

Cas des équations  
différentielles

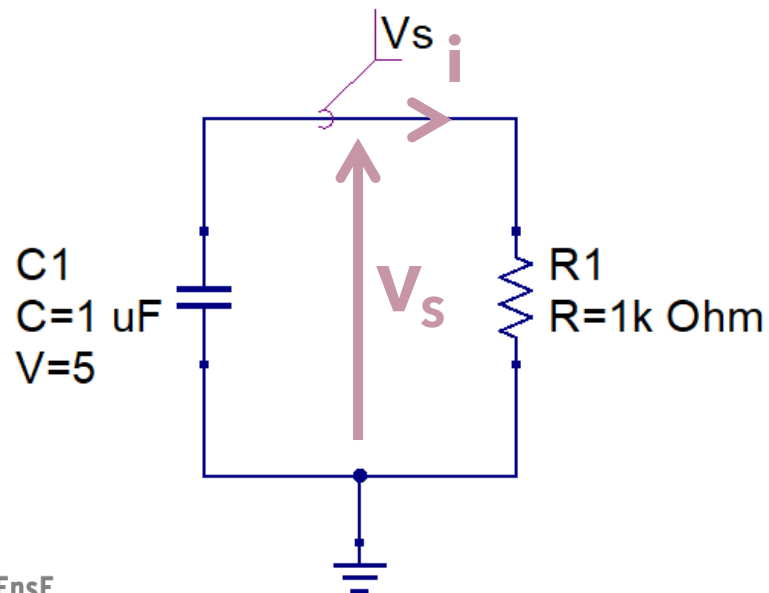
# Intégration Numérique (Euler)

Outils Numériques / Semestre 5  
/ Institut d'Optique / B1\_4

# Approche analytique



- Approche analytique



$$\Rightarrow V_s = -R_1 \cdot C_1 \cdot \frac{dV_s}{dt}$$

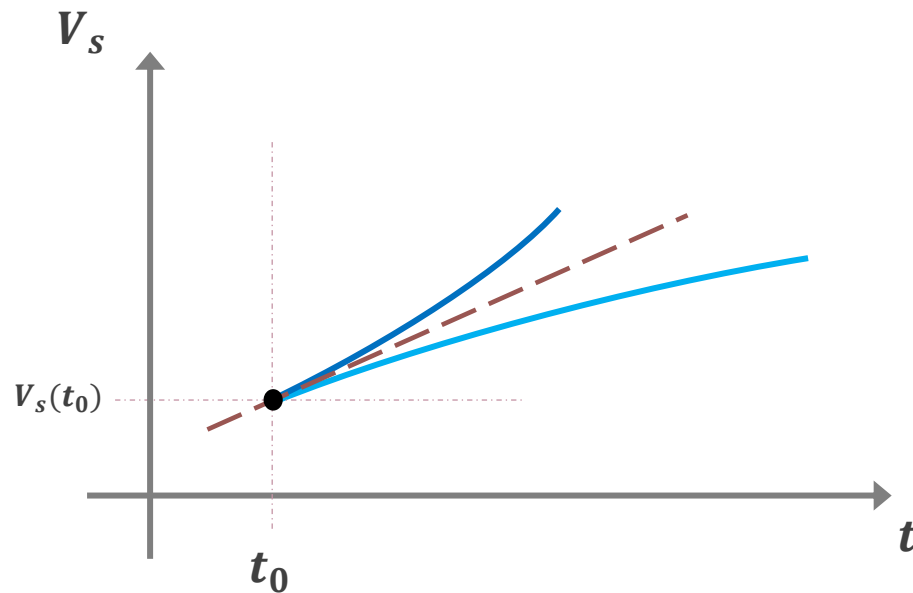
*Equation différentielle d'ordre 1*

$$\Rightarrow \frac{dV_s}{dt} = -\frac{1}{R_1 \cdot C_1} \cdot V_s$$

# Approche graphique



- Intégration numérique



$$\Rightarrow V_s = -R_1 \cdot C_1 \cdot \frac{dV_s}{dt}$$

*Equation différentielle d'ordre 1*

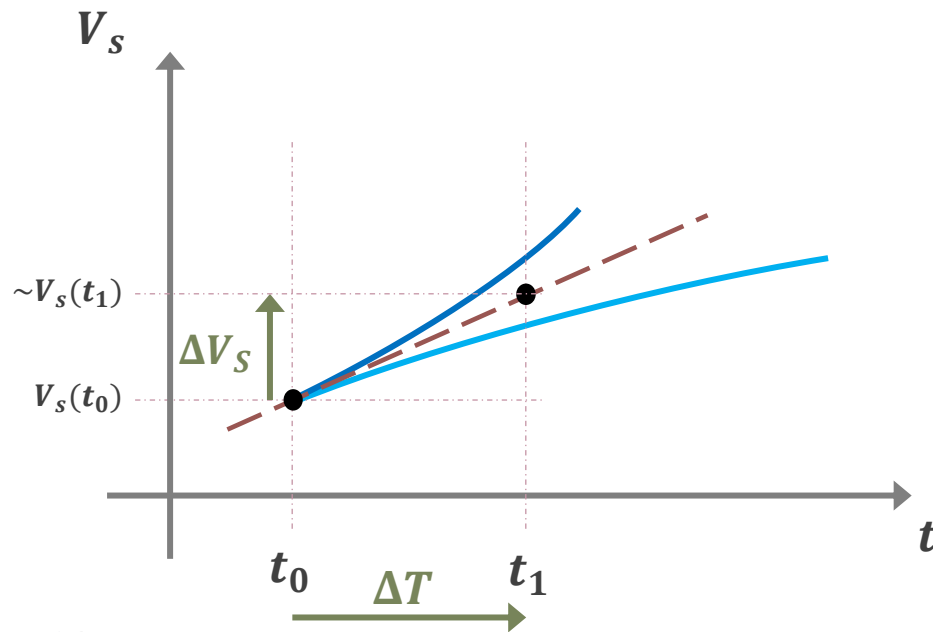
$$\Rightarrow \frac{dV_s}{dt} = -\frac{1}{R_1 \cdot C_1} \cdot V_s$$

