Rapport PROJ0001

Pierre Sluse (s2404824)

Louis MEAN (s2403169)

Joel WANDEGE (s2406718)

Prof. Olivier Bruls, Prof. Quentin Louveaux, Prof. Frédéric Nguyen

Méthodes numériques

06 avril 2025

Table des matières

1 Question 1 - Recherche de racines

1.1 Méthode de la bissection

La méthode de la bissection, aussi appelée méthode de la dichotomie, repose sur le principe du changement de signe d'une fonction continue pour localiser une racine. Elle est applicable lorsqu'une fonction $f(x)$ est continue sur un intervalle $[x_0, x_1]$ et que $f(x_0)$ et $f(x_1)$ sont de signes opposés.

1.2 Étapes de la méthode

1. Vérifier que $f(x_0)$ et $f(x_1)$ sont de **signes opposés** et qu'elles sont bien **définies**. Si ce n'est pas le cas, il se peut que la fonction ne possède pas de racine **unique** dans l'intervalle, ou alors que la racine soit une **extrémité** de l'intervalle et le message adéquat est retourné.
2. Calculer le point **médian** $x_2 = \frac{x_0+x_1}{2}$.
3. Évaluer la fonction en ce point :
   — Sinon, si $f(x_0) \cdot f(x_2) < 0$, alors la racine se trouve dans $[x_0, x_2]$, donc on met à jour $x_1 = x_2$.
   — Sinon, la racine se trouve dans $[x_2, x_1]$ et on met à jour $x_0 = x_2$.
4. Répéter jusqu'à atteindre une précision conforme à la tolérance ou le nombre maximal d'itérations.

Le **nombre d'itérations** nécessaire est donné par la formule suivante (arrondie par le bas) :

$$\boldsymbol{N} = \log_2 \left(\frac{\boldsymbol{x_1 - x_0}}{2 \cdot \textbf{tol}} \right) + 1 \tag{1}$$

qui est une réécriture de l'originale fournie dans le cours Si N dépasse un certain seuil (ici fixé à 50), la méthode est stoppée pour éviter une boucle infinie. La tolérance choisie en appelant la fonction est par défaut égale à $\mathbf{0.5 \times 10^{-7}}$ pour assurer une bonne précision.

Nos deux fonctions vérifient également si elles **convergent** ($|f(x_2)| < \text{tol}$) mais ne prennent pas en compte des **racines doubles**.

1.3 Méthode de la sécante

La méthode de la sécante est une alternative de la méthode de la bissection, elle converge plus rapidement mais moins systématiquement (sensible aux points d'inflexion). Elle consiste à tracer une droite entre deux points et d'y chercher l'intersection entre les abscisses.

1.4 Étapes de la méthode

1. Utiliser HasRoots de manière analogue à la bissection sans regarder le signe de la fonction.

2. Calculer une nouvelle approximation de la racine en utilisant la relation et évitant de diviser par le zéro machine :

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} \tag{2}$$

3. Vérifier si la différence $|x_{n+1} - x_n|$ est inférieure à la tolérance. Si oui, la racine est trouvée.
4. Répéter jusqu'à convergence ou atteinte du nombre maximal d'itérations.

2 Question 2 - Fonction G(t) pertes et gains de chaleur

2.1 Importation des données

Ces données sont fournies dans un fichier texte `PerteEtGain.txt` et sont exprimées en W/m^2 sur une période de 24 heures. Elles sont extraites du fichier en utilisant la fonction `numpy.loadtxt()`, qui permet de lire directement les valeurs sous forme d'un tableau (array) de 2 lignes et 25 colonnes.

2.2 Interpolation des données

Nous allons interpoler ces données afin d'obtenir une fonction continue manipulable. Nous avons choisi celle par *spline*, plus précisément `cubicSpline` de `scipy.interpolate`. L'interpolation est faite avec le paramètre *'periodic'* étant donné que nous étudions un cycle récurrent. La fin d'une journée est en effet le début de la suivante.

