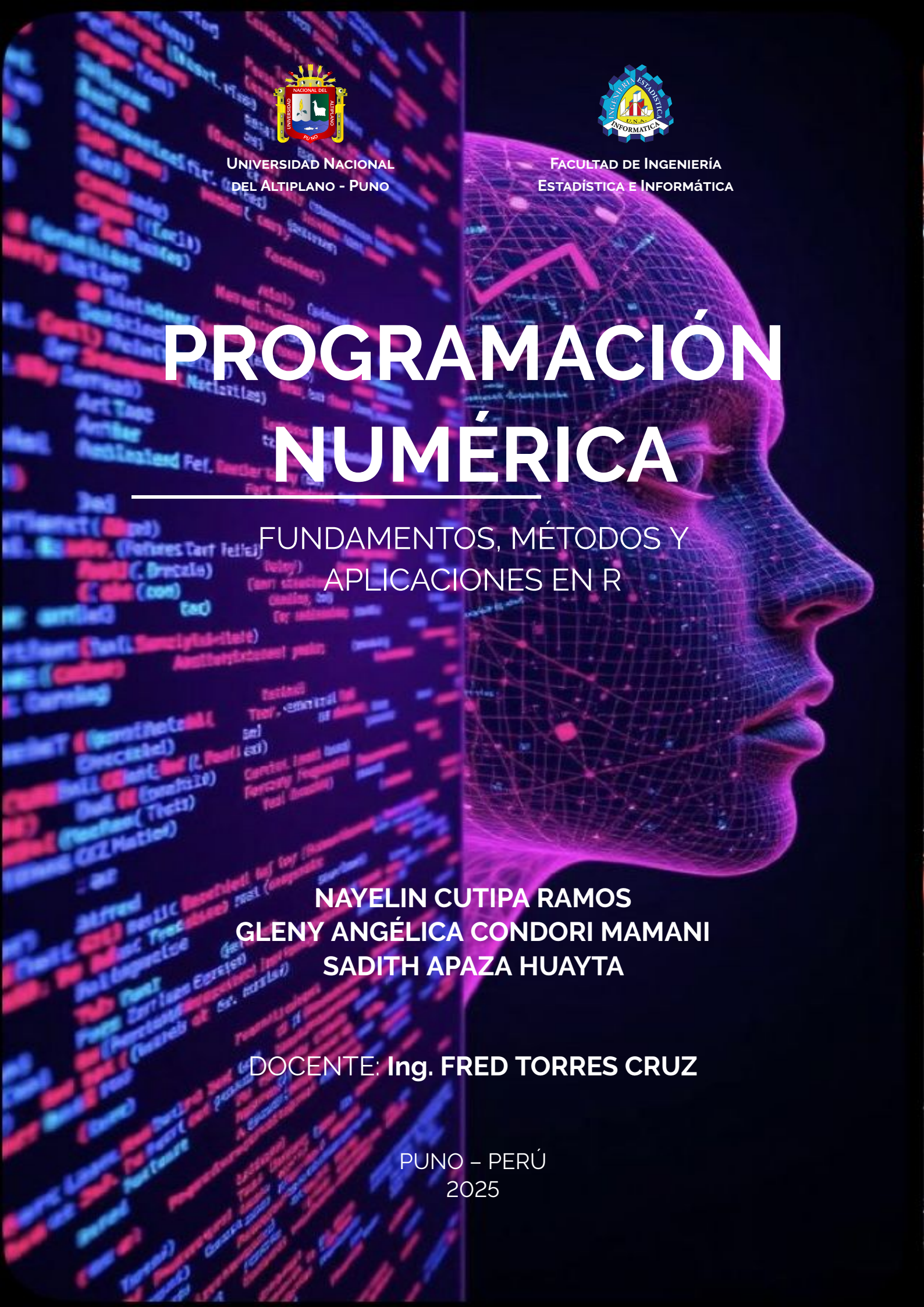


UNIVERSIDAD NACIONAL
DEL ALTIPLANO - PUNO

FACULTAD DE INGENIERÍA
ESTADÍSTICA E INFORMÁTICA

PROGRAMACIÓN NUMÉRICA

FUNDAMENTOS, MÉTODOS Y APLICACIONES EN R

NAYELIN CUTIPA RAMOS
GLENY ANGÉLICA CONDORI MAMANI
SADITH APAZA HUAYTA

DOCENTE: **Ing. FRED TORRES CRUZ**

PUNO – PERÚ
2025

Título de la obra:

Programación Numérica: Fundamentos, Métodos y Aplicaciones en R

Autores:

Nayelin Brisbany Cutipa Ramos

Gleny Angélica Condori Mamani

Sadith lina Apaza Huayta

Curso: Programación Numérica

Escuela Profesional: Ingeniería Estadística e Informática

Institución: Universidad Nacional del Altiplano

Advertencia de uso educativo

El contenido de este libro está destinado exclusivamente al uso académico y formativo. Los métodos, algoritmos, ejemplos y códigos presentados han sido desarrollados con fines didácticos, por lo que su aplicación en contextos profesionales reales debe realizarse bajo la supervisión de especialistas y considerando las condiciones particulares de cada problema.

La Universidad Nacional del Altiplano no se responsabiliza por el uso indebido de la información contenida en esta obra.

Lugar y año de publicación:

Puno – Perú

2025

Este libro está dedicado, en primer lugar, a nuestras familias, por su apoyo constante, comprensión y motivación a lo largo de nuestra formación universitaria.

Asimismo, dedicamos este trabajo a nuestros docentes, quienes con su enseñanza, exigencia académica y orientación han contribuido de manera significativa a nuestro desarrollo profesional.

Finalmente, dedicamos esta obra a todos los estudiantes de la carrera de Ingeniería Estadística e Informática que, con esfuerzo y perseverancia, buscan fortalecer sus conocimientos en Programación Numérica y áreas afines, como parte de su formación académica y científica.

Agradecimientos

Las autoras del presente libro expresan su más sincero agradecimiento a la Universidad Nacional del Altiplano, por brindar la formación académica y los recursos necesarios que hicieron posible el desarrollo de este trabajo.

De manera especial, agradecemos al Ing. Torres Cruz, Fred, docente del curso de Programación Numérica, por su orientación académica, exigencia constante y valiosos aportes durante el desarrollo del semestre, los cuales han sido fundamentales para la consolidación de los conocimientos presentados en esta obra.

Asimismo, extendemos nuestro agradecimiento a la Escuela Profesional de Ingeniería Estadística e Informática, por fomentar la investigación, el pensamiento crítico y el uso de herramientas computacionales aplicadas a la solución de problemas reales.

Finalmente, hacemos un reconocimiento especial a los integrantes del **Grupo .^"(Semestre 2024-II)**, quienes formaron parte del entorno académico y colaborativo en el cual se gestó este trabajo final:

- Aguilar Ccopa, Leydy Griselda
- Apaza Curtihuanca, Job Edward
- Apaza Huayta, Sadith Lina
- Aquino Sandoval, Jean Carlos
- Caira Huancollo, Wily Calib
- Ccoarite Dueñas, Henry
- Churata Mamani, Milena Kely
- Condori Mamani, Gleny Angelica
- Cutipa Ramos, Nayelin Brisbany
- Deza Cuela, Angeline Santamaria
- Incacutipa Muñuico, Ronald Wilder
- Mamani Apaza, Jhon Gilmer
- Mamani Yucra, Nexu Yohan
- Paricahua Pari, Clyde Neil
- Paucar Yupanqui, Cristian Ronaldo
- Puma Huanca, Anthony Rusbel
- Quenaya Loza, Luis Angel
- Quilca Mestas, Maria Anabel
- Quispe Coaguila, Yair Dylan
- Quispe Ito, Luz Leidy
- Quispe Ramos, Henrry Higinio
- Ticona Miramira, Roberto Angel

Puno, 2025

Prólogo

La Programación Numérica constituye una de las áreas fundamentales dentro de la formación de los ingenieros, debido a su capacidad para proporcionar herramientas matemáticas y computacionales orientadas a la solución de problemas reales que, en muchos casos, no admiten una solución analítica exacta. En un contexto donde el uso de datos, modelos matemáticos y algoritmos computacionales es cada vez más relevante, el dominio de los métodos numéricos se convierte en una competencia esencial para el ejercicio profesional.

El presente libro ha sido elaborado como resultado del trabajo académico desarrollado a lo largo del semestre en el curso de Programación Numérica, perteneciente a la carrera de Ingeniería Estadística e Informática de la Universidad Nacional del Altiplano. Su propósito principal es consolidar, organizar y profundizar los conocimientos adquiridos durante el desarrollo del curso, integrando los fundamentos teóricos con su correspondiente implementación computacional mediante el lenguaje de programación R.

Uno de los principales objetivos de esta obra es presentar los métodos numéricos de manera clara, progresiva y aplicada, permitiendo al lector comprender no solo el fundamento matemático de cada técnica, sino también su utilidad práctica en la resolución de problemas de ingeniería. Para ello, cada tema ha sido desarrollado siguiendo una estructura homogénea que incluye la introducción conceptual, el desarrollo matemático, el algoritmo numérico, ejemplos resueltos, implementación en R y ejercicios propuestos.

La elección del lenguaje de programación R responde a su amplia aceptación en el ámbito académico y profesional, especialmente en áreas relacionadas con la estadística, el análisis de datos y la computación científica. R ofrece un entorno flexible y potente para la implementación de algoritmos numéricos, la visualización de resultados y la experimentación con modelos matemáticos, lo que lo convierte en una herramienta idónea para el aprendizaje y la aplicación de la Programación Numérica.

El contenido del libro abarca desde conceptos fundamentales, como variables, funciones y métodos iterativos para el cálculo de raíces, hasta temas más avanzados, tales como métodos de optimización, pseudoinversas, diferenciación e interpolación numérica, valores propios, cadenas de Markov y herramientas de análisis y evaluación del rendimiento computacional. Esta diversidad temática permite al lector obtener una visión integral de la Programación Numérica y su

aplicación en distintos contextos.

La elaboración de esta obra ha sido realizada de manera colaborativa por las autoras, distribuyendo equitativamente los temas abordados y manteniendo un criterio uniforme en cuanto a estilo, notación y nivel de profundidad. Este enfoque colaborativo refleja el trabajo en equipo y la integración de conocimientos, habilidades y experiencias adquiridas durante la formación universitaria.

El presente libro está dirigido principalmente a estudiantes de Ingeniería Estadística e Informática y carreras afines, así como a docentes y profesionales interesados en fortalecer sus conocimientos en métodos numéricos y su implementación computacional. No se asume un conocimiento previo avanzado, por lo que los temas han sido desarrollados de forma progresiva, facilitando su comprensión y aplicación.

Finalmente, se espera que este libro sirva como material de apoyo académico, guía de estudio y referencia básica para el aprendizaje de la Programación Numérica, contribuyendo al desarrollo del pensamiento analítico, la capacidad de modelación matemática y el uso eficiente de herramientas computacionales en la solución de problemas reales.

Índice general

Índice de figuras

Capítulo 1

Variables y Funciones en Programación Numérica con R

1.1. Introducción

La Programación Numérica surge como una respuesta a la imposibilidad práctica de resolver muchos problemas matemáticos de forma analítica. En ingeniería, ciencias aplicadas y estadística, los modelos matemáticos suelen involucrar ecuaciones no lineales, sistemas de gran tamaño o funciones complejas que requieren aproximaciones computacionales. En este contexto, las variables y funciones constituyen los pilares fundamentales sobre los cuales se construyen todos los métodos numéricos.

Una variable representa una magnitud susceptible de cambio, mientras que una función describe la relación existente entre una o más variables. En programación numérica, estas nociones adquieren una dimensión computacional: las variables se almacenan en memoria y se actualizan iterativamente, y las funciones se implementan como entidades programables que pueden evaluarse, derivarse o aproximarse numéricamente.

El lenguaje R se ha consolidado como una herramienta poderosa para la programación numérica y el análisis estadístico debido a su sintaxis clara, su orientación matemática y su capacidad gráfica. A diferencia de otros lenguajes, R permite expresar conceptos matemáticos de forma casi directa, lo que facilita el aprendizaje y la correcta implementación de algoritmos numéricos.

Este capítulo tiene como objetivo establecer las bases conceptuales, matemáticas y computacionales necesarias para comprender el uso de variables y funciones en programación numérica. Estos conceptos serán utilizados de manera recurrente en capítulos posteriores, como el cálculo de raíces, la optimización, la interpolación y la modelación estadística.

1.2. Fundamentos Teóricos

1.2.1. Variable: definición matemática y computacional

Desde el punto de vista matemático, una variable es un símbolo que representa un elemento de un conjunto numérico. En programación, una variable es una referencia a una posición de memoria donde se almacena un valor que puede modificarse durante la ejecución de un programa.

Formalmente, una variable puede representarse como:

$$v = (n, d, t)$$

donde:

- n es el nombre o identificador,
- d es el dominio,
- t es el tipo de dato.

En programación numérica, las variables suelen pertenecer a dominios continuos o discretos, y su correcta definición es crucial para evitar errores de cálculo y de interpretación (EBAC, s.f.).

1.2.2. Tipos de variables en R

R es un lenguaje de tipado dinámico, lo que significa que el tipo de una variable se asigna automáticamente según el valor que contiene. Los tipos más relevantes en programación numérica son:

- **Numeric:** números reales.
- **Integer:** números enteros.
- **Complex:** números complejos.
- **Logical:** valores booleanos.

```r
x <- 3.5
y <- 2L
z <- 1 + 2i
```

Listing 1.1: Declaración de variables en R

En métodos numéricos, las variables numéricas son utilizadas para almacenar aproximaciones sucesivas, errores y parámetros de control.

1.2.3. Funciones matemáticas

Una función matemática es una relación que asigna a cada elemento de un conjunto de partida exactamente un elemento de un conjunto de llegada:

$$f : X \to Y$$

Las funciones permiten modelar fenómenos físicos, económicos y estadísticos. En programación numérica, las funciones suelen representar:

- Ecuaciones no lineales.
- Funciones objetivo.
- Modelos de simulación.

1.2.4. Funciones en R

En R, una función es un objeto de primera clase. Esto significa que puede asignarse a una variable, pasarse como argumento y devolverse como resultado.

Estructura general:

```r
f <- function(x) {
    x^2 - 4
}
```

Listing 1.2: Definición de función en R

Esta característica es esencial para implementar algoritmos iterativos como Newton-Raphson o Descenso de Gradiente (IBM, 2023).

1.3. Desarrollo Matemático

1.3.1. Variables en métodos iterativos

En métodos numéricos, una variable no representa un valor fijo, sino una sucesión de aproximaciones:

$$x_0, x_1, x_2, \ldots, x_n$$

Por ejemplo, en el método de Newton-Raphson:

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

Cada iteración actualiza la variable hasta alcanzar una tolerancia previamente establecida (Burden & Faires, 2011).

1.3.2. Dominio, continuidad y derivabilidad

Para aplicar correctamente un método numérico, es necesario analizar:

- Dominio de la función.
- Continuidad.
- Existencia de derivadas.

Una función no continua puede provocar divergencia o resultados erróneos.

1.4. Algoritmo Numérico

1.4.1. Idea general

Todo algoritmo numérico basado en funciones sigue este esquema:

1. Definir variables iniciales.
2. Definir la función matemática.
3. Evaluar la función.
4. Actualizar variables.
5. Verificar criterio de parada.

1.4.2. Pseudocódigo general

Pseudocódigo: Estructura Iterativa

Entrada: x_0, Tolerancia, Función $f(x)$
Inicio:

- Mientras error $>$ tolerancia:
  - Calcular nuevo valor de x
  - Actualizar error
- Mostrar resultado

Fin

1.5. Ejemplos Resueltos

Ejemplo 1: Análisis de función cuadrática

Problema:
Sea la función:

$$f(x) = x^2 - 4$$

Evaluación manual:

$$f(2) = 0, \quad f(-2) = 0$$

Las raíces se encuentran en $x = \pm 2$.
Implementación en R:

```
f <- function(x) {
    x^2 - 4
}

f(2)
f(-2)
```

Gráfica en R:

```
curve(f(x), from = -3, to = 3,
main = "f(x) = x^2 - 4",
xlab = "x", ylab = "f(x)")
abline(h = 0)
```

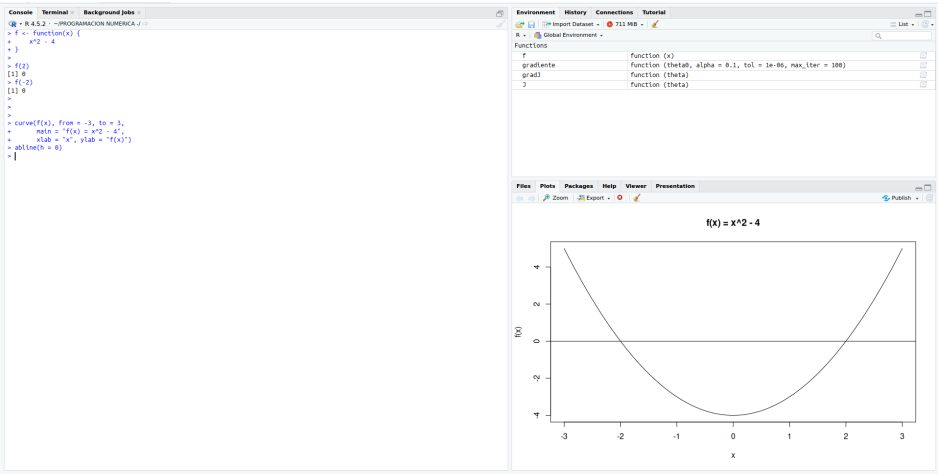


Figura 1.1: Gráfica de la función cuadrática generada en R.

Ejemplo 2: Función exponencial

Manual:

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

Implementación en R:

```r
f <- function(x) {
    exp(x) - 3
}

curve(f(x), from = 0, to = 2)
abline(h = 0)
```

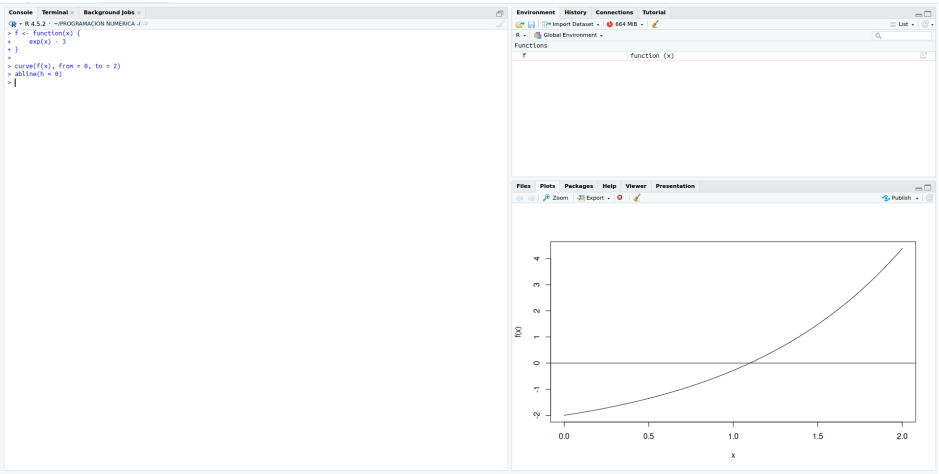


Figura 1.2: Gráfica de la función exponencial generada en R.

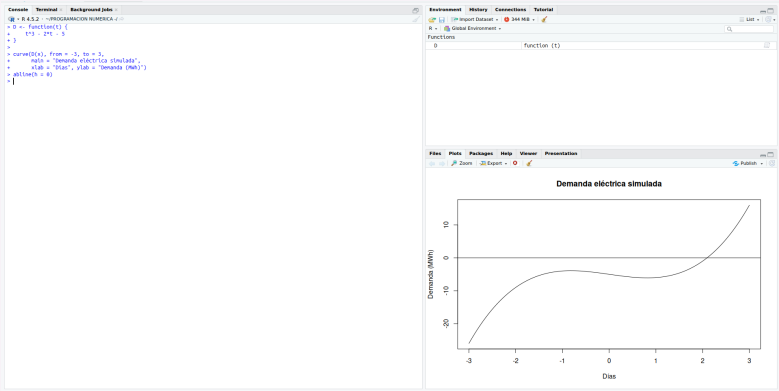
Ejercicio Aplicado 1: Demanda eléctrica

Contexto: Una empresa eléctrica de la región de Puno modela la demanda diaria de energía (en MWh) mediante la función:

$$D(t) = t^3 - 2t - 5$$

donde t representa el número de días desde el inicio del mes.
Análisis matemático: Se desea analizar el comportamiento de la demanda e identificar visualmente si existen puntos donde la demanda cambia de signo, lo cual será relevante para estudios de estabilidad energética.
Implementación en R:

```r
D <- function(t) {
    t^3 - 2*t - 5
}

curve(D(x), from = -3, to = 3,
main = "Demanda␣eléctrica␣simulada",
xlab = "Días", ylab = "Demanda␣(MWh)")
abline(h = 0)
```



Interpretación: La gráfica permite identificar intervalos donde la demanda es positiva o negativa, información clave para la planificación energética regional.

1.6. Aplicaciones Prácticas

Las variables y funciones son esenciales en:

- Modelos de regresión.
- Simulaciones Monte Carlo.
- Optimización de costos.
- Análisis estadístico.

Por ejemplo, una función de costo puede representarse como (Mínimos Cuadrados Ordinarios; Investopedia, 2023):

$$J(\theta) = \sum (y_i - \hat{y}_i)^2$$

1.7. Ejercicios Resueltos

Ejercicio Resuelto 1

Problema:
Analizar la función $f(x) = x^3 - 2x - 5$.
Solución manual:
Identificar dominio, evaluar valores y analizar comportamiento.
Implementación en R:

```
f <- function(x) {
    x^3 - 2*x - 5
}

curve(f(x), from = -3, to = 3)
abline(h = 0)
```

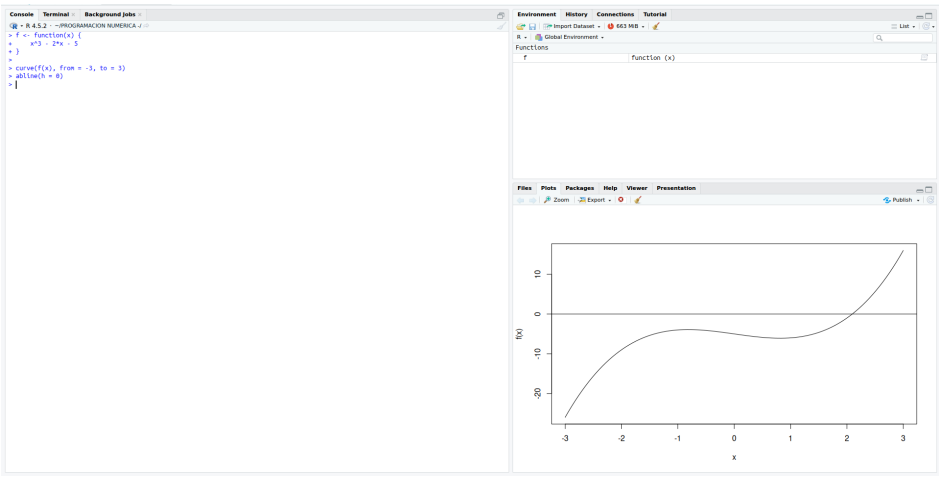


Figura 1.3: Comportamiento de la función cúbica.

1.8. Ejercicios Propuestos

1. **Producción agrícola en el Altiplano** Una cooperativa agrícola modela la producción diaria de quinua (en toneladas) como:

$$P(t) = t^2 + 3t - 10$$

donde t es el tiempo en semanas.
   - Defina la función en R.
   - Grafique la producción.
   - Interprete los valores negativos.

2. **Costo de transporte lacustre** El costo de transporte entre islas del Lago Titicaca se modela mediante:

$$C(x) = \ln(x - 2)$$

donde x representa la distancia en kilómetros.
- Determine el dominio.
- Grafique la función.
- Explique por qué ciertos valores no son físicamente válidos.

3. **Temperatura ambiental** La variación de temperatura diaria se aproxima con:

$$T(t) = 15 + 5\sin(t)$$

- Defina la función.
- Grafique un ciclo completo.
- Interprete máximos y mínimos.

4. **Ingreso económico turístico** El ingreso mensual por turismo se modela como:

$$I(n) = 1000n - 50n^2$$

donde n es el número de tours ofrecidos.
- Analice la función.
- Grafique el ingreso.
- Discuta implicaciones económicas.

5. **Relación con métodos numéricos** Explique por qué el análisis gráfico de funciones es un paso previo fundamental antes de aplicar métodos de búsqueda de raíces u optimización.

Capítulo 2

Visualización Numérica de Funciones Matemáticas en R

2.1. Introducción

La visualización gráfica constituye una herramienta esencial en la Programación Numérica, ya que permite interpretar, analizar y validar el comportamiento de funciones matemáticas y resultados numéricos obtenidos mediante algoritmos computacionales. En muchos casos, una representación gráfica adecuada facilita la comprensión de un problema más eficazmente que una expresión analítica compleja.

En ingeniería estadística e informática, la visualización numérica se utiliza para identificar raíces de ecuaciones, analizar tendencias, detectar errores numéricos, estudiar la convergencia de métodos iterativos y comunicar resultados de forma clara y precisa. Por esta razón, la graficación de funciones no debe considerarse un complemento opcional, sino una etapa fundamental del análisis numérico.

El lenguaje R ofrece una amplia variedad de herramientas gráficas que permiten representar funciones matemáticas de manera eficiente y flexible. A través de funciones como `plot()` y `curve()`, es posible visualizar funciones continuas, discretas y aproximaciones numéricas con un alto nivel de detalle y control.

Este capítulo tiene como objetivo presentar los fundamentos teóricos y prácticos de la visualización numérica en R, enfatizando su aplicación en problemas de programación numérica. Asimismo, se desarrollan ejemplos completos y ejercicios resueltos que integran el análisis matemático tradicional con su implementación computacional.

2.2. Fundamentos Teóricos

2.2.1. Importancia de la visualización en programación numérica

En programación numérica, la visualización gráfica cumple múltiples funciones, entre las cuales destacan:

- Identificar raíces aproximadas de una función.
- Analizar la continuidad y el comportamiento global de una función.
- Detectar oscilaciones o inestabilidades numéricas.
- Comparar soluciones analíticas y aproximadas.
- Validar resultados obtenidos mediante métodos iterativos.

Según R Charts (s.f.), la visualización es una de las formas más directas de interpretar el comportamiento dinámico de funciones matemáticas.

2.2.2. Representación gráfica de funciones

Sea una función real de variable real:

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

Su gráfica es el conjunto de puntos:

$$G(f) = \{(x, f(x)) \mid x \in \mathbb{R}\}$$

La representación gráfica permite estudiar propiedades como:

- Crecimiento y decrecimiento.
- Máximos y mínimos.
- Puntos de inflexión.
- Intersecciones con los ejes.

2.2.3. Tipos de gráficos en programación numérica

En R, los gráficos utilizados en programación numérica pueden clasificarse en:

- Gráficos de funciones continuas.
- Gráficos de datos discretos.
- Gráficos comparativos.
- Gráficos de convergencia.

Cada uno cumple un rol específico en el análisis numérico.

2.3. Desarrollo Matemático

2.3.1. Dominio y rango en la visualización

Antes de graficar una función, es imprescindible analizar su dominio matemático. Por ejemplo, la función:

$$f(x) = \ln(x - 1)$$

solo está definida para:

$$x > 1$$

Graficar una función fuera de su dominio puede generar errores computacionales o interpretaciones incorrectas.

2.3.2. Discretización del dominio

En la práctica computacional, las funciones continuas se representan mediante una discretización del dominio:

$$x_1, x_2, \ldots, x_n$$

y se evalúa:

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

Este proceso introduce errores de aproximación que deben ser considerados en el análisis numérico (ULPGC, s.f.).

2.3.3. Visualización y análisis de raíces

Una de las aplicaciones más importantes de la visualización numérica es la localización gráfica de raíces, es decir, los valores de x tales que:

$$f(x) = 0$$

El análisis gráfico permite seleccionar intervalos adecuados para métodos como bisección o falsa posición.

2.4. Algoritmo Numérico de Graficación

2.4.1. Idea general

El proceso de graficación numérica de una función sigue los siguientes pasos:

1. Definir la función matemática.
2. Seleccionar el intervalo de visualización.
3. Evaluar la función en el intervalo.
4. Representar gráficamente los resultados.
5. Interpretar el comportamiento observado.

2.4.2. Pseudocódigo general

Pseudocódigo: Graficación

Entrada: Función $f(x)$, Intervalo $[a, b]$
Proceso:

- Generar valores de x en el intervalo
- Evaluar $f(x)$ para cada x
- Graficar par $(x, f(x))$
- Analizar resultados visuales

Fin

2.5. Ejemplos Resueltos

Ejemplo 1: Análisis de función cuadrática

Problema:
Sea la función:

$$f(x) = x^2 - 4$$

Análisis matemático:

- Dominio: $\mathbb{R}$
- Raíces: $x = \pm 2$
- Función continua y derivable.

Implementación en R:

```
f <- function(x) {
    x^2 - 4
}

curve(f(x), from = -3, to = 3,
main = "Grafica de f(x) = x^2 - 4",
xlab = "x", ylab = "f(x)")
abline(h = 0)
```

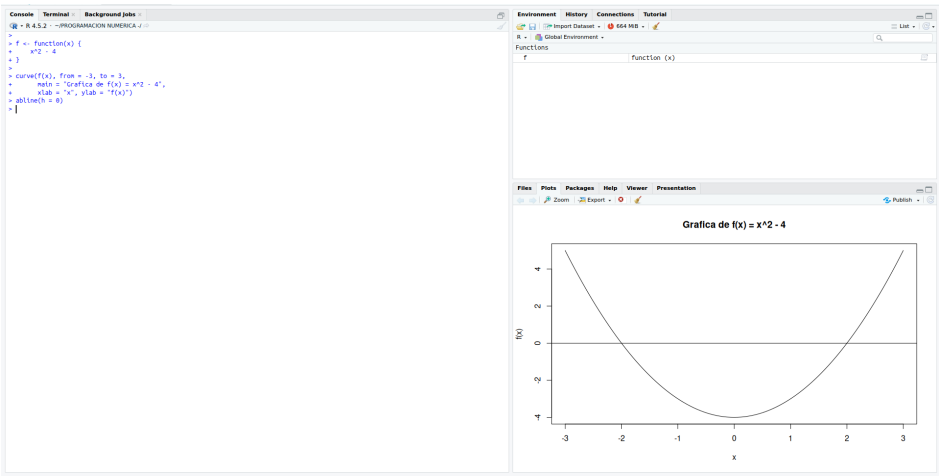


Figura 2.1: Gráfica de la función cuadrática mostrando sus raíces.

Ejemplo 2: Función exponencial

Análisis manual:

$$f(x) = e^x - 2$$

Raíz aproximada:

$$x = \ln(2) \approx 0.693$$

Implementación en R:

```
f <- function(x) {
    exp(x) - 2
}

curve(f(x), from = 0, to = 2,
main = "Grafica de f(x) = e^x - 2",
xlab = "x", ylab = "f(x)")
abline(h = 0)
```

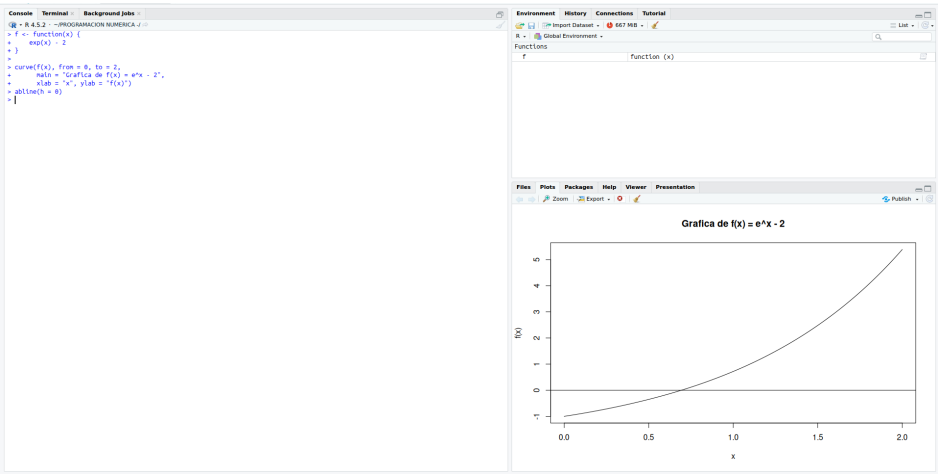


Figura 2.2: Gráfica de la función exponencial.

Ejemplo 3: Función logarítmica

Manual:

$$f(x) = \ln(x)$$

Dominio: $x > 0$

Implementación en R:

```
f <- function(x) {
    log(x)
}

curve(f(x), from = 0.1, to = 4,
main = "Grafica de f(x) = ln(x)",
xlab = "x", ylab = "f(x)")
```

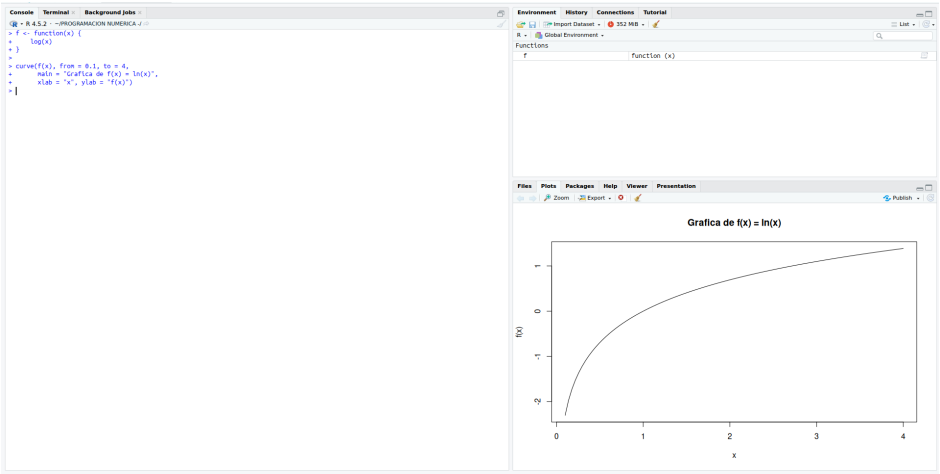


Figura 2.3: Gráfica de la función logarítmica.

2.6. Aplicaciones Prácticas

La visualización numérica se aplica en múltiples contextos reales, tales como:

- Selección de intervalos iniciales para métodos de cálculo de raíces en problemas de ingeniería.
- Análisis de funciones objetivo en procesos de optimización económica e industrial.
- Evaluación gráfica del ajuste de modelos estadísticos a datos reales.
- Validación visual de resultados numéricos antes de su interpretación final.

En regresión por mínimos cuadrados, la comparación gráfica entre datos observados y valores ajustados es esencial para evaluar la calidad del modelo y detectar posibles anomalías [5].

2.7. Ejercicios Resueltos

Ejercicio Resuelto 1

Problema:
Graficar y analizar la función:

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

Análisis manual:

- Dominio: $\mathbb{R}$
- Función continua.
- Existe al menos una raíz real.

Implementación en R:

```
f <- function(x) {
    x^3 - x - 1
}

curve(f(x), from = -2, to = 2,
main = "f(x) = x^3 - x - 1",
xlab = "x", ylab = "f(x)")
abline(h = 0)
```

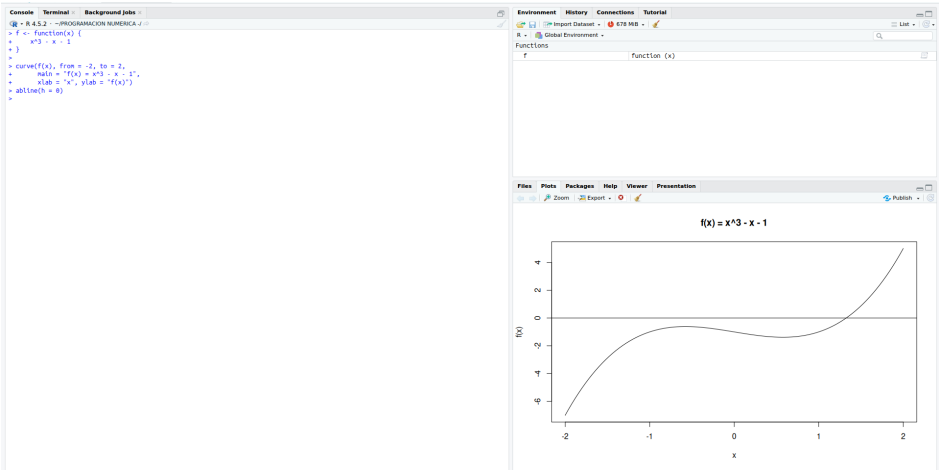


Figura 2.4: Análisis gráfico de la función cúbica.

Ejercicio Resuelto 2: Comparación de funciones

Problema:
Comparar dos funciones en un mismo gráfico.

$$f(x) = x^2, \quad g(x) = 2x + 1$$

Implementación en R:

```
f <- function(x) x^2
g <- function(x) 2*x + 1

curve(f(x), from = -3, to = 3, col = "blue",
ylab = "y", xlab = "x")
curve(g(x), add = TRUE, col = "red")
legend("topleft", legend = c("x^2", "2x+1"),
col = c("blue", "red"), lty = 1)
```

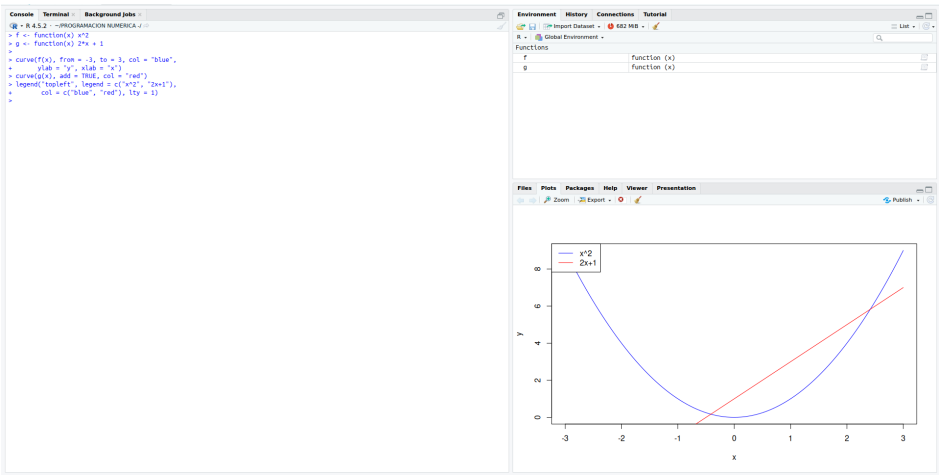


Figura 2.5: Comparación gráfica de una parábola y una recta.

Ejercicio Resuelto 1: Consumo de combustible

Contexto: El consumo aproximado de combustible de una embarcación turística en el Lago Titicaca puede modelarse mediante la función:

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

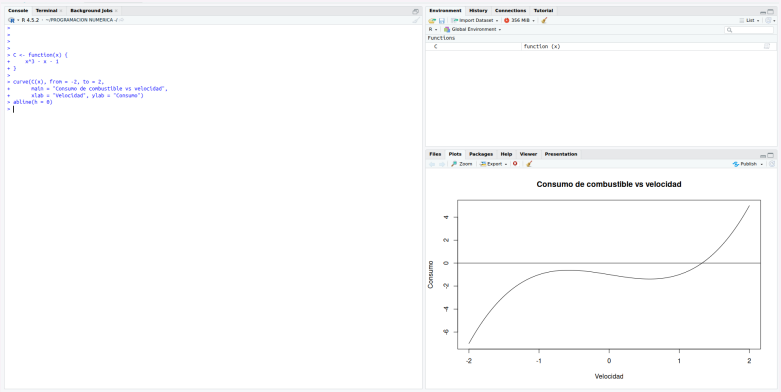
donde x representa la velocidad de la embarcación.

Análisis matemático:

- Dominio: $\mathbb{R}$
- Función continua.
- La visualización permite identificar velocidades críticas donde el consumo cambia de comportamiento.

Implementación en R:

```
C <- function(x) {
    x^3 - x - 1
}

curve(C(x), from = -2, to = 2,
main = "Consumo de combustible vs velocidad",
xlab = "Velocidad", ylab = "Consumo")
abline(h = 0)
```



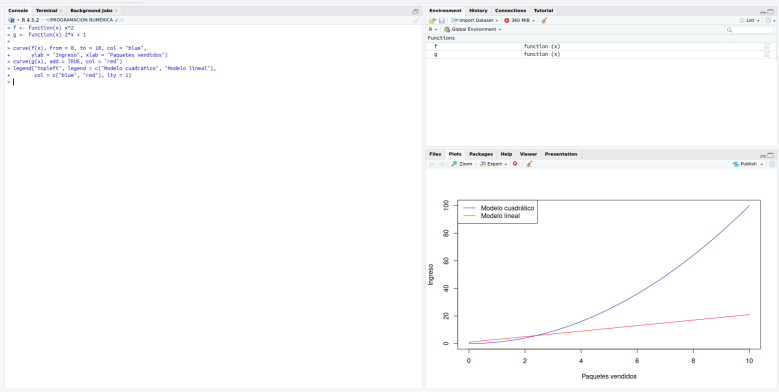
Ejercicio Resuelto 2: Comparación de ingresos

Contexto: Dos modelos estiman el ingreso mensual de una empresa turística:

$$f(x) = x^2, \quad g(x) = 2x + 1$$

donde x representa el número de paquetes turísticos vendidos.
Interpretación: La visualización conjunta permite comparar qué modelo genera mayores ingresos según el nivel de ventas.
Implementación en R:

```r
f <- function(x) x^2
g <- function(x) 2*x + 1

curve(f(x), from = 0, to = 10, col = "blue",
ylab = "Ingreso", xlab = "Paquetes vendidos")
curve(g(x), add = TRUE, col = "red")
legend("topleft", legend = c("Modelo cuadrático"
    , "Modelo lineal"),
col = c("blue", "red"), lty = 1)
```



2.8. Ejercicios Propuestos

1. **Producción agrícola** La producción semanal de papa en una comunidad del Altiplano se modela como:
$$P(t) = t^2 + 2t - 3$$
Grafique la función, analice sus raíces e interprete su significado económico.
2. **Enfriamiento de un sistema** La temperatura de un motor después de apagarse se aproxima mediante:
$$T(t) = e^{-t}$$
Grafique la función en el intervalo $[0, 5]$ y describa el comportamiento observado.
3. **Costo de transporte** El costo por kilómetro de transporte se modela con una función racional. Defina una función adecuada, grafíquela y analice su dominio.
4. **Fenómeno periódico** Modele la variación diaria de temperatura usando una función trigonométrica y represéntela gráficamente.

5. **Tarifas por tramos** Defina y grafique una función por tramos que modele el cobro de energía eléctrica según consumo.
6. **Discontinuidad** Construya una función no continua, grafíquela y explique las implicancias físicas de dicha discontinuidad.
7. **Modelo con parámetro** Grafique una función que dependa de un parámetro y analice cómo cambia su comportamiento.
8. **Convergencia gráfica** Represente gráficamente la convergencia de una sucesión numérica.
9. **Función de costo** Visualice una función de costo típica usada en optimización y explique su importancia.
10. **Relación con optimización** Explique por qué la visualización es un paso previo fundamental antes de aplicar métodos de optimización numérica.

Capítulo 3

Restricciones y Modelamiento Matemático en Programación Numérica con R

3.1. Introducción

En la Programación Numérica y en la Ingeniería Estadística e Informática, los problemas reales rara vez se presentan de forma libre o sin limitaciones. Por el contrario, la mayoría de situaciones prácticas involucran restricciones, las cuales delimitan el conjunto de soluciones admisibles de un modelo matemático. Estas restricciones pueden representar límites físicos, económicos, tecnológicos o estadísticos, y su correcta formulación es esencial para obtener soluciones realistas y útiles.

