Équation de la chaleur

RAPPORT DU PROJET

PRAP - Programmation avancée et projet

Claire TUALLE, Jaad BELHOUARI

ENSIIE - 2A

5 janvier 2024

Table des matières

1 Simulation théorique : équation de la chaleur

On cherche à modéliser l'évolution de la température dans un matériau à partir de l'équation de la chaleur :

$$\boxed{\frac{\partial u}{\partial t} = \frac{\lambda}{\rho c}\Delta u + \frac{F}{\rho c}}$$

Avec λ, ρ, c des coefficients physiques propres à chaque matériau représentant respectivement : la conductivité thermique, la masse volumique et la chaleur massique ; u la température et F le terme de création de chaleur.

Il s'agira ensuite de comparer cette évolution pour 4 matériaux différents, le cuivre, le fer, le verre, et le polystyrène dans un problème à une dimension puis à deux dimensions.

1.1 Premier cas : la barre

On considère d'abord une barre de taille L, selon l'axe Ox, nous allons observer la propagation de la chaleur au travers de celle-ci, jusqu'au temps t_{max}.

On peut donc réécrire l'équation de la chaleur en une seule dimension spatiale de la manière suivante :

$$\forall (t,x) \in [0, t_{max}] \times [0, L], \quad \frac{\partial u}{\partial t}(t,x) = \frac{\lambda}{\rho c}\frac{\partial^2 u}{\partial x^2}(t,x) + \frac{F(x)}{\rho c} \quad (*)$$

1.1.1 Discrétisation de l'équation

1.1.1.1 Notations

On posera pour la suite, avec $N = 1001$ le nombre de points (temporels et spatiaux) :

$$\forall (i,j) \in [\![0, N]\!]^2, t_i = \frac{i \times t_{max}}{N} \text{ et } x_j = \frac{j \times L}{N}$$

Pour faciliter les notations, on écrira :

$$\forall (i,j) \in [\![0, N]\!]^2, u(t_i, x_j) = u_{i,j}$$

1.1.1.2 Conditions

Ainsi, la condition initiale donne :

$$\boxed{\forall j \in [\![0, N]\!], u_{0,j} = u_0}$$

On a aussi la condition de Dirichlet :

$$\boxed{\forall i \in [\![0, N]\!], u_{i,N} = u_0}$$

De plus, la condition de Neumann est :

$$\forall t \in [0, t_{max}], \frac{\partial u}{\partial x}(t, 0) = 0$$

ce qui revient à dire :

$$\boxed{\forall i \in [\![0, N]\!], u_{i,0} = u_{i,1}}$$

Les trois équations ainsi obtenues vont nous servir dans la suite pour nos différentes approximations, afin de connaître la valeur de $u_{i,j}$ pour tout $(i, j) \in [\![0, N]\!]^2$ à l'aide de notre équation de la chaleur.

1.1.1.3 Approximation

L'approximation (à droite) de la dérivée temporelle de la température donne :

$$\begin{aligned}\frac{\partial u(t_i, x_j)}{\partial t_i} &= \frac{u(t_{i+1}, x_j) - u(t_i, x_j)}{t_{i+1} - t_i} \\ &= \frac{u_{i+1,j} - u_{i,j}}{\Delta t}, \text{ avec } \Delta t = \frac{t_{max}}{N}\end{aligned}$$

De même, (à gauche), on a :

$$\frac{\partial u(t_i, x_j)}{\partial t_i} = \frac{u_{i-1,j} - u_{i,j}}{\Delta t}$$

En soustrayant les développements précédents, ce qui revient à faire la moyenne des deux différences finies antérieure et postérieure à u(t,x), on obtient :

$$\frac{\partial u(t_i, x_j)}{\partial t_i} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta t}$$

qui est une approximation de $\frac{\partial u(t_i, x_j)}{\partial t_i}$ du 2^e ordre en Δt.

