Portafolio

Física Computacional II

JOSÉ IGNACIO ROSAS SEPÚLVEDA
CIENCIAS FÍSICAS
DEPARTAMENTO
Facultad de Ciencias Físicas y Matemáticas
UNIVERSIDAD DE CONCEPCIÓN

Octubre-Diciembre 2024
Concepción, Chile

Profesor Guía: Dr. Roberto Navarro Maldonado

Índice general

Introducción

Este portafolio incluirá evidencias de aprendizaje relacionadas a la asignatura de Física Computacional II (510240), dictada en el segundo semestre de 2024 en el departamento de Física de la Universidad de Concepción.

Esta asignatura se enfoca en la resolución de problemas en Física usando métodos numéricos y el lenguaje de programación `python`. Al finalizar este portafolio, se espera que los y las estudiantes logren:

1. Aplicar herramientas computacionales en la resolución numérica de problemas en Física.
2. Generar programas computacionales basados en algoritmos y conceptos de la física matemática y estadística.
3. Diferenciar, integrar y resolver ecuaciones diferenciales ordinarias, en forma numérica.

La evaluación se realizará a través de este portafolio, el cual debe ser **entregado mediante un repositorio** de GitHub que será alojado en la organización fiscomp2-UdeC2024 especialmente habilitado para esta asignatura.

1. El repositorio será revisado periódicamente por el profesor o los ayudantes de la asignatura, para poder dar retroalimentación oportuna antes de la entrega oficial el día **viernes 29 de noviembre de 2024**. Si el protafolio no tiene avances en cada revisión parcial, a ocurrir a mediados de cada mes, puede costar puntaje de la nota final.
2. Las evidencias de aprendizaje serán la resolución de problemas **seleccionados por el estudiante** a partir de las guías del curso. Cada guía cuenta con una serie de problemas con dificultades ponderadas de 0 a 6 puntos, siendo 0 un problema de solución trivial o que no requiere mayor trabajo, y 6 un problema que podría ser desafiante. **Por cada guía, el estudiante debe elegir y responder problemas hasta completar 6 puntos**.
3. El portafolio debe contener una sección de conclusiones final donde el/la estudiante resumirá los principales logros de este portafolio y autoevaluará su desempeño a través de una reflexión final.

Criterios de evaluación

Como criterios de evaluación, se considerará:

1. Coherencia de las evidencias con los resultados de aprendizaje del curso y la autoevaluación al final del documento.
2. Redacción y competencias comunicativas, incluyendo ortografía, gramática, sintaxis, presentación de figuras, uso de LaTeX y explicación clara de la parte relevante de scripts de `python`.
3. Presentación: Claridad, limpieza y orden del documento.

Este portafolio, además de ser una herramienta de evaluación, es una oportunidad para que el/la estudiante reflexione sobre su proceso de aprendizaje, consolide sus conocimientos y desarrolle habilidades clave en el ámbito de la computación aplicada a la física. Se espera que este documento sirva como un registro tangible de su progreso y de las competencias adquiridas en la asignatura, las cuales podrán ser de gran utilidad en futuros desafíos académicos y profesionales.

Información personal y académica

Datos personales

Nombre completo	José Ignacio Rosas Sepúlveda
Matrícula	2023402508
Fecha de Nacimiento	2 de septiembre de 2000
Nacionalidad	Chileno
E-Mail institucional	jrosas2022@udec.cl

Breve biografía académica

Soy José Rosas, estudiante de segundo año de la carrera Ciencias Físicas, ingresado en el año 2023. Realicé mi educación media en el colegio Gran Bretaña de Concepción. Desde los 14 años me dediqué a la composición musical. En pandemia desarrolle un profundo interés por las ciencias. Mis objetivos académicos luego del pregrado son desarrollar mis conocimientos en física de partículas, me interesa investigar la fisica implicada en procesos bioquímicos. Por otra parte, deseo ampliar mis conocimientos en Biología para complementar mi formación científica.

Visión general e interés sobre la asignatura

Mi percepción inicial de Física Computacional II es que será clave para aplicar herramientas computacionales en problemas físicos complejos. La asignatura integra teoría, simulaciones y cálculos numéricos, fortaleciendo mi formación académica. Espero adquirir destrezas en software y algoritmos aplicados a la física, esenciales para abordar problemas multidisciplinarios. Confío en que este conocimiento será valioso en cursos avanzados y en mi futura carrera, especialmente en simulación y modelado físico. Me interesa particularmente el uso de técnicas numéricas en sistemas dinámicos.

Resultados esperados de este portafolio

Al completar este portafolio, espero consolidar mis habilidades en herramientas computacionales aplicadas a la física y en la resolución de problemas numéricos. Este documento organizará los conceptos aprendidos, facilitando su análisis y aplicación en diversos contextos. También será una referencia valiosa para mi formación futura, documentando ejemplos y metodologías reutilizables en proyectos académicos o profesionales. La autoevaluación y recopilación de evidencias me permitirán identificar áreas de mejora, evaluar mi progreso y fortalecer mi comprensión de los temas tratados.

Evidencias de aprendizaje

Capítulo 1

Estimación de la Derivada y Error Absoluto

Fecha de la actividad: 26 de noviembre de 2024

En este capítulo, se desarrolla un esquema de diferenciación numérica para estimar la primera derivada de una función $f(x)$ dada una serie de puntos (x_i, f_i, f_i'), con $f_i = f(x_i)$ y f_i' representando la derivada analítica. El esquema numérico utilizado es el siguiente:

$$f'(x) = \frac{f(x-3h) - 27f(x-h) + 27f(x+h) - f(x+3h)}{48h} + \frac{9}{5!}f^{(5)}(\xi)h^4. \tag{1.1}$$

1.1. Script de la estimación numérica la primera derivada de f

Implementando `python` sobre el esquema de diferencias finitas (1.1) de orden $O(h^4)$. Se define una función `numerical_derivative` con parámetros: `x` y `f`, ambos de tipo `array`. Estos parámetros representan datos (x_i, f_i). Se calcula h como la diferencia entre los puntos $x_1 - x_0$. Por limitaciones del esquema, se advierte que este solo puede aplicarse correctamente en el calculo de $f'_{i,\text{num}}$ para los x_i desde x_3 hasta x_{n-4}. La función retorna un par iterable con los x_i donde se calculo $f'_{i,\text{num}}$.

```
def numerical_derivative(x, f):
    h = x[1] - x[0]

    valid_indices = range(3, len(x) - 3)
    x_rec = x[valid_indices]
    deriv = [(f[i - 3] - 27 * f[i - 1] + 27 * f[i + 1] - f[i + 3]) / (48 * h) for i in valid_indices]
    return np.array(x_rec), np.array(deriv)
```

1.2. Gráficas de Comparación y Error Absoluto

A continuación, se presentan las gráficas comparando la derivada exacta $f'(x)$, la estimación numérica $f'_{\text{num}}(x)$, y el error absoluto para dos casos de prueba.

Se define una función `analyze_data` con parámetros: `x`, `f`, `df`, todos de tipo `array`. Estos parámetros representan datos (x_i, f_i, f_i'). Se llama a la función `numerical_derivative` para calcular la estimación de $f'_{i,\text{num}}$ para los $i \in \mathbb{N} : 3 < i < n - 3$. Se ajustan las listas que guardan los datos de x_i, f_i y f_i' de modo que coincidan con los valores estimados de f_i' calculados. El **error absoluto** entre la derivada exacta y la estimada se calcula como:

$$E_{\text{abs}} = |f'(x) - f'_{\text{num}}(x)|.$$

Se generan gráficas:

- Función original,
- Comparación de derivadas analítica vs numérica,
- Error absoluto de la función dada por la función original.

Probando el script en casos donde se entregan los valores de las tres series, dados en el problema, se obtienen las siguientes gráficas:

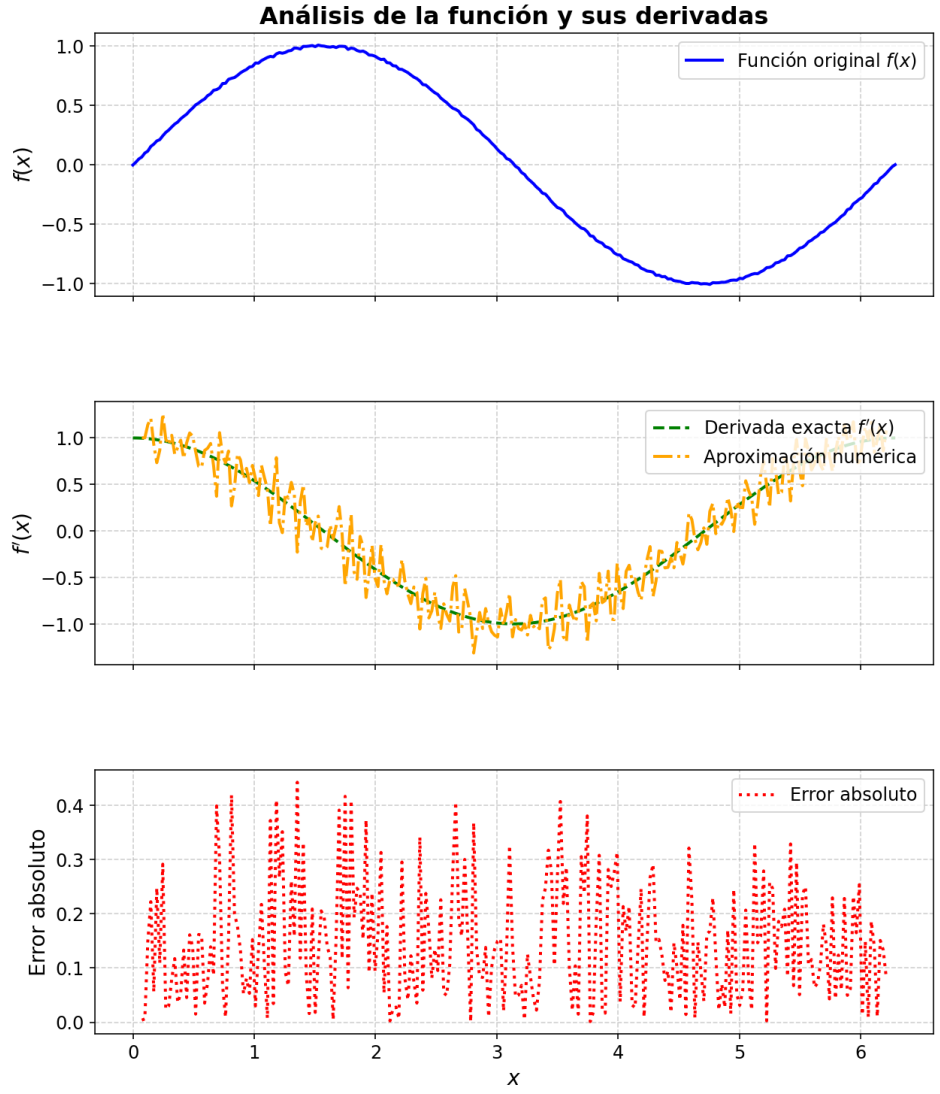

Figura 1.1: Función original, comparación de la derivada exacta, la estimación numérica y el error absoluto para una función sinusoidal.

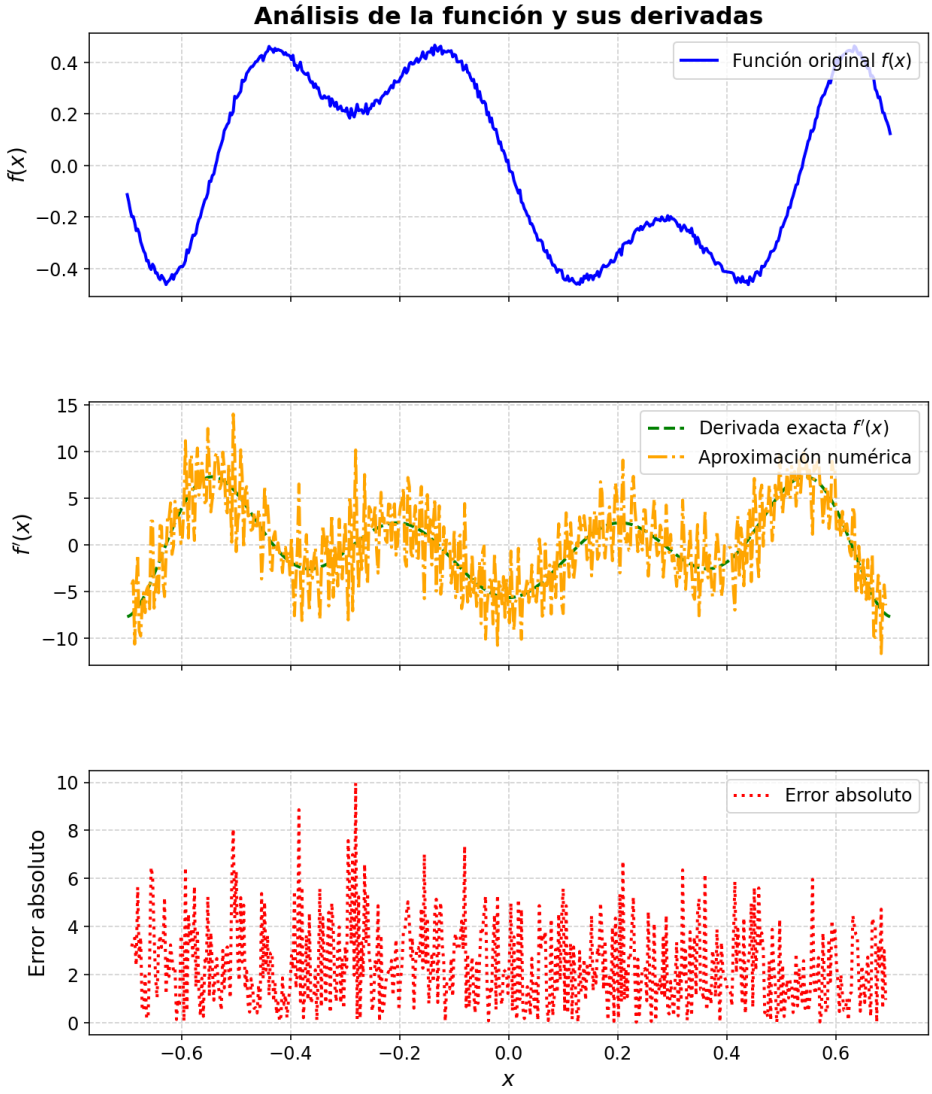

Figura 1.2: Función original, comparación de la derivada exacta, la estimación numérica y el error absoluto para las funciones de Legendre.

1.3. Conclusión

Se implementó un esquema numérico de diferenciación basado en diferencias finitas de orden $O(h^4)$ para estimar la primera derivada de $f(x)$ a partir de datos discretos. El método, restringido al intervalo desde x_3 hasta x_{n-4}, mostró alta precisión al compararse con derivadas analíticas. Las gráficas de prueba ilustraron una excelente concordancia entre las derivadas exactas y numéricas, con errores absolutos mínimos, validando la eficacia del esquema para aplicaciones en análisis numérico con datos equiespaciados.

Capítulo 2

Métodos de Diferenciación Numérica: Derivada Adelantada y Error Absoluto

Fecha de la actividad: 25 de noviembre de 2024

En este capítulo, se analiza el esquema de diferencias adelantadas de tres puntos para aproximar la derivada de una función $f(x)$.

2.1. Orden del Error de Truncamiento

El esquema de diferencias adelantadas de tres puntos es:

$$f'_{\text{num}}(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h}. \tag{2.1}$$

Para determinar el orden del error de truncamiento entre este esquema y $f'(x)$, vamos a expandir $f(x+h)$ y $f(x+2h)$ en series de Taylor alrededor de x hasta el quinto término.

Usamos las siguientes expansiones en series de Taylor:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f^{(3)}(x)}{6}h^3 + \frac{f^{(4)}(x)}{24}h^4 + O(h^5), \tag{2.2}$$

$$f(x+2h) = f(x) + 2f'(x)h + \frac{2^2 f''(x)}{2}h^2 + \frac{2^3 f^{(3)}(x)}{6}h^3 + \frac{2^4 f^{(4)}(x)}{24}h^4 + O(h^5). \tag{2.3}$$

Sustituyendo estas expansiones (2.2) y (2.3) en el esquema de la derivada numérica (2.1), obtenemos:

$$f'_{\text{num}}(x) = \frac{-3f(x) + 4\left(f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + O(h^3)\right) - \left(f(x) + 2f'(x)h + 2f''(x)h^2 + O(h^3)\right)}{2h}.$$

Simplificando los términos, finalmente obtenemos:

$$f'_{\text{num}}(x) = f'(x) + O(h^3). \tag{2.4}$$

Por lo tanto, el **orden del error de truncamiento** entre la derivada analítica $f'(x)$ y el esquema de derivada numérica (2.1) es de $O(h^3)$.

2.2. Estimación del Error Absoluto

El error absoluto E_{abs} entre la derivada exacta $f'(x)$ y la derivada numérica $f'_{\text{num}}(x)$ es dado por:

$$E_{\text{abs}} = |f'(x) - f'_{\text{num}}(x)| \sim O(h^3) + \epsilon_{\text{redondeo}}. \tag{2.5}$$

El error de redondeo $\epsilon_{\text{redondeo}}$ se refiere a la imprecisión que ocurre debido a las limitaciones de la representación numérica de los números en la computadora.

2.3. Conclusiones

El esquema de diferencias adelantadas de tres puntos proporciona una aproximación de orden $O(h^3)$ para la derivada de una función. Sin embargo, los errores de redondeo $\epsilon_{\text{redondeo}}$ deben ser considerados cuando se utiliza este método con **valores pequeños** de h.

Capítulo 3

Métodos Numéricos de Integración

Fecha de la actividad: 28 de noviembre de 2024

En este capitulo, queremos aproximar numéricamente la integral:

$$\int_{-1}^{1} f(x) \sin\left(\frac{\pi x}{2}\right) dx = w_{-1} f(-1) + w_0 f(0) + w_1 f(1). \tag{3.1}$$

Donde la regla es exacta para polinomios de hasta grado 2.

3.1. Determinación de los Pesos

Para encontrar los pesos w_i, evaluamos la regla para polinomios de grado 0, 1 y 2.

- **Polinomio de grado 0**: $f(x) = 1$

$$\int_{-1}^{1} \sin\left(\frac{\pi x}{2}\right) dx = w_{-1} + w_0 + w_1. \tag{3.2}$$

- **Polinomio de grado 1**: $f(x) = x$. La función $x \sin\left(\frac{\pi x}{2}\right)$ es impar, por lo que la integral es cero:

$$\int_{-1}^{1} x \sin\left(\frac{\pi x}{2}\right) dx = -w_{-1} + w_1 = 0 \implies w_{-1} = w_1. \tag{3.3}$$

- **Polinomio de grado 2**: $f(x) = x^2$. La regla se aplica como:

$$\int_{-1}^{1} x^2 \sin\left(\frac{\pi x}{2}\right) dx = w_{-1} + w_0 + w_1. \tag{3.4}$$

Al resolver el sistema de ecuaciones, obtenemos:

$$w_{-1} = -\frac{1}{2}, \quad w_0 = 1, \quad w_1 = -\frac{1}{2}. \tag{3.5}$$

3.2. Aproximación de la Integral Dada

La integral a calcular es:

$$\int_{-1}^{1} x^2 \sin^2\left(\frac{\pi x}{2}\right) dx. \tag{3.6}$$

Utilizando los pesos obtenidos, la aproximación es:

$$f(-1) = (-1)^2 = 1, \tag{3.7}$$
$$f(0) = 0^2 = 0, \tag{3.8}$$
$$f(1) = 1^2 = 1. \tag{3.9}$$

Entonces:

$$w_{-1}f(-1) + w_0 f(0) + w_1 f(1) = -\frac{1}{2}(1) + 1(0) - \frac{1}{2}(1) = -1. \tag{3.10}$$

El valor exacto de la integral es:

$$\int_{-1}^{1} x^2 \sin^2\left(\frac{\pi x}{2}\right) dx = \frac{1}{3} + \frac{2}{\pi^2} \approx 0.53598. \tag{3.11}$$

3.3. Análisis de Error

El error absoluto es:

$$|-1 - 0.53598| \approx 1.53598. \tag{3.12}$$

El error es significativo, ya que la regla es exacta solo para polinomios de grado 2. La función dada contiene términos no polinómicos ($\sin^2$), lo que introduce errores debido a la limitada precisión del método.

3.4. Conclusión

La regla de cuadratura utilizada es exacta para polinomios de grado 2, pero presenta un error significativo al aproximar funciones más complejas, como aquellas con términos trigonométricos. Para problemas de mayor complejidad, es necesario emplear métodos de integración más avanzados o de mayor orden.

Capítulo 4

Problema de caída libre

Fecha de la actividad: 26 de octubre de 2024

En este capítulo, desarrollaremos una simulación computacional de la caída libre de una pelota desde una altura inicial $y_0 = H$, partiendo del reposo ($v_0 = 0$), bajo la acción de la aceleración gravitatoria g. Ignoraremos la resistencia del aire y otras fuerzas externas para simplificar el problema. Utilizaremos las ecuaciones del Movimiento Rectilíneo Uniformemente Acelerado (MRUA):

- **Posición en función del tiempo**:

$$y(t) = H - \frac{1}{2} \cdot g \cdot t^2 \tag{4.1}$$

- **Velocidad en función del tiempo**:

$$v(t) = -g \cdot t \tag{4.2}$$

Donde $H > 0$ es la altura inicial, y $g = 9.81\,\mathrm{m/s^2}$ es la aceleración gravitatoria en la superficie terrestre. La simulación calculará las velocidades en intervalos regulares de tiempo Δt mediante un ciclo `for`, e imprimirá los resultados en pantalla. Adicionalmente, se generará un gráfico de altura vs tiempo que destacará con una flecha el instante de impacto con el suelo.

