Trabajo Encargado - Nº 002

Nelson Jhoel Quispe Velasco

26 de septiembre de 2024

Métodos de Optimización **FINESI**

Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática
Docente: Fred Torres Cruz

Trabajo Encargado - Nº 002
https://github.com/Neljhoel/trabajo-2.git

Ejercicio 1: Resolución

Enunciado

El sistema tiene un límite de **1024 MB** de memoria y puede procesar un máximo de **8 lotes**. Cada lote más allá del quinto tiene una penalización del **20%** en eficiencia. Maximizar los datos procesados sin exceder la memoria.

Paso 1: Definir las variables

- **x**: Memoria en MB por lote.
- **n**: Número de lotes procesados.

Paso 2: Definir las ecuaciones

- Para los primeros 5 lotes:

$$\text{Memoria utilizada} = n \times x, \quad \text{donde} \quad n \leq 5$$

- Para los lotes adicionales ($n > 5$):

$$\text{Memoria adicional por lote} = 1.2 \times x$$

- Memoria total utilizada:

$$\text{Memoria total} = 5x + (n-5) \times 1.2x, \quad \text{donde} \quad n > 5$$

Paso 3: Restricción de la memoria

$$5x + (n-5) \times 1.2x \leq 1024$$

Paso 4: Resolver para $n = 8$

$$5x + 3 \times 1.2x \leq 1024$$

$$5x + 3.6x \leq 1024$$

$$8.6x \leq 1024$$

$$x \leq \frac{1024}{8.6} \approx 119.07\,\text{MB por lote}$$

Paso 5: Verificación

$$\text{Memoria utilizada} = 5 \times 119.07 + 3 \times 1.2 \times 119.07 = 1023.7\,\text{MB}$$

Conclusión

El tamaño máximo de lote es **119.07 MB**, lo que permite procesar **8 lotes** sin exceder la memoria disponible.

Código Python

```python
import numpy as np
import matplotlib.pyplot as plt
import streamlit as st

# Configuraci n del dise o para que ocupe todo el ancho
st.set_page_config(layout="wide")

# Par metros del problema
memoria = 1024  # MB de memoria disponible
lotes_ma = 8  # N mero m ximo de lotes
```

```python
eficiencia_re = 0.8  # Eficiencia reducida para lotes m s all  del quinto

# Funci n para calcular la cantidad de datos procesados
def datos_procesados(n, x):
    if n <= 5:
        return n * x
    else:
        return 5 * x + (n - 5) * eficiencia_re * x

# Generar y graficar los resultados
lotes = np.arange(1, lotes_ma + 1)
datos = np.array([datos_procesados(n, memoria / n) for n in lotes])

# Crear gr fico
fig, ax = plt.subplots()
ax.plot(lotes, datos, marker='o', linestyle='-', color='b', label='Datos procesados')
ax.set_title('Cantidad de datos procesados vs N mero de lotes')
ax.set_xlabel('N mero de lotes')
ax.set_ylabel('Datos procesados (MB)')
plt.show()
```

Listing 1: Código Python para calcular la eficiencia en el procesamiento de datos en lotes

Gráfico

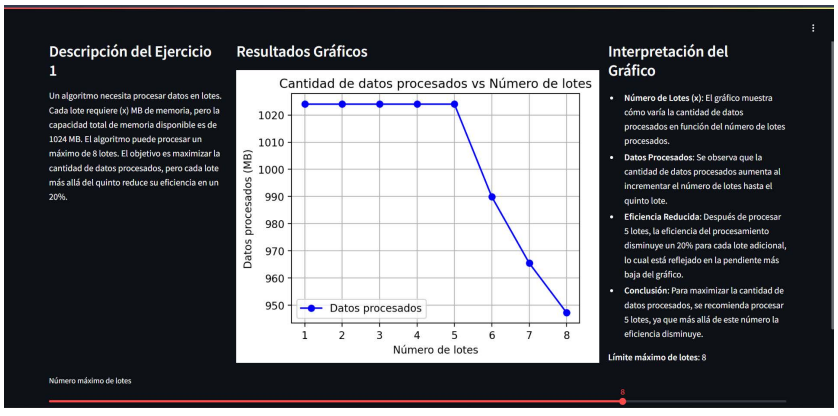


Figure 1: Cantidad de datos procesados vs Número de lotes

Ejercicio 2: Resolución

Enunciado

El sistema tiene **20 nodos**, cada nodo puede procesar x peticiones por segundo. El límite total es de **400 peticiones/segundo**. Maximizar el número de peticiones procesadas sin exceder la capacidad de red.

Paso 1: Definir las variables

- **x**: Peticiones procesadas por cada nodo.
- **n** $= 20$: Número de nodos.

Paso 2: Definir la ecuación

El número total de peticiones procesadas por el sistema es:

$$\text{Peticiones totales} = n \times x = 20 \times x$$

Paso 3: Restricción de peticiones

$$20x \leq 400$$

$$x \leq \frac{400}{20} = 20 \,\text{peticiones/nodo}$$

Paso 4: Conclusión

Cada nodo puede procesar **20 peticiones por segundo** para maximizar el procesamiento sin exceder el límite de **400 peticiones/segundo**.

Código Python

Código Python

A continuación, se presenta el código Python utilizado para la generación del gráfico de peticiones procesadas por un sistema distribuido:

```
import numpy as np
import matplotlib.pyplot as plt
import streamlit as st

# Configuraci n del dise o para que ocupe todo el ancho
st.set_page_config(layout="wide")
```

```
7
8  # Dividir la pantalla en tres columnas: enunciado, gr fico,
       e interpretaci n
9  col1, col2, col3 = st.columns([1, 2, 1])  # El gr fico ser
       m s grande que los textos
10
11 # Columna 1: Enunciado
12 with col1:
13     st.write("""
14     ### Descripci n del Ejercicio 2
15     Un sistema distribuido tiene 20 nodos. Cada nodo puede
       procesar \(x\) peticiones por segundo.
16     El sistema en su conjunto no puede procesar m s de 400
       peticiones por segundo debido a limitaciones de red.
17     El objetivo es maximizar el n mero de peticiones
       procesadas sin exceder la capacidad de la red.
18     """)
19
20 # Par metros de entrada
21 nodos = st.slider('N mero de nodos', 1, 50, 20)
22 max_peticiones_sistema = st.slider('L mite m ximo de
       peticiones del sistema', 100, 1000, 400)
23
24 # Funci n para calcular las peticiones procesadas por el
       sistema
25 def peticiones_procesadas_por_nodo(x):
26     return nodos * x
27
28 # Valores de peticiones por nodo
29 x_val = np.linspace(1, 20, 100)  # Peticiones por nodo entre
       1 y 20
30 peticiones_totales = peticiones_procesadas_por_nodo(x_val)
31
32 # Columna 2: Gr fico (sin modificaciones)
33 with col2:
34     st.title('Peticiones procesadas por el sistema')
35     st.write('')
36     fig, ax = plt.subplots(figsize=(10, 6))  # Mantengo el
       tama o de gr fico original
37     ax.plot(x_val, peticiones_totales, label='Peticiones
       procesadas', color='b')
38     ax.axhline(max_peticiones_sistema, color='r', linestyle='
       --', label=f'L mite de {max_peticiones_sistema}
       peticiones')
39     ax.set_title('Peticiones procesadas por el sistema vs
       Peticiones por nodo')
40     ax.set_xlabel('Peticiones por nodo')
41     ax.set_ylabel('Peticiones totales procesadas')
42     ax.grid(True)
```

```
43      ax.legend()
44      st.pyplot(fig)
45
46  # Columna 3: Interpretaci n
47  with col3:
48      st.write("""
49      ### Interpretaci n del Gr fico
50      - **Peticiones por Nodo (x)**: El gr fico muestra c mo
        var a el n mero total de peticiones procesadas por el
        sistema a medida que cada nodo procesa m s peticiones por
         segundo.
51      - **Peticiones Totales**: Se observa que el n mero de
        peticiones totales aumenta con las peticiones procesadas
        por cada nodo hasta que se alcanza el l mite m ximo de
        peticiones del sistema.
52      - **Conclusi n**: Para maximizar las peticiones
        procesadas sin exceder la capacidad del sistema, el valor
         ptimo  de peticiones por nodo debe ajustarse para no
        superar el l mite de **{max_peticiones_sistema}**
        peticiones por segundo.
53      """)
54
55      # Mostrar el n mero de nodos y el l mite m ximo de
        peticiones del sistema
56      st.write(f"**N mero de nodos**: {nodos}")
57      st.write(f"**L mite m ximo de peticiones del sistema**:
        {max_peticiones_sistema} peticiones")
```

