

```
(R=cor(state.x77))
```

```
##             Population     Income  Illiteracy    Life Exp     Murder
## Population  1.00000000  0.2082276  0.10762237 -0.06805195  0.3436428
## Income      0.20822756  1.0000000 -0.43707519  0.34025534 -0.2300776
## Illiteracy  0.10762237 -0.4370752  1.00000000 -0.58847793  0.7029752
## Life Exp   -0.06805195  0.3402553 -0.58847793  1.00000000 -0.7808458
## Murder      0.34364275 -0.2300776  0.70297520 -0.78084575  1.0000000
## HS Grad    -0.09848975  0.6199323 -0.65718861  0.58221620 -0.4879710
## Frost      -0.33215245  0.2262822 -0.67194697  0.26206801 -0.5388834
## Area        0.02254384  0.3633154  0.07726113 -0.10733194  0.2283902
##                HS Grad      Frost        Area
## Population -0.09848975 -0.3321525  0.02254384
## Income      0.61993232  0.2262822  0.36331544
## Illiteracy -0.65718861 -0.6719470  0.07726113
## Life Exp    0.58221620  0.2620680 -0.10733194
## Murder     -0.48797102 -0.5388834  0.22839021
## HS Grad     1.00000000  0.3667797  0.33354187
## Frost       0.36677970  1.0000000  0.05922910
## Area        0.33354187  0.0592291  1.00000000
```

```
corrplot(R,diag=F)
```

Function "cor" calculates the Pearson correlation coefficient between each variable, returning a matrix of correlation coefficient. "corrplot" provides visualization of the correlation matrix, where "diag=F" omits the diagonals. Colors represent correlation strength and direction, where red represents negative and blue represents positive correlation.

2.2 Correlation test

t-test

```
r=R["Murder","Frost"]
cor.test(~ Murder+Frost,data=state.x77)
```

```
##
##  Pearson's product-moment correlation
##
## data:  Murder and Frost
## t = -4.4321, df = 48, p-value = 5.405e-05
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.7106377 -0.3065115
## sample estimates:
##        cor
## -0.5388834
```

z-test

```
r=R["Murder","Frost"]
z=sqrt(50-2)*r
pvalue=2*(1-pnorm(abs(z)))
pvalue
```

```
## [1] 0.0001888417
```

permutation test

```
x=state.x77[,"Frost"]
y=state.x77[,"Murder"]
n=length(x)
t0=sum(x*y)-n*mean(x)*mean(y)
t_per=NULL
N=1000000
for (i in 1:N){
  x_per=sample(x)#permute
  t_per[i]=sum(x_per*y)-n*mean(x_per)*mean(y)
}
p2=mean(abs(t_per)>=abs(t0))#mean can apply to logical values
p2
```

```
## [1] 6.9e-05
```

Now we generate binary normal distributed data to check whether the p-value of t-test, z-test and permutation test are close:

```
#generate data
set.seed(111)
n=20
x=rnorm(n)
y=rnorm(n)
r=cor(x,y)
#t-test:sqrt(n-2)*r/sqrt(1-r^2)
cor.test(x,y)->tmp
pvalue.ttest=tmp$p.value
print(pvalue.ttest)
```

```
## [1] 0.7485246
```

```
#z-test:sqrt(n-2)*r
z=sqrt(n-2)*r
pvalue.ztest=2*(1-pnorm(abs(z)))
print(pvalue.ztest)
```

```
## [1] 0.745493
```

```r
#permutation test
r0=cor(x,y)
R_per=NULL
N=100000
for(i in 1:N){
  x_per=sample(x)
  R_per[i]=cor(x_per,y)
}
pvalue.per=mean(abs(R_per)>=abs(r0))
print(pvalue.per)
```

```
## [1] 0.74752
```

Exercise 1

Now we generate binary non-Gaussian distributed data,where $x_i \sim B(1, 0.3), y_i \sim B(1, 0.6)$:

```r
#generate data
set.seed(666)
n=20
x <- rbinom(n, size = 1, prob = 0.3)
y <- rbinom(n, size = 1, prob = 0.6)
r=cor(x,y)
#t-test
cor.test(x,y)->tmp
pvalue.ttest=tmp$p.value
print(pvalue.ttest)
```

```
## [1] 0.3033128
```

```r
#z-test:sqrt(n-2)*r
z=sqrt(n-2)*r
pvalue.ztest=2*(1-pnorm(abs(z)))
print(pvalue.ztest)
```

```
## [1] 0.3039134
```

```r
#permutation test
r0=cor(x,y)
R_per=NULL
N=100000
for(i in 1:N){
  x_per=sample(x)
  R_per[i]=cor(x_per,y)
}
pvalue.per=mean(abs(R_per)>=abs(r0))
print(pvalue.per)
```

```
## [1] 0.37528
```

