LIÈGE université
Sciences Appliquées

PROJ0001-1 INTRODUCTION
AUX MÉTHODES NUMÉRIQUES ET PROJET

UNIVERSITÉ DE LIÈGE

FACULTÉ DES SCIENCES APPLIQUÉES

ANNÉE ACADÉMIQUE 2024-2025

Rapport de Projet : Régulation d'un bâtiment thermiquement actif

Professeurs :
Olivier BRULS
Quentin LOUVEAUX
Frédéric NGUYEN

Auteurs :
Benjamin BOCK (s2304467)
Odayfa DAKIR (s2305341)
Yazan SALOUM (s2304645)

Liège, le 6 avril 2025

Table des matières

1 Recherche de racines **2**
1.1 Bissection . . . 2
1.2 Sécante . . . 3
1.3 Comparaison des deux fonctions . . . 3

2 Pertes et gains de chaleur **3**
2.1 Importer `PerteEtGain.txt` . . . 3
2.2 Interpoler $G(t)$ . . . 4
2.3 Vérification graphique . . . 4

3 Mise en place de la modélisation **4**
3.1 Encodage de `odefunction` . . . 4
3.2 Méthode d'Euler . . . 6
3.3 Méthode Runge-Kutta d'ordre 4/5 . . . 6
3.4 Convergence d'Euler en fonction de h . . . 7
3.5 État stationnaire . . . 7
3.6 Comparer les scénarios . . . 8

4 Régulation automatisée du bâtiment **8**
4.1 Détermination de T_{max} . . . 8
4.2 $\Delta t_{optimal}$ pour 24 heures . . . 10
4.3 $\Delta t_{optimal}$ à l'état stationnaire . . . 10

Table des figures

1 Allure de la fonction $G(t)$ . . . 5
2 Comparaison des températures entre Euler et Runge-Kutta 45 . . . 6
3 Détermination de l'erreur moyenne de la méthode d'Euler . . . 7
4 Convergence vers l'état stationnaire des températures de la pièce et de la partie inférieure du béton. . . . 8
5 Évolution des températures suivant les différents scénarios . . . 9
6 Détermination de la température max lors des premières 24h avec $\Delta t = 1$ . . . 10
7 Évolution de la température de confort avec $\Delta t_{optimal} = 2.46h$ et $T^d_{max} = 21°\text{C}$ . . . 11
8 Convergence du système vers l'état stationnaire . . . 11
9 Évolution de $T_{confort}$ à l'état stationnaire . . . 11

Liste des tableaux

1 Comparaison des fonctions `bissection` et `secante` . . . 4

1 Recherche de racines

Cette première question vise à modéliser deux fonctions : `secante(f,x0,x1,tol)` et `bissection(f,x0,x1,tol)`. Ces fonctions jouent le même rôle, qui est de rechercher la racine d'une fonction $f(x)$ donnée, mais ont des fonctionnements différents. Ainsi, selon le cas, c'est à l'étudiant de choisir, judicieusement, laquelle des deux est la plus appropriée dans le contexte rencontré. Ces fonctions doivent retourner un objet de type *list* avec deux valeurs, `x` et `statut`.

1.1 Bissection

La fonction `bissection` se base sur la méthode d'analyse numérique éponyme de recherche de racine. Celle-ci prend en arguments la fonction $f(x)$ elle-même, deux points d'abscisses différents et la tolérance d'erreur maximale admissible par l'algorithme. Afin d'assurer le bon fonctionnement de l'algorithme, il faut vérifier les points suivants :

i. Hypothèses de la méthode :
   - Les images des deux points initiaux, `x0` et `x1`, doivent être de signes contraires. Ceci peut être vérifié en testant le signe de l'expression $f(x_1)f(x_0)$. Ainsi, viennent deux cas possibles :
     - Cas 1, $f(x_1)f(x_0) < 0$:

       Les deux images sont de signes contraires, l'hypothèse est donc vérifiée.
     - Cas 2, $f(x_1)f(x_0) > 0$:

