Faculté des sciences de Montpellier

Rapport séance 1

.

Étudiant: Giscard Leonel Zouakeu

Partie 1

Résolvons l'équation différentielle suivante:

$$u'(x) = -\lambda u(x) \tag{1}$$
$$u(0) = 1$$

avec λ une constate et $x \in [0, 1]$

une solution de cette équation différentielle est : $u(x) = e^{-\lambda t}$

Résolution de l'équation avec Python pour $\lambda = 1$

On peut écrire(1) de la façon suivante: $y'(t) = f(t, y(t))$ avec $y(t_0) = y_0$. Ce problème est appelé problème de Cauchy.

```
import numpy as np
#import math
#from scipy.integrate import odeint
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
#import scipy
#from scipy.integrate import quad

# ---------------------------------------------------------------------------
# on a par exemple l'equation differentielle du 1ere ordre:
    #-----------------------------------------------------
             # u'(t)+ a.u(t)= b avec a,b des constantes
             # u(t_0)= c avec c une constante
    #-----------------------------------------------------
# pour resoudre cette equation differentielle nous la mettons sous la forme:
    # u'(t)= -a u(t) + b
# ---------------------------------------------------------------------------


# definition de l'equation differentielle du 1ere ordre

def equation (t,u):
    lamb= -1
    return lamb*u


t_initial = 0 # temps initial
t_final =1  # temps final
u_0= 1 # condition initiale
t= np.linspace(t_initial,t_final, 100)

```

```
#solution python scipy

sol_solve_ivp=solve_ivp(equation,[t_initial,t_final],[u_0],
    dense_output=True)

z=sol_solve_ivp.sol(t)
plt.plot(t,z.T)


# solution de l'equation
def U_exact (s,nn):
    u=np.zeros(nn)
    s= np.linspace(t_initial,t_final, nn)
    for i in range(nn):
        u[i]= np.exp(-s[i])
    return u
resultat= U_exact(t,100)


#solution_odeint= odeint(equation,u_0,t)
#plt.plot(t,solution_odeint,label="scipy")
plt.plot(t,resultat)
plt.xlabel("Temps")
plt.ylabel("")
plt.show

# definition de la methode de euleur explicite

def Euleur_explicite(fonction,t_i,t_f,n,u_i):
    y= np.zeros(n+1) # vecteur de y initialise a 0.
    y[0]= u_i # condition initiale
    h= (t_f - t_i)/n # pas du temps
    t1= np.linspace(t_initial,t_final, n+1)
    for i in np.arange(n):
        y[i+1]= y[i]+ h*fonction(t1[i],y[i])
    return t1, y

t2,solution_Euleur= Euleur_explicite(equation, t_initial,
    t_final,99,u_0)

plt.plot(t2,solution_Euleur)
plt.legend(["Solution_scipy","Solution_exacte","
    Euleur_explicite"])
plt.show()
```

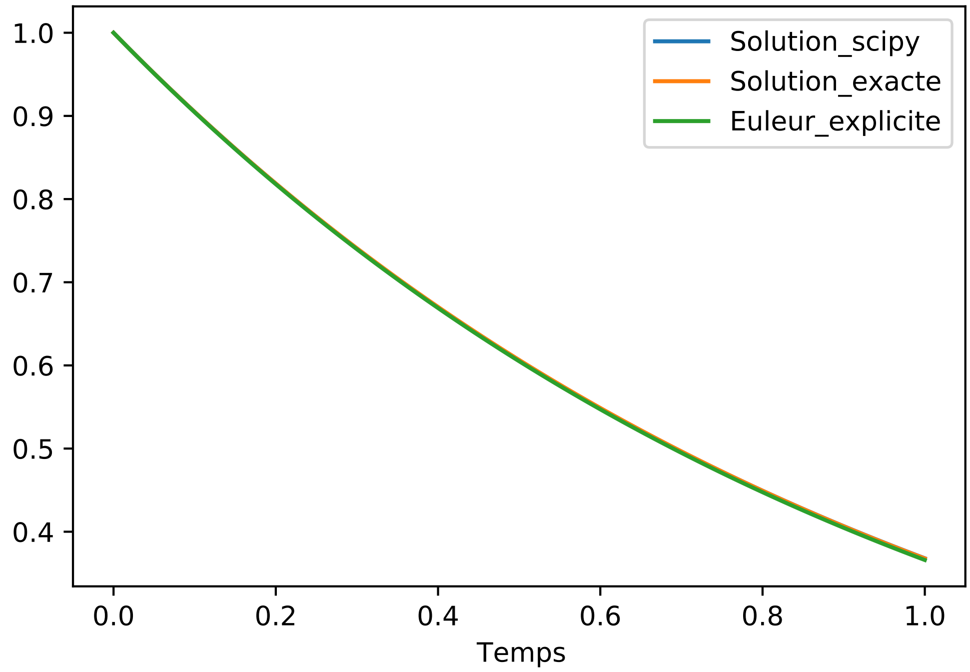


Figure 1: comparaison des Solutions

Traçons la courbe d'erreur L_2

On définit l'erreur L_2 :