4.1. Deducción del instante de impacto

Cuando la pelota toca el suelo, $y(t_f) = 0$. Evaluando este instante en la ecuación (4.1), obtenemos:

$$0 = H - \frac{1}{2} \cdot g \cdot t_f^2 \tag{4.3}$$

Despejando para t_f:

$$t_f = \sqrt{\frac{2H}{g}} \tag{4.4}$$

4.2. Implementación de Python

Definimos los parámetros de simulación, estos son H en metros, g en metros sobre segundos cuadrados y Δt segundos. Suponiendo $H = 100$ m, $g = 9.81$ m/s ^{2} y $\Delta t = 0.01$ s. Así, tenemos en `Python`:

```
H = 100
Delta t = 0.01
g = 9.81
```

Creamos una matriz de tiempos desde $t = 0$ hasta t_f, con $t_f > 0$:

```
tiempo = np.arange(0, np.sqrt(2 * H / g), Dt)
```

Calculamos e imprimimos la velocidad en cada instante, empleando la ecuación (4.2):

```
for t in tiempo:
    velocidad = - g * t
    print(f"t = {t:.2f} s, v(t) = {velocidad:.2f} m/s")
```

4.3. Gráfico Altura vs Tiempo

Guardamos espacio en la memoria para las posiciones correspondientes a cada instante, y las calculamos en un ciclo `for`:

```
posiciones = np.zeros(len(tiempo))

for i in range(len(tiempo)):
    posiciones[i] = H - 0.5 * g * tiempo[i]**2
```

Generamos el gráfico con `matplotlib`:

```
plt.plot(tiempo, posiciones, label="Altura (m) vs Tiempo (s)")
```

Para el titulo del gráfico, queremos que indique el valor de H en el titulo, para ello usamos la siguiente linea de codigo:

```
plt.title(f"Caída libre de una pelota desde H = {H} m respecto del suelo")
```

Empleando la función `annotate` de matplotlib, se inserta una flecha roja en el instante de impacto, junto a una etiqueta que indica el "Momento de impacto". Esta es acción es llamada con el código siguiente:

```
plt.plot(tiempo, posiciones)
plt.annotate('Momento de impacto',
             xy=(tiempo[-1], 0),
             xytext=(tiempo[-1]-1, H*(2/5)),
             arrowprops=dict(facecolor='red', shrink=0.05))
```

Podemos observar el gráfico obtenido en la figura (4.1).

4.4. Conclusión

En esta actividad utilizamos las ecuaciones del Movimiento Rectilíneo Uniformemente Acelerado (MRUA) para modelar y simular la caída libre de una pelota mediante programación en `Python`. Se trabajó con matrices de tiempo, las cuales fueron generadas con la función `linspace` de `numpy`, donde el final de este array fue el tiempo de impacto $tf = \sqrt{\frac{2H}{g}}$. Se empleó un ciclo `for` para calcular las

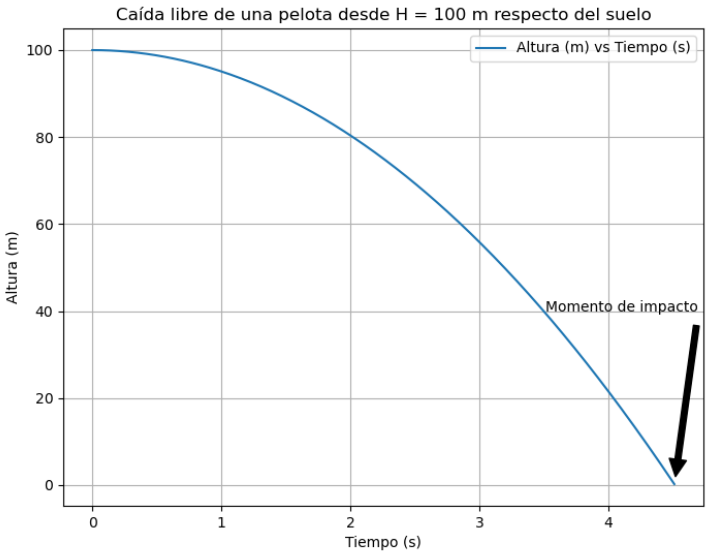

Figura 4.1: Gráfico Altura vs Tiempo, con flecha roja indicando el momento de impacto

velocidades y posiciones en intervalos regulares. Además, se exploró el uso de la librería `matplotlib` para visualizar los resultados a través de gráficos, esto en el uso de la función `annotate`, Matplotlib (2024).

Capítulo 5

Cálculo y análisis de los números de Catalán

Fecha de la actividad: 16 de noviembre de 2024

Este capítulo aborda las propiedades fundamentales de los números de Catalán, definidos como:

$$C_n = \frac{(2n)!}{(n+1)!n!}, \quad n \geq 0. \tag{5.1}$$

Se demuestra que esta definición puede expresarse recursivamente como:

$$C_0 = 1, \quad C_{n+1} = \frac{4n+2}{n+2} C_n. \tag{5.2}$$

Además, se utiliza `Python` para:

- Graficar todos los números de Catalán que satisfacen $C_n < M$, con $M > 10^{15}$, empleando precisión simple (32 bits).
- Comparar los resultados obtenidos mediante la definición exacta y la relación recursiva.
- Verificar gráficamente el comportamiento asintótico de la serie para $n \gg 1$.

Se incluye un análisis visual que destaca la relación entre los valores exactos y su aproximación, resaltando las propiedades de esta importante secuencia combinatoria.

5.1. Definición y forma de recurrencia

Se puede demostrar que la definición (5.1) es equivalente a la recurrencia:

$$C_0 = 1, \quad C_{n+1} = \frac{4n+2}{n+2} C_n \tag{5.3}$$

Demostración

1. Observemos que, según la definición (5.1), para $n = 0$ se tiene:

$$\begin{aligned} C_0 &= \frac{(2\cdot 0)!}{(0+1)!\cdot 0!} \\ &= \frac{(0)!}{1!\cdot 0!} \\ &= \frac{1}{1\cdot 1} \\ &= 1. \end{aligned}$$

Por lo tanto, queda demostrado que $C_0 = 1$.

2. La recurrencia $C_{n+1} = \frac{4n+2}{n+2} C_n$ se demuestra mediante el principio de inducción matemática. Procedemos de la siguiente manera:

   - **Hipótesis inductiva:** Suponemos que la fórmula es válida para un entero no negativo k, es decir, que $C_k = \frac{(2k)!}{(k+1)!\,k!}$.
   - **Tesis inductiva:** Queremos demostrar que $C_{k+1} = \frac{4k+2}{k+2} C_k$.

   A partir de la definición de los números de Catalán, evaluamos C_{k+1}:

$$C_{k+1} = \frac{(2(k+1))!}{((k+1)+1)!\,(k+1)!} \tag{5.4}$$

$$= \frac{(2k+2)!}{(k+2)!\,(k+1)!}, \tag{5.5}$$

   donde simplemente hemos expandido $2(k+1)$ como $2k+2$ y $(k+1)+1$ como $k+2$ en los factoriales. A continuación, separamos el factorial $(2k+2)!$ utilizando la propiedad de los factoriales $(n+1)! = (n+1)\cdot n!$:

$$C_{k+1} = \frac{(2k)!\,(2k+1)\,(2k+2)}{(k+1)!\,(k+2)!} \tag{5.6}$$

$$= \frac{(2k)!\,(2k+1)\,(2k+2)}{(k+1)!\,(k+2)\,k!\,(k+1)}, \tag{5.7}$$

   donde el término $(k+2)!$ fue descompuesto como $(k+2)\cdot(k+1)!$. Luego, reorganizamos los términos agrupando los factores comunes:

$$C_{k+1} = \frac{(2k)!\,(2k+1)\,2\,(k+1)}{(k+1)!\,(k+2)\,k!\,(k+1)}. \tag{5.8}$$

   Aquí, se utilizó $2k+2 = 2\cdot(k+1)$ para simplificar el último factor. Al cancelar $(k+1)$ en el numerador y denominador, el resultado queda como:

$$C_{k+1} = \frac{(2k)!\,(4k+2)}{(k+1)!\,(k+2)\,k!}. \tag{5.9}$$

   Luego, identificamos que el primer término $\frac{(2k)!}{(k+1)!\,k!}$ corresponde a C_k, lo que permite expresar la ecuación como:

$$C_{k+1} = \frac{(2k)!}{(k+1)!\,k!} \cdot \frac{4k+2}{k+2} \tag{5.10}$$

$$= C_k \cdot \frac{4k+2}{k+2}. \tag{5.11}$$

Por lo tanto, hemos demostrado que:

$$C_{n+1} = \frac{4n+2}{n+2} C_n, \quad \forall n \in \mathbb{N}_0. \tag{5.12}$$

De esta forma, se valida que los números de Catalán pueden expresarse de manera equivalente mediante la relación de recurrencia. Cabe notar que, al evaluar $n = 0$, se obtiene directamente que $C_0 = 1$, lo cual cierra la demostración de la base inductiva.

5.2. Gráfica y comparación

Para calcular los números de Catalán y graficarlos utilizaremos `Python`, empleando las bibliotecas `numpy` y `matplotlib.pyplot`, para calcular hasta $C_n < M$, con una cota M dada por el usuario.

Primero, definimos una función para calcular el operador factorial.

```
def fact(n):
    fact=1
    for n in list(range(0,int(n+1),1)):
        if n==0 or n==1:
            fact*=1
        elif n>1:
            fact*= n
    return fact
```

Definimos un par de funciones para calcular el n-esimo numero de Catalán, asegurándonos de usar precisión **simple** con la función `float32` de `numpy`, por:

- **Definición** (5.1):

```
def catalan_def(n):
    catalan_n = np.float32 ( fact(2*n)/( fact(n+1)* fact(n) ) )
    return catalan_n
```

- **Recursividad** (5.12):

```
def catalan_rec(n):
    catalan_0 = 1
    for n in list( range( 0, n, 1) ):
        catalan_rec = np.float32( catalan_0 * ( 4*n + 2 ) / ( n + 2 ) )
        catalan_0 = catalan_rec
    return catalan_0
```

Empleando la función `input` se programa `M` para que sea una cota ingresada por el usuario. La función `float` interpreta el valor ingresado por el usuario como un float, si y solo si la sintaxis es correcta.

```
M = float(input("Ingrese el valor de M: "))
```

Establecemos las siguientes listas vacías para almacenar los resultados:

- `n_values`: contendrá los números naturales
- `list_catalan_def`: almacenará los valores calculados por definición.
- `list_catalan_rec`: guardará los valores obtenidos mediante recurrencia.

El código es el siguiente:

```
n_values = []
list_catalan_def = []
list_catalan_rec = []
```

Se define una variable `n = 0` para iniciar un ciclo `while`, donde se calculan los números de Catalán C_n mediante la definición (5.1) y la recurrencia (5.12). En cada iteración, los valores calculados y el índice n se almacenan en listas respectivas. El ciclo finaliza cuando $M \geq C_n$.

```
# Cálculo iterativo de los números de Catalán
n = 0
while True:
    C_def = catalan_def(n)  # Número de Catalán por definición
    C_rec = catalan_rec(n)  # Número de Catalán por recurrencia

    # Condición para detener el ciclo si se supera la cota M
    if C_def >= M or C_rec >= M:
        break

    # Almacenamos los valores calculados y el índice actual
    list_catalan_def.append(C_def)
    list_catalan_rec.append(C_rec)
    n_values.append(n)

    # Incrementamos el índice
    n += 1
```

Para visualizar los números de Catalán que cumplen $M \geq C_n$, se generó un gráfico empleando `matplotlib.pyplot`. En este se comparan los resultados obtenidos mediante las expresiones (5.1) y (5.12). Además, se incluye una línea horizontal roja que representa la cota M. El eje x corresponde a los primeros $n \in \mathbb{N}_0$, mientras que el eje y muestra los valores de C_n calculados bajo dicha restricción.

```

# Gráfico de los números de Catalán calculados por ambos métodos
plt.plot(n_values, list_catalan_def, 'x', label='Por definición (1)', color='blue')
plt.plot(n_values, list_catalan_rec, '.', label='Por recurrencia (2)', color='orange')

# Línea horizontal que representa la cota M
plt.axhline(y=M, color='red', linestyle='--', label=f'Cota M = {M}')

```

Para el valor $M = 10^{16}$, podemos observar el gráfico obtenido desde el codigo que hemos programado en la figura (5.1).

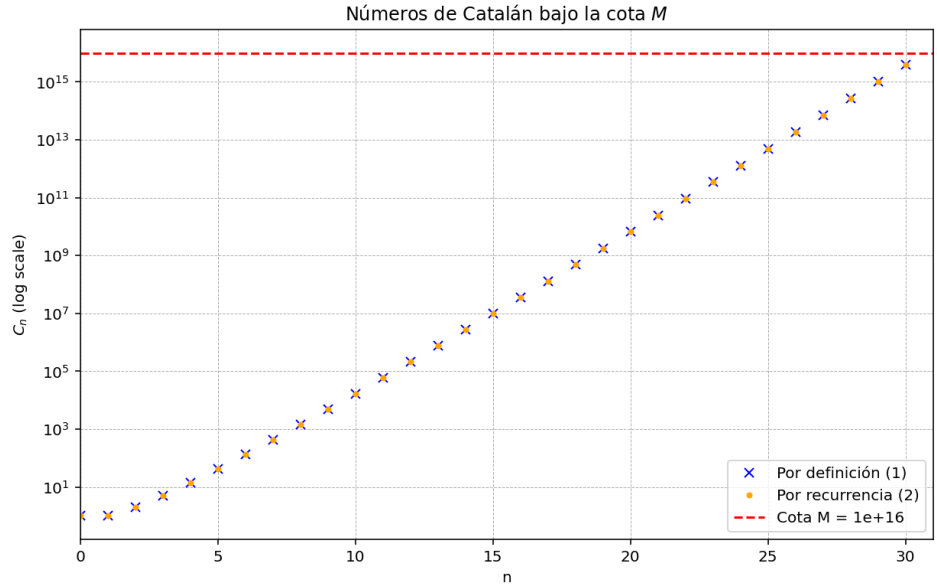

Figura 5.1: Gráfico Números de Catalán bajo la cota $M = 10^{16}$

5.3. Comportamiento asintótico

Para $n \gg 1$, los números de Catalán tienen un comportamiento asintótico de la forma:

$$C_n \approx \frac{4^n}{n^{3/2}\sqrt{\pi}} \tag{5.13}$$

A continuación, comprobaremos gráficamente este comportamiento asintótico. Para ello, implementando `Python`, definimos una función que calcula la aproximación al n-ésimo número de Catalán según la ecuación (5.13). Se utiliza precisión **simple** mediante la función `float32` del paquete `numpy`.

```
def catalan_def(n):
    catalan_n = np.float32 ( fact(2*n)/( fact(n+1)* fact(n) ) )
    return catalan_n
```

Se define un rango de valores enteros, que representa los primeros 99 números naturales, y se almacena en la variable `n_values`. Estos valores se manejan utilizando precisión **simple** (`float32`) .

```
n_values = np.arange(1, 100, dtype=np.float32)
```

Los números de Catalán (`Cn`) y su aproximación asintótica (`Cn_aprox`) se calculan para cada valor en `n_values` utilizando comprensiones de listas. Los resultados se almacenan como arreglos de `NumPy`.

```
Cn = np.array([catalan_def(i) for i in n_values])
Cn_aprox = np.array([catalan_aprox(i) for i in n_values])
```

Es importante notar que, al usar precisión simple en las funciones `catalan_def` y `catalan_aprox`, se observan las siguientes limitaciones:

- Con `catalan_def`, es posible calcular únicamente los primeros $n = 68$ números de Catalán de forma explícita (excluyendo $n = 0$).

- Con `catalan_aprox`, se pueden determinar los primeros $n = 62$ números de Catalán mediante la aproximación, excluyendo $n = 0$.

Los valores restantes en los arreglos `Cn` y `Cn_aprox` serán representados como `inf`, lo que en `Python` indica infinito GeeksforGeeks (2024).

Al generar un gráfico para comparar las curvas de los valores C_n calculados por definición (5) y por la aproximación asintótica (5.13), se obtiene la siguiente representación gráfica:

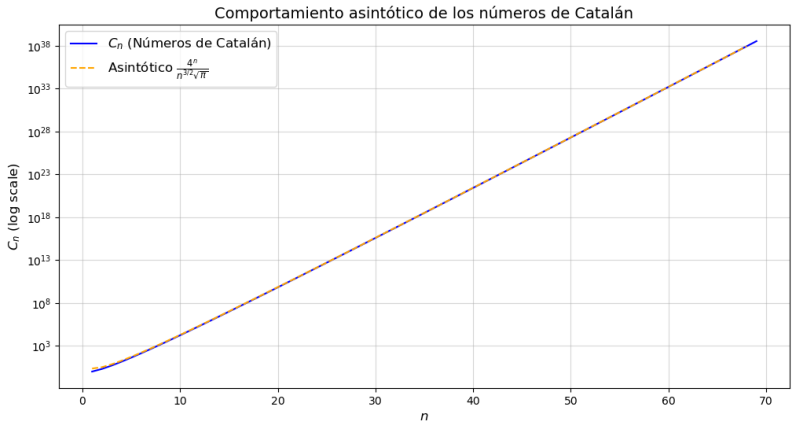

Figura 5.2: Comparación gráfica entre C_n y su aproximación asintótica.

Si se realiza un acercamiento ("zoom") entre las curvas, se puede observar con mayor claridad el comportamiento asintótico de la aproximación. Se aprecia esto en la figura 5.3.

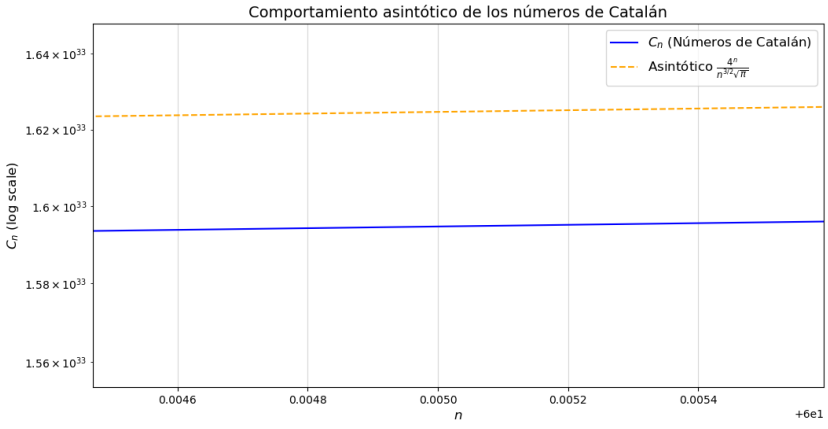

Figura 5.3: Detalle gráfico del comportamiento asintótico entre C_n y su aproximación.

5.4. Conclusión

El estudio de los números de Catalán permitió validar su definición y recurrencia, explorar su comportamiento asintótico y analizar gráficamente sus propiedades, destacando su relevancia como una secuencia combinatoria fundamental mediante métodos numéricos y computacionales.

Capítulo 6

Potencial electrostático de dos cargas puntuales sobre el Plano

Fecha de la actividad: 29 de octubre de 2024

En este capítulo, utilizando las librerias `Numpy` y `Matplotlib` de `Python`, graficaremos el **potencial electrostático** generado por dos cargas puntuales q_1 y q_2 separadas por una distancia fija a en el plano (x, y). Utilizaremos la ecuación del potencial debido a una carga puntual en un punto del plano:

$$V = k\frac{q}{r} \tag{6.1}$$

Donde $k = \frac{1}{4\pi\epsilon_0} = 8.99 \times 10^9\,\mathrm{N{\cdot}m^2/C^2}$ es la constante de Coulomb, q es la carga que genera el campo eléctrico, r es la distancia entre la carga y el punto en consideración. Fernández (2024)

Tomaremos para las cargas q_1 y q_2 las siguientes consideraciones:

- Ambas cargas son positivas.
- Ambas cargas son negativas.
- Una carga es positiva y la otra negativa.

6.1. Planteamiento de las cargas en el Plano

Para simplificar los cálculos que realizará el computador en la simulación, ubicaremos las cargas q_1 y q_2 sobre uno de los ejes del plano (x, y), en este caso el eje de las abscisas. De este modo, ambas cargas estarán situadas a una distancia equidistante del origen, y la distancia de separación entre ellas será a, donde $a > 0$. Así, q_1 se posicionará en el punto $\left(-\frac{a}{2}, 0\right)$ y q_2 en $\left(\frac{a}{2}, 0\right)$.

6.2. Cálculo del Potencial Electrostático en el Plano

En primer lugar, definimos los parámetros fundamentales para la simulación. La separación entre las cargas es $a = 2\,\mathrm{m}$, mientras que el valor absoluto de la magnitud de cada carga es $|q_1| = |q_2| = 1 \times 10^{-9}\,\mathrm{C}$. Además, consideramos la constante de Coulomb $k = 8.99 \times 10^9\,\mathrm{N{\cdot}m^2/C^2}$. Estos parámetros se implementan en `Python` de la siguiente manera:

```
a = 2
q1 = 1e-9
q2 = 1e-9
k = 8.99e9
```

Las posiciones de las cargas puntuales q_1 y q_2 se definen como los pares ordenados (x_0, y_0), situados simétricamente respecto al origen. Estas posiciones se asignan en `Python` de la siguiente manera:

```
pos_q1 = (-a/2, 0)
pos_q2 = (a/2, 0)
```

A continuación, definimos una función para calcular el potencial electrostático generado por una carga puntual q_0 en una posición dada pos $= (x_0, y_0)$. El potencial en un punto (x, y) se calcula según la fórmula:

$$V = \frac{k \cdot q_0}{r}$$

donde r es la distancia entre la carga y el punto de interés. La implementación en `Python` es la siguiente:

```
def potencial(x, y, q, pos):
    r = np.sqrt((x - pos[0])**2 + (y - pos[1])**2)
    return k * q / r
```

Los puntos (x, y) donde se calculará el potencial pertenecen al intervalo $I = [x_i, x_f] \times [y_i, y_f]$. Para simular esta región, utilizamos la función `linspace` para generar dos conjuntos de 400 puntos equidistantes entre -5 y 5 en ambos ejes:

```
x = np.linspace(-5, 5, 400)
y = np.linspace(-5, 5, 400)
```

Luego, empleamos la función `meshgrid` (MathWorks (2024)) para generar una cuadrícula que representa el subconjunto I del plano (x, y), permitiendo calcular el potencial en cada punto:

```
X, Y = np.meshgrid(x, y)
```

Con las posiciones de las cargas definidas y la cuadrícula generada, calculamos el potencial electrostático debido a cada carga q_1 y q_2 por separado. Posteriormente, aplicamos el principio de superposición para obtener el potencial total en cada punto del plano I. Para ello, utilizamos la función `potencial` definida anteriormente, y creamos la siguiente función para sumar los potenciales:

```
def calcular_potencial_total(q1, q2):
    V1 = potencial(X, Y, q1, pos_q1)
    V2 = potencial(X, Y, q2, pos_q2)
    return V1 + V2
```

6.3. Gráficos del Potencial

A continuación, mostramos los gráficos del potencial para los tres casos descritos:

- Ambas cargas son positivas.

```
V = calcular_potencial_total(q1, q2)
axes[0].contourf(X, Y, V, levels=50, cmap="coolwarm")
```

- Ambas cargas son negativas.

```
V = calcular_potencial_total(-q1, -q2)
axes[1].contourf(X, Y, V, levels=50, cmap="coolwarm")
```

- Una carga es positiva y la otra negativa.

```
V = calcular_potencial_total(q1, -q2)
axes[2].contourf(X, Y, V, levels=50, cmap="coolwarm")
```

6.4. Interpretación de los Resultados

En la figura (6.1), se presenta el gráfico del potencial electrostático en los tres casos estudiados. A partir de estos resultados, se identifican las siguientes configuraciones relevantes:

- **Ambas cargas son positivas**: Esta configuración genera un campo repulsivo, donde el potencial es positivo en toda la región circundante.
- **Ambas cargas son negativas**: El campo resultante es similar al del caso anterior, pero el potencial es negativo en todo el espacio considerado.
- **Una carga positiva y una negativa**: En esta situación se forma un dipolo eléctrico. El potencial cambia de signo en diferentes regiones del espacio, reflejando la naturaleza atractiva entre las dos cargas opuestas.

