SISTEMAS ESTOCÁSTICOS 2023-2024

Parcial 2

Autor: Álvaro José Álvarez Arranz

Importación de las funciones

En esta sección se importarán las funciones a usar y la iniciación de la semilla `seed` para la reproducción de experimentos.

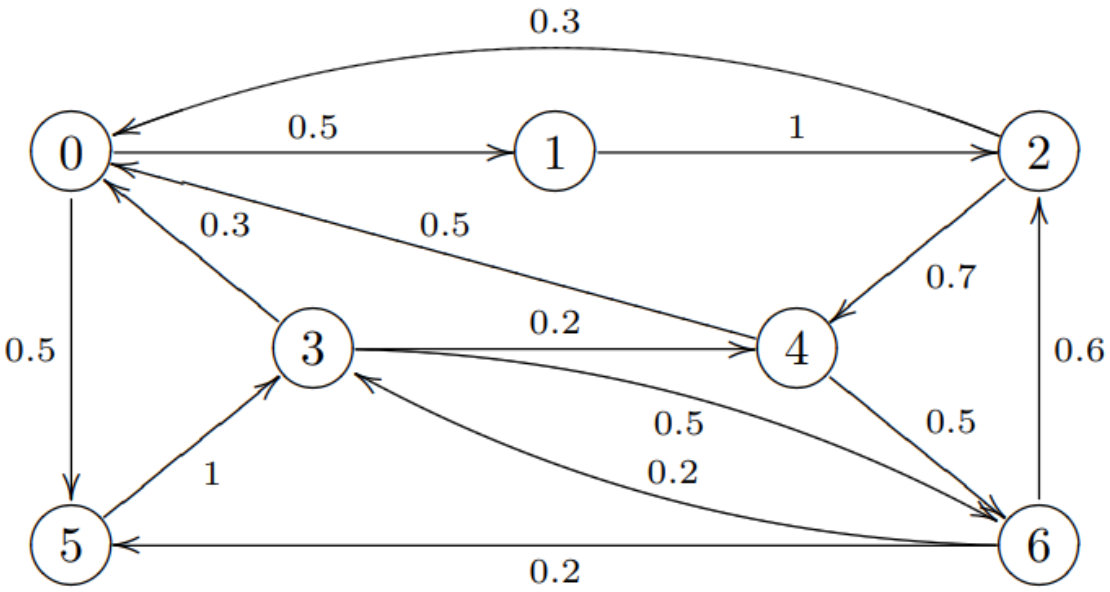
In [1]:
```
import numpy as np
import matplotlib. pyplot as plt
import random
from random import randint, choices
from sympy import symbols, Eq, Matrix, solve

random.seed(1234)
np.random.seed(1234)
```

1.- Cadenas de Markov a tiempo continuo

Planteamiento

Consideramos la siguiente cadena de markov a tiempo continuo. Los tiempos de permanencia en cada estado tienen una distribución exponencial con un parámetro λ_i para el estado i, donde

$$\Lambda = [\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6]' = [2.0, 0.1, 1.8, 4.0, 2.5, 0.8, 1.5]'$$



En esta cadena buscaremos la distribución de probabilidad estacionaria utilizando tres métodos.

Establecimiento de cadena y distribución

A continuación vamos a poner el código para las exponenciales y la cadena de Markov del enunciado.

In [2]:
```
# Inicialización de tiempos y cadena
Lambdas = [2.0, 0.1, 1.8, 4.0, 2.5, 0.8, 1.5]

Markov = np.array([[0, 0.5, 0, 0, 0, 0.5, 0], # Nodo 0
                   [0, 0, 1, 0, 0, 0, 0], # Nodo 1
                   [0.3, 0, 0, 0, 0.7, 0, 0], # Nodo 2
                   [0.3, 0, 0, 0, 0.2, 0, 0.5], # Nodo 3
                   [0.5, 0, 0, 0, 0, 0, 0.5], # Nodo 4
                   [0, 0, 0, 1, 0, 0, 0], # Nodo 5
                   [0, 0, 0.6, 0.2, 0, 0.2, 0]]) # Nodo 6
```

Método I

Este método se basa en la relación con la *jump chain*. Asumiendo que la cadena a tiempo discreto es ergódica, se hará una simulación con un tiempo lo suficientemente largo como para poder estimar la probabilidad de ocupación de la cadena discreta de una manera independiente del estado inicial. Sea $\tilde{\pi}_i$ la probabilidad límite de ocupación del estado i en la cadena discreta. Entonces, las probabilidades límite de ocupación en la cadena a tiempo continuo son

$$\pi_i \sim \frac{\pi_i}{\lambda_i}$$

Siendo necesario normalizar.