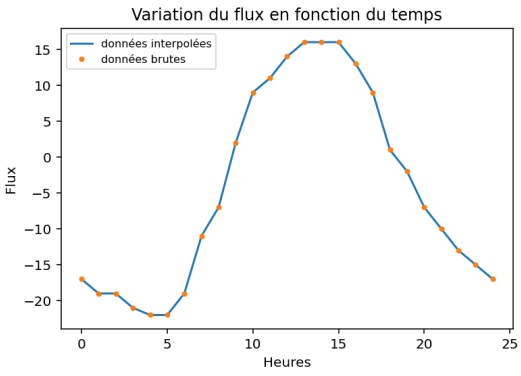

FIGURE 1 – Perte et gain de chaleur

3 Question 3 - Mise en place de la modélisation

3.1 Equations différentielles

Définition de la fonction d'évolution des températures

Afin de modéliser l'évolution des températures dans les différentes couches du bâtiment, nous avons défini une fonction `odefunction` qui retourne les dérivées des températures dT/dt en fonction des équations différentielles régissant le système. Cette fonction prend en entrée :

— Le temps t (en secondes).
— Un tableau T contenant les températures actuelles en degrés Celsius :

$$T = \left[T_{\text{room}}, T_t, T_{cc}, T_{c1}, T_{c2}\right]^T$$

Ce tableau peut être en 1D ou 2D, grâce à la flexibilité de codage de python et de la manière dont nous récupérons les valeurs. C'est pour cela que malgré que l'énoncé demande une matrice de dimensions [5,1], nous pouvons traiter cette matrice comme de dimensions [5], ce qui accélère un peu plus les calculs. Elle retourne un tableau contenant les cinq dérivées dT/dt en K/s.

3.2 Méthode d'Euler et choix du pas

Méthode d'Euler pour le calcul des températures

La résolution des équations différentielles décrivant l'évolution des températures est effectuée à l'aide de la méthode d'Euler explicite. Cette méthode numérique permet d'approcher la solution d'un système d'équations différentielles sous la forme :

$$\frac{dT}{dt} = f(t, T) \tag{3}$$

Principe de la méthode d'Euler

La méthode d'Euler repose sur une discrétisation du temps en instants t_i espacés d'un pas h. À chaque instant t_i, la température T est mise à jour selon :

$$T_{i+1} = T_i + h \cdot \frac{dT}{dt}(t_i, T_i) \tag{4}$$

où $\frac{dT}{dt}$ est calculé à partir du modèle thermique donné par la fonction `odefunction`.

Choix du pas h

Nous avons choisi notre pas empiriquement en testant différents pas. Nous en sommes venus à la conclusion que $h = 0.01$ était le plus optimal, alliant à la fois précision et rapidité. Nous préférons des puissances de 10, car la division d'une journée en un nombre de pas irrationnel (exemple : $333,3333\overline{3}$) pourrait être problématique.

On remarque au passage que la division par un facteur 10 du pas d'Euler (passage de $h = 0.01$ à $h = 0.001$ par exemple) ne fait qu'augmenter la précision par un facteur 10. Nous estimons qu'une précision à $0.008°C$ est largement suffisante pour ce problème. Nous choisissons donc un pas de $h = 0.01$.

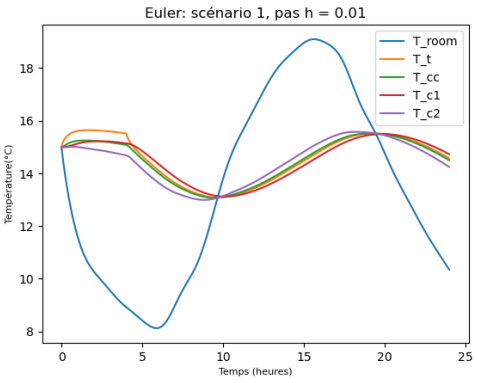

FIGURE 2 – Euler - Pas = 0.01

3.3 Etude de scénarios de chauffage et refroidissement

Résolution avec `solve_ivp`

La fonction `calculTemperaturesIVP` utilise l'implémentation de `solve_ivp` de `scipy`, telle que présentée dans le tutoriel, pour résoudre le système d'équations différentielles. Cette méthode permet une résolution adaptative en contrôlant l'erreur numérique avec une tolérance relative par défaut de 10^{-7}.
Son graphique est, sans surprise, extrêmement similaire à celui de la résolution par Euler (cf. Fig. 3).

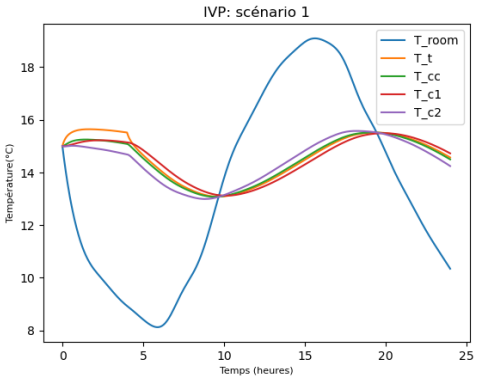

FIGURE 3 –

3.4 Observation de la convergence du calcul des dérivées par la méthode de Runge-Kutta et par la méthode d'Euler

Ici, nous avions deux méthodes pour observer la convergence de l'approche par Euler avec l'approche par Runge-Kutta (qui est la méthode employée par `solve_ivp`).