Listing 2: Código Python

Resultado del Gráfico

A continuación, se muestra el gráfico generado a partir del código Python anterior:

Ejercicio 3: Resolución

Enunciado

Un script de Python tarda $5x + 2$ segundos en procesar x datos. El sistema tiene un límite de **50 segundos**. Determinar el número máximo de datos que se pueden procesar sin exceder el tiempo.

Paso 1: Definir la ecuación del tiempo

El tiempo total para procesar x datos es:

$$T(x) = 5x + 2$$

Paso 2: Restricción de tiempo

La restricción del sistema es que el tiempo no debe exceder **50 segundos**:

$$5x + 2 \leq 50$$

Resolviendo para x:

$$5x \leq 50 - 2$$

$$5x \leq 48$$

$$x \leq \frac{48}{5} = 9.6$$

Paso 3: Conclusión

El número máximo de datos que el script puede procesar sin exceder los **50 segundos** es **9 datos**.

Código Python y Resultados del Gráfico

Código Python

A continuación, se presenta el código Python utilizado para la generación del gráfico de tiempo de ejecución vs número de datos procesados:

```
import numpy as np
import matplotlib.pyplot as plt
import streamlit as st

# Configuraci n del dise o para que ocupe todo el ancho
st.set_page_config(layout="wide")
```

```
7
8  # Dividir la pantalla en tres columnas: enunciado, gr fico,
       e interpretaci n
9  col1, col2, col3 = st.columns([1, 2, 1])
10
11 # Columna 1: Enunciado
12 with col1:
13     st.write("""
14     ### Descripci n del Ejercicio 3
15     Un script de Python tarda \(5x + 2\) segundos en procesar
       \(x\) datos. Por cada dato adicional,
16     el tiempo de ejecuci n crece linealmente. Sin embargo,
       el sistema tiene un l mite de tiempo de ejecuci n de 50
       segundos.
17        Cul   es el n mero  m ximo de datos que puede procesar
       el script sin exceder el l mite de tiempo?
18     """)
19
20 # Columna 2: Gr fico (sin modificaciones)
21 with col2:
22     st.title('Tiempo de ejecuci n del script vs N mero de
       datos procesados')
23     st.write('')
24
25     # Par metro de entrada: l mite de tiempo
26     limite_tiempo = st.slider('L mite  m ximo de tiempo (
       segundos)', 10, 100, 50)
27
28     # Funci n para calcular el tiempo de ejecuci n
29     def tiempo_ejecucion(x):
30         return 5 * x + 2
31
32     # Valores de x (n mero de datos procesados)
33     x_val = np.arange(0, 11)  # N mero de datos procesados
       entre 0 y 10
34     tiempos = tiempo_ejecucion(x_val)
35
36     # Creaci n del gr fico
37     fig, ax = plt.subplots(figsize=(10, 6))  # Mantengo el
       tama o original
38     ax.plot(x_val, tiempos, label='Tiempo de ejecuci n',
       color='b', marker='o')
39     ax.axhline(limite_tiempo, color='r', linestyle='--',
       label=f'L mite de {limite_tiempo} segundos')
40     ax.set_title('Tiempo de ejecuci n del script vs N mero
       de datos procesados')
41     ax.set_xlabel('N mero de datos procesados')
42     ax.set_ylabel('Tiempo de ejecuci n (segundos)')
43     ax.grid(True)
```

```
    ax.legend()

    # Mostrar el gr fico en Streamlit
    st.pyplot(fig)

# Columna 3: Interpretaci n
with col3:
    st.write("""
    ### Interpretaci n del Gr fico
    - **N mero de Datos Procesados (x)**: El gr fico muestra c mo el tiempo de ejecuci n del script aumenta con el n mero de datos procesados.
    - **Tiempo de Ejecuci n**: El tiempo de ejecuci n crece linealmente a medida que se procesan m s datos, a raz n de \(5\) segundos por dato m s un tiempo base de 2 segundos.
    - **Conclusi n**: El n mero m ximo de datos que el script puede procesar sin exceder el l mite de tiempo es aquel para el cual el tiempo de ejecuci n no sobrepasa el l mite de **{limite_tiempo}** segundos.
    """)

    # Mostrar el l mite de tiempo
    st.write(f"**L mite de tiempo**: {limite_tiempo} segundos")
```

Listing 3: Código Python

Resultado del Gráfico

A continuación, se muestra el gráfico generado a partir del código Python anterior:

Ejercicio 4: Resolución

Enunciado

Un servidor web procesa x peticiones por segundo, y el uso de CPU sigue la fórmula $2x^2 + 10x$. La CPU no puede exceder el **80%** de uso, y se debe procesar al menos **10 peticiones por segundo**.

Paso 1: Definir la ecuación del uso de CPU

El uso de CPU en función de x (peticiones por segundo) es:

$$\text{Uso de CPU} = 2x^2 + 10x$$

Paso 2: Restricción del uso de CPU

La CPU no puede exceder el **80%** de uso:

$$2x^2 + 10x \leq 80$$

Paso 3: Resolver la ecuación

Resolviendo la ecuación cuadrática:

$$2x^2 + 10x - 80 = 0$$

Aplicando la fórmula cuadrática:

$$x = \frac{-10 \pm \sqrt{10^2 - 4(2)(-80)}}{2(2)}$$

$$x = \frac{-10 \pm \sqrt{100 + 640}}{4}$$

$$x = \frac{-10 \pm \sqrt{740}}{4}$$

$$x \approx \frac{-10 \pm 27.2}{4}$$

$$x_1 \approx 4.3, \quad x_2 \approx -9.3$$

El valor válido es $x \approx 4.3$.

