ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

METODOS NUMERICOS (GR1CC)

PROYECTO IB

Alumnos:

Erick Eduardo Carcelén P.
Luis Ángel Morocho S.
Andrés Alejandro Pérez P.
Juan Esteban Flores C.

PROFESOR: Jonathan Alejandro Zea G.

FECHA DE ENTREGA: 18/06/2024

ESCUELA POLITÉCNICA NACIONAL Métodos Numéricos Computación		Proyecto IB
NOMBRES:	Erick Carcelén, Luis Morocho, Andrés Pérez, Juan Flores	

1. OBJETIVOS

- Adaptar la fórmula básica del interés compuesto proporcionada por los técnicos del banco para incluir aportes semanales, mensuales, bimestrales y trimestrales.
- Diseñar y desarrollar una interfaz de usuario intuitiva y fácil de usar que permita a los clientes del banco ingresar sus datos de depósito inicial, frecuencia de aportes, monto de los aportes y tasa de interés.
- Implementar la funcionalidad para que la calculadora genere tablas de cálculo detalladas como la proporcionada en el ejemplo, mostrando el aporte, capital, ganancia y total para cada periodo.
- Desarrollar una funcionalidad que permita a los usuarios determinar la tasa de interés efectiva a partir de diferentes escenarios de ahorro ingresados.
- Documentar la calculadora completa, incluyendo una descripción detallada del código creado.
- Implementar validaciones para garantizar que los aportes y el depósito inicial cumplan con los mínimos establecidos.
- Incluir gráficos que ilustren la evolución del valor total en la cuenta de ahorros, facilitando la interpretación visual de los datos.
- Proveer una tabla detallada que muestre la evolución del capital, las ganancias y el total en cada periodo, para una mejor comprensión de cómo crece la inversión con el tiempo.

2. INTRODUCCIÓN

En el ámbito de las finanzas personales, la planificación y el análisis de inversiones son esenciales para maximizar los rendimientos y asegurar una gestión efectiva del ahorro. Este informe presenta una aplicación interactiva desarrollada en Jupyter Notebook, que permite a los usuarios calcular el valor futuro de una cuenta de ahorros basada en varios parámetros, como el depósito inicial, los aportes periódicos, la tasa de interés anual y la frecuencia de los aportes. Mediante el uso de bibliotecas como ipywidgets para la creación de interfaces interactivas y matplotlib para la visualización de datos, esta herramienta no solo facilita el cálculo del valor final de las inversiones, sino que también proporciona una tabla detallada de resultados y gráficos que ilustran la evolución del ahorro a lo largo del tiempo.

El objetivo de este informe es detallar el diseño y la implementación de esta herramienta, discutir los resultados obtenidos y analizar su utilidad en la planificación financiera personal. La importancia de contar con herramientas interactivas y visuales radica en su capacidad para simplificar conceptos financieros complejos, permitiendo a los usuarios experimentar con diferentes escenarios de inversión y observar sus impactos de manera intuitiva. Además, se explorará el cálculo de la tasa de interés requerida para alcanzar un valor futuro deseado, demostrando la flexibilidad y potencia de los métodos numéricos aplicados.

Para desarrollar esta aplicación, se emplearon tecnologías clave en el análisis de datos y visualización. Jupyter Notebook se utiliza como plataforma principal debido a su capacidad para integrar código, visualizaciones y textos explicativos en un entorno interactivo. Las bibliotecas ipywidgets y matplotlib se integran para ofrecer una interfaz de usuario amigable y gráficos dinámicos, mientras que pandas y numpy facilitan el manejo y procesamiento eficiente de datos financieros. Este enfoque permite a los usuarios introducir datos de manera sencilla y obtener resultados visualmente atractivos y fáciles de interpretar.

El informe aborda la implementación detallada del código, explicando las funciones principales y la lógica detrás de ellas. La herramienta permite calcular el valor final de una cuenta de ahorros a lo largo del tiempo, considerando diferentes frecuencias de aportes y tasas de interés. Además, se proporciona una función adicional para calcular la tasa de interés necesaria para alcanzar un objetivo financiero específico, utilizando el método de Newton-Raphson. Este método numérico es especialmente útil para resolver ecuaciones no lineales que surgen en contextos financieros complejos.

Finalmente, se analizan los resultados obtenidos con diferentes parámetros de entrada, comparando escenarios con diversas frecuencias de aportes y tasas de interés. Se discuten las ventajas y limitaciones de la herramienta, así como su potencial impacto en la educación financiera y en la toma de decisiones informadas sobre inversiones personales. Concluimos con recomendaciones para futuras mejoras y extensiones de la herramienta, subrayando la importancia de continuar desarrollando recursos que faciliten la comprensión y gestión de las finanzas personales.