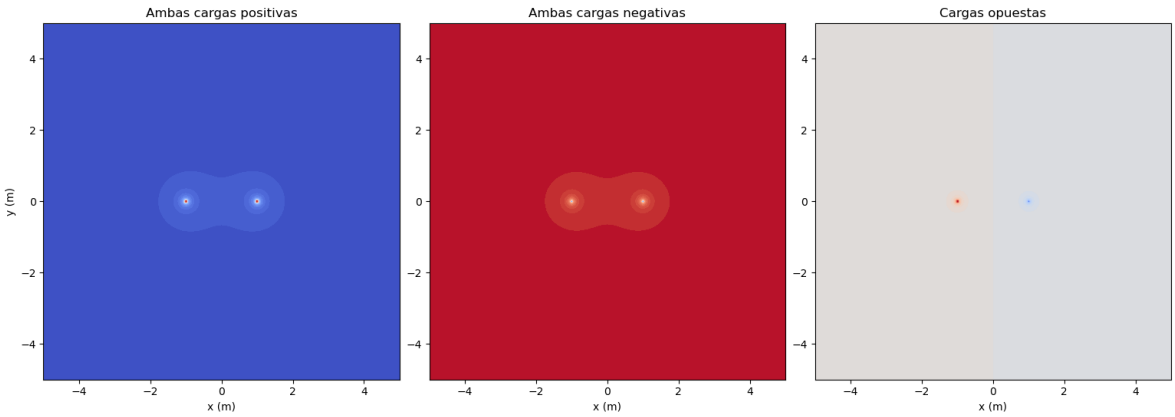

Figura 6.1: Distribución del potencial electrostático para las tres configuraciones de cargas: ambas positivas, ambas negativas y configuración dipolar.

6.5. Conclusión

En esta actividad se realizó la simulación y visualización del potencial electrostático generado por dos cargas puntuales para diferentes combinaciones de signos. La ecuación utilizada para calcular el

potencial fue la expresión clásica del potencial de Coulomb, y la visualización gráfica se llevó a cabo mediante la librería `matplotlib` en `Python`.

Los resultados obtenidos permiten analizar cómo la distribución espacial del potencial varía de forma significativa dependiendo del signo de las cargas. En particular, se observó una simetría característica cuando ambas cargas tienen el mismo signo, y una mayor complejidad en la configuración dipolar, donde el potencial cambia de signo en distintas regiones del espacio.

Capítulo 7

Órbita de un satélite alrededor de la Tierra

Fecha de la actividad: 28 de octubre de 2024

En este capítulo, estudiaremos la órbita circular de un satélite lanzado alrededor de la Tierra. El objetivo es determinar la altura del satélite en función de su periodo T. Se demostrará la fórmula para la altura h sobre la superficie terrestre y, además, se generará una tabla y un gráfico utilizando `Python`. Por último, se comparará la altura de un satélite geosincrónico con periodos de 24 horas y un día sideral de 23.93 horas.

7.1. Demostración de la fórmula para la altura h

Se tiene un satélite de masa m orbitando circularmente sobre la Tierra de masa M a una altura h respecto la superficie de esta, la distancia de separación entre los centros de masa de los cuerpos es $R + h$, donde R el radio de la Tierra. Por la Ley de gravitación universal, se deduce la fuerza gravitatoria que ejerce la Tierra sobre el satélite:

$$F_g = G\frac{mM}{(R+h)^2} \tag{7.1}$$

Sabemos por la segunda ley de Newton que $F = ma$. En este caso, se tiene que $F_g = a_g m$, luego tenemos en (7.1) que:

$$a_g m = G\frac{mM}{(R+h)^2}$$

Simplificamos la ecuación dividiendo por la masa m del satélite, obtenemos:

$$a_g = G\frac{M}{(R+h)^2} \tag{7.2}$$

Dado que estamos considerando el caso de un satélite en órbita circular alrededor de un objeto masivo, en este caso la Tierra, podemos abordar la situación utilizando los principios del Movimiento Circular Uniforme (MCU). Luego, tendremos que a_g sera la aceleración centripeta del satelite, la cual satisface la ecuación:

$$a_g = \omega^2 r$$

donde ω es la velocidad angular del satélite y r es un radio, que en este caso se interpreta como el centro de masa de la tierra el orgen de la circunferencia por la que orbita el satelite, por lo tanto

$r := R + h$. Por otra parte, sabemos que la velocidad angular ω se relaciona con el periodo de la forma:

$$\omega = \frac{2\pi}{T}$$

Así, la aceleración a_g del satélite quedara dada por:

$$a_g = \left(\frac{2\pi}{T}\right)^2 (R + h) \tag{7.3}$$

Reemplazando (7.3) en (7.2), se obtiene:

$$\left(\frac{2\pi}{T}\right)^2 (R + h) = \frac{GM}{(R + h)^2} \tag{7.4}$$

Multiplicamos la ecuación (7.4) por $\left(\frac{T}{2\pi}\right)^2$, se obtiene:

$$(R + h) = \frac{GM}{(R + h)^2} \left(\frac{T}{2\pi}\right)^2 \tag{7.5}$$

Multiplicamos la ecuación (7.5) por $(R + h)^2$, se obtiene:

$$(R + h)^3 = \frac{GMT^2}{4\pi^2} \tag{7.6}$$

Elevamos la ecuación (7.6) por $\frac{1}{3}$, obtenemos:

$$R + h = \left(\frac{GMT^2}{4\pi^2}\right)^{1/3} \tag{7.7}$$

Restamos R en la ecuación (7.7). Luego, queda demostrado que:

$$h = \left(\frac{GMT^2}{4\pi^2}\right)^{1/3} - R \tag{7.8}$$

donde se tendran las constantes:

- $G = 6.67430 \times 10^{-11}\,\mathrm{m^3\,kg^{-1}\,s^{-2}}$ es la constante gravitacional,
- $M = 5.97 \times 10^{24}\,\mathrm{kg}$ es la masa de la Tierra,
- $R = 6371\,\mathrm{km}$ es el radio de la Tierra.

7.2. Implementación en Python: Tabla de Periodo vs Altura

A continuación, generaremos una tabla que muestra la altura h del satélite para diferentes valores del periodo T, variando en múltiplos de 15 minutos. El siguiente código en `Python` utiliza las bibliotecas `numpy` para manejar arreglos y `Jinja2` para generar automáticamente un archivo en LaTeX.

Definimos tres variables para las contantes presentes en la ecuación (7.8), donde `G` es la constante gravitacional ($\mathrm{m^3kg^{-1}}s^{-2}$), `M` la masa de la tierra (kg) y `R` el radio de la Tierra (m).

```
G = 6.67430e-11
M = 5.97e24
R = 6371e3
```

Definimos una función `height_h` para calcular la altura del satélite en función del periodo orbital, según la ecuación (7.8). Esta función toma como entrada el periodo T (en segundos) y devuelve la altura h (en metros) restando el radio de la Tierra al radio orbital total calculado:

```
def height_h(T):
    return (G * M * T**2 / (4 * np.pi**2))**(1/3) - R
```

Generamos una lista de 24 valores para el periodo (T), en múltiplos de 15 minutos (en segundos) hasta 6 horas, usando la función `arange` de `numpy`.

```
T_values = np.arange(15 * 60, 6 * 3600 + 1, 15 * 60)
```

De manera análoga, generamos una lista que contiene los valores de la altura del satélite (h) para cada uno de los periodos (T). Para ello, empleamos una comprensión de listas en Python, iterando con un ciclo `for` sobre los elementos de `T_values` y aplicando la función `height_h` a cada uno:

```
h_values = [height_h(T) for T in T_values]
```

Se preparan los datos para LaTeX combinando `T_values` y `h_values` con `zip`, y generando una lista de diccionarios con las claves `"T"` y `"h"`.data.

```
data = [{"T": T, "h": h} for T, h in zip(T_values, h_values)]
```

7.3. Gráfico de Altura vs Periodo

El siguiente gráfico muestra la altura del satélite (h) en función del periodo (T) utilizando la biblioteca `matplotlib`.

```
plt.plot(T_values, h_values, label="Altitud $h(T)$")
plt.xlabel("Periodo $T$ (s)")
plt.ylabel("Altitud $h$ (m)")
plt.title("Altitud $h$ en función del Periodo $T$")
```

Podemos guardar este gráfico en formato `pdf` con la linea siguiente:

```
plt.savefig("grafico_altitud.pdf")
```

7.4. Automatización de Tablas y Gráficos en LaTeX Mediante `python` y `jinja2`

Empleando la librería `Jinja2`, generaremos un archivo LaTeX dinámico que incluya una tabla con pares ordenados de los períodos (T) en segundos y alturas (h) en metros calculados en la sección 7.2, ademas se incluye el gráfico de Altura vs Periodo obtenido en la seccion anterior. A continuación, explicamos los pasos principales de la implementación.

Primero, definimos una plantilla en `Jinja2` almacenada en la variable `Texto`. Esta plantilla contiene la estructura básica de un documento en LaTeX, con secciones y una tabla que se llena dinámicamente utilizando bucles de `Jinja2`:

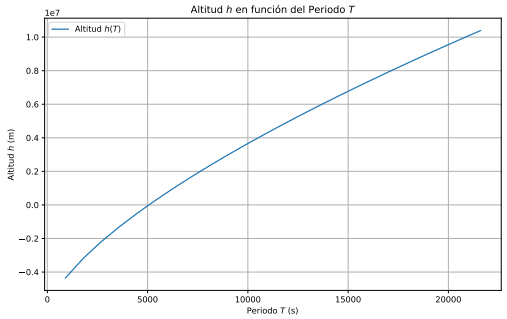

Figura 7.1: Gráfico de Altura vs Periodo

```
Texto = r"""
\documentclass{article}
\usepackage{graphicx}
\usepackage{booktabs}
\begin{document}

\section*{Tabla de Altitudes para Diferentes Períodos}

\begin{table}[h!]
    \centering
        \begin{tabular}{@{}cc@{}}
            \toprule
                \textbf{Periodo $T$ (s)} & \textbf{Altitud $h$ (m)} \\ \midrule
                    {% for row in data %}
                    {{ row.T }} & {{ row.h | round(2)}} \\
                    {% endfor %}
            \bottomrule
    \end{tabular}
\end{table}

\section*{Gráfico de Altitud $h$ en Función de $T$}

\includegraphics[width=\textwidth]{grafico_altitud.pdf}

\end{document}
"""
```

En la plantilla anterior:

- El bloque `% for row in data %` itera sobre los datos provistos, generando una fila en la tabla por cada entrada.
- `row.T` y `row.h` son valores dinámicos insertados en la tabla, con `row.h` redondeado a dos decimales mediante el filtro `round(2)`.
- Se incluye una sección donde se espera un gráfico `grafico_altitud.pdf`.

Posteriormente, utilizamos `Jinja2` para renderizar la plantilla, reemplazando los marcadores con los datos proporcionados:

```
latex_template = jinja2.Template(Texto)
latex_output = latex_template.render(data=data)
```

En el código anterior:

- `latex_template` es una instancia de la plantilla `Jinja2`.
- `latex_output` contiene el código LaTeX generado tras reemplazar los marcadores de la plantilla con los valores de `data`.

Finalmente, guardamos el documento generado en un archivo `.tex`:

```
with open("tabla_altitudes.tex", "w") as f:
    f.write(latex_output)
```

Este último bloque abre un archivo llamado `tabla_altitudes.tex` en modo escritura y guarda en él el código LaTeX generado. El archivo resultante está listo para ser compilado con herramientas como `pdflatex`.

7.5. Comparación entre día solar y día sideral

Un día sideral tiene $T_{\text{sideral}} = 23.93\,\text{horas} = 86148\,\text{segundos}$, mientras que un día solar tiene $T_{\text{solar}} = 24\,\text{horas} = 86400\,\text{segundos}$. Calculamos ambas alturas :

$$h_{\text{sideral}} = \left(\frac{GMT_{\text{sideral}}^2}{4\pi^2}\right)^{1/3} - R \tag{7.9}$$

$$h_{\text{solar}} = \left(\frac{GMT_{\text{solar}}^2}{4\pi^2}\right)^{1/3} - R \tag{7.10}$$

La diferencia estara dada por:

$$\Delta h = |h_{\text{solar}} - h_{\text{sideral}}| \tag{7.11}$$

Llevando este calculo a `python`, tenemos:

```
# Definimos los periodos sideral y solar:
T_sideral = 86148  # segundos
T_solar = 86400  # segundos

# Calculamos la altura sideral y solar:
h_sideral = height_h(T_sideral)
h_solar = height_h(T_solar)

# Calculamos la diferencia:
diferencia = abs(h_solar - h_sideral)
print(f"Diferencia de altitud: {diferencia:.2f} metros")
```

El efectuar dicho calculo, se obtiene $\Delta h = 82165.50\,\text{m}$. Este resultado muestra que la diferencia en altitud entre ambas definiciones es del orden de decenas de kilómetros, por lo que aunque $\Delta T := |T_{\text{solar}} - T_{\text{sideral}}| < 0$, se observa que Δh es grande.

7.6. Conclusión

En esta actividad, derivamos la fórmula para calcular la altura de un satélite en órbita circular en función de su período. Utilizamos `Python` y la biblioteca `jinja2` para generar dinámicamente un documento en LaTeX que incluye una tabla y un gráfico que relacionan la altura con el período. Además, comparamos las altitudes correspondientes a un día solar y un día sideral, encontrando que la diferencia es significativa, del orden de decenas de kilómetros. Este resultado subraya la importancia de considerar esta discrepancia en aplicaciones donde la precisión es crucial, como en la navegación satelital.

Capítulo 8

Demostración de una Aproximación Numérica de *f'(x)* usando Series de Taylor

Fecha de la actividad: 4 de noviembre de 2024

En este capítulo, utilizaremos series de Taylor para derivar una aproximación numérica de segundo orden, $O(h^2)$, para la primera derivada de una función $f : \mathbb{R} \to \mathbb{R}$ que sea suficientemente suave alrededor de un punto x. Esta aproximación se expresará en términos de los valores $f(x)$, $f(x+h)$ y $f(x+2h)$.

8.1. Análisis

Conocemos una aproximación numérica de segundo orden para la primera derivada Wikipedia (2024) en términos de $f(x)$, $f(x+h)$ y $f(x+2h)$:

$$f'(x) \approx \frac{4f(x+h) - f(x+2h) - 3f(x)}{2h} \,. \tag{8.1}$$

Para demostrar la aproximación numérica de $f(x)$ en la ecuación (8.1), utilizamos las expansiones de Taylor de $f(x+h)$ y $f(x+2h)$ alrededor de x. Sea $h \in \mathbb{R} : 0 < h < 1$.

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \ldots \tag{8.2}$$

$$f(x+2h) = f(x) + 2hf'(x) + \frac{(2h)^2}{2}f''(x) + \frac{(2h)^3}{6}f'''(x) + \ldots \tag{8.3}$$

Multiplicamos la ecuación (8.2) por 4 y la ecuación (8.3) por -1:

$$4f(x+h) = 4f(x) + 4hf'(x) + 4\frac{h^2}{2}f''(x) + 4\frac{h^3}{6}f'''(x) + \ldots \tag{8.4}$$

$$-f(x+2h) = -f(x) - 2hf'(x) - \frac{(2h)^2}{2}f''(x) - \frac{(2h)^3}{6}f'''(x) - \ldots \tag{8.5}$$

Sumamos las ecuaciones (8.4) y (8.5) miembro a miembro y desarrollando, obtenemos:

$$4f(x+h) - f(x+2h) = 3f(x) + 2hf'(x) - \frac{4h^3}{6}f'''(x) + \ldots$$

Despejamos $f'(x)$:

$$f'(x) = \frac{4f(x+h) - f(x+2h) - 3f(x)}{2h} + \frac{2h^2}{6}f'''(x) + \dots$$

Sea $\xi \in \mathbb{R} : x - h < \xi < x + h$. Se tiene:

$$f'(x) \approx \frac{4f(x+h) - f(x+2h) - 3f(x)}{2h} + \frac{2h^2}{6}f'''(\xi) \tag{8.6}$$

donde el término adicional $\frac{2h^2}{6}f'''(\xi)$ representa el error de truncamiento, de orden $O(h^2)$.

Así, observamos que la expresión (8.6) es una aproximación de segundo orden para $f'(x)$. Finalmente, omitiendo el error de truncamiento, obtenemos la diferencia aproximada:

$$f'(x) \approx \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h}.$$

8.2. Conclusión

En este capitulo, aprendí a demostrar una aproximación numérica de segundo orden para $f'(x)$ mediante series de Taylor, tomando los valores $f(x)$, $f(x+h)$ y $f(x+2h)$. Este proceso destacó la utilidad de las series de Taylor para mejorar la precisión en aproximaciones numéricas.

Capítulo 9

Derivación Numérica de Funciones Usando Diferencias Finitas

Fecha de la actividad: 21 de noviembre de 2024

En este capítulo, implementaremos algoritmos en `Python` para calcular la primera derivada numérica de una función $f(x)$ utilizando los esquemas de diferencias **adelantadas**, **centradas** y **retrasadas**. Las funciones evaluadas incluyen $f(x) = \sin(x)$, $f(x) = e^x$, $f(x) = \ln(x)$ y $f(x) = x^2$. Posteriormente, analizaremos el desempeño de estos métodos mediante **gráficos** que comparan la derivada numérica con la derivada analítica, además de gráficos de error relativo, para evaluar la precisión de cada esquema.

9.1. Implementación de Python

Utilizando en `python` las bibliotecas `numpy` y `matplotlib.pyplot`. He desarrollado la función `Esquemas_de_diferencias`, que calcula y gráfica la derivada analítica y las derivadas numéricas mediante esquemas adelantado, retrasado y centrado. Los parámetros son:

- `f` (function): Función a derivar numéricamente.
- `df` (function): Derivada analítica de la función f.
- `x_vals` (array): Valores de x para evaluar las derivadas.
- `h` (float): Tamaño del paso para las diferencias finitas.

Las derivadas se calculan con las fórmulas:

- **Diferencia adelantada:** $f'(x) \approx \frac{f(x+h)-f(x)}{h}$.
- **Diferencia retrasada:** $f'(x) \approx \frac{f(x)-f(x-h)}{h}$.
- **Diferencia centrada:** $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$.

Definimos un rango de valores de x y un paso h, de tal modo que no se tengan problemas al momento de hacer cálculos con la función $\ln(x)$, la cual esta definida para valores reales $x > 0$:

```
x_values = np.linspace(0.5, 12, 200)
h = 0.1
```

El cálculo de las derivadas numéricas y analítica itera sobre `x_values`, almacenando los resultados en un `numpy.array`. Finalmente, el programa genera y retorna un gráfico que muestra la comparación entre la derivada analítica y los tres esquemas de derivadas numéricas aplicados a la función ingresada. A continuación se muestra el algoritmo:

```
1
2  def Esquemas_de_diferencias(f,df, x_vals=x_values, h=h):
3
4      df_analitica = np.array([df(x) for x in x_vals])
5
6      df_adelantada = np.array([(f(x + h) - f(x)) / h for x in x_vals])
7
8      df_retrasada = np.array([(f(x) - f(x - h)) / h for x in x_vals])
9
10     df_centrada = np.array([(f(x + h) - f(x - h)) / (2 * h) for x in x_vals])
11
12     plt.figure(figsize=(10, 6))
13
14     plt.plot(x_vals, df_analitica, color="orange", linewidth=2.5, linestyle="--",
       ↪  label="Derivada analítica")
15     plt.plot(x_vals, df_adelantada, color="green", label="Derivada adelantada")
16     plt.plot(x_vals, df_retrasada, color="red", label="Derivada retrasada")
17     plt.plot(x_vals, df_centrada, color="blue", label="Derivada centrada")
18
19     plt.legend()
20     plt.title(f"Comparación entre las Derivadas numéricas y analitica para la función
       ↪  {f.nombre}")
21     plt.xlabel("x")
22     plt.ylabel(f"{f.nombre}")
23     plt.grid(True)
24
25     plt.tight_layout()
26     return plt.show()
27
```

Nota: Aunque el enunciado del problema no requiere explícitamente que el algoritmo calcule la derivada analítica para un conjunto de valores finitos, decidí implementarlo de esa manera para que pudiera resolver automáticamente las dos primeras partes del problema.

