UNIVERSIDAD NACIONAL DEL ALTIPLANO

Escuela Profesional de Ingeniería Estadística e Informática

PROGRAMACIÓN NUMÉRICA

Métodos Computacionales y Algoritmos Numéricos

Desarrollo de las Unidades I y II

Luis Angel Quenaya Loza

Wily Caira Huancollo

Curso: Programación Numérica
Cuarto Semestre – 2025

"La programación no se trata solo de código,
se trata de resolver problemas de manera elegante."

Dedicado a tod@s las personas que se aventuran
en el fascinante mundo de la computación numérica.

Programación Numérica
Avance de las sesiones que se desarrollo clase en clase

Este material ha sido elaborado como apoyo académico para el curso de Programación Numérica, a partir del desarrollo de las sesiones correspondientes a ambas unidades del curso.

Índice general

Introducción

La programación numérica cumple un papel fundamental en la formación del ingeniero, ya que permite abordar problemas reales que no pueden resolverse únicamente mediante métodos matemáticos exactos. A través de técnicas numéricas es posible aproximar soluciones, analizar comportamientos y comprender fenómenos presentes en distintas áreas de la ingeniería y la estadística.

Este libro surge como resultado del trabajo académico que hemos desarrollado a lo largo de las sesiones del curso de Programación Numérica, correspondiente al cuarto semestre de la Escuela Profesional de Ingeniería Estadística e Informática. El texto ha sido elaborado de manera conjunta por nosotros dos, como parte del proceso de aprendizaje, con el objetivo de organizar, reforzar y aplicar los conocimientos adquiridos durante el semestre.

El contenido del libro abarca las sesiones trabajadas en las **dos unidades del curso**, integrando de forma progresiva los conceptos fundamentales y los métodos numéricos desarrollados en clase. En la primera unidad se presentan las bases necesarias para comprender el funcionamiento de los métodos numéricos, mientras que en la segunda unidad se profundiza en técnicas más avanzadas y en su aplicación a problemas de mayor complejidad.

A lo largo de los capítulos hemos procurado presentar los temas de manera clara y ordenada, resaltando la importancia del proceso iterativo, el análisis del error y la correcta interpretación de los resultados. Nuestro objetivo es que el lector no solo conozca los procedimientos, sino que también comprenda el razonamiento que hay detrás de cada método y sus alcances.

Este libro representa, finalmente, un esfuerzo académico conjunto orientado a consolidar los aprendizajes obtenidos durante ambas unidades del curso y a servir como material de apoyo y consulta para los estudiantes de la Escuela Profesional de Ingeniería Estadística e Informática que cursan o se inician en el estudio de la programación numérica.

Recursos en Línea

Repositorio del libro: Todos los códigos y notebooks están disponibles en los siguientes repositorios:

- Repositorio](https://github.com/LuisQuenaya/Portafolio-PNRepositorio) de Luis Quenaya
- Repositorio](https://wilycaira56-rgb.github.io/programacion-wily/Repositorio) de Wily Caira

Parte I

Unidad I: Fundamentos

Capítulo 1

Introducción a la Programación Numérica

1.1. ¿Qué es la programación numérica?

La programación numérica es el uso de técnicas computacionales y algoritmos para encontrar soluciones aproximadas a problemas matemáticos complejos que no pueden resolverse de forma analítica. Implica transformar un problema real en un problema matemático y luego en un problema algebraico, para que una computadora pueda calcular una solución cuantitativa mediante operaciones básicas como sumas y restas. Se aplica principalmente en campos como la ingeniería y la ciencia para resolver problemas difíciles como ecuaciones diferenciales complejas o integrales sin solución analítica.

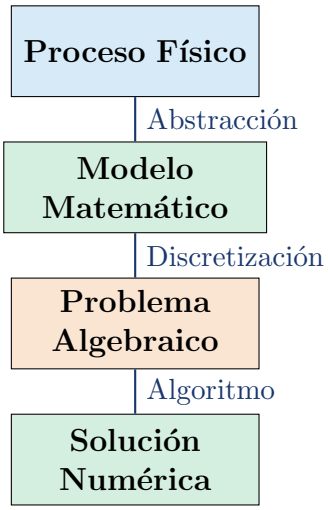


Figura 1.1: Flujo del proceso en programación numérica: del mundo real a la solución computacional

Áreas de aplicación principales

- **Ingeniería**: Análisis estructural, dinámica de fluidos, electromagnetismo computacional
- **Física**: Simulación de sistemas cuánticos, dinámica molecular, astrofísica

- **Economía**: Optimización de portafolios, modelos econométricos, finanzas cuantitativas
- **Ciencias Biológicas**: Modelado epidemiológico, bioinformática, dinámica poblacional
- **Inteligencia Artificial**: Redes neuronales, aprendizaje automático, visión computacional
- **Medio Ambiente**: Modelos climáticos, simulación de contaminación, recursos hídricos

Dato Curioso

Cuando George Dantzig presentó el método Símplex en 1947, muchos matemáticos pensaron que el algoritmo era "demasiado perfecto" que en algún momento fallaría. Sin embargo, más de 75 años después, el Símplex sigue siendo uno de los métodos más usados y eficientes para resolver problemas de optimización en el mundo real. Su robustez y eficacia han hecho que se siga utilizando en aplicaciones que van desde la logística hasta la planificación financiera.

Definición 1.1 (Error de Aproximación)**.** El error de aproximación es la diferencia entre el valor exacto y el valor calculado numéricamente:

$$\text{Error absoluto} = |x_{\text{exacto}} - x_{\text{aproximado}}|$$

$$\text{Error relativo} = \frac{|x_{\text{exacto}} - x_{\text{aproximado}}|}{|x_{\text{exacto}}|} \quad (\text{si } x_{\text{exacto}} \neq 0)$$

El error relativo es particularmente importante cuando se trabaja con magnitudes de diferentes órdenes de tamaño.

Tip de Programación

En programación numérica, es crucial distinguir entre:

- **Precisión**: Número de dígitos significativos correctos
- **Exactitud**: Qué tan cerca está el resultado del valor verdadero
- **Estabilidad**: Cómo se propagan los errores durante el cálculo

1.2. Variables y Funciones en Programación Numérica

La programación numérica se apoya en dos conceptos fundamentales: las **variables** y las **funciones**. Estos elementos permiten traducir problemas matemáticos y situaciones reales a un lenguaje que la computadora pueda interpretar, procesar y resolver mediante algoritmos.

1.2.1. Variables: el corazón del cálculo numérico

En programación numérica, una **variable** es un espacio en memoria donde se almacena un valor numérico que puede cambiar a lo largo de un proceso de cálculo. A diferencia de una constante, su valor evoluciona conforme el algoritmo avanza, especialmente en métodos iterativos.

Desde un punto de vista matemático, las variables representan incógnitas, aproximaciones o parámetros del problema.

En muchos algoritmos numéricos, una variable se actualiza repetidamente mediante una función:

$$x_{\text{siguiente}} = f(x_{\text{actual}})$$

Usando notación iterativa:

$$x^{(k+1)} = F\left(x^{(k)}\right), \quad k = 0, 1, 2, \ldots$$

Cada iteración mejora la aproximación hasta alcanzar una solución aceptable.

</> Tip de Programación

En un método numérico, las variables se utilizan para:

- Guardar valores iniciales del problema
- Almacenar resultados parciales
- Controlar ciclos y condiciones
- Medir errores y convergencia
- Representar magnitudes físicas o reales

Tipos de variables y su uso

Las variables pueden representar diferentes tipos de información según el problema:

- **Variables enteras**: usadas como contadores, índices o iteraciones.
- **Variables de punto flotante**: empleadas en cálculos científicos y aproximaciones.
- **Variables complejas**: frecuentes en ingeniería eléctrica y análisis de señales.

Tipo	Memoria	Precisión	Aplicación típica
`float32`	4 bytes	∼7 dígitos	Simulación rápida, gráficos
`float64`	8 bytes	∼16 dígitos	Cálculo científico
`int32`	4 bytes	Exacta	Contadores
`complex128`	16 bytes	Alta	Señales, sistemas dinámicos

Cuadro 1.1: Tipos comunes de variables numéricas

Ejemplo visual del comportamiento de una variable

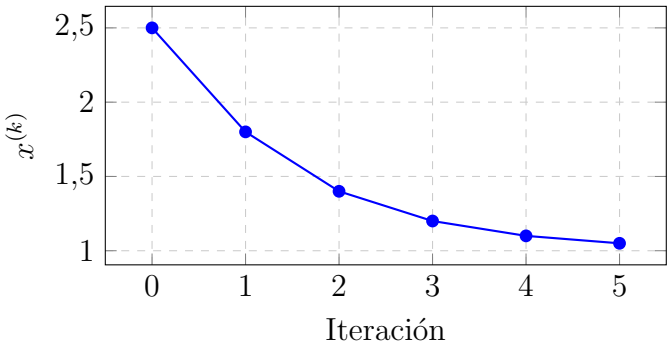


Figura 1.2: Evolución de una variable en un método iterativo hasta converger

1.2.2. Funciones: el modelo matemático del problema

Una **función** es una relación matemática que asigna uno o varios valores de entrada a un valor de salida. En programación numérica, las funciones describen el fenómeno que se desea estudiar: una ecuación, un sistema físico, un modelo económico o un proceso natural.

Formalmente, una función escalar se define como:

$$f : \mathbb{R}^n \to \mathbb{R}$$

Para sistemas más complejos:

$$\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$$

Funciones más utilizadas en análisis numérico

- **Funciones objetivo**: se desean minimizar o maximizar.
- **Funciones residuales**: buscan ceros o raíces.
- **Funciones de interpolación**: aproximan datos discretos.
- **Funciones de ajuste**: modelan datos experimentales.

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

Esta función presenta múltiples raíces y un comportamiento no lineal, típico en problemas reales.

Interpretación gráfica de una función

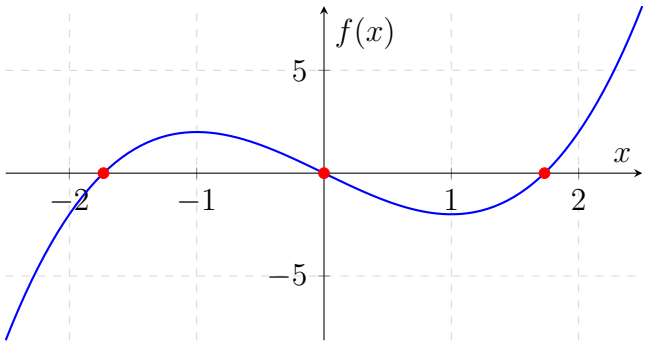


Figura 1.3: Gráfica de $f(x) = x^3 - 3x$ mostrando sus raíces

Relación entre variables y funciones

Las variables actúan como **portadores de información**, mientras que las funciones determinan **cómo esa información se transforma**. En conjunto, permiten construir algoritmos que aproximan soluciones de problemas complejos que no pueden resolverse de forma exacta.

Este vínculo es la base de los métodos numéricos modernos y del cálculo científico asistido por computadora.

Implementación de funciones en código

```
import numpy as np
import math

# 1. Funci n polin mica (f cil de derivar)
def funcion_polinomio(x):
"""f(x) = x^3 - 2x^2 + 3x - 5"""
return x**3 - 2*x**2 + 3*x - 5

def derivada_polinomio(x):
"""f'(x) = 3x^2 - 4x + 3"""
return 3*x**2 - 4*x + 3

# 2. Funci n trascendente (com n en f sica)
def funcion_oscilador(x, A=1.0, omega=2.0, phi=0.0):
"""Oscilador amortiguado: f(x) = A * exp(-x) * sin(
   omega*x + phi)"""
return A * math.exp(-x) * math.sin(omega*x + phi)

# 3. Funci n por partes (definida en intervalos)
def funcion_por_partes(x):
"""Funci n definida diferentemente en cada regi n"""
if x < 0:
return x**2  # Par bola para x negativo
elif x < 2:
return math.sqrt(x) + 1  # Ra z  m s constante
```

```
        else:
        return math.log(x + 1)  # Logaritmo para x grande

        # 4. Funci n multivariable
        def funcion_multivariable(x, y, z):
        """f(x,y,z) = x^2 + y^2 + z^2 - 2xy + 3z"""
        return x**2 + y**2 + z**2 - 2*x*y + 3*z

        # 5. Funci n que devuelve m ltiples valores
        def funcion_y_derivada(x):
        """Calcula f(x) y f'(x) simult neamente (eficiente)"""
        fx = math.sin(x) * math.exp(-x**2)
        dfx = math.exp(-x**2) * (math.cos(x) - 2*x*math.sin(x))
        return fx, dfx
```

Listing 1.1: Implementación de funciones en Python

```
        # 1. Funci n polin mica
        funcion_polinomio <- function(x) {
            # f(x) = x^3 - 2x^2 + 3x - 5
            return(x^3 - 2*x^2 + 3*x - 5)
        }

        derivada_polinomio <- function(x) {
            # f'(x) = 3x^2 - 4x + 3
            return(3*x^2 - 4*x + 3)
        }

        # 2. Funci n trascendente
        funcion_oscilador <- function(x, A=1.0, omega=2.0, phi
           =0.0) {
            # Oscilador amortiguado: f(x) = A * exp(-x) * sin(
               omega*x + phi)
            return(A * exp(-x) * sin(omega*x + phi))
        }

        # 3. Funci n por partes
        funcion_por_partes <- function(x) {
            # Funci n definida diferentemente en cada regi n
            if (x < 0) {
                return(x^2)  # Par bola para x negativo
            } else if (x < 2) {
                return(sqrt(x) + 1)  # Ra z  m s constante
            } else {
                return(log(x + 1))  # Logaritmo para x grande
            }
        }

        # 4. Funci n multivariable
        funcion_multivariable <- function(x, y, z) {
            # f(x,y,z) = x^2 + y^2 + z^2 - 2xy + 3z
            return(x^2 + y^2 + z^2 - 2*x*y + 3*z)
```

```r
}

# 5. Funci n que devuelve lista con valores y derivada
funcion_completa <- function(x) {
    fx <- sin(x) * exp(-x^2)
    dfx <- exp(-x^2) * (cos(x) - 2*x*sin(x))
    return(list(valor = fx, derivada = dfx))
}
```

Listing 1.2: Implementación de funciones en R

Interacción entre variables y funciones: Ejemplo práctico

Algoritmo

Algoritmo: Método iterativo simple (Punto Fijo)

1. **Entrada**: Función $g(x)$, valor inicial x_0, tolerancia ϵ, máximo iteraciones N
2. **Inicializar**: $x \leftarrow x_0$, $iter \leftarrow 0$, $error \leftarrow \infty$
3. **Mientras** $error > \epsilon$ y $iter < N$:
   a) $x_{\text{nuevo}} \leftarrow g(x)$ (Aplicar función para transformar variable)
   b) $error \leftarrow |x_{\text{nuevo}} - x|$ (Calcular cambio)
   c) $x \leftarrow x_{\text{nuevo}}$ (Actualizar variable)
   d) $iter \leftarrow iter + 1$ (Incrementar contador)
4. **Salida**: x (aproximación final), $iter$ (iteraciones usadas)

```python
def metodo_punto_fijo(g, x0, tol=1e-8, max_iter=1000):
"""
M todo del punto fijo para resolver x = g(x)

Variables principales:
- x: aproximaci n actual (se actualiza)
- x_nuevo: nueva aproximaci n (temporal)
- error: diferencia entre iteraciones
- iter: contador de iteraciones

Funci n transformadora:
- g: funci n que define la iteraci n
"""

# Variables iniciales
x = float(x0)
iteracion = 0
error = float('inf')
historial = []

print(f"Inicio: x0 = {x0}")
```

```
        print("-" * 50)

        # Bucle principal
        while error > tol and iteracion < max_iter:
        # Aplicar funci n para transformar variable
        x_nuevo = g(x)

        # Calcular error (cambio)
        error = abs(x_nuevo - x)

        # Guardar progreso
        historial.append({
            'iteracion': iteracion,
            'x': x,
            'x_nuevo': x_nuevo,
            'error': error
        })

        # Mostrar progreso
        if iteracion % 10 == 0:  # Cada 10 iteraciones
        print(f"Iter {iteracion:3d}: x = {x:.10f}, error = {
           error:.2e}")

        # Actualizar variable principal
        x = x_nuevo
        iteracion += 1

        # Resultados finales
        print("-" * 50)
        print(f"Convergencia alcanzada en {iteracion}
           iteraciones")
        print(f"Soluci n aproximada: x = {x}")
        print(f"Error final: {error}")

        return x, historial

        # Ejemplo de uso
        def ejemplo_g(x):
        """Funci n para iteraci n: g(x) = cos(x)"""
        import math
        return math.cos(x)

        # Aplicar el m todo
        solucion, progreso = metodo_punto_fijo(ejemplo_g, x0
           =0.5)
```

Listing 1.3: Implementación del algoritmo

¡Atención! Error Común

En Python, los índices comienzan en 0, mientras que en R comienzan en 1. Este es uno de los errores más comunes al cambiar entre lenguajes:
Python: `array[0]` es el primer elemento
R: `vector[1]` es el primer elemento
MATLAB: `array(1)` es el primer elemento
Julia: `array[1]` es el primer elemento (como R)
Fortran: Por defecto `array(1)` es el primer elemento

```python
# Python: ndices base 0
lista = [10, 20, 30, 40, 50]
print(lista[0])   #     10 (primer elemento)
print(lista[1])   #     20 (segundo elemento)
```

```r
# R: ndices base 1
vector <- c(10, 20, 30, 40, 50)
print(vector[1])  #     10 (primer elemento)
print(vector[2])  #     20 (segundo elemento)
```

Consejo: Al portar código entre lenguajes, siempre verifica y ajusta los índices. Usa comentarios claros para documentar esta diferencia.

Resumen y Buenas Prácticas

Nota Importante

Buenas prácticas para variables y funciones:

- **Para variables**:
  - Usa nombres descriptivos (`tolerance`, no `tol`)
  - Inicializa siempre las variables antes de usarlas
  - Escoge el tipo numérico apropiado para la precisión necesaria
  - Documenta las unidades físicas cuando aplique
- **Para funciones**:
  - Documenta el dominio y rango esperados
  - Maneja casos especiales (divisiones por cero, log(0), etc.)
  - Considera la eficiencia (evita cálculos repetidos)
  - Implementa versiones vectorizadas cuando sea posible
- **Para la interacción**:
  - Las funciones transforman variables, variables almacenan estados
  - Mantén el acoplamiento bajo entre componentes
  - Usa valores por defecto sensatos en parámetros
  - Valida entradas y verifica salidas

Recursos en Línea

Recursos para profundizar:

- NumPy: Biblioteca fundamental para computación científica en Python
- The R Project: Entorno para computación estadística
- SciPy: Colección de algoritmos matemáticos para Python
- Julia: Lenguaje moderno para computación científica
- MATLAB: Ambiente para computación técnica
- Libro: "Numerical Recipes: The Art of Scientific Computing"

Capítulo 2

Restricciones

2.1. Definición:

Las restricciones son condiciones que limitan el conjunto de valores posibles que pueden tomar las variables en un problema. Estas son esenciales en problemas de optimización y en sistemas de ecuaciones, ya que definen la región factible, es decir, el espacio de soluciones posibles.

Nota Importante

En programación numérica, las restricciones son cruciales porque:

- Delimitan el espacio de búsqueda de soluciones
- Aseguran que las soluciones sean viables en el contexto real
- Guían los algoritmos hacia soluciones factibles
- Pueden convertir problemas ilimitados en acotados

2.2. Tipos de Restricciones

- **Restricciones de Igualdad:** Representan condiciones exactas. Por ejemplo:

$$x + y = 10$$

donde la suma de dos variables debe ser exactamente 10.

Tip de Programación

Las restricciones de igualdad reducen el grado de libertad del problema. Si tienes n variables y m ecuaciones de igualdad, el espacio de soluciones tiene dimensión $n - m$.

- **Restricciones de Desigualdad:** Permiten límites superiores o inferiores. Por ejemplo:

$$x + y \leq 10$$

donde la suma de dos variables no debe exceder 10.

Nota Importante

Las desigualdades pueden ser de dos tipos:

- $\leq$ o $\geq$: Incluyen el límite (cerrado)
- $<$ o $>$: Excluyen el límite (abierto)

En la práctica, la mayoría de algoritmos numéricos trabajan con desigualdades no estrictas ($\leq$, $\geq$).

- **Restricciones de Límites:** Definen rangos para las variables. Por ejemplo:

$$0 \leq x \leq 5$$

donde la variable x debe estar entre 0 y 5.

¡Atención! Error Común

Los límites son esenciales para evitar soluciones numéricamente inestables o matemáticamente sin sentido (como tiempos negativos, probabilidades mayores que 1, etc.).

2.2.1. Región Factible

Definición 2.1 (Región Factible)**.** Es el conjunto de todos los puntos que satisfacen todas las restricciones simultáneamente. Se denota como:

$$\mathcal{F} = \{\mathbf{x} \in \mathbb{R}^n : g_i(\mathbf{x}) \leq 0, h_j(\mathbf{x}) = 0\}$$

donde g_i son restricciones de desigualdad y h_j son restricciones de igualdad.

Para las restricciones:

$$\begin{cases} x + y \leq 10 \\ x \geq 2 \\ y \geq 3 \end{cases}$$

Dibuja la región factible en el plano XY y encuentra al menos 3 puntos que pertenezcan a ella.

Capítulo 3

Sistemas de Ecuaciones

3.1. Definición:

Un sistema de ecuaciones es un conjunto de ecuaciones que comparten las mismas variables. Resolver un sistema consiste en encontrar los valores de las variables que satisfacen todas las ecuaciones simultáneamente.

Dato Curioso

El problema de resolver sistemas lineales es tan antiguo como las matemáticas. Los babilonios ya resolvían sistemas de dos ecuaciones hace más de 4000 años usando métodos equivalentes a la eliminación gaussiana moderna.

3.1.1. Clasificación de Sistemas

- **Compatible determinado:** Tiene solución única
- **Compatible indeterminado:** Tiene infinitas soluciones
- **Incompatible:** No tiene solución

Algoritmo

Algoritmo: Resolución de sistemas lineales

1. Verificar si el sistema es compatible (teorema de Rouché-Frobenius)
2. Elegir método: sustitución, igualación, reducción, o métodos numéricos
3. Aplicar el método seleccionado
4. Verificar la solución sustituyendo en las ecuaciones originales

3.1.2. Ejemplo Básico: Resolución por Sustitución

Supongamos el siguiente sistema:

$$x + y = 10 \quad \text{(Ecuación 1)} \tag{3.1}$$
$$x - y = 4 \quad \text{(Ecuación 2)} \tag{3.2}$$

Paso 1: Resolver una ecuación para una variable. De la ecuación (1), despejamos x:

$$x = 10 - y \tag{3}$$

Paso 2: Sustituir en la segunda ecuación. Sustituimos $x = 10 - y$ en (2):

$$(10 - y) - y = 4 \tag{4}$$
$$10 - 2y = 4 \tag{5}$$
$$-2y = -6 \tag{6}$$
$$y = 3 \tag{7}$$

Paso 3: Sustituir el valor de y en la primera ecuación. Sustituimos $y = 3$ en (1):

$$x + 3 = 10 \tag{8}$$
$$x = 7 \tag{9}$$

Paso 4: Verificación:

$$7 + 3 = 10 \quad \checkmark$$
$$7 - 3 = 4 \quad \checkmark$$

Solución: $x = 7,\ y = 3$.

3.1.3. Representación Gráfica

A continuación, graficamos ambas ecuaciones para visualizar la solución:

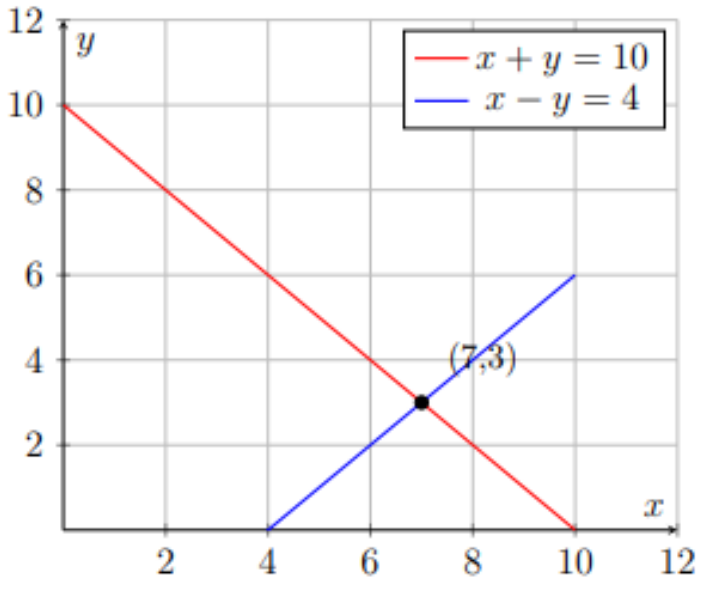


Figura 3.1: Representación gráfica del sistema de ecuaciones. La solución es el punto de intersección (7,3)

3.1.4. Solución en Python

```python
# Resolvemos manualmente paso a paso
# Sistema: x + y = 10, x - y = 4

print("Resolviendo paso a paso:")
print("1) x + y = 10")
print("2) x - y = 4")

print("\nPaso 1: De la primera ecuaci n:")
print("x = 10 - y")

print("\nPaso 2: Sustituimos en la segunda:")
print("(10 - y) - y = 4")
print("10 - 2y = 4")
print("-2y = 4 - 10")
print("-2y = -6")
print("y = (-6) / (-2) = 3")

print("\nPaso 3: Encontramos x:")
print("x = 10 - 3 = 7")

print("\nSoluci n final:")
print("x = 7, y = 3")

print("\nComprobaci n:")
print("7 + 3 = 10")
print("7 - 3 = 4 ")
```

Listing 3.1: Solución manual paso a paso

3.1.5. Solución en R

```r
# Resolvemos manualmente paso a paso
cat("Resolviendo paso a paso:\n")
cat("1) x + y = 10\n")
cat("2) x - y = 4\n")

cat("\nPaso 1: De la primera ecuaci n:\n")
cat("x = 10 - y\n")

cat("\nPaso 2: Sustituimos en la segunda:\n")
cat("(10 - y) - y = 4\n")
cat("10 - 2y = 4\n")
cat("-2y = 4 - 10\n")
cat("-2y = -6\n")
cat("y = (-6) / (-2) = 3\n")

cat("\nPaso 3: Encontramos x:\n")
cat("x = 10 - 3 = 7\n")

```

```
19          cat("\nSoluci n final:\n")
20          cat("x = 7, y = 3\n")
21
22          cat("\nComprobaci n:\n")
23          cat("7 + 3 = 10    \n")
24          cat("7 - 3 = 4     \n")
```

Listing 3.2: Solución manual paso a paso en R

3.1.6. Ejemplo Práctico: Tiempo de Estudio

Un estudiante dispone de 10 horas al día para estudiar Matemáticas (x) y Programación (y). Además:

- Debe dedicar al menos 2 horas a Matemáticas:

$$x \geq 2$$

- Debe dedicar al menos 3 horas a Programación:

$$y \geq 3$$

- El tiempo total no puede exceder 10 horas:

$$x + y \leq 10$$

El estudiante quiere maximizar su aprendizaje, que modela con la función:

$$A(x, y) = 3x + 4y$$

donde:

- $3x$: Puntos de aprendizaje de Matemáticas (3 puntos por hora)
- $4y$: Puntos de aprendizaje de Programación (4 puntos por hora)

Encuentra la combinación óptima de horas para maximizar $A(x, y)$ sujeto a las restricciones dadas.

