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 LEYDI 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 el 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.

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)

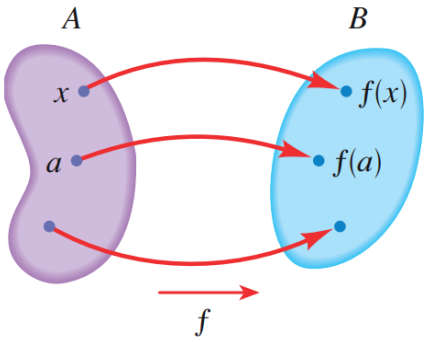


Figura 2.1: Diagrama de flechas de f

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

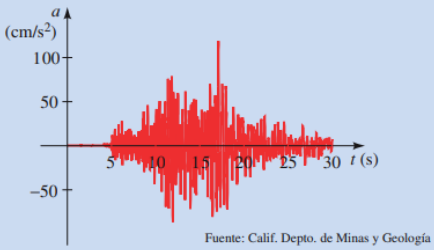
Por medio de una fórmula:

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

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)

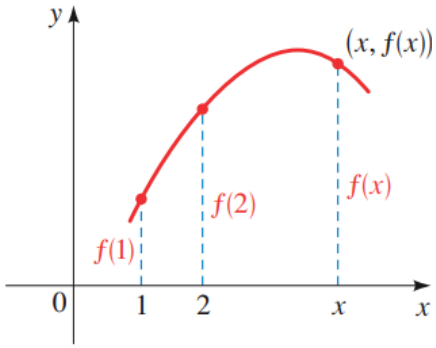


Figura 2.3: La altura de la gráfica arriba del punto x es el valor de $f(x)$

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 $funcinconstante$ porque todos sus valores son el mismo número, es decir, b. Su gráfica es la recta horizontal $y = b$. (Stewart et al., 2016)

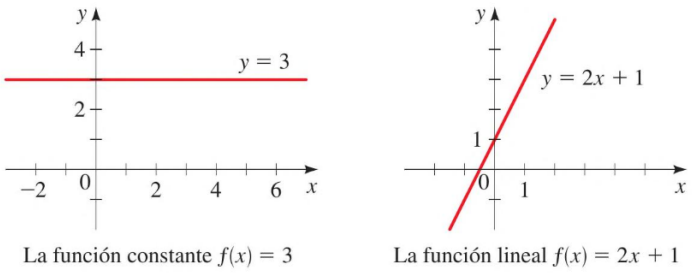


Figura 2.4: Funciones lineal y constante

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.

