Proyecto final Probabilidad Honores

Departamento de Matemáticas, Universidad de los Andes
Colombia

Nombre del estudiante:	Luis Alejandro Rubiano
Código de estudiante:	202013482
Dirección de correo electrónico:	la.rubiano@uniandes.edu.co
Clase:	Probabilidad Honores MATE2510
Profesor:	Mauricio Junca

1 Introducción

Hay m clientes que se mueven entre k nodos de una red (estaciones de servicio), la red es cerrada por lo tanto no entran ni salen nuevos clientes. La forma en la que los clientes se mueven dentro de la red está dada por una matriz $P = (P_{ij})$ en donde un cliente que sale del nodo i tiene probabilidad P_{ij} de ir al nodo j; La probabilidad de que un cliente vaya de nuevo al mismo nodo es 0, es decir las entradas de la diagonal principal de la matriz son 0. Para que esto sea un espacio de probabilidad, las filas de la matriz P suman 1. El tiempo de servicio en un nodo j tiene una distribución Exponencial(β_j) y es independiente del servicio de otros nodos y de los tiempos de servicio anteriores.

En este proyecto se discutirán aspectos de los cambios en las configuraciones de la red, se llevarán a cabo simulaciones de esta red para estimar algunas de sus propiedades, y se estudiarán propiedades adicionales de la matriz de movimiento dentro de la red.

2 Preguntas teóricas

Sea

$$S = \left\{ n = (n_1, ..., n_k) : 0 \leq n_i \in \mathbb{N}, 1 \leq i \leq k, \sum_{i=1}^{k} n_i = m \right\}$$

Entonces S es el conjunto de configuraciones distintas que puede haber en la red. Se define el proceso

$$Q(t) = (Q_1(t), ..., Q_k(t)) \in S,$$

donde $Q_i(t)$ es la cantidad de clientes en el nodo i en el instante t (tanto en servicio como en cola).

1. Suponga que en algún momento el estado de la red es $n = (n_1, ..., n_k) \in S$

   a Diga que distribución tiene el tiempo que dura la red en el estado n. Considere el caso que algún nodo no tenga clientes:

   Por la propiedad de perdida de memoria de la distribución exponencial, esto es igual a la probabilidad de que los tiempos de servicio de cada nodo sea menor o igual a t.

   Sea T_j el tiempo de servicio en un nodo j, sabemos que $T_j \sim$ Exponencial(β_j), entonces $min\{T_1, ..., T_k\} \sim$ Exponencial($\sum_{i=1}^{k} \beta_i \mathbb{1}_{n_i}$), donde $\mathbb{1}_{n_i} = 1$ si hay personas en n_i y 0 si no.

   Prueba:

   Sin perdida de generalidad asumimos que $n_j > 0$ para cada $n_j \in n$
   $F_{T_j}(t) = \mathbb{P}(T_j < t) = 1 - e^{-\beta_j t}$

Sea $T = min\{T_1, ..., T_k\}$

$F_T(t) = \mathbb{P}(T \leq t) = 1 - \mathbb{P}(T \geq t) = 1 - \mathbb{P}(min\{T_1, ..., T_k\} \geq t) = 1 - \mathbb{P}(T_1 \geq t, ...T_k \geq t) = 1 - (\mathbb{P}(T_1 \geq t) \cdot ... \cdot \mathbb{P}(T_k \geq t)) = 1 - e^{-\beta_1 t} ... e^{-\beta_k t} = 1 - e^{-t\sum_{i=1}^{k}\beta_i}$.

Por lo tanto, T $\sim$ Exponencial($\sum_{i=1}^{k}\beta_i \mathbb{1}_{n_i}$)

b Suponga que $n_i > 0$. Calcule la probabilidad de que el siguiente estado de la red sea $(n_1, ..., n_i - 1, ..., n_j + 1, ..., n_k)$, para $1 \leq j \leq k$

Esto acaba siendo igual a la probabilidad de que el minimo de los tiempos de servicio sea igual al del i-ésimo nodo, multiplicado por P_{ij} (probabilidad de que vaya a j desde i)

Esta probabilidad es $P_{ij}\frac{\beta_i}{\sum_{l=1}^{k}\beta_l}$

Prueba:

Sin pérdida de generalidad, asumimos que i = 1 y que $n_i > 0$. Si $n_i = 0$, la probabilidad es 0 de manera trivial

Definimos $T_{2-k} = min\{T_2, ..., T_k\}$

Por el resultado del numeral anterior tenemos que $T_{2-k} \sim$ Exponencial($\sum_{l=2}^{k}\beta_l \mathbb{1}_{n_l}$)

Por ley de probabilidad total e independencia de T_1 y T_{2-k}:

