Compás
capacitación e investigación

2025

Cálculo fácil con Python:
guía de prácticas y ejemplos

Bladimir Homero Serrano Rugel
Nathaly Ragde Riofrío Romero

Cálculo fácil con Python:
guía de prácticas y ejemplos

Bladimir Homero Serrano Rugel
Nathaly Ragde Riofrío Romero

ISBN: 978-9942-53-054-7 DOI: http://doi.org/10.48190/9789942530547

© **Bladimir Homero Serrano Rugel**
Correo: bserrano@utmachala.edu.ec
Filial: Universidad Técnica de Machala
https://orcid.org/0000-0001-6859-5563

Nathaly Ragde Riofrío Romero
Correo: nriofrio@utmachala.edu.ec
Filial: Universidad Técnica de Machala
https://orcid.org/0000-0002-2332-824X

Primera edición, 2025-11-08

ISBN: 978-9942-53-054-7
DOI: http://doi.org/10.48190/9789942530547
Distribución online
Acceso abierto

Cita
Serrano, B., Riofrío, N. (2025) Cálculo Fácil con Python: Guía de Prácticas y Ejemplos. Integrando Python en el Aprendizaje de Cálculo para Ciencias Aplicadas. Editorial Grupo Compás

Reseña de los Autores

Bladimir Homero Serrano Rugel

Máster Universitario en Ingeniería Computacional y Matemática

Mi vocación docente comenzó en mayo de 2002 en la Unidad Educativa Particular La Inmaculada, donde durante más de una década formé a estudiantes de educación media, transmitiendo no solo conocimientos técnicos, sino también valores fundamentales para su desarrollo personal y profesional.

Desde mayo de 2014, desempeño funciones como docente en la Universidad Técnica de Machala, donde contribuyo a la formación de futuros ingenieros y científicos. En esta etapa, he combinado la enseñanza de fundamentos teóricos con herramientas computacionales modernas como Python, promoviendo un aprendizaje activo, práctico y orientado a la resolución de problemas reales.

Mi compromiso con la excelencia académica se refleja en mi formación continua, que culminó con la obtención del título de Máster en Ingeniería Computacional y Matemática por la Universitat Rovira i Virgili. Como investigador,

he participado en proyectos relacionados con simulación de procesos, aplicaciones de la física conceptual y control de calidad en industrias locales. He publicado artículos científicos y colaborado en obras colectivas de investigación.

A lo largo de mi carrera, he mantenido un equilibrio entre la docencia, la investigación aplicada y la innovación tecnológica, consolidando un perfil profesional que integra el pensamiento matemático riguroso con el uso creativo de las tecnologías de la información.

Nathaly Ragde Riofrío Romero

Ingeniera en Producción y Operaciones, Magíster en Gestión de Operaciones

Cuento con una trayectoria de más de diez años en el ámbito académico, desempeñándome como docente en instituciones de educación superior. Mi labor se ha centrado en áreas clave como matemáticas, estadística e investigación de operaciones, contribuyendo activamente a la formación de profesionales con pensamiento analítico y enfoque operativo.

A lo largo de mi carrera, he demostrado habilidades destacadas en el levantamiento y gestión de procesos, aportando a la mejora continua y eficiencia de las estructuras organizacionales. Mi perfil combina una sólida formación técnica con un firme compromiso con la educación, lo que me

posiciona como una profesional íntegra, orientada a la excelencia académica y al desarrollo de soluciones prácticas en entornos complejos.

Prefacio

Este libro ha sido concebido con la intención de transformar el paradigma del aprendizaje de las matemáticas en la carrera de Acuicultura, donde tradicionalmente se ha trabajado con enfoques que producen estudiantes mecánicos, centrados en la repetición y no en la comprensión profunda. En este contexto, se propone un enfoque basado en el razonamiento matemático, integrando la programación como herramienta pedagógica. Las clases se matizan con la resolución de problemas contextualizados en el cálculo diferencial e integral, utilizando el lenguaje de programación Python como puente entre la teoría matemática y su aplicación en situaciones reales del ámbito acuícola.

Objetivo General

Proporcionar a los estudiantes una guía que les permita comprender y aplicar los conceptos fundamentales del cálculo diferencial e integral, utilizando el lenguaje de programación Python como medio para la visualización, experimentación y resolución de problemas reales relacionados

con su campo de estudio.

Estructura del Libro

El contenido del libro está dividido en capítulos que abordan temas centrales del cálculo, integrando teoría esencial con ejercicios resueltos en Python:

- Introducción al entorno de trabajo en Python (Google Colab, Jupyter Notebook).
- Derivadas: interpretación gráfica, reglas de derivación, y aplicaciones en modelos simples.
- Integrales: conceptos básicos, integración numérica, y cálculo de áreas.
- Aplicaciones prácticas en problemas vinculados a la biología, acuicultura y producción agropecuaria.

Cada capítulo incluye ejemplos detallados, ejercicios propuestos, y un laboratorio con código comentado para facilitar el aprendizaje autónomo.

Características Pedagógicas

Este libro incorpora una metodología activa, centrada en el estudiante, con las siguientes características:

- Uso de Python como lenguaje accesible para la exploración de conceptos matemáticos.

- Explicaciones claras y ejemplos contextualizados en el área agropecuaria.
- Códigos ejecutables y modificables que permiten experimentar y aprender haciendo.
- Diseño modular que facilita su uso en cursos semestrales, talleres o autoaprendizaje.

Esperamos que esta obra sirva como puente entre la teoría matemática y su aplicación práctica, motivando a los estudiantes a desarrollar competencias analíticas y tecnológicas que contribuyan a su formación integral y profesional.

Índice general

1 Introducción al Mundo de Python

Objetivos específicos del capítulo

- Comprender la evolución, características y ventajas del lenguaje Python como herramienta educativa en ciencias aplicadas.
- Identificar las principales estructuras de datos en Python y su relevancia en el desarrollo de algoritmos matemáticos.
- Instalar y configurar el entorno de desarrollo Python, explorando sus herramientas básicas de ejecución de código.
- Utilizar bibliotecas fundamentales como NumPy, SymPy y Matplotlib para representar funciones matemáticas y resolver problemas básicos de cálculo.

El **Cálculo Diferencial e Integral** ha sido una de las áreas más influyentes en el desarrollo de la ciencia y la tecnología. Su impacto se extiende desde la física y la ingeniería hasta la economía y la inteligencia artificial. No obstante,

su enseñanza ha sido tradicionalmente teórica, basada en la manipulación algebraica de ecuaciones y la demostración rigurosa de teoremas. Si bien este enfoque es fundamental para el desarrollo del pensamiento matemático, la creciente digitalización del conocimiento y la expansión de la computación científica han generado la necesidad de complementar el aprendizaje del cálculo con herramientas computacionales que permitan visualizar, simular y resolver problemas de manera más efectiva (Diaz, 2020).

El **razonamiento matemático** es la base del pensamiento abstracto y analítico en las matemáticas. Consiste en la capacidad de formular conjeturas, estructurar argumentos lógicos y demostrar propiedades de los objetos matemáticos. En el contexto del cálculo, esto implica comprender los conceptos de límite, derivada e integral, así como sus aplicaciones en la modelización de fenómenos naturales y artificiales. Sin embargo, en el mundo actual, donde el volumen de datos y la complejidad de los problemas crecen exponencialmente, es esencial desarrollar también el **razonamiento computacional matemático**, el cual permite abordar problemas desde una perspectiva algorítmica y programática (Martinez, 2019).

El razonamiento computacional en matemáticas involucra el uso de algoritmos, estructuras de datos y simulaciones numéricas para analizar y resolver problemas de cálculo de manera eficiente. Con herramientas como `Python`, los estudiantes pueden realizar cálculos simbólicos y numéri-

cos con bibliotecas especializadas como `SymPy`, `NumPy` y `SciPy`. Esto les permite no solo verificar resultados obtenidos analíticamente, sino también experimentar con problemas que serían demasiado complejos para resolver manualmente. Además, la representación gráfica de funciones y soluciones de ecuaciones diferenciales proporciona una comprensión visual de los conceptos matemáticos, facilitando su interpretación (Perez, 2021).

El uso de un **laboratorio computacional** en la enseñanza del cálculo presenta múltiples beneficios. En primer lugar, permite a los estudiantes interactuar con los conceptos matemáticos de manera dinámica, lo que fomenta un aprendizaje más significativo. En segundo lugar, proporciona herramientas para la resolución de problemas del mundo real, como la optimización de recursos, el modelado de sistemas físicos y el análisis de datos en distintas disciplinas. Finalmente, promueve una formación interdisciplinaria, integrando conocimientos matemáticos con habilidades de programación y computación científica, competencias altamente valoradas en el ámbito académico y profesional (Gomez, 2022).

Python fue desarrollado por Guido van Rossum en 1991 como un lenguaje de programación interpretado, interactivo y orientado a objetos (Van Rossum, 1991). Su diseño enfatiza la legibilidad del código y la simplicidad, convirtiéndolo en una herramienta ideal tanto para principiantes como para expertos.

1.0.1 Características principales

Python destaca por las siguientes características (Lutz, 2013; Nelli, 2015):

- Es multiplataforma: compatible con Windows, macOS y Linux.
- Soporte para programación estructurada, funcional y orientada a objetos.
- Extensibilidad mediante bibliotecas y módulos.
- Comunidad activa que mantiene y mejora el lenguaje constantemente.

1.1 Instalación y configuraciones iniciales

Instalar Python es sencillo y puede realizarse desde su sitio oficial (Python Software Foundation, 2025). Se recomienda usar distribuciones como Anaconda para entornos científicos (Harrington, 2016).

1. **Descarga**: Accede al sitio oficial de Anaconda en `https://www.anaconda.com/` y selecciona la versión de Python 3 para tu sistema operativo (Windows, macOS o Linux).
2. **Ejecución del instalador**: Descarga el archivo y ejecútalo. Sigue las instrucciones del asistente de instalación. Se recomienda instalar Anaconda para un

usuario específico y no requerir privilegios de administrador.

3. **Configuración del entorno**: Durante la instalación, selecciona la opción para añadir Anaconda a las variables de entorno (PATH). Esto facilita el uso de Python desde la terminal.

4. **Verificación**: Una vez instalado, abre la terminal o el símbolo del sistema y escribe:

```
conda --version
python --version
```

1.1: Verificación de la instalación de Anaconda

   Si los comandos muestran las versiones de Conda y Python, la instalación fue exitosa.

5. **Actualización**: Asegúrate de tener las últimas versiones de Anaconda y sus paquetes escribiendo:

```
conda update conda
conda update anaconda
```

1.2: Actualización de Anaconda

Con Anaconda instalado, puedes usar herramientas como Jupyter Notebook y Spyder para proyectos científicos y educativos.

1.1.1 Primeros pasos con Python

El intérprete de Python permite ejecutar comandos interactivos. Por ejemplo:

```
1
2               print("Hola, mundo!")
```

1.3: Ejemplo: Hola Mundo en Python

1.2 Bibliotecas clave para matemáticas y visualización

Python cuenta con bibliotecas especializadas que potencian su uso en matemáticas y ciencias. Entre ellas destacan:

- `NumPy`: Manipulación de matrices y álgebra lineal (Oliphant, 2006).
- `matplotlib`: Creación de gráficos y visualizaciones (Hunter, 2007).
- `SymPy`: Cálculo simbólico (Meurer et al., 2017).

1.3 Python en educación y ciencia

En la actualidad, el lenguaje de programación **Python** se ha convertido en una herramienta fundamental en la enseñanza de las matemáticas y la ciencia. Su sintaxis clara y

su flexibilidad lo hacen ideal para estudiantes y docentes que desean aplicar conceptos matemáticos y científicos sin la complejidad de lenguajes más técnicos. Python no solo es utilizado en el ámbito académico, sino que también es ampliamente empleado en la investigación científica, la ingeniería y la inteligencia artificial (VanRossum, 2009).

