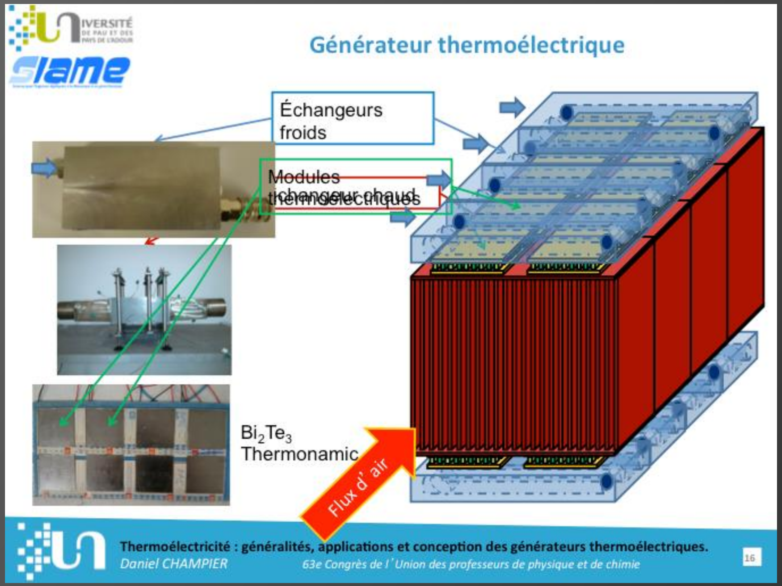
TIPE 2018-2019

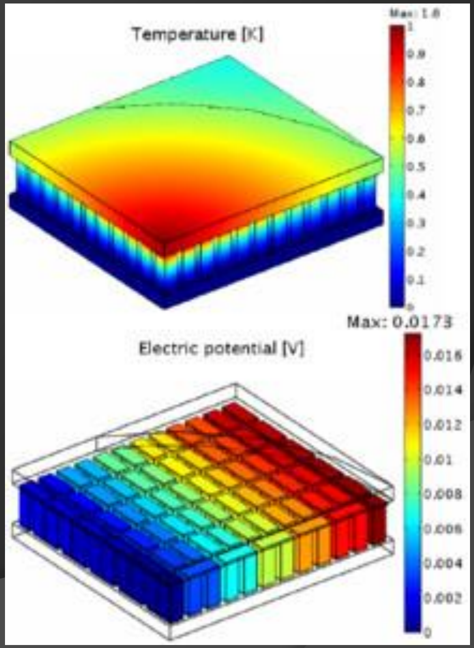
Thème: Transport

Num Scei:22684

Etude du couplage des transports thermiques et électriques:

PROBLÉMATIQUE :COMMENT CONVERTIR DE MANIÈRE EFFICACE L'ÉNERGIE THERMIQUE EN ÉNERGIE ÉLECTRIQUE?



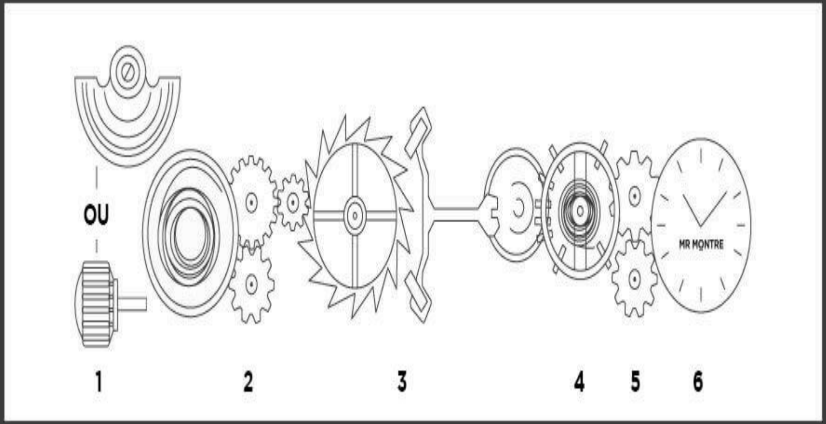


Plan:

- Introduction aux effets thermoélectriques: effet Seebeck
- Couplage et équations d’Onsager
- Implémentation avec python

Contributions:

- Rencontre avec Mr Edward Tentart professeur de physique au lycée Saint-Louis pendant le mois de Janvier et Février.
- Implémentation d'un programme informatique en langage ''python'' en vue de résoudre des équations aux dérivées partielles.



