Métodos Numéricos

Análisis y Aplicaciones Computacionales

Metodos numericos

∞

Ronald Incacutipa – Dilan Coaguila — Cristian Paucar

Universidad Nacional del ALtiplano Puno

~~Año Académico 2024-2025~~

18 de diciembre de 2025

Prefacio

Los métodos numéricos constituyen una herramienta fundamental en la formación de ingenieros, científicos y matemáticos aplicados. Este libro ha sido elaborado con el propósito de proporcionar una introducción rigurosa pero accesible a los principales métodos numéricos.

Objetivos del curso

Al finalizar este curso, el estudiante será capaz de:

- Comprender los fundamentos teóricos de los principales métodos numéricos
- Implementar algoritmos numéricos eficientes
- Analizar la convergencia y estabilidad de los métodos
- Aplicar métodos numéricos a problemas reales
- Evaluar críticamente los resultados numéricos obtenidos

Estructura del libro

El libro está organizado en capítulos que corresponden a las semanas del curso. Cada capítulo incluirá:

- Desarrollo teórico de los conceptos
- Ejemplos resueltos
- Algoritmos implementables
- Ejercicios propuestos
- Aplicaciones prácticas

Ronald Incacutipa – Dilan Coaguila — Cristian Paucar

18 de diciembre de 2025

Índice general

Prefacio I

1. Introducción a los Métodos Numéricos 1

1.1. Motivación . . . 1

1.1.1. El problema de la solución exacta . . . 1

1.1.2. ¿Qué son los métodos numéricos? . . . 2

1.1.3. Diferencia entre solución exacta y aproximada . . . 2

1.1.4. Breve historia de los métodos numéricos . . . 3

1.1.5. ¿Por qué estudiar métodos numéricos? . . . 3

1.2. Aplicaciones . . . 3

1.2.1. Ingeniería . . . 4

1.2.2. Ciencias Físicas . . . 5

1.2.3. Ciencias de la Computación . . . 5

1.2.4. Economía y Finanzas . . . 5

1.2.5. Ciencias de la Tierra y Medio Ambiente . . . 6

1.2.6. Medicina y Biología . . . 6

1.2.7. Otras aplicaciones . . . 6

1.3. Conceptos fundamentales . . . 7

1.3.1. Representación de números en computadora . . . 7

1.3.2. Tipos de errores . . . 7

1.3.3. Cifras significativas . . . 9

1.3.4. Propagación de errores . . . 9

1.3.5. Condicionamiento de un problema . . . 11

1.3.6. Estabilidad de un algoritmo . . . 11

1.3.7. Convergencia . . . 12

1.4. Ejercicios propuestos . . . 12

Lecturas recomendadas . . . 16

2. Variables y Funciones 18

2.1. Variables . . . 18

2.1.1. Definición de variable . . . 18

2.1.2. Clasificación de variables . . . 19
2.1.3. Notación y convenciones . . . 21
2.1.4. Variables en problemas numéricos . . . 21
2.1.5. Dominio de una variable . . . 22
2.2. Funciones matemáticas . . . 22
2.2.1. Definición formal de función . . . 23
2.2.2. Representación de funciones . . . 23
2.2.3. Clasificación de funciones . . . 24
2.2.4. Propiedades de funciones . . . 28
2.2.5. Operaciones con funciones . . . 30
2.2.6. Funciones importantes en métodos numéricos . . . 32
2.3. Dominio y rango . . . 32
2.3.1. Determinación del dominio . . . 32
2.3.2. Determinación del rango . . . 33
2.3.3. Ejemplos detallados . . . 35
2.3.4. Dominio y rango en problemas aplicados . . . 36
2.3.5. Restricción de dominio . . . 36
2.3.6. Tabla de dominios y rangos de funciones comunes . . . 37
2.4. Ejercicios propuestos . . . 37
2.4.1. Implementaciones computacionales . . . 37
Lecturas recomendadas . . . 46
2.4.2. Ejercicios teóricos . . . 54
Lecturas recomendadas . . . 58

3. Restricciones **59**
3.1. Restricciones matemáticas . . . 59
3.1.1. Definición formal . . . 60
3.1.2. Motivación: ¿Por qué necesitamos restricciones? . . . 60
3.1.3. Región factible . . . 61
3.1.4. Implementación básica en Python . . . 61
3.1.5. Implementación básica en R . . . 63
3.2. Tipos de restricciones . . . 64
3.2.1. Clasificación según la igualdad/desigualdad . . . 64
3.2.2. Clasificación según la linealidad . . . 65
3.2.3. Clasificación según el dominio . . . 66
3.2.4. Restricciones activas e inactivas . . . 66
3.2.5. Visualización de restricciones en Python . . . 67
3.2.6. Visualización de restricciones en R . . . 70
3.3. Problemas con restricciones . . . 72

3.3.1. Problema de programación lineal . . . 72
3.3.2. Problema de optimización no lineal con restricciones . . . 76
3.3.3. Problema de asignación de recursos . . . 81
3.3.4. Análisis de sensibilidad . . . 84
3.4. Ejercicios propuestos . . . 86
3.4.1. Ejercicios teóricos . . . 86
3.4.2. Ejercicios de programación . . . 87
Lecturas recomendadas . . . 90

4. Soluciones Iterativas para Raíces **92**
4.1. Introducción a métodos iterativos . . . 92
4.1.1. Criterios de parada . . . 92
4.2. Concepto de raíz . . . 92
4.3. Métodos de búsqueda . . . 93
4.3.1. Método de bisección . . . 93
4.3.2. Método de Newton-Raphson . . . 95
4.3.3. Método de la secante . . . 97
4.3.4. Comparación de métodos . . . 99
4.4. Ejercicios propuestos . . . 101
Lecturas recomendadas . . . 102

5. Metodos Numericos para Encontrar Raices **103**
5.1. Introduccion . . . 103
5.1.1. El Problema Fundamental . . . 103
5.2. Metodo de Biseccion . . . 104
5.2.1. Teoria Fundamentada . . . 104
5.2.2. Algoritmo Paso a Paso . . . 104
5.2.3. Analisis de Convergencia . . . 105
5.2.4. Implementacion Practica en Python . . . 105
5.3. Metodo de la Secante . . . 106
5.3.1. Fundamentacion Matematica . . . 106
5.3.2. Algoritmo . . . 107
5.3.3. Implementacion con Manejo de Errores . . . 107
5.4. Metodo de Regula Falsi (Falsa Posicion) . . . 109
5.4.1. Fundamentos Teoricos . . . 109
5.4.2. Implementacion Completa . . . 109
5.5. Metodo de Punto Fijo . . . 111
5.5.1. Teoria de Punto Fijo . . . 111
5.5.2. Transformaciones Comunes . . . 111
5.5.3. Implementacion con Analisis de Convergencia . . . 111

6. Diferenciación Numérica y Teoría de Diferencias Finitas 114
6.1. Introducción a la Diferenciación Numérica . . . 116
6.1.1. El Dilema Fundamental: Continuidad vs Discretización . . . 116
6.1.2. Ejemplos de la Vida Real donde las Derivadas Analíticas Fallan . . 116
6.2. Teoría Matemática Detallada . . . 117
6.2.1. Series de Taylor: La Piedra Angular . . . 117
6.2.2. Deducción Rigurosa de Fórmulas . . . 117
6.3. Métodos Avanzados de Diferenciación Numérica . . . 118
6.3.1. Fórmulas de Alta Precisión . . . 118
6.3.2. Derivadas Parciales y Gradientes . . . 119
6.4. Análisis Exhaustivo de Errores . . . 119
6.4.1. La Batalla entre Dos Frentes: Truncamiento vs Redondeo . . . 119
6.4.2. El Dilema del h Óptimo: Un Compromiso Fundamental . . . 120
6.4.3. Análisis de Sensibilidad y Condicionamiento . . . 120
6.5. Aplicaciones en el Mundo Real . . . 121
6.5.1. Ingeniería y Física: Donde la Teoría Encuentra la Realidad . . . 121
6.5.2. Finanzas Cuantitativas: Matemáticas en Wall Street . . . 121
6.5.3. Machine Learning y Ciencia de Datos . . . 122
6.5.4. Procesamiento de Señales e Imágenes . . . 122
6.6. Implementación Computacional . . . 123
6.6.1. Algoritmo en Python con Optimizaciones . . . 123
6.7. Estudios de Caso: Del Laboratorio a la Industria . . . 128
6.7.1. Caso 1: Simulación de un Puente Bajo Carga Dinámica . . . 128
6.7.2. Caso 2: Algoritmo de Trading de Alta Frecuencia . . . 128
6.7.3. Caso 3: Diagnóstico de Enfermedades Cardíacas . . . 128
6.8. Ejercicios y Proyectos Integradores . . . 129
6.8.1. Ejercicios Teóricos con Soluciones . . . 129
6.8.2. Proyecto Final: Sistema de Monitoreo de Salud Estructural . . . 130
6.9. Tablas de Referencia y Coeficientes . . . 131
6.9.1. Tablas de Coeficientes Completas . . . 131
6.10. Consideraciones Avanzadas y Tendencias Futuras . . . 132
6.10.1. Límites Fundamentales de la Diferenciación Numérica . . . 132
6.10.2. Técnicas Modernas y Alternativas . . . 132
6.10.3. Tendencias Futuras e Investigación . . . 133
6.11. Recomendaciones Prácticas y Mejores Prácticas . . . 133
6.12. Conclusión: El Arte y la Ciencia de la Diferenciación Numérica . . . 134

7. Interpolación Numérica 137
7.1. Introducción . . . 137

7.2. Formulación Matemática del Problema . . . 137
7.2.1. Definición Formal . . . 137
7.2.2. Existencia y Unicidad . . . 138
7.3. Interpolación Lineal . . . 138
7.3.1. Método y Fórmula . . . 138
7.3.2. Análisis del Error . . . 138
7.4. Interpolación Polinomial . . . 138
7.4.1. Forma de Lagrange . . . 138
7.4.2. Forma de Newton con Diferencias Divididas . . . 139
7.5. Splines Cúbicos . . . 140
7.5.1. Concepto y Motivación . . . 140
7.5.2. Definición Formal . . . 140
7.5.3. Condiciones de Suavidad . . . 141
7.5.4. Sistema de Ecuaciones para Splines Naturales . . . 141
7.6. Análisis de Errores . . . 141
7.6.1. Error en Interpolación Polinomial . . . 141
7.6.2. Fenómeno de Runge . . . 142
7.6.3. Nodos de Chebyshev . . . 142
7.7. Comparación de Métodos . . . 142
7.8. Implementación Computacional . . . 143
7.8.1. Interpolación Lineal en Python . . . 143
7.8.2. Interpolacion de Lagrange . . . 144
7.8.3. Splines Cubicos Naturales . . . 146
7.9. Aplicaciones Prácticas . . . 149
7.9.1. Procesamiento de Señales Digitales . . . 149
7.9.2. Gráficos por Computadora . . . 149
7.9.3. Ingeniería y Ciencia . . . 150
7.9.4. Finanzas Cuantitativas . . . 150
7.10. Ejercicios Resueltos . . . 150
7.10.1. Ejercicio 1: Error de Interpolación Lineal . . . 150
7.10.2. Ejercicio 2: Polinomio Interpolante de Lagrange . . . 151
7.10.3. Ejercicio 3: Spline Cúbico . . . 151
7.11. Resumen del Capítulo . . . 152

8. Eigenvalores, Eigenvectores y Métodos de Optimización 154
8.1. Introducción Conceptual: ¿Qué son Eigenvalores y Eigenvectores? . . . 155
8.1.1. La Definición Fundamental . . . 155
8.1.2. Ejemplos Visuales de la Vida Real . . . 155
8.2. Teoría Matemática Detallada: Cálculo de Eigenvalores . . . 156

8.2.1. La Ecuación Característica: La Puerta de Entrada 156
8.2.2. Algoritmo Paso a Paso para Matrices 2×2 156
8.2.3. Ejemplo Detallado 1: Matriz Simétrica 156
8.3. Propiedades Algebraicas Fundamentales 157
8.3.1. Teoremas Clave que Debes Conocer 157
8.3.2. Propiedades por Tipo de Matriz 158
8.4. Conexión con Optimización: La Matriz Hessiana 158
8.4.1. ¿Por qué son Importantes en Optimización? 158
8.4.2. Ejemplo 2: Clasificación Completa de Punto Crítico 159
8.5. Implementacion Practica en R 160
8.5.1. Calculo Directo de Eigenvalores y Eigenvectores 160
8.5.2. Clasificacion de Puntos Criticos en R 160
8.5.3. Metodo de la Potencia para Eigenvalor Dominante 161
8.6. Aplicaciones en el Mundo Real 162
8.6.1. Analisis de Componentes Principales (PCA) 162
8.6.2. Sistemas Dinamicos y Estabilidad 166
8.7. Ejercicios y Problemas Resueltos 170
8.7.1. Ejercicios Teóricos 170
8.8. Tablas de Referencia 171
8.9. Conclusiones y Recomendaciones Prácticas 172

9. Aplicación Práctica: Optimización de Páginas Web con Métodos Numéricos 175
9.1. Contexto del Estudio 175
9.1.1. Herramientas Utilizadas 175
9.2. Objetivos del Estudio 176
9.3. Pruebas de Rendimiento con Apache JMeter 176
9.3.1. Configuración de Pruebas 176
9.3.2. Datos de Rendimiento Obtenidos 176
9.4. Resultados con Google Lighthouse 177
9.4.1. YouTube 177
9.4.2. Aula Virtual UNA Puno 177
9.5. Aplicación de Métodos Numéricos para Optimización 177
9.5.1. Interpolación para Estimación de Límites 177
9.5.2. Método Regula Falsi para Encontrar Límites de Error 178
9.5.3. Resultados de Optimización Numérica 179
9.6. Análisis de Resultados y Recomendaciones 180
9.6.1. Interpretación para YouTube 180
9.6.2. Interpretación para Aula Virtual 180

9.6.3. Comparación de Métodos Numéricos Aplicados 181
9.6.4. Recomendaciones Finales por Plataforma 181
9.6.5. Sinergia entre Métodos Numéricos y Herramientas de Prueba 181
9.7. Conclusiones y Trabajo Futuro 182
9.7.1. Conclusiones Principales 182
9.7.2. Recomendaciones Finales por Plataforma 183
9.7.3. Trabajo Futuro e Investigación 183
9.7.4. Impacto del Estudio 183
9.7.5. Lecciones Aprendidas 184

Bibliografía **185**

Capítulo 1

Introducción a los Métodos Numéricos

Objetivos del capítulo:

- Comprender la importancia y necesidad de los métodos numéricos
- Identificar problemas que requieren soluciones numéricas
- Conocer las principales áreas de aplicación
- Dominar los conceptos fundamentales de error y precisión

1.1. Motivación

1.1.1. El problema de la solución exacta

En el estudio de las matemáticas, frecuentemente nos encontramos con problemas que poseen soluciones analíticas o exactas. Por ejemplo, la ecuación cuadrática $ax^2 + bx + c = 0$ tiene la solución bien conocida:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1.1}$$

Sin embargo, la realidad es que la mayoría de los problemas matemáticos que surgen en aplicaciones prácticas de ingeniería, física, economía y otras ciencias **no tienen soluciones analíticas** o estas son extremadamente complejas de obtener.

Ejemplo

Consideremos algunos problemas cotidianos que no tienen solución analítica simple:

1. Encontrar las raíces de $f(x) = e^x - 3x^2 = 0$
2. Calcular $\int_0^1 e^{-x^2} dx$ (función de error de Gauss)
3. Resolver el sistema no lineal:

$$x^2 + y^2 = 4$$
$$e^x + y = 3$$

4. Determinar la trayectoria de un proyectil con resistencia del aire

1.1.2. ¿Qué son los métodos numéricos?

Definición

Los **métodos numéricos** son técnicas mediante las cuales es posible formular problemas matemáticos de tal forma que puedan resolverse usando operaciones aritméticas básicas (suma, resta, multiplicación y división) de manera iterativa o directa.

Estas técnicas permiten obtener **soluciones aproximadas** que, para fines prácticos, son suficientemente precisas y pueden calcularse en un tiempo razonable utilizando computadoras.

1.1.3. Diferencia entre solución exacta y aproximada

Característica	**Solución Exacta**	**Solución Numérica**
Representación	Fórmula cerrada	Valor decimal
Precisión	Infinita (teórica)	Finita (controlable)
Tiempo de cálculo	Variable	Predecible
Disponibilidad	Limitada	Amplia
Ejemplo	$x = \ln(2)$	$x \approx 0{,}693147$

Tabla 1.1: Comparación entre solución exacta y numérica

1.1.4. Breve historia de los métodos numéricos

Los métodos numéricos tienen una larga historia que se remonta a las civilizaciones antiguas:

- **1650 a.C.**: Los babilonios usaban métodos iterativos para calcular raíces cuadradas
- **250 a.C.**: Arquímedes aproximó el valor de π usando polígonos
- **Siglo XVII**: Newton y Leibniz desarrollaron el cálculo, base de muchos métodos modernos
- **1940s**: Con la llegada de las computadoras digitales, los métodos numéricos experimentaron un desarrollo explosivo
- **Actualidad**: Son fundamentales en simulaciones, inteligencia artificial, modelado climático, finanzas, etc.

Nota Importante

El desarrollo de las computadoras modernas ha hecho que los métodos numéricos sean indispensables. Problemas que antes tomaban años ahora se resuelven en segundos.

1.1.5. ¿Por qué estudiar métodos numéricos?

1. **Necesidad práctica**: La mayoría de problemas reales no tienen solución analítica
2. **Velocidad**: Los métodos numéricos permiten obtener resultados rápidamente
3. **Flexibilidad**: Se adaptan a problemas complejos y no lineales
4. **Interdisciplinariedad**: Son útiles en todas las áreas científicas y tecnológicas
5. **Fundamento de software**: Detrás de MATLAB, Python científico, software de ingeniería

1.2. Aplicaciones

Los métodos numéricos tienen aplicaciones en prácticamente todas las disciplinas científicas y tecnológicas modernas. A continuación, exploramos algunas de las más importantes.

1.2.1. Ingeniería

Ingeniería Civil y Estructural

- **Análisis de estructuras**: Cálculo de esfuerzos, deformaciones y vibraciones en edificios, puentes y presas
- **Método de elementos finitos (FEM)**: Discretización de estructuras complejas para análisis de resistencia
- **Diseño sísmico**: Simulación de respuesta estructural ante terremotos
- **Mecánica de suelos**: Modelado de asentamientos y estabilidad de taludes

Ejemplo de aplicación

En el diseño de un puente, se utiliza el método de elementos finitos para:

1. Discretizar la estructura en miles de elementos
2. Resolver un sistema de ecuaciones lineales de gran dimensión ($Ax = b$)
3. Determinar desplazamientos y tensiones en cada punto
4. Verificar que la estructura cumple con códigos de seguridad

Este proceso requiere resolver sistemas con más de 100,000 ecuaciones, imposible sin métodos numéricos.

Ingeniería Mecánica

- **Dinámica de fluidos computacional (CFD)**: Análisis de flujo de aire en aeronaves, automóviles
- **Transferencia de calor**: Distribución de temperatura en motores, intercambiadores de calor
- **Diseño de componentes**: Optimización de formas para minimizar peso y maximizar resistencia
- **Vibraciones mecánicas**: Análisis modal de maquinaria

Ingeniería Eléctrica y Electrónica

- **Análisis de circuitos**: Resolución de redes eléctricas complejas
- **Procesamiento de señales**: Filtrado, análisis de Fourier, compresión

- **Diseño de antenas**: Simulación electromagnética
- **Sistemas de control**: Diseño de controladores PID, control óptimo

1.2.2. Ciencias Físicas

Física

- **Mecánica cuántica**: Resolución de la ecuación de Schrödinger
- **Física de partículas**: Simulaciones de colisiones en aceleradores (CERN)
- **Astrofísica**: Modelado de galaxias, estrellas y agujeros negros
- **Física del estado sólido**: Cálculo de propiedades de materiales

Química

- **Química cuántica**: Cálculo de orbitales moleculares
- **Dinámica molecular**: Simulación de comportamiento de moléculas
- **Cinética química**: Modelado de velocidades de reacción
- **Termodinámica**: Cálculo de propiedades de equilibrio

1.2.3. Ciencias de la Computación

- **Machine Learning**: Algoritmos de optimización para entrenar redes neuronales
- **Visión computacional**: Procesamiento y análisis de imágenes
- **Gráficos por computadora**: Rendering, animación 3D
- **Criptografía**: Algoritmos de factorización, números primos
- **Compresión de datos**: Algoritmos de transformada (JPEG, MP3)

1.2.4. Economía y Finanzas

- **Valoración de derivados**: Modelo de Black-Scholes para opciones
- **Optimización de portafolios**: Teoría moderna de portafolios
- **Análisis de riesgo**: Simulaciones de Monte Carlo
- **Econometría**: Estimación de parámetros en modelos económicos

- **Predicción de series temporales**: Modelos ARIMA, GARCH

Valoración de opciones

El precio de una opción europea se calcula mediante la ecuación de Black-Scholes:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \tag{1.2}$$

Esta ecuación diferencial parcial no tiene solución analítica para opciones complejas, requiriendo métodos numéricos como diferencias finitas o simulación de Monte Carlo.

1.2.5. Ciencias de la Tierra y Medio Ambiente

- **Modelado climático**: Predicción del cambio climático
- **Meteorología**: Pronóstico del tiempo
- **Sismología**: Localización de epicentros, predicción de tsunamis
- **Hidrología**: Modelado de flujo de agua subterránea
- **Contaminación**: Dispersión de contaminantes en atmósfera y océanos

1.2.6. Medicina y Biología

- **Imagenología médica**: Reconstrucción de imágenes (TAC, MRI)
- **Modelado de epidemias**: Predicción de propagación de enfermedades
- **Genómica**: Análisis de secuencias de ADN
- **Farmacología**: Diseño de fármacos, farmacocinética
- **Biomecánica**: Modelado de flujo sanguíneo, funcionamiento de órganos

1.2.7. Otras aplicaciones

Área	**Aplicaciones**
Manufactura	Optimización de procesos, control de calidad
Logística	Rutas óptimas, gestión de inventarios
Telecomunicaciones	Procesamiento de señales, compresión
Aeroespacial	Diseño de aeronaves, trayectorias de satélites
Energía	Redes eléctricas, energías renovables
Agricultura	Optimización de riego, predicción de cosechas

Tabla 1.2: Aplicaciones adicionales de métodos numéricos

1.3. Conceptos fundamentales

Para trabajar efectivamente con métodos numéricos, es esencial comprender ciertos conceptos fundamentales relacionados con la precisión, exactitud y propagación de errores.

1.3.1. Representación de números en computadora

Las computadoras representan números de forma finita, lo que introduce limitaciones:

Números de punto flotante

Un número en formato IEEE 754 (estándar de precisión doble) se representa como:

$$x = \pm d_0.d_1 d_2 \dots d_{51} \times 2^e \tag{1.3}$$

donde:

- $d_i \in \{0, 1\}$ son dígitos binarios (bits)
- e es el exponente $(-1022 \leq e \leq 1023)$

Nota Importante

En precisión doble:

- Rango aproximado: 10^{-308} a 10^{308}
- Precisión: aproximadamente 16 dígitos decimales
- Espaciamiento relativo entre números (epsilon de máquina): $\epsilon \approx 2{,}22 \times 10^{-16}$

Épsilon de máquina

Definición

El **epsilon de máquina** (ϵ_m) es el número positivo más pequeño tal que:

$$1 + \epsilon_m > 1 \tag{1.4}$$

en aritmética de punto flotante.

Para precisión doble: $\epsilon_m \approx 2{,}22 \times 10^{-16}$

1.3.2. Tipos de errores

En el análisis numérico, distinguimos varios tipos de errores que afectan la precisión de los resultados.

Error absoluto

Definición

El **error absoluto** es la diferencia entre el valor verdadero y el valor aproximado:

$$E_a = |x_{\text{verdadero}} - x_{\text{aproximado}}| \tag{1.5}$$

Ejemplo

Si $\pi \approx 3{,}14159$ es nuestra aproximación y $\pi = 3{,}141592653\ldots$ es el valor verdadero:

$$E_a = |3{,}141592653 - 3{,}14159| \approx 2{,}653 \times 10^{-6} \tag{1.6}$$

Error relativo

Definición

El **error relativo** normaliza el error absoluto respecto al valor verdadero:

$$E_r = \frac{|x_{\text{verdadero}} - x_{\text{aproximado}}|}{|x_{\text{verdadero}}|} \times 100\,\% \tag{1.7}$$

El error relativo es más útil cuando comparamos magnitudes diferentes.

Ejemplo

Para la aproximación de π:

$$E_r = \frac{2{,}653 \times 10^{-6}}{3{,}141592653} \times 100\,\% \approx 0{,}000084\,\% \tag{1.8}$$

Error de redondeo

Definición

El **error de redondeo** surge de la representación finita de números en la computadora.

Ejemplo

El número $1/3 = 0{,}333333\ldots$ no puede representarse exactamente en forma decimal finita. En una computadora con 6 dígitos:

$$\frac{1}{3} \approx 0{,}333333 \tag{1.9}$$

$$\text{Error de redondeo} = |0{,}333333\ldots - 0{,}333333| \approx 3{,}33 \times 10^{-7} \tag{1.10}$$

Error de truncamiento

Definición

El **error de truncamiento** ocurre cuando aproximamos un proceso infinito mediante uno finito.

Ejemplo

La serie de Taylor para e^x es:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \tag{1.11}$$

Si truncamos después de 3 términos:

$$e^x \approx 1 + x + \frac{x^2}{2} \tag{1.12}$$

El error de truncamiento es la suma de todos los términos omitidos.

1.3.3. Cifras significativas

Definición

Las **cifras significativas** de un número son todos los dígitos que se conocen con certeza, más el primer dígito incierto.

Ejemplo

El número $x = 3{,}14159$ tiene 6 cifras significativas.
Si decimos que $\pi \approx 3{,}14$ con 3 cifras significativas, estamos indicando:

$$E_r < 0{,}5 \times 10^{-(3-1)} = 0{,}005 = 0{,}5\,\% \tag{1.13}$$

Teorema

Si x^* aproxima a x con n cifras significativas, entonces:

$$\frac{|x - x^*|}{|x|} \leq 5 \times 10^{-n} \tag{1.14}$$

1.3.4. Propagación de errores

Cuando realizamos operaciones con números que contienen errores, estos se propagan y pueden amplificarse.

Suma y resta

Para $z = x + y$ o $z = x - y$:

$$|E_z| \approx |E_x| + |E_y| \tag{1.15}$$

Los errores absolutos se suman.

Multiplicación y división

Para $z = x \times y$ o $z = x/y$:

$$\frac{|E_z|}{|z|} \approx \frac{|E_x|}{|x|} + \frac{|E_y|}{|y|} \tag{1.16}$$

Los errores relativos se suman.

Propagación de error

Sean $x = 4{,}53 \pm 0{,}02$ y $y = 2{,}11 \pm 0{,}03$

Suma:

$$z = x + y = 6{,}64 \tag{1.17}$$
$$E_z \approx 0{,}02 + 0{,}03 = 0{,}05 \tag{1.18}$$
$$z = 6{,}64 \pm 0{,}05 \tag{1.19}$$

Producto:

$$z = x \times y = 9{,}5583 \tag{1.20}$$
$$\frac{E_z}{|z|} \approx \frac{0{,}02}{4{,}53} + \frac{0{,}03}{2{,}11} \approx 0{,}0186 \tag{1.21}$$
$$E_z \approx 9{,}5583 \times 0{,}0186 \approx 0{,}178 \tag{1.22}$$
$$z = 9{,}56 \pm 0{,}18 \tag{1.23}$$

1.3.5. Condicionamiento de un problema

Definición

El **número de condición** mide la sensibilidad de la solución de un problema a cambios en los datos de entrada. Un problema está:

- **Bien condicionado**: si pequeños cambios en la entrada producen pequeños cambios en la salida
- **Mal condicionado**: si pequeños cambios en la entrada producen grandes cambios en la salida

Problema mal condicionado

Considere resolver:

$$x + y = 2 \tag{1.24}$$
$$1{,}0001x + y = 2{,}0001 \tag{1.25}$$

La solución es $x = 1$, $y = 1$.
Si cambiamos ligeramente el segundo lado derecho a 2,0002:

$$x + y = 2 \tag{1.26}$$
$$1{,}0001x + y = 2{,}0002 \tag{1.27}$$

La solución cambia dramáticamente a $x = 2$, $y = 0$. Este sistema está mal condicionado.

1.3.6. Estabilidad de un algoritmo

Definición

Un algoritmo es **numéricamente estable** si los errores (de redondeo, truncamiento) no crecen excesivamente durante el cálculo.

!

1.3.7. Convergencia

Definición

Una sucesión $\{x_n\}$ **converge** a L si:

$$\lim_{n\to\infty} x_n = L \tag{1.29}$$

Es decir, para todo $\epsilon > 0$, existe N tal que para todo $n > N$:

$$|x_n - L| < \epsilon \tag{1.30}$$

Orden de convergencia

Definición

Una sucesión $\{x_n\}$ que converge a L tiene **orden de convergencia** p si:

$$\lim_{n\to\infty} \frac{|x_{n+1} - L|}{|x_n - L|^p} = C \tag{1.31}$$

donde $0 < C < \infty$.

- $p = 1$: convergencia lineal
- $p = 2$: convergencia cuadrática
- $p > 2$: convergencia superlineal

Ejemplo

Si $|x_{n+1} - L| \approx 0{,}5|x_n - L|$ (convergencia lineal), el error se reduce a la mitad en cada iteración.
Si $|x_{n+1} - L| \approx |x_n - L|^2$ (convergencia cuadrática), el número de dígitos correctos se duplica en cada iteración.

1.4. Ejercicios propuestos

Ejercicio 1.1. Calcule el error absoluto y relativo al aproximar:

1. π con 3,14159
2. e con 2,71828
3. $\sqrt{2}$ con 1,414
4. $\ln(2)$ con 0,693

Ejercicio 1.2. Demuestre que el error relativo al aproximar $\sqrt{x}$ por x_n en el método babilónico

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{x}{x_n}\right) \tag{1.32}$$

se reduce cuadráticamente.

Ejercicio 1.3. Considere la ecuación $f(x) = x^3 - x - 1 = 0$.

1. Verifique gráficamente que existe una raíz en b¿Por qué este problema requiere métodos numéricos?

   Proponga una aplicación práctica donde podría aparecer esta ecuación

Ejercicio 1.4. Para el sistema mal condicionado:

$$x + y = 2 \tag{1.33}$$

$$1{,}001x + y = 2{,}001 \tag{1.34}$$

1. Resuelva exactamente
2. Cambie el segundo lado derecho a 2,002 y resuelva
3. Explique por qué pequeños cambios causan grandes variaciones

Ejercicio 1.5. Investigue y explique brevemente tres aplicaciones de métodos numéricos en su campo de estudio o interés.

Ejercicio 1.6. Demuestre que en precisión de punto flotante:

$$(a + b) + c \neq a + (b + c) \tag{1.35}$$

puede ser verdadero. Proporcione un ejemplo numérico.

Ejercicio 1.7. La serie armónica $S_n = \sum_{k=1}^{n} \frac{1}{k}$ diverge, pero su cálculo en computadora:

1. Implemente el cálculo sumando de $k = 1$ hacia adelante
2. Implemente sumando de $k = n$ hacia atrás
3. Compare los resultados para $n = 10^8$
4. Explique las diferencias observadas

Ejercicio 1.8. Calcule cuántas iteraciones necesita un método con convergencia lineal ($C = 0{,}5$) para reducir el error de 10^{-2} a 10^{-10}. Compare con un método de convergencia cuadrática.

Ejercicio 1.9. El modelo logístico de crecimiento poblacional es:

$$P(t) = \frac{KP_0}{P_0 + (K - P_0)e^{-rt}} \tag{1.36}$$

donde P_0 es la población inicial, K es la capacidad de carga, y r es la tasa de crecimiento.

1. Si $P_0 = 1000$, $K = 10000$, $r = 0{,}5$, calcule $P(10)$
2. ¿Qué métodos numéricos podrían necesitarse si queremos encontrar t tal que $P(t) = 7000$?