3. DESARROLLO

Descripción del Proyecto:

Un prestigioso banco lo ha contratado para implementar la calculadora de un programa de ahorro moderno. Este programa consiste en:

- Un depósito inicial de al menos 50 dólares y
- Aportes semanales de al menos 5 dólares.

Los técnicos del banco le han proporcionado la fórmula básica para calcular el interés compuesto:

$\boldsymbol{V_F = V_O(1 + i)^n}$ Donde:

$\boldsymbol{V_F}$ es el valor final en la cuenta de ahorros,
$\boldsymbol{V_O}$ es el valor de depósito inicial,
i es la tasa de interés,
n es el número de periodos transcurridos.

Sin embargo, los técnicos del banco desconocen cómo se debe calcular los intereses en base a los aportes semanales. Y en su lugar le han dado el siguiente ejemplo:

Ejemplo:

Dado un depósito inicial de 100 dólares, aportes semanales de 5 dólares y una tasa de interés anual del 8%, la tabla de cálculo sería la siguiente:

ESCUELA POLITÉCNICA NACIONAL Métodos Numéricos Computación		Proyecto IB
NOMBRES:	Erick Carcelén, Luis Morocho, Andrés Pérez, Juan Flores	

Semana	Aporte ($)	Capital ($)	Ganancia ($)	Total ($)
1	100	100	0.15	100.15
2	5	105.15	0.16	105.31
3	5	110.31	0.17	110.48
4	5	115.48	0.18	115.66
5	5	120.66	0.19	120.85
...				
51	5	367.65	0.57	368.22
52	5	373.22	0.57	373.79

Desarrollo matemático:

1. Interés Compuesto Periódico

El problema se centra en calcular el valor final de una cuenta de ahorros con interés compuesto, que recibe aportes periódicos y tiene una tasa de interés anual. El valor final \(V \) después de \(n \) periodos se calcula sumando los aportes periódicos y los intereses generados en cada periodo.

Para cada periodo i:

$$V_i = V_{(i-1)} + A + \left(V_{i-1} \times \frac{rp}{100}\right)$$

donde:
- V_i es el valor de la cuenta al final del periodo i.
- V_{i-1}es el valor de la cuenta al final del periodo anterior $i-1$.
- A es el aporte periódico.
- rp es la tasa de interés periódica, derivada de la tasa de interés anual r_a y la frecuencia de periodos por año p:

$$r_p = \frac{r_a}{p}$$

2. Método de Bisección

El método de bisección es una técnica numérica para encontrar raíces de funciones continuas. En este caso, estamos buscando la tasa de interés anual r_a que hace que el valor final V sea igual a un objetivo de valor final $V_{objetivo}$.

El método de bisección sigue estos pasos:

1. Definir el intervalo inicial: Supongamos que la tasa de interés anual r_a estará entre 0% y 100%, es decir, $a = 0$ y $b = 100$.

2. Evaluar la función en los extremos del intervalo:

$$f(a) = V(a) - V_{objetivo}$$
$$f(b) = V(b) - V_{objetivo}$$

3. Verificar que el producto de $f(a)$ y $f(b)$ sea negativo: Esto asegura que la función cambia de signo en el intervalo y que hay una raíz (punto donde la función se anula).

$$f(a) \cdot f(b) < 0$$

4. Iterar para reducir el intervalo:
- Calcular el punto medio del intervalo:
$c = \frac{a+b}{2}$
- Evaluar la función en el punto medio:
$f(c) = V(c) - V_{objetivo}$
- Si $f(c)$ es suficientemente cercano a 0 (dentro de una tolerancia predefinida $\in$), entonces c es la raíz buscada.
- Si $f(a) \cdot f(c) < 0$, entonces la raíz está en el intervalo $[a, c]$. Actualizar b:
$b = c$
- De lo contrario, actualizar a:
$a = c$

5. Repetir hasta converger o alcanzar el número máximo de iteraciones: Si el intervalo se ha reducido suficientemente o el número máximo de iteraciones se ha alcanzado sin encontrar una solución precisa, el proceso se detiene.

3. Aplicación al Problema

Para aplicar el método de bisección en el cálculo de la tasa de interés anual:
1. Definimos la función $V(r_a)$ que calcula el valor final de la cuenta con una tasa de interés anual r_a.
2. Evaluamos $V(r_a)$ en los extremos del intervalo inicial.
3. Usamos el método de bisección para iterar y encontrar la tasa de interés r_a que hace que $V(r_a) \approx V_{objetivo}$.

