TP N° 1

Exercice 1

1 Import

Import

- `numpy` : Les fonctions mathématiques prédéfinies, et `linspace` pour créer les vecteur des coordonnées x de chaque fonctions.
- `matplotlib` : Dessine les graphes.
- `collections` : Crée une nouvelle structure avec `namedtuple`.

```
import numpy as np
import matplotlib.pyplot as plt
from collections import namedtuple
```

2 Définition Des Fonctions Mathématiques

```
def function_1(x):
    return x**6 - x -1;

def function_2(x):
    return 1 - 1/4 * np.cos(x)

def function_3(x):
    return np.cos(x) - np.exp(-x)

def function_4(x):
    return x**4 - 56.101*x**3 + 785.6561*x**2 - 72.7856*x + 0.0078
```

3 Génération des Vecteurs

```
x1 = np.linspace(-2, 2, 400)
x2 = np.linspace(-2*np.pi, 2*np.pi, 400)
x3 = np.linspace(-1, 2*np.pi, 400)
x4 = np.linspace(-0.1, 0.5, 400)

y1 = function_1(x1)
y2 = function_2(x2)
y3 = function_3(x3)
y4 = function_4(x4)
```

4 Nouvelle Structure root

root

On a créé une nouvelle structure en utilisant `namedtuple` root. Elle représente les racines pour $f(x) = 0$ et possède trois attributs :

- **position** : coordonnée x de la racine
- **a** : extrémité gauche de l'intervalle auquel la racine appartient
- **b** : extrémité droite de l'intervalle auquel la racine appartient

```
root = namedtuple("root", ["position","a", "b"])
```

5 Scatter

Scatter

- `scatter_single(rootElement, colorValue, marker)` : Dessine un point qui représente une racine pour $f(x)$. Elle vérifie si les extrémités de l'intervalle de la racine sont égales. Si oui, le label est : $\alpha = a$, sinon $\alpha \in [a, b]$. On utilise cette fonction si f admet une seule racine.
- `scatter_many(rootElement, colorValue, index, marker)` : Dessine un point qui représente une racine pour $f(x)$. Elle vérifie si les extrémités de l'intervalle de la racine sont égales. Si oui, le label est : $\alpha_{\text{index}} = a$, sinon $\alpha_{\text{index}} \in [a, b]$. On utilise cette fonction si f admet plusieurs racines.

Paramètres :

- **rootElement** : de type `root`
- **colorValue** : couleur du marqueur
- **index** : indice de la racine
- **marker** : style du marqueur

```
def scatter_single(rootElement,colorValue,marker):
    if rootElement.a == rootElement.b :
      plt.scatter(rootElement.position, 0, color = colorValue , marker = marker,label = fr"$\alpha = {rootElement.a}$")
    else:
      plt.scatter(rootElement.position, 0, color = colorValue , marker = marker,label = fr"$\alpha \in [{rootElement.a},{rootElement.b}]$")

def scatter_many(rootElement,colorValue,index,marker):
    if rootElement.a == rootElement.b :
      plt.scatter(rootElement.position, 0, color = colorValue , marker = marker,label = fr"$\alpha_{index} = {rootElement.a}$")
    else:
      plt.scatter(rootElement.position, 0, color = colorValue , marker = marker,
        label = fr"$\alpha_{index} \in [{rootElement.a},{rootElement.b}]$")
```

6 Draw

Draw

La fonction `draw()` dessine le graphe et les points des racines grâce à la liste des racines `rootList` et aux fonctions `scatter` que nous avons précédemment définies, Et retourne la valeur de l'indice incrémenté de 1 pour la mettre à jour.

Paramètres :