Definimos en el código las funciones y sus derivadas analíticas que queremos utilizar para probar el algoritmo programado. Además, añadimos las variables `f.nombre`, las cuales permiten incluir en el gráfico el nombre de la función ingresada al algoritmo. De este modo, tenemos:

- $f(x) = \sin(x)$ y $f'(x) = \cos(x)$,

```
1 def f(x):   return np.sin(x)
2 f.nombre = r'$\sin(x)$'
3 def df(x):  return np.cos(x)
```

- $f(x) = e^x$ y $f'(x) = e^x$,

```
1 def g(x):   return np.exp(x)
2 g.nombre = r'$e^x$'
3 def dg(x):  return np.exp(x)
```

- $f(x) = \ln(x)$ y $f'(x) = \frac{1}{x}, \quad x > 0,$

```
def ln(x):  return np.log(x)
ln.nombre = r'$\ln(x)$'
def dln(x): return 1/x
```

- $f(x) = x^2$ y $f'(x) = 2x$

```
def g(x):   return np.exp(x)
g.nombre = r'$e^x$'
def dg(x):  return np.exp(x)
```

Finalmente, llamamos al algoritmo que hemos programado, `Esquemas_de_diferencias`, con cada función y su derivada analítica, donde `x_vals` y `h` almacenan los valores que hemos definido anteriormente por defecto. Esto de la siguiente manera:

```
Esquemas_de_diferencias(f, df, x_vals=x_values, h=h)
Esquemas_de_diferencias(g, dg, x_vals=x_values, h=h)
Esquemas_de_diferencias(ln, dln, x_vals=x_values, h=h)
Esquemas_de_diferencias(j, dj, x_vals=x_values, h=h)
```

9.2. Comparación entre las Derivadas Numéricas y Analítica

Luego de llamar al algoritmo `Esquemas_de_diferencias`, como se mostró al final de la sección anterior, se obtienen los siguientes gráficos. Estos proporcionan una comparación visual entre la derivada numérica y la derivada analítica:

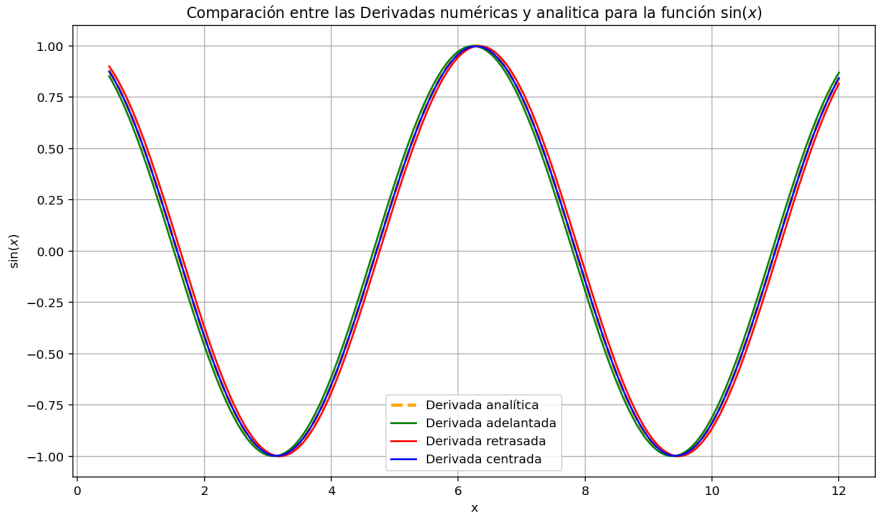

Figura 9.1: Comparación entre las Derivadas numéricas y analítica para la función $\sin(x)$

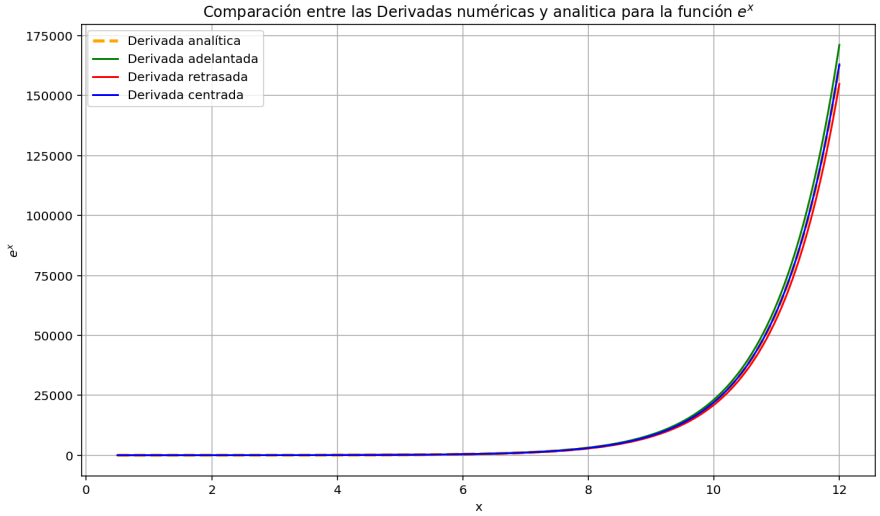

Figura 9.2: Comparación entre las Derivadas numéricas y analítica para la función e^x

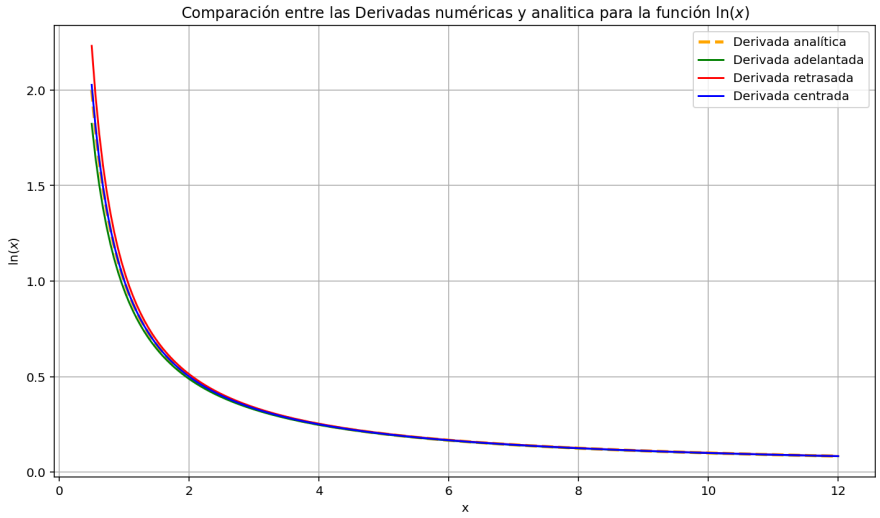

Figura 9.3: Comparación entre las Derivadas numéricas y analítica para la función $\ln(x)$

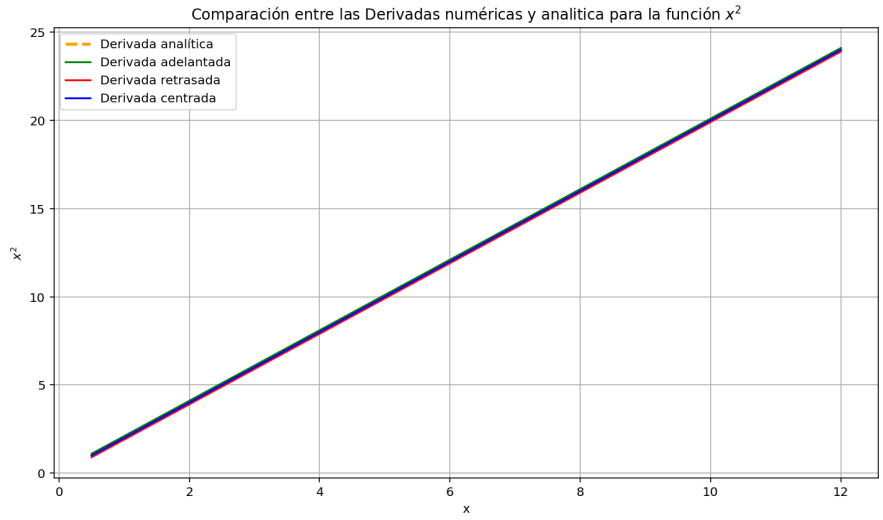

Figura 9.4: Comparación entre las Derivadas numéricas y analítica para la función x^2

9.2.1. Error Relativo

El error relativo entre las derivadas numéricas y analíticas se calcula como:

$$\text{Error relativo} = \frac{|\text{Derivada numérica} - \text{Derivada analítica}|}{|\text{Derivada analítica}|}. \tag{9.1}$$

A continuación se presenta un segundo algoritmo, `Error_relativo`, el cual es una modificación del primero, `Esquemas_de_diferencias`. Este segundo algoritmo cuenta con los mismos parámetros que el primero. Ambos calculan de forma iterada las derivadas numéricas sobre `x_values`, almacenando los resultados en un `numpy.array`. Las particularidades que lo diferencian del primero son, para una función $f(x)$ y su derivada analítica $f'(x)$ dadas, las siguientes:

- se llevan a cabo los cálculos del **error relativo**, según la ecuación (9.1), con cada esquema de derivada numérica (atrasada, adelantada y centrada).
- se genera un gráfico **error relativo vs x** en el cual se comparan los errores relativos por esquema de derivada numérica,
- se retorna el gráfico **error relativo vs x**

El algoritmo `Error_relativo` lo he programado en `Python` del siguiente modo:

```
1
2  def Error_relativo(f, df, x_vals=x_values, h=h):
3
4      df_analitica = np.array([df(x) for x in x_vals])
5      df_adelantada = np.array([(f(x + h) - f(x)) / h for x in x_vals])
6      df_retrasada = np.array([(f(x) - f(x - h)) / h for x in x_vals])
7      df_centrada = np.array([(f(x + h) - f(x - h)) / (2 * h) for x in x_vals])
8
9      error_adelantada = np.abs(df_adelantada - df_analitica) / np.abs(df_analitica)
10     error_retrasada = np.abs(df_retrasada - df_analitica) / np.abs(df_analitica)
11     error_centrada = np.abs(df_centrada - df_analitica) / np.abs(df_analitica)
12
```

```
    plt.figure(figsize=(10, 6))
    # Graficamos el error relativo para los tres esquemas.
    plt.plot(x_vals, error_adelantada, color="green", label="Error derivada adelantada")
    plt.plot(x_vals, error_retrasada, color="red", label="Error derivada retrasada")
    plt.plot(x_vals, error_centrada, color="blue", label="Error derivada centrada")

    plt.legend()
    plt.title(f"Error relativo para la función {f.nombre}")
    plt.xlabel("x")
    plt.ylabel("Error relativo")
    plt.grid(True)
    plt.tight_layout()
    return plt.show()
```

Llamamos al algoritmo, `Error_relativo`, con cada función y su derivada analítica, donde `x_vals` y `h` almacenan los valores que hemos definido anteriormente por defecto.

```
Error_relativo(f, df, x_vals=x_values, h=h)
Error_relativo(g, dg, x_vals=x_values, h=h)
Error_relativo(ln, dln, x_vals=x_values, h=h)
Error_relativo(j, dj, x_vals=x_values, h=h)
```

Finalmente, se han generado los siguientes gráficos. Estos proporcionan una comparación visual entre el error relativo por derivada numérica.

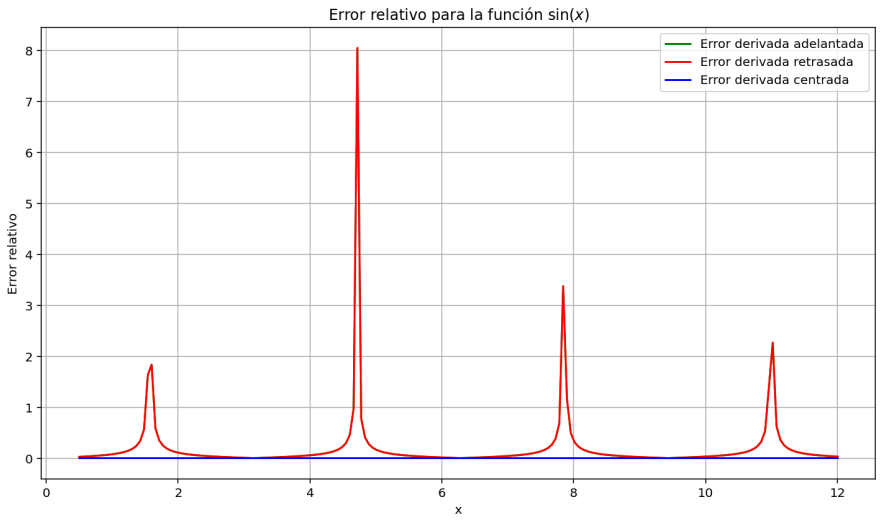

Figura 9.5: Error relativo para la función $\sin(x)$

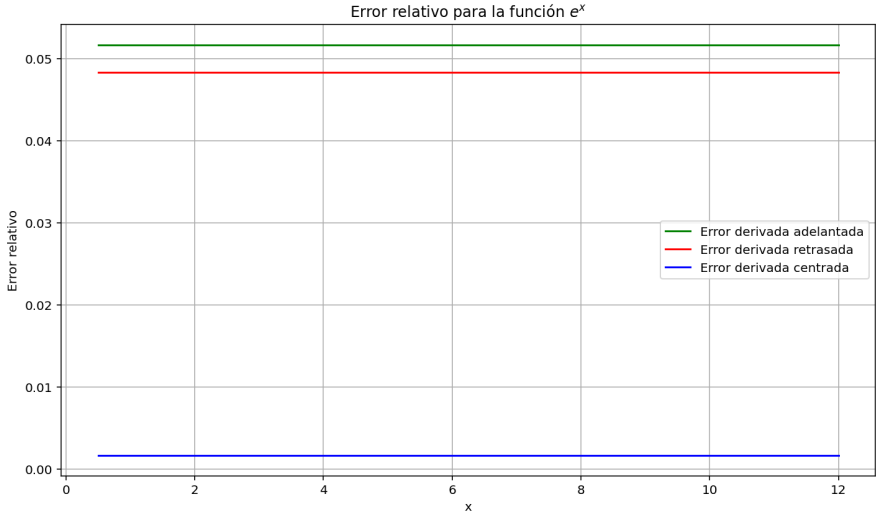

Figura 9.6: Error relativo para la función e^x

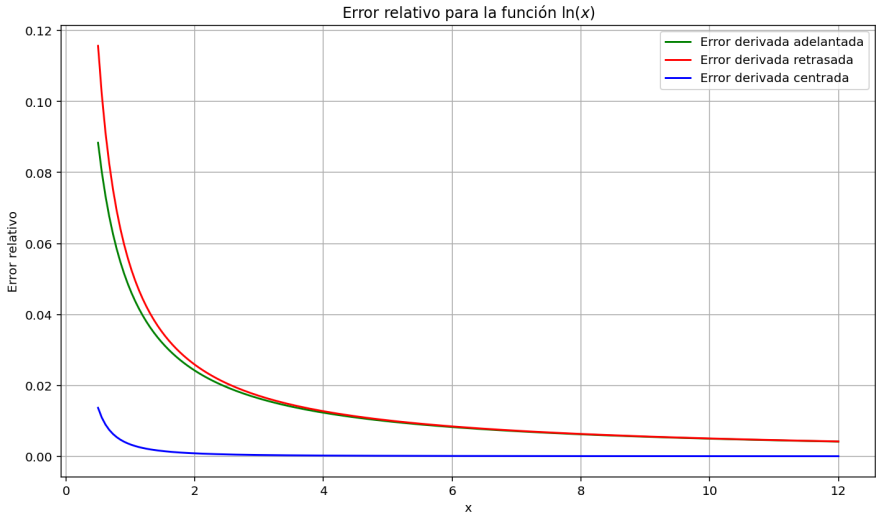

Figura 9.7: Error relativo para la función $\ln(x)$

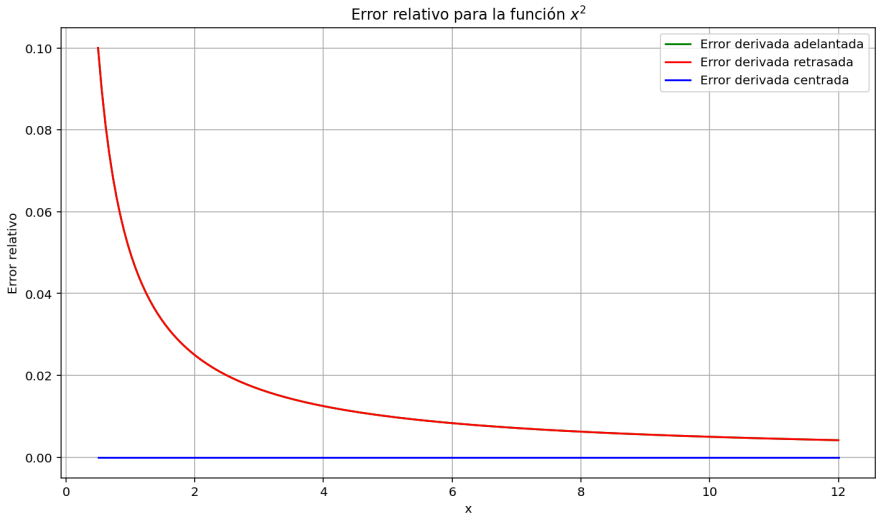

Figura 9.8: Error relativo para la función x^2

9.3. Observaciones sobre los resultados

Precisión de los métodos

- La diferencia centrada presenta menor error relativo que las diferencias adelantada y retrasada, debido a que su error de truncamiento es de segundo orden, $O(h^2)$.
- La diferencia adelantada y la diferencia retrasada tienen errores de truncamiento de primero orden, $O(h)$, por lo que son de menor precisión que la centrada.

Funciones más difíciles de derivar numéricamente

- $f(x) = \ln(x)$: En valores cercanos a $x = 0.1$, se observa un mayor error relativo, ya que la derivada crece rápidamente, amplificando errores de redondeo.
- $f(x) = \sin(x)$ y $f(x) = e^x$: Son más fáciles de derivar, ya que las funciones son suaves y no presentan cambios abruptos.

Errores relativos y tipo de función

- Funciones como $f(x) = x^2$, con derivadas lineales, tienen errores más consistentes en todo el dominio.
- Las funciones exponenciales y trigonométricas son más estables debido a la naturaleza bien condicionada de sus derivadas.

9.4. Conclusión

En esta actividad, hemos implementado en `Python` algoritmos para el cálculo de derivadas numéricas y del error relativo, aplicados a un conjunto finito de valores. Además, se generaron gráficos comparativos que muestran las diferencias entre los esquemas de derivación numérica y la derivada analítica de una función dada.

Los resultados obtenidos en las gráficas fueron analizados, permitiendo discutir tanto la precisión de los métodos como la dificultad de derivar numéricamente las funciones evaluadas.

Capítulo 10

Cálculo Numérico de Derivadas: Esquemas, Errores y Optimización

Fecha de la actividad: 20 de noviembre de 2024

En este apartado se emplearán expansiones en series de Taylor de una función $f(x)$ para derivar tres esquemas de cálculo de la primera derivada numérica. Los esquemas considerados son los siguientes:

- **Diferencia adelantada:** $f'(x) \approx \frac{f(x+h)-f(x)}{h}$.
- **Diferencia retrasada:** $f'(x) \approx \frac{f(x)-f(x-h)}{h}$.
- **Diferencia centrada:** $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$.

Adicionalmente, se llevaron a cabo las siguientes tareas:

1. Determinar el orden del error de truncamiento en cada uno de los esquemas anteriores.
2. Analizar cuál de estos esquemas es más conveniente para estimar la derivada numérica.
3. Considerando el error de redondeo de la máquina, calcular el valor óptimo de h que minimiza el error en cada esquema y comparar los resultados obtenidos.

Sea $h \in \mathbb{R} : 0 < h < 1$.

10.1. Derivada adelantada

Expandiendo $f(x+h)$ en series de Taylor:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \cdots \tag{10.1}$$

Restando en (10.1) el polinomio $\left[f(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \cdots\right]$, se obtiene:

$$f(x+h) - \left[f(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \cdots\right] = hf'(x)$$

Se reordenan términos:

$$hf'(x) = f(x+h) - f(x) - \left[\frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \cdots\right]$$

Dividimos la ecuación por h y despejamos $f'(x)$, obtenemos:

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \left[\frac{h}{2!}f''(x) + \frac{h^2}{3!}f'''(x) + \cdots\right] \tag{10.2}$$

Así, se obtiene una forma para la **derivada adelantada** de $f(x)$ en términos de $f(x)$ y $f(x+h)$.

10.2. Derivada retrasada

Expandiendo $f(x-h)$ en series de Taylor:

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \cdots \tag{10.3}$$

Restando en (10.3) el polinomio $\left[f(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \cdots\right]$, se obtiene:

$$f(x-h) - \left[f(x) - \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \cdots\right] = -hf'(x)$$

Multiplicando la ecuación por -1 y reordenando términos:

$$hf'(x) = f(x) - f(x-h) + \left[\frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \cdots\right]$$

Dividiendo la ecuación por h y despejamos $f'(x)$, obtenemos:

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \left[\frac{h}{2!}f''(x) - \frac{h^2}{3!}f'''(x) + \cdots\right] \tag{10.4}$$

Así, obtenemos una forma para la **derivada retrasada** de $f(x)$ en términos de $f(x)$ y $f(x-h)$.

10.3. Derivada centrada

Consideremos las expansiones en series de Taylor de $f(x+h)$ y $f(x-h)$, obtenidas en las ecuaciones (10.1) y (10.3), respectivamente.

Restando las ecuaciones (10.1) y (10.3) en ese orden, obtenemos:

$$f(x+h) - f(x-h) = 2hf'(x) + 2\frac{h^3}{3!}f'''(x) + 2\frac{h^5}{5!}f^{(5)}(x) + \cdots$$

Dividiendo la ecuación por $2h$ y despejamos $f'(x)$, reordenamos terminos y obtenemos:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \left[\frac{h^2}{3!}f'''(x) + \frac{h^4}{5!}f^{(5)}(x) + \cdots\right] \tag{10.5}$$

Así, obtenemos una forma para la **derivada centrada** de $f(x)$ en términos de $f(x+h)$ y $f(x-h)$.

10.4. Orden del error de truncamiento en los esquemas numéricos

Para el esquema de la **derivada adelantada** de $f(x)$. Notamos en la ecuación (10.2) que el error de truncamiento será del orden $O(h)$. Luego:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} + \frac{h}{2!}f''(\xi) \tag{10.6}$$

Para el esquema de la **derivada retrasada** de $f(x)$. Notamos en la ecuación (10.4) que el error de truncamiento, nuevamente, será del orden $O(h)$. Luego:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h} + \frac{h}{2!} f''(\xi) \tag{10.7}$$

Para el esquema de la **derivada centrada** de $f(x)$. Notamos en la ecuación (10.5) que el error de truncamiento esta vez será del orden $O(h^2)$. Luego:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{3!} f'''(\xi) \tag{10.8}$$

10.4.1. ¿Cuál de estos esquemas es más conveniente para estimar la derivada numérica?