L'approximation de la dérivée seconde de la position donne :

$$\begin{aligned}\frac{\partial^2 u(t_i, x_j)}{\partial x_j^2} &= \frac{u(t_i, x_{j+1}) - 2u(t_i, x_j) + u(t_i, x_{j-1})}{(x_{i+1} - x_i)^2} \\ &= \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta x^2}, \text{ avec } \Delta x = \frac{L}{N}\end{aligned}$$

On réinjecte les expressions trouvées dans l'équation $(*)$, on trouve :

$$\frac{u_{i,j} - u_{i-1,j}}{\Delta t} = \frac{\lambda}{\rho c}\left(\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta x^2}\right) + \frac{F(x_j)}{\rho c}$$

En posant

$$\beta = \frac{\Delta t \lambda}{\rho c \Delta x^2}$$

On a l'expression suivante :

$$\implies u_{i,j} - u_{i-1,j} = \beta \times (u_{i,j+1} - 2u_{i,j} + u_{i,j-1}) + \frac{\beta \Delta x^2}{\lambda} \times F(x_j)$$

$$\implies \boxed{u_{i-1,j} = (1+2\beta)u_{i,j} - \beta(u_{i,j+1} + u_{i,j-1}) - \frac{\beta \Delta x^2 F(x_j)}{\lambda}}$$

1.1.2 Passage à la forme matricielle : méthode des différences finies implicites

Soit $N = 1001$, on passe ensuite à la forme matricielle, ce qui donne :

$$\forall i \in [\![0, N]\!], BU_{i+1} + C = U_i$$

D'où

$$\boxed{BU_{i+1} = U_i - C}$$

avec $U_n, U_{i+1}, C \in \mathcal{M}_{N,1}(\mathbb{R})$ tels que :

$$U_{i+1} = \begin{pmatrix} u_{i+1,1} \\ u_{i+1,2} \\ \vdots \\ \vdots \\ u_{i+1,N-1} \end{pmatrix}, U_i = \begin{pmatrix} u_{i,1} \\ u_{i,2} \\ \vdots \\ \vdots \\ u_{i,N-1} + \beta u_0 \end{pmatrix} \text{ et } C = \begin{pmatrix} \frac{\beta \Delta x^2 F(x_1)}{\lambda} \\ \frac{\beta \Delta x^2 F(x_2)}{\lambda} \\ \vdots \\ \vdots \\ \frac{\beta \Delta x^2 F(x_N - 1)}{\lambda} \end{pmatrix}$$

Ces vecteurs sont pris entre 1 et $N-1$ et non entre 0 et N de façon à mettre à part les conditions aux bords. Dans l'expression du vector U_i, un terme βu_0 à été ajouté. En effet si on considère l'équation :

$$u_{i-1,N-1} = (1+2\beta)u_{i,N-1} - \beta(u_{i,N-2} + u_{i,N}) - \frac{\beta \Delta x^2 F(x_j)}{\lambda}$$

on a $u_{i,N} = u_0$, d'où un terme $-\beta u_0$, ou de manière équivalente un terme βu_0 qui peut être ajouté à $u_{i-1,N-1}$.

Pour ce qui concerne la matrice $B \in \mathcal{M}_{N,N}(\mathbb{R})$, on a :

$$B = \begin{pmatrix} 1+\beta & -\beta & 0 & \cdots & \cdots & \cdots & 0 \\ -\beta & 1+2\beta & -\beta & 0 & \cdots & \cdots & 0 \\ 0 & -\beta & 1+2\beta & -\beta & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & -\beta \\ 0 & \cdots & \cdots & \cdots & 0 & -\beta & 1+2\beta \end{pmatrix}$$

Dans l'expression de la matrice B, le tout premier terme est $1+\beta$ et non $1+2\beta$ afin de prendre en compte la condition de Neuman, avec un raisonnement similaire au précédent :

$$u_{i-1,1} = (1+2\beta)u_{i,1} - \beta(u_{i,0} + u_{i,2}) - \frac{\beta \Delta x^2 F(x_j)}{\lambda}$$

avec cette fois-ci $u_{i,0} = u_{i,1}$.

1.1.3 Résolution

On reconnaît une équation matricielle de type $Ax = b$, avec A une matrice tridiagonale de taille nxn et x et b des vecteurs colonnes de taille n.

On applique donc l'algorithme de Thomas pour trouver les valeurs de x (dans notre cas U_{j+1}). Il faudra ainsi itérer cet algorithme N fois pour obtenir tous les vecteurs U_j décrivant la température à l'instant $t = j \times \Delta t$.

Soit $j \in [\![1, N]\!]$, on suppose que l'on connaît U_j et que l'on cherche U_{j+1}.