2.3 Nonparametric test

```
x=c(2,-2,-11,3,4)
y=c(0,-1,-3,99,7)
rankx=rank(x)
ranky=rank(y)
rankx
```

```
## [1] 3 2 1 4 5
```

```
ranky
```

```
## [1] 3 2 1 5 4
```

```
pearson=cor(x,y)
print(pearson)
```

```
## [1] 0.407719
```

```
spearman=cor(rankx,ranky)
print(spearman)
```

```
## [1] 0.9
```

```
cor.test(x,y,method="spearman")
```

```
##
##  Spearman's rank correlation rho
##
## data:  x and y
## S = 2, p-value = 0.08333
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
## rho
## 0.9
```

Exercise 2

```
set.seed(666)
x=c(2,-2,-11,3,4)
y=c(0,-1,-3,99,7)
rankx=rank(x)
ranky=rank(y)
r0=cor(rankx,ranky)
pvalue<-function(n,x,y,r0){
  r=NULL
  for (i in 1:n){
```

```
    x_per=sample(x)
    rankx_per=rank(x_per)
    r[i]=cor(rankx_per,ranky)
  }
  p=mean(abs(r)>=abs(r0))
  return(p)
}
p1=pvalue(n=1000,x=x,y=y,r0=r0)
p2=pvalue(n=10000,x=x,y=y,r0=r0)
p3=pvalue(n=100000,x=x,y=y,r0=r0)
p1
```

```
## [1] 0.094
```

```
p2
```

```
## [1] 0.0808
```

```
p3
```

```
## [1] 0.08441
```

We can see that as n increases, p-value of permutation test becomes closer to Spearman test.

3 Monte Carlo method

```
n=100000
x=runif(n,-1,1);y=runif(n,-1,1)
m=sum(x^2+y^2<=1)
p=m/n
S=4*p
S
```

```
## [1] 3.1426
```

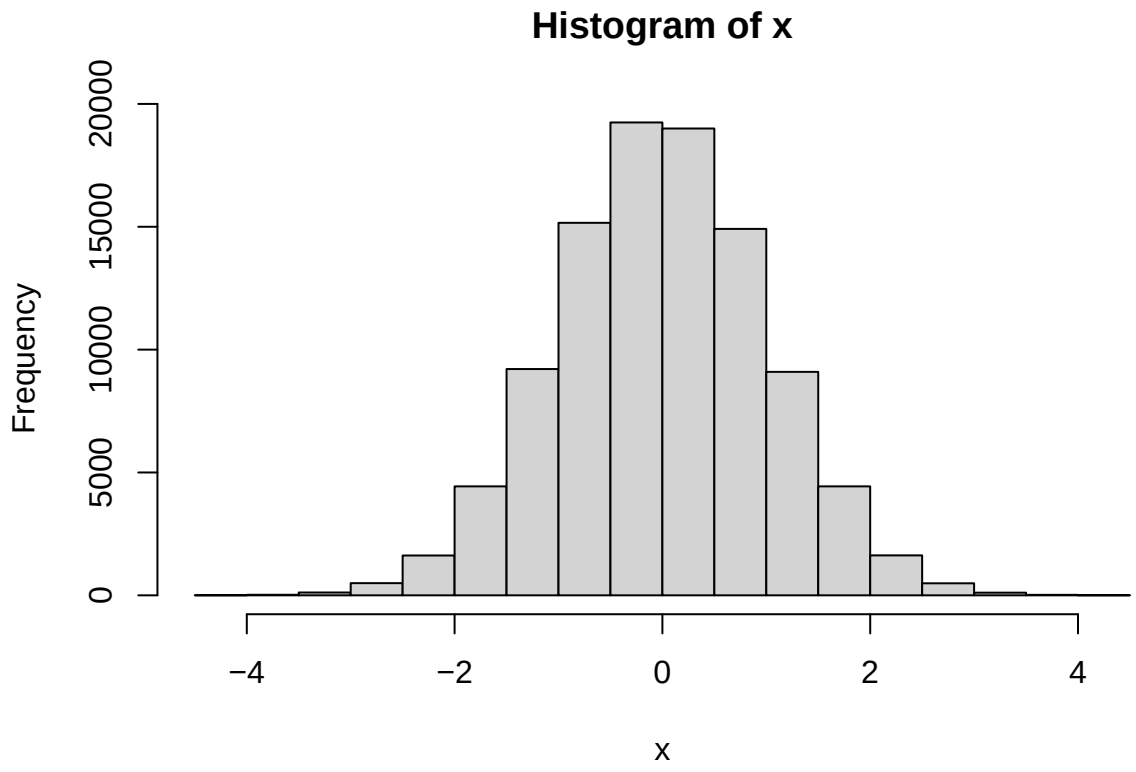
```
n=100000
x=rnorm(n)
mean(sqrt(abs(x)))
```

```
## [1] 0.8221677
```

