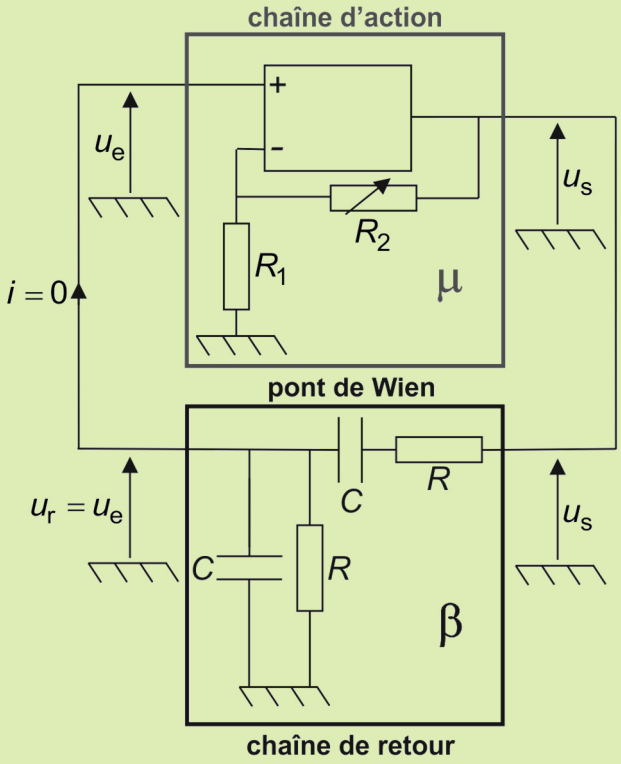
Capacité numérique : à l'aide d'un langage de programmation, simuler l'évolution temporelle d'un signal généré par un oscillateur.

Rappel de cours : oscillateur à pont de Wien



La chaîne d'action est un montage amplificateur non inverseur, dont la fonction de transfert est $\mu = \dfrac{\underline{u_s}}{\underline{u_e}} = 1 + \dfrac{R_2}{R_1}$. La chaîne de retour est un pont de Wien, passe-bande du second ordre, dont la fonction de transfert est $\beta = \dfrac{\underline{u_e}}{\underline{u_s}} = \dfrac{1}{3 + j\left[\dfrac{\omega}{\omega_0} - \dfrac{\omega_0}{\omega}\right]} = \dfrac{j\dfrac{\omega}{\omega_0}}{1 + 3j\dfrac{\omega}{\omega_0} - \left(\dfrac{\omega}{\omega_0}\right)^2}$, où $\omega_0 = \dfrac{1}{RC}$.

On en déduit l'équation différentielle linéaire reliant u_e à u_s par la chaîne de retour :

$$(j\omega)^2 \underline{u_e} + 3j\omega\omega_0 \underline{u_e} + \omega_0{}^2 \underline{u_e} = j\omega\omega_0 \underline{u_s} \leftrightarrow \frac{d^2 u_e}{dt^2} + 3\omega_0 \frac{du_e}{dt} + \omega_0{}^2 u_e = \omega_0 \frac{du_s}{dt}.$$

Pour la chaîne directe, on a :

— $u_s = \left[1 + \dfrac{R_2}{R_1}\right] u_e = \mu u_e$ si l'A.L.I est en fonctionnement linéaire, ce qui est vérifié si on a $-V_{sat} < u_s < V_{sat} \Leftrightarrow -\dfrac{V_{sat}}{\mu} < u_e < \dfrac{V_{sat}}{\mu}$.

— $u_s = +V_{sat}$ si $u_e \geq \dfrac{V_{sat}}{\mu}$, et $u_s = -V_{sat}$ si $u_e \leq -\dfrac{V_{sat}}{\mu}$, l'A.L.I étant alors saturé.

On peut à l'aide de ces équations qui couplent u_s et u_e étudier le régime transitoire (démarrage des oscillations) et le régime établi d'oscillations. Toutefois, les oscillations n'ont lieu qu'à une condition sur R_2 / R_1 que nous allons déterminer.

En raison du bruit de fond, les conditions initiales $u_e(0^+)$ et $\frac{du_e}{dt}(0^+)$ ne sont pas rigoureusement nulles car il existe un bruit de fond électronique. Néanmoins on a $-V_{sat}/\mu < u_e(0^+) < V_{sat}/\mu$, donc l'A.L.I est initialement en fonctionnement linéaire, ce qui entraîne $u_s = \mu u_e$. On en déduit :

$\frac{d^2u_e}{dt^2} + \omega_0[3-\mu]\frac{du_e}{dt} + {\omega_0}^2 u_e = 0 \Leftrightarrow \frac{d^2u_e}{dt^2} + \omega_0\left[2 - \frac{R_2}{R_1}\right]\frac{du_e}{dt} + {\omega_0}^2 u_e = 0$. Ainsi :

— Si $R_2/R_1 < 2$, le régime est amorti. Les signaux restent du bruit de fond.