Algorithme de Thomas :

1.1.3.1 Initialisation

On utilisera pour cela deux vecteurs colonnes intermédiaires α et γ pour faciliter les calculs. On pose d'abord :

$$\alpha_1 = 1 + 2\beta$$

et

$$\gamma_1 = \frac{u_{1,j} - C_1}{\alpha_1}$$

1.1.3.2 Itération i

Et on a : $\forall i \in [\![2, N]\!]$,

$$\alpha_i = 1 + 2\beta - \frac{\beta^2}{\alpha_{i-1}}$$

et

$$\gamma_i = \frac{(u_{i,j} - C_i) - \beta\gamma_{i-1}}{\alpha_i}$$

À partir de ces coefficients, on trouve donc :

$$u_{i,j+1} = \gamma_i - \frac{-\beta \times u_{i+1,j+1}}{\alpha_i}$$

Sachant que $u_{N,j+1} = \gamma_N$

1.1.3.3 Terminaison

On peut donc grâce à l'algorithme de Thomas remplir le vecteur colonne U_{j+1}.

Il faut appliquer cet algorithme N fois, pour obtenir tous les vecteurs $(U_1, ..., U_N)$ contenant la température à l'instant $t_j = j \times \Delta t$ pour tout x_i.

Après chaque calcul de U_j, le vecteur est stocké en tant que colonne dans une matrice U qui contiendra donc sur les lignes l'information spatiale et sur les colonnes l'information temporelle.

1.1.4 Autres méthodes

D'autres méthodes étaient possibles pour obtenir la matrice U :

1.1.4.1 Méthode explicite La méthode explicite consiste simplement à résoudre :

$$u_{i,j+1} = u_{i,j} + \frac{\Delta t}{\rho c}\left(\lambda\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}\right) + F(i\Delta x)\right)$$

Il était ainsi possible de stocker les résultats dans une matrice que l'on complète à partir des conditions initiales données.

Cette méthode est moins coûteuse que la méthode implicite mais est plus instable.
Cette méthode a tout de même été implémentée, vous pourrez la retrouver commentée dans le fichier `edo.cpp`

1.1.4.2 Résolution par le pivot de Gauss À partir de l'équation matricielle ci-contre :

$$BU_{j+1} = U_j - C$$

Il était également possible de voir le problème de cette façon suivante :

$$U_{j+1} = B^{-1}U_j - B^{-1}C$$

Or cela revient à inverser la matrice B de taille $n \times n$. Étant donné le coût de telles opérations, nous avons décidé de choisir l'algorithme de Thomas propre aux matrices

tridiagonales, dans une démarche d'optimisation de complexité temporelle.

Cette méthode a tout de même été implémentée, vous pourrez la retrouver commentée dans le fichier `edo.cpp`

1.2 Deuxième cas : la plaque

On considère désormais une plaque de dimension $L \times L$, selon les axes Ox et Oy, pendant un temps t_{max}.

On peut donc réécrire l'équation de la chaleur en deux dimensions :

$$\forall (x,y,t) \in [0,L]^2 \times [0,t_{max}], \quad \frac{\partial u}{\partial t}(x,y,t) = \frac{\lambda}{\rho c}\left(\frac{\partial^2 u}{\partial x^2}(x,y,t) + \frac{\partial^2 u}{\partial y^2}(x,y,t)\right) + \frac{F(x,y} {\rho c} \quad (*)$$

1.2.1 Discrétisation de l'équation

1.2.1.1 Notations On posera pour la suite, avec $N = 1001$ le nmobre de points :

$$\forall (i,j,k) \in [\![0,N]\!]^3, x_i = \frac{i \times L}{N} \text{et } y_j = \frac{j \times L}{N}, t_k = \frac{k \times t_{max}}{N},$$

Pour faciliter les notations, on écrira :

$$\forall (i,j,k), \in [\![0,N]\!]^3, u(x_i,y_j,t_k) = u^n_{i,j}$$

1.2.1.2 Conditions Ainsi, la condition <u>initiale</u> donne :

$$\boxed{\forall (i,j) \in [\![0,N]\!]^2, u^0_{i,j} = u_0}$$

On a aussi les conditions de <u>Dirichlet</u> :

$$\boxed{\forall (j,k) \in [\![0,N]\!]^2, u^k_{N,j} = u_0}$$

$$\boxed{\forall (i,k) \in [\![0,N]\!]^2, u^k_{i,N} = u_0}$$

De plus, les conditions de <u>Neumann</u> sont (le raisonnement est le même que pour la barre) :

$$\boxed{\forall (j,k) \in [\![0,N]\!]^2, u^k_{0,j} = u^k_{1,j}}$$

$$\boxed{\forall (i,k) \in [\![0,N]\!]^2, u^k_{i,0} = u^k_{i,1}}$$

1.2.1.3 Approximation

L'approximation de la dérivée temporelle de la température est toujours :

$$\begin{aligned}\frac{\partial u(x_i, y_j, t_k)}{\partial t_k} &= \frac{u(x_i, , y_j, t_{k+1}) - u(x_i, y_j, t_k)}{t_{k+1} - t_k} \\ &= \frac{u_{i,j}^{k+1} - u_{i,j}^k}{\Delta t}, \text{ avec } \Delta t = \frac{t_{max}}{N}\end{aligned}$$

