Procesos Estocásticos y Series Temporales

Problemas Propuestos Práctica 1

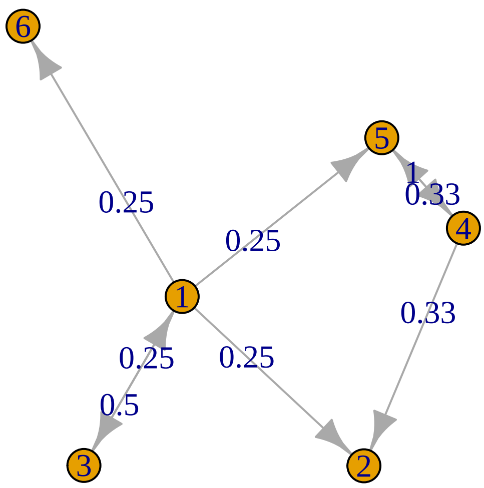
Francisco Javier Mercader Martínez

Problema 1. Para la cadena dada por la siguiente matriz de transición:

$$\begin{bmatrix} 0 & 1/4 & 1/4 & 0 & 1/4 & 1/4 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 1/3 & 1/3 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

1. Dibuja el grafo, encuentra las clases irreducibles y clasifica sus estados.

```r
library('markovchain')
set.seed(1234)

estados <- as.character(1:6) # Definimos los estados
P <- rbind(c(0, 1/4, 1/4, 0, 1/4, 1/4),
           c(0, 1, 0, 0, 0, 0),
           c(1/2, 0, 1/2, 0, 0, 0),
           c(0, 1/3, 0, 1/3, 1/3, 0),
           c(0, 0, 0, 1, 0, 0),
           c(0, 0, 0, 0, 0, 1))

mc <- new("markovchain", states = estados, transitionMatrix = P)

plot(mc)
```



2. Calcula las probabilidades de absorción de los estados absorbentes.

```r
absorptionProbabilities(mc)
```

```
##           2         6
## 1 0.6666667 0.3333333
## 3 0.6666667 0.3333333
## 4 1.0000000 0.0000000
## 5 1.0000000 0.0000000
```

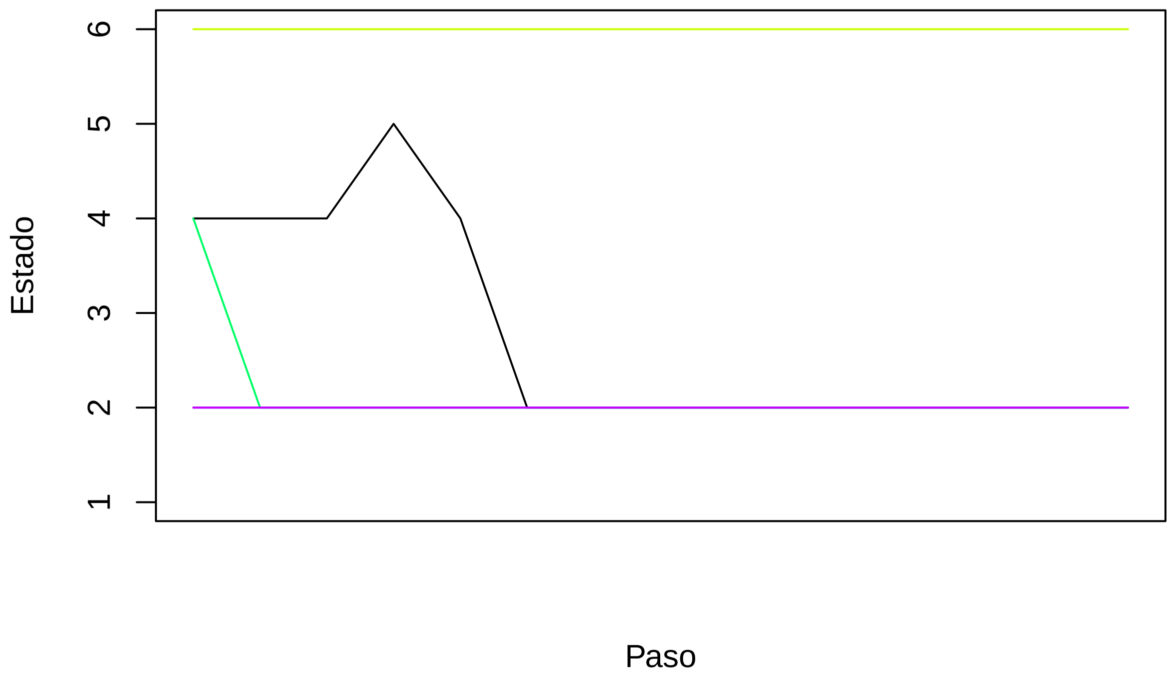
3. Calcular el tiempo medio de absorción por algún estado absorbente.

```r
meanAbsorptionTime(mc)
```

```
##        1        3        4        5
## 3.666667 5.666667 4.000000 5.000000
```

