Proyecto de laboratorio de Sistemas Dinamicos

FACULTAD DE MINAS
GEOLOGÍA Y CIVIL

3 de julio de 2023

MA-547.: ELEMENTOS DE LOS SISTEMAS DINÁMICOS.

MOROTE RODRÍGUEZ Alberto Brayan.
ROJAS TABOADA, Hernan

UNIVERSIDAD NACIONAL DE SAN CRISTOBAL DE HUAMANGA

Proyecto de laboratorio de Sistemas Dinamicos

FACULTAD DE MINAS
GEOLOGÍA Y CIVIL
3 de julio de 2023

by

MOROTE RODRÍGUEZ Alberto Brayan.
ROJAS TABOADA, Hernan

Student Name	Student Number
Name	Number
(...)	

Docente:	CONDORI HUAMAN, Alexander Paul
Institution:	Univerdiad nacional de San Cristobal de Huamanga
curso	MA-547.: ELEMENTOS DE LOS SISTEMAS DINÁMImos
Project Duration:	10 de enero 2023 - 3 de julio de 2023

Ayacucho-Peru

Dedicatoria

Estimado profesor Alexander Paul CONDORI HUAMAN

Nos dirijamos a usted en este momento con la esperanza de que pueda escucharnos la petición y otorgarnos su aprobación en el curso. Como estudiantes del último año, hemos trabajado arduamente durante los últimos meses en nuestra investigación en matemáticas y programación, aplicando los conocimientos que hemos adquirido en su curso.

No puedo negar que ha sido un camino desafiante, pero estoy seguro de que hemos aprendido mucho gracias a su dedicación y enseñanza. Por esta razón, quisiéramos suplicarle que tenga en cuenta el esfuerzo que he dedicado a este curso y me brinde la oportunidad de aprobarlo.

Agradezco de antemano su consideración y espero que tenga en cuenta el esfuerzo dedicado al trabajo de laboratorio.
Atentamente,

MOROTE RODRÍGUEZ Alberto Brayan y ROJAS TABOADA Hernan

Índice general

Summary I

Índice de figuras III

Índice de cuadros IV

1 Uso de lenguajes de progrmacion para aplicaciones matematicas 2

2 Python 3
2.1 Leguaje de Programación Python . . . 3
2.1.1 ¿ CÓMO INSTALAR PYTHON? . . . 3
2.2 Entornos de desarrollo en python . . . 4
2.3 Que es anaconda. . . . 4
2.3.1 ¿ Como instalar anaconda? . . . 5
2.4 Colab de Google . . . 5
2.4.1 Como usar colab de google . . . 5
2.5 Librerías de Python. . . . 6
2.6 Numpy . . . 6
2.6.1 Operaciones matriciales en Numpy . . . 7
2.7 Librerias para graficar en Python . . . 8
2.8 Sympy. . . . 10
2.8.1 Calculo de operaciones con Sympy . . . 10
2.8.2 Generacion de graficas en Sympy. . . . 12
2.8.3 Resolución de una ecuacion diferencial ordinaria de primer orden con sympy . . . 13
2.8.4 Resolución de una ecuacion diferencial ordinaria de segundo orden con sympy . . 14
2.8.5 Resolución de un sistema de ecuaciones diferenciales ordinarias con sympy . . . 15
2.9 Scipy . . . 16
2.9.1 EJEMPLOS CON SCIPY. . . . 16
2.10 Modulo Odeint . . . 17

3 autovalores, autovectores, diagonalizacion de matrices 21
3.1 Autovalores . . . 21
3.2 Autovectores . . . 21
3.3 Diagonalizacion de Matrices . . . 22

4 Matrices exponenciales 24

5 Formas canonicas de Jordan 26

6 Retratos de fase 28
6.1 Librerias para hallar retratos de fase con python . . . 28
6.1.1 ejemplo: . . . 28

7 Conclusion 32

8 Referencias bibliográficas 33

Índice de figuras

2.1. Entorno Anaconda o Conda 5
2.2. Colab tiene gran integracion Python 6
2.3. Grafica en 2D 9
2.4. Grafica en 3D 10
2.5. Grafica de sympy 13
2.6. Gráfica de Ecuación diferencial de segundo orden de un resorte 18
2.7. Ecuación diferencial de orden superior 19
2.8. Atractor de Lorenz 20

6.1. Solucion con variaciones de a y b 29
6.2. grafica de un sistema 30
6.3. grafico de un sistema EDO 31

Índice de cuadros

1.1. Comparación entre Python y MATLAB 2
2.1. Comparación de los principales entornos de programación en Python 4

Introducción

Python es un lenguaje de programación utilizado para desarrollar aplicaciones de cualquier tipo. Se trata de un lenguaje interpretativo, lo que quiere decir que no es necesario transformarlo para procesar las aplicaciones escritas en Python. Este se ejecuta directamente por el ordenador empleando un programa llamado interpretador, por lo que no es necesario convertirlo al lenguaje máquina.
Python fue creado por Guido van Rossum, un programador holandés a finales de los 80 y principio de los 90 cuando se encontraba trabajando en el sistema operativo Amoeba. Primariamente se concibe para manejar excepciones y tener interfaces con Amoeba como sucesor del lenguaje ABC.
El 16 de octubre del 2000 se lanza Python 2.0 que con tenía nuevas características como completa recolección de basura y completo soporte a Unicode. Pero el mayor avance lo constituye que este comenzó a ser verdaderamente desarrollado por la comunidad, bajo la dirección de Guido.
El Python 3.0 es una versión mayor e incompatible con las anteriores en muchos aspectos, que llega después de un largo período de pruebas el 3 de diciembre del 2008. Muchas de las características introducidas en la versión 3 han sido compatibilizadas en la versión 2.6 para hacer de forma más sencilla la transición entre estas.
A Guido van Rossum le fue otorgado el Free Software Award (Premio del Software Libre) en el 2001, por sus trabajos en la creación y desarrollo del lenguaje Python. En el 2005 fue contratado por Google, donde trab aja en la actualidad, aunque sigue liderando los esfuerzos en el desarrollo del Python.

Capítulo 1

Uso de lenguajes de progrmacion para aplicaciones matematicas

El uso de lenguajes de programación para aplicaciones en matemáticas se refiere a la utilización de software para resolver problemas matemáticos complejos. Los lenguajes de programación, como Python, MATLAB, R, entre otros, ofrecen una amplia variedad de herramientas y bibliotecas para realizar cálculos numéricos, modelar sistemas matemáticos, visualizar datos, y mucho más.

Con la ayuda de estos lenguajes de programación, los matemáticos pueden automatizar tareas repetitivas, analizar grandes conjuntos de datos, simular sistemas complejos, y desarrollar modelos matemáticos precisos y efectivos. Además, los lenguajes de programación también permiten a los matemáticos colaborar y compartir sus códigos y resultados con la comunidad científica.

En resumen, el uso de lenguajes de programación en aplicaciones en matemáticas permite a los matemáticos resolver problemas de manera más eficiente y efectiva, ahorrando tiempo y esfuerzo, y proporcionando soluciones precisas y confiables.

Cuadro comparativo entre Python y MATLAB en aplicaciones matemáticas, hecho en LaTeX:

Cuadro 1.1: Comparación entre Python y MATLAB

Característica	**Python**	**MATLAB**
Lenguaje	Interpretado y compilado	Interpretado
Sintaxis	Concisa y clara	Verbosa
Tipos de datos	Dinámico	Estático
Visualización	Completa y personalizable	Limitada y no personalizable
Paquetes y bibliotecas	Amplia selección	Limitada selección
Licencia	Gratis y de código abierto	Requiere licencia
Comunidad	Activa y creciente	Estable y madura
Documentación	Amplia y actualizada	Limitada y no siempre actualizada
Curva de aprendizaje	Empinada	Suave
Desempeño	Rápido para cálculos numéricos	Lento para grandes conjuntos de datos

Este cuadro comparativo muestra las principales diferencias entre Python y MATLAB en términos de lenguaje, sintaxis, tipos de datos, visualización, paquetes y bibliotecas, licencia, comunidad, documentación, curva de aprendizaje y desempeño.

Capítulo 2

Python

2.1. Leguaje de Programación Python

¿Qué es python?

Python es un lenguaje de programación interpretado de tipado dinámico cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.
Python, es un programa fácil y sencillo para la programación, con el cual será más fácil poder guiarlos con la programación a comparación a otros software comerciales como Matlab, Math ematica, etc. A demás de ello python es un sotfware libre fácil y disponible para todos(?, ?).

Características

- **Interpretado**: Se ejecuta sin necesidad de ser procesado por el compilador y se detectan los errores en tiempo de ejecución.
- **dinámico**: Las variables se comprueban en tiempo de ejecución.
- **Multiplataforma**: disponible para plataformas de Windows, Linux o MAC.
- **Multiparadigma**: Soporta programación funcional, programación imperativa y programación orientada a objetos.

