FACULTAD DE INGENIERIA
Universidad de Buenos Aires

U.B.A. FACULTAD DE INGENIERÍA

Departamento de Computación

Modelos y Simulación 7526 - 9519

TRABAJO PRÁCTICO #2

Procesos de Poisson, Cadenas de Markov, Sistemas dinámicos, Simpy

Curso: 2019 - 1er Cuatrimestre

Turno: Miércoles

GRUPO N° 1	
Integrantes	**Padrón**
Amurrio, Gastón	93584
Gamarra Silva, Cynthia Marlene	92702
Pinto, Tomás	98757
Fecha de Entrega:	19-06-2019
Fecha de aprobación:	
Calificación:	
Firma de aprobación:	

Observaciones:

Índice

Índice 1

1. Enunciado del trabajo práctico 2

2. Introducción 5

3. Implementación y resultados 5
3.1. Ejercicio 1 . . . 5
3.2. Ejercicio 2 . . . 5
3.3. Ejercicio 3 . . . 5
3.4. Ejercicio 4 . . . 5
3.5. Ejercicio 5 . . . 5

4. Conclusiones y aclaraciones 6

Referencias 6

A. Código fuente 7
A.1. Resolución ejercicio 1 . . . 7
A.1.1. instruccion . . . 7
A.1.2. cache.py . . . 7
A.1.3. memoria principal . . . 8
A.1.4. OS . . . 8
A.1.5. procesador . . . 9
A.1.6. main . . . 9
A.1.7. grafico . . . 9
A.2. Resolución ejercicio 2 . . . 10
A.2.1. ejercicio2.py . . . 10
A.3. Resolución ejercicio 3 . . . 11
A.3.1. ejercicio3.py . . . 11
A.4. Resolución ejercicio 4 . . . 12
A.4.1. ejercicio4.py . . . 12
A.5. Resolución ejercicio 5 . . . 13
A.5.1. ejercicio5.py . . . 13

1. Enunciado del trabajo práctico

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2

Simulación - 75.26 – 95.19

Consideraciones generales

La entrega de este TP debe contar con un informe explicando el procedimiento utilizado para resolver cada ejercicio y justificando las conclusiones a las que se arriba en cada punto. Se debe incluir también el código fuente utilizado para resolver cada ejercicio en uno o más archivo(s) por ejercicio.

Procesos de Poisson

Ejercicio 1

Utilizando Matlab, Octave o Python (sin utilizar Simpy) se debe determinar con cuál de las siguientes 2 arquitecturas nos permite tener un menor tiempo de procesamiento.

A un procesador arriban instrucciones siguiendo una distribución exponencial con una tasa de 250 instrucciones por microsegundo.

Las instrucciones se pueden agrupar en las siguientes categorías, y en función de ella es el tiempo que demoran en ser ejecutadas el cual sigue una distribución exponencial con parámetro lambda especificado a continuación.

Tipo de instrucción	Probabilidad	Tasa del tiempo de ejecución (instrucciones por µseg)
Simple	0,6	60
Compleja	0,4	10

Se sabe que el 65% de las instrucciones requieren buscar datos en memoria para antes de ser ejecutadas.

En este punto se plantean dos alternativas:

Alternativa 1: Cada vez que se requiera un dato de memoria se lo busca en memoria principal. Tarea cuyo tiempo corresponde a una distribución exponencial con una tasa de 2000 instrucciones por microsegundo.

Alternativa 2: Implementar un Caché, con el cual existiría una probabilidad de .6 de encontrar el dato requerido y no tener la necesidad de buscarlo en memoria principal.

El tiempo de acceso a este caché se puede modelar siguiendo una distribución exponencial con una tasa de 500 instrucciones por microsegundo.

Cuando el dato no es encontrado en el caché y debe buscarse en memoria principal donde su demora también sigue una distribución exponencial con una tasa de 1500 instrucciones por microsegundo.

