FACULTAD LATINOAMERICANA DE CIENCIAS SOCIALES (FLACSO)

NOMBRE INTEGRANTES:

- Jhon Diego Cachimuel Aguilar: jcachimuelagufl@flacso.edu.ec
- Melanie Karina Cifuentes Suarez: mkcifuentesfl@flacso.edu.ec

Enlace a perfiles Github:

- Jhon Diego Cachimuel Aguilar: https://github.com/Diego171020
- Melanie Karina Cifuentes Suarez: https://github.com/MelaCifuentes

OBJETIVO:

Desarrollar un programa en Python que analice y visualice sistemas de ecuaciones diferenciales 2x2, incluyendo el cálculo de valores propios y la generación de gráficas según diferentes casos.

Resolucion del programa

Importaciones de las bibliotecas necesarias

Para ejecutar la programación primero realizaremos las importaciones de las bibliotecas necesarias que nos permiten realizar las operaciones matemáticas, análisis y visualizaciones pertinentes en Python. Para ello es necesario especificar los codigos que se utilizaron:

- (numpy): Manipulación de matrices y vectores numéricos.
- (matplotlib): Visualización de datos en gráficos.
- (scipy.linalg.eig): Cálculo de valores propios y vectores propios de una matriz.
- (scipy.integrate.odeint): Resolución numérica de ecuaciones diferenciales ordinarias.

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

Diego cachimuel Aguilar
10:14 Hoy

En este apartado se especifica los comandos para que nuestro programa funcione correcta y eficasmente, de acuerdo a los parametros que estableceremos.

Función para ingresar la matriz del sistema de ecuaciones 2x2

Nuestro programa requiere de una codificacion en donde podamos ingresar de manera dinamica sistemas de ecuaciones en forma matricial, de 2x2, para este caso, el programa nos devolvera un cuadro donde podremos escribir los valores de la matriz seguidos de espacios por ejemplo (1 -2 -2 -1). De un a forma dinámica hememos utilizado las siguientes codificaciones def: se utiliza para definir las funciones. print: para proporcionar la directriz de la ecuación. input: para obtener los datos requeridos. return: para devolver a la matriz.

Diego cachimuel Aguilar
10:18 Hoy

Con este codigo podremos realizar nuestro ingreso de los valores de la matriz, de forma 2x2, en forma lineal como se da el ejemplo.

```
# Función para ingresar la matriz del sistema de ecuaciones diferenciales 2x2
#------------------------------------------------------------------------------------------------
def ingresar_matriz():
    print("Ingrese los 4 elementos de la matriz 2x2 del sistema de ecuaciones diferenciales:")
    elementos = input("Formato: a b c d (por ejemplo, 6 -4 1 2): ").split()
    matriz = np.array([[float(elementos[0]), float(elementos[1])],
                       [float(elementos[2]), float(elementos[3])]])
    return matriz
#------------------------------------------------------------------------------------------------
```

Función para escribir el sistema de ecuaciones diferenciales

Dado el caso del programa se añade la posibilidad de que el programa nos escriba el sistema de ecuaciones diferenciales en base a los valores de la matriz ingresada anteriormente de la forma:

- $dx/dt = a11x + a12y$
- $dx/dt = a21x + a22y$

Definimos la entrada del sistema de ecuaciones, para cada fila con la funcion def escribir_sistema(A). La función recibe una matriz (A) de 2x2, que contiene los coeficientes de las ecuaciones diferenciales. Variables: Extrae los elementos de la matriz (A)

Diego cachimuel Aguilar
10:24 Hoy

Se realiza este apartado en nocion de que el programa nos establesca el sistema de ecuaciones diferenciales a trabajarse.

```
# Función para escribir el sistema de ecuaciones diferenciales
#-------------------------------------------------------------------------------
def escribir_sistema(A):
    a11, a12 = A[0, 0], A[0, 1]
    a21, a22 = A[1, 0], A[1, 1]
    print("\nSistema de ecuaciones diferenciales:")
    print(f"dx/dt = {a11}*x + {a12}*y")
    print(f"dy/dt = {a21}*x + {a22}*y")
#-------------------------------------------------------------------------------
```

Diego cachimuel Aguilar
10:32 Hoy

El codigo define el proceso para el calculo de la matriz 2x2

Función para calcular los valores propios, el discriminantes y resolucion del sistema de ecuaciones diferenciales.

```
# Función para calcular los valores propios y el discriminante
#-------------------------------------------------------------------------------
# La función `calcular_valores_vectores_propios(A)` recibe una matriz cuadrada `A`.
def calcular_valores_vectores_propios(A):
    eigenvalues, eigenvectors = eig(A)
    discriminante = (A[0, 0] + A[1, 1])**2 - 4 * np.linalg.det(A)
    return eigenvalues, eigenvectors, discriminante
#-------------------------------------------------------------------------------
```

La siguiente linea de codigo tiene como proposito imprimir los vectores propios y sus correspondientes valores propios o lambdas.