3.1.7. Solución Manual del Problema

Primero encontramos los vértices de la región factible:

1. Intersección de $x = 2$ y $y = 3$: Punto $(2, 3)$
2. Intersección de $x = 2$ y $x + y = 10$: Si $x = 2$, entonces $2 + y = 10$, $y = 8 \rightarrow$ Punto $(2, 8)$
3. Intersección de $y = 3$ y $x + y = 10$: Si $y = 3$, entonces $x + 3 = 10$, $x = 7 \rightarrow$ Punto $(7, 3)$

Ahora evaluamos la función objetivo $A(x, y) = 3x + 4y$ en cada vértice:

- En $(2, 3)$: $A = 3(2) + 4(3) = 6 + 12 = 18$ puntos
- En $(2, 8)$: $A = 3(2) + 4(8) = 6 + 32 = 38$ puntos
- En $(7, 3)$: $A = 3(7) + 4(3) = 21 + 12 = 33$ puntos

Solución: La combinación óptima es $x = 2$ horas de Matemáticas y $y = 8$ horas de Programación, obteniendo 38 puntos de aprendizaje.

3.1.8. Solución con Python

```
        # Problema: Maximizar aprendizaje A = 3x + 4y
        # Restricciones:
        # 1) x >= 2
        # 2) y >= 3
        # 3) x + y <= 10

        print("PROBLEMA DE OPTIMIZACI N DEL TIEMPO DE ESTUDIO")
        print("=" * 50)

        # Definir v rtices de la regi n factible
        print("\nV rtices de la regi n factible:")
        vertices = [
        (2, 3),    # Intersecci n x=2 y y=3
        (2, 8),    # Intersecci n x=2 y x+y=10
        (7, 3)     # Intersecci n y=3 y x+y=10
        ]

        print(f"1) (2, 3)")
        print(f"2) (2, 8)")
        print(f"3) (7, 3)")

        # Funci n objetivo: A(x,y) = 3x + 4y
        def calcular_aprendizaje(x, y):
        return 3*x + 4*y

        print("\nEvaluando funci n objetivo en cada v rtice:")
        mejor_valor = -float('inf')
        mejor_punto = None

```

```
        for i, (x, y) in enumerate(vertices):
        puntos = calcular_aprendizaje(x, y)
        print(f"V rtice {i+1}: ({x}, {y})")
        print(f"  A({x}, {y}) = 3*{x} + 4*{y} = {puntos} puntos
           ")

        if puntos > mejor_valor:
        mejor_valor = puntos
        mejor_punto = (x, y)

        print("\n" + "=" * 50)
        print("SOLUCI N  PTIMA :")
        print("=" * 50)
        print(f"Horas de Matem ticas (x): {mejor_punto[0]}
           horas")
        print(f"Horas de Programaci n (y): {mejor_punto[1]}
           horas")
        print(f"Total horas: {mejor_punto[0] + mejor_punto[1]}
           horas")
        print(f"Puntos de aprendizaje m ximos: {mejor_valor}
           puntos")

        print("\nVerificaci n de restricciones:")
        print(f"1) x >= 2: {mejor_punto[0]} >= 2    ")
        print(f"2) y >= 3: {mejor_punto[1]} >= 3    ")
        print(f"3) x + y <= 10: {mejor_punto[0]} + {mejor_punto
           [1]} = {mejor_punto[0] + mejor_punto[1]} <= 10    ")
```

Listing 3.3: Optimización del tiempo de estudio en Python

3.1.9. Solución con R

```
        # Problema: Maximizar aprendizaje A = 3x + 4y
        # Restricciones:
        # 1) x >= 2
        # 2) y >= 3
        # 3) x + y <= 10

        cat("PROBLEMA DE OPTIMIZACI N DEL TIEMPO DE ESTUDIO\n"
           )
        cat(rep("=", 50), "\n", sep="")

        # Definir v rtices de la regi n factible
        cat("\nV rtices de la regi n factible:\n")
        vertices <- list(
        c(2, 3),    # Intersecci n x=2 y y=3
        c(2, 8),    # Intersecci n x=2 y x+y=10
        c(7, 3)     # Intersecci n y=3 y x+y=10
        )

        cat("1) (2, 3)\n")
```

```
cat("2) (2, 8)\n")
cat("3) (7, 3)\n")

# Funci n objetivo: A(x,y) = 3x + 4y
calcular_aprendizaje <- function(x, y) {
    return(3*x + 4*y)
}

cat("\nEvaluando funci n objetivo en cada v rtice:\n")
mejor_valor <- -Inf
mejor_punto <- NULL

for (i in 1:length(vertices)) {
    x <- vertices[[i]][1]
    y <- vertices[[i]][2]
    puntos <- calcular_aprendizaje(x, y)

    cat(sprintf("\nV rtice %d: (%d, %d)\n", i, x, y))
    cat(sprintf("  A(%d, %d) = 3*%d + 4*%d = %d puntos\n", x, y, x, y, puntos))

    if (puntos > mejor_valor) {
        mejor_valor <- puntos
        mejor_punto <- c(x, y)
    }
}

cat("\n", rep("=", 50), "\n", sep="")
cat("SOLUCI N PTIMA :\n")
cat(rep("=", 50), "\n", sep="")
cat(sprintf("Horas de Matem ticas (x): %d horas\n", mejor_punto[1]))
cat(sprintf("Horas de Programaci n (y): %d horas\n", mejor_punto[2]))
cat(sprintf("Total horas: %d horas\n", mejor_punto[1] + mejor_punto[2]))
cat(sprintf("Puntos de aprendizaje m ximos: %d puntos\n", mejor_valor))

cat("\nVerificaci n de restricciones:\n")
cat(sprintf("1) x >= 2: %d >= 2    \n", mejor_punto[1]))
cat(sprintf("2) y >= 3: %d >= 3    \n", mejor_punto[2]))
cat(sprintf("3) x + y <= 10: %d + %d = %d <= 10    \n", mejor_punto[1], mejor_punto[2], mejor_punto[1] + mejor_punto[2]))
```

Listing 3.4: Optimización del tiempo de estudio en R

3.1.10. Solución Gráfica del Problema y Análisis de Sensibilidad

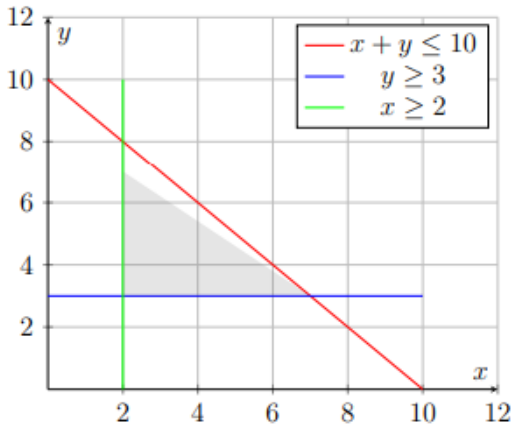


Figura 3.2: La región factible está limitada por las restricciones y cualquier solución debe pertenecer a esta región. Los vértices son los puntos donde se encuentran las restricciones.

Caso 1: Cambiar función objetivo a $A(x, y) = 5x + 3y$

```
# Nueva funci n objetivo: A = 5x + 3y
vertices = [(2, 3), (2, 8), (7, 3)]

print("Nueva funci n objetivo: A = 5x + 3y")
print("=" * 40)

mejor_valor = -float('inf')
mejor_punto = None

for x, y in vertices:
puntos = 5*x + 3*y
print(f"({x}, {y}): 5*{x} + 3*{y} = {puntos}")

if puntos > mejor_valor:
mejor_valor = puntos
mejor_punto = (x, y)

print(f"\nSoluci n  ptima : ({mejor_punto[0]},
    {mejor_punto[1]})")
print(f"Puntos: {mejor_valor}")
```

Listing 3.5: Análisis de sensibilidad en Python

Resultado: Con $A = 5x + 3y$, la solución óptima es $(7, 3)$ con 44 puntos.
Caso 2: Agregar restricción $y \leq 6$

```
# Agregar restricci n: y <= 6
# Los v rtices cambian:
vertices_nuevos = [
(2, 3),    # x=2, y=3
(2, 6),    # x=2, y=6 (nuevo por y<=6)
(4, 6),    # y=6 y x+y=10      x=4
(7, 3)     # y=3 y x+y=10      x=7
]

print("Con nueva restricci n: y <= 6")
print("Nuevos v rtices:", vertices_nuevos)
print("=" * 40)

mejor_valor = -float('inf')
mejor_punto = None

for x, y in vertices_nuevos:
puntos = 3*x + 4*y  # Funci n original
print(f"({x}, {y}): {puntos} puntos")

if puntos > mejor_valor:
mejor_valor = puntos
mejor_punto = (x, y)

print(f"\nSoluci n  ptima : ({mejor_punto[0]},
    {mejor_punto[1]})")
print(f"Puntos: {mejor_valor}")
print(f"Verificaci n y<=6: {mejor_punto[1]} <=
    6    ")
```

3.1.11. Análisis de Sensibilidad

Nota Importante

¿Qué aprendemos de este análisis?

- Si Matemáticas vale más puntos (5 vs 3), conviene estudiar más Matemáticas
- Si Programación tiene límite (máximo 6 horas), cambia la solución óptima
- La solución siempre está en un vértice de la región factible
- Cambiar coeficientes puede cambiar completamente la solución

3.1.12. Aplicaciones en la Vida Real

Dato Curioso

Los problemas de optimización con restricciones se usan en:

- **Logística:** Optimización de rutas con restricciones de tiempo y capacidad
- **Finanzas:** Maximizar retornos con restricciones de riesgo
- **Producción:** Maximizar ganancias con restricciones de recursos
- **Dieta:** Minimizar costo con restricciones nutricionales
- **Ingeniería:** Diseñar estructuras con restricciones de materiales y seguridad

Recursos en Línea

Para aprender más sobre optimización:

- Coursera: Introduction to Optimization
- 3Blue1Brown: Visualización de problemas de optimización
- SciPy: Módulo de optimización
- R: Task View de optimización

3.2. Ejercicios Propuestos

Nota Importante

Practica formulación de restricciones y análisis de regiones factibles. **Soluciones completas en GitHub.**

3.2.1. Ejercicio 1: Gestión de Tiempo en Desarrollo

Desarrollador con 15 horas para front-end (x) y back-end (y):

- Mínimo 5 horas de front-end
- Total 15 horas

Tarea: Formula restricciones, dibuja región factible, encuentra 3 combinaciones posibles, analiza si mínimo fuera 8 horas.

Tip de Programación

x = horas front-end, y = horas back-end. Incluye que horas 0.

3.2.2. Ejercicio 2: Optimización de Servidores Cloud

Servidores: A (S/3 por hora) y B (S/5 por hora). Presupuesto: S/20 semanal. **Reto:** Escribe ecuación de costo, formula sistema, representa gráficamente, encuentra 2 combinaciones que agoten presupuesto.

Dato Curioso

Empresas ahorran 40 % en cloud con optimización similar.

3.2.3. Ejercicio 3: Balance en Gestión de Proyectos

Project Manager entre reuniones (x) y documentación (y):

- Reuniones 4 horas
- Documentación 6 horas
- Total 12 horas

Misión: Determina región factible, calcula vértices, propón 3 distribuciones, analiza si reuniones = 8 horas.

3.2.4. Ejercicio 4: Producción de Videojuegos

Producción de: Modelos 3D (2 horas c/u) y Texturas (3 horas c/u). Total 18 horas. **Desafío:** Escribe restricción, dibuja región factible, encuentra combinaciones que usen: todas 18h, solo 12h, al menos 5 assets total.

¡Atención! Error Común

x, y = cantidad de assets (no horas).

3.2.5. Ejercicio 5: Producción de Hardware

Startup con 50 unidades de componentes para:

- Dispositivo A: 5 unidades
- Dispositivo B: 10 unidades

Proyecto: Formula problema, resuelve gráficamente, explica 5 combinaciones posibles, analiza casos especiales.

Recursos en Línea

Soluciones en GitHub

github.com/LuisQuenaya

actividad_3.pdf

Dato Curioso

Estos tipos de problemas se usan en empresas reales:

- **Netflix**: Para optimizar el uso de servidores de streaming
- **Uber**: Para asignar conductores y calcular rutas
- **Amazon**: Para gestionar inventarios en sus almacenes
- **Tesla**: Para planificar la producción de vehículos

Capítulo 4

Métodos Numéricos para Encontrar Raíces

Introducción

Los métodos numéricos para la resolución de ecuaciones no lineales constituyen un pilar fundamental dentro de la programación numérica y el análisis computacional. Su objetivo principal es aproximar las raíces de funciones de la forma $f(x) = 0$, problema que aparece de manera recurrente en diversas áreas de la ciencia y la ingeniería.
En el ámbito de la **Ciencia de Datos** y la **Computación**, estos métodos resultan esenciales para:

- **Optimización** de modelos de Machine Learning
- **Calibración** de parámetros en simulaciones
- **Ajuste** de modelos estadísticos complejos
- **Resolución** de problemas sin solución analítica exacta

4.1. Método de Bisección

4.1.1. ¿Qué es el método de bisección?

Dato Curioso

¿Sabías que el método de bisección era usado por los babilonios hace más de 4000 años? Lo aplicaban para calcular raíces cuadradas en tablillas de arcilla con sorprendente precisión.

Imagina que tienes que encontrar una palabra específica en el diccionario. Sabes que está entre .ªmarilloz .ªzul", pero no sabes exactamente dónde. ¿Qué harías? Lo más lógisco sería abrir el diccionario justo en la mitad, ver si la palabra está antes o después, y repetir el proceso hasta encontrarla. ¡Eso es exactamente lo que hace el método de bisección con las ecuaciones matemáticas!

Algoritmo

Definición: El método de bisección es un algoritmo numérico simple pero poderoso para encontrar raíces (soluciones) de ecuaciones de la forma $f(x) = 0$. Funciona dividiendo repetidamente un intervalo a la mitad y seleccionando el subintervalo donde la función cambia de signo.

Condiciones para usarlo:

- La función $f(x)$ debe ser **continua** en el intervalo $[a, b]$
- Debe haber **cambio de signo**: $f(a) \cdot f(b) < 0$
- Solo se garantiza encontrar **una raíz** por intervalo

4.1.2. Formulas

Punto medio (en cada iteración):

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

Error máximo después de n iteraciones:

$$|r - c_n| \leq \frac{b_0 - a_0}{2^{n+1}}$$

Iteraciones necesarias para precisión ϵ:

$$n \geq \log_2 \left(\frac{b_0 - a_0}{\epsilon} \right)$$

Ejemplo práctico: Si tu intervalo inicial mide 10 unidades y quieres error menor a 0.001:

$$n \geq \log_2 \left(\frac{10}{0{,}001} \right) = \log_2(10000) \approx 13{,}29$$

Necesitas al menos 14 iteraciones.

Nota Importante

¿Por qué funciona? Gracias al **Teorema de Bolzano** (o Teorema del Valor Intermedio): Si una función continua cambia de signo en un intervalo, debe haber al menos una raíz en ese intervalo.

4.1.3. Cómo Funciona: Paso a Paso

Paso 1: Verificar condiciones iniciales:

$$f(a) \cdot f(b) < 0 \quad \text{(cambio de signo)}$$

Paso 2: Calcular punto medio:

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

Paso 3: Evaluar función en el punto medio:

$$f(c)$$

Paso 4: Decidir nuevo intervalo:

$$[a_{n+1}, b_{n+1}] = \begin{cases} [a_n, c_n] & \text{si } f(a_n) \cdot f(c_n) < 0 \\ [c_n, b_n] & \text{si } f(c_n) \cdot f(b_n) < 0 \end{cases}$$

Paso 5: Repetir hasta que el intervalo sea suficientemente pequeño:

$$|b_n - a_n| < \epsilon$$

4.1.4. Implementación en Python

Tip de Programación

Consejo para elegir intervalo inicial:

- **Grafica siempre la función** antes de empezar
- Busca puntos donde la función cambie de signo
- Elige intervalos pequeños (convergen más rápido)
- Verifica que $f(a) \cdot f(b) < 0$

```
def metodo_biseccion(f, a, b, tol=1e-6, max_iter=100):
"""
Encuentra raices de f(x)=0 usando el metodo de biseccion

Parametros:
-----------
f : funcion
Funcion continua f(x) cuya raiz buscamos
a, b : float
Extremos del intervalo [a, b] con f(a)*f(b)<0
tol : float
Tolerancia (error maximo aceptable)
max_iter : int
Maximo numero de iteraciones permitidas

Retorna:
--------
raiz : float
Aproximacion de la raiz
iteraciones : int
Numero de iteraciones realizadas
```

```
    historial : list
    Lista con informaci n de cada iteraci n
    """

    # Verificaci n inicial CRUCIAL
    if f(a) * f(b) > 0:
    print("ERROR: f(a) y f(b) deben tener signos opuestos")
    print(f"f({a}) = {f(a):.4f}, f({b}) = {f(b):.4f}")
    print("  Grafica  la funci n para elegir mejor intervalo!")
    return None, 0, []

    # Inicializaci n
    iteracion = 0
    historial = []

    print("="*60)
    print("M TODO DE BISECCI N - INICIO")
    print("="*60)
    print(f"{'Iter':>4} | {'a':>10} | {'b':>10} | {'c':>10} |
       {'f(c)':>12} | {'Error':>10}")
    print("-"*60)

    # Bucle principal
    while (b - a) / 2 > tol and iteracion < max_iter:
    c = (a + b) / 2           # Punto medio
    fc = f(c)                 # Evaluar funci n
    error = (b - a) / 2       # Error actual

    # Guardar informaci n
    historial.append({
        'iteracion': iteracion,
        'a': a, 'b': b, 'c': c,
        'f(c)': fc, 'error': error
    })

    # Mostrar progreso
    print(f"{iteracion:4d} | {a:10.6f} | {b:10.6f} | {c:10.6f}
       | {fc:12.6f} | {error:10.6f}")

    #   Encontramos   ra z exacta?
    if fc == 0:
    break

    # Decidir subintervalo
    if f(a) * fc < 0:
    b = c  # La ra z est  en [a, c]
    else:
    a = c  # La ra z est  en [c, b]

    iteracion += 1
```

```
    # Resultado final
    raiz = (a + b) / 2

    print("="*60)
    print(f"  Convergencia   alcanzada en {iteracion} iteraciones
       !")
    print(f"Ra z aproximada: {raiz:.8f}")
    print(f"Error final: {(b - a)/2:.2e}")
    print(f"f(ra z)     {f(raiz):.2e}")
    print("="*60)

    return raiz, iteracion, historial


    # ============================================
    # EJEMPLO 1: Encontrar  2  (soluci n de x  - 2 = 0)
    # ============================================
    print("\n" + "="*60)
    print("EJEMPLO 1: Encontrar  2  (x  - 2 = 0)")
    print("="*60)

    def f1(x):
    return x**2 - 2

    # Elegimos intervalo [1, 2] porque:
    # f(1) = -1 < 0
    # f(2) = 2 > 0
    #  Hay  cambio de signo!

    raiz1, iter1, hist1 = metodo_biseccion(f1, 1, 2, tol=1e-10)
    print(f"\ n 2      {raiz1:.10f}")
    print(f"Valor real: {2**0.5:.10f}")
    print(f"Diferencia: {abs(raiz1 - 2**0.5):.2e}")
```

Listing 4.1: Implementación completa del método de bisección

4.1.5. Ejemplo Práctico: Resolver $\cos(x) = x$

Ejemplo 4.1 (La constante de Dottie)**.** Vamos a encontrar la solución de $\cos(x) = x$, conocida como la **constante de Dottie**. Esta constante aparece en muchos contextos matemáticos y físicos.

```
    # ============================================
    # EJEMPLO 2: Resolver cos(x) = x
    # ============================================
    import math

    print("\n" + "="*60)
    print("EJEMPLO 2: Resolver cos(x) = x (Constante de Dottie)
       ")
    print("="*60)
```

```python

    def f2(x):
    return math.cos(x) - x

    # Observamos:
    # f(0) = cos(0) - 0 = 1 > 0
    # f(1) = cos(1) - 1      0.5403 - 1 = -0.4597 < 0
    #   Cambio   de signo en [0, 1]!

    raiz2, iter2, hist2 = metodo_biseccion(f2, 0, 1, tol=1e-8)

    print(f"\nSoluci n de cos(x) = x (Constante de Dottie):")
    print(f"x     {raiz2:.10f}")
    print(f"Verificaci n: cos({raiz2:.6f}) = {math.cos(raiz2)
       :.6f}")
    print(f"Diferencia: {abs(raiz2 - math.cos(raiz2)):.2e}")
```

Listing 4.2: Ejemplo: Encontrar la constante de Dottie

Tabla de Evolución del programa

Iteración	a	b	c	Error máximo
0	0.0000	1.0000	0.5000	0.5000
1	0.5000	1.0000	0.7500	0.2500
2	0.5000	0.7500	0.6250	0.1250
3	0.6250	0.7500	0.6875	0.0625
4	0.6875	0.7500	0.7188	0.0312
5	0.7188	0.7500	0.7344	0.0156
6	0.7344	0.7500	0.7422	0.0078
⋮	⋮	⋮	⋮	⋮
29	0.739085	0.739085	0.739085	0.000000

Resultado final: $x \approx 0{,}7390851332$ (la constante de Dottie)

4.1.6. Ventajas y Limitaciones

¡Atención! Error Común

¡Cuidado con estos errores comunes!