Primero vamos a inicializar algunas de las variables que necesitaremos

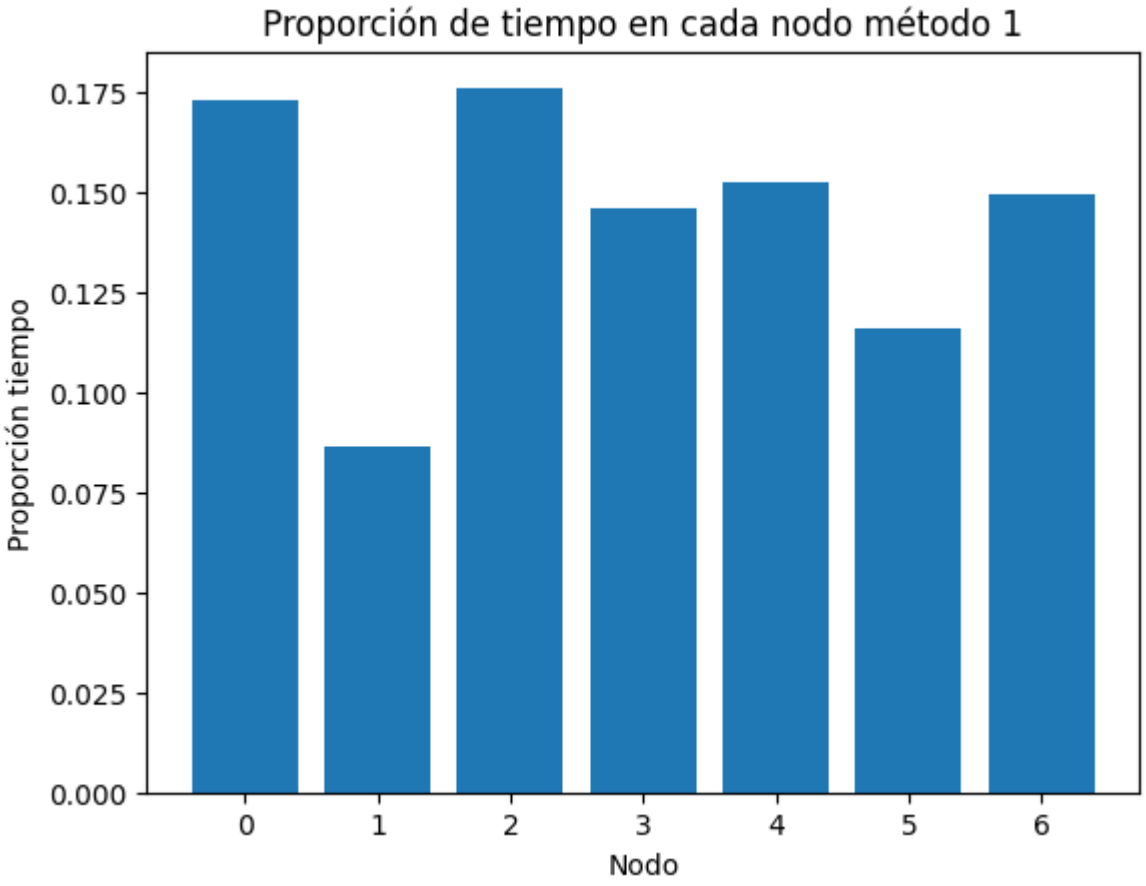
In [3]:
```
# Definimos el número de pasos que se darán en la cadena de Markov
num_pasos = 10000

# Definición del número de experimentos
num_exp = 100
```

Ahora vamos a ver cómo se comporta en los diversos paseos.

In [4]:
```
# Iniciamos el array del tiempo que estamos en cada nodo durante el experimento
t_total_m1 = np.zeros(len(Markov))

for _ in range(num_exp):
  #Iniciamos el nodo en el que empezamos
  curr_node = np.random.randint(0, len(Markov))

  # Incrementamos en uno el nodo en el que empezamos
  t_total_m1[curr_node] += 1

  # Caminamos por la cadena
  for _ in range(num_pasos):
    # Elegimos el nodo siguiente al que vamos a ir
    curr_node = np.random.choice(range(Markov.shape[1]), p=Markov[curr_node])

    # Incremento de contador
    t_total_m1[curr_node] += 1

t_prop_m1 = t_total_m1 / np.sum(t_total_m1)
```

In [5]:
```
plt.bar(range(len(t_prop_m1)), t_prop_m1)
plt.title("Proporción de tiempo en cada nodo método 1")
plt.xlabel("Nodo")
plt.ylabel("Proporción tiempo")
```

Out[5]:
```
Text(0, 0.5, 'Proporción tiempo')
```



Podemos observar que el tiempo en el que estamos en cada nodo es prácticamente la misma salvo para el nodo 1. Esto se debe a que todos los nodos tienen dos o más caminos que llevan a ellos, salvo al 1 que solo hay una forma en la que podemos llegar a él.

Método II

En este método se hará uso de una simulación, al igual que en el método anterior, pero considerando los tiempos de permanencia. Si $P[i]$ es el tiempo que se pasa en el estado i, entonces cada vez que la cadena está en i se genera el tiempo de permanencia $T \sim exp(\lambda)$ y se añade a $P[i]$ antes de cambiar de estado. Al final de la simulación, $P[i]$ contiene el tiempo total que se ha pasado en el estado i. Normalizando se consigue una estimación de la probabilidad π_i.

Al igual que hicimos en el método anterior, inicializamos algunas variables.

In [6]:
```
# Definimos el tiempo que se paseará por la cadena de Markov
Tmax = 10000

# Definición del número de experimentos
num_exp = 100
```