Propósito: Esta función tiene como objetivo imprimir los vectores propios y sus correspondientes valores propios de una matriz. Los vectores propios son fundamentales en el análisis de sistemas lineales y en la diagonalización de matrices.

- eigenvalues : Es un array que contiene los valores propios (λ) de una matriz. Cada valor propio está asociado a un vector propio correspondiente.
- eigenvectors : Es una matriz donde cada columna representa un vector propio asociado a los valores propios, for para iterar a través de los índices de los valores propios.

Diego cachimuel Aguilar
19:19 Hoy

Son codigos para que el programa me pueda devolver las soluciones en vectores propios en base a las resoluciones

La funcion range(len(eigenvalues))para iterar a través de los índices de los valores propios.

Función para escribir los vectores propios

```
# Función para escribir los vectores propios
#-------------------------------------------------------------------------------

def escribir_vectores_propios(eigenvalues, eigenvectors):
    print("\nVectores propios:")
    for i in range(len(eigenvalues)):
        print(f"Vector propio {i+1}: [{eigenvectors[0, i]:.3f}, {eigenvectors[1, i]:.3f}] asociado

#-------------------------------------------------------------------------------

# Función para resolver el sistema de ecuaciones diferenciales
#-------------------------------------------------------------------------------
def resolver_sistema(A, t, condiciones_iniciales):
    def sistema(y, t):
        return np.dot(A, y)

    sol = odeint(sistema, condiciones_iniciales, t)
```

```
    return sol
#-------------------------------------------------------------------------------
```

RESULTADOS Y GRAFICAS

Dibujamos un diagrama de fase que es una representación geométrica del comportamiento de las soluciones en el espacio de fases. Se usa para analizar la dinámica de un sistema sin necesidad de resolverlo explícitamente.

La función graficar_soluciones tiene como objetivo representar gráficamente las soluciones de un sistema de ecuaciones diferenciales en dos contextos: en función del tiempo y en un diagrama de fase.

Un diagrama de fase es una representación gráfica que muestra cómo las variables del sistema (en este caso (x) e (y)) evolucionan entre sí a lo largo del tiempo. En este contexto, se grafica (y) en función de (x), lo que permite visualizar la dinámica del sistema.

Mela Cifuentes
22:54 Ayer

plt.quiver(...): Esta función se utiliza para dibujar vectores en el gráfico.

Mela Cifuentes
23:04 Ayer

Valores Propios: Indican cómo las soluciones del sistema escalan en ciertas direcciones.
Vectores Propios: Indican las direcciones en las que estas escalas ocurren.
Discriminante: Puede referirse a una cantidad que ayuda a determinar la naturaleza (real o compleja) de los valores propios.

Mela Cifuentes
23:16 Ayer

Las condiciones iniciales son cruciales porque determinan la solución particular del sistema. En sistemas de ecuaciones diferenciales lineales, diferentes condiciones iniciales pueden llevar a diferentes trayectorias en el espacio de soluciones.