       Les deux images sont de même signe, la méthode de la bissection n'est pas possible sur l'intervalle considéré. On retourne `statut = 1`.
   - La fonction $f(x)$ doit être continue dans l'intervalle $[x_0, x_1]$, soit $f \in C_0([x_0, x1])$. Malheureusement, aucune fonction des bibliothèques autorisées ne permet de vérifier ce caractère. Supposons pour le projet que la fonction $f(x)$ dont on cherche la racine est bien continue sur l'intervalle. Nous verrons par la suite que cette hypothèse est acceptable car nous étudions un phénomène physique d'évolution de température dans un espace, en trois dimensions, continu, de sorte que la fonction est bien continue sur l'intervalle et l'hypothèse est vérifiée.

ii. Nombre d'appels de la fonction $f(x)$:
   - Une attention particulière a été portée à ce sujet. En effet, l'algorithme n'appelle qu'une seule fois $f(x)$ par itération. Ce qui est logique car la méthode de la bissection consiste à calculer un nouveau point pour chaque nouvelle itération, en se rapprochant, pas à pas, de la solution exacte.

1.2 Sécante

La fonction `secante` se base sur la méthode d'analyse numérique éponyme de recherche de racine. Son but est le même que la fonction précédente mais possède son lot de différences auxquelles le programmeur doit être rigoureux. Tout comme `bissection`, celle-ci prend en arguments la fonction $f(x)$ elle-même, deux points d'abscisses différents et la tolérance d'erreur maximale admissible par l'algorithme. Afin d'assurer le bon fonctionnement de l'algorithme, il vient de vérifier les points suivants :

i. Hypothèses de la méthode :
   — Celles-ci sont exactement les mêmes que pour la bissection. On procède de manière analogue à précédemment.

ii. Nombre d'appels de la fonction $f(x)$:
   — Idem bissection

1.3 Comparaison des deux fonctions

Le tableau 1 met en évidence les différences entre les deux fonctions `bissection` et `secante`. Notons que pour les deux fonctions, il est possible de déterminer une approximation du nombre d'itérations maximales en fonction de la tolérance `tol` et des points initiaux `x0`, `x1`. Ceci peut être donné avec la formule

$$k_{\max} = \log_2 \left(\frac{|x_1 - x_0|}{2 \cdot \text{tol}} \right). \tag{1}$$

De plus, en termes de vitesse, la fonction `secante` s'en sort beaucoup mieux que `bissection`. Pour comparer les deux, une fonction test $f_{test}(x) = x^3 - 2x - 5$ a été fournie aux deux fonctions avec les mêmes points initiaux $(x_0, x_1) = (-5, 10)$ et la tolérance 10^{-6}. Les résultats étaient similaires, les deux méthodes ont convergé mais `secante` prend quasiment deux fois moins de temps à converger que `bissection`. Selon le `profiler` de Spyder, la première met 45 µs et la seconde 90 µs en moyenne pour trouver la racine $x = 2$.

2 Pertes et gains de chaleur

2.1 Importer `PerteEtGain.txt`

Pour importer et interpréter des données numériques stockées dans un fichier `.txt` dans un module Python, on peut utiliser la fonction `loadtxt` de la bibliothèque `numpy`. Cela permet de placer les données du `.txt` dans un objet de type *Array of float64* de taille [2,25] avec la première ligne qui contient les temps de 0 à 24 heures et la seconde, les flux de chaleur correspondants.

Critère	Bissection	Sécante
Vitesse	Lente, surtout pour des intervalles larges : 48 µs.	Plus rapide que la bissection : 32 µs
Ordre de convergence	Linéaire : 1	Superlinéaire : $\varphi \approx 1.618$
Robustesse	Très robuste : la méthode converge toujours si les conditions sont remplies.	Moins robuste : peut échouer si les points initiaux sont mal choisis ou si la fonction n'est pas bien définie autour des points initiaux.
Exigence des conditions initiales	Nécessite seulement deux bornes avec un signe opposé.	Nécessite deux points initiaux distincts et un bon choix pour garantir la convergence.

