De la Teoría al Código

Implementación real y lógica aplicada de los métodos numéricos

Roberto A. Ticona Miramira
Clyde N. Paricahua Pari
Leydy G. Aguilar Ccopa

Universidad Nacional del Altiplano

UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE ING. ESTADÍSTICA E INFORMÁTICA

ESCUELA PROFESIONAL DE ING. ESTADÍSTICA E INFORMÁTICA

PROGRAMACIÓN NUMÉRICA

DOCENTE:

Dr. TORRES CRUZ FRED

ALUMNOS:

- AGUILAR CCOPA LEYDY GRISELDA
- PARICAHUA PARI CLIDE NEIL
- TICONA MIRAMIRA ROBERTO ANGEL

SECCIÓN:

"IV SEMESTRE - A"

PUNO - PERÚ

2025

Introducción

La programación numérica es una rama fundamental dentro de las ciencias computacionales y aplicadas, cuyo objetivo es proporcionar métodos y herramientas para resolver problemas matemáticos mediante el uso de algoritmos y programas informáticos. A través de ella, es posible aproximar soluciones a ecuaciones que no pueden resolverse de forma analítica, optimizar funciones complejas, y analizar sistemas dinámicos en diversas áreas del conocimiento.

El propósito de este libro es introducir al lector en los principios y aplicaciones prácticas de la programación numérica, utilizando lenguajes de programación de propósito científico como `Python` y `R`. A lo largo de los capítulos, se desarrollan los fundamentos teóricos y computacionales necesarios para comprender los métodos numéricos más empleados en ingeniería, física, economía y nutrición, entre otros campos.

En la primera unidad, se abordarán los conceptos básicos de la programación numérica, la definición de funciones matemáticas y sus restricciones, así como los métodos clásicos para encontrar raíces de ecuaciones no lineales, tales como el método de bisección, el método de Newton-Raphson y el método de la secante. Estos métodos serán implementados paso a paso en código, permitiendo observar su comportamiento, eficiencia y convergencia.

En la segunda unidad, se explorarán métodos más avanzados, entre ellos el gradiente descendente y sus aplicaciones en optimización, la interpolación polinómica como técnica para aproximar funciones a partir de datos discretos, y la diferenciación numérica, que permite estimar derivadas de funciones cuando no se dispone de una expresión analítica. Cada tema será acompañado de ejemplos prácticos y ejercicios que ayudarán a reforzar la comprensión teórica mediante la experimentación computacional.

Además, se presentarán recomendaciones sobre buenas prácticas en la escritura de código científico, el uso eficiente de librerías numéricas, y la validación de resultados mediante análisis de error. El enfoque de este libro combina la precisión matemática con la aplicación práctica, fomentando el desarrollo de habilidades analíticas y computacionales en el lector.

En conjunto, este material busca no solo enseñar los fundamentos de la programación numérica, sino también promover un pensamiento crítico y estructurado al enfrentar problemas reales que requieren soluciones computacionales.

Índice general

Índice de figuras

Parte I

Unidad I

1 Programación Numérica

La programación numérica es una disciplina que combina las matemáticas aplicadas y la informática con el objetivo de resolver problemas cuantitativos mediante métodos computacionales. Se centra en el diseño, análisis e implementación de algoritmos que permiten obtener soluciones aproximadas a ecuaciones, sistemas y modelos que, en la mayoría de los casos, no pueden resolverse de forma analítica (Burden et al., 2016; Chapra & Canale, 2015).

A diferencia de la programación convencional, que busca desarrollar aplicaciones funcionales o sistemas de información, la programación numérica se orienta a la resolución eficiente y precisa de problemas matemáticos. Entre sus principales aplicaciones se encuentran la simulación de fenómenos físicos y biológicos, la modelización económica, la ingeniería de datos, la inteligencia artificial y el análisis estadístico en ciencias de la salud (Press et al., 2007).

El objetivo fundamental de esta área es transformar problemas continuos en representaciones discretas que puedan ser tratadas por un computador. De esta forma, se logra aproximar soluciones a problemas de optimización, integración, derivación, interpolación, ajuste de curvas y resolución de ecuaciones diferenciales.

En términos prácticos, la programación numérica permite al investigador o profesional:

- Resolver ecuaciones no lineales mediante métodos iterativos.
- Aproximar derivadas e integrales de funciones cuando no se dispone de una forma analítica.
- Interpolar o ajustar funciones a datos experimentales.
- Optimizar funciones de una o varias variables bajo restricciones.
- Analizar errores y estimar la estabilidad numérica de los métodos empleados.

Actualmente, lenguajes como `Python`, `R`, `MATLAB` y `Julia` ofrecen bibliotecas especializadas que facilitan el desarrollo de algoritmos numéricos de alto rendimiento. Estos entornos han hecho posible que la programación numérica sea una herramienta accesible y poderosa para la investigación científica, la ingeniería y la docencia (Chapra & Canale, 2015).

En síntesis, la programación numérica constituye una base esencial para la solución computacional de problemas científicos y técnicos, integrando el razonamiento matemático con la capacidad de cómputo moderna.

1.1 Ejemplo

Utilizando la programación numérica desarrollar una aplicación que permita analizar operaciones matemáticas (Identificador de funciones).

```python
import re

def analizar_funcion(funcion):
funcion = funcion.replace(" ", "")
variables = sorted(set(re.findall(r'[a-zA-Z]', funcion)))
terminos = re.findall(r'[+-]?\d*[a-zA-Z]?\^?\d*', funcion)
terminos = [t for t in terminos if t]
coeficientes = []
signos = []
for t in terminos:
if t.startswith('-'):
signos.append('-')
else:
signos.append('+')
c = re.findall(r'[+-]?\d+', t)
if c:
coef = int(c[0])
else:
coef = -1 if t.startswith('-') else 1
coeficientes.append(coef)
print(f"\n   Función ingresada: {funcion}")
print(f"   Variables encontradas: {variables}")
print(f"   Términos: {terminos}")
print(f"   Coeficientes: {coeficientes}")
print(f"   Signos: {signos}")

print("=== ANALIZADOR DE FUNCIONES ALGEBRAICAS ===")
funcion = input("   Ingresa una función (ejemplo: 3x^2 - 4x + 7): ")
analizar_funcion(funcion)
```

```
=== ANALIZADOR DE FUNCIONES ALGEBRAICAS ===
   Ingresa una función (ejemplo: 3x^2 - 4x + 7): 3x^2 - 4x + 7

   Función ingresada: 3x^2-4x+7
   Variables encontradas: ['x']
   Términos: ['3x^2', '-4x', '+7']
   Coeficientes: [3, -4, 7]
   Signos: ['+', '-', '+']
```

Figura 1.1: Analizador de funciones

Explicación

El problema se resolvió utilizando el lenguaje de programación Python y el módulo `re` (expresiones regulares), que permite analizar y extraer patrones dentro de una cadena de texto.

1. Limpieza de la función

```python
funcion = funcion.replace(" ", "")
```

Se eliminan los espacios en blanco para facilitar el análisis de la expresión algebraica.

2. Identificación de variables

```
variables = sorted(set(re.findall(r'[a-zA-Z]', funcion)))
```

- Se usa una expresión regular para encontrar todas las letras.
- Se eliminan duplicados con `set`.
- Se ordenan alfabéticamente.

3. Separación de términos algebraicos

```
terminos = re.findall(r'[+-]?\d*[a-zA-Z]?\^?\d*', funcion)
terminos = [t for t in terminos if t]
```

Se extraen los términos considerando:

- Signo opcional (+ o -)
- Coeficiente numérico
- Variable
- Potencia (si existe)

4. Obtención de coeficientes y signos

```
for t in terminos:
if t.startswith('-'):
signos.append('-')
else:
signos.append('+')
```

Se determina el signo de cada término.

```
c = re.findall(r'[+-]?\d+', t)
if c:
coef = int(c[0])
else:
coef = -1 if t.startswith('-') else 1
```

- Si hay número explícito, se toma como coeficiente.

- Si no hay número, se asume `1` o `-1`.

5. Presentación de resultados

```
print(f"   Función ingresada: {funcion}")
print(f"   Variables encontradas: {variables}")
print(f"   Términos: {terminos}")
print(f"   Coeficientes: {coeficientes}")
print(f"   Signos: {signos}")
```

Objetivo del código

El objetivo principal de este programa es analizar una función algebraica ingresada por el usuario y extraer sus componentes fundamentales, específicamente:

- Identificar las variables presentes en la función.
- Separar la función en términos algebraicos.
- Determinar los coeficientes numéricos de cada término.
- Reconocer el signo (positivo o negativo) de cada término.

Este análisis es útil como paso previo en aplicaciones como:

- Simplificación de expresiones algebraicas.
- Derivación e integración simbólica.
- Clasificación de polinomios.
- Desarrollo de software educativo matemático.

Código en Python usando Tkinter

```
import re
import tkinter as tk
from tkinter import messagebox

def analizar_funcion():
funcion = entrada_funcion.get().replace(" ", "")

if not funcion:
messagebox.showwarning("Advertencia", "Ingrese una función algebraica")
return

variables = sorted(set(re.findall(r'[a-zA-Z]', funcion)))
```

```
terminos = re.findall(r'[+-]?\d*[a-zA-Z]?\^?\d*', funcion)
terminos = [t for t in terminos if t]

coeficientes = []
signos = []

for t in terminos:
if t.startswith('-'):
signos.append('-')
else:
signos.append('+')

c = re.findall(r'[+-]?\d+', t)
if c:
coef = int(c[0])
else:
coef = -1 if t.startswith('-') else 1

coeficientes.append(coef)

salida.config(state="normal")
salida.delete("1.0", tk.END)
salida.insert(tk.END, f"   Función ingresada: {funcion}\n")
salida.insert(tk.END, f"   Variables encontradas: {variables}\n")
salida.insert(tk.END, f"   Términos: {terminos}\n")
salida.insert(tk.END, f"   Coeficientes: {coeficientes}\n")
salida.insert(tk.END, f"   Signos: {signos}\n")
salida.config(state="disabled")

ventana = tk.Tk()
ventana.title("Analizador de Funciones Algebraicas")
ventana.geometry("500x400")
ventana.resizable(False, False)

titulo = tk.Label(
ventana,
text="ANALIZADOR DE FUNCIONES ALGEBRAICAS",
font=("Arial", 14, "bold")
)
titulo.pack(pady=10)

label_funcion = tk.Label(
ventana,
text="Ingrese la función (ejemplo: 3x^2 - 4x + 7):"
)
label_funcion.pack()

entrada_funcion = tk.Entry(
ventana,
width=40,
font=("Arial", 12)
)
```

```
entrada_funcion.pack(pady=5)

boton = tk.Button(
ventana,
text="Analizar función",
font=("Arial", 11),
command=analizar_funcion
)
boton.pack(pady=10)

salida = tk.Text(
ventana,
height=10,
width=55,
state="disabled",
font=("Consolas", 10)
)
salida.pack(pady=10)

ventana.mainloop()
```

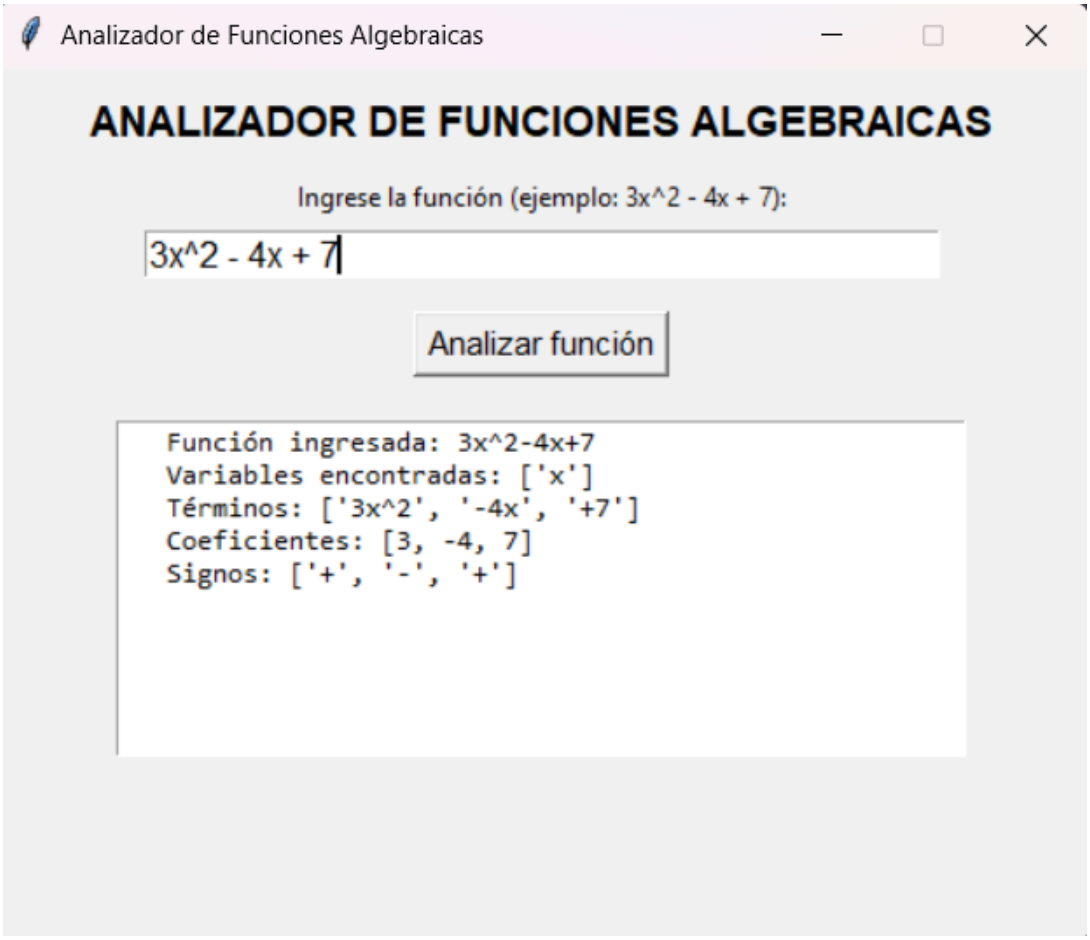


Figura 1.2: Analizador de funciones con Tkinter

2 Funciones

2.1 Funciones a nuestro alrededor

En casi todos los fenómenos físicos observamos que una cantidad depende de otra. Por ejemplo, la estatura de una persona depende de su edad, la temperatura de la fecha, el costo de enviar un paquete por correo depende de su peso. Usamos el término función para describir esta dependencia de una cantidad con respecto a otra. Esto es, decimos lo siguiente: (Stewart et al., 2001)

- La estatura es una función de la edad.
- La temperatura es una función de la fecha.
- El costo de enviar un paquete por correo depende de su peso.

2.2 Definición de función

Una función f es una regla que asigna a cada elemento x de un conjunto A exactamente un elemento, llamado $f(x)$, de un conjunto B.

Para hablar de una función, es necesario darle un nombre. Usaremos letras como $f, g, h, ...$ para representar funciones. Por ejemplo, podemos usar la letra f para representar una regla como sigue:

"f" es la regla "elevar al cuadrado el número"

cuando escribimos $f(2)$ queremos decir "aplicar la regla f al número 2". La aplicación de la regla da $f(2) = 2^2 = 4$. Del mismo modo, $f(3) = 3^2 = 9, f(4) = 4^2 = 16$, y en general $f(x) = x^2$. (Stewart et al., 2001)

Por lo general consideramos funciones para las cuales los conjuntos A y B son conjuntos de número reales. El símbolo $f(x)$ se lee "f de x" o "f en x" y se denomina valor de f en x, o la imagen de x bajo f. El conjunto A recibe el nombre de dominio de la función. El rango de f es el conjunto de todos los valores posibles de $f(x)$ cuando x varía en todo el dominio. El símbolo que representa un número arbitrario del dominio de una función f se llama variable independiente. El símbolo que representa un número en el rango de f se llama variable dependiente. Por tanto, si escribimos $y = f(x)$, entonces x es la variable independiente y y es la variable dependiente. (Stewart et al., 2001)

Es útil considerar una función como una máquina. Si x está en el dominio de la función f, entonces cuando x entra a la máquina, es aceptada como entrada y la máquina produce una salida $f(x)$ de acuerdo con la regla de la función. Así, podemos considerar el dominio como el conjunto de todas las posibles entradas y el rango como el conjunto de todas las posibles salidas. (Stewart et al., 2001)

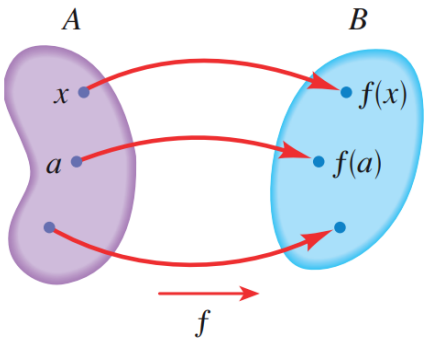


Figura 2.1: Diagrama de flechas de f

2.3 Cuatro formas de representar una función

Para entender mejor lo que es una función, podemos describir una función específica en las siguientes cuatro formas: (Stewart et al., 2016)

- verbalmente (por descripción en palabras)
- algebraicamente (por una fórmula explícita)
- visualmente (por una gráfica)
- numéricamente (por una tabla de valores)

Una función individual puede estar representada en las cuatro formas, y con frecuencia es útil pasar de una representación a otra para adquirir más conocimientos sobre la función. No obstante, ciertas funciones se describen en forma más natural por medio de un método que por los otros. Un ejemplo de una descripción verbal es la siguiente regla para convertir entres escalas de temperatura: (Stewart et al., 2016)

"Para hallar el equivalente Fahrenheit de una temperatura Celsius, multiplicar por $\frac{9}{5}$ la temperatura Celsius y luego sumar 32"

Verbal

Con palabras:

$P(t)$ es la "población del mundo en el instante t"

Relación de la población P y el tiempo t

Algebraica

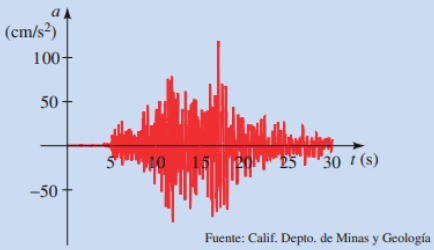
Por medio de una fórmula:

$$A(r) = \pi r^2$$

Área de un círculo

Visual

Por medio de una gráfica:



Aceleración vertical durante un terremoto

Numérica

Por medio de una tabla de valores:

w (onzas)	$C(w)$ (dólares)
$0 < w \leq 1$	0.37
$1 < w \leq 2$	0.60
$2 < w \leq 3$	0.83
$3 < w \leq 4$	1.06
$4 < w \leq 5$	1.29
⋮	⋮

Costo de enviar una carta por correo de primera clase

Figura 2.2: Cuatro formas de representar una función

2.4 Gráficas de funciones

2.4.1 Gráficas de funciones por localización de puntos

Para graficar una función f localizamos los puntos $(x, f(x))$ en un plano de coordenadas. En otras palabras, localizamos los puntos (x, y) cuya coordenada x es una entrada y cuya coordenada y es la correspondiente salida de la función. (Stewart et al., 2016)

Si f es una función con dominio A, entonces la gráfica de f es el conjunto de pares ordenados

$$(x, f(x))|x \in A$$

localizados en un plano de coordendas. En otras palabras, la gráfica de f es el conjunto de todos los puntos (x, y) tales que $y = f(x)$; esto es, la gráfica de f es la gráfica de la ecuación $y = f(x)$.

La gráfica de una función f da un retrato del comportamiento o "historia de la vida" de la función. Podemos leer el valor de $f(x)$ a partir de la gráfica como la altura de la gráfica arriba del punto x. (Stewart et al., 2016)

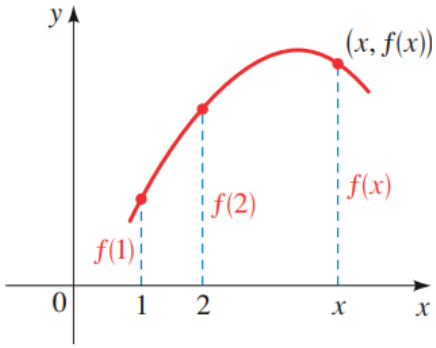


Figura 2.3: La altura de la gráfica arriba del punto x es el valor de $f(x)$

Una función f de la forma $f(x) = mx + b$ se denomina función lineal porque su gráfica es la gráfica de la ecuación $y = mx + b$, que representa una recta con pendiente m y punto de intersección b en y. Un caso especial de una función lineal se presenta cuando la pendiente es m = 0. La función $f(x) = b$, donde b es un número determinado, recibe el nombre de función constante porque todos sus valores son el mismo número, es decir, b. Su gráfica es la recta horizontal $y = b$. (Stewart et al., 2016)

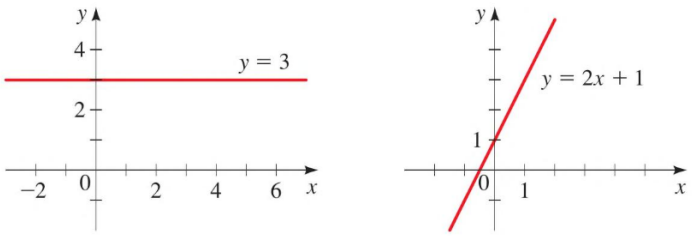


Figura 2.4: Funciones lineal y constante

2.4.2 La prueba de la recta vertical

La gráfica de una función es una curva en el plano xy. Pero surge la pregunta. ¿Cuáles curvas del plano xy son gráficas de funciones? Esto se contesta por medio de la prueba siguiente. (Stewart et al., 2016)

> Una curva en el plano de coordenadas es la gráfica de una función si y sólo si ninguna recta vertical cruza la curva más de una vez.

Podemos ver la Figura 2.5 para entender por qué la Prueba de la Recta Vertical es verdadera. Si cada recta vertical $x = a$ cruza la curva sólo una vez en (a, b), entonces exactamente un valor funcional está definido por $f(a) = b$. Pero si una recta $x = a$ cruza la curva dos veces, en (a, b) y en (a, c), entonces la curva no puede representar una función porque una función no puede asignar dos valores diferentes a a.

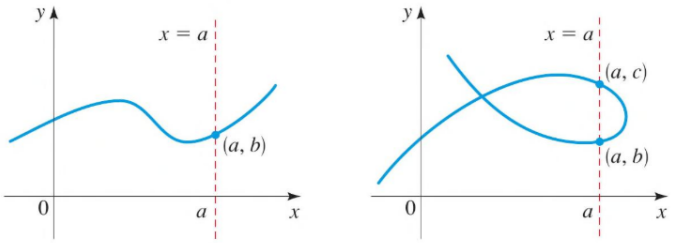


Figura 2.5: Prueba de la recta vertical

2.5 Aplicación 1

Se presentará un código en Python que sea capaz de graficar funciones, según los datos de entrada que se pidan.

```python
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

# === Ingreso de la función por el usuario ===
expr_str = input("Ingrese la función en términos de x (ejemplo: sin(x), x**2 + 3*x - 5, exp(-x)*cos(x)): ")

# === Definición de variable simbólica ===
x = sp.Symbol('x')

# === Conversión del texto a expresión simbólica ===
try:
expr = sp.sympify(expr_str)
except sp.SympifyError:
print("Error: la función ingresada no es válida.")
exit()

# === Creación de función numérica evaluable ===
f = sp.lambdify(x, expr, modules=['numpy'])

# === Intervalo de graficación ===
x_vals = np.linspace(-10, 10, 400)
y_vals = f(x_vals)

# === Graficar ===
plt.figure(figsize=(7,5))
plt.plot(x_vals, y_vals, label=f"$f(x) = {sp.latex(expr)}$", color='navy')
plt.title("Gráfica de la función ingresada", fontsize=13)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.grid(True, linestyle='--', alpha=0.6)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.legend()
plt.show()
```

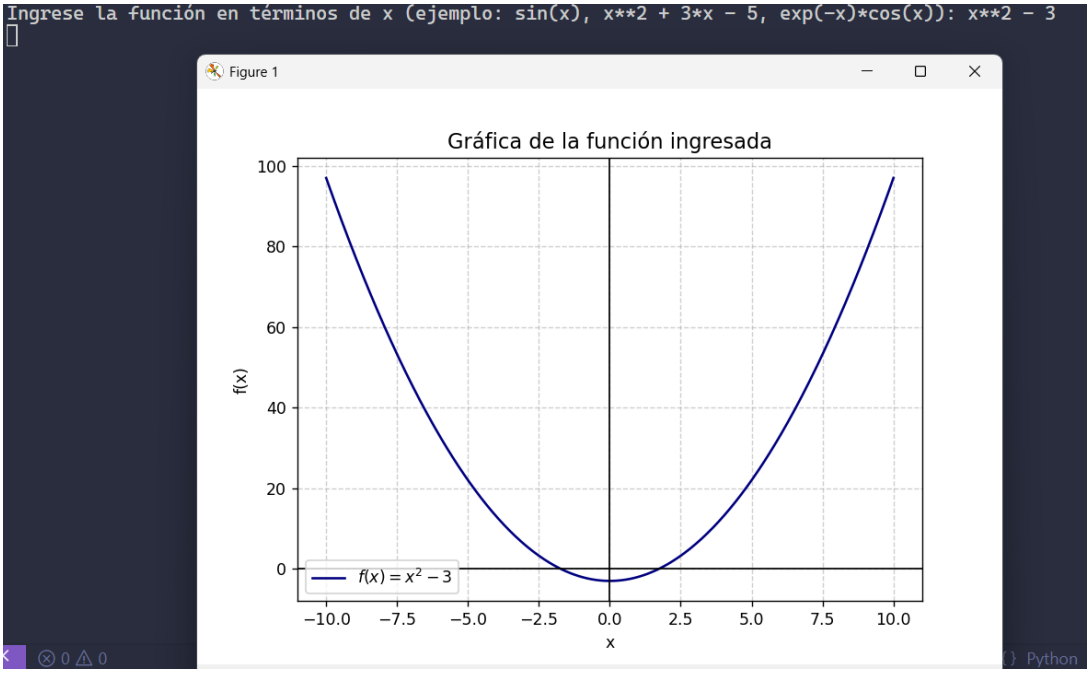


Figura 2.6: Graficar funciones

2.6 Aplicación 2

Se presentará un código en Python que sea capaz de graficar 2 funciones lineales.

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

x = np.linspace(-10, 10, 100)

y1 = 2 * x + 3
y2 = -1 * x + 5

plt.figure(figsize=(8, 6))
plt.plot(x, y1, label=' f(x) = 2x + 3', color='blue', linewidth=2)
plt.plot(x, y2, label=' f(x) = -x + 5', color='red', linewidth=2)

plt.title('Gráfico de dos funciones lineales')
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)

plt.show()
```

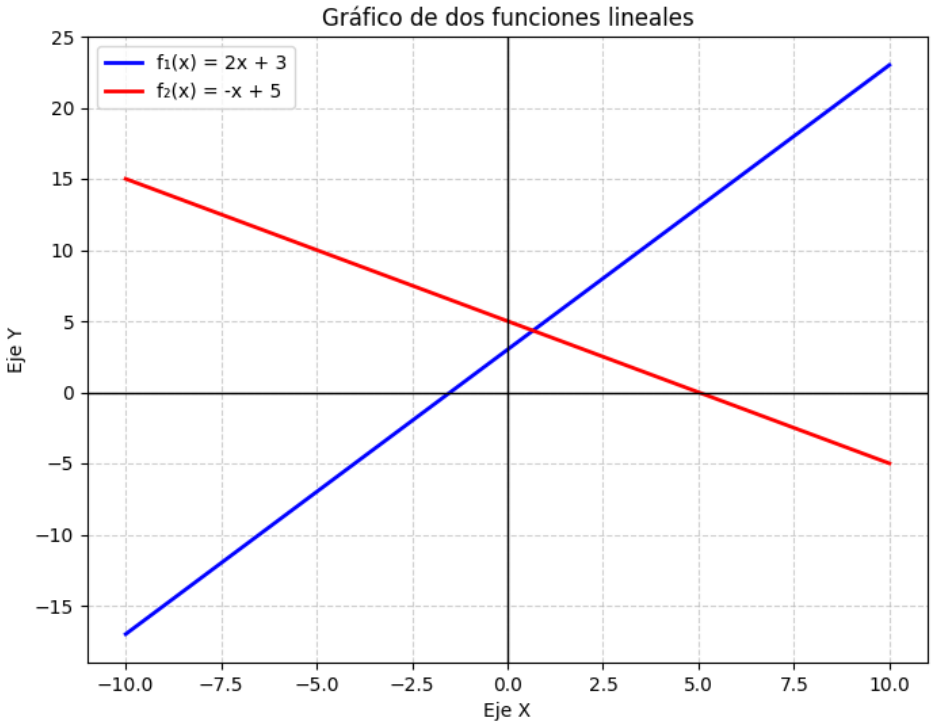


Figura 2.7: Graficar 2 funciones lineales

Explicación

1. Importación de librerías

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

NumPy se utiliza para crear arreglos numéricos y manejar datos de forma eficiente. Matplotlib permite generar gráficos científicos en dos dimensiones.

2. Definición del rango de valores

```
x = np.linspace(-10, 10, 100)
```

Se genera un arreglo de 100 valores equidistantes en el intervalo $[-10, 10]$. Estos valores representan el dominio sobre el cual se evaluarán las funciones.

3. Definición de las funciones lineales

Se definen dos funciones lineales:

- $f_1(x) = 2x + 3$: pendiente positiva, función creciente.

- $f_2(x) = -x + 5$: pendiente negativa, función decreciente.

Cada función se evalúa para todos los valores del arreglo x.

4. Creación del gráfico

```
plt.figure(figsize=(8, 6))
plt.plot(x, y1, label='f (x) = 2x + 3', color='blue', linewidth=2)
plt.plot(x, y2, label='f (x) = -x + 5', color='red', linewidth=2)
```

Se crea una figura y se grafican ambas funciones con diferentes colores y etiquetas para distinguirlas visualmente.

5. Personalización del gráfico

```
plt.title('Gráfico de dos funciones lineales')
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.axhline(0)
plt.axvline(0)
```

6. Visualización final

```
plt.show()
```

Objetivo del código

El objetivo del programa es representar gráficamente funciones lineales para analizar su comportamiento, permitiendo observar la pendiente, el crecimiento o decrecimiento y la intersección con los ejes coordenados.

Código en Python usando Tkinter

```
import tkinter as tk
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

def graficar():
x = np.linspace(-10, 10, 100)
y1 = 2 * x + 3
y2 = -1 * x + 5

fig, ax = plt.subplots(figsize=(6, 4))
```

```
ax.plot(x, y1, label='f(x) = 2x + 3', linewidth=2)
ax.plot(x, y2, label='f(x) = -x + 5', linewidth=2)

ax.set_title("Gráfico de dos funciones lineales")
ax.set_xlabel("Eje X")
ax.set_ylabel("Eje Y")
ax.axhline(0)
ax.axvline(0)
ax.grid(True)
ax.legend()

canvas = FigureCanvasTkAgg(fig, master=ventana)
canvas.draw()
canvas.get_tk_widget().pack(pady=10)

ventana = tk.Tk()
ventana.title("Gráfico de funciones lineales")
ventana.geometry("700x500")

titulo = tk.Label(
ventana,
text="GRÁFICO DE FUNCIONES LINEALES",
font=("Arial", 14, "bold")
)
titulo.pack(pady=10)

boton = tk.Button(
ventana,
text="Mostrar gráfica",
font=("Arial", 11),
command=graficar
)
boton.pack(pady=5)

ventana.mainloop()
```

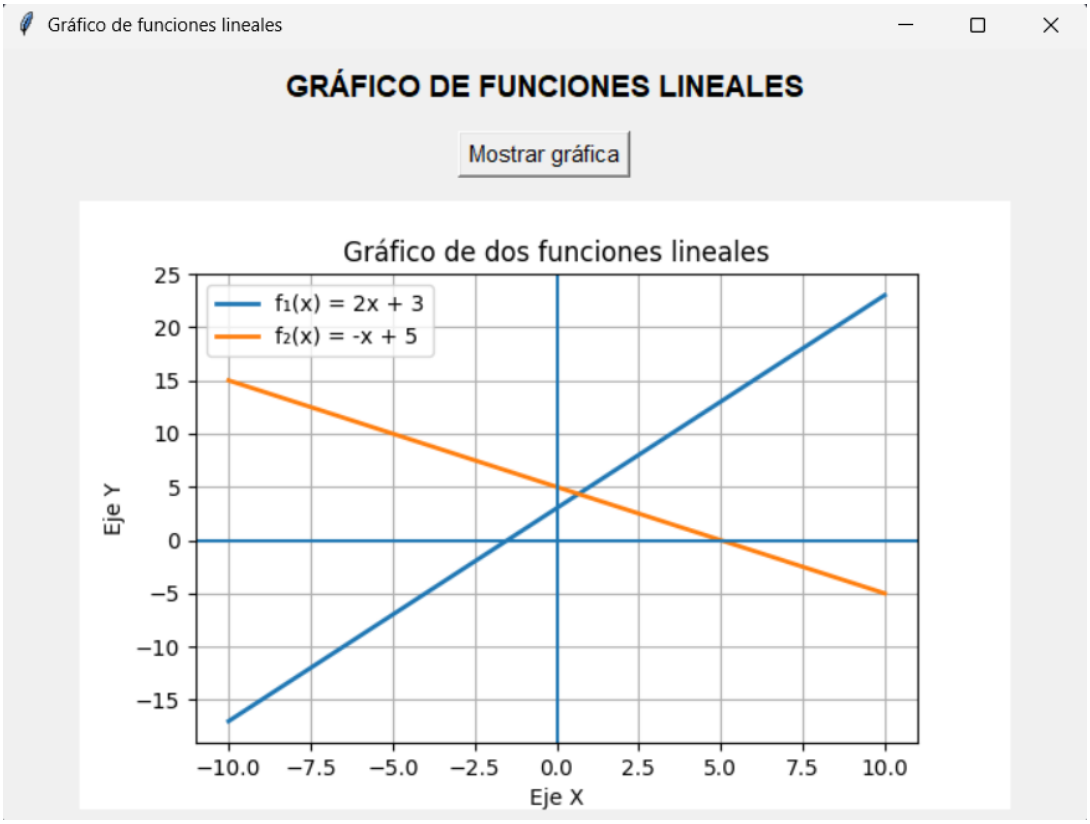


Figura 2.8: Graficador de 2 funciones lineales con Tkinter

3 Restricciones

Las restricciones son condiciones que limitan el conjunto de valores posibles que pueden tomar las variables en un problema matemático. Estas limitaciones definen el espacio factible o región factible, dentro del cual se busca una o más soluciones que cumplan con determinados criterios. Las restricciones pueden ser igualdades (por ejemplo, $x + y = 10$) o desigualdades (por ejemplo, $x \geq 0$), y desempeñan un papel fundamental en los problemas de optimización, programación lineal y análisis numérico. (Chapra & Canale, 2015)

Un sistema de ecuaciones consiste en un conjunto de ecuaciones que comparten las mismas variables. Resolverlo implica encontrar los valores que satisfacen todas las ecuaciones simultáneamente. Estos sistemas pueden clasificarse en lineales y no lineales, y sus métodos de resolución varían desde técnicas algebraicas clásicas (como la sustitución o igualación) hasta procedimientos numéricos avanzados (como el método de Gauss-Seidel o Newton-Raphson). (Chapra & Canale, 2015)

En el contexto de la programación numérica, las restricciones se representan y manipulan mediante código, permitiendo no solo resolver el sistema de ecuaciones sino también visualizar gráficamente la región factible y el punto óptimo. Esta representación es especialmente útil para comprender la interacción entre las ecuaciones y las limitaciones impuestas, facilitando el análisis y la toma de decisiones en problemas aplicados de ingeniería, economía y ciencias computacionales. (Chapra & Canale, 2015)

3.1 Ejemplo 1

Un desarrollador tiene 15 horas semanales para dedicar al desarrollo de software de front-end (x) y back-end (y). Además:

- Debe dedicar al menos 5 horas al desarrollo de front-end para cumplir con los entregables del cliente.
- El tiempo total no puede exceder 15 horas por restricciones de tiempo del sprint.

Formule las restricciones, represéntelas gráficamente e identifique las combinaciones posibles de tiempo a invertir en cada actividad.

Variables

- x = horas dedicadas al front-end
- y = horas dedicadas al back-end

Restricciones

1. Debe dedicar al menos 5 horas al mes.
   - $x \geq 5$
2. El tiempo total no puede exceder a 15 horas.
   - $x + y \leq 15$

Gráfico generado con python

Figura 3.1: Aplicación de restricciones 1 en Python

Interpretación del gráfico

En la representación gráfica se observa la región factible determinada por las restricciones planteadas:

- $x \geq 5$: el desarrollador debe dedicar al menos 5 horas al front-end. Esta condición aparece como una línea vertical en $x = 5$ y la región válida se encuentra a la derecha de ella.
- $y \geq 0$: el tiempo dedicado al back-end no puede ser negativo, por lo que la región se limita a la parte superior del eje x.
- $x + y \leq 15$: la suma de horas asignadas a front-end y back-end no puede superar las 15 horas semanales. Gráficamente, corresponde a la semirrecta bajo la línea $x + y = 15$.

La intersección de estas tres restricciones genera una región triangular factible delimitada por los puntos $(5, 0)$, $(15, 0)$ y $(5, 10)$. Esto significa que cualquier combinación de horas ubicada dentro o sobre este triángulo cumple con las condiciones del problema. Por ejemplo, el desarrollador podría dedicar:

- 5 horas a front-end y 10 horas a back-end,
- 10 horas a front-end y 5 horas a back-end,
- o bien 15 horas únicamente a front-end.

En conclusión, la región sombreada representa todas las combinaciones posibles de tiempo de trabajo entre front-end y back-end que respetan tanto el mínimo requerido en front-end como la restricción máxima de 15 horas semanales.

Código en Python