La première, celle que nous avons employée, consistait à calculer avec une précision demandée de 10^{-10}, puis à rajouter un argument permettant d'évaluer la fonction à d'autres points supplémentaires (le paramètre `t_eval`) afin de comparer les deux méthodes à des temps égaux.

En effet, il ne serait pas pertinent de comparer les températures à des temps différents puisque, contrairement à Euler, le calcul des points de résolution ne se fait pas à des temps équidistants.

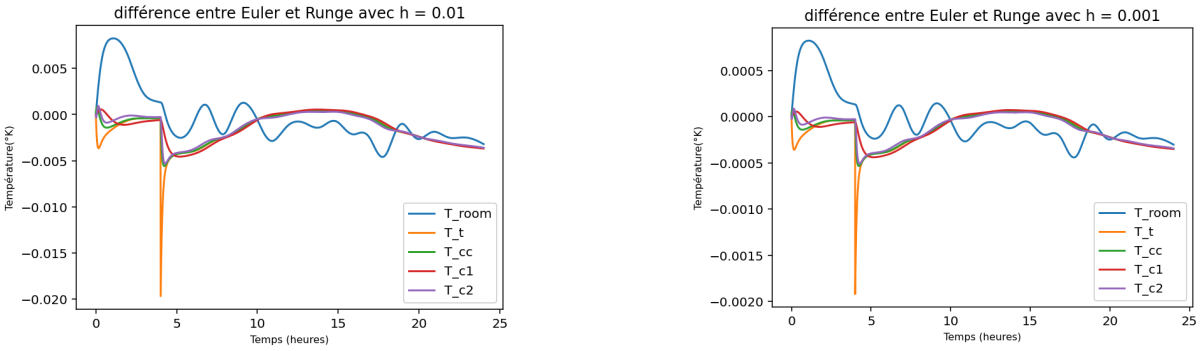

FIGURE 4 – Comparaison des différences de températures pour différents pas de temps.

On observe par ailleurs qu'un pas 10 fois plus petit entraîne une précision de convergence 10 fois plus grande, ce qui avait été observé plus tôt lors du choix du pas d'Euler.

Il aurait été possible de calculer la fonction par Runge-Kutta, puis interpoler les points pour en faire une fonction continue et l'évaluer à ces points-là. Cependant, l'incertitude sur

la précision d'une interpolation, notamment à des points anguleux tels que $t = 4h$, lorsque la dérivée de la température de la partie centrale en béton change de dernier terme radicalement, pose un problème. Il est d'ailleurs bon de remarquer que ces points anguleux restent toujours visibles avec l'approche par Euler. En effet, cette méthode, certes simple, demeure sensible à ce type de point.

3.5 Recherche d'un état stationnaire du système

Ici, nous observons la stabilisation des températures du scénario numéro 1. Pour cela, l'algorithme par calculer le premier jour et puis vérifie si la stabilisation est déjà atteinte, et si ce n'est pas le cas, continuer à ajouter des jours et revérifier jusqu'à ce qu'une stabilisation soit atteinte avec une tolérance qui peut aussi bien être définie dans les arguments de la fonction que par défaut.

Ainsi nous évitons de calculer plus de jours que nécessaire. Il y a une limite de jours pour atteindre la stabilisation pour éviter que le système ne se stabilise jamais en cas d'erreur de manipulation (définie par défaut à 30 jours, ajustable). La figure 5 montre bien la stabilisation des températures sur une période de 9 jours.

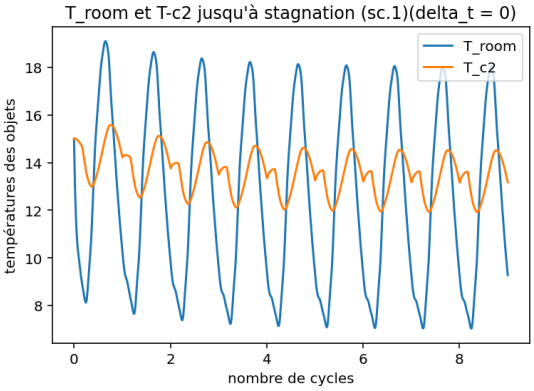

FIGURE 5 – Température de la pièce et de la partie inférieure en béton au scénario 1.

3.6 Comparaison de différent cycles de chauffe.

Utilisant l'exacte même fonction que dans la question 3.5, il est facile de changer le numéro du scénario et de tracer les graphiques des températures pour les scénarios 2 et 3.