4. Simula y representa cinco trayectorias empezando en un estado al azar. Indica si observas patrones.

```r
set.seed(1234)
n_tray <- 5
n_steps <- 15

# Generar y alamcenar trayectorias (cada fila una trayectoria)
trayectorias <- replicate(n_tray, rmarkovchain(n = n_steps, object = mc), simplify = FALSE)

# Mapear estados a índices numéricos 1..6
state_to_idx <- setNames(1:6, estados)
tray_idx <- lapply(trayectorias, function(x) as.numeric(state_to_idx[x]))

plot(1:n_steps, tray_idx[[1]], type = 'l', ylim = c(1,6), xaxt = 'n',
     xlab = "Paso", ylab = "Estado")
cols <- rainbow(n_tray)
for(i in 2:n_tray){
  lines(1:n_steps, tray_idx[[i]], type = 'l', col = cols[i])
}
```



Problema 2. Un estudio de mercado de una determinada marcad de productos estima que el 20% de la gente que compra un producto un mes, no lo comprará el mes siguiente. Además, el 30% de quienes no lo compren un mes lo adquirirá al mes siguiente.

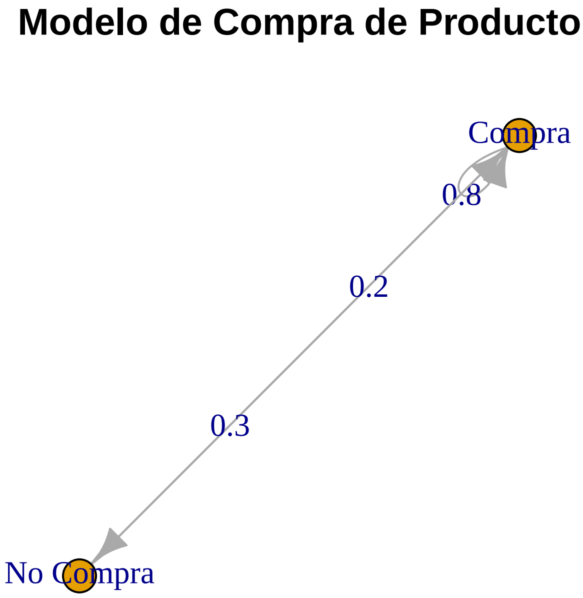
a. Modelar el problema a través de un grafo y determina la matriz de transición

```r
set.seed(1234)
library('markovchain')

# Definir los estados
estados <- c("Compra", "No Compra")

# Matriz de transición
# Filas: estado_actual, Columnas: estado siguiente
P <- rbind(c(0.8, 0.2), # Desde Compra
           c(0.3, 0.7)) # Desde No Compra

# Crear el objeto markovchain
mc <- new('markovchain',
```

```r
         states = estados,
         transitionMatrix = P,
         name = "Cadena de Compra")

# Mostrar la matriz de transición
print(mc)
```

```
##            Compra No Compra
## Compra        0.8       0.2
## No Compra     0.3       0.7
```

```r
# Graficar el modelo
plot(mc, main = "Modelo de Compra de Producto")
```