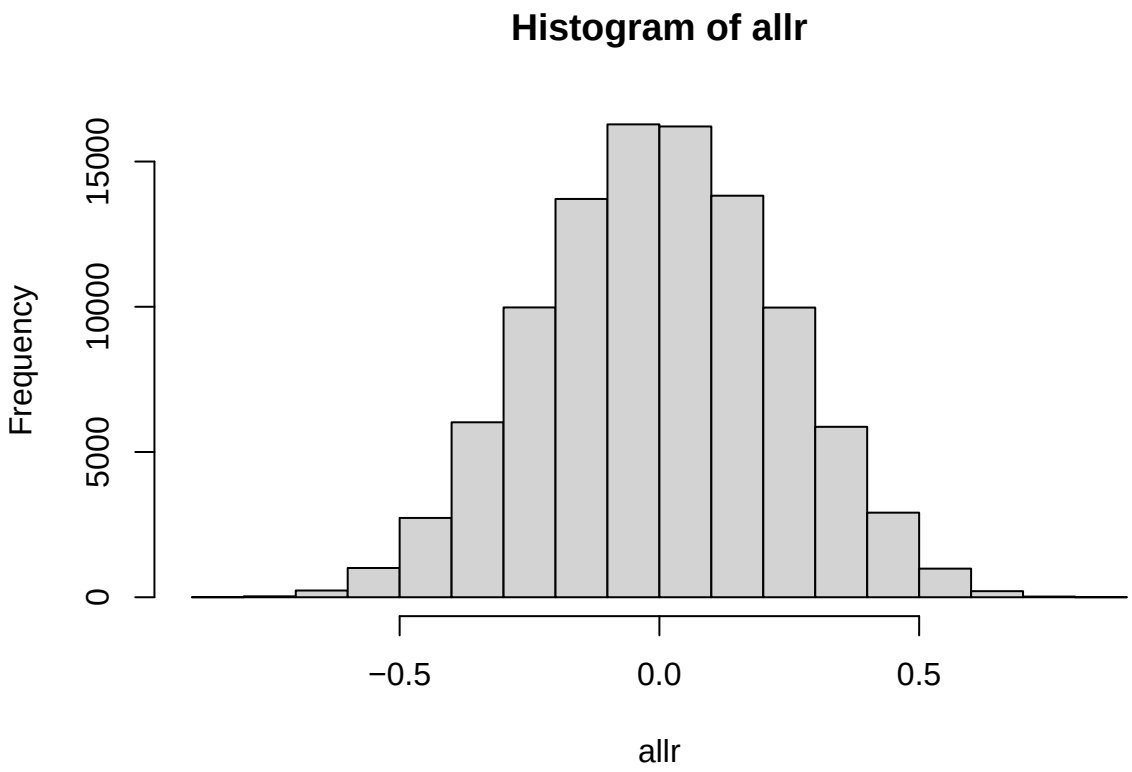
```
var(sqrt(abs(x)))
```

```
## [1] 0.1219015
```

```r
hist(x)
```



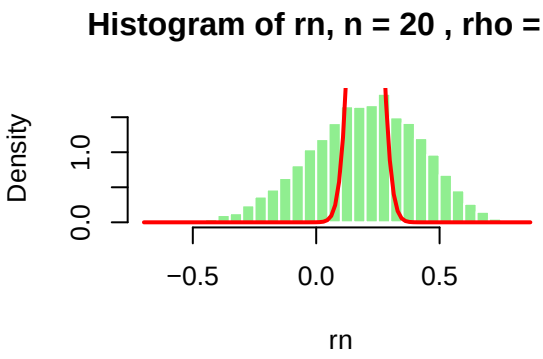

```r
set.seed(666)
n=20
N=100000
allr=NULL
for (k in 1:N){
  x=rnorm(n)
  y=rnorm(n)
  r.k=cor(x,y)
  allr=c(allr,r.k)
}
hist(allr)
```