On observe que plus le temps de chauffe dans une journée est important (4 heures de refroidissement pour le premier scénario, fig. 6, 4 heures de refroidissement et 12 heures de chauffe pour le second, fig. 7, et 12 heures de chauffe pour le troisième, fig. 6), plus la stabilisation est atteinte rapidement.

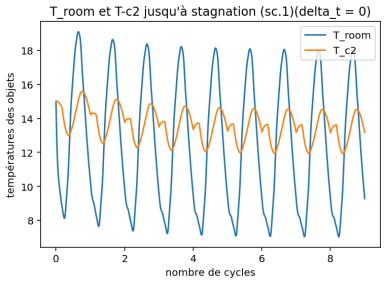

FIGURE 6 – Scénario 1

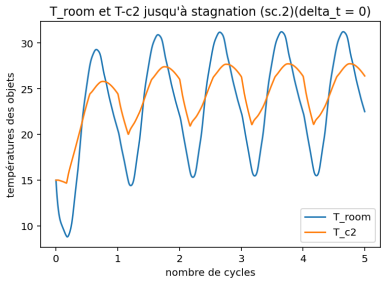

FIGURE 7 – Scénario 2

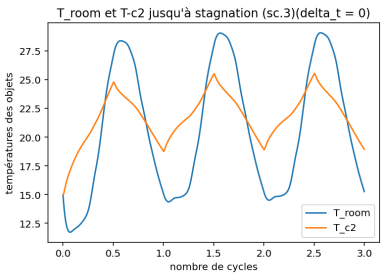

FIGURE 8 – Scénario 3

4 Question 4 - Régulation automatisée – calcul d'un temps de chauffe pour atteindre un critère de performance thermique

4.1 Déterminer le maximum de température sur un cycle de 24h

On cherche à définir une fonction qui décrit l'évolution de la température sur un cycle de 24 heures suivant le scénario 4 avec $\Delta t = 0$ pour l'instant.

Méthode utilisée :

1. Calcul de la température de confort à partir des températures *T_room* et *T_c2* que l'on obtient avec la fonction `calculTemperaturesEuler`.
2. Trouver le max de la température de confort sur un cycle de 24h avec la fonction `T_max`.
3. Faire un graphique pour montrer comment la température de confort évolue.

D'après notre fonction `T_max`, la température maximale est de 17.123°C et est atteinte après 58 572 secondes, c'est-à-dire un peu moins de 16h20'.

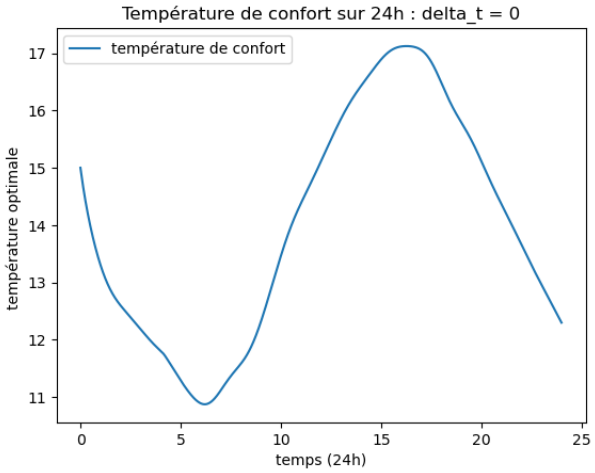

FIGURE 9 – évolution des températures d'une journée - Scénario 1, $\Delta t = 0$.

4.2 Déterminer un Δt pour ne pas dépasser un $T_d^{\max}$ donné

On cherche à définir une fonction qui recherche le temps de chauffe supplémentaire Δt pour atteindre $T_d^{\max}$ sur le premier cycle de 24 heures. Pour cela, nous employons la méthode suivante :

1. Utilisation de la fonction `recherche_delta_t` pour trouver la période Δt qu'il faut pour que $T_{\max}$ atteigne une valeur donnée $T_d^{\max}$.
2. Utilisation d'une méthode de recherche de racine (bissection) avec une tolérance assez faible, car elle ne peut pas dépasser la valeur du pas (0.01 en l'occurrence, ce qui signifie que la racine est trouvée avec une précision de 0.01 près). Cette bissection sert donc à résoudre l'équation $T_{\max}(\Delta t) - T_d^{\max} = 0$.
3. Génération d'un graphique de la température de confort pour le Δt trouvé à l'aide de la fonction `question_4_1`.

La figure 10 montre une situation exemple avec $T_d^{\max} = 24°C$.

La durée optimale de chauffe pour satisfaire cette condition de 4,883 heures avec une tolérance de soit 17 578 secondes.

