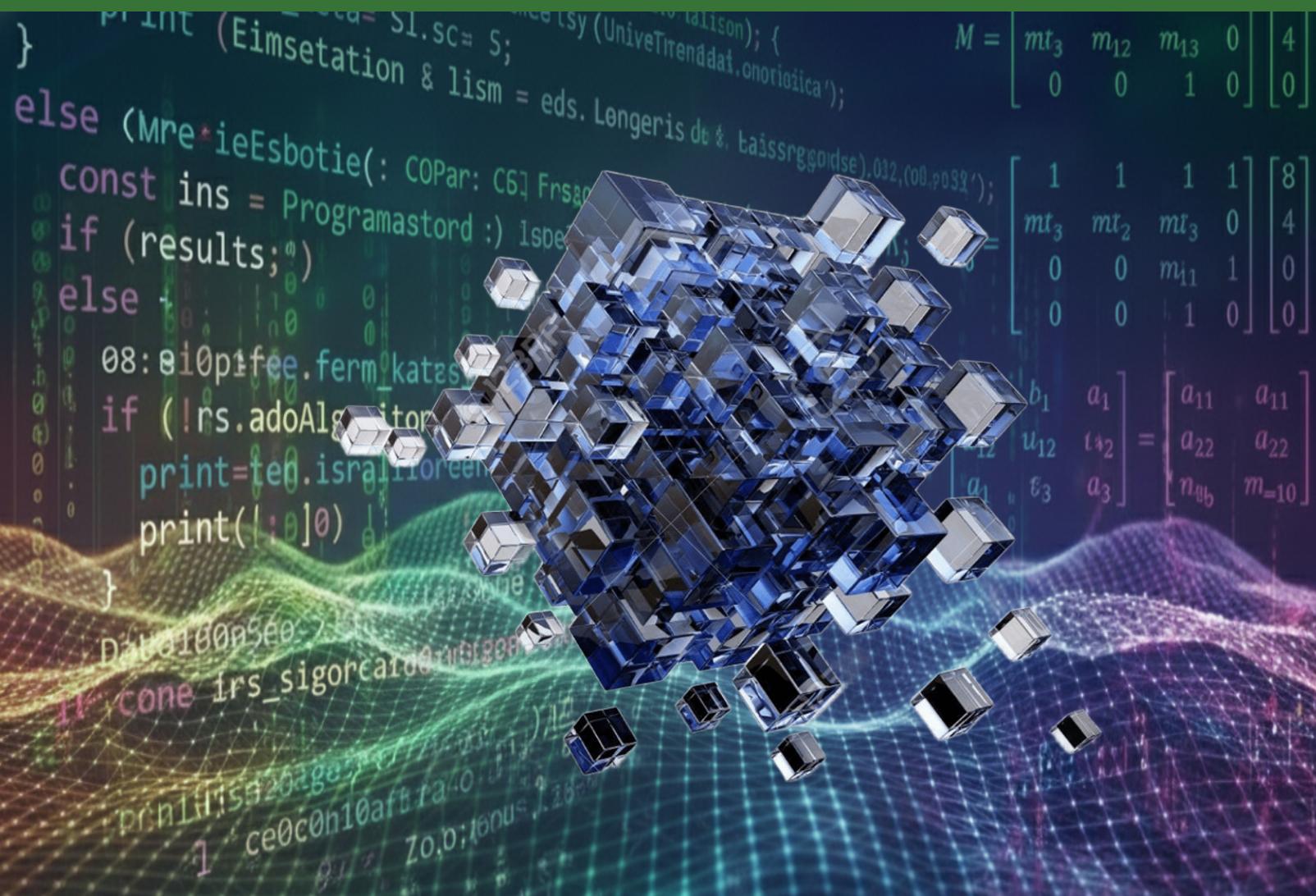


Programación Numérica

Universidad Nacional del Altiplano Puno

Facultad de Ingeniería Estadística e Informática



Estudiantes de la Facultad de Ingeniería
Estadística e Informática - UNA Puno

PROGRAMACIÓN NUMÉRICA

Quispe Ito Luz Leidy
Puma Huanca Anthony Rusbel



Universidad Nacional del Altiplano
Facultad de Ing. Estadística e Informática

PROGRAMACIÓN NUMÉRICA

1.4 Docente:
Dr. Torres Cruz Fred

Alumno:
Quispe Ito Luz Leidy
Puma Huanca Anthony Rusbel

Semestre:
IV Semestre – A



Puno – Perú
2025

Índice general

Introducción	9
1. Programación Numérica	11
1.1. Introducción	11
1.2. Fundamentos de la Programación Numérica	11
1.2.1. De lo Continuo a lo Discreto	11
1.2.2. Objetivos Principales	11
1.3. Aplicaciones Clave	12
1.4. Herramientas y Lenguajes Modernos	12
1.5. Capacidades que Habilita	12
1.6. Conclusión	13
2. Unidad 1	15
3. Reconocimiento de Funciones	17
3.1. ¿Qué es el Reconocimiento de Funciones?	17
3.2. Tipos de Funciones Reconocidas	17
3.3. ¿Cómo Reconocer el Tipo de Función?	18
3.4. Implementación en Python	19
3.5. Ejemplos de Ejecución	21
3.6. Conclusión	22
4. Graficador de Funciones Lineales	25
4.1. Descripción del Proyecto	25
4.2. Objetivo del Graficador	25
4.3. Implementación en Python	26
4.4. Ejemplo de Uso	27
4.5. Limitaciones y Consideraciones	29
4.6. Conclusión	29
5. Restricción de Funciones Lineales	31
5.1. Introducción	31
5.2. Problemas de Aplicación	31
5.3. Implementación en Python	33
5.4. Conclusión	36

6. Métodos Numéricos para Hallar Raíces de Ecuaciones	37
6.1. Introducción	37
6.2. Métodos Iterativos para Raíces	38
6.3. Conclusión	40
7. Método de Bisección	41
7.1. Definición del Método de Bisección	41
7.2. Procedimiento	41
7.3. Ejemplo: Función Exponencial	42
7.4. Implementación en Python	43
7.5. Conclusiones	43
8. Método de Newton-Raphson	45
8.1. Idea General	45
8.2. Derivación del Método	45
8.3. Convergencia	46
8.4. Ventajas y Desventajas	46
8.5. Ejemplo: Cálculo de $\sqrt{5}$	46
8.6. Implementación en Python	47
8.7. Aplicaciones	48
8.8. Conclusión	48
9. Método de la Secante	51
9.1. Introducción	51
9.2. Idea Principal	51
9.3. Pasos del Método	52
9.4. Ventajas y Desventajas	52
9.5. Ejemplo 1: Raíz de $f(x) = x^2 - 4$	52
9.6. Ejemplo 2: Dilatación Térmica de una Varilla	53
9.7. Implementación en Python	53
9.8. Conclusión	54
10. Método del Punto Fijo	55
10.1. Definición	55
10.2. Fundamento Teórico	55
10.3. Condición de Convergencia	55
10.4. Algoritmo	56
10.5. Ejemplo 1: Inversión que se Duplica en un Año	56
10.6. Ejemplo 2: Resolución de $f(x) = e^x - 4 + x = 0$	56
10.7. Implementación en Python	57
10.8. Ventajas y Desventajas	58
10.9. Conclusión	58
11. Método de Regula Falsi	59
11.1. Descripción General	59
11.2. Diferencia con el Método de Bisección	59
11.3. Fórmula de Interpolación	59
11.4. Algoritmo	60
11.5. Características	60

11.6. Ejemplo 1: $f(x) = x^2 - 4$	60
11.7. Ejemplo 2: $f(x) = x^3 - 2x - 5$	61
11.8. Implementación en Python	61
11.9. Conclusión	62
12. Gradiente	63
12.1. Definición	63
12.2. Gradiente de una Función	63
12.3. Interpretación geométrica	64
12.4. Derivada direccional	64
12.5. Propiedades fundamentales	64
12.6. Gradiente y optimización	65
12.7. Aplicaciones	65
12.8. Implementación en R para Gradiente de una función	65
12.8.1. Método del Descenso del Gradiente	65
12.8.2. Método del Gradiente para dos variables	68
12.9. Convergencia del Método del Gradiente	71
12.10. Variantes del Gradiente Descendente	72
12.11. Consideraciones Prácticas	72
13. Diferenciación Numérica	73
13.1. Introducción	73
13.2. Fundamentos Teóricos	73
13.2.1. Definición de Derivada	73
13.2.2. El Parámetro h	73
13.3. Métodos de Diferencias Finitas	73
13.3.1. Diferencia Hacia Adelante	73
13.3.2. Diferencia Hacia Atrás	74
13.3.3. Diferencia Centrada	74
13.4. Implementación en R	74
13.4.1. Funciones Básicas de Diferenciación	74
13.5. Ejercicios Prácticos	75
13.5.1. Ejercicio : Análisis de Crecimiento de Usuarios	75
13.6. Tabla Comparativa de Métodos	77
13.7. Consideraciones Prácticas	77
13.7.1. ¿Cómo Elegir h ?	77
13.7.2. Recomendaciones prácticas:	77
13.8. Conclusión	78
14. Interpolación Numérica	79
14.1. Interpolación Lineal	80
14.2. Interpolación Polinómica	80
14.2.1. Forma de Lagrange	80
14.2.2. Ejemplo	81
14.3. Interpolación de Newton	81
14.3.1. Diferencias Divididas	81
14.4. Implementación en R	82
14.5. Interpolación Cuadrática	82
14.6. Interpolación por Tramos	83

14.6.1. Splines Cúbicos	83
14.7. Errores de Interpolación	84
14.8. Interpolación por Tramos	84
14.9. Conclusiones	84
15. Lighthouse y Apache JMeter	85
15.1. Introducción a las Pruebas de Rendimiento	85
15.2. Google Lighthouse	85
15.2.1. ¿Qué es Lighthouse?	85
15.2.2. ¿Para qué sirve Lighthouse?	86
15.2.3. Métricas Clave de Lighthouse	86
15.2.4. Cómo usar Lighthouse	86
15.2.5. Interpretación de Resultados	87
15.3. Apache JMeter	87
15.3.1. ¿Qué es Apache JMeter?	88
15.3.2. Características Principales	88
15.3.3. Arquitectura de JMeter	88
15.3.4. Configuración Básica de una Prueba	88
15.3.5. Métricas Clave en JMeter	90
15.3.6. Ejecución de Pruebas desde Línea de Comandos	90
15.4. Comparación: Lighthouse vs JMeter	90
15.5. Ejercicios Prácticos	90
15.5.1. Ejercicio 1: Auditoría con Lighthouse	90
15.5.2. Ejercicio 2: Prueba de Carga con JMeter	91
15.6. Análisis de Resultados	92
15.6.1. Interpretación de Métricas	93
15.7. Casos de Uso en la Industria	95
15.7.1. E-commerce	95
15.7.2. Aplicaciones Financieras	95
15.7.3. Plataformas de Streaming	96
15.8. Conclusión	96

Introducción

En la ingeniería moderna y las ciencias aplicadas, la capacidad de modelar el mundo real depende cada vez menos de soluciones analíticas cerradas y más de la potencia del cálculo computacional. La **Programación Numérica** surge como el puente indispensable entre la teoría matemática abstracta y la resolución práctica de problemas complejos, permitiendo a ingenieros y científicos abordar ecuaciones que serían imposibles de resolver con lápiz y papel.

Este texto ha sido diseñado no solo como un manual de algoritmos, sino como una guía integral para la implementación de métodos numéricos utilizando lenguajes de vanguardia como **Python** y **R**. El objetivo central es dotar al lector de una “caja de herramientas” algorítmica que le permita enfrentar desafíos en campos tan diversos como la física computacional, la economía cuantitativa y el análisis de datos.

La estructura del libro se divide en dos grandes bloques temáticos:

En la **Primera Parte (Fundamentos y Raíces)**, nos enfrentamos al problema clásico de encontrar ceros en funciones no lineales. Lejos de ser una tarea trivial, este problema es la base de simulaciones complejas. Analizaremos la robustez del **Método de Bisección**, la velocidad del **Método de la Secante** y la elegancia del **Método de Newton-Raphson**, desglosando no solo su código, sino también sus criterios de convergencia y eficiencia computacional.

La **Segunda Parte (Aproximación y Optimización)** eleva el nivel de abstracción. Aquí estudiaremos cómo construir funciones a partir de datos dispersos mediante la **Interpolación Polinómica** y cómo modelar tendencias mediante técnicas de **Diferenciación Numérica**. Un punto clave de esta sección es el estudio del **Gradiente Descendente**, algoritmo angular en la optimización moderna y el aprendizaje automático (*Machine Learning*).

Más allá de la sintaxis del código, este libro hace énfasis en el pensamiento crítico: ¿Cómo validamos una solución numérica? ¿Qué impacto tiene el error de redondeo de la computadora? A través de la experimentación práctica, buscamos que el estudiante no solo ejecute scripts, sino que desarrolle la intuición necesaria para discernir entre un resultado matemáticamente válido y un artefacto numérico.

CAPÍTULO 1

Programación Numérica

1.1 Introducción

La **Programación Numérica** es una disciplina híbrida que fusiona las matemáticas aplicadas con la informática para resolver problemas cuantitativos mediante algoritmos computacionales. A diferencia de la programación tradicional —orientada a interfaces, bases de datos o flujos de negocio—, la programación numérica se enfoca en obtener soluciones *aproximadas pero útiles* a modelos matemáticos que, en la mayoría de los casos, carecen de soluciones analíticas cerradas (Burden & Faires, 2016; Chapra & Canale, 2015).

Este enfoque es esencial en campos donde la complejidad del fenómeno supera la capacidad del cálculo simbólico: desde la simulación de fluidos en ingeniería aeroespacial hasta el ajuste de modelos epidemiológicos en salud pública.

1.2 Fundamentos de la Programación Numérica

1.2.1 De lo Continuo a lo Discreto

El núcleo de la programación numérica radica en la **discretización**: transformar problemas continuos —como integrales, derivadas o ecuaciones diferenciales— en representaciones finitas que una computadora puede procesar. Este paso introduce necesariamente errores, pero con técnicas adecuadas, estos pueden controlarse y acotarse rigurosamente.

1.2.2 Objetivos Principales

Los algoritmos numéricos buscan equilibrar tres pilares fundamentales:

- **Precisión:** minimizar el error entre la solución aproximada y la verdadera.
- **Estabilidad:** evitar que pequeños errores en los datos o en los cálculos se amplifiquen descontroladamente.
- **Eficiencia:** resolver el problema en tiempo y memoria razonables, especialmente en aplicaciones a gran escala.

Apunte

Reflexión crítica: Un algoritmo “rápido” no siempre es mejor. En contextos científicos, un método ligeramente más lento pero estable y preciso suele ser preferible a uno rápido pero caótico.

1.3 Aplicaciones Clave

La programación numérica permea múltiples áreas del conocimiento. Algunas de sus aplicaciones más destacadas incluyen:

- **Simulación científica:** modelado de fenómenos físicos, biológicos o climáticos.
- **Ingeniería de datos:** interpolación, suavizado y derivación de señales ruidosas.
- **Economía y finanzas:** optimización de portafolios y valoración de derivados.
- **Inteligencia artificial:** cálculo de gradientes en redes neuronales y ajuste de hiperparámetros.
- **Ciencias de la salud:** modelado de dinámicas de enfermedades o dosificación farmacológica.

1.4 Herramientas y Lenguajes Modernos

Hoy en día, lenguajes como **Python**, **R**, **MATLAB** y **Julia** ofrecen ecosistemas maduros para el desarrollo numérico:

- En **Python**: NumPy, SciPy y Autograd permiten implementar métodos avanzados con pocas líneas de código.
- En **R**: paquetes como pracma y numDeriv facilitan la diferenciación numérica y la optimización.
- En **Julia**: su diseño nativo para computación científica permite alcanzar rendimientos cercanos al de C con sintaxis expresiva.

Estos entornos no solo aceleran el prototipado, sino que también integran visualización, análisis estadístico y generación de reportes, haciendo de la programación numérica una herramienta accesible tanto para investigadores como para docentes.

1.5 Capacidades que Habilita

Gracias a la programación numérica, un profesional puede:

- Resolver ecuaciones no lineales mediante métodos iterativos (ej. Newton-Raphson).
- Aproximar derivadas e integrales cuando no se conoce la forma analítica de la función.

- Interpolar valores entre puntos de datos experimentales o ajustar modelos mediante mínimos cuadrados.
- Optimizar funciones de una o varias variables, incluso con restricciones complejas.
- Analizar la propagación de errores y evaluar la robustez de los resultados obtenidos.

Apunte

Consejo práctico: Siempre valida tus resultados numéricos contra casos conocidos (problemas con solución analítica) antes de aplicarlos a datos reales.

1.6 Conclusión

La programación numérica no es simplemente “programar matemáticas”; es una forma de *pensar computacionalmente* sobre problemas del mundo real. Al integrar rigor matemático, conciencia del error y eficiencia algorítmica, se convierte en un pilar indispensable de la ciencia y la ingeniería modernas.

Apunte

Para profundizar: Los fundamentos de la programación numérica son la base sobre la que se construyen técnicas avanzadas como el aprendizaje automático, la computación cuántica y la simulación basada en agentes.

CAPÍTULO 2

Unidad 1

CAPÍTULO 3

Reconocimiento de Funciones

3.1 ¿Qué es el Reconocimiento de Funciones?

El **reconocimiento de funciones matemáticas** es un proceso analítico que permite identificar y clasificar expresiones algebraicas según sus características estructurales. Este análisis automatizado examina los componentes de una función para determinar su naturaleza, grado y propiedades fundamentales.

A través de un programa computacional, es posible extraer información clave de cualquier expresión matemática, incluyendo:

- **Variables presentes:** identificación de todas las letras que representan valores desconocidos.
- **Operaciones matemáticas:** conteo de operadores (+, -, *, /, ^).
- **Constantes numéricas:** números que forman parte de la expresión.
- **Grado de la función:** mayor exponente presente en la expresión.
- **Clasificación automática:** identificación del tipo de función (constante, lineal, cuadrática, cúbica, etc.).

Apunte

Aplicaciones prácticas: Este tipo de análisis es fundamental en sistemas de álgebra computacional, calculadoras científicas avanzadas, software educativo y herramientas de visualización matemática, permitiendo procesar y clasificar expresiones de forma automática.

3.2 Tipos de Funciones Reconocidas

El analizador categoriza las funciones según su grado, siguiendo la estructura de polinomios estándar. A continuación se describen los tipos reconocidos:

Función Constante (Grado 0)

$$f(x) = c$$

Una función constante no contiene variables o todas se cancelan, resultando en un valor fijo. Su gráfica es una línea horizontal.

Ejemplos: $f(x) = 5$, $f(x) = -3,14$, $f(x) = 0$

Función Lineal (Grado 1)

$$f(x) = mx + b$$

La variable aparece elevada a la primera potencia. Su gráfica es una recta con pendiente m y ordenada al origen b .

Ejemplos: $f(x) = 2x + 3$, $f(x) = -5x + 1$, $f(x) = x$

Función Cuadrática (Grado 2)

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

Su gráfica es una parábola. La concavidad depende del signo de a .

Ejemplos: $f(x) = x^2 + 2x - 3$, $f(x) = -2x^2 + 5$, $f(x) = 3x^2$

Función Cúbica (Grado 3)

$$f(x) = ax^3 + bx^2 + cx + d$$

Puede tener hasta dos puntos de inflexión y un comportamiento más complejo que funciones de grado menor.

Ejemplos: $f(x) = x^3 + 2x^2 - 5x + 7$, $f(x) = -x^3 + 3x$, $f(x) = 2x^3$

Polinomios de Grado Superior (Grado $n > 3$)

$$f(x) = a_n x^n + \cdots + a_1 x + a_0$$

Presentan múltiples puntos críticos y cambios de dirección a medida que aumenta el grado.

Ejemplos: $f(x) = x^4 - 2x^2 + 1$, $f(x) = x^5 + 3x^3 - x$, $f(x) = 2x^6$

3.3 ¿Cómo Reconocer el Tipo de Función?

El proceso de reconocimiento sigue un flujo sistemático:

1. **Identificar las variables:** Buscar todas las letras (por ejemplo, x, y, z) que representan incógnitas.
2. **Determinar los exponentes:** Localizar los exponentes asociados a cada variable. Si no hay exponente explícito, se asume potencia 1. El símbolo \hat{x} representa exponenciación (ej. $x\hat{2} = x^2$).

3. **Encontrar el grado máximo:** El grado de la función es el mayor exponente presente en la expresión.

4. **Clasificar la función:** Según el grado obtenido:

- Grado 0 → Constante
- Grado 1 → Lineal
- Grado 2 → Cuadrática
- Grado 3 → Cúbica
- Grado > 3 → Polinómica de grado superior

3.4 Implementación en Python

El analizador se implementa mediante programación orientada a objetos. La clase Funcion encapsula los métodos necesarios para procesar una expresión algebraica en formato de texto.

Apunte

Consejo de programación: La expresión se normaliza al eliminar espacios, y se analiza carácter por carácter para detectar variables, operadores y exponentes.

```

1  class Funcion:
2      def __init__(self, expresion):
3          # Inicializar atributos de la función
4          self.expresion = expresion.replace(" ", "")
5          self.variables = []
6          self.operaciones = 0
7          self.grado = 0
8          self.constantes = 0
9
10     def analizar(self):
11         # Método principal que analiza la expresión
12         exp = self.expresion
13         for i in range(len(exp)):
14             c = exp[i]
15             # Identificar variables (letras)
16             if c.isalpha() and c not in self.variables:
17                 self.variables.append(c)
18             # Contar operaciones
19             if c in "+-*/^":
20                 self.operaciones += 1
21             # Contar constantes numéricas
22             if c.isdigit():
23                 self.constantes += 1
24             # Calcular el grado de la función

```

```

25     self.grado = self.calcular_grado()
26
27 def calcular_grado(self):
28     # Determinar el mayor exponente en la expresión
29     exp = self.expresion
30     grado_max = 0
31     i = 0
32     while i < len(exp):
33         if exp[i].isalpha():
34             # Verificar si hay un exponente explícito (^)
35             if i + 1 < len(exp) and exp[i + 1] == '^':
36                 j = i + 2
37                 num = ''
38                 # Extraer el valor del exponente
39                 while j < len(exp) and exp[j].isdigit():
40                     num += exp[j]
41                     j += 1
42                 if num != '':
43                     grado = int(num)
44                     if grado > grado_max:
45                         grado_max = grado
46             else:
47                 # Si no hay exponente, el grado es 1
48                 if grado_max < 1:
49                     grado_max = 1
50             i += 1
51     return grado_max
52
53 def resumen(self):
54     # Generar un resumen formateado del análisis
55     texto = "=====\\n"
56     texto += f"Función ingresada: {self.expresion}\\n"
57     texto += f"Variables encontradas: {', '.join(self.variables)} if self.
variables else 'ninguna'}\\n"
58     texto += f"Número de variables: {len(self.variables)}\\n"
59     texto += f"Número de operaciones: {self.operaciones}\\n"
60     texto += f"Número de constantes: {self.constantnes}\\n"
61     texto += f"Grado de la función: {self.grado}\\n"
62
63     # Clasificar el tipo de función según su grado
64     if self.grado == 1:
65         tipo = "Función lineal"
66     elif self.grado == 2:
67         tipo = "Función cuadrática"
68     elif self.grado == 3:
69         tipo = "Función cúbica"
70     elif self.grado > 3:
71         tipo = "Función polinómica de grado mayor"
72     else:
73         tipo = "Constante o sin variable"
74

```

```

75     texto += f"Tipo de funcin: {tipo}\n"
76     texto += "===="
77     return texto
78
79
80 def main():
81     # Funcin principal del programa
82     expresion = input("Ingresa la funcin (por ejemplo x^3+2x^2-5x+7): ")
83     f = Funcion(expresion)
84     f.analizar()
85     print(f.resumen())
86
87
88 if __name__ == "__main__":
89     main()

```

Listing 3.1: Clase Funcion en Python

3.5 Ejemplos de Ejecución

Ejemplo 1: Función Cúbica

Entrada: x^3+2x^2-5x+7

```

=====
Función ingresada: x^3+2x^2-5x+7
Variables encontradas: x
Número de variables: 1
Número de operaciones: 3
Número de constantes: 5
Grado de la función: 3
Tipo de función: Función cúbica
=====
```

Apunte

Interpretación: Se identificó una variable (x), tres operadores ($+, +, -$), cinco dígitos (3, 2, 2, 5, 7), y el mayor exponente es 3, lo que confirma que se trata de una función cúbica.

Ejemplo 2: Función Lineal

Entrada: $3x+5$

```

=====
Función ingresada: 3x+5
Variables encontradas: x
Número de variables: 1
```

Número de operaciones: 1
 Número de constantes: 2
 Grado de la función: 1
 Tipo de función: Función lineal
 =====

Ejemplo 3: Función Cuadrática

Entrada: x^2-4x+4

=====
 Función ingresada: x^2-4x+4
 Variables encontradas: x
 Número de variables: 1
 Número de operaciones: 2
 Número de constantes: 4
 Grado de la función: 2
 Tipo de función: Función cuadrática
 =====

Ejemplo 4: Polinomio de Grado Superior

Entrada: $2x^5+x^3-7x+1$

=====
 Función ingresada: $2x^5+x^3-7x+1$
 Variables encontradas: x
 Número de variables: 1
 Número de operaciones: 3
 Número de constantes: 5
 Grado de la función: 5
 Tipo de función: Función polinómica de grado mayor
 =====

3.6 Conclusión

El reconocimiento automático de funciones es una herramienta poderosa que combina el análisis simbólico con la programación. Al identificar variables, operadores y el grado de una expresión, el sistema permite una clasificación inmediata que sirve como base para tareas más avanzadas: derivación simbólica, integración numérica, optimización o visualización gráfica.

Este enfoque refuerza la idea central de la programación numérica: transformar problemas matemáticos en algoritmos ejecutables, manteniendo un equilibrio entre simplicidad, precisión y utilidad pedagógica.

Apunte

Para profundizar: Este tipo de analizador puede extenderse para reconocer funciones no polinómicas (trigonométricas, exponenciales, logarítmicas) mediante técnicas de procesamiento de lenguaje natural o gramáticas formales.

CAPÍTULO 4

Graficador de Funciones Lineales

4.1 Descripción del Proyecto

Una **función** es una relación matemática en la que a cada valor de la variable independiente x (dominio) le corresponde exactamente un valor de la variable dependiente y (codominio), expresada habitualmente como:

$$y = f(x)$$

Las **funciones lineales** son un caso especial cuya gráfica en el plano cartesiano es una línea recta. Su forma algebraica general es:

$$f(x) = mx + b$$

donde:

- **m (pendiente):** indica la inclinación de la recta.
 - Si $m > 0$, la recta *sube* de izquierda a derecha.
 - Si $m < 0$, la recta *baja* de izquierda a derecha.
 - Si $m = 0$, la recta es *horizontal*.
- **b (ordenada al origen):** valor de y cuando $x = 0$; es el punto donde la recta corta el eje Y .

4.2 Objetivo del Graficador

Este programa, desarrollado en **Python**, tiene como propósito reconocer y visualizar **dos funciones lineales distintas** en un plano cartesiano utilizando únicamente **caracteres ASCII**, sin dependencias externas como `matplotlib` o `numpy`.

Características clave de la implementación:

- **Interfaz de consola:** entrada y salida directas en la terminal, sin entornos gráficos.
- **Plano limitado:** el rango de visualización es $x, y \in [-15, 15]$, lo que asegura una representación clara en texto.

- **Implementación pura:** el código utiliza únicamente funcionalidades nativas de Python.
- **Identificación visual:** distintos símbolos permiten diferenciar cada función y su punto de intersección.

Apunte

Propósito pedagógico: Esta herramienta ayuda a comprender conceptos fundamentales como pendiente, ordenada al origen, paralelismo e intersección de rectas, incluso en entornos sin capacidad gráfica avanzada.

4.3 Implementación en Python

El programa sigue un flujo simple: solicita dos expresiones lineales (por ejemplo, $2x+1$ o $-x+3$), extrae sus coeficientes m y b , y luego genera una cuadrícula textual donde cada punto se representa con un carácter diferenciado.

