Simulation numérique de l'effet d'un filtre

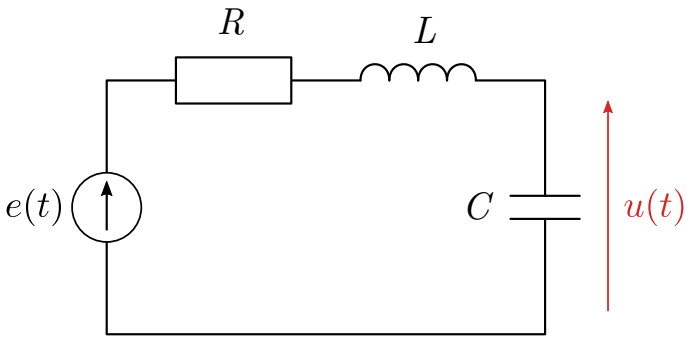
Capacités exigibles

- Simuler, à l'aide d'un langage de programmation, l'action d'un filtre sur un signal périodique.
- Mettre en évidence l'influence des caractéristiques du filtre sur l'opération de filtrage.

I Documents

Document 1 : RLC série

Le circuit RLC série permet d'étudier plusieurs types de filtres. On rappelle sa forme :



En prenant la tension $u(t)$ aux bornes du condensateur, on a alors un passe-bas, dont l'équation différentielle est la suivante :

$$LC\,\ddot{u} + \frac{L}{R}\,\dot{u} + u = e$$

On peut en extraire une pulsation de coupure ω_0 et un facteur de qualité Q :

$$\omega_0 = \frac{1}{\sqrt{LC}} \qquad Q = \frac{1}{R}\sqrt{L/C}$$

On mènera l'étude en régime sinusoïdal forcé :

$$e(t) = a\cos\omega t$$

II Énoncé

1. Créer un fichier python et le sauvegarder immédiatement.
2. Créer des variables pour chacun des paramètres physiques et numériques et donner leurs des valeurs arbitraire (mais cohérentes), quitte à la modifier plus tard :
   - `R` qui représente la valeur de la résistance ;
   - `C` qui représente la valeur de la capacité du condensateur ;
   - `L` qui représente la valeur de l'inductance propre de la bobine ;
   - `U0` les conditions initiales : $[u(0), \dot{u}(0)]$;
   - `a` l'amplitude l'excitation $e(t)$;
   - `omega` la pulsation de l'excitation $e(t)$;
   - `k` le nombre de périodes sur lesquelles on va simuler le circuit ;
   - `N` le nombre de points de la simulation.
3. À partir de ces grandeurs, créer les variables suivantes :
   - `temps` le tableau contenant les instants sur lesquels on va mener l'intégration
   - `entree(t)` représentant la fonction $e(t)$. Vous ajouterez un bruit au résultat, en utilisant la fonction `np.random.random` (cf. Annexe), en veillant à ce que le bruit
     - soit de moyenne nulle ;
     - ne soit pas d'amplitude trop élevée.
4. Afficher la forme de $e(t)$ et vérifier que le bruit est visible, mais suffisamment faible. Modifier votre fonction `entree(t)` jusqu'à ce que ce soit correct.
5. Coder une fonction `Filtre(U, t)` renvoyant la dérivée du vecteur $U = (u, \dot{u})$ à l'instant t :

$$U(t) = \begin{pmatrix} u(t) \\ \dot{u}(t) \end{pmatrix} \xrightarrow[\texttt{Filtre}]{} \dot{U}(t) = \begin{pmatrix} \dot{u}(t) \\ \ddot{u}(t) \end{pmatrix}$$

   Il faudra utiliser l'équation différentielle donnée en document 1.
6. Avec la fonction `odeint` et en utilisant le slicing de numpy (cf. Annexe), résoudre numériquement l'équation différentielle.
7. Afficher sur un même graphique l'entrée $e(t)$ et la sortie $u(t)$ et vérifier que le filtre agit bien comme un passe-bas.

Pour aller plus loin :

10. Modifier votre entrée dans le programme python en ajoutant des sinusoïdes supplémentaires afin d'enrichir le spectre. Par soucis de réalisme, les amplitudes des harmoniques devront être de plus en plus faibles.

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 :

`slicing`

Il est très facile d'extraire et de modifier certaines valeurs d'un tableau `numpy`.
Exemple : Si on a une matrice 2D notée `M` :

```
M = array([[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]])
```

On peut extraire des valeurs :

```
>>> M[0]    # Renvoie la première ligne
array([1, 2, 3])
>>> M[0, 2]     # Renvoie la valeur en première ligne, troisième colonne
3
>>> M[:, 1]     # Renvoie toutes les valeurs de la deuxième colonne
array([2, 5, 8])
```

On remarque donc que les deux points ":" signifient "toutes les valeurs".

`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.random.random(n)`

Créer un tableau 1D de `n` valeurs aléatoires choisies uniformément entre 0 et 1 :
Exemple :

```
>>> np.random.random(4)
array([0.85568752, 0.94219879, 0.01725846, 0.2365631 ])
```

`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).

odeint

Dans la bibliothèque `scipy.integrate`, se trouve une fonction `odeint` permettant de résoudre numériquement des équations différentielles, et de manière bien plus précise que la méthode d'Euler :

```
from scipy.integrate import odeint

...

solutions = odeint(F, Y0, X)
```

Les arguments doivent être les suivants :

- `F` la fonction qui encode l'équation différentielle (ici par exemple une équation d'ordre n sur une fonction $y : x \to y(x)$) :

$$Y'(x) = F(Y(x), x)$$

  Elle doit prendre elle-même en entrée :

  - `Y` un vecteur de taille n (liste python) contenant pour un x donné, les valeurs de $y(x)$ et ses $n-1$ premières dérivées :

$$Y(x) = \begin{pmatrix} y(x) \\ y'(x) \\ \vdots \\ y^{(n-1)}(x) \end{pmatrix}$$

  - `x` l'abscisse à laquelle on mène le calcul.

  Elle doit renvoyer le vecteur dérivé (toujours sous forme de liste python) :

$$Y'(x) = \begin{pmatrix} y'(x) \\ y''(x) \\ \vdots \\ y^{(n)}(x) \end{pmatrix}$$

- `Y0` les conditions initiales

$$Y(x_1) = \begin{pmatrix} y(x_1) \\ y'(x_1) \\ \vdots \\ y^{(n-1)}(x_1) \end{pmatrix}$$

- `X` La liste des abscisses sur lesquelles on va mener l'intégration :

$$X = [\, x_1, x_2, \ldots, x_N \,]$$

  Souvent on utilise la fonction `np.numpy` (cf. précédemment), pour créer cette liste.

Alors `odeint` renvoie un tableau `numpy` de taille `N` $\times$ `n` avec

- `N` le nombre de points choisis pour intégrer l'équation
- `n` l'ordre de l'équation différentielle

$$\texttt{odeint} \implies \begin{pmatrix} y(x_1) & y'(x_1) & \ldots & y^{(n-1)}(x_1) \\ y(x_2) & y'(x_2) & \ldots & y^{(n-1)}(x_2) \\ \vdots & \vdots & & \vdots \\ y(x_N) & y'(x_N) & \ldots & y^{(n-1)}(x_N) \end{pmatrix}$$