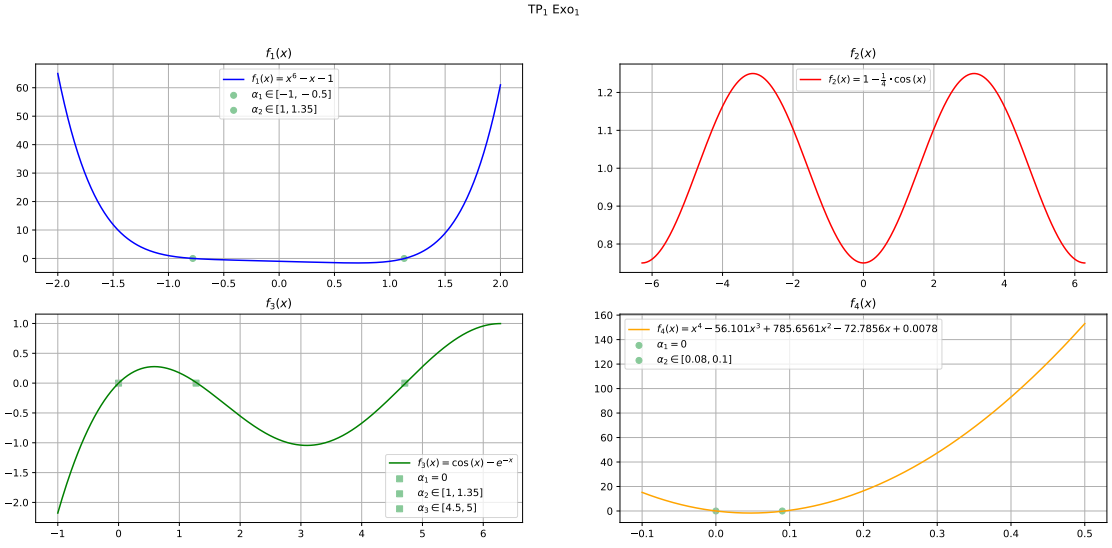
- **x** : Vecteur numpy des coordonnées x
- **y** : Vecteur numpy des coordonnées y
- **functionLabel** : Label de la fonction
- **colorValue** : Couleur du graphe
- **rootList** : Liste de racines de type `root`
- **index** : Indice du subplot
- **title** : Titre du graph
- **marker** (optionnel, valeur par défaut = `'o'`) : Style du marqueur pour les racines

```
def draw(x,y,functionLabel,colorValue,rootList,index,title,marker='o'):
    plt.subplot(2,2,index)
    plt.plot(x, y, label=fr"{functionLabel}", color=colorValue)
    sizeList = len(rootList)

    if sizeList == 1:
        scatter_single(rootList[0],'#88c999',marker)
    elif sizeList > 1 :
      for i in range(sizeList):
        scatter_many(rootList[i],'#88c999',i+1,marker)
    plt.legend()
    plt.grid()
    plt.title(title)
    return index+1
```

7 Reste Du Code

```

index = 1
index = draw(x1,y1,fr"$f_{index}(x) = x^6 - x - 1$","blue",[root(-0.78,-1,-0.5),root(1.13,1,1.35)],index,r"$f_1(x)$")
index = draw(x2, y2,fr"$f_{index}(x) = 1 - \frac{{1}}{{4}} \cdot \cos{{(x)}}$","red",[],index,r"$f_2(x)$",'.')
index = draw(x3, y3,fr"$f_3(x) = \cos{{(x)}} - e^{{-x}}$","green",[root(0,0,0),root(1.275,1,1.35),root(4.71,4.5,5)],index,r"$f_3(x)$",'s')
index = draw(x4, y4,fr"$f_4(x) = x^4 - 56.101x^{{3}} + 785.6561x^{{2}} - 72.7856x + 0.0078$","orange",[root(0,0,0),root(0.09,0.08,0.1)],
     index,r"$f_4(x)$",'o')

plt.suptitle(r"TP$_1$ Exo$_1$")
plt.savefig("fig.pdf")
plt.show()
```

8 Figure


TP1 Exo1
f1(x)
f1(x) = x^6 − x − 1
α1 ∈ [−1, −0.5]
α2 ∈ [1, 1.35]
f2(x)
f2(x) = 1 − 1/4 · cos(x)
f3(x)
f3(x) = cos(x) − e^−x
α1 = 0
α2 ∈ [1, 1.35]
α3 ∈ [4.5, 5]
f4(x)
f4(x) = x^4 − 56.101x^3 + 785.6561x^2 − 72.7856x + 0.0078
α1 = 0
α2 ∈ [0.08, 0.1]

Exercice 2

1 Import

Import

- `numpy` : Les fonctions mathématiques prédéfinies, et `linspace` pour créer les vecteur des coordonnées x de chaque fonctions.
- `matplotlib` : Dessine les graphes.
- `collections` : Crée une nouvelle structure avec `namedtuple`.

```
import numpy as np
import matplotlib.pyplot as plt
from collections import namedtuple
```

2 Définition Des Fonctions Mathématiques

```
def function_1(x):
    return np.log(x) - x + 2

def function_2(x):
    return np.exp(x) + x**2/2 + x - 1
```

3 ϵ

```
eps = 10 ** (-5)
```

4 Génération des Vecteurs

```
x_1 = np.linspace(0.1,5,400)
x_2 = np.linspace(-2,2,400)

y_1 = function_1(x_1)
y_2 = function_2(x_2)
```

5 Nouvelle Structure root

root

On a créé une nouvelle structure en utilisant `namedtuple` root. Elle représente les racines pour $f(x) = 0$ et possède trois attributs :

- **position** : coordonnée x de la racine
- **a** : extrémité gauche de l'intervalle auquel la racine appartient
- **b** : extrémité droite de l'intervalle auquel la racine appartient
- **error** : Estimation de l'erreur de la method dichotomie.
- **iteration**: Nombre d'iteration de la method dichotomie.

```
root = namedtuple("root", ["position","a", "b","error","iteration"])
```

