2TSI. TD PYTHON 03

Équations différentielles d'ordre 1

Il faut savoir que les méthodes qui suivent s'appliquent généralement pour des équations différentielles quelconques, c'est-à -dire non nécessairement linéaires. Or seules ces dernières sont au programme officiel en TSI2 en Mathématiques. Maintenant, rien ne vous empêchera d'utiliser ces méthodes dans les autres épreuves (Modélisation, Physique, SI, Informatique pour tous), où peuvent apparaître des équations différentielles non linéaires.

❑ **Méthode 0.1.— Comment résoudre de façon exacte ou numériquement une équation différentielle linéaire d'ordre** 1 **avec Python**

On commence par écrire l'équation différentielle sous la forme $y'(t) = f(y(t), t)$ et la condition initiale $y(t_0) = y_0$, ce qui permet de généraliser à une équation non linéaire, c'est-à -dire dans le cas où $f(y(t), t)$ n'est pas de la forme $a(t)y(t) + b(t)$.

- ***Première piste. On peut utiliser le calcul formel*** et donc le module *sympy*. Il faut avoir chargé le module *sympy* sur votre ordinateur si ce n'est déjà fait puis l'importer par :
  $from\ sympy\ import\ *$
  On utilise alors trois fonctions de *sympy* qui sont *dsolve*, *Function* et *Derivative*. Donnons un exemple. On veut trouver les solutions de $y'(t) - y(t) = e^t$. On tape :
  $y\ =\ Function('y')\ ;\ t\ =\ symbols('t')$
  $dsolve(Derivative(y(t), t) - y(t) - exp(t), y(t))$
  Il s'affiche alors :
  $y(t)\ ==\ (C1\ +\ t) * exp(t)$
  Notez l'utilisation obligatoire des fonctions *Function* et *symbols* sans lesquelles on obtient un message d'erreur.
- ***Deuxième piste. On peut utiliser la fonction*** *odeint* ***du sous-module*** *scipy.integrate.* Cette fonction nécessite une liste de valeurs de t, commençant en t_0 et une condition initiale y_0. La fonction *odeint* renvoie des valeurs approchées (aux points contenus dans la liste des valeurs de t) de la solution y de l'équation différentielle qui vérifie $y(t_0) = y_0$. Plus précisément, si l'on désire les solutions pour $t \in [t_0, t_n]$ avec un pas h, on tape :
  $import\ scipy.integrate\ as\ integr$
  $def\ f(y,t)\ :$
  $\quad return\ expression$
  $T\ =\ np.arange(t0, tn, h)\ ;\ Y\ =\ integr.odeint(f, y0, T)$
  Y renvoyé est un tableau dont chaque ligne correspond à la valeur d'une fonction à un instant donné. Ainsi $Y[0]$ renvoie y_0 et $Y[-1]$ renvoie $y(t_1)$. Si l'on tape alors :
  $plt.plot(T, Y)$
  $plt.show()$
  alors on a le tracé de la courbe intégrale.
- ***Troisième piste. On peut construire une procédure Python basée sur l'algorithme d'Euler.***

T.S.V.P →

Exercice 01 : utilisation des fonctions prédefinies

Déterminer sur $[0,1]$ la solution de $y'(t) = t\,y(t) + t$ qui vérifie $y(0) = 1$ en utilisant d'abord *sympy* puis *odeint*.

Exercice 02 : algorithme d'Euler

On cherche à déterminer une solution approchàe de $y'(t) = f(y(t), t)$ valable sur un intervalle donné, c'est-à -dire $t \in [t_0, t_n]$. On suppose la condition initiale $y(t_0) = y_0$. Pour cela, on choisit une subdivision $(t_0, t_1, ..., t_n)$ de $[t_0, t_n]$ à pas constant $h = (t_n - t_0)/n$. Donc $\forall k \in [\![0, n-1]\!]$, $t_{k+1} = t_k + h$. On définit alors la liste $(y_1, ..., y_n)$ telle que $\forall k \in [\![0, n-1]\!]$, $y_{k+1} = hf(y_k, t_k)$.

1. Expliquer l'algorithme dit **méthode d'Euler**.
2. Écrire une fonction *Euler* qui prend en argument f, $t0$, tn, n et $y0$. On créera dans une boucle la liste T des abscisses et celle Y des valeurs prises par la solution y aux points considérées.
   On renverra Y. Appliquer à $(t+1)y'(t) + y(t) = \cos t$ pour $t \in [0, 10]$ avec $y(0) = 1$ et $n = 10$.

   Écrire ensuite une fonction $Euler\_Affich$ de mêmes arguments, qui doit renvoyer l'affichage de la courbe reliant les points calculés. En notant Δt et Δy les amplitudes des valeurs manipulées, on créera une fenêtre graphique qui déborde de 10% de chaque côté du tracé.
3. Appliquer à $(t+1)y'(t) + y(t) = \cos t$ pour $t \in [0, 10]$ avec $y(0) = 1$ et $n = 35$.
4. Appliquer à l'équation différentielle de l'exercice 01 avec $n = 100$.