Esto asegura que la tasa de interés anual encontrada es la que hace que el valor final de la cuenta de ahorros sea igual al objetivo de valor final dentro de un margen de error aceptable.

Fórmulas y Pasos

1. Valor Final con Aportes Periódicos:

$$V_i = V_{(i-1)} + A + \left(V_{i-1} \times \frac{rp}{100}\right)$$
$$r_p = \frac{r_a}{p}$$

2. Método de Bisección:

$$f(r_a) = V(r_a) - V_{objetivo}$$
$$a, b = 0, 100$$
$$c = \frac{a+b}{2}$$

Iterar hasta que:

$$|f(c)| < \epsilon$$

Pseudocódigo:

1. Importar bibliotecas necesarias:
- ipywidgets as widgets
- IPython.display (display, clear_output)
- matplotlib.pyplot as plt
- pandas as pd
- numpy as np

ESCUELA POLITÉCNICA NACIONAL Métodos Numéricos Computación		Proyecto IB
NOMBRES:	Erick Carcelén, Luis Morocho, Andrés Pérez, Juan Flores	

2. Definir función calcular_valor_final_y_tabla:
   Input: deposito_inicial, aporte_periodico, tasa_interes_anual, meses, frecuencia
   Output: valor_final, tabla_resultados

   - Inicializar diccionario frecuencias con valores para 'semanal', 'mensual', 'bimestral', 'trimestral'
   - Validar que frecuencia esté en frecuencias
   - Calcular periodos_por_ano y tasa_interes_periodica
   - Calcular total_periodos
   - Inicializar valor_final con deposito_inicial
   - Inicializar lista tabla_resultados

   - Para cada periodo en el rango de 1 a total_periodos:
     - Si periodo > 1, sumar aporte_periodico a valor_final
     - Calcular ganancia como valor_final * (tasa_interes_periodica / 100)
     - Sumar ganancia a valor_final
     - Agregar registro a tabla_resultados con datos del periodo actual

   - Retornar valor_final y tabla_resultados

3. Definir función solicitar_valores:
   Output: deposito_inicial, aporte_periodico, frecuencia, meses, tasa_interes_anual, objetivo_valor_final

   - Obtener valores de widgets: deposito_inicial, aporte_periodico, frecuencia, meses, tasa_interes_anual, objetivo_valor_final
   - Validar que deposito_inicial sea >= 50, si no, mostrar mensaje de error y retornar None
   - Validar que aporte_periodico sea >= 5, si no, mostrar mensaje de error y retornar None
   - Validar que frecuencia esté en ['semanal', 'mensual', 'bimestral', 'trimestral'], si no, mostrar mensaje de error y retornar None
   - Validar que meses sea > 0, si no, mostrar mensaje de error y retornar None
   - Validar que tasa_interes_anual sea positiva si está definida, si no, mostrar mensaje de error y retornar None

   - Retornar valores obtenidos

4. Definir función actualizar_grafica:
   Input: tabla_resultados

   - Obtener periodos y totales de tabla_resultados
   - Crear gráfica con plt mostrando la evolución del valor total

5. Definir función calcular_interes_anual_biseccion:
   Input: deposito_inicial, aporte_periodico, meses, frecuencia, objetivo_valor_final, tol, max_iter
   Output: tasa_interes_anual

   - Inicializar diccionario frecuencias con valores para 'semanal', 'mensual', 'bimestral', 'trimestral'
   - Calcular periodos_por_ano y total_periodos

   - Definir función valor_final_con_tasa:
     Input: tasa_interes_anual
     Output: valor_final

     - Calcular tasa_interes_periodica y valor_final inicial
     - Para cada periodo en el rango de 1 a total_periodos:
       - Si periodo > 1, sumar aporte_periodico a valor_final

- Calcular ganancia y sumar a valor_final
- Retornar valor_final

- Inicializar intervalo [a, b] con 0 y 100
- Calcular fa y fb usando valor_final_con_tasa(a) y valor_final_con_tasa(b)
- Si fa y fb tienen el mismo signo, lanzar error

- Para cada iteración en el rango de max_iter:
  - Calcular punto medio c
  - Calcular fc usando valor_final_con_tasa(c)
  - Si fc está dentro de la tolerancia, retornar c
  - Si fa y fc tienen signos opuestos, actualizar b y fb, si no, actualizar a y fa

