mn_tarea01

August 29, 2023

1 Curso de Métodos Numéricos (DEMAT)

1.1 Tarea 1

Descripción:	Fechas
Fecha de publicación del documento:	**Agosto 19, 2023**
Fecha límite de entrega de la tarea:	**Agosto 27, 2023**

1.1.1 Indicaciones

El propósito de esta tarea es poner en praćtica lo que hemos revisado sobre Python, por lo que los ejercicios son de programación.

Puede escribir el código de los algoritmos que se piden en una celda de este notebook o si lo prefiere, escribir las funciones en un archivo `.py` independiente e importar la funciones para usarlas en este notebook. Lo importante es que en el notebook aparezcan los resultados de la pruebas realizadas y que:

- Si se requieren otros archivos para poder reproducir los resultados, para mandar la tarea cree un archivo ZIP en el que incluya el notebook y los archivos adicionales.
- Si todos los códigos para que se requieren para reproducir los resultados están en el notebook, no hace falta comprimirlo y puede anexar sólo el notebook en la tarea del Classroom.
- Exportar el notebook a un archivo PDF y anexarlo en la tarea del Classroom como un archivo independiente. **No lo incluya dentro del ZIP**, porque la idea que lo pueda accesar directamente para poner anotaciones y la calificación de cada ejercicio.

En la descripción de los ejercicios se nombran algunas variables para el algoritmo, pero sólo es para facilitar la descripción. En la implementación pueden nombrar sus variables como gusten.

En los algoritmos se describen las entradas de las funciones. La intención es que tomen en cuenta lo que requiere el algoritmo y que tiene que haber parámetros que permitan controlar el comportamiento del algoritmo, evitando que dejen fijo un valor y que no se puede modificar para hacer diferentes pruebas. Si quieren dar esta información usando un tipo de dato que contenga todos los valores o usar variables por separado, etc., lo pueden hacer y no usen variables globales si no es necesario.

Lo mismo para los valores que devuelve una función. Pueden codificar como gusten la manera en que regresa los cálculos. El punto es que podamos tener acceso a los resultados, sin usar variables globales, y que la función no sólo imprima los valores que después no los podamos usar.

1.2 Ejercicio 1 (4 puntos)

Programar una función que imprime los primeros y los últimos los elementos de un arreglo 1D.

1. Escribir una función que reciba un arreglo 1D v, un entero n y una cadena que indique el formato en que las cantidades que se van a imprimir, por ejemplo "%.3f", "%.4e", "%d".

- La secuencia de valores los tiene que imprimir en una sola fila.
- Si la longitud de v es menor o igual que $2n$, la función debe imprimir todos los elementos de v usando el formato indicado. Si no, entonces la función debe imprimir los primeros n elementos de v, poner "..." e imprimir los últimos n elementos de v.

2. Pruebe la función usando los arreglos que se definen en la celda que aparece más abajo. Siga las indicaciones que aparecen en los comentarios para saber las pruebas que tiene que realizar.

Por ejemplo, el resultado que se espera al mandar a ejecutar la función con el arreglo

```
[ 2.70778878  1.16223895 -0.9442264  -4.93729647  3.09544082
 -0.37185081 -3.99943434  4.73035645  2.54931701  0.2270238 ]
```

Cuando se usa $n = 3$ y "% .3f" para el formato de los números, se debe obtener algo como:

```
 2.708  1.162 -0.944  ...  4.730  2.549  0.227
```

Si cambiamos la cadena de formato a "% .1e" se debe obtener:

```
 2.71e+00  1.16e+00 -9.44e-01  ...  4.73e+00  2.55e+00  2.27e-01
```

> Note que en la cadena de formato hay un espacio entre "%" y ".1e". Esto es para indicar que al imprimir el valor reserve un espacio para el signo de la cantidad. También note que hay un espacio después de ".1e", que sirve para separar los elementos.
>
> Revise el notebook de la clase 1 para ver las opciones para dar formato a las cadenas de caracteres y la función `print` para imprimirlas.

1.2.1 Solución:

[ ]:
```python
# En esta celda puede poner el código de la función
# o poner la instrucción para importar la función de un archivo .py
def imp(arr, n, format):
    print('[', end='')
    for a in arr[:min(n,(len(arr)+1)//2)]:
        print(format % a, end='')
    for a in arr[max(len(arr)-n,(len(arr)+1)//2):]:
        print(format % a, end='')
    print(']')
```