1.4 Python como herramienta educativa

La enseñanza de las matemáticas tradicionalmente se ha basado en métodos analíticos y algebraicos, lo que puede resultar abstracto para muchos estudiantes. La incorporación de Python en el aula permite una aproximación más intuitiva y visual a los conceptos matemáticos. Gracias a bibliotecas como `Matplotlib` y `SymPy`, los alumnos pueden representar gráficamente funciones, calcular derivadas e integrales simbólicamente y experimentar con modelos matemáticos interactivos (Lutz, 2013).

Además, Python fomenta el aprendizaje basado en proyectos, un enfoque pedagógico en el que los estudiantes aplican sus conocimientos para resolver problemas reales. Este método ha demostrado mejorar la comprensión y retención del conocimiento, ya que los alumnos pueden ver la utilidad práctica de las matemáticas en la resolución de problemas del mundo real (Grus, 2019).

1.5 Aplicaciones científicas de Python

En el ámbito de la ciencia y la ingeniería, Python se ha consolidado como una herramienta indispensable para la modelización matemática, el análisis de datos y la simulación de sistemas complejos. Gracias a bibliotecas como `NumPy`, `SciPy` y `Pandas`, los científicos pueden procesar grandes volúmenes de datos y realizar cálculos avanzados con facilidad (Oliphant, 2015).
En la física, por ejemplo, Python es utilizado para resolver ecuaciones diferenciales que modelan fenómenos naturales como el movimiento de partículas, la propagación de ondas y la dinámica de fluidos. En biología, se emplea para analizar secuencias genéticas y modelar procesos bioquímicos. En economía y finanzas, es una herramienta clave para el análisis de series temporales y la predicción de mercados (McKinney, 2017).

1.5.1 Python en la inteligencia artificial y el aprendizaje automático

El auge del aprendizaje automático y la inteligencia artificial ha impulsado aún más la popularidad de Python. Bibliotecas como `TensorFlow` y `scikit-learn` han permitido que investigadores y desarrolladores construyan modelos de aprendizaje profundo para el reconocimiento de imágenes, el procesamiento de lenguaje natural y la toma de decisiones automatizadas (Goodfellow, 2016).

En la educación, estos avances han llevado al desarrollo de sistemas de tutoría inteligentes que personalizan el aprendizaje según el progreso del estudiante. Los algoritmos de aprendizaje automático pueden analizar patrones en los datos de los estudiantes y proporcionar retroalimentación adaptativa, mejorando significativamente la enseñanza personalizada (Russell, 2020).

1.6 Ejemplo práctico: Gráfico de una función

A continuación, se muestra un ejemplo para graficar $f(x) = x^2$ usando `matplotlib`:

```
1
2               import matplotlib.pyplot as plt
3               import numpy as np
4
5               x = np.linspace(-10, 10, 100)
6               y = x**2
7
8               plt.plot(x, y, label='f(x) = x^2')
9               plt.xlabel('x')
10              plt.ylabel('f(x)')
11              plt.title('Gráfico de la función cuadrática')
12              plt.legend()
13              plt.grid()
14              plt.show()
```

1.4: Gráfico de una función cuadrática

1.7 Introducción al Mundo de Python

Python es un lenguaje de programación poderoso y versátil que ha ganado popularidad en una amplia variedad de disciplinas, incluyendo el cálculo diferencial e integral. Este capítulo proporciona una introducción detallada al lenguaje, destacando sus características principales y su aplicabilidad en contextos científicos y educativos.

1.8 Historia y características de Python

Python fue desarrollado por Guido van Rossum en 1991 como un lenguaje de programación interpretado, interactivo y orientado a objetos (Van Rossum, 1991). Su diseño enfatiza la legibilidad del código y la simplicidad, convirtiéndolo en una herramienta ideal tanto para principiantes como para expertos.

1.8.1 Características principales

Python destaca por las siguientes características (Lutz, 2013; Nelli, 2015):

- Es multiplataforma: compatible con Windows, macOS y Linux.
- Soporte para programación estructurada, funcional y orientada a objetos.
- Extensibilidad mediante bibliotecas y módulos.

- Comunidad activa que mantiene y mejora el lenguaje constantemente.

1.8.2 Importación de bibliotecas y funciones

En Python, puedes importar bibliotecas y funciones para ampliar la funcionalidad básica del lenguaje. Existen varias formas de hacerlo, dependiendo de lo que necesites:

- **Importar toda la biblioteca**:

```
1 import numpy
2 array = numpy.array([1, 2, 3])
```

1.5: Importar toda la biblioteca

Esto importa toda la biblioteca y accedes a sus funciones mediante el prefijo `numpy.`.

- **Importar con un alias**:

```
1 import numpy as np
2 array = np.array([1, 2, 3])
```

1.6: Importar con alias

Usar un alias (por ejemplo, `np`) simplifica el uso de bibliotecas con nombres largos.

- **Importar funciones específicas**:

```
from math import sqrt, pi
result = sqrt(16)
print(pi)
```

1.7: Importar funciones específicas

Esto importa solo las funciones necesarias, evitando cargar toda la biblioteca.

- **Importar todas las funciones (no recomendado)**:

```
from math import *
result = sqrt(16)
```

1.8: Importar todo de una biblioteca

Aunque funcional, esta práctica puede causar conflictos si diferentes bibliotecas tienen funciones con el mismo nombre.

La elección del método de importación depende del contexto y las necesidades del proyecto.

1.9 Estructuras de datos en Python

Las estructuras de datos son fundamentales para almacenar, organizar y manipular datos de manera eficiente. Python incluye diversas estructuras de datos nativas que permiten trabajar con colecciones y relaciones entre datos (Lutz, 2013; Hetland, 2005).

1.9.1 Listas

Las listas son colecciones ordenadas y mutables que permiten almacenar elementos heterogéneos:

```
mi_lista = [1, 2, 3, "Python"]
mi_lista.append(4) # Agrega un elemento
print(mi_lista[2]) # Accede al tercer elemento
```

1.9: Ejemplo de listas

1.9.2 Tuplas

Las tuplas son similares a las listas, pero son inmutables, lo que significa que no pueden ser modificadas después de su creación:

```
mi_tupla = (1, 2, 3, "Python")
print(mi_tupla[1]) # Accede al segundo elemento
```

1.10: Ejemplo de tuplas

1.9.3 Diccionarios

Los diccionarios almacenan pares clave-valor y son útiles para representar datos estructurados:

```
mi_diccionario = {"nombre": "Python", "año": 1991}
print(mi_diccionario["nombre"]) # Accede al valor
    asociado a la clave "nombre"
```

1.11: Ejemplo de diccionarios

1.9.4 Conjuntos

Los conjuntos son colecciones no ordenadas de elementos únicos:

```
mi_conjunto = {1, 2, 3, 3} # Elimina duplicados automáticamente
print(mi_conjunto)
```

1.12: Ejemplo de conjuntos

1.9.5 Uso avanzado

Estas estructuras pueden combinarse para formar estructuras complejas, como listas de diccionarios o diccionarios de listas, que son útiles en análisis de datos y programación avanzada (Hetland, 2005).

1.9.6 Operaciones aritméticas

Las operaciones aritméticas básicas son sencillas en Python:

```
a = 10
b = 3
print(a + b) # Suma
print(a - b) # Resta
print(a * b) # Multiplicación
print(a / b) # División
print(a // b) # División entera
print(a % b) # Módulo
print(a ** b) # Potencia
```

1.13: Operaciones aritméticas

1.9.7 Operaciones con cadenas de texto

Python facilita trabajar con cadenas de texto:

```
cadena = "Python"
print(cadena + " es genial") # Concatenación
print(cadena * 3) # Repetición
print(len(cadena)) # Longitud de la cadena
print(cadena[0]) # Acceso al primer carácter
```

1.14: Operaciones con cadenas

1.9.8 Comparaciones

Las operaciones de comparación devuelven valores booleanos:

```
a = 5
b = 10
print(a == b) # Igualdad
print(a != b) # Diferencia
print(a < b) # Menor que
print(a > b) # Mayor que
print(a <= b) # Menor o igual que
print(a >= b) # Mayor o igual que
```

1.15: Operaciones de comparación

1.9.9 Operaciones lógicas

Python incluye operadores lógicos como `and`, `or` y `not`:

```
a = True
b = False
print(a and b) # AND lógico
print(a or b) # OR lógico
print(not a) # NOT lógico
```

1.16: Operaciones lógicas

1.10 Funciones y Módulos

Las funciones y módulos son elementos clave para estructurar y organizar el código en Python. Permiten reutilizar bloques de código y mantener programas más legibles y escalables (Lutz, 2013).

1.10.1 Funciones

Una función es un bloque de código reutilizable que realiza una tarea específica. Las funciones se definen con la palabra clave `def`:

```
def saludo(nombre):
"""Imprime un saludo personalizado."""
print(f"Hola, {nombre}!")

saludo("Python")
```

1.17: Definición de una función

Funciones con valores de retorno

Las funciones pueden devolver valores utilizando la palabra clave `return`:

```
def suma(a, b):
return a + b

resultado = suma(3, 5)
print(resultado) # Imprime 8
```

1.18: Función con retorno

Parámetros opcionales

Es posible definir parámetros con valores predeterminados:

```
def saludo(nombre, mensaje="Bienvenido"):
print(f"Hola, {nombre}. {mensaje}")

saludo("Python") # Usa el valor predeterminado para
    'mensaje'
saludo("Python", "¡Es un placer verte!")
```

1.19: Función con parámetros opcionales

1.10.2 Módulos

Un módulo es un archivo de Python que contiene funciones, clases y variables definidas para ser reutilizadas en otros programas. Python incluye numerosos módulos en su biblioteca estándar (Downey, 2012).

Importar un módulo

Puedes importar un módulo utilizando la palabra clave `import`:

```
import math

print(math.sqrt(16)) # Calcula la raíz cuadrada
print(math.pi) # Imprime el valor de pi
```

1.20: Uso del módulo math

Crear un módulo propio

Para crear un módulo, guarda tus funciones en un archivo `.py`. Por ejemplo, crea un archivo llamado `mimodulo.py`:

```
def saludo():
print("¡Hola desde mi módulo!")
```

1.21: Definición de un módulo

Luego, puedes importarlo y usarlo:

```
import mimodulo

mimodulo.saludo()
```

1.22: Uso de un módulo propio

1.11 Estructuras de Control

Las estructuras de control en Python permiten tomar decisiones y repetir bloques de código de manera eficien-

te. Estas estructuras incluyen condicionales, bucles y otras herramientas para controlar el flujo del programa (Lutz, 2013; Hetland, 2005).