Python, es el tercer lenguaje de programación más popular del mundo, y ha tenido un rápido crecimiento durante los últimos años

Objetivos Generales:

- Programar mediante algoritmos la inversa de una matriz.
- hacer un algoritmo para calcular el determinante de una matriz.
- Aprender a usar "Python", un sotfware nuevo para todos.

Objetivos Específicos:

Desarrollar algoritmos que permitan hacer un programa que grafique el retrato de face del sistema lineal y como se comportan las órbitas.

2.1.1. ¿ CÓMO INSTALAR PYTHON?

¿ CÓMO INSTALAR PYTHON?

Para instalar Python en Windows, siga estos sencillos pasos:

1. Descargue la última versión de Python desde el sitio web oficial de Python: https://www.python.org/downloads/.
2. Ejecute el archivo de instalador descargado y haga clic en "Next".

3. Seleccione Ïnstall for all usersz haga clic en ”Next”. Si prefiere instalar Python solo para su usuario, seleccione Ïnstall just for me”.
4. Seleccione la ruta de instalación deseada y haga clic en ”Next”. Se recomienda dejar la ruta predeterminada.
5. Seleccione .Add Python 3.x to PATHz haga clic en ”Next”. Esta opción agrega Python a la variable de entorno PATH, lo que permite ejecutar Python desde cualquier lugar en el sistema.
6. Haga clic en Ïnstall”para iniciar la instalación. Una vez finalizada la instalación, haga clic en Çlose”.

Una vez finalizado este proceso, puede abrir la línea de comandos de Windows y verificar la versión de Python instalada ejecutando el siguiente comando:

2.2. Entornos de desarrollo en python

cuadro comparativo que describe las características y ventajas de los principales entornos de programación en Python:

Cuadro 2.1: Comparación de los principales entornos de programación en Python

Entorno	Características	Ventajas	Desventajas
IDLE	Simple y fácil de usar	Ligero y rápido	Limitado en características
PyCharm	Potente y completo	Excelente depurador	Requiere recursos del sistema
Jupyter Notebook	Interactivo y colaborativo	Visualización de datos	No es un entorno de desarrollo completo
Spyder	Entorno científico y de datos	Integración de herramientas	Lentitud en grandes conjuntos de datos
Visual Studio Code	Personalizable y extensible	Amplia selección de extensiones	Requiere configuración para la programación científica
Anaconda	Paquetes científicos integrados	Facilidad de instalación y gestión	Requiere mucho espacio en el disco

Este cuadro comparativo muestra las características y ventajas de los principales entornos de programación en Python, como IDLE, PyCharm, Jupyter Notebook, Spyder, Visual Studio Code y Anaconda. Se destacan sus características principales, así como las ventajas y desventajas de cada uno.

2.3. Que es anaconda

Anaconda es un entorno de desarrollo de Python que incluye una amplia variedad de paquetes y herramientas para el análisis de datos, la inteligencia artificial y el aprendizaje automático, la ciencia computacional y el desarrollo de aplicaciones en Python. Anaconda es una distribución autónoma de Python que incluye la mayoría de las librerías y herramientas necesarias para la mayoría de los proyectos de ciencia de datos y aprendizaje automático(?, ?).

Anaconda se destaca por su gestor de paquetes conda, que permite la instalación y gestión de paquetes de Python y de otros lenguajes de programación, así como de entornos virtuales separados para diferentes proyectos. Además, Anaconda incluye una consola Jupyter, que permite la creación y ejecución de cuadernos interactivos para el análisis de datos y la visualización de resultados.

En resumen, Anaconda es un entorno de desarrollo de Python que ofrece una gran cantidad de paquetes y herramientas para el análisis de datos y el aprendizaje automático, lo que lo hace una opción popular para los profesionales de la ciencia de datos y los desarrolladores de Python.

Figura 2.1: Entorno Anaconda o Conda

2.3.1. ¿ Como instalar anaconda?

¿ CÓMO INSTALAR ANACONDA?

Para instalar Python en Windows, siga estos sencillos pasos:
Anaconda es una distribución popular de Python que incluye muchas de las bibliotecas y herramientas más utilizadas en ciencia de datos y aprendizaje automático. Para instalar Anaconda en Windows, siga estos sencillos pasos:

1. Descargue la última versión de Anaconda desde el sitio web oficial de Anaconda: https://www.anaconda.com/products/distribution.
2. Ejecute el archivo de instalador descargado y haga clic en "Next".
3. Acepte los términos de licencia y haga clic en "Next".
4. Seleccione la ruta de instalación deseada y haga clic en "Next". Se recomienda dejar la ruta predeterminada.
5. Seleccione "Just Me"si desea instalar Anaconda solo para su usuario, o seleccione .All Users"si desea instalar Anaconda para todos los usuarios del sistema.
6. Haga clic en Ïnstall"para iniciar la instalación. Una vez finalizada la instalación, haga clic en "Next".
7. Seleccione .Add Anaconda to my PATH environment variablez haga clic en "Next". Esta opción agrega Anaconda a la variable de entorno PATH, lo que permite ejecutar comandos Anaconda desde cualquier lugar en el sistema.

2.4. Colab de Google

Google Colab es un entorno de desarrollo en línea gratuito y basado en la nube que permite a los usuarios programar en Python y otros lenguajes de programación. Ofrece recursos de cálculo potentes y almacenamiento en la nube, lo que lo hace ideal para el aprendizaje, la investigación y la colaboración en proyectos de ciencia de datos y aprendizaje automático. Con Google Colab, los usuarios pueden escribir y ejecutar código, visualizar resultados y guardar y compartir sus proyectos en línea. Todo esto se puede hacer desde cualquier dispositivo con conexión a Internet, sin la necesidad de instalar software adicional.

2.4.1. Como usar colab de google

Google Colab para programar en Python:

1. Ir a la página de Google Colab en https://colab.research.google.com/.
2. Haga clic en el botón "Nuevo archivo de Jupyter.en la esquina superior derecha.
3. Seleccione "Python 3çomo el tipo de archivo para crear.

Figura 2.2: Colab tiene gran integracion Python

4. Escriba el código Python en el cuaderno de Jupyter.
5. Ejecute el código haciendo clic en el botón .Ejecutar.º usando el atajo de teclado ”Shift + Enter”.
6. Almacene los cambios haciendo clic en .Archivoz luego en ”Guardar”.
7. Comparta el cuaderno con otros usuarios haciendo clic en Çompartir.en la esquina superior derecha.
8. Descargue el cuaderno como un archivo ”.ipynb.º ”.py”haciendo clic en ”Descargar.en la esquina superior derecha.

2.5. Librerías de Python

Las bibliotecas de Python son un conjunto de herramientas y módulos que se pueden importar y utilizar en los programas de Python para agregar funcionalidades adicionales. Hay una gran cantidad de bibliotecas disponibles en Python, cada una con un enfoque específico en tareas como el análisis de datos, el aprendizaje automático, la visualización de datos, el desarrollo de aplicaciones web, entre otros.

Una de las bibliotecas más populares y ampliamente utilizadas en Python es NumPy, que proporciona una amplia gama de funciones matemáticas y herramientas para trabajar con matrices y arreglos multidimensionales. Otro paquete popular en Python es Pandas, que proporciona estructuras de datos y herramientas de análisis de datos avanzadas.

En el campo del aprendizaje automático, Python cuenta con bibliotecas como scikit-learn, que proporciona una amplia gama de algoritmos de aprendizaje supervisado y no supervisado, y TensorFlow, una biblioteca de aprendizaje profundo y computación numérica desarrollada por Google.

Para la visualización de datos, Matplotlib es una de las bibliotecas más populares en Python, que permite crear gráficos y visualizaciones de datos en 2D y 3D. Otros paquetes de visualización de datos populares en Python incluyen Seaborn y Plotly.

En el desarrollo de aplicaciones web, Django es una de las bibliotecas más populares en Python, que proporciona un marco de desarrollo de aplicaciones web completo y de alto nivel. Flask es otro marco de desarrollo de aplicaciones web de código abierto que es popular en la comunidad de Python.

En resumen, las bibliotecas de Python son una parte esencial del lenguaje y amplían las capacidades y funcionalidades de Python, lo que lo hace una herramienta valiosa para una amplia gama de aplicaciones y tareas.

2.6. Numpy

NumPy es una biblioteca de Python que se especializa en el cálculo numérico y la manipulación de arrays multidimensionales. Fue creada en 2005 y se ha convertido en uno de los paquetes más populares y ampliamente utilizados en Python debido a sus funciones potentes y eficientes para realizar operaciones matemáticas en grandes conjuntos de datos.

Uno de los principales componentes de NumPy es el objeto de array de NumPy, que proporciona una forma eficiente de representar y manipular matrices y arreglos multidimensionales. Estos arrays se pueden utilizar para realizar cálculos matemáticos y estadísticos complejos en grandes conjuntos de datos con una sola línea de código.

Además de los arrays de NumPy, la biblioteca también proporciona una amplia gama de funciones matemáticas y estadísticas, como la resolución de sistemas lineales, la integración numérica, el cálculo de la transformada de Fourier y la optimización. Todas estas funciones se pueden aplicar directamente a los arrays de NumPy, lo que hace que el cálculo numérico en Python sea muy eficiente y fácil de usar.

