# PE: ENTREGA I

Jon Zorrilla Gamboa

## 1. Ejercicio 1

Se dibuja un cuadrado unitario (de lado 1) con un circulo unitario (de diámetro 1) inscrito. Se generan puntos aleatorios con distribución uniforme dentro del cuadrado.

### 1.1.

**¿Cuál es el universo de eventos de este experimento?**

El universo de eventos es el conjunto de puntos que definiremos a continuación que son suficientes para determinar los posibles sucesos del experimento. Definiremos el conjunto $C = \{m, n\}$, donde $m$ representan los puntos que caen dentro del círculo, y $n$ los que caen fuera del círculo (pero dentro del cuadrado). Como estos puntos se generan de manera aleatoria, definiremos las siguientes funciones:

$$X(m) = 1 \qquad X(n) = 0 \tag{1}$$

De esta manera, podremos cuantificar la cantidad total de puntos que caen dentro del círculo, dividiendo los puntos que caen dentro del círculo entre los puntos totales generados.

### 1.2.

**¿Cuál es la probabilidad que un punto generado esté incluido dentro del círculo?**

Si los puntos se generan de manera aleatoria dentro del cuadrado, la probabilidad de que caigan en un punto específico será $p_{total} = \frac{1}{A_{total}} = \frac{1}{(2r)^2}$. De esta manera, podemos comparar las probabilidades de que el punto caiga en el círculo conociendo el área del círculo: $A_{circ} = \pi r^2$. Así, podemos hallar la probabilidad de que un punto esté incluido en en círculo:

$$p_{circ} = p_{total} \cdot A_{circ} = \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4} \approx 0{,}785398 \tag{2}$$

Que, como podemos ver, obtenemos una probabilidad menor a 1, como era de esperar.

## 1.3.

**Usar la respuesta del punto ii) para escribir un programa que calcule $\pi$ con tres cifras significativas. Entregar el código (es corto...), el valor conseguido (primera cinco cifras decimales), explicar el criterio de parada que se ha usado para determinar cuando terminar el cálculo (sin usar el valor de $\pi$, claro...) y el número de puntos que se han generado.**

Por la ley de los grandes números, sabemos que si $p$ es la media muestral, esta tenderá al valor verdadero a medida se usen más muestras; es decir $\hat{p} \to p$. Por ello, podemos calcular el error que cometeremos al realizar este cálculo para asegurar que podemos representar hasta 3 cifras significativas de $\pi$. Por el teorema central del límite, la distribución se comportará como una normal:

$$S_n \equiv \mathcal{N}(p, \frac{\sigma^2}{n}) \tag{3}$$

donde, debido a que la distribución es una distribución de Bernoulli, la varianza será $Var[X] = \sigma^2 = p(1-p)$. Por ello, los intervalos de confianza para un valor de $\alpha$ determinado son los siguientes:

$$(S_n - z_{1-\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}}, S_n - z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}}) \tag{4}$$

Pero no conocemos el valor de $\sigma^2$. Por ello, consideraremos también el estimador de la varianza, sustituyendo:

$$\sigma^2 = s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - S_n)^2 \tag{5}$$

Así, podemos acotar el error debido a la aproximación, con probabilidad 1 - $\alpha$, por:

$$\epsilon = 2 z_{1-\frac{\alpha}{2}} \frac{s}{\sqrt{n}} \tag{6}$$

En este caso, un intervalo de confianza de 99 % será suficiente. Además, como queremos una precisión de 3 cifras decimales, deberemos conseguir que el error sea menor a $10^{-3}$. En este caso, debido a las aproximaciones que hemos realizado, deberemos fijar un error menor, por ejemplo $10^{-4}$. Entonces, realizaremos la simulación creando distribuciones aleatorias uniformes en los intervalos [0,1] para ambos ejes, y veremos si estos puntos caen dentro del área del círculo o no. Luego, contaremos las que han caído dentro del círculo, y, multiplicando por 4 y teniendo en cuenta el error cometido, podremos aproximar $\pi$.

```
import numpy as np
from scipy.stats import norm


np.random.seed(12345)
s = 0
```

```
def sample_points():
    x = np.random.uniform(0.0, 1.0, 1)
    y = np.random.uniform(0.0, 1.0, 1)

    return 1 if ((x-0.5)**2 + (y-0.5)**2 <= 0.5**2) else 0

def main(alpha, epsilon):

    sn = sample_points()
    z_alpha = norm.ppf(1 - alpha/2)
    n = 1
    s = 0

    while n:
        value = sample_points()
        s = s + (n-1)*((value - sn)**2)/n
        sn = sn + (value - sn) / n
        n += 1

        #Check stopping criteria
        if s!= 0:
            eps = 2 * z_alpha * (s/(n-1)) / np.sqrt(n)
            #print(eps)
            if eps < epsilon:
                break
    return 4*sn, n

pi, n = main(alpha = 0.01, epsilon=5e-4)
```

