```
##
## Resolution numerique de l'equation de la chaleur
##
## A partir du sujet de modelisation physique CCP PC 2015
## Version combinaison de plongee
## Transferts conductifs
## Conditions limites Températures imposées
## Schémas explicite et implicite
##
#############################################################

## Importation des modules
############################

import matplotlib.pyplot as plt
import numpy as np


## Definition des fonctions
#############################

def schema_explicite(T0, ItMax =2000, Dt = 0.05,
                     e = 7e-3, Lambda = 0.192, Rho = 960, c = 2180,
                     T_int = 310, T_ext = 283, epsilon = 1e-2):
    '''renvoie le nombre d'iterations effectuees et une matrice de N + 2 lignes contenant les
temperatures a l'instant k en chaque point declare de la paroi, par la methode des differences finies
 en utilisant un schéma explicite.'''
# Question (i)
    Alpha = Rho * c / Lambda
    N = len(T0) - 2
    Dx = e / (N + 1)

    r = Dt / (Alpha * pow(Dx, 2))

    if r >= 0.5 :
        raise Exception('''r = Dt / ( (rho * c / lambda) * Dx^2 ) doit etre inferieur à 0.5.''') #
Question (ii), un simple print etait possible

    T_tous_k = np.zeros((N + 2, ItMax + 1)) # Question (iii)

    for i in range(N + 2):
        T_tous_k[i, 0] = T0[i, 0]   # Conditions initiales
    for k in range(1, ItMax):
        T_tous_k[0, k] = T_int      # Conditions limites x = 0
    for k in range(1, ItMax):
        T_tous_k[N + 1, k] = T_ext  # Conditions limites x = e
    # Question (iv)

    for i in range(1, N + 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] # Question (v)

    k = 2
    condition = True

    while condition and k < ItMax :
        for i in range(1, N + 1):
            T_tous_k[i, k] = r * T_tous_k[i - 1, k - 1] + \
            (1 - 2 * r) * T_tous_k[i, k - 1] + \
            r * T_tous_k[i + 1, k - 1]

        if calc_norme(T_tous_k[:, k] - T_tous_k[:, k - 1]) < epsilon :
            condition = False
        k += 1 # nbIter, Question (vii)

    return (k, T_tous_k)

## CODE A COMPLETER avec vos fonctions de la partie 1 du DM
## Il est fortement recommandé de tester vos fonctions avant de les utiliser
################################


def calc_norme(V):
    '''renvoie la norme 2 du vecteur V de type array. La norme 2 vaut
 sqr ( somme (i de 1 à N) Vi^2 ).'''
    norme = 0
```

```
    for n in V:
        norme += n
    print(norme)
    return pow(abs(norme), 0.5)


def CalcTkp1 (M, d) :
    '''renvoie le vecteur u tel que M.u = d selon l'algorithme de Thomas.
M est une matrice tridiagonale de type array,
d est un vecteur colonne de type array.
Le vecteur u renvoye est aussi un vecteur colonne de type array.'''
    u = np.zeros((N, 1))
    u[N, 0] = (d[N, 0] - M[N, N-1] * d[N-1, 0]) / (M[N, N] - M[N, N-1] * M[N-1, N])

    for i in range(N-1, 1, -1):
        ci = M[i, i+1] / (M[i, i] - M[i, i-1] * M[i-1, i])
        di = (d[i, 0] - M[i, i-1] * d[i-1, 0]) / (M[i, i] - M[i, i-1] * M[i-1, i])
        u[i, 0] = di - ci * u[i+1, 0]
    u[0, 0] = (d[0, 0]/M[0, 0]) - (M[0, 1]/M[0, 0]) * u[1, 0]
    return u

## Corrigé du code de la partie 1 du DM : INUTILE pour la partie 2
################################


def schema_implicite(T0, ItMax =2000, Dt = 0.05, e = 7e-3,
                     Lambda = 0.192, Rho = 960, c = 2180, T_int = 310, T_ext = 283,
                     epsilon = 1e-2):
    '''renvoie le nombre d'iterations effectuees et une matrice de N + 2 lignes
contenant les temperatures a l'instant k en chaque point declare de la paroi,
par la methode des differences finies en utilisant un schéma implicite.'''
    Alpha = Rho * c / Lambda
    N = len(T0) - 2
    Dx = e / (N + 1)
    r = Dt / (Alpha * pow(Dx, 2))

    if r >= 0.5 :
        raise Exception('''r = Dt / ( (rho * c / lambda) * Dx^2 ) doit etre
inferieur à 0.5.''')

