P4 : problème stationnaire à une dimension

Résoudre f(x)=0

L'objectif de ce TP, une introduction aux méthodes numériques, est de maitriser le module `numpy` (qui permet de faire du calcul numérique), le module `matplotlib` (qui permet de tracer des courbes) ainsi que la recherche d'un zéro d'une fonction.

Vous trouverez en annexes quelques rappels permettant d'utiliser les bibliothèques `numpy` et `matplotlib` de Python. Taper celles-ci dans la console ou dans un script pour tester ce qu'elles font.

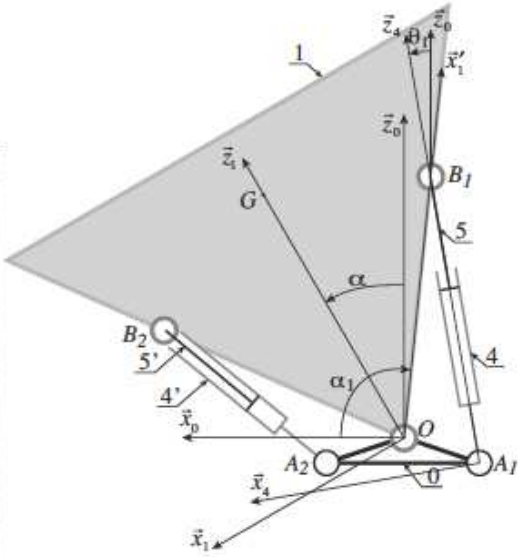
Commencez donc par importer les bibliothèques nécessaires :

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

CINEMATIQUE DU VEHICULE CLEVER

On s'intéresse au mouvement de la cabine du véhicule à trois roues Clever dont la cabine s'incline à l'image d'une moto pour prendre un virage. Une vidéo de ce véhicule est disponible ici:

https://www.youtube.com/watch?v=wLqR6hjkeYY



$\alpha = (\vec{z}_0, \vec{z}_1)$
$\alpha_1 = (\vec{x}_0, \vec{x}_1')$
$\theta_1 = (\vec{z}_0, \vec{z}_4)$
$\overrightarrow{OA_1} = -a\vec{x}_0 - b\vec{z}_0$
$\overrightarrow{A_1B_1} = \lambda_1\vec{z}_4$
$\overrightarrow{OB_1} = L\vec{x}_1'$

Pour piloter le mécanisme, il est nécessaire de connaître l'angle de la cabine en fonction de l'élongation des vérins. L'étude géométrique permet d'obtenir facilement l'élongation en fonction de l'angle :

$$\lambda_1(\alpha) = \sqrt{\left(L.\cos(\alpha - 130°) + a\right)^2 + \left(L.\sin(\alpha - 130°) - b\right)^2}$$

avec $\alpha \in [-50°, 50°]$, $a = 0.14m$, $b = 0.046m$ et $L = 0.49m$.

L'objectif du TP est de déterminer l'angle α pour une valeur d'élongation λ_1 donnée du vérin de commande. Cela revient donc à résoudre l'équation suivante du type $f(x) = 0$:

$$\sqrt{\left(L.\cos(\alpha - 130°) + a\right)^2 + \left(L.\sin(\alpha - 130°) - b\right)^2} - \lambda_1 = 0$$

1 - Approche graphique

Question 1 :

Créer une fonction `lambda1(alpha)` qui renvoie la valeur de λ_1 pour un angle α donné en degrés. Attention toutes les fonctions trigonométriques utilisent des **angles en radians**.

Pour tracer une courbe avec matplotlib, il suffit de créer 1 liste `X` pour les abscisses, 1 liste `Y` pour les ordonnées et de taper la suite de commande suivante :

```
plt.plot(X,Y)
plt.show()
```

`X` contient les valeurs de `alpha` entre -50° et +50° :

```
X=np.linspace(-50,50,100)
```

Question 2:

Créer une fonction `liste_lambda1(L)` qui renvoie la liste des valeurs de λ_1 pour les valeurs de `alpha` présentes dans la liste `L`.

Tester votre fonction en l'appelant pour la liste `X` :

```
print(liste_lambda1(X))
```

Question 3 :

Tracer l'élongation λ_1 en fonction de α dans le domaine d'étude considéré. Une centaine de points suffira.

