TP1: Consiste en que cada uno de Ustedes establezca los parámetros requeridos para simular un calentador de agua eléctrico, de acuerdo a sus preferencias.

La idea que tenía para el calentador de agua es que sea un termito que pueda enchufar en el auto y calentar agua para los mates en los viajes, debido a esto los parámetros que tendré serán:

1. Material (aislante) a emplear: fibra de vidrio
2. Forma y capacidad del recipiente: cilíndrica (con un diámetro de 9 cm y 23,5 cm de alto), 1,5 litros.
3. Propósito del calentador: agua para el mate.
4. Fluido a calentar: agua.
5. Tiempo en el que se desea alcanzar esa temperatura: 7 minutos
6. Tensión de alimentación del dispositivo: 12 Volts.
7. Para este diseño, qué valor de Resistencia Eléctrica debemos emplear? 0,138 ohms

Para calcular la resistencia hicimos:

Primero, calculemos la cantidad de calor requerida:

$Q = mc\Delta T$

Donde:

$m = 1,5\ kg$ (la masa de agua, equivalente a $1,5\ litros$).

c es el calor específico del agua, que es aproximadamente $4180\ J/kg°C$.

$\Delta T = 85 - 15 = 70°C$ (la diferencia de temperatura deseada).

Sustituyendo los valores conocidos:

$Q = 1,5\ x\ 4180\ x\ 70$

$Q \approx 439830\ J$

Ahora, calculemos la potencia necesaria:

$P = \frac{Q}{t}$

Donde $t = 7\ min = 420\ segundos$

$P = \frac{439830}{420}$

$P \approx 1047,93\ vatios$

Finalmente, calculemos la resistencia eléctrica necesaria usando la ley de Ohm:

$R = \frac{V^2}{P}$

Donde $V = 12\ V$ (tensión de alimentación).

$R = \frac{12^2}{1047,93}$

$R \approx 0,138\ \Omega$

8. Cuál será la temperatura inicial del fluido al conectarlo al calentador? 15°C
9. Cuál será la temperatura ambiente al iniciar el proceso? 15°C
10. Calcular el aumento de temperatura del fluido luego de 1 segundo de conectar la alimentación, suponiendo que no existe pérdida de calor. 15,1664°C

Para calcularlo hacemos:

Primero, calculamos la potencia suministrada al calentador usando la ley de Ohm:

$P = \frac{V^2}{R}$

Donde: V=12 V (tensión de alimentación)

R=0.138 ohmios (resistencia eléctrica)

Entonces,

$P = \frac{12^2}{0,138}$

$P \approx 1047,93\ vatios$

La cantidad de calor (Q) transferida al agua en 1 segundo sería igual a la potencia suministrada, ya que la potencia es la energía transferida por unidad de tiempo:

$Q = P\ x\ t = 1043,48\ x\ 1$

$Q = 1043,48\ J$

Ahora, para encontrar el cambio en la temperatura del agua, usamos la fórmula del calor específico:

$Q = mc\Delta T$

Donde:

$m = 1,5\ kg$ (la masa de agua)

$c = 4180\ J/kg°C$ (calor específico del agua)

ΔT es el cambio en la temperatura.

Sustituyendo los valores conocidos:

$1043,48 = 1,5\ x\ 1000\ x\ 4,18\ x\ \Delta T$

Resolviendo para

$\Delta T = \frac{1043,48}{1,5\ x\ 1000\ x\ 4,18} \approx \frac{1043,48}{6270}$

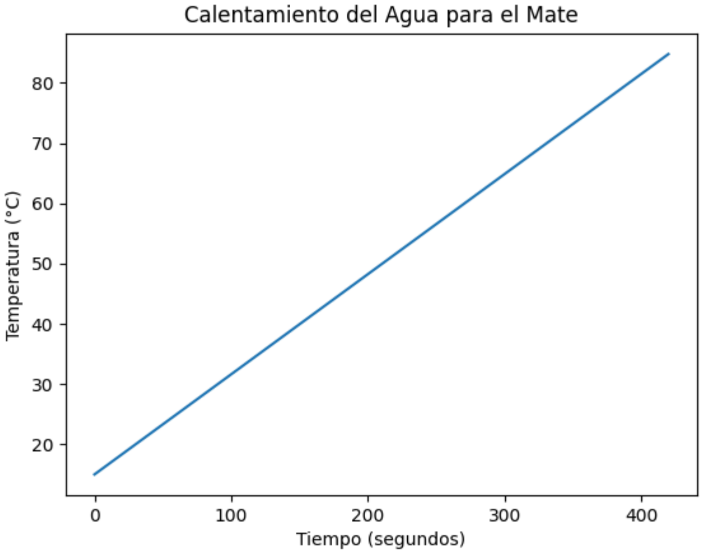
$\Delta T \approx 0,1664°C$

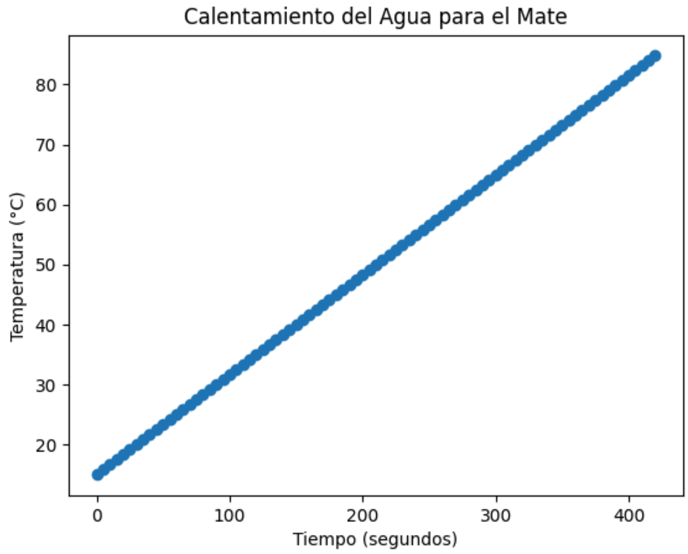
Entonces, la temperatura del agua después de 1 segundo de ser conectada a la tensión sería aproximadamente $15°C + 0,1664°C$, que es aproximadamente igual a $15,1664°C$.

La idea que tenía para el calentador de agua es que sea un termito que pueda enchufar en el auto y calentar agua para los mates en los viajes. Debido a que lo voy a usar en el auto el recipiente va a tener un diámetro de 9 cm y una altura de 23,5 cm, para poder apoyarlo en el apoya vasos del auto, y el material va hacer fibra de carbono, debido a su alta resistencia. Con una capacidad de 1,5 litros que es la medida de agua de la mayoría de los termos. Va a funcionar a una tensión de 12V, siendo esta la tensión de la batería del auto. El termo tiene que calentar el agua de 15°C a 85°C en 7 minutos, para poder lograrlo necesita una resistencia de 0,138Ω, luego de 1 segundo de enchufada a la alimentación su temperatura, sin pérdidas de calor, será 15,1664°C, podemos ver en el gráfico como aumenta la temperatura sin pérdidas de calor.

TP2: Graficar la temperatura en función del tiempo sin pérdidas de calor.

Con una separación de 5 segundos





El código, en python, para lograr el gráfico es el siguiente:

```
import matplotlib.pyplot as plt

# Parámetros iniciales

temperatura_inicial = 15 # Temperatura inicial del agua en °C

temperatura_ambiente = 15  # Temperatura ambiente en °C

potencia = 1043.48  # Potencia del calentador en Watts

resistencia = 0.138  # Resistencia del calentador en ohmios

capacidad_calorifica = 4.186  # Capacidad calorífica del agua en J/g°C

masa_agua = 1500  # Masa del agua en gramos (1.5 litro de agua)

tiempo_total = 420  # Tiempo total en segundos

# Lista para almacenar las temperaturas

temperaturas = [temperatura_inicial]

# Bucle para calcular la temperatura en cada segundo

for segundo in range(5, tiempo_total + 1, 5):

    delta_T = potencia / (masa_agua * capacidad_calorifica)
```

```
    nueva_temperatura = temperaturas[-1] + delta_T * 5   # Multiplicamos delta_T por 5 para tener el cambio en 5 segundos

    temperaturas.append(nueva_temperatura)

# Imprimir las temperaturas

for i, temp in enumerate(temperaturas):

    print(f"Segundo {i * 5}: {temp:.2f}°C")

# Crear la gráfica de dispersión

plt.scatter(range(0, tiempo_total + 1, 5), temperaturas, marker='o')

plt.xlabel('Tiempo (segundos)')

plt.ylabel('Temperatura (°C)')

plt.title('Calentamiento del Agua para el Mate')

plt.show()
```

TP3: Calcular la pérdida de calor de nuestro dispositivo, según las especificaciones de diseño. A modo de referencia, les comenté que un dispositivo de telgopor, de 1 litro de capacidad, y de espesor 1mm, suele presentar pérdidas aproximadas de 2,1 Watts/grado Kelvin.