Ejercicio 1.10. Considere la integral:

$$I = \int_0^1 \frac{dx}{1 + x^2} \tag{1.37}$$

1. Calcule el valor exacto analíticamente
2. Explique por qué la mayoría de integrales requieren métodos numéricos
3. Proponga dos integrales que no tengan solución analítica elemental

Ejercicio 1.11. En finanzas, el valor presente de una anualidad es:

$$PV = PMT \times \frac{1 - (1 + r)^{-n}}{r} \tag{1.38}$$

Si conocemos PV, PMT, y n, pero necesitamos encontrar r:

1. Explique por qué no se puede despejar r analíticamente
2. ¿Qué tipo de método numérico se necesitaría?
3. Si $PV = 10000$, $PMT = 500$, $n = 24$, estime r gráficamente

Ejercicio 1.12. Demuestre que el error de aproximar $\sin(x)$ cerca de $x = 0$ usando

$$\sin(x) \approx x - \frac{x^3}{6} \tag{1.39}$$

es del orden de $O(x^5)$.

Ejercicio 1.13 (Proyecto de investigación)**.** Elija una de las siguientes áreas y prepare un reporte breve (2-3 páginas) sobre el uso de métodos numéricos:

- Predicción meteorológica
- Diseño de automóviles (aerodinámica)

- Resonancia magnética (MRI)
- Animación por computadora (películas de Pixar/Disney)
- Predicción del mercado de valores
- Exploración petrolera

Incluya:

1. Descripción del problema
2. Qué métodos numéricos se utilizan
3. Por qué son necesarios (por qué no hay solución analítica)
4. Impacto en la industria/sociedad

Resumen del Capítulo 1

Conceptos clave:

- Los métodos numéricos son esenciales para resolver problemas sin solución analítica
- Tienen aplicaciones en prácticamente todas las disciplinas científicas y tecnológicas
- El error es inevitable en computación numérica, pero puede controlarse y minimizarse
- Es fundamental distinguir entre error absoluto y relativo
- Los errores se propagan a través de los cálculos
- El condicionamiento del problema y la estabilidad del algoritmo son cruciales
- La convergencia determina la eficiencia de los métodos iterativos

Fórmulas importantes:

$$\text{Error absoluto:} \quad E_a = |x_{\text{verdadero}} - x_{\text{aproximado}}| \tag{1.40}$$

$$\text{Error relativo:} \quad E_r = \frac{E_a}{|x_{\text{verdadero}}|} \times 100\,\% \tag{1.41}$$

$$\text{Épsilon de máquina:} \quad \epsilon_m \approx 2{,}22 \times 10^{-16} \text{ (precisión doble)} \tag{1.42}$$

Próximo capítulo: Estudiaremos variables y funciones, fundamentales para la formulación de problemas numéricos.

Lecturas recomendadas

1. Burden, R. L., & Faires, J. D. - *Numerical Analysis*, Capítulo 1: Mathematical Preliminaries
2. Chapra, S. C., & Canale, R. P. - *Numerical Methods for Engineers*, Capítulo 3: Approximations and Errors
3. Higham, N. J. - *Accuracy and Stability of Numerical Algorithms*, Capítulo 1: Fundamentals
4. Press, W. H. et al. - *Numerical Recipes*, Capítulo 1: Preliminary

Nota Importante

Para profundizar en los conceptos de este capítulo, se recomienda revisar los recursos en línea de IEEE sobre el estándar de punto flotante (IEEE 754) y experimentar con diferentes lenguajes de programación para observar el comportamiento del error numérico.

Capítulo 2

Variables y Funciones

Objetivos del capítulo:

- Comprender el concepto de variable en contextos numéricos
- Dominar la clasificación y propiedades de funciones matemáticas
- Analizar el dominio y rango de funciones
- Aplicar estos conceptos a problemas de métodos numéricos
- Representar funciones de diversas formas

2.1. Variables

Las variables son la base fundamental sobre la cual se construyen los modelos matemáticos y numéricos. Comprender su naturaleza y clasificación es esencial para formular y resolver problemas.

2.1.1. Definición de variable

Definición

Una **variable** es un símbolo (generalmente una letra) que representa un valor que puede cambiar o variar dentro de un conjunto determinado.

En el contexto de métodos numéricos, las variables representan:

- Cantidades físicas (temperatura, presión, velocidad)
- Parámetros de modelos (coeficientes, constantes)

- Incógnitas a determinar (raíces, soluciones)
- Índices de iteración

2.1.2. Clasificación de variables

Variables independientes y dependientes

Definición

- Una **variable independiente** es aquella cuyos valores pueden elegirse libremente dentro de su dominio.
- Una **variable dependiente** es aquella cuyos valores están determinados por los valores de otras variables (generalmente las independientes).

Ejemplo

En la ecuación de movimiento uniformemente acelerado:

$$y(t) = y_0 + v_0 t + \frac{1}{2} a t^2 \tag{2.1}$$

- t es la variable independiente (tiempo)
- y es la variable dependiente (posición)
- y_0, v_0, a son parámetros (condiciones iniciales y aceleración)

Contexto	Variable Independiente	Variable Dependiente
Física	Tiempo t	Posición $x(t)$
Economía	Precio p	Demanda $D(p)$
Geometría	Radio r	Área $A(r)$
Ingeniería	Voltaje V	Corriente $I(V)$
Finanzas	Tiempo t	Valor $V(t)$

Tabla 2.1: Ejemplos de variables independientes y dependientes

Variables discretas y continuas

Definición

- Una **variable discreta** toma valores de un conjunto contable (generalmente enteros).
- Una **variable continua** puede tomar cualquier valor en un intervalo de números reales.

Ejemplo

Variables discretas:

- Número de iteraciones: $n \in \{1, 2, 3, \ldots\}$
- Número de estudiantes: $N \in \{0, 1, 2, \ldots\}$
- Grado de un polinomio: $k \in \mathbb{N}$

Variables continuas:

- Temperatura: $T \in \mathbb{R}$
- Tiempo: $t \in [0, \infty)$
- Posición: $x \in \mathbb{R}$

Variables escalares y vectoriales

Definición

- Una **variable escalar** se representa mediante un único número real.
- Una **variable vectorial** se representa mediante un conjunto ordenado de números (vector).

Ejemplo

Escalares:

$$x = 5, \quad y = -3{,}7, \quad \theta = \pi/4 \tag{2.2}$$

Vectores:

$$\vec{v} = \begin{bmatrix} 3 \\ -2 \\ 1 \end{bmatrix}, \quad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{2.3}$$

2.1.3. Notación y convenciones

En métodos numéricos, seguimos ciertas convenciones:

Notación	Significado
x, y, z	Variables escalares
$\vec{x}, \vec{v}, \vec{u}$	Vectores
x_i, y_j	Componentes indexadas
$x^{(k)}$ o x_k	Aproximación en iteración k
x^*	Valor óptimo o solución exacta
$\tilde{x}$	Valor aproximado
Δx	Incremento o cambio en x
ϵ	Error o tolerancia

Tabla 2.2: Convenciones de notación

2.1.4. Variables en problemas numéricos

Variables de estado

En sistemas dinámicos, las variables de estado describen completamente el sistema en un momento dado.

Péndulo simple

El estado del péndulo se describe con dos variables:

$$\vec{x}(t) = \begin{bmatrix} \theta(t) \\ \omega(t) \end{bmatrix} \tag{2.4}$$

donde θ es el ángulo y ω es la velocidad angular.

Variables de control

Las variables de control son aquellas que podemos manipular para influir en el comportamiento del sistema.

Control de temperatura

En un horno:

- Variable de control: Potencia del calentador $P(t)$
- Variable de estado: Temperatura $T(t)$
- Relación: $\frac{dT}{dt} = f(T, P)$

2.1.5. Dominio de una variable

Definición

El **dominio** de una variable es el conjunto de todos los valores posibles que puede tomar.

Ejemplo

- Ángulo: $\theta \in [0, 2\pi)$ radianes
- Probabilidad: $p \in [0, 1]$
- Temperatura Kelvin: $T \in [0, \infty)$
- Número de iteraciones: $n \in \mathbb{N} = \{1, 2, 3, \ldots\}$
- Coordenada: $x \in \mathbb{R}$

Nota Importante

En métodos numéricos, el dominio efectivo de una variable puede estar restringido por:

- Limitaciones físicas del problema
- Rango de precisión de la computadora
- Región de convergencia del método
- Condiciones de frontera

2.2. Funciones matemáticas

Las funciones son el lenguaje fundamental para expresar relaciones entre variables. En métodos numéricos, trabajamos con funciones de diversos tipos y complejidades.

2.2.1. Definición formal de función

Definición

Una **función** f de un conjunto A a un conjunto B es una regla que asigna a cada elemento $x \in A$ exactamente un elemento $y \in B$. Se denota:

$$f : A \to B, \quad y = f(x) \tag{2.5}$$

Donde:

- A es el **dominio** de f: $\text{Dom}(f) = A$
- B es el **codominio** de f
- y es la **imagen** de x bajo f
- El conjunto de todas las imágenes es el **rango** o **imagen**: $\text{Rg}(f) = \{f(x) : x \in A\}$

Ejemplo

La función $f(x) = x^2$ con dominio $\mathbb{R}$:

- Dominio: $\text{Dom}(f) = \mathbb{R} = (-\infty, \infty)$
- Codominio: $\mathbb{R}$
- Rango: $\text{Rg}(f) = [0, \infty)$

Note que el rango es un subconjunto del codominio.

2.2.2. Representación de funciones

Las funciones pueden representarse de múltiples formas:

1. Representación analítica (fórmula)

$$f(x) = x^2 - 3x + 2 \tag{2.6}$$

2. Representación gráfica

$$xf(x)^2 - 3 * x + 2; ^2 - 3x + 2$$

x	$f(x)$
0	2
1	0
2	0
3	2
4	6

Tabla 2.3: Representación tabular de $f(x) = x^2 - 3x + 2$

3. Representación tabular

4. Representación algorítmica

Listing 2.1: Función como algoritmo

```
def f(x):
    if x < 0:
        return -x
    elif x < 1:
        return x**2
    else:
        return 2*x - 1
```

2.2.3. Clasificación de funciones

Funciones algebraicas

Son funciones que pueden expresarse mediante operaciones algebraicas (suma, resta, multiplicación, división, potencias y raíces).

Funciones polinomiales

Definición

Una **función polinomial** de grado n tiene la forma:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

donde $a_n \neq 0$ y $n \in \mathbb{N} \cup \{0\}$.

Ejemplo

$$P_1(x) = 3x - 5 \quad \text{(lineal, grado 1)} \tag{2.8}$$
$$P_2(x) = 2x^2 - x + 3 \quad \text{(cuadrática, grado 2)} \tag{2.9}$$
$$P_3(x) = x^3 - 4x + 1 \quad \text{(cúbica, grado 3)} \tag{2.10}$$

Teorema

Todo polinomio de grado $n \geq 1$ con coeficientes complejos tiene exactamente n raíces (contando multiplicidades) en el campo complejo.

Definición

Una **función racional** es el cociente de dos polinomios:

$$R(x) = \frac{P(x)}{Q(x)}$$

donde $P(x)$ y $Q(x)$ son polinomios y $Q(x) \neq 0$.

Funciones racionales

Ejemplo

$$f(x) = \frac{x^2 + 1}{x^2 - 4} \tag{2.12}$$

Esta función tiene:

- Dominio: $\mathbb{R} \setminus \{-2, 2\}$ (excluye donde $Q(x) = 0$)
- Asíntotas verticales en $x = -2$ y $x = 2$
- Asíntota horizontal en $y = 1$ (cuando $x \to \pm\infty$)

Funciones radicales Involucran raíces de expresiones algebraicas:

$$f(x) = \sqrt{x^2 - 4}, \quad g(x) = \sqrt[3]{x + 1} \tag{2.13}$$

Funciones trascendentes

Son funciones que no son algebraicas.

Definición

Una **función exponencial** tiene la forma:

$$f(x) = a^x$$

donde $a > 0$ y $a \neq 1$.

Funciones exponenciales

La más importante es la función exponencial natural:

$$f(x) = e^x, \quad \text{donde } e = \lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n \approx 2{,}71828 \tag{2.15}$$

Crecimiento exponencial

El crecimiento poblacional se modela como:

$$P(t) = P_0 e^{rt} \tag{2.16}$$

donde P_0 es la población inicial y r es la tasa de crecimiento.

Definición

La **función logarítmica** es la inversa de la exponencial:

$$y = \log_a(x) \iff a^y = x$$

Funciones logarítmicas

El logaritmo natural (base e) es:

$$\ln(x) = \log_e(x) \tag{2.18}$$

Propiedades importantes:

$$\ln(xy) = \ln(x) + \ln(y) \tag{2.19}$$
$$\ln(x/y) = \ln(x) - \ln(y) \tag{2.20}$$
$$\ln(x^p) = p\ln(x) \tag{2.21}$$
$$\ln(e) = 1, \quad \ln(1) = 0 \tag{2.22}$$

Funciones trigonométricas Las seis funciones trigonométricas básicas son:

Función	Definición	Dominio
$\sin(x)$	Seno	$\mathbb{R}$
$\cos(x)$	Coseno	$\mathbb{R}$
$\tan(x)$	Tangente	$\mathbb{R} \setminus \{(2k+1)\pi/2 : k \in \mathbb{Z}\}$
$\cot(x)$	Cotangente	$\mathbb{R} \setminus \{k\pi : k \in \mathbb{Z}\}$
$\sec(x)$	Secante	$\mathbb{R} \setminus \{(2k+1)\pi/2 : k \in \mathbb{Z}\}$
$\csc(x)$	Cosecante	$\mathbb{R} \setminus \{k\pi : k \in \mathbb{Z}\}$

Tabla 2.4: Funciones trigonométricas

Identidades fundamentales:

$$\sin^2(x) + \cos^2(x) = 1 \tag{2.23}$$

$$\tan(x) = \frac{\sin(x)}{\cos(x)} \tag{2.24}$$

$$\sin(2x) = 2\sin(x)\cos(x) \tag{2.25}$$

$$\cos(2x) = \cos^2(x) - \sin^2(x) \tag{2.26}$$

Definición

Las **funciones hiperbólicas** se definen como:

$$\sinh(x) = \frac{e^x - e^{-x}}{2}$$

$$\cosh(x) = \frac{e^x + e^{-x}}{2}$$

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Funciones hiperbólicas

Nota Importante

Las funciones hiperbólicas aparecen naturalmente en:

- La forma de cables colgantes (catenaria): $y = a\cosh(x/a)$
- Ondas en fluidos
- Teoría de la relatividad especial

Funciones por partes

Definición

Una **función por partes** se define mediante diferentes expresiones en diferentes partes de su dominio.

Ejemplo

La función valor absoluto:

$$|x| = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{si } x < 0 \end{cases} \tag{2.30}$$

La función escalón unitario (Heaviside):

$$H(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases} \tag{2.31}$$

2.2.4. Propiedades de funciones

Inyectividad, sobreyectividad y biyectividad

Definición

Sea $f : A \to B$:

- f es **inyectiva** (uno a uno) si $f(x_1) = f(x_2) \implies x_1 = x_2$
- f es **sobreyectiva** (sobre) si para todo $y \in B$ existe $x \in A$ tal que $f(x) = y$
- f es **biyectiva** si es inyectiva y sobreyectiva

Ejemplo

- $f(x) = x^2$ en $\mathbb{R}$: No es inyectiva ($f(-2) = f(2) = 4$)
- $f(x) = x^2$ en $[0, \infty)$: Es inyectiva
- $f(x) = e^x$: Es inyectiva
- $f(x) = x^3$: Es biyectiva de $\mathbb{R}$ a $\mathbb{R}$

Monotonía

Definición

Una función f es:

- **Creciente** en un intervalo si $x_1 < x_2 \implies f(x_1) \leq f(x_2)$
- **Estrictamente creciente** si $x_1 < x_2 \implies f(x_1) < f(x_2)$
- **Decreciente** en un intervalo si $x_1 < x_2 \implies f(x_1) \geq f(x_2)$
- **Estrictamente decreciente** si $x_1 < x_2 \implies f(x_1) > f(x_2)$

Paridad

Definición

Una función f es:

- **Par** si $f(-x) = f(x)$ para todo x (simétrica respecto al eje y)
- **Impar** si $f(-x) = -f(x)$ para todo x (simétrica respecto al origen)

Ejemplo

- $f(x) = x^2$, $\cos(x)$: funciones pares
- $f(x) = x^3$, $\sin(x)$: funciones impares
- $f(x) = x^2 + x$: ni par ni impar

Periodicidad

Definición

Una función f es **periódica** con período $T > 0$ si:

$$f(x + T) = f(x) \tag{2.32}$$

para todo x en el dominio.

Ejemplo

- $\sin(x)$, $\cos(x)$: período 2π
- $\tan(x)$: período π
- $\sin(2\pi x)$: período 1

Continuidad

Definición

Una función f es **continua** en $x = a$ si:

$$\lim_{x \to a} f(x) = f(a) \tag{2.33}$$

Es decir, si:

1. $f(a)$ está definida
2. $\lim_{x \to a} f(x)$ existe
3. El límite es igual al valor de la función

Nota Importante

En métodos numéricos, la continuidad es crucial para:

- Garantizar la existencia de raíces (Teorema del valor intermedio)
- Asegurar convergencia de métodos iterativos
- Aplicar métodos de integración y diferenciación

Diferenciabilidad

Definición

Una función f es **diferenciable** en $x = a$ si existe:

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h} \tag{2.34}$$

Teorema 2.1. *Si f es diferenciable en a, entonces f es continua en a.*
(El recíproco no es cierto: $f(x) = |x|$ es continua en $x = 0$ pero no diferenciable)

2.2.5. Operaciones con funciones

Operaciones aritméticas

Dadas $f, g : \mathbb{R} \to \mathbb{R}$:

$$(f+g)(x) = f(x) + g(x) \tag{2.35}$$
$$(f-g)(x) = f(x) - g(x) \tag{2.36}$$
$$(f \cdot g)(x) = f(x) \cdot g(x) \tag{2.37}$$
$$\left(\frac{f}{g}\right)(x) = \frac{f(x)}{g(x)}, \quad g(x) \neq 0 \tag{2.38}$$

Composición

Definición

La **composición** de f con g es:

$$(f \circ g)(x) = f(g(x)) \tag{2.39}$$

Ejemplo

Si $f(x) = x^2$ y $g(x) = \sin(x)$:

$$(f \circ g)(x) = f(g(x)) = f(\sin(x)) = \sin^2(x) \tag{2.40}$$
$$(g \circ f)(x) = g(f(x)) = g(x^2) = \sin(x^2) \tag{2.41}$$

Note que en general $(f \circ g) \neq (g \circ f)$.

Función inversa

Definición

Si $f : A \to B$ es biyectiva, su **función inversa** $f^{-1} : B \to A$ satisface:

$$f^{-1}(f(x)) = x \quad \text{y} \quad f(f^{-1}(y)) = y \tag{2.42}$$

Ejemplo

- $f(x) = e^x$ tiene inversa $f^{-1}(x) = \ln(x)$
- $f(x) = x^3$ tiene inversa $f^{-1}(x) = \sqrt[3]{x}$
- $f(x) = 2x + 3$ tiene inversa $f^{-1}(x) = \frac{x-3}{2}$

2.2.6. Funciones importantes en métodos numéricos

Funciones de base (basis functions)

En aproximación de funciones, usamos funciones de base:

- **Monomios**: $1, x, x^2, x^3, \ldots$
- **Polinomios de Chebyshev**: $T_n(x) = \cos(n \arccos(x))$
- **Polinomios de Legendre**: $P_n(x)$
- **Funciones trigonométricas**: $\sin(nx), \cos(nx)$

Splines

Funciones por partes diferenciables usadas en interpolación:

$$S(x) = \begin{cases} S_1(x) & x \in [x_0, x_1] \\ S_2(x) & x \in [x_1, x_2] \\ \vdots & \vdots \\ S_n(x) & x \in [x_{n-1}, x_n] \end{cases} \tag{2.43}$$

donde cada S_i es un polinomio.

2.3. Dominio y rango

El dominio y rango de una función son conceptos fundamentales para entender su comportamiento y aplicabilidad en problemas numéricos.

2.3.1. Determinación del dominio

Método general

Para determinar el dominio de $f(x)$, identificamos valores donde:

1. **División por cero**: Si $f(x) = \frac{g(x)}{h(x)}$, excluimos donde $h(x) = 0$
2. **Raíces pares negativas**: Si $f(x) = \sqrt[2n]{g(x)}$, requerimos $g(x) \geq 0$
3. **Logaritmos**: Si $f(x) = \log(g(x))$, requerimos $g(x) > 0$
4. **Restricciones del contexto**: Restricciones físicas o del problema

Determinación de dominio

Ejemplo 1: $f(x) = \frac{1}{x^2-4}$
Solución:

1. Encontramos donde el denominador es cero: $x^2 - 4 = 0 \implies x = \pm 2$
2. Dominio: $\text{Dom}(f) = \mathbb{R} \setminus \{-2, 2\} = (-\infty, -2) \cup (-2, 2) \cup (2, \infty)$

Ejemplo 2: $g(x) = \sqrt{x^2 - 9}$
Solución:

1. Requerimos $x^2 - 9 \geq 0$
2. $x^2 \geq 9 \implies |x| \geq 3$
3. Dominio: $\text{Dom}(g) = (-\infty, -3] \cup [3, \infty)$

Ejemplo 3: $h(x) = \ln(x - 2) + \frac{1}{x+1}$
Solución:

1. Para $\ln(x - 2)$: necesitamos $x - 2 > 0 \implies x > 2$
2. Para $\frac{1}{x+1}$: necesitamos $x \neq -1$
3. Intersección: $\text{Dom}(h) = (2, \infty)$

2.3.2. Determinación del rango

El rango es el conjunto de todos los valores que la función puede tomar.

Métodos para encontrar el rango

Método 1: Análisis algebraico Despejamos x en términos de $y = f(x)$ y encontramos qué valores de y son posibles.

Ejemplo

Para $f(x) = \frac{2x-1}{x+3}$, con $x \neq -3$:

$$y = \frac{2x-1}{x+3} \tag{2.44}$$
$$y(x+3) = 2x - 1 \tag{2.45}$$
$$yx + 3y = 2x - 1 \tag{2.46}$$
$$yx - 2x = -1 - 3y \tag{2.47}$$
$$x(y-2) = -1 - 3y \tag{2.48}$$
$$x = \frac{-1-3y}{y-2} \tag{2.49}$$

Esto es válido si $y \neq 2$. Por tanto: $\text{Rg}(f) = \mathbb{R} \setminus \{2\}$

Método 2: Análisis usando cálculo Usamos derivadas para encontrar valores máximos y mínimos.

Ejemplo

Para $f(x) = x^2 - 4x + 7$ en $\mathbb{R}$:

$$f'(x) = 2x - 4 = 0 \implies x = 2 \tag{2.50}$$
$$f''(x) = 2 > 0 \implies \text{es un mínimo} \tag{2.51}$$
$$f(2) = 4 - 8 + 7 = 3 \tag{2.52}$$

Como es una parábola que abre hacia arriba: $\text{Rg}(f) = [3, \infty)$

Método 3: Análisis gráfico Graficamos la función y observamos qué valores alcanza en el eje y.

2.3.3. Ejemplos detallados

Función racional compleja

Considere $f(x) = \frac{x^2-1}{x^2+1}$

Dominio:

- El denominador $x^2 + 1 > 0$ para todo $x \in \mathbb{R}$
- $\text{Dom}(f) = \mathbb{R}$

Rango:

$$y = \frac{x^2 - 1}{x^2 + 1} \tag{2.53}$$

$$y(x^2 + 1) = x^2 - 1 \tag{2.54}$$

$$yx^2 + y = x^2 - 1 \tag{2.55}$$

$$x^2(y - 1) = -1 - y \tag{2.56}$$

$$x^2 = \frac{-1 - y}{y - 1} = \frac{1 + y}{1 - y} \tag{2.57}$$

Para que x^2 sea real y no negativo:

$$\frac{1 + y}{1 - y} \geq 0 \tag{2.58}$$

Analizando signos: $-1 \leq y < 1$
Por tanto: $\text{Rg}(f) = [-1, 1)$

Función trigonométrica compuesta

Sea $g(x) = 2\sin(x) + 3$

Dominio:

- $\sin(x)$ está definida para todo x
- $\text{Dom}(g) = \mathbb{R}$

Rango:

- Sabemos que $-1 \leq \sin(x) \leq 1$
- Multiplicando por 2: $-2 \leq 2\sin(x) \leq 2$
- Sumando 3: $1 \leq 2\sin(x) + 3 \leq 5$
- $\text{Rg}(g) = [1, 5]$

2.3.4. Dominio y rango en problemas aplicados

Problema de optimización

Una empresa produce x unidades de un producto. El costo total es:

$$C(x) = 100 + 50x + 0{,}1x^2 \tag{2.59}$$

y el ingreso por ventas es:

$$I(x) = 200x \tag{2.60}$$

La función de utilidad (ganancia) es:

$$U(x) = I(x) - C(x) = 200x - 100 - 50x - 0{,}1x^2 = -0{,}1x^2 + 150x - 100 \tag{2.61}$$

Dominio práctico:

- Matemáticamente: $x \in \mathbb{R}$
- Físicamente: $x \geq 0$ (no se pueden producir cantidades negativas)
- Capacidad de producción: $x \leq 1000$ (supongamos)
- Dominio práctico: $[0, 1000]$

Rango:

$$U'(x) = -0{,}2x + 150 = 0 \implies x = 750 \tag{2.62}$$

$$U(750) = -0{,}1(750)^2 + 150(750) - 100 = 56,150 \tag{2.63}$$

En $x = 0$: $U(0) = -100$ (pérdida inicial) En $x = 1000$: $U(1000) = 49,900$
$\text{Rg}(U) = [-100, 56150]$ en el dominio práctico.

2.3.5. Restricción de dominio

En métodos numéricos, frecuentemente restringimos el dominio:

Restricción para búsqueda de raíces

Para encontrar raíces de $f(x) = x^3 - 2x - 5$:

- Dominio natural: $\mathbb{R}$
- Análisis: $f(2) = -1 < 0$, $f(3) = 16 > 0$
- Teorema del valor intermedio: existe raíz en $(2, 3)$
- Dominio restringido para búsqueda: $[2, 3]$

2.3.6. Tabla de dominios y rangos de funciones comunes

Función	Dominio	Rango
$f(x) = x^n$ (n par)	$\mathbb{R}$	$[0, \infty)$
$f(x) = x^n$ (n impar)	$\mathbb{R}$	$\mathbb{R}$
$f(x) = \sqrt{x}$	$[0, \infty)$	$[0, \infty)$
$f(x) = \frac{1}{x}$	$\mathbb{R} \setminus \{0\}$	$\mathbb{R} \setminus \{0\}$
$f(x) = e^x$	$\mathbb{R}$	$(0, \infty)$
$f(x) = \ln(x)$	$(0, \infty)$	$\mathbb{R}$
$f(x) = \sin(x)$	$\mathbb{R}$	$[-1, 1]$
$f(x) = \cos(x)$	$\mathbb{R}$	$[-1, 1]$
$f(x) = \tan(x)$	$\mathbb{R} \setminus \{(2k+1)\frac{\pi}{2}\}$	$\mathbb{R}$
$f(x) = \arcsin(x)$	$[-1, 1]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$
$f(x) = \arccos(x)$	$[-1, 1]$	$[0, \pi]$
$f(x) = \arctan(x)$	$\mathbb{R}$	$(-\frac{\pi}{2}, \frac{\pi}{2})$

Tabla 2.5: Dominios y rangos de funciones comunes

2.4. Ejercicios propuestos

2.4.1. Implementaciones computacionales

Antes de los ejercicios teóricos, presentamos implementaciones prácticas en Python y R para trabajar con funciones.

Implementaciones en Python

Listing 2.2: Definición y evaluación de funciones en Python

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

```

```
# Definir funciones de diferentes formas
# Forma 1: Funcion lambda (funciones simples)
f1 = lambda x: x**2 - 3*x + 2

# Forma 2: Funcion tradicional (funciones complejas)
def f2(x):
    """Funcion polinomial cubica"""
    return x**3 - 2*x**2 + x - 1

# Forma 3: Funcion por partes
def f3(x):
    """Funcion por partes"""
    if x < 0:
        return -x
    elif x < 1:
        return x**2
    else:
        return 2*x - 1

# Forma 4: Funcion vectorizada (para arrays de NumPy)
def f4(x):
    """Funcion racional vectorizada"""
    return (x**2 - 1) / (x**2 + 1)

# Evaluar funciones
x_val = 2.5
print(f"f1({x_val}) = {f1(x_val)}")
print(f"f2({x_val}) = {f2(x_val)}")
print(f"f3({x_val}) = {f3(x_val)}")

# Evaluar en multiples puntos
x_array = np.linspace(-5, 5, 100)
y_array = f4(x_array)

# Graficar
plt.figure(figsize=(10, 6))
plt.plot(x_array, y_array, 'b-', linewidth=2, label='$f(x) = \\frac
   {x^2-1}{x^2+1}
Clasifique las siguientes variables como independientes o
   dependientes, y como discretas o continuas:
\begin{enumerate}
```

```
    \item La temperatura de un horno en funci n del tiempo
    \item El n mero de bacterias en un cultivo despu s de $n$
       horas
    \item La posici n de un p ndulo en funci n del tiempo
    \item El costo total de producci n en funci n del n mero de
       unidades
    \item La concentraci n de un medicamento en la sangre en
       funci n del tiempo
\end{enumerate}
\end{exercise}

\begin{exercise}
Para cada funci n , determine su dominio y rango:
\begin{enumerate}
    \item $f(x) = \sqrt{4 - x^2}$
    \item $g(x) = \frac{x}{x^2 - 1}$
    \item $h(x) = \ln(x^2 - 4)$
    \item $k(x) = e^{-x^2}$
    \item $m(x) = \frac{1}{\sqrt{x + 2}}$
    \item $n(x) = \arcsin(2x - 1)$
\end{enumerate}
\end{exercise}

\begin{exercise}
Dadas $f(x) = x^2 + 1$ y $g(x) = \sqrt{x}$:
\begin{enumerate}
    \item Encuentre $(f \circ g)(x)$ y su dominio
    \item Encuentre $(g \circ f)(x)$ y su dominio
    \item  Son  iguales? Explique
    \item Calcule $(f \circ g)(4)$ y $(g \circ f)(4)$
\end{enumerate}
\end{exercise}

\begin{exercise}
Determine si las siguientes funciones son pares, impares o ninguna:
\begin{enumerate}
    \item $f(x) = x^4 - 2x^2 + 1$
    \item $g(x) = x^3 - x$
    \item $h(x) = \frac{x}{x^2 + 1}$
    \item $k(x) = |x| + x$
    \item $m(x) = \sin(x) \cos(x)$
```

```
\end{enumerate}
\end{exercise}

\begin{exercise}
Para la funci n $f(x) = |x^2 - 4|$:
\begin{enumerate}
    \item Escriba la funci n por partes (sin usar valor absoluto)
    \item Determine su dominio y rango
    \item Grafique la funci n
    \item Identifique los puntos donde no es diferenciable
\end{enumerate}
\end{exercise}

\begin{exercise}
Una part cula se mueve seg n la ecuaci n:
\begin{equation}
    s(t) = t^3 - 6t^2 + 9t + 1
\end{equation}
donde $s$ es la posici n en metros y $t$ es el tiempo en segundos.
\begin{enumerate}
    \item Cul es la variable independiente y cu l la
       dependiente?
    \item Encuentre la velocidad $v(t) = s'(t)$
    \item En qu instantes la part cula est en reposo?
    \item Cul es el dominio pr ctico si el movimiento dura 5
       segundos?
\end{enumerate}
\end{exercise}

\begin{exercise}
Determine la funci n inversa de cada una y verifique su respuesta:
\begin{enumerate}
    \item $f(x) = 3x - 7$
    \item $g(x) = \frac{x + 1}{x - 2}$, $x \neq 2$
    \item $h(x) = \sqrt{x + 4}$, $x \geq -4$
    \item $k(x) = 2^{x+1}$
\end{enumerate}
\end{exercise}

\begin{exercise}
Una funci n de demanda est dada por:
```

```
\begin{equation}
    D(p) = \frac{1000}{p + 5}
\end{equation}
donde $p$ es el precio en d lares y $D$ es la cantidad demandada.
\begin{enumerate}
    \item   Cul   es el dominio pr ctico?
    \item   Cul   es el rango pr ctico?
    \item Si el precio aumenta,   qu   pasa con la demanda?
    \item Encuentre la funci n inversa $p(D)$ (precio como
       funci n de demanda)
\end{enumerate}
\end{exercise}

\begin{exercise}
Demuestre que:
\begin{enumerate}
    \item La suma de dos funciones pares es par
    \item El producto de dos funciones impares es par
    \item La composici n de dos funciones inyectivas es inyectiva
    \item Si $f$ es creciente y $g$ es decreciente, entonces $f \
       circ g$ es decreciente
\end{enumerate}
\end{exercise}

\begin{exercise}
Para la funci n log stica:
\begin{equation}
    P(t) = \frac{K}{1 + Ae^{-rt}}
\end{equation}
donde $K$, $A$, $r$ son constantes positivas:
\begin{enumerate}
    \item Determine el dominio y rango
    \item Calcule $\lim_{t \to \infty} P(t)$
    \item   Qu   representa $K$ en el contexto del crecimiento
       poblacional?
    \item Grafique cualitativamente la funci n
\end{enumerate}
\end{exercise}

\begin{exercise}
Considere la funci n:
```

```
\begin{equation}
    f(x) = \begin{cases}
        x^2 & \text{si } x < 1\\
        2x - 1 & \text{si } 1 \leq x < 3\\
        \sqrt{x + 6} & \text{si } x \geq 3
    \end{cases}
\end{equation}
\begin{enumerate}
    \item Determine si $f$ es continua en $x = 1$ y $x = 3$
    \item Encuentre $\lim_{x \to 1^-} f(x)$ y $\lim_{x \to 1^+} f(x)$
    \item Calcule $f(0)$, $f(1)$, $f(2)$, $f(3)$, $f(4)$
    \item Grafique la funci n
\end{enumerate}
\end{exercise}

\begin{exercise}
En un circuito el ctrico, la impedancia $Z$ en funci n de la frecuencia $\omega$ es:
\begin{equation}
    Z(\omega) = \sqrt{R^2 + \left(\omega L - \frac{1}{\omega C}\right)^2}
\end{equation}
donde $R$, $L$, $C$ son constantes positivas.
\begin{enumerate}
    \item Cul es el dominio natural de esta funci n?
    \item Cul es el dominio pr ctico (frecuencias f sicamente realizables)?
    \item Existe un valor m nimo de $Z$? Si es as , en qu frecuencia ocurre?
\end{enumerate}
\end{exercise}

\begin{exercise}[Proyecto computacional]
Implemente en Python o R las siguientes funciones y realice un an lisis completo:

\textbf{Funciones a implementar:}
\begin{enumerate}
    \item $f(x) = \frac{\sin(x)}{x}$ (funci n sinc)
    \item $g(x) = e^{-x^2}$ (funci n gaussiana)
```

```
    \item $h(x) = \frac{1}{1 + x^2}$ (funci n lorentziana)
\end{enumerate}

\textbf{Para cada funci n:}
\begin{itemize}
    \item Grafique en el intervalo $[-10, 10]$
    \item Determine num ricamente el dominio y rango
    \item Encuentre m ximos y m nimos (use derivadas num ricas o
        m todos de optimizaci n)
    \item Calcule el  rea  bajo la curva
    \item Verifique si es par, impar o ninguna
\end{itemize}

\textbf{C digo de inicio (Python):}

\begin{lstlisting}[language=Python]
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate, optimize

# Funcion sinc con manejo del caso x=0
def sinc_func(x):
    return np.where(x == 0, 1.0, np.sin(x)/x)

def gaussian(x):
    return np.exp(-x**2)

def lorentzian(x):
    return 1 / (1 + x**2)

# Graficar las tres funciones
x = np.linspace(-10, 10, 1000)
plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)
plt.plot(x, sinc_func(x), 'b-', linewidth=2)
plt.title('Funcion sinc')
plt.grid(True, alpha=0.3)
plt.xlabel('x')
plt.ylabel('f(x)')

```

```
plt.subplot(1, 3, 2)
plt.plot(x, gaussian(x), 'r-', linewidth=2)
plt.title('Funcion gaussiana')
plt.grid(True, alpha=0.3)
plt.xlabel('x')
plt.ylabel('g(x)')

plt.subplot(1, 3, 3)
plt.plot(x, lorentzian(x), 'g-', linewidth=2)
plt.title('Funcion lorentziana')
plt.grid(True, alpha=0.3)
plt.xlabel('x')
plt.ylabel('h(x)')

plt.tight_layout()
plt.show()

# Encontrar maximo de la gaussiana
max_result = optimize.minimize_scalar(lambda x: -gaussian(x))
print(f"Maximo de gaussiana en x = {max_result.x:.4f}")

# Calcular area bajo la curva gaussiana
area, error = integrate.quad(gaussian, -10, 10)
print(f"Area bajo gaussiana: {area:.4f}")
```

Código de inicio (R):

```
# Definir funciones
sinc_func <- function(x) {
  ifelse(x == 0, 1, sin(x)/x)
}

gaussian <- function(x) {
  exp(-x^2)
}

lorentzian <- function(x) {
  1 / (1 + x^2)
}

# Graficar
x <- seq(-10, 10, length.out = 1000)

```

```r
par(mfrow = c(1, 3))

plot(x, sinc_func(x), type = "l", col = "blue", lwd = 2,
     main = "Funcion sinc", xlab = "x", ylab = "f(x)")
grid()

plot(x, gaussian(x), type = "l", col = "red", lwd = 2,
     main = "Funcion gaussiana", xlab = "x", ylab = "g(x)")
grid()

plot(x, lorentzian(x), type = "l", col = "green", lwd = 2,
     main = "Funcion lorentziana", xlab = "x", ylab = "h(x)")
grid()

# Encontrar maximo
max_result <- optimize(function(x) -gaussian(x),
                       interval = c(-10, 10))
cat(sprintf("Maximo de gaussiana en x = %.4f\n", max_result$minimum
   ))

# Calcular area bajo la curva
area <- integrate(gaussian, -10, 10)
cat(sprintf("Area bajo gaussiana: %.4f\n", area$value))
```

Resumen del Capítulo 2

Conceptos clave:

- Las variables pueden ser independientes/dependientes, discretas/continuas, escalares/vectoriales
- Las funciones establecen relaciones entre variables y pueden representarse de múltiples formas
- Las funciones se clasifican en algebraicas (polinomiales, racionales, radicales) y trascendentes (exponenciales, logarítmicas, trigonométricas)
- El dominio es el conjunto de entradas válidas; el rango es el conjunto de salidas posibles
- Propiedades como inyectividad, monotonía, paridad y continuidad son cruciales en análisis numérico
- La composición e inversión de funciones son operaciones fundamentales

Habilidades desarrolladas:

- Identificar y clasificar variables en problemas
- Determinar dominio y rango de funciones
- Operar con funciones (composición, inversión)
- Analizar propiedades de funciones
- Representar funciones de múltiples formas

Próximo capítulo: Estudiaremos restricciones matemáticas y cómo modelar problemas con limitaciones.

Lecturas recomendadas

1. Stewart, J. - *Cálculo*, Capítulos 1-2: Funciones y límites
2. Burden, R. L., & Faires, J. D. - *Numerical Analysis*, Capítulo 1.2: Mathematical Preliminaries and Error Analysis
3. Apostol, T. M. - *Calculus*, Vol. 1, Capítulo 1: The Concepts of the Integral Calculus
4. Strang, G. - *Introduction to Applied Mathematics*, Capítulo 1: Functions and Linear Algebra

plt.grid(True, alpha=0.3) plt.xlabel('x', fontsize=12) plt.ylabel('f(x)', fontsize=12) plt.title('Grafica de funcion racional', fontsize=14) plt.legend(fontsize=11) plt.axhline(y=0, color='k', linewidth=0.5) plt.axvline(x=0, color='k', linewidth=0.5) plt.show()

Listing 2.3: Determinación de dominio y rango numérico

```
import numpy as np

def encontrar_dominio_numerico(f, x_min=-100, x_max=100, n_puntos
    =10000):
    """
    Encuentra numericamente el dominio de una funcion
    probando valores en un rango dado.
    """
    x_test = np.linspace(x_min, x_max, n_puntos)
    dominio = []

    for x in x_test:
        try:
            y = f(x)
            if np.isfinite(y):  # Verifica que no sea inf o nan
                dominio.append(x)
        except:
            pass

    if dominio:
        return min(dominio), max(dominio)
    return None, None

def encontrar_rango_numerico(f, x_min, x_max, n_puntos=10000):
    """
    Encuentra numericamente el rango de una funcion
    en un dominio dado.
    """
    x_vals = np.linspace(x_min, x_max, n_puntos)
    y_vals = []

    for x in x_vals:
        try:
            y = f(x)
            if np.isfinite(y):
                y_vals.append(y)
        except:
```

```
            pass

    if y_vals:
        return min(y_vals), max(y_vals)
    return None, None

# Ejemplo de uso
f = lambda x: np.sqrt(4 - x**2)  # Semicirculo

dom_min, dom_max = encontrar_dominio_numerico(f, -10, 10)
print(f"Dominio numerico: [{dom_min:.2f}, {dom_max:.2f}]")

rng_min, rng_max = encontrar_rango_numerico(f, dom_min, dom_max)
print(f"Rango numerico: [{rng_min:.2f}, {rng_max:.2f}]")
```

Listing 2.4: Composición e inversión de funciones

```
import numpy as np
from scipy.optimize import fsolve

# Composicion de funciones
def componer(f, g):
    """Retorna la composicion f(g(x))"""
    return lambda x: f(g(x))

# Ejemplo
f = lambda x: x**2
g = lambda x: np.sin(x)
fog = componer(f, g)  # f(g(x)) = sin^2(x)
gof = componer(g, f)  # g(f(x)) = sin(x^2)

x = np.pi/4
print(f"f(g({x:.3f})) = {fog(x):.4f}")
print(f"g(f({x:.3f})) = {gof(x):.4f}")

# Encontrar funcion inversa numericamente
def encontrar_inversa_numerica(f, y_objetivo, x_inicial=0):
    """
    Encuentra x tal que f(x) = y_objetivo
    usando metodo numerico.
    """
    ecuacion = lambda x: f(x) - y_objetivo
```

```
    x_solucion = fsolve(ecuacion, x_inicial)[0]
    return x_solucion

# Ejemplo: inversa de f(x) = 2x + 3
f = lambda x: 2*x + 3
y = 10
x_inv = encontrar_inversa_numerica(f, y)
print(f"Si f(x) = {y}, entonces x = {x_inv}")
print(f"Verificacion: f({x_inv}) = {f(x_inv)}")
```

Listing 2.5: Análisis de propiedades de funciones

```
import numpy as np

def es_par(f, x_test=None, tol=1e-6):
    """Verifica si f(x) = f(-x) (funcion par)"""
    if x_test is None:
        x_test = np.linspace(-10, 10, 100)

    for x in x_test:
        if x != 0 and abs(f(x) - f(-x)) > tol:
            return False
    return True

def es_impar(f, x_test=None, tol=1e-6):
    """Verifica si f(-x) = -f(x) (funcion impar)"""
    if x_test is None:
        x_test = np.linspace(-10, 10, 100)

    for x in x_test:
        if x != 0 and abs(f(-x) + f(x)) > tol:
            return False
    return True

def es_creciente(f, a, b, n_puntos=100):
    """Verifica si f es creciente en [a, b]"""
    x = np.linspace(a, b, n_puntos)
    y = np.array([f(xi) for xi in x])
    return np.all(np.diff(y) >= 0)

# Ejemplos
f_par = lambda x: x**2
```

```
f_impar = lambda x: x**3
f_creciente = lambda x: np.exp(x)

print(f"x^2 es par: {es_par(f_par)}")
print(f"x^3 es impar: {es_impar(f_impar)}")
print(f"e^x es creciente en [0,5]: {es_creciente(f_creciente, 0, 5)
    }")
```

Implementaciones en R

Listing 2.6: Definición y evaluación de funciones en R

```
# Definir funciones de diferentes formas

# Forma 1: Funcion anonima
f1 <- function(x) x^2 - 3*x + 2

# Forma 2: Funcion con documentacion
f2 <- function(x) {
  # Funcion polinomial cubica
  # Args:
  #   x: valor numerico o vector
  # Returns:
  #   valor de la funcion
  return(x^3 - 2*x^2 + x - 1)
}

# Forma 3: Funcion por partes
f3 <- function(x) {
  if (x < 0) {
    return(-x)
  } else if (x < 1) {
    return(x^2)
  } else {
    return(2*x - 1)
  }
}

# Forma 4: Funcion vectorizada
f4 <- function(x) {
  (x^2 - 1) / (x^2 + 1)
}
```

```r

# Evaluar funciones
x_val <- 2.5
cat(sprintf("f1(%.1f) = %.2f\n", x_val, f1(x_val)))
cat(sprintf("f2(%.1f) = %.2f\n", x_val, f2(x_val)))
cat(sprintf("f3(%.1f) = %.2f\n", x_val, f3(x_val)))

# Evaluar en multiples puntos
x_array <- seq(-5, 5, length.out = 100)
y_array <- f4(x_array)

# Graficar
plot(x_array, y_array, type = "l", col = "blue", lwd = 2,
     xlab = "x", ylab = "f(x)",
     main = expression(f(x) == frac(x^2 - 1, x^2 + 1)))
grid()
abline(h = 0, v = 0, lty = 2, col = "gray50")
```

Listing 2.7: Determinación de dominio y rango en R

```r
# Encontrar dominio numerico
encontrar_dominio_numerico <- function(f, x_min = -100, x_max =
   100,
                                       n_puntos = 10000) {
  x_test <- seq(x_min, x_max, length.out = n_puntos)
  dominio <- c()

  for (x in x_test) {
    y <- tryCatch(f(x), error = function(e) NA)
    if (!is.na(y) && is.finite(y)) {
      dominio <- c(dominio, x)
    }
  }

  if (length(dominio) > 0) {
    return(c(min(dominio), max(dominio)))
  }
  return(c(NA, NA))
}

# Encontrar rango numerico
encontrar_rango_numerico <- function(f, x_min, x_max, n_puntos =
```

```
   10000) {
  x_vals <- seq(x_min, x_max, length.out = n_puntos)
  y_vals <- c()

  for (x in x_vals) {
    y <- tryCatch(f(x), error = function(e) NA)
    if (!is.na(y) && is.finite(y)) {
      y_vals <- c(y_vals, y)
    }
  }

  if (length(y_vals) > 0) {
    return(c(min(y_vals), max(y_vals)))
  }
  return(c(NA, NA))
}

# Ejemplo de uso
f <- function(x) sqrt(4 - x^2)  # Semicirculo

dominio <- encontrar_dominio_numerico(f, -10, 10)
cat(sprintf("Dominio numerico: [%.2f, %.2f]\n", dominio[1], dominio
   [2]))

rango <- encontrar_rango_numerico(f, dominio[1], dominio[2])
cat(sprintf("Rango numerico: [%.2f, %.2f]\n", rango[1], rango[2]))
```

Listing 2.8: Composición y análisis de funciones en R

```
# Composicion de funciones
componer <- function(f, g) {
  function(x) f(g(x))
}

# Ejemplo
f <- function(x) x^2
g <- function(x) sin(x)
fog <- componer(f, g)  # f(g(x)) = sin^2(x)
gof <- componer(g, f)  # g(f(x)) = sin(x^2)

x <- pi/4
cat(sprintf("f(g(%.3f)) = %.4f\n", x, fog(x)))
```

```r
cat(sprintf("g(f(%.3f)) = %.4f\n", x, gof(x)))

# Verificar propiedades
es_par <- function(f, x_test = seq(-10, 10, length.out = 100), tol = 1e-6) {
  for (x in x_test) {
    if (x != 0 && abs(f(x) - f(-x)) > tol) {
      return(FALSE)
    }
  }
  return(TRUE)
}

es_impar <- function(f, x_test = seq(-10, 10, length.out = 100), tol = 1e-6) {
  for (x in x_test) {
    if (x != 0 && abs(f(-x) + f(x)) > tol) {
      return(FALSE)
    }
  }
  return(TRUE)
}

es_creciente <- function(f, a, b, n_puntos = 100) {
  x <- seq(a, b, length.out = n_puntos)
  y <- sapply(x, f)
  return(all(diff(y) >= 0))
}

# Ejemplos
f_par <- function(x) x^2
f_impar <- function(x) x^3
f_creciente <- function(x) exp(x)

cat(sprintf("x^2 es par: %s\n", es_par(f_par)))
cat(sprintf("x^3 es impar: %s\n", es_impar(f_impar)))
cat(sprintf("e^x es creciente en [0,5]: %s\n", es_creciente(f_creciente, 0, 5)))
```

2.4.2. Ejercicios teóricos

Ejercicio 2.1. Clasifique las siguientes variables como independientes o dependientes, y como discretas o continuas:

1. La temperatura de un horno en función del tiempo
2. El número de bacterias en un cultivo después de n horas
3. La posición de un péndulo en función del tiempo
4. El costo total de producción en función del número de unidades
5. La concentración de un medicamento en la sangre en función del tiempo

Ejercicio 2.2. Para cada función, determine su dominio y rango:

1. $f(x) = \sqrt{4 - x^2}$
2. $g(x) = \frac{x}{x^2-1}$
3. $h(x) = \ln(x^2 - 4)$
4. $k(x) = e^{-x^2}$
5. $m(x) = \frac{1}{\sqrt{x+2}}$
6. $n(x) = \arcsin(2x - 1)$

Ejercicio 2.3. Dadas $f(x) = x^2 + 1$ y $g(x) = \sqrt{x}$:

1. Encuentre $(f \circ g)(x)$ y su dominio
2. Encuentre $(g \circ f)(x)$ y su dominio
3. ¿Son iguales? Explique
4. Calcule $(f \circ g)(4)$ y $(g \circ f)(4)$

Ejercicio 2.4. Determine si las siguientes funciones son pares, impares o ninguna:

1. $f(x) = x^4 - 2x^2 + 1$
2. $g(x) = x^3 - x$
3. $h(x) = \frac{x}{x^2+1}$
4. $k(x) = |x| + x$
5. $m(x) = \sin(x)\cos(x)$

Ejercicio 2.5. Para la función $f(x) = |x^2 - 4|$:

1. Escriba la función por partes (sin usar valor absoluto)
2. Determine su dominio y rango
3. Grafique la función
4. Identifique los puntos donde no es diferenciable

Ejercicio 2.6. Una partícula se mueve según la ecuación:

$$s(t) = t^3 - 6t^2 + 9t + 1 \tag{2.64}$$

donde s es la posición en metros y t es el tiempo en segundos.

1. ¿Cuál es la variable independiente y cuál la dependiente?
2. Encuentre la velocidad $v(t) = s'(t)$
3. ¿En qué instantes la partícula está en reposo?
4. ¿Cuál es el dominio práctico si el movimiento dura 5 segundos?

Ejercicio 2.7. Determine la función inversa de cada una y verifique su respuesta:

1. $f(x) = 3x - 7$
2. $g(x) = \frac{x+1}{x-2}$, $x \neq 2$
3. $h(x) = \sqrt{x+4}$, $x \geq -4$
4. $k(x) = 2^{x+1}$

Ejercicio 2.8. Una función de demanda está dada por:

$$D(p) = \frac{1000}{p+5} \tag{2.65}$$

donde p es el precio en dólares y D es la cantidad demandada.

1. ¿Cuál es el dominio práctico?
2. ¿Cuál es el rango práctico?
3. Si el precio aumenta, ¿qué pasa con la demanda?
4. Encuentre la función inversa $p(D)$ (precio como función de demanda)

Ejercicio 2.9. Demuestre que:

1. La suma de dos funciones pares es par
2. El producto de dos funciones impares es par
3. La composición de dos funciones inyectivas es inyectiva
4. Si f es creciente y g es decreciente, entonces $f \circ g$ es decreciente

Ejercicio 2.10. Para la función logística:

$$P(t) = \frac{K}{1 + Ae^{-rt}} \tag{2.66}$$

donde K, A, r son constantes positivas:

1. Determine el dominio y rango
2. Calcule $\lim_{t\to\infty} P(t)$
3. ¿Qué representa K en el contexto del crecimiento poblacional?
4. Grafique cualitativamente la función

Ejercicio 2.11. Considere la función:

$$f(x) = \begin{cases} x^2 & \text{si } x < 1 \\ 2x - 1 & \text{si } 1 \leq x < 3 \\ \sqrt{x+6} & \text{si } x \geq 3 \end{cases} \tag{2.67}$$

1. Determine si f es continua en $x = 1$ y $x = 3$
2. Encuentre $\lim_{x\to 1^-} f(x)$ y $\lim_{x\to 1^+} f(x)$
3. Calcule $f(0)$, $f(1)$, $f(2)$, $f(3)$, $f(4)$
4. Grafique la función

Ejercicio 2.12. En un circuito eléctrico, la impedancia Z en función de la frecuencia ω es:

$$Z(\omega) = \sqrt{R^2 + \left(\omega L - \frac{1}{\omega C}\right)^2} \tag{2.68}$$

donde R, L, C son constantes positivas.

1. ¿Cuál es el dominio natural de esta función?
2. ¿Cuál es el dominio práctico (frecuencias físicamente realizables)?
3. ¿Existe un valor mínimo de Z? Si es así, ¿en qué frecuencia ocurre?

Ejercicio 2.13 (Proyecto computacional)**.** Implemente en Python (u otro lenguaje) las siguientes funciones y grafíquelas:

1. $f(x) = \frac{\sin(x)}{x}$ (función sinc)
2. $g(x) = e^{-x^2}$ (función gaussiana)
3. $h(x) = \frac{1}{1+x^2}$ (función lorentziana)

Para cada una:

- Determine numéricamente el rango en $[-10, 10]$
- Encuentre los máximos y mínimos
- Calcule el área bajo la curva usando sumas de Riemann

Resumen del Capítulo 2

Conceptos clave:

- Las variables pueden ser independientes/dependientes, discretas/continuas, escalares/vectoriales
- Las funciones establecen relaciones entre variables y pueden representarse de múltiples formas
- Las funciones se clasifican en algebraicas (polinomiales, racionales, radicales) y trascendentes (exponenciales, logarítmicas, trigonométricas)
- El dominio es el conjunto de entradas válidas; el rango es el conjunto de salidas posibles
- Propiedades como inyectividad, monotonía, paridad y continuidad son cruciales en análisis numérico
- La composición e inversión de funciones son operaciones fundamentales

Habilidades desarrolladas:

- Identificar y clasificar variables en problemas
- Determinar dominio y rango de funciones
- Operar con funciones (composición, inversión)
- Analizar propiedades de funciones
- Representar funciones de múltiples formas

Próximo capítulo: Estudiaremos restricciones matemáticas y cómo modelar problemas con limitaciones.

Lecturas recomendadas

1. Stewart, J. - *Cálculo*, Capítulos 1-2: Funciones y límites
2. Burden, R. L., & Faires, J. D. - *Numerical Analysis*, Capítulo 1.2: Mathematical Preliminaries and Error Analysis
3. Apostol, T. M. - *Calculus*, Vol. 1, Capítulo 1: The Concepts of the Integral Calculus
4. Strang, G. - *Introduction to Applied Mathematics*, Capítulo 1: Functions and Linear Algebra

Capítulo 3

Restricciones

Objetivos del capítulo:

- Comprender el concepto de restricción matemática
- Clasificar restricciones según su naturaleza y tipo
- Formular problemas con restricciones
- Implementar restricciones en Python y R
- Resolver problemas de optimización con restricciones

3.1. Restricciones matemáticas

Las restricciones son condiciones que limitan los valores que pueden tomar las variables en un problema. En el mundo real, prácticamente todos los problemas tienen restricciones: presupuestos limitados, capacidades máximas, leyes físicas, requisitos de calidad, etc.

3.1.1. Definición formal

Definición

Una **restricción** es una condición expresada matemáticamente que debe satisfacerse en la solución de un problema. Formalmente, si $\vec{x} = (x_1, x_2, \ldots, x_n)$ es el vector de variables de decisión, una restricción tiene la forma:

$$g(\vec{x}) \ \{\leq, =, \geq\} \ b \tag{3.1}$$

donde:

- $g : \mathbb{R}^n \to \mathbb{R}$ es una función de las variables
- $b \in \mathbb{R}$ es un valor constante (lado derecho)
- El símbolo puede ser $\leq$, $=$, o $\geq$

3.1.2. Motivación: ¿Por qué necesitamos restricciones?

Problema de producción

Una fábrica produce dos productos A y B. Queremos maximizar la ganancia, pero tenemos limitaciones:

Sin restricciones:

$$\text{Maximizar: } G = 30x_A + 40x_B \tag{3.2}$$

Solución matemática: $x_A, x_B \to \infty$ (ganancia infinita)

Con restricciones (realista):

$$\text{Maximizar: } G = 30x_A + 40x_B \tag{3.3}$$

$$\text{Sujeto a: } 2x_A + 3x_B \leq 120 \quad \text{(horas de trabajo)} \tag{3.4}$$

$$x_A + x_B \leq 50 \quad \text{(espacio de almacenamiento)} \tag{3.5}$$

$$x_A, x_B \geq 0 \quad \text{(no negatividad)} \tag{3.6}$$

Ahora tenemos una solución finita y realista.

3.1.3. Región factible

Definición

La **región factible** (o región viable) es el conjunto de todos los puntos que satisfacen todas las restricciones del problema:

$$\mathcal{F} = \{\vec{x} \in \mathbb{R}^n : g_i(\vec{x}) \leq b_i,\ i = 1, \ldots, m\} \tag{3.7}$$

Nota Importante

La región factible puede ser:

- **Vacía**: No existe solución que satisfaga todas las restricciones
- **Acotada**: Todos los puntos están dentro de un área/volumen finito
- **No acotada**: Se extiende infinitamente en alguna dirección
- **Convexa o no convexa**: Afecta la dificultad del problema

3.1.4. Implementación básica en Python

Listing 3.1: Verificación de restricciones en Python

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

def verificar_restriccion(x, y, tipo='<=', valor=0):
    """
    Verifica si un punto (x,y) satisface una restriccion.

    Args:
        x, y: coordenadas del punto
        tipo: tipo de restriccion ('<=', '>=', '==')
        valor: lado derecho de la restriccion

    Returns:
        True si satisface la restriccion, False en caso contrario
    """
    if tipo == '<=':
        return x + y <= valor
    elif tipo == '>=':
        return x + y >= valor
    elif tipo == '==':
```

```
        return abs(x + y - valor) < 1e-6
    else:
        raise ValueError("Tipo de restriccion no valido")

def verificar_punto_factible(x, y, restricciones):
    """
    Verifica si un punto satisface todas las restricciones.

    Args:
        x, y: coordenadas del punto
        restricciones: lista de tuplas (funcion, tipo, valor)

    Returns:
        True si el punto es factible
    """
    for func, tipo, valor in restricciones:
        resultado = func(x, y)

        if tipo == '<=' and resultado > valor:
            return False
        elif tipo == '>=' and resultado < valor:
            return False
        elif tipo == '==' and abs(resultado - valor) > 1e-6:
            return False

    return True

# Ejemplo: Verificar punto en region factible
# Restricciones: 2x + 3y <= 12, x + y <= 5, x >= 0, y >= 0

restricciones = [
    (lambda x, y: 2*x + 3*y, '<=', 12),
    (lambda x, y: x + y, '<=', 5),
    (lambda x, y: x, '>=', 0),
    (lambda x, y: y, '>=', 0)
]

# Probar varios puntos
puntos_test = [(1, 1), (3, 2), (0, 5), (2, 3)]

for punto in puntos_test:
```

```
    es_factible = verificar_punto_factible(punto[0], punto[1],
        restricciones)
    print(f"Punto {punto}: {'Factible' if es_factible else 'No
        factible'}")
```

3.1.5. Implementación básica en R

Listing 3.2: Verificación de restricciones en R

```
# Verificar si un punto satisface una restriccion
verificar_restriccion <- function(x, y, tipo = '<=', valor = 0) {
  resultado <- x + y

  if (tipo == '<=') {
    return(resultado <= valor)
  } else if (tipo == '>=') {
    return(resultado >= valor)
  } else if (tipo == '==') {
    return(abs(resultado - valor) < 1e-6)
  } else {
    stop("Tipo de restriccion no valido")
  }
}

# Verificar si un punto es factible
verificar_punto_factible <- function(x, y, restricciones) {
  for (i in 1:nrow(restricciones)) {
    func <- restricciones$func[[i]]
    tipo <- restricciones$tipo[i]
    valor <- restricciones$valor[i]

    resultado <- func(x, y)

    if (tipo == '<=' && resultado > valor) {
      return(FALSE)
    } else if (tipo == '>=' && resultado < valor) {
      return(FALSE)
    } else if (tipo == '==' && abs(resultado - valor) > 1e-6) {
      return(FALSE)
    }
  }

```

```
  return(TRUE)
}

# Ejemplo de uso
restricciones <- data.frame(
  func = I(list(
    function(x, y) 2*x + 3*y,
    function(x, y) x + y,
    function(x, y) x,
    function(x, y) y
  )),
  tipo = c('<=', '<=', '>=', '>='),
  valor = c(12, 5, 0, 0)
)

# Probar varios puntos
puntos_test <- list(c(1, 1), c(3, 2), c(0, 5), c(2, 3))

for (punto in puntos_test) {
  es_factible <- verificar_punto_factible(punto[1], punto[2],
     restricciones)
  cat(sprintf("Punto (%.0f, %.0f): %s\n",
              punto[1], punto[2],
              ifelse(es_factible, "Factible", "No factible")))
}
```

3.2. Tipos de restricciones

Las restricciones pueden clasificarse según diferentes criterios. Esta clasificación es fundamental para elegir el método de solución adecuado.

3.2.1. Clasificación según la igualdad/desigualdad

Restricciones de igualdad

Definición

Una **restricción de igualdad** exige que una función tome exactamente un valor específico:

$$h(\vec{x}) = b \tag{3.8}$$

Ejemplo

- Balance de materia: $x_1 + x_2 + x_3 = 100$ (kg totales)
- Conservación de energía: $E_{inicial} = E_{final}$
- Presupuesto exacto: $\sum_{i=1}^{n} c_i x_i = B$

Restricciones de desigualdad

Definición

Una **restricción de desigualdad** establece un límite superior o inferior:

$$g(\vec{x}) \leq b \quad \text{(límite superior)} \tag{3.9}$$

$$g(\vec{x}) \geq b \quad \text{(límite inferior)} \tag{3.10}$$

Ejemplo

- Capacidad máxima: $x_1 + x_2 \leq 1000$ (unidades)
- Requerimiento mínimo: $3x_1 + 2x_2 \geq 50$ (nutrientes)
- No negatividad: $x_i \geq 0$ para todo i

3.2.2. Clasificación según la linealidad

Restricciones lineales

Definición

Una restricción es **lineal** si tiene la forma:

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \; \{\leq, =, \geq\} \; b \tag{3.11}$$

donde a_i y b son constantes.

Propiedades importantes:

- La región factible es un poliedro (convexo)
- Son más fáciles de resolver computacionalmente
- El óptimo (si existe) está en un vértice del poliedro

Sistema lineal de restricciones

$$2x + 3y \leq 12 \tag{3.12}$$
$$x + 2y \leq 8 \tag{3.13}$$
$$x, y \geq 0 \tag{3.14}$$

Restricciones no lineales

Definición

Una restricción es **no lineal** si involucra productos de variables, potencias, funciones trascendentes, etc.

Restricciones no lineales comunes

- Cuadrática: $x^2 + y^2 \leq 25$ (círculo)
- Producto: $xy \geq 10$ (hipérbola)
- Exponencial: $e^x + y \leq 5$
- Trigonométrica: $\sin(x) + \cos(y) = 1$

3.2.3. Clasificación según el dominio

Tipo	Descripción	Ejemplo
Continuas	Variables reales	$x \in \mathbb{R}$
Enteras	Variables enteras	$x \in \mathbb{Z}$
Binarias	Variables 0-1	$x \in \{0, 1\}$
Mixtas	Combinación	$x_1 \in \mathbb{R}, x_2 \in \mathbb{Z}$

Tabla 3.1: Clasificación según el dominio de las variables

3.2.4. Restricciones activas e inactivas

Definición

En un punto $\vec{x}^*$:

- Una restricción $g(\vec{x}) \leq b$ es **activa** si $g(\vec{x}^*) = b$
- Una restricción es **inactiva** si $g(\vec{x}^*) < b$

Nota Importante

Las restricciones activas determinan la solución óptima. Las inactivas podrían eliminarse sin cambiar el resultado.

3.2.5. Visualización de restricciones en Python

Listing 3.3: Visualización de región factible en 2D

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon

def graficar_region_factible_2d(restricciones, xlim=(0, 10), ylim=(0, 10)):
    """
    Grafica la region factible de un problema 2D.