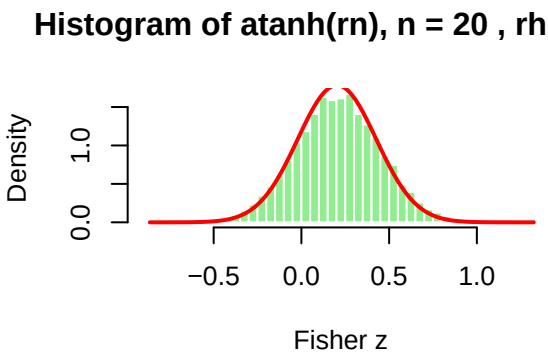


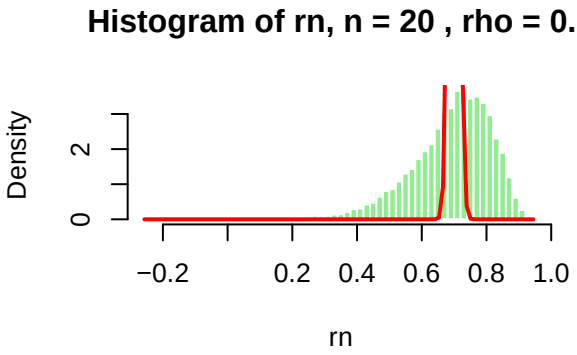
Exercise 3

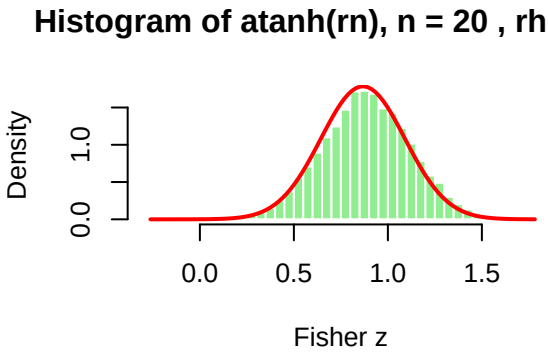
```
# generate data
set.seed(666)
samples<-function(n,rho){
  x=rnorm(n);y=rnorm(n)
  y=rho*x+sqrt(1-rho^2)*y
  return (list(x=x,y=y))
}
# Monte Carlo
Monte_carlo<-function(N,n,rho){
  rn=numeric(N)
  for (i in 1:N){
      data<-samples(n,rho)
      x=data$x;y=data$y
      rn[i]=cor(x,y)
  }
  atanh_rn=atanh(rn)
  return (list(rn=rn,atanh_rn=atanh_rn))
}
# plot
N=10000
par(mfrow = c(2, 2))
for (n in c(20,100)) {
```