$$L_2 = \sqrt{\sum_i^n (U^{app} - U^{exac})^2}$$

```
# definition de l'erreur L2
f_erreur=np.zeros(Nb)
dt=np.linspace(0,1,Nb)
Erreur_L2=np.zeros(Nb)
for i in range (Nb):
    f_erreur[i]=np.sum((solution_Euleur[i]-resultat[i])**2)
    Erreur_L2[i]=np.sqrt(f_erreur[i])

plt.loglog(dt,Erreur_L2)
plt.xlabel("pas Temps")
plt.ylabel("function_erreur")
plt.legend(["Erreur"])
plt.savefig('Erreur.pdf', dpi=300, bbox_inches='tight')
plt.show()
```

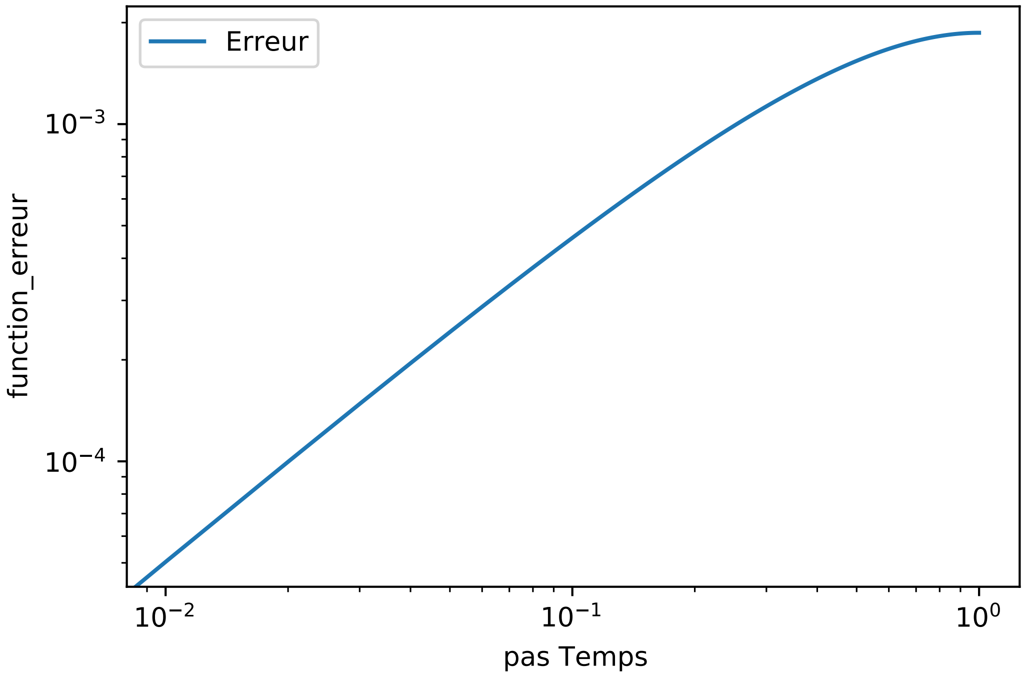


Figure 2: Courbe de l'erreur

Partie 2

On pose: $j'(x) = sin(x), \quad j''(x) = cos(x) \quad x \in [0, \pi]$

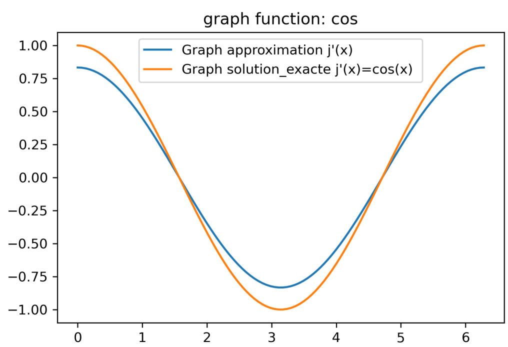
Méthode de différence finie
soit $L_h u$ L'opérateur approché:

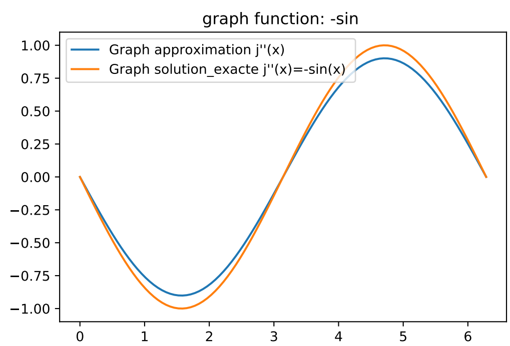
$L_h u = \frac{u(x+h)-2u(x)-u(x-h)}{h}$ pour la dérivée seconde

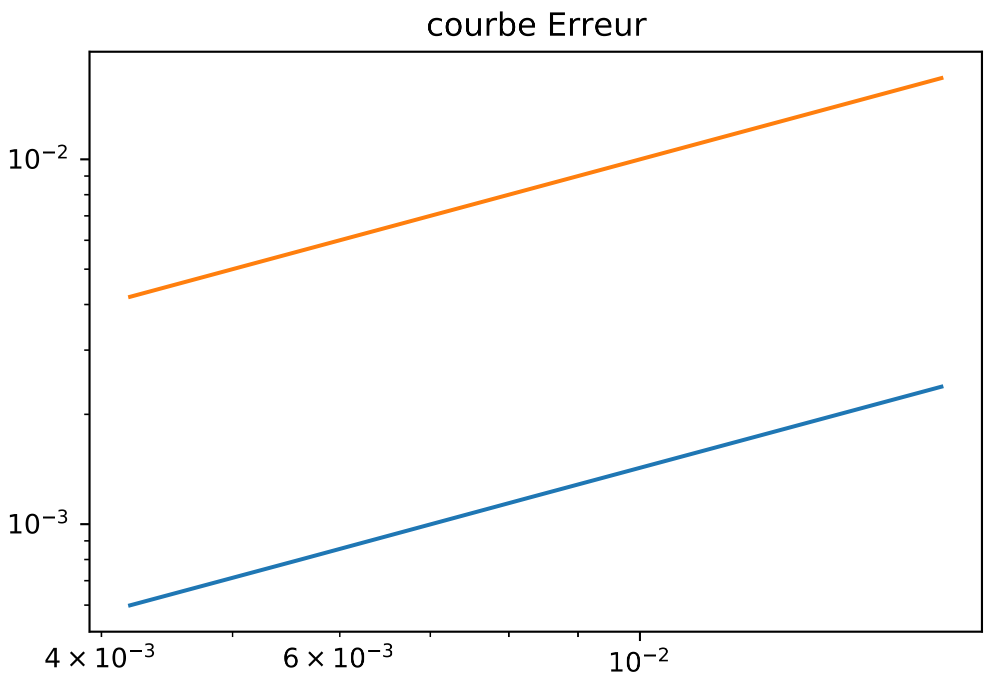
$L_h u = \frac{u(x+h)-u(x-h)}{2h}$ pour la dérivée première

```
# ----------------------------------
# j(x)=sin(x) calculons j'(x) et j''(x) par la methode:


# Methode de la difference finie

N=100
h= 1/N+1#[1.0E-01,5.0E-02,1.0E-02,5.0E-03,1.0E-03 ]
x=np.linspace(0,2*np.pi,N)
J=lambda x : np.sin(x)

Y=np.cos(x) # fonction  exacte
YY= -1*np.sin(x) # fonction exacte
```

```
J_diff_1 =np.zeros(N)
J_diff_2 =np.zeros(N)
fun_sin  =np.zeros(N)

for i in range(N):
    J_diff_2[i]=( J(x[i]+1) -2*J(x[i]) +J(x[i]-1))/(h**2) # derive seconde
    J_diff_1[i]=( J(x[i]+1) - J(x[i]-1))/(h*2) # la derive premiere

plt.plot(x,J_diff_1)
plt.plot(x,Y)
plt.legend(["Graph approximation j'(x)","Graph solution_exacte j'(x)=cos(x) "])
plt.title("graph function: cos")
plt.savefig('approxi_j_diff', dpi=300, bbox_inches='tight')
plt.show()

plt.plot(x,J_diff_2)
plt.plot(x,YY)
plt.legend(["Graph approximation j''(x)","Graph solution_exacte j''(x)=-sin(x) "])
plt.title("graph function: -sin ")
plt.savefig('approx_j_diffdiff.pdf', dpi=300, bbox_inches='tight')
plt.show()
```