b. Si un individuo compra el producto el primer mes, ¿cuál es la probabilidad de que lo vuelva a comprar dentro de un año?

```r
library(expm)
# Probabilidad de compra después de 1 año
# Si compra el primer mes, el estado inicial es "Compra"
P_año <- P %^% 12

# La probabilidad está en la posición [1, 1] (de Compra a Compra)
prob_compra_1_año <- P_año[1, 1]

cat("Probabilidad de que vuelva a comprar dentro de un año:",
    round(prob_compra_1_año, 4), "\n")
```

```
## Probabilidad de que vuelva a comprar dentro de un año: 0.6001
```

c. En una población de 1000 individuos, 100 compraron el producto el primer mes. ¿Cuántos lo comprarán el mes próximo? ¿Y dentro de un año?

```r
# Distribución inicial: 100 compran, 900 no compran
distribucion_inicial <- c(100, 900) / 1000

# Mes próximo (multiplicar por P)
distribucion_mes_1 <- distribucion_inicial %*% P
compradores_mes_1 <- distribucion_mes_1[1] * 1000

# Dentro de un año (multiplicar por P^12)
distribucion_año <- distribucion_inicial %*% (P %^% 12)
compradores_año <- distribucion_año[1] * 1000

cat("Compradores el mes próximo:", round(compradores_mes_1), "\n")
```

```
## Compradores el mes próximo: 350
```

```r
cat("Compradores dentro de un año:", round(compradores_año), "\n")
```

```
## Compradores dentro de un año: 600
```

Problema 3. Una urna contiene cuatro bolas rojas y dos verdes. Se van tomando bolas de una en una al azar. Si se obtiene una bola roja, se pinta de verde y se devuelve a la urna. Si se obtiene bola verde se pinta de rojo y se devuelve a la urna. El proceso continúa hasta que no queden bolas rojas en la urna.

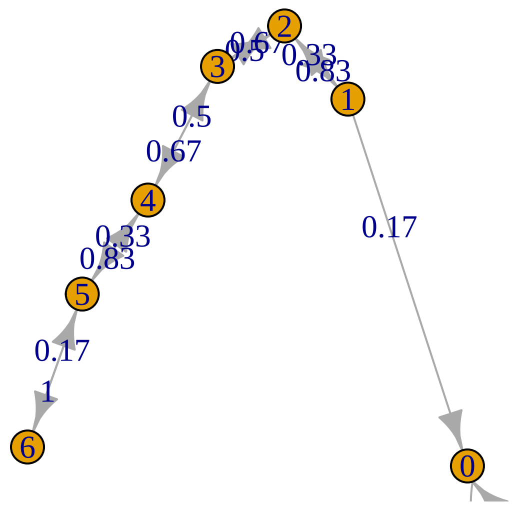
(a) Modelar el problema a través de un grafo y determina la matriz de transición.

```r
set.seed(11)
library('markovchain')

# Define los estados como el número de bolas rojas en la urna (0 a 6)
estados_bolas <- as.character(0:6)

# Inicializa la matriz de transición con ceros
# Las filas representan el estado actual, las columnas el estado siguiente
P_bolas <- matrix(0, nrow = 7, ncol = 7, dimnames = list(estados_bolas, estados_bolas))

# Estado 0 (0 bolas rojas) es un estado absorbente
P_bolas["0", "0"] <- 1

# Definir las transiciones para los estados 1 a 5
# En el estado 'i' (i bolas rojas, 6-i bolas verdes):
# - Si se saca una bola roja (prob = i/6), se pinta de verde y se devuelve: i-1 bolas rojas.
# - Si se saca una bola verde (prob = (6-i)/6), se pinta de rojo y se devuelve: i+1 bolas rojas.
for (i in 1:5) {
   # Transición a i-1 (sacar roja)
   P_bolas[as.character(i), as.character(i-1)] <- i / 6
   # Transición a i+1 (sacar verde)
   P_bolas[as.character(i), as.character(i+1)] <- (6-i) / 6
 }

# Definir la transición para el estado 6 (6 bolas rojas, 0 bolas verdes)
# En el estado 6:
# - Solo se pueden sacar bolas rojas (prob = 6/6 = 1). Se pinta de verde y se devuelve: 5 bolas
↪  rojas.
P_bolas["6", "5"] <- 1

# Crear el objeto markovchain
mc_bolas <- new("markovchain",
                 states = estados_bolas,
                 transitionMatrix = P_bolas,
                 name = "Urna de Bolas")

# Mostrar la matriz de transición
print(mc_bolas)
```

```
##           0         1         2         3         4         5         6
## 0 1.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## 1 0.1666667 0.0000000 0.8333333 0.0000000 0.0000000 0.0000000 0.0000000
## 2 0.0000000 0.3333333 0.0000000 0.6666667 0.0000000 0.0000000 0.0000000
## 3 0.0000000 0.0000000 0.5000000 0.0000000 0.5000000 0.0000000 0.0000000
## 4 0.0000000 0.0000000 0.0000000 0.6666667 0.0000000 0.3333333 0.0000000
## 5 0.0000000 0.0000000 0.0000000 0.0000000 0.8333333 0.0000000 0.1666667
## 6 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 1.0000000 0.0000000
```

```r
# Graficar el modelo
plot(mc_bolas, main = "Grafo de Transición de la Urna de Bolas")
```