- Lanzar error si no converge en max_iter

6. Definir función main:
   Input: b

- Llamar a solicitar_valores
- Si los valores son None, retornar
- Desempaquetar valores obtenidos

- Si tasa_interes_anual es None y objetivo_valor_final no es None:
  - Calcular tasa_interes_anual usando calcular_interes_anual_biseccion
  - Mostrar tasa de interés calculada

- Si tasa_interes_anual sigue siendo None, mostrar mensaje de error y retornar

- Llamar a calcular_valor_final_y_tabla con los valores obtenidos
- Mostrar resultados y actualizar gráfica

7. Configurar widgets y evento del botón:
   - Crear widgets para deposito_inicial, aporte_periodico, tasa_interes_anual, frecuencia, meses, objetivo_valor_final, imprimir_tabla
   - Crear botón calcular_button y definir su evento on_click como main
   - Mostrar widgets y output usando display

Documentación:

1. **Importaciones:**
   Se importan bibliotecas necesarias para los widgets (ipywidgets), para la visualización de gráficos (matplotlib), para manejar la visualización en Jupyter Notebook (IPython.display), y para el manejo de datos (pandas y numpy).

```
# Importo la biblioteca ipywidgets para crear widgets interactivos en Jupyter Notebook
import ipywidgets as widgets
# Importo funciones para mostrar y limpiar la salida en Jupyter Notebook
from IPython.display import display, clear_output
# Importo la biblioteca matplotlib para crear gráficos
import matplotlib.pyplot as plt
# Importo la biblioteca pandas para el manejo de datos en estructuras tipo DataFrame
import pandas as pd
# Importo la biblioteca numpy para operaciones numéricas
import numpy as np
```

2. **Función calcular_valor_final_y_tabla:**
   Define una función para calcular el valor final y la tabla de resultados. Inicialmente, se define un diccionario que traduce la frecuencia de los aportes en número de periodos por año.

```python
# Define una función para calcular el valor final de una cuenta de ahorros y generar una tabla de resultados
def calcular_valor_final_y_tabla(deposito_inicial, aporte_periodico, tasa_interes_anual, meses, frecuencia):
    # Diccionario que convierte la frecuencia en el número de periodos por año
    frecuencias = {
        'semanal': 52,
        'mensual': 12,
        'bimestral': 6,
        'trimestral': 4
    }
```

3. **Validación de frecuencia:**
   Se verifica si la frecuencia ingresada es válida.

```python
# Verifico si la frecuencia es válida
if (frecuencia not in frecuencias):
    raise ValueError("Frecuencia no válida. Debe ser 'semanal', 'mensual', 'bimestral' o 'trimestral'.")
```

4. **Cálculo de periodos:**
   Se calculan los periodos por año, la tasa de interés por periodo y el número total de periodos.

```python
# Obtiene el número de periodos por año y la tasa de interés por periodo
periodos_por_ano = frecuencias[frecuencia]
tasa_interes_periodica = tasa_interes_anual / periodos_por_ano
# Calcula el número total de periodos en función del número de meses
total_periodos = int(meses * (periodos_por_ano / 12))
```

5. **Inicialización de variables:**
   Se inicializan el valor final con el depósito inicial y una lista para almacenar los resultados de cada periodo.

```python
# Inicializa el valor final con el depósito inicial y crea una lista para almacenar los resultados de cada periodo
valor_final = deposito_inicial
tabla_resultados = []
```

6. **Cálculo por periodo:**
   Se itera sobre cada periodo, calculando la ganancia de interés y actualizando el valor final. El aporte periódico se agrega a partir del segundo periodo.

```python
# Itera sobre cada periodo para calcular el valor final
for periodo in range(1, total_periodos + 1):
    # Calcula la ganancia por intereses en el periodo actual
    ganancia = valor_final * tasa_interes_periodica
    if periodo == 1:
        valor_final += ganancia
        aporte = 0  # No hay aporte periódico en el primer periodo
    else:
        valor_final += ganancia + aporte_periodico
        aporte = aporte_periodico  # Aporte periódico a partir del segundo periodo
```

7. **Almacenamiento de resultados:**
   Se almacenan los datos de cada periodo en la lista de resultados.

```
# Almaceno los resultados del periodo actual en la lista
tabla_resultados.append({
    'Periodo': periodo,
    'Aporte ($)': aporte,
    'Capital ($)': valor_final - ganancia - aporte,
    'Ganancia ($)': ganancia,
    'Total ($)': valor_final
})
```

