CCP PC 2015
Modélisation de systèmes physiques ou chimiques
Corrigé - version Python

I.A.1 Lorsque les dimensions transversales (selon y et z) du mur sont grandes devant l'épaisseur e du mur, on peut négliger les effets de bords : $T(x,t)$.

I.A.2 Equation de la chaleur en l'absence de sources d'énergie (et λ supposée constante) :

$$\rho c_p \frac{\partial T}{\partial t} = \lambda \Delta T$$

Si T est indépendante de y et z alors :

$$\rho c_p \frac{\partial T}{\partial t} = \lambda \frac{\partial^2 T}{\partial^2 x}$$

I.B.1 (i) température imposée aux limites du système :

$$T(0,t) = T_{int} \text{ et } T(e,t) = T_{ext}$$

(ii) température imposée à l'intérieur et paroi extérieure isolée par un matériau de faible conductivité :

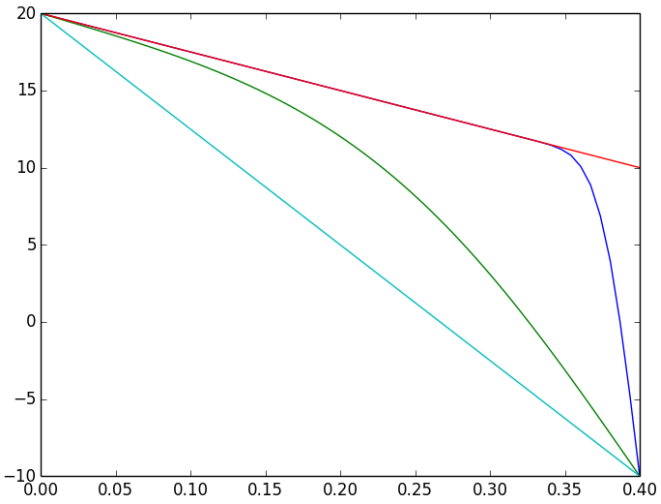
$$T(0,t) = T_{int} \text{ et } j_Q(e,t) = -\lambda \frac{\partial T}{\partial x}(e,t) = 0$$

I.C.1 En régime permanent,

$$\frac{\partial T}{\partial t} = 0 \Rightarrow \frac{\partial^2 T}{\partial^2 x} = 0 \Rightarrow T(x) = ax + b \text{ avec } T(0) = T_{int} \text{ et } T(e) = T_{ext}$$

$$T(x, t_1 \leq 0) = (T_{ext1} - T_{int})\, x/e + T_{int}$$

$$T(x, t_2 \to \infty) = (T_{ext2} - T_{int})\, x/e + T_{int}$$



I.C.2 $T(x, t_1 \leq 0)$ et $T(x, t_2 \to \infty)$ sont tracés en rouge et bleu clair.

I.C.3 Juste après le changement brutal, la température n'a significativement varié qu'au voisinage de e (tracé en bleu foncé).
Après un temps suffisamment long le changement de température se fait ressentir dans toute l'épaisseur du mur (tracé vert).
On peut utiliser le temps caractéristique $\tau = \alpha L^2$ d'évolution de la température sur une longueur L :

- $\tau = 10\text{s}$ donne $L \approx 3\text{mm}$
- $\tau = 500\text{s}$ donne $L \approx 2\text{cm}$.

II.A.1 $\alpha = \rho c_p / \lambda$ est l'inverse de la diffusivité thermique.