Como el esquema de la derivada centrada de $f(x)$ presenta un error de truncamiento $O(h^2)$, con $0 < h < 1$, es más conveniente el uso de este respecto a los otros para estimar la derivada numérica. Dado que el error en el esquema centrado disminuye más rápidamente a medida que h se reduce, resulta más preciso para valores pequeños de h.

10.5. Cálculo de h óptimo para el esquema de la derivada retrasada

Consideramos $\hat{a} = a(1+\varepsilon)$, donde $\hat{a}$ representa el valor de a registrado por el computador, afectado por un error de redondeo ε, el cual puede depender de a. Para el esquema de la derivada retrasada, usamos la aproximación:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h} + \frac{h}{2!} f''(\xi), \tag{10.9}$$

donde los valores de $f(x-h)$ y $f(x)$ se redondean como:

$$\hat{f}(x-h) = f(x-h)(1+\varepsilon_2), \quad \text{y} \quad \hat{f}(x) = f(x)(1+\varepsilon_1). \tag{10.10}$$

Sustituyendo estos valores en (10.9), obtenemos:

$$f'(x) - \frac{\hat{f}(x) - \hat{f}(x-h)}{h} = \frac{h}{2!} f''(\xi) \tag{10.11}$$

Expandiendo y simplificando:

$$f'(x) - \frac{f(x)(1+\varepsilon_1) - f(x-h)(1+\varepsilon_2)}{h} = \frac{h}{2!} f''(\xi)$$

$$f'(x) - \frac{f(x) - f(x-h)}{h} = \frac{h}{2!} f''(\xi) + \frac{f(x)\varepsilon_1 - f(x-h)\varepsilon_2}{h}$$

Aplicando la desigualdad triangular:

$$\left| f'(x) - \frac{f(x) - f(x-h)}{h} \right| \leq \frac{h}{2} |f''(\xi)| + \frac{|\varepsilon_1|}{h} |f(x)| + \frac{|\varepsilon_2|}{h} |f(x-h)|$$

Podemos asumir que:

1. Existe un valor ε^* tal que $\varepsilon^* \gg |\varepsilon_1|$ y $\varepsilon^* \gg |\varepsilon_2|$.
2. Si $h \ll 1$, entonces $\xi \approx x \quad \wedge \quad |f(x-h)| \approx |f(x)|$.

Entonces:

$$\left| f'(x) - \frac{f(x) - f(x-h)}{h} \right| \leq \frac{h}{2} |f''(x)| + \frac{2\varepsilon^*}{h} |f(x)|$$

De aquí, el **error absoluto** (analítico) es:

$$E_{\text{abs}} \leq \frac{2\varepsilon^*}{h} |f(x)| + \frac{h}{2} |f''(x)| \tag{10.12}$$

El error absoluto **derivado** es:

$$E'_{\text{abs}} \leq -\frac{2\varepsilon^*}{h^2} |f(x)| + \frac{1}{2} |f''(x)| \tag{10.13}$$

Por otro lado, el **valor óptimo** de h se obtiene a partir de $E'_{\text{abs}} = 0$.

$$-\frac{2\varepsilon^*}{h^2} |f(x)| + \frac{1}{2} |f''(x)| = 0$$

Despejamos h:

$$h = \sqrt{\frac{4\varepsilon^* |f(x)|}{|f''(x)|}} \tag{10.14}$$

Se obtiene el valor optimo de h para el esquema de la derivada retrasada.

10.6. h óptimo para el esquema de la derivada adelantada

Ya sabemos que el esquema de la derivada adelantada es:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} + \frac{h}{2} f''(\xi) \tag{10.15}$$

Donde el termino $\frac{h}{2} f''(\xi)$ es el error de truncamiento que depende de la segunda derivada de $f(x)$. Para considerar los errores de redondeo de la máquina, utilizamos las siguientes expresiones:

$$\hat{f}(x+h) = f(x+h)(1+\varepsilon_2), \quad \text{y} \quad \hat{f}(x) = f(x)(1+\varepsilon_1). \tag{10.16}$$

Consideramos los errores de redondeo de la maquina (10.16) en el esquema de la derivada adelantada (10.15), y expandimos términos:

$$\begin{aligned} f'(x) &= \frac{\hat{f}(x+h) - \hat{f}(x)}{h} + \frac{h}{2} f''(\xi) \\ &= \frac{f(x+h)(1+\varepsilon_2) - f(x)(1+\varepsilon_1)}{h} + \frac{h}{2} f''(\xi) \\ &= \frac{f(x+h) - f(x)}{h} + \frac{\varepsilon_2 f(x+h) - \varepsilon_1 f(x)}{h} + \frac{h}{2} f''(\xi) \end{aligned}$$

Restamos $\frac{f(x+h)-f(x)}{h}$ en ambos miembros de la ecuación, obtenemos:

$$f'(x) - \frac{f(x+h) - f(x)}{h} = \frac{\varepsilon_2 f(x+h) - \varepsilon_1 f(x)}{h} + \frac{h}{2} f''(\xi)$$

Aplicando desigualdad triangular:

$$\left| f'(x) - \frac{f(x+h) - f(x)}{h} \right| \leq \left| \frac{\varepsilon_2 f(x+h) - \varepsilon_1 f(x)}{h} \right| + \left| \frac{h}{2} f''(\xi) \right|$$

Podemos asumir que:

1. Existe un valor ε^* tal que $\varepsilon^* \gg |\varepsilon_1|$ y $\varepsilon^* \gg |\varepsilon_2|$.
2. Si $h \ll 1$, entonces $\xi \approx x \quad \wedge \quad |f(x+h)| \approx |f(x)|$.

Luego, tenemos para el **error absoluto**:

$$\begin{aligned} E_{\text{abs}} &\leq \frac{\varepsilon_2}{h}|f(x+h)| + \frac{\varepsilon_1}{h}|f(x)| + \frac{h}{2}|f''(\xi)| \\ &\leq \frac{\varepsilon^*}{h}|f(x)| + \frac{\varepsilon^*}{h}|f(x)| + \frac{h}{2}|f''(\xi)| \\ &= \frac{2\varepsilon^*}{h}|f(x)| + \frac{h}{2}|f''(\xi)| \end{aligned}$$

Así, deducimos el **error absoluto** del esquema de derivada adelantada.

$$E_{\text{abs}} \leq \frac{2\varepsilon^*}{h}|f(x)| + \frac{h}{2}|f''(\xi)|$$

Derivamos respecto a h el error absoluto E_{abs}.

$$E'_{\text{abs}} \leq -\frac{\varepsilon^*}{h^2}|f(x)| + \frac{1}{2}|f''(\xi)|$$

Para encontrar el valor optimo de h, resolvemos la ecuación $E'_{\text{abs}} = 0$ para optimizar el error, obtenemos la siguiente ecuación:

$$0 = -\frac{\varepsilon^*}{h^2}|f(x)| + \frac{1}{2}|f''(\xi)|$$

Despejando h, encontramos el valor óptimo:

$$h = \sqrt{\frac{4\varepsilon^*|f(x)|}{|f''(x)|}} \tag{10.17}$$

10.7. h óptimo para el esquema de la derivada centrada

Conocemos el esquema de la derivada centrada:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{3!}f'''(\xi) \tag{10.18}$$

Donde el termino $\frac{h^2}{3!}f'''(\xi)$ es el error de truncamiento que depende de la tercera derivada de $f(x)$. Para considerar los errores de redondeo de la máquina, utilizamos las siguientes expresiones:

$$\hat{f}(x+h) = f(x+h)(1+\varepsilon_2), \quad \text{y} \quad \hat{f}(x-h) = f(x-h)(1+\varepsilon_1). \tag{10.19}$$

Consideramos los errores de redondeo de la maquina (10.19) en el esquema de la derivada centrada (10.18), y expandimos términos:

$$\begin{aligned} f'(x) &= \frac{\hat{f}(x+h) - \hat{f}(x-h)}{2h} + \frac{h^2}{3!} f'''(\xi) \\ &= \frac{f(x+h)(1+\varepsilon_2) - f(x-h)(1+\varepsilon_1)}{2h} + \frac{h^2}{3!} f'''(\xi) \\ &= \frac{f(x+h) - f(x)}{2h} + \frac{\varepsilon_2 f(x+h) - \varepsilon_1 f(x)}{2h} + \frac{h^2}{3!} f'''(\xi) \end{aligned}$$

Restamos $\frac{f(x+h)-f(x-h)}{2h}$ en ambos miembros de la ecuación, obtenemos:

$$f'(x) - \frac{f(x+h) - f(x-h)}{2h} = \frac{\varepsilon_2 f(x+h) - \varepsilon_1 f(x-h)}{2h} + \frac{h^2}{3!} f'''(\xi)$$

Aplicando desigualdad triangular, y asumiendo que:

1. Existe un valor ε^* tal que $\varepsilon^* \gg |\varepsilon_1|$ y $\varepsilon^* \gg |\varepsilon_2|$.
2. Si $h \ll 1$, entonces $\xi \approx x \quad \wedge \quad |f(x+h)| \approx |f(x)| \quad \wedge \quad |f(x-h)| \approx |f(x)|$.

Se tiene el siguiente desarrollo:

$$\begin{aligned} \left| f'(x) - \frac{f(x+h) - f(x-h)}{2h} \right| &\leq \left| \frac{\varepsilon_2 f(x+h) - \varepsilon_1 f(x-h)}{2h} \right| + \left| \frac{h^2}{3!} f'''(\xi) \right| \\ &\leq \left| \frac{\varepsilon_2 f(x+h)}{2h} \right| + \left| \frac{\varepsilon_1 f(x-h)}{2h} \right| + \left| \frac{h^2}{6} f'''(\xi) \right| \\ &= \frac{\varepsilon_2}{2h} |f(x+h)| + \frac{\varepsilon_1}{2h} |f(x-h)| + \frac{h^2}{6} |f'''(\xi)| \\ &\leq \frac{\varepsilon^*}{2h} |f(x)| + \frac{\varepsilon^*}{2h} |f(x)| + \frac{h^2}{6} |f'''(\xi)| \\ &= \frac{\varepsilon^*}{h} |f(x)| + \frac{h^2}{6} |f'''(\xi)| \end{aligned}$$

Notamos que el **error absoluto** estara dado por:

$$E_{\text{abs}} \leq \frac{h^2}{6} |f'''(\xi)| + \frac{\varepsilon^*}{h} |f(x)|. \tag{10.20}$$

Para el valor optimo de h, nuevamente derivamos el error absoluto respecto a h y resolvemos $E'_{\text{abs}} = 0$:

$$-\frac{\varepsilon^*}{h^2} |f(x)| + \frac{h}{3} |f'''(x)| = 0. \tag{10.21}$$

Resolviendo para h, obtenemos:

$$h = \left(\frac{3\varepsilon^* |f(x)|}{|f'''(x)|} \right)^{1/3}. \tag{10.22}$$

10.8. Comparación de resultados

Se observa que el valor óptimo de h es el mismo para los esquemas de derivada adelantada y retrasada, pero difiere del obtenido para el esquema de derivada centrada. En todos los casos, el valor óptimo de h se expresa como una raíz: cuadrada para los esquemas adelantado y retrasado, y cúbica para el esquema centrado. El argumento de la raíz incluye un factor proporcional a ε^* y la relación entre los valores absolutos de $f(x)$ y sus derivadas: $f''(x)$ en el caso de la raíz cuadrada y $f'''(x)$ en el caso de la raíz cúbica.

10.9. Conclusión

El desarrollo de la actividad permite aprender a calcular derivadas numéricas mediante tres esquemas (adelantado, retrasado y centrado), analizar sus errores de truncamiento y redondeo, y determinar el valor óptimo de h para minimizar el error total. Además, fomenta el uso de técnicas matemáticas como series de Taylor y análisis de la concavidad para evaluar y optimizar estos métodos, destacando la precisión superior del esquema centrado.

Capítulo 11

Derivación Numérica en Datos Discretos con Ruido

Fecha de la actividad: 23 de noviembre de 2024

En este capítulo, analizaremos las limitaciones de un esquema de derivada centrada aplicado a una función f, en la que solo se dispone de un conjunto finito de pares ordenados (x_i, f_i). Implementaremos un esquema numérico de derivación centrada para datos discretos y evaluaremos cómo el ruido afecta los resultados.

Utilizaremos una función generada en `Python` que simula una señal sinusoidal con un ruido uniforme de magnitud 10^{-2}. Posteriormente, exploraremos estrategias para mitigar el impacto del ruido en el cálculo de la derivada, mejorando así la precisión de las estimaciones.

11.1. Análisis del esquema de deriva centrada de *f(x)* para pares ordenados dados

- Se nos proporciona un número finito $n \in \mathbb{N}$ de valores $x_i \in X \subseteq \mathbb{R}$, con $0 \leq i \leq n$. Estos valores están ordenados de manera creciente desde x_0 hasta x_n y presentan una separación equidistante entre cada par de elementos consecutivos, es decir, cada x_i equidista de su sucesor y de su antecesor.

$$X = \{x_0,\, x_1,\, \ldots\, , x_{n-1}\, , x_n\}$$

- Consideramos una función f definida en el conjunto X, al menos derivable en $X \setminus \{x_0,\, x_n\}$. Se nos proporcionan valores f_i, tales que $\forall\, x_i \in X,\, \exists\, f_i = f(x_i) \in F \subseteq \mathbb{R}$. Por lo tanto, el conjunto F contiene el mismo número finito n de valores f_i correspondientes a los x_i.

$$F = \{f_0,\, f_1,\, \ldots\, , f_{n-1}\, , f_n\}$$

- Consideramos el producto cartesiano de X y F,

$$X \times F = \{(x_0,\, f_0),\, (x_1,\, f_1),\, \ldots\, , (x_{n-1},\, f_{n-1})\, , (x_n,\, f_n)\}$$

  donde $X \times F$ contiene a los n pares ordenados $(x_i,\, f_i)$.

- Analizamos h el esquema de la derivada centrada para $f(x)$. Este esquema usualmente se expresa de la siguiente manera:

$$f'(x_i) \approx \frac{f(x_i + h) - f(x_i - h)}{2h}. \tag{11.1}$$

Consideramos h como la distancia entre dos valores consecutivos en el conjunto X. Por la equidistancia entre dos valores consecutivos cualesquiera en X, podemos representar h de dos maneras:

$$h = x_{i+1} - x_i \tag{11.2}$$

$$h = x_i - x_{i-1} \tag{11.3}$$

Dependiendo de la forma de representar h que se escoga, notaremos que:

1. Representando h como en (11.2), en $x_i + h$ se tiene $x_i + h = x_i + (x_{i+1} - x_i) = x_{i+1}$, por lo tanto

$$x_i + h = x_{i+1} \ . \tag{11.4}$$

2. Representando h como en (11.3), en $x_i - h$ se tiene $x_i - h = x_i - (x_i - x_{i-1}) = x_{i-1}$, por lo tanto

$$x_i - h = x_{i-1} \ . \tag{11.5}$$

Reemplazando (11.4) y (11.5) en (11.1), se obtiene

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{2h} \ . \tag{11.6}$$

- Para estimar la derivada de f, dado que solo se conocen los pares de valores $(x_i,\, f_i) \in X \times F$, utilizamos el esquema de derivada centrada definido en (11.6). Dado que $f_i = f(x_i)$, el esquema en este contexto puede expresarse de la siguiente forma:

$$f'(x_i) \approx \frac{f_{i+1} - f_{i-1}}{2h} \tag{11.7}$$

- Notamos que para los pares $(x_0,\, f_0)$ y $(x_n,\, f_n)$ no es posible aplicar el esquema (11.7).
  - Si $(x_i,\, f_i) = (x_0,\, f_0)$, se tendrá en (11.7) que $f'(x_0) \approx \frac{f_1 - f_{-1}}{2h}$, pero $f_{-1} \notin F$, por lo tanto no podemos estimar $f'(x_0)$ con este esquema.
  - Si $(x_i,\, f_i) = (x_n,\, f_n)$, se tendrá en (11.7) que $f'(x_n) \approx \frac{f_{n+1} - f_{n-1}}{2h}$, pero $f_{n+1} \notin F$, por lo tanto no podemos estimar $f'(x_n)$ con este esquema.

- Determinamos las cotas máximas y mínimas de f_{i+1} y f_{i-1} en el esquema de derivada centrada (11.7), basándonos en las siguientes estimaciones:

1. Para $f'(x_1)$, observamos que $f'(x_1) \approx \frac{f_2 - f_0}{2h}$. Por lo tanto, los valores mínimos que pueden asumir f_{i+1} y f_{i-1} son f_2 y f_0, respectivamente.
2. Para $f'(x_{n-1})$, observamos que $f'(x_{n-1}) \approx \frac{f_n - f_{n-2}}{2h}$. En este caso, los valores máximos que pueden asumir f_{i+1} y f_{i-1} son f_n y f_{n-2}, respectivamente.

Estas consideraciones son fundamentales para comprender el programa que calculará la derivada centrada de f, suponiendo que solo se disponen de datos discretos.

11.2. Programa para calcular la derivada centrada de *f(x)* con pares ordenados dados

A continuación, se presenta una función programada en `Python` que estima la derivada de una función f, cuyos valores $(x_i,\ f_i)$ son conocidos y discretos, donde los valores consecutivos en x_i están uniformemente espaciados.

La función utiliza el esquema de derivada centrada (11.7), como se explicó en la sección anterior. El código es el siguiente:

```
def derivada_centrada(x, f):
    h = x[1] - x[0]
    df = (f[2:] - f[:-2]) / (2 * h)
    return df
```

La función cuenta con dos parámetros:

- `x`: Un `array` de valores que representan los $x_i \in X$, uniformemente espaciados.
- `f`: Un `array` de valores que representan los $f_i \in F$.

En primer lugar, la función calcula el paso h, el cual es constante entre dos valores consecutivos en el array `x`.

A continuación, se utiliza el esquema de derivada centrada (11.7) para calcular $f'(x_i)$ en los pares de valores desde $(x_1,\ f_1)$ hasta $(x_{n-1},\ f_{n-1})$, considerando las siguientes características de f_{i+1} y f_{i-1}:

1. `f[2:]`: Representa f_{i+1}, con valor mínimo f_2 y valor máximo f_n.
2. `f[:-2]`: Representa f_{i-1}, con valor mínimo f_0 y valor máximo f_{n-2}.

Finalmente, la función retorna un `array` con los valores de las derivadas aproximadas calculados.

11.3. Generación de datos

Se genera en `Python` una serie de datos de la siguiente manera:

```
from numpy import pi, linspace, sin, random
gen = random.default_rng()
x = linspace(0, 2*pi, 256)
f = sin(x) + gen.uniform(low=-1e-2, high=1e-2, size=x.size)
```

En este caso, f presenta una tendencia sinusoidal con la forma de $\sin(x)$, a la cual se le agrega un ruido uniforme aleatorio de magnitud 10^{-2}.

Utilizando `matplotlib.pyplot`, se grafica la función generada con el ruido aleatorio, obteniendo el gráfico presentado en la Figura 11.1, generado en parte mediante el siguiente código:

```
plt.plot(x, f, label="f(x) con ruido", color="blue")
```

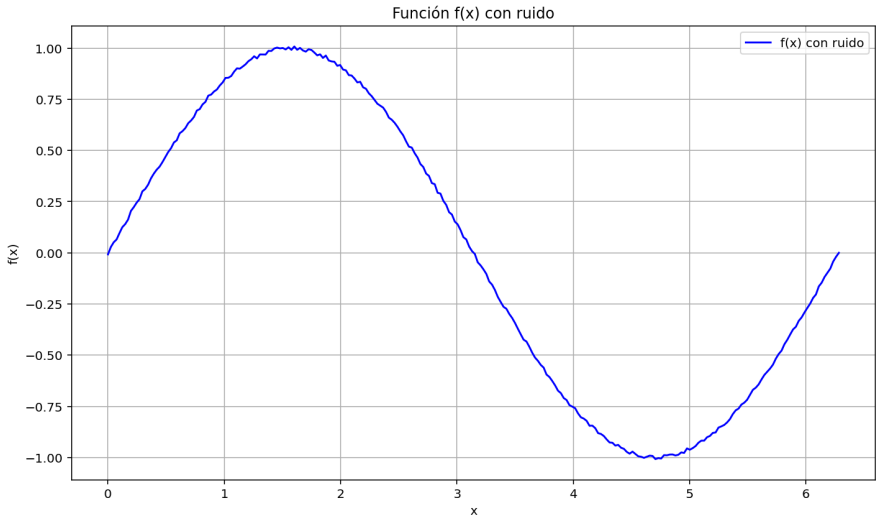

Figura 11.1: Función con tendencia sinusoidal de la forma $\sin(x)$, con ruido uniforme aleatorio de magnitud 10^{-2} agregado.

Antes de graficar la derivada centrada de la función generada, es importante considerar lo siguiente:

El programa utilizado para calcular las derivadas retorna un `array` con los valores desde $f'(x_1)$ hasta $f'(x_{n-1})$. Dado que el esquema centrado no permite calcular $f'(x_0)$ ni $f'(x_n)$, el `array x` contiene dos elementos más que el `array` de derivadas retornado por la función `derivada_centrada`.