```python
# Función para preparar expresiones lineales
def preparar_expresion(expr: str) -> str:
expr = expr.replace(" ", "")          # quitar espacios
expr = expr.replace("^", "**")        # potencia
expr = expr.replace("-x", "-1*x")     # caso -x
expr = expr.replace("+x", "+1*x")     # caso +x
if expr.startswith("x"):              # si empieza con x
expr = "1*" + expr
expr = expr.replace("x", "*x")        # poner multiplicación
expr = expr.replace("**x", "*x")      # corregir si se duplicó
return expr

# Restricciones:
# 1) x = 5  (vertical)
# 2) x + y = 15  ->  y = -x + 15
func2 = preparar_expresion("-x+15")

# Rango de la gráfica
xmin, xmax = -5, 20
ymin, ymax = -5, 20

# Recorremos el plano
for y in range(ymax, ymin - 1, -1):
```

```
linea = ""
for x in range(xmin, xmax + 1):
# Recta 1: x = 5
cond1 = (x == 5)

# Recta 2: y = -x + 15
try:
y2 = eval(func2)
except:
y2 = None
cond2 = (y2 is not None and abs(y - y2) < 0.5)

# Región factible: x>=5, y>=0, x+y<=15
region = (x >= 5 and y >= 0 and x + y <= 15)

# Qué dibujar
if cond1 and cond2:
linea += "#"
elif cond1:
linea += "*"
elif cond2:
linea += "o"
elif x == 0 and y == 0:
linea += "+"
elif x == 0:
linea += "|"
elif y == 0:
linea += "-"
elif region:
linea += "."
else:
linea += " "
print(linea)

# Leyenda
print("\nLeyenda del gráfico:")
print("  * = x = 5")
print("  o = x + y = 15")
print("  # = Intersección")
print("  . = Región factible")
print("  | = Eje Y")
print("  - = Eje X")
print("  + = Origen (0,0)")
```

3.2 Ejemplo 2

Un ingeniero de datos administra dos tipos de servidores en la nube: Servidores A y Servidores B. El costo por hora de Servidor A es S/3 y de Servidor B es S/5. El presupuesto máximo semanal asignado para mantener los servidores es de S/20. Determine cúantas horas puede mantener activos cada tipo de servidor, formule el sistema de ecuaciones y represéntelo gráficamente.

Variables

- x = horas de uso de Servidor A
- y = horas de uso de Servidor B

Restricción

1. Se sabe que la hora de Servidor A cuesta S/3, cada hora de Servidor B cuesta S/5 y el presupuesto máximo es S/20.
   - $3x + 5y \leq 20$

Sistema de ecuaciones lineales

$$\begin{cases} 3x + 5y \leq 20 \\ x \geq 0 \\ y \geq 0 \end{cases}$$

Gráfico generado en Python

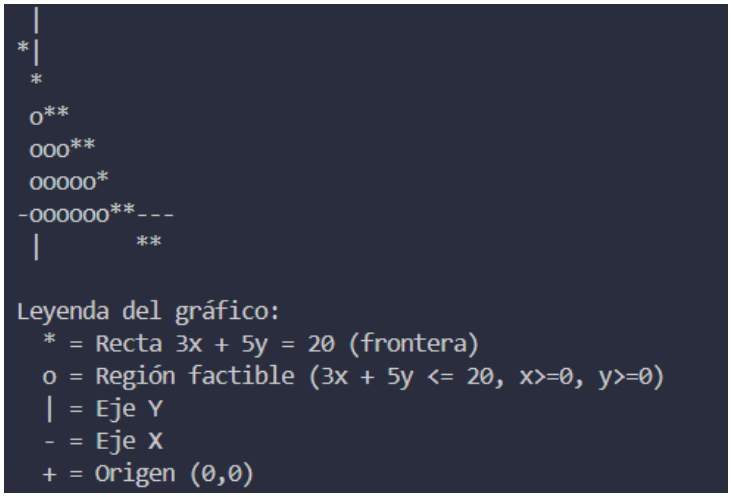


Figura 3.2: Aplicación de restricciones 2 en Python

Interpretación del gráfico

La recta límite está dada por $3x + 5y = 20$. Al interceptar con los ejes coordenados se obtiene que, si $x = 0$, entonces $y = 4$, y si $y = 0$, entonces $x = \frac{20}{3} \approx 6.67$.

La región factible corresponde al triángulo delimitado por los puntos $(0, 0)$, $(6.67, 0)$ y $(0, 4)$. Cualquier combinación de horas (x, y) dentro de esta región satisface las condiciones de costo máximo.

En conclusión, el ingeniero de datos puede asignar:

- hasta 6 horas con 40 minutos únicamente a servidores A,
- hasta 4 horas únicamente a servidores B,
- o bien cualquier combinación intermedia que cumpla con el presupuesto, como por ejemplo $(x = 2, y = 2)$ o $(x = 3, y = 1)$.

Código en Python

```
# Restricción principal: 3x + 5y <= 20  →  y <= (20 - 3x)/5
func = "(20 - 3*x)/5"

# Definimos el rango del gráfico (valores de X y Y)
xmin, xmax = -1, 10     # un poco más para ver bien
ymin, ymax = -1, 6

# Recorremos los valores de Y de arriba hacia abajo
for y in range(ymax, ymin - 1, -1):
linea = ""

for x in range(xmin, xmax + 1):
try:
y_line = eval(func)
except:
y_line = None

# Condición para la recta
cond_line = (y_line is not None and abs(y - y_line) < 0.5)

# Condición para la región factible: debajo de la recta y en el primer cuadrante
cond_region = (y_line is not None and y <= y_line and x >= 0 and y >= 0)

if cond_line:
linea += "*"
elif cond_region:
linea += "o"
elif x == 0 and y == 0:
linea += "+"
elif x == 0:
linea += "|"
elif y == 0:
linea += "-"
else:
linea += " "
print(linea)

# Leyenda
print("\nLeyenda del gráfico:")
print("  * = Recta 3x + 5y = 20 (frontera)")
print("  o = Región factible (3x + 5y <= 20, x>=0, y>=0)")
print("  | = Eje Y")
print("  - = Eje X")
```

```
print("  + = Origen (0,0)")
```

3.3 Ejemplo 3

Un administrador de proyectos tecnológicos organiza su tiempo entre reuniones con stakeholders (x) y trabajo en la documentación técnica (y). Las reuniones requieren al menos 4 horas semanales y la documentación al menos 6 horas. Si dispone de 12 horas para ambas actividades, determine la región factible y analice las combinaciones posibles de tiempo.

Variables

- x = horas dedicadas a reuniones con stakeholders
- y = horas dedicadas a la documentación técnica

Condiciones

1. Reuniones de al menos 4 horas
   - $x \geq 4$
2. Documentación de al menos 6 horas
   - $y \geq 6$
3. Tiempo total disponible máximo de 12 horas
   - $x + y \leq 12$

Sistema de ecuaciones lineales

$$\begin{cases} x \geq 4 \\ y \geq 6 \\ x + y \leq 12 \end{cases}$$

Gráfico generado en Python

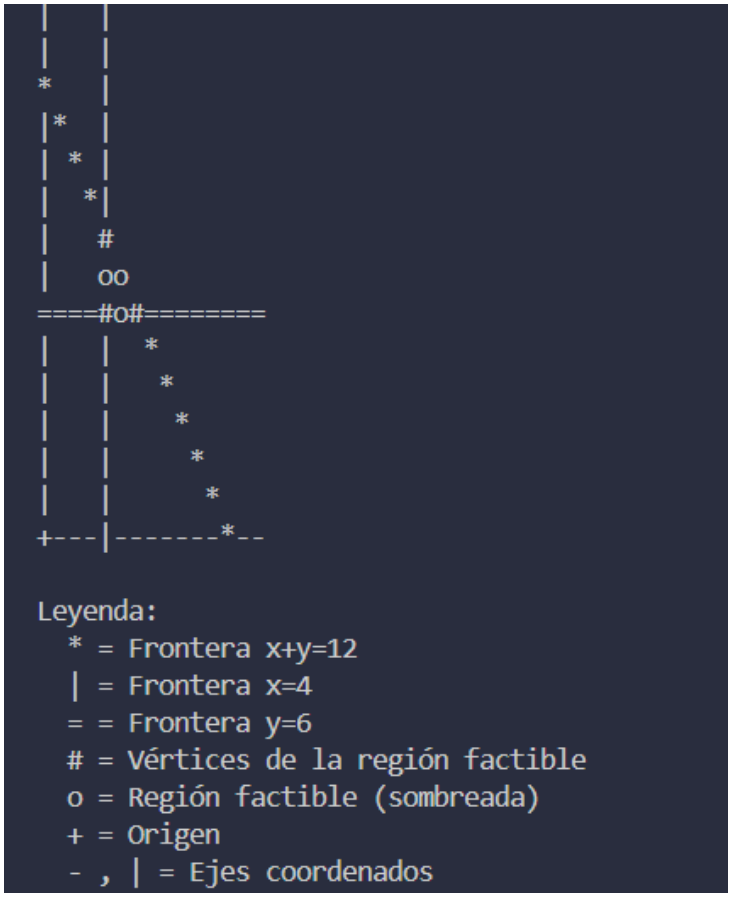


Figura 3.3: Aplicación de restricciones 3 en Python

Interpretación del gráfico

La recta límite es $x + y = 12$, que corta los ejes en los puntos $(12, 0)$ y $(0, 12)$. Aplicando las condiciones $x \geq 4$ y $y \geq 6$, la región factible se reduce al triángulo formado por los puntos:

$$(4, 6), \quad (4, 8), \quad (6, 6).$$

Esto significa que el administrador debe dedicar al menos 4 horas a reuniones y 6 horas a documentación, pero no más de 12 horas en total.

En consecuencia, puede combinar su tiempo de distintas formas dentro de ese triángulo, por ejemplo:

- $(x = 4, y = 7)$: 4 horas de reuniones y 7 de documentación.
- $(x = 5, y = 6)$: 5 horas de reuniones y 6 de documentación.
- $(x = 6, y = 6)$: combinación en el límite máximo permitido.

Código en Python

```
# Graficar restricciones del problema 3 en ASCII con sombreado de la región factible

xmin, xmax = 0, 14
ymin, ymax = 0, 14

for y in range(ymax, ymin - 1, -1):
linea = ""
for x in range(xmin, xmax + 1):

# Restricciones
cond_recta = (abs(x + y - 12) < 0.5)    # frontera x+y=12
cond_x = (x == 4)                       # frontera x=4
cond_y = (y == 6)                       # frontera y=6

# Región factible: x>=4, y>=6, x+y<=12
cond_region = (x >= 4 and y >= 6 and (x + y <= 12))

# Puntos de intersección (vértices)
vertices = [(4,6), (4,8), (6,6)]
cond_vertice = (x,y) in vertices

# Dibujar con prioridad
if cond_vertice:
linea += "#"
elif cond_region:
linea += "o"
elif cond_recta:
linea += "*"
elif cond_x:
linea += "|"
elif cond_y:
linea += "="
elif x == 0 and y == 0:
linea += "+"
elif x == 0:
linea += "|"
elif y == 0:
linea += "-"
else:
linea += " "
print(linea)

print("\nLeyenda:")
print("  * = Frontera x+y=12")
print("  | = Frontera x=4")
print("  = = Frontera y=6")
print("  # = Vértices de la región factible")
print("  o = Región factible (sombreada)")
print("  + = Origen")
print("  - , | = Ejes coordenados")
```

3.4 Ejemplo 4

Una empresa de desarrollo de videojuegos produce dos tipos de assets: Modelos 3D (P1) y Texturas (P2). Cada modelo 3D requiere 2 horas de trabajo y cada textura requiere 3 horas. El equipo de arte tiene un total de 18 horas disponibles semanalmente. Formule las restricciones, represéntelas gráficamente y determine cuántos assets de cada tipo pueden producirse en función del tiempo disponible.

Variables

- x = número de modelos 3D (P1)
- y = número de texturas (P2)

Restricciones

- Restricción principal: $2x + 3y \leq 18$
- Restricciones adicionales: $x \geq 0,\ y \geq 0$

Gráfico generado con Python

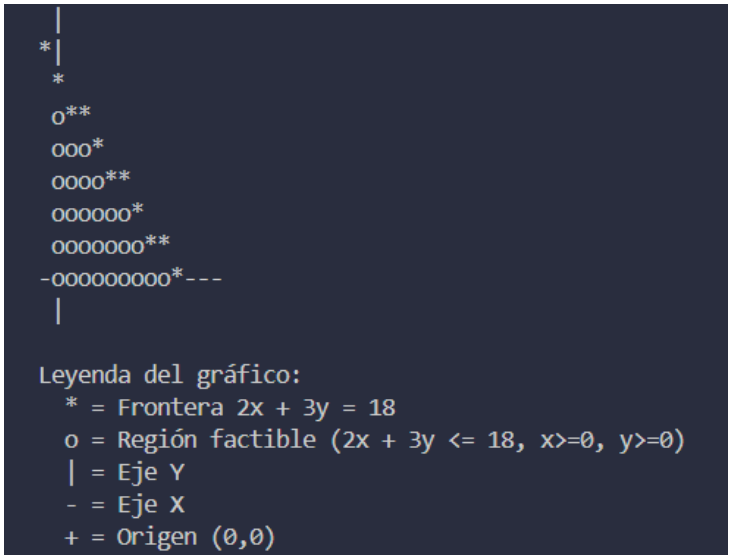


Figura 3.4: Aplicación de restricciones 4 en Python

Interpretación del gráfico

La frontera de la región factible está dada por la recta $2x + 3y = 18$, que corta al eje x en el punto $(9, 0)$ y al eje y en $(0, 6)$. La región factible corresponde al triángulo con vértices $(0, 0)$, $(9, 0)$ y $(0, 6)$.

El equipo puede producir hasta 9 modelos 3D si dedica todo el tiempo a ellos, o hasta 6 texturas si dedica todo a ese tipo de asset. También puede realizar combinaciones intermedias, siempre

que se cumpla $2x + 3y \leq 18$. Por ejemplo, 3 modelos y 4 texturas consumen exactamente las 18 horas disponibles, mientras que 5 modelos y 2 texturas consumen 16 horas, quedando dentro de la región factible.

Código en Python

```python
# Problema 4 - Modelos 3D y Texturas

# Definimos el rango del gráfico
xmin, xmax = -1, 12
ymin, ymax = -1, 8

# Recorremos los valores de Y de arriba hacia abajo
for y in range(ymax, ymin - 1, -1):
linea = ""

for x in range(xmin, xmax + 1):

# Ecuación de la frontera: 2x + 3y = 18
y_line = (18 - 2*x) / 3 if (18 - 2*x) >= 0 else None

# Condición para estar en la recta
cond_line = (y_line is not None and abs(y - y_line) < 0.5)

# Condición para estar en la región factible
cond_region = (x >= 0 and y >= 0 and (2*x + 3*y <= 18))

if cond_line:
linea += "*"
elif cond_region:
linea += "o"
elif x == 0 and y == 0:
linea += "+"
elif x == 0:
linea += "|"
elif y == 0:
linea += "-"
else:
linea += " "
print(linea)

# Leyenda
print("\nLeyenda del gráfico:")
print("  * = Frontera 2x + 3y = 18")
print("  o = Región factible (2x + 3y <= 18, x>=0, y>=0)")
print("  | = Eje Y")
print("  - = Eje X")
print("  + = Origen (0,0)")
```

3.5 Ejemplo 5

Una startup de hardware dispobe de un máximo de 50 unidades de componentes electrónicos. Para ensamblar un dispositivo tipo A se necesitan 5 unidades y para un dispotivo tipo B se necesitan 10 unidades. Determine cuántos dispositivos de cada tipo puede ensamblar sin exceder las 50 unidades de componentes. Formule el problema, resuélalo gráficamnete y explique las posibles combinaciones de producción.

Variables

- x = número de dispositivos tipo A
- y = número de dispositivos tipo B

Restricciones

- Restricción principal: $5x + 10y \leq 50$
- Restricciones adicionales: $x \geq 0,\ y \geq 0$

Gráfico generado con Python

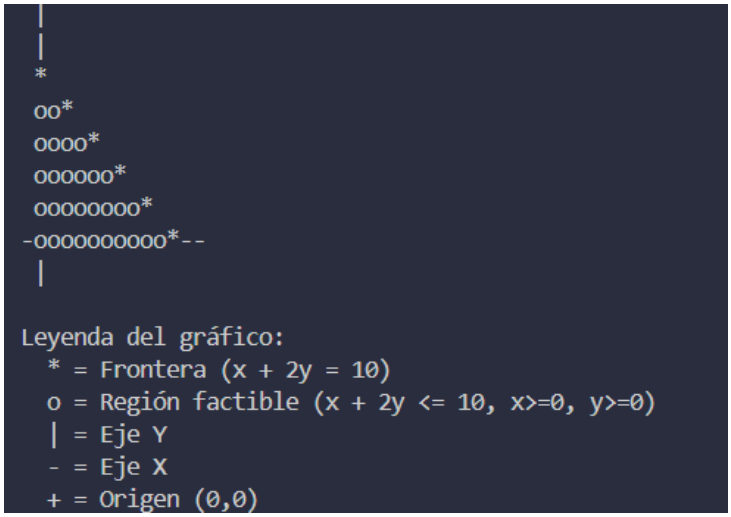


Figura 3.5: Aplicación de restricciones 5 en Python

Interpretación

La frontera de la región factible está dada por la recta $5x+10y = 50$, equivalente a $x+2y = 10$. Esta recta corta al eje x en el punto $(10, 0)$ y al eje y en $(0, 5)$. La región factible corresponde al triángulo con vértices $(0, 0)$, $(10, 0)$ y $(0, 5)$.

La empresa puede ensamblar hasta 10 dispositivos tipo A si dedica todos los componentes a ellos, o hasta 5 dispositivos tipo B si dedica todos los componentes a ese tipo. También puede realizar

combinaciones intermedias, siempre que se cumpla $5x + 10y \leq 50$. Por ejemplo, 6 dispositivos tipo A y 2 tipo B utilizan exactamente las 50 unidades, al igual que 4 dispositivos tipo A y 3 tipo B.

Código en Python

```
# Problema 5 - Dispositivos A y B

# Definimos el rango del gráfico
xmin, xmax = -1, 12
ymin, ymax = -1, 7

# Recorremos los valores de Y de arriba hacia abajo
for y in range(ymax, ymin - 1, -1):
linea = ""

for x in range(xmin, xmax + 1):

# Recta de frontera: x + 2y = 10 -> y = (10 - x)/2
y_line = (10 - x) / 2 if (10 - x) >= 0 else None

# Condición para estar en la recta
cond_line = (y_line is not None and abs(y - y_line) < 0.5)

# Condición para estar en la región factible
cond_region = (x >= 0 and y >= 0 and (x + 2*y <= 10))

if cond_line:
linea += "*"
elif cond_region:
linea += "o"
elif x == 0 and y == 0:
linea += "+"
elif x == 0:
linea += "|"
elif y == 0:
linea += "-"
else:
linea += " "
print(linea)

# Leyenda
print("\nLeyenda del gráfico:")
print("  * = Frontera (x + 2y = 10)")
print("  o = Región factible (x + 2y <= 10, x>=0, y>=0)")
print("  | = Eje Y")
print("  - = Eje X")
print("  + = Origen (0,0)")
```

4 Método de Newton-Raphson

El método de Newton-Raphson es una técnica iterativa utilizada para encontrar raíces de ecuaciones no lineales de la forma:

$$f(x) = 0$$

Es uno de los métodos más eficaces y ampliamente utilizados debido a su rapidez de convergencia cuando se cumplen las condiciones necesarias. Fue propuesto originalmente por Isaac Newton y posteriormente generalizado por Joseph Raphson (Chapra & Canale, 2015)

4.1 Fundamento teórico

La idea básica del método consiste en aproximar la función $f(x)$ mediante su expansión de Taylor alrededor de un punto x_i y despreciar los términos de orden superior:

$$f(x) \approx f(x_i) + f'(x_i)(x - x_i)$$

Para hallar la raíz, se hace $f(x) = 0$, y despejando x se obtiene la fórmula iterativa:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Esta expresión permite calcular una mejor aproximación de la raíz en cada iteración. El proceso continúa hasta que el valor de x_{i+1} converge a la raíz real con una tolerancia previamente establecida (Chapra & Canale, 2015)

4.2 Condiciones de convergencia

El método de Newton-Raphson presenta convergencia cuadrática, es decir, el error disminuye aproximadamente al cuadrado en cada iteración, siempre que se cumplan las siguientes condiciones (Sullivan, 2004):

- La función $f(x)$ es continua y derivable en un intervalo que contiene la raíz buscada.
- La derivada $f'(x)$ no se anula en el entorno de la raíz.
- La estimación inicial x_0 está suficientemente cerca de la raíz real.

Sin embargo, si $f'(x_i)$ se aproxima a cero o si la estimación inicial está muy alejada, el método puede divergir o generar oscilaciones (Burden et al., 2016).

4.3 Criterio de parada

El proceso iterativo se detiene cuando se cumple alguna de las siguientes condiciones (Chapra & Canale, 2015):

- $|f(x_{i+1})| < \varepsilon$
- $|x_{i+1} - x_i| < \varepsilon$

donde ε es la tolerancia o el error máximo permitido.

4.4 Algoritmo del método de Newton-Raphson

1. Elegir una estimación inicial x_0.
2. Calcular $f(x_0)$ y $f'(x_0)$.
3. Evaluar:
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$
4. Repetir el proceso:
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$
hasta que se cumpla el criterio de convergencia.

4.5 Aplicación del método en Python

El objetivo fue desarrollar en el lenguaje de programación Python un programa que permita al usuario ingresar una función $f(x)$ y graficarla en un intervalo definido. Esta etapa inicial tiene como propósito ayudar al usuario a identificar visualmente las posibles raíces y decidir si desea aplicar el método de Newton-Raphson.

En caso afirmativo, el programa solicita un valor inicial x_1 basado en la observación de la gráfica y ejecuta el algoritmo iterativo de Newton-Raphson hasta que la diferencia entre dos aproximaciones sucesivas sea menor a una tolerancia predefinida. Finalmente, el programa muestra en pantalla la raíz aproximada encontrada y el número de iteraciones necesarias para alcanzarla.

Este enfoque combina la interpretación gráfica con el análisis numérico, promoviendo una comprensión más completa del comportamiento de la función y de la eficacia del método iterativo.

Entrada

Una cadena de texto que representa una función matemática.

$$f(x) = x^3 - x - 1$$

Salida

- Gráfica para evaluar si realizar o no el método.
- La raíz encontrada.
- Número de iteraciones realizadas hasta encontrar la raíz.

Código

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

# === 1. Ingreso de la función ===
func_str = input("Ingrese la función f(x): ")  # Ejemplo: x**3 - x - 1

# Definimos la función y su derivada (usando derivada numérica)
def f(x):
return eval(func_str)

def f_prime(x, h=1e-6):  # derivada numérica
return (f(x + h) - f(x - h)) / (2 * h)

# === 2. Graficar la función ===
xmin = float(input("Ingrese el valor mínimo de x: "))
xmax = float(input("Ingrese el valor máximo de x: "))

x = np.linspace(xmin, xmax, 400)
y = f(x)

plt.figure(figsize=(8, 5))
plt.plot(x, y, label=f"f(x) = {func_str}", color='blue')
plt.axhline(0, color='black', linestyle='--')
plt.axvline(0, color='black', linestyle='--')
plt.title("Gráfico de la función ingresada")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.show()

# === 3. Pregunta si desea aplicar Newton-Raphson ===
op = input("¿Desea encontrar una raíz con el método de Newton-Raphson? (s/n): ").
    ↪ lower()

if op == "s":
# === 4. Ingreso de punto inicial ===
```

```
x0 = float(input("Basado en la gráfica, ingrese el valor inicial x1: "))

# Parámetros del método
tol = 1e-6
max_iter = 100

# Iteraciones
for i in range(1, max_iter + 1):
fx = f(x0)
fpx = f_prime(x0)

if fpx == 0:
print(f"La derivada es cero en x = {x0}. El método no puede continuar.")
break

x1 = x0 - fx / fpx

# Verificar convergencia
if abs(x1 - x0) < tol:
print(f"\n Raíz aproximada encontrada: {x1:.6f}")
print(f"Iteraciones realizadas: {i}")
break

x0 = x1
else:
print("\n No se alcanzó la convergencia después de", max_iter, "iteraciones.")

else:
print("No se aplicó el método de Newton-Raphson.")
```

Ejecución

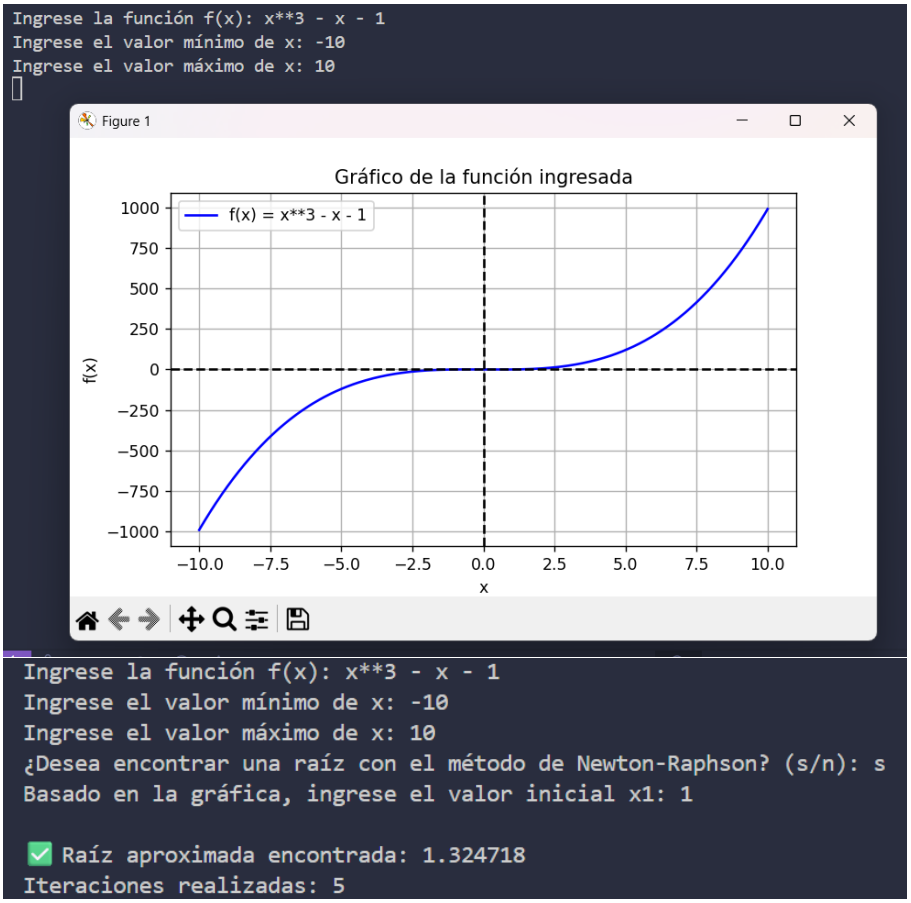


Figura 4.1: Método de Newthon-Rapshon en Python

5 Método de Bisección

El método de bisección es uno de los algoritmos más simples, robustos y confiables para la localización de raíces de ecuaciones no lineales de la forma

$$f(x) = 0.$$

Su fundamento teórico se basa en el Teorema del Valor Intermedio, el cual garantiza la existencia de una raíz siempre que la función sea continua en un intervalo cerrado $[a, b]$ y que se cumpla la condición

$$f(a)\, f(b) < 0.$$

Esto indica que en el intervalo ocurre un cambio de signo y, por tanto, debe existir al menos una raíz en su interior (Burden et al., 2016).

5.1 Idea fundamental del método

El método consiste en dividir sucesivamente el intervalo $[a, b]$ por su punto medio. Si llamamos

$$c = \frac{a+b}{2},$$

entonces evaluamos $f(c)$ y determinamos en qué subintervalo persiste el cambio de signo:

- Si $f(a)\, f(c) < 0$, la raíz se encuentra en $[a, c]$.
- Si $f(c)\, f(b) < 0$, la raíz se encuentra en $[c, b]$.

Este proceso se repite recursivamente, produciendo intervalos cada vez más pequeños. Debido a su naturaleza, el método siempre converge cuando la función es continua y se inicia con un intervalo válido, aunque la convergencia es relativamente lenta (Chapra & Canale, 2015).

5.2 Criterios de parada

El método se detiene cuando se cumple alguna de las siguientes condiciones:

1. El ancho del intervalo es menor que una tolerancia establecida:

$$|b - a| < \varepsilon.$$

2. El valor funcional es cercano a cero:

$$|f(c)| < \delta.$$

3. Se alcanza un número máximo de iteraciones:

$$n \geq n_{\text{máx}}.$$

El error después de n iteraciones está acotado por

$$|x - c_n| \leq \frac{b-a}{2^n},$$

lo cual permite estimar de manera precisa el número de pasos necesarios para obtener una tolerancia deseada (Süli & Mayers, 2003).

5.3 Ventajas del método

El método de bisección posee varias ventajas importantes:

- Garantiza convergencia siempre que exista un cambio de signo en el intervalo.
- Es simple de implementar.
- Permite una estimación directa del error en cada iteración.
- Es estable numéricamente y no requiere derivadas.

Estas características lo convierten en un método ideal para iniciar procesos de localización de raíces o para validar aproximaciones obtenidas mediante métodos más rápidos pero menos robustos, como Newton-Raphson o la secante (Press et al., 2007).

5.4 Desventajas

A pesar de su robustez, el método de bisección presenta algunas limitaciones:

- La convergencia es lineal y relativamente lenta en comparación con otros métodos.
- Requiere conocer un intervalo donde la función cambie de signo.
- No obtiene raíces múltiples o raíces donde la función no cambia de signo.

Aun así, su combinación de simplicidad y fiabilidad lo convierte en un método fundamental dentro de los algoritmos de análisis numérico (Burden et al., 2016).

5.5 Aplicación del método en Python

Se desarrolló en Python un programa que permite al usuario ingresar una función matemática y visualizar su gráfico en un intervalo definido. Esta etapa inicial facilita la elección del intervalo $[a, b]$ donde la función cambia de signo.

Una vez seleccionado el intervalo, el programa ejecuta el método de bisección iterativamente, mostrando en pantalla una tabla con los valores de a, b, c, $f(c)$ y el error de cada iteración. El proceso finaliza cuando se cumple la condición de tolerancia o se alcanza el número máximo de iteraciones. Finalmente, se presenta la raíz aproximada encontrada.

Este enfoque combina el análisis gráfico con la interpretación numérica, permitiendo comprender mejor el comportamiento de la función y la estabilidad del método.