Con esto, obtenemos un valor aproximado de $\pi = 3{,}1416936376906053$ para $n = 3015274$ iteraciones. Como podemos ver, hemos obtenido un valor con una precisión de $10^{-3}$ para esta semilla determinada, pero para otra semilla diferente, podríamos no haber obtenido lo mismo. Esto es porque la única condición que hemos tomado es tener un error absoluto menor a $5 \cdot 10^{-4}$ para el 99 % de los casos, pero debido a las

aproximaciones, no es seguro obtener un resultado tan preciso para $\pi$. Además, si queremos obtener una precisión, el tiempo de cálculo se dispara, por lo que podemos asegurar que este no es el mejor método para realizar este cálculo de $\pi$.

# 2. Ejercicio 2

**Como ejercicio de probabilidad, este es muy fácil, pero es muy gracioso, me gusta, y quiero preguntarlo. Se trata del juego de las tres puertas, también conocido como el Monty Hall problem. Hay tres puertas, y detrás de una de ellas hay un premio. El presentador os pide elegir una, sin abrirla. Después, abre una de las otras dos, y os muestra que allí no está el premio. Finalmente os pide si queréis cambiar la puerta o confirmar la elección inicial. ¿Cuál es la decisión que maximiza la probabilidad de ganar, y que probabilidad de ganar os da?**

La respuesta a este problema es muy sencilla pese a que puede crear muchos rompecabezas. En mi opinión, la mejor manera de entender la manera óptima para saber cual es la decisión que maximiza la probablidad de ganar, es representando todas las opciones. Para ello, haremos la siguiente tabla:

| Elección inicial | ¿Cambiamos de puerta? | Resultado |
|---|---|---|
| Premio | Sí | Perdemos |
| Premio | No | Ganamos |
| No premio | Sí | Ganamos |
| No premio | No | Perdemos |
| No premio | Sí | Ganamos |
| No premio | No | Perdemos |

**Tabla *1*:** Resultados en función de si cambiamos de puerta o nos mantenemos.

Como podemos observar en esta tabla, si elegimos no cambiar de puerta, conseguiremos el premio 1 de cada 3 veces, es decir, con una probabilidad del $33.\widehat{3}\,\%$ de las veces. En cambio, si decidimos cambiar de puerta, ganaremos un $66.\widehat{6}\,\%$ de las veces. La explicación, pese a que pueda parecer contraintuitiva, es sencilla. Si decidimos quedarnos con la puerta que hemos elegido en primera instancia, siempre tendremos un $33.\widehat{3}\,\%$ de probabilidad de llevarnos el premio, independientemente de que el presentador nos enseñe una puerta vacía. En cambio, una vez se ha enseñado esta puerta, si decidimos cambiar de puerta es como si estuviésemos eligiendo 2 de las 3 puertas de manera simultánea, por lo que tendremos una probabilidad mayor.

# 3. Ejercicio 3

Dada una cadena de Markov con un conjunto finito de estados 1, ... , M, sea $p_m$(t) la probabilidad que la cadena esté en el estado m al instante t. Se defina el vector: $p(t) = [p_1(t), ..., p_m(t)]'$.

