Métodos Runge-Kuttas

March 5, 2023

1 Programa 1

[1]:
```
import pandas as pd
import numpy as np
pd.set_option('display.max_rows', None)
```

1.1 Métodos de Runge-Kutta-Merson y de grado 4

1.1.1 Rubrica

Elemento	Pts.
Cambia PVI	3
Lee valores iniciales	2
Pide h y tolerancia	2
Presenta tabla (x, y) correcta [R-K-4]	4
Presenta tabla (x, y) correcta [R-K-M]	4
Da opción de presentar tabla (K's)	3
Compara métodos con diferencia entre ellos	2
Permite volver a ejecutar y cambiar valores iniciales	2
Aplica control de error (Extra)	2

1.2 Código

[2]:
```
class RungeKuttas(object):

    # Encabezados
    header=list(("x","y","k1","k2","k3","k4","k5","Error"))

    # Constructor
    def __init__(self, initialValue:list, point:float, h_value:float, error:
 ↪float = 0.0, equation:str="0") -> None:
        self._initialValue = initialValue
        self.h_value = h_value
        self.error = error
        self.solTab = np.zeros((1,8),dtype=float)
        self.point = point
        self.equation = equation
```

```python
    # Lo que muestra al llamar la función print
    def __str__(self:any) -> str:
        self.rkmMethod()
        return "Objeto de tipo Método Runge-Kutta"

    #region Método de Runge-Kutta-4
    def rk4(self:any):
        self.firstRowOrder4()
        self.iterationsOrder4()
        return pd.DataFrame(self.solTab, columns=self.
↪header)[['x','y','k1','k2','k3','k4','Error']]

    # Método para crear la primera iteración de RK4
    def firstRowOrder4(self:any) -> None:
        x,y = self._initialValue[0], self._initialValue[1]
        self.solTab[0,0] = x
        self.solTab[0,1] = y
        self.solTab[0,2] = self.k1(x,y)
        self.solTab[0,3] = self.k2(x,y,self.solTab[0,2])
        self.solTab[0,4] = self.k3(x,y,self.solTab[0,3])
        self.solTab[0,5] = self.k4(x,y,self.solTab[0,2],self.solTab[0,3],self.
↪solTab[0,4])

    # Método para crear el resto de las iteraciones de RK4
    def iterationsOrder4(self:any) -> None:
        rango = int(np.abs((self._initialValue[0]-self.point)/self.h_value))
        for i in range(0,rango):
            yn = np.zeros(shape=(1,8))
            x,y = self.solTab[i,0] + self.h_value,self.solTab[i,1]
            yn[0,0] = x
            yn[0,1] = self.get_y(y=y, k1=self.solTab[i,2], k2=self.solTab[i,3],␣
↪k3=self.solTab[i,4], k4=self.solTab[i,5])
            y = yn[0,1]
            yn[0,2] = self.k1(x,y)
            yn[0,3] = self.k2(x,y,yn[0,2])
            yn[0,4] = self.k3(x,y,yn[0,3])
            yn[0,5] = self.k4(x,y,yn[0,2],yn[0,3],yn[0,4])
            self.solTab = np.append(self.solTab,yn, axis=0)
            del yn,x,y

    # Métodos para obtener las k del metodo de RK4
    def k1(self, x:float, y:float) -> float:
        return self.fdexy(x,y)

    def k2(self, x:float, y:float, k1:float) -> float:
        return self.fdexy(x + self.h_value/2, y + k1 * self.h_value /2)
```

```
    def k3(self, x:float, y:float, k2:float) -> float:
        return self.fdexy(x + self.h_value/2, y + k2 * self.h_value /2)

    def k4(self, x:float, y:float, k1:float, k2:float, k3:float) -> float:
        return self.fdexy(x + self.h_value, y + k3 * self.h_value )

    def get_y(self,y:float, k1:float, k2:float, k3:float, k4:float) -> float:
        """Return the next y with a global error of O(h^3)"""
        return y + (k1 + 2 * k2 + 2 * k3 + k4) * self.h_value/6