Entrada

Una cadena de texto que representa una función matemática.

$$f(x) = x^3 - x - 1$$

Salida

- Gráfica para evaluar el intervalo de búsqueda.
- Tabla con las iteraciones del método.
- Raíz aproximada y número de iteraciones realizadas.

Restricciones

El método requiere que:

- $f(x)$ sea continua en el intervalo $[a, b]$.
- Se cumpla la condición $f(a) \cdot f(b) < 0$.

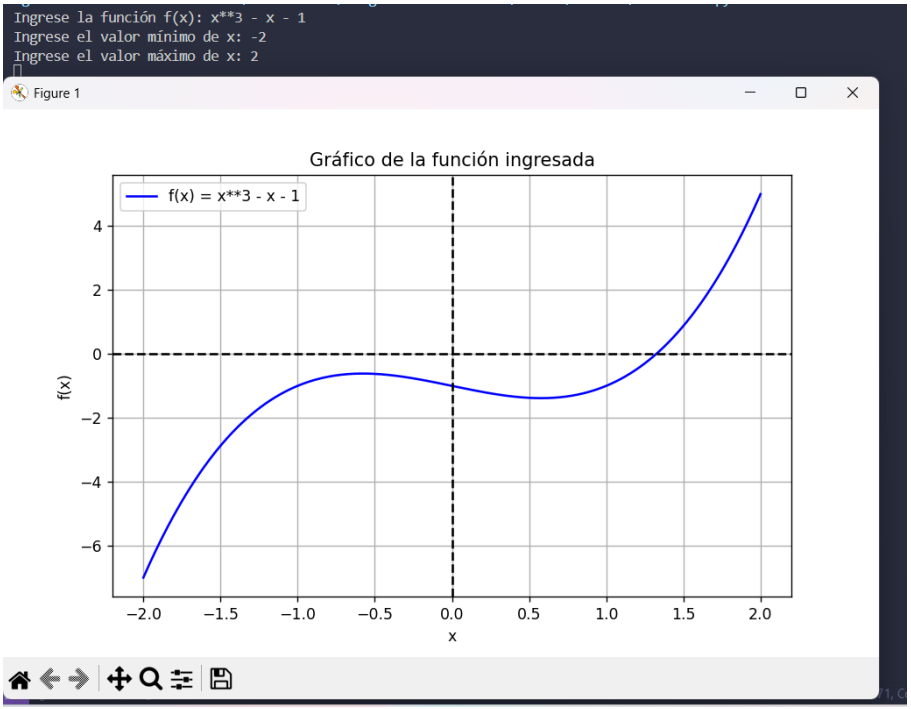
Código

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

# === 1. Ingreso de la función ===
func_str = input("Ingrese la función f(x): ")  # Ejemplo: x**3 - x - 1

# Definimos la función
def f(x):
return eval(func_str, {"np": np, "x": x})

# === 2. Graficar la función ===
xmin = float(input("Ingrese el valor mínimo de x: "))
xmax = float(input("Ingrese el valor máximo de x: "))

x = np.linspace(xmin, xmax, 400)
```

```
y = f(x)

plt.figure(figsize=(8, 5))
plt.plot(x, y, label=f"f(x) = {func_str}", color='blue')
plt.axhline(0, color='black', linestyle='--')
plt.axvline(0, color='black', linestyle='--')
plt.title("Gráfico de la función ingresada")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.show()

# === 3. Pregunta si desea aplicar el método de Bisección ===
op = input("¿Desea encontrar una raíz con el método de Bisección? (s/n): ").lower()

if op == "s":
# === 4. Ingreso del intervalo inicial ===
a = float(input("Ingrese el extremo izquierdo del intervalo (a): "))
b = float(input("Ingrese el extremo derecho del intervalo (b): "))

# Verificación del cambio de signo
if f(a) * f(b) > 0:
print("\n No hay cambio de signo en el intervalo [a, b]. Intente con otro intervalo
    ↪ .")
else:
# Parámetros del método
tol = 1e-6
max_iter = 100

print("\nIteración |     a       |      b       |      c       |    f(c)     |
    ↪ Error")
print("-------------------------------------------------------------------")

for i in range(1, max_iter + 1):
c = (a + b) / 2
fc = f(c)
error = abs(b - a) / 2

print(f"{i:9d} | {a:10.6f} | {b:10.6f} | {c:10.6f} | {fc:10.6f} | {error:10.6f}")

if abs(fc) < tol or error < tol:
print(f"\n Raíz aproximada encontrada: {c:.6f}")
print(f"Iteraciones realizadas: {i}")
break

if f(a) * fc < 0:
b = c
else:
a = c
else:
print("\n No se alcanzó la convergencia después de", max_iter, "iteraciones.")
```

```
66 else:
67 print("No se aplicó el método de Bisección.")
```

Ejecución




```
Ingrese la función f(x): x**3 - x - 1
Ingrese el valor mínimo de x: -2
Ingrese el valor máximo de x: 2
¿Desea encontrar una raíz con el método de Bisección? (s/n): s
Ingrese el extremo izquierdo del intervalo (a): 1
Ingrese el extremo derecho del intervalo (b): 2

Iteración |     a       |     b       |     c       |    f(c)     |   Error
-------------------------------------------------------------------------
        1 |   1.000000  |   2.000000  |   1.500000  |   0.875000  |   0.500000
        2 |   1.000000  |   1.500000  |   1.250000  |  -0.296875  |   0.250000
        3 |   1.250000  |   1.500000  |   1.375000  |   0.224609  |   0.125000
        4 |   1.250000  |   1.375000  |   1.312500  |  -0.051514  |   0.062500
        5 |   1.312500  |   1.375000  |   1.343750  |   0.082611  |   0.031250
        6 |   1.312500  |   1.343750  |   1.328125  |   0.014576  |   0.015625
        7 |   1.312500  |   1.328125  |   1.320312  |  -0.018711  |   0.007812
        8 |   1.320312  |   1.328125  |   1.324219  |  -0.002128  |   0.003906
        9 |   1.324219  |   1.328125  |   1.326172  |   0.006209  |   0.001953
       10 |   1.324219  |   1.326172  |   1.325195  |   0.002037  |   0.000977
       11 |   1.324219  |   1.325195  |   1.324707  |  -0.000047  |   0.000488
       12 |   1.324707  |   1.325195  |   1.324951  |   0.000995  |   0.000244
       13 |   1.324707  |   1.324951  |   1.324829  |   0.000474  |   0.000122
       14 |   1.324707  |   1.324829  |   1.324768  |   0.000214  |   0.000061
       15 |   1.324707  |   1.324768  |   1.324738  |   0.000084  |   0.000031
       16 |   1.324707  |   1.324738  |   1.324722  |   0.000018  |   0.000015
       17 |   1.324707  |   1.324722  |   1.324715  |  -0.000014  |   0.000008
       18 |   1.324715  |   1.324722  |   1.324718  |   0.000002  |   0.000004
       19 |   1.324715  |   1.324718  |   1.324717  |  -0.000006  |   0.000002
       20 |   1.324717  |   1.324718  |   1.324718  |  -0.000002  |   0.000001

✅ Raíz aproximada encontrada: 1.324718
Iteraciones realizadas: 20
```

Figura 5.1: Método de Bisección en Python

6 Método de la Secante

El método de la secante es una técnica numérica ampliamente utilizada para la resolución de ecuaciones no lineales, especialmente en contextos donde la derivada de la función no está disponible o resulta difícil de calcular. A diferencia del método de Newton–Raphson, que requiere la evaluación explícita de la derivada, la secante se basa en una aproximación numérica de la misma utilizando dos puntos consecutivos de la función (Burden & Faires, 2011; Chapra & Canale, 2015).

6.1 Fundamento teórico

El método surge de la idea de aproximar la derivada de la función mediante un cociente incremental, reemplazando la derivada verdadera por:

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

Sustituyendo esta expresión en la fórmula de Newton–Raphson se obtiene la iteración del método de la secante:

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

La deducción formal y el análisis matemático del método pueden consultarse en textos clásicos de análisis numérico (Burden & Faires, 2011), así como en manuales modernos de métodos computacionales (Atkinson, 2009; Chapra & Canale, 2015).

6.2 Interpretación geométrica

Geométricamente, el método consiste en trazar la recta secante que pasa por los puntos $(x_{n-1}, f(x_{n-1}))$ y $(x_n, f(x_n))$, y encontrar su intersección con el eje x. Esta construcción geométrica proporciona una aproximación sucesiva a la raíz de la ecuación $f(x) = 0$ (Sullivan, 2004).

La secante generada en cada iteración reemplaza el uso de la tangente del método de Newton, permitiendo que el método avance sin necesidad de calcular derivadas. Esta propiedad convierte a la secante en un algoritmo eficiente en problemas aplicados donde la evaluación de $f'(x)$ puede ser costosa o inestable (Cheney & Kincaid, 2009).

6.3 Convergencia

El método posee una convergencia superlineal, cuya razón de convergencia es aproximadamente 1.618, el número áureo. Aunque es menos rápido que Newton–Raphson, suele ser más eficiente cuando la derivada no está disponible o cuando su cálculo es propenso a errores numéricos (Atkinson, 2009). Sin embargo, su éxito requiere seleccionar dos valores iniciales adecuados para evitar divisiones entre valores de función muy pequeños o iteraciones divergentes (Burden & Faires, 2011).

6.4 Ventajas y limitaciones

Entre sus principales ventajas destacan:

- No requiere el cálculo de la derivada de la función.
- Tiende a ser más estable que Newton–Raphson cuando la derivada es difícil de evaluar.
- Posee una tasa de convergencia superior al método de bisección.

Sus principales limitaciones incluyen:

- No garantiza convergencia global.
- Requiere dos valores iniciales definidos.
- Puede divergir si los valores iniciales no son adecuados o si la función presenta discontinuidades o múltiples raíces cercanas.

Para un análisis detallado de sus propiedades teóricas y aplicaciones prácticas pueden consultarse referencias especializadas en análisis numérico (Atkinson, 2009; Burden & Faires, 2011; Chapra & Canale, 2015; Cheney & Kincaid, 2009).

6.5 Aplicación del método en Python

Se desarrolló un programa en Python que permite al usuario ingresar una función $f(x)$ y graficarla en un intervalo definido. Esta etapa inicial facilita la identificación visual de posibles raíces y la selección de los valores iniciales x_0 y x_1.

Luego, el programa aplica el método de la secante iterativamente hasta alcanzar una tolerancia establecida o un número máximo de iteraciones. En cada iteración, se muestran los valores de x_0, x_1, $f(x_0)$, $f(x_1)$, la nueva aproximación x_2 y el error absoluto. Finalmente, se imprime la raíz aproximada encontrada y el número de iteraciones necesarias.

Este procedimiento combina la exploración gráfica con el razonamiento numérico, fomentando una comprensión más completa de la convergencia y del comportamiento de la función.

Entrada

Una cadena de texto que representa una función matemática.

$$f(x) = x^3 - x - 1$$

Salida

- Gráfica para visualizar la función y elegir los puntos iniciales.
- Tabla con los valores de cada iteración.
- Raíz aproximada y número de iteraciones necesarias.

Restricciones

- $f(x)$ debe ser continua en el intervalo analizado.
- Los valores iniciales x_0 y x_1 deben ser distintos y cercanos a la raíz.

Código

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

# === 1. Ingreso de la función ===
func_str = input("Ingrese la función f(x): ")  # Ejemplo: x**3 - x - 1

# Definimos la función
def f(x):
return eval(func_str, {"np": np, "x": x})

# === 2. Graficar la función ===
xmin = float(input("Ingrese el valor mínimo de x: "))
xmax = float(input("Ingrese el valor máximo de x: "))

x = np.linspace(xmin, xmax, 400)
y = f(x)

plt.figure(figsize=(8, 5))
plt.plot(x, y, label=f"f(x) = {func_str}", color='blue')
plt.axhline(0, color='black', linestyle='--')
plt.axvline(0, color='black', linestyle='--')
plt.title("Gráfico de la función ingresada")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
```

```
plt.grid(True)
plt.show()

# === 3. Pregunta si desea aplicar el método de la secante ===
op = input("¿Desea encontrar una raíz con el método de la Secante? (s/n): ").lower
    ↪ ()

if op == "s":
# === 4. Ingreso de puntos iniciales ===
x0 = float(input("Ingrese el primer valor inicial x0: "))
x1 = float(input("Ingrese el segundo valor inicial x1: "))

# Parámetros del método
tol = 1e-6
max_iter = 100

print("\nIteración |      x0      |      x1      |     f(x0)     |     f(x1)     |
    ↪     x2      |   Error")
print("-------------------------------------------------------------------------")

for i in range(1, max_iter + 1):
f0 = f(x0)
f1 = f(x1)

if f1 - f0 == 0:
print(f"\n División por cero en la iteración {i}. El método no puede continuar.")
break

x2 = x1 - f1 * (x1 - x0) / (f1 - f0)
error = abs(x2 - x1)

print(f"{i:9d} | {x0:10.6f} | {x1:10.6f} | {f0:12.6f} | {f1:12.6f} | {x2:10.6f} | {
    ↪ error:10.6f}")

if error < tol:
print(f"\n Raíz aproximada encontrada: {x2:.6f}")
print(f"Iteraciones realizadas: {i}")
break

x0, x1 = x1, x2

else:
print("\n No se alcanzó la convergencia después de", max_iter, "iteraciones.")
else:
print("No se aplicó el método de la Secante.")
```

Ejecución

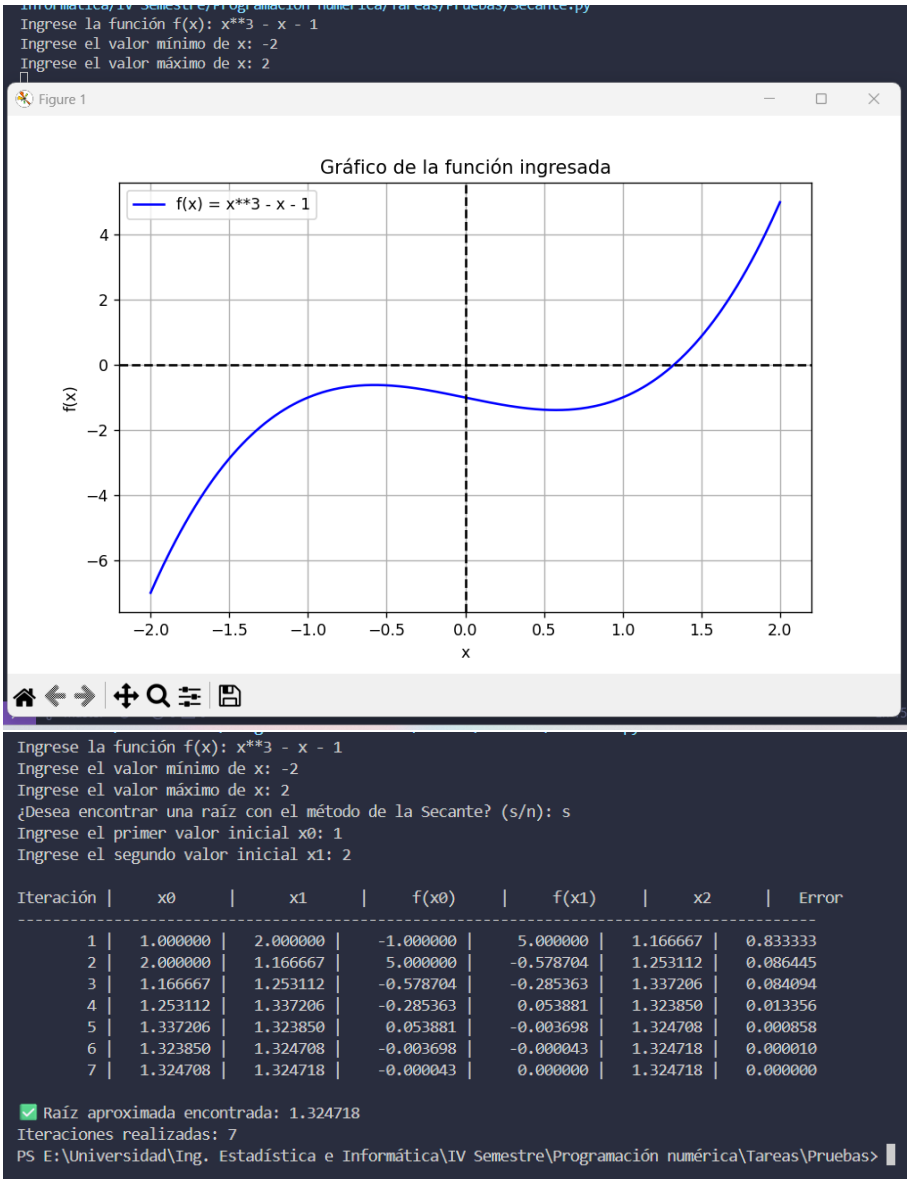


Figura 6.1: Método de la Secante en Python

7 Método de Punto Fijo

El método de punto fijo es uno de los procedimientos fundamentales para resolver ecuaciones no lineales de la forma

$$f(x) = 0.$$

A diferencia de otros métodos basados directamente en la función $f(x)$, el método de punto fijo se construye a partir de una transformación adecuada de la ecuación original en una función equivalente

$$x = g(x),$$

de modo que la raíz buscada es un punto fijo de la función g, es decir, un valor r tal que $g(r) = r$ (Burden et al., 2016).

La ecuación original puede expresarse de diversas maneras para obtener distintas funciones de iteración $g(x)$. La elección de esta función es crucial para garantizar la convergencia y determinar la eficiencia del método.

7.1 Definición y formulación básica

Dado un valor inicial x_0, la aproximación iterativa se construye mediante el proceso repetitivo:

$$x_{n+1} = g(x_n), \qquad n = 0, 1, 2, \ldots$$

Si esta sucesión converge a un valor r, entonces

$$\lim_{n\to\infty} x_n = r \quad \Rightarrow \quad r = g(r),$$

lo que confirma que r es un punto fijo de g y, por ende, una solución de la ecuación original (Chapra & Canale, 2015).

7.2 Condición de convergencia

El comportamiento del método depende de las propiedades de la función $g(x)$. Uno de los resultados más importantes para garantizar convergencia es el siguiente:

Criterio de convergencia. Supongamos que la función g es continua en un intervalo I y que además su derivada $g'(x)$ es continua en el mismo intervalo. Si existe una constante $0 < k < 1$ tal que

$$|g'(x)| \leq k \qquad \text{para todo } x \in I,$$

entonces el método de punto fijo es convergente para cualquier elección de $x_0 \in I$, y la sucesión generada converge a un único punto fijo en dicho intervalo (Burden & Faires, 2011).

Este criterio implica que la derivada de g controla la "fuerza" con la que la iteración se acerca al punto fijo. En términos prácticos:

$$|g'(r)| < 1 \quad \Rightarrow \quad \text{convergencia local},$$
$$|g'(r)| > 1 \quad \Rightarrow \quad \text{divergencia}.$$

Esto hace evidente que la elección de la forma $g(x)$ no es arbitraria. Algunas transformaciones del tipo $x = g(x)$ producen convergencia, otras generan oscilaciones y otras divergencia total (Cheney & Kincaid, 2009).

7.3 Orden de convergencia

El método de punto fijo tiene convergencia lineal bajo las condiciones mencionadas. Esto significa que existe una constante C tal que

$$|x_{n+1} - r| \approx C\,|x_n - r|,$$

lo cual sugiere una velocidad de aproximación más lenta frente a métodos como Newton-Raphson, cuya convergencia es cuadrática (Atkinson, 2009). Sin embargo, su simplicidad lo convierte en una herramienta útil para la enseñanza y para establecer intervalos iniciales seguros para otros métodos más rápidos.

7.4 Errores y criterios de parada

El error en la iteración puede estimarse mediante:

$$|x_{n+1} - x_n| < \varepsilon \quad \text{o bien} \quad |g(x_n) - x_n| < \varepsilon,$$

donde ε es una tolerancia escogida. Otra forma consiste en fijar un número máximo de iteraciones si la función converge lentamente o presenta oscilaciones (Press et al., 2007).

7.5 Ventajas del método

El método de punto fijo presenta varias virtudes importantes:

- Su implementación es extremadamente sencilla.
- No requiere derivadas como Newton-Raphson.
- Es útil para analizar transformaciones funcionales equivalentes.
- Proporciona un marco general para estudiar muchos métodos iterativos.

- Permite visualizar la convergencia mediante diagramas cobweb (trampas de telaraña), muy utilizados en la enseñanza (Anton et al., 2012).

Además, su estructura simple hace posible analizar la estabilidad de los puntos fijos en modelos matemáticos aplicados, como ecuaciones diferenciales, sistemas dinámicos y procesos iterativos en ingeniería (Süli & Mayers, 2003).

7.6 Desventajas

A pesar de su utilidad conceptual y didáctica, el método presenta ciertas limitaciones:

- La convergencia depende fuertemente de la elección de $g(x)$.
- Incluso con una buena elección, la convergencia es lineal, por lo que puede ser lenta.
- Si $|g'(x)| \geq 1$, la iteración diverge.
- No siempre es sencillo encontrar una transformación apropiada.

En aplicaciones industriales o científicas, suele preferirse utilizar punto fijo como método preliminar para garantizar la existencia de una raíz antes de usar algoritmos más robustos o rápidos, como Newton-Raphson, la secante o métodos híbridos (Chapra & Canale, 2015).

7.7 Interpretación gráfica

Gráficamente, el método consiste en trazar la curva $y = g(x)$ junto con la recta $y = x$. El punto donde ambas se intersectan es el punto fijo. La iteración sigue los pasos:

1. Comenzar en $(x_0, g(x_0))$.
2. Proyectar verticalmente hacia la curva.
3. Proyectar horizontalmente hacia la recta $y = x$.

Este proceso se repite hasta alcanzar una convergencia visual. Este tipo de representación, conocida como diagrama cobweb, es muy útil para estudiar estabilidad y para reforzar la intuición del método (Riley et al., 2006).

7.8 Conclusión

El método de punto fijo constituye una herramienta esencial en el análisis numérico. Aunque no es el método más rápido ni el más eficiente, su simplicidad, su fundamento teórico sólido y su relación con otros métodos iterativos lo hacen indispensable en la formación matemática y en el estudio formal de la convergencia de algoritmos numéricos.

Su aplicación correcta depende de una adecuada elección de la función $g(x)$ y de un análisis cuidadoso de las condiciones de convergencia. Esto permite utilizarlo como punto de partida en el estudio y aplicación de métodos más avanzados para la resolución de ecuaciones no lineales (Burden et al., 2016).

7.9 Aplicación del método en Python

El objetivo fue desarrollar un programa en Python que permita ingresar una función $f(x)$ y su forma iterativa $g(x)$, graficar la función y luego aplicar el método de Punto Fijo si el usuario lo desea. El programa calcula sucesivamente los valores de $x_{n+1} = g(x_n)$ hasta alcanzar una precisión deseada y muestra los resultados obtenidos.

Entrada

- Una cadena de texto que representa la función original $f(x)$.
- La forma iterativa correspondiente $g(x)$.
- El intervalo para graficar.
- El valor inicial x_0.

$$f(x) = \cos(x) - x$$
$$g(x) = \cos(x)$$

Salida

- Gráfica de la función $f(x)$.
- La raíz aproximada encontrada.
- Número de iteraciones realizadas hasta cumplir la tolerancia.

Restricciones

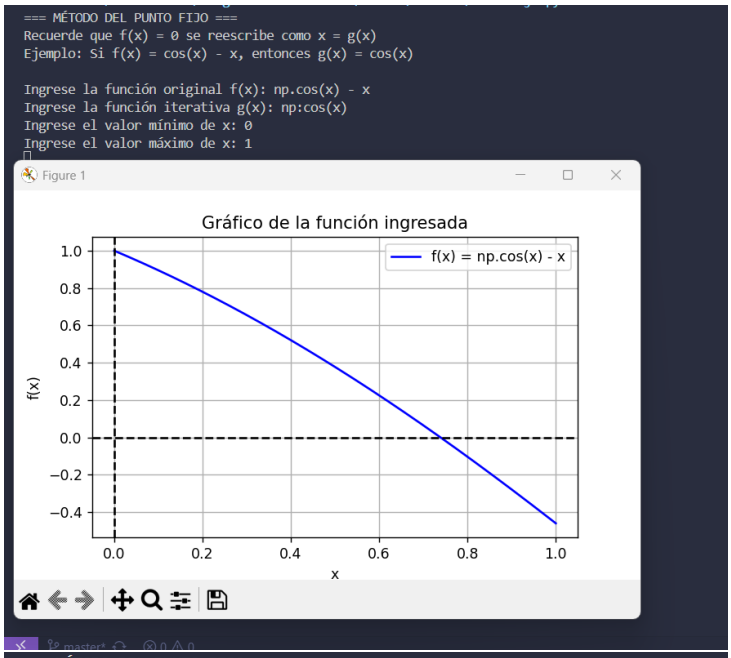
- El método requiere que $|g'(x)| < 1$ en el entorno de la raíz para garantizar la convergencia.
- La función debe ser continua y evaluable en todo el intervalo seleccionado.

Código

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

# === 1. Ingreso de la función y su forma iterativa g(x) ===
print("=== MÉTODO DEL PUNTO FIJO ===")
print("Recuerde que f(x) = 0 se reescribe como x = g(x)")
print("Ejemplo: Si f(x) = cos(x) - x, entonces g(x) = cos(x)\n")

func_str = input("Ingrese la función original f(x): ")   # Ejemplo: np.cos(x) - x
g_str = input("Ingrese la función iterativa g(x): ")     # Ejemplo: np.cos(x)

# Definición de funciones
def f(x):
return eval(func_str, {"np": np, "x": x})

def g(x):
return eval(g_str, {"np": np, "x": x})

# === 2. Graficar la función f(x) ===
xmin = float(input("Ingrese el valor mínimo de x: "))
xmax = float(input("Ingrese el valor máximo de x: "))

x = np.linspace(xmin, xmax, 400)
y = f(x)

plt.figure(figsize=(8, 5))
plt.plot(x, y, label=f"f(x) = {func_str}", color='blue')
plt.axhline(0, color='black', linestyle='--')
plt.axvline(0, color='black', linestyle='--')
plt.title("Gráfico de la función ingresada")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.show()

# === 3. Pregunta si desea aplicar el método de punto fijo ===
op = input("¿Desea aplicar el método de Punto Fijo? (s/n): ").lower()

if op == "s":
# === 4. Ingreso de valor inicial ===
x0 = float(input("Ingrese el valor inicial x0: "))

tol = 1e-6
max_iter = 100

print("\nIteración |     x0      |     g(x0)     |     f(x0)     |   Error")
print("-------------------------------------------------------------------")

for i in range(1, max_iter + 1):
x1 = g(x0)
```

```
52 error = abs(x1 - x0)
53
54 print(f"{i:9d} | {x0:10.6f} | {x1:12.6f} | {f(x1):12.6f} | {error:10.6f}")
55
56 if error < tol:
57 print(f"\n Raíz aproximada encontrada: {x1:.6f}")
58 print(f"Iteraciones realizadas: {i}")
59 break
60
61 x0 = x1
62 else:
63 print("\n No se alcanzó la convergencia después de", max_iter, "iteraciones.")
64 else:
65 print("No se aplicó el método de Punto Fijo.")
```

Ejecución




```
=== MÉTODO DEL PUNTO FIJO ===
Recuerde que f(x) = 0 se reescribe como x = g(x)
Ejemplo: Si f(x) = cos(x) - x, entonces g(x) = cos(x)

Ingrese la función original f(x): np.cos(x) - x
Ingrese la función iterativa g(x): np.cos(x)
Ingrese el valor mínimo de x: 0
Ingrese el valor máximo de x: 1
¿Desea aplicar el método de Punto Fijo? (s/n): s
Ingrese el valor inicial x0: 0.5

Iteración |     x0      |     g(x0)     |     f(x0)     |   Error
-----------------------------------------------------------------
        1 |   0.500000  |     0.877583  |    -0.238570  |   0.377583
        2 |   0.877583  |     0.639012  |     0.163673  |   0.238570
        3 |   0.639012  |     0.802685  |    -0.107907  |   0.163673
        4 |   0.802685  |     0.694778  |     0.073418  |   0.107907
        5 |   0.694778  |     0.768196  |    -0.049030  |   0.073418
        6 |   0.768196  |     0.719165  |     0.033190  |   0.049030
        7 |   0.719165  |     0.752356  |    -0.022275  |   0.033190
        8 |   0.752356  |     0.730081  |     0.015039  |   0.022275
        9 |   0.730081  |     0.745120  |    -0.010114  |   0.015039
       10 |   0.745120  |     0.735006  |     0.006820  |   0.010114
       11 |   0.735006  |     0.741827  |    -0.004591  |   0.006820
       12 |   0.741827  |     0.737236  |     0.003094  |   0.004591
       13 |   0.737236  |     0.740330  |    -0.002083  |   0.003094
       14 |   0.740330  |     0.738246  |     0.001404  |   0.002083
       15 |   0.738246  |     0.739650  |    -0.000945  |   0.001404
       16 |   0.739650  |     0.738705  |     0.000637  |   0.000945
       17 |   0.738705  |     0.739341  |    -0.000429  |   0.000637
       18 |   0.739341  |     0.738912  |     0.000289  |   0.000429
       19 |   0.738912  |     0.739201  |    -0.000195  |   0.000289
       20 |   0.739201  |     0.739007  |     0.000131  |   0.000195
       21 |   0.739007  |     0.739138  |    -0.000088  |   0.000131
       22 |   0.739138  |     0.739050  |     0.000060  |   0.000088
       23 |   0.739050  |     0.739109  |    -0.000040  |   0.000060
       24 |   0.739109  |     0.739069  |     0.000027  |   0.000040
       25 |   0.739069  |     0.739096  |    -0.000018  |   0.000027
       26 |   0.739096  |     0.739078  |     0.000012  |   0.000018
       27 |   0.739078  |     0.739090  |    -0.000008  |   0.000012
       28 |   0.739090  |     0.739082  |     0.000006  |   0.000008
       29 |   0.739082  |     0.739087  |    -0.000004  |   0.000006
       30 |   0.739087  |     0.739084  |     0.000003  |   0.000004
       31 |   0.739084  |     0.739086  |    -0.000002  |   0.000003
       32 |   0.739086  |     0.739084  |     0.000001  |   0.000002
       33 |   0.739084  |     0.739086  |    -0.000001  |   0.000001
       34 |   0.739086  |     0.739085  |     0.000001  |   0.000001