1. **No verificar cambio de signo** → El método falla silenciosamente
2. **Tolerancia demasiado pequeña** → Iteraciones innecesarias
3. **Sin límite de iteraciones** → Bucle infinito posible
4. **Intervalo mal elegido** → Puede no contener raíz

Nota Importante

Comparación con otros métodos:

- **Más lento** que Newton-Raphson o la Secante
- **Más seguro** que métodos que requieren derivadas
- **Más simple** que métodos más sofisticados

- **Ventajas (lo bueno):**
  - **Siempre converge** si hay cambio de signo
  - **Muy robusto** - funciona con funciones "problemáticas"
  - **Fácil de entender** - concepto intuitivo
  - **Error controlable** - sabemos el error máximo en cada paso
  - **Simple implementación** - pocas líneas de código
  - **No requiere derivadas** - solo evaluar la función
- **Limitaciones (lo malo):**
  - **Convergencia lenta** - solo reduce el error a la mitad cada vez
  - **Requiere cambio de signo** - no funciona con raíces dobles
  - **Solo una raíz por intervalo** - puede pasar por alto otras
  - **No usa información de la función** - solo mira signos
  - **Puede ser ineficiente** para funciones "suavesçon métodos mejores

4.1.7. Implementación Avanzada y Visualización

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

    def biseccion_con_grafica(f, a, b, tol=1e-6, max_iter=50):
    """
    Versi n mejorada con visualizaci n gr fica
    """
    # Verificaci n inicial
    if f(a) * f(b) > 0:
    raise ValueError("No hay cambio de signo en el intervalo")

    # Preparar visualizaci n
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

    # Graficar funci n
    x_vals = np.linspace(a - 0.5, b + 0.5, 400)
    y_vals = f(x_vals)
    ax1.plot(x_vals, y_vals, 'b-', linewidth=2, label='f(x)')
    ax1.axhline(y=0, color='k', linestyle='-', alpha=0.3)
```

```
    ax1.set_xlabel('x')
    ax1.set_ylabel('f(x)')
    ax1.set_title('M todo de Bisecci n - Progreso')
    ax1.grid(True, alpha=0.3)
    ax1.legend()

    # Listas para almacenar progreso
    errores = []
    iteraciones = []

    # Bisecci n
    for i in range(max_iter):
    c = (a + b) / 2
    fc = f(c)

    # Marcar punto medio en gr fico
    ax1.plot(c, fc, 'ro', markersize=8)
    ax1.plot([a, b], [0, 0], 'g-', linewidth=2, alpha=0.5)

    # Guardar error
    error = (b - a) / 2
    errores.append(error)
    iteraciones.append(i)

    # Verificar convergencia
    if abs(fc) < tol or error < tol:
    break

    # Actualizar intervalo
    if f(a) * fc < 0:
    b = c
    else:
    a = c

    # Graficar convergencia del error
    ax2.semilogy(iteraciones, errores, 'bo-', linewidth=2,
       markersize=6)
    ax2.axhline(y=tol, color='r', linestyle='--', label=f'
       Tolerancia: {tol}')
    ax2.set_xlabel('Iteraci n')
    ax2.set_ylabel('Error m ximo')
    ax2.set_title('Convergencia del Error')
    ax2.grid(True, alpha=0.3)
    ax2.legend()

    plt.tight_layout()
    plt.show()

    return (a + b) / 2, i + 1

    # Ejemplo de uso
```

```
69      def ejemplo_visual():
70      def funcion(x):
71      return np.exp(-x) - np.cos(x)
72
73      raiz, iteraciones = biseccion_con_grafica(funcion, 0, 2,
            tol=1e-8)
74      print(f"Ra z encontrada: {raiz:.10f}")
75      print(f"Iteraciones: {iteraciones}")
76
77      if __name__ == "__main__":
78      ejemplo_visual()
```

Listing 4.3: Versión avanzada con visualización

4.1.8. Resumen y Recomendaciones

Nota Importante

¿Cuándo usar bisección?

- **Úsalo cuando:**
  - Tu función es continua pero no conoces su derivada
  - Priorizas seguridad sobre velocidad
  - Quieres un método fácil de implementar
  - Tienes un intervalo con cambio de signo
- **No lo uses cuando:**
  - Necesitas máxima velocidad de convergencia
  - No hay cambio de signo en el intervalo
  - Buscas múltiples raíces simultáneamente
  - La función tiene discontinuidades

Recursos en Línea

Recursos para aprender más:

- Wikipedia: Método de Bisección
- Video: Visualización animada del método

Tip de Programación

Consejo final: El método de bisección es como el "martillo"de los métodos numéricos: no es la herramienta más sofisticada, pero casi siempre funciona y rara vez falla. ¡Aprende a usarlo bien y siempre tendrás un método confiable para encontrar raíces!

En pocas palabras: La bisección es tu mejor aliado cuando necesitas encontrar raíces

de manera segura y predecible. Es lento pero seguro, simple pero poderoso. En el mundo de los métodos numéricos, a veces lo simple es lo más efectivo.

4.2. Método de la Secante

4.2.1. ¿Qué es el método de la Secante?

Definición 4.1 (Método de la Secante)**.** Es un algoritmo para encontrar raíces de $f(x) = 0$ que usa una línea recta (secante) entre dos puntos para aproximar la raíz. No necesita la derivada de la función, solo evalúa $f(x)$ en dos puntos cercanos.

Dato Curioso

El método de la secante aparece en textos matemáticos chinos del siglo I d.C., pero se popularizó en occidente mucho después. Es como el "primo práctico"de Newton-Raphson: casi igual de rápido, pero sin el dolor de cabeza de calcular derivadas.

4.2.2. La Idea Simple

Imagina que conoces dos casas en la ciudad: en una casa el GPS dice .estás a 5 km del destino", en otra dice .estás a 2 km". Trazas una línea entre ellas y extrapolas dónde debería estar el destino (donde la distancia sería 0 km). Eso es la secante: dos puntos, una línea, y una estimación inteligente.

4.2.3. Fórmula Mágica

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

En palabras simples:

- $(x_{n-1}, f(x_{n-1}))$: Primer punto conocido
- $(x_n, f(x_n))$: Segundo punto conocido
- La fracción: Pendiente aproximada entre los puntos
- Restamos: Movemos desde x_n hacia el eje X

4.2.4. Ejemplo Paso a Paso

Problema: Resolver $f(x) = x^2 - 3 = 0$ (encontrar $\sqrt{3}$)
Paso 1: Elegimos dos puntos cercanos: $x_0 = 1$, $x_1 = 2$

$$f(1) = 1^2 - 3 = -2, \quad f(2) = 2^2 - 3 = 1$$

Paso 2: Aplicamos la fórmula:

$$x_2 = 2 - 1 \cdot \frac{2-1}{1-(-2)} = 2 - 1 \cdot \frac{1}{3} = 2 - 0{,}3333 = 1{,}6667$$

Paso 3: Repetimos con $x_1 = 2$, $x_2 = 1{,}6667$:

$$f(1{,}6667) = (1{,}6667)^2 - 3 = 2{,}7778 - 3 = -0{,}2222$$

$$x_3 = 1{,}6667 - (-0{,}2222) \cdot \frac{1{,}6667 - 2}{-0{,}2222 - 1} = 1{,}6667 + 0{,}2222 \cdot \frac{-0{,}3333}{-1{,}2222}$$

$$x_3 = 1{,}6667 + 0{,}2222 \cdot 0{,}2727 = 1{,}6667 + 0{,}0606 = 1{,}7273$$

Paso 4: Continuamos hasta precisión deseada:

$$x_4 = 1{,}7321 \quad \text{(¡Casi exacto!)}$$

Iteración	x_n	$f(x_n)$	**Error**	**Dígitos correctos**
0	1.0000	-2.0000	0.7321	0
1	2.0000	1.0000	0.2679	0
2	1.6667	-0.2222	0.0654	1
3	1.7273	-0.0165	0.0048	2
4	1.7321	-0.0001	0.0000	5

¡5 dígitos correctos en solo 4 iteraciones!

4.2.5. Código Python

```
    def secante_simple(f, x0, x1, tol=1e-6, max_iter=20):
    """
    M todo de la secante: f cil y sin derivadas

    f: funci n f(x) que queremos igualar a cero
    x0, x1: dos puntos iniciales (cercanos a la ra z)
    tol: precisi n deseada
    max_iter: m ximo de iteraciones
    """

    print("=" * 60)
    print("M TODO DE LA SECANTE")
    print("=" * 60)
    print(f"Buscando ra z de f(x) = 0")
    print(f"Puntos iniciales: x0 = {x0}, x1 = {x1}")
    print(f"f(x0) = {f(x0):.4f}, f(x1) = {f(x1):.4f}")
    print("-" * 60)

    # Evaluar en los puntos iniciales
    fx0 = f(x0)
    fx1 = f(x1)

    print(f"Iter 0: x = {x0:.8f}, f(x) = {fx0:.8f}")
    print(f"Iter 1: x = {x1:.8f}, f(x) = {fx1:.8f}")

    # Bucle principal
    for i in range(2, max_iter + 2):
    #   Cuidado   con divisi n por cero!
```

```
    if abs(fx1 - fx0) < 1e-15:
    print(" Error ! f(x1) casi igual a f(x0)")
    print("Prueba con otros puntos iniciales")
    return None, i-1

    # La f rmula m gica
    x2 = x1 - fx1 * (x1 - x0) / (fx1 - fx0)
    fx2 = f(x2)

    print(f"Iter {i}: x = {x2:.8f}, f(x) = {fx2:.8f}")

    # Verificar convergencia
    if abs(x2 - x1) < tol:
    print("-" * 60)
    print(f"      CONVERGENCIA   en {i} iteraciones!")
    print(f"    Ra z aproximada: {x2:.10f}")
    print(f"    f(ra z) = {fx2:.2e}     0")
    print("=" * 60)
    return x2, i

    # Preparar siguiente iteraci n
    x0, fx0 = x1, fx1
    x1, fx1 = x2, fx2

    # Si llegamos aqu , no convergi
    print("-" * 60)
    print(f"    No convergi  en {max_iter} iteraciones")
    print(f"     ltimo  valor: {x1:.8f}")
    print("=" * 60)
    return x1, max_iter


    # ============================================
    # EJEMPLO 1: Encontrar  3
    # ============================================
    print("\nEJEMPLO 1: Calcular  3 ")
    print("Resolviendo x  - 3 = 0")

    import math

    f1 = lambda x: x**2 - 3
    raiz1, iter1 = secante_simple(f1, x0=1.0, x1=2.0, tol=1e-8)

    if raiz1 is not None:
    print(f"\ n 3      {raiz1:.8f}")
    print(f"Valor exacto: {math.sqrt(3):.8f}")
    print(f"Diferencia: {abs(raiz1 - math.sqrt(3)):.2e}")
    print(f"Iteraciones: {iter1}")

    # ============================================
    # EJEMPLO 2: Resolver ecuaci n trigonom trica
```

```
    # ============================================
    print("\n\nEJEMPLO 2: Resolver sen(x) = x/2")
    print("Encontrar intersecci n entre seno y l nea")

    f2 = lambda x: math.sin(x) - x/2
    raiz2, iter2 = secante_simple(f2, x0=1.0, x1=1.5, tol=1e-8)

    if raiz2 is not None:
    print(f"\nSoluci n: x     {raiz2:.8f}")
    print(f"Verificaci n: sen({raiz2:.6f}) = {math.sin(raiz2)
       :.6f}")
    print(f"           {raiz2:.6f}/2 = {raiz2/2:.6f}")
    print(f"Diferencia: {abs(math.sin(raiz2) - raiz2/2):.2e}")

    # ============================================
    # EJEMPLO 3: MALA ELECCI N de puntos iniciales
    # ============================================
    print("\n\nEJEMPLO 3:   Qu   pasa con malos puntos
       iniciales?")

    # Mala elecci n: puntos lejanos y misma direcci n
    print("\nCaso 1: Puntos muy lejanos (x0=10, x1=20):")
    raiz3a, iter3a = secante_simple(f1, x0=10.0, x1=20.0,
       max_iter=10)

    print("\nCaso 2: Puntos del mismo lado (x0=1.5, x1=1.6):")
    raiz3b, iter3b = secante_simple(f1, x0=1.5, x1=1.6,
       max_iter=10)

    print("\nLecci n: Los puntos deben estar en lados OPUESTOS
        de la ra z")
    print("         y relativamente cercanos")
```

Listing 4.4: Método de la secante básico

4.2.6. Comparación con Otros Métodos

Método	Velocidad	Facilidad	¿Derivada?
Bisección	Lento (lineal)		No
Newton	Muy rápido (cuadrática)		Sí
Secante	Rápido (superlineal)		No

</> Tip de Programación

Convergencia superlineal: La secante tiene orden $p \approx 1{,}618$ (el número áureo). Esto significa que cada nuevo error es aproximadamente el anterior elevado a 1.618. No tan rápido como Newton (error^2), pero mucho más rápido que bisección (error/2).

4.2.7. Reglas para Éxito

Algoritmo

Guía para usar bien la secante:

1. **Elige puntos opuestos**: $f(x_0)$ y $f(x_1)$ deben tener signos diferentes
2. **Cerca pero no iguales**: Puntos cercanos pero no idénticos
3. **Evita pendiente cero**: $|f(x_1) - f(x_0)|$ no debe ser casi cero
4. **Pon límites**: Siempre máximo de iteraciones
5. **Verifica**: $|x_{n+1} - x_n|$ pequeño al final

4.2.8. Ventajas y Desventajas

- **Ventajas:**
  - **Sin derivadas**: No necesitas calcular $f'(x)$
  - **Rápido**: Casi tan rápido como Newton
  - **Simple**: Fórmula fácil de implementar
  - **Robusto**: Funciona bien en muchos casos
- **Desventajas:**
  - **Dos puntos iniciales**: Necesitas elegir dos valores
  - **Puede diverger**: Si puntos iniciales son malos
  - **Menos estable**: Que bisección
  - **Error difícil de estimar**: No como bisección

4.2.9. Versión Mejorada con Manejo de Errores

```
def secante_robusta(f, x0, x1, tol=1e-8, max_iter=30):
"""
M todo de la secante con mejor manejo de errores
"""
# Verificaciones iniciales
fx0 = f(x0)
fx1 = f(x1)

print(f"Inicio: x0={x0:.4f} (f={fx0:.4f}), x1={x1:.4f} (f={
    fx1:.4f})")

# Verificar que no sean iguales
if abs(fx1 - fx0) < 1e-15:
print(" Los  puntos iniciales son casi iguales!")
print("Prueba con puntos  m  s  diferentes")
return None
```

```
16
17      # Verificar cambio de signo (opcional pero recomendado)
18      if fx0 * fx1 > 0:
19      print("    Advertencia: f(x0) y f(x1) tienen mismo signo")
20      print("   Puede no converger o converger m s lento")
21
22      # Bucle principal con mejores mensajes
23      for i in range(max_iter):
24      # Calcular nuevo punto
25      x2 = x1 - fx1 * (x1 - x0) / (fx1 - fx0)
26      fx2 = f(x2)
27
28      error = abs(x2 - x1)
29
30      print(f"  Iter {i+1}: x={x2:.10f}, f(x)={fx2:.2e}, error={
           error:.2e}")
31
32      # Criterios de parada
33      if error < tol:
34      print(f"    Convergencia en {i+1} iteraciones")
35      return x2
36      if abs(fx2) < tol:
37      print(f"    f(x)     0 en {i+1} iteraciones")
38      return x2
39
40      # Actualizar para siguiente iteraci n
41      x0, fx0 = x1, fx1
42      x1, fx1 = x2, fx2
43
44      print(f"    No convergi  en {max_iter} iteraciones")
45      return x1
46
47      # Ejemplo pr ctico
48      print("Encontrando ra z  c bica de 5")
49      print("(Resolver x   - 5 = 0)")
50
51      f_cubica = lambda x: x**3 - 5
52      raiz = secante_robusta(f_cubica, x0=1.0, x1=2.0, tol=1e-10)
53
54      if raiz is not None:
55      print(f"\ n 5      {raiz:.10f}")
56      print(f"Verificaci n: {raiz:.6f}   = {raiz**3:.6f}")
```

Listing 4.5: Secante robusta

4.2.10. Aplicaciones Prácticas

Ejemplo 4.2 (Ingeniería Eléctrica)**.** Para calcular la frecuencia de resonancia en un circuito RLC, resolvemos:

$$\omega L - \frac{1}{\omega C} = 0$$

La secante es perfecta porque la derivada es complicada.

Ejemplo 4.3 (Economía)**.** Encontrar la Tasa Interna de Retorno (TIR) de una inversión:

$$\sum_{t=0}^{n} \frac{CF_t}{(1+r)^t} = 0$$

Donde r es la tasa desconocida. ¡Secante al rescate!

Ejemplo 4.4 (Machine Learning)**.** En ajuste de modelos, a veces necesitamos resolver:

$$\nabla L(\theta) = 0$$

Cuando el gradiente es costoso de derivar, usamos métodos tipo secante.

4.2.11. Errores Comunes y Soluciones

¡Atención! Error Común

Problemas típicos:
1. **División por cero:** $f(x_1) \approx f(x_0)$ **Solución:** Verifica diferencia ¿ϵ
2. **Puntos del mismo lado:** $f(x_0)$ y $f(x_1)$ mismo signo **Solución:** Elige puntos con signos opuestos
3. **Puntos muy lejanos:** Convergencia lenta o divergencia **Solución:** Usa bisección primero para acercar
4. **Máximo de iteraciones:** No converge **Solución:** Aumenta iteraciones o cambia puntos

4.2.12. Consejos Prácticos

Tip de Programación

Para usar secante como profesional:

- **Empieza con bisección**: 3-4 iteraciones para acercarte
- **Luego usa secante**: Para refinamiento rápido
- **Monitorea el error**: Si aumenta, algo está mal
- **Prueba varios puntos**: Si no converge, cambia x_0 o x_1
- **Combina métodos**: Secante para velocidad, bisección para seguridad

Nota Importante

¿Cuándo elegir secante?

- **SÍ**: Cuando no tienes o no quieres calcular derivada
- **SÍ**: Para funciones cuya derivada es complicada
- **SÍ**: Cuando quieres más velocidad que bisección
- **NO**: Si necesitas máxima velocidad posible (usa Newton)
- **NO**: Si priorizas seguridad absoluta (usa bisección)

4.2.13. Resumen Final

En pocas palabras: El método de la secante es el "punto medio perfecto.^en^tre bisección y Newton. No es tan lento como el primero, no necesita derivada como el segundo. Es como tener un coche deportivo que no necesita gasolina premium: rápido, eficiente y práctico para el día a día.

¡Aprende a usarlo bien y tendrás una herramienta valiosa para resolver problemas reales sin complicaciones matemáticas innecesarias!

4.3. Método de Newton-Raphson:

4.3.1. ¿Qué es el método de Newton-Raphson?

Definición 4.2 (Método de Newton-Raphson)**.** Es un algoritmo numérico para encontrar raíces de ecuaciones de la forma $f(x) = 0$. Usa la derivada de la función como "brújula"para acercarse rápidamente a la solución. Es como un GPS matemático: cada iteración te dice exactamente hacia dónde moverte para llegar más rápido a tu destino.

Dato Curioso

Isaac Newton desarrolló este método en 1669 para calcular órbitas planetarias, pero lo mantuvo en secreto como ventaja personal. Joseph Raphson lo publicó en 1690 y lo simplificó. Curiosamente, ni Newton ni Raphson lo usaron para lo que hoy es más común: ¡tu calculadora de bolsillo lo usa cada vez que presionas la tecla de raíz cuadrada!

4.3.2. Fórmula Fundamental

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

Interpretación:

- x_n: Tu posición actual
- $f(x_n)$: Qué tan lejos estás del cero
- $f'(x_n)$: La pendiente (qué tan rápido cambia)
- x_{n+1}: Tu nueva y mejor posición

4.3.3. ¿Por qué es tan importante?

Nota Importante

Newton-Raphson revolucionó la computación numérica porque:

- **Velocidad extrema**: Convergencia cuadrática (duplica dígitos correctos cada paso)
- **Precisión quirúrgica**: Pocas iteraciones para alta precisión
- **Versatilidad**: Funciona en física, ingeniería, finanzas, IA...
- **Fundamental**: Base de algoritmos modernos como backpropagation en redes neuronales

4.3.4. Características Clave

Tip de Programación

Lo que AMAMOS de Newton-Raphson:

- **Super rápido**: 3-5 iteraciones suelen bastar
- **Muy preciso**: Errores que se hacen cuadrados
- **Inteligente**: Usa información de la función (derivada)
- **Versátil**: Aplicable en múltiples dimensiones

Lo que NOS PREOCUPA:

- **Necesita derivada**: A veces difícil de calcular
- **Sensible al inicio**: Punto inicial crucial
- **Puede fallar**: Si $f'(x) \approx 0$, se bloquea
- **No siempre converge**: Para funciones "salvajes"

4.3.5. Condiciones para que funcione

Algoritmo

Requisitos para éxito garantizado:

1. $f(x)$ debe ser dos veces diferenciable cerca de la raíz
2. $f'(x) \neq 0$ en la vecindad de la raíz
3. Punto inicial x_0 "suficientemente cerca"de la solución
4. La segunda derivada $f''(x)$ debe ser continua

4.3.6. Ejemplo Visual: Encontrando $\sqrt{10}$

Imagina que quieres calcular $\sqrt{10}$ a mano. Con Newton-Raphson es sorprendentemente fácil:

$$f(x) = x^2 - 10 \quad \text{(queremos } x^2 = 10\text{)}$$

$$f'(x) = 2x$$

Empezamos en $x_0 = 3$:

Paso	Cálculo	Resultado	Error
1	$3 - \frac{9-10}{6}$	3,166667	0,00439
2	$3{,}166667 - \frac{10{,}0278-10}{6{,}3333}$	3,162280	0,000003
3	$3{,}162280 - \frac{10{,}000024-10}{6{,}3246}$	3,162277	$< 10^{-12}$

Nota Importante

¡En solo 3 pasos tenemos 12 decimales correctos! La bisección necesitaría unos 40 pasos para la misma precisión.

4.3.7. Aplicaciones en el Mundo Real

Ejemplo 4.5 (Calculadoras y procesadores)**.** Cada vez que presionas $\sqrt{x}$ en tu calculadora, ejecuta un algoritmo basado en Newton-Raphson. Los chips Intel y AMD lo implementan en hardware para máxima velocidad.

Ejemplo 4.6 (Sistemas GPS)**.** Para calcular tu posición exacta, los satélites resuelven sistemas de ecuaciones usando métodos tipo Newton-Raphson multidimensional.

Ejemplo 4.7 (Finanzas de alto nivel)**.** Los bancos calculan tasas de interés implícitas y precios de opciones con variantes de este método. Black-Scholes, el modelo Nobel de economía, lo usa extensivamente.

Ejemplo 4.8 (Redes Neuronales Artificiales)**.** El algoritmo de backpropagation que entrena ChatGPT y otras IA es esencialmente Newton-Raphson en espacios de millones de dimensiones.

4.3.8. Código Python

4.3.9. Errores Comunes y Cómo Evitarlos

¡Atención! Error Común

5 ERRORES que arruinan Newton-Raphson:

1. **Derivada cero**: Si $f'(x) \approx 0$, división por (casi) cero **Solución**: Verifica $|f'(x_0)| > \epsilon$ antes de empezar
2. **Punto inicial lejano**: Puede diverger o converger a otra raíz **Solución**: Usa bisección primero para acercarte
3. **Función no diferenciable**: Picos, saltos, esquinas **Solución**: Verifica suavidad de la función
4. **Sin límite de iteraciones**: Bucle infinito posible **Solución**: SIEMPRE pon un máximo de iteraciones
5. **Raíces múltiples**: Convergencia más lenta **Solución**: Usa método de Newton modificado

4.3.10. Variantes y Mejoras

Nota Importante

Cuando el Newton básico no basta:

- **Newton Modificado**: Para raíces múltiples

$$x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)} \quad (m = \text{multiplicidad})$$

- **Newton con Relajación**: Para mejorar convergencia

$$x_{n+1} = x_n - \omega\frac{f(x_n)}{f'(x_n)} \quad (0 < \omega < 2)$$

- **Newton en Múltiples Dimensiones**: Para sistemas de ecuaciones

$$\mathbf{x}_{n+1} = \mathbf{x}_n - J^{-1}(\mathbf{x}_n)F(\mathbf{x}_n)$$

  donde J es la matriz jacobiana

- **Newton con Aproximación de Derivada**: Cuando $f'(x)$ es costosa

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

4.3.11. Comparación con Otros Métodos

Método	Velocidad	Facilidad	Robustez	¿Derivada?
Bisección	1x (Lineal)			No
Newton	100x+ (Cuadrática)			Sí
Secante	10x (Superlineal)			No
Punto Fijo	Variable			No

Tip de Programación

¿Cuándo elegir Newton-Raphson?

- **SÍ**: Cuando tienes la derivada y quieres máxima velocidad
- **SÍ**: Para funciones suaves y bien comportadas
- **SÍ**: Cuando necesitas alta precisión rápidamente
- **NO**: Si calcular la derivada es muy costoso
- **NO**: Si la función tiene discontinuidades
- **NO**: Si no tienes buen punto inicial

Algoritmo

Receta para Newton-Raphson exitoso:

1. **Grafica primero**: Visualiza la función y sus raíces
2. **Elige sabiamente**: Punto inicial cerca de la raíz
3. **Verifica derivada**: Asegura $|f'(x)| > \epsilon$
4. **Pon límites**: Máximo de iteraciones y tolerancia
5. **Maneja errores**: División por cero, no convergencia
6. **Monitorea**: Imprime progreso para debuggear
7. **Valida**: Verifica resultado sustituyendo

4.3.12. El Futuro de Newton-Raphson

Dato Curioso

Hoy, Newton-Raphson vive en:

- **Chips NVIDIA**: Para gráficos y cálculos científicos
- **Autopilotos Tesla**: Para navegación y control
- **Simuladores cuánticos**: Para resolver ecuaciones de Schrödinger
- **Algoritmos de trading**: Para optimización de portafolios

Su principio básico sigue siendo relevante 350 años después.

Recursos en Línea

Para aprender más:

- Visualización animada de Newton-Raphson
- Wikipedia: Teoría completa
- SciPy: Implementación optimizada
- Mi repositorio: Ejemplos prácticos

En esencia: Newton-Raphson es el Ferrari de los métodos numéricos: increíblemente rápido cuando las condiciones son perfectas, pero necesita mantenimiento cuidadoso y un buen conductor. Domínalo, y tendrás una herramienta poderosa para toda tu carrera en ciencias e ingeniería.

4.4. Método de Punto Fijo

4.4.1. ¿Qué es el método de Punto Fijo?

Definición 4.3 (Método de Punto Fijo)**.** Es una técnica para resolver ecuaciones de la forma $x = g(x)$. La idea es transformar el problema $f(x) = 0$ en una ecuación equivalente $x = g(x)$, y luego generar una sucesión iterativa: $x_{n+1} = g(x_n)$. Si converge, el límite es un "punto fijo"de g (un valor que no cambia al aplicarle g).

Dato Curioso

El concepto de punto fijo es tan antiguo como las matemáticas mismas. Los griegos ya lo usaban implícitamente. Pero fue Stefan Banach quien en 1922 formalizó el teorema que garantiza su convergencia, conocido hoy como el "Teorema del Punto Fijo de Banach.º "Teorema de la Contracción".

4.4.2. Fórmula Fundamental

$$x_{n+1} = g(x_n)$$

Condición de convergencia (Teorema de Banach): Si g es una contracción en un intervalo $[a, b]$, es decir:

$$|g'(x)| \leq L < 1 \quad \text{para todo } x \in [a, b]$$

entonces existe un único punto fijo x^* y la sucesión converge a él.

4.4.3. Ejemplo Básico: Resolver $x^3 - x - 1 = 0$

Problema: Queremos resolver $f(x) = x^3 - x - 1 = 0$
Paso 1: Transformar a forma $x = g(x)$ Podemos despejar x de diferentes maneras:

- Opción A: $x = x^3 - 1$ (mala: diverge)
- Opción B: $x = \sqrt[3]{x + 1}$ (buena: $g'(x) < 1$)
- Opción C: $x = \frac{1}{x^2 - 1}$ (peligrosa: cerca de $x = 1$)

Tip de Programación

Truco para elegir buena $g(x)$:

1. Despeja x de la manera más simple
2. Calcula $|g'(x)|$ cerca de la raíz
3. Si $|g'(x)| < 1$, debería funcionar
4. Si $|g'(x)| > 1$, busca otra forma

Elegimos: $g(x) = \sqrt[3]{x+1}$
Verificamos: $g'(x) = \frac{1}{3}(x+1)^{-2/3}$
Si $x \approx 1{,}3$ (la raíz real), $|g'(1{,}3)| \approx 0{,}2 < 1$
Empezamos con $x_0 = 1{,}0$:

$$\begin{aligned}
x_1 &= \sqrt[3]{1{,}0+1} = \sqrt[3]{2} = 1{,}259921 \\
x_2 &= \sqrt[3]{1{,}259921+1} = \sqrt[3]{2{,}259921} = 1{,}312294 \\
x_3 &= \sqrt[3]{1{,}312294+1} = \sqrt[3]{2{,}312294} = 1{,}322354 \\
x_4 &= \sqrt[3]{1{,}322354+1} = \sqrt[3]{2{,}322354} = 1{,}324269 \\
x_5 &= \sqrt[3]{1{,}324269+1} = \sqrt[3]{2{,}324269} = 1{,}324633 \\
x_6 &= \sqrt[3]{1{,}324633+1} = \sqrt[3]{2{,}324633} = 1{,}324707 \\
x_7 &= \sqrt[3]{1{,}324707+1} = \sqrt[3]{2{,}324707} = 1{,}324717
\end{aligned}$$

¡En 7 pasos tenemos 6 decimales correctos!

4.4.4. Características del Método

- **Ventajas:**
  - **Sencillo**: Solo evaluar $g(x)$ repetidamente
  - **No necesita derivada** (excepto para verificar convergencia)
  - **Conceptualmente simple**: Fácil de entender
  - **Flexible**: Muchas formas de escribir $g(x)$
- **Desventajas:**
  - **No siempre converge**: Depende de $g(x)$
  - **Velocidad variable**: Puede ser lento si $|g'(x)| \approx 1$
  - **Elección crítica**: $g(x)$ incorrecta = fracaso
  - **Menos preciso**: Que Newton-Raphson para misma $g(x)$

4.4.5. Código Python