Para graficar correctamente la derivada centrada de la función generada, es necesario definir un nuevo `array` para `x`, excluyendo los valores x_0 y x_n. Esto se logra con el siguiente código:

```
x_df = x[1:-1]
```

Una vez definido este nuevo `array`, se grafica la derivada centrada de la función generada. El gráfico resultante se muestra en la Figura 11.2, generado en parte mediante el siguiente código:

```
plt.plot(x_df, df_dx, label="Derivada centrada de f(x)", color="red")
```

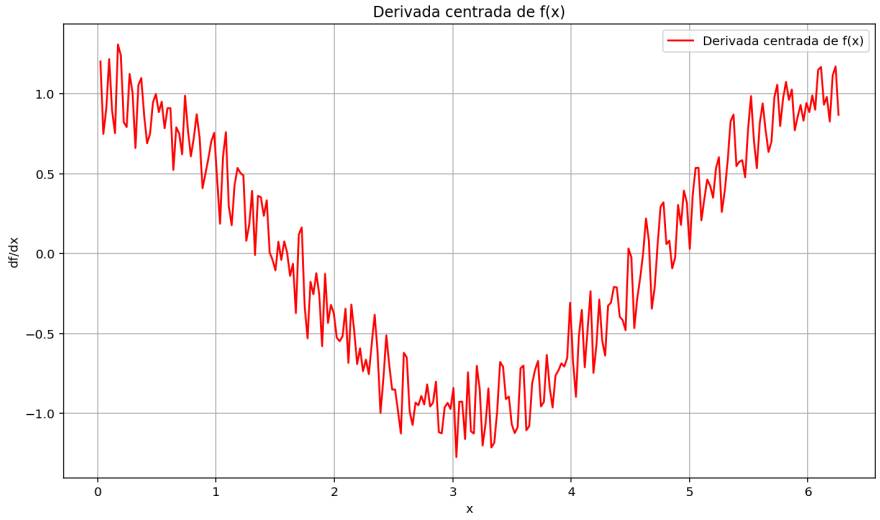

Figura 11.2: Derivada centrada de la función $f(x)$ generada con tendencia sinusoidal de la forma $\sin(x)$, con ruido uniforme aleatorio de magnitud 10^{-2} agregado.

11.4. Observaciones

Resultados Observados

- El ruido presente en $f(x)$ se amplifica significativamente al aplicar el esquema de derivada centrada. Este es un efecto común en los métodos de diferenciación numérica, ya que pequeñas perturbaciones en los datos originales se reflejan como grandes oscilaciones en el cálculo de las pendientes.
- A pesar del ruido, se aprecia que la tendencia general de la derivada sigue una forma similar a $\cos(x)$, que corresponde a la derivada analítica de $\sin(x)$.

11.4.1. Mitigación del Ruido

Para mitigar los efectos del ruido en el cálculo de la derivada, se proponen las siguientes estrategias:

1. **Filtrado previo de los datos:** Suavizar los valores de $f(x)$ antes de calcular la derivada. Un filtro gaussiano es una opción adecuada para este propósito:

```
from scipy.ndimage import gaussian_filter1d
f_suavizado = gaussian_filter1d(f, sigma=2)
df_dx = derivada_centrada(x, f_suavizado)
```

2. **Uso de pasos mayores h:** Incrementar el espaciado entre los puntos en los datos discretos reduce la sensibilidad al ruido. No obstante, esto también disminuye la resolución de los cálculos.

11.5. Conclusión

En esta actividad, se analizaron las limitaciones de un esquema de diferencias centradas aplicado a un conjunto de valores discretos $(x_i,\ f_i)$ de una función f. Se programó en `Python` una función que calcula la derivada centrada de una función, partiendo únicamente de un conjunto de valores discretos.

Además, se implementó un esquema de diferencias centradas para estimar la derivada numérica de datos discretos que incluyen ruido. Se observó que el ruido amplifica los errores en la derivada calculada; sin embargo, estrategias como el filtrado previo de los datos o ajustes en el paso pueden mejorar significativamente los resultados obtenidos.

Capítulo 12

Deducción de los pesos de una regla exacta para polinomios de grado 2

Fecha de la actividad: 20 de octubre de 2024

En este capítulo, analizaremos el proceso de deducción de los pesos w_i de una regla de cuadratura que es exacta para polinomios de hasta grado 2. Para ello, consideraremos que se desea aproximar una integral utilizando una combinación lineal de los valores de la función en ciertos puntos específicos. El problema plantea que esta regla debe ser exacta para polinomios de grado 2.

12.1. Planteamiento del Problema

Suponga que se quiere aproximar la siguiente integral como:

$$\int_{-1}^{1} dx\, f(x) \sin\left(\frac{\pi x}{2}\right) = w_{-1} \cdot f(-1) + w_0 \cdot f(0) + w_1 \cdot f(1) \tag{12.1}$$

Nuestro objetivo es determinar los valores de los pesos w_{-1}, w_0, y w_1 bajo la suposición de que esta regla es exacta para polinomios de hasta grado 2.

12.2. Análisis y Condiciones de Exactitud

Para asegurar que la regla es exacta para polinomios de hasta grado 2, la integral será exacta para las siguientes funciones:

- $f(x) = 1$
- $f(x) = x$
- $f(x) = x^2$

La exactitud de la regla implica que para cada uno de estos casos la igualdad planteada en la ecuación (12.1) debe cumplirse.

12.3. Cálculo de los Pesos para cada Caso

Para cada función mencionada, plantearemos la ecuación correspondiente y resolveremos los valores de los pesos w_i.

Caso $f(x) = 1$: Sustituyendo $f(x) = 1$ en la ecuación de la integral, tenemos:

$$\int_{-1}^{1} dx \, \sin\left(\frac{\pi x}{2}\right) = w_{-1} + w_0 + w_1 \tag{12.2}$$

Evaluando esta integral se llega a la ecuación:

$$0 = w_{-1} + w_0 + w_1 \tag{12.3}$$

Caso $f(x) = x$: Sustituyendo $f(x) = x$, obtenemos:

$$\int_{-1}^{1} dx \, x \sin\left(\frac{\pi x}{2}\right) = -w_{-1} + w_1 \tag{12.4}$$

Al resolver la integral, obtenemos:

$$\frac{8}{\pi^2} = -w_{-1} + w_1 \tag{12.5}$$

Caso $f(x) = x^2$: Finalmente, para $f(x) = x^2$:

$$\int_{-1}^{1} dx \, x^2 \sin\left(\frac{\pi x}{2}\right) = w_{-1} + w_1 \tag{12.6}$$

Resolviendo la integral, se llega a la ecuación:

$$0 = w_{-1} + w_1 \tag{12.7}$$

12.4. Resolución del Sistema de Ecuaciones

A partir de las ecuaciones obtenidas (12.3), (12.5) y (12.7), podemos construir un sistema de ecuaciones para resolver los valores de w_{-1}, w_0, y w_1.

$$\begin{cases} 0 = w_{-1} + w_0 + w_1 \\ \frac{8}{\pi^2} = -w_{-1} + w_1 \\ 0 = w_{-1} + w_1 \end{cases} \tag{12.8}$$

Procedemos a resolver este sistema para encontrar los pesos. Restando w_{-1} de ambos lados en la ecuación (12.7), se obtiene:

$$w_1 = -w_{-1} \tag{12.9}$$

Sustituyendo la ecuación (12.9) en (12.5), tenemos:

$$\frac{8}{\pi^2} = 2w_1 \tag{12.10}$$

Dividiendo entre 2 en ambos lados de la ecuación (12.10), deducimos:

$$w_1 = \frac{4}{\pi^2} \tag{12.11}$$

Por lo tanto, de la ecuación (12.9) también se deduce:

$$w_{-1} = -\frac{4}{\pi^2} \tag{12.12}$$

Finalmente, sustituyendo los valores de w_{-1} y w_1 obtenidos en las ecuaciones (12.12) y (12.11) en la ecuación (12.3), encontramos:

$$w_0 = 0 \tag{12.13}$$

Por lo tanto, los valores de los pesos w_i en la ecuación (12.1) son:

$$w_{-1} = -\frac{4}{\pi^2}, \quad w_0 = 0, \quad w_1 = \frac{4}{\pi^2}.$$

12.5. Conclusión

Hemos deducido los valores de los pesos w_{-1}, w_0, y w_1 bajo las condiciones de exactitud para polinomios de hasta grado 2. Estos pesos permiten aproximar la integral planteada con una regla de cuadratura adecuada para funciones polinómicas de segundo grado.

Capítulo 13

Deducción de los pesos de una regla exacta para polinomios de grado 3

Fecha de la actividad: 1 de diciembre de 2024

En este capítulo, se aborda el problema de encontrar una regla de integración que sea exacta para polinomios de hasta grado 3, basada en el conocimiento de $f(x)$ y su derivada $f'(x)$ en los extremos del intervalo $[0, 1]$. La integral a aproximar está dada por:

$$\int_0^1 f(x)\,dx = w_0 f(0) + w_1 f(1) + w_2 f'(0) + w_3 f'(1), \tag{13.1}$$

donde se deben determinar los pesos w_0, w_1, w_2, y w_3. Además, demostraremos que el error de este método depende de la cuarta derivada de $f(x)$, siendo proporcional a h^5, con $h = 1$.

13.1. Cálculo de los pesos

Consideremos $f(x)$ como un polinomio de grado 3:

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

La integral exacta es:

$$\int_0^1 f(x)\,dx = \int_0^1 \left(a_0 + a_1 x + a_2 x^2 + a_3 x^3\right) dx = a_0 + \frac{a_1}{2} + \frac{a_2}{3} + \frac{a_3}{4}.$$

Evaluemos $f(x)$ y $f'(x)$ en $x = 0$ y $x = 1$:

$$f(0) = a_0, \quad f'(0) = a_1, \quad f(1) = a_0 + a_1 + a_2 + a_3, \quad f'(1) = a_1 + 2a_2 + 3a_3.$$

Sustituyendo estos valores en la regla de integración:

$$\int_0^1 f(x)\,dx = w_0 f(0) + w_1 f(1) + w_2 f'(0) + w_3 f'(1).$$

Expandiendo y agrupando por coeficientes:

$$\int_0^1 f(x)\,dx = (w_0 + w_1)a_0 + (w_1 + w_2 + w_3)a_1 + (w_1 + 2w_3)a_2 + (w_1 + 3w_3)a_3.$$

Igualamos término a término con la integral exacta:

$$\begin{aligned} w_0 + w_1 &= 1, \\ w_1 + w_2 + w_3 &= \frac{1}{2}, \\ w_1 + 2w_3 &= \frac{1}{3}, \\ w_1 + 3w_3 &= \frac{1}{4}. \end{aligned}$$

Resolviendo el sistema:

$$\begin{aligned} w_0 &= \frac{11}{24}, \\ w_1 &= \frac{1}{8}, \\ w_2 &= \frac{1}{3}, \\ w_3 &= -\frac{1}{6}. \end{aligned}$$

13.2. Demostración del error

Sea $R(f)$ el residuo de la regla para funciones generales $f(x)$. Utilizando la fórmula de Taylor, expandimos $f(x)$ alrededor de $x = \xi \in [0, 1]$:

$$f(x) = f(\xi) + f'(\xi)(x-\xi) + \frac{f''(\xi)}{2!}(x-\xi)^2 + \frac{f^{(3)}(\xi)}{3!}(x-\xi)^3 + \frac{f^{(4)}(\xi)}{4!}(x-\xi)^4 + \cdots$$

Integrando término a término en $[0, 1]$, los términos hasta grado 3 son exactos por construcción. El error se origina en el término de cuarto grado:

$$R(f) = \int_0^1 \frac{f^{(4)}(\xi)}{4!}(x-\xi)^4 dx.$$

Resolviendo esta integral para $x \in [0, 1]$, encontramos que:

$$R(f) = \frac{f^{(4)}(\xi)}{720}.$$

Por lo tanto, el error del método depende de la cuarta derivada de $f(x)$ como $E = \frac{f^{(4)}(\xi)}{720}$, lo cual verifica lo solicitado.

13.3. Conclusión

Hemos deducido los pesos w_0, w_1, w_2, w_3 de la regla de integración y demostrado que el error del método depende de la cuarta derivada de $f(x)$, siendo proporcional a h^5 con $h = 1$ en este caso.

Capítulo 14

Demostraciones de Reglas de Integración

Fecha de la actividad: 2 de diciembre de 2024

En este capítulo, se demuestran diversas reglas de integración numérica, incluyendo métodos de cuadratura abiertos, como las reglas Gaussianas, y métodos cerrados, como las reglas de Newton-Cotes. Estas demostraciones utilizan el método de coeficientes no determinados para garantizar la exactitud en polinomios de ciertos grados.

14.1. Demostración con la regla de cuadratura cerrada de Newton-Cotes

La regla de integración es:

$$\int_{-\pi}^{\pi} f(x)\,dx = \frac{1}{2}\left[\frac{1}{\pi^2}\Big(f(\pi) - f(-\pi)\Big) + \frac{16}{\pi^2}\Big(f\left(\frac{\pi}{2}\right) - f\left(-\frac{\pi}{2}\right)\Big)\right].$$

Demostración:

Supongamos que la integral se aproxima como:

$$\int_{-\pi}^{\pi} f(x)\,dx \approx Af(\pi) + Bf(-\pi) + Cf\left(\frac{\pi}{2}\right) + Df\left(-\frac{\pi}{2}\right).$$

Queremos determinar los coeficientes A, B, C, D de forma que la regla sea exacta para polinomios de grado $n \leq 3$.

1. Expansión de Taylor: Sea $f(x)$ un polinomio de grado 3:

$$f(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3.$$

2. Sustitución en la regla de cuadratura: Sustituimos los valores de $f(x)$ en $\pm\pi$ y $\pm\frac{\pi}{2}$:

$$\begin{aligned} f(\pi) &= c_0 + c_1\pi + c_2\pi^2 + c_3\pi^3, \\ f(-\pi) &= c_0 - c_1\pi + c_2\pi^2 - c_3\pi^3, \\ f\left(\frac{\pi}{2}\right) &= c_0 + c_1\frac{\pi}{2} + c_2\frac{\pi^2}{4} + c_3\frac{\pi^3}{8}, \\ f\left(-\frac{\pi}{2}\right) &= c_0 - c_1\frac{\pi}{2} + c_2\frac{\pi^2}{4} - c_3\frac{\pi^3}{8}. \end{aligned}$$

3. Integración exacta: Calculamos la integral exacta para cada término:

$$\int_{-\pi}^{\pi} c_0\,dx = 2\pi c_0, \quad \int_{-\pi}^{\pi} c_1 x\,dx = 0, \quad \int_{-\pi}^{\pi} c_2 x^2\,dx = \frac{2}{3}\pi^3 c_2, \quad \int_{-\pi}^{\pi} c_3 x^3\,dx = 0.$$

4. Igualación de coeficientes: Comparando los términos, determinamos los valores de A, B, C, D:

$$A = B = \frac{1}{2\pi^2}, \quad C = -D = \frac{8}{\pi^2}.$$

Por lo tanto, la regla queda demostrada.

14.2. Demostración con la regla de cuadratura abierta de Newton-Cotes

La regla de integración es:

$$\int_0^{2h} f(x)\,dx = \frac{h}{15}\Big[7f(0) + 16f(h) + 7f(2h)\Big] + \frac{h^2}{15}\Big[f'(0) - f'(2h)\Big].$$

Demostración:

Partimos de la expansión de Taylor para $f(x)$ alrededor de $x = 0$:

$$f(x) = f(0) + f'(0)x + \frac{f''(0)x^2}{2!} + \frac{f'''(0)x^3}{3!} + \cdots.$$

1. Sustitución en los puntos: Evaluamos $f(x)$ en $x = 0$, $x = h$, y $x = 2h$:

$$\begin{aligned} f(0) &= f(0), \\ f(h) &= f(0) + f'(0)h + \frac{f''(0)h^2}{2} + \frac{f'''(0)h^3}{6}, \\ f(2h) &= f(0) + 2f'(0)h + 4\frac{f''(0)h^2}{2} + 8\frac{f'''(0)h^3}{6}. \end{aligned}$$

2. Sustitución en la regla: Sustituimos estos valores en la fórmula:

$$\int_0^{2h} f(x)\,dx = \frac{h}{15}\Big[7f(0) + 16f(h) + 7f(2h)\Big] + \frac{h^2}{15}\Big[f'(0) - f'(2h)\Big].$$

Verificamos que los términos coinciden con la expansión exacta de la integral.

14.3. Demostración con la regla de regla de Gauss-Legendre

La regla de integración es:

$$\int_{-1}^{1} f(x)\,dx = \frac{1}{4}\Big[3f\left(-\frac{2}{3}\right) + 2f(0) + 3f\left(\frac{2}{3}\right)\Big].$$

Demostración:

Esta es una regla de Gauss-Legendre para $n = 3$, por lo que los nodos y pesos se calculan para maximizar la precisión.

1. Nodos y pesos: Los nodos son:

$$x_1 = -\frac{2}{3}, \quad x_2 = 0, \quad x_3 = \frac{2}{3}.$$

Los pesos son:

$$w_1 = w_3 = \frac{3}{4}, \quad w_2 = \frac{2}{4}.$$

2. Verificación: La regla de Gauss-Legendre es exacta para polinomios de grado $2n - 1 = 5$. Sustituimos $f(x) = 1$, $f(x) = x$, $f(x) = x^2$, etc., y comprobamos que ambos lados de la fórmula coinciden.

14.4. Conclusión

Hemos demostrado diversas reglas de integración numérica, incluyendo métodos abiertos y cerrados, utilizando el método de coeficientes no determinados. Estas reglas son exactas para polinomios de ciertos grados y proporcionan una base teórica para su aplicación en cálculos numéricos precisos.

Capítulo 15

Implementación de la Regla Trapezoidal para Datos Tabulados

Fecha de la actividad: 2 de diciembre de 2024

La regla trapezoidal es un método numérico común para calcular la integral de una función conocida en un conjunto discreto de puntos. Este método aproxima la curva de la función mediante segmentos lineales entre los puntos tabulados y calcula el área bajo estas líneas.

La fórmula general para integrar en un intervalo $[a, b]$, donde se tienen valores tabulados $f(x_0), f(x_1), \ldots, f(x_n)$, es:

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left(f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right),$$

donde $h = \frac{b-a}{n}$ es el ancho de los subintervalos, y n es el número de subintervalos.

En este capítulo, se implementará este método para integrar datos tabulados generados a partir de una función arbitraria y se ilustrará su aplicación mediante un ejemplo práctico.

15.1. Implementación en Python

La función `regla_trapezoidal` implementa la regla trapezoidal para aproximar la integral de una función cuyos valores son conocidos únicamente en un conjunto discreto de puntos tabulados. A continuación, se describe cómo esta función utiliza la regla trapezoidal para calcular la integral:

Definición de la función

La función está definida como:

```
def regla_trapezoidal(x, y):
    """
    Calcula la integral aproximada usando la regla trapezoidal para datos tabulados.

    Parámetros:
        x (array): Valores de las coordenadas x (deben estar ordenados).
        y (array): Valores de las coordenadas y correspondientes a x.

    Retorna:
        float: Valor aproximado de la integral.
    """
```

Asociación con la regla trapezoidal

La regla trapezoidal para datos tabulados divide el intervalo total en subintervalos definidos por los puntos en el arreglo `x`. Para cada subintervalo $[x_{i-1}, x_i]$, calcula el área del trapecio correspondiente, que se expresa como:

$$\text{Área del trapecio} = \frac{h}{2}\left(f(x_{i-1}) + f(x_i)\right),$$

donde $h = x_i - x_{i-1}$ es el ancho del subintervalo.

La función itera sobre todos los subintervalos en los datos tabulados para acumular el área total. Esto se implementa como:

```
for i in range(1, n):
    h = x[i] - x[i-1]
    integral += (y[i-1] + y[i]) * h / 2
```

Validación de los datos

La función también verifica que los arreglos de entrada `x` e `y` sean válidos:

- Deben tener la misma longitud.
- Deben contener al menos dos puntos, ya que la regla trapezoidal requiere al menos un subintervalo.

Esto se verifica al inicio de la función:

```
if n < 2 or len(y) != n:
    raise ValueError("x e y deben tener el mismo número de puntos y al menos 2 valores.")
```

Resultado

Finalmente, la función retorna el valor de la integral aproximada como la suma de las áreas de todos los trapecios:

```
return integral
```

15.2. Ejemplo de uso

En esta sección, se utiliza la función `regla_trapezoidal` previamente explicada para calcular la integral aproximada de un conjunto de datos tabulados. Supongamos que los valores de `x` y `y` son los siguientes:

- `x = [0, 1, 2, 3]`, correspondientes a los puntos en el eje x.
- `y = [1, 2, 0, 3]`, correspondientes a los valores de la función evaluada en dichos puntos.

La rutina se implementa y evalúa como sigue:

```python
# Datos tabulados
x = np.array([0, 1, 2, 3])
y = np.array([1, 2, 0, 3])

# Calcular la integral aproximada usando la regla trapezoidal
resultado = regla_trapezoidal(x, y)

# Mostrar el resultado
print(f"Resultado de la integral: {resultado}")
```

Cálculo paso a paso

La función divide el intervalo $[0, 3]$ en subintervalos definidos por los puntos $x = [0, 1, 2, 3]$. Para cada subintervalo, calcula el área de los trapecios:

$$\text{Integral} \approx \frac{1}{2}(1+2)(1) + \frac{1}{2}(2+0)(1) + \frac{1}{2}(0+3)(1).$$

La suma de estas áreas es:

$$\text{Integral} \approx 1.5 + 1 + 1.5 = 4.0$$

15.3. Conclusión

La implementación de la regla trapezoidal demuestra ser una herramienta útil para integrar datos tabulados. En el ejemplo presentado, calculamos la integral aproximada de una función tabulada en puntos discretos, obteniendo un resultado de 4.0. Esta metodología es eficiente y versátil para problemas en los que la función no se conoce de manera analítica, pero se dispone de datos experimentales o simulados.

Capítulo 16

Análisis del Problema de Kepler y Solución Numérica con el Método del Salto de la Rana

Fecha de la actividad: 24 de noviembre de 2024

En este capitulo, analisaremos las trayectorias de un cometeta bajo la acción gravitacional del Sol. La ecuación de movimiento estará gobernada por la ley de gravitación universal. Normalizaremos esta ecuación y las condiciones iniciales. Implementando métodos numéricos para resolver la ecuación diferencial. Variaremos las condiciones iniciales y observaremos diferentes tipos de trayectorias para el cometa. Finalmente, nos concentremos en la trayectoria eliptica, calcularemos sus propiedades como elipse y verificaremos si en ella se cumplen las leyes de Kepler.

16.1. Normalización de la ecuación

La ecuación de movimiento para un cometa bajo la acción gravitacional del Sol está dada por:

$$\vec{r}''(t) = -\frac{GM}{|\vec{r}|^3}\vec{r}, \tag{16.1}$$

donde $\vec{r}$ es el vector posición del cometa, G es la constante gravitacional y M es la masa del Sol.

