1. An urn problem

$N =$ total number of balls in the urn - some orange balls and some black balls.

$p =$ probability that a tossed coin will land heads up, and $1 - p =$ probability that it will with tails up. Suppose a ball is chosen at random from the urn and is replaced by an orange ball if a tossed coin lands heads up, otherwise is replaced by a black ball.

Let $X_n =$ number of orange balls in the urn after $n^{th}$ stage and the remaining number of balls will be $N - X_n$. The state space, $S = \{0, 1, 2, ..., N\}$. Then the stochastic process is defined by $\{X_n, n \geq 0\}$.

a. Explain why this process is a Markov chain

$\{X_n, n \geq 0\}$ is a Markov chain because the next state depends only on the present state, and not on the past. In other words, this process satisfies the Markov property.

b. State space of the Markov chain

The state space is given by $S = \{0, 1, 2, ..., N\}$.

c. Diagram of Markov chain

For the case $N = 4$, $S = \{0, 1, 2, 3, 4\}$ and the transition diagram is given below:
d. Transition probability matrix

Suppose we currently have \( i \) orange balls in the urn, then remaining \( N - i \) balls are black. In the general case, the transition probability matrix \( P \) has the following entries:

\[
P_{i,i+1} = \frac{N-i}{N} p \\
P_{i,i-1} = \frac{i}{N} (1-p) \\
P_{i,i} = \frac{i}{N} p + \frac{N-i}{N} (1-p)
\]

The transition probability matrix for the case \( N = 4 \) is given below;

\[
P = \begin{bmatrix}
1 - p & p & 0 & 0 & 0 \\
\frac{1}{4}(1-p) & \frac{1}{2}p + \frac{3}{4}(1-p) & \frac{3}{4}p & 0 & 0 \\
0 & \frac{1}{2}(1-p) & \frac{1}{2}p + \frac{1}{2}p & 0 & 0 \\
0 & 0 & \frac{3}{4}(1-p) & \frac{3}{4}p + \frac{1}{4}(1-p) & \frac{1}{4}p \\
0 & 0 & 0 & 1 - p & p
\end{bmatrix}
\]

e. Classification of states

The classes of the states depends on the value of \( p \). Since \( p \in (0,1) \), then all states belong to one class and that class is recurrent (since the MC is finite, that class is positive recurrent). The period of each state is 1. Hence, the Markov chain is irreducible and aperiodic.

f. Stationary distribution

Since \( p \in (0,1) \), the Markov chain is irreducible, aperiodic, and positive recurrent (i.e. ergodic). This means that there exist a unique stationary distribution for the Markov chain. The stationary distribution is computed below for the case \( N = 2 \) and \( p = \frac{2}{3} \).

```r
TransitionMatrix <- function(N,p) # Generates the transition matrix
{
  M = matrix(0,(N+1),(N+1))
  for (i in 0:N )
  {
    for (k in 0:N)
    {
      M[(i+1),(k+1)] <- ifelse(k==(i+1), p*(N-i)/N,
        ifelse(k==(i-1), (1-p)*i/N,
          ifelse(k==i, p*(i/N)+(1-p)*(N-i)/N, 0)))
    }
  }
  return(M)
}
```
g. Transition probability matrix when $p = 1$

The transition probability matrix when $p = 1$ has the following entries:

\[
\begin{align*}
P_{i,i+1} &= \frac{N - i}{N} \\
P_{i,i-1} &= 0 \\
P_{i,i} &= \frac{i}{N}
\end{align*}
\]

\[
P = \begin{bmatrix}
0 & 1 & 0 & 0 & \ldots & 0 \\
0 & \frac{1}{N} & \frac{N-1}{N} & 0 & \ldots & 0 \\
0 & 0 & \frac{2}{N} & \frac{N-2}{N} & \ldots & 0 \\
\vdots & & & & & \vdots \\
0 & 0 & 0 & \ldots & \frac{N-1}{N} & 1 \\
0 & 0 & 0 & \ldots & 0 & \frac{1}{N}
\end{bmatrix}
\]

The state classes in this case are in three categories:

- Recurrent class: \{N\}
- Transient classes: \{0\}, \{1\}, \{2\}, \ldots, \{N-1\}
- Absorbing state: \{N\}