[ ]:
```python
# Esta celda es para generar los datos de prueba.
# Puede agregar aquí las instrucciones que muestren los resultados
# o hacerlo en celdas adicionales.

import numpy as np
```

```
v1 = np.linspace(1.2, 8.6, 6)
v2 = np.concatenate((-v1[::-1], v1))
print('v1=', v1)
print('v2=', v2)

# Imprimir v1 y v2 usando:
# n=4 la cadena "% .3f "
print('v1= ', end='')
imp(v1, 4, "% .3f ")
print('v2= ', end='')
imp(v2, 4, "% .3f ")
# n=3 la cadena "% .3e "
print('v1= ', end='')
imp(v1, 3, "% .3e ")
print('v2= ', end='')
imp(v2, 3, "% .3e ")
# n=5 la cadena % 2d "
print('v1= ', end='')
imp(v1, 5, "% .2d ")
print('v2= ', end='')
imp(v2, 5, "% .2d ")
```

```
v1= [1.2  2.68 4.16 5.64 7.12 8.6 ]
v2= [-8.6  -7.12 -5.64 -4.16 -2.68 -1.2   1.2   2.68  4.16  5.64  7.12  8.6 ]
v1= [ 1.200  2.680  4.160  5.640  7.120  8.600 ]
v2= [-8.600 -7.120 -5.640 -4.160  4.160  5.640  7.120  8.600 ]
v1= [ 1.200e+00  2.680e+00  4.160e+00  5.640e+00  7.120e+00  8.600e+00 ]
v2= [-8.600e+00 -7.120e+00 -5.640e+00  5.640e+00  7.120e+00  8.600e+00 ]
v1= [ 01  02  04  05  07  08 ]
v2= [-08 -07 -05 -04 -02  02  04  05  07  08 ]
```

1.3 Ejercicio 2 (4 puntos)

Tenemos que

$$\log(x+1) \approx \sum_{i=1}^{n} \frac{1}{i} \left(\frac{x}{x+1} \right)^{i} = Lxp1(x,n).$$

1. Programe la función $Lxp1(x,n)$, donde x es un número flotante y el número de n es la cantidad de términos en la suma. La función devuelve la suma de los términos.
2. Importe la librería `numpy` para poder usar la función `numpy.log`
3. Escriba una función que reciba un arreglo de valores y el entero n. Esta función tiene un ciclo que itera sobre los valores x del arreglo dado. En cada iteración hay que imprimir el valor x, el valor de $numpy.log(1+x)$, el valor $Lxp1(x,n)$ y el error absoluto $numpy.abs(numpy.log(1+x) - Lxp1(x,n))$
4. Pruebe la función anterior para $n = 10, 100, 1000$ para ver que también aproxima la función

Lxp1 a la función $\log(x+1)$ en los puntos $x = -0.55, -0.5, 0.5, 0.95, 100$, y observar si la aproximación mejora conforme se incluyen más términos en la suma.

1.3.1 Solución:

[ ]:
```
# En esta celda puede poner el código de las funciones
# o poner la instrucción para importar la función de un archivo .py
def Lxp1(x,n):
    return sum([1/i*(x/(x+1))**i for i in range(1,n+1)])

def test(lx, n):
    for x in lx:
        print(x, np.log(1+x), Lxp1(x,n), np.abs(np.log(x+1)-Lxp1(x,n)))
```

[ ]:
```
# Prueba de la función

lx = [-0.55, -0.5, 0.5, 0.95, 100]
test(lx,10)
```

```
-0.55 -0.7985076962177717 -0.40787871010566223 0.3906289861121095
-0.5 -0.6931471805599453 -0.6456349206349207 0.047512259925024614
0.5 0.4054651081081644 0.40546436817203463 7.399361297566465e-07
0.95 0.6678293725756554 0.6677688464802967 6.052609535878162e-05
100 4.61512051684126 2.832125695235509 1.7829948216057505
```

1.4 Ejercicio 3 (2 puntos)

1. Programe la función `comparacion(a,b)`, la cual recibe dos valores, a y b. La función imprime el mensaje "Los valores a y b son iguales" si se cumple que a es igual a b. En caso contrario, imprime el mensaje "Los valores de a y b son diferentes".
2. Ejecute las instrucciones de las siguientes celdas para comparar los valores a y b dados.
3. Escriba un comentario sobre los resultados obtenidos.

[ ]:
```
# En esta celda puede poner el código de la función
# o poner la instrucción para importarlas de un archivo .py
def comparacion(a,b):
    print("Los valores a y b son iguales" if a == b else "Los valores de a y b␣
 ↪son diferentes")
```

[ ]:
```
# Pruebas

c = 893.0
a = (1.0/c)*c
b = 1.0
comparacion(a,b)

c = 765.0
a = (1.0/c)*c
```

```
b = 1.0
comparacion(a,b)

c = np.sqrt(2.0)
a = c*c
b = 2.0
comparacion(a,b)
```

```
Los valores a y b son iguales
Los valores de a y b son diferentes
Los valores de a y b son diferentes
```

Comentarios:

Debido a la precisión del punto flotante podemos observar que a pesar de aplicar una función y su función inversa no obtenemos el mismo resultado.