Utilisations de la méthode d'Euler

Capacités exigibles

- Mettre en œuvre la méthode d'Euler à l'aide d'un langage de programmation pour simuler la réponse d'un système linéaire du premier ordre à une excitation de forme quelconque
- À l'aide d'un langage de programmation, résoudre numériquement une équation différentielle du deuxième ordre non-linéaire et faire apparaître l'effet des termes non-linéaires
- À l'aide d'un langage de programmation, obtenir des trajectoires d'un point matériel soumis à un champ de force centrale conservatif
- À l'aide d'un langage de programmation, mettre en évidence le non isochronisme des oscillations du pendule simple

I Documents

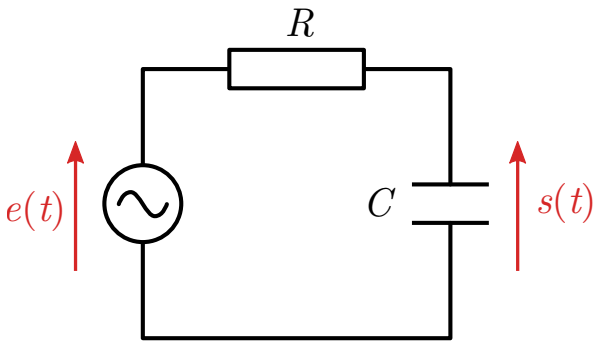
Document 1 : Filtre RC

Un filtre RC est caractérisé par le schéma ci-contre, et l'équation ci-dessous :

$$\frac{\mathrm{d}s}{\mathrm{d}t} + \frac{s}{RC} = \frac{e}{RC}$$

On prendra un signal d'entrée sinusoïdal, de la forme

$$e(t) = e_0 \sin(\omega t)$$



Document 2 : Équation du pendule simple

L'état du pendule simple est caractérisé par son angle θ qui évolue selon l'équation différentielle suivante :

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + \frac{g}{l}\sin\theta = 0$$

Document 3 : Méthode d'Euler explicite pour une équation d'ordre 1

Lorsque l'on a une équation différentielle, il est parfois impossible d'exprimer des solutions générales. Cependant on peut en trouver une forme approchée à l'aide de méthode numérique.

Prenons l'exemple d'une équation différentielle du premier ordre quelconque pour une fonction $u(t)$. On peut la mettre sous la forme

$$\dot{u} = f(u)$$

Par exemple $\dot{u} = -ku$ pour une équation linéaire homogène...

Discrétisation :
On va considérer que le temps est discret, et on note $\mathrm{d}t$ le pas d'évolution temporel. Ainsi, on pourra indicer les valeurs prises par u :

$$u(t) = u(n \cdot \mathrm{d}t) \underset{\text{discrétisation}}{\longrightarrow} u_n$$

Évolution :
On peut alors calculer l'état u_{n+1} à partir de l'état u_n. Pour cela, on peut écrire le développement limité de u :

$$u(t+\mathrm{d}t) = u(t) + \dot{u}\,\mathrm{d}t = u(t) + f(u)\,\mathrm{d}t$$

$$\downarrow \text{ discrétisation}$$

$$u_{n+1} = u_n + f(u_n)\,\mathrm{d}t$$

On choisit alors une situation initiale u_0, puis l'algorithme calcule tous les u_n de proche en proche. Ce qui nous donne au final l'évolution de $u(t)$!

⚠ Évidemment, plus $\mathrm{d}t$ est petit, plus la résolution sera précise. Par contre elle prendra plus de temps