 Raíz aproximada encontrada: 0.739085
Iteraciones realizadas: 34
```

Figura 7.1: Método de Punto Fijo en Python

8 Método de Regula Falsi

El método de regula falsi, también conocido como método de la falsa posición, es una técnica clásica para la solución numérica de ecuaciones no lineales de la forma

$$f(x) = 0.$$

Este método combina aspectos del método de la bisección y del método de la secante, manteniendo siempre un intervalo que contiene la raíz mientras utiliza una aproximación lineal similar a la secante para generar nuevas aproximaciones. Su origen se remonta a prácticas matemáticas antiguas, pero su formalización moderna aparece descrita en textos fundamentales de análisis numérico (Burden & Faires, 2011; Chapra & Canale, 2015; Cheney & Kincaid, 2009).

8.1 Idea fundamental del método

Sea un intervalo cerrado $[a, b]$ tal que la función f sea continua en él y satisfaga la condición

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

lo cual garantiza, mediante el Teorema del Valor Intermedio, la existencia de al menos una raíz en dicho intervalo. A diferencia del método de la bisección, que toma el punto medio como nueva aproximación, el método de regula falsi utiliza la recta secante que une los puntos $(a, f(a))$ y $(b, f(b))$ para obtener una aproximación más refinada a la raíz (Burden et al., 2016; Chapra & Canale, 2015).

El punto donde esta recta corta al eje x se calcula mediante la fórmula:

$$x_r = b - \frac{f(b)\,(a - b)}{f(a) - f(b)}.$$

Este x_r se denomina falsa posición y representa la aproximación actual de la raíz.

8.2 Actualización del intervalo

Una vez calculado x_r, se evalúa $f(x_r)$ y se conserva el intervalo que garantice el cambio de signo. Es decir:

$$\begin{cases} a = x_r & \text{si } f(a)\, f(x_r) < 0, \\ b = x_r & \text{si } f(a)\, f(x_r) > 0. \end{cases}$$

Este proceso asegura que la raíz permanece dentro del intervalo, proporcionando la robustez característica del método de bisección, pero con un paso adicional guiado por la pendiente de

la secante, lo que permite avanzar de manera más eficiente en muchas situaciones (Atkinson, 2009; Süli & Mayers, 2003).

8.3 Convergencia

El método de regula falsi es linealmente convergente, aunque en algunos casos su velocidad puede ser lenta. Esto ocurre especialmente cuando uno de los extremos del intervalo permanece fijo durante muchas iteraciones, lo cual puede suceder si la función presenta curvaturas pronunciadas o cambios de pendiente que provocan que la secante tienda a acercarse preferentemente a uno de los extremos (Burden & Faires, 2011; Press et al., 2007).

A pesar de ello, su principal ventaja es que **nunca pierde la raíz**, ya que siempre mantiene el intervalo acotado con un cambio de signo, lo que lo hace más seguro que métodos que no usan acotamiento, como el método de la secante clásico.

8.4 Criterios de parada

Los criterios más empleados para finalizar el proceso iterativo son:

- Error relativo entre iteraciones:

$$\left| \frac{x_r^{(k)} - x_r^{(k-1)}}{x_r^{(k)}} \right| < \varepsilon.$$

- Evaluación de la función:

$$|f(x_r)| < \varepsilon.$$

- Máximo número de iteraciones: Se detiene el proceso cuando se supera un número preestablecido de iteraciones.

Estos criterios permiten controlar la precisión deseada y asegurar que la aproximación obtenida es suficientemente cercana a la raíz (Burden et al., 2016; Chapra & Canale, 2015).

8.5 Ventajas y desventajas

Ventajas:

- Conserva la propiedad de acotamiento del método de la bisección.
- Puede converger más rápido que la bisección en muchas funciones.
- No requiere cálculo de derivadas.

Desventajas:

- La convergencia puede ser muy lenta si uno de los extremos del intervalo queda “atrapado”.
- Usualmente tiene una convergencia lineal, más lenta que la del método de la secante o Newton-Raphson.

8.6 Resumen conceptual

El método de regula falsi representa un compromiso entre estabilidad y velocidad: mantiene la seguridad del acotamiento, pero intenta mejorar la rapidez usando una aproximación lineal. Por ello, es especialmente útil cuando se requiere una garantía estricta de que la raíz no se perderá, pero se desea evitar la lentitud del método de la bisección tradicional (Burden & Faires, 2011; Chapra & Canale, 2015; Cheney & Kincaid, 2009).

8.7 Aplicación del método en Python

El objetivo fue desarrollar un programa en Python que permita al usuario ingresar una función $f(x)$, graficarla en un intervalo definido y aplicar el método de Regula Falsi si así lo desea. El programa realiza iteraciones hasta que se cumpla un criterio de tolerancia, mostrando en cada paso los valores de a, b, c, $f(c)$ y el error estimado.

Entrada

- Una cadena de texto que representa la función matemática $f(x)$.
- Los valores iniciales a y b que delimitan el intervalo de búsqueda.
- El intervalo para graficar la función.

$$f(x) = x^3 - x - 1$$

Salida

- Gráfica de la función $f(x)$ en el intervalo ingresado.
- La raíz aproximada encontrada.
- Número de iteraciones realizadas hasta cumplir la tolerancia.

Restricciones

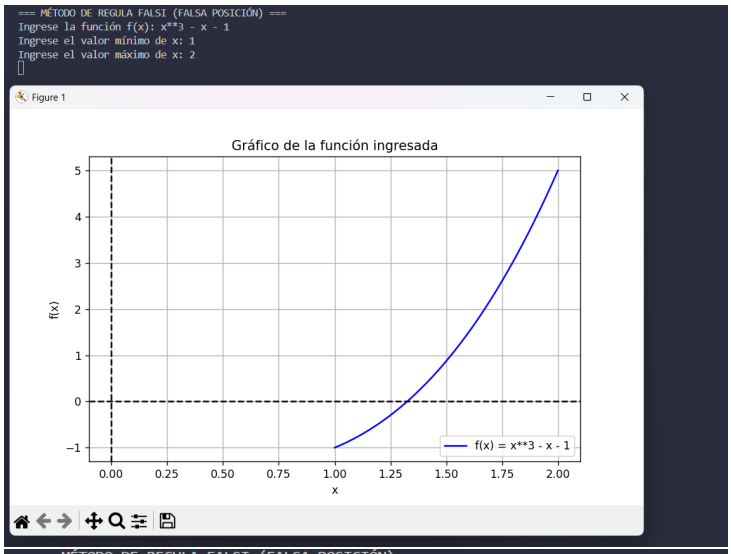
- La función debe ser continua en el intervalo $[a, b]$.
- Debe cumplirse la condición de cambio de signo $f(a)f(b) < 0$.

Código

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

# === 1. Ingreso de la función ===
print("=== MÉTODO DE REGULA FALSI (FALSA POSICIÓN) ===")
func_str = input("Ingrese la función f(x): ")  # Ejemplo: x**3 - x - 1

# Definición de la función
def f(x):
return eval(func_str, {"np": np, "x": x})

# === 2. Graficar la función ===
xmin = float(input("Ingrese el valor mínimo de x: "))
xmax = float(input("Ingrese el valor máximo de x: "))

x = np.linspace(xmin, xmax, 400)
y = f(x)

plt.figure(figsize=(8, 5))
plt.plot(x, y, label=f"f(x) = {func_str}", color='blue')
plt.axhline(0, color='black', linestyle='--')
plt.axvline(0, color='black', linestyle='--')
plt.title("Gráfico de la función ingresada")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.show()

# === 3. Pregunta si desea aplicar el método de Regula Falsi ===
op = input("¿Desea aplicar el método de Regula Falsi? (s/n): ").lower()

if op == "s":
# === 4. Ingreso de los extremos del intervalo ===
a = float(input("Ingrese el valor de a (extremo izquierdo): "))
b = float(input("Ingrese el valor de b (extremo derecho): "))

# Comprobación del cambio de signo
if f(a) * f(b) > 0:
print(" La función no cambia de signo en el intervalo. No se puede aplicar el
    ↪ método.")
else:
tol = 1e-6
max_iter = 100

print("\nIteración |      a      |      b      |      c      |    f(c)     |
    ↪ Error")
print("-------------------------------------------------------------------")

c_old = a
for i in range(1, max_iter + 1):
```

```
# Fórmula de Regula Falsi
c = b - (f(b) * (a - b)) / (f(a) - f(b))
error = abs(c - c_old)
c_old = c

print(f"{i:9d} | {a:10.6f} | {b:10.6f} | {c:10.6f} | {f(c):10.6f} | {error:10.6f}")

if abs(f(c)) < tol or error < tol:
print(f"\n Raíz aproximada encontrada: {c:.6f}")
print(f"Iteraciones realizadas: {i}")
break

# Actualización de intervalos
if f(a) * f(c) < 0:
b = c
else:
a = c
else:
print("\n No se alcanzó la convergencia después de", max_iter, "iteraciones.")
else:
print("No se aplicó el método de Regula Falsi.")
```

Ejecución




```
=== MÉTODO DE REGULA FALSI (FALSA POSICIÓN) ===
Ingrese la función f(x): x**3 - x - 1
Ingrese el valor mínimo de x: 1
Ingrese el valor máximo de x: 2
¿Desea aplicar el método de Regula Falsi? (s/n): s
Ingrese el valor de a (extremo izquierdo): 1
Ingrese el valor de b (extremo derecho): 2

Iteración |      a      |      b      |      c      |    f(c)     |   Error
-------------------------------------------------------------------------------
        1 |   1.000000  |   2.000000  |   1.166667  |  -0.578704  |  0.166667
        2 |   1.166667  |   2.000000  |   1.253112  |  -0.285363  |  0.086445
        3 |   1.253112  |   2.000000  |   1.293437  |  -0.129542  |  0.040325
        4 |   1.293437  |   2.000000  |   1.311281  |  -0.056588  |  0.017844
        5 |   1.311281  |   2.000000  |   1.318989  |  -0.024304  |  0.007707
        6 |   1.318989  |   2.000000  |   1.322283  |  -0.010362  |  0.003294
        7 |   1.322283  |   2.000000  |   1.323684  |  -0.004404  |  0.001402
        8 |   1.323684  |   2.000000  |   1.324279  |  -0.001869  |  0.000595
        9 |   1.324279  |   2.000000  |   1.324532  |  -0.000793  |  0.000253
       10 |   1.324532  |   2.000000  |   1.324639  |  -0.000336  |  0.000107
       11 |   1.324639  |   2.000000  |   1.324685  |  -0.000143  |  0.000045
       12 |   1.324685  |   2.000000  |   1.324704  |  -0.000060  |  0.000019
       13 |   1.324704  |   2.000000  |   1.324712  |  -0.000026  |  0.000008
       14 |   1.324712  |   2.000000  |   1.324715  |  -0.000011  |  0.000003
       15 |   1.324715  |   2.000000  |   1.324717  |  -0.000005  |  0.000001
       16 |   1.324717  |   2.000000  |   1.324717  |  -0.000002  |  0.000001

✅ Raíz aproximada encontrada: 1.324717
Iteraciones realizadas: 16
```

Figura 8.1: Método de Regula Falsi en Python

Parte II

Unidad II

9 Gradiente de una función

La gradiente es uno de los conceptos fundamentales del cálculo multivariable y del análisis numérico. Representa la dirección de mayor crecimiento de una función escalar y constituye una herramienta esencial en optimización, análisis de modelos matemáticos y métodos iterativos numéricos (Anton et al., 2012; Stewart et al., 2016). Su aplicación se extiende a la física, ingeniería, economía, aprendizaje automático y solución de ecuaciones diferenciales (Burden et al., 2016; Riley et al., 2006).

9.1 Definición

Sea una función escalar de varias variables

$$f(x_1, x_2, \ldots, x_n),$$

que es diferenciable en un punto del dominio. La gradiente de f, denotada por ∇f o grad f, se define como el vector cuyas componentes son las derivadas parciales primeras de f:

$$\nabla f(x_1, \ldots, x_n) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right).$$

Este vector constituye una generalización natural del concepto de derivada en una dimensión, extendiéndolo a espacios de dimensión superior (Anton et al., 2012; Sullivan, 2004).

9.2 Interpretación geométrica

El vector gradiente posee una interpretación geométrica fundamental:

$$\nabla f(\mathbf{x}) \text{ apunta en la dirección de máximo crecimiento de } f.$$

Además:

- La magnitud $\|\nabla f\|$ indica la tasa máxima de cambio.
- Es perpendicular (normal) a las curvas (o superficies) de nivel de f, es decir,

$$f(x_1, x_2, \ldots, x_n) = c.$$

Esta propiedad es clave en optimización y geometría diferencial, donde el análisis de superficies y sus normales es crucial (Anton et al., 2012; Riley et al., 2006).

9.3 Derivada direccional

La gradiente se relaciona directamente con la derivada direccional. Para un vector unitario $\mathbf{u}$, la derivada direccional de f en la dirección de $\mathbf{u}$ se define como:

$$D_{\mathbf{u}} f(\mathbf{x}) = \nabla f(\mathbf{x}) \cdot \mathbf{u}.$$

Esta expresión demuestra que la gradiente actúa como "coeficiente de cambio" de la función en cualquier dirección dada. De esta fórmula se deduce que $D_{\mathbf{u}} f$ es máximo cuando $\mathbf{u}$ es paralelo a ∇f, lo que reafirma la interpretación geométrica (Anton et al., 2012; Riley et al., 2006).

9.4 Propiedades fundamentales

- Linealidad:
$$\nabla(af + bg) = a\nabla f + b\nabla g.$$
- Producto:
$$\nabla(fg) = f\nabla g + g\nabla f.$$
- Cociente:
$$\nabla\left(\frac{f}{g}\right) = \frac{g\nabla f - f\nabla g}{g^2}.$$
- Composición (regla de la cadena):
$$\nabla(f \circ \mathbf{g}) = J_{\mathbf{g}}^T \nabla f(\mathbf{g}(x)),$$
donde $J_{\mathbf{g}}$ es la matriz Jacobiana.

Estas propiedades son esenciales en el diseño de algoritmos numéricos de optimización y métodos iterativos (Burden et al., 2016; Chapra & Canale, 2015).

9.5 Gradiente y optimización

El gradiente es el pilar de numerosos métodos de optimización, tanto teóricos como computacionales:

- Condición de óptimo: Un punto crítico $\mathbf{x}^*$ cumple
$$\nabla f(\mathbf{x}^*) = \mathbf{0}.$$
- Método de descenso del gradiente: El movimiento en dirección opuesta al gradiente,
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k),$$

permite aproximar mínimos locales de manera iterativa.

- Método de Newton multivariable: Utiliza gradiente y Hessiana:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - H_f^{-1}(\mathbf{x}_k)\nabla f(\mathbf{x}_k).$$

Estos métodos son fundamentales en ingeniería, estadística, ciencia de datos y aprendizaje automático (Burden et al., 2016; Chapra & Canale, 2015; Riley et al., 2006).

9.6 Aplicaciones

La gradiente aparece de forma natural en numerosas áreas:

9.6.1 Física clásica y electromagnetismo.

La fuerza conservativa se relaciona mediante:

$$\mathbf{F} = -\nabla U,$$

donde U es la energía potencial (Riley et al., 2006).

9.6.2 Ingeniería y modelado.

Gradientes se utilizan en:

- Modelos térmicos (flujo de calor).
- Dinámica de fluidos.
- Elasticidad de materiales.
- Procesos de control.

9.6.3 Aprendizaje automático y regresión.

Los algoritmos de entrenamiento, como regresión lineal, redes neuronales y SVM, utilizan métodos basados en gradiente para minimizar funciones de costo.

9.6.4 Economía y análisis de funciones multivariables.

El gradiente describe sensibilidades o tasas de variación respecto a variables económicas, como producción, precios o utilidades (Stewart et al., 2016).

9.6.5 Métodos numéricos.

En métodos iterativos para resolver ecuaciones no lineales y optimización se requiere calcular gradientes en cada iteración (Burden et al., 2016; Chapra & Canale, 2015).

9.7 Aplicación de la gradiente en R

El siguiente código en R simula el proceso del gradiente descendente para la función $f(x) = x^2$, cuya derivada es $f'(x) = 2x$. Se observa cómo los valores de x van disminuyendo gradualmente hasta aproximarse al punto mínimo en $x = 0$.

```r
# ------------------------------------------------------------
# Simulación y gráfico de f(x), f'(x) y gradiente en R
# grad guarda el "nuevo" valor de x: x(i) - n * f'(x(i))
# ------------------------------------------------------------

# Parámetro n
n <- 0.01

# Definimos la función f(x) = x^2 y su derivada f'(x) = 2x
f <- function(x) x^2
f_deriv <- function(x) 2 * x

# Valor inicial y número de iteraciones
x0 <- 3
iter <- 21

# Vectores vacíos
x <- numeric(iter)
fx <- numeric(iter)
fpx <- numeric(iter)
grad <- numeric(iter)

# Primer valor
x[1] <- x0

# Bucle de cálculo
for (i in 1:iter) {
    fx[i]  <- f(x[i])
    fpx[i] <- f_deriv(x[i])
    grad[i] <- x[i] - n * fpx[i]
    if (i < iter) {
        x[i + 1] <- grad[i]
    }
}

# Crear tabla
tabla <- data.frame(
xo  = x,
fx  = fx,
fpx = fpx,
grad = grad
)
print(tabla)

# Gráfico con ggplot2
if(!require(ggplot2)) install.packages("ggplot2", repos = "https://cloud.r-project.
```

```
    ↪ org")
if(!require(tidyr))  install.packages("tidyr",  repos = "https://cloud.r-project.
    ↪ org")

library(ggplot2)
library(tidyr)

datos_long <- tabla |>
pivot_longer(cols = c(fx, fpx, grad),
names_to = "variable",
values_to = "valor")

ggplot(datos_long, aes(x = xo, y = valor, color = variable)) +
geom_point(size = 2) +
geom_line(linewidth = 1) +
scale_color_manual(values = c("blue", "red", "green"),
labels = c("f(x)", "f'(x)", "gradiente (x actualizado)")) +
labs(title = "Comparación de f(x), f'(x) y gradiente (x actualizado)",
x = "x",
y = "Valor",
color = "Variable") +
theme_minimal(base_size = 13)
```

Ejecución

gradiente.R | datos_long | tabla

Filter

	xo	fx	fpx	grad
1	3.000000	9.000000	6.000000	2.940000
2	2.940000	8.643600	5.880000	2.881200
3	2.881200	8.301313	5.762400	2.823576
4	2.823576	7.972581	5.647152	2.767104
5	2.767104	7.656867	5.534209	2.711762
6	2.711762	7.353655	5.423525	2.657527
7	2.657527	7.062451	5.315054	2.604377
8	2.604377	6.782777	5.208753	2.552289
9	2.552289	6.514179	5.104578	2.501243
10	2.501243	6.256218	5.002487	2.451218
11	2.451218	6.008472	4.902437	2.402194
12	2.402194	5.770536	4.804388	2.354150
13	2.354150	5.542023	4.708300	2.307067
14	2.307067	5.322559	4.614134	2.260926
15	2.260926	5.111786	4.521852	2.215707
16	2.215707	4.909359	4.431415	2.171393
17	2.171393	4.714948	4.342786	2.127965
18	2.127965	4.528236	4.255931	2.085406
19	2.085406	4.348918	4.170812	2.043698
20	2.043698	4.176701	4.087396	2.002824
21	2.002824	4.011304	4.005648	1.962767

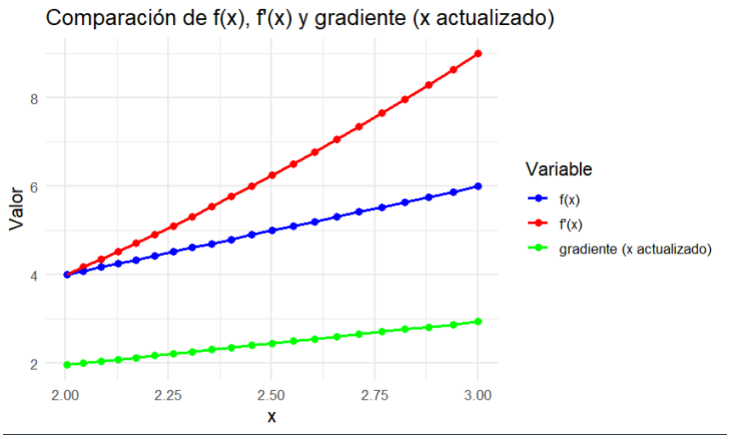


Figura 9.1: Gradiente de una función en R

9.8 Aplicación de la gradiente en un artículo de investigación

En el artículo Comparing the Moore-Penrose Pesudoinverse and Gradient Descent for Solving Linear Regression Problems: A Performance Analysis (Adams, 2025) se comparan dos métodos fundamentales para resolver problemas de regresión lineal: el pseudoinverso de Moore–Penrose y el descenso de gradiente. La regresión lineal busca encontrar los parámetros que mejor predicen una variable dependiente a partir de variables independientes, minimizando los errores cuadrados (OLS).

El pseudoinverso de Moore–Penrose ofrece una solución analítica exacta mediante operaciones matriciales, proporcionando el mínimo error posible. Sin embargo, puede ser computacionalmente costoso e inestable cuando la matriz de características es muy grande o está mal condicionada. En cambio, el descenso de gradiente es un método iterativo que ajusta los parámetros gradualmente en función de la dirección del gradiente negativo del error, controlado por una tasa de aprendizaje. Aunque no garantiza una solución exacta, es escalable y adaptable a grandes volúmenes de datos.

En el marco teórico, se explica que la estabilidad numérica y la eficiencia de cada método dependen del tamaño del conjunto de datos (n), el número de variables (d) y la condición de la matriz X. El pseudoinverso utiliza descomposición SVD, mientras que el descenso de gradiente requiere pasos controlados y puede verse afectado por la escala de las variables.

En la metodología, el autor realizó experimentos con datos sintéticos y reales. Los datos sintéticos permitieron manipular parámetros como el número de muestras, la dimensionalidad y el factor de condición. Para los datos reales se usaron los conjuntos California Housing (20,640 muestras y 8 variables) y Diabetes (442 muestras y 10 variables). En ambos casos se evaluaron tres métricas: tiempo de ejecución, error cuadrático medio (MSE) y, para el descenso de gradiente, el número de iteraciones hasta la convergencia. Los experimentos se implementaron en Python, usando la función `pinv()` para el pseudoinverso y un algoritmo de gradiente con tasa de aprendizaje $\alpha = 0.01$ y tolerancia 10^{-6}.

Los resultados mostraron que el pseudoinverso fue más rápido y preciso en conjuntos pequeños o moderados, manteniendo una alta estabilidad numérica. En cambio, el descenso de gradiente necesitó más iteraciones y su rendimiento dependió fuertemente del escalado de las variables y del valor de la tasa de aprendizaje. Sin embargo, en escenarios de alta dimensionalidad o grandes volúmenes de datos, el descenso de gradiente (y sus variantes como el estocástico) resultó más escalable y eficiente.

En conclusión, el pseudoinverso de Moore–Penrose es ideal para conjuntos de datos bien condicionados y de tamaño moderado, ofreciendo soluciones exactas con bajo error. El descenso de gradiente es preferible para problemas de gran escala, aunque requiere ajuste de hiperparámetros y un buen preprocesamiento. Ambos métodos son esenciales en el análisis de regresión, y su elección depende del equilibrio entre exactitud, estabilidad y costo computacional.

10 Diferenciación Numérica

10.1 Introducción

La diferenciación numérica es una herramienta fundamental en el análisis numérico y en la computación científica. Su objetivo es aproximar derivadas de funciones cuando no es posible obtenerlas de manera analítica, o cuando la función se conoce únicamente a través de datos discretos (Burden et al., 2016; Chapra & Canale, 2015). Este enfoque es ampliamente utilizado en ingeniería, física, análisis de datos, modelos matemáticos aplicados y simulaciones computacionales.

El problema central consiste en aproximar derivadas mediante expresiones que utilizan valores de la función evaluados en puntos cercanos. Dichas expresiones se obtienen mediante la expansión en series de Taylor y constituyen la base de los métodos de diferencias finitas (Anton et al., 2012; Riley et al., 2006).

10.2 Fundamento teórico

10.2.1 Series de Taylor

Sea f una función suficientemente diferenciable en un intervalo que contiene el punto x. La expansión de Taylor alrededor de x permite escribir:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f^{(3)}(x) + \cdots,$$

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

Estas expresiones permiten deducir fórmulas para aproximar derivadas y analizar el error local de truncamiento asociado a cada aproximación (Atkinson, 2009; Burden et al., 2016).

10.3 Fórmulas de diferencias finitas

Las fórmulas para aproximar derivadas se clasifican según la manera en que utilizan los valores de la función.

10.3.1 Diferencia hacia adelante

Usando la expansión de Taylor se obtiene la fórmula:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h},$$

con un error dado por:

$$E = O(h).$$

Esta aproximación es de primer orden y suele utilizarse por su simplicidad computacional, aunque es menos precisa que otras alternativas (Burden et al., 2016; Chapra & Canale, 2015).

10.3.2 Diferencia hacia atrás

De manera análoga, se obtiene:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h},$$

con el mismo error $O(h)$. Esta fórmula se utiliza cuando la evaluación hacia adelante no está disponible, como en métodos que avanzan en el tiempo y solo tienen datos previos (Chapra & Canale, 2015).

10.3.3 Diferencia centrada

Restando las expansiones de $f(x+h)$ y $f(x-h)$ se obtiene:

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

con un error:

$$E = O(h^2).$$

Esta fórmula es más precisa porque utiliza información simétrica alrededor de x. Es ampliamente preferida en aplicaciones científicas donde se busca alta precisión (Anton et al., 2012; Riley et al., 2006).

10.4 Aproximaciones para derivadas de orden superior

10.4.1 Segunda derivada

Sumando las expansiones de Taylor se obtiene:

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2},$$

con error:

$$E = O(h^2).$$

Esta expresión es fundamental en la solución numérica de ecuaciones diferenciales, problemas de vibraciones, transferencia de calor y modelos físicos discretizados (Chapra & Canale, 2015; Riley et al., 2006).

10.4.2 Derivadas superiores

Usando combinaciones de series de Taylor o matrices de diferencias finitas, es posible obtener fórmulas generales para derivadas de orden superior. Estas aproximaciones suelen presentar errores más sensibles al tamaño de paso y requieren mayor precisión numérica (Atkinson, 2009; Burden et al., 2016).

10.5 Análisis del error

10.5.1 Error de truncamiento

El error de truncamiento proviene de ignorar términos de orden superior en la serie de Taylor. Para una fórmula dada, el orden del método indica cómo disminuye el error cuando h se hace más pequeño. Por ejemplo:

- Diferencia hacia adelante: $O(h)$
- Diferencia centrada: $O(h^2)$
- Segunda derivada centrada: $O(h^2)$

El análisis de error es esencial para comprender la estabilidad y precisión de los métodos numéricos (Atkinson, 2009; Burden et al., 2016).

10.5.2 Error de redondeo

Cuando h es demasiado pequeño, el error por cancelación y redondeo puede aumentar significativamente debido a las limitaciones de la aritmética de punto flotante. Esto genera un equilibrio óptimo entre reducir h y mantener un nivel adecuado de estabilidad numérica (Chapra & Canale, 2015; Press et al., 2007).

10.6 Aplicaciones de la diferenciación numérica

10.6.1 Solución de ecuaciones diferenciales

Las discretizaciones basadas en derivadas numéricas se utilizan en:

- Ecuaciones diferenciales ordinarias (EDO).
- Ecuaciones diferenciales parciales (EDP).
- Modelos de dinámica de fluidos, calor, ondas y elasticidad.

Estas técnicas permiten transformar problemas continuos en sistemas algebraicos manejables computacionalmente (Press et al., 2007; Riley et al., 2006).

10.6.2 Optimización

Los métodos de optimización requieren derivadas para estimar direcciones de búsqueda, calcular gradientes y aproximar Hessianas. En muchos casos se emplean derivadas numéricas cuando la función no es diferenciable analíticamente (Burden et al., 2016).

10.6.3 Procesamiento de datos y señales

En análisis de datos discretos se utilizan derivadas numéricas para:

- detectar cambios bruscos,
- hallar picos en señales,
- calcular tasas de crecimiento,
- aproximar tendencias locales.

Estas aplicaciones son comunes en ingeniería, finanzas y ciencia experimental.

10.7 Ejemplos

10.7.1 Ejemplo 1

Un startup de tecnología registra el número acumulado de usuarios activos mensuales:

Mes	1	2	3	4	5	6	7
Usuarios (miles)	10	15	23	34	48	65	85

Tareas

1. Calcula la tasa de crecimiento de usuarios (usuarios nuevos por mes) en el mes 4 usando diferencia centrada.
2. Calcula la tasa de crecimiento en el mes 1 (usa diferencia adelante)
3. Calcula la tasa de crecimiento en el mes 7 (usa diferencia atrás)
4. ¿En qué mes fue mayor la aceleración del crecimiento? (calcula la sgeunda derivada en cada punto)
5. Interpreta: ¿la startup está creciendo de forma acelerada o desacelerada?

Datos:

Mes	1	2	3	4	5	6	7
Usuarios (miles)	10	15	23	34	48	65	85

Sea $h = 1$ mes (intervalo entre observaciones).

—

1) Tasa de crecimiento en el mes 4 (diferencia centrada)

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

Para $x = 4$:

$$f'(4) = \frac{f(5) - f(3)}{2} = \frac{48 - 23}{2} = 12.5$$

$$\boxed{f'(4) = 12.5 \text{ miles de usuarios/mes}}$$

—

2) Tasa de crecimiento en el mes 1 (diferencia hacia adelante)

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Para $x = 1$:

$$f'(1) = f(2) - f(1) = 15 - 10 = 5$$

$$\boxed{f'(1) = 5 \text{ miles de usuarios/mes}}$$

—

3) Tasa de crecimiento en el mes 7 (diferencia hacia atrás)

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

Para $x = 7$:

$$f'(7) = f(7) - f(6) = 85 - 65 = 20$$

$$\boxed{f'(7) = 20 \text{ miles de usuarios/mes}}$$

—

4) Aceleración del crecimiento (segunda derivada)

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Para los puntos interiores ($x = 2, 3, 4, 5, 6$):

$$\begin{aligned} f''(2) &= 23 - 2(15) + 10 = 3 \\ f''(3) &= 34 - 2(23) + 15 = 3 \\ f''(4) &= 48 - 2(34) + 23 = 3 \\ f''(5) &= 65 - 2(48) + 34 = 3 \\ f''(6) &= 85 - 2(65) + 48 = 3 \end{aligned}$$

$$\boxed{f''(x) = 3 \text{ miles de usuarios/mes}^2 \text{ (constante en los meses 2 a 6)}}$$

—

5) Interpretación

Como la segunda derivada es positiva y constante, la tasa de crecimiento aumenta de forma continua.

$$\boxed{\text{La startup crece de forma acelerada.}}$$

En términos prácticos, cada mes la tasa de nuevos usuarios aumenta en aproximadamente 3 mil usuarios/mes^2, lo que refleja un crecimiento sostenido y cada vez más rápido.

Código en R

```r
# Datos
mes <- 1:7
usuarios <- c(10, 15, 23, 34, 48, 65, 85)
h <- 1

# 1. Derivadas (tasa de crecimiento)
# Diferencia adelante, atrás y centrada
dif_adelante <- (usuarios[2] - usuarios[1]) / h
dif_atras <- (usuarios[7] - usuarios[6]) / h
dif_centrada <- (usuarios[5] - usuarios[3]) / (2*h)

cat("Tasa mes 1 (adelante):", dif_adelante, "miles/mes\n")
cat("Tasa mes 4 (centrada):", dif_centrada, "miles/mes\n")
cat("Tasa mes 7 (atrás):", dif_atras, "miles/mes\n")

# 2. Segunda derivada (aceleración) en puntos interiores
f2 <- numeric(length(usuarios))
for (i in 2:(length(usuarios)-1)) {
    f2[i] <- (usuarios[i+1] - 2*usuarios[i] + usuarios[i-1]) / (h^2)
}
f2

cat("\nSegunda derivada (aceleración) por mes:\n")
print(data.frame(mes=mes, aceleracion=f2))

# 3. Interpretación
if (all(f2[2:6] > 0)) {
    cat("\nLa aceleración es positiva → crecimiento acelerado.\n")
} else if (all(f2[2:6] < 0)) {
    cat("\nLa aceleración es negativa → crecimiento desacelerado.\n")
} else {
    cat("\nLa aceleración varía → crecimiento irregular.\n")
}
```

Ejecución

```
> # Datos
> mes <- 1:7
> usuarios <- c(10, 15, 23, 34, 48, 65, 85)
> h <- 1
>
> # 1. Derivadas (tasa de crecimiento)
> # Diferencia adelante, atrás y centrada
> dif_adelante <- (usuarios[2] - usuarios[1]) / h
> dif_atras <- (usuarios[7] - usuarios[6]) / h
> dif_centrada <- (usuarios[5] - usuarios[3]) / (2*h)
>
> cat("Tasa mes 1 (adelante):", dif_adelante, "miles/mes\n")
Tasa mes 1 (adelante): 5 miles/mes
> cat("Tasa mes 4 (centrada):", dif_centrada, "miles/mes\n")
Tasa mes 4 (centrada): 12.5 miles/mes
```

```
> cat("Tasa mes 7 (atrás):", dif_atras, "miles/mes\n")
Tasa mes 7 (atrás): 20 miles/mes
>
> # 2. Segunda derivada (aceleración) en puntos interiores
> f2 <- numeric(length(usuarios))
> for (i in 2:(length(usuarios)-1)) {
    +   f2[i] <- (usuarios[i+1] - 2*usuarios[i] + usuarios[i-1]) / (h^2)
    + }
> f2
[1] 0 3 3 3 3 3 0
>
> cat("\nSegunda derivada (aceleración) por mes:\n")

Segunda derivada (aceleración) por mes:
> print(data.frame(mes=mes, aceleracion=f2))
mes aceleracion
1   1           0
2   2           3
3   3           3
4   4           3
5   5           3
6   6           3
7   7           0
>
> # 3. Interpretación
> if (all(f2[2:6] > 0)) {
    +   cat("\nLa aceleración es positiva → crecimiento acelerado.\n")
    + } else if (all(f2[2:6] < 0)) {
    +   cat("\nLa aceleración es negativa → crecimiento desacelerado.\n")
    + } else {
    +   cat("\nLa aceleración varía → crecimiento irregular.\n")
    + }

La aceleración es positiva → crecimiento acelerado.
```

10.7.2 Ejemplo 2

Durante el entrenamiento de un modelo de Machine Learning, se registra la función de pérdida (loss) en cada época:

Época	0	10	20	30	40	50
Loss	2.45	1.82	1.35	1.08	0.95	0.89

Tareas

1. Calcula la tasa de cambio del loss en la época 20 usando diferencia centrada con h = 10.
2. Calcula la segunda derivada en la época 30. ¿Qué indica sobre la convergencia?

3. Si la tasa de cambio del loss es menor que 0.01 por época, el entrenamiento puede detenerse. ¿En qué época se alcanza este criterio?

4. Estima el loss en la época 25 usando interpolación lineal basada en las derivadas.

Datos:

Época	0	10	20	30	40	50
Loss	2.45	1.82	1.35	1.08	0.95	0.89

Sea $h = 10$ (épocas entre mediciones).

1) Tasa de cambio del loss en la época 20 (diferencia centrada, $h = 10$)

La diferencia centrada con paso h:

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

Para $x = 20$:

$$f'(20) \approx \frac{f(30) - f(10)}{2 \cdot 10} = \frac{1.08 - 1.82}{20} = \frac{-0.74}{20} = -0.037.$$

$$\boxed{f'(20) \approx -0.037 \text{ (loss por época)}}$$

(es decir, la loss decrece aproximadamente 0.037 por época).

2) Segunda derivada en la época 30 (centrada, $h = 10$)

Fórmula de segunda derivada centrada:

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

Para $x = 30$:

$$f''(30) \approx \frac{f(40) - 2f(30) + f(20)}{10^2} = \frac{0.95 - 2(1.08) + 1.35}{100} = \frac{0.14}{100} = 0.0014.$$

$$\boxed{f''(30) \approx 0.0014 \text{ (loss por época}^2\text{)}}$$

Interpretación: la segunda derivada es positiva y pequeña, lo que indica que la pendiente (tasa de decremento del loss) se está volviendo menos negativa: la disminución del loss se está frenando — es decir, el entrenamiento está tendiendo a converger (la curva se aplana).

3) ¿En qué época la tasa de cambio del loss es menor que 0.01 por época?

Calculamos las derivadas aproximadas en cada época usando diferencias hacia adelante/centradas/atrás con $h = 10$:

$$
\begin{aligned}
f'(0) &\approx \frac{f(10) - f(0)}{10} = \frac{1.82 - 2.45}{10} = -0.063 \\
f'(10) &\approx \frac{f(20) - f(0)}{20} = \frac{1.35 - 2.45}{20} = -0.055 \\
f'(20) &\approx -0.037 \quad \text{(ya calculado)} \\
f'(30) &\approx \frac{f(40) - f(20)}{20} = \frac{0.95 - 1.35}{20} = -0.020 \\
f'(40) &\approx \frac{f(50) - f(30)}{20} = \frac{0.89 - 1.08}{20} = -0.0095 \\
f'(50) &\approx \frac{f(50) - f(40)}{10} = \frac{0.89 - 0.95}{10} = -0.006
\end{aligned}
$$

El criterio es $|f'| < 0.01$. El primer punto donde se cumple es la $\boxed{\text{época 40}}$ (a partir de la época 40 la magnitud de la tasa cae por debajo de 0.01).

4) Estima el loss en la época 25 usando interpolación lineal basada en las derivadas

Usamos una aproximación lineal (expansión de primer orden) tomando el punto conocido en época 20 y la derivada en 20:

$$f(25) \approx f(20) + f'(20) \cdot (25 - 20).$$

Sustituyendo:

$$f(25) \approx 1.35 + (-0.037) \cdot 5 = 1.35 - 0.185 = 1.165.$$

$$\boxed{f(25) \approx 1.165}$$

Nota: la interpolación lineal clásica entre los puntos (20, 1.35) y (30, 1.08) daría el punto medio $(1.35 + 1.08)/2 = 1.215$. La estimación basada en la derivada (linealización en 20) da 1.165; ambas son aproximaciones distintas (la elección depende del método que prefieras).