Diego cachimuel Aguilar
19:19 Hoy

Se agrega en las graficas comandos de visualizacion, tales como los colores de los vectores en el diagrama de fase, colores de los ejes, titulos etc.

```
## Función para graficar las soluciones y el diagrama de fase

# Variables de Entrada:

#                 ( t ): Un vector que representa el tiempo, donde ( t = [t_0, t_1, \ldots, t_n] )
#                        con ( n ) puntos temporales. Este vector es fundamental para evaluar cómo
#                        cambian
#                        las soluciones a lo largo del tiempo.

# ( \text{soluciones} ): Una matriz donde cada columna representa una variable del
#                        sistema.

# ( \text{eigenvectors} ): Aunque no se utiliza directamente en la parte del código
#                          compartido, se refiere a los vectores propios asociados
#                          a la matriz del sistema. Estos pueden ser útiles para analizar
#                          la estabilidad y el comportamiento cualitativo del sistema.

#                 (caso ): Una etiqueta o descriptor que indica qué tipo de análisis o condiciones
#                          iniciales se están utilizando.

#-------------------------------------------------------------------------------
def graficar_soluciones(t, soluciones, caso, ax):
    # Graficar x(t) y y(t)
    ax.plot(t, soluciones[:, 0], label="x(t)", color="b")
    ax.plot(t, soluciones[:, 1], label="y(t)", color="r")
    ax.set_xlabel("Tiempo (t)")
    ax.set_ylabel("Valor")
    ax.set_title(f"Gráfica de Soluciones - {caso}")
    ax.legend()
    ax.grid(True)
#-------------------------------------------------------------------------------
    # Diagrama de fase (x vs y)

# - plt.subplot(1, 2, 2): Crea un subgráfico en una figura con 1 fila y 2 columnas,
#   seleccionando el segundo subgráfico.
# - plt.plot(soluciones[:, 0], soluciones[:, 1], ...): Grafica la trayectoria en
#   el espacio de fases usando las soluciones obtenidas. Aquí:

# - soluciones[:, 0] representa los valores de ( x(t) ).
# - soluciones[:, 1] representa los valores de ( y(t) ).
# - La línea verde (color="g") representa cómo cambian ( x ) e ( y ) a lo largo del tiempo.

#-----------------------------------------------------------------------------------

# Diagrama de fase (x vs y)
def graficar_diagrama_fase(A, eigenvectors, ax):
    x = np.linspace(-2, 2, 20)
    y = np.linspace(-2, 2, 20)
    X, Y = np.meshgrid(x, y)
    U = A[0, 0] * X + A[0, 1] * Y
    V = A[1, 0] * X + A[1, 1] * Y

    ax.streamplot(X, Y, U, V, color='gray', density=2, arrowsize=1)
```

```
# Colores para los vectores propios
    colores = ['b', 'r']  # Azul para el primero, rojo para el segundo

    for i in range(eigenvectors.shape[1]):
        vector = eigenvectors[:, i] / np.linalg.norm(eigenvectors[:, i]) * 1.5
        ax.quiver(0, 0, vector[0], vector[1], angles='xy', scale_units='xy', scale=1,
                  color=colores[i], width=0.01, label=f"Vector propio {i+1}")

    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_title("Diagrama de Fase con Vectores Propios")
    ax.axhline(0, color='black', linewidth=0.5)
    ax.axvline(0, color='black', linewidth=0.5)
    ax.grid(True, linestyle='--', alpha=0.6)
    ax.legend()



# Función principal

# def main Esta línea define la función principal del programa, donde se ejecutarán las operaciones p
# ingresar_matriz(),permite al usuario introducir la matriz de coeficientes ( A ) del sistema de ecua

# Función principal

# def main Esta línea define la función principal del programa, donde se ejecutarán las
# operaciones principales para analizar el sistema.

def main():


    A = ingresar_matriz() # 1. Ingresar la matriz de coeficientes del sistema
                           #En este apartado se ingrea la matriz

    escribir_sistema(A) # 2. Escribir el sistema de ecuaciones diferenciales
                          # Muestra el sistema de ecuaciones diferenciales asociado a la matriz ( A ).
                          # Esto ayuda a los usuarios a visualizar el problema que están analizando

    eigenvalues, eigenvectors, discriminante = calcular_valores_vectores_propios(A)
                         # 3. Calcular los valores propios y el discriminante
                        # calcula los valores propios (eigenvalues) y los vectores propios (eigenvectors)
                        # de la matriz ( A ). También se calcula un "discriminante",
                        # que puede ser utilizado para clasificar el tipo de soluciones del sistema.

    escribir_vectores_propios(eigenvalues, eigenvectors)  # 4. Imprimir los vectores propios
                                                         # Devuelve los vectores propios que calculamos


#  Clasificación de los casos según el discriminante

    caso = ""
    if np.all(np.isreal(eigenvalues)):  # Caso 1: Valores propios reales
        if np.isclose(eigenvalues[0], eigenvalues[1]):  # Caso 2: Jordan
            caso = "Caso Jordan"
        else:
            caso = "Valores propios reales"
    else:  # Caso 3: Valores propios complejos
        caso = "Valores propios complejos"

    print(f"\nValores propios: {eigenvalues}")
    print(f"Discriminante: {discriminante}")
    print(f"Clasificación: {caso}")

# Se determina el caso de la matriz con el que se esta trabajando, dependiendo
# del valor del discriminante (np.isreal): si todos los valores son reales, nos
# encontramos en el caso de valores propios reales.
# Otro caso es el de si nos presentamos en raices reales e iguales que corresponde
# al caso de Jordan, y por ultimo el caso 3, si hablamos dell caso de valores complejos.


# Resolver el sistema con condiciones iniciales arbitrarias (por ejemplo, [1, 0])
    t = np.linspace(0, 10, 100)  # Tiempo de 0 a 10
    condiciones_iniciales = [1, 0]  # Condiciones iniciales
    soluciones = resolver_sistema(A, t, condiciones_iniciales)

# Crear una figura con dos subgráficos
    fig, axs = plt.subplots(1, 2, figsize=(12, 6))
```