$Pérdida\ de\ calor = \frac{CT\ x\ Sup}{Esp}$

Superficie lateral: $2\pi rh$

Superficie de las caras: πr^2

$Sup = 2\pi rh + 2\pi r^2 = 2\pi\ x\ 4,5\ cm\ x\ 23,5\ cm + 2\pi\ x\ (4,5\ cm)^2 = 791,68\ cm^2 = 0,079\ m^2$

Coeficiente Térmico de la fibra de vidrio CT: $0,04\frac{W}{mK}$

Espesor: $3\ mm = 0,003\ m$

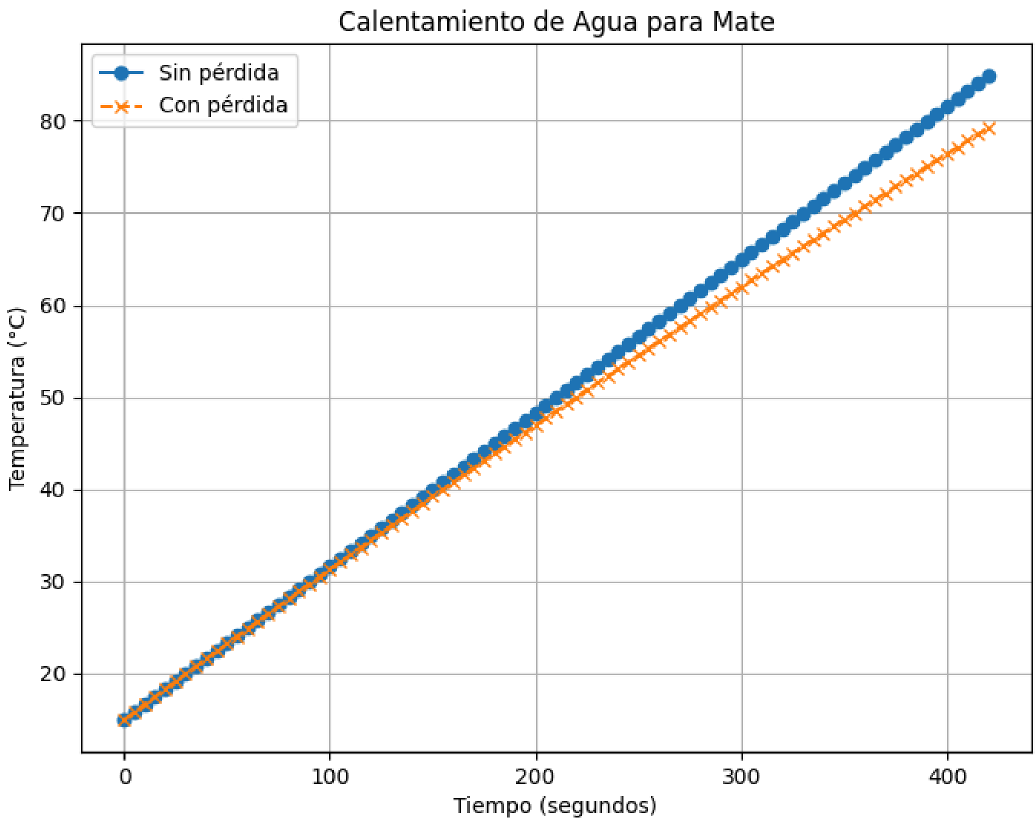
$Pérdida\ de\ calor = \frac{CT\ x\ Sup}{Esp} = \frac{0,04\frac{W}{mK}\ x\ 0,079\ m^2}{0,003\ m} = 1,053\frac{W}{K}$

TP4: Graficar la temperatura del fluido dentro del calentador sin pérdidas y con pérdidas para cada tick de tiempo, hasta llegar al tiempo deseado para que el dispositivo cumpla su tarea.

Fórmula:

$$temp\ nueva = temp\ inicial + tasa\ de\ aumento * tiempo - \frac{pérdida}{masa * calor\ esp}(temp\ anterior - 15) * tiempo$$

$$temp\ nueva = 15 + 0,1664 * tiempo - \frac{13288}{1.5*4180}(temp\ anterior - 15) * tiempo$$




```
import numpy as np

import matplotlib.pyplot as plt



# Datos

t_discreto = np.arange(0, 421, 5)  # Tiempo en segundos (discreto, intervalo de 5 segundos)

tasa_aumento = 0.1664  # Tasa de aumento de temperatura (°C por segundo)

temperatura_inicial = 15  # Temperatura inicial en °C

tasa_perdida_calor = 1.3288  # Tasa de pérdida de calor (W/°C)
```

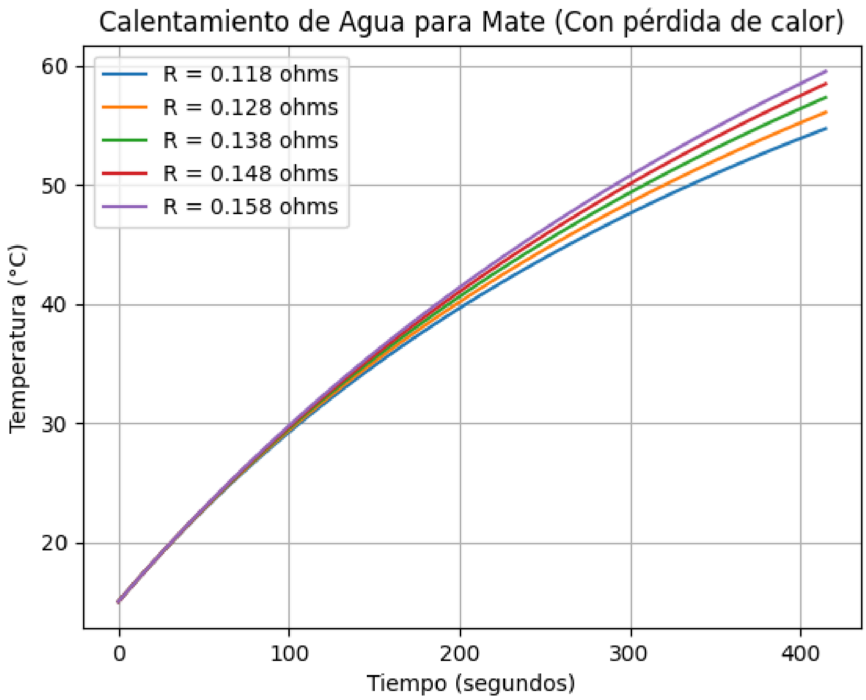
```python
masa_termo = 1.5  # Masa del termo en kilogramos

calor_especifico_agua = 4180  # Calor específico del agua en J/kg·°C


# Calculamos la temperatura en función del tiempo (sin pérdida de calor)

temperatura_sin_perdida = tasa_aumento * t_discreto + temperatura_inicial


# Calculamos la temperatura con pérdida

temperatura_con_perdida = []

temperatura_anterior = temperatura_inicial

for t in t_discreto:

     #delta_temp = (tasa_aumento * t - tasa_perdida_calor) / (masa_termo *
calor_especifico_agua)

            #temperatura_nueva   =   temperatura_anterior   +   delta_temp   *
(temperatura_anterior - temperatura_inicial)

    delta_temp = 1.3288 * t * (temperatura_anterior - 15) / (1.5 * 4180)

    temperatura_nueva = 15 + tasa_aumento * t - delta_temp

    temperatura_con_perdida.append(temperatura_nueva)

    temperatura_anterior = temperatura_nueva


# Graficamos ambas temperaturas

plt.figure(figsize=(8, 6))

plt.plot(t_discreto,  temperatura_sin_perdida,  marker="o",  linestyle="-",
label="Sin pérdida")

plt.plot(t_discreto,  temperatura_con_perdida,  marker="x",  linestyle="--",
label="Con pérdida")

plt.xlabel("Tiempo (segundos)")

plt.ylabel("Temperatura (°C)")
```

```
plt.title("Calentamiento de Agua para Mate")

plt.grid(True)

plt.legend()

plt.show()
```

TP5: Realizar familias de curvas con:

a. Distribución uniforme de 5 valores de resistencia (5 valores distintos cercanos al elegido)

   En el siguiente gráfico podemos observar como cambia el calentamiento del agua al ponerle diferentes valores de resistencia y cómo esto puede mejorar o empeorar el tiempo de calentamiento

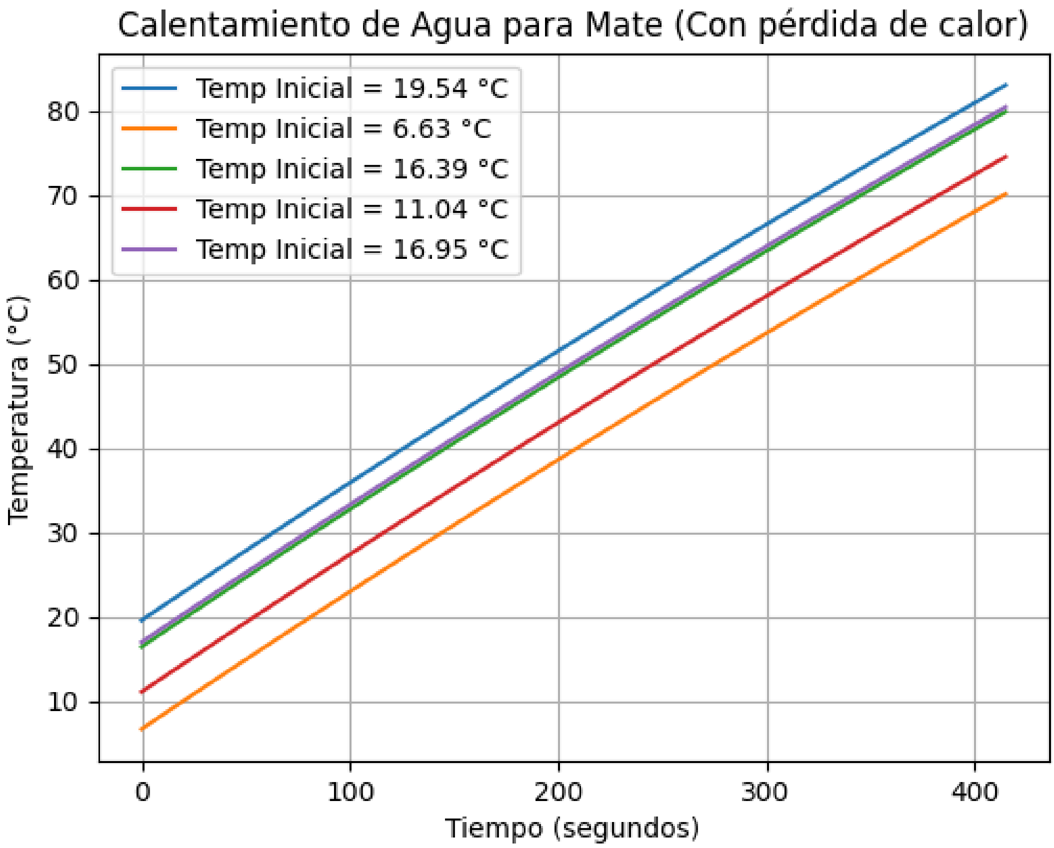



```
import numpy as np

import matplotlib.pyplot as plt



# Datos

t_discreto = np.arange(0, 420, 5)  # Tiempo en segundos (discreto, intervalo
de 5 segundos)

tasa_aumento = 0.1664  # Tasa de aumento de temperatura (°C por segundo)

temperatura_inicial = 15  # Temperatura inicial en °C

resistencia_inicial = 0.118  # Resistencia inicial en ohmios



# Valores de resistencia (con una diferencia de 0.010 ohms)
```

```
valores_resistencia = np.linspace(resistencia_inicial, resistencia_inicial +
0.010 * 4, 5)


# Crear una familia de curvas para diferentes resistencias

for resistencia in valores_resistencia:

    temperatura_con_perdida = []

    temperatura_anterior = temperatura_inicial

    for t in t_discreto:

         delta_temp = 1.3288 * t * (temperatura_anterior - 15) / (1.5 * 4180
* resistencia)

        temperatura_nueva = 15 + tasa_aumento * t - delta_temp

        temperatura_con_perdida.append(temperatura_nueva)

        temperatura_anterior = temperatura_nueva

             plt.plot(t_discreto,   temperatura_con_perdida,   label=f"R   =
{resistencia:.3f} ohms")


plt.xlabel("Tiempo (segundos)")

plt.ylabel("Temperatura (°C)")

plt.title("Calentamiento de Agua para Mate (Con pérdida de calor)")

plt.grid(True)

plt.legend()

plt.show()
```

b. Distribución normal de las temperatura iniciales del agua (con media 10 y desvío estándar de 5)

   En el siguiente gráfico veremos el comportamiento del calentador de agua en distintas temperaturas iniciales, teniendo en cuenta una distribución normal de la temperatura ambiente donde tenemos una media de 10 con una desviación estándar de 5.