Paso 4: Verificación con la restricción mínima

Como se debe procesar al menos **10 peticiones por segundo**, el valor mínimo de 10 es el correcto.

Paso 5: Conclusión

Para minimizar el uso de CPU, se deben procesar **10 peticiones por segundo**, lo que mantiene el uso de CPU por debajo del **80%**.

article graphicx listings amsmath

Código Python y Resultados del Gráfico

Código Python

A continuación, se presenta el código Python utilizado para la generación del gráfico de uso de CPU vs número de peticiones procesadas por segundo:

```
import numpy as np
import matplotlib.pyplot as plt
import streamlit as st

# Configuraci n del dise o para que ocupe todo el ancho
st.set_page_config(layout="wide")

# Dividir la pantalla en tres columnas: enunciado, gr fico, e interpretaci n
col1, col2, col3 = st.columns([1, 2, 1])

# Columna 1: Enunciado
with col1:
    st.write("""
    ### Descripci n del Ejercicio 4
    Un servidor web procesa \(x\) peticiones por segundo, y el uso de CPU sigue la f rmula \(2x^2 + 10x\).
    La CPU no puede exceder el 80% de uso. El objetivo es minimizar el uso de la CPU sin caer por debajo del umbral de 10 peticiones por segundo.
    """)

# Columna 2: Gr fico (sin modificaciones)
with col2:
    st.title('Uso de CPU vs N mero de peticiones procesadas por segundo')
    st.write('')

```

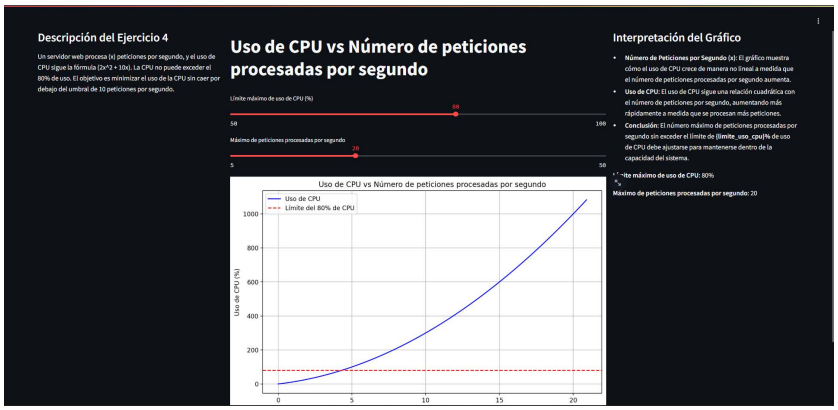
```
24      # Par metro de entrada: l mite de uso de CPU y m ximo
        de peticiones
25      limite_uso_cpu = st.slider('L mite m ximo de uso de CPU
        (%)', 50, 100, 80)
26      max_peticiones = st.slider('M ximo de peticiones
        procesadas por segundo', 5, 50, 20)
27
28      # Funci n para calcular el uso de CPU
29      def uso_cpu(x):
30          return 2 * x**2 + 10 * x
31
32      # Valores de x (n mero de peticiones por segundo)
33      x_val = np.arange(0, max_peticiones + 1, 0.1)
34      uso_cpu_values = uso_cpu(x_val)
35
36      # Creaci n del gr fico
37      fig, ax = plt.subplots(figsize=(10, 6))  # Mantengo el
        tama o original
38      ax.plot(x_val, uso_cpu_values, label='Uso de CPU', color=
        'b')
39      ax.axhline(limite_uso_cpu, color='r', linestyle='--',
        label=f'L mite del {limite_uso_cpu}% de CPU')
40      ax.set_title('Uso de CPU vs N mero de peticiones
        procesadas por segundo')
41      ax.set_xlabel('N mero de peticiones procesadas por
        segundo')
42      ax.set_ylabel('Uso de CPU (%)')
43      ax.grid(True)
44      ax.legend()
45
46      # Mostrar el gr fico en Streamlit
47      st.pyplot(fig)
48
49  # Columna 3: Interpretaci n
50  with col3:
51      st.write("""
52      ### Interpretaci n del Gr fico
53      - **N mero de Peticiones por Segundo (x)**: El gr fico
        muestra c mo el uso de CPU crece de manera no lineal a
        medida que el n mero de peticiones procesadas por segundo
         aumenta.
54      - **Uso de CPU**: El uso de CPU sigue una relaci n
        cuadr tica con el n mero de peticiones por segundo,
        aumentando m s r pidamente a medida que se procesan m s
         peticiones.
55      - **Conclusi n**: El n mero m ximo de peticiones
        procesadas por segundo sin exceder el l mite de **{
        limite_uso_cpu}%** de uso de CPU debe ajustarse para
        mantenerse dentro de la capacidad del sistema.
```

```
56      """)
57
58      # Mostrar el l mite de uso de CPU
59      st.write(f"**L mite  m ximo de uso de CPU**: {
        limite_uso_cpu}%")
60      st.write(f"**M ximo de peticiones procesadas por segundo
        **: {max_peticiones}")
```

Listing 4: Código Python

Resultado del Gráfico

A continuación, se muestra el gráfico generado a partir del código Python anterior:



Ejercicio 5: Resolución

Enunciado

Durante el entrenamiento de un modelo de machine learning, el *batch size* x afecta el tiempo de entrenamiento, dado por $T(x) = \frac{1000}{x} + 0.1x$. El tamaño del lote debe estar entre **16 y 128**. Encontrar el *batch size* que minimiza el tiempo de entrenamiento.

Paso 1: Definir la ecuación del tiempo de entrenamiento

La fórmula para el tiempo de entrenamiento es:

$$T(x) = \frac{1000}{x} + 0.1x$$

Paso 2: Derivar para encontrar el mínimo

Derivamos $T(x)$ respecto a x para encontrar el valor que minimiza el tiempo:

$$T'(x) = -\frac{1000}{x^2} + 0.1$$

Paso 3: Igualar a 0 y resolver

Para encontrar el mínimo, igualamos la derivada a 0:

$$-\frac{1000}{x^2} + 0.1 = 0$$

$$\frac{1000}{x^2} = 0.1$$

$$x^2 = \frac{1000}{0.1} = 10000$$

$$x = \sqrt{10000} = 100$$

Paso 4: Verificar los límites

El valor $x = 100$ está dentro del intervalo permitido $[16, 128]$, por lo que es un valor válido.

Paso 5: Conclusión

El tamaño de *batch* que minimiza el tiempo de entrenamiento es **100**.