*Coefficient directeur de la tangente  
à la courbe en un point donné*

# Approche graphique



- Intégration numérique



$$\Rightarrow V_s = -R_1 \cdot C_1 \cdot \frac{dV_s}{dt}$$

Equation différentielle d'ordre 1

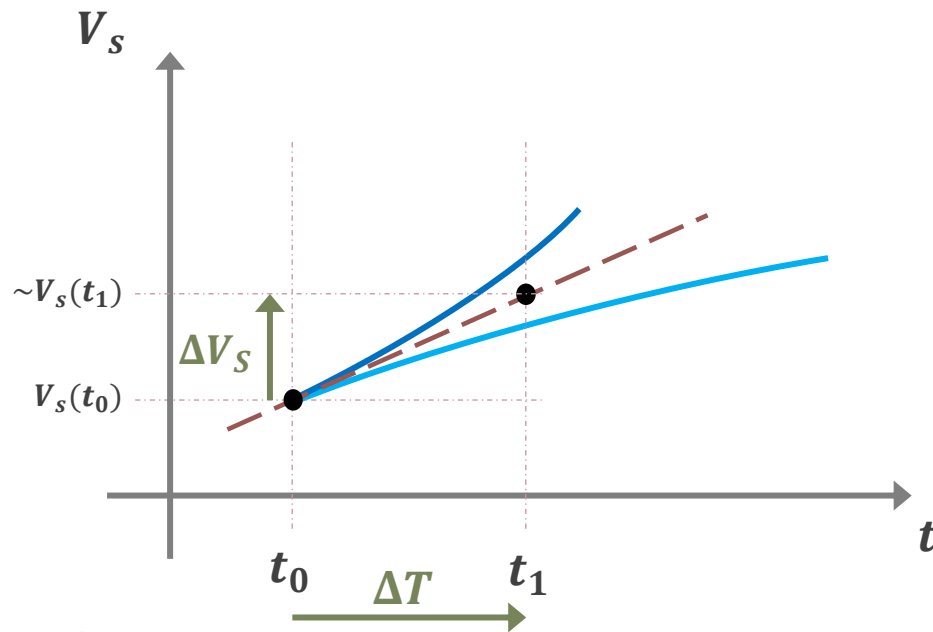
$$\Rightarrow \frac{dV_s}{dt} = -\frac{1}{R_1 \cdot C_1} \cdot V_s$$

Coefficient directeur de la tangente à la courbe en un point donné

# Approche graphique



- Intégration numérique



$$\Rightarrow \frac{dV_s}{dt} = -\frac{1}{R_1 \cdot C_1} \cdot V_s$$

*Coefficient directeur de la tangente  
à la courbe en un point donné*

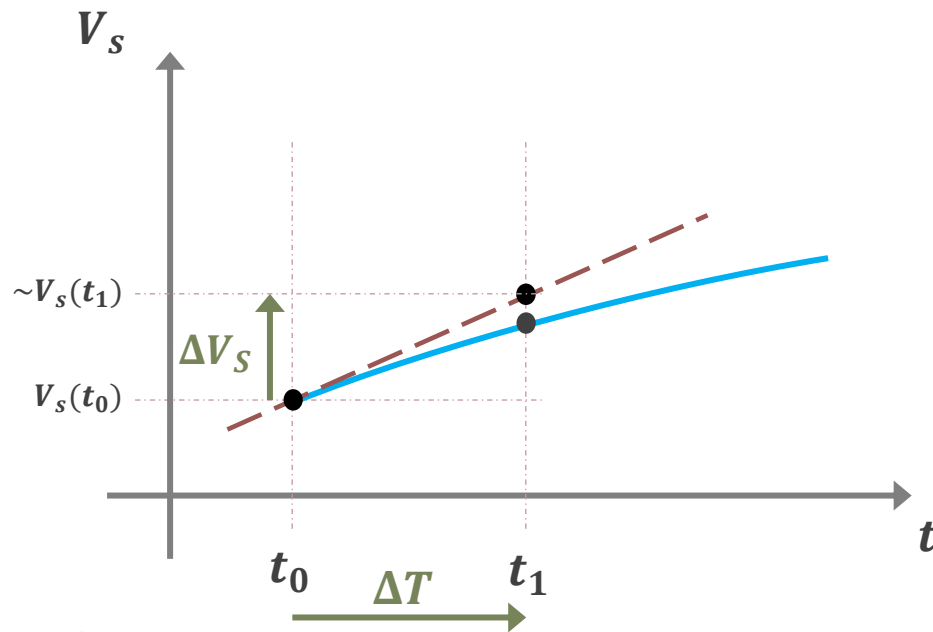
$$\sim V_s(t_1) = V_s(t_0) + \Delta V_s$$

avec 
$$\Delta V_s = \Delta T \cdot \left. \frac{dV_s}{dt} \right|_{t=t_0}$$

# Approche graphique



- Intégration numérique



$$\Rightarrow \frac{dV_s}{dt} = -\frac{1}{R_1 \cdot C_1} \cdot V_s$$

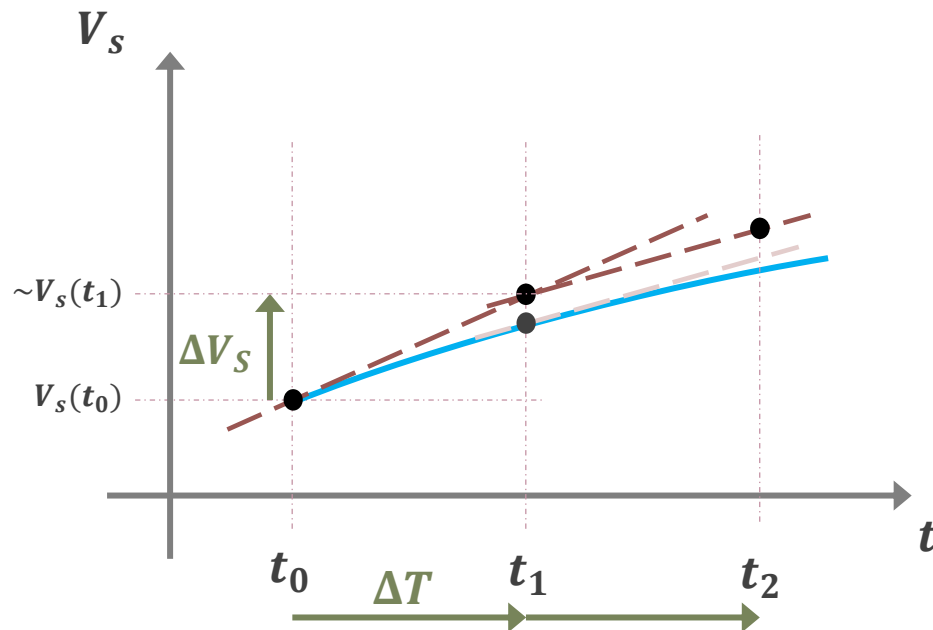
*Coefficient directeur de la tangente  
à la courbe en un point donné*

$$V_s(t_1) \sim V_s(t_0) + \Delta T \cdot \left. \frac{dV_s}{dt} \right|_{t=t_0}$$