En términos de aplicaciones matemáticas, NumPy es ampliamente utilizado en el análisis de datos y la estadística, la optimización y la resolución de sistemas lineales. También es común en el ámbito de la investigación en ciencia de datos y aprendizaje automático, donde se utiliza para procesar y manipular grandes conjuntos de datos.

NumPy también se integra fácilmente con otras bibliotecas de Python, como Matplotlib y Pandas, para crear soluciones más complejas y potentes. Por ejemplo, se pueden utilizar arrays de NumPy con Matplotlib para crear gráficos y visualizaciones de datos en 2D y 3D, y con Pandas para analizar y manipular grandes conjuntos de datos en formato tabular.

Además, NumPy es una biblioteca muy eficiente en términos de tiempo de ejecución, ya que está escrita en lenguaje de bajo nivel, como C y Fortran, y está optimizada para realizar cálculos numéricos de manera rápida y eficiente. Esto la hace ideal para aplicaciones que requieren cálculos intensivos en tiempo real, como la simulación científica o el aprendizaje automático en grandes conjuntos de datos

2.6.1. Operaciones matriciales en Numpy

A continuación se describen algunas de las principales operaciones de matrices en NumPy:

1. Creación de matrices: Con NumPy, se pueden crear matrices utilizando la función numpy.array(). Por ejemplo, se puede crear una matriz de ceros con el siguiente código:

```
import numpy as np
matrix = np.zeros((3, 3))
```

2. Indexación y selección de elementos: NumPy permite acceder a los elementos individuales de una matriz mediante indexación y selección. Por ejemplo, se puede acceder al primer elemento de una matriz con el siguiente código:

```
import numpy as np
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
first_element = matrix[0, 0]
```

3. Transposición de matrices: La transposición de una matriz se refiere a la rotación de su estructura 90 grados en el sentido de las agujas del reloj. NumPy proporciona la función numpy.transpose() para realizar esta operación. Por ejemplo, se puede transponer una matriz con el siguiente código:

```
import numpy as np
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
transposed_matrix = np.transpose(matrix)
```

4. Suma y multiplicación de matrices: NumPy permite realizar operaciones aritméticas básicas, como la suma y la multiplicación de matrices. Por ejemplo, se pueden sumar dos matrices con el siguiente código:

```
import numpy as np
matrix1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
matrix2 = np.array([[9, 8, 7], [6, 5, 4], [3, 2, 1]])
sum_matrix = matrix1 + matrix2
```

5. Producto matricial: El producto matricial se refiere a la multiplicación de dos matrices en un orden específico. NumPy proporciona la función numpy.dot() para realizar esta operación. Por ejemplo, se pueden multiplicar dos matrices con el siguiente código:

```
import numpy as np
matrix1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
matrix2 = np.array([[9, 8, 7], [6, 5, 4], [3, 2, 1]])
product_matrix = np.dot(matrix1, matrix2)
```

6. Inversa de matrices:La inversa de una matriz se refiere a otra matriz que, cuando se multiplica por la matriz original, resulta en la matriz identidad. NumPy proporciona la función numpy.linalg.inv() para calcular la inversa de una matriz. Por ejemplo, se puede calcular la inversa de una matriz con el siguiente código:

```
import numpy as np
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
inverse_matrix = np.linalg.inv(matrix)
```

7. Determinante de matrices: El determinante de una matriz es un número que representa la magnitud y el signo de una transformación lineal. NumPy proporciona la función numpy.linalg.det() para calcular el determinante de una matriz. Por ejemplo, se puede calcular el determinante de una matriz con el siguiente código:

```
import numpy as np
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
determinant = np.linalg.det(matrix)
```

2.7. Librerias para graficar en Python

¿ QUE LIBRERIAS SE UTILIZAN EN PYTHON PARA GRAFICAR?

Hay muchas librerías en Python que se pueden utilizar para gráficar, algunas de las más comunes son:

Matplotlib: es la librería más antigua y popular para gráficos en Python, es muy versátil y permite crear una amplia variedad de gráficos y visualizaciones.

Seaborn: es una librería de visualización de datos basada en Matplotlib, proporciona una interfaz más fácil de usar y estilos de gráficos más atractivos.

Plotly: es una librería de visualización de datos interactiva que permite crear gráficos dinámicos y de alta calidad.

Bokeh: es otra librería de visualización de datos interactiva que permite crear gráficos para la web.

ggplot: es una librería basada en ggplot2 de R, proporciona una sintaxis fácil de usar para crear gráficos complejos.

Estas son solo algunas de las opciones disponibles, la elección de la librería adecuada depende de las necesidades específicas de cada proyecto.

¿ CÓMO GRAFICAR EN DOS DIMENCIONES EN PYTHON?

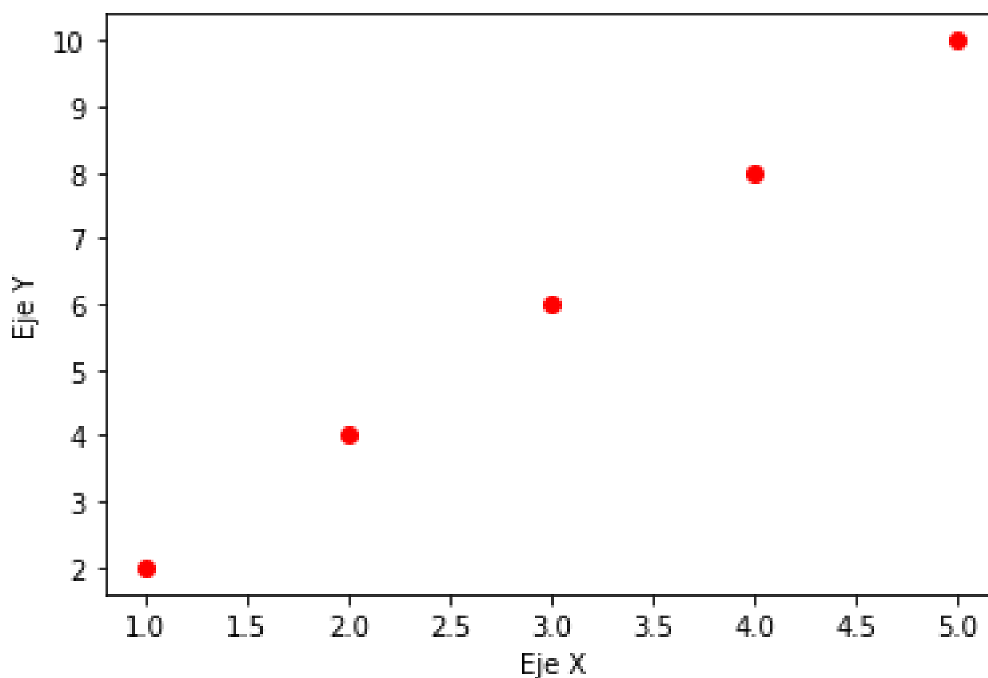
se puede utilizar la librería Matplotlib. Aquí hay un ejemplo de código para graficar puntos en dos dimensiones:

Este código creará un gráfico de línea con los puntos ”(1, 2)”, ”(2, 4)”, ”(3, 6)”, ”(4, 8)¿ ”(5, 10)”marcados con círculos rojos.

```
import matplotlib.pyplot as plt

# Datos a graficar
x = [1, 2, 3, 4, 5]
```

```
y = [2, 4, 6, 8, 10]

# Crear gráfico
plt.plot(x, y, 'ro')

# Agregar etiquetas a los ejes
plt.xlabel('Eje X')
plt.ylabel('Eje Y')

# Mostrar gráfico
plt.show()
```

Figura 2.3: Grafica en 2D



¿ CÓMO GRAFICAR EN TRES DIMENCIONES EN PYTHON?