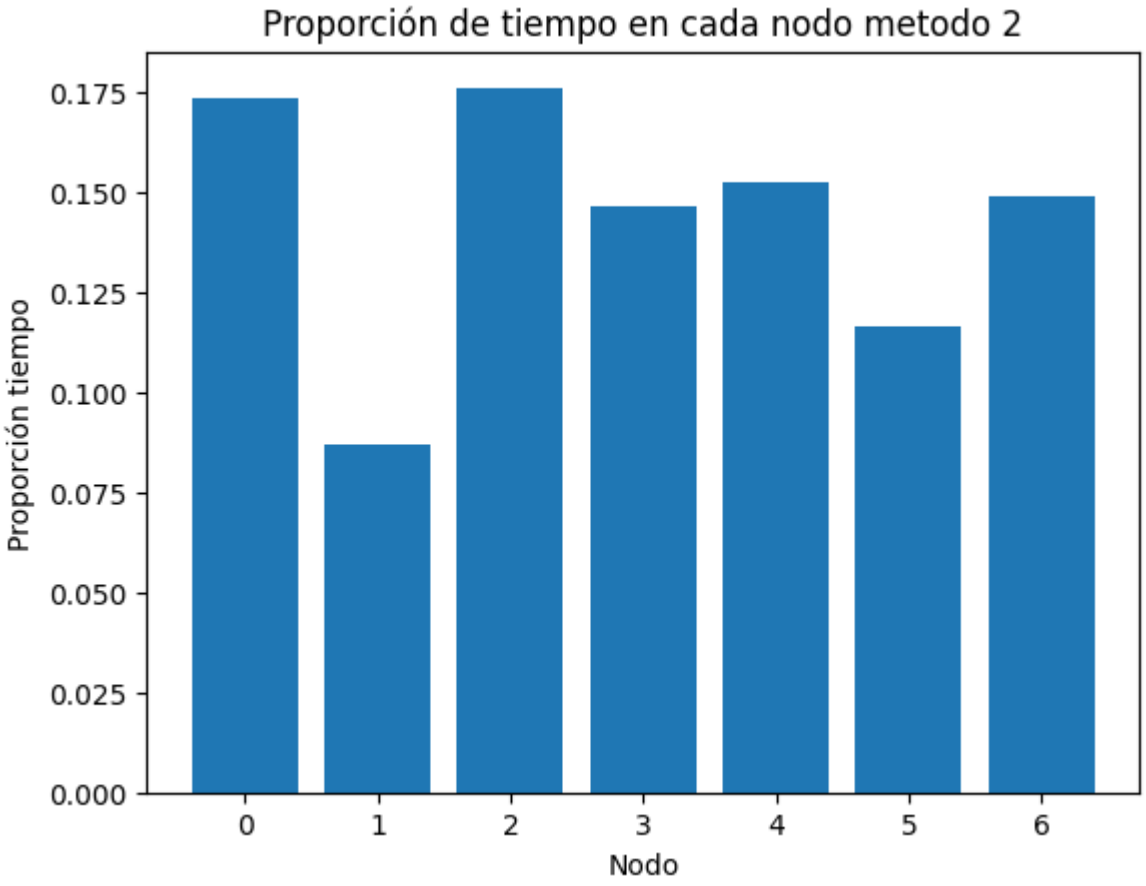
Ahora vamos a programar el experimento.

In [7]:
```
# Iniciamos el tiempo que vamos a estar en cada nodo
```

```
t_total_m2 = np.zeros(len(Markov))

for _ in range(num_exp):
  #Iniciamos el nodo en el que empezamos
  curr_node = np.random.randint(0, len(Markov))

  # Iniciamos los tiempos del experimento
  t_exp = np.zeros(len(Markov))

  # Caminamos por la cadena mientras el tiempo no se cumpla
  while True:
    # Elegimos el nodo siguiente al que vamos a ir
    next_node = np.random.choice(range(Markov.shape[1]), p=Markov[curr_node])

    # Generamos el tiempo de permanencia con la exponencial
    estancia = np.random.exponential(1 / Lambdas[curr_node])

    # Acumulamos el tiempo
    t_exp[curr_node] += 1

    # Actualizamos al proximo nodo
    curr_node = next_node

    # Verificamos si el tiempo ha pasado
    if np.sum(t_exp) >= Tmax:
      break

  t_total_m2 += t_exp

t_prop_m2 = t_total_m2 / sum(t_total_m2)
```

In [8]:
```
plt.bar(range(len(t_prop_m2)), t_prop_m2)
plt.title("Proporción de tiempo en cada nodo metodo 2")
plt.xlabel("Nodo")
plt.ylabel("Proporción tiempo")
```

Out[8]:
```
Text(0, 0.5, 'Proporción tiempo')
```



Podemos observar que el tiempo de estancia de cada cada nodo es similar al caso anterior. En este caso, se debe a que el tiempo de estancia depende del valor de cada λ_i además de los caminos que llevan a cada nodo.

Método III

En este caso se hará una estimación de conjunto. Se definen $N = 1\ 000$ cadenas distintas. Se elige un tiempo $Tmax = 10\ 000$, que es lo suficientemente grande, y se ejecuta una simulación como en el *Método II* por cada una de las N cadenas, terminando la simulación cuando el tiempo simulado llega a $Tmax$. Por cada cadena, se apunta el estado en que la cadena se encuentra al tiempo $Tmax$. El histograma normalizado de estos datos proporcionala estimación de π_i.

Al igual que el en casi anterior, vamos a empezar inicializando algunos valores.

