***Informatique***

***Concours Blanc - Éléments de correction***

**Q1 :**

**def** **main**():

[L,n,m,k,c]=**geometrie**(0)

**...**

**Q2** :

L = 0.5 ; n = 3 ;

m = [0.1,0.2,0.1] ; k=[20000,20000,1000] ; c = [1000,10,10]

**Q3 :**

**def** **lire\_fichier**(nom\_fic):

f=**open**(nom\_fic,”r”)

L=**float**(f.**readline**())

n=**float**(f.**readline**())

m,k,c=[],[],[]

**for** i **in** **range**(n):

m.**append**(**float**(f.**readline**()))

**for** i **in** **range**(n):

k.**append**(**float**(f.**readline**()))

**for** i **in** **range**(n):

c.**append**(**float**(f.**readline**()))

**return** [L,n,m,k,c]

**Q4 :**

requete1=”**SELECT** nom,date,id **FROM** CALCUL”

**Q5 :**

T=**interroge\_bdd**(“**SELECT** L,n **FROM** CALCUL **WHERE** id=id\_calcul”)

L\_local=T[0][0]

n\_local=T[0][1]

T=**interroge\_bdd**(“**SELECT** masse **FROM** POUTRE **WHERE** id\_calcul=id\_calcul”)

#syntaxe numpy

m\_local=T[:,0]

#syntaxe python

m\_local=[]

**for** x **in** T :

m\_local.**append**(x[0])

T=**interroge\_bdd**(“**SELECT** raideur,amortissement **FROM** LIAISON **WHERE** id\_calcul=id\_calcul”)

#syntaxe numpy

k\_local=T[:,0]

c\_local=T[:,1]

#syntaxe python

k\_local=[]

c\_local=[]

**for** x **in** T :

k\_local.**append**(x[0])

c\_local.**append**(x[1])

**Q6 :**

M=

0

0

K=

0

0

C=

0

0

**Q7 :**

**def** **creation\_operateur**(n,m,k,c):

M=**zeros**((n,n),float)

**for** i **in** **range**(n):

M[i,i]=m[i]

K=**zeros**((n,n),float)

C=**zeros**((n,n),float)

**for** i **in** **rang**e(1,n-1):

K[i,i]=k[i]+k[i+1]

C[i,i]=c[i]+c[i+1]

K[i,i-1]=-k[i]

C[i,i-1]=-c[i]

K[i,i+1]=-k[i+1]

C[i,i+1]=-c[i+1]

K[0,0]=k[0]+k[1]

K[0,1]=-k[1]

C[0,0]=c[0]+c[1]

C[0,1]=-c[1]

K[n-1,n-1]=k[n-1]

K[n-1,n-2]=-k[n-1]

C[n-1,n-1]=c[n-1]

C[n-1,n-2]=-c[n-1]

**return** [M,K,C]

**Q8 :**

 et 

Système matriciel : 









**Q9 :**

**def** **calcul**(n,M,K,C,npts,dt,fmax,omega):

H=M/dt\*\*2+C/dt+K

F=**zeros**((n,1),float)

X=[**zeros**((n,1),float),**zeros**((n,1),float)] # X0 et X1 sont nuls

**for** i **in** **range**(2,npts+2):

F[n-1]=fmax\***sin**(omega\*(i-1)\*dt)

# dot : produit matriciel

G=**dot**((2\*M/dt\*\*2+C/dt),X[i-1])-**dot**(M/dt\*\*2,X[i-2])+F

X.**append**(**resoud**(H,G))

#X.**append**(**np.solve**(H,G)) résolution numpy

**return** X[1:]

**Q10 :**

**def** **resoud**(H,G):

G2=**copy**(G) # copie par valeurs et non par référence

H2=**copy**(H) # les tableaux H et G ne seront pas modifiés

n=**len**(G2)

X=**zeros**((n,1),float)

**for** i **in** **range**(1,n):

a=H2[i-1,i-1]

b=H2[i,i-1]

H2[i]=H2[i]\*a-H2[i-1]\*b

G2[i]=G2[i]\*a-G2[i-1]\*b

X[n-1]=G2[n-1]/H2[n-1,n-1]

**for** i **in** **range**(n-2,-1,-1):

X[i]=(G2[i]-H2[i,i+1]\*X[i+1])/H2[i,i]

**return** X

**Q11 :**

Les instructions suivantes

H[i]=H[i]\*a-H[i-1]\*b

G[i]=G[i]\*a-G[i-1]\*b

sont de complexité linéaire en n (il y a n colonnes).

La complexité de l’algorithme de la fonction **resoud** est donc de type quadratique.

La complexité de l’algorithme classique du pivot de Gauss est de type cubique.

**Q12 :**

Ntps \* n \* 64 = 192E7 bits = 0,24 Go.