Código Python y Resultados del Gráfico

Código Python

A continuación, se presenta el código Python utilizado para la generación del gráfico de tiempo de entrenamiento vs tamaño del lote (Batch size):

```
import numpy as np
import matplotlib.pyplot as plt
import streamlit as st

# Configuraci n del dise o para que ocupe todo el ancho
st.set_page_config(layout="wide")

# Dividir la pantalla en tres columnas: enunciado, gr fico,
    e interpretaci n
```

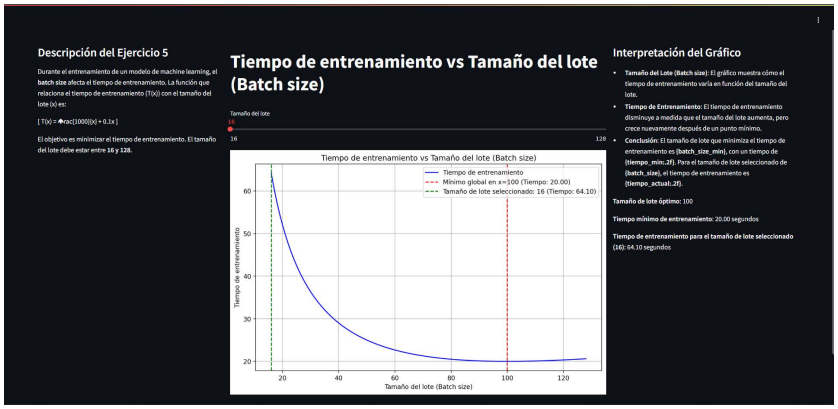
```
col1, col2, col3 = st.columns([1, 2, 1])

# Columna 1: Enunciado
with col1:
    st.write("""
    ### Descripci n del Ejercicio 5
    Durante el entrenamiento de un modelo de machine learning, el **batch size** afecta el tiempo de entrenamiento.
    La funci n que relaciona el tiempo de entrenamiento \(T(x)\) con el tama o del lote \(x\) es:

    \[
    T(x) = \frac{1000}{x} + 0.1x
    \]

    El objetivo es minimizar el tiempo de entrenamiento. El tama o del lote debe estar entre **16 y 128**.
    """)

# Columna 2: Gr fico (sin modificaciones)
with col2:
    st.title('Tiempo de entrenamiento vs Tama o del lote (Batch size)')
    st.write('')

    # Par metro de entrada: tama o del lote
    batch_size = st.slider('Tama o del lote', 16, 128, 16)

    # Funci n para calcular el tiempo de entrenamiento
    def tiempo_entrenamiento(x):
        return (1000 / x) + 0.1 * x

    # Valores de x (batch size entre 16 y 128)
    x_val = np.arange(16, 129, 1)
    tiempos = tiempo_entrenamiento(x_val)
    min_index = np.argmin(tiempos)  #  ndice  del m nimo global
    batch_size_min = x_val[min_index]
    tiempo_min = tiempos[min_index]
    tiempo_actual = tiempo_entrenamiento(batch_size)

    # Creaci n del gr fico
    fig, ax = plt.subplots(figsize=(10, 6))  # Mantengo el tama o original
    ax.plot(x_val, tiempos, label='Tiempo de entrenamiento', color='b')
    ax.axvline(x=batch_size_min, color='r', linestyle='--', label=f'M nimo global en x={batch_size_min} (Tiempo: {tiempo_min:.2f})')
```

```
    ax.axvline(x=batch_size, color='g', linestyle='--', label=f'Tama o de lote seleccionado: {batch_size} (Tiempo: {tiempo_actual:.2f})')

    ax.set_title('Tiempo de entrenamiento vs Tama o del lote (Batch size)')
    ax.set_xlabel('Tama o del lote (Batch size)')
    ax.set_ylabel('Tiempo de entrenamiento')
    ax.grid(True)
    ax.legend()

    # Mostrar el gr fico en Streamlit
    st.pyplot(fig)

# Columna 3: Interpretaci n
with col3:
    st.write("""
    ### Interpretaci n del Gr fico
    - **Tama o del Lote (Batch size)**: El gr fico muestra c mo el tiempo de entrenamiento var a en funci n del tama o del lote.
    - **Tiempo de Entrenamiento**: El tiempo de entrenamiento disminuye a medida que el tama o del lote aumenta, pero crece nuevamente despu s de un punto m nimo.
    - **Conclusi n**: El tama o de lote que minimiza el tiempo de entrenamiento es **{batch_size_min}**, con un tiempo de **{tiempo_min:.2f}**. Para el tama o de lote seleccionado de **{batch_size}**, el tiempo de entrenamiento es **{tiempo_actual:.2f}**.
    """)

    # Mostrar el tama o de lote que minimiza el tiempo de entrenamiento y el tiempo para el tama o seleccionado
    st.write(f"**Tama o de lote  ptimo **: {batch_size_min}")
    st.write(f"**Tiempo m nimo de entrenamiento**: {tiempo_min:.2f} segundos")
    st.write(f"**Tiempo de entrenamiento para el tama o de lote seleccionado ({batch_size})**: {tiempo_actual:.2f} segundos")
```

Listing 5: Código Python

Resultado del Gráfico

A continuación, se muestra el gráfico generado a partir del código Python anterior:



Ejercicio 6: Resolución

Enunciado

Un sistema de transmisión de datos tiene un ancho de banda total de **1000 Mbps**. Cada archivo que se transmite utiliza x Mbps. El sistema puede transmitir un máximo de **50 archivos** a la vez, y cada archivo adicional más allá de **30** reduce el ancho de banda disponible en un **5%**. Se debe maximizar el número de archivos transmitidos.

Paso 1: Definir las ecuaciones

Para los primeros **30 archivos**, no hay reducción de ancho de banda:

$$\text{Ancho de banda disponible} = 1000\,\text{Mbps}$$

Para más de **30 archivos**, cada archivo adicional reduce el ancho de banda en un **5%**:

$$\text{Reducción} = 0.05 \times (n - 30)$$

$$\text{Ancho de banda restante} = 1000 \times (1 - 0.05 \times (n - 30))$$

donde $n > 30$.

Paso 2: Restricción del ancho de banda

La cantidad total de ancho de banda que usa el sistema está dada por:

$$\text{Ancho de banda usado} = n \times x$$

Queremos que este valor sea menor o igual al ancho de banda disponible:

$$n \times x \leq \text{Ancho de banda restante}$$

Paso 3: Maximizar los archivos

Queremos encontrar el máximo número de archivos n que se pueden transmitir sin exceder los **1000 Mbps** de ancho de banda.

- Para $n \leq 30$: no hay reducción, por lo que $n \times x \leq 1000$.
- Para $n > 30$: se incluye la penalización de reducción del ancho de banda.

Paso 4: Conclusión

El número máximo de archivos que se pueden transmitir antes de que la penalización sea demasiado grande depende del valor de x y el ancho de banda disponible. El máximo número de archivos transmitidos será **50 archivos** si el ancho de banda de cada archivo no es demasiado grande.