## 3.1.

**Usar la definición de la matriz de transición P para demostrar que: $p'(t+1) = p'(t)P$, y por tanto, si $\lambda$ es la distribución inicial, $p'(t) = \lambda' P^t$**

La matriz de transición da información en cada elemento de la probabilidad de pasar de un estado a otro, es decir:

$$P = \begin{pmatrix} p_{11} & p_{12} & ... & p_{1n} \\ p_{21} & p_{22} & ... & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m0} & p_{m2} & ... & p_{mn} \end{pmatrix} \tag{7}$$

donde $p_{ij}$ es la probabilidad de pasar de una posición $i$ a una posición $j$. Si hacemos el producto entre el vector $p(t)$ y la matriz de transición:

$$p(t)'P = (p_1(t), ..., p_m(t)) \cdot \begin{pmatrix} p_{11} & p_{12} & ... & p_{1n} \\ p_{21} & p_{22} & ... & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m0} & p_{m2} & ... & p_{mn} \end{pmatrix} = \left( \sum_{j=1}^{n} p_{j1}(t)p_j(t), ..., \sum_{j=1}^{n} p_{jm}(t)p_j(t) \right) \tag{8}$$

Donde, cada elemento del vector resultante hace referencia a la probabilidad de encontrarnos en el estado $i$ en un tiempo $t+1$, pues es el producto de la suma de todas las probabilidades de transición hasta el i-ésimo estado y la probabilidad de estar en cada uno de ellos en un tiempo $t$. Entonces si representamos la probabilidad de estar en el i-ésimo estado en un tiempo $t$:

$$P = \sum_{j=1}^{n} p_{ji}p_j(t) = [p(t)'P]|_i \tag{9}$$

Es decir, es el i-ésimo elemento del vector. Entonces,

$$[p(t+1)']|_i = p(i, t+1) = \sum_{j+1}^{n} p(i, t+1|j, t)p(j, t) = \sum_{j=1}^{n} p_{ji}p_j(t) = [p(t)'P]|_i \tag{10}$$

Luego, se demuestra $[p(t+1)']|_i = [p(t)'P]|_i$.

Ahora, si $\lambda$ es la distribución original:

$$p(t)' = p(t-1)'P = p(t-2)'PP = ... = p(0)'P^t = \lambda' P^t \tag{11}$$

# 4. Ejercicio 4

Consideremos la siguiente cadena de Markov: Se asuma que cada estado tiene un *self-loop* con la

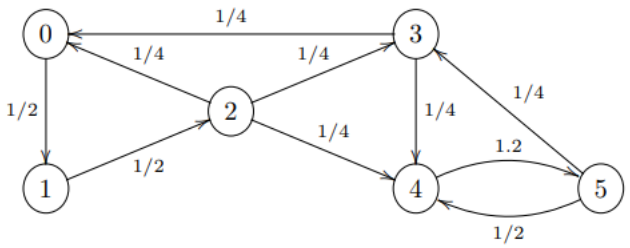


**Figura 1:** Cadena de Markov

probabilidad necesaria para que la suma de las probabilidades en salida sea 1. Escribir un programa (son como mucho 10 líneas en python... nada de que preocuparse) para contestar a las preguntas siguientes:

## 4.1.

**Considerando $\lambda = [1, 0, 0, 0, 0, 0]'$ (es decir: se empieza a $t = 0$ en el estado 0), calcular p(2), p(10), p(100)**

## 4.2.

**Repetir el cálculo de p(2), p(10), p(100) con $\lambda = [0, 0, 1, 0, 0, 0]'$.**

## 4.3.

**Repetir el cálculo de p(2), p(10), p(100), con $\lambda = [0, 0, 0, 0, 0, 1]'$**

Responderemos a las tres preguntas anteriores a continuación:

En primer lugar, escribimos la matriz de transición P:

$$P = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 \\ \frac{1}{4} & 0 & 0 & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{2} \end{pmatrix} \tag{12}$$

