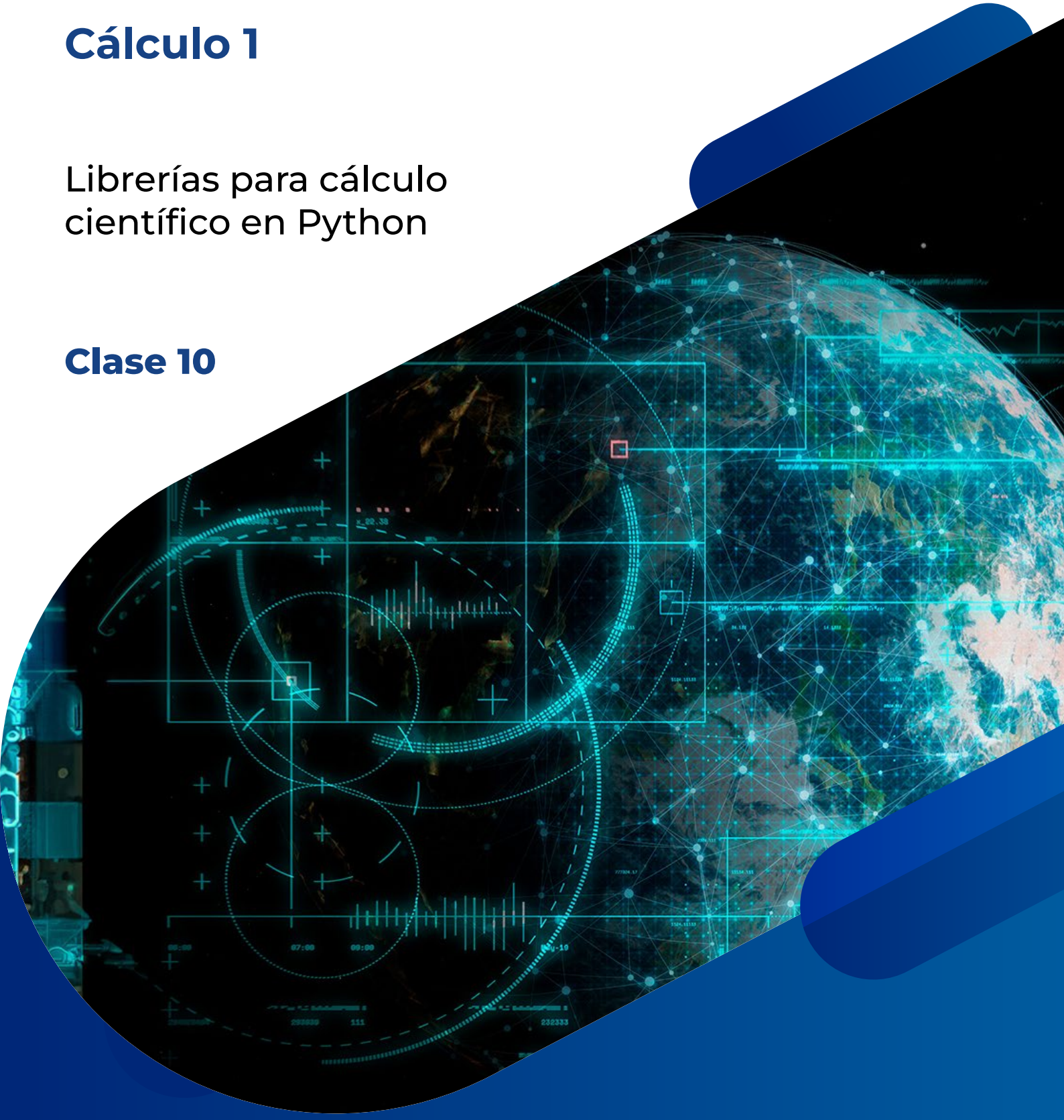
Cálculo 1

Librerías para cálculo científico en Python

Clase 10



Ingeniería en ciberseguridad

La excelencia no se improvisa

Clase 10: Librerías para cálculo científico en Python

Hoy vamos a aprender sobre **NumPy**, una librería esencial para la computación científica en Python. Veremos los conceptos básicos y cómo utilizarlos en nuestros proyectos.