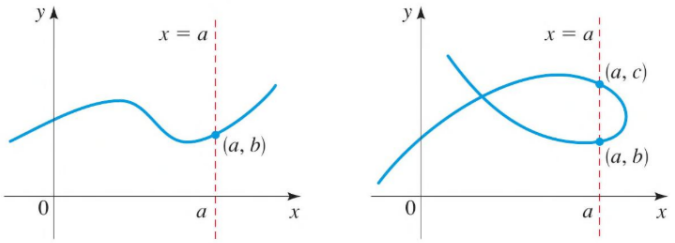


Figura 2.5: Prueba de la recta vertical

2.5 Aplicación

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()
```

```
35 plt.show()
```

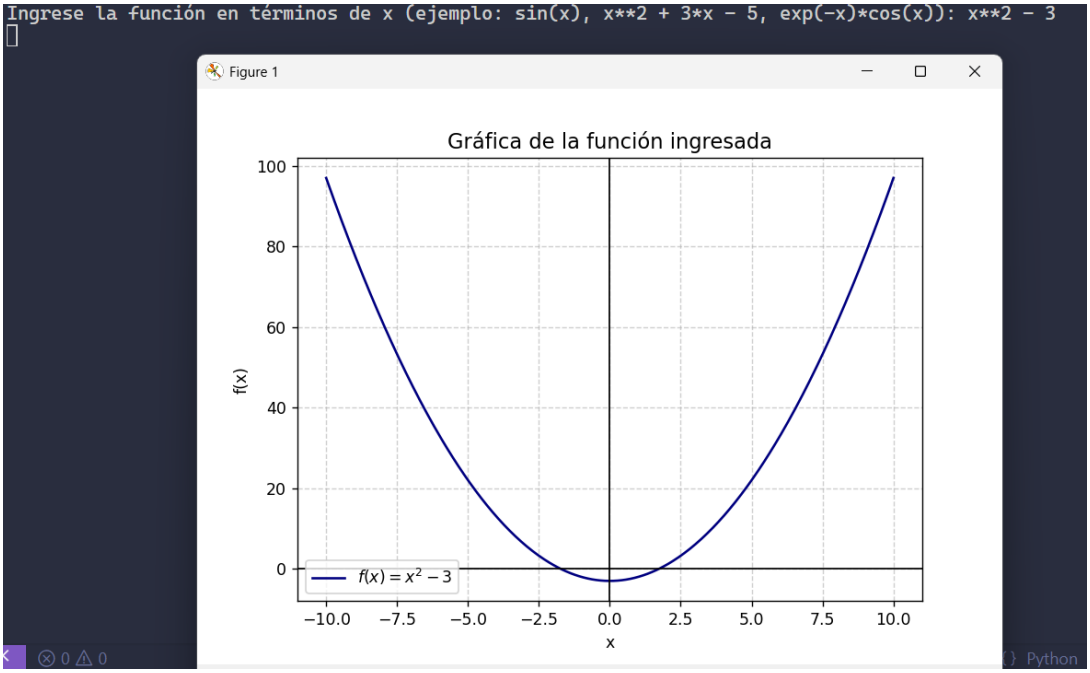


Figura 2.6: Graficando funciones con Python

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

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 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