Question 4 :

Déterminer graphiquement, en ajoutant une grille `plt.grid()`, l'angle α pour un allongement $\lambda_1 = 0.4\,m.$

3 – Différents algorithmes de résolution

Vous allez dans cette partie implanter différents algorithmes de résolution.

Ces algorithmes sont des algorithmes itératifs qu'il convient d'arrêter lorsque l'on atteint un critère. **Le critère d'arrêt retenu sera $|x_n - x_{n-1}| < 10^{-10}$ sauf indication contraire à modifier.**

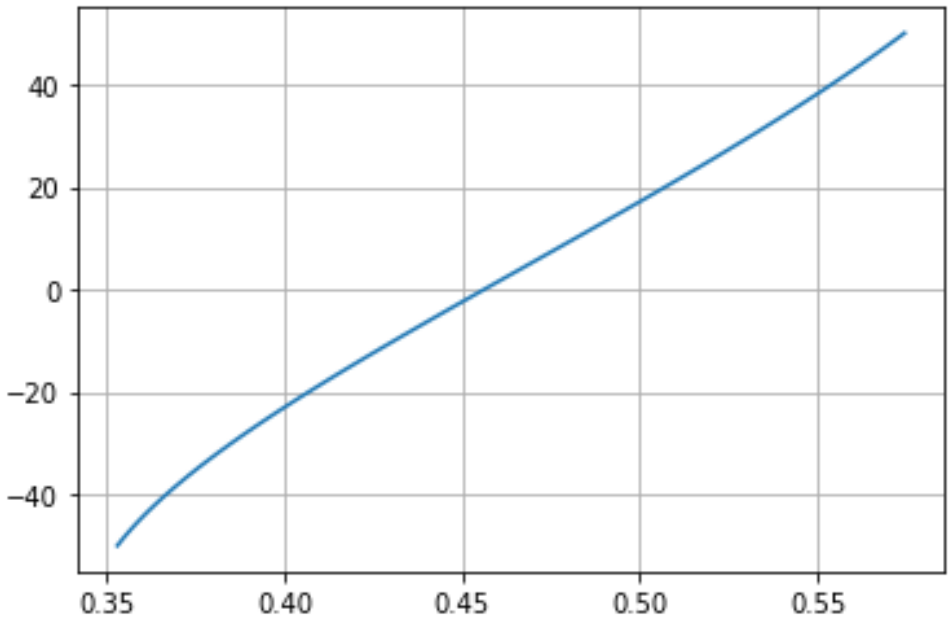
3.1 Algorithme par dichotomie

La méthode par dichotomie est d'approcher la solution par réduction successive de l'intervalle de recherche. Cette méthode est adaptée pour les fonctions monotones dont on recherche le zéro.

Entrée : p : précison ; a et b : valeurs encadrant grossièrement la solution cherchée (par lecture sur un tracé de courbe par exemple)

Traitement :

Si $f(a) \times f(b) > 0$ alors
    Afficher un message d'erreur (du type : vérifier les valeurs de a et b)
sinon
    Tant que $abs(a - b) > p$
        $c = \frac{a+b}{2}$
        Si $f(a) \times f(c) < 0$ alors
            $a, b \leftarrow a, c$ (Affectation multiple en python)
        Sinon
            $a, b \leftarrow c, b$
Afficher ('une valeur approchée de la solution est : ', $\frac{a+b}{2}$)



L'objectif du TP est de déterminer l'angle α pour une valeur d'élongation $\lambda_1 = 0.4\ m$ donnée du vérin de commande. Cela revient donc à résoudre l'équation suivante du type $f(x) = 0$:

$$\sqrt{\left(L.\cos(\alpha - 130°) + a\right)^2 + \left(L.\sin(\alpha - 130°) - b\right)^2} - 0.4 = 0$$

Question 5 :

Coder la fonction `f(alpha)` dont on recherche le zéro.

Question 6: Implanter l'algorithme par dichotomie dans une fonction `dichotomie(f,a,b)`.