10.1 Instalación y Configuración

Para usar NumPy, primero debemos importarla. Utilizaremos el siguiente comando:

Figura 17

Importación de librería NumPy

```python
import numpy as np
```

Nota. Código para importar librerías en el NumPy

Con esto, cada vez que queramos usar alguna función de NumPy, simplemente escribiremos *np* seguido de la función correspondiente.

10.2 Operaciones básicas

Creación de Arrays

Los arrays son estructuras de datos fundamentales en NumPy. Veamos cómo crear y manipular arrays.

Creación de un Array Unidimensional

Podemos crear un array unidimensional (vector) de la siguiente manera:

Figura 18

Código para estructura de datos arrays

```python
arreglo1 = np.array([1, 2, 3, 4, 5])
arreglo2 = np.array([10, 34, 27, 89])
```

Nota. Creación de un código de array unidimensional

Creación de una Matriz

Para crear una matriz (array bidimensional), es necesario definir las filas y las columnas:

Figura 19

Código para una matriz 2X3 en Python

```python
matriz = np.array([[1, 2, 3], [4, 5, 6]])
print(matriz)
```

Nota. Esto generará una matriz con dos filas y tres columnas.

Si queremos una matriz de 3x3, añadimos una fila más:

Figura 20

Código para una matriz 3X3 en Python

```python
matriz = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(matriz)
```

Nota. Esto generará una matriz con tres filas y tres columnas

10.3 Operaciones con Arrays

NumPy permite realizar operaciones matemáticas entre arrays de manera eficiente.

Suma de Arrays

Podemos sumar dos arrays de la siguiente manera:

Figura 21

Suma de elementos en Python

```python
arreglo1 = np.array([1, 2, 3, 4, 5])
arreglo2 = np.array([6, 7, 8, 9, 10])
suma = arreglo1 + arreglo2
print(suma)
```

Nota. Esto sumará elemento por elemento de los dos arrays, resultando en un nuevo array [7, 9, 11, 13, 15].

Slicing (Subconjuntos de Arrays)

Podemos extraer subconjuntos de un array utilizando slicing. Por ejemplo, para obtener los elementos del segundo al cuarto:

Figura 22

Impresión de resultados

```python
subarray = arreglo1[1:4]
print(subarray)
```

Nota. Esto imprimirá [2, 3, 4].

Funciones en NumPy

Las funciones nos permiten realizar operaciones repetitivas de manera más sencilla y eficiente.

Definición de una Función

Supongamos que queremos calcular el promedio de los elementos de un array. Podemos definir una función para hacerlo.

Figura 23

Código para promedio

```python
def calcular_promedio(arreglo):
    promedio = np.mean(arreglo)
    return promedio
```

Nota. Con esto de obtendrá el promedio de elementos en un array.

Uso de la Función

Podemos usar esta función para calcular el promedio de diferentes arrays:

Figura 24

Promedio de diferentes arrays

```python
promedio1 = calcular_promedio(arreglo1)
promedio2 = calcular_promedio(arreglo2)
print(f"El promedio de arreglo1 es: {promedio1}")
print(f"El promedio de arreglo2 es: {promedio2}")
```

Nota. Se muestra el promedio de cada uno de los promedios

En esta sección se han revisado los siguientes temas:

- Cómo importar y configurar NumPy.
- Crear arrays unidimensionales y matrices.
- Realizar operaciones básicas con arrays.
- Extraer subconjuntos de arrays.
- Definir y utilizar funciones en NumPy.

Practiquen estos conceptos para familiarizarse con NumPy y sus capacidades.

10.4 Listas, diccionarios, dataframes

Python es un lenguaje de programación ampliamente utilizado en el análisis de datos, el desarrollo web, la automatización y más. Entre las estructuras de datos fundamentales en Python se encuentran las listas, los diccionarios y los DataFrames, cada una con sus características y usos específicos.