# Approche graphique



- Intégration numérique



$$\Rightarrow \frac{dV_s}{dt} = -\frac{1}{R_1 \cdot C_1} \cdot V_s$$

*Coefficient directeur de la tangente à la courbe en un point donné*

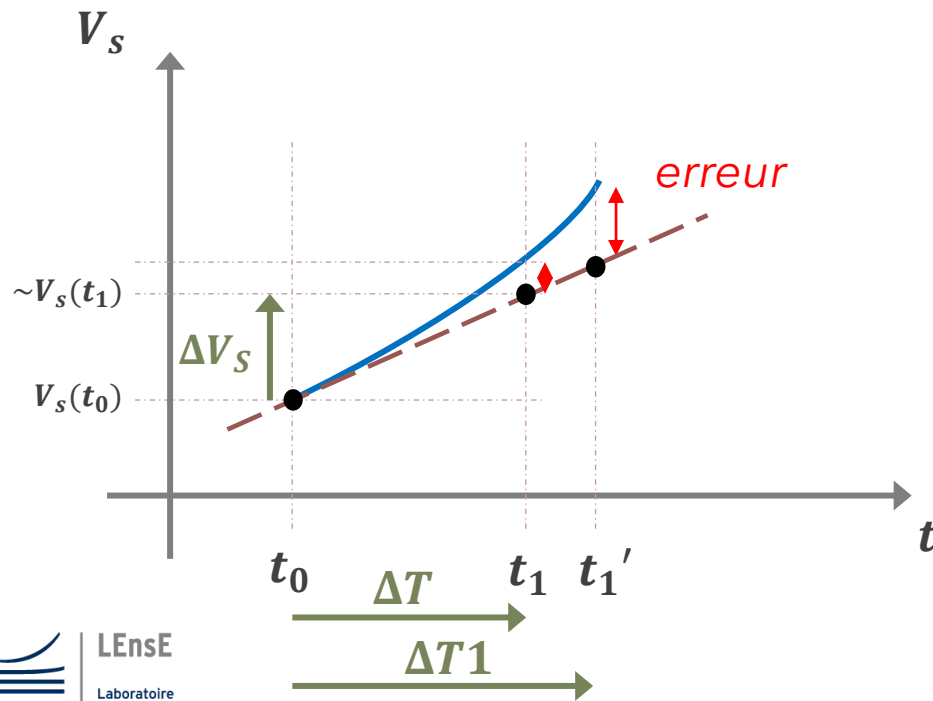
$$V_s(t_1) \sim V_s(t_0) + \Delta T \cdot \left. \frac{dV_s}{dt} \right|_{t=t_0}$$

$$V_s(t_2) \sim V_s(t_1) + \Delta T \cdot \left. \frac{dV_s}{dt} \right|_{t=t_1}$$

# Approche graphique



- Méthode d'**Euler** (explicite)



$$\rightarrow \frac{dV_s}{dt} = -\frac{1}{R_1 \cdot C_1} \cdot V_s$$

Coefficient directeur de la tangente  
à la courbe en un point donné

$$V_s(t_{n+1}) \sim V_s(t_n) + \Delta T \cdot \left. \frac{dV_s}{dt} \right|_{t=t_n}$$



**APPROXIMATION  
NUMERIQUE**



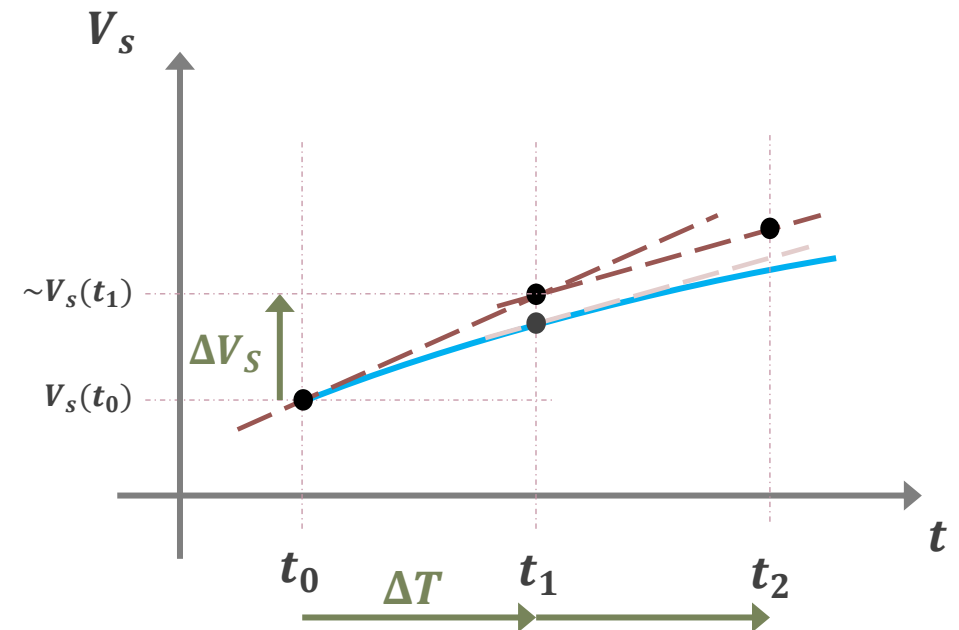
# Intégration Numérique

- Intégration Numérique

Calculer des **valeurs approchées** d'une fonction  **$y(t)$**

sur un intervalle de  $[t_0, n \cdot \Delta T]$  de  **$t$**

en connaissant sa **dérivée** et un point particulier de la fonction



# Implémentation de la méthode d'Euler

- Données d'entrée

$F(t, V_S(t))$  : fonction qui définit l'équation différentielle

$V_S(t_0)$  : condition initiale

$N$  et  $\Delta T$  : le nombre de points souhaités et le pas de calcul d'intégration

ou  $T_{total}$  et  $\Delta T$

- Algorithme de calcul

$$t_{n+1} = t_n + \Delta T$$

$$V_S(t_{n+1}) \sim V_S(t_n) + \Delta T \cdot F(t_n, V_S(t_n))$$

# Implémentation de la méthode d'Euler

**S'ENTRAINER**

- Résolution par intégration

$$\Rightarrow \frac{dV_s}{dt} = -\frac{1}{R_1 \cdot C_1} \cdot V_s$$



+ Définir la fonction  $F(V_s(t_n), t_n)$

+ Implémenter l'algorithme de la méthode d'Euler explicite dans une fonction prenant comme paramètres :  $F, V_s(0), T_{total}, N, R, C$

