

```

# <tmp 1>

-----
#Code utilisé pour le projet
-----

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation
import math
import copy
from matplotlib import cm

-----
#Paramètres poutre
-----

L #longueur de la poutre(m)
b #largeur(m)
h #hauteur(m)
E #Module d'Young(GPa)
I #moment quadratique(m^4)
ro #masse volumique du matériau(kg/m^3)
delta = ro *h * b / (E*I) #coefficient constant de l'EDP
M #masse sismique au bout de la poutre(kg)
lambd = -M * g * (L2-L) / (E*I) #sert pour les conditions aux limites en prenant en compte la masse sismique

-----
#Paramètres simulation
-----

T #Temps (s)
Nt #nombre d'étapes de temps
Nx #nombre de mailles selon x
dt= Time/Nt #pas temporal(s)
dx= L/Nx #pas spatial(m)

```

```

w #pulsation des oscillations de la base vibrante
gamma= dt**2 / (delta * dx**4)

def mouvement_base(n, w):
    return math.sin(w*n)

#-----
#Résolution de l'équation avec masse sismique
#-----

def liste_masse():
    Z = [] #liste des Yt

    RHS = np.linspace(0, L, Nx) #permet d'alléger les
expressions

    X = np.linspace(0, L, Nx) #liste des pas spatiaux

    Y0 = np.zeros((Nx))
    Y1 = np.zeros((Nx))

#on va initialiser Y1 en utilisant le schéma de résolution mais
en tronquant le terme Yt-1

    for x in range(2, Nx-2):
        RHS[x] = gamma * ( Y0[x-2] - 4*Y0[x-1] + 6*Y0[x] -
4*Y0[x+1] + Y0[x+2] )

    for x in range(2, Nx-2):
        Y1[x] = 2*Y0[x] - RHS[x]

#On rajoute les conditions aux limites pour Y0 et Y1 et qui
seront les mêmes pour toutes les étapes de temps

    Y0[0] ,Y0[1] = mouvement_base(0,w), mouvement_base(0,w)
    Y1[0], Y1[1] = mouvement_base(dt,w), mouvement_base(dt,w)
    Y0[Nx-3], Y1[Nx-3] = Y0[Nx-4], Y1[Nx-4]
    Y0[Nx-2], Y1[Nx-2] = Y0[Nx-4]/5, Y1[Nx-4]/5

```

```

Y0[Nx-1], Y1[Nx-1] = Y0[Nx-4]/5, Y1[Nx-4]/5

#Boucle principale
for t in range(Nt):

    for x in range(2, Nx-2):

        RHS[x] = gamma * (Y1[x-2] - 4*Y1[x-1] + 6*Y1[x] -
4*Y1[x+1] + Y1[x+2])

        for x in range(2, Nx-2):

            H = copy.deepcopy(Y1)

            Y1[x] = 2*Y1[x] - Y0[x]-RHS[x]

            Y0 = H

        Y1[0], Y1[1] = mouvement_base(n*dt,w),
mouvement_base(n*dt,w)

        Y1[Nx-3] = Y1[Nx-4]

        Y1[Nx-2] = Y1[Nx-4]/5

        Y1[Nx-1] = Y1[Nx-4]/5

        if t%800==0: #On ne récupère que tous les 800 pas
temporels

            O = copy.deepcopy(Y1)

            Z.append(O)
return Z

-----
#Calcul de l'accélération en bout de poutre
-----

def acceleration(a, b, c, dt): # a = Yt+1[Nx-1], b = Yt[Nx-1], c
= Yt-1[Nx-1]

    return (a - 2 * b + c) / (800 * dt)**2 #formule de Taylor

def Liste_acceleration_bout_poutre(Liste, dt):