    #endregion

    #region Método de Runge-Kutta-Merson
    def rkmMethod(self:any) -> pd.DataFrame:
        self.firstRowMerson()
        self.iterationsMerson()
        return pd.DataFrame(self.solTab, columns=self.
↪header)[['x','y','k1','k2','k3','k4','k5','Error']]

    # Método para crear la primera iteración de Merson
    def firstRowMerson(self:any) -> None:
        x,y = self._initialValue[0], self._initialValue[1]
        self.solTab[0,0] = x
        self.solTab[0,1] = y
        self.solTab[0,2] = self.mersonk1(x,y)
        self.solTab[0,3] = self.mersonk2(x,y,self.solTab[0,2])
        self.solTab[0,4] = self.mersonk3(x,y,self.solTab[0,2],self.solTab[0,3])
        self.solTab[0,5] = self.mersonk4(x,y,self.solTab[0,2],self.
↪solTab[0,3],self.solTab[0,4])
        self.solTab[0,6] = self.mersonk5(x,y,self.solTab[0,2],self.
↪solTab[0,3],self.solTab[0,4],self.solTab[0,5])

    # Método para crear el resto de las iteraciones de Merson
    def iterationsMerson(self) -> None:
        rango = int(np.abs((self._initialValue[0]-self.point)/self.h_value))
        for i in range(0,rango):
            yn = np.zeros(shape=(1,8))
            x,y = self.solTab[i,0] + self.h_value,self.solTab[i,1]
            yn[0,0] = x
            yn[0,1] = self.get_yMerson(y=y, k1=self.solTab[i,2], k3=self.
↪solTab[i,4], k4=self.solTab[i,5], k5=self.solTab[i,6])
            y = yn[0,1]
            yn[0,2] = self.mersonk1(x,y)
            yn[0,3] = self.mersonk2(x,y,yn[0,2])
            yn[0,4] = self.mersonk3(x,y,yn[0,2],yn[0,3])
            yn[0,5] = self.mersonk4(x,y,yn[0,2],yn[0,3],yn[0,4])
```

```python
            yn[0,6] = self.mersonk5(x,y,yn[0,2],yn[0,3],yn[0,4],yn[0,5])
            self.solTab = np.append(self.solTab,yn, axis=0)
            del yn,x,y

    # Esta función recibirá una cadena que representará la función, usaremos la función "eval"
    def fdexy(self,x:float, y:float) -> float:
        """This is the equation we want to get the numeric solution"""
        # Notita: las variables x, y si se ocupan ¡aunque no lo parezca!
        return eval(self.equation)

    # Métodos para obtener las k del metodo de Merson
    def mersonk1(self, x:float, y:float) -> float:
        return self.h_value * self.fdexy(x,y)

    def mersonk2(self, x:float, y:float, k1:float) -> float:
        return self.h_value * self.fdexy(x + self.h_value/3, y + k1/3)

    def mersonk3(self, x:float, y:float, k1:float, k2:float) -> float:
        return self.h_value * self.fdexy(x + self.h_value / 3, y + k1/6 + k2/6)

    def mersonk4(self, x:float, y:float, k1:float, k2:float, k3:float) -> float:
        return self.h_value * self.fdexy(x + self.h_value / 2, y + k1 / 8 + 3/8 * k3)

    def mersonk5(self,x:float,y:float, k1:float, k2:float, k3:float, k4:float) -> float:
        return self.h_value * self.fdexy(x + self.h_value, y + k1/2 - 3/2 * k3 + 2*k4)

    def mersonk6(self, x:float, y:float, k1:float, k2:float, k3:float, k4:float, k5:float) -> float:
        return self.h_value * self.fdexy(x + self.h_value/2, y - 8/27*k1 + 2*k2 - 3544/2565*k3 + 1859/4104*k4 - 11/40*k5)

    # Regresa la y de Merson
    def get_yMerson(self,y:float, k1:float, k3:float, k4:float, k5:float) -> float:
        """Return the next y with a global error of O(h^4)"""
        return y + (k1 + 4*k4 + k5)/6

    #endregion
```