Tableau 1 – Comparaison des fonctions `bissection` et `secante`

2.2 Interpoler $G(t)$

L'interpolation de $G(t)$ se fait en discrétisant l'intervalle de temps en un grand nombre de valeurs entre 0 et 24 heures. En l'occurrence, un nombre de 400 points a été choisi afin de garantir une bonne pseudo-continuité. Ce nombre est assez grand pour la suite du projet sans consommer trop de mémoire. Ensuite, il faut appeler la fonction `CubicSpline` de la sous-bibliothèque `interpolate` de `scipy`. Ce choix est motivé par les variations du flux au cours de la journée. En effet, $G(t)$ est décroissant jusque 10 heures, croissant de 10 à 20 heures et de nouveau décroissant jusque minuit. Ce comportement se prête bien à une fonction cubique d'où le choix de `CubicSpline`.

2.3 Vérification graphique

Afin de vérifier que la fonction $G(t)$ a bien été interpolée, traçons l'allure de $G(t)$ sur un graphe à l'aide de la sous-bibliothèque `pyplot` de `matplotlib`. Celle-ci nous retourne la figure 1. Nous remarquons que la fonction a bien été interpolée car elle coïncide avec tous les points expérimentaux fournis dans `PerteEtGain.txt`.

3 Mise en place de la modélisation

3.1 Encodage de `odefunction`

L'énoncé impose qu'`odefunction` reçoive en arguments les températures en $°C$, il faut donc incrémenter chaque température de $273,15°C$ pour passer de degré Celsius à Kelvin, ce qui est logique car les équations, traitées par la suite, sont exprimées en Kelvin. La fonction consiste à

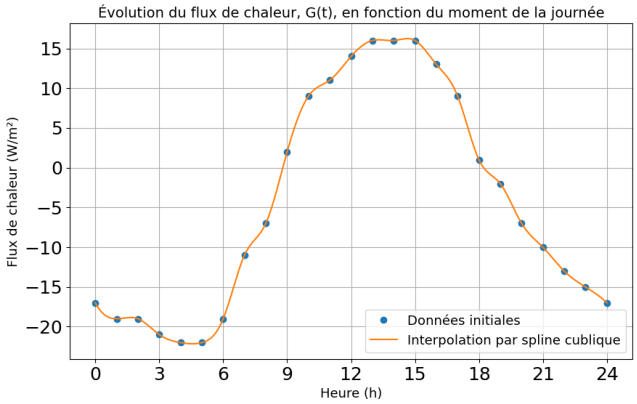


FIGURE 1 – Allure de la fonction $G(t)$

définir le système d'équations différentielles ordinaires

$$C_{cc}\frac{dT_{cc}}{dt} = -\frac{1}{R_{cc-c1}}(T_{cc} - T_{c1}) - \frac{1}{R_x}(T_{cc} - T_t) + \frac{1}{R_{c2-cc}}(T_{c2} - T_{cc}) \tag{2}$$

$$C_{c1}\frac{dT_{c1}}{dt} = -\frac{1}{R_{cc-c1}}(T_{c1} - T_{cc}) \tag{3}$$

$$C_{c2}\frac{dT_{c2}}{dt} = -\frac{1}{R_{c2-cc}}(T_{c2} - T_{cc}) + \frac{1}{R_{r-s} + R_{s-c2}}(T_{room} - T_{c2}) \tag{4}$$

$$C_{room}\frac{dT_{room}}{dt} = -\frac{1}{R_{r-s} + R_{s-c2}}(T_{room} - T_{c2}) + G(t) \tag{5}$$

$$C_w\frac{dT_t}{dt} = -\frac{1}{R_x}(T_t - T_{cc}) - \frac{1}{R_w}(T_t - T_w) \tag{6}$$