$\mathbb{P}(T_1 < Z)P_{ij} = P_{ij}\int_0^\infty \mathbb{P}(T_1 < T_{2-k}|T_1 = t) \cdot f_{T_1}(t)dt = P_{ij}\int_0^\infty \mathbb{P}(t < T_{2-k}|T_1 = t) \cdot f_{T_1}(t)dt = P_{ij}\int_0^\infty (1 - \mathbb{P}(T_{2-k} \leq t)) \cdot f_{T_1}(t)dt = P_{ij}\beta_1\int_0^\infty e^{-t\sum_{l=2}^{k}\beta_l}e^{-\beta_1 t}dt = P_{ij}\beta_1\int_0^\infty e^{-t\sum_{l=1}^{k}\beta_l}dt = P_{ij}\frac{\beta_i}{\sum_{l=1}^{k}\beta_l \mathbb{1}_{n_l}}$

2. Use el teorema de Perron-Frobenius para mostrar que existe un vector $\pi \in \mathbb{R}_+^k$ tal que $\pi^T P = \pi^T$ y $\sum_{i=1}^{k}\pi_k = 1$. Calcule π para la matriz P que escogió y verifique que el nodo que más acumula clientes es el nodo cuya razón $\frac{\pi_j}{\beta_j}$ es mayor.

Usando el teorema, y software encontré que el valor-propio de Perron–Frobenius de la matriz P^T es el valor-propio 1. Ahora el vector propio asociado a este valor propio (aproximado) y normalizado es

$\pi^T = (0.567996685153517, 0.471906437636160, 0.145730013080882, 0.445874830530333, 0.484399090171687)$, se puede verificar que $\pi^T P = \pi^T$.

El vector cociente $\frac{\pi_j}{\beta_j}$ para $1 < j < k$ es (4.54397348122814, 6.13478368927008, 0.728650065404408, 1.33762449159100, 1.93759636068675)

$\frac{\pi_2}{\beta_2}$ es la componente más grande de este vector, más adelante se ve que el nodo 2 es el que acumula más clientes

3 Código y detalles de implementación

Código para generar simulación:

```
from numpy import random

def simulacion(n, P, betas, T):
    """
    n = Q_0, es la configuraci n de la red
    en el instante 0, tiene estructura
    de lista de Python. A medida que toma
    lugar la simulaci n, n va cambiando

    P, matriz cuya entrada ij representa
    la probabilidad de que al salir un cliente
    del nodo i, vaya al nodo j. Estructura
```

```
    de lista de listas de Python.

    betas, son el parametro del tiempo de
    servicio para un nodo j, que toma
    distribuci n exponencial(b_j)

    T, tiempo final hasta el que se va a
    realizar la simulaci n. Esta inicia
    en el tiempo t = 0. int de Python

    Retorna: n final, integrales --> tupla (t,Q(t))
    """

    k = len(n) #N mero de nodos
    m = sum(n) #N mero total de clientes

    assert len(P)==k, "Tama o de la matriz no coincide con k"
    assert len(P[0])==k, "Tama o de la matriz no coincide con k" #Verifica que dim(P)=kxk

    assert len(betas)==k, "Tama o de betas no coincide con k" #Verifica que dim(betas) = k

    assert type(T) == int, "T no es un entero" #Verifica que T sea un entero positivo
    assert T>=0, "T es negativo"

    for i in P: #Verifica que las filas de P sumen 1

        assert sum(i) == 1, "Las filas de P no suman 1"



    esperas = [float('inf') for i in range(k)] #Crea lista con tiempos de espera para las personas en cada nodo

    suma = [0] * k #Suma Q para cada tiempo

    valores_integral = [] #Valores que toma la integral, lista de tuplas de tipo (t,Q(t))


    for t in range(T):

        for i in range(k):


            if esperas[i] != float('inf') and n[i]>0: #Si hay tiempo de espera lo disminuye en 1

                esperas[i] = esperas[i] - 1

            elif esperas[i] == float('inf') and n[i]>0: #Si no hay tiempo de espera y hay personas, inicia el tiempo de espera

                esperas[i] = random.exponential(1/betas[i]) #Toma como parametro el inverso de beta


            if esperas[i]<=0 and n[i]>0: #Si se acab  el tiempo de espera

                n[i] = n[i] - 1 #Disminuye en 1 la cantidad de personas en i

                probabilidades = P[i]

                j = random.choice([j for j in range(k)], p=probabilidades) #Selecciona el nodo j para enviar a la persona

                n[j] = n[j] + 1 #Aumenta en 1 la cantidad de personas en j

                esperas[i] = float('inf') #Pone en infinito el tiempo de espera para el nodo i


        suma = [x + y for x, y in zip(suma, n)] #Agrega a la suma el n mero de personas en n para t.

        if t !=0: #evita divisi n por 0