— Si $R_2/R_1 > 2$, le régime est amplifié. Les signaux divergent et le système donne naissance à un signal.

On n'a donc un oscillateur que si $R_2/R_1 > 2$.

Le but de cette capacité numérique est de résoudre l'équation différentielle du second ordre :

— $\frac{d^2u_e}{dt^2} = -\omega_0[3-\mu]\frac{du_e}{dt} - {\omega_0}^2 u_e$ avec $\mu = 1 + \frac{R_2}{R_1} = 1 + A$, si $|u_e(t)| < V_{sat}/\mu$

— $\frac{d^2u_e}{dt^2} = -3\omega_0\frac{du_e}{dt} - {\omega_0}^2 u_e$ si $|u_e(t)| > V_{sat}/\mu$.

Les conditions initiales $u_e(0^+)$ et $\frac{du_e}{dt}(0^+)$ sont faibles et sans grande importance du moment que $|u_e(0^+)| < V_{sat}/\mu$. On prendra $u_e(0^+) = 0{,}01$ V et $\frac{du_e}{dt}(0^+) = \omega_0 u_e(0^+)$.

Plutôt que d'utiliser la bibliothèque permettant de réaliser ces intégrations, on peut programmer un schéma d'intégration (on a choisi le schéma de Runge-Kutta, bien plus performant que le schéma d'Euler), afin de gérer le changement d'équation différentielle à résoudre selon la valeur de $u_e(t)$.

Méthode numérique : résolution numérique d'équations différentielles

On cherche à résoudre numériquement l'équation différentielle d'ordre 1 : $\frac{dx}{dt} = \varphi(t, x(t)) = F(t)$ sur l'intervalle $[t_0, t_n]$ avec la condition initiale $x(t_0) = x_0$.

On divise l'intervalle $[t_0, t_n]$ en n intervalles $[t_i, t_{i+1}]$ avec $i \in [\![0, n-1]\!]$, de longueur $dt = \frac{t_n - t_0}{n}$, en posant $t_i = t_0 + i\,dt$.

Sans approximation, on a $x_{i+1} = x_i + \int_{t_i}^{t_{i+1}} F(t)dt$. La méthode d'Euler consiste à considérer que F est constante, égale à $F(t_i)$, entre t_i et t_{i+1} : $x_{i+1} = x_i + \frac{dx}{dt}(t_i)\cdot dt = x_i + \varphi(t_i, x_i)\cdot dt$. Elle nécessite un grand nombre de pas d'intégration pour obtenir une solution approchée satisfaisante (on montre que l'erreur commise est de l'ordre de dt).

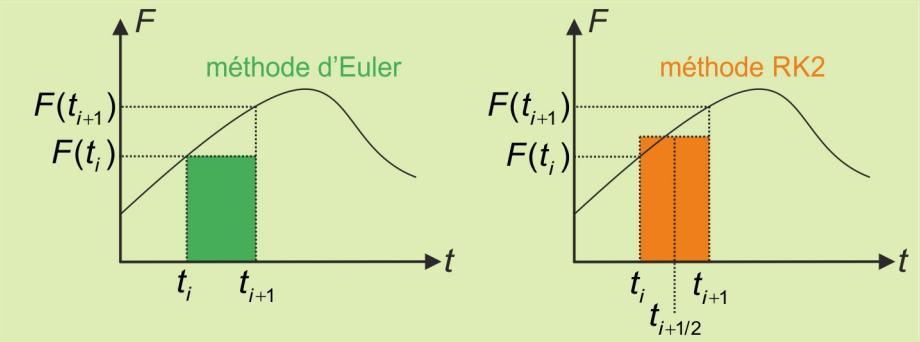
Pour la même valeur de n, on approche mieux la solution exacte en augmentant le nombre d'évaluations à chaque pas de temps. Par exemple la méthode de Runge-Kutta d'ordre 2 (RK2) consiste à supposer que F est constante, égale à $F(t_{i+1/2})$ où $t_{i+1/2} = \frac{1}{2}\left[t_i + t_{i+1}\right]$. On a donc : $x_{i+1} = x_i + \varphi(t_{i+1/2}, x_{i+1/2}) \cdot \mathrm{d}t$. La méthode est cependant implicite car le second membre dépend de $x_{i+1/2}$ qui est inconnu. On le calcule donc par la méthode d'Euler : $x_{i+1/2} = x_i + \varphi(t_i, x_i) \cdot \frac{\mathrm{d}t}{2}$. Ainsi, il faut à chaque itération effectuer les calculs suivants :

— Calcul de $K_1 = \varphi(t_i, x_i)$.

— Calcul de $K_2 = \varphi(t_{i+1/2}, x_{i+1/2}) = \varphi\left(t_i + \frac{\mathrm{d}t}{2}, x_i + K_1 \cdot \frac{\mathrm{d}t}{2}\right)$.