Las condiciones iniciales son:

$$\vec{r}(0) = r_0\hat{x}, \quad \vec{r}'(0) = v_0\hat{y}. \tag{16.2}$$

A continuación se efectúa un **análisis dimensional** a cada variable en la ecuación (16.1):

$$[t] = T, \quad [G] = \frac{L^3}{M \cdot T^2}, \quad [M] = M, \quad [\vec{r}] = L, \quad [\vec{r}''] = \frac{L}{T^2}$$

Notamos que (16.1) tiene $\mathbf{n} = 5$ variables, donde cada una de las cuales puede expresarse en función de $\mathbf{k} = 3$ magnitudes fundamentales:

- longitud (L)
- tiempo (T)
- masa (M)

Según el **Teorema de Pi**, la ecuación (16.1) puede reescribirse de forma equivalente como una ecuación que involucra **n - k** = 2 variables **normalizadas**, construidas a partir de las variables originales. Por lo tanto, introducimos las siguientes definiciones para las variables normalizadas de tiempo (T) y posición ($\vec{R}$):

$$T = t\sqrt{\frac{GM}{r_0^3}}, \quad \vec{R} = \frac{\vec{r_i}}{r_0} \tag{16.3}$$

Despejando t y $\vec{r}$ a partir de sus respectivas expresiones en (16.3), se obtiene:

$$t = T\sqrt{\frac{r_0^3}{GM}}, \quad \vec{r} = \vec{R}r_0 \tag{16.4}$$

A continuación, se realiza un cambio de variable en (16.1), sustituyendo en dicha ecuación los valores de t y $\vec{r}$ dados en (16.4). Considerando $\vec{r}'' = \frac{d^2\vec{r}}{dt^2}$, tenemos que:

$$\frac{d^2(\vec{R}r_0)}{d\left(T\sqrt{\frac{r_0^3}{GM}}\right)^2} = -\frac{GM}{|\vec{R}r_0|^3}\vec{R}r_0$$

Luego de desarrollar y simplificar términos, se obtiene la normalización de la ecuación (16.1):

$$\frac{d^2\vec{R}}{dT^2} = -\frac{\vec{R}}{|\vec{R}|^3} \tag{16.5}$$

Para **normalizar las condiciones iniciales**, se sustituyen en (16.2) los valores de t y $\vec{r}$ en (16.4).

$$\vec{R}(0)r_0 = r_0\hat{x}, \quad \frac{d\vec{R}(0)r_0}{d\left(T\sqrt{\frac{r_0^3}{GM}}\right)} = v_0\hat{y}.$$

Desarrollando estas ecuaciones, se obtienen las condiciones iniciales normalizadas:

$$\vec{R}(0) = \hat{x}, \quad \frac{d\vec{R}(0)}{dT} = \tilde{v}_0\hat{y}. \tag{16.6}$$

Donde $\tilde{v}_0 = v_0\sqrt{\frac{r_0}{GM}}$.

16.2. Resolución numérica mediante el método del salto de la rana

Empleamos el método del **salto de la rana** (*leapfrog*) para resolver numéricamente el problema de Kepler normalizado (16.5), y verificaremos la conservación de la energía y el momento angular en la solución del sistema .

Para esto implementaremos un programa en `Python`, en el cual importamos las siguientes librerías:

- `numpy`: Para cálculos matemáticos eficientes.
- `matplotlib.pyplot`: Para graficar las trayectorias, energía total y momento angular.
- `EcuacionesDiferencialesOrdinarias`: Este módulo contiene la implementación del método del *salto de la rana*, utilizado para resolver ecuaciones diferenciales acopladas. Navarro (2024)

Definimos una función en `a_kepler_normalizado` que represente la aceleración normalizada del cuerpo, sabemos que esta es dada por la ecuación (16.5). Esto de la forma:

```
def a_kepler_normalizado(r, t):
    norm_r3 = np.linalg.norm(r) ** 3
    return -r / norm_r3
```

Se establecen las condiciones iniciales normalizadas la de posición y velocidad para diferentes valores de $\tilde{v}_0$, según lo demostrado en (16.6). Los tiempos se definen en un rango $[0, t_{\max}]$ con un paso Δt pequeño. Esto de forma:

```
v0_tildes = [0.5, 1.0, 1.5]
t_max = 20
dt = 0.01
t = np.arange(0, t_max, dt)

for v0_tilde in v0_tildes:
    # Condiciones iniciales normalizadas
    r0 = np.array([1.0, 0.0])  # Inicialmente en el eje x
    v0 = np.array([0.0, v0_tilde])  # Velocidad inicial en dirección y
```

Utilizamos el método del *salto de la rana*, implementado en la función `SaltoRana`, para resolver las ecuaciones acopladas de posición y velocidad. La posición y velocidad se calculan iterativamente.

```
 r, v = SaltoRana(a_kepler_normalizado, r0, v0, t)

```

Después de obtener las soluciones, verificamos la validez de la simulación numérica calculando en cada instante la Energía total y el Momento angular del sistema, esto según las ecuaciones:

- **Energía total:**

$$E = \frac{1}{2}\|\vec{V}\|^2 - \frac{1}{\|\vec{R}\|}$$

  que incluye la energía cinética y potencial gravitacional.

- El **Momento angular:**

$$L = \vec{R} \times \vec{V},$$

  que es el producto cruz entre posición y velocidad.

Programamos esto de forma:

```
    energia = 0.5 * np.sum(v**2, axis=1) - 1.0 / np.linalg.norm(r, axis=1)
    momento_angular = np.cross(r, v)
```

Finalmente generamos tres gráficos:

1. Trayectoria (normalizada) en el plano (x, y) (Figura 16.1).
2. Conservación de la energía total a lo largo del tiempo (Figura 16.2).
3. Conservación del momento angular a lo largo del tiempo (Figura 16.3).

Cada gráfico permite visualizar el impacto de distintos valores de $\tilde{v}_0$.

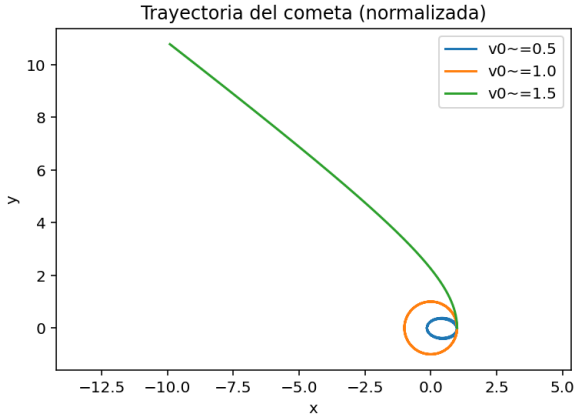

Figura 16.1: Trayectoria del cometa (normalizada)

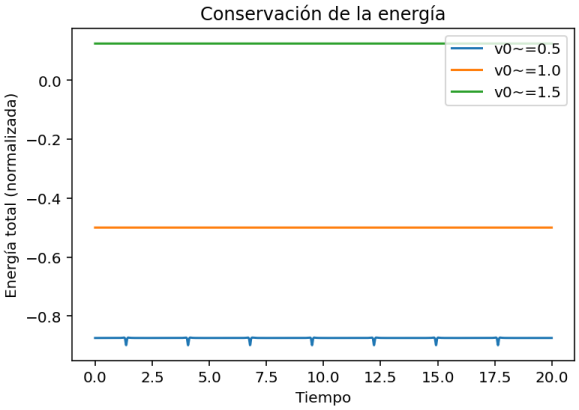

Figura 16.2: Conservación de la energía

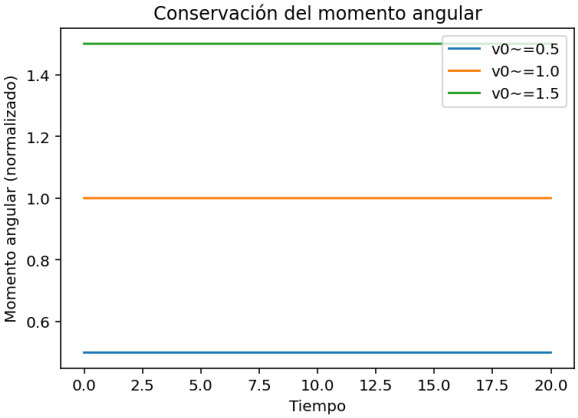

Figura 16.3: Conservación del momento angular

16.3. Implementación numérica

A partir de los resultados obtenidos con el método del salto de la rana, se procede a:

- Extraer $\vec{R}(t)$ y $\vec{V}(t)$ en intervalos de tiempo discretos.
- Determinar $R_{\max}$ y $r_{\min}$ directamente del conjunto de datos.
- Calcular los parámetros orbitales (a, b, e).
- Comparar las propiedades de la órbita obtenida con las leyes de Kepler.

```python
def calcular_parametros_orbitales(r):

    distancias = np.linalg.norm(r, axis=1)
    r_max = np.max(distancias)  # Distancia máxima
    r_min = np.min(distancias)  # Distancia mínima

    a = (r_max + r_min) / 2
    e = (r_max - r_min) / (r_max + r_min)

    b = a * np.sqrt(1 - e**2)

    return a, b, e

```

16.4. Análisis de la trayectoria elíptica

En el gráfico 16.1, vemos que para una velocidad inicial normalizada $\tilde{v} \cong 0.5$ el cometa tiene una trayectoria elíptica cerrada. Implementamos `python` para calcular los siguientes parámetros de esta trayectoria elíptica:

- **Semieje mayor (a)**: El semieje mayor se calcula como el promedio de la distancia máxima ($r_{\max}$) y mínima ($r_{\min}$):

$$a = \frac{R_{\max} + R_{\min}}{2}. \tag{16.7}$$

  Al calcular se obtuvo un semieje mayor $a = 0.5718$.

- **Semieje menor (b)**: El semieje menor está relacionado con el semieje mayor y la excentricidad (e) mediante:

$$b = a\sqrt{1 - e^2}. \tag{16.8}$$

  Al calcular se obtuvo un semieje menor $b = 0.3789$.

- **Excentricidad (e)**: La excentricidad se define como:

$$e = \frac{R_{\max} - R_{\min}}{R_{\max} + R_{\min}}. \tag{16.9}$$

  Al calcular se obtuvo una excentricidad $e = 0.7490$

16.5. Verificación de las leyes de Kepler

Se verifican las tres leyes de Kepler Venegas (2023) utilizando los resultados obtenidos a partir de la simulación numérica.

- **Primera ley de Kepler**: Para verificar esta ley, se comparan las posiciones (x, y) calculadas numéricamente con la ecuación de una elipse:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1, \tag{16.10}$$

  Al terminar la verificación se concluye que la primera ley no se cumple.

- **Segunda ley de Kepler**: Establece que el radio vector que une el cuerpo al centro de masas barre áreas iguales en intervalos de tiempo iguales.

  Numéricamente, se calcula el área barrida en un intervalo de tiempo Δt como:

$$\Delta A = \frac{1}{2}|\vec{r}(t) \times \vec{v}(t)|\Delta t, \tag{16.11}$$

  donde $\vec{R}(t)$ es la posición y $\vec{V}(t)$ la velocidad del cometa. Al terminar la verificación se concluye que la segunda ley si se cumple.

- **Tercera ley de Kepler**: Establece que el cuadrado del periodo orbital T es proporcional al cubo del semieje mayor a:

$$T^2 \propto a^3. \tag{16.12}$$

  El periodo orbital $T = 1.26$ se determina como el tiempo necesario para completar una revolución, y el semieje mayor fue calculado, obteniendo $a = 0.5718$. Se calcula la relación $\frac{T^2}{a^3}$ y se obtiene un valor constante de 8.4934. Por lo tanto se cumple la tercera ley de Kepler.

16.6. Conclusión

La normalización de la ecuación de movimiento del cometa, junto con las condiciones iniciales, nos proporciona un sistema más manejable desde el punto de vista numérico y conceptual. La ecuación resultante, que depende únicamente de parámetros adimensionales, es universal y aplicable a cualquier problema similar mediante un simple reescalado. Esto sienta las bases para su resolución numérica y para el análisis detallado de las trayectorias orbitales y su comparación con las leyes de Kepler.

Capítulo 17

Ecuaciones Diferenciales Ordinarias: Solución Analítica y Método de Euler

Fecha de la actividad: 25 de noviembre de 2024

En este capítulo se analiza un problema con valor inicial dado por la ecuación diferencial

$$y' = -3y\sin(t), \quad y(0) = \frac{1}{2}, \tag{17.1}$$

encontrando su solución analítica obtenida por el método de **separación de variables** y estimando una solución numérica aproximada mediante el **método de Euler**.

Se compararan las soluciones obtenidas en términos de precisión, analizando las diferencias para diferentes particiones del intervalo $0 \leq t \leq 4\pi$, con $N = 2^8, 2^9, 2^{10}, 2^{11}$ subintervalos. Finalmente, se comentan las ventajas y limitaciones del método numérico para problemas de este tipo.

17.1. Solución Analítica

En la ecuación diferencial (17.1) , considerando $y' = \frac{dy}{dt}$, y multiplicando la ecuación $\frac{dt}{y}$, notamos que se puede escribir en forma separable:

$$\frac{dy}{y} = -3\sin(t)\,dt.$$

Integrando sobre ambos miembros, obtenemos:

$$\ln|y| = 3\cos(t) + C, \tag{17.2}$$

donde C es una constante de integración. Despejamos y aplicando la exponencial sobre la ecuación (17.2), obteniendo:

$$y(t) = e^{3\cos(t)}e^{C}. \tag{17.3}$$

Aplicando la condición inicial $y(0) = \frac{1}{2}$, se encuentra que

$$\frac{1}{2} = e^{3}e^{C}$$

Aplicando el logaritmo natural sobre la ecuación, y despeja C, se obtiene:

$$C = \ln\left(\frac{1}{2}\right) - 3 \tag{17.4}$$

Reemplazando (17.4) en (17.3), y desarrollando, se obtiene la solución analitica de (17.1).

$$y(t) = \frac{1}{2e^3} e^{3\cos(t)}.$$

17.2. Solución Numérica: Método de Euler

Se para Resolvemos numéricamente la ecuación diferencial (17.1), implementando `python` y utilizando el modulo `EcuacionesDiferencialesOrdinarias` Navarro (2024). La aproximación para $y(t)$ en este caso se calcula mediante:

$$y_{n+1} = y_n + \Delta t f(t_n, y_n),$$

donde

- $f(t, y) = -3y\sin(t)$,
- $t_n = n\Delta t$
- $\Delta t = (t_{\text{máx}} - t_{\text{mín}})/N$.

Se resuelve el problema para $N = 2^8, 2^9, 2^{10}, 2^{11}$, y se compara con la solución analítica.

```
def f(y, t):
    return -3 * y * np.sin(t)
y0 = 1 / 2

t_min, t_max = 0, 4 * np.pi

N_values = [2**8, 2**9, 2**10, 2**11]

for N in N_values:
    t = np.linspace(t_min, t_max, N + 1)
    y = Euler(f, r0=y0, t=t)
```

17.3. Resultados y Comparación

En la Figura 17.1 se muestran las soluciones obtenidas para cada valor de N, junto con la solución analítica. Se observa que conforme aumenta N, las soluciones numéricas convergen hacia la solución analítica, mostrando la dependencia del método de Euler en el tamaño del paso Δt.

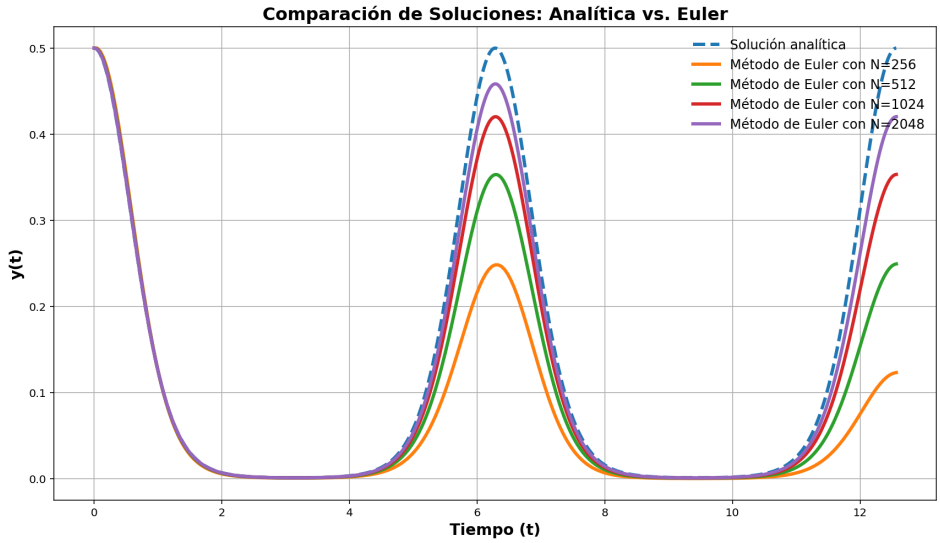

Figura 17.1: Comparación entre la solución analítica y las soluciones numéricas para diferentes valores de N.

17.4. Conclusiones

En este capítulo se analizó la solución de una ecuación diferencial ordinaria mediante enfoques analítico y numérico. Se observó que, aunque el método de Euler es sencillo y efectivo, su precisión es limitada, especialmente con pasos grandes o intervalos largos. Se destaca la importancia de comparar las soluciones analíticas y numéricas para verificar la exactitud en problemas con valor inicial.

Capítulo 18

Demostración del Error en el Método de Runge-Kutta de Cuarto Orden

Fecha de la actividad: 28 de noviembre de 2024

El método de Runge-Kutta de cuarto orden es un método numérico utilizado para resolver ecuaciones diferenciales de la forma:

$$\vec{x}'(t) = \vec{f}(\vec{x}), \tag{18.1}$$

cuyo proceso iterativo se define mediante:

$$\vec{K}_1 = \Delta t\, \vec{f}(\vec{x}_n), \tag{18.2}$$

$$\vec{K}_2 = \Delta t\, \vec{f}\left(\vec{x}_n + \frac{1}{2}\vec{K}_1\right), \tag{18.3}$$

$$\vec{K}_3 = \Delta t\, \vec{f}\left(\vec{x}_n + \frac{1}{2}\vec{K}_2\right), \tag{18.4}$$

$$\vec{K}_4 = \Delta t\, \vec{f}(\vec{x}_n + \vec{K}_3), \tag{18.5}$$

$$\vec{x}_{n+1} = \vec{x}_n + \frac{1}{6}\left(\vec{K}_1 + 2\vec{K}_2 + 2\vec{K}_3 + \vec{K}_4\right). \tag{18.6}$$

En esta sección, demostraremos que su error local es del orden $O(\Delta t^5)$.

18.1. Demostración del Orden de Error

Expandimos $\vec{x}(t + \Delta t)$ en serie de Taylor:

$$\vec{x}(t + \Delta t) = \vec{x}(t) + \Delta t\vec{x}'(t) + \frac{\Delta t^2}{2!}\vec{x}''(t) + \frac{\Delta t^3}{3!}\vec{x}'''(t) + O(\Delta t^4).$$

El análisis muestra que las fórmulas de $\vec{K}_1$, $\vec{K}_2$, $\vec{K}_3$ y $\vec{K}_4$ eliminan los términos de orden $\mathcal{O}(\Delta t^2)$, $\mathcal{O}(\Delta t^3)$ y $\mathcal{O}(\Delta t^4)$, confirmando el orden del error local.

18.2. Demostración del Orden de Error

Comenzamos expandiendo $\vec{x}(t + \Delta t)$ en una serie de Taylor en torno a t:

$$\vec{x}(t + \Delta t) = \vec{x}(t) + \Delta t\vec{x}'(t) + \frac{\Delta t^2}{2!}\vec{x}''(t) + \frac{\Delta t^3}{3!}\vec{x}'''(t) + \frac{\Delta t^4}{4!}\vec{x}^{(4)}(t) + O(\Delta t^5). \tag{18.7}$$

La ecuación diferencial (18.1) nos permite expresar las derivadas superiores de $\vec{x}(t)$ en términos de $\vec{f}$ y sus derivadas parciales:

$$\begin{aligned}
\vec{x}'(t) &= \vec{f}(\vec{x}), \\
\vec{x}''(t) &= \frac{\partial \vec{f}}{\partial \vec{x}} \vec{f}, \\
\vec{x}'''(t) &= \frac{\partial^2 \vec{f}}{\partial \vec{x}^2}(\vec{f}, \vec{f}) + \frac{\partial \vec{f}}{\partial \vec{x}} \frac{\partial \vec{f}}{\partial \vec{x}} \vec{f}, \\
\vec{x}^{(4)}(t) &= \dots .
\end{aligned}$$

Paso 2: Expansión del Método de Runge-Kutta de Cuarto Orden

Consideramos el valor $\vec{x}_{n+1}$ dado por el método de Runge-Kutta es:

$$\vec{x}_{n+1} = \vec{x}_n + \frac{\Delta t}{6} \left(K_1 + 2K_2 + 2K_3 + K_4 \right)$$

Sustituyendo las definiciones de K_1, K_2, K_3, K_4:

$$\vec{x}_{n+1} = \vec{x}_n + \frac{\Delta t}{6} \left(\vec{f}(\vec{x}_n) + 2\, \vec{f}(\vec{x}_n + \frac{1}{2}\Delta t\, \vec{f}(\vec{x}_n)) + 2\, \vec{f}(\vec{x}_n + \frac{1}{2}\Delta t\, \vec{f}(\vec{x}_n)) + \vec{f}(\vec{x}_n + \Delta t\, \vec{f}(\vec{x}_n)) \right)$$

Esta expresión se puede expandir en una serie de Taylor alrededor de t_n, con términos adicionales debido a los desplazamientos Δt y las evaluaciones de $\vec{f}$. Tras hacer estas expansiones, el valor obtenido estará en forma:

$$\vec{x}_{n+1} = \vec{x}_n + \Delta t\, \vec{f}(\vec{x}_n) + \frac{\Delta t^3}{24}\, \vec{f}'''(\vec{x}_n) + O(\Delta t^5)$$

Expandimos cada término de $\vec{K}_i$ en serie de Taylor en función de Δt:

$$\begin{aligned}
\vec{K}_1 &= \Delta t \vec{f}(\vec{x}_n), \\
\vec{K}_2 &= \Delta t \left[\vec{f}(\vec{x}_n) + \frac{\Delta t}{2} \frac{\partial \vec{f}}{\partial \vec{x}} \vec{f} + O(\Delta t^2) \right], \\
\vec{K}_3 &= \Delta t \left[\vec{f}(\vec{x}_n) + \frac{\Delta t}{2} \frac{\partial \vec{f}}{\partial \vec{x}} \vec{f} + O(\Delta t^2) \right], \\
\vec{K}_4 &= \Delta t \left[\vec{f}(\vec{x}_n) + \Delta t \frac{\partial \vec{f}}{\partial \vec{x}} \vec{f} + O(\Delta t^2) \right].
\end{aligned}$$

Sustituyendo estas expresiones en la fórmula del método de Runge-Kutta y agrupando términos, verificamos que las contribuciones de orden $O(\Delta t^2)$, $O(\Delta t^3)$ y $O(\Delta t^4)$ se cancelan debido a los coeficientes específicos del método. El término dominante restante es de orden $O(\Delta t^5)$.