8. **Retorno de resultados:**
   La función devuelve el valor final y la tabla de resultados.

```
# Devuelve el valor final y la tabla de resultados
return valor_final, tabla_resultados
```

9. **Función solicitar_valores:**
   Esta función obtiene y valida los valores de entrada de los widgets.

```
# Define una función para solicitar y validar los valores de entrada
def solicitar_valores():
    # Obtiene el valor del widget de depósito inicial y lo convierte a float
    deposito_inicial = float(deposito_inicial_widget.value)
    if deposito_inicial < 50:
        # Si el depósito inicial es menor a $50, muestro un mensaje de error y termino la función
        with output_widget:
            clear_output()
            print("El depósito inicial debe ser de al menos $50.")
        return None
```

10. **Validación del aporte periódico:**
    Se verifica si el aporte periódico es al menos $5.

```
# Obtiene el valor del widget de aporte periódico y lo convierte a float
aporte_periodico = float(aporte_periodico_widget.value)
if aporte_periodico < 5:
    # Si el aporte periódico es menor a $5, muestro un mensaje de error y termino la función
    with output_widget:
        clear_output()
        print("El aporte periódico debe ser de al menos $5.")
    return None
```

11. **Validación de la frecuencia:**
    Se verifica si la frecuencia es válida.

```
# Obtiene el valor del widget de frecuencia
frecuencia = frecuencia_widget.value
if frecuencia not in ['semanal', 'mensual', 'bimestral', 'trimestral']:
    # Si la frecuencia no es válida, muestro un mensaje de error y termino la función
    with output_widget:
        clear_output()
        print("Frecuencia no válida. Debe ser 'semanal', 'mensual', 'bimestral' o 'trimestral'.")
    return None
```

12. **Validación de meses:**
    Se verifica si el número de meses es mayor a 0.

```
# Obtiene el valor del widget de meses y lo convierte a entero
meses = int(meses_widget.value)
if meses <= 0:
    # Si el número de meses es menor o igual a 0, muestro un mensaje de error y termino la función
    with output_widget:
        clear_output()
        print("El número de meses debe ser mayor a 0.")
    return None
```

13. **Retorno de valores:**
    La función devuelve los valores si son válidos.

```
# Devuelve los valores obtenidos si son válidos
return deposito_inicial, aporte_periodico, frecuencia, meses
```

14. **Función actualizar_grafica:**
    Esta función toma la tabla de resultados y crea una gráfica de la evolución del valor total en la cuenta de ahorros.

```
# Define una función para actualizar la gráfica con los resultados
def actualizar_grafica(tabla_resultados):
    # Extrae los periodos y los totales de la tabla de resultados
    periodos = [fila['Periodo'] for fila in tabla_resultados]
    totales = [fila['Total ($)'] for fila in tabla_resultados]

    # Crea una figura y grafica los totales a lo largo de los periodos
    plt.figure(figsize=(10, 6))
    plt.plot(periodos, totales, marker='o')
    plt.title('Evolución del Valor Total en la Cuenta de Ahorros')
    plt.xlabel('Periodo')
    plt.ylabel('Total ($)')
    plt.grid(True)
    plt.show()
```

15. **Función main:**
    Esta función se ejecuta cuando se hace clic en el botón de calcular. Obtiene los valores de los widgets, realiza el cálculo del valor final y la tabla de resultados, y luego muestra la tabla y la gráfica.

```python
# Define la función principal que se ejecuta al hacer clic en el botón
def main(b):
    # Solicita los valores de entrada
    valores = solicitar_valores()
    if valores is None:
        return  # Si los valores no son válidos, termina la función

    # Desempaqueta los valores obtenidos
    deposito_inicial, aporte_periodico, frecuencia, meses = valores

    # Define una tasa de interés anual fija del 8%
    tasa_interes_anual = 0.08

    # Calcula el valor final y la tabla de resultados
    valor_final, tabla_resultados = calcular_valor_final_y_tabla(deposito_inicial, aporte_periodico, tasa_interes_anual, meses, frecuencia)

    # Muestra los resultados en el widget de salida
    with output_widget:
        clear_output()
        # Si la opción de imprimir la tabla está seleccionada, muestra la tabla
        if imprimir_tabla_widget.value:
            df = pd.DataFrame(tabla_resultados)
            display(df)
        # Muestra el valor final de la cuenta de ahorros
        print(f"\nEl valor final en la cuenta de ahorros es: ${valor_final:.2f}")
        # Actualiza la gráfica con los resultados
        actualizar_grafica(tabla_resultados)
```