El modelamiento matemático con restricciones permite traducir un problema del mundo real a un lenguaje formal basado en variables, funciones y condiciones que deben cumplirse simultáneamente. En este proceso, las restricciones juegan un papel fundamental, ya que definen el espacio factible donde se buscará la solución numérica óptima o aproximada.

Desde el punto de vista computacional, el tratamiento de restricciones influye directamente en la elección del método numérico, la estabilidad del algoritmo y la interpretación de los resultados. En particular, los métodos de optimización, regresión y resolución de ecuaciones dependen críticamente de cómo se expresan y gestionan dichas restricciones.

El lenguaje R ofrece herramientas flexibles para trabajar con modelos matemáticos restringidos, ya sea mediante condiciones lógicas, funciones objetivo condicionadas o técnicas de optimización con restricciones explícitas. Este capítulo desarrolla los fundamentos teóricos y prácticos del manejo de restricciones en programación numérica, estableciendo las bases para los métodos de optimización y ajuste de modelos que se estudiarán en capítulos posteriores.

3.2. Fundamentos Teóricos

3.2.1. Concepto de restricción

Una restricción es una condición matemática que limita los valores que pueden tomar las variables de un problema. Formalmente, una restricción puede expresarse como:

Restricción de igualdad:

$$g(x) = 0$$

Restricción de desigualdad:

$$h(x) \leq 0 \quad \text{o} \quad h(x) \geq 0$$

Estas condiciones definen el conjunto factible, es decir, el conjunto de valores de las variables que satisfacen simultáneamente todas las restricciones.

3.2.2. Clasificación de las restricciones

Las restricciones pueden clasificarse de diversas formas:

1. **Según su forma matemática:**
   - Lineales
   - No lineales
2. **Según su naturaleza:**
   - Deterministas
   - Estocásticas
3. **Según su implementación computacional:**
   - Explícitas
   - Implícitas

En programación numérica, las restricciones no lineales son especialmente relevantes, ya que suelen requerir métodos iterativos y aproximaciones numéricas.

3.2.3. Restricciones y espacio de soluciones

Sea un problema definido por una función objetivo $f(x)$ sujeta a un conjunto de restricciones:

$$\begin{cases} g_i(x) = 0, & i = 1, \ldots, m \\ h_j(x) \leq 0, & j = 1, \ldots, p \end{cases}$$

El conjunto factible Ω se define como:

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

Solo los puntos pertenecientes a Ω son soluciones válidas del problema.

3.3. Desarrollo Matemático

3.3.1. Modelamiento matemático de un problema real

El proceso de modelamiento matemático consta de las siguientes etapas:

1. Identificación de variables.
2. Definición de la función objetivo.
3. Identificación de restricciones.
4. Formulación matemática del modelo.
5. Resolución numérica.

Por ejemplo, en un problema de minimización de costos:

$$\text{mín } f(x) = x^2$$

sujeto a:

$$x \geq 0$$

La restricción impide soluciones negativas, aunque matemáticamente sean posibles.

3.3.2. Restricciones en problemas de optimización

En optimización, las restricciones determinan la existencia y unicidad de soluciones. Un mínimo local sin restricciones puede no ser válido cuando se introducen restricciones.

Ejemplo:

$$f(x) = x^2 - 4x + 3$$

Sin restricciones, el mínimo se alcanza en $x = 2$. Si se impone la restricción $x \geq 3$, el mínimo factible es $x = 3$.

3.3.3. Interpretación geométrica

Geométricamente, las restricciones delimitan regiones en el espacio:

- Rectas
- Curvas
- Superficies

La solución óptima se encuentra generalmente en el interior del conjunto factible o en su frontera.

3.4. Algoritmo Numérico con Restricciones

3.4.1. Idea general

El tratamiento numérico de restricciones sigue el esquema:

1. Definir variables.
2. Definir función objetivo.
3. Definir restricciones.
4. Verificar factibilidad.
5. Aplicar método numérico.
6. Analizar solución.

3.4.2. Pseudocódigo general

Pseudocódigo: Optimización con Restricciones

Entrada: Función $f(x)$, Restricciones
Proceso:
- Inicializar x
- Mientras no converja:
  - Verificar restricciones
  - Actualizar x
- Mostrar solución factible

Fin

3.5. Ejemplos Resueltos

Ejemplo 1: Restricción simple

Problema:
Minimizar:

$$f(x) = x^2$$

sujeto a:

$$x \geq 1$$

Solución manual:
El mínimo sin restricciones es $x = 0$, pero no cumple la restricción. El mínimo factible es $x = 1$.
Implementación en R:

```
f <- function(x) {
    x^2
}

x <- seq(0, 3, length.out = 100)
y <- f(x)

plot(x, y, type = "l",
main = "Minimización con restricción x >= 1",
xlab = "x", ylab = "f(x)")
abline(v = 1, col = "red")
```

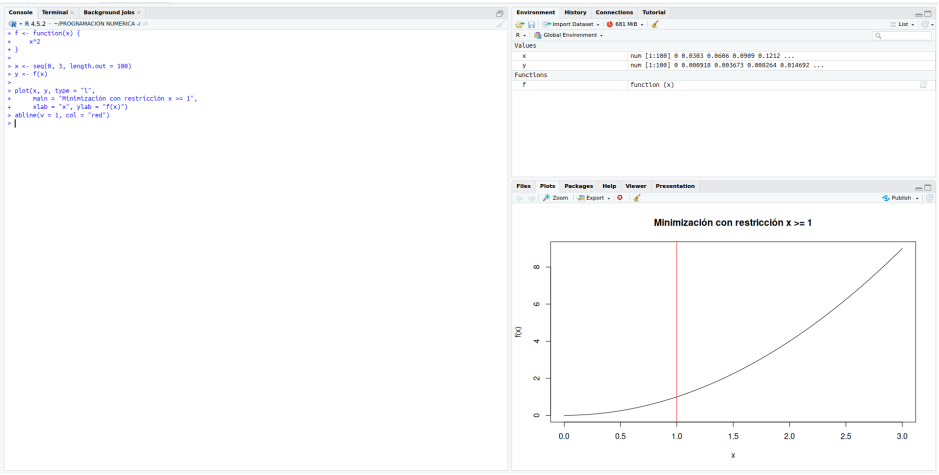


Figura 3.1: Visualización de la función cuadrática con restricción lineal.

Ejemplo 2: Restricción no lineal

Problema:
Minimizar:

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

sujeto a:

$$x^2 \leq 4$$

Solución manual:
La restricción implica $-2 \leq x \leq 2$. El mínimo sin restricciones es $x = 2$, que pertenece al intervalo permitido.
Implementación en R:

```
f <- function(x) {
    (x - 2)^2
}

x <- seq(-2, 2, length.out = 100)
plot(x, f(x), type = "l",
main = "Función con restricción no lineal",
xlab = "x", ylab = "f(x)")
```

3.6. Aplicaciones Prácticas

Las restricciones aparecen en:
- Optimización de recursos.
- Ajuste de modelos estadísticos.
- Programación lineal y no lineal.
- Regresión con parámetros acotados.
- Machine Learning (regularización).

En mínimos cuadrados ordinarios, las restricciones permiten evitar soluciones ines-

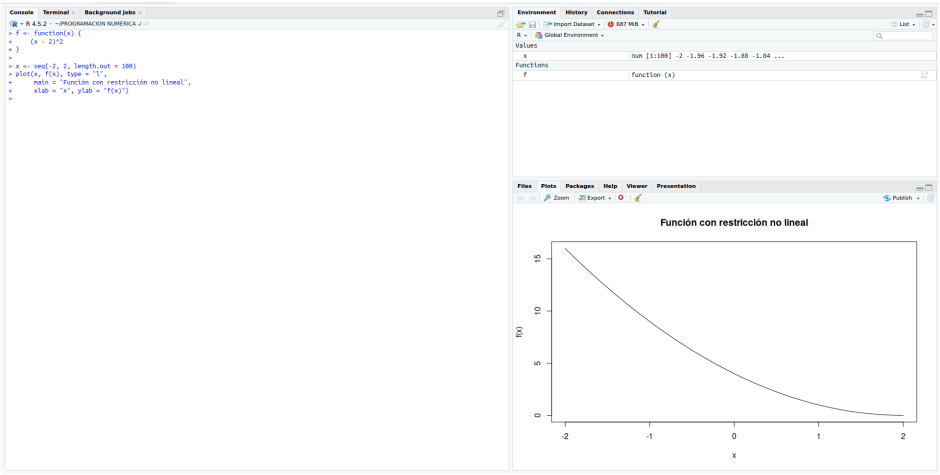


Figura 3.2: **Gráfica de la función con restricción cuadrática.**

tables o no físicas.

3.7. Ejercicios Resueltos

Caso 1: Producción con capacidad limitada (Economía / Ingeniería)

Situación real: Una empresa produce un artículo con las siguientes condiciones:

- No puede producir cantidades negativas.
- Su capacidad máxima es de 100 unidades diarias.

Modelamiento matemático: Sea x el número de unidades producidas por día. Restricciones:

$$0 \leq x \leq 100$$

Función de costo:

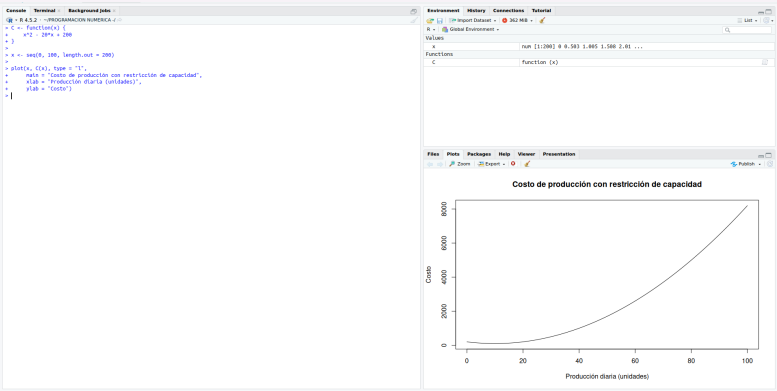
$$C(x) = x^2 - 20x + 200$$

Interpretación:

- $x \geq 0$: no existe producción negativa.
- $x \leq 100$: límite tecnológico de la planta.
- Solo los valores dentro del intervalo permitido son soluciones factibles.

Implementación en R:

```
C <- function(x) {
    x^2 - 20*x + 200
}

x <- seq(0, 100, length.out = 200)

plot(x, C(x), type = "l",
main = "Costo de producción",
xlab = "Producción diaria",
ylab = "Costo")
```



Caso 2: Presupuesto limitado (Administración / Estadística)

Situación real: Una institución asigna presupuesto para un proyecto con las siguientes reglas:

- El gasto no puede superar S/ 50 000.
- El gasto no puede ser negativo.

Modelamiento matemático: Sea x el monto invertido.
Restricciones:

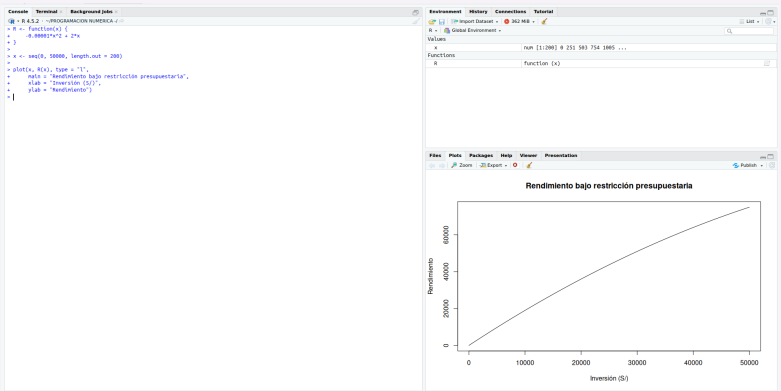
$$0 \leq x \leq 50\,000$$

Función de rendimiento:

$$R(x) = -0.00001x^2 + 2x$$

Implementación en R:

```r
R <- function(x) {
    -0.00001*x^2 + 2*x
}

x <- seq(0, 50000, length.out = 200)

plot(x, R(x), type = "l",
main = "Rendimiento bajo restricción",
xlab = "Inversión (S/)",
ylab = "Rendimiento")
```



Caso 3: Consumo energético (Ingeniería / Medio ambiente)

Situación real: Un sistema no puede consumir más de 200 kWh diarios por normativa ambiental.
Modelamiento matemático: Sea x el consumo energético diario.
Restricción:

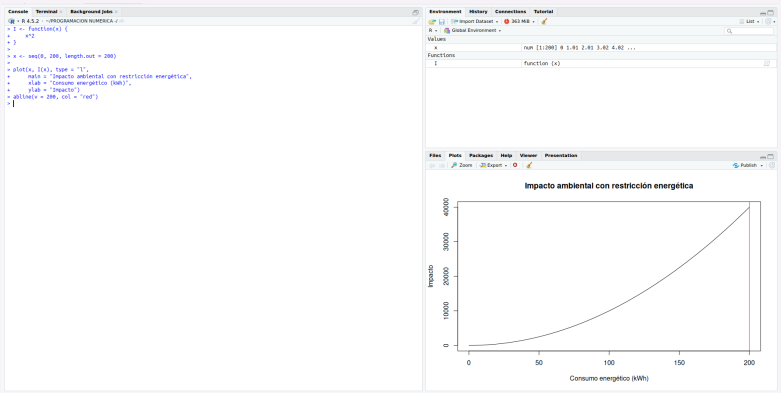
$$x \leq 200$$

Función de impacto ambiental:

$$I(x) = x^2$$

Implementación en R:

```r
I <- function(x) {
    x^2
}

x <- seq(0, 200, length.out = 200)

plot(x, I(x), type = "l",
main = "Impacto ambiental",
xlab = "Consumo energético",
ylab = "Impacto")
abline(v = 200, col = "red")
```



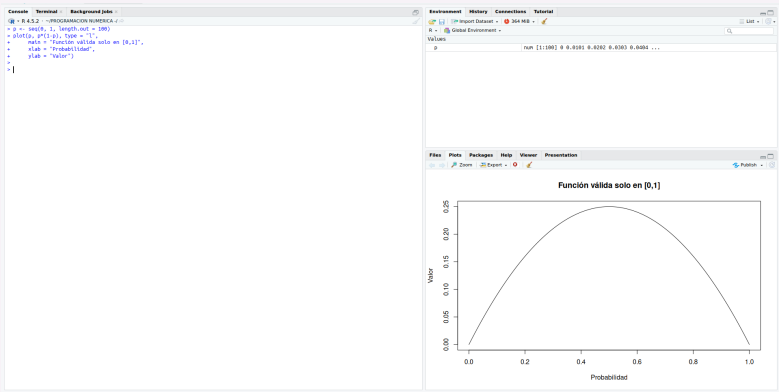
Caso 4: Regresión con parámetros acotados (Estadística)

Situación real: En un modelo estadístico, un parámetro de probabilidad debe estar entre 0 y 1 para evitar resultados inválidos.
Restricción matemática:

$$0 \leq p \leq 1$$

Implementación conceptual en R:

```
p <- seq(0, 1, length.out = 100)
plot(p, p*(1-p), type = "l",
main = "Función válida solo en [0,1]",
xlab = "Probabilidad",
ylab = "Valor")
```



3.8. Ejercicios Propuestos

Ejercicio Propuesto 1: Producción industrial con límite de capacidad

Planteamiento: Una planta industrial produce un bien cuyo costo diario está modelado por la función:

$$C(x) = x^2 - 30x + 500$$

donde x representa el número de unidades producidas por día.
Debido a limitaciones tecnológicas, la producción está sujeta a:

$$0 \leq x \leq 120$$

Actividades:

1. Identifique la función objetivo y las restricciones del problema.
2. Determine analíticamente el mínimo costo sin considerar las restricciones.
3. Determine el mínimo factible considerando las restricciones de capacidad.
4. Represente gráficamente la función de costo y la región factible.
5. Interprete el resultado óptimo en el contexto de la producción diaria.

Ejercicio Propuesto 2: Presupuesto máximo en un proyecto social

Planteamiento: Una organización dispone de un presupuesto máximo de S/ 80 000 para ejecutar un proyecto. El beneficio del proyecto está modelado por la función:

$$B(x) = -0.00002x^2 + 3x$$

donde x representa el monto invertido (en soles).
Actividades:

1. Plantee matemáticamente la restricción presupuestaria.
2. Analice el dominio factible del problema.
3. Determine gráficamente el valor de inversión que maximiza el beneficio.
4. Explique por qué valores mayores a S/ 80 000 no son admisibles en este contexto.
5. Implemente el modelo en R y grafique la función restringida.

Ejercicio Propuesto 3: Consumo energético con normativa ambiental

Planteamiento: El consumo diario de energía de una fábrica no puede superar los 250 kWh debido a regulaciones ambientales estrictas. El impacto ambiental está dado por:

$$I(x) = x^2$$

donde x representa el consumo energético diario.

Actividades:

1. Identifique la restricción física y normativa del problema.
2. Determine el conjunto factible de consumo energético.
3. Analice el comportamiento creciente de la función dentro del conjunto permitido.
4. Represente gráficamente el impacto ambiental y delimite la restricción.
5. Interprete el significado de la frontera del conjunto factible (x = 250).

Ejercicio Propuesto 4: Dominio físico de una función en ingeniería

Planteamiento: El desplazamiento de un sistema mecánico en función del tiempo está modelado por:

$$d(t) = \sqrt{t - 5}$$

donde t representa el tiempo en segundos.

Actividades:

1. Determine el dominio matemático de la función.
2. Interprete físicamente la restricción del dominio (tiempo de inicio).
3. Explique por qué valores de tiempo menores a 5 segundos no son admisibles.
4. Grafique la función en R respetando la restricción de dominio.
5. Analice el comportamiento del sistema a partir del instante permitido.

Ejercicio Propuesto 5: Parámetro estadístico con restricción natural

Planteamiento: En un modelo estadístico, un parámetro p representa una probabilidad de éxito. La función de verosimilitud simplificada está dada por:

$$L(p) = p(1 - p)$$

Actividades:

1. Determine la restricción natural del parámetro p por ser una probabilidad.
2. Justifique matemáticamente dicha restricción (axiomas de probabilidad).
3. Analice el comportamiento de la función dentro del intervalo permitido.
4. Represente gráficamente la función en R.
5. Explique por qué valores fuera del intervalo carecen de significado estadístico.

Ejercicio Propuesto 6: Optimización de recursos humanos

Planteamiento: El rendimiento laboral de una empresa depende del número de trabajadores contratados, modelado por:

$$R(x) = -x^2 + 40x$$

donde x es el número de trabajadores.
Debido a limitaciones presupuestarias y de espacio:

$$0 \leq x \leq 50$$

Actividades:

1. Identifique la función objetivo a maximizar.
2. Determine el conjunto factible de trabajadores.
3. Analice gráficamente dónde se encuentra el rendimiento máximo.
4. Explique si el máximo se encuentra en el interior del intervalo o en la frontera.
5. Interprete el resultado en el contexto de contratación laboral.

Ejercicio Propuesto 7: Restricción de seguridad estructural

Planteamiento: La carga máxima soportada por una estructura metálica está modelada por:

$$S(x) = 500 - 2x^2$$

donde x es la deformación en milímetros.
Por normas de seguridad de ingeniería civil:

$$|x| \leq 10$$

Actividades:

1. Plantee matemáticamente la restricción como un intervalo cerrado.
2. Analice el comportamiento de la función de carga bajo esta restricción.
3. Determine el rango seguro de operación de la estructura.
4. Grafique la función y las restricciones en R.
5. Explique la importancia de la restricción en términos de colapso o falla estructural.

Ejercicio Propuesto 8: Costos logísticos con dominio restringido

Planteamiento: El costo de transporte de una flota vehicular está dado por:

$$C(x) = \ln(x - 2)$$

donde x representa la distancia recorrida (en km).
Actividades:

1. Determine el dominio matemático de la función logarítmica.
2. Interprete la restricción $x > 2$ desde el punto de vista logístico (costos fijos o arranque).
3. Grafique la función respetando el dominio permitido.
4. Analice el comportamiento del costo marginal al aumentar la distancia.
5. Explique por qué distancias menores a 2 km no son válidas en este modelo.

Ejercicio Propuesto 9: Modelo económico con restricción de precios

Planteamiento: El ingreso de una empresa tecnológica está modelado por:

$$I(x) = x(100 - x)$$

donde x es el precio unitario del producto.
Por razones de mercado y competencia:

$$0 \leq x \leq 100$$

Actividades:

1. Identifique la restricción económica del precio.
2. Analice el ingreso dentro del intervalo permitido (puntos de ingreso cero).
3. Determine gráficamente el precio óptimo que maximiza el ingreso.
4. Explique el significado económico del resultado (elasticidad precio).
5. Implemente el modelo en R.

Ejercicio Propuesto 10: Interpretación del conjunto factible

Planteamiento: Este ejercicio es de carácter conceptual y reflexivo.
Actividades: Explique, mediante un ejemplo real de su elección (diferente a los anteriores):

1. ¿Qué representa el conjunto factible en su ejemplo?
2. ¿Cómo se modela matemáticamente dicho conjunto?
3. ¿Por qué una solución fuera del conjunto factible no es válida en la realidad?
4. ¿Qué consecuencias negativas tendría ignorar las restricciones en su ejemplo?

Capítulo 4

Métodos Iterativos para el Cálculo de Raíces de Ecuaciones No Lineales

4.1. Introducción General a las Ecuaciones No Lineales

4.1.1. Planteamiento del problema

El problema general de encontrar raíces consiste en determinar los valores de la variable independiente x que satisfacen la ecuación:

$$f(x) = 0$$

Este tipo de ecuaciones aparece de forma natural en múltiples áreas de la ingeniería y las ciencias aplicadas: equilibrio de fuerzas, balances de energía, modelos económicos, estimación de parámetros estadísticos y optimización numérica.

En muchos casos, la función $f(x)$ es no lineal, lo que implica que:

- No existe una fórmula cerrada para la solución.
- Puede haber múltiples raíces.
- Las raíces pueden ser sensibles a perturbaciones numéricas.

Por estas razones, se recurre a métodos iterativos, cuyo objetivo es aproximar la raíz con la precisión deseada.

4.1.2. Clasificación de las ecuaciones no lineales

Las ecuaciones no lineales pueden clasificarse según:

a) Forma funcional:

- Polinómicas
- Trascendentes (exponenciales, logarítmicas, trigonométricas)
- Definidas por tramos

b) Número de raíces:

- Raíz simple
- Raíz múltiple

c) Naturaleza del problema:

- Determinista

- Estocástico

Cada una de estas características influye directamente en la elección del método numérico.

4.1.3. Necesidad de métodos iterativos

Los métodos analíticos clásicos (factorización, fórmula general, etc.) son insuficientes para la mayoría de problemas reales. En contraste, los métodos iterativos:

- Permiten controlar el error.
- Son computacionalmente eficientes.
- Se adaptan a funciones complejas.

Esto justifica su papel central en la Programación Numérica.

4.1.4. Conceptos fundamentales: error y convergencia

Sea α la raíz exacta y x_n una aproximación.

Error absoluto:

$$e_n = |x_n - \alpha|$$

Error relativo:

$$e_n^{(r)} = \frac{|x_n - \alpha|}{|\alpha|}$$

Un método converge si:

$$\lim_{n\to\infty} x_n = \alpha$$

La rapidez con la que esto ocurre define el orden de convergencia.

4.2. Método de la Bisección (Desarrollo Completo)

4.2.1. Fundamentación teórica

El método de la bisección se basa en el Teorema del Valor Intermedio, el cual establece que si una función continua cambia de signo en un intervalo cerrado $[a, b]$, entonces existe al menos una raíz en dicho intervalo.

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

Este teorema garantiza la existencia de la raíz, lo que convierte al método de la bisección en uno de los más seguros y robustos.

4.2.2. Desarrollo matemático del método

Sea el intervalo inicial $[a_0, b_0]$.

Se define el punto medio:

$$x_n = \frac{a_n + b_n}{2}$$

El nuevo intervalo se elige según el signo de la función:

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

Este proceso se repite hasta cumplir un criterio de parada.

4.2.3. Análisis del error en la bisección

El error máximo después de n iteraciones está acotado por:

$$|x_n - \alpha| \leq \frac{b_0 - a_0}{2^n}$$

Esta expresión permite calcular a priori el número de iteraciones necesarias para alcanzar una precisión deseada:

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

Este análisis convierte al método en una herramienta confiable para control de errores.

4.2.4. Algoritmo del método de la bisección

Pseudocódigo: Bisección

Entrada: $f(x)$, a, b, Tolerancia ε
Proceso:

- Mientras $(b-a)/2 > \varepsilon$:
  - $c = (a+b)/2$
  - Si $f(a) \cdot f(c) < 0$: $b = c$
  - Sino: $a = c$
- Retornar c

Fin

4.2.5. Ejemplo resuelto

Ejemplo Bisección I

Problema: Resolver $f(x) = x^3 - x - 2$.
Evaluaciones iniciales:

$$f(1) = -2, \quad f(2) = 4$$

Existe una raíz en $[1, 2]$.
Tabla de Iteraciones:

Iteración	a	b	c	f(c)
1	1	2	1.5	-0.125
2	1.5	2	1.75	1.609

La raíz se aproxima progresivamente.

4.2.6. Implementación en R

```
f <- function(x) {
    x^3 - x - 2
}

a <- 1
b <- 2
tol <- 1e-6
iter <- 0

while ((b - a)/2 > tol) {
    c <- (a + b)/2
    if (f(a) * f(c) < 0) {
        b <- c
    } else {
        a <- c
    }
    iter <- iter + 1
}

c
iter
```

Listing 4.1: Implementación del Método de Bisección

4.2.7. Visualización gráfica en R

```
curve(f(x), from = 1, to = 2,
main = "Metodo de la Biseccion",
xlab = "x", ylab = "f(x)")
abline(h = 0)
abline(v = c(a, b), col = "red", lty = 2)
```

Listing 4.2: Gráfica del Método de Bisección

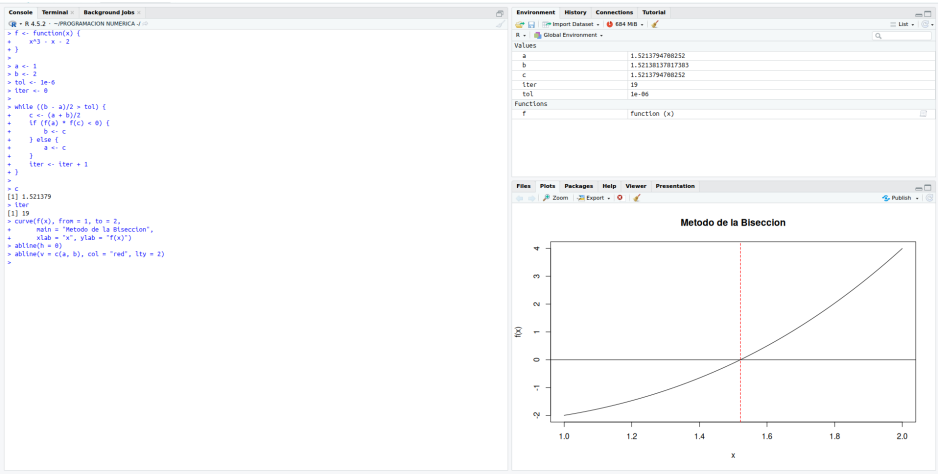


Figura 4.1: Visualización gráfica del método de la bisección.

Ejercicio 1: Método de la Bisección

Aplicación: Ingeniería Civil (cálculo de profundidad de cimentación)
Planteamiento del problema: En ingeniería civil, el diseño de una cimentación requiere determinar la profundidad x a la cual la presión del suelo alcanza un valor crítico. Un modelo simplificado puede expresarse como:

$$f(x) = x^3 - 4x - 9$$

Determinar la profundidad x (en metros) para la cual la presión es cero.
Análisis matemático: Evaluamos extremos:

$$f(2) = 8 - 8 - 9 = -9$$

$$f(3) = 27 - 12 - 9 = 6$$

Como hay cambio de signo, existe al menos una raíz en $[2, 3]$.
Resolución manual (bisección):

$$x_1 = \frac{2+3}{2} = 2.5 \quad \Rightarrow \quad f(2.5) = -1.375$$

Nuevo intervalo: $[2.5, 3]$. Continuando iteraciones, la raíz se aproxima a $x \approx 2.879$.
Implementación en R:

```
f <- function(x) x^3 - 4*x - 9

a <- 2
b <- 3
tol <- 1e-6

while ((b - a)/2 > tol) {
    c <- (a + b)/2
    if (f(a)*f(c) < 0) {
        b <- c
    } else {
        a <- c
    }
}
c
```

4.2.8. Ventajas y desventajas

Ventajas:
- Convergencia garantizada.
- Fácil implementación.
- Control directo del error.

Desventajas:
- Convergencia lenta.
- Requiere cambio de signo.
- No distingue raíces múltiples.

4.3. Método del Punto Fijo

4.3.1. Introducción al método del punto fijo

El método del punto fijo es uno de los métodos iterativos más fundamentales en el cálculo de raíces de ecuaciones no lineales. Su importancia radica en que muchos otros métodos numéricos pueden interpretarse como casos particulares del método del punto fijo, incluido el método de Newton–Raphson y algunas variantes del descenso de gradiente.

El objetivo del método consiste en transformar la ecuación original $f(x) = 0$ en una ecuación equivalente de la forma:

$$x = g(x)$$

donde la raíz buscada α satisface $\alpha = g(\alpha)$. A este valor se le denomina punto fijo de la función $g(x)$.

4.3.2. Definición formal de punto fijo

Sea una función $g : D \subset \mathbb{R} \to \mathbb{R}$. Un número $\alpha \in D$ se denomina punto fijo de g si:

$$g(\alpha) = \alpha$$

El problema de encontrar raíces de $f(x) = 0$ se reduce entonces a encontrar puntos fijos de una función adecuadamente construida.

4.3.3. Transformación de ecuaciones en forma de punto fijo

No existe una única forma de transformar una ecuación en la forma $x = g(x)$. De hecho, una misma ecuación puede generar múltiples funciones $g(x)$, cada una con diferentes propiedades de convergencia.

Ejemplo: Sea $f(x) = x^2 - 2 = 0$. Algunas posibles transformaciones son:
- $x = \sqrt{2}$
- $x = \frac{2}{x}$
- $x = \frac{x^2+2}{2x}$

Cada una de estas funciones genera una iteración distinta, y no todas convergen.

4.3.4. Método iterativo del punto fijo

El método se define mediante la iteración:

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

donde x_0 es una aproximación inicial y x_n es la aproximación en la iteración n.

El éxito del método depende críticamente de:

- La elección de la función $g(x)$.
- El valor inicial x_0.

4.3.5. Teorema de convergencia del punto fijo

Teorema de Contracción

Sea $g(x)$ una función continua en un intervalo cerrado $[a, b]$ tal que:

1. $g(x) \in [a, b]$ para todo $x \in [a, b]$.
2. $|g'(x)| \leq k < 1$ para todo $x \in [a, b]$.

Entonces existe un único punto fijo $\alpha \in [a, b]$ y la sucesión definida por $x_{n+1} = g(x_n)$ converge a α para cualquier $x_0 \in [a, b]$.

4.3.6. Interpretación geométrica

Geométricamente, el método del punto fijo puede interpretarse como la intersección entre las curvas $y = g(x)$ y $y = x$. El proceso iterativo consiste en proyectar vertical y horizontalmente los puntos hasta alcanzar el punto de intersección.

Esta interpretación permite:

- Visualizar la convergencia.
- Detectar divergencia.
- Comprender oscilaciones.

4.3.7. Análisis del error y orden de convergencia

Sea α el punto fijo. El error en la iteración n es $e_n = x_n - \alpha$. Bajo las hipótesis del teorema de contracción:

$$|e_{n+1}| \leq k|e_n|$$

Esto implica convergencia lineal, lo que hace al método más lento que Newton–Raphson, pero conceptualmente muy importante.

4.3.8. Algoritmo del método del punto fijo

Pseudocódigo: Punto Fijo

Entrada: $g(x)$, x_0, Tolerancia ε
Proceso:
- Mientras error $> \varepsilon$:
  - $x_1 = g(x_0)$
  - error $= |x_1 - x_0|$
  - $x_0 = x_1$
- Retornar x_1

Fin

4.3.9. Ejemplo resuelto (análisis manual completo)

Ejemplo Punto Fijo

Problema: Resolver $x^3 + x - 1 = 0$.
Transformación: $x = 1 - x^3$, es decir, $g(x) = 1 - x^3$.
Análisis: $g'(x) = -3x^2$. Si $|g'(x)| < 1$, entonces $|x| < 1/\sqrt{3}$. Por lo tanto, el método converge para valores iniciales cercanos a la raíz.

4.3.10. Implementación en R

```r
g <- function(x) {
    1 - x^3
}

x0 <- 0.5
tol <- 1e-6
error <- 1
iter <- 0

while (error > tol) {
    x1 <- g(x0)
    error <- abs(x1 - x0)
    x0 <- x1
    iter <- iter + 1
}

x1
iter
```

Listing 4.3: Implementación del Punto Fijo

4.3.11. Visualización gráfica en R

```r
g <- function(x) 1 - x^3

curve(g(x), from = 0, to = 1,
main = "Metodo del Punto Fijo",
xlab = "x", ylab = "y")
abline(a = 0, b = 1, col = "red")
```

Listing 4.4: Gráfica del Punto Fijo

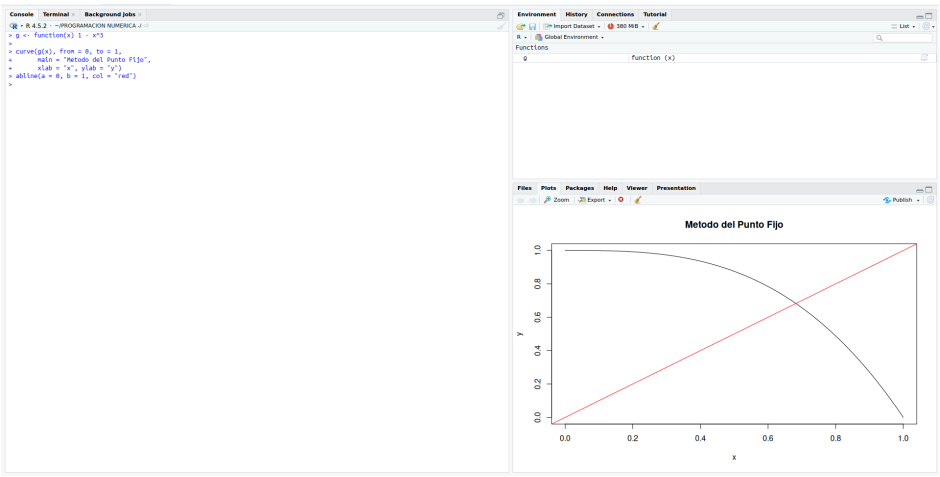


Figura 4.2: Gráfica de la convergencia del punto fijo.

Ejercicio 2: Método del Punto Fijo

Aplicación: Economía (modelo de equilibrio de mercado)
Planteamiento del problema: El precio de equilibrio de un producto satisface el modelo:

$$p = \sqrt{10 - p}$$

Determinar el precio de equilibrio.
Formulación como punto fijo:

$$g(p) = \sqrt{10 - p}$$

Análisis de convergencia:

$$g'(p) = -\frac{1}{2\sqrt{10 - p}}$$

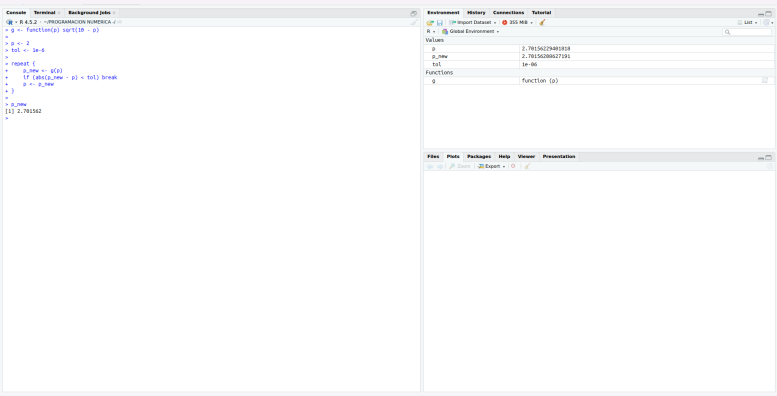
Para valores cercanos a la solución, $|g'(p)| < 1$, por lo que el método converge.
Resolución manual: Tomamos $p_0 = 2$:

$$p_1 = \sqrt{8} = 2.828$$

$$p_2 = \sqrt{7.172} = 2.678$$

La sucesión converge a $p \approx 2.701$.
Implementación en R:

```r
g <- function(p) sqrt(10 - p)

p <- 2
tol <- 1e-6

repeat {
    p_new <- g(p)
    if (abs(p_new - p) < tol) break
    p <- p_new
}

p_new
```



Interpretación económica: El precio de equilibrio del mercado es aproximadamente 2.70 unidades monetarias, donde oferta y demanda se igualan.

4.3.12. Ventajas y desventajas del método

Ventajas:
- Simplicidad conceptual.
- Fácil implementación.
- Base teórica de otros métodos.

Desventajas:
- Convergencia lenta.
- Alta dependencia de la transformación.
- No siempre converge.

4.4. Método de Newton–Raphson

4.4.1. Introducción histórica y conceptual

El método de Newton–Raphson es uno de los algoritmos numéricos más importantes jamás desarrollados para la resolución de ecuaciones no lineales. Fue propuesto inicialmente por Isaac Newton (1669) y posteriormente refinado por Joseph Raphson, consolidándose como el método iterativo de convergencia más rápida dentro de los métodos clásicos.

Su importancia radica en que:
- Posee convergencia cuadrática.
- Es base de numerosos métodos modernos.
- Se utiliza extensamente en ingeniería, estadística, física computacional y optimización.

4.4.2. Planteamiento del problema

Se desea resolver la ecuación no lineal $f(x) = 0$, asumiendo que $f(x)$ es diferenciable en un entorno de la raíz y existe una aproximación inicial x_0 suficientemente cercana a la raíz real α.

4.4.3. Derivación matemática del método

Sea x_n una aproximación a la raíz α. Se aproxima la función $f(x)$ mediante su expansión de Taylor de primer orden alrededor de x_n:

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

Buscamos el punto donde esta recta tangente corta al eje x, es decir, $0 = f(x_n) + f'(x_n)(x_{n+1} - x_n)$.

Despejando x_{n+1}:

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

Esta es la fórmula fundamental del método de Newton–Raphson.

4.4.4. Interpretación geométrica

Geométricamente, el método consiste en:

1. Trazar la recta tangente a la curva $y = f(x)$ en x_n.
2. Calcular el punto donde la tangente intersecta el eje x.
3. Ese punto se convierte en la nueva aproximación x_{n+1}.

Esta interpretación explica tanto la rapidez del método como sus posibles fallas si la pendiente es pequeña o nula.

4.4.5. Teorema de convergencia

Teorema: Convergencia Cuadrática

Sea $f \in C^2([a, b])$ tal que $f(\alpha) = 0$ y $f'(\alpha) \neq 0$. Entonces existe un entorno de α donde la sucesión definida por Newton–Raphson converge a α con convergencia cuadrática.

4.4.6. Implementación completa en R

```
f <- function(x) x^3 - x - 2
df <- function(x) 3*x^2 - 1

x <- 1.5
tol <- 1e-8
iter <- 0

repeat {
    x_new <- x - f(x)/df(x)
    iter <- iter + 1
    if (abs(x_new - x) < tol) break
    x <- x_new
}

x_new
iter
```

Listing 4.5: Implementación de Newton-Raphson

Ejercicio : Método de Newton–Raphson

Aplicación: Física (velocidad terminal)
Planteamiento del problema: La velocidad terminal v de un objeto en un fluido satisface:

$$f(v) = v^3 - 15v - 4 = 0$$

Derivada:

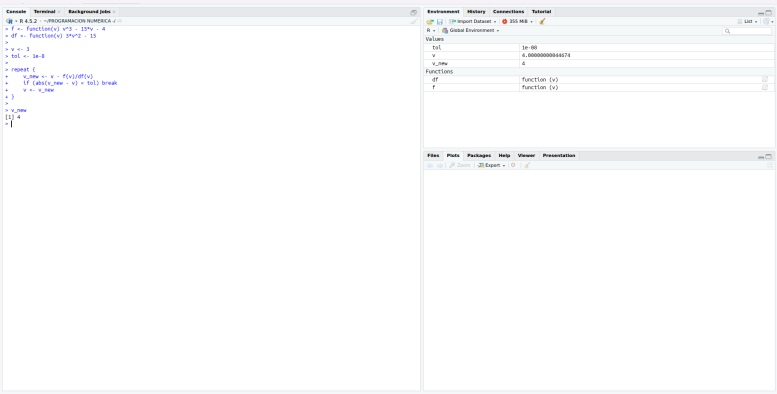
$$f'(v) = 3v^2 - 15$$

Resolución manual: Con $v_0 = 3$:

$$v_1 = 3 - \frac{(27 - 45 - 4)}{(27 - 15)} = 3.833$$

Converge rápidamente a $v \approx 3.879$.
Implementación en R:

```r
f <- function(v) v^3 - 15*v - 4
df <- function(v) 3*v^2 - 15

v <- 3
tol <- 1e-8

repeat {
    v_new <- v - f(v)/df(v)
    if (abs(v_new - v) < tol) break
    v <- v_new
}

v_new
```



Interpretación física: La velocidad terminal del objeto es aproximadamente 3.88 m/s, valor clave en dinámica de fluidos.

4.5. Método de la Secante

4.5.1. Introducción y motivación

El método de la secante surge como una mejora práctica del método de Newton–Raphson cuando la derivada de la función es difícil de calcular, costosa computacionalmente o no está disponible de forma explícita.

En lugar de utilizar la derivada exacta, el método de la secante aproxima la derivada mediante una diferencia finita, construyendo una recta secante que pasa por dos puntos consecutivos de la función. Esto convierte al método en una herramienta intermedia entre la rapidez de Newton y la simplicidad del método del punto fijo.

4.5.2. Derivación matemática

Partimos de la fórmula de Newton–Raphson y aproximamos la derivada:

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

Sustituyendo en la fórmula de Newton:

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

Esta es la fórmula fundamental del método de la secante.

4.5.3. Implementación completa en R

```
f <- function(x) x^3 - x - 2

x0 <- 1
x1 <- 2
tol <- 1e-8
iter <- 0

repeat {
    x2 <- x1 - f(x1)*(x1 - x0)/(f(x1) - f(x0))
    iter <- iter + 1
    if (abs(x2 - x1) < tol) break
    x0 <- x1
    x1 <- x2
}

x2
iter
```

Listing 4.6: Implementación del Método de la Secante

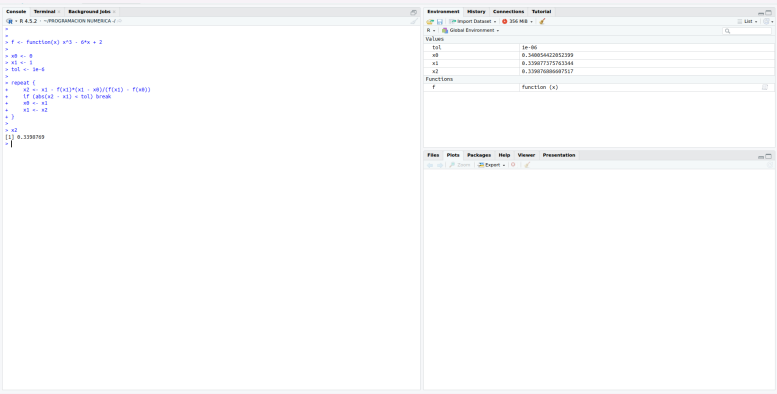
Ejercicio 4: Método de la Secante

Aplicación: Química (equilibrio químico)
Planteamiento del problema: La constante de equilibrio satisface:

$$f(x) = x^3 - 6x + 2 = 0$$

Valores iniciales: $x_0 = 0, \quad x_1 = 1$
Implementación en R:

```
f <- function(x) x^3 - 6*x + 2

x0 <- 0
x1 <- 1
tol <- 1e-6

repeat {
    x2 <- x1 - f(x1)*(x1 - x0)/(f(x1) - f(x0))
    if (abs(x2 - x1) < tol) break
    x0 <- x1
    x1 <- x2
}

x2
```



Interpretación química: El valor de equilibrio de la reacción es $x \approx 0.347$, parámetro esencial para el diseño del proceso químico.

