Informatique - TP N°1

Méthode d'Euler (1)

- La **méthode d'Euler** est une méthode simple de **résolution numérique des équations différentielles *du premier ordre***.
  Dans une équation différentielle du premier ordre sur un intervalle I, l'inconnue est une fonction y de la variable t. Pour tout t dans I, le nombre dérivé $y'(t)$ s'exprime en fonction de $y(t)$ et de la variable t, donc vérifie une équation de la forme :

$$\forall t \in I, y'(t) = f(t, y(t)) \qquad (E)$$

  où f est une fonction connue à deux variables.

  Sous certaines conditions sur f, on sait qu'il existe une unique solution y vérifiant la condition initiale $y(a) = y_0$ ($a \in I$ et $y_0 \in \mathbb{R}$ sont fixés).

- **En pratique, il n'est pas toujours possible de déterminer explicitement cette solution**. On cherche alors à en construire une **approximation discrète** : une suite (y_k) de valeurs approchées de y, permettant d'obtenir des points "proches" de la courbe exacte.
  Remarque : dans ce TP, nous utilisons cette méthode pour obtenir une courbe approchée d'une solution y que l'on sait trouver de façon exacte, ce qui permet de comparer la courbe exacte avec les courbes approchées obtenues.

- Pour calculer une solution approchée sur un intervalle $[a; b]$ par la méthode d'Euler :

  - On subdivise l'intervalle à l'aide de $n+1$ valeurs $t_0 = a, t_1, ..., t_n = b$ régulièrement espacées d'un **pas**

$$h = \frac{b-a}{n} \qquad (\text{avec } n \in \mathbb{N}^*)$$

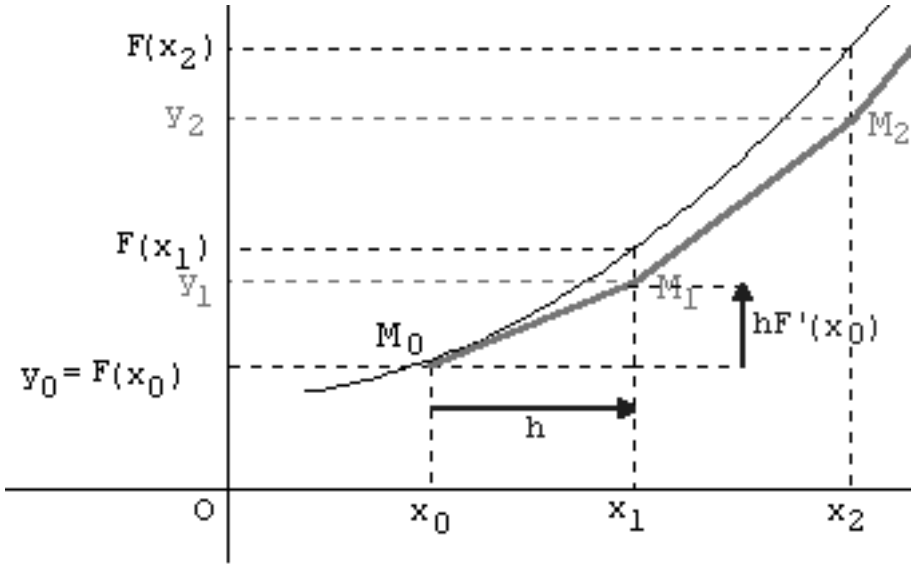
  - La valeur de $y_0 = y(a)$ est **connue** (c'est la condition initiale) ;
  - On approche successivement les valeurs de y en $t_1, t_2, ..., t_n$ en utilisant le fait que, pour h 'petit', la tangente à la courbe de la fonction y est 'très proche' de la courbe de y : $y(t_{k+1}) \approx y_{k+1}$, où

$$y_{k+1} = y'(t_k)(t_{k+1} - t_k) + y(t_k)$$

  est l'ordonnée du point d'abscisse t_{k+1} sur la **tangente à la courbe** de y au point $M_k(t_k; y(t_k))$
  Comme $t_{k+1} - t_k = h$ (le pas) et $y(t_k) \approx y_k$, on obtient :

$$\boxed{y_{k+1} = y'(t_k) \times h + y_k}$$

  On peut ainsi calculer toutes les valeurs approchées y_k de façon récurrente à partir de y_0.



Ci-dessus : les valeurs de la solution exacte y sont notées $F(x_k)$, les valeurs approchées y_k. On a noté $x_0 = a$, $x_1 = x_0 + h$,...,$x_n = b$.

1ère équation différentielle simple

On considère une équation (E_0) issue de la physique (décharge de condensateur) :

$$\forall t \in \mathbb{R}_+, \quad \text{R}.y'(t) + \frac{1}{\text{C}}y(t) = 0 \quad (E_0), \text{avec condition initiale } t_0 = 0, y_0 = \text{U}_0.$$

En posant $K = (\text{RC})^{-1}$, on peut l'écrire

$$y'(t) = -Ky(t)$$

et la solution exacte de cette équation différentielle est facile à obtenir, c'est

$$y(t) = \text{U}_0.e^{-K.t}$$

On appellera $\mathbf{g_0}$ la fonction y solution **exacte** qui vérifie $g_0(0) = \text{U}_0$.

On prendra $K = 70\ s^{-1}$ et $\text{U}_0 = 1800$ mV .
Remarque : les valeurs de K et U_0 sont à définir dans le programme principal pour pouvoir les utiliser dans les fonctions sans avoir besoin de les passer en paramètre.

