```
                          ## Generating Discrete Random Variables ##

##############################################


getwd()                                       # determine the working directory

setwd("./Documents/Teaching")
                                              # change the working directory
                                              # . refers to the current directory
getwd()

##############################################

## The Inverse Transform Method

# Example 4a Ross (2006)

x<-1:4
p<-c(0.2,0.15,0.25,0.40)
Fx<-cumsum(p)

U<-runif(1)
X<-1
while (Fx[X]<U){
      X<-X+1
}
print(X)

                                              # Simulation study
N<-5000
set.seed(1)
U<-runif(N)
X<-rep(0,N)
for (i in 1:N){
      j<-1
      while (Fx[j]<U[i]){
            j<-j+1
      }
      X[i]<-j
}
#print(X)

freq<-rep(0,4)
for (i in x) freq[i]<-sum(X==i)/N
#freq<-as.numeric(table(X))/N
plot(x,freq,type="h",lwd=3,ylim=c(0,max(p,freq)))
lines(x+0.05,p,type="h",col="red",lwd=3)

# pdf("inv_discr1.pdf",paper="special")

# plot(x,freq,type="h",lwd=3,ylim=c(0,max(p,freq)),axes=F,main="Simulation study")
# lines(x+0.05,p,type="h",col="red",lwd=3)
# axis(1,1:4)
# axis(2)
# box()
# legend("topleft",c("observed","theoretical"),lty=c(1,1),
#        lwd=c(3,3),col=c(1,2),bty="n")

# dev.off()

                                               # Using of sample()

X<-sample(x,size=N,replace=T,prob=p)

freq<-as.numeric(table(X))/N
plot(x,freq,type="h",lwd=3,ylim=c(0,max(p,freq)))
lines(x+0.05,p,type="h",col="red",lwd=3)


## Uniform discrete rv's
                                               # Using 'floor()'
?floor

N<-5000
n<-10
U<-runif(N)
X<-floor(n*U)+1

freq<-as.numeric(table(X))/N
plot(1:n,freq,type="h",lwd=3,ylim=c(0,max(1/n,freq)))
lines(1:n+0.05,rep(1/n,n),type="h",col="red",lwd=3)

                                               # Using 'sample()'

X<-sample(1:n,size=N,replace=T)

freq<-as.numeric(table(X))/N
plot(1:n,freq,type="h",lwd=3,ylim=c(0,max(1/n,freq)))
lines(1:n+0.05,rep(1/n,n),type="h",col="red",lwd=3)


## Random permutation

# Es 4b, Ross (2006)

                                               # take 1
n<-10
pi.0<-1:n
pi<-rep(0,n)
k<-n
while (k>1){
      #cat("k=",k,"\n")
      U<-runif(1)
      I<-floor(k*U)+1
      #cat("I=",I,"\n")
      pi[k]<-pi.0[I]
      pi.0<-pi.0[-I]
      #cat("pi=",pi,"\n")
      #cat("pi.0=",pi.0,"\n")
      k<-k-1
}
pi[1]<-pi.0
print(pi)

                                               # take 2
n<-10
pi<-1:n
k<-n
while (k>1){
      #cat("k=",k,"\n")
      U<-runif(1)
      I<-floor(k*U)+1
      #cat("I=",I,"\n")
      x<-pi[k]
      pi[k]<-pi[I]
      pi[I]<-x
      #cat("pi=",pi,"\n")
      k<-k-1
}
print(pi)

                                              # Using 'sample()'

pi<-sample(1:n)
print(pi)

# random subset of size r<= n/2

                                               # take 1
n<-10
r<-4
pi.0<-1:n
pi<-rep(0,r)
k<-n
while (k>(n-r)){
      #cat("k=",k,"\n")
      U<-runif(1)
      I<-floor(k*U)+1
      #cat("I=",I,"\n")
      pi[k-n+r]<-pi.0[I]
      pi.0<-pi.0[-I]
      #cat("pi=",pi,"\n")
      #cat("pi.0=",pi.0,"\n")
      k<-k-1
}
print(pi)

                                               # take 2
n<-10
r<-4
pi<-1:n
k<-n
while (k>(n-r)){
      #cat("k=",k,"\n")
      U<-runif(1)
      I<-floor(k*U)+1
      #cat("I=",I,"\n")
      x<-pi[k]
      pi[k]<-pi[I]
      pi[I]<-x
      #cat("pi=",pi,"\n")
      k<-k-1
}
print(pi[(n-r+1):n])

                                               # Using 'sample()'

pi<-sample(1:n,size=r)
print(pi)


## Binomial random variable

                                               # calculate cdf of binom rv
binom.cdf<-function(x,n,p){
      Fx<-0
      for (i in 0:x){
            Fx<-Fx+choose(n, i)*p^i*(1-p)^(n-i)   # 'choose()' compute binomial coef
      }
      return(Fx)
}

n<-10
p<-0.5
binom.cdf(1,n,p)
pbinom(1,size=n,prob=p)

                                               # simulate X ~ F
cdf.sim<-function(F,...){
      X<-0
      U<-runif(1)
      while (F(X,...)<U){
            X<-X+1
      }
      return(X)
}

cdf.sim(binom.cdf,n,p)
rbinom(1,size=n,prob=p)

                                               # Simulation study
N<-5000
n<-10
p<-0.5
X<-rep(0,N)
set.seed(1)
for (i in 1:N){
      X[i]<-cdf.sim(binom.cdf,n,p)
```

```
}
#print(X)

freq<-rep(0,n+1)
for (i in 0:n) freq[i+1]<-sum(X==i)/N