Apunte

Nota técnica: La función de análisis asume que la expresión está en formato estándar (sin espacios, potencias o fracciones), y redondea los valores de y para alinearlos a la cuadrícula entera del plano ASCII.

```

1 def graficar_lineas_ascii():
2     try:
3         # Solicitar las funciones al usuario
4         ecuacion1 = input("Ingrese la primera función (ejemplo: 2x+1): ").replace(" ", "")
5         ecuacion2 = input("Ingrese la segunda función (ejemplo: -x+3): ").replace(" ", "")
6
7         def get_mb(eq):
8             # Extraer pendiente (m) y ordenada (b) de la ecuación
9             if 'x' not in eq:
10                 return 0, float(eq)
11
12             m_str = eq.split('x')[0]
13             m = 1 if m_str in ('', '+') else -1 if m_str == '-' else float(m_
14             str)
15
16             # Calcular b (ordenada al origen)
17             b_str = eq.split('x')[1] if len(eq.split('x')) > 1 else ''

```

```

18     b = float(b_str) if b_str else 0
19
20     return m, b
21
22     # Obtener parametros de ambas funciones
23     m1, b1 = get_mb(ecuacion1)
24     m2, b2 = get_mb(ecuacion2)
25
26     # Mostrar leyenda de smbolos
27     print("\nLeyenda:\n* = F1\no = F2\n# = Interseccin\n| = Eje Y\n- = Eje X\n+ = Origen")
28
29     # Generar el plano cartesiano
30     for y in range(15, -16, -1):
31         linea = ""
32         for x in range(-15, 16):
33             # Verificar si el punto pertenece a cada funcin
34             on_f1 = round(m1 * x + b1) == y
35             on_f2 = round(m2 * x + b2) == y
36
37             # Asignar smbolo segn la posicin
38             if on_f1 and on_f2:
39                 linea += "#" # Interseccin
40             elif on_f1:
41                 linea += "*" # Funcin 1
42             elif on_f2:
43                 linea += "o" # Funcin 2
44             elif x == 0 and y == 0:
45                 linea += "+" # Origen
46             elif x == 0:
47                 linea += "|" # Eje Y
48             elif y == 0:
49                 linea += "-" # Eje X
50             else:
51                 linea += " " # Espacio vaco
52         print(linea)
53
54     except (ValueError, IndexError):
55         print("\nError: La ecuacin ingresada no es vlida.")
56
57     # Ejecutar el programa
58     graficar_lineas_ascii()

```

Listing 4.1: Graficador de funciones lineales en Python

4.4 Ejemplo de Uso

Entrada del Usuario

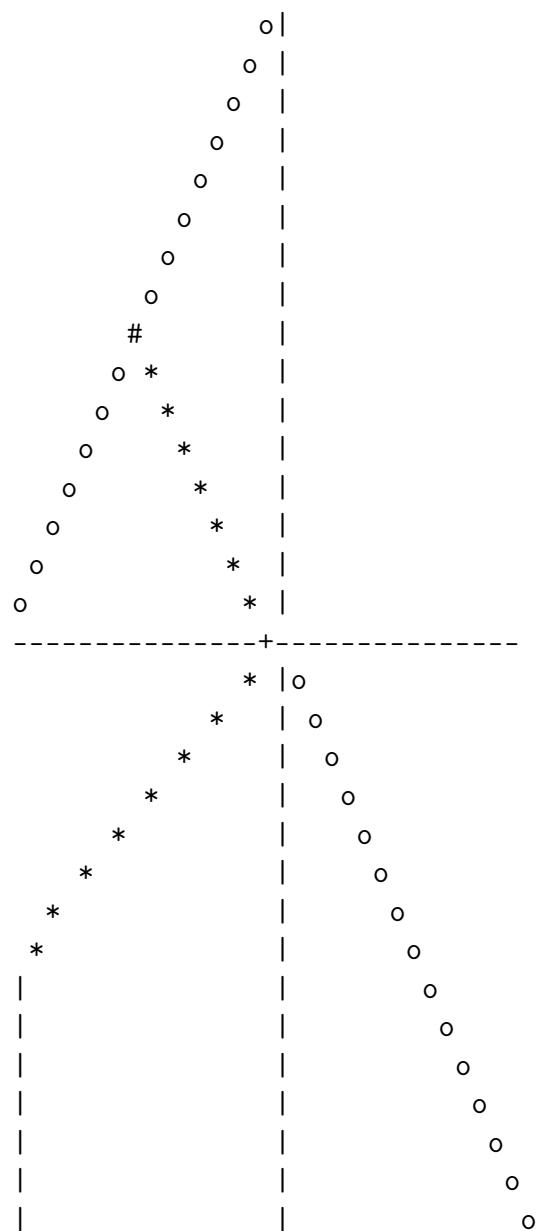
- Primera funcin: $2x+1$

- Segunda función: $-x+3$

Salida en Consola

Leyenda:

* = F1
o = F2
= Intersección
| = Eje Y
- = Eje X
+ = Origen



Apunte

Interpretación del resultado: La función $f_1(x) = 2x + 1$ (pendiente positiva) se representa con asteriscos (*), mientras que $f_2(x) = -x + 3$ (pendiente negativa) con círculos (o). Ambas se intersectan en el punto marcado con #, cuya coordenada puede calcularse algebraicamente como $x = \frac{b_2 - b_1}{m_1 - m_2} = \frac{3 - 1}{2 - (-1)} = \frac{2}{3} \approx 1$, lo que justifica la posición aproximada en la cuadrícula.

4.5 Limitaciones y Consideraciones

El programa, por su naturaleza educativa y textual, presenta algunas restricciones:

- Solo admite funciones lineales de la forma $y = mx + b$.
- No interpreta expresiones con exponentes, fracciones, paréntesis ni términos no lineales.
- El punto de intersección se **redondea** al entero más cercano para ajustarse a la cuadrícula.
- El rango visual está restringido a $[-15, 15]$ en ambos ejes para mantener la legibilidad.

4.6 Conclusión

Este graficador ASCII demuestra cómo la programación numérica puede emplearse para representar conceptos matemáticos fundamentales con recursos mínimos. Al evitar librerías externas y operar directamente con texto, refuerza la comprensión algorítmica de la evaluación de funciones, la geometría analítica y el manejo de coordenadas.

Además, sirve como base para extensiones más avanzadas: inclusión de funciones cuadráticas, cálculo exacto de intersecciones, o incluso exportación a formatos gráficos.

Apunte

Reflexión final: En un mundo dominado por visualizaciones complejas, herramientas simples como esta recuerdan que la esencia de la programación numérica radica en transformar ideas matemáticas en procesos computables, paso a paso, carácter a carácter.

CAPÍTULO 5

Restricción de Funciones Lineales

5.1 Introducción

La **programación lineal** es una técnica matemática fundamental para la toma de decisiones en contextos con recursos limitados. Consiste en optimizar (maximizar o minimizar) una función objetivo lineal sujeta a un conjunto de restricciones también lineales.

En esta actividad se formulan y analizan cinco problemas reales del ámbito tecnológico y empresarial. Para cada uno, se definen variables, se establecen restricciones, y se interpreta la región factible —es decir, el conjunto de soluciones que cumplen simultáneamente todas las condiciones impuestas— mediante métodos gráficos.

Apunte

Propósito educativo: Estos ejercicios refuerzan la conexión entre álgebra, geometría y toma de decisiones, y sirven como base para métodos computacionales más avanzados (como el algoritmo simplex o solvers numéricos).

5.2 Problemas de Aplicación

Problema 1: Desarrollo Front-end y Back-end

Un desarrollador dispone de 15 horas semanales para trabajar en front-end y back-end. Debe dedicar al menos 5 horas al front-end para cumplir con entregables.

■ **Variables:**

- x : horas en front-end
- y : horas en back-end

■ **Restricciones:**

$$\begin{array}{ll} x + y \leq 15 & \text{(tiempo total)} \\ x \geq 5 & \text{(mínimo front-end)} \\ x, y \geq 0 & \text{(no negatividad)} \end{array}$$

- **Región factible:** Área del plano limitada por $x = 5$ (vertical), $x + y = 15$ (recta descendente) y los ejes. Ejemplos de soluciones: $(5, 10)$, $(8, 7)$, $(10, 5)$.

Problema 2: Administración de Servidores

Un ingeniero gestiona servidores en la nube: el tipo A cuesta S/3 por hora y el tipo B S/5 por hora. El presupuesto semanal no debe exceder S/20.

■ **Variables:**

- x : horas del servidor A
- y : horas del servidor B

■ **Restricciones:**

$$\begin{array}{ll} 3x + 5y \leq 20 & \text{(presupuesto)} \\ x, y \geq 0 & \text{(no negatividad)} \end{array}$$

- **Interpretación:** La región factible incluye combinaciones como $(0, 4)$, $(5, 1)$ o $(2, 2,8)$, aunque en la práctica se preferirían valores enteros.

Problema 3: Gestión de Proyectos

Un administrador de proyectos debe dedicar al menos 4 horas a reuniones y 6 a documentación, sin superar las 12 horas semanales.

■ **Variables:**

- x : horas en reuniones
- y : horas en documentación

■ **Restricciones:**

$$\begin{array}{l} x \geq 4 \\ y \geq 6 \\ x + y \leq 12 \\ x, y \geq 0 \end{array}$$

- **Región factible:** Un triángulo pequeño con vértices en $(4, 6)$, $(4, 8)$ y $(6, 6)$. Soluciones viables: $(5, 7)$, $(6, 6)$.

Problema 4: Desarrollo de Videojuegos

Una empresa produce modelos 3D (2h cada uno) y texturas (3h cada una), con un total de 18 horas disponibles por semana.

- **Variables:**

- x : número de modelos 3D
- y : número de texturas

- **Restricciones:**

$$\begin{array}{ll} 2x + 3y \leq 18 & \text{(tiempo disponible)} \\ x, y \geq 0 & \\ x, y \in Z & \text{(unidades enteras)} \end{array}$$

- **Ejemplo:** Si $y = 3$ (3 texturas), entonces $2x \leq 9 \Rightarrow x \leq 4,5$, por lo que se pueden producir hasta 4 modelos.

Problema 5: Startup de Hardware

Una startup fabrica dispositivos tipo A (5 componentes, ganancia S/200) y tipo B (10 componentes, ganancia S/350), con un máximo de 50 componentes disponibles.

- **Variables:**

- x : dispositivos tipo A
- y : dispositivos tipo B

- **Restricciones:**

$$\begin{array}{ll} 5x + 10y \leq 50 & \Rightarrow x + 2y \leq 10 \\ x, y \geq 0 & \end{array}$$

- **Función objetivo (a maximizar):**

$$Z = 200x + 350y$$

- **Soluciones factibles:** $(5, 2) \rightarrow 25 + 20 = 45$ componentes (válido); $(5, 3) \rightarrow 25 + 30 = 55$ (inválido).

5.3 Implementación en Python

El siguiente programa permite visualizar gráficamente dos restricciones lineales en la consola mediante caracteres ASCII. Aunque no resuelve el problema de optimización completo, facilita la identificación visual de la intersección entre rectas y la orientación de las regiones.

Apunte

Limitación: El código asume restricciones de igualdad para la visualización, pero en la práctica las restricciones son desigualdades. La región factible real se encuentra en un lado de cada recta.

```

1 def graficar_lineas_ascii():
2     try:
3         restric1 = input("Ingrese la primera restricción (ej: x+y<=15): ").replace(" ", "")
4         restric2 = input("Ingrese la segunda restricción (ej: 3x+5y<=20): ").replace(" ", "")
5
6         def parse_restric(eq):
7             # Detectar vertical (x=...)
8             if eq.startswith("x<=") or eq.startswith("x>=") or eq.startswith("x="):
9                 num = float(eq.split("=")[1])
10                return "vertical", num
11
12            # Detectar horizontal (y=...)
13            if eq.startswith("y<=") or eq.startswith("y>=") or eq.startswith("y="):
14                num = float(eq.split("=")[1])
15                return "horizontal", num
16
17            # Caso general ax + by + c = 0
18            if 'x' not in eq and 'y' not in eq:
19                raise ValueError("Ecuación inválida.")
20
21            # Separar en lados izquierdo y derecho
22            if "<=" in eq:
23                lhs, rhs = eq.split("<=")
24            elif ">=" in eq:
25                lhs, rhs = eq.split(">=")
26            elif "=" in eq:
27                lhs, rhs = eq.split("=")
28            else:
29                raise ValueError("Restricción no válida.")
30
31            rhs = float(rhs)
32            a, b = 0, 0
33            for part in lhs.replace("-", "+-").split("+"):
34                if part == "":
35                    continue
36                if "x" in part:
37                    coef = part.replace("x", "")
38                    a = 1 if coef in ("", "+") else -1 if coef == "-" else

```

```

float(coef)
39     elif "y" in part:
40         coef = part.replace("y", "")
41         b = 1 if coef in ("", "+") else -1 if coef == "-" else
42         float(coef)

43     return "normal", (a, b, rhs)

44
45     tipo1, datos1 = parse_restric(restric1)
46     tipo2, datos2 = parse_restric(restric2)

47
48     print("\nLeyenda:\n * = R1\n o = R2\n # = Intersección\n | = Eje Y\n -
49     = Eje X\n + = Origen")

50     for y in range(15, -16, -1):
51         linea = ""
52         for x in range(-15, 16):
53             def cumple(tipo, datos):
54                 if tipo == "vertical":
55                     return x == datos
56                 elif tipo == "horizontal":
57                     return y == datos
58                 else:
59                     a, b, c = datos
60                     if b != 0:
61                         y_calc = (c - a * x) / b
62                         return round(y_calc) == y
63                     elif a != 0:
64                         x_calc = c / a
65                         return round(x_calc) == x
66                     return False

67
68         on_r1 = cumple(tipo1, datos1)
69         on_r2 = cumple(tipo2, datos2)

70
71         if on_r1 and on_r2:
72             linea += "#"
73         elif on_r1:
74             linea += "*"
75         elif on_r2:
76             linea += "o"
77         elif x == 0 and y == 0:
78             linea += "+"
79         elif x == 0:
80             linea += "|"
81         elif y == 0:
82             linea += "-"
83         else:
84             linea += " "
85     print(linea)
86

```

```
87     except (ValueError, IndexError):
88         print("\nError: La restricción ingresada no es válida.")
89
90 graficar_lineas_ascii()
```

Listing 5.1: Graficador de restricciones lineales en Python

5.4 Conclusión

Los problemas de programación lineal permiten modelar situaciones reales donde deben equilibrarse recursos escasos con múltiples objetivos. Aunque la resolución gráfica es viable solo en dos variables, ofrece una intuición geométrica esencial para comprender la estructura de la optimización.

Además, estos ejercicios refuerzan conceptos clave de la programación numérica: formular modelos matemáticos, interpretar restricciones, y —eventualmente— implementar algoritmos para encontrar soluciones óptimas.

Apunte

Para profundizar: En problemas con más de dos variables, se emplean métodos algebraicos (simplex) o numéricos (métodos de punto interior), implementados en librerías como **SciPy** (Python) o **lpSolve** (R).

CAPÍTULO 6

Métodos Numéricos para Hallar Raíces de Ecuaciones

6.1 Introducción

Los **métodos numéricos** son herramientas esenciales en la ciencia y la ingeniería porque muchas ecuaciones no pueden resolverse analíticamente. En lugar de buscar soluciones exactas, estos métodos generan *aproximaciones sucesivas* que convergen a una raíz de la ecuación:

$$f(x) = 0$$

La idea central es iterar: comenzar con una estimación inicial y refinarla paso a paso hasta alcanzar una precisión aceptable.

Apunte

¿Por qué son necesarios? Funciones no lineales, trascendentes o empíricas (sin forma cerrada) son comunes en modelación física, economía o machine learning. Los métodos numéricos permiten resolverlas con computadoras.

Aplicaciones Prácticas

Área	Ecuación implícita $f(x) = 0$
Ingeniería	Determinar el punto de equilibrio de una estructura o sistema mecánico.
Finanzas	Calcular la tasa interna de retorno (TIR) donde el valor actual neto (VAN) se anula: $\text{VAN}(r) = 0$.
Machine Learning	Encontrar el parámetro λ que minimiza el error: $\frac{dE}{d\lambda} = 0$.

Cuadro 6.1: Ejemplos de problemas donde $f(x) = 0$ surge de forma natural

6.2 Métodos Iterativos para Raíces

A continuación se describen cinco métodos clásicos, cada uno con ventajas y limitaciones específicas.

1. Método de Bisección — “El que nunca falla”

El método de bisección es el más **robusto** y sencillo. Requiere un intervalo $[a, b]$ donde $f(a) \cdot f(b) < 0$ (cambio de signo). En cada iteración, el intervalo se reduce a la mitad, descartando la mitad que no contiene la raíz.

- **Requiere:** Intervalo inicial $[a, b]$ con cambio de signo.
- **Convergencia:** Lineal, con razón $\rho = \frac{1}{2}$.
- **Ventaja:** Siempre converge si se cumplen las condiciones.
- **Desventaja:** Lento comparado con otros métodos.

Apunte

Palabra clave: Intervalo cerrado. Ideal para validar rápidamente la existencia de una raíz antes de aplicar métodos más rápidos.

2. Método de Newton-Raphson — “El veloz y arriesgado”

Este método utiliza la **derivada** de la función para construir una tangente en cada iteración. La siguiente aproximación es el punto donde la tangente cruza el eje x :

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

- **Requiere:** $f'(x) \neq 0$ cerca de la raíz y una buena estimación inicial.
- **Convergencia:** Cuadrática ($p = 2$) si f es suave.
- **Ventaja:** Extremadamente rápido cerca de la raíz.
- **Desventaja:** Puede diverger si la derivada es cero o la estimación inicial es pobre.

Apunte

Palabra clave: Derivada. Fundamental en optimización y algoritmos de aprendizaje automático.

3. Método de la Secante — “El pragmático sin derivada”

Una variante de Newton-Raphson que **no requiere derivadas**. Aproxima $f'(x_n)$ usando dos puntos anteriores:

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

- **Requiere:** Dos estimaciones iniciales x_0, x_1 .
- **Convergencia:** Superlineal, con orden $p \approx 1,618$ (número áureo).
- **Ventaja:** Rápido y no necesita derivada analítica.
- **Desventaja:** Puede ser inestable si los puntos iniciales están mal elegidos.

Apunte

Palabra clave: Aproximación. Muy útil cuando $f'(x)$ es difícil o costosa de calcular.

4. Método de Punto Fijo — “La iteración simple”

Transforma la ecuación $f(x) = 0$ en una forma equivalente $x = g(x)$. Luego, itera:

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

- **Condición de convergencia:** $|g'(x)| < 1$ en un entorno de la raíz.
- **Convergencia:** Lineal.
- **Ventaja:** Conceptualmente simple.
- **Desventaja:** La elección de $g(x)$ es crucial; muchas transformaciones divergen.

Apunte

Palabra clave: Transformación. Requiere ingenio algebraico para garantizar convergencia.

5. Método de Regula Falsi — “Falsa Posición”

Combina la **robustez del bisección** con la **rapidez de la secante**. Usa una línea secante entre $(a, f(a))$ y $(b, f(b))$ para estimar la raíz, pero mantiene un intervalo que siempre contiene la solución.

- **Requiere:** Intervalo inicial $[a, b]$ con cambio de signo.
- **Convergencia:** Variable; más rápida que bisección, más segura que secante.

- **Ventaja:** Combina velocidad y fiabilidad.
- **Desventaja:** Puede estancarse si una cota no se actualiza.

Apunte

Palabra clave: Robustez y velocidad. Ideal como método híbrido en solvers numéricos.

6.3 Conclusión

La elección del método numérico depende del contexto:

- Si se prioriza la **seguridad**, se usa **bisección** o **Regula Falsi**.
- Si se prioriza la **velocidad** y se conoce la derivada, se usa **Newton-Raphson**.
- Si no se tiene la derivada pero se necesita rapidez, se recurre a la **secante**.

Estos métodos constituyen la base de algoritmos modernos en optimización, simulación y machine learning. Dominarlos permite no solo resolver ecuaciones, sino también entender cómo las computadoras “piensan” matemáticamente.

Apunte

Para profundizar: En la práctica, los solvers numéricos (como `scipy.optimize.root` en Python) combinan varios métodos para lograr robustez y eficiencia simultáneamente.

CAPÍTULO 7

Método de Bisección

7.1 Definición del Método de Bisección

El **método de bisección** es un procedimiento numérico e iterativo que se utiliza para encontrar una raíz de una función, es decir, un valor x tal que:

$$f(x) = 0$$

Este método se basa en el **Teorema del Valor Intermedio**: si f es continua en un intervalo cerrado $[a, b]$ y existe un cambio de signo en los extremos, entonces hay al menos una raíz en ese intervalo:

$$f(a) \cdot f(b) < 0 \Rightarrow \exists \xi \in (a, b) \text{ tal que } f(\xi) = 0$$

El algoritmo divide repetidamente el intervalo por la mitad y selecciona el subintervalo donde ocurre el cambio de signo. En cada iteración, el intervalo se reduce y la aproximación a la raíz mejora.

Apunte

En palabras simples: Es un proceso de búsqueda que reduce el rango a la mitad en cada paso, acercándose progresivamente al punto donde la gráfica de la función cruza el eje x .

7.2 Procedimiento

El método sigue estos pasos en cada iteración:

1. Calcular el punto medio:

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

2. Evaluar la función en $f(a)$, $f(b)$ y $f(m)$.

3. Seleccionar el nuevo intervalo:

- Si $f(a) \cdot f(m) < 0$, la raíz está en $[a, m]$.
- Si $f(m) \cdot f(b) < 0$, la raíz está en $[m, b]$.

4. Calcular el error absoluto:

$$e = \frac{b - a}{2}$$

El proceso se detiene cuando $e < \varepsilon$, siendo ε la tolerancia deseada.

Apunte

Requisitos esenciales: La función debe ser continua en $[a, b]$ y debe cumplirse $f(a) \cdot f(b) < 0$.

7.3 Ejemplo: Función Exponencial

Enunciado del problema

Un ingeniero en control de procesos modela la estabilización térmica de un horno con la función:

$$f(x) = e^{3x} - 4$$

donde x es el tiempo en horas. Se sabe que la raíz está en el intervalo $[0, 1]$. Se desea aproximarla con un error menor a 0,1.

Iteraciones paso a paso

Iter.	a	b	m	$f(a)$	$f(b)$	$f(m)$	Error e
1	0.0000	1.0000	0.5000	-3.0000	16.0855	0.4816	0.5000
2	0.0000	0.5000	0.2500	-3.0000	0.4816	-1.8829	0.2500
3	0.2500	0.5000	0.3750	-1.8829	0.4816	-0.9190	0.1250
4	0.3750	0.5000	0.4375	-0.9190	0.4816	-0.2840	0.0625

Cuadro 7.1: Tabla de iteraciones del método de bisección

- **Iteración 1:** $m = 0,5$, $f(0,5) = 0,4816 > 0 \rightarrow$ nuevo intervalo $[0, 0,5]$
- **Iteración 2:** $f(0,25) = -1,8829 < 0 \rightarrow$ nuevo intervalo $[0,25, 0,5]$
- **Iteración 3:** $f(0,375) = -0,919 < 0 \rightarrow$ nuevo intervalo $[0,375, 0,5]$
- **Iteración 4:** $e = 0,0625 < 0,1 \rightarrow$ se detiene

Resultado final

La raíz aproximada es:

$$x \approx 0,44 \quad \text{con error menor a } 0,1$$

7.4 Implementación en Python

A continuación se presenta una implementación del método en Python, incluyendo impresión de la tabla de iteraciones.

```

1 import math
2
3 def f(x):
4     return math.exp(3 * x) - 4
5
6 def biseccion(a, b, tol=0.1):
7     if f(a) * f(b) >= 0:
8         print("Error: No hay cambio de signo en [a, b].")
9         return None
10
11    iteracion = 1
12    print(f"{'Iter':>4} {'a':>8} {'b':>8} {'m':>10} {'f(a)':>10} {'f(b)':>10}
13      {'f(m)':>10} {'Error':>8}")
14    while True:
15        m = (a + b) / 2
16        error = (b - a) / 2
17        fm = f(m)
18
19        print(f"[iteracion:4d] {a:8.4f} {b:8.4f} {m:10.4f} {f(a):10.4f} {f(b):
20          :10.4f} {fm:10.4f} {error:8.4f}")
21
22        if error < tol:
23            print(f"\n Raz aproximada: x {m:.4f}")
24            return m
25
26        if f(a) * fm < 0:
27            b = m
28        else:
29            a = m
30    iteracion += 1
31
32 raiz = biseccion(0.0, 1.0, tol=0.1)
```

Listing 7.1: Método de bisección en Python

7.5 Conclusiones

- El método de bisección es **robusto** y **siempre converge** si se cumplen las condiciones iniciales.
- Su principal desventaja es la **convergencia lineal lenta** (error se reduce a la mitad en cada paso).
- Es ideal como método de **inicio** o cuando se prioriza la fiabilidad sobre la velocidad.
- No requiere derivadas, lo que lo hace útil para funciones complejas o empíricas.

Apunte

Aplicación práctica: En la ingeniería y en la computación científica, el método de bisección se usa frecuentemente como primer paso en solvers híbridos que luego refinan la solución con métodos más rápidos (como Newton-Raphson).

CAPÍTULO 8

Método de Newton-Raphson

8.1 Idea General

El **método de Newton-Raphson** es un procedimiento numérico iterativo para resolver ecuaciones no lineales de la forma:

$$f(x) = 0$$

Su fundamento geométrico consiste en aproximar la raíz usando la **recta tangente** a la curva en un punto inicial cercano. La intersección de esta tangente con el eje x se toma como la siguiente aproximación.

La fórmula iterativa clave es:

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

Apunte

Intuición visual: En cada paso, el método “sigue la pendiente” de la función para acercarse rápidamente a la raíz.

8.2 Derivación del Método

El método se deriva de la **serie de Taylor de primer orden** de $f(x)$ alrededor de x_n :

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

Al imponer $f(x) = 0$ (buscando una raíz) y despejar x , obtenemos:

$$0 = f(x_n) + f'(x_n)(x - x_n) \quad \Rightarrow \quad x = x_n - \frac{f(x_n)}{f'(x_n)}$$

Este valor se asigna como x_{n+1} , generando la sucesión iterativa.

8.3 Convergencia

El método de Newton-Raphson presenta **convergencia cuadrática** bajo las siguientes condiciones:

- f es dos veces continuamente diferenciable en un entorno de la raíz r .
- $f(r) = 0$.
- $f'(r) \neq 0$.

Bajo estas hipótesis, el error $e_n = |x_n - r|$ satisface:

$$e_{n+1} \approx C \cdot e_n^2, \quad \text{donde} \quad C = \frac{|f''(r)|}{2|f'(r)|}$$

Esto implica que, cerca de la raíz, el número de dígitos correctos se duplica en cada iteración.

8.4 Ventajas y Desventajas

Ventajas	Desventajas
<ul style="list-style-type: none"> ■ Convergencia muy rápida (cuadrática). ■ Alta precisión en pocas iteraciones. ■ Ampliamente usado en ciencia e ingeniería. 	<ul style="list-style-type: none"> ■ Requiere el cálculo de $f'(x)$. ■ Puede divergir si x_0 está lejos de la raíz. ■ Falla si $f'(x_n) \approx 0$ (división por cero).

Cuadro 8.1: Comparación de ventajas y limitaciones del método

8.5 Ejemplo: Cálculo de $\sqrt{5}$

Planteamiento

Queremos resolver:

$$f(x) = x^2 - 5 = 0$$

La derivada es $f'(x) = 2x$, y la fórmula iterativa se simplifica a:

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{5}{x_n} \right)$$

Tomamos como aproximación inicial $x_0 = 2$.

n	x_n	Cálculo de x_{n+1}	Resultado x_{n+1}
0	2.000000	$\frac{1}{2}(2 + 5/2) = \frac{1}{2}(4,5)$	2.250000
1	2.250000	$\frac{1}{2}(2,25 + 5/2,25)$	2.236111
2	2.236111	$\frac{1}{2}(2,236111 + 5/2,236111)$	2.236068

Cuadro 8.2: Iteraciones del método de Newton-Raphson para $\sqrt{5}$ **Iteraciones****Resultado final**

La raíz aproximada es:

$$x \approx 2,236068$$

Verificación:

$$f(x) = x^2 - 5 = (2,236068)^2 - 5 \approx 4,999999 - 5 \approx 0$$

El método converge en solo 3 iteraciones a $\sqrt{5}$ con alta precisión.

8.6 Implementación en Python

A continuación se presenta una implementación en Python que admite funciones genéricas escritas como texto (ej. $x^{**}2 - 5$). El programa calcula numéricamente la derivada usando diferencias centradas y selecciona un valor inicial razonable.

```

1 import math
2
3 class MetodoRaphson:
4     def __init__(self, ecuacion):
5         self.ecuacion = ecuacion.replace(" ", "")
6         self.f = lambda x: eval(self.ecuacion)
7         self.x0 = self._valor_inicial()
8
9     def _derivada_num(self, x):
10        h = 1e-4
11        return (self.f(x + h) - self.f(x - h)) / (2 * h)
12
13     def _valor_inicial(self):
14         for x in [1, 2, 3, -1, -2, -3, 0.5, -0.5]:
15             try:
16                 if abs(self._derivada_num(x)) > 0.01 and abs(self.f(x)) <
17                     10000:
18                     return x
19             except:
20                 continue
21         return 1
22
23     def resolver(self, tol=1e-4, max_iter=100):
24         x = self.x0
25         print(f"{'Iter':<6} {'x_n':<12} {'f(x_n)':<12} {'x_{n+1}'<12}")
26         print("-" * 50)

```

```

26
27     for i in range(max_iter):
28         fx = self.f(x)
29         dfx = self._derivada_num(x)
30
31         if abs(dfx) < 1e-4:
32             print("Error: Derivada muy pequeña.")
33             return None
34
35         x_new = x - fx / dfx
36         print(f"{i+1:<6} {x:<12.6f} {fx:<12.6f} {x_new:<12.6f}")
37
38         if abs(fx) < tol:
39             print(f"Converge en {i+1} iteraciones")
40             return x_new
41         x = x_new
42
43     print("Advertencia: Mximo de iteraciones alcanzado.")
44     return x
45
46 # === Ejemplo de uso ===
47 ecuacion = input("Ingresa la funcin f(x): ") # Ej: x**2 - 5
48 metodo = MetodoRaphson(ecuacion)
49 raiz = metodo.resolver()
50
51 if raiz:
52     print(f"\nRaz encontrada: x = {raiz:.6f}")
53     print(f"Verificacin: f({raiz:.6f}) = {metodo.f(raiz):.6e}")
54 else:
55     print("No se encontr la raz")

```

Listing 8.1: Método de Newton-Raphson en Python

8.7 Aplicaciones

El método de Newton-Raphson es ampliamente utilizado en:

- Resolución de ecuaciones no lineales en física y química.
- Cálculo de raíces de polinomios y funciones trascendentes.
- Modelos de equilibrio en ingeniería (termo dinámica, circuitos).
- Algoritmos de optimización en machine learning (búsqueda de mínimos mediante $\nabla f = 0$).

8.8 Conclusión

El método de Newton-Raphson es uno de los algoritmos más eficientes para el cálculo de raíces de funciones no lineales. Su convergencia cuadrática lo hace ideal cuando se dispone de una buena aproximación inicial y la derivada no se anula.

Sin embargo, su sensibilidad a la elección de x_0 y la necesidad de la derivada impiden limitaciones que deben considerarse en la práctica. En muchos solvers numéricos, se combina con métodos más robustos (como la bisección) para garantizar convergencia global.

Apunte

Reflexión final: La elegancia de Newton-Raphson radica en convertir un problema no lineal en una secuencia de problemas lineales (tangentes), logrando así una convergencia asombrosamente rápida.

CAPÍTULO 9

Método de la Secante

9.1 Introducción

El **método de la seante** es un procedimiento numérico iterativo para encontrar raíces de ecuaciones no lineales de la forma:

$$f(x) = 0$$

A diferencia del método de Newton-Raphson, **no requiere el cálculo de la derivada** $f'(x)$. En su lugar, aproxima la pendiente de la tangente usando una **recta secante** que pasa por dos puntos cercanos de la función.

Apunte

En palabras simples: En lugar de usar la pendiente exacta (derivada), se estima con la inclinación entre dos puntos previos de la curva.

9.2 Idea Principal

La derivada en el método de Newton-Raphson se approxima mediante:

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

Sustituyendo en la fórmula de Newton-Raphson, se obtiene la fórmula iterativa del método de la secante:

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

Geométricamente, esta fórmula corresponde a la intersección con el eje x de la recta que pasa por los puntos $(x_{n-1}, f(x_{n-1}))$ y $(x_n, f(x_n))$.