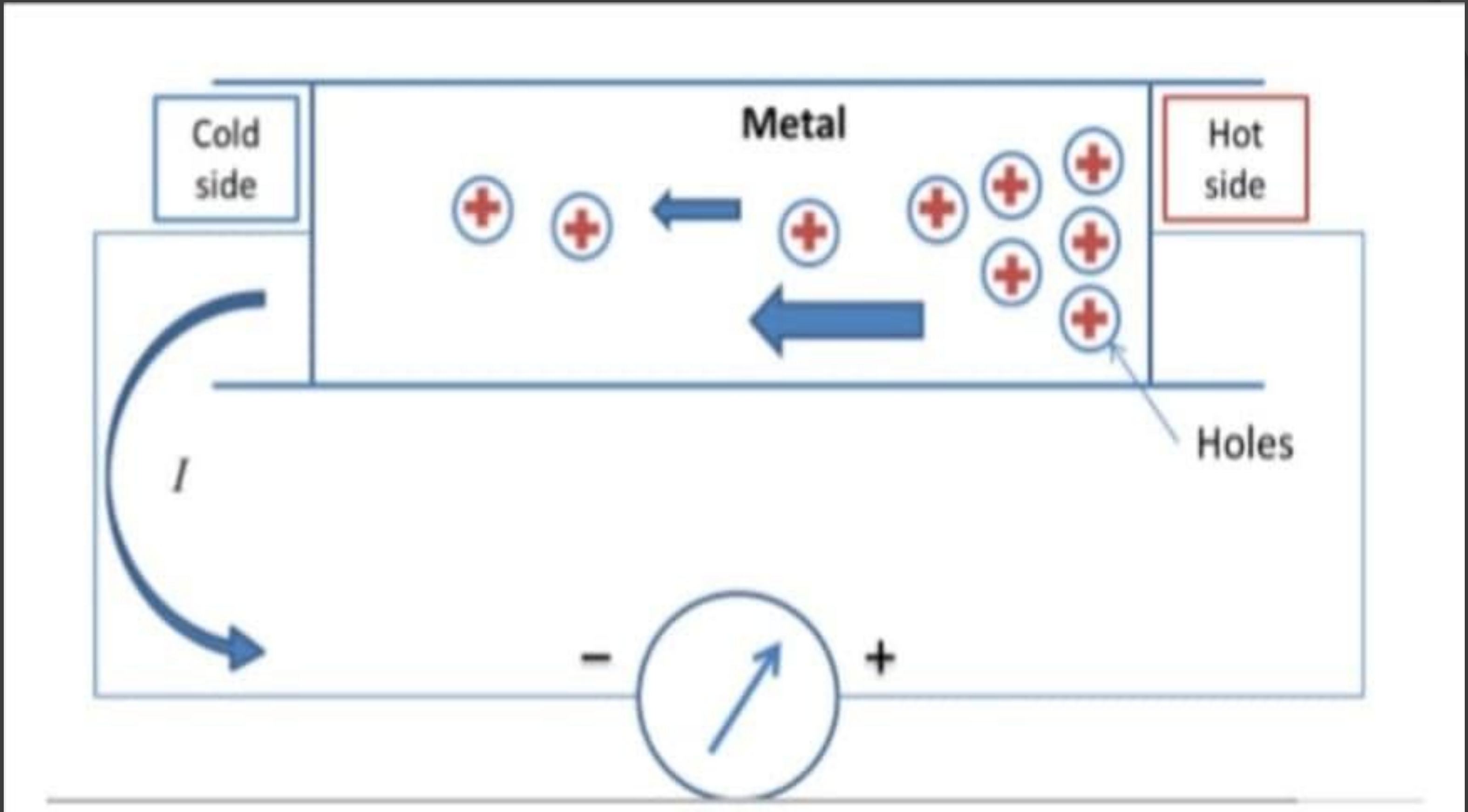
Montre mécanique
Méthode de rechargement: fonctionnement via l’énergie emmagasinée dans un ressort spiral.
Inconvénient: nécessité de charger la montre chaque jour.

