# TP : Modélisation physique: reprise de CCP 2015

Vous trouverez en fin de document une annexe sur `numpy` et `matplotlib`.

## 1 L'équation à résoudre

**Équation de la chaleur.** On souhaite résoudre de manière numérique l'équation aux dérivées partielles dans un mur d'épaisseur $e$ paramétré par l'abscisse $x \in [0, e]$ :

$$\alpha \frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2}$$

avec $\alpha = \frac{\rho C_p}{\lambda}$ et les conditions aux limites :

$$\begin{cases} T(0,t) &= T_{\text{int}} & \text{pour tout } t > 0 \\ T(e,t) &= T_{\text{ext2}} & \text{pour tout } t > 0 \\ T(x,0) &= ax+b & \text{pour tout } x \in [0,e] \end{cases}$$

**Température avant la simulation.** La température a chu brusquement à l'instrant $t = 0$, on suppose que le profil de température pour $x < e$ est $T(x, 0) = ax + b$ avec $a = \frac{T_{\text{ext1}} - T_{\text{int}}}{e}$ et $b = T_{\text{int}}$.

**Discrétisation.** L'idée de la résolution explicite est de discrétiser le temps et l'espace. On prend un pas de temps $\Delta x = \frac{e}{N+1}$ pour l'épaisser du mur, et on pose $x_i = i\Delta x$ pour $i \in [\![0, N+1]\!]$. On discrétise également le temps en intervalles de durée $\Delta t$, et on pose $t_k = k\Delta t$.

**Initialisation des constantes.** On fixe les constantes suivantes :

- $e = 0.40\,\text{m}$
- $\lambda = 1.65\,\text{W.m}^{-1}\text{K}^{-1}$
- $C_p = 1000\,\text{J.kg}^{-1}\text{K}^{-1}$
- $\rho = 2150\,\text{kg.m}^{-3}$
- $T_{\text{ext1}} = 10^\circ\,\text{C}$
- $T_{\text{ext2}} = -10^\circ\,\text{C}$
- $T_{\text{int}} = 20^\circ\,\text{C}$

**Question 1.** *Initialisation des constantes.* Commencez par ranger les variables dans des constantes aux noms appropriés. Calculer également les coefficients $a$, $b$ et $\alpha$.

**Question 2.** *Paramètres de la discrétisation et de la simulation.* On fixe $N = 60$ et $\Delta t = 25\,\text{s}$. Ranger ces deux quantités dans des constantes, de même que $\Delta x$ (à calculer), ainsi que $r = \frac{\Delta t}{\alpha \Delta x^2}$, et Itmax = 2000 (nombre d'itérations maximal dans une simulation).

**Question 3.** *Le vecteur $T_0$.* À l'aide de la fonction `np.linspace` par exemple, créez un tableau Numpy `T0` à $N$ éléments stockant les températures au temps $t = 0$, aux abscisses situées *à l'intérieur du mur* (donc pour $x_i$ avec $1 \leq i \leq N$). Attention à ne pas vous faire avoir par le décalage d'indice.

## 2 Résolution explicite

En utilisant des développements limités (essentiellement la méthode d'Euler, en fait), on avait vu que l'équation se ramenait, une fois discrétisée, à :

$$T_i^{k+1} = rT_{i-1}^k + (1-2r)T_i^k + rT_{i+1}^k \text{ pour } 1 \leq i \leq N$$

avec $r = \frac{\Delta t}{\alpha \Delta x^2}$, et où $T_i^k$ est une approximation de la température à l'abscisse $x_i$ et au temps $t_k$.

**Question 4.** Écrire une fonction `calcul_norme(V)` calculant la norme d'un tableau Numpy (à une dimension). On pourra utiliser `**0.5` pour la racine.

**Question 5.** Écrire une fonction `schema_explicite(T0,r,Itmax)` créant une matrice Numpy de taille $N \times \mathrm{Itmax}$ calculant successivement des approximations des $(T_i^k)$ pour $1 \leq i \leq N$ et $k < \mathrm{Itmax}$.