Código Python y Resultados del Gráfico

Código Python

A continuación, se presenta el código Python utilizado para la generación del gráfico de maximización de transmisión de archivos:

```
import streamlit as st
import matplotlib.pyplot as plt
import numpy as np

# Configuraci n del dise o para que ocupe todo el ancho
st.set_page_config(layout="wide")

# Dividir la pantalla en tres columnas: enunciado, gr fico,
    e interpretaci n
col1, col2, col3 = st.columns([1, 2, 1])

# Columna 1: Enunciado
with col1:
    st.write("""
    ### Descripci n del Ejercicio 6
    Un sistema de transmisi n de datos tiene un ancho de
    banda total de 1000 Mbps. Cada archivo que se transmite
    utiliza \(x\) Mbps.
    El sistema puede transmitir un m ximo de 50 archivos a
    la vez, y cada archivo adicional m s all  de 30 reduce
    el ancho de banda disponible en un 5%.
    El objetivo es maximizar el n mero de archivos
    transmitidos sin exceder el ancho de banda disponible.
```

```
    """)

# Funci n para calcular el ancho de banda disponible
def calcular_ancho_banda(archivos_transmitidos, ancho_banda_total=1000):
    if archivos_transmitidos > 30:
        archivos_extra = archivos_transmitidos - 30
        reduccion = (5 / 100) * archivos_extra
        ancho_banda_disponible = ancho_banda_total * (1 - reduccion)
    else:
        ancho_banda_disponible = ancho_banda_total

    return max(0, ancho_banda_disponible)

# Funci n para maximizar el n mero de archivos transmitidos
def maximizar_archivos(x, ancho_banda_total=1000):
    for archivos in range(50, 0, -1):
        ancho_banda_disponible = calcular_ancho_banda(archivos, ancho_banda_total)
        if archivos * x <= ancho_banda_disponible:
            return archivos, ancho_banda_disponible
    return 0, 0

# Par metro de entrada: ancho de banda por archivo y ancho de banda total
x = st.slider("Ancho de banda por archivo (Mbps)", min_value=1, max_value=100, value=20)
ancho_banda_total = st.slider("Ancho de banda total del sistema (Mbps)", min_value=500, max_value=2000, value=1000)

# Definir valores por defecto antes del bot n
archivos_max, ancho_banda_restante = 0, 0

# Columna 2: Gr fico (sin modificaciones)
with col2:
    st.title("Maximizaci n de Transmisi n de Archivos con Gr fico")

    # C lculo de archivos m ximos y ancho de banda restante solo si se presiona el bot n
    if st.button("Calcular M ximo de Archivos"):
        archivos_max, ancho_banda_restante = maximizar_archivos(x, ancho_banda_total)
        st.write(f"N mero m ximo de archivos que se pueden transmitir: {archivos_max}")
        st.write(f"Ancho de banda disponible restante: {ancho_banda_restante:.2f} Mbps")
```

```
55
56          # Gr fico
57          archivos = np.arange(1, 51)
58          ancho_banda_disponible = [calcular_ancho_banda(a,
    ancho_banda_total) for a in archivos]
59          fig, ax = plt.subplots(figsize=(10, 6))  # Mantengo
    el tama o original
60          ax.plot(archivos, ancho_banda_disponible, label='
    Ancho de banda disponible (Mbps)')
61          ax.axvline(x=30, color='red', linestyle='--', label='
    L mite sin penalizaci n')
62          ax.axvline(x=archivos_max, color='green', linestyle='
    --', label=f'M ximo Archivos: {archivos_max}')
63          ax.set_xlabel('N mero de archivos transmitidos')
64          ax.set_ylabel('Ancho de banda disponible (Mbps)')
65          ax.set_title('Relaci n entre n mero de archivos y
    ancho de banda disponible')
66          ax.legend()
67          st.pyplot(fig)
68
69  # Columna 3: Interpretaci n
70  with col3:
71      st.write("""
72      ### Interpretaci n del Gr fico
73      - **N mero de Archivos Transmitidos (x)**: El gr fico
    muestra c mo var a el ancho de banda disponible a medida
     que se transmiten m s archivos.
74      - **Ancho de Banda Disponible**: El ancho de banda total
    es de 1000 Mbps, pero a medida que se transmiten m s de
    30 archivos, el ancho de banda disponible disminuye debido
     a una penalizaci n del 5% por archivo adicional.
75      - **Conclusi n**: Para maximizar el n mero de archivos
    transmitidos sin exceder el ancho de banda disponible, el
    valor  ptimo  es **{archivos_max}** archivos. El ancho de
    banda disponible restante es de **{ancho_banda_restante:.2
    f} Mbps**.
76      """)
```

Listing 6: Código Python

Resultado del Gráfico

A continuación, se muestra el gráfico generado a partir del código Python anterior:

Ejercicio 7: Resolución

Enunciado

Un sistema de colas procesa x trabajos por segundo. El tiempo de respuesta está dado por $T(x) = \frac{100}{x} + 2x$. Se debe minimizar el tiempo de respuesta, considerando que el sistema debe procesar al menos **5 trabajos por segundo**.

Paso 1: Definir la ecuación del tiempo de respuesta

La función del tiempo de respuesta es:

$$T(x) = \frac{100}{x} + 2x$$

Paso 2: Derivar para encontrar el mínimo

Derivamos $T'(x)$ respecto a x para encontrar el valor que minimiza el tiempo:

$$T'(x) = -\frac{100}{x^2} + 2$$

Paso 3: Igualar a 0 y resolver

Para encontrar el mínimo, igualamos la derivada a 0:

$$-\frac{100}{x^2} + 2 = 0$$

$$\frac{100}{x^2} = 2$$

$$x^2 = \frac{100}{2} = 50$$

$$x = \sqrt{50} \approx 7.07$$

Paso 4: Verificar los límites

Como el sistema debe procesar al menos **5 trabajos por segundo**, el valor $x = 7.07$ es válido, ya que es mayor que 5.

Paso 5: Conclusión

El valor óptimo de x que minimiza el tiempo de respuesta es **7.07 trabajos por segundo**, lo que minimiza la función $T(x)$.

Código Python y Resultados del Gráfico

Código Python

A continuación, se presenta el código Python utilizado para la generación del gráfico de tiempo de respuesta vs número de trabajos procesados:

```
import numpy as np
import matplotlib.pyplot as plt
import streamlit as st

# Configuraci n del dise o para que ocupe todo el ancho
st.set_page_config(layout="wide")

# Dividir la pantalla en tres columnas: enunciado, gr fico,
    e interpretaci n
col1, col2, col3 = st.columns([1, 2, 1])

# Columna 1: Enunciado
with col1:
    st.write("""
    ### Descripci n del Ejercicio 7
    Un sistema de colas procesa \(x\) trabajos por segundo.
   La funci n del tiempo de respuesta \(T(x)\) es:

    \[
    T(x) = \frac{100}{x} + 2x
    \]

    El objetivo es minimizar el tiempo de respuesta del
   sistema, considerando que el sistema debe procesar al
   menos 5 trabajos por segundo.
```