16. **Widgets de entrada:**
    Se crean widgets para la entrada de los datos: depósito inicial, aporte periódico, frecuencia, número de meses y una opción para imprimir la tabla de resultados.

```python
# Define los widgets de entrada para el cálculo principal
deposito_inicial_widget = widgets.FloatText(description="Depósito Inicial ($):", value=50)
aporte_periodico_widget = widgets.FloatText(description="Aporte Periódico ($):", value=5)
frecuencia_widget = widgets.Dropdown(
    options=['semanal', 'mensual', 'bimestral', 'trimestral'],
    description="Frecuencia:"
)
meses_widget = widgets.IntText(description="Meses:", value=12)
imprimir_tabla_widget = widgets.Checkbox(description="Imprimir Tabla de Resultados", value=True)
calcular_button = widgets.Button(description="Calcular")

# Define un widget de salida para mostrar los resultados
output_widget = widgets.Output()
```

17. **Configuración de eventos y despliegue de widgets:**
    Se configura el botón para llamar a la función main al hacer clic y se muestran los widgets en la interfaz.

```python
# Configura el botón para que llame a la función principal al hacer clic
calcular_button.on_click(main)

# Muestra los widgets en la interfaz de usuario
display(deposito_inicial_widget, aporte_periodico_widget, frecuencia_widget, meses_widget, imprimir_tabla_widget, calcular_button, output_widget)
```

18. **Función calcular_tasa_interes_anual_biseccion:**
    Define una función para calcular la tasa de interés necesaria usando el método de bisección.

```python
def calcular_interes_anual_biseccion(deposito_inicial, aporte_periodico, meses, frecuencia, objetivo_valor_final, tol=1e-6, ma
    frecuencias = {
        'semanal': 52,
        'mensual': 12,
        'bimestral': 6,
        'trimestral': 4
    }
    periodos_por_ano = frecuencias[frecuencia]
    total_periodos = int(meses * (periodos_por_ano / 12))

    def valor_final_con_tasa(tasa_interes_anual):
        tasa_interes_periodica = tasa_interes_anual / periodos_por_ano
        valor_final = deposito_inicial

        for periodo in range(1, total_periodos + 1):
            if periodo > 1:
                valor_final += aporte_periodico
            ganancia = valor_final * (tasa_interes_periodica / 100)
            valor_final += ganancia

        return valor_final
```

```python
# Itero para encontrar la tasa de interés usando el método de Newton-Raphson
for i in range(max_iter):
    f_val = f(r)
    f_prime_val = f_prime(r)
    r_new = r - f_val / f_prime_val

    if abs(r_new - r) < tol:
        return r_new * 100  # Retorno la tasa como porcentaje

    r = r_new

# Si no converge, lanzo un error
raise ValueError("El método de Newton-Raphson no convergió.")
```

19. **Función solicitar_valores_tasa:**
Esta función obtiene y valida los valores de entrada para calcular la tasa de interés, y maneja errores si la tasa no converge.

```python
# Define una función para solicitar y validar los valores de entrada para el cálculo de la tasa de interés
def solicitar_valores_tasa(e):
    # Obtiene y convierte los valores de los widgets
    FV_deseado = float(FV_deseado_widget.value)
    PV = float(PV_widget.value)
    P = float(P_widget.value)
    meses = int(meses_widget_tasa.value)
    frecuencia = frecuencia_tasa_widget.value

    try:
        # Calcula la tasa de interés requerida
        tasa_interes = calcular_tasa_interes(FV_deseado, PV, P, meses, frecuencia)
        # Muestra el resultado en el widget de salida
        with output_widget_tasa:
            clear_output()
            print(f"La tasa de interés necesaria es: {tasa_interes:.4f}%")
    except ValueError as e:
        # Muestra un mensaje de error si no converge
        with output_widget_tasa:
            clear_output()
            print(str(e))
```

20. **Widgets para calcular la tasa de interés:**
Se crean widgets específicos para la entrada de datos relacionados con el cálculo de la tasa de interés.

```
# Define los widgets para la entrada de valores relacionados con la tasa de interés
FV_deseado_widget = widgets.FloatText(description="Valor Final Deseado ($):", value=20000)
PV_widget = widgets.FloatText(description="Depósito Inicial ($):", value=1000)
P_widget = widgets.FloatText(description="Aporte Periódico ($):", value=100)
meses_widget_tasa = widgets.IntText(description="Meses:", value=60)
frecuencia_tasa_widget = widgets.Dropdown(
    options=['semanal', 'mensual', 'bimestral', 'trimestral'],
    description="Frecuencia:"
)
calcular_tasa_button = widgets.Button(description="Calcular Tasa")
output_widget_tasa = widgets.Output()
```