Código en R

```r
# Datos
epoca <- c(0,10,20,30,40,50)
loss  <- c(2.45, 1.82, 1.35, 1.08, 0.95, 0.89)
h <- 10

# Funciones para derivadas (usando h fijo = 10)
# Diferencia hacia adelante (para primer punto)
d_forward <- function(i) {
    (loss[i+1] - loss[i]) / h
```

```
}
# Diferencia hacia atrás (para último punto)
d_backward <- function(i) {
    (loss[i] - loss[i-1]) / h
}
# Diferencia centrada (para puntos interiores)
d_center <- function(i) {
    (loss[i+1] - loss[i-1]) / (2*h)
}
# Segunda derivada centrada
d2_center <- function(i) {
    (loss[i+1] - 2*loss[i] + loss[i-1]) / (h^2)
}

# Calcular derivadas en cada época
deriv <- numeric(length(loss))
for (i in seq_along(loss)) {
    if (i == 1) deriv[i] <- d_forward(i)
    else if (i == length(loss)) deriv[i] <- d_backward(i)
    else deriv[i] <- d_center(i)
}

# Calcular segunda derivada en puntos interiores
d2 <- rep(NA, length(loss))
for (i in 2:(length(loss)-1)) {
    d2[i] <- d2_center(i)
}

# Resultados pedidos:
cat("Derivadas (loss por época):\n")
print(data.frame(epoca=epoca, loss=loss, derivada=round(deriv, 5)))

cat("\nSegunda derivada (solo puntos interiores):\n")
print(data.frame(epoca=epoca, segunda_derivada=round(d2, 6)))

# 1) Tasa en época 20 (índice de época 3)
idx20 <- which(epoca == 20)
cat("\n1) f'(20) (centrada, h=10):", round(deriv[idx20], 5), " (loss/época)\n")

# 2) Segunda derivada en época 30
idx30 <- which(epoca == 30)
cat("2) f''(30):", round(d2[idx30], 6), " (loss/época^2)\n")

# Interpretación (impresa)
if (!is.na(d2[idx30])) {
    if (d2[idx30] > 0) {
        cat("   Interpretación: f''(30) > 0 → la magnitud de la pendiente se está
    ↪ reduciendo; la disminución del loss se está frenando (convergencia en curso)
    ↪ .\n")
    } else if (d2[idx30] < 0) {
        cat("   Interpretación: f''(30) < 0 → la disminución se está acelerando.\n")
    } else {
```

```
        cat("   Interpretación: f''(30)  0 → curva cercana a lineal localmente.\n")
    }
}

# 3) ¿En qué época |f'| < 0.01 ?
idx_crit <- which(abs(deriv) < 0.01)
if (length(idx_crit) > 0) {
    cat("\n3) Épocas donde |f'| < 0.01:", epoca[idx_crit], "\n")
    cat("   Primera época que cumple el criterio:", epoca[min(idx_crit)], "\n")
} else {
    cat("\n3) No se alcanza |f'| < 0.01 en las épocas dadas.\n")
}

# 4) Estimación de loss en época 25 usando linearización en época 20:
t_target <- 25
t0 <- 20
f_t0 <- loss[which(epoca == t0)]
fprime_t0 <- deriv[which(epoca == t0)]
f25_est <- f_t0 + fprime_t0 * (t_target - t0)
cat("\n4) Estimación lineal (usando derivada en 20): f(25)  ", round(f25_est, 4), "\
    ↪ n")

# Para comparación: interpolación lineal directa entre 20 y 30
f20 <- f_t0
f30 <- loss[which(epoca == 30)]
f25_lin_between <- f20 + (f30 - f20) * ( (t_target - 20) / (30 - 20) )
cat("   Interpolación lineal entre 20 y 30: f(25)  ", round(f25_lin_between,4), "\n"
    ↪ )
```

Ejecución

```
> # Datos
> epoca <- c(0,10,20,30,40,50)
> loss  <- c(2.45, 1.82, 1.35, 1.08, 0.95, 0.89)
> h <- 10
>
> # Funciones para derivadas (usando h fijo = 10)
> # Diferencia hacia adelante (para primer punto)
> d_forward <- function(i) {
    +   (loss[i+1] - loss[i]) / h
    + }
> # Diferencia hacia atrás (para último punto)
> d_backward <- function(i) {
    +   (loss[i] - loss[i-1]) / h
    + }
> # Diferencia centrada (para puntos interiores)
> d_center <- function(i) {
    +   (loss[i+1] - loss[i-1]) / (2*h)
    + }
> # Segunda derivada centrada
> d2_center <- function(i) {
    +   (loss[i+1] - 2*loss[i] + loss[i-1]) / (h^2)
```

```
    + }
>
> # Calcular derivadas en cada época
> deriv <- numeric(length(loss))
> for (i in seq_along(loss)) {
    +   if (i == 1) deriv[i] <- d_forward(i)
    +   else if (i == length(loss)) deriv[i] <- d_backward(i)
    +   else deriv[i] <- d_center(i)
    + }
>
> # Calcular segunda derivada en puntos interiores
> d2 <- rep(NA, length(loss))
> for (i in 2:(length(loss)-1)) {
    +   d2[i] <- d2_center(i)
    + }
>
> # Resultados pedidos:
> cat("Derivadas (loss por época):\n")
Derivadas (loss por época):
> print(data.frame(epoca=epoca, loss=loss, derivada=round(deriv, 5)))
epoca loss derivada
1     0 2.45  -0.0630
2    10 1.82  -0.0550
3    20 1.35  -0.0370
4    30 1.08  -0.0200
5    40 0.95  -0.0095
6    50 0.89  -0.0060
>
> cat("\nSegunda derivada (solo puntos interiores):\n")

Segunda derivada (solo puntos interiores):
> print(data.frame(epoca=epoca, segunda_derivada=round(d2, 6)))
epoca segunda_derivada
1     0               NA
2    10           0.0016
3    20           0.0020
4    30           0.0014
5    40           0.0007
6    50               NA
>
> # 1) Tasa en época 20 (índice de época 3)
> idx20 <- which(epoca == 20)
> cat("\n1) f'(20) (centrada, h=10):", round(deriv[idx20], 5), " (loss/época)\n")

1) f'(20) (centrada, h=10): -0.037  (loss/época)
>
> # 2) Segunda derivada en época 30
> idx30 <- which(epoca == 30)
> cat("2) f''(30):", round(d2[idx30], 6), " (loss/época^2)\n")
2) f''(30): 0.0014  (loss/época^2)
>
> # Interpretación (impresa)
```

```
> if (!is.na(d2[idx30])) {
    +   if (d2[idx30] > 0) {
        +     cat("   Interpretación: f''(30) > 0 → la magnitud de la pendiente se
    ↪ está reduciendo; la disminución del loss se está frenando (convergencia en
    ↪ curso).\n")
        +   } else if (d2[idx30] < 0) {
        +     cat("   Interpretación: f''(30) < 0 → la disminución se está
    ↪ acelerando.\n")
        +   } else {
        +     cat("   Interpretación: f''(30)  0 → curva cercana a lineal
    ↪ localmente.\n")
        +   }
    + }
Interpretación: f''(30) > 0 → la magnitud de la pendiente se está reduciendo; la
    ↪ disminución del loss se está frenando (convergencia en curso).
>
> # 3) ¿En qué época |f'| < 0.01 ?
> idx_crit <- which(abs(deriv) < 0.01)
> if (length(idx_crit) > 0) {
    +   cat("\n3) Épocas donde |f'| < 0.01:", epoca[idx_crit], "\n")
    +   cat("   Primera época que cumple el criterio:", epoca[min(idx_crit)], "\n")
    + } else {
    +   cat("\n3) No se alcanza |f'| < 0.01 en las épocas dadas.\n")
    + }

3) Épocas donde |f'| < 0.01: 40 50
Primera época que cumple el criterio: 40
>
> # 4) Estimación de loss en época 25 usando linearización en época 20:
> t_target <- 25
> t0 <- 20
> f_t0 <- loss[which(epoca == t0)]
> fprime_t0 <- deriv[which(epoca == t0)]
> f25_est <- f_t0 + fprime_t0 * (t_target - t0)
> cat("\n4) Estimación lineal (usando derivada en 20): f(25)  ", round(f25_est, 4),
    ↪ "\n")

4) Estimación lineal (usando derivada en 20): f(25)   1.165
>
> # Para comparación: interpolación lineal directa entre 20 y 30
> f20 <- f_t0
> f30 <- loss[which(epoca == 30)]
> f25_lin_between <- f20 + (f30 - f20) * ( (t_target - 20) / (30 - 20) )
> cat("   Interpolación lineal entre 20 y 30: f(25)  ", round(f25_lin_between,4), "\
    ↪ n")
Interpolación lineal entre 20 y 30: f(25)   1.215
```

10.7.3 Ejemplo 3

Una empresa de e-commerce registra sus ventas diarias (en miles de dólares) durante una semana de campaña:

Día	Lun	Mar	Mié	Jue	Vie	sáb	Dom
Ventas ($k)	45	52	61	58	73	89	95

Tareas

1. calcula la velocidad de crecimiento de ventas (derivada) para cada día usando diferencias finitas apropiadas.
2. Identifica el día con mayor aceleración de ventas (segunda derivada positiva máxima)
3. El jueves hubo una caída en la tendencia. Calcula la magnitud de esta desaceleración.
4. Si la tendencia del domingo se mantiene, ¿Cuántas ventas esperarías el lunes siguiente? (usa extrapolación lineal con la derivada)

Datos:

Día	Lun	Mar	Mié	Jue	Vie	Sáb	Dom
Ventas ($\$k$)	45	52	61	58	73	89	95

Sea $h = 1$ día.

—

1) Velocidad de crecimiento (primera derivada) — diferencias finitas

Usamos diferencia hacia adelante en el primer día, hacia atrás en el último, y centrada en los puntos interiores:

$$f'(1)\ \text{(Lun, adelante)} = \frac{f(2) - f(1)}{1} = 52 - 45 = 7$$
$$f'(2)\ \text{(Mar, centrada)} = \frac{f(3) - f(1)}{2} = \frac{61 - 45}{2} = 8$$
$$f'(3)\ \text{(Mié, centrada)} = \frac{f(4) - f(2)}{2} = \frac{58 - 52}{2} = 3$$
$$f'(4)\ \text{(Jue, centrada)} = \frac{f(5) - f(3)}{2} = \frac{73 - 61}{2} = 6$$
$$f'(5)\ \text{(Vie, centrada)} = \frac{f(6) - f(4)}{2} = \frac{89 - 58}{2} = 15.5$$
$$f'(6)\ \text{(Sáb, centrada)} = \frac{f(7) - f(5)}{2} = \frac{95 - 73}{2} = 11$$
$$f'(7)\ \text{(Dom, atrás)} = \frac{f(7) - f(6)}{1} = 95 - 89 = 6$$

$$\boxed{f' = [7,\ 8,\ 3,\ 6,\ 15.5,\ 11,\ 6]\ \ (\$k \text{ por día})}$$

—

2) Aceleración (segunda derivada) y día con mayor aceleración positiva

Segunda derivada centrada para puntos interiores:

$$f''(x) \approx f(x+1) - 2f(x) + f(x-1)$$

Cálculos:

$$f''(\text{Mar}) = 61 - 2(52) + 45 = 2$$
$$f''(\text{Mié}) = 58 - 2(61) + 52 = -12$$
$$f''(\text{Jue}) = 73 - 2(58) + 61 = 18$$
$$f''(\text{Vie}) = 89 - 2(73) + 58 = 1$$
$$f''(\text{Sáb}) = 95 - 2(89) + 73 = -10$$

$$\boxed{f'' = [\text{NA},\ 2,\ -12,\ 18,\ 1,\ -10,\ \text{NA}]\ \ (\$k/\text{día}^2)}$$

La segunda derivada máxima positiva es $f''(\text{Jue}) = 18$, por lo que $\boxed{\text{Jueves}}$ es el día con mayor aceleración positiva de ventas (magnitud 18 $\$k/\text{día}^2$).

—

3) Caída en la tendencia el jueves — magnitud de la desaceleración

Hay dos formas de cuantificar lo pedido:

- Caída directa en ventas (valor absoluto): las ventas bajaron de miércoles a jueves: $58-61 = -3$. Es decir, hubo una caída de \$3k en ventas ese día.
- Desaceleración medida por la segunda derivada: el valor $f''(\text{Mié}) = -12$ indica una fuerte desaceleración alrededor del miércoles (la pendiente se hace más negativa allí). En cambio $f''(\text{Jue}) = +18$ porque inmediatamente después (viernes) las ventas suben mucho (73), lo que provoca una aceleración positiva en el punto jueves cuando se evalúa con el vecino viernes.

Conclusión: la caída instantánea de ventas el jueves fue $\boxed{3\ \$k}$. La desaceleración local más fuerte corresponde a $f''(\text{Mié}) = -12$ ($\$k/\text{día}^2$), indicando que justo en torno al miércoles la tendencia se frenó fuertemente.

—

4) Extrapolación lineal: ventas el lunes siguiente

Usamos la derivada en el domingo (diferencia atrás) como tasa diaria y extrapolamos 1 día:

$$f(\text{Lun siguiente}) \approx f(\text{Dom}) + f'(\text{Dom}) \cdot 1 = 95 + 6 = 101$$

$$\boxed{\text{Ventas esperadas el lunes siguiente } \approx 101\ \$k}$$

—

Resumen numérico rápido:

$$\begin{aligned} f' &: [7,\ 8,\ 3,\ 6,\ 15.5,\ 11,\ 6]\ (\$k/\text{día}) \\ f'' &: [\text{NA},\ 2,\ -12,\ 18,\ 1,\ -10,\ \text{NA}]\ (\$k/\text{día}^2) \\ \text{Mayor aceleración positiva} &: \text{Jue}\ (18\ \$k/\text{día}^2) \\ \text{Caída en Jue (ventas)} &: -3\ \$k \\ \text{Proyección Lun siguiente} &: 101\ \$k \end{aligned}$$

Código en R

```r
# Datos
dias <- c("Lun","Mar","Mié","Jue","Vie","Sáb","Dom")
ventas <- c(45, 52, 61, 58, 73, 89, 95)  # en $k
h <- 1

# 1) Primera derivada: adelante en primer punto, atrás en último, centrada en interiores
deriv <- numeric(length(ventas))
for (i in seq_along(ventas)) {
    if (i == 1) {
        deriv[i] <- (ventas[i+1] - ventas[i]) / h
    } else if (i == length(ventas)) {
        deriv[i] <- (ventas[i] - ventas[i-1]) / h
    } else {
        deriv[i] <- (ventas[i+1] - ventas[i-1]) / (2*h)
    }
}

# 2) Segunda derivada (centrada) en puntos interiores
d2 <- rep(NA, length(ventas))
for (i in 2:(length(ventas)-1)) {
    d2[i] <- (ventas[i+1] - 2*ventas[i] + ventas[i-1]) / (h^2)
}

# Identificar día con mayor aceleración positiva
idx_max_acc <- which.max(d2)  # returns first max (NA treated as -Inf)
# but ensure it's interior (non-NA)
if (is.na(d2[idx_max_acc]) || length(idx_max_acc) == 0) {
    dia_max_acc <- NA
    max_acc <- NA
} else {
    dia_max_acc <- dias[idx_max_acc]
    max_acc <- d2[idx_max_acc]
}
```

```
34
35 # 3) Magnitud de la caída el jueves (drop Mié -> Jue)
36 drop_jue <- ventas[4] - ventas[3]  # index 4 = Jue, 3 = Mié
37
38 # 4) Extrapolación lineal para Lunes siguiente usando derivada en Domingo
39 f_dom <- ventas[length(ventas)]
40 fprime_dom <- deriv[length(ventas)]
41 f_lun_sig <- f_dom + fprime_dom * 1
42
43 # Mostrar resultados
44 cat("Primera derivada (ventas $k/día) por día:\n")
45 print(data.frame(dia=dias, ventas=ventas, derivada=deriv))
46
47 cat("\nSegunda derivada (ventas $k/día^2) en puntos interiores:\n")
48 print(data.frame(dia=dias, segunda_derivada=d2))
49
50 cat("\nDía con mayor aceleración positiva:\n")
51 cat("  ", dia_max_acc, "con f'' =", max_acc, "\n")
52
53 cat("\nCaída en ventas Mié -> Jue: ", drop_jue, " $k (si es negativo, es caída)\n")
54
55 cat("\nProyección lunes siguiente (usando derivada en domingo):\n")
56 cat("  Ventas domingo =", f_dom, " $k\n")
57 cat("  Tasa domingo =", fprime_dom, " $k/día\n")
58 cat("  Estimación lunes siguiente =", f_lun_sig, " $k\n")
```

Ejecución

```
1  > # Datos
2  > dias <- c("Lun","Mar","Mié","Jue","Vie","Sáb","Dom")
3  > ventas <- c(45, 52, 61, 58, 73, 89, 95)  # en $k
4  > h <- 1
5  >
6  > # 1) Primera derivada: adelante en primer punto, atrás en último, centrada en
       ↪ interiores
7  > deriv <- numeric(length(ventas))
8  > for (i in seq_along(ventas)) {
9      +   if (i == 1) {
10     +     deriv[i] <- (ventas[i+1] - ventas[i]) / h
11     +   } else if (i == length(ventas)) {
12     +     deriv[i] <- (ventas[i] - ventas[i-1]) / h
13     +   } else {
14     +     deriv[i] <- (ventas[i+1] - ventas[i-1]) / (2*h)
15     +   }
16     + }
17 >
18 > # 2) Segunda derivada (centrada) en puntos interiores
19 > d2 <- rep(NA, length(ventas))
20 > for (i in 2:(length(ventas)-1)) {
21     +   d2[i] <- (ventas[i+1] - 2*ventas[i] + ventas[i-1]) / (h^2)
22     + }
23 >
```

```
> # Identificar día con mayor aceleración positiva
> idx_max_acc <- which.max(d2)  # returns first max (NA treated as -Inf)
> # but ensure it's interior (non-NA)
> if (is.na(d2[idx_max_acc]) || length(idx_max_acc) == 0) {
    +   dia_max_acc <- NA
    +   max_acc <- NA
    + } else {
    +   dia_max_acc <- dias[idx_max_acc]
    +   max_acc <- d2[idx_max_acc]
    + }
>
> # 3) Magnitud de la caída el jueves (drop Mié -> Jue)
> drop_jue <- ventas[4] - ventas[3]  # index 4 = Jue, 3 = Mié
>
> # 4) Extrapolación lineal para Lunes siguiente usando derivada en Domingo
> f_dom <- ventas[length(ventas)]
> fprime_dom <- deriv[length(ventas)]
> f_lun_sig <- f_dom + fprime_dom * 1
>
> # Mostrar resultados
> cat("Primera derivada (ventas $k/día) por día:\n")
Primera derivada (ventas $k/día) por día:
> print(data.frame(dia=dias, ventas=ventas, derivada=deriv))
dia ventas derivada
1 Lun     45      7.0
2 Mar     52      8.0
3 Mié     61      3.0
4 Jue     58      6.0
5 Vie     73     15.5
6 Sáb     89     11.0
7 Dom     95      6.0
>
> cat("\nSegunda derivada (ventas $k/día^2) en puntos interiores:\n")

Segunda derivada (ventas $k/día^2) en puntos interiores:
> print(data.frame(dia=dias, segunda_derivada=d2))
dia segunda_derivada
1 Lun               NA
2 Mar                2
3 Mié              -12
4 Jue               18
5 Vie                1
6 Sáb              -10
7 Dom               NA
>
> cat("\nDía con mayor aceleración positiva:\n")

Día con mayor aceleración positiva:
> cat("  ", dia_max_acc, "con f'' =", max_acc, "\n")
Jue con f'' = 18
>
> cat("\nCaída en ventas Mié -> Jue: ", drop_jue, " $k (si es negativo, es caída)\n
```

```
    ↪ ")
76
77 Caída en ventas Mié -> Jue:  -3  $k (si es negativo, es caída)
78 >
79 > cat("\nProyección lunes siguiente (usando derivada en domingo):\n")
80
81 Proyección lunes siguiente (usando derivada en domingo):
82 > cat("  Ventas domingo =", f_dom, " $k\n")
83 Ventas domingo = 95  $k
84 > cat("  Tasa domingo =", fprime_dom, " $k/día\n")
85 Tasa domingo = 6  $k/día
86 > cat("  Estimación lunes siguiente =", f_lun_sig, " $k\n")
87 Estimación lunes siguiente = 101  $k
```

10.7.4 Ejemplo 4

En redes neuronales, necesitas calcular el gradiente de la función sigmoide $\sigma(x) = \dfrac{1}{1+e^{-x}}$ en varios puntos para el backpropagation. No tienes la derivada analítica programada.

Datos evaluados de la función:

x	-3.0	-2.0	-1.0	0.0	1.0	2.0	3.0
$\sigma(x)$	0.0474	0.1192	0.2689	0.5000	0.7311	0.8808	0.9526

Tareas

1. Calcula $\sigma'(0)$ usando diferencia centrada con h = 1.
2. Calcula $\sigma'(-2)$ y $\sigma'(2)$ usando diferencias centradas.
3. Compara tus resultados numéricos con la derivada analítica: $\sigma'(x) = \sigma(x)(1-\sigma(x))$
4. ¿Qué tamaño de h recomendarías para este cálculo? Justifica.
5. ¿Por qué la derivada es simétrica alrededor de x = 0?

Datos:

x	-3.0	-2.0	-1.0	0.0	1.0	2.0	3.0
$\sigma(x)$	0.0474	0.1192	0.2689	0.5000	0.7311	0.8808	0.9526

Sea $h = 1$.

—

1) Cálculo numérico de $\sigma'(0)$ (diferencia centrada, $h = 1$)

$$\sigma'(x) \approx \frac{\sigma(x+h) - \sigma(x-h)}{2h}.$$

Para $x = 0$:

$$\sigma'(0) \approx \frac{\sigma(1) - \sigma(-1)}{2} = \frac{0.7311 - 0.2689}{2} = \frac{0.4622}{2} = 0.2311.$$

$$\boxed{\sigma'(0)_{\text{num}} \approx 0.2311}$$

—

2) Cálculo numérico de $\sigma'(-2)$ y $\sigma'(2)$ (centradas, $h = 1$)

Para $x = -2$:

$$\sigma'(-2) \approx \frac{\sigma(-1) - \sigma(-3)}{2} = \frac{0.2689 - 0.0474}{2} = \frac{0.2215}{2} = 0.11075.$$

Para $x = 2$:

$$\sigma'(2) \approx \frac{\sigma(3) - \sigma(1)}{2} = \frac{0.9526 - 0.7311}{2} = \frac{0.2215}{2} = 0.11075.$$

$$\boxed{\sigma'(-2)_{\text{num}} = \sigma'(2)_{\text{num}} \approx 0.11075}$$

—

3) Comparación con la derivada analítica

La derivada analítica es

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big).$$

Calculando:

$$\begin{aligned}
\sigma'(0)_{\text{ana}} &= 0.5(1 - 0.5) = 0.25, \\
\sigma'(-2)_{\text{ana}} &= 0.1192(1 - 0.1192) = 0.1192 \cdot 0.8808 \approx 0.10499136, \\
\sigma'(2)_{\text{ana}} &= 0.8808(1 - 0.8808) = 0.8808 \cdot 0.1192 \approx 0.10499136.
\end{aligned}$$

Errores absolutos y relativos:

$$\begin{aligned}
\text{En } x = 0: &\quad \text{num} = 0.2311,\ \text{ana} = 0.25,\ \Delta = -0.0189\ (\approx -7.56\,\%) \\
\text{En } x = \pm 2: &\quad \text{num} = 0.11075,\ \text{ana} \approx 0.10499136,\ \Delta \approx 0.00575864\ (\approx +5.48\,\%)
\end{aligned}$$

Comentario: las aproximaciones centradas con $h = 1$ están cerca pero muestran errores de varios por ciento; la diferencia es mayor en $x = 0$ porque la curvatura local y la escala hacen que un paso grande (1) afecte más la precisión.

—

4) Recomendación sobre el tamaño de h

- Si puedes evaluar la función analíticamente (como la sigmoide), usa un h muy pequeño para las diferencias finitas — por ejemplo $h \approx 10^{-5}$ a 10^{-6} suele ser razonable en doble precisión, o usar la fórmula analítica directamente (lo ideal).
- Si solo dispones de valores tabulados con separación discreta (aquí los datos están a paso 1), entonces el único h disponible para diferencias finitas sin interpolar es $h = 1$. Para mejorar la precisión necesitarías interpolar o ajustar una función (p. ej. ajustar la sigmoide) y luego derivar esa función.
- Regla práctica (análisis numérico): el error truncamiento $O(h^2)$ para la diferencia centrada, pero el error de redondeo aumenta al tomar h muy pequeño. Si usas la función analítica, elegir h en torno a $\sqrt{\varepsilon} \cdot$ escala (donde ε es la máquina, 2.2e-16) da un buen compromiso; en práctica $h \approx 10^{-5}$ suele funcionar bien.

—

5) ¿Por qué la derivada es simétrica alrededor de $x = 0$?

La sigmoide cumple la relación

$$\sigma(-x) = 1 - \sigma(x).$$

Entonces

$$\sigma'(-x) = \sigma(-x)\big(1 - \sigma(-x)\big) = (1 - \sigma(x))\big(1 - (1 - \sigma(x))\big) = \sigma(x)(1 - \sigma(x)) = \sigma'(x).$$

Por tanto $\sigma'(x)$ es una función par (even): $\sigma'(-x) = \sigma'(x)$. Esa es la razón de la simetría alrededor de $x = 0$.

Código en R

```r
# Datos
x <- c(-3, -2, -1, 0, 1, 2, 3)
sigma <- c(0.0474, 0.1192, 0.2689, 0.5000, 0.7311, 0.8808, 0.9526)
h <- 1

# Indices helper
idx <- function(val) which(x == val)

# 1) Derivada centrada para x = 0, -2, 2
deriv_centered <- function(xval) {
    i <- idx(xval)
```

```
    # asumiendo que xval tiene vecinos (x-h) y (x+h) disponibles
    (sigma[i + 1] - sigma[i - 1]) / (2 * h)
}

d0_num <- deriv_centered(0)
dminus2_num <- deriv_centered(-2)
d2_num <- deriv_centered(2)

# 2) Derivada analítica sigma*(1-sigma)
d_analytic <- sigma * (1 - sigma)
d0_ana <- d_analytic[idx(0)]
dminus2_ana <- d_analytic[idx(-2)]
d2_ana <- d_analytic[idx(2)]

# 3) Errores
abs_err_0 <- d0_num - d0_ana
rel_err_0 <- abs_err_0 / d0_ana * 100

abs_err_m2 <- dminus2_num - dminus2_ana
rel_err_m2 <- abs_err_m2 / dminus2_ana * 100

abs_err_2 <- d2_num - d2_ana
rel_err_2 <- abs_err_2 / d2_ana * 100

# Mostrar resultados
cat("Resultados (h = 1):\n\n")

cat("x = 0:\n")
cat("  derivada num (centrada) =", d0_num, "\n")
cat("  derivada analítica      =", d0_ana, "\n")
cat("  error absoluto =", abs_err_0, ", error relativo (%) =", rel_err_0, "\n\n")

cat("x = -2:\n")
cat("  derivada num (centrada) =", dminus2_num, "\n")
cat("  derivada analítica      =", dminus2_ana, "\n")
cat("  error absoluto =", abs_err_m2, ", error relativo (%) =", rel_err_m2, "\n\n")

cat("x = 2:\n")
cat("  derivada num (centrada) =", d2_num, "\n")
cat("  derivada analítica      =", d2_ana, "\n")
cat("  error absoluto =", abs_err_2, ", error relativo (%) =", rel_err_2, "\n\n")

# Tabla resumen
res <- data.frame(
x = x,
sigma = sigma,
deriv_analitica = round(d_analytic, 8),
deriv_centrada = c(NA, dminus2_num, NA, d0_num, NA, d2_num, NA)
)
cat("Tabla resumen:\n")
print(res)

```

```
# Recomendación sobre h (comentario)
cat("\nNota: si puedes evaluar sigma(x) analíticamente, usa h pequeño (p.ej. 1e-5)
    ↪ o mejor aún usa la derivada analítica.\n")
```

Ejecución

```
> # Datos
> x <- c(-3, -2, -1, 0, 1, 2, 3)
> sigma <- c(0.0474, 0.1192, 0.2689, 0.5000, 0.7311, 0.8808, 0.9526)
> h <- 1
>
> # Indices helper
> idx <- function(val) which(x == val)
>
> # 1) Derivada centrada para x = 0, -2, 2
> deriv_centered <- function(xval) {
    +   i <- idx(xval)
    +   # asumiendo que xval tiene vecinos (x-h) y (x+h) disponibles
    +   (sigma[i + 1] - sigma[i - 1]) / (2 * h)
    + }
>
> d0_num <- deriv_centered(0)
> dminus2_num <- deriv_centered(-2)
> d2_num <- deriv_centered(2)
>
> # 2) Derivada analítica sigma*(1-sigma)
> d_analytic <- sigma * (1 - sigma)
> d0_ana <- d_analytic[idx(0)]
> dminus2_ana <- d_analytic[idx(-2)]
> d2_ana <- d_analytic[idx(2)]
>
> # 3) Errores
> abs_err_0 <- d0_num - d0_ana
> rel_err_0 <- abs_err_0 / d0_ana * 100
>
> abs_err_m2 <- dminus2_num - dminus2_ana
> rel_err_m2 <- abs_err_m2 / dminus2_ana * 100
>
> abs_err_2 <- d2_num - d2_ana
> rel_err_2 <- abs_err_2 / d2_ana * 100
>
> # Mostrar resultados
> cat("Resultados (h = 1):\n\n")
Resultados (h = 1):

>
> cat("x = 0:\n")
x = 0:
> cat("  derivada num (centrada) =", d0_num, "\n")
derivada num (centrada) = 0.2311
> cat("  derivada analítica      =", d0_ana, "\n")
derivada analítica      = 0.25
```

```
> cat("  error absoluto =", abs_err_0, ", error relativo (%) =", rel_err_0, "\n\n")
error absoluto = -0.0189 , error relativo (%) = -7.56

>
> cat("x = -2:\n")
x = -2:
> cat("  derivada num (centrada) =", dminus2_num, "\n")
derivada num (centrada) = 0.11075
> cat("  derivada analítica      =", dminus2_ana, "\n")
derivada analítica      = 0.1049914
> cat("  error absoluto =", abs_err_m2, ", error relativo (%) =", rel_err_m2, "\n\n
    ↪ ")
error absoluto = 0.00575864 , error relativo (%) = 5.48487

>
> cat("x = 2:\n")
x = 2:
> cat("  derivada num (centrada) =", d2_num, "\n")
derivada num (centrada) = 0.11075
> cat("  derivada analítica      =", d2_ana, "\n")
derivada analítica      = 0.1049914
> cat("  error absoluto =", abs_err_2, ", error relativo (%) =", rel_err_2, "\n\n")
error absoluto = 0.00575864 , error relativo (%) = 5.48487

>
> # Tabla resumen
> res <- data.frame(
+   x = x,
+   sigma = sigma,
+   deriv_analitica = round(d_analytic, 8),
+   deriv_centrada = c(NA, dminus2_num, NA, d0_num, NA, d2_num, NA)
+ )
> cat("Tabla resumen:\n")
Tabla resumen:
> print(res)
x  sigma deriv_analitica deriv_centrada
1 -3 0.0474      0.04515324             NA
2 -2 0.1192      0.10499136        0.11075
3 -1 0.2689      0.19659279             NA
4  0 0.5000      0.25000000        0.23110
5  1 0.7311      0.19659279             NA
6  2 0.8808      0.10499136        0.11075
7  3 0.9526      0.04515324             NA
>
> # Recomendación sobre h (comentario)
> cat("\nNota: si puedes evaluar sigma(x) analíticamente, usa h pequeño (p.ej. 1e
    ↪ -5) o mejor aún usa la derivada analítica.\n")

Nota: si puedes evaluar sigma(x) analíticamente, usa h pequeño (p.ej. 1e-5) o mejor
    ↪  aún usa la derivada analítica.
```

10.7.5 Ejemplo 5

Un sistema de monitoreo registra el tiempo de respuesta (en ms) de una API cada hora durante un incidente:

Hora	0	1	2	3	4	5	6	7
Latencia (ms)	120	125	128	135	280	290	275	155

Tareas

1. Calcula la primera derivada (tasa de cambio) para cada hora usando diferencias finitas apropiadas.

2. Identifica el momento del "pico de anomalía calculando dónde la segunda derivada cambia de signo (de positiva a negativa)

3. Entre las horas 3 y 4 hay un salto brusco. Calcula la magnitud de este cambio.

4. A partir de la hora 6, el sistema comienza a recuperarse. Calcula la tasa de recuperación (derivada negativa)

5. Si defines una anomalía como un cambio mayor a 50 ms/hora, ¿en qué momentos se detectarían anomalías?

Datos:

Hora	0	1	2	3	4	5	6	7
Latencia (ms)	120	125	128	135	280	290	275	155

Sea $h = 1$ hora.