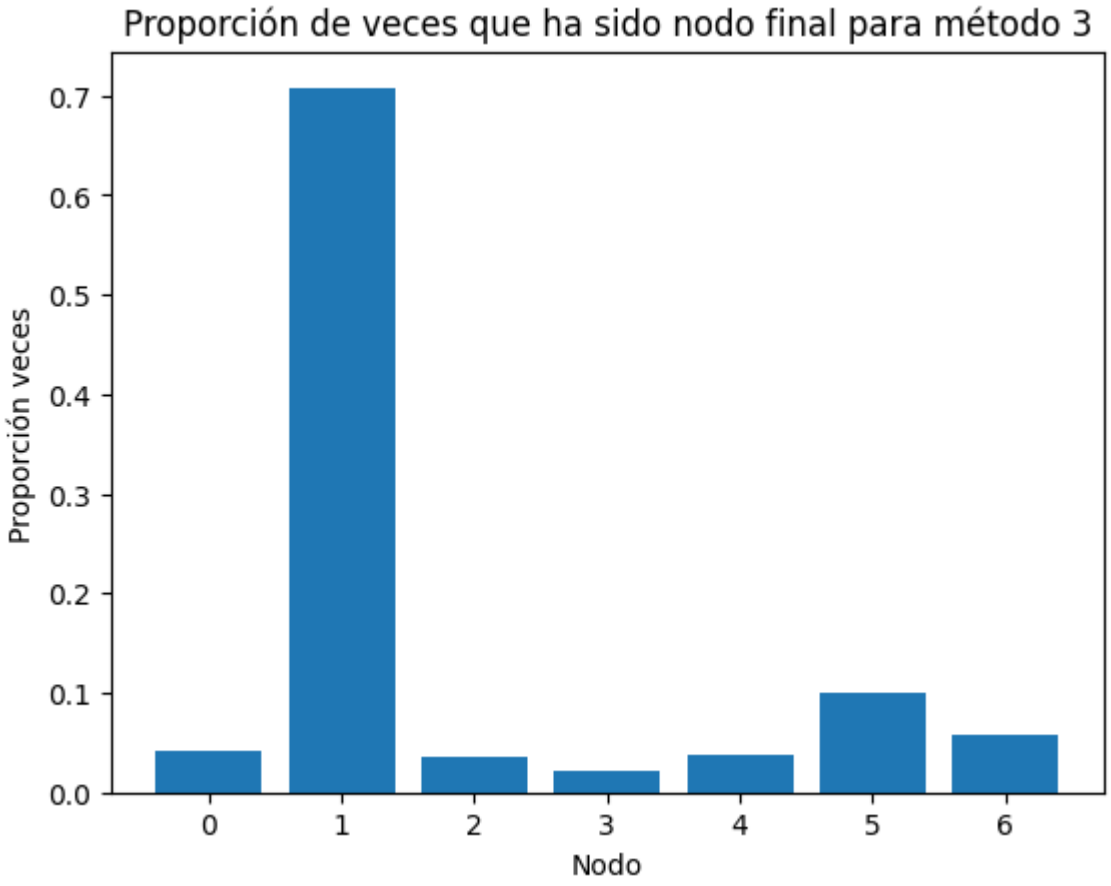
In [9]:
```
# Definimos el tiempo que se paseará por la cadena de Markov
Tmax = 10000

# Definición del número de cadenas a crear
n_chains = 1000

# Obtenemos el número de nodos para cada cadena
n_nodes = len(Markov)
```

Ahora vamos con la implementación del código según las especificaciones

In [10]:
```
# Declaramos el array de estancias finales de cada nodo
t_total_m3 = np.zeros(len(Markov))

# Realizamos la simulación para cada cadena
for _ in range(n_chains):
  # Generamos una nueva cadena
  Markov = np.random.rand(n_nodes, n_nodes)

  # Aseguramos que cada fila sume 1
  Markov = Markov/Markov.sum(axis=1, keepdims=True)

  # Establecemos el nodo en el que iniciamos en la cadena
  curr_node = np.random.randint(0, len(Markov))

  # Iniciamos el tiempo de estancia en la cadena
  t_chain = 0.0

  # Mientras no se cumpla el tiempo
  while True:
    # Elegimos el nodo siguiente al que vamos a ir
    next_node = np.random.choice(range(Markov.shape[1]), p=Markov[curr_node])

    # Generamos el tiempo de permanencia con la exponencial
    estancia = np.random.exponential(1 / Lambdas[curr_node])

    # Sumamos el tiempo en el que estamos en la cadena
    t_chain += estancia

    # Comprobamos si se ha cumplido el tiempo
    if t_chain >= Tmax:
      break

    # Cambiamos al siguiente nodo
    curr_node = next_node

  # Actualizamos el contador del nodo final de la cadena
  t_total_m3[curr_node] += 1

t_prop_m3 = t_total_m3 / n_chains
```

In [11]:
```
plt.bar(range(len(t_prop_m3)), t_prop_m3)
plt.title("Proporción de veces que ha sido nodo final para método 3")
plt.xlabel("Nodo")
plt.ylabel("Proporción veces")
```

Out[11]:
```
Text(0, 0.5, 'Proporción veces')
```



Podemos observar en la gráfica anterior, que las cadenas de markov tienden a terminar el recorrido en el nodo 1. Esta gráfica no es comparable con las obtenidas en los otros dos métodos, puesto que se están midiendo cosas completamente diferentes.

Conclusión

Vamos a comprobar los valores de $\tilde{\pi}_i$ de la cadena de Markov en la distribución estacionaria de la *jump chain*.

Para ello, hay que hacer la siguiente equación:

$$(\tilde{\pi}_0\ \tilde{\pi}_1\ \tilde{\pi}_2\ \tilde{\pi}_3\ \tilde{\pi}_4\ \tilde{\pi}_5\ \tilde{\pi}_6) \begin{bmatrix} 0 & 0.5 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0.3 & 0 & 0 & 0 & 0.7 & 0 & 0 \\ 0.3 & 0 & 0 & 0 & 0.2 & 0 & 0.5 \\ 0.5 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.6 & 0.2 & 0 & 0.2 & 0 \end{bmatrix} = (\tilde{\pi}_0\ \tilde{\pi}_1\ \tilde{\pi}_2\ \tilde{\pi}_3\ \tilde{\pi}_4\ \tilde{\pi}_5\ \tilde{\pi}_6)$$

Dado que las estimaciones de los métodos 1 y 2 son prácticamente las mismas, podemos estimar los valores de $\pi_0, \ldots, \pi_6$ obteniendo los valores de `t_prop_m2`, por ejemplo, obteniendo los siguientes valores:

In [12]:
```
t_prop_m2
```

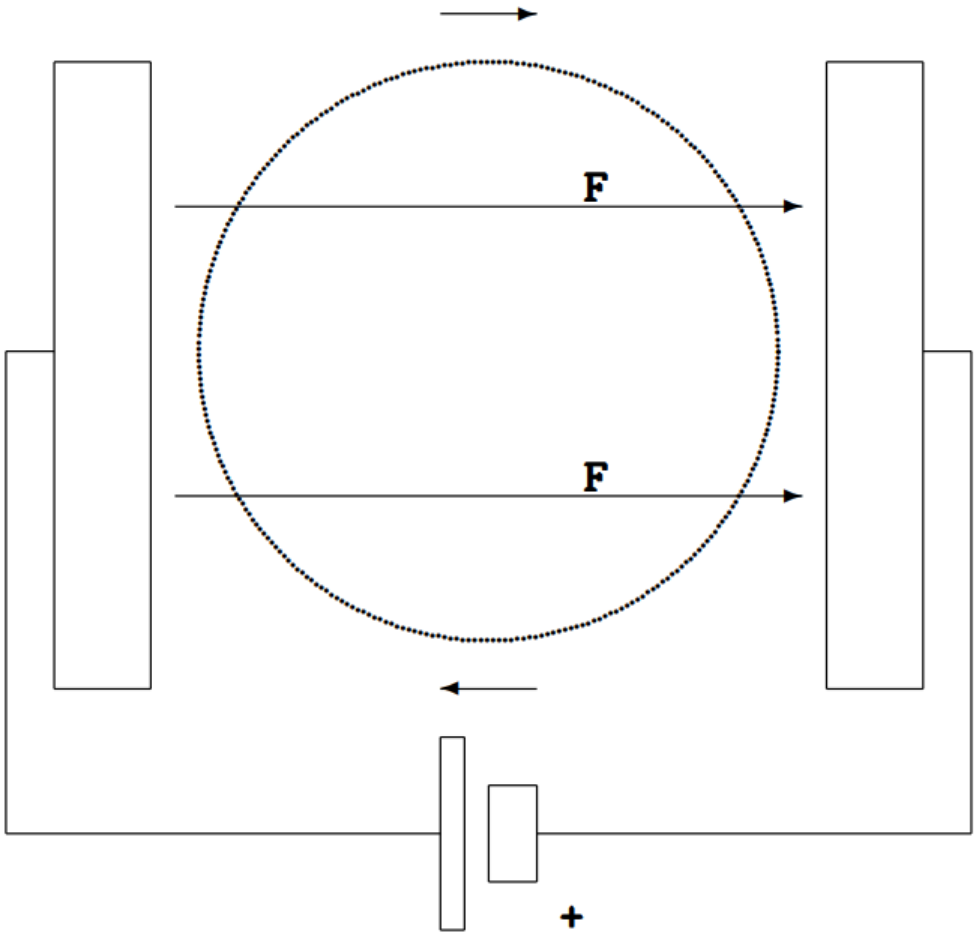
Out[12]:
```
array([0.173315, 0.086749, 0.175981, 0.14618 , 0.152474, 0.116452,
       0.148849])
```

En el caso del *método III* no se está obteniendo los mismos resultados al estár haciendo cadenas diferentes con probabilidad de pasar del nodo i al nodo j diferentes, por lo que los valores $[\pi_0, \ldots \pi_6]$ serán completamente diferentes, aunque se tenga los mismos valores de λ_i.

2.- Movimiento Browniano en un campo eléctrico

Planteamiento

Consideremos un electrón que se mueve en un líquido con colisiones con los átomos del líquido que genera un movimiento Browniano. El sistem está compuesto por un plato que se mueve con movimiento rotatorio con una fracuencia ν en un campo eléctrico, como el esquema siguiente:



El electrón está sujeto a dos fuerzas: un campo eléctrico que, en el sistema de referencia del

plato, en el que nos pondremos, rueda con frecuencia ν, y las colisiones que generan el movimiento Browniano. Si $\mathbf{X} = [x, y]'$ es la posición del electrón (en un sistema de referencia fijado al plato), entonces su movimiento es descrito por una ecuación diferencial estocástica

$$d\mathbf{X} = \mathbf{F}(t)dt + d\mathbf{W}(t)$$

donde $F(t) = [cos(2\pi\nu t), sin(2\pi\nu t)]'$ y $d\mathbf{W}$ es el diferencial de un proceso de Wiener bidimensional con componentes independientes.

Aproximar la ecuación en una cuadrícula regular con $\Delta t = 0.1$, crear una trayectoria de este movimiento y dibujarla en los casos $\nu = 0,\ \nu = 1,\ \nu = 1/10,\ \nu = 1/100$ por $t \in [0, 100]$. Vamos a considerar que el plato es muy grande respecto al recorrido, por lo que asumimos que la partícula no llega en ningún momento al límite del plato.

Ahora, con las mismas frecuencias ν, se realizarán 100 experimentos, de los que se obtendrán 100 trayectorias, de las que obtendremos media y varianza de los procesos. Se dibujará la trayectoria media y compararemos cómo varía la varianza con la frecuencia.

Realización de los experimentos

Para la realización del experimento, vamos ir paso a paso.