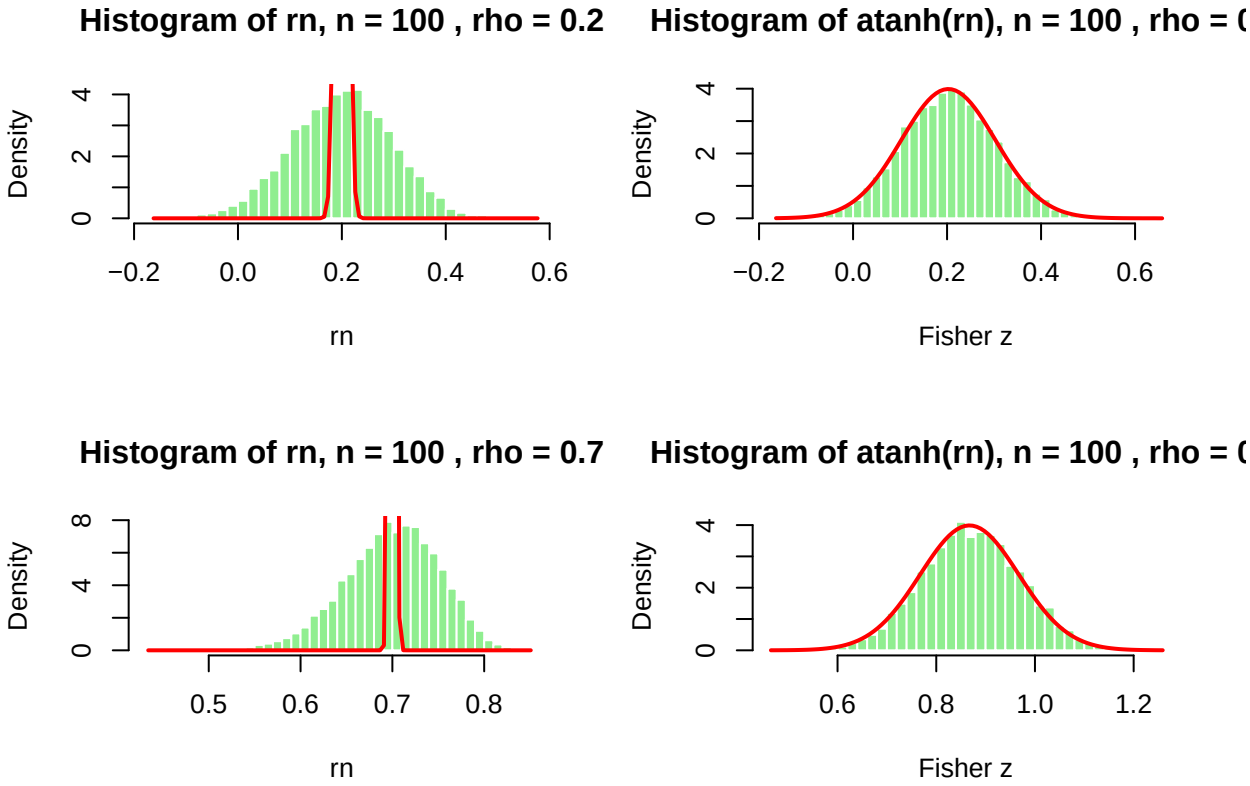
```
  for (rho in c(0.2,0.7)) {
result=Monte_carlo(N=N,n=n,rho=rho)
rn=result$rn;atanh_rn=result$atanh_rn
# Histogram for rn with normal curve
    hist_data = hist(rn, breaks = 50, freq = FALSE,
                     main = paste("Histogram of rn, n =", n, ", rho =", rho),
                     col = "lightgreen", border = "white")
    # Theoretical mean and sd for rn
    mean_z = rho
    sd_z = (1-rho^2)^2/n
    # Generate normal curve
    x_vals = seq(min(rn), max(rn), length.out = 100)
    y_vals = dnorm(x_vals, mean = mean_z, sd = sd_z)
    lines(x_vals, y_vals, col = "red", lwd = 2)
# Histogram for atanh_rn with normal curve
    hist_data = hist(atanh_rn, breaks = 50, freq = FALSE,
                     main = paste("Histogram of atanh(rn), n =", n, ", rho =", rho),
                     xlab = "Fisher z", col = "lightgreen", border = "white")
    # Theoretical mean and sd for atanh_rn
    mean_z = atanh(rho)
    sd_z = sqrt(1 / (n))
    # Generate normal curve
    x_vals = seq(min(atanh_rn), max(atanh_rn), length.out = 100)
    y_vals = dnorm(x_vals, mean = mean_z, sd = sd_z)
    lines(x_vals, y_vals, col = "red", lwd = 2)
  }
}
```


Histogram of rn, n = 20 , rho = 0.2
Density
0.0
1.0
−0.5
0.0
0.5
rn


Histogram of atanh(rn), n = 20 , rho = 0
Density
0.0
1.0
−0.5
0.0
0.5
1.0
Fisher z


Histogram of rn, n = 20 , rho = 0.7
Density
0
2
−0.2
0.2
0.4
0.6
0.8
1.0
rn


Histogram of atanh(rn), n = 20 , rho = 0
Density
0.0
1.0
0.0
0.5
1.0
1.5
Fisher z



As is shown in the plot, the red curve shows the asymptotic normal distribution and the green curve shows the histogram of Monte Carlo method. We can see that the asymptotic distribution of r(n) is not a suitable choice, since its variance depends on its mean, which is also the correlation coefficient rho. When rho is large, the variance is large, making it extremely concentrated and peaked. On the other hand, the histogram of atanh(rn) apparently suits asymptotic distribution better.After Fisher transformation, the asymptotic distribution becomes more stable, as its variance no longer depend on its mean. Therefore, using Fisher transformation yields a better approximation.