#freq<-as.numeric(table(X))/N

p.t<-dbinom(0:n,size=n,prob=p)
plot(0:n,freq,type="h",lwd=3,ylim=c(0,max(freq,p.t)))
lines(0:n+0.05,p.t,type="h",col="red",lwd=3)

# pdf("binom1.pdf",paper="special")

# plot(0:n,freq,type="h",lwd=3,ylim=c(0,max(freq,p.t)),main="binom(n=10,p=0.5)")
# lines(0:n+0.05,p.t,type="h",col="red",lwd=3)
# legend("topleft",c("observed","theoretical"),lty=c(1,1),
       # lwd=c(3,3),col=c(1,2),bty="n")

# dev.off()

                                               # combine loop in cdf.sim with
                                               # the loop in binom.cdf
binom.sim <- function(n,p){
      X<-0
      px<-(1-p)^n
      Fx<-px
      U<-runif(1)
      while (Fx<U) {
            X<-X+1
            px<-px*((n-X+1)*p)/(X*(1-p))         # compute px via recursive formula
            Fx<-Fx+px
      }
      return(X)
}

set.seed(1)
system.time(                                   # returns CPU time taken for execution
for (i in 1:N){
      X[i]<-cdf.sim(binom.cdf,n,p)
}
)
                                               # check higher efficiency, i.e.
                                               # less computing time
set.seed(1)
system.time(
for (i in 1:N){
      X[i]<-binom.sim(n,p)
}
)


## Sequences of independent trials

p<-0.5
U<-runif(1)
```

```
                                               # simulate B ~ Bernulli(p)
if (U<p) {
      B<-1
} else B<-0

print(B)

                                               # simulate n iid B_i ~ Bernulli(p)
n<-10
B<-rep(0,n)
for (i in 1:n) {
      U<-runif(1)
      if (U<p) {
            B[i]<-1
      } else B[i]<-0
}
print(B)

                                               # simulate X ~ binom(n,p) as
                                               # X=sum_{i=1}^n B_i
n<-10
p<-0.5
X<-0
for (i in 1:n){
      U<-runif(1)
      if (U<p) X<-X+1
}
print(X)

X<-sum(runif(n)<p)                             # simpler
print(X)


# Geometric random variable

p<-0.5

                                               # simulate Y ~ geom(p) as smaller i
                                               # such that B_j=0 for j=1,...,i-1
                                               # and B_i=1
Y<-0
success <-FALSE
while (!success) {
      U<-runif(1)
      if (U<p) {
            success <-TRUE
      } else {
            Y<-Y+1
      }
}

                                               # using inverse transform method
U<-runif(1)
Y<-floor(log(U)/log(1-p))+1
print(Y)
```

```
                                                # Simulation study
N<-5000
set.seed(100)
U<-runif(N)
Y<-floor(log(U)/log(1-p))+1
#print(Y)

y.max<-max(Y)
freq<-rep(0,y.max)
for (i in 1:y.max) freq[i]<-sum(Y==i)/N
plot(1:y.max,freq,type="h",lwd=3,ylim=c(0,p))
lines(1:y.max+0.05,dgeom(0:(y.max-1),prob=p),type="h",col="red",lwd=3)
                                                # check '?dgeom'

# pdf("geom1.pdf",paper="special")

# plot(1:y.max,freq,type="h",lwd=3,ylim=c(0,p),main="geom(p=0.5)")
# lines(1:y.max+0.05,dgeom(0:(y.max-1),prob=p),type="h",col="red",lwd=3)
# legend("topright",c("observed","theoretical"),lty=c(1,1),
       # lwd=c(3,3),col=c(1,2),bty="n")

# dev.off()


## Poisson random variable

l<-3                                            # set value of parameter lambda

                                                # inverse transform algorithm
pois.sim <- function(l){
    X<-0
    px<-exp(-l)
    Fx<-px
    U<-runif(1)
    iter<-0                                     # dummy var, counts how many searches
    while (Fx<U) {
        iter<-iter+1
        X<-X+1
        px<-px*l/X
        Fx<-Fx+px
    }
    cat("X=",X,"\n")
    cat("num searches=",iter,"\n")
    return(X)
}

pois.sim(l)

                                                # Simulation study
set.seed(5)
N<-1000
X<-rep(0,N)
for (i in 1:N){
      X[i]<-pois.sim(l)
```

```
}
#print(X)

x.max<-max(X)+5
freq<-rep(0,x.max+1)
for (i in 0:x.max) freq[i+1]<-sum(X==i)/N
plot(0:x.max,freq,type="h",lwd=3)
lines(0:x.max+0.1,dpois(0:x.max,lambda=l),type="h",col="red",lwd=3)

# pdf("pois1.pdf",paper="special")

# plot(0:x.max,freq,type="h",lwd=3,xlab="x",ylab="prob",
     # main=expression(paste("Pois(",lambda,"=3), ",N==1000)))
# lines(0:x.max+0.1,dpois(0:x.max,lambda=l),type="h",col="red",lwd=3)
# legend("topright",c("observed","theoretical"),lty=c(1,1),
       # lwd=c(3,3),col=c(1,2),bty="n")

# dev.off()