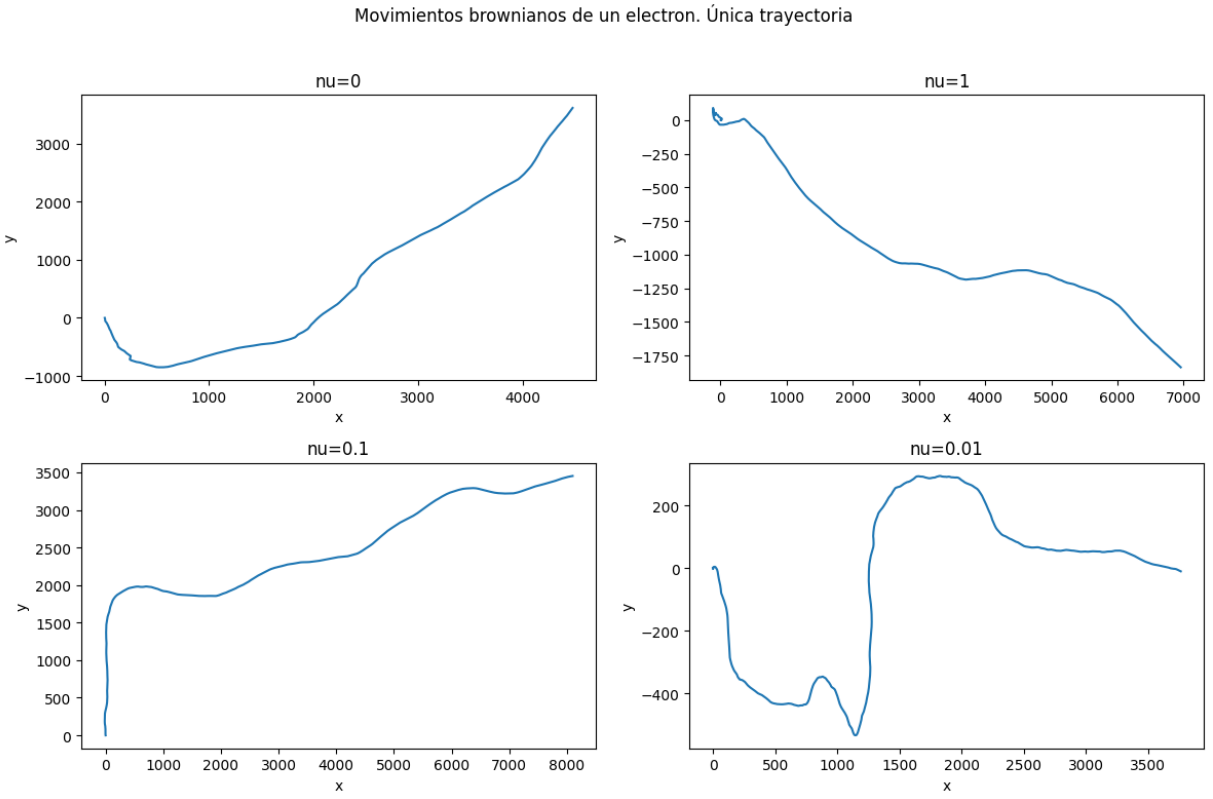
1. Lo primero será establecer las constantes y valores que vamos a necesitar en este experimento:

In [2]:
```
# Paso de tiempo
dt = 0.1

# Número de pasos
num_steps = int(100/dt)

# Valores de t
t_values = np.linspace(0, 100, num_steps+1)

# Trayectorias a realizar
trayectorias = 100

# Valores de nu
nu_valores = [0, 1, 1/10, 1/100]
```

2. Ahora que parece que tenemos todos los datos, vamos a empezar con la trayectoria del electrón por el plato. Vamos a realizar un experimento para $\nu = 0$

In [3]:
```
fig, axs = plt.subplots(2,2,figsize=(12,8))
fig.suptitle("Movimientos brownianos de un electron. Única trayectoria")


for k, nu in enumerate(nu_valores):
```

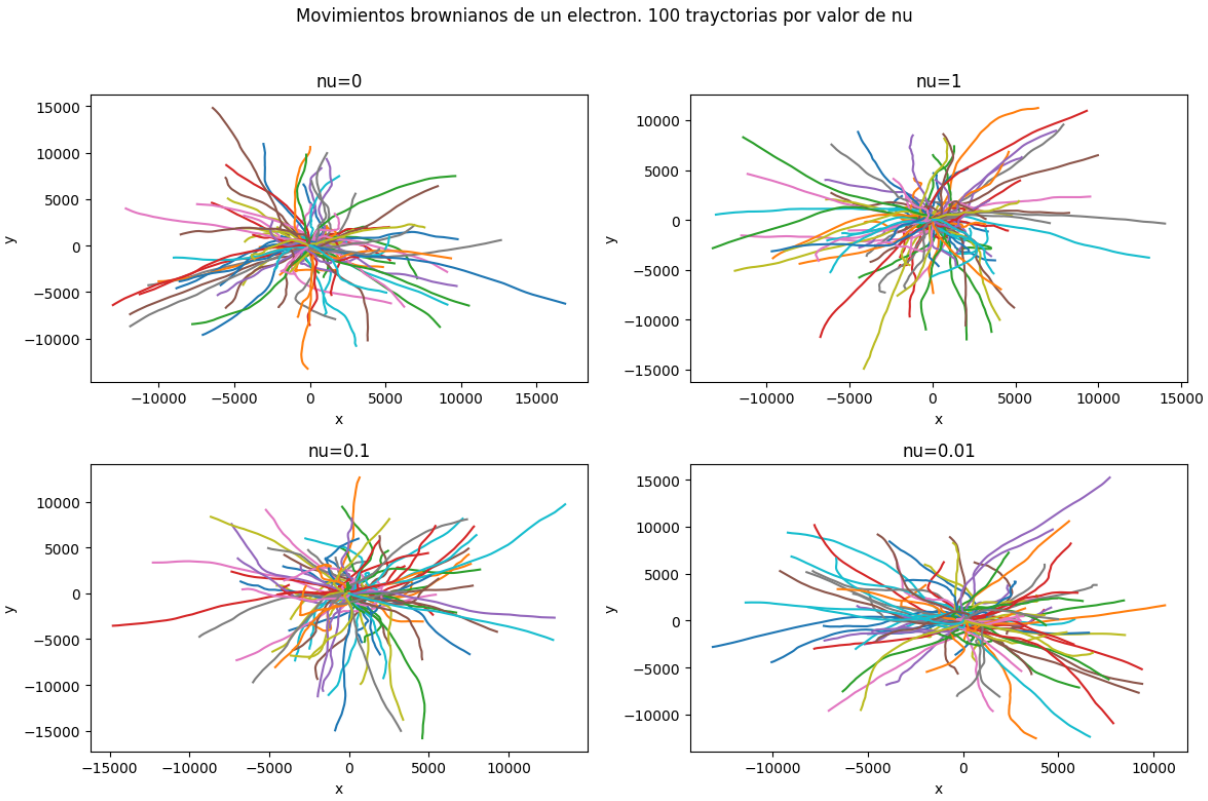
```
  row = k // 2
  col = k % 2
  dW = np.sqrt(dt) * np.cumsum(np.random.normal(size=(2, num_steps)), axis=1)
  X = np.zeros((2, num_steps + 1))

  for i in range(1, num_steps + 1):
    F_t = np.array([np.cos(2 * np.pi * nu * t_values[i]),np.sin(2 * np.pi * nu * t_
    X[:,i] = X[:, i-1] + F_t * dt + dW[:, i-1]

  axs[row, col].plot(X[0,:], X[1,:])
  axs[row, col].set_title(f'nu={nu}')
  axs[row, col].set_xlabel('x')
  axs[row, col].set_ylabel('y')

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



3. Ahora que tenemos una trayectoria, vamos con la creación de 100 trayectorias para cada valor de ν.

In [5]:
```
fig, axs = plt.subplots(2,2,figsize=(12,8))
fig.suptitle("Movimientos brownianos de un electron. 100 trayctorias por valor de n