Diego cachimuel Aguilar 17:19 Hoy

El codigo define el caso de trabajo de la matriz segun el valor de la determinante:
Caso 1: Valores propios reales e iguales
Caso 2: Valores propios reales y distintas
Caso 3: Caso de los complejos

Diego cachimuel Aguilar 17:21 Hoy

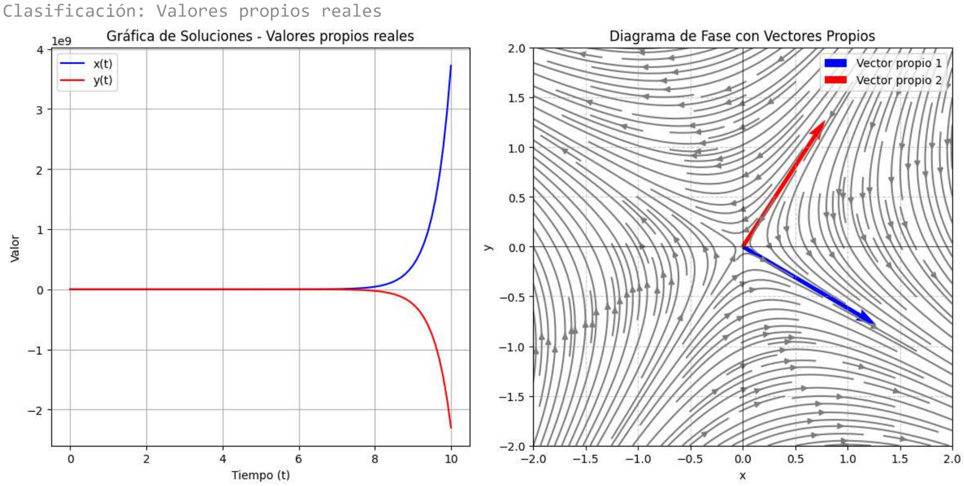
Dependiendo de la consideracion del caso, el programa decide que camino tomar para la resolucion de este sistema.

Diego cachimuel Aguilar 19:20 Hoy

Finalmente tenemos el apartado en donde se clasifica las soluciones y seleccion de caso. ademas de imprimir las solcuiones requeridas.

```
# Graficar en los subgráficos
    graficar_soluciones(t, soluciones, caso, axs[0])
    graficar_diagrama_fase(A, eigenvectors, axs[1])

# Mostrar ambas gráficas juntas
    plt.tight_layout()
    plt.show()


if __name__ == "__main__":
    main()
```

```
Ingrese los 4 elementos de la matriz 2x2 del sistema de ecuaciones diferenciales:
Formato: a b c d (por ejemplo, 6 -4 1 2): 1 -2 -2 -1

Sistema de ecuaciones diferenciales:
dx/dt = 1.0*x + -2.0*y
dy/dt = -2.0*x + -1.0*y

Vectores propios:
Vector propio 1: [0.851, -0.526] asociado a λ = 2.236+0.000j
Vector propio 2: [0.526, 0.851] asociado a λ = -2.236+0.000j

Valores propios: [ 2.23606798+0.j -2.23606798+0.j]
Discriminante: 20.000000000000004
Clasificación: Valores propios reales
```