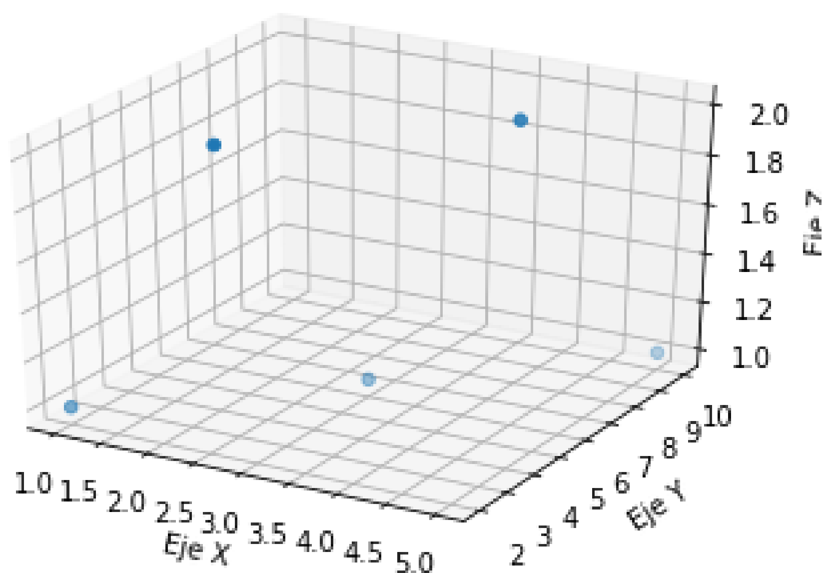
Para graficar en tres dimensiones en Python se puede utilizar la librería Matplotlib. Aquí hay un ejemplo de código para graficar puntos en tres dimensiones:

Este código creará un gráfico en 3D con los puntos ”(1, 2, 1)”, ”(2, 4, 2)”, ”(3, 6, 1)”, ”(4, 8, 2)z ”(5, 10, 1)”.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Datos a graficar
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
z = [1, 2, 1, 2, 1]

# Crear gráfico en 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

```
# Agregar datos al gráfico
ax.scatter(x, y, z)

# Agregar etiquetas a los ejes
ax.set_xlabel('Eje X')
ax.set_ylabel('Eje Y')
ax.set_zlabel('Eje Z')

# Mostrar gráfico
plt.show()
```

Figura 2.4: Grafica en 3D



2.8. Sympy

SymPy es una biblioteca de código abierto de Python para matemáticas simbólicas. Es una biblioteca de cálculo simbólico que permite realizar cálculos matemáticos complejos, simplificar expresiones y resolver ecuaciones. SymPy también proporciona herramientas para la manipulación y visualización de expresiones matemáticas, y se integra fácilmente con otras bibliotecas de cálculo numérico y gráfico.

SymPy es utilizado en una amplia variedad de campos, incluyendo física, ingeniería, finanzas, y economía, y es útil para docentes y estudiantes que necesitan realizar cálculos matemáticos complejos. Con SymPy, se pueden crear, manipular y visualizar expresiones matemáticas en forma exacta, en lugar de aproximada. Además, se pueden resolver sistemas de ecuaciones, simplificar expresiones complejas, integrar y diferenciar funciones, entre muchas otras tareas matemáticas.

2.8.1. Calculo de operaciones con Sympy

Aquí hay algunos ejemplos de código en Python que muestran cómo resolver operaciones matemáticas utilizando la biblioteca SymPy:

```
import sympy

# Resolviendo una ecuación lineal
x = sympy.symbols('x')
```

```
ecuacion = x + 2
solucion = sympy.solve(ecuacion, x)
print(solucion)

# Simplificando una expresión matemática
a, b, c = sympy.symbols('a b c')
expresion = (a + b)**2
simplificado = sympy.simplify(expresion)
print(simplificado)

# Integrando una función
x = sympy.symbols('x')
funcion = sympy.exp(x)
integral = sympy.integrate(funcion, x)
print(integral)

# Diferenciando una función
x = sympy.symbols('x')
funcion = sympy.sin(x)
derivada = sympy.diff(funcion, x)
print(derivada)
# Resolviendo un sistema de ecuaciones lineales
x, y = sympy.symbols('x y')
ecuacion1 = x + y - 3
ecuacion2 = x - y + 1
solucion = sympy.solve((ecuacion1, ecuacion2), (x, y))
print(solucion)

# Calculando la inversa de una matriz
matriz = sympy.Matrix([[1, 2], [3, 4]])
inversa = matriz.inv()
print(inversa)

# Evaluando una función en un punto específico
x = sympy.symbols('x')
funcion = sympy.sin(x)
evaluacion = funcion.evalf(subs={x: 2})
print(evaluacion)

# Expandiendo una expresión matemática
a, b, c = sympy.symbols('a b c')
expresion = (a + b + c)**3
expandido = sympy.expand(expresion)
print(expandido)

# Factorizando una expresión matemática
x = sympy.symbols('x')
expresion = x**3 + x**2 - x - 1
factorizado = sympy.factor(expresion)
print(factorizado)

# Derivando una función
x = sympy.symbols('x')
funcion = sympy.sin(x)
derivada = sympy.diff(funcion, x)
print(derivada)
```

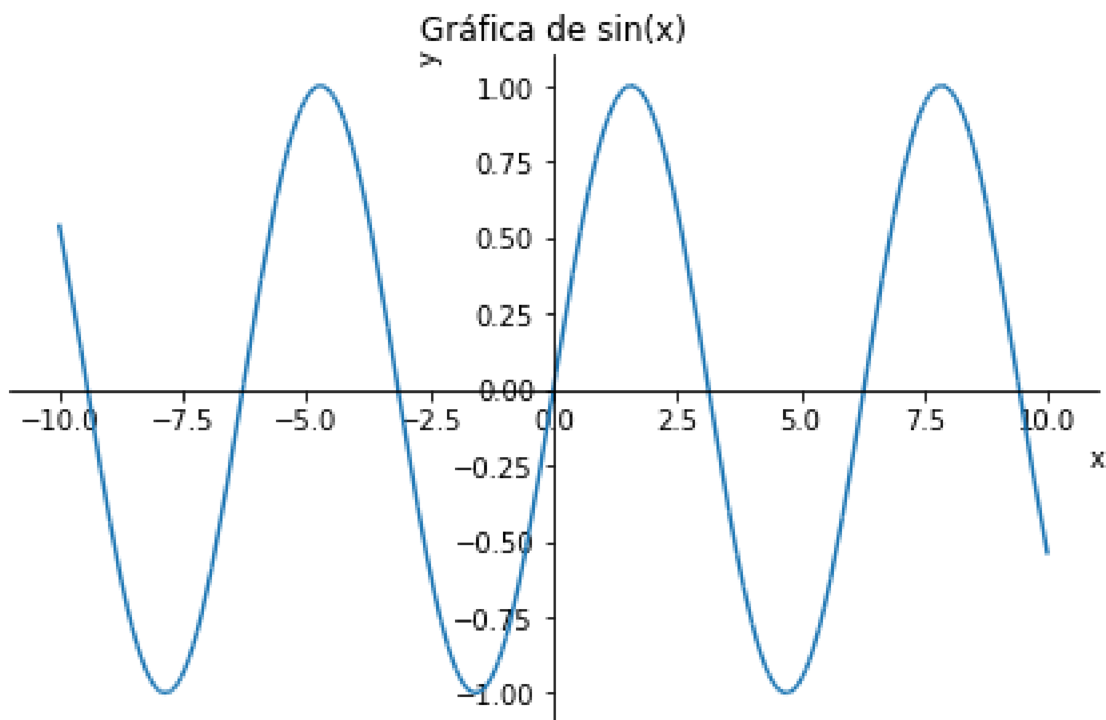
```
# Integrando una función
x = sympy.symbols('x')
funcion = sympy.exp(x)
integral = sympy.integrate(funcion, x)
print(integral)

# Calculando el valor numérico de una integral
x = sympy.symbols('x')
funcion = sympy.sin(x)
integral = sympy.integrate(funcion, (x, 0, sympy.pi/2))
valor_numerico = integral.evalf()
print(valor_numerico)

# Resolviendo una ecuación algebraica
x = sympy.symbols('x')
ecuacion = x**2 - 1
solucion = sympy.solve(ecuacion, x)
print(solucion)

# Simplificando una expresión matemática
x, y = sympy.symbols('x y')
expresion = sympy.sqrt(x**2 + y**2)
simplificado = sympy.simplify(expresion)
print(simplificado)

# Verificando si una expresión es identica a otra
x, y = sympy.symbols('x y')
expresion1 = x + y
expresion2 = y + x
identicas = sympy.identical(expresion1, expresion2)
print(identicas)
```

2.8.2. Generacion de graficas en Sympy

Aquí hay un ejemplo de código en Python que muestra cómo generar una gráfica de una función utilizando la biblioteca SymPy y su módulo sympy.plotting:

```
import sympy
import sympy.plotting as syp

# Generando una gráfica de una función
x = sympy.symbols('x')
funcion = sympy.sin(x)
grafica = syp.plot(funcion, show=False)
grafica.title = 'Gráfica de sin(x)'
grafica.xlabel = 'x'
grafica.ylabel = 'y'
grafica.show()
```

Figura 2.5: Grafica de sympy



2.8.3. Resolución de una ecuacion diferencial ordinaria de primer orden con sympy

Aquí hay tres ejemplos de código en Python que utilizan la biblioteca SymPy para resolver ecuaciones diferenciales de primer orden:

1. Ecuación diferencial $y' + y = 0$ con condición inicial $y(0) = 1$:

```
import sympy

t = sympy.symbols('t')
y = sympy.Function('y')(t)

# Definir la ecuación diferencial
eq = y.diff(t) + y

# Resolviendo la ecuación diferencial
sol = sympy.dsolve(eq)

# Aplicando la condición inicial
sol = sol.subs(y, 1).subs(t, 0)
```