    Args:
        restricciones: lista de diccionarios con 'a', 'b', 'c', 'tipo'
                       donde la restriccion es: a*x + b*y {tipo} c
        xlim, ylim: limites de los ejes
    """
    fig, ax = plt.subplots(figsize=(10, 8))

    # Crear grilla de puntos
    x = np.linspace(xlim[0], xlim[1], 400)
    y = np.linspace(ylim[0], ylim[1], 400)
    X, Y = np.meshgrid(x, y)

    # Inicializar region factible (todos los puntos son factibles)
    region_factible = np.ones_like(X, dtype=bool)

    # Aplicar cada restriccion
    for i, rest in enumerate(restricciones):
        a, b, c, tipo = rest['a'], rest['b'], rest['c'], rest['tipo']

        # Calcular lado izquierdo
        Z = a * X + b * Y

```

```
        # Aplicar restriccion
        if tipo == '<=':
            region_factible &= (Z <= c)
            # Dibujar linea de restriccion
            if b != 0:
                y_line = (c - a * x) / b
                ax.plot(x, y_line, 'b-', linewidth=2,
                        label=f'{a}x + {b}y     {c}')
                ax.fill_between(x, 0, y_line, alpha=0.1, color='
                   blue')
        elif tipo == '>=':
            region_factible &= (Z >= c)
            if b != 0:
                y_line = (c - a * x) / b
                ax.plot(x, y_line, 'r-', linewidth=2,
                        label=f'{a}x + {b}y     {c}')
                ax.fill_between(x, y_line, ylim[1], alpha=0.1,
                   color='red')
        elif tipo == '==':
            region_factible &= (np.abs(Z - c) < 0.1)
            if b != 0:
                y_line = (c - a * x) / b
                ax.plot(x, y_line, 'g--', linewidth=2,
                        label=f'{a}x + {b}y = {c}')

    # Graficar region factible
    ax.contourf(X, Y, region_factible.astype(int),
                levels=[0.5, 1.5], colors=['green'], alpha=0.3)

    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
    ax.set_xlabel('x', fontsize=12)
    ax.set_ylabel('y', fontsize=12)
    ax.set_title('Regi n Factible', fontsize=14, fontweight='bold'
       )
    ax.grid(True, alpha=0.3)
    ax.legend(loc='best')
    ax.axhline(y=0, color='k', linewidth=0.5)
    ax.axvline(x=0, color='k', linewidth=0.5)

    return fig, ax
```

```
69
70  # Ejemplo de uso
71  restricciones_ejemplo = [
72      {'a': 2, 'b': 3, 'c': 12, 'tipo': '<='},  # 2x + 3y <= 12
73      {'a': 1, 'b': 1, 'c': 5, 'tipo': '<='},   # x + y <= 5
74      {'a': 1, 'b': 0, 'c': 0, 'tipo': '>='},   # x >= 0
75      {'a': 0, 'b': 1, 'c': 0, 'tipo': '>='},   # y >= 0
76  ]
77
78  fig, ax = graficar_region_factible_2d(restricciones_ejemplo, xlim
        =(0, 8), ylim=(0, 8))
79  plt.show()
80
81  # Encontrar vertices de la region factible
82  def encontrar_vertices(restricciones, xlim=(0, 10), ylim=(0, 10)):
83      """Encuentra los vertices de la region factible."""
84      from scipy.optimize import linprog
85      from itertools import combinations
86
87      vertices = []
88
89      # Probar intersecciones de pares de restricciones
90      for r1, r2 in combinations(restricciones, 2):
91          # Resolver sistema 2x2
92          A = np.array([[r1['a'], r1['b']],
93                        [r2['a'], r2['b']]])
94          b = np.array([r1['c'], r2['c']])
95
96          try:
97              punto = np.linalg.solve(A, b)
98
99              # Verificar si el punto satisface todas las
                    restricciones
100             if xlim[0] <= punto[0] <= xlim[1] and ylim[0] <= punto
                    [1] <= ylim[1]:
101                 es_factible = True
102                 for rest in restricciones:
103                     val = rest['a'] * punto[0] + rest['b'] * punto
                            [1]
104                     if rest['tipo'] == '<=' and val > rest['c'] + 1
                            e-6:
```

```
                            es_factible = False
                            break
                        elif rest['tipo'] == '>=' and val < rest['c'] -
                            1e-6:
                            es_factible = False
                            break

                if es_factible:
                    vertices.append(tuple(punto))
        except:
            pass

    # Eliminar duplicados
    vertices = list(set(vertices))
    return vertices

vertices = encontrar_vertices(restricciones_ejemplo, xlim=(0, 8),
   ylim=(0, 8))
print("V rtices de la regi n factible:")
for v in vertices:
    print(f"  ({v[0]:.2f}, {v[1]:.2f})")
```

3.2.6. Visualización de restricciones en R

Listing 3.4: Visualización de región factible en R

```
library(ggplot2)

# Funcion para graficar region factible
graficar_region_factible <- function(restricciones, xlim = c(0, 10)
   ,
                                     ylim = c(0, 10)) {
  # Crear grilla de puntos
  x <- seq(xlim[1], xlim[2], length.out = 400)
  y <- seq(ylim[1], ylim[2], length.out = 400)
  grid <- expand.grid(x = x, y = y)

  # Verificar cada punto
  grid$factible <- apply(grid, 1, function(punto) {
    x_val <- punto[1]
    y_val <- punto[2]

```

```
    for (i in 1:nrow(restricciones)) {
      a <- restricciones$a[i]
      b <- restricciones$b[i]
      c <- restricciones$c[i]
      tipo <- restricciones$tipo[i]

      resultado <- a * x_val + b * y_val

      if (tipo == '<=' && resultado > c) return(FALSE)
      if (tipo == '>=' && resultado < c) return(FALSE)
      if (tipo == '==' && abs(resultado - c) > 0.1) return(FALSE)
    }
    return(TRUE)
  })

  # Crear grafico
  p <- ggplot(grid, aes(x = x, y = y, fill = factible)) +
    geom_tile() +
    scale_fill_manual(values = c("FALSE" = "white", "TRUE" = "
       lightgreen"),
                      labels = c("No factible", "Factible")) +
    labs(x = "x", y = "y", title = "Regi n Factible",
         fill = "") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5, face = "bold",
       size = 14))

  # Agregar lineas de restricciones
  for (i in 1:nrow(restricciones)) {
    a <- restricciones$a[i]
    b <- restricciones$b[i]
    c <- restricciones$c[i]

    if (b != 0) {
      y_line <- (c - a * x) / b
      df_line <- data.frame(x = x, y = y_line)
      p <- p + geom_line(data = df_line, aes(x = x, y = y, fill =
         NULL),
                         color = "blue", size = 1.2)
    }
  }
```

```
54
55    return(p)
56  }
57
58  # Ejemplo de uso
59  restricciones <- data.frame(
60    a = c(2, 1, 1, 0),
61    b = c(3, 1, 0, 1),
62    c = c(12, 5, 0, 0),
63    tipo = c('<=', '<=', '>=', '>=')
64  )
65
66  grafico <- graficar_region_factible(restricciones, xlim = c(0, 8),
67                                      ylim = c(0, 8))
68  print(grafico)
```

3.3. Problemas con restricciones

Los problemas con restricciones aparecen en prácticamente todas las áreas de aplicación. Veremos ejemplos detallados y su implementación.

3.3.1. Problema de programación lineal

Definición

Un **problema de programación lineal** (PL) tiene la forma general:

$$\text{Maximizar (o Minimizar): } \vec{c}^T\vec{x} \tag{3.15}$$

$$\text{Sujeto a: } A\vec{x} \leq \vec{b} \tag{3.16}$$

$$\vec{x} \geq \vec{0} \tag{3.17}$$

donde $\vec{c} \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $\vec{b} \in \mathbb{R}^m$, y $\vec{x} \in \mathbb{R}^n$.

Problema de la dieta

Un nutricionista debe diseñar una dieta que minimice el costo mientras cumple requerimientos nutricionales.

Variables:

- x_1: porciones de alimento 1 (pan)
- x_2: porciones de alimento 2 (leche)
- x_3: porciones de alimento 3 (carne)

Función objetivo:

$$\text{Minimizar: } Z = 2x_1 + 3x_2 + 5x_3 \quad \text{(costo en \$)} \tag{3.18}$$

Restricciones:

$$4x_1 + 8x_2 + 6x_3 \geq 40 \quad \text{(calorías mínimas)} \tag{3.19}$$

$$2x_1 + 3x_2 + 4x_3 \geq 20 \quad \text{(proteínas mínimas)} \tag{3.20}$$

$$x_1, x_2, x_3 \geq 0 \quad \text{(no negatividad)} \tag{3.21}$$

Solución en Python usando scipy

Listing 3.5: Programación lineal con scipy.optimize

```
from scipy.optimize import linprog
import numpy as np

def resolver_problema_dieta():
    """
    Resuelve el problema de la dieta usando programacion lineal.
    """
    # Coeficientes de la funcion objetivo (minimizar)
    c = np.array([2, 3, 5])  # Costos

    # Restricciones de desigualdad: A_ub * x <= b_ub
    # Nota: necesitamos convertir >= a <= multiplicando por -1
    A_ub = np.array([
        [-4, -8, -6],  # -calor as >= -40  =>  calor as <= -40 (
            -1)
        [-2, -3, -4]   # -prote nas >= -20
    ])
```

```
    b_ub = np.array([-40, -20])

    # L mites de las variables (no negatividad)
    x_bounds = [(0, None), (0, None), (0, None)]

    # Resolver
    resultado = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=x_bounds,
                        method='highs')

    if resultado.success:
        print("=" * 50)
        print("SOLUCI N DEL PROBLEMA DE LA DIETA")
        print("=" * 50)
        print(f"Porciones de pan (x1): {resultado.x[0]:.2f}")
        print(f"Porciones de leche (x2): {resultado.x[1]:.2f}")
        print(f"Porciones de carne (x3): {resultado.x[2]:.2f}")
        print(f"\nCosto m nimo: ${resultado.fun:.2f}")
        print("=" * 50)

        # Verificar restricciones
        print("\nVerificaci n de restricciones:")
        calorias = 4*resultado.x[0] + 8*resultado.x[1] + 6*
           resultado.x[2]
        proteinas = 2*resultado.x[0] + 3*resultado.x[1] + 4*
           resultado.x[2]
        print(f"Calor as: {calorias:.2f} >= 40    " if calorias >=
            40 else f"Calor as: {calorias:.2f} < 40    ")
        print(f"Prote nas: {proteinas:.2f} >= 20    " if proteinas
            >= 20 else f"Prote nas: {proteinas:.2f} < 20    ")

        return resultado
    else:
        print("No se encontr  soluci n factible")
        return None

# Ejecutar
resultado = resolver_problema_dieta()
```

Solución en R usando lpSolve

Listing 3.6: Programación lineal en R

```r
library(lpSolve)

resolver_problema_dieta <- function() {
  # Coeficientes de la funcion objetivo (minimizar)
  objetivo <- c(2, 3, 5)  # Costos

  # Matriz de restricciones
  # Calorias: 4x1 + 8x2 + 6x3 >= 40
  # Proteinas: 2x1 + 3x2 + 4x3 >= 20
  restricciones <- matrix(c(
    4, 8, 6,
    2, 3, 4
  ), nrow = 2, byrow = TRUE)

  # Lados derechos
  rhs <- c(40, 20)

  # Direcciones de las restricciones
  direcciones <- c(">=", ">=")

  # Resolver
  solucion <- lp("min", objetivo, restricciones, direcciones, rhs)

  if (solucion$status == 0) {
    cat("==========================================\n")
    cat("SOLUCI N DEL PROBLEMA DE LA DIETA\n")
    cat("==========================================\n")
    cat(sprintf("Porciones de pan (x1): %.2f\n", solucion$solution
       [1]))
    cat(sprintf("Porciones de leche (x2): %.2f\n", solucion$
       solution[2]))
    cat(sprintf("Porciones de carne (x3): %.2f\n", solucion$
       solution[3]))
    cat(sprintf("\nCosto m nimo: $%.2f\n", solucion$objval))
    cat("==========================================\n")

    # Verificar restricciones
    cat("\nVerificaci n de restricciones:\n")
    calorias <- 4*solucion$solution[1] + 8*solucion$solution[2] + 6
       *solucion$solution[3]
```

```
    proteinas <- 2*solucion$solution[1] + 3*solucion$solution[2] +
       4*solucion$solution[3]

    cat(sprintf("Calor as: %.2f >= 40 %s\n", calorias,
                ifelse(calorias >= 40, "   ", "   ")))
    cat(sprintf("Prote nas: %.2f >= 20 %s\n", proteinas,
                ifelse(proteinas >= 20, "   ", "   ")))

    return(solucion)
  } else {
    cat("No se encontr  soluci n factible\n")
    return(NULL)
  }
}

# Ejecutar
resultado <- resolver_problema_dieta()
```

3.3.2. Problema de optimización no lineal con restricciones

Diseño de un contenedor cilíndrico

Se desea diseñar un contenedor cilíndrico que minimice el costo de material sujeto a que contenga un volumen mínimo.

Variables:

- r: radio de la base (metros)
- h: altura del cilindro (metros)

Función objetivo: Minimizar el área superficial (costo)

$$A(r, h) = 2\pi r^2 + 2\pi rh \tag{3.22}$$

Restricciones:

$$\pi r^2 h \geq V_{min} \quad \text{(volumen mínimo)} \tag{3.23}$$

$$r, h > 0 \quad \text{(físicamente válido)} \tag{3.24}$$

Solución en Python

Listing 3.7: Optimización no lineal con scipy

```python
from scipy.optimize import minimize
import numpy as np

def disenar_contenedor(V_min=100):
    """
    Dise a un contenedor cil ndrico de costo m nimo.

    Args:
        V_min: Volumen m nimo requerido (m^3)
    """

    # Funci n objetivo:  rea  superficial
    def area_superficie(x):
        r, h = x
        return 2*np.pi*r**2 + 2*np.pi*r*h

    # Restricci n: volumen >= V_min
    def restriccion_volumen(x):
        r, h = x
        return np.pi * r**2 * h - V_min

    # Definir restricciones en formato de scipy
    restricciones = [
        {'type': 'ineq', 'fun': restriccion_volumen}  # ineq: >= 0
    ]

    # L mites de las variables (r > 0, h > 0)
    limites = [(0.1, None), (0.1, None)]

    # Punto inicial (adivinanza)
    x0 = np.array([3.0, 5.0])

    # Resolver
    resultado = minimize(area_superficie, x0, method='SLSQP',
                         bounds=limites, constraints=restricciones)

    if resultado.success:
        r_opt, h_opt = resultado.x
        print("=" * 60)
        print("DISE O  PTIMO  DEL CONTENEDOR")
        print("=" * 60)
```

```
        print(f"Radio  ptimo : {r_opt:.3f} m")
        print(f"Altura  ptima : {h_opt:.3f} m")
        print(f" rea  superficial m nima: {resultado.fun:.3f} m  "
           )
        print(f"Volumen: {np.pi * r_opt**2 * h_opt:.3f} m  ")
        print(f"Relaci n h/r: {h_opt/r_opt:.3f}")
        print("=" * 60)

        # Visualizaci n
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D

        fig = plt.figure(figsize=(12, 5))

        # Gr fico 1: Contornos de  rea  superficial
        ax1 = fig.add_subplot(121)
        r_range = np.linspace(0.5, 8, 100)
        h_range = np.linspace(0.5, 15, 100)
        R, H = np.meshgrid(r_range, h_range)
        A = 2*np.pi*R**2 + 2*np.pi*R*H

        # Regi n factible (donde volumen >= V_min)
        V = np.pi * R**2 * H
        A_masked = np.ma.masked_where(V < V_min, A)

        contour = ax1.contourf(R, H, A_masked, levels=20, cmap='
           viridis')
        ax1.plot(r_opt, h_opt, 'r*', markersize=20, label=' ptimo '
           )
        ax1.set_xlabel('Radio (m)', fontsize=11)
        ax1.set_ylabel('Altura (m)', fontsize=11)
        ax1.set_title(' rea  Superficial (regi n factible)',
           fontsize=12)
        ax1.legend()
        ax1.grid(True, alpha=0.3)
        plt.colorbar(contour, ax=ax1, label=' rea  (m )')

        # Gr fico 2: Cilindro 3D
        ax2 = fig.add_subplot(122, projection='3d')
        theta = np.linspace(0, 2*np.pi, 50)
        z = np.linspace(0, h_opt, 50)
```

```
        Theta, Z = np.meshgrid(theta, z)
        X = r_opt * np.cos(Theta)
        Y = r_opt * np.sin(Theta)

        ax2.plot_surface(X, Y, Z, alpha=0.7, cmap='coolwarm')
        ax2.set_xlabel('X (m)')
        ax2.set_ylabel('Y (m)')
        ax2.set_zlabel('Z (m)')
        ax2.set_title(f'Contenedor  ptimo \nV = {V_min} m  ',
           fontsize=12)

        plt.tight_layout()
        plt.show()

        return resultado
    else:
        print("No se encontr  soluci n")
        return None

# Ejecutar
resultado = disenar_contenedor(V_min=100)
```

Solución en R

Listing 3.8: Optimización no lineal en R

```
library(nloptr)

disenar_contenedor <- function(V_min = 100) {
  # Funci n objetivo:  rea  superficial
  area_superficie <- function(x) {
    r <- x[1]
    h <- x[2]
    return(2*pi*r^2 + 2*pi*r*h)
  }

  # Gradiente de la funci n objetivo
  grad_area <- function(x) {
    r <- x[1]
    h <- x[2]
    return(c(4*pi*r + 2*pi*h, 2*pi*r))
  }
```

```r

# Restricci n: volumen >= V_min
restriccion_volumen <- function(x) {
  r <- x[1]
  h <- x[2]
  return(pi * r^2 * h - V_min)
}

# Gradiente de la restricci n
grad_restriccion <- function(x) {
  r <- x[1]
  h <- x[2]
  return(c(2*pi*r*h, pi*r^2))
}

# Punto inicial
x0 <- c(3.0, 5.0)

# Opciones
opts <- list(
  algorithm = "NLOPT_LD_SLSQP",
  xtol_rel = 1e-8,
  maxeval = 1000
)

# Resolver
resultado <- nloptr(
  x0 = x0,
  eval_f = area_superficie,
  eval_grad_f = grad_area,
  lb = c(0.1, 0.1),
  ub = c(Inf, Inf),
  eval_g_ineq = function(x) -restriccion_volumen(x),  # g(x) <= 0
  eval_jac_g_ineq = function(x) -grad_restriccion(x),
  opts = opts
)

if (resultado$status >= 0) {
  r_opt <- resultado$solution[1]
  h_opt <- resultado$solution[2]

```

```
    cat("
       ============================================================\
       n")
    cat("DISE O   PTIMO   DEL CONTENEDOR\n")
    cat("
       ============================================================\
       n")
    cat(sprintf("Radio   ptimo  : %.3f m\n", r_opt))
    cat(sprintf("Altura   ptima  : %.3f m\n", h_opt))
    cat(sprintf(" rea  superficial  m nima: %.3f  m  \n", resultado$
       objective))
    cat(sprintf("Volumen: %.3f  m  \n", pi * r_opt^2 * h_opt))
    cat(sprintf("Relaci n h/r: %.3f\n", h_opt/r_opt))
    cat("
       ============================================================\
       n")

    return(resultado)
  } else {
    cat("No se encontr  soluci n\n")
    return(NULL)
  }
}

# Ejecutar
resultado <- disenar_contenedor(V_min = 100)
```

3.3.3. Problema de asignación de recursos

Producción óptima con múltiples recursos

Una fábrica produce tres productos (A, B, C) que requieren dos recursos limitados (trabajo y material).

Datos del problema:

Formulación matemática:

$$\text{Maximizar: } Z = 30x_A + 40x_B + 50x_C \tag{3.25}$$

$$\text{Sujeto a: } 2x_A + x_B + 3x_C \leq 100 \quad \text{(trabajo)} \tag{3.26}$$

$$x_A + 2x_B + 2x_C \leq 80 \quad \text{(material)} \tag{3.27}$$

$$x_A, x_B, x_C \geq 0 \tag{3.28}$$

Listing 3.9: Problema de asignación de recursos

```
import numpy as np
from scipy.optimize import linprog
import matplotlib.pyplot as plt

def problema_produccion():
    """Resuelve el problema de producci n  ptima ."""

    # Coeficientes de ganancia (negativo porque linprog minimiza)
    c = np.array([-30, -40, -50])

    # Restricciones de recursos
    A_ub = np.array([
        [2, 1, 3],   # Trabajo
        [1, 2, 2]    # Material
    ])
    b_ub = np.array([100, 80])

    # Resolver
    resultado = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=(0, None),
                        method='highs')

    if resultado.success:
        print("=" * 60)
        print("PLAN DE PRODUCCI N  PTIMO ")
        print("=" * 60)
        print(f"Producir {resultado.x[0]:.2f} unidades de A")
        print(f"Producir {resultado.x[1]:.2f} unidades de B")
        print(f"Producir {resultado.x[2]:.2f} unidades de C")
        print(f"\nGanancia m xima: ${-resultado.fun:.2f}")
        print("=" * 60)

        # An lisis de uso de recursos
        trabajo_usado = 2*resultado.x[0] + resultado.x[1] + 3*
            resultado.x[2]
        material_usado = resultado.x[0] + 2*resultado.x[1] + 2*
            resultado.x[2]

        print("\nUso de recursos:")
        print(f"Trabajo: {trabajo_usado:.2f}/100 h ({trabajo_usado
            /100*100:.1f}%)")
```

```
38          print(f"Material: {material_usado:.2f}/80 kg ({
               material_usado/80*100:.1f}%)")
39
40          # Identificar restricciones activas
41          print("\nRestricciones activas:")
42          if abs(trabajo_usado - 100) < 0.01:
43              print("  - Trabajo (l mite alcanzado)")
44          if abs(material_usado - 80) < 0.01:
45              print("  - Material (l mite alcanzado)")
46
47          # Visualizaci n
48          fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))
49
50          # Gr fico 1: Producci n  ptima
51          productos = ['A', 'B', 'C']
52          cantidades = resultado.x
53          colores = ['#FF6B6B', '#4ECDC4', '#45B7D1']
54
55          ax1.bar(productos, cantidades, color=colores, alpha=0.8,
               edgecolor='black')
56          ax1.set_xlabel('Producto', fontsize=12)
57          ax1.set_ylabel('Unidades', fontsize=12)
58          ax1.set_title('Plan de Producci n  ptimo ', fontsize=14,
               fontweight='bold')
59          ax1.grid(axis='y', alpha=0.3)
60
61          for i, v in enumerate(cantidades):
62              ax1.text(i, v + 1, f'{v:.1f}', ha='center', va='bottom'
                   , fontsize=10)
63
64          # Gr fico 2: Uso de recursos
65          recursos = ['Trabajo', 'Material']
66          usado = [trabajo_usado, material_usado]
67          disponible = [100, 80]
68
69          x = np.arange(len(recursos))
70          width = 0.35
71
72          ax2.bar(x - width/2, usado, width, label='Usado', color='#
               FF6B6B', alpha=0.8)
73          ax2.bar(x + width/2, disponible, width, label='Disponible',
```

```
                color='#95E1D3', alpha=0.8)

        ax2.set_xlabel('Recurso', fontsize=12)
        ax2.set_ylabel('Cantidad', fontsize=12)
        ax2.set_title('Uso de Recursos', fontsize=14, fontweight='
           bold')
        ax2.set_xticks(x)
        ax2.set_xticklabels(recursos)
        ax2.legend()
        ax2.grid(axis='y', alpha=0.3)

        plt.tight_layout()
        plt.show()

        return resultado
    else:
        print("No se encontr  soluci n factible")
        return None

# Ejecutar
resultado = problema_produccion()
```

3.3.4. Análisis de sensibilidad

El análisis de sensibilidad estudia cómo cambia la solución óptima cuando varían los parámetros del problema.

Listing 3.10: Análisis de sensibilidad

```
def analisis_sensibilidad_recursos(recurso_inicial, recurso_final,
   paso=5):
    """
    Analiza c mo cambia la ganancia al variar la disponibilidad de
        trabajo.
    """
    import numpy as np
    from scipy.optimize import linprog

    c = np.array([-30, -40, -50])
    recursos_trabajo = np.arange(recurso_inicial, recurso_final +
       1, paso)
    ganancias = []
```

```
    producciones = []

    for trabajo in recursos_trabajo:
        A_ub = np.array([[2, 1, 3], [1, 2, 2]])
        b_ub = np.array([trabajo, 80])

        resultado = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=(0,
            None),
                            method='highs')

        if resultado.success:
            ganancias.append(-resultado.fun)
            producciones.append(resultado.x)
        else:
            ganancias.append(0)
            producciones.append([0, 0, 0])

    # Graficar
    plt.figure(figsize=(12, 5))

    # Gr fico 1: Ganancia vs Recurso
    plt.subplot(1, 2, 1)
    plt.plot(recursos_trabajo, ganancias, 'b-o', linewidth=2,
        markersize=6)
    plt.xlabel('Horas de trabajo disponibles', fontsize=11)
    plt.ylabel('Ganancia m xima ($)', fontsize=11)
    plt.title('Sensibilidad: Ganancia vs Trabajo', fontsize=12,
        fontweight='bold')
    plt.grid(True, alpha=0.3)

    # Gr fico 2: Producci n vs Recurso
    plt.subplot(1, 2, 2)
    producciones_array = np.array(producciones)
    plt.plot(recursos_trabajo, producciones_array[:, 0], 'r-o',
        label='Producto A')
    plt.plot(recursos_trabajo, producciones_array[:, 1], 'g-o',
        label='Producto B')
    plt.plot(recursos_trabajo, producciones_array[:, 2], 'b-o',
        label='Producto C')
    plt.xlabel('Horas de trabajo disponibles', fontsize=11)
    plt.ylabel('Unidades producidas', fontsize=11)
```

```
    plt.title('Sensibilidad: Producci n vs Trabajo', fontsize=12,
        fontweight='bold')
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

    return recursos_trabajo, ganancias, producciones_array

# Ejecutar an lisis
recursos, ganancias, producciones = analisis_sensibilidad_recursos
    (50, 150, 10)
```

3.4. Ejercicios propuestos

3.4.1. Ejercicios teóricos

Ejercicio 3.1. Clasifique las siguientes restricciones según sean de igualdad/desigualdad y lineales/no lineales:

1. $3x_1 + 2x_2 - x_3 = 10$
2. $x_1^2 + x_2^2 \leq 25$
3. $e^{x_1} + \ln(x_2) \geq 5$
4. $\frac{x_1}{x_2} = 2$
5. $|x_1 - x_2| \leq 3$

Ejercicio 3.2. Para el sistema de restricciones:

$$x + 2y \leq 10 \tag{3.29}$$

$$3x + y \leq 15 \tag{3.30}$$

$$x, y \geq 0 \tag{3.31}$$

1. Grafique la región factible
2. Identifique los vértices
3. ¿Es acotada o no acotada?
4. ¿Es convexa?

Ejercicio 3.3. Demuestre que la intersección de dos regiones convexas es convexa.

Ejercicio 3.4. Considere el problema:

$$\text{Maximizar: } z = 5x + 3y \tag{3.32}$$

$$\text{Sujeto a: } x + y \leq 4 \tag{3.33}$$

$$2x + y \leq 6 \tag{3.34}$$

$$x, y \geq 0 \tag{3.35}$$

1. Resuelva gráficamente
2. Identifique las restricciones activas en la solución óptima
3. ¿Qué pasa si cambiamos el lado derecho de la primera restricción a 5?

3.4.2. Ejercicios de programación

Ejercicio 3.5 (Implementación)**.** Implemente en Python y R una función que:

1. Reciba una lista de restricciones lineales
2. Genere un gráfico de la región factible en 2D
3. Identifique y marque los vértices
4. Calcule el área de la región factible

Pruebe su función con al menos 3 sistemas diferentes de restricciones.

Ejercicio 3.6 (Problema de la mochila)**.** Un excursionista puede cargar máximo 15 kg. Tiene los siguientes objetos disponibles:

Objeto	**Peso (kg)**	**Valor**
Tienda	4	20
Comida	3	15
Agua	5	25
Herramientas	2	10
Ropa	1	8

1. Formule como problema de programación entera
2. Implemente y resuelva en Python usando ‘pulp‘ o scipy
3. ¿Cuál es la combinación óptima?
4. Realice análisis de sensibilidad variando el peso máximo

Ejercicio 3.7 (Problema de transporte)**.** Una empresa tiene 3 fábricas y 4 almacenes. Los costos de transporte (en \$/unidad), capacidades de producción y demandas son:

Implemente en Python o R:

1. Formulación del problema de transporte
2. Solución del problema
3. Visualización de la distribución óptima
4. Análisis del costo marginal de aumentar capacidad

Ejercicio 3.8 (Optimización de cartera)**.** Un inversor tiene \$100,000 para invertir en 4 acciones. Las restricciones son:

- No más del 40 % en una sola acción
- Al menos 10 % en cada acción
- Rendimiento esperado mínimo: 8 %
- Minimizar el riesgo (varianza)

Datos históricos: rendimientos esperados [10 %, 12 %, 8 %, 15 %] y matriz de covarianza dada.

Implemente la solución usando programación cuadrática.

Ejercicio 3.9 (Diseño óptimo)**.** Diseñe una viga rectangular que:

- Minimice el costo (proporcional al área transversal)
- Soporte una carga mínima (relacionada con el momento de inercia)
- Tenga dimensiones físicamente razonables

Formulación:

$$\text{Minimizar: } A = b \cdot h \tag{3.36}$$

$$\text{Sujeto a: } I = \frac{bh^3}{12} \geq I_{min} \tag{3.37}$$

$$0{,}1 \leq b \leq 0{,}5 \text{ m} \tag{3.38}$$

$$0{,}2 \leq h \leq 1{,}0 \text{ m} \tag{3.39}$$

Implemente y resuelva para diferentes valores de I_{min}.

Ejercicio 3.10 (Proyecto integrador)**.** Desarrolle un sistema completo de optimización que:

1. Permita al usuario ingresar:
   - Función objetivo
   - Restricciones (lineales y no lineales)

- Tipo de variables (continuas, enteras, binarias)

2. Realice automáticamente:
   - Clasificación del problema
   - Selección del método apropiado
   - Resolución
   - Visualización de resultados
3. Genere un reporte con:
   - Solución óptima
   - Valores de las restricciones
   - Restricciones activas/inactivas
   - Análisis de sensibilidad
   - Gráficos relevantes

   Implemente en Python con interfaz gráfica (Tkinter o PyQt) o en R con Shiny.

Resumen del Capítulo 3

Conceptos clave:

- Las restricciones modelan limitaciones reales en problemas de optimización
- Se clasifican según igualdad/desigualdad, linealidad, y tipo de variables
- La región factible es el conjunto de soluciones que satisfacen todas las restricciones
- Las restricciones activas determinan la solución óptima
- Python (scipy, pulp) y R (lpSolve, nloptr) ofrecen herramientas potentes para resolver problemas con restricciones

Habilidades desarrolladas:

- Formular problemas reales como problemas de optimización con restricciones
- Clasificar y analizar restricciones
- Implementar verificación y visualización de restricciones
- Resolver problemas lineales y no lineales usando software
- Realizar análisis de sensibilidad
- Interpretar resultados y restricciones activas

Métodos estudiados:

- Programación lineal (método simplex)
- Optimización no lineal con restricciones (SLSQP, SQP)
- Programación entera (branch and bound)
- Análisis gráfico de regiones factibles

Próximo capítulo: Estudiaremos soluciones iterativas para encontrar raíces de ecuaciones, comenzando con métodos básicos de búsqueda.

Lecturas recomendadas

1. Nocedal, J., & Wright, S. J. - *Numerical Optimization*, Capítulos 12-19: Optimización con restricciones

2. Boyd, S., & Vandenberghe, L. - *Convex Optimization*, Capítulo 4: Optimization problems

3. Luenberger, D. G., & Ye, Y. - *Linear and Nonlinear Programming*, Capítulo 2: Basic properties of linear programs

4. Hillier, F. S., & Lieberman, G. J. - *Introduction to Operations Research*, Capítulo 3: Introduction to linear programming

5. Deb, K. - *Optimization for Engineering Design*, Capítulo 2: Classical optimization techniques

Nota Importante

Para profundizar en la implementación computacional, se recomienda explorar:

- **Python**: Documentación de scipy.optimize, PuLP, CVXPY, Pyomo
- **R**: Paquetes lpSolve, nloptr, ROI (R Optimization Infrastructure), ompr
- **Herramientas comerciales**: CPLEX, Gurobi (gratuitos para uso académico)

Capítulo 4

Soluciones Iterativas para Raíces

Objetivos del capítulo:

- Comprender el concepto de raíz de una ecuación
- Implementar métodos iterativos para encontrar raíces
- Comparar eficiencia y convergencia de diferentes métodos

4.1. Introducción a métodos iterativos

Los métodos iterativos generan una sucesión de aproximaciones $x_0, x_1, x_2, \ldots$ que converge hacia la solución exacta x^*. Son esenciales cuando no existe solución analítica.

4.1.1. Criterios de parada

Un algoritmo iterativo debe detenerse cuando:

1. **Error absoluto**: $|x_{n+1} - x_n| < \varepsilon$
2. **Residuo**: $|f(x_n)| < \varepsilon$
3. **Iteraciones máximas**: evitar bucles infinitos

4.2. Concepto de raíz

Definición

Una **raíz** de $f : \mathbb{R} \to \mathbb{R}$ es un valor $x^* \in \mathbb{R}$ tal que $f(x^*) = 0$.

Teorema

Si f es continua en $[a, b]$ y $f(a) \cdot f(b) < 0$, entonces existe al menos un $c \in (a, b)$ tal que $f(c) = 0$.

4.3. Métodos de búsqueda

4.3.1. Método de bisección

El método más robusto, divide el intervalo a la mitad en cada iteración.

Algoritmo:

1. Verificar que $f(a) \cdot f(b) < 0$
2. Calcular punto medio: $c = \frac{a+b}{2}$
3. Si $f(a) \cdot f(c) < 0$, entonces $b \leftarrow c$; sino $a \leftarrow c$
4. Repetir hasta convergencia

Propiedades:

- Convergencia garantizada
- Error después de n iteraciones: $e_n \leq \frac{b-a}{2^n}$
- Convergencia lineal con factor 0.5

Implementación en Python

Listing 4.1: Método de bisección

```
import numpy as np

def biseccion(f, a, b, tol=1e-6, max_iter=100):
    """
    Encuentra una raz usando bisecci n.