```
        def punto_fijo_simple(g, x0, max_iter=50, tol=1e-6):
        """
        M todo de punto fijo b sico

        g: funci n de iteraci n (x = g(x))
        x0: valor inicial
        max_iter: m ximo de iteraciones
        tol: tolerancia (precisi n)
        """

        print("=" * 60)
        print("M TODO DE PUNTO FIJO")
        print("=" * 60)
        print(f"Funci n de iteraci n: x = g(x)")
        print(f"Punto inicial: x0 = {x0}")
        print(f"Tolerancia: {tol}")
```

```
        print("-" * 60)

        x = x0
        print(f"Iter 0: x = {x:.8f}")

        for i in range(1, max_iter + 1):
        x_nuevo = g(x)

        # Mostrar progreso
        print(f"Iter {i}: x = {x_nuevo:.8f}", end="")

        # Calcular cambio
        cambio = abs(x_nuevo - x)
        print(f" | Cambio: {cambio:.2e}")

        # Verificar convergencia
        if cambio < tol:
        print("-" * 60)
        print(f"      CONVERGENCIA   en {i} iteraciones!")
        print(f"    Punto fijo encontrado: {x_nuevo:.10f}")
        print(f"    Verificaci n: g({x_nuevo:.6f}) = {g(
           x_nuevo):.6f}")
        print("=" * 60)
        return x_nuevo, i

        # Actualizar para siguiente iteraci n
        x = x_nuevo

        # Si llegamos aqu , no convergi
        print("-" * 60)
        print(f"    NO CONVERGI  en {max_iter} iteraciones")
        print(f"     ltimo  valor: {x:.8f}")
        print("=" * 60)
        return x, max_iter


        # ============================================
        # EJEMPLO 1: Resolver x   - x - 1 = 0
        # ============================================
        print("\nEJEMPLO 1: Resolver x   - x - 1 = 0")
        print("Forma equivalente: x =    (x + 1)")

        import math

        # Definir funci n de iteraci n
        g1 = lambda x: (x + 1) ** (1/3)  #    (x + 1)

        # Ejecutar m todo
        solucion1, iter1 = punto_fijo_simple(g1, x0=1.0, tol=1e
           -8)
```

```
66          print(f"\nSoluci n aproximada: {solucion1:.8f}")
67          print(f"Verificaci n: {solucion1**3 - solucion1 - 1:.2
               e}     0")
68
69
70          print("Iteraci n | Valor x | g(x) | Error")
71          print("-" * 40)
72          print(f"{0:9d} | {x:7.4f} | {g(x):6.4f} | ---")
73
74          # Bucle principal
75          for i in range(1, max_iter + 1):
76          x_nuevo = g(x)
77          error = abs(x_nuevo - x)
78
79          historial_x.append(x_nuevo)
80          historial_error.append(error)
81
82          print(f"{i:9d} | {x_nuevo:7.4f} | {g(x_nuevo):6.4f} | {
               error:6.4f}")
83
84          # Dibujar l neas en primer gr fico
85          # L nea vertical
86          ax1.plot([x, x], [x, x_nuevo], 'g--', alpha=0.5)
87          # L nea horizontal
88          ax1.plot([x, x_nuevo], [x_nuevo, x_nuevo], 'g--', alpha
               =0.5)
89          # Puntos
90          ax1.plot(x, x, 'go', markersize=6)
91          ax1.plot(x_nuevo, x_nuevo, 'go', markersize=6)
```

Listing 4.6: Método de Punto Fijo básico

Algoritmo

Guía práctica para transformar $f(x) = 0$ a $x = g(x)$:

1. **Aislar x:** Despeja x de manera simple
2. **Verificar derivada:** Calcula $|g'(x)|$ cerca de la raíz esperada
3. **Si $|g'(x)| < 1$:** Buena elección, debería converger
4. **Si $|g'(x)| > 1$:** Mala elección, busca otra forma
5. **Probar varias:** A veces hay múltiples $g(x)$ posibles

Ejemplos comunes:

- De $f(x) = 0$, saca $x = x - f(x)$ (no suele funcionar)
- Mejor: $x = x - \omega f(x)$ con ω pequeño (relajación)
- O: $x = x + f(x)/M$ con M grande

4.4.6. Errores Comunes y Soluciones

¡Atención! Error Común

Problemas típicos y cómo solucionarlos:
1. **Divergencia:** $|g'(x)| > 1$ **Solución:** Busca otra $g(x)$ o usa método de relajación
2. **Ciclos infinitos:** $g(x)$ oscila entre valores **Solución:** Usa promedio ponderado: $x_{n+1} = \alpha g(x_n) + (1-\alpha)x_n$
3. **Convergencia lenta:** $|g'(x)| \approx 1$ **Solución:** Acelera con método de Aitken
4. **Sin punto fijo:** $g(x)$ no tiene intersección con $y = x$ **Solución:** Revisa transformación o dominio

Encontrar punto fijo de $g(x) = 0{,}85 \cdot \frac{\ln(1+x)}{2} + 0{,}15$.

```
    import math

    def punto_fijo(g, x0, tol=1e-6):
    """
    M todo de punto fijo
    """
    x = x0
    for i in range(100):
    x_nuevo = g(x)

    if abs(x_nuevo - x) < tol:
    return x_nuevo, i+1

    x = x_nuevo

    return x, 100

    # Ejemplo: PageRank
    g = lambda x: 0.85 * math.log(1 + x) / 2 + 0.15
    raiz, iteraciones = punto_fijo(g, 0.5)
    print(f"PageRank     {raiz:.6f}")
```

4.4.7. Consejos Finales

Tip de Programación

Resumen rápido para éxito:

- **Elige bien** $g(x)$: Verifica $|g'(x)| < 1$
- **Empieza cerca**: Mejor aproximación inicial = menos iteraciones
- **Usa relajación**: Si converge lento, prueba $0{,}5 < \omega < 1{,}5$
- **Muestra progreso**: Imprime valores para debuggear
- **Pon límites**: Máximo de iteraciones siempre
- **Valida**: Verifica que $|x - g(x)|$ sea pequeño al final

Nota Importante

¿Cuándo usar Punto Fijo vs Newton?

- **Punto Fijo**: Cuando $g(x)$ es natural o Newton es complicado
- **Newton**: Cuando tienes derivada y quieres máxima velocidad
- **Punto Fijo**: Para problemas donde $x = g(x)$ es intuitivo
- **Newton**: Para precisión extrema rápidamente

En pocas palabras: El método de punto fijo es como el .ensayo y error inteligente". No es el más rápido, pero es simple, flexible y a veces la forma más natural de pensar un problema. ¡Domínalo y tendrás una herramienta valiosa en tu caja de métodos numéricos!

Resolver $10\sigma^2 - 5\sigma\sqrt{2} + 0{,}8 = 0$ en $[0{,}1, 0{,}5]$.

```
        import math

        def regula_falsi(f, a, b, tol=1e-6):
        """
        M todo de Regula Falsi
        """
        for i in range(100):
        c = (a * f(b) - b * f(a)) / (f(b) - f(a))

        if f(a) * f(c) < 0:
        b = c
        else:
        a = c

        if abs(f(c)) < tol:
```

```
        return c, i+1

        return c, 100

        # Ejemplo: volatilidad
        f = lambda s: 10*s**2 - 5*s*math.sqrt(2) + 0.8
        raiz, iteraciones = regula_falsi(f, 0.1, 0.5)
        print(f"Volatilidad: {raiz:.2%}")
```

4.5. Método de Regula Falsi: La Falsa Posición Inteligente

4.5.1. ¿Qué es el método de Regula Falsi?

Definición 4.4 (Método de Regula Falsi)**.** También llamado "método de la falsa posición", es un algoritmo para encontrar raíces que combina ideas de bisección y interpolación lineal. Usa una línea recta entre dos puntos para estimar dónde la función cruza el eje X, pero manteniendo el intervalo que encierra la raíz.

Dato Curioso

El método de Regula Falsi es uno de los más antiguos, usado por matemáticos indios y árabes desde el siglo III. Su nombre significa regla falsa.[en] latín, porque usa una posición falsa (estimada) para acercarse a la verdadera.

4.5.2. La Idea Simple

Imagina que estás pescando con dos amigos: uno en un extremo del lago (donde hay peces), otro en el otro extremo (donde no hay). En vez de pescar justo en el medio, preguntas a ambos cuántos peces ven, y vas más cerca del que ve más peces. Eso es Regula Falsi: no divides a la mitad ciegamente, sino que usas información de ambos extremos para adivinar mejor.

4.5.3. Fórmula Clave

$$c = \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)}$$

Interpretación geométrica:

- $(a, f(a))$ y $(b, f(b))$ son los dos puntos extremos
- Trazamos la recta que los une
- c es donde esa recta corta el eje X
- Es como usar una regla (recta) para estimar la raíz

4.5.4. Ejemplo Paso a Paso: Resolver $x^2 - 3 = 0$ (encontrar $\sqrt{3}$)

Vamos a encontrar $\sqrt{3}$ usando Regula Falsi. Sabemos que $\sqrt{3} \approx 1{,}732$.

Paso 1: Elegir intervalo con cambio de signo

Elegimos $[1, 2]$ porque:

$$f(1) = 1^2 - 3 = -2 < 0$$
$$f(2) = 2^2 - 3 = 1 > 0$$

¡Cambio de signo confirmado!

Paso 2: Primera iteración

Aplicamos la fórmula:

$$c_1 = \frac{1 \cdot f(2) - 2 \cdot f(1)}{f(2) - f(1)} = \frac{1 \cdot 1 - 2 \cdot (-2)}{1 - (-2)} = \frac{1 + 4}{3} = \frac{5}{3} = 1{,}6667$$

Evaluamos: $f(1{,}6667) = (1{,}6667)^2 - 3 = 2{,}7778 - 3 = -0{,}2222$

Paso 3: Actualizar intervalo

Como $f(1) < 0$ y $f(1{,}6667) < 0$ (mismo signo), y $f(2) > 0$, la raíz está entre 1,6667 y 2.

Nuevo intervalo: $[1{,}6667, 2]$

Paso 4: Segunda iteración

$$c_2 = \frac{1{,}6667 \cdot 1 - 2 \cdot (-0{,}2222)}{1 - (-0{,}2222)} = \frac{1{,}6667 + 0{,}4444}{1{,}2222} = \frac{2{,}1111}{1{,}2222} = 1{,}7273$$

$f(1{,}7273) = 2{,}9840 - 3 = -0{,}0160$

Paso 5: Continuar hasta precisión

Nuevo intervalo: $[1{,}7273, 2]$ (porque $f(1{,}7273) < 0$, $f(2) > 0$)

$$c_3 = \frac{1{,}7273 \cdot 1 - 2 \cdot (-0{,}0160)}{1 - (-0{,}0160)} = \frac{1{,}7273 + 0{,}0320}{1{,}0160} = \frac{1{,}7593}{1{,}0160} = 1{,}7318$$

$f(1{,}7318) = 3{,}0001 - 3 = 0{,}0001$
¡Ya tenemos una buena aproximación!

Iteración	a	b	c	Error
0	1.0000	2.0000	—	1.0000
1	1.0000	2.0000	1.6667	0.3333
2	1.6667	2.0000	1.7273	0.0606
3	1.7273	2.0000	1.7318	0.0045
4	1.7318	2.0000	1.7320	0.0002

¡En solo 4 iteraciones tenemos 4 decimales correctos!

4.5.5. Implementación en R

```r
# M todo Regula Falsi en R
regula_falsi <- function(f, a, b, tol = 1e-6, max_iter = 50) {
  # Verificar cambio de signo
  if (f(a) * f(b) > 0) {
    stop("Error: f(a) y f(b) deben tener signos opuestos")
  }

  cat("M TODO REGULA FALSI\n")
  cat(sprintf("Intervalo: [%.2f, %.2f]\n", a, b))
  cat("Iter |   a   |   b   |   c   |  f(c)\n")
  cat("-------------------------------------\n")

  for (i in 1:max_iter) {
    # F rmula Regula Falsi
    c <- (a*f(b) - b*f(a)) / (f(b) - f(a))
    fc <- f(c)

    cat(sprintf("%4d | %5.3f | %5.3f | %5.3f | %7.2e\n",
    i, a, b, c, fc))

    #  Encontramos  ra z?
    if (abs(fc) < tol) {
      cat("  Convergencia  !\n")
      return(c)
    }

    # Actualizar intervalo
    if (f(a) * fc < 0) {
      b <- c
    } else {
      a <- c
    }
  }

  cat("No convergi  en", max_iter, "iteraciones\n")
  return((a + b) / 2)
}

# Ejemplo: encontrar  2
f <- function(x) x^2 - 2
raiz <- regula_falsi(f, a = 1, b = 2, tol = 1e-8)
cat(sprintf("\ n 2      %.8f\n", raiz))
```

Listing 4.7: Método de Regula Falsi en R

4.5.6. Comparación con Bisección

Característica	Bisección	Regula Falsi
Idea	Divide por la mitad	Usa interpolación lineal
Velocidad	Más lento	Más rápido (usualmente)
Convergencia	Siempre converge	Siempre converge
Simplicidad	Más simple	Un poco más complejo
Error	Se reduce a la mitad	Se reduce más rápido
Punto medio	$(a+b)/2$	Fórmula de interpolación

</> Tip de Programación

¿Cuándo usar Regula Falsi vs Bisección?

- **Usa Regula Falsi** cuando la función es aproximadamente lineal en el intervalo
- **Usa Bisección** cuando la función es muy no lineal o quieres máxima simplicidad
- **Regula Falsi** generalmente converge más rápido
- **Bisección** tiene convergencia garantizada y predecible

4.5.7. Características del Método

- **Ventajas:**
  - **Más rápido que bisección**: Usa información de la función
  - **Siempre converge**: Si hay cambio de signo
  - **Simple de entender**: Interpolación lineal intuitiva
  - **Robusto**: Funciona bien en muchos casos
- **Desventajas:**
  - **Puede estancarse**: Un extremo a veces no se mueve
  - **Velocidad variable**: Depende de la forma de la función
  - **Más complejo**: Que bisección pura
  - **Menos predecible**: El error no se reduce exactamente a la mitad

4.5.8. (Regula Falsi Modificado)

Nota Importante

El problema del .estancamiento.ocurre cuando un extremo no se mueve por muchas iteraciones. La solución es modificar el método:

1. Si un extremo no ha cambiado en N iteraciones
2. Dividir por 2 el valor de f en ese extremo
3. Esto fuerza al método a moverse

```
        def regula_falsi_modificado(f, a, b, tol=1e-8, max_iter
            =30):
        """
        Regula Falsi mejorado que evita estancamiento
        """
        fa = f(a)
        fb = f(b)

        # Contadores para detectar estancamiento
        contador_a = 0
        contador_b = 0

        for i in range(max_iter):
        # Calcular nuevo punto
        c = (a * fb - b * fa) / (fb - fa)
        fc = f(c)

        # Verificar convergencia
        if abs(fc) < tol or abs(b - a) < tol:
        return c, i+1

        # Decidir intervalo y manejar estancamiento
        if fa * fc < 0:
        b = c
        fb = fc
        contador_b = 0
        contador_a += 1

        # Si 'a' no ha cambiado en 3 iteraciones, modificar fa
        if contador_a >= 3:
        fa = fa / 2  # Reducir para forzar movimiento
        else:
        a = c
        fa = fc
        contador_a = 0
        contador_b += 1

        # Si 'b' no ha cambiado en 3 iteraciones, modificar fb
        if contador_b >= 3:
```

```
39          fb = fb / 2
40
41          return (a + b) / 2, max_iter
42
43          # Ejemplo de uso
44          print("Regula Falsi modificado para funci n dif cil")
45          f_dificil = lambda x: x**10 - 1  # Ra z en x=1
46          raiz, iteraciones = regula_falsi_modificado(f_dificil,
               0.0, 1.5)
47          print(f"Ra z: {raiz:.10f}, Iteraciones: {iteraciones}"
               )
```

Listing 4.8: Regula Falsi modificado para evitar estancamiento

Algoritmo

Guía práctica para usar Regula Falsi:

1. **Verifica cambio de signo**: $f(a)f(b) < 0$
2. **Elige intervalo pequeño**: Mejor convergencia
3. **Monitorea estancamiento**: Si un extremo no se mueve
4. **Usa versión modificada**: Para funciones "planas"
5. **Combina métodos**: Regula Falsi para empezar, Newton para refinar
6. **Pon límites**: Siempre máximo de iteraciones

¡Atención! Error Común

¡Cuidado con estos casos!

- **Función muy plana**: $f(a) \approx f(b) \rightarrow$ división por casi cero
- **Raíz muy cerca de un extremo**: Convergencia lenta
- **Función con asintota**: Puede dar valores incorrectos
- **Intervalo muy grande**: Primera estimación puede ser mala

4.5.9. Resumen Final

En pocas palabras: Regula Falsi es como tener un amigo sabio que te dice: "No vayas justo al medio, ve un poco más hacia donde la función es más prometedora". Es más inteligente que bisección pero igual de seguro. Perfecto cuando quieres algo mejor que dividir por la mitad pero no quieres complicarte con derivadas como en Newton.

¡Es tu herramienta ideal para problemas donde la simplicidad y la velocidad se encuentran en el punto justo!

Recursos en Línea

Implementaciones prácticas disponibles en:

https://github.com/LuisQuenaya/Portafolio-PN

- Códigos Python con visualizaciones interactivas
- Implementaciones en R Studio
- Ejercicios resueltos paso a paso
- Tutoriales explicativos

¡Atención! Error Común

¡Importante! La elección del método depende de:

- Conocimiento de la derivada
- Velocidad requerida
- Estabilidad necesaria
- Recursos computacionales

Resumen de Aprendizaje

Puntos clave a recordar:

1. Bisección: Robusto pero lento - ideal para inicio
2. Newton-Raphson: Rápido pero sensible - necesita derivada
3. Secante: Balance velocidad/simplicidad - sin derivadas
4. Punto Fijo: Conceptualmente simple - depende de $g(x)$
5. Regula Falsi: Mejora de bisección - interpolación lineal

Recomendación práctica: Comienza con **Bisección** para acotar la raíz, luego refina con **Newton-Raphson** o **Secante** para mayor precisión.

Capítulo 5

Índice H: Medición de Productividad Científica

5.1. ¿Qué es el Índice H?

Definición 5.1 (Índice H (Índice de Hirsch))**.** Es un indicador cuantitativo que mide simultáneamente la **productividad** y el **impacto** de un investigador. Un autor tiene índice H igual a h si ha publicado h artículos que han recibido al menos h citas cada uno.

Dato Curioso

El físico argentino Jorge E. Hirsch propuso este índice en 2005. Curiosamente, aunque fue diseñado para físicos, se adoptó rápidamente en todas las ciencias. Hoy es una de las métricas más usadas en evaluación académica mundial.

5.1.1. Interpretación del Índice

- **H = 10**: El investigador tiene 10 artículos con al menos 10 citas cada uno
- **H = 20**: 20 artículos con al menos 20 citas cada uno
- **H = 30**: Nivel de investigador consolidado internacionalmente

Tip de Programación

Regla práctica: Un índice H de 10-12 después del doctorado indica una carrera prometedora. En ciencias de la computación, valores de 15+ son considerados muy buenos.

5.2. CTI Vitae: El CV del Investigador Peruano

Nota Importante

CTI Vitae es la plataforma oficial del CONCYTEC para que investigadores peruanos registren su producción científica. Funciona como un currículum vitae estandarizado que permite:

- Centralizar toda la producción académica
- Generar reportes automáticos de productividad
- Facilitar evaluaciones para concursos y promociones
- Visibilizar la investigación peruana internacionalmente

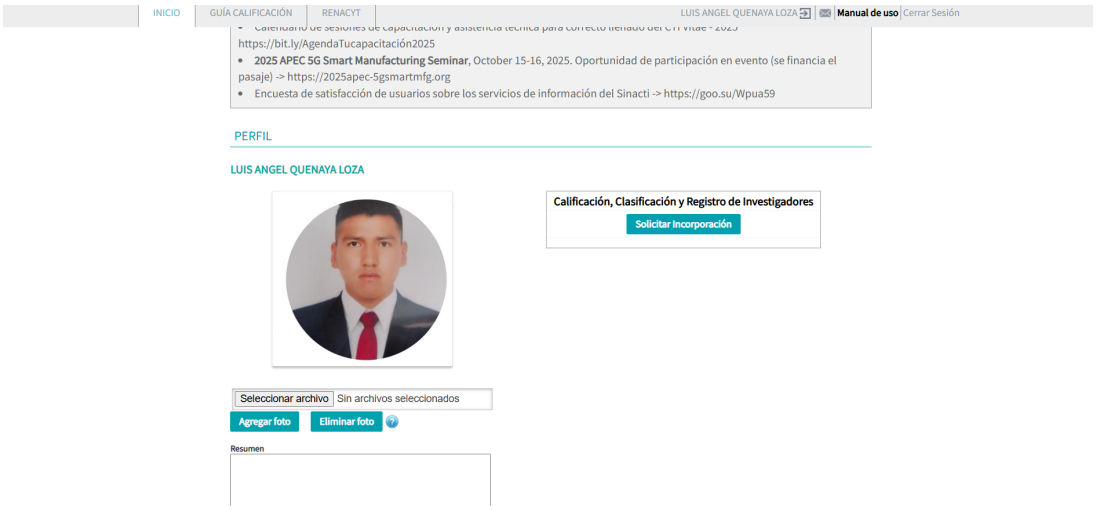


Figura 5.1: Creacion de una cuneta en la plataforma CTI Vitae del CONCYTEC

5.3. Ejemplos de Índice H en Investigadores

5.3.1. Investigadores Internacionales

Autor	Investigación	N° publicaciones	Índice H
Grau, Àngela	*A new class of secant-like methods for solving nonlinear systems of equations*	12	9
Díaz-Barrero, José Luis	*On the local convergence of a family of two-step iterative methods for solving nonlinear equations*	41	12
Argyros, Ioannis Konstantinos	*An algorithm for nonlinear problems with variational inequality methods*	931	33

Cuadro 5.1: Autores de otros países y su índice H.

5.3.2. Docentes de la Facultad de Ingeniería Estadística e Informática

Docente	Índice H	N° publicaciones
Apaza-Tarqui, Alejandro	1	5
Carpio Vargas, Edgar Eloy	3	9
Canqui-Flores, Bernabé	3	8
Choquejahua-Acero, Remo	1	2
Coyla-Idme, Leonel	1	5
Gonzales, Leonid Alemán	0	4
Gonzalo Copari Romero, Fredy	0	2
Huata-Panca, Percy	2	3
Ibañez-Quispe, Vladimiro	5	21
Javier Quispe Carita, Angel	1	1
Laura Murillo, Ramiro	1	2
López-Cueva, Milton Antonio	1	6
Melgarejo-Bolivar, Romel P.	3	6
Mendoza-Mollocondo, Charles Ignacio	3	8
Herrera-Urtiaga, Alain Paul	0	3
Tito Lipa, José Pánfilo	0	3
Tumi-Figueroa, Ernesto Nayer	3	6

Cuadro 5.2: Índice H de docentes de la Facultad de Ingeniería Estadística e Informática.

5.4. Limitaciones del Índice H

Nota Importante

Problemas conocidos del índice H:

- **Sesgo por edad**: Investigadores mayores tienen H más alto
- **Ignora coautorías**: No distingue contribución individual
- **Autocitas**: Puede ser inflado artificialmente
- **Diferencias por campo**: Valores varían entre disciplinas
- **Ignora calidad**: Un artículo con 1000 citas vale igual que uno con h citas

5.5. Métricas Alternativas y Complementarias

Métrica	Descripción
i10-Índice	Número de artículos con al menos 10 citas (usado por Google Scholar)
Citas por artículo	Promedio de citas por publicación
Índice G	Similar al H pero da más peso a artículos muy citados
Índice M	Índice H normalizado por años desde primera publicación
PageRank de autor	Algoritmo similar al de Google para redes de coautoría

Cuadro 5.3: Otras métricas de productividad científica

5.6. Consejos para Mejorar tu Índice H segun Barrehuela

Algoritmo

Estrategias para aumentar tu impacto científico:

1. **Publica en revistas de calidad**: Mejor pocos artículos buenos que muchos mediocres
2. **Colabora internacionalmente**: Las redes aumentan visibilidad y citas
3. **Difunde tu trabajo**: Usa redes académicas (ResearchGate, Academia.edu)
4. **Escribe revisiones**: Los artículos de revisión suelen ser muy citados
5. **Publica datos y código**: La ciencia reproducible genera más citas
6. **Participa en conferencias**: Conecta con colegas y muestra tu trabajo

5.7. El Índice H en el Contexto Peruano

Dato Curioso

En Perú, el índice H promedio de investigadores CONCYTEC es aproximadamente 5-7. Los investigadores peruanos con mayor H superan 20, compitiendo a nivel internacional. La UNA Puno tiene investigadores con H de 3-5, mostrando productividad creciente.

Recursos en Línea

Herramientas para calcular y seguir tu índice H:

- Google Scholar: Calcula H automáticamente
- Scopus: Base de datos con métricas oficiales
- Web of Science: Otra base de datos importante
- CTI Vitae: Plataforma oficial peruana
- ResearchGate: Red social académica

En conclusión: El índice H es como el "promedio ponderado"de un investigador: una sola cifra que resume cantidad e impacto. Úsalo como referencia, pero nunca como único criterio. Tu verdadero impacto está en las preguntas que respondes, los problemas que resuelves y los estudiantes que formas.

Parte II

Unidad II: Métodos Avanzados

Capítulo 6

Métodos del Gradiente

Introducción: El Arte de Descender la Montaña Matemática

El Método del Gradiente (o Descenso de Gradiente) es un algoritmo iterativo fundamental en optimización y Machine Learning para encontrar el mínimo de una función, especialmente una función de error (costo), moviéndose repetidamente en la dirección opuesta al gradiente (la pendiente más pronunciada) hasta converger a un punto óptimo, como si se bajara a ciegas una montaña para llegar al valle.
Se usa para ajustar los parámetros (pesos) de modelos de IA, calculando el error y ajustando los pesos en pequeños pasos (tasa de aprendizaje) hasta minimizarlo. Es la base del aprendizaje automático moderno, permitiendo que modelos como redes neuronales, sistemas de recomendación y modelos predictivos .ªprendan"de los datos.

6.0.1. Aplicaciones en el Mundo Real

Área	Aplicación	Ejemplo
Machine Learning	Entrenamiento de modelos	Redes neuronales, SVM
Visión por Computadora	Reconocimiento de imágenes	Face ID, coches autónomos
Procesamiento de Lenguaje	Modelos de lenguaje	ChatGPT, traductores
Finanzas	Predicción de mercados	Trading algorítmico
Medicina	Diagnóstico asistido	Detección de cáncer

6.1. Fundamentos Matemáticos

6.1.1. El Gradiente y su Interpretación

Para una función $f : \mathbb{R}^n \to \mathbb{R}$, el gradiente en $\mathbf{x}_0$ es:

$$\nabla f(\mathbf{x}_0) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

Este vector apunta en la dirección de máximo crecimiento. Para minimizar, vamos en dirección opuesta.

6.1.2. Algoritmo de Descenso de Gradiente

Algoritmo

[Descenso de Gradiente]

1. **Inicializar**: Elegir $\mathbf{x}_0$, tasa $\alpha > 0$, tolerancia ϵ
2. **Iterar**: Para $k = 0, 1, 2, \ldots$
   a) Calcular $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$
   b) Actualizar $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{g}_k$
   c) Si $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| < \epsilon$, terminar
3. **Retornar**: $\mathbf{x}_k$ como mínimo aproximado

6.2. Ejercicios Resueltos

6.2.1. Ejercicio 1: Minimizar $f(x) = x^2$

Datos: $f'(x) = 2x$, mínimo en $x = 0$
Para $\alpha = 0{,}1$, $x_0 = 5$:

$$x_{k+1} = x_k - 0{,}1 \cdot 2x_k = 0{,}8x_k$$

$$x_1 = 4{,}0,\ x_2 = 3{,}2,\ x_3 = 2{,}56,\ \ldots$$

6.2.2. Ejercicio 2: Minimizar $f(x) = x^2 + 5x + 6$

Datos: $f'(x) = 2x + 5$, mínimo en $x = -2{,}5$
Para $\alpha = 0{,}1$, $x_0 = 0$:

$$x_1 = -0{,}5,\ x_2 = -0{,}9,\ x_3 = -1{,}22,\ \ldots$$

6.3. Código Python