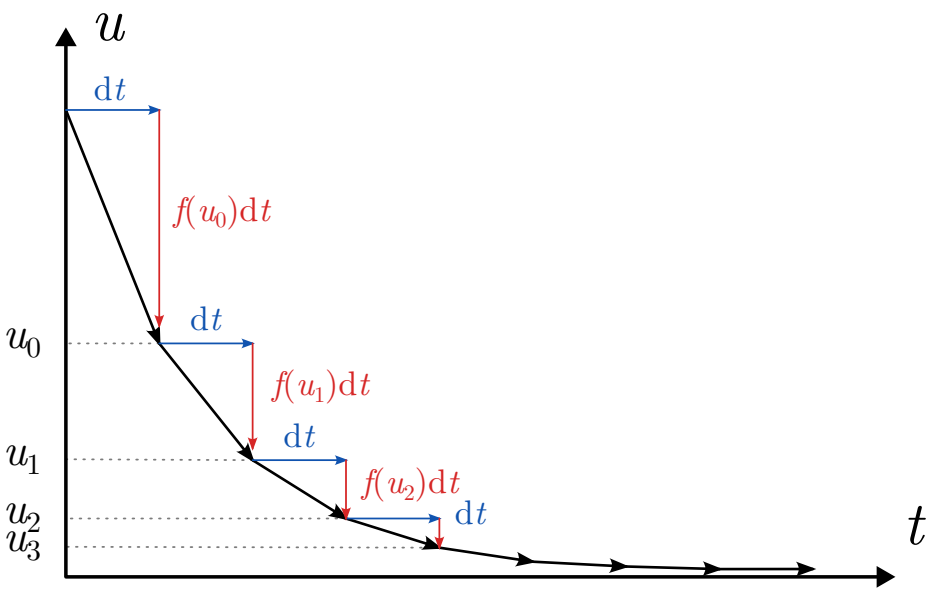


Figure 1 – Représentation schématique de la forme des solutions dans le cas où $f(u) = -ku$. On retrouve bien l'allure exponentielle attendue

Document 4 : Généralisation aux ordres supérieurs

Il est facile de généraliser la méthode d'Euler pour des équations différentielles aux ordres 2, 3, ...

Prenons l'exemple d'une équation d'ordre 2. Cette fois-ci on peut la mettre sous la forme

$$\ddot{u} = f(u, \dot{u})$$

On pose alors le vecteur

$$U(t) = \begin{pmatrix} u(t) \\ \dot{u}(t) \end{pmatrix}$$

Cette nouvelle grandeur suit elle-même une équation différentielle du premier ordre ! En effet, on a

$$\dot{U} = \begin{pmatrix} \dot{u} \\ \ddot{u} \end{pmatrix} = F(U) \qquad \text{avec} \quad F\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} y \\ f(x,y) \end{pmatrix}$$

On peut alors appliquer la méthode précédente à ce vecteur, ce qui nous donnera une suite contenant pour chaque instant la valeur de u et de sa dérivée $\dot{u}$. On retrouve ici le théorème de Cauchy-Lipschitz : il faut donner comme conditions initiales U_0, c'est-à-dire u_0 ET $\dot{u}_0$, pour calculer les états suivants de proche en proche.

II Énoncé

A Simulation d'un filtre du premier ordre

On étudie le cas du circuit RC détaillé en document 1.

1. Créer un premier fichier python et le sauvegarder immédiatement.
2. Introduire les paramètres de la simulation et donner leur les valeurs que vous voulez (on les ajustera à la fin) :
   - `N` le nombre de points temporels de la simulation
   - `t_max` le temps final de la simulation
   - `RC` le produit RC
   - `omega` la pulsation du signal d'entrée ω
   - `e0` l'amplitude du signal d'entrée e_0
   - `s0` la valeur initiale de S
3. **À partir de ce variables**, construire les grandeurs ci-dessous :
   - un tableau `temps` contenant tous les instants de 0 à `t_max` en utilisant la fonction `np.linspace`.
   - la fonction `entree` renvoyant la valeur du signal d'entrée e à l'instant t.
4. Coder la fonction `circuit_RC(s, t)` renvoyant $\dot{s}$ la dérivée de la sortie pour des valeurs données de s et de t (vous serez amené · es à utiliser la fonction `entree(t)`).
5. Créer la fonction `Euler1(F, U0, instants)` ci-dessous prenant pour arguments :
   - `F(U, t)` la fonction renvoyant $\dot{U}(t)$: la dérivée de U à l'instant t
   - `U0` la condition initiale $U(0)$
   - `instants` le tableau contenant tous les instants de la simulation

   et la compléter de sorte à lui faire renvoyer le tableau contenant toute l'évolution de $U(t)$ grâce à la méthode d'Euler (cf. document 3).

   ⚠ Vous veillerez à bien utiliser **uniquement** les arguments `F`, `U0` et `instants` (pas de variables globales).

```
def Euler1(F, U0, instants):

    '''
    Résout l'équation différentielle d'ordre 1 représentée par la fonction 'F', à
     partir d'une condition initiale 'U0', sur tous les instants du tableau '
    instants'.
    '''

    Etats = np.zeros( ?? )  # Etats est un tableau dont la n-ième valeur sera Un
    (représentant U(t))
    Etats[0] = ??   # Définition de la condition initiale

    for k in range( ?? , ?? ):

        dt = ??
        t = ??
        Etats[k + 1] = ??

    return ??
```

6. Utiliser les fonctions `circuit_RC` et `Euler1` pour simuler l'évolution de la réponse $s(t)$ pendant 10 périodes de l'entrée, à partir de

$$s(0) = 0$$

   (Si vous n'obtenez pas de résultat satisfaisant, vérifiez que votre valeur de `N` est suffisamment grande)

B Pendule simple

À présent, on cherche à coder une version généralisée de cette méthode, pouvant s'appliquer sur une équation différentielle d'ordre supérieur.