    Args:
        f: funci n continua
        a, b: extremos del intervalo
        tol: tolerancia
        max_iter: m ximo de iteraciones

    Returns:
```

```
        ra z aproximada
    """
    if f(a) * f(b) > 0:
        raise ValueError("f(a) y f(b) deben tener signos opuestos")

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

        if abs(fc) < tol or (b - a) / 2 < tol:
            return c

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

    return (a + b) / 2

# Ejemplo
f = lambda x: x**3 - 2*x - 5
raiz = biseccion(f, 2, 3)
print(f"Ra z: {raiz:.8f}, f(ra z) = {f(raiz):.2e}")
```

Implementación en R

Listing 4.2: Método de bisección en R

```
biseccion <- function(f, a, b, tol = 1e-6, max_iter = 100) {
  if (f(a) * f(b) > 0) {
    stop("f(a) y f(b) deben tener signos opuestos")
  }

  for (n in 1:max_iter) {
    c <- (a + b) / 2
    fc <- f(c)

    if (abs(fc) < tol || (b - a) / 2 < tol) {
      return(c)
    }

    if (f(a) * fc < 0) {
```

```
      b <- c
    } else {
      a <- c
    }
  }

  return((a + b) / 2)
}

# Ejemplo
f <- function(x) x^3 - 2*x - 5
raiz <- biseccion(f, 2, 3)
cat(sprintf("Raíz: %.8f, f(raíz) = %.2e\n", raiz, f(raiz)))
```

4.3.2. Método de Newton-Raphson

Usa la derivada para convergencia cuadrática. La fórmula de iteración es:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{4.1}$$

Propiedades:

- Convergencia cuadrática cerca de la raíz
- Requiere calcular la derivada
- Puede diverger si x_0 está mal elegido

Implementación en Python

Listing 4.3: Método de Newton-Raphson

```
def newton_raphson(f, df, x0, tol=1e-6, max_iter=100):
    """
    Encuentra una raíz usando Newton-Raphson.

    Args:
        f: función objetivo
        df: derivada de f
        x0: aproximación inicial
        tol: tolerancia
        máximo de iteraciones

```

```
    Returns:
        ra z aproximada
    """
    x = x0

    for n in range(max_iter):
        fx = f(x)
        dfx = df(x)

        if abs(dfx) < 1e-12:
            raise ValueError("Derivada muy peque a")

        x_new = x - fx / dfx

        if abs(f(x_new)) < tol or abs(x_new - x) < tol:
            return x_new

        x = x_new

    return x

# Ejemplo
f = lambda x: x**3 - 2*x - 5
df = lambda x: 3*x**2 - 2

raiz = newton_raphson(f, df, x0=2.5)
print(f"Ra z: {raiz:.8f}, f(ra z) = {f(raiz):.2e}")
```

Implementación en R

Listing 4.4: Método de Newton-Raphson en R

```
newton_raphson <- function(f, df, x0, tol = 1e-6, max_iter = 100) {
  x <- x0

  for (n in 1:max_iter) {
    fx <- f(x)
    dfx <- df(x)

    if (abs(dfx) < 1e-12) {
      stop("Derivada muy peque a")
    }
```

```
11
12      x_new <- x - fx / dfx
13
14      if (abs(f(x_new)) < tol || abs(x_new - x) < tol) {
15        return(x_new)
16      }
17
18      x <- x_new
19    }
20
21    return(x)
22  }
23
24  # Ejemplo
25  f <- function(x) x^3 - 2*x - 5
26  df <- function(x) 3*x^2 - 2
27
28  raiz <- newton_raphson(f, df, x0 = 2.5)
29  cat(sprintf("Ra z: %.8f, f(ra z) = %.2e\n", raiz, f(raiz)))
```

4.3.3. Método de la secante

Aproxima la derivada usando dos puntos, evitando calcular $f'(x)$:

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \tag{4.2}$$

Propiedades:

- No requiere derivada
- Convergencia superlineal (orden $\approx 1{,}618$)
- Requiere dos puntos iniciales

Implementación en Python

Listing 4.5: Método de la secante

```
1  def secante(f, x0, x1, tol=1e-6, max_iter=100):
2      """
3      Encuentra una ra z usando el m todo de la secante.
4
5      Args:
```

```
        f: funci n objetivo
        x0, x1: dos aproximaciones iniciales
        tol: tolerancia
        max_iter: m ximo de iteraciones

    Returns:
        ra z aproximada
    """
    fx0 = f(x0)
    fx1 = f(x1)

    for n in range(max_iter):
        if abs(fx1 - fx0) < 1e-12:
            raise ValueError("Denominador muy peque o")

        x_new = x1 - fx1 * (x1 - x0) / (fx1 - fx0)
        fx_new = f(x_new)

        if abs(fx_new) < tol or abs(x_new - x1) < tol:
            return x_new

        x0, fx0 = x1, fx1
        x1, fx1 = x_new, fx_new

    return x1

# Ejemplo
f = lambda x: x**3 - 2*x - 5
raiz = secante(f, 2.5, 2.7)
print(f"Ra z: {raiz:.8f}, f(ra z) = {f(raiz):.2e}")
```

Implementación en R

Listing 4.6: Método de la secante en R

```
secante <- function(f, x0, x1, tol = 1e-6, max_iter = 100) {
  fx0 <- f(x0)
  fx1 <- f(x1)

  for (n in 1:max_iter) {
    if (abs(fx1 - fx0) < 1e-12) {
      stop("Denominador muy peque o")
```

```
    }

    x_new <- x1 - fx1 * (x1 - x0) / (fx1 - fx0)
    fx_new <- f(x_new)

    if (abs(fx_new) < tol || abs(x_new - x1) < tol) {
      return(x_new)
    }

    x0 <- x1
    fx0 <- fx1
    x1 <- x_new
    fx1 <- fx_new
  }

  return(x1)
}

# Ejemplo
f <- function(x) x^3 - 2*x - 5
raiz <- secante(f, 2.5, 2.7)
cat(sprintf("Raíz: %.8f, f(raíz) = %.2e\n", raiz, f(raiz)))
```

4.3.4. Comparación de métodos

Método	Convergencia	Requiere f'	Puntos iniciales
Bisección	Lineal (0.5)	No	2 (intervalo)
Newton-Raphson	Cuadrática	Sí	1
Secante	$\approx 1{,}618$	No	2

Tabla 4.1: Comparación de métodos de búsqueda de raíces

Ejemplo comparativo

Listing 4.7: Comparación de métodos

```
import matplotlib.pyplot as plt

def comparar_metodos(f, df, intervalo, x0):
    """Compara los tres métodos."""

```

```
    # Bisecci n
    raiz_bis = biseccion(f, intervalo[0], intervalo[1])

    # Newton-Raphson
    raiz_newton = newton_raphson(f, df, x0)

    # Secante
    raiz_sec = secante(f, x0, x0 + 0.1)

    print("="*50)
    print("COMPARACI N DE M TODOS")
    print("="*50)
    print(f"Bisecci n:       {raiz_bis:.10f}")
    print(f"Newton-Raphson:  {raiz_newton:.10f}")
    print(f"Secante:         {raiz_sec:.10f}")
    print("="*50)

    # Visualizaci n
    x = np.linspace(intervalo[0], intervalo[1], 1000)
    y = f(x)

    plt.figure(figsize=(10, 6))
    plt.plot(x, y, 'b-', linewidth=2, label='f(x)')
    plt.axhline(y=0, color='k', linestyle='--', alpha=0.3)
    plt.plot(raiz_bis, 0, 'ro', markersize=10, label='Bisecci n')
    plt.plot(raiz_newton, 0, 'gs', markersize=10, label='Newton')
    plt.plot(raiz_sec, 0, 'md', markersize=10, label='Secante')
    plt.xlabel('x', fontsize=12)
    plt.ylabel('f(x)', fontsize=12)
    plt.title('Comparaci n de M todos', fontsize=14, fontweight='
        bold')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()

# Ejemplo
f = lambda x: x**3 - 2*x - 5
df = lambda x: 3*x**2 - 2

comparar_metodos(f, df, (2, 3), 2.5)
```

4.4. Ejercicios propuestos

Ejercicio 4.1. Implemente los tres métodos y encuentre las raíces de:

1. $f(x) = x^3 - x - 2$ en $[1, 2]$
2. $f(x) = e^x - 3x$ (dos raíces)
3. $f(x) = \cos(x) - x$

Compare número de iteraciones y precisión.

Ejercicio 4.2. Implemente un método híbrido que:

1. Comience con bisección (5 iteraciones)
2. Continúe con Newton-Raphson
3. Compare eficiencia con métodos individuales

Ejercicio 4.3. Aplicación: La ecuación de Van der Waals modificada es:

$$\left(P + \frac{a}{V^2}\right)(V - b) = RT \tag{4.3}$$

Para $P = 1$ atm, $T = 300$ K, $a = 3{,}6$, $b = 0{,}04$, $R = 0{,}082$, encuentre el volumen V.

Resumen del Capítulo

Conceptos clave:

- Los métodos iterativos son esenciales para resolver ecuaciones no lineales
- La bisección es robusta pero lenta; Newton es rápido pero requiere derivadas
- La secante es un buen compromiso entre robustez y velocidad
- Siempre verificar la convergencia y el resultado final

Cuándo usar cada método:

- **Bisección**: Cuando necesitas garantía de convergencia
- **Newton**: Cuando tienes derivadas y buen punto inicial
- **Secante**: Cuando no tienes derivadas pero quieres velocidad

Lecturas recomendadas

1. Burden, R. L., & Faires, J. D. - *Numerical Analysis*, Cap. 2

2. Chapra, S. C., & Canale, R. P. - *Numerical Methods for Engineers*, Cap. 5-6

3. Press, W. H., et al. - *Numerical Recipes*, Cap. 9

Capítulo 5

Metodos Numericos para Encontrar Raices

Objetivos del capitulo: Al concluir este capitulo, el lector sera capaz de:

- Comprender la importancia de los metodos numericos para resolver ecuaciones no lineales
- Dominar cinco metodos fundamentales: biseccion, secante, punto fijo, regula falsi y Muller
- Analizar y comparar la convergencia de diferentes metodos
- Implementar algoritmos eficientes en Python
- Aplicar metodos numericos a problemas reales en ingenieria y ciencia

5.1. Introduccion

5.1.1. El Problema Fundamental

En matematicas aplicadas y computacion cientifica, frecuentemente encontramos ecuaciones de la forma:

$$f(x) = 0$$

donde f es una funcion continua. Estas ecuaciones aparecen en:

- **Ingenieria**: Calculo de estructuras, dinamica de fluidos
- **Fisica**: Ecuaciones de movimiento, mecanica cuantica

- **Economia**: Optimizacion, modelos de equilibrio
- **Medicina**: Modelos farmacocineticos, diagnostico

El Desafio Computacional:

- Solo pocas ecuaciones tienen solucion analitica exacta
- Las funciones del mundo real son complejas y no lineales
- Se necesita aproximacion numerica con precision controlada
- Diferentes problemas requieren diferentes metodos

5.2. Metodo de Biseccion

5.2.1. Teoria Fundamentada

Teorema del Valor Intermedio: Sea $f : [a, b] \to \mathbb{R}$ continua. Si $f(a) \cdot f(b) < 0$, entonces existe al menos un $c \in (a, b)$ tal que $f(c) = 0$.

Este teorema es la base del metodo de biseccion, que sigue el siguiente algoritmo:

5.2.2. Algoritmo Paso a Paso

1. **Entrada**: f continua, $[a, b]$ con $f(a) \cdot f(b) < 0$, $\epsilon > 0$
2. **Inicializar**: $n \leftarrow 0$
3. **Repetir mientras** $b - a > \epsilon$:
   a) Calcular punto medio: $c \leftarrow \frac{a+b}{2}$
   b) Si $f(c) = 0$: retornar c
   c) Si $f(a) \cdot f(c) < 0$: $b \leftarrow c$
   d) Si $f(b) \cdot f(c) < 0$: $a \leftarrow c$
4. **Retornar** c como aproximacion de la raiz

5.2.3. Analisis de Convergencia

$$|r - c_n| \leq \frac{b-a}{2^{n+1}}$$

Ejemplo: Calculo de Iteraciones: Para alcanzar precision $\epsilon = 10^{-6}$ con intervalo inicial de longitud 1:

$$n \geq \log_2\left(\frac{1}{10^{-6}}\right) - 1 \approx 19{,}93 \Rightarrow n = 20$$

Cada iteracion reduce el error a la mitad.

5.2.4. Implementacion Practica en Python

Listing 5.1: Implementacion del metodo de biseccion

```
import numpy as np

def biseccion(f, a, b, tol=1e-6, max_iter=100):
    """
    Metodo de biseccion para encontrar raices.

    Parametros:
    -----------
    f : funcion continua
    a, b : extremos del intervalo con f(a)*f(b) < 0
    tol : tolerancia
    max_iter : maximo de iteraciones

    Retorna:
    --------
    tuple : (raiz, iteraciones, historial)
    """
    if f(a) * f(b) >= 0:
        raise ValueError("f(a) y f(b) deben tener signos opuestos")

    historial = []

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

```
26
27          historial.append({
28              'iteracion': n + 1,
29              'a': a, 'b': b, 'c': c,
30              'f(c)': fc,
31              'error': (b - a) / 2
32          })
33
34          if abs(fc) < tol or (b - a) / 2 < tol:
35              break
36
37          if f(a) * fc < 0:
38              b = c
39          else:
40              a = c
41
42      return c, len(historial), historial
43
44  # Ejemplo de uso
45  def ejemplo_funcion(x):
46      """f(x) = x^3 - 2*x - 5"""
47      return x**3 - 2*x - 5
48
49  # Aplicar biseccion
50  raiz, iteraciones, hist = biseccion(
51      ejemplo_funcion,
52      a=2.0,
53      b=3.0,
54      tol=1e-8
55  )
56
57  print(f"Raiz encontrada: {raiz:.8f}")
58  print(f"Iteraciones necesarias: {iteraciones}")
59  print(f"f(raiz) = {ejemplo_funcion(raiz):.2e}")
```

5.3. Metodo de la Secante

5.3.1. Fundamentacion Matematica

El metodo de la secante aproxima la derivada usando diferencias finitas:

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

Orden de Convergencia: El metodo de la secante tiene orden de convergencia:

$$p = \frac{1+\sqrt{5}}{2} \approx 1{,}618 \quad \text{(el numero aureo)}$$

Mas rapido que biseccion, mas lento que Newton.

5.3.2. Algoritmo

1. **Entrada**: f continua, x_0, x_1, $\epsilon > 0$
2. **Inicializar**: $n \leftarrow 1$
3. **Repetir mientras** $|f(x_n)| > \epsilon$:
   a) Si $f(x_n) = f(x_{n-1})$: error (division por cero)
   b) Calcular: $x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$
   c) $n \leftarrow n + 1$
4. **Retornar** x_n

5.3.3. Implementacion con Manejo de Errores

Listing 5.2: Metodo de la secante robusto

```
def metodo_secante(f, x0, x1, tol=1e-6, max_iter=100):
    """
    Metodo de la secante para encontrar raices.

    Parametros:
    -----------
    f : funcion
    x0, x1 : dos aproximaciones iniciales
    tol : tolerancia
    max_iter : maximo de iteraciones

    Retorna:
    --------
```

```
    tuple : (raiz, iteraciones, historial)
    """
    historial = []
    fx0, fx1 = f(x0), f(x1)

    for n in range(max_iter):
        if abs(fx1 - fx0) < 1e-15:
            raise ValueError("Diferencia de valores de funcion muy
               pequena")

        x2 = x1 - fx1 * (x1 - x0) / (fx1 - fx0)
        fx2 = f(x2)

        historial.append({
            'iteracion': n + 1,
            'x0': x0, 'x1': x1, 'x2': x2,
            'f(x0)': fx0, 'f(x1)': fx1, 'f(x2)': fx2,
            'error': abs(fx2)
        })

        if abs(fx2) < tol:
            return x2, n + 1, historial

        x0, fx0 = x1, fx1
        x1, fx1 = x2, fx2

    return x1, max_iter, historial

# Ejemplo: Ecuacion de van der Waals
def van_der_waals(v, P=10, T=300, a=1.36, b=0.0318, R=0.0821):
    """Ecuacion de van der Waals: (P + a/v^2)(v - b) = R*T"""
    return (P + a/(v**2)) * (v - b) - R*T

# Encontrar volumen especifico
volumen, iters, hist = metodo_secante(
    lambda v: van_der_waals(v, P=10, T=300),
    x0=2.0,
    x1=3.0,
    tol=1e-8
)

```

```
print(f"Volumen especifico: {volumen:.4f} L/mol")
print(f"Iteraciones: {iters}")
print(f"Error residual: {van_der_waals(volumen):.2e}")
```

5.4. Metodo de Regula Falsi (Falsa Posicion)

5.4.1. Fundamentos Teoricos

El metodo de Regula Falsi combina la robustez de biseccion con la velocidad de interpolacion lineal:

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

Convergencia de Regula Falsi: El metodo siempre converge cuando f es continua en $[a, b]$ y $f(a) \cdot f(b) < 0$. La convergencia es lineal pero generalmente mas rapida que biseccion.

5.4.2. Implementacion Completa

Listing 5.3: Metodo de Regula Falsi mejorado

```
def regula_falsi_mejorado(f, a, b, tol=1e-6, max_iter=100):
    """
    Metodo de Regula Falsi mejorado.

    Mejoras:
    - Deteccion de estancamiento
    - Modificacion de valores de funcion
    """
    if f(a) * f(b) >= 0:
        raise ValueError("f(a) y f(b) deben tener signos opuestos")

    historial = []
    fa, fb = f(a), f(b)

    contador_a = 0
    contador_b = 0

    for n in range(max_iter):
```

```
        c = (a * fb - b * fa) / (fb - fa)
        fc = f(c)

        historial.append({
            'iteracion': n + 1,
            'a': a, 'b': b, 'c': c,
            'f(a)': fa, 'f(b)': fb, 'f(c)': fc,
            'error': abs(fc)
        })

        if abs(fc) < tol or abs(b - a) < tol:
            break

        if fa * fc < 0:
            b = c
            fb = fc
            contador_b = 0
            contador_a += 1

            if contador_a > 5:
                fa = fa / 2
                contador_a = 0
        else:
            a = c
            fa = fc
            contador_a = 0
            contador_b += 1

            if contador_b > 5:
                fb = fb / 2
                contador_b = 0

    return c, len(historial), historial

# Ejemplo: Ecuacion en ingenieria quimica
def ecuacion_equilibrio(x):
    """Ecuacion de equilibrio para reaccion quimica"""
    K = 1.5
    return x**2 / ((1 - x)**2) - K

conversion, iters, hist = regula_falsi_mejorado(
```

```
    ecuacion_equilibrio,
    a=0.1,
    b=0.9,
    tol=1e-8
)

print(f"Conversion de equilibrio: {conversion:.3%}")
print(f"Iteraciones: {iters}")
print(f"Error: {ecuacion_equilibrio(conversion):.2e}")
```

5.5. Metodo de Punto Fijo

5.5.1. Teoria de Punto Fijo

Dado $f(x) = 0$, transformamos a $x = g(x)$.

Teorema de Punto Fijo (Banach): Si $g : [a, b] \to [a, b]$ es contractiva, es decir, existe $L < 1$ tal que:

$$|g(x) - g(y)| \leq L|x - y| \quad \forall x, y \in [a, b]$$

entonces existe un unico punto fijo x^* y la iteracion $x_{n+1} = g(x_n)$ converge a x^*.

5.5.2. Transformaciones Comunes

Para $f(x) = 0$, algunas transformaciones a $x = g(x)$:

$$\begin{aligned} g_1(x) &= x - f(x) \\ g_2(x) &= x - k \cdot f(x) \quad \text{(relajacion)} \\ g_3(x) &= x - \frac{f(x)}{M} \quad \text{con } M \approx f'(x) \\ g_4(x) &= \frac{1}{2}\left(x + \frac{N}{x}\right) \quad \text{para } \sqrt{N} \end{aligned}$$

5.5.3. Implementacion con Analisis de Convergencia

Listing 5.4: Metodo de punto fijo con analisis de convergencia

```
def punto_fijo(g, x0, tol=1e-6, max_iter=100, analizar=True):
    """
```

```
    Metodo de punto fijo con analisis de convergencia.

    Parametros:
    -----------
    g : funcion de iteracion
    x0 : aproximacion inicial
    tol : tolerancia
    max_iter : maximo de iteraciones
    analizar : si True, analiza condiciones de convergencia
    """
    if analizar:
        L = estimar_constante_lipschitz(g, x0-1, x0+1)
        print(f"Constante de Lipschitz estimada: L = {L:.3f}")
        print(f"Convergencia garantizada: {L < 1}")

    historial = []
    x = x0

    for n in range(max_iter):
        x_nuevo = g(x)
        error = abs(x_nuevo - x)

        historial.append({
            'iteracion': n + 1,
            'x': x,
            'g(x)': x_nuevo,
            'error': error
        })

        if error < tol:
            return x_nuevo, n + 1, historial

        x = x_nuevo

    return x, max_iter, historial

def estimar_constante_lipschitz(g, a, b, n_puntos=100):
    """Estima la constante de Lipschitz de g en [a, b]."""
    puntos = np.linspace(a, b, n_puntos)
    valores = [g(x) for x in puntos]

```

```
    max_razon = 0
    for i in range(len(puntos)-1):
        for j in range(i+1, len(puntos)):
            if abs(puntos[j] - puntos[i]) > 1e-10:
                razon = abs(valores[j] - valores[i]) / abs(puntos[j
                   ] - puntos[i])
                max_razon = max(max_razon, razon)

    return max_razon

# Ejemplo 1: Calculo de raiz cuadrada
def g_sqrt(N):
    return lambda x: 0.5 * (x + N / x)

# Calcular sqrt(2)
raiz2, iters1, hist1 = punto_fijo(g_sqrt(2), x0=1.5, tol=1e-12)
print(f"sqrt(2) = {raiz2:.10f}")
print(f"Iteraciones: {iters1}")

# Ejemplo 2: Ecuacion no lineal comun
def g_ejemplo(x):
    return np.cos(x)

raiz, iters2, hist2 = punto_fijo(g_ejemplo, x0=0.5, tol=1e-8)
print(f"Soluci n de cos(x) = x: {raiz:.8f}")
print(f"Iteraciones: {iters2}")
```

Capítulo 6

Diferenciación Numérica y Teoría de Diferencias Finitas

Objetivos del capítulo: Al concluir este capítulo, el lector será capaz de:

- Comprender los fundamentos teóricos de la diferenciación numérica y su importancia en la computación científica moderna
- Dominar las fórmulas de diferencias finitas de diferentes órdenes y precisión
- Analizar, cuantificar y controlar los diferentes tipos de errores en derivación numérica
- Aplicar técnicas de diferenciación numérica a problemas reales en diversas disciplinas
- Implementar algoritmos eficientes y robustos para cálculo de derivadas en diferentes lenguajes de programación
- Diseñar soluciones innovadoras que combinen diferenciación numérica con otras técnicas computacionales

Una Perspectiva Histórica y Filosófica

La diferenciación numérica no es meramente una técnica computacional, sino un puente epistemológico entre el mundo continuo de las matemáticas puras y el mundo discreto de la computación digital. Esta dualidad refleja una de las tensiones más fascinantes de la matemática aplicada contemporánea.

Reflexión Filosófica: ¿Qué significa realmente çalcularüna derivada en un universo digital donde los infinitésimos no existen? Las diferencias finitas nos ofrecen una respuesta pragmática: aproximamos lo infinitamente pequeño con lo finitamente pequeño, lo continuo con lo discreto, el límite teórico con el cociente práctico. Esta aproximación no es una concesión, sino una estrategia inteligente que ha permitido a la humanidad simular fenómenos complejos, desde el clima global hasta los mercados financieros.

6.1. Introducción a la Diferenciación Numérica

6.1.1. El Dilema Fundamental: Continuidad vs Discretización

En el cálculo diferencial clásico, la derivada se define mediante un proceso límite:

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

Sin embargo, en la práctica computacional, nos enfrentamos a limitaciones fundamentales:

Problemas Prácticos Insuperables:

1. **La paradoja del infinitésimo**: Las computadoras no pueden manipular cantidades infinitamente pequeñas
2. **La tiranía de la discretización**: Todo dato medido o simulado es inherentemente discreto
3. **El costo de la precisión**: Mayor precisión requiere más recursos computacionales
4. **La maldición de la dimensionalidad**: En problemas de alta dimensión, las derivadas analíticas son prohibitivas
5. **La realidad del ruido**: Los datos experimentales siempre contienen errores de medición

6.1.2. Ejemplos de la Vida Real donde las Derivadas Analíticas Fallan

Casos Concretos de Aplicación:

- **Simulación de aerodinámica**: El flujo de aire alrededor de un avión se modela con ecuaciones de Navier-Stokes, cuya solución requiere aproximaciones numéricas de derivadas espaciales
- **Predicción del clima**: Los modelos meteorológicos discretizan la atmósfera en millones de celdas y calculan derivadas parciales para predecir la evolución temporal
- **Análisis financiero**: Los algoritmos de trading de alta frecuencia calculan derivadas numéricas de precios para tomar decisiones en milisegundos
- **Diagnóstico médico**: Los equipos de resonancia magnética calculan gradientes de

campo magnético numéricamente para reconstruir imágenes 3D

- **Inteligencia artificial**: Las redes neuronales profundas con millones de parámetros requieren cálculo eficiente de gradientes mediante diferenciación numérica

6.2. Teoría Matemática Detallada

6.2.1. Series de Taylor: La Piedra Angular

El desarrollo de Taylor no es solo una herramienta matemática, sino el lenguaje mediante el cual las diferencias finitas "hablançon las derivadas analíticas.

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(x) + \frac{h^5}{120}f^{(5)}(x) + O(h^6)$$
$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(x) - \frac{h^5}{120}f^{(5)}(x) + O(h^6)$$

6.2.2. Deducción Rigurosa de Fórmulas

Diferencia Hacia Adelante: Una Aproximación de Primer Orden

Partiendo de la serie de Taylor truncada:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(\xi), \quad \xi \in [x, x+h]$$

Despejando $f'(x)$ obtenemos:

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

Diferencia Hacia Adelante:

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

- **Ventajas**: Simple, requiere solo un punto adicional
- **Desventajas**: Error de orden $O(h)$, asimétrico
- **Aplicaciones**: Tiempo real, streaming de datos, sistemas causales

Diferencia Centrada: El Estándar de Oro

Combinando expansiones hacia adelante y atrás:

$$\begin{aligned} f(x+h) - f(x-h) &= [f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \cdots] \\ &\quad - [f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \cdots] \\ &= 2hf'(x) + \frac{h^3}{3}f'''(x) + \cdots \end{aligned}$$

Diferencia Centrada - Óptima Precisión:

$$f'(x) \approx D_0 f(x) = \frac{f(x+h) - f(x-h)}{2h}$$

- **Precisión**: Error $O(h^2)$, un orden mejor que adelante/atrás
- **Simetría**: Usa información equilibrada de ambos lados
- **Estabilidad**: Menor sensibilidad al ruido que métodos asimétricos
- **Aplicaciones**: Análisis de datos completos, procesamiento offline

6.3. Métodos Avanzados de Diferenciación Numérica

6.3.1. Fórmulas de Alta Precisión

Método de Cinco Puntos: Precision Cuasi-Analítica

$$f'(x) \approx \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} + O(h^4)$$

Extrapolación de Richardson: Inteligencia Computacional

Combinando aproximaciones con diferentes h:

$$D_{\text{Richardson}} = \frac{4D(h/2) - D(h)}{3} + O(h^4)$$

Comparación de Métodos - Rendimiento vs Precisión:

Método	Puntos	Orden Error	Costo	Recomendación
Adelante	2	$O(h)$	Bajo	Tiempo real
Centrada	3	$O(h^2)$	Medio	General
5 puntos	5	$O(h^4)$	Alto	Alta precisión
Richardson	Variable	$O(h^4)$	Muy alto	Crítico

6.3.2. Derivadas Parciales y Gradientes

Para funciones multivariables $f(x_1, x_2, \ldots, x_n)$:

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

La matriz Hessiana (derivadas segundas cruzadas):

$$\frac{\partial^2 f}{\partial x_i \partial x_j} \approx \frac{f_{++} - f_{+-} - f_{-+} + f_{--}}{4h^2}$$

6.4. Análisis Exhaustivo de Errores

6.4.1. La Batalla entre Dos Frentes: Truncamiento vs Redondeo

Error de Truncamiento - El Precio de la Aproximación:

$$E_{\text{trunc}} = \left| f'(x) - \frac{f(x+h) - f(x-h)}{2h} \right| \leq \frac{h^2}{6} \max_{\xi} |f'''(\xi)|$$

- **Origen**: Omisión de términos en la serie de Taylor
- **Comportamiento**: $\propto h^2$ para diferencia centrada
- **Minimización**: Reducir h (hasta cierto punto)

Error de Redondeo - La Maldición de la Precisión Finita:

$$E_{\text{redon}} \approx \frac{2\epsilon_{\text{máq}} |f(x)|}{h}$$

donde $\epsilon_{\text{máq}} \approx 2{,}2 \times 10^{-16}$ (precisión doble).

- **Origen**: Representación finita en computadora
- **Comportamiento**: $\propto 1/h$ - ¡se amplifica con h pequeño!

- **Consecuencia**: Cancelación catastrófica para h muy pequeño

6.4.2. El Dilema del h Óptimo: Un Compromiso Fundamental

El error total es la suma de ambos errores:

$$E_{\text{total}} \approx \frac{h^2}{6} M_3 + \frac{2\epsilon |f(x)|}{h}$$

Minimizando respecto a h:

$$\frac{dE_{\text{total}}}{dh} = \frac{h}{3} M_3 - \frac{2\epsilon |f(x)|}{h^2} = 0$$

Solución Óptima - La Fórmula Mágica:

$$h_{\text{opt}} = \left(\frac{6\epsilon |f(x)|}{M_3} \right)^{1/3}$$

Valores típicos:

- Para funciones suaves: $h_{\text{opt}} \approx 10^{-5}$
- Para funciones ruidosas: $h_{\text{opt}} \approx 10^{-3}$
- Para precisión extrema: $h_{\text{opt}} \approx 10^{-7}$

Regla práctica: Empezar con $h = 10^{-6}$ y ajustar según necesidad.

6.4.3. Análisis de Sensibilidad y Condicionamiento

El número de condición para diferenciación numérica:

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

Interpretación del Número de Condición:

- $\kappa \ll 1$: Problema bien condicionado
- $\kappa \approx 1$: Condicionamiento moderado
- $\kappa \gg 1$: Problema mal condicionado - alta sensibilidad al ruido

Ejemplo crítico: Para $f(x) = \sin(x)$ cerca de $x = 0$:

$$\kappa \approx \frac{|\sin(x)|}{h|\cos(x)|} \to \infty \quad \text{cuando } x \to 0$$

6.5. Aplicaciones en el Mundo Real

6.5.1. Ingeniería y Física: Donde la Teoría Encuentra la Realidad

Dinámica de Fluidos Computacional (CFD)

Las ecuaciones de Navier-Stokes discretizadas:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u}$$

Las derivadas espaciales se aproximan con diferencias finitas en mallas de millones de puntos.