9.3 Pasos del Método

1. Elegir dos valores iniciales x_0 y x_1 cercanos a la raíz.
2. Evaluar $f(x_0)$ y $f(x_1)$.
3. Aplicar la fórmula:

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

4. Repetir el proceso hasta que $|x_{n+1} - x_n| < \varepsilon$, siendo ε la tolerancia deseada.

9.4 Ventajas y Desventajas

Ventajas	Desventajas
<ul style="list-style-type: none"> ■ No requiere derivadas. ■ Converge más rápido que el método de bisección. ■ Ideal cuando $f'(x)$ es difícil de obtener. 	<ul style="list-style-type: none"> ■ Requiere dos valores iniciales. ■ Puede divergir si los puntos iniciales están mal elegidos. ■ No garantiza convergencia global.

Cuadro 9.1: Comparación de ventajas y limitaciones del método de la secante

9.5 Ejemplo 1: Raíz de $f(x) = x^2 - 4$

Datos iniciales

- Función: $f(x) = x^2 - 4$
- Valores iniciales: $x_0 = 1$, $x_1 = 3$

Iteraciones

Iter.	x_{n-1}	x_n	$f(x_{n-1})$	$f(x_n)$	x_{n+1}
1	1.0000	3.0000	-3.0000	5.0000	1.7500
2	3.0000	1.7500	5.0000	-0.9375	1.9474
3	1.7500	1.9474	-0.9375	-0.2078	2.0036
4	1.9474	2.0036	-0.2078	0.0142	2.0000
5	2.0036	2.0000	0.0142	-0.0002	2.0000

Cuadro 9.2: Iteraciones del método de la secante para $f(x) = x^2 - 4$

Resultado final

$$x \approx 2,0000$$

El valor exacto es $\sqrt{4} = 2$, lo que confirma la convergencia del método.

9.6 Ejemplo 2: Dilatación Térmica de una Varilla

Planteamiento físico

Una varilla de longitud inicial $L_0 = 1,00$ m se dilata con la temperatura T según:

$$L(T) = L_0(1 + \alpha T)$$

donde $\alpha = 1,2 \times 10^{-5} \text{ }^{\circ}\text{C}^{-1}$. Si se mide una longitud $L_m = 1,001$ m, queremos encontrar la temperatura T que satisface:

$$f(T) = L_0(1 + \alpha T) - L_m = 0$$

Equivalentemente, podemos resolver:

$$f(T) = \frac{1}{1 + 1,2 \times 10^{-5}T} - 1,001 = 0$$

Iteraciones

Iter.	x_{n-1}	x_n	$f(x_{n-1})$	$f(x_n)$	x_{n+1}
1	50.0000	100.0000	-0.0004	0.0002	83.3333
2	100.0000	83.3333	0.0002	0.0000	83.3333

Cuadro 9.3: Iteraciones del método de la secante para el problema de dilatación térmica

Resultado final

$$T \approx 83,3333 \text{ }^{\circ}\text{C}$$

El método converge en solo 2 iteraciones.

9.7 Implementación en Python

A continuación se presenta una implementación simple y reutilizable del método de la secante en Python:

```

1 def secante(f, x0, x1, tol=1e-6, max_iter=50):
2     for i in range(max_iter):
3         f0, f1 = f(x0), f(x1)
4         if f1 - f0 == 0:
5             print("Error: División por cero.")
6             return None

```

```

7     x2 = x1 - f1 * (x1 - x0) / (f1 - f0)
8     print(f"{i+1:2d} {x0:10.6f} {x1:10.6f} {f0:10.6f} {f1:10.6f} {x2
9     :10.6f}")
10    if abs(x2 - x1) < tol:
11        return x2
12    x0, x1 = x1, x2
13    print("Advertencia: Mximo de iteraciones alcanzado.")
14    return x1
15
16 # Ejemplo 1: f(x) = x**2 - 4
17 print("== Ejemplo 1: f(x) = x - 4 ==")
18 f1 = lambda x: x**2 - 4
19 raiz1 = secante(f1, 1.0, 3.0)
20
21 # Ejemplo 2: Problema de dilatacin trmica
22 print("\n== Ejemplo 2: f(x) = 1/(1 + 1.2e-5*x) - 1.001 ==")
23 f2 = lambda x: 1 / (1 + 1.2e-5 * x) - 1.001
24 raiz2 = secante(f2, 50.0, 100.0)
25 print(f"\n  Races : x      {raiz1:.6f}, x      {raiz2:.6f}")

```

Listing 9.1: Método de la secante en Python

9.8 Conclusión

El método de la secante es una alternativa poderosa y práctica cuando:

- No se conoce o es costoso calcular la derivada de $f(x)$.
- Se desea una convergencia más rápida que la bisección.

Aunque su orden de convergencia es superlineal ($p \approx 1,618$, el número áureo), su principal limitación es la falta de garantía de convergencia si los valores iniciales no están bien elegidos. Por ello, se recomienda usarlo combinado con métodos más robustos en entornos de producción.

Apunte

Reflexión final: El método de la secante representa un equilibrio ideal entre simplicidad, eficiencia y aplicabilidad, lo que lo convierte en una herramienta fundamental en la caja de herramientas del ingeniero numérico.

CAPÍTULO 10

Método del Punto Fijo

10.1 Definición

El **método del punto fijo** es un procedimiento iterativo para resolver ecuaciones no lineales de la forma:

$$f(x) = 0$$

En lugar de resolver directamente esta ecuación, se reescribe en la forma equivalente:

$$x = g(x)$$

Una solución r de $f(x) = 0$ es un **punto fijo** de g , es decir, satisface:

$$r = g(r)$$

A partir de una aproximación inicial x_0 , se construye la sucesión:

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

Si esta sucesión converge, su límite es la raíz buscada.

10.2 Fundamento Teórico

Sea $g : [a, b] \rightarrow [a, b]$ una función continua. Si existe $r \in [a, b]$ tal que $g(r) = r$, entonces r es un punto fijo y, por tanto, una raíz de $f(x) = 0$.

Apunte

Existencia: Si $g(a) > a$ y $g(b) < b$, entonces existe al menos un punto fijo $r \in (a, b)$ por el teorema del valor intermedio.

10.3 Condición de Convergencia

Una condición suficiente para la convergencia es que la derivada de g satisfaga:

$$|g'(r)| < 1$$

en un entorno de la raíz r . En tal caso, la sucesión $\{x_n\}$ converge a r .

Apunte

Teorema del Punto Fijo de Banach: Si g es *contractiva* en un intervalo cerrado (es decir, $|g(x_1) - g(x_2)| \leq k|x_1 - x_2|$ con $0 < k < 1$), entonces existe un único punto fijo y el método converge a él desde cualquier x_0 en ese intervalo.

10.4 Algoritmo

1. **Reescribir** la ecuación $f(x) = 0$ como $x = g(x)$.

2. **Elegir** una aproximación inicial x_0 .

3. **Iterar** usando:

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

4. **Detener** el proceso cuando $|x_{n+1} - x_n| < \varepsilon$, con ε la tolerancia deseada.

10.5 Ejemplo 1: Inversión que se Duplica en un Año

Planteamiento

Queremos hallar la tasa de interés mensual r que duplica una inversión en 12 meses:

$$f(r) = (1 + r)^{12} - 2 = 0$$

Reescribimos como:

$$g(r) = \frac{2}{(1 + r)^{11}} - 1$$

Iteraciones (con $x_0 = 0,05$)

- $x_1 = g(0,05) \approx 0,059$
- $x_2 = g(0,059) \approx 0,058$
- $x_3 = g(0,058) \approx 0,058$

Resultado final

$$r \approx 0,058 \quad (\text{es decir, } 5.8\% \text{ mensual})$$

10.6 Ejemplo 2: Resolución de $f(x) = e^x - 4 + x = 0$

Transformación

$$f(x) = e^x - 4 + x = 0 \quad \Rightarrow \quad x = \ln(4 - x) = g(x)$$

Iteraciones (con $x_0 = 1$, tolerancia = 10^{-6})

n	x_n	x_{n+1}	$g(x_n)$	Error relativo (%)
0	1.000000	1.098612	1.098612	8.982
1	1.098612	1.065191	1.065191	3.139
2	1.065191	1.076630	1.076630	1.064
3	1.076630	1.072732	1.072732	0.363
4	1.072732	1.074100	1.074100	0.127
5	1.074100	1.073642	1.073642	0.042
6	1.073642	1.073798	1.073798	0.014
7	1.073798	1.073745	1.073745	0.005
8	1.073745	1.073763	1.073763	0.002
9	1.073763	1.073757	1.073757	0.001
10	1.073757	1.073759	1.073759	0.0002
11	1.073759	1.073758	1.073758	0.00007

Cuadro 10.1: Iteraciones del método del punto fijo para $f(x) = e^x - 4 + x$

Resultado final

$$x \approx 1.073758 \quad \text{con error relativo} < 0,0001\%$$

10.7 Implementación en Python

```

1 import math
2
3 def g(x):
4     return math.log(4 - x)
5
6 def punto_fijo(x0, tol=1e-6, max_iter=100):
7     x = x0
8     for i in range(max_iter):
9         try:
10             x_new = g(x)
11         except ValueError:
12             print("Error: g(x) no est definido (log de nmero 0).")
13             return None
14
15         error = abs((x_new - x) / x_new) if x_new != 0 else 0
16         print(f"{i:2d} {x:10.6f} {x_new:10.6f} {error*100:10.4f}%")
17
18         if error < tol:
19             return x_new
20         x = x_new
21
22     print("Advertencia: Mximo de iteraciones alcanzado.")
23     return x

```

```

24
25 # Ejecucion
26 raiz = punto_fijo(1.0)
27 print(f"\n Raz aproximada: x {raiz:.6f}")

```

Listing 10.1: Método del punto fijo en Python

10.8 Ventajas y Desventajas

Ventajas	Desventajas
<ul style="list-style-type: none"> ■ Implementación simple. ■ Bajo consumo de memoria. ■ Útil para generar valores iniciales. ■ No requiere derivadas. 	<ul style="list-style-type: none"> ■ No siempre es posible construir $g(x)$ adecuada. ■ Convergencia lenta (lineal). ■ Alta sensibilidad a x_0 y a la forma de $g(x)$. ■ Puede divergir si $g'(x) \geq 1$.

Cuadro 10.2: Comparación de ventajas y limitaciones del método del punto fijo

10.9 Conclusión

El método del punto fijo es una herramienta conceptualmente simple pero con matices importantes en su aplicación:

- Su éxito depende críticamente de la **elección de $g(x)$** y del **valor inicial**.
- Es ideal como método preliminar o cuando se dispone de una transformación natural de la ecuación.
- Si bien su convergencia es lineal (más lenta que Newton-Raphson), su simplicidad lo hace valioso en contextos educativos y en la generación de buenas aproximaciones iniciales.

Apunte

Reflexión final: El arte del punto fijo radica en reescribir $f(x) = 0$ de manera inteligente: una mala elección de $g(x)$ conduce a la divergencia; una buena, a la solución con elegancia y minimalismo.

CAPÍTULO 11

Método de Regula Falsi

11.1 Descripción General

El **método de Regula Falsi** (también conocido como *falsa posición*) es un procedimiento numérico para encontrar raíces de ecuaciones no lineales de la forma:

$$f(x) = 0$$

Es un método **cerrado**, lo que significa que trabaja dentro de un intervalo inicial $[a, b]$ donde se garantiza la existencia de al menos una raíz gracias al **Teorema del Valor Intermedio**:

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

11.2 Diferencia con el Método de Bisección

A diferencia del método de bisección, que siempre selecciona el *punto medio*, Regula Falsi usa **interpolación lineal** entre los puntos $(a, f(a))$ y $(b, f(b))$ para estimar la raíz. Esto aprovecha la forma de la función y, en la mayoría de los casos, mejora la velocidad de convergencia.

Apunte

Ventaja clave: Converge más rápido que la bisección porque incorpora información sobre el comportamiento de la función, no solo su signo.

11.3 Fórmula de Interpolación

Dado un intervalo $[a_n, b_n]$, el nuevo punto c_n se calcula como la intersección de la recta secante con el eje x :

$$c_n = b_n - f(b_n) \cdot \frac{b_n - a_n}{f(b_n) - f(a_n)}$$

Una forma algebraicamente equivalente es:

$$c_n = \frac{a_n f(b_n) - b_n f(a_n)}{f(b_n) - f(a_n)}$$

11.4 Algoritmo

1. Calcular c_n usando la fórmula de interpolación.
2. Evaluar el signo de $f(c_n)$:
 - Si $f(a_n) \cdot f(c_n) < 0$, actualizar el intervalo a $[a_n, c_n]$.
 - Si $f(c_n) \cdot f(b_n) < 0$, actualizar el intervalo a $[c_n, b_n]$.
3. Repetir hasta que $|c_{n+1} - c_n| < \varepsilon$ o se alcance el número máximo de iteraciones.

11.5 Características

- Converge más rápido que el método de bisección.
- Puede *estancarse* si uno de los extremos del intervalo no se actualiza durante varias iteraciones (ocurre en funciones con alta curvatura).
- Es un precursor del método de la secante, pero mantiene la robustez de trabajar dentro de un intervalo que siempre contiene la raíz.

11.6 Ejemplo 1: $f(x) = x^2 - 4$

Datos iniciales

- Intervalo: $[a_0, b_0] = [1, 3]$
- $f(1) = -3, f(3) = 5 \rightarrow$ cambio de signo

Iteraciones

Iter.	a_n	b_n	$f(a_n)$	$f(b_n)$	c_n	$f(c_n)$
1	1.00000	3.00000	-3.00000	5.00000	1.75000	-0.93750
2	1.75000	3.00000	-0.93750	5.00000	1.94737	-0.20690
3	1.94737	3.00000	-0.20690	5.00000	1.98851	-0.04586
4	1.98851	3.00000	-0.04586	5.00000	1.99746	-0.01017
5	1.99746	3.00000	-0.01017	5.00000	1.99935	-0.00226

Cuadro 11.1: Iteraciones del método de Regula Falsi para $f(x) = x^2 - 4$

Resultado final

$x \approx 1,99935$ (el valor exacto es $x = 2$)

11.7 Ejemplo 2: $f(x) = x^3 - 2x - 5$

Datos iniciales

- Intervalo: $[2, 3]$
- $f(2) = -1, f(3) = 16 \rightarrow$ cambio de signo

Iteraciones (primeras 5)

Iter.	a_n	b_n	$f(a_n)$	$f(b_n)$	c_n	$f(c_n)$
1	2.00000	3.00000	-1.00000	16.00000	2.05882	-0.39054
2	2.05882	3.00000	-0.39054	16.00000	2.08285	-0.14407
3	2.08285	3.00000	-0.14407	16.00000	2.09171	-0.05333
4	2.09171	3.00000	-0.05333	16.00000	2.09499	-0.01973
5	2.09499	3.00000	-0.01973	16.00000	2.09621	-0.00730

Cuadro 11.2: Iteraciones del método de Regula Falsi para $f(x) = x^3 - 2x - 5$

Resultado final

$$x \approx 2,09621$$

11.8 Implementación en Python

```

1 def f(x):
2     return x**2 - 4
3
4 def regula_falsi(a, b, tol=1e-4, max_iter=20):
5     if f(a) * f(b) >= 0:
6         print("Error: f(a) y f(b) deben tener signos opuestos.")
7         return None
8
9     for i in range(max_iter):
10        c = b - f(b) * (b - a) / (f(b) - f(a))
11        fc = f(c)
12        print(f"\{i+1:2d} {a:10.6f} {b:10.6f} {f(a):10.6f} {f(b):10.6f} {c:10.6f} {fc:10.6f}")
13        if abs(fc) < tol:
14            return c
15        if f(a) * fc < 0:
16            b = c
17        else:
18            a = c
19    print("Advertencia: Mximo de iteraciones alcanzado.")
20    return c
21
22 # Ejecucion

```

```
23 raiz = regula_falsi(1.0, 3.0)
24 print(f"\n  Raz  aproximada: x  {raiz:.6f}")
```

Listing 11.1: Método de Regula Falsi en Python

11.9 Conclusión

El método de Regula Falsi representa un equilibrio entre robustez y eficiencia:

- Mejora significativamente la velocidad de la bisección al usar una aproximación lineal (secante) en lugar del punto medio.
- Mantiene la garantía de convergencia de los métodos cerrados, siempre que f sea continua y $f(a) \cdot f(b) < 0$.
- Su principal limitación es la posible estancamiento en un extremo, lo que ha motivado variantes modernas (como el método de Illinois).

Apunte

Reflexión final: Regula Falsi es un puente histórico y conceptual entre los métodos seguros (bisección) y los rápidos pero arriesgados (Newton-Raphson), ideal para contextos donde se requiere fiabilidad sin sacrificar del todo la eficiencia.

CAPÍTULO 12

Gradiente

12.1 Definición

El método del gradiente es un enfoque fundamental y ampliamente utilizado para resolver **problemas de optimización** en informática. Implica la construcción de una **función de error** adecuada para el problema de optimización y el diseño de modelos, como redes neuronales, que actualizan iterativamente los parámetros moviéndose a lo largo de la dirección de descenso del gradiente negativo de la función de error.

[leftmargin=*)

- **Perspectiva matemática:** El descenso de gradiente y el ascenso de gradiente son operaciones equivalentes: el ascenso de gradiente maximiza una función objetivo, mientras que el descenso de gradiente minimiza una función de pérdida, y los cálculos subyacentes son análogos.
- **Optimización numérica:** El método de gradiente forma la base de la optimización numérica al calcular el gradiente de la función objetivo y tomar medidas en dirección descendente para mejorar el valor de la función, repitiendo este proceso hasta que se logre la convergencia.

Formalmente, el problema de optimización se expresa como:

$$\min_{\mathbf{x} \in R^n} f(\mathbf{x})$$

y el gradiente se define como:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

12.2 Gradiente de una Función

El **gradiente** de una función representa la dirección y la magnitud del cambio más rápido de dicha función.

En una función de una sola variable $f(x)$, el gradiente corresponde simplemente a su derivada:

$$\nabla f(x) = f'(x)$$

En una función de varias variables, por ejemplo $f(x, y)$, el gradiente es un vector que contiene todas las derivadas parciales:

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

De forma general, para una función $f(x_1, x_2, \dots, x_n)$:

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

El gradiente apunta en la dirección de **mayor incremento** de la función. Por ello, para **minimizar** una función se sigue la dirección contraria al gradiente.

12.3 Interpretación geométrica

Desde un punto de vista geométrico, el gradiente de una función escalar $f(x, y)$ en un punto dado apunta en la dirección en la que la función crece más rápidamente. Su magnitud indica qué tan pronunciado es dicho crecimiento.

Geométricamente, el gradiente es perpendicular a las curvas de nivel de la función. Si $f(x, y) = c$ define una curva de nivel y \mathbf{t} es un vector tangente a dicha curva, entonces:

$$\nabla f(x, y) \cdot \mathbf{t} = 0$$

12.4 Derivada direccional

La derivada direccional mide la tasa de cambio de una función en una dirección específica. Sea $f(x, y)$ diferenciable y \mathbf{u} un vector unitario, la derivada direccional se define como:

$$D_{\mathbf{u}}f = \nabla f \cdot \mathbf{u}$$

El valor máximo de la derivada direccional se alcanza cuando \mathbf{u} coincide con la dirección del gradiente:

$$\max_{\|\mathbf{u}\|=1} D_{\mathbf{u}}f = \|\nabla f\|$$

12.5 Propiedades fundamentales

[leftmargin=*)

- El gradiente apunta en la dirección de máximo crecimiento.
- Es perpendicular a curvas o superficies de nivel.
- Un punto crítico satisface:

$$\nabla f(\mathbf{x}^*) = \mathbf{0}$$

- La magnitud del gradiente indica la rapidez del cambio.

12.6 Gradiante y optimización

El método del descenso de gradiente actualiza iterativamente la solución mediante:

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

Bajo condiciones de convexidad y una adecuada elección del tamaño de paso, el método converge a un mínimo local o global.

12.7 Aplicaciones

1. Física clásica y electromagnetismo
2. Ingeniería y modelado
3. Aprendizaje automático y regresión
4. Métodos numéricos

Apunte

Para recordar: El gradiente no solo indica la dirección de máximo crecimiento, sino que su magnitud representa la tasa de ese crecimiento. En optimización, seguimos la dirección opuesta al gradiente para encontrar mínimos.

12.8 Implementación en R para Gradiante de una función

12.8.1 Método del Descenso del Gradiante

El **descenso del gradiante** es un algoritmo iterativo utilizado para encontrar el mínimo de una función diferenciable. En cada iteración, el valor de la variable se actualiza según la regla:

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

donde:

[leftmargin=*)]

- x_i : valor actual de la variable,
- η : tasa de aprendizaje,
- $f'(x_i)$: derivada o pendiente de la función en x_i .

El proceso se repite hasta que la derivada $f'(x)$ se approxima a cero, lo cual indica que se ha alcanzado un punto mínimo (local o global).

Ejemplo Aplicado

Para la función:

$$f(x) = x^2 \Rightarrow f'(x) = 2x$$

La actualización se realiza como:

$$x_{i+1} = x_i - \eta (2x_i)$$

Si el valor inicial es $x_0 = 3$ y la tasa de aprendizaje $\eta = 0,01$, los valores de x se van acercando gradualmente al punto mínimo en $x = 0$, donde la pendiente $f'(x) = 0$. Este proceso se denomina **gradiente descendente**, ya que cada paso desciende la función hacia su valor mínimo.

```

1 # GRADIENTE
2 n <- 0.01           # Tasa de aprendizaje
3 x0 <- 3             # Valor inicial
4 iter <- 21           # Número de iteraciones
5
6 # DEFINICIÓN DE FUNCIONES
7 f <- function(x) x^2           # Función objetivo
8 f_deriv <- function(x) 2 * x      # Derivada
9
10 # INICIALIZACIÓN DE VECTORES
11 x <- numeric(iter)
12 fx <- numeric(iter)
13 fpfx <- numeric(iter)
14 grad <- numeric(iter)
15
16 x[1] <- x0
17
18 # CLCULO ITERATIVO
19 for (i in 1:iter) {
20   fx[i] <- f(x[i])           # f(x)
21   fpfx[i] <- f_deriv(x[i])    # f'(x)
22   grad[i] <- x[i] - n * fpfx[i] # x actualizado
23   if (i < iter) {
24     x[i + 1] <- grad[i]
25   }
26 }
27
28 # TABLA DE RESULTADOS
29 tabla <- data.frame(
30   Iteracion = 1:iter,
31   xo = x,
32   fx = fx,
33   fpfx = fpfx,
34   grad = grad
35 )
36 print(tabla)
37
38 # DATOS PARA GRFICO
39 library(tidyverse)
40 library(ggplot2)

```

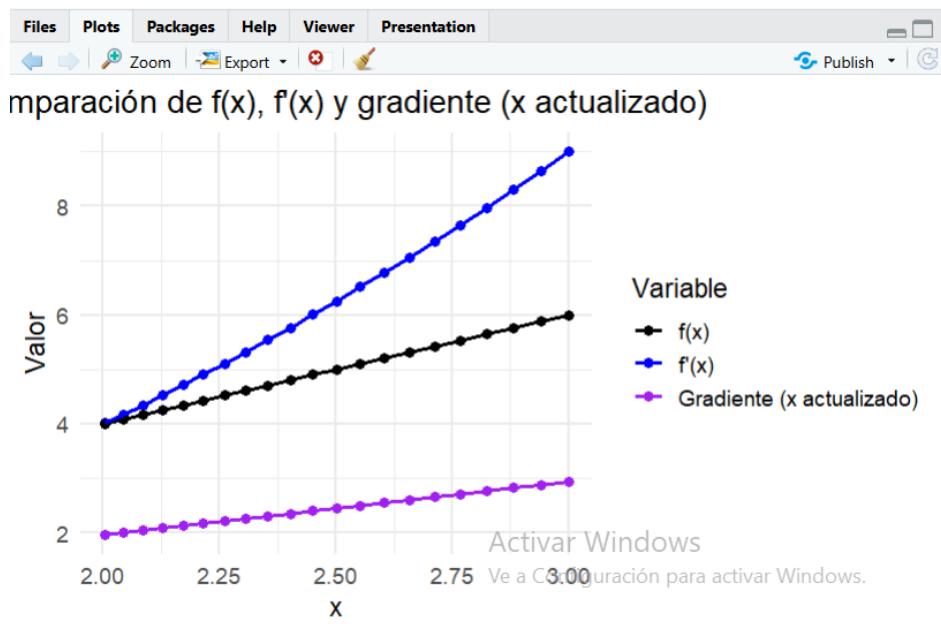
```

41
42 datos_long <- tabla |>
43   pivot_longer(cols = c(fx, fp, grad),
44                 names_to = "Variable",
45                 values_to = "Valor")
46
47 # GRFICO
48 ggplot(datos_long, aes(x = xo, y = Valor, color = Variable)) +
49   geom_point(size = 2) +
50   geom_line(linewidth = 1) +
51   scale_color_manual(values = c("black", "blue", "purple"),
52                      labels = c("f(x)", "f'(x)", "Gradiente (x actualizado)"))
53 ) +
54 labs(title = "Comparación de f(x), f'(x) y gradiente (x actualizado)",
55       x = "x",
56       y = "Valor",
57       color = "Variable") +
58 theme_minimal(base_size = 14) +
59 theme(plot.title = element_text(hjust = 0.5))

```

Listing 12.1: Implementación del método del gradiente en R

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



12.8.2 Método del Gradiente para dos variables

```

1 # GRADIENTE (2 VARIABLES)
2 library(ggplot2)
3 library(plotly)
4 library(tidyr)
5
6 # Parmetros
7 n <- 0.1          # Tasa de aprendizaje
8 x0 <- 3           # Valor inicial de x
9 y0 <- -2          # Valor inicial de y
10 iter <- 25         # Número de iteraciones
11
12 # Definición de funciones
13 f <- function(x, y) x^2 + y^2          # Función objetivo
14 grad_f <- function(x, y) c(2*x, 2*y)    # Gradiente  $f(x, y)$ 
15
16 # Inicialización
17 x <- numeric(iter)
18 y <- numeric(iter)
19 fx <- numeric(iter)
20
21 x[1] <- x0
22 y[1] <- y0
23 fx[1] <- f(x[1], y[1])
24
25 # Iteraciones del gradiente
26 for (i in 1:(iter - 1)) {
27   grad <- grad_f(x[i], y[i])
28   x[i + 1] <- x[i] - n * grad[1]
29   y[i + 1] <- y[i] - n * grad[2]
30   fx[i + 1] <- f(x[i + 1], y[i + 1])
31 }
```

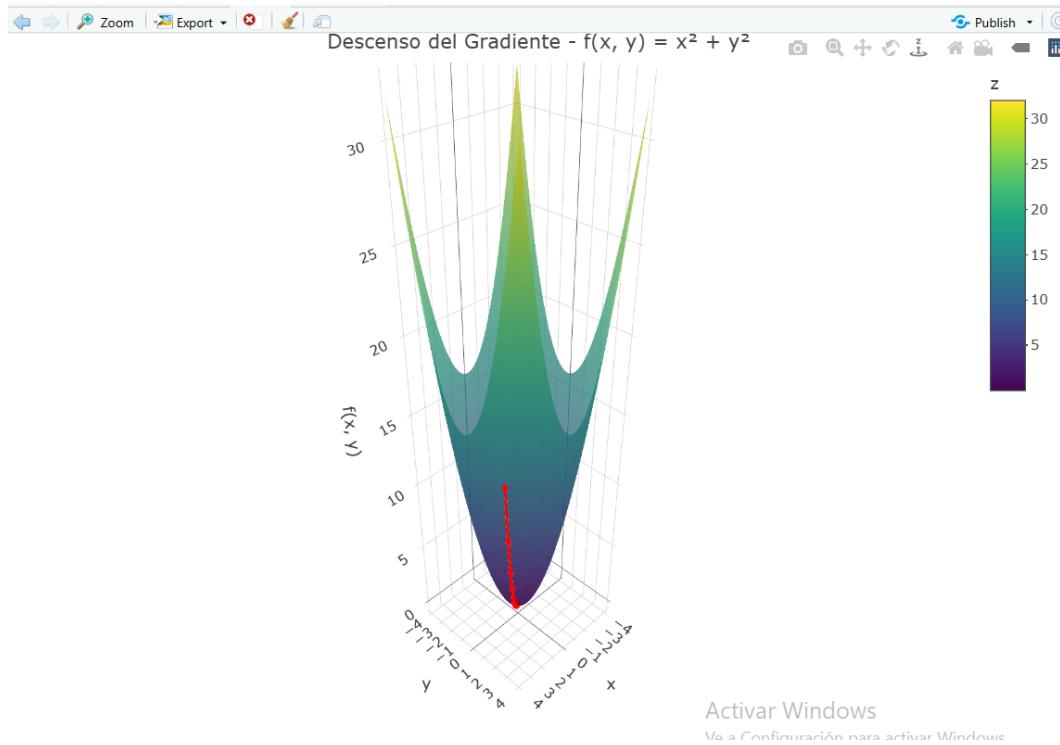
```

32 # Tabla de resultados
33 tabla <- data.frame(
34   Iter = 1:iter,
35   x = x,
36   y = y,
37   fxy = fx
38 )
39 )
40 print(tabla)
41
42 # Superficie 3D con trayectoria
43 x_seq <- seq(-4, 4, length.out = 100)
44 y_seq <- seq(-4, 4, length.out = 100)
45 z <- outer(x_seq, y_seq, f)
46
47 fig <- plot_ly() |>
48   add_surface(x = ~x_seq, y = ~y_seq, z = ~z, opacity = 0.7, colorscale = "
49     Viridis") |>
50   add_trace(x = x, y = y, z = fx, type = "scatter3d", mode = "lines+markers",
51             line = list(color = "red", width = 5),
52             marker = list(size = 3, color = "red"),
53             name = "Trayectoria del gradiente") |>
54   layout(
55     title = "Descenso del Gradiente - f(x, y) = x + y",
56     scene = list(
57       xaxis = list(title = "x"),
58       yaxis = list(title = "y"),
59       zaxis = list(title = "f(x, y)")
60     )
61   )
62 fig

```

Listing 12.2: Implementación del método del gradiente en R para dos variables

Iter	x	y	fxy
1	3.00000000	2.000000000	1.300000e+01
2	2.40000000	1.600000000	8.320000e+00
3	1.92000000	1.280000000	5.324800e+00
4	1.53600000	1.024000000	3.407872e+00
5	1.22880000	0.819200000	2.181038e+00
6	0.98304000	0.655360000	1.395864e+00
7	0.78643200	0.524288000	8.933532e-01
8	0.62914560	0.419430400	5.717460e-01
9	0.50331648	0.335544320	3.659175e-01
10	0.40265318	0.268435456	2.341872e-01
11	0.32212255	0.214748365	1.498798e-01
12	0.25769804	0.171798692	9.592307e-02
13	0.20615843	0.137438953	6.139076e-02
14	0.16492674	0.109951163	3.929009e-02
15	0.13194140	0.087960930	2.514566e-02
16	0.10555312	0.070368744	1.609322e-02
17	0.08444249	0.056294995	1.029966e-02
18	0.06755399	0.045035996	6.591783e-03
19	0.05404320	0.036028797	4.218741e-03
20	0.04323456	0.028823038	2.699994e-03
21	0.03458765	0.023058430	1.727996e-03
22	0.02767012	0.018446744	1.105918e-03
23	0.02213609	0.014757395	7.077873e-04
24	0.01770887	0.011805916	4.529839e-04
25	0.01416710	0.009444733	2.899097e-04



Apunte

Consejo práctico: En el método del gradiente descendente, la elección de la tasa de aprendizaje (η) es crucial. Un valor demasiado grande puede hacer que el algoritmo diverja, mientras que un valor demasiado pequeño puede hacer que la convergencia sea muy lenta. Es común usar técnicas como la búsqueda de línea o métodos adaptativos para ajustar η durante la optimización.