FACULTAD DE INGENIERIA Universidad de Buenos Aires

Trabajo Práctico 2

Simulación - 75.26 – 95.19

Cadenas de Markov

Ejercicio 2

Una entidad bancaria se encuentra analizando cuantos clientes en simultáneo están conectados a su home banking.
Para ello comenzó a monitorear cada minuto la cantidad de clientes activos, y analizó como se modifica esta cantidad de una observación a la siguiente.
Se determinó que la probabilidad que la cantidad de clientes conectados aumentara, o se mantenga igual, podía modelarse utilizando una distribución de probabilidad Binomial, mientras que la probabilidad de tener menos clientes conectados respondía a una distribución uniforme.
Siendo i y j dos cantidades de usuarios conectados al sistema.

$$P\left(i \to j\right) = \begin{cases} \binom{n}{j-i} p^{j-i}(1-p)^{n-j-i} & si\ j \geq i \\ \dfrac{1 - \sum_{j=i}^{j=n} P\left(i \to j\right)}{i-1} & si\ j < i \end{cases}$$

Se sabe además que la probabilidad que un cliente se conecte al sitio es p=0,7
(Por simplicidad supondremos que el servidor de este home banking permite cómo máximo 50 clientes en simultáneo)

a) Determinar la matriz de transición de estados explicando cómo se obtiene la misma.
b) Utilizando Matlab, Octave o Python simular una posible evolución del sistema a lo largo de 100 observaciones graficando cómo se modifica la cantidad de clientes conectados en cada momento.
c) Simulando 100.000 observaciones realizar un histograma mostrando cuantas veces el sistema estuvo en cada estado.
(Recomendación: utilizar tantas categorías en el histograma como estados tiene el sistema).
d) Determinar el % de tiempo que en el home banking no tuvo clientes conectados.
e) Se está evaluando migrar el home banking a un servidor que permitiría cómo máximo 40 clientes conectados en simultáneo. La migración sólo se realizaría si la probabilidad de tener más de 40 clientes es menor a 10%. Indique si se puede recomendar realizar la migración.

Sistemas dinámicos

Ejercicio 3

El siguiente sistema dinámico es conocido como “Modelo de la telaraña”.
Mediante el mismo se puede modelar la formación de precios de productos cuya oferta se establece en función del precio de mercado en el período de tiempo anterior.

$$\begin{cases} Q_t^{demanda} = a - bP_t \\ Q_t^{oferta} = dP_{t-1} - c \\ Q_t^{demanda} = Q_t^{oferta} = Q_t \end{cases}$$

Con:
a=9
b=1,1
c=0,4
d=1
P_0=8

Se pide:

1. Determinar el tipo de sistema (discreto/continuo, lineal/ no lineal, autónomo/no autónomo, de primer orden / orden superior)
2. Hallar el o los puntos de equilibrio.
3. Realizar un análisis asintótico del sistema.
   ¿Cómo afectan las condiciones iniciales al estado asintótico del sistema? Justifique.
   Simule el sistema para distintas condiciones iniciales e interprete los resultados.
4. Graficar la variable precio en función del período (t) para 100 períodos.
5. Graficar el espacio de faces del sistema a través de 100 períodos para pares ordenados (Q_t , P_t) (mostrar los puntos vinculados mediante una recta).

Simpy

Ejercicio 4

Resolver el Ejercicio 1 utilizando Simpy y comparar los resultados obtenidos con los obtenidos en el Ejercicio1.

Ejercicio 5

Siguiendo una distribución Exponencial negativa de media 45 ms arriban solicitudes a un cluster de base de datos, compuesta por 6 servidores y un balanceador de carga.
El tiempo de procesamiento de cada solicitud dependerá del tipo de solicitud que se trate:

Tipo	Probabilidad	Tiempo de proceso (mseg)
A	.6	75 ± 10
B	.25	60 ± 15
C	.15	90 ± 20

Se necesita determinar la mejor política de asignación de procesos a utilizar en el balanceador entre las siguientes:

a. Utilizando una política Round Robin (la primer solicitud se asigna al servidor 1, la segunda al 2, etc).
b. La solicitud es asignada al azar entre los 6 servidores.