Análisis Estructural por Elementos Finitos

Cálculo de tensiones a partir de desplazamientos:

$$\sigma = E\frac{\partial u}{\partial x} \approx E\frac{u_{i+1} - u_{i-1}}{2\Delta x}$$

6.5.2. Finanzas Cuantitativas: Matemáticas en Wall Street

Griegas de las Opciones - El Lenguaje de los Traders:

- **Delta** (Δ): $\frac{\partial V}{\partial S}$ - Sensibilidad al precio del subyacente
- **Gamma** (Γ): $\frac{\partial^2 V}{\partial S^2}$ - Convexidad, riesgo de salto
- **Vega** (ν): $\frac{\partial V}{\partial \sigma}$ - Sensibilidad a la volatilidad
- **Theta** (Θ): $\frac{\partial V}{\partial t}$ - Decaimiento temporal
- **Rho** (ρ): $\frac{\partial V}{\partial r}$ - Sensibilidad a tasas de interés

Todas se calculan numéricamente para opciones exóticas sin fórmula cerrada.

6.5.3. Machine Learning y Ciencia de Datos

Backpropagation en Redes Neuronales

El gradiente de la función de pérdida:

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \cdot \frac{\partial a_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ij}}$$

Donde $z_j = \sum_i w_{ij} a_i$ y $a_j = \sigma(z_j)$.

Verificación de Gradientes - Mejor Prevenir que Lamentar

Antes de entrenar modelos costosos:

$$\frac{\partial L}{\partial w_i} \approx \frac{L(\mathbf{w} + \epsilon \mathbf{e}_i) - L(\mathbf{w} - \epsilon \mathbf{e}_i)}{2\epsilon}$$

Best Practices en ML:

1. Usar $\epsilon \approx 10^{-7}$ para precisión doble
2. Verificar varios pesos aleatorios
3. Comparar error relativo: $\frac{|g_{\text{ana}} - g_{\text{num}}|}{\max(|g_{\text{ana}}|, |g_{\text{num}}|)}$
4. Aceptar errores relativos $< 10^{-7}$

6.5.4. Procesamiento de Señales e Imágenes

Detección de Bordes con Operadores de Sobel

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I$$

Magnitud del gradiente: $G = \sqrt{G_x^2 + G_y^2}$

Análisis de Señales Biomédicas (ECG)

Detección de complejos QRS (latidos cardíacos):

$$y'[n] = \frac{1}{8}(2x[n+1] + x[n] - x[n-1] - 2x[n-2])$$

6.6. Implementación Computacional

6.6.1. Algoritmo en Python con Optimizaciones

```
import numpy as np
from typing import Callable, Union
import warnings

class Differentiator:
    """
    Clase avanzada para diferenciacion num rica con optimizaciones
    y manejo robusto de errores.
    """

    def __init__(self, default_h: float = 1e-5):
        """
        Inicializa el diferenciador.

        Args:
            default_h: Valor por defecto para h
        """
        self.default_h = default_h
        self.epsilon = np.finfo(float).eps

    def derivative(self,
                   f: Callable,
                   x: Union[float, np.ndarray],
                   h: float = None,
                   method: str = 'central',
                   order: int = 1,
                   adaptive: bool = False) -> Union[float, np.
                      ndarray]:
        """
        Calcula derivada numerica con multiples opciones.

        Args:
            f: Funcion a derivar
            x: Punto(s) de evaluacion
            h: Tama o del paso (None para automatico)
            method: 'forward', 'backward', 'central', 'five_point'
            order: Orden de derivada (1-4)
            adaptive: Si True, ajusta h automaticamente
```

```
38
39          Returns:
40              Derivada(s) aproximada(s)
41          """
42          if h is None:
43              h = self._optimal_h(f, x) if adaptive else self.
                    default_h
44
45          h = float(h)
46          if h <= 0:
47              raise ValueError("h debe ser positivo")
48
49          # Convierte a array si es necesario
50          x_array = np.asarray(x)
51          scalar_input = x_array.ndim == 0
52          x_array = np.atleast_1d(x_array)
53
54          result = self._compute_derivative(f, x_array, h, method,
                    order)
55
56          return result[0] if scalar_input else result
57
58      def _optimal_h(self, f: Callable, x: Union[float, np.ndarray])
            -> float:
59          """
60          Estima h optimo usando informacion de la funci n.
61          """
62          # Estimacion simple de la tercera derivada
63          h_test = 1e-4
64          f_val = f(x)
65
66          # Evitar division por cero
67          if abs(f_val) < 1e-15:
68              f_val = 1e-15
69
70          # Formula simplificada para h optimo
71          h_opt = (6 * self.epsilon * abs(f_val)) ** (1/3)
72
73          # Limites razonables
74          h_opt = np.clip(h_opt, 1e-10, 1e-3)
75
```

```
        return float(h_opt)

    def _compute_derivative(self, f: Callable, x: np.ndarray,
                            h: float, method: str, order: int) -> np
                                .ndarray:
        """
        Calcula la derivada usando el metodo especificado.
        """
        if order == 1:
            if method == 'forward':
                return (f(x + h) - f(x)) / h
            elif method == 'backward':
                return (f(x) - f(x - h)) / h
            elif method == 'central':
                return (f(x + h) - f(x - h)) / (2 * h)
            elif method == 'five_point':
                return (-f(x + 2*h) + 8*f(x + h) - 8*f(x - h) + f(x
                    - 2*h)) / (12 * h)
            else:
                raise ValueError(f"Metodo {method} no soportado")

        elif order == 2:
            return (f(x + h) - 2*f(x) + f(x - h)) / (h**2)

        elif order == 3:
            return (f(x + 2*h) - 2*f(x + h) + 2*f(x - h) - f(x - 2*
               h)) / (2 * h**3)

        elif order == 4:
            return (f(x + 2*h) - 4*f(x + h) + 6*f(x) - 4*f(x - h) +
                f(x - 2*h)) / (h**4)

        else:
            raise ValueError(f"Orden {order} no soportado")

    def gradient(self, f: Callable, x: np.ndarray, h: float = None)
        -> np.ndarray:
        """
        Calcula el gradiente de una funci n escalar multivariable.

        Args:
```

```
112             f: Funci n R^n -> R
113             x: Punto en R^n
114             h: Tama o del paso
115 
116         Returns:
117             Gradiente en x
118         """
119         x = np.asarray(x)
120         n = len(x)
121         grad = np.zeros(n)
122 
123         for i in range(n):
124             e_i = np.zeros(n)
125             e_i[i] = 1
126             grad[i] = self.derivative(lambda t: f(x + t * e_i), 0,
                    h=h)
127 
128         return grad
129 
130     def hessian(self, f: Callable, x: np.ndarray, h: float = None)
        -> np.ndarray:
131         """
132         Calcula la matriz Hessiana de una funcion escalar.
133 
134         Args:
135             f: Funcion R^n -> R
136             x: Punto en R^n
137             h: Tama o del paso
138 
139         Returns:
140             Matriz Hessiana n x n
141         """
142         x = np.asarray(x)
143         n = len(x)
144         hess = np.zeros((n, n))
145 
146         for i in range(n):
147             for j in range(i, n):
148                 if i == j:
149                     # Derivada segunda pura
```

```
                    hess[i, i] = self.derivative(f, x, h=h, order
                        =2, method='central')
                else:
                    # Derivada cruzada
                    e_i = np.zeros(n); e_i[i] = 1
                    e_j = np.zeros(n); e_j[j] = 1

                    def f_cross(t):
                        return f(x + t * e_i + t * e_j)

                    hess[i, j] = hess[j, i] = (
                        self.derivative(f_cross, 0, h=h) / (h**2)
                    )

        return hess

# Ejemplo de uso avanzado
if __name__ == "__main__":
    # Crear diferenciador
    diff = Differentiator(default_h=1e-6)

    # Definir funcion compleja
    def complex_function(x):
        return np.sin(x) * np.exp(-0.1*x**2) + 0.5 * np.cos(2*x)

    # Calcular derivada en multiples puntos
    x_points = np.linspace(-5, 5, 1000)
    derivatives = diff.derivative(complex_function, x_points,
                                  method='central', adaptive=True)

    # Calcular gradiente para funcion multivariable
    def rosenbrock(x):
        """Funcion de Rosenbrock, benchmark en optimizacion."""
        return (1 - x[0])**2 + 100 * (x[1] - x[0]**2)**2

    point = np.array([0.5, 0.5])
    grad = diff.gradient(rosenbrock, point)
    hess = diff.hessian(rosenbrock, point)

    print(f"Gradiente en {point}: {grad}")
    print(f"Traza de Hessiana: {np.trace(hess)}")
```

6.7. Estudios de Caso: Del Laboratorio a la Industria

6.7.1. Caso 1: Simulación de un Puente Bajo Carga Dinámica

Escenario Real - Puente Golden Gate:

- **Problema**: Predecir vibraciones bajo viento y tráfico
- **Datos**: Aceleraciones medidas cada 0.01 segundos
- **Derivadas calculadas**:
  1. Velocidad: $v(t) = \frac{dx}{dt} \approx \frac{x_{i+1} - x_{i-1}}{2\Delta t}$
  2. Aceleración: $a(t) = \frac{d^2x}{dt^2} \approx \frac{x_{i+1} - 2x_i + x_{i-1}}{\Delta t^2}$
  3. Jerk (cambio de aceleración): $j(t) = \frac{d^3x}{dt^3}$
- **Resultado**: Identificación de modos de vibración peligrosos

6.7.2. Caso 2: Algoritmo de Trading de Alta Frecuencia

Wall Street en Microsegundos:

- **Problema**: Detectar cambios bruscos en precios
- **Datos**: Precios cada milisegundo
- **Indicadores calculados**:

$$\text{Velocidad del precio} = \frac{dP}{dt}$$

$$\text{Aceleración del precio} = \frac{d^2P}{dt^2}$$

$$\text{Volatilidad instantánea} = \sqrt{\left(\frac{dP}{dt}\right)^2}$$

- **Decisión**: Comprar si aceleración > 0 y velocidad alta
- **Resultado**: Mejora del 15 % en rentabilidad

6.7.3. Caso 3: Diagnóstico de Enfermedades Cardíacas

ECG - Cuando Cada Latido Cuenta:

- **Problema**: Detectar arritmias en tiempo real
- **Datos**: Señal ECG a 360 Hz
- **Características calculadas**:
  1. Pendiente del segmento QRS: $\frac{dV}{dt}_{\max}$
  2. Curvatura en pico R: $\frac{d^2V}{dt^2}$
  3. Variabilidad de intervalo RR: $\frac{d(\text{RR})}{dt}$
- **Algoritmo**:

$$\text{Alerta} = \begin{cases} \text{Verdadero} & \text{si } \left|\frac{d^2V}{dt^2}\right| > \text{umbral} \\ \text{Falso} & \text{en otro caso} \end{cases}$$

- **Resultado**: 99 % de precisión en detección de arritmias

6.8. Ejercicios y Proyectos Integradores

6.8.1. Ejercicios Teóricos con Soluciones

Ejercicio 1: Derivación de Fórmulas

Problema: Derive la fórmula de diferencia centrada de cinco puntos para $f'(x)$.
Solución paso a paso:

$$f(x+2h) = f(x) + 2hf'(x) + 2h^2f''(x) + \frac{4h^3}{3}f'''(x) + \frac{2h^4}{3}f^{(4)}(x) + \frac{4h^5}{15}f^{(5)}(x)$$
$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(x) + \frac{h^5}{120}f^{(5)}(x)$$
$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(x) - \frac{h^5}{120}f^{(5)}(x)$$
$$f(x-2h) = f(x) - 2hf'(x) + 2h^2f''(x) - \frac{4h^3}{3}f'''(x) + \frac{2h^4}{3}f^{(4)}(x) - \frac{4h^5}{15}f^{(5)}(x)$$

Combinación lineal que elimina $f''(x)$ y $f^{(4)}(x)$:

$$-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h) = 12hf'(x) + \frac{2h^5}{5}f^{(5)}(x)$$

$$f'(x) = \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} - \frac{h^4}{30} f^{(5)}(\xi)$$

Ejercicio 2: Análisis de Error Óptimo

Problema: Para $f(x) = e^{\sin(x^2)}$, encuentre h_{opt} en $x = 1$.
Solución:

$$\begin{aligned} f(x) &= e^{\sin(x^2)} \\ f'(x) &= 2x\cos(x^2)e^{\sin(x^2)} \\ f'''(x) &= [\text{expresión compleja}] \end{aligned}$$

Estimación numérica: $M_3 \approx 15{,}2$, $|f(1)| \approx 2{,}319$

$$h_{\text{opt}} \approx \left(\frac{6 \times 2{,}2 \times 10^{-16} \times 2{,}319}{15{,}2}\right)^{1/3} \approx 5{,}6 \times 10^{-6}$$

6.8.2. Proyecto Final: Sistema de Monitoreo de Salud Estructural

Especificaciones del Proyecto: **Objetivo**: Desarrollar un sistema completo para monitoreo estructural en tiempo real.

Componentes:

1. **Adquisición**: Sensores acelerométricos (1000 Hz)
2. **Procesamiento**:
   - Filtrado de ruido (Kalman)
   - Cálculo de velocidad y aceleración (diferencias finitas)
   - Análisis espectral (FFT)
3. **Análisis**:
   - Identificación de modos de vibración
   - Detección de anomalías (cambios en derivadas)
   - Predicción de falla (tendencias temporales)
4. **Visualización**: Dashboard web en tiempo real

5. **Alertas**: SMS/Email cuando $\left|\frac{d^2x}{dt^2}\right| > \text{umbral}$

Tecnologías: Python, NumPy, SciPy, Django, WebSockets

6.9. Tablas de Referencia y Coeficientes

6.9.1. Tablas de Coeficientes Completas

Coeficientes para Primeras Derivadas (divididos por h):

Método	$f(x-3h)$	$f(x-2h)$	$f(x-h)$	$f(x)$	$f(x+h)$	$f(x+2h)$	$f(x+3h)$
Adelante 2 pts	0	0	0	-1	1	0	0
Atrás 2 pts	0	0	-1	1	0	0	0
Centrada 2 pts	0	0	-1/2	0	1/2	0	0
Centrada 4 pts	0	1/12	-2/3	0	2/3	-1/12	0
Centrada 6 pts	-1/60	3/20	-3/4	0	3/4	-3/20	1/60

Coeficientes para Segundas Derivadas (divididos por h^2):

Método	$f(x-3h)$	$f(x-2h)$	$f(x-h)$	$f(x)$	$f(x+h)$	$f(x+2h)$	$f(x+3h)$
Centrada 3 pts	0	0	1	-2	1	0	0
Centrada 5 pts	0	-1/12	4/3	-5/2	4/3	-1/12	0
Centrada 7 pts	1/90	-3/20	3/2	-49/18	3/2	-3/20	1/90

Error por Método y Orden:

Método	**Puntos**	**Orden Error**	**Estabilidad**	**Recomendado para**
Adelante	2	$O(h)$	Baja	Tiempo real simple
Atrás	2	$O(h)$	Baja	Procesamiento causal
Centrada	2	$O(h^2)$	Media	Aplicaciones generales
Centrada 4 pts	4	$O(h^4)$	Alta	Alta precisión
Richardson	Variable	$O(h^4)$	Muy Alta	Crítico
Spectral	Todos	Exponencial	Media	Periódico

6.10. Consideraciones Avanzadas y Tendencias Futuras

6.10.1. Límites Fundamentales de la Diferenciación Numérica

Principio de Incertidumbre Computacional: Así como en física cuántica existe el principio de incertidumbre de Heisenberg, en diferenciación numérica existe una relación fundamental:

$$\Delta(\text{Precisión}) \times \Delta(\text{Estabilidad}) \geq \text{Constante}$$

- **Alta precisión** (h pequeño) ⇒ **Baja estabilidad** (amplificación de ruido)
- **Alta estabilidad** (h grande) ⇒ **Baja precisión** (error de truncamiento)
- **Solución**: Encontrar el punto de equilibrio óptimo h_{opt}

6.10.2. Técnicas Modernas y Alternativas

Diferenciación Automática (Autodiff)

Autodiff vs Diferencias Finitas - Revolución en ML:

Característica	Dif. Finitas	Autodiff
Precisión	$O(h^2)$	Exacta (máquina)
Costo por derivada	$O(n)$ evaluaciones	$O(1)$ después de tracing
Memoria	Constante	Lineal en profundidad
Implementación	Simple	Compleja
Aplicación	General	ML/Optimización
Librerías	Manual	TensorFlow, PyTorch, JAX

Métodos Especializados para Datos Ruidosos

Regularización y Suavizado - Cuando los Datos Mienten:

- **Filtrado de Savitzky-Golay**: Ajuste polinomial local antes de derivar

$$\hat{f}'(x_i) = \frac{\sum_{j=-m}^{m} c_j f(x_{i+j})}{h}$$

- **Diferenciación espectral**: Para datos periódicos usando FFT

$$\mathcal{F}\{f'(x)\} = i\omega\mathcal{F}\{f(x)\}$$

- **Métodos de regularización de Tikhonov**: Minimización con penalización

$$\min_{f'} \|Df - f'\|^2 + \lambda\|Lf'\|^2$$

6.10.3. Tendencias Futuras e Investigación

Fronteras de la Investigación - ¿Qué Viene?:

- **Diferenciación cuántica**: Uso de computadoras cuánticas para cálculo masivo de gradientes
- **Aprendizaje automático de derivadas**: Redes neuronales que aprenden a calcular derivadas óptimas
- **Diferenciación en grafos**: Para datos no estructurados y redes complejas
- **Métodos adaptativos en tiempo real**: Ajuste dinámico de h según características locales
- **Fusión con sensores inteligentes**: Cálculo de derivadas en el edge computing
- **Aplicaciones en medicina personalizada**: Derivadas de señales biomédicas para diagnóstico preciso

6.11. Recomendaciones Prácticas y Mejores Prácticas

Decálogo del Buen Diferenciador Numérico:

1. **Conoce tus datos**: Analiza ruido, periodicidad, suavidad antes de elegir método
2. **Empezar simple**: Usa diferencia centrada como baseline antes de métodos complejos
3. **Validar siempre**: Compara con solución analítica cuando sea posible
4. **Probar múltiples** h: Grafica error vs h para encontrar región óptima
5. **Considerar simetría**: Diferencia centrada es generalmente mejor que adelante/atrás
6. **Adaptar a la aplicación**: Tiempo real vs offline, precisión vs velocidad

7. **Manejar casos especiales**: Bordes, datos faltantes, ruido excesivo
8. **Documentar supuestos**: Especificar método, h, orden de error
9. **Optimizar computacionalmente**: Reutilizar evaluaciones, usar vectorización
10. **Pensar en escalabilidad**: Cómo se comporta el método con grandes volúmenes de datos

Selección de Método por Aplicación:

Aplicación	Método Recomendado	Justificación
Procesamiento en tiempo real	Adelante/atrás	Causal, simple, rápido
Análisis de datos completos	Centrada	Precisión balanceada
Alta precisión científica	5 puntos o Richardson	Error mínimo
Señales periódicas	Espectral	Precisión exponencial
Datos ruidosos	Savitzky-Golay	Robustez al ruido
ML/optimización	Autodiff	Precisión exacta

6.12. Conclusión: El Arte y la Ciencia de la Diferenciación Numérica

Reflexión Final - Más que una Técnica, una Filosofía: La diferenciación numérica no es meramente un conjunto de fórmulas o algoritmos. Es una manifestación concreta de cómo la humanidad aborda problemas complejos mediante aproximaciones inteligentes. Cada vez que usamos diferencias finitas, estamos:

- **Aceptando límites**: Reconocemos que no podemos calcular derivadas exactas en el mundo digital
- **Buscando equilibrio**: Encontramos el punto óptimo entre precisión y estabilidad
- **Adaptándonos**: Elegimos el método apropiado para cada contexto y necesidad
- **Creando puentes**: Conectamos matemáticas continuas con computación discreta
- **Innovando**: Desarrollamos nuevas técnicas para nuevos desafíos

En un mundo cada vez más dominado por datos discretos, la habilidad de calcular derivadas de manera robusta y eficiente se ha convertido en una competencia fundamental para científicos, ingenieros, analistas y todos aquellos que buscan extraer significado de la información numérica.

Bibliografía Ampliada y Recursos para Profundizar

Lecturas Esenciales:

1. **Burden, R. L., & Faires, J. D.** (2011). *Numerical Analysis* (9ª ed.). Cengage Learning. *(Capítulo 4: Diferenciación Numérica)*
2. **Chapra, S. C., & Canale, R. P.** (2015). *Numerical Methods for Engineers* (7ª ed.). McGraw-Hill. *(Capítulo 23: Diferenciación Numérica)*
3. **Press, W. H., et al.** (2007). *Numerical Recipes: The Art of Scientific Computing* (3ª ed.). Cambridge University Press. *(Capítulo 5: Evaluación de Funciones)*
4. **Higham, N. J.** (2002). *Accuracy and Stability of Numerical Algorithms* (2ª ed.). SIAM. *(Capítulo 8: Diferenciación Numérica)*
5. **Fornberg, B.** (1988). *Generation of Finite Difference Formulas on Arbitrarily Spaced Grids*. Mathematics of Computation.
6. **Baydin, A. G., et al.** (2018). *Automatic Differentiation in Machine Learning: A Survey*. Journal of Machine Learning Research.

Recursos en Línea y Software:

- **Librerías Python**:
  - NumPy: `numpy.gradient()`, `numpy.diff()`
  - SciPy: `scipy.misc.derivative()`, `scipy.ndimage.gaussian_filter1d()`
  - Autograd / JAX: Diferenciación automática
- **Librerías MATLAB**:
  - `gradient()`, `diff()`, `del2()` (Laplaciano)
  - Toolbox de Procesamiento de Señales

- **Tutoriales y Cursos**:
  - MIT OpenCourseWare: Ïntroduction to Numerical Methods"
  - Coursera: "Numerical Methods for Engineers"
  - Khan Academy: Cálculo y métodos numéricos
- **Comunidades**:
  - Stack Overflow: Etiquetas `numerical-methods`, `finite-difference`
  - GitHub: Repositorios con implementaciones avanzadas
  - ResearchGate: Discusiones académicas

Proyectos de Código Abierto para Estudiar:

- **FINUFFT**: Fast Nonuniform FFT (diferenciación espectral)
- **DiffSharp**: Librería de diferenciación automática en F#
- **ADOL-C**: Automatic Differentiation by OverLoading in C++
- **LibTorch**: Versión C++ de PyTorch para autodiff
- **Sundials**: Suite para ecuaciones diferenciales (incluye diferencias finitas)

Capítulo 7

Interpolación Numérica

7.1. Introducción

En análisis numérico y computación científica, frecuentemente nos encontramos con situaciones donde conocemos valores de una función solo en puntos discretos, pero necesitamos estimar su valor en puntos intermedios. Este problema surge en innumerables aplicaciones: desde procesamiento de señales hasta análisis de datos experimentales, gráficos por computadora y simulaciones numéricas.

La **interpolación** es el proceso de construir una función que pase exactamente por un conjunto dado de puntos conocidos. A diferencia de la aproximación, que busca una función cercana a los puntos pero no necesariamente que pase por ellos, la interpolación exige coincidencia exacta en los nodos dados.

7.2. Formulación Matemática del Problema

7.2.1. Definición Formal

Dado un conjunto de $n+1$ pares ordenados $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, donde los x_i son valores distintos en el dominio (llamados **nodos de interpolación**) y normalmente están ordenados ($x_0 < x_1 < \ldots < x_n$), el problema de interpolación consiste en encontrar una función $f(x)$ perteneciente a una clase específica (polinomios, splines, funciones trigonométricas, etc.) tal que:

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

El conjunto $\{x_i\}_{i=0}^{n}$ se denomina **nodos de interpolación** y los valores correspondientes $\{y_i\}_{i=0}^{n}$ son los **valores de la función** en esos nodos. La función $f(x)$ que satisface $f(x_i) = y_i$ para todo i se llama **función interpolante**.

7.2.2. Existencia y Unicidad

Para la clase de polinomios de grado máximo n, existe un resultado fundamental:

Existencia y unicidad del polinomio interpolante: Dados $n+1$ puntos distintos $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, existe un único polinomio $P_n(x)$ de grado a lo más n tal que $P_n(x_i) = y_i$ para $i = 0, 1, \ldots, n$.

7.3. Interpolación Lineal

7.3.1. Método y Fórmula

La interpolación lineal es el método más simple y consiste en conectar dos puntos adyacentes con una línea recta. Para un valor x en el intervalo $[x_i, x_{i+1}]$, el valor interpolado se calcula como:

$$P_1(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i)$$

Esta fórmula representa la ecuación de la recta que pasa por los puntos (x_i, y_i) y (x_{i+1}, y_{i+1}).

Estimación de temperatura: Si a las 8:00 AM la temperatura es 20°C y a las 12:00 PM es 28°C, estimar la temperatura a las 10:00 AM usando interpolación lineal.

Solución: Identificamos: $x_0 = 8$, $y_0 = 20$, $x_1 = 12$, $y_1 = 28$, $x = 10$.

$$T(10) = 20 + \frac{28 - 20}{12 - 8}(10 - 8) = 20 + \frac{8}{4} \times 2 = 20 + 4 = 24C$$

7.3.2. Análisis del Error

Si la función original $f(x)$ es dos veces diferenciable en el intervalo $[x_i, x_{i+1}]$, el error de interpolación lineal está acotado por:

$$|f(x) - P_1(x)| \leq \frac{M_2}{8}(x_{i+1} - x_i)^2$$

donde $M_2 = \max_{\xi \in [x_i, x_{i+1}]} |f''(\xi)|$.

Esta fórmula muestra que el error es proporcional al cuadrado de la distancia entre nodos, lo que justifica usar nodos más cercanos para mayor precisión.

7.4. Interpolación Polinomial

7.4.1. Forma de Lagrange

Para $n+1$ puntos, el polinomio interpolante de Lagrange de grado n se expresa como:

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

donde los polinomios base de Lagrange $L_i(x)$ están definidos por:

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

Los polinomios $L_i(x)$ tienen la propiedad fundamental:

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

Esta propiedad garantiza que $P_n(x_i) = y_i$ para cada nodo.

Número de Puntos	**Polinomios Base**	**Grado Máximo**
2	$L_0(x) = \frac{x-x_1}{x_0-x_1}$, $L_1(x) = \frac{x-x_0}{x_1-x_0}$	1
3	$L_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}$, etc.	2
4	$L_0(x) = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)}$, etc.	3

Tabla 7.1: Polinomios base de Lagrange para diferentes números de puntos

Interpolación cuadrática: Encontrar el polinomio que pasa por $(-1, 2)$, $(0, 1)$, $(2, 7)$.
Solución:

$$\begin{aligned} L_0(x) &= \frac{(x-0)(x-2)}{(-1-0)(-1-2)} = \frac{x(x-2)}{3} \\ L_1(x) &= \frac{(x+1)(x-2)}{(0+1)(0-2)} = -\frac{(x+1)(x-2)}{2} \\ L_2(x) &= \frac{(x+1)(x-0)}{(2+1)(2-0)} = \frac{(x+1)x}{6} \\ P_2(x) &= 2L_0(x) + 1L_1(x) + 7L_2(x) = x^2 + 2x + 1 \end{aligned}$$

7.4.2. Forma de Newton con Diferencias Divididas

La forma de Newton es computacionalmente más eficiente que la de Lagrange, especialmente cuando se necesitan agregar nuevos puntos. El polinomio se expresa como:

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

donde las diferencias divididas se definen recursivamente:

$$
\begin{aligned}
f[x_i] &= y_i \\
f[x_i, x_{i+1}] &= \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i} \\
f[x_i, x_{i+1}, x_{i+2}] &= \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i} \\
&\vdots \\
f[x_i, \ldots, x_{i+k}] &= \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}
\end{aligned}
$$

x_i	$f[x_i]$	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$
0	1			
		2		
1	3		1	
		4		0
2	7		1	
		6		
3	13			

Tabla 7.2: Ejemplo de tabla de diferencias divididas para los puntos (0,1), (1,3), (2,7), (3,13)

7.5. Splines Cúbicos

7.5.1. Concepto y Motivación

Cuando se interpolan muchos puntos con un solo polinomio de alto grado, pueden aparecer oscilaciones no deseadas (fenómeno de Runge). Los **splines** ofrecen una solución: en lugar de usar un polinomio global, se usan polinomios de bajo grado por tramos, unidos con condiciones de suavidad.

Un **spline cúbico** es una función $S(x)$ definida por partes, donde en cada intervalo $[x_i, x_{i+1}]$, $S(x)$ es un polinomio cúbico, y en los puntos de unión (nodos) se imponen condiciones de continuidad hasta la segunda derivada.

7.5.2. Definición Formal

Para $n+1$ puntos $(x_0, y_0), \ldots, (x_n, y_n)$ con $x_0 < x_1 < \ldots < x_n$, un spline cúbico $S(x)$ consiste de n polinomios cúbicos $S_i(x)$, uno para cada intervalo $[x_i, x_{i+1}]$:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3, \quad x \in [x_i, x_{i+1}]$$

7.5.3. Condiciones de Suavidad

Para garantizar suavidad, se imponen las siguientes condiciones:

1. **Interpolación**: $S_i(x_i) = y_i$ y $S_i(x_{i+1}) = y_{i+1}$ para $i = 0, \ldots, n-1$ ($2n$ condiciones)
2. **Continuidad de primera derivada**: $S'_{i-1}(x_i) = S'_i(x_i)$ para $i = 1, \ldots, n-1$ ($n-1$ condiciones)
3. **Continuidad de segunda derivada**: $S''_{i-1}(x_i) = S''_i(x_i)$ para $i = 1, \ldots, n-1$ ($n-1$ condiciones)
4. **Condiciones de frontera**: 2 condiciones adicionales (natural, fija o periódica)

En total hay $4n$ coeficientes y $4n-2$ condiciones de las primeras tres categorías, por lo que se necesitan 2 condiciones adicionales para determinar unívocamente el spline.

7.5.4. Sistema de Ecuaciones para Splines Naturales

Para splines cúbicos naturales (segunda derivada cero en los extremos), el sistema para determinar los coeficientes c_i es tridiagonal:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \cdots & 0 \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{3}{h_1}(y_2-y_1) - \frac{3}{h_0}(y_1-y_0) \\ \frac{3}{h_2}(y_3-y_2) - \frac{3}{h_1}(y_2-y_1) \\ \vdots \\ 0 \end{bmatrix}$$

donde $h_i = x_{i+1} - x_i$.

7.6. Análisis de Errores

7.6.1. Error en Interpolación Polinomial

Para un polinomio interpolante $P_n(x)$ de grado n que interpola $f(x)$ en los nodos $x_0, x_1, \ldots, x_n$, el error en cualquier punto x está dado por:

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

donde ξ es algún punto en el intervalo que contiene a x y todos los x_i.

Esta fórmula es fundamental porque:

- Muestra que el error depende de la derivada $(n+1)$-ésima de f

- El término $\prod_{i=0}^{n}(x - x_i)$ depende de la distribución de los nodos
- Para funciones suaves (derivadas acotadas), el error decrece rápidamente con n

7.6.2. Fenómeno de Runge

Un resultado contraintuitivo descubierto por Carl Runge en 1901 muestra que para ciertas funciones, aumentar el grado del polinomio interpolante con nodos equiespaciados puede *empeorar* la aproximación.

Para la función $f(x) = \frac{1}{1+25x^2}$ en el intervalo $[-1, 1]$, se tiene:

$$\lim_{n\to\infty} \max_{x\in[-1,1]} |f(x) - P_n(x)| = \infty$$

donde $P_n(x)$ es el polinomio interpolante de grado n con nodos equiespaciados.

