# Themodynamique
# Corrigé Série Supplémentaire 2:
# Equation de la chaleur

S. Guinchard*
*Section de Physique, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Suisse*
(Supervised by Prof. J.P. Ansermet)†
(Dated: May 25, 2022)

## I. EXERCICE 1: THERMALISATION DE DEUX BLOCS EN CONTACT THERMIQUE

### A. Questions analytiques

1. En régime stationnaire, montrer que les puissances thermiques exercées par l'environnement sur le premier bloc, par le premier bloc sur le deuxième, et par le deuxième bloc sur l'environnement sont égales et écrites comme,

$$P_Q \equiv P_Q^{(01)} = P_Q^{(12)} = P_Q^{(20)}. \tag{1}$$

**Solution:** En régime stationnaire, l'entropie de chaque bloc est constante,

$$\dot{S}_i = 0 \qquad\qquad i = 1, 2. \tag{2}$$

Les blocs sont des systèmes simples. Par conséquent, le taux de production d'entropie de chaque bloc est nul. Dans ce cas, la variation d'entropie de chaque sous-système est donnée par

$$\begin{aligned} \dot{S}_1 &= \frac{P_Q^{(01)} - P_Q^{(12)}}{T_1(S_1)} = 0 \implies P_Q^{(01)} = P_Q^{(12)}, \\ \dot{S}_2 &= \frac{P_Q^{(12)} - P_Q^{(20)}}{T_2(S_2)} = 0 \implies P_Q^{(12)} = P_Q^{(20)}. \end{aligned} \tag{3}$$

Par conséquent

$$P_Q \equiv P_Q^{(01)} = P_Q^{(12)} = P_Q^{(20)} \tag{4}$$

2. Considérez maintenant que les deux blocs sont constitués de $N_1$ et $N_2$ moles de métal respectivement. Ils sont initialement séparés et à températures $T_1$ et $T_2$. Lorsqu'ils sont mis en contact, ils atteignent progressivement l'équilibre thermique.

   i) Déterminer la température finale $T_f$ du système de deux blocs à l'équilibre thermique.

   **Solution:** Etant donné que le système est isolé, l'énergie interne est constante. Par conséquent, la variation d'énergie interne $\Delta U$ du système est nulle,

$$\Delta U = \Delta U_1 + \Delta U_2 = 0. \tag{5}$$

* salomon.guinchard@epfl.ch
† Laboratoire de Physique des Matériaux Nanostructurés, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Suisse

La variation d'énergie interne de chaque bloc est donnée par

$$\Delta U_i = 2N_j R \int_{T_j}^{T_f} dT = 3N_j R(T_f - T_j). \tag{6}$$

Ainsi, on obtient la température finale du système:

$$T_f = \frac{N_1 T_1 + N_2 T_2}{N_1 + N_2}. \tag{7}$$

ii) Calculer la variation d'entropie $\Delta S$ du système de deux blocs lors du processus qui l'amène à l'équilibre thermique.

**Solution:** La variation d'entropie $\Delta S$ du système est exprimée comme

$$\Delta S = \Delta S_1 + \Delta S_2. \tag{8}$$

La variation infinitésimale d'entropie de chaque bloc s'écrit

$$dS_i = \frac{dU_i}{T_i} = 3N_i R \frac{dT_i}{T_i}, \tag{9}$$

ce qui implique que la variation d'entropie de chaque bloc au cours du processus est de la forme

$$\Delta S_i = 3N_i R \int_{T_i}^{T_f} \frac{dT}{T} = 3N_i R \ln\left(\frac{T_f}{T_i}\right). \tag{10}$$

