TP 9 : Algorithmes numériques, utilisation de modules.

Dans ce TP, on présente quelques algorithmes numériques classiques : résolution d'équations numériques, calculs d'intégrale, résolution d'équations différentielles, et on (re)voit l'utilisation de modules en Python qui sont notamment utilisés en physique. Veuillez télécharger le squelette du TP sur le site web, il n'y a qu'à compléter les fonctions !

On travaille ici dans le monde flottant : les résolutions effectuées seront approchées. Notamment, si on demande de vérifier expérimentalement que $a = b$, cela signifie visualiser que a et b sont très proches, mais pas que `a==b`, qui s'évaluera souvent en `False` !

1 Résolution approchée d'une équation de la forme $f(x) = 0$

1.1 Méthode dichotomique

On se donne une fonction $f : [a, b] \to \mathbb{R}$ continue, **vérifiant** $f(a)f(b) \leq 0$. Le fait que f change de signe entre les deux extrémités du segment assure par théorème des valeurs intermédiaires que f possède au moins un zéro sur l'intervalle $[a, b]$. Le but est de trouver une approximation d'un tel zéro à ε près. On procède ainsi :

— on démarre avec $g = a$ et $d = b$;

— à chaque étape, on calcule le milieu m de l'intervalle $[g, d]$, et on fait prendre cette valeur à g ou à d, pour maintenir *l'invariant de boucle* $f(g)f(d) \leq 0$ (qui assure que f possède un zéro dans l'intervalle $[g, d]$).

— on s'arrête lorsque $d - g \leq 2\varepsilon$: le milieu est alors une approximation à ε près d'un zéro de f.

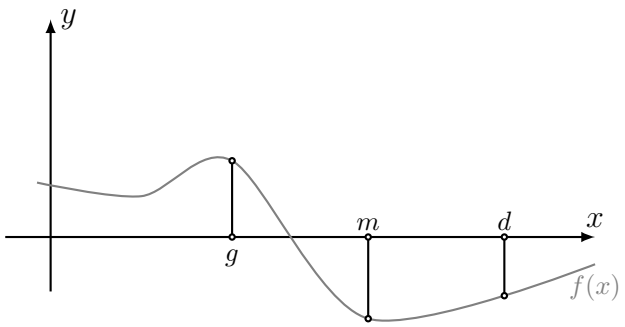

FIGURE 1: Méthode de la dichotomie : ici $f(g)f(m) \leq 0$, donc c'est d qui prend la valeur m.

Exercice 1. Compléter la fonction `dicho(f,a,b,eps)` prenant en entrée la fonction `f`, les bornes a et b et le réel ε, et renvoyant un zéro à ε près de f sur $[a, b]$.
Remarques :

— attention à utiliser la division flottante (`/`), et pas entière (`//`) dans vos calculs de milieux.

— le code démarre avec l'instruction `assert f(a)*f(b)<=0` : on vérifie que $f(a)f(b) \leq 0$ en début de fonction, si ceci n'est pas vérifié la fonction s'arrête avec une erreur.

Tester ensuite votre code avec la fonction `test_dicho` (sans argument) fournie. On pourra afficher à chaque étape les valeurs prises par m dans le code de `dicho`. Vérifiez que l'on gagne un chiffre correct sur le résultat tous les 3 ou 4 étapes (en effet, on gagne un bit correct à chaque étape car l'intervalle de recherche est divisé par 2, et $2^3 < 10 < 2^4$).

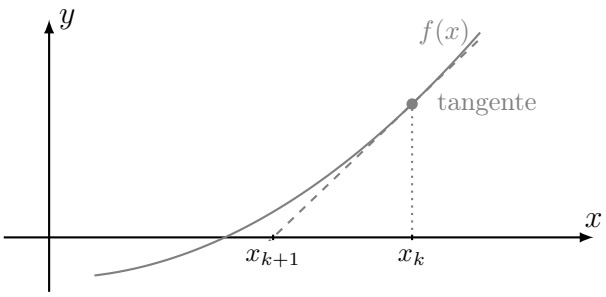

FIGURE 2: Une itération dans la méthode de Newton.

1.2 Méthode de Newton

La méthode de Newton pour le caclul d'une solution de $f(x) = 0$ nécessite que f soit dérivable. On part simplement d'une abscisse x_0, et à chaque étape x_{k+1} est l'intersection de la tangente à f au point $(x_k, f(x_k))$ avec l'axe des abscisses (voir figure 2). On espère que la suite (x_k) ainsi définie converge vers un zéro de f.