1.11.1 Condicionales

Los condicionales permiten ejecutar bloques de código dependiendo de si una condición es verdadera o falsa:

```
# Ejemplo de if, elif y else
x = 10
if x > 0:
print("El número es positivo")
elif x == 0:
print("El número es cero")
else:
print("El número es negativo")
```

1.23: Uso de condicionales

1.11.2 Bucles

Los bucles permiten repetir un bloque de código varias veces:

Bucle for

El bucle `for` se utiliza para iterar sobre secuencias como listas o rangos:

```
# Iterar sobre una lista
numeros = [1, 2, 3, 4]
for numero in numeros:
print(numero)

# Iterar usando range
for i in range(5):
print(i)
```

1.24: Bucle for

Bucle while

El bucle `while` se utiliza cuando el número de iteraciones no está determinado de antemano:

```
# Ejemplo de while
contador = 0
while contador < 5:
print(contador)
contador += 1
```

1.25: Bucle while

1.11.3 Interrupción de bucles

Las sentencias `break` y `continue` permiten controlar el flujo dentro de los bucles:

```
        # Uso de break
        for i in range(10):
        if i == 5:
        break # Salir del bucle
        print(i)

        # Uso de continue
        for i in range(10):
        if i % 2 == 0:
        continue # Saltar a la siguiente iteración
        print(i)
```

1.26: Uso de break y continue

1.11.4 Comprensiones

Las comprensiones permiten crear listas, conjuntos y diccionarios de forma compacta:

```
        # Crear una lista con números al cuadrado
        cuadrados = [x**2 for x in range(10)]
        print(cuadrados)

        # Crear un diccionario con números y sus cuadrados
        cuadrados_dic = {x: x**2 for x in range(10)}
        print(cuadrados_dic)
```

1.27: Comprensiones de listas

1.12 Manipulación de Listas, Tuplas y Diccionarios

Las listas, tuplas y diccionarios son estructuras de datos esenciales en Python. Ofrecen diversas formas de almacenar, acceder y manipular datos, lo que las hace fundamentales para programar de manera eficiente (Lutz, 2013; Hetland, 2005).

1.12.1 Manipulación de listas

Las listas son mutables y permiten almacenar elementos heterogéneos. Las operaciones comunes incluyen:

Acceso a elementos

```
1       mi_lista = [10, 20, 30, 40, 50]
2       print(mi_lista[0]) # Primer elemento
3       print(mi_lista[-1]) # Último elemento
```

1.28: Acceso a elementos de una lista

Modificación de elementos

```
1       mi_lista = [1, 2, 3, 4]
2       mi_lista[2] = 99 # Cambia el tercer elemento
3       print(mi_lista)
```

1.29: Modificación de listas

Métodos comunes

```
mi_lista = [1, 2, 3]
mi_lista.append(4) # Agrega un elemento al final
mi_lista.insert(1, 99) # Inserta un elemento en la posición 1
mi_lista.remove(2) # Elimina el elemento 2
print(mi_lista.pop()) # Elimina y devuelve el último elemento
print(mi_lista)
```

1.30: Uso de métodos en listas

Slicing

El slicing permite obtener sublistas:

```
mi_lista = [0, 1, 2, 3, 4, 5]
print(mi_lista[1:4]) # Elementos del índice 1 al 3
print(mi_lista[:3]) # Primeros tres elementos
print(mi_lista[::2]) # Elementos con paso 2
```

1.31: Uso de slicing

1.12.2 Manipulación de tuplas

Las tuplas son inmutables, lo que significa que no pueden modificarse después de su creación. Esto las hace ideales para almacenar datos que no cambian.

Acceso a elementos

```
mi_tupla = (10, 20, 30, 40)
print(mi_tupla[1]) # Segundo elemento
print(mi_tupla[-1]) # Último elemento
```

1.32: Acceso a elementos de una tupla

Conversión entre tuplas y listas

Es posible convertir entre tuplas y listas:

```
mi_tupla = (1, 2, 3)
mi_lista = list(mi_tupla) # Convierte tupla a lista
mi_tupla_nueva = tuple(mi_lista) # Convierte lista a tupla
```

1.33: Conversión entre tuplas y listas

1.12.3 Manipulación de diccionarios

Los diccionarios almacenan pares clave-valor, lo que permite un acceso eficiente a los datos.

Acceso y modificación de elementos

```
mi_diccionario = {"nombre": "Python", "año": 1991}
print(mi_diccionario["nombre"]) # Acceso a un valor
mi_diccionario["año"] = 2023 # Modificación de un valor
mi_diccionario["creador"] = "Guido" # Agregar un nuevo
    par clave-valor
```

1.34: Acceso y modificación en diccionarios

Métodos comunes

```
mi_diccionario = {"a": 1, "b": 2}
print(mi_diccionario.keys()) # Devuelve las claves
print(mi_diccionario.values()) # Devuelve los valores
print(mi_diccionario.items()) # Devuelve pares clave-valor
mi_diccionario.pop("a") # Elimina un par clave-valor
print(mi_diccionario)
```

1.35: Uso de métodos en diccionarios

Iteración sobre diccionarios

```
mi_diccionario = {"nombre": "Python", "año": 1991}
for clave, valor in mi_diccionario.items():
print(f"{clave}: {valor}")
```

1.36: Iterar sobre un diccionario

1.13 La biblioteca SymPy

SymPy es una biblioteca de Python diseñada para realizar cálculos simbólicos. Proporciona herramientas para álgebra, cálculo, ecuaciones diferenciales y más, haciendo que sea una herramienta poderosa para matemáticos, ingenieros y científicos (Meurer et al., 2017).

1.13.1 Instalación de SymPy

Para instalar SymPy, puedes usar el gestor de paquetes `pip`:

```
pip install sympy
```

1.37: Instalación de SymPy

1.13.2 Operaciones básicas en SymPy

SymPy permite trabajar con expresiones simbólicas para realizar cálculos exactos. A continuación, se presentan algunas operaciones comunes:

Declaración de símbolos

Los símbolos son la base para los cálculos simbólicos:

```
from sympy import symbols

x, y = symbols('x y')
print(x + y) # Muestra x + y
```

1.38: Declaración de símbolos

Cálculo de derivadas

SymPy facilita el cálculo de derivadas:

```
from sympy import diff

f = x**3 + 2*x**2 + x
f_prime = diff(f, x)
print(f_prime) # Muestra 3*x**2 + 4*x + 1
```

1.39: Cálculo de derivadas

Resolución de ecuaciones

SymPy permite resolver ecuaciones algebraicas:

```
from sympy import Eq, solve

# Define la ecuación
ecuacion = Eq(x**2 - 4, 0)
soluciones = solve(ecuacion, x)
print(soluciones) # Muestra [-2, 2]
```

1.40: Resolución de ecuaciones

Integración

La integración simbólica también es posible con SymPy:

```
from sympy import integrate

f = x**2
area = integrate(f, (x, 0, 3))
print(area) # Muestra 9
```

1.41: Integración simbólica

1.13.3 Visualización de expresiones

SymPy incluye herramientas para la visualización de gráficos:

```
from sympy.plotting import plot

f = x**2
plot(f, (x, -10, 10))
```

1.42: Gráficos con SymPy

1.13.4 Ecuaciones diferenciales

SymPy puede resolver ecuaciones diferenciales ordinarias:

```
from sympy import Function, dsolve

f = Function('f')
ecuacion = f(x).diff(x, x) - 3*f(x)
solucion = dsolve(ecuacion)
print(solucion)
```

1.43: Resolución de ecuaciones diferenciales

1.14 Análisis de temperatura en estanques de tilapia

El uso de listas, bucles y condicionales en Python permite procesar datos reales y generar reportes útiles en acuicultura.

Aplicación en Acuicultura

En una granja de tilapia, se registran las temperaturas diarias del agua durante una semana. El rango ideal está entre

24 °C y 30 °C. El objetivo es calcular el promedio semanal, identificar valores fuera del rango, y visualizar los datos.

Datos de Temperatura

- Día 1: 23.5 °C
- Día 2: 24.8 °C
- Día 3: 26.1 °C
- Día 4: 29.5 °C
- Día 5: 30.2 °C
- Día 6: 25.0 °C
- Día 7: 22.9 °C

Solución Manual

1. Sumar todas las temperaturas: $23{,}5 + 24{,}8 + 26{,}1 + 29{,}5 + 30{,}2 + 25{,}0 + 22{,}9 = 182{,}0$ °C
2. Dividir entre 7 días: $182{,}0 \div 7 = 26{,}0$ °C promedio
3. Verificamos qué días están fuera del rango: Día 1 y Día 7.

Codificación en Python

```python
import matplotlib.pyplot as plt

temperaturas = [23.5, 24.8, 26.1, 29.5, 30.2, 25.0, 22.9]
dias = ["Lun", "Mar", "Mié", "Jue", "Vie", "Sáb", "Dom"]

# Cálculo del promedio
promedio = sum(temperaturas) / len(temperaturas)
print(f"Temperatura promedio: {promedio:.1f} °C")

# Días fuera del rango ideal
for i, temp in enumerate(temperaturas):
if temp < 24 or temp > 30:
print(f"{dias[i]}: {temp} °C (fuera del rango)")

# Gráfico
plt.plot(dias, temperaturas, marker='o')
plt.axhline(24, color='r', linestyle='--', label='Límite
    inferior')
plt.axhline(30, color='r', linestyle='--', label='Límite
    superior')
plt.title("Temperatura Semanal del Estanque")
plt.xlabel("Día")
plt.ylabel("Temperatura (°C)")
plt.legend()
plt.grid(True)
plt.show()
```

1.44: Análisis de temperatura semanal

Reflexión Didáctica

Este análisis simula el uso real de sensores y sistemas de alerta en acuicultura. Python permite automatizar este tipo de control de forma accesible y didáctica.

Conclusión del capítulo

Este capítulo ha introducido a los estudiantes en el uso de Python como herramienta fundamental en el estudio del cálculo. Se ha abordado desde su historia y características hasta su instalación y aplicación práctica en entornos científicos. Se revisaron estructuras de datos, operaciones aritméticas, visualización de funciones, y bibliotecas clave como NumPy, SymPy y matplotlib. Estos conocimientos establecen una base sólida para enfrentar los siguientes capítulos, donde se aplicará Python al análisis de límites, derivadas, integrales y otros conceptos esenciales del cálculo.

Conclusión del capítulo

Este capítulo ha introducido a los estudiantes en el uso de Python como herramienta fundamental en el estudio del cálculo. Se ha abordado desde su historia y características hasta su instalación y aplicación práctica en entornos científicos. Se revisaron estructuras de datos, operaciones aritméticas, visualización de funciones, y bibliotecas clave como NumPy, SymPy y matplotlib. Estos conocimientos establecen una base sólida para enfrentar los siguientes capítulos, donde se aplicará Python al análisis de límites, derivadas, integrales y otros conceptos esenciales del cálculo.

2 Mini Programación con Python

Objetivos específicos del capítulo

- Comprender los fundamentos de la programación estructurada aplicados al desarrollo de soluciones matemáticas en Python.
- Aplicar estructuras condicionales y de control de flujo para resolver problemas elementales de cálculo.
- Emplear funciones y ciclos iterativos para automatizar procedimientos computacionales en contextos educativos.
- Integrar buenas prácticas de codificación en Python para el diseño de scripts eficientes y legibles.

2.1 Importancia del pseudocódigo

Antes de escribir código en Python, es recomendable estructurar la solución del problema en un lenguaje intermedio, conocido como pseudocódigo. Esta herramienta per-

mite organizar las ideas de forma lógica sin preocuparse por la sintaxis, facilitando la comprensión del algoritmo y reduciendo errores en la implementación (Knuth, 1997). El uso del pseudocódigo está ampliamente aceptado en entornos educativos como una etapa previa esencial en la enseñanza de la programación (Guzdial & Ericson, 2013).

2.2 Implementación en Python

El pseudocódigo se traduce fácilmente en Python. Por ejemplo, para calcular el ŏ0e1rea de un círculo:

```
1       import math
2
3       def calcular_area_circulo(radio):
4       return math.pi * radio**2
5
6       radio = float(input("Ingrese el radio del círculo: "))
7       area = calcular_area_circulo(radio)
8       print("El área del círculo es:", area)
```

2.1: Importación de bibliotecas en Python

Este código ilustra la traducción clara del razonamiento algorítmico al código funcional.

2.3 Estructuras de control en Python

Las estructuras de control permiten tomar decisiones y repetir procesos, elementos esenciales en la resolución algorítmica de problemas.