II.A.2 Pour les conditions initiales on reprend le résultat du I.C.1

$$T(x, 0^-) = ax + b \text{ avec } a = \frac{T_{ext1} - T_{int}}{e} \text{ et } b = T_{int}$$

II.B.1 $\Delta x = e/(N+1)$ et $x_i = i\,\Delta x$

II.B.2.a $T(x + \Delta x, t) = T(x,t) + \Delta x \frac{\partial T}{\partial x}(x,t) + \frac{(\Delta x)^2}{2} \frac{\partial^2 T}{\partial^2 x}(x,t) + \frac{(\Delta x)^3}{3!} \frac{\partial^3 T}{\partial^3 x}(x,t) + o((\Delta x)^3)$

$$T(x - \Delta x, t) = T(x,t) - \Delta x \frac{\partial T}{\partial x}(x,t) + \frac{(\Delta x)^2}{2} \frac{\partial^2 T}{\partial^2 x}(x,t) - \frac{(\Delta x)^3}{3!} \frac{\partial^3 T}{\partial^3 x}(x,t) + o((\Delta x)^3)$$

II.B.2.b donc en sommant : $\frac{\partial^2 T}{\partial^2 x}(x,t) \cong \frac{-2 \times T(x,t) + T(x + \Delta x, t) + T(x - \Delta x, t)}{(\Delta x)^2}$

II.B.2.c En remplaçant dans l'expression précédente $T(x_i, t_k)$ par son évaluation numérique T_i^k, on obtient :

$$\frac{\partial^2 T}{\partial^2 x}(x_i,t_k) \cong \frac{-2T_i^k + T_{i+1}^k + T_{i-1}^k}{(\Delta x)^2}$$

II.B.2.d $T(x,t+\Delta t) = T(x,t) + \Delta t \frac{\partial T}{\partial t}(x,t) + o(\Delta t)$

II.B.2.e $\frac{\partial T}{\partial t}(x,t) \cong \frac{T(x,t+\Delta t) - T(x,t)}{\Delta t}$

II.B.2.f En remplaçant dans l'expression précédente $T(x_i,t_k)$ par son évaluation numérique T_i^k, on obtient :

$$\frac{\partial T}{\partial t}(x_i,t_k) \cong \frac{T_i^{k+1} - T_i^k}{\Delta t}$$

II.B.2.g En remplaçant les résultats du b et f dans l'équation (1) on obtient :

$$\alpha \frac{T_i^{k+1} - T_i^k}{\Delta t} \cong \frac{-2T_i^k + T_{i+1}^k + T_{i-1}^k}{(\Delta x)^2}$$

II.B.2.h d'où (2) $T_i^{k+1} = rT_{i-1}^k + (1-2r)T_i^k + rT_{i+1}^k$ avec $r = \frac{\Delta t}{\alpha(\Delta x)^2}$

II.B.2.i (2) n'est valable que si T_{i+1}^k et T_{i-1}^k existent, donc pour $i \in \{1; N\}$.
$T_0^k = T_{int}$ et $T_{N+1}^k = T_{ext2}$ $\forall k$

II.B.2.j.*(i)* et *(viii)* On utilise des tableau numpy (nd.array) comme le suggère l'annexe B , la programmation pourrait être faite avec des listes Python, à ce sujet regarder la note du II.B.2.j.*(vii)*.
def schema_explicite(T0): # T0 est un tableau numpy (nd.array) contenant les valeurs initiales $T_{i\in\{1;N\}}^0$
    ...
    return k,T_tous_k # k = entier = nombre d'itérations, T_tous_k est un tableau numpy

II.B.2.j.*(ii)* à placer dans le programme principal
```
r=dt/(dx**2*alpha)
if r>=0.5 : print("""simulation risquée dt trop grand ou dx trop petit""")
```
ou
```
r=dt/(dx**2*alpha)
assert r<=0.5,"""simulation risquée dt trop grand ou dx trop petit"""
```
Cette dernière proposition interrompt le programme.