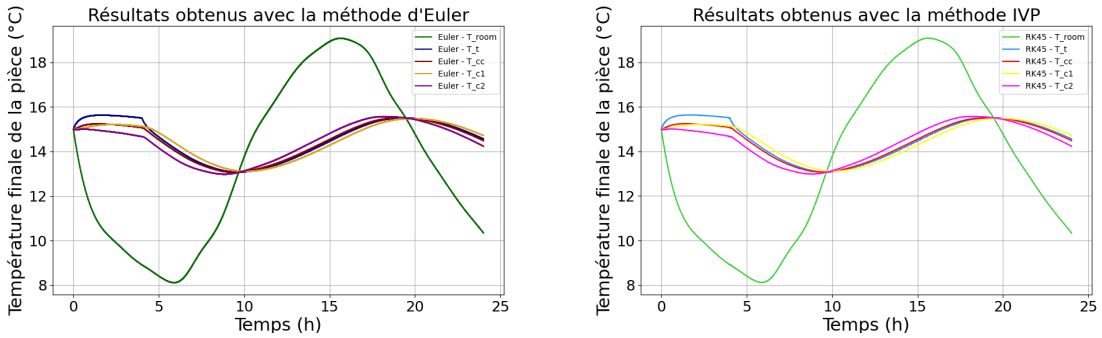
où T, R et C représentent respectivement la température (K), la résistance thermique (mK/W) et la capacité thermique spécifique (kJ/mK).

Il faut ensuite traiter les différentes valeurs que peut prendre la température de l'eau dans les tubes, T_w. Il faut modifier sa valeur en fonction de la valeur courante du temps `t`. Dans le cas du scénario 1, on a

— Si `t` $\in [0, 4]$: $T_w = 18°C = 291,15K$

— Si `t` $\in]4, 24]$: Le dernier terme de l'équation 6 peut être rejeté.

Les dérivées des températures à retourner doivent être multipliées par 3600 afin de passer de K/s à K/h.



(a) Méthode d'Euler **(b)** Méthode de Runge-Kutta 45

FIGURE 2 – Comparaison des températures entre Euler et Runge-Kutta 45

3.2 Méthode d'Euler

La méthode d'Euler explicite est une méthode numérique simple d'approximation des solutions d'équations différentielles ordinaires. L'approximation repose sur le processus itératif suivant :

$$T_{i+1} = T_i + h \cdot f(t_i, T_i) \tag{7}$$

où $f(t_i, T_i)$ représente `odefunction(t[i], T[i])` et h est le pas. Les paramètres de la fonction sont

- `[t0,tf]` : l'intervalle de temps sur lequel résoudre le système d'équations différentielles ordinaires,
- `T0` : les valeurs de température initiales, toutes égales à $15°C$,
- `h` : le pas, représentant l'intervalle entre deux points consécutifs dans la discrétisation du temps.

3.3 Méthode Runge-Kutta d'ordre 4/5

La méthode de Runge-Kutta d'ordre 4/5, aussi appelée RK45, utilisée par la fonction `solve_ivp` de la sous-bibliothèque `integrate` de `scipy`, permet une plus grande précision qu'Euler car elle ajuste dynamiquement le pas pour optimiser la précision de la solution. Les paramètres de la fonction sont

- `[t0,tf]` : l'intervalle de temps sur lequel résoudre le système d'EDO,
- `T0` : les valeurs de température initiales, toutes égales à $15°C$,
- `rtol` : la tolérance relative du solveur, `solve_ivp`.

Les deux méthodes sont valables et aboutissent, en principe, aux mêmes résultats numériques, comme en témoigne la figure 2.