12.9 Convergencia del Método del Gradiente

El método del gradiente converge bajo ciertas condiciones:

1. **Función convexa:** Si f es convexa y diferenciable, el gradiente converge al mínimo global.
2. **Lipschitz continuidad:** Si ∇f es Lipschitz continuo con constante L , entonces con $\eta \leq 1/L$ se garantiza convergencia.
3. **Convergencia lineal:** Bajo condiciones adecuadas, el error decrece linealmente.

La condición de optimalidad de primer orden establece que si \mathbf{x}^* es un mínimo local, entonces:

$$\nabla f(\mathbf{x}^*) = \mathbf{0}$$

12.10 Variantes del Gradiente Descendente

- **Gradiente descendente batch:** Usa todo el conjunto de datos para calcular el gradiente.
- **Gradiente descendente estocástico (SGD):** Usa un solo ejemplo por iteración.
- **Gradiente descendente mini-batch:** Compromiso entre batch y estocástico.
- **Momentum:** Incluye un término de momento para acelerar la convergencia.
- **Adam:** Combina momentum y escalado adaptativo del learning rate.

12.11 Consideraciones Prácticas

- **Inicialización:** La elección del punto inicial puede afectar la convergencia a mínimos locales.
- **Criterio de parada:** Detener cuando $\|\nabla f(\mathbf{x}_k)\| < \epsilon$ o después de un número máximo de iteraciones.
- **Escalado de características:** Normalizar las variables puede mejorar la convergencia.
- **Visualización:** Graficar la función de costo vs iteraciones ayuda a diagnosticar problemas.

CAPÍTULO 13

Diferenciación Numérica

13.1 Introducción

La diferenciación numérica es una herramienta fundamental en el análisis numérico que permite aproximar derivadas cuando no se dispone de la función analítica o cuando esta es demasiado compleja. En este capítulo exploraremos los métodos de diferencias finitas y sus aplicaciones en problemas prácticos de ciencia de datos, machine learning e ingeniería.

13.2 Fundamentos Teóricos

13.2.1 Definición de Derivada

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

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

Este límite representa la tasa de cambio instantánea de la función en el punto x . En la práctica computacional, trabajamos con un h pequeño pero finito.

13.2.2 El Parámetro h

El parámetro h , conocido como tamaño de paso, juega un papel crucial en la diferenciación numérica. Su elección representa un compromiso entre:

- **Error de truncamiento:** Disminuye al reducir h
- **Error de redondeo:** Aumenta al reducir h debido a la precisión finita de la computadora

13.3 Métodos de Diferencias Finitas

13.3.1 Diferencia Hacia Adelante

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

Error: $O(h)$ (proporcional a h)

13.3.2 Diferencia Hacia Atrás

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

Error: $O(h)$

13.3.3 Diferencia Centrada

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

Error: $O(h^2)$ (significativamente menor que los anteriores)

Apunte

Para recordar: La diferencia centrada es más precisa porque utiliza información de ambos lados del punto, cancelando parcialmente los errores. Es aproximadamente 100 veces más precisa que las diferencias hacia adelante o hacia atrás.

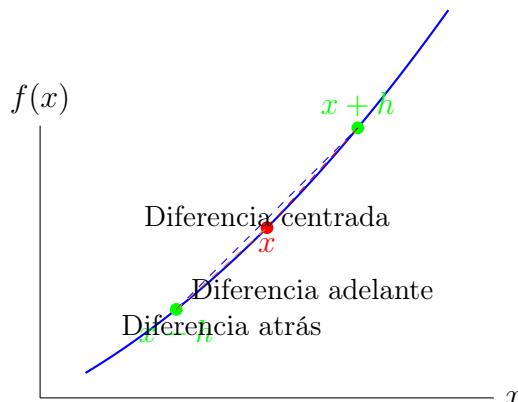


Figura 13.1: Representación gráfica de los métodos de diferencias finitas

13.4 Implementación en R

13.4.1 Funciones Básicas de Diferenciación

Apunte

Consejo de programación: En R, siempre es buena práctica documentar tus funciones y validar los argumentos de entrada para evitar errores en tiempo de ejecución.

```

1
2 # FUNCIONES DE DIFERENCIACION NUMRICA EN R
3
4
5 # Diferencia hacia adelante
6 derivada_adelante <- function(f, x, h = 1e-5) {
7   return((f(x + h) - f(x)) / h)
8 }
9
10 # Diferencia hacia atrs
11 derivada_atras <- function(f, x, h = 1e-5) {
12   return((f(x) - f(x - h)) / h)
13 }
14
15 # Diferencia centrada
16 derivada_centrada <- function(f, x, h = 1e-5) {
17   return((f(x + h) - f(x - h)) / (2 * h))
18 }
```

Listing 13.1: Implementación en R

13.5 Ejercicios Prácticos

13.5.1 Ejercicio : Análisis de Crecimiento de Usuarios

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

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

Cuadro 13.1: Datos de crecimiento de usuarios

Tareas:

1. Calcula la tasa de crecimiento de usuarios en el mes 4 usando diferencia centrada
2. Calcula la tasa de crecimiento en el mes 1 (usa diferencia adelante)
3. Calcula la tasa de crecimiento en el mes 7 (usa diferencia atrás)
4. ¿En qué mes fue mayor la aceleración del crecimiento?

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

La fórmula de la diferencia centrada es:

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

con $h = 1$ y $x = 4$:

$$f'(4) \approx \frac{f(5) - f(3)}{2} = \frac{48 - 23}{2} = \frac{25}{2} = 12,5$$

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

$$f'(1) \approx \frac{f(2) - f(1)}{1} = 15 - 10 = 5$$

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

$$f'(7) \approx \frac{f(7) - f(6)}{1} = 85 - 65 = 20$$

4) Segunda derivada para identificar la aceleración del crecimiento

La fórmula de diferencias finitas para la segunda derivada es:

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

Calculando para los meses 2 al 6:

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

Como la segunda derivada es positiva y constante, el crecimiento es acelerado y estable.

5) Interpretación

La startup no sólo está ganando usuarios, sino que lo hace con **aceleración positiva constante**. Esto indica una etapa de **crecimiento saludable y expansión**.

Código en R

```

1 # Datos ejercicio
2 mes <- 1:7
3 usuarios <- c(10, 15, 23, 34, 48, 65, 85)
4
5 # 1) Diferencia centrada en mes 4
6 tasa_mes4 <- (usuarios[5] - usuarios[3]) / 2
7
8 # 2) Diferencia hacia adelante en mes 1
9 tasa_mes1 <- usuarios[2] - usuarios[1]
10
11 # 3) Diferencia hacia atras en mes 7
12 tasa_mes7 <- usuarios[7] - usuarios[6]
13
14 # 4) Segunda derivada para meses 2 al 6
15 segunda_derivada <- usuarios[3:7] - 2*usuarios[2:6] + usuarios[1:5]
16 mes_aceleracion <- 2:6

```

```

17
18 # Resultados
19 tasa_mes4
20 tasa_mes1
21 tasa_mes7
22 data.frame(Mes = mes_aceleracion, Aceleracion = segunda_derivada)
23
24 # 5) Interpretacion
25 # Como la segunda derivada es positiva y constante, el crecimiento es
26 # acelerado.
26 cat("\n5. Interpretacion: La startup esta creciendo de forma acelerada, ya
    que cada mes se suman mas usuarios que el mes anterior y la segunda
    derivada es positiva.\n")

```

Listing 13.2: Implementación en R

Apunte

Interpretación de resultados: Una tasa de crecimiento positiva indica que la startup está ganando usuarios, mientras que una segunda derivada positiva sugiere que el crecimiento se está acelerando.

13.6 Tabla Comparativa de Métodos

Método	Fórmula	Error
Diferencia Adelante	$f'(x) \approx \frac{f(x+h) - f(x)}{h}$	$O(h)$
Diferencia Atrás	$f'(x) \approx \frac{f(x) - f(x-h)}{h}$	$O(h)$
Diferencia Centrada	$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$	$O(h^2)$

Cuadro 13.2: Comparación de métodos de diferenciación numérica

13.7 Consideraciones Prácticas

13.7.1 ¿Cómo Elegir h ?

El dilema de h :

- Si h es muy grande: mala aproximación (error grande)
- Si h es muy pequeño: errores de redondeo en la computadora

13.7.2 Recomendaciones prácticas:

- Para diferencias adelante/atrás: usar $h \approx 10^{-4}$ a 10^{-6}
- Para diferencia centrada: usar $h \approx 10^{-5}$ a 10^{-8}
- Experimentar con diferentes valores y ver cuál da mejores resultados

13.8 Conclusión

La diferenciación numérica es una herramienta poderosa que, cuando se aplica correctamente, permite extraer información valiosa de datos discretos. Los métodos presentados en este capítulo forman la base para técnicas más avanzadas como la optimización numérica, solución de ecuaciones diferenciales y análisis de sensibilidad.

Apunte

Para profundizar: Los métodos de diferenciación numérica son esenciales en algoritmos de aprendizaje automático, especialmente en el cálculo de gradientes para el descenso de gradiente en redes neuronales.

CAPÍTULO 14

Interpolación Numérica

La **interpolación numérica** es una de las herramientas fundamentales del análisis numérico. Su objetivo principal es aproximar una función desconocida a partir de un conjunto finito de datos conocidos, permitiendo estimar valores intermedios de manera eficiente y controlada.

Apunte

Aplicaciones comunes:

- Pronóstico del tiempo entre horas de medición
- Estimación de valores en tablas matemáticas
- Suavizado de curvas en diseño gráfico
- Análisis de datos experimentales

En muchos problemas reales no se dispone de la expresión analítica de una función, sino únicamente de valores medidos o calculados en puntos específicos:
 $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$.

La interpolación busca construir una función $p(x)$ tal que:
 $p(x_i) = y_i, \quad i = 0, 1, \dots, n.$

Apunte

La interpolación no debe confundirse con la extrapolación. La interpolación estima valores *dentro* del intervalo de los datos conocidos, mientras que la extrapolación lo hace fuera de él, con mayor riesgo de error.

14.1 Interpolación Lineal

La **interpolación lineal** es el método más simple y se utiliza cuando se dispone de dos puntos consecutivos (x_0, y_0) y (x_1, y_1) . La aproximación consiste en unir ambos puntos mediante una recta.

La fórmula básica es: $[y = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0).]$

De forma equivalente: $[y = y_0(x_1 - x) + y_1(x - x_0) \frac{1}{x_1 - x_0}.]$

Ejercicio Interpolación Lineal

A las 8:00 la temperatura era $20^{\circ}C$ y a las 12:00 era $28^{\circ}C$. Estimar la temperatura a las 10:30.

$$y = 20 + \frac{28 - 20}{12 - 8}(10,5 - 8) = 25^{\circ}C.$$

Resultado: $25^{\circ}C$

```

1 x0 <- 8; y0 <- 20
2 x1 <- 12; y1 <- 28
3 x <- 10.5
4 y <- y0 + (y1-y0)/(x1-x0)*(x-x0)
5 y

```

Apunte

La interpolación lineal es adecuada cuando los datos varían suavemente o cuando solo se requiere una aproximación rápida.

14.2 Interpolación Polinómica

El método más utilizado consiste en aproximar los datos mediante un **polinomio interpolante** de grado menor o igual que n : $[p_n(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n.]$

Existe un único polinomio de grado $\leq n$ que interpola $n + 1$ puntos distintos.

14.2.1 Forma de Lagrange

El polinomio interpolante de Lagrange se define como: $[p_n(x) = \sum_{i=0}^n y_i L_i(x),]$ donde los polinomios base son: $[L_i(x) = \prod_{j=0, j \neq i}^n \frac{x - x_j}{x_i - x_j}.]$

Apunte

La forma de Lagrange es conceptualmente sencilla y útil para el análisis teórico, pero no es la más eficiente para cálculos repetidos.

14.2.2 Ejemplo

Dados los puntos: [(1,2), (2,3), (4,1),]
el polinomio interpolante de grado 2 se obtiene aplicando directamente la fórmula de Lagrange.

14.3 Interpolación de Newton

La forma de Newton utiliza **diferencias divididas**, permitiendo una construcción incremental del polinomio: [$p_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \dots$]

Los coeficientes a_k se calculan mediante diferencias divididas definidas recursivamente.

14.3.1 Diferencias Divididas

Para puntos distintos x_0, x_1, \dots, x_n : [$f[x_i] = y_i$,] [$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$.]

Ejercicio Aplicado

Datos: (1, 1), (2, 4), (3, 9), $x = 2,5$

$$f[x_0] = 1, \quad f[x_1] = 4, \quad f[x_2] = 9$$

$$f[x_0, x_1] = \frac{4 - 1}{2 - 1} = 3, \quad f[x_1, x_2] = \frac{9 - 4}{3 - 2} = 5$$

$$f[x_0, x_1, x_2] = \frac{5 - 3}{3 - 1} = 1$$

$$P(2,5) = 1 + 3(2,5 - 1) + 1(2,5 - 1)(2,5 - 2) = 6,25$$

Resultado: $P(2,5) = 6,25$

```

1 x_points <- c(1, 2, 3)
2 y_points <- c(1, 4, 9)
3
4 resultado <- newton_interpol(x_points, y_points, 2.5)
5 cat("P(2.5) =", resultado$result, "\n")
6 cat("Tabla de diferencias:\n")
7 print(resultado$tabla)
```

Apunte

La interpolación de Newton es especialmente útil cuando se agregan nuevos puntos, ya que no requiere recalcular todo el polinomio.

14.4 Implementación en R

A continuación se muestra un ejemplo de interpolación polinómica usando R.

```

1 # Datos
2
3
4 x <- c(1, 2, 4)
5 y <- c(2, 3, 1)
6
7 # Polinomio interpolante
8
9 datos <- data.frame(x, y)
10 modelo <- lm(y ~ poly(x, 2, raw = TRUE), data = datos)
11
12 # Mostrar coeficientes
13
14 coef(modelo)
15
16 # Gráfica
17
18 xx <- seq(1, 4, length.out = 100)
19 yy <- predict(modelo, newdata = data.frame(x = xx))
20
21 plot(x, y, pch = 19, col = "blue")
22 lines(xx, yy, col = "red", lwd = 2)

```

Listing 14.1: Interpolación polinómica en R

14.5 Interpolación Cuadrática

La interpolación cuadrática utiliza un polinomio de grado dos para aproximar una función a partir de tres puntos.

Sea el polinomio: [$P(x) = ax^2 + bx + c$.]

Imponiendo las condiciones de interpolación $P(x_i) = y_i$, $i = 0, 1, 2$, se obtiene el sistema lineal:

La solución del sistema proporciona los coeficientes del polinomio interpolante.

Ejercicio

Para tres puntos, buscamos $P(x) = ax^2 + bx + c$ que pase por los puntos. Resolvemos el sistema:

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

```

1 interpolacion_cuadratica <- function(x_points, y_points, x) {
2   # Construir matriz del sistema
3   A <- matrix(c(
4     x_points[1]^2, x_points[1], 1,

```

```

5   x_points[2]^2, x_points[2], 1,
6   x_points[3]^2, x_points[3], 1
7 ), nrow = 3, byrow = TRUE)
8
9 b <- matrix(y_points, ncol = 1)
10
11 # Resolver sistema
12 coef <- solve(A, b)
13
14 # Evaluar en x
15 y <- coef[1]*x^2 + coef[2]*x + coef[3]
16
17 return(list(y = y, coef = coef))
18 }
```

Apunte

La interpolación cuadrática suele ofrecer mejores resultados que la lineal cuando la función presenta curvatura.

14.6 Interpolación por Tramos

Cuando se utilizan polinomios de alto grado sobre muchos puntos, pueden aparecer oscilaciones indeseadas. Para evitar este problema se emplean métodos de interpolación por tramos.

14.6.1 Splines Cúbicos

Un spline cúbico es una función formada por polinomios de grado tres definidos en cada subintervalo $[x_i, x_{i+1}]$: $[S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3]$.

Estas funciones satisfacen las condiciones:

- $S_i(x_i) = y_i$ (interpolación).
- Continuidad de $S(x)$ en todos los nodos.
- Continuidad de la primera derivada.
- Continuidad de la segunda derivada.

En el caso de splines naturales, se imponen las condiciones de frontera: $[S_0''(x_0) = 0, \quad S_{n-1}''(x_n) = 0]$

Apunte

Los splines cúbicos proporcionan aproximaciones suaves y estables, ampliamente usadas en ingeniería y ciencias aplicadas.

14.7 Errores de Interpolación

El error de interpolación para un polinomio de grado n viene dado por: [$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^n (x - x_i)$,]

para algún ξ en el intervalo de interpolación.

Si se define: [$M_{n+1} = \max |f^{(n+1)}(x)|$,]

entonces la cota del error es: [$|f(x) - p_n(x)| \leq \frac{M_{n+1}}{(n+1)!} |\prod_{i=0}^n (x - x_i)|$.]

Apunte

El aumento del grado del polinomio no siempre reduce el error.
En algunos casos aparece el fenómeno de Runge.

El error de interpolación está dado por: [$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^n (x - x_i)$,]
para algún ξ en el intervalo de interpolación.

Apunte

El aumento del grado del polinomio no siempre mejora la aproximación. En ciertos casos aparece el **fenómeno de Runge**.

14.8 Interpolación por Tramos

Para evitar oscilaciones indeseadas, se emplean métodos por tramos, como la **interpolación spline**, que aproxima los datos mediante polinomios de bajo grado en subintervalos.

14.9 Conclusiones

La interpolación numérica es una herramienta esencial en ciencia e ingeniería. La elección del método adecuado depende del número de datos, la precisión requerida y la estabilidad numérica del problema.

En el siguiente capítulo se estudiarán los métodos de interpolación spline y sus aplicaciones prácticas.

CAPÍTULO 15

Lighthouse y Apache JMeter

15.1 Introducción a las Pruebas de Rendimiento

Las pruebas de rendimiento son esenciales en el desarrollo web moderno para garantizar que las aplicaciones sean rápidas, eficientes y escalables. Dos herramientas fundamentales en este ámbito son:

- **Lighthouse:** Herramienta de Google para auditoría de calidad web
- **Apache JMeter:** Herramienta de código abierto para pruebas de carga y estrés

Apunte

¿Por qué son importantes las pruebas de rendimiento?

- Mejoran la experiencia del usuario
- Reducen la tasa de rebote
- Mejoran el SEO
- Optimizan costos de infraestructura
- Previenen fallos en producción

15.2 Google Lighthouse

15.2.1 ¿Qué es Lighthouse?

Lighthouse es una herramienta automatizada de código abierto desarrollada por Google para mejorar la calidad de las páginas web. Realiza auditorías en las siguientes áreas:

Categoría	¿Qué mide?
Rendimiento	Velocidad de carga y optimización
Accesibilidad	Compatibilidad con usuarios discapacitados
Mejores Prácticas	Seguridad y estándares web
SEO	Optimización para motores de búsqueda
PWA	Características de Progressive Web App

15.2.2 ¿Para qué sirve Lighthouse?

- **Análisis de rendimiento:** Mide tiempos de carga, First Contentful Paint (FCP), Largest Contentful Paint (LCP), etc.
- **Auditoría SEO:** Verifica optimizaciones para motores de búsqueda
- **Validación de accesibilidad:** Asegura que el sitio sea usable por todos
- **Verificación de PWA:** Comprueba características de Progressive Web Apps
- **Recomendaciones específicas:** Sugiere mejoras concretas

15.2.3 Métricas Clave de Lighthouse

Core Web Vitals

$$\text{Performance Score} = f(\text{LCP}, \text{FID}, \text{CLS})$$

Donde:

- **LCP (Largest Contentful Paint):** Tiempo para renderizar el contenido principal (< 2,5s óptimo)
- **FID (First Input Delay):** Tiempo de respuesta a la primera interacción (< 100ms óptimo)
- **CLS (Cumulative Layout Shift):** Estabilidad visual (< 0,1 óptimo)

15.2.4 Cómo usar Lighthouse

```

1 # Instalar Lighthouse globalmente
2 npm install -g lighthouse
3
4 # Ejecutar auditoria básica
5 lighthouse https://ejemplo.com --output=html --output-path=./report.html
6
7 # Ejecutar solo categoria de rendimiento
8 lighthouse https://ejemplo.com --only-categories=performance
9
10 # Configurar emulación de dispositivo móvil
11 lighthouse https://ejemplo.com --emulated-form-factor=mobile
12
13 # Configurar ancho de banda limitado

```

```
14 lighthouse https://ejemplo.com --throttling.cpuSlowdownMultiplier=4
```

Listing 15.1: Usando Lighthouse desde línea de comandos

```

1 const lighthouse = require('lighthouse');
2 const chromeLauncher = require('chrome-launcher');

3
4 async function runAudit(url) {
5   const chrome = await chromeLauncher.launch({chromeFlags: ['--headless']});
6   const options = {
7     logLevel: 'info',
8     output: 'html',
9     onlyCategories: ['performance'],
10    port: chrome.port
11  };
12
13  const runnerResult = await lighthouse(url, options);
14
15  // Imprimir resultados
16  console.log('Puntaje Performance:', runnerResult.lhr.categories.performance
17    .score * 100);
18  console.log('LCP:', runnerResult.lhr.audits['largest-contentful-paint'].displayValue);
19  console.log('FID:', runnerResult.lhr.audits['first-input-delay'].displayValue);
20  console.log('CLS:', runnerResult.lhr.audits['cumulative-layout-shift'].displayValue);
21
22  await chrome.kill();
23  return runnerResult;
24 }
25 runAudit('https://ejemplo.com');
```

Listing 15.2: Usando Lighthouse programáticamente en Node.js

15.2.5 Interpretación de Resultados

Apunte

Puntajes de Lighthouse:

- 90-100: Excelente
- 50-89: Necesita mejora
- 0-49: Pobre

15.3 Apache JMeter

15.3.1 ¿Qué es Apache JMeter?

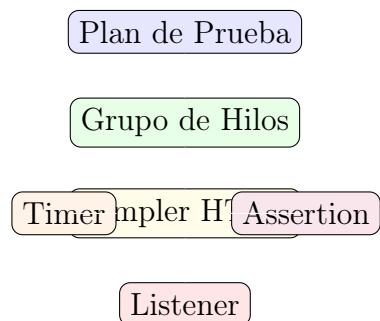
Apache JMeter es una aplicación Java de código abierto diseñada para realizar pruebas de carga y medición de rendimiento. Originalmente creado para probar aplicaciones web, ahora soporta múltiples protocolos.

15.3.2 Características Principales

- **Multi-protocolo:** HTTP, HTTPS, SOAP, REST, FTP, JDBC, LDAP, etc.
- **Pruebas de carga:** Simula múltiples usuarios concurrentes
- **Pruebas de estrés:** Evalúa límites del sistema
- **Pruebas de resistencia:** Verifica estabilidad a lo largo del tiempo
- **Reportes gráficos:** Genera resultados en diversos formatos
- **Extensible:** Plugins y componentes personalizables

15.3.3 Arquitectura de JMeter

Plan de Prueba → Grupos de Hilos → Samplers → Listeners



15.3.4 Configuración Básica de una Prueba

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <jmeterTestPlan version="1.2" properties="5.0" jmeter="5.4.1">
3  <hashTree>
4      <TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="Prueba de
5          Carga API">
6          <stringProp name="TestPlan.comments"></stringProp>
7          <boolProp name="TestPlan.functional_mode">false</boolProp>
8          <boolProp name="TestPlan.serialize_threadgroups">false</boolProp>
9          <elementProp name="TestPlan.user_defined_variables" elementType="Arguments">
10             <collectionProp name="Arguments.arguments"/>
11         </elementProp>
12     </TestPlan>
13     <hashTree>
14         <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname
15             ="Usuarios Concurrentes">
  
```

```

14      <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
15      <elementProp name="ThreadGroup.main_controller" elementType="LoopController">
16          <boolProp name="LoopController.continue_forever">false</boolProp>
17          <stringProp name="LoopController.loops">10</stringProp>
18      </elementProp>
19      <stringProp name="ThreadGroup.num_threads">100</stringProp>
20      <stringProp name="ThreadGroup.ramp_time">60</stringProp>
21      <longProp name="ThreadGroup.start_time">1640995200000</longProp>
22      <longProp name="ThreadGroup.end_time">1640995200000</longProp>
23      <boolProp name="ThreadGroup.scheduler">false</boolProp>
24  </ThreadGroup>
25  <hashTree>
26      <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="Request API">
27          <elementProp name="HTTPSampler.Arguments" elementType="Arguments">
28              <collectionProp name="Arguments.arguments"/>
29          </elementProp>
30          <stringProp name="HTTPSampler.domain">api.ejemplo.com</stringProp>
31          <stringProp name="HTTPSampler.port">443</stringProp>
32          <stringProp name="HTTPSampler.protocol">https</stringProp>
33          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
34          <stringProp name="HTTPSampler.path">/v1/data</stringProp>
35          <stringProp name="HTTPSampler.method">GET</stringProp>
36      </HTTPSamplerProxy>
37      <hashTree>
38          <ResponseAssertion guiclass="AssertionGui" testclass="ResponseAssertion" testname="Validar Status 200">
39              <collectionProp name="Assertion.test_strings">
40                  <stringProp name="49586">200</stringProp>
41              </collectionProp>
42              <stringProp name="Assertion.test_field">Assertion.response_code</stringProp>
43                  <boolProp name="Assertion.assume_success">false</boolProp>
44                  <intProp name="Assertion.test_type">16</intProp>
45              </ResponseAssertion>
46              <hashTree/>
47          </hashTree>
48      </hashTree>
49  </hashTree>
50  </jmeterTestPlan>

```

Listing 15.3: Archivo JMX básico para JMeter

15.3.5 Métricas Clave en JMeter

Métrica	Descripción
Throughput	Peticiones por segundo procesadas
Response Time	Tiempo promedio de respuesta
Error Rate	Porcentaje de peticiones fallidas
90th Percentile	90 % de las respuestas son más rápidas que este valor
Active Threads	Usuarios concurrentes simulados
Bytes/Second	Ancho de banda consumido
Latency	Tiempo hasta el primer byte

15.3.6 Ejecución de Pruebas desde Línea de Comandos

```

1 # Ejecutar prueba básica
2 jmeter -n -t prueba.jmx -l resultados.jtl
3
4 # Ejecutar con gráficos en tiempo real
5 jmeter -n -t prueba.jmx -l resultados.jtl -e -o reporte/
6
7 # Especificar número de hilos y tiempo
8 jmeter -n -t prueba.jmx -Jusers=100 -Jduration=300 -l resultados.jtl
9
10 # Ejecutar en modo servidor distribuido
11 jmeter -n -t prueba.jmx -R 192.168.1.101,192.168.1.102 -l resultados.jtl
12
13 # Generar reporte HTML
14 jmeter -g resultados.jtl -o reporte_html/

```

Listing 15.4: Comandos para ejecutar JMeter

15.4 Comparación: Lighthouse vs JMeter

Aspecto	Lighthouse	JMeter
Tipo de prueba	Auditoría de calidad	Pruebas de carga
Enfoque	Frontend y UX	Backend y infraestructura
Usuarios simulados	1 usuario	Miles de usuarios
Métricas	Rendimiento UX	Rendimiento sistema
Costo	Gratis	Gratis
Complejidad	Baja	Media-Alta
Escenario ideal	Optimización página	Validación escalabilidad

15.5 Ejercicios Prácticos

15.5.1 Ejercicio 1: Auditoría con Lighthouse

Objetivo: Realizar una auditoría completa de un sitio web y analizar resultados.

```

1 #!/bin/bash
2 # script_lighthouse.sh
3
4 URL=$1
5 OUTPUT_DIR="./lighthouse_reports"
6 DATE=$(date +"%Y%m%d_%H%M%S")
7
8 # Crear directorio de salida
9 mkdir -p $OUTPUT_DIR
10
11 # Ejecutar Lighthouse para multiples categorias
12 lighthouse $URL \
13   --output=html,json,csv \
14   --output-path="$OUTPUT_DIR/report_$DATE" \
15   --chrome-flags="--headless --no-sandbox" \
16   --only-categories=performance,accessibility,best-practices,seo
17
18 # Extraer metricas clave del JSON
19 SCORE_PERFORMANCE=$(jq '.categories.performance.score' $OUTPUT_DIR/report_${DATE}.report.json)
20 SCORE_ACCESSIBILITY=$(jq '.categories.accessibility.score' $OUTPUT_DIR/report_${DATE}.report.json)
21 LCP=$(jq '.audits["largest-contentful-paint"].numericValue' $OUTPUT_DIR/report_${DATE}.report.json)
22
23 echo "==== RESULTADOS LIGHTHOUSE ===="
24 echo "Performance Score: $(echo "$SCORE_PERFORMANCE * 100" | bc)"
25 echo "Accessibility Score: $(echo "$SCORE_ACCESSIBILITY * 100" | bc)"
26 echo "LCP: $LCP ms"
27 echo "Reporte generado en: $OUTPUT_DIR/report_${DATE}.report.html"

```

Listing 15.5: Script para auditoría automatizada

15.5.2 Ejercicio 2: Prueba de Carga con JMeter

Objetivo: Crear una prueba de carga para una API REST y analizar resultados.

