TD1

1 Méthode d'Euler pour la résolution d'équations différentielles

1 a) Problèmes dits de Cauchy

1 b) Rappel de l'algorithme d'Euler explicite

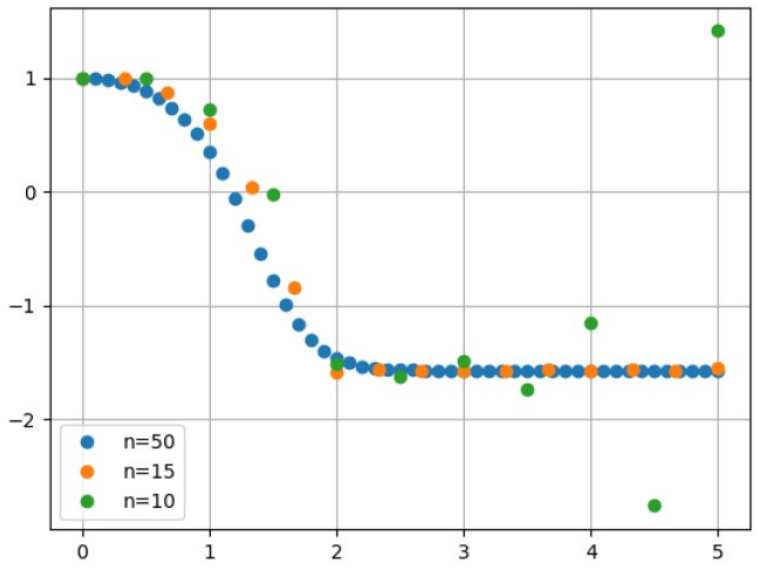
1. Après avoir importé les modules nécessaires, on définit f et `EulerExplicite1( f, t0, tf, n, y0)` :

```
import math

def f(y,t):
        return -2*t*math.cos(y)
def EulerExplicite1(f,t0,tf,n,y0):
        t=[0]*(n+1)
        y=[0]*(n+1)
        t[0]=t0
        y[0]=y0
        h=(tf-t0)/n
        for i in range(0,n):
                t[i+1]=t[i]+h
                y[i+1]=y[i]+h*f(y[i],t[i])
        return t,y
```

2.

```
## Euler explicite

t0=0.0
tf=5.0
y0=1.0

n=50
temps1, solutionEE1=EulerExplicite1(f,t0,tf,n,y0)
n=15

temps2, solutionEE2=EulerExplicite1(f,t0,tf,n,y0)
n=10

temps3, solutionEE3=EulerExplicite1(f,t0,tf,n,y0)

# Plot de la figure

plt.figure()
plt.plot(temps1,solutionEE1,"o",label='n=50')
plt.plot(temps2,solutionEE2,"o",label='n=15')
plt.plot(temps3,solutionEE3,"o",label='n=10')
plt.grid()
plt.legend()
plt.show()
1
```

3.



1 c) Rappel de l'algorithme d'Euler implicite

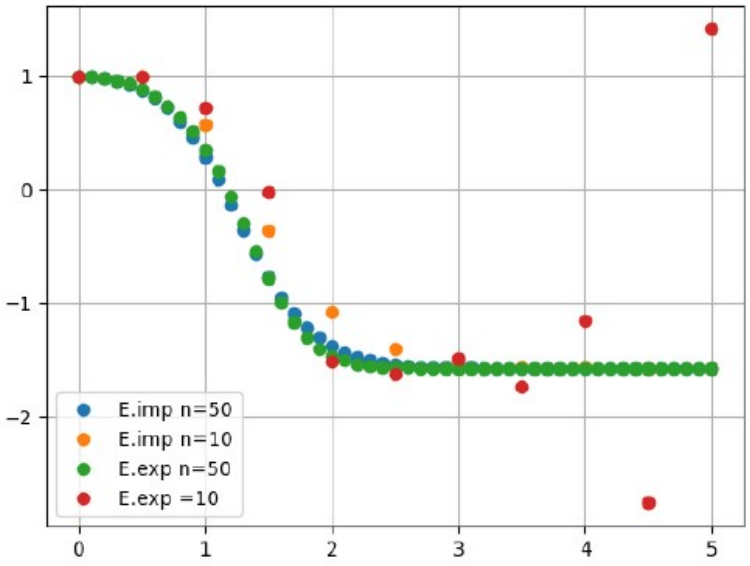
4. Après avoir importé les modules nécessaires, on définit f et `euler_implicite(f,t0,tf,n,y0)` :
   Résolution numérique d'une équation différentielle par la méthode d'Euler implicite.
   - f : Fonction f(y, t) représentant l'équation différentielle $dy/dt = f(y,t)$
   - t0 : Temps initial
   - tf : Temps final
   - n : Nombre de pas
   - y0 : Condition initiale
   - return : Deux listes, liste_t (abscisses) et liste_y (ordonnées)

```
def euler_implicite(f, t0, tf, n, y0):

        # Calcul du pas de temps

        h = (tf - t0) / n

        # Initialisation des listes
        liste_t = [t0 + i * h for i in range(n + 1)]
        liste_y = [0]*(n + 1)
        liste_y[0] = y0

        # Boucle principale d'Euler implicite
        for i in range(n):

        # Fonction g(x) a resoudre pour obtenir yn+1
                g = lambda x: x - liste_y[i] - h * f(x, liste_t[i])

                # Resolution de l'equation par la dichotomie
                liste_y[i+1] = dichotomie(g, liste_y[i] - 5 * h, liste_y[i] + 5 * h,1e-4)
        return liste_t, liste_y
```

5.
```
temps_EI_1 ,sol_EI_1 = euler_implicite (f ,0 ,5 ,50 ,1)
temps_EI_2 ,sol_EI_2 = euler_implicite (f ,0 ,5 ,10 ,1)

plt.figure()
plt.plot(temps_EI_1,sol_EI_1,'o',label='E.imp_n=50')
```

```
plt.plot(temps_EI_2,sol_EI_2,'o',label='E.imp_n=10')
plt.plot(temps1,solutionEE1,"o",label='E.exp_n=50')
plt.plot(temps3,solutionEE3,"o",label='E.exp_=10')
plt.grid()
plt.legend()
plt.show()
```

6.



1 d) Application au comportement dynamique d'un gyroscope

7. L'équation s'écrit $A\ddot{\beta} + B\dot{\beta}' + C\beta = K1.u(t)$ ou s'écrit $\frac{\beta}{\omega_0^2} + \frac{2\zeta}{\omega_0}\beta + \beta = K.u(t)$ Écrivons le problème de Cauchy associé à l'équation différentielle :

$$\frac{1}{\omega_0^2}\ddot{\beta} + \frac{2\xi}{\omega_0}\dot{\beta} + \beta = Ku(t)$$

Pour simplifier le système, nous définissons une nouvelle variable $q(t) = \dot{\beta}(t)$. Nous pouvons alors réécrire l'équation sous forme d'un système de deux équations différentielles du premier ordre :

$$\begin{cases} \dot{\beta}(t) = q(t) \\ \dot{q}(t) = -\omega_0^2\beta(t) - 2\xi\omega_0 q(t) + \omega_0^2 Ku(t) \end{cases}$$

Les conditions initiales sont :

$$\beta(0) = 0, \quad q(0) = 0$$

Ainsi, le problème de Cauchy associé est le système suivant avec les conditions initiales :

$$\begin{cases} \dot{\beta}(t) = q(t) \\ \dot{q}(t) = -\omega_0^2\beta(t) - 2\xi\omega_0 q(t) + \omega_0^2 Ku(t) \end{cases}$$

avec $\beta(0) = 0$ et $q(0) = 0$.

8. Écrivons les deux équations de récurrence permettant de résoudre l'équation différentielle par la méthode d'Euler explicite.

$$\beta_{n+1} = \beta_n + \Delta t \cdot q_n$$
$$q_{n+1} = q_n + \Delta t \cdot \left(-\omega_0^2\beta_n - 2\xi\omega_0 q_n + \omega_0^2 Ku\left(t_n\right)\right)$$

avec $\beta(0) = 0$ et $q(0) = 0$ comme conditions initiales, et $u(t)$ étant une fonction de Heaviside.

9.
```
# t0: temps initial
# beta0: valeur initiale
# q0: vitesse initiale
# T: temps de simulation
# w0: pulsation propre non amorti
# ksi: coef d'amortissement
# K: gain
#

A=6.6E-6
B=1.5E-4
C=(4.4E-6)+(0.0049)
K1=0.0015
w0=(C/A)**0.5
ksi=0.5*w0*B/C
K=K1/C
T=1
t0=0
beta0=0
q0=0
N=500

#
#gyroscope explicite
def gyro_explicite(t0,beta0,q0,N,T,w0,ksi,K):
    h=T/N
    listet=[t0]
    listebeta=[beta0]
    listeomega=[q0]
    for i in range(1,N+1,1):
        listet.append(t0+i*h)
        listebeta.append(listebeta[i-1]+h*listeomega[i-1])
        listeomega.append(listeomega[i-1]+h*w0*(w0*K-2*ksi*listeomega[i-1]-w0*listebeta[i-1]))
    return listet,listebeta

X1,Y1=gyro_explicite(t0,beta0,q0,5000,T,w0,ksi,K)
plt.plot(X1,Y1,label='Solution euler explicite N=5000')
```

10.
```
def second_ordre_exacte(t0,T,w0,ksi,K,N):
    h=T/N
    x=np.linspace(t0,h*N,N)
    y=K*(1-np.exp(-ksi*w0*x)*(np.cos(w0*np.sqrt(1-ksi**2)*x)+(ksi/np.sqrt(1-ksi**2))*np.sin(w0*np.sqr
    return x,y

X2,Y2=second_ordre_exacte(t0,T,w0,ksi,K,1000)
plt.plot(X2,Y2,label='Solution exacte')
```

11. cf. question 3

12. Écrivons les deux équations de récurrence permettant de résoudre l'équation différentielle par la méthode d'Euler implicite.

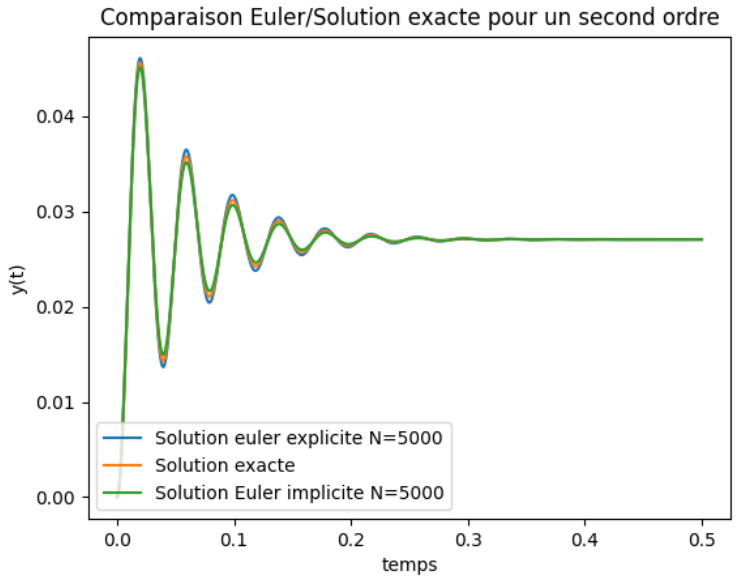
$$\beta_{n+1} = \beta_n + \Delta t \cdot q_{n+1}$$
$$q_{n+1} = q_n + \Delta t \cdot \left(-\omega_0^2 \beta_{n+1} - 2\xi\omega_0 q_{n+1} + \omega_0^2 K u\left(t_{n+1}\right)\right)$$

avec $\beta(0) = 0$ et $q(0) = 0$ comme conditions initiales, et $u(t)$ étant une fonction de Heaviside.
On injecte la première équation dans la seconde afin d'obtenir le résultat suivant :

$$q_{n+1} = \frac{q_n - \Delta t \omega_0^2 \beta_n + \omega_0^2 K u(t_{n+1})}{1 + 2\xi\omega_0 + \Delta t^2 \omega_0^2}$$
$$\beta_{n+1} = \beta_n + \Delta t \cdot q_{n+1}$$

13.
```
def gyro_implicite(t0,beta0,q0,N,T,w0,ksi,K):
    h=T/N
    listet=[t0]
    listebeta=[beta0]
    listeomega=[q0]
    for i in range(1,N+1,1):
        listet.append(t0+i*h)
        listeomega.append((listeomega[i-1]+h*K*w0*w0-h*w0*w0*listebeta[i-1])/(1+2*ksi*w0*h+h*h*w0*w0)
        listebeta.append(listebeta[i-1]+h*listeomega[i])
    return listet,listebeta
```

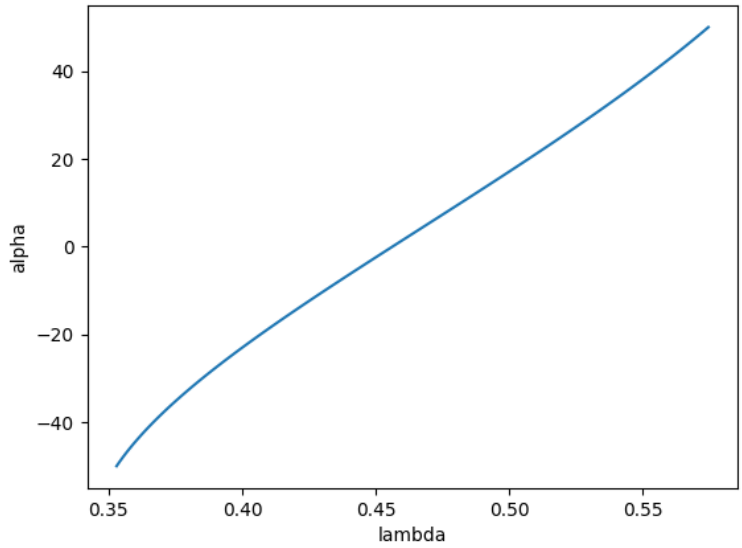
14.
```
X3,Y3=gyro_implicite(t0,beta0,q0,5000,T,w0,ksi,K)
plt.plot(X3,Y3,label='Solution Euler implicite N=5000')
plt.xlabel('temps')
plt.ylabel('y(t)')
plt.title('Comparaison Euler/Solution exacte pour un second ordre')
plt.legend(loc=3)
plt.show()
```



2 Dichotomie et méthode de Newton pour la recherche de zéro

2).1 Approche graphique

15.
```
a = 0.14
b = 0.046
L = 0.49
alpha = np.linspace(-50,50,101)*np.pi/180 #en radian

def lambda1(alpha):
    return np.sqrt((L*np.cos(-130*np.pi/180+alpha)+a)**2 + (L*np.sin(-130*np.pi/180+alpha)-b)**2)
```



16. Graphiquement, pour $\lambda = 0.4$ on lit $\alpha = -23$

2).2 Dichotomie

17.

18.

19.
```
def f(alpha,lambda0=0.4):
    return lambda0 - lambda1(alpha)
#
def fp(alpha):
    return ((a*L*np.sin(-130*np.pi/180+alpha)+b*L*np.cos(-130*np.pi/180+alpha))/lambda1(alpha))
#
# Utilisation de la methode Dichotomie
#
def dichotomie(f,a,b):
    niter=0
    err=[]
    xv=[(a+b)/2]
    while (abs(a-b)>1e-10 and niter < 500):
        if f((a+b)/2)*f(a) < 0 : #donc signe different
            b = (a+b) / 2
        else:
            a = (a+b) / 2
        niter+=1
        err.append(abs(a-b))
        xv.append((a+b)/2)
    return (a+b)/2,niter,err,xv
```

20.
```
tic = time.time()
xd,iterd,errd,xvd = dichotomie(f,-50*np.pi/180,50*np.pi/180)
print('Dichotomie:',xd*180/np.pi,time.time()-tic,iterd)
```

21.
```
def ordre(x):
    ord = []
    for i in range(2,len(x)):
        ord.append(np.log(abs(x[i]-x[i-1]))/np.log(abs(x[i-1]-x[i-2])))
    return ord
```

2).3 Newton

22.
```
def newton(f,fp,xini):
    err=[]
    xv=[]
    x = xini
    xo = 1e8
    niter = 0
    while(abs(f(x)-f(xo))> 1e-10 and niter < 500):
        xo = x
        fp1 = fp(x)
        x = x - f(x) / fp1
        niter+=1
        err.append(abs(x-xo))
        xv.append(x)
    return x,niter,err,xv

tic = time.time()
xn,itern,errn,xvn = newton(f,fp,0)
print('Newton exacte :',xn*180/np.pi,time.time()-tic,itern)
```

23.
```
def newton2(f,xini):
    err=[]
    xv=[]
    x = xini
    xo = 1e8
    h = .0001
    niter = 0
    while(abs(f(x)-f(xo))> 1e-10 and niter < 500):
        xo = x
        fp1 = (f(x+h)-f(x)) / h
        x = x - f(x) / fp1
        niter+=1
        err.append(abs(x-xo))
        xv.append(x)
    return x,niter,err,xv

tic = time.time()
xn2,itern2,errn2,xvn2 = newton2(f,0)
print('Newton approchee :',xn*180/np.pi,time.time()-tic,itern2)
```