Les approximations des dérivées secondes de la position donnent :

$$\begin{aligned}\frac{\partial^2 u(x_i, y_j, t_k)}{\partial x_i^2} &= \frac{u(x_{i+1}, y_j, t_k) - 2u(x_i, y_j, t_k) + u(x_{i-1}, y_j, t_k)}{(x_{i+1} - x_i)^2} \\ &= \frac{u_{i+1,j}^k - 2u_{i,j}^k + u_{i-1,j}^k}{\Delta x^2}, \text{ avec } \Delta x = \frac{L}{N}\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 u(x_i, y_j, t_k)}{\partial y_j^2} &= \frac{u(x_i, y_{j+1}, t_k) - 2u(x_i, y_j, t_k) + u(x_i, y_{j-1}, t_k)}{(y_{j+1} - y_j)^2} \\ &= \frac{u_{i,j+1}^k - 2u_{i,j}^k + u_{i,j-1}^k}{\Delta y^2}, \text{ avec } \Delta y = \frac{L}{N}\end{aligned}$$

On réinjecte les expressions trouvées dans l'équation $(*)$, on trouve :

$$\frac{u_{i,j}^{k+1} - u_{i,j}^k}{\Delta t} = \frac{\lambda}{\rho c}\left(\frac{u_{i+1,j}^k - 2u_{i,j}^k + u_{i-1,j}^k}{\Delta x^2} + \frac{u_{i,j+1}^k - 2u_{i,j}^k + u_{i,j-1}^k}{\Delta y^2}\right) + \frac{F(x_i, y_j)}{\rho c}$$

On a l'expression suivante :

$$u_{i,j}^{k+1} = u_{i,j}^k(1 - 4\mu) + \mu(u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k) + \frac{F(x_i, y_i)}{\rho c}$$

Avec

$$\mu = \frac{\lambda \Delta t}{\rho c \Delta x^2} = \frac{\lambda \Delta t}{\rho c \Delta y^2}$$

car $\Delta x = \Delta y$

Cette expression permet un calcul direct de l'évolution de u^k que nous avons implémentée. Il se trouve cependant que la méthode explicite ne converge pas. Nous n'avons cependant pas réussi à implémenter la méthode implicite dans le cas de la plaque.

1.2.2 Passage à la forme matricielle : méthode des différences finies implicite

De même que pour la barre nous stockerons nos valeurs dans des matrices, cependant il est beaucoup plus lourd ici d'exprimer $u_{i,j}^k$ en fonction des autres termes : cela donne une matrice très grosse à inverser, qui nécessiterait à notre avis l'usage de méthode de matrices creuses.

L'usage de l'équation implicite revient juste à changer le signe de μ et de F.

Soit $N = 1001$, on pourrait réécrire l'équation trouvée en équation matricielle comme ci-dessous :

$$U_k = GU_{k+1} + H$$

Avec : $\forall k \in [\![0, N]\!], U_k, H \in \mathcal{M}_{N^2,1}(\mathbb{R})$,

$$U_k = \begin{pmatrix} u_{1,1}^k \\ u_{1,2}^k \\ \vdots \\ u_{1,N}^k \\ u_{2,1}^k \\ u_{2,2}^k \\ \vdots \\ \vdots \\ u_{N-1,N}^k \\ u_{N,1}^k \\ \vdots \\ u_{N,N}^k \end{pmatrix}, H = \begin{pmatrix} \frac{F(x_1,y_1)}{\rho c} \\ \frac{F(x_1,y_2)}{\rho c} \\ \vdots \\ \frac{F(x_1,y_N)}{\rho c} \\ \frac{F(x_2,y_1)}{\rho c} \\ \frac{F(x_1,y_2)}{\rho c} \\ \vdots \\ \vdots \\ \frac{F(x_{N-1},y_N)}{\rho c} \\ \frac{F(x_N,y_1)}{\rho c} \\ \vdots \\ \frac{F(x_N,y_N)}{\rho c} \end{pmatrix}$$