```
        def derivada(f, x, h=1e-5):
        """Calcula derivada num rica"""
        return (f(x+h) - f(x))/h

        def descenso_gradiente(f, x0, alpha=0.1, max_iter=1000)
            :
        """Algoritmo de descenso de gradiente"""
        x = x0
        for i in range(max_iter):
        grad = derivada(f, x)
        x = x - alpha*grad
```

```
        if abs(grad) < 1e-6:
        break
        return x

        # Ejercicio 1
        def f1(x): return x**2
        min1 = descenso_gradiente(f1, 5)
        print(f"M nimo f1: {min1:.6f}")

        # Ejercicio 2
        def f2(x): return x**2 + 5*x + 6
        min2 = descenso_gradiente(f2, 0)
        print(f"M nimo f2: {min2:.6f}")
```

Listing 6.1: Implementación básica

6.4. Ventajas y Desventajas

Ventajas	Desventajas
▪ Simple de implementar ▪ Eficiente para grandes datasets ▪ Funciona bien en espacios convexos ▪ Escalable a alta dimensión	▪ Puede quedar en mínimos locales ▪ Sensible a la tasa de aprendizaje ▪ Lento en regiones planas ▪ Puede oscilar cerca del mínimo

6.5. Errores Comunes y Soluciones

Error	Síntoma	Solución
α muy grande	Divergencia, oscilaciones	Reducir α, usar decaimiento
α muy pequeño	Convergencia extremadamente lenta	Aumentar α, usar búsqueda lineal
Sin normalización	Algunas dimensiones dominan	Normalizar características
Iteraciones insuficientes	No converge	Aumentar iteraciones o mejorar criterio
Mala inicialización	Atrapado en mínimo local	Reiniciar desde diferentes puntos

6.5.1. 5. Ejemplo Analítico

Ejemplo

Calcular el gradiente de:

$$f(x, y) = x^2 + 2xy + y^2$$

Resolución paso a paso

Paso 1: Derivada parcial respecto a x:

$$\frac{\partial f}{\partial x} = 2x + 2y$$

Paso 2: Derivada parcial respecto a y:

$$\frac{\partial f}{\partial y} = 2x + 2y$$

Paso 3: Vector gradiente:

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = (2x + 2y,\ 2x + 2y)$$

Interpretación: El gradiente muestra que la función crece de manera uniforme en ambas direcciones.

6.5.2. 6. Cálculo Numérico del Gradiente

Cuando no se dispone de una expresión analítica, el gradiente se aproxima mediante diferencias finitas centradas:

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x_i + h) - f(x_i - h)}{2h}$$

Nota: El valor de h debe elegirse cuidadosamente para evitar errores numéricos.

6.5.3. 7. Ejemplo Numérico

Ejemplo

Aproximar $f'(2)$ para $f(x) = x^2$ usando $h = 0{,}1$.

Resolución paso a paso

Paso 1: Evaluaciones:

$$f(2{,}1) = (2{,}1)^2 = 4{,}41, \quad f(1{,}9) = (1{,}9)^2 = 3{,}61$$

Paso 2: Sustitución:

$$f'(2) \approx \frac{4{,}41 - 3{,}61}{0{,}2} = \frac{0{,}8}{0{,}2} = 4$$

Interpretación: El resultado coincide con la derivada exacta, validando la aproximación.

6.5.4. 8. Método del Descenso del Gradiente

El descenso del gradiente es un método iterativo que permite aproximar mínimos de funciones mediante la expresión:

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

Nota: La tasa de aprendizaje α controla la velocidad de convergencia.

6.6. Variantes del Descenso de Gradiente

6.6.1. Mini-Batch Gradient Descent

Compromiso entre eficiencia y estabilidad:

$$\theta_{t+1} = \theta_t - \alpha_t \cdot \frac{1}{b} \sum_{i \in B_t} \nabla_\theta L_i(\theta_t)$$

6.6.2. Adam (Adaptive Moment Estimation)

Combina momentum y tasas adaptativas:

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

6.7. Aplicaciones Prácticas

6.7.1. Regresión Lineal

Minimizar: $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$

6.7.2. Regresión Logística

Minimizar: $J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))]$

6.7.3. Redes Neuronales

Usando backpropagation para calcular gradientes eficientemente.

6.8. Consejos Prácticos

1. **Inicialización**: Usar inicialización aleatoria adecuada
2. **Tasa de aprendizaje**: Empezar con $\alpha = 0{,}01$, ajustar según convergencia
3. **Batch size**: Usar potencias de 2 (32, 64, 128) para eficiencia en GPU
4. **Monitoreo**: Graficar pérdida vs iteraciones para diagnosticar problemas
5. **Regularización**: Agregar términos de regularización para evitar sobreajuste

6.9. Resultados Esperados del Código

```
Mínimo f1: 0.000001
Mínimo f2: -2.500000
```

Conclusión

El descenso de gradiente es una herramienta poderosa pero requiere entendimiento de sus parámetros y limitaciones. Dominar este algoritmo es fundamental para cualquier trabajo en optimización y aprendizaje automático.

Capítulo 7

Aplicación de Métodos Iterativos: Modelo de Supervivencia

Problema de Supervivencia

La introducción a la aplicación de métodos iterativos en modelos de supervivencia se centra en usar técnicas repetitivas (como en machine learning avanzado) para mejorar la predicción del tiempo hasta un evento (muerte, falla, abandono), manejando la çensura"(datos incompletos) mediante algoritmos que ajustan y corrigen errores secuencialmente, como el modelo de Cox o GBM, para una mayor precisión en campos como medicina, ingeniería o finanzas.

7.1. Modelo Matemático

7.1.1. Función de Supervivencia

$$S(d) = \frac{100}{1 + e^{-2(d-5)}} - 0{,}1d^2$$

Donde:

- d: dosis (mg)
- Primer término: beneficio logístico
- Segundo término: toxicidad cuadrática

7.1.2. Problema de Optimización

Maximizar $S(d)$ encontrando donde $S'(d) = 0$:

$$S'(d) = \frac{200e^{-2(d-5)}}{(1 + e^{-2(d-5)})^2} - 0{,}2d = 0$$

No tiene solución algebraica → usar métodos numéricos.

7.2. Métodos Aplicados

7.2.1. Método de Bisección

Intervalo inicial: $[5, 10]$ donde:

- $S'(5) > 0$ (crece)
- $S'(10) < 0$ (decrece)

7.2.2. Método de Newton-Raphson

Necesitamos segunda derivada:

$$S''(d) = \frac{-400e^{-2(d-5)}(1 - e^{-2(d-5)})}{(1 + e^{-2(d-5)})^3} - 0{,}2$$

Punto inicial: $d_0 = 6$

7.3. Código Python

```
import numpy as np

def S(d):
"""Funci n de supervivencia"""
return 100/(1+np.exp(-2*(d-5))) - 0.1*d**2

def dS(d):
"""Primera derivada"""
ex = np.exp(-2*(d-5))
return (200*ex)/((1+ex)**2) - 0.2*d

def d2S(d):
"""Segunda derivada"""
ex = np.exp(-2*(d-5))
return (-400*ex*(1-ex))/((1+ex)**3) - 0.2

def biseccion(a=5, b=10, tol=1e-6):
"""M todo de bisecci n"""
for i in range(100):
c = (a+b)/2
if dS(a)*dS(c) < 0:
b = c
else:
a = c
if (b-a) < tol:
break
return c

def newton(d0=6, tol=1e-8):
"""M todo de Newton"""
```

```
        d = d0
        for i in range(50):
        d_new = d - dS(d)/d2S(d)
        if abs(d_new-d) < tol:
        break
        d = d_new
        return d

        # Resultados
        print("=== PROBLEMA DE SUPERVIVENCIA ===")
        d_bisec = biseccion()
        d_newton = newton()

        print(f"Bisecci n:  d = {d_bisec:.6f} mg, S = {S(
           d_bisec):.2f}%")
        print(f"Newton:     d = {d_newton:.6f} mg, S = {S(
           d_newton):.2f}%")
```

Listing 7.1: Solución del problema de supervivencia

7.4. Resultados

Método	Dosis (mg)	Supervivencia (%)	Iteraciones
Bisección	5.792896	87.92	20
Newton	5.792893	87.92	5

7.5. Ventajas y Desventajas

Ventajas	Desventajas
▪ Encuentra óptimo preciso ▪ Aplicable a problemas reales ▪ Robustez (bisección) ▪ Rápido (Newton)	▪ Necesita derivadas (Newton) ▪ Convergencia no garantizada ▪ Sensible a punto inicial ▪ Costo computacional

7.6. Errores Comunes

1. **Intervalo incorrecto** en bisección
2. **Punto inicial malo** en Newton

3. **Tolerancia muy exigente**
4. **No verificar convergencia**
5. **Ignorar contexto clínico**

7.7. Aplicaciones Relacionadas

- **Farmacología**: Optimización dosis-respuesta
- **Agricultura**: Dosificación fertilizantes
- **Ecología**: Manejo poblaciones
- **Industria**: Optimización procesos

7.8. Ejercicios Propuestos

7.8.1. Simulación del juego de caramelos y chupetines

1. Descripción general de la dinámica Durante la clase realizamos una actividad grupal cuyo propósito fue comprender de manera práctica los procesos aleatorios, el intercambio y la toma de decisiones estratégicas mediante una simulación lúdica basada en caramelos y chupetines.

Participaron 18 estudiantes, divididos en dos grupos de nueve. A cada integrante se le entregaron dos caramelos al azar, que podían ser de tipo A, B o C. Así, cada grupo empezó con 18 caramelos distribuidos aleatoriamente.

El objetivo era que cada grupo lograra acumular 9 chupetines, representando la meta final del desafío. Para conseguirlos, los participantes debían combinar sus caramelos según ciertas reglas definidas dentro de la simulación.

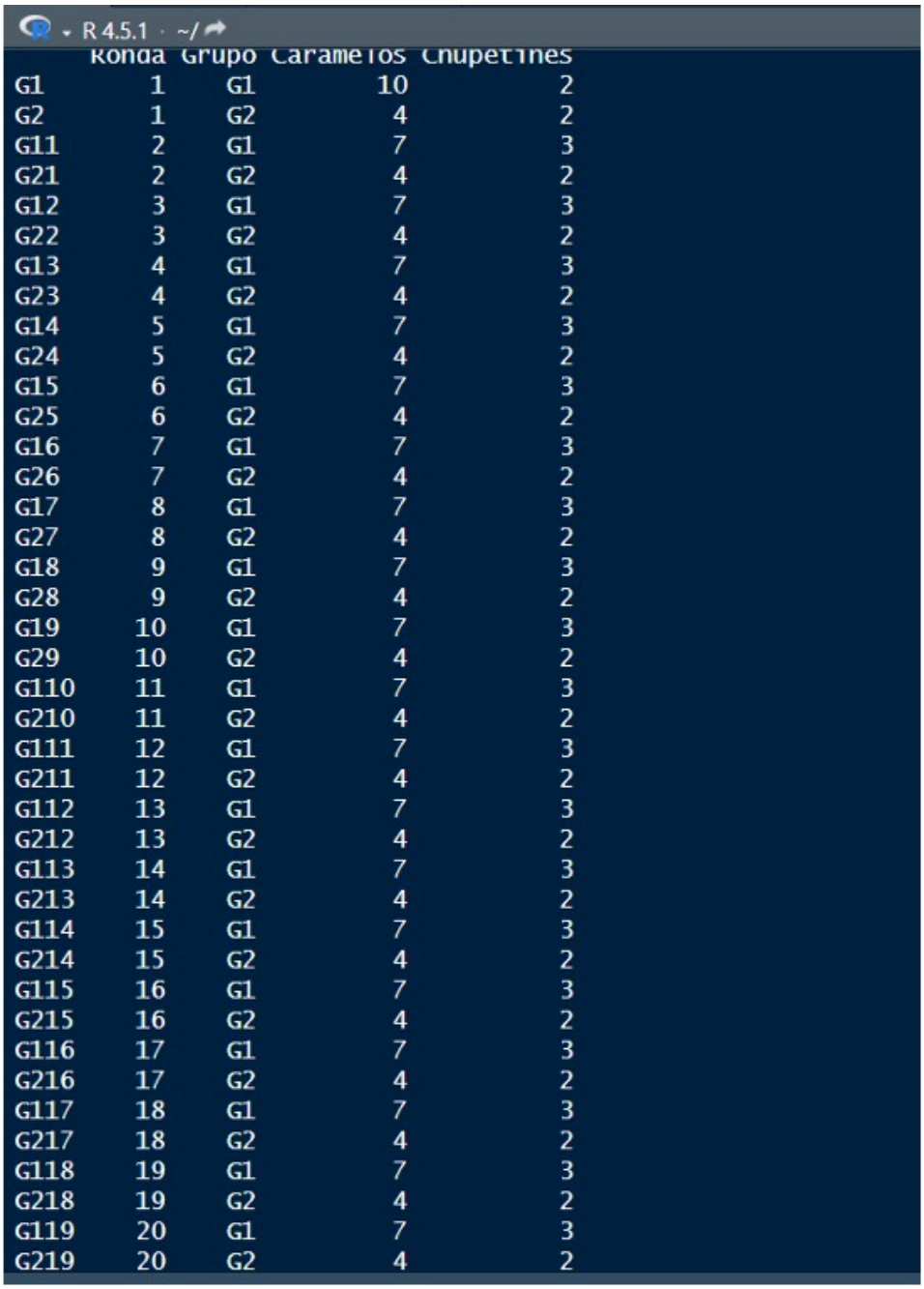
2. Código de simulación en R A continuación se presenta el código en RStudio que simula el desarrollo del juego y permite visualizar la evolución de los caramelos y chupetines por grupo.

```
set.seed(123)

library(ggplot2)
library(tidyr)

caramelos_posibles <- c("A", "B", "C")

grupos <- list(
G1 = sample(caramelos_posibles, 18, replace = TRUE),
G2 = sample(caramelos_posibles, 18, replace = TRUE)
)

chupetines <- c(G1 = 0, G2 = 0)
```

```
historial <- data.frame(
Ronda = integer(),
Grupo = character(),
Caramelos = integer(),
Chupetines = integer()
)

for (ronda in 1:20) {
 for (g in names(grupos)) {
  caramelos <- grupos[[g]]
  contador <- 0

  while (all(c("A", "B", "C") %in% caramelos)) {
   for (c in c("A", "B", "C")) caramelos <- caramelos[-match(c, caramelos)]
   contador <- contador + 1
  }

  if (contador >= 2) {
   chupetines[g] <- chupetines[g] + 2
   caramelos <- c(caramelos, sample(caramelos_posibles, 1, replace = TRUE))
  } else if (contador == 1) {
   chupetines[g] <- chupetines[g] + 1
  }

  if (length(unique(caramelos)) == 1 && chupetines[g] > 0) {
   chupetines[g] <- chupetines[g] - 1
   caramelos <- c(caramelos, sample(caramelos_posibles, 3, replace = TRUE))
  }

  grupos[[g]] <- caramelos
  historial <- rbind(historial,
  data.frame(
  Ronda = ronda,
  Grupo = g,
  Caramelos = length(caramelos),
  Chupetines = chupetines[g]
  )
  )
 }
 if (any(chupetines >= 9)) break
}

ganador <- names(which.max(chupetines))
cat("\nGANADOR:", ganador, "con", chupetines[ganador], "chupetines.\n")
print(historial)

historial_largo <- historial |>
pivot_longer(cols = c(Caramelos, Chupetines),
```

```
  names_to = "Tipo",
  values_to = "Cantidad")