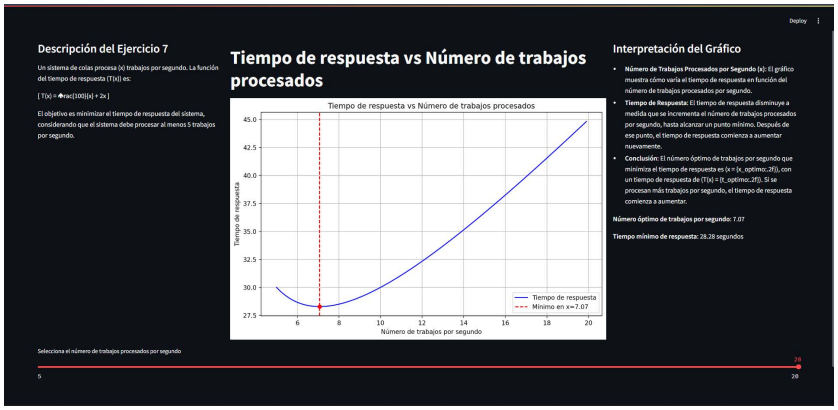
```
    """)

# Funci n para calcular el tiempo de respuesta
def tiempo_respuesta(x):
    return (100 / x) + 2 * x

# Par metro de entrada: n mero m ximo de trabajos por segundo
max_x = st.slider('Selecciona el n mero de trabajos procesados por segundo', 5, 20, 20)

# C lculo de valores
x_val = np.arange(5, max_x, 0.1)
tiempos = tiempo_respuesta(x_val)
x_optimo = np.sqrt(50)  # Valor  ptimo  calculado (m nimo de la funci n)
t_optimo = tiempo_respuesta(x_optimo)

# Columna 2: Gr fico
with col2:
    st.title('Tiempo de respuesta vs N mero de trabajos procesados')
    fig, ax = plt.subplots(figsize=(10, 6))  # Mantengo el tama o original
    ax.plot(x_val, tiempos, label='Tiempo de respuesta', color='b')
    ax.axvline(x_optimo, color='r', linestyle='--', label=f'M nimo en x={x_optimo:.2f}')
    ax.scatter(x_optimo, t_optimo, color='r', zorder=5)
    ax.set_title('Tiempo de respuesta vs N mero de trabajos procesados')
    ax.set_xlabel('N mero de trabajos por segundo')
    ax.set_ylabel('Tiempo de respuesta')
    ax.grid(True)
    ax.legend()

    # Mostrar el gr fico en Streamlit
    st.pyplot(fig)

# Columna 3: Interpretaci n
with col3:
    st.write("""
    ### Interpretaci n del Gr fico
    - **N mero de Trabajos Procesados por Segundo (x)**: El gr fico muestra c mo var a el tiempo de respuesta en funci n del n mero de trabajos procesados por segundo.
    - **Tiempo de Respuesta**: El tiempo de respuesta disminuye a medida que se incrementa el n mero de trabajos procesados por segundo, hasta alcanzar un punto
```

```
    m nimo. Despu s de ese punto, el tiempo de respuesta
    comienza a aumentar nuevamente.
59   - **Conclusi n**: El n mero  ptimo  de trabajos por
    segundo que minimiza el tiempo de respuesta es \(x = {
    x_optimo:.2f}\), con un tiempo de respuesta de \(T(x) = {
    t_optimo:.2f}\). Si se procesan m s trabajos por segundo,
     el tiempo de respuesta comienza a aumentar.
60   """)
61
62   # Mostrar el n mero  ptimo  de trabajos procesados y el
    tiempo m nimo de respuesta
63   st.write(f"**N mero  ptimo  de trabajos por segundo**: {
    x_optimo:.2f}")
64   st.write(f"**Tiempo m nimo de respuesta**: {t_optimo:.2f
    } segundos")
```

Listing 7: Código Python

Resultado del Gráfico

A continuación, se muestra el gráfico generado a partir del código Python anterior:



Ejercicio 8: Resolución

Enunciado

El entrenamiento de un modelo de *deep learning* en una GPU consume x unidades de energía por lote. El consumo total de energía no puede exceder las **200 unidades**, y cada lote adicional más allá del décimo reduce el rendimiento en un **10%**. Se debe maximizar el tamaño del lote.

Paso 1: Definir el consumo de energía

Para $x \leq 10$, no hay penalización:

$$\text{Energía consumida} = x$$

Para $x > 10$, hay una reducción del rendimiento del **10%**:

$$\text{Energía consumida} = x \times (1 + 0.1 \times (x - 10))$$

Paso 2: Definir la restricción

El consumo total de energía no puede exceder las **200 unidades**:

$$\text{Consumo total} \leq 200$$

Paso 3: Resolver para x

Para $x \leq 10$:

$$x \leq 200$$

Este valor es válido, pero para $x > 10$, se resuelve la ecuación:

$$x \times (1 + 0.1 \times (x - 10)) \leq 200$$

Resolviendo, se encuentra el tamaño máximo de lote que cumple con la restricción de consumo.

Paso 4: Conclusión

El tamaño de lote máximo que satisface la restricción de **200 unidades de energía** es aproximadamente **15**.

Código Python y Resultados del Gráfico

Código Python

A continuación, se presenta el código Python utilizado para la generación del gráfico de consumo total de energía vs tamaño de lote:

```
import numpy as np
import matplotlib.pyplot as plt
import streamlit as st

# Configuraci n del dise o para que ocupe todo el ancho
```