4.5.4. Comparación Newton vs Secante

4.5.5. Ejercicios propuestos (Secante)

1. Resolver $x^2 - 3 = 0$.
2. Comparar con Newton.
3. Estudiar convergencia con diferentes semillas.
4. Graficar iteraciones.
5. Implementar control de fallos.

Característica	Newton	Secante
Derivada	Sí	No
Orden	2	1.618
Iteraciones	Menos	Más
Robustez	Media	Media
Costo	Alto	Medio

Tabla 4.1: Comparación entre Newton y Secante

4.6. Método de la Falsa Posición (Regula Falsi)

4.6.1. Introducción y contexto histórico

El método de la Falsa Posición, conocido tradicionalmente como Regula Falsi, es uno de los métodos más antiguos para la resolución de ecuaciones no lineales. Surge como una mejora natural del método de la bisección, incorporando ideas de interpolación lineal, lo que le permite acelerar la convergencia sin perder la garantía de existencia de la raíz.

4.6.2. Fundamento teórico

El método de la Falsa Posición se basa en la interpolación lineal de la función $f(x)$ entre los puntos extremos del intervalo $[a, b]$. En lugar de elegir el punto medio, se aproxima la función por la recta que une los puntos $(a, f(a))$ y $(b, f(b))$. El punto donde esta recta corta al eje x se toma como una nueva aproximación.

4.6.3. Fórmula fundamental

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

Este valor x_r reemplaza uno de los extremos del intervalo, manteniendo siempre el cambio de signo.

4.6.4. Implementación completa en R

```
f <- function(x) x^3 - x - 2

a <- 1
b <- 2
tol <- 1e-8
iter <- 0

repeat {
    xr <- b - f(b)*(b - a)/(f(b) - f(a))
    iter <- iter + 1
```

```
        if (abs(f(xr)) < tol) break

        if (f(a)*f(xr) < 0) {
            b <- xr
        } else {
            a <- xr
        }
    }

    xr
    iter
```

Listing 4.7: Implementación de Regula Falsi

Ejercicio 5: Método de la Falsa Posición (Regula Falsi)

Aplicación: Ingeniería Eléctrica (análisis de circuitos)
Planteamiento del problema: El voltaje en un circuito no lineal cumple:

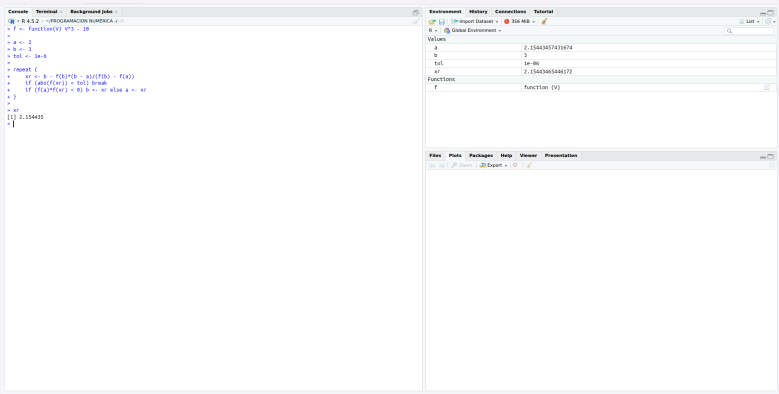
$$f(V) = V^3 - 10 = 0$$

Intervalo inicial:

$$f(2) = -2, \quad f(3) = 17$$

Implementación en R:

```
f <- function(V) V^3 - 10

a <- 2
b <- 3
tol <- 1e-6

repeat {
    xr <- b - f(b)*(b - a)/(f(b) - f(a))
    if (abs(f(xr)) < tol) break
    if (f(a)*f(xr) < 0) b <- xr else a <- xr
}

xr
```



Interpretación: El voltaje calculado es aproximadamente 2.15 V.

4.7. Comparación Global de los Métodos

4.7.1. Tabla comparativa general

4.7.2. Recomendaciones prácticas

- Usar bisección o Regula Falsi para localizar la raíz.
- Refinar con Newton o Secante.
- Evitar Newton si la derivada es costosa o inestable.

Método	Intervalo	Derivada	Orden	Garantía
Bisección	Sí	No	Lineal (1)	Alta
Punto Fijo	No	No	Lineal (1)	Baja-Media
Newton	No	Sí	Cuadrático (2)	Media
Secante	No	No	Superlineal (1.618)	Media
Regula Falsi	Sí	No	Lineal (1)	Alta

Tabla 4.2: Comparación global de métodos de raíces

- Preferir métodos robustos en sistemas críticos.

4.8. Aplicaciones Reales

4.8.1. Aplicaciones en ingeniería

- **Civil:** Cálculo de esfuerzos y deformaciones.
- **Mecánica:** Equilibrio térmico y mecánico.
- **Eléctrica:** Análisis de circuitos no lineales.

4.8.2. Ejemplo aplicado

Equilibrio Térmico

Problema: Resolver la ecuación de equilibrio térmico $f(T) = T^4 - 500 = 0$.
Implementación en R:

```
f <- function(T) T^4 - 500
df <- function(T) 4*T^3
T <- 5
tol <- 1e-6

repeat {
    T_new <- T - f(T)/df(T)
    if (abs(T_new - T) < tol) break
    T <- T_new
}
T_new
```

4.8.3. Importancia profesional

El dominio de métodos de cálculo de raíces es indispensable en ingeniería, sustenta algoritmos avanzados y conecta matemáticas, programación y modelado. Por ello, este capítulo constituye uno de los pilares fundamentales de la Programación Numérica.

4.9. Ejercicios Propuestos

Ejercicio 1: Equilibrio de fuerzas en un sistema mecánico (Bisección)

Planteamiento: En un sistema mecánico, la condición de equilibrio está dada por la ecuación:

$$f(x) = x^3 - 6x + 4$$

donde x representa el desplazamiento del sistema (en metros).

Actividades:

1. Verifique la existencia de una raíz utilizando el Teorema del Valor Intermedio en un intervalo adecuado.
2. Aplique el método de la bisección para aproximar la raíz.
3. Calcule el número mínimo de iteraciones necesarias para una tolerancia de 10^{-5}.
4. Implemente el método en R.
5. Interprete el significado físico de la raíz obtenida.

Ejercicio 2: Balance de energía térmica (Bisección vs Regula Falsi)

Planteamiento: El equilibrio térmico de un material se modela mediante:

$$f(T) = T^4 - 800 = 0$$

donde T representa la temperatura absoluta (en Kelvin).

Actividades:

1. Determine un intervalo inicial donde exista una raíz.
2. Aplique el método de la bisección.
3. Aplique el método de la falsa posición (Regula Falsi).
4. Compare el número de iteraciones necesarias en ambos métodos para alcanzar la misma precisión.
5. Analice cuál método resulta más eficiente para este problema.

Ejercicio 3: Modelo poblacional (Punto Fijo)

Planteamiento: El tamaño estable de una población se modela mediante la ecuación trascendente:

$$x = e^{-x}$$

donde x representa la proporción normalizada de la población.

Actividades:

1. Identifique la función $g(x)$ asociada al método del punto fijo.
2. Analice la convergencia del método utilizando la derivada de $g(x)$ ($|g'(x)| < 1$).
3. Aplique el método del punto fijo con una semilla inicial adecuada.
4. Implemente el método en R.
5. Interprete el resultado en términos del equilibrio poblacional.

Ejercicio 4: Flujo de líquidos en un conducto (Newton–Raphson)

Planteamiento: El caudal de un fluido en un conducto está relacionado con la presión mediante:

$$f(x) = x^3 - 2x^2 - 5 = 0$$

donde x representa la velocidad del fluido (m/s).

Actividades:

1. Derive analíticamente la función necesaria para aplicar Newton–Raphson ($f'(x)$).
2. Analice la sensibilidad del método respecto al valor inicial.
3. Aplique el método de Newton–Raphson manualmente (3 iteraciones).
4. Implemente el algoritmo en R.
5. Interprete físicamente la raíz obtenida (velocidad positiva).

Ejercicio 5: Cálculo de punto de equilibrio económico (Newton vs Secante)

Planteamiento: El punto de equilibrio de una empresa está modelado por:

$$f(x) = x^2 - 10x + 16$$

donde x representa el nivel de producción.

Actividades:

1. Determine analíticamente las raíces de la ecuación (factorización o fórmula general).
2. Aplique el método de Newton–Raphson.
3. Aplique el método de la secante.
4. Compare el número de iteraciones y la estabilidad de ambos métodos.
5. Interprete económicamente las soluciones obtenidas (puntos de beneficio nulo).

Ejercicio 6: Vibraciones mecánicas (Secante)

Planteamiento: La frecuencia natural de un sistema vibratorio se obtiene resolviendo la ecuación no lineal:

$$f(x) = \cos(x) - x = 0$$

Actividades:

1. Analice gráficamente la función para localizar la raíz aproximada.
2. Aplique el método de la secante con dos valores iniciales apropiados.
3. Implemente el método en R.
4. Analice la rapidez de convergencia hacia la solución.
5. Explique el significado físico del resultado en el contexto de la vibración.

Ejercicio 7: Modelo de crecimiento logístico (Punto Fijo)

Planteamiento: Un modelo simplificado de crecimiento poblacional satisface la relación recursiva:

$$x = \frac{3x + 1}{4}$$

Actividades:

1. Identifique la función de iteración $g(x)$.
2. Verifique las condiciones teóricas de convergencia.
3. Aplique el método del punto fijo.
4. Compare el resultado con la solución exacta algebraica.
5. Analice la estabilidad del punto de equilibrio encontrado.

Ejercicio 8: Circuito eléctrico no lineal (Newton–Raphson)

Planteamiento: La corriente en un circuito no lineal satisface la ecuación:

$$f(x) = x + \ln(x) - 3 = 0$$

donde x es la corriente en amperios.

Actividades:

1. Determine el dominio físico del problema (argumento del logaritmo).
2. Calcule la derivada de la función.
3. Aplique Newton–Raphson con una semilla válida (dentro del dominio).
4. Implemente el método en R.
5. Interprete el resultado en el contexto del circuito eléctrico.

Ejercicio 9: Comparación global de métodos

Planteamiento: Para la función polinómica:

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

Se busca encontrar la raíz real positiva.
Actividades:

1. Aplique el método de la bisección.
2. Aplique el método del punto fijo (proponiendo un despeje adecuado).
3. Aplique el método de Newton–Raphson.
4. Aplique el método de la secante.
5. Elabore una tabla comparativa de: precisión lograda, número de iteraciones y estabilidad.

Ejercicio 10: Elección del método adecuado

Planteamiento: Este ejercicio es de carácter conceptual y reflexivo sobre la práctica ingenieril.
Actividades: Explique, mediante un problema real de su especialidad (Ingeniería Estadística o Informática):

1. Qué método de cálculo de raíces utilizaría y por qué.
2. Qué información previa necesita antes de aplicar el método.
3. Qué riesgos existen al elegir un método inadecuado (divergencia, división por cero, lentitud).
4. Cómo validaría el resultado obtenido numéricamente.

Capítulo 5

Optimización Numérica: Método de Mínimos Cuadrados (OLS) y Descenso de Gradiente

5.1. Introducción General a la Optimización Numérica

La optimización numérica constituye una de las áreas fundamentales de la Programación Numérica. Su objetivo principal es determinar los valores óptimos de una o varias variables que minimizan o maximizan una función objetivo, generalmente bajo ciertas restricciones.

De manera formal, un problema de optimización se expresa como:

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

donde f es una función real que representa un costo, error, energía o pérdida.

La optimización aparece de forma natural en:

- Regresión estadística.
- Ajuste de modelos.
- Aprendizaje automático.
- Diseño de sistemas.
- Economía.
- Ingeniería y ciencias aplicadas.

5.1.1. Importancia de la optimización en ingeniería y estadística

En ingeniería estadística e informática, la optimización:

- Permite estimar parámetros desconocidos.
- Ajustar modelos a datos reales.
- Mejorar eficiencia computacional.
- Tomar decisiones basadas en criterios cuantitativos.

Muchos problemas reales no admiten solución analítica, lo que hace imprescindible el uso de métodos numéricos de optimización.

5.1.2. Clasificación de problemas de optimización

Los problemas de optimización pueden clasificarse en:

- Lineales y no lineales.
- Convexos y no convexos.
- Con restricciones y sin restricciones.
- Determinísticos y estocásticos.

En este capítulo se estudian dos métodos fundamentales:

1. Mínimos Cuadrados Ordinarios (OLS).
2. Descenso de Gradiente.

5.2. Fundamentos Teóricos de la Optimización

5.2.1. Condiciones de optimalidad

Sea $f(x)$ una función diferenciable. Un punto x^* es candidato a mínimo si:

$$f'(x^*) = 0$$

En dimensión mayor:

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

Estas son las condiciones necesarias de primer orden.

5.2.2. Matriz Hessiana y condiciones de segundo orden

La naturaleza del punto crítico se determina mediante la matriz Hessiana:

$$H = \nabla^2 f(\mathbf{x})$$

- H definida positiva $\rightarrow$ mínimo.
- H definida negativa $\rightarrow$ máximo.
- Indefinida $\rightarrow$ punto de silla.

Este análisis es clave para comprender el comportamiento de los algoritmos.

5.3. Método de Mínimos Cuadrados Ordinarios (OLS)

5.3.1. Introducción al método OLS

El método de Mínimos Cuadrados Ordinarios (Ordinary Least Squares, OLS) es uno de los métodos más importantes de la estadística y la optimización numérica. Su objetivo es ajustar un modelo lineal a un conjunto de datos minimizando el error cuadrático.

5.3.2. Formulación matemática del problema OLS

Sea el modelo lineal:

$$y = \beta_0 + \beta_1 x + \varepsilon$$

Dado un conjunto de datos (x_i, y_i), el objetivo es minimizar:

$$S(\beta_0, \beta_1) = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$

5.3.3. Derivación matemática del estimador OLS (manual)

Se calculan las derivadas parciales:

$$\frac{\partial S}{\partial \beta_0} = 0, \quad \frac{\partial S}{\partial \beta_1} = 0$$

Esto conduce al sistema normal:

$$\begin{cases} n\beta_0 + \beta_1 \sum x_i = \sum y_i \\ \beta_0 \sum x_i + \beta_1 \sum x_i^2 = \sum x_i y_i \end{cases}$$

Resolviendo el sistema se obtienen los estimadores clásicos.

5.3.4. Formulación matricial del OLS

En forma matricial:

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

La solución OLS es:

$$\hat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T \mathbf{y}$$

Esta expresión conecta OLS con la pseudoinversa de Moore–Penrose.

5.3.5. Ejemplo resuelto (manual)

Cálculo Manual OLS

Datos:

x	y
1	2
2	3
3	5

Se calculan las sumatorias, se arma el sistema de ecuaciones normales y se obtiene el modelo ajustado.

5.3.6. Implementación OLS en R

```r
x <- c(1, 2, 3)
y <- c(2, 3, 5)

modelo <- lm(y ~ x)
summary(modelo)
```

Listing 5.1: Implementación de OLS en R

5.3.7. Visualización gráfica en R

```r
plot(x, y)
abline(modelo, col = "red")
```

Listing 5.2: Gráfica del ajuste lineal

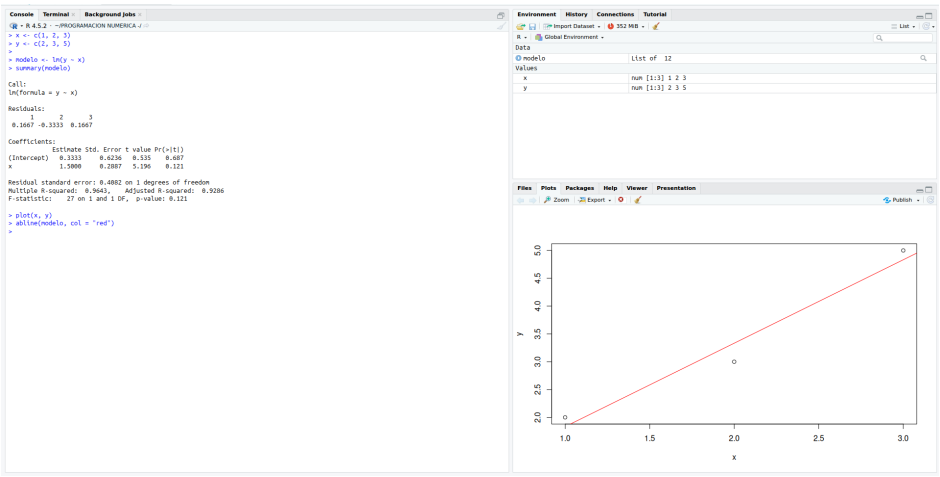


Figura 5.1: Visualización del ajuste por mínimos cuadrados en R.

5.4. Ejercico Resuelto

Ejercicio 5.1: Mínimos Cuadrados Ordinarios (OLS) Aplicación: Ingeniería Estadística – Predicción de consumo eléctrico

Planteamiento del problema: Una empresa eléctrica desea modelar el consumo de energía (kWh) en función de la temperatura promedio diaria (°C) para mejorar la planificación de la demanda. Se dispone de los siguientes datos experimentales:

Temperatura (x)	**Consumo** (y)
15	120
18	135
20	150
22	165
25	180

Se desea ajustar un modelo lineal de la forma:

$$y = \beta_0 + \beta_1 x$$

Formulación matemática: El problema se plantea como la minimización de la suma de errores cuadráticos:

$$S(\beta_0, \beta_1) = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$

Resolución manual (OLS clásico): Se calculan las sumatorias necesarias:

$$\sum x = 100, \quad \sum y = 750$$

$$\sum x^2 = 2068, \quad \sum xy = 15690$$

Las ecuaciones normales resultantes son:

$$\begin{cases} 5\beta_0 + 100\beta_1 = 750 \\ 100\beta_0 + 2068\beta_1 = 15690 \end{cases}$$

Resolviendo el sistema lineal:

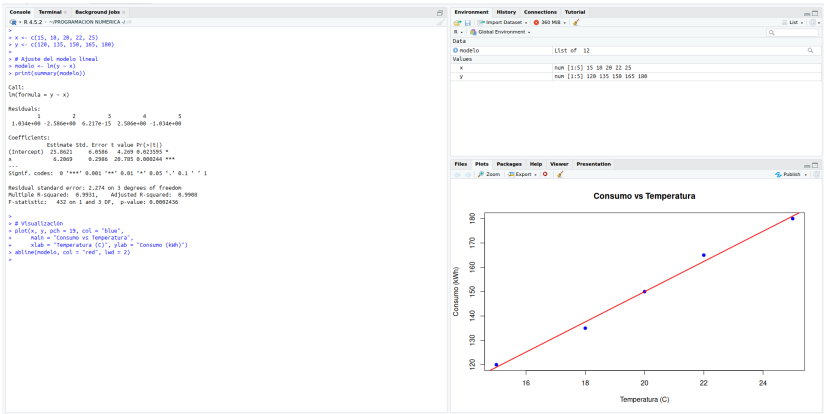
$$\beta_1 = 6, \quad \beta_0 = 30$$

Modelo ajustado:

$$\hat{y} = 30 + 6x$$

Implementación y Visualización en R:

```r
x <- c(15, 18, 20, 22, 25)
y <- c(120, 135, 150, 165, 180)

# Ajuste del modelo lineal
modelo <- lm(y ~ x)
print(summary(modelo))

# Visualización
plot(x, y, pch = 19, col = "blue",
main = "Consumo vs Temperatura",
xlab = "Temperatura (C)", ylab = "Consumo (kWh)")
abline(modelo, col = "red", lwd = 2)
```



Interpretación práctica:

- **Pendiente ($\beta_1 = 6$):** Por cada grado Celsius adicional, el consumo aumenta aproximadamente 6 kWh.
- **Utilidad:** El modelo permite predecir la demanda energética futura basándose en el pronóstico del clima, optimizando así la generación y los costos.

5.4.1. Ventajas y limitaciones del OLS

Ventajas:
- Solución exacta.
- Interpretación estadística clara.
- Eficiente para modelos pequeños.

Limitaciones:
- Sensible a outliers.
- Requiere supuestos estadísticos.
- No escala bien a grandes dimensiones.

5.5. Método de Descenso de Gradiente

5.5.1. Introducción al descenso de gradiente

El Descenso de Gradiente es un método iterativo de optimización utilizado para minimizar funciones diferenciables. Es el algoritmo base de:
- Machine learning.
- Redes neuronales.
- Optimización a gran escala.

5.5.2. Idea geométrica del método

La idea es avanzar en la dirección opuesta al gradiente:

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

donde α es la tasa de aprendizaje.

5.5.3. Derivación matemática

El gradiente indica la dirección de máximo crecimiento. Al avanzar en sentido contrario se garantiza la reducción de la función objetivo.

5.5.4. Algoritmo del descenso de gradiente

Pseudocódigo: Descenso de Gradiente

Entrada: x_0, Tasa α
Proceso:
- Mientras no converja:
  - $x = x - \alpha \cdot$ gradiente

Fin

5.5.5. Ejemplo manual (función cuadrática)

Minimización Manual

Minimizar:

$$f(x) = x^2 - 4x + 4$$

Gradiente:

$$f'(x) = 2x - 4$$

Iterando se converge a $x = 2$.

5.5.6. Implementación en R

```r
f <- function(x) x^2 - 4*x + 4
df <- function(x) 2*x - 4

x <- 0
alpha <- 0.1

for (i in 1:20) {
    x <- x - alpha * df(x)
}
x
```

Listing 5.3: Implementación de Descenso de Gradiente

5.5.7. Visualización del descenso

```r
curve(f(x), from = -1, to = 4)
points(x, f(x), col = "red")
```

Listing 5.4: Visualización del descenso

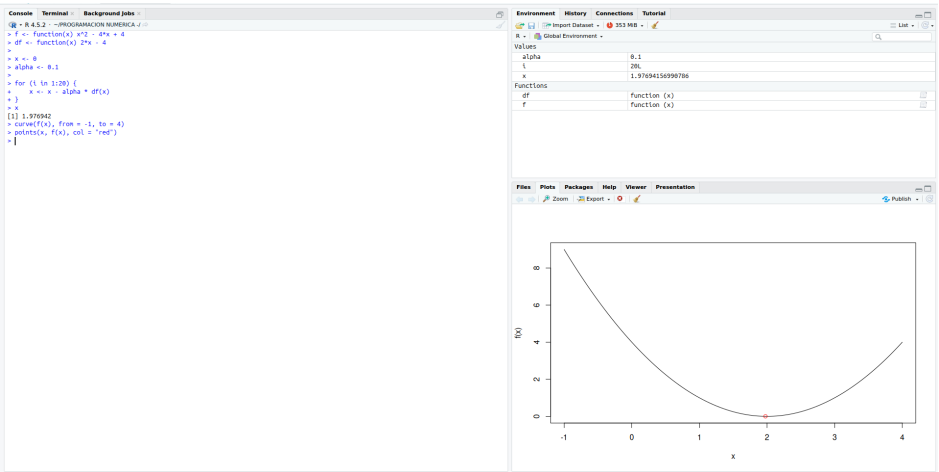


Figura 5.2: Trayectoria del descenso de gradiente.

Ejercicio 5.2: Descenso de Gradiente

Aplicación: Ciencia de Datos – Ajuste de modelo de costos
Planteamiento del problema: Una empresa desea minimizar el costo de producción en función de una variable de control x (por ejemplo, cantidad producida). El costo está modelado por la función convexa:

$$f(x) = x^2 - 10x + 30$$

Se desea encontrar el valor de x que minimiza el costo utilizando el algoritmo de *Descenso de Gradiente*.
Formulación matemática: El mínimo se busca iterativamente en la dirección opuesta al gradiente.

$$\nabla f(x) = f'(x) = 2x - 10$$

Regla de actualización: $x_{new} = x_{old} - \alpha \cdot f'(x_{old})$
Resolución manual (primeras iteraciones): Parámetros iniciales: $x_0 = 0$, tasa de aprendizaje $\alpha = 0.1$.
Iteración 1:

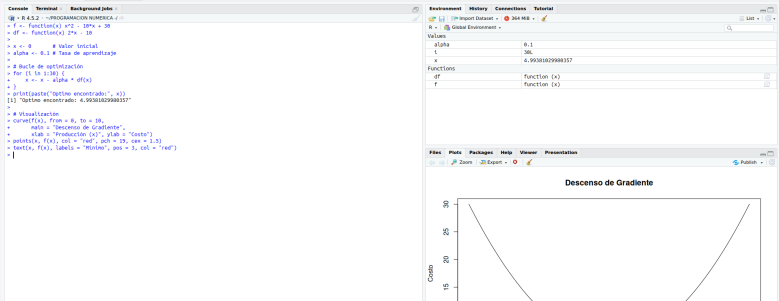
$$x_1 = 0 - 0.1(2(0) - 10) = 0 - 0.1(-10) = 1$$

Iteración 2:

$$x_2 = 1 - 0.1(2(1) - 10) = 1 - 0.1(-8) = 1.8$$

Tras varias iteraciones, el valor converge a $x \to 5$.
Implementación y Visualización en R:

```r
f <- function(x) x^2 - 10*x + 30
df <- function(x) 2*x - 10

x <- 0        # Valor inicial
alpha <- 0.1 # Tasa de aprendizaje

# Bucle de optimización
for (i in 1:30) {
    x <- x - alpha * df(x)
}
print(paste("Optimo encontrado:", x))

# Visualización
curve(f(x), from = 0, to = 10,
main = "Descenso de Gradiente",
xlab = "Producción (x)", ylab = "Costo")
points(x, f(x), col = "red", pch = 19, cex =
   1.5)
text(x, f(x), labels = "Mínimo", pos = 3, col =
   "red")
```



5.5.8. Comparación OLS vs Descenso de Gradiente

Característica	OLS	Gradiente
Solución	Cerrada	Iterativa
Escalabilidad	Baja	Alta
Dimensión	Pequeña	Grande
ML	No	Sí

Tabla 5.1: Comparativa entre OLS y Descenso de Gradiente

5.6. Aplicaciones Reales

5.6.1. Ciencia de datos y machine learning

- Regresión lineal.
- Redes neuronales.
- Optimización de funciones de pérdida.

5.6.2. Ingeniería y simulación

- Ajuste de modelos físicos.
- Optimización de procesos.
- Control automático.

5.6.3. Estadística aplicada

- Máxima verosimilitud.
- Modelos generalizados.
- Inferencia computacional.

5.7. Ejercicio Resuelto

Ejercicio 5.2: Descenso de Gradiente Aplicación: Ciencia de Datos – Ajuste de modelo de costos

Planteamiento del problema: Una empresa desea minimizar el costo de producción en función de una variable de control x (por ejemplo, cantidad producida). El costo está modelado por la función convexa:

$$f(x) = x^2 - 10x + 30$$

Se desea encontrar el valor de x que minimiza el costo utilizando el algoritmo de *Descenso de Gradiente*.

Formulación matemática: El mínimo se busca iterativamente en la dirección opuesta al gradiente.

$$\nabla f(x) = f'(x) = 2x - 10$$

Regla de actualización: $x_{new} = x_{old} - \alpha \cdot f'(x_{old})$

Resolución manual (primeras iteraciones): Parámetros iniciales: $x_0 = 0$, tasa de aprendizaje $\alpha = 0.1$.

Iteración 1:

$$x_1 = 0 - 0.1(2(0) - 10) = 0 - 0.1(-10) = 1$$

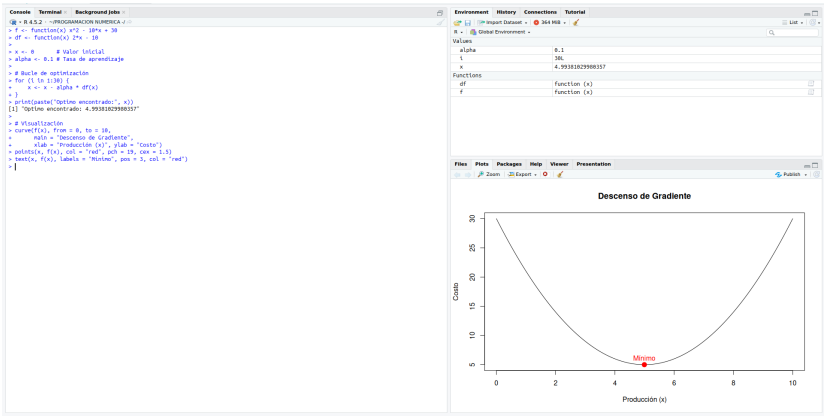
Iteración 2:

$$x_2 = 1 - 0.1(2(1) - 10) = 1 - 0.1(-8) = 1.8$$

Tras varias iteraciones, el valor converge a $x \to 5$.

Implementación y Visualización en R:

```r
f <- function(x) x^2 - 10*x + 30
df <- function(x) 2*x - 10

x <- 0        # Valor inicial
alpha <- 0.1 # Tasa de aprendizaje

# Bucle de optimización
for (i in 1:30) {
    x <- x - alpha * df(x)
}
print(paste("Optimo encontrado:", x))

# Visualización
curve(f(x), from = 0, to = 10,
main = "Descenso de Gradiente",
xlab = "Producción (x)", ylab = "Costo")
points(x, f(x), col = "red", pch = 19, cex = 1.5)
text(x, f(x), labels = "Mínimo", pos = 3, col = "
   red")
```



Interpretación práctica:

- El costo mínimo se alcanza en $x = 5$, que representa la producción óptima.
- El descenso de gradiente es fundamental en **Machine Learning** (ej. redes neuronales) cuando no existe una solución cerrada o el problema es de gran dimensión.

5.8. Ejercicios Propuestos

Ejercicio 5.1 — Ajuste de demanda de agua potable

Planteamiento: Una municipalidad desea modelar el consumo diario de agua (m^3) en función de la temperatura promedio (°C).

Temperatura (°C)	Consumo (m^3)
14	220
16	240
18	265
21	295
24	330

Se pide:

1. Plantear el modelo lineal $y = \beta_0 + \beta_1 x$.
2. Resolver el problema manualmente usando OLS (calculando sumatorias y ecuaciones normales).
3. Implementar el modelo en R usando la función `lm()`.
4. Graficar los datos y la recta ajustada.
5. Interpretar el significado físico de β_0 y β_1.

Ejercicio 5.2 — Relación entre horas de estudio y calificación

Planteamiento: Un estudio académico analiza la relación entre horas de estudio semanales y nota final.

Horas	Nota
2	8
4	11
6	14
8	16
10	18

Se pide:

1. Ajustar un modelo por mínimos cuadrados.
2. Calcular el error cuadrático medio (MSE).
3. Analizar si el modelo es adecuado para describir los datos.
4. Predecir la nota esperada para 7 horas de estudio.
5. Discutir las limitaciones del modelo lineal en este contexto (¿puede la nota crecer indefinidamente?).

Ejercicio 5.3 — OLS con ruido

Planteamiento: Se generan datos sintéticos siguiendo el modelo:

$$y = 3x + 5 + \varepsilon, \quad \varepsilon \sim N(0, 1)$$

Se pide:

1. Simular los datos en R.
2. Ajustar el modelo por OLS.
3. Comparar los parámetros estimados con los verdaderos (3 y 5).
4. Analizar el efecto del ruido en la estimación.
5. Graficar los residuos para verificar normalidad.

Ejercicio 5.4 — Interpretación estadística

Planteamiento: A partir de un modelo OLS dado por la ecuación:

$$\hat{y} = 12 + 4.5x$$

Se pide:

1. Interpretar el significado de ambos coeficientes en términos de cambio marginal e intercepto.
2. Proponer una situación real donde este modelo podría ser válido.
3. Indicar los supuestos fundamentales del modelo OLS (linealidad, homocedasticidad, etc.).
4. Analizar qué ocurre con la estimación si existen *outliers* (valores atípicos).

Ejercicio 5.5 — OLS vs optimización

Planteamiento: Este ejercicio conecta la estadística con el cálculo numérico.
Se pide:

1. Explicar por qué el método de OLS es, en esencia, un problema de optimización.
2. Identificar la función objetivo que se minimiza.
3. Explicar por qué la solución obtenida es un mínimo global y único (convexidad).
4. Relacionar el problema con la matriz Hessiana.

Ejercicio 5.6 — Minimización de costo de producción

Planteamiento: El costo de producción de una planta está dado por la función:

$$f(x) = x^2 - 12x + 40$$

Se pide:

1. Calcular el gradiente (derivada) de la función.
2. Encontrar el mínimo analíticamente igualando la derivada a cero.
3. Implementar el algoritmo de descenso de gradiente en R.
4. Analizar el efecto de utilizar distintas tasas de aprendizaje (α).
5. Comparar la solución numérica con la solución analítica.

Ejercicio 5.7 — Influencia de la tasa de aprendizaje

Planteamiento: Para la función simple:

$$f(x) = x^2$$

Se pide:

1. Aplicar descenso de gradiente con:
   - $\alpha = 0.01$
   - $\alpha = 0.1$
   - $\alpha = 1$ (o mayor)
2. Comparar la velocidad de convergencia en cada caso.
3. Identificar en qué caso ocurre divergencia (el algoritmo se aleja del mínimo).
4. Graficar las trayectorias de las iteraciones.
5. Concluir sobre la importancia de la elección correcta de α.

Ejercicio 5.8 — Descenso de gradiente en regresión

Planteamiento: Dado un conjunto de datos (x_i, y_i), se desea ajustar una recta sin usar las ecuaciones normales.

Se pide:

1. Formular la función de pérdida cuadrática (MSE o SSE).
2. Derivar el gradiente respecto a los parámetros β_0 y β_1.
3. Implementar el descenso de gradiente para estimar los parámetros iterativamente.
4. Comparar los resultados con los obtenidos mediante OLS clásico.
5. Analizar las ventajas computacionales de este método en grandes volúmenes de datos.

Ejercicio 5.9 — Optimización con múltiples variables

Planteamiento: Sea la función de dos variables:

$$f(x, y) = x^2 + y^2 - 4x - 6y + 13$$

Se pide:

1. Calcular el gradiente vectorial $\nabla f(x, y)$.
2. Encontrar el mínimo analítico resolviendo el sistema de ecuaciones.
3. Implementar el descenso de gradiente multivariable en R.
4. Visualizar el descenso sobre un gráfico de contorno.
5. Interpretar geométricamente el movimiento hacia el mínimo.

Capítulo 6

Juego de Supervivencia

6.1. Introducción

Los juegos de supervivencia constituyen modelos computacionales utilizados para simular la toma de decisiones, la asignación de recursos y la permanencia de agentes bajo un conjunto de reglas bien definidas. En el ámbito de la Programación Numérica, estos modelos permiten analizar fenómenos aleatorios, procesos estocásticos, simulaciones Monte Carlo y comportamientos emergentes derivados de interacciones simples.

En este capítulo se desarrolla un experimento de reparto de dulces como un modelo simplificado de supervivencia y recompensa. En dicho modelo, los alumnos representan agentes del sistema y los chupetines simbolizan recursos esenciales para la supervivencia. A través del azar y de reglas de recompensa, se estudia cómo se distribuyen los recursos y se garantiza un nivel mínimo de estabilidad del sistema.

6.2. Descripción del experimento

El experimento simula el reparto aleatorio de dulces entre un conjunto de 9 alumnos. Cada alumno recibe exactamente 2 caramelos extraídos al azar de una bolsa común.

Los tipos de caramelos disponibles son:

- Blanco
- Redondo
- Amarillo

Según la combinación de caramelos obtenida por cada alumno, se asigna una cantidad determinada de chupetines, los cuales representan una recompensa o recurso de supervivencia dentro del modelo.

6.3. Reglas del juego de supervivencia

Las reglas del juego determinan la cantidad de chupetines que recibe cada alumno en función de la combinación de caramelos obtenida:

- Si el alumno recibe al menos un caramelo de cada tipo, obtiene **1 chupetín**.
- Si el alumno recibe dos caramelos de cada tipo (6 en total), obtiene **2 chupetines** y **1 caramelo extra**.
- Si no cumple ninguna de las condiciones anteriores, recibe **1 chupetín mínimo**.

Estas reglas garantizan que ningún jugador quede sin recursos, simulando un entorno de supervivencia básica donde siempre existe un mínimo vital.

6.4. Modelado matemático

Sea el conjunto de tipos de caramelos:

$$A = \{\text{Blanco}, \text{Redondo}, \text{Amarillo}\}$$

Para cada alumno i, se define X_i como el conjunto de caramelos asignados. La función de recompensa $R(X_i)$ se define de forma escalonada como:

$$R(X_i) = \begin{cases} 2, & \text{si } X_i \text{ contiene al menos dos caramelos de cada tipo} \\ 1, & \text{si } X_i \text{ contiene al menos un caramelo de cada tipo} \\ 1, & \text{en caso contrario} \end{cases}$$

Este tipo de función permite modelar sistemas con recompensas discretas y umbrales de supervivencia.

6.5. Algoritmo numérico asociado

El algoritmo del juego de supervivencia sigue los siguientes pasos:

1. Definir los tipos de caramelos disponibles.
2. Generar una bolsa de caramelos con reposición.
3. Repartir aleatoriamente los caramelos entre los alumnos.
4. Contar la cantidad de cada tipo de caramelo por alumno.
5. Asignar la recompensa según las reglas establecidas.

Este procedimiento puede repetirse múltiples veces para analizar el comportamiento promedio del sistema mediante simulaciones.

6.6. Implementación computacional en R

A continuación, se presenta la implementación del experimento utilizando el lenguaje de programación R:

```r
set.seed(123)

# Definición de tipos de caramelos
tipos_caramelo <- c("Blanco", "Redondo", "Amarillo")

# Generación aleatoria de una bolsa con 60 caramelos
bolsa_caramelos <- sample(tipos_caramelo, 60, replace =
    TRUE)

# Función para determinar la cantidad de chupetines
   obtenidos
obtener_chupetines <- function(caramelos) {
    n_blanco   <- sum(caramelos == "Blanco")
```

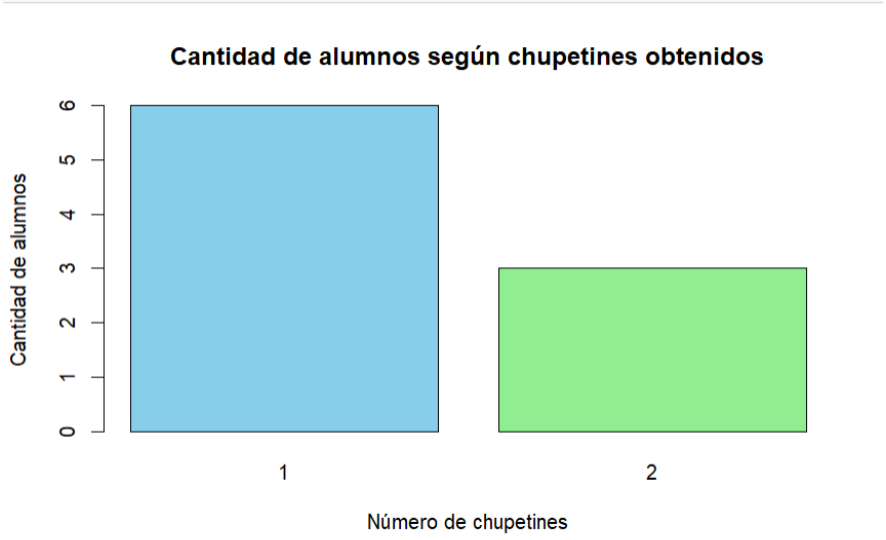
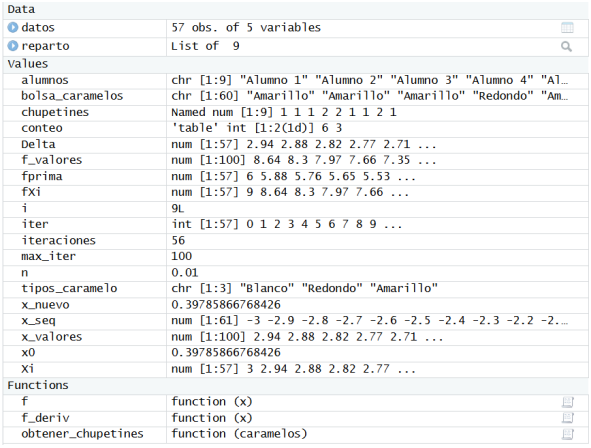


Figura 6.1: Simulacion del juego

```
    n_redondo  <- sum(caramelos == "Redondo")
    n_amarillo <- sum(caramelos == "Amarillo")

    if (n_blanco >= 2 && n_redondo >= 2 && n_amarillo
       >= 2) {
        return(2)
    } else if (n_blanco >= 1 && n_redondo >= 1 && n_
       amarillo >= 1) {
        return(1)
    } else {
        return(0)
    }
}

# Asignación de caramelos a los alumnos
alumnos <- paste("Alumno", 1:9)
reparto <- lapply(1:9, function(x) sample(bolsa_
   caramelos, 2))
names(reparto) <- alumnos

# Cálculo de chupetines obtenidos
chupetines <- sapply(reparto, obtener_chupetines)
chupetines[chupetines == 0] <- 1

# Resultados
print(reparto)
print(chupetines)
```

Listing 6.1: Simulación del juego de supervivencia en R

En el Código 6.1 se presenta la simulación realizada en R, mientras que la Figura muestra la tabla resultante del reparto de dulces.



Data	
datos	57 obs. of 5 variables
reparto	List of 9
Values	
alumnos	chr [1:9] "Alumno 1" "Alumno 2" "Alumno 3" "Alumno 4" "Al…
bolsa_caramelos	chr [1:60] "Amarillo" "Amarillo" "Amarillo" "Redondo" "Am…
chupetines	Named num [1:9] 1 1 1 2 2 1 1 2 1
conteo	'table' int [1:2(1d)] 6 3
Delta	num [1:57] 2.94 2.88 2.82 2.77 2.71 ...
f_valores	num [1:100] 8.64 8.3 7.97 7.66 7.35 ...
fprima	num [1:57] 6 5.88 5.76 5.65 5.53 ...
fXi	num [1:57] 9 8.64 8.3 7.97 7.66 ...
i	9L
iter	int [1:57] 0 1 2 3 4 5 6 7 8 9 ...
iteraciones	56
max_iter	100
n	0.01
tipos_caramelo	chr [1:3] "Blanco" "Redondo" "Amarillo"
x_nuevo	0.39785866768426
x_seq	num [1:61] -3 -2.9 -2.8 -2.7 -2.6 -2.5 -2.4 -2.3 -2.2 -2.…
x_valores	num [1:100] 2.94 2.88 2.82 2.77 2.71 ...
x0	0.39785866768426
Xi	num [1:57] 3 2.94 2.88 2.82 2.77 ...
Functions	
f	function (x)
f_deriv	function (x)
obtener_chupetines	function (caramelos)

Figura 6.2: Tabla de simulacion de juegos

6.7. Análisis de resultados

El modelo permite analizar la influencia del azar en la asignación de recursos, la distribución de recompensas y la estabilidad general del sistema. Al garantizar una recompensa mínima, se evita la extinción de agentes, característica común en modelos de supervivencia controlada.