```

```

Acc = []

for t in range(1, len(Liste) - 1):
    Acc.append(acceleration( L[i+1][-1], L[i][-1], L[i-1]
[-1], dt))

return Acc

def liste_acceleration_encastrement(w):
    T = np.linspace(0, 6000000*dt, 7498)
    L = [-w**2*math.sin(w*t) for t in T]
    return L

def choix_mat(Liste, dt, w):
    LE = liste_acceleration_encastrement(Liste, w)
    LB = liste_acceleration_bout_poutre(Liste, dt)
    L = []
    for i in range(len(LE)):
        L.append(LE[i]-LB[i])
    return L
#-----
#Calcul de R(x,t)
#-----

def derivee_tiers(a, b, c, d, dx): #calcule la dérivée tiers de
y(x,t) selon x, à l'instant t
#a = Yt[x+2], b = Yt[x+1], c = Yt[x-1], d = Yt[x-2]

    return (a - 2 * b + 2 * c - d) / (2 * dx**3) #formule de
Taylor

def Liste_derivee_tiers(Liste, t, dx):
    T = []
    for x in range(2, len(Liste[t]) - 2):
        T.append(derivee_tiers(L[t][x+2], L[t][x+1], L[t][x-1],
L[t][x-2]))

```

```

    return T

def liste_vitesse(Liste, x):
    V = []
    for t in range(1, len(Liste)):
        V.append((L[t][x] - L[t-1][x]) / (800 * dt)) #formule de Taylor
    return V

def changement(Liste):
    T=[]
    for t in range(len(Liste[0])): #le temps
        T_inter = []
        for x in range(len(L)): #l'espace
            T_inter.append(L[x][t]) #à t fixé, on construit la liste des points variant selon x
        T.append(Tt)
    return T

def multiplication_liste(L1,L2):
    Produit = []
    for t in range(len(L1)):
        Produit_inter = []
        for x in range(len(L1[0])):
            Produit_inter.append(L1[i][j] * L2[i][j])
        Produit.append(Produit_inter)
    return Produit

def mulplication_scalaire(L1,c):
    for t in range(len(L1)):

```

```

        for x in range(len(L1[0])):
            L1[t][x] = c * L1[t][x]

    return L1

def Puissance(Liste, dt, dx):
    Vit = []
    Tiers = []
    for t in range(len(Liste)):
        Tiers.append(Liste_derivee_tiers(Liste, t))
        for x in range(len(L[0])):
            Vit.append(liste_vitesse(Liste, x))

    Vit.pop(0) #pas de dérivée tiers calculée pour les 2 premiers/derniers pas spatiaux
    Vit.pop(1)
    Vit.pop(-1)
    Vit.pop(-2)

    Vit=swift(Vit)

    Tiers.pop(0) #car il n'y a pas de vitesse calculée pour t=0

    return multiplication_scalaire(E*I,
multiplication_liste(Tiers, Vit))[]

#-----
#Optimisation de la résolution de l'équation
#-----
def matrice(Nx): #Retourne la matrice A utile au schéma de résolution

    M = np.zeros((Nx,Nx))
    for i in range(2, Nx-2):

        M[i,i-2] = 1
        M[i,i-1] = -4
        M[i,i] = 6

```

```

M[i,i+1] = -4
M[i,i+2] = 1
return gamma*M

RHS=np.zeros((Nx))
Y0=np.zeros((Nx))
Y1=np.zeros((Nx))

U=np.zeros((Nt,Nx)) #tableau des déformées

for j in range(2, Nx-2):
    RHS[j]=gamma*(Y0[j-2] - 4*Y0[j-1] + 6*Y0[j] - 4*Y0[j+1] +
Y0[j+2])
for j in range(2, Nx-2):
    Y1[j]=2*Y0[j] - RHS[j]

Y0[0], Y0[1] = mouvement_base(0,w), mouvement_base(0,w)
Y1[0], Y1[1] = mouvement_base(dt,w), mouvement_base(dt,w)
Y0[Nx-3], Y1[Nx-3] = Y0[Nx-4], Y1[Nx-4]
Y0[Nx-2], Y1[Nx-2] = Y0[Nx-4], Y1[Nx-4]
Y0[Nx-1], Y1[Nx-1] = Y0[Nx-4], Y1[Nx-4]

U[0,:] = Y0
U[1,:] = Y1
A=matrice(Nx)

for t in range(2, Nt):
    U[t,:] = 2*U[t-1,:] - U[t-2,:] - np.dot(A, U[t-1,:])
    U[t,0], U[t,1] = mouvement_base(t*dt,w),
mouvement_base(t*dt,w)
    U[t,Nx-3] = U[t,Nx-4]

```

$$U[t, Nx-2] = U[t, Nx-4]$$

$$U[t, Nx-1] = U[t, Nx-4]$$