Classification of the states for the case $N = 2$ and $p = 1$ is shown below:

\[
\begin{align*}
N &= 2 \\
dtmcA &\gets \text{new("markovchain", transitionMatrix=TransitionMatrix(2, (2/3)), states=paste(0:2), name="An Urn problem," )} \\
\text{steadyStates(dtmcA)}
\end{align*}
\]

\[
\begin{bmatrix}
0 & 1 & 2 \\
1,] & 0.1111111 & 0.4444444 & 0.4444444
\end{bmatrix}
\]
Recurrent classes: {2}
Transient classes: {0}, {1}
The Markov chain is not irreducible
The absorbing states are: 2

h. BONUS: Expected time until only orange balls in the urn

For $1 \leq k \leq N$, let $T_k$ = waiting time for the chain to move from state $k$ to state $k+1$. Then $T_k \sim$ geometric RV with probability of success $p_k = \frac{N-k}{N} = \text{probability of moving one state up}$. The mean of $T_k$ is

$$E[T_k] = \frac{1}{p_k} = \frac{N}{N-k}.$$ 

Since $W = T_1 + T_2 + \cdots + T_{N-1}$, we have that

$$E[W] = E[T_1] + \cdots + E[T_{N-1}] = \frac{N}{N-1} + \frac{N}{N-2} + \cdots + \frac{N}{1} = \sum_{i=1}^{N-1} \frac{N}{i}.$$ 

2. An algorithm for simulating a branching process

An algorithm for simulating a branching process is given below. It starts with $X_0 = 1$ in generation 0. The next generation is given by the random number (binomially distributed with parameters $n = 5$ and $p = 0.25$) of offspring produced by the first individual, i.e. $X_1 \sim \text{binomial}(5,0.25)$. In general, the size of generation $i$ is $X_i$ and is given by the sum of $X_{i-1}$ i.i.d. $\sim \text{binomial}(5,0.25)$ RVs. Theoretically, the population will persist since the mean $E(X_i) = np = 5(0.25) = 1.25 > 1$. The function ‘BP.sim’, the plot of a sample path, the plot of the 100 sample means, and the computation of the mean of mean population sizes are all given below.

```r
# Branching process simulation - modified from MC.sim code from class
# X_0 = 1, offspring distribution is binomial(n=10, p=0.25)

BP.sim <- function(n,nbinom,p,x1) {
    sim <- vector("numeric",n+1)
    if (missing(x1)) {
        sim[1] <- 1 # initial condition: start with 1 individual at generation 0
    } else {
        sim[1] <- x1
    }
    for (i in 2:n) {
        sim[i] <- sum(rbinom(nbinom,5,p))
    }
    return(sim)
}
```
# vector of probabilities from the binomial offspring distribution
prob_vec <- dbinom(0:nbinom, nbinom, p)
for (i in 2:(n+1)) {
    offspring <- sample(as.numeric(0:nbinom), sim[i-1], prob=prob_vec, replace=T)
    newstate <- sum(offspring)
    sim[i] <- newstate
}
sim

# Generate a sample path and plot number of individuals as a function of
# generation (time)
n = 40
set.seed(17) # this shows an increasing population
run <- BP.sim(n, 5, 0.25, 1)
plot(0:n, run, type="l", col="blue", xlab="Generation", ylab="Number of Individuals")
BONUS: Compute the mean size after 40 generations.

```r
# Run the function 100 times, each time stopping at generation 40.
mean_size <- vector("numeric",100)
for (i in 1:100){
  run2 <- BP.sim(n,5,0.25,1)
  mean_size[i] <- mean(run2)
}

# Compute and plot the mean population size after 40 generations
plot(mean_size, type="b", col="blue", xlab="Sample",
ylab="Mean Population Size after 40 Generations",pch=19)
```

```
Compute the mean of the 100 sample means:
mean(mean_size)

[1] 931.9759
```
3. An algorithm for simulating a hyperexponential random variable, $Y$

Suppose $X_1 \sim \text{exponential}(\lambda_1 = 1)$, $X_2 \sim \text{exponential}(\lambda_2 = 0.1)$, $p_1 = 0.3$, and $p_2 = 0.7$. Then a hyperexponential random variable, $Y$, has a probability density function (PDF) of the form

$$f_Y(y) = p_1 f_{X_1}(y) + p_2 f_{X_2}(y) = 0.3e^{-y} + 0.07e^{-0.1y}, \quad y > 0.$$  