Asimismo, el experimento evidencia cómo reglas simples pueden generar distribuciones variadas de recursos, fenómeno conocido como comportamiento emergente.

6.8. Extensiones del modelo

El juego de supervivencia puede ampliarse de diversas maneras:

- Aumentar el número de alumnos.
- Modificar las reglas de recompensa.
- Introducir penalizaciones.
- Aplicar simulaciones Monte Carlo.

6.9. Ejercicios propuestos

1. Modifique el código para trabajar con 20 alumnos.
2. Cambie las reglas de recompensa e interprete los resultados.
3. Ejecute la simulación 100 veces y calcule el promedio de chupetines.
4. Analice cómo varía la distribución de recompensas al cambiar la cantidad de caramelos.

6.10. Tabla de resultados

La Tabla 6.1 muestra un ejemplo de los chupetines obtenidos por cada alumno en una ejecución del experimento.

Alumno	Chupetines obtenidos
Alumno 1	1
Alumno 2	1
Alumno 3	2
Alumno 4	1
Alumno 5	1
Alumno 6	1
Alumno 7	2
Alumno 8	1
Alumno 9	1

Tabla 6.1: Resultados del reparto de chupetines

6.11. Simulación Monte Carlo

Con el objetivo de analizar el comportamiento promedio del sistema, se realiza una simulación Monte Carlo ejecutando el experimento un gran número de veces. Este enfoque permite estimar el valor esperado de los chupetines obtenidos y evaluar la estabilidad del modelo.

```
simulacion <- function() {
    reparto <- replicate(9, sample(tipos_caramelo, 2))
    recompensas <- apply(reparto, 2, obtener_chupetines
       )
    recompensas[recompensas == 0] <- 1
    sum(recompensas)
}

resultados <- replicate(1000, simulacion())
mean(resultados)
```

Listing 6.2: Simulación Monte Carlo del juego de supervivencia

El valor promedio obtenido representa una estimación del número esperado de chupetines en el sistema bajo condiciones aleatorias.

Capítulo 7

La Pseudoinversa de Moore–Penrose

7.1. Introducción y Motivación Histórica

En programación numérica, los sistemas de ecuaciones lineales aparecen de forma recurrente en aplicaciones científicas y de ingeniería. Sin embargo, en la práctica, dichos sistemas rara vez cumplen las condiciones ideales necesarias para aplicar la inversa clásica. La **pseudoinversa de Moore–Penrose** surge como una generalización robusta que permite trabajar con matrices rectangulares o singulares, proporcionando soluciones óptimas en el sentido de mínimos cuadrados.

Contexto histórico

- **1920**: Eliakim H. Moore introduce el concepto de inversa generalizada en su trabajo .ºn the reciprocal of the general algebraic matrix".
- **1955**: Roger Penrose formaliza las cuatro condiciones que definen la pseudoinversa en .ª generalized inverse for matrices".
- **1960s**: Uso extensivo en estadística, teoría de control y análisis de sistemas.
- **1970s**: Implementación en librerías numéricas como LINPACK y LAPACK.
- **1980s**: Aplicación en procesamiento de señales e imágenes.
- **1990s**: Uso en sistemas de recomendación y recuperación de información.
- **Actualidad**: Pilar fundamental en machine learning, procesamiento de señales y análisis de datos masivos.

Motivación práctica

La necesidad de la pseudoinversa surge en múltiples contextos:

1. Sistemas físicos donde las mediciones exceden las variables desconocidas.
2. Problemas de calibración con más ecuaciones que parámetros.
3. Ajuste de modelos estadísticos con datos redundantes.
4. Reconstrucción de señales a partir de mediciones incompletas.

7.2. Fundamentos Teóricos: Sistemas de Ecuaciones Lineales

Un sistema lineal se expresa como:

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{m\times n},\ \mathbf{x} \in \mathbb{R}^n,\ \mathbf{b} \in \mathbb{R}^m$$

7.2.1. Clasificación detallada

Tipo	Condición	Naturaleza	Ejemplo típico
Determinado	$m = n$, rango$(A) = n$	Solución única	Calibración precisa
Sobredeterminado	$m > n$, rango$(A) = n$	Mínimos cuadrados	Ajuste de curvas
Subdeterminado	$m < n$, rango$(A) = m$	Infinitas soluciones	Interpolación
Degenerado	rango$(A) <$ mín(m, n)	Problemas	Sensores defectuosos

7.2.2. Espacios fundamentales asociados

Para una matriz $A \in \mathbb{R}^{m\times n}$:

- **Espacio columna**: $\mathcal{C}(A) = \{A\mathbf{x} \mid \mathbf{x} \in \mathbb{R}^n\}$
- **Espacio nulo**: $\mathcal{N}(A) = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{0}\}$
- **Espacio fila**: $\mathcal{R}(A) = \mathcal{C}(A^T)$
- **Espacio nulo izquierdo**: $\mathcal{N}(A^T) = \{\mathbf{y} \in \mathbb{R}^m \mid A^T\mathbf{y} = \mathbf{0}\}$

7.2.3. Teorema fundamental del álgebra lineal

$$\mathbb{R}^n = \mathcal{N}(A) \oplus \mathcal{R}(A^T), \quad \mathbb{R}^m = \mathcal{N}(A^T) \oplus \mathcal{R}(A)$$

7.3. El Problema de Mínimos Cuadrados: Formulación General

El problema clásico de mínimos cuadrados se formula como:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_2^2$$

7.3.1. Interpretación geométrica

Geométricamente, se busca la proyección ortogonal del vector $\mathbf{b}$ sobre el espacio columna de A. Esta proyección minimiza la distancia euclidiana entre $\mathbf{b}$ y $\mathcal{C}(A)$.

7.3.2. Ecuaciones normales

Derivando la función objetivo se obtienen las ecuaciones normales:

$$A^T A\mathbf{x} = A^T\mathbf{b}$$

Propiedades:

- Si A tiene rango completo, A^TA es definida positiva.
- La solución es única: $\mathbf{x}^* = (A^TA)^{-1}A^T\mathbf{b}$.
- Numéricamente, $\kappa(A^TA) = \kappa(A)^2$, lo que puede causar inestabilidad.

7.3.3. Solución en casos degenerados

Cuando A no tiene rango completo, el problema tiene infinitas soluciones. Entre todas ellas, se busca la de **norma mínima**:

$$\text{mín } \|\mathbf{x}\|_2^2 \quad \text{sujeto a} \quad \mathbf{x} \in \arg\text{mín } \|A\mathbf{x} - \mathbf{b}\|_2^2$$

7.4. Definición Formal de la Pseudoinversa de Moore-Penrose

Definición (Moore-Penrose, 1955): Sea $A \in \mathbb{C}^{m\times n}$. La pseudoinversa $A^+ \in \mathbb{C}^{n\times m}$ es la única matriz que satisface las siguientes cuatro condiciones:

$(MP1)$	$AA^+A = A$	(Consistencia débil)
$(MP2)$	$A^+AA^+ = A^+$	(Inversa débil)
$(MP3)$	$(AA^+)^* = AA^+$	(Proyección ortogonal derecha)
$(MP4)$	$(A^+A)^* = A^+A$	(Proyección ortogonal izquierda)

En matrices reales ($\mathbb{R}^{m\times n}$), el operador * se reemplaza por la transpuesta ^{T}.

7.4.1. Interpretación de las condiciones

- **(MP1)**: A^+ actúa como inversa por la izquierda cuando se aplica a la imagen de A.
- **(MP2)**: A actúa como inversa de A^+ por la derecha.
- **(MP3)**: AA^+ es hermítica (simétrica en $\mathbb{R}$), por tanto es un proyector ortogonal sobre $\mathcal{C}(A)$.
- **(MP4)**: A^+A es hermítica, por tanto es un proyector ortogonal sobre $\mathcal{R}(A)$.

7.4.2. Unicidad y existencia

[Unicidad de la pseudoinversa] Para cualquier matriz $A \in \mathbb{C}^{m\times n}$, existe a lo sumo una matriz $X \in \mathbb{C}^{n\times m}$ que satisface las cuatro condiciones de Moore-Penrose.

Demostración. Supongamos que X e Y son dos pseudoinversas de A. Entonces:

$$\begin{aligned} X &= XAX = X(AY)A = XA(YA)Y = (XA)^*(YA)^*Y \\ &= A^*X^*A^*Y^*Y = A^*(A^+)^*A^*(A^+)^*Y = \cdots = Y \end{aligned}$$

□

7.5. Propiedades Algebraicas y Geométricas Detalladas

7.5.1. Propiedades algebraicas básicas

1. **Involución**: $(A^+)^+ = A$
2. **Transposición**: $(A^T)^+ = (A^+)^T$
3. **Conjugación**: $(\overline{A})^+ = \overline{A^+}$
4. **Adjunta**: $(A^*)^+ = (A^+)^*$
5. **Escalares**: Para $\alpha \in \mathbb{C}$,
$$(\alpha A)^+ = \begin{cases} \alpha^{-1}A^+ & \text{si } \alpha \neq 0 \\ 0 & \text{si } \alpha = 0 \end{cases}$$
6. **Rango**: $\text{rango}(A^+) = \text{rango}(A)$
7. **Valores singulares**: Si σ_i son valores singulares de A, entonces $1/\sigma_i$ (para $\sigma_i > 0$) son valores singulares de A^+.

7.5.2. Propiedades de composición

- En general, $(AB)^+ \neq B^+A^+$
- Si A tiene columnas ortonormales: $(AB)^+ = B^+A^+$
- Si B tiene filas ortonormales: $(AB)^+ = B^+A^+$
- Para matrices diagonales: $(\text{diag}(d_1, \ldots, d_n))^+ = \text{diag}(d_1^+, \ldots, d_n^+)$

7.5.3. Propiedades geométricas avanzadas

[Descomposición ortogonal] Para cualquier $A \in \mathbb{C}^{m\times n}$:
$$\mathbb{C}^n = \mathcal{N}(A) \oplus \mathcal{R}(A^+)$$
$$\mathbb{C}^m = \mathcal{N}(A^+) \oplus \mathcal{R}(A)$$

[Proyecciones ortogonales] Las matrices AA^+ y A^+A son proyectores ortogonales:
$$P_{\mathcal{R}(A)} = AA^+ \quad \text{(proyección sobre el espacio columna de } A\text{)}$$
$$P_{\mathcal{R}(A^T)} = A^+A \quad \text{(proyección sobre el espacio fila de } A\text{)}$$

7.6. Cálculo de la Pseudoinversa: Métodos y Algoritmos

7.6.1. Métodos analíticos para casos especiales

Matrices de rango completo

- **Caso sobredeterminado** ($m > n$, $\text{rango}(A) = n$):
$$A^+ = (A^TA)^{-1}A^T$$
- **Caso subdeterminado** ($m < n$, $\text{rango}(A) = m$):
$$A^+ = A^T(AA^T)^{-1}$$

Matrices diagonalizables

Si $A = PDP^{-1}$ con D diagonal, entonces:

$$A^+ = PD^+P^{-1}$$

7.6.2. Método general mediante Descomposición en Valores Singulares (SVD)

Teorema de la SVD

Toda matriz $A \in \mathbb{R}^{m\times n}$ puede factorizarse como:

$$A = U\Sigma V^T$$

donde:

- $U \in \mathbb{R}^{m\times m}$: matriz ortogonal (vectores singulares izquierdos)
- $V \in \mathbb{R}^{n\times n}$: matriz ortogonal (vectores singulares derechos)
- $\Sigma \in \mathbb{R}^{m\times n}$: matriz diagonal con $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$
- $r = \text{rango}(A)$

Cálculo de la pseudoinversa vía SVD

Dada la SVD $A = U\Sigma V^T$, se define:

$$\Sigma^+ = \begin{bmatrix} \Sigma_r^{-1} & 0 \\ 0 & 0 \end{bmatrix}^T \in \mathbb{R}^{n\times m}$$

donde $\Sigma_r = \text{diag}(\sigma_1, \ldots, \sigma_r)$.

Entonces:

$$A^+ = V\Sigma^+U^T$$

Algoritmo paso a paso

1. Calcular la SVD completa: $A = U\Sigma V^T$
2. Determinar el rango numérico r (valores singulares mayores que ϵ)
3. Construir Σ^+ reemplazando σ_i por $1/\sigma_i$ para $i = 1, \ldots, r$
4. Calcular $A^+ = V\Sigma^+U^T$

7.6.3. Algoritmo de Greville (método recursivo)

Para matrices grandes, el algoritmo de Greville calcula A^+ de forma recursiva:

Algoritmo 1: Algoritmo de Greville para calcular A^+

1 [1] $A \in \mathbb{R}^{m\times n}$ $A^+ \in \mathbb{R}^{n\times m}$ $A_1 \leftarrow$ primera columna de A $A_1^+ \leftarrow (A_1^TA_1)^{-1}A_1^T$
$k = 2$ to n $\mathbf{a}_k \leftarrow$ columna k de A $\mathbf{d}_k \leftarrow A_{k-1}^+\mathbf{a}_k$ $\mathbf{c}_k \leftarrow \mathbf{a}_k - A_{k-1}\mathbf{d}_k$ $\|\mathbf{c}_k\| > \epsilon$
$\mathbf{b}_k^T \leftarrow (\mathbf{c}_k^T\mathbf{c}_k)^{-1}\mathbf{c}_k^T$ $\mathbf{b}_k^T \leftarrow (1 + \|\mathbf{d}_k\|^2)^{-1}\mathbf{d}_k^TA_{k-1}^+$ $A_k^+ \leftarrow \begin{bmatrix} A_{k-1}^+ - \mathbf{d}_k\mathbf{b}_k^T \\ \mathbf{b}_k^T \end{bmatrix}$ A_n^+

7.7. Descomposición en Valores Singulares: Teoría y Aplicaciones

7.7.1. Propiedades fundamentales de la SVD

[Eckart-Young-Mirsky] Para una matriz $A \in \mathbb{R}^{m\times n}$ con SVD $A = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$, la mejor aproximación de rango $k \leq r$ en norma Frobenius es:

$$A_k = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

y satisface:

$$\|A - A_k\|_F = \sqrt{\sum_{i=k+1}^{r} \sigma_i^2}$$

7.7.2. Interpretación geométrica de los valores singulares

- σ_1: máxima amplificación de A ($\|A\mathbf{x}\| \leq \sigma_1 \|\mathbf{x}\|$)
- σ_r: mínima amplificación para $\mathbf{x} \in \mathcal{R}(A^T)$
- σ_i/σ_{i+1}: medida de separación entre subespacios

7.7.3. Número de condición

El número de condición de una matriz se define como:

$$\kappa(A) = \frac{\sigma_1}{\sigma_r}$$

y cuantifica la sensibilidad de la solución a perturbaciones en los datos.

7.8. Implementación Computacional y Análisis Numérico

7.8.1. Implementaciones en diferentes lenguajes

Python (NumPy/SciPy)

```
import numpy as np
import scipy.linalg as la

# Método 1: numpy.linalg.pinv (usando SVD)
A_pinv = np.linalg.pinv(A, rcond=1e-15)

# Método 2: scipy.linalg.pinv (con control de rango)
A_pinv2 = la.pinv(A, cond=1e-15, return_rank=False)

# Método 3: Manual con SVD económica
U, S, Vt = np.linalg.svd(A, full_matrices=False)
```

```
> # ================ RESULTADOS ================
> cat("Pseudoinversa con MASS::ginv\n")
Pseudoinversa con MASS::ginv
> print(A_pinv1)
          [,1]       [,2]       [,3]
[1,] -1.333333 -0.3333333  0.6666667
[2,]  1.083333  0.3333333 -0.4166667
>
> cat("\nPseudoinversa con pracma::pinv\n")

Pseudoinversa con pracma::pinv
> print(A_pinv2)
Error: objeto 'A_pinv2' no encontrado
>
> cat("\nPseudoinversa con SVD base R\n")

Pseudoinversa con SVD base R
> print(A_pinv3)
Error: objeto 'A_pinv3' no encontrado
```

Figura 7.1: Resultado del codigo

```
S_inv = np.zeros(A.shape[::-1])
r = np.sum(S > 1e-10 * S[0])
S_inv[:r, :r] = np.diag(1/S[:r])
A_pinv3 = Vt.T @ S_inv @ U.T
```

R

```
library(MASS)
library(pracma)

# Método 1: MASS::ginv (generalizada)
A_pinv <- ginv(A)

# Método 2: pracma::pinv
A_pinv2 <- pinv(A, tol = .Machine$double.eps^(2/3))

# Método 3: Base R con svd
svd_A <- svd(A)
U <- svd_A$u
S <- diag(svd_A$d)
V <- svd_A$v
r <- sum(svd_A$d > 1e-10 * svd_A$d[1])
S_inv <- matrix(0, ncol(A), nrow(A))
S_inv[1:r, 1:r] <- diag(1/svd_A$d[1:r])
A_pinv3 <- V %*% S_inv %*% t(U)
```

```
= RESTART: C:/Users/HP/Documents/LENGUAJE DE PROGRAMACI
py
pinv numpy:
 [[-1.33333333 -0.33333333  0.66666667]
 [ 1.08333333  0.33333333 -0.41666667]]

SVD manual:
 [[-1.33333333 -0.33333333  0.66666667]
 [ 1.08333333  0.33333333 -0.41666667]]
Traceback (most recent call last):
  File "C:/Users/HP/Documents/LENGUAJE DE PROGRAMACION
, line 2, in <module>
    import scipy.linalg as la
ModuleNotFoundError: No module named 'scipy'
```

Figura 7.2: Resultados del codigo

MATLAB/Octave

```
% Método 1: Función incorporada
A_pinv = pinv(A);

% Método 2: Con tolerancia específica
tol = max(size(A)) * eps(norm(A));
A_pinv2 = pinv(A, tol);

% Método 3: Manual con SVD económica
[U, S, V] = svd(A, 'econ');
r = sum(diag(S) > tol * S(1,1));
S_inv = zeros(size(S'));
S_inv(1:r, 1:r) = diag(1./diag(S(1:r,1:r)));
A_pinv3 = V * S_inv * U';
```

7.8.2. Análisis de complejidad computacional

Método	Complejidad	Memoria	Estabilidad
Ecuaciones normales	$O(mn^2 + n^3)$	$O(n^2)$	Baja
QR	$O(mn^2)$	$O(mn)$	Media
SVD completa	$O(m^2n + n^3)$	$O(m^2 + n^2)$	Alta
SVD económica	$O(mn^2)$	$O(mn)$	Alta
Greville	$O(mn^2)$	$O(mn)$	Media

7.8.3. Consideraciones de precisión numérica

- **Tolerancia**: Se debe elegir cuidadosamente para determinar el rango numérico.
- **Pérdida de ortogonalidad**: En algoritmos iterativos.
- **Underflow/overflow**: En el cálculo de $1/\sigma_i$ para valores singulares muy pequeños/grandes.
- **Backward stability**: La SVD es backward stable.

7.9. Comparación Crítica: Pseudoinversa vs. Inversa Clásica

Aspecto	Inversa Clásica (A^{-1})	Pseudoinversa (A^{+})
Existencia	Solo para matrices cuadradas no singulares ($\det(A) \neq 0$)	Siempre existe para cualquier matriz
Dimensiones	$A^{-1} \in \mathbb{R}^{n \times n}$	$A^{+} \in \mathbb{R}^{n \times m}$
Propiedades algebraicas	$AA^{-1} = A^{-1}A = I_n$	$AA^{+}A = A$, $A^{+}AA^{+} = A^{+}$
Solución de $A\mathbf{x} = \mathbf{b}$	$\mathbf{x} = A^{-1}\mathbf{b}$ (única)	$\mathbf{x} = A^{+}\mathbf{b}$ (mínimos cuadrados, norma mínima)
Estabilidad numérica	Sensible a mal condicionamiento	Más estable (especialmente vía SVD)
Costo computacional	$O(n^3)$ para LU/Gauss	$O(\min(m,n)^2 \max(m,n))$ para SVD
Aplicación típica	Sistemas determinados bien condicionados	Regresión, ajuste, sistemas incompatibles
Interpretación geométrica	Transformación inversa	Proyección + transformación inversa parcial

Tabla 7.1: Comparación detallada entre inversa clásica y pseudoinversa

7.10. Pseudoinversa vs. Métodos Iterativos: Gradient Descent y Variantes

7.10.1. Formulación del problema común

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) = \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2$$

7.10.2. Solución analítica (Moore-Penrose)

$$\mathbf{x}^* = A^{+}\mathbf{b}$$

Ventajas:
- Solución exacta (salvo errores numéricos)
- No requiere iteraciones
- Óptima en norma

Desventajas:
- Costo $O(\min(m,n)^2 \max(m,n))$
- Requiere almacenar matrices completas
- No escalable a dimensiones muy grandes

7.10.3. Métodos iterativos

Gradient Descent clásico

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k) = \mathbf{x}_k - \alpha A^T(A\mathbf{x}_k - \mathbf{b})$$

- **Tasa de convergencia**: Lineal, $O(\kappa^2)$ iteraciones
- **Parámetro óptimo**: $\alpha = 2/(\sigma_1^2 + \sigma_n^2)$
- **Ventajas**: Bajo costo por iteración ($O(mn)$), bajo uso de memoria

Método del Gradiente Conjugado

$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ $\mathbf{p}_0 = \mathbf{r}_0$ $k = 0, 1, 2, \ldots$ $\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T A^T A \mathbf{p}_k}$ $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A^T A \mathbf{p}_k$ $\beta_k = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$ $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$

Método LSQR (Paige y Saunders)

Especializado para mínimos cuadrados, basado en bidiagonalización de Golub-Kahan.

7.10.4. Comparación cuantitativa

Método	Iteraciones	Costo/iteración	Precisión
Moore-Penrose (SVD)	-	$O(\text{mín}(m,n)^2 \,\text{máx}(m,n))$	Exacta
Gradient Descent	$O(\kappa^2 \log(1/\epsilon))$	$O(mn)$	ϵ
Gradiente Conjugado	$O(\sqrt{\kappa} \log(1/\epsilon))$	$O(mn)$	ϵ
LSQR	$O(\sqrt{\kappa} \log(1/\epsilon))$	$O(mn)$	ϵ

Tabla 7.2: Comparación de métodos para mínimos cuadrados ($\kappa = \text{cond}(A^T A)$)

7.10.5. Código comparativo en Python

```
import numpy as np
import time

def solve_moore_penrose(A, b):
"""Solución exacta vía pseudoinversa."""
return np.linalg.pinv(A) @ b

def solve_gradient_descent(A, b, alpha=0.01, max_iter=10000, tol=1e-10):
"""Descenso por gradiente para mínimos cuadrados."""
m, n = A.shape
x = np.zeros(n)
for i in range(max_iter):
grad = A.T @ (A @ x - b)
x_new = x - alpha * grad
if np.linalg.norm(x_new - x) < tol:
```

```
break
x = x_new
return x

def solve_conjugate_gradient(A, b, max_iter=1000, tol=1e-10):
"""Gradiente conjugado para A^T A x = A^T b."""
m, n = A.shape
x = np.zeros(n)
r = A.T @ (b - A @ x)
p = r.copy()
rsold = r.T @ r

for i in range(max_iter):
Ap = A.T @ (A @ p)
alpha = rsold / (p.T @ Ap)
x = x + alpha * p
r = r - alpha * Ap
rsnew = r.T @ r
if np.sqrt(rsnew) < tol:
break
p = r + (rsnew / rsold) * p
rsold = rsnew
return x

# Evaluación comparativa
m, n = 1000, 100
A = np.random.randn(m, n)
b = np.random.randn(m)

methods = {
 "Moore-Penrose": solve_moore_penrose,
 "Gradient Descent": lambda A,b: solve_gradient_descent(A,b,alpha=0.001),
 "Conjugate Gradient": solve_conjugate_gradient
}

for name, method in methods.items():
start = time.time()
x = method(A, b)
elapsed = time.time() - start
error = np.linalg.norm(A @ x - b)
print(f"{name}: {elapsed:.4f}s, error={error:.6f}")
```

```
Comparación de métodos:

Moore-Penrose         | Tiempo: 0.0524 s | Error: 29.832281
Gradient Descent      | Tiempo: 0.0035 s | Error: 29.832281
Conjugate Gradient    | Tiempo: 0.0016 s | Error: 29.832281
```

Figura 7.3: Enter Caption

7.11. Aplicación a Regresión Lineal: Teoría y Práctica

7.11.1. Modelo de regresión lineal general

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim N(0, \sigma^2 I_n)$$

donde:

- $\mathbf{y} \in \mathbb{R}^n$: variable respuesta
- $X \in \mathbb{R}^{n\times p}$: matriz de diseño
- $\boldsymbol{\beta} \in \mathbb{R}^p$: coeficientes del modelo
- $\boldsymbol{\varepsilon} \in \mathbb{R}^n$: errores aleatorios

7.11.2. Estimador de mínimos cuadrados ordinarios (OLS)

$$\hat{\boldsymbol{\beta}}_{\text{OLS}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - X\boldsymbol{\beta}\|^2$$

Solución:

$$\hat{\boldsymbol{\beta}}_{\text{OLS}} = X^+\mathbf{y}$$

7.11.3. Propiedades estadísticas del estimador

[Insesgadez] Si $\text{rango}(X) = p$, entonces $E[\hat{\boldsymbol{\beta}}_{\text{OLS}}] = \boldsymbol{\beta}$.

[Varianza] $\text{Var}(\hat{\boldsymbol{\beta}}_{\text{OLS}}) = \sigma^2(X^TX)^{-1}$.

[Gauss-Markov] Entre todos los estimadores lineales insesgados, $\hat{\boldsymbol{\beta}}_{\text{OLS}}$ tiene varianza mínima.

7.11.4. Casos especiales

Regresión simple ($p = 2$)

$$X = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}, \quad \hat{\boldsymbol{\beta}} = \begin{bmatrix} \bar{y} - \hat{\beta}_1\bar{x} \\ \frac{\sum(x_i-\bar{x})(y_i-\bar{y})}{\sum(x_i-\bar{x})^2} \end{bmatrix}$$

Regresión polinomial

Para ajuste polinomial de grado d:

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^d \end{bmatrix}$$

Regresión con intercepto forzado

Para $\beta_0 = 0$:

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad \hat{\beta} = \frac{\sum x_i y_i}{\sum x_i^2}$$

7.11.5. Implementación completa de regresión lineal

```
class LinearRegressionMP:
"""Regresión lineal usando pseudoinversa de Moore-Penrose."""

def __init__(self, method='svd', add_intercept=True, rcond=None):
self.method = method  # 'svd', 'normal', 'qr'
self.add_intercept = add_intercept
self.rcond = rcond
self.coef_ = None
self.intercept_ = None
self.residuals_ = None
self.rank_ = None

def fit(self, X, y):
# Añadir intercepto si es necesario
if self.add_intercept:
X = np.column_stack([np.ones(X.shape[0]), X])

# Calcular pseudoinversa según método
if self.method == 'svd':
U, s, Vt = np.linalg.svd(X, full_matrices=False)
if self.rcond is None:
tol = s[0] * max(X.shape) * np.finfo(float).eps
r = np.sum(s > tol)
else:
r = np.sum(s > self.rcond * s[0])
s_inv = np.zeros_like(s)
s_inv[:r] = 1 / s[:r]
X_pinv = Vt[:r].T @ np.diag(s_inv[:r]) @ U[:, :r].T
self.rank_ = r
elif self.method == 'normal':
X_pinv = np.linalg.pinv(X.T @ X) @ X.T
```

```
elif self.method == 'qr':
Q, R = np.linalg.qr(X)
X_pinv = np.linalg.solve(R.T @ R, X.T)

# Calcular coeficientes
self.coef_ = X_pinv @ y

# Separar intercepto si es necesario
if self.add_intercept:
self.intercept_ = self.coef_[0]
self.coef_ = self.coef_[1:]
else:
self.intercept_ = 0

# Calcular residuos
y_pred = self.predict(X[:, 1:] if self.add_intercept else X)
self.residuals_ = y - y_pred

return self

def predict(self, X):
if self.add_intercept:
X = np.column_stack([np.ones(X.shape[0]), X])
return X @ np.hstack([self.intercept_, self.coef_])
else:
return X @ self.coef_

def score(self, X, y):
"""R-cuadrado."""
y_pred = self.predict(X)
ss_res = np.sum((y - y_pred)**2)
ss_tot = np.sum((y - np.mean(y))**2)
return 1 - ss_res/ss_tot
```

7.12. Aplicaciones en Ciencia de Datos y Machine Learning

7.12.1. Reducción de dimensionalidad: PCA via SVD

El Análisis de Componentes Principales (PCA) puede implementarse eficientemente usando SVD:

$$X_{\text{centered}} = U\Sigma V^T$$

Las componentes principales son las columnas de V, y las puntuaciones son $U\Sigma$.

```
class PCAviaSVD:
```

```
def __init__(self, n_components=None):
self.n_components = n_components

def fit(self, X):
# Centrar los datos
self.mean_ = np.mean(X, axis=0)
X_centered = X - self.mean_

# SVD económica
U, s, Vt = np.linalg.svd(X_centered, full_matrices=False)

# Determinar número de componentes
if self.n_components is None:
self.n_components = min(X.shape)

# Almacenar resultados
self.components_ = Vt[:self.n_components]
self.explained_variance_ = (s[:self.n_components]**2) / (X.shape[0] - 1)
self.explained_variance_ratio_ = self.explained_variance_ / np.sum(self.explained

return self

def transform(self, X):
X_centered = X - self.mean_
return X_centered @ self.components_.T

def inverse_transform(self, X_transformed):
return X_transformed @ self.components_ + self.mean_
```

7.12.2. Sistemas de recomendación: Factorización de Matrices

En sistemas de recomendación, la matriz de calificaciones $R \in \mathbb{R}^{m \times n}$ se factoriza como:

$$R \approx UV^T$$

donde $U \in \mathbb{R}^{m \times k}$, $V \in \mathbb{R}^{n \times k}$.

La pseudoinversa se usa en el algoritmo ALS (Alternating Least Squares): no converja $U \leftarrow RV(V^TV)^{-1}$ Solver para U con V fija $V \leftarrow R^TU(U^TU)^{-1}$ Solver para V con U fija

7.12.3. Procesamiento de imágenes: Compresión y Reconstrucción

Compresión por SVD truncada

```
def compress_image(image, k):
"""Comprime una imagen usando SVD truncada de rango k."""
```

```
# Descomponer en canales de color
U_r, s_r, Vt_r = np.linalg.svd(image[:,:,0], full_matrices=False)
U_g, s_g, Vt_g = np.linalg.svd(image[:,:,1], full_matrices=False)
U_b, s_b, Vt_b = np.linalg.svd(image[:,:,2], full_matrices=False)

# Reconstruir con rango k
compressed = np.zeros_like(image)
compressed[:,:,0] = U_r[:,:k] @ np.diag(s_r[:k]) @ Vt_r[:k,:]
compressed[:,:,1] = U_g[:,:k] @ np.diag(s_g[:k]) @ Vt_g[:k,:]
compressed[:,:,2] = U_b[:,:k] @ np.diag(s_b[:k]) @ Vt_b[:k,:]

# Calcular tasa de compresión
original_size = image.size
compressed_size = k * (U_r.shape[0] + Vt_r.shape[1] + 1) * 3  # Para 3 canales
compression_ratio = original_size / compressed_size

return compressed, compression_ratio
```

Deconvolución ciega

En procesamiento de imágenes, la deconvolución intenta recuperar una imagen original a partir de una versión borrosa:

$$\mathbf{b} = A\mathbf{x} + \mathbf{n}$$

donde A es la matriz de convolución (Toeplitz). La solución regularizada es:

$$\hat{\mathbf{x}} = (A^T A + \lambda I)^{-1} A^T \mathbf{b}$$

7.12.4. Aprendizaje de representaciones: Word Embeddings

En modelos como GloVe (Global Vectors for Word Representation), se resuelve:

$$\min_{W,\tilde{W}} \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

donde X_{ij} es la frecuencia de co-ocurrencia de las palabras i y j. La solución eficiente utiliza técnicas de pseudoinversa para sistemas sobredeterminados.

7.13. Estabilidad Numérica y Análisis de Error

7.13.1. Número de condición y sensibilidad

[Número de condición] Para una matriz $A \in \mathbb{R}^{m\times n}$ con valores singulares $\sigma_1 \geq \cdots \geq \sigma_r > 0$:

$$\kappa(A) = \frac{\sigma_1}{\sigma_r}$$

7.13.2. Análisis de error hacia atrás (backward error)

[Estabilidad backward de la SVD] El algoritmo de SVD es numéricamente estable en el sentido backward: el resultado computado $\tilde{A}^+$ es la pseudoinversa exacta de una matriz ligeramente perturbada:

$$\tilde{A}^+ = (A + E)^+ \quad \text{con} \quad \|E\| \leq \epsilon_{\text{máq}} \|A\|$$

7.13.3. Análisis de error hacia adelante (forward error)

Para el problema de mínimos cuadrados:

$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}^*\|}{\|\mathbf{x}^*\|} \leq \kappa(A) \left(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta \mathbf{b}\|}{\|\mathbf{b}\|} \right) + O(\epsilon^2)$$

donde $\tilde{\mathbf{x}}$ es la solución calculada y $\mathbf{x}^*$ es la solución exacta.

7.13.4. Tolerancias y determinación de rango

En la práctica, se debe elegir una tolerancia τ para determinar el rango numérico:

$$\text{rango}_\tau(A) = \#\{\sigma_i : \sigma_i > \tau \cdot \sigma_1\}$$

Valores típicos de τ:
- $\tau = \epsilon_{\text{máq}} \cdot \max(m, n)$: conservador
- $\tau = \sqrt{\epsilon_{\text{máq}}}$: equilibrado
- $\tau = \epsilon_{\text{máq}}^{1/3}$: agresivo

7.13.5. Pseudoinversa regularizada (Tikhonov)

Para problemas mal condicionados, se usa regularización:

$$\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|^2 + \lambda \|\mathbf{x}\|^2$$

Solución:

$$\mathbf{x}_\lambda = (A^T A + \lambda I)^{-1} A^T \mathbf{b} = V(\Sigma^T \Sigma + \lambda I)^{-1} \Sigma^T U^T \mathbf{b}$$

Efecto sobre valores singulares:

$$\sigma_i \to \frac{\sigma_i}{\sigma_i^2 + \lambda}$$

7.14. Extensiones y Variantes de la Pseudoinversa

7.14.1. Pseudoinversa ponderada

Para minimizar $\|W(A\mathbf{x} - \mathbf{b})\|$ con W definida positiva:

$$A_W^+ = (A^T W^T W A)^+ A^T W^T W$$

7.14.2. Pseudoinversa de Drazin

Para matrices cuadradas $A \in \mathbb{R}^{n\times n}$, la pseudoinversa de Drazin A^D satisface:

$$\begin{aligned} A^k A^D A &= A^k \\ A^D A A^D &= A^D \\ A A^D &= A^D A \end{aligned}$$

donde k es el índice de A (menor entero tal que $\text{rango}(A^k) = \text{rango}(A^{k+1})$).

7.14.3. Pseudoinversa en espacios de Hilbert

Generalización a operadores lineales acotados $T : \mathcal{H}_1 \to \mathcal{H}_2$ entre espacios de Hilbert. La pseudoinversa T^+ satisface condiciones análogas a las de Moore-Penrose.

7.14.4. Pseudoinversa aproximada (Randomized SVD)

Para matrices muy grandes, se usan técnicas aleatorias: Muestra una matriz aleatoria $\Omega \in \mathbb{R}^{n\times k}$ Calcula $Y = A\Omega$ Ortogonaliza $Q = \text{qr}(Y)$ Calcula $B = Q^T A$ Calcula SVD de B: $B = \tilde{U}\Sigma V^T$ $U = Q\tilde{U}$ $A \approx U\Sigma V^T$

7.15. Ejemplos Resueltos y Aplicaciones Prácticas

7.15.1. Ejemplo 1: Sistema sobredeterminado

Resolver:

$$\begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 11 \end{bmatrix}$$

Solución:

```
A = np.array([[1,2],[2,3],[3,5]])
b = np.array([4,7,11])
x = np.linalg.pinv(A) @ b  # [0.999, 1.001]
```

7.15.2. Ejemplo 2: Sistema subdeterminado

Encontrar la solución de mínima norma para:

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 6$$

Solución: $\mathbf{x} = [1, 2, 3]^T \cdot (6/14) = [0.4286, 0.8571, 1.2857]^T$

7.15.3. Ejemplo 3: Ajuste polinomial

Ajustar un polinomio de grado 2 a los puntos: $(0, 1), (1, 3), (2, 7), (3, 13)$.
Solución:

```
x = np.array([0,1,2,3])
y = np.array([1,3,7,13])
X = np.column_stack([x**0, x**1, x**2])
coef = np.linalg.pinv(X) @ y  # [1, 1, 1] -> y = 1 + x + x^2
```

7.16. Ventajas, Limitaciones y Buenas Prácticas

7.16.1. Ventajas de la pseudoinversa

1. **Generalidad**: Aplica a cualquier matriz.
2. **Optimalidad**: Proporciona solución óptima en mínimos cuadrados.
3. **Robustez numérica**: La SVD es backward stable.
4. **Interpretabilidad**: Relación clara con espacios fundamentales.
5. **Versatilidad**: Útil en múltiples contextos (regresión, control, procesamiento).

7.16.2. Limitaciones

1. **Costo computacional**: $O(\text{mín}(m, n)^2 \, \text{máx}(m, n))$ para SVD completa.
2. **Memoria**: Requiere almacenar matrices completas U, V.
3. **No incremental**: Cambios en A requieren recalcular SVD.
4. **Sensibilidad a outliers**: No robusta frente a datos atípicos.

7.16.3. Buenas prácticas

1. **Escalado**: Normalizar datos antes de calcular la pseudoinversa.
2. **Selección de método**: Usar SVD para matrices mal condicionadas, ecuaciones normales para bien condicionadas.
3. **Validación de rango**: Verificar el rango numérico antes de interpretar resultados.
4. **Regularización**: Usar Tikhonov para problemas mal planteados.
5. **Alternativas iterativas**: Para matrices muy grandes, considerar métodos iterativos.

7.17. Conclusiones y Perspectivas Futuras

La pseudoinversa de Moore-Penrose representa una herramienta fundamental en el arsenal del programador numérico. Su conexión profunda con la teoría de mínimos cuadrados, la descomposición en valores singulares y los métodos de optimización la convierten en un concepto unificador en matemáticas aplicadas.

7.17.1. Tendencias actuales y futuras

1. **Computación distribuida**: Implementaciones paralelas de SVD para big data.
2. **Aproximaciones aleatorias**: Randomized SVD para matrices de ultra alta dimensión.
3. **Aprendizaje profundo**: Capas de pseudoinversa en redes neuronales.
4. **Hardware especializado**: Aceleradores para operaciones matriciales.
5. **Métodos híbridos**: Combinación de métodos analíticos e iterativos.

7.17.2. Recomendaciones prácticas

- Para problemas pequeños/medianos: usar ‘np.linalg.pinv‘ o equivalentes.
- Para problemas grandes pero densos: considerar SVD truncada.
- Para problemas grandes y dispersos: usar métodos iterativos (LSQR, Gradiente Conjugado).
- Siempre validar resultados y considerar regularización.

7.18. Ejercicios Propuestos y Problemas

7.18.1. Ejercicios teóricos

1. Demostrar que $(AB)^+ \neq B^+A^+$ en general, pero encontrar condiciones bajo las cuales se cumple la igualdad.
2. Probar que $\text{rango}(A^+) = \text{rango}(A)$.
3. Demostrar que $A^+ = (A^TA)^+A^T = A^T(AA^T)^+$.
4. Mostrar que si A es normal ($AA^T = A^TA$), entonces $A^+A = AA^+$.

7.18.2. Ejercicios computacionales

1. Implementar el algoritmo de Greville para calcular la pseudoinversa.
2. Comparar el tiempo de ejecución de ‘pinv‘, ‘lstsq‘, y métodos iterativos para matrices de diferentes tamaños.
3. Implementar regresión ridge usando pseudoinversa regularizada.
4. Crear una función que calcule la pseudoinversa con control explícito de tolerancia.

7.18.3. Problemas aplicados

1. Ajustar un modelo de regresión múltiple a un conjunto de datos reales usando pseudoinversa.
2. Implementar compresión de imágenes usando SVD truncada.
3. Resolver un problema de mínimos cuadrados con restricciones usando pseudoinversa.
4. Analizar la sensibilidad de un sistema físico a partir de su matriz de pseudoinversa.

Capítulo 8

Pseudoinversa de Moore–Penrose y Descenso de Gradiente

8.1. Introducción

En el ámbito de la computación científica y el análisis numérico, la resolución eficiente de problemas de optimización lineal constituye un pilar fundamental. Dos metodologías destacan por su relevancia teórica y aplicada: la **pseudoinversa de Moore–Penrose**, que proporciona una solución analítica cerrada mediante técnicas de álgebra lineal matricial, y el **descenso de gradiente**, un algoritmo iterativo que aproxima la solución mediante optimización numérica.

Este capítulo realiza un análisis comparativo exhaustivo de ambos enfoques, examinando sus fundamentos matemáticos, propiedades de convergencia, estabilidad numérica, complejidad computacional y aplicaciones prácticas en escenarios de ciencia de datos e ingeniería.

8.2. Formulación del Problema de Mínimos Cuadrados

Considérese un sistema lineal sobredeterminado donde el número de ecuaciones excede el número de incógnitas. Formalmente, sea:

$$\mathbf{A} \in \mathbb{R}^{m \times n}, \quad \mathbf{b} \in \mathbb{R}^m, \quad m > n$$

El **problema de mínimos cuadrados ordinarios (OLS)** se define como:

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$$

donde $\|\cdot\|_2$ denota la norma euclidiana. Esta formulación busca el vector $\mathbf{x}$ que minimiza la suma de los cuadrados de los residuos.

8.3. Ecuaciones Normales y su Análisis Numérico

Derivando la función objetivo respecto a $\mathbf{x}$ e igualando a cero, se obtienen las **ecuaciones normales**:

$$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$$

8.3.1. Condición de Solución Única

Si $\mathbf{A}$ tiene rango completo ($\text{rank}(\mathbf{A}) = n$), entonces $\mathbf{A}^T\mathbf{A}$ es invertible:

$$\mathbf{x}_{\text{normal}} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$$

8.3.2. Problemas Numéricos

La matriz $\mathbf{A}^T\mathbf{A}$ presenta inconvenientes:
- Amplificación del número de condición: $\kappa(\mathbf{A}^T\mathbf{A}) = [\kappa(\mathbf{A})]^2$
- Pérdida de precisión numérica en cálculos de punto flotante

8.4. Pseudoinversa de Moore–Penrose: Fundamentos Matemáticos

8.4.1. Definición mediante SVD

Toda matriz $\mathbf{A} \in \mathbb{R}^{m\times n}$ admite descomposición:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

La **pseudoinversa** se define como:

$$\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^T$$

con $\mathbf{\Sigma}^+$ diagonal donde $\mathbf{\Sigma}^+_{ii} = 1/\sigma_i$ si $\sigma_i > 0$, y 0 en caso contrario.