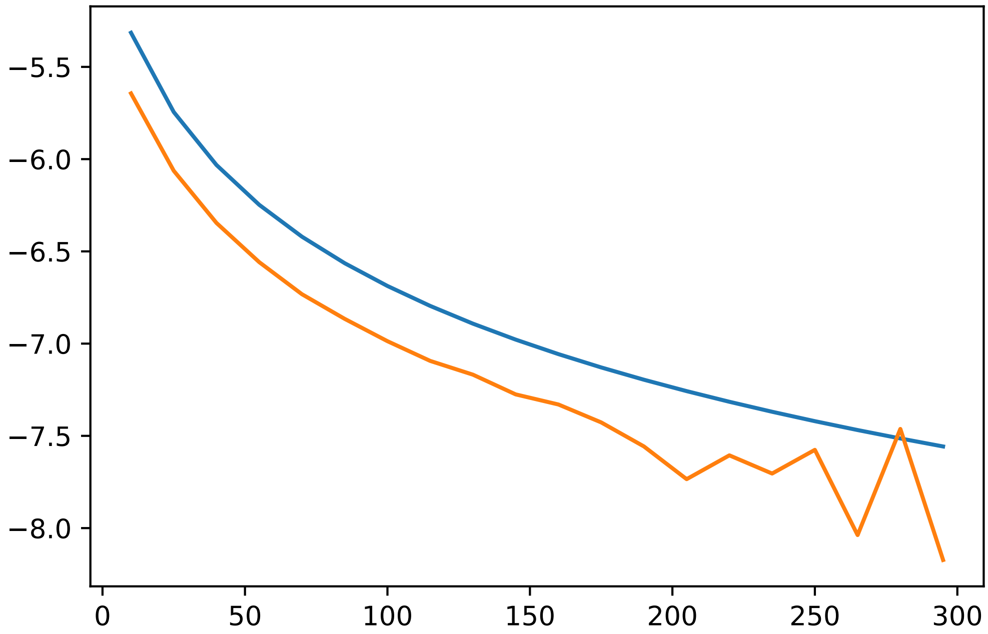







```
def func(x):

    T = np.sin(x)
    Txe = np.cos(x)
    Txxe = -np.sin(x)

    return T,Txe,Txxe

#############################################################
def derivatives(NX):

    NX=int(NX)
    L=2*math.pi
    dx = L/(NX-1)

    x = np.linspace(0.0,L,NX)

    T,Txe,Txxe=func(x)

    Tx = np.zeros(NX)
    Txx = np.zeros(NX)

    for j in range (1, NX-1):
        Tx[j] = (T[j+1]-T[j-1])/(dx*2)
        Txx[j] = (T[j-1]-2*T[j]+T[j+1])/(dx**2)

    Tx[0] = (T[1]-T[0])/dx
```

```
    Tx[NX-1] = (T[NX-1]-T[NX-2])/dx
    Txx[0] = 2*Txx[1]-Txx[2]
    Txx[NX-1] = 2*Txx[NX-2]-Txx[NX-3]

    errTx=0
    errTxx=0
    for j in range (0, NX):
        errTx+=abs(Tx[j]-Txe[j])*dx
        errTxx+=abs(Txx[j]-Txxe[j])*dx

#complex variable method
    coef=100
    dx/=coef
    errcTx=0
    errcTxx=0
    for j in range (0, NX):
        eps = complex(0,dx)
        Tc0 = np.sin(x[j])
        Tc = np.sin(x[j]+eps)
        Tcx=Tc.imag/eps.imag
        Tcxx=(Tc0-Tc).real/(eps.imag)**2*2
        errcTx+=abs(Tcx-Txe[j])*dx*coef
        errcTxx+=abs(Tcxx-Txxe[j])*dx*coef

    return errTx,errTxx,errcTx,errcTxx
##############################################################

Nsample=20
DNX=round(300/Nsample)

sample=np.zeros((Nsample))
terrTx=np.zeros((Nsample))
terrTxx=np.zeros((Nsample))
terrcTx=np.zeros((Nsample))
terrcTxx=np.zeros((Nsample))

NX=10
for j in range (0, Nsample):
    sample[j]=NX
    NX+=DNX
    terrTx[j],terrTxx[j],terrcTx[j],terrcTxx[j]=derivatives(NX)
#    print(NX,terrTx[j],terrTxx[j],terrcTx[j],terrcTxx[j])

#
plt.figure(1)
plt.plot(sample,np.log10(terrTx))
plt.plot(sample,np.log10(terrTxx))
plt.figure(2)
plt.title(u'L1 ERROR ON TX & TXX by CVM')
plt.xlabel(u'$NX$', fontsize=26)
plt.ylabel(u'$|Tx - Tx_{cvm}|$', fontsize=26, rotation=90)
plt.plot(sample,np.log10(terrcTx))
plt.plot(sample,np.log10(terrcTxx))
plt.show()
```


−5.5
−6.0
−6.5
−7.0
−7.5
−8.0
0
50
100
150
200
250
300