```

```
            integral = [i/t for i in suma] #Calcula la integral del Teorema Erg dico , divide por T la suma.

            valores_integral.append((t, integral))

    return n, valores_integral
```

Listing 1: main.py

Simulación punto 3

Aquí $n_0 = [4, 5, 1, 7, 3]$

P es la matriz que se ve en el código, fue generada usando Numpy Dirichlet distribution

Los betas son = [1/8,1/13,1/5,1/3,1/4]

T = 50000

100 simulaciones

```

from main import *
import matplotlib.pyplot as plt

n_0 = [4,5,1,7,3]

#P's tomadas usando Numpy Dirichlet distribution

P = [[0,0.37240585, 0.02273727 ,0.49338616, 0.11147072],
     [0.27647299, 0, 0.05890617, 0.12458766, 0.54003318],
     [0.06716767, 0.06858368, 0, 0.22524363, 0.63900502],
     [0.51885639, 0.31111651, 0.00604218, 0, 0.16398492],
     [0.40543961, 0.23052689, 0.21123723, 0.15279627, 0]]

ts = [] #Valores para t
x_1 = [] #Valores para Q_x1(t)
x_2 = []
x_3 = []
x_4 = []
x_5 = []

betas = [1/8,1/13,1/5,1/3,1/4]

T = 50000

for _ in range(100):

    n, integrales = simulacion(n_0,P,betas,T)

    for t, Q in integrales:

        x_1.append(Q[0])
        x_2.append(Q[1])
        x_3.append(Q[2])
        x_4.append(Q[3])
        x_5.append(Q[4])
        ts.append(t)

    if _%10==0:
        print(_)

k = 0
for xi in [x_1,x_2,x_3,x_4,x_5]:

    plt.title(f'Valor medio de n_{k+1} vs t')
    plt.ylabel(f'Valor medio de n_{k+1}')
    plt.xlabel('t')

    plt.scatter(ts, xi, s = 1)
    plt.savefig(f'{k+1}.jpg')
    plt.clf()
    k+=1
```

Listing 2: pruebas p3.py

Aquí n_0 es proporcional a m para m = 50,100 y 200, en contraste a m = 20 para el punto anterior, P y betas son los mismos que antes

100 simulaciones

T = 100000

```python
from main import *
import matplotlib.pyplot as plt
import numpy as np


#P's tomadas usando Numpy Dirichlet distribution

P = [[0,0.37240585, 0.02273727 ,0.49338616, 0.11147072],
     [0.27647299, 0, 0.05890617, 0.12458766, 0.54003318],
     [0.06716767, 0.06858368, 0, 0.22524363, 0.63900502],
     [0.51885639, 0.31111651, 0.00604218, 0, 0.16398492],
     [0.40543961, 0.23052689, 0.21123723, 0.15279627, 0]]

m50 = [] #Valores para m = 50
m100 = []
m200 = []

betas = [1/8,1/13,1/5,1/3,1/4]

T = 100000

n_0m = zip(([8,2,20,11,9], [16,4,40,22,18], [32,8,80,44,36]), [m50,m100,m200])

for n_0, m in n_0m:

    for _ in range(100):

        n, integrales = simulacion(n_0,P,betas,T)

        m.append(integrales[-1][1]) #Solo el valor

        if _%10==0:
            print(_)

m50x1 = sum(i[0] for i in m50)/len(m50) #Promedios para cada x
m50x2 = sum(i[1] for i in m50)/len(m50)
m50x3 = sum(i[2] for i in m50)/len(m50)
m50x4 = sum(i[3] for i in m50)/len(m50)
m50x5 = sum(i[4] for i in m50)/len(m50)

m100x1 = sum(i[0] for i in m100)/len(m100)
m100x2 = sum(i[1] for i in m100)/len(m100)
m100x3 = sum(i[2] for i in m100)/len(m100)
m100x4 = sum(i[3] for i in m100)/len(m100)
m100x5 = sum(i[4] for i in m100)/len(m100)

m200x1 = sum(i[0] for i in m200)/len(m200)
m200x2 = sum(i[1] for i in m200)/len(m200)
m200x3 = sum(i[2] for i in m200)/len(m200)
m200x4 = sum(i[3] for i in m200)/len(m200)
m200x5 = sum(i[4] for i in m200)/len(m200)

x = np.array([50,100,200])

#################################################################
plt.title('Limite del valor medio de n_1 vs m')
plt.ylabel('Limite del valor medio de n_1')
plt.xlabel('m')
plt.scatter(x, [m50x1,m100x1,m200x1], s = 10)

a, b = np.polyfit(x, [m50x1,m100x1,m200x1], 1)
plt.plot(x, a*x + b)

print(a,b)

plt.savefig('limitesx1.jpg')
plt.clf()
#################################################################
plt.title('Limite del valor medio de n_2 vs m')
plt.ylabel('Limite del valor medio de n_2')
plt.xlabel('m')
plt.scatter(x, [m50x2,m100x2,m200x2], s = 10)

a, b = np.polyfit(x, [m50x2,m100x2,m200x2], 1)
plt.plot(x, a*x + b)
```

```

print(a,b)

plt.savefig('limitesx2.jpg')
plt.clf()
#################################################################
plt.title('Limite del valor medio de n_3 vs m')
plt.ylabel('Limite del valor medio de n_3')
plt.xlabel('m')
plt.scatter(x, [m50x3,m100x3,m200x3], s = 10)

a, b = np.polyfit(x, [m50x3,m100x3,m200x3], 1)
plt.plot(x, a*x + b)

print(a,b)

plt.savefig('limitesx3.jpg')
plt.clf()
#################################################################
plt.title('Limite del valor medio de n_4 vs m')
plt.ylabel('Limite del valor medio de n_4')
plt.xlabel('m')
plt.scatter(x, [m50x4,m100x4,m200x4], s = 10)

a, b = np.polyfit(x, [m50x4,m100x4,m200x4], 1)
plt.plot(x, a*x + b)

print(a,b)

plt.savefig('limitesx4.jpg')
plt.clf()
#################################################################
plt.title('Limite del valor medio de n_5 vs m')
plt.ylabel('Limite del valor medio de n_5')
plt.xlabel('m')
plt.scatter(x, [m50x5,m100x5,m200x5], s = 10)

a, b = np.polyfit(x, [m50x5,m100x5,m200x5], 1)
plt.plot(x, a*x + b)

print(a,b)

plt.savefig('limitesx5.jpg')
plt.clf()
```