Grafo de Transición de la Urna de Bolas

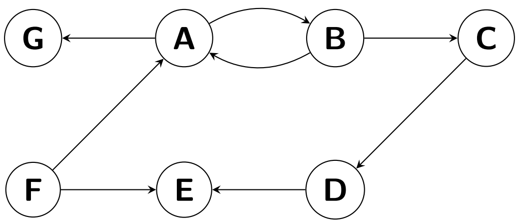


(b) ¿Cuál es la duración media del juego?

```
mean_abs_times <- meanAbsorptionTime(mc_bolas)

# La duración media del juego es el valor correspondiente al estado inicial "4".
duracion_media_juego <- mean_abs_times["4"]
cat("La duración media del juego, comenzando con 4 bolas rojas, es de aproximadamente",
     round(duracion_media_juego, 2), "pasos.\n")
```

```
## La duración media del juego, comenzando con 4 bolas rojas, es de aproximadamente 80.8 pasos.
```

Problema 4. Para un conjunto de páginas web A, B, C, D, E, F y G, calcula el pageRank de cada página e indica en qué orden aparecen las páginas en el buscador para el siguiente grafo de enlaces entre las páginas (utiliza factor de amortiguación $d = 0.85$):




```
library(igraph)

# nodos en el mismo orden del grafo
edges <- c("A B","A G","B A","B C","C D","D E","F A","F E")
g <- graph_from_edgelist(do.call(rbind, strsplit(edges, " ")), directed = TRUE)
d <- 0.85
pr <- page_rank(g, algo = "prpack", directed = TRUE, damping = d)$vector
round(pr, 6)
```

```
##        A        B        G        C        D        E        F
## 0.148865 0.129201 0.129201 0.120843 0.168650 0.237307 0.065933
```

Problema 5. Resolver el apartado 11 b) de la Hoja 3 de problemas.