Exercice 2. Quelle est l'équation de la tangente à f en x_k ? Vérifier que si $f'(x_k)$ est non nul, cette tangente intersecte bien l'axe des abscisses, et donner l'expression de son intersection x_{k+1} avec l'axe des abscisses en fonction de x_k.

Exercice 3. Compléter la fonction `newton(f,g,x0,N)` réalisant N itérations de la méthode de Newton pour la fonction f à partir du point x_0. La fonction g est la dérivée de f, votre fonction renverra la dernière abscisse calculée. Tester avec `test_newton` (sans argument). En faisant usage d'un `print` dans votre fonction `newton`, vérifier que le nombre de chiffres corrects *double* à chaque étape.

Exercice 4. *Non convergence*. On cherche à résoudre $x^3 + cx + 1 = 0$, avec c un réel. Compléter la fonction `test_polynome(c)` appelant la méthode de Newton avec f la fonction $x \mapsto x^3 + cx + 1$ avec 100 itérations (laisser `print` dans votre fonction `newton` pour voir les itérations).

1. Vérifier que pour $c > 0$, les itérés convergent (on prendra $x_0 = 0$)
2. Vérifier que pour $c = -1.286$ et $x_0 = 0$ par exemple, la méthode n'a pas l'air de converger.

Remarque : la méthode dichotomique est « lente, mais sûre » : du moment qu'on a isolé un intervalle sur lequel f change de signe, la méthode trouvera une approximation d'un zéro. La méthode de Newton, lorsqu'elle fonctionne, est beaucoup plus rapide. Néanmoins, elle ne converge pas tout le temps !

1.3 Utilisation du module Scipy

Le module `scipy` possède un sous module `scipy.optimize` (importé comme `sco` dans le squelette) dédié à l'optimisation. Dans ce module on trouve :

— la fonction `bisect` : c'est la méthode dichotomique. Elle s'utilise sous la forme `sco.bisect(f, a, b)` comme ci-dessus (on peut ajouter une précision optionnelle).

— la fonction `newton` : c'est la méthode de Newton. Elle s'utilise sous la forme `sco.newton(f, x0, fprime=...)`, où `fprime` est un argument optionnel pour la dérivée [1].

— la fonction `fsolve` : une généralisation de la méthode de Newton. Peut-être l'avez vous déja utilisée en physique.

Exercice 5. Tester les fonctions `bisect` et `newton` !

2 Calcul approché d'intégrales

2.1 Méthode des rectangles à gauche

On souhaite calculer de manière approchée l'intégrale d'une fonction $\int_a^b f(t)\,\mathrm{d}t$. Pour ce faire, on peut découper l'intervalle en n de morceaux de longueur $h = \frac{b-a}{n}$. Sur un petit intervalle $[a+kh, a+(k+1)h]$, on approche simplement l'intégrale comme suit : $\int_{a+kh}^{a+(k+1)h} f(t)\,\mathrm{d}t \simeq h \cdot f(a+kh)$, voir figure 3.

1. Si cet argument n'est pas donné, c'est la méthode de la sécante qui est utilisée.

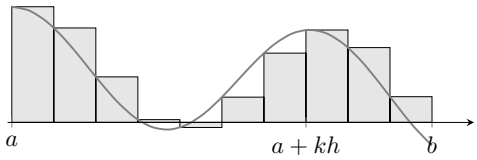

FIGURE 3: Méthode des rectangles à gauche

Exercice 6. Compléter la fonction `int_rec_g(f, a, b, n)` prenant en entrée une fonction f définie sur un intervalle $[a, b]$, et calculant de manière approchée l'intégrale $\int_a^b f(t)\,\mathrm{d}t$ en suivant la méthode ci-dessus. Tester avec la fonction `test_integrale` (sans argument).

2.2 Utilisation du module Scipy

Le module `scipy` possède un sous-module `scipy.integrate` (importé comme `sci` dans le squelette) dédié au calcul d'intégrales et à la résolution d'équations différentielles. Dans ce sous-module on trouve notamment la fonction `quad` qui permet le calcul d'intégrale. Elle s'utilise sous la forme `sci.quad(f, a, b)` pour calculer l'intégrale $\int_a^b f(t)\,\mathrm{d}t$. Elle renvoie un couple (v, e) où v est la valeur approchée de l'intégrale, et e le terme d'erreur.

Exercice 7. Tester la fonction `quad`.