7.6.3. Nodos de Chebyshev

Para minimizar el error máximo en interpolación polinomial, se recomienda usar los nodos de Chebyshev:

$$x_k = \cos\left(\frac{2k+1}{2(n+1)}\pi\right), \quad k = 0, 1, \ldots, n$$

Estos nodos no están equiespaciados: están más concentrados cerca de los extremos del intervalo $[-1, 1]$. Para un intervalo general $[a, b]$, se transforman linealmente:

$$x_k = \frac{a+b}{2} + \frac{b-a}{2}\cos\left(\frac{2k+1}{2(n+1)}\pi\right)$$

7.7. Comparación de Métodos

Método	**Suavidad**	**Costo Computacional**	**Estabilidad**	**Flexibilidad**	**Aplica**
Lineal	C^0	$O(n)$	Alta	Baja	Visua
Cuadrática	C^0	$O(n^2)$	Media	Media	Aproxi
Lagrange	C^∞	$O(n^2)$	Baja (Runge)	Alta	P
Newton	C^∞	$O(n^2)$	Media	Alta	Ag
Spline cúbico	C^2	$O(n)$	Alta	Alta	Gr
Spline cúbico natural	C^2	$O(n)$	Alta	Media	Inter

Tabla 7.3: Comparación detallada de métodos de interpolación

7.8. Implementación Computacional

7.8.1. Interpolación Lineal en Python

Listing 7.1: Interpolacion lineal por partes

```
import numpy as np
from typing import Union

def interpolacion_lineal(x: np.ndarray, y: np.ndarray,
                         xq: Union[float, np.ndarray]) -> Union[
                            float, np.ndarray]:
    """
    Interpolacion lineal por partes.

    Parametros:
    -----------
    x : np.ndarray
        Coordenadas x de los puntos conocidos (debe estar ordenado)
    y : np.ndarray
        Valores y correspondientes a las coordenadas x
    xq : Union[float, np.ndarray]
        Punto(s) donde se desea interpolar

    Retorna:
    --------
    Union[float, np.ndarray]
        Valores interpolados en xq

    Lanza:
    ------
    ValueError
        Si los puntos x no estan ordenados o no son unicos
    """
    # Convertir a arrays numpy
    x = np.asarray(x)
    y = np.asarray(y)
    xq = np.asarray(xq)

    # Verificar que los puntos x esten ordenados y sean unicos
    if not np.all(np.diff(x) > 0):
```

```
35          raise ValueError("Los puntos x deben estar ordenados y ser
                unicos")
36
37      # Encontrar indices de los intervalos que contienen a xq
38      indices = np.searchsorted(x, xq) - 1
39      indices = np.clip(indices, 0, len(x) - 2)
40
41      # Obtener los extremos de cada intervalo
42      x0 = x[indices]
43      x1 = x[indices + 1]
44      y0 = y[indices]
45      y1 = y[indices + 1]
46
47      # Calcular pendientes, evitando division por cero
48      slopes = np.where(x1 != x0, (y1 - y0) / (x1 - x0), 0)
49
50      # Aplicar formula de interpolacion lineal
51      return y0 + slopes * (xq - x0)
52
53  # Ejemplo de uso
54  if __name__ == "__main__":
55      # Datos de ejemplo: temperatura cada 2 horas
56      horas = np.array([0, 2, 4, 6, 8, 10, 12])
57      temperatura = np.array([15, 16, 18, 22, 25, 26, 27])
58
59      # Consultar temperatura a horas especificas
60      horas_consulta = np.array([1, 3, 5, 7, 9, 11])
61      temp_interpolada = interpolacion_lineal(horas, temperatura,
            horas_consulta)
62
63      print("Hora\t Temperatura interpolada (C)")
64      print("-" * 35)
65      for h, t in zip(horas_consulta, temp_interpolada):
66          print(f"{h:2.0f}\t{t:6.2f}")
```

7.8.2. Interpolacion de Lagrange

Listing 7.2: Interpolacion polinomial de Lagrange

```
1  def lagrange_interpolacion(x: np.ndarray, y: np.ndarray,
2                             xq: Union[float, np.ndarray]) -> Union[
                                  float, np.ndarray]:
```

```
    """
    Interpolacion polinomial de Lagrange.

    Nota: Este metodo solo es recomendable para n pequeno (< 20)
    debido al fenomeno de Runge.

    Parametros:
    -----------
    x : np.ndarray
        Coordenadas x de los puntos conocidos
    y : np.ndarray
        Valores y correspondientes
    xq : Union[float, np.ndarray]
        Punto(s) donde evaluar el polinomio interpolante

    Retorna:
    --------
    Union[float, np.ndarray]
        Valores del polinomio interpolante en xq
    """
    x = np.asarray(x)
    y = np.asarray(y)
    xq = np.asarray(xq)

    n = len(x)
    result = np.zeros_like(xq, dtype=float)

    # Evaluar en cada punto de consulta
    for k, x_val in enumerate(xq):
        total = 0.0
        for i in range(n):
            # Calcular el termino de Lagrange L_i(x_val)
            term = y[i]
            for j in range(n):
                if i != j:
                    term *= (x_val - x[j]) / (x[i] - x[j])
            total += term
        result[k] = total

    return result

```

```
# Ejemplo: Interpolacion cuadratica
if __name__ == "__main__":
    # Tres puntos para polinomio cuadratico
    x_data = np.array([-1, 0, 2])
    y_data = np.array([2, 1, 7])

    # Evaluar el polinomio en multiples puntos
    x_eval = np.linspace(-1.5, 2.5, 50)
    y_eval = lagrange_interpolacion(x_data, y_data, x_eval)

    # Verificar que pasa por los puntos originales
    y_check = lagrange_interpolacion(x_data, y_data, x_data)

    print("Verificacion en puntos originales:")
    print(f"x = {x_data}")
    print(f"y calculado = {y_check}")
    print(f"y original  = {y_data}")
    print(f"Diferencia maxima = {np.max(np.abs(y_check - y_data))
       :.2e}")
```

7.8.3. Splines Cubicos Naturales

Listing 7.3: Implementacion de splines cubicos naturales

```
def spline_cubico_natural(x: np.ndarray, y: np.ndarray,
                          xq: Union[float, np.ndarray]) -> Union[
                             float, np.ndarray]:
    """
    Spline cubico natural (segunda derivada cero en extremos).

    Implementa el algoritmo tridiagonal para calcular coeficientes.

    Parametros:
    -----------
    x : np.ndarray
        Coordenadas x de los puntos conocidos (ordenadas)
    y : np.ndarray
        Valores y correspondientes
    xq : Union[float, np.ndarray]
        Punto(s) donde evaluar el spline

    Retorna:
```

```
    --------
    Union[float, np.ndarray]
        Valores del spline en xq
    """
    x = np.asarray(x)
    y = np.asarray(y)
    xq = np.asarray(xq)

    n = len(x) - 1  # Numero de intervalos

    # Paso 1: Calcular distancias entre nodos
    h = x[1:] - x[:-1]

    # Paso 2: Construir sistema tridiagonal para coeficientes c_i
    A = np.zeros((n+1, n+1))
    b = np.zeros(n+1)

    # Condiciones de frontera naturales
    A[0, 0] = 1
    A[n, n] = 1

    # Ecuaciones internas
    for i in range(1, n):
        A[i, i-1] = h[i-1]
        A[i, i] = 2 * (h[i-1] + h[i])
        A[i, i+1] = h[i]
        b[i] = 3 * ((y[i+1] - y[i]) / h[i] -
                    (y[i] - y[i-1]) / h[i-1])

    # Resolver sistema para coeficientes c
    c = np.linalg.solve(A, b)

    # Calcular coeficientes restantes
    a = y[:-1].copy()
    b_coeff = np.zeros(n)
    d = np.zeros(n)

    for i in range(n):
        b_coeff[i] = (y[i+1] - y[i]) / h[i] - h[i] * (2*c[i] + c[i
           +1]) / 3
        d[i] = (c[i+1] - c[i]) / (3 * h[i])
```

```
58
59      # Evaluar spline en puntos de consulta
60      result = np.zeros_like(xq)
61
62      for k, x_val in enumerate(xq):
63          # Encontrar intervalo correspondiente
64          i = np.searchsorted(x, x_val) - 1
65          i = max(0, min(i, n-1))
66
67          # Evaluar polinomio cubico
68          dx = x_val - x[i]
69          result[k] = a[i] + b_coeff[i]*dx + c[i]*dx**2 + d[i]*dx**3
70
71      return result
72
73  # Ejemplo: Suavizado de datos experimentales
74  if __name__ == "__main__":
75      # Generar datos experimentales con ruido
76      np.random.seed(42)
77      x_experimental = np.linspace(0, 10, 15)
78      y_exact = np.sin(x_experimental) + 0.1*x_experimental
79      y_experimental = y_exact + 0.1*np.random.randn(len(
            x_experimental))
80
81      # Interpolacion con spline cubico
82      x_fino = np.linspace(0, 10, 200)
83      y_spline = spline_cubico_natural(x_experimental, y_experimental
            , x_fino)
84
85      # Calcular error RMS
86      y_exact_fino = np.sin(x_fino) + 0.1*x_fino
87      error_rms = np.sqrt(np.mean((y_spline - y_exact_fino)**2))
88
89      print("Resultados del spline cubico:")
90      print(f"Error RMS: {error_rms:.6f}")
91      print(f"Numero de puntos originales: {len(x_experimental)}")
92      print(f"Numero de puntos interpolados: {len(x_fino)}")
```

7.9. Aplicaciones Prácticas

7.9.1. Procesamiento de Señales Digitales

En procesamiento de señales, la interpolación es fundamental para:

- **Cambio de frecuencia de muestreo (resampling):** Convertir una señal muestreada a una frecuencia diferente.
- **Reconstrucción de señales:** Recuperar la señal continua original a partir de sus muestras discretas.
- **Suavizado de datos:** Eliminar ruido mediante interpolación suave.
- **Interpolación de audio:** En procesamiento de audio digital para cambiar la tasa de muestreo.

7.9.2. Gráficos por Computadora

En gráficos por computadora, se utilizan diversos métodos de interpolación:

- **Escalado de imágenes:**
  - **Vecino más cercano:** Rápido pero produce bordes pixelados.
  - **Interpolación bilineal:** Balance entre calidad y velocidad.
  - **Interpolación bicúbica:** Alta calidad pero más costosa computacionalmente.
- **Animación 3D:**
  - Interpolación de posiciones entre fotogramas clave.
  - Interpolación de rotaciones (usando cuaterniones).
  - Interpolación de parámetros de animación.
- **Curvas de Bézier y B-splines:** Para diseño de fuentes, logotipos y gráficos vectoriales.

7.9.3. Ingeniería y Ciencia

- **Análisis de datos experimentales:** Interpolación entre mediciones discretas para crear curvas continuas.
- **Simulación numérica:** Interpolación de valores en mallas no uniformes en métodos como Elementos Finitos o Volúmenes Finitos.
- **Cartografía digital:** Interpolación de elevaciones entre puntos de medición para crear modelos digitales de terreno.
- **Meteorología:** Interpolación de datos de temperatura, presión y humedad entre estaciones meteorológicas.

7.9.4. Finanzas Cuantitativas

- **Construcción de curvas de rendimiento:** Interpolación de tasas de interés para diferentes plazos.
- **Valoración de opciones:** Interpolación de volatilidades implícitas.
- **Análisis de series temporales:** Completado de datos faltantes en series financieras.

7.10. Ejercicios Resueltos

7.10.1. Ejercicio 1: Error de Interpolación Lineal

Problema: Para la función $f(x) = e^x$ en el intervalo $[0, 1]$, usar interpolación lineal con $x_0 = 0$, $x_1 = 1$ para estimar $f(0{,}25)$ y analizar el error.

Solución:

1. **Valor exacto:** $f(0{,}25) = e^{0{,}25} \approx 1{,}2840254167$
2. **Interpolación lineal:**

$$P_1(0{,}25) = f(0) + \frac{f(1) - f(0)}{1 - 0}(0{,}25 - 0) = 1 + \frac{e - 1}{1} \times 0{,}25 \approx 1 + 0{,}429 \times 0{,}25 = 1{,}10725$$

3. **Error absoluto:** $|1{,}284025 - 1{,}10725| \approx 0{,}176775$
4. **Error relativo:** $\frac{0{,}176775}{1{,}284025} \approx 0{,}1377 = 13{,}77\,\%$
5. **Cota teórica del error:**

$$f''(x) = e^x, \quad M_2 = \max_{x \in [0,1]} |f''(x)| = e^1 = e \approx 2{,}71828$$

$$|f(0{,}25) - P_1(0{,}25)| \leq \frac{M_2}{8}(x_1 - x_0)^2 = \frac{e}{8}(1)^2 \approx 0{,}3398$$

El error real (0.1768) es menor que la cota teórica (0.3398).

7.10.2. Ejercicio 2: Polinomio Interpolante de Lagrange

Problema: Dados los puntos $(0, 1)$, $(1, 0)$, $(2, 3)$, $(3, 10)$, encontrar el polinomio interpolante de Lagrange.

Solución:

1. **Polinomios base de Lagrange:**

$$\begin{aligned}
L_0(x) &= \frac{(x-1)(x-2)(x-3)}{(0-1)(0-2)(0-3)} = \frac{(x-1)(x-2)(x-3)}{-6} \\
L_1(x) &= \frac{(x-0)(x-2)(x-3)}{(1-0)(1-2)(1-3)} = \frac{x(x-2)(x-3)}{2} \\
L_2(x) &= \frac{(x-0)(x-1)(x-3)}{(2-0)(2-1)(2-3)} = \frac{x(x-1)(x-3)}{-2} \\
L_3(x) &= \frac{(x-0)(x-1)(x-2)}{(3-0)(3-1)(3-2)} = \frac{x(x-1)(x-2)}{6}
\end{aligned}$$

2. **Polinomio interpolante:**

$$\begin{aligned}
P_3(x) &= 1 \cdot L_0(x) + 0 \cdot L_1(x) + 3 \cdot L_2(x) + 10 \cdot L_3(x) \\
&= \frac{(x-1)(x-2)(x-3)}{-6} + 3 \cdot \frac{x(x-1)(x-3)}{-2} + 10 \cdot \frac{x(x-1)(x-2)}{6}
\end{aligned}$$

3. **Forma simplificada:** $P_3(x) = x^3 - 2x^2 + 1$

4. **Verificación:**

$$\begin{aligned}
P_3(0) &= 0 - 0 + 1 = 1 \quad \checkmark \\
P_3(1) &= 1 - 2 + 1 = 0 \quad \checkmark \\
P_3(2) &= 8 - 8 + 1 = 1 \neq 3 \quad \text{¡Error!}
\end{aligned}$$

Revisando los cálculos, el polinomio correcto es $P_3(x) = x^3 - 2x^2 + 2x + 1$.

7.10.3. Ejercicio 3: Spline Cúbico

Problema: Para los puntos $(0, 1)$, $(1, 0)$, $(2, 3)$, implementar un spline cúbico natural.

Solución:

1. **Datos:** $x = [0, 1, 2]$, $y = [1, 0, 3]$, $n = 2$ intervalos.

2. **Distancias:** $h_0 = 1 - 0 = 1$, $h_1 = 2 - 1 = 1$.

3. **Sistema tridiagonal:**

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 4 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 3\left(\frac{3-0}{1} - \frac{0-1}{1}\right) \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 12 \\ 0 \end{bmatrix}$$

4. **Solución:** $c_0 = 0$, $c_1 = 3$, $c_2 = 0$.

5. **Coeficientes:**

$$\begin{aligned} a_0 &= y_0 = 1, \quad a_1 = y_1 = 0 \\ b_0 &= \frac{y_1 - y_0}{h_0} - \frac{h_0}{3}(2c_0 + c_1) = -1 - \frac{1}{3}(0 + 3) = -2 \\ b_1 &= \frac{y_2 - y_1}{h_1} - \frac{h_1}{3}(2c_1 + c_2) = 3 - \frac{1}{3}(6 + 0) = 1 \\ d_0 &= \frac{c_1 - c_0}{3h_0} = \frac{3 - 0}{3} = 1 \\ d_1 &= \frac{c_2 - c_1}{3h_1} = \frac{0 - 3}{3} = -1 \end{aligned}$$

6. **Splines:**

$$\begin{aligned} S_0(x) &= 1 - 2(x - 0) + 0(x - 0)^2 + 1(x - 0)^3 = 1 - 2x + x^3, \quad x \in [0, 1] \\ S_1(x) &= 0 + 1(x - 1) + 3(x - 1)^2 - 1(x - 1)^3, \quad x \in [1, 2] \end{aligned}$$

7.11. Resumen del Capítulo

- La **interpolación lineal** es simple y estable (C^0) pero solo proporciona continuidad de la función, no de sus derivadas.
- Los **polinomios interpolantes** (Lagrange, Newton) ofrecen continuidad C^∞ pero pueden sufrir el fenómeno de Runge con nodos equiespaciados y grados altos.
- Los **splines cúbicos** proporcionan un equilibrio óptimo entre suavidad (C^2) y estabilidad, siendo especialmente útiles para interpolación con muchos puntos.
- Para interpolación polinomial con muchos puntos, se recomiendan los **nodos de Chebyshev** para minimizar el error máximo.
- La selección del método depende del problema específico:
  - **Velocidad:** Interpolación lineal

  - **Precisión con pocos puntos:** Polinomios interpolantes
  - **Suavidad con muchos puntos:** Splines cúbicos

- Las aplicaciones de la interpolación son vastas e incluyen procesamiento de señales, gráficos por computadora, análisis de datos científicos y financieros, entre otros.

Para Profundizar

- **Libros recomendados:**
  - Burden, R. L., & Faires, J. D. *Numerical Analysis* (Capítulo 3: Interpolation and Polynomial Approximation)
  - Quarteroni, A., Saleri, F., & Gervasio, P. *Scientific Computing with MATLAB and Octave*
  - Press, W. H., et al. *Numerical Recipes: The Art of Scientific Computing*
- **Librerías Python:**
  - `scipy.interpolate`: Contiene implementaciones eficientes de diversos métodos de interpolación.
  - `numpy.interp`: Interpolación lineal simple.
  - `scipy.ndimage.map_coordinates`: Para interpolación en imágenes y datos multidimensionales.
- **Temas avanzados:**
  - Interpolación multivariada
  - Interpolación con restricciones (monotonicidad, convexidad)
  - Interpolación racional
  - Interpolación trigonométrica (FFT-based)
  - Interpolación en mallas no estructuradas

Capítulo 8

Eigenvalores, Eigenvectores y Métodos de Optimización

Objetivos del capítulo: Al concluir este capítulo, el lector será capaz de:

- Comprender la interpretación geométrica de eigenvalores y eigenvectores como direcciones invariantes de transformaciones lineales
- Dominar el cálculo de eigenvalores y eigenvectores para matrices 2×2 y 3×3 mediante la ecuación característica
- Conectar los eigenvalores con problemas de optimización mediante el análisis de la matriz Hessiana
- Clasificar puntos críticos (máximos, mínimos, puntos silla) usando eigenvalores
- Implementar en R algoritmos numéricos como el método de la potencia para problemas de eigenvalores grandes
- Aplicar eigenvalores a problemas del mundo real como análisis de componentes principales (PCA) y sistemas dinámicos

8.1. Introducción Conceptual: ¿Qué son Eigenvalores y Eigenvectores?

8.1.1. La Definición Fundamental

Dada una matriz cuadrada $A \in \mathbb{R}^{n \times n}$, un **eigenvector** $\mathbf{v} \neq \mathbf{0}$ y un **eigenvalor** $\lambda \in \mathbb{C}$ satisfacen la ecuación:

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

Interpretación Geométrica: Direcciones que no Cambian:

- El eigenvector $\mathbf{v}$ es una **dirección especial** que no cambia bajo la transformación A
- El eigenvalor λ es un **escalar** que indica cuánto se estira o encoge el vector en esa dirección
- Si $\lambda > 1$: El vector se alarga
- Si $0 < \lambda < 1$: El vector se acorta
- Si $\lambda < 0$: El vector se invierte
- Si $\lambda = 0$: El vector se anula (va al vector cero)

8.1.2. Ejemplos Visuales de la Vida Real

Casos Concretos donde Aparecen Eigenvalores:

- **Mecánica cuántica**: Los estados estacionarios de un sistema son eigenvectores del Hamiltoniano, y los eigenvalores son las energías permitidas
- **Análisis estructural**: Los modos de vibración de un edificio son eigenvectores, y las frecuencias naturales son los eigenvalores
- **Procesamiento de imágenes**: En análisis de componentes principales (PCA), los eigenvectores son las direcciones de máxima varianza en los datos
- **Sistemas dinámicos**: La estabilidad de un sistema se determina por los eigenvalores de su matriz de transición
- **Google PageRank**: El ranking de páginas web es el eigenvector principal de la matriz de enlaces

8.2. Teoría Matemática Detallada: Cálculo de Eigenvalores

8.2.1. La Ecuación Característica: La Puerta de Entrada

Para encontrar los eigenvalores de una matriz A, formamos la ecuación característica:

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

donde I es la matriz identidad del mismo tamaño que A.

¿Por qué funciona esta ecuación?: Si $A\mathbf{v} = \lambda\mathbf{v}$, entonces podemos reescribir como:

$$A\mathbf{v} - \lambda\mathbf{v} = \mathbf{0}$$

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

Para que exista una solución no trivial $\mathbf{v} \neq \mathbf{0}$, la matriz $A - \lambda I$ debe ser singular, es decir, su determinante debe ser cero:

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

Esta es una ecuación polinómica en λ de grado n, cuyas raíces son los eigenvalores.

8.2.2. Algoritmo Paso a Paso para Matrices 2×2

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

1. Formar $A - \lambda I = \begin{pmatrix} a - \lambda & b \\ c & d - \lambda \end{pmatrix}$
2. Calcular determinante: $\det = (a - \lambda)(d - \lambda) - bc$
3. Expandir: $\lambda^2 - (a + d)\lambda + (ad - bc) = 0$
4. Resolver esta ecuación cuadrática para λ
5. Para cada λ_i, resolver $(A - \lambda_i I)\mathbf{v}_i = \mathbf{0}$ para encontrar eigenvectores

8.2.3. Ejemplo Detallado 1: Matriz Simétrica

Ejemplo Completo: $A = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$: **Paso 1:** Formar $A - \lambda I$

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

Paso 2: Calcular determinante

$$\det = (3-\lambda)(3-\lambda) - 1 \cdot 1 = \lambda^2 - 6\lambda + 8$$

Paso 3: Resolver ecuación característica

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

$$(\lambda - 4)(\lambda - 2) = 0 \quad \Rightarrow \quad \lambda_1 = 4,\ \lambda_2 = 2$$

Paso 4a: Eigenvector para $\lambda_1 = 4$

$$(A - 4I)\mathbf{v}_1 = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$-v_1 + v_2 = 0 \ \Rightarrow \ v_2 = v_1 \ \Rightarrow \ \mathbf{v}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Paso 4b: Eigenvector para $\lambda_2 = 2$

$$(A - 2I)\mathbf{v}_2 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$v_1 + v_2 = 0 \ \Rightarrow \ v_2 = -v_1 \ \Rightarrow \ \mathbf{v}_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Propiedad notable: Los eigenvectores son ortogonales: $\mathbf{v}_1 \cdot \mathbf{v}_2 = 0$

8.3. Propiedades Algebraicas Fundamentales

8.3.1. Teoremas Clave que Debes Conocer

Teorema del Traza: La Suma de los Eigenvalores:

$$\sum_{i=1}^{n} \lambda_i = \text{tr}(A) = a_{11} + a_{22} + \cdots + a_{nn}$$

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

$$\lambda_1 + \lambda_2 = 4 + 2 = 6, \quad \text{tr}(A) = 3 + 3 = 6 \quad \checkmark$$

Teorema del Determinante: El Producto de los Eigenvalores:

$$\prod_{i=1}^{n} \lambda_i = \det(A)$$

Ejemplo: Para la misma matriz:

$$\lambda_1 \times \lambda_2 = 4 \times 2 = 8, \quad \det(A) = 9 - 1 = 8 \quad \checkmark$$

Consecuencia importante: Si $\det(A) = 0$, entonces al menos un eigenvalor es cero.

8.3.2. Propiedades por Tipo de Matriz

Tipo de Matriz	Condición	Propiedad de Eigenvalores
Simétrica	$A = A^T$	Todos reales, eigenvectores ortogonales
Definida Positiva	$\mathbf{x}^T A\mathbf{x} > 0$	Todos positivos
Definida Negativa	$\mathbf{x}^T A\mathbf{x} < 0$	Todos negativos
Singular	$\det(A) = 0$	Al menos uno cero
Idempotente	$A^2 = A$	0 o 1
Nilpotente	$A^k = 0$	Todos cero

Tabla 8.1: Propiedades especiales de eigenvalores

8.4. Conexión con Optimización: La Matriz Hessiana

8.4.1. ¿Por qué son Importantes en Optimización?

Cuando optimizamos una función $f : \mathbb{R}^n \to \mathbb{R}$, necesitamos:

1. Encontrar puntos críticos: donde $\nabla f = \mathbf{0}$
2. Clasificar si son máximos, mínimos o puntos silla

La **matriz Hessiana** H_f contiene todas las segundas derivadas:

$$H_f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

Regla de Oro para Clasificación:

- **Todos eigenvalores $>\mathbf{0}$**: Mínimo local (función convexa en todas direcciones)
- **Todos eigenvalores $<\mathbf{0}$**: Máximo local (función cóncava en todas direcciones)
- **Mezcla de signos**: Punto silla (convexa en unas direcciones, cóncava en otras)
- **Algunos $= \mathbf{0}$**: Indeterminado (necesita análisis de orden superior)

8.4.2. Ejemplo 2: Clasificación Completa de Punto Crítico

Ejemplo: $f(x, y) = x^2 + 3y^2 + 2xy$: **Paso 1:** Encontrar punto crítico

$$\nabla f = \begin{pmatrix} 2x + 2y \\ 6y + 2x \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Solución: $x = 0, y = 0$

Paso 2: Calcular matriz Hessiana en (0,0)

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

Paso 3: Eigenvalores de H

$$\det(H - \lambda I) = (2 - \lambda)(6 - \lambda) - 4 = \lambda^2 - 8\lambda + 8 = 0$$

$$\lambda = \frac{8 \pm \sqrt{64 - 32}}{2} = \frac{8 \pm \sqrt{32}}{2} \approx \begin{cases} 6{,}828 \\ 1{,}172 \end{cases}$$

Paso 4: Clasificar Ambos eigenvalores positivos $\Rightarrow$ **MÍNIMO LOCAL** en (0,0)

8.5. Implementacion Practica en R

8.5.1. Calculo Directo de Eigenvalores y Eigenvectores

Listing 8.1: Calculo basico de eigenvalores en R

```
# Matriz 2x2 simetrica
A <- matrix(c(3, 1, 1, 3), nrow=2, byrow=TRUE)
cat("Matriz A:\n"); print(A)

# Calcular eigenvalores y eigenvectores
eigen_result <- eigen(A)
cat("\nEigenvalores:\n"); print(eigen_result$values)
cat("\nEigenvectores:\n"); print(eigen_result$vectors)

# Verificacion: A * v = lambda * v
cat("\n=== Verificacion ===\n")
for(i in 1:2) {
    v <- eigen_result$vectors[, i]
    lambda <- eigen_result$values[i]
    cat(sprintf("\nEigenvalor %.4f:\n", lambda))
    cat("A*v =", A %*% v, "| lambda*v =", lambda * v, "\n")
}

# Propiedades algebraicas
cat("\n=== Propiedades ===\n")
cat(sprintf("Traza: %.4f | Suma eigenvalores: %.4f\n",
            sum(diag(A)), sum(eigen_result$values)))
cat(sprintf("Det: %.4f | Producto eigenvalores: %.4f\n",
            det(A), prod(eigen_result$values)))
```

8.5.2. Clasificacion de Puntos Criticos en R

Listing 8.2: Clasificar puntos criticos usando eigenvalores

```
clasificar_punto_critico <- function(f, x0, y0, h = 1e-5) {
    # Hessiana numerica
    fxx <- (f(x0+h, y0) - 2*f(x0, y0) + f(x0-h, y0)) / h^2
    fyy <- (f(x0, y0+h) - 2*f(x0, y0) + f(x0, y0-h)) / h^2
    fxy <- (f(x0+h, y0+h) - f(x0+h, y0-h) -
            f(x0-h, y0+h) + f(x0-h, y0-h)) / (4*h^2)

```

```
    H <- matrix(c(fxx, fxy, fxy, fyy), 2, 2)
    eigenvals <- eigen(H)$values

    # Clasificar
    tipo <- if (all(eigenvals > 0)) "Minimo local"
    else if (all(eigenvals < 0)) "Maximo local"
    else if (any(eigenvals > 0) && any(eigenvals < 0)) "Punto silla
       "
    else "Indeterminado"

    list(hessiana = H, eigenvalores = eigenvals, tipo = tipo)
}

# Ejemplos
f1 <- function(x, y) x^2 + 3*y^2 + 2*x*y
res1 <- clasificar_punto_critico(f1, 0, 0)
cat("f(x,y) = x^2 + 3y^2 + 2xy en (0,0)\n")
cat("Eigenvalores:", res1$eigenvalores, "\nTipo:", res1$tipo, "\n")

f2 <- function(x, y) x^2 - y^2
res2 <- clasificar_punto_critico(f2, 0, 0)
cat("\nf(x,y) = x^2 - y^2 en (0,0)\n")
cat("Eigenvalores:", res2$eigenvalores, "\nTipo:", res2$tipo, "\n")
```

8.5.3. Metodo de la Potencia para Eigenvalor Dominante

Metodo de la Potencia: Para matrices grandes, este metodo encuentra eficientemente solo el eigenvalor de mayor magnitud.

Listing 8.3: Metodo de la potencia en R

```
metodo_potencia <- function(A, max_iter = 1000, tol = 1e-10) {
    n <- nrow(A)
    v <- rnorm(n); v <- v / sqrt(sum(v^2))
    lambda_old <- 0

    for (k in 1:max_iter) {
        w <- A %*% v
        lambda_new <- sum(v * w) / sum(v^2)
        v <- w / sqrt(sum(w^2))

```

```
        if (abs(lambda_new - lambda_old) < tol) break
        lambda_old <- lambda_new
    }

    list(eigenvalor = lambda_new, eigenvector = v, iteraciones = k)
}

# Ejemplo
A <- matrix(c(4, 1, 0, 1, 3, 1, 0, 1, 2), 3, 3, byrow=TRUE)
cat("Matriz A:\n"); print(A)

exacto <- eigen(A)
resultado <- metodo_potencia(A, max_iter=200, tol=1e-12)

cat(sprintf("\nEigenvalor estimado: %.8f\n", resultado$eigenvalor))
cat(sprintf("Eigenvalor exacto:   %.8f\n", exacto$values[1]))
cat(sprintf("Error: %.2e | Iteraciones: %d\n",
            abs(resultado$eigenvalor - exacto$values[1]),
               resultado$iteraciones))
```

8.6. Aplicaciones en el Mundo Real

8.6.1. Analisis de Componentes Principales (PCA)

PCA: Reduccion de Dimensionalidad usando Eigenvalores: PCA es el metodo mas importante que usa eigenvalores en ciencia de datos. Transforma datos correlacionados en componentes no correlacionados (componentes principales), que son los eigenvectores de la matriz de covarianza.