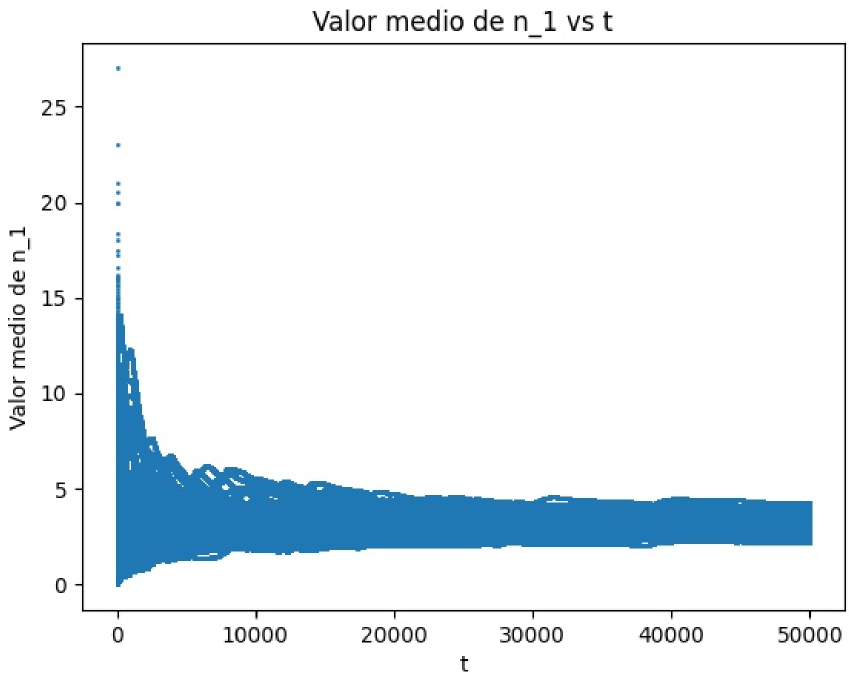
Listing 3: pruebas p3.py

Todos los códigos también se encuentran adjuntos

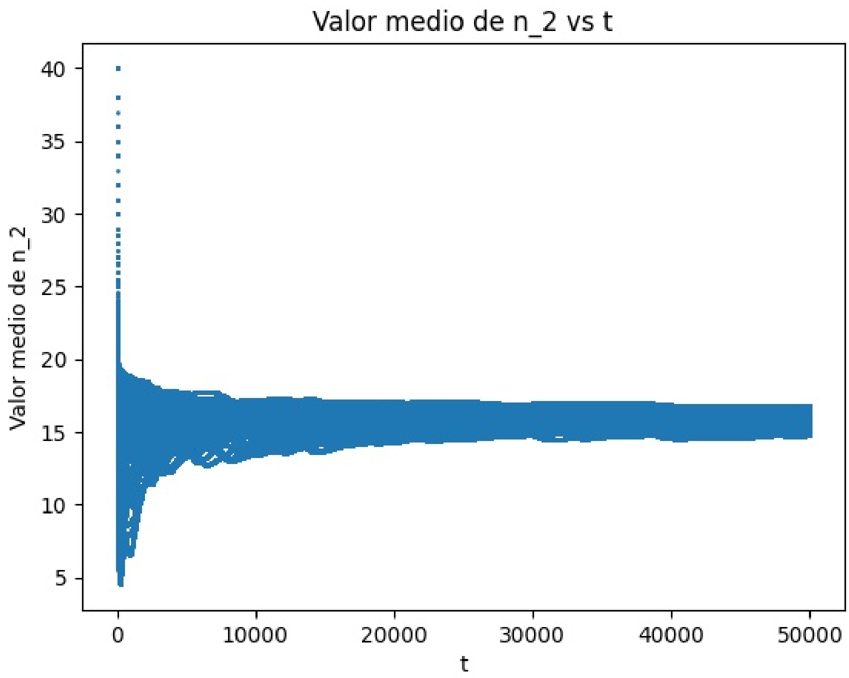
4 Resultados de las simulaciones