2. Ecuación diferencial $y' + 2y = t$ con condición inicial $y(0) = 1$:

```
import sympy

t = sympy.symbols('t')
y = sympy.Function('y')(t)

# Definir la ecuación diferencial
```

```
eq = y.diff(t) + 2*y - t

# Resolviendo la ecuación diferencial
sol = sympy.dsolve(eq)

# Aplicando la condición inicial
sol = sol.subs(y, 1).subs(t, 0)
```

3. Ecuación diferencial $y' = t^2 - y^2$ con condición inicial $y(0) = 1$:

```
import sympy

t = sympy.symbols('t')
y = sympy.Function('y')(t)

# Definir la ecuación diferencial
eq = y.diff(t) - (t**2 - y**2)

# Resolviendo la ecuación diferencial
sol = sympy.dsolve(eq)

# Aplicando la condición inicial
sol = sol.subs(y, 1).subs(t, 0)
```

2.8.4. Resolución de una ecuacion diferencial ordinaria de segundo orden con sympy

Aquí hay tres ejemplos de código en Python que utilizan la biblioteca SymPy para resolver ecuaciones diferenciales de segundo orden:

1. Ecuación diferencial $y'' + y = 0$ con condiciones iniciales $y(0) = 1$ y $y'(0) = 0$:

```
import sympy

t = sympy.symbols('t')
y = sympy.Function('y')(t)

# Definir la ecuación diferencial
eq = y.diff(t, t) + y

# Resolviendo la ecuación diferencial
sol = sympy.dsolve(eq)

# Aplicando las condiciones iniciales
sol = sol.subs(y, 1).subs(y.diff(t), 0).subs(t, 0)
```

2. Ecuación diferencial $y'' + 4y = t^2$ con condiciones iniciales $y(0) = 1$ y $y'(0) = 0$:

```
import sympy

t = sympy.symbols('t')
y = sympy.Function('y')(t)

# Definir la ecuación diferencial
eq = y.diff(t, t) + 4*y - t**2
```

```
# Resolviendo la ecuación diferencial
sol = sympy.dsolve(eq)

# Aplicando las condiciones iniciales
sol = sol.subs(y, 1).subs(y.diff(t), 0).subs(t, 0)
```

3. Ecuación diferencial $y'' - 2y' + y = t^2$ con condiciones iniciales $y(0) = 1$ y $y'(0) = 0$:

```
import sympy

t = sympy.symbols('t')
y = sympy.Function('y')(t)

# Definir la ecuación diferencial
eq = y.diff(t, t) - 2*y.diff(t) + y - t**2

# Resolviendo la ecuación diferencial
sol = sympy.dsolve(eq)

# Aplicando las condiciones iniciales
sol = sol.subs(y, 1).subs(y.diff(t), 0).subs(t, 0)
```

2.8.5. Resolución de un sistema de ecuaciones diferenciales ordinarias con sympy

Aquí hay dos ejemplos de código en Python usando SymPy para resolver sistemas de ecuaciones diferenciales:

1. Sistema de dos ecuaciones lineales:

```
from sympy import symbols, Function, dsolve, Eq
from sympy.abc import x, y

f, g = symbols('f g', cls=Function)
f = Function('f')(x)
g = Function('g')(x)
ode1 = Eq(f.diff(x) - 2*g, 0)
ode2 = Eq(g.diff(x) + 3*f, 0)
sol = dsolve((ode1, ode2))
sol
```

2. Sistema de dos ecuaciones no lineales:

```
from sympy import symbols, Function, dsolve, Eq
from sympy.abc import x, y

f, g = symbols('f g', cls=Function)
f = Function('f')(x)
g = Function('g')(x)
ode1 = Eq(f.diff(x) - f*g, 0)
ode2 = Eq(g.diff(x) + f**2 - g**2, 0)
sol = dsolve((ode1, ode2))
sol
```

2.9. Scipy

¿ QUE ES SCIPY EN PYTHON?

SciPy es una biblioteca de software libre para Python que se utiliza para resolver problemas de ciencia de datos y cálculo científico. Es una colección de módulos y herramientas construidas sobre NumPy, una biblioteca de Python para el manejo de arrays y matrices numéricas.

SciPy incluye una amplia variedad de funciones y algoritmos, como:

- Procesamiento de señales y imágenes
- Optimización y ajuste de curvas
- Integración y solución de ecuaciones diferenciales
- Interpolación y suavizado de datos
- Estadística y análisis de datos
- Análisis de componentes principales (PCA) y clustering

SciPy es una herramienta poderosa y ampliamente utilizada por científicos, ingenieros y analistas de datos para resolver problemas complejos y para realizar análisis y visualizaciones de datos.

2.9.1. EJEMPLOS CON SCIPY

1. Integración numérica:

```
import scipy.integrate as spi

def f(x):
    return x**2

# Integrar la función f desde 0 hasta 2
result, error = spi.quad(f, 0, 2)
print("Resultado de la integración:", result)
```

2. Solución de un sistema de ecuaciones lineales:

```
import numpy as np
from scipy.linalg import solve

A = np.array([[3, 1], [1, 2]])
b = np.array([9, 8])

# Solucionar el sistema Ax = b
x = solve(A, b)
print("Solución del sistema:", x)
```

3. Interpolación de datos:

```
import numpy as np
import scipy.interpolate as spi

x = np.array([0, 1, 2, 3, 4, 5])
y = np.array([1, 2, 1, 2, 1, 2])

# Interpolar los datos usando una spline cubica
f = spi.CubicSpline(x, y)

# Evaluar la spline en un punto
print("Valor de la spline en x = 2.5:", f(2.5))
```

Estos son solo algunos ejemplos de las muchas funciones y algoritmos disponibles en SciPy. La biblioteca es muy amplia y puede ser utilizada para resolver una gran variedad de problemas científicos y matemáticos.

2.10. Modulo Odeint

¿ QUE ES EL ODEINT EN PYTHON

odeint es una función integrada en el módulo scipy.integrate de Python que se utiliza para resolver sistemas de ecuaciones diferenciales ordinarias (ODEs). Esta función permite integrar un sistema de ODEs de forma numérica, utilizando diversos métodos de integración, como el método de Runge-Kutta, y proporciona una solución aproximada a las ODEs. Es una herramienta valiosa en la resolución de problemas en campos como la física, la biología, la ingeniería y las matemáticas.

1. EJEMPLO Ecuación diferencial de segundo orden de un resorte La ecuación diferencial es:

$$m\frac{d^2}{dt^2}y + k\frac{dt}{dy} + ky = 0$$

donde m es la masa del objeto, k es la constante elástica y y es la posición del objeto. *Código en Python con odeint:*

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

def resorte(y, t, m, k):
    dy = y[1]
    ddy = -k/m * y[0]
    return [dy, ddy]

t = np.linspace(0, 10, 100)
y0 = [0, 1]
m = 1
k = 1

sol = odeint(resorte, y0, t, args=(m, k))

plt.plot(t, sol[:, 0], label='Posición')
plt.xlabel('Tiempo (s)')
plt.ylabel('Posición (m)')
plt.title('Gráfico de la posición del resorte')
plt.legend()
plt.show()
```

la grafica del ejemplo anterior es:

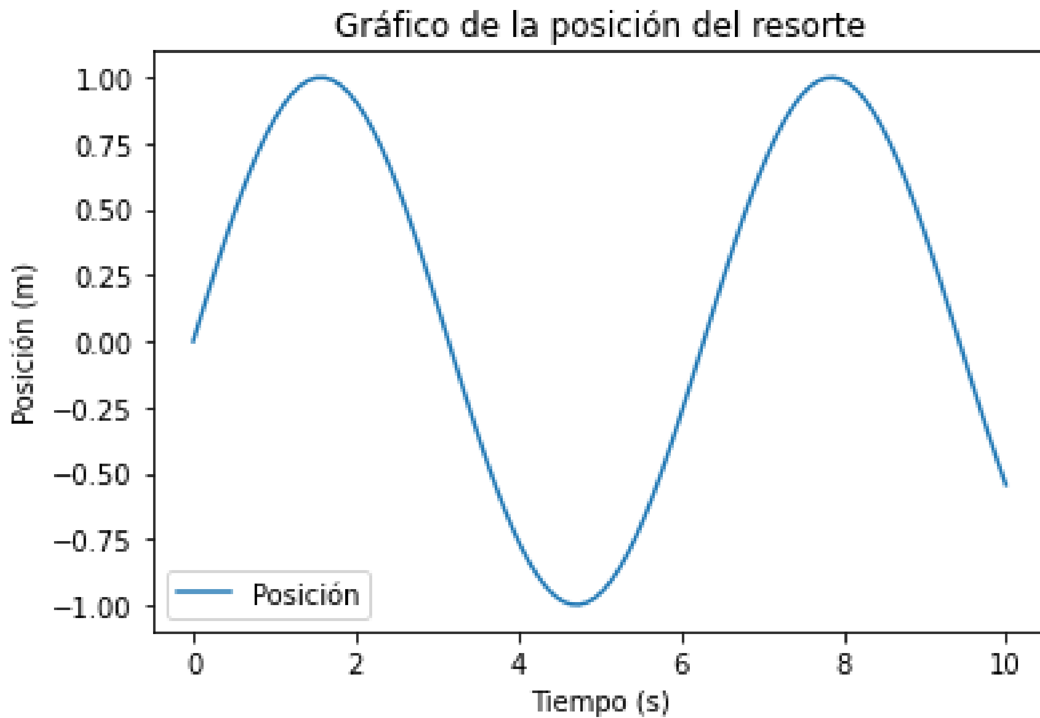
2. EJEMPLO Ecuación diferencial de orden superior:

$$\frac{d^3}{dt^3}y + \frac{d^2}{dt^2}y + y = 0$$

Código en Python con odeint:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
```