```
    # 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)")
```

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

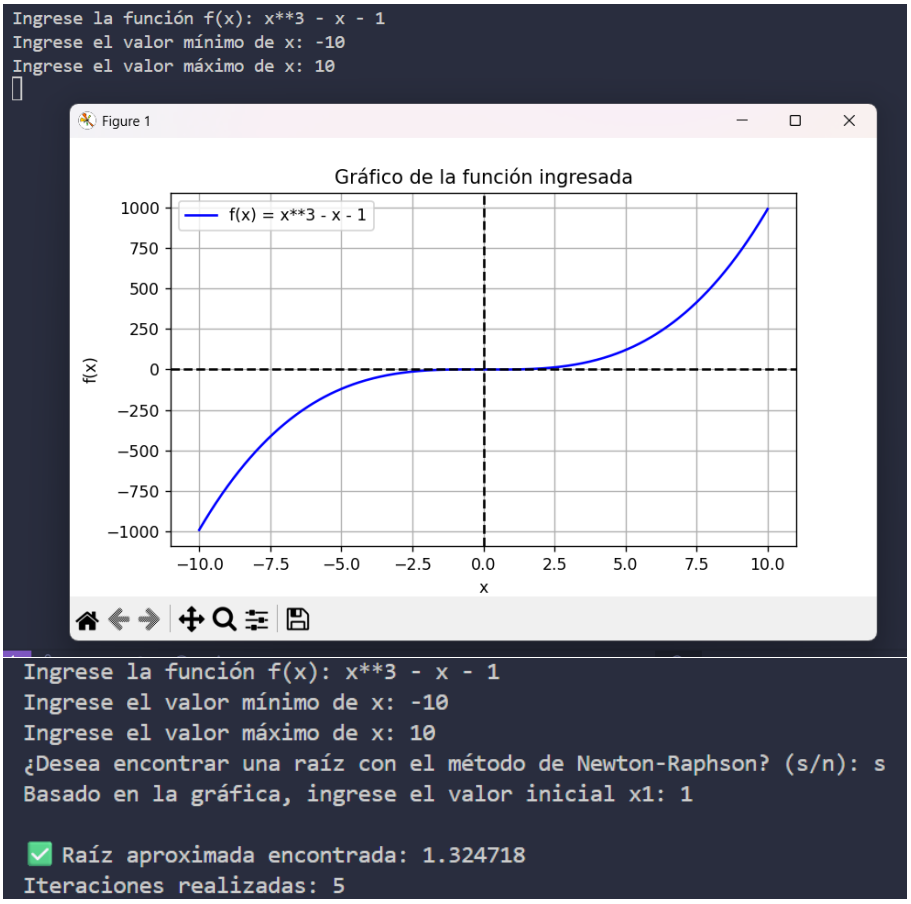


Figura 4.1: Método de Newthon-Rapshon en Python

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

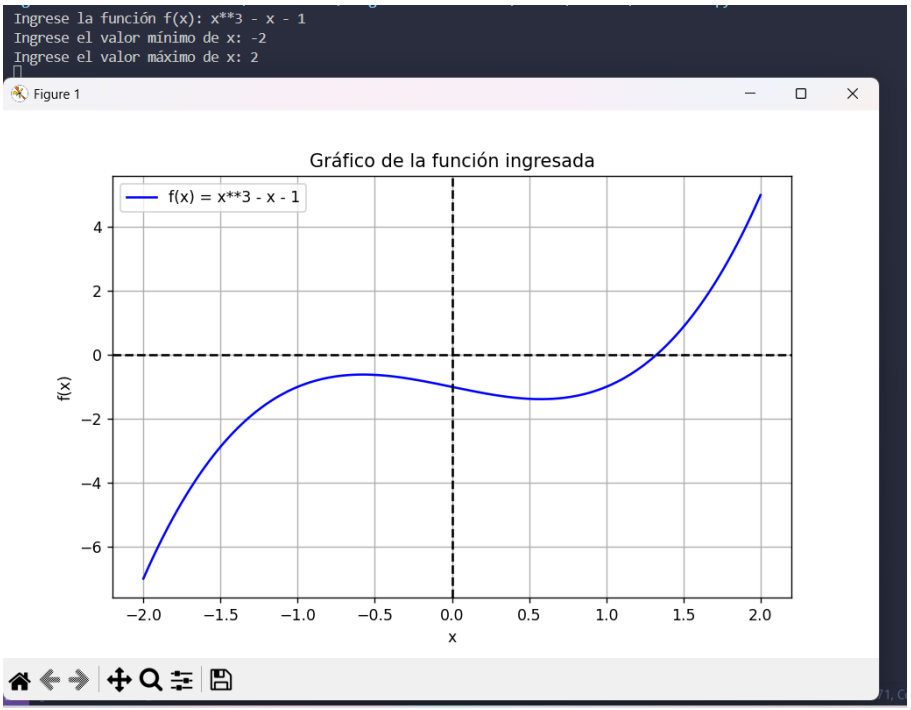
Código