10.4.1 Listas

Las listas en Python son estructuras de datos que permiten almacenar una colección ordenada y mutable de elementos. Los elementos de una lista pueden ser de cualquier tipo de dato, y las listas pueden contener elementos de diferentes tipos al mismo tiempo. Se definen usando corchetes [] y se pueden modificar después de su creación. Las características son las siguientes:

- **Indexación:** Los elementos de una lista se pueden acceder mediante índices, empezando desde 0.
- **Mutabilidad:** Se pueden cambiar, agregar o eliminar elementos de una lista.
- **Funciones y métodos:** Python proporciona una variedad de funciones y métodos para manipular listas, como append(), remove(), sort(), len(), etc.

Ejemplo:

Figura 25

Manipulación de listas

```python
mi_lista = [1, "dos", 3.0, [4, 5]]
mi_lista.append("seis")
```

Nota. Se muestra código de función append

10.4.2 Diccionarios

Los diccionarios son estructuras de datos en Python que almacenan pares de clave-valor. A diferencia de las listas, los diccionarios no tienen un orden específico y las claves deben ser únicas e inmutables (como cadenas, números o tuplas). Se definen usando llaves {}.

Las características principales son:

- **Acceso a valores:** Los valores se acceden usando las claves asociadas.
- **Mutabilidad:** Se pueden agregar, modificar o eliminar pares de clave-valor.
- **Funciones y métodos:** Incluyen métodos como get(), keys(), values(), items(), etc.

Ejemplo:

Figura 26

Código de estructura de datos "diccionario"

```python
mi_diccionario = {"nombre": "Juan", "edad": 30, "ciudad": "Madrid"}
mi_diccionario["edad"] = 31
```

Nota. Código con estructura de diccionario basado en datos personales

10.4.3 DataFrames

Los DataFrames son estructuras de datos bidimensionales proporcionadas por la biblioteca pandas en Python. Son similares a una tabla en una base de datos o una hoja de cálculo de Excel, con filas y columnas. Los DataFrames son ideales para el análisis de datos y son ampliamente utilizados en ciencia de datos. Las características principales son:

- **Indexación:** Los DataFrames tienen índices para filas y columnas, que pueden ser etiquetados.
- **Manipulación de datos:** Permiten la manipulación eficiente de datos, incluyendo filtrado, agrupamiento, agregación y más.
- **Integración con otros datos:** Pueden ser creados a partir de archivos CSV, bases de datos y otras fuentes de datos.

Ejemplo:

Figura 27

Estructura de pandas

```python
import pandas as pd

data = {
    "Nombre": ["Ana", "Luis", "María"],
    "Edad": [25, 30, 22],
    "Ciudad": ["Barcelona", "Madrid", "Valencia"]
}

df = pd.DataFrame(data)
print(df)
```

Nota. Ejemplo de pandas en Python

Estas estructuras de datos son esenciales para organizar y manipular datos en Python, facilitando tareas como análisis de datos, desarrollo de software y modelado matemático.

10.4.4 Pandas y Matplotlib

En esta sección, veremos cómo aplicamos Python a las funciones de costo, precio, demanda, ingresos, lucro y puntos de quiebre que ya hemos estudiado teóricamente. Usaremos las bibliotecas NumPy y Matplotlib para visualizar estas funciones y sus resultados.

Importación de Librerías

Primero, importamos las librerías necesarias:

Figura 28

Importación de NumPy y Matplotlib

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

Código para importar librerías NumPy y Matplotlib

NumPy nos ayudará con las operaciones numéricas y Matplotlib con la visualización de gráficos. Pueden encontrar el tutorial de Matplotlib adjunto en nuestra aula virtual.

Definición de funciones

Definimos las funciones que utilizaremos: costo, precio, demanda, ingreso y lucro. Recordemos que:

- **Costo:** Tiene una parte fija y una parte variable.
- **Precio y demanda:** Dependen de dos constantes, m y n, donde n se multiplica por x.
- **Ingreso:** Es *x* multiplicado por el precio de demanda.
- **Lucro:** Es la resta entre ingreso y costo.

En todas estas funciones, *x* representa el número de ítems vendidos.

Ejemplo:

Figura 29

Definición de Funciones

```python
def costo(x):
    costo_fijo = 1000
    costo_variable = 50 * x
    return costo_fijo + costo_variable

def precio_demanda(x):
    m = 200
    n = -0.5
    return m + n * x

def ingreso(x):
    return x * precio_demanda(x)

def lucro(x):
    return ingreso(x) - costo(x)
```

Nota. Definición de función costo en Python

Generación de datos

Utilizaremos np.linspace para generar un vector de números de ítems vendidos.