Et $G \in \mathcal{M}_{N^2,N^2}(\mathbb{R})$ que l'on construit par bloc de cette manière :

$$G = \begin{pmatrix} T & D & \mathcal{O}_N & \cdots & \cdots & \cdots & \mathcal{O}_N \\ D & T & D & \mathcal{O}_N & \cdots & \cdots & 0 \\ \mathcal{O}_N & D & T & D & \mathcal{O}_N & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \mathcal{O}_N \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & D \\ \mathcal{O}_N & \cdots & \cdots & \cdots & \mathcal{O}_N & D & T \end{pmatrix}$$

Avec $T \in \mathcal{M}_{N,N}(\mathbb{R})$ la matrice tridiagonale telle que :

$$T = \begin{pmatrix} 1-4\mu & \mu & 0 & \cdots & \cdots & \cdots & 0 \\ \mu & 1-4\mu & \mu & 0 & \cdots & \cdots & 0 \\ 0 & \mu & 1-4\mu & \mu & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \mu \\ 0 & \cdots & \cdots & \cdots & 0 & \mu & 1-4\mu \end{pmatrix}$$

$D \in \mathcal{M}_{N,N}(\mathbb{R})$ la matrice diagonale telle que :

$$D = \begin{pmatrix} \mu & 0 & \cdots & \cdots & 0 \\ 0 & \mu & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & \mu \end{pmatrix}$$

Et $\mathcal{O}_N$ la matrice nulle de taille $N \times N$:

$$\mathcal{O}_N = \begin{pmatrix} 0 & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 0 \end{pmatrix}$$

1.2.3 Résolution

Pour résoudre ce schéma implicite, il faudrait inverser la matrice G (résolution du système d'équations. Or cette matrice, de taille $10^6 \times 10^6$ a 10^12 éléments et ne peut être implémentée : il faudrait la définir en tant que matrice creuse, et écrire une procédure de résolution par pivot de Gauss avec les matrices creuses. Nous n'avons pas réussi à faire cela dans le temps imparti.

2 Implémentation du programme

Nous avons décidé de créer trois classes pour ce programme, les voici détaillées dans le diagramme ci-dessous :

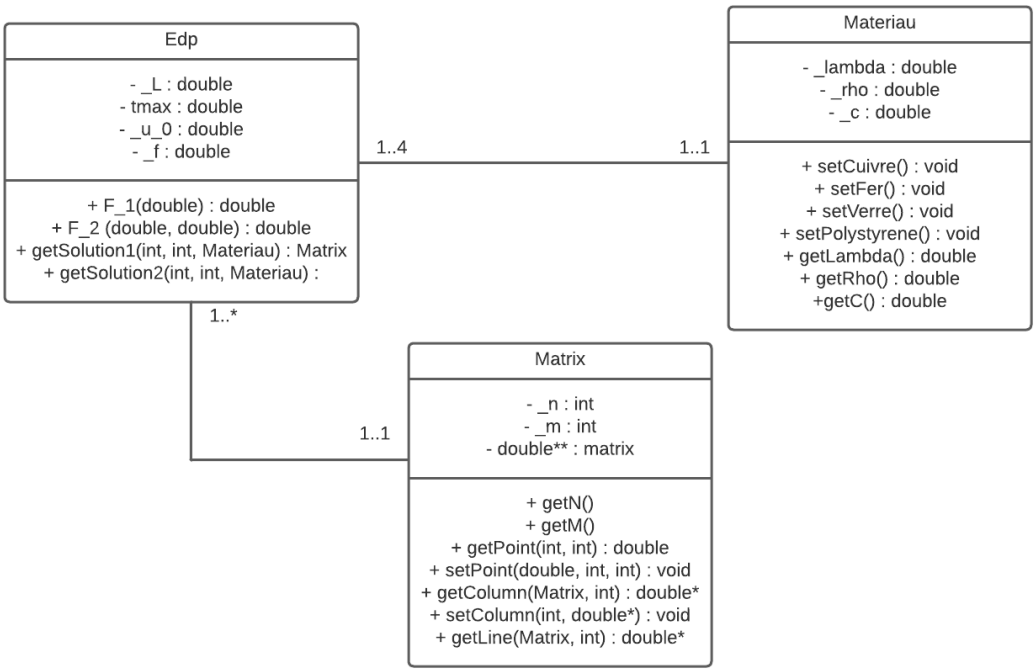

Figure 1 – Diagramme UML présentant les classes définies avec leurs attributs et méthodes

— La classe `Edp` permet de calculer les solutions des équations aux dérivées partielles à partir des informations sur l'objet ; elle contient aussi le calcul de la fonction de création de chaleur F ;

— La classe `Materiau` contient les informations sur les 4 matériaux donnés ;

— La classe `Matrix` définit la matrice et permet de les manipuler.