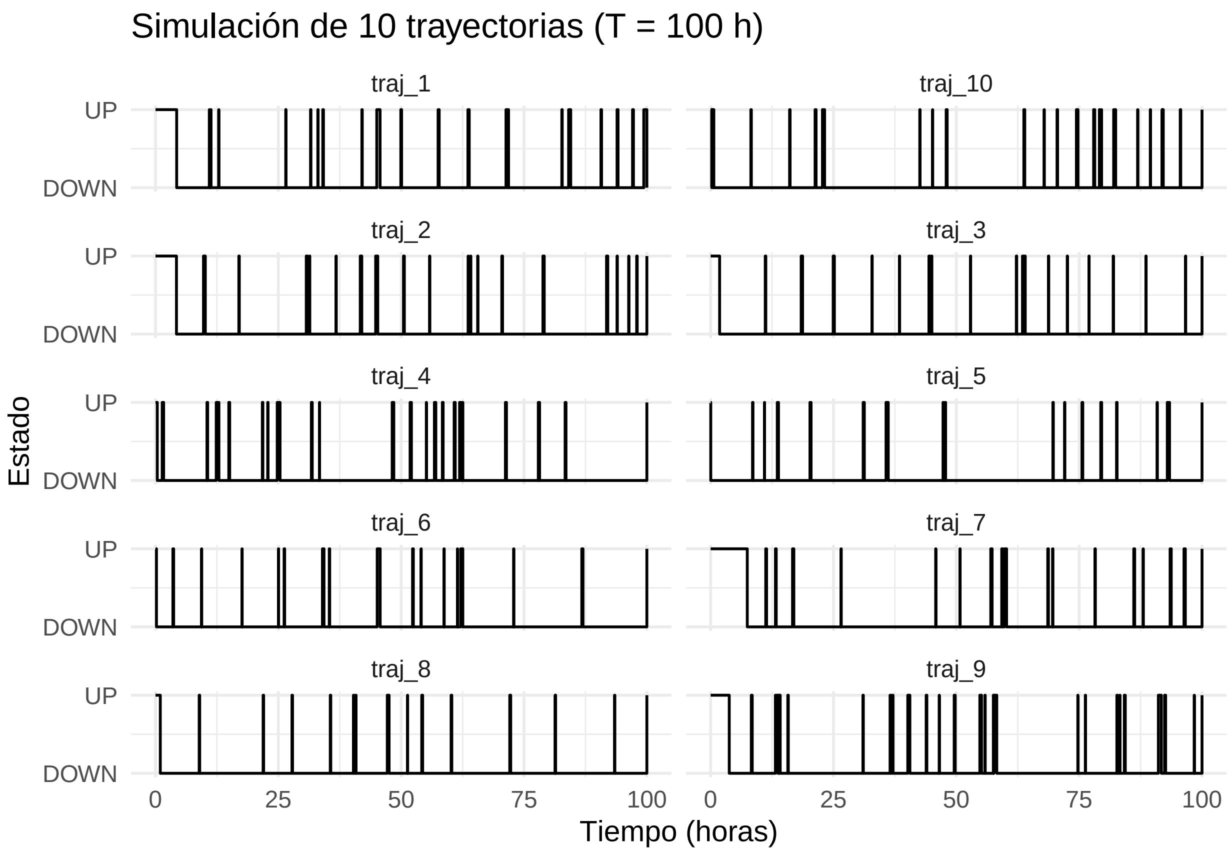
```
library(ggplot2)

set.seed(123)    # reproducibilidad

simulate_trajectory <- function(T = 100,
                                lambda_fail = 1/5,  # tasa de fallo (1/mean uptime)
                                mu_repair = 6) {    # tasa de reparación (1/mean repair time in
                                 ↪  hours)
  times <- c(0)          # instantes donde cambia el estado (inicio de intervalos)
```

```r
  states <- c(1)          # 1 = funcionando, 0 = en reparación
  t_cur <- 0
  state_cur <- 1

  while (t_cur < T) {
    if (state_cur == 1) {
      # tiempo hasta el siguiente fallo
      dur <- rexp(1, rate = lambda_fail)
      next_state <- 0
    } else {
      # tiempo de reparación
      dur <- rexp(1, rate = mu_repair)
      next_state <- 1
    }
    t_next <- t_cur + dur
    # si el siguiente evento supera T, acortamos para terminar en T
    if (t_next >= T) {
      # añadimos el punto final T (estado permanece igual desde t_cur hasta T)
      times <- c(times, T)
      states <- c(states, state_cur)
      break
    } else {
      # evento dentro del horizonte
      times <- c(times, t_next)
      states <- c(states, state_cur)  # el estado en el instante t_next representa el valor hasta
↪   ese tiempo
      # actualizamos
      t_cur <- t_next
      state_cur <- next_state
    }
  }
  # Construir data.frame con los instantes donde definimos el valor (para geom_step)
  # Queremos que en cada tiempo almacenemos el estado *a la derecha* del punto (step)
  data.frame(time = times, state = states)
}

# parámetros
n_trajectories <- 10
T <- 100

# simular todas las trayectorias
trajectories <- lapply(seq_len(n_trajectories), function(i) {
  df <- simulate_trajectory(T = T)
  df$id <- paste0("traj_", i)
  # calcular fracción de tiempo UP (aprox, suma de intervalos)
  df
})

# Convertir a data.frame tipo "largo" y asegurar que cada trayectoria tenga punto inicial y final
↪   T
library(dplyr)
all_traj <- bind_rows(trajectories)

# Para seguridad: si alguna trayectoria no tiene punto final exactamente en T, forzarlo
all_traj <- all_traj %>%