II.B.2.j suite et fin
```
def schema_explicite(T0):
    N=len(T0) # N points intermédiaires (0 et N+1 exclus)                    (iii)
    ItMax=2000
    T_tous_k=np.zeros((N,ItMax)) # float64 par defaut
    T_tous_k[:,0]=T0[:]   # 1ère colonne : température initiale des points xi   (iv)
    k=1 # itération n°1, instant dt                                          (v)
    T_tous_k[0,1]=r*Tint+(1-2*r)*T_tous_k[0,0]+r*T_tous_k[1,0]   # ligne 0 = point x1
    for i in range(1,N-1) :   # ligne i = point x i+1
        T_tous_k[i,1]=r*T_tous_k[i-1,0]+(1-2*r)*T_tous_k[i,0]+r*T_tous_k[i+1,0]
    T_tous_k[N-1,1]=r*T_tous_k[N-2,0]+(1-2*r)*T_tous_k[N-1,0]+r*Text2   # point xN
    norme2=calc_norme(T_tous_k[:,1]-T_tous_k[:,0])                           (vii)
    while (k<ItMax-1 and norme2>0.01):
        T_tous_k[0,k+1]=r*Tint+(1-2*r)*T_tous_k[0,k]+r*T_tous_k[1,k]   # point x1
        for i in range(1,N-1) :   # point x i+1
            T_tous_k[i,k+1]=r*T_tous_k[i-1,k]+(1-2*r)*T_tous_k[i,k]+r*T_tous_k[i+1,k]
        T_tous_k[N-1,k+1]=r*T_tous_k[N-2,k]+(1-2*r)*T_tous_k[N-1,k]+r*Text2   # point xN
        norme2=calc_norme(T_tous_k[:,k+1]-T_tous_k[:,k])
        k+=1 # itération k+1 terminée
    return k,T_tous_k
```

Note pour le calcul de la norme :
La fonction calc_norme définie ci-dessous prend en argument un vecteur qui est la différence $T^{k+1} - T^k$.

T_tous_k[:,k] représente la (k-1)ème colonne du tableau T_tous_k (indices commençant à 0).
L'utilisation de nd.array présente les avantages suivants :

- technique du slicing : T_tous_k[:,k] n'a pas d'équivalent pour les listes, assurément pas T_tous_k[:][k] !
- opérations algébriques sur les tableaux cf C.1.d

II.B.2.j.*(vi)* dans le code, à placer avant la fonction schema_explicite

```
def calc_norme(vecteur):
    N=len(vecteur)
    norme=0
    for i in range(N):
        norme+=(vecteur[i])**2
    return norme**(0.5)
```

II.B.3.a&b $\frac{\partial^2 T}{\partial^2 x}(x,t) \cong \frac{\partial^2 T}{\partial^2 x}(x_i,t_{k+1}) \cong \frac{-2T_i^{k+1}+T_{i+1}^{k+1}+T_{i-1}^{k+1}}{(\Delta x)^2}$ avec $k \to k+1$ dans B. 2. a

$$\alpha \frac{T_i^{k+1}-T_i^k}{\Delta t} \cong \alpha \frac{\partial T}{\partial t}(x_i,t_k) = \frac{\partial^2 T}{\partial^2 x} \cong \frac{-2T_i^{k+1}+T_{i+1}^{k+1}+T_{i-1}^{k+1}}{(\Delta x)^2}$$

d'où (3) $T_i^k = (1+2r)T_i^{k+1} - rT_{i+1}^{k+1} - rT_{i-1}^{k+1}$ avec $r = \frac{\Delta t}{\alpha(\Delta x)^2}$

II.B.3.c On dispose des N équations suivantes :

$$T_i^k = \underbrace{-r}_{M_{i-1,i}} T_{i-1}^{k+1} + \underbrace{(1+2r)}_{M_{ii}} T_i^{k+1} \underbrace{-r}_{M_{i+1,i}} T_{i+1}^{k+1}$$

$T_1^k = -rT_{int} + (1+2r)T_1^{k+1} - rT_2^{k+1}$ car $T_0^{k+1} = T_{int}$
$T_N^k = -rT_{N-1}^{k+1} + (1+2r)T_N^{k+1} - rT_{ext2}$ car $T_{N+1}^{k+1} = T_{ext2}$
Elles s'écrivent sous la forme matricielle suivante :