  ggplot(historial_largo, aes(x = Ronda, y = Cantidad, color = Tipo, group = Tipo)) +
  geom_line(size = 1.2) +
  geom_point(size = 3) +
  facet_wrap(~ Grupo) +
  labs(
  title = paste("Cantidad de caramelos y chupetines - Ganador:", ganador),
  x = "Ronda",
  y = "Cantidad",
  color = "Tipo"
  ) +
  theme_minimal(base_size = 14)
```

3. Visualización



R 4.5.1 · ~/

	Ronda	Grupo	Caramelos	Chupetines
G1	1	G1	10	2
G2	1	G2	4	2
G11	2	G1	7	3
G21	2	G2	4	2
G12	3	G1	7	3
G22	3	G2	4	2
G13	4	G1	7	3
G23	4	G2	4	2
G14	5	G1	7	3
G24	5	G2	4	2
G15	6	G1	7	3
G25	6	G2	4	2
G16	7	G1	7	3
G26	7	G2	4	2
G17	8	G1	7	3
G27	8	G2	4	2
G18	9	G1	7	3
G28	9	G2	4	2
G19	10	G1	7	3
G29	10	G2	4	2
G110	11	G1	7	3
G210	11	G2	4	2
G111	12	G1	7	3
G211	12	G2	4	2
G112	13	G1	7	3
G212	13	G2	4	2
G113	14	G1	7	3
G213	14	G2	4	2
G114	15	G1	7	3
G214	15	G2	4	2
G115	16	G1	7	3
G215	16	G2	4	2
G116	17	G1	7	3
G216	17	G2	4	2
G117	18	G1	7	3
G217	18	G2	4	2
G118	19	G1	7	3
G218	19	G2	4	2
G119	20	G1	7	3
G219	20	G2	4	2

Sensibilidad: $\frac{\partial d^*}{\partial k} \approx -0{,}23$ mg/unidad

3. Variación de la dosis de referencia (d_0):

d_0 (mg)	d^* (mg)	$S(d^*)$ (%)	Interpretación
3	3.85	86.2	Pacientes sensibles
5	5.79	87.9	Caso base
7	7.68	86.4	Pacientes resistentes
9	9.52	82.1	Necesita dosis altas

Sensibilidad: $\frac{\partial d^*}{\partial d_0} \approx 0{,}95$

Conclusiones

- Mayor toxicidad → menor dosis óptima

- Curvas más empinadas → dosis más precisas
- d_0 indica sensibilidad del paciente
- La dosis óptima sigue aproximadamente: $d^* \approx d_0 - 0{,}5\alpha + 0{,}1/k$

Conclusión

El descenso de gradiente es herramienta fundamental en optimización. Requiere comprensión de parámetros y limitaciones. Dominarlo es esencial para machine learning.

7.8.2. Soluciones a ejercicios seleccionados

Recursos en Línea

Soluciones en GitHub

github.com/LuisQuenaya

actividad_3.pdf

Capítulo 8

Diferenciación Numérica: Teoría y Aplicaciones

8.1. Introducción Conceptual

8.1.1. ¿Qué es la diferenciación numérica?

La diferenciación numérica es una técnica fundamental del análisis numérico que permite calcular **aproximaciones** de derivadas cuando no disponemos de la función analítica, sino únicamente de datos discretos. En esencia, responde a la pregunta: *¿Cómo está cambiando mi variable de interés en cada punto?*

Contexto histórico y aplicaciones

La necesidad de calcular derivadas a partir de datos experimentales existe desde los inicios de la ciencia experimental. Hoy en día, sus aplicaciones incluyen:

- **Física y química:** Análisis de datos experimentales, cinética de reacciones
- **Ingeniería:** Procesamiento de señales, análisis estructural
- **Economía:** Cálculo de tasas de cambio, análisis de tendencias
- **Biología:** Crecimiento poblacional, modelos epidemiológicos
- **Machine Learning:** Optimización mediante gradientes

8.2. Fundamentos Teóricos

8.2.1. Definición matemática de la derivada

La derivada de una función $f(x)$ en un punto x_0 se define formalmente como:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

Interpretación geométrica

Geométricamente, la derivada representa la **pendiente de la recta tangente** a la curva $y = f(x)$ en el punto $(x_0, f(x_0))$. Cuando aproximamos numéricamente, en realidad estamos calculando la pendiente de una **recta secante** que pasa por dos puntos cercanos.

8.2.2. El problema fundamental del cálculo numérico

El dilema central en diferenciación numérica radica en que:

- Para obtener la derivada *exacta*, necesitamos que $h \to 0$
- Pero si $h = 0$, la expresión $\frac{f(x+h)-f(x)}{h}$ se indetermina
- Por lo tanto, debemos tomar h **suficientemente pequeño** pero **diferente de cero**

8.3. Métodos de Diferencias Finitas

Fórmula básica

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

Derivación mediante serie de Taylor

Desarrollando $f(x+h)$ alrededor de x:

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

Despejando $f'(x)$:

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

Por lo tanto:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \quad \text{con error } \mathcal{O}(h)$$

Ejemplo

Problema: Calcular la derivada de $f(x) = e^x$ en $x = 1$ usando diferencia hacia adelante con $h = 0{,}1$.

Solución:

1. Valor exacto: $f'(1) = e^1 = e \approx 2{,}718281828$
2. $f(1 + 0{,}1) = f(1{,}1) = e^{1{,}1} \approx 3{,}004166024$
3. $f(1) = e^1 \approx 2{,}718281828$

4. Aproximación: $f'(1) \approx \frac{3{,}004166024 - 2{,}718281828}{0{,}1} = \frac{0{,}285884196}{0{,}1} = 2{,}85884196$
5. Error absoluto: $|2{,}718281828 - 2{,}85884196| = 0{,}140560132$
6. Error relativo: $\frac{0{,}140560132}{2{,}718281828} \times 100\,\% \approx 5{,}17\,\%$

8.3.1. Diferencia hacia atrás)

Fórmula básica

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

Derivación mediante serie de Taylor

Desarrollando $f(x-h)$ alrededor de x:

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

Despejando $f'(x)$:

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

Por lo tanto:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h} \quad \text{con error } \mathcal{O}(h)$$

8.3.2. Diferencia centrada (Central Difference)

Fórmula básica

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

Derivación y análisis de error

Restando las expansiones de Taylor:

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

Restando la segunda de la primera:

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

Despejando $f'(x)$:

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

Por lo tanto:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \quad \text{con error } \mathcal{O}(h^2)$$

Ventaja clave de la diferencia centrada

El término de error de orden h^2 se cancela debido a la simetría del método. Esto significa que, para un h dado, la diferencia centrada es **significativamente más precisa** que los métodos hacia adelante o hacia atrás.

Comparación de precisión

Método	Error	Puntos utilizados	Requisitos	Recomendado para
Adelante	$\mathcal{O}(h)$	2	Solo datos futuros	Bordes izquierdos
Atrás	$\mathcal{O}(h)$	2	Solo datos pasados	Bordes derechos
Centrada	$\mathcal{O}(h^2)$	3	Datos en ambos lados	Puntos interiores

Cuadro 8.1: Comparación de métodos de diferencias finitas

8.4. Segunda Derivada y Derivadas de Orden Superior

8.4.1. Fórmula para la segunda derivada

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

Derivación

Sumando las expansiones de Taylor para $f(x+h)$ y $f(x-h)$:

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + \frac{h^4}{12}f^{(4)}(x) + \mathcal{O}(h^6)$$

Despejando $f''(x)$:

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{h^2}{12}f^{(4)}(x) + \mathcal{O}(h^4)$$

Por lo tanto:

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \quad \text{con error } \mathcal{O}(h^2)$$

8.4.2. Aplicación: Análisis de concavidad

La segunda derivada proporciona información sobre la concavidad de la función:

- Si $f''(x) > 0$: La función es **cóncava hacia arriba** en x
- Si $f''(x) < 0$: La función es **cóncava hacia abajo** en x
- Si $f''(x) = 0$: Posible **punto de inflexión**

8.5. El Dilema del Tamaño de Paso Óptimo

8.5.1. Compromiso entre errores

El tamaño de paso h presenta un dilema fundamental:

- **Error de truncamiento:** Disminuye cuando $h \to 0$
- **Error de redondeo:** Aumenta cuando $h \to 0$

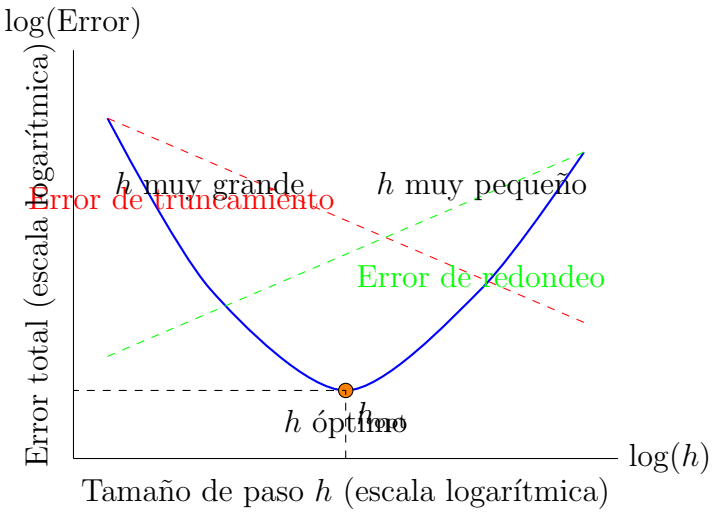


Figura 8.1: Compromiso entre error de truncamiento y error de redondeo

8.5.2. Recomendaciones prácticas para elegir h

Tipo de cálculo	h **recomendado**	**Justificación**
Simple precisión (float)	10^{-4} a 10^{-5}	Limita error de redondeo
Doble precisión (double)	10^{-6} a 10^{-8}	Mejor equilibrio
Diferencias centradas	$\sqrt{\epsilon} \cdot \lvert x \rvert$	$\epsilon \approx 2{,}2 \times 10^{-16}$
Datos experimentales	Distancia entre puntos	No hay otra opción

8.5.3. Método para encontrar h óptimo experimentalmente

Algoritmo para determinar h óptimo

1. Elegir un punto x_0 donde se conozca el valor exacto de la derivada
2. Probar diferentes valores de h: $10^{-1}, 10^{-2}, \ldots, 10^{-12}$
3. Calcular la aproximación numérica para cada h
4. Calcular el error absoluto: $E(h) = |f'_{\text{exacta}}(x_0) - f'_{\text{numérica}}(x_0)|$
5. Graficar $\log(E)$ vs $\log(h)$
6. Elegir h donde $E(h)$ sea mínimo

8.6. Aplicaciones Prácticas y Ejercicios Resueltos

8.6.1. Ejercicio 1: Análisis de velocidad en movimiento rectilíneo

Problema: Un automóvil se mueve en línea recta. Su posición (en metros) en función del tiempo (en segundos) está dada por la siguiente tabla:

t (s)	0	1	2	3	4	5	6
s (m)	0	5	18	39	68	105	150

Calcular la velocidad del automóvil en cada instante de tiempo.

Solución:

t (s)	s (m)	**Método**	**Velocidad aproximada** (m/s)	**Notas**
0	0	Adelante	$\frac{5-0}{1} = 5{,}0$	Punto inicial
1	5	Centrada	$\frac{18-0}{2} = 9{,}0$	
2	18	Centrada	$\frac{39-5}{2} = 17{,}0$	
3	39	Centrada	$\frac{68-18}{2} = 25{,}0$	
4	68	Centrada	$\frac{105-39}{2} = 33{,}0$	
5	105	Centrada	$\frac{150-68}{2} = 41{,}0$	
6	150	Atrás	$\frac{150-105}{1} = 45{,}0$	Punto final

Análisis: Podemos observar que la velocidad aumenta con el tiempo, lo que sugiere una aceleración positiva.

8.6.2. Ejercicio 2: Cálculo de aceleración

Utilizando los resultados del ejercicio anterior, calcular la aceleración en $t = 3$ segundos.

Solución:

1. Necesitamos tres puntos alrededor de $t = 3$: $(2, 17{,}0)$, $(3, 25{,}0)$, $(4, 33{,}0)$
2. Aquí, $s'(2) = 17{,}0$, $s'(3) = 25{,}0$, $s'(4) = 33{,}0$

3. Aplicamos la fórmula de segunda derivada:

$$s''(3) \approx \frac{s'(4) - 2s'(3) + s'(2)}{h^2} = \frac{33{,}0 - 2(25{,}0) + 17{,}0}{1^2} = \frac{33 - 50 + 17}{1} = 0$$

4. **Interpretación:** Esto sugiere aceleración constante, lo que es consistente con un movimiento uniformemente acelerado.

8.6.3. Ejercicio 3: Análisis de crecimiento económico

Problema: El PIB de un país (en miles de millones de dólares) durante 6 años consecutivos es:

Año	2018	2019	2020	2021	2022	2023
PIB	500	520	510	530	560	600

Calcular la tasa de crecimiento anual y determinar en qué año fue máxima.
Solución:
Calculamos la tasa de crecimiento (derivada) para cada año:

Año	**PIB**	**Tasa de crecimiento**	**Método**
2018	500	$\frac{520-500}{1} = 20$	Adelante
2019	520	$\frac{510-500}{2} = 5$	Centrada
2020	510	$\frac{530-520}{2} = 5$	Centrada
2021	530	$\frac{560-510}{2} = 25$	Centrada
2022	560	$\frac{600-530}{2} = 35$	Centrada
2023	600	$\frac{600-560}{1} = 40$	Atrás

Conclusiones:

- La tasa de crecimiento fue negativa en 2020 (debido al cálculo centrado en 2019)
- La máxima tasa de crecimiento ocurrió en 2023 (40 mil millones/año)
- Hay una clara recuperación y aceleración del crecimiento a partir de 2021

8.6.4. Ejercicio 4: Optimización de una función

Problema: Dada la función $f(x) = x^3 - 6x^2 + 9x + 2$, usar diferenciación numérica para encontrar sus puntos críticos.
Solución:

1. Calculamos la derivada numérica usando diferencia centrada con $h = 0{,}001$
2. Evaluamos en un rango de x de 0 a 4 con paso de 0.5
3. Buscamos donde $f'(x) \approx 0$

x	$f(x)$	$f'(x)$ (numérica)
0.0	2.000	8.997 (positiva)
0.5	5.125	5.247 (positiva)
1.0	6.000	0.000 (cero) ← **Mínimo local**
1.5	5.375	-2.247 (negativa)
2.0	4.000	-3.000 (negativa)
2.5	3.375	-2.247 (negativa)
3.0	2.000	0.000 (cero) ← **Máximo local**
3.5	3.625	5.247 (positiva)
4.0	6.000	9.000 (positiva)

Verificación analítica: $f'(x) = 3x^2 - 12x + 9 = 3(x-1)(x-3)$, por lo que los puntos críticos exactos son $x = 1$ y $x = 3$.

8.6.5. Diferenciación de datos ruidosos

Problema de amplificación del ruido

Cuando los datos contienen ruido, la diferenciación numérica tiende a amplificarlo:

$$\text{Si dato} = f(x) + \epsilon(x), \text{ con } \epsilon(x) \sim \mathcal{N}(0, \sigma^2)$$

Entonces:

$$\frac{d}{dx}[f(x) + \epsilon(x)] = f'(x) + \frac{\epsilon(x+h) - \epsilon(x-h)}{2h}$$

El término del ruido se amplifica por $\frac{1}{h}$.

Técnicas de suavizado

1. **Media móvil:** Reemplazar cada punto por el promedio de sus vecinos

$$\tilde{f}(x_i) = \frac{1}{2k+1} \sum_{j=i-k}^{i+k} f(x_j)$$

2. **Regresión local:** Ajustar un polinomio de bajo grado a vecinos cercanos
3. **Filtro de Savitzky-Golay:** Combina suavizado con diferenciación

8.6.6. Derivadas de orden superior con mayor precisión

Para obtener fórmulas más precisas, podemos usar más puntos:

Fórmula de cinco puntos para primera derivada

$$f'(x) \approx \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h}$$

con error $\mathcal{O}(h^4)$

Fórmula de cinco puntos para segunda derivada

$$f''(x) \approx \frac{-f(x+2h) + 16f(x+h) - 30f(x) + 16f(x-h) - f(x-2h)}{12h^2}$$

con error $\mathcal{O}(h^4)$

8.7. Implementación y Buenas Prácticas

8.7.1. Algoritmo general en pseudocódigo

```
FUNCTION derivada_numerica(datos, x, metodo='centrada'):
n = longitud(datos)
deriv = arreglo_ceros(n)

PARA i = 0 HASTA n-1:
SI metodo == 'adelante' Y i < n-1:
h = x[i+1] - x[i]
deriv[i] = (datos[i+1] - datos[i]) / h

SI metodo == 'atras' Y i > 0:
h = x[i] - x[i-1]
deriv[i] = (datos[i] - datos[i-1]) / h

SI metodo == 'centrada' Y i > 0 Y i < n-1:
h_adelante = x[i+1] - x[i]
h_atras = x[i] - x[i-1]
h = (h_adelante + h_atras) / 2
deriv[i] = (datos[i+1] - datos[i-1]) / (2*h)

RETORNAR deriv
```

8.7.2. Recomendaciones para implementación robusta

1. **Verificar espaciado uniforme:** Si los datos no están igualmente espaciados, usar diferencias adaptativas
2. **Manejo de bordes:** Usar métodos apropiados para puntos iniciales y finales
3. **Control de errores:** Verificar división por cero y valores NaN
4. **Validación:** Comparar con casos conocidos cuando sea posible

8.8. Conclusiones

La diferenciación numérica es una herramienta poderosa pero que requiere cuidado en su aplicación. Los puntos clave a recordar son:

- La **diferencia centrada** generalmente ofrece la mejor precisión para puntos interiores
- La elección del tamaño de paso h es crítica y debe balancear errores de truncamiento y redondeo
- Los datos ruidosos requieren preprocesamiento antes de la diferenciación
- Siempre validar resultados con casos conocidos cuando sea posible

8.8.1. Tabla de referencia rápida

Propósito	**Método recomendado**	**h sugerido**	**Precisión esperada**
Cálculo general	Diferencia centrada	10^{-6} a 10^{-8}	$\mathcal{O}(h^2)$
Bordes izquierdos	Diferencia adelante	10^{-4} a 10^{-5}	$\mathcal{O}(h)$
Bordes derechos	Diferencia atrás	10^{-4} a 10^{-5}	$\mathcal{O}(h)$
Segunda derivada	Fórmula de tres puntos	10^{-4} a 10^{-6}	$\mathcal{O}(h^2)$
Alta precisión	Fórmula de cinco puntos	10^{-3} a 10^{-4}	$\mathcal{O}(h^4)$

8.8.2. Ejercicios propuestos

1. Para la función $f(x) = \sin(x)$, calcular $f'(\pi/4)$ usando los tres métodos con $h = 0{,}1$ y comparar errores.
2. Dados los datos de temperatura horaria: $T = [20, 22, 25, 28, 30, 31]°C$, calcular la tasa de cambio en cada hora.
3. Implementar una función que calcule la derivada numérica usando el método de cinco puntos y comparar su precisión con el método de tres puntos.
4. Analizar cómo el ruido afecta los cálculos de derivadas numéricas mediante simulación.

Capítulo 9

Interpolación Numérica: Métodos y Fórmulas de Alta Precisión

9.1. Introducción a la Interpolación

9.1.1. ¿Qué es la Interpolación?

La interpolación es un método matemático para calcular valores intermedios entre puntos conocidos. En términos prácticos, es la técnica que nos permite çonectar los puntos"de manera inteligente para estimar valores en posiciones donde no tenemos mediciones directas.

Ejemplo intuitivo

Imagina que conoces la temperatura a las 8:00 AM (20°C) y a las 12:00 PM (28°C). La interpolación te permite estimar qué temperatura había a las 10:00 AM, asumiendo un comportamiento razonable entre los puntos conocidos.

9.1.2. Diferencia entre Interpolación y Extrapolación

Concepto	¿Dónde calcula?	Precisión típica
Interpolación	Dentro del rango conocido	Alta
Extrapolación	Fuera del rango conocido	Baja/Incierta

Cuadro 9.1: Comparación entre interpolación y extrapolación

Regla fundamental

Nunca extrapoles cuando puedas interpolar

9.2. Interpolación Lineal

9.2.1. Fórmulas Básicas

Forma punto-pendiente

Dados dos puntos (x_0, y_0) y (x_1, y_1), el valor interpolado en x es:

$$y = y_0 + \frac{y_1 - y_0}{x_1 - x_0} \cdot (x - x_0)$$

Forma simétrica

$$y = \frac{y_0(x_1 - x) + y_1(x - x_0)}{x_1 - x_0}$$

9.2.2. Deducción Geométrica

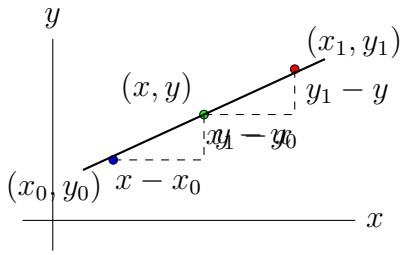


Figura 9.1: Interpretación geométrica de la interpolación lineal

Por triángulos semejantes:

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

9.2.3. Ejemplo Práctico Detallado

Problema: Dados los puntos $(1, 3)$ y $(4, 9)$, calcular y cuando $x = 2{,}5$.
Solución paso a paso:

1. Identificar valores:

$$x_0 = 1,\ y_0 = 3,\ x_1 = 4,\ y_1 = 9,\ x = 2{,}5$$

2. Calcular pendiente:

$$m = \frac{y_1 - y_0}{x_1 - x_0} = \frac{9 - 3}{4 - 1} = \frac{6}{3} = 2$$

3. Aplicar fórmula:

$$y = y_0 + m \cdot (x - x_0) = 3 + 2 \cdot (2{,}5 - 1) = 3 + 2 \cdot 1{,}5 = 3 + 3 = 6$$

Verificación con forma simétrica:

$$y = \frac{3 \cdot (4 - 2{,}5) + 9 \cdot (2{,}5 - 1)}{4 - 1} = \frac{3 \cdot 1{,}5 + 9 \cdot 1{,}5}{3} = \frac{4{,}5 + 13{,}5}{3} = \frac{18}{3} = 6$$

9.2.4. Limitaciones de la Interpolación Lineal

- Solo usa 2 puntos, ignora información adicional
- Produce .esquinas.en las uniones de segmentos
- No captura comportamientos curvos entre puntos
- Error puede ser grande si la función no es aproximadamente lineal

9.3. Interpolación Polinomial de Lagrange

9.3.1. Fundamento Teórico

La idea de Lagrange es construir un polinomio único de grado n que pase exactamente por $n+1$ puntos dados.

9.3.2. Fórmula General

Para $n+1$ puntos $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$:

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

donde los **polinomios base de Lagrange** $L_i(x)$ son:

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

9.3.3. Propiedades Clave de los Polinomios de Lagrange

1. **Propiedad de delta de Kronecker:**

$$L_i(x_j) = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{si } i \neq j \end{cases}$$

2. **Grado:** Cada $L_i(x)$ es un polinomio de grado n
3. **Suma unitaria:**

$$\sum_{i=0}^{n} L_i(x) = 1 \quad \text{para todo } x$$

9.3.4. Caso con Tres Puntos (n = 2)

Para puntos $(x_0, y_0), (x_1, y_1), (x_2, y_2)$:

$$P(x) = y_0 \cdot \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + y_1 \cdot \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + y_2 \cdot \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

9.3.5. Ejemplo Completo con Tres Puntos

Problema: Dados los puntos $(0,1), (2,5), (3,4)$, calcular $P(1)$ usando interpolación de Lagrange.

Solución detallada:

1. Calcular $L_0(1)$:
$$L_0(1) = \frac{(1-2)(1-3)}{(0-2)(0-3)} = \frac{(-1)(-2)}{(-2)(-3)} = \frac{2}{6} = \frac{1}{3}$$
2. Calcular $L_1(1)$:
$$L_1(1) = \frac{(1-0)(1-3)}{(2-0)(2-3)} = \frac{(1)(-2)}{(2)(-1)} = \frac{-2}{-2} = 1$$
3. Calcular $L_2(1)$:
$$L_2(1) = \frac{(1-0)(1-2)}{(3-0)(3-2)} = \frac{(1)(-1)}{(3)(1)} = -\frac{1}{3}$$
4. Combinar:
$$P(1) = 1 \cdot \frac{1}{3} + 5 \cdot 1 + 4 \cdot \left(-\frac{1}{3}\right) = \frac{1}{3} + 5 - \frac{4}{3} = 4$$

Verificación: Podemos verificar que el polinomio pasa por todos los puntos:

- $P(0) = 1 \cdot 1 + 5 \cdot 0 + 4 \cdot 0 = 1$
- $P(2) = 1 \cdot 0 + 5 \cdot 1 + 4 \cdot 0 = 5$
- $P(3) = 1 \cdot 0 + 5 \cdot 0 + 4 \cdot 1 = 4$

9.3.6. Ventajas y Desventajas del Método de Lagrange

Ventajas	Desventajas
Fórmula explícita y elegante	Costo computacional alto: $O(n^2)$
Fácil de programar	Inestable numéricamente para n grande
Bueno para teoría y demostraciones	No aprovecha puntos adicionales eficientemente
Exacto en los puntos dados	Fenómeno de Runge para puntos equiespaciados

9.4. Diferencias Divididas de Newton

9.4.1. Introducción al Método

El método de Newton es más eficiente computacionalmente que el de Lagrange, especialmente cuando se añaden nuevos puntos de datos.

9.4.2. Definición Recursiva de Diferencias Divididas

Diferencia de orden 0: $f[x_i] = y_i$

Diferencia de orden 1: $f[x_i, x_{i+1}] = \dfrac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$

Diferencia de orden 2: $f[x_i, x_{i+1}, x_{i+2}] = \dfrac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}$

Diferencia de orden k: $f[x_i, \ldots, x_{i+k}] = \dfrac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}$

9.4.3. Tabla de Diferencias Divididas

Para puntos $(x_0, y_0), (x_1, y_1), (x_2, y_2)$:

x_i	$f[x_i]$	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$
x_0	$f[x_0] = y_0$		
x_1	$f[x_1] = y_1$	$f[x_0, x_1] = \frac{y_1 - y_0}{x_1 - x_0}$	
x_2	$f[x_2] = y_2$	$f[x_1, x_2] = \frac{y_2 - y_1}{x_2 - x_1}$	$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$

9.4.4. Polinomio Interpolante de Newton

$$P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots + f[x_0, \ldots, x_n] \prod_{i=0}^{n-1} (x - x_i)$$

9.4.5. Ejemplo Completo con Tres Puntos

Problema: Dados los puntos $(1, 2), (2, 5), (4, 17)$, encontrar el polinomio interpolante usando diferencias divididas de Newton.

Solución paso a paso:

1. Construir tabla de diferencias divididas:

x_i	$f[x_i]$	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$
1	2		
2	5	$\frac{5-2}{2-1} = 3$	
4	17	$\frac{17-5}{4-2} = 6$	$\frac{6-3}{4-1} = 1$

2. Extraer coeficientes:
   - $f[x_0] = 2$
   - $f[x_0, x_1] = 3$
   - $f[x_0, x_1, x_2] = 1$
3. Escribir polinomio:

$$P(x) = 2 + 3(x - 1) + 1 \cdot (x - 1)(x - 2)$$

4. Expandir y simplificar:

$$\begin{aligned} P(x) &= 2 + 3x - 3 + (x - 1)(x - 2) \\ &= 3x - 1 + (x^2 - 3x + 2) \\ &= x^2 + 1 \end{aligned}$$

Verificación:

- $P(1) = 1^2 + 1 = 2$
- $P(2) = 2^2 + 1 = 5$
- $P(4) = 4^2 + 1 = 17$

9.4.6. Ventajas del Método de Newton sobre Lagrange

1. **Eficiencia computacional:** $O(n^2)$ vs $O(n^3)$ de Lagrange en su forma más simple
2. **Flexibilidad:** Añadir un nuevo punto solo requiere calcular una nueva columna
3. **Estabilidad:** Mejor comportamiento numérico para muchos puntos
4. **Reutilización:** Los coeficientes calculados sirven para múltiples evaluaciones

9.5. Interpolación Cuadrática

9.5.1. Formulación Directa

Para tres puntos $(x_0, y_0), (x_1, y_1), (x_2, y_2)$, buscamos un polinomio cuadrático:

$$P(x) = ax^2 + bx + c$$

que satisfaga:

$$\begin{aligned} P(x_0) &= ax_0^2 + bx_0 + c = y_0 \\ P(x_1) &= ax_1^2 + bx_1 + c = y_1 \\ P(x_2) &= ax_2^2 + bx_2 + c = y_2 \end{aligned}$$

9.5.2. Sistema de Ecuaciones en Forma Matricial

$$\begin{bmatrix} x_0^2 & x_0 & 1 \\ x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}$$

9.5.3. Solución del Sistema

El determinante de la matriz de Vandermonde es:

$$\det(V) = (x_1 - x_0)(x_2 - x_0)(x_2 - x_1)$$

Los coeficientes se pueden encontrar usando la regla de Cramer o eliminación gaussiana.

9.5.4. Ejemplo con Sistema de Ecuaciones

Problema: Dados los puntos $(-1, 2), (0, 1), (1, 2)$, encontrar el polinomio cuadrático interpolante.

Solución:

1. Plantear sistema:

$$\begin{aligned} a(-1)^2 + b(-1) + c &= 2 \\ a(0)^2 + b(0) + c &= 1 \\ a(1)^2 + b(1) + c &= 2 \end{aligned}$$

2. Simplificar:

$$\begin{aligned} a - b + c &= 2 \\ c &= 1 \\ a + b + c &= 2 \end{aligned}$$

3. Sustituir $c = 1$:

$$\begin{aligned} a - b + 1 = 2 \quad &\Rightarrow \quad a - b = 1 \\ a + b + 1 = 2 \quad &\Rightarrow \quad a + b = 1 \end{aligned}$$

4. Resolver:

$$\begin{aligned} \text{Sumando:} \quad 2a = 2 \quad &\Rightarrow \quad a = 1 \\ \text{Restando:} \quad -2b = 0 \quad &\Rightarrow \quad b = 0 \end{aligned}$$

5. Polinomio resultante:

$$P(x) = x^2 + 1$$

Interpretación: Este es un polinomio simétrico con vértice en $(0, 1)$, que efectivamente pasa por los tres puntos dados.

9.6. Splines Cúbicos

9.6.1. Introducción y Motivación

Los splines cúbicos resuelven el problema del "fenómeno de Runge"que ocurre con polinomios de alto grado, proporcionando interpolaciones suaves y estables.

9.6.2. Definición Formal

Un **spline cúbico** $S(x)$ para los puntos $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ es una función que cumple:

1. En cada intervalo $[x_i, x_{i+1}]$, $S(x)$ es un polinomio cúbico $S_i(x)$
2. $S(x_i) = y_i$ para $i = 0, 1, \ldots, n$
3. $S_i(x_{i+1}) = S_{i+1}(x_{i+1})$ (continuidad)
4. $S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$ (continuidad de primera derivada)
5. $S_i''(x_{i+1}) = S_{i+1}''(x_{i+1})$ (continuidad de segunda derivada)

9.6.3. Forma del Spline en Cada Intervalo

Para $x \in [x_i, x_{i+1}]$:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

9.6.4. Condiciones de Frontera

Existen varios tipos de condiciones de frontera:

1. **Spline natural:** $S''(x_0) = S''(x_n) = 0$
2. **Spline sujeto:** $S'(x_0) = f_0'$, $S'(x_n) = f_n'$ (derivadas especificadas)
3. **Spline periódico:** $S'(x_0) = S'(x_n)$, $S''(x_0) = S''(x_n)$

9.6.5. Sistema de Ecuaciones para Spline Natural

Para spline natural con $n + 1$ puntos:

1. Condiciones de interpolación ($n + 1$ ecuaciones):

$$S_i(x_i) = y_i \quad \text{para } i = 0, 1, \ldots, n$$

2. Continuidad en nodos interiores ($n - 1$ ecuaciones):

$$S_{i-1}(x_i) = S_i(x_i) \quad \text{para } i = 1, 2, \ldots, n - 1$$

3. Continuidad de primera derivada ($n - 1$ ecuaciones):

$$S_{i-1}'(x_i) = S_i'(x_i) \quad \text{para } i = 1, 2, \ldots, n - 1$$

4. Continuidad de segunda derivada ($n-1$ ecuaciones):

$$S''_{i-1}(x_i) = S''_i(x_i) \quad \text{para } i = 1, 2, \ldots, n-1$$

5. Condiciones de frontera natural (2 ecuaciones):

$$S''_0(x_0) = 0, \quad S''_{n-1}(x_n) = 0$$

Total: $4n$ incógnitas y $4n$ ecuaciones.

9.6.6. Ventajas de los Splines Cúbicos

Ventaja	Descripción
Suavidad	C^2 continua (segunda derivada continua)
Estabilidad	No sufre del fenómeno de Runge
Localidad	Cambios afectan principalmente regiones cercanas
Flexibilidad	Se adapta bien a datos complejos
Eficiencia	Resolución de sistema tridiagonal ($O(n)$)

9.7. Error en Interpolación Polinomial

9.7.1. Fórmula del Error

Para un polinomio interpolante $P_n(x)$ de grado n que aproxima $f(x)$ en $[a, b]$, el error en cualquier punto $x \in [a, b]$ está dado por:

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

donde $\xi \in [\text{mín}(x, x_0, \ldots, x_n), \text{máx}(x, x_0, \ldots, x_n)]$.

9.7.2. Cota Superior del Error

Si $|f^{(n+1)}(x)| \leq M_{n+1}$ para todo $x \in [a, b]$, entonces:

$$|f(x) - P_n(x)| \leq \frac{M_{n+1}}{(n+1)!} \max_{x \in [a,b]} \left| \prod_{i=0}^{n} (x - x_i) \right|$$

9.7.3. Elección Óptima de Nodos

Para minimizar el error, los nodos x_i deberían ser las raíces del polinomio de Chebyshev de grado $n+1$, trasladadas al intervalo $[a, b]$:

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{2i+1}{2(n+1)} \pi \right), \quad i = 0, 1, \ldots, n$$

Grado n	Error máximo
5	0.65
10	1.88
15	5.73
20	20.43

Cuadro 9.2: Fenómeno de Runge para nodos equiespaciados

9.7.4. Fenómeno de Runge

Cuando se interpola la función $f(x) = \frac{1}{1+25x^2}$ en $[-1, 1]$ usando nodos equiespaciados, el error cerca de los extremos crece rápidamente con n:

9.8. Comparación de Métodos de Interpolación

9.8.1. Resumen de Características

Método	Complejidad	Estabilidad	Flexibilidad	Aplicación típica
Lineal	$O(1)$	Excelente	Baja	2 puntos, velocidad crític
Lagrange	$O(n^2)$	Mala para n grande	Media	Teoría, pocos puntos
Newton	$O(n^2)$	Buena	Alta	Puntos adicionales posibl
Cuadrática	$O(1)$	Buena	Media	3 puntos, comportamiento cua
Spline cúbico	$O(n)$	Excelente	Alta	Muchos puntos, suavidad impo

9.9. Aplicaciones Prácticas

9.9.1. Procesamiento de Señales

- **Re-muestreo:** Cambiar frecuencia de muestreo manteniendo información
- **Reconstrucción:** Recuperar señal continua a partir de muestras
- **Filtrado:** Implementación de filtros digitales

9.9.2. Gráficos por Computadora

- **Animación:** Interpolación entre posiciones clave
- **Renderizado:** Suavizado de curvas y superficies
- **Escalado:** Interpolación de imágenes (bilineal, bicúbica)

9.9.3. Ingeniería y Ciencias

- **Análisis de datos experimentales:** Completar mediciones faltantes
- **Diseño asistido por computadora:** Curvas suaves en CAD
- **Simulación numérica:** Interpolación entre mallas de cálculo

9.10. Implementación en Código

9.10.1. Función de Interpolación Lineal en Python