Figura 2.6: Gráfica de Ecuación diferencial de segundo orden de un resorte




```
def ecuacion(y, x):
                return [y[1], y[2], -2*y[1]-y[0]]

x = np.linspace(0, 10, 100)
y0 = [1, 0, 0]
y = odeint(ecuacion, y0, x)

plt.plot(x, y[:,0])
plt.xlabel("x")
plt.ylabel("y")
plt.title("Ecuación diferencial de orden superior")
plt.show()
```

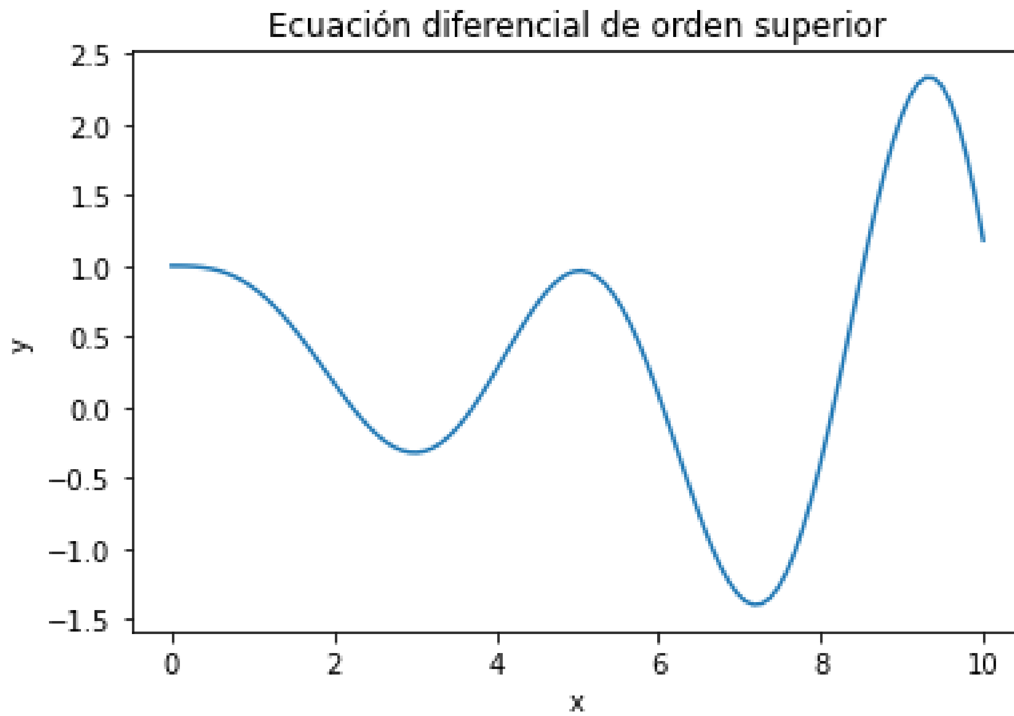
Gráfica del ejemplo:

3. Ejemplo

   Ecuación de Lorentz

```
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import numpy as np

def lorenz(y, t, sigma, beta, rho):
    x, y, z = y
    dxdt = sigma * (y - x)
    dydt = x * (rho - z) - y
    dzdt = x * y - beta * z
    return dxdt, dydt, dzdt
```

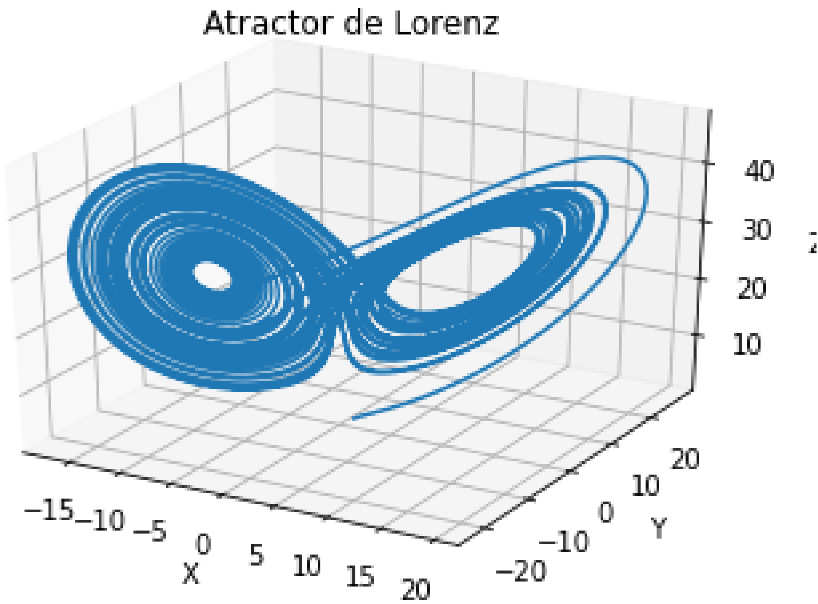
Figura 2.7: Ecuación diferencial de orden superior




```
sigma = 10
beta = 8 / 3
rho = 28
y0 = [1, 1, 1]
t = np.linspace(0, 50, 10000)
sol = odeint(lorenz, y0, t, args=(sigma, beta, rho))

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot(sol[:,0], sol[:,1], sol[:,2])
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
ax.set_title("Atractor de Lorenz")
plt.show()
```

Gráfica del ejemplo anterior usando matplotlib:

Figura 2.8: Atractor de Lorenz



Capítulo 3

autovalores, autovectores, diagonalizacion de matrices

3.1. Autovalores

Autovalores se pueden calcular resolviendo la ecuación característica det($\mathbf{A} - \lambda\mathbf{I}) = 0$, donde **A** es la matriz cuya autovalores se quieren calcular y **I** es la matriz identidad. La solución de esta ecuación son los autovalores de la matriz **A**.

Aquí hay un ejemplo de código en Sympy para calcular autovalores de una matriz:

```
from sympy import Matrix

A = Matrix([[3, 2], [1, 3]])
autovalores = A.eigenvals()
print(autovalores)
```

Aquí hay un ejemplo de código en Sympy para calcular autovalores de una matriz 3x3:

```
from sympy import Matrix

A = Matrix([[3, 2, 0], [1, 3, 4], [5, 6, 7]])
autovalores = A.eigenvals()
print(autovalores)
```

Este código imprimirá los autovalores de la matriz A, que son 10, 3 y -1.

```
from sympy import Matrix

A = Matrix([[2, -1, 0], [-1, 2, -1], [0, -1, 2]])
autovalores = A.eigenvals()
print(autovalores)
```

Este código imprimirá los autovalores de la matriz A, que son 3, 2 y 1.

3.2. Autovectores

Aquí hay un ejemplo de código en Sympy para calcular autovectores de una matriz 3x3:

```
from sympy import Matrix

A = Matrix([[3, 2, 0], [1, 3, 4], [5, 6, 7]])
autovectores = A.eigenvects()
print(autovectores)
```

Este código imprimirá los autovectores y autovalores de la matriz A junto con sus respectivas multiplicidades.

Aquí hay otro ejemplo de código en Sympy para calcular autovectores de una matriz 3x3:

```
from sympy import Matrix

A = Matrix([[2, -1, 0], [-1, 2, -1], [0, -1, 2]])
autovectores = A.eigenvects()
print(autovectores)
```

Este código imprimirá los autovectores y autovalores de la matriz A junto con sus respectivas multiplicidades.

3.3. Diagonalizacion de Matrices

Aquí está un programa en Python usando Sympy para calcular autovalores, autovectores y generar la matriz diagonal a partir de una matriz de orden 2:

```
from sympy import Matrix, diag

# Matriz de orden 2
A = Matrix([[3, 2], [1, 3]])

# Calcular autovalores y autovectores
autovalores, autovectores = A.eigenvects()[0][0], A.eigenvects()[0][2][0].tolist()

# Crear matriz diagonal con autovalores
D = diag(*autovalores)

print("Autovalores:", autovalores)
print("Autovectores:", autovectores)
print("Matriz diagonal:")
print(D)
```

Este programa imprimirá los autovalores de la matriz, los autovectores y la matriz diagonal correspondiente.