```
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

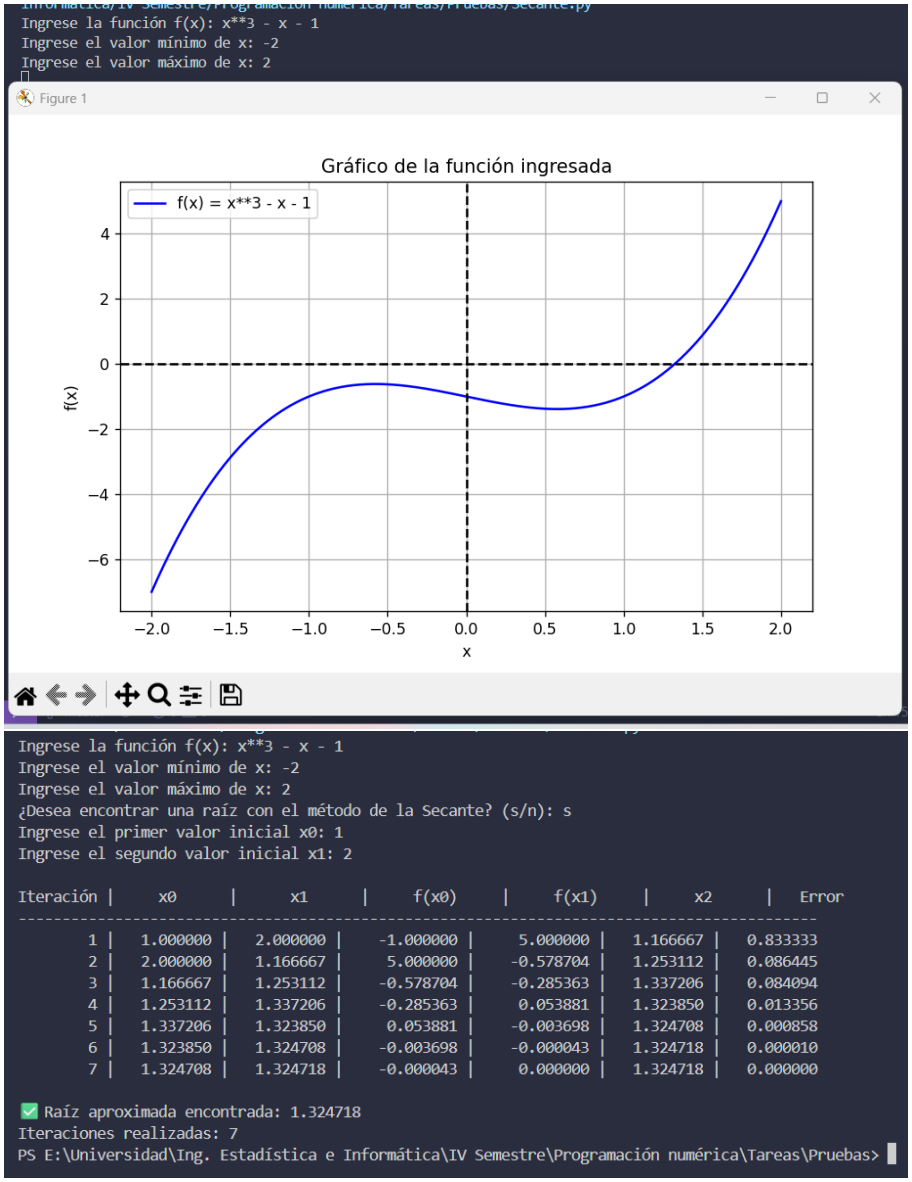


Figura 6.1: Método de la Secante en Python

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

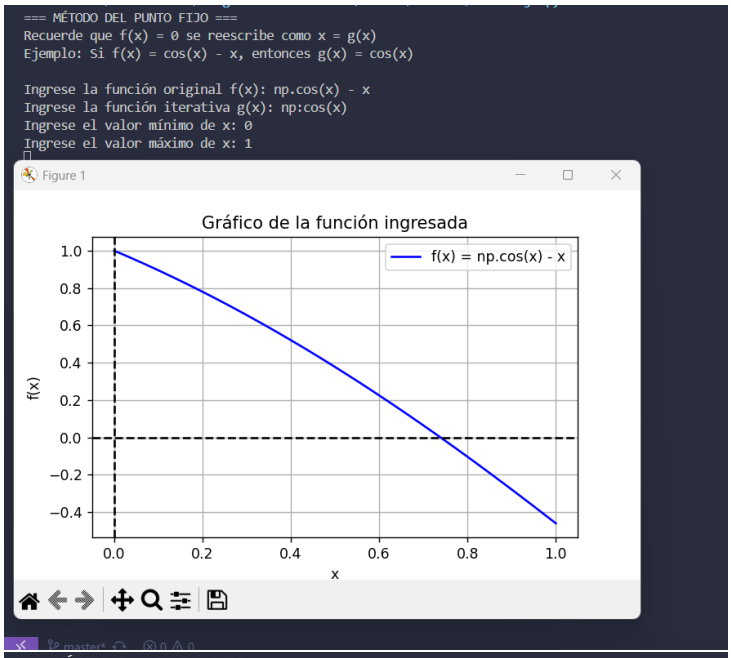
- El método requiere que $|g'(x)| < 1$ en el entorno de la raíz para garantizar la convergencia.
- La función debe ser continua y evaluable en todo el intervalo seleccionado.