    T_tous_k = np.zeros((N + 2, ItMax + 1))

    for i in range(N + 2):
        T_tous_k[i, 0] = T0[i, 0]      # Conditions initiales
    for k in range(1, ItMax):
        T_tous_k[0, k] = T_int       # Conditions limites x = 0
    for k in range(1, ItMax):
        T_tous_k[N + 1, k] = T_ext  # Conditions limites x = e
    # Question (iv)

    # Definition de M
    M = np.zeros((N, N))

    M[0, 0], M[0, 1] = 1. + 2. * r, -r
    for ligne in range(1, N - 1) :
        M[ligne, ligne - 1], M[ligne, ligne], M[ligne, ligne + 1] = \
                 -r, 1. + 2. * r, -r
    M[-1, -2], M[-1, -1] = -r, 1. + 2. * r

    # Definition de v
    v = np.zeros((N, 1))

    v[0, 0], v[-1, 0] = T_int, T_ext

    # Calcul de T^1
    d = T0[1: N + 1] + r * v
    res = CalcTkp1(M, d)
    for i in range(1, N + 1):
        T_tous_k[i, 1] = res[i - 1]

    # Calcul des T^k
    k = 2
    condition = True

    while condition and k < ItMax :
        D = np.zeros((N, 1))
        for i in range(1, N + 1):
```

```
            D[i - 1] = T_tous_k[i, k - 1] + r * v[i - 1]
        res = CalcTkp1(M, D)
        for i in range(1, N + 1):
            T_tous_k[i, k] = res[i - 1]

        if calc_norme(T_tous_k[:, k] - T_tous_k[:, k - 1]) < epsilon :
            condition = False
        k += 1

    return (k, T_tous_k)


## Transfert conducto-convectif : A COMPLETER
################################

def sch_exp_cond_conv(T0, ItMax =2000, Dt = 0.05, \
e = 7e-3, Lambda = 0.192, Rho = 960, c = 2180, \
T_int = 310, T_ext = 283, epsilon = 1e-2, h = 200):
    '''renvoie le nombre d'iterations effectuees et une matrice de N + 2 lignes contenant les
temperatures a l'instant k en chaque point declare de la paroi, par la methode des differences finies
 en utilisant un schéma explicite.
    Transferts conducto-convectifs'''
    Alpha = Rho * c / Lambda
    N = len(T0) - 2
    Dx = e / (N + 1)
    r = Dt / (Alpha * pow(Dx, 2))

    if r >= 0.5:
        raise Exception('''r = Dt / ( (rho * c / lambda) * Dx^2 ) doit etre
    inferieur à 0.5.''')

    T_tous_k = np.zeros((N + 2, ItMax + 1))

    for i in range(N + 2):
        T_tous_k[i, 0] = T0[i, 0]  # Conditions initiales
    for k in range(1, ItMax):
        T_tous_k[0, k] = T_int  # Conditions limites x = 0
    T_tous_k[N+1, 0] = (Lambda * T_tous_k[N, 0] + h * Dx * T_ext) / (Lambda + h * Dx)
    # Question (I.2.b.1)

    # Calcul des T^k
    k = 1
    condition = True

    while condition and k < ItMax:
        for i in range(0, N + 1):
            T_tous_k[i, k] = r * T_tous_k[i-1, k-1] + (1 - 2*r) * T_tous_k[i, k-1] + r * T_tous_k[i+1
, k-1]
        T_tous_k[N+1, k] = (Lambda * T_tous_k[N, k] + h * Dx * T_ext) / (Lambda + h * Dx)
                                                                        # Modification de la
T de surface à k

        if calc_norme(T_tous_k[:, k] - T_tous_k[:, k - 1]) < epsilon:
            condition = False
        k += 1
    # Question (I.2.b.2)

    return (k, T_tous_k)



## Transfert rayonnement : A COMPLETER
################################

def sch_exp_ray(T0, ItMax =2000, Dt = 0.05,
                e = 7e-3, Lambda = 0.192, Rho = 960, c = 2180,
                T_int = 310, T_eau2 = 283, epsilon = 1e-2, sigma = 5.67e-8):
    '''renvoie le nombre d'iterations effectuees et une matrice de N + 2 lignes contenant les
temperatures a l'instant k en chaque point declare de la paroi, par la methode des differences finies
 en utilisant un schéma explicite.
