Activity 03 : *Root Finding*

Bisection Method & Newton Method

1. Use of `lambda` function :

a. Within your folder, create a new notebook and name it: ***TP03.ipynb***

b. Write, then execute the following cell code:

```python
import numpy as np
import matplotlib.pyplot as plt
```
Python

c. After that, within a new cell code, program a function which return the value for a given mathematical function $\boldsymbol{f}(\boldsymbol{x})$. Use the example of $\boldsymbol{f}(\boldsymbol{x}) = \mathbf{cos}(\boldsymbol{x}) - \boldsymbol{x}$

```python
def f(x):
    f = np.cos(x)-x
    return f
```
Python

d. Try you function on the interval $\left[0, \frac{\pi}{2}\right]$ by using the following cell code

```python
x = np.linspace(0,np.pi/2, 10)
for x in x:
    print(f'x = {x:.4f} --> f({x:.4f}) = {f(x):.5f}','\n')
```
Python

e. Now, let's define the mathematical function in more concise way, using built-in func. `Lambda`

```python
x = np.linspace(0,np.pi/2, 10)
g = lambda x: np.cos(x)-x
for i in x:
    print(f'x = {i:.4f} --> g({i:.4f}) = {g(i):.5f}')
```
Python

f. Now, try to find the rood of this function, by guessing each time the value of

```python
xr = 0.5 # try with different values from 0.5 with 0.05 step. Reduce the step
plt.plot(x,f, label='f(x) = cos(x)', c='b')
plt.plot(x,g, label='g(x) = x', c='r')
ax.vlines(x=xr, ymin=0, ymax=1.6, ls='--', lw=0.75, colors='k')
ax.hlines(y=xr, xmin=0, xmax=1.6, ls='--', lw=0.75, colors='k')
plt.legend();
```
Python

Activity 03 : *Root Finding*

Bisection Method & Newton Method

2. Bisection method :

a. Write the following code in one cell code, then execute it

```python
def my_bisection(f, a, b, tol):
    # check if a and b bound a root
    if np.sign(f(a)) == np.sign(f(b)):
        raise Exception(
         "The scalars a and b do not bound a root")
    # get midpoint
    m = (a + b)/2
    if np.abs(f(m)) < tol:
        # stopping condition, report m as root
        return m
    elif np.sign(f(a)) == np.sign(f(m)):
        # case where m is an improvement on a.
        # Make recursive call with a = m
        return my_bisection(f, m, b, tol)
    elif np.sign(f(b)) == np.sign(f(m)):
        # case where m is an improvement on b.
        # Make recursive call with b = m
        return my_bisection(f, a, m, tol)
```
Python

b. Try this method with different mathematical functions. Use the following code as a template:

```python
f = lambda x: x**2 - 2

r01 = my_bisection(f, 0, 2, 0.1)
print("r01 =", r01)
r001 = my_bisection(f, 0, 2, 0.01)
print("r001 =", r001)

print("f(r01) =", f(r01))
print("f(r001) =", f(r001))
```
Python

c. Verify the intermediate value theorem, by making the following test. Comment!

```python
my_bisection(f, 2, 4, 0.01)
```
Python

d. Use the bisection method to find the rood of the following functions on the corresponding interval:

$(i) f(x) = \cos(x) , D \equiv [0, \pi]$;
$(ii) g(x) = x^2 - 4x + 1, D \equiv [1,5]$;
$(iii) h(x) = x^3 - x, D \equiv [-2,0]$

Activity 03 : *Root Finding*

Bisection Method & Newton Method

3. Newton method :

a. Write the following code (Newton-Raphson method)

```python
def my_newton(f, df, x0, tol):
    # output is an estimation of the root of f
    # using the Newton Raphson method
    # recursive implementation
    if abs(f(x0)) < tol:
        return x0
    else:
        return my_newton(f, df, x0 - f(x0)/df(x0), tol)
```

Python

b. Define the following function and its derivative, then apply Newton method to find the approximative root. Use this template for other function as mentioned above in **$2.d**

```python
def f(x):
    y = np.log(x) + x
    return y
def df(x):
    y = 1.0 / x + 1.0
    return y
```

Python

c. Apply the Newton program to find the root

```python
xr = my_newton(f, df, 1., 1e-5)
print('The aproximate solution is: ', xr)
print('And the value f(x) is: ', f(xr))
x=np.linspace(0,10,100)[1:]
plt.plot(x, f(x), c='r')
plt.grid()
plt.vlines(x=xr, ymin=-2, ymax=12, colors='k', ls='--', lw=0.75);
```

Python