7. Créer un nouveau fichier python et le sauvegarder.
8. Introduire les paramètres (en choisissant des valeurs cohérentes) :
   - `g = 9.81` l'intensité du champ de pesanteur (en SI)
   - `l` la longueur du pendule (en m)
   - `m` la masse du pendule (en kg)
9. Coder la fonction `pendule(U, t)`, renvoyant la dérivée du vecteur $U = (\theta, \dot{\theta})$ sous forme d'un tableau `numpy`.
10. En vous inspirant de votre première fonction `Euler1`, créer la fonction `Euler(F, U0, instants)` renvoyant le tableau contenant toute l'évolution du **vecteur** $U(t)$ grâce à la méthode d'Euler (document 4).
11. **Analyse :**
    a) Représenter l'allure de $\theta(t)$ pour différentes conditions initiales, en particulier que se passe-t-il lorsque $\theta_0 \to \pi$ (sans vitesse initiale) ?
    b) Superposer les évolutions de l'énergie cinétique E_c, de l'énergie potentielle et de l'énergie mécanique.

$$\begin{cases} E_c = \dfrac{1}{2} m (l\dot{\theta})^2 \\ E_p = mgl\,(1 - \cos\theta) \end{cases} \qquad E_m = E_c + E_p$$

Le tout est-il cohérent ?

Remarque

La fonction `Euler` que vous avez créée s'utilise de la même manière qu'une fonction déjà implémentée :

`odeint` de la bibliothèque `scipy.integrate`

Cette dernière **est à savoir utiliser** au concours !

Vous pouvez vous amusez à vérifier que les résultats que vous obtenez sont cohérents avec ceux d'`odeint`.

Pour aller plus loin :
À la place de vos lignes d'affichage, introduisez le bout de code suivant :

```
1  def func_anim(k):
2
3      [theta, v_theta] = Etats[int(k * vitesse_anim) % N]
4      line.set_data([?, ?], [?, ?])
5      return line,
6
7  vitesse_anim = 10   # Vitesse de l'animation. Ne change pas les calculs, mais
       vous pouvez l'ajuster pour voir une évolution satisfaisante
8
9  fig = plt.figure()
10 ax = fig.add_subplot(111)
11 ax.set_aspect('equal')  # Axes x et y représentés avec la même échelle
12 ax.set_xlim((-1.1, 1.1))
13 ax.set_ylim((-1.1, 1.1))
14
15 # line va contenir les coordonnées des points à afficher. On pourra les mettre à
       jour avec la fonction line.set_data
16 line, = ax.plot([], [])
17
18 anim = FuncAnimation(fig, func_anim, interval=10)
19 # interval représente le nombre de millisecondes à attendre entre deux frames
20
21 plt.show()
```

Et n'oubliez d'ajouter la ligne d'import au début de votre programme :

```
1  from matplotlib.animation import FuncAnimation
```

(12) Recopier ce code et complétez la ligne 4 en remplaçant les ? par ce qu'il faut, de sorte que l'animation trace en tout temps la tige du pendule (de l'origine à la masse)

III Annexe

Pour ce TP vous aurez besoin des modules `numpy` et `pyplot` :

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

Vous aurez ensuite à utiliser les fonctions suivantes :

`np.array(L)`

Transforme une liste classique en un tableau `numpy`. Il s'agit de l'objet de base sur lequel on travaille. Les opérations effectuées ont la même syntaxe que pour des flottants.

Exemple :

```
>>> x = np.array([1, 2, -1.5])
array([ 1. ,  2. , -1.5])
>>> x + 10
array([ 11. ,  12. , 8.5])
>>> 2 * x
array([ 2. ,  4. , -3.])
>>> x ** 2
array([ 1. ,  4. , 2.25.])
```

`np.zeros(dim)`

Créer un tableau remplis de `0`, de dimension `dim`.

Exemples :

```
>>> x = np.zeros((2, 4))
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

```
>>> x = np.zeros(3)
array([0., 0., 0.])
```

Vous pouvez ensuite changer les valeurs avec le slicing :

```
>>> x[0, 1] = -5    # Dans la première ligne, change la deuxième valeur
>>> x[1, :] = [1, 2, 3, 4]  # Les : signalent "toutes les valeurs"
>>> x
array([[0., 0., -5., 0.],
       [1., 2., 3., 4.]])
```

`np.linspace(xi, xf, N)`

Créer un tableau 1D allant de `xi` à `xf` en `N` points.

Exemple :

```
>>> np.linspace(0, 1, 11)
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ])
```

`np.sin(x), np.cos(x)`

Calcule le sinus ou cosinus d'un tableau `x` (qui peut être de dimension `(1,)` donc être simplement un flottant).