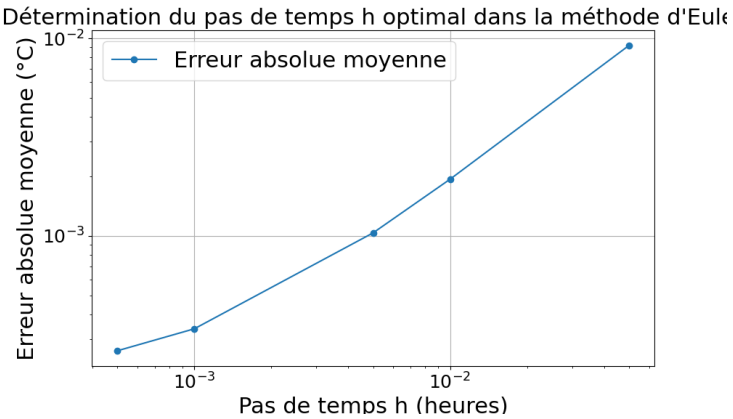


FIGURE 3 – Détermination de l'erreur moyenne de la méthode d'Euler

3.4 Convergence d'Euler en fonction de h

Une analyse de l'erreur en fonction du pas de temps h a été réalisée en comparant la solution obtenue avec Euler à une solution de référence avec `solve_ivp`. Pour ce faire, la solution de référence a été interpolée sur les pas d'Euler. Ensuite, l'erreur absolue moyenne pour chaque pas candidat h_{test} a été évaluée comme la moyenne de $\left|T_{Euler} - T_{réf}\right|$. Celle-ci peut être représentée sur la figure 3. Remarquons que l'erreur diminue lorsque h diminue, ce qui est cohérent car si le pas diminue, la résolution est d'autant plus précise. De plus, la fonction tracée est linéaire lorsqu'on trace le graphe sur une échelle logarithmique. Cela s'explique par le fait que la méthode d'Euler possède un ordre de convergence égal à 1, ce qui implique une erreur en $O(h)$. Il vient alors de faire un compromis et choisir $h = 10^{-2}$ qui est plus précis que $h = 10^{-1}$ et plus rapide que $h = 10^{-3}$.

3.5 État stationnaire

Pour calculer l'évolution des températures jusqu'à atteindre le critère d'arrêt, signe d'état stationnaire,

$$T_{room}(t_f) - T_{room}(t_f - 24) < 0.01°C \tag{8}$$

il suffit de résoudre le système d'équations différentielles ordinaires pour 24 heures, en boucle, en remplaçant les températures initiales par les températures finales calculées après chaque fin de journée, jusqu'à satisfaire le critère d'arrêt et donc atteindre l'état stationnaire du système. Le graphe traçant l'évolution des températures de la pièce et de la partie inférieure du béton en fonction du temps jusqu'à l'état stationnaire est illustré à la figure 4. Remarquons que la température de la pièce converge vers une valeur beaucoup plus faible que celle du béton. Ceci s'explique par l'influence des flux de chaleur $G(t)$, baissant la température de la pièce tout au long de la journée.

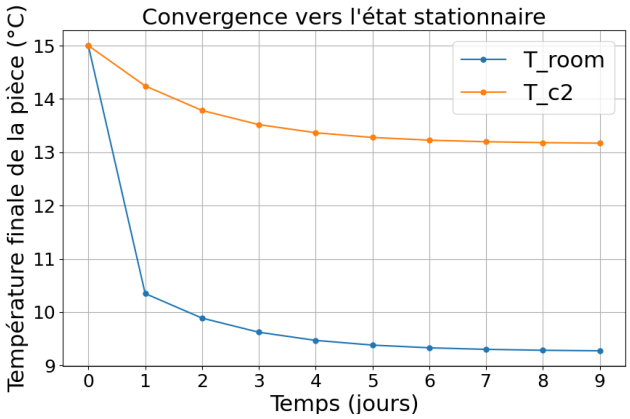


FIGURE 4 – Convergence vers l'état stationnaire des températures de la pièce et de la partie inférieure du béton.

3.6 Comparer les scénarios

Pour les calculs il suffit de reprendre la démarche de la question 3.5, il faut adapter `odefunction` aux scénarios 2 et 3.

