Curso de Métodos Numéricos (DEMAT)

Tarea 3

Descripción:	Fechas
Fecha de publicación del documento:	**Agosto 30, 2023**
Fecha límite de entrega de la tarea:	**Septiembre 10, 2023**

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 comprimir el notebook y puede anexar este archivo en la tarea del Classroom.
- Exportar el notebook a un archivo PDF y anexarlo en la tarea del Classroom como un archivo independiente. **No incluya el PDF 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.

Ejercicio 1 (puntos)

1. Calcule la suma de los elementos del arreglo $[101.1, 101.1, 101.1, -202.2, -101.1]$, sumando los elementos en el orden que tienen en el arreglo. Imprima el valor de la suma.

2. Repita lo anterior sumando los elementos en orden inverso.
3. Programe el algoritmo de Kahan (Clase 6, diapositiva 28). La función recibe un arreglo de valores y devuelve la suma de los elementos.
4. Calcule e imprima el valor que devuelve el algoritmo de Kahan dando como entrada el arreglo delprimer inciso y ese arreglo en orden inverso.
5. Repita la prueba anterior usando el arreglo $[101.1, 101.1, 101.1, -303.3]$. ¿Afecta el orden en que se suman los números en el resultado del algoritmo?

Solución:

```
from IPython.lib.display import Iterable
# Esta celda es para escribir el código de la función o importarla de un archivo

def kahanSum(arr: Iterable):
  sum = 0
  err = 0
  for a_k in arr:
    y = a_k - err # aplicar correccion
    t = sum + y # incrementar sum
    err = (t-sum) -y # actualizar error
    sum = t # actualizar suma
  return sum


```

En este caso usaremos la función `sum` que incorpora `python`, pues segun la documentación suma los elemntos en un `itearable` en orden.
Entonces para sumar en orden inverso, simplemente "invertimos" el orden del `iterable`.

```
# Pruebas
a = [101.1,101.1,101.1,-202.2,-101.1]

# Suma normal de elementos
print(f"El arreglo es {a}")
print(f"El valor de la suma en orden es: {sum(a)}")
print(f"El valor de la suma en orden inverso es: {sum(reversed(a))}")

# Suma usnado el algoritmo de Kahan
print(f"El valor de la suma con Kahan es: {kahanSum(a)}")
print(f"El valor de la suma con Kahan en orden inverso es: {kahanSum(reversed(a))}")

print()

b =   [101.1,101.1,101.1,-303.3]
print(f"El arreglo es {b}")
print(f"El valor de la suma en orden es: {sum(b)}")
print(f"El valor de la suma en orden inverso es: {sum(reversed(b))}")

# Suma usnado el algoritmo de Kahan
print(f"El valor de la suma con Kahan es: {kahanSum(b)}")
```

```
print(f"El valor de la suma con Kahan en orden inverso es: {kahanSum(reversed(b))}")

```

```
El arreglo es [101.1, 101.1, 101.1, -202.2, -101.1]
El valor de la suma en orden es: -2.842170943040401e-14
El valor de la suma en orden inverso es: 2.842170943040401e-14
El valor de la suma con Kahan es: 0.0
El valor de la suma con Kahan en orden inverso es: 0.0

El arreglo es [101.1, 101.1, 101.1, -303.3]
El valor de la suma en orden es: -5.684341886080802e-14
El valor de la suma en orden inverso es: -2.842170943040401e-14
El valor de la suma con Kahan es: 0.0
El valor de la suma con Kahan en orden inverso es: -2.842170943040401e-14
```

De lo anterior podemos notar que el orden de los elementos a la hora de sumar si afecta la suma.

–

Ejercicio 2 (puntos)

Considere la función $f(x) = x + \ln\sqrt{x} - 2.5$. Verique que la función $g(x) = 2.5 - \ln\sqrt{x}$ cumple con las condiciones del teorema de punto fijo en el intervalo $I = [1, 3]$, de modo que puede usarse para encontrar una raíz de $f(x)$.

Estime el número de iteraciones que requiere el algoritmo de punto fijo para aproximar a la raíz de $f(x)$ con un error menor a 10^{-6}, partiendo de cualquier punto en el intervalo I.

Solución:

Primero notemos que g es continua, pues composición de funciones continuas (en el intervalo $[1, 3]$). Sea $x \in [1, 3]$, entonces $1 \leq x \leq 3$, lo cual implica que $1 \leq \sqrt{x} \leq \sqrt{3}$, luego $0 \leq \ln\sqrt{x} \leq \ln\sqrt{3}$, como $\sqrt{3} = e^{1/2 \ln 3}$ entonces $\ln\sqrt{3} = 1/2 \ln 3 \approx 0.54$. De lo anterior tenemos que $-0.54 \leq -\ln\sqrt{x} \leq 0$ y por tanto $1.95 \leq 2.5 - \ln\sqrt{x} \leq 2.5$, es decir, $g(x) \in [1, 3]$.

Más aún, notemos que

$$g'(x) = -\frac{1}{\sqrt{x}}\frac{1}{2}\frac{1}{\sqrt{x}} = -\frac{1}{2x} \implies |g'(x)| = \frac{1}{2x}.$$

Si $x \in [1,3]$ entonces $1/2 \geq 1/2x \geq 1/6$, por lo cual g tiene un unico punto fijo.