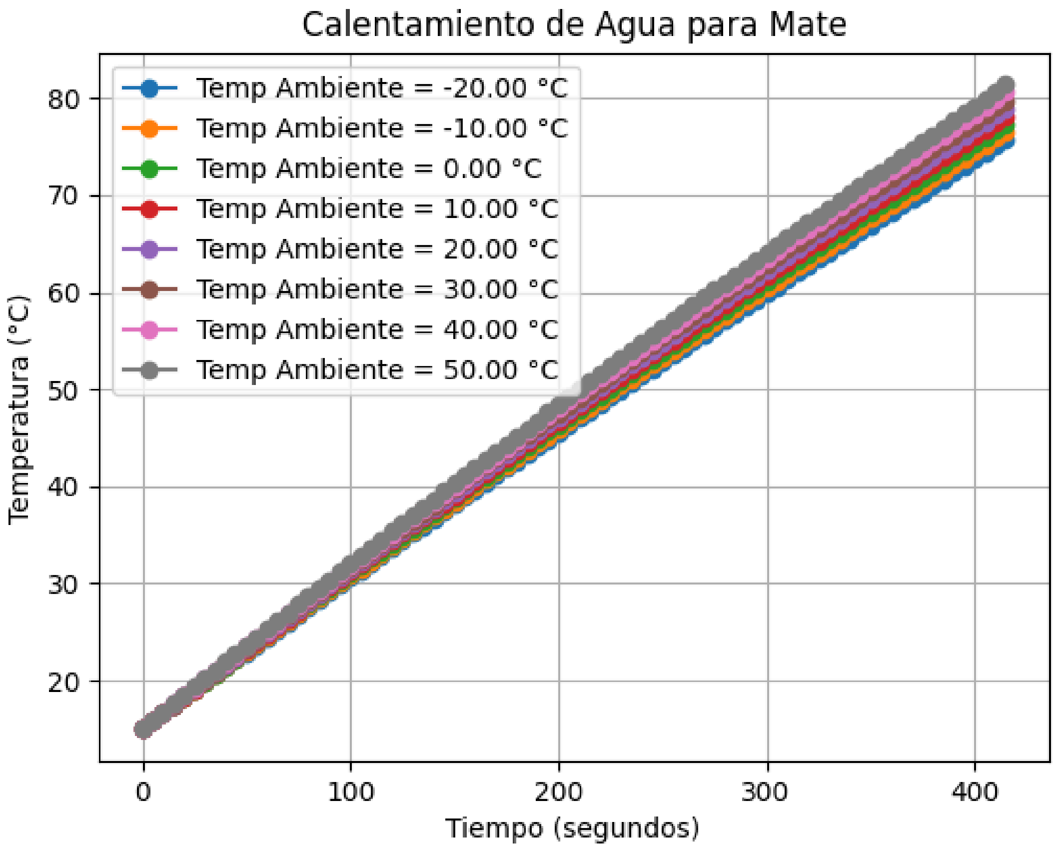



```
import numpy as np

import matplotlib.pyplot as plt



# Datos

t_discreto = np.arange(0, 420, 5)  # Tiempo en segundos (discreto, intervalo
de 5 segundos)

tasa_aumento = 0.1664  # Tasa de aumento de temperatura (°C por segundo)



# Generamos 5 valores de temperatura inicial siguiendo una distribución
normal

media_temperatura_inicial = 10

desviacion_estandar_temperatura_inicial = 5

num_valores = 5
```

```python
temperaturas_iniciales = np.random.normal(loc=media_temperatura_inicial, scale=desviacion_estandar_temperatura_inicial, size=num_valores)


# Calculamos la temperatura en función del tiempo para cada valor de temperatura inicial

for temperatura_inicial in temperaturas_iniciales:

    temperatura_con_perdida = []

    temperatura_anterior = temperatura_inicial

    for t in t_discreto:

        delta_temp = 1.3288 * t * (temperatura_anterior - temperatura_inicial) / (1.5 * 4180)

        temperatura_nueva = temperatura_inicial + tasa_aumento * t - delta_temp

        temperatura_con_perdida.append(temperatura_nueva)

        temperatura_anterior = temperatura_nueva

    plt.plot(t_discreto, temperatura_con_perdida, label=f"Temp Inicial = {temperatura_inicial:.2f} °C")


plt.xlabel("Tiempo (segundos)")

plt.ylabel("Temperatura (°C)")

plt.title("Calentamiento de Agua para Mate (Con pérdida de calor)")

plt.grid(True)

plt.legend()

plt.show()
```

c. Distribución uniforme de 8 temperaturas iniciales del ambiente (entre -20 y 50 grados)

En el gráfico podemos observar como la temperatura ambiente nos va afectar el calentamiento del agua, el gráfico nos muestra una serie de 8 temperaturas ambientales menores y mayores a la óptima decidida con anterioridad.

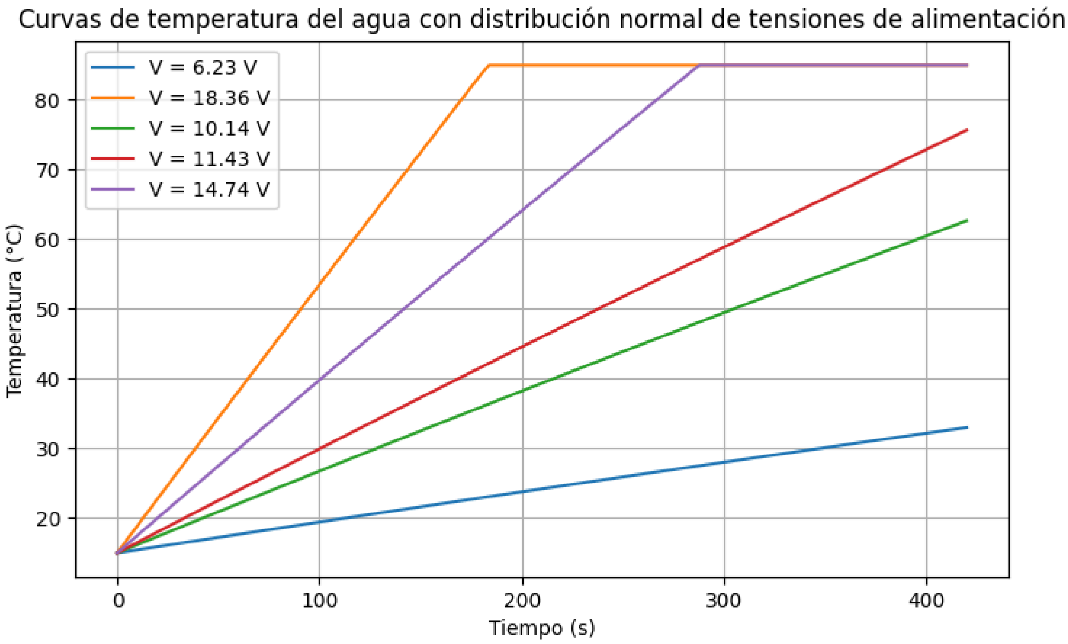



```
import numpy as np

import matplotlib.pyplot as plt



# Datos

t_discreto = np.arange(0, 420, 5)  # Tiempo en segundos (discreto, intervalo
de 5 segundos)

tasa_aumento = 0.1664  # Tasa de aumento de temperatura (°C por segundo)

temperatura_inicial_agua = 15  # Temperatura inicial del agua en °C

temperaturas_ambiente = np.linspace(-20, 50, 8)  # Valores de temperatura
ambiente



# Crear una familia de curvas para diferentes temperaturas ambiente
```

```
for temperatura_ambiente in temperaturas_ambiente:

    temperatura_con_perdida = []

    temperatura_anterior = temperatura_inicial_agua

    for t in t_discreto:

                  delta_temp  =  1.3288  *  t  *  (temperatura_anterior  -
temperatura_ambiente) / (1.5 * 4180)

          temperatura_nueva = temperatura_inicial_agua + tasa_aumento * t -
delta_temp

        temperatura_con_perdida.append(temperatura_nueva)

        temperatura_anterior = temperatura_nueva

     plt.plot(t_discreto, temperatura_con_perdida, marker="o", linestyle="-",
label=f"Temp Ambiente = {temperatura_ambiente:.2f} °C")



plt.xlabel("Tiempo (segundos)")

plt.ylabel("Temperatura (°C)")

plt.title("Calentamiento de Agua para Mate")

plt.grid(True)

plt.legend()

plt.show()
```

d. Distribución normal de valores de tensión (media de 12V y desvío estándar de 4)

   Este gráfico nos muestra como va a variar el tiempo de calentamiento ante posibles inestabilidades de la tensión, para poder hacer esto calculamos nuevamente la potencia que tendría para cada tensión en particular y con eso el descenso de temperatura que va a tener para cada nueva variación de temperatura.