Primera simulación:

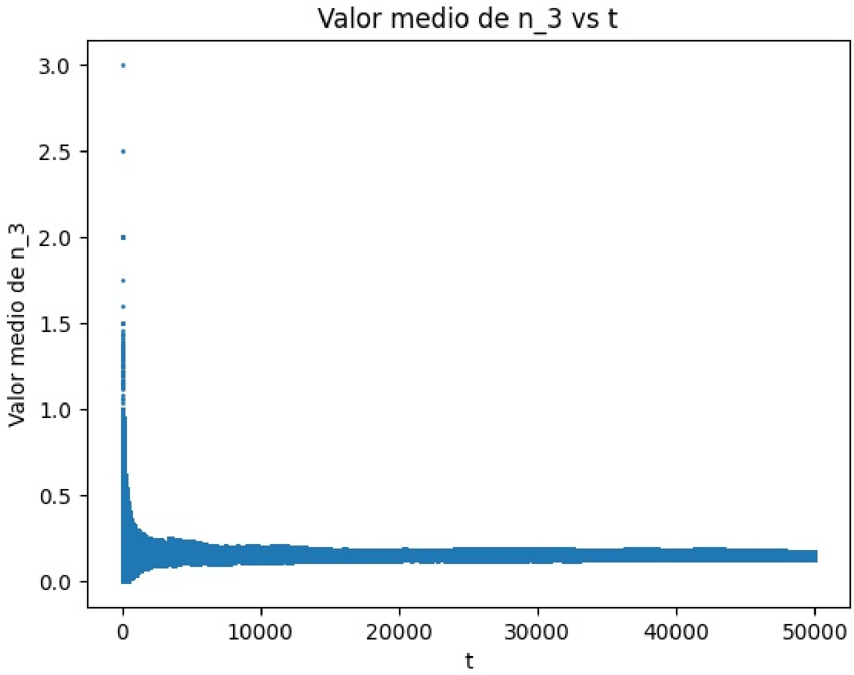
Para m = 20



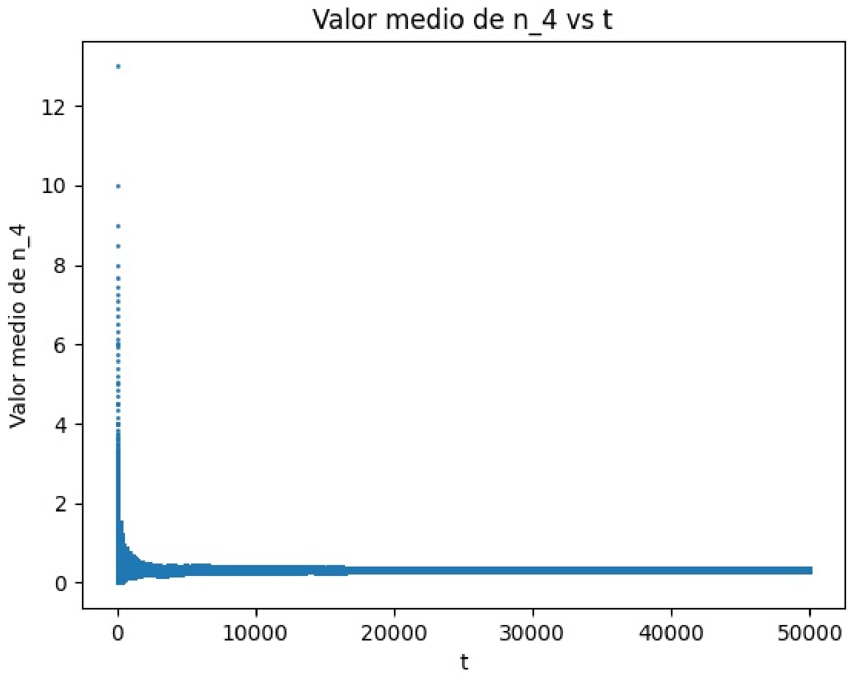
Parece que converge a un valor entre 3 y 5.