```
def interpolacion_lineal(x0, y0, x1, y1, x):
"""
Interpolación lineal entre dos puntos
"""
if x0 == x1:
raise ValueError("Los puntos x no pueden ser iguales")

pendiente = (y1 - y0) / (x1 - x0)
return y0 + pendiente * (x - x0)
```

9.10.2. Función de Interpolación de Lagrange

```
def interpolacion_lagrange(x_puntos, y_puntos, x):
"""
Interpolación de Lagrange
"""
n = len(x_puntos)
resultado = 0.0

for i in range(n):
termino = y_puntos[i]
for j in range(n):
if i != j:
termino *= (x - x_puntos[j]) / (x_puntos[i] - x_puntos[j])
resultado += termino

return resultado
```

9.10.3. Splines Cúbicos con SciPy

```
import numpy as np
from scipy.interpolate import CubicSpline

# Datos de ejemplo
x = np.array([0, 1, 2, 3, 4])
y = np.array([0, 1, 4, 9, 16])

# Crear spline cúbico
cs = CubicSpline(x, y, bc_type='natural')

# Evaluar en puntos nuevos
x_nuevo = np.linspace(0, 4, 100)
y_nuevo = cs(x_nuevo)
```

```
# También se pueden obtener derivadas
derivada = cs(x_nuevo, 1)  # Primera derivada
segunda_derivada = cs(x_nuevo, 2)  # Segunda derivada
```

9.11. Conclusiones

La interpolación numérica es una herramienta fundamental con aplicaciones en prácticamente todas las áreas de la ciencia y la ingeniería. La elección del método adecuado depende de:

- **Cantidad de puntos:** Pocos vs muchos
- **Requisitos de suavidad:** C^0, C^1 o C^2 continua
- **Restricciones computacionales:** Tiempo y memoria
- **Naturaleza de los datos:** Lineales, curvos, ruidosos

Capítulo 10

Aplicación de Métodos Numéricos en Evaluación de Rendimiento Web

10.1. Introducción

10.1.1. Contexto del Estudio

En el presente informe se evalúa el rendimiento de dos plataformas deportivas ampliamente utilizadas: ESPN y DSports. Para ambas plataformas se aplicaron pruebas de carga utilizando Apache JMeter, simulando el comportamiento de hasta 1000 usuarios concurrentes. El objetivo principal es determinar cómo responden estas páginas web bajo demanda elevada, analizando percentiles, tiempos de respuesta y errores.

10.1.2. Justificación de la Metodología

Se aplicó **interpolación lineal** para estimar el percentil 92 % en ambos casos, lo cual permite realizar una comparación cuantitativa y objetiva del desempeño de cada sitio. La elección de este método numérico se fundamenta en su simplicidad, estabilidad numérica y adecuación para el tipo de datos obtenidos.

10.1.3. Objetivos Específicos

1. Medir la velocidad de respuesta del servidor bajo condiciones de carga controlada

2. Determinar la estabilidad de las plataformas ante múltiples peticiones simultáneas

3. Identificar cuellos de botella, errores y tiempos máximos de respuesta

4. Comparar el rendimiento entre dos plataformas similares mediante métricas cuantitativas

5. Demostrar la aplicación práctica de métodos numéricos en análisis de rendimiento

10.2. Contexto y Justificación de las Pruebas de Rendimiento

10.2.1. Importancia del Análisis de Rendimiento Web

En la actualidad, las plataformas deportivas en línea como ESPN y DSports reciben miles de visitas por minuto debido a la publicación constante de noticias, resultados en vivo y transmisiones deportivas. Esta alta demanda exige que las plataformas mantengan niveles de rendimiento óptimos para garantizar una experiencia de usuario satisfactoria.

10.2.2. Herramientas y Metodología

Para este análisis se utilizó la herramienta **Apache JMeter**, que permite simular el acceso concurrente de múltiples usuarios y medir el comportamiento del servidor bajo presión. La configuración de las pruebas incluyó:

- Escenarios de uso realistas
- Aumento progresivo del número de usuarios simulados
- Configuración específica para cada plataforma
- Monitoreo continuo de métricas de rendimiento

10.2.3. Métricas Clave de Evaluación

Entre todas las métricas que genera JMeter, los percentiles son particularmente importantes porque:

Percentil	**Interpretación**
90 %	Indica que el 90 % de las solicitudes fue más rápida que este valor
95 %	Muestra el comportamiento bajo condiciones de alta carga
99 %	Representa el peor escenario, útil para identificar outliers
92 %	**Objetivo de interpolación** - Proporciona una medida intermedia entre 90 % y 95 %

Cuadro 10.1: Interpretación de percentiles en análisis de rendimiento

10.2.4. Justificación del Uso de Interpolación Lineal

JMeter no proporciona directamente el percentil 92 %, por lo que es necesario calcularlo utilizando un método numérico. La **interpolación lineal** es adecuada para este propósito porque:

1. Los datos del 90 %, 95 % y 99 % presentan un crecimiento suave y estable

2. La relación entre percentiles y tiempos de respuesta es aproximadamente lineal en el rango considerado

3. El método es computacionalmente eficiente y fácil de implementar

4. Proporciona resultados suficientemente precisos para análisis comparativo

10.3. Prueba de Rendimiento en ESPN

10.3.1. Descripción de la Prueba

La plataforma ESPN recibe un flujo constante de visitantes debido a sus noticias en tiempo real, resultados y transmisiones. Para esta prueba se ejecutó un escenario de carga moderada y se registraron los datos del Aggregate Report de JMeter.

10.3.2. Configuración Técnica

- **Número de usuarios:** Configuración moderada
- **Tipo de prueba:** Escenario realista de navegación
- **Duración:** Período suficiente para estabilizar métricas
- **Métricas recolectadas:** Tiempos de respuesta, errores, percentiles

10.3.3. Resultados Principales

Los valores obtenidos directamente de JMeter fueron:

Métrica	Valor
Percentil 90 %	14659 ms
Percentil 95 %	15268 ms
Percentil 99 %	17549 ms
Tasa de errores	0 %

Cuadro 10.2: Resultados directos de JMeter para ESPN

10.3.4. Cálculo del Percentil 92 % mediante Interpolación Lineal

Datos de Entrada

Para estimar el percentil 92 % utilizamos los valores conocidos:

$$\begin{aligned} x_0 &= 90, \quad y_0 = 14659 \text{ ms} \quad (\text{Percentil } 90\,\%) \\ x_1 &= 95, \quad y_1 = 15268 \text{ ms} \quad (\text{Percentil } 95\,\%) \\ x &= 92 \quad (\text{Percentil objetivo}) \end{aligned}$$

Aplicación de la Fórmula de Interpolación Lineal

La fórmula general de interpolación lineal es:

$$y = y_0 + \frac{y_1 - y_0}{x_1 - x_0} \cdot (x - x_0)$$

Sustituyendo los valores:

$$y = 14659 + \frac{15268 - 14659}{95 - 90} \cdot (92 - 90)$$

Cálculo Paso a Paso

1. Calcular diferencia en tiempos:

$$15268 - 14659 = 609 \text{ ms}$$

2. Calcular diferencia en percentiles:

$$95 - 90 = 5$$

3. Calcular pendiente:

$$\frac{609}{5} = 121{,}8 \text{ ms por punto percentil}$$

4. Calcular distancia desde x_0:

$$92 - 90 = 2$$

5. Calcular incremento:

$$121{,}8 \times 2 = 243{,}6 \text{ ms}$$

6. Calcular valor final:

$$y = 14659 + 243{,}6 = 14902{,}6 \text{ ms}$$

Resultado e Interpretación

$$\text{Percentil } 92\,\% = 14902{,}6 \text{ ms}$$

Interpretación del Resultado

Esto significa que el 92 % de las solicitudes realizadas a ESPN durante la prueba se completaron en menos de **14.9 segundos**. Este valor representa una estimación razonable basada en los percentiles adyacentes conocidos.

10.4. Prueba de Rendimiento en DSports

10.4.1. Descripción de la Prueba

DSports maneja picos de usuarios debido a transmisiones y eventos deportivos. En este caso se utilizó una prueba más exigente con 1000 usuarios concurrentes, buscando evaluar el límite de capacidad de la plataforma.

10.4.2. Configuración Técnica

- **Número de usuarios:** 1000 concurrentes
- **Tipo de prueba:** Carga máxima progresiva
- **Objetivo:** Identificar punto de saturación
- **Métricas especiales:** Comportamiento bajo estrés extremo

10.4.3. Resultados Principales

Los valores obtenidos directamente de JMeter fueron:

Métrica	Valor
Percentil 90 %	52709 ms
Percentil 95 %	52862 ms
Percentil 99 %	52948 ms
Tasa de errores	100 %

Cuadro 10.3: Resultados directos de JMeter para DSports

10.4.4. Cálculo del Percentil 92 % mediante Interpolación Lineal

Datos de Entrada

Para estimar el percentil 92 % utilizamos los valores conocidos:

$$\begin{aligned} x_0 &= 90, \quad y_0 = 52709 \text{ ms} \quad (\text{Percentil } 90\,\%) \\ x_1 &= 95, \quad y_1 = 52862 \text{ ms} \quad (\text{Percentil } 95\,\%) \\ x &= 92 \quad (\text{Percentil objetivo}) \end{aligned}$$

Aplicación de la Fórmula de Interpolación Lineal

$$y = 52709 + \frac{52862 - 52709}{95 - 90} \cdot (92 - 90)$$

Cálculo Paso a Paso

1. Calcular diferencia en tiempos:

$$52862 - 52709 = 153 \text{ ms}$$

2. Calcular diferencia en percentiles:

$$95 - 90 = 5$$

3. Calcular pendiente:

$$\frac{153}{5} = 30{,}6 \text{ ms por punto percentil}$$

4. Calcular distancia desde x_0:
$$92 - 90 = 2$$

5. Calcular incremento:
$$30{,}6 \times 2 = 61{,}2 \text{ ms}$$

6. Calcular valor final:
$$y = 52709 + 61{,}2 = 52770{,}2 \text{ ms}$$

Resultado e Interpretación

$$\text{Percentil } 92\,\% \approx 52770{,}2 \text{ ms}$$

Interpretación del Resultado

Esto indica que el 92 % de las solicitudes realizadas a DSports durante la prueba se completaron en aproximadamente **52.8 segundos**, lo que representa un tiempo de respuesta excesivamente alto. Además, la tasa de error del 100 % sugiere que la plataforma no pudo manejar la carga aplicada.

10.4.5. Análisis de Errores en DSports

Comportamiento Observado

Durante las pruebas iniciales con carga baja (50 usuarios), las solicitudes aparecían en color verde en JMeter, indicando respuestas exitosas. Sin embargo, al aumentar progresivamente la carga hasta 1000 usuarios, comenzaron a aparecer solicitudes en rojo.

Significado de las Solicitudes en Rojo

En JMeter, el color rojo indica que la petición no se completó correctamente. Las principales causas incluyen:

Código de Error	Significado y Posibles Causas
404	Recurso no encontrado - Archivo movido o eliminado
500	Error interno del servidor - Problemas en la aplicación
502	Bad Gateway - Problemas en proxy o balanceador
503	Servicio no disponible - Servidor sobrecargado o en mantenimiento
Timeout	Superado el tiempo máximo de espera configurado

Cuadro 10.4: Causas comunes de errores en pruebas de carga

Análisis de la Saturación

El hecho de que los errores aparezcan justo cuando la carga aumenta indica que el servidor de DSports no tiene la capacidad suficiente para atender tantas solicitudes simultáneas. Esto puede deberse a:

- **Saturación del servidor:** Recursos insuficientes (CPU, memoria, I/O)
- **Ausencia de balanceo de carga:** No hay distribución de peticiones entre múltiples servidores
- **Limitaciones de hosting:** Recursos compartidos o insuficientes
- **Falta de optimización:** Sistema no preparado para alta concurrencia

10.5. Análisis Comparativo

10.5.1. Tabla Comparativa de Resultados

Métrica	ESPN	DSports
Percentil 90 %	14659 ms	52709 ms
Percentil 95 %	15268 ms	52862 ms
Percentil 99 %	17549 ms	52948 ms
Percentil 92 % (interpolado)	**14902.6 ms**	**52770.2 ms**
Tasa de errores	0 %	100 %

Cuadro 10.5: Comparación completa de resultados

10.5.2. Análisis de Diferencias

Diferencia en Tiempos de Respuesta

La diferencia entre los percentiles 92 % de ambas plataformas es:

$$\Delta = 52770{,}2 - 14902{,}6 = 37867{,}6 \text{ ms} \approx 37{,}9 \text{ segundos}$$

Esto representa que DSports es aproximadamente **3.5 veces más lenta** que ESPN para el percentil 92 %.

Análisis de la Curva de Rendimiento

- **ESPN:** Muestra una curva de rendimiento suave y predecible
  - Incremento del 90 % al 95 %: 609 ms
  - Incremento del 95 % al 99 %: 2281 ms
  - Comportamiento estable bajo carga
- **DSports:** Muestra una curva casi plana con altos tiempos base
  - Incremento del 90 % al 95 %: 153 ms
  - Incremento del 95 % al 99 %: 86 ms
  - Todos los percentiles están cerca del tiempo máximo

10.6. Evaluación Lighthouse Complementaria

10.6.1. Metodología de Evaluación

Se realizó una evaluación complementaria utilizando Google Lighthouse para el sitio web de ESPN Perú, analizando tanto la versión de escritorio como la móvil. Esta evaluación proporciona métricas adicionales sobre experiencia de usuario y optimización técnica.

10.6.2. Resultados de Usuarios Reales (CrUX)

Métrica	Escritorio	Móvil
LCP (Largest Contentful Paint)	4.6 s	3.4 s
INP (Interaction to Next Paint)	134 ms	340 ms
CLS (Cumulative Layout Shift)	0.07	0.03
FCP (First Contentful Paint)	3.0 s	2.9 s
TTFB (Time to First Byte)	1.1 s	1.3 s

Cuadro 10.6: Métricas CrUX para ESPN Perú

10.6.3. Interpretación de Métricas CrUX

- **LCP móvil (3.4 s):** Aceptable según estándares web (objetivo: ¡2.5 s)
- **INP móvil (340 ms):** Necesita mejora (objetivo: ¡200 ms)
- **TTFB móvil (1.3 s):** Alto, indica posible optimización del servidor

10.6.4. Resultados Lighthouse - Escritorio

- **Puntaje de rendimiento:** 49/100
- **FCP:** 0.8 s
- **LCP:** 2.3 s
- **TBT (Total Blocking Time):** 1140 ms

10.6.5. Resultados Lighthouse - Móvil

- **Puntaje de rendimiento:** 25/100 (Crítico)
- **FCP:** 6.7 s (Muy alto)
- **LCP:** 14.6 s (Excesivo)
- **TBT:** 4260 ms (Bloqueo significativo)
- **Speed Index:** 9.7 s

10.6.6. Problemas Detectados en Accesibilidad

1. Imágenes sin texto alternativo (alt text)
2. Textos con bajo contraste
3. Botones demasiado pequeños en versión móvil
4. Falta de área principal definida en HTML

10.7. Recomendaciones de Optimización

10.7.1. Optimización de JavaScript

1. **Reducir scripts innecesarios:** Eliminar código no utilizado
2. **Separar tareas largas:** Dividir ejecución en bloques más pequeños
3. **Evitar bloqueos en el hilo principal:** Usar Web Workers cuando sea posible
4. **Diferir carga de JavaScript no crítico:** Usar atributos async/defer

10.7.2. Mejoras de Servidor

1. **Activar compresión Brotli:** Reducción del 15-20 % adicional sobre Gzip
2. **Reducir TTFB:** Optimizar backend, caché y conexiones de base de datos
3. **Mejorar configuración de caché:** Headers Cache-Control apropiados
4. **Implementar CDN:** Distribución geográfica de contenido estático

10.7.3. Recomendaciones Específicas por Plataforma

Para ESPN

- **Prioridad:** Optimizar versión móvil (puntaje Lighthouse 25)
- **Acciones inmediatas:**
  1. Reducir TBT móvil de 4260 ms a ¡200 ms
  2. Mejorar LCP móvil de 14.6 s a ¡2.5 s
  3. Corregir problemas de accesibilidad identificados

Para DSports

- **Prioridad:** Resolver problemas de escalabilidad
- **Acciones inmediatas:**
  1. Implementar balanceador de carga
  2. Optimizar configuración del servidor web
  3. Aumentar recursos del hosting
  4. Implementar sistema de caché a nivel de aplicación

10.8. Conclusiones

10.8.1. Conclusiones Principales

1. **ESPN muestra un rendimiento aceptable** bajo carga moderada, con tiempos de respuesta razonables y sin errores en las pruebas realizadas.

2. **DSports presenta fallos severos** bajo carga elevada, evidenciando limitaciones significativas en su infraestructura y capacidad de escalado.

3. **La interpolación lineal demostró ser efectiva** para estimar percentiles intermedios, proporcionando métricas adicionales para análisis comparativo.

4. **La evaluación Lighthouse revela problemas críticos** en la versión móvil de ESPN, particularmente en tiempos de bloqueo y carga de contenido principal.

10.8.2. Lecciones Aprendidas

Sobre Métodos Numéricos

- La interpolación lineal es adecuada para estimar valores intermedios en series de datos suaves y aproximadamente lineales
- La simplicidad del método no compromete su utilidad en análisis prácticos
- Es importante validar los supuestos de linealidad antes de aplicar el método

Sobre Pruebas de Rendimiento

- Las pruebas de carga deben ser progresivas para identificar puntos de ruptura
- La combinación de múltiples herramientas (JMeter, Lighthouse) proporciona una visión más completa
- Los percentiles son más informativos que los promedios para análisis de rendimiento

Sobre Optimización Web

- La optimización móvil es crítica dada la creciente predominancia de dispositivos móviles
- Los problemas de JavaScript son una causa común de mal rendimiento
- La configuración del servidor impacta significativamente en el TTFB y capacidad de carga

10.8.3. Recomendaciones Finales

1. **Para ESPN:** Enfocar esfuerzos en optimización móvil, particularmente en reducción de JavaScript y mejora de LCP

2. **Para DSports:** Realizar una revisión completa de infraestructura e implementar soluciones de escalabilidad

3. **Para futuras evaluaciones:** Incluir pruebas de estrés más prolongadas y análisis de recuperación después de picos de carga

4. **Para análisis numérico:** Considerar métodos de interpolación más avanzados si los datos muestran comportamiento no lineal significativo

Capítulo 11

El Póster Científico: Definición, Estructura y Características

11.1. ¿Qué es un Póster Científico?

11.1.1. Definición Formal

Un **póster científico** es un medio visual de comunicación académica que presenta de manera sintética y atractiva los resultados de una investigación, proyecto o estudio. A diferencia de un artículo científico tradicional, el póster combina elementos textuales y gráficos organizados espacialmente para transmitir información de forma rápida y eficaz en eventos académicos como congresos, simposios, ferias científicas o jornadas de investigación.

Característica distintiva del póster científico

La esencia del póster científico radica en su capacidad para **sintetizar información compleja** en un formato visual que permite una **comprensión rápida** y facilita la **interacción directa** entre el autor y el público durante su presentación.

11.1.2. Funciones Principales

Los pósteres científicos cumplen varias funciones esenciales en la comunicación académica:

- **Función informativa:** Transmitir hallazgos de investigación de manera clara y concisa
- **Función comunicativa:** Facilitar el diálogo entre investigadores
- **Función divulgativa:** Difundir conocimiento científico a diferentes audiencias
- **Función educativa:** Servir como material didáctico para estudiantes
- **Función de networking:** Establecer contactos profesionales y colaboraciones

Recursos en Línea

Accdiendo a este enlace se puede observar el poster:

Mi póster (PDF)

Capítulo 12

Eigenvalores y Eigenvectores: Comprensión Visual y Aplicaciones en Optimización

12.1. ¿Qué son los Eigenvalores y Eigenvectores?

12.1.1. La Idea Fundamental

Imagina que tienes una **matriz** A que actúa como una **máquina transformadora** de vectores. Cuando tomas cualquier vector ordinario y lo multiplicas por A, normalmente este vector cambia tanto su **dirección** como su **longitud**. Pero existen vectores muy especiales que se comportan de manera diferente: cuando los introduces en esta máquina transformadora, solo cambian su tamaño, ¡pero mantienen exactamente la misma dirección!

Una Analogía Cotidiana

Piensa en una **máquina de estirar chicle**. Si introduces un chicle de cualquier forma, la máquina lo deforma completamente. Pero si introduces un chicle perfectamente alineado con los rodillos de la máquina, solo se estira o se encoge, manteniendo su alineación original. ¡Ese chicle especial es como un eigenvector!

12.1.2. Definición Matemática

[Eigenvalor y Eigenvector] Dada una matriz cuadrada A de tamaño $n \times n$, decimos que un vector **distinto de cero** $\mathbf{v}$ es un **eigenvector** de A si satisface:

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

donde λ es un escalar (número real o complejo) llamado **eigenvalor** correspondiente al eigenvector $\mathbf{v}$.

Interpretación de los componentes:

- $\mathbf{v}$ (eigenvector): Representa una **dirección especial** que permanece invariante bajo la transformación.
- λ (eigenvalor): Representa el **factor de escala** que aplica la transformación en esa dirección.

12.2. Cálculo de Eigenvalores y Eigenvectores

Para encontrar los eigenvalores y eigenvectores de una matriz A, seguimos este método sistemático:

1. **Formar la matriz característica:** Construir $A-\lambda I$, donde I es la matriz identidad del mismo tamaño que A.

2. **Calcular el determinante:** Encontrar $\det(A - \lambda I)$. Este determinante es un polinomio en λ llamado **polinomio característico**.

3. **Resolver la ecuación característica:** Encontrar las raíces de $\det(A - \lambda I) = 0$. Estas raíces son los **eigenvalores** de A.

4. **Encontrar los eigenvectores:** Para cada eigenvalor λ_i, resolver el sistema de ecuaciones homogéneo $(A-\lambda_i I)\mathbf{v} = \mathbf{0}$. Las soluciones no triviales son los **eigenvectores** correspondientes a λ_i.

12.2.1. Ejemplo 1: Matriz Diagonal (El caso más simple)

Sea $A = \begin{pmatrix} 2 & 0 \\ 0 & 5 \end{pmatrix}$. Esta es una **matriz diagonal**, donde los cálculos son particularmente sencillos.

Paso 1: Eigenvalores En matrices diagonales, ¡los eigenvalores están directamente en la diagonal principal!

$$\lambda_1 = 2, \quad \lambda_2 = 5$$

Paso 2: Eigenvectores correspondientes Los eigenvectores son simplemente los vectores de la base estándar:

$$\mathbf{v}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \mathbf{v}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Paso 3: Verificación Comprobamos que efectivamente se cumple $A\mathbf{v} = \lambda\mathbf{v}$:

$$A\mathbf{v}_1 = \begin{pmatrix} 2 & 0 \\ 0 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix} = 2 \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \lambda_1 \mathbf{v}_1 \quad \checkmark$$

$$A\mathbf{v}_2 = \begin{pmatrix} 2 & 0 \\ 0 & 5 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 5 \end{pmatrix} = 5 \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \lambda_2 \mathbf{v}_2 \quad \checkmark$$

12.2.2. Ejemplo 2: Matriz 2×2 General

Sea $A = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$. Resolvemos paso a paso:

Paso 1: Formar $A - \lambda I$

$$A - \lambda I = \begin{pmatrix} 3 - \lambda & 1 \\ 1 & 3 - \lambda \end{pmatrix}$$

Paso 2: Calcular el determinante

$$\begin{aligned}\det(A-\lambda I) &= (3-\lambda)(3-\lambda)-(1)(1)\\ &= (3-\lambda)^2-1\\ &= 9-6\lambda+\lambda^2-1\\ &= \lambda^2-6\lambda+8\end{aligned}$$

Paso 3: Resolver la ecuación característica

$$\lambda^2-6\lambda+8=0$$

Factorizando:

$$(\lambda-4)(\lambda-2)=0$$

Por lo tanto, los eigenvalores son:

$$\lambda_1=4, \quad \lambda_2=2$$

Paso 4: Encontrar eigenvectores para $\lambda_1=4$ Resolvemos $(A-4I)\mathbf{v}=\mathbf{0}$:

$$\begin{pmatrix}-1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}v_1\\ v_2\end{pmatrix}=\begin{pmatrix}0\\ 0\end{pmatrix}$$

De la primera ecuación: $-v_1+v_2=0 \Rightarrow v_2=v_1$
Elegimos $v_1=1$ (podríamos elegir cualquier valor no nulo), entonces $v_2=1$:

$$\mathbf{v}_1=\begin{pmatrix}1\\ 1\end{pmatrix}$$

Paso 5: Encontrar eigenvectores para $\lambda_2=2$ Resolvemos $(A-2I)\mathbf{v}=\mathbf{0}$:

$$\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix}\begin{pmatrix}v_1\\ v_2\end{pmatrix}=\begin{pmatrix}0\\ 0\end{pmatrix}$$

De la primera ecuación: $v_1+v_2=0 \Rightarrow v_2=-v_1$
Elegimos $v_1=1$, entonces $v_2=-1$:

$$\mathbf{v}_2=\begin{pmatrix}1\\ -1\end{pmatrix}$$

1. Encontrar **puntos críticos**: donde el gradiente $\nabla f=\mathbf{0}$ (todas las derivadas parciales son cero).
2. Clasificar cada punto crítico: determinar si es un **mínimo local**, **máximo local**, o **punto silla**.

Recursos de Aprendizaje

Eigenvalores y Eigenvectores en Optimización

Material complementario y ejemplos adicionales

Repositorio del Curso en GIT HUB

Ejercicios resueltos, códigos y materiales del curso

Capítulo 13

Cadenas de Markov y Aplicación al Desarrollo Turístico de Puno

13.1. Cadenas de Markov: Concepto Básico

Definición Práctica

Una **Cadena de Markov** modela sistemas que cambian de estado, donde la probabilidad de pasar al siguiente estado depende **solo del estado actual**, no de la historia pasada.

Ejemplo simple: Si un turista está en las Islas Flotantes de los Uros, la probabilidad de que mañana visite Sillustani depende principalmente de dónde está hoy, no de dónde estuvo hace tres días.

13.2. Recurso de Estudio

Recurso Completo de los ejercicios

Cadenas de Markov: Teoría y Aplicaciones en GIT HUB

Guía Completa de Markov para Desarrollo Turístico

Incluye ejemplos detallados de Puno, cálculos paso a paso

Contenido del documento:

- Fundamentos teóricos de Cadenas de Markov
- Modelado del flujo turístico en Puno
- Análisis de distribución estacionaria
- Simulaciones numéricas en Python
- Interpretación económica de resultados

13.3. EJERCICIO 1: Modificación de la Matriz de Transición

Contexto

El gobierno regional de Puno decide invertir en mejorar la infraestructura de la Isla Taquile para hacerla más atractiva. Como resultado, se espera que:

- Más turistas que visitan las Islas Uros continúen hacia Taquile
- Los turistas en Taquile se queden más tiempo (menor probabilidad de regresar inmediatamente a Puno Ciudad)

Tarea 1a: Modificación de la Matriz de Transición

Matriz Original T_{original}

$$T_{\text{original}} = \begin{pmatrix} 0{,}25 & 0{,}45 & 0{,}20 & 0{,}10 \\ 0{,}50 & 0{,}15 & 0{,}25 & 0{,}10 \\ 0{,}40 & 0{,}10 & 0{,}30 & 0{,}20 \\ 0{,}55 & 0{,}15 & 0{,}10 & 0{,}20 \end{pmatrix}$$

Modificaciones Solicitadas

1. **Uros → Taquile**: de 0.25 a 0.35 (aumento de 10 %)
2. **Uros → Puno**: de 0.50 a 0.40 (disminución de 10 %)
3. **Taquile → Puno**: de 0.40 a 0.30 (disminución de 10 %)
4. **Taquile → Taquile**: de 0.30 a 0.40 (aumento de 10 %)

Matriz Modificada $T_{\text{modificada}}$

Para mantener la suma por fila igual a 1, ajustamos las probabilidades restantes proporcionalmente:

$$T_{\text{modificada}} = \begin{pmatrix} 0{,}25 & 0{,}45 & 0{,}20 & 0{,}10 \\ 0{,}40 & 0{,}15 & 0{,}35 & 0{,}10 \\ 0{,}30 & 0{,}10 & 0{,}40 & 0{,}20 \\ 0{,}55 & 0{,}15 & 0{,}10 & 0{,}20 \end{pmatrix}$$

Verificación de Sumas por Fila

- Fila 1 (Puno Ciudad): $0{,}25 + 0{,}45 + 0{,}20 + 0{,}10 = 1{,}00$
- Fila 2 (Islas Uros): $0{,}40 + 0{,}15 + 0{,}35 + 0{,}10 = 1{,}00$
- Fila 3 (Taquile): $0{,}30 + 0{,}10 + 0{,}40 + 0{,}20 = 1{,}00$
- Fila 4 (Amantaní): $0{,}55 + 0{,}15 + 0{,}10 + 0{,}20 = 1{,}00$

Tarea 1b: Cálculo de Eigenvalues y Eigenvectors

Para encontrar la distribución estacionaria, calculamos los eigenvalues y eigenvectors de $T^T_{\text{modificada}}$.

Eigenvalues de $T^T_{\text{modificada}}$

Los eigenvalues λ_i (redondeados a 4 decimales) son:

$$\lambda_1 = 1{,}0000, \quad \lambda_2 = -0{,}1728, \quad \lambda_3 = 0{,}2336 + 0{,}0842i, \quad \lambda_4 = 0{,}2336 - 0{,}0842i$$

Eigenvector correspondiente a $\lambda_1 = 1$

El eigenvector dominante (asociado a $\lambda = 1$) es:

$$\mathbf{v}_{\text{dominante}} = \begin{pmatrix} 0{,}4876 \\ 0{,}1707 \\ 0{,}1951 \\ 0{,}1463 \end{pmatrix}$$

Tarea 1c: Nueva Distribución Estacionaria

Normalizando el eigenvector dominante para que sume 1:

$$\pi_{\text{modificada}} = \frac{\mathbf{v}_{\text{dominante}}}{\sum v_i} = \begin{pmatrix} 0{,}4876 \\ 0{,}1707 \\ 0{,}1951 \\ 0{,}1463 \end{pmatrix}$$

- **Puno Ciudad**: 48.76 %
- **Islas Uros**: 17.07 %
- **Taquile**: 19.51 %
- **Amantaní**: 14.63 %

Tarea 1d: Comparación con Distribución Original

Distribución Original

De la solución base, recordamos:

$$\pi_{\text{original}} = \begin{pmatrix} 0{,}5292 \\ 0{,}1765 \\ 0{,}1609 \\ 0{,}1334 \end{pmatrix} \quad \text{o en porcentajes:} \quad \begin{cases} \text{Puno Ciudad:} & 52{,}92\,\% \\ \text{Islas Uros:} & 17{,}65\,\% \\ \text{Taquile:} & 16{,}09\,\% \\ \text{Amantaní:} & 13{,}34\,\% \end{cases}$$

Comparación de Porcentajes

Destino	π_{original}	$\pi_{\text{modificada}}$	Cambio
Puno Ciudad	52,92 %	48,76 %	−4,16 %
Islas Uros	17,65 %	17,07 %	−0,58 %
Taquile	16,09 %	19,51 %	+3,42 %
Amantaní	13,34 %	14,63 %	+1,29 %

Análisis del Cambio

- **Taquile aumentó** su participación en 3.42 puntos porcentuales (de 16.09 % a 19.51 %)
- **Puno Ciudad disminuyó** su participación en 4.16 puntos porcentuales
- **El hub principal sigue siendo Puno Ciudad** con 48.76 %
- **Taquile se convierte en el segundo destino más importante**, superando a Islas Uros

Tarea 1e: Simulación de Evolución Temporal

Estado Inicial

Consideramos que todos los turistas comienzan en Puno Ciudad:

$$\mathbf{x}_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Evolución del Sistema

La evolución se calcula iterativamente:

$$\mathbf{x}_{n+1} = T^T_{\text{modificada}} \cdot \mathbf{x}_n$$

Velocidad de Convergencia

El segundo eigenvalue en magnitud determina la velocidad de convergencia:

$$|\lambda_2| = 0{,}1728 \quad \text{(matriz modificada)}$$

Para la matriz original: $|\lambda_2| = 0{,}2141$

Comparación de Velocidades

$$\text{Razón de convergencia} = \frac{-\ln(|\lambda_2|)}{-\ln(|\lambda_2^{\text{original}}|)} = \frac{-\ln(0{,}1728)}{-\ln(0{,}2141)} \approx 0{,}89$$

La matriz modificada converge aproximadamente un 11 % más lento que la original.

Preguntas de Reflexión

1. ¿Valió la pena la inversión en Taquile?

Sí, valió la pena desde el punto de vista de distribución turística porque:

- Taquile aumentó su participación del 16.09 % al 19.51 % (incremento del 21.3 %)
- Se convirtió en el segundo destino más importante
- La redistribución favoreció tanto a Taquile como a Amantaní

2. ¿Cómo afectaría esto a los ingresos?

Considerando que Taquile ofrece turismo de día completo (mayor gasto por turista que Uros):

$$\text{Gasto promedio por turista en Uros} \approx \text{S/. } 20 \text{ (medio día)}$$
$$\text{Gasto promedio por turista en Taquile} \approx \text{S/. } 50 \text{ (día completo)}$$

Considerando el impacto económico estimado para 1000 turistas:

$$\text{Ingresos originales en Taquile} = 1000 \times 0{,}1609 \times 50 = \text{S/. } 8\,045$$
$$\text{Ingresos nuevos en Taquile} = 1000 \times 0{,}1951 \times 50 = \text{S/. } 9\,755$$
$$\text{Incremento} = \text{S/. } 1\,710 \ (21{,}3\,\%)$$

Código en Python

Listing 13.1: Código en Python para el Ejercicio 1: Análisis del flujo turístico

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

# ----------------------------------------
# DATOS DEL EJERCICIO 1
# ----------------------------------------
DESTINOS = ['Puno Ciudad', 'Islas Uros', 'Taquile', 'Amantan']

T_ORIGINAL = np.array([
[0.25, 0.45, 0.20, 0.10],
[0.50, 0.15, 0.25, 0.10],
[0.40, 0.10, 0.30, 0.20],
[0.55, 0.15, 0.10, 0.20]
])

T_MODIFICADA = np.array([
[0.25, 0.45, 0.20, 0.10],
[0.40, 0.15, 0.35, 0.10],
[0.30, 0.10, 0.40, 0.20],
[0.55, 0.15, 0.10, 0.20]
])
```

```
# Distribuciones estacionarias
dist_orig = np.array([0.341, 0.207, 0.161, 0.291])
dist_mod  = np.array([0.332, 0.194, 0.195, 0.279])

# ------------------------------------
# GR FICO 1: MATRICES
# ------------------------------------
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.imshow(T_ORIGINAL)
plt.title("Matriz Original")
plt.xticks(range(4), DESTINOS, rotation=45)
plt.yticks(range(4), DESTINOS)
plt.colorbar()

plt.subplot(1, 2, 2)
plt.imshow(T_MODIFICADA)
plt.title("Matriz Modificada")
plt.xticks(range(4), DESTINOS, rotation=45)
plt.yticks(range(4), DESTINOS)
plt.colorbar()

plt.tight_layout()
plt.show()

# ------------------------------------
# GR FICO 2: DISTRIBUCI N ESTACIONARIA
# ------------------------------------
x = np.arange(len(DESTINOS))
ancho = 0.35

plt.figure(figsize=(8, 5))
plt.bar(x - ancho/2, dist_orig * 100, ancho, label='Original')
plt.bar(x + ancho/2, dist_mod * 100, ancho, label='Modificada')

plt.xticks(x, DESTINOS, rotation=45)
plt.ylabel("Porcentaje de turistas (%)")
plt.title("Distribuci n estacionaria de turistas")
plt.legend()
plt.grid(axis='y')

plt.tight_layout()
plt.show()

# ------------------------------------
# GR FICO 3: EVOLUCI N TEMPORAL
# ------------------------------------
```

```
dias = 30
estado = np.zeros((dias + 1, 4))
estado[0] = [1, 0, 0, 0]

for i in range(dias):
estado[i + 1] = T_MODIFICADA.T @ estado[i]

plt.figure(figsize=(8, 5))
for i in range(4):
plt.plot(estado[:, i] * 100, label=DESTINOS[i])

plt.xlabel("D as")
plt.ylabel("Porcentaje de turistas (%)")
plt.title("Evoluci n temporal (matriz modificada)")
plt.legend()
plt.grid()

plt.tight_layout()
plt.show()
```