  group_by(id) %>%
  do({
    d <- .
    if (tail(d$time, 1) < T) {
      d <- rbind(d, data.frame(time = T, state = tail(d$state,1), id = unique(d$id)))
    }
    d
  }) %>%
  ungroup()
```

```r
# calcular fracción de tiempo UP para cada trayectoria (exacta a partir de los nodos)
uptime_summary <- all_traj %>%
  group_by(id) %>%
  arrange(time) %>%
  mutate(time_next = lead(time, default = T),
         interval = time_next - time) %>%
  summarise(uptime_fraction = sum(interval[state == 1]) / T,
            uptime_hours = sum(interval[state == 1]),
            downtime_hours = sum(interval[state == 0])) %>%
  arrange(desc(uptime_fraction))

print(uptime_summary)
```

```
## # A tibble: 10 x 4
##    id      uptime_fraction uptime_hours downtime_hours
##    <chr>             <dbl>        <dbl>          <dbl>
##  1 traj_7           0.101         10.1            89.9
##  2 traj_9           0.0815         8.15           91.8
##  3 traj_1           0.0813         8.13           91.9
##  4 traj_2           0.0693         6.93           93.1
##  5 traj_4           0.0462         4.62           95.4
##  6 traj_3           0.0344         3.44           96.6
##  7 traj_10          0.0339         3.39           96.6
##  8 traj_5           0.0255         2.55           97.5
##  9 traj_8           0.0252         2.52           97.5
## 10 traj_6           0.0249         2.49           97.5
```

```r
# Preparar datos para geom_step: geom_step dibuja el valor en cada time y lo mantiene hasta el
↪ siguiente punto
# (ya tenemos puntos en los cambios y el final T)
p <- ggplot(all_traj, aes(x = time, y = state, group = id)) +
  geom_step(direction = "hv") +
  facet_wrap(~ id, ncol = 2) +
  scale_y_continuous(breaks = c(0,1), labels = c("DOWN","UP")) +
  labs(x = "Tiempo (horas)", y = "Estado", title = paste("Simulación de", n_trajectories,
  ↪ "trayectorias (T =", T, "h)")) +
  theme_minimal()

print(p)
```



Simulación de 10 trayectorias (T = 100 h)
traj_1
traj_10
traj_2
traj_3
traj_4
traj_5
traj_6
traj_7
traj_8
traj_9
UP
DOWN
Estado
0
25
50
75
100
Tiempo (horas)