2.3.1 Condicionales

Se emplean con las sentencias `if`, `elif` y `else`. Ejemplo:

```
numero = float(input("Ingrese un número: "))

if numero > 0:
print("El número es positivo.")
elif numero < 0:
print("El número es negativo.")
else:
print("El número es cero.")
```

2.2: Condicional simple en Python

2.3.2 Bucles

Los bucles permiten ejecutar bloques de código varias veces. Ejemplo con `while`:

```
contador = 1
while contador <= 5:
print("Iteración número", contador)
contador += 1
```

2.3: Bucle while en Python

Con `for`:

```
for numero in range(1, 6):
print(numero)
```

Estas estructuras fomentan el pensamiento iterativo necesario en programación científica (Downey, 2015).

2.4 Errores comunes al programar

- Olvidar la indentación correcta.
- Usar comillas inconsistentes en cadenas.
- Usar variables sin inicializarlas.
- Confundir el tipo de datos (`int`, `str`, `float`).

Reconocer y corregir estos errores mejora la calidad del código y fortalece el aprendizaje.

2.5 Buenas prácticas de programación

- Uso de nombres descriptivos en variables y funciones.
- Inclusión de comentarios que expliquen la lógica del código.
- Modularización del código en funciones reutilizables.
- Validación de entradas del usuario.

Estas estrategias no solo facilitan la lectura del código, sino que también mejoran su mantenibilidad (McConnell, 2004).

2.6 Ejercicios propuestos

1. Escribe un pseudocódigo que calcule el promedio de 3 temperaturas del agua.

2. Implementa un programa que determine si un número ingresado es múltiplo de 3.

3. Utiliza un bucle para sumar los primeros 10 números pares.

4. Crea una función que calcule el Ŏ0edndice de conversión alimenticia (FCR) usando datos de entrada.

5. Escribe un programa que lea una lista de pesos de peces y calcule el promedio.

2.7 Uso de estructuras de control para calcular el índice FCR

En acuicultura, uno de los indicadores más utilizados para evaluar la eficiencia alimenticia es el Índice de Conversión Alimenticia (FCR, por sus siglas en inglés). Se calcula dividiendo la cantidad de alimento proporcionado entre el incremento de biomasa.

Aplicación en Acuicultura

Durante una semana, se suministraron 16 kg de alimento a una piscina de tilapias. La biomasa inicial fue de 48 kg y

la final fue de 60 kg.

Problema

Calcular el FCR y clasificar la eficiencia alimenticia según los siguientes criterios:

- FCR < 1.5: Excelente
- $1.5 \leq$ FCR ≤ 2.0: Aceptable
- FCR > 2.0: Deficiente

Solución Manual

1. Incremento de biomasa: $60 - 48 = 12$ kg
2. FCR: $16/12 = 1{,}33$
3. Clasificación: Excelente

Codificación en Python

```
        alimento = 16 # kg de alimento suministrado
        biomasa_inicial = 48 # kg
        biomasa_final = 60 # kg

        incremento_biomasa = biomasa_final - biomasa_inicial
        fcr = alimento / incremento_biomasa

        print(f"FCR: {fcr:.2f}")

        if fcr < 1.5:
        print("Clasificación: Excelente")
        elif 1.5 <= fcr <= 2.0:
        print("Clasificación: Aceptable")
        else:
        print("Clasificación: Deficiente")
```

2.4: Cálculo del FCR y clasificación

Reflexión Didáctica

Este ejemplo ilustra cómo aplicar estructuras condicionales y operaciones aritméticas básicas para resolver un problema real en la gestión de cultivos acuícolas.

Conclusión del capítulo

El uso del pseudocódigo como herramienta previa a la programación permite a los estudiantes estructurar su pensamiento algorítmico y reducir errores. La posterior implementación en Python demuestra cómo conceptos teóricos pueden traducirse en soluciones funcionales. Con la incorporación de estructuras de control, buenas prácticas de

programación, y ejercicios contextualizados, se establece una base sólida para desarrollar programas eficaces y mantenibles, esenciales en contextos científicos y educativos.

3 Cálculo de Límites

Objetivos específicos del capítulo

- Comprender el concepto de límite, incluyendo límites laterales, al infinito y en funciones por tramos, como base del análisis matemático.
- Aplicar las propiedades fundamentales de los límites para simplificar expresiones y resolver problemas simbólicos.
- Utilizar la biblioteca `SymPy` de Python para calcular límites algebraicos, racionales, trigonométricos y exponenciales.
- Modelar situaciones reales, como el crecimiento de organismos en acuicultura, mediante el cálculo de límites simbólicos en Python.

3.1 Introducción

El cálculo de límites es un concepto fundamental del análisis matemático que describe el comportamiento de una función cuando la variable independiente se aproxima a un valor determinado o al infinito. Este concepto es esencial para definir otros temas avanzados como la derivada y la integral.

3.2 Teoría de los límites

3.2.1 Definición de límite

El límite de una función $f(x)$ cuando x tiende a un valor c se denota como:

$$\lim_{x \to c} f(x) = L$$

Esto significa que, a medida que x se aproxima a c, los valores de $f(x)$ se aproximan a L.

3.2.2 Límites laterales

Los límites laterales describen el comportamiento de la función al aproximarse al valor c desde la derecha o desde la izquierda:

$$\lim_{x \to c^+} f(x) \quad \text{(límite por la derecha)}$$

$$\lim_{x \to c^-} f(x) \quad \text{(límite por la izquierda)}$$

El límite existe si y sólo si ambos límites laterales son iguales.

3.2.3 Propiedades de los límites

- **Linealidad**: $\lim_{x\to c}[af(x)+bg(x)] = a\lim_{x\to c} f(x) + b\lim_{x\to c} g(x)$
- **Producto**: $\lim_{x\to c}[f(x)\cdot g(x)] = \lim_{x\to c} f(x)\cdot\lim_{x\to c} g(x)$
- **Cociente**: $\lim_{x\to c}\frac{f(x)}{g(x)} = \frac{\lim_{x\to c} f(x)}{\lim_{x\to c} g(x)}, \quad g(x)\neq 0$
- **Potencia**: $\lim_{x\to c}[f(x)]^n = [\lim_{x\to c} f(x)]^n$

3.2.4 Límites al infinito

Cuando x tiende al infinito, el límite describe el comportamiento asintótico de la función:

$$\lim_{x\to\infty}\frac{1}{x} = 0$$

3.3 Cálculo de límites con Python

La biblioteca `SymPy` de Python permite realizar cálculos simbólicos de límites. A continuación, se presentan 10 ejemplos de cálculo de límites con explicaciones.

3.3.1 Ejemplo 1: Límite básico

```
from sympy import symbols, limit

x = symbols('x')
limite = limit(x**2, x, 2)
print(f"L\'imite de x^2 cuando x tiende a 2: {limite}")
```

3.1: Límite básico

3.3.2 Ejemplo 2: Límite al infinito

```
from sympy import oo

limite = limit(1/x, x, oo)
print(f"L\'imite de 1/x cuando x tiende a infinito:
    {limite}")
```

3.2: Límite al infinito

3.3.3 Ejemplo 3: Límite lateral por la derecha

```
limite_derecha = limit(1/x, x, 0, dir='+')
print(f"Límite por la derecha de 1/x cuando x tiende a 0:
    {limite_derecha}")
```

3.3: Límite lateral

3.3.4 Ejemplo 4: Límite lateral por la izquierda

```
limite_izquierda = limit(1/x, x, 0, dir='-')
print(f"Límite por la izquierda de 1/x cuando x tiende a 0: {limite_izquierda}")
```

3.4: Límite lateral

3.3.5 Ejemplo 5: Límite de una función racional

```
limite = limit((x**2 - 4)/(x - 2), x, 2)
print(f"L\'imite de (x^2 - 4)/(x - 2) cuando x tiende a 2: {limite}")
```

3.5: Límite de función racional

3.3.6 Ejemplo 6: Límite de una función trigonométrica

```
from sympy import sin

limite = limit(sin(x)/x, x, 0)
print(f"L\'imite de sin(x)/x cuando x tiende a 0: {limite}")
```

3.6: Límite trigonométrico

3.3.7 Ejemplo 7: Límite al infinito de una exponencial

```
    from sympy import exp

    limite = limit(exp(-x), x, oo)
    print(f"L\'imite de exp(-x) cuando x tiende a infinito:
        {limite}")
```

3.7: Límite exponencial

3.3.8 Ejemplo 8: Límite con una función definida por tramos

```
    from sympy import Piecewise

    f = Piecewise((x**2, x < 0), (x, x >= 0))
    limite = limit(f, x, 0, dir='+')
    print(f"L\'imite de una funci\'on por tramos: {limite}")
```

3.8: Función por tramos

3.3.9 Ejemplo 9: Límite con raíces cuadradas

```
    from sympy import sqrt

    limite = limit(sqrt(x) - 2, x, 4)
    print(f"L\'imite de sqrt(x) - 2 cuando x tiende a 4:
        {limite}")
```

3.9: Límite con raíces cuadradas

3.3.10 Ejemplo 10: Límite de una función logarítmica

```
        from sympy import log

        limite = limit(log(x), x, 1)
        print(f"L\'imite de log(x) cuando x tiende a 1: {limite}")
```

3.10: Límite logarítmico

3.4 Aplicación del cálculo de límites: tasa de crecimiento al inicio del cultivo

En acuicultura, es común modelar el crecimiento inicial de peces con funciones cuadráticas o exponenciales. El cálculo de límites permite estimar la tasa de crecimiento en los primeros días.

Aplicación en Acuicultura

Supongamos que el crecimiento de la masa corporal de un grupo de alevines de tilapia está modelado por la función:

$$f(t) = 0{,}1t^2 + 0{,}5t$$

donde $f(t)$ es la masa (en gramos) al día t. Se desea conocer la tasa de crecimiento instantáneo justo al iniciar el cultivo, es decir, cuando $t \to 0$.

Problema

Calcular el siguiente límite:

$$\lim_{t \to 0} \frac{f(t) - f(0)}{t}$$

Solución Manual

1. $f(0) = 0{,}1(0)^2 + 0{,}5(0) = 0$
2. $\frac{f(t)-f(0)}{t} = \frac{0{,}1t^2+0{,}5t-0}{t} = 0{,}1t + 0{,}5$
3. Tomar el límite: $\lim_{t\to 0}(0{,}1t + 0{,}5) = 0{,}5$

Codificación en Python

```
from sympy import symbols, limit

t = symbols('t')
f = 0.1*t**2 + 0.5*t
limite = limit((f - f.subs(t, 0))/t, t, 0)
print(f"El límite es: {limite}")
```

3.11: Cálculo de límite simbólico con SymPy

Reflexión Didáctica

Este tipo de límite representa la derivada en $t = 0$, interpretada como la tasa de cambio inicial. Es clave en el análisis de crecimiento en biología y acuicultura.

Conclusión del capítulo

El cálculo de límites es una herramienta clave en el análisis matemático y la programación simbólica. Utilizar Python

y SymPy para calcular límites facilita la comprensión de conceptos abstractos y permite explorar una amplia gama de funciones y comportamientos. Esta metodología fortalece la capacidad analítica del estudiante y sirve como base para los temas siguientes: derivación, continuidad y análisis de funciones.

4 Derivación de Funciones de Variable Real

Objetivos específicos del capítulo

- Comprender el concepto de derivada como tasa de cambio y su interpretación geométrica y física.
- Aplicar reglas de derivación para resolver problemas simbólicos y contextuales con funciones algebraicas, trigonométricas y compuestas.
- Utilizar Python y la biblioteca `SymPy` para calcular derivadas, analizar puntos críticos y modelar situaciones reales.
- Resolver problemas de optimización y determinar puntos de inflexión en funciones, aplicando derivadas a contextos como la física y la acuicultura.