En la demostración del Teorema vimos lo siguiente:

$$|x_{n+1} - x^*| \leq \rho^{n+1}|x_0 - x^*|,$$

entonces tenemos que $\rho = 1/2$, la maxima distancia entre un punto inical y el punto fijo es $3 - 1 = 2$, por lo cual buscamos n tal que

$$|x_{n+1} - x^*| \leq \left(\frac{1}{2}\right)^{n+1}(2) = \left(\frac{1}{2}\right)^{n} \leq 10^{-6},$$

para ello notemos que

$$\begin{aligned}\left(\frac{1}{2}\right)^{n} \leq 10^{-6} &\iff \frac{1}{2^n} \leq \frac{1}{10^6} \\ &\iff 10^6 \leq 2^n \\ &\iff \log_2(10^6) \leq n,\end{aligned}$$

como $\log_2(10^6) = 19.9$, tenemos que el minimo que cumple es $n = 20$, por lo cual con x_{21} obtenemos la presición deseada, es decir, en la 21 iteración.

–

Ejercicio 3 (puntos)

Programar y probar el método de Newton-Raphson.

1. Escriba la función correspondiente a este algoritmo, la cual debe recibir como parámetros

- La función f a la que se le quiere calcular una raíz.
- La derivada de f
- El punto inicial x_0
- El valor de la tolerancia τ
- El máximo número N de iteraciones para el algoritmo

2. La información que debe devolver la función que implementa el algoritmo es:

- El último punto x_k que genera el algoritmo
- El valor $f(x_k)$
- El número k de iteraciones realizadas
- Una variable res que indica la condición en la que terminó el algoritmo: $res = 0$ si $|f(x_k)| < \tau$, $res = 1$ si el algoritmo termina porque se alcanzó el número de iteraciones.

3. Prueba la implementación del algoritmo usando las siguientes funciones, con N=1000 y una tolerancia $\tau = \sqrt{(}\epsilon_m)$, donde ϵ_m es el épsilon de la máquina.

- $f(x) = e^{2x} - x - 6$, con $x_0 = 3$.
- $f(x) = 3\cos(3\pi x) - 4x$, con $x_0 = -0.75$.
- $f(x) = \arctan x$, con $x_0 = 0.5$.
- $f(x) = \arctan x$, con $x_0 = 2$.

4. En cada caso haga imprima los datos:

$$x_0, \quad f(x_0), \quad k, \quad x_k, \quad f(x_k), \quad res$$

Solución:

```
# Esta celda es para escribir el código de la función o importarla de un archivo

def newthon_raphson(f, f_p, x0, err, N):
  xk = x0
  for i in range(N):
    if abs(f(xk)) < err:
      return xk, f(xk), i+1, 0
    fk_p = f_p(xk)
    if fk_p ≠ 0:
      xk = xk - f(xk)/fk_p
  return xk, f(xk), N, 1
```

```
# Pruebas de la función
import numpy as np
N = 1000
err = np.finfo(float).eps
err = np.sqrt(err)

```

$f(x) = e^{2x} - x - 6$, con x0 = 3

```
f = lambda x: np.exp(2*x) - x-6
f_p = lambda x: 2*np.exp(2*x)-1
```

```
x0 = 3
(xk, fxk, k, res) = newthon_raphson(f, f_p, x0, err, N)
print(f"x0 = {x0}, f(x0) = {f(x0)}, k = {k}, xk = {xk}, f(xk) = {fxk}, res = {res}")
```

```
x0 = 3, f(x0) = 394.4287934927351, k = 9, xk = 0.9708700208593961, f(xk) = 7.552745984185094e-09, res = 0
```

- $f(x) = 3\cos(3\pi x) - 4x$, con $x_0 = -0.75$.

```
f = lambda x: 3*np.cos(3*np.pi*x) - 4*x
f_p = lambda x: -9*np.pi*np.sin(3*np.pi*x) -4
x0 = -0.75
(xk, fxk, k, res) = newthon_raphson(f, f_p, x0, err, N)
print(f"x0 = {x0}, f(x0) = {f(x0)}, k = {k}, xk = {xk}, f(xk) = {fxk}, res = {res}")
```

```
x0 = -0.75, f(x0) = 5.121320343559643, k = 252, xk = -0.1945011025300539, f(xk) = 1.7494832249553838e-09, res = 0
```

- $f(x) = \arctan x$, con $x_0 = 0.5$

```
f = lambda x: np.arctan(x)
f_p = lambda x: 1/(1+x**2)
x0 = 0.5
(xk, fxk, k, res) = newthon_raphson(f, f_p, x0, err, N)
print(f"x0 = {x0}, f(x0) = {f(x0)}, k = {k}, xk = {xk}, f(xk) = {fxk}, res = {res}")
```

```
x0 = 0.5, f(x0) = 0.4636476090008061, k = 4, xk = -2.5131473616567257e-11, f(xk) = -2.5131473616567257e-11, res = 0
```

- $f(x) = \arctan x$, con $x_0 = 2$.

```
f = lambda x: np.arctan(x)
f_p = lambda x: 1/(1+x**2)
x0 = 2
(xk, fxk, k, res) = newthon_raphson(f, f_p, x0, err, N)
print(f"x0 = {x0}, f(x0) = {f(x0)}, k = {k}, xk = {xk}, f(xk) = {fxk}, res = {res}")
```

```
x0 = 2, f(x0) = 1.1071487177940904, k = 1000, xk = -6.9999433953175654e+168, f(xk) = -1.5707963267948966, res = 1
<ipython-input-8-76cca78a3562>:2: RuntimeWarning: overflow encountered in double_scalars
  f_p = lambda x: 1/(1+x**2)
```

✓ 0s completed at 1:06 AM