Capacité numérique 1

Les méthodes d'Euler

Application au filtrage passe-bas

I- Méthode d'Euler.

A - Méthode d'Euler explicite

Les méthodes d'Euler sont des procédures algorithmiques de résolution des problèmes de Cauchy, lorsque l'équation différentielle est d'ordre 1.

Ces méthodes permettent de réaliser facilement un code numérique afin de trouver une solution approchée du problème. La précision de la solution étant fonction unique paramètre.
On distingue deux types de méthode d'Euler :

- la méthode explicite
- la méthode implicite
  On expose le principe des méthodes d'Euler explicite et implicite, puis on appliquera ces méthodes à la résolution d'un problème de Cauchy décrivant l'évolution de la tension aux bornes d'un condensateur. Ces soltuions seront comparées à la solution réelle et à un traitement harmonique.

Considérons le problème suivant :

$$\begin{cases} y'(t) = f(t, y(t)) \\ y(t=0) = y_0 \end{cases}$$

Les méthodes d'Euler ne permettront pas d'obtenir l'expression de $y(t)$ mais d'avoir une approximation du graphe de cette fonction sur un intervalle $[0, T]$.

Commençons en tout premier lieu par intégrer $y'(t)$:

$$y(t) = \int f(t, y(t)dt$$

Pour calculer cette intégrale on subdivise l'intervalle $[0, T]$ en N sous intervalles : $[t_0, t_1]$; $[t_1, t_2]$; ... ; $[t_{N-1}, t_N]$.
Avec $t_0 = 0$ et $t_N = T$. On ferra également attention à ce que $t_{k+1} - t_k = h$ soit une constante.

Vient alors que :

$$y(t_{k+1}) - y(t_k) = \int_{t_k}^{t_{k+1}} f(t, y(t))dt$$

La méthode **d'Euler explicite** revient à utiliser la méthode des rectangles à gauche pour estimer cette intégrale. Détaillons :

Euler_exp.png

Considérons le graphe de $f(t, y(t))$. L'intégrale précédente revient à calculer l'air en bleue. La méthode des rectangles à gauche revient à calculer l'air achurée en vert.
Alors on a l'approximation :

$$y(t_{k+1}) - y(t_k) \approx h.\, f(t_k, y(t_k))$$

Connaissant $y(t_0)$ on peut déduire à partir de cette expression :

$$y(t_1) = y(t_0) + h.\, f(t_0, y(t_0))$$

Et ainsi de suite sur tout l'intervalle $[0, T]$.
On remarque alors que les erreurs successives se cumulent.

B - Méthode d'Euler implicite.

La méthode **d'Euler implicite** revient à utiliser la méthode des rectangles à droite pour estimer cette intégrale. Détaillons :

Euler_imp.png

Cette fois ci on fait l'approximation suivante :

$$y(t_{k+1}) - y(t_k) = h.\, f(t_{k+1}, y(t_{k+1})$$

Si la méthode d'Euler explicite sous-estime légèrement l'intégrale, la méthode d'Euler implicite la sur-estime. On remarque également que pour une équation différentielle linéaire il suffit d'appliquer un décalage d'indice pour passer d'une méthode à l'autre.

C - Mise en oeuvre. Filtrage linéaire passe-bas.

Considérons l'équation différentielle descriptive de l'évolution du système :

$$\frac{ds}{dt} + \frac{s}{\tau} = \frac{H_0.\, e}{\tau}$$

avec $\tau = \frac{1}{\omega_c} = \frac{1}{2\pi.f_c}$

Si en plus on adjoint la condition initiale : $s(t=0) = 0$ alors le problème suivant, appelé **problème de Cauchy**, possède une unique solution.

$$\begin{cases} \frac{ds}{dt} + \frac{s}{\tau} = \frac{H_0.\, e}{\tau} \\ s(t=0) = 0 \end{cases}$$

On connait déjà la solution de ce probème de Cauchy. Supposons que $e(t)$ soit un échelon de tension :

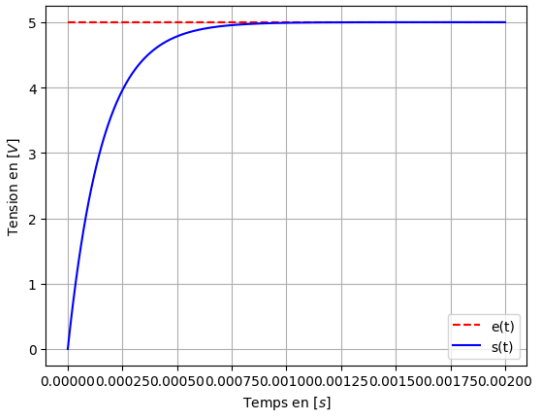
$$\begin{cases} e(t) = 0 \text{ si } t < 0 \\ e(t) = E \text{ si } t \geq 0 \end{cases}$$

$s(t) = E(1 - exp(-t/\tau))$

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

```
H0 = 1.0
fc = 1000.0
```

```
tmin = 0.0
tmax = 0.002

tau = 1/(2*np.pi*fc)

t = np.linspace(tmin,tmax,1000)

E = 5.0

s = E*(1 - np.exp(-t/tau))

plt.figure()
plt.plot([tmin,tmax],[E,E],'r--',label='e(t)')
plt.plot(t,s,'b-',label='s(t)')
plt.grid(True)
plt.xlabel('Temps en $[s]$')
plt.ylabel('Tension en $[V]$')
plt.legend(loc = 'best')
plt.show()
plt.close()
```



La méthode d'Euler explicite est un algorithme permettant de résoudre une équation différentiel d'ordre 1 en exploitant la définition de la dérivée d'une fonction en un point.
En un point $t = a$:

$$\frac{ds}{dt}(t = a) = \lim_{dt \to 0} \frac{s(a + dt) - s(a)}{dt}$$

A la fin on veut pouvoir tracer le graphe de $s(t)$ fonction de t. Identifions alors la nature numérique des différents objets :

1. $s(t)$ est une liste de `float` dont les éléments sont notés `s[i]` .
2. t est également une liste de `float` dont les éléments sont noté `t[i]` .
3. dt est un `float` choisi et tel que `dt = t[i+1] - t[i]` .
4. $e(t)$ est une liste de `float` dont les éléments sont notés `e[i]` .
5. τ et H_0 sont des `float` .

L'algorithme d'Euler explicite consiste alors à donner une écriture approché de l'équation différentielle sous la forme :

$$\frac{s[i + 1] - s[i]}{dt} + \frac{s[i]}{\tau} = \frac{H_0 . e[i]}{\tau}$$

Numériquement cette approximation est d'autant plus fidèle que dt est petit. Cependant cela augmente la durée du calcul numérique.

Revenons à l'équation différentielle. Celle-ci peut s'écrire sous la forme :

$$\frac{ds}{dt} = \frac{1}{\tau}(H_0 . e - s) = f(e, s)$$

1. Ecrire une fonction `Ordre_un` prenant en arguement deux listes et deux réels et retournant la fonction $f(e, s)$ définie ci-dessus.

```
def Ordre_un(e,s,H0,tau):
    return(1/tau*(H0*e - s))
```

2. Ecrire une fonction `Euler_exp` prenant en argument deux listes, ainsi que l'ensemble des paramètres permettant de mettre en oeuvre l'algorithme d'Euler explicite et retournant une liste de date et `s[i]` . Vous utiliserez une boucle `while` pour programmer la méthode d'Euler.

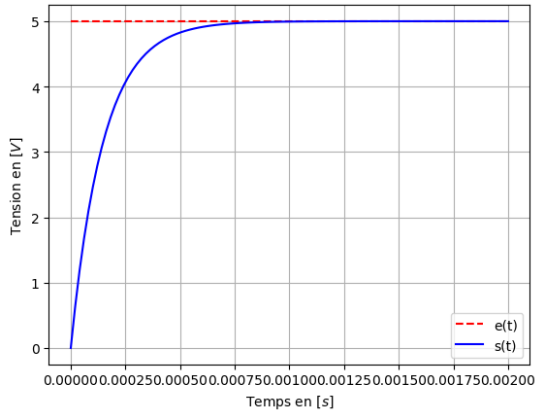
```
def Euler_exp(e,s0,tau,H0,tmin,tmax,N):
    dt = (tmax-tmin)/N
    t = [0]
    s = [s0]
    i = 0
    while i<N:
        s.append(dt*Ordre_un(e[i],s[i],H0,tau) + s[i])
        t.append(t[i] + dt)
        i+=1
    return(t,s)
```

3. Créer une liste `e[i]` contenant 100 valeurs et permettant de représenter $e(t)$ sous forme d'un échelon de tension $E = 5,0\ V$.

```
E = 5.0
N = 100
e = np.linspace(E,E,N)
```

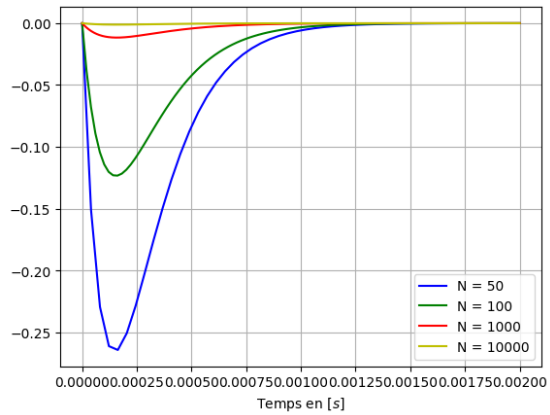
4. Ecrire un code permettant de représenter sur un même graphe $e(t)$ et $s(t)$ déterminé par la méthode d'Euler pour `N = 100`.

```
t,s = Euler_exp(e,0,tau,H0,tmin,tmax,N-1)

plt.figure()
plt.plot(t,e,'r--',label='e(t)')
plt.plot(t,s,'b-',label='s(t)')
plt.grid(True)
plt.xlabel('Temps en $[s]$')
plt.ylabel('Tension en $[V]$')
plt.legend(loc='best')
plt.show()
plt.close()
```



5. Comparer la fidélité de la méthode d'Euler pour `N = 50`; `N = 100`; `N = 1000` et `N = 10 000`.

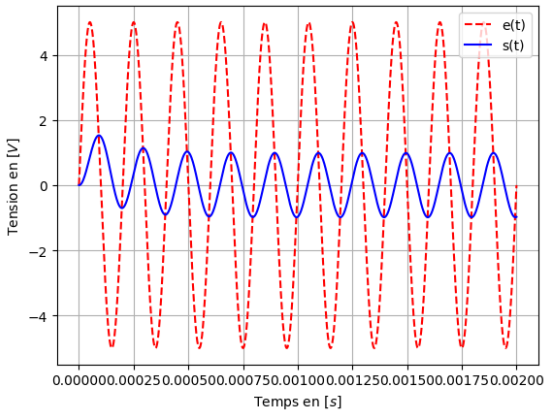
```
Diff = []
T = []
for N in [50,100,1000,10000]:
    e = np.linspace(E,E,N)
    t,s = Euler_exp(e,0,tau,H0,tmin,tmax,N-1)
    t_analog = np.linspace(tmin,tmax,N)
    s_analog = E*(1 - np.exp(-t_analog/tau))
    T.append(t)
    Diff.append(s_analog - s)

plt.figure()
plt.plot(T[0],Diff[0],'b-',label='N = 50')
plt.plot(T[1],Diff[1],'g-',label='N = 100')
plt.plot(T[2],Diff[2],'r-',label='N = 1000')
plt.plot(T[3],Diff[3],'y-',label='N = 10000')
plt.grid(True)
plt.xlabel('Temps en $[s]$')
plt.legend(loc='best')
plt.show()
plt.close()
```



6. Reprendre toute l'étude précédente pour $e(t) = 5,0.\sin(2\pi ft)$ avec $f = 5000\ Hz$.

```
N = 1000
t1 = np.linspace(tmin,tmax,N)
```

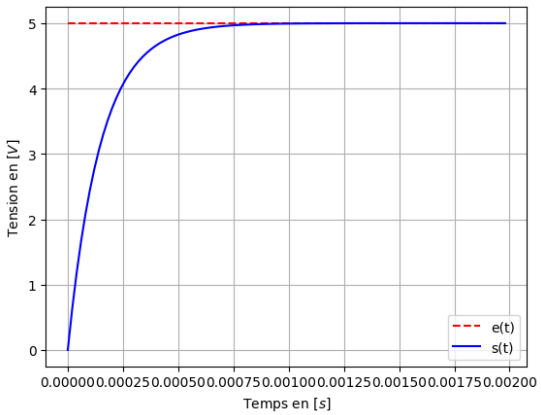
```
e = 5.0*np.sin(2*np.pi*5000*t1)

t,s = Euler_exp(e,0,tau,H0,tmin,tmax,N-1)

plt.figure()
plt.plot(t,e,'r--',label='e(t)')
plt.plot(t,s,'b-',label='s(t)')
plt.grid(True)
plt.xlabel('Temps en $[s]$')
plt.ylabel('Tension en $[V]$')
plt.legend(loc='best')
plt.show()
plt.close()
```



7. Ecrire une fonction `Euler_imp` prenant en argument deux listes, ainsi que l'ensemble des paramètres permettant de mettre en oeuvre l'algorithme d'Euler implicite et retournant une liste de date et `s[i]` . Vous utiliserez une boucle `while` pour programmer la méthode d'Euler.

```
def Euler_imp(e,tau,H0,tmin,tmax,N):
    dt = (tmax-tmin)/N
    t = [0]
    s = np.zeros(N)
    for i in range(0,N-1):
        s[i+1] = s[i] + dt*(1/tau*H0*e[i] - s[i]/tau)
        t.append(t[i] + dt)
    return(t,s)
```

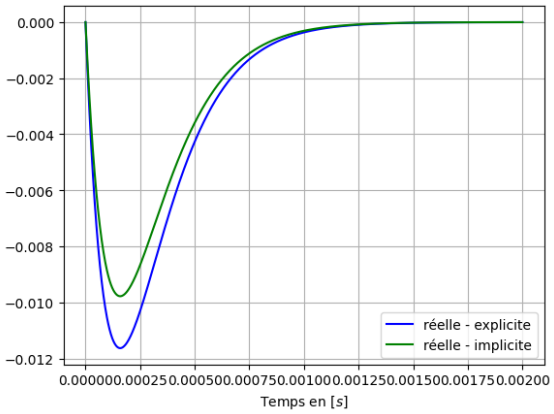
```
E = 5.0
N = 100
e = np.linspace(E,E,N)


t_imp,s_imp = Euler_imp(e,tau,H0,tmin,tmax,N)

plt.figure()
plt.plot(t_imp,e,'r--',label='e(t)')
plt.plot(t_imp,s_imp,'b-',label='s(t)')
plt.grid(True)
plt.xlabel('Temps en $[s]$')
plt.ylabel('Tension en $[V]$')
plt.legend(loc='best')
plt.show()
plt.close()
```



8. Ecrire un algorithme permettant de comparer pour un même nombre de point d'intégration `N` la méthode d'Euler implicite, explicite et la solution reélle.

```
N = 1000

e = np.linspace(E,E,N)
t_analog = np.linspace(tmin,tmax,N)

t,s = Euler_exp(e,0,tau,H0,tmin,tmax,N-1)
```

```
t_imp,s_imp = Euler_imp(e,tau,H0,tmin,tmax,N)
s_analog = E*(1 - np.exp(-t_analog/tau))

plt.figure()
plt.plot(t,s_analog-s,'b-',label='réelle - explicite')
plt.plot(t,s_analog-s_imp,'g-',label='réelle - implicite')
plt.grid(True)
plt.xlabel('Temps en $[s]$')
plt.legend(loc='best')
plt.show()
plt.close()
```



II- Application au filtrage passe-bas.

A- Etude de la fonction de transfert

Un filtre passe-bas peut être représenter par sa fonction de transfert harmonique :

$$\underline{H} = \frac{H_0}{1 + j\frac{\omega}{\omega_c}}$$

On propose dans un premier temps de réaliser un petit programme permettant de tracer le diagramme de Bode d'un tel filtre pour :
$f_c = 1000\ Hz$ et $H_0 = 1$

7. Construire une liste de $N = 100000$ valeurs de fréquences f comprises entre $f_{min} = 10\ Hz$ et $f_{max} = 100000\ Hz$.

```
N = 100000
fmin = 10.0
fmax = 100000.0
f = np.linspace(fmin,fmax,N)
```

8. Utiliser la liste `f=np.linspace(fmin,fmax,N)` ainsi créée pour construire la liste des valeurs complexes de $\underline{H}$. On rappelle que l'imaginaire pur s'écrit `1j` .

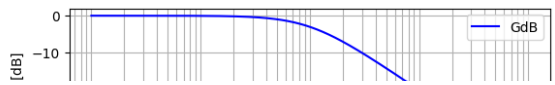
```
H0 = 1.0
fc = 1000.0
Hbar = H0/(1.0+1j*f/fc)
```

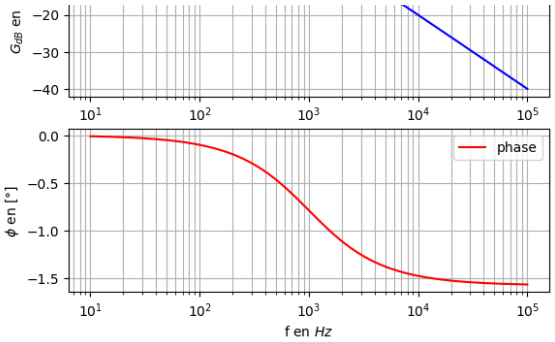
9. Utiliser la fonctions `np.abs()` pour calculer la liste des modules de la fonction de transfert. Utiliser les fonctions trigonométrique du module `numpy` pour calculer la liste des phases.

```
H = np.abs(Hbar)
phi = -np.arctan(f/fc)
```

10. En vous appuyant sur le code suivant réaliser une représentation graphique du diagramme de Bode du filtre passe-bas définit plus haut. Vous commenterez chaque ligne du code suivant en indiquant leur utilité.

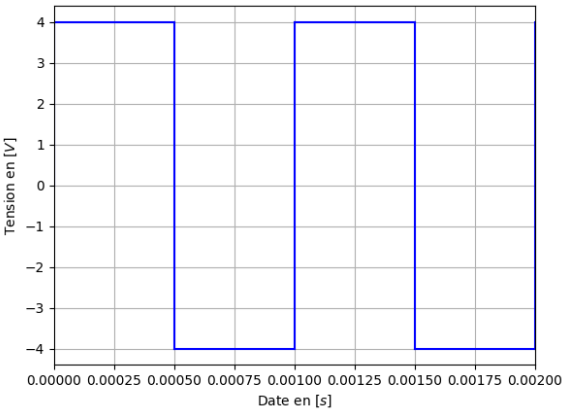
```
plt.subplot(211)
plt.plot(f,20.0*np.log10(H),'b-',label='GdB')
plt.xscale('log')
plt.grid(True,'both')
plt.legend(loc='best')
plt.xlabel('f en $Hz$')
plt.ylabel('$G_{dB}$ en [dB]')

plt.subplot(212)
plt.plot(f,phi,'r-',label='phase')
plt.xscale('log')
plt.grid(True,'both')
plt.legend(loc='best')
plt.xlabel('f en $Hz$')
plt.ylabel('$\phi$ en $[°]$')
plt.show()
plt.close()
```





On souhaite maintenant étudier numériquement l'effet de ce filtre sur une tension créneau périodique. Pour cela on donne une représentation de cette tension.

```
Emax = 4.0
Emin = -4.0
T2 = 1.0e-3/2.0

plt.figure()
plt.plot([0.0,T2],[Emax,Emax],'b-')
plt.plot([T2,2*T2],[Emin,Emin],'b-')
plt.plot([2*T2,3*T2],[Emax,Emax],'b-')
plt.plot([3*T2,4*T2],[Emin,Emin],'b-')
plt.plot([0.0,0.0],[0.0,Emax],'b-')
plt.plot([T2,T2],[Emax,Emin],'b-')
plt.plot([2*T2,2*T2],[Emin,Emax],'b-')
plt.plot([3*T2,3*T2],[Emax,Emin],'b-')
plt.plot([4*T2,4*T2],[Emin,Emax],'b-')
plt.grid(True)
plt.ylabel('Tension en $[V]$')
plt.xlabel('Date en $[s]$')
plt.xlim(0.0,2.0e-3)
plt.show()
plt.close()
```

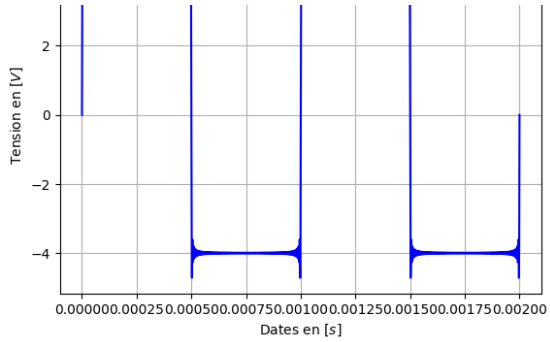


La décomposition en série de Fourier d'une tension périodique impaire créneau d'amplitude E et de fréquence f_0 est donnée par :

$$e(t) = \frac{4.E}{\pi} \sum_{k=0}^{\infty} \frac{1}{2k+1} \sin(2.\pi.(2k+1).f_0.t)$$

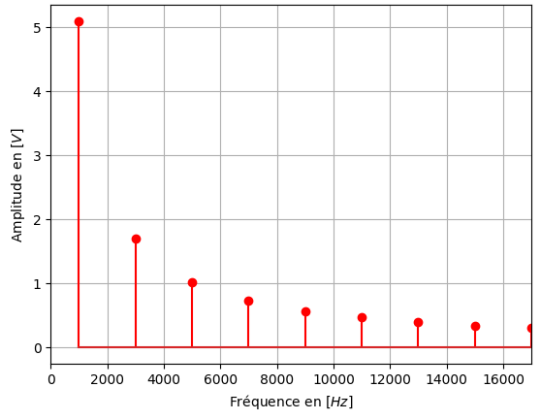
11. A l'aide d'un programme python reconstruire par supperposition des 100 premières harmoniques, une tension créneau comme celle représentée sur le chronogramme précédent. On utilisera une boule `for` pour créer une listte des 100 premières harmoniques et la fonction `np.sum` pour réaliser la supperposition.

```
M = 100
f0 = 1000.0
tmin = 0.0
tmax = 2.0e-3
t = np.linspace(tmin,tmax,100000)
harmo_e = []

for i in range(M):
    harmo_e.append(4*Emax/(np.pi*(2*i+1))*np.sin(2*np.pi*(2*i+1)*f0*t))

e = np.sum(harmo_e,axis = 0)

plt.figure()
plt.plot(t,e,'b-')
plt.grid(True)
plt.xlabel('Dates en $[s]$')
plt.ylabel('Tension en $[V]$')
plt.show()
plt.close()
```



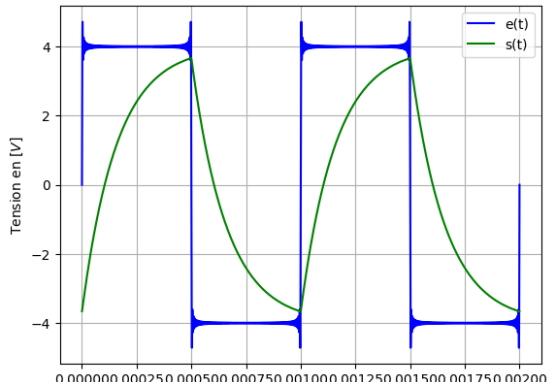
12. Représenter le spectre de cette tension $e(t)$ en fréquence pour les 9 premières harmoniques.

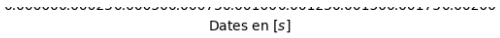
```
fi = []
ei = []
for i in range(M):
    fi.append((2*i+1)*f0)
    ei.append(4*Emax/(np.pi*(2*i+1)))

plt.figure()
plt.stem(fi,ei,'r-')
plt.grid(True)
plt.xlim(0,17*f0)
plt.xlabel('Fréquence en $[Hz]$')
plt.ylabel('Amplitude en $[V]$')
plt.show()
plt.close()
```



13. En utilisant les listes des harmoniques que vous venez de créer, écrire une routine python permettant de mettre en évidance l'effet du filtre passe-bas précédent sur la tension créneau étudiée.

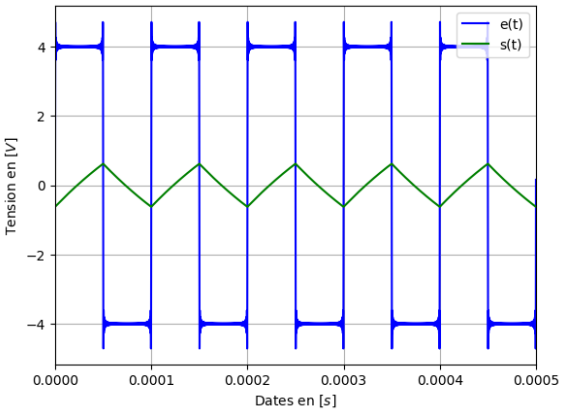
```
harmo_s = []

for i in range(M):
    harmo_s.append(4*Emax/(np.pi*(2*i+1))*np.abs(H0/(1.0+1j*(2*i+1)*f0/fc))*np.sin(2*np.pi*(2*i+1)*f0*t-np.arctan((2*i+1)*f0/fc)))

s = np.sum(harmo_s,axis = 0)

plt.figure()
plt.plot(t,e,'b-',label='e(t)')
plt.plot(t,s,'g-',label='s(t)')
plt.grid(True)
plt.xlabel('Dates en $[s]$')
plt.ylabel('Tension en $[V]$')
plt.legend(loc='best')
plt.show()
plt.close()
```





14. Par modification d'un paramètre, montrer qu'à "haute-fréquence" le filtre passe-bas se comporte comme un intégrateur. Représenter alors le spectre de la tension de sortie.

```
f0 = 10000.0

harmo_e = []

for i in range(M):
    harmo_e.append(4*Emax/(np.pi*(2*i+1))*np.sin(2*np.pi*(2*i+1)*f0*t))

e = np.sum(harmo_e,axis = 0)

harmo_s = []

for i in range(M):
    harmo_s.append(4*Emax/(np.pi*(2*i+1))*np.abs(H0/(1.0+1j*(2*i+1)*f0/fc))*np.sin(2*np.pi*(2*i+1)*f0*t-np.arctan((2*i+1)*f0/fc)))

s = np.sum(harmo_s,axis = 0)

plt.figure()
plt.plot(t,e,'b-',label='e(t)')
plt.plot(t,s,'g-',label='s(t)')
plt.grid(True)
plt.xlim(0,5e-4)
plt.xlabel('Dates en $[s]$')
plt.ylabel('Tension en $[V]$')
plt.legend(loc='best')
plt.show()
plt.close()
```



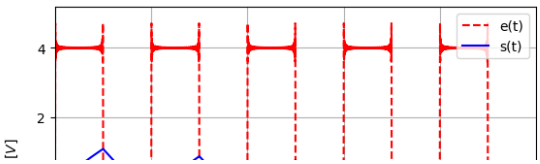
B- Utilisation de l'équation différentielle.

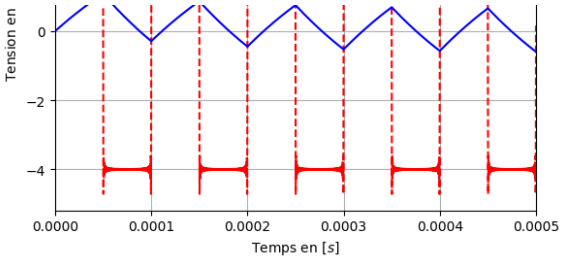
L'étude précédente de la fonction de transfert peut être menée, comme précédemment, à partir de l'équation différentielle. Pour cela il suffit d'utiliser la fonction $e(t)$ définie par synthèse spectrale.

```
f0 = 10000.0

harmo_e1 = []



#N = 1000
#t1 = np.linspace(tmin,tmax,N)
```

```
for i in range(M):
    harmo_e1.append(4*Emax/(np.pi*(2*i+1))*np.sin(2*np.pi*(2*i+1)*f0*t))

e = np.sum(harmo_e1,axis = 0)
```

```
t,s = Euler_exp(e,0,tau,H0,tmin,tmax,N-1)

plt.figure()
plt.plot(t,e,'r--',label='e(t)')
plt.plot(t,s,'b-',label='s(t)')
plt.grid(True)
plt.xlabel('Temps en $[s]$')
plt.ylabel('Tension en $[V]$')
plt.legend(loc='best')
plt.xlim(0,0.0005)
plt.show()
plt.close()
```





Commencez à coder ou à générer avec l'IA.