+ Tracer l'évolution en fonction du temps pour  $R = 100 \text{ k}\Omega$  et  $C = 1 \text{ }\mu\text{F}$  et au moins 3 valeurs de  $\Delta T$  différentes (pour un temps total équivalent)

$$V_s(t_{n+1}) \sim V_s(t_n) + \Delta T \cdot F(V_s(t_n), t_n)$$

# Implémentation de la méthode d'Euler

**CORRECTION**

- Définition de la fonction ***F***

```
def F(t, Vs, R=1e5, C=1e-6):  
    return -Vs/(R*C)
```

- Implémentation Euler

```
def explicit_euler(F, Vs0, Ttot, N, R=1e5, C=1e-6):  
    dt = Ttot/N  
    t = numpy.zeros(N+1)  
    Vs = numpy.zeros(N+1)  
    Vs[0] = Vs0  
  
    for n in range(N):  
        t[n+1] = t[n] + dt  
        Vs[n+1] = Vs[n] + F(t[n], Vs[n], R, C)*dt  
  
    return t, Vs
```

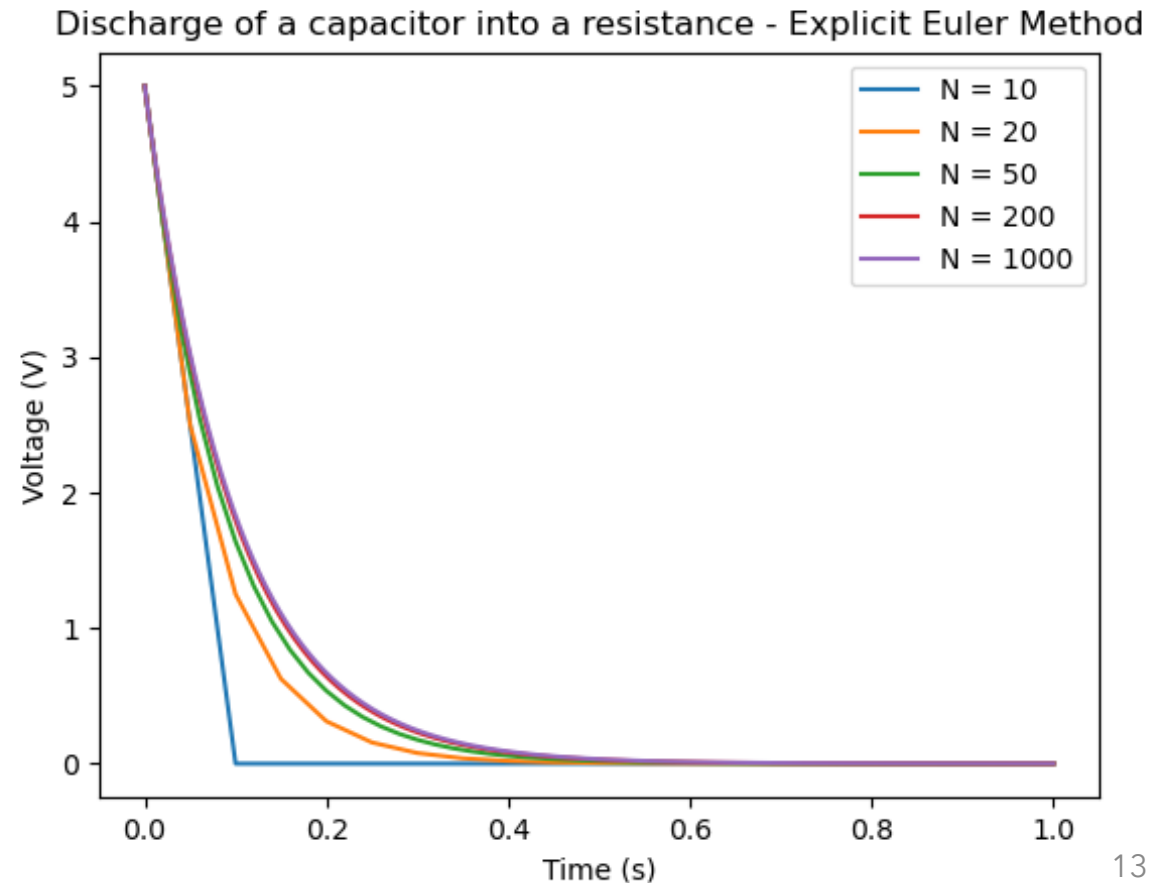
# Implémentation de la méthode d'Euler

## RESULTATS

- Simulation paramétrique

```
N_val = [10, 20, 50, 200, 1000]
plt.figure()
for k in range(len(N_val)):
    t, Vs = explicit_euler(F, 5, 1, N_val[k])
    plt.plot(t, Vs, label=f'N = {N_val[k]}')

plt.legend()
plt.show()
```