(Justifique la respuesta midiendo todos los indicadores que considere necesarios)

2. Introducción

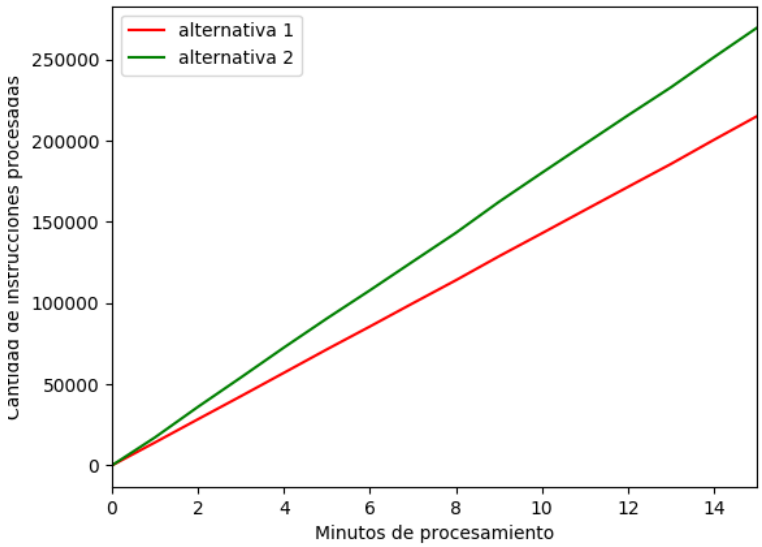
3. Implementación y resultados

Para cada uno de los ejercicios pedidos se realiza una explicación de cada uno de ellos. Se toma como base teórica lo explicado en clase tanto teórica como clase práctica.

3.1. Ejercicio 1

En readme.md se encuentran las instrucciones de como ejecutar el programa Se utilizaron threads distintos para simular el OS que genera instrucciones, el procesador que las procesa y la memoria/cache (segun la alternativa ejecutada) que busca el dato en memoria de ser necesario, estos threads comparten una cola de instrucciones a ser ejecutadas, mientras que solo el OS y la memoria/cache comparten una cola de instrucciones que primero deben acceder a memoria

Los resultados obtenidos luego de ejecutar ambas alternativas durante 15 minutos fueron los siguientes:



Sin duda, la mejor alternativa es la segunda. Luego de 15 minutos pudo procesar alrededor de 50mil instrucciones mas que la primer alternativa.

3.2. Ejercicio 2

3.3. Ejercicio 3

3.4. Ejercicio 4

3.5. Ejercicio 5

Utilizando los datos del ejercicio procesamos nuestro archivo creado usando *Simpy* y obtuvimos los siguientes resultados:

- Con política Round Robin(en segundos):
  - Tiempo espera: 87.6
  - Tiempo max: 416.4
  - Tiempo min: 45.0
  - Tiempo total: 451783.9
- Las solicitudes son asignadas al azar entre los servidores:
  - Tiempo espera: 73.6
  - Tiempo max: 116.1
  - Tiempo min: 45.0
  - Tiempo total: 449154.5

Podemos observar que la mejor política de asignación es en donde la solicitud es asignada al azar entre los 6 servidores. Hay una diferencia significativa en el tiempo máximo entre ambos y el tiempo total.

4. Conclusiones y aclaraciones

El trabajo práctico nos permitió conocer y realizar simulaciones teniendo como base teórica los conceptos explicados en clase . Además, nos permitió conocer herramientas que permiten realizar simulaciones que son muy utilizadas en el campo científico.

Referencias

[1] Python, Generación de números con distintas distribuciones de probabilidad, https://docs.python.org/3/library/random.html.

A. Código fuente

A.1. Resolución ejercicio 1

A.1.1. instruccion

```
import random
SECOND_TO_MICROSECOND = 1000000
class InstruccionFactory(object):

  @staticmethod
  def nueva_instruccion():
    # Tiro un random entre 0 y 1, si es menor o igual a 0.6 entonces
    # es una instruccion simple
    # idem memoria
    tipo = random.uniform(0,1)
    memoria = random.uniform(0,1)
    lee_memoria = False
    if (memoria <= 0.65):
      lee_memoria = True
    if (tipo <= 0.6):
      return InstruccionSimple(lee_memoria)
    else:
      return InstruccionCompleja(lee_memoria)

class InstruccionCompleja(object):

  def __init__(self, lee_memoria):
    self.lee_memoria = lee_memoria

  def costo(self):
    return 10/float(SECOND_TO_MICROSECOND)

class InstruccionSimple(object):

  def __init__(self, lee_memoria):
    self.lee_memoria = lee_memoria

  def costo(self):
    return 60/float(SECOND_TO_MICROSECOND)
```