— Calcul de $x_{i+1} = x_i + K_2 \cdot \mathrm{d}t$.

Cette fois-ci, l'erreur commise est de l'ordre de $(\mathrm{d}t)^2$ et décroît en $1/n^2$.



Dans le schéma de Runge Kutta, l'erreur commise est de l'ordre de $(\mathrm{d}t)^4$, en utilisant les évaluations suivantes :

— Calcul de $K_1 = \varphi(t_i, x_i)$.

— Calcul de $K_2 = \varphi\left(t_i + \frac{\mathrm{d}t}{2}, x_i + K_1 \cdot \frac{\mathrm{d}t}{2}\right)$.

— Calcul de $K_3 = \varphi\left(t_i + \frac{\mathrm{d}t}{2}, x_i + K_2 \cdot \frac{\mathrm{d}t}{2}\right)$.

— Calcul de $K_4 = \varphi\left(t_i + \mathrm{d}t, x_i + K_3 \cdot \mathrm{d}t\right)$.

— Calcul de $x_{i+1} = x_i + \left[K_1 + 2K_2 + 2K_3 + K_4\right] \cdot \frac{\mathrm{d}t}{6}$.

Pour résoudre une équation différentielle d'ordre 2 : $\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = \varphi\left(t, x, \frac{\mathrm{d}x}{\mathrm{d}t}\right)$ avec les conditions initiales $x(t_0) = x_0$ et $\dot{x}(t_0) = \dot{x}_0$, on pose $X = \begin{bmatrix} x \\ \frac{\mathrm{d}x}{\mathrm{d}t} \end{bmatrix}$ et on a donc $\frac{\mathrm{d}X}{\mathrm{d}t} = \begin{bmatrix} \frac{\mathrm{d}x}{\mathrm{d}t} \\ \frac{\mathrm{d}^2x}{\mathrm{d}t^2} = \varphi\left(t, x, \frac{\mathrm{d}x}{\mathrm{d}t}\right) \end{bmatrix} = \Phi(t, X(t))$ et on se ramène à une équation différentielle du premier ordre, la fonction recherchée X étant cette fois-ci un vecteur.

Pour en savoir plus, vous pouvez consulter le document pdf sur la résolution numérique d'équations différentielles.

Q.1) Introduire les constantes t_0, t_n, n, $A = R_2 / R_1$, $\mu = 1 + A$, ω_0, V_{sat}, x_0 (valeur initiale de la fonction u_e, notée x) et $\dot{x}_0$:

```
t0 = 0. # temps initial
tn = 2E-1 # temps final
n = 15000 # nombre d'intervalles
A = 2.01 # rapport R2/R1
mu = 1+A # gain de l'ampli non inverseur
omega0 = 2*np.pi*1E3 # pulsation 1/RC
Vsat = 14
x0 = 0.01 # valeur initiale de la fonction ue, notée x
xp0 = omega0*x0 # valeur initiale de sa dérivée
```

Définir les deux équations différentielles intervenant selon la valeur de à l'aide des fonctions `Phi1(t,X)` (si $|u_e(t)| < V_{sat} / \mu$) et `Phi2(t,X)` (si $|u_e(t)| > V_{sat} / \mu$). Ces fonctions renvoient les composantes `xp` ,`xpp` du vecteur $\frac{dX}{dt}$.

Q.2) Écrire une fonction `RK4(t0,tn,x0,xp0,n)` qui renvoie, en appliquant le schéma de Runge-Kutta d'ordre 4, le tableau numpy 1D, `T`, des valeurs de `ti`, et le tableau numpy 2D, `X`, dont la ligne `i` contient les deux valeurs $x(t_i)$ et $\dot{x}(t_i)$.

Q.3) Écrire les commandes permettant de tracer la courbe représentative de $t \mapsto u_e(t)$ ainsi que la trajectoire de phase.
Tracer ces courbes en prenant d'abord $A = 1{,}95$. Commenter l'allure du signal obtenu.
Recommencer pour $A = 2{,}01$. Commenter la durée du régime transitoire et la forme du signal en régime établi.
Répondre aux mêmes questions si $A = 3$ (on pourra alors prendre `tn = 2E-2`).

Créer des tableaux numpy

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

np.array([xp,xpp]) # Crée un tableau numpy 1D à deux colonnes, conte-
nant les valeurs xp et xpp.

T = np.zeros(n+1) # Crée un tableau 1D de n + 1 colonnes rempli de 0
T[0] = t0 # Affecte t0 à la première colonne de ce tableau
X = np.zeros((n+1,2)) # Crée un tableau de n+1 lignes et 2 colonnes,
rempli de 0
X[0] = [x0,xp0] # Place les valeurs initiales dans la première ligne
de ce tableau
```