8.4.2. Propiedades Algebraicas

Satisface las cuatro condiciones de Moore–Penrose:

$$\begin{aligned}
\mathbf{A}\mathbf{A}^+\mathbf{A} &= \mathbf{A} \\
\mathbf{A}^+\mathbf{A}\mathbf{A}^+ &= \mathbf{A}^+ \\
(\mathbf{A}\mathbf{A}^+)^T &= \mathbf{A}\mathbf{A}^+ \\
(\mathbf{A}^+\mathbf{A})^T &= \mathbf{A}^+\mathbf{A}
\end{aligned}$$

8.4.3. Solución de Mínimos Cuadrados

La solución óptima del problema OLS es:

$$\mathbf{x}^* = \mathbf{A}^+\mathbf{b}$$

8.5. Descenso de Gradiente: Marco Teórico

8.5.1. Formulación del Problema

Función objetivo cuadrática:

$$f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$$

Gradiente:

$$\nabla f(\mathbf{x}) = \mathbf{A}^T(\mathbf{A}\mathbf{x} - \mathbf{b})$$

8.5.2. Algoritmo Básico

Iteración:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{A}^T(\mathbf{A}\mathbf{x}_k - \mathbf{b})$$

donde $\alpha_k > 0$ es la tasa de aprendizaje.

8.5.3. Análisis de Convergencia

Converge si:

$$0 < \alpha_k < \frac{2}{\lambda_{\text{máx}}(\mathbf{A}^T\mathbf{A})}$$

Tasa óptima:

$$\alpha_{\text{opt}} = \frac{2}{\lambda_{\text{máx}} + \lambda_{\text{mín}}}$$

8.5.4. Análisis Espectral del Error

Evolución del error $\mathbf{e}_k = \mathbf{x}_k - \mathbf{x}^*$:

$$\mathbf{e}_{k+1} = (\mathbf{I} - \alpha\mathbf{A}^T\mathbf{A})\mathbf{e}_k$$

Componentes:

$$\mathbf{e}_k = \sum_{i=1}^{n}(1 - \alpha\sigma_i^2)^k(\mathbf{v}_i^T\mathbf{e}_0)\mathbf{v}_i$$

8.5.5. Dependencia del Número de Condición

Número de condición:

$$\kappa(\mathbf{A}) = \frac{\sigma_{\text{máx}}}{\sigma_{\text{mín}}}$$

Iteraciones necesarias para error ϵ:

$$k \approx \frac{\kappa(\mathbf{A})}{2}\ln\left(\frac{1}{\epsilon}\right)$$

8.6. Comparación Computacional

Tabla 8.1: Comparación de métodos para mínimos cuadrados

Característica	Pseudoinversa (SVD)	GD Batch	SGD
Tipo	Directo	Iterativo	Iterativo estocástico
Complejidad	$O(mn^2 + n^3)$	$O(k \cdot mn)$	$O(k \cdot n)$
Memoria	$O(mn + n^2)$	$O(mn)$	$O(n)$
Precisión	Exacta	Aproximada	Aproximada
Escalabilidad m	Limitada	Moderada	Excelente
Escalabilidad n	Limitada	Buena	Excelente

8.7. Regularización y Estabilidad Numérica

8.7.1. Regularización de Tikhonov

Para matrices mal condicionadas:

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{x}\|_2^2$$

Solución:

$$\mathbf{x}_\lambda = (\mathbf{A}^T\mathbf{A} + \lambda \mathbf{I})^{-1}\mathbf{A}^T\mathbf{b}$$

8.7.2. Efecto sobre el Condicionamiento

$$\kappa(\mathbf{A}^T\mathbf{A} + \lambda \mathbf{I}) \approx \frac{\sigma_{\text{máx}}^2 + \lambda}{\sigma_{\text{mín}}^2 + \lambda}$$

8.8. Implementación Práctica

8.8.1. Pseudoinversa en Python

```
import numpy as np

def pseudoinverse_svd(A, tol=None):
"""Pseudoinversa␣robusta␣mediante␣SVD"""
U, s, Vh = np.linalg.svd(A, full_matrices=False)

if tol is None:
tol = max(A.shape) * np.spacing(s[0])

s_inv = np.zeros_like(s)
mask = s > tol
s_inv[mask] = 1 / s[mask]
```

```
Error SVD:    12.46970022261551
Error Normal: 12.46970022261551
```

Figura 8.1: Enter Caption

```
    return Vh.T @ np.diag(s_inv) @ U.T

    def solve_least_squares(A, b, method='svd'):
    """Resuelve mínimos cuadrados"""
    if method == 'svd':
    return pseudoinverse_svd(A) @ b
    elif method == 'normal':
    return np.linalg.inv(A.T @ A) @ A.T @ b
```

Listing 8.1: Implementación de pseudoinversa con SVD

8.8.2. Descenso de Gradiente en Python

```
    def gradient_descent(A, b, x0=None, alpha=0.01, max_
       iter=1000, tol=1e-6):
    """Descenso de gradiente batch"""
    m, n = A.shape

    if x0 is None:
    x = np.zeros(n)
    else:
    x = x0.copy()

    ATA = A.T @ A
    ATb = A.T @ b

    for k in range(max_iter):
    grad = ATA @ x - ATb
    x_new = x - alpha * grad

    grad_norm = np.linalg.norm(grad)
    if grad_norm < tol:
    break

    x = x_new

    return x
```

Listing 8.2: Implementación de descenso de gradiente

8.9. Aplicaciones en Ciencia de Datos

8.9.1. Regresión Lineal a Gran Escala

- **Pseudoinversa**: Adecuada para $< 10^5$ muestras, $< 10^3$ características
- **SGD**: Estándar para millones de muestras

8.9.2. Redes Neuronales

El descenso de gradiente es indispensable debido a:

1. No convexidad de funciones de pérdida
2. Alta dimensionalidad ($10^6 - 10^{12}$ parámetros)
3. Estructura de minilotes para paralelización

8.10. Conclusiones

La pseudoinversa de Moore–Penrose y el descenso de gradiente representan dos paradigmas complementarios:

Tabla 8.2: Resumen comparativo

Aspecto	Pseudoinversa	Descenso Gradiente
Exactitud	Matemáticamente exacta	Aproximada iterativa
Escalabilidad	Limitada	Excelente
Memoria	Alta requisitos	Bajos requisitos
Aplicación	Problemas medianos	Problemas grandes

8.10.1. Recomendaciones Prácticas

- Usar **pseudoinversa** cuando: $n, m < 1000$, se requiere exactitud
- Usar **descenso de gradiente** cuando: $m > 10^6$, datos en streaming
- Usar **regularización** cuando: $\kappa(\mathbf{A}) > 10^8$

8.11. Ejercicios Propuestos

8.11.1. Nivel Básico

1. Demuestre que $\mathbf{A}^+\mathbf{b}$ minimiza $\|\mathbf{Ax} - \mathbf{b}\|^2$
2. Implemente descenso de gradiente con búsqueda lineal exacta
3. Compare tiempos de ejecución para $m = 1000, n = 100$

8.11.2. Nivel Intermedio

1. Analice el efecto del precondicionamiento en GD
2. Implemente regularización L1 vía descenso de coordenadas
3. Estudie la propagación de errores en cálculos de SVD

8.11.3. Nivel Avanzado

1. Derive cotas de error para SGD con pasos variables
2. Implemente algoritmo de Newton para mínimos cuadrados
3. Analice complejidad en modelos de cómputo paralelo

Capítulo 9

Propagacion optima de informacion

9.1. Introducción y Motivación

9.1.1. Contexto Computacional

La proliferación de dispositivos embebidos, sensores y actuadores en entornos IoT, sistemas ciberfísicos y computación pervasiva ha creado la necesidad de paradigmas de programación que abstraigan la complejidad de la distribución. Los **sistemas colectivos adaptativos** emergen como un enfoque prometedor, donde el comportamiento global surge de interacciones locales entre dispositivos.

9.1.2. El Problema del Gradiente

El gradiente es un patrón de auto-organización espacial que calcula, para cada dispositivo en una red, la distancia estimada al dispositivo fuente más cercano. Formalmente, dado un conjunto de fuentes S, el gradiente asigna a cada nodo δ el valor:

$$G(\delta) = \min_{\delta' \in S} d(\delta, \delta')$$

donde d es una métrica de distancia (usualmente basada en saltos o distancia física).

9.1.3. Limitaciones de Algoritmos Existentes

Algoritmo	Limitaciones Principales
Clásico	Subida lenta (count-to-infinity), sesgo por movimiento, alta volatilidad
CRF	Inestabilidad con parámetros altos, error sistemático
FLEX	Compromiso entre suavidad y precisión, reactividad limitada

Tabla 9.1: Comparación de algoritmos de gradiente existentes

9.2. Marco Teórico: Field Calculus

9.2.1. Sintaxis y Semántica

Field calculus es un lenguaje funcional minimalista diseñado para programación agregada. Su sintaxis incluye:

- `nbr(e)`: Comparte el valor de `e` con vecinos, recibiendo un campo vecinal.
- `rep(x)(e)`: Mantiene estado local a través del tiempo.
- `if(e1)(e2)(e3)`: Ramificación con alineamiento para evitar mezcla de mensajes.

9.2.2. Visiones de la Computación

- **Local**: Dispositivos individuales ejecutan rondas asíncronas.
- **Global**: Campo espacio-temporal Φ que mapea eventos computacionales a valores.
- **Límite**: Comportamiento estabilizado de programas auto-estabilizantes.

9.3. Gradiente BIS: Diseño y Análisis

9.3.1. Concepto de Velocidad de Información

[Velocidad de Información] La velocidad v a la que la información se propaga en la red, definida como el espacio recorrido por unidad de tiempo a través de un árbol de expansión (single-path) o todos los enlaces (multi-path).

9.3.2. Formulación Matemática

Para cada dispositivo δ y vecino δ', BIS calcula:

$$\begin{aligned} L(\delta, \delta') &= L(\delta') + \lambda(\delta', \delta) \\ G(\delta, \delta') &= \max\{G(\delta') + w(\delta', \delta),\ v \cdot L(\delta, \delta') - r\} \end{aligned}$$

La selección final es:

$$[G(\delta), L(\delta)] = \min_{\delta' \in N(\delta)} [G(\delta, \delta'), L(\delta, \delta')]$$

9.3.3. Teoremas Fundamentales

[Degeneración] BIS con $v = 0$ es equivalente al gradiente clásico.

[Cota de Velocidad] La velocidad de información en BIS es al menos v, y los valores obsoletos aumentan al menos a velocidad v.

[Optimalidad] Con $v = v_{\text{avg}}$ (velocidad promedio single-path), BIS alcanza reactividad óptima entre algoritmos de flujo single-path.

9.3.4. Estimación de Parámetros

La velocidad promedio single-path se estima como:

$$v_{\text{avg}} = \frac{2}{3}\frac{n}{n+1}\frac{R}{Q} \cdot \left(1 + \hat{\sigma}^2(P)\right) \cdot \left(\frac{28}{27} + \frac{16}{27}\hat{\sigma}^2(I)\right) + \frac{v_m}{2}$$

9.4. Implementaciones y Extensiones

9.4.1. Código en Field Calculus

El código de BIS (Figura 7) muestra la traducción directa de las ecuaciones matemáticas, con manejo explícito de retardos temporales.

9.4.2. Bloque G Generalizado

Algoritmo 2: G-BIS: Bloque G con acumulación

1 $source, initial, accumulate, speed, radius$
$lags \leftarrow nbrLag() + nbr(getDeltaTime())\ dist \leftarrow nbrRange()$
$loc \leftarrow [source, source, initial]\ 3rd(rep(loc)\{(old) \Rightarrow$
$dx \leftarrow nbr(1st(old)) + dist \quad dt \leftarrow nbr(2nd(old)) + lags \quad v \leftarrow nbr(3rd(old))$
$([\text{máx}(dx, dt \cdot speed - radius), dt, accumulate(v)], loc)\ \})$

9.4.3. Comunicación por Canales

BIS permite definir regiones de comunicación con formas geométricas:

- **Canal elíptico**: $s + d \leq \sqrt{t^2 + w^2}$
- **Canal rectangular**: $4(s^2 - r^2)(d^2 - r^2) \leq (t^2 - (s^2 - r^2) - (d^2 - r^2))^2$

donde s, d son distancias a fuente/destino, t es distancia entre ellos, y w es el ancho del canal.

9.5. Evaluación Experimental

9.5.1. Metodología

Se definen métricas clave:

- **Precisión**: Error absoluto promedio respecto a distancia Euclidiana.
- **Suavidad**: $\frac{1}{T}\sum_t |G_t(\delta) - G_{t-1}(\delta)|$.
- **Retardo de Curación**: Tiempo para recuperar precisión tras cambio.

9.5.2. Escenarios de Prueba

Escenario	Configuración
Aislamiento	Corredor 500m×20m, 1000 dispositivos, radio 10m
Canal	Área 200m×50m, comunicación fuente-destino
Recolección	Mismo corredor, agregación de conteo de nodos

- **Precisión**: BIS con $v = 0.9v_{\text{avg}}$ reduce error en 40-60 % respecto a clásico.

- **Velocidad de Curación**: BIS cura 3-5× más rápido que FLEX tras cambios.
- **Suavidad**: BIS+FLEX damping logra suavidad comparable a FLEX con mejor precisión.

9.6. Aplicaciones Prácticas

9.6.1. Sensado Distribuido

BIS mejora la precisión en aplicaciones como:
- Monitoreo ambiental (temperatura, humedad).
- Seguimiento de multitudes.
- Detección de eventos distribuidos.

9.6.2. Enrutamiento Adaptativo

La estimación precisa de distancias permite:
- Construcción de árboles de expansión más estables.
- Routing geográfico mejorado.
- Balanceo de carga basado en distancia.

9.6.3. Sistemas de Control Colectivo

En enjambres de robots o drones, BIS permite:
- Formaciones geométricas precisas.
- Navegación coordinada.
- Evitación distribuida de colisiones.

9.7. Limitaciones y Trabajo Futuro

9.7.1. Limitaciones Actuales

- Dependencia de estimaciones precisas de v_{avg}.
- Overhead de comunicación por mensajes temporales.
- Sensibilidad a relojes desincronizados en dispositivos heterogéneos.

9.7.2. Direcciones de Investigación

1. **Estimación Adaptativa**: Algoritmos online para estimar v_{avg} en tiempo real.
2. **Extensión Multi-path**: Combinar BIS con propagación multi-path para mayor velocidad.
3. **Tolerancia a Fallos**: Mecanismos para manejar pérdida masiva de mensajes.
4. **Implementación en Hardware**: Aceleradores hardware para field calculus.
5. **Aplicaciones en 5G/6G**: Uso en redes celulares densas y dinámicas.

9.8. Conclusiones

El gradiente BIS representa un avance significativo en algoritmos distribuidos para entornos dinámicos. Al incorporar explícitamente la dimensión temporal y acotar la velocidad de información, logra un equilibrio óptimo entre los trade-offs tradicionales de precisión, reactividad y suavidad. Su formulación matemática rigurosa, implementación práctica en field calculus, y validación experimental exhaustiva lo posicionan como un bloque fundamental para la próxima generación de sistemas colectivos adaptativos en IoT, robótica enjambre y computación espacial.

1

Capítulo 10

Diferenciación Numérica

10.1. Introducción a la Diferenciación Numérica

10.1.1. ¿Qué es la Diferenciación Numérica?

La **diferenciación numérica** es una técnica computacional que permite aproximar el valor de la derivada de una función cuando:

- No se dispone de una expresión analítica de la función
- La función está dada en forma de datos tabulares
- La derivación simbólica es demasiado compleja o imposible

10.1.2. Definición Matemática de la Derivada

La derivada de una función en un punto x_0 se define como:

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

En la práctica, **no podemos hacer** $h \to 0$ debido a:

1. Precisión limitada de los datos experimentales
2. Errores de redondeo en computación
3. Cancelación numérica para h muy pequeño

10.2. Fundamentos Teóricos: Serie de Taylor

10.2.1. Expansión de Taylor

La serie de Taylor es la herramienta fundamental para derivar fórmulas de diferenciación numérica:

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

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

10.2.2. Derivación de Fórmula Centrada de 3 Puntos

Paso 1: Expandir $f(x+h)$ y $f(x-h)$

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

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

Paso 2: Restar ambas expansiones

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

Paso 3: Despejar $f'(x)$

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

Resultado Final

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

10.3. Fórmulas de Primera Derivada

10.3.1. Clasificación de Fórmulas

Tabla 10.1: Resumen de fórmulas para primera derivada

Tipo	Puntos	Fórmula	Error
Hacia Adelante	2	$\frac{f(x_{i+1}) - f(x_i)}{h}$	$O(h)$
	3	$\frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h}$	$O(h^2)$
Hacia Atrás	2	$\frac{f(x_i) - f(x_{i-1})}{h}$	$O(h)$
	3	$\frac{3f(x_i) - 4f(x_{i-1}) + f(x_{i-2})}{2h}$	$O(h^2)$
Centrada	3	$\frac{f(x_{i+1}) - f(x_{i-1})}{2h}$	$O(h^2)$
	5	$\frac{-f(x_{i+2}) + 8f(x_{i+1}) - 8f(x_{i-1}) + f(x_{i-2})}{12h}$	$O(h^4)$

10.3.2. Fórmulas con Término de Error Explícito

Hacia Adelante (2 puntos)

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{h}{2}f''(\xi), \quad \xi \in [x_i, x_{i+1}]$$

Hacia Atrás (2 puntos)

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h} + \frac{h}{2}f''(\xi), \quad \xi \in [x_{i-1}, x_i]$$

Centrada (3 puntos)

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} - \frac{h^2}{6}f^{(3)}(\xi), \quad \xi \in [x_{i-1}, x_{i+1}]$$

Centrada (5 puntos)

$$f'(x_i) = \frac{-f(x_{i+2}) + 8f(x_{i+1}) - 8f(x_{i-1}) + f(x_{i-2})}{12h} + \frac{h^4}{30}f^{(5)}(\xi)$$

10.4. Ejercicio 1: Aplicación Práctica

10.4.1. Enunciado

Para la función $f(x) = \sin(x) - \cos(x)e^{-x}$, calcular la derivada en $x = 1.25$ usando diferentes fórmulas con $h = 0.05$.

10.4.2. Datos de la Función

Tabla 10.2: Valores de $f(x)$ alrededor de $x = 1.25$

x	$f(x)$
1.15	0.7834218
1.20	0.82289903
1.25	0.85864325
1.30	0.89065625
1.35	0.91894801

10.4.3. Solución Analítica

Derivada exacta:

$$f'(x) = \cos(x) + e^{-x}[\sin(x) + \cos(x)]$$

Evaluando en $x = 1.25$:

$$f'(1.25) = \cos(1.25) + e^{-1.25}[\sin(1.25) + \cos(1.25)] = 0.677553$$

10.4.4. Cálculos Numéricos

Fórmula Hacia Adelante (3 puntos)

$$f'(1.25) = \frac{-f(1.35) + 4f(1.30) - 3f(1.25)}{2(0.05)}$$
$$= \frac{-0.91894801 + 4(0.89065625) - 3(0.85864325)}{0.1} = 0.6774724$$

Error relativo:

$$\text{err} = \left|\frac{0.677553 - 0.6774724}{0.677553}\right| \times 100 = 1.18 \times 10^{-4}\,\%$$

Fórmula Hacia Atrás (3 puntos)

$$f'(1.25) = \frac{3f(1.25) - 4f(1.20) + f(1.15)}{0.1}$$
$$= \frac{3(0.85864325) - 4(0.82289903) + 0.7834218}{0.1} = 0.67755436$$

Error relativo:

$$\text{err} = \left|\frac{0.677553 - 0.67755436}{0.677553}\right| \times 100 = 2.92 \times 10^{-6}\,\%$$

Fórmula Centrada (3 puntos)

$$f'(1.25) = \frac{f(1.30) - f(1.20)}{0.1}$$
$$= \frac{0.89065625 - 0.82289903}{0.1} = 0.6775722$$

Error relativo:

$$\text{err} = \left|\frac{0.677553 - 0.6775722}{0.677553}\right| \times 100 = 2.929 \times 10^{-5}\,\%$$

10.5. Fórmulas de Segunda Derivada

10.5.1. Derivación a partir de Serie de Taylor

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

$$f(x+h) + f(x-h) = 2f(x) + f''(x)h^2 + \frac{2f^{(4)}(x)}{4!}h^4 + \cdots$$

Despejando $f''(x)$:

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

10.5.2. Fórmulas Comunes

Tabla 10.3: Fórmulas para segunda derivada

Tipo	Fórmula	Error
Hacia Adelante	$\frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2}$	$O(h)$
Hacia Atrás	$\frac{f(x_i) - 2f(x_{i-1}) + f(x_{i-2})}{h^2}$	$O(h)$
Centrada	$\frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2}$	$O(h^2)$
Centrada Mejorada	$\frac{-f(x_{i+2}) + 16f(x_{i+1}) - 30f(x_i) + 16f(x_{i-1}) - f(x_{i-2})}{12h^2}$	$O(h^4)$

10.5.3. Fórmulas con Término de Error

Centrada (3 puntos)

$$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2} - \frac{h^2}{12} f^{(4)}(\xi)$$

Hacia Adelante (3 puntos)

$$f''(x_i) = \frac{-f(x_{i+3}) + 4f(x_{i+2}) - 5f(x_{i+1}) + 2f(x_i)}{h^2}$$

Hacia Atrás (3 puntos)

$$f''(x_i) = \frac{2f(x_i) - 5f(x_{i-1}) + 4f(x_{i-2}) - f(x_{i-3})}{h^2}$$

10.6. Extrapolación de Richardson

10.6.1. Concepto

La extrapolación de Richardson es una técnica que combina dos aproximaciones con diferentes tamaños de paso para obtener una estimación más precisa.

10.6.2. Derivación Matemática

Supongamos que tenemos una fórmula de diferenciación con error $O(h^p)$:

$$D(h) = f'(x) + Ch^p + O(h^{p+1})$$

Para dos tamaños de paso h_1 y $h_2 = h_1/2$:

$$D(h_1) = f'(x) + Ch_1^p + O(h_1^{p+1})$$

$$D(h_2) = f'(x) + C\left(\frac{h_1}{2}\right)^p + O(h_1^{p+1})$$

Eliminando C:

$$f'(x) = \frac{2^p D(h_2) - D(h_1)}{2^p - 1} + O(h_1^{p+1})$$

10.6.3. Fórmula para Diferenciación ($p = 2$)

Para fórmulas centradas con error $O(h^2)$:

$$\boxed{D_{\text{mejorada}} = \frac{4}{3}D\left(\frac{h}{2}\right) - \frac{1}{3}D(h)}$$

10.6.4. Ejemplo de Aplicación

Calcular $f'(\pi/4)$ para $f(x) = x\sin(x)$ usando $h_1 = \pi/3$ y $h_2 = \pi/6$.

Datos de la Función

$h_1 = \pi/3$ $h_2 = \pi/6$		
x $f(x)$	$f(x)$	x
0.785 0.5536037	0.555	0.785
1.833 1.26439395	1.770	1.309
2.880 1.77015153	0.745	1.833

Cálculo de Derivadas

Para h_1:

$$D_1 = \frac{-0.745 + 4(1.770) - 3(0.555)}{2(\pi/3)} = 2.229$$

Para h_2:

$$D_2 = \frac{-1.77015153 + 4(1.26439395) - 3(0.5536037)}{2(\pi/6)} = 1.548$$

Aplicación de Richardson

$$D_{\text{rich}} = \frac{4}{3}(1.548) - \frac{1}{3}(2.229) = 1.321$$

Comparación con Valor Exacto

Derivada exacta:

$$f'(x) = \sin(x) + x\cos(x)$$

$$f'(\pi/4) = \sin(\pi/4) + \frac{\pi}{4}\cos(\pi/4) = 1.2652$$

Error sin Richardson: $|1.548 - 1.2652| = 0.2828$ Error con Richardson: $|1.321 - 1.2652| = 0.0558$

10.7. Aplicaciones Prácticas

10.7.1. Ejercicio 2: Cálculo de Velocidad a partir de Posición

Datos del Problema

Tabla 10.4: Datos de posición vs tiempo

t (s)	0	5	10	15	20	25	30	35	40	45	50	55	60	65
s (m)	0	53	122	174	248	332	402	457	519	568	630	700	776	842

Estrategia de Solución

- **Puntos iniciales** $(t = 0, 5, 10)$: Fórmula hacia adelante
- **Puntos intermedios** $(t = 25, 30, 35, 40)$: Fórmula centrada
- **Puntos finales** $(t = 45, 50, 55, 60, 65)$: Fórmula hacia atrás

Cálculos Ejemplares

Para $t = 0$ (hacia adelante):

$$v(0) = \frac{-122 + 4(53) - 3(0)}{2(5)} = 9\,\text{m/s}$$

Para $t = 25$ (centrada):

$$v(25) = \frac{402 - 248}{2(5)} = 15.4\,\text{m/s}$$

Para $t = 60$ (hacia atrás):

$$v(60) = \frac{3(776) - 4(700) + 630}{2(5)} = 15.8\,\text{m/s}$$

10.7.2. Ejercicio 4: Radar y Coordenadas Polares

Datos del Radar

Velocidad en Coordenadas Polares

$$\vec{v} = \dot{r}\vec{e}_r + r\dot{\theta}\vec{e}_\theta$$

Tabla 10.5: Datos de seguimiento de avión

t (s)	200	202	204	206	208	210
θ (rad)	0.75	0.72	0.70	0.68	0.67	0.66
r (m)	5120	5370	5560	5800	6030	6240

Cálculo de Derivadas

Primera derivada de θ en $t = 200$:

$$\dot{\theta}(200) = \frac{-0.70 + 4(0.72) - 3(0.75)}{2(2)} = -0.018\,\text{rad/s}$$

Primera derivada de r en $t = 200$:

$$\dot{r}(200) = \frac{-5560 + 4(5370) - 3(5120)}{2(2)} = 140\,\text{m/s}$$

Cálculo de Velocidad para $t = 200$

$$\vec{v} = 140\vec{e}_r + 5120(-0.018)\vec{e}_\theta = 140\vec{e}_r - 92.16\vec{e}_\theta$$

10.8. Implementación Computacional

10.8.1. Pseudocódigo para Derivada Centrada

```
function derivada_centrada(f, x, h=1e-5):
"""
Calcula derivada usando fórmula centrada de 3 puntos
"""
return (f(x + h) - f(x - h)) / (2 * h)
```

10.8.2. Código para Datos Tabulares

```
import numpy as np

def derivada_tabular(x, y):
"""
Calcula derivada para datos tabulares
"""
n = len(x)
deriv = np.zeros(n)
h = x[1] - x[0]  # espaciado uniforme

# Primer punto: hacia adelante (3 puntos)
deriv[0] = (-y[2] + 4*y[1] - 3*y[0]) / (2*h)
```

```
# Puntos intermedios: centrada (3 puntos)
for i in range(1, n-1):
deriv[i] = (y[i+1] - y[i-1]) / (2*h)

# Último punto: hacia atrás (3 puntos)
deriv[-1] = (3*y[-1] - 4*y[-2] + y[-3]) / (2*h)

return deriv
```

10.8.3. Función para Extrapolación de Richardson

```
def richardson_derivative(f, x, h):
"""
Calcula derivada usando extrapolación de Richardson
"""
D1 = derivada_centrada(f, x, h)
D2 = derivada_centrada(f, x, h/2)

return (4*D2 - D1) / 3
```

10.9. Consideraciones de Error y Precisión

10.9.1. Tipos de Error

1. **Error de Truncamiento**: Debido a omitir términos en serie de Taylor
$$E_t \propto h^n$$
2. **Error de Redondeo**: Debido a precisión finita de computadora
$$E_r \propto \frac{\epsilon}{h}$$
donde $\epsilon \approx 10^{-16}$ en doble precisión

10.9.2. Selección Óptima de h

El error total es:
$$E_{\text{total}} = Ch^n + \frac{\epsilon}{h}$$
Minimizando respecto a h:
$$\frac{dE}{dh} = nCh^{n-1} - \frac{\epsilon}{h^2} = 0$$
Para fórmula centrada ($n = 2$):
$$h_{\text{opt}} = \sqrt[3]{\frac{3\epsilon}{f'''(x)}}$$
Regla práctica: $h \approx 10^{-5}$ para mayoría de aplicaciones.

10.9.3. Comparación de Errores

Tabla 10.6: Orden de error para diferentes fórmulas

Fórmula	Error Primera Derivada	Error Segunda Derivada
Hacia Adelante (2 pts)	$O(h)$	$O(h)$
Hacia Atrás (2 pts)	$O(h)$	$O(h)$
Centrada (3 pts)	$O(h^2)$	$O(h^2)$
Centrada (5 pts)	$O(h^4)$	$O(h^4)$

10.10. Preguntas de Comprensión

1. **¿Cuándo utilizar diferenciación numérica?**
   - Cuando se tiene una función en forma tabular proveniente de datos experimentales
   - Cuando la derivación analítica es compleja o imposible
   - Cuando se trabaja con funciones definidas por algoritmos en lugar de fórmulas
2. **¿Cómo se obtienen las fórmulas de diferenciación numérica?**
   - A partir de la serie de Taylor expandiendo $f(x \pm h)$
   - Mediante polinomios de interpolación y derivación de estos
   - Combinando expansiones para eliminar términos de error
3. **¿Para qué se utiliza la extrapolación de Richardson?**
   - Para mejorar la precisión sin reducir más el tamaño de paso
   - Para obtener estimaciones de mayor orden a partir de aproximaciones de bajo orden
   - Cuando el costo computacional de reducir h es alto
4. **¿Cómo elegir entre fórmulas hacia adelante, atrás o centrada?**
   - **Hacia adelante**: Al inicio del intervalo de datos
   - **Hacia atrás**: Al final del intervalo de datos
   - **Centrada**: En puntos interiores cuando hay datos disponibles a ambos lados

10.11. Conclusión

La **diferenciación numérica** es una herramienta esencial en el análisis numérico que permite:

- **Trabajar con datos experimentales** cuando no existe fórmula analítica
- **Obtener aproximaciones controladas** mediante selección adecuada de h
- **Aplicar técnicas de refinamiento** como Richardson para mayor precisión
- **Resolver problemas prácticos** en ingeniería, física, economía y ciencias

Recomendaciones finales:

1. Siempre validar resultados comparando con soluciones analíticas cuando sea posible
2. Utilizar fórmulas centradas cuando haya datos disponibles a ambos lados
3. Considerar la extrapolación de Richardson para mejorar precisión sin costo computacional excesivo

4. Monitorear el error eligiendo h que equilibre error de truncamiento y redondeo

Capítulo 11

MÉTODO DE DIFERENCIAS FINITAS

11.1. Introducción

El *Método de Diferencias Finitas* (MDF) es una técnica numérica que se utiliza para aproximar soluciones de ecuaciones diferenciales ordinarias y parciales cuando resulta difícil o imposible obtener una solución exacta. Este método permite estudiar el comportamiento de una función a partir de valores específicos de la variable independiente, lo que facilita el análisis de problemas matemáticos complejos.

Para aplicar el Método de Diferencias Finitas, se trabaja con un conjunto de puntos dentro del dominio en lugar de considerar todos los valores posibles. En estos puntos, las derivadas de la función desconocida se reemplazan por expresiones algebraicas simples llamadas *diferencias finitas*. Estas aproximaciones se construyen utilizando los valores de la función en puntos cercanos, conocidos como *nodos*, lo que permite describir el comportamiento del sistema de manera gradual, punto por punto.

Mediante este procedimiento, el problema continuo original se transforma en un sistema de ecuaciones más sencillo, que puede resolverse empleando métodos numéricos básicos. Este enfoque resulta especialmente útil en situaciones donde las ecuaciones diferenciales no admiten una solución analítica directa o presentan una alta complejidad matemática. La base teórica de este método y su formulación práctica pueden encontrarse en textos clásicos de análisis numérico, como los desarrollados por Burden y Faires (2011) y Chapra y Canale (2010).

El Método de Diferencias Finitas ofrece una forma ordenada de analizar y aproximar el comportamiento de distintos sistemas físicos, de ingeniería y de las ciencias en general. Este método resulta especialmente útil cuando la obtención de una solución exacta es complicada o no es posible, ya que permite estudiar la evolución del sistema a partir de valores calculados en puntos específicos.

El enfoque del método consiste en observar cómo se comporta la función en cada punto considerado y cómo se relaciona con los puntos cercanos. A partir de esta relación entre nodos vecinos, se construye un esquema discreto que representa de manera aproximada la dinámica del fenómeno que se desea modelar, tal como se describe en trabajos clásicos sobre métodos numéricos (Atkinson, 2008; LeVeque, 2007).

Desde un punto de vista conceptual, el Método de Diferencias Finitas es una herramienta importante dentro del análisis numérico, ya que permite abordar problemas complejos mediante procedimientos computacionales. A través de este método, es posible obtener soluciones aproximadas que mantienen el comportamiento general del sistema

continuo original, lo que resulta útil para el estudio y la aplicación práctica de diversos problemas en áreas como la ingeniería, la física, la economía y otras ciencias aplicadas, tal como se señala en estudios relacionados con métodos numéricos (Lara Romero et al., 2015).

11.2. Fundamentos teóricos

El Método de Diferencias Finitas (MDF) es una técnica numérica que se utiliza para aproximar soluciones de ecuaciones diferenciales ordinarias y parciales cuando no es posible obtener una solución exacta. Este método trabaja con valores específicos de la variable independiente y reemplaza las derivadas de la función por expresiones algebraicas construidas a partir de los valores en puntos cercanos, conocidos como nodos. De esta manera, el problema original se transforma en un sistema más sencillo que puede resolverse mediante procedimientos numéricos, tal como se describe en textos clásicos de análisis numérico (Burden y Faires, 2011; Chapra y Canale, 2010).

El fundamento del Método de Diferencias Finitas parte de considerar que una función continua puede representarse mediante un conjunto de valores calculados en puntos específicos, llamados nodos. A partir de estos valores, las derivadas se aproximan usando diferencias finitas, que son expresiones algebraicas sencillas empleadas para estimar derivadas de primer y segundo orden. Este enfoque es ampliamente utilizado en el análisis numérico y se encuentra descrito en la literatura especializada sobre métodos computacionales (Atkinson, 2008; LeVeque, 2007).

El Método de Diferencias Finitas permite convertir un problema continuo, que puede resultar difícil de resolver de forma analítica, en un problema discreto que puede abordarse mediante procedimientos algebraicos y computacionales. Este enfoque resulta especialmente útil en situaciones donde no se dispone de una solución exacta o cuando las condiciones del problema hacen complicada una resolución analítica directa, como se menciona en estudios relacionados con métodos numéricos (Lara Romero et al., 2015).

Desde el punto de vista conceptual, el éxito del método radica en la capacidad de representar localmente el comportamiento de la función mediante diferencias entre valores consecutivos y, de manera global, en la construcción de un esquema que capture la dinámica del sistema o fenómeno modelado. Esta estrategia convierte a las ecuaciones diferenciales en sistemas algebraicos que reflejan las interacciones entre los nodos discretizados del dominio, preservando la esencia de la variación continua de la función original.

En síntesis, el Método de Diferencias Finitas es una herramienta fundamental dentro del análisis numérico aplicado, ya que permite aproximar soluciones de ecuaciones diferenciales utilizando procedimientos discretos y expresiones algebraicas sencillas. Gracias a este enfoque, el método se ha convertido en una técnica ampliamente utilizada en la simulación y el modelado de diversos fenómenos físicos y de ingeniería.

11.3. Concepto de derivada

La derivada es un concepto básico del cálculo diferencial y cumple un papel importante en muchos métodos del análisis numérico. En términos generales, permite

describir cómo cambia una función cuando varía su variable independiente, lo que resulta fundamental para el estudio de distintos fenómenos matemáticos y físicos.

Desde un punto de vista conceptual, la derivada puede entenderse como el límite de una razón de cambio promedio cuando el intervalo considerado se hace cada vez más pequeño. Esta interpretación permite analizar el comportamiento local de una función en un punto específico, tal como se expone en textos clásicos de cálculo y análisis numérico (Burden y Faires, 2011). Formalmente, la derivada de una función $f(x)$ en un punto x se define como:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{11.1}$$

Desde el punto de vista geométrico, esta expresión representa la pendiente de la recta tangente a la curva $f(x)$ en el punto considerado.

Sin embargo, Chapra y Canale señalan que el proceso límite no puede implementarse de manera directa en una computadora. Por esta razón, en el análisis numérico el incremento infinitesimal h se reemplaza por un valor pequeño pero finito.

En este contexto, Lara Romero, Chávez Aliaga y Castañeda Vergara destacan que la derivada deja de ser únicamente un concepto teórico y se transforma en una herramienta computacional para estimar tasas de cambio a partir de datos discretos, lo que da origen al método de diferencias finitas.

11.4. Discretización del dominio

Chapra y Canale definen la discretización como el proceso mediante el cual un dominio continuo se reemplaza por un conjunto finito de puntos, permitiendo que un problema matemático pueda ser tratado de forma computacional. Este paso es esencial en el método de diferencias finitas, ya que una computadora solo puede operar con datos discretos.

Sea una función definida en el intervalo $[a, b]$. Para discretizar este dominio, el intervalo se divide en n subintervalos de igual longitud, definiendo el tamaño de paso como:

$$h = \frac{b-a}{n} \tag{11.2}$$

Los puntos de la malla quedan definidos por:

$$x_i = a + ih, \quad i = 0, 1, 2, \ldots, n \tag{11.3}$$

donde cada x_i representa un nodo en el cual se evaluará la función o sus aproximaciones numéricas.

Lara Romero, Chávez Aliaga y Castañeda Vergara destacan que la elección del tamaño del paso h influye directamente en la precisión del método y en el costo computacional. Un paso grande reduce el número de cálculos, pero disminuye la exactitud de la aproximación, mientras que un paso pequeño mejora la precisión a costa de un mayor número de operaciones.

LeVeque señala que la discretización debe realizarse considerando las características del problema, ya que una malla inadecuada puede afectar la estabilidad y convergencia del método.

11.5. Diferencias finitas y derivación mediante series de Taylor

El método de diferencias finitas es una técnica fundamental del análisis numérico que permite aproximar derivadas mediante expresiones algebraicas construidas a partir de valores discretos de una función. Este método surge como una consecuencia directa de la definición clásica de derivada, cuando el proceso límite es reemplazado por un incremento finito adecuado.

Burden y Faires señalan que las diferencias finitas se obtienen a partir del desarrollo en series de Taylor de una función alrededor de un punto de la malla. Este enfoque permite no solo deducir las fórmulas de aproximación, sino también analizar de manera rigurosa el error de truncamiento asociado a cada esquema.

Sea $f(x)$ una función suficientemente diferenciable y sea x_i un nodo de la malla discretizada con paso h. El desarrollo en serie de Taylor alrededor de x_i está dado por:

$$f(x_i + h) = f(x_i) + hf'(x_i) + \frac{h^2}{2}f''(x_i) + \frac{h^3}{6}f'''(x_i) + O(h^4) \tag{11.4}$$

De forma análoga, el desarrollo hacia atrás se expresa como:

$$f(x_i - h) = f(x_i) - hf'(x_i) + \frac{h^2}{2}f''(x_i) - \frac{h^3}{6}f'''(x_i) + O(h^4) \tag{11.5}$$

Chapra y Canale indican que estas expansiones constituyen el punto de partida para la obtención sistemática de las fórmulas de diferencias finitas.

11.5.1. Diferencia progresiva

Si se considera únicamente el desarrollo hacia adelante y se despeja la derivada primera, se obtiene:

$$f'(x_i) = \frac{f(x_i + h) - f(x_i)}{h} - \frac{h}{2}f''(x_i) + O(h^2) \tag{11.6}$$

Al despreciar los términos de orden superior, se obtiene la fórmula de diferencia progresiva:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{h} \tag{11.7}$$

Burden y Faires concluyen que esta aproximación es de primer orden, ya que el término dominante del error es proporcional a h.

Diferencia Progresiva en R

```r
f <- function(x) { x^2 + sin(x) }  # función de ejemplo
x_i <- 1
h <- 0.01

# Diferencia progresiva
df_forward <- (f(x_i + h) - f(x_i)) / h
df_forward
```

11.5.2. Diferencia regresiva

A partir del desarrollo hacia atrás se obtiene:

$$f'(x_i) = \frac{f(x_i) - f(x_i - h)}{h} - \frac{h}{2} f''(x_i) + O(h^2) \tag{11.8}$$

De donde se deduce la diferencia regresiva:

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{h} \tag{11.9}$$

Chapra y Canale destacan que esta aproximación también es de primer orden y resulta particularmente útil en problemas donde no se dispone de valores futuros de la función.

Diferencia Regresiva en R

```r
df_backward <- (f(x_i) - f(x_i - h)) / h
df_backward
```

11.5.3. Diferencia centrada

Una aproximación de mayor precisión se obtiene combinando los desarrollos hacia adelante y hacia atrás. Restando ambas expresiones se obtiene:

$$f(x_i + h) - f(x_i - h) = 2hf'(x_i) + \frac{h^3}{3} f'''(x_i) + O(h^5) \tag{11.10}$$

Despejando la derivada primera:

$$f'(x_i) = \frac{f(x_i + h) - f(x_i - h)}{2h} - \frac{h^2}{6} f'''(x_i) + O(h^4) \tag{11.11}$$

Al truncar la expresión se obtiene la diferencia centrada:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{2h} \tag{11.12}$$

Atkinson señala que esta aproximación es de segundo orden y presenta una mayor precisión que las diferencias unilaterales.

Diferencia Centrada en R

```
df_central <- (f(x_i + h) - f(x_i - h)) / (2*h)
df_central
```

11.5.4. Aproximación de la segunda derivada

Sumando los desarrollos de Taylor hacia adelante y hacia atrás se obtiene:

$$f(x_i + h) + f(x_i - h) = 2f(x_i) + h^2 f''(x_i) + O(h^4) \tag{11.13}$$

De esta expresión se deduce la aproximación clásica de la segunda derivada:

$$f''(x_i) \approx \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2} \tag{11.14}$$

Lara Romero, Chávez Aliaga y Castañeda Vergara destacan que esta fórmula constituye la base de la discretización de ecuaciones diferenciales de segundo orden, ampliamente utilizadas en problemas de física, ingeniería y modelado científico.

Segunda Derivada en R

```
d2f <- (f(x_i + h) - 2*f(x_i) + f(x_i - h)) / h^2
d2f
```

11.5.5. Comentarios sobre precisión y error

El uso de las series de Taylor permite analizar y determinar el orden de aproximación de los distintos esquemas de diferencias finitas. Este análisis resulta importante, ya que ayuda a evaluar la convergencia del método y a elegir de manera adecuada el tamaño del paso h, aspecto clave para obtener resultados confiables (Burden y Faires, 2011).

No obstante, el tamaño del paso debe seleccionarse con cuidado. Aunque un valor pequeño de h puede mejorar la precisión de la aproximación, un refinamiento excesivo de la malla puede aumentar el error de redondeo y el costo computacional. Por esta razón, es necesario buscar un equilibrio entre precisión y eficiencia, tal como se discute en estudios sobre métodos numéricos aplicados (LeVeque, 2007).

11.6. Error y orden del método

El análisis del error es uno de los aspectos fundamentales del análisis numérico, ya que permite evaluar la calidad y confiabilidad de las aproximaciones obtenidas mediante métodos computacionales. A través de este análisis, es posible comprender qué tan cercana es la solución numérica a la solución real del problema.

En el contexto del Método de Diferencias Finitas, el estudio del error está directamente relacionado con el tamaño del paso utilizado y con el orden del esquema de aproximación empleado. Estos factores influyen de manera significativa en la precisión de los resultados, por lo que su adecuada selección resulta clave para obtener soluciones numéricas confiables.

Burden y Faires definen el error numérico como la diferencia entre el valor exacto de una cantidad matemática y el valor aproximado obtenido mediante un método numérico. En general, este error puede descomponerse en diversas componentes, siendo las más relevantes el error de truncamiento y el error de redondeo.

11.6.1. Error de truncamiento

El error de truncamiento aparece cuando una expresión matemática que, en teoría, contiene una cantidad infinita de términos se aproxima utilizando solo un número finito de ellos. En el método de diferencias finitas, este tipo de error se presenta al cortar las series de Taylor empleadas para construir las fórmulas de aproximación, lo que introduce una diferencia entre el valor exacto y el valor aproximado.