Por lo tanto, el error local del método es:

$$E_{\text{local}} = O(\Delta t^5). \tag{18.8}$$

18.3. Resultados Numéricos

Para validar el análisis teórico, se resuelve la ecuación $\frac{dx}{dt} = -x$ con condición inicial $x(0) = 1$. Se compara la solución exacta $x(t) = e^{-t}$ con la solución numérica del método de Runge-Kutta para particiones $N = 10, 20, 40, 80$.

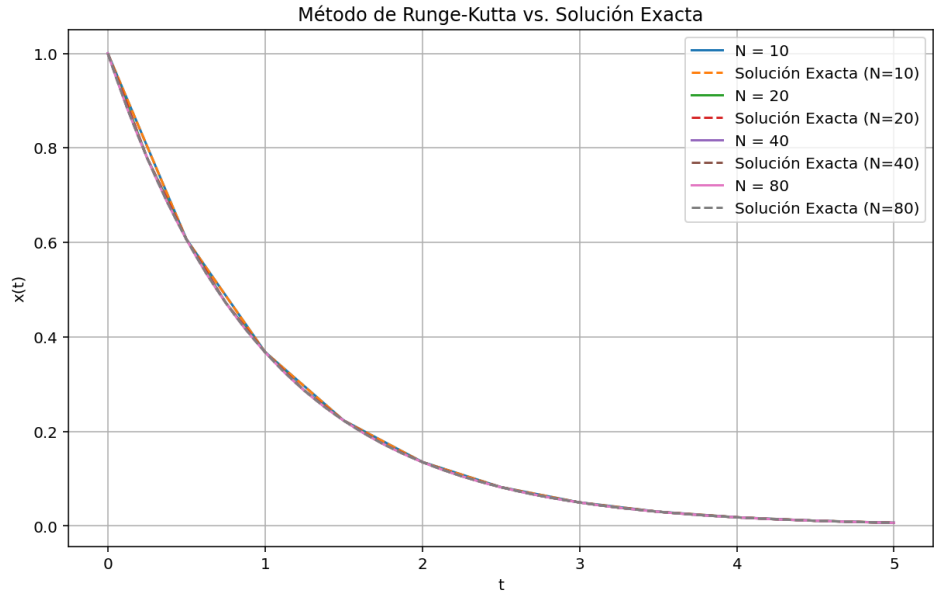

Figura 18.1: Comparación entre la solución exacta y la solución numérica obtenida con el método de Runge-Kutta para varios valores de N.

18.4. Conclusiones

El método de Runge-Kutta de cuarto orden ofrece una solución precisa, con un error local del orden $O(\Delta t^5)$. Tanto analítica como numéricamente, hemos visto su utilidad para resolver ecuaciones diferenciales con alta precisión y eficiencia.

Capítulo 19

Resolución de la Ecuación de Bessel de Primer Tipo y Orden Cero

Fecha de la actividad: 27 de noviembre de 2024

La ecuación diferencial de Bessel de primer tipo y orden cero está dada por:

$$x^2 \frac{d^2 J_0}{dx^2} + x \frac{dJ_0}{dx} + x^2 J_0 = 0. \tag{19.1}$$

Este capítulo aborda la resolución numérica de esta ecuación mediante el método de Runge-Kutta de orden 4 y la verificación de su solución integral:

$$J_0(x) = \frac{1}{\pi} \int_0^{\pi} \cos(x \sin \phi)\, d\phi.$$

19.1. Resolución Numérica

Comenzamos reescribiendo la ecuación diferencial (19.1) como un sistema de ecuaciones de primer orden:

$$y_1 = J_0, \quad y_2 = \frac{dJ_0}{dx}, \quad \frac{dy_1}{dx} = y_2, \quad \frac{dy_2}{dx} = -\frac{1}{x} y_2 - y_1 \quad .$$

Definimos en `python` una función que represente a este sistema:

```python
def edo_bessel(y, x):
    y1, y2 = y
    dy1dx = y2
    dy2dx = -y2 / x - y1
    return np.array([dy1dx, dy2dx])
```

Definimos el intervalo donde se resolvió el sistema, $0 < x \leq 20$ con condiciones iniciales $J_0(0) = 1$, $J_0'(0) = 0$, evaluadas en $x = \epsilon = 10^{-6}$:

```python
x_min, x_max = 1e-6, 20
pasos = 1000
x_intervalo = np.linspace(x_min, x_max, pasos)
condiciones_iniciales = np.array([1, 0])
```

Resolvemos numéricamente el sistema utilizando el método de Runge-Kutta de orden 4, importado la función `RungeKutta4` desde el modulo `EcuacionesDiferencialesOrdinarias.py`.. Se presenta la gráfica de esta solución en la figura 19.1.

```
y_rk4 = RungeKutta4(edo_bessel, r0=condiciones_iniciales, t=x_intervalo)
J0_rk4 = y_rk4[:, 0]
```

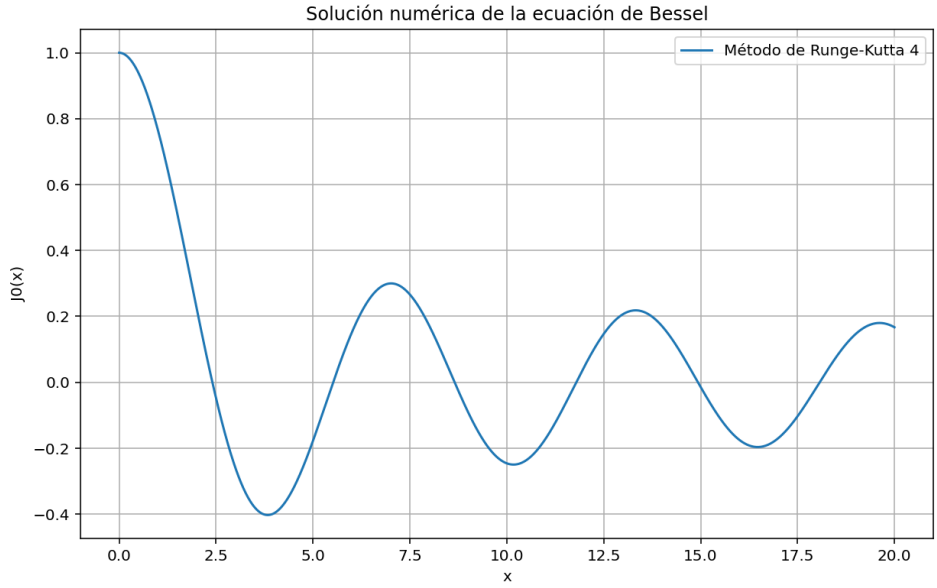

Figura 19.1: Solucion de Runge-Kutta.

19.2. Solución Integral

La solución integral se calculó mediante la regla trapezoidal, discretizando ϕ en 1000 puntos uniformes en $[0, \pi]$. Esto fue calculado con la siguiente función en `python`:

```
def J0_integral(x):
    phi = np.linspace(0, np.pi, 1000)  # Discretización de phi
    dphi = phi[1] - phi[0]
    integral = np.trapz(np.cos(x[:, None] * np.sin(phi)), dx=dphi, axis=1)
    return integral / np.pi
```

Se presenta la gráfica de esta solución en la figura 19.2.

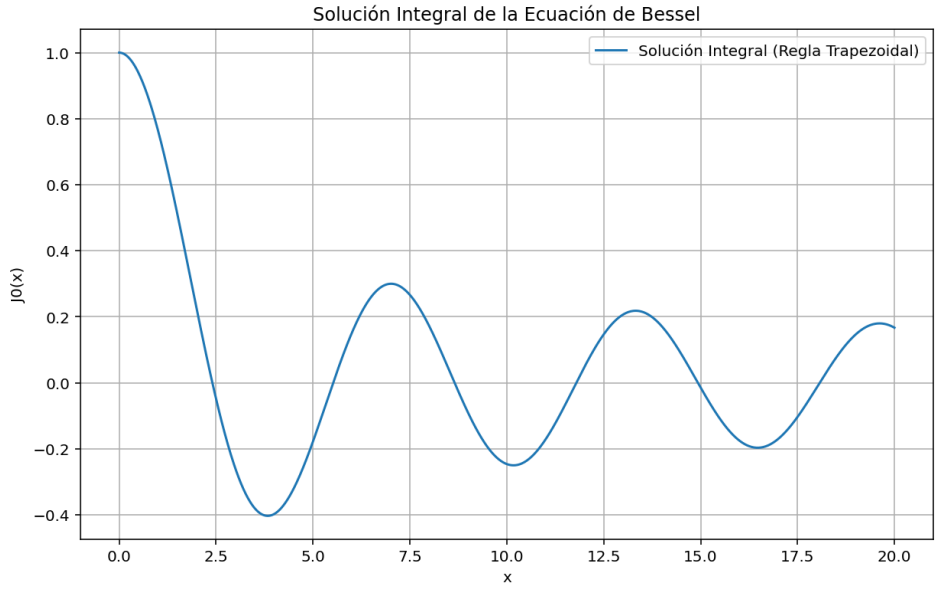

Figura 19.2: Soluciones de la solución integral.

19.3. Error Relativo

El error relativo entre las soluciones numérica e integral se calculara como:

$$\text{Error Relativo} = \frac{|J_0^{\text{RK}}(x) - J_0^{\text{Int}}(x)|}{|J_0^{\text{Int}}(x)|}.$$

Se muestra la gráfica del error relativo en la figura 19.3.

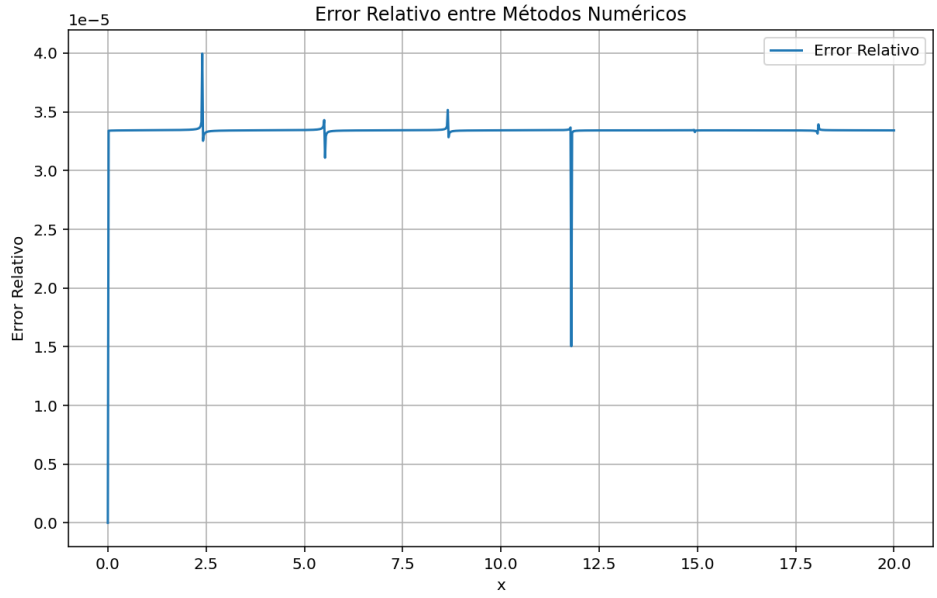

Figura 19.3: Error relativo entre las soluciones numérica e integral.

19.4. Conclusiones

La resolución numérica de la ecuación de Bessel mediante el método de Runge-Kutta de cuarto orden y la verificación integral muestran una excelente concordancia en el intervalo $0 < x \leq 20$. La baja magnitud del error relativo confirma la validez de ambas soluciones y destaca la precisión del método numérico frente a la solución integral calculada con la regla trapezoidal.

Este trabajo permitió profundizar en la implementación de métodos numéricos como Runge-Kutta y el uso de técnicas integrales para verificar soluciones de ecuaciones diferenciales.

Capítulo 20

Cálculo de los Polinomios de Hermite y sus Ceros

Fecha de la actividad: 26 de noviembre de 2024

En este capítulo:

- Calculamos los primeros cinco polinomios de Hermite (H_1 a H_5) utilizando derivadas numéricas centradas.
- Encontramos los ceros de estos polinomios en el rango $-3 < x < 3$.

Polinomios de Hermite

Los polinomios de Hermite son definidos por la relación de recurrencia:

$$H_{n+1}(x) = 2xH_n(x) - H'_n(x),$$

donde $H'_n(x)$ es la derivada del polinomio $H_n(x)$. Con la condición inicial:

$$H_0(x) = 1,$$

se puede construir la familia completa de polinomios.

20.1. Derivada Numérica Centrada

La derivada centrada para una función $f(x)$ en un punto x_i se calcula como:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{2h},$$

donde h es el tamaño del paso entre los puntos de la discretización.

20.2. Resultados

Usando la relación de recurrencia y la derivada centrada, se obtuvieron los primeros cinco polinomios de Hermite. En la Figura 20.1, se grafican estos polinomios, y se marcan los ceros correspondientes en el rango $-3 < x < 3$.

Figura 20.1: Primeros cinco polinomios de Hermite y sus ceros.

En la Tabla 20.1, se presentan los ceros encontrados para cada polinomio.

Polinomio	Ceros
$H_1(x)$	$x = 0$
$H_2(x)$	$x = \pm 0.707$
$H_3(x)$	$x = 0, \pm 1.225$
$H_4(x)$	$x = \pm 0.524, \pm 1.651$
$H_5(x)$	$x = 0, \pm 0.959, \pm 2.020$

Cuadro 20.1: Ceros de los primeros cinco polinomios de Hermite.

20.3. Conclusiones

Los primeros cinco polinomios de Hermite fueron calculados utilizando derivadas numéricas centradas. Los ceros fueron encontrados en el rango $-3 < x < 3$, y los resultados coinciden con los valores esperados teóricos para estos polinomios. Estos resultados son fundamentales para aplicaciones físicas, como la descripción de los estados del oscilador armónico cuántico.

Capítulo 21

Métodos Iterativos para la Búsqueda de Ceros

Fecha de la actividad: 28 de noviembre de 2024

El problema consiste en encontrar una solución de la ecuación $f(x^*) = 0$ usando métodos iterativos basados en una expansión en series de Taylor de $f(x)$. Este capítulo aborda la derivación de un método iterativo con corrección de segundo orden y su comparación con el método de Newton-Raphson.

21.1. Derivación del Método Iterativo

A partir de la expansión en series de Taylor de $f(x)$ y la solución de la ecuación cuadrática resultante, obtenemos el método iterativo:

$$x_{n+1} = x_n - \frac{2f(x_n)}{f'(x_n)} \pm \frac{\sqrt{(f'(x_n))^2 - 2f(x_n)f''(x_n)}}{f'(x_n)}.$$

El signo $\pm$ se elige de manera que el denominador sea positivo y de mayor magnitud, lo que evita oscilaciones y asegura la estabilidad del método.

21.2. Equivalencia con el Método de Newton-Raphson

El método de Newton-Raphson regular se define como:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

El método iterativo derivado en la sección anterior es una extensión del método de Newton-Raphson al considerar la segunda derivada $f''(x_n)$, lo que mejora la precisión en la aproximación de la raíz.

21.3. Convergencia Cúbica

El método iterativo convergerá cúbicamente a la raíz x^* si la raíz tiene multiplicidad simple, es decir, si $f'(x^*) \neq 0$. Esto significa que el error disminuye de forma proporcional al cubo del error anterior, lo que hace que el método sea más eficiente que el método de Newton-Raphson, que tiene convergencia cuadrática.

21.4. Conclusiones

El método derivado en este capítulo es más eficiente que el método de Newton-Raphson bajo ciertas condiciones, especialmente cuando $f''(x_n)$ es pequeño en comparación con $f'(x_n)$. Este método tiene la ventaja de converger más rápido en algunos casos, especialmente cuando la función $f(x)$ tiene una segunda derivada significativa.

Capítulo 22

Solución de Ecuaciones

Fecha de la actividad: 27 de noviembre de 2024

En esta sección, determinaremos numéricamente las soluciones de las siguientes ecuaciones:

1. **Ecuación 1**: $x^3 - 21x^2 + 120x - 100 = 0$
2. **Ecuación 2**: $x = 2^{-x}$
3. **Ecuación 3**: $\tan x = \frac{1}{x}$

Para ello, implementaremos el método de Newton-Raphson, un procedimiento iterativo diseñado para encontrar las raíces de una función $f(x)$. A partir de un valor inicial o semilla x_0, la fórmula de recurrencia está dada por:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \tag{22.1}$$

donde $f'(x)$ es la derivada de $f(x)$.

Nota: No estudiaremos la multiplicidad de las soluciones.

22.1. Definimos funciones y sus derivadas para cada ecuación

1. **Ecuación 1**: $x^3 - 21x^2 + 120x - 100 = 0$.

$$f(x) = x^3 - 21x^2 + 120x - 100$$

$$f'(x) = 3x^2 - 42x + 120$$

2. **Ecuación 2**: $x = 2^{-x}$. Igualando a 0.

$$g(x) = x - 2^{-x}$$

$$g'(x) = 1 + \ln(x) \cdot 2^{-x}$$

3. **Ecuación 3**: $\tan x = \frac{1}{x}$. Igualando a 0.

$$h(x) = \tan x - \frac{1}{x}$$

$$h'(x) = \sec^2 x + \frac{1}{x^2}$$

22.2. Implementación del método de Newton-Raphson

Se implementa el método de Newton-Raphson, utilizando la siguiente función programada en `Python`.

```python
def newton_raphson(f, df, x0, tolerancia=1e-6, iteracion_max=1000):
    x = x0
    iteraciones = 0
    while abs(f(x)) > tolerancia and iteraciones < iteracion_max:
        h = f(x) / df(x)
        x = x - h
        iteraciones += 1
    if abs(f(x)) <= tolerancia:
        return x
    else:
        return None
```

Finalmente, se obtienen las soluciones a cada ecuación:

1. **Soluciones ecuación 1**: $x_0 = 1$, $x_1 = 10$
2. **Soluciones ecuación 2**: $x_0 \approx 0.64$
3. **Soluciones ecuación 3**: $x_0 \approx 0.86$, $x_1 \approx 3.42$

22.3. Conclusión

En este capítulo, utilizamos el método de Newton-Raphson para encontrar numéricamente las soluciones de las ecuaciones planteadas en el problema.

Conclusiones

Fecha de presentación: Viernes 29 de noviembre de 2024

Capítulo 23

Conclusión del portafolio

23.1. Resumen de los objetivos del portafolio

El portafolio tuvo como objetivo principal documentar y reflejar el aprendizaje adquirido a lo largo del curso mediante la recopilación de evidencias significativas. Este instrumento permitió analizar de forma crítica los conceptos aprendidos, identificar fortalezas y áreas de mejora, así como reflexionar sobre la aplicación futura de los conocimientos adquiridos.

23.2. Resumen de los contenidos

En este portafolio se incluyen diversas evidencias de aprendizaje, cada una asociada a actividades realizadas durante el curso. Se implemento python para resolver numéricamente problemas matemáticos y físicos complejos. Se llevaron a cabo demostraciones matemáticas cuando fue solicitado.

23.3. Autoevaluación del alumno/a

Durante el curso, considero que mi desempeño fue mediocre. A futuro, me gustaría enfocarme más en distribuir mejor mi tiempo y profundizar en la comprensión teórica antes de abordar la implementación práctica. Los contenidos de este portafolio son aplicables a futuros proyectos, especialmente aquellos que involucren simulaciones numéricas y análisis de datos.

23.4. Evaluación del curso

Al inicio del curso, no esperaba aprender a resolver problemas numéricos y modelar sistemas dinámicos. Las actividades del portafolio me permitieron introducirme a técnicas avanzadas y herramientas de programación que desconocía.

Aspectos positivos del curso incluyen la integración de conceptos teóricos y prácticos, así como el enfoque en la validación de resultados. Sin embargo, considero que podría mejorarse la claridad en algunos materiales de referencia y en las instrucciones de las actividades.

En cuanto a las evidencias incluidas, creo que la simulación de la trayectoria del cometa fue la más relevante, ya que integró conocimientos de programación, física y matemáticas de forma aplicada. Esto la diferencia de otras actividades que fueron más específicas y conceptuales.

Finalmente, considero que el portafolio es una herramienta útil no solo para evaluar el aprendizaje, sino también para reflexionar y consolidar los conocimientos adquiridos.

23.5. Sugerencias para futuras versiones del curso

- Incluir más ejemplos prácticos en los materiales de referencia.
- Proveer guías detalladas para la interpretación de resultados numéricos.
- Fomentar el trabajo colaborativo en las actividades del portafolio.

Bibliografía

Fernández, J. L., https://www.fisicalab.com/apartado/potencial-electrico-punto (2024), [Online; accessed 29-October-2024].

GeeksforGeeks,, "Python infinity," (2024), accessed: 2024-11-20.

MathWorks,, "https://es.mathworks.com/help/matlab/ref/meshgrid.html" (2024), [Online; accessed 30-October-2024].

Matplotlib,, "Annotating plots," https://matplotlib.org/stable/gallery/text_labels_and_annotations/annotation_demo.html (2024), [Online; accessed 26-October-2024].

Navarro, R., "Ecuaciones diferenciales ordinarias en python," (2024), disponible en el repositorio de GitHub como parte de las utilidades para Física Computacional II, Universidad de Concepción.

Venegas, S., "Las leyes de kepler: qué son, historia y ejemplos," (2023), accedido el 25 de noviembre de 2024. Recurso educativo sobre las tres leyes de Kepler, su historia y aplicaciones prácticas.

Wikipedia,, "Diferencia finita — wikipedia, la enciclopedia libre," (2024), [En línea; accedido el 4 de noviembre de 2024].