- Scénario 2 :
  - Si $t \in [0, 4]$: $T_w = 18°C = 291,15K$,
  - Si $t \in]4, 13]$: $T_w = 28°C = 301,15K$,
  - Si $t \in]13, 24]$: Le dernier terme de l'équation 6 peut être rejeté.
- Scénario 3 :
  - Si $t \in [0, 12]$: $T_w = 28°C = 301,15K$,
  - Si $t \in]12, 24]$: $T_w = 18°C = 291,15K$.

On obtient ainsi les résultats, pour chaque température, illustrés sur les figures 5,
Comme souligné précédemment, la température de la pièce est la seule à présenter une différence notable dans sa variation temporelle comparée à toutes les autres.

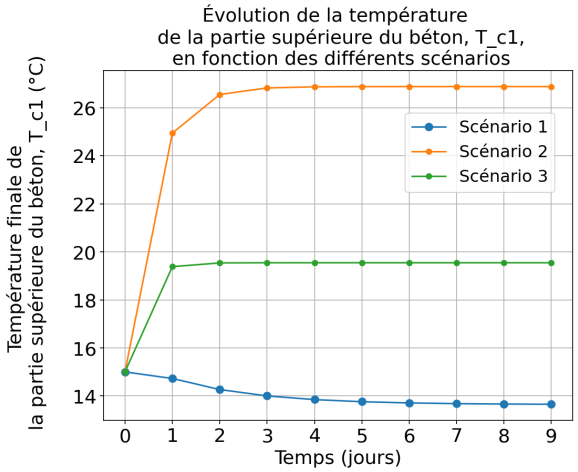
4 Régulation automatisée du bâtiment

4.1 Détermination de T_{max}

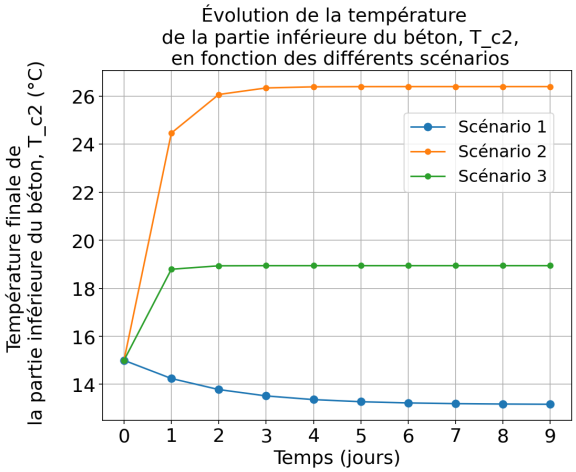
Il suffit d'adapter `odefunction` en fonction de Δt et à l'aide de la fonction `max` et `argmax` de `numpy`, il est possible de trouver le point culminant de la température de confort, T_{max}, défini selon

$$T_{max} = \max\left(T_{confort}(t, \Delta t)\right) \tag{9}$$

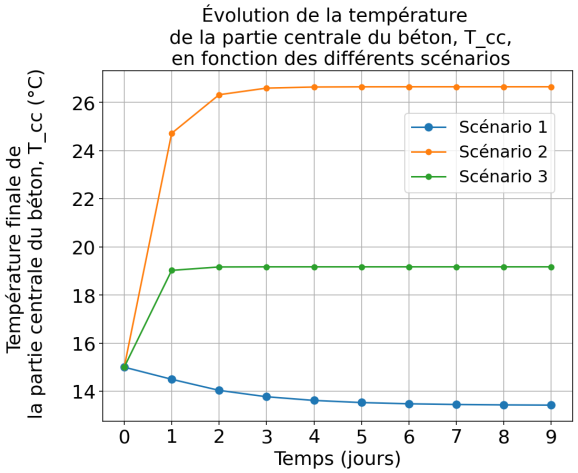
et d'en imposer une valeur maximale. Cette fonction a permis le dessin de la figure 6 qui trouve la température de confort maximale pour un $\Delta t = 1$ fixé arbitrairement.