Vérifier que le résultat renvoyé correspond à celui attendu en appelant votre fonction :

```
print(dichotomie(f,-50,50))
```

Question 7: Modifier votre fonction `dichotomie` pour afficher le nombre d'itérations.

Question 8: Modifier votre algorithme pour que celui-ci renvoie la liste contenant les solutions successives obtenues à chaque itération.

Question 9 : Tracer l'évolution de la solution de l'algorithme en fonction du nombre d'itérations.

On observera l'ordre de convergence : $ordre(i) = \frac{log(|x_i - x_{i-1}|)}{log(|x_{i-1} - x_{i-2}|)}$.

Plus l'ordre est élevé, plus l'algorithme converge rapidement vers la solution.

Question 10: Ecrire une fonction `ordre(liste_x)` qui renvoie la liste contenant l'ordre de convergence en fonction des itérations. En déduire l'ordre de convergence de la méthode par dichotomie.

Nous cherchons maintenant à mesurer le temps de recherche de la solution, pour cela nous utilisons les commandes suivantes :

```
import time
start=time.time_ns()      /// on lance le chrono
/// on fait quelque chose
stop=time.time_ns()       /// on arrete le chrono
duree=stop-start
print(duree) //// on affiche la durée en nanosecondes
```

Question 11 : Déterminer le temps mis par l'algorithme pour trouver la solution.

3 – Utilisation de la bibliothèque Scipy

Un certain nombre d'algorithmes existent pour résoudre une équation, la plupart d'entre eux sont déjà implantées dans le module Scipy.
Nous allons utiliser la méthode de Newton pour déterminer une solution de référence. Cette méthode permet de résoudre une équation mise sous la forme $f(x)=0$. Pour utiliser la fonction de résolution avec l'algorithme de Newton, il faut taper les commandes :

```
import scipy.optimize as opt
opt.newton (f,-50)        #la valeur de départ de la recherche est -50
```

Question 12 : Déterminer numériquement l'angle α pour un allongement $\lambda_1 = 0.4\,m$. Comparer la valeur obtenue en fonction de la valeur initiale.

Question 13 : Déterminer le temps mis par l'algorithme pour trouver la solution et comparer le à la méthode par dichotomie.

3.2 Algorithme de Newton (bonus)

Le principe de la méthode de Newton est de chercher le zéro d'une fonction en prenant comme nouvelle approximation l'abscisse du point d'intersection de la tangente à la fonction au point d'approximation précédente. Ici on peut calculer la dérivée exacte de la fonction :

$$\lambda_1'(\alpha) = \frac{-aL\sin(\alpha-130°) - bL\cos(\alpha-130°)}{\sqrt{\left(L.\cos(\alpha-130°)+a\right)^2+\left(L.\sin(\alpha-130°)-b\right)^2}}$$

On pourra aussi utiliser la valeur approchée de la dérivée de la fonction en utilisant un taux d'accroissement: $f'(x_k) = \frac{f(x_k+h)-f(x_k)}{h}$.

Question 14: Ecrire une fonction `newton(f,xini)` qui affiche la solution, le nombre d'itérations et qui renvoie la liste des approximations $x_k$successives en prenant la dérivée exacte.

Question 15: Ecrire une fonction `newton2(f,xini)` qui affiche la solution, le nombre d'itérations et qui renvoie la liste des approximations $x_k$successives en prenant la dérivée approchée pour $h = 0.1m$.

Question 16: Déterminer l'évolution de l'ordre de convergence en fonction des itérations et en déduire l'ordrede convergence pour différentes valeurs de h.

Un camarade (je ne dirais pas qui) initialise son algorithme avec la valeur −20 et obtient un résultat aberrant !

Question 17: Tester votre algorithme avec cette valeur et expliquer ce qu'il se passe.

4 – Conclusion

Question 18: Comparer les différentes méthodes mises en œuvre en terme de nombres d'itérations pour converger, l'ordre de convergence. Conclure.

ANNEXE 1: BIBLIOTHÈQUE NUMPY

Numpy est un module de calcul numérique qui permet de réaliser des opérations complexes via unmode non interprété sur des données de type vecteur, matrice, tableau multidimensionnel.Pour charger le module, il faut commencer par taper :

```
importnumpy as np
```

Contrairement au mode standard de Python, toutes les variables sont stockés en mémoire en suivant les conventions du C, on retrouve ainsi les codages des entiers non signés (uint16, uint32, uint64), les entiers signés (int8, int16, int32, int64), les flottants (float16, float32, float64).