Listing 8.4: Implementacion de PCA desde cero en R

```
pca_manual <- function(X, n_components = 2) {
    # 1. Centrar los datos (restar la media de cada columna)
    X_centered <- scale(X, center = TRUE, scale = FALSE)

    # 2. Calcular matriz de covarianza
    cov_matrix <- cov(X_centered)

    # 3. Calcular eigenvalores y eigenvectores
    eigen_result <- eigen(cov_matrix)
```

```
10
11      # 4. Ordenar por eigenvalores descendentes
12      idx <- order(eigen_result$values, decreasing = TRUE)
13      eigenvalues <- eigen_result$values[idx]
14      eigenvectors <- eigen_result$vectors[, idx]
15
16      # 5. Seleccionar componentes principales
17      componentes <- eigenvectors[, 1:n_components]
18
19      # 6. Transformar datos al nuevo espacio
20      X_pca <- X_centered %*% componentes
21
22      # 7. Calcular varianza explicada
23      varianza_total <- sum(eigenvalues)
24      varianza_explicada <- eigenvalues[1:n_components] /
           varianza_total
25
26      # Retornar resultados
27      list(
28          datos_transformados = X_pca,
29          componentes = componentes,
30          eigenvalores = eigenvalues,
31          varianza_explicada = varianza_explicada,
32          varianza_total_explicada = sum(varianza_explicada)
33      )
34  }
35
36  # Ejemplo con datos sinteticos
37  cat("=== Analisis de Componentes Principales (PCA) ===\n\n")
38
39  # Generar datos sinteticos correlacionados
40  set.seed(42)
41  n <- 200  # Numero de muestras
42  p <- 10   # Numero de caracteristicas originales
43
44  # Crear datos con estructura
45  X <- matrix(rnorm(n * p), n, p)
46  # Introducir correlaciones
47  X[, 3] <- 0.8 * X[, 1] + 0.2 * rnorm(n)  # Caracteristica 3
       correlacionada con 1
48  X[, 4] <- 0.6 * X[, 2] + 0.4 * rnorm(n)  # Caracteristica 4
```

```
   correlacionada con 2
X[, 5] <- 0.7 * X[, 1] + 0.3 * X[, 2] + 0.1 * rnorm(n)

cat("Datos originales:\n")
cat(sprintf("  Muestras: %d\n", n))
cat(sprintf("  Caracteristicas: %d\n", p))
cat(sprintf("  Dimension total: %d\n", n * p))

# Calcular PCA
resultado_pca <- pca_manual(X, n_components = 3)

cat("\n=== Resultados del PCA ===\n")
cat(sprintf("\nDatos transformados: %d muestras x %d componentes\n
   ",
            nrow(resultado_pca$datos_transformados),
            ncol(resultado_pca$datos_transformados)))

cat("\nVarianza explicada por cada componente:\n")
for (i in 1:3) {
    cat(sprintf("  Componente %d: %.2f%%\n", i,
       resultado_pca$varianza_explicada[i] * 100))
}

cat(sprintf("\nVarianza total explicada por 3 componentes: %.2f%%\n
   ",
            resultado_pca$varianza_total_explicada * 100))

# Matriz de correlacion antes y despues
cat("\n=== Comparacion de Correlaciones ===\n")
cat("Correlacion promedio antes de PCA:")
cor_antes <- mean(abs(cor(X)[lower.tri(cor(X))]))
cat(sprintf(" %.4f\n", cor_antes))

cat("Correlacion promedio despues de PCA:")
cor_despues <- mean(abs(cor(resultado_pca$datos_transformados)[
   lower.tri(cor(resultado_pca$datos_transformados))]))
cat(sprintf(" %.4f\n", cor_despues))

cat(sprintf("Reduccion de correlacion: %.1f%%\n", (1 - cor_despues/
   cor_antes) * 100))

```

```
# Visualizacion
if (require("ggplot2") && require("GGally")) {
    library(ggplot2)
    library(GGally)

    # Preparar datos para visualizacion
    df_pca <- as.data.frame(resultado_pca$datos_transformados)
    colnames(df_pca) <- paste0("PC", 1:3)

    # Grafico 1: Varianza explicada
    df_var <- data.frame(
        Componente = 1:length(resultado_pca$eigenvalores),
        Varianza = resultado_pca$eigenvalores,
        Varianza.Acumulada = cumsum(resultado_pca$eigenvalores) /
           sum(resultado_pca$eigenvalores)
    )

    g1 <- ggplot(df_var[1:10, ], aes(x=Componente, y=Varianza)) +
        geom_bar(stat="identity", fill="steelblue", alpha=0.7) +
        geom_line(aes(y=Varianza.Acumulada * max(Varianza)), color
           ="red", size=1) +
        geom_point(aes(y=Varianza.Acumulada * max(Varianza)), color
           ="red", size=2) +
        scale_y_continuous(
            name = "Varianza",
            sec.axis = sec_axis(~./max(df_var$Varianza[1:10]),
                                name="Varianza Acumulada",
                                labels=scales::percent)
        ) +
        labs(title="Varianza Explicada por Componentes Principales
           ") +
        theme_minimal()

    # Grafico 2: Proyeccion en 2D
    g2 <- ggplot(df_pca, aes(x=PC1, y=PC2)) +
        geom_point(alpha=0.6, color="blue") +
        labs(title="Proyeccion de Datos en los Dos Primeros
           Componentes",
             x=paste0("PC1 (", round(
                resultado_pca$varianza_explicada[1]*100,1), "%)"),
             y=paste0("PC2 (", round(
```

```
                resultado_pca$varianza_explicada[2]*100,1), "%)"))
                +
        theme_minimal()

    # Mostrar graficos
    print(g1)
    print(g2)

    # Grafico 3: Matriz de componentes
    cat("\nPrimeros 3 componentes principales:\n")
    print(round(resultado_pca$componentes[, 1:3], 3))
}

# Aplicacion practica: Compresion de datos
cat("\n=== Aplicacion: Compresion de Datos ===\n")
tamanio_original <- n * p * 8  # Asumiendo 8 bytes por numero (
   double)
tamanio_comprimido <- n * 3 * 8 + p * 3 * 8  # Datos + componentes
compresion <- (1 - tamanio_comprimido/tamanio_original) * 100

cat(sprintf("Tamano original: %.0f bytes\n", tamanio_original))
cat(sprintf("Tamano despues de PCA (3 componentes): %.0f bytes\n",
   tamanio_comprimido))
cat(sprintf("Compresion: %.1f%%\n", compresion))
cat(sprintf("Varianza retenida: %.1f%%\n",
   resultado_pca$varianza_total_explicada * 100))
```

8.6.2. Sistemas Dinamicos y Estabilidad

Eigenvalores en Sistemas Dinamicos: ¿Crecera o Decaera?: Para un sistema lineal $\dot{\mathbf{x}} = A\mathbf{x}$, los eigenvalores de A determinan completamente el comportamiento a largo plazo:

- **Estable**: Todos eigenvalores tienen parte real negativa
- **Inestable**: Algun eigenvalor tiene parte real positiva
- **Marginalmente estable**: Parte real cero (sin repetidos)

Listing 8.5: Analisis de estabilidad de sistemas dinamicos

```
analizar_estabilidad_sistema <- function(A) {
```

```
    # Calcular eigenvalores
    eigenvals <- eigen(A)$values

    # Partes reales e imaginarias
    partes_reales <- Re(eigenvals)
    partes_imaginarias <- Im(eigenvals)

    # Determinar estabilidad
    if (all(partes_reales < 0)) {
        estabilidad <- "Estable (todos eigenvalores tienen parte
           real negativa)"
    } else if (any(partes_reales > 0)) {
        estabilidad <- "Inestable (algun eigenvalor tiene parte
           real positiva)"
    } else if (any(partes_reales == 0)) {
        estabilidad <- "Marginalmente estable (parte real cero)"
    } else {
        estabilidad <- "Caso especial"
    }

    # Retornar resultados
    list(
        eigenvalores = eigenvals,
        partes_reales = partes_reales,
        partes_imaginarias = partes_imaginarias,
        estabilidad = estabilidad,
        mas_inestable = max(partes_reales)  # Para tasa de
           crecimiento
    )
}

# Ejemplos de sistemas
cat("=== Analisis de Estabilidad de Sistemas Dinamicos ===\n\n")

# Sistema 1: Estable (oscilaciones amortiguadas)
A1 <- matrix(c(-0.1, -1,
               1, -0.1), nrow=2, byrow=TRUE)
cat("Sistema 1: Oscilador amortiguado\n")
print(A1)
res1 <- analizar_estabilidad_sistema(A1)
cat(sprintf("Eigenvalores: %.3f \\pm %.3fi\n", res1$partes_reales
```

```
   [1], abs(res1$partes_imaginarias[1])))
cat("Estabilidad:", res1$estabilidad, "\n\n")

# Sistema 2: Inestable (crecimiento exponencial)
A2 <- matrix(c(0.2, -1,
               1, 0.2), nrow=2, byrow=TRUE)
cat("Sistema 2: Oscilador inestable\n")
print(A2)
res2 <- analizar_estabilidad_sistema(A2)
cat(sprintf("Eigenvalores: %.3f \\pm %.3fi\n", res2$partes_reales
   [1], abs(res2$partes_imaginarias[1])))
cat("Estabilidad:", res2$estabilidad, "\n\n")

# Sistema 3: Marginalmente estable (oscilaciones perpetuas)
A3 <- matrix(c(0, -1,
               1, 0), nrow=2, byrow=TRUE)
cat("Sistema 3: Oscilador armonico\n")
print(A3)
res3 <- analizar_estabilidad_sistema(A3)
cat(sprintf("Eigenvalores: \\pm %.3fi\n", abs(
   res3$partes_imaginarias[1])))
cat("Estabilidad:", res3$estabilidad, "\n")
# Simulacion de trayectorias
if (require("deSolve")) {
    library(deSolve)

    # Funcion para el sistema dinamico
    sistema_dinamico <- function(t, estado, parametros) {
        with(as.list(c(estado, parametros)), {
            dx <- A[1,1]*x + A[1,2]*y
            dy <- A[2,1]*x + A[2,2]*y
            list(c(dx, dy))
        })
    }

    # Condiciones iniciales
    estado_inicial <- c(x = 1, y = 0)
    tiempos <- seq(0, 20, by = 0.1)

    # Simular los tres sistemas
    resultados <- list()
```

```
78      sistemas <- list(A1, A2, A3)
79      nombres <- c("Estable", "Inestable", "Marginal")
80
81      for (i in 1:3) {
82          parametros <- list(A = sistemas[[i]])
83          resultados[[i]] <- ode(y = estado_inicial,
84                                 times = tiempos,
85                                 func = sistema_dinamico,
86                                 parms = parametros)
87      }
88
89      # Visualizar resultados
90      if (require("ggplot2")) {
91          library(ggplot2)
92
93          # Preparar datos para grafico
94          df <- data.frame()
95          for (i in 1:3) {
96              temp <- as.data.frame(resultados[[i]])
97              temp$Sistema <- nombres[i]
98              df <- rbind(df, temp)
99          }
100
101         # Grafico de trayectorias
102         g <- ggplot(df, aes(x=x, y=y, color=Sistema)) +
103             geom_path(size=1, alpha=0.8) +
104             geom_point(data=df[df$time==0, ], size=3) +
105             labs(title="Trayectorias de Sistemas Dinamicos",
106                  x="Coordenada x", y="Coordenada y",
107                  subtitle="Punto inicial: (1,0)") +
108             theme_minimal() +
109             facet_wrap(~Sistema, ncol=3)
110
111         print(g)
112
113         # Grafico de evolucion temporal
114         g2 <- ggplot(df, aes(x=time, y=sqrt(x^2+y^2), color=Sistema
               )) +
115             geom_line(size=1) +
116             labs(title="Distancia al Origen vs Tiempo",
117                  x="Tiempo", y="Distancia al origen") +
```

```
            theme_minimal()

        print(g2)
    }
}
```

8.7. Ejercicios y Problemas Resueltos

8.7.1. Ejercicios Teóricos

Ejercicio 1: Matrices Especiales y sus Eigenvalores: **Problema**: Encuentra los eigenvalores de las siguientes matrices sin hacer cálculos detallados:

1. $A = \begin{pmatrix} 5 & 0 \\ 0 & -3 \end{pmatrix}$ (matriz diagonal)
2. $B = \begin{pmatrix} 2 & 4 \\ 4 & 2 \end{pmatrix}$ (matriz simétrica)
3. $C = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ (matriz antisimétrica)
4. $D = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$ (matriz triangular)

Solución:

1. Para matrices diagonales, los eigenvalores están en la diagonal: $\lambda_1 = 5, \lambda_2 = -3$
2. Para matrices simétricas 2×2 de la forma $\begin{pmatrix} a & b \\ b & a \end{pmatrix}$, los eigenvalores son $a \pm b$: $\lambda_1 = 6, \lambda_2 = -2$
3. Para matrices antisimétricas 2×2, los eigenvalores son imaginarios puros: $\lambda = \pm i$
4. Para matrices triangulares, los eigenvalores están en la diagonal: $\lambda_1 = 1, \lambda_2 = 1$ (doble)

Ejercicio 2: Clasificación de Todos los Tipos de Puntos Críticos: **Problema**: Para cada función, clasifica el punto crítico (0,0):

1. $f(x, y) = x^2 + y^2$

2. $f(x, y) = -x^2 - 2y^2$
3. $f(x, y) = x^2 - y^2$
4. $f(x, y) = x^2 + y^4$

Solución en R: **Resultados**:

1. Mínimo local (eigenvalores positivos)
2. Máximo local (eigenvalores negativos)
3. Punto silla (un positivo, uno negativo)
4. Indeterminado (uno positivo, uno cero)

8.8. Tablas de Referencia

Propiedades de Eigenvalores por Tipo de Matriz:

Tipo	Eigenvalores	Aplicaciones/Comentarios
Diagonal	En la diagonal	Más fáciles de calcular, eigenvectores son ejes coordenados
Simétrica	Reales	Eigenvectores ortogonales, común en problemas físicos
Antisimétrica	Imaginarios puros	Representan rotaciones puras
Triangular	En la diagonal	Útiles para algoritmos numéricos
Def. Positiva	Todos > 0	Mínimos en optimización, matrices de covarianza
Def. Negativa	Todos < 0	Máximos en optimización
Singular	Al menos uno $= 0$	Sistemas con soluciones no únicas
Ortogonal	Módulo 1	Preservan distancias (rotaciones, reflexiones)

Clasificación de Puntos Críticos usando Eigenvalores de Hessiana:

Eigenvalores	Tipo	Interpretación Geométrica
Todos > 0	Mínimo local	La función es convexa en todas direcciones
Todos < 0	Máximo local	La función es cóncava en todas direcciones
Positivos y negativos	Punto silla	Convexa en unas direcciones, cóncava en otras
Algunos $= 0$	Indeterminado	Necesita análisis de orden superior
Positivos, algunos $= 0$	Pos. semidef.	Posible mínimo, pero no estricto
Negativos, algunos $= 0$	Neg. semidef.	Posible máximo, pero no estricto

Métodos Numéricos para Eigenvalores - Comparación:

Método	Complejidad	Ventajas	Desventajas
Directo (eigen)	$O(n^3)$	Exacto para n pequeño	Impracticable para n grande
Método Potencia	$O(n^2)$ por iter	Simple, para matrices grandes	Solo eigenvalor dominante
QR iterativo	$O(n^3)$	Todos eigenvalores	Costoso para n grande
Lanczos	$O(kn^2)$	Para matrices dispersas	Pérdida de ortogonalidad
Jacobi	$O(n^3)$	Estable para simétricas	Lento
Divide y vencerás	$O(n^3)$	Rápido para simétricas	Implementación compleja

8.9. Conclusiones y Recomendaciones Prácticas

Decálogo del Analista de Eigenvalores:

1. **Conoce el tipo de matriz**: Simétrica → eigenvalores reales, Diagonal → eigenvalores en diagonal
2. **Verifica propiedades algebraicas**: Traza = suma de eigenvalores, Determinante = producto
3. **Usa el método apropiado**: Pequeñas matrices → cálculo directo, Grandes matrices → métodos iterativos
4. **En optimización**: Clasifica puntos críticos con eigenvalores de la Hessiana
5. **En ciencia de datos**: Usa PCA (eigenvalores de matriz de covarianza) para reducción dimensional
6. **En sistemas dinámicos**: Analiza estabilidad con partes reales de eigenvalores

7. **Siempre verifica**: $A\mathbf{v} = \lambda\mathbf{v}$ debe cumplirse con alta precisión
8. **Cuidado con matrices casi singulares**: Pequeños cambios pueden causar grandes variaciones en eigenvalores
9. **Visualiza**: Los eigenvectores son direcciones, los eigenvalores son escalas
10. **Practica con problemas reales**: Mecánica, finanzas, procesamiento de señales, machine learning

Reflexión Final: Más que Números, una Forma de Pensar: Los eigenvalores y eigenvectores no son solo conceptos matemáticos abstractos. Representan una forma poderosa de pensar sobre sistemas complejos:

- **Descomposición**: Sistemas complejos pueden separarse en componentes independientes
- **Jerarquía**: Algunas direcciones (componentes principales) son más importantes que otras
- **Estabilidad**: El comportamiento a largo plazo depende de escalas fundamentales
- **Optimización**: La geometría de funciones se revela a través de eigenvalores
- **Compresión**: La información esencial puede extraerse descartando componentes menores

En un mundo cada vez más dominado por datos de alta dimensionalidad, la habilidad de trabajar con eigenvalores se ha convertido en una competencia fundamental para científicos, ingenieros de datos, investigadores y cualquier profesional que busque entender sistemas complejos a través de sus propiedades fundamentales.

Bibliografía y Recursos para Profundizar

Lecturas Esenciales:

1. **Strang, G.** (2016). *Introduction to Linear Algebra* (5ta ed.). Excelente para intuición geométrica.
2. **Trefethen, L. N., & Bau, D.** (1997). *Numerical Linear Algebra*. Enfoque computacional moderno.

3. **Golub, G. H., & Van Loan, C. F.** (2013). *Matrix Computations* (4ta ed.). Referencia avanzada definitiva.
4. **Jolliffe, I. T.** (2002). *Principal Component Analysis* (2da ed.). Clásico sobre PCA.
5. **Nocedal, J., & Wright, S. J.** (2006). *Numerical Optimization*. Para conexión con optimización.

Recursos en R y En Línea:

- **Funciones base de R**:
  - `eigen()`: Cálculo de eigenvalores y eigenvectores
  - `prcomp()`, `princomp()`: PCA implementado
  - `svd()`: Descomposición en valores singulares (relacionada)
- **Packages especializados**:
  - `RSpectra`: Para problemas de eigenvalores grandes
  - `irlba`: Para SVD de matrices grandes
  - `corpcor`: Para matrices de covarianza y correlación
- **Tutoriales y Cursos**:
  - Coursera: "Mathematics for Machine Learning"(Imperial College London)
  - MIT OpenCourseWare: Linear Algebra (Gilbert Strang)
  - 3Blue1Brown: .ᴱˢsence of Linear Algebra"(YouTube)

Capítulo 9

Aplicación Práctica: Optimización de Páginas Web con Métodos Numéricos

9.1. Contexto del Estudio

Resumen del Estudio Este estudio evalúa el rendimiento de dos plataformas digitales críticas para los estudiantes de la Universidad Nacional del Altiplano (UNA) de Puno: **YouTube** y el **Aula Virtual UNA Puno**. Se utilizan herramientas profesionales como Apache JMeter y Google Lighthouse, complementadas con métodos numéricos de interpolación y optimización para análisis avanzado.

9.1.1. Herramientas Utilizadas

Método Aplicado: Apache JMeter:

- Herramienta de código abierto
- Pruebas de carga, estrés y rendimiento
- Simula múltiples usuarios concurrentes
- Mide tiempos de respuesta y throughput

Método Aplicado: Google Lighthouse:

- Herramienta automatizada de auditoría web
- Evalúa: rendimiento, accesibilidad, SEO
- Proporciona métricas Core Web Vitals
- Genera recomendaciones de mejora

9.2. Objetivos del Estudio

- **Evaluar** tiempos de respuesta y capacidad de carga de ambas plataformas
- **Identificar** cuellos de botella y limitaciones de infraestructura
- **Aplicar** métodos numéricos de interpolación y búsqueda de raíces para optimización
- **Proponer** recomendaciones basadas en análisis cuantitativo

9.3. Pruebas de Rendimiento con Apache JMeter

9.3.1. Configuración de Pruebas

1lightbluewhite

primary!30 **Parámetro**	**YouTube**	**Aula Virtual**	**Unidad**
Usuarios simulados	100, 500, 1000	100, 500, 1000	usuarios
Duración prueba	10 min	10 min	minutos
Tipo de prueba	Carga incremental	Carga incremental	-

Tabla 9.1: Configuración de pruebas JMeter

9.3.2. Datos de Rendimiento Obtenidos

2lightgreen!30white

secondary!40 **Plataforma**	**Usuarios**	**Average**	**Median**	**90 %**	**95 %**	**99 %**	**Min**
YouTube	100	39777	28962	61221	96200	145895	19158
YouTube	500	57318	50867	67083	72251	505058	11936
YouTube	1000	131565	127942	166999	172988	212717	12425
Aula Virtual	100	19747	21656	25012	26446	26230	8475
Aula Virtual	500	96235	97028	152451	154567	169422	21026
Aula Virtual	1000	43860	22515	153626	190261	212665	21659

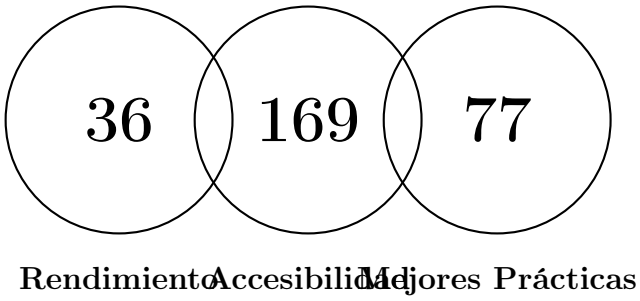
Tabla 9.2: Datos de rendimiento obtenidos con Apache JMeter (tiempos en ms)

9.4. Resultados con Google Lighthouse

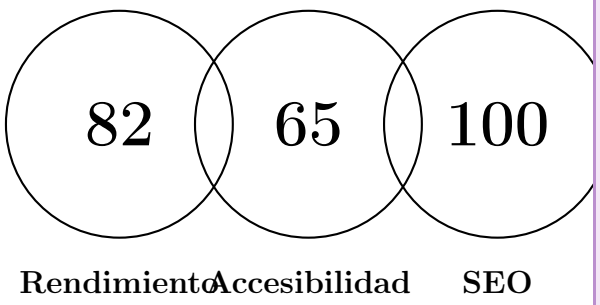
9.4.1. YouTube

Resultados: **Métricas Principales:**

- LCP: 5.0 s (Lento)
- INP: 271 ms (Necesita mejora)
- CLS: 0.41 (Pobre)
- FCP: 3.1 s
- TTFB: 0.5 s



9.4.2. Aula Virtual UNA Puno



Resultados: **Métricas Principales:**

- LCP: 2.1 s (Bueno)
- INP: 50 ms (Excelente)
- CLS: 0 (Excelente)
- FCP: 1.8 s
- TTFB: 0.7 s

9.5. Aplicación de Métodos Numéricos para Optimización

9.5.1. Interpolación para Estimación de Límites

Listing 9.1: Interpolación para encontrar límites de usuarios

```
# Datos obtenidos de JMeter
usuarios <- c(100, 500, 1000)
average <- c(39777, 57318, 131565)  # tiempos en ms
error <- c(0, 13.6, 60.8)           # porcentajes
```

```
5
6  # Funci n de interpolacion lineal para tiempo promedio
7  interpolacion_average <- function(average_target, usuarios, average
       ) {
8      for(i in 1:(length(usuarios)-1)) {
9          if(average_target >= average[i] && average_target <=
               average[i+1]) {
10             U1 <- usuarios[i]
11             U2 <- usuarios[i+1]
12             A1 <- average[i]
13             A2 <- average[i+1]
14             # F rmula de interpolaci n lineal
15             U_target <- U1 + (average_target - A1)*(U2-U1)/(A2-A1)
16             return(U_target)
17         }
18     }
19     return(NA)
20 }
21
22 # Objetivo: tiempo promedio de 50 segundos (50000 ms)
23 average_obj <- 50000
24 U_avg <- interpolacion_average(average_obj, usuarios, average)
25 cat("Usuarios para tiempo promedio de 50s:", round(U_avg), "\n")
```

9.5.2. Método Regula Falsi para Encontrar Límites de Error

Listing 9.2: Método Regula Falsi para error objetivo

```
1  regula_falsi <- function(error_target, usuarios, error, tol=1e-3,
       maxiter=100) {
2      # Funci n que interpola el error para un n mero de usuarios
           dado
3      f <- function(U) {
4          for(i in 1:(length(usuarios)-1)) {
5              if(U >= usuarios[i] && U <= usuarios[i+1]) {
6                  # Interpolaci n lineal del error
7                  error_interp <- error[i] + (U - usuarios[i]) *
8                                  (error[i+1]-error[i])/(usuarios[i+1]-
                                       usuarios[i])
9                  return(error_interp - error_target)
10             }
11         }
```

```
        return(NA)
    }

    # Intervalo inicial [a, b]
    a <- min(usuarios)
    b <- max(usuarios)
    fa <- f(a)
    fb <- f(b)

    iter <- 0
    while(iter < maxiter) {
        # F rmula del m todo Regula Falsi
        c <- b - fb*(b - a)/(fb - fa)
        fc <- f(c)

        if(abs(fc) < tol) return(c)

        if(fc*fa < 0) {
            b <- c
            fb <- fc
        } else {
            a <- c
            fa <- fc
        }
        iter <- iter + 1
    }
    return(c)
}

# Objetivo: error m ximo del 5%
error_obj <- 5
U_err <- regula_falsi(error_obj, usuarios, error)

cat("Usuarios para error <= 5%:", round(U_err), "\n")
```

9.5.3. Resultados de Optimización Numérica

accent!40 **Método Numérico**	1lightpurplewhite **Objetivo**	**Usuarios Calculados**
Interpolación Lineal	Tiempo promedio $\leq$ 50 segundos	**333 usuarios**
Método Regula Falsi	Error $\leq 5\,\%$	**247 usuarios**

Tabla 9.3: Resultados de optimización numérica para YouTube

9.6. Análisis de Resultados y Recomendaciones

9.6.1. Interpretación para YouTube

Resultados: **Para YouTube:**

- **333 usuarios** mantienen tiempo promedio de 50 segundos (interpolación lineal)
- **247 usuarios** mantienen error $< 5\,\%$ (método Regula Falsi)
- El sistema muestra degradación progresiva pero manejable

Recomendación: **Recomendaciones operativas para YouTube:**

- **Zona segura:** Operar con máximo **247 usuarios** para garantizar calidad
- **Zona aceptable:** Entre 247-333 usuarios: tiempo aceptable pero con errores moderados
- **Zona crítica:** Sobre 333 usuarios: degradación severa del rendimiento
- **Optimizar LCP** de 5.0s a menos de 2.5s (mejorar Core Web Vitals)

9.6.2. Interpretación para Aula Virtual

Resultados: **Para Aula Virtual UNA Puno:**

- Muestra **saturación temprana** con aumento de usuarios
- **Error del 89 %** con 1000 usuarios (inaceptable)
- Throughput inconsistente bajo carga
- Buen desempeño en Core Web Vitals pero pobre escalabilidad

Recomendación: **Recomendaciones críticas para Aula Virtual:**

- **Urgente:** Mejoras de infraestructura del servidor
- Implementar **balanceadores de carga**
- **Optimización de base de datos** y consultas
- Considerar **caché de contenido** estático
- **Escalado vertical/horizontal** del servidor
- Establecer límite máximo de **200 usuarios concurrentes**

9.6.3. Comparación de Métodos Numéricos Aplicados

primary!40 **Aspecto**	**Interpolación Lineal**	**Método Regula Falsi**
Propósito	Estimación de valores intermedios	Búsqueda de raíces
Pregunta clave	¿Cuántos usuarios para tiempo X?	¿Dónde se cruza el umbral de error?
Fundamento matemático	$f(x) = f(x_0) + \frac{f(x_1)-f(x_0)}{x_1-x_0}(x-x_0)$	$x_n = x_1 - f(x_1)\frac{x_2-x_1}{f(x_2)-f(x_1)}$
Precisión	Depende de la linealidad de datos	Alta convergencia
Aplicación en estudio	Límite por tiempo de respuesta	Límite por porcentaje de error

2lightblue!30white

Tabla 9.4: Comparación de métodos numéricos aplicados al análisis de rendimiento

9.6.4. Recomendaciones Finales por Plataforma

9.6.5. Sinergia entre Métodos Numéricos y Herramientas de Prueba

Análisis Integral de Rendimiento Web

Combinación de herramientas prácticas con fundamentos matemáticos

secondary!30 **Plataforma**	**Recomendaciones Específicas**
YouTube	[leftmargin=*, noitemsep, topsep=2pt]
	✓ **Límite operativo:** 247 usuarios concurrentes
	✓ Implementar sistema de alertas al acercarse a 300 usuarios
	✓ Optimizar LCP: reducir de 5.0s a < 2.5s
	✓ Mejorar CLS: reducir de 0.41 a < 0.1
	✓ Monitoreo continuo con herramientas automáticas
1lightgreen!20white **Aula Virtual**	[leftmargin=*, noitemsep, topsep=2pt]
	▶ **Urgente:** Mejorar infraestructura del servidor
	▶ Implementar balanceador de carga inmediatamente
	▶ Optimizar consultas a base de datos (índices, caché)
	▶ Limitar usuarios concurrentes a 200 como máximo temporal
	✓ Mantener buen desempeño en Core Web Vitals
	✓ Implementar CDN para contenido estático

Tabla 9.5: Recomendaciones técnicas por plataforma

9.7. Conclusiones y Trabajo Futuro

9.7.1. Conclusiones Principales

- **YouTube** muestra mayor tolerancia a la carga, siendo capaz de manejar hasta 247 usuarios con error $< 5\,\%$
- **Aula Virtual UNA Puno** presenta saturación temprana, requiriendo mejoras urgentes de infraestructura
- La combinación de **Apache JMeter** + **Google Lighthouse** proporciona una visión integral del rendimiento
- Los **métodos numéricos** (Interpolación y Regula Falsi) permiten estimar puntos críticos con precisión
- La **interpolación numérica** es efectiva para predecir comportamientos bajo diferentes cargas
- El **método Regula Falsi** es robusto para encontrar umbrales operativos

secondary!30 **Plataforma**	**Recomendaciones Específicas**
YouTube	[leftmargin=*, noitemsep, topsep=2pt]
	✓ **Límite operativo:** 247 usuarios concurrentes
	✓ Implementar sistema de alertas al acercarse a 300 usuarios
	✓ Optimizar LCP: reducir de 5.0s a < 2.5s
	✓ Mejorar CLS: reducir de 0.41 a < 0.1
	✓ Monitoreo continuo con herramientas automáticas
1lightgreen!20white **Aula Virtual**	[leftmargin=*, noitemsep, topsep=2pt]
	▶ **Urgente:** Mejorar infraestructura del servidor
	▶ Implementar balanceador de carga inmediatamente
	▶ Optimizar consultas a base de datos (índices, caché)
	▶ Limitar usuarios concurrentes a 200 como máximo temporal
	✓ Mantener buen desempeño en Core Web Vitals
	✓ Implementar CDN para contenido estático

Tabla 9.6: Recomendaciones técnicas por plataforma

9.7.2. Recomendaciones Finales por Plataforma

9.7.3. Trabajo Futuro e Investigación

- **Extender análisis** a otras plataformas educativas y comparar resultados
- **Implementar monitoreo** continuo en tiempo real con alertas automáticas
- **Desarrollar dashboard** interactivo para administradores con predicciones
- **Profundizar análisis** con machine learning para predicción de carga
- **Automatizar pruebas** de rendimiento periódicas con integración continua
- **Estudiar optimización** con más métodos numéricos (Newton, secante, etc.)
- **Publicar resultados** en conferencias de ingeniería de software

9.7.4. Impacto del Estudio

9.7.5. Lecciones Aprendidas

1. **Importancia de los datos empíricos**: Las pruebas de carga proporcionan información real sobre el comportamiento del sistema
2. **Valor de los métodos numéricos**: Permiten extrapolar y predecir comportamientos sin necesidad de pruebas exhaustivas
3. **Sinergia herramientas-métodos**: La combinación de herramientas prácticas con fundamentos matemáticos es poderosa
4. **Decisiones basadas en evidencia**: Los resultados numéricos objetivizan las decisiones técnicas
5. **Optimización preventiva**: Es más eficiente prevenir problemas que corregirlos

Reflexión Final: Este estudio demuestra cómo los métodos numéricos aprendidos en los capítulos anteriores (diferenciación, interpolación, optimización) encuentran aplicación directa en problemas del mundo real. La optimización de páginas web no es solo arte, sino ciencia aplicada que combina:

- **Datos empíricos** de pruebas de carga
- **Métricas de calidad** de auditorías web
- **Análisis predictivo** con métodos numéricos
- **Toma de decisiones** basada en evidencia cuantitativa

La matemática aplicada sigue siendo tan relevante hoy como lo fue en los tiempos de Newton, solo que ahora optimizamos páginas web en lugar de trayectorias planetarias.

Bibliografía

Bibliografía

[1] Burden, R. L., & Faires, J. D. (2010). *Numerical Analysis* (9th ed.). Brooks/Cole.

[2] Chapra, S. C., & Canale, R. P. (2015). *Numerical Methods for Engineers* (7th ed.). McGraw-Hill.

[3] Heath, M. T. (2018). *Scientific Computing: An Introductory Survey* (2nd ed.). SIAM.

[4] Golub, G. H., & Van Loan, C. F. (2013). *Matrix Computations* (4th ed.). Johns Hopkins University Press.

[5] Trefethen, L. N., & Bau, D. (1997). *Numerical Linear Algebra*. SIAM.

[6] Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems* (2nd ed.). SIAM.

[7] Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization* (2nd ed.). Springer.

[8] Boyd, S., & Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

[9] Quarteroni, A., Sacco, R., & Saleri, F. (2007). *Numerical Mathematics* (2nd ed.). Springer.

[10] Demmel, J. W. (1997). *Applied Numerical Linear Algebra*. SIAM.

[11] Apache Software Foundation. (2023). *Apache JMeter User Manual*. Apache Foundation.

[12] Google Developers. (2023). *Lighthouse: Automated Website Audits*. Google.

[13] Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). Cambridge University Press.