The cumulative distribution function (CDF) is obtained as mixture of the CDFs of $X_1$ and $X_2$ each evaluated at $y$. The CDF of $Y$ is derived as follows:

$$F_Y(y) = p_1 F_{X_1}(y) + p_2 F_{X_2}(y) = 0.3 \left(1 - e^{-y}\right) + 0.7 \left(1 - e^{-0.1y}\right), \quad y > 0.$$  

To simulate $Y$, sample a standard uniform random variable $U$ and then compute $Y$ as follows:

$$Y = \begin{cases} 
X_1 \sim \text{exp}(\lambda_1), & \text{if } U \leq 0.3 \\
X_2 \sim \text{exp}(\lambda_2), & \text{if } U > 0.3 
\end{cases}$$

# Define number of samples
n = 5000

# Define rates of the 2 exponential RVs
lambda1 = 1
lambda2 = 0.1

# Define mixture probabilities
p1 = 0.3
p2 = 0.7

# Exponential RV with parameter lambda_1 = 1
X1 <- rexp(n,lambda1)

# Exponential RV with parameter lambda_1 = 1
X2 <- rexp(n,lambda2)

# Plot the 2 exponential distributions
par(mfrow = c(1,2))
hist(X1, freq=F, breaks=30, col="tomato", main="Exponential(1) Distribution", xlab="Sampled Values")
hist(X2, freq=F, breaks=30, col="darkgreen", main="Exponential(0.1) Distribution", xlab="Sampled Values")
# Make a function to draw samples from the hyperexponential density
# (analogous to `rexp`)
rhyperexp <- function(n,p1,p2,lambda1,lambda2){
  sample <- vector("numeric", n)
  for (i in 1:n){
    U <- runif(1)
    if(U < p1){
      sample[i] <- rexp(1,lambda1)
    }
    else{
      sample[i] <- rexp(1,lambda2)
    }
  }
  sample
}

# Hyperexponential RV, mixture of the 2 exponential RVs above
par(mfrow = c(1,1))
hist(rhyperexp(5000,p1,p2,lambda1,lambda2), freq=F, breaks=50, col="wheat", main="Hyperexponential Distribution", xlab="Sampled Values")
The histograms above show the hyperexponential and the two exponential distributions. We notice that the shape looks similar but there is a sharper decrease in the height of the second bar in the hyperexponential as compared to the exponential distributions. Also, there is a larger spread (variance) in the hyperexponential distribution.

4. Sampling from a hyperexponential distribution using Metropolis-Hasting MCMC

We use the Metropolis-Hasting MCMC algorithm to sample from the hyperexponential distribution given in Problem 3 using a standard normal proposal distribution. For a sample of size 50,000, the Markov chain seems to converge to a stationary distribution - that is, the hyperexponential distribution given in question 3 when the two are compared. The histograms should look almost identical!

```r
# Density function of hyperexponential distribution (analogous to 'dexp')
dhyperexp <- function(y,p1,p2,lambda1,lambda2){
  x <- p1*dexp(y,lambda1) + p2*dexp(y,lambda2)
  x
}
```
# MH MCMC function - modified from MH_norm function in MCMC R script from class

MH_hyperexp <- function(n,p1,p2,lambda1,lambda2)
{
  samples <- vector("numeric", n)
  x <- 10
  samples[1] <- x
  for (i in 2:n) {
    innov <- rnorm(1,0,1)
    proposal <- x + innov
    aprob <- min(1,dhyperexp(proposal,p1,p2,lambda1,lambda2)/
                 dhyperexp(x,p1,p2,lambda1,lambda2))
    u <- runif(1)
    if (u < aprob){
      x <- proposal
    }
    samples[i] <- x
  }
  samples
}

# Run the MH function above
run <- MH_hyperexp(50000,p1,p2,lambda1,lambda2)

# Plot sampled values as a time series and histogram
#par(mfrow=c(2,1))
plot(ts(run), xlab="Iteration", ylab="Sampled Values", col="darkgreen", main="Metropolis-Hastings MCMC: Hyperexponential Target Distribution")
Metropolis–Hastings MCMC: Hyperexponential Target Distribution

```r
hist(run, breaks=50, freq=FALSE, col="wheat", main="Histogram of Sampled Values", xlab="Sampled Values")
```
Histogram of Sampled Values

Sampled Values

Density

0 10 20 30 40 50

0.00 0.10 0.20