4.1 Introducción

La derivación es una operación fundamental en el cálculo diferencial que mide la tasa de cambio de una función res-

pecto a su variable independiente. Este concepto permite analizar el comportamiento local de una función, determinar puntos críticos, y resolver problemas de optimización en diversas áreas de la ciencia y la ingeniería.

4.1.1 Definición de la Derivada

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

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

Este límite, si existe, representa la pendiente de la recta tangente a la curva de $f(x)$ en el punto $(a, f(a))$.

4.1.2 Reglas Fundamentales de Derivación

Algunas reglas importantes para calcular derivadas son:

- **Regla de la Potencia**: $\frac{d}{dx}[x^n] = n \cdot x^{n-1}$
- **Regla del Producto**: $\frac{d}{dx}[u \cdot v] = u' \cdot v + u \cdot v'$
- **Regla del Cociente**: $\frac{d}{dx}\left[\frac{u}{v}\right] = \frac{u' \cdot v - u \cdot v'}{v^2}$
- **Regla de la Cadena**: $\frac{d}{dx}[f(g(x))] = f'(g(x)) \cdot g'(x)$

4.1.3 Aplicaciones de la Derivada

La derivada se utiliza en una variedad de aplicaciones, incluyendo:

- Determinar la pendiente de una curva.
- Identificar puntos críticos y clasificar extremos locales.
- Resolver problemas de optimización.
- Analizar la concavidad y los puntos de inflexión de una función.

4.2 Cálculo de Derivadas con Python

La biblioteca SymPy en Python permite calcular derivadas simbólicas y evaluarlas en puntos específicos. A continuación, se presentan ejemplos resueltos con Python.

4.2.1 Problemas de Derivadas en Física

Velocidad como la Derivada de la Posición

```
1       from sympy import symbols, diff
2
3       # Posici\'on en funci\'on del tiempo
4       x, t = symbols('x t')
5       posicion = 5*t**2 + 3*t + 2
6       velocidad = diff(posicion, t)
7       print(f"Velocidad: {velocidad}")
```

4.1: Velocidad como la derivada de la posición

Aceleración como la Derivada de la Velocidad

```
        # Aceleraci\'on como la segunda derivada de la posici\'on
        aceleracion = diff(velocidad, t)
        print(f"Aceleraci\'on: {aceleracion}")
```

4.2: Aceleración como la derivada de la velocidad

Ley de Hooke: Fuerza como Derivada de la Energía

```
        k, x = symbols('k x')
        energia_potencial = 1/2 * k * x**2
        fuerza = -diff(energia_potencial, x)
        print(f"Fuerza: {fuerza}")
```

4.3: Ley de Hooke

Ley de Enfriamiento de Newton

```
        T, t = symbols('T t')
        constante = symbols('k')
        temperatura = T * exp(-constante * t)
        derivada_temperatura = diff(temperatura, t)
        print(f"Tasa de cambio de la temperatura:
            {derivada_temperatura}")
```

4.4: Velocidad de enfriamiento

Movimiento Circular: Velocidad Angular

```
theta, t = symbols('theta t')
movimiento = theta**3 - 4*theta + 6
derivada_angular = diff(movimiento, t)
print(f"Velocidad angular: {derivada_angular}")
```

4.5: Velocidad angular

4.3 Conclusión

La derivación de funciones es esencial para modelar y resolver problemas en física. Python, con SymPy, facilita el cálculo y la interpretación de derivadas, permitiendo aplicaciones prácticas en diversas disciplinas.

4.3.1 Problemas de Derivadas en Física

Velocidad como la Derivada de la Posición

```
from sympy import symbols, diff

# Posici\'on en funci\'on del tiempo
x, t = symbols('x t')
posicion = 5*t**2 + 3*t + 2
velocidad = diff(posicion, t)
print(f"Velocidad: {velocidad}")
```

4.6: Velocidad como la derivada de la posición

Aceleración como la Derivada de la Velocidad

```
# Aceleraci\'on como la segunda derivada de la posici\'on
aceleracion = diff(velocidad, t)
print(f"Aceleraci\'on: {aceleracion}")
```

4.7: Aceleración como la derivada de la velocidad

Ley de Hooke: Fuerza como Derivada de la Energía

```
k, x = symbols('k x')
energia_potencial = 1/2 * k * x**2
fuerza = -diff(energia_potencial, x)
print(f"Fuerza: {fuerza}")
```

4.8: Ley de Hooke

Ley de Enfriamiento de Newton

```
T, t = symbols('T t')
constante = symbols('k')
temperatura = T * exp(-constante * t)
derivada_temperatura = diff(temperatura, t)
print(f"Tasa de cambio de la temperatura:
    {derivada_temperatura}")
```

4.9: Velocidad de enfriamiento

Movimiento Circular: Velocidad Angular

```
theta, t = symbols('theta t')
movimiento = theta**3 - 4*theta + 6
derivada_angular = diff(movimiento, t)
print(f"Velocidad angular: {derivada_angular}")
```

4.10: Velocidad angular

4.4 Conclusión

La derivación de funciones es esencial para modelar y resolver problemas en física. Python, con `SymPy`, facilita el cálculo y la interpretación de derivadas, permitiendo aplicaciones prácticas en diversas disciplinas.

4.5 Cálculo de Máximos y Mínimos

4.5.1 Teoría

Para determinar los máximos y mínimos de una función $f(x)$, se utilizan las siguientes herramientas basadas en la derivada:

Primera Derivada

La primera derivada $f'(x)$ se utiliza para encontrar los puntos críticos, que son aquellos valores de x donde $f'(x) = 0$ o $f'(x)$ no está definida.

Segunda Derivada

La segunda derivada $f''(x)$ ayuda a clasificar los puntos críticos:

- Si $f''(x) > 0$, el punto crítico es un mínimo local.
- Si $f''(x) < 0$, el punto crítico es un máximo local.
- Si $f''(x) = 0$, el criterio es inconcluso.

4.5.2 Pasos para Calcular Máximos y Mínimos

1. Calcular la primera derivada $f'(x)$.
2. Resolver $f'(x) = 0$ para encontrar los puntos críticos.
3. Calcular la segunda derivada $f''(x)$.
4. Evaluar $f''(x)$ en cada punto crítico para clasificarlo como máximo o mínimo.

4.6 Ejemplos con Python

A continuación se presentan 10 ejemplos resueltos utilizando Python para calcular máximos y mínimos.

4.6.1 Ejemplo 1: Máximo de una Parábola

```
from sympy import symbols, diff, solve

x = symbols('x')
funcion = -x**2 + 4*x + 5
primera_derivada = diff(funcion, x)
puntos_criticos = solve(primera_derivada, x)
segunda_derivada = diff(funcion, x, 2)

for punto in puntos_criticos:
clasificacion = "M\'aximo" if segunda_derivada.subs(x,
    punto) < 0 else "M\'inimo"
print(f"Punto cr\'itico en x={punto}: {clasificacion}")
```

4.11: Máximo de una parábola

4.6.2 Ejemplo 2: Mínimo de una Parábola

```
funcion = x**2 - 6*x + 8
primera_derivada = diff(funcion, x)
puntos_criticos = solve(primera_derivada, x)
segunda_derivada = diff(funcion, x, 2)

for punto in puntos_criticos:
clasificacion = "M\'aximo" if segunda_derivada.subs(x,
    punto) < 0 else "M\'inimo"
print(f"Punto cr\'itico en x={punto}: {clasificacion}")
```

4.12: Mínimo de una parábola

4.6.3 Ejemplo 3: Funciones Trigonométricas

```
        from sympy import sin

        funcion = sin(x)
        primera_derivada = diff(funcion, x)
        puntos_criticos = solve(primera_derivada, x)
        segunda_derivada = diff(funcion, x, 2)

        print(f"Puntos cr\'iticos: {puntos_criticos}")
        for punto in puntos_criticos:
        print(f"Segunda derivada en x={punto}:
            {segunda_derivada.subs(x, punto)}")
```

4.13: Funciones trigonométricas

4.6.4 Ejemplo 4: Clasificación con Segunda Derivada

```
        funcion = x**3 - 3*x**2 - 9*x + 27
        primera_derivada = diff(funcion, x)
        puntos_criticos = solve(primera_derivada, x)
        segunda_derivada = diff(funcion, x, 2)

        for punto in puntos_criticos:
        print(f"x={punto}, f''(x)={segunda_derivada.subs(x,
            punto)}")
```

4.14: Clasificación con segunda derivada

4.6.5 Ejemplo 5: Optimización de un Producto

```
funcion = x**2 * (10 - x)
primera_derivada = diff(funcion, x)
puntos_criticos = solve(primera_derivada, x)
segunda_derivada = diff(funcion, x, 2)

for punto in puntos_criticos:
print(f"Punto cr\'itico en x={punto},
    f''(x)={segunda_derivada.subs(x, punto)}")
```

4.15: Optimización de un producto

4.6.6 Ejemplo 6: Problema de Física

```
funcion = -5*x**2 + 20*x
primera_derivada = diff(funcion, x)
puntos_criticos = solve(primera_derivada, x)
segunda_derivada = diff(funcion, x, 2)

print(f"Puntos cr\'iticos: {puntos_criticos}")
for punto in puntos_criticos:
clasificacion = "M\'aximo" if segunda_derivada.subs(x,
    punto) < 0 else "M\'inimo"
print(f"Punto cr\'itico en x={punto}: {clasificacion}")
```

4.16: Problema físico

4.7 Conclusión

El cálculo de máximos y mínimos utilizando la primera y segunda derivada es una herramienta poderosa para analizar el comportamiento de funciones y resolver problemas de optimización.

4.8 Cálculo de Puntos de Inflexión

4.8.1 Teoría

Un punto de inflexión es un punto en el que una función cambia la concavidad, es decir, pasa de ser cóncava hacia arriba ($f''(x) > 0$) a cóncava hacia abajo ($f''(x) < 0$) o viceversa. Para determinar los puntos de inflexión se siguen los siguientes pasos:

1. Calcular la segunda derivada $f''(x)$.
2. Resolver $f''(x) = 0$ para encontrar los puntos candidatos.
3. Verificar el cambio de signo de $f''(x)$ en los intervalos alrededor de los puntos candidatos.

4.8.2 Pasos para Identificar Puntos de Inflexión

- **Concavidad hacia arriba**: $f''(x) > 0$.
- **Concavidad hacia abajo**: $f''(x) < 0$.

Si la concavidad cambia en un punto candidato, ese punto es un punto de inflexión.

4.9 Ejemplos con Python

4.9.1 Ejemplo 1: Punto de Inflexión de una Cúbica

```
from sympy import symbols, diff, solve

x = symbols('x')
funcion = x**3 - 3*x**2 + 4
segunda_derivada = diff(funcion, x, 2)
puntos_candidatos = solve(segunda_derivada, x)

print(f"Puntos candidatos: {puntos_candidatos}")
for punto in puntos_candidatos:
signo_antes = segunda_derivada.subs(x, punto - 1)
signo_despues = segunda_derivada.subs(x, punto + 1)
if signo_antes * signo_despues < 0:
print(f"Punto de inflexi\'on en x={punto}")
```

4.17: Punto de inflexión de una cúbica

4.9.2 Ejemplo 2: Punto de Inflexión de una Función Trigonométrica

```
from sympy import sin

funcion = sin(x)
segunda_derivada = diff(funcion, x, 2)
puntos_candidatos = solve(segunda_derivada, x)

print(f"Puntos candidatos: {puntos_candidatos}")
for punto in puntos_candidatos:
print(f"x={punto}, segunda derivada cambia de signo")
```

4.18: Punto de inflexión de una función trigonométrica

4.9.3 Ejemplo 3: Clasificación de Puntos Candidatos

```
funcion = x**4 - 4*x**2
segunda_derivada = diff(funcion, x, 2)
puntos_candidatos = solve(segunda_derivada, x)

for punto in puntos_candidatos:
valor_antes = segunda_derivada.subs(x, punto - 1)
valor_despues = segunda_derivada.subs(x, punto + 1)
if valor_antes * valor_despues < 0:
print(f"Punto de inflexi\'on en x={punto}")
```

4.19: Clasificación de puntos candidatos

4.9.4 Ejemplo 4: Función con Raíces

```
from sympy import sqrt

funcion = sqrt(x**3)
segunda_derivada = diff(funcion, x, 2)
puntos_candidatos = solve(segunda_derivada, x)
print(f"Puntos candidatos: {puntos_candidatos}")
```

4.20: Función con raíces cuadradas

4.10 Aplicación de derivadas: optimización del uso de alimento

En acuicultura, el uso eficiente del alimento es clave para maximizar el crecimiento de los peces y minimizar los costos. La derivada permite encontrar el punto donde se maximiza la ganancia de peso respecto a la cantidad de alimento suministrado.