A.1.2. cache.py

```
import queue
import time
import numpy
import random

SECOND_TO_MICROSECOND = 1000000
TASA_ALT1 = 2000

def buscar(cola_acceso_memoria, cola_procesador):
  while True:
    if not (cola_acceso_memoria.empty()):
      instruccion = cola_acceso_memoria.get()
      #Busco el dato
      u = random.uniform(0,1)
      tiempo_busqueda_memoria = 1500
      if (u <= 0.6):
        tiempo_busqueda_memoria = 500

```

```
19        tiempo_de_espera_busqueda_dato = numpy.random.exponential(
      tiempo_busqueda_memoria)
20        time.sleep(tiempo_de_espera_busqueda_dato/float(SECOND_TO_MICROSECOND))
21        #Mando la instruccion a procesar
22        cola_procesador.put(instruccion)
```

A.1.3. memoria principal

```
1  import queue
2  import time
3  import numpy
4
5  SECOND_TO_MICROSECOND = 1000000
6  TASA_ALT1 = 2000
7
8  def buscar(cola_acceso_memoria, cola_procesador):
9    while True:
10     if not (cola_acceso_memoria.empty()):
11       instruccion = cola_acceso_memoria.get()
12       #Busco el dato
13       tiempo_de_espera_busqueda_dato = numpy.random.exponential(TASA_ALT1)
14       time.sleep(tiempo_de_espera_busqueda_dato/float(SECOND_TO_MICROSECOND))
15       #Mando la instruccion a procesar
16       cola_procesador.put(instruccion)
```

A.1.4. OS

```
1
2
3  SECOND_TO_MICROSECOND = 1000000
4  TASA_ARRIBO_INSTRUC = 250
5
6
7
8  def run(modo):
9    import instruccion
10   import procesador
11   import queue
12   import threading
13   import time
14   import numpy
15   import memoria_principal
16   import cache
17   generador_instrucciones = instruccion.InstruccionFactory()
18   cola_acceder_memoria = queue.Queue()
19   cola_instrucciones = queue.Queue()
20
21   threading.Thread(target=procesador.procesar, args=(cola_instrucciones, modo)).start
       ()
22   if modo == "alternativa_1":
23     threading.Thread(target=memoria_principal.buscar, args=(cola_acceder_memoria,
       cola_instrucciones,)).start()
24   elif modo == "alternativa_2":
25     threading.Thread(target=cache.buscar, args=(cola_acceder_memoria,
       cola_instrucciones,)).start()
26   while True:
27     tiempo_de_espera_arribo_instruc = numpy.random.exponential(TASA_ARRIBO_INSTRUC)
28     time.sleep(tiempo_de_espera_arribo_instruc/float(SECOND_TO_MICROSECOND))
29     instruccion = generador_instrucciones.nueva_instruccion()
30     if not(instruccion.lee_memoria):
31       cola_instrucciones.put(instruccion)
32     else:
33       cola_acceder_memoria.put(instruccion)
```

A.1.5. procesador

```
import queue
import time
import csv
def procesar(cola_instrucciones, modo):
  cantidad_procesada = 0
  minutos_procesamiento = 0
  start_time = time.time()
  while True:
    if not (cola_instrucciones.empty()):
      time.sleep(cola_instrucciones.get().costo())
      cantidad_procesada += 1
      print("\r" + modo + " Cantidad procesada: "+str(cantidad_procesada), end="")

    # cada 60 segundos registro la cantidad de instrucciones que fueron procesadas
    actual_time = time.time()
    if actual_time - start_time >= 60:
      minutos_procesamiento += 1
      start_time = actual_time
      with open(modo+'.csv', mode='a') as registro_procesamiento:
        writer = csv.writer(registro_procesamiento, delimiter=',', quotechar='"',
    quoting=csv.QUOTE_MINIMAL)
        writer.writerow([str(minutos_procesamiento), str(cantidad_procesada)])
```