—

1) Primera derivada (tasa de cambio) — diferencias finitas

Usamos diferencia hacia adelante en la hora 0, hacia atrás en la hora 7 y centrada en los puntos interiores:

$$
\begin{aligned}
f'(0) &= \frac{f(1)-f(0)}{1} = 125-120 = 5\\
f'(1) &= \frac{f(2)-f(0)}{2} = \frac{128-120}{2} = 4\\
f'(2) &= \frac{f(3)-f(1)}{2} = \frac{135-125}{2} = 5\\
f'(3) &= \frac{f(4)-f(2)}{2} = \frac{280-128}{2} = 76\\
f'(4) &= \frac{f(5)-f(3)}{2} = \frac{290-135}{2} = 77.5\\
f'(5) &= \frac{f(6)-f(4)}{2} = \frac{275-280}{2} = -2.5\\
f'(6) &= \frac{f(7)-f(5)}{2} = \frac{155-290}{2} = -67.5\\
f'(7) &= \frac{f(7)-f(6)}{1} = 155-275 = -120
\end{aligned}
$$

$$\boxed{f' = [5,\ 4,\ 5,\ 76,\ 77.5,\ -2.5,\ -67.5,\ -120]\ \ \text{(ms/h)}}$$

—

2) Pico de anomalía (cambio de signo en la segunda derivada)

Segunda derivada centrada (para puntos interiores):

$$f''(x) \approx f(x+1) - 2f(x) + f(x-1)$$

Cálculos:

$$
\begin{aligned}
f''(1) &= 128 - 2(125) + 120 = -2\\
f''(2) &= 135 - 2(128) + 125 = 4\\
f''(3) &= 280 - 2(135) + 128 = 138\\
f''(4) &= 290 - 2(280) + 135 = -135\\
f''(5) &= 275 - 2(290) + 280 = -25\\
f''(6) &= 155 - 2(275) + 290 = -105
\end{aligned}
$$

La segunda derivada pasa de positiva (en $x = 3$, $f''(3) = 138$) a negativa (en $x = 4$, $f''(4) = -135$). Por tanto el pico de anomalía ocurre entre la $\boxed{\text{hora 3 y hora 4}}$, interpretándose el momento máximo del choque en la $\boxed{\text{hora 4}}$ (cuando la latencia alcanza 280 ms).

—

3) Magnitud del salto brusco entre hora 3 y 4

$$\Delta = f(4) - f(3) = 280 - 135 = 145 \text{ ms.}$$

$$\boxed{\text{Salto brusco} = 145 \text{ ms}}$$

—

4) Tasa de recuperación a partir de la hora 6

Usando la derivada calculada:

- Derivada centrada en hora 6: $f'(6) = -67.5$ ms/h — tasa de recuperación promedio alrededor de la hora 6. - Entre hora 6 y 7 (derivada hacia atrás en 7): $f'(7) = -120$ ms/h — recuperación más rápida en ese intervalo.

$$\boxed{\text{Tasa de recuperación (aprox.)} \approx -67.5 \text{ ms/h (cent.) ; entre } 6\to7\text{: } -120 \text{ ms/h}}$$

—

5) Detección de anomalías (|cambio| > 50 ms/h)

Tomando los valores absolutos de f':

$$|f'| = [5,\ 4,\ 5,\ 76,\ 77.5,\ 2.5,\ 67.5,\ 120]$$

(he puesto 2.5 positivo para la magnitud en la hora 5).

Los instantes donde $|f'| > 50$ son las horas: $\boxed{3,\ 4,\ 6,\ 7}$ — es decir, se detectarían anomalías a partir de la hora 3 (inicio del ascenso brusco), en la hora 4 (máximo del salto), y durante la recuperación rápida en las horas 6 y 7.

—

Resumen:

$f' = [5, 4, 5, 76, 77.5, -2.5, -67.5, -120]$ (ms/h)

f''(interiores) $= [-2, 4, 138, -135, -25, -105]$ (ms/h^2)

Pico de anomalía: entre h=3 y h=4 (latencia máxima en h=4)

Salto 3→4: 145 ms

Tasa recuperación (h 6): -67.5 ms/h (cent.) ; 6→7: -120 ms/h

Anomalías (|f'|>50): horas $3, 4, 6, 7$

Código en R

```r
# Código R para calcular automáticamente lo anterior

# Datos
hora <- 0:7
latencia <- c(120, 125, 128, 135, 280, 290, 275, 155)
h <- 1
```

```r

# 1) Primera derivada: adelante, centrada, atrás
fprime <- numeric(length(latencia))
for (i in seq_along(latencia)) {
    if (i == 1) {
        fprime[i] <- (latencia[i+1] - latencia[i]) / h          # forward
    } else if (i == length(latencia)) {
        fprime[i] <- (latencia[i] - latencia[i-1]) / h          # backward
    } else {
        fprime[i] <- (latencia[i+1] - latencia[i-1]) / (2*h)    # centered
    }
}

# 2) Segunda derivada centrada (puntos interiores)
f2 <- rep(NA, length(latencia))
for (i in 2:(length(latencia)-1)) {
    f2[i] <- (latencia[i+1] - 2*latencia[i] + latencia[i-1]) / (h^2)
}

# 3) Magnitud salto 3->4
salto_3_4 <- latencia[5] - latencia[4]  # indices: hora 4 is index 5, hora 3 is
    ↪ index 4

# 4) Tasas de recuperación (a partir de la hora 6)
idx6 <- which(hora == 6)
tasa_recuperacion_cent <- fprime[idx6]
# derivada entre 6->7 (forward from 6 or backward at 7)
tasa_6_7 <- (latencia[8] - latencia[7]) / h  # index 8=hora7, 7=hora6

# 5) Detección de anomalías: |f'| > 50 ms/h
umbral <- 50
anomalias_idx <- which(abs(fprime) > umbral)
anomalias_horas <- hora[anomalias_idx]

# Imprimir resultados
cat("Primera derivada f' (ms/h) por hora:\n")
print(data.frame(hora=hora, latencia=latencia, fprime=round(fprime,4)))

cat("\nSegunda derivada f'' (ms/h^2) en puntos interiores:\n")
print(data.frame(hora=hora, f2=f2))

cat("\nSalto brusco 3->4 (ms):", salto_3_4, "\n")

cat("\nTasa de recuperación (hora 6, centrada):", tasa_recuperacion_cent, "ms/h\n")
cat("Tasa entre 6->7:", tasa_6_7, "ms/h\n")

if (length(anomalias_idx) > 0) {
    cat("\nAnomalías detectadas (|f'| >", umbral, "ms/h) en horas:", anomalias_
    ↪ horas, "\n")
} else {
    cat("\nNo se detectaron anomalías con el umbral dado.\n")
}
```

Ejecución

```
> # Código R para calcular automáticamente lo anterior
>
> # Datos
> hora <- 0:7
> latencia <- c(120, 125, 128, 135, 280, 290, 275, 155)
> h <- 1
>
> # 1) Primera derivada: adelante, centrada, atrás
> fprime <- numeric(length(latencia))
> for (i in seq_along(latencia)) {
    +   if (i == 1) {
        +     fprime[i] <- (latencia[i+1] - latencia[i]) / h          # forward
        +   } else if (i == length(latencia)) {
        +     fprime[i] <- (latencia[i] - latencia[i-1]) / h          # backward
        +   } else {
        +     fprime[i] <- (latencia[i+1] - latencia[i-1]) / (2*h)    # centered
        +   }
    + }
>
> # 2) Segunda derivada centrada (puntos interiores)
> f2 <- rep(NA, length(latencia))
> for (i in 2:(length(latencia)-1)) {
    +   f2[i] <- (latencia[i+1] - 2*latencia[i] + latencia[i-1]) / (h^2)
    + }
>
> # 3) Magnitud salto 3->4
> salto_3_4 <- latencia[5] - latencia[4]  # indices: hora 4 is index 5, hora 3 is
    ↪ index 4
>
> # 4) Tasas de recuperación (a partir de la hora 6)
> idx6 <- which(hora == 6)
> tasa_recuperacion_cent <- fprime[idx6]
> # derivada entre 6->7 (forward from 6 or backward at 7)
> tasa_6_7 <- (latencia[8] - latencia[7]) / h  # index 8=hora7, 7=hora6
>
> # 5) Detección de anomalías: |f'| > 50 ms/h
> umbral <- 50
> anomalias_idx <- which(abs(fprime) > umbral)
> anomalias_horas <- hora[anomalias_idx]
>
> # Imprimir resultados
> cat("Primera derivada f' (ms/h) por hora:\n")
Primera derivada f' (ms/h) por hora:
> print(data.frame(hora=hora, latencia=latencia, fprime=round(fprime,4)))
hora latencia fprime
1    0      120    5.0
2    1      125    4.0
3    2      128    5.0
4    3      135   76.0
5    4      280   77.5
6    5      290   -2.5
```

```
7    6      275  -67.5
8    7      155 -120.0
>
> cat("\nSegunda derivada f'' (ms/h^2) en puntos interiores:\n")

Segunda derivada f'' (ms/h^2) en puntos interiores:
> print(data.frame(hora=hora, f2=f2))
hora   f2
1    0   NA
2    1   -2
3    2    4
4    3  138
5    4 -135
6    5  -25
7    6 -105
8    7   NA
>
> cat("\nSalto brusco 3->4 (ms):", salto_3_4, "\n")

Salto brusco 3->4 (ms): 145
>
> cat("\nTasa de recuperación (hora 6, centrada):", tasa_recuperacion_cent, "ms/h\n
    ↪ ")

Tasa de recuperación (hora 6, centrada): -67.5 ms/h
> cat("Tasa entre 6->7:", tasa_6_7, "ms/h\n")
Tasa entre 6->7: -120 ms/h
>
> if (length(anomalias_idx) > 0) {
    +   cat("\nAnomalías detectadas (|f'| >", umbral, "ms/h) en horas:", anomalias_
    ↪ horas, "\n")
    + } else {
    +   cat("\nNo se detectaron anomalías con el umbral dado.\n")
    + }

Anomalías detectadas (|f'| > 50 ms/h) en horas: 3 4 6 7
```

10.7.6 Ejemplo 6

Una campaña de marketing muestra la siguiente tasa de conversión (porcentaje) en función del gasto en publicidad (en miles de dólares):

Gasto ($k)	0	5	10	15	20	25
Conversión (%)	2.1	3.8	5.2	6.1	6.7	7.0

Tareas

1. Calcula el ROI marginal (derivada de conversión respecto al gasto) en cada punto usando diferencias centradas.

2. ¿En qué momento de gasto el ROI marginal es mayor que 0.2 % por cada $1000 invertido?

3. La segunda derivada indica rendimientos decrecientes. Calcula la segunda derivada en $15k.

4. con base en las derivadas ¿recomendarías aumentar el gasto más allá de $25k? Justifique matemáticamente.

Datos:

Gasto ($\$k$)	0	5	10	15	20	25
Conversión (%)	2.1	3.8	5.2	6.1	6.7	7.0

Sea $h = 5$ (miles de dólares entre puntos).

—

1) ROI marginal (derivada de conversión respecto al gasto)

Usamos diferencia centrada para los puntos interiores y diferencias adelante/atrás en los extremos (paso $h = 5$):

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

Cálculos (en % por $1k):

$$\begin{aligned}
f'(0)\ \text{(adelante)} &= \frac{3.8 - 2.1}{5} = 0.34 \\
f'(5)\ \text{(centrada)} &= \frac{5.2 - 2.1}{10} = 0.31 \\
f'(10)\ \text{(centrada)} &= \frac{6.1 - 3.8}{10} = 0.23 \\
f'(15)\ \text{(centrada)} &= \frac{6.7 - 5.2}{10} = 0.15 \\
f'(20)\ \text{(centrada)} &= \frac{7.0 - 6.1}{10} = 0.09 \\
f'(25)\ \text{(atrás)} &= \frac{7.0 - 6.7}{5} = 0.06
\end{aligned}$$

$$\boxed{f' = [0.34,\ 0.31,\ 0.23,\ 0.15,\ 0.09,\ 0.06]\ \ (\%\ \text{por}\ \$1\text{k})}$$

—

2) ¿Cuándo el ROI marginal > 0.2 % por $1k?

Comparando los valores anteriores, las posiciones donde $f'(x) > 0.2$ son:

$$x = 0,\ 5,\ 10\ (\$0k,\ \$5k,\ \$10k).$$

Es decir, hasta \$10k el ROI marginal supera $0.2\,\%$/\$1k. Si se desea el punto exacto donde $f'(x) = 0.2$ entre 10 y 15k, interpolando linealmente entre $f'(10) = 0.23$ y $f'(15) = 0.15$ obtenemos:

$$x^* \approx 10 + \frac{0.23 - 0.20}{0.23 - 0.15} \cdot 5 \approx 11.875\ (\$k).$$

Por tanto, para gasto mayor a $\approx$ \$11.88k el ROI marginal cae por debajo de $0.2\,\%$ por \$1k.

—

3) Segunda derivada en \$15k (indicador de rendimientos decrecientes)

La fórmula centrada de segunda derivada:

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

Para $x = 15$ (con $h = 5$):

$$f''(15) \approx \frac{6.7 - 2(6.1) + 5.2}{5^2} = \frac{6.7 - 12.2 + 5.2}{25} = \frac{-0.3}{25} = -0.012.$$

$$\boxed{f''(15) \approx -0.012\ \left(\,\% \ /\ (\$1\text{k})^2\right)}$$

El valor negativo confirma rendimientos decrecientes (la ganancia marginal de conversión disminuye con gasto).

—

4) Recomendación sobre aumentar gasto más allá de \$25k (interpretación matemática)

- La primera derivada en \$25k es $f'(25) = 0.06\,\%$ por \$1k — positiva pero muy pequeña. - La segunda derivada en los puntos interiores es negativa (ej. $f''(15) \approx -0.012$), lo que indica que el ROI marginal está decreciendo al aumentar el gasto: $f'(\cdot)$ es decreciente. - Además, el umbral práctico que propusimos ($0.2\,\%$/\$1k) ya se supera sólo hasta $\approx$ \$11.88k; más allá de eso la ganancia marginal cae por debajo de un valor atractivo.

Por estas razones matemáticas (ROI marginal pequeño y decreciente), no es recomendable aumentar el gasto mucho más allá de \$25k si el objetivo es maximizar la conversión por dólar adicional.

Sin embargo, la decisión final puede depender de razones externas (p. ej. límites de escala, objetivos de branding, o expectativa de nonlinealidades fuera del rango observado).

—

Resumen numérico

$$f' : [0.34,\ 0.31,\ 0.23,\ 0.15,\ 0.09,\ 0.06]\,\%/\$1\text{k}$$

$$f''(15) \approx -0.012\,\%/(\$1\text{k})^2$$

Umbral $0.2\,\%/\$1\text{k}$ alcanzado hasta \$10k (corte $\approx$ \$11.875k)

Recomendación : No aumentar significativamente más allá de \$25k (rendimientos decrecientes).

Código en R

```r
# Código R: cálculo automático de derivadas y decisiones
gasto <- c(0, 5, 10, 15, 20, 25)   # en $k
conv  <- c(2.1, 3.8, 5.2, 6.1, 6.7, 7.0)  # en %
h <- 5  # en $k

# 1) Derivadas (ROI marginal) - forward/back/centered
deriv <- numeric(length(conv))
for (i in seq_along(conv)) {
    if (i == 1) {
        deriv[i] <- (conv[i+1] - conv[i]) / h    # forward
    } else if (i == length(conv)) {
        deriv[i] <- (conv[i] - conv[i-1]) / h    # backward
    } else {
        deriv[i] <- (conv[i+1] - conv[i-1]) / (2*h)  # centered
    }
}

# 2) Segunda derivada centrada (interiores)
d2 <- rep(NA, length(conv))
for (i in 2:(length(conv)-1)) {
    d2[i] <- (conv[i+1] - 2*conv[i] + conv[i-1]) / (h^2)
}

# 3) ¿Dónde f'(x) > 0.2 % por $1k?
idx_gt_02 <- which(deriv > 0.2)
gasto_gt_02 <- gasto[idx_gt_02]

# punto aproximado donde f'(x)=0.2 (lineal entre 10 y 15 si desea precisión)
d10 <- deriv[which(gasto==10)]
d15 <- deriv[which(gasto==15)]
if (d10 != d15) {
    frac <- (d10 - 0.2) / (d10 - d15)
    x_threshold <- 10 + frac * (15 - 10)  # en $k
} else {
    x_threshold <- NA
}

# Impresión de resultados
```

```
cat("Gasto ($k)  :", gasto, "\n")
cat("Conversion % :", conv, "\n\n")
cat("ROI marginal f' (% per $1k):\n")
print(round(deriv, 4))

cat("\nSegunda derivada (interiores) f'' (% per ($1k)^2):\n")
print(round(d2, 5))

cat("\nPuntos donde f' > 0.2 (% per $1k):", gasto_gt_02, "\n")
cat("Umbral f'(x)=0.2 ocurre aproximadamente en gasto = ", round(x_threshold, 3), "
    ↪  $k\n\n")

# Recomendación basada en derivadas
cat("Interpretación:\n")
cat("- f'(25) =", round(deriv[length(deriv)],4), "%/ $1k  (marginal positivo pero
    ↪ bajo)\n")
cat("- f'' en interiores (ej. f''(15) =", round(d2[which(gasto==15)],5), ") indica
    ↪ rendimientos decrecientes\n")
cat("--> Matemáticamente no conviene aumentar demasiado el gasto más allá de 25k si
    ↪  la meta es obtener conversion % por dolar adicional.\n")
```

Ejecución

```
> # Código R: cálculo automático de derivadas y decisiones
> gasto <- c(0, 5, 10, 15, 20, 25)   # en $k
> conv  <- c(2.1, 3.8, 5.2, 6.1, 6.7, 7.0)  # en %
> h <- 5  # en $k
>
> # 1) Derivadas (ROI marginal) - forward/back/centered
> deriv <- numeric(length(conv))
> for (i in seq_along(conv)) {
    +   if (i == 1) {
        +     deriv[i] <- (conv[i+1] - conv[i]) / h    # forward
        +   } else if (i == length(conv)) {
        +     deriv[i] <- (conv[i] - conv[i-1]) / h    # backward
        +   } else {
        +     deriv[i] <- (conv[i+1] - conv[i-1]) / (2*h)  # centered
        +   }
    + }
>
> # 2) Segunda derivada centrada (interiores)
> d2 <- rep(NA, length(conv))
> for (i in 2:(length(conv)-1)) {
    +   d2[i] <- (conv[i+1] - 2*conv[i] + conv[i-1]) / (h^2)
    + }
>
> # 3) ¿Dónde f'(x) > 0.2 % por $1k?
> idx_gt_02 <- which(deriv > 0.2)
> gasto_gt_02 <- gasto[idx_gt_02]
>
> # punto aproximado donde f'(x)=0.2 (lineal entre 10 y 15 si desea precisión)
> d10 <- deriv[which(gasto==10)]
```

```
> d15 <- deriv[which(gasto==15)]
> if (d10 != d15) {
    +   frac <- (d10 - 0.2) / (d10 - d15)
    +   x_threshold <- 10 + frac * (15 - 10)  # en $k
    + } else {
    +   x_threshold <- NA
    + }
>
> # Impresión de resultados
> cat("Gasto ($k)  :", gasto, "\n")
Gasto ($k)  : 0 5 10 15 20 25
> cat("Conversion % :", conv, "\n\n")
Conversion % : 2.1 3.8 5.2 6.1 6.7 7

> cat("ROI marginal f' (% per $1k):\n")
ROI marginal f' (% per $1k):
> print(round(deriv, 4))
[1] 0.34 0.31 0.23 0.15 0.09 0.06
>
> cat("\nSegunda derivada (interiores) f'' (% per ($1k)^2):\n")

Segunda derivada (interiores) f'' (% per ($1k)^2):
> print(round(d2, 5))
[1]     NA -0.012 -0.020 -0.012 -0.012     NA
>
> cat("\nPuntos donde f' > 0.2 (% per $1k):", gasto_gt_02, "\n")

Puntos donde f' > 0.2 (% per $1k): 0 5 10
> cat("Umbral f'(x)=0.2 ocurre aproximadamente en gasto = ", round(x_threshold, 3),
    ↪  " $k\n\n")
Umbral f'(x)=0.2 ocurre aproximadamente en gasto =  11.875  $k


>
> # Recomendación basada en derivadas
> cat("Interpretación:\n")
Interpretación:
> cat("- f'(25) =", round(deriv[length(deriv)],4), "%/ $1k  (marginal positivo pero
    ↪  bajo)\n")
- f'(25) = 0.06 %/ $1k  (marginal positivo pero bajo)
> cat("- f'' en interiores (ej. f''(15) =", round(d2[which(gasto==15)],5), ")
    ↪ indica rendimientos decrecientes\n")
- f'' en interiores (ej. f''(15) = -0.012 ) indica rendimientos decrecientes
> cat("--> Matemáticamente no conviene aumentar demasiado el gasto más allá de 25k
    ↪ si la meta es obtener conversion % por dolar adicional.\n")
--> Matemáticamente no conviene aumentar demasiado el gasto más allá de 25k si la
    ↪ meta es obtener conversion % por dolar adicional.
```

10.7.7 Ejemplo 7

Tienes un dataset con la señal de un sensor de temperatura en un proceso industrial medida cada segundo:

Tiempo (s)	0	1	2	3	4	5	6	7
Temp (°C)	20.1	20.3	20.8	21.5	22.6	24.2	26.1	28.5

Tareas

1. Crea una nueva feature: la velocidad de cambio de temperatura (primera derivada en cada punto.

2. Crea otra feature: la aceleración del cambio (segunda derivada)

3. Un aumento de temperatura mayor a 0.8°C/s indica un problema. ¿En qué momentos se detectaría esta alerta?

4. Normaliza las features derivadas usando min-max scaling.

5. Explica por qué estas features derivadas pueden ser útiles para un modelo de clasificación que detecta anomalías.

Datos:

Tiempo (s)	0	1	2	3	4	5	6	7
$T(C)$	20.1	20.3	20.8	21.5	22.6	24.2	26.1	28.5

Sea $h = 1\,\text{s}$.

—

1) Velocidad de cambio de temperatura (primera derivada)

$$T'(t) \approx \frac{T(t+h) - T(t-h)}{2h}$$

(con diferencias centradas; adelante y atrás en los extremos)

$$T'(0) = \frac{20.3 - 20.1}{1} = 0.2$$
$$T'(1) = \frac{20.8 - 20.1}{2} = 0.35$$
$$T'(2) = \frac{21.5 - 20.3}{2} = 0.6$$
$$T'(3) = \frac{22.6 - 20.8}{2} = 0.9$$
$$T'(4) = \frac{24.2 - 21.5}{2} = 1.35$$
$$T'(5) = \frac{26.1 - 22.6}{2} = 1.75$$
$$T'(6) = \frac{28.5 - 24.2}{2} = 2.15$$
$$T'(7) = \frac{28.5 - 26.1}{1} = 2.4$$

$$\boxed{T' = [0.2,\ 0.35,\ 0.6,\ 0.9,\ 1.35,\ 1.75,\ 2.15,\ 2.4]\ °\text{C/s}}$$

—

2) Aceleración del cambio (segunda derivada)

$$T''(t) \approx \frac{T(t+h) - 2T(t) + T(t-h)}{h^2}$$

$$T''(1) = 20.8 - 2(20.3) + 20.1 = 0.3$$
$$T''(2) = 21.5 - 2(20.8) + 20.3 = 0.2$$
$$T''(3) = 22.6 - 2(21.5) + 20.8 = 0.4$$
$$T''(4) = 24.2 - 2(22.6) + 21.5 = 0.5$$
$$T''(5) = 26.1 - 2(24.2) + 22.6 = 0.3$$
$$T''(6) = 28.5 - 2(26.1) + 24.2 = 0.5$$

$$\boxed{T'' = [\text{NA},\ 0.3,\ 0.2,\ 0.4,\ 0.5,\ 0.3,\ 0.5,\ \text{NA}]}$$

—

3) Alertas de temperatura (cuando $T'(t) > 0.8\ C/s$)

$$T'(t) > 0.8 \Rightarrow t = 3, 4, 5, 6, 7$$

$$\boxed{\text{Alerta desde } t = 3 \text{ s hasta } t = 7 \text{ s}}$$

—

4) Normalización (Min-Max Scaling)

Para cada feature derivada x_i:

$$x' = \frac{x_i - \text{mín}(x)}{\text{máx}(x) - \text{mín}(x)}$$

$$\text{mín}(T') = 0.2,\ \text{máx}(T') = 2.4$$

$$T'_{norm} = \frac{T' - 0.2}{2.4 - 0.2} = \frac{T' - 0.2}{2.2}$$

$$\Rightarrow T'_{norm} = [0.0,\ 0.068,\ 0.182,\ 0.318,\ 0.523,\ 0.682,\ 0.875,\ 1.0]$$

Para la segunda derivada (ignorando extremos):

$$\text{mín}(T'') = 0.2,\ \text{máx}(T'') = 0.5$$

$$T''_{norm} = \frac{T'' - 0.2}{0.3} \Rightarrow [\text{NA},\ 0.33,\ 0.0,\ 0.67,\ 1.0,\ 0.33,\ 1.0,\ \text{NA}]$$

—

5) Interpretación (utilidad en modelos de clasificación)

Las features derivadas (T' y T'') capturan no solo el nivel absoluto de la temperatura, sino también:

- **Cambios rápidos (gradiente):** ayudan a detectar aumentos anómalos incluso si la temperatura absoluta sigue dentro del rango normal. - **Patrones de aceleración:** la segunda derivada permite distinguir un incremento sostenido (problema) de una fluctuación puntual. - En modelos de clasificación (por ejemplo, detección de anomalías o mantenimiento predictivo), estas características enriquecen la representación temporal, permitiendo al modelo aprender dinámicas en lugar de solo estados estáticos.

Las derivadas actúan como detectores de tendencia y estabilidad del sistema.

Código en R

```r
# Datos
tiempo <- 0:7
temp <- c(20.1, 20.3, 20.8, 21.5, 22.6, 24.2, 26.1, 28.5)
h <- 1

# 1. Primera derivada (diferencias finitas)
vel <- numeric(length(temp))
```

```r
for (i in seq_along(temp)) {
    if (i == 1) vel[i] <- (temp[i+1] - temp[i]) / h
    else if (i == length(temp)) vel[i] <- (temp[i] - temp[i-1]) / h
    else vel[i] <- (temp[i+1] - temp[i-1]) / (2*h)
}

# 2. Segunda derivada
acc <- rep(NA, length(temp))
for (i in 2:(length(temp)-1)) {
    acc[i] <- (temp[i+1] - 2*temp[i] + temp[i-1]) / (h^2)
}

# 3. Alertas (cuando velocidad > 0.8 °C/s)
alerta <- tiempo[vel > 0.8]

# 4. Normalización min-max
minmax <- function(x) (x - min(x, na.rm=TRUE)) / (max(x, na.rm=TRUE) - min(x, na.rm
    ↪ =TRUE))
vel_norm <- minmax(vel)
acc_norm <- minmax(acc)

# Mostrar resultados
df <- data.frame(tiempo, temp, vel, acc, vel_norm, acc_norm)
print(round(df, 3))

cat("\nMomentos con alerta (>0.8°C/s):", alerta, "segundos\n")
```

Ejecución

```
> # Datos
> tiempo <- 0:7
> temp <- c(20.1, 20.3, 20.8, 21.5, 22.6, 24.2, 26.1, 28.5)
> h <- 1
>
> # 1. Primera derivada (diferencias finitas)
> vel <- numeric(length(temp))
> for (i in seq_along(temp)) {
    +   if (i == 1) vel[i] <- (temp[i+1] - temp[i]) / h
    +   else if (i == length(temp)) vel[i] <- (temp[i] - temp[i-1]) / h
    +   else vel[i] <- (temp[i+1] - temp[i-1]) / (2*h)
    + }
>
> # 2. Segunda derivada
> acc <- rep(NA, length(temp))
> for (i in 2:(length(temp)-1)) {
    +   acc[i] <- (temp[i+1] - 2*temp[i] + temp[i-1]) / (h^2)
    + }
>
> # 3. Alertas (cuando velocidad > 0.8 °C/s)
> alerta <- tiempo[vel > 0.8]
>
> # 4. Normalización min-max
```

```
> minmax <- function(x) (x - min(x, na.rm=TRUE)) / (max(x, na.rm=TRUE) - min(x, na.
    ↪ rm=TRUE))
> vel_norm <- minmax(vel)
> acc_norm <- minmax(acc)
>
> # Mostrar resultados
> df <- data.frame(tiempo, temp, vel, acc, vel_norm, acc_norm)
> print(round(df, 3))
tiempo temp  vel acc vel_norm acc_norm
1      0 20.1 0.20  NA    0.000       NA
2      1 20.3 0.35 0.3    0.068    0.333
3      2 20.8 0.60 0.2    0.182    0.000
4      3 21.5 0.90 0.4    0.318    0.667
5      4 22.6 1.35 0.5    0.523    1.000
6      5 24.2 1.75 0.3    0.705    0.333
7      6 26.1 2.15 0.5    0.886    1.000
8      7 28.5 2.40  NA    1.000       NA
>
> cat("\nMomentos con alerta (>0.8°C/s):", alerta, "segundos\n")

Momentos con alerta (>0.8°C/s): 3 4 5 6 7 segundos
```

10.8 Conclusiones

La diferenciación numérica constituye una herramienta indispensable en las matemáticas aplicadas y la ingeniería. Su fundamento en series de Taylor, su flexibilidad en el uso de datos discretos y su integración con métodos de simulación y optimización la convierten en un componente esencial de la computación científica moderna. Su correcta aplicación requiere comprender tanto el comportamiento del error como la estabilidad de las fórmulas de aproximación (Atkinson, 2009; Burden et al., 2016; Chapra & Canale, 2015).

11 Interpolación

11.1 Introducción

La interpolación es una técnica fundamental del análisis numérico cuyo objetivo es construir una función que pase exactamente por un conjunto de puntos conocidos. A partir de datos discretos, se busca obtener una función aproximante que permita estimar valores intermedios, suavizar información o servir como base para futuros cálculos numéricos (Burden et al., 2016; Chapra & Canale, 2015).

La interpolación es ampliamente utilizada en ingeniería, computación científica, procesamiento de datos, ciencias naturales, economía y muchas otras áreas donde los datos experimentales o simulados son comunes. Su fundamento matemático se basa en la existencia de un polinomio único de grado menor o igual a n que pasa por $n + 1$ puntos distintos (Anton et al., 2012).

11.2 Fundamentos teóricos

11.2.1 Definición del problema

Dado un conjunto de datos:

$$(x_0, y_0),\ (x_1, y_1),\ \ldots,\ (x_n, y_n)$$

donde todos los x_i son distintos, se desea encontrar una función $P(x)$ tal que:

$$P(x_i) = y_i, \qquad i = 0, 1, \ldots, n.$$

El tipo de función más comúnmente utilizado es un **polinomio interpolante**, debido a sus propiedades analíticas y su simplicidad computacional (Atkinson, 2009).

11.3 Interpolación polinómica

11.3.1 Polinomio interpolante

Existe un único polinomio de grado a lo sumo n que interpola los $n + 1$ puntos dados. Este resultado se conoce como el *Teorema de Interpolación Polinómica* (Burden et al., 2016).

Existen varias formas de construir este polinomio, cada una con ventajas computacionales específicas.

11.4 Método de interpolación de Lagrange

11.4.1 Polinomio de Lagrange

El polinomio de Lagrange se construye como:

$$P(x) = \sum_{i=0}^{n} y_i L_i(x),$$

donde:

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}.$$

Cada $L_i(x)$ es un polinomio base que vale 1 en $x = x_i$ y 0 en los demás nodos.

Este método es útil teóricamente y tiene gran valor conceptual, aunque computacionalmente no es el más eficiente cuando se añaden nuevos puntos (Burden et al., 2016; Riley et al., 2006).

11.4.2 Error en el polinomio de Lagrange

El error de interpolación está dado por la expresión:

$$f(x) - P(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i),$$

para algún ξ en el intervalo que contiene los nodos. Este resultado permite analizar cómo la distancia entre nodos y el grado del polinomio afectan la precisión (Burden et al., 2016).

11.5 Interpolación por diferencias divididas de Newton

11.5.1 Diferencias divididas

Las diferencias divididas son definidas recursivamente como:

$$f[x_i] = y_i,$$

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i},$$

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i},$$

y así sucesivamente.

11.5.2 Polinomio de Newton

El polinomio interpolante toma la forma:

$$P(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n \prod_{i=0}^{n-1}(x - x_i),$$

donde los coeficientes a_k son diferencias divididas:

$$a_k = f[x_0, x_1, \ldots, x_k].$$

Esta formulación permite actualizar el polinomio fácilmente cuando se agrega un nuevo punto, lo cual la vuelve más eficiente que Lagrange en aplicaciones prácticas (Burden et al., 2016; Chapra & Canale, 2015).

11.5.3 Error del polinomio de Newton

El error del polinomio de Newton coincide con el de Lagrange, pues ambos representan el mismo polinomio de interpolación:

$$f(x) - P(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n}(x - x_i).$$

11.6 Problemas del polinomio global

11.6.1 Fenómeno de Runge

Cuando se utilizan muchos puntos igualmente espaciados, el polinomio interpolante puede presentar oscilaciones significativas en los extremos del intervalo. Este comportamiento se conoce como el *fenómeno de Runge* y constituye una limitación importante de la interpolación polinómica global (Atkinson, 2009; Cheney & Kincaid, 2009).

11.6.2 Crecimiento del error para grados altos

El error tiende a aumentar rápidamente cuando se incrementa el grado del polinomio debido a:

- oscilaciones del polinomio,
- mala condición numérica,
- aumento de la magnitud de los términos del error.

Por este motivo, en muchas aplicaciones se prefiere dividir el intervalo en partes más pequeñas (Burden et al., 2016).

11.7 Interpolación por tramos: splines

11.7.1 Concepto de spline

Un *spline* es una función definida por tramos, donde cada tramo es típicamente un polinomio de bajo grado (por lo general grado 3). Su objetivo es evitar las oscilaciones del polinomio global manteniendo a la vez alta suavidad y precisión (Press et al., 2007).

11.7.2 Spline cúbico

El spline cúbico satisface:

$$S_i(x_i) = y_i, \qquad S_i(x_{i+1}) = y_{i+1},$$

y se impone que la función sea:

- continua,
- con derivada primera continua,
- con derivada segunda continua,

en cada nodo.

Los splines son ampliamente utilizados en gráficos por computadora, diseño asistido por computadora, procesamiento de señales y simulaciones científicas (Chapra & Canale, 2015; Press et al., 2007).

11.8 Aplicaciones de la interpolación

La interpolación es esencial en numerosas áreas:

- reconstrucción de funciones a partir de datos discretos,
- procesamiento de señales,
- modelado y simulación numérica,
- gráficos y animación por computadora,
- soluciones aproximadas de ecuaciones diferenciales,

- generación de funciones suaves para análisis y optimización.