Finalement, la fonction `getSolution_2` fonctionne pour les deux matérériaux les moins conducteurs, à savoir le polystyrène et le cuivre. Quant aux deux autres matériaux, elle diverge. Nous avons affiché les résultats plus bas.

Alors que les résultats de `getSolution_1` sont corrects : on remarque bien les baisses et hausses de température attendues conforme aux sources de températures et à leurs emplacement.

3 Animation à l'aide de la bibliothèque SDL

3.1 Installation de la bibliothèque SDL sur Linux

Pour installer la bibliothèque SDL sur notre système Linux, nous avons suivi les étapes suivantes :

— **Utilisation de gestionnaires de paquets :** Nous avons utilisé le gestionnaire de paquets de notre distribution, `apt` sur Ubuntu, pour simplifier l'installation. Les commandes exécutées dans le terminal étaient :

```
sudo apt update
sudo apt install libsdl2-dev
```

La première commande actualise la liste des paquets disponibles et la seconde installe le paquet `libsdl2-dev`, contenant les fichiers de développement de SDL.

— **Vérification de l'installation :** Après l'installation, nous avons écrit un programme de test en C++ utilisant SDL. Nous avons compilé ce programme avec la commande :

```
g++ -o testSDL testSDL.cpp -lSDL2
```

La compilation réussie et l'exécution sans erreur du programme de test confirment l'installation correcte de la bibliothèque SDL.

3.2 Le fichier graphique.cpp

Nous avons implémenté dans un fichier `graphique.cpp` trois procédures :

— La procédure `sdlLine` qui permet de tracer une ligne.
— La procédure `test` qui montre une animation de l'évolution de la température sur la barre.
— La procédure `test_2D` qui affiche l'évolution de la température sur la plaque en fausses couleurs.

3.3 1ère application : Les résultats obtenus

Voici les résultats qu'on obtient :

3.4 Résultats pour la deuxième application

Pour la deuxième application, l'animation pour la plaque de polystyrène et de verre a bien fonctionnée, on obtient les résultats suivants :