# Implémentation de la méthode d'Euler

## RESULTATS

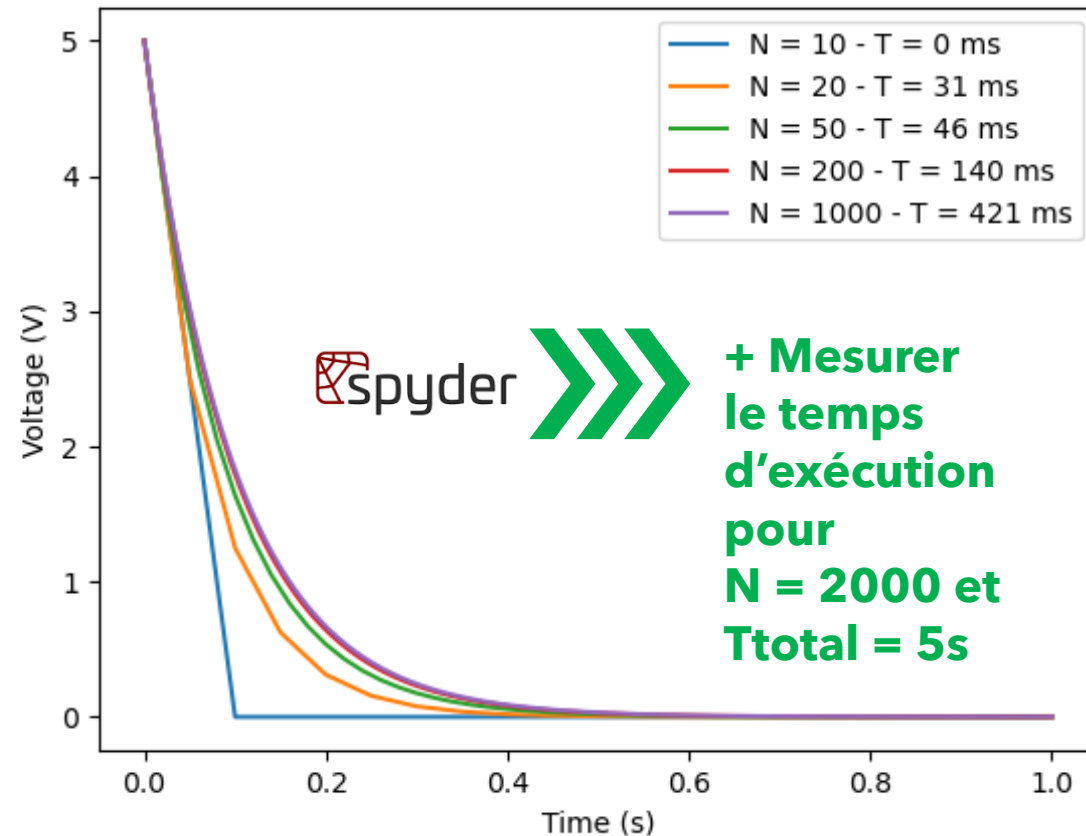
- Temps de calcul vs N

```
import time
```

```
st = time.process_time()  
[ bloc d'instructions à évaluer ]  
et = time.process_time()
```

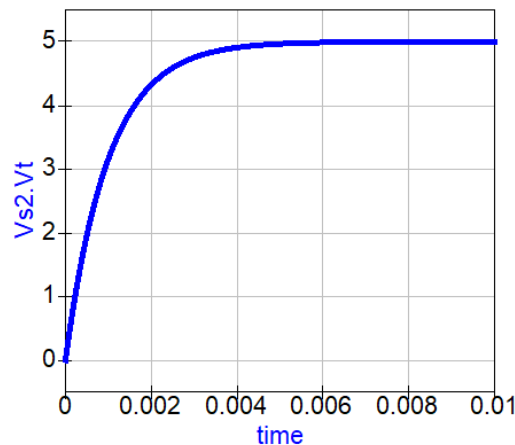
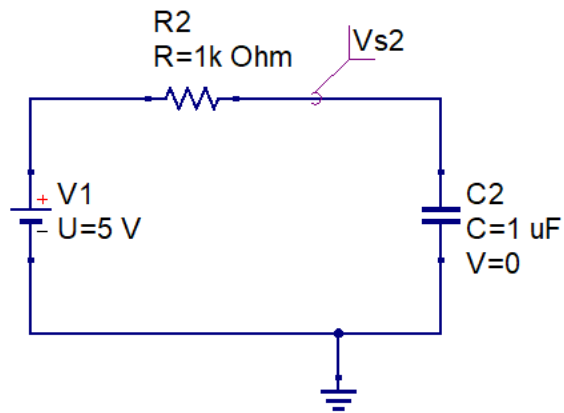
```
vt = int((et-st) * 1e3)  
print(f'Execution Time = {vt} ms')
```

Discharge of a capacitor into a resistance - Explicit Euler Method

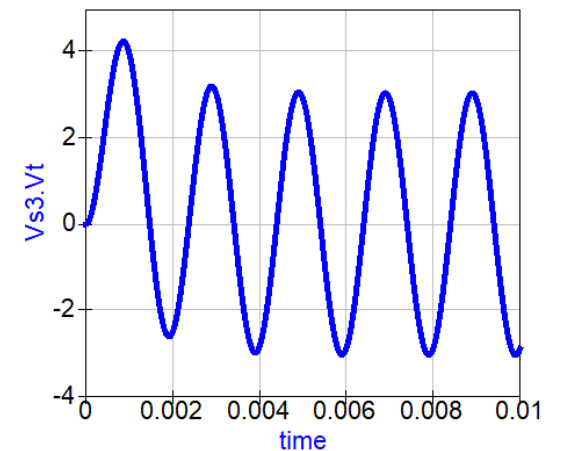
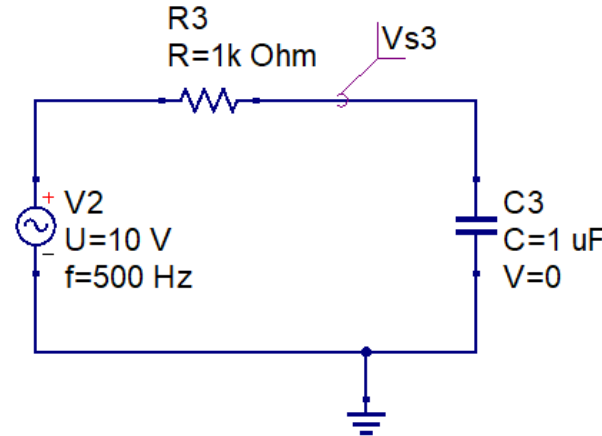


# Circuits similaires / Généralisation

- Réponse à un échelon



- Régime forcé



$$\frac{dV_s}{dt} = -\frac{1}{R_1 \cdot C_1} \cdot (V_s - V_e)$$

# Implémentation de la méthode d'Euler

## OPTIMISATION

- Limite

$F(V_S(t), t, R, C, \mathbf{Ve}(t))$  : fonction qui définit l'équation différentielle