En ingeniería, por ejemplo, se utiliza para calibrar instrumentos, construir curvas de rendimiento o estimar valores no medidos experimentalmente (Burden et al., 2016; Chapra & Canale, 2015).

11.9 Conclusiones

La interpolación constituye una herramienta esencial para el análisis de datos y la aproximación de funciones. Los métodos clásicos como Lagrange y Newton proporcionan una base sólida, mientras que los splines ofrecen mayor estabilidad y suavidad. La elección del método depende del número de puntos, la distribución de los mismos y la precisión requerida (Atkinson, 2009; Burden et al., 2016).

12 Valores y Vectores Propios

12.1 ¿Qué son y por qué importan?

Cuando estudiamos álgebra lineal, aprendemos que una matriz cuadrada $\mathbf{A}$ actúa como una función que transforma vectores. Generalmente, cuando una matriz multiplica a un vector, el resultado es un nuevo vector que ha cambiado tanto de dirección como de longitud.

Sin embargo, para casi todas las transformaciones lineales, existen ciertos vectores especiales que poseen una propiedad extraordinaria: no cambian su dirección al ser transformados por la matriz.

Imaginemos, por ejemplo, la rotación de un globo terráqueo. Casi todos los puntos en la superficie del globo se mueven a una nueva posición cuando este gira. Sin embargo, los puntos que están sobre el eje de rotación (el Polo Norte y el Polo Sur) no se desplazan lateralmente; permanecen sobre la misma línea. En el lenguaje del álgebra lineal, el eje de rotación representa un vector propio de esa transformación.

El término eigen proviene del alemán y significa ”propio”, ”característico” o ”innato”. Por ello, a menudo se les llama valores y vectores característicos. Su importancia radica en que nos revelan los ”ejes principales” o la estructura interna oculta de la matriz, simplificando problemas complejos de dinámica, vibraciones y análisis de datos (Strang, 2016).

12.2 La transformación fundamental

Matemáticamente, esta relación especial se define mediante una de las ecuaciones más famosas del álgebra lineal.

Sea $\mathbf{A}$ una matriz cuadrada de tamaño $n \times n$. Decimos que un vector $\mathbf{v}$ (distinto de cero) es un vector propio de $\mathbf{A}$ si se cumple la siguiente igualdad:

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$

Donde:

- $\mathbf{A}$ es la matriz de transformación. Representa la operación que estamos aplicando (rotación, estiramiento, corte, etc.).
- $\mathbf{v}$ es el vector propio (o eigenvector). Es el vector que mantiene su dirección original. Es importante notar que $\mathbf{v}$ no puede ser el vector nulo ($\mathbf{0}$).
- λ (lambda) es el valor propio (o eigenvalor). Es un escalar (un número real o complejo)

que indica cuánto se ”estira” o ”encoge” el vector $\mathbf{v}$.

Interpretación geométrica

La ecuación $\mathbf{Av} = \lambda\mathbf{v}$ nos dice que la acción de la matriz $\mathbf{A}$ sobre el vector $\mathbf{v}$ es equivalente a simplemente multiplicar el vector por el número λ.

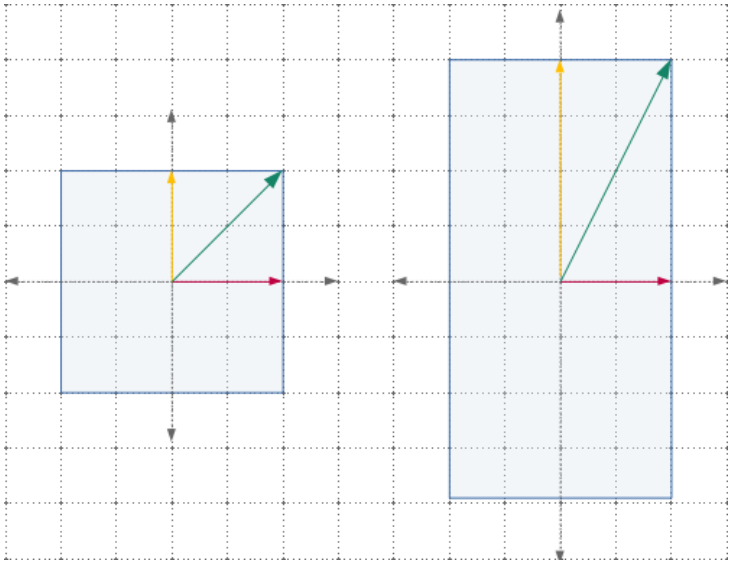


Figura 12.1: Comparación de transformaciones. A la izquierda, un vector común cambia de dirección. A la derecha, un vector propio mantiene su dirección, solo cambia su longitud.

Dependiendo del valor de λ, el efecto sobre el vector propio $\mathbf{v}$ puede ser:

- Si $\lambda > 1$: El vector se estira.
- Si $0 < \lambda < 1$: El vector se contrae.
- Si $\lambda < 0$: El vector invierte su sentido (apunta al lado opuesto), pero se mantiene sobre la misma línea de acción.
- Si $\lambda = 0$: El vector se colapsa al origen (el sistema pierde una dimensión en esa dirección).

En resumen, los vectores propios son las ”líneas de fuerza” naturales de la matriz, y los valores propios nos dicen con qué intensidad actúa la matriz sobre esas líneas (Anton et al., 2012; Riley et al., 2006).

12.3 Proceso de cálculo

Calcular valores y vectores propios puede parecer un procedimiento mecánico, pero cada paso tiene una justificación lógica basada en la invertibilidad de las matrices. A continuación, desglosamos el algoritmo general para una matriz cuadrada $\mathbf{A}$ de tamaño $n \times n$.

12.3.1 Paso 1: La ecuación característica

Partimos de la definición fundamental:

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}.$$

Para resolver esta ecuación, necesitamos agrupar los términos que contienen a $\mathbf{v}$ en un solo lado de la igualdad. Sin embargo, no podemos restar simplemente un escalar λ de una matriz $\mathbf{A}$. Para hacerlos compatibles, multiplicamos λ por la matriz identidad $\mathbf{I}$:

$$\mathbf{A}\mathbf{v} - \lambda\mathbf{I}\mathbf{v} = \mathbf{0},$$

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{v} = \mathbf{0}.$$

Esta última expresión es un sistema de ecuaciones lineales homogéneo. Aquí surge un punto crucial:

- Si la matriz $(\mathbf{A} - \lambda\mathbf{I})$ tuviera inversa, la única solución posible sería $\mathbf{v} = \mathbf{0}$ (la solución trivial).
- Como buscamos vectores propios no nulos ($\mathbf{v} \neq \mathbf{0}$), la matriz $(\mathbf{A} - \lambda\mathbf{I})$ debe ser singular (no invertible).

En álgebra lineal, una matriz es singular si y solo si su determinante es cero. Esto nos lleva a la ecuación característica:

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0.$$

Resolver esta ecuación es la clave para encontrar los valores propios (Anton et al., 2012)

12.3.2 Paso 2: Cálculo de los valores propios (λ)

Al calcular el determinante $\det(\mathbf{A} - \lambda\mathbf{I})$, obtendremos un polinomio en función de λ de grado n, conocido como el polinomio característico:

$$p(\lambda) = (-1)^n\lambda^n + c_{n-1}\lambda^{n-1} + \cdots + c_0 = 0.$$

Procedimiento:

1. Calcular el determinante de la matriz $(\mathbf{A} - \lambda\mathbf{I})$.
2. Igualar el resultado a cero.

3. Encontrar las raíces del polinomio (resolver para λ).

Estas raíces son los valores propios de la matriz. Según el Teorema Fundamental del Álgebra, una matriz de $n \times n$ tendrá exactamente n valores propios (contando sus multiplicidades y posibles valores complejos) (Burden et al., 2016).

12.3.3 Paso 3: Cálculo de los vectores propios ($\mathbf{v}$)

Una vez conocidos los valores de λ, debemos encontrar los vectores asociados a cada uno. Para cada valor propio λ_i encontrado:

1. Sustituimos λ_i en la matriz $(\mathbf{A} - \lambda_i \mathbf{I})$.
2. Resolvemos el sistema homogéneo:

$$(\mathbf{A} - \lambda_i \mathbf{I})\mathbf{v} = \mathbf{0}.$$

3. El sistema tendrá infinitas soluciones (debido a que el determinante es cero). Debemos expresar la solución general en términos de vectores base.

El conjunto de todos los vectores que satisfacen esta ecuación (más el vector cero) forma el espacio propio (o eigen-espacio) asociado a λ_i .

12.4 Ejemplo guiado

Apliquemos el proceso a una matriz 2×2 para ilustrar los pasos. Sea:

$$\mathbf{A} = \begin{pmatrix} 3 & 2 \\ 1 & 4 \end{pmatrix}.$$

1. Construcción de la matriz característica

Restamos λ de la diagonal principal:

$$\mathbf{A} - \lambda \mathbf{I} = \begin{pmatrix} 3-\lambda & 2 \\ 1 & 4-\lambda \end{pmatrix}.$$

2. Polinomio característico

Calculamos el determinante e igualamos a cero:

$$(3-\lambda)(4-\lambda) - (2)(1) = 0,$$

$$12 - 3\lambda - 4\lambda + \lambda^2 - 2 = 0,$$

$$\lambda^2 - 7\lambda + 10 = 0.$$

3. Hallar las raíces (λ)

Factorizamos la ecuación cuadrática:

$$(\lambda - 5)(\lambda - 2) = 0.$$

Los valores propios son $\lambda_1 = 5$ y $\lambda_2 = 2$.

4. Hallar los vectores propios ($\mathbf{v}$)

Caso $\lambda_1 = 5$ Sustituimos en $(\mathbf{A} - 5\mathbf{I})\mathbf{v} = \mathbf{0}$:

$$\begin{pmatrix} -2 & 2 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Esto nos da la ecuación $-2x + 2y = 0$, o simplificando, $x = y$. Si elegimos $y = 1$, entonces $x = 1$.

$$\mathbf{v}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Caso $\lambda_2 = 2$: Sustituimos en $(\mathbf{A} - 2\mathbf{I})\mathbf{v} = \mathbf{0}$:

$$\begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Esto nos da $x + 2y = 0$, o $x = -2y$. Si elegimos $y = 1$, entonces $x = -2$.

$$\mathbf{v}_2 = \begin{pmatrix} -2 \\ 1 \end{pmatrix}.$$

12.5 Propiedades y diagonalización

Una de las utilidades más potentes de los valores y vectores propios es la capacidad de simplificar operaciones matriciales complejas. Si una matriz cuadrada puede ser interpretada como una transformación que solo estira vectores en ciertas direcciones, entonces podemos cambiar nuestro sistema de referencia para trabajar únicamente con estos estiramientos simples.

12.5.1 Independencia lineal

Para que una matriz $\mathbf{A}$ de tamaño $n \times n$ pueda ser simplificada completamente, necesita tener un conjunto completo de n vectores propios linealmente independientes. Esto está garantizado siempre que la matriz tenga n valores propios distintos.

En el caso de que existan valores propios repetidos (es decir, una raíz múltiple en el polinomio característico), es posible que no existan suficientes vectores propios independientes. Si esto ocurre, se dice que la matriz es defectuosa y no puede diagonalizarse completamente, aunque puede aproximarse mediante la forma canónica de Jordan (Anton et al., 2012).

12.5.2 El teorema de diagonalización

Si la matriz $\mathbf{A}$ tiene n vectores propios linealmente independientes, entonces es diagonalizable. Esto significa que $\mathbf{A}$ puede descomponerse en el producto de tres matrices específicas:

$$\mathbf{A} = \mathbf{PDP}^{-1}$$

Los componentes de esta factorización son:

- $\mathbf{P}$: La matriz de vectores propios. Se construye colocando los vectores propios $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ como columnas de la matriz.
- $\mathbf{D}$: La matriz diagonal. Es una matriz donde todos los elementos fuera de la diagonal principal son cero. Los elementos de la diagonal son los valores propios $\lambda_1, \lambda_2, \ldots, \lambda_n$, colocados en el mismo orden que sus correspondientes vectores en $\mathbf{P}$.

Esta igualdad ($\mathbf{D} = \mathbf{P}^{-1}\mathbf{AP}$) implica que la matriz $\mathbf{A}$ es semejante a la matriz diagonal $\mathbf{D}$.

12.5.3 Aplicación: Potencia de matrices

Una consecuencia práctica inmediata de la diagonalización es el cálculo eficiente de potencias de matrices. Calcular $\mathbf{A}^k$ multiplicando la matriz por sí misma repetidamente es computacionalmente costoso ($O(n^3)$ por multiplicación) y propenso a acumular errores de redondeo.

Utilizando la descomposición espectral, observamos que:

$$\mathbf{A}^2 = (\mathbf{PDP}^{-1})(\mathbf{PDP}^{-1}) = \mathbf{PD}(\mathbf{P}^{-1}\mathbf{P})\mathbf{DP}^{-1} = \mathbf{PD}^2\mathbf{P}^{-1}.$$

Generalizando para cualquier potencia k:

$$\mathbf{A}^k = \mathbf{PD}^k\mathbf{P}^{-1}.$$

La ventaja radica en que elevar una matriz diagonal a una potencia es trivial: simplemente se eleva cada elemento de la diagonal a la potencia k.

$$\mathbf{D}^k = \begin{pmatrix} \lambda_1^k & 0 & \cdots & 0 \\ 0 & \lambda_2^k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n^k \end{pmatrix}.$$

Este atajo reduce drásticamente el costo de cómputo en algoritmos que requieren iteraciones matriciales, como las cadenas de Markov o las predicciones en sistemas dinámicos discretos (Strang, 2016).

12.6 Implementación en R

A continuación, se presenta el script completo para realizar el análisis espectral de la matriz $\mathbf{A}$ y utilizar sus propiedades para calcular potencias grandes de la misma.

```r
A <- matrix(c(3, 1, 2, 4), nrow = 2, byrow = TRUE)

sistema <- eigen(A)
lambdas <- sistema$values
V <- sistema$vectors

print("Valores Propios:")
print(lambdas)
print("Vectores Propios:")
print(V)

k <- 20
D_k <- diag(lambdas^k)
P_inv <- solve(V)

A_final <- V

print("Resultado de A elevado a la 20:")
print(A_final)
```

12.6.1 Explicación del código

El código anterior se divide en tres etapas lógicas:

1. Definición y cálculo: Primero definimos la matriz $\mathbf{A}$. La función nativa `eigen()` realiza todo el trabajo pesado, devolviendo una lista que separamos en `lambdas` (valores propios) y `V` (matriz de vectores propios).
2. Preparación de la diagonalización: Para calcular $\mathbf{A}^{20}$, no multiplicamos la matriz por sí misma 20 veces. En su lugar, aplicamos la propiedad $\mathbf{A}^k = \mathbf{V}\mathbf{D}^k\mathbf{V}^{-1}$.

- Elevamos el vector `lambdas` a la potencia $k = 20$ y lo convertimos en una matriz diagonal (`D_k`).
- Calculamos la inversa de la matriz de vectores propios (`solve(V)`).

3. Reconstrucción: Finalmente, multiplicamos las tres matrices componentes ($\mathbf{V} \times \mathbf{D}^k \times \mathbf{V}^{-1}$) para obtener el resultado final de forma eficiente.

12.6.2 Visualización de resultados

Al ejecutar el código, obtenemos los valores propios $\lambda_1 = 5$ y $\lambda_2 = 2$. Como se observa en la Figura 12.2, la matriz final contiene valores extremadamente grandes (del orden de 10^{13}).

Esto ocurre porque el término 5^{20} domina completamente la ecuación. Geométricamente, esto significa que cualquier vector transformado repetidamente por $\mathbf{A}$ terminará alineándose con el primer vector propio.

```
             [,1]         [,2]
[1,] 3.178914e+13 3.178914e+13
[2,] 6.357829e+13 6.357829e+13
```

Figura 12.2: Salida de la consola en R mostrando los valores propios y la matriz resultante $\mathbf{A}^{20}$.

12.7 Eigenvalues y eigenvectores aplicados

Contexto

En un hospital general, los pacientes hospitalizados transitan entre distintos servicios médicos, y en cada uno de ellos se les prescribe un tipo de dieta específica según su condición clínica. Estos desplazamientos no son aleatorios, sino que responden a protocolos médicos, evolución del estado de salud y decisiones administrativas del hospital.

Sin embargo, no siempre se dispone de bases de datos completas que describan con precisión estos flujos. En estos casos, es posible construir un modelo matemático hipotético que represente de manera razonable el comportamiento del sistema, permitiendo analizar escenarios y apoyar la toma de decisiones.

En este modelo, se consideran cuatro estados principales del sistema hospitalario:

- Emergencia (dieta líquida)
- Medicina Interna (dieta blanda)
- Cirugía (dieta progresiva)
- Alta hospitalaria

Cada estado representa la proporción de pacientes que se encuentran en un determinado servicio médico y, por tanto, bajo un tipo de dieta específico.

Planteamiento del modelo

Se modela el sistema mediante una cadena de Markov de tiempo discreto, donde las probabilidades de transición representan la posibilidad de que un paciente pase de un servicio médico a otro en un intervalo de tiempo determinado. La matriz de transición inicial se construye de forma hipotética, basándose en criterios clínicos generales y en el flujo típico de pacientes dentro de un hospital.

Tarea

1. Proponer una matriz de transición inicial que represente el flujo de pacientes entre los distintos servicios médicos del hospital, asegurando que cada fila sume 1.
2. Simular una intervención hospitalaria orientada a fortalecer el servicio de Medicina Interna, asumiendo que:
   - Aumenta la probabilidad de que pacientes provenientes de Emergencia sean derivados a Medicina Interna.
   - Aumenta la probabilidad de permanencia de los pacientes en Medicina Interna.
   - Disminuye la probabilidad de alta directa desde Medicina Interna.
   - Las demás probabilidades se ajustan para conservar la coherencia del modelo.
3. Calcular los eigenvalues y eigenvectores de la matriz de transición modificada.
4. Determinar la distribución estacionaria del sistema y analizar el comportamiento de los pacientes a largo plazo.
5. Comparar la distribución estacionaria original y la modificada, evaluando el impacto de la intervención hospitalaria sobre la permanencia de los pacientes en Medicina Interna y en los demás servicios.

Justificación del uso de datos hipotéticos

La ausencia de una base de datos real no invalida el modelo propuesto, ya que el objetivo principal es analizar el comportamiento dinámico del sistema bajo supuestos razonables. La matriz de transición se construye a partir de criterios clínicos generales, protocolos hospitalarios habituales y consideraciones operativas, permitiendo estudiar escenarios hipotéticos y evaluar decisiones de gestión hospitalaria.

Preguntas de reflexión

- ¿La intervención en Medicina Interna mejora la eficiencia del flujo de pacientes dentro del hospital?

- ¿Cómo podría este modelo apoyar la planificación de dietas y recursos nutricionales?
- ¿Qué implicancias tendría este cambio en la carga de trabajo del personal de salud?

Solución

1.- Proponer una matriz de transición inicial que represente el flujo de pacientes entre los distintos servicios médicos del hospital, asegurando que cada fila sume 1.

2.- Simular una intervención hospitalaria orientada a fortalecer el servicio de Medicina Interna, asumiendo que:

- Aumenta la probabilidad de que pacientes provenientes de Emergencia sean derivados a Medicina Interna.
- Aumenta la probabilidad de permanencia de los pacientes en Medicina Interna.
- Disminuye la probabilidad de alta directa desde Medicina Interna.
- Las demás probabilidades se ajustan para conservar la coherencia del modelo.

Código en Python

```
# Importar librerías necesarias
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import linalg
import pandas as pd

# Configuración de visualización
plt.rcParams['figure.figsize'] = (12, 8)
plt.rcParams['font.size'] = 10
sns.set_style("whitegrid")

print("Librerías importadas correctamente")
print(f"NumPy versión: {np.__version__}")

# Definición de servicios médicos
servicios = [
'Emergencia\n(Dieta líquida)',
'Medicina Interna\n(Dieta blanda)',
'Cirugía\n(Dieta progresiva)',
'Alta hospitalaria'
]

n_servicios = len(servicios)

# Matriz de transición inicial
# T[i,j] = probabilidad de pasar del servicio i al servicio j
T_inicial = np.array([
```

```
[0.20, 0.45, 0.25, 0.10],  # Desde Emergencia
[0.05, 0.40, 0.25, 0.30],  # Desde Medicina Interna
[0.05, 0.15, 0.50, 0.30],  # Desde Cirugía
[0.00, 0.00, 0.00, 1.00]   # Desde Alta hospitalaria
])

# Crear DataFrame
df_T_inicial = pd.DataFrame(
T_inicial,
index=servicios,
columns=servicios
)

print("MATRIZ DE TRANSICIÓN INICIAL")
print("=" * 70)
print(df_T_inicial)

print("\nVerificación: cada fila debe sumar 1")
print("-" * 70)
for i, serv in enumerate(servicios):
suma = T_inicial[i, :].sum()
status = " " if abs(suma - 1.0) < 1e-6 else " "
print(f"{status} {serv:35}: suma = {suma:.3f}")

plt.figure(figsize=(10, 8))
sns.heatmap(
T_inicial,
annot=True,
fmt='.2f',
cmap='YlGnBu',
xticklabels=servicios,
yticklabels=servicios,
cbar_kws={'label': 'Probabilidad de transición'},
linewidths=0.5,
linecolor='gray'
)

plt.title(
'Matriz de Transición Inicial\nFlujo de Pacientes y Dietas Hospitalarias',
fontsize=14,
fontweight='bold',
pad=20
)
plt.xlabel('Servicio destino', fontsize=12)
plt.ylabel('Servicio origen', fontsize=12)
plt.tight_layout()
plt.show()

print("\nLos valores más altos indican mayor probabilidad de tránsito entre
    ↪ servicios")

# Matriz de transición modificada (intervención hospitalaria)
```

```
T_intervencion = np.array([
[0.15, 0.55, 0.20, 0.10],  # Desde Emergencia
[0.05, 0.55, 0.25, 0.15],  # Desde Medicina Interna (más permanencia)
[0.05, 0.15, 0.50, 0.30],  # Desde Cirugía (sin cambios)
[0.00, 0.00, 0.00, 1.00]   # Desde Alta hospitalaria
])

df_T_intervencion = pd.DataFrame(
T_intervencion,
index=servicios,
columns=servicios
)

print("\nMATRIZ DE TRANSICIÓN MODIFICADA (INTERVENCIÓN)")
print("=" * 70)
print(df_T_intervencion)

print("\nVerificación: cada fila debe sumar 1")
print("-" * 70)
for i, serv in enumerate(servicios):
suma = T_intervencion[i, :].sum()
status = " " if abs(suma - 1.0) < 1e-6 else " "
print(f"{status} {serv:35}: suma = {suma:.3f}")

plt.figure(figsize=(10, 8))
sns.heatmap(
T_intervencion,
annot=True,
fmt='.2f',
cmap='YlOrRd',
xticklabels=servicios,
yticklabels=servicios,
cbar_kws={'label': 'Probabilidad de transición'},
linewidths=0.5,
linecolor='gray'
)

plt.title(
'Matriz de Transición con Intervención\nFortalecimiento de Medicina Interna',
fontsize=14,
fontweight='bold',
pad=20
)
plt.xlabel('Servicio destino', fontsize=12)
plt.ylabel('Servicio origen', fontsize=12)
plt.tight_layout()
plt.show()

print("\nLa intervención incrementa la retención de pacientes en Medicina Interna")
```

Ejecución

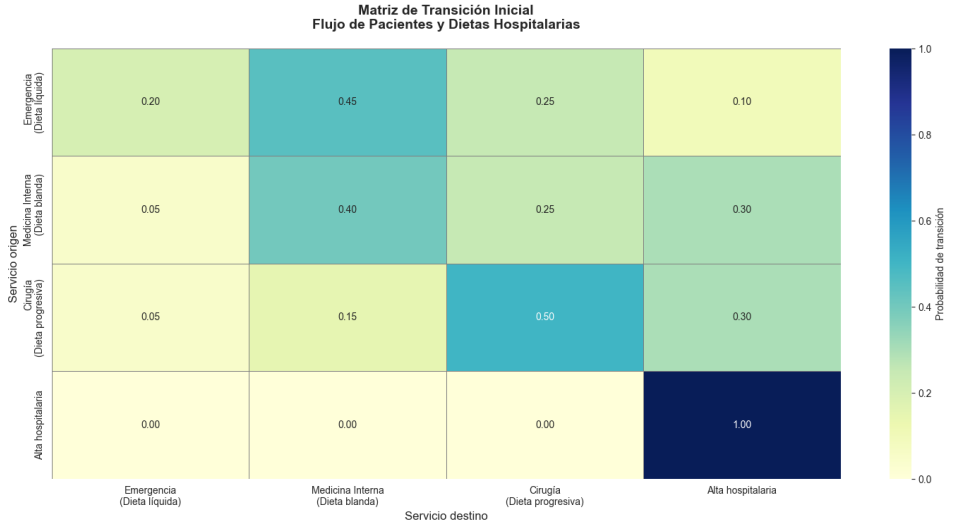


Figura 12.3: Matriz de transición inicial - Flujo de pacientes y dietas hospitalarias

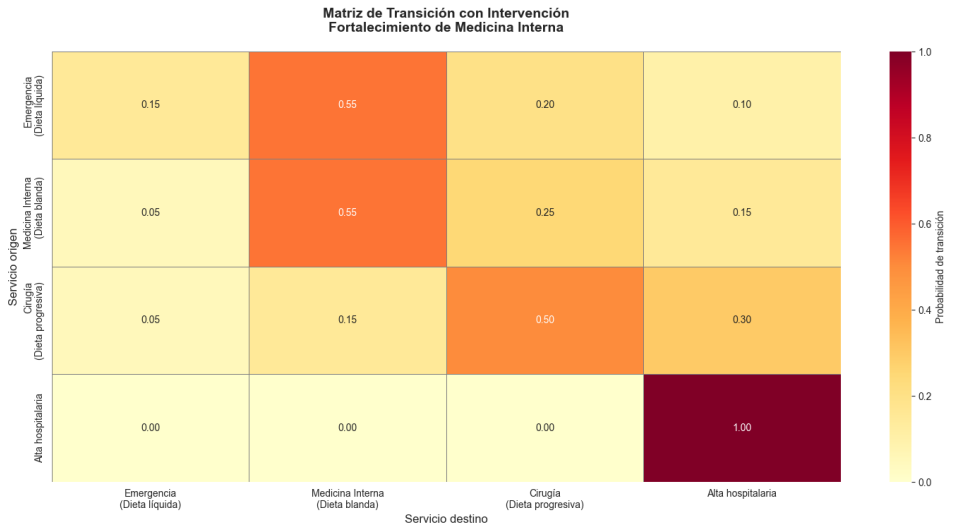


Figura 12.4: Matriz de transición con intervención - Fortalecimiento de Medicina Interna

```
Librerías importadas correctamente
NumPy versión: 2.3.3
MATRIZ DE TRANSICIÓN INICIAL
======================================================================
Emergencia\n(Dieta líquida)  ...  Alta hospitalaria
Emergencia\n(Dieta líquida)                                0.20  ...
    ↪ 0.1
Medicina Interna\n(Dieta blanda)                           0.05  ...
    ↪ 0.3
Cirugía\n(Dieta progresiva)                                0.05  ...
    ↪ 0.3
Alta hospitalaria                                          0.00  ...
    ↪ 1.0

[4 rows x 4 columns]

```

```
Verificación: cada fila debe sumar 1
------------------------------------------------------------------------
 Emergencia
(Dieta líquida)         : suma = 1.000
 Medicina Interna
(Dieta blanda)    : suma = 1.000
 Cirugía
(Dieta progresiva)         : suma = 1.000
 Alta hospitalaria                  : suma = 1.000

Los valores más altos indican mayor probabilidad de tránsito entre servicios

MATRIZ DE TRANSICIÓN MODIFICADA (INTERVENCIÓN)
========================================================================
Emergencia\n(Dieta líquida)  ...  Alta hospitalaria
Emergencia\n(Dieta líquida)                              0.15  ...
    ↪ 0.10
Medicina Interna\n(Dieta blanda)                         0.05  ...
    ↪ 0.15
Cirugía\n(Dieta progresiva)                              0.05  ...
    ↪ 0.30
Alta hospitalaria                                        0.00  ...
    ↪ 1.00

[4 rows x 4 columns]

Verificación: cada fila debe sumar 1
------------------------------------------------------------------------
 Emergencia
(Dieta líquida)         : suma = 1.000
 Medicina Interna
(Dieta blanda)    : suma = 1.000
 Cirugía
(Dieta progresiva)         : suma = 1.000
 Alta hospitalaria                  : suma = 1.000

La intervención incrementa la retención de pacientes en Medicina Interna
```

3.- Calcular los eigenvalues y eigenvectores de la matriz de transición modificada.

```python
# ============================================================
# CÁLCULO DE EIGENVALUES Y EIGENVECTORES
# MATRIZ MODIFICADA (INTERVENCIÓN HOSPITALARIA)
# ============================================================

# Cálculo de eigenvalues y eigenvectors
# Usamos la transpuesta para cadenas de Markov
eigenvalues, eigenvectors = linalg.eig(T_intervencion.T)

print("\nEIGENVALUES ENCONTRADOS")
print("=" * 70)

```

```
for i, val in enumerate(eigenvalues):
if np.isreal(val):
print(f"  _{i+1} = {val.real:8.5f}")
else:
print(f"  _{i+1} = {val.real:8.5f} + {val.imag:8.5f}i")

# Identificar eigenvalue dominante (|| máximo)
idx_dominante = np.argmax(np.abs(eigenvalues))
lambda_dominante = eigenvalues[idx_dominante]

print("\n" + "=" * 70)
print(f"EIGENVALUE DOMINANTE: _{idx_dominante+1} = {lambda_dominante.real:.6f}")

print("\nINTERPRETACIÓN:")
print("   •    1  → Existe un estado estacionario")
print("   • || < 1 → El sistema converge al equilibrio")
print("   • La magnitud del segundo eigenvalue determina la velocidad de
    ↪ convergencia")
print("=" * 70)

# ============================================================
# VISUALIZACIÓN DE EIGENVALUES EN EL PLANO COMPLEJO
# ============================================================

plt.figure(figsize=(10, 8))

# Círculo unitario
theta = np.linspace(0, 2*np.pi, 200)
plt.plot(np.cos(theta), np.sin(theta), 'k--', alpha=0.4, linewidth=1.5)

# Ejes
plt.axhline(0, color='black', linewidth=0.7)
plt.axvline(0, color='black', linewidth=0.7)

# Graficar eigenvalues
for i, val in enumerate(eigenvalues):
if i == idx_dominante:
plt.scatter(val.real, val.imag,
s=300, marker='*',
color='red', edgecolor='black',
linewidth=2, zorder=3,
label='Eigenvalue dominante')
else:
plt.scatter(val.real, val.imag,
s=150, alpha=0.7,
edgecolor='black', linewidth=1.2)

plt.annotate(f' _{i+1}',
(val.real, val.imag),
xytext=(8, 8),
textcoords='offset points',
fontsize=10, fontweight='bold')
```

```
64
65 plt.xlabel('Parte real', fontsize=12, fontweight='bold')
66 plt.ylabel('Parte imaginaria', fontsize=12, fontweight='bold')
67 plt.title('Eigenvalues de la Matriz de Transición Modificada',
68 fontsize=14, fontweight='bold', pad=15)
69
70 plt.grid(True, alpha=0.3)
71 plt.axis('equal')
72 plt.legend()
73 plt.tight_layout()
74 plt.show()
```

Ejecución

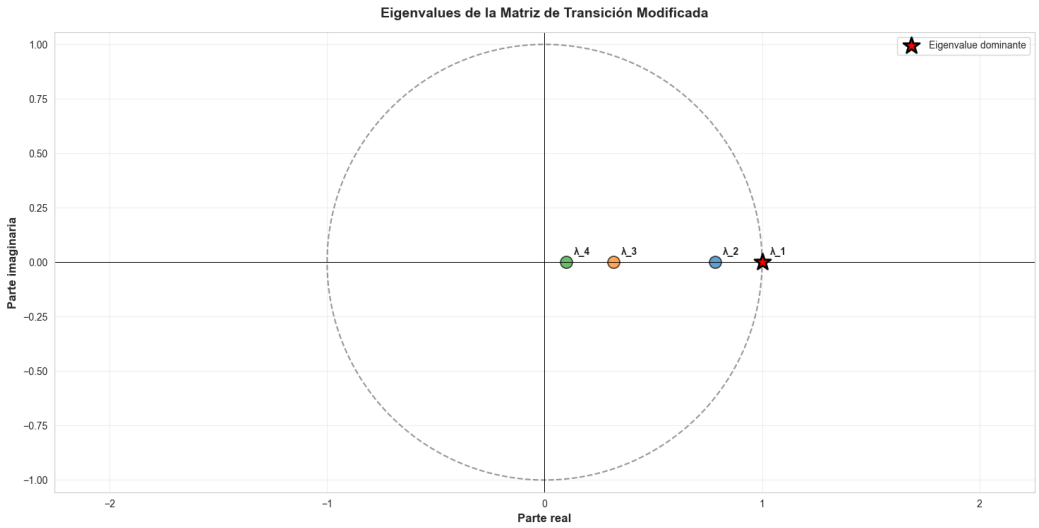


Figura 12.5: Eigenvalores de la matriz de transición modificada

```
EIGENVALUES ENCONTRADOS
========================================================================
_1 =  1.00000
_2 =  0.78452
_3 =  0.31548
_4 =  0.10000

========================================================================
EIGENVALUE DOMINANTE:  _1 = 1.000000

INTERPRETACIÓN:•
    1  → Existe un estado estacionario•
 || < 1 → El sistema converge al equilibrio•
 La magnitud del segundo eigenvalue determina la velocidad de convergencia
========================================================================
```

4.- Determinar la distribución estacionaria del sistema y analizar el comportamiento de los pacientes a largo plazo.