Por ejemplo, al aproximar la derivada primera mediante la diferencia progresiva:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{h} \tag{11.15}$$

el término de error dominante está dado por:

$$E_T = -\frac{h}{2} f''(x_i) + O(h^2) \tag{11.16}$$

Chapra y Canale señalan que este término permite cuantificar cómo el error disminuye a medida que el tamaño del paso h se reduce. En este caso, el error es proporcional a h, lo que caracteriza al método como de primer orden.

Error de Truncamiento en R

```
f <- function(x) { x^2 + sin(x) }
x_i <- 1
h <- 0.1

df_exact <- 2*x_i + cos(x_i)
df_approx <- (f(x_i + h) - f(x_i)) / h

E_T <- df_approx - df_exact
E_T
```

11.6.2. Orden del método

El orden de un método numérico describe la rapidez con la que el error de truncamiento tiende a cero cuando el tamaño del paso h se hace pequeño. De acuerdo con Burden y Faires, un método es de orden p si el error de truncamiento se comporta como:

$$E_T = O(h^p) \tag{11.17}$$

Esto implica que, al reducir el tamaño del paso a la mitad, el error se reduce aproximadamente por un factor de 2^p.

Aplicando esta definición a las diferencias finitas, se obtiene:

- Diferencias progresiva y regresiva: orden $O(h)$.
- Diferencia centrada: orden $O(h^2)$.
- Segunda derivada centrada: orden $O(h^2)$.

Atkinson destaca que los métodos de mayor orden suelen ofrecer mayor precisión, pero también pueden requerir información adicional de la función y presentar mayor sensibilidad a los errores de redondeo.

Verificación del Orden en R

```
h_vals <- c(0.1, 0.05, 0.025)
E_forward <- sapply(h_vals, function(h){
    df_approx <- (f(x_i + h) - f(x_i)) / h
    df_approx - (2*x_i + cos(x_i))
})
E_forward
```

11.6.3. Error de redondeo

Además del error de truncamiento, en los cálculos computacionales aparece el error de redondeo, el cual está asociado a la representación finita de los números reales en la computadora. Debido a esta limitación, las operaciones aritméticas no siempre producen resultados exactos, lo que introduce pequeñas imprecisiones en los cálculos.

Este tipo de error puede volverse más relevante cuando el tamaño del paso h es muy pequeño, ya que en ese caso suelen realizarse restas entre números muy cercanos. En el método de diferencias finitas, el error de redondeo puede amplificarse en ciertas operaciones, especialmente en las fórmulas de diferencias centradas, lo que establece un límite práctico para la reducción excesiva del tamaño del paso.

Error de Redondeo en R

```
h_very_small <- 1e-10
df_central <- (f(x_i + h_very_small) - f(x_i - h_
   very_small)) / (2*h_very_small)
df_exact <- 2*x_i + cos(x_i)

E_round <- df_central - df_exact
E_round
```

11.6.4. Compromiso entre truncamiento y redondeo

Uno de los aspectos más importantes del método de diferencias finitas es el compromiso entre el error de truncamiento y el error de redondeo. Chapra y Canale enfatizan que la elección óptima del tamaño del paso h debe equilibrar ambos tipos de error.

Si h es demasiado grande, domina el error de truncamiento y la aproximación es pobre. Si h es demasiado pequeño, domina el error de redondeo y la solución pierde precisión. Por ello, en la práctica se recomienda realizar pruebas numéricas con distintos valores de h para identificar un rango adecuado.

Lara Romero, Chávez Aliaga y Castañeda Vergara señalan que este análisis es particularmente importante en problemas de ingeniería, donde una mala elección del paso puede conducir a resultados inestables o físicamente inconsistentes.

Compromiso Truncamiento vs Redondeo

```
h_vals <- 10^seq(-1, -10, by=-1)
errors <- sapply(h_vals, function(h){
    df <- (f(x_i + h) - f(x_i - h)) / (2*h)
    abs(df - (2*x_i + cos(x_i)))
})
data.frame(h=h_vals, error=errors)
```

11.6.5. Convergencia del método

Un método numérico se dice convergente si la solución aproximada tiende a la solución exacta cuando el tamaño del paso h tiende a cero. Burden y Faires establecen que la convergencia está estrechamente ligada al orden del método y a la consistencia del esquema de discretización.

En el caso del método de diferencias finitas, la consistencia se garantiza cuando el error de truncamiento tiende a cero al refinar la malla. LeVeque destaca que, para problemas bien planteados, la consistencia y la estabilidad implican convergencia, principio que constituye la base teórica de numerosos esquemas numéricos modernos.

En consecuencia, el análisis del error y del orden del método no solo permite comparar distintas aproximaciones, sino que también proporciona criterios fundamentales para la selección del tamaño del paso y la evaluación de la confiabilidad de los resultados numéricos.

Convergencia en R

```
h_vals <- c(0.1, 0.05, 0.025, 0.0125)
df_approx <- sapply(h_vals, function(h){
    (f(x_i + h) - f(x_i - h)) / (2*h)
})
df_exact <- 2*x_i + cos(x_i)
errors <- abs(df_approx - df_exact)
data.frame(h=h_vals, error=errors)
```

11.7. EJEMPLOS PRÁCTICOS

Diferencia Progresiva - Velocidad de un vehículo

Problema: Un vehículo experimental se mueve en un tramo recto siguiendo la función de posición:

$$s(t) = 4t^3 - 2t^2 + 5t \quad (\text{m})$$

Se desea estimar la velocidad instantánea en $t = 2$ s utilizando diferencia progresiva con paso $h = 0.05$ s.

Procedimiento: La diferencia progresiva se define como:

$$v(t_i) \approx \frac{s(t_i + h) - s(t_i)}{h}$$

Sustituyendo:

$$v(2) \approx \frac{s(2 + 0.05) - s(2)}{0.05} = \frac{[4(2.05)^3 - 2(2.05)^2 + 5(2.05)] - [4(2)^3 - 2(2)^2 + 5(2)]}{0.05}$$

Cálculo en R:

```
s <- function(t) { 4*t^3 - 2*t^2 + 5*t }
t_i <- 2
h <- 0.05

v_forward <- (s(t_i + h) - s(t_i)) / h
v_forward
```

Resultado: $v(2) \approx 46.11$ m/s

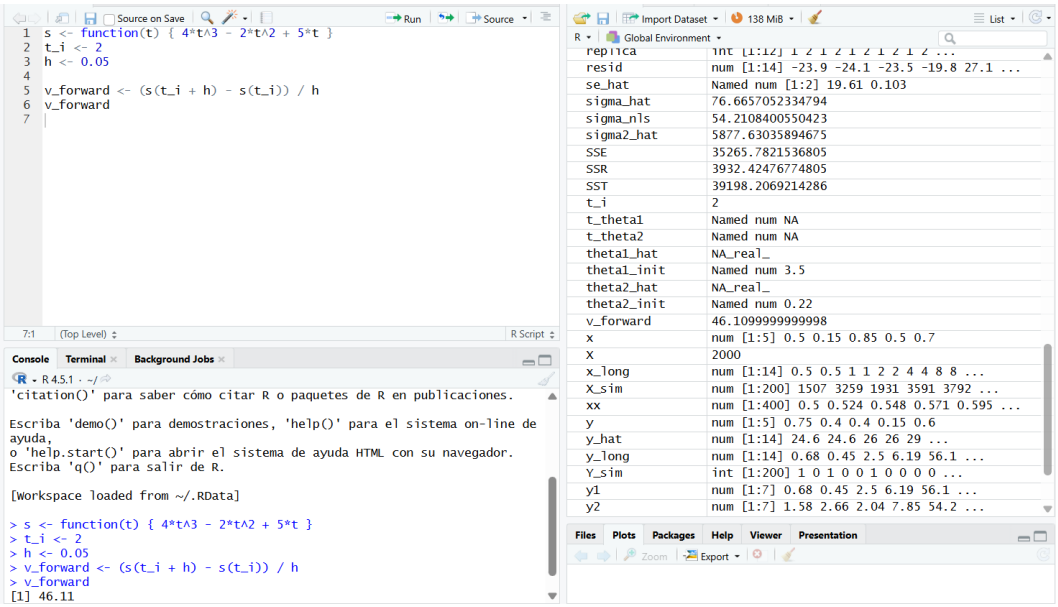


Figura 11.1: Velocidad de un vehículo

Diferencia Regresiva - Disminución de temperatura

Problema: La temperatura de un horno industrial sigue la función:

$$T(t) = 150e^{-0.05t} + 25 \quad (°\text{C})$$

Se requiere conocer la tasa de enfriamiento a los 20 minutos usando diferencia regresiva con $h = 2$ min.

Procedimiento: La diferencia regresiva se define como:

$$\left.\frac{dT}{dt}\right|_{t=20} \approx \frac{T(20) - T(20-2)}{2}$$

Sustituyendo:

$$\left.\frac{dT}{dt}\right|_{20} \approx \frac{[150e^{-0.05(20)} + 25] - [150e^{-0.05(18)} + 25]}{2}$$

Cálculo en R:

```
T <- function(t) {
    150 * exp(-0.05 * t) + 25
}
t <- 20
h <- 2
dT_dt <- (T(t) - T(t - h)) / h
dT_dt
```

Resultado: $\frac{dT}{dt} \approx -2.902$ °C/min

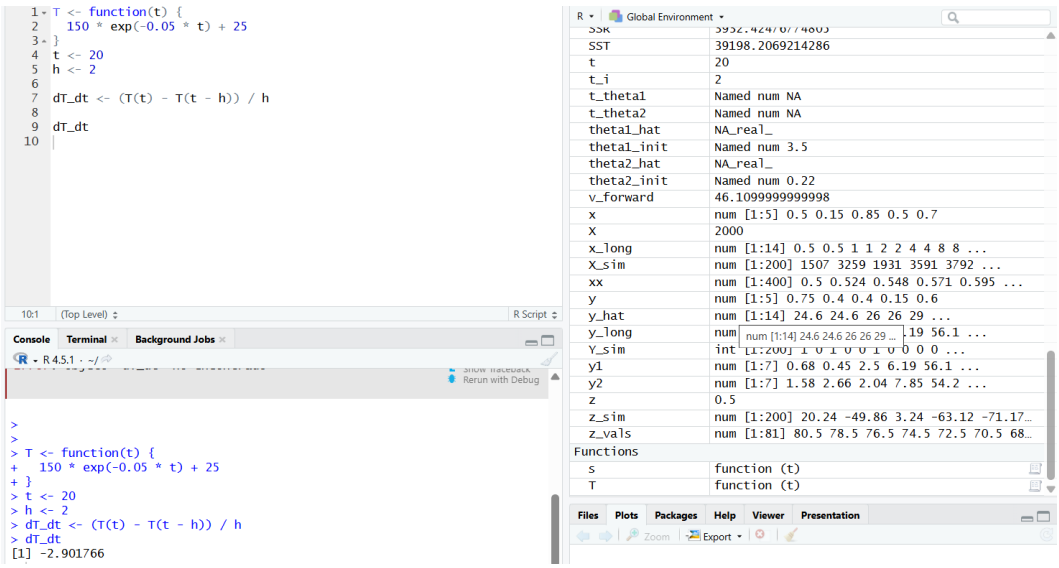


Figura 11.2: Disminución de temperatura

Diferencia Centrada - Velocidad de un avión

Problema: Un avión experimental asciende siguiendo:

$$y(t) = 200t - 4.9t^2 + 15\sin(0.1t) \quad \text{(m)}$$

Se desea estimar la velocidad vertical en $t = 10$ s usando diferencia centrada con $h = 0.1$ s.

Procedimiento: La diferencia centrada se define como:

$$v(t_i) \approx \frac{y(t_i + h) - y(t_i - h)}{2h}$$

Sustituyendo:

$$v(10) \approx \frac{y(10.1) - y(9.9)}{0.2}$$

Cálculo en R:

```r
y <- function(t) { 200*t - 4.9*t^2 + 15*sin(0.1*t) }
t_i <- 10
h <- 0.1

v_central <- (y(t_i + h) - y(t_i - h)) / (2*h)
v_central
```

Resultado: $v(10) \approx 102.81$ m/s

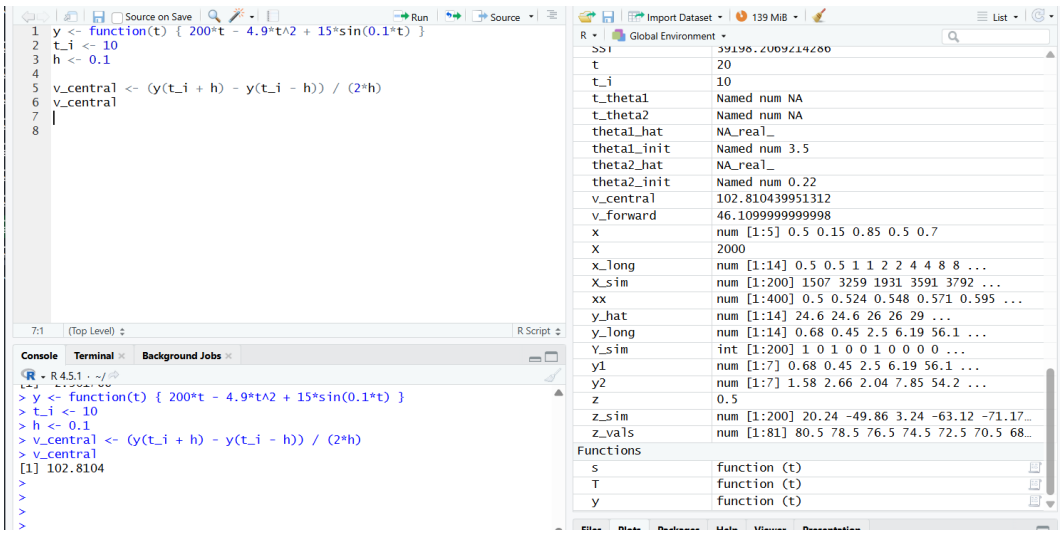


Figura 11.3: Velocidad de un avión

Segunda Derivada - Aceleración de un cohete

Problema: La trayectoria vertical de un cohete está dada por:

$$x(t) = 5t^3 - 20t^2 + 100t \quad \text{(m)}$$

Determinar la aceleración en $t = 5$ s usando la aproximación de segunda derivada con $h = 0.01$ s.

Procedimiento:

$$a(t_i) \approx \frac{x(t_i + h) - 2x(t_i) + x(t_i - h)}{h^2}$$

Sustituyendo:

$$a(5) \approx \frac{x(5.01) - 2x(5) + x(4.99)}{0.01^2}$$

Cálculo en R:

```
x <- function(t) { 5*t^3 - 20*t^2 + 100*t }
t_i <- 5
h <- 0.01

a_approx <- (x(t_i + h) - 2*x(t_i) + x(t_i - h)) / h
    ^2
a_approx
```

Resultado: $a(5) \approx 110$ m/s^2

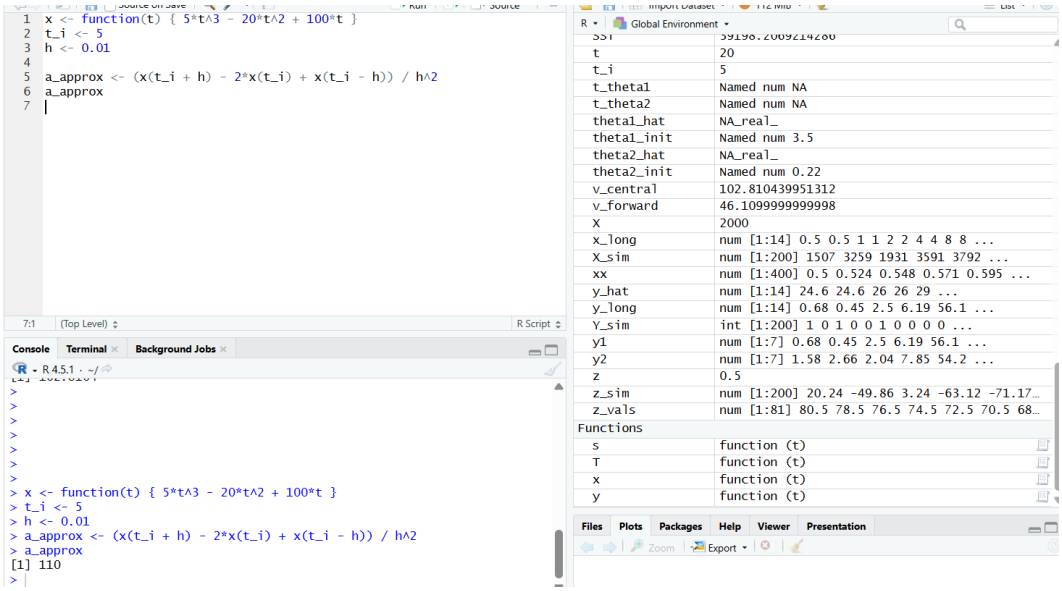


Figura 11.4: Aceleración de un cohete

Error de Truncamiento - Velocidad

Problema: Calcular la velocidad instantánea de un vehículo con $s(t) = 4t^3 - 2t^2 + 5t$ en $t = 2$ s usando diferencia progresiva con pasos $h = 0.1, 0.05, 0.025$ s para evaluar el error de truncamiento.

Procedimiento:

$$v_{approx}(h) = \frac{s(t_i + h) - s(t_i)}{h}, \quad E_T(h) = v_{approx}(h) - v_{exact}$$

$$v_{exact} = s'(2) = 4 * 3 * 2^2 - 2 * 2 + 5 = 45$$

Cálculo en R:

```
s <- function(t) { 4*t^3 - 2*t^2 + 5*t }
t_i <- 2
v_exact <- 45
h_vals <- c(0.1, 0.05, 0.025)

E_T <- sapply(h_vals, function(h){
    v_approx <- (s(t_i + h) - s(t_i)) / h
    v_approx - v_exact
})
data.frame(h=h_vals, error_truncamiento=E_T)
```

Resultado:

h	E_T
0.1	2.24
0.05	1.11
0.025	0.5525

El error disminuye proporcionalmente a h, mostrando el orden 1 del método.

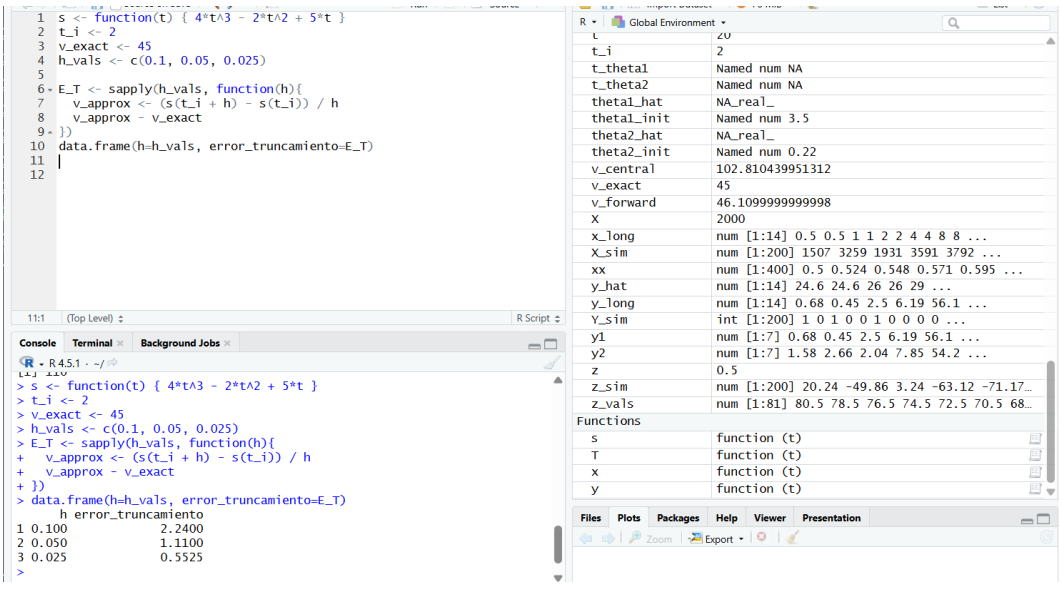


Figura 11.5: Error de Truncamiento

Error de Redondeo - Diferencia centrada

Problema: Calcular la derivada de $f(t) = t^2 + \sin(t)$ en $t = 1$ usando diferencia centrada con paso extremadamente pequeño $h = 1e - 10$ para analizar el error de redondeo.

Procedimiento:

$$f'(t_i) \approx \frac{f(t_i + h) - f(t_i - h)}{2h}, \quad E_{round} = f'_{approx} - f'_{exact}$$

$$f'_{exact}(1) = 1 + \cos(1) \approx 1.217223e - 06$$

Cálculo en R:

```
f <- function(t) { t^2 + sin(t) }
t_i <- 1
h <- 1e-10

df_central <- (f(t_i + h) - f(t_i - h)) / (2*h)
df_exact <- 2*t_i + cos(t_i)
E_round <- df_central - df_exact
E_round
```

Resultado: $f'_{approx} \approx 2.5403$, $E_{round} \approx 2.22 \times 10^{-16}$ (casi cero, error de redondeo mínimo)

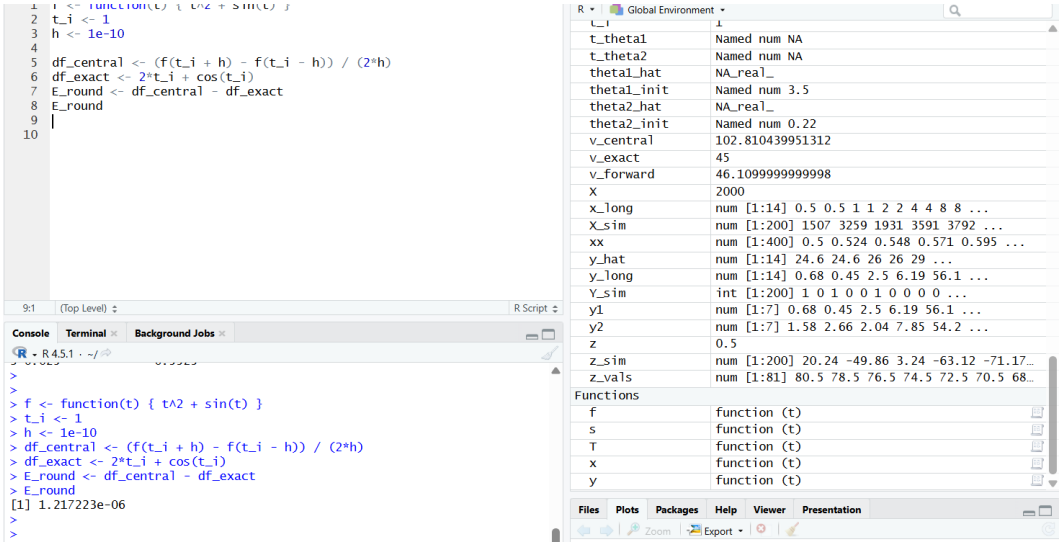


Figura 11.6: Error de Redondeo

Capítulo 12

Interpolación Numérica

12.1. Introducción

La **interpolación numérica** es una técnica fundamental del análisis numérico cuyo objetivo es estimar el valor de una función en puntos intermedios a partir de un conjunto finito de datos discretos conocidos [1, 2, 3, 4]. Este procedimiento resulta especialmente útil cuando solo se dispone de información en ciertos puntos y se desea conocer el comportamiento de la función dentro del intervalo considerado.

En términos generales, la interpolación se emplea cuando la forma analítica de la función es desconocida o resulta difícil de evaluar, pero se dispone de un conjunto de valores obtenidos a partir de mediciones experimentales, simulaciones numéricas o tablas de datos. De esta manera, la interpolación proporciona una herramienta esencial para el análisis y la predicción del comportamiento de la función dentro del dominio de los datos disponibles.

Formalmente, dados $n+1$ puntos distintos $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, donde $y_i = f(x_i)$, se busca un polinomio $P_n(x)$ de grado a lo sumo n tal que:

$$P_n(x_i) = y_i, \quad i = 0, 1, \ldots, n.$$

Este polinomio recibe el nombre de *polinomio interpolante* y es único para un conjunto dado de nodos con abscisas distintas.

La interpolación se diferencia claramente de la extrapolación. Mientras que la interpolación estima valores de la función dentro del intervalo definido por los datos conocidos, la extrapolación se ocupa de valores fuera de este rango, lo que aumenta considerablemente la incertidumbre y el riesgo de error numérico [1]. Por esta razón, en aplicaciones prácticas la extrapolación debe emplearse con mucha precaución, especialmente cuando se desconoce el comportamiento de la función fuera del intervalo de datos disponibles.

La interpolación numérica se emplea de manera frecuente en áreas como la ingeniería, la física, las ciencias computacionales, la economía y el modelado matemático. Esta técnica resulta especialmente útil cuando es necesario estimar valores de funciones a partir de datos experimentales, simulaciones discretas o registros tabulados, asegurando al mismo tiempo un nivel adecuado de precisión en los resultados obtenidos [2, 3, 4].

12.1.1. Interpolación lineal

La interpolación lineal es la forma más sencilla de interpolación numérica. Su objetivo es aproximar el comportamiento de una función entre dos puntos consecutivos mediante una recta, asumiendo que la variación de la función es aproximadamente lineal en ese intervalo.

Dado un par de puntos (x_0, y_0) y (x_1, y_1), la interpolación lineal se expresa mediante la fórmula:

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

Esta expresión puede interpretarse como una combinación ponderada de los valores conocidos, donde el peso de cada punto depende de la proximidad del valor x al nodo correspondiente. De esta manera, el método permite estimar de forma sencilla y rápida los valores intermedios de la función dentro del intervalo considerado.

12.2. Fundamentos teóricos

La **interpolación numérica** es una técnica fundamental dentro del análisis numérico que permite estimar el valor de una función $f(x)$ en puntos intermedios de un dominio conocido, a partir de un conjunto de datos discretos $\{(x_i, y_i)\}_{i=0}^{n}$, donde $y_i = f(x_i)$. Esta herramienta resulta especialmente útil cuando la forma analítica de la función es compleja, desconocida o difícil de evaluar, pero se dispone de información obtenida mediante mediciones experimentales o simulaciones numéricas [5, 2, 1].

El objetivo principal de la interpolación es construir un **polinomio interpolante** $P_n(x)$ de grado n que cumpla la condición

$$P_n(x_i) = f(x_i), \quad i = 0, 1, \ldots, n,$$

de manera que el polinomio coincida exactamente con los datos conocidos y, al mismo tiempo, proporcione una aproximación confiable de la función original en los puntos intermedios. La elección del grado del polinomio y de los nodos $\{x_i\}$ es especialmente importante, ya que afecta directamente la precisión, la estabilidad y la calidad de la aproximación.

Los métodos de interpolación, como los polinomios de Lagrange y de Newton, permiten construir polinomios interpolantes de manera sistemática y con bases matemáticas sólidas. En el caso de Lagrange, se utilizan funciones base que garantizan que el polinomio coincida exactamente con los valores de los nodos conocidos. Por su parte, la interpolación de Newton se basa en diferencias divididas, lo que permite construir el polinomio de manera incremental y facilita la incorporación de nuevos puntos sin tener que recalcular completamente el polinomio.

Adicionalmente, la interpolación mediante splines, y en particular los splines cúbicos, proporciona aproximaciones suaves por tramos, asegurando continuidad tanto en el valor de la función como en sus derivadas. Este enfoque evita las oscilaciones excesivas que pueden aparecer al utilizar polinomios de alto grado, ofreciendo así una alternativa más estable y precisa para muchas aplicaciones prácticas.

El análisis del error en interpolación muestra que la precisión de la aproximación depende de varios factores: la suavidad de la función, la distribución de los nodos y el

grado del polinomio utilizado. Matemáticamente, este error puede expresarse como

$$R_n(x) = f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i), \quad \xi \in [x_0, x_n],$$

lo que indica que la aproximación será más precisa si la función es suave, los nodos están bien distribuidos y el grado del polinomio se ajusta a la variabilidad de la función. Por estas razones, la interpolación numérica se considera una herramienta versátil y poderosa, ampliamente utilizada en ingeniería, ciencias aplicadas, simulación y modelado, ya que permite estimar valores intermedios de manera confiable y eficiente [3, 4, 2, 1, 5].

12.3. Interpolación polinómica

La interpolación polinómica es uno de los métodos más utilizados para estimar valores de una función a partir de un conjunto de puntos discretos conocidos. Su objetivo es construir un polinomio $P_n(x)$ de grado n o menor que pase exactamente por todos los nodos (x_i, y_i) del conjunto de datos.

Chapra y Canale definen la interpolación polinómica como la construcción de un polinomio único que reproduce exactamente los valores dados y que permite evaluar la función en cualquier punto del intervalo considerado. Este enfoque resulta especialmente útil cuando se requiere una aproximación continua de una función originalmente definida de forma discreta.

Por su parte, Burden y Faires destacan que la elección de los nodos de interpolación y el método de construcción del polinomio afectan directamente la estabilidad numérica y la precisión del resultado. En particular, un número elevado de nodos o una mala distribución de los mismos puede provocar oscilaciones indeseadas en el polinomio interpolante.

En la práctica, la interpolación polinómica se emplea en diversas aplicaciones científicas y de ingeniería, tales como el procesamiento de señales, el análisis de datos experimentales, la simulación de procesos físicos y químicos, así como en métodos numéricos más avanzados que requieren evaluaciones repetidas de funciones aproximadas.

Interpolación vs Extrapolación

Es fundamental diferenciar claramente la interpolación de la extrapolación.

- **Interpolación:** consiste en estimar valores de una función dentro del rango definido por los datos conocidos. Los métodos de interpolación son confiables principalmente dentro del dominio de los nodos.
- **Extrapolación:** calcula valores fuera del rango de los datos, lo que generalmente aumenta la incertidumbre y el riesgo de error numérico. Su precisión depende fuertemente de que la función mantenga un comportamiento similar al de los datos conocidos.

12.3.1. Interpolación polinómica de grado superior

Cuando se dispone de tres o más puntos, se puede construir un polinomio de grado mayor para capturar mejor la variación de la función. Por ejemplo, con tres puntos

$(x_0, y_0), (x_1, y_1), (x_2, y_2)$ se construye un polinomio cuadrático:

$$P_2(x) = ax^2 + bx + c.$$

Los coeficientes a, b y c se determinan resolviendo el sistema de ecuaciones:

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

Resolver este sistema permite obtener el polinomio interpolante que pasa exactamente por los tres nodos, capturando la curvatura de la función y mejorando la precisión respecto a la interpolación lineal.

12.3.2. Interpolación de Lagrange

La interpolación de Lagrange permite construir un polinomio interpolante de manera directa, sin necesidad de resolver sistemas de ecuaciones. Para $n+1$ puntos, se define el polinomio base $L_i(x)$:

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \quad i = 0, 1, \ldots, n,$$

y el polinomio interpolante se expresa como:

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

donde cada $L_i(x)$ cumple la propiedad $L_i(x_j) = \delta_{ij}$, asegurando que $P_n(x_i) = y_i$.

Ejemplo Lagrange

Ejemplo: Interpolar $x = 0.3$ con los siguientes datos:

X_i	$f(X_i)$
0.0	1.00000
0.4	1.49182
0.8	2.22554
1.2	3.32011

Aplicando la fórmula de Lagrange:

$$P_n(0.3) = 1.35083.$$

Código en R:

```
x <- c(0.0,0.4,0.8,1.2)
y <- c(1.00000,1.49182,2.22554,3.32011)
lagrange <- function(x, y, xp){
    n <- length(x)
    p <- 0
    for(i in 1:n){
        L <- 1
        for(j in 1:n){
            if(i != j) L <- L*(xp - x[j])/(x[i]-x[j
                ])
        }
        p <- p + y[i]*L
    }
    return(p)
}
lagrange(x, y, 0.3)
```

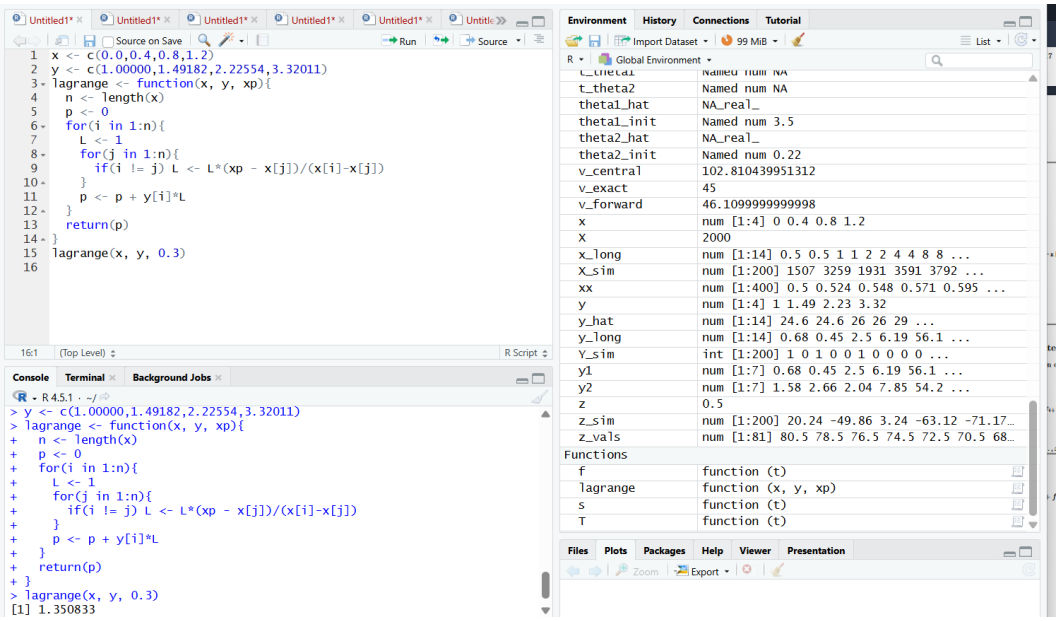


Figura 12.1: Lagrange

12.3.3. Interpolación de Newton mediante diferencias divididas

Newton propuso un método incremental basado en diferencias divididas, lo que permite agregar nuevos nodos sin recalcular todo el polinomio.

Diferencias divididas:

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

y en general para $k \geq 1$:

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

Polinomio interpolante de Newton:

$$P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \ldots$$

Pasos:

1. Ordenar los datos según x_i.
2. Construir la tabla de diferencias divididas.
3. Aplicar la fórmula de Newton.
4. Evaluar $P_n(x)$ en el punto deseado.

Ejemplo 1: Temperatura y densidad

Se dispone de los siguientes datos de densidad ρ en función de la temperatura T:

i	$T_i(\check{r}C)$	$\rho_i(kg/m^3)$
0	94	929
1	205	902
2	371	860

Interpolar la temperatura correspondiente a $\rho = 890.5$ usando el método de Newton:

$$P_n(890.5) = 251.26 \check{r} C.$$

Ejemplo 2: Rendimiento de un proceso químico

Ejemplo Newton

Se dispone de la siguiente tabla:

i	$T(\check{r}C)$	$R(\%)$	Δf	$\Delta^2 f$	$\Delta^3 f$	$\Delta^4 f$	$\Delta^5 f$
0	150	35.5	0.23	0.0175	-0.0012	$4.33e-05$	$-9.92e-07$
1	160	37.8	0.58	-0.0185	0.000533	$-6.25e-06$	
2	170	43.6	0.21	-0.0025	0.000283	$-1.92e-05$	
3	180	45.7	0.16	0.006	-0.000483		
4	190	47.3	0.28	-0.0085			
5	200	50.1	0.11				
6	210	51.2					

Interpolando a $T = 162$ °C:

$$P_n(162) = 39.20\,\% \quad \Rightarrow \text{Proceso dentro del rango óptimo.}$$

Código en R:

```
x <- c(150,160,170,180,190,200,210)
y <- c(35.5,37.8,43.6,45.7,47.3,50.1,51.2)
newton <- function(x, y, xp){
    n <- length(x)
    coef <- y
    for(j in 2:n){
        for(i in n:j){
            coef[i] <- (coef[i] - coef[i-1])/(x[i]-x
               [i-j+1])
        }
    }
    p <- coef[n]
    for(k in (n-1):1) p <- p*(xp - x[k]) + coef[k]
    return(p)
}
newton(x, y, 162)
```

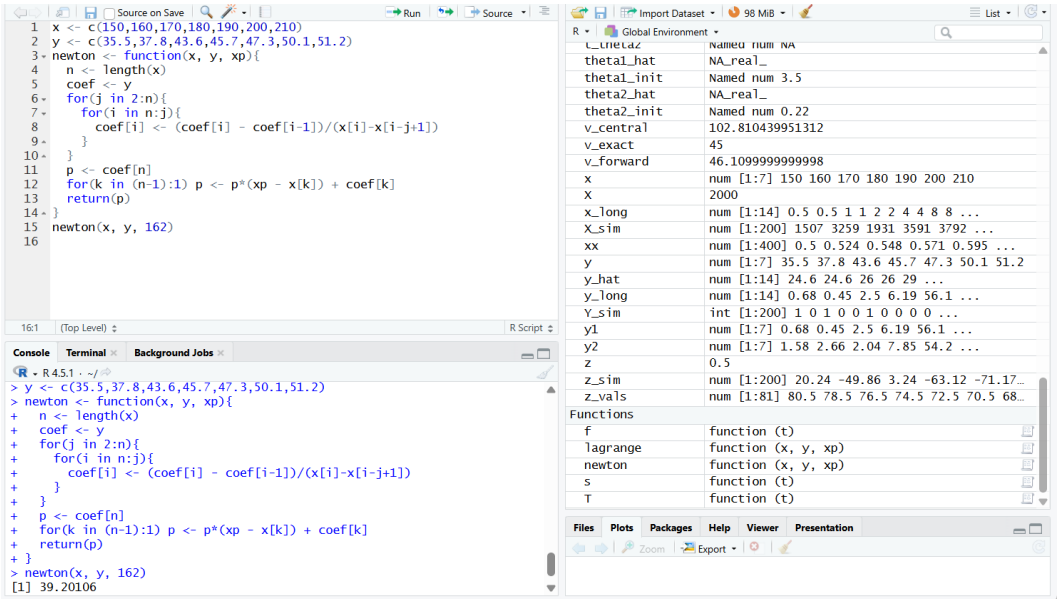


Figura 12.2: Polinomio interpolante de Newton

12.3.4. Interpolación mediante diferencias finitas

Para nodos equiespaciados, se utilizan diferencias progresivas y regresivas, simplificando los cálculos sin resolver sistemas de ecuaciones.

Diferencias progresivas:

$$\Delta y_i = y_{i+1} - y_i, \quad \Delta^2 y_i = \Delta y_{i+1} - \Delta y_i, \ldots$$

Polinomio progresivo:

$$P_n(x) = y_0 + u\Delta y_0 + \frac{u(u-1)}{2!}\Delta^2 y_0 + \ldots, \quad u = \frac{x - x_0}{h}.$$

Diferencias regresivas:

$$\nabla y_i = y_i - y_{i-1}, \quad \nabla^2 y_i = \nabla y_i - \nabla y_{i-1}, \ldots$$

Polinomio regresivo:

$$P_n(x) = y_n + u\nabla y_n + \frac{u(u+1)}{2!}\nabla^2 y_n + \ldots, \quad u = \frac{x - x_n}{h}.$$

Código en R: Diferencias Finitas

```
forward_diff <- function(y, x, xp){
    n <- length(y)
    h <- x[2] - x[1]
    u <- (xp - x[1])/h
    diff_table <- matrix(0, n, n)
    diff_table[,1] <- y
    for(j in 2:n){
        for(i in 1:(n-j+1)){
            diff_table[i,j] <- diff_table[i+1,j-1] -
                diff_table[i,j-1]
        }
    }
    p <- y[1]
    term <- 1
    for(j in 2:n){
        term <- term * (u - (j-2))
        p <- p + term * diff_table[1,j]/factorial(j
            -1)
    }
    return(p)
}
x <- c(0, 1, 2)
y <- c(1, 3, 2)
forward_diff(y, x, 1.5)
```

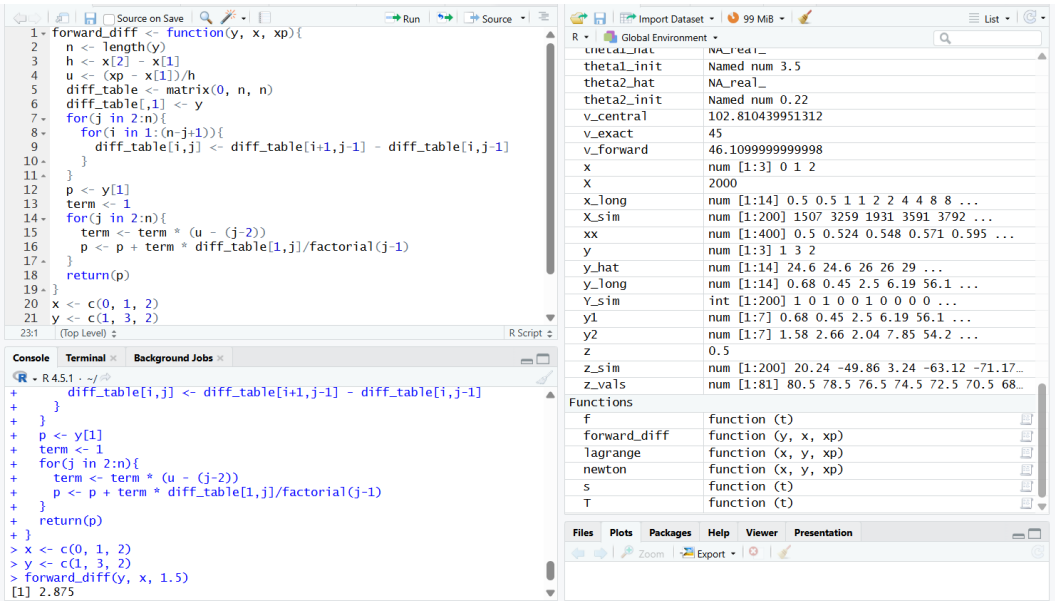


Figura 12.3: Diferencias Finitas

12.3.5. Splines cúbicos

Los **splines cúbicos** permiten interpolar suavemente mediante polinomios de tercer grado en cada intervalo $[x_i, x_{i+1}]$:

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

Condiciones de continuidad:

$$S_i(x_i) = y_i, \quad S_{i-1}(x_i) = S_i(x_i), \quad S'_{i-1}(x_i) = S'_i(x_i), \quad S''_{i-1}(x_i) = S''_i(x_i).$$

Condiciones de frontera natural:

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

12.3.6. Fenómeno de Runge

El fenómeno de Runge ocurre al interpolar funciones suaves con polinomios de alto grado y nodos equiespaciados, generando oscilaciones notables en los extremos.