Exercice 8. *Utilisation de `quad`.*
1. Les bornes prises par la fonction `quad` peuvent être infinies (on utilisera `float('inf')` pour créer $+\infty$). Vérifier expérimentalement que $\int_{-\infty}^{+\infty} \mathrm{e}^{-t^2}\,\mathrm{d}t = \sqrt{\pi}$.
2. Que donne[2] le calcul de $\int_1^{+\infty} \frac{\sin(t)}{t}\,\mathrm{d}t$?

3 Résolution approchée d'équations différentielles

On considère un problème différentiel sous la forme suivante :

$$\begin{cases} y'(t) &= f(y(t), t) \\ y(t_0) &= y_0 \end{cases}$$

où la fonction inconnue y cherchée est définie au voisinage de $t_0 \in \mathbb{R}$ et à valeurs dans $\mathbb{R}^k$ (ainsi f est définie sur un sous ensemble de $\mathbb{R}^k \times \mathbb{R}$ et $y_0 \in \mathbb{R}^k$). Nous allons coder une méthode pour résoudre une telle équation, puis examiner les possibilités offertes par le module `scipy`. Dans un premier temps, on aura $k = 1$: il s'agit d'une équation différentielle « classique » d'ordre 1.

3.1 Méthode d'Euler explicite et résolution d'un système RC

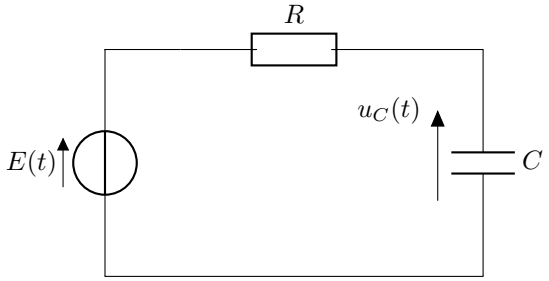

FIGURE 4: Circuit RC - Charge d'un condensateur

On rappelle que l'équation vérifiée par la tension u aux bornes du condensateur satisfait l'équation

$$\mathrm{RC}\frac{\mathrm{d}\,u}{\mathrm{d}t}(t) + u(t) = E(t)$$

2. On peut montrer que $\lim_{x\to+\infty} \int_1^x \frac{\sin(t)}{t}\,\mathrm{d}t$ existe et est finie. Par contre, comme $\lim_{x\to+\infty} \int_1^x \frac{|\sin(t)|}{t}\,\mathrm{d}t = +\infty$, le calcul numérique n'est pas aisé, et Python donne une réponse erronée.

où E est la force électromotrice du générateur, supposée connue. En notant u_0 la tension au temps $t_0 = 0$, on veut donc résoudre :

$$\left\{ \begin{array}{rcl} u'(t) & = & f(u(t),t) \\ u(t_0) & = & u_0 \end{array} \right. \qquad \text{avec} \qquad f(y,t) = \frac{1}{\mathrm{RC}}(E(t) - y)$$

qui est bien de la forme précédente.

La méthode d'Euler explicite pour la résolution d'une telle équation calcule de proche en proche des approximations de y aux temps t_0, $t_0 + h$, $t_0 + 2h, \ldots$, avec h un petit pas de temps. Pour cela, on utilise l'approximation

$$y(t+h) \simeq y(t) + hy'(t) = y(t) + hf(y(t),t)$$

D'où le schéma numérique :

$$y_0 \text{ donné} ; \qquad y_i = y_{i-1} + (t_i - t_{i-1}) f(y_{i-1}, t_{i-1}) \text{ pour } i \geq 1$$

Exercice 9. *Réécriture de la méthode d'Euler*. Écrire une fonction `euler_explicite(f,x0,T)` qui prend comme arguments une fonction `f`, une valeur intiale `x0`, et un tableau de temps $(t_i)_{0 \leq i < n}$, et renvoyant une liste d'approximations $(x_i)_{0 \leq i < n}$ obtenue par la méthode d'Euler explicite à l'aide du schéma précédent. Évidemment $x_i \simeq x(t_i)$ est calculé pour $i \geq 1$ à partir de x_{i-1}.

Dans la suite, on aura $t_0 = 0$, et $x_0 = 0$: on ferme l'interrupteur au temps $t = 0$ et le condensateur est initialement déchargé. Dans l'annexe, on a posé Res $= 500\Omega$, Cap $= 10^{-6}$F et tau $=$ Res $\cdot$ Cap. On définit plusieurs tensions E_1, E_2 et E_3 (tension constante, tension en créneaux, tension en triangles), et on a défini informatiquement trois fonctions `fRC1`, `fRC2`, `fRC3` égales à la fonction f dans chacun des cas.