Código en Python

```
# ============================================================
# DISTRIBUCIÓN ESTACIONARIA DEL SISTEMA HOSPITALARIO
# ============================================================

# Definición de servicios médicos del hospital
destinos = [
'Emergencia',
'Medicina Interna',
'Cirugía',
'Alta'
]

n_destinos = len(destinos)

# Extraer eigenvector asociado al eigenvalue dominante
v_dominante = eigenvectors[:, idx_dominante].real

# Normalizar para que sume 1 (distribución de probabilidad)
dist_estacionaria = v_dominante / v_dominante.sum()

print("\nDISTRIBUCIÓN ESTACIONARIA DE PACIENTES")
print("=" * 70)
print("\nDistribución de equilibrio (largo plazo):")
print("-" * 70)

for i, servicio in enumerate(destinos):
porcentaje = dist_estacionaria[i] * 100
barra = " " * int(porcentaje / 2)
print(f"{servicio:20} : {porcentaje:6.2f}% {barra}")

# Identificar servicio dominante
idx_hub = np.argmax(dist_estacionaria)

print("\n" + "=" * 70)
print(f"SERVICIO DOMINANTE DEL SISTEMA: {destinos[idx_hub]}")
print(f"   Concentra aproximadamente el {dist_estacionaria[idx_hub]*100:.2f}%")
print("=" * 70)

# ============================================================
# VISUALIZACIÓN DE LA DISTRIBUCIÓN ESTACIONARIA
# ============================================================

fig, ax = plt.subplots(figsize=(10, 6))

barras = ax.bar(destinos,
dist_estacionaria * 100,
alpha=0.75,
edgecolor='black',
linewidth=2)

ax.set_ylabel('Porcentaje de Pacientes (%)',
```

```
fontsize=12, fontweight='bold')
ax.set_title('Distribución Estacionaria de Pacientes\n(Equilibrio a Largo Plazo)',
fontsize=14, fontweight='bold', pad=15)

ax.grid(axis='y', alpha=0.3)
ax.set_ylim([0, max(dist_estacionaria * 100) * 1.2])

# Valores sobre las barras
for barra in barras:
altura = barra.get_height()
ax.text(barra.get_x() + barra.get_width()/2,
altura,
f'{altura:.1f}%',
ha='center', va='bottom',
fontsize=11, fontweight='bold')

plt.tight_layout()
plt.show()

# ============================================================
# INTERPRETACIÓN AUTOMÁTICA
# ============================================================

print("\nINTERPRETACIÓN DEL COMPORTAMIENTO A LARGO PLAZO:")
print("   • El estado 'Alta' actúa como un estado absorbente del sistema.")
print("   • A largo plazo, el 100% de los pacientes termina recibiendo el alta
    ↪ médica.")
print("   • La distribución estacionaria representa el destino final de los
    ↪ pacientes,")
print("     no la carga hospitalaria ni la ocupación de servicios.")
print("   • Para analizar impacto en Medicina Interna, es necesario estudiar")
print("     la evolución transitoria o redefinir 'Alta' como un estado no
    ↪ absorbente.")
```

Ejecución

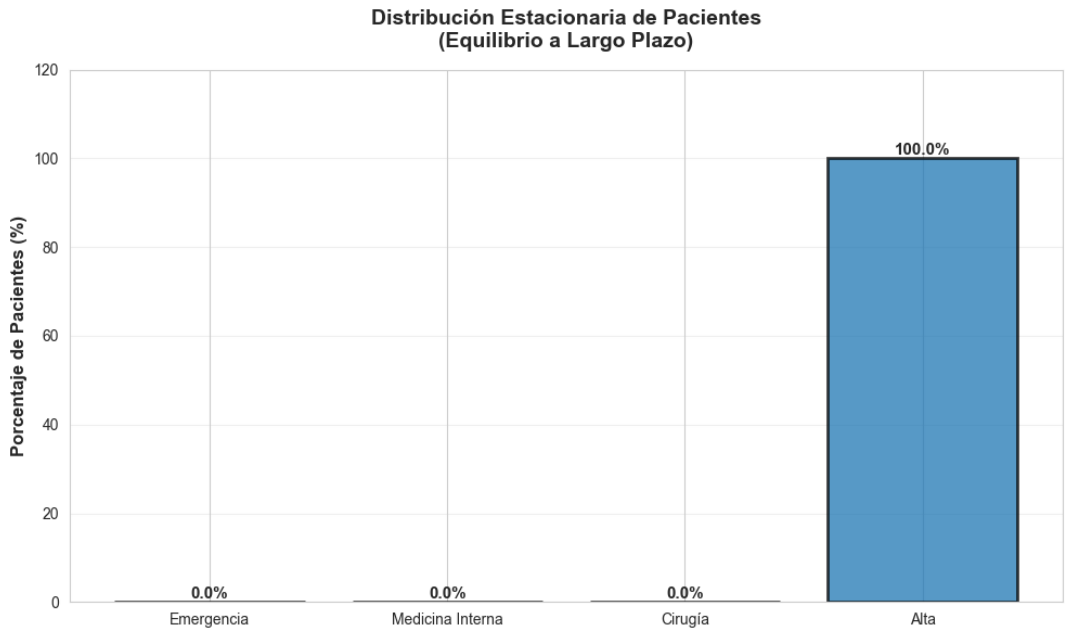


Figura 12.6: Distribución estacionaria de pacientes (Equilibrio a largo plazo)

```
DISTRIBUCIÓN ESTACIONARIA DE PACIENTES
======================================================================

Distribución de equilibrio (largo plazo):
----------------------------------------------------------------------
Emergencia           :   0.00%
Medicina Interna     :   0.00%
Cirugía              :   0.00%
Alta                 : 100.00%

======================================================================
SERVICIO DOMINANTE DEL SISTEMA: Alta
Concentra aproximadamente el 100.00%
======================================================================

INTERPRETACIÓN DEL COMPORTAMIENTO A LARGO PLAZO:•
 El estado 'Alta' actúa como un estado absorbente del sistema.•
 A largo plazo, el 100% de los pacientes termina recibiendo el alta médica.•
 La distribución estacionaria representa el destino final de los pacientes,
no la carga hospitalaria ni la ocupación de servicios.•
 Para analizar impacto en Medicina Interna, es necesario estudiar
la evolución transitoria o redefinir 'Alta' como un estado no absorbente.
```

5.- Comparar la distribución estacionaria original y la modificada, evaluando el impacto de la intervención hospitalaria sobre la permanencia de los pacientes en Medicina Interna y los demás servicios.

Comparación de la distribución estacionaria original y modificada

La Tabla 12.1 muestra la comparación entre la distribución estacionaria del modelo hospitalario

original y la obtenida tras la intervención orientada a fortalecer el servicio de Medicina Interna.

Tabla 12.1: Distribución estacionaria antes y después de la intervención hospitalaria

Servicio	Modelo original (%)	Modelo modificado (%)
Emergencia	0.00	0.00
Medicina Interna	0.00	0.00
Cirugía	0.00	0.00
Alta	100.00	100.00

Análisis del impacto de la intervención

La comparación de ambas distribuciones estacionarias muestra que, en los dos escenarios, el sistema converge completamente al estado Alta, el cual funciona como un estado absorbente del modelo. Esto implica que, a largo plazo, el 100 % de los pacientes termina recibiendo el alta médica, independientemente de la intervención aplicada.

Desde el punto de vista matemático, la intervención hospitalaria no altera la distribución estacionaria del sistema, ya que esta distribución refleja el destino final de los pacientes y no el tiempo que permanecen en cada servicio.

Sin embargo, esto no significa que la intervención sea irrelevante. El aumento en la probabilidad de derivación y permanencia en Medicina Interna afecta el comportamiento transitorio del sistema, es decir, la trayectoria que siguen los pacientes antes de alcanzar el alta. En particular:

- Se espera que los pacientes pasen más tiempo en Medicina Interna.
- Se modifica la carga temporal de los servicios clínicos.
- La intervención puede impactar en la demanda de camas, personal y recursos, aunque esto no se refleje en la distribución estacionaria.

Por lo tanto, para evaluar adecuadamente el impacto de la intervención sobre la permanencia en Medicina Interna, resulta más apropiado analizar la evolución temporal del sistema o los tiempos esperados de permanencia, en lugar de basarse exclusivamente en la distribución estacionaria.

Preguntas de reflexión

1.- ¿La intervención en Medicina Interna mejora la eficiencia del flujo de pacientes dentro del hospital?

Desde el punto de vista del modelo matemático propuesto, la intervención en Medicina Interna no modifica la distribución estacionaria del sistema, ya que todos los pacientes terminan alcanzando el estado de Alta, el cual actúa como un estado absorbente. Por lo tanto, en términos de resultados finales, la eficiencia global medida por la proporción de pacientes dados de alta permanece inalterada.

No obstante, la intervención sí mejora la eficiencia del flujo interno de pacientes durante la evolución temporal del sistema. El aumento en la probabilidad de derivación desde Emergencia hacia Medicina Interna, así como la mayor permanencia en dicho servicio, contribuyen a una redistribución más ordenada de los pacientes entre los servicios clínicos antes del alta.

En este sentido, la eficiencia se manifiesta en aspectos operativos como:

- Mejor canalización de pacientes desde Emergencia.
- Reducción de derivaciones innecesarias a otros servicios.
- Mayor estabilidad en la atención clínica dentro de Medicina Interna.

Por lo tanto, aunque la intervención no altera el resultado final del sistema, sí puede considerarse una mejora en la eficiencia operativa del flujo hospitalario, especialmente en el corto y mediano plazo.

2.- ¿Cómo podría este modelo apoyar la planificación de dietas y recursos nutricionales?

El modelo de cadenas de Markov permite estimar la distribución esperada de pacientes en cada servicio hospitalario a lo largo del tiempo, lo cual resulta fundamental para la planificación de dietas y recursos nutricionales.

Durante la evolución temporal del sistema, el modelo muestra cómo los pacientes transitan entre Emergencia, Medicina Interna, Cirugía y Alta, permitiendo identificar qué servicios concentran una mayor carga asistencial antes del alta. Esta información puede utilizarse para:

- Estimar la demanda diaria de dietas hospitalarias por servicio.
- Ajustar la producción de alimentos según la permanencia esperada de los pacientes.
- Planificar insumos nutricionales específicos (dietas blandas, hiposódicas, enterales, etc.).
- Optimizar la asignación del personal de nutrición clínica.

En particular, el fortalecimiento del servicio de Medicina Interna incrementa el tiempo de permanencia de los pacientes en dicho servicio, lo que implica una mayor demanda de dietas terapéuticas continuas y seguimiento nutricional. El modelo permite anticipar este efecto y adecuar los recursos antes de que ocurran cuellos de botella.

De esta manera, aunque el sistema converge finalmente al alta, el análisis transitorio proporciona una herramienta cuantitativa para la gestión proactiva de la alimentación hospitalaria, mejorando la eficiencia operativa y la calidad de la atención nutricional.

3.- ¿Qué implicancias tendría este cambio en la carga de trabajo del personal de salud?

La intervención hospitalaria orientada a fortalecer el servicio de Medicina Interna modifica principalmente la dinámica temporal del flujo de pacientes, más que la distribución final del

sistema, que converge al estado de alta. Sin embargo, este cambio tiene implicancias relevantes en la carga de trabajo del personal de salud durante el proceso de atención.

El aumento de la probabilidad de permanencia en Medicina Interna implica que los pacientes permanecen más tiempo bajo supervisión clínica continua, lo que genera:

- Mayor demanda de atención médica y de enfermería en Medicina Interna.
- Incremento en la carga de seguimiento clínico, monitoreo y registros.
- Mayor necesidad de coordinación con servicios de apoyo (nutrición, laboratorio, farmacia).

Al mismo tiempo, la reducción de altas directas desde Medicina Interna puede aliviar momentáneamente la presión sobre los servicios administrativos asociados al egreso, pero traslada la carga operativa hacia el personal asistencial del servicio.

Desde el punto de vista de la gestión hospitalaria, el modelo permite anticipar estos cambios y resalta la necesidad de:

- Ajustar la dotación de personal en Medicina Interna.
- Reorganizar turnos y cargas horarias.
- Fortalecer el trabajo interdisciplinario para evitar la saturación del servicio.

En conclusión, aunque el sistema no incrementa el número total de pacientes atendidos a largo plazo, la intervención redistribuye la carga laboral en el tiempo, haciendo indispensable una planificación adecuada del recurso humano.

Parte III

Apéndice

13 Índice H

El índice H (o índice de Hirsch) es un indicador usado para medir la productividad científica y el impacto de un investigador, grupo o institución.

Se basa en la cantidad de publicaciones y el número de citas que estas reciben. Un autor tiene un índice H igual a h si ha publicado h artículos que han sido citados al menos h veces cada uno.

Ejemplo

Si un investigador tiene 10 artículos y al menos 5 de ellos han sido citados 5 veces o más, su índice H = 5. En resumen, el índice H combina cantidad y calidad de la producción científica en una sola medida.

13.1 Autores con índice H con investigaciones en métodos numéricos

Autor	Investigación	Número de publicaciones	Índice H
Hernández-Paricio, Luis Javier	Bivariate Newton-Raphson method and toroidal attraction basins	31	8
Hali, Aissa	Photovoltaic panel parameters determination using two numerical methods	6	3

13.2 Índice H de docentes de la Facultad de Ingeniería Estadística e Informática

Docente	Índice H	Número de publicaciones
Torres-Cruz, Fred	4	40
Coyla-Idme, Leonel	1	5
Tumi-Figueroa, Ernesto Nayer	3	6
Tito Lipa, José Pánfilo	0	3
Canqui-Flores, Bernabé	3	8
Gonzales, Leonid Alemán	0	4
Mendoza-Mollocondo, Charles Ignacio	3	8
Huata-Panca, Percy	2	3
Apaza-Tarqui, Alejandro	1	5
Carpio Vargas, Edgar Eloy	3	9
Javier Quispe Carita, Angel	1	1
López-Cueva, Milton Antonio	1	6
Ibañez-Quispe, Vladimiro	5	21
Melgarejo-Bolivar, Romel P.	3	6
Herrera-Urtiaga, Alain Paul	0	3
Laura Murillo, Ramiro	1	2
Choquejahua-Acero, Remo	1	2
Gonzalo Copari Romero, Fredy	0	2

14 El problema de los caramelos

El siguiente capítulo presenta la simulación de un juego cooperativo en el que varios jugadores comparten una bolsa de caramelos de tres tipos: A, B y C. El objetivo es que, mediante combinaciones e intercambios, el grupo logre formar al menos un chupetín (D) por cada jugador.

Cada jugador inicia con dos caramelos aleatorios. En cada turno, se aplican las siguientes reglas:

- Si en la bolsa hay al menos un caramelo A, uno B y uno C, se forma un chupetín (D) y se retiran esos tres caramelos.
- Si hay dos de cada tipo (AA, BB, CC), se forman dos chupetines (2D) y se añade un caramelo adicional aleatorio a la bolsa.
- Si no es posible formar nuevos chupetines, se puede devolver un chupetín (D) a cambio de tres caramelos aleatorios (uno por cada tipo al azar).
- El juego finaliza cuando se consigue al menos un chupetín por jugador o cuando se alcanza el número máximo de iteraciones.

Este problema ilustra el uso de simulaciones estocásticas y bucles en R para modelar sistemas con componentes aleatorios y condiciones de parada.

Código en R

```r
# ------------------------------------------------------------
# Simulación del juego de caramelos y chupetines en R
# Autor: Ticona Miramira Roberto Angel
# ------------------------------------------------------------

# Genera un caramelo aleatorio (A, B o C)
random_candy <- function() {
    sample(c("A", "B", "C"), 1)
}

# Verifica si hay al menos un grupo A,B,C
can_form_set <- function(bolsa) {
    all(c("A", "B", "C") %in% bolsa)
}

# Elimina un grupo A,B,C
remove_set <- function(bolsa) {
    for (candy in c("A", "B", "C")) {
        idx <- match(candy, bolsa)
        if (is.na(idx)) return(bolsa)
        bolsa <- bolsa[-idx]
    }
```

```
    return(bolsa)
}

# Elimina n ocurrencias de un caramelo
remove_n <- function(bolsa, candy, n) {
    idx <- which(bolsa == candy)
    if (length(idx) >= n) {
        bolsa <- bolsa[-idx[1:n]]
    } else if (length(idx) > 0) {
        bolsa <- bolsa[-idx]
    }
    return(bolsa)
}

# Muestra la bolsa como texto
mostrar_bolsa <- function(bolsa) {
    paste(bolsa, collapse = " ")
}

# --- Programa principal ---
juego_caramelos <- function(n_jugadores = 3, max_iter = 1000000) {
    if (n_jugadores < 3) {
        cat("Debe haber al menos 3 jugadores\n")
        return()
    }

    set.seed(as.integer(Sys.time()))
    bolsa <- replicate(n_jugadores * 2, random_candy())
    D <- 0
    iter <- 0

    cat("\n--- Estado inicial ---\n")
    cat("Bolsa:", mostrar_bolsa(bolsa), "\n")

    while (iter < max_iter && D < n_jugadores) {
        iter <- iter + 1
        cambio <- FALSE

        a <- sum(bolsa == "A")
        b <- sum(bolsa == "B")
        c <- sum(bolsa == "C")

        # Regla de 6 caramelos -> 2D + 1 extra
        dobles <- min(floor(a/2), floor(b/2), floor(c/2))
        while (dobles > 0) {
            bolsa <- remove_n(bolsa, "A", 2)
            bolsa <- remove_n(bolsa, "B", 2)
            bolsa <- remove_n(bolsa, "C", 2)
            D <- D + 2
            extra <- random_candy()
            bolsa <- c(bolsa, extra)
            cambio <- TRUE
```

```
            cat("Regla 6 caramelos -> +2D + extra (", extra, ")\n", sep = "")
            cat("Total D =", D, ", Bolsa:", mostrar_bolsa(bolsa), "\n")

            a <- sum(bolsa == "A")
            b <- sum(bolsa == "B")
            c <- sum(bolsa == "C")
            dobles <- min(floor(a/2), floor(b/2), floor(c/2))
        }

        # Regla simple A,B,C -> 1D
        simples <- min(a, b, c)
        while (simples > 0) {
            bolsa <- remove_set(bolsa)
            D <- D + 1
            cambio <- TRUE
            cat("Se formo un chupetin (D). Total D =", D, "\n")
            cat("Bolsa:", mostrar_bolsa(bolsa), "\n")

            a <- sum(bolsa == "A")
            b <- sum(bolsa == "B")
            c <- sum(bolsa == "C")
            simples <- min(a, b, c)
        }

        # Si no hubo cambios, devolver un D -> +3 caramelos
        if (!cambio && D > 0) {
            D <- D - 1
            nuevos <- replicate(3, random_candy())
            bolsa <- c(bolsa, nuevos)
            cat("Se devolvio un D. Nuevos caramelos:", mostrar_bolsa(nuevos), "\n")
            cat("Total D =", D, ", Bolsa:", mostrar_bolsa(bolsa), "\n")
            cambio <- TRUE
        }

        # Sin movimientos posibles
        if (!cambio && D == 0 && !can_form_set(bolsa)) {
            cat("No hay mas movimientos posibles.\n")
            break
        }

        # Objetivo alcanzado
        if (D >= n_jugadores) {
            cat("Objetivo alcanzado en", iter, "iteraciones.\n")
            break
        }
    }

    cat("\n--- RESULTADOS FINALES ---\n")
    cat("Jugadores:", n_jugadores, "\n")
    cat("Chupetines obtenidos:", D, "\n")
    cat("Caramelos finales:", mostrar_bolsa(bolsa), "\n")
    cat("Iteraciones:", iter, "\n")
```

```
127
128     if (D >= n_jugadores)
129     cat("Todos los jugadores consiguieron su chupetin.\n")
130     else
131     cat("No se logro el objetivo dentro del limite.\n")
132 }
133
134 # Ejemplo de ejecución
135 juego_caramelos(4)
```

Ejecución

```
--- Estado inicial ---
Bolsa: B A A C C A
Se formo un chupetin (D). Total D = 1
Bolsa: A C A
Se devolvio un D. Nuevos caramelos: B C B
Total D = 0 , Bolsa: A C A B C B
Regla 6 caramelos -> +2D + extra (C)
Total D = 2 , Bolsa: C
Se devolvio un D. Nuevos caramelos: C A B
Total D = 1 , Bolsa: C C A B
Se formo un chupetin (D). Total D = 2
Bolsa: C
Se devolvio un D. Nuevos caramelos: B C B
Total D = 1 , Bolsa: C B C B
Se devolvio un D. Nuevos caramelos: A B A
Total D = 0 , Bolsa: C B C B A B A
Regla 6 caramelos -> +2D + extra (A)
Total D = 2 , Bolsa: B A
Se devolvio un D. Nuevos caramelos: B B B
Total D = 1 , Bolsa: B A B B B
Se devolvio un D. Nuevos caramelos: A A C
Total D = 0 , Bolsa: B A B B B A A C
Se formo un chupetin (D). Total D = 1
Bolsa: B B B A A
Se devolvio un D. Nuevos caramelos: C B B
Total D = 0 , Bolsa: B B B A A C B B
Se formo un chupetin (D). Total D = 1
Bolsa: B B A B B
Se devolvio un D. Nuevos caramelos: C A C
Total D = 0 , Bolsa: B B A B B C A C
```

(a) Distribución inicial de los caramelos

```
Regla 6 caramelos -> +2D + extra (A)
Total D = 2 , Bolsa: B B A
Se devolvio un D. Nuevos caramelos: C C C
Total D = 1 , Bolsa: B B A C C C
Se formo un chupetin (D). Total D = 2
Bolsa: B C C
Se devolvio un D. Nuevos caramelos: B A A
Total D = 1 , Bolsa: B C C B A A
Regla 6 caramelos -> +2D + extra (A)
Total D = 3 , Bolsa: A
Objetivo alcanzado en 19 iteraciones.

--- RESULTADOS FINALES ---
Jugadores: 3
Chupetines obtenidos: 3
Caramelos finales en bolsa: A
Iteraciones: 19
Todos los jugadores consiguieron su chupetin.
>
```

(b) Resultado del reparto

Figura 14.1: El problema de los caramelos

Conclusión

Este ejercicio demuestra cómo un sistema aleatorio con reglas definidas puede modelarse mediante un proceso iterativo en R. El número de iteraciones y el éxito del juego dependen del azar, mostrando cómo la simulación permite explorar escenarios variables sin necesidad de resolver el modelo de manera determinista.

15 Aplicación del algoritmo Demons

En el artículo Non-rigid medical image regsitration using image field in Demons altorithm (Lan et al., 2019) se aborda uno de los principales retos en el procesamiento de imágenes médicas: el registro preciso de imágenes deformadas. Este proceso permite alinear múltiples imágenes de un mismo paciente, obtenidas en distintas modalidades o momentos, para mejorar el diagnóstico, el seguimiento de enfermedades y la planificación de tratamientos. Aunque las transformaciones rígidas (traslación y rotación) son útiles en casos simples, la mayoría de escenarios clínicos requieren métodos no rígidos capaces de modelar deformaciones anatómicas complejas.

Entre estos, el algoritmo Demons destaca por su precisión y capacidad para manejar grandes deformaciones. Este modelo interpreta el registro como una fuerza que desplaza cada píxel de la imagen móvil hasta alinearlo con la imagen de referencia. Sin embargo, las versiones clásicas dependen solo del gradiente de intensidad, sin aprovechar la información direccional de las estructuras anatómicas, lo que limita su desempeño en imágenes con texturas complejas o baja variación de intensidad.

Campo de imagen (Image Field): El campo de imagen es una representación integral que combina información de intensidad, gradiente, orientación y frecuencia. Entre estos, el campo de gradiente y el campo de orientación son los más utilizados en imágenes médicas, ya que describen la variación espacial y la dirección predominante de las estructuras anatómicas. Para una imagen digital I, el gradiente se define como:

$$\vec{I} = (dI_x, dI_y), \quad dI_x(i,j) = I(i+1,j) - I(i,j), \quad dI_y(i,j) = I(i,j+1) - I(i,j)$$

donde (i, j) son las coordenadas del píxel. La magnitud del gradiente indica la intensidad de cambio local, mientras que su dirección corresponde a la normal del borde. Este concepto permite modelar la variación direccional de los tejidos, lo que es esencial para un registro más realista.

Métodos Demons basados en registro no rígido: El algoritmo Demons, propuesto por Thirion, se fundamenta en el principio del flujo óptico. La imagen de referencia aplica una fuerza de desplazamiento sobre la imagen deformada para lograr la coincidencia entre ambas. En su formulación clásica, la función objetivo busca minimizar la diferencia de intensidad entre las imágenes y, simultáneamente, imponer suavidad al campo de deformación:

$$E(\vec{u}) = ||R - F \circ \vec{u}||^2 + \sigma^2 ||\vec{u}||^2$$

donde R es la imagen de referencia, F la imagen móvil, $\vec{u}$ el incremento del campo de deformación y σ un parámetro de regularización. El desplazamiento en cada iteración se calcula como:

$$\vec{u} = \frac{||R - F||\,\vec{R}}{||\vec{R}||^2 + ||R - F||^2}$$

En esta expresión, $\vec{R}$ representa el gradiente de la imagen de referencia. Sin embargo, este modelo solo considera la información direccional de R, por lo que resulta adecuado únicamente para deformaciones pequeñas.

Para mejorar esta limitación, Wang et al. incorporaron el gradiente de la imagen móvil F y un parámetro α que ajusta la fuerza de registro:

$$\vec{u} = \frac{||R - F||\,\vec{R}}{||\vec{R}||^2 + \alpha^2||R - F||^2} + \frac{\vec{F}}{||\vec{F}||^2 + \alpha^2||R - F||^2}$$

Posteriormente, Tang et al. añadieron un coeficiente de balance k, que permitió adaptar la fuerza de los demonios de forma más estable:

$$\vec{u} = \frac{||R - F||\,\vec{R}}{k^2||\vec{R}||^2 + \alpha^2||R - F||^2} + \frac{\vec{F}}{k^2||\vec{F}||^2 + \alpha^2||R - F||^2}$$

Modelo propuesto basado en campo de imagen: Con el fin de mejorar la precisión del registro, los autores propusieron incorporar el campo de orientación dentro del modelo de Demons. En lugar de usar las direcciones del campo de gradiente $(\vec{e}_R, \vec{e}_F)$, se utilizan las direcciones del campo de orientación $(\vec{e}_O^R, \vec{e}_O^F)$, combinadas con las magnitudes de los gradientes correspondientes. Así, el modelo se redefine como:

$$\vec{u} = (R - F)\left(\frac{\vec{F}}{k^2\vec{F}^2 + \alpha(R - F)^2}\vec{e}_O^F + \frac{\vec{R}}{k^2\vec{R}^2 + \alpha(R - F)^2}\vec{e}_O^R\right)$$

y puede simplificarse como:

$$\vec{u} = \frac{\sin^2 F \cdot \vec{e}_O^F + \sin^2 R \cdot \vec{e}_O^R}{2k\alpha}$$

Esta formulación permite utilizar la información direccional inherente a las estructuras anatómicas, logrando una estimación más precisa de las deformaciones. El proceso iterativo consta de cinco pasos: cálculo de los campos de gradiente y orientación de R y F; construcción del mapa de diferencia $\alpha(R - F)$; cálculo del campo de deformación $\vec{u}$; aplicación del campo de deformación sobre la imagen móvil; y evaluación de la convergencia mediante una medida de similitud.

Para validar el método, se utilizaron dos conjuntos de datos: 181 pares de imágenes cerebrales por resonancia magnética y 134 pares de imágenes de retina con resolución de 2912×2912 píxeles. Las comparaciones se realizaron frente a los métodos Demons clásico, Demons mejorado por

Wang, Demons con transformación total (TD) y registro basado en B-spline. Los resultados se evaluaron mediante el Error Cuadrático Medio (MSE), la Correlación (Corre), la Información Mutua Normalizada (NMI) y el Error de Junta Media (MJE).

Los resultados mostraron que el método propuesto logró el menor MSE y los valores más altos de Corre y NMI, evidenciando una mejor alineación y preservación de estructuras anatómicas. Además, el análisis de los parámetros α y k mostró que valores intermedios ($\alpha = 1.2, k = 0.9$ para MRI y $\alpha = k = 0.6$ para retina) equilibran precisión y velocidad de convergencia. Valores muy pequeños generan errores elevados, mientras que valores excesivos provocan sobreajuste.

En conclusión, la integración del campo de imagen dentro del algoritmo Demons proporciona una mejora significativa en el registro no rígido de imágenes médicas. Este modelo aprovecha la información direccional de los tejidos para estimar deformaciones más realistas, manteniendo la coherencia estructural sin requerir segmentaciones previas. Por su estabilidad, precisión y eficiencia, el método constituye una contribución relevante al diagnóstico y análisis médico asistido por computadora.

Conclusiones

A lo largo de este libro se ha desarrollado de manera progresiva y estructurada los fundamentos teóricos y prácticos de la programación numérica, partiendo desde conceptos matemáticos básicos como las funciones y sus representaciones, hasta métodos numéricos avanzados aplicados a problemas reales de ingeniería, ciencia y análisis computacional. Esta organización permitió construir una base sólida que facilita la comprensión de métodos más complejos, mostrando cómo la matemática teórica se transforma en herramientas computacionales efectivas.

En la primera unidad se establecieron los conceptos esenciales que sustentan la programación numérica, destacando la importancia de las funciones, las restricciones y los sistemas de ecuaciones como modelos matemáticos fundamentales para describir fenómenos reales. El estudio de métodos para el cálculo de raíces de ecuaciones, como el método de Newton-Raphson, bisección, secante, punto fijo y Regula Falsi, permitió analizar distintas estrategias numéricas, comparar su eficiencia, convergencia y limitaciones, y comprender que la elección de un método adecuado depende del problema específico y de las condiciones iniciales disponibles.

Asimismo, la inclusión de implementaciones en lenguajes como Python y R fortaleció el enfoque práctico del libro, evidenciando cómo los métodos numéricos no solo son conceptos teóricos, sino herramientas computacionales indispensables en la resolución de problemas reales. La visualización gráfica y el análisis del error numérico contribuyeron a una comprensión más profunda del comportamiento de los algoritmos y de sus resultados aproximados.

En la segunda unidad se abordaron temas fundamentales del análisis numérico multivariable, como el gradiente de una función, la diferenciación numérica y la interpolación. Estos contenidos resaltan el papel central de la programación numérica en problemas de optimización, modelado, análisis de datos y simulación. El estudio del gradiente permitió comprender su relevancia en la optimización y en aplicaciones modernas como el aprendizaje automático, mientras que la diferenciación numérica mostró cómo aproximar derivadas cuando las expresiones analíticas no están disponibles o los datos provienen de mediciones experimentales.

Por otro lado, la interpolación numérica se presentó como una herramienta clave para la aproximación de funciones y el análisis de datos discretos, destacando métodos clásicos como Lagrange, Newton y splines cúbicos. El análisis de los errores de interpolación y fenómenos como el de Runge permitió reflexionar sobre las limitaciones de los modelos globales y la necesidad de métodos más estables y precisos en aplicaciones reales.

Finalmente, el estudio de valores y vectores propios consolidó la relación entre el álgebra lineal y la programación numérica, mostrando su importancia en múltiples áreas como la ingeniería, la física, el análisis de sistemas dinámicos y el procesamiento de datos. La implementación computacional de estos conceptos reforzó la idea de que la programación numérica es un puente esencial entre la teoría matemática y su aplicación práctica.

En conclusión, la programación numérica constituye una herramienta indispensable en la formación de profesionales en ciencias e ingeniería, ya que permite abordar problemas complejos que no admiten soluciones analíticas exactas. Este libro busca proporcionar una visión integral, equilibrando teoría, algoritmos y aplicaciones computacionales, con el objetivo de fortalecer el pensamiento crítico, la capacidad de modelar problemas reales y el uso eficiente de métodos numéricos en el contexto académico y profesional.

Bibliografía

Adams, A. (2025). Comparing the Moore–Penrose Pseudoinverse and Gradient Descent for Solving Linear Regression Problems: A Performance Analysis [Preprint, University of Pittsburgh, May 29, 2025]. arXiv preprint, arXiv:2505.23552. https://arxiv.org/abs/2505.23552

Anton, H., Bivens, I., & Davis, S. (2012). Cálculo (10.ª ed.). Wiley.

Atkinson, K. (2009). An Introduction to Numerical Analysis (2nd). Wiley.

Burden, R. L., & Faires, J. D. (2011). Análisis Numérico (9.ª ed.). Cengage Learning.

Burden, R. L., Faires, J. D., & Burden, A. M. (2016). Análisis numérico (10.ª ed.). Cengage Learning.

Chapra, S. C., & Canale, R. P. (2015). Métodos numéricos para ingenieros (7.ª ed.). McGraw-Hill Education.

Cheney, W., & Kincaid, D. (2009). Numerical Mathematics and Computing (6th). Brooks/Cole.

Lan, S., Guo, Z., & You, J. (2019). Non-rigid medical image registration using image field in Demons algorithm. Pattern Recognition Letters, 125, 98-104.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Numerical Recipes: The Art of Scientific Computing (3.ª ed.). Cambridge University Press.

Riley, K. F., Hobson, M. P., & Bence, S. J. (2006). Mathematical Methods for Physics and Engineering (3.ª ed.). Cambridge University Press.

Stewart, J., Redlin, L., & Watson, S. (2001). Precálculo: Matemáticas para el Cálculo (5.ª ed.). Thomson.

Stewart, J., Redlin, L., & Watson, S. (2016). Precálculo: Matemáticas para el Cálculo (7.ª ed.). Cengage Learning.

Strang, G. (2016). Introduction to Linear Algebra (5th). Wellesley-Cambridge Press.

Süli, E., & Mayers, D. F. (2003). An Introduction to Numerical Analysis. Cambridge University Press.

Sullivan, F. (2004). Numerical Methods for Engineers. Prentice Hall.

Este libro invita al lector a explorar la programación numérica de forma clara y práctica, mostrando cómo los conceptos matemáticos fundamentales se traducen en soluciones reales. Se inicia con variables y funciones, definiendo restricciones y explorando soluciones iterativas y métodos para calcular raíces, incluyendo Newton-Raphson.

Luego se presentan técnicas de optimización mediante gradientes, diferenciación e interpolación numérica, así como el análisis de eigenvectores y valores propios. También se introduce a las cadenas de Markov, siempre destacando su aplicación real en problemas de ingeniería y ciencias computacionales.

La obra combina rigor académico con un enfoque aplicado, haciendo que la programación numérica sea comprensible y útil para quienes buscan construir soluciones sólidas y eficientes.

Menos pasos, más comprensión: destilando la programación numérica en sus pilares fundamentales para el mundo real.