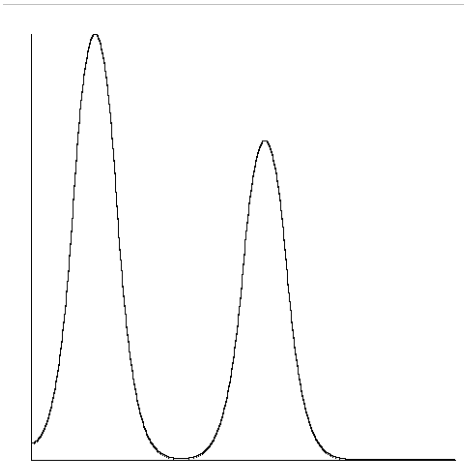

Figure 2 – Température dans la barre de cuivre à t=16 sec

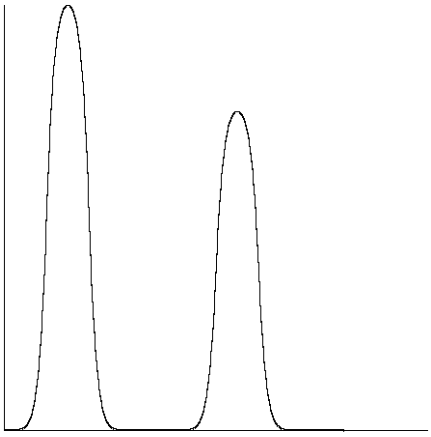

Figure 3 – Température dans la barre de fer à t=16 sec

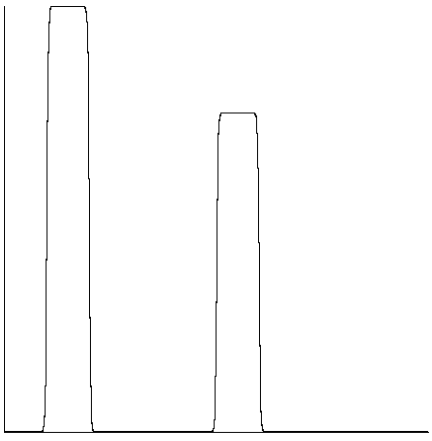

Figure 4 – Température dans la barre de verre à t=16 sec

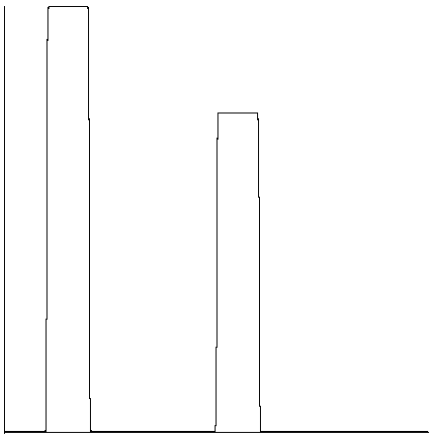

Figure 5 – Température dans la barre de polystyrène à t=16 sec

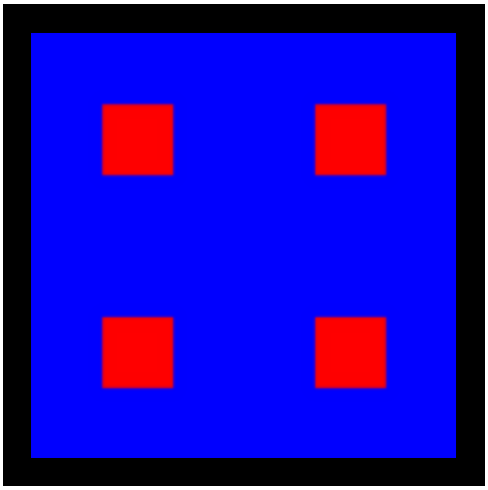

Figure 6 – Température dans la plaque de verre à t=16 sec

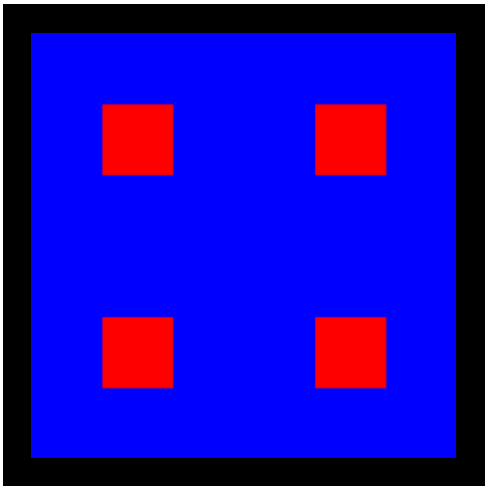

Figure 7 – Température dans la plaque de polystyrène à t=16 sec

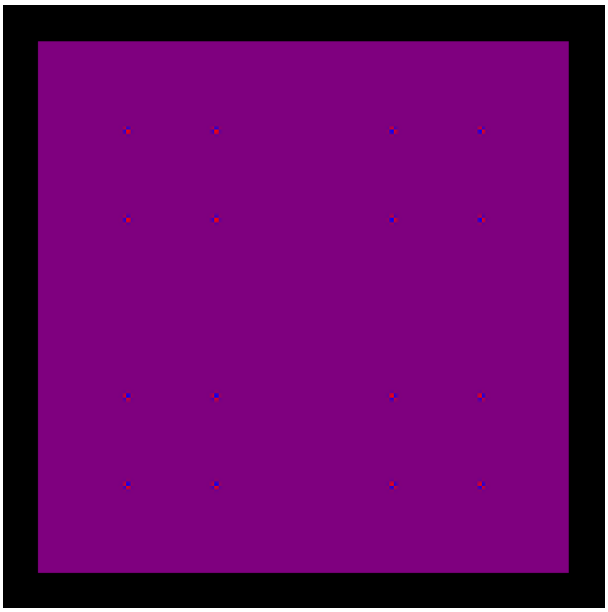

Figure 8 – Température dans la plaque de cuivre à t=16 sec

Pour la plaque de cuivre et de fer, nous remarquons que l'animation diverge, nous ne voyons que les sommets des zones chauffantes, on a pour les deux cette même image :

Nous nous apercevons que pour les métaux conducteurs de chaleur tels que le cuivre ou le fer la simulation explicite ne fonctionne pas.

4 Annexe

4.1 Problèmes rencontrés et solution choisie

Au départ, nous n'avons pas réussi à réaliser l'implémentation de l'animation de $u(t, .)$ à l'aide de la bibliothèque SDL.

C'est pourquoi, nous avons alors pris la décision de faire tout de même l'animation à l'aide d'un script python présent dans le fichier `animation.py`, afin de pouvoir visualiser et donner du sens à notre travail, et ne pas avoir uniquement des résultats sous forme de nombres stockés dans des tableaux.