Exercice 10. *Comparaison avec la solution exacte*. Dans le cas de la tension E_1 constante, la solution exacte de l'équation différentielle est : $u(t) = E_1(1 - \exp^{-t/\tau})$, avec τ ayant pour nom tau dans l'annexe. Tracer sur un même graphique :

— la solution exacte pour une f.e.m E_1 constante, sur l'intervalle $[0, 5\tau]$ (on pourra définir un tableau Numpy de temps relativement proches, par exemple 1000 points dans l'intervalle, via `np.linspace`).

— l'approximation donnée par la méthode d'Euler explicite, pour un pas de temps variable (on pourra utiliser `np.linspace` également, en prenant par exemple 10, 20 ou 100 points).

Rappel : `plt.plot(X,Y)` avec `X` et `Y` de même taille, contenant des réels $(x_i)_{0 \leq i < n}$ et $(y_i)_{0 \leq i < n}$, trace la courbe obtenue en reliant les points (x_i, y_i) par des segments. `plt.show()` permet d'afficher un graphique. Pour superposer plusieurs courbes, faire plusieurs `plt.plot` avant d'appeler `plt.show()`.

3.2 Utilisation du module Scipy

Le sous-module `scipy.integrate` contient la fonction `odeint` qui permet la résolution d'équations différentielles. Elle s'utilise exactement comme notre fonction `euler_explicite` (c'est pourquoi on l'a écrite ainsi). Les algorithmes sous-jacents sont bien plus évolués que la méthode d'Euler explicite.

Exercice 11. Reprendre la question précédente avec `fRC2` et `fRC3`, en remplaçant la solution exacte par la solution fournie par `odeint` (qui s'utilise comme notre fonction `euler_explicite`), sur l'intervalle $[0, 30\tau]$. Vérifier qu'avec 10 ou 20 points, la méthode d'Euler explicite est assez mauvaise. Vérifier tout de même qu'avec un grand nombre de points (1000 par exemple), la solution obtenue avec la méthode d'Euler converge vers la solution obtenue par `odeint` (qu'on peut considérer exacte).

4 Pour aller plus loin...

4.1 Méthode des trapèzes. Lecture d'un fichier.

Exercice 12. On se donne un fichier représentant une série de mesures (à récupérer sur le site web). Les lignes du fichier sont de la forme `x;y` où `x` et `y` sont des flottants. On suppose que ces lignes sont le résultat de mesures, c'est à dire qu'il existe une fonction f telle que $y_i = f(x_i)$ pour chaque ligne $x_i; y_i$. Les abscisses x_i sont supposées ordonnées de façon croissante, par contre elles ne sont pas supposées ordonnées régulièrement : la distance $x_{i+1} - x_i$ n'est pas constante. En supposant que le fichier a ℓ lignes ($x_0; y_0$ jusqu'à $x_{\ell-1}; y_{\ell-1}$), on souhaite approcher $\int_{x_0}^{x_{\ell-1}} f(t)\,\mathrm{d}t$, en utilisant la méthode des trapèzes, qui est plus précise que la méthode des rectangles à gauche. Voici la formule :

$$\int_{x_0}^{x_{\ell-1}} f(t)\,\mathrm{d}t \simeq \sum_{k=0}^{\ell-2}(x_{k+1}-x_k)\frac{f(x_k)+f(x_{k+1})}{2} = \sum_{k=0}^{\ell-2}(x_{k+1}-x_k)\frac{y_k+y_{k+1}}{2}$$

Écrire une fonction `integrale(fichier)` permettant d'automatiser le procédé, la fonction `integrale` prenant en entrée un fichier ouvert en lecture. On pourra par exemple utiliser `readlines`, `split` pour scinder une chaîne de caractères autour d'un élément (par exemple, `chaine.split(';')` fournit une liste correspondant à la séparation de la chaîne de caractères `chaine` autour de `;`), et on n'oubliera pas `float` permettant de convertir une chaîne de caractères en flottant.

J'obtiens ceci :

```
>>> f1=open("fichier1.txt",'r') ; integrale(f1)
464.5123984199802
>>> f2=open("fichier2.txt",'r') ; integrale(f2)
0.20764416913028605
```

Attention : pour ouvrir le fichier il faut préciser le chemin absolu, peut-être `"U:/Downloads/..."` au lycée.

