Programación orientada a objetos

Informe - Proyecto series de Taylor

Nicolas Steven Moreno Linares ID: 000756520

Introducción

El objetivo del presente informe es implementar las series de Taylor para diversas funciones matemáticas en Python con el lector de código VS Code, siguiendo el patrón de arquitectura Modelo-Vista-Controlador (MVC).

Ahora bien, este modelo permite la separación detallada entre la lógica matemática, la presentación de resultados y el control del flujo de ejecución, mejorando su estructura, lectura y posible mantenimiento del código.

Análisis

Se requiere implementar en este proyecto las siguientes funciones de las series de Taylor:

- Exponencial e^x
- Seno $sen\ x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$ para toda x
- Coseno $cos\ x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}$ para toda x
- Arcoseno $arcsen\ x = \sum_{n=0}^{\infty} \frac{(2n)!}{4n(n!)^2(2n+1)!} x^{2n+1}$ para toda $|x| < 1$
- Arcocoseno $arcos\ x = \frac{\pi}{2} - arcsen\ x$ para toda x
- Seno hiperbólico $senh\ x = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!}$ para toda x
- Coseno hiperbólico $cosh\ x = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}$ para toda x

A través del modelo Modelo-Vista-Controlador (MVC) permitiéndole al usuario ingresar el valor y el número de términos de la serie para darle un resultado con todas las funciones nombradas anteriormente. Sabiendo que internamente se debe llevar una programación orientada a objetos en el lector de código VS Code con la mejor claridad y simplicidad posible.

Diseño

Se entiende por Modelo-Vista-Controlador (MVC):

- **Modelo** (model) → Contiene la lógica matemática para calcular las Series de Taylor.
- **Vista (**view**)** → Se encarga de mostrar los resultados al usuario.
- **Controlador (**controller**)** → Gestiona la interacción entre el modelo y la vista.

Adicionalmente, el proyecto se estructuró de la siguiente forma:

P_Series_Taylor_MVC/

│── **model/** → Contiene la lógica matemática (cálculo de la serie de Taylor)

│── **view/** → Contiene la interfaz de usuario (mostrar resultados)

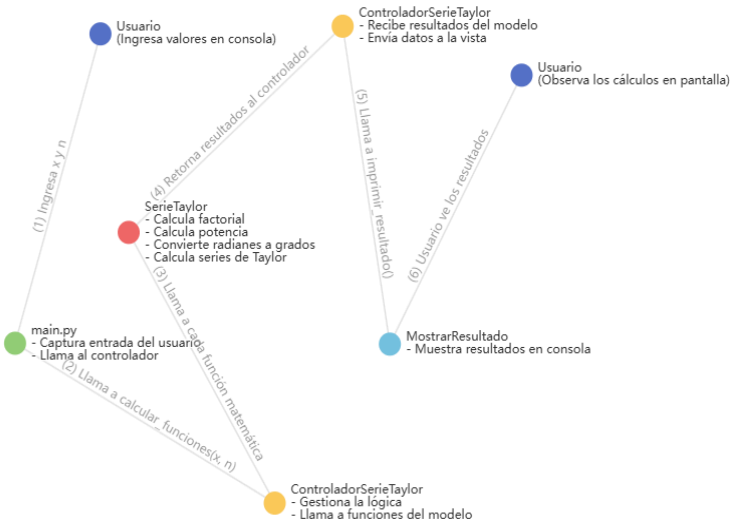
│── **controller/** → Contiene la lógica de control (conectar modelo y vista)

│── **main.py** → Punto de entrada del programa

│── **README.md** → Documentación del proyecto

│── **.gitignore** → Archivos a ignorar en Git

Por otra parte, se demuestra el diagrama de las clases que representa la estructura del proyecto y la relación entre los componentes Modelo-Vista-Controlador.



1. Se ingresan los valores (x y n) en la consola.
2. El archivo main.py captura la entrada y llama al controlador.
3. El controlador llama al modelo (Serie_Taylor) para calcular las funciones matemáticas.
4. El modelo realiza los cálculos y devuelve los resultados correspondientes al controlador.
5. El controlador envía los resultados a la vista (Mostrar_resultado).
6. La vista muestra los resultados para su respectiva validación.

Codificación

Se muestra la implementación del modelo, donde se define las funciones matemáticas para calcular las Series de Taylor.

```
from model.mdl_serie_taylor import Serie_Taylor
from view.vta_serie_taylor import Mostrar_resultado

class ControladorSerieTaylor:
    def __init__(self):
        self.modelo = Serie_Taylor()
        self.vista = Mostrar_resultado()

    def calcular_funciones(self, valor_x, cantidad_de_terminos=10):
        funciones = {
            "Exponencial (e^x)":
self.modelo.calcular_funcion_exponencial(valor_x, cantidad_de_terminos),
            "Seno (sen x)": self.modelo.calcular_funcion_seno(valor_x,
cantidad_de_terminos),
            "Coseno (cos x)":
self.modelo.calcular_funcion_coseno(valor_x, cantidad_de_terminos),
            "Arcsen (arcsen x)":
self.modelo.calcular_funcion_arcsen(valor_x, cantidad_de_terminos),
            "Arccos (arccos x)":
self.modelo.calcular_funcion_arccos(valor_x, cantidad_de_terminos),
            "Seno hiperbólico (senh x)":
self.modelo.calcular_funcion_senh(valor_x, cantidad_de_terminos),
            "Coseno hiperbólico (cosh x)":
self.modelo.calcular_funcion_cosh(valor_x, cantidad_de_terminos),
        }

        for nombre, resultado in funciones.items():
            self.vista.imprimir_resultado(resultado, nombre)
```

