```
# -*- coding: utf-8 -*-
"""
Created on Sun Jul  5 08:59:30 2015

@author: arno
"""

# II.B.1.a. Delta x=e/(N+1)

# II.B.1.b. x_i=i*Delta x=i*e/(N+1)

# II.B.2.a. T(x+Delta x,t)=T(x,t) + Delta x. dT/dx(x,t) + (Delta x^2 / 2).d^2 T/
dx^2(x,t)
# + (Delta x^3 / 6).d^3 T/dx^3(x,t) + o( Delta x^3 )

# T(x-Delta x,t)=T(x,t) - Delta x. dT/dx(x,t) + (Delta x^2 / 2).d^2 T/dx^2(x,t)
# - (Delta x^3 / 6).d^3 T/dx^3(x,t) + o( Delta x^3 )

# II.B.2.b. Donc
# d^2 T/dx^2(x,t) = ( T(x+Delta x,t) + T(x-Delta x,t) - 2.T(x,t) )/( Delta x^2 )
 + o( Delta x )

# II.B.2.c.
# l'expression approchée demandée est
# ( T^k_{i+1} + T^k_{i-1} - 2.T^k_i )/( Delta x^2 )

# II.B.2.d. T(x,t+Delta t)=T(x,t) + Delta t. dT/dt(x,t) + o( Delta t )

# II.B.2.e. dT/dt(x,t) = ( T(x,t+Delta t) - T(x,t) )/( Delta t ) + o(1)

# II.B.2.f.
# l'expression approchée demandée est
# ( T^{k+1}_i - T^k_i )/( Delta t )

# II.B.2.g. l'équation (1) donne
# alpha * ( T^{k+1}_i - T^k_i )/( Delta t ) = ( T^k_{i+1} + T^k_{i-1} - 2.T^k_i
)/( Delta x^2 )

# II.B.2.h. on trouve
# T^{k+1}_i=(Delta t / (alpha*Delta x^2))*T^k_{i-1} + (1 -2*(Delta t / (alpha*De
lta x^2)))*T^k_i + (Delta t / (alpha*Delta x^2))*T^k_{i+1}
# donc r=(Delta t / (alpha*Delta x^2))

# II.B.2.i. l'approximation de la dérivée partielle seconde spatiale nécessite d
e ne pas être sur le "bord" donc que 1 <= i <= N
# T^k_0=T_int et T^k_{n+1}=T_ext2

# II.B.2.j.(i)
# def schema_explicite(T0,alpha,e,Delta_t,Tint,Text):
#     ....
#     return nbIter,T_tous_k

# II.B.2.j.(ii)
N=len(T0)
Deltax=e/float(N+1)
r=(Deltat/float(alpha*Deltax**2))
if r>=1/2. :
    print "la condition r<1/2 n'est pas respectée"

# II.B.2.j.(iii)
ItMax=2000
T_tous_k=np.zeros((N,ItMax))
```



```

# II.B.2.j.(iv)
T_tous_k[:,0]=T0

# II.B.2.j.(v)
T_tous_k[0,1]=r*Tint+(1-2*r)*T_tous_k[0,0]+r*T_tous_k[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*Text
for i in range(2,N):
    T_tous_k[i-1,1]=r*T_tous_k[i-2,0]+(1-2*r)*T_tous_k[i-1,0]+r*T_tous_k[i,0]

# II.B.2.j.(vi)
def calc_norme(V):
    N=len(V)
    s=0
    for i in range(N):
        s+=V[i]**2
    return sqrt(s)

# II.B.2.j.(vii)
k=1
while calc_norm(T[:,k]-T[:,k-1])>=10**(-2) and k<ItMax-1:
    k+=1
    T_tous_k[0,k]=r*Tint+(1-2*r)*T_tous_k[0,k-1]+r*T_tous_k[1,k-1]
    T_tous_k[N-1,k]=r*T_tous_k[N-2,k-1]+(1-2*r)*T_tous_k[N-1,k-1]+r*Text
    for i in range(2,N):
        T_tous_k[i-1,k]=r*T_tous_k[i-2,k-1]+(1-2*r)*T_tous_k[i-1,k-1]+r*T_tous_k
[i,k-1]

# II.B.2.j.(viii)
nbIter=k
return nbIter,T_tous_k

# II.B.3.a. l'équation (1) donne
# alpha * ( T^{k+1}_i - T^k_i )/( Delta t ) = ( T^{k+1}_{i+1} + T^{k+1}_{i-1} -
2.T^{k+1}_i )/( Delta x^2 )

# II.B.3.b.on a donc
# T^k_i=-r.T^{k+1}_{i-1}+(1+2r).T^{k+1}_i-r.t^{k+1}_{i+1}
# toujours avec r=(Delta t / (alpha*Delta x^2))