1.2.1 Tabla con el método de grado 4

[3]:
```
tab1=RungeKuttas(initialValue=(0,3),h_value=0.5, point=7, equation="-0.06*y**(1/
↪2)").rk4()
tab1
```

[3]:

	x	y	k1	k2	k3	k4	Error
0	0.0	3.000000	-0.103923	-0.103472	-0.103474	-0.103023	0.0
1	0.5	2.948263	-0.103023	-0.102572	-0.102574	-0.102123	0.0
2	1.0	2.896977	-0.102123	-0.101672	-0.101674	-0.101223	0.0
3	1.5	2.846140	-0.101223	-0.100772	-0.100774	-0.100323	0.0
4	2.0	2.795754	-0.100323	-0.099872	-0.099874	-0.099423	0.0
5	2.5	2.745817	-0.099423	-0.098972	-0.098974	-0.098523	0.0
6	3.0	2.696331	-0.098523	-0.098072	-0.098074	-0.097623	0.0
7	3.5	2.647294	-0.097623	-0.097172	-0.097174	-0.096723	0.0
8	4.0	2.598708	-0.096723	-0.096272	-0.096274	-0.095823	0.0
9	4.5	2.550571	-0.095823	-0.095372	-0.095374	-0.094923	0.0
10	5.0	2.502885	-0.094923	-0.094472	-0.094474	-0.094023	0.0
11	5.5	2.455648	-0.094023	-0.093572	-0.093574	-0.093123	0.0
12	6.0	2.408862	-0.093123	-0.092672	-0.092674	-0.092223	0.0
13	6.5	2.362525	-0.092223	-0.091772	-0.091774	-0.091323	0.0
14	7.0	2.316639	-0.091323	-0.090872	-0.090874	-0.090423	0.0

1.2.2 Tabla con el método de R-K-M

[4]:
```
tab2=RungeKuttas(initialValue=(0,3),h_value=0.5, point=7, equation="-0.06*y**(1/
↪2)").rkmMethod()
tab2
```

[4]:

	x	y	k1	k2	k3	k4	k5	Error
0	0.0	3.000000	-0.051962	-0.051811	-0.051812	-0.051737	-0.051512	0.0
1	0.5	2.948263	-0.051512	-0.051361	-0.051362	-0.051287	-0.051062	0.0
2	1.0	2.896977	-0.051062	-0.050911	-0.050912	-0.050837	-0.050612	0.0
3	1.5	2.846140	-0.050612	-0.050461	-0.050462	-0.050387	-0.050162	0.0
4	2.0	2.795754	-0.050162	-0.050011	-0.050012	-0.049937	-0.049712	0.0
5	2.5	2.745817	-0.049712	-0.049561	-0.049562	-0.049487	-0.049262	0.0
6	3.0	2.696331	-0.049262	-0.049111	-0.049112	-0.049037	-0.048812	0.0
7	3.5	2.647294	-0.048812	-0.048661	-0.048662	-0.048587	-0.048362	0.0
8	4.0	2.598708	-0.048362	-0.048211	-0.048212	-0.048137	-0.047912	0.0
9	4.5	2.550571	-0.047912	-0.047761	-0.047762	-0.047687	-0.047462	0.0
10	5.0	2.502885	-0.047462	-0.047311	-0.047312	-0.047237	-0.047012	0.0
11	5.5	2.455648	-0.047012	-0.046861	-0.046862	-0.046787	-0.046562	0.0
12	6.0	2.408862	-0.046562	-0.046411	-0.046412	-0.046337	-0.046112	0.0
13	6.5	2.362525	-0.046112	-0.045961	-0.045962	-0.045887	-0.045662	0.0
14	7.0	2.316639	-0.045662	-0.045511	-0.045512	-0.045437	-0.045212	0.0

2 Interfaz

[5]:
```python
import wx
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[5], line 1
----> 1 import wx

ModuleNotFoundError: No module named 'wx'
```