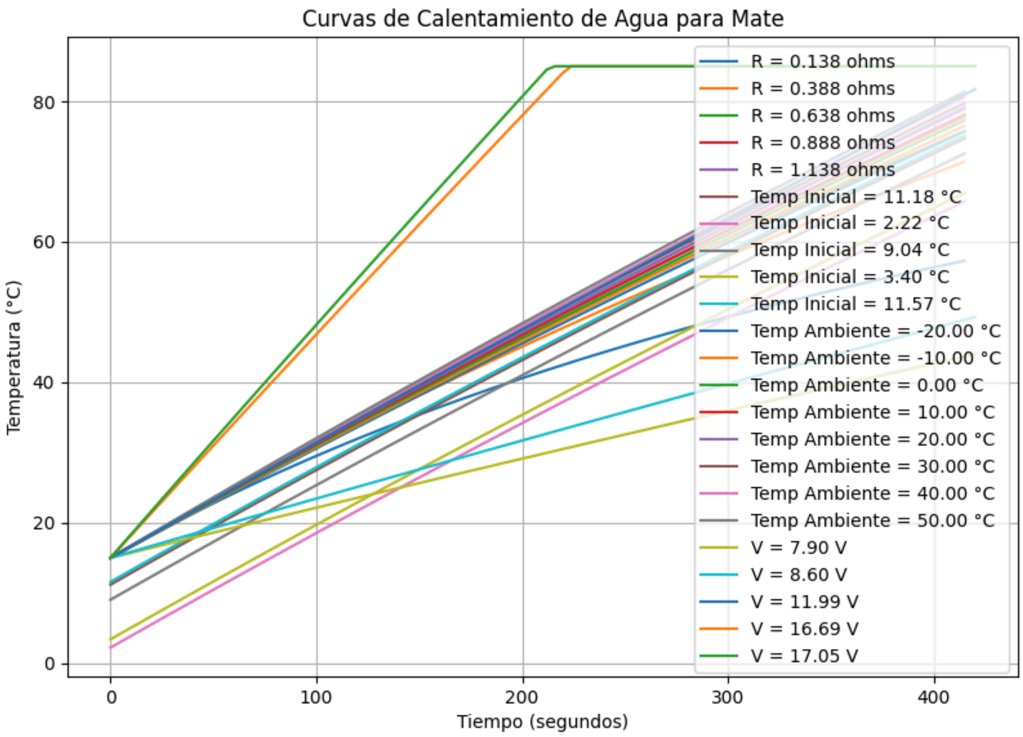



```
import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import odeint



# Datos

t_discreto = np.arange(0, 420, 5)   # Tiempo en segundos (discreto, intervalo de 5 segundos)

tasa_aumento = 0.1664   # Tasa de aumento de temperatura (°C por segundo)

Ta = 15  # Temperatura inicial en °C

k = 1.3288   # K/ °C

R = 0.138    # ohm

c = 4.186  # capacidad calorifica del agua

m = 1500    # en g

tiempo = 420
```

```python
# Generamos 5 valores de temperatura inicial siguiendo una distribución normal

media_tension= 12

desviacion_estandar_tension= 4

num_valores = 5

tensiones = np.random.normal(loc=media_tension, scale=desviacion_estandar_tension, size=num_valores)




# Definir la ecuación diferencial para el cambio de temperatura del agua

def dTdt(T, t, V):

    P = V**2 / R  # Potencia en función de la tensión y la resistencia

    dTdt = P/(m*c)- (k*(T - Ta) ) / (m*c)

    if T >= 85:

            dTdt = 0  # Detener el calentamiento si se alcanza la temperatura máxima

    return dTdt



# Resolver la ecuación diferencial numéricamente y graficar para cada tensión de alimentación

for V in tensiones:

    T_values = odeint(dTdt, Ta, np.arange(0, tiempo + 4, 4), args=(V,))

    plt.plot(np.arange(0, tiempo + 4, 4), T_values, label=f"V = {V:.2f} V")



# Personalización de la gráfica
```

```
plt.title("Curvas de temperatura del agua con distribución normal de tensiones de alimentación")

plt.xlabel("Tiempo (s)")

plt.ylabel("Temperatura (°C)")

plt.legend()

plt.grid(True)

plt.show()
```

e. Simulaciones que contengan todas las familias de curvas previas



import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import odeint

Datos comunes

```
t_discreto = np.arange(0, 420, 5)  # Tiempo en segundos (discreto, intervalo de 5 segundos)

tasa_aumento = 0.1664  # Tasa de aumento de temperatura (°C por segundo)


# Código 1: Variación de resistencia

temperatura_inicial = 15  # Temperatura inicial en °C

resistencia_inicial = 0.138  # Resistencia inicial en ohmios

valores_resistencia = np.linspace(resistencia_inicial, resistencia_inicial + 0.25 * 4, 5)


for resistencia in valores_resistencia:

    temperatura_con_perdida = []

    temperatura_anterior = temperatura_inicial

    for t in t_discreto:

        delta_temp = 1.3288 * t * (temperatura_anterior - 15) / (1.5 * 4180 * resistencia)

        temperatura_nueva = 15 + tasa_aumento * t - delta_temp

        temperatura_con_perdida.append(temperatura_nueva)

        temperatura_anterior = temperatura_nueva

    plt.plot(t_discreto, temperatura_con_perdida, label=f"R = {resistencia:.3f} ohms")


# Código 2: Variación de temperatura inicial

media_temperatura_inicial = 10

desviacion_estandar_temperatura_inicial = 5

num_valores = 5

temperaturas_iniciales = np.random.normal(loc=media_temperatura_inicial, scale=desviacion_estandar_temperatura_inicial, size=num_valores)


for temperatura_inicial in temperaturas_iniciales:
```

```
    temperatura_con_perdida = []

    temperatura_anterior = temperatura_inicial

    for t in t_discreto:

        delta_temp = 1.3288 * t * (temperatura_anterior - temperatura_inicial) / (1.5 * 4180)

        temperatura_nueva = temperatura_inicial + tasa_aumento * t - delta_temp

        temperatura_con_perdida.append(temperatura_nueva)

        temperatura_anterior = temperatura_nueva

            plt.plot(t_discreto, temperatura_con_perdida, label=f"Temp Inicial = {temperatura_inicial:.2f} °C")


# Código 3: Variación de temperatura ambiente

temperatura_inicial_agua = 15 # Temperatura inicial del agua en °C

temperaturas_ambiente = np.linspace(-20, 50, 8)  # Valores de temperatura ambiente


for temperatura_ambiente in temperaturas_ambiente:

    temperatura_con_perdida = []

    temperatura_anterior = temperatura_inicial_agua

    for t in t_discreto:

        delta_temp = 1.3288 * t * (temperatura_anterior - temperatura_ambiente) / (1.5 * 4180)

        temperatura_nueva = temperatura_inicial_agua + tasa_aumento * t - delta_temp

        temperatura_con_perdida.append(temperatura_nueva)

        temperatura_anterior = temperatura_nueva

        plt.plot(t_discreto, temperatura_con_perdida, linestyle="-", label=f"Temp Ambiente = {temperatura_ambiente:.2f} °C")


# Código 4: Variación de tensión

Ta = 15  # Temperatura inicial en °C
```

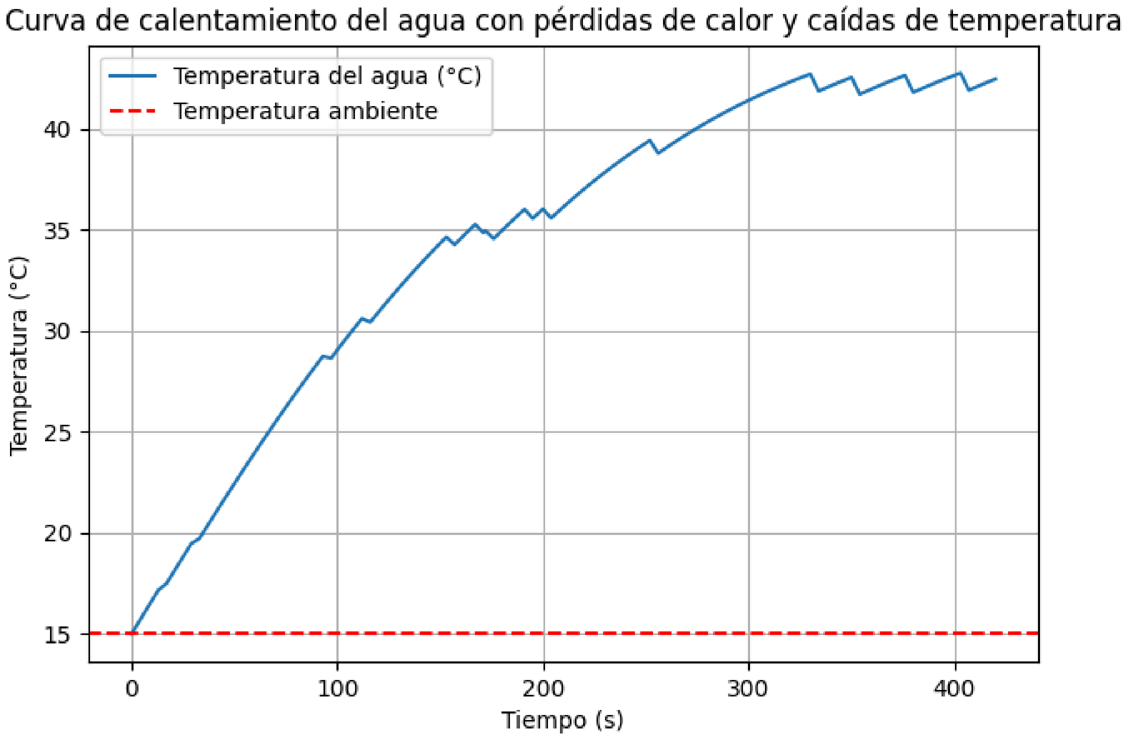
```
k = 1.3288  # K/ °C

R = 0.138  # ohm

c = 4.186  # capacidad calorífica del agua

m = 1500  # en g

tiempo = 420

media_tension = 12

desviacion_estandar_tension = 4

tensiones = np.random.normal(loc=media_tension, scale=desviacion_estandar_tension, size=num_valores)


def dTdt(T, t, V):

    P = V**2 / R  # Potencia en función de la tensión y la resistencia

    dTdt = P / (m * c) - (k * (T - Ta)) / (m * c)

    if T >= 85:

        dTdt = 0  # Detener el calentamiento si se alcanza la temperatura máxima

    return dTdt


for V in tensiones:

    T_values = odeint(dTdt, Ta, np.arange(0, tiempo + 4, 4), args=(V,))

    plt.plot(np.arange(0, tiempo + 4, 4), T_values, label=f"V = {V:.2f} V")


# Personalización de la gráfica

plt.xlabel("Tiempo (segundos)")

plt.ylabel("Temperatura (°C)")

plt.title("Curvas de Calentamiento de Agua para Mate")

plt.grid(True)

plt.legend()
```

```
plt.show()
```

TP6: x segundos después de iniciar el experimento y durante “y” segundos llega un frente frío que hace descender la temperatura externa z grados. Rehacer el gráfico de temperaturas.