Aplicación en Acuicultura

Supongamos que la ganancia de peso de una población de peces, en gramos, depende de la cantidad de alimento suministrado x en kilogramos, según la función:

$$G(x) = -2x^2 + 12x$$

Se desea encontrar la cantidad de alimento x que maximiza la ganancia de peso.

Problema

Determinar el valor de x que maximiza $G(x)$ usando derivadas.

Solución Manual

- Derivada: $G'(x) = -4x + 12$
- Igualar a cero: $-4x + 12 = 0 \Rightarrow x = 3$
- Segunda derivada: $G''(x) = -4 < 0 \rightarrow$ máximo local.

Entonces, la cantidad óptima de alimento es de 3 kg.

Codificación en Python

```python
from sympy import symbols, diff, solve

x = symbols('x')
G = -2*x**2 + 12*x

G_derivada = diff(G, x)
critico = solve(G_derivada, x)[0]

segunda = diff(G, x, 2)

print(f"Valor crítico: x = {critico}")
print(f"Segunda derivada: G''(x) = {segunda}")
```

4.21: Cálculo del máximo de ganancia de peso

Reflexión Didáctica

Este ejemplo muestra cómo usar derivadas para optimizar recursos. En acuicultura, este tipo de análisis puede aplicarse a la alimentación, temperatura, oxígeno, etc.

Conclusiones

En este capítulo se ha demostrado cómo la derivada, entendida como tasa de cambio, es una herramienta fundamental para el análisis de variaciones en contextos reales. Su aplicación en el ámbito de la acuicultura permite optimizar procesos esenciales como la alimentación, el crecimiento de los organismos y la gestión de recursos.
A través de ejemplos contextualizados y su codificación en Python, los estudiantes pudieron identificar puntos crí-

ticos en funciones, clasificar máximos y mínimos locales, y validar sus hallazgos con el uso de la segunda derivada. Esta aproximación no solo fortalece el razonamiento matemático, sino que también promueve el pensamiento computacional y la toma de decisiones basadas en modelos cuantitativos.

El uso de bibliotecas como SymPy, y la integración de Python en el proceso de aprendizaje, ha facilitado el tratamiento simbólico de derivadas, reforzando la conexión entre teoría y práctica. En resumen, este capítulo consolida las bases del cálculo diferencial orientadas a la resolución de problemas reales en el ámbito acuícola.

5 La Integral Indefinida

Objetivos del capítulo

- Comprender el concepto de integral indefinida y sus propiedades fundamentales.
- Aplicar las reglas de integración a funciones comunes.
- Utilizar Python (SymPy) para calcular integrales simbólicas.
- Resolver problemas aplicados en contexto agropecuario con integrales.

5.1 Introducción

La integral indefinida es uno de los conceptos fundamentales del cálculo integral. Representa el conjunto de todas las primitivas de una función dada y se denota generalmente por:

$$\int f(x)\,dx$$

Donde $f(x)$ es la función integranda y dx indica que la integral se toma con respecto a la variable x. El resultado de una integral indefinida incluye una constante de integración C, ya que las primitivas difieren entre sí por una constante.

5.2 Teoría de la Integral Indefinida

5.2.1 Definición

La integral indefinida de una función $f(x)$ es una función $F(x)$ tal que $F'(x) = f(x)$. Es decir:

$$\int f(x)\,dx = F(x) + C$$

5.2.2 Propiedades de la Integral Indefinida

- **Linealidad**: $\int [af(x) + bg(x)]\,dx = a\int f(x)\,dx + b\int g(x)\,dx$.
- **Constante por fuera**: $\int cf(x)\,dx = c\int f(x)\,dx$, donde c es una constante.
- **Suma de integrales**: $\int [f(x) + g(x)]\,dx = \int f(x)\,dx + \int g(x)\,dx$.

5.2.3 Integrales Comunes

$$\int x^n \, dx = \frac{x^{n+1}}{n+1} + C, \quad n \neq -1$$

$$\int e^x \, dx = e^x + C$$

$$\int \frac{1}{x} \, dx = \ln|x| + C$$

$$\int \sin(x) \, dx = -\cos(x) + C$$

$$\int \cos(x) \, dx = \sin(x) + C$$

5.3 Cálculo de Integrales Indefinidas con Python

La biblioteca `SymPy` en Python proporciona herramientas para realizar cálculos simbólicos de integrales indefinidas. A continuación se presentan 10 ejemplos resueltos.

5.3.1 Ejemplo 1: Integral de una Potencia

```
1       from sympy import symbols, integrate
2
3       x = symbols('x')
4       funcion = x**3
5       resultado = integrate(funcion, x)
6       print(f"Integral de x^3: {resultado}")
```

5.1: Integral de una potencia

5.3.2 Ejemplo 2: Integral de una Suma

```
funcion = x**2 + 3*x + 5
resultado = integrate(funcion, x)
print(f"Integral de x^2 + 3x + 5: {resultado}")
```

5.2: Integral de una suma

5.3.3 Ejemplo 3: Integral Exponencial

```
from sympy import exp

funcion = exp(x)
resultado = integrate(funcion, x)
print(f"Integral de e^x: {resultado}")
```

5.3: Integral exponencial

5.3.4 Ejemplo 4: Integral Logarítmica

```
from sympy import log

funcion = log(x)
resultado = integrate(funcion, x)
print(f"Integral de log(x): {resultado}")
```

5.4: Integral logarítmica

5.3.5 Ejemplo 5: Integral Trigonométrica

```
from sympy import sin

funcion = sin(x)
resultado = integrate(funcion, x)
print(f"Integral de sin(x): {resultado}")
```

5.5: Integral trigonométrica

5.3.6 Ejemplo 6: Integral por Partes

```
from sympy import cos

funcion = x * cos(x)
resultado = integrate(funcion, x)
print(f"Integral de x*cos(x): {resultado}")
```

5.6: Integral por partes

5.3.7 Ejemplo 7: Integral de una Raíz Cuadrada

```
from sympy import sqrt

funcion = sqrt(x)
resultado = integrate(funcion, x)
print(f"Integral de sqrt(x): {resultado}")
```

5.7: Integral de una raíz cuadrada

5.3.8 Ejemplo 8: Integral de una Función Racional

```
funcion = 1/(x**2 + 1)
resultado = integrate(funcion, x)
print(f"Integral de 1/(x^2 + 1): {resultado}")
```

5.8: Integral de una función racional

5.3.9 Ejemplo 9: Integral de un Producto

```
funcion = x * exp(-x**2)
resultado = integrate(funcion, x)
print(f"Integral de x*exp(-x^2): {resultado}")
```

5.9: Integral de un producto

5.3.10 Ejemplo 10: Integral con Constantes

```
c = symbols('c')
funcion = c * x**2
resultado = integrate(funcion, x)
print(f"Integral de c*x^2: {resultado}")
```

5.10: Integral con constantes

Conclusiones

La integral indefinida es esencial en la comprensión del cambio acumulativo. Python, a través de SymPy, permite un enfoque didáctico para su cálculo. Esta herramienta permite a los estudiantes resolver problemas simbólicos de

forma clara, reforzando el aprendizaje de los fundamentos del cálculo.

5.4 Aplicación de integrales: acumulación de biomasa

La integral indefinida permite calcular funciones acumulativas. En acuicultura, puede emplearse para estimar la biomasa acumulada en función del tiempo a partir de una tasa de crecimiento conocida.

Aplicación en Acuicultura

Supongamos que la tasa de crecimiento de biomasa de una población de peces está dada por la función:

$$g(t) = 3t^2 + 2t \quad \text{(gramos por día)}$$

donde t es el tiempo en días. Se desea conocer la función que representa la biomasa acumulada desde el día cero.

Problema

Calcular la integral indefinida de $g(t)$ para obtener la función de biomasa acumulada $B(t)$.

Solución Manual

- $\int g(t)\,dt = \int (3t^2 + 2t)\,dt = t^3 + t^2 + C$

La función $B(t) = t^3 + t^2 + C$ representa la biomasa acumulada (en gramos), con C como constante de integración.

Codificación en Python

```
from sympy import symbols, integrate

t = symbols('t')
g = 3*t**2 + 2*t

B = integrate(g, t)
print(f"Función de biomasa acumulada: B(t) = {B} + C")
```

5.11: Cálculo de integral indefinida con SymPy

Reflexión Didáctica

Este tipo de modelos ayuda a predecir el crecimiento acumulado a lo largo del tiempo y a planificar actividades como la cosecha, el manejo de densidades y la alimentación.

6 La Integral Definida

Objetivos específicos del capítulo

- Interpretar la integral definida como área neta bajo una curva y su significado geométrico.
- Aplicar el teorema fundamental del cálculo para evaluar integrales definidas de funciones continuas.
- Utilizar Python y SymPy para resolver integrales definidas simbólica y numéricamente.
- Resolver problemas aplicados relacionados con áreas, volúmenes y acumulación en contextos agropecuarios.

6.1 Introducción

La integral definida es un concepto central en el cálculo integral, utilizado para calcular *áreas bajo la curva*, entre otras aplicaciones. Una integral definida de una función

continua $f(x)$ en el intervalo $[a,b]$ se expresa como:

$$\int_a^b f(x)\,dx$$

Donde:

- a es el límite inferior de integración.
- b es el límite superior de integración.
- $f(x)$ es la función integranda.

El valor de la integral definida se interpreta como el *área neta* entre la función y el eje x en el intervalo dado.

6.2 Teoría de la Integral Definida

6.2.1 Definición Formal

La integral definida se define como el límite de una suma de Riemann:

$$\int_a^b f(x)\,dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i^*)\Delta x$$

Donde:

- $\Delta x = \frac{b-a}{n}$ es el ancho de cada subintervalo.
- x_i^* es un punto dentro del subintervalo $[x_{i-1}, x_i]$.

6.2.2 Teorema Fundamental del Cálculo

El teorema fundamental del cálculo conecta la integral definida con la derivada:

1. **Primera parte**: Si $F(x)$ es una primitiva de $f(x)$, entonces:

$$\int_a^b f(x)\,dx = F(b) - F(a)$$

2. **Segunda parte**: Si $f(x)$ es continua en $[a,b]$, entonces:

$$\frac{d}{dx}\left(\int_a^x f(t)\,dt\right) = f(x)$$

6.3 Propiedades de la Integral Definida

- $\int_a^a f(x)\,dx = 0$.
- $\int_a^b f(x)\,dx = -\int_b^a f(x)\,dx$.
- Si $f(x) \geq 0$ en $[a,b]$, entonces $\int_a^b f(x)\,dx \geq 0$.
- Linealidad: $\int_a^b [af(x)+bg(x)]\,dx = a\int_a^b f(x)\,dx + b\int_a^b g(x)\,dx$.