21. **Configuración de eventos y despliegue de widgets:**
Se configura el botón para calcular la tasa de interés y se muestran los widgets en la interfaz.

```
# Configuro el botón para que llame a la función de cálculo de la tasa de interés al hacer clic
calcular_tasa_button.on_click(solicitar_valores_tasa)

# Muestro los widgets en la interfaz de usuario
display(FV_deseado_widget, PV_widget, P_widget, meses_widget_tasa, frecuencia_tasa_widget, calcular_tasa_button, output_widget_tasa)
```

Parte 1: Cálculo del Valor Final y Generación de la Tabla de Resultados

Entradas del Usuario:

- Depósito inicial (mínimo $50)
- Aporte periódico (mínimo $5)
- Frecuencia de los aportes (semanal, mensual, bimestral, trimestral)
- Número de meses
- Tasa de interés anual (opcional)
- Objetivo de valor final (opcional)
- Opción de imprimir la tabla de resultados

Procedimiento:

Se definieron las frecuencias posibles de los aportes y se calcularon los periodos por año para cada frecuencia.

Se calculan los valores finales y se genera una tabla de resultados que muestra, para cada periodo:

Periodo

Aporte realizado

Capital acumulado (sin ganancias)

Ganancias obtenidas en el periodo

Total, acumulado (capital + ganancias)

Para cada periodo, se calculan las ganancias y se suman al capital y aporte realizado, iterando por el número total de periodos especificados.

Parte 2: Cálculo de la Tasa de Interés Requerida

- Entradas del Usuario:
- Valor final deseado
- Depósito inicial
- Aporte periódico
- Número de meses
- Frecuencia de los aportes (semanal, mensual, bimestral, trimestral)

Procedimiento:

Se utilizó el método de bisección para encontrar la tasa de interés requerida.

La función objetivo se define como la diferencia entre el valor final calculado con una tasa de interés supuesta y el valor final deseado.

Se ajusta la tasa de interés hasta que la función objetivo sea cercana a cero dentro de una tolerancia definida.

Implementación

El código se desarrolló en Python utilizando ipywidgets para la creación de interfaces interactivas que permiten al usuario ingresar los valores necesarios y visualizar los resultados.

Visualización

Se implementaron gráficos utilizando matplotlib para mostrar la evolución del valor total en la cuenta de ahorros a lo largo del tiempo, proporcionando una representación visual clara de cómo se acumulan los ahorros y las ganancias con el tiempo.

Implementación del Código

Cálculo del Valor Final y Tabla de Resultados

Funciones Principales:

- **`calcular_valor_final_y_tabla`**: Calcula el valor final en una cuenta de ahorros con aportes periódicos y genera una tabla de resultados que detalla los aportes, el capital acumulado, las ganancias y el total acumulado en cada periodo.

- **`solicitar_valores`**: Solicita los valores de entrada del usuario, como depósito inicial, aporte periódico, frecuencia y número de meses, y valida que cumplan con los requisitos mínimos.
- **`actualizar_grafica`**: Toma los resultados de la tabla y genera un gráfico que muestra la evolución del valor total en la cuenta de ahorros a lo largo del tiempo.

Widget e Interactividad:

- **Widgets para la entrada de datos**: Se crean widgets usando `ipywidgets` para permitir al usuario ingresar los valores necesarios (depósito inicial, aporte periódico, frecuencia, meses).
- **Botón de cálculo**: Un botón que, al ser presionado, llama a la función main para realizar los cálculos y mostrar los resultados.
- **`main`**: Coordina la entrada de valores, llama a `calcular_valor_final_y_tabla` y `actualizar_grafica`, y muestra los resultados y la tabla (si está habilitada la opción).

Cálculo de la Tasa de Interés Requerida

Funciones Principales:

- **`calcular_interes_anual_biseccion`**: Utiliza el método de bisección para calcular la tasa de interés necesaria para alcanzar un valor final deseado. Define una función objetivo para iterar y encontrar la tasa de interés correcta.
- **`solicitar_valores`**: Solicita los valores necesarios del usuario (valor final deseado, depósito inicial, aporte periódico, meses y frecuencia) y llama a `calcular_interes_anual_biseccion` para realizar el cálculo.