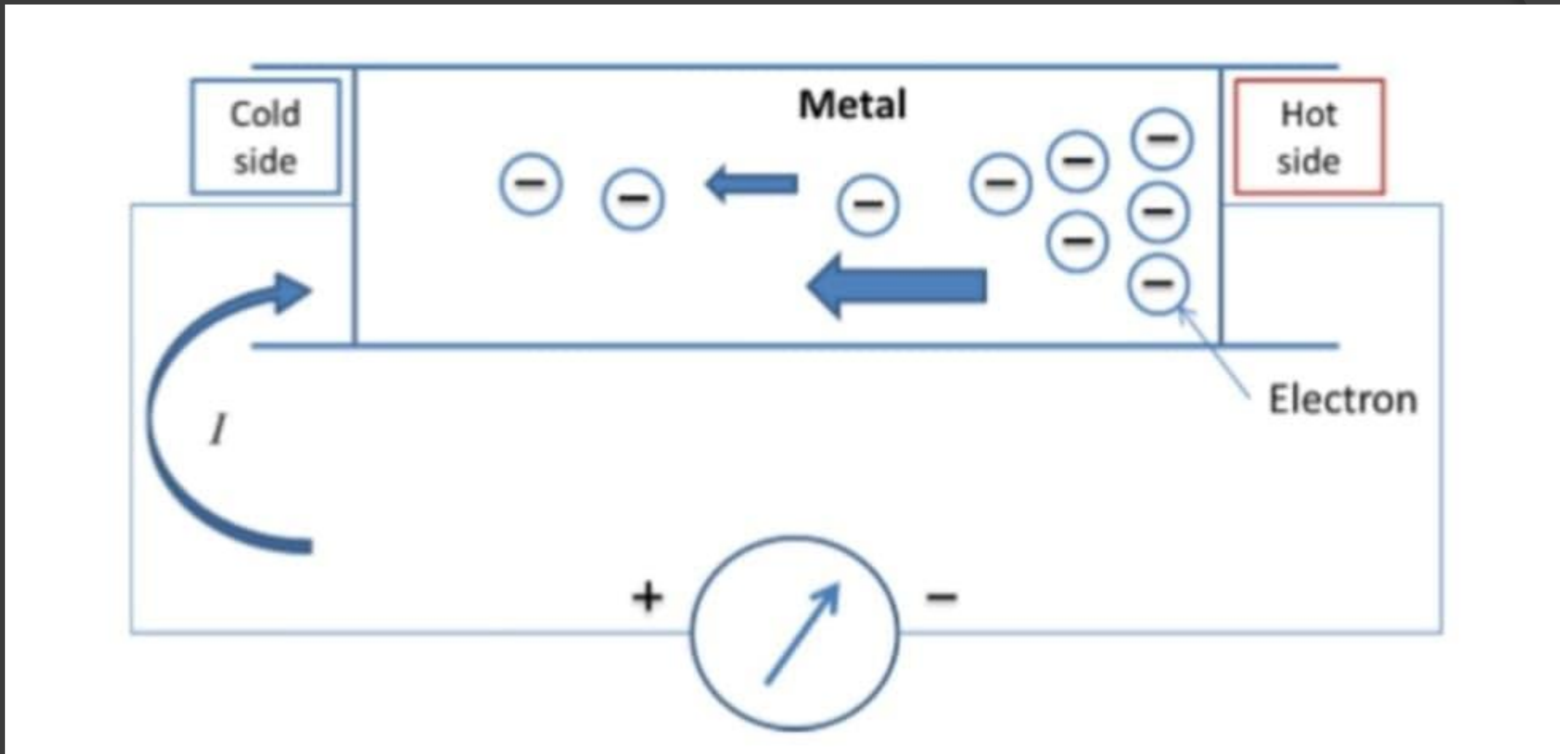
Montre à piles
Inconvénient: matériaux toxiques au corps humain : risque de contamination avec des éléments nocifs en contact avec la peau.

Montre pendule inertiel:
Montre de luxe trop chère

➡ Nécessité de trouver une autre source d’énergie plus propre, plus adéquate, et bon marché.

- Existence d’un gradient de température entre le corps humain(T=37°C)et la montre (température de l’environnement chaud ou froid).
- Question: Comment exploiter ce gradient de température pour générer de l’énergie?
- Réponse: utilisation de l’effet Seebeck.


Cold side
Metal
Hot side
Holes
I
−
+



Existence d’une relation de proportionnalité :

$\Delta V = \alpha \, \Delta T$

Comment peut-on déterminer α ?