Pour ce faire, nous avons modifié notre fichier `main.cpp` pour permettre la création et l'écriture d'un ficher `Barre_nom_du_matériau.csv` pour chacun des 4 matériaux. Nous avons ainsi obtenu 4 tableaux de 1001 lignes contenant sur chaque ligne une chaîne de caractère de 1000 nombres séparés par un espace. Chaque chaîne de caractère correspondant à l'évolution de la température d'un même point de la barre au cours du temps.

Le fichier `animation.py`, nous à alors permis de transformer nos fichiers .cvs en DataFrame utilisable pour réaliser nos animations à l'aide de la bibliotèque `matplotlib.pyplot` et `matplotlib.animation`. Ainsi, pour lancer une animation d'un matériau en particulier, il suffit de lire le fichier .csv contenant les donnés du matériau. Par exemple, pour le polystyrène :

```
df = pd.read_csv('Barre_polystyrene.csv')
```

Puis de compiler le fichier à l'aide de la commande : `'python animation.py'` dans un terminal.

4.2 Résultats

Dans le cas de la barre, pour nos 4 différents matériaux à $t = 14$ secondes, nous avons les résultats suivants, réprésentés dans les **figures 9 à 12** :

4.3 Conclusion

Ainsi, nos résultats correspondent plutôt bien à nos attentes, à savoir que le polystyrène et le verre sont les matériaux les plus isolants, alors que le cuivre et le fer permettent plus facilement à la chaleur de circuler à travers la barre, ce qui est cohérent au vu de leur conductivité thermique respectives.

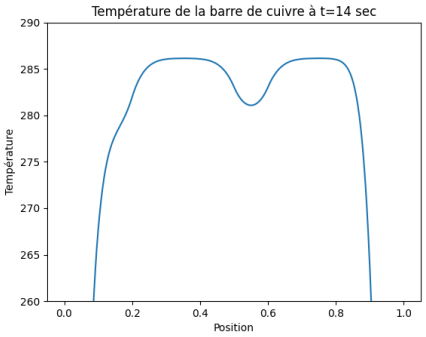

Figure 9 – Température dans la barre de cuivre à t=14 sec

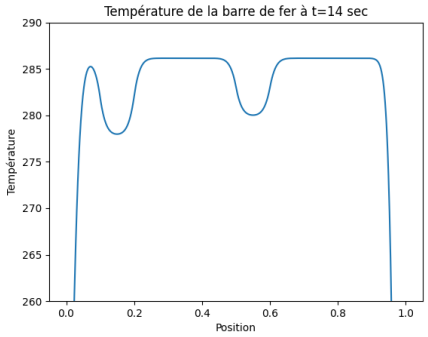

Figure 10 – Température dans la barre de fer à t=14 sec

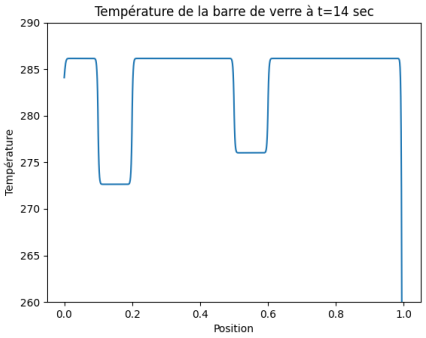

Figure 11 – Température dans la barre de verre à t=14 sec

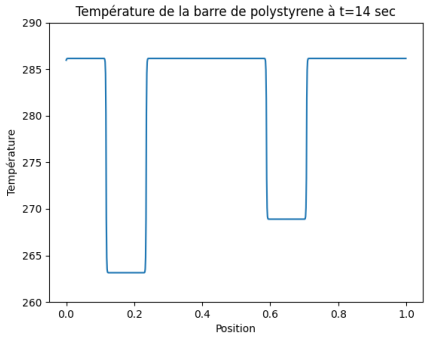

Figure 12 – Température dans la barre de polystyrène à t=14 sec

Cependant, bien que les graphiques et les animations semblent représenter de manière cohérente l'évolution de la température le long de la barre pour les points intérieurs (n'étant pas sur les bords), vis à vis des deux différentes sources de chaleur de l'énoncé et des propriétés inérhentes à chacun des matériaux ; il persirste encore 2 problèmes.

D'une part, les conditions aux bords ne semblent pas être totallement respectées. Et d'autre part, les deux sources de chaleurs viennent réduire les valeurs de la température là où elles se trouvent, alors qu'on s'attend à ce que ce soit l'inverse.

Ainsi, le fait d'avoir pu observer visuellement nos données sous forme de graphes nous a permis de déceler des petites érreurs dans notre code. Nous avons su ensuite régler ces erruers après avoir finalement réussi à faire fonctionner la bibliothèque SDL.