Explicacion Matematica

El programa analiza, el caso de resolucion de sistemas de ecuaciones diferenciales mediante un proceso dinamico en el que se ingresa la matriz de dimensiones 2x2. Para el caso de ejemplo tenemos una matriz de la forma:

$$A = \begin{bmatrix} 1 & -2 \\ -2 & -1 \end{bmatrix}$$

El sistema de ecuaciones para el caso se escribe de la siguiente manera:

$$\dot{x} = x - 2y$$
$$\dot{y} = -2x - y$$

La forma en del sistema de ecuaciones diferenciales que tenemos es de la siguiente manera:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ -2 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Para el caso tenemos la nocion de que la expresion es de la forma: $\dot{X} = AX$

El calculo de los valores propios Primero realizamos el calculo de acuerdo a la ecuacion caracteristica:

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

Para el calculo de la determinante realizamos lo siguiente:

$$det = \begin{bmatrix} 1-\lambda & -2 \\ -2 & -1-\lambda \end{bmatrix}$$
$$(1-\lambda)(-1-\lambda) - (2)(-2) = (-1 - \lambda + \lambda + \lambda^2)$$
$$\lambda = + - \sqrt{5}$$

Tenemos que los valores propios o lambdas encontrados son

$$\lambda 1 = +\sqrt{5}$$
$$\lambda 2 = -\sqrt{5}$$

El programa nos envia las mismas repuestas de manera automatica de acuerdo a las matrices ingresadas del caso.

Para el calculo de los vectores propios tenemos la forma:

$$(A - \lambda I)v = 0$$
$$\begin{bmatrix} 1-\sqrt{5} & -2 \\ -2 & -1-\sqrt{5} \end{bmatrix} \begin{bmatrix} v1 \\ v2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Resolviendo y teniendo la solucion de que:

$$V1 = \begin{bmatrix} 0.85 \\ -0.52 \end{bmatrix}$$
$$V2 = \begin{bmatrix} 0.52 \\ 0.85 \end{bmatrix}$$

De manera general el programa primero tiene una codificacion por la cual podemos ingresar los valores de una matriz cualquiera de dimensiones 2x2, despues esta se encarga de analizar el caso que se esta presentado, si son valores propios reales e iguales, reales y distintas o en la situacion de los complejos. Ahora bien de acuerdo a la solucion matematica el programa se encarga de aplicar el debido proceso para la resolucion en base al discriminante.Para el ejemplo de demostracion se realizo el caso de los valores propios reales. Y finalmente el programa se encarga de realizar un grafico en funcion de t y un diagrama de fase de las soluciones.

Bibliografia

1. Encalada,K(2022). Interfaz de resolución de las ecuaciones diferenciales (Tesis de http://repositorio.ucsg.edu.ec/bitstream/3317/18436/1/T-UCSG-PRE-ING-CIC-20.pdf)
2. García Fronti, V. et.al. /Revista de Investigación en Modelos Matemáticos aplicados a la Gestión y la Economía Año 8 Volumen II (2021-II). 18-34
3. https://www.researchgate.net/publication/360109804_Solving_Differential_Equations_using_Python
4. Navrachana (2022). Solving Differential Equations using Python. Researchgate. DOI: 10.13140/RG.2.2.26067.04641
5. Raffo Lecca, Eduardo; Mejía Puente, Miguel Aplicaciones computacionales de las ecuaciones diferenciales estocásticas Industrial Data, vol. 9, núm. 1, 2006, pp. 64-75 Universidad Nacional Mayor de San Marcos Lima, Perú.
6. Aguayo, R., Lizarraga, C., & Quiñonez, Y. (2021). Evaluación del desempeño académico en entornos virtuales utilizando el modelo PNL. RISTI: Revista Ibérica de Sistemas e Tecnologías de Información, (41), 34–49. https://doi.org/10.17013/risti
7. Pa q aprendes (2022, 14 de junio). Ecuaciones diferenciales con Python. Librería Sympy [https://www.youtube.com/watch?v=pByxDUy214M]. YouTube
8. Básicos de ingeniería (2021, 24 de marzo). Como resolver Ecuaciones Diferenciales en Python [https://www.youtube.com/watch?v=ZVDwM2Db1As.] YouTube
9. Gallardo, J. M. (2018). Análisis Numérico de Ecuaciones Diferenciales: Teoría y Ejemplos con Python. España: Publicado por el autor.

10. Mirjalili, V., Raschka, S. (2020). Python machine learning: aprendizaje automático y aprendizaje profundo con Python, scikit-learn y TensorFlow. España: Marcombo.