Ainsi, la variation d'entropie lors du processus est donnée par

$$\Delta S = \sum_{i=1,2} 3N_i R \ln\left(\frac{T_f}{T_i}\right). \tag{11}$$

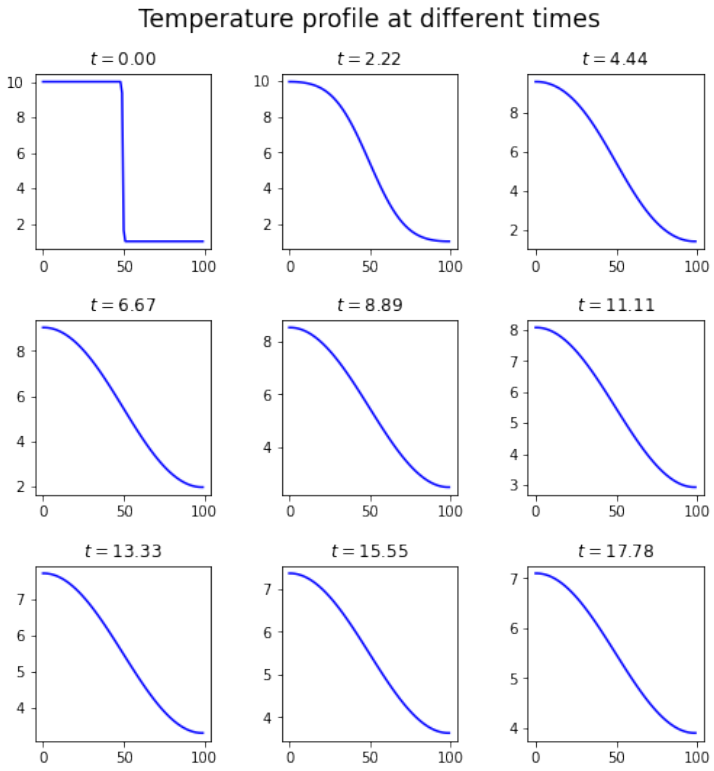
## II. IMPLÉMENTATION NUMÉRIQUE DU PROBLÈME

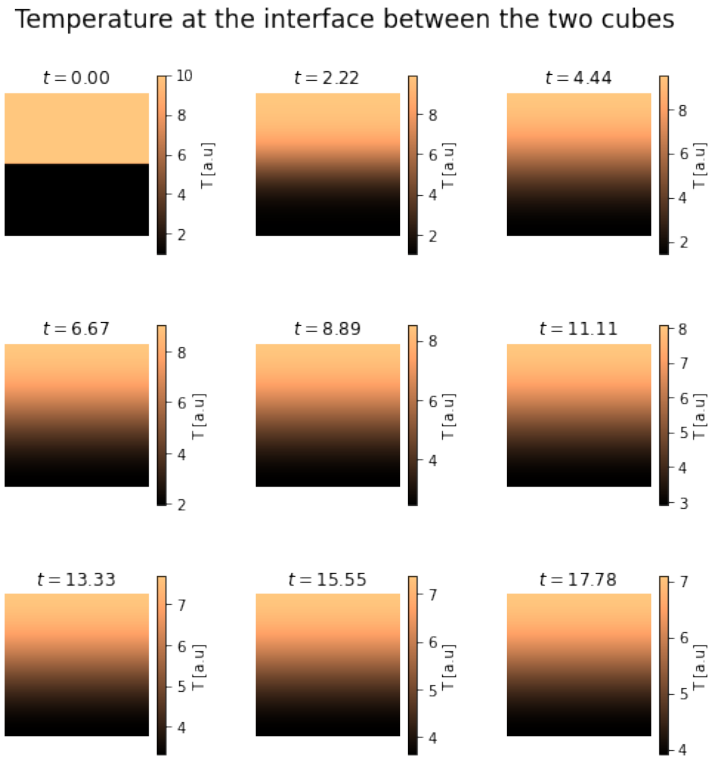
Ci dessous, le corrigé du code *Jupyter Notebook* pour des exemples de paramètres. Vous pouvez évidemment changer ceux-ci pour vous familiariser avec le fonctionnement du code.