    Transferts avec rayonnement'''

    Alpha = Rho * c / Lambda
    N = len(T0) - 2
    Dx = e / (N + 1)
```

```
    r = Dt / (Alpha * pow(Dx, 2))

    if r >= 0.5 :
        raise Exception('''r = Dt / ( (rho * c / lambda) * Dx^2 ) doit etre inferieur à 0.5.''') #
Question (ii), un simple print etait possible

    T_tous_k = np.zeros((N + 2, ItMax + 1)) # Question (iii)


    for i in range(N + 2):
        T_tous_k[i, 0] = T0[i, 0]   # Conditions initiales
    for k in range(1, ItMax):
        T_tous_k[0, k] = T_int      # Conditions limites x = 0
    for k in range(1, ItMax):
        T_tous_k[N + 1, k] = T_eau2 # Conditions limites x = e




    # Question (iv)

    for i in range(1, N + 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] # Question (v)

    k = 2
    condition = True

    while condition and k < ItMax :
        for i in range(1, N + 1):
            T_tous_k[i, k] = r * T_tous_k[i - 1, k - 1] + \
            (1 - 2 * r) * T_tous_k[i, k - 1] + \
            r * T_tous_k[i + 1, k - 1]
            T_tous_k[i, k] = r * T_tous_k[i-1, k-1] + (1 - 2*r) * T_tous_k[i, k-1] + r * T_tous_k[i+1
, k-1]
        j = 0
        Tek = [T_tous_k[N + 1, k - 1], 1]
        ldx = Lambda / Dx
        while Tek[-1] - Tek[-2] >= 0.01:
            res = Tek[j] - (sigma * (Tek[j] ** 4 + T_eau2 ** 4) + ldx * (Tek[j] - T_tous_k[N, k
])) / (
                        4 * sigma * Tek[j] ** 3 + ldx)
            Tek[j+1] = res
            j += 1
        T_tous_k[N + 1, k] = Tek[-1] # On prend en compte le rayonnement


        if calc_norme(T_tous_k[:, k] - T_tous_k[:, k - 1]) < epsilon :
            condition = False
        k += 1 # nbIter, Question (vii)

    return (k, T_tous_k)


## Definition des constantes
##############################

ItMax = 2000

epais = 7e-3
conduc = 0.192
rho = 960
Cp = 2180
Tint = 310
Text1 = 293
Text2 = 283
epsilon = 1e-2
N = 60
Dt = 0.05

a = (Text1 - Tint) / epais
b = Tint

Dx = epais / (N + 1)
x = np.linspace(0, epais, N + 2)
```

```
T0 = np.zeros((N + 2, 1))
T0[0, 0] = Tint
for i in range(1, N + 1):
    T0[i, 0] = a * x[i] + b
T0[N + 1, 0] = Text1

alpha = rho * Cp / conduc
r = Dt / (alpha * pow(Dx, 2))


## Programme principal : A COMPLETER
########################



## Tracé des figures pour l'allure des courbes : A COMPLETER
###################################

def courbe_conducto_convection():
    (nb_iter, T_tous_k) = sch_exp_cond_conv(T0, ItMax =2000, Dt = 0.05, e = 7e-3, Lambda = 0.192, Rho
 = 960, c = 2180, \
                                        T_int = 310, T_ext = 283, epsilon = 1e-2, h = 200)
    for i in range(nb_iter//100):
        plt.plot(x, T_tous_k[:, 100*i], label = 't = '+str(i*5)+'s')
    plt.xlabel('distance $x$')
    plt.ylabel('Température')
    #plt.legend()
    plt.title('Profils de température $-$ cas de la conducto-convection')
    plt.show()

def courbe_rayonnement():
    (nb_iter, T_tous_k) = sch_exp_ray(T0, ItMax =2000, Dt = 0.05, e = 7e-3, Lambda = 0.192, Rho = 960
, c = 2180, \
                                        T_int = 310, T_eau2 = 283, epsilon = 1e-2, sigma = 5.670374e-8)
    for i in range(nb_iter//100):
        plt.plot(x, T_tous_k[:, 100*i], label = 't = '+str(i*5)+'s')
    plt.xlabel('distance $x$')
    plt.ylabel('Température')
    #plt.legend()
    plt.title('Profils de température $-$ cas du rayonnement')
    plt.show()

courbe_conducto_convection()
courbe_rayonnement()
```