$$\underbrace{\begin{pmatrix} T_1^k \\ \vdots \\ T_i^k \\ \vdots \\ T_N^k \end{pmatrix}}_{\boldsymbol{T^k}} = \underbrace{\begin{pmatrix} 1+2r & -r & & & & \\ \vdots & \ddots & & & & \\ & -r & 1+2r & -r & & \\ & & & \ddots & \vdots & \\ & & & -r & 1+2r \end{pmatrix}}_{M} \underbrace{\begin{pmatrix} T_1^{k+1} \\ \vdots \\ T_i^{k+1} \\ \vdots \\ T_N^{k+1} \end{pmatrix}}_{\boldsymbol{T^{k+1}}} - r \underbrace{\begin{pmatrix} T_{int} \\ 0 \\ \vdots \\ 0 \\ T_{ext2} \end{pmatrix}}_{\boldsymbol{v}} \Leftrightarrow M\boldsymbol{T^{k+1}} = \boldsymbol{T^k} + r\boldsymbol{v}$$

II.B.3.d On transpose simplement, en ayant au préalable initialisé à 0 les variables utilisées : c'_i, d'_i et u_i.

```
def CalcTkp1(M,d):
    N=len(d)
    cprim,dprim,u=np.zeros(N),np.zeros(N),np.zeros(N)
    cprim[0]=M[0,1]/M[0,0] # calcul de c'_1
    dprim[0]=d[0]/M[0,0] # calcul de d'_1
    for i in range(1,N-1): # jusque N-2 correspondant à c'_{N-1} et d'_{N-1}
        cprim[i]=M[i,i+1]/(M[i,i]-M[i,i-1]*cprim[i-1]) # calcul de c'_{i+1}
        dprim[i]=(d[i]-M[i,i-1]*dprim[i-1])/(M[i,i]-M[i,i-1]*cprim[i-1]) # calcul de d'_{i+1}
    dprim[N-1]=(d[N-1]-M[N-1,N-2]*dprim[N-2])/(M[N-1,N-1]-M[N-1,N-2]*cprim[N-2]) # calcul de d'_N
    u[N-1]=dprim[N-1] # calcul de u_N
    for i in range(N-2,-1,-1): # de N-2 à 0
        u[i]=dprim[i]-cprim[i]*u[i+1] # calcul de u_{i+1}
    return u
```

II.B.3.e

```
def schema_implicite(T0):                                          (i)
    N=len(T0) # N points intermédiaires entre les pts 0 et N+1
    ItMax=2000                                                     (ii)
    T_tous_k=np.zeros((N,ItMax))
    T_tous_k[:,0]=T0[:] # 1ère colonne : température initiale des points xi   (iii)
    # construction de la matrice M
    M=np.zeros((N,N))                                              (iv)
    M[0,0],M[0,1]=1+2*r,-r # 1ère ligne
    for i in range(1,N-1):
        M[i,i-1],M[i,i],M[i,i+1]=-r,1+2*r,-r # lignes intermédiaires
```

```
    M[N-1,N-2],M[N-1,N-1]=-r,1+2*r # dernière ligne
    # construction du vecteur v
    v=np.zeros(N)
    k=1 # itération n°1, instant dt                                    (v)
    v[0],v[N-1]=Tint,Text2
    T_tous_k[:,1]=CalcTkp1(M,T0+r*v)
    norme2=calc_norme(T_tous_k[:,1]-T_tous_k[:,0])
    while (k <ItMax-1 and norme2>0.01):                               (vi)
        T_tous_k[:,k +1]=CalcTkp1(M,T_tous_k[:,k]+r*v)
        norme2=calc_norme(T_tous_k[:,k +1]-T_tous_k[:,k])
        k +=1 # itération k +1 terminée
    return k,T_tous_k                                                 (vii)
```

C.1.a # paramètres
```
epais=0.4
Tint,Text1,Text2=20,10,-10
conduc,rho,c=1.65,2150,1000
N,dt=60,25
```

C.1.b # état initial
```
a=(Text1-Tint)/epais
b=Tint
```

C.1.c # tableau des abscisses
```
dx=epais/(N+1)
x=np.linspace(dx,epais-dx,N)  # N pts, 2 bornes incluses
```