Les éléments créés par numpy sont tous codés suivant un de ses types, accessible par l'attribut dtype.Il faut bien garder à l'esprit que la représentation des données en mémoire est limité, par exemple taper dans la console :

```
np.float16 (0.3)
np.float32 (0.3)
np.float64 (0.3)
np.float16 (0.3) == 0.3
np.float32 (0.3) == 0.3
np.float64 (0.3) == 0.3
```

La comparaison de manière exacte entre deux valeurs numériques ne donne pas forcément le résultat attendu. C'est la raison pour laquelle en calcul numérique, la comparaison à 0 se fait toujours à un ε près.Il existe de nombreuses commandes pour créer des vecteurs avec numpy, essayer :

```
a=np.array([0,1,2,3])                    # construction d'un vecteur
a.dtype
a=np.array([0,1,2,3.0])
a.dtype # python determiner le type automatiquement
a=np.array([0,1,2,3.0],np.float16) # mais on peut aussi imposer le type
a.dtype
a.shape # donne la dimension du vecteur
b=a.reshape(2,2) # permet de transformer le vecteur en matrice
```

On peut aussi créer des éléments remplies de 0 ou de 1, utile pour initialiser les variables :

```
np.zeros(5)
np.zeros((4 ,3))
np.ones((2 ,3))
```

On peut aussi créer des vecteurs par un objet de type range :

```
np.arange(0,1,0.1)             # arange(ini,fin,pas)
np.linspace(0,1,101)         # linspace(ini,fin,nbval)
```

Numpy définit la plupart des opérations mathématiques sous forme vectorielle, par exemple pour le sinus :

```
t=np.arange(0,1,0.01)     # intervalle de temps
T=0.25                    # periode du sinus
f=np.sin(2*np.pi/T*t)     # calcul de la fonction pour toutes les valeurs
```

ANNEXE 2: BIBLIOTHÈQUE MATPLOTLIB

Matplotlib est un module qui permet de tracer des courbes comme sur les logiciels de calcul scientifique de type Matlab/Scilab. Pour charger le module, il faut commencer par taper :

```
import matplotlib . pyplot as plt
f2=np.sin(np.pi/T*t)                                    #une autre courbe
plt.plot(t,f)                              # permet de tracer la courbe f
plt.plot(t,f2)                            # permet de tracer la courbe f2

# mettre des titres aux axes , figure , legende est obligatoire
plt.xlabel('Temps (s)')
plt.ylabel('f(t)')
plt.title('Trace de la courbe sin en fonction du temps ')
plt.legend(('f1','f2'))
plt.show ()
```

Si vous voulez réaliser plusieurs figures alors il faudra utiliser explicitement la création de celles-ci :

```
fig1=plt.figure()
fig11=fig1.add_subplot(1,2,1)
            # add_subplot (l, c, s) permet de creer une zone avec
            #l ligne , c colonne et on selectionne la case s
            # puis selectionner la 1ere
fig11.plot(t,f)
plt.xlabel('Temps (s)')
plt.ylabel('f(t)')
fig12=fig1.add_subplot(1 ,2 ,2) #on selectionne la seconde
fig12.plot(t,f)
plt.xlabel('Temps (s)')
plt.ylabel('f2(t)')
fig1.suptitle('Trace de deux courbes differentes : fig1 ')
#2eme figure on superpose les courbes
fig2=plt.figure()
fig21=fig2.add_subplot(1,1,1) # creer une figure avec une seule zone de trace
fig21.plot(t,f) # permet de tracer la courbe f
fig21.plot(t,f2) # permet de tracer la courbe f2
plt.xlabel('Temps (s)')
pltylabel('f(t)')
plt.title('Trace de la courbe sin en fonction du temps : fig2 ')
plt.legend(('f1','f2'))
fig1.savefig('fig1 .png ') # sauvegarde des figures dans un fichier
fig2.savefig ('fig2 .png ') # sauvegarde des figures dans un fichier
plt.show()
```