$V_S(t_0)$  : condition initiale

$N$  et  $T_{total}$  : le nombre de points souhaités et le temps total

*Comment rendre paramétrable  $R$  et  $C$  ?*

*Comment rendre dépendant  $F$  de  $\mathbf{Ve}(t)$  ?*



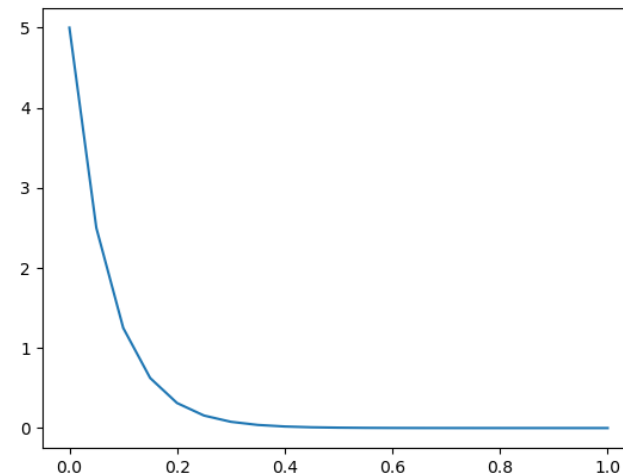
# Implémentation de la méthode d'Euler

## OPTIMISATION

- Définition d'une classe ***F***

```
class F:  
    def __init__(self, R, C, Vs0):  
        self.C = C  
        self.R = R  
        self.Vs0 = Vs0  
  
    def __call__(self, t, Vs):  
        return -Vs/(self.R*self.C)
```

```
newF = F(1e5, 1e-6, 5)  
t, Vs = explicit_euler(newF, F.Vs0, 1, 1000)
```



# Implémentation de la méthode d'Euler

ALLER PLUS LOIN

OPTIMISATION

- Définition d'une classe ***F***

```
class F:
    def __init__(self, R, C, Vs0, Ve):
        self.C = C
        self.R = R
        self.Vs0 = Vs0
        self.Ve = Ve

    def __call__(self, t, Vs):
        return (self.Ve - Vs) / (self.R * self.C)
```

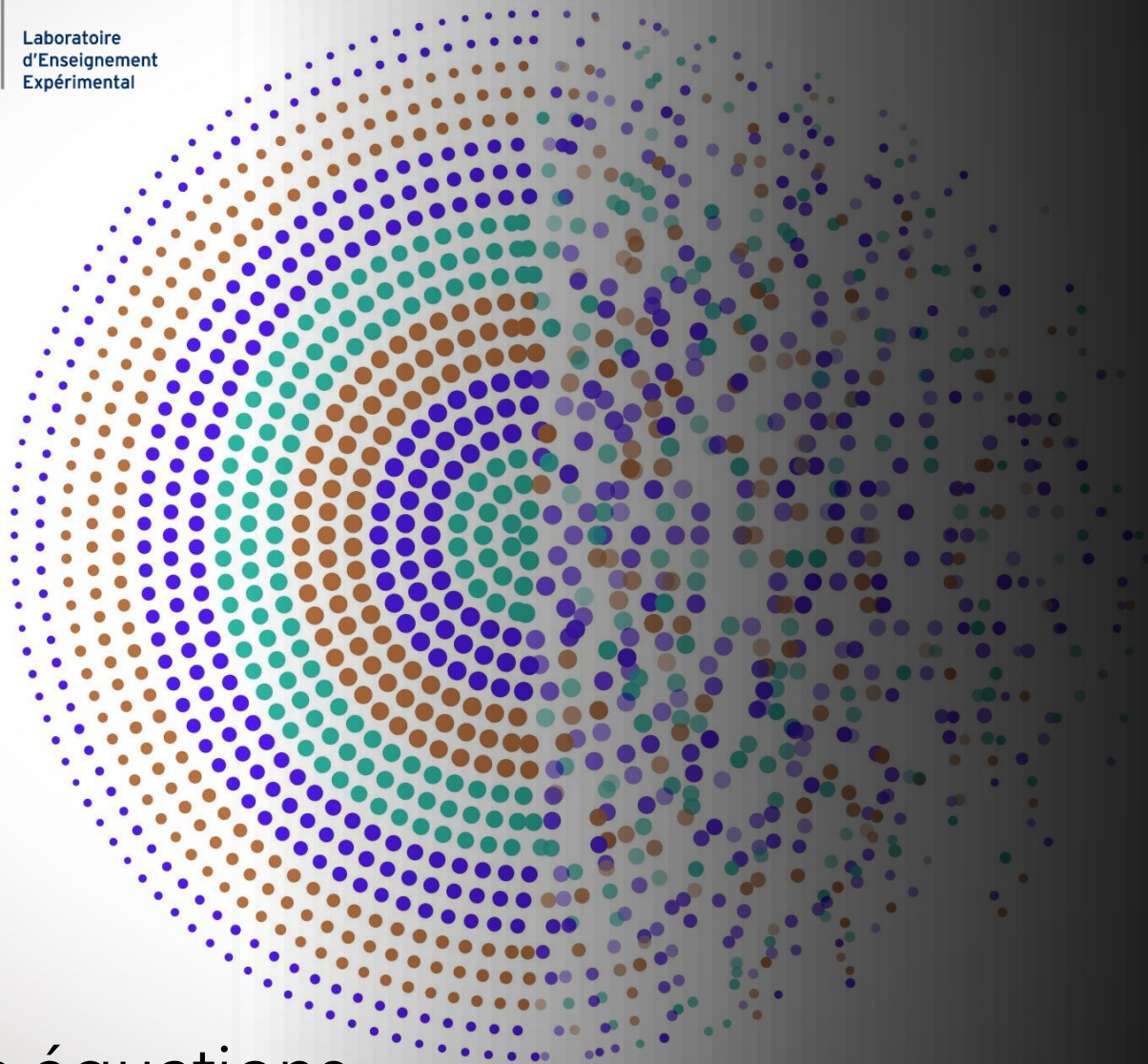
```
newF = F(1e5, 1e-6, 0, ??)
t, Vs = explicit_euler(newF, F.Vs0, 1, 1000)
```



+ Définir une nouvelle classe ***F\_sin***, ayant pour paramètre une amplitude et une fréquence  
+ Tracer l'évolution d'un régime forcé pour une entrée sinusoïdale



$$\frac{dV_s}{dt} = -\frac{1}{R_1.C_1} \cdot (V_s - V_e)$$



Cas des équations  
différentielles

# Intégration Numérique (Scipy)

Outils Numériques / Semestre 5  
/ Institut d'Optique / B1\_4

# Scientific Python / SciPy



- Boite à outils pour les sciences

## SciPy User Guide

SciPy

Introduction

Special functions (`scipy.special`)

Integration (`scipy.integrate`)

Optimization (`scipy.optimize`)

Interpolation (`scipy.interpolate`)

Fourier Transforms (`scipy.fft`)

Signal Processing (`scipy.signal`)

Linear Algebra (`scipy.linalg`)

Sparse eigenvalue problems with ARPACK

Compressed Sparse Graph Routines (`scipy.sparse.csgraph`)

Spatial data structures and algorithms (`scipy.spatial`)

Statistics (`scipy.stats`)