6.4 Cálculo de Integrales Definidas con Python

La biblioteca `SymPy` en Python facilita el cálculo simbólico y numérico de integrales definidas. A continuación, se presentan 10 ejemplos resueltos.

6.4.1 Ejemplo 1: Integral de una Potencia

```
from sympy import symbols, integrate

x = symbols('x')
resultado = integrate(x**3, (x, 0, 2))
print(f"Integral de x^3 desde 0 hasta 2: {resultado}")
```

6.1: Integral de una potencia

6.4.2 Ejemplo 2: Integral Exponencial

```
from sympy import exp

resultado = integrate(exp(x), (x, 1, 3))
print(f"Integral de e^x desde 1 hasta 3: {resultado}")
```

6.2: Integral exponencial

6.4.3 Ejemplo 3: Integral Logarítmica

```
from sympy import log

resultado = integrate(log(x), (x, 1, 4))
print(f"Integral de log(x) desde 1 hasta 4: {resultado}")
```

6.3: Integral logarítmica

6.4.4 Ejemplo 4: Integral Trigonométrica

```
        from sympy import sin

        resultado = integrate(sin(x), (x, 0, 3.1416))
        print(f"Integral de sin(x) desde 0 hasta pi: {resultado}")
```

6.4: Integral trigonométrica

6.4.5 Ejemplo 5: Integral de una Función Cuadrática

```
        funcion = x**2 - 4*x + 6
        resultado = integrate(funcion, (x, -1, 3))
        print(f"Integral de x^2 - 4x + 6 desde -1 hasta 3:
            {resultado}")
```

6.5: Integral de una función cuadrática

6.4.6 Ejemplo 6: Integral de una Función Racional

```
        funcion = 1/(x**2 + 1)
        resultado = integrate(funcion, (x, 0, 1))
        print(f"Integral de 1/(x^2 + 1) desde 0 hasta 1:
            {resultado}")
```

6.6: Integral de una función racional

6.4.7 Ejemplo 7: Área Bajo una Curva

```
funcion = x**2
resultado = integrate(funcion, (x, 0, 5))
print(f"\u00c1rea bajo la curva x^2 desde 0 hasta 5:
    {resultado}")
```

6.7: Área bajo una curva

6.4.8 Ejemplo 8: Integral con Constantes

```
a, b = symbols('a b')
resultado = integrate(a*x**2 + b, (x, 1, 2))
print(f"Integral de a*x^2 + b desde 1 hasta 2:
    {resultado}")
```

6.8: Integral con constantes

6.4.9 Ejemplo 9: Integral de una Función con Raíces

```
from sympy import sqrt

funcion = sqrt(x)
resultado = integrate(funcion, (x, 0, 4))
print(f"Integral de sqrt(x) desde 0 hasta 4: {resultado}")
```

6.9: Integral con raíces cuadradas

6.4.10 Ejemplo 10: Integral de un Producto

```
funcion = x * exp(-x**2)
resultado = integrate(funcion, (x, 0, 1))
print(f"Integral de x*exp(-x^2) desde 0 hasta 1:
    {resultado}")
```

6.10: Integral de un producto

6.5 Conclusión

La integral definida es una herramienta poderosa para calcular *áreas netas* y resolver problemas de acumulación. Con Python y `SymPy`, podemos abordar estos problemas de manera eficiente y precisa.

6.6 Cálculo de Áreas y Volúmenes

6.6.1 Cálculo de Áreas

El cálculo de *áreas bajo la curva* es una aplicación directa de la integral definida. La integral:

$$\text{Área} = \int_a^b |f(x)|\,dx$$

permite calcular el *área total*, independientemente de si la función es positiva o negativa en el intervalo dado.

Ejemplo: ŏ0c1rea Bajo una Parábola

```
from sympy import symbols, integrate

x = symbols('x')
funcion = x**2 - 4
resultado = integrate(abs(funcion), (x, -3, 3))
print(f"\u00c1rea bajo la curva x^2 - 4 entre -3 y 3:
    {resultado}")
```

6.11: Cálculo del área bajo una parábola

6.6.2 Cálculo de Volúmenes

El cálculo de volúmenes de sólidos de revolución se realiza mediante la técnica del disco o la técnica del anillo. Por ejemplo, usando el eje x:

$$V = \pi \int_a^b [f(x)]^2 \, dx$$

Ejemplo: Volumen de un Sólido de Revolución

```
funcion = x**2
resultado = integrate(funcion**2 * 3.1416, (x, 0, 2))
print(f"Volumen del s\u00f3lido de revoluci\u00f3n de x^2
    entre 0 y 2: {resultado}")
```

6.12: Volumen de un sólido de revolución

6.7 Cálculo de Integrales Definidas con Python

La biblioteca SymPy en Python facilita el cálculo simbólico y numérico de integrales definidas. A continuación, se presentan 10 ejemplos resueltos.

6.7.1 Ejemplo 1: Integral de una Potencia

```
from sympy import symbols, integrate

x = symbols('x')
resultado = integrate(x**3, (x, 0, 2))
print(f"Integral de x^3 desde 0 hasta 2: {resultado}")
```

6.13: Integral de una potencia

6.7.2 Ejemplo 2: Integral Exponencial

```
from sympy import exp

resultado = integrate(exp(x), (x, 1, 3))
print(f"Integral de e^x desde 1 hasta 3: {resultado}")
```

6.14: Integral exponencial

6.8 Conclusión

El cálculo de *áreas* y *volúmenes* amplía las aplicaciones de la integral definida, permitiendo resolver problemas geo-

métricos y físicos. Python, a través de `SymPy`, ofrece herramientas eficientes para abordar estos problemas.

6.9 Cálculo de Áreas y Volúmenes

6.9.1 Cálculo de Áreas

El cálculo de *áreas bajo la curva* es una aplicación directa de la integral definida. La integral:

$$\text{Área} = \int_a^b |f(x)|\,dx$$

permite calcular el *área total*, independientemente de si la función es positiva o negativa en el intervalo dado.

Ejemplo: Áreas Bajo una Parábola

```
from sympy import symbols, integrate

x = symbols('x')
funcion = x**2 - 4
resultado = integrate(abs(funcion), (x, -3, 3))
print(f"\u00c1rea bajo la curva x^2 - 4 entre -3 y 3:
    {resultado}")
```

6.15: Cálculo del área bajo una parábola

6.9.2 Cálculo de Volúmenes

El cálculo de volúmenes de sólidos de revolución se realiza mediante la técnica del disco o la técnica del anillo. Por ejemplo, usando el eje x:

$$V = \pi \int_a^b [f(x)]^2 \, dx$$

Ejemplo: Volumen de un Sólido de Revolución

```
        funcion = x**2
        resultado = integrate(funcion**2 * 3.1416, (x, 0, 2))
        print(f"Volumen del s\u00f3lido de revoluci\u00f3n de x^2
             entre 0 y 2: {resultado}")
```

6.16: Volumen de un sólido de revolución

6.10 Cálculo de Integrales Definidas con Python

La biblioteca `SymPy` en Python facilita el cálculo simbólico y numérico de integrales definidas. A continuación, se presentan 10 ejemplos resueltos.

6.10.1 Ejemplo 1: Integral de una Potencia

```
from sympy import symbols, integrate

x = symbols('x')
resultado = integrate(x**3, (x, 0, 2))
print(f"Integral de x^3 desde 0 hasta 2: {resultado}")
```

6.17: Integral de una potencia

6.10.2 Ejemplo 2: Integral Exponencial

```
from sympy import exp

resultado = integrate(exp(x), (x, 1, 3))
print(f"Integral de e^x desde 1 hasta 3: {resultado}")
```

6.18: Integral exponencial

6.11 Aplicación de integrales: acumulación de biomasa

La integral indefinida permite calcular funciones acumulativas. En acuicultura, puede emplearse para estimar la biomasa acumulada en función del tiempo a partir de una tasa de crecimiento conocida.

Aplicación en Acuicultura

Supongamos que la tasa de crecimiento de biomasa de una población de peces está dada por la función:

$$g(t) = 3t^2 + 2t \quad \text{(gramos por día)}$$

donde t es el tiempo en días. Se desea conocer la función que representa la biomasa acumulada desde el día cero.

Problema

Calcular la integral indefinida de $g(t)$ para obtener la función de biomasa acumulada $B(t)$.

Solución Manual

- $\int g(t)\,dt = \int (3t^2 + 2t)\,dt = t^3 + t^2 + C$

La función $B(t) = t^3 + t^2 + C$ representa la biomasa acumulada (en gramos), con C como constante de integración.

Codificación en Python

```
from sympy import symbols, integrate

t = symbols('t')
g = 3*t**2 + 2*t

B = integrate(g, t)
print(f"Función de biomasa acumulada: B(t) = {B} + C")
```

6.19: Cálculo de integral indefinida con SymPy

Reflexión Didáctica

Este tipo de modelos ayuda a predecir el crecimiento acumulado a lo largo del tiempo y a planificar actividades como la cosecha, el manejo de densidades y la alimentación.

Conclusión del capítulo

La integral definida permite resolver una amplia variedad de problemas geométricos y físicos. Con Python, estos conceptos se exploran de forma accesible, consolidando la comprensión del cálculo aplicado.

7 Modelación Matemática en Acuicultura

Objetivos del capítulo

- Comprender los fundamentos de la modelación matemática y su aplicación en problemas reales.
- Aplicar modelos de regresión lineal simple a datos experimentales en acuicultura.
- Utilizar técnicas de optimización para mejorar la eficiencia en el uso de recursos acuícolas.
- Resolver problemas prácticos mediante programación en Python.

7.1 Introducción

La modelación matemática permite representar fenómenos del mundo real mediante expresiones matemáticas, facilitando el análisis, la predicción y la toma de decisiones. En acuicultura, esta herramienta es clave para optimizar procesos como la alimentación, el crecimiento de especies y

la gestión del entorno.

7.2 Regresión Lineal Simple

La regresión lineal simple permite modelar la relación entre dos variables cuantitativas. Por ejemplo, se puede analizar la relación entre el peso de los peces y los días de cultivo.

Ejemplo: Relación peso-tiempo

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Datos simulados: días de cultivo y peso en gramos
dias = np.array([10, 20, 30, 40, 50, 60]).reshape(-1, 1)
peso = np.array([5, 15, 30, 45, 60, 80])

modelo = LinearRegression()
modelo.fit(dias, peso)

pendiente = modelo.coef_[0]
intercepto = modelo.intercept_
print(f"Modelo: peso = {pendiente:.2f} * días +
    {intercepto:.2f}")

# Gráfico
plt.scatter(dias, peso, color='blue', label='Datos')
plt.plot(dias, modelo.predict(dias), color='red',
    label='Recta de ajuste')
plt.xlabel("Días de cultivo")
plt.ylabel("Peso (g)")
plt.legend()
plt.title("Regresión lineal simple: Peso vs Días")
plt.show()
```

7.1: Modelo de regresión lineal simple

7.3 Optimización de Recursos en Acuicultura

La optimización permite encontrar condiciones que maximizan o minimizan una función objetivo, sujeta a ciertas restricciones. Un ejemplo común en acuicultura es optimizar el uso del alimento para maximizar el crecimiento.

Ejemplo: Maximizar crecimiento con restricción de alimento

```
from scipy.optimize import minimize

# Función objetivo: crecimiento en función del alimento (simplificada)
def crecimiento(alimento):
return -(-0.1*alimento**2 + 2*alimento)

resultado = minimize(crecimiento, x0=1, bounds=[(0, 20)])

print(f"Cantidad óptima de alimento (kg/día): {resultado.x[0]:.2f}")
print(f"Crecimiento estimado: {-resultado.fun:.2f} g")
```

7.2: Modelo de crecimiento con restricción de alimento

7.4 Modelación de Temperatura y Crecimiento

La temperatura del agua influye en el metabolismo de los peces. Podemos modelar esta relación para predecir el crecimiento según temperatura promedio.