C.1.d # tableau des T initiales
```
T0=a*x+b
```

C.1.e # paramètres modélisation
```
alpha=rho*c/conduc
r=dt/(dx**2*alpha)
```

C.2.3 # programme principal et tracés
```
abscisses=[0]+list(x)+[epais]  # concaténation de listes incluant les parois
fig=plt.figure()
plt.xlim(0,epais)  # taille de l'axe des abscisses
plt.ylim(-10,20) # taille de l'axe des ordonnées
rep=input('schéma explicite (ne rien taper) sinon implicite (taper n'importe quoi : ') # string vide = False
if rep:
    it,tab_T=schema_implicite(T0)
    temps=it*dt
    h,s=divmod(it*dt,3600)
    plt.title("schéma implicite, régime permanent en {0}h{1}mn".format(h,s//60))
else :
    if r>=0.5 : print("""simulation explicite risquée : dt trop grand ou dx trop petit""")
    it,tab_T=schema_explicite(T0)
    temps=it*dt
    h,s=divmod(it*dt,3600)
    plt.title("schéma explicite, régime permanent en {0}h{1}mn".format(h,s//60))
for i in range(it//100):
    plt.plot(abscisses,[Tint]+list(tab_T[:,100*i])+[Text2])
```

```
# Lionel Sautière MP* Wallon Valenciennes
# à associer au fichier Word
import numpy as np
import matplotlib.pyplot as plt
# remplaçables par pylab
from matplotlib import animation

def calc_norme(vecteur):
    N=len(vecteur)
    norme=0
    for i in range(N):
        norme+=(vecteur[i])**2
    norme=norme**(0.5)
    return norme

def schema_explicite(T0):
    N=len(T0) # N points intermédiaires (0 et N+1 exclus)
    ItMax=2000
    T_tous_k=np.zeros((N,ItMax)) # float64 par defaut
    T_tous_k[:,0]=T0[:] # 1ère colonne : température initiale des points xi
    T_tous_k[0,1]=r*Tint+(1-2*r)*T_tous_k[0,0]+r*T_tous_k[1,0] # x1
    for i in range(1,N-1) : # ligne i = point x(i+1)
        T_tous_k[i,1]=r*T_tous_k[i-1,0]+(1-2*r)*T_tous_k[i,0]+r*T_tous_k[i+1,0]
    T_tous_k[N-1,1]=r*T_tous_k[N-2,0]+(1-2*r)*T_tous_k[N-1,0]+r*Text2 # point xN
    norme2=calc_norme(T_tous_k[:,1]-T_tous_k[:,0]) # T_tous_k[:,1] = nd.array d'une
ligne = colonne 1 de T_tous_k
    nbIter=1 # itération n°1 terminée, instant dt
    while (nbIter<ItMax-1 and norme2>0.01): # itérations suivantes
        T_tous_k[0,nbIter+1]=r*Tint+(1-2*r)*T_tous_k[0,nbIter]+r*T_tous_k[1,nbIter] #
x1
        for i in range(1,N-1) : # ligne i pour le point x(i+1)
            T_tous_k[i,nbIter+1]=r*T_tous_k[i-1,nbIter]+(1-2*r)*T_tous_k[i,nbIter]+r*
T_tous_k[i+1,nbIter]
        T_tous_k[N-1,nbIter+1]=r*T_tous_k[N-2,nbIter]+(1-2*r)*T_tous_k[N-1,nbIter]+r*
Text2
        norme2=calc_norme(T_tous_k[:,nbIter+1]-T_tous_k[:,nbIter])
        nbIter+=1 # itération nbIter+1 terminée
    return nbIter,T_tous_k

def CalcTkp1(M,d):
    N=len(d)
    cprim,dprim,u=np.zeros(N),np.zeros(N),np.zeros(N)
    cprim[0]=M[0,1]/M[0,0] # calcul de c'1
    dprim[0]=d[0]/M[0,0] # # calcul de d'1
    for i in range(1,N-1): # jusque N-2 correspondant à c'(N-1) et d'(N-1)
        cprim[i]=M[i,i+1]/(M[i,i]-M[i,i-1]*cprim[i-1])