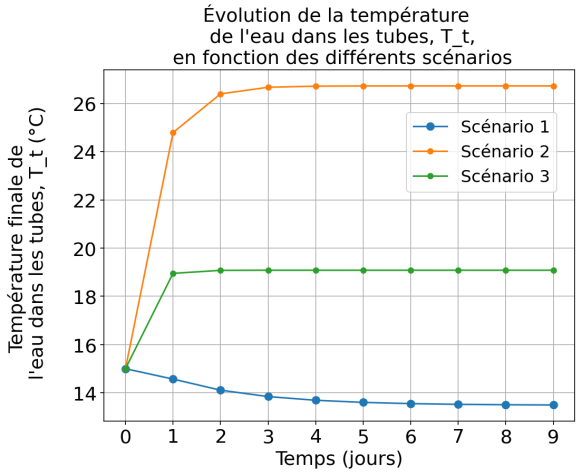
(a) Partie supérieure du béton



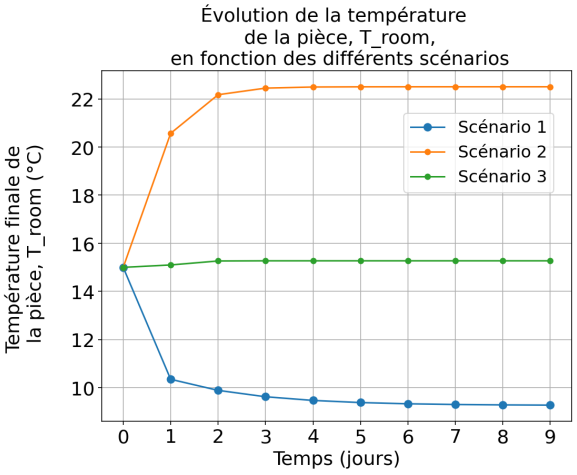
(b) Partie inférieure du béton



(c) Partie centrale du béton



(d) Tubes



(e) Pièce

FIGURE 5 – Évolution des températures suivant les différents scénarios

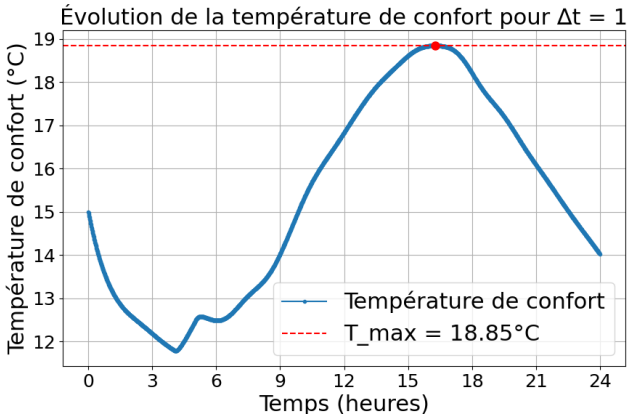


FIGURE 6 – Détermination de la température max lors des premières 24h avec $\Delta t = 1$

4.2 $\Delta t_{optimal}$ pour 24 heures

Par définition de $\Delta t_{optimal}$, on a

$$\Delta t_{optimal} = \arg\max_{\Delta t} \left(T_{confort}(t, \Delta t) \right) \tag{10}$$

de sorte que

$$T^d_{max} = T_{max}(\Delta t_{optimal}) = \max_{\Delta t} \left(T_{max}(\Delta t) \right) \tag{11}$$

La recherche revient donc à trouver la racine de la fonction $f(\Delta t)$ définie comme

$$f(\Delta t) = T_{max}(\Delta t) - T^d_{max} \tag{12}$$

La méthode de la sécante semble adéquate pour cette tâche car elle converge rapidement et fournit un résultat plus précis que la bissection. Il vient ensuite de choisir l'intervalle compris entre 0.1 et 8 heures. La vérification de cet intervalle repose sur le théorème des valeurs intermédiaires. En effet, La condition $f(0.1) \cdot f(8.0) < 0$ garantit l'existence d'une racine dans l'intervalle choisi, confirmant ainsi la pertinence de ces bornes. La détermination de ce $\Delta t_{optimal}$ est représentée à la figure 7.