Une durée de chauffe trop courte empêche d'atteindre la température cible $T_d^{\max}$ et une durée trop longue entraîne une surchauffe du système au-delà. On constate que peu importe le Δt déterminé, la température maximale est systématiquement atteinte au même moment de la journée avec des variations de moins d'une minute avec une précision inférieure à 36 secondes en raison du pas utilisé valant 0.01 (excepté lorsque le maximum se trouve dans les conditions initiales, mais c'est un cas limite).

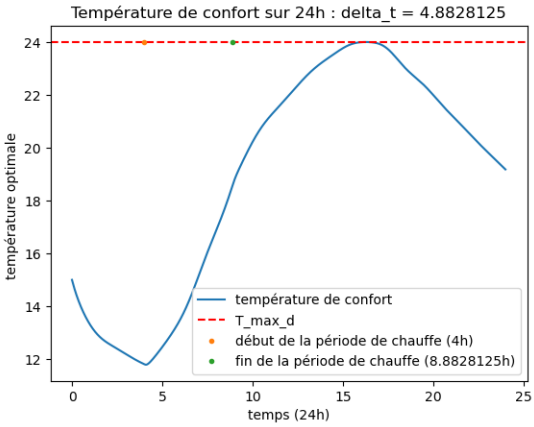

FIGURE 10 – Température de confort sur un cycle de 24h.

4.3 Déterminer un Δt pour ne pas dépasser un $T_d^{\max}$ lorsque le système se stabilise

Dans la continuité de la question précédente, on nous demande de déterminer Δt tel que $T_d^{\max}$ est atteint après une stabilisation du cycle de température.

En réutilisant la fonction `recherche_delta_t`, on détermine $\Delta t = 2.9297$h (avec une tolérance égale au pas , soit 10 547 secondes.

On constate également avec la fonction `verification_EN15251` et avec une tolérance *tol_temp* = 0.01°Celsius que la norme `EN15251` n'est pas respectée dans ces conditions. Nous mettons une tolérance sur la température ici, car il y a une imprécision sur la racine.

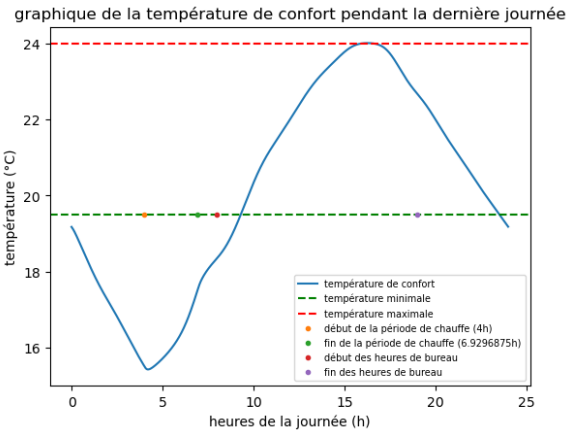

FIGURE 11 – Température de confort durant la dernière journée

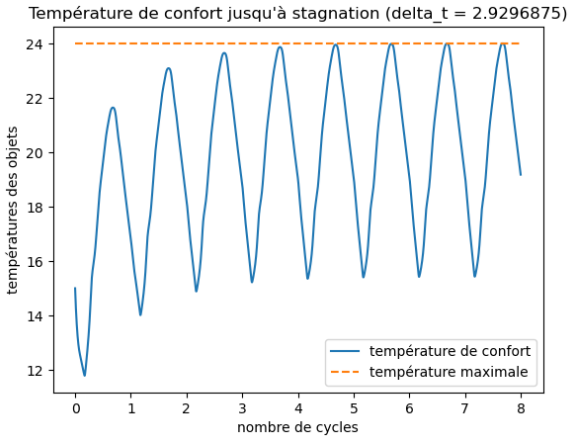

FIGURE 12 – Evolution des températures jusqu'à stabilisation.

On remarque que la température de confort met 8 jours pour se stabiliser si la température à stabilisation $T_d^{\max}$ est de 24°C (cf fig. 12).

5 Commentaires

Nous avons utilisé les ***keyword arguments*** en définissant la plupart des fonctions afin que les variables se transfèrent plus facilement entre celles-ci et pour que le code soit plus intelligible et flexible.

Un petit index des arguments est disponible dans `arguments_passables.txt` et un des fonctions et constantes dans `arguments_index.txt`.

Toutes les fonctions "`question_x_y`" ne servent qu'à interpréter les résultats obtenus par les fonctions intermédiaires.

La plupart des variables par défaut se trouvent dans le fichier `config.py`, il y a