[ ]:
```python
# Create the Application Object
app = wx.App()
# Now create a Frame (representing the window)
frame = wx.Frame(parent=None, title='Simple Hello World') # And add a text␣
 ↪label to it
text = wx.StaticText(parent=frame, label='Hello Python')
# Display the window (frame)
frame.Show()
# Start the event loop
app.MainLoop()
```

[ ]:
```python
import tkinter as tk

def save_info():
    info = []
    for entry in entries:
        info.append(entry.get())
    print(info)

root = tk.Tk()
root.title("Mi Ventana")

entries = []

for i in range(5):
    label = tk.Label(root, text=f"Ingrese la cadena #{i+1}")
    label.pack()
    entry = tk.Entry(root)
    entry.pack()
    entries.append(entry)

button = tk.Button(root, text="Guardar información", command=save_info)
button.pack()

root.mainloop()
```

```
[ ]: import tkinter as tk
     from tkinter import ttk
     import pandas as pd

     class App:
         def __init__(self, master):
             self.master = master
             master.title("Interfaz gráfica")

             # Crear los widgets para la entrada de datos
             tk.Label(master, text="Valor x ").grid(row=0, column=0)
             self.float1_entry = tk.Entry(master)
             self.float1_entry.grid(row=0, column=1)
             tk.Label(master, text="Valor y").grid(row=1, column=0)
             self.float2_entry = tk.Entry(master)
             self.float2_entry.grid(row=1, column=1)
             tk.Label(master, text="Valor h").grid(row=2, column=0)
             self.float3_entry = tk.Entry(master)
             self.float3_entry.grid(row=2, column=1)
             tk.Label(master, text="Punto a interpolar").grid(row=3, column=0)
             self.float4_entry = tk.Entry(master)
             self.float4_entry.grid(row=3, column=1)
             tk.Label(master, text="Ecuación ").grid(row=5, column=0)
             self.cadena_entry = tk.Entry(master)
             self.cadena_entry.grid(row=5, column=1)

             # Crear el botón para enviar los datos y mostrar el DataFrame
             self.button = tk.Button(master, text="Mostrar DataFrame", command=self.
      ↪mostrar_dataframe)
             self.button.grid(row=6, column=1)

         def mostrar_dataframe(self,df:pd.DataFrame):
             # Obtener los datos de las entradas
             # float1 = float(self.float1_entry.get())
             # float2 = float(self.float2_entry.get())
             # float3 = float(self.float3_entry.get())
             # float4 = float(self.float4_entry.get())
             # cadena = self.cadena_entry.get()

             # # Crear un diccionario con los datos
             # data = {"Float 1": [float1], "Float 2": [float2], "Float 3":␣
      ↪[float3], "Float 4": [float4], "Float 5": [float5], "Cadena": [cadena]}

             # Crear un DataFrame a partir del diccionario y mostrarlo en una tabla
             # df = pd.DataFrame(data)
             window = tk.Toplevel(self.master)
             window.title("DataFrame")
```

```
        frame = tk.Frame(window)
        frame.pack(fill="both", expand=True)
        table = ttk.Treeview(frame, columns=df.columns, show="headings")
        table.pack(side="left", fill="both", expand=True)
        for col in df.columns:
            table.heading(col, text=col)
        for index, row in df.iterrows():
            table.insert("", "end", values=[row[col] for col in df.columns])

root = tk.Tk()
app = App(root)
root.mainloop()
```

[ ]:
```
import tkinter as tk
import pandas as pd

# Crear DataFrame de ejemplo
data = {'Nombre': ['Juan', 'Ana', 'Pedro'], 'Edad': [25, 30, 35]}
df = pd.DataFrame(data)

# Crear ventana de tkinter
root = tk.Tk()

# Crear widget Text para mostrar DataFrame
text_widget = tk.Text(root)
text_widget.pack()

# Convertir DataFrame a cadena y mostrarlo en el widget Text
text_widget.insert('1.0', df.to_string())

# Ejecutar loop principal de la ventana de tkinter
root.mainloop()
```

[ ]:
```

```