TP2:dif_num

April 26, 2025

1 TP de Diferenciación Numérica

1.1 Ejercicio especial Covid-19

Grupo 4:

- Costa Felipe
- Gutierrez Juan Santiago
- Indeau Federico
- Lucero Julieta

1.1.1 Código

Utilizando python 3.12

Se definieron las siguientes funciones, El código se aloja en github

[2]:
```
import numpy as np
import math
import pandas as pd
import scipy as sc

def generar_indices(n: int, traslacion:int = 0) -> np.array:
    """Genera un array simétrico de n/2 enteros alrededor de cero incluyendo el␣
 ↪0 \n
    ,y si n es par prefiere a la derecha \n
    Traslada el array antes de devolverlo"""
    array = np.arange(-int(n/2), int(n/2)+1)
    if n%2 == 0:
        array = array[1:]
    return array + traslacion

def gen_matrix(derivada: int = 1, truncamiento: int = 2, traslacion: int = 0)␣
 ↪-> np.ndarray:
    """Crea una matrix cuadrada para el calculo una n-derivada usando␣
 ↪expansiones de Taylor alrededor de un punto"""
    puntos = derivada + truncamiento
    indices = generar_indices(puntos, traslacion)
```

```
    # Definir una matrix cuadrada vacia
    A = np.ones((puntos,puntos))

    # Los elementos de la matrix se calculan así
    def coef(i:int ,j:int) -> float:
        return math.pow(j,i) / math.factorial(i)

    # Recorrer la matrix y generar cada elemento
    for i in range(puntos):
        for j in indices:
            k = j + sum(1 for i in indices if indices[i] < 0) # Ajusta el indice j, con el indice de la matrix de forma que indices[0] -> i,0 de la matrix
            A[i][k] = round(coef(i=i,j=j), truncamiento + 2)

    # Retornar la matrix
    return A

def coef_derivada(matrix: np.ndarray[float], derivada:int =1) -> np.ndarray[float]:
    """gen_matrix() -> A\n
    Matrix A para derivadas, aplica las condiciones para la n-derivada\n
    devolviendo los coeficientes X \n
    X -> evaluar_derivada()"""
    # Define un vector vacio y aplica las condiciones
    v = np.zeros((len(matrix)))
    v[derivada] = 1
    # Resuelve el sistema y lo retorna
    return np.linalg.solve(a=matrix, b=v)

def evaluar_derivada(X:np.ndarray,f:np.ufunc ,x:float=0.0, dx: float = 0.1, traslacion:int = 0, derivada:int = 1) -> float:
    """coef_derivada() -> X \n
    Evalua la derivada en un punto en base a los coeficientes X y a cual derivada se dirije \n
    -> f'(x)"""
    df = 0
    # Usa los coeficientes de X y los índices de los puntos para calcular df, recorriendo ambos arrays en simultaneo.
    for i,j in zip(X, generar_indices(len(X), traslacion)):
        df += i * f(x + j * dx )
    return df / math.pow(dx,derivada)

def derivar(x:float,f:np.ufunc,dx:float,derivada: int =1, orden:int = 2, traslacion:int=0)-> float:
```

```
    """Deriva una funcion númericamente, usando expansiones de Taylor y orden␣
 ↪de Landau"""
    # Calcular la matrix, por defecto da la 1er derivada
    A = gen_matrix(derivada=derivada,truncamiento=orden,traslacion= traslacion)

    # Calcular los coeficientes
    X = coef_derivada(matrix=A,derivada=derivada)

    # Evaluar la derivada
    df = evaluar_derivada(X=X, f=f, x=x, dx=dx,traslacion=traslacion,␣
 ↪derivada=derivada)
    return df

def extrapolacion_Richarson(D0:float, D1:float, dx0:float, dx1:float, n:int)->␣
 ↪tuple[float]:
    """Aproximación de Richarson, El orden aumenta en e^(n+2)"""
    df = np.round(D0 + ((D0 - D1)/(math.pow((dx1/dx0),n)-1)),5)
    err = abs(np.round((D0-D1)/(math.pow(dx1,n)-math.pow(dx0,n)),5))
    return (df, err)
```

1.1.2 Datos y guía

Con las funciones anteriormente definidas se realizan los ejercicios en la guía de ejercicios: * *Ejercicio especial Derivacion 2021.pdf*

Con datos extraídos de https://www.argentina.gob.ar/coronavirus/informe-diario en uno de los reportes diarios. * En partícular: *"Reporte diario COVID-19.csv"*

1.1.3 Resolución

- La resolución consta de el código realizado y la salida. Para los ejercicios 1, 2, 3, 5 se utiliza el siguiente código.
- Se presentan los 5 primeros datos.