Como podemos ver, todas las filas suman 1, así que podemos suponer que está bien escrita. Si, por ejemplo $\lambda = [1, 0, 0, 0, 0, 0]'$, empezamos en el estado 0 en $t = 0$. Hacemos el siguiente programa en python para hallar las probabilidades p(2), p(10), p(100) para todos los $\lambda s$.

```
import numpy as np
np.set_printoptions(precision=5)

P = np.array([[0.5, 0.5, 0, 0, 0, 0],
              [0, 0.5, 0.5, 0, 0, 0],
              [0.25, 0, 0.25, 0.25, 0.25, 0],
              [0.25, 0, 0, 0.5, 0.25, 0],
              [0, 0, 0, 0, 0.5, 0.5],
              [0, 0, 0, 0.25, 0.5, 0.25]])

lambda_1 = np.array([1, 0, 0, 0, 0, 0])
lambda_2 = np.array([0, 0, 1, 0, 0, 0])
lambda_3 = np.array([0, 0, 0, 0, 0, 1])

t1 = 2
t2 = 10
t3 = 100

print("El resultado para cada par vector inicial y p iteraciones:")
for item in [lambda_1, lambda_2, lambda_3]:
    for t in [t1, t2, t3]:
        res = item @ np.linalg.matrix_power(P, t)
        print(f"Inicio: ({item}), p=({t}):", res)
```

Y, el resultado que se obtiene para todos los estados iniciales y todas las iteraciones que pide el enunciado es:

```
El resultado para cada par vector inicial y p iteraciones:
Inicio: ([1 0 0 0 0 0]), p=(2): [0.25 0.5  0.25 0.   0.   0.  ]
Inicio: ([1 0 0 0 0 0]), p=(10): [0.13752 0.15517 0.11391 0.14052 0.27938 0.1735 ]
Inicio: ([1 0 0 0 0 0]), p=(100): [0.11111 0.11111 0.07407 0.14815 0.33333 0.22222]
Inicio: ([0 0 1 0 0 0]), p=(2): [0.25   0.125  0.0625 0.1875 0.25   0.125 ]
Inicio: ([0 0 1 0 0 0]), p=(10): [0.12096 0.12722 0.08841 0.14542 0.31359 0.2044 ]
Inicio: ([0 0 1 0 0 0]), p=(100): [0.11111 0.11111 0.07407 0.14815 0.33333 0.22222]
Inicio: ([0 0 0 0 0 1]), p=(2): [0.0625 0.     0.     0.1875 0.4375 0.3125]
```

```
Inicio: ([0 0 0 0 0 1]), p=(10): [0.1022  0.09632 0.0607  0.15069 0.35148 0.23862]
Inicio: ([0 0 0 0 0 1]), p=(100): [0.11111 0.11111 0.07407 0.14815 0.33333 0.22222]
```

## 4.4.

**¿Cómo cambian los tres vectores p(2), p(10), p(100)?, ¿Cómo cambia su dependencia de las condiciones iniciales?, ¿Qué impica esto para p($\infty$)?**

En el primer caso, para $\lambda = [1, 0, 0, 0, 0, 0]'$, a medida aumenta el número de iteraciones, existen diferentes probabilidades de estar en diferentes estados finales. Para $p = 2$, solo podemos estar en uno de los estados $0, 1$ o 2. En cambio, para $p = 10$, ya existe una probabilidad no nula de estar en cualquiera de los estados del sistema. Para $p = 100$, se reduce mucho la probabilidad de estar en los estados $0, 1$ y 2 previamente mencionados, y la probabilidad máxima es para el estado 4.

Para el segundo caso, $\lambda = [0, 0, 1, 0, 0, 0]'$, para $p = 2$ ya existe probabilidad no nula de estar en cualquier estado. Esto es porque partimos de un estado inicial más central, y llega al resto de estados con mayor facilidad. Para $p = 10$, las probabilidades empiezan a tener la pinta que acaban teniendo para $p = 100$, que da lugar a las mismas probabilidades que para el caso inicial.

Por último, estudiamos el caso $\lambda = [0, 0, 0, 0, 0, 1]'$. Para $p = 2$, hay probabilidad nula de estar en los estados 1 y 2, pues aún no existen iteraciones suficientes para llegar a esos estados finales. Para $p = 10$, empieza a tener la misma forma que para $p = 100$, como en el caso anterior.

Visto esto, en todos los casos, para $p = 100$ llegamos a la misma situación, por lo que podemos decir que estamos en un sistema en equilibrio a partir de estas iteraciones independientemente de donde hemos empezado. Con esto, podemos estimar que estas probabilidades se mantendrán fijas para cuando $p \to \infty$.