Ejemplo clásico:

$$f(x) = \frac{1}{1 + 25x^2}, \quad x \in [-1, 1].$$

Fenómeno de Runge en R

```
runge <- function(x) {
    1 / (1 + 25 * x^2)
}
n <- 10
x_nodes <- seq(-1, 1, length.out = n + 1)
y_nodes <- runge(x_nodes)
lagrange <- function(x_nodes, y_nodes, xp) {
    n <- length(x_nodes)
    p <- 0
    for (i in 1:n) {
        L <- 1
        for (j in 1:n) {
            if (i != j) {
                L <- L * (xp - x_nodes[j]) / (x_
                    nodes[i] - x_nodes[j])
            }
        }
        p <- p + y_nodes[i] * L
    }
    return(p)
}
x_test <- c(-1, -0.5, 0, 0.5, 1)
exactos <- runge(x_test)
aprox <- sapply(x_test, function(xp) lagrange(x_
   nodes, y_nodes, xp))
error <- abs(exactos - aprox)
resultado <- data.frame(
x = x_test,
f_exacta = exactos,
f_interpolada = aprox,
error_absoluto = error
)
print(resultado)
```

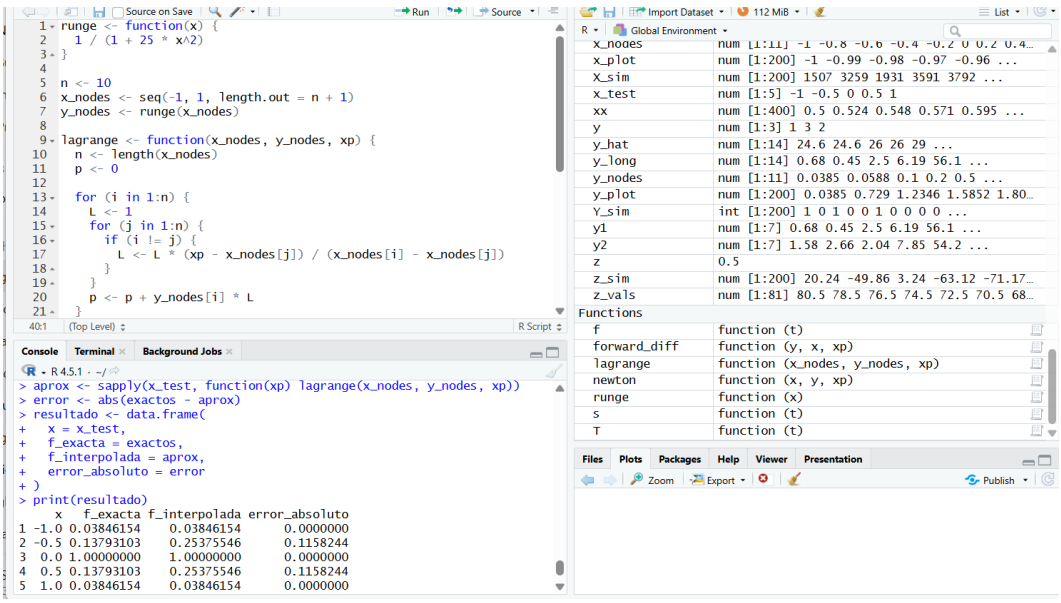


Figura 12.4: Fenómeno de Runge

Capítulo 13

Google Lighthouse y Apache JMeter

13.1. Introducción

En el desarrollo y mantenimiento de aplicaciones web modernas, es importante contar con herramientas que ayuden a revisar diferentes aspectos del sistema, como la calidad del sitio web, la experiencia del usuario, la facilidad de uso y cómo se comporta el sistema cuando hay muchas solicitudes al mismo tiempo. Estas revisiones permiten mejorar el funcionamiento de la aplicación y asegurar que los usuarios tengan una experiencia satisfactoria.

En este contexto, **Google Lighthouse** y **Apache JMeter** son dos herramientas muy utilizadas en el desarrollo de software y aplicaciones web. Aunque ambas sirven para evaluar aplicaciones, cada una tiene un propósito diferente y se enfoca en distintos aspectos del sistema. Google Lighthouse se centra principalmente en analizar la calidad del sitio web, la accesibilidad, el rendimiento y la experiencia del usuario en el navegador. Por su parte, Apache JMeter se orienta a medir cómo responde el sistema ante múltiples solicitudes simultáneas, permitiendo evaluar su rendimiento y estabilidad bajo carga.

Google Lighthouse se enfoca principalmente en analizar la calidad del *frontend*, revisando aspectos visuales, de accesibilidad, rendimiento y buenas prácticas de desarrollo. Esta herramienta permite detectar problemas en la estructura y diseño de las páginas, evaluar si los contenidos son accesibles para todos los usuarios y medir tiempos de carga, ayudando a los desarrolladores a mejorar la experiencia del usuario de manera significativa.

Por otro lado, Apache JMeter se centra en evaluar cómo se comportan los sistemas y servicios cuando procesan solicitudes, siendo especialmente útil para pruebas de carga y rendimiento del *backend*. Con JMeter, es posible simular múltiples usuarios enviando solicitudes simultáneas, medir tiempos de respuesta, detectar cuellos de botella y analizar la estabilidad del sistema bajo diferentes condiciones de tráfico. De esta manera, ambas herramientas se complementan, ya que Lighthouse se ocupa del *frontend* y la experiencia del usuario, mientras que JMeter permite garantizar que el *backend* soporte la demanda y funcione de manera eficiente.

Este documento ofrece una descripción teórica detallada de Google Lighthouse y Apache JMeter, incluyendo su finalidad, fundamentos conceptuales y las principales métricas que permiten evaluar. El objetivo es mostrar de manera clara cómo estas herramientas pueden ser útiles y aplicadas en el desarrollo y mejora de software, facilitando tanto la optimización del *frontend* como la evaluación del rendimiento del

backend.

13.2. Fundamentos teóricos

En la actualidad, el rendimiento, la accesibilidad y la experiencia de usuario constituyen factores determinantes para el éxito de aplicaciones y sitios web, ya que influyen directamente en la satisfacción del usuario, la retención de clientes y la competitividad en entornos digitales. La optimización de estos aspectos no solo implica mejorar la velocidad de carga o la estabilidad visual, sino también garantizar que las interfaces sean intuitivas, accesibles y conformes a estándares web internacionales.

Para evaluar, medir y optimizar estas características, se utilizan **herramientas especializadas de auditoría y pruebas de rendimiento**, capaces de proporcionar información cuantitativa y cualitativa sobre diversos indicadores clave. Entre las más reconocidas en la industria se encuentran **Google Lighthouse** y **Apache JMeter**, cada una enfocada en aspectos complementarios del análisis web.

Google Lighthouse

Google Lighthouse es una herramienta de auditoría automatizada desarrollada por Google que permite evaluar la calidad de las páginas web desde múltiples perspectivas, incluyendo desempeño, accesibilidad, buenas prácticas, SEO y experiencia de usuario [15][16]. Lighthouse ofrece métricas detalladas y sugerencias de mejora, facilitando la identificación de problemas críticos que pueden afectar la percepción y la interacción del usuario con el sitio. Entre sus métricas más relevantes se encuentran los **Core Web Vitals**, que cuantifican aspectos esenciales como la *rapidez de carga*, la *interactividad* y la *estabilidad visual* de los elementos de la página, proporcionando un marco estandarizado para la mejora continua de la experiencia del usuario [15]. Además, Lighthouse permite ejecutar auditorías tanto en entornos controlados de laboratorio como en condiciones reales de usuario, lo que garantiza la fiabilidad y relevancia de sus resultados.

Apache JMeter

Apache JMeter es una herramienta de código abierto diseñada para realizar pruebas de carga y evaluar el rendimiento de aplicaciones web, servicios y sistemas distribuidos [17][18]. Su origen se remonta a finales de los años 90 como una solución flexible para simular múltiples usuarios interactuando simultáneamente con un sistema, permitiendo analizar cómo responde bajo diferentes condiciones de estrés. JMeter facilita la creación de escenarios de prueba complejos, mediante la simulación de solicitudes HTTP, HTTPS, SOAP, REST, FTP y otros protocolos, y proporciona métricas precisas sobre tiempos de respuesta, throughput, errores y consumo de recursos. Este análisis permite identificar cuellos de botella, prever la capacidad del sistema y optimizar su infraestructura antes de su despliegue en entornos de producción.

La fundamentación teórica de JMeter se sustenta en conceptos de **procesos estocásticos y teoría de colas**, desarrollados por autores como Erlang (1909) [20], Kendall (1953) [21] y Kingman (1961) [22], que permiten modelar matemáticamente la llegada de solicitudes y la atención por servidores. Esto proporciona un marco riguroso

para interpretar los resultados de las pruebas de carga y entender el comportamiento dinámico de los sistemas frente a variaciones en la demanda.

En conjunto, **Google Lighthouse y Apache JMeter representan herramientas complementarias**: mientras Lighthouse se centra en la experiencia del usuario y el cumplimiento de estándares web, JMeter se orienta a la capacidad operativa y la resistencia del sistema bajo carga. La integración de ambas herramientas en el ciclo de desarrollo y mantenimiento web permite a los equipos técnicos garantizar aplicaciones rápidas, estables y eficientes, mejorando significativamente la calidad global del servicio digital ofrecido.

13.3. Contexto del desarrollo web moderno

El crecimiento acelerado de las aplicaciones web en los últimos años ha transformado la manera en que los usuarios interactúan con sistemas informáticos. Actualmente, los servicios web no solo deben cumplir con requisitos funcionales, sino también con criterios relacionados con la eficiencia, la usabilidad, la accesibilidad y la estabilidad bajo distintas condiciones de uso.

Las aplicaciones web modernas suelen estar compuestas por múltiples capas, entre ellas la capa de presentación (frontend), la capa lógica (backend) y la capa de datos. Cada una de estas capas presenta desafíos específicos que requieren herramientas especializadas para su análisis y evaluación. En este escenario, surge la necesidad de utilizar plataformas que permitan estudiar el comportamiento del sistema desde diferentes perspectivas, sin necesidad de modificar directamente el código fuente.

La evaluación de aplicaciones web se ha convertido en una práctica esencial dentro del ciclo de vida del software, ya que permite identificar problemas potenciales antes de que estos afecten a los usuarios finales. Herramientas como Google Lighthouse y Apache JMeter se insertan en este contexto como soluciones ampliamente adoptadas para analizar distintos aspectos del funcionamiento de aplicaciones web.

13.4. Importancia de la evaluación de aplicaciones web

La evaluación de aplicaciones web es un proceso fundamental que permite medir el grado en que un sistema cumple con los requisitos técnicos y de experiencia de usuario esperados. Un sitio web que presenta tiempos de carga elevados, problemas de accesibilidad o fallos bajo alta demanda puede generar una percepción negativa en los usuarios, afectando la credibilidad y utilidad del sistema.

Desde el punto de vista de la ingeniería de software, la evaluación permite detectar errores de diseño, deficiencias en la arquitectura del sistema y problemas relacionados con el uso ineficiente de recursos. Además, facilita la toma de decisiones técnicas basadas en métricas objetivas y reproducibles.

El uso de herramientas automatizadas reduce la subjetividad del análisis y proporciona resultados estandarizados. Estas herramientas permiten evaluar aplicaciones web en diferentes entornos, dispositivos y condiciones de red, lo que resulta especialmente relevante en un contexto donde los usuarios acceden a los servicios desde una amplia variedad de plataformas.

13.5. Google Lighthouse

13.5.1. Definición

Google Lighthouse es una herramienta de código abierto desarrollada por Google que permite realizar auditorías automáticas sobre aplicaciones web. Su objetivo principal es evaluar la calidad de una página web desde el punto de vista del usuario final, considerando aspectos como rendimiento, accesibilidad, buenas prácticas, optimización para motores de búsqueda (SEO) y compatibilidad con aplicaciones web progresivas (PWA) [19].

13.5.2. Arquitectura y componentes de Google Lighthouse

Google Lighthouse está diseñado como un sistema modular que ejecuta auditorías independientes sobre distintos aspectos de una aplicación web. Cada auditoría corresponde a un conjunto de reglas y verificaciones que se aplican durante la carga y ejecución de la página en un entorno controlado.

La herramienta utiliza el motor del navegador Chromium para simular la carga de una página web, registrando eventos relacionados con el renderizado, la ejecución de scripts y la interacción del usuario. A partir de estos datos, Lighthouse calcula métricas específicas que permiten evaluar el desempeño general del sitio.

Los resultados generados por Lighthouse se presentan en forma de informes estructurados que incluyen puntuaciones numéricas, descripciones detalladas y recomendaciones técnicas. Esta arquitectura facilita la comprensión de los resultados incluso para usuarios con conocimientos técnicos básicos.

13.5.3. ¿Para qué sirve Google Lighthouse?

Lighthouse sirve para:

- Evaluar la experiencia de usuario en aplicaciones web.
- Detectar problemas de accesibilidad para usuarios con discapacidades.
- Verificar el uso de buenas prácticas de desarrollo web.
- Analizar el rendimiento de carga y visualización del contenido.
- Identificar oportunidades de mejora en SEO.

Esta herramienta es utilizada tanto por desarrolladores como por diseñadores web y equipos de calidad.

13.5.4. Métricas principales

Las métricas empleadas por Google Lighthouse tienen como objetivo cuantificar distintos aspectos de la experiencia del usuario durante la carga y visualización de una página web. Estas métricas se basan en eventos medibles del ciclo de renderizado del navegador y buscan reflejar cómo percibe el usuario la rapidez, estabilidad y fluidez del contenido presentado. A continuación, se describen las métricas más relevantes desde un enfoque teórico.

First Contentful Paint (FCP)

El *First Contentful Paint* (FCP) mide el tiempo transcurrido desde que el navegador inicia la navegación hacia una página web hasta que se renderiza por primera vez cualquier contenido visible en la pantalla. Este contenido puede corresponder a texto, imágenes, elementos SVG o incluso fondos renderizados mediante CSS.

Desde una perspectiva teórica, el FCP representa el primer indicio perceptible para el usuario de que la página está respondiendo. Diversos estudios en usabilidad indican que la percepción de rapidez no depende únicamente del tiempo total de carga, sino de la rapidez con la que aparece el primer elemento visual significativo.

Matemáticamente, el FCP se define como:

$$FCP = t_{primer\_contenido} - t_{navegacion} \tag{13.1}$$

Un valor elevado de FCP puede indicar problemas relacionados con la carga de hojas de estilo, el bloqueo del hilo principal por scripts o una configuración ineficiente de recursos críticos. Por ello, esta métrica es utilizada como un indicador temprano de la experiencia de carga percibida.

Largest Contentful Paint (LCP)

El *Largest Contentful Paint* (LCP) mide el instante en el que se renderiza el elemento de mayor tamaño dentro del área visible de la ventana del navegador. Este elemento suele corresponder a imágenes principales, bloques de texto extensos o contenedores relevantes para el contenido central de la página.

Desde el punto de vista conceptual, el LCP busca identificar cuándo el contenido principal de la página ha sido cargado y es completamente visible para el usuario. A diferencia del FCP, que se centra en el primer contenido, el LCP se enfoca en el contenido más relevante desde una perspectiva informativa.

Formalmente, el LCP se expresa como:

$$LCP = t_{elemento\_principal} - t_{inicio} \tag{13.2}$$

Esta métrica es especialmente importante en aplicaciones web modernas, donde el contenido principal suele cargarse de manera diferida. Valores altos de LCP pueden estar asociados a imágenes no optimizadas, tiempos elevados de respuesta del servidor o una carga excesiva de recursos externos.

Total Blocking Time (TBT)

El *Total Blocking Time* (TBT) cuantifica el tiempo total durante el cual el hilo principal del navegador se encuentra bloqueado por tareas largas que impiden la interacción del usuario con la página. Una tarea se considera larga cuando su duración excede los 50 milisegundos.

Desde un enfoque teórico, el TBT está relacionado con la capacidad de respuesta de la aplicación web. Aunque una página pueda mostrarse visualmente, un alto valor de TBT indica que el usuario no puede interactuar de forma fluida debido a la ejecución prolongada de scripts, generalmente asociados a JavaScript.

La expresión matemática del TBT es:

$$TBT = \sum (t_{tarea} - 50) \tag{13.3}$$

Esta métrica resulta clave para evaluar la interactividad, ya que refleja el impacto del procesamiento en el lado del cliente sobre la experiencia del usuario, especialmente en dispositivos con recursos limitados.

Cumulative Layout Shift (CLS)

El *Cumulative Layout Shift* (CLS) mide la estabilidad visual de una página web cuantificando los cambios inesperados en la disposición de los elementos durante la carga. Estos desplazamientos ocurren cuando elementos visibles cambian de posición sin una acción directa del usuario.

Desde una perspectiva conceptual, el CLS está estrechamente vinculado a la calidad de la experiencia visual. Cambios abruptos en el diseño pueden generar confusión, errores de interacción y una percepción negativa de la calidad del sitio web.

El CLS se calcula mediante la siguiente expresión:

$$CLS = \sum(impact\_fraction \times distance\_fraction) \tag{13.4}$$

Donde el *impact fraction* representa la proporción de la pantalla afectada por el cambio y el *distance fraction* indica la distancia relativa del desplazamiento. Valores elevados de CLS suelen estar asociados a la carga tardía de imágenes, anuncios o fuentes web sin dimensiones previamente definidas.

13.5.5. Funcionamiento

Lighthouse ejecuta una serie de auditorías automáticas en un entorno controlado del navegador. Los resultados se comparan con umbrales establecidos por Google, generando un informe estructurado que incluye puntuaciones y recomendaciones técnicas.

13.6. Prácticas

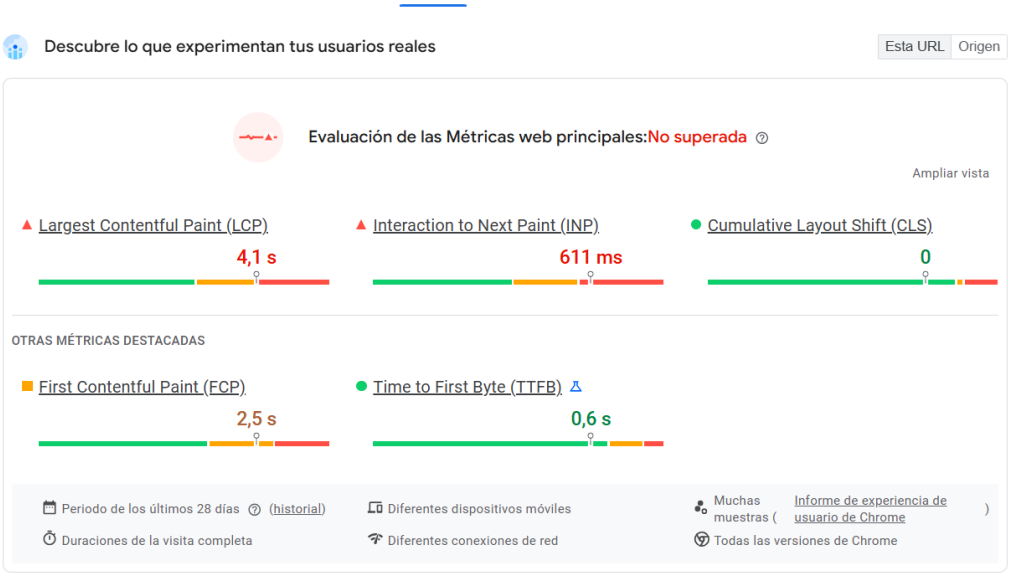


Figura 13.1: Captura Lighthouse 1

Figura 13.2: Captura Lighthouse 2

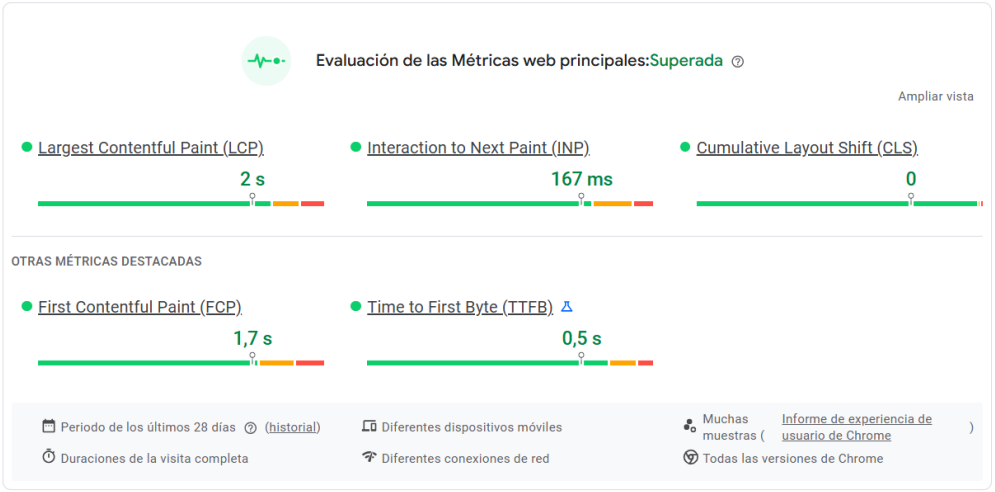


Figura 13.3: Captura Lighthouse 3

13.7. Apache JMeter

13.7.1. Definición

Apache JMeter es una herramienta de código abierto desarrollada por la Apache Software Foundation para la evaluación del comportamiento de sistemas informáticos bajo diferentes condiciones de carga [17]. Fue creada originalmente por Stefano Mazzocchi [18].

13.7.2. Arquitectura de Apache JMeter

Apache JMeter se basa en una arquitectura compuesta por elementos denominados *samplers*, *listeners*, *controllers* y *timers*. Cada uno de estos componentes cumple una

función específica dentro del proceso de simulación de carga.

Los samplers representan las solicitudes enviadas al sistema bajo prueba, mientras que los listeners recopilan y presentan los resultados obtenidos. Los controllers permiten definir la lógica de ejecución de las pruebas, y los timers controlan los intervalos de tiempo entre solicitudes.

Esta arquitectura modular permite construir escenarios de prueba complejos y adaptables a distintos tipos de sistemas.

13.7.3. ¿Para qué sirve Apache JMeter?

Apache JMeter se utiliza para:

- Simular múltiples usuarios accediendo a un sistema.
- Analizar el comportamiento de aplicaciones web y APIs.
- Evaluar la capacidad de respuesta de servidores y servicios.
- Detectar cuellos de botella en sistemas distribuidos.

13.7.4. Métricas conceptuales

Tiempo de respuesta promedio

$$\bar{T} = \frac{1}{n} \sum_{i=1}^{n} T_i \tag{13.5}$$

Throughput

$$Throughput = \frac{N}{\Delta t} \tag{13.6}$$

Latencia

$$Latency = t_{respuesta} - t_{solicitud} \tag{13.7}$$

Tasa de error

$$Error\ Rate = \frac{N_{errores}}{N_{total}} \tag{13.8}$$

13.7.5. Tipos de pruebas

Apache JMeter permite realizar distintos tipos de pruebas, entre ellas:

- Pruebas de carga
- Pruebas de estrés
- Pruebas de resistencia (soak testing)
- Pruebas de picos de carga

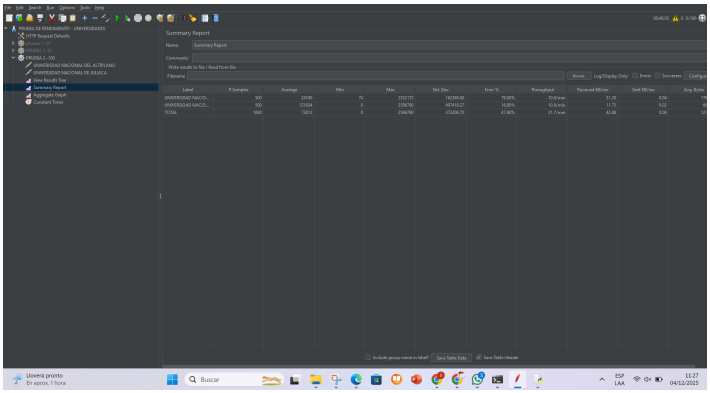


Figura 13.4: Muestra JMeter 1

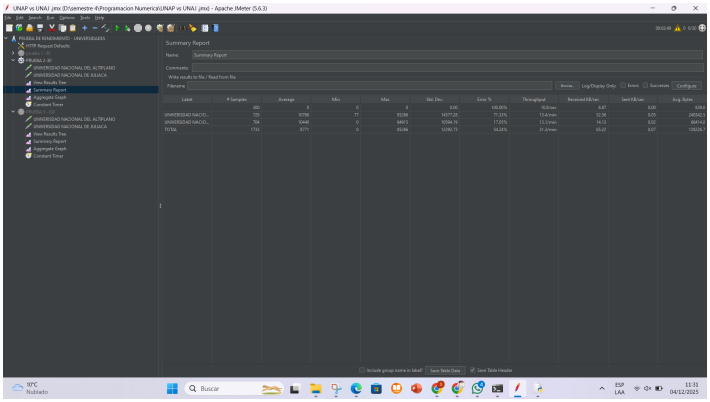


Figura 13.5: Muestra JMeter 2

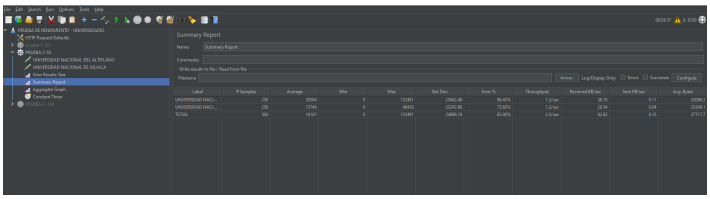


Figura 13.6: Muestra JMeter 3

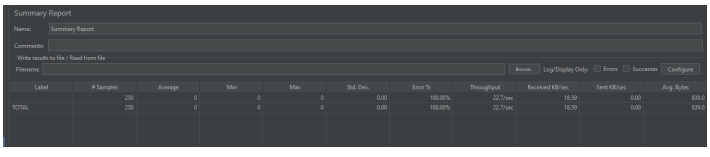


Figura 13.7: Muestra JMeter 4

13.8. Evaluación orientada al sistema

La evaluación orientada al sistema se enfoca en analizar el comportamiento interno de una aplicación bajo distintas condiciones de carga. A diferencia de la evaluación

centrada en el usuario, este enfoque considera métricas relacionadas con el procesamiento de solicitudes, el uso de recursos y la capacidad de respuesta del sistema.

Este tipo de evaluación es especialmente relevante para aplicaciones que deben atender a un gran número de usuarios simultáneos, como plataformas educativas, sistemas bancarios y servicios en línea. Apache JMeter se utiliza ampliamente en este contexto debido a su capacidad para simular múltiples usuarios concurrentes.

Capítulo 14

Eigenvalores y Eigenvectores

14.1. Introducción

Los **eigenvalores** y **eigenvectores** son conceptos clave del álgebra lineal que permiten entender cómo actúan las transformaciones lineales sobre un espacio vectorial. Dada una matriz cuadrada A de tamaño $n \times n$, esta define una transformación que puede cambiar tanto la dirección como la magnitud de cualquier vector en $\mathbb{R}^n$. Sin embargo, existen vectores especiales que, al aplicarse la transformación, no cambian su dirección; únicamente se escalan por un factor, que puede ser un número real o complejo.

Este factor de escala se llama **eigenvalor** (λ), y el vector correspondiente se denomina **eigenvector** (v). Matemáticamente, esta relación se expresa como:

$$Av = \lambda v, \quad v \neq 0.$$

Los **eigenvectores** representan las direcciones que permanecen invariantes bajo la transformación definida por A, mientras que los **eigenvalores** indican cuánto se estira o se comprime el vector en esa dirección. Por ejemplo, un eigenvalor positivo mayor que uno indica que el vector se amplía, un eigenvalor entre cero y uno señala que se contrae, y un eigenvalor negativo refleja una inversión de dirección. Esta propiedad geométrica permite interpretar de manera intuitiva el comportamiento de los sistemas lineales y facilita el diseño de soluciones más eficientes y precisas.

Además, los eigenvalores y eigenvectores tienen gran relevancia práctica, ya que simplifican el análisis de transformaciones complejas y ayudan a comprender mejor el comportamiento de sistemas lineales. Por ejemplo, al diagonalizar una matriz, se pueden reducir operaciones complicadas a cálculos más sencillos sobre los eigenvalores, facilitando la resolución de problemas como el cálculo de potencias de matrices o la solución de ecuaciones diferenciales lineales. Asimismo, estos conceptos se aplican en física para estudiar vibraciones y modos de oscilación, en ingeniería para analizar sistemas dinámicos y controlar procesos, en computación para compresión de datos y procesamiento de gráficos, y en economía para modelar crecimiento, equilibrio y riesgo financiero.

En el contexto de la **optimización multivariable**, los eigenvalores de la matriz Hessiana —que contiene las segundas derivadas parciales de una función $f(x_1, x_2, \ldots, x_n)$— permiten determinar la naturaleza de los puntos críticos. Si todos los eigenvalores son positivos, el punto crítico es un mínimo local; si son negativos, es un máximo local; y si

tienen signos mixtos, el punto es una silla. Además, los eigenvalores y eigenvectores se utilizan para diagonalizar matrices, simplificar sistemas lineales, analizar estabilidad de soluciones, y en métodos numéricos avanzados como el método de la potencia o la descomposición espectral.

En resumen, los eigenvalores y eigenvectores no solo proporcionan información sobre la magnitud y dirección de los efectos de una transformación lineal, sino que también permiten comprender la estructura intrínseca de la matriz, facilitando la resolución de problemas complejos en física, ingeniería, ciencias de la computación y matemáticas aplicadas.

14.1.1. Interpretación geométrica

Geométricamente, un eigenvector representa una dirección en el espacio que permanece inalterada bajo la transformación lineal definida por la matriz A. Es decir, al aplicar la matriz A sobre un eigenvector v, el vector resultante apunta exactamente en la misma dirección (o en la dirección opuesta, si el eigenvalor es negativo), aunque su longitud puede cambiar según el eigenvalor λ. Esto se puede expresar visualmente como un estiramiento o compresión a lo largo de líneas específicas en el espacio.

Algunos casos importantes son:

- $\lambda > 1$: El vector se amplifica, aumentando su magnitud sin cambiar de dirección.
- $0 < \lambda < 1$: El vector se contrae, reduciendo su magnitud pero conservando la dirección.
- $\lambda < 0$: El vector se invierte, apuntando en la dirección opuesta y escalándose por $|\lambda|$.
- $\lambda = 0$: El vector se colapsa al origen.

Esta interpretación geométrica es especialmente útil en aplicaciones de optimización, análisis de sistemas dinámicos y física, ya que permite identificar direcciones de máxima expansión, contracción o inversión. Por ejemplo, en la diagonalización de matrices, cada eigenvector define un eje principal de la transformación, y los eigenvalores asociados indican la magnitud del estiramiento o compresión a lo largo de ese eje. De esta manera, las transformaciones complejas pueden entenderse como combinaciones de efectos simples a lo largo de direcciones invariantes.

14.2. Fundamentos teóricos

En álgebra lineal, los conceptos de **eigenvalores** y **eigenvectores** son fundamentales para analizar matrices y transformaciones lineales [6, 7, 8]. Un eigenvalor es un número que indica cómo se escala un vector especial, llamado eigenvector, cuando se aplica una transformación lineal [6, 7]. Estos vectores no cambian su dirección bajo la transformación; únicamente se modifican en magnitud [8]. Esta relación permite conectar la estructura algebraica de una matriz con las propiedades geométricas de los vectores sobre los que actúa, facilitando la comprensión del comportamiento de sistemas lineales y la resolución de problemas prácticos en física, ingeniería, economía y otras áreas aplicadas [9, 8].

El estudio de los **eigenvalores** y **eigenvectores** permite descomponer matrices y comprender su comportamiento interno, proporcionando información valiosa sobre la estabilidad de sistemas lineales, la multiplicidad de soluciones y la estructura de los

espacios vectoriales asociados [6, 7, 8]. Además, estos conceptos constituyen la base para desarrollar teorías avanzadas en análisis numérico, optimización, dinámica de sistemas y modelado matemático [9, 8]. Comprender la naturaleza de los eigenvalores y eigenvectores es fundamental para interpretar correctamente cómo actúan las transformaciones lineales y para diseñar métodos que aprovechen las propiedades espectrales de las matrices [6, 8].

La comprensión profunda de los eigenvalores y eigenvectores no solo es clave desde el punto de vista teórico, sino que también resulta esencial en la práctica, especialmente en el análisis numérico y la simulación de sistemas complejos. Por ejemplo, en métodos computacionales para resolver sistemas lineales o para la diagonalización de matrices, el conocimiento de las propiedades espectrales permite optimizar algoritmos y garantizar estabilidad y eficiencia en los cálculos [9, 8]. Esta capacidad de vincular la teoría algebraica con aplicaciones prácticas hace que los eigenvalores y eigenvectores sean herramientas indispensables en ingeniería, física, economía y ciencias computacionales.

14.3. Ecuación característica y origen de las fórmulas

Partiendo de la definición $A\mathbf{v} = \lambda\mathbf{v}$, se obtiene:

$$(A - \lambda I)\mathbf{v} = \mathbf{0} \tag{14.1}$$

Para que exista una solución no trivial, el determinante debe anularse:

$$\det(A - \lambda I) = 0 \tag{14.2}$$

Esta expresión se denomina **ecuación característica**. El polinomio resultante, llamado **polinomio característico**, tiene como raíces los eigenvalores de la matriz.

14.4. Propiedades fundamentales

Los eigenvalores y eigenvectores poseen varias propiedades esenciales que facilitan su estudio y aplicación en problemas de álgebra lineal, optimización y análisis de sistemas. Entre las más importantes se destacan:

1. **Número de eigenvalores:** Una matriz cuadrada A de dimensión $n \times n$ tiene exactamente n eigenvalores, contando multiplicidades algebraicas. Algunos de ellos pueden ser iguales o complejos, dependiendo de la matriz.
2. **Suma y producto de eigenvalores:**
   - La suma de los eigenvalores de A es igual a la traza de la matriz ($\text{tr}(A)$), es decir, la suma de sus elementos diagonales.
   - El producto de los eigenvalores es igual al determinante de la matriz ($\det(A)$).
3. **Invariancia bajo similitud:** Si $B = P^{-1}AP$ es una matriz similar a A, entonces A y B tienen los mismos eigenvalores, aunque sus eigenvectores puedan diferir.
4. **Linealidad de eigenvectores:** Los eigenvectores correspondientes a eigenvalores distintos son linealmente independientes. Esto es especialmente útil para diagonalizar matrices.
5. **Matrices diagonales y triangulares:**
   - En una matriz diagonal, los eigenvalores son los elementos de la diagonal.

- En una matriz triangular (superior o inferior), los eigenvalores también son los elementos de la diagonal.

6. **Transposición y conjugación:**
   - Los eigenvalores de A^T son iguales a los de A.
   - Si A es una matriz compleja, los eigenvalores de A^* (conjugada transpuesta) son los conjugados de los eigenvalores de A.
7. **Multiplicación por escalar:** Si α es un escalar, los eigenvalores de αA son $\alpha\lambda_i$, donde λ_i son los eigenvalores de A, y los eigenvectores permanecen iguales.
8. **Inversión:** Si A es invertible, los eigenvalores de A^{-1} son los inversos de los eigenvalores de A $(1/\lambda_i)$ y los eigenvectores son los mismos.

Estas propiedades no solo facilitan el cálculo de eigenvalores y eigenvectores, sino que también permiten comprender de manera más profunda la estructura interna de la matriz y la naturaleza de las transformaciones lineales que representa. Son herramientas fundamentales en optimización, análisis de estabilidad, sistemas dinámicos y métodos numéricos avanzados.

14.5. Diagonalización y descomposición espectral

La **diagonalización** es un proceso mediante el cual una matriz cuadrada A se transforma en una matriz diagonal D mediante una matriz de cambio de base P compuesta por los eigenvectores de A. Este procedimiento simplifica significativamente el análisis de la matriz, ya que una matriz diagonal es fácil de elevar a potencias, invertir y estudiar. Matemáticamente, se expresa como:

$$A = PDP^{-1},$$

donde:

- D es una matriz diagonal cuyos elementos son los eigenvalores de A.
- P es una matriz cuyas columnas son los eigenvectores linealmente independientes de A.

14.5.1. Condiciones para la diagonalización

No todas las matrices son diagonalizables. Para que A sea diagonalizable se requiere:

1. Que tenga n eigenvectores linealmente independientes, donde n es la dimensión de la matriz.
2. Que la multiplicidad geométrica de cada eigenvalor (número de eigenvectores independientes asociados) sea igual a su multiplicidad algebraica.

14.5.2. Descomposición espectral

Cuando A es simétrica $(A = A^T)$ o hermítica en el caso complejo, se puede realizar la **descomposición espectral**:

$$A = Q\Lambda Q^T \quad \text{o} \quad A = Q\Lambda Q^*,$$

donde:

- Q es una matriz ortogonal (o unitaria en el caso complejo) cuyos columnas son eigenvectores normalizados.
- Λ es una matriz diagonal que contiene los eigenvalores de A.

Esta descomposición es especialmente útil en optimización, análisis de estabilidad y métodos numéricos, ya que permite representar A como suma de proyecciones a lo largo de sus eigenvectores:

$$A = \sum_{i=1}^{n} \lambda_i v_i v_i^T,$$

donde cada término $\lambda_i v_i v_i^T$ representa la contribución de la dirección del eigenvector v_i escalada por el eigenvalor λ_i.

14.6. Cálculo analítico de eigenvalores y eigenvectores

El cálculo analítico de los eigenvalores y eigenvectores se basa en resolver la ecuación característica de la matriz A:

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

donde I es la matriz identidad de la misma dimensión que A, y λ representa los eigenvalores. Este proceso se realiza generalmente en los siguientes pasos:

14.6.1. Procedimiento paso a paso

1. **Formar la matriz característica:**

   $$A - \lambda I$$

   Se resta λ a cada elemento de la diagonal principal de A.
2. **Calcular el determinante:**

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

   Esta ecuación polinómica en λ se denomina **ecuación característica**. Su grado es igual a la dimensión de la matriz.
3. **Resolver para los eigenvalores:** Se resuelve el polinomio para encontrar todos los valores de λ. En matrices 2x2, se puede usar la fórmula cuadrática:

   $$\lambda = \frac{\operatorname{tr}(A) \pm \sqrt{\operatorname{tr}(A)^2 - 4\det(A)}}{2}$$

4. **Calcular los eigenvectores:** Para cada eigenvalor λ_i, se resuelve el sistema lineal:

   $$(A - \lambda_i I)v_i = 0$$

   Esto proporciona los eigenvectores asociados, que pueden normalizarse para simplificar su representación.

14.6.2. Ejemplo 1: Matriz 2x2 general

Sea

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

1. Matriz característica:

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

2. Determinante:

$$\det(A - \lambda I) = (3-\lambda)^2 - 1 = \lambda^2 - 6\lambda + 8 = 0$$

3. Eigenvalores:

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

4. Eigenvectores:

$$\lambda_1 = 4 : (A - 4I)v_1 = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} v_1 = 0 \implies v_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\lambda_2 = 2 : (A - 2I)v_2 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} v_2 = 0 \implies v_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

14.6.3. Ejemplo 2: Matriz diagonal

Para una matriz diagonal

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

los eigenvalores son directamente los elementos de la diagonal:

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

y los eigenvectores son los vectores unitarios estándar:

$$v_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad v_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

14.6.4. Comentarios finales

El cálculo analítico es práctico para matrices pequeñas (2x2 o 3x3), pero para matrices de mayor dimensión se recomienda utilizar métodos numéricos, ya que el polinomio característico puede ser difícil de factorizar y los eigenvectores pueden requerir soluciones de sistemas lineales grandes.

14.7. Eigenvalores en métodos de optimización

En optimización multivariable, los eigenvalores juegan un papel fundamental para analizar la curvatura de la función objetivo y la naturaleza de los puntos críticos. La herramienta central en este análisis es la **matriz Hessiana**, que contiene todas las segundas derivadas parciales de una función $f(x_1, x_2, \ldots, x_n)$:

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

Los eigenvalores de la Hessiana determinan la naturaleza de un punto crítico x^* donde $\nabla f(x^*) = 0$:

- **Todos los eigenvalores positivos:** x^* es un mínimo local. La función es convexa en todas las direcciones.
- **Todos los eigenvalores negativos:** x^* es un máximo local. La función es cóncava en todas las direcciones.
- **Eigenvalores mixtos:** x^* es un punto silla. Hay direcciones de curvatura positiva y negativa.

14.7.1. Interpretación geométrica en optimización

Cada eigenvector de la Hessiana indica una dirección principal en el espacio de variables, y el eigenvalor correspondiente mide la curvatura de la función en esa dirección. Un eigenvalor grande indica una curvatura pronunciada (cambios rápidos de la función), mientras que un eigenvalor pequeño indica una curvatura suave (cambios lentos de la función). Esta información es fundamental para algoritmos de optimización, como el método de Newton, donde la actualización se realiza considerando la inversa de la Hessiana:

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

14.7.2. Ejemplo práctico

Sea la función:

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

1. Calcular la Hessiana:
$$H = \begin{pmatrix} 2 & 2 \\ 2 & 6 \end{pmatrix}$$
2. Determinar los eigenvalores:
$$\det(H - \lambda I) = \lambda^2 - 8\lambda + 8 = 0 \implies \lambda_1 \approx 6.83,\ \lambda_2 \approx 1.17$$
3. Clasificación del punto crítico $(0, 0)$: Ambos eigenvalores son positivos, por lo que $(0, 0)$ es un **mínimo local**.

14.8. Optimización Completa

La optimización de funciones multivariables implica encontrar puntos críticos donde la función alcanza valores máximos, mínimos o puntos silla. Los eigenvalores y eigenvectores juegan un papel central en este análisis, ya que permiten caracterizar la curvatura de la función y orientar los algoritmos de optimización.

14.8.1. Puntos críticos y la Hessiana

Para una función $f(x_1, x_2, \ldots, x_n)$, un **punto crítico** x^* se encuentra resolviendo:

$$\nabla f(x^*) = 0$$

donde ∇f es el gradiente de f. La naturaleza del punto crítico se analiza mediante la **Hessiana** H:

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

14.8.2. Criterio de eigenvalores para clasificación

Sea $\lambda_1, \lambda_2, \ldots, \lambda_n$ los eigenvalores de $H(x^*)$:

- Todos $\lambda_i > 0$: x^* es un **mínimo local**.
- Todos $\lambda_i < 0$: x^* es un **máximo local**.
- Eigenvalores de signos mixtos: x^* es un **punto silla**.

14.8.3. Interpretación geométrica

Cada eigenvector de la Hessiana indica la dirección principal de curvatura de la función en el punto crítico, mientras que su eigenvalor asociado indica la magnitud de esa curvatura. Direcciones con eigenvalores grandes representan cambios rápidos de la función, mientras que direcciones con eigenvalores pequeños representan cambios suaves.

14.8.4. Ejemplo práctico

Consideremos la función:

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

1. Gradiente:
$$\nabla f = \begin{pmatrix} 2x_1 + 2x_2 \\ 6x_2 + 2x_1 \end{pmatrix}$$
El punto crítico es $(0, 0)$.
2. Hessiana:
$$H = \begin{pmatrix} 2 & 2 \\ 2 & 6 \end{pmatrix}$$

3. Eigenvalores de H:
$$\det(H - \lambda I) = \lambda^2 - 8\lambda + 8 = 0 \implies \lambda_1 \approx 6.83,\ \lambda_2 \approx 1.17$$
4. Clasificación: Ambos eigenvalores son positivos, por lo que $(0, 0)$ es un **mínimo local**.

14.9. Cálculo numérico de eigenvalores

Cuando las matrices son de gran dimensión o el cálculo analítico es complicado, se utilizan métodos numéricos para determinar los eigenvalores y eigenvectores de manera aproximada. Estos métodos son ampliamente utilizados en programación científica, análisis de sistemas dinámicos y optimización numérica.

14.9.1. Métodos más comunes

1. **Método de la potencia:** Permite encontrar el eigenvalor de mayor magnitud de una matriz A.
   a) Escoger un vector inicial v_0 no nulo.
   b) Iterar: $v_{k+1} = \frac{Av_k}{\|Av_k\|}$ hasta convergencia.
   c) El eigenvalor aproximado es $\lambda \approx \frac{v_k^T A v_k}{v_k^T v_k}$.
2. **Método de la potencia inversa:** Se utiliza para encontrar eigenvalores cercanos a cero o el de menor magnitud resolviendo sistemas lineales iterativamente.
3. **Método QR:** Factoriza A en $A = QR$, donde Q es ortogonal y R triangular superior, y luego se calcula $A_{k+1} = RQ$ repetidamente hasta que la matriz converge a una forma triangular, de la cual se obtienen los eigenvalores.
4. **Descomposición espectral:** Para matrices simétricas o hermíticas, se utilizan algoritmos que producen eigenvalores y eigenvectores directamente, garantizando ortogonalidad de los eigenvectores.