La probabilidad de que llegue este frente frío durante la realización del experimento en cada tick es de 1/300.

En la siguiente curva podemos observar como se eleva la temperatura cuando entran distintos tipos de frentes fríos, que hacen que se ralentice el calentamiento del agua.



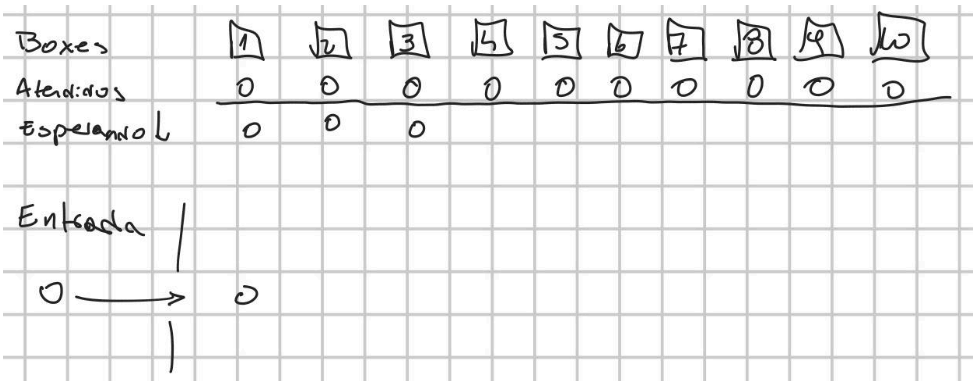

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import odeint


# Constantes

R = 0.138 # En Ohms

V = 12

P = (V*V)/R  # Potencia suministrada (W)

m = 1500     # Masa de agua (g)
```

```
c = 4.18      # Capacidad calorífica del agua (J/g°C)

k_ambiente = 1.053   # Coeficiente de transferencia de calor (W/K)

Ta = 15       # Temperatura ambiente (°C)

T0 = 15       # Temperatura inicial del agua (°C)

tiempo_final = 420  # segundos


# Variables para la caída de temperatura

probabilidad_caida = 3 / 100

duracion_caida = 5

caida_temperatura = 20


# Definir la ecuación diferencial para el cambio de temperatura del agua

def dTdt(T, t, Ta_actual):

    k = k_ambiente * (T - Ta_actual)

    dTdt = P/(m*c) - (k*(T - Ta_actual)) / (m*c)

    if T >= 1000:

          dTdt = 0  # Detener el calentamiento si se alcanza la temperatura
máxima

    return dTdt


# Generar valores de tiempo

t_values = np.arange(0, tiempo_final + 1, 1)


# Inicializar la temperatura

T_values = [T0]
```

```
Ta_actual = Ta

tiempo_caida = 0


# Simular la dinámica de la temperatura

for t in t_values[1:]:

    # Verificar si se debe iniciar una caída de temperatura

    if np.random.rand() < probabilidad_caida and tiempo_caida == 0:

        tiempo_caida = duracion_caida

        Ta_actual -= caida_temperatura


    # Actualizar la temperatura ambiente si la caída está en efecto

    if tiempo_caida > 0:

        tiempo_caida -= 1

        if tiempo_caida == 0:

            Ta_actual = Ta  # Restaurar la temperatura ambiente original


    # Resolver la ecuación diferencial para el siguiente paso

    T = odeint(dTdt, T_values[-1], [t-1, t], args=(Ta_actual,))[-1, 0]

    T_values.append(T)


# Graficar la curva de calentamiento

plt.plot(t_values, T_values, label='Temperatura del agua (°C)')

plt.xlabel('Tiempo (s)')

plt.ylabel('Temperatura (°C)')
```

```
plt.title('Curva de calentamiento del agua con pérdidas de calor y caídas de temperatura')

plt.axhline(Ta, color='r', linestyle='--', label='Temperatura ambiente')

plt.legend()

plt.grid(True)

plt.show()
```

TP7: Modelo de Atencion al publico



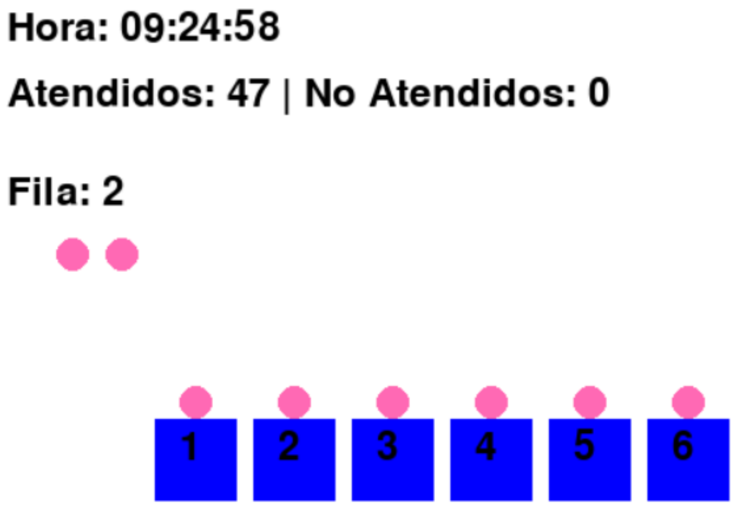
Consignas:

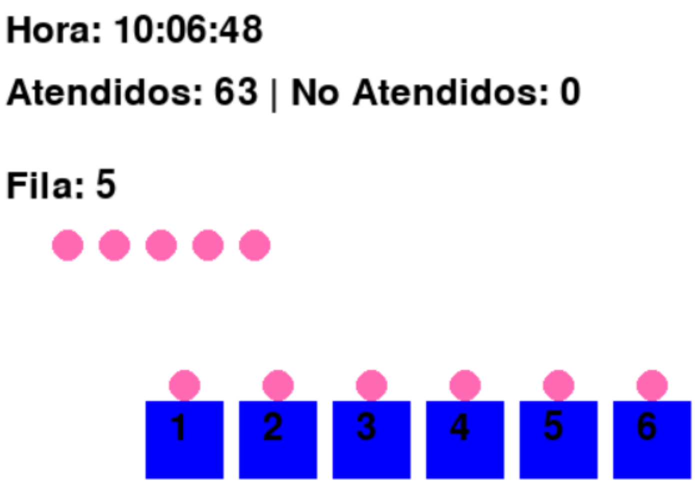
1. El servicio abre a las 8 y cierra a las 12.
2. Los clientes que están en la cola o siendo atendidos pueden permanecer dentro del local después del cierre.
3. Los que no estén siendo atendidos abandonan el local a los 30 minutos.
4. En cada segundo que transcurre la probabilidad de que ingrese un cliente es $p = \frac{1}{144}$.
5. Los boxes activos se establecen al inicio de la simulación (1-10).
6. Tiempo de atención en boxes ($\mu = 10\, min\, ;\ SD = 5\, min$).
7. Cada box cuesta $1000 abrirlo.
8. Cada cliente no atendido se pierde, con un costo de $10000.

Preguntas a resolver:

a. Cuántos clientes ingresaron?
b. Cuántos fueron atendidos?
c. Cuántos se fueron sin ser atendidos?
d. Costo total de la operación (sólo consideró los costos del 7 y 8)
e. Tiempo máximo de atención
f. Tiempo máximo de espera dentro del local
g. Graficar la distribución de personas en el local tiempo - animacion, instante por instante

En este trabajo práctico lo que hice fue realizar una simulación que me muestre el comportamiento de atención al público, para poder ver la simulación yo tengo que especificar la cantidad de boxes abiertos y si quiero que la simulación transcurra en tiempo real o a un mayor tiempo. Además luego de la simulación nos muestra un histograma para así poder ver la distribución de clientes mientras el local está funcionando. Luego de que la simulación termina, se guarda un video con la simulación en nuestra computadora.
A continuación se ven las fotos de la simulación, el histograma y todos los datos que nos dan al final de la simulación.





Clientes ingresados: 113

Clientes atendidos: 113

Clientes no atendidos: 0

Tiempo mínimo de atención en box: 0.00 minutos

Tiempo máximo de atención en box: 21.36 minutos

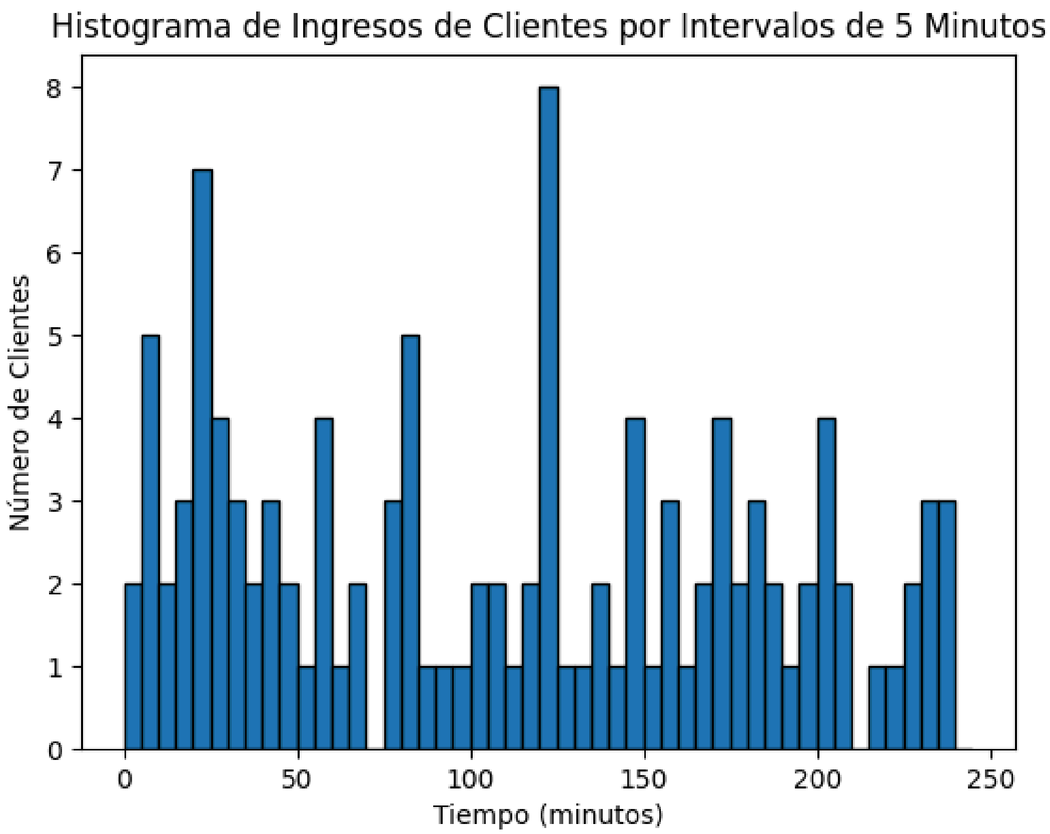
Tiempo máximo de espera experimentado por un cliente: 8.52 minu

Tiempo máximo de espera permitido: 30 minutos

Costo de la operación: $6000.00

Aca podemos ver que nos indica la terminal
Ingrese la cantidad de boxes (entre 1 y 10): 6
Ingrese la velocidad de la simulación (1.0 = tiempo real, 2.0 = el doble de rápido, etc.): 5
Clientes ingresados: 113
Clientes atendidos: 113

Clientes no atendidos: 0
Tiempo mínimo de atención en box: 0.00 minutos
Tiempo máximo de atención en box: 21.36 minutos
Tiempo máximo de espera experimentado por un cliente: 8.52 minutos
Tiempo máximo de espera permitido: 30 minutos
Costo de la operación: $6000.00
Y el histograma que nos muestra es el siguiente:



El código utilizado fue:

```
import pygame
import random
import cv2
import numpy as np
from collections import deque
import matplotlib.pyplot as plt

# Parámetros de la simulación
horas_apertura = 4
probabilidad_ingreso = 1 / 144
media_atencion = 10  # minutos
desvio_atencion = 5  # minutos
costo_box = 1000
costo_perdida_cliente = 10000
```

```python
tiempo_maximo_espera = 30  # minutos

# Inicialización de Pygame
pygame.init()
width, height = 800, 600
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Simulación de Local")
clock = pygame.time.Clock()

# Colores
pink = (255, 105, 180)
blue = (0, 0, 255)
black = (0, 0, 0)
white = (255, 255, 255)

# Crear video writer
fourcc = cv2.VideoWriter_fourcc(*'XVID')
video = cv2.VideoWriter('simulacion3.avi', fourcc, 30.0, (width, height))

# Función para dibujar el estado actual
def draw_state(second, fila_espera, clientes_en_atencion, clientes_atendidos,
clientes_no_atendidos):
    screen.fill(white)

    # Dibujar fila de espera
    fila_text = f"Fila: {len(fila_espera)}"
    fila_surface = pygame.font.SysFont(None, 36).render(fila_text, True,
black)
    screen.blit(fila_surface, (10, 150))

    # Dibujar los clientes en fila
    for i, cliente in enumerate(fila_espera):
        pygame.draw.circle(screen, pink, (50 + i * 30, 200), 10)

    # Dibujar los boxes y clientes siendo atendidos
    for i, cliente in enumerate(clientes_en_atencion):
        pygame.draw.rect(screen, blue, (100 + i * 60, 300, 50, 50))
        box_text = f"{i+1}"
        box_surface = pygame.font.SysFont(None, 36).render(box_text, True,
black)
        screen.blit(box_surface, (115 + i * 60, 305))
        if cliente is not None:
            pygame.draw.circle(screen, pink, (125 + i * 60, 290), 10)
```

```python
    # Dibujar la hora actual
    hora_apertura_segundos = 8 * 3600  # 08:00:00 en segundos
    tiempo_actual = hora_apertura_segundos + second
    horas, minutos, segundos = tiempo_actual // 3600, (tiempo_actual % 3600) // 60, tiempo_actual % 60
    time_text = f"Hora: {horas:02d}:{minutos:02d}:{segundos:02d}"
    time_surface = pygame.font.SysFont(None, 36).render(time_text, True, black)
    screen.blit(time_surface, (10, 50))

    # Dibujar la cantidad de clientes atendidos y no atendidos
    atendidos_text = f"Atendidos: {clientes_atendidos} | No Atendidos: {clientes_no_atendidos}"
    atendidos_surface = pygame.font.SysFont(None, 36).render(atendidos_text, True, black)
    screen.blit(atendidos_surface, (10, 90))

    pygame.display.flip()

    # Guardar frame en el video
    frame = pygame.surfarray.array3d(pygame.display.get_surface())
    frame = frame.transpose([1, 0, 2])
    video.write(cv2.cvtColor(frame, cv2.COLOR_RGB2BGR))

def simular_local():
    try:
        num_boxes = int(input("Ingrese la cantidad de boxes (entre 1 y 10): "))
        if num_boxes < 1 or num_boxes > 10:
            raise ValueError("Número de boxes inválido. Debe estar entre 1 y 10.")
    except ValueError as e:
        print(e)
        return

    try:
        velocidad_simulacion = float(input("Ingrese la velocidad de la simulación (1.0 = tiempo real, 2.0 = el doble de rápido, etc.): "))
        if velocidad_simulacion <= 0:
            raise ValueError("La velocidad de la simulación debe ser mayor que 0.")
    except ValueError as e:
```

```
        print(e)
        return

    # Inicialización
    clientes_ingresados = 0
    clientes_atendidos = 0
    clientes_no_atendidos = 0
    tiempo_minimo_atencion = float('inf')
    tiempo_maximo_atencion = 0
    tiempo_maximo_espera_actual = 0

    # Lista para rastrear los clientes actualmente siendo atendidos en cada
box
    clientes_en_atencion = [None] * num_boxes
    # Fila de espera de clientes (almacena el tiempo de ingreso de cada
cliente)
    fila_espera = deque()
    # Lista para almacenar los tiempos de ingreso de los clientes
    tiempos_ingreso = []

    segundos_totales = horas_apertura * 3600
    segundo = 0
    running = True

    while running and (segundo < segundos_totales or fila_espera or
any(clientes_en_atencion)):
        # Manejo de eventos de Pygame
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        # Simulación de ingreso de cliente dentro del horario de apertura
        if segundo < segundos_totales and random.random() <
probabilidad_ingreso:
            clientes_ingresados += 1
            fila_espera.append(segundo)
            tiempos_ingreso.append(segundo)  # Almacenar el tiempo de ingreso
del cliente

        # Intentar asignar boxes libres a clientes en espera
        for i in range(num_boxes):
            if clientes_en_atencion[i] is None and fila_espera:
                tiempo_espera = segundo - fila_espera[0]
```

```
                if tiempo_espera <= tiempo_maximo_espera * 60:
                    # Simulación de atención
                    tiempo_atencion =
max(random.normalvariate(media_atencion, desvio_atencion), 0) * 60  #
Convertir a segundos
                    tiempo_minimo_atencion = min(tiempo_minimo_atencion,
tiempo_atencion)
                    tiempo_maximo_atencion = max(tiempo_maximo_atencion,
tiempo_atencion)

                    clientes_en_atencion[i] = tiempo_atencion
                    clientes_atendidos += 1
                    tiempo_maximo_espera_actual =
max(tiempo_maximo_espera_actual, tiempo_espera)
                    fila_espera.popleft()
                else:
                    fila_espera.popleft()
                    clientes_no_atendidos += 1

        # Actualizar el tiempo restante de atención para los clientes en cada
box
        for i in range(num_boxes):
            if clientes_en_atencion[i] is not None:
                clientes_en_atencion[i] -= 1
                if clientes_en_atencion[i] <= 0:
                    clientes_en_atencion[i] = None

        # Dibujar el estado actual
        draw_state(segundo, fila_espera, clientes_en_atencion,
clientes_atendidos, clientes_no_atendidos)
        clock.tick(30 * velocidad_simulacion)  # Ajustar la velocidad de la
simulación

        segundo += 1

    # Cálculo del costo total
    costo_total = num_boxes * costo_box + clientes_no_atendidos *
costo_perdida_cliente

    # Resultados
    print(f"Clientes ingresados: {clientes_ingresados}")
    print(f"Clientes atendidos: {clientes_atendidos}")
    print(f"Clientes no atendidos: {clientes_no_atendidos}")
```

```python
    print(f"Tiempo mínimo de atención en box: {tiempo_minimo_atencion / 60:.2f} minutos")
    print(f"Tiempo máximo de atención en box: {tiempo_maximo_atencion / 60:.2f} minutos")
    print(f"Tiempo máximo de espera experimentado por un cliente: {tiempo_maximo_espera_actual / 60:.2f} minutos")
    print(f"Tiempo máximo de espera permitido: {tiempo_maximo_espera} minutos")
    print(f"Costo de la operación: ${costo_total:.2f}")

    # Mostrar los resultados finales en la animación
    resultados_texto = [
        f"Clientes ingresados: {clientes_ingresados}",
        f"Clientes atendidos: {clientes_atendidos}",
        f"Clientes no atendidos: {clientes_no_atendidos}",
        f"Tiempo mínimo de atención en box: {tiempo_minimo_atencion / 60:.2f} minutos",
        f"Tiempo máximo de atención en box: {tiempo_maximo_atencion / 60:.2f} minutos",
        f"Tiempo máximo de espera experimentado por un cliente: {tiempo_maximo_espera_actual / 60:.2f} minutos",
        f"Tiempo máximo de espera permitido: {tiempo_maximo_espera} minutos",
        f"Costo de la operación: ${costo_total:.2f}"
    ]

    screen.fill(white)
    for i, linea in enumerate(resultados_texto):
        resultado_surface = pygame.font.SysFont(None, 36).render(linea, True, black)
        screen.blit(resultado_surface, (10, 50 + i * 40))
    pygame.display.flip()
    pygame.time.wait(5000)