[15]:
```
DF = pd.read_csv("../Data/Reporte diario COVID-19.csv")
```

[16]:
```
"""1. Aproxime la derivada primera para cada día con un esquema de orden 2 y un␣
 ↪paso de 1 día."""
def ejer1(DF):
    # Variables
    traslacion = 1
    derivada = 1
    orden = 2
    pasos = 1 #
    def contagios(dia):
        return DF["acumulado"][dia-1]
    L = []
    for dia in DF["dia"]:
```

```
        traslacion = 0
        while True:
            try:
                df = derivar(x=dia, f=contagios, dx=pasos, derivada=derivada,␣
 ↪orden=orden, traslacion=traslacion)
                break
            except:
                if dia < len(DF)//2: traslacion+=1
                else: traslacion-=1
        L += [df]
    DF = DF.assign(df_ejer1 = L)
    print(DF.head())
    return DF
DF = ejer1(DF)
```

```
   dia  nuevo  acumulado  df_ejer1
0    1      1          1      -3.5
1    2      1          2       5.5
2    3      3         12       7.5
3    4      5         17       3.5
4    5      2         19       2.0
```

[17]:
```
def ejer2(DF):
    # Variables
    traslacion = 1
    derivada = 1
    orden = 2
    pasos = 2 #
    def contagios(dia):
        return DF["acumulado"][dia-1]
    L = []
    for dia in DF["dia"]:
        traslacion = 0
        while True:
            try:
                df = derivar(x=dia, f=contagios, dx=pasos, derivada=derivada,␣
 ↪orden=orden, traslacion=traslacion)
                break
            except:
                if dia < len(DF)//2: traslacion+=1
                else: traslacion-=1
        L += [df]
    DF = DF.assign(df_ejer2 = L)
    print(DF.head())
    return DF
DF = ejer2(DF)
```

```
   dia  nuevo  acumulado  df_ejer1  df_ejer2
```

```
0    1      1           1      -3.5      6.50
1    2      1           2       5.5     10.25
2    3      3          12       7.5      4.50
3    4      5          17       3.5      4.75
4    5      2          19       2.0      4.75
```

[18]:
```python
"""3. Mejore la aproximación usando la extrapolación de Richardson O(dx4)."""
def ejer3(DF):
    def D1(dia):
        return DF["df_ejer1"][dia-1]
    def D0(dia):
        return DF["df_ejer2"][dia-1]
    L = []
    L1= []
    for dia in DF["dia"]:
        df = extrapolacion_Richarson(D0=D0(dia), D1=D1(dia), dx0=2, dx1=1, n=2)
        L += [df[0]]
        L1 += [df[1]]
    DF = DF.assign(df1_Richarson_4_O = L)
    DF = DF.assign(df1_Richarson_error = L1)
    print(DF.head())
    return DF
DF = ejer3(DF)
```

```
   dia  nuevo  acumulado  df_ejer1  df_ejer2  df1_Richarson_4_O  \
0    1      1          1      -3.5      6.50           -6.83333
1    2      1          2       5.5     10.25            3.91667
2    3      3         12       7.5      4.50            8.50000
3    4      5         17       3.5      4.75            3.08333
4    5      2         19       2.0      4.75            1.08333

   df1_Richarson_error
0              3.33333
1              1.58333
2              1.00000
3              0.41667
4              0.91667
```

[12]:
```python
"""4. Determine una fórmula de derivación centrada que sea de orden O(dx )."""
def ejer4(DF):
    derivada = 1
    traslacion = 0
    orden = 4
    A = gen_matrix(derivada=derivada, truncamiento=orden, traslacion=traslacion)
    X = coef_derivada(A, derivada=derivada)
    X = np.round(X,5)
    print(f"({X[0]}*f-2 + {X[1]}*f-1+ {X[2]}*f0 + {X[3]}*f1+ {X[4]}*f2)/dx")
    return DF
```

[25]:
```
"""5. Aproxime la derivada primera para cada día con el esquema de orden 4 y un␣
 ↪paso de 1 día."""
def ejer5(DF):
    # Variables
    traslacion = 0
    derivada = 1
    orden = 4
    pasos = 1
    def contagios(dia):
        return DF["acumulado"][dia-1]
    L = []
    for dia in DF["dia"]:
        traslacion = 0
        while True:
            try:
                df = derivar(x=dia, f=contagios, dx=pasos, derivada=derivada,␣
 ↪orden=orden, traslacion=traslacion)
                break
            except:
                if dia < len(DF)//2: traslacion+=1
                else: traslacion-=1
        L += [df]
    DF = DF.assign(df_ejer5= L)
    print(DF.head(5))
    return DF
DF = ejer5(DF)
```

```
   dia  nuevo  acumulado  df_ejer1  df_ejer2  df1_Richarson_4_0  \
3    4      5         17       3.5      4.75            3.08333
4    5      2         19       2.0      4.75            1.08333
5    6      2         21       6.0      4.25            6.58333
6    7     10         31       6.5      6.50            6.50000
7    8      3         34       7.0      8.75            6.41667

   df1_Richarson_error   df_ejer5
3              0.41667  10.416577
4              0.91667  -1.250009
5              0.58333   6.583335
6              0.00000   6.500000
7              0.58333   6.416665
```