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



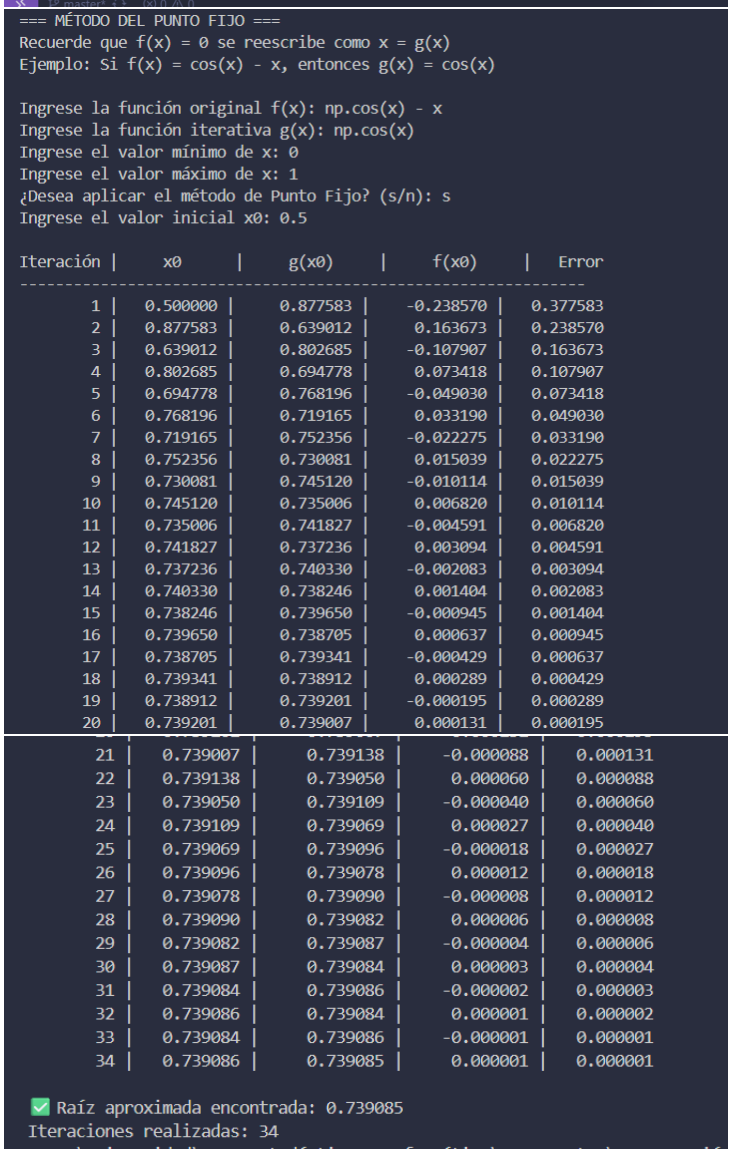


Figura 7.1: Método de Punto Fijo en Python

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

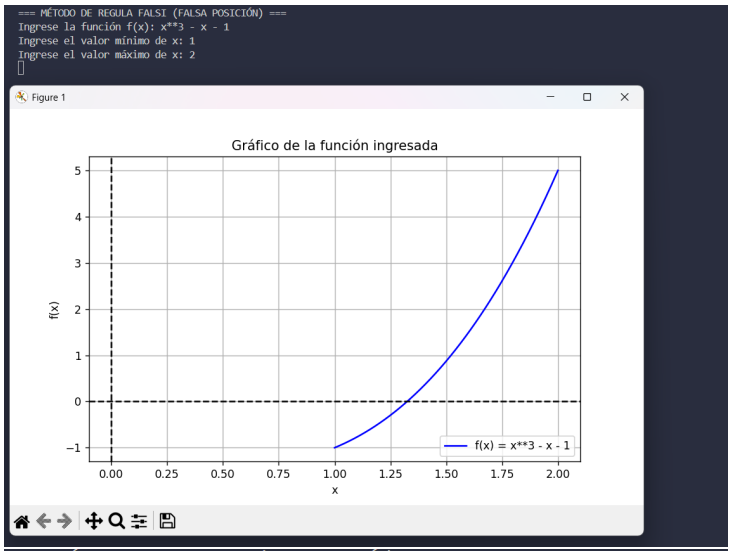
- La función debe ser continua en el intervalo $[a, b]$.
- Debe cumplirse la condición de cambio de signo $f(a)f(b) < 0$.