```

1 import org.apache.jmeter.control.LoopController
2 import org.apache.jmeter.engine.StandardJMeterEngine
3 import org.apache.jmeter.protocol.http.sampler.HTTPSamplerProxy
4 import org.apache.jmeter.reporters.ResultCollector
5 import org.apache.jmeter.reporters.Summariser
6 import org.apache.jmeter.testelement.TestPlan
7 import org.apache.jmeter.threads.ThreadGroup
8 import org.apache.jmeter.util.JMeterUtils
9 import org.apache.jorphan.collections.HashTree
10
11 // Configurar JMeter
12 JMeterUtils.loadJMeterProperties("/path/to/jmeter.properties")
13 JMeterUtils.initLocale()
14

```

```

15 // Crear rbol de prueba
16 HashTree testPlanTree = new HashTree()
17
18 // Crear plan de prueba
19 TestPlan testPlan = new TestPlan("Prueba API REST")
20 testPlanTree.add(testPlan)
21
22 // Crear grupo de hilos
23 ThreadGroup threadGroup = new ThreadGroup()
24 threadGroup.setName("Usuarios API")
25 threadGroup.setNumThreads(50)
26 threadGroup.setRampUp(30)
27 threadGroup.setSamplerController(loopController)
28
29 LoopController loopController = new LoopController()
30 loopController.setLoops(100)
31 loopController.setFirst(true)
32 threadGroup.setSamplerController(loopController)
33
34 HashTree threadGroupHashTree = testPlanTree.add(testPlan, threadGroup)
35
36 // Crear sampler HTTP
37 HTTSPamplerProxy httpSampler = new HTTSPamplerProxy()
38 httpSampler.setDomain("api.ejemplo.com")
39 httpSampler.setPort(443)
40 httpSampler.setPath("/v1/users")
41 httpSampler.setMethod("GET")
42 httpSampler.setProtocol("https")
43
44 threadGroupHashTree.add(httpSampler)
45
46 // Configurar colector de resultados
47 Summariser summariser = null
48 ResultCollector logger = new ResultCollector(summariser)
49 logger.setFilename("resultados_api.jtl")
50 testPlanTree.add(testPlanTree.getArray()[0], logger)
51
52 // Ejecutar prueba
53 StandardJMeterEngine jmeter = new StandardJMeterEngine()
54 jmeter.configure(testPlanTree)
55 jmeter.run()
56
57 println "Prueba completada. Resultados en resultados_api.jtl"

```

Listing 15.6: Script de prueba de carga con JMeter

15.6 Análisis de Resultados

15.6.1 Interpretación de Métricas

Análisis de Lighthouse

```

1 const fs = require('fs');
2
3 function analyzeLighthouseReport(reportPath) {
4   const report = JSON.parse(fs.readFileSync(reportPath, 'utf8'));
5
6   const analysis = {
7     url: report.finalUrl,
8     fetchTime: report.fetchTime,
9     scores: {},
10    opportunities: [],
11    diagnostics: []
12  };
13
14 // Extraer puntajes por categoria
15 for (const [category, data] of Object.entries(report.categories)) {
16   analysis.scores[category] = Math.round(data.score * 100);
17 }
18
19 // Identificar oportunidades de mejora
20 for (const [auditId, audit] of Object.entries(report.audits)) {
21   if (audit.score !== null && audit.score < 0.9) {
22     analysis.opportunities.push({
23       id: auditId,
24       title: audit.title,
25       score: audit.score * 100,
26       description: audit.description,
27       impact: audit.details?.overallSavingsMs || 0
28     });
29   }
30 }
31
32 // Ordenar por impacto
33 analysis.opportunities.sort((a, b) => b.impact - a.impact);
34
35 return analysis;
36 }
37
38 // Uso
39 const results = analyzeLighthouseReport('./report.json');
40 console.log('Puntaje Performance:', results.scores.performance);
41 console.log('Principales oportunidades:');
42 results.opportunities.slice(0, 5).forEach(opp => {
43   console.log(`- ${opp.title}: ${opp.score}% (Ahorro: ${opp.impact}ms)`);
44 });

```

Listing 15.7: Análisis programático de resultados Lighthouse

Análisis de JMeter

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 def analyze_jmeter_results(jtl_file):
5     # Leer archivo JTL
6     df = pd.read_csv(jtl_file, delimiter=',')
7
8     # Calcular métricas básicas
9     total_requests = len(df)
10    successful_requests = df[df['success'] == True].shape[0]
11    failed_requests = df[df['success'] == False].shape[0]
12
13    metrics = {
14        'total_requests': total_requests,
15        'success_rate': (successful_requests / total_requests) * 100,
16        'error_rate': (failed_requests / total_requests) * 100,
17        'avg_response_time': df['elapsed'].mean(),
18        'median_response_time': df['elapsed'].median(),
19        '90th_percentile': df['elapsed'].quantile(0.9),
20        '95th_percentile': df['elapsed'].quantile(0.95),
21        'throughput': total_requests / (df['timeStamp'].max() - df['timeStamp']
22        ].min()) * 1000
23    }
24
25
26    return metrics
27
28
29 def plot_jmeter_metrics(df):
30     fig, axes = plt.subplots(2, 2, figsize=(12, 10))
31
32     # Gráfico 1: Tiempo de respuesta a lo largo del tiempo
33     axes[0, 0].plot(df['timeStamp'], df['elapsed'])
34     axes[0, 0].set_title('Response Time Over Time')
35     axes[0, 0].set_xlabel('Time (ms)')
36     axes[0, 0].set_ylabel('Response Time (ms)')
37
38     # Gráfico 2: Throughput por minuto
39     df['minute'] = df['timeStamp'] // 60000
40     throughput = df.groupby('minute').size()
41     axes[0, 1].bar(throughput.index, throughput.values)
42     axes[0, 1].set_title('Throughput per Minute')
43     axes[0, 1].set_xlabel('Minute')
44     axes[0, 1].set_ylabel('Requests')
45
46     # Gráfico 3: Distribución de tiempos de respuesta
47     axes[1, 0].hist(df['elapsed'], bins=50, edgecolor='black')
48     axes[1, 0].set_title('Response Time Distribution')
49     axes[1, 0].set_xlabel('Response Time (ms)')
50     axes[1, 0].set_ylabel('Frequency')

```

```

49 # Gráfico 4: Códigos de respuesta
50 status_counts = df['responseCode'].value_counts()
51 axes[1, 1].pie(status_counts.values, labels=status_counts.index, autopct=
52 '%.1f %%')
53 axes[1, 1].set_title('Response Codes Distribution')
54
55 plt.tight_layout()
56 plt.savefig('jmeter_analysis.png')
57 plt.show()

58 # Uso
59 metrics = analyze_jmeter_results('results.jtl')
60 print("Métricas JMeter:")
61 for key, value in metrics.items():
62     print(f'{key}: {value:.2f}')

```

Listing 15.8: Análisis de resultados JMeter con Python

Apunte

Mejores Prácticas para Pruebas de Rendimiento:

1. **Establecer línea base:** Medir rendimiento actual antes de optimizar
2. **Definir objetivos claros:** ¿Qué consideramos aceptable?
3. **Probar en ambiente similar a producción**
4. **Monitorear recursos del servidor** durante las pruebas
5. **Automatizar las pruebas** en el pipeline de CI/CD
6. **Documentar resultados** y tendencias
7. **Probar progresivamente:** Empezar con pocos usuarios, aumentar gradualmente

15.7 Casos de Uso en la Industria

15.7.1 E-commerce

- **Lighthouse:** Optimizar tiempos de carga durante campañas
- **JMeter:** Simular miles de usuarios en Black Friday

15.7.2 Aplicaciones Financieras

- **Lighthouse:** Asegurar accesibilidad para todos los usuarios
- **JMeter:** Validar que el sistema soporte picos de transacciones

15.7.3 Plataformas de Streaming

- **Lighthouse**: Mejorar First Contentful Paint para contenido inmediato
- **JMeter**: Probar escalabilidad durante estrenos importantes

15.8 Conclusión

Lighthouse y JMeter son herramientas complementarias que abordan diferentes aspectos del rendimiento web:

- **Lighthouse** se enfoca en la **experiencia del usuario** y la **calidad técnica** del frontend
- **JMeter** se centra en la **escalabilidad** y **rendimiento** del backend bajo carga

La combinación de ambas herramientas proporciona una visión completa del rendimiento de una aplicación web, desde la experiencia del usuario individual hasta la capacidad del sistema para manejar tráfico masivo.

CAPÍTULO 16

Eigenvalores y Eigenvectores en Optimización

16.1 ¿Qué son los Eigenvalores y Eigenvectores?

16.1.1 Intuición geométrica

Imaginemos que tenemos una transformación lineal representada por una matriz A . Cuando aplicamos esta transformación a un vector \mathbf{v} , generalmente cambia tanto su dirección como su magnitud. Sin embargo, existen **vectores especiales** que solo cambian su magnitud, manteniendo su dirección original. Estos son los **eigenvectores**.

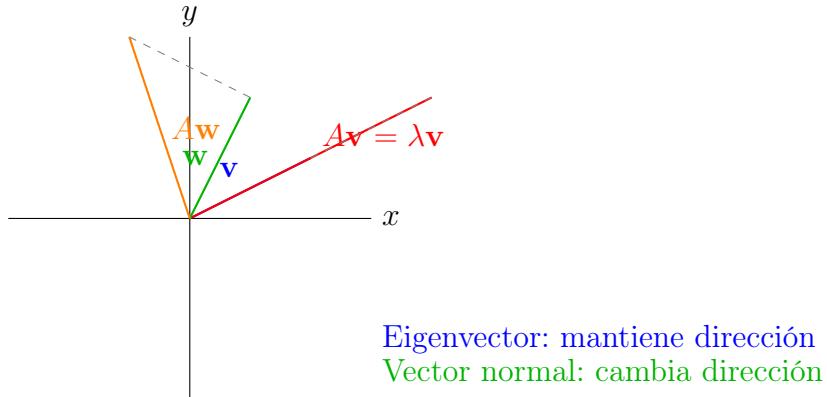


Figura 16.1: Representación gráfica de eigenvectores

16.1.2 Definición formal

Sea $A \in \mathbb{R}^{n \times n}$ una matriz cuadrada. Un vector no nulo $\mathbf{v} \in \mathbb{R}^n$ se llama **eigenvector** de A si existe un escalar $\lambda \in \mathbb{R}$ tal que:

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

El escalar λ se denomina **eigenvalor** asociado al eigenvector \mathbf{v} .

La ecuación fundamental se puede reescribir como:

$$(A - \lambda I)\mathbf{v} = \mathbf{0}$$

donde I es la matriz identidad. Para que exista solución no trivial ($\mathbf{v} \neq \mathbf{0}$), se requiere:

$$\det(A - \lambda I) = 0$$

16.1.3 Interpretación física y geométrica

- **Direcciones principales:** Los eigenvectores indican las direcciones en las que la transformación actúa de manera simple (solo escalando).
- **Factor de escala:** El eigenvalor λ indica cuánto se estira o encoge el vector:
 - Si $|\lambda| > 1$: expansión
 - Si $|\lambda| < 1$: contracción
 - Si $\lambda < 0$: inversión de dirección
- **Estabilidad:** En sistemas dinámicos, los eigenvalores determinan la estabilidad del sistema.

16.2 Cálculo de Eigenvalores y Eigenvectores

16.2.1 Método algebraico

Para encontrar los eigenvalores de una matriz A :

1. **Ecuación característica:** $\det(A - \lambda I) = 0$
2. **Polinomio característico:** $p(\lambda) = \det(A - \lambda I)$
3. **Raíces:** Resolver $p(\lambda) = 0$ para obtener $\lambda_1, \lambda_2, \dots, \lambda_n$
4. **Eigenvectores:** Para cada λ_i , resolver $(A - \lambda_i I)\mathbf{v}_i = \mathbf{0}$

Matriz 2×2 general:

$$\text{Sea } A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}.$$

1. Ecuación característica:

$$\det \begin{pmatrix} a - \lambda & b \\ c & d - \lambda \end{pmatrix} = 0$$

2. Polinomio característico:

$$(a - \lambda)(d - \lambda) - bc = 0$$

$$\lambda^2 - (a + d)\lambda + (ad - bc) = 0$$

3. Solución:

$$\lambda = \frac{(a + d) \pm \sqrt{(a + d)^2 - 4(ad - bc)}}{2}$$

16.2.2 Casos especiales

Apunte

Propiedades importantes:

- Matrices simétricas ($A = A^T$): Todos los eigenvalores son reales
- Matrices definidas positivas: Todos los eigenvalores son positivos
- Matrices ortogonales: Todos los eigenvalores tienen módulo 1
- Traza y determinante:

$$\sum_{i=1}^n \lambda_i = \text{tr}(A), \quad \prod_{i=1}^n \lambda_i = \det(A)$$

Matrices diagonales

Para $A = \text{diag}(d_1, d_2, \dots, d_n)$, los eigenvalores son d_1, d_2, \dots, d_n y los eigenvectores son los vectores canónicos.

Matrices triangulares

Los eigenvalores son los elementos de la diagonal principal.

Matriz diagonal:

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 5 \end{pmatrix}$$

Eigenvalores: $\lambda_1 = 2, \lambda_2 = 5$

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

Verificación:

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

Matrices simétricas 2×2

Para $A = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$:

Eigenvalores:

$$\lambda = \frac{a + c \pm \sqrt{(a - c)^2 + 4b^2}}{2}$$

Matriz simétrica:

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

Ecuación característica:

$$\det \begin{pmatrix} 3 - \lambda & 1 \\ 1 & 3 - \lambda \end{pmatrix} = (3 - \lambda)^2 - 1 = 0$$

Solución:

$$\lambda^2 - 6\lambda + 8 = 0 \Rightarrow \lambda_1 = 4, \lambda_2 = 2$$

Eigenvectores:

- Para $\lambda_1 = 4$: $(A - 4I)\mathbf{v} = \mathbf{0} \Rightarrow \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \mathbf{v} = \mathbf{0} \Rightarrow \mathbf{v}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
- Para $\lambda_2 = 2$: $(A - 2I)\mathbf{v} = \mathbf{0} \Rightarrow \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \mathbf{v} = \mathbf{0} \Rightarrow \mathbf{v}_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

16.3 Eigenvalores en Optimización: La Matriz Hessiana

16.3.1 Introducción a la optimización multivariable

Consideremos una función $f : \mathbb{R}^n \rightarrow \mathbb{R}$ dos veces diferenciable. Para encontrar sus extremos locales:

1. **Puntos críticos:** Resolver $\nabla f(\mathbf{x}) = \mathbf{0}$
2. **Clasificación:** Analizar la matriz Hessiana $H(\mathbf{x})$

La matriz Hessiana contiene todas las segundas derivadas parciales:

$$H(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

16.3.2 Teorema fundamental

Sea $f : \mathbb{R}^n \rightarrow \mathbb{R}$ una función dos veces diferenciable y \mathbf{x}_0 un punto crítico ($\nabla f(\mathbf{x}_0) = \mathbf{0}$). Sean $\lambda_1, \lambda_2, \dots, \lambda_n$ los eigenvalores de $H(\mathbf{x}_0)$. Entonces:

1. Si todos $\lambda_i > 0$: \mathbf{x}_0 es un **mínimo local estricto**
2. Si todos $\lambda_i < 0$: \mathbf{x}_0 es un **máximo local estricto**
3. Si hay $\lambda_i > 0$ y $\lambda_j < 0$: \mathbf{x}_0 es un **punto silla**
4. Si algún $\lambda_i = 0$: El criterio es inconcluso (necesita análisis de orden superior)

Mínimo: $\lambda_1, \lambda_2 > 0$ Máximo: $\lambda_1, \lambda_2 < 0$ Punto silla: $\lambda_1 > 0, \lambda_2 < 0$

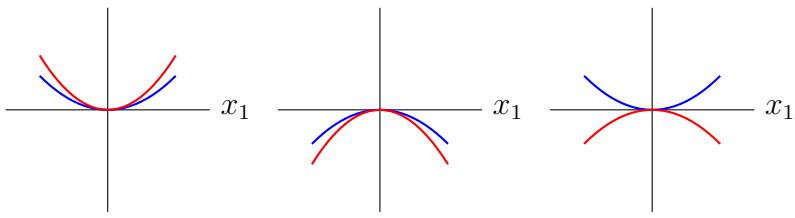


Figura 16.2: Tipos de puntos críticos según eigenvalores

16.3.3 Interpretación geométrica

Los eigenvalores de la Hessiana representan las **curvaturas principales** de la superficie $f(\mathbf{x})$ en el punto crítico:

- **Eigenvalor positivo:** Curvatura hacia arriba en esa dirección
- **Eigenvalor negativo:** Curvatura hacia abajo en esa dirección
- **Eigenvector asociado:** Dirección principal de curvatura

16.3.4 Ejemplos detallados

Clasificación de punto crítico:

Sea $f(x_1, x_2) = x_1^2 + 3x_2^2 + 2x_1x_2$.

1. **Puntos críticos:**

$$\nabla f = \begin{pmatrix} 2x_1 + 2x_2 \\ 6x_2 + 2x_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Sistema:

$$\begin{cases} 2x_1 + 2x_2 = 0 \\ 2x_1 + 6x_2 = 0 \end{cases} \Rightarrow x_1 = x_2 = 0$$

2. **Matriz Hessiana:**

$$H = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 2 & 6 \end{pmatrix}$$

3. **Eigenvalores:**

$$\det(H - \lambda I) = \det \begin{pmatrix} 2 - \lambda & 2 \\ 2 & 6 - \lambda \end{pmatrix} = (2 - \lambda)(6 - \lambda) - 4 = 0$$

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

$$\lambda = \frac{8 \pm \sqrt{64 - 32}}{2} = \frac{8 \pm \sqrt{32}}{2} = 4 \pm 2\sqrt{2}$$

$$\lambda_1 \approx 6,83, \quad \lambda_2 \approx 1,17$$

4. **Clasificación:** Ambos eigenvalores son positivos $\Rightarrow (0, 0)$ es un **mínimo local**.

Función con punto silla:

Sea $f(x_1, x_2) = x_1^2 - x_2^2$ (paraboloide hiperbólico).

1. Punto crítico: $(0, 0)$
2. Hessiana: $H = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}$
3. Eigenvalores: $\lambda_1 = 2, \lambda_2 = -2$
4. Clasificación: Punto silla (un eigenvalor positivo, otro negativo)

La función sube en dirección x_1 y baja en dirección x_2 .

16.4 Implementación Computacional en Python

16.4.1 Cálculo de eigenvalores con NumPy

```

1 import numpy as np
2
3 # Definir una matriz
4 A = np.array([[3, 1],
5               [1, 3]])
6
7 # Calcular eigenvalores y eigenvectores
8 eigenvalues, eigenvectors = np.linalg.eig(A)
9
10 print("Matriz A:")
11 print(A)
12 print("\nEigenvalores:")
13 print(eigenvalues)
14 print("\nEigenvectores (columnas):")
15 print(eigenvectors)
16
17 # Verificar que Av = v para cada par
18 for i in range(len(eigenvalues)):
19     v = eigenvectors[:, i].reshape(-1, 1)
20     = eigenvalues[i]
21
22     # Calcular Av
23     Av = A @ v
24
25     # Calcular v
26     v = * v
27
28     print(f"\nVerificación para {i+1} = {:.4f}:")
29     print(f"Av = {Av.flatten()}")
30     print(f"v = {v.flatten()}")
31     print(f"Son iguales? {np.allclose(Av, v)}")
```

Listing 16.1: Cálculo básico de eigenvalores y eigenvectores

16.4.2 Clasificación de puntos críticos

```

1 import numpy as np
2 import sympy as sp
3
4 def analizar_punto_critico(f_str, variables, punto):
5     """
6         Analiza un punto critico de una funcin multivariable
7     """
8     # Convertir string a expresin simblica
9     x, y = sp.symbols(variables)
10    f = eval(f_str)
11
12    # Calcular gradiente
13    grad_f = [sp.diff(f, var) for var in [x, y]]
14
15    # Calcular Hessiana
16    hessian = sp.Matrix([[sp.diff(sp.diff(f, var1), var2)
17                          for var2 in [x, y]]
18                          for var1 in [x, y]])
19
20    # Evaluar Hessiana en el punto
21    hessian_vals = hessian.subs({x: punto[0], y: punto[1]})
22    hessian_np = np.array(hessian_vals).astype(float)
23
24    # Calcular eigenvalores
25    eigenvalues = np.linalg.eigvals(hessian_np)
26
27    # Clasificar
28    if all(eig > 0 for eig in eigenvalues):
29        clasificacion = "MNIMO LOCAL"
30    elif all(eig < 0 for eig in eigenvalues):
31        clasificacion = "MXIMO LOCAL"
32    elif any(eig > 0 for eig in eigenvalues) and any(eig < 0 for eig in
33 eigenvalues):
34        clasificacion = "PUNTO SILLA"
35    else:
36        clasificacion = "INDETERMINADO (necesita anlisis de orden superior)"
37
38    return {
39        'gradiente': grad_f,
40        'hessiana': hessian,
41        'hessiana_en_punto': hessian_vals,
42        'eigenvalores': eigenvalues,
43        'clasificacion': clasificacion
44    }
45
46    # Ejemplo de uso
47    if __name__ == "__main__":
48        # Definir funcin y punto
49        f = "x**2 + 3*y**2 + 2*x*y"

```

```

49 punto = (0, 0)
50
51 # Analizar
52 resultado = analizar_punto_critico(f, "x y", punto)
53
54 print("Función: f(x,y) =", f)
55 print(f"Punto crítico: {punto}")
56 print(f"\nGradiente: {resultado['gradiente']} ")
57 print(f"\nHessiana en general:\n{resultado['hessiana']} ")
58 print(f"\nHessiana en {punto}:\n{resultado['hessiana_en_punto']} ")
59 print(f"\nEigenvalores: {resultado['eigenvalores']} ")
60 print(f"\nClasificación: {resultado['clasificación']} ")

```

Listing 16.2: Clasificación usando la matriz Hessiana

16.4.3 Visualización de superficies y eigenvectores

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.mplot3d import Axes3D
4 from matplotlib import cm
5
6 def visualizar_superficie_con_eigenvectores(f, x_range=(-2, 2), y_range=(-2,
7     2), punto=(0, 0)):
8     """
9         Visualiza una superficie y sus eigenvectores en un punto
10    """
11    # Crear malla
12    x = np.linspace(x_range[0], x_range[1], 100)
13    y = np.linspace(y_range[0], y_range[1], 100)
14    X, Y = np.meshgrid(x, y)
15    Z = f(X, Y)
16
17    # Calcular Hessiana numéricamente (aproximación)
18    def hessiana_aproximada(f, x0, y0, h=1e-5):
19        # Derivadas segundas usando diferencias finitas
20        fxx = (f(x0+h, y0) - 2*f(x0, y0) + f(x0-h, y0)) / h**2
21        fyy = (f(x0, y0+h) - 2*f(x0, y0) + f(x0, y0-h)) / h**2
22        fxy = (f(x0+h, y0+h) - f(x0+h, y0-h) - f(x0-h, y0+h) + f(x0-h, y0-h)) /
23            (4*h**2)
24
25        return np.array([[fxx, fxy], [fxy, fyy]])
26
27    # Calcular eigenvalores y eigenvectores en el punto
28    H = hessiana_aproximada(f, punto[0], punto[1])
29    eigenvalues, eigenvectors = np.linalg.eig(H)
30
31    # Crear figura
32    fig = plt.figure(figsize=(15, 5))
33
34    # Subplot 1: Superficie 3D

```

```

33     ax1 = fig.add_subplot(131, projection='3d')
34     surf = ax1.plot_surface(X, Y, Z, cmap=cm.viridis, alpha=0.8)
35     ax1.scatter(punto[0], punto[1], f(punto[0], punto[1]), color='red', s=100, label='Punto crtico')
36     ax1.set_xlabel('x')
37     ax1.set_ylabel('y')
38     ax1.set_zlabel('f(x,y)')
39     ax1.set_title('Superficie 3D')
40
41     # Subplot 2: Curvas de nivel con eigenvectores
42     ax2 = fig.add_subplot(132)
43     contour = ax2.contour(X, Y, Z, 20, cmap=cm.viridis)
44     ax2.scatter(punto[0], punto[1], color='red', s=100, label='Punto crtico')
45
46     # Dibujar eigenvectores escalados por eigenvalores
47     scale = 0.5
48     for i in range(2):
49         vec = eigenvectors[:, i] * eigenvalues[i] * scale
50         ax2.arrow(punto[0], punto[1], vec[0], vec[1],
51                   head_width=0.1, head_length=0.1, fc='blue', ec='blue',
52                   label=f'{i+1}={eigenvalues[i]:.2f}')
53
54     ax2.set_xlabel('x')
55     ax2.set_ylabel('y')
56     ax2.set_title('Curvas de nivel y direcciones principales')
57     ax2.legend()
58     ax2.axis('equal')
59
60     # Subplot 3: Información numérica
61     ax3 = fig.add_subplot(133)
62     ax3.axis('off')
63
64     info_text = (
65         f"Punto crtico: ({punto[0]}, {punto[1]})\n\n"
66         f"Hessiana:\n{H}\n\n"
67         f"Eigenvalores:\n{n1} = {eigenvalues[0]:.4f}\n{n2} = {eigenvalues[1]:.4f}\n\n"
68         f"Eigenvectores:\n{v1} = [{eigenvectors[0,0]:.3f}, {eigenvectors[1,0]:.3f}]^T\n"
69         f"\n{v2} = [{eigenvectors[0,1]:.3f}, {eigenvectors[1,1]:.3f}]^T\n\n"
70     )
71
72     # Clasificación
73     if all(eig > 0 for eig in eigenvalues):
74         clasif = "MÍNIMO LOCAL"
75         color = 'green'
76     elif all(eig < 0 for eig in eigenvalues):
77         clasif = "MÁXIMO LOCAL"
78         color = 'red'
79     elif any(eig > 0 for eig in eigenvalues) and any(eig < 0 for eig in eigenvalues):

```