Aquí está un programa en Python usando Sympy para calcular autovalores, autovectores y generar la matriz diagonal a partir de una matriz de orden 3:

```
from sympy import Matrix, diag

# Matriz de orden 3
A = Matrix([[2, -1, 0], [-1, 2, -1], [0, -1, 2]])

# Calcular autovalores y autovectores
autovalores = []
autovectores = []
for i in range(len(A.eigenvects())):
    autovalores.append(A.eigenvects()[i][0])
    autovectores.append(A.eigenvects()[i][2][0].tolist())

# Crear matriz diagonal con autovalores
D = diag(*autovalores)

print("Autovalores:", autovalores)
print("Autovectores:", autovectores)
print("Matriz diagonal:")
```

```
print(D)
Este programa imprimirá los autovalores de la matriz, los autovectores y la matriz diagonal correspo
```

Este programa imprimirá los autovalores de la matriz, los autovectores y la matriz diagonal correspondiente.

Capítulo 4

Matrices exponenciales

xisten varias bibliotecas en Python que puedes usar para calcular matrices exponenciales:

- NumPy: Es una biblioteca de cálculo numérico de alto rendimiento para Python. Proporciona funciones para calcular la matriz exponencial a través de la función numpy.linalg.expm.
- SciPy: Es una biblioteca de Python que brinda soporte para cálculo científico. Proporciona una función para calcular la matriz exponencial a través de la función scipy.linalg.expm.
- SymPy: Es una biblioteca de Python para matemáticas simbólicas. Proporciona funciones para calcular la matriz exponencial a través de la función sympy.matrices.expressions.MatrixExpr.exp.

En general, es recomendable utilizar NumPy o SciPy para aplicaciones de cálculo numérico debido a su alta eficiencia en comparación con otras bibliotecas. Sin embargo, si necesitas realizar cálculos simbólicos, entonces es recomendable usar SymPy.

Aquí hay ejemplos de código para calcular la matriz exponencial usando NumPy, SciPy y SymPy:

1. Numpy

```
import numpy as np

# Matriz de orden 2
A = np.array([[0, 1], [-1, 0]])

# Calcular la matriz exponencial
expm = np.linalg.expm(A)

print("Matriz exponencial:")
print(expm)
```

2. Scipy

```
import scipy.linalg as sla

# Matriz de orden 2
A = np.array([[0, 1], [-1, 0]])

# Calcular la matriz exponencial
expm = sla.expm(A)

print("Matriz exponencial:")
print(expm)
```

3. Sympy

```
import sympy

# Matriz de orden 2
A = sympy.Matrix([[0, 1], [-1, 0]])

# Calcular la matriz exponencial
expm = A.exp()

print("Matriz exponencial:")
print(expm)
```

Estos programas imprimirán la matriz exponencial de la matriz de orden 2.

Capítulo 5

Formas canonicas de Jordan

Hay varias bibliotecas en Python que puedes usar para encontrar formas canónicas de matrices:

- NumPy: Es una biblioteca de cálculo numérico de alto rendimiento para Python. No proporciona una función específica para encontrar formas canónicas, pero puedes usar sus funciones de álgebra lineal para calcular los autovalores y autovectores de una matriz y utilizarlos para encontrar formas canónicas.
- SciPy: Es una biblioteca de Python que brinda soporte para cálculo científico. Al igual que NumPy, no proporciona una función específica para encontrar formas canónicas, pero puedes usar sus funciones de álgebra lineal para calcular los autovalores y autovectores de una matriz y utilizarlos para encontrar formas canónicas.
- SymPy: Es una biblioteca de Python para matemáticas simbólicas. Proporciona funciones para calcular los autovalores y autovectores de una matriz y utilizarlos para encontrar formas canónicas.

En general, si necesitas realizar cálculos numéricos, es recomendable utilizar NumPy o SciPy debido a su alta eficiencia en comparación con otras bibliotecas. Sin embargo, si necesitas realizar cálculos simbólicos, entonces es recomendable usar SymPy.

Aquí hay ejemplos de código para calcular la forma canónica de Jordan de una matriz usando NumPy, SciPy y SymPy

1. Numpy

```
import numpy as np

# Matriz de orden 3
A = np.array([[2, -1, 0], [1, 2, -1], [0, 1, 2]])

# Calcular autovalores y autovectores
eigenvalues, eigenvectors = np.linalg.eig(A)

# Ordenar los autovalores y autovectores en función de los autovalores
idx = eigenvalues.argsort()[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:,idx]

# Formar la matriz diagonal con los autovalores
D = np.diag(eigenvalues)

# Formar la forma canónica de Jordan con los autovectores
J = np.dot(eigenvectors, np.dot(D, np.linalg.inv(eigenvectors)))

print("Forma canónica de Jordan:")
print(J)
```

2. Scipy

```
import scipy.linalg as sla

# Matriz de orden 3
A = np.array([[2, -1, 0], [1, 2, -1], [0, 1, 2]])

# Calcular autovalores y autovectores
eigenvalues, eigenvectors = sla.eig(A)

# Ordenar los autovalores y autovectores en función de los autovalores
idx = eigenvalues.argsort()[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:,idx]

# Formar la matriz diagonal con los autovalores
D = np.diag(eigenvalues)

# Formar la forma canónica de Jordan con los autovectores
J = np.dot(eigenvectors, np.dot(D, np.linalg.inv(eigenvectors)))

print("Forma canónica de Jordan:")
print(J)
```

3. Sympy

```
import sympy

# Matriz de orden 3
A = sympy.Matrix([[2, -1, 0], [1, 2, -1], [0, 1, 2]])

# Calcular autovalores y autovectores
eigenvalues, eigenvectors = A.eigenvects()

# Formar la matriz diagonal con los autovalores
D = sympy.diag(*[x[0] for x in eigenvalues])

# Formar la forma canónica de Jordan con los autovectores
J = eigenvectors[0][2][0].transpose() * D * eigenvectors[0][2][0]

print("Forma canónica de Jordan:")
print(J)
```

Estos programas imprimirán la forma canónica de Jordan de las matrices de orden 3x3

Capítulo 6

Retratos de fase

Matemáticamente, un retrato de fase es un gráfico que representa el comportamiento de un sistema dinámico en el espacio de estado. Este gráfico muestra cómo las variables de estado del sistema cambian con el tiempo, y se utiliza para visualizar la dinámica del sistema.

Un retrato de fase se construye graficando el valor de una o más variables de estado del sistema contra el tiempo. Cada punto en el retrato de fase representa un estado particular del sistema en un momento particular. La trayectoria de un punto en el retrato de fase representa la evolución temporal del sistema a partir de ese estado inicial.

Los retratos de fase se utilizan ampliamente en la teoría de sistemas dinámicos para estudiar la estabilidad, los atractores y las fuentes, y para identificar patrones repetitivos o periódicos en el comportamiento del sistema. Además, los retratos de fase pueden ser útiles para predecir el comportamiento a largo plazo de un sistema a partir de un estado inicial dado.

6.1. Librerias para hallar retratos de fase con python

Con Numpy:

1. Integre las ecuaciones de un sistema dinámico usando la función odeint de Numpy para calcular las trayectorias en el espacio de estado.
2. Use Matplotlib para graficar las trayectorias en el espacio de estado y crear el retrato de fase.

Con Scipy:

1. Integre las ecuaciones de un sistema dinámico usando la función odeint de Scipy para calcular las trayectorias en el espacio de estado.
2. Use Matplotlib para graficar las trayectorias en el espacio de estado y crear el retrato de fase.

Con Sympy:

1. Calcule las trayectorias en el espacio de estado resolviendo las ecuaciones de un sistema dinámico de forma simbólica.
2. Use Matplotlib para graficar las trayectorias en el espacio de estado y crear el retrato de fase.

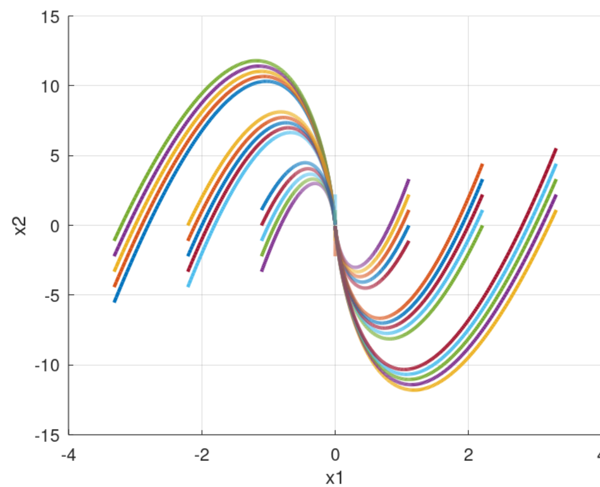
6.1.1. ejemplo:

retratos de fase para una variación: de a y b del sistema:

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

t = np.arange(-99, 1, 0.1)
```

```
a = 1/10
b = 1

for i in range(-3, 4):
    for j in range(-2, 3):
        c1 = i
        c2 = j
        plt.plot(np.exp(a*t)*c1, np.exp(a*t)*(c1*b*t + c2), linewidth=2)

plt.xlabel('x1')
plt.ylabel('x2')
plt.zlabel('t')
plt.grid(True)
plt.show()
```

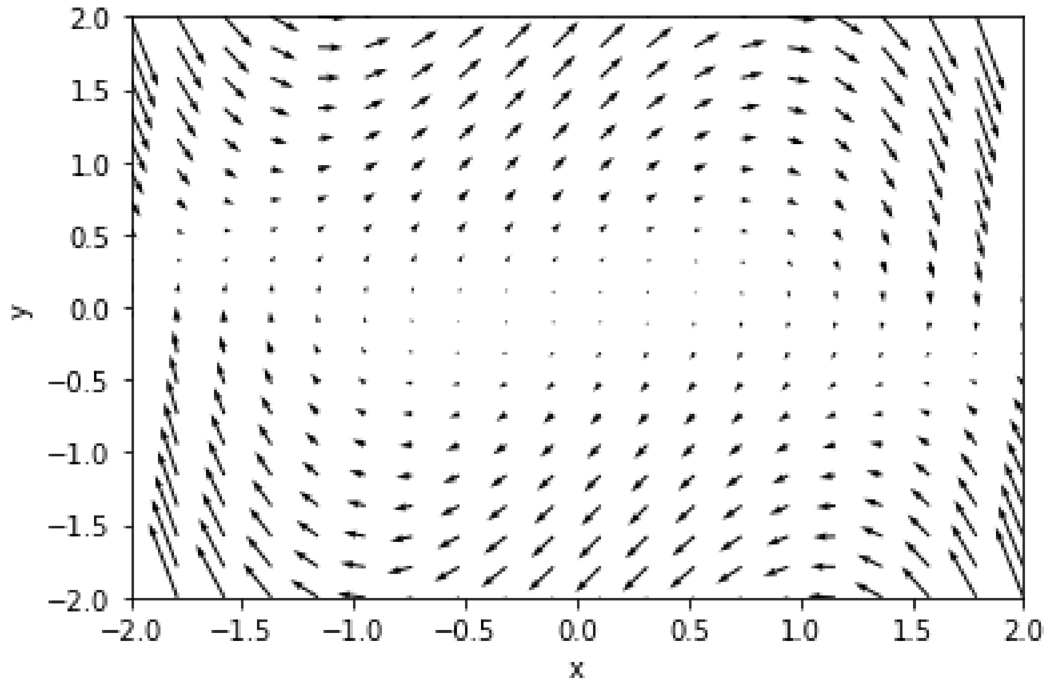
Figura 6.1: Solucion con variaciones de a y b




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

# definir el sistema dinámico
def dx_dt(x, y):
    return y, -0.5*x + y*(1 - x**2)

# crear una cuadrícula de puntos en el plano
x, y = np.meshgrid(np.linspace(-2, 2, 20), np.linspace(-2, 2, 20))

# calcular las velocidades en cada punto de la cuadrícula
dx, dy = dx_dt(x, y)

# graficar los vectores de velocidad
plt.quiver(x, y, dx, dy)

# establecer límites y etiquetas de los ejes
plt.xlim([-2, 2])
plt.ylim([-2, 2])
```

```
plt.xlabel('x')
plt.ylabel('y')

# mostrar el gráfico
plt.show()
```

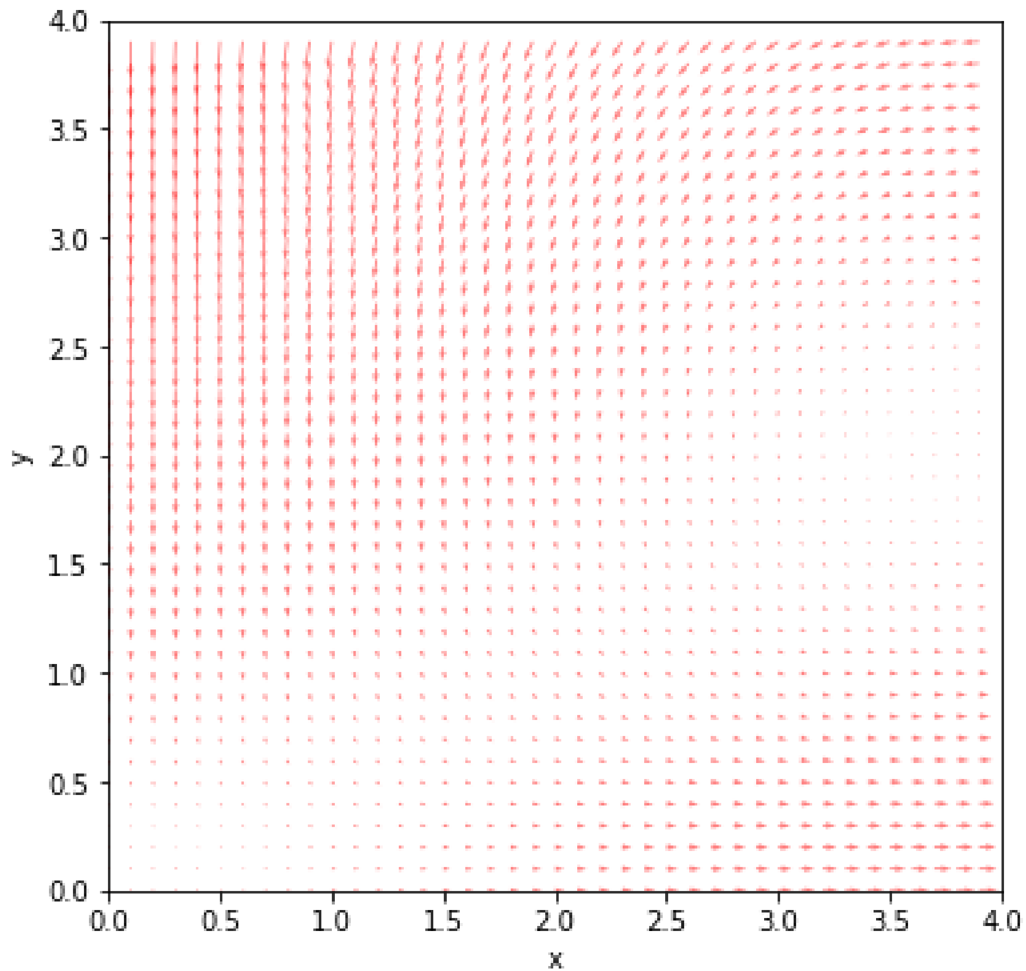
Figura 6.2: grafica de un sistema




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

# Define el sistema dinámico de Lotka-Volterra
def lotka_volterra(X, t, a, b, c, d):
    x, y = X
    dx_dt = a*x - b*x*y
    dy_dt = c*x*y - d*y
    return [dx_dt, dy_dt]

# Define los parámetros del sistema dinámico
a = 1
b = 0.5
c = 0.5
d = 2

# Define el rango de valores de x y y
x_range = np.arange(0, 4, 0.1)
y_range = np.arange(0, 4, 0.1)

# Crea la malla de puntos (x, y)
X, Y = np.meshgrid(x_range, y_range)

# Calcula las derivadas dx/dt y dy/dt en cada punto de la malla
```

```
DX, DY = lotka_volterra([X, Y], 0, a, b, c, d)

# Crea el retrato de fase
fig, ax = plt.subplots(figsize=(6, 6))
ax.quiver(X, Y, DX, DY, color='r', alpha=0.5)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_xlim([0, 4])
ax.set_ylim([0, 4])
plt.show()
```

Figura 6.3: grafico de un sistema EDO



Capítulo 7

Conclusion

Python es un lenguaje de programación muy popular y versátil que se utiliza en una amplia variedad de campos, incluyendo matemáticas y sistemas dinámicos. En matemáticas, Python es ampliamente utilizado para realizar cálculos numéricos y simbólicos, así como para graficar y visualizar datos.

Una de las bibliotecas más utilizadas para matemáticas en Python es NumPy, que proporciona una amplia variedad de funciones y herramientas para realizar operaciones numéricas y matriciales. También existe la biblioteca de cálculo simbólico SymPy, que permite realizar cálculos simbólicos y algebraicos, lo que es útil para resolver ecuaciones y realizar análisis matemáticos avanzados.

En cuanto al uso de sistemas dinámicos, Python es una herramienta muy útil y popular para modelar y simular sistemas dinámicos complejos. Existen varias bibliotecas para Python que se utilizan en el análisis y modelado de sistemas dinámicos, como SciPy y control, que proporcionan herramientas para la simulación y el control de sistemas dinámicos lineales y no lineales. Además, existen bibliotecas especializadas en la simulación de sistemas dinámicos complejos, como el paquete de modelado de sistemas complejos NetworKit.

En conclusión, Python es una herramienta muy útil y versátil para el análisis matemático y la simulación de sistemas dinámicos. Las bibliotecas de Python, como NumPy, SymPy, SciPy y control, proporcionan una amplia variedad de funciones y herramientas para realizar cálculos numéricos y simbólicos, así como para modelar y simular sistemas dinámicos complejos. Python es una herramienta valiosa para los matemáticos y científicos en muchos campos, desde la física y la ingeniería hasta la biología y la economía.

Capítulo 8

Referencias bibliográficas