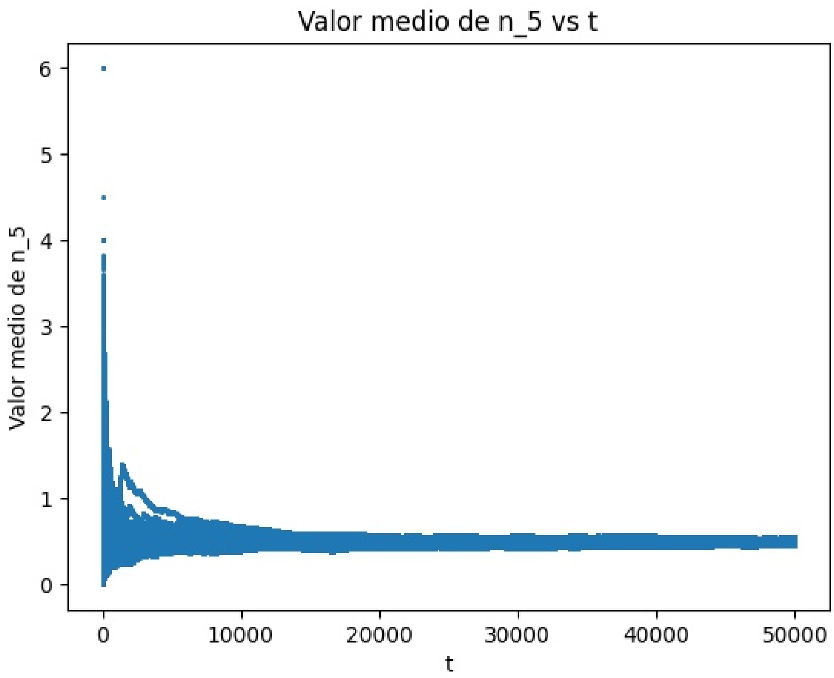
Parece que converge a un valor entre 14 y 16.



Parece que converge a un valor entre 1 y 3.



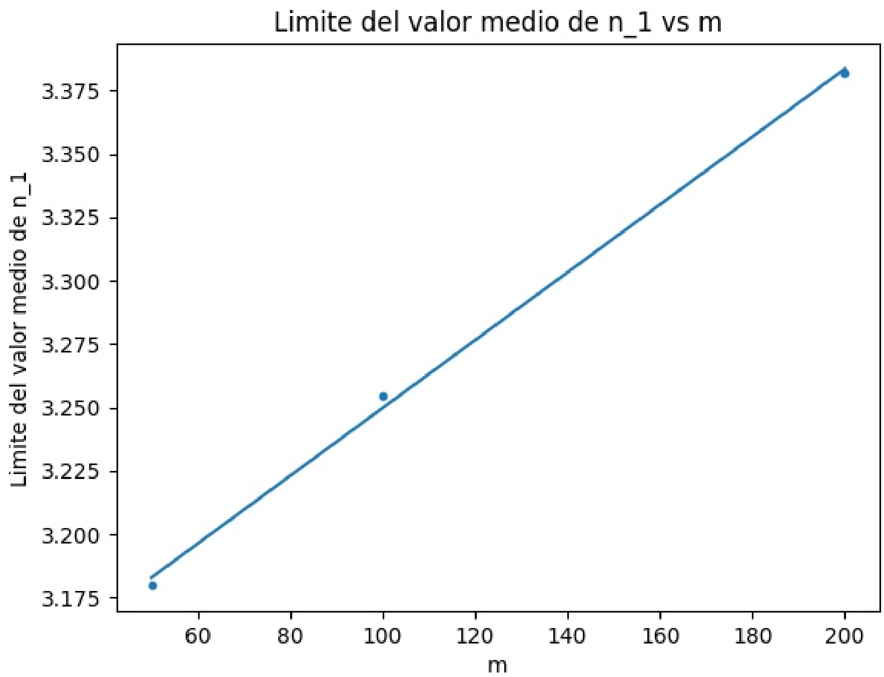
Parece que converge a 0.