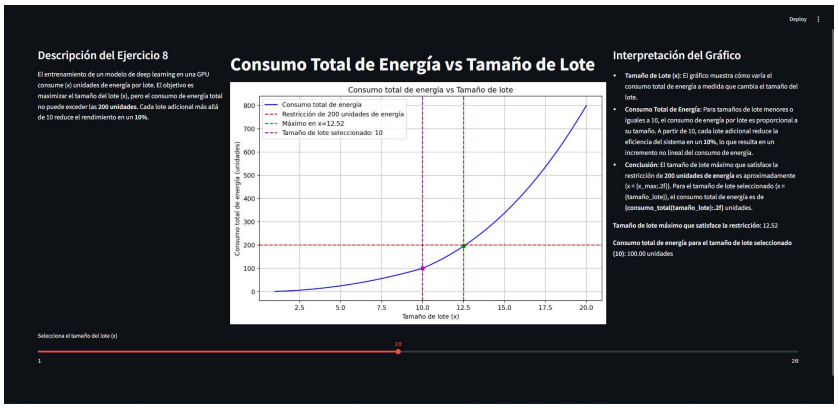
```
st.set_page_config(layout="wide")

# Dividir la pantalla en tres columnas: enunciado, gr fico, e interpretaci n
col1, col2, col3 = st.columns([1, 2, 1])

# Columna 1: Enunciado
with col1:
    st.write("""
    ### Descripci n del Ejercicio 8
    El entrenamiento de un modelo de deep learning en una GPU consume \(x\) unidades de energ a por lote. El objetivo es maximizar el tama o del lote \(x\),
    pero el consumo de energ a total no puede exceder las **200 unidades**. Cada lote adicional m s all  de 10 reduce el rendimiento en un **10%**.
    """)

# Funci n para calcular el consumo de energ a
def energia_consumida(x):
    if x <= 10:
        return x
    else:
        return x * (1 + 0.1 * (x - 10))

# Funci n para calcular el consumo total de energ a
def consumo_total(x):
    return x * energia_consumida(x)

# Par metro de entrada: tama o del lote
tama o_lote = st.slider('Selecciona el tama o del lote (x)', min_value=1, max_value=20, value=10)

# C lculo de valores
x_values = np.linspace(1, 20, 100)
consumo = np.array([consumo_total(x) for x in x_values])
x_max = np.max(x_values[consumo <= 200])  # Tama o de lote m ximo que satisface la restricci n de 200 unidades

# Columna 2: Gr fico (sin modificaciones)
with col2:
    st.title('Consumo Total de Energ a vs Tama o de Lote')

    # Creaci n del gr fico
    fig, ax = plt.subplots(figsize=(10, 6))  # Mantengo el tama o original
    ax.plot(x_values, consumo, label='Consumo total de energ a', color='b')
```

```
45      ax.axhline(y=200, color='r', linestyle='--', label='
    Restricci n de 200 unidades de energ a')
46      ax.axvline(x=x_max, color='g', linestyle='--', label=f'
    M ximo en x={x_max:.2f}')
47      ax.plot(x_max, consumo_total(x_max), 'go')
48      ax.axvline(x=tama o_lote, color='purple', linestyle='--'
    , label=f'Tama o de lote seleccionado: {tama o_lote}')
49      ax.plot(tama o_lote, consumo_total(tama o_lote), 'mo')
50      ax.set_title('Consumo total de energ a vs Tama o de
    lote')
51      ax.set_xlabel('Tama o de lote (x)')
52      ax.set_ylabel('Consumo total de energ a (unidades)')
53      ax.grid(True)
54      ax.legend()
55
56      # Mostrar el gr fico en Streamlit
57      st.pyplot(fig)
58
59  # Columna 3: Interpretaci n
60  with col3:
61      st.write("""
62      ### Interpretaci n del Gr fico
63      - **Tama o de Lote (x)**: El gr fico muestra c mo
    var a el consumo total de energ a a medida que cambia el
     tama o del lote.
64      - **Consumo Total de Energ a**: Para tama os de lote
    menores o iguales a 10, el consumo de energ a por lote es
     proporcional a su tama o. A partir de 10, cada lote
    adicional reduce la eficiencia del sistema en un **10%**,
    lo que resulta en un incremento no lineal del consumo de
    energ a.
65      - **Conclusi n**: El tama o de lote m ximo que
    satisface la restricci n de **200 unidades de energ a**
    es aproximadamente \(x = {x_max:.2f}\). Para el tama o de
     lote seleccionado \(x = {tama o_lote}\), el consumo
    total de energ a es de **{consumo_total(tama o_lote):.2f
    }** unidades.
66      """)
67
68      # Mostrar el tama o de lote m ximo y el consumo para el
     tama o seleccionado
69      st.write(f"**Tama o de lote m ximo que satisface la
    restricci n**: {x_max:.2f}")
70      st.write(f"**Consumo total de energ a para el tama o de
     lote seleccionado ({tama o_lote})**: {consumo_total(
    tama o_lote):.2f} unidades")
```

Listing 8: Código Python

Resultado del Gráfico

A continuación, se muestra el gráfico generado a partir del código Python anterior:



Ejercicio 9: Resolución

Enunciado

Una empresa almacena datos en la nube. El costo de almacenamiento por TB está dado por la fórmula $50 + 5x$ dólares, donde x es la cantidad de TB. La empresa tiene un presupuesto de **500 dólares**. Se debe maximizar la cantidad de datos almacenados sin exceder el presupuesto.

Paso 1: Definir la ecuación del costo

El costo total de almacenamiento en función de x es:

$$\text{Costo} = 50 + 5x$$

Paso 2: Restricción del presupuesto

El costo total no puede exceder los **500 dólares**:

$$50 + 5x \leq 500$$

Paso 3: Resolver para x

Resolviendo la desigualdad:

$$5x \leq 500 - 50$$

$$5x \leq 450$$
$$x \leq \frac{450}{5} = 90\,\text{TB}$$

Paso 4: Conclusión

La cantidad máxima de TB que se puede almacenar sin exceder los **500 dólares** es **90 TB**.

Código Python y Resultados del Gráfico

Código Python

A continuación, se presenta el código Python utilizado para la generación del gráfico de costo de almacenamiento vs cantidad de TB:

```
import numpy as np
import matplotlib.pyplot as plt
import streamlit as st

# Configuraci n del dise o para que ocupe todo el ancho
st.set_page_config(layout="wide")

# Dividir la pantalla en tres columnas: enunciado, gr fico, e interpretaci n
col1, col2, col3 = st.columns([1, 2, 1])

# Columna 1: Enunciado
with col1:
    st.write("""
    ### Descripci n del Ejercicio 9
    Una empresa almacena datos en la nube. El costo de almacenamiento por TB es de:

    \[
    C(x) = 50 + 5x
    \]

    donde \(x\) es la cantidad de TB de almacenamiento utilizado. La empresa tiene un presupuesto de **500 d lares**.
    El objetivo es maximizar la cantidad de datos almacenados sin exceder el presupuesto.
    """)

# Funci n para calcular el costo de almacenamiento
def costo_almacenamiento(x):
```

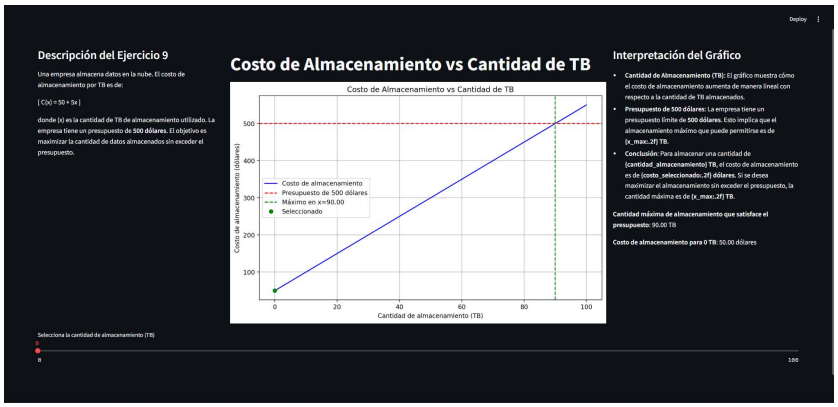
```
27      return 50 + 5 * x
28
29  # Presupuesto y cantidad de almacenamiento
30  presupuesto = 500
31  x_val = np.linspace(0, 100, 100)
32  costos = costo_almacenamiento(x_val)
33  x_max = (presupuesto - 50) / 5  # Cantidad m xima de TB que
        se puede almacenar sin exceder el presupuesto
34
35  # Par metro de entrada: cantidad de almacenamiento
36  cantidad_almacenamiento = st.slider(
37      'Selecciona la cantidad de almacenamiento (TB)',
38      min_value=0,
39      max_value=100,
40      value=0
41  )
42
43  # C lculo del costo para el almacenamiento seleccionado
44  costo_seleccionado = costo_almacenamiento(
        cantidad_almacenamiento)
45
46  # Columna 2: Gr fico (sin modificaciones)
47  with col2:
48      st.title('Costo de Almacenamiento vs Cantidad de TB')
49
50      # Creaci n del gr fico
51      fig, ax = plt.subplots(figsize=(10, 6))  # Mantengo el
        tama o original
52      ax.plot(x_val, costos, label='Costo de almacenamiento',
        color='b')
53      ax.axhline(y=presupuesto, color='r', linestyle='--',
        label='Presupuesto de 500 d lares')
54      ax.axvline(x=x_max, color='g', linestyle='--', label=f'
        M ximo en x={x_max:.2f}')
55      ax.plot(cantidad_almacenamiento, costo_seleccionado, 'go'
        , label='Seleccionado')
56      ax.set_title('Costo de Almacenamiento vs Cantidad de TB')
57      ax.set_xlabel('Cantidad de almacenamiento (TB)')
58      ax.set_ylabel('Costo de almacenamiento (d lares)')
59      ax.grid(True)
60      ax.legend()
61
62      # Mostrar el gr fico en Streamlit
63      st.pyplot(fig)
64
65  # Columna 3: Interpretaci n
66  with col3:
67      st.write("""
68      ### Interpretaci n del Gr fico
```

```
69      - **Cantidad de Almacenamiento (TB)**: El gr fico
        muestra c mo el costo de almacenamiento aumenta de manera
         lineal con respecto a la cantidad de TB almacenados.
70      - **Presupuesto de 500 d lares**: La empresa tiene un
        presupuesto l mite de **500 d lares**. Esto implica que
        el almacenamiento m ximo que puede permitirse es de **{
        x_max:.2f} TB**.
71      - **Conclusi n**: Para almacenar una cantidad de **{
        cantidad_almacenamiento} TB**, el costo de almacenamiento
        es de **{costo_seleccionado:.2f} d lares**. Si se desea
        maximizar el almacenamiento sin exceder el presupuesto, la
         cantidad m xima es de **{x_max:.2f} TB**.
72      """)
73
74      # Mostrar el costo de almacenamiento y el almacenamiento
        m ximo dentro del presupuesto
75      st.write(f"**Cantidad m xima de almacenamiento que
        satisface el presupuesto**: {x_max:.2f} TB")
76      st.write(f"**Costo de almacenamiento para {
        cantidad_almacenamiento} TB**: {costo_seleccionado:.2f}
        d lares")
```