La théorie des phénomènes de transport:

Loi d’Ohm:
Dans un conducteur ohmique un gradient de potentiel électrique induit un courant électrique:
c’est la loi d’ohm locale:

$\vec{J} = -\gamma \, \overrightarrow{grad} \, v$

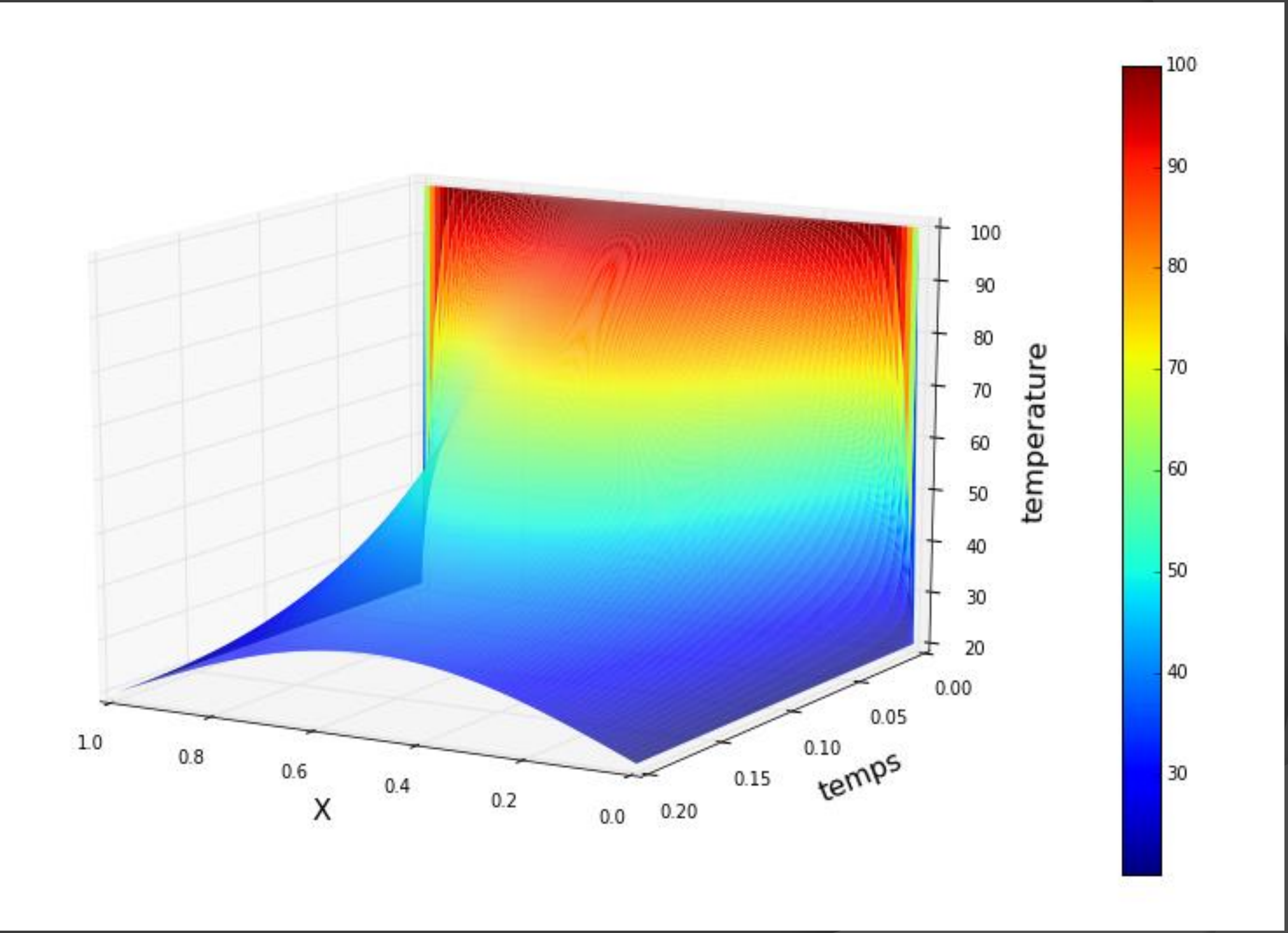
Loi de Fourrier:
Dans un milieu solide un gradient de température induit un flux de chaleur:
c’est la loi de Fourrier:

$\vec{J} = -\lambda \, \overrightarrow{grad} \, T$

couplage

Généralisation avec Onsager: Introduction de coefficient de couplage

$$
\begin{aligned}
J_u &= L_{uu}\nabla\left(\frac{1}{T}\right) + L_{ur}\nabla\left(-\frac{m}{T}\right) \\
J_r &= L_{ru}\nabla\left(\frac{1}{T}\right) + L_{rr}\nabla\left(-\frac{m}{T}\right)
\end{aligned}
$$



X
1.0
0.8
0.6
0.4
0.2
0.0
temps
0.00
0.05
0.10
0.15
0.20
temperature
20
30
40
50
60
70
80
90
100

```
"""Example of matrix formulation of 1D finite difference schemes"""

# Import Pylab
import numpy as np
import matplotlib.pyplot as plt

# For sparse matrices
import scipy.sparse as sp
from scipy.sparse.linalg.dsolve import spsolve

# To estimate execution time
from time import time


N=100              # Number of points in the domain (in each direction)
dx = 1./(N-1);     # Space spacing
x = np.linspace(0.0,1.0,N)

# Definition of the 1D Lalace operator
data = [np.ones(N),-2*np.ones(N),np.ones(N)]      # Diagonal terms
offsets = np.array([-1,0,1])                      # Their positions
LAP = sp.dia_matrix( (data,offsets), shape=(N,N))

# To plot the matrix
print(LAP.todense())



# Number of non-null elements in the 1D Laplace operator
#   print 'Number of non-null elements',LAP.nnz
```

```
# Number of non-null elements in the 1D Laplace operator
#   print 'Number of non-null elements',LAP.nnz

# To plot the structure of the sparse matrix
plt.figure()
plt.spy(LAP)
plt.draw()

f = -np.ones(N)*dx**2  # Right hand side

# Solving the linear system
t = time(); T = spsolve(LAP,f); print('temps sparse=',time()-t)

# In order to compare with the full resolution
LAPfull = LAP.todense()
t = time(); T2 = np.linalg.solve(LAPfull,f); print('temps full = ',time()-t)


# Plotting
plt.figure()
plt.plot(x,T,'ko')
plt.xlabel(u'$x$', fontsize=26)
plt.ylabel(u'$T$', fontsize=26, rotation=0)
plt.show()
```

```
"""2D Heat equation using finite differences"""

import numpy as np
import matplotlib.pyplot as plt

# PHYSICAL PARAMETERS
K = 0.5       #Diffusion coefficient
Lx = 1.0      #Domain size x
Ly = 1.0      #Domain size y
Time = 0.4 #Integration time
S = 1.0       #Source term

# NUMERICAL PARAMETERS

NT = 2000          #Number of time steps
NX = 50            #Number of grid points in x
NY = 50            #Number of grid points in y
dt = Time/NT       #Grid step (time)
dx = Lx/(NX-1) #Grid step in x (space)
dy = Ly/(NY-1) #Grid step in y (space)

xx = np.linspace(0,Lx,NX)
yy = np.linspace(0,Ly,NY)

plt.ion()
plt.figure()

### MAIN PROGRAM ###

T = np.zeros((NX,NY))
RHS = np.zeros((NX,NY))
```

```
NT = 2000       #Number of time steps
NX = 50         #Number of grid points in x
NY = 50         #Number of grid points in y
dt = Time/NT    #Grid step (time)
dx = Lx/(NX-1)  #Grid step in x (space)
dy = Ly/(NY-1)  #Grid step in y (space)

xx = np.linspace(0,Lx,NX)
yy = np.linspace(0,Ly,NY)

plt.ion()
plt.figure()

### MAIN PROGRAM ###

T = np.zeros((NX,NY))
RHS = np.zeros((NX,NY))

# Main loop
for n in range(0,NT):
   RHS[1:-1,1:-1] = dt*K*( (T[:-2,1:-1]-2*T[1:-1,1:-1]+T[2:,1:-1])/(dx**2)  \
                      + (T[1:-1,:-2]-2*T[1:-1,1:-1]+T[1:-1,2:])/(dy**2) )
   T[1:-1,1:-1] += (RHS[1:-1,1:-1]+dt*S)


#Plot every 100 time steps

plotlabel = "t = %1.2f" %(n * dt)
plt.pcolor(xx,yy,T, shading='flat')
plt.title(plotlabel)
plt.axis('image')
plt.show()
```

Conclusion:

- **L’application du principe de Seebeck est très répandue:**