ÉTUDE DE L'OSCILLATEUR A PONT DE WIEN

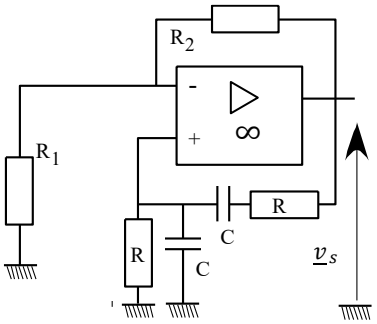
I. Equations de l'oscillateur à pont de Wien (avant de programmer en python)

I.1. Rappels théoriques

On rappelle que, si on pose $\alpha = 1 + \frac{R_2}{R_1}$, on a en régime harmonique :

$\underline{v}^- = \frac{1}{\alpha}\,\underline{v}_s$ et $\underline{v}^+ = \frac{jRC\omega}{1+3jRC\omega-R^2C^2\omega^2}\,\underline{v}_s$.

On pourra poser $\omega_0 = \frac{1}{R*C}$



- Ecrire l'équation différentielle (1) reliant $v_s(t)$ à $v^+(t)$
- Ecrire l'équation (2) reliant $v_s(t)$ à $v^-(t)$

I.2. Fonctionnement en régime linéaire

- En régime linéaire, en utilisant (2), écrire l'équation (3) (très simple) liant $v_s(t)$ à $v^+(t)$.
- Quelle inégalité doit vérifier $|v_s(t)|$, et donc $|v^+(t)|$ pour que l'ALI fonctionne en régime linéaire ?
- Exprimer alors $\frac{dv_s}{dt}(t)$ en fonction de $\frac{dv^+}{dt}(t)$.

I.3. Fonctionnement en régime de saturation haute

- En régime de saturation haute, que vaut $v_s(t)$? que vaut $v^-(t)$?
- Exprimer alors $\frac{dv_s}{dt}(t)$.
- Quelle inégalité doit vérifier $v^+(t)$ pour que l'ALI fonctionne en saturation haute ?

I.4. Fonctionnement en régime de saturation basse

- En régime de saturation basse, que vaut $v_s(t)$? que vaut $v^-(t)$?
- Exprimer alors $\frac{dv_s}{dt}(t)$.
- Quelle inégalité doit vérifier $v^+(t)$ pour que l'ALI fonctionne en saturation basse ?

I.5. Lien général entre $v_s(t)$ et $v^+(t)$

- Regrouper les résultats obtenus en I.2, I.3 et I.4 en donnant les différentes expressions de $\frac{dv_s}{dt}$ selon la valeur que prend v^+ (donc selon que l'ALI est en fonctionnement linéaire, saturé à l'état haut ou saturé à l'état bas).
- On définit à présent la fonction $f_2(v^+)$ donnant la valeur de $\frac{dv_s}{dt}$ en fonction de celle de v^+. C'est une fonction constante par morceaux La représenter, c'est-à-dire tracer $f_2(v^+)$ en fonction de v^+.
- Montrer que, quel que soit le régime de fonctionnement de l'ALI (linéaire, saturé haut ou saturé bas), $\frac{dv_s}{dt}(t)$ peut se calculer au moyen de $f_2(v^+(t))$ et de $\frac{dv^+}{dt}(t)$.
- En déduire (grâce à l'équation différentielle (1) du I.1) l'équation différentielle du second ordre vérifiée par $v^+(t)$ quel que soit le type de fonctionnement, linéaire ou saturé. Cette équation différentielle, que l'on notera (4) utilise notamment la fonction $f_2(v^+)$.

II. Simulation numérique de l'oscillateur à pont de Wien avec python

II.1. Utilisation de l'instruction odeint du module scipy.integrate

Après avoir installé `odeint` du module scipy.integrate (`from scipy.integrate import odeint`), on peut résoudre numériquement une équation différentielle de la forme :

$$\frac{dy}{dt} = f(y,t)$$

Et ceci de façon bien plus performante qu'avec la méthode d'Euler.
La variable `y` est, soit un scalaire, soit un vecteur.
La syntaxe est de la forme suivante :

- Définir une fonction python `f`, correspondant au second membre de l'équation différentielle ;
- Déclarer une valeur initiale `yini` de `y`.
- Créer un tableau `t` à une dimension, contenant les instants pour lesquels on souhaite calculer les valeurs successives de la solution `y`.