4.2 Une autre approche du calcul d'intégrale : la méthode de Monte-Carlo

Cette méthode de calcul approché est basée sur les probabilités : on importera le module `random` de Python. On utilisera la fonction `random()` qui donne un nombre réel compris entre 0 et 1, en suivant une loi uniforme.
On se donne une fonction $f : [a,b] \to \mathbb{R}$. On se donne n points $(x_0, \dots, x_{n-1})$ choisis de manière aléatoire dans $[a,b]$ (en suivant une loi uniforme). Alors $\frac{b-a}{n}\sum_{i=0}^{n-1} f(x_i)$ est une approximation de $\int_a^b f$, avec un intervalle de confiance à 95% de largeur $\alpha/\sqrt{n}$.

Exercice 13. *1.* Écrire une fonction `point(a,b)` renvoyant un réel aléatoire de $[a,b]$.

2. En déduire une implémentation `monte_carlo(f,a,b,n)` de la méthode.

3. Comparer la précision de cette méthode avec la méthode des rectangles, et avec `scipy.integrate` en faisant plusieurs essais.

Remarque : cette méthode n'est pas très précise. Néanmoins elle se généralise très bien en dimension supérieure, et est la seule réellement applicable en grandes dimensions.

4.3 Utilisation de `odeint` pour des systèmes d'équations différentielles

On rappelle que la forme générale d'un système différentiel vue plus haut est :

$$\begin{cases} y'(t) &= f(y(t),t) \\ y(t_0) &= y_0 \end{cases}$$

où f et la fonction inconnue y sont à valeurs dans $\mathbb{R}^k$ pour un certain k. Cette forme générale permet de résoudre des équations d'ordre 1, mais également des équations d'ordre supérieur [3] ou même des systèmes d'équations. On peut alors utiliser `odeint` de la même manière, mais l'entrée y_0 est maintenant un tableau Numpy, et la fonction f renvoie un tableau Numpy de la même taille. Le résultat de `odeint` est un tableau Numpy `Y` à 2 dimensions : `Y[i]` est un tableau Numpy de taille k contenant une approximation de y au temps t_i.

Exercice 14. *Équations de Lotka-Volterra.* On considère un écosystème où vivent des proies (par exemple des sardines), et des prédateurs (par exemple des requins), dépendant du temps. On note $r(t)$ la densité de prédateurs au temps t, et $s(t)$ la densité de proies dans le milieu. On considère le système différentiel suivant, qui modélise l'évolution de ces densités, où interviennent 4 constantes A, B, C et D toutes strictement positives.

$$\begin{cases} r'(t) &= Ar(t)s(t) - Br(t) \\ s'(t) &= -Cr(t)s(t) + Ds(t) \end{cases}$$

L'idée derière ces équations est naturelle : la population des requins augmente en proportion du nombre de sardines, et décroît en l'absence de proies. Inversement, les sardines se multiplient en l'absence de prédateurs, mais diminuent en proportion du nombre de requins !

3. L'astuce est la suivante : par exemple pour une équation d'ordre 2 de la forme $y''(t) + ay'(t) + by(t) + c = 0$, on introduit le vecteur $Y(t) = \binom{y(t)}{y'(t)}$. Puisque $Y'(t) = \binom{y'(t)}{y''(t)} = \binom{y'(t)}{-ay'(t)-by(t)-c}$, on peut bien écrire $Y'(t)$ sous la forme $f(Y(t),t)$, où $f(\binom{y_0}{y_1},t) = \binom{y_1}{-ay_1-by_0-c}$ (qui dépend de son second argument t si les fonctions a, b et c ne sont pas constantes).

On se donne également deux réels r_0 et s_0 donnant les densités (qui sont donc des réels positifs) au temps $t = 0$. On admettra par la suite que le système admet une unique solution $t \mapsto (r(t), s(t))$ vérifiant $x(0) = r_0$ et $y(0) = s_0$ définie sur $\mathbb{R}_+$.

Travail demandé.

1. Introduire une fonction f pour réécrire le système sous la forme $y'(t) = f(y(t), t)$, avec $y(t) = \binom{r(t)}{s(t)}$.
2. Définir informatiquement une telle fonction `fLotka`.
3. Résoudre le système avec comme condition initiale $(r_0, s_0) = (0.8, 0.7)$, sur l'intervalle de temps $[0, 30]$ (on prendra 5000 points sur l'intervalle).
4. Tracer sur un graphique $r(t)$ en abscisse et $s(t)$ en ordonnée. Observer que la solution est périodique !