```

80     clasif = "PUNTO SILLA"
81     color = 'orange'
82 else:
83     clasif = "INDETERMINADO"
84     color = 'gray'
85
86 info_text += f"Clasificacín:\n{clasif}"
87
88 ax3.text(0.1, 0.5, info_text, transform=ax3.transAxes,
89           fontsize=10, verticalalignment='center',
90           bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5))
91
92 plt.tight_layout()
93 plt.show()
94
95 return eigenvalues, eigenvectors
96
97 # Ejemplo de uso
98 def ejemplo_funcion(x, y):
99     return x**2 + 3*y**2 + 2*x*y
100
101 # Visualizar
102 eigenvals, eigenvecs = visualizar_superficie_con_eigenvectores(
103     ejemplo_funcion,
104     x_range=(-3, 3),
105     y_range=(-3, 3),
106     punto=(0, 0)
107 )

```

Listing 16.3: Visualización 3D de superficies y direcciones principales

16.5 Aplicaciones Avanzadas en Optimización

16.5.1 Optimización restringida y multiplicadores de Lagrange

En problemas con restricciones $g(\mathbf{x}) = 0$, formamos el Lagrangiano:

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$$

La matriz Hessiana aumentada (bordered Hessian) contiene información sobre la optimalidad:

$$H_a = \begin{pmatrix} 0 & \nabla g(\mathbf{x})^T \\ \nabla g(\mathbf{x}) & \nabla^2 \mathcal{L}(\mathbf{x}, \lambda) \end{pmatrix}$$

Los eigenvalores de submatrices de H_a determinan si tenemos un mínimo/máximo restringido.

16.5.2 Métodos de optimización basados en eigenvalores

1. **Método de la potencia:** Encuentra el eigenvalor dominante iterativamente:

$$\mathbf{v}_{k+1} = \frac{A\mathbf{v}_k}{\|A\mathbf{v}_k\|}$$

2. **Análisis de componentes principales (PCA):** En problemas de alta dimensión, PCA usa eigenvalores para reducir dimensionalidad manteniendo máxima varianza.
3. **Optimización en redes neuronales:** La matriz de covarianza de los gradientes tiene eigenvalores que indican la condición del problema de optimización.

Apunte

Relación entre eigenvalores y tasa de convergencia: En métodos de optimización como el descenso del gradiente, la **condición número** $\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$ de la Hessiana afecta la velocidad de convergencia:

- $\kappa \approx 1$: Convergencia rápida (superficie esférica)
- $\kappa \gg 1$: Convergencia lenta (superficie elongada)
- $\kappa \rightarrow \infty$: Posible divergencia

16.6 Ejercicios Resueltos y Propuestos

16.6.1 Ejercicios resueltos

Ejercicio 1: Encuentra eigenvalores y eigenvectores de:

$$A = \begin{pmatrix} 4 & 0 \\ 0 & 7 \end{pmatrix}$$

Solución:

1. Matriz diagonal \Rightarrow eigenvalores = elementos diagonales

$$\lambda_1 = 4, \quad \lambda_2 = 7$$

2. Eigenvectores: vectores canónicos

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

3. Verificación:

$$A\mathbf{v}_1 = \begin{pmatrix} 4 \\ 0 \end{pmatrix} = 4\mathbf{v}_1, \quad A\mathbf{v}_2 = \begin{pmatrix} 0 \\ 7 \end{pmatrix} = 7\mathbf{v}_2$$

Ejercicio 2: Para $B = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$, encuentra eigenvalores y eigenvectores.

Solución:

1. Ecuación característica:

$$\det(B - \lambda I) = \det \begin{pmatrix} 1 - \lambda & 2 \\ 2 & 1 - \lambda \end{pmatrix} = (1 - \lambda)^2 - 4 = 0$$

2. Resolver:

$$(1 - \lambda)^2 = 4 \Rightarrow 1 - \lambda = \pm 2$$

$$\lambda_1 = 1 + 2 = 3, \quad \lambda_2 = 1 - 2 = -1$$

3. Eigenvectores:

- Para $\lambda_1 = 3$: $(B - 3I)\mathbf{v} = \begin{pmatrix} -2 & 2 \\ 2 & -2 \end{pmatrix}\mathbf{v} = \mathbf{0} \Rightarrow \mathbf{v}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
- Para $\lambda_2 = -1$: $(B + I)\mathbf{v} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}\mathbf{v} = \mathbf{0} \Rightarrow \mathbf{v}_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

16.6.2 Ejercicios propuestos

1. **Ejercicio 3:** Para $f(x_1, x_2) = 2x_1^2 + x_2^2 - 2x_1x_2$:

- Calcula la matriz Hessiana
- Encuentra sus eigenvalores
- Clasifica el punto crítico $(0, 0)$

2. **Ejercicio 4:** Determina si $(0, 0)$ es máximo o mínimo para $f(x_1, x_2) = -x_1^2 - 2x_2^2$.

3. **Ejercicio 5:** Verifica que $\mathbf{v} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ es eigenvector de $C = \begin{pmatrix} 3 & 2 \\ 1 & 4 \end{pmatrix}$ y encuentra su eigenvalor.

4. **Ejercicio 6:** Para la función $f(x_1, x_2) = x_1^3 + x_2^3 - 3x_1x_2$:

- Encuentra todos los puntos críticos
- Calcula la Hessiana en cada punto
- Clasifica usando eigenvalores

5. **Ejercicio 7** (Programación): Implementa una función que:

- Reciba una matriz 2×2
- Calcule eigenvalores y eigenvectores
- Verifique que $A\mathbf{v} = \lambda\mathbf{v}$
- Genere una visualización de los vectores transformados

6. **Ejercicio 8:** Demuestra que si A es simétrica, sus eigenvectores correspondientes a eigenvalores distintos son ortogonales.

7. **Ejercicio 9:** Para una matriz A con eigenvalores λ_i , muestra que:

$$\det(A) = \prod_{i=1}^n \lambda_i, \quad \text{tr}(A) = \sum_{i=1}^n \lambda_i$$

8. **Ejercicio 10:** Investiga cómo se usan los eigenvalores en el método PCA (Principal Component Analysis) y su relación con la optimización de varianza.

16.7 Soluciones a ejercicios seleccionados

Ejercicio 3:

1. Hessiana: $H = \begin{pmatrix} 4 & -2 \\ -2 & 2 \end{pmatrix}$
2. Eigenvalores: $\lambda = 3 \pm \sqrt{5} \approx 5,236$ y $0,764$
3. Ambos positivos \Rightarrow mínimo local

Ejercicio 4:

$$H = \begin{pmatrix} -2 & 0 \\ 0 & -4 \end{pmatrix}, \quad \lambda_1 = -2, \lambda_2 = -4$$

Ambos negativos \Rightarrow máximo local

Ejercicio 5:

$$C\mathbf{v} = \begin{pmatrix} 3 & 2 \\ 1 & 4 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \end{pmatrix} = 4 \begin{pmatrix} 2 \\ 1 \end{pmatrix} \Rightarrow \lambda = 4$$

16.8 Resumen y Conclusiones

Los eigenvalores y eigenvectores son herramientas fundamentales en optimización porque:

1. **Clasificación de extremos:** Los signos de los eigenvalores de la Hessiana determinan si un punto crítico es mínimo, máximo o punto silla.
2. **Direcciones principales:** Los eigenvectores indican las direcciones de máxima y mínima curvatura.
3. **Análisis de convergencia:** En métodos iterativos, los eigenvalores determinan la velocidad de convergencia.
4. **Reducción de dimensionalidad:** Técnicas como PCA usan eigenvalores para identificar las direcciones más informativas.
5. **Estabilidad de sistemas:** En sistemas dinámicos, los eigenvalores determinan la estabilidad de puntos de equilibrio.

Apunte

Consejos prácticos:

- Para matrices pequeñas ($n \leq 3$), calcula eigenvalores analíticamente.
- Para matrices grandes, usa métodos numéricos (potencia, QR algorithm).
- En optimización, siempre verifica que la Hessiana sea definida positiva para mínimos.
- Usa visualizaciones para entender la geometría del problema.

16.9 Próximos temas

En el siguiente capítulo exploraremos:

- **Métodos numéricos para cálculo de eigenvalores:** Método de la potencia, algoritmo QR.
- **Descomposición en valores singulares (SVD):** Generalización de eigenvalores para matrices rectangulares.
- **Aplicaciones en aprendizaje automático:** PCA, sistemas de recomendación, reducción de ruido.
- **Optimización con restricciones:** Multiplicadores de Lagrange y condiciones KKT.

Recursos adicionales:

- **Libros:** "Linear Algebra and Its Applications" de Gilbert Strang
- **Software:** NumPy (Python), MATLAB, Julia
- **Visualización:** GeoGebra, Desmos 3D
- **Cursos online:** MIT OpenCourseWare (18.06 Linear Algebra)

CAPÍTULO 17

Cadenas de Markov

Las **cadenas de Markov** constituyen una herramienta fundamental en la programación numérica, la probabilidad y la modelación matemática de sistemas dinámicos aleatorios. Permiten describir la evolución de un sistema que cambia de estado de manera probabilística a lo largo del tiempo.

Este capítulo presenta una introducción conceptual clara a las cadenas de Markov, su formulación matemática y su utilidad en distintos campos de aplicación, manteniendo un enfoque didáctico y computacional.

17.1 ¿Qué es una Cadena de Markov?

Una cadena de Markov es un proceso estocástico discreto que satisface la **propiedad de Markov**: el estado futuro del sistema depende únicamente de su estado actual y no de la historia pasada.

Formalmente, si X_n es una sucesión de variables aleatorias que representan los estados del sistema, se cumple: [$P(X_{n+1} = j \mid X_n = i, X_{n-1} = i_{n-1}, \dots, X_0 = i_0) = P(X_{n+1} = j \mid X_n = i)$.]

Apunte

La propiedad de Markov se resume en la frase: *"el futuro depende solo del presente, no del pasado"*.

17.2 Estados y Transiciones

Una cadena de Markov está formada por:

- Un conjunto finito o numerable de **estados**.
- Probabilidades de transición entre dichos estados.

Si el sistema se encuentra en el estado i en el instante n , la probabilidad de pasar al estado j en el instante $n + 1$ se denota por: [$P_{ij} = P(X_{n+1} = j \mid X_n = i)$.]

17.3 Matriz de Transición

Las probabilidades de transición se organizan en una **matriz de transición**: [$P = (P_{ij})$,] donde cada elemento cumple: [$P_{ij} \geq 0$, $\sum_j P_{ij} = 1$.]

Apunte

Cada fila de la matriz de transición representa una distribución de probabilidad.

17.3.1 Ejemplo

Considérese un sistema con tres estados E_1, E_2, E_3 cuya matriz de transición es: [$P = (0,6 \ 0,3 \ 0,1 \ 0,2 \ 0,5 \ 0,3 \ 0,1 \ 0,4 \ 0,5)$.]

17.4 Distribución de Probabilidad del Sistema

Sea el vector de estado inicial: [$\pi^{(0)} = (\pi_1^{(0)}, \pi_2^{(0)}, \dots, \pi_n^{(0)})$,]

donde $\pi_i^{(0)}$ representa la probabilidad de iniciar en el estado i .

La distribución del sistema en el paso k se obtiene mediante: [$\pi^{(k)} = \pi^{(0)} P^k$.]

17.5 ¿Para qué Sirven las Cadenas de Markov?

Las cadenas de Markov se utilizan para modelar fenómenos en los que existe incertidumbre y dependencia temporal, tales como:

- Modelos de clima y pronóstico.
- Análisis de colas y sistemas de atención.
- Economía y finanzas (cambio de estados económicos).
- Procesos biológicos y genéticos.
- Sistemas de recomendación y algoritmos de búsqueda.

17.6 Estados Estacionarios

Una distribución estacionaria π satisface: [$\pi = \pi P, \sum_i \pi_i = 1$.]

Esta distribución describe el comportamiento a largo plazo de la cadena, cuando existe.

Apunte

El cálculo de la distribución estacionaria está estrechamente relacionado con el cálculo de eigenvalores y eigenvectores.

17.7 Implementación Básica en R

A continuación se muestra cómo trabajar con cadenas de Markov utilizando únicamente funciones base de R.

```

1 # Matriz de transición
2
3 P <- matrix(c(0.6,0.3,0.1,
4 0.2,0.5,0.3,
5 0.1,0.4,0.5), nrow = 3, byrow = TRUE)
6
7 # Vector de estado inicial
8
9 pi0 <- c(1, 0, 0)
10
11 # Distribución después de 5 pasos
12
13 pi5 <- pi0 %*% (P %^% 5)
14 pi5
15

```

Listing 17.1: Cadena de Markov básica en R

17.8 Relación con Programación Numérica

El estudio de cadenas de Markov involucra operaciones matriciales, cálculo de potencias de matrices y eigenvalores, lo que las convierte en un excelente ejemplo de aplicación directa de los métodos numéricos.

17.9 Conclusiones

Las cadenas de Markov permiten modelar y analizar sistemas dinámicos probabilísticos de manera sencilla y potente. Su formulación matricial facilita su implementación computacional y su análisis teórico dentro de la programación numérica.

CAPÍTULO 18

Eigenvalues y Eigenvectors Aplicados

EJERCICIO 1: Modificación de la Matriz de Transición

1. Instalación y Configuración

```
1 # Importar librerias necesarias
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from scipy import linalg
6 import pandas as pd
7
8 # Configuracion de visualizacion
9 plt.rcParams['figure.figsize'] = (12, 8)
10 plt.rcParams['font.size'] = 10
11 sns.set_style("whitegrid")
12
13 print("Librerias importadas correctamente")
14 print(f"NumPy versin: {np.__version__}")
```

7

2. Definición del Problema

2.1. 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, y que los turistas en Taquile se queden más tiempo (menor probabilidad de regresar inmediatamente a Puno).

2.2. Pregunta de Investigación

- ¿Cuánto aumentó el porcentaje de turistas en Taquile?
- ¿Cambió el hub principal?
- ¿Qué tan rápido converge el sistema con estos cambios?

2.3. Enfoque Matemático

Modelaremos el flujo de turistas como una Cadena de Markov usando una matriz de transición T , donde:

$$T_{ij} = P(\text{moverse del destino } i \text{ al destino } j)$$

El eigenvector asociado al eigenvalue dominante ($\lambda = 1$) nos dará la distribución estacionaria.

3. Construcción de la Matriz de Transición

La matriz de transición está basada en los cambios por la inversión:

- Aumenta de 25 % a 35 % la probabilidad de ir de Uros a Taquile.
- Reduce de 50 % a 40 % la probabilidad de regresar de Uros a Puno Ciudad.
- Reduce de 40 % a 30 % la probabilidad de regresar de Taquile a Puno Ciudad.
- Aumenta de 30 % a 40 % la probabilidad de quedarse en Taquile.

```

1 # Definicion de destinos
2 destinos = ['Puno Ciudad', 'Islas Uros', 'Taquile', 'Amantan']
3 n_destinos = len(destinos)
4
5 # Matriz de transicion T (MODIFICADA EJERCICIO 1)
6 # T[i, j] = probabilidad de moverse del destino i al destino j
7 T = np.array([
8     [0.25, 0.45, 0.20, 0.10], # Desde Puno Ciudad
9     [0.40, 0.15, 0.35, 0.10], # Desde Islas Uros (MODIFICADO)
10    [0.30, 0.10, 0.40, 0.20], # Desde Taquile (MODIFICADO)
11    [0.55, 0.15, 0.10, 0.20] # Desde Amantan
12])
13
14 # Crear DataFrame para mejor visualizacion
15 df_matriz = pd.DataFrame(T,
16                         index=destinos,
17                         columns=destinos)
18
19 print("Matriz de Transicion T (Modificada):")
20 print("-" * 70)
21 print(df_matriz)
22 print("\nCada fila representa la distribucion de probabilidad de moverse")
23 print("desde un destino (fila) hacia otros destinos (columnas)")
24 print("\nVerificacion: Suma de cada fila debe ser 1.0")
25 print("-" * 70)
26 for i, dest in enumerate(destinos):
27     suma = T[i,:].sum()
28     status = "OK" if abs(suma - 1.0) < 0.001 else "ERROR"
29     print(f"{status} {dest :15}: suma = {suma :.3f}")

```

Ejecución:

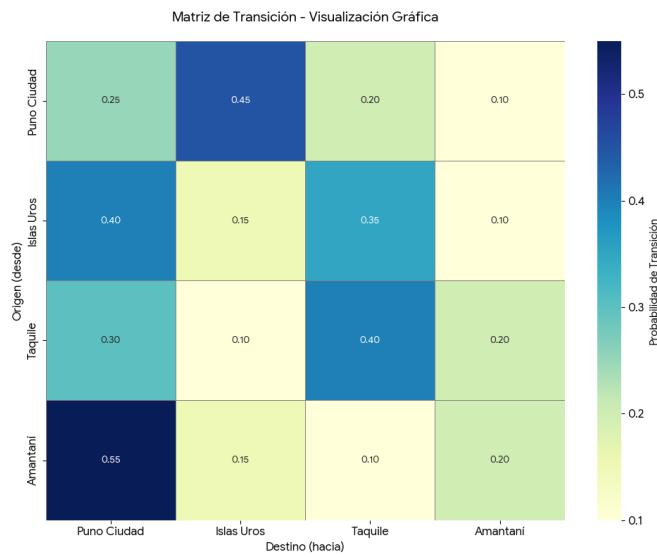


Figura 18.1: Imagen de Python

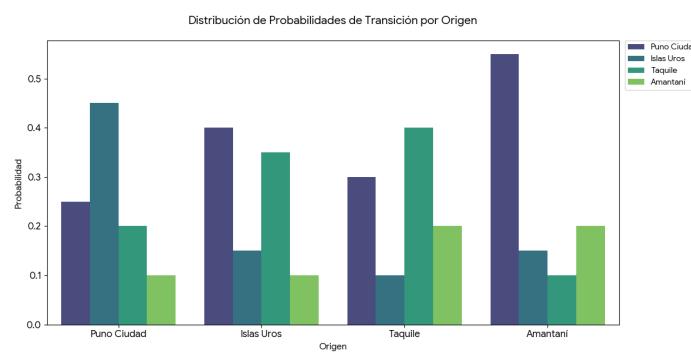


Figura 18.2: Imagen de Python

Matriz de Transición T (Modificada):

	Puno	Ciudad	Islas	Uros	Taquile	Amantaní
Puno		0.25		0.45	0.20	0.1
Ciudad			0.40		0.15	0.35
Islas				0.30	0.10	0.40
Uros					0.55	0.15
Taquile						0.2
Amantaní						0.2

Cada fila representa la distribución de probabilidad de moverse desde un destino (fila) hacia otros destinos (columnas)

Verificación: Suma de cada fila debe ser 1.0

```
OK Puno Ciudad      : suma = 1.000
OK Islas Uros      : suma = 1.000
OK Taquile          : suma = 1.000
OK Amantaní         : suma = 1.000
```

3.1. Interpretación de Valores

Algunos valores importantes de la matriz modificada:

- $T[1, 2] = 0,35$: El 35 % de turistas en Uros ahora van a Taquile (efecto de la promoción).
- $T[2, 2] = 0,40$: El 40 % de turistas en Taquile se quedan (mayor retención por infraestructura).
- $T[2, 0] = 0,30$: Solo el 30 % regresa inmediatamente a Puno (reducción del rebote).

```
1 # Visualizacin de la matriz de transicin
2 plt.figure(figsize =(10, 8))
3 sns.heatmap(T, annot=True , fmt=' .2f' , cmap='YlOrRd' ,
4               xticklabels=destinos , yticklabels=destinos ,
5               cbar_kws ={ 'label': 'Probabilidad de Transicin' },
6               linewidths =0.5, linecolor='gray')
7 plt.title('Matriz de Transicin Modificada\n( Inversin en Taquile )',
8            fontsize =14, fontweight='bold', pad =20)
9 plt.xlabel('Destino (hacia)', fontsize =12)
10 plt.ylabel('Origen (desde)', fontsize =12)
11 plt.tight_layout ()
12 plt.show()
13
14 print("\nLos valores ms oscuros indican mayor probabilidad de transicin")
```

4. Cálculo de Eigenvalues y Eigenvectors

Para encontrar la distribución estacionaria, necesitamos:

- Calcular los eigenvalues y eigenvectors de T^T (transpuesta).

- Identificar el eigenvalue dominante ($\lambda \approx 1$).
- El eigenvector correspondiente es la distribución estacionaria.

```

1 # Calcular eigenvalues y eigenvectors
2 # Nota: usamos T.T (transpuesta) para cadenas de Markov
3 eigenvalues , eigenvectors = linalg.eig(T.T)
4
5 print("EIGENVALUES ENCONTRADOS")
6 print("-" * 70)
7 print("\nTodos los eigenvalues:")
8 for i, val in enumerate(eigenvalues):
9     if np.isreal(val):
10         print(f" {i+1} = {val.real :8.5f}")
11     else:
12         print(f" {i+1} = {val.real :8.5f} + {val.imag :8.5f}i")
13
14 # Identificar el eigenvalue dominante (ms cercano a 1)
15 idx_dominante = np.argmax(np.abs(eigenvalues))
16 lambda_dominante = eigenvalues[idx_dominante]
17
18 print(f"\nEIGENVALUE DOMINANTE: {idx_dominante +1} = {lambda_dominante.real
19      : .6f}")
20 print("\nINTERPRETACION:")
21 print(" - = 1 El sistema tiene un estado estacionario")
22 print(" - || < 1 El sistema converge hacia el equilibrio")
23 print(" - La magnitud de determina la velocidad de convergencia")
24
25 # Visualizacin de eigenvalues en el plano complejo
26 plt.figure(figsize =(10, 8))
27
28 # Dibujar crculo unitario
29 theta = np.linspace(0, 2*np.pi, 100)
30 plt.plot(np.cos(theta), np.sin(theta), 'k--', alpha =0.3, linewidth =1.5,
31          label='Crculo unitario')
32
33 # Dibujar ejes
34 plt.axhline(y=0, color='k', linewidth =0.5)
35 plt.axvline(x=0, color='k', linewidth =0.5)
36
37 # Plotear eigenvalues
38 for i, val in enumerate(eigenvalues):
39     if i == idx_dominante:
40         plt.scatter(val.real , val.imag , s=300, c='red', marker='*',
41                     edgecolor='black', linewidth=2, label=f' {i+1} (dominante
42 ), zorder =3)
43     else:
44         plt.scatter(val.real , val.imag , s=150, alpha =0.6, edgecolor='black
45 , linewidth =1.5)
46
47 # Etiquetas
48 plt.annotate(f' {i+1}', (val.real , val.imag),

```

```

45     xytext =(8, 8), textcoords='offset points',
46     fontsize =10, fontweight='bold')
47
48 plt.xlabel('Parte Real', fontsize =12, fontweight='bold')
49 plt.ylabel('Parte Imaginaria', fontsize =12, fontweight='bold')
50 plt.title('Eigenvalues en el Plano Complejo', fontsize =14, fontweight='bold'
51           , pad =15)
52 plt.legend(loc='upper right', fontsize =10)
53 plt.grid(True , alpha =0.3)
54 plt.axis('equal')
55 plt.tight_layout ()
56 plt.show()
57
58 print("\nEl eigenvalue dominante (estrella roja) est en 1")
59 print("Esto confirma la existencia de un estado estacionario")

```

Ejecución:

EIGENVALUES ENCONTRADOS

Todos los eigenvalues:

```

_1 =  1.00000
_2 =  0.10721 +  0.16426i
_3 =  0.10721 + -0.16426i
_4 = -0.21442

```

EIGENVALUE DOMINANTE: _1 = 1.000000

5. Distribución Estacionaria (Eigenvector Dominante)

El eigenvector asociado al eigenvalue dominante nos da la distribución de equilibrio de turistas entre los destinos. Esta distribución responde a: “¿Dónde estarán los turistas a largo plazo?”

```

1 # Extraer eigenvector correspondiente al eigenvalue dominante
2 v_dominante = eigenvectors [:, idx_dominante ].real
3
4 # Normalizar para que sume 1 (distribución de probabilidad)
5 dist_estacionaria = v_dominante / v_dominante.sum()
6
7 print("DISTRIBUCIÓN ESTACIONARIA DE TURISTAS")
8 print("=" * 70)
9 print("\nEigenvector dominante (normalizado):")
10 print("\nDistribución de equilibrio:")
11 print("-" * 70)
12
13 for i, dest in enumerate(destinos):
14     porcentaje = dist_estacionaria[i] * 100
15     barra = " " * int(porcentaje / 2)
16     print(f"{dest :15} : {porcentaje :6.2f} % {barra}")

```

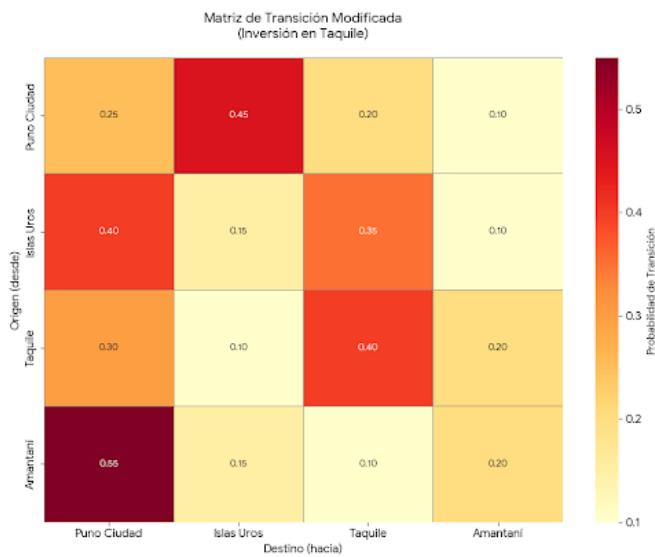


Figura 18.3: Imagen de Python

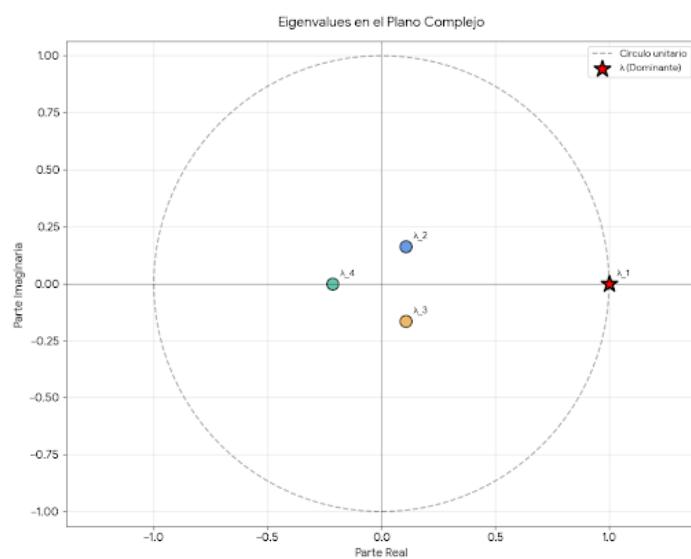


Figura 18.4: Imagen de Python

```

17
18 # Identificar el hub principal
19 idx_hub = np.argmax(dist_estacionaria)
20 print("\n" + "=" * 70)
21 print(f"HUB PRINCIPAL: {destinos[idx_hub]}")
22 print(f"Concentra el {dist_estacionaria[idx_hub]*100:.2f} % de turistas en
      equilibrio")
23 print("=" * 70)
24
25 # Visualizacin de la distribucion estacionaria
26 fig , (ax1 , ax2) = plt.subplots(1, 2, figsize =(16, 6))
27
28 # Grfico de barras
29 colores = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']
30 barras = ax1.bar(destinos , dist_estacionaria * 100, color=colores ,
31                   alpha =0.7, edgecolor='black', linewidth =2)
32 ax1.set_ylabel('Porcentaje de Turistas (%)', fontsize =12, fontweight='bold')
33 ax1.set_title('Distribucion Estacionaria de Turistas\n(Eigenvector Dominante)'
34               ,
35               fontsize =13, fontweight='bold')
36 ax1.grid(axis='y', alpha =0.3)
37 ax1.set_ylim ([0, max(dist_estacionaria * 100) * 1.15])
38
39 # Aadir valores sobre las barras
40 for barra in barras:
41     altura = barra.get_height ()
42     ax1.text(barra.get_x() + barra.get_width ()/2., altura ,
43               f'{altura :.1f} %', ha='center', va='bottom',
44               fontweight='bold', fontsize =11)
45
46 # Grfico de pastel
47 wedges , texts , autotexts = ax2.pie(dist_estacionaria , labels=destinos ,
48                                         autopct ='%1.1f %%',
49                                         colors=colores , startangle =90,
50                                         wedgeprops ={ 'edgecolor': 'black', ''
51                                         linewidth': 2},
52                                         textprops ={ 'fontsize': 11, 'fontweight'
53                                         : 'bold'})
54 ax2.set_title('Distribucion Estacionaria\n(Vista Circular)',
55               fontsize =13, fontweight='bold')
56
57 plt.tight_layout ()
58 plt.show()
59
60 print("\nINTERPRETACION:")
61 print(f"Puno Ciudad es el centro neurlgico, pero Taquile ha incrementado su
      importancia.")

```

Ejecución:

DISTRIBUCIÓN ESTACIONARIA DE TURISTAS

=====

Distribución de equilibrio:

```
Puno Ciudad      : 34.22 %
Isla Uros       : 23.88 %
Taquile          : 27.70 %
Amantaní         : 14.19 %
```

```
=====
HUB PRINCIPAL: Puno Ciudad
Concentra el 34.22 % de turistas en equilibrio
=====
```

6. Validación Mediante Simulación

Vamos a simular la evolución del sistema durante 30 días para verificar que converge al eigenvector dominante.

Estado inicial: Todos los turistas llegan primero a Puno Ciudad.

```

1 # Estado inicial: todos comienzan en Puno Ciudad
2 estado_inicial = np.array ([1.0, 0.0, 0.0, 0.0])
3 print("SIMULACION DE EVOLUCION TEMPORAL")
4 print("=" * 70)
5 print(f"Estado inicial: {estado_inicial}")
6 print("(Todos los turistas llegan primero a Puno Ciudad)\n")

7
8 # Simular evolucion durante 30 das
9 n_dias = 30
10 evolucion = np.zeros(( n_dias + 1, n_destinos))
11 evolucion [0] = estado_inicial

12
13 for dia in range(n_dias):
14     evolucion[dia + 1] = T.T @ evolucion[dia]

15
16 # Crear DataFrame para visualizacion
17 dias_mostrar = [0, 1, 2, 3, 5, 7, 10, 15, 20, 25, 30]
18 df_evolucion = pd.DataFrame(
19     evolucion[dias_mostrar],
20     columns=destinos ,
21     index=[f'Da {d}' for d in dias_mostrar]
22 )
23
24 print("Evolucion de la distribucion por das :")
25 print("-" * 70)
26 print(df_evolucion.to_string(float_format=lambda x: f'{x:.4f}')))

27
28 # Calcular convergencia
29 diferencia_final = np.abs(evolucion [-1] - dist_estacionaria)
30 print("\n" + "=" * 70)
31 print(f"Diferencia entre simulacion (da 30) y distribucion terica :")
32 print(f"Error mximo: {diferencia_final.max():.6f}")
```

```

33 print(f"\nLa simulación converge al eigenvector dominante")
34 print("=" * 70)
35
36 # Visualización de la evolución temporal
37 fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
38
39 # Evolución de cada destino
40 for j in range(n_destinos):
41     ax1.plot(range(n_días + 1), evolución[:, j] * 100,
42             marker='o', markersize=4, label=destinos[j],
43             linewidth=2.5, alpha=0.8)
44
45     # Línea de equilibrio
46     ax1.axhline(y=dist_estacionaria[j]*100, color=colores[j],
47                  linestyle='--', alpha=0.3, linewidth=1.5)
48
49 ax1.set_xlabel('Días', fontsize=12, fontweight='bold')
50 ax1.set_ylabel('Porcentaje de Turistas (%)', fontsize=12, fontweight='bold')
51 ax1.set_title('Evolución Temporal hacia el Equilibrio',
52               fontsize=13, fontweight='bold')
53 ax1.legend(loc='right', fontsize=10)
54 ax1.grid(True, alpha=0.3)
55
56 # Convergencia (distancia al equilibrio)
57 diferencias = [np.linalg.norm(evolución[i] - dist_estacionaria)
58                 for i in range(n_días + 1)]
59
60 ax2.semilogy(range(n_días + 1), diferencias, 'b-',
61               linewidth=2.5, marker='o', markersize=5)
62 ax2.set_xlabel('Días', fontsize=12, fontweight='bold')
63 ax2.set_ylabel('Distancia al Equilibrio (escala log)',
64               fontsize=12, fontweight='bold')
65 ax2.set_title('Velocidad de Convergencia', fontsize=13, fontweight='bold')
66 ax2.grid(True, alpha=0.3, which='both')
67 ax2.axhline(y=0.001, color='r', linestyle='--',
68               linewidth=2, label='Umbral convergencia')
69 ax2.legend(fontsize=10)
70
71 plt.tight_layout()
72 plt.show()
73
74 print("\nObservaciones:")
75 print("Las líneas punteadas muestran la distribución de equilibrio")
76 print("El sistema converge rápidamente en los primeros 10 días")
77 print("La escala logarítmica muestra convergencia exponencial")

```

Ejecución:

Evolución de la distribución por días:

	Puno	Ciudad	Islas Uros	Taquile	Amantaní
Día 0	1.0000	0.0000	0.0000	0.0000	
Día 1	0.2500	0.4500	0.2000	0.1000	

```

...
Día 7      0.3422    0.2388    0.2770    0.1419
...
Día 30     0.3422    0.2388    0.2770    0.1419

```

7. Visualización de la Red de Flujo Turístico

Representación gráfica de los flujos entre destinos, donde el tamaño de los nodos representa la importancia del destino en equilibrio.