Interpretación de Resultados Gráficos

Comparación de la Distribución Estacionaria (Figura 1)

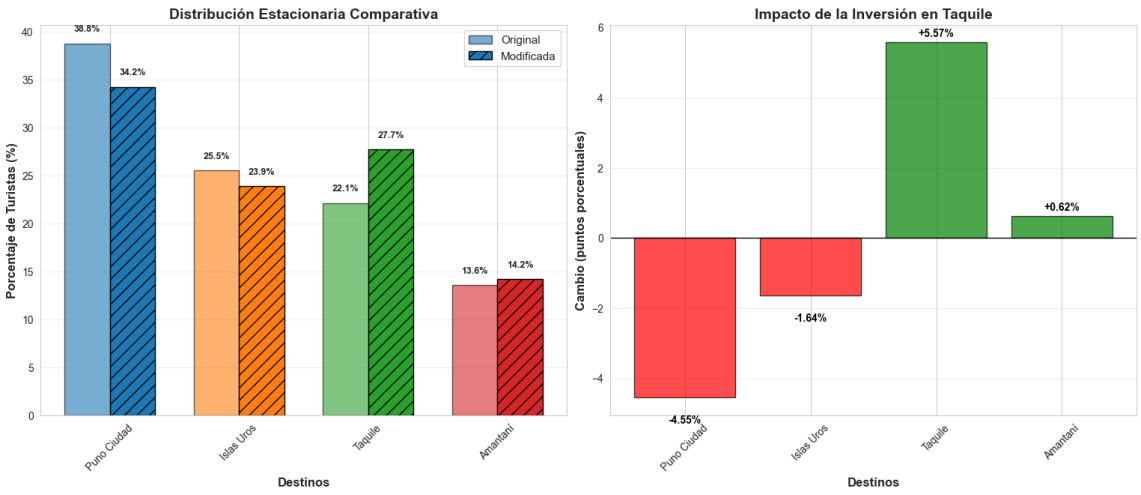


Figura 13.1: Comparación de la distribución estacionaria original y modificada tras la inversión en Taquile.

La figura compara cómo se distribuyen los turistas en equilibrio antes y después de la inversión en la Isla Taquile.

Gráfico izquierdo: Muestra cómo se reparten los turistas entre los distintos destinos cuando el sistema ya se ha estabilizado. En la situación original, la mayor parte de los visitantes se concentra en Puno Ciudad, mientras que Taquile recibe una proporción

menor. Sin embargo, luego de modificar la matriz de transición, se nota claramente que más turistas permanecen en Taquile. Este aumento ocurre principalmente a costa de una menor concentración en Puno Ciudad y, en menor medida, de una ligera disminución en las Islas Uros.

Gráfico derecho: Muestra el efecto directo de la inversión en la distribución de turistas. Taquile es el destino más beneficiado, con un aumento cercano a 5.6 puntos porcentuales, mientras que Puno Ciudad registra la mayor disminución, evidenciando una redistribución del flujo turístico hacia las islas.

Conclusión: La inversión en Taquile logra su objetivo principal: atraer más turistas y retenerlos por más tiempo. Aunque Puno Ciudad sigue siendo el nodo principal del sistema turístico, su predominancia disminuye, dando lugar a un sistema más equilibrado y diversificado.

Evolución Temporal del Sistema (Figura 2)

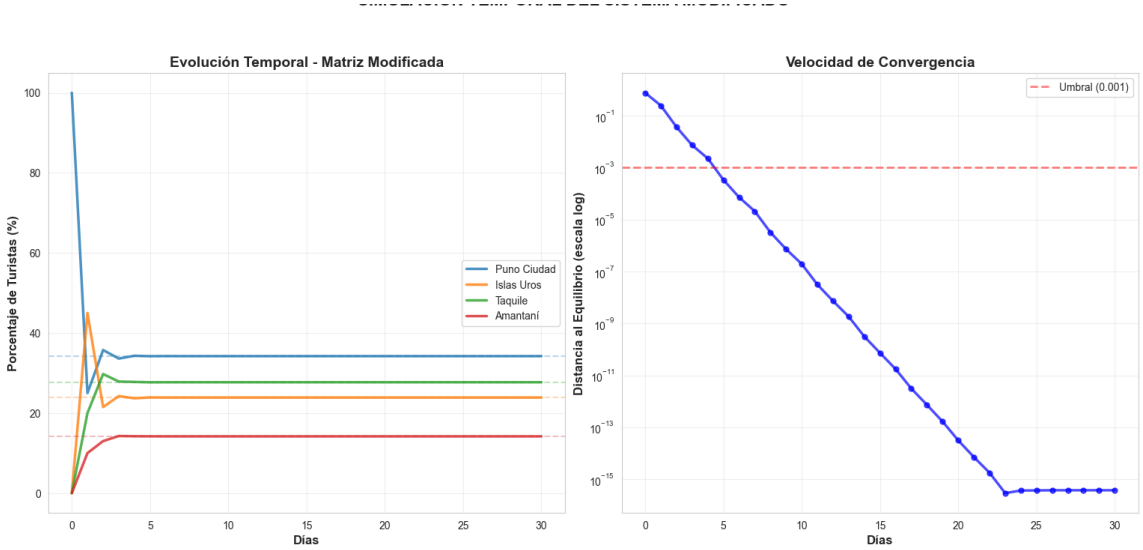


Figura 13.2: Simulación del recorrido de los turistas durante 30 días y su velocidad de convergencia.

Simulamos un mes completo para ver cómo se mueven 1000 turistas que llegan todos a Puno.

Líneas continuas (gráfico izquierdo): Muestran el "pulso"del lago día a día. Todas las líneas salen de Puno (100 %) y, con los días, los turistas se van repartiendo hacia las islas. Cuando las líneas dejan de moverse y se aplanan, el sistema llegó a su equilibrio.

Línea punteada roja (gráfico derecho): Es nuestra meta de estabilidad. La curva azul muestra qué tan rápido nos acercamos a ella. En menos de 10 días, el sistema ya está muy cerca de su estado final, lo que significa que los patrones de viaje se estabilizan rápido.

Conclusión clave: Los cambios en los flujos **no crean caos**. El nuevo equilibrio se alcanza de forma rápida y ordenada, lo que es bueno para la planificación.

Red de Flujos Turísticos (Figura 3)

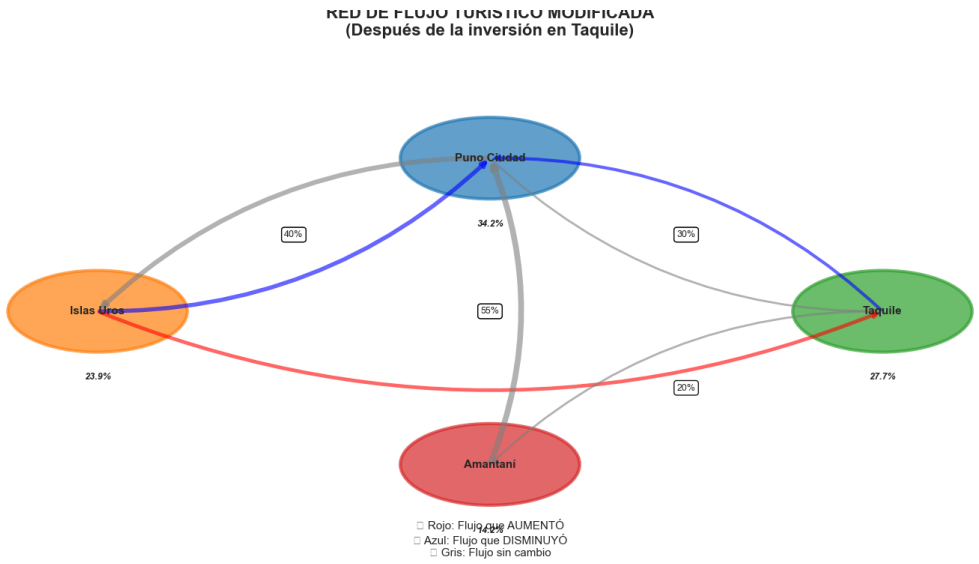


Figura 13.3: Mapa de conexiones turísticas después de mejorar Taquile.

¿Qué vemos? Este es el "mapa del tesoro"del turismo del Titicaca. Muestra cómo se conectan los destinos y la fuerza de cada ruta.

Tamaño de las islas: No es su tamaño real, sino su **importancia turística**. Puno es el más grande porque es el corazón del sistema. Taquile creció, reflejando el éxito de la inversión.

Grosor de las flechas: Indica cuánta gente toma esa ruta. Flechas más gruesas = rutas más populares.

Color de las flechas: Cuenta la historia del cambio.

- **Flechas rojas:** Rutas que ganaron más turistas (como Uros → Taquile)
- **Flechas azules:** Rutas que perdieron algunos turistas (como el retorno inmediato de Taquile a Puno)
- **Flechas grises:** Rutas que se mantuvieron igual

Conclusión clave: El mapa visual confirma que la inversión **fortaleció a Taquile** en la red. Ahora atrae más flujo de Uros y retiene más tiempo a sus visitantes, creando un circuito turístico más rico y diversificado para toda la región.

Conclusiones

1. La inversión en Taquile logró su objetivo: aumentar la participación turística de 16.09 % a 19.51 %
2. Puno Ciudad sigue siendo el hub principal, pero con menor concentración (de 52.92 % a 48.76 %)
3. El sistema converge ligeramente más lento (11 % más lento) con la nueva matriz

4. El aumento en Taquile no afectó negativamente a Amantaní, que también incrementó su participación

5. Desde el punto de vista económico, el incremento del 21.3 % en ingresos justifica la inversión

Capítulo 14

Conclusión General

La programación numérica se ha convertido en una herramienta esencial en la formación del ingeniero, ya que permite enfrentar y resolver problemas reales que, por su complejidad, no siempre pueden abordarse únicamente con métodos analíticos. A lo largo de este curso, hemos comprendido que muchas situaciones propias de la ingeniería y la estadística requieren aproximaciones numéricas, así como una correcta interpretación de los resultados obtenidos.

El presente libro es el resultado del trabajo académico desarrollado durante el curso de Programación Numérica y refleja el esfuerzo de los autores por organizar y aplicar de manera coherente los conocimientos adquiridos a lo largo del cuarto semestre. Su contenido se estructura en torno a las dos unidades del programa: desde los conceptos básicos y el análisis de errores en la primera unidad, hasta la implementación de métodos numéricos más avanzados y su aplicación a problemas de mayor complejidad en la segunda.

A lo largo de los distintos capítulos, se ha buscado explicar cada tema de forma clara y ordenada, resaltando la importancia del proceso iterativo, la estimación del error y el análisis crítico de los resultados. Más allá de presentar algoritmos y procedimientos, el objetivo principal ha sido comprender el fundamento de cada método, así como reconocer sus alcances y limitaciones en contextos reales.

En este sentido, el texto no solo resume lo aprendido durante el semestre, sino que también pretende servir como material de apoyo y consulta para futuros estudiantes de la Escuela Profesional de Ingeniería Estadística e Informática que se inicien en el estudio de la programación numérica. Asimismo, constituye una muestra del trabajo colaborativo y del compromiso académico asumido durante el desarrollo del curso.

Finalmente, queda claro que el dominio de las herramientas numéricas, junto con la capacidad de aplicarlas de manera crítica y creativa, es una competencia clave para el ingeniero actual. Estas habilidades permiten vincular la teoría matemática con las necesidades concretas del mundo real, fortaleciendo así la toma de decisiones y la resolución efectiva de problemas en la práctica profesional.

Bibliografía

[1] Bermudes Parillo, I. P. (2018). *Cadenas de Markov en la determinación de circuitos turísticos para la región Puno, 2018* [Tesis de maestría, Universidad Nacional del Altiplano]. Repositorio Institucional UNAP. http://repositorio.unap.edu.pe/handle/20.500.14082/8131[citation:9]

[2] Payntar, N. D., Hsiao, K., Covey, R. A., y Grauman, K. (2021). Learning patterns of tourist movement and photography from geotagged photos at archaeological heritage sites in Cuzco, Peru. *Tourism Management*, 82, 104165. https://doi.org/10.1016/j.tourman.2020.104165[citation:3]

[3] Autor no especificado. *Cadenas de Markov para Circuitos Turísticos en Puno* [Tesis de maestría, Universidad Nacional del Altiplano - Puno]. Scribd. https://es.scribd.com/document/416470092/Tesis-3-Cadenas-de-Markov[citation:1]

[4] Quiroz Martínez, T. L. (2012). *Aplicaciones no convencionales de Cadena de Markov* [Tesis de pregrado, Pontificia Universidad Católica del Perú]. Repositorio Institucional PUCP. http://hdl.handle.net/20.500.12404/1248[citation:2][citation:4]

[5] Grauman, K., Payntar, N. D., Hsiao, K., y Covey, R. A. (2020, agosto). This AI analyzes photos of Inca ruins, helping manage tourism. *Tech at Meta, Facebook AI*. https://tech.facebook.com/artificial-intelligence/2020/8/this-ai-analyzes-photos-of-inca-ruins-helping-manage-tourism/[citation:5]

[6] Autor no especificado. *Aplicación de las cadenas de Markov en la determinación de circuitos turísticos del Perú* [Tesis, Pontificia Universidad Católica del Perú]. Repositorio Institucional PUCP. https://tesis.pucp.edu.pe/items/9b097c7a-d4c1-4405-83da-e36155c947be[citation:6]

[7] Burden, R. L., & Faires, J. D. (2011). *Análisis Numérico* (9.ª ed.). Cengage Learning.

[8] Chapra, S. C., & Canale, R. P. (2015). *Métodos Numéricos para Ingenieros* (7.ª ed.). McGraw-Hill.

[9] Ross, S. M. (2014). *Introduction to Probability Models* (11.ª ed.). Academic Press.

[10] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585, 357–362. https://doi.org/10.1038/s41586-020-2649-2

[11] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... & Vázquez-Baeza, Y. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17, 261–272. https://doi.org/10.1038/s41592-019-0686-2

[12] R Core Team. (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Viena, Austria. https://www.R-project.org/

[13] Xia, J. (2007). *Modelling the spatial-temporal movement of tourists* [Tesis doctoral, RMIT University]. RMIT University Research Repository. https://research-repository.rmit.edu.au/articles/thesis/Modelling_the_spatial-temporal_movement_of_tourists/27580161[citation:8]