I Equations différentielles

1.1 Rappels

La méthode d'Euler pour la résolution approchée d'équations différentielles du type $y'(t) = f(y(t), t)$ est la suivante : (on note y la solution vérifiant $y(t_0) = y_0$)

- on découpe l'intervalle $[a, b]$ de résolution en n parties et on pose $h = \frac{b-a}{n}$.
- les valeurs de $y(t+kh)$ sont notées y_k et définies par $y_{k+1} = y_k + h \times f(y_k, t+kh)$.

Ceci représente l'approximation de la courbe par des tangentes successives (on crée une fonction affine par morceaux). Cette formule est aussi valable quand y est à valeurs vectorielles.

1.2 Application

On cherche la trajectoire $\mathrm{M}(t)$ d'un point (boulet de canon par exemple) de vitesse initiale $\vec{v_0} = v_0 \begin{pmatrix}\cos(\alpha)\\ \sin(\alpha)\end{pmatrix}$, position initiale $\begin{pmatrix}0\\0\end{pmatrix}$ et de masse m. La résistance de l'air est modélisée par une force de la forme $-kv\vec{v}$ où $k > 0$ est une constante.

1. Ecrire l'équation différentielle vectorielle correspondante.
2. Implémenter la méthode d'Euler pour obtenir les valeurs successives du vecteur vitesse. On pourra créer une fonction `vitesse(V0, tf, n)` où tf est le temps final (on résout sur $[0, tf]$, $n+1$ le nombre le valeurs demandées et V0 le vecteur vitesse initial.

   La valeur de retour devra être T, V où T est la liste des temps et V la liste des valeurs de du vecteur vitesse.
3. Tester votre fonction avec :

```
T, V = trajectoire(v0, tf, n)
Vhoriz = [v[0] for v in V]
Vvert = [v[1] for v in V]
plt.plot(T, Vhoriz)
plt.plot(T, Vvert)
```

4. Comparer les résultats obtenus avec ceux de la fonction `scipy.integrate.odeint`
5. Créer la fonction `trajectoire(T, V)` qui prend la liste des temps et la liste des valeurs du vecteur vitesse précédemment calculées comme paramètre et trace la trajectoire de notre mobile. On utilisera la même approximation que pour la méthode d'Euler

   Observer les trajectoires pour différentes valeurs de m, α. On pourra prendre $k = 0.1$.

1.3 Intégrer

La méthode pour retrouver la trajectoire précédemment utilisée est équivalente à intégrer pas à pas le vecteur vitesse en utilisant la méthode des rectangles pour calculer l'aire sous les courbes. Si on note $\mathrm{M}_0, \ldots, \mathrm{M}_n$ les positions successives de notre mobile et $\mathrm{T} = [t_0, \ldots, t_n]$ la liste des temps où on a évalué le vecteur vitesse, on a en fait utilisé la formule

$$\mathrm{M}_{i+1} = \mathrm{M}_i + \int_{t_i}^{t_{i+1}} \vec{v}(t)\mathrm{d}t$$

en approximant l'intégrale par $(t_{i+1} - t_i)\vec{v}(t_i)$.
On souhaite maintenant utiliser la méthode des trapèzes pour améliorer la précision de nos résultats. Implémenter ceci dans la fonction `traj2(T, V)`. Observer la différence entre les courbes obtenues pour des valeurs de n de $50, 100, 500$.

II Résolution approchée d'équations, dérivation

2.1 Méthode de Newton

Retrouver rapidement la relation de récurrence traduisant la méthode de Newton. Rappel : il s'agit de trouver des points d'intersection successifs entre les tangentes à la courbe de f et l'axe des abscisses. On part de $x_0 = a$ une abscisse fixée et on cherche x_{k+1} en fonction de x_k, le prochain point d'intersection.

```
def newton(f, fprime, a, nbiter=15):
        x = a
        for i in range(nbiter):
                x =  # à compléter
        return x
```

2.2 Problème

Imaginons que nous ne connaissions f que par ses valeurs (pas d'expression, valeurs données dans une liste L) et donc que f' soit inaccessible. Il va nous falloir approximer f'. Nos données sont T, L les listes $[t_0, \ldots, t_n]$ et $[f(t_0), \ldots, f(t_n)]$.
Proposer une méthode qui, étant donné f et i, retourne une valeur approchée de $f'(t_i)$ (au temps t_i, c'est à dire pour la valeur d'indice i dans la liste L) et l'implémenter en python. Evaluer l'efficacité de la méthode en utilisant une fonction fixée (dont vous connaissez la dérivée) et en comparant la valeur exacte et la qualité de l'approximation. On pourra faire varier la longueur de T et utiliser des fonctions numpy pour obtenir L directement.

2.2.1 Meilleure approche

En utilisant Taylor-Young (préciser les hypothèses à vérifier sur f), donner un développement limité à l'ordre 1 de $\frac{f(a+h)-f(a)}{h}$ et un de $\frac{f(a+h)-f(a-h)}{2h}$ (valable quand $h \to 0$).

En déduire une meilleure méthode, l'implémenter en python et comparer la qualité de l'approximation avec la valeur de h. Que se passe-t-il pour $i = 0$ ou n ?

2.2.2 Pour aller plus loin

Créer une fonction `derive (f)` qui prend une **fonction python** f comme argument et renvoie sa **fonction** dérivée. En particulier il faudra créer une fonction "locale" en tant que valeur à retourner. On utilisera $f'(a) \approx \frac{f(a+h)-f(a-h)}{2h}$ pour une valeur de h "petite".
Nous sommes maintenant en mesure de calculer le développement limité (il nous manque juste une petite fonction, que nous avons déjà rencontré plusieurs fois...) de la fonction f à tout ordre.
Ecrire la fonction `dl(f, a, n)` qui retourne la liste des coefficients du développement limité de f en a à l'ordre n. Comparer à des développements connus.