X_tray = []

for k, nu in enumerate(nu_valores):
  row = k // 2
  col = k % 2

  tray_set = np.zeros((trayectorias, 2, num_steps + 1))
```

```
  for j in range(trayectorias):
    X = np.zeros((2, num_steps + 1))
    dW = np.sqrt(dt) * np.cumsum(np.random.normal(size=(2, num_steps)), axis=1)

    for i in range(1, num_steps + 1):
      F_t = np.array([np.cos(2 * np.pi * nu * t_values[i]),np.sin(2 * np.pi * nu *
      X[:,i] = X[:, i-1] + F_t * dt + dW[:, i-1]

    tray_set[j, :, :] = X
    axs[row, col].plot(X[0,:], X[1,:])
  axs[row, col].set_title(f'nu={nu}')
  axs[row, col].set_xlabel('x')
  axs[row, col].set_ylabel('y')

  X_tray.append(tray_set)

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

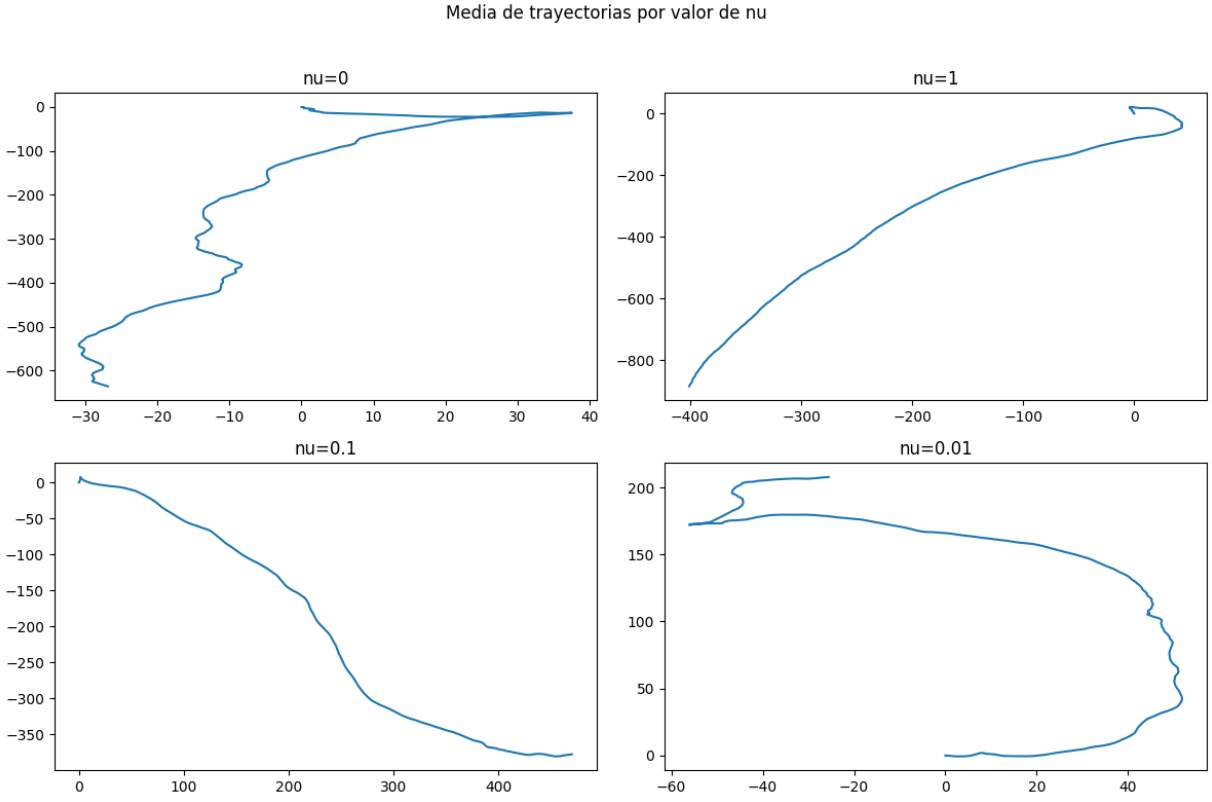


En la figura anterior podemos observar los diferentes caminos que realiza un electrón en el plato en los 100 experimentos realizados por cada una de las frecuencias de ν.