A.1.6. main

```
import OS
import sys
import threading
opciones = ["alternativa_1", "alternativa_2"]
def main():
  if len(sys.argv) != 2:
    print("Ingrese un argumento: alternativa_1 o alternativa_2")
    return
  elif sys.argv[1] not in opciones:
    print("Argumento invalido. Usar alternativa_1 o alternativa_2")
  else:
    threading.Thread(target=OS.run, args=(sys.argv[1],)).start()
main()
```

A.1.7. grafico

```
import pandas as pd
import matplotlib.pyplot as plt
alt1 = pd.read_csv("alternativa_1.csv", names=["minuto", "cantidad_procesada"])
alt2 = pd.read_csv("alternativa_2.csv", names=["minuto", "cantidad_procesada"])

# gca stands for 'get current axis'
ax = plt.gca()
alt1.plot(x = 'minuto', y = 'cantidad_procesada', label = 'alternativa 1', color = '
    red', ax = ax)
alt2.plot(x = 'minuto', y = 'cantidad_procesada', label = 'alternativa 2', color = '
    green', ax = ax)

ax.set_xlabel('Minutos de procesamiento')
ax.set_ylabel('Cantidad de instrucciones procesadas')
plt.show()
```

A.2. Resolución ejercicio 2

A.2.1. ejercicio2.py

A.3. Resolución ejercicio 3

A.3.1. ejercicio3.py

A.4. Resolución ejercicio 4

A.4.1. ejercicio4.py

A.5. Resolución ejercicio 5

A.5.1. ejercicio5.py

```
import random
from random import randint
import simpy
import numpy
from numpy import mean


client_count = 10000
arrival_rate = 45
tiempo_esperas = []

class Client(object):

    def process_duration(self):
        if (self.type == 'A'):
            return (75 + randint(-10, 10))
        if (self.type == 'B'):
            return (60 + randint(-15, 15))
        return (90 + randint(-20, 20))


    def __init__(self, env):
        self.tiempo_entrada = env.now
        self.env = env
        self.type = numpy.random.choice(['A', 'B', 'C'], p=[0.6, 0.25, 0.15])

    def process(self, cashier):
        global tiempo_esperas
        with cashier.request() as req:
            process_duration = self.get_process_duration()
            yield req
            yield self.env.timeout(process_duration)

        # print("%.2f Client type %s attended" % (self.env.now, self.type))
        tiempo_esperas.append(env.now - self.tiempo_entrada)

    def get_process_duration(self):
        return self.process_duration()

class Balanceador(object):

    def __init__(self, round_robin):
        self.resources = []
        self.round_robin = round_robin
        self.last_assigned = 0

        for i in range(6):
            self.resources.append(simpy.Resource(env, capacity = 1))
            self.cantidad_colas = 6


    def get_resource(self):
        #If we use Round Robin politic
        if self.round_robin:
            self.last_assigned = (self.last_assigned + 1) % 6
            return self.resources[self.last_assigned]
        else:
            # If we assign randomly
            random_value = randint(0,5)
```

```
60              return_value = self.resources[random_value]
61
62              return return_value
63
64
65  def generate_clients(environment, count, interval):
66      #Select one the options in the exercise
67      balanceador_round_robin = Balanceador(False)
68      balanceador_2 = Balanceador(True)
69
70      balanceador = balanceador_2
71
72      for i in range(count):
73          client = Client(env)
74          environment.process(client.process(balanceador.get_resource()))
75          t = random.expovariate(1.0 / interval)
76          yield environment.timeout(t)
77
78
79  env = simpy.Environment()
80  env.process(generate_clients(env, client_count, arrival_rate))
81  env.run()
82
83  print("Tiempo espera: %s" % mean(tiempo_esperas))
84  print("Tiempo max: %s" % max(tiempo_esperas))
85  print("Tiempo min: %s" % min(tiempo_esperas))
86  print("Tiempo total: %s" % env.now)
```