Multidimensional image processing (`scipy.ndimage`)

File IO (`scipy.io`)

# Intégration Numérique / SciPy

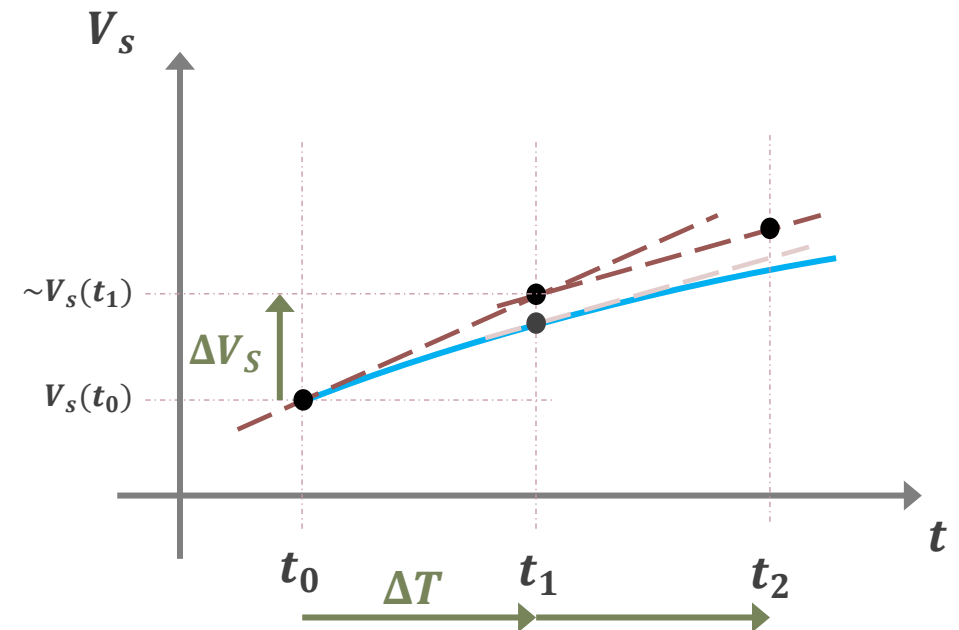


- Intégration Numérique

Calculer des **valeurs approchées**  
d'une fonction  **$y(t)$**

sur un intervalle de  $[t_0, n \cdot \Delta T]$  de  **$t$**

en connaissant sa **dérivée** et un  
point particulier de la fonction



# Intégration Numérique / SciPy

**S'ENTRAINER**

- Résolution par intégration



+ Comparer la méthode d'Euler explicite et la méthode `solve_ivp` implémentée dans `Scipy.integrate`

- sur la décharge d'un condensateur
- sur le régime forcé d'un filtre RC avec une entrée sinusoïdale

```
scipy.integrate.solve_ivp( F , [t0, Ttotal] , [ VS(t0) ] )
```

*Cette fonction retourne des données encapsulées sous la forme de deux vecteurs : **t** et **y***  
***y** doit être transposé pour pouvoir être affiché par rapport à **t***

# D'autres méthodes plus optimisées

- Méthodes de Runge-Kutta

**Méthode d'Euler** → tangente en un point pour trouver le suivant

## **Méthodes de Runge Kutta**

(ordre 2) → création d'un point intermédiaire entre deux points expérimentaux

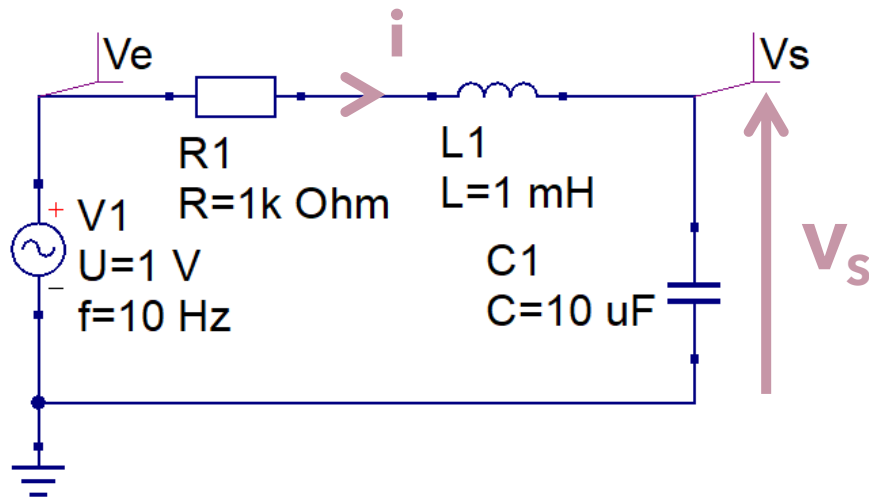
(ordre 4) → création de 3 points intermédiaires

```
scipy.integrate.solve_ivp( ... , method='RK23')
```

```
scipy.integrate.solve_ivp( ... , method='RK45')
```

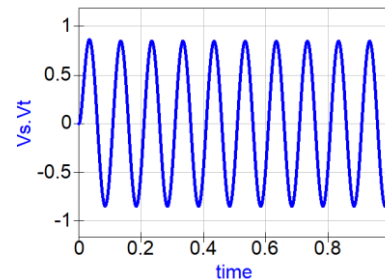
# Autre cas / Equation du second ordre

- Circuit RLC



$$V_e = L_1 \cdot C_1 \cdot \frac{d^2 V_s}{dt^2} + R_1 \cdot C_1 \cdot \frac{d V_s}{dt} + V_s$$