Widget e Interactividad:

- **Widgets para la entrada de datos**: Se crean widgets para permitir al usuario ingresar los valores necesarios para calcular la tasa de interés.
- **Botón de cálculo**: Un botón que, al ser presionado, llama a la función main para realizar los cálculos y mostrar los resultados.
- **`main`**: Coordina la entrada de valores, llama a `calcular_interes_anual_biseccion` y muestra los resultados.

Conclusiones

- Aunque la calculadora es robusta y funcional, siempre hay margen para mejoras y extensiones, como la inclusión de más opciones de frecuencia de aportes, la personalización de tasas de interés más complejas y la incorporación de análisis más avanzados de riesgo y retorno.
- La implementación de la fórmula del interés compuesto asegura que los resultados sean precisos y reflejen fielmente el crecimiento del capital a lo largo del tiempo, considerando tanto el depósito inicial como los aportes periódicos.
- La integración de gráficos facilita la comprensión visual del crecimiento de los ahorros, permitiendo a los usuarios interpretar rápidamente cómo varían los valores en función del tiempo y los aportes.
- La validación de los valores de entrada asegura que los cálculos sean precisos y evita errores derivados de valores fuera de los límites aceptables, mejorando la fiabilidad de la herramienta.
- La calculadora desarrollada proporciona una herramienta versátil que permite a los usuarios del banco planificar y analizar sus ahorros bajo diferentes escenarios de aportes y tasas de interés.
- Al implementar `ipywidgets`, la calculadora ofrece una interfaz amigable y accesible, facilitando la entrada de datos y la obtención inmediata de resultados.

- Al ofrecer una calculadora de ahorros moderna y fácil de usar, el banco puede mejorar la satisfacción del cliente al proporcionar un servicio adicional que ayuda a los clientes a gestionar sus finanzas de manera más efectiva y eficiente.
- La herramienta proporciona a los usuarios del banco la capacidad de tomar decisiones más informadas sobre sus planes de ahorro, al permitirles visualizar los efectos a largo plazo de diferentes estrategias de inversión.

Recomendaciones

1. Ampliar Opciones de Frecuencia de Aportes:

   Incluir opciones adicionales como aportes diarios, quincenales y anuales. Esto proporcionaría mayor flexibilidad y permitiría a los usuarios personalizar sus planes de ahorro de manera más precisa.

2. Incorporar Tasa de Interés Variable:

   Permitir la inclusión de tasas de interés variables o escalonadas a lo largo del tiempo. Esto sería útil para simular escenarios más realistas, donde las tasas de interés pueden cambiar según condiciones del mercado.

3. Añadir Funcionalidad de Retiro:

   Implementar la posibilidad de calcular retiros periódicos o puntuales. Esto ayudaría a los usuarios a planificar no solo sus ahorros, sino también sus gastos futuros.

4. Simulación de Escenarios de Mercado:

   Incluir la capacidad de simular diferentes escenarios económicos (como inflación, recesión, crecimiento económico) que afecten la tasa de interés y, por ende, el crecimiento de los ahorros.

5. Análisis de Riesgo y Retorno:

   Incorporar herramientas para analizar el riesgo y el retorno de diferentes estrategias de ahorro. Esto puede incluir métricas como el valor esperado, desviación estándar, y análisis de escenarios.

6. Optimización de Aportes:

   Implementar algoritmos que sugieran estrategias de ahorro óptimas para alcanzar un objetivo específico, minimizando el riesgo o maximizando el retorno.

7. Integración con Datos Reales:

   Permitir la importación de datos reales de cuentas de ahorro o inversiones, para que los usuarios puedan ver cómo se comportarían sus ahorros bajo diferentes estrategias y condiciones de mercado.

8. Mejorar la Interfaz de Usuario:

   Diseñar una interfaz más intuitiva y visualmente atractiva, quizás utilizando librerías como Plotly para gráficos interactivos, y agregando explicaciones y tutoriales dentro de la herramienta para ayudar a los usuarios a entender mejor las funcionalidades disponibles.

4. REFERENCIAS BIBLIOGRÁFICAS

- "ipywidgets Documentation," [Online]. Disponible: https://ipywidgets.readthedocs.io/en/latest/.
- "IPython.display — IPython 8.11.0 documentation," [Online]. Disponible: https://ipython.readthedocs.io/en/stable/api/generated/IPython.display.html.
- "Matplotlib: Pyplot API," [Online]. Disponible: https://matplotlib.org/stable/api/pyplot_summary.html.
- "pandas Documentation," [Online]. Disponible: https://pandas.pydata.org/pandas-docs/stable/.
- "NumPy Documentation," [Online]. Disponible: https://numpy.org/doc/stable/.