[22]:
```
################################################
######    Thermalisation de deux blocs     #####
################################################

import numpy as np
import matplotlib.pyplot as plt


########################################
######    NUMERICAL PARAMETERS     #####
########################################

# THE NUMERICAL PARAMETERS CAN BE CHANGED
# TODO: change parameters

size = 100          # size of the 2D grid
dx = 2. / size      # space step
T = 20              # total time (default 40)
dt = .001           # time step
n = int(T / dt)     # number of iterations

########################################
# Initialize a container for the profile
# at the interface
y = np.arange(start=0, stop=size, step=1)

# Initialize a container for the temperature distribution on the grid
U = np.zeros((size, size))
V = np.zeros((size, size))

# TODO: if desired change values below
for i in range(50):
    for j in range(100):
        U[i][j]=10        #T1 value

### ^^ TODO: CHANGE VALUE of U[i,j] for T1 ^^ ###
#################################################

for i in range(50,100):
    for j in range(100):
        U[i][j]=1         #T2 value

### ^^ TODO: CHANGE VALUE of U[i,j] for T2 ^^ ###
#################################################


############################################
# DEF OF LAPLACIAN OPERATOR W/ FINITE DIFF #
############################################

def laplacian(Z):
    Ztop = Z[0:-2, 1:-1]
    Zleft = Z[1:-1, 0:-2]
    Zbottom = Z[2:, 1:-1]
    Zright = Z[1:-1, 2:]
    Zcenter = Z[1:-1, 1:-1]
    return (Ztop + Zleft + Zbottom + Zright -
            4 * Zcenter) / dx**2
```

```
#Function that displays the temperature on the grid
def show_TEMPERATURE(U, ax=None):
    graph =  ax.imshow(U, cmap=plt.cm.copper,
              interpolation='bilinear',
              extent=[-1, 1, -1, 1])
    ax.set_axis_off()
    fig.colorbar(graph, ax=ax, label = 'T [a.u]')

#Function that displays the temperature profile
def show_profile(U, ax=None):
    graph = ax.plot(y,U[:,int(size//2)], color='blue')

###### FIGURE TO RECEIVE TEMP PROFILE #######
fig, axes = plt.subplots(3, 3, figsize=(8, 8))
fig.suptitle("Temperature profile at different times" , fontsize = 17)
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)


#### FIGURE TO RECEIVE TEMP DISTRIBUTION #####
fig, axes2 = plt.subplots(3, 3, figsize=(8, 8))
plt.xlabel('X', fontsize=20, fontweight='bold')
plt.ylabel('T', fontsize=20, fontweight='bold')

fig.suptitle("Temperature at the interface between the two cubes" , fontsize = 17)
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)

##### FIG FOR TEMP_PROFILE CUMUL ######
fig, axes3 = plt.subplots(1,1, figsize=(7, 7))
fig.suptitle("Superposition of temperature profiles" , fontsize = 17)
plt.xlabel('X', fontsize=20, fontweight='bold')
plt.ylabel('T', fontsize=20, fontweight='bold')

##### FIG FOR TEMP_PROFILE CURSOR ######
#fig, axes4 = plt.subplots(1,1, figsize=(8, 8))

step_plot = n // 9


###################################################
# We solve the PDE with finite differences       #
###################################################
for i in range(n):
    # We compute the Laplacian of u and v.
    deltaU = laplacian(U)

    # We take the values of u and v inside the grid.
    Uc = U[1:-1, 1:-1]
    Vc = V[1:-1, 1:-1]
    # We update the variables.
    U[1:-1, 1:-1]= \
```

```
        Uc + dt * (a * deltaU )

    # Neumann conditions: derivatives at the edges
    # are null. (ZERO GRADIENT AT THE BOUNDARY)
    for Z in (U, V):
        Z[0, :] = Z[1, :]
        Z[-1, :] = Z[-2, :]
        Z[:, 0] = Z[:, 1]
        Z[:, -1] = Z[:, -2]

    # We plot the state of the system at
    # 9 different times.
    if i % step_plot == 0 and i < 9 * step_plot:
        ax = axes.flat[i // step_plot]
        ax2 = axes2.flat[i // step_plot]
        ax3 = axes3
        show_profile(U,ax=ax)
        ax.set_title(f'$t={i * dt:.2f}$')
        show_TEMPERATURE(U, ax=ax2)
        ax2.set_title(f'$t={i * dt:.2f}$')
        show_profile(U,ax=ax3)



plt.show()
```