        dprim[i]=(d[i]-M[i,i-1]*dprim[i-1])/(M[i,i]-M[i,i-1]*cprim[i-1])
    dprim[N-1]=(d[N-1]-M[N-1,N-2]*dprim[N-2])/(M[N-1,N-1]-M[N-1,N-2]*cprim[N-2]) #
calcul de d'N
    u[N-1]=dprim[N-1]
    for i in range(N-2,-1,-1): # de N-2 à 0
        u[i]=dprim[i]-cprim[i]*u[i+1]
    return u

def schema_implicite(T0):
    N=len(T0) # N points intermédiaires entre les pts 0 et N+1
    ItMax=2000
    T_tous_k=np.zeros((N,ItMax))
    T_tous_k[:,0]=T0[:] # 1ère colonne : température initiale des points xi
    # construction de la matrice M
    M=np.zeros((N,N))
    M[0,0],M[0,1]=1+2*r,-r # 1ère ligne
    for i in range(1,N-1):
        M[i,i-1],M[i,i],M[i,i+1]=-r,1+2*r,-r # ligne i
    M[N-1,N-2],M[N-1,N-1]=-r,1+2*r # dernière ligne
    # construction du vecteur v
    v=np.zeros(N)
```

```
    v[0],v[N-1]=Tint,Text2
    T_tous_k[:,1]=CalcTkp1(M,T0+r*v)
    norme2=calc_norme(T_tous_k[:,1]-T_tous_k[:,0])
    nbIter=1 # itération n°1, instant dt
    while (nbIter<ItMax-1 and norme2>0.01):
        T_tous_k[:,nbIter+1]=CalcTkp1(M,T_tous_k[:,nbIter]+r*v)
        norme2=calc_norme(T_tous_k[:,nbIter+1]-T_tous_k[:,nbIter])
        nbIter+=1 # itération k+1 terminée
    return nbIter,T_tous_k

# paramètres système
epais=0.4
Tint,Text1,Text2=20,10,-10
conduc,rho,c=1.65,2150,1000
alpha=rho*c/conduc
# paramètres modelisation
N,dt=60,25
# état initial
a=(Text1-Tint)/epais
b=Tint
# tableau des T initiales
dx=epais/(N+1)
r=dt/(dx**2*alpha)
x=np.linspace(dx,epais-dx,N) # les 2 bornes sont incluses
T0=a*x+b # opération sur un nd.array

# tracés
abscisses=[0]+list(x)+[epais] # concaténation de listes, pour avoir les bords
fig=plt.figure()
plt.xlim(0,epais)
plt.ylim(-10,20)
rep=input("schéma explicite (ne rien taper) implicite (taper n'importe quoi) : ") #
booléen, string vide = False
if rep:
    it,tab_T=schema_implicite(T0)
    temps=it*dt
    h,s=divmod(it*dt,3600)
    plt.title("paroi soumise à un échelon de T\nschéma implicite, régime permanent en
{0}h{1}mn".format(h,s//60))
else :
    if r>=0.5 : print("""simulation explicite risquée : dt trop grand ou dx trop
petit""")
    it,tab_T=schema_explicite(T0)
    temps=it*dt
    h,s=divmod(it*dt,3600)
    plt.title("paroi soumise à un échelon de T\nschéma explicite, régime permanent en
{0}h{1}mn".format(h,s//60))
for i in range(it//100):
    plt.plot(abscisses,[Tint]+list(tab_T[:,100*i])+[Text2]) # la courbe t=0 présente
un point anguleux à cause de la discontinuité de Text

# pour réaliser une animation remplacer les deux lignes précédentes par le code ci-
dessous

#line,=plt.plot(abscisses,[Tint]+list(T0)+[Text2])
#def animate(i): # i = n° de la frame
#    line.set_data(abscisses,[Tint]+list(tab_T[:,i])+[Text2]) # virgule parfois
nécessaire parce qu'il construit une liste
#    return line,
#anim=animation.FuncAnimation(fig,animate,interval=1)
```