4.3 $\Delta t_{optimal}$ à l'état stationnaire

Il faut refaire la même chose qu'à la question 4.2 mais lorsque le système est à l'état stationnaire. Dès lors, il suffit de réutiliser l'implémentation de la question 3.5. Il se trouve que celui-ci se stabilise après 8 jours, comme le montre la figure 8. Il faut ensuite vérifier que la pièce respecte bel et bien la norme EN15251 imposant de garder une température comprise entre 19.5 et 24°C pendant les heures de bureaux, comprises entre 8 et 19 heures. En traçant l'évolution de $T_{confort}$ le dernier jour, nous remarquons, à la figure 9 que la norme n'est pas respectée tout au début de la journée.

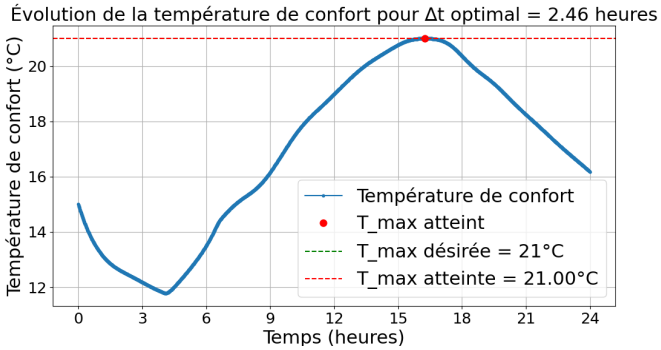
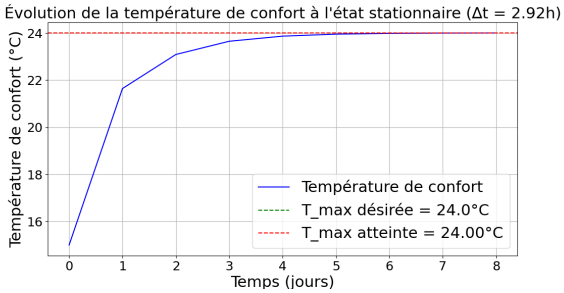
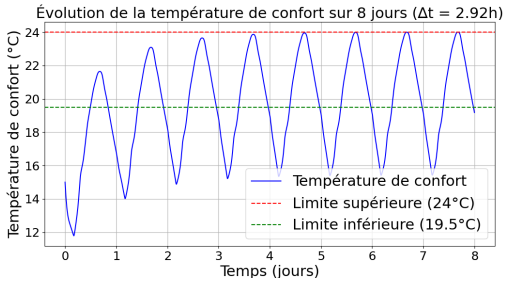


FIGURE 7 – Évolution de la température de confort avec $\Delta t_{optimal} = 2.46h$ et $T^d_{max} = 21°C$



(a) Évolution de T_{max} jusqu'à T^d_{max}



(b) Évolution de $T_{confort}$ jusqu'à l'état stationnaire

FIGURE 8 – Convergence du système vers l'état stationnaire

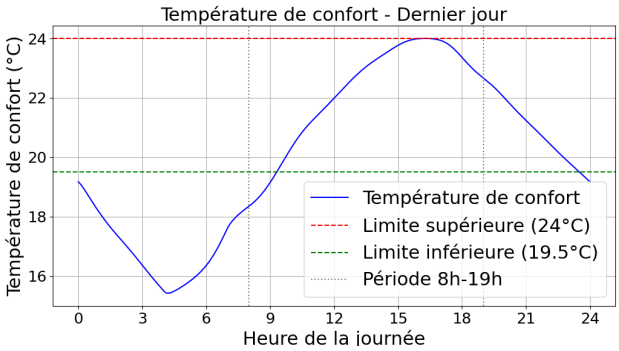


FIGURE 9 – Évolution de $T_{confort}$ à l'état stationnaire