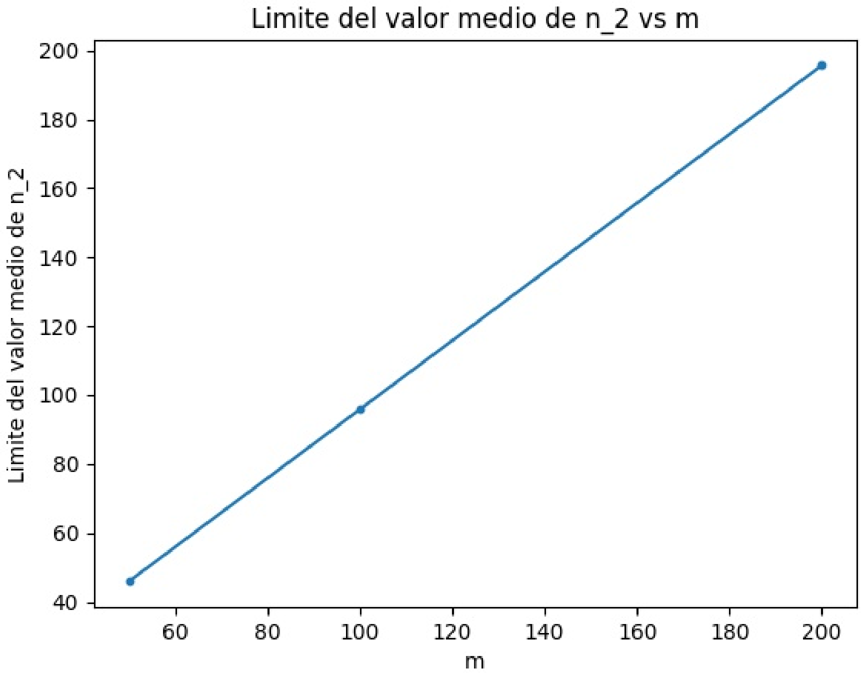
Parece converger a un valor entre 0 y 1.

Segunda simulación:

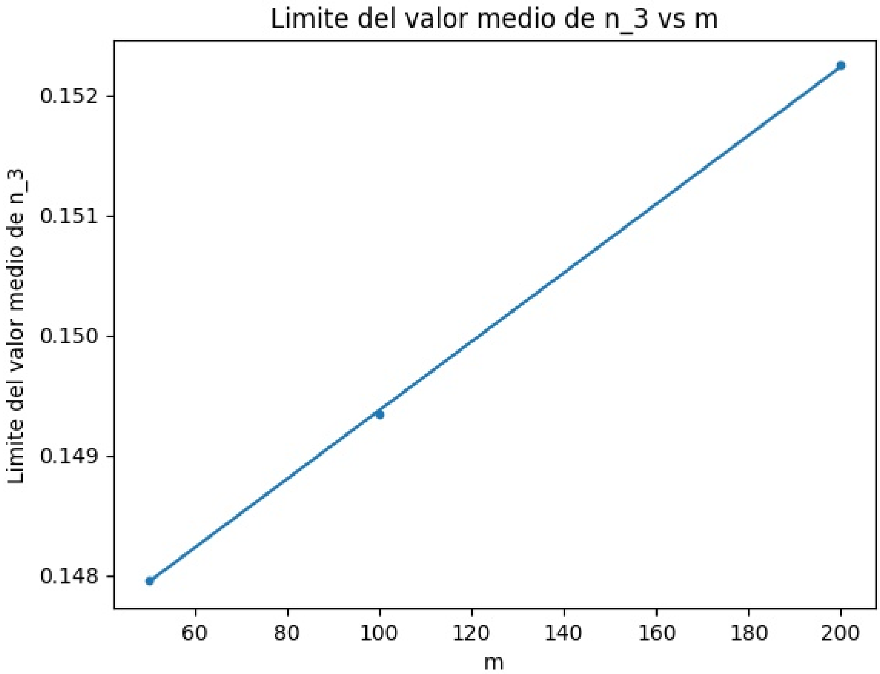
M toma valores de 50, 100 y 200



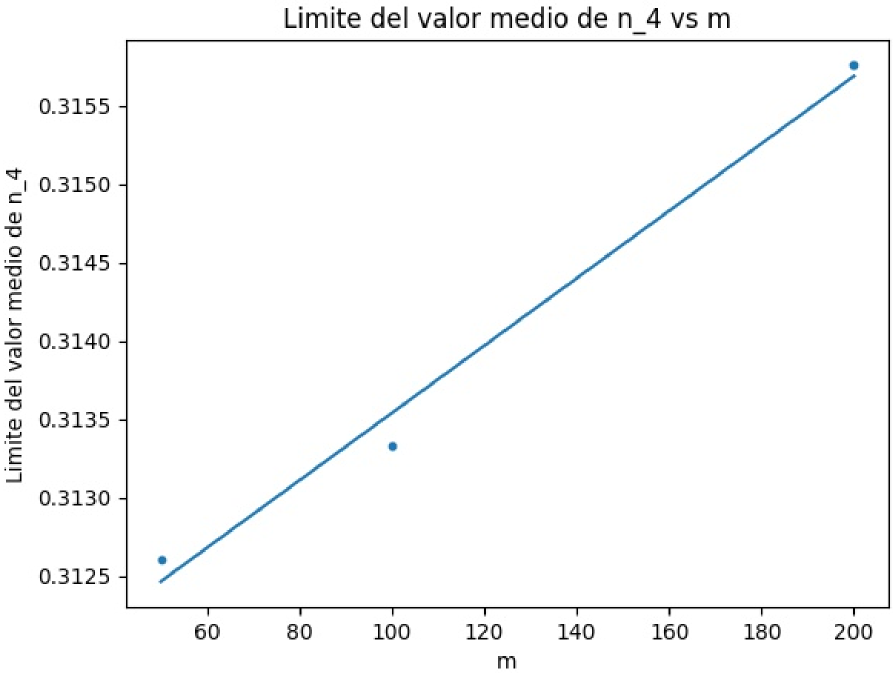
Se acuerdo a la regresión lineal de Numpy, n_1 converge a 0.001337m + 3.11605586, cuando T tiende a infinito



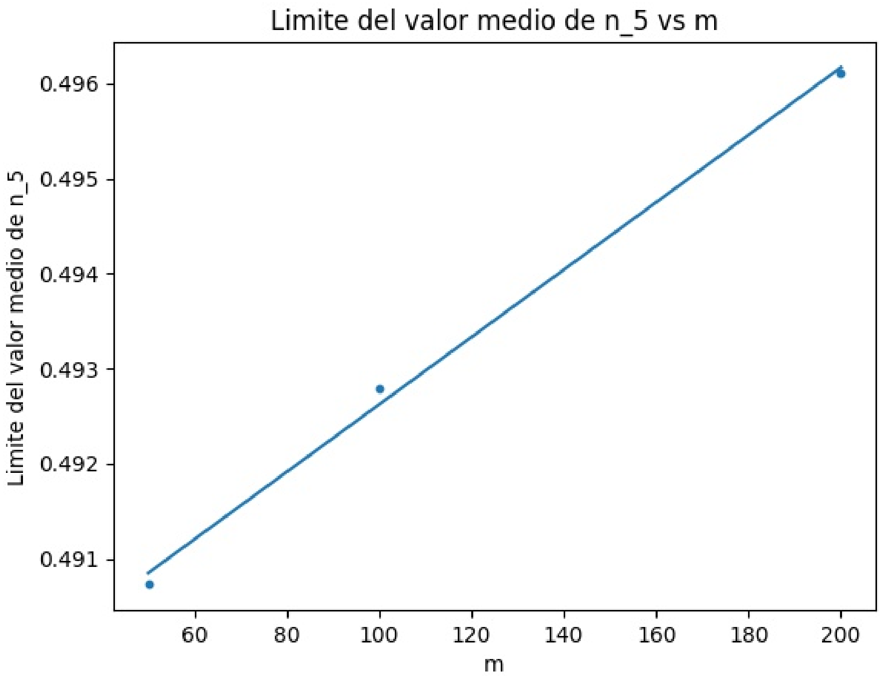
De acuerdo a la regresión lineal de Numpy, n_2 converge a 0.9985869m - 4.063027, cuando T tiende a infinito



De acuerdo a la regresión lineal de Numpy, n_3 converge a 2.868942975144093e-05m + 0.14650366503665035, cuando T tiende a infinito



De acuerdo a la regresión lineal de Numpy, n_4 converge a 2.150878651643697e-05m + 0.31138961389613895, cuando T tiende a infinito



De acuerdo a la regresión lineal de Numpy, n_5 converge a 3.544292585783255e-05 + 0.48907, cuando T tiende a infinito

5 Discusión de los resultados y conclusiones.

Se pueden extraer diferentes conclusiones de los resultados, primero pudimos ver que el $\lim_{T\to\infty} \frac{1}{T}\int_0^T f(Q(t))dt$ existe casi siempre, esto se puede ver en las primeras gráficas, en donde a medida que T se hacer más grande, el valor medio de n_j parece converger a un valor en todos los casos. En las primeras gráficas también se puede observar como el valor medio de n_2 es considerablemente mayor al de los demás, es aproximadamente 3/4 del valor de m, esto cumple la relación dada por el teorema de Perron-Froebenius, la cual nos dice que el j con mayor $\frac{\pi_j}{\beta_j}$ es el nodo que más acumula clientes (en este caso el 2.)

Por otro lado, con las últimas gráficas logramos ver que límite del valor medio de cada n crece de manera lineal en m, esto lo logramos graficando los limites para m = 50, 100 y 200, y usando una regresión lineal de Numpy.