**Q13 :**

**def** **affiche\_deplacement\_pdt**(q,X,L,n,ampl):

T=[]

**for** i **in** **range**(n):

T.**append**((X[q][i]+(i+1)\*L/n)\*ampl)

plt.**plot**(T,n\*[0.0],"D")

NB : Une version améliorée pour afficher la poutre au cours des itérations :

**def** **affiche\_deplacement\_pdt**(q,X,L,n,ampl):

T=[]

**for** i **in** **range**(n):

T.**append**((X[q][i]+(i+1)\*L/n)\*ampl)

plt.**plot**(T,n\*[0.5\*q],"D")

L=0.3

n=3

m=[0.1,0.2,0.1]

k=[20000,20000,1000]

c=[1000,10,10]

npts=20

fmax=100

omega=1

dt=2\*pi/omega/npts

[M,K,C]=creation\_operateur(n,m,k,c)

X=**calcul**(n,M,K,C,npts,dt,fmax,omega)

ampl=1

**for** i **in** **range**(npts+1):

**affiche\_deplacement\_pdt**(i,X,L,n,ampl)

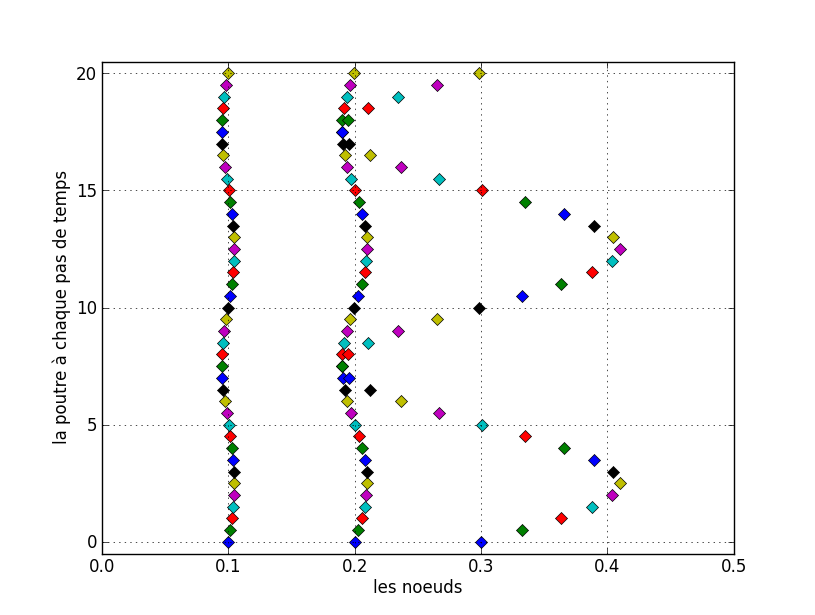
plt.**axis**([0,0.5,-0.5,0.5\*(npts+1)])

plt.**ylabel**(u'la poutre à chaque pas de temps')

plt.**xlabel**('les noeuds')

plt.**grid**()

plt.**show**()

****

**Q14 :**

**def** **lieu\_temps\_depl\_max**(X):

npts=**len**(X)

n=**len**(X[0])

k=0

t=0

**fo**r i **i**n **range**(npts):

**fo**r j **in** **rang**e(1,n):

**if** **abs**(X[i][j])>**abs**(X[t][k]):

t=i

k=j

**return** k+1,t,**abs**(X[t][k][0])

**Q15 :**

**def** **calcul\_energie**(X,c,dt):

m=**len**(X)

n=**len**(X[0])

E=0

TE=[]

Tt=[]

TP=[]

# calcul des vitesses

V=[n\*[0]]

**for** i **in** **range**(1,m):

T=[]

**for** j **in** **range**(0,n):

T.**appen**d((X[i][j][0]-X[i-1][j][0])/dt)

V.**append**(T)

**for** i **in** **range**(0,m):

Ps=0

**for** j **in** **range**(0,n):

**if** j==0:

Ps+=c[j]\*(V[i][j])\*\*2

**else**:

Ps+=c[j]\*(V[i][j]-V[i][j-1])\*\*2

E+=Ps\*dt # Méthode des rectangles à gauche

TP.**append**(Ps)

TE.**append**(E)

Tt.**append**(i\*dt)

#Tracés

plt.**grid**()

plt.**ylabel**(u"Energie dissipée")

plt.**xlabel**("temps")

plt.**plot**(Tt,TE)

plt.**figure**(2)

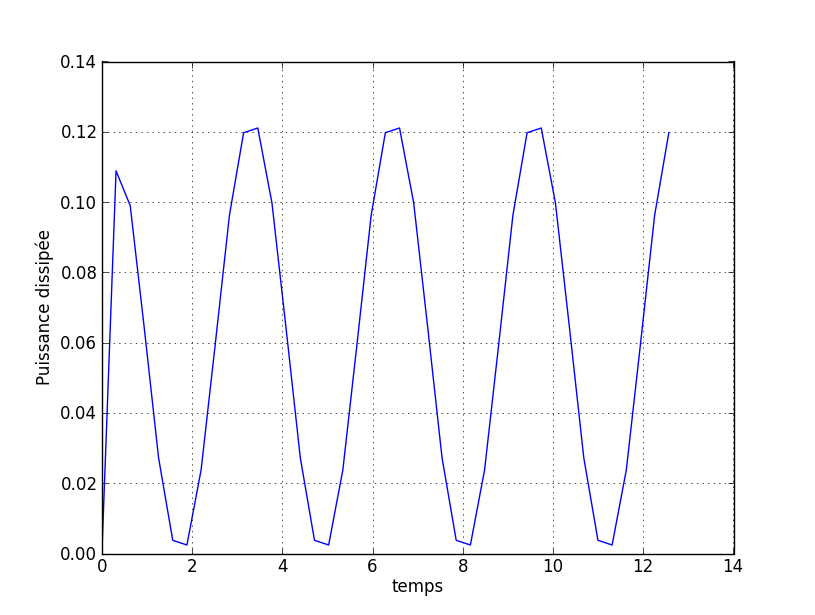
plt.**plot**(Tt,TP)