Ainsi, `odeint(f, yini,t)` renvoie les valeurs de `y` pour les différentes valeurs successives du tableau `t`.

Avant de passer à la programmation de l'oscillateur quasi-sinusoïdal à pont de Wien, étudions deux exemples plus simples :

Exemple 1 : On cherche à résoudre l'équation différentielle du **premier ordre** : $\frac{dy}{dt}+\frac{y}{\tau}=0$, avec $y(t=0)=yini$

```python
import matplotlib.pyplot  as plt
import numpy as np
from scipy.integrate import odeint
Ne=400;tau=0.01;Te=5*tau/Ne # Déclaration des constantes

def f(y,t_ech):
    return -y/tau # Définition de la fonction f
yini=2.  # Condition initiale
t_ech=np.array([n*Te for n in range(Ne)])  # tableau de valeurs des instants t

sol=odeint(f,yini,t_ech)
plt.plot(t_ech,sol)
plt.axis([0,5*tau,0,yini])
plt.show()
```

Exemple 2 : On cherche à résoudre l'équation différentielle du **second ordre** : $\frac{d^2v}{dt^2}+2\zeta\omega_0\frac{dv}{dt}+\omega_0^2 v=0$, avec $v(t=0)=vini$ et $\frac{dv}{dt}\ (t=0)=vpini$.

On définit le « vecteur d'état » $y(t)=\left(v(t),\frac{dv}{dt}(t)\right)$. Les deux composantes du vecteur y sont notées y_0 et y_1 : $y=(y_0,y_1)$.

On a donc $y_0=v$ et $y_1=\frac{dv}{dt}$

Et on a aussi $\frac{dy}{dt}\ (t)=\left(\frac{dv}{dt}(t),\frac{d^2v}{dt^2}(t)\right)$, c'est-à-dire $\frac{dy}{dt}\ =\left(y_1,\frac{dy_1}{dt}\right)$

Avec cette « astuce », on se ramène à une équation différentielle d'ordre 1, puisque ce que l'on a à résoudre est
$\frac{dy}{dt}=f(y,t)$,
avec $f(y,t)=(y_1,-2\zeta\omega_0 y_1-\omega_0^2 y_0\ )$

```python
import matplotlib.pyplot  as plt
import numpy as np
from scipy.integrate import odeint

# Déclaration des constantes
Ne=400
tau=0.01
zeta=-0.1 # essayer d'autres valeurs de zeta, en changeant la valeur et le signe ; zeta est le coefficient d'amortissement
f0=1000
w0=2*np.pi*f0
Te=5/f0/Ne

def f(y,t_ech):
 # à compléter

yini=np.array([0.1,0.])  # Condition initiale

t_ech=np.array([n*Te for n in range(Ne)])  # tableau de valeurs des instants t

sol=odeint(f,yini,t_ech)
plt.plot(t_ech,sol[:,0])
plt.show()
```

II.2. Application à l'oscillateur à pont de Wien

- Définir les constantes (Vsat=15 V, R = 10 kΩ, C = 100 nF, R_1 = 1,2 kΩ, R_2 = 2,52 kΩ pour commencer), le nombre d'échantillons, la période d'échantillonnage de la variable temporelle.
- Définir la fonction f_2 du I.5.
- Définir la fonction f pour la résolution de l'équation différentielle (elle correspond à (4) du I.5).
- Créer la variable vectorielle `t` (ou `t_ech`) pour le temps
- Résoudre l'équation différentielle.
- Tracer v^+ en fonction du temps.
- Tracer la trajectoire de phase pour v^+, c'est-à-dire $\frac{dv^+}{dt}$ en fonction de v^+