Ejemplo:

Figura 30

Generación de Vectores

```python
x = np.linspace(1, 500, 1000)  # Generamos 1000 divisiones entre 1 y 500
```

Nota. Esto nos permitirá crear un rango continuo de datos para nuestras funciones.

Cálculo y visualización

Definimos nuestras variables para almacenar los valores de costo, precio, ingreso y lucro:

Figura 31

Definición de variables para funciones

```python
costo_valores = costo(x)
precio_valores = precio_demanda(x)
ingreso_valores = ingreso(x)
lucro_valores = lucro(x)
```

Nota. Aquí se define las variables a trabajar en la operación

Determinación del lucro máximo

Calculamos el número de ítems que debemos vender para obtener el máximo lucro:

Figura 32

Determinación de función lucro máximo

```python
x_max_lucro = x[np.argmax(lucro_valores)]
lucro_maximo = np.max(lucro_valores)
print(f"Número de ítems para máximo lucro: {x_max_lucro}")
print(f"Lucro máximo: {lucro_maximo}")
```

Nota. Aquí se determina la función lucro máximo

Puntos de equilibrio

Usamos una función para resolver ecuaciones y encontrar los puntos de quiebre:

Figura 33

Función de Resolución de ecuaciones

```python
from scipy.optimize import fsolve

def punto_equilibrio(x):
    return ingreso(x) - costo(x)

puntos_quiebre = fsolve(punto_equilibrio, [100, 400])
print(f"Puntos de equilibrio: {puntos_quiebre}")
```

Nota. Código para resolver punto de equilibrio

Visualización de resultados

Finalmente, visualizamos los resultados usando Matplotlib:

Figura 34

Utilización de Matplotlib

```python
plt.plot(x, costo_valores, label='Costo')
plt.plot(x, ingreso_valores, label='Ingreso')
plt.plot(x, lucro_valores, label='Lucro')
plt.scatter(x_max_lucro, lucro_maximo, color='red', label='Máximo Lucro')
plt.axvline(puntos_quiebre[0], color='grey', linestyle='--'
, label='Puntos de Quiebre')
plt.axvline(puntos_quiebre[1], color='grey', linestyle='--')
plt.legend()
plt.xlabel('Ítems Vendidos')
plt.ylabel('Valor ($)')
plt.title('Análisis Económico')
plt.show()
```

Nota. Ejemplo de visualización de resultados con Matplotlib

Interpretación de los resultados

- **Costo (azul):** Es una función lineal.
- **Ingreso (naranja):** Es una función cuadrática.
- **Lucro (verde):** Es la diferencia entre ingreso y costo.

- **Máximo lucro (punto rojo):** Indica el punto donde se obtiene el mayor beneficio.
- **Puntos de quiebre (líneas grises):** Indican los rangos donde el lucro es positivo.

Entre los puntos de quiebre es donde se obtiene un lucro positivo. Fuera de este rango, los ingresos no cubren los costos.

Referencias

Gómez, M., & Rodríguez, J. (2019). *GeoGebra y su aplicación en la enseñanza de las matemáticas*. Editorial Educativa.

Martínez, S., & Torres, A. (2020). *Simulación y modelado en ingeniería: Herramientas y aplicaciones*. Editorial Técnica.

Pérez, L., & López, J. (2021). *Tecnologías educativas: Integración de software en la enseñanza de las ciencias*. Editorial Académica.

Glosario de los términos citados

Matplotlib: Biblioteca de Python utilizada para crear visualizaciones estáticas, animadas e interactivas. Es particularmente útil para la generación de gráficos y diagramas en aplicaciones científicas y de ingeniería.

NumPy: Biblioteca fundamental para la computación científica en Python. Proporciona soporte para matrices y grandes conjuntos de datos multidimensionales, así como una colección de funciones matemáticas de alto nivel para operar con estos datos.

PUCE Virtual

La excelencia no se improvisa

síguenos

ICYS - 12 - 241