                                               # 2 take
                                               # inverse transform
                                               # with more efficient search (?)
pois.sim1 <- function(l){
    X<-floor(l)
    px<-rep(exp(-l),3*X)
    Fx<-px[1]
    for (i in 1:X){
      px[i+1]<-px[i]*l/i
      Fx<-Fx+px[i+1]
    }
    #cat("px=",px[1:(X+1)],"\n")
    #cat("true px=",dpois(0:X,lambda=l),"\n")
    #cat("X=",X,"\n")
    U<-runif(1)
    #cat("U=",U,"\n")
    #cat("Fx=",Fx,"\n")
    #cat("true Fx=",ppois(X,lambda=l),"\n")
    iter<-0                                    # dummy var, counts how many searches
    if (Fx<U) {
      while (Fx<U) {
            iter<-iter+1
            X<-X+1
            #cat("X=",X,"\n")
            px[X+1]<-px[X]*l/X
            #cat("px=",px[1:(X+1)],"\n")
            #cat("true px=",dpois(0:X,lambda=l),"\n")
            Fx<-Fx+px[X+1]
            #cat("U=",U,"\n")
            #cat("Fx",Fx,"\n")
            #cat("true Fx=",ppois(X,lambda=l),"\n")
      }
     } else {
      while (Fx>=U) {
            iter<-iter+1
            #cat("current px=",px[X+1],"\n")
            Fx<-Fx-px[X+1]
            #cat("U=",U,"\n")
            #cat("Fx",Fx,"\n")
            #cat("true Fx=",ppois(X,lambda=l),"\n")
            X<-X-1
            #cat("X=",X,"\n")
      }
      X<-X+1
    }
    cat("X=",X,"\n")
    cat("num searches=",iter,"\n")
    return(X)
}

l<-100

pois.sim1(l)

                                              # check that pois.sim1 is equivalent
                                              # to pois.sim by setting the same
                                              # seed. Note smaller num of searches
                                              # in particular for l=100
seed<-1
seed<-seed+1
set.seed(seed)
pois.sim1(l)
set.seed(seed)
pois.sim(l)

                                              # check higher efficiency, i.e.
                                              # less computing time
seed<-seed+1
set.seed(seed)
system.time(
pois.sim1(l)
)

set.seed(seed)
system.time(
pois.sim(l)
)
                                              # not clear from CPU time

## Rejection Method

# Example 4f, Ross (2006)

n<-10
p<-c(0.11,0.12,0.09,0.08,0.12,0.10,0.09,0.09,0.10,0.10)
q<-rep(1/n,n)
#sum(p);sum(q)

c<-max(p/q)
print(c)                                       # [1] 1.2
```

```
prob.accept<-0
iter<-0
U<-1
while (U>=prob.accept){
      Y<-floor(n*runif(1))+1
      prob.accept<-p[Y]/(c*q[Y])
      U<-runif(1)
      #cat("prob accept=",prob.accept,"\n")
      #cat("U=",U,"\n")
      iter<-iter+1
}
X<-Y
print(X)
cat("num of iter=",iter,"\n")

set.seed(50)
N<-10000
num.iter<-rep(0,N)
X<-rep(0,N)
for (i in 1:N) {
      prob.accept<-0
      iter<-0
      U<-1
      while (U>=prob.accept){
            Y<-floor(n*runif(1))+1
            prob.accept<-p[Y]/(c*q[Y])
            U<-runif(1)
            iter<-iter+1
      }
      X[i]<-Y
      num.iter[i]<-iter
}
#print(X)
freq<-rep(0,n)
for (i in 1:n) freq[i]<-sum(X==i)/N
plot(1:n,freq,ylim=c(0,max(c(0.12,freq))),type="h",lwd=3)
lines(1:n+0.1,p,type="h",col="red",lwd=3)

#print(num.iter)
mean(num.iter)
c

# pdf("accept1.pdf",paper="special")

# plot(1:n,freq,ylim=c(0,max(c(0.12,freq))),
     # type="h",lwd=3,main="Example 4f, Ross (2006)")
# lines(1:n+0.1,p,type="h",col="red",lwd=3)
# legend("topright",c("observed","theoretical"),lty=c(1,1),
       # lwd=c(3,3),col=c(1,2),bty="n")

# dev.off()
```