Rapport de projet : Simulation numérique d'un dispositif de refroidissement

Abdou WADE

15 décembre 2023

Université de Strasbourg
UFR de mathématique et d'informatique
Master 1 Calcul scientifique et mathématiques de l'information **(CSMI)**

Table des matières

1 Introduction

1.1 Présentation du sujet

objectif

L'objectif de ce projet est de réaliser un programme **C++** qui permet d'étudier le comportement thermique d'un dispositif de refroidissement d'un micro-processeur. Pour cela, on va utiliser un ventilateur pour réguler la temperature du micro-processeur.

model

Pour ce projet, on s'intresse uniquement à la simulation thermique d'une seule ailette du dissipateur. La geometrie de l'ailette est decrite par les longueurs L_x, L_y et L_z, le flux de chaleur Φ_p généré par le processeur et la temperature T_e de l'air ambiant. De plus on suppose que l'ailette est sufficement mince, du coup on considére le problème unidimensionnel. La temperature T en un point donné va dépendre uniquement de sa position x et du temps t.

En tenant compte des pertes latérales par convection de l'ailette, l'équation de la chaleur dans une ailette définie sur le domaine $x = [0, L_x]$ s'écrit sous la forme suivante :

$$\rho C_p \frac{\partial T}{\partial t} - \kappa \frac{\partial^2 T}{\partial x^2} + \frac{h_c p}{S}(T - T_e) = 0 \tag{1}$$

avec ρ la densité, C_p la chaleur spécifique à pression constante, κ la conductivité thermique, h_c le coefficient de transfert de chaleur surfacique, $S = L_y L_z$ l'aire d'une section tranversalle de l'ailette et $p = 2(L_y + L_z)$ le périmètre d'une section tranversalle de l'ailette.

Pour les conditions aux limites, nous supposons que le flux de chaleur Φ_p est connu à l'extrémité $x = 0$ de l'ailette et nous avons l'hypothèse que le transfert de chaleur à l'extrémité $x = L_x$ est négligeable. Donc on cherce la temperature T sur le domaine $x = [0, L_x]$ qui verifie les conditions suivantes :

$$\rho C_p \frac{\partial T}{\partial t} - \kappa \frac{\partial^2 T}{\partial x^2} + \frac{h_c p}{S}(T - T_e) = 0 \tag{2}$$

$$-\kappa \frac{\partial T}{\partial x}|_{x=0} = \Phi_p \tag{3}$$

$$-\kappa \frac{\partial T}{\partial x}|_{x=L_x} = 0 \tag{4}$$

1.2 Paramètres

Les paramètres du problème pour les simulations sont dans le tableau suivant :

Paramètre	Valeur	Unité
ρ	2700	kg/m^3
C_p	940	$J/(kg.K)$
κ	164	$W/(m.K)$
h_c	200	$W/(m^2.K)$
L_x	0.04	m
L_y	0.004	m
L_z	0.05	m
T_e	20	$°C$
Φ_p	$1.25.10^5$	W/m^2

TABLE 1 – Valeurs des paramètres géométriques et physiques

2 Compilation et exécution du programme

Pour compiler le programme, voici la commande à utiliser :
cd SRC pour se placer dans le dossier **SRC**
Et ensuite la commande suivante :
g++ -03 *.cpp -o main pour compiler touts les fichiers **.cpp**.
Et pour exécuter le programme, on utilise la commande suivante :
./main ../PARAMETRE/simu.cfg pour exécuter le programme.

3 Modéle stationnaire

3.1 Modéle stationnaire

Pour le modéle stationnaire, on calcule la distribution de la temperature T losrque le temps t tend vers l'infini. Pour cela, on enleve alors la derivé temporelle dans les équations 1 et 2.

Et pour la résolution numérique, on utilise la méthode des différences finies, on décompose le domaine $x = [0, L_x]$ en M intervalles de taille $h = L_x/M$. Ainsi on obtient un maillage de $M + 1$ points $x_i = ih$ avec $i = 0, 1, \ldots, M$. La solution numérique sera alors decrite par ces $M + 1$ points et on note T_i la valeur de la temperature au point x_i.

Pour la discrétisation de l'**EDP**, on utilise un developpement de Taylor, ce qui nous permet d'obtenir une approximation de la dérivée seconde de T en x_i :

$$\frac{\partial^2 T}{\partial x^2}(x_i) \approx \frac{T_{i+1} - 2T_i + T_{i-1}}{h^2}$$

En remplaçant cette approximation dans l'équation 1, on obtient :

$$-\kappa \frac{T_{i+1} - 2T_i + T_{i-1}}{h^2} + \frac{h_c p}{S}(T_i - T_e) = 0, \quad \forall i \in [1, M-1]$$

Et pour les conditions au limites on a :

$$-\kappa\frac{T_1 - T_0}{h}|_{x=0} = \Phi_p \tag{5}$$

$$-\kappa\frac{T_{M-1} - T_M}{h}|_{x=L_x} = 0 \tag{6}$$

Avec toutes ces équations, on obtient un système $AX = F$ avec A une matrice tridiagonale de taille $M \times M$ et F le vecteur second membre de (taille M) et X le vecteur solution représentant la distribution de la temperature T sur le domaine $x = [0, L_x]$.

on a le système linéaire suivant :

$$\underbrace{\begin{pmatrix} b_0 & c_0 & 0 & 0 & 0 & \dots & 0 \\ a_1 & b_1 & c_1 & 0 & 0 & \dots & 0 \\ 0 & a_2 & b_2 & c_2 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & a_{M-1} & b_{M-1} & c_{M-1} \\ 0 & \dots & \dots & 0 & 0 & a_M & b_M \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} T_0 \\ T_1 \\ T_2 \\ \vdots \\ \vdots \\ T_{M-1} \\ T_M \end{pmatrix}}_{X} = \underbrace{\begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ \vdots \\ \vdots \\ F_{M-1} \\ F_M \end{pmatrix}}_{F}$$

avec :

$$\begin{cases} \bullet a_i = -\frac{\kappa}{h^2} & \forall i \in [1, M-1] \\ \bullet a_M = -\frac{\kappa}{h} \end{cases} \quad \begin{cases} \bullet b_i = \frac{2\kappa}{h^2} + \frac{h_c p}{S} & \forall i \in [1, M-1] \\ \bullet b_0 = \frac{\kappa}{h} \\ \bullet b_M = \frac{\kappa}{h} \end{cases} \quad \begin{cases} \bullet c_i = -\frac{\kappa}{h^2} & \forall i \in [1, M-1] \\ \bullet c_0 = -\frac{\kappa}{h} \end{cases}$$

$$\begin{cases} \bullet F_i = \frac{h_c p}{S} T_e & \forall i \in [1, M-1] \\ \bullet F_0 = \Phi_p \\ \bullet F_M = 0 \end{cases}$$

Pour la résolution de ce système, on utilise la decomposition LU de la matrice A avec les matrices L et U de la forme suivante :

$$L = \begin{pmatrix} b_0^* & 0 & 0 & 0 & 0 & \dots & 0 \\ a_1 & b_1^* & 0 & 0 & 0 & \dots & 0 \\ 0 & a_2 & b_2^* & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & a_{M-1} & b_{M-1}^* & 0 \\ 0 & \dots & \dots & 0 & 0 & a_M & b_M^* \end{pmatrix}, U = \begin{pmatrix} 1 & c_0^* & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & c_1^* & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & c_2^* & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & 0 & 1 & c_{M-1}^* \\ 0 & \dots & \dots & 0 & 0 & 0 & 1 \end{pmatrix}$$

Les coefficients b_i^* et c_i^* sont calculés comme suit :

— 1. $b_0^* = b_0$ et $c_0^* = \frac{c_0}{b_0^*}$
— 2. Pour i allant de 1 à $M-1$:
  — $\bullet b_i^* = b_i - a_i c_{i-1}^*$

— • $c_i^* = \frac{c_i}{b_i^*}$
— 3. $b_M^* = b_M - a_M c_{M-1}^*$

Aprés avoir calculé les matrices L et U, on résoud le système $AX = F$ en deux étapes :

— On résoud le système $LY = F$.
— On résoud le système $UX = Y$.
— On obtient alors la solution X du système $AX = F$
— **Remarque :** Pour la résolution du système $LY = F$, on a :
$Y_0 = \frac{F_0}{b_0^*}$
et pour $i \in [1, M]$:

$$Y_i = \frac{F_i - a_i Y_{i-1}}{b_i^*}$$

— **Remarque :** Pour la résolution du système $UX = Y$, on a :
$X_M = Y_M$
et pour $i \in [M-1, 0]$:

$$X_i = Y_i - c_i^* X_{i+1}$$

De plus, le problème admet une solution analytique, notée T_{exacte}, qui est donnée par :

$$T_{exacte} = T_e + \frac{\Phi_p}{\kappa} \frac{\cosh(\sqrt{a} L_x)}{\sqrt{a}\sinh(\sqrt{a} L_x)} \frac{\cosh(\sqrt{a}(L_x - x))}{\cosh(\sqrt{a} L_x)} \quad avec \quad a = \frac{h_c p}{\kappa S}. \tag{7}$$

3.2 Implémentation du modéle stationnaire

3.2.1 Implémentation du modéle stationnaire

Pour ce projet, on a les dossiers suivants :
— **INCLUDE** : contient les fichiers d'en-tête(.hpp)
— **SRC** : contient les fichiers sources(.cpp)
— **PARAMETRE** : contient les paramètres du problème(.cfg)
— **CSV** : contient les fichiers csv(.csv)
— **VTK** : contient les fichiers vtk(.vtk)

On commence par écrire les parametres du projet nécéssaires pour la simulation dans le fichier **PARAMETRE/simu.cfg**.
Et pour lire ce fichier on a une classe **parametres.cpp** dont son constructeur prend en parametre le nom du fichier **.cfg** et qui permet de lire les parametres du fichier **.cfg** et de les stocker dans des variables.

On a une classe **Matrice** implémentatée dans le fichier **matrice.cpp** qui permet de créer une matrice tridiagonale dans le cas du modéle stationnaire.
Dans cette classe, on a un constructeur qui prend en parametre les vecteurs a, b et c.
Ce constructeur prend les vecteurs et les remplis avec les valeurs décrites dans le projet.
Aprés on remplis la diagonale, la diagonale superieur et la diagonale inferieur de la matrice avec les vecteurs b et c et a respectivement.

On a une méthode **matriceTridiagonale()** qui permet de remplir la diagonale, la diagonale superieur et la diagonale inferieur de la matrice matrice tridiagonale du système $AX = F$ dans le cas du modéle stationnaire et de retourner la matrice.

Dans cette classe on a aussi une méthode **secondMembre()** qui permet de calculer le vecteur second membre du système $AX = F$ dans le cas du modéle stationnaire.

Pour la résolution du système $AX = F$, on utilise la decomposition LU de la matrice A.

Pour cela, on a une classe **Resolution** implémentatée dans le fichier **resolution.cpp** avec un constructeur qui prend en parametres une matrice de type **Matrice** et un entier **taille**.
Ce constructeur permet de remplir b^* la diagonale de la matrice L et c^* la diagonale superieur de la matrice U avec les valeurs decrites en haut.

On a aussi une méthode **decompositionLU()** dans cette classe qui prend en parametre une matrice de type **Matrice** et un entier **taille** et qui permet de faire la decomposition LU de la matrice tridiagonale A du système.

Et pour la résolution du système $AX = F$, on procéde en deux étapes :
1. On résoud le système $LY = F$ avec la méthode **resoudreLYF()**
Dans cette méthode, on a un vecteur **Y** qui dont la première valeur est F_0/b_0^* et on a une boucle qui permet de calculer les valeurs de Y_i avec $i \in [1, M]$ et qui sont stockées dans le vecteur **Y**.
Et on retourne le vecteur **Y** qui est la solution du système $LY = F$.
c'est la méthode de montée.

2. On résoud ensuite le système $UX = Y$ avec la méthode **resoudreUXY()**
Dans cette méthode, on a un vecteur **X** qui dont la dernière valeur est Y_M et on a une boucle qui permet de calculer les valeurs de X_i avec $i \in [M-1, 0]$ et qui sont stockées dans le vecteur **X**.
Et on retourne le vecteur **X** qui est la solution du système $UX = Y$ mais aussi la solution du système $AX = F$.
c'est la méthode de descente.

Dans cette même clase on a aussi les méthodes qui permettent d'écrire les résultats dans un fichier csv et dans un fichier vtk.

Pour l'écriture des résultats dans un fichier csv, on a la méthode **ecrireFichier()** qui prend en parametre un **vecteur** et le nom du fichier **.csv** et qui permet d'écrire les résultats dans le fichier.

Pour l'écriture des résultats dans un fichier vtk, on a la méthode **writeVTKFile()** qui prend en parametre un **vecteur** et le nom du fichier **.vtk** et qui permet d'écrire les résultats dans le fichier.

Pour cette fonction, on a écrit l'entête du fichier vtk comme décrit dans le projet et aprés on a fait la localisation des points du maillage dans le fichier en utilisant l'Alogorithme 2 :

Algorithm 2 :

Pour i allant de 0 à Mx :
localiser x_{i00} dans le maillage $1d$ (i.e. trouver k tel que $x_{i00} \in [x_k, x_{k+1}]$)
calculer les coefficients de la droite $y = ax + b$ passant par les points (x_k, T_k et x_{k+1}, T_{k+1})
évaluer $T_i^{chap} = ax_i00 + b$
end for

On a aussi une méthode **solutionExacte()** qui permet de calculer la solution exacte du problème.

3.2.2 Visualisation des résultats avec Paraview

Les solutions stoctées dans le fichier csv (**stationnaire.csv**) pour le cas stationnaire sont utilisées pour la visualisation sur **Paraview**.

Celui-ci nous permet de visualiser les résultats de la simulation et nous permet d'avoir les figures suivantes :

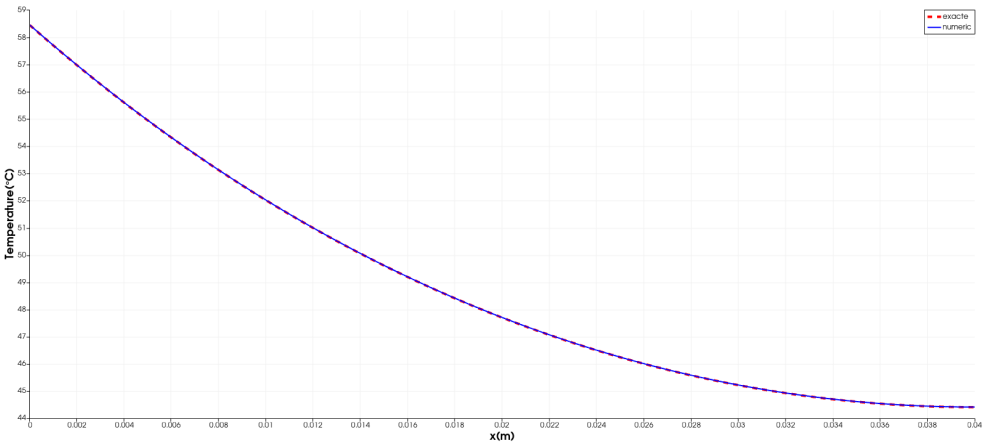


FIGURE 1 – Tracé pour $L_x = 40mm$

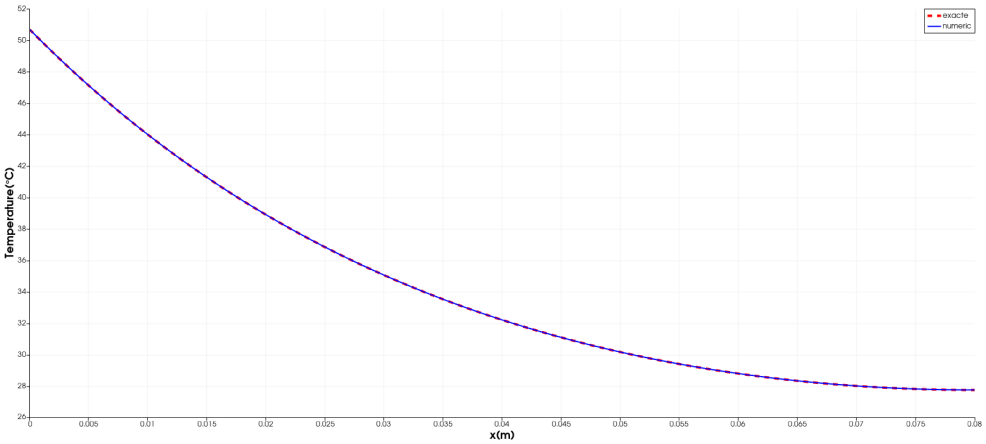


FIGURE 2 – Tracé pour $L_x = 80mm$

Pour le cas stationnaire, on a aussi un fichier vtk (**stationnaire.vtk**) qui permet de visualiser les résultats sur **Paraview**.

On a les figures suivantes :

Remarque : Pour le cas stationnaire, j'ai fiat la visualisation du fichier vtk mais j'ai pas eu la même figure que celle demandée dans le projet.

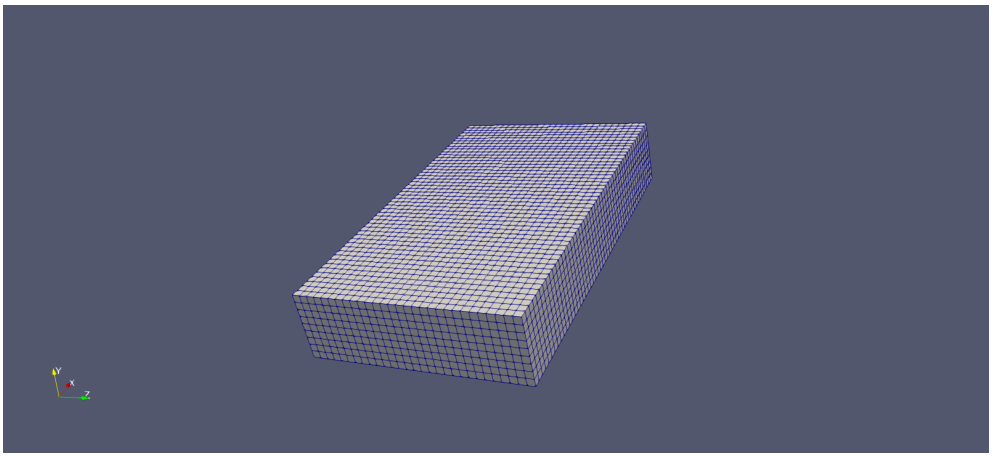
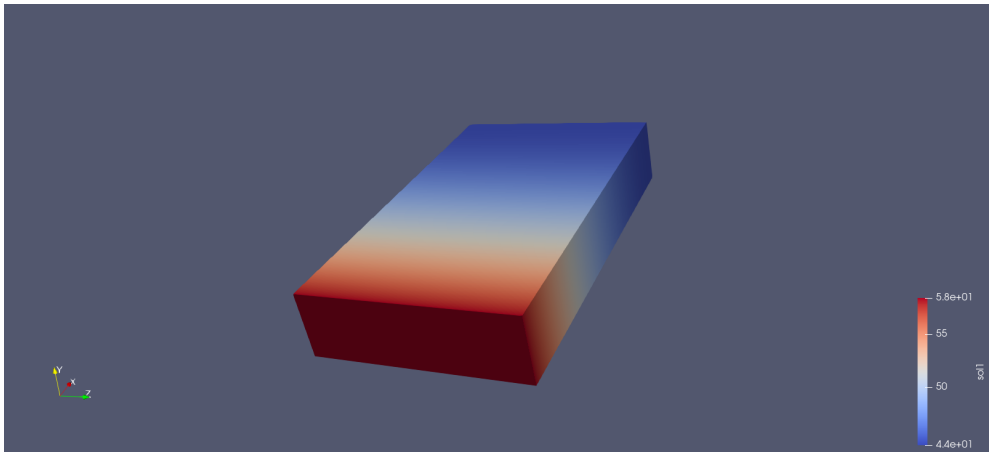


FIGURE 3 – Maillage 3d($M_x = 50$, $M_y = 10$, $M_z = 30$)



h !

FIGURE 4 – Température calculée sur le maillage 3d : cas stationnaire

4 Modéle instationnaire

4.1 Modéle instationnaire

Pour le modéle instationnaire, on commence par echantillonner l'intervalle $[0, t_{finale}]$ en $N+1$ temps discret. Ainsi le pas de temps est Δt. Et sur l'ensemble on a $t_n = n\Delta t$ avec $n = 0, 1, \ldots, N$. Pour cette partie, on utilise $t_{finale} = 300s$ et $N = 600$ ce qui nous donne $\Delta t = t_{finale}/N = 0.5s$. On note T_i^n la valeur de la temperature au point x_i au temps t_n.

La dérivée temporelle est discretisée par un schéma d'Euler :

$$\frac{\partial T}{\partial t}(x_i, t_n) \approx \frac{T_i^{n+1} - T_i^n}{\Delta t}$$

En remplaçant cette approximation dans l'équation 1, on obtient :

$$\rho C_p \frac{T_i^{n+1} - T_i^n}{\Delta t} - \kappa \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{h^2} + \frac{h_c p}{S}(T_i^{n+1} - T_e) = 0, \quad \forall i \in [1, M-1] \qquad (8)$$

Et pour les conditions au limites on a :

$$-\kappa\frac{\partial T}{\partial x}|_{x=0} \approx -\kappa\frac{T_1^{n+1}-T_0^{n+1}}{h} = \Phi_p \tag{9}$$

$$-\kappa\frac{\partial T}{\partial x}|_{x=L_x} \approx -\kappa\frac{T_{M-1}^{n+1}-T_M^{n+1}}{h} = 0 \tag{10}$$

De plus, on a la condition initiale suivante :

$$T_i^0 = T_e, \quad \forall i \in [0, M] \tag{11}$$

Avec les conditions aux limites, on a :

$\frac{\kappa}{h}T_0^{n+1} - \frac{\kappa}{h}T_1^{n+1} = \Phi_p$ et $\frac{\kappa}{h}T_{M-1}^{n+1} - \frac{\kappa}{h}T_M^{n+1} = 0$

Donc on obtient :
$b'_0 = \frac{\kappa}{h}$, $c'_0 = -\frac{\kappa}{h}$, $F'_0 = \Phi_p$, $b'_M = \frac{\kappa}{h}$, $a'_M = -\frac{\kappa}{h}$ et $F'_M = 0$

On a le système linéaire suivant :

$$\underbrace{\begin{pmatrix} b'_0 & c'_0 & 0 & 0 & 0 & \dots & 0 \\ a'_1 & b'_1 & c'_1 & 0 & 0 & \dots & 0 \\ 0 & a'_2 & b'_2 & c'_2 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & a'_{M-1} & b'_{M-1} & c'_{M-1} \\ 0 & \dots & \dots & 0 & 0 & a'_M & b'_M \end{pmatrix}}_{A'} \underbrace{\begin{pmatrix} T_0^{n+1} \\ T_1^{n+1} \\ T_2^{n+1} \\ \vdots \\ \vdots \\ T_{M-1}^{n+1} \\ T_M^{n+1} \end{pmatrix}}_{X^{n+1}} = \underbrace{\begin{pmatrix} F'_0 \\ F'_1 \\ F'_2 \\ \vdots \\ \vdots \\ F'_{M-1} \\ F'_M \end{pmatrix}}_{F'} + \underbrace{\begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ \vdots \\ \vdots \\ m_{M-1} \\ m_M \end{pmatrix}}_{m} \underbrace{\begin{pmatrix} T_0^n \\ T_1^n \\ T_2^n \\ \vdots \\ \vdots \\ T_{M-1}^n \\ T_M^n \end{pmatrix}}_{X^n}$$

avec :

$$\begin{cases} \bullet a'_i = -\frac{\kappa}{h^2} & \forall i \in [1, M-1] \\ \bullet a'_M = -\frac{\kappa}{h} \end{cases} \quad \begin{cases} \bullet b'_i = \frac{\rho C_p}{\Delta t} + \frac{2\kappa}{h^2} + \frac{h_c p}{S} & \forall i \in [1, M-1] \\ \bullet b'_0 = \frac{\kappa}{h} \\ \bullet b'_M = \frac{\kappa}{h} \end{cases} \quad \begin{cases} \bullet c'_i = -\frac{\kappa}{h^2} & \forall i \in [1, M-1] \\ \bullet c'_0 = -\frac{\kappa}{h} \end{cases}$$

$$\begin{cases} \bullet F'_i = \frac{h_c p}{S} T_e & \forall i \in [1, M-1] \\ \bullet F'_0 = \Phi_p \\ \bullet F'_M = 0 \end{cases} \quad \begin{cases} \bullet m_i = \frac{\rho C_p}{\Delta t} & \forall i \in [1, M-1] \\ \bullet m_0 = 0 \\ \bullet m_M = 0 \end{cases}$$

4.2 Implémentation du modéle instationnaire

4.2.1 Implémentation du modéle instationnaire

Pour la résolution du modéle instationnaire, on procéde ainsi :

Comme pour le modéle stationnaire, on a une classe **MatInst** implémentatée dans le fichier **matriceInst.cpp** qui permet de créer une matrice tridiagonale dans le cas du modéle instationnaire.
Dans cette classe, on a un constructeur qui prend en parametre les vecteurs a', b' et c'.
Ce constructeur remplis les vecteurs comme décrits ci dessus.

Aprés on remplis la diagonale, la diagonale superieur et la diagonale inferieur de la matrice avec les vecteurs b' et c' et a' respectivement.

Dans cette classe on a aussi deux méthodes :

La méthode **matriceTridiagInst()** qui construit la matrice tridiagonale du système dans le cas du modéle instationnaire et qui retourne la matrice.

La méthode **FInst() qui permet de calculer le vecteur F' du système mais pas tout le second membre du cas instationnaire.**
Et cette méthode retourne le vecteur **F'**.

Pour la résolution du système $AX^{n+1} = F' + mX^n$, on utilise la factorisation LU de la matrice A du système.

Ainsi, on a une classe **ResolutionInst** implémentatée dans le fichier **resolutionInst.cpp** avec un constructeur qui prend en parametres une matrice de type **MatInst** et un entier **taille**.

Ce constructeur permet de remplir la diagonale de la matrice L et la surdiagonale de la matrice U.

On construit les vecteurs **bInstaRes** et **cInstaRes** on les remplis de telles sortes que **bInstaRes** contient les valeurs de la diagonale de la matrice L et **cInstaRes** contient les valeurs de la surdiagonale de la matrice U.

On a aussi une méthode **FactoLU()** qui prend en parametre une matrice de type **MatInst** et qui retourne un vecteur de type **MatInst** qui contient la matrice **L** et la matrice **U**.

Et pour la résolution du système $AX^{n+1} = F' + mX^n$, on procéde comme suit :

On applique l'Algorithm donné dans le projet :

Algorithm 1 : Résolution du système $AX^{n+1} = F' + mX^n$.

Initialiser T^0
Pour $n = 0, 1, \ldots, N$ faire
calculer T^{n+1} en fonction de T^n
$T^n = T^{n+1}$
Fin Pour

Comme dans le cas stationnaire, on a les méthodes de montée et descente.

Mais pour notre méthode montée ici on a :

La méthode **LYFInst()** qui prend en parametre le vecteur **Y** qu'on veut calculer, une matrice de type **MatInst**, un entier **taille** et un vecteur **F**.
Mais dans notre cas ici, on prend notre vecteur F qui est égal à F' + mT0 avec T0 le vecteur qui contient les temperatures initiales et m = $\frac{\rho C_p}{\Delta t}$ = CFL dans l'Implémentation.
Et cette méthode permet de calculer le vecteur **Y** et de le retourner.

Et pour la méthode descente, on a :

La méthode **UXYInst()** qui prend en parametre le vecteur **Y** de la méthode montée, et qui nous permet de calculer le vecteur **X** et de le retourner.

Et pour la résolution du système $AX^{n+1} = F' + mX^n$, On applique l'Algorithm 1.

Pour cela on a une méthode **resoudreInst()** qui prend en parametre une matrice de type **MatInst**, un entier **taille**.

Cette méthode permet de calculer la solution du système $AX^{n+1} = F' + mX^n$ en appliquant l'Algorithm 1, et les méthodes montée et descente, et cette méthode retourne le vecteur X^{n+1} qui est la solution du système $AX^{n+1} = F' + mX^n$ qui contient les valeurs des températures.

Dans cette même classe on a aussi les méthodes qui permettent d'écrire les résultats dans un fichier csv et dans un fichier vtk.

Pour l'écriture des résultats dans un fichier csv, on a la méthode **ecritureCSVInst()** qui prend en parametre un vecteur **Solve** vecteur qui contient les valeurs des températures et le nom du fichier **.csv**.
Cette méthode permet d'écrire les résultats dans le fichier.

Pour l'écriture des résultats dans un fichier vtk, on a la méthode **VTKParPasDeTemps()** qui prend en parametre un vecteur **Solve** et le pas de temps **n**.

Cette méthode permet d'écrire les résultats dans le fichier.

Pour cette fonction, on a boucler pour t allant de 0 à t_{finale} pour pouvoir écrire les fichiers vtk pour chaque pas de temps dans le même dossier.

4.2.2 Visualisation des résultats avec Paraview

Le fichier csv (**instationnaire.csv**) contient les solutions pour le cas instationnaire et est utilisé pour la visualisation sur **Paraview**.

Celui-ci nous permet de visualiser les résultats de la simulation et nous permet d'avoir les figures suivantes :

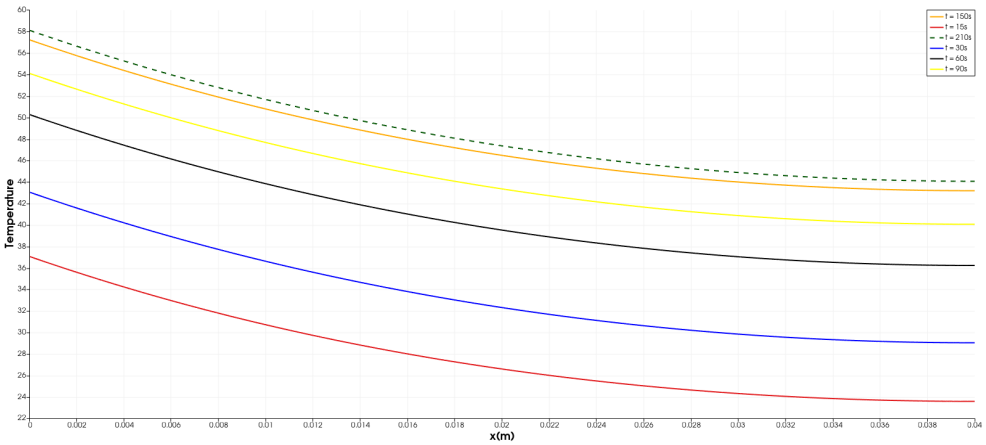


FIGURE 5 – Flux de chaleur constant

Pour le cas instationnaire, on a aussi des fichiers vtk qui sont dans un même dossier qu'on pu visualiser sur **Paraview**.

Mais pour la visualisation, J'arrive pas à avoir la même figure que celle demandée dans le projet.
Du coup, j'ai fait la visualisation des fichiers vtk mais pour la figure en bas pour t = 599s.

Qui est la figure suivante :

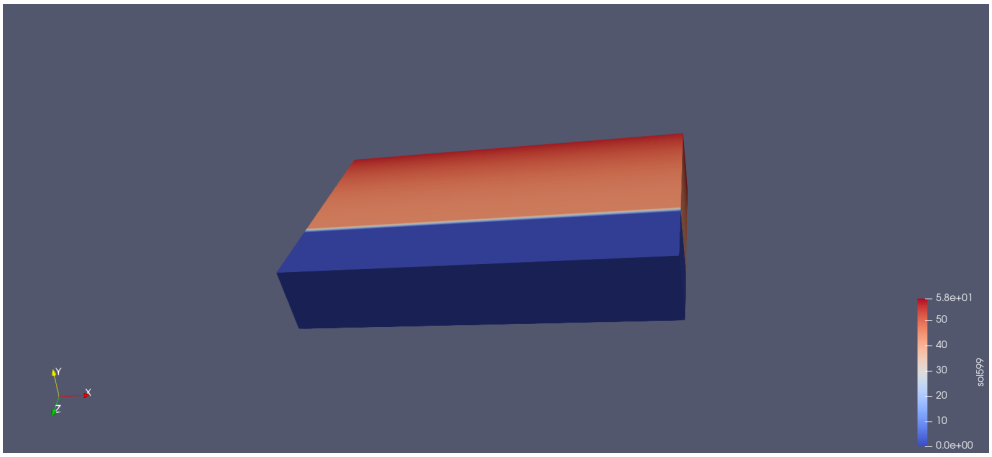


FIGURE 6 – Température calculée sur le maillage 3d : cas instationnaire

5 Conclusion

Pour conclure, ce projet nous a permis de mettre en pratique les connaissances acquises en cours et de les approfondir.
Il nous a permis de comprendre l'importance de la discrétisation et de la résolution des équations aux dérivées partielles.

Ce projet nous a aussi permis de comprendre l'importance de la programmation orientée objet et de la modularité.

Et pour finir, ce projet nous a permis d'approfondir nos connaissances sur le logiciel **Paraview** qui est un outil très puissant pour la visualisation des résultats de la simulation.