6 Implementation De La Method Dichotomie

6.1 Estimation De L'erreur

```
def ErrorEstimation(a,b):
    return (b-a)/2
```

6.2 Algorithm De Dichotomie

```
def dichotomy(eps,a,b,function,max_iter=100):

    n = 1
    a_0 = a
    b_0 = b

    while (error := ErrorEstimation(a,b)) > eps and n <=max_iter :

        x = (a+b)/2

        if function(x) * function(a) < 0:
            b = x
        elif function(x) * function(b) < 0:
            a = x

        else:
            return root(x,a,n,"No Error",n)

        n = n+1

    return root((a+b)/2,a_0,b_0,f"{error:.2e}",n)
```

7 Dessiner Les Graph & Racines

7.1 Scatter

```python
def scatter_many(root,index,marker="*"):
    if root.error == "No Error":
        plt.scatter(root.position,0,marker=marker,color="#88c999",label=fr"$\alpha_{index} \in [{root.a},{root.b}] = {root.position}$")
    else:
        plt.scatter(root.position,0,marker=marker,color="#88c999",label=fr"$\alpha_{index} \in [{root.a},{root.b}] \approx {root.position}$"
            )


def scatter_single(root,marker="*"):
    if root.error=="No Error":
        plt.scatter(root.position,0,marker=marker,color="#88c999",label=fr"$\alpha \in [{root.a},{root.b}] = {root.position}$")
    else:
        plt.scatter(root.position,0,marker=marker,color="#88c999",label=fr"$\alpha \in [{root.a},{root.b}] \approx {root.position}$")

def scatter(rootList,marker="*"):
    size = len(rootList)

    if size == 1:
        scatter_single(rootList[0])
    else:
        for index in range(size):
            scatter_many(rootList[index],index)
```

7.2 Graphes

```python
def draw_graph(x,y,index,color,label,rootList,marker="*"):
    plt.subplot(2,2,index)
    plt.plot(x,y,color=color,label=fr"$f(x)_{index} = {label} $")
    plt.title(fr"$f_{index}(x)$")
    scatter(rootList)
    plt.legend()
    plt.grid()
    return index+1
```

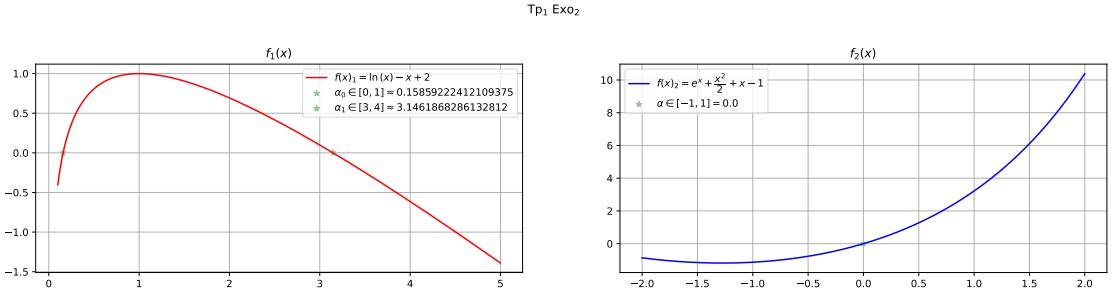
8 Tables

```python
def draw_table(rootList,index):
    data = [[r.position, r.a, r.b, r.error, r.iteration] for r in rootList]
    headers = ["Position", "a", "b", "Error", "Iteration"]
    plt.subplot(2,2,index)
    plt.axis("off")
    the_table = plt.table(cellText=data, colLabels=headers, loc="center", cellLoc="center")
    the_table.auto_set_column_width(col=list(range(len(headers))))
    the_table.scale(xscale=1, yscale=2)
    the_table.set_fontsize(12.5)
    return index+1
```

9 Rest Du Code

```
label_2 = r"e^{x} + \dfrac{x^2}{2} + x - 1"

color_1 = "red"
color_2 = "blue"

index = 1

rootList_1 = [dichotomy(eps,0,1,function_1)]
rootList_1.append(dichotomy(eps,3,4,function_1))

rootList_2 = [dichotomy(eps,-1,1,function_2)]

index = draw_graph(x_1,y_1,index,color_1,label_1,rootList_1)
index = draw_graph(x_2,y_2,index,color_2,label_2,rootList_2)
index = draw_table(rootList_1,index)
index = draw_table(rootList_2,index)

plt.suptitle(r"Tp$_{1}$ Exo$_{2}$")
plt.show()
```

10 Figure



Position	a	b	Error	Iteration
0.15859222412109375	0	1	7.63e-06	17
3.1461868286132812	3	4	7.63e-06	17

Position	a	b	Error	Iteration
0.0	-1	1	No Error	1