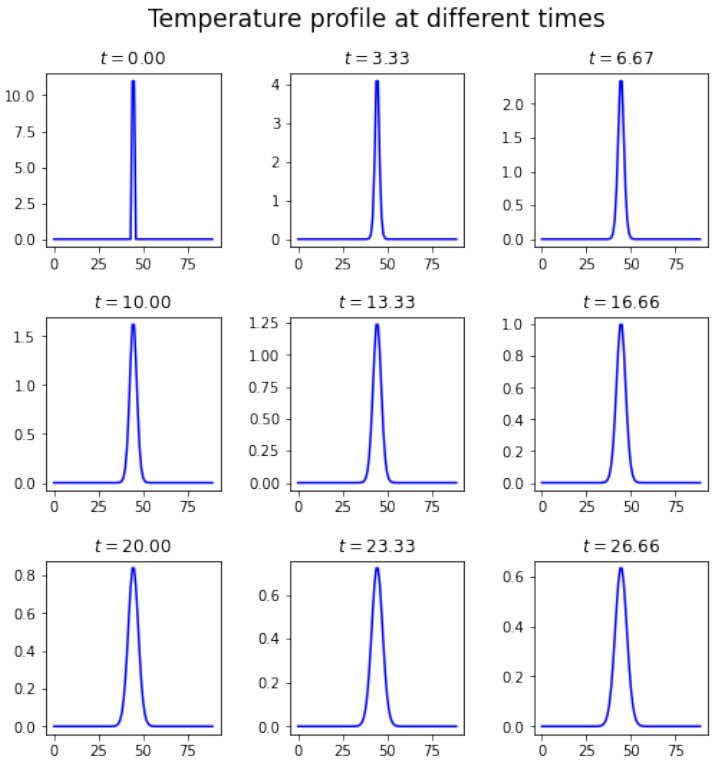

Temperature at the interface between the two cubes
t = 0.00
t = 2.22
t = 4.44
t = 6.67
t = 8.89
t = 11.11
t = 13.33
t = 15.55
t = 17.78
T [a.u.]