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

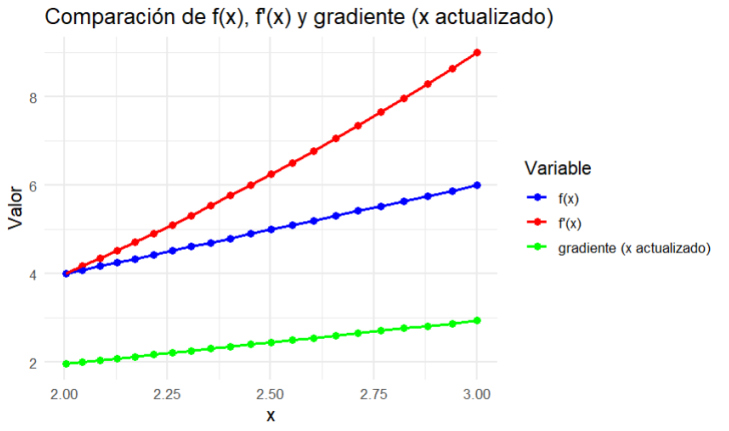


Figura 9.1: Gradiente de una función en R

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 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.

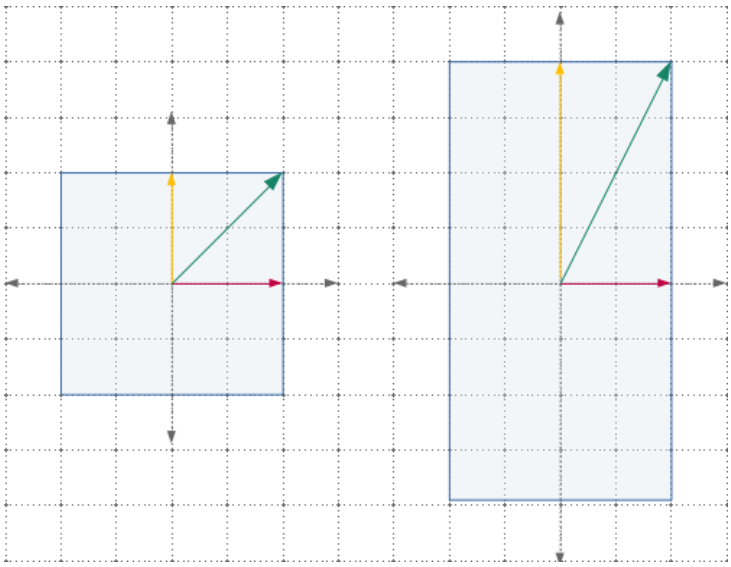


Figura 12.1: Comparación de transformaciones. A la izquierda, un vector común cambia de dirección. A la derecha, un vector propio mantiene su dirección, solo cambia su longitud.

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:

$$\mathbf{A} = \mathbf{P}\mathbf{D}\mathbf{P}^{-1}$$

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 ($\mathbf{D} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P}$) 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:

$$\mathbf{A}^2 = (\mathbf{P}\mathbf{D}\mathbf{P}^{-1})(\mathbf{P}\mathbf{D}\mathbf{P}^{-1}) = \mathbf{P}\mathbf{D}(\mathbf{P}^{-1}\mathbf{P})\mathbf{D}\mathbf{P}^{-1} = \mathbf{P}\mathbf{D}^2\mathbf{P}^{-1}.$$

Generalizando para cualquier potencia k:

$$\mathbf{A}^k = \mathbf{P}\mathbf{D}^k\mathbf{P}^{-1}.$$

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 $\mathbf{A}^{20}$.

Conclusiones

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Bibliografía

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.

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.