    # Guardar frame en el video
    frame = pygame.surfarray.array3d(pygame.display.get_surface())
    frame = frame.transpose([1, 0, 2])
    video.write(cv2.cvtColor(frame, cv2.COLOR_RGB2BGR))

    # Liberar el video
    video.release()
    pygame.quit()

    # Crear y mostrar el histograma de ingresos de clientes cada 5 minutos
```

```
    tiempos_ingreso_minutos = [t // 60 for t in tiempos_ingreso]  # Convertir tiempos a minutos
    plt.figure()
    plt.hist(tiempos_ingreso_minutos, bins=range(0, (segundos_totales // 60) + 6, 5), edgecolor='black')  # Bins de 5 minutos
    plt.xlabel('Tiempo (minutos)')
    plt.ylabel('Número de Clientes')
    plt.title('Histograma de Ingresos de Clientes por Intervalos de 5 Minutos')
    plt.show()

if __name__ == "__main__":
    simular_local()
```

TP8: Modelo y Simulación de un Sistema de Atención al Público. Distribución Normal de Afluencia.

Consiste en modificar el TP7 de modo que la afluencia del público responda a una distribución normal, con media 10 de la mañana, y desvío estándar de 2 horas. Despreciando las colas izquierda y derecha de la distribución normal por debajo de las 8 horas y por encima de las 12 horas de la mañana.

La esperanza matemática continúa siendo de 100 personas en el transcurso de la mañana.

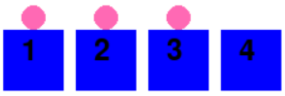
En este trabajo práctico lo que hice fue realizar una simulación que me muestre el comportamiento de atención al público, cuando los clientes entran al local con una distribución normal teniendo una media a las 10 de la mañana y una desviación estándar de 2 horas. Para poder ver la simulación yo tengo que especificar la cantidad de boxes abiertos y si quiero que la simulación transcurra en tiempo real o a un mayor tiempo. Además luego de la simulación nos muestra un histograma para así poder ver la distribución normal de clientes mientras el local está funcionando. Luego de que la simulación termina, se guarda un video con la simulación en nuestra computadora.

A continuación se ven las fotos de la simulación, el histograma y todos los datos que nos dan al final de la simulación.

Hora: 09:02:05

Atendidos: 16 | No Atendidos: 0

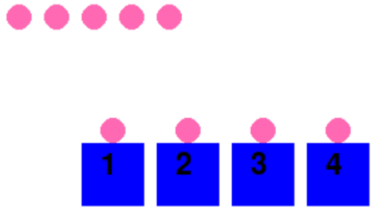
Fila: 0



Hora: 10:12:32

Atendidos: 45 | No Atendidos: 0

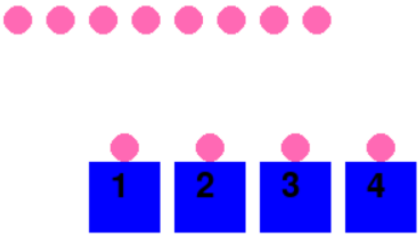
Fila: 5



Hora: 10:38:57

Atendidos: 57 | No Atendidos: 0

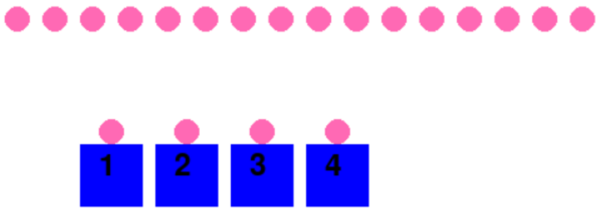
Fila: 8



Hora: 11:00:36

Atendidos: 67 | No Atendidos: 0

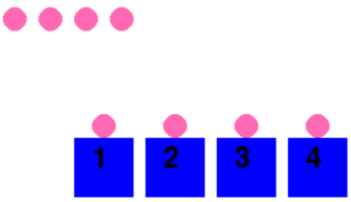
Fila: 16



Hora: 12:00:24

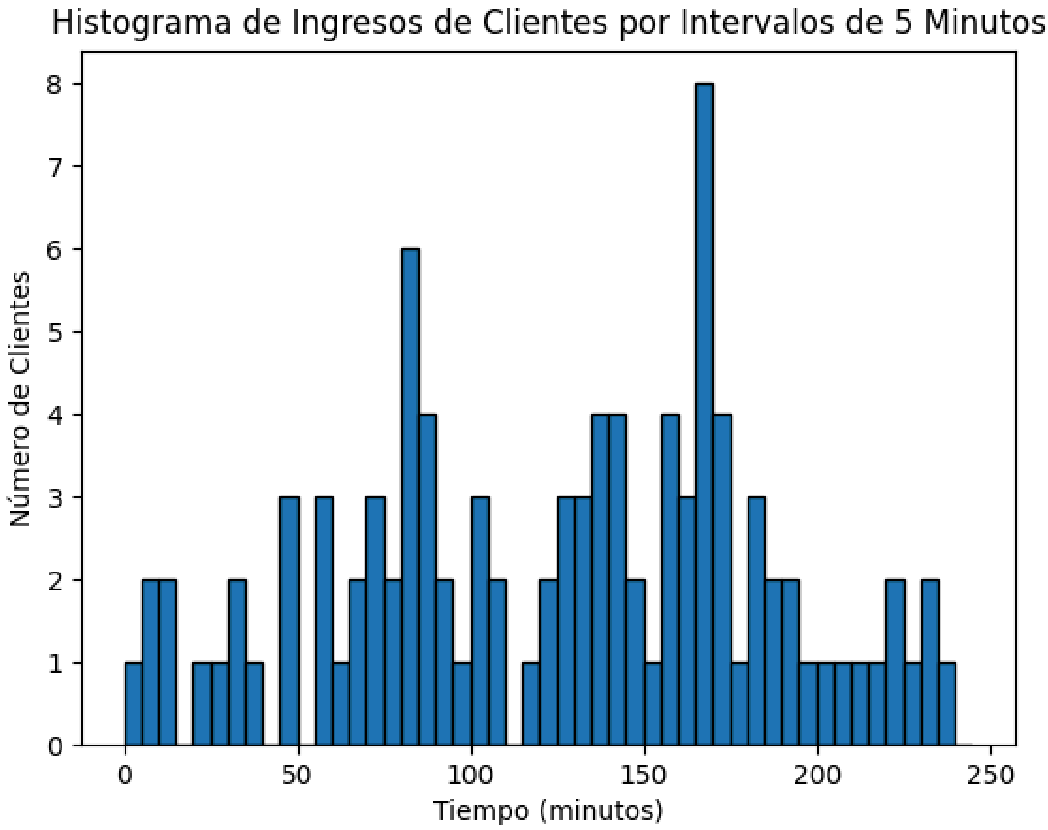
Atendidos: 90 | No Atendidos: 6

Fila: 4



Clientes ingresados: 100
Clientes atendidos: 94
Clientes no atendidos: 6
Tiempo mínimo de atención en box: 0.00 minutos
Tiempo máximo de atención en box: 20.00 minutos
Tiempo máximo de espera experimentado por un cliente: 30.00 min
Tiempo máximo de espera permitido: 30 minutos
Costo de la operación: $64000.00

El histograma que nos muestra es el siguiente:



Aca podemos ver que nos indica la terminal, donde también se muestra el costo de la operación:
Ingrese la cantidad de boxes (entre 1 y 10): 4
Ingrese la velocidad de la simulación (1.0 = tiempo real, 2.0 = el doble de rápido, etc.): 5
Clientes ingresados: 100
Clientes atendidos: 94
Clientes no atendidos: 6
Tiempo mínimo de atención en box: 0.00 minutos
Tiempo máximo de atención en box: 20.00 minutos
Tiempo máximo de espera experimentado por un cliente: 30.00 minutos
Tiempo máximo de espera permitido: 30 minutos
Costo de la operación: $64000.00

El código que utilice fue:

```
import pygame
import random
import cv2
import numpy as np
from collections import deque
import matplotlib.pyplot as plt

# Parámetros de la simulación
horas_apertura = 4
```

```
media_ingreso = 120  # minutos
desvio_ingreso = 120  # minutos
media_atencion = 10  # minutos
desvio_atencion = 5  # minutos
costo_box = 1000
costo_perdida_cliente = 10000
tiempo_maximo_espera = 30  # minutos
esperanza_clientes = 100  # clientes

# Inicialización de Pygame
pygame.init()
width, height = 800, 600
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Simulación de Local")
clock = pygame.time.Clock()

# Colores
pink = (255, 105, 180)
blue = (0, 0, 255)
black = (0, 0, 0)
white = (255, 255, 255)

# Crear video writer
fourcc = cv2.VideoWriter_fourcc(*'XVID')
video = cv2.VideoWriter('simulacion3.avi', fourcc, 30.0, (width,
height))

# Función para dibujar el estado actual
def draw_state(second, fila_espera, clientes_en_atencion,
clientes_atendidos, clientes_no_atendidos):
    screen.fill(white)

    # Dibujar fila de espera
    fila_text = f"Fila: {len(fila_espera)}"
    fila_surface = pygame.font.SysFont(None, 36).render(fila_text, True,
black)
    screen.blit(fila_surface, (10, 150))

    # Dibujar los clientes en fila
    for i, cliente in enumerate(fila_espera):
        pygame.draw.circle(screen, pink, (50 + i * 30, 200), 10)

    # Dibujar los boxes y clientes siendo atendidos
```

```
    for i, cliente in enumerate(clientes_en_atencion):
        pygame.draw.rect(screen, blue, (100 + i * 60, 300, 50, 50))
        box_text = f"{i+1}"
        box_surface = pygame.font.SysFont(None, 36).render(box_text,
True, black)
        screen.blit(box_surface, (115 + i * 60, 305))
        if cliente is not None:
            pygame.draw.circle(screen, pink, (125 + i * 60, 290), 10)