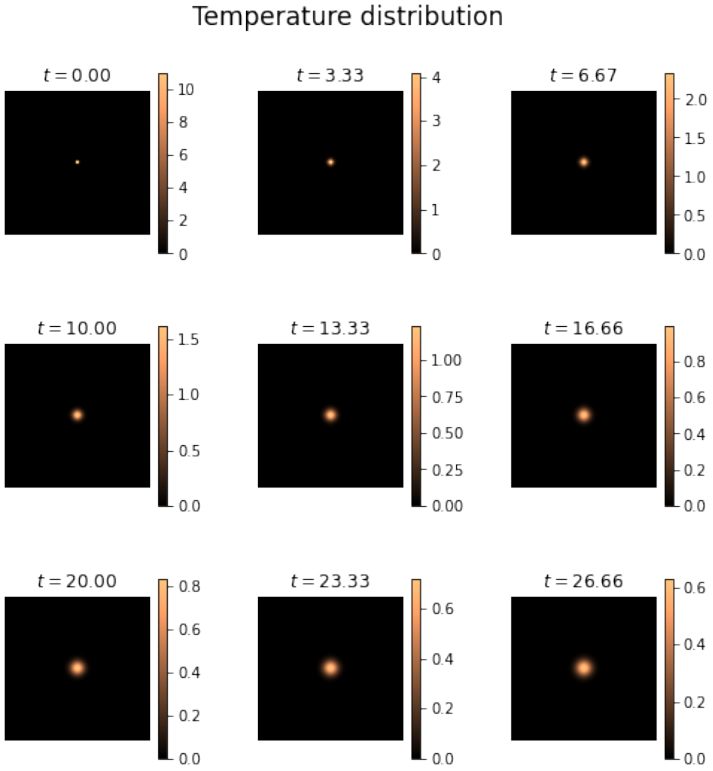
[3]:
```
########################################
# HEAT DIFFUSION ON A 2D SQUARE DOMAIN #
########################################


#Defines the parameters of the simulation#
##########################################


######### size of the 2D grid ###########
lect_size = input("Please enter an (even) integer - Size of the 2D grid (default:␣
↪100):\n")
size = int(lect_size)
dx = 2. / size  # space step

#Initialize a container for heat distribution
U = np.zeros((size, size))
V = np.zeros((size, size))

#Initialize the temperature of the heat source
temp = input("Give the temperature of the heat source (in arbitrary units - default:
↪ 10):\n")

######### simulation time  ###########
lect_time = input("Simulation time (recommended of about 20 to 40):\n")
T = float(lect_time)  # total time
dt = .001  # time step small enough to ensure stability of the numerical scheme
n = int(T / dt)  # number of iterations

x = np.arange(start=0, stop=size, step=1)

########### Initial heat distribution - square source ############
lect = input("Please enter an integer - half range of the heat source (the more␣
↪narrow, the more the initial distribution looks like a delta function): \n")
value = int(lect)
print(f'Width of the heat source: {2*value} meshpoints')

length = float(size)
for i in range (int(length/2)-value,int(length/2)+value):
    for j in range (int(length/2)-value,int(length/2)+value):
        U[i][j]=temp


############### Diffusion coefficient a ####################
lect_a = input("Diffusion coefficient - order of 10-4 to 10-6 recommended (default␣
↪3e-4):\n") # diffusion coefficient
a = float(lect_a)
#a = 2.8e-4 # diffusion coefficient


############################################
# DEF OF LAPLACIAN OPERATOR W/ FINITE DIFF #
############################################


def laplacian(Z):
    Ztop = Z[0:-2, 1:-1]
    Zleft = Z[1:-1, 0:-2]
    Zbottom = Z[2:, 1:-1]
    Zright = Z[1:-1, 2:]
```

```
    Zcenter = Z[1:-1, 1:-1]
    return (Ztop + Zleft + Zbottom + Zright -
            4 * Zcenter) / dx**2


#Function that displays the temperature on the grid
def show_temperature(U, ax=None):
    graph = ax.imshow(U, cmap=plt.cm.copper,
              interpolation='bilinear',
              extent=[-1, 1, -1, 1])
    ax.set_axis_off()
    fig.colorbar(graph, ax=ax)
    #fig.colorbar(graph, ax=ax, orientation='horizontal')

def show_profile(U, ax=None):
    graph = ax.plot(x,U[:,int(size//2)], color='blue')


step_plot = n // 9


###### FIGURE TO RECEIVE TEMP PROFILE #######
fig, axes = plt.subplots(3, 3, figsize=(8, 8))
fig.suptitle("Temperature profile at different times" , fontsize = 17)
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)

#### FIGURE TO RECEIVE TEMP DISTRIBUTION #####
fig, axes2 = plt.subplots(3, 3, figsize=(8, 8))
fig.suptitle("Temperature distribution" , fontsize = 17)
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)

##### FIG FOR TEMP_PROFILE CUMUL ######
fig, axes3 = plt.subplots(1,1, figsize=(8, 8))
fig.suptitle("Superposition of temperature profiles" , fontsize = 17)
plt.xlabel('X', fontsize=20, fontweight='bold')
plt.ylabel('T', fontsize=20, fontweight='bold')

step_plot = n // 9
#####################################################
# We solve the PDE with finite differences       #
#####################################################

for i in range(n):
    # We compute the Laplacian of u.
    deltaU = laplacian(U)

    # We take the values of u inside the grid.
    Uc = U[1:-1, 1:-1]

    # We update the variables.
```

```
    U[1:-1, 1:-1] = Uc + dt * (a * deltaU)


    # Neumann conditions: derivatives at the edges
    # are null (values at the boundary are set constant)
    for Z in (U,V):
        Z[0, :] = Z[1, :]
        Z[-1, :] = Z[-2, :]
        Z[:, 0] = Z[:, 1]
        Z[:, -1] = Z[:, -2]

    # We plot the state of the system at
    # 9 different times.
    if i % step_plot == 0 and i < 9 * step_plot:
        ax = axes.flat[i // step_plot]
        ax2 = axes2.flat[i // step_plot]
        ax3 = axes3
        show_profile(U,ax=ax)
        ax.set_title(f'$t={i * dt:.2f}$')
        show_temperature(U, ax=ax2)
        ax2.set_title(f'$t={i * dt:.2f}$')
        show_profile(U,ax=ax3)


plt.show()
```

```
Please enter an (even) integer - Size of the 2D grid (default: 100):
 90
Give the temperature of the heat source (in arbitrary units - default: 10):
 11
Simulation time (recommended of about 20 to 40):
 30
Please enter an integer - half range of the heat source (the more narrow, the
more the initial distribution looks like a delta function):
 1
 Width of the heat source: 2 meshpoints
 Diffusion coefficient - order of 10-4 to 10-6 recommended (default 3e-4):
 1e-4
```


Temperature profile at different times
t = 0.00
t = 3.33
t = 6.67
t = 10.00
t = 13.33
t = 16.66
t = 20.00
t = 23.33
t = 26.66


Temperature distribution
t = 0.00
t = 3.33
t = 6.67
t = 10.00
t = 13.33
t = 16.66
t = 20.00
t = 23.33
t = 26.66