Seguido a ello se define como se pueden visualizar los resultados al usuario (view).

```
class Mostrar_resultado:
    @staticmethod
    def imprimir_resultado(resultado_de_la_operacion,
nombre_de_la_funcion):
        print(f"El resultado de la función {nombre_de_la_funcion} es:
{resultado_de_la_operacion:.6f}")
```

Adicionalmente, por medio del controlador se establece la comunicación entre el modelo y la vista.

```
class Serie_Taylor:
    @staticmethod
    def calcular_factorial(numero):
        """Se calcula el factorial de un número."""
        if numero == 0 or numero == 1:
            return 1
        factorial = 1
        for i in range(2, numero + 1):
```

```
            factorial *= i
        return factorial

    @staticmethod
    def calcular_potencia(base, exponente):
        """Se calcula la potencia de un número."""
        resultado = 1
        for _ in range(exponente):
            resultado *= base
        return resultado

    @staticmethod
    def convertir_a_grados(radianes):
        """Se convierte un valor en radianes a grados."""
        grados_por_radianes = 180 / 3.141592653589793  # Se hace una
proximación de π
        return radianes * grados_por_radianes

    @staticmethod
    def calcular_funcion_exponencial(valor_x, cantidad_de_terminos=10):
        """Se calcula la función exponencial (e^x) usando la serie de
Taylor."""
        resultado = 0
        for n in range(cantidad_de_terminos):
            resultado += Serie_Taylor.calcular_potencia(valor_x, n) /
Serie_Taylor.calcular_factorial(n)
        return resultado

    @staticmethod
    def calcular_funcion_seno(valor_x, cantidad_de_terminos=10):
        """Se calcula la función seno de valor_x usando la serie de Taylor
con 'cantidad_terminos' términos."""
        resultado_funcion_seno = 0
        for numero_de_terminos in range(cantidad_de_terminos):
            coeficiente = (-1)**numero_de_terminos
            numerador = Serie_Taylor.calcular_potencia(valor_x,
2*numero_de_terminos + 1)
            denominador =
Serie_Taylor.calcular_factorial(2*numero_de_terminos + 1)
            resultado_funcion_seno += coeficiente * (numerador /
denominador)
        return resultado_funcion_seno

    @staticmethod
    def calcular_funcion_coseno(valor_x, cantidad_de_terminos=10):
        """Se calcula la función coseno de valor_x usando la serie de
Taylor con 'cantidad_terminos' términos."""
        resultado_funcion_coseno = 0
        for numero_de_terminos in range(cantidad_de_terminos):
            coeficiente = (-1)**numero_de_terminos
            numerador = Serie_Taylor.calcular_potencia(valor_x,
2*numero_de_terminos)
            denominador =
Serie_Taylor.calcular_factorial(2*numero_de_terminos)
            resultado_funcion_coseno += coeficiente * (numerador /
denominador)
        return resultado_funcion_coseno
```

```
    @staticmethod
    def calcular_funcion_arcsen(valor_x, cantidad_de_terminos=10):
        """Se calcula la función arcsen(x) usando la serie de Taylor."""
        resultado = 0
        for n in range(cantidad_de_terminos):
            coeficiente = Serie_Taylor.calcular_factorial(2*n) /
(Serie_Taylor.calcular_potencia(4, n) *
Serie_Taylor.calcular_factorial(n)**2 * (2*n + 1))
            resultado += coeficiente *
Serie_Taylor.calcular_potencia(valor_x, 2*n + 1)
        return resultado

    @staticmethod
    def calcular_funcion_arccos(valor_x, cantidad_de_terminos=10):
        """Se calcula la función arccos(x) usando la serie de Taylor."""
        return (3.141592653589793 / 2) -
Serie_Taylor.calcular_funcion_arcsen(valor_x, cantidad_de_terminos)

    @staticmethod
    def calcular_funcion_senh(valor_x, cantidad_de_terminos=10):
        """Se calcula la función de seno hiperbólico usando la serie de
Taylor."""
        resultado = 0
        for n in range(cantidad_de_terminos):
            numerador = Serie_Taylor.calcular_potencia(valor_x, 2*n + 1)
            denominador = Serie_Taylor.calcular_factorial(2*n + 1)
            resultado += numerador / denominador
        return resultado

    @staticmethod
    def calcular_funcion_cosh(valor_x, cantidad_de_terminos=10):
        """Se calcula la función de coseno hiperbólico usando la serie de
Taylor."""
        resultado = 0
        for n in range(cantidad_de_terminos):
            numerador = Serie_Taylor.calcular_potencia(valor_x, 2*n)
            denominador = Serie_Taylor.calcular_factorial(2*n)
            resultado += numerador / denominador
        return resultado
```

Por último, se programa el archivo principal (main.py) el cual ejecuta el programa y permite la interacción con el usuario.

```
from controller.ctrl_serie_taylor import ControladorSerieTaylor

if __name__ == "__main__":
    controlador = ControladorSerieTaylor()

    valor_x = float(input("Ingrese el valor de x en radianes: "))
    cantidad_de_terminos = int(input("Ingrese el número de términos para la
serie de Taylor: "))

    controlador.calcular_funciones(valor_x, cantidad_de_terminos)
```

Conclusiones

- Se evidencia el uso de la arquitectura Modelo-Vista-Controlador (MVC) por medio de la programación en Python orientada a objetos para ejecutar la serie de Taylor para diversas funciones matemáticas.

- Se refleja uso correcto del código al ejecutarlo en el CMD del pc o en la terminar del lector de código VS Code.

- Se realiza el respectivo cargue del proyecto en GitHub.