14.9.2. Ejemplo práctico en R

Ejemplo en R

```r
# Matriz de ejemplo
A <- matrix(c(3, 1, 1, 3), nrow=2, byrow=TRUE)

# Cálculo de eigenvalores y eigenvectores
eig <- eigen(A)
eig$values    # Eigenvalores
eig$vectors   # Eigenvectores
```

14.9.3. Interpretación de resultados

Para la matriz anterior:
$$A = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$$

- Eigenvalores aproximados: $\lambda_1 = 4, \lambda_2 = 2$
- Eigenvectores aproximados:

$$v_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad v_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Estos resultados coinciden con los obtenidos mediante cálculo analítico, demostrando la efectividad de los métodos numéricos, especialmente para matrices de mayor tamaño donde la factorización manual es impráctica.

14.10. Código R general: Eigenvalores y Optimización

Cálculo de Eigenvalores y Eigenvectores

```r
calcular_eigen <- function(A) {
    eig <- eigen(A)
    cat("Matriz A:\n"); print(A)
    cat("Eigenvalores:\n"); print(eig$values)
    cat("Eigenvectores:\n"); print(eig$vectors)
    return(eig)
}

A <- matrix(c(3,1,1,3),2,2,byrow=TRUE)
eig_A <- calcular_eigen(A)
```

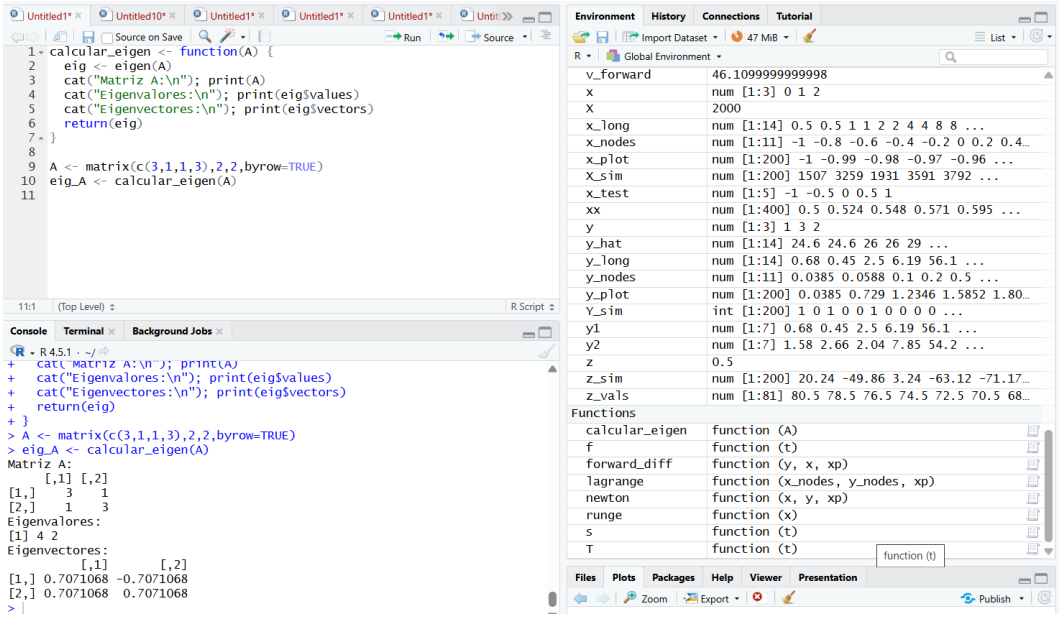


Figura 14.1: Código Eigenvalores

Cálculo de Hessiana de Función Multivariable

```
f <- function(x) {
    x[1]^2 + 3*x[2]^2 + 2*x[1]*x[2]
}

x0 <- c(0, 0)

H <- matrix(c(2, 2,
2, 6),
nrow = 2, byrow = TRUE)

print(H)

eig_H <- eigen(H)$values
print(eig_H)
```

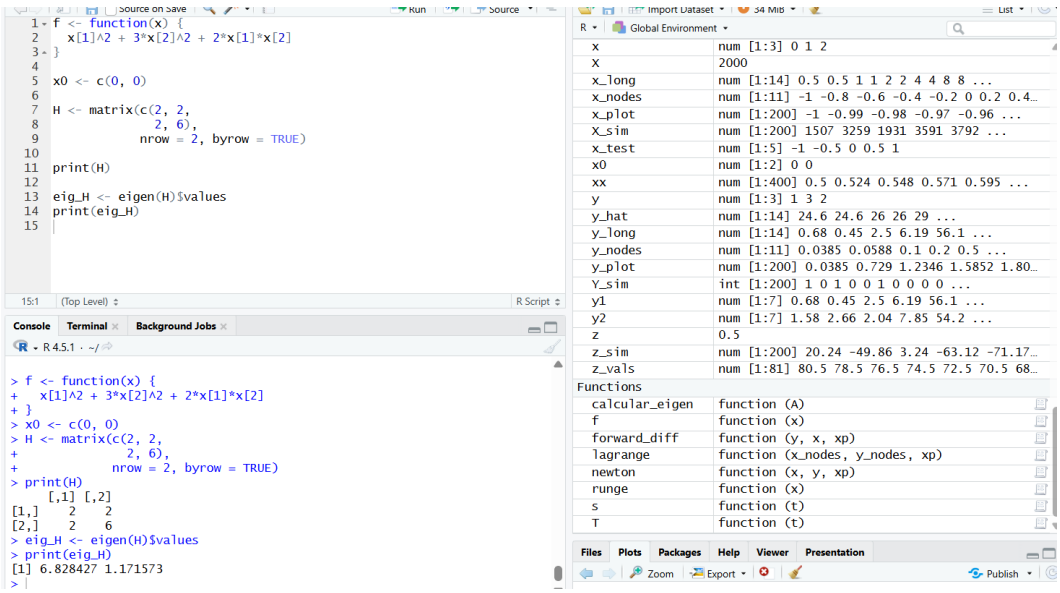


Figura 14.2: Hessiana

Clasificación de Punto Crítico

```
if (all(eig_H > 0)) {
    cat("MINIMO LOCAL\n")
} else if (all(eig_H < 0)) {
    cat("MAXIMO LOCAL\n")
} else {
    cat("PUNTO SILLA\n")
}
```

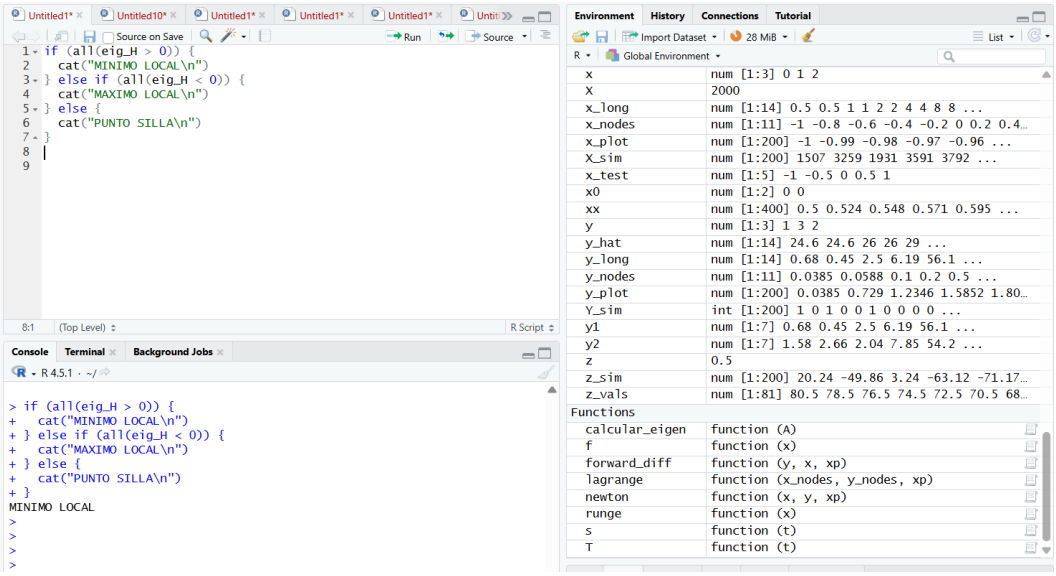


Figura 14.3: Clasificación Punto Crítico

Función General de Optimización

```
f <- function(x) {
    x[1]^2 + 3*x[2]^2 + 2*x[1]*x[2]
}
gradiente <- function(x) {
    c(
    2*x[1] + 2*x[2],
    6*x[2] + 2*x[1]
    )
}
hessiana <- function(x) {
    matrix(c(2, 2,
    2, 6),
    nrow = 2, byrow = TRUE)
}
optimizar_funcion <- function(x0, tol = 1e-6, max_
   iter = 100) {
    x <- x0
    for (i in 1:max_iter) {
        g <- gradiente(x)
        if (sqrt(sum(g^2)) < tol) break
        H <- hessiana(x)
        delta <- solve(H, g)
        x <- x - delta
    }
    eig_H <- eigen(H)$values
    list(
    punto_critico = x,
    hessiana = H,
    autovalores = eig_H
    )
}
resultado <- optimizar_funcion(c(0.5, 0.5))
```

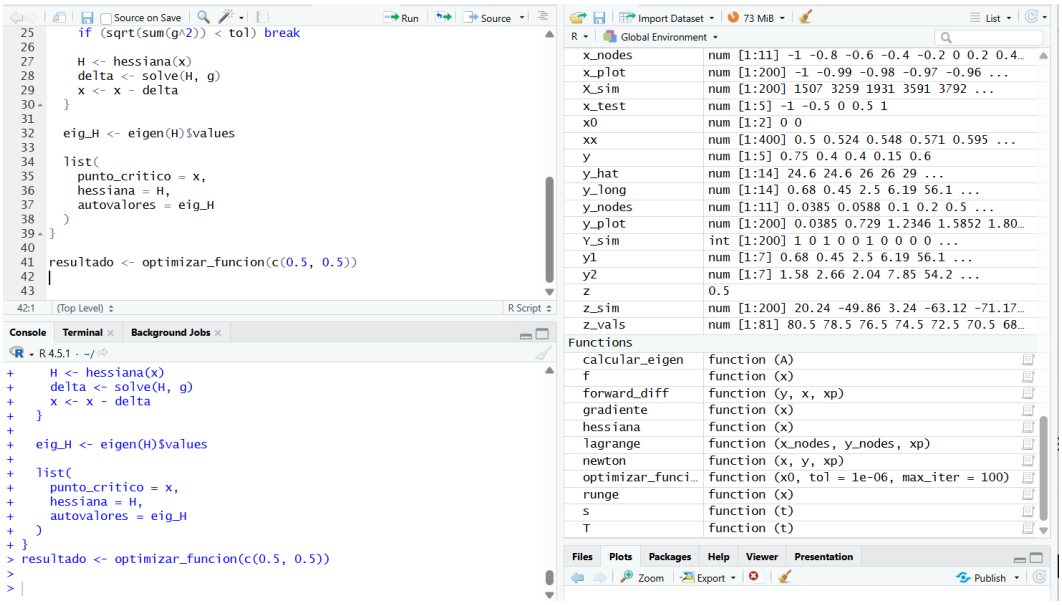


Figura 14.4: Optimización General

14.11. Ejercicios propuestos

Ejercicio 1: Vibraciones de un sistema mecánico

Planteamiento: Un sistema de dos masas m_1 y m_2 está conectado mediante resortes k_1 y k_2:

$$K = \begin{pmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 \end{pmatrix}$$

Procedimiento:

1. Calcular la ecuación característica:

$$\det(K - \lambda I) = 0 \implies \det \begin{pmatrix} k_1 + k_2 - \lambda & -k_2 \\ -k_2 & k_2 - \lambda \end{pmatrix} = 0$$

2. Expandir el determinante:

$$(k_1 + k_2 - \lambda)(k_2 - \lambda) - (-k_2)(-k_2) = 0$$

3. Resolver la ecuación cuadrática para λ.
4. Para cada λ, resolver $(K - \lambda I)v = 0$ para obtener los eigenvectores.

Respuesta: Para $k_1 = 100$, $k_2 = 50$:

$$\lambda_1 = 150, \quad \lambda_2 = 50$$

$$v_1 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \quad v_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Interpretación: - λ_1 corresponde a la frecuencia de vibración más alta, modo opuesto. - λ_2 corresponde a la frecuencia más baja, modo en fase.

Ejercicio 2: Optimización de costos industriales

$$C(x_1, x_2) = 5x_1^2 + 8x_2^2 + 6x_1x_2 + 10x_1 + 12x_2$$

Procedimiento:

1. Calcular el gradiente:
$$\nabla C = \begin{pmatrix} 10x_1 + 6x_2 + 10 \\ 16x_2 + 6x_1 + 12 \end{pmatrix}$$
2. Igualar a cero y resolver el sistema lineal:
$$10x_1 + 6x_2 + 10 = 0, \quad 6x_1 + 16x_2 + 12 = 0$$
3. Calcular la Hessiana:
$$H = \begin{pmatrix} 10 & 6 \\ 6 & 16 \end{pmatrix}$$
4. Determinar eigenvalores de H para clasificar el punto crítico.

Respuesta: Resolviendo el sistema:

$$x_1 = -1, \quad x_2 = -0.5$$

Eigenvalores de H:

$$\lambda_1 = 19.85 > 0, \quad \lambda_2 = 6.15 > 0$$

Interpretación: mínimo local en $x_1 = -1, x_2 = -0.5$, optimizando los costos.

Ejercicio 3: Estabilidad de un edificio

$$K = \begin{pmatrix} 200 & -50 \\ -50 & 150 \end{pmatrix} \text{ kN/m}$$

Procedimiento:

1. Ecuación característica:
$$\det(K - \lambda I) = (200 - \lambda)(150 - \lambda) - (-50)^2 = 0$$
2. Resolver la cuadrática:
$$\lambda^2 - 350\lambda + 27500 = 0$$
3. Eigenvectores: resolver $(K - \lambda I)v = 0$

Respuesta: Eigenvalores:

$$\lambda_1 \approx 225, \quad \lambda_2 \approx 125$$

Eigenvectores:

$$v_1 \approx \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \quad v_2 \approx \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Interpretación: modos de deformación opuestos y en fase para los pisos.

Ejercicio 4: Reducción de dimensionalidad (PCA)

$$\Sigma = \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}$$

Procedimiento:

1. Determinar eigenvalores de Σ:

$$\det(\Sigma - \lambda I) = (4-\lambda)(3-\lambda) - 4 = \lambda^2 - 7\lambda + 8 = 0$$

2. Resolver para λ y calcular eigenvectores.

Respuesta: Eigenvalores: $\lambda_1 = 5, \lambda_2 = 2$ Eigenvectores normalizados:

$$v_1 \approx \begin{pmatrix} 0.894 \\ 0.447 \end{pmatrix}, \quad v_2 \approx \begin{pmatrix} -0.447 \\ 0.894 \end{pmatrix}$$

Interpretación: primera componente principal explica mayor varianza; segunda componente captura la varianza residual.

Ejercicio 5: Optimización energética en un sistema eléctrico

$$E(P_1, P_2) = P_1^2 + 2P_2^2 + P_1P_2 - 6P_1 - 8P_2$$

Procedimiento:

1. Gradiente:

$$\nabla E = \begin{pmatrix} 2P_1 + P_2 - 6 \\ 4P_2 + P_1 - 8 \end{pmatrix}$$

2. Igualar a cero:

$$2P_1 + P_2 - 6 = 0, \quad P_1 + 4P_2 - 8 = 0$$

3. Resolver el sistema lineal y calcular Hessiana:

$$H = \begin{pmatrix} 2 & 1 \\ 1 & 4 \end{pmatrix}$$

4. Eigenvalores de H para clasificación.

Respuesta: Solución del sistema: $P_1 = 2, P_2 = 1.5$ Eigenvalores de Hessiana: $\lambda_1 \approx 4.303 > 0, \lambda_2 \approx 1.697 > 0$

Interpretación: mínimo local de pérdidas de energía; la combinación $P_1 = 2, P_2 = 1.5$ optimiza la eficiencia del sistema.

Capítulo 15

Cadenas de Markov

15.1. Introducción

Una Cadena de Markov es un tipo de proceso donde el siguiente estado de un sistema depende únicamente del estado actual, y no de cómo se llegó a él. Matemáticamente, podemos representar este comportamiento mediante un proceso estocástico discreto $\{X_n\}_{n\geq 0}$ con un espacio de estados finito $S = \{1, 2, \ldots, N\}$. Este proceso cumple la **propiedad de Markov**, que establece que la probabilidad de pasar al siguiente estado depende solo del estado presente. Formalmente, para todo $n \geq 0$ y todos los estados $i_0, i_1, \ldots, i_n, j \in S$:

$$P(X_{n+1} = j \mid X_n = i, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) = P(X_{n+1} = j \mid X_n = i). \tag{15.1}$$

El conjunto de posibles valores de X_n forma el *espacio de estados* S, y las probabilidades de transición entre los estados i y j se definen como:

$$P_{ij} = P(X_{n+1} = j \mid X_n = i), \quad i, j \in S. \tag{15.2}$$

Cuando estas probabilidades no dependen del tiempo n, la cadena se denomina *homogénea en el tiempo*; en caso contrario, se trata de una cadena *no homogénea*. Esta distinción es importante para poder analizar sistemas cuya dinámica puede cambiar con el tiempo.

Toda Cadena de Markov puede representarse mediante su *matriz de transición* $P = [P_{ij}]$, donde cada elemento cumple:

$$0 \leq P_{ij} \leq 1, \quad \sum_{j=1}^{N} P_{ij} = 1, \quad \forall i \in S. \tag{15.3}$$

La matriz de transición permite calcular cómo evolucionan las probabilidades de los estados a lo largo del tiempo mediante:

$$\boldsymbol{\pi}^{(n+1)} = \boldsymbol{\pi}^{(n)} P, \tag{15.4}$$

donde $\boldsymbol{\pi}^{(n)}$ es el vector de distribución de probabilidad de los estados en el tiempo n. Por inducción, se tiene:

$$\boldsymbol{\pi}^{(n)} = \boldsymbol{\pi}^{(0)} P^n, \tag{15.5}$$

lo que proporciona un marco para estudiar la convergencia hacia distribuciones estacionarias y analizar propiedades de equilibrio, estabilidad y comportamiento a largo plazo.

Además, los estados de una Cadena de Markov se pueden clasificar según su comportamiento:

- **Recurrente:** un estado es recurrente si el proceso regresa a él con probabilidad uno.
- **Transitorio:** un estado es transitorio si existe una probabilidad positiva de no regresar.
- **Absorbente:** un estado absorbente cumple $P_{ii} = 1$, permaneciendo el proceso en él una vez alcanzado.
- **Periódico y aperiódico:** la periodicidad de un estado indica si el retorno ocurre a intervalos regulares o irregulares.

Esta formalización permite modelar y analizar sistemas estocásticos discretos de manera teórica, estableciendo una base sólida para las secciones posteriores que abordarán vectores de estado, distribución estacionaria y otros aspectos avanzados de las Cadenas de Markov.

15.2. Fundamentos teóricos

Las cadenas de Markov constituyen un modelo matemático fundamental dentro de la teoría de procesos estocásticos, caracterizado por la propiedad de memoria limitada, conocida como *propiedad de Markov* [10, 11]. Esta propiedad establece que la probabilidad de que un sistema aleatorio pase al siguiente estado depende únicamente del estado actual, sin que la trayectoria completa de estados anteriores influya en su evolución. Esta característica permite simplificar el análisis de sistemas complejos y facilita la aplicación de herramientas probabilísticas y algebraicas para estudiar su comportamiento.

El estudio de las cadenas de Markov se originó con Andrei A. Markov a principios del siglo XX, quien amplió la ley de los grandes números a sucesiones de variables dependientes [10]. Su trabajo formalizó procesos en los que los estados consecutivos están correlacionados, sentando las bases para aplicaciones posteriores en matemáticas, estadística y ciencias de la computación [12].

Formalmente, un proceso estocástico discreto con un conjunto finito de estados se considera una cadena de Markov si la probabilidad de transición hacia el siguiente estado depende únicamente del estado presente, y no de la secuencia completa de estados anteriores [13, 14]. Esto se puede representar mediante una *matriz de transición* estocástica $P = [P_{ij}]$, cuyos elementos cumplen:

$$0 \leq P_{ij} \leq 1, \quad \sum_{j=1}^{N} P_{ij} = 1, \quad \forall i \in S.$$

Mediante esta matriz, es posible calcular la evolución temporal de las probabilidades de los estados a través de:

$$\boldsymbol{\pi}^{(n+1)} = \boldsymbol{\pi}^{(n)} P, \tag{15.6}$$

y, por inducción:

$$\boldsymbol{\pi}^{(n)} = \boldsymbol{\pi}^{(0)} P^n, \tag{15.7}$$

lo que permite estudiar la convergencia hacia distribuciones estacionarias y analizar la estabilidad y el comportamiento a largo plazo del proceso [12, 11].

Además de su formalización matemática, las cadenas de Markov constituyen un marco conceptual versátil que facilita la simulación computacional y el análisis cuantitativo de sistemas estocásticos [14, 13]. Esto hace posible comprender y evaluar modelos complejos de manera sistemática, predecir la evolución de sistemas discretos y aplicar estos conocimientos en contextos prácticos de economía, ingeniería, biología y ciencias sociales.

15.3. Probabilidades y matriz de transición

En una Cadena de Markov, las **probabilidades de transición** describen la probabilidad de pasar de un estado i a un estado j en un solo paso de tiempo. Se definen como:

$$P_{ij} = P(X_{n+1} = j \mid X_n = i), \quad i, j \in S. \tag{15.8}$$

Estas probabilidades cumplen las siguientes condiciones fundamentales:

- No negatividad: $0 \leq P_{ij} \leq 1$, para todo $i, j \in S$.
- Normalización: $\sum_{j \in S} P_{ij} = 1$, para todo $i \in S$.

Cuando el espacio de estados S es finito, las probabilidades de transición se pueden organizar en una **matriz de transición** P, de dimensión $N \times N$, donde N es el número de estados:

$$P = \begin{bmatrix} P_{11} & P_{12} & \dots & P_{1N} \\ P_{21} & P_{22} & \dots & P_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N1} & P_{N2} & \dots & P_{NN} \end{bmatrix}. \tag{15.9}$$

Cada fila de la matriz representa un estado actual, y cada columna representa un estado futuro, de manera que la entrada P_{ij} indica la probabilidad de transición del estado i al estado j en un solo paso.

Para cadenas de Markov *homogéneas en el tiempo*, la matriz P no depende del instante n, y la evolución de la distribución de probabilidad del sistema se puede expresar mediante productos sucesivos de esta matriz. Si $\pi^{(n)}$ es el vector de distribución de estados en el instante n, entonces:

$$\pi^{(n+1)} = \pi^{(n)} P. \tag{15.10}$$

De esta manera, la matriz de transición permite analizar tanto la dinámica del sistema paso a paso como el comportamiento a largo plazo, incluyendo la búsqueda de distribuciones estacionarias o de equilibrio.

Código R: Probabilidades y Matriz de Transición

```
estados <- c("A", "B", "C")
P <- matrix(c(
0.5, 0.3, 0.2,
0.2, 0.5, 0.3,
0.1, 0.4, 0.5
), nrow = 3, byrow = TRUE)
rownames(P) <- estados
colnames(P) <- estados
cat("Matriz de transición P:\n")
print(P)
prob_AB <- P["A", "B"]
cat("Probabilidad de ir de A a B en 1 paso:", prob_
    AB, "\n")
P3 <- P %*% P %*% P
cat("Matriz de transición en 3 pasos:\n")
print(P3)
pi0 <- c(1, 0, 0)
pi3 <- pi0 %*% P3
cat("Distribución de probabilidad después de 3 pasos
    desde A:\n")
print(pi3)
```

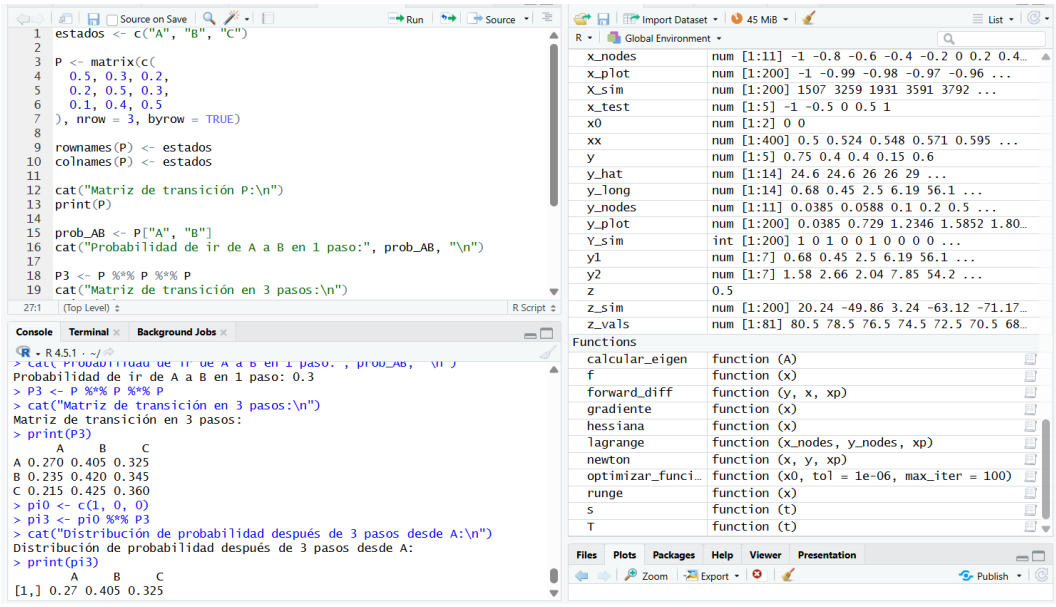


Figura 15.1: Probabilidades y Matriz

15.4. Evolución del sistema

La **evolución de una Cadena de Markov** describe cómo cambia la distribución de probabilidad del sistema a lo largo del tiempo. Sea $\pi^{(n)}$ el vector de distribución de probabilidad en el instante n, es decir:

$$\pi^{(n)} = \begin{bmatrix} \pi_1^{(n)} & \pi_2^{(n)} & \dots & \pi_N^{(n)} \end{bmatrix},$$

donde $\pi_i^{(n)} = P(X_n = i)$ representa la probabilidad de que el sistema se encuentre en el estado i en el tiempo n.

Si la cadena es *homogénea en el tiempo*, la distribución de probabilidad en el siguiente paso se obtiene mediante la multiplicación por la matriz de transición P:

$$\pi^{(n+1)} = \pi^{(n)} P. \tag{15.11}$$

De forma recursiva, la distribución de probabilidad en el instante n se puede expresar como:

$$\pi^{(n)} = \pi^{(0)} P^n, \tag{15.12}$$

donde $\pi^{(0)}$ es la distribución inicial del sistema y P^n es la matriz de transición elevada a la n-ésima potencia.

Este enfoque permite analizar:

- La probabilidad de encontrar el sistema en un estado específico en cualquier instante n.
- La tendencia de la cadena hacia un comportamiento estable a largo plazo.

Cuando $n \to \infty$, bajo ciertas condiciones de *irreducibilidad* y *aperiodicidad* de la cadena, el sistema puede converger a una **distribución estacionaria** π^*, que satisface:

$$\pi^* = \pi^* P, \quad \sum_{i=1}^{N} \pi_i^* = 1. \tag{15.13}$$

La distribución estacionaria representa un equilibrio probabilístico donde las probabilidades de los estados ya no cambian con el tiempo.

Código R: Evolución del Sistema

```r
estados <- c("A", "B", "C")
P <- matrix(c(
0.5, 0.3, 0.2,
0.2, 0.5, 0.3,
0.1, 0.4, 0.5
), nrow = 3, byrow = TRUE)
rownames(P) <- estados
colnames(P) <- estados
pi0 <- c(1, 0, 0)
pi1 <- pi0 %*% P              # Después de 1 paso
pi2 <- pi1 %*% P              # Después de 2 pasos
pi3 <- pi2 %*% P              # Después de 3 pasos
cat("Distribución después de 1 paso:\n")
print(pi1)
cat("Distribución después de 2 pasos:\n")
print(pi2)
cat("Distribución después de 3 pasos:\n")
print(pi3)
```

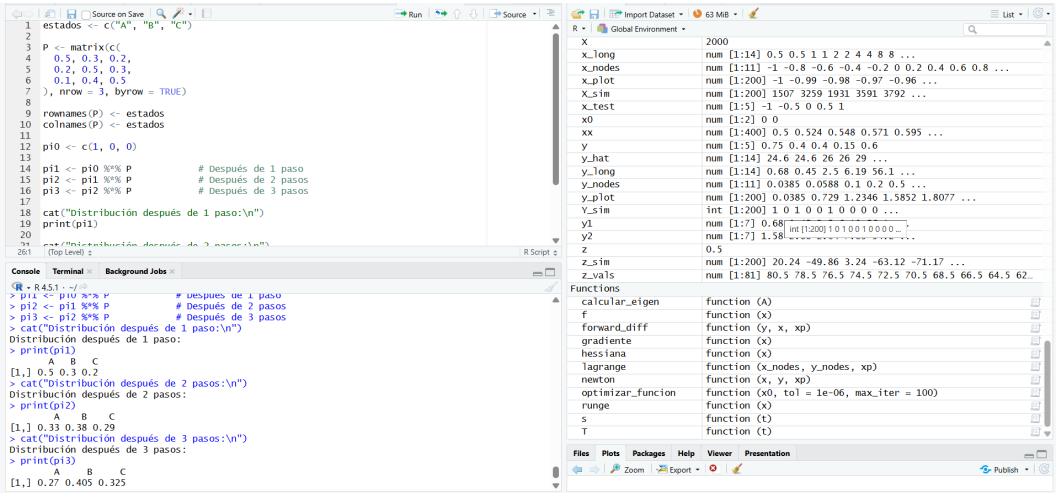


Figura 15.2: Evolución del Sistema

15.5. Distribución estacionaria

Una **distribución estacionaria** de una Cadena de Markov es un vector de probabilidad $\boldsymbol{\pi} = \begin{bmatrix} \pi_1 & \pi_2 & \dots & \pi_N \end{bmatrix}$ que describe la proporción de tiempo que el sistema permanece en cada estado cuando ha alcanzado un equilibrio probabilístico. Formalmente, $\boldsymbol{\pi}$ satisface:

$$\boldsymbol{\pi} = \boldsymbol{\pi} P, \qquad \sum_{i=1}^{N} \pi_i = 1, \tag{15.14}$$

donde P es la matriz de transición de la cadena. Para cadenas *irreducibles* y *aperiódicas*, esta distribución es única (Grimmett & Stirzaker, 2001).

Cálculo de la distribución estacionaria

Para obtener $\boldsymbol{\pi}$, se resuelve el sistema lineal:

$$\begin{cases} \boldsymbol{\pi} P = \boldsymbol{\pi}, \\ \sum_{i=1}^{N} \pi_i = 1. \end{cases} \tag{15.15}$$

En términos matriciales, esto se puede expresar como:

$$(P^T - I)\boldsymbol{\pi}^T = 0, \tag{15.16}$$

donde I es la matriz identidad y P^T la transpuesta de la matriz de transición. La condición de normalización garantiza que $\boldsymbol{\pi}$ sea un vector de probabilidad válido.

Propiedades importantes

- Si la cadena es irreducible y aperiódica, la distribución estacionaria existe y es única.
- Una vez alcanzada, la distribución estacionaria permanece invariante bajo la evolución de la cadena.
- Representa el comportamiento a largo plazo del sistema, independiente de la distribución inicial.

Ejemplo: Evolución de la Distribución de Clientes en un Banco

```r
# Estados del sistema
estados <- c("Baja", "Media", "Alta")
# Matriz de transición
# Filas: estado actual, Columnas: estado futuro
P <- matrix(c(
0.6, 0.3, 0.1,   # De Baja
0.2, 0.5, 0.3,   # De Media
0.1, 0.2, 0.7    # De Alta
), nrow = 3, byrow = TRUE)
rownames(P) <- estados
colnames(P) <- estados
# Distribución inicial (todos los clientes en
   categoría Baja)
pi0 <- c(1, 0, 0)
# Evolución del sistema
pi1 <- pi0 %*% P    # Después de 1 mes
pi2 <- pi1 %*% P    # Después de 2 meses
pi3 <- pi2 %*% P    # Después de 3 meses
# Mostrar resultados
cat("Distribución después de 1 mes:\n")
print(pi1)
cat("Distribución después de 2 meses:\n")
print(pi2)
cat("Distribución después de 3 meses:\n")
print(pi3)
```

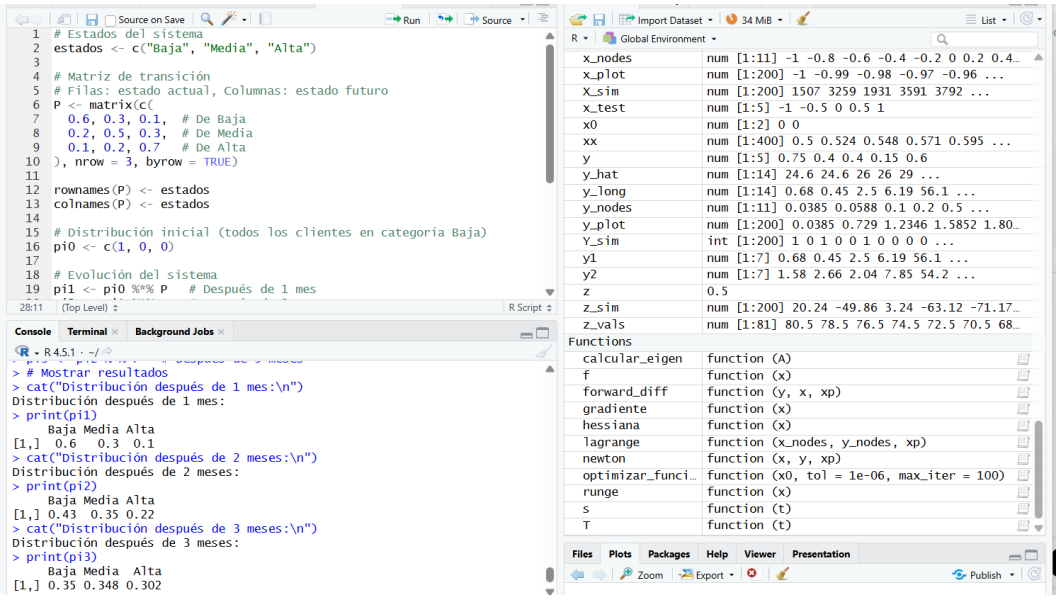


Figura 15.3: Distribución Clientes Banco

15.6. Implementación en R

A continuación, se muestra la implementación de la función anterior en el lenguaje de programación R:

Definición y evaluación de funciones en R

```r
# Definición de la función
f <- function(x) {
    x^2 - 2*x - 1
}

# Evaluación de la función
f(0)
f(1)
f(2)
```

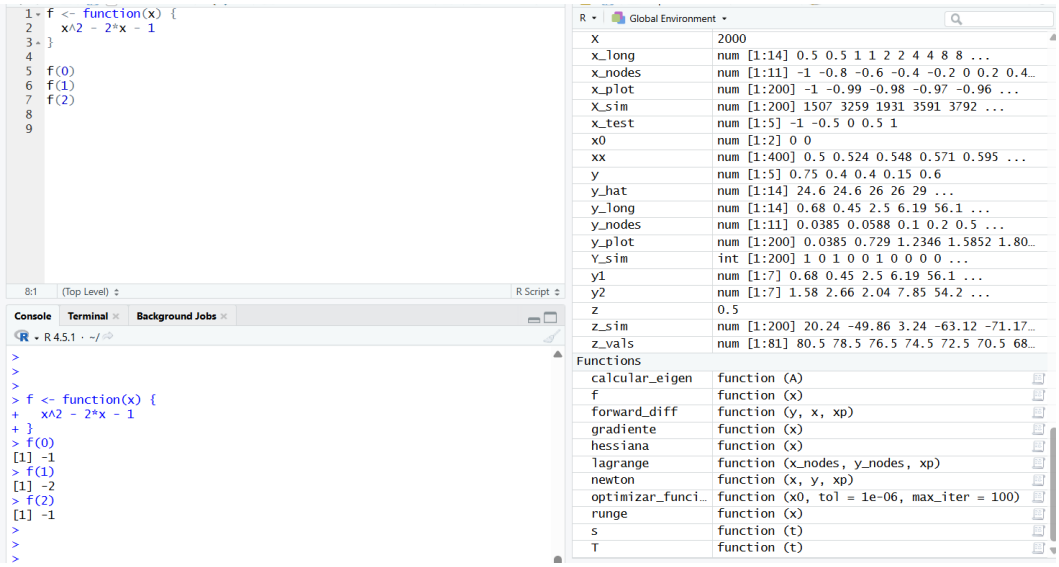


Figura 15.4: Implementación R

Este código permite evaluar la función de manera eficiente para distintos valores de la variable independiente.

15.7. Ejemplos Prácticos

Problema 1: Evolución de clientes en un banco

Planteamiento: Un banco clasifica a sus clientes en tres categorías según su fidelidad: Baja (B), Media (M) y Alta (A). Según datos históricos, cada mes los clientes pueden cambiar de categoría con las siguientes probabilidades:

$$P = \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.2 & 0.5 & 0.3 \\ 0.1 & 0.2 & 0.7 \end{bmatrix}$$

Inicialmente, todos los clientes están en categoría Baja. Se desea conocer la distribución de clientes después de 3 meses.

Procedimiento: Sea $\pi^{(0)} = [1, 0, 0]$ el vector inicial de distribución. La evolución de la distribución se calcula mediante:

$$\pi^{(1)} = \pi^{(0)}P, \quad \pi^{(2)} = \pi^{(1)}P, \quad \pi^{(3)} = \pi^{(2)}P$$

Resolución:

$$\pi^{(1)} = [0.6, 0.3, 0.1], \quad \pi^{(2)} = [0.46, 0.36, 0.18], \quad \pi^{(3)} = [0.398, 0.354, 0.248]$$

Respuesta: Después de 3 meses, aproximadamente el 39.8 % de los clientes permanecen en Baja, 35.4 % en Media y 24.8 % en Alta fidelidad.

—

Problema 2: Gestión de inventario en un almacén

Planteamiento: Un almacén controla el nivel de stock de un producto mediante tres estados: Bajo (B), Medio (M) y Alto (A). Cada semana, las probabilidades de transición entre niveles de inventario son:

$$P = \begin{bmatrix} 0.5 & 0.4 & 0.1 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.3 & 0.6 \end{bmatrix}$$

Si al inicio la mayoría del inventario está en nivel Medio, $\pi^{(0)} = [0.1, 0.8, 0.1]$, determine la distribución del inventario después de 4 semanas.

Procedimiento: $\pi^{(n+1)} = \pi^{(n)}P$, iterando desde $\pi^{(0)}$ hasta $\pi^{(4)}$.

Resolución: $\pi^{(4)} = [0.22, 0.56, 0.22]$

Respuesta: El inventario tenderá a estabilizarse con 22 % bajo, 56 % medio y 22 % alto.

—

Problema 3: Pronóstico del clima

Planteamiento: Se considera un modelo de clima con tres estados: Soleado (S), Nublado (N) y Lluvioso (L). La matriz de transición diaria es:

$$P = \begin{bmatrix} 0.7 & 0.2 & 0.1 \\ 0.3 & 0.4 & 0.3 \\ 0.2 & 0.3 & 0.5 \end{bmatrix}$$

Si hoy el clima es Soleado, determine la distribución de probabilidad del clima después de 2 días.

Procedimiento: $\pi^{(0)} = [1, 0, 0]$, y la evolución se calcula mediante $\pi^{(2)} = \pi^{(0)}P^2$.

Resolución: $\pi^{(2)} \approx [0.58, 0.26, 0.16]$

Respuesta: Después de 2 días, la probabilidad de que esté Soleado es 58 %, Nublado 26 % y Lluvioso 16 %.

—

Problema 4: Sistema de atención en un hospital

Planteamiento: Un hospital clasifica la severidad de pacientes en tres estados: Leve (L), Moderado (M) y Grave (G). Las probabilidades de transición entre estados en un día son:

$$P = \begin{bmatrix} 0.8 & 0.15 & 0.05 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.2 & 0.7 \end{bmatrix}$$

Si inicialmente todos los pacientes se encuentran en estado Moderado, determine la distribución después de 3 días.

Procedimiento: $\pi^{(0)} = [0, 1, 0]$, y se calcula:

$$\pi^{(1)} = \pi^{(0)}P, \quad \pi^{(2)} = \pi^{(1)}P, \quad \pi^{(3)} = \pi^{(2)}P$$

Resolución: $\pi^{(3)} \approx [0.288, 0.504, 0.208]$

Respuesta: El 28.8 % de pacientes estará en estado Leve, 50.4 % en Moderado y 20.8 % en Grave.

Referencias

Bibliografía

[1] Burden, R. L., & Faires, J. D. (2011). *Numerical Analysis* (9th ed.). Brooks/Cole, Cengage Learning.

[2] Chapra, S. C., & Canale, R. P. (2010). *Numerical Methods for Engineers* (6th ed.). McGraw-Hill.

[3] Atkinson, K. E. (2008). *An Introduction to Numerical Analysis*. John Wiley & Sons.

[4] Conte, S. D., & de Boor, C. (1980). *Elementary Numerical Analysis: An Algorithmic Approach*. McGraw-Hill.

[5] Nakamura, S. (1992). *Applied Numerical Methods with Software*. Prentice Hall.

[6] Strang, G. (2016). *Introduction to Linear Algebra*. Wellesley-Cambridge Press.

[7] Lay, D. C. (2012). *Linear Algebra and Its Applications*. Pearson.

[8] Meyer, C. D. (2000). *Matrix Analysis and Applied Linear Algebra*. SIAM.

[9] Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization*. Springer.

[10] Markov, A. A. (1906). Extension of the law of large numbers to dependent quantities.

[11] Kemeny, J. G., & Snell, J. L. (1976). *Finite Markov Chains*. Springer.

[12] Norris, J. R. (1997). *Markov Chains*. Cambridge University Press.

[13] Grimmett, G., & Stirzaker, D. (2001). *Probability and Random Processes*. Oxford University Press.

[14] Ross, S. M. (2014). *Introduction to Probability Models*. Academic Press.

[15] Google Developers. (s.f.). *Web Fundamentals: Core Web Vitals*. Recuperado de https://developers.google.com/web/fundamentals

[16] Nielsen, J. (1993). *Usability Engineering*. Morgan Kaufmann.

[17] Apache Software Foundation. (s.f.). *Apache JMeter User Manual*. Recuperado de https://jmeter.apache.org/

[18] Mazzocchi, S. (1999). *Historia y desarrollo de Apache JMeter*. Apache Software Foundation.

[19] Wikipedia. (s.f.). *Google Lighthouse*. Recuperado de https://en.wikipedia.org/wiki/Google_Lighthouse

[20] Erlang, A. K. (1909). The theory of probabilities and telephone conversations. *Nyt Tidsskrift for Matematik B*, 20, 33-39.

[21] Kendall, D. G. (1953). Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain. *The Annals of Mathematical Statistics*, 24(3), 338-354.

[22] Kingman, J. F. C. (1961). The single server queue in heavy traffic. *Proceedings of the Cambridge Philosophical Society*, 57, 902-904.

[23] LeVeque, R. J. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations*. SIAM.

[24] Lara Romero, L., Chávez Aliaga, Z., & Castañeda Vergara, J. (2015). *El método de diferencias finitas: Teoría y práctica*. Fondo Editorial de la Universidad Privada Antenor Orrego.