# II.B.3.c.
# M est tridiagonale: sur la diagonale principale tous les coefficients valent 1
+2r
# sur la diagonale supérieure tous les coefficients valent r
# sur la diagonale inférieure tous les coefficients valent r
# v=[Tint,0,.....,0,Text2]

# II.B.3.d.
def CalcTkp1(M,d):
    N=len(d)
    cprime=[ 0 for k in range(N) ]
    cprime[0]=M[0,1]/float(M[0,0]) # Python 2.7
    for i in range(1,N-1):
        cprime[i]=float(M[i,i+1])/(M[i,i]-M[i,i-1]*cprime[i-1]) # Python 2.7
    dprime=[ 0 for k in range(N) ]
    dprime[0]=float(d[0])/M[0,0] # Python 2.7
    for i in range(1,N):
        dprime[i]=float(d[i]-M[i,i-1]*dprime[i-1])/(M[i,i]-M[i,i-1]*cprime[i-1])
 # Python 2.7
    u=[ 0 for k in range(N) ]
    u[-1]=dprime[-1]
    for i in range(N-2,-1,-1):
```

```
        u[i]=dprime[i]-cprime[i]*u[i+1]
    return u

# II.B.3.e.(i)
# def schema_implicite(T0,alpha,e,Delta_t,Tint,Text):
#     ....
#     return nbIter,T_tous_k

# II.B.3.e.(ii)
ItMax=2000
N=len(T0)
T_tous_k=np.zeros((N,ItMax))

# II.B.3.e.(iii)
T_tous_k[:,0]=T0

# II.B.3.e.(iv)
Deltax=e/float(N+1)
r=(Deltat/float(alpha*Deltax**2))
M=zeros((N,N))
for i in range(N):
    M[i,i]=1+2*r
    if i!=0:
        M[i-1,i]=-r
        M[i,i-1]=-r
v=zeros((N,1))
v[0]=Tint
v[N-1]=Text

# II.B.3.e.(v)
T_tous_k[:,1]=CalcTkp1(M,T_tous_k[:,0]+r*v[:,0]) # mauvais résultat avec v à la
place de v[:,0]

# II.B.3.e.(vi)
k=1
while calc_norm(T_tous_k[:,k]-T_tous_k[:,k-1])>=10**(-2) and k<ItMax-1:
    k+=1
    T_tous_k[:,k]=CalcTkp1(M,T_tous_k[:,k-1]+r*v[:,0]) # idem

# II.B.3.e.(vii)
nbIter=k
return nbIter,T_tous_k

# II.C.1.a.
epais=0.4 # 40cm
conduc=1.65
rho=2150
Cp=1000
Tint=20
Text1=10
Text2=-10
N=60
Dt=25

# II.C.1.b.
b=Tint
a=(Text1-Tint)/float(epais) # Python 2.7

# II.C.1.c.
x=np.linspace(0,epais,N+2)

# II.C.1.d.
```



```
T0=a*x+b

# II.C.1.e.
alpha=rho*Cp/float(conduc) # Python 2.7
Delta_x=epais/float(N+1)
r=Dt/(alpha*Delta_x**2)

# II.C.2.a.
def choix(T0,alpha,epais,Delta_t,Tint,Text2):
    x=int(input('schéma explicite choix 0, schéma implicite choix 1'))
    if x==0:
        schema_explicite(T0,alpha,epais,Delta_t,Tint,Text2)
    else:
        schema_implicite(T0,alpha,epais,Delta_t,Tint,Text2)

# II.C.3.a.
[nbIter,T_tous_k]=choix(T0,alpha,epais,Dt,Tint,Text2)
for k in range(nbIter/100): # Python 2.7
    plt.plot(x,T_tous_k[:,k])
#plt.xlabel('abscisse x')
#plt.ylabel('température')
plt.show()

# II.C.3.b.
temps=nbIter*Dt
print "Le régime permanent est atteint au bout de "+str(temps/60.)+" heures."
# si nbIter=ItMax il faut songer à augmenter la valeur de ItMax
```