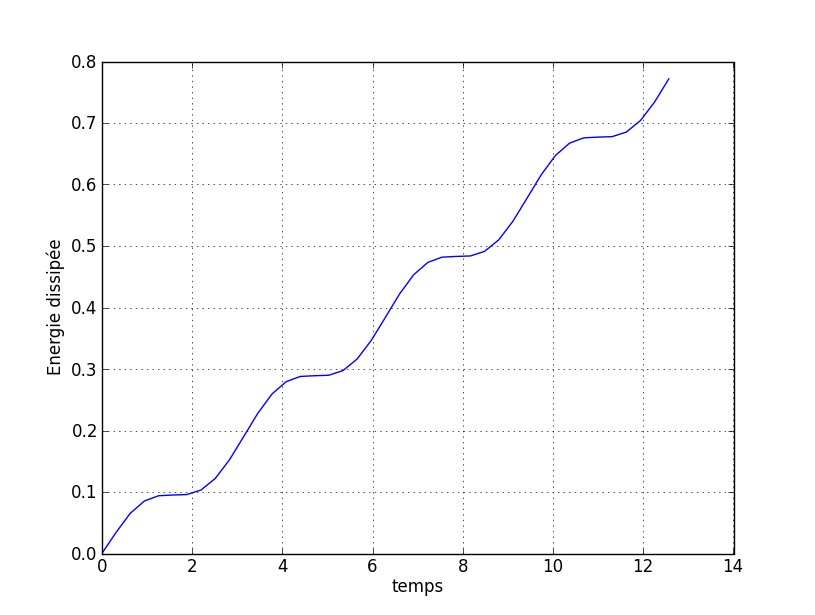
plt.**grid**()

plt.**ylabel**(u"Puissance dissipée")

plt.**xlabel**("temps")

plt.**show**()





Remarques

Ce sujet 0 est a été publié sur le site du concours CCP.

Il comporte quelques erreurs et manques, sans conséquences sur la compréhension et la notation de l’épreuve.

* Page 3/11 : erreurs dans les équations, sans conséquences pour la suite du sujet.

 (1)

 (2)

 (3)

* Page 4/11 : programme en Python.

Il manque quelques arguments aux fonctions, sans conséquences pour la suite du sujet.

X=calcul(n,M,K,C,npts,dt,fmax,omega)

posttraitement(X,L,n,c,dt)

* Page 9/11 : Erreur dans la formule de la puissance dissipée :



Q15, un argument supplémentaire est nécessaire : Ediss=calcul\_energie(X,c,dt).

* Page 10/11 : méthode readline()

Une ligne est une chaîne de caractères qui se termine par le caractère ASCII « retour ligne » (Line Feed : ‘\n’).

La méthode readline() lit et **retourne** une ligne.

float(‘12\n’) retourne le réel 12.0.

* Page 11/11 : Python et numpy

**On suppose que pour tout le sujet, la bibliothèque** numpy **a été importée.**

* La définition d’un tableau peut se faire de plusieurs manières :
  + Python : T=[[1,2,3],[2,1,5],[4,5,8]]
  + Numpy : T=array(([1,2,3],[2,1,5],[4,5,8]))

On peut alors accéder à des fonctionnalités supplémentaires.

Par exemple :

T[i,j] identique à T[i][j]

T[:,0:3) : extrait les trois premières colonnes.

T=zeros((3,3),float) : T est de type ’array’

* **Le produit matriciel n’est pas « \* » pour les tableaux**.

Il existe pour cela la fonction numpy « dot ».

Par exemple : B = dot(A,X) (B=AX où A est une matrice, X un vecteur)

* Il est possible avec numpy de créer une variable de type ‘matrix’ (matrice) qui admet alors l’opérateur « \* » pour produit, à la manière de Scilab ou Matlab. Il existe alors aussi un certain nombre d’opérateurs et de méthodes associés au calcul matriciel.

Par exemple : M=matrix(([1,2,3],[2,1,5],[4,5,8]))

M.T : transposée de la matrice M

M\*\*2 : M\*M

* Pour effectuer une copie par valeurs et non par référence d’un tableau, il existe la fonction numpy « copy ». Par exemple : A = copy(B).

La modification de A, n’aura alors aucune conséquence sur B.

Cela n’est pas le cas si l’instruction « A = B » a été tapée.

* Bibliothèque « matplotlib.pyplot »

**On suppose que pour tout le sujet, la bibliothèque** matplotlib.pyplot **a été importée.**