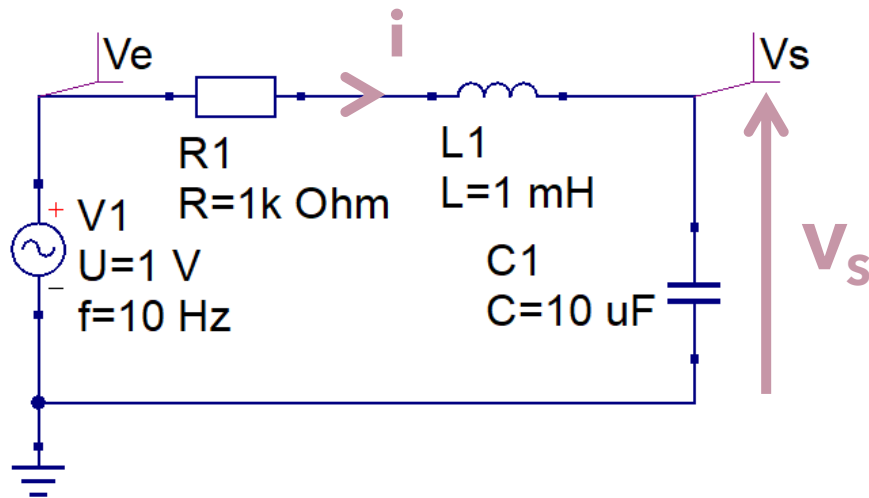
*Equation différentielle d'ordre 2*





# Autre cas / Equation du second ordre

- Circuit RLC

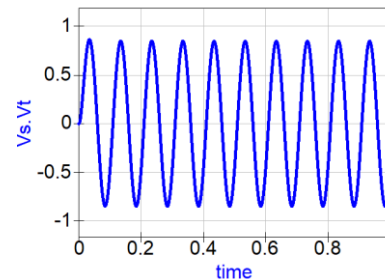


$$V_e = L_1 \cdot C_1 \cdot \frac{d^2 V_s}{dt^2} + R_1 \cdot C_1 \cdot \frac{d V_s}{dt} + V_s$$

Equation différentielle d'ordre 2



**solve\_ivp** uniquement pour **ordre 1**



# Cas d'une équation différentielle d'ordre 2

- Approche analytique



$$V_e = L_1 \cdot C_1 \cdot \frac{d^2 V_s}{dt^2} + R_1 \cdot C_1 \cdot \frac{d V_s}{dt} + V_s$$



$$\frac{d^2 V_s}{dt^2} = -\frac{R_1}{L_1} \cdot \frac{d V_s}{dt} - \frac{1}{L_1 \cdot C_1} (V_s - V_e)$$

*Système de deux équations  
différentielles d'ordre 1*

$$\frac{d V_s}{dt} = u$$

$$\frac{du}{dt} = -\frac{R_1}{L_1} \cdot u - \frac{1}{L_1 \cdot C_1} (V_s - V_e)$$

# Cas d'une équation différentielle d'ordre 2

- Méthode d'Euler  
forme matricielle

Posons :

$$\mathbf{y} = \begin{pmatrix} V_s \\ u \end{pmatrix} \rightarrow \frac{d\mathbf{y}}{dt} = \begin{pmatrix} u \\ F(t, u) \end{pmatrix} = \mathbf{F}(t, \mathbf{y}(t))$$

Ainsi :

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta T \cdot \mathbf{F}(t_n, \mathbf{y}_n)$$

Systeme de deux équations  
différentielles d'ordre 1

$$F(t, u) = -\frac{R_1}{L_1} \cdot u - \frac{1}{L_1 \cdot C_1} (V_s - V_e)$$

$$\begin{aligned} \frac{dV_s}{dt} &= u \\ \frac{du}{dt} &= F(t, u) \end{aligned}$$

# Cas d'une équation différentielle d'ordre 2

- Mise en équation

```
class RLC_forced:
    def __init__(self, R, L, C, Ve):
        [...]
    def __call__(self, t, y):
        Vs = y[0]
        u = y[1]
        return [u, -R/L*u - (Vs - Ve(t)) / (L*C)]
```

$$\rightarrow \mathbf{y} = \begin{pmatrix} V_s \\ u \end{pmatrix} \quad \frac{d\mathbf{y}}{dt} = \begin{pmatrix} u \\ F(t, u) \end{pmatrix} = F(t, \mathbf{y}(t))$$

$$F(t, u) = -\frac{R_1}{L_1} \cdot u - \frac{1}{L_1 \cdot C_1} (V_s - V_e)$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta T \cdot F(t_n, \mathbf{y}_n)$$

Vecteur  
 $y[0] = V_s$   
 $y[1] = u = V_s'$

# Intégration Numérique / SciPy

ALLER PLUS LOIN

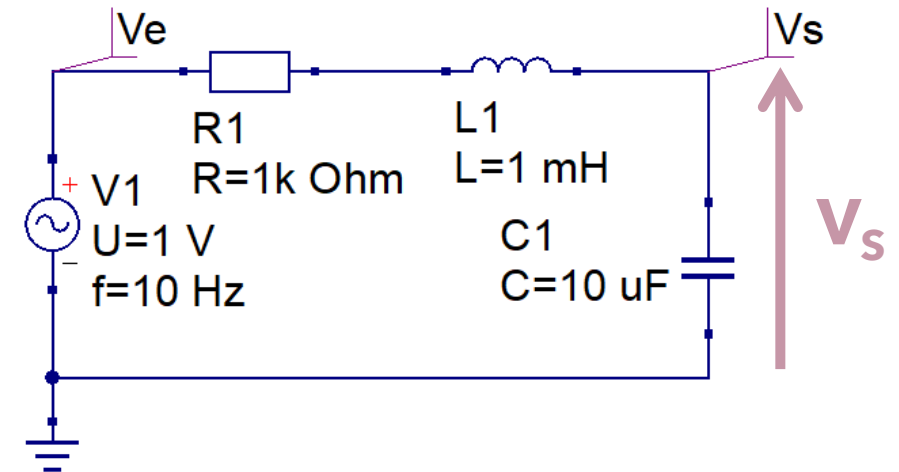
- Résolution par intégration



spyder



+ Tracer l'évolution de  $V_S(t)$  dans un circuit RLC soumis à une tension sinusoïdale de fréquence 20 Hz, avec  $R = 1 \text{ k}\Omega$ ,  $L = 1 \text{ mH}$  et  $C = 10 \text{ }\mu\text{F}$



```
scipy.integrate.solve_ipc(  $F$  ,  $[t_0, T_{total}]$  ,  $[V_S(t_0), Vs'(t_0)]$  )
```

Cette fonction retourne des données encapsulées sous la forme de deux vecteurs :  $\mathbf{t}$  et  $\mathbf{y}$   
 $\mathbf{y}$  doit être transposé pour pouvoir être affiché par rapport à  $\mathbf{t}$

# Bibliographie

- **Intégration des équations différentielles : méthode d'Euler** – Frédéric LEGRAND  
<https://www.f-legrand.fr/scidoc/srcdoc/numerique/euler/eulers/eulers-pdf.pdf>
- **Introduction à la méthode d'Euler en python** – Physique TSI1 Troyes  
<https://www.youtube.com/watch?v=-d7qrNkPDtQ>
- **Cours d'introduction à l'analyse numérique** – Femto-Physique  
<https://femto-physique.fr/analyse-numerique/euler.php>