```
temperatura = np.array([20, 22, 24, 26, 28,
    30]).reshape(-1, 1)
crecimiento = np.array([100, 120, 150, 170, 160, 140])

modelo = LinearRegression()
modelo.fit(temperatura, crecimiento)

plt.scatter(temperatura, crecimiento, color='green',
    label='Datos')
plt.plot(temperatura, modelo.predict(temperatura),
    color='black', label='Modelo')
plt.xlabel("Temperatura (\\textdegree C)")
plt.ylabel("Crecimiento (g/mes)")
plt.title("Relación entre temperatura y crecimiento")
plt.legend()
plt.show()
```

7.3: Modelo de temperatura y crecimiento

7.5 Modelación matemática: estimación de crecimiento con visualización

La modelación matemática permite describir y predecir fenómenos naturales mediante funciones. En acuicultura, es clave para proyectar el crecimiento de los peces a lo largo del tiempo.

Aplicación en Acuicultura

Se ha observado que el crecimiento de una población de tilapias puede aproximarse con la función:

$$P(t) = \frac{5}{1+4e^{-0,6t}} \quad \text{(kg por pez)}$$

donde t es el tiempo en semanas. Esta es una función logística típica en crecimiento poblacional.

Problema

1. Calcular el peso estimado por pez en la semana 10.
2. Graficar la función de crecimiento para $t \in [0, 20]$.

Solución Manual

1. Sustituimos en la función:
$P(10) = \frac{5}{1+4e^{-6}} \approx \frac{5}{1+4(0,0025)} = \frac{5}{1,01} \approx 4,95$ kg.

Codificación en Python

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

t = np.linspace(0, 20, 100)
P = 5 / (1 + 4 * np.exp(-0.6 * t))

plt.plot(t, P)
plt.title("Crecimiento de tilapia")
plt.xlabel("Semanas")
plt.ylabel("Peso (kg)")
plt.grid(True)
plt.show()

# Cálculo en t = 10
peso_10 = 5 / (1 + 4 * np.exp(-0.6 * 10))
print(f"Peso estimado en la semana 10: {peso_10:.2f} kg")
```

7.4: Modelación del crecimiento logístico

Reflexión Didáctica

Este ejemplo integra cálculo, funciones exponenciales y visualización para interpretar datos reales. Fomenta el pensamiento sistémico y el análisis predictivo aplicado a procesos biológicos.

7.6 Proyección futura del uso de Python en la educación STEM

La integración de Python en el ámbito educativo ha trascendido la enseñanza básica de programación, consolidándose como una herramienta fundamental en la educación

STEM (Ciencia, Tecnología, Ingeniería y Matemáticas). Esta tendencia se ha intensificado con el avance de tecnologías emergentes como la inteligencia artificial, el análisis de datos y la modelación matemática, disciplinas donde Python ocupa un lugar privilegiado por su sintaxis simple y su robusto ecosistema de bibliotecas (Oliphant, 2006; VanderPlas, 2016).

Diversos estudios destacan el potencial de Python como puente entre la teoría matemática y la práctica computacional, fomentando un aprendizaje activo, visual e interdisciplinario (Zayas-Batista, 2023). En particular, bibliotecas como SymPy, NumPy y Matplotlib permiten abordar conceptos abstractos como funciones, derivadas o integrales desde una perspectiva interactiva, mejorando la comprensión y el pensamiento computacional.

En el contexto de la educación matemática superior, Python facilita el desarrollo de competencias clave del siglo XXI como la resolución de problemas, el análisis de datos y la alfabetización digital. La tendencia apunta hacia la adopción de entornos de trabajo reproducibles como Jupyter Notebooks, así como la incorporación de proyectos colaborativos y evaluación automatizada (Martínez & López, 2019).

La consolidación de Python como lenguaje transversal en disciplinas STEM sugiere que su uso en entornos educativos seguirá en aumento, apoyando una enseñanza más contextualizada, abierta y orientada a la práctica.

Conclusión del capítulo

La modelación matemática, aplicada al contexto acuícola, permite extraer conclusiones valiosas a partir de datos y tomar decisiones informadas. Mediante regresión lineal y optimización, es posible predecir comportamientos y mejorar procesos como la alimentación y el crecimiento de especies. Python facilita esta tarea al proporcionar herramientas accesibles para el análisis numérico y visualización.

Fundamento teórico y enfoque pedagógico del libro

La enseñanza de las matemáticas en educación superior enfrenta hoy el desafío de combinar solidez conceptual con herramientas tecnológicas que favorezcan la comprensión y motivación de los estudiantes. Este libro propone el uso del lenguaje de programación Python como recurso didáctico para fortalecer la formación matemática, integrando cálculos simbólicos, gráficos interactivos y automatización de procesos algebraicos.

Diversas investigaciones respaldan este enfoque. Por ejemplo, se ha demostrado que la integración de asistentes matemáticos (como los que se pueden construir con Python y SymPy) mejora significativamente la comprensión de conceptos clave en carreras técnicas (Zayas-Batista, 2023). Además, la incorporación de tecnologías móviles y cuestionarios interactivos en clase ha sido identificada como una estrategia eficaz para aumentar la motivación y mejorar el rendimiento de los estudiantes en matemáticas (Masero Moreno, Sánchez-García & Pérez Rodríguez, 2019).

Desde un punto de vista didáctico, el desarrollo de la creatividad ha sido identificado como un objetivo esencial de la educación matemática contemporánea. Promover entornos donde los estudiantes construyan conocimiento mediante resolución de problemas, exploración gráfica y modelación numérica es una necesidad pedagógica ampliamente documentada (Arteaga Valdés & Suárez Padrón, 2016).

Asimismo, los enfoques de enseñanza problémica centrados en la construcción activa del conocimiento por parte del estudiante son fundamentales en el contexto universitario actual (Valdés Capote & González Pérez, 2018). En este libro, se propone una secuencia de actividades y ejercicios que estimulan el razonamiento lógico y la toma de decisiones, aprovechando las capacidades de automatización y visualización que ofrece Python.

Finalmente, se reconoce la necesidad de repensar el uso de la informática en la clase de matemáticas. Las propuestas tradicionales ya no son suficientes para responder a los contextos actuales, y es imprescindible incorporar nuevas herramientas tecnológicas de forma significativa y pedagógicamente fundamentada (Vargas Quiñonez, 2017).

Este enfoque no solo responde a las demandas metodológicas modernas, sino que también ha sido puesto en práctica y retroalimentado en escenarios reales de aula, como se detalla en la siguiente sección.

Aplicación del material en el aula y reflexión metodológica

El contenido de este libro fue concebido y validado en el contexto real de clases impartidas por el autor en la carrera de Acuicultura de la Facultad de Ciencias Agropecuarias de la Universidad Técnica de Machala. Esta experiencia permitió probar de manera directa los capítulos, ejemplos y actividades propuestas, dentro de un entorno de formación profesional en ciencias aplicadas.

El uso de Python en las clases de matemáticas permitió abordar temas complejos como funciones, derivadas, integrales y modelamiento de datos de forma más visual, interactiva y comprensible. Las prácticas con herramientas como SymPy, Matplotlib y Pandas se integraron a sesiones de clase habituales, donde los estudiantes desarrollaron competencias tanto matemáticas como digitales.

Durante el proceso, se observó una mejora notable en la participación de los estudiantes y en su comprensión conceptual, especialmente al trabajar con visualización gráfica de funciones y simulación de datos. Además, el enfoque

práctico y contextualizado al área de acuicultura resultó motivador y útil para los estudiantes, que lograron ver la utilidad directa del cálculo en su futura profesión.

Como resultado de esta implementación, se realizaron ajustes al material inicial, incluyendo más ejemplos guiados, problemas contextualizados a biología marina y ejercicios enfocados en la interpretación de gráficos y datos reales. Esta retroalimentación directa del aula ha contribuido a enriquecer la calidad didáctica del libro.

Referencias bibliográficas

Apostol, T. M. (1967). *Calculus, volumes 1 and 2*. Wiley.

Arteaga Valdés, M., & Suárez Padrón, C. (2016). Creatividad y educación matemática: Perspectivas didácticas. *Revista Latinoamericana de Educación Matemática*, 24(3), 35–54.

Díaz, J., & Rojas, P. (2020). *Matemáticas y computación: Un enfoque integrado para la enseñanza del cálculo*. Editorial Académica.

Downey, A. (2012). *Think Python: How to think like a computer scientist*. Green Tea Press.

Gómez, L. (2022). El papel del laboratorio computacional en la enseñanza del cálculo. *Revista de Innovación Educativa*, 40(1), 12–28.

Guzdial, M., & Ericson, B. (2013). *Introduction to computing and programming in Python*. Pearson.

Harrington, P. (2016). *Machine learning in action*. Manning Publications.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95.

Knuth, D. E. (1997). *The art of computer programming, Vol. 1: Fundamental algorithms* (3rd ed.). Addison-Wesley.
Leithold, L. (1976). *The calculus 7*. HarperCollins.
Lutz, M. (2013). *Learning Python*. O'Reilly Media.
Martínez, A., & López, C. (2019). El impacto de las herramientas tecnológicas en la educación matemática. *Revista de Educación Matemática*, 35(2), 45–62.
Masero Moreno, M. A., Sánchez-García, J. A., & Pérez Rodríguez, J. (2019). Cuestionarios interactivos en dispositivos móviles como herramienta de motivación en matemáticas. *Pixel-Bit. Revista de Medios y Educación*, 55, 53–68.
McConnell, S. (2004). *Code complete: A practical handbook of software construction* (2nd ed.). Microsoft Press.
Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., ... & Granger, B. E. (2017). SymPy: Symbolic computing in Python. *PeerJ Computer Science*, 3, e103.
Montgomery, D. C., & Runger, G. C. (2018). *Applied statistics and probability for engineers*. Wiley.
Nelli, F. (2015). *Python data analytics*. Apress.
Oliphant, T. E. (2006). *A guide to NumPy*. Trelgol Publishing.
Organización de las Naciones Unidas para la Agricultura y la Alimentación. (2020). *El estado mundial de la pesca y la acuicultura 2020*. FAO.
Pérez, R. (2021). *Computación aplicada a las matemáti-

cas: Un enfoque práctico*. Ediciones Científicas.
Pita, R. (2015). *Cálculo de una variable*. Prentice.
Python Software Foundation. (2025). *Python downloads*. https://www.python.org/downloads/
Seber, G. A. F., & Lee, A. J. (2012). *Linear regression analysis*. Wiley.
Spivak, M. (1994). *Calculus*. Publish or Perish.
Stewart, J. (2015). *Cálculo*. Cengage Learning.
Valdés Capote, M., & González Pérez, A. (2018). Enseñanza problémica en la universidad: Una vía para la construcción del conocimiento. *Educación y Sociedad*, 20(2), 67–80.
Van Rossum, G. (1991). *Python programming language*. Python Software Foundation.
VanderPlas, J. (2016). *Python data science handbook*. O'Reilly Media.
Vargas Quiñonez, R. (2017). La integración de TIC en la enseñanza de las matemáticas: Un enfoque pedagógico. *Revista Colombiana de Educación*, 73, 101–120.
Zayas-Batista, A. (2023). El uso de asistentes matemáticos en la enseñanza de cálculo diferencial. *Revista Iberoamericana de Tecnología Educativa*, 19(2), 34–45.

ISBN: 978-9942-53-054-7

Compás
capacitación e investigación