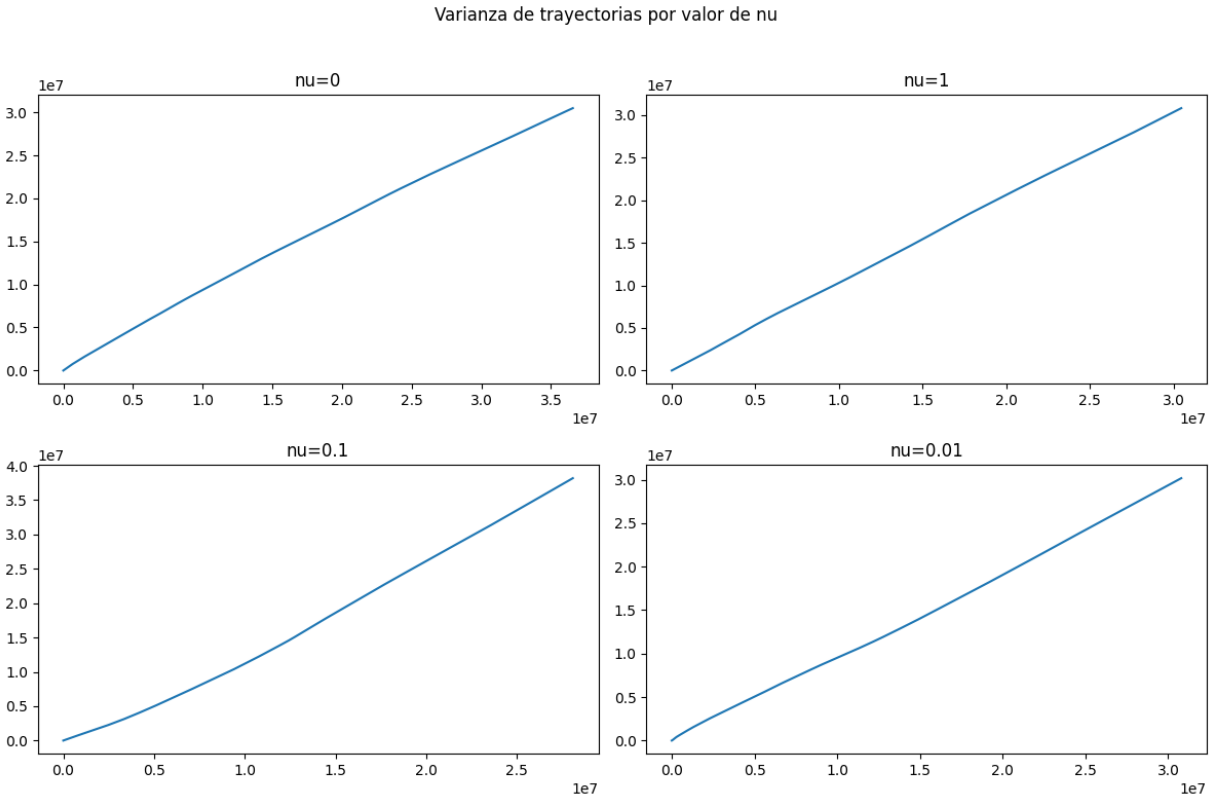
4. Como vemos muchas líneas, vamos a ver la media y varianza con la frecuencia.

In [20]:
```
fig, axs = plt.subplots(2,2,figsize=(12,8))
fig.suptitle("Media de trayectorias por valor de nu")

for k, nu in enumerate(nu_valores):
  row = k // 2
```

```
  col = k % 2

  tray = X_tray[k]

  mean_tray = np.mean(tray, axis=0)

  axs[row, col].set_title(f'nu={nu}')
  axs[row, col].plot(mean_tray[0, :], mean_tray[1, :], label="Trayecto medio")

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



In [21]:
```
fig, axs = plt.subplots(2,2,figsize=(12,8))
fig.suptitle("Varianza de trayectorias por valor de nu")

for k, nu in enumerate(nu_valores):
  row = k // 2
  col = k % 2

  tray = X_tray[k]

  var_tray = np.var(tray, axis=0)

  axs[row, col].set_title(f'nu={nu}')
  axs[row, col].plot(var_tray[0, :], var_tray[1, :], label="Varianza")

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



Podemos observar en las dos últimas gráficas la media y varianza de los caminos para cada frecuencia ν. Mientras que vemos que la media de las trayectorias marcan caminos diferentes, vemos que la varianza de todas las frecuencias mantienen las mismas rectas.

Conclusiones

Vemos que la media para cada frecuencia ν no se parecen en nada a lo obtenido por una única trayectoria, mientras que la varianza para todas las frecuancias tienen una recta con, aproximadamente, la misma pendiente.

La variabilidad de las trayectorias de las medias se debe a la interacción de los términos estocásticos de la ecuación diferencial estocástica definida en el enunciado.

En cambio, la similitud de las curvas de las varianzas para diferentes valores de ν se debe a que la varianza es sensible a las fluctuaciones estocásticas en las trayectorias individuales.