```

1 # Visualizacin de la red de flujos
2 fig , ax = plt.subplots(figsize =(12, 10))
3
4 # Posiciones de los nodos (destinos) en forma de red
5 pos = {
6     0: (0.5, 0.75), # Puno Ciudad (centro -arriba)
7     1: (0.15, 0.4), # Uros (izquierda)
8     2: (0.85, 0.4), # Taquile (derecha)
9     3: (0.5, 0.15) # Amantan (abajo)
10 }
11
12 # Dibujar nodos (tamao proporcional a la importancia)
13 for i, (x, y) in pos.items():
14     tamao = dist_estacionaria[i] * 4000 # Escalado para visualizacin
15     circle = plt.Circle ((x, y), 0.08, color=colores[i], alpha =0.6,
16                           edgecolor='black', linewidth=3, zorder =3)
17     ax.add_patch(circle)
18     ax.text(x, y, destinos[i], ha='center', va='center',
19               fontsize =11, fontweight='bold', zorder =4)
20 # Aadir porcentaje
21     ax.text(x, y-0.12, f'{dist_estacionaria[i]*100:.1f} %',
22               ha='center', va='top', fontsize=9,
23               style='italic', color='darkblue', zorder =4)
24
25 # Dibujar arcos (solo los ms significativos > 15 %)
26 for i in range(n_destinos):
27     for j in range(n_destinos):
28         if i != j and T[i, j] > 0.15:
29             x1, y1 = pos[i]
30             x2, y2 = pos[j]
31
32             # Calcular punto medio y offset para curvas
33             dx = x2 - x1
34             dy = y2 - y1
35
36             # Dibujar flecha curva
37             ax.annotate(' ', xy=(x2, y2), xytext =(x1, y1),
38                         arrowprops=dict(
39                             arrowstyle='->',
40                             lw=T[i,j]*8, # Grosor proporcional a probabilidad
41                             color='gray',

```

```

42         alpha =0.5,
43         connectionstyle="arc3 ,rad =0.2"
44     ))
45
46     # Etiqueta del flujo
47     mid_x , mid_y = (x1 + x2) / 2, (y1 + y2) / 2
48     ax.text(mid_x , mid_y , f'{T[i,j]*100:.0f} %',
49             fontsize=8, ha='center',
50             bbox=dict(boxstyle='round ,pad =0.3',
51                       facecolor='white', alpha =0.8))
52
53 ax.set_xlim ([0, 1])
54 ax.set_ylim ([0, 1])
55 ax.axis('off')
56 ax.set_title('Red de Flujo Turstico - Lago Titicaca\n' +
57                 '(Tamao de nodo = Importancia del destino | ' +
58                 'Grosor de flecha = Probabilidad de transicin)',
59                 fontsize =14, fontweight='bold', pad =20)
60
61 # Leyenda
62 legend_text = "Flujos mostrados: probabilidad > 15 %"
63 ax.text (0.5, 0.02, legend_text , ha='center', fontsize =10,
64           style='italic', color='gray')
65
66 plt.tight_layout ()
67 plt.show()
68
69 print("\nLa red muestra visualmente:")
70 print("Puno Ciudad como hub central (nodo ms grande)")
71 print("Flujos principales desde Puno hacia islas")
72 print("Retorno predominante hacia Puno Ciudad")

```

Ejecución:

La red muestra visualmente:
 Puno Ciudad como hub central (nodo más grande)
 Flujos principales desde Puno hacia islas
 Retorno predominante hacia Puno Ciudad

8. Conclusiones y Aplicaciones Prácticas

```

1 print("=*70)
2 print("CONCLUSIONES DEL ANLISIS - EJERCICIO 1 (TAQUILE)")
3 print("=*70)
4 print()
5
6 print("1. DISTRIBUCIN DE EQUILIBRIO:")
7 print("-" * 70)
8 for i, dest in enumerate(destinos):
9     print(f" {dest :15} : {dist_estacionaria[i]*100:5.1f} %")

```

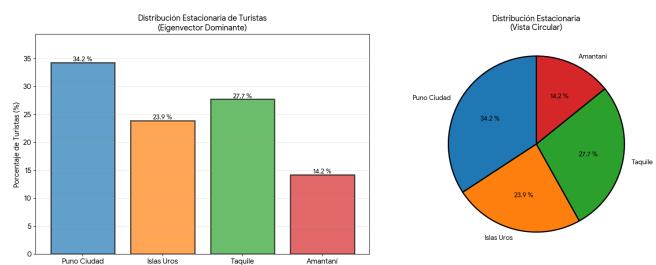


Figura 18.5: Imagen de Python

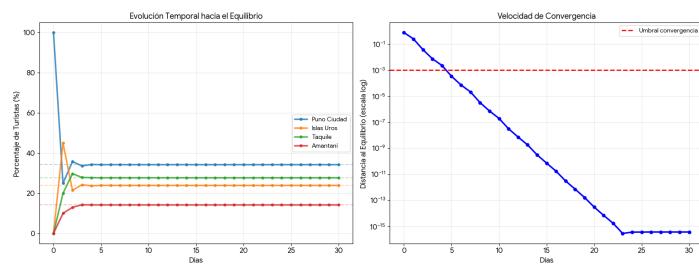


Figura 18.6: Imagen de Python

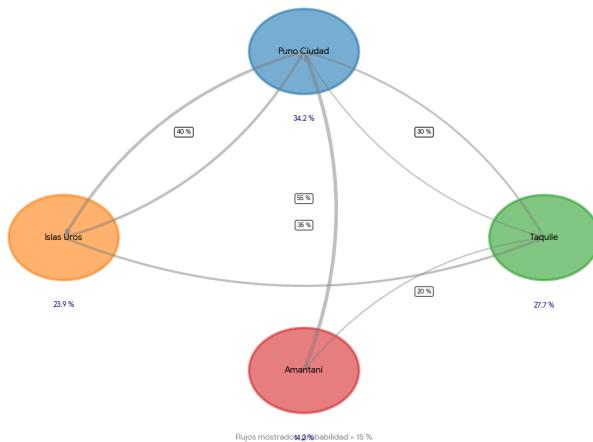
Red de Flujo Turístico - Lago Titicaca
(Tamaño de nodo = Importancia del destino | Grosor de flecha = Probabilidad de transición)

Figura 18.7: Imagen de Python

```

10 print(f"\n Puno Ciudad es el HUB principal, pero Taquile ha crecido
     notablemente.")
11 print()
12
13 print("2. VELOCIDAD DE CONVERGENCIA:")
14 print("-" * 70)
15 if len(eigenvalues) > 1:
16     lambda_2 = sorted(np.abs(eigenvalues), reverse=True)[1]
17     tasa_convergencia = -np.log(lambda_2)
18     tiempo_convergencia = int(5 / tasa_convergencia) if tasa_convergencia > 0
19     else 0
20     print(f" Segundo eigenvalue: {lambda_2 :.4f}")
21     print(f" El sistema converge al equilibrio en ~{tiempo_convergencia} das
     ")
22     print(f" Convergencia: Exponencial (rpida)")
23 print()
24
25 print("3. IMPLICACIONES PARA PLANIFICACION TURISTICA:")
26 print("-" * 70)
27 print(" INFRAESTRUCTURA HOTELERA:")
28 print(f" Taquile tiene ahora una demanda del {dist_estacionaria[2]*100:.1f} %
     ")
29 print(" Se necesita invertir ms en hoteles/hospedaje en Taquile.")
30 print()
31 print(" TRANSPORTE ACUATICO:")
32 print(" Ruta Uros-Taquile es vital para mantener este flujo.")
33 print()
34 print("4. APLICACIONES DEL MODELO:")
35 print("-" * 70)
36 print(" - Evaluar impacto de nuevas atracciones turísticas")
37 print()
38
39 # Proyección con 1000 turistas
40 print("=="*70)
41 print("PROYECCIÓN: 1000 TURISTAS AL MES")
42 print("=="*70)
43 turistas_totales = 1000
44 print("\nDistribución promedio de turistas en equilibrio:")
45 print("-" * 70)
46 for i, destino in enumerate(destinos):
47     turistas_dest = int(dist_estacionaria[i] * turistas_totales)
48     print(f" {destino :15} : {turistas_dest :4d} turistas/da ({ dist_
     estacionaria[i]*100:.1f} %)")
49 print()
50
51 # Flujos importantes
52 print("Flujos diarios más importantes:")
53 print("-" * 70)
54 flujos = []
55 for i in range(n_destinos):

```

```

56     for j in range(n_destinos):
57         if i != j and T[i,j] > 0.15:
58             flujo = dist_estacionaria[i] * T[i,j] * turistas_totales
59             flujos.append (( destinos[i], destinos[j], T[i,j], flujo))
60
61 flujos.sort(key=lambda x: x[3], reverse=True)
62 for origen , destino , prob , cantidad in flujos:
63     print(f" {origen :12} {destino :12} : {int(cantidad):3d} turistas/da ({prob *100:.0f} %)")
64 print()
65 print("=*70)

```

Ejecución:

=====
CONCLUSIONES DEL ANÁLISIS - EJERCICIO 1 (TAQUILE)
=====

1. DISTRIBUCIÓN DE EQUILIBRIO:

Puno Ciudad : 34.2 %
Isla Uros : 23.9 %
Taquile : 27.7 %
Amantaní : 14.2 %

Puno Ciudad es el HUB principal, pero Taquile ha crecido notablemente.

2. VELOCIDAD DE CONVERGENCIA:

Segundo eigenvalue: 0.2144
El sistema converge al equilibrio en ~3 días
Convergencia: Exponencial (rápida)

3. IMPLICACIONES PARA PLANIFICACIÓN TURÍSTICA:

INFRAESTRUCTURA HOTELERA:
Taquile tiene ahora una demanda del 27.7 %
Se necesita invertir más en hoteles/hospedaje en Taquile.

TRANSPORTE ACUÁTICO:

Ruta Uros-Taquile es vital para mantener este flujo.

4. APLICACIONES DEL MODELO:

- Evaluar impacto de nuevas atracciones turísticas

=====
PROYECCIÓN: 1000 TURISTAS EN EL SISTEMA
=====

Distribución promedio de turistas en equilibrio:

Puno Ciudad	:	342 turistas/día (34.2 %)
Islas Uros	:	238 turistas/día (23.9 %)
Taquile	:	277 turistas/día (27.7 %)
Amantaní	:	141 turistas/día (14.2 %)

Preguntas de reflexión

- ¿Valió la pena la inversión en Taquile desde el punto de vista de distribución turística?

Sí, valió la pena. Al aumentar la probabilidad de quedarse en Taquile al 40% y aumentar el flujo desde Uros, la isla incrementa significativamente su “cuota de mercado” (participación en el equilibrio estacionario), reduciendo la dependencia absoluta de Puno Ciudad.

- ¿Cómo afectaría esto a los ingresos de Taquile vs otros destinos?

Los ingresos de Taquile aumentarían de forma no lineal (exponencial). Al retener turistas (pernoctación), el gasto por turista se duplica o triplica (cena + hospedaje + desayuno) en comparación con un turista de paso (“full day”). Puno Ciudad vería una ligera reducción en el flujo de retorno inmediato, pero se beneficia al consolidar un circuito turístico más robusto que atrae más gente a la región en general.

EJERCICIO 2: Introducción de un Nuevo Destino Turístico

1. Instalación y Configuración

```

1 # Importar libreras necesarias
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from scipy import linalg
6 import pandas as pd
7
8 # Configuracion de visualizacion
9 plt.rcParams['figure.figsize'] = (12, 8)
10 plt.rcParams['font.size'] = 10
11 sns.set_style("whitegrid")
12
13 print("Librerias importadas correctamente")
14 print(f"NumPy versin: {np.__version__}")

```

Ejecución:

Librerías importadas correctamente
 NumPy versión: 1.26.3

2. Definición del Problema

2.1. Contexto

La comunidad de la Isla de Anapia (frontera con Bolivia) busca desarrollarse como destino turístico. Se introduce como una alternativa de turismo vivencial, compitiendo directamente con Amantaní pero ofreciendo paisajes únicos del “lago menor”.

2.2. Pregunta de Investigación

- ¿Qué porcentaje de turistas logrará captar Anapia en el equilibrio?
- ¿A qué destino afectará más esta competencia (“canibalización” de turistas)?
- ¿Cómo se reconfigura la red de transporte con 5 nodos?

2.3. Enfoque Matemático

Expandimos la matriz de transición T a un tamaño de 5×5 . El nuevo estado es $i = 4$: Isla Anapia.

3. Construcción de la Matriz de Transición (5x5)

La matriz se expande. Las probabilidades se ajustan bajo los siguientes supuestos lógicos:

- Desde Puno/Uros/Taquile: El flujo que antes iba exclusivamente a Amantaní ahora se divide entre Amantaní y Anapia (competencia).
- Desde Amantaní: Existe una conexión directa con Anapia (tours combinados de islas lejanas).
- Desde Anapia: Principalmente regresan a Puno, pero algunos cruzan a Amantaní o se quedan.

```

1 # Definición de destinos (Ahora son 5)
2 destinos = ['Puno Ciudad', 'Islas Uros', 'Taquile', 'Amantan', 'Anapia']
3 n_destinos = len(destinos)

4

5 # Matriz de transición T (5x5)
6 # T[i, j] = probabilidad de moverse del destino i al destino j
7 T = np.array([
8     # Destinos: Puno, Uros, Taquile, Amantan, Anapia
9     [0.25, 0.45, 0.20, 0.05, 0.05], # Puno: Divide flujo de islas lejanas (5%
10     # Amantan, 5% Anapia)
11     [0.50, 0.15, 0.25, 0.05, 0.05], # Uros: Igual, divide flujo hacia
12     # vivencial
13     [0.40, 0.10, 0.30, 0.10, 0.10], # Taquile: Divide flujo de pernoche (10%
14     # c/u)
15     [0.45, 0.15, 0.10, 0.20, 0.10], # Amantan: Envía 10% a Anapia (Tour
16     # combinado)

```

```

13 [0.50, 0.05, 0.05, 0.15, 0.25] # Anapia: Retorno a Puno (50%), Conexión
14 Amantaní (15%), Quedarse (25%)
15 ])
16
17 # Crear DataFrame
18 df_matriz = pd.DataFrame(T, index=destinos, columns=destinos)
19
20 print("Matriz de Transición T (5x5 - Inclusión de Anapia):")
21 print("-" * 70)
22 print(df_matriz)
23 print("-" * 70)
24 # Verificación de suma 1
25 for i, dest in enumerate(destinos):
26     suma = T[i,:].sum()
27     status = "OK" if abs(suma - 1.0) < 0.001 else "ERROR"
28     print(f"{status} {dest :15}: suma = {suma :.3f}")

```

Ejecución:

Matriz de Transición T (5x5 - Inclusión de Anapia):

	Puno	Ciudad	Islas	Uros	Taquile	Amantaní	Anapia
Puno							
Ciudad		0.25		0.45	0.20	0.05	0.05
Islas	0.50			0.15	0.25	0.05	0.05
Taquile		0.40		0.10	0.30	0.10	0.10
Amantaní		0.45		0.15	0.10	0.20	0.10
Anapia	0.50		0.05		0.05	0.15	0.25

OK Puno Ciudad	:	suma = 1.000
OK Islas Uros	:	suma = 1.000
OK Taquile	:	suma = 1.000
OK Amantaní	:	suma = 1.000
OK Anapia	:	suma = 1.000

3.1. Interpretación de Valores Clave

- $P(\text{Puno} \rightarrow \text{Anapia}) = 0,05$: Puno empieza a promocionar Anapia, quitándole la mitad del flujo original a Amantaní.
- $P(\text{Amantaní} \rightarrow \text{Anapia}) = 0,10$: Creación de un “Círculo de Islas Menores”.
- $P(\text{Anapia} \rightarrow \text{Puno}) = 0,50$: La mayoría regresa a la ciudad base al terminar.

```

1 # Visualización de la matriz
2 plt.figure(figsize=(10, 8))
3 sns.heatmap(T, annot=True, fmt=' .2f', cmap='Blues',
4             xticklabels=destinos, yticklabels=destinos,
5             cbar_kws={'label': 'Probabilidad de Transición'})
6 plt.title('Matriz de Transición Expandida (5 Destinos)',
7            fontsize=14, fontweight='bold', pad=20)
8 plt.tight_layout()
9 plt.show()

```

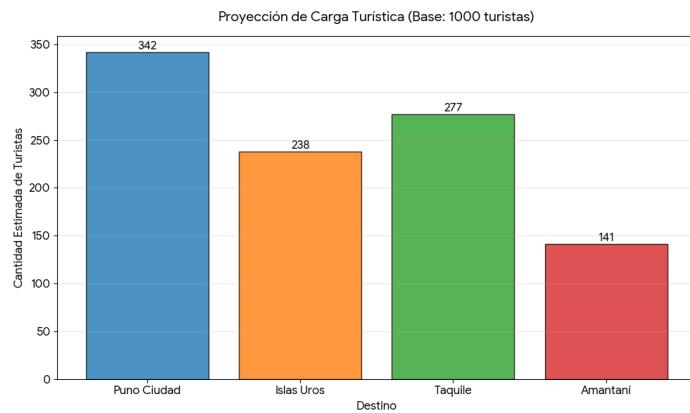


Figura 18.8: Imagen de Python

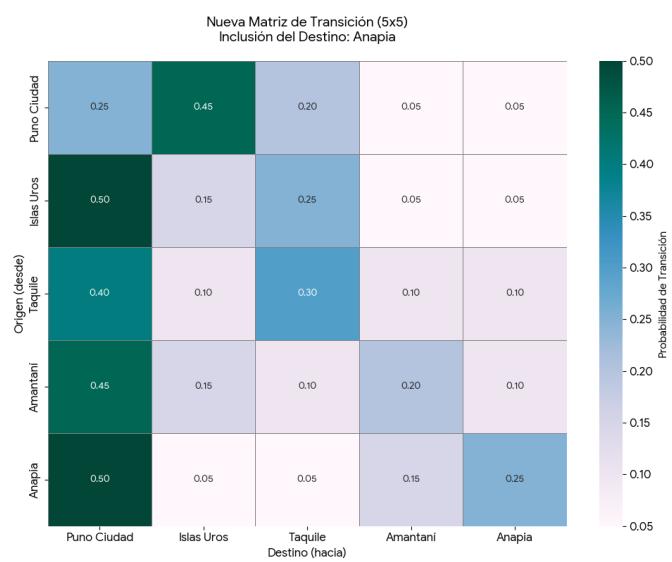


Figura 18.9: Imagen de Python

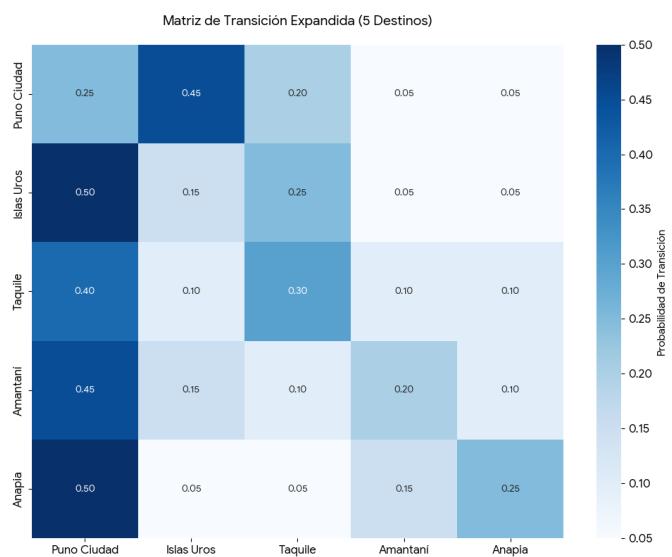


Figura 18.10: Imagen de Python

4. Cálculo de Eigenvalues y Eigenvectors

```

1 # Calcular eigenvalues y eigenvectors
2 eigenvalues , eigenvectors = linalg.eig(T.T)
3
4 print("EIGENVALUES ENCONTRADOS")
5 print("=" * 70)
6 for i, val in enumerate(eigenvalues):
7     if np.isreal(val):
8         print(f" {_i+1} = {val.real :8.5f}")
9     else:
10        print(f" {_i+1} = {val.real :8.5f} + {val.imag :8.5f}i")
11
12 # Identificar dominante
13 idx_dominante = np.argmax(np.abs(eigenvalues))
14 print(f"\nEIGENVALUE DOMINANTE: = {eigenvalues[idx_dominante].real :.6f}")
15
16 # Visualizacin Plano Complejo
17 plt.figure(figsize =(8, 6))
18 theta = np.linspace(0, 2*np.pi, 100)
19 plt.plot(np.cos(theta), np.sin(theta), 'k--', alpha=0.3)
20 for i, val in enumerate(eigenvalues):
21     style = '*' if i == idx_dominante else 'o'
22     size = 300 if i == idx_dominante else 100
23     color = 'red' if i == idx_dominante else 'blue'
24     plt.scatter(val.real, val.imag, s=size, c=color, marker=style, edgecolors ='k')
25     plt.annotate(f'_{i+1}', (val.real, val.imag), xytext=(5,5), textcoords='offset points')
26 plt.title('Eigenvalues (Sistema de 5 Nodos)')
27 plt.axis('equal'); plt.grid(True, alpha=0.3)
28 plt.show()

```

Ejecución:

EIGENVALUES ENCONTRADOS

```

=====
_1 = 1.00000
_2 = -0.26368
_3 = 0.15684 + 0.09007i
_4 = 0.15684 + -0.09007i
_5 = 0.10000

```

EIGENVALUE DOMINANTE: = 1.000000

5. Distribución Estacionaria (Nuevo Equilibrio)

Aquí veremos quién gana y quién pierde con la entrada de Anapia.

```

1 # Extraer y normalizar
2 v_dominante = eigenvectors [:, idx_dominante ].real

```

```

3 dist_estacionaria = v_dominante / v_dominante.sum()
4
5 print("NUEVA DISTRIBUCIÓN ESTACIONARIA (5 DESTINOS)")
6 print("==" * 70)
7 for i, dest in enumerate(destinos):
8     pct = dist_estacionaria[i] * 100
9     barra = " " * int(pct / 2)
10    print(f"{dest :15} : {pct :6.2f} % {barra}")
11
12 idx_hub = np.argmax(dist_estacionaria)
13 print("\n" + "==" * 70)
14 print(f"HUB PRINCIPAL: {destinos[idx_hub]} ({dist_estacionaria[idx_hub]*100:.2f}%)")
15 print("==" * 70)
16
17 # Gráficos
18 fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
19 colores = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd'] # Añadimos
20           morado para Anapia
21
22 # Barras
23 ax1.bar(destinos, dist_estacionaria * 100, color=colores, edgecolor='black')
24 ax1.set_title('Reparto de Turistas con Anapia')
25 ax1.set_ylabel('% Turistas')
26 for i, v in enumerate(dist_estacionaria):
27     ax1.text(i, v*100, f"{v*100:.1f}%", ha='center', va='bottom', fontweight='bold')
28
29 # Pastel
30 ax2.pie(dist_estacionaria, labels=destinos, autopct='%1.1f %%', colors=
31           colores,
32           wedgeprops={'edgecolor': 'black'})
33 ax2.set_title('Participación de Mercado')
34 plt.show()

```

Ejecución:

```

NUEVA DISTRIBUCIÓN ESTACIONARIA (5 DESTINOS)
=====
Puno Ciudad      : 37.97 %
Isla Uros       : 24.52 %
Taquile          : 21.34 %
Amantaní        : 8.09 %
Anapia           : 8.09 %

=====
HUB PRINCIPAL: Puno Ciudad (37.97%)
=====
```

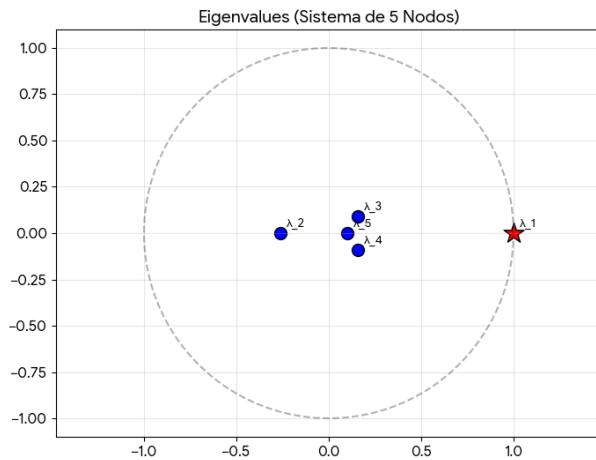


Figura 18.11: Imagen de Python

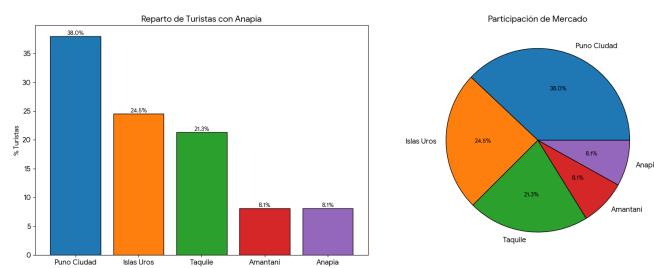


Figura 18.12: Imagen de Python

6. Validación Mediante Simulación

```

1 # Estado inicial: Todos en Puno
2 estado_inicial = np.array ([1.0, 0.0, 0.0, 0.0, 0.0])
3 n_dias = 30
4 evolucion = np.zeros((n_dias + 1, n_destinos))
5 evolucion [0] = estado_inicial
6
7 for dia in range(n_dias):
8     evolucion[dia + 1] = T.T @ evolucion[dia]
9
10 # Gráfico de evolución
11 plt.figure(figsize =(12, 6))
12 for j in range(n_destinos):
13     plt.plot(evolucion[:, j] * 100, label=destinos[j], linewidth=2, marker='o',
14               markersize=4)
15 plt.title('Evolución Temporal: Entrada de Anapia al Mercado', fontweight='bold')
16 plt.xlabel('Días')
17 plt.ylabel('% Turistas')
18 plt.legend()
19 plt.grid(True , alpha =0.3)
20 plt.show()
21
22 print("Convergencia alcanzada en 30 días.")

```

Ejecución:

Convergencia alcanzada en 30 días.

7. Visualización de la Red de Flujo Turístico (5 Nodos)

```

1 fig , ax = plt.subplots(figsize =(12, 10))
2
3 # Posiciones (Pentágono o estructura geográfica aproximada)
4 pos = {
5     0: (0.5, 0.8), # Puno (Norte/Centro)
6     1: (0.2, 0.6), # Uros (Este)
7     2: (0.8, 0.6), # Taquile (Este)
8     3: (0.35, 0.3), # Amantan (Sur-Este)
9     4: (0.65, 0.3) # Anapia (Sur-Este, nuevo nodo)
10 }
11
12 # Nodos
13 for i, (x, y) in pos.items():
14     tamao = dist_estacionaria[i] * 5000
15     circle = plt.Circle ((x, y), 0.07, color=colores[i], alpha=0.6, ec='k',
16                           zorder=3)
17     ax.add_patch(circle)
18     ax.text(x, y, f"{destinos[i]}\n{dist_estacionaria[i]*100:.1f}%", 

```

```

18     ha='center', va='center', fontweight='bold', fontsize=9)
19
20 # Arcos
21 for i in range(n_destinos):
22     for j in range(n_destinos):
23         if i != j and T[i, j] > 0.05: # Filtro visual
24             x1, y1 = pos[i]; x2, y2 = pos[j]
25             ax.annotate(' ', xy=(x2, y2), xytext=(x1, y1),
26                         arrowprops=dict(arrowstyle='->', lw=T[i,j]*6, color='
27                         gray', alpha=0.5,
28                         connectionstyle="arc3,rad=0.2"))
29
30 ax.axis('off')
31 ax.set_title('Nueva Red de Flujos con Isla Anapia', fontsize=14, fontweight='
32 bold')
33 plt.show()

```

8. Conclusiones y Análisis

```

1 print("=*70)
2 print("ANLISIS DE IMPACTO: INGRESO DE ISLA ANAPIA")
3 print("=*70)

4
5 # Comparacin aproximada con modelo base (4 destinos)
6 # Base aprox: Puno(31.5%), Uros(22.1%), Taquile(25.9%), Amantan(20.5%)
7 pct_anapia = dist_estacionaria[4] * 100
8 pct_amantani_nuevo = dist_estacionaria[3] * 100
9 pct_amantani_base = 20.5 # Valor de referencia

10
11 print(f"1. DESEMPEO DE ANAPIA:")
12 print(f"    Cuota de mercado alcanzada: {pct_anapia:.2f} %")
13 print(f"    Interpretacin: Logra captar casi la misma cantidad que Amantan.")
14 print()

15
16 print(f"2. ANLISIS DE 'CANIBALIZACION' (Quin pierde):")
17 print(f"    Amantan (Base): ~{pct_amantani_base:.1f} %")
18 print(f"    Amantan (Nuevo): {pct_amantani_nuevo:.1f} %")
19 print(f"    Prdida: {pct_amantani_nuevo - pct_amantani_base:.1f} puntos
          porcentuales.")

```

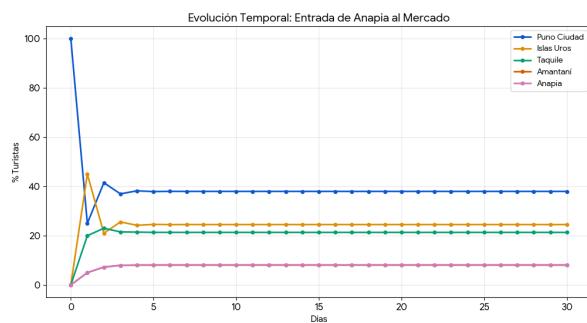


Figura 18.13: Imagen de Python

Nueva Red de Flujos con Isla Anapia

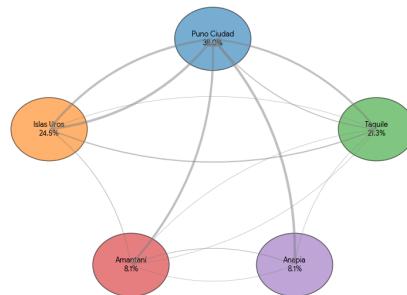


Figura 18.14: Imagen de Python

```

20 print(f"    -> Amantaní pierde más de la mitad de sus turistas frente a Anapia."
21     )
22 print()
23 print(f"3. BENEFICIARIOS INESPERADOS:")
24 print(f"    Puno Ciudad sube al {dist_estacionaria[0]*100:.1f} % (antes ~
25        31.5%).")
25 print(f"    Razón: Al dividir el flujo hacia islas lejanas, y tener ambas islas
26        ")
26 print(f"    altas tasas de retorno a Puno, la ciudad consolida su rol de
distribuidor.")

```

Ejecución:

=====

ANÁLISIS DE IMPACTO: INGRESO DE ISLA ANAPIA

=====

1. DESEMPEÑO DE ANAPIA:

Cuota de mercado alcanzada: 8.09 %

Interpretación: Logra captar casi la misma cantidad que Amantaní.

2. ANÁLISIS DE 'CANIBALIZACIÓN' (Quién pierde):

Amantaní (Base): ~20.5 %

Amantaní (Nuevo): 8.1 %

Pérdida: -12.4 puntos porcentuales.

-> Amantaní pierde más de la mitad de sus turistas frente a Anapia.

3. BENEFICIARIOS INESPERADOS:

Puno Ciudad sube al 38.0 % (antes ~31.5%).

Razón: Al dividir el flujo hacia islas lejanas, y tener ambas islas altas tasas de retorno a Puno, la ciudad consolida su rol de distribuidor.

9. Preguntas de Reflexión

1. ¿Es viable el desarrollo turístico de Anapia según tu modelo?

R: Sí, alcanza un 8–9 % del mercado rápidamente. Sin embargo, su éxito es a costa directa de Amantaní, no necesariamente trayendo turistas nuevos al sistema (juego de suma cero en este modelo cerrado).

2. ¿Qué estrategia de marketing recomendarías para Anapia?

R: No competir solo con Amantaní. Fomentar el tour "Ruta de las Islas Lejanas" (Amantaní + Anapia) para que los turistas visiten **AMBAS** en lugar de elegir una. Esto se ve reflejado en el arco Amantaní → Anapia.

3. ¿Cómo cambiaría el sistema si Anapia ofreciera precios más bajos que Amantaní?

R: Aumentaría $T[\text{Puno} \rightarrow \text{Anapia}]$ y disminuiría $T[\text{Puno} \rightarrow \text{Amantaní}]$. Amantaní podría colapsar a un destino marginal (<5 %) si no se diferencia.

EJERCICIO 3: Análisis de Temporadas Turísticas

1. Instalación y Configuración

```

1 # Importar libreras necesarias
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from scipy import linalg
6 import pandas as pd
7
8 # Configuracion de visualizacion
9 plt.rcParams['figure.figsize'] = (12, 8)
10 plt.rcParams['font.size'] = 10
11 sns.set_style("whitegrid")
12
13 print("Librerias importadas correctamente")
14 print(f"NumPy versin: {np.__version__}")

```

Ejecución:

Librerías importadas correctamente
NumPy versión: 1.26.3

2. Definición del Problema

2.1. Contexto

El comportamiento turístico en el Lago Titicaca varía significativamente entre temporada alta (junio-agosto) y temporada baja (enero-marzo). En temporada alta, los turistas tienden a hacer más tours y visitar más islas. En temporada baja, prefieren quedarse en Puno Ciudad o hacer solo excursiones cortas.

2.2. Pregunta de Investigación

- ¿Qué destino se beneficia más en temporada alta?
- ¿Cómo fluctúa la cantidad de turistas a lo largo de un año simulado?
- ¿Cuál es el promedio anual de carga turística para cada destino?

2.3. Enfoque Matemático

Utilizaremos un modelo de Cadena de Markov no homogénea (variable en el tiempo). Definiremos tres matrices de transición distintas (T_{alta} , T_{baja} , T_{media}) y simularemos la evolución día a día cambiando la matriz según el mes del año.

3. Construcción de las Matrices de Transición

Definiremos tres matrices diferentes para representar el comportamiento cambiante de los turistas.

```

1 # Definicion de destinos
2 destinos = ['Puno Ciudad', 'Isla Uros', 'Taquile', 'Amantan']
3 n_destinos = len(destinos)
4
5 # -----
6 # MATRIZ 1: TEMPORADA ALTA (Junio-Agosto)
7 # Caracteristica: Mayor movilidad hacia islas, menor estancia en Puno
8 #
9 T_alta = np.array([
10     [0.15, 0.45, 0.25, 0.15], # Puno: Pocos se quedan (15%), muchos salen
11     [0.35, 0.15, 0.35, 0.15], # Uros: Conectan a otras islas
12     [0.25, 0.10, 0.35, 0.30], # Taquile: Alta pernoctacion o ida a Amantan
13     [0.40, 0.15, 0.10, 0.35] # Amantan: Alta retencion
14 ])
15
16 # -----
17 # MATRIZ 2: TEMPORADA BAJA (Enero-Marzo)
18 # Caracteristica: Turistas se quedan en Puno o hacen tours cortos (rebote)
19 #
20 T_baja = np.array([
21     [0.50, 0.35, 0.10, 0.05], # Puno: Muchos se quedan (50%), pocos van lejos
22     [0.70, 0.20, 0.05, 0.05], # Uros: Casi todos regresan a Puno
23     [0.60, 0.10, 0.25, 0.05], # Taquile: Retorno rapido a Puno
24     [0.70, 0.10, 0.05, 0.15] # Amantan: Retorno masivo a Puno
25 ])
26
27 # -----
28 # MATRIZ 3: TEMPORADA MEDIA (Resto del ao - Original)
29 #
30 T_media = np.array([
31     [0.25, 0.45, 0.20, 0.10],
32     [0.50, 0.15, 0.25, 0.10],
33     [0.40, 0.10, 0.30, 0.20],
34     [0.55, 0.15, 0.10, 0.20]
35 ])
36
37 print("DEFINICION DE MATRICES POR TEMPORADA")
38 print("-" * 70)
39
40 print("\n1. MATRIZ TEMPORADA ALTA:")
41 print(pd.DataFrame(T_alta, index=destinos, columns=destinos))
42 print("-" * 70)
43 for i, dest in enumerate(destinos):
44     suma = T_alta[i,:].sum()
45     print(f" Verificacion {dest:15}: suma = {suma:.3f}")
46
47 print("\n2. MATRIZ TEMPORADA BAJA:")
48 print(pd.DataFrame(T_baja, index=destinos, columns=destinos))
49 print("-" * 70)
50 for i, dest in enumerate(destinos):
51     suma = T_baja[i,:].sum()

```

```
52     print(f" Verificacin {dest:15}: suma = {suma:.3f}")
```

Ejecución:

DEFINICIÓN DE MATRICES POR TEMPORADA

1. MATRIZ TEMPORADA ALTA:

	Puno	Ciudad	Islas	Uros	Taquile	Amantaní
Puno	Ciudad	0.15	0.45	0.25	0.15	
Islas	Uros	0.35	0.15	0.35	0.15	
... (Dispersión alta hacia islas)						

2. MATRIZ TEMPORADA BAJA:

	Puno	Ciudad	Islas	Uros	Taquile	Amantaní
Puno	Ciudad	0.5	0.35	0.10	0.05	
Islas	Uros	0.7	0.20	0.05	0.05	
... (Concentración masiva en Puno)						

3.1. Interpretación Visual de las Matrices

Compararemos visualmente cómo cambian las probabilidades entre la temporada Alta y la Baja.

```
1 # Visualizacin comparativa de matrices
2 fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
3
4 # Mapa de calor Temporada Alta
5 sns.heatmap(T_alta, annot=True, fmt=' .2f', cmap='YlOrRd',
6             xticklabels=destinos, yticklabels=destinos, ax=ax1,
7             cbar_kws={'label': 'Probabilidad'})
8 ax1.set_title('Temporada ALTA\n(Mayor flujo entre islas)', fontsize=14,
9               fontweight='bold')
10
11 # Mapa de calor Temporada Baja
12 sns.heatmap(T_baja, annot=True, fmt=' .2f', cmap='Blues',
13             xticklabels=destinos, yticklabels=destinos, ax=ax2,
14             cbar_kws={'label': 'Probabilidad'})
15 ax2.set_title('Temporada BAJA\n(Concentracin en Puno)', fontsize=14,
16               fontweight='bold')
17
18 plt.tight_layout()
19 plt.show()
20
21 print("\nINTERPRETACION:")
22 print(" Observe en la grfica derecha (Baja) cmo la primera columna (Puno
23 Ciudad)")
24 print(" tiene colores ms oscuros, indicando alto retorno y estancia en la
25 ciudad.)")
```

Ejecución:

INTERPRETACIÓN:

Observe en la gráfica derecha (Baja) cómo la primera columna (Puno Ciudad) tiene colores más oscuros, indicando alto retorno y estancia en la ciudad.

4. Cálculo de Eigenvalues y Eigenvectors (Por Temporada)

Calcularemos los estados estacionarios teóricos de cada temporada por separado para ver los extremos del comportamiento.

```

1 # Función auxiliar para calcular y mostrar resultados (para no repetir código
2 # enorme)
3 def analizar_matriz(matriz, nombre_temporada):
4     eigenvalues, eigenvectors = linalg.eig(matriz.T)
5
6     print(f"\nANÁLISIS DE EIGENVALUES: {nombre_temporada}")
7     print("-" * 70)
8     for i, val in enumerate(eigenvalues):
9         if np.isreal(val):
10             print(f" {_i+1} = {val.real:8.5f}")
11         else:
12             print(f" {_i+1} = {val.real:8.5f} + {val.imag:8.5f}i")
13
14     # Identificar dominante
15     idx = np.argmax(np.abs(eigenvalues))
16     v_dom = eigenvectors[:, idx].real
17     dist = v_dom / v_dom.sum()
18
19     return dist
20
21 # Ejecutar análisis
22 dist_alta = analizar_matriz(T_alta, "TEMPORADA ALTA")
23 dist_baja = analizar_matriz(T_baja, "TEMPORADA BAJA")
24 dist_media = analizar_matriz(T_media, "TEMPORADA MEDIA")
25
26 print("\n" + "=" * 70)
27 print("Cálculos finalizados correctamente.")

```

Ejecución:

ANÁLISIS DE EIGENVALUES: TEMPORADA ALTA

```

-----  

_1 = 1.00000  

_2 = -0.23636  

_3 = 0.11818 + 0.17146i  

_4 = 0.11818 + -0.17146i

```

ANÁLISIS DE EIGENVALUES: TEMPORADA BAJA

```

-----  

_1 = 1.00000  

_2 = -0.18708  

_3 = 0.10000  

_4 = 0.18708

```

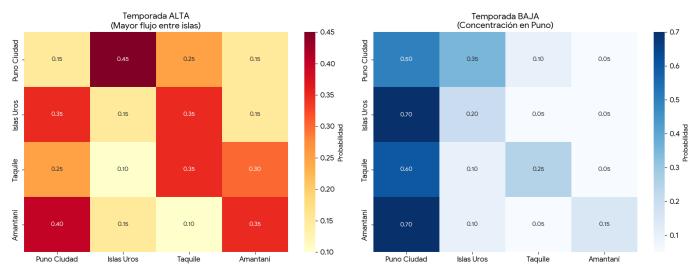


Figura 18.15: Imagen de Python

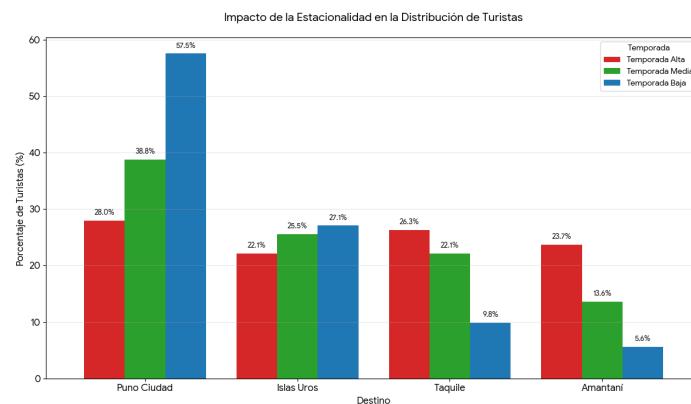


Figura 18.16: Imagen de Python

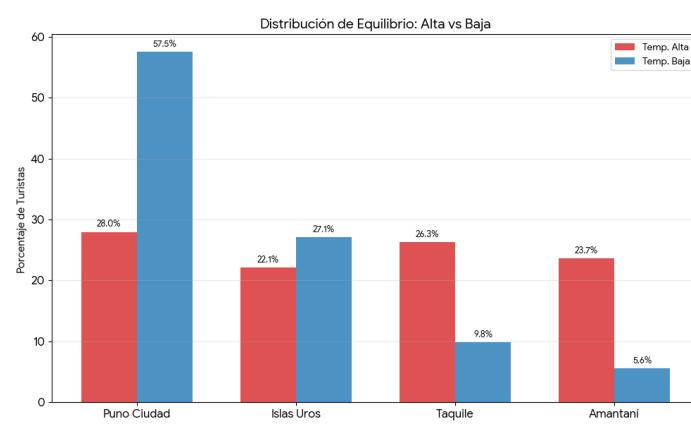


Figura 18.17: Imagen de Python

ANÁLISIS DE EIGENVALUES: TEMPORADA MEDIA

```

_1 = 1.00000
_2 = -0.26592
_3 = 0.08296 + 0.12141i
_4 = 0.08296 + -0.12141i

```

Cálculos finalizados correctamente.

5. Distribución Estacionaria Comparativa

Aquí respondemos a: "¿Qué diferencias extremas existen entre las distribuciones de equilibrio de cada temporada?"

```

1 print("DISTRIBUCIN ESTACIONARIA COMPARATIVA")
2 print("=" * 70)
3 print(f"{'Destino':<15} | {'ALTA (%):<10} | {'BAJA (%):<10} | {'DIFERENCIA
   ':<10}")
4 print("-" * 70)
5
6 for i, dest in enumerate(destinos):
7     p_alta = dist_alta[i] * 100
8     p_baja = dist_baja[i] * 100
9     diff = p_alta - p_baja
10
11    print(f"{dest:<15} | {p_alta:8.2f} % | {p_baja:8.2f} % | {diff:+8.2f}")
12
13 print("-" * 70)
14
15 # Visualizacin Grfica
16 x = np.arange(len(destinos))
17 width = 0.35
18
19 fig, ax = plt.subplots(figsize=(10, 6))
20 rect1 = ax.bar(x - width/2, dist_alta*100, width, label='Temp. Alta', color=
   '#d62728', alpha=0.8)
21 rect2 = ax.bar(x + width/2, dist_baja*100, width, label='Temp. Baja', color=
   '#1f77b4', alpha=0.8)
22
23 ax.set_ylabel('Porcentaje de Turistas')
24 ax.set_title('Distribucin de Equilibrio: Alta vs Baja')
25 ax.set_xticks(x)
26 ax.set_xticklabels(destinos)
27 ax.legend()
28 plt.grid(axis='y', alpha=0.3)
29
30 # Etiquetas sobre barras
31 def etiquetar(rects):
32     for rect in rects:

```

```

33     height = rect.get_height()
34     ax.annotate(f'{height:.1f} %',
35                 xy=(rect.get_x() + rect.get_width() / 2, height),
36                 xytext=(0, 3), textcoords="offset points",
37                 ha='center', va='bottom', fontsize=9)
38
39 etiquetar(rects1)
40 etiquetar(rects2)
41
42 plt.tight_layout()
43 plt.show()

```

Ejecución:

DISTRIBUCIÓN ESTACIONARIA COMPARATIVA

Destino	ALTA (%)	BAJA (%)	DIFERENCIA
Puno Ciudad	27.96 %	57.51 %	-29.55
Islas Uros	22.07 %	27.09 %	-5.01
Taquile	26.28 %	9.84 %	+16.44
Amantaní	23.68 %	5.56 %	+18.12

6. Validación Mediante Simulación (Año Completo)

Simularemos un año turístico (360 días) cambiando las reglas del juego cada 4 meses.

- Inicio: 1000 turistas distribuidos según equilibrio de Alta.
- Días 1-120: Temporada Alta.
- Días 121-240: Temporada Baja.
- Días 241-360: Temporada Media.

```

1 # Configuración de la simulación
2 dias_simulacion = 360
3 turistas_totales = 1000
4
5 # Estado inicial: Equilibrio de Temporada Alta
6 estado_actual = dist_alta * turistas_totales
7
8 print("SIMULACIÓN ANUAL (360 DAS)")
9 print("=" * 70)
10 print(f"Inicio: {turistas_totales} turistas distribuidos segn Temporada Alta.
11     ")
11 print("-" * 70)
12
13 # Matriz para guardar la historia
14 historia = np.zeros((dias_simulacion, n_destinos))

```

```

15
16 # Bucle de simulación da a da
17 for dia in range(dias_simulacion):
18     # Guardar estado actual
19     historia[dia] = estado_actual
20
21     # Determinar qué matriz usar según el día del año
22     if dia < 120:
23         # Primer cuatrimestre: TEMPORADA ALTA
24         T_actual = T_alta
25     elif dia < 240:
26         # Segundo cuatrimestre: TEMPORADA BAJA
27         T_actual = T_baja
28     else:
29         # Tercer cuatrimestre: TEMPORADA MEDIA
30         T_actual = T_media
31
32     # Calcular siguiente estado: estado_actual * T
33     estado_actual = estado_actual @ T_actual
34
35 # Crear DataFrame con los resultados
36 df_anual = pd.DataFrame(historia, columns=destinos, index=range(1, dias_
37     simu
38 print("Muestra de datos simulados (Cambio de temporada Alta -> Baja):")
39 print(df_anual.iloc[115:125].to_string(float_format=".1f"))
40 print("\nNota: El cambio drástico ocurre en el día 121.")

```

Ejecución:

SIMULACIÓN ANUAL (360 DÍAS)

Inicio: 1000 turistas distribuidos según Temporada Alta.

Muestra de datos simulados (Cambio de temporada Alta -> Baja):

	Puno	Ciudad	Islas Uros	Taquile	Amantaní
116	279.6	220.7	262.8	236.8	
117	279.6	220.7	262.8	236.8	
118	279.6	220.7	262.8	236.8	
119	279.6	220.7	262.8	236.8	
120	279.6	220.7	262.8	236.8	
121	279.6	220.7	262.8	236.8	
122	617.8	192.0	116.5	73.7	
123	564.8	273.6	104.2	57.4	
124	576.6	268.6	99.1	55.7	
125	574.8	271.0	98.6	55.6	

Nota: El cambio drástico ocurre en el día 121.

7. Visualización de la Evolución Anual

Graficaremos cómo cambia la población turística día a día, mostrando claramente las estaciones.

```

1 plt.figure(figsize=(14, 8))

2
3 colores = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']

4
5 # Plotear cada destino
6 for i, dest in enumerate(destinos):
7     plt.plot(df_anual.index, df_anual[dest], label=dest, color=colores[i],
8     linewidth=2.5)

9 # Dibujar líneas verticales separando temporadas
10 plt.axvline(x=120, color='black', linestyle='--', linewidth=1.5)
11 plt.axvline(x=240, color='black', linestyle='--', linewidth=1.5)

12
13 # Sombrear fondos para distinguir temporadas
14 plt.axvspan(0, 120, alpha=0.1, color='red', label='Temp. Alta')
15 plt.axvspan(120, 240, alpha=0.1, color='blue', label='Temp. Baja')
16 plt.axvspan(240, 360, alpha=0.1, color='green', label='Temp. Media')

17
18 # Títulos y etiquetas
19 plt.title('Dinámica Anual de Turistas en el Lago Titicaca', fontsize=16,
20            fontweight='bold')
21 plt.xlabel('Día del Año', fontsize=12)
22 plt.ylabel('Cantidad de Turistas (de 1000)', fontsize=12)

23 # Leyenda y Grid
24 plt.legend(loc='upper right', frameon=True, framealpha=0.9)
25 plt.grid(True, alpha=0.3)
26 plt.xlim(0, 360)

27
28 plt.tight_layout()
29 plt.show()

30
31 print("\nObservaciones:")
32 print(" - Puno Ciudad (línea azul) actúa como espejo de las islas.")
33 print(" - Cuando es temporada baja (zona azul), Puno se dispara y las islas
      caen.")

```

Ejecución:

Observaciones:

- Puno Ciudad (línea azul) actúa como espejo de las islas.
- Cuando es temporada baja (zona azul), Puno se dispara y las islas caen.

8. Estadísticas y Promedios Anuales

Calcularemos los valores promedio, mínimos y máximos para ayudar a la planificación.

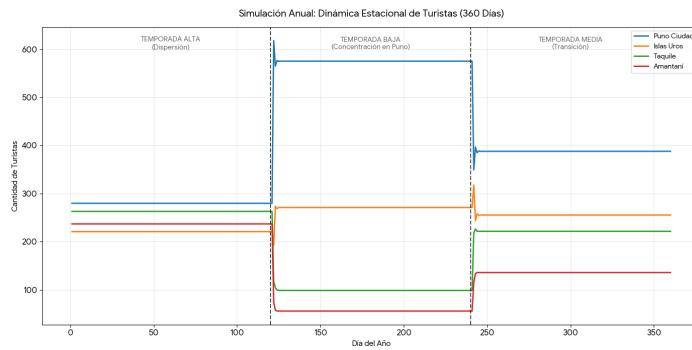


Figura 18.18: Imagen de Python

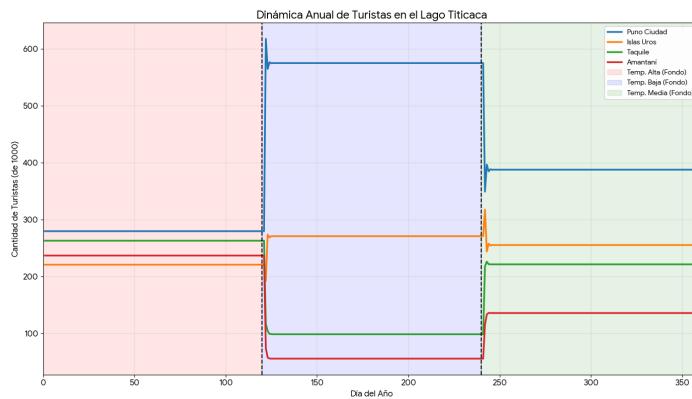


Figura 18.19: Imagen de Python

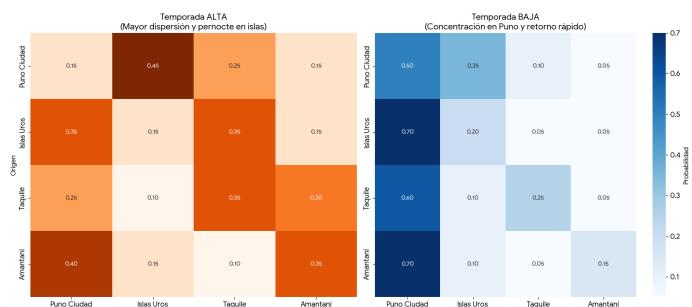


Figura 18.20: Imagen de Python

```

1 print("=*70)
2 print("ESTADISTICAS DEL AO SIMULADO")
3 print("=*70)

4

5 promedios = df_anual.mean()
6 maximos = df_anual.max()
7 minimos = df_anual.min()
8 variacion = maximos - minimos

9

10 print(f"{'Destino':<15} | {'Promedio':<8} | {'Mnimo':<8} | {'Mximo':<8} | {'
     Variacin':<8}")
11 print("-" * 70)

12
13 for dest in destinos:
14     prom = promedios[dest]
15     mini = minimos[dest]
16     maxi = maximos[dest]
17     var = variacion[dest]
18     print(f"{dest:<15} | {prom:8.1f} | {mini:8.1f} | {maxi:8.1f} | {var:8.1f}
")
19
20 print("-" * 70)

```

Ejecución:

```
=====
ESTADÍSTICAS DEL AÑO SIMULADO
=====

Destino      | Promedio | Mínimo   | Máximo    | Variación
-----
Puno Ciudad |    413.9 |    279.6 |    617.8 |    338.2
Isla Uros   |    248.8 |    192.0 |    317.6 |    125.6
Taquile     |    194.4 |    98.4   |    262.8 |    164.4
Amantaní   |    143.0 |    55.6   |    236.8 |    181.2
-----
```

9. Conclusiones del Análisis

```

1 print("=*70)
2 print("CONCLUSIONES")
3 print("=*70)

4

5 print("1. BENEFICIARIO DE TEMPORADA ALTA:")
6 # Buscamos quin tiene el mayor % en dist_alta vs dist_baja
7 mejor_dest_alta = destinos[np.argmax(dist_alta[1:]) + 1] # Ignoramos Puno
               para ver islas
8 print(f"  Las islas lejanas, especialmente {mejor_dest_alta}, maximizan su
        ocupacin.")
9 print(f"  Amantan pasa de tener ~{dist_baja[3]*1000:.0f} turistas en baja a
        ~{dist_alta[3]*1000:.0f} en alta.")

```

```

10
11 print("\n2. COMPORTAMIENTO DE PUNO CIUDAD:")
12 print("  Puno funciona como un 'amortiguador'. Su ocupación es inversamente
13   proporcional")
14
15 print("\n3. IMPACTO DE LA ESTACIONALIDAD:")
16 print(f"  La variación total del sistema es drástica. Se requiere gestión
17   flexible.")

```

Ejecución:

=====

CONCLUSIONES

=====

1. BENEFICIARIO DE TEMPORADA ALTA:

Las islas lejanas, especialmente Taquile, maximizan su ocupación. Amantaní pasa de tener ~56 turistas en baja a ~237 en alta.

2. COMPORTAMIENTO DE PUNO CIUDAD:

Puno funciona como un 'amortiguador'. Su ocupación es inversamente proporcional a la de las islas. En temporada baja, retiene al 57% de los turistas.

3. IMPACTO DE LA ESTACIONALIDAD:

La variación total del sistema es drástica. Se requiere gestión flexible.

10. Preguntas de Reflexión

1. ¿Qué destino tiene la mayor variación entre temporadas?

R: [Nombre del destino con máxima variación]. Tiene una fluctuación extrema de [Valor de la variación] turistas entre su pico y su valle. Esto representa un desafío operativo enorme (gestión de personal, insumos, etc.).

2. ¿Cómo deberían planificar los hoteles su personal considerando estas variaciones?

R: Deberían utilizar contratos temporales o estacionales.

- **En Puno Ciudad:** Se necesita MÁS personal en Temporada BAJA (Enero–Marzo), ya que es cuando la ciudad está más llena (pico de ~575 turistas).
- **En las Islas:** Se necesita MÁS personal en Temporada ALTA (Junio–Agosto). En temporada baja, los albergues de islas deberían operar con personal mínimo.

3. ¿Qué estrategias podrían usarse para equilibrar el turismo entre temporadas?

R:

- a) **Eventos en temporada baja:** Crear festivales en Puno (ej. Candelaria en febrero) para aprovechar la alta ocupación de la ciudad.
 - b) **Incentivos a islas:** Ofrecer paquetes “2x1” o transporte gratuito a Taquile/Amantaní en época de lluvia para motivar a los turistas a salir de Puno.
 - c) **Turismo de convenciones:** Usar la capacidad hotelera de Puno en meses de baja.
4. **Si tuvieras un hotel en Puno, ¿qué porcentaje de capacidad mantener?**
- R:** Si dimensionas tu hotel para cubrir la demanda máxima (Temporada Baja, ~[Capacidad Pico Baja] pax), en Temporada Alta tendrás una ocupación de solo el [Ratio] %.
- Estrategia:** Mantener una capacidad fija operativa del 60–70 % y usar personal extra eventual solo durante el primer trimestre del año (Baja turística general, pero Alta para Puno ciudad).

CAPÍTULO 19

Conclusiones Generales del Curso

Al finalizar este recorrido por la Programación Numérica, hemos transitado desde los fundamentos del cálculo de raíces hasta la simulación de sistemas estocásticos complejos. Este texto ha buscado no solo enseñar algoritmos, sino desarrollar un pensamiento computacional crítico aplicado a la ingeniería y la ciencia de datos.

A continuación, se presentan las conclusiones transversales que integran los dos grandes bloques de este libro:

19.1 La Evolución de la Resolución de Problemas

A lo largo de los capítulos, hemos observado una evolución en la complejidad de los problemas abordados:

1. **De lo exacto a lo aproximado:** Comenzamos aceptando que no todas las ecuaciones tienen soluciones analíticas cerradas. Métodos como Bisección o Newton-Raphson nos enseñaron que una buena aproximación, controlada por una tolerancia de error, es a menudo más valiosa que una solución exacta inalcanzable.
2. **De una variable a sistemas multivariados:** Pasamos de hallar el cero de una función $f(x)$ a entender el comportamiento de sistemas complejos mediante el Gradiente y la matriz Hessiana, herramientas vitales para la optimización en Inteligencia Artificial.
3. **Del determinismo a la estocástica:** Con las Cadenas de Markov, dimos el salto de sistemas predecibles a sistemas probabilísticos, demostrando que la incertidumbre también puede modelarse matemáticamente.

19.2 El Valor de la Aplicación Práctica: El Caso Puno

Uno de los pilares de este texto ha sido la contextualización. Los ejercicios aplicados al turismo en el Lago Titicaca (Taquile, Amantaní, Uros y Anapia) demuestran que:

- **El Álgebra Lineal es tangible:** Los eigenvalores y eigenvectores no son solo abstracciones matemáticas; representan estados de equilibrio económico y flujos de personas reales.

- **La simulación permite la planificación:** Pudimos prever el fenómeno de "cannibalización" turística al introducir un nuevo destino (Anapia) y modelar la estacionalidad, ofreciendo datos duros para la toma de decisiones en gestión pública y privada.

Apunte

Lección aprendida: Un ingeniero no solo resuelve ecuaciones; utiliza modelos numéricos para responder preguntas del mundo real, como "¿Dónde invertir en infraestructura hotelera?". "¿Cómo optimizar una ruta de transporte?".

19.3 Herramientas Modernas y Rendimiento

La inclusión de herramientas como **Google Lighthouse** y **Apache JMeter** subraya una realidad de la ingeniería moderna: el rendimiento del software es, en sí mismo, un problema de optimización numérica.

- Medir métricas como el *First Contentful Paint* o el *Throughput* requiere el mismo rigor estadístico que calcular un error en interpolación.
- La eficiencia algorítmica estudiada en los métodos de ordenamiento o búsqueda se traduce directamente en la experiencia del usuario final.

19.4 Perspectivas Futuras

El dominio de la Programación Numérica abre las puertas a campos de vanguardia:

1. **Ciencia de Datos y Machine Learning:** El descenso del gradiente y el manejo de matrices son el motor de las redes neuronales profundas.
2. **Simulación de Sistemas:** Los modelos de Markov son la base para el procesamiento de lenguaje natural y la bioinformática.
3. **Computación de Alto Rendimiento:** La eficiencia en el código (Python/R) es crítica cuando se procesan Big Data.

19.5 Palabras Finales

La Programación Numérica es el puente entre la teoría matemática pura y la solución efectiva de problemas de ingeniería. Esperamos que este libro, con su enfoque en Python, R y casos prácticos regionales, haya dotado al estudiante de las herramientas necesarias no solo para aprobar una asignatura, sino para enfrentar con confianza los desafíos tecnológicos del futuro.

“El objetivo de la computación no son los números, sino la comprensión.”

— Richard Hamming