TP 7 - Régimes transitoires
Étude numérique d'un circuit (R, C) série

Objectifs :

- Comprendre et implémenter différentes méthodes pour résoudre l'équation $f(x) = 0$.
- Etudier l'influence du nombre d'itération sur le résultat et le temps de calcul.
- Comparer avec une fonction d'une bibliothèque.

Vous écrirez vos scripts Python dans les fichiers .py correspondants à chaque méthode. Vous pouvez écrire des commentaires et/ou des réponses aux questions dans le fichier .py lui même en faisant commencer la ligne commentée par un # . N'oubliez pas de sauvegarder les figures dans votre répertoire de travail.
A la fin du TP, placer tous vos fichier dans le drive `Physique>TP>TP7`

I Méthode de dichotomie

1. Quel est le principe de la méthode de dichotomie pour résoudre une équation $f(x) = 0$?
2. Ecrire la fonction `recherche_racine_dichotomie(f,inf,sup,eps)` avec f la fonction dont on veut connaître les racines (c'est à dire x tel que $f(x) = 0$) sur l'intervalle `[inf,sup]`, avec une précision de `eps`
3. Tester votre fonction `recherche_racine_dichotomie` sur la fonction suivante : $f(x) = (x-3)(x-10)$ sur [-5,5] à une précision de 10^{-6}.
4. Modifier votre fonction `recherche_racine_dichotomie` pour renvoyer le nombre d'iterations nécessaires pour converger.

II Méthode de Newton

1. Quel est le principe de la méthode de Newton pour résoudre une équation $f(x) = 0$?
2. Ecrire la fonction `recherche_racine_newton(f,df,x0,eps)` avec f la fonction dont on veut connaître les racines, *df* sa dérivée et $x0$ le point de départ, *eps* étant la précision voulue.
3. Tester votre fonction `recherche_racine_newton` sur la fonction suivante : $f(x) = (x-3)(x-10)$ à une précision de 10^{-6} avec un point de départ $x_0 = 5$. Comparer le résultat avec celui obtenu par la méthode de dichotomie
4. Modifier votre fonction `recherche_racine_newton` pour renvoyer le nombre d'iterations nécessaires pour converger.
5. Combien d'itérations sont nécessaires pour trouver la racine $x = 3$ de la fonction $f(x) = (x-3)(x-10)$ avec la méthode de Newton ? Avec la méthode de dichotomie ? Commentaires ?

III Influence de la précision sur le nombre d'itérations

1. Calculez le nombre d'iterations nécessaires pour trouver la racine $x = 3$ de la fonction $f(x) = (x-3)(x-10)$ avec la méthode de dichotomie à différentes précisions entre 10^{-1} et 10^{-8}.
2. Même question, avec la méthode de Newton
3. Tracez ces deux nombre d'itérations en fonction du logarithme de la précision.
4. En conclure sur quelle est la meilleur méthode dans ce cas particulier

IV Utiliser numpy et scipy

Les bibliothèques proposent des fonctions pour résoudre des équations $f(x) = 0$:

1 numpy

la fonction **numpy.roots** détermine les racines d'un polynôme donné par la liste de ses coefficients : Pour rechercher les solutions de $x^3 + 2x^2 - x - 2 = 0$ on a le code suivant :

```
import numpy
coeff=[1,3,-1,-2]
nympy.roots(coeff)
```

2 scipy

le module scipy proposent plusieurs fonctions pour résoudre les équations du type $f(x) = 0$ qui utilisent différentes méthode

- **scipy.optimize.bisect** (méthode de dichotomie) s'utilise avec 3 paramètres : scipy.optimize.bisect(f,a,b) avec f la fonction dont on cherche une racine, et a et b les bornes de l'intervalle de recherche. Par exemple :

```
import scipy.optmize
import math
scipy.optimize.bisect(math.sin,3,4)
```

renvoie le nombre π

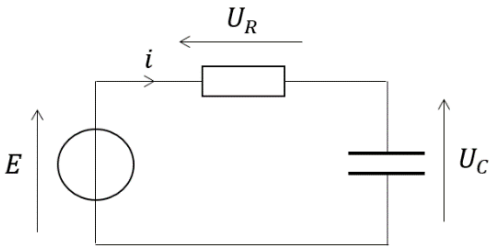
- **scipy.optimize.newton** (méthode de Newton) s'utilise aussi avec 3 paramètres : scipy.optimize.newton(f,x0,df) avec f la fonction dont on cherche une racine, df sa déricée et x0 le point de départ. Exemple :

```
import scipy.optmize
import math
scipy.optimize.newton(math.sin,3,math.cos)
```

1. Le paramètre exprimant la précision du résultat est nommé `tol` pour les deux fonctions scipy.optimize.bisect et scipy.optimize.newton et a une valeur par défaut. Utiliser la fonction `help` et donner cette valeur par défaut
2. Quelle est la syntaxe pour utiliser ces fonctions avec une autre précision que la valeur par défaut ?
3. Utiliser la fonction scipy.optimize.bisect pour déterminer les racines de la fonction $f(x) = (x-3)(x-10)$ à différentes précisions entre 10^{-1}, 10^{-3} 10^{-4} et 10^{-8}. Comparer avec les résultats obtenu avec votre fonction `recherche_racine_newton` aux mêmes précisions.
4. Même question avec la fonction scipy.optimize.newton, à comparer avec votre fonction `recherche_racine_newton`.

V Problème : temps de réponse d'un circuit RC

On étudie un circuit RC soumis à un échelon de tension E, le condensateur étant initialement déchargé. On cherche à évaluer numériquement les temps de réponse à 0.1%, 1% ou 5% et les comparer avec les valeurs couramment utilisées : 7τ, 5τ et 3τ



1. Montrer que la tension aux bornes du condensateur vaut : $U_C(t) = E(1 - e^{-\frac{t}{\tau}})$ en exprimant τ en fonction de R et C
2. Ecrire la fonction Python `tension_condensateur(t,E,R,C)` qui renvoie la tension aux bornes du condensateur en fonction du temps `t`, de la tension `E`, de la valeur de la résistance `R` et de la capacité `C`
3. Tracer la tension aux bornes du condensateur en fonction du temps, avec $E = 1V$, $R = 1M\Omega$ et $C = 1\mu F$
4. Pour connaître le temps de réponse à 0.1% de la tension E, quelle est l'équation à résoudre ? Même question pour le temps de réponse à 1% et 5%.
5. Utilisez votre fonction `recherche_racine_dichotomie` pour déterminer les temps de réponse à 0.1%, 1% et 5% avec une précision de 10^{-3} et tracer ces temps de réponse en fonction de la valeur de τ, sur 3 figures différentes
6. Utilisez votre fonction `recherche_racine_newton` pour déterminer les temps de réponse à 0.1%, 1% et 5% avec une précision de 10^{-3} et superposez ces temps de réponse sur les figures créées dans la question précédente.
7. Conclure vis-à-vis des valeurs communément utilisées pour les temps de réponse à 0.1%, 1% et 5%