1. Soit f_0 la fonction telle que l'équation différentielle (E_0) s'écrive

$$\forall t \in \mathbb{R}, y'(t) = f_0(t, y(t))$$

   - Écrire l'expression de f_0 en fonction de de t et y.
   n.b. : ici (E_0) est très simple... La valeur de t est-elle utilisée explicitement dans le calcul de la dérivée ? Est-ce toujours le cas ?
   - Écrire une fonction python `f0(t,y)`, qui renverra donc la valeur de $y'(t)$.
   - Enregistrer sous `TP1_Euler_v1.py`

2. Écrire également une fonction python `g0(t)` qui renvoie la valeur de la **solution exacte** pour une valeur de t donnée.

3. Écrire une fonction python, `euler`, qui reçoit en entrée *cinq* paramètres :
la fonction `f` qui permet de calculer le nombre dérivé, trois réels `y0`, `a`, `b` (`a` < `b`), et un entier non nul `n`.
Cette fonction calcule de proche en proche chaque t_{k+1} et y_{k+1} et renvoie la liste `Y` contenant $y_0, y_1, ..., y_n$.

4. Tester (dans le shell) la fonction `euler` pour l'équation (E_0), avec comme condition initiale $y(0) = 1800mV$. On prendra `a=0`, `b=0.5`, `n=20` (d'où un pas $h = 0.025\ s$).

5. Comparer (dans le shell) l'approximation obtenue en $t = 0.5$ (c'est la dernière valeur de la liste) à la valeur en $t = 0.5$ de la solution exacte. Est-ce satisfaisant ?

6. On veut maintenant visualiser le résultat.
Compléter le programme principal : il doit appeler la fonction `euler` et tracer la courbe approchée obtenue pour `n=20`. (Pour le tracé il faut aussi construire la liste des $n + 1$ valeurs d'abscisses t_k dans $[a; b]$, cela peut se faire rapidement avec une liste en compréhension).
Le programme calculera aussi les $n + 1$ valeurs exactes `g0(t)` pour tous les t_k de la liste d'abscisses, afin de tracer g sur le même graphique.

7. Modifier le programme principal pour qu'il trace, sur le même graphique, g et les approximations de y obtenues pour `n=20, 30, 40, 50, 80, 160`. Commenter.

2ème équation différentielle

8. On considère l'équation différentielle suivante définie sur $\mathbb{R}$:

$$\forall t \in \mathbb{R}, y'(t) = y(t) + 2t \qquad (E_1)$$

Résoudre sur papier cette équation différentielle, avec la condition initiale $y(0) = 1$.
On appellera g_1 la fonction y solution exacte qui vérifie $g_1(0) = 1$.

9. Soit f_1 la fonction telle que l'équation différentielle (E_1) s'écrive

$$\forall t \in \mathbb{R}, y'(t) = f_1(t, y(t))$$

- Écrire l'expression de f_1 en fonction de de t et y.

- Enregistrer grâce à "save as" une version 2,TP1_Euler_v2.py

- Écrire une fonction python `f1(t,y)` qui renvoie la valeur de $y'(t)$ pour cette équation.

- Écrire également une fonction python `g1(t)` qui renvoie la valeur exacte.

- Modifier le programme principal : il utilise la fonction `euler` pour l'équation différentielle (E_1), avec comme condition initiale $y(0) = 1$. On prendra `a=0`, `b=4` ; `n=40`, puis `400`, puis `4000` (d'où un pas h de 0.1, puis 0.01, puis 0.001) : tracer la courbe exacte et chaque courbe approchée sur le même graphique.

10. Mesure de l'erreur maximale.
- Écrire une fonction `ecartMax(L1,L2)` qui retourne le plus grand écart (en valeur absolue) entre deux éléments de `L1` et `L2` de même indice ; `L1` et `L2` sont deux listes de nombres qui ont la même longueur.
- Compléter le programme principal pour qu'il affiche pour chaque valeur de n l'écart maximum (en valeur absolue) entre la liste de valeurs de g_1 $[g_1(t_0), g_1(t_1), ..., g_1(t_n)]$ et la liste des approximations $y_0, y_1, ..., y_n$.
- Observer l'évolution de cet écart maximum en fonction de h.Que peut-on conjecturer ?

11. *Pour aller plus loin* : Méthode de Heun

Dans la méthode d'Euler, à partir du point (t_k, y_k), on a utilisé l'approximation de $y(t_{k+1})$ par $y_k + hy'(t_k)$ en utilisant le coefficient directeur de la tangente à la courbe au point $M_k(t_k, y(t_k))$, soit $y'(t_k)$.

Dans la méthode de Heun, au lieu de $y'(t_k)$, on utilise la **moyenne** entre le coefficient directeur $y'(t_k)$ et celui de la tangente au point approché suivant
$M_{k+1}(t_{k+1}, y_k + hy'(t_k))$ (trouvé par la méthode d'Euler).
Avec cette valeur moyenne des deux nombres dérivés, on détermine un nouveau point M'_{k+1} d'abscisse t_{k+1}, qu'on espère plus précis.
A partir du point obtenu, on continue selon le même principe.

- Rédiger rapidement le calcul nécessaire pour obtenir l'ordonnée du point M'_{k+1} obtenu par la méthode de Heun à partir du point $M'_k(t_k, y_k)$, puis écrire une fonction Python, `heun`, qui fournira une approximation de la solution par cette méthode.

- Sauvegarder sous un nouveau nom et modifier le programme principal pour visualiser sur un graphique la comparaison entre solution exacte et solution approchée par la méthode de Heun.

- Comparer numériquement la méthode de Heun avec la solution exacte ; observer l'évolution de l'écart maximum en fonction de h.