    # Dibujar la hora actual
    hora_apertura_segundos = 8 * 3600  # 08:00:00 en segundos
    tiempo_actual = hora_apertura_segundos + second
    horas, minutos, segundos = tiempo_actual // 3600, (tiempo_actual %
3600) // 60, tiempo_actual % 60
    time_text = f"Hora: {horas:02d}:{minutos:02d}:{segundos:02d}"
    time_surface = pygame.font.SysFont(None, 36).render(time_text, True,
black)
    screen.blit(time_surface, (10, 50))

    # Dibujar la cantidad de clientes atendidos y no atendidos
    atendidos_text = f"Atendidos: {clientes_atendidos} | No Atendidos:
{clientes_no_atendidos}"
    atendidos_surface = pygame.font.SysFont(None,
36).render(atendidos_text, True, black)
    screen.blit(atendidos_surface, (10, 90))

    pygame.display.flip()

    # Guardar frame en el video
    frame = pygame.surfarray.array3d(pygame.display.get_surface())
    frame = frame.transpose([1, 0, 2])
    video.write(cv2.cvtColor(frame, cv2.COLOR_RGB2BGR))

def simular_local():
    try:
        num_boxes = int(input("Ingrese la cantidad de boxes (entre 1 y
10): "))
        if num_boxes < 1 or num_boxes > 10:
            raise ValueError("Número de boxes inválido. Debe estar entre
1 y 10.")
    except ValueError as e:
        print(e)
        return
```

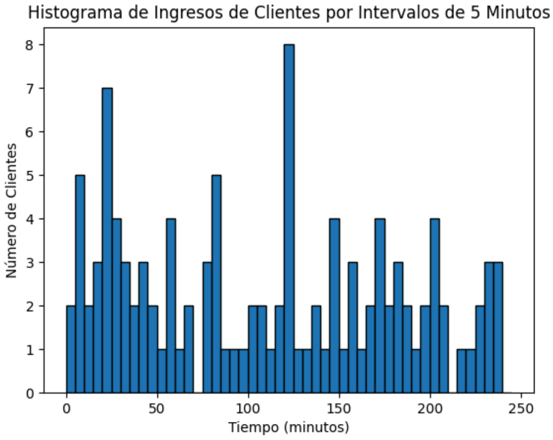
```
    try:
        velocidad_simulacion = float(input("Ingrese la velocidad de la
simulación (1.0 = tiempo real, 2.0 = el doble de rápido, etc.): "))
        if velocidad_simulacion <= 0:
            raise ValueError("La velocidad de la simulación debe ser
mayor que 0.")
    except ValueError as e:
        print(e)
        return

    # Inicialización
    clientes_ingresados = 0
    clientes_atendidos = 0
    clientes_no_atendidos = 0
    tiempo_minimo_atencion = float('inf')
    tiempo_maximo_atencion = 0
    tiempo_maximo_espera_actual = 0

    # Lista para rastrear los clientes actualmente siendo atendidos en
cada box
    clientes_en_atencion = [None] * num_boxes
    # Fila de espera de clientes (almacena el tiempo de ingreso de cada
cliente)
    fila_espera = deque()
    # Lista para almacenar los tiempos de ingreso de los clientes
    tiempos_ingreso = []

    segundos_totales = horas_apertura * 3600

    # Generar tiempos de ingreso de clientes
    while len(tiempos_ingreso) < esperanza_clientes:
        ingreso = int(random.normalvariate(media_ingreso,
desvio_ingreso) * 60)  # Convertir a segundos
        if 0 <= ingreso < segundos_totales:
            tiempos_ingreso.append(ingreso)
    tiempos_ingreso.sort()  # Ordenar los tiempos de ingreso

    segundo = 0
    running = True
    indice_tiempo_ingreso = 0
```

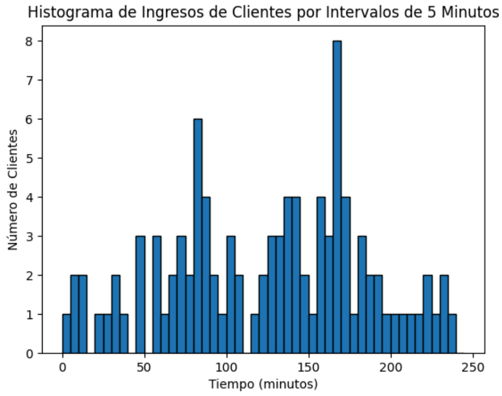
```
    while running and (segundo < segundos_totales or fila_espera or
any(clientes_en_atencion)):
        # Manejo de eventos de Pygame
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        # Simulación de ingreso de cliente dentro del horario de
apertura
        if indice_tiempo_ingreso < len(tiempos_ingreso) and
tiempos_ingreso[indice_tiempo_ingreso] == segundo:
            clientes_ingresados += 1
            fila_espera.append(segundo)
            indice_tiempo_ingreso += 1

        # Intentar asignar boxes libres a clientes en espera
        for i in range(num_boxes):
            if clientes_en_atencion[i] is None and fila_espera:
                tiempo_espera = segundo - fila_espera[0]
                if tiempo_espera <= tiempo_maximo_espera * 60:
                    # Simulación de atención
                    tiempo_atencion =
max(random.normalvariate(media_atencion, desvio_atencion), 0) * 60  #
Convertir a segundos
                    tiempo_minimo_atencion = min(tiempo_minimo_atencion,
tiempo_atencion)
                    tiempo_maximo_atencion = max(tiempo_maximo_atencion,
tiempo_atencion)

                    clientes_en_atencion[i] = tiempo_atencion
                    clientes_atendidos += 1
                    tiempo_maximo_espera_actual =
max(tiempo_maximo_espera_actual, tiempo_espera)
                    fila_espera.popleft()
                else:
                    fila_espera.popleft()
                    clientes_no_atendidos += 1

        # Actualizar el tiempo restante de atención para los clientes en
cada box
        for i in range(num_boxes):
            if clientes_en_atencion[i] is not None:
                clientes_en_atencion[i] -= 1
```

```
                if clientes_en_atencion[i] <= 0:
                    clientes_en_atencion[i] = None

        # Dibujar el estado actual
        draw_state(segundo, fila_espera, clientes_en_atencion,
clientes_atendidos, clientes_no_atendidos)
        clock.tick(30 * velocidad_simulacion)  # Ajustar la velocidad de
la simulación

        segundo += 1

    # Cálculo del costo total
    costo_total = num_boxes * costo_box + clientes_no_atendidos *
costo_perdida_cliente

    # Resultados
    print(f"Clientes ingresados: {clientes_ingresados}")
    print(f"Clientes atendidos: {clientes_atendidos}")
    print(f"Clientes no atendidos: {clientes_no_atendidos}")
    print(f"Tiempo mínimo de atención en box: {tiempo_minimo_atencion /
60:.2f} minutos")
    print(f"Tiempo máximo de atención en box: {tiempo_maximo_atencion /
60:.2f} minutos")
    print(f"Tiempo máximo de espera experimentado por un cliente:
{tiempo_maximo_espera_actual / 60:.2f} minutos")
    print(f"Tiempo máximo de espera permitido: {tiempo_maximo_espera}
minutos")
    print(f"Costo de la operación: ${costo_total:.2f}")

    # Mostrar los resultados finales en la animación
    resultados_texto = [
        f"Clientes ingresados: {clientes_ingresados}",
        f"Clientes atendidos: {clientes_atendidos}",
        f"Clientes no atendidos: {clientes_no_atendidos}",
        f"Tiempo mínimo de atención en box: {tiempo_minimo_atencion /
60:.2f} minutos",
        f"Tiempo máximo de atención en box: {tiempo_maximo_atencion /
60:.2f} minutos",
        f"Tiempo máximo de espera experimentado por un cliente:
{tiempo_maximo_espera_actual / 60:.2f} minutos",
        f"Tiempo máximo de espera permitido: {tiempo_maximo_espera}
minutos",
        f"Costo de la operación: ${costo_total:.2f}"
```

```
    ]

    screen.fill(white)
    for i, linea in enumerate(resultados_texto):
        resultado_surface = pygame.font.SysFont(None, 36).render(linea, True, black)
        screen.blit(resultado_surface, (10, 50 + i * 40))
    pygame.display.flip()
    pygame.time.wait(5000)

    # Guardar frame en el video
    frame = pygame.surfarray.array3d(pygame.display.get_surface())
    frame = frame.transpose([1, 0, 2])
    video.write(cv2.cvtColor(frame, cv2.COLOR_RGB2BGR))

    # Liberar el video
    video.release()
    pygame.quit()

    # Crear y mostrar el histograma de ingresos de clientes cada 5 minutos
    tiempos_ingreso_minutos = [t // 60 for t in tiempos_ingreso]  # Convertir tiempos a minutos
    plt.figure()
    plt.hist(tiempos_ingreso_minutos, bins=range(0, (segundos_totales // 60) + 6, 5), edgecolor='black')  # Bins de 5 minutos
    plt.xlabel('Tiempo (minutos)')
    plt.ylabel('Número de Clientes')
    plt.title('Histograma de Ingresos de Clientes por Intervalos de 5 Minutos')
    plt.show()

if __name__ == "__main__":
    simular_local()
```

Vamos a poner lado a lado los histogramas del TP7 y del TP8 para compararlo y poder ver las diferencias entre los distintos tipos de distribuciones que tienen los clientes al ingresar al local.





El histograma de la izquierda corresponde al TP7 donde podemos ver que el ingreso de personas está más distribuido en el tiempo, mientras que en el TP8 el ingreso de personas es menor a la apertura y al cierre del local y tiene un pico alrededor de las 10, 10:30.

TP9: MODELOS ESTOCÁSTICOS DE CRECIMIENTO POR AGREGACIÓN EN CONFINAMIENTO

Descripción:

Se trata de un conducto circular en el que se generan partículas en su centro, que pueden tener distinto tamaño. Estas partículas son cuadradas con longitudes de lado variables.

Esa partícula está dotada de un movimiento aleatorio hacia arriba, abajo, izquierda y derecha. En otros términos, en cada segundo se va a mover aleatoriamente.

Cuando esa partícula toca el borde del conducto o toca otra partícula, queda adherida a la zona donde tocó.

Cada vez que termina el movimiento, se genera una nueva partícula en el centro.

Variantes:

- Dimensiones del conducto en milímetros, forma (circular, cuadrada o rectangular), entre 1 mm y 200 mm.
- Dimensión de la partícula, entre 1 mm y 10 mm.