Listing 9: Código Python

Resultado del Gráfico

A continuación, se muestra el gráfico generado a partir del código Python anterior:



Ejercicio 10: Resolución

Enunciado

Un sistema de mensajería tiene una latencia $L(x) = 100 - 2x$, donde x es el número de mensajes por segundo. La latencia no puede ser inferior a **20 ms**. Se debe maximizar el número de mensajes enviados sin que la latencia caiga por debajo de este límite.

Paso 1: Definir la ecuación de latencia

La latencia en función de x es:

$$L(x) = 100 - 2x$$

Paso 2: Restricción de la latencia

La latencia mínima permitida es **20 ms**:

$$100 - 2x \geq 20$$

Paso 3: Resolver para x

Resolviendo la desigualdad:

$$100 - 2x \geq 20$$

$$-2x \geq 20 - 100$$

$$-2x \geq -80$$

$$x \leq \frac{80}{2} = 40$$

Paso 4: Conclusión

El número máximo de mensajes que se pueden enviar sin que la latencia caiga por debajo de **20 ms** es **40 mensajes por segundo**.

Código Python y Resultados del Gráfico

Código Python

A continuación, se presenta el código Python utilizado para la generación del gráfico de latencia vs número de mensajes:

```
import numpy as np
import matplotlib.pyplot as plt
import streamlit as st

# Configuraci n del dise o para que ocupe todo el ancho
st.set_page_config(layout="wide")

# Dividir la pantalla en tres columnas: enunciado, gr fico, e interpretaci n
col1, col2, col3 = st.columns([1, 2, 1])

# Columna 1: Enunciado
with col1:
    st.write("""
    ### Descripci n del Ejercicio 10
    Un sistema de mensajer a tiene una latencia \(L(x)\) definida como:

    \[
    L(x) = 100 - 2x
    \]

    donde \(x\) es el n mero de mensajes enviados por segundo. La latencia no puede ser inferior a **20 ms** debido a restricciones del protocolo.
    El objetivo es maximizar el n mero de mensajes enviados sin que la latencia caiga por debajo de este l mite.
    """)

# Funci n para calcular la latencia
def latencia(x):
    return 100 - 2 * x

# Par metro de entrada: n mero de mensajes por segundo
x_val = np.linspace(0, 60, 100)
latencias = latencia(x_val)
x_max = 40  # N mero m ximo de mensajes sin que la latencia baje de 20 ms

# Par metro de entrada: n mero de mensajes por segundo seleccionado por el usuario
numero_mensajes = st.slider(
    'Selecciona el n mero de mensajes por segundo',
    min_value=0,
```

```
38      max_value=60,
39      value=0
40  )
41
42  # C lculo de la latencia seleccionada
43  latencia_seleccionada = latencia(numero_mensajes)
44
45  # Columna 2: Gr fico (sin modificaciones)
46  with col2:
47      st.title('Latencia vs N mero de Mensajes')
48
49      # Creaci n del gr fico
50      fig, ax = plt.subplots(figsize=(10, 6))  # Mantengo el
    tama o original
51      ax.plot(x_val, latencias, label='Latencia (ms)', color='b
    ')
52      ax.axhline(y=20, color='r', linestyle='--', label='
    Latencia m nima de 20 ms')
53      ax.axvline(x=x_max, color='g', linestyle='--', label=f'
    M ximo en x={x_max}')
54      ax.plot(numero_mensajes, latencia_seleccionada, 'go',
    label='Seleccionado')
55      ax.set_title('Latencia vs N mero de Mensajes')
56      ax.set_xlabel('N mero de mensajes por segundo')
57      ax.set_ylabel('Latencia (ms)')
58      ax.grid(True)
59      ax.legend()
60
61      # Mostrar el gr fico en Streamlit
62      st.pyplot(fig)
63
64  # Columna 3: Interpretaci n
65  with col3:
66      st.write("""
67      ### Interpretaci n del Gr fico
68      - **N mero de Mensajes por Segundo (x)**: El gr fico
    muestra c mo la latencia disminuye a medida que aumenta
    el n mero de mensajes enviados por segundo.
69      - **Latencia M nima**: La latencia no puede caer por
    debajo de **20 ms** debido a restricciones del protocolo.
    Esto establece un l mite en el n mero de mensajes que se
     pueden enviar sin reducir la latencia por debajo de este
    valor.
70      - **Conclusi n**: El n mero m ximo de mensajes que se
    pueden enviar sin que la latencia caiga por debajo de **20
     ms** es aproximadamente \(x = {x_max}\). Para \(x = {
    numero_mensajes}\), la latencia es de **{
    latencia_seleccionada:.2f} ms**.
71      """)
```

```
72
73      # Mostrar la latencia para el n mero seleccionado de
        mensajes
74      st.write(f"**N mero m ximo de mensajes por segundo**: {
        x_max}")
75      st.write(f"**Latencia para {numero_mensajes} mensajes por
         segundo**: {latencia_seleccionada:.2f} ms")
```

Listing 10: Código Python

Resultado del Gráfico

A continuación, se muestra el gráfico generado a partir del código Python anterior: