LECOURTIER Frédérique 15 décembre 2021

Rapport - Projet : Simulation numérique d'un dispositif de refroidissement

Table des matières

1 Présentation du projet

Le but de ce projet est d'écrire un programme en C++ permettant d'analyser le comportement thermique d'un dispositif de refroidissement d'un micro-processeur. Un des moyens utilisé pour réguler la température d'un processeur est l'utilisation d'un ventilateur. Pour améliorer ce processus, le processeur est généralement attaché à un dissipateur (composé de plusieurs ailettes) qui est un élément très conducteur de chaleur. Ce dispositif supplémentaire permet d'augmenter la surface d'échange avec le flux d'air et d'ainsi refroidir plus efficacement le composant électronique. Nous allons ici nous intéresser seulement à la simulation thermique d'une seule ailette du dissipateur. Nous supposerons que l'ailette est assez fine pour considérer le problème unidimensionnel, c'est-à-dire que la température T en un point donné dépendra uniquement de sa position selon x et de l'instant t.
Les paramètres à prendre en compte par la suite sont donnés dans le tableau suivant :

ρ	2700	$kg/(m^3)$	densité
C_p	940	$J/(kg.K)$	chaleur spécifique à la pression constante
κ	164	$W/(m.K)$	conductivité thermique
T_e	20	$°C$	température ambiante
Φ_p	$1,25.10^5$	$W/(m^2)$	flux de chaleur généré par le processeur
h_c	10 ou 200	$W/(m^2.K)$	coefficient de transfert de chaleur surfacique
L_x	$0,04$	m	dimensions de l'ailette
L_y	$0,004$	m	
L_z	$0,05$	m	

$S = L_y L_z$ l'aire d'une section transversale $\quad p = 2(L_y + L_z)$ son périmètre

Le paramètre h_c dépend de si le ventilateur est allumé ($h_c = 200$) ou non ($h_c = 10$).
Finalement, on cherchera la température T définie sur le domaine $x = [0, L_x]$ qui vérifie :

$$\begin{aligned} \rho C_p \tfrac{\partial T}{\partial t} - \kappa \tfrac{\partial^2 T}{\partial x^2} + \tfrac{h_c p}{S}(T - T_e) &= 0 \\ -\kappa \tfrac{\partial T}{\partial x}\big|_{x=0} &= \Phi_p \\ -\kappa \tfrac{\partial T}{\partial x}\big|_{x=L_x} &= 0 \end{aligned}$$

1.1 Modèle stationnaire

Nous commencerons par traiter le cas stationnaire, c'est-à-dire calculer la température lorsque le temps t tend vers l'infini. On enlèvera ainsi la dérivée en temps présente dans l'équation ci-dessus, qui se réécrit alors :

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

Par la méthode des différences finies, on peut résoudre cette EDP en discrétisant le segment $[0, L_x]$ en M intervalles de longueur $h = L_x/M$ (et donc en $M + 1$ points définis par $x_i = ih$ pour $i = 0, \ldots, M$). On notera alors T_i la température calculée au point x_i.
En utilisant, un développement de Taylor pour approcher la dérivée seconde, on a pour $i = 1, \ldots, M - 1$:

$$\begin{aligned} T(x_i + h) &= T(x_i) + hT'(x_i) + \frac{h^2}{2}T''(x_i) + \frac{h^3}{6}T'''(x_i) + O(h^4) \\ T(x_i - h) &= T(x_i) - hT'(x_i) + \frac{h^2}{2}T''(x_i) - \frac{h^3}{6}T'''(x_i) + O(h^4) \end{aligned}$$

En les additionnant, on obtient :

$$T(x_i + h) + T(x_i - h) = 2T(x_i) + h^2 T''(x_i) + O(h^4)$$

et ainsi :

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

De plus, pour les conditions au bord, un développement de Taylor à l'ordre 1 nous suffit pour discrétiser le problème :

$$\begin{aligned} T(x + h) &= T(x) + hT'(x) + O(h) \\ T(x - h) &= T(x) - hT'(x) + O(h) \end{aligned}$$

Ceci nous donne les conditions aux bords :

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

Le problème se réécrit alors :

$$-\kappa\frac{T_{i-1} - 2T_i + T_{i+1}}{h^2} + \frac{h_c p}{S}(T_i - T - e) = 0 \quad \forall i \in [1, M-1]$$
$$-\kappa\frac{T_1 - T_0}{h} = \Phi_p$$
$$-\kappa\frac{T_{M-1} - T_M}{h} = 0$$

ou encore

$$-\frac{\kappa}{h^2}T_{i-1} + \left(\frac{2\kappa}{h^2} + \frac{h_c p}{S}\right)T_i - \frac{\kappa}{h^2}T_{i+1} = \frac{h_c p}{S}T_e \quad \forall i \in [1, M-1]$$
$$-\frac{\kappa}{h}(T_1 - T_0) = \Phi_p$$
$$-\frac{\kappa}{h}(T_{M-1} - T_M) = 0$$

Ainsi, ce problème peut s'écrire sous forme matricielle $AX = F$ avec A une matrice tridiagonale (de taille $M+1$), F le vecteur second membre (de taille $M+1$) et X le vecteur solution (de taille $M+1$) qui représente la température en chaque point de discrétisation x_i pour $i = 0, \ldots, M$.
La sous-diagonale (a de taille M), la diagonale (b de taille $M+1$) et la sur-diagonale (c de taille M) de la matrice A sont donc de la forme :

$$\text{a} = \left[-\frac{\kappa}{h^2}, \ldots, -\frac{\kappa}{h^2}, -\frac{\kappa}{h}\right]$$
$$\text{b} = \left[\frac{\kappa}{h}, \left(\frac{2\kappa}{h^2} + \frac{h_c p}{S}\right), \ldots, \left(\frac{2\kappa}{h^2} + \frac{h_c p}{S}\right), \frac{\kappa}{h}\right]$$
$$\text{c} = \left[-\frac{\kappa}{h}, -\frac{\kappa}{h^2}, \ldots, -\frac{\kappa}{h^2}\right]$$

Les vecteur X et F sont des vecteurs colonnes de la forme :

$$\text{X} = [T_0, \ldots, T_M]^T$$
$$\text{F} = \left[\Phi_p, \frac{h_c p}{S}T_e, \ldots, \frac{h_c p}{S}T_e, 0\right]^T$$

Sans trop détailler, pour résoudre ce système algébrique, nous aurons besoin de déterminer la factorisation LU de A, ce qui est plutôt facile car c'est une matrice tridiagonale. On appliquera ensuite les algorithmes de descente et de remontée qui nous conduiront directement au vecteur X recherché.
Pour savoir le fonctionnement du programme permettant de résoudre ce système, consulter la section 2. Pour une analyse des résultats, consulter la section 3.1.1.

1.2 Modèle instationnaire

Nous souhaitons maintenant suivre l'évolution en temps des températures de l'ailette pendant une durée notée t_{final}. Pour cela nous devons cette fois-ci prendre en compte la dérivée en temps dans l'équation initiale.
Nous commençons par discrétiser l'intervalle $[0, t_{final}]$ en $N+1$ temps. Le pas de temps sera noté Δt et on notera T_i^n la température calculée au point x_i et à l'instant $t_n = n\Delta t$ pour $n = 0, \ldots, N$. Par la suite, nous utiliserons $t_{final} = 300s$ et $N = 600$, soit un pas de temps $\Delta t = 0.5s$.
Par le schéma explicite d'Euler, on peut discrétiser la dérivée en temps et ainsi on obtient :

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

et de la même manière que pour le modèle stationnaire, le système initial s'écrit :

$$-\frac{\kappa}{h^2}T_{i-1}^{n+1} + \left(\boxed{\frac{\rho C_p}{\Delta t}} + \frac{2\kappa}{h^2} + \frac{h_c p}{S}\right)T_i^{n+1} - \frac{\kappa}{h^2}T_{i+1}^{n+1} = \boxed{\frac{\rho C_p}{\Delta t}T_i^n} + \frac{h_c p}{S}T_e \quad \forall i \in [1, M-1]$$
$$-\frac{\kappa}{h}\left(T_1^{n+1} - T_0^{n+1}\right) = \Phi_p$$
$$-\frac{\kappa}{h}\left(T_{M-1}^{n+1} - T_M^{n+1}\right) = 0$$

On constate que ce système est très similaire à celui du modèle stationnaire. En effet, il n'y a que les parties encadrées qui ont changé. Ainsi, ce problème peut s'écrire sous forme matricielle $A_2 X_{n+1} = \alpha X_n + F$ avec A_2 une matrice tridiagonale (de taille $M+1$), $\alpha X_n + F$ le vecteur second membre (de taille $M+1$) et X_{n+1} le vecteur solution au temps t_{n+1} (de taille $M+1$) qui représente la température en chaque point de discrétisation x_i pour $i = 0, \ldots, M$ au temps t_{n+1}.
La sous-diagonale (a_2 de taille M), la diagonale (b_2 de taille $M+1$) et la sur-diagonale (c_2 de taille M) de la matrice A_2 sont donc de la forme :

$$a_2 = \left[-\frac{\kappa}{h^2}, \ldots, -\frac{\kappa}{h^2}, -\frac{\kappa}{h}\right]$$
$$b_2 = \left[\frac{\kappa}{h}, \left(\frac{\rho C_p}{\Delta t} + \frac{2\kappa}{h^2} + \frac{h_c p}{S}\right), \ldots, \frac{\rho C_p}{\Delta t} + \left(\frac{2\kappa}{h^2} + \frac{h_c p}{S}\right), \frac{\kappa}{h}\right]$$
$$c_2 = \left[-\frac{\kappa}{h}, -\frac{\kappa}{h^2}, \ldots, -\frac{\kappa}{h^2}\right]$$

Le vecteur α est un vecteur ligne et les vecteur X_n, F et X_{n+1} sont des vecteurs colonnes de la forme :

$$\alpha = \left[0, \frac{\rho C_p}{\Delta t}, \ldots, \frac{\rho C_p}{\Delta t}, 0\right]$$
$$X_n = \left[T_0^n, \ldots, T_M^n\right]^T$$
$$\mathrm{F} = \left[\Phi_p, \frac{h_c p}{S}T_e, \ldots, \frac{h_c p}{S}T_e, 0\right]^T$$
$$X_{n+1} = \left[T_0^{n+1}, \ldots, T_M^{n+1}\right]^T$$

Nous partirons de $T_i^0 = T_e$ pour $i = 0, \ldots, M$, c'est-à dire qu'au temps t_0 la solution initiale est la température ambiante. Par la suite nous calculerons X_{n+1} à partir de X_n de la même manière que pour le modèle stationnaire, c'est-à-dire une fois la factorisation LU de A_2 déterminée, on appliquera les algorithmes de descente et de remontée.
Nous traiterons 2 cas du modèle instationnaire. Le premier est un cas où le flux de chaleur restera constant au cours du temps, c'est-à-dire $\Phi_p = 1,25.10^5$ comme dans le cas stationnaire. Le deuxième consistera à activer et désactiver ce flux de chaleur toutes les $30s$ en commençant par l'activer, c'est-à-dire $\Phi_p = 1,25.10^5$ les 30 premières secondes puis $\Phi_p = 0$ les 30 secondes suivantes...
Pour savoir le fonctionnement du programme permettant de résoudre ce système, consulter la section 2. Pour une analyse des résultats, consulter la section 3.1.2.

1.3 Visualisation 3D de la solution

Nous souhaitons maintenant visualiser la solution sur la géométrie 3D de l'ailette qui est un pavé droit paramétré par les longueurs L_x, L_y et L_z. Nous commençons par discrétiser les intervalles $[0, L_x]$, $[0, L_y]$ et $[0, L_z]$ en $M_x + 1$, $M_y + 1$ et $M_z + 1$ points respectivement. On prendra par la suite $M_x = 50$, $M_y = 10$ et $M_z = 30$. Le produit cartésien de ces 3 discrétisations nous donnera un maillage composée de $(M_x+1)(M_y+1)(M_z+1)$ points.
La température en chaque point de notre maillage P_{ijk} sera notée T_{ijk} et pourra être calculée à partir de la solution numérique 1D calculée sur $[0, L_x]$: $T_{ijk} = \hat{T}_i \quad \forall i = 0, \ldots, M_x, j = 0, \ldots, M_y, k = 0, \ldots, M_z$

C'est un interpolant linéaire de la solution 1D au point x_{ijk} qui ne dépendra que de sa position en x et sera calculée à partir de l'algorithme suivant :

Algorithm 2 Calcul de $\hat{T}_i$

for i=0 à M_x **do**
  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 $\hat{T}_i = a\ x_{i00} + b$
end for

Nous aurons grâce à cet algorithme une visualisation 3D de la solution pour le modèle stationnaire mais également pour le modèle instationnaire (cas où le flux est constant et cas activation/désactivation du flux).
Pour savoir le fonctionnement du programme permettant de résoudre ce système, consulter la section 2. Pour une analyse des résultats dans le cas stationnaire consulter la section 3.2.2 et dans le cas instationnaire la section 3.2.3.

2 Structure et contenu du programme

2.1 Compilation-Debug

Avant de pouvoir présenter la structure et le contenu du programme, il faut d'abord savoir comment le compiler et l'exécuter. Pour simplifier l'utilisation du code, j'ai décidé de faire un cmake dans un fichier nommé *CMakeLists.txt* permettant plus précisément de simplifier la compilation et l'exécution de ce programme. Des commentaires ont été rajouté directement dans ce fichier afin d'expliquer la structure de celui-ci.
Par la suite, la compilation et l'exécution du programme se font dans le répertoire *build* prévu uniquement à cet effet. Une fois dans le répertoire il suffit de créer un *Makefile* via la commande "cmake" Pour compiler il suffit de faire "make" et une fois cela fait on peut exécuter notre programme en utilisant la commande "./run.e".
De plus, un fichier *simu.cfg* a été placé dans le répertoire initial *Projet* afin de n'avoir à modifier que ce fichier lors des différents tests réalisés via la modification des paramètres. Un autre fichier de configuration peut être donné en argument lors de l'exécution mais par défaut le fichier *simu.cfg* sera utilisé. Voici la structure à respecter pour que la lecture de ce fichier soit correcte :

simu.cfg

```
Lx 40 Ly 4 Lz 50
M 10000
Phi 0.125
hc 0.0002
Te 20
stationary 1
TFinal 300
N 600
Mx 50 My 10 Mz 30
```

Notons que c'est le paramètre "stationnary" qu'il faut modifier pour traiter le modèle stationnaire (stationnary=1) et instationnaire (stationnary=0).
Notons également que pour vérifier qu'il n'y ait aucune perte mémoire, j'ai utilisé un débogueur intitulé valgrind permettant de résoudre ce type de problème. A la fin du projet, il n'y en avait plus aucune dans le cas stationnaire comme dans le cas instationnaire. On constate dans les images suivantes que toute la mémoire allouée a bien été libérée.

Cas stationnaire :

```
Création fichier solution 2D modèle stationnaire.
Création fichier solution 3D modèle stationnaire.
==28346==
==28346== HEAP SUMMARY:
==28346==     in use at exit: 0 bytes in 0 blocks
==28346==   total heap usage: 59 allocs, 59 frees, 2,191,186 bytes allocated
==28346==
==28346== All heap blocks were freed -- no leaks are possible
==28346==
==28346== For lists of detected and suppressed errors, rerun with: -s
==28346== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Cas instationnaire :

```
Création fichier solution 2D modèle instationnaire flux constant.
Création fichier solution 3D modèle instationnaire flux constant :
[====================] 100%
Création fichier solution 2D modèle instationnaire activ/desactiv.
Création fichier solution 3D modèle instationnaire activ/desactiv :
[====================] 100%
==21042==
==21042== HEAP SUMMARY:
==21042==     in use at exit: 0 bytes in 0 blocks
==21042==   total heap usage: 13,336 allocs, 13,336 frees, 400,171,234 bytes allocated
==21042==
==21042== All heap blocks were freed -- no leaks are possible
==21042==
==21042== For lists of detected and suppressed errors, rerun with: -s
==21042== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

2.2 Structure et contenu du répertoire *Projet*

Nous pouvons maintenant nous intéresser un peu plus à la structure du programme. Voici un schéma qui représente l'arborescence du dossier *Projet* :

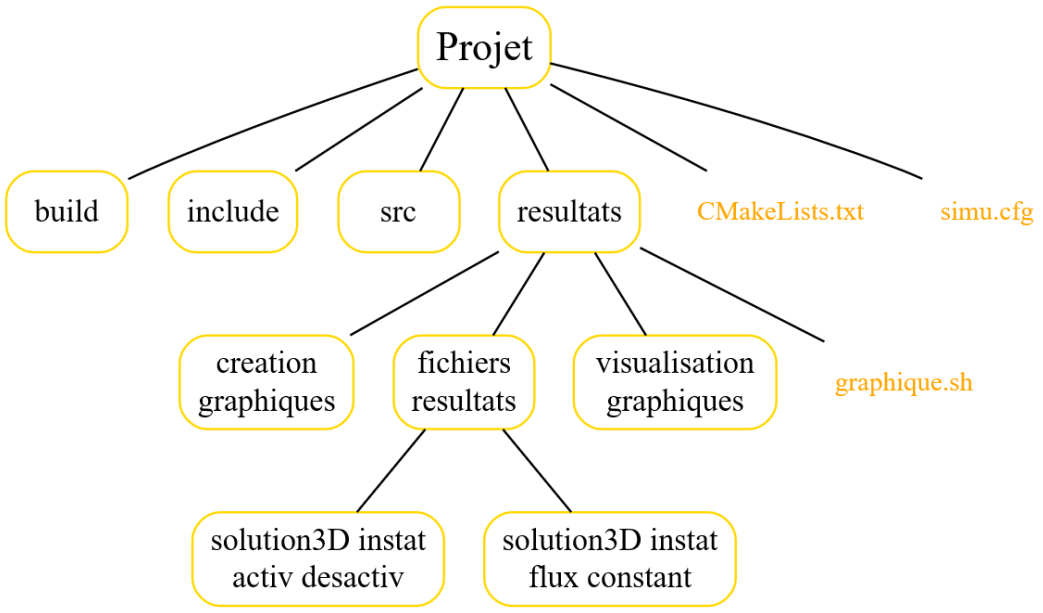

FIGURE 1 – Arborescence du répertoire *Projet*

Nous détaillerons par la suite le contenu précis de chacun de ces répertoires mais en voici un résumé :

- Le dossier *build* sert uniquement à stocker les fichiers générés par la compilation, dont l'exécutable "run.e". Pour plus de détails, consulter la section précédente 2.1.

- Les dossier *include* et *src* stockent respectivement les fichiers headers (".hpp") et les fichiers sources (".cpp"). Pour plus de détails, consulter la section suivante 2.3.
- Le dossier *résultats* contient trois autres répertoires :
  - *creation_graphiques* qui contient des fichiers python (".py") pour pouvoir créer les représentations visuelles des résultats.
  - *fichiers_resultats* qui contient tous les fichiers ".csv" et ".vtk" correspondant aux résultats pour les 2 modèles.
  - *visualisation_graphiques* qui contient toutes les images et vidéos générées par les fichiers python.

  Il contient également un script shell "graphique.sh" permettant de générer plus facilement les images et vidéos voulues. Pour plus de détails, consulter la section 2.4.
- Le fichier "CMakeLists.txt" contient le cmake de ce projet, permettant de compiler le programme. Pour plus de détails, consulter la section précédente 2.1.
- Le fichier "simu.cfg" est un fichier de configuration des paramètres des modèles stationnaire et instationnaire. Pour plus de détails, consulter la section précédente 2.1.

2.3 Contenu des répertoires *include/src*

Nous pouvons à présent nous concentrer sur le projet lui-même. Voici un schéma qui représente le contenu des répertoires *include/src* :

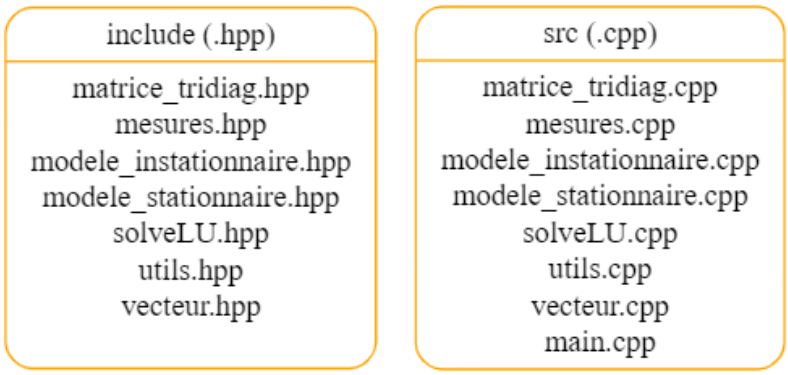

FIGURE 2 – Contenu des répertoires *include* et *src*

Nous allons donc nous intéresser un peu plus à la structure du code lui-même. A noter que les commentaires dans ces fichiers expliquent le but de chaque fonction mais nous allons tout de même ici détailler leur utilité.
Pour information, je n'ai pas trouvé utile de créer une classe maillage, car les points de celui-ci ne sont pas utiles lors de la résolution des modèles stationnaire et instationnaire. Ils sont juste utilisés lors de l'écriture dans le fichiers vtk ce qui ne rend pas nécessaire selon moi le stockage de toutes ces coordonnées.

2.3.1 Fichiers communs aux deux modèles

Tout d'abord, avant de nous intéresser aux fonctions et classes spécifiques à la résolution des modèles, il a fallu développer des outils permettant de résoudre plus facilement les modèles stationnaire et instationnaire.

- La première étape consistait à créer une classe **Mesures**, qui lit le fichier de configuration directement dans le constructeur et initialise dans un même objet tous les paramètres nécessaires à la résolution. Ainsi, au lieu de donner à nos modèles énormément d'attributs il suffira de leur passer un objet de type Mesures permettant d'accéder plus facilement à chaque paramètre. Voici une liste des attributs de cette classe :

```
const int rho=2700, C_p=940, K=164;
double L_x,L_y,L_z;
int M;
double phi,h_c;
int T_e;
bool stationary;
int T_final;
int N;
int M_x,M_y,M_z;
```

Cette classe est entièrement implémentée dans les fichiers "mesures.hpp" et "mesure.cpp".

- Par la suite, j'ai décidé d'implémenter une classe **Matrice_tridiag** dans le but de simplifier la résolution des systèmes algébriques. Elle possède un attribut M (entier) qui représente la taille de la matrice et regroupe 3 tableaux M_a (de taille M-1), M_b (de taille M) et M_c (de taille M-1) qui représentent respectivement la sous-diagonale, la diagonale et la sure-diagonale de la matrice.
  Elle possède également les constructeurs, le destructeur et les accesseurs nécessaires à son utilisation et est entièrement implémentée dans les fichiers "matrice_tridiag.hpp" et "matrice_tridiag.cpp".
- De la même manière, j'ai trouvé cela plus facile d'implémenter une classe **Vecteur** ne possédant comme attribut qu'un entier M pour sa taille et un tableau M_v.
  Elle possède également les constructeurs, le destructeur, les accesseurs et les surcharges d'opérateurs nécessaires à son utilisation et est entièrement implémentée dans les fichiers "vecteur.hpp" et "vecteur.cpp".
- Pour finir, un fichier nommé "utils.cpp" (avec son header "utils.hpp") contient toutes les fonctions communes nécessaires aux 2 modèles, autrement dit les fonctions "générales" du programme. Nous en parlerons au fur et à mesure dans la suite.

2.3.2 Fichiers du modèle stationnaire

La résolution complète du modèle stationnaire est dans les fichiers "modele_stationnaire.hpp" et "modele_stationnaire.cpp". Pour simplifier l'implémentation, une classe nommée **Modele_stationnaire** a été créée. En voici ses attributs :

```
Mesures* M_mesure;
Matrice_tridiag* M_A;
Vecteur* M_F;
Vecteur* M_sol;
```

Elle contient donc un pointeur sur une Mesure, regroupant tous les paramètres nécessaires à la résolution du modèle. Elle contient également un pointeur sur une Matrice_tridiag M_A, un pointeur sur un Vecteur M_F et un autre pointeur sur un Vecteur M_sol tels que $M\_A * M\_sol = M\_F$.
La résolution du système algébrique se fait directement à l'instanciation de l'objet. Le constructeur prend en paramètre un pointeur sur une Mesures et utilise les fonctions privées suivantes :

- Un appel à une fonction privée **def_matrice_tridiag_stationnaire** permet d'initialiser l'attribut M_A via les paramètres d'un objet de la classe Mesures.
- Un appel à une fonction privée **def_second_membre** permet d'initialiser l'attribut M_F via les mêmes paramètres.
- Un dernier appel à une fonction privée **resolution_stat_1D** permettra de résoudre (dans le plan) le modèle stationnaire. Pour une question de lisibilité et étant donné que nous en aurons également besoin pour le modèle instationnaire, cette fonction utilise d'autres fonctions implémentées dans le fichier "utils" :
  - Un appel à une fonction **def_LU(const Matrice_tridiag A)** prenant en paramètre une Matrice_tridiag et renvoyant un tableau de deux pointeurs sur des Matrice_tridiag permet d'obtenir la factorisation LU de la matrice tridiagonale M_A.
  - Un appel à la fonction **Descente_mat_tridiag(Vecteur& Y, const Matrice_tridiag& L ,const Vecteur& F)** prenant en paramètres deux références sur des Vecteur et une référence sur une Matrice_tridiag permet d'effectuer l'algorithme de descente et de stocker le résultat dans le Vecteur Y passé en paramètre.
  - Un appel à la fonction **Remontee_mat_tridiag(Vecteur& X, const Matrice_tridiag& U ,const Vecteur& Y)** prenant en paramètres deux références sur des Vecteur et une référence sur une Matrice_tridiag permet d'effectuer l'algorithme de remontée et de stocker le résultat dans le Vecteur X passé en paramètre. A noter que ce Vecteur sera l'attribut M_sol de notre classe.

Enfin, pour pouvoir visualiser la solution du modèle stationnaire dans le plan et la comparer avec la solution analytique, il faudra faire appel à une fonction publique nommée **visualisation1D_solution_stationnaire**. Cette méthode aura but de créer un fichier nommé "solution1D_stat_Lx40.csv" (si par exemple Lx=40) et d'y écrire en première colonne la discrétisation de notre intervalle $[0, Lx]$, en deuxième colonne notre solution M_sol et en troisième et dernière colonne notre solution analytique obtenue à partir d'un appel à une méthode privée **T_exact** de notre classe Modele_stationnaire.
Pour finir, pour pouvoir visualiser la solution 3D du modèle stationnaire, il faudra faire appel à une fonction publique nommée **visualisation3D_solution_stationnaire**. Cette méthode aura but de créer un fichier nommé

"solution3D_stat.vtk" et d'y écrire dans un premier temps les points de discrétisation de notre maillage en faisant appel à une fonction **def_maillage(const Mesures& mesure, std::ofstream& ofile)** implémentée dans le fichier "utils" qui prend en paramètre une Mesures ainsi qu'une référence sur un std::ofstream. Puis, dans un second temps notre fonction de visualisation 3D pour le modèle stationnaire écrira la température interpolée associée à chaque point de notre maillage et calculée grâce à la fonction **def_temperature_interpolee(const Mesures& mesure, Vecteur& sol)** qui se trouve également dans le fichier "utils".
Pour plus d'informations sur la visualisation des résultats, consulter la section 2.4 qui permet de comprendre comment est structuré le répertoire avec les résultats et explique comment créer les graphiques associés. Consulter également la section 3 pour une analyse de ces résultats.

2.3.3 Fichiers du modèle instationnaire

Contrairement au modèle stationnaire, nous distinguons ici 2 cas : un cas où le flux de chaleur Φ_p généré par le processeur reste constant, un autre cas où il y aura une activation/désactivation de ce flux de chaleur (toutes les 30 secondes).
Dans un premier temps, j'ai décidé de créer une classe **Modele_instationnaire** qui se trouve dans les fichiers "modele_instationnaire.hpp" et "modele_instationnaire.cpp". En voici ses attributs :

```
Mesures* M_mesure;
Matrice_tridiag* M_A;
Vecteur* M_F;
Vecteur** M_sol;
```

Elle contient donc un pointeur sur une Mesure, regroupant tous les paramètres nécessaires à la résolution du modèle. Elle contient également un pointeur sur une Matrice_tridiag M_A, un pointeur sur un Vecteur M_F et un tableau de pointeurs nommé M_sol sur des Vecteur (tableau à N+1 lignes de pointeurs sur des Vecteur de taille M+1).
Par la suite, l'instanciation d'un objet de cette classe appellera le constructeur (de paramètre un pointeur sur une mesure) qui utilise lui-même des fonctions privées permettant de résoudre le système algébrique :

- Un appel à une fonction privée **def_matrice_tridiag_instationnaire** permet d'initialiser l'attribut M_A via les paramètres d'un objet de la classe Mesures.
- Un appel à une fonction privée **def_second_membre** permet d'initialiser l'attribut M_F via les mêmes paramètres.
- Contrairement au modèle stationnaire, l'initialisation de M_sol ne se fait pas directement dans ce constructeur car cet attribut diffère en fonction du cas dans lequel on se trouve (flux de chaleur constant ou activation/désactivation du flux de chaleur).

Elle possède également un destructeur (virtuel à cause de l'héritage) qui est plus que nécessaire ici étant donné la taille de M_sol.
Par la suite, j'ai choisi de traiter les 2 cas en créant deux classes, héritant toutes deux de la classe Modele_instationnaire.

2.3.3.1 Cas flux constant

Dans le cas où le flux est constant, j'ai créé une classe **Modele_instationnaire_flux_const** héritant de Modele_instationnaire qui ne possède aucun attribut supplémentaire à la classe mère. Cependant son constructeur qui appelle le constructeur de la classe mère Modele_instationnaire initialise le tableau de Vecteur M_sol en faisant appel à une fonction **def_solution_flux_constant**. Cette fonction appelle, de la même manière que dans le modèle stationnaire, les fonctions **def_LU**, **Descente_mat_tridiag** ainsi que **Remontee_mat_tridiag** qui se trouvent dans le fichier "utils". Cette classe possède également un destructeur.
Enfin pour pouvoir visualiser la solution dans le plan, il faudra faire appel à une fonction publique nommée **visualisation1D_solution_instationnaire_flux_constant**. Cette méthode aura but de créer un fichier nommé "solution1D_instat_flux_constant.csv" et d'y écrire en première colonne la discrétisation de notre intervalle $[0, Lx]$. Cette fonction appelle une méthode privée **donnees_a_analyser_instationnaire_flux_constant** qui permet de renvoyer un tableau avec les Vecteurs au temps demandés (ici un tableau de taille 6 au temps t=15s,30s,60s,90s,150s et 210s). Les colonnes suivantes du fichier csv contiendront ainsi les températures aux temps voulus.
Pour finir pour pouvoir visualiser la solution 3D, il faudra faire appel à une fonction publique nommée **visualisation3D_solution_instationnaire_flux_constant**. Cette méthode aura but de créer $N+1$ fichiers nommés "solution3D_flux_constant."n".vtk" (avec $n = 0, \ldots, N$). Chacun de ces fichiers contiendra dans un premier temps les points de discrétisation de notre maillage via un appel à la même fonction que pour le modèle stationnaire **def_maillage(const Mesures& mesure,std::ofstream& ofile)**. Puis pour chacun des fichiers notre fonction

de visualisation 3D écrira la température interpolée associée à chaque point de notre maillage et calculée grâce à la même fonction que pour le modèle stationnaire **def_temperature_interpolee(const Mesures& mesure, Vecteur& sol)** qui se trouve également dans le fichier "utils".
Étant donné que la création de ces $N+1$ fichiers peut être un peu lente, j'ai décidé de créer une "barre de progression" qui permet de visualiser un peu mieux l'évolution de cette étape. Pour ce faire, notre fonction de visualisation 3D fait appel au cours de ses itérations à une fonction **affiche_progression** qui se trouve dans le fichier "utils". En voici un aperçu :

```
Création fichier solution 3D modèle instationnaire flux constant :
[=========>          ]  45%
Création fichier solution 3D modèle instationnaire flux constant :
[====================] 100%
```

De plus, pour vérifier que notre solution converge vers la solution du modèle stationnaire, j'ai ajouté dans cette classe une méthode publique nommée **visualisation_convergence_modele(Vecteur* sol_stat)** qui prend en paramètre un pointeur sur un Vecteur qui est la solution du modèle stationnaire. Cette fonction génère un fichier nommé "convergence_instat_vers_stat.csv" avec en première colonne les temps discrets et en deuxième colonne le maximum des différences de température en norme absolue entre le modèle instationnaire et le modèle stationnaire pour chaque temps.
Pour plus d'informations sur la visualisation des résultats, consulter la section 2.4 qui permet de comprendre comment est structuré le répertoire avec les résultats et explique comment créer les graphiques associés. Consulter également la section 3 pour une analyse de ces résultats.

2.3.3.2 Cas activation-désactivation du flux

J'ai traité de la même manière le cas où il y a activation/désactivation du flux que le cas où celui-ci est constant. J'ai créé une classe **Modele_instationnaire_activ_desactiv** héritant de Modele_instationnaire qui ne possède aucun attribut supplémentaire à la classe mère. Cependant son constructeur qui appelle le constructeur de la classe mère Modele_instationnaire initialise le tableau de Vecteur M_sol en faisant appel à une fonction **def_solution_activ_desactiv**. Cette fonction appelle, de la même manière que dans la modèle stationnaire, les fonctions **def_LU**, **Descente_mat_tridiag** ainsi que **Remontee_mat_tridiag** qui se trouvent dans le fichier "utils". Cette classe possède également un destructeur.
Enfin pour pouvoir visualiser la solution dans le plan, il faudra faire appel à une fonction publique nommée **visualisation1D_solution_instationnaire_activ_desactiv**. Cette méthode aura but de créer un fichier nommé "solution1D_instat_activ_desactiv.csv" et d'y écrire cette-fois-ci en première colonne les temps traités (de 0s à 300s avec un pas de 0.5 soit 601 temps). L'appel de la fonction privée **donnees_a_analyser_instationnaire_activ_desactiv** permet cette-fois-ci de sauvegarder un tableau avec les températures au cours du temps associées à certains points (ici un tableau de taille 3 pour x_0, $x_{M/2}$ et x_M). Les colonnes suivantes du fichier csv contiendront ainsi ces valeurs.
Pour finir pour pouvoir visualiser la solution 3D, il faudra faire appel à une fonction publique nommée **visualisation3D_solution_instationnaire_activ_desactiv**. Cette méthode aura but de créer $N+1$ fichiers nommés "solution3D_activ_desactiv."n".vtk" (avec $n = 0, \dots, N$). Chacun de ces fichiers contiendra dans un premier temps les points de discrétisation de notre maillage via un appel à la même fonction que précédemment **def_maillage(const Mesures& mesure,std::ofstream& ofile)**. Puis pour chacun des fichiers notre fonction de visualisation 3D écrira la température interpolée associée à chaque point de notre maillage et calculée grâce à la même fonction que précédemment **def_temperature_interpolee(const Mesures& mesure, Vecteur& sol)** qui se trouve également dans le fichier "utils".
Étant donné que la création de ces $N+1$ fichiers peut être un peu lente, j'ai décidé de créer une "barre de progression" qui permet de visualiser un peu mieux l'évolution de cette étape. Pour ce faire, notre fonction de visualisation 3D fait appel au cours de ses itérations à une fonction **affiche_progression** qui se trouve dans le fichier utils. En voici un appreçu :

```
Création fichier solution 3D modèle instationnaire activ/desactiv :
[=========>          ]  45%
Création fichier solution 3D modèle instationnaire activ/desactiv :
[====================] 100%
```

Pour plus d'informations sur la visualisation des résultats, consulter la section 2.4 qui permet de comprendre comment est structuré le répertoire avec les résultats et explique comment créer les graphiques associés. Consulter également la section 3 pour une analyse de ces résultats.

2.4 Structure et contenu du répertoire *resultats*

Afin de pouvoir au mieux visualiser les solutions 1D et 3D de chaque modèle, un répertoire *resultats* a été créé. Voici un schéma qui représente l'arborescence de ce dossier :

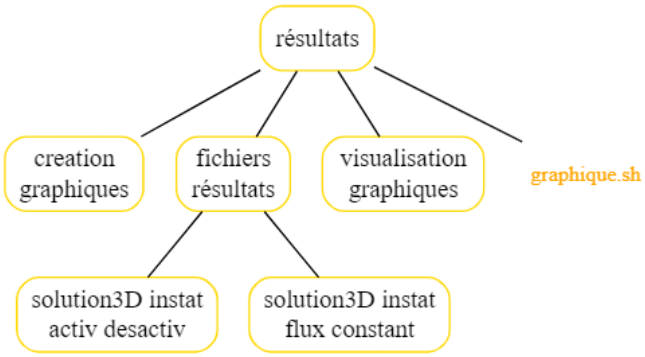

FIGURE 3 – Arborescence du répertoire *resultats*

2.4.1 Contenu du répertoire "fichier_resultats"

Dans un premier temps, il est important de savoir que c'est le répertoire *fichiers_resultat* qui contient tous les fichiers ".csv" et ".vtk" qui ont été créés lors de l'exécution. Voici ce qu'il contient :

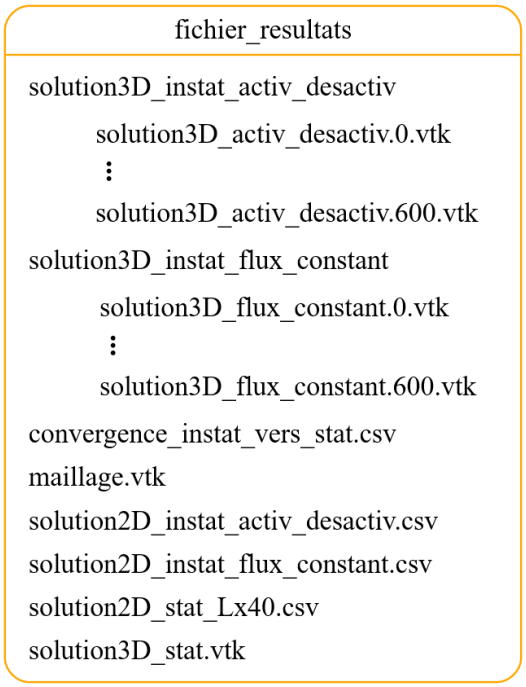

FIGURE 4 – Contenu du répertoire *fichiers_resultats*

Ce repertoire possède un fichier "maillage.vtk" permettant de visualiser le maillage 3D de l'ailette.
Il contient aussi la solution 1D du modèle stationnaire nommée "solution1D_stat_Lx40.csv" (si Lx=40) ainsi que sa solution 3D nommée "solution3D_stat.vtk".
Il contient également les solutions 1D du modèle instationnaire dans les 2 cas, nommés "solution1D_instat_flux_constant.csv" et "solution1D_instat_activ_desactiv.csv" et possède 2 dossiers **solution3D_instat_activ_desactiv** et **solution3D_instat_flux_constant** contenant chacun les $N+1$ fichiers ".vtk" avec la solution 3D en fonction du temps pour les 2 cas du modèle instationnaire. Il possède aussi un fichier "convergence_instat_vers_stat.csv" permettant de vérifier que le modèle instationnaire (dans le cas où le flux est constant) converge bien vers le modèle stationnaire.

2.4.2 Visualisation des résultats

Pour une question de simplicité, j'ai décidé de créer un script shell nommé "graphique.sh" qui permet de générer les graphiques et vidéos à partir des fichiers ".csv" et ".vtk" qui se trouvent dans le répertoire *fichiers_ resultats* (après l'exécution du programme). Ce script utilise des fichiers python qui créent les images et graphiques souhaités.

- Tout d'abord, intéressons-nous au répertoire *creation_ graphiques* où se trouvent tous les fichiers python qui génèrent nos images et nos vidéos. Ce répertoire est dans le dossier *resultats*, voir la Figure 3. Ces images et vidéos seront générées dans le répertoire *visualisation_ graphiques*.
  Voici donc ce que contiennent ces 2 répertoires :

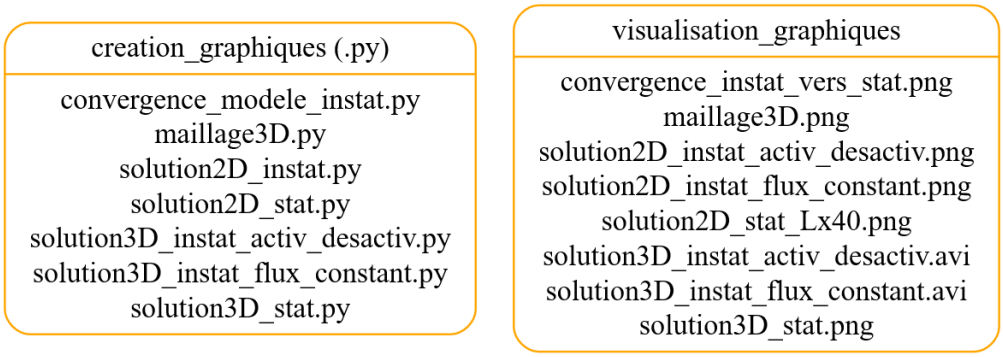

FIGURE 5 – Contenu des répertoires *creation_ graphiques* et *visualisation_ graphiques*

Les fichiers python qui permettent la création des représentations en 3D ont été générés grâce à l'outil Trace proposé par Paraview. Cet outil permet d'exporter l'ensemble des actions effectuées sur le logiciel sous la forme d'un script python. Il a tout de même fallu effectuer quelques modifications de ces scripts.
A noter que l'installation d'un programme nommé **pvpython** m'a été nécessaire pour exécuter ces scripts. De plus, pour les graphiques en 1D, seul **matplotlib** a été nécessaire.
Voici un schéma qui permet de visualiser quelle est l'image ou la vidéo qui est créée par chacun des fichiers pythons :

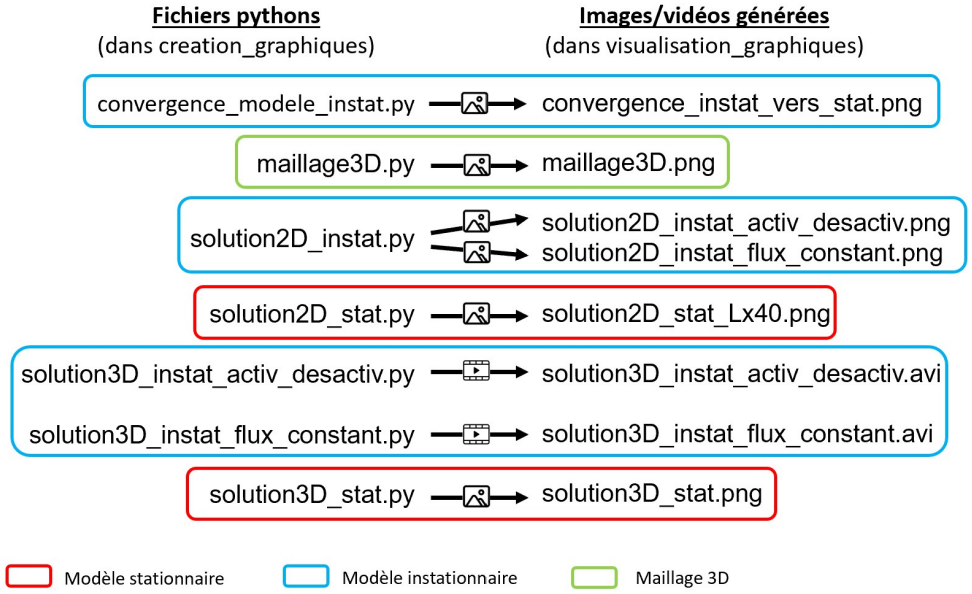

FIGURE 6 – Lien entre les fichiers python et les représentations visuelles

Pour une analyse de ces images, consulter la section 3.

- Par la suite, pour éviter de devoir manipuler ces fichiers python, un script shell nommé "graphique.sh" a été créé. Pour l'executer il suffit de se trouver dans le répertoire *resultats*. Ce script prend en paramètre 1 argument permettant de choisir quelles image et/ou vidéos seront créées. L'exécution de ce script sans argument (via la commande "./graphique.sh") affiche une aide indiquant les choix possibles pour le paramètre :

```
0 = Tout
1 = Cas stationnaire
2 = Cas instationnaire
3 = Maillage 3D
```

FIGURE 7 – Options du script shell "graphique.sh"

- La valeur 0 permettra de tout exécuter. Autrement-dit c'est le regroupement des valeurs 1, 2 et 3.
- La valeur 1 permettra de ne créer que les images du modèle stationnaire (exécute les fichiers python en rouge sur la Figure 6).
- La valeur 2 permettra de ne créer que les images et vidéos du modèle instationnaire (exécute les fichiers python en bleu sur la Figure 6).
  A noter que pendant l'exécution avec cet argument, la question : "Voulez-vous créer les vidéos du modèle instationnaire ? (0=Non,1=Oui)" vous sera posée car la création de ces vidéos peut prendre du temps, c'est pourquoi vous avez le choix de les créer ou non (avec 1 pour Oui, avec 0 pour Non).
- La valeur 3 permettra de créer l'image représentant le maillage 3D (exécute le fichier python en vert sur la Figure 6).

Pour plus d'informations sur la conception de ce script, consulter le directement. Il se trouve dans le répertoire *resultats*.

3 Analyse des résultats

Tous les graphiques et toutes les vidéos qui se trouvent dans la suite peuvent être générés à partir des fichiers ".csv" et ".vtk". On les trouve également à l'adresse suivante :
https://drive.google.com/drive/folders/1bbHcF0OYLV-O3zvVthPx1fRulQHv9CMd?usp=sharing

3.1 Solution 1D des modèles

3.1.1 Modèle stationnaire

Pour la solution 1D du modèle stationnaire, comparons les résultats pour 2 tailles d'ailettes différentes (Lx=40 et Lx=80) :

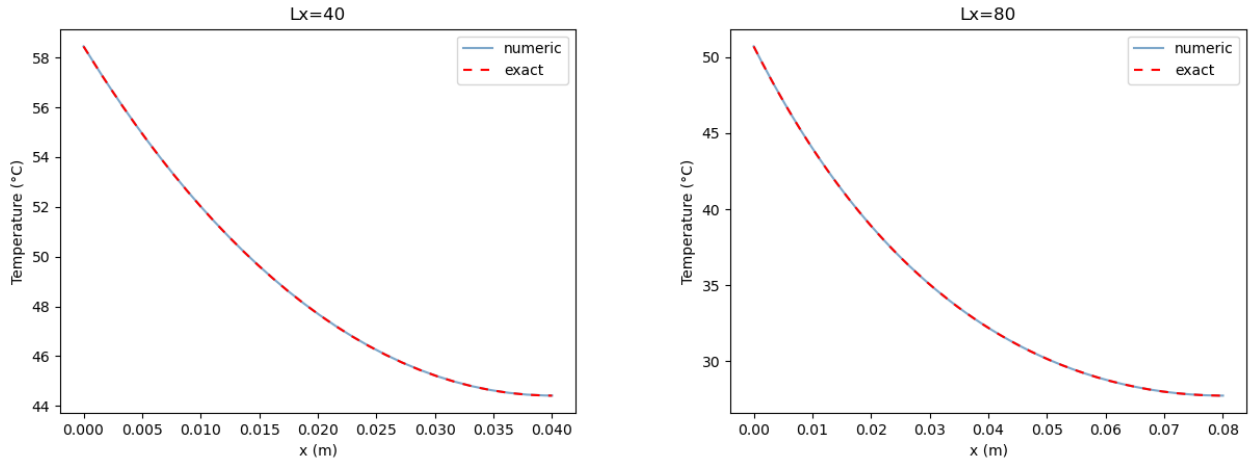

FIGURE 8 – Solution 1D du modèle stationnaire (pour Lx=40 et Lx=80)

Les courbes en bleu représentent les solution calculées à partir du système algébrique.
Celles en tirets rouges sont les solutions analytiques calculées à partir de la formule suivante :

$$T_{exact}(x) = T_e + \frac{\Phi_p}{\kappa} \frac{\cosh(\sqrt{a}L_x)}{\sqrt{a}\sinh(\sqrt{a}L_x)} \frac{\cosh(\sqrt{a}(L_x - x))}{\cosh(\sqrt{a}L_x)} \qquad \text{avec } a = \frac{h_c p}{\kappa S}$$

On constate dans un premier temps que visuellement les solutions calculées semblent correspondre aux solutions exactes.
Par la suite, il semble logique que la température à gauche de l'ailette, proche du processeur soit plus élevée et qu'elle diminue plus on s'en éloigne. Cela montre qu'il y a bien dissipation de la chaleur. De plus, plus la longueur de l'ailette est grande et plus les températures semblent diminuer. En effet, pour Lx=40 les températures sont comprises environ entre 44 et 59 °C et pour Lx=80 elles sont comprises entre 25 et 51 °C.
On remarque également que l'utilisation du ventilateur a un fort impact sur la température de l'ailette. Voici ce que l'on obtient lorsque le ventilateur est éteint (simulé par une modification du paramètre h_c à 10 au lieu de 200) :

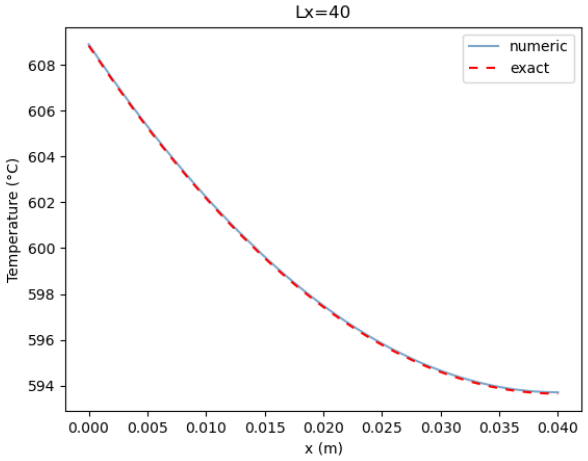

FIGURE 9 – Solution 1D du modèle stationnaire (ventilateur éteint)

Pour les mêmes paramètres que précédemment (excepté h_c), la température dans l'ailette semble avoir été multipliée par 10.

3.1.2 Modèle instationnaire

Pour le modèle instationnaire, nous avons traité 2 cas :

- Dans le cas où le flux de chaleur reste constant, nous avons choisi de tracer la température de l'ailette à différents temps :

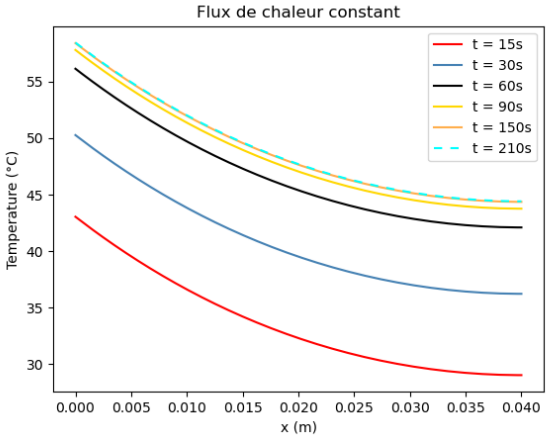

FIGURE 10 – Solution 1D du modèle instationnaire (flux constant)

On constate dans un premier temps que, comme pour le modèle stationnaire, la température diminue plus on s'éloigne du processeur (plus x augmente). De plus, plus le temps augmente plus l'ailette chauffe, ce qui paraît logique étant donné que le flux de chaleur est constant. La solution semble stagner à partir d'un certain temps mais l'ailette ne refroidira pas.
De plus, il est intéressant de vérifier que le modèle instationnaire (flux constant) converge vers le modèle stationnaire. Pour cela, j'ai tracé la différence maximale en norme absolue entre chaque Vecteur X_n (solution

du modèle instationnaire flux constant au temps t_n) et X (solution du modèle stationnaire). On constate sur le graphique suivant que cette différence diminue lorsque le temps augmente :

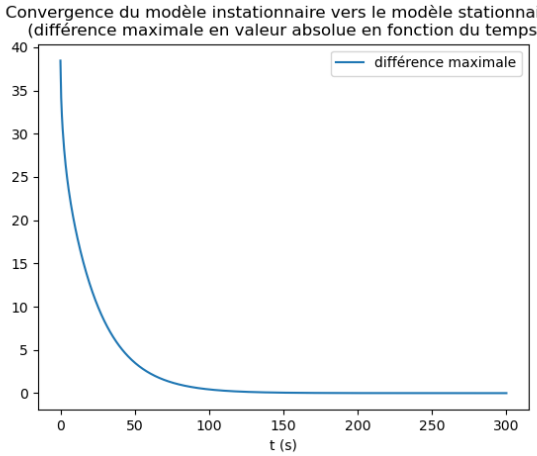

FIGURE 11 – Convergence du modèle instationnaire vers le modèle stationnaire

- Dans le cas où il y activation et désactivation du flux, nous avons choisi de tracer la température en fonction du temps en 3 points de l'ailette (au début, au milieu et à la fin) :

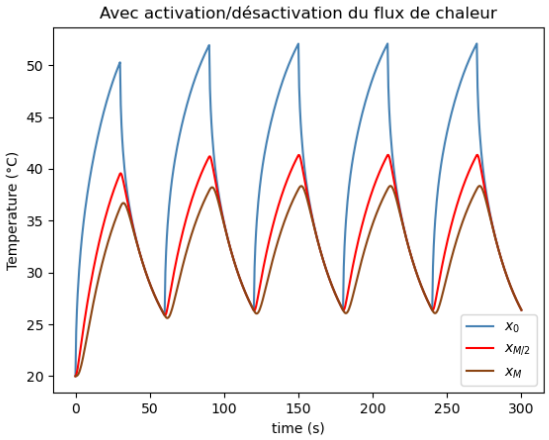

FIGURE 12 – Solution 1D du modèle instationnaire (activation/desactivation du flux)

Tout d'abord, on constate que la température au début de l'ailette (en $x_0 = 0$) est toujours supérieure (ou presque égale) à celle au milieu de l'ailette (en $x_{M/2} = L_x/2$) qui est elle-même toujours supérieure (ou presque égale) à celle à la fin de l'ailette (en $x_M = L_x$) et ceci peu importe le temps. Cela semble plutôt similaire au modèle stationnaire et au modèle instationnaire avec flux de chaleur constant où la chaleur était plus importante proche du processeur qu'au bout de l'ailette.
De plus, dans nos essais, nous avons choisi d'activer et de désactiver le flux de chaleur toutes les 30 secondes (en commençant par l'activer de 0 à 30 secondes). Ceci se voit de manière évidente dans l'image. En effet, les 3 courbes sont croissantes les 30 premières secondes (lorsque le flux de chaleur est activé) puis décroient les 30 secondes suivantes (lorsque le flux est désactivé) et ceci se répète par la suite au cours du temps.

3.2 Solution 3D des modèles

Nous souhaitons maintenant interpoler les solutions précédentes pour nous représenter la température sur toute l'ailette (en 3D) :

3.2.1 Maillage

Nous avons donc commencé par définir un maillage comprenant $(M_x + 1) * (M_y + 1) * (M_z + 1)$ points. Voici à quoi il ressemble pour $M_x = 50$, $M_y = 10$ et $M_z = 30$:

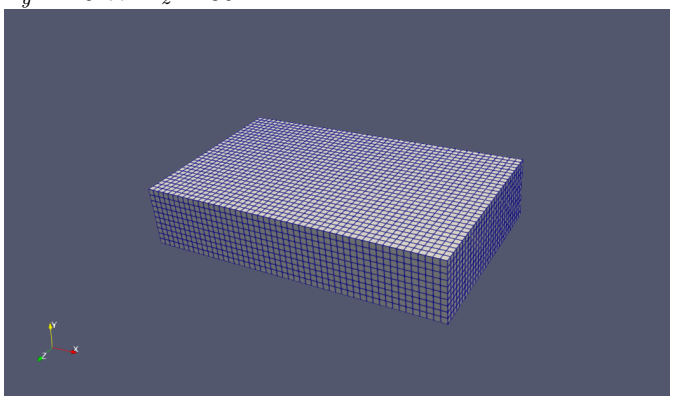

FIGURE 13 – Maillage 3D (pour $M_x = 50$, $M_y = 10$ et $M_z = 30$)

Ce sont ces paramètres qui ont été choisis pour les tests qui suivent.

3.2.2 Modèle stationnaire

Dans le cadre du modèle stationnaire, voici ce que l'on obtient :

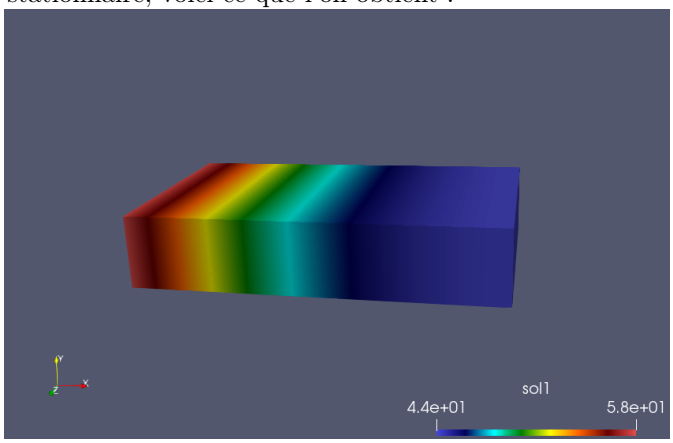

FIGURE 14 – Solution 3D du modèle stationnaire

On constate comme pour la solution 1D que plus on s'éloigne du bord gauche de l'ailette plus les températures sont basses.

3.2.3 Modèle instationnaire

Pour le modèle instationnaire, nous avons traité 2 cas et avons obtenu des vidéos représentant les températures interpolées sur toute l'ailette au cours du temps. Je vous conseille donc de les regarder. J'ai cependant trouvé cela intéressant de mettre en avant quelques moments importants de la vidéo.

- Dans le cas où le flux est constant, voici ce que l'on obtient :

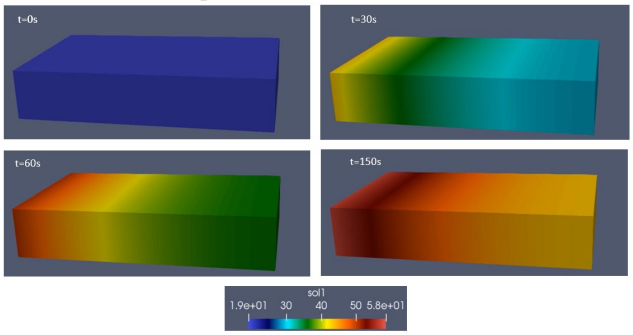

FIGURE 15 – Solution 3D du modèle instationnaire (flux constant)

Comme précédemment, il semblerait qu'en tout temps, la chaleur la plus élevée se situe sur la gauche de l'ailette et que le flux de chaleur la parcourt de gauche à droite. De plus, comme pour la solution 1D dans le cadre instationnaire flux constant la chaleur semble augmenter au fur et à mesure du temps et stagner à partir d'un moment. Consulter la vidéo pour une meilleure visualisation du résultat.

- Dans le cas où il y a activation et désactivation du flux, voici ce que l'on obtient :

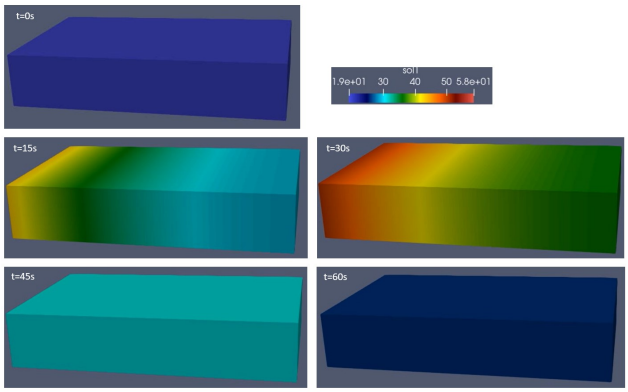

FIGURE 16 – Solution 3D du modèle instationnaire (activation/désactivation du flux)

Pour $t = 0s$, on a bien toute l'ailette qui est à $T_e = 20°C$ puis pendant les 30 premières secondes (lorsque le flux de chaleur est activé), la chaleur augmente puis diminue pendant les 30 secondes suivantes (lorsqu'il est désactivé). Ce phénomène se répète jusqu'à la fin. De plus, le flux de chaleur parcourt bien l'ailette comme on s'y attendait, c'est-à-dire de gauche à droite. Consulter la vidéo pour une meilleure visualisation du résultat.