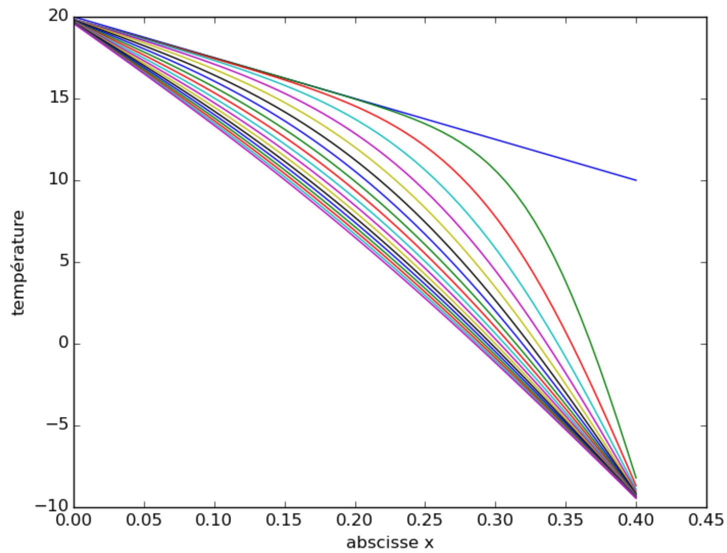
- on commencera par mettre $T_0$ comme première colonne de la matrice ;
- on remplira aussi la colonne d'indice 1 avec les $T_i^1$, pour $1 \leq i \leq N$. Attention, $T_1^1 = r(T_{\text{int}} + T_2^0) + (1-2r)T_1^0$ et $T_N^1 = r(T_{N-1}^0 + T_{\text{ext2}}) + (1-2r)T_N^0$. Prenez garde encore une fois au décalage entre l'indexation du sujet et les lignes de la matrice.
- on poursuivra la simulation tant que le nombre de colonnes remplies est inférieur à Itmax, et que la norme de la différence entre deux colonnes de $T$ (remplies) est supérieure à $10^{-2}$.
- Votre fonction renverra le nombre d'itérations effectuées et la matrice ainsi créée.

Le profil de la matrice à créer est le suivant :

$$\begin{pmatrix} T_1^0 & T_1^1 & \cdots & T_1^j & \cdots \\ T_2^0 & T_2^1 & \cdots & T_2^j & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ T_N^0 & T_N^1 & \cdots & T_N^j & \cdots \end{pmatrix}$$

**Question 6.** Réaliser la simulation explicite. Tracer le profil de température tous les 100 pas d'itération (rappel : faire plusieurs `plt.plot` rajoute des tracés dans une même fenêtre, qui sera affichée lorsque `plt.show()` est exécuté). Une boucle `for` est toute indiquée ici.

**Question 7.** Vérifier que la résolution se passe effectivement mal si $r > 1/2$, comme annoncé dans le sujet.




## 3 Résolution implicite

Poursuivre le sujet ! Je vous donne la fonction `calcTkp1`, elle est sur mon site pour la résolution implicite. Vérifiez d'abord que les questions de fin de sujet n'ont pas de secret pour vous (`input`...)

## 4 Rappels : Numpy et Matplotlib

On rappelle ici des fonctions utiles provenant des modules Numpy et Matplotlib, qu'on importera comme suit :

```
import numpy as np
import matplotlib.pyplot as plt
```

## 4.1 Numpy

Pour convertir une liste en tableau Numpy, on utilise simplement `np.array`. L'argument peut être une liste à une dimension (vecteur) ou 2 (matrice), ou plus...

```
>>> np.array([1,2]) #vecteur (ligne)
array([1, 2])
>>> np.array([[1,2],[3,4]]) #matrice
array([[1, 2],
       [3, 4]])
```

Pour accéder à un élément d'une telle matrice, on peut écrire au choix `M[i][j]` ou `M[i,j]`. Lignes et colonnes sont indexées à partir de zéro. On peut directement extraire les coefficients d'une ligne ou d'une colonne avec `M[i,:]` ou `M[:,j]`.

```
>>> M=np.array([[1,2],[3,4]])
>>> M[0,0]
1
>>> M[1][0]
3
>>> M[0,:]
array([1, 2])
>>> M[:,1]
array([2, 4])
```

`np.zeros` est utile pour créer des vecteurs/matrices remplis de zéros.

```
>>> np.zeros(4)
array([ 0.,  0.,  0.,  0.])
>>> np.zeros((3,2))
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])
```

`np.linspace` fournit facilement un tableau de flottants régulièrement espacés dans un intervalle : `np.linspace(a,b,N)` fournit $N$ points dont le premier est $a$, le dernier $b$, et ils sont espacés de $\frac{b-a}{N-1}$.

```
>>> np.linspace(0,5,8)
array([ 0. ,  0.71428571,  1.42857143,  2.14285714,  2.85714286, 3.57142857,  4.28571429,  5.])
```

## 4.2 Matplotlib

`plt.plot(X,Y)` permet de tracer une courbe reliant par lignes brisées les points $(x_i, y_i)$, avec `X` et `Y` deux tableaux Numpy (ou simplement deux listes) de même taille.

```
def f(x):
    return x*x+x-4

X=np.linspace(-5,5,1000)
Y=[f(x) for x in X]
plt.plot(X,Y)
plt.xlabel("x") #une chaîne de caractères pour mettre une étiquette sur l'axe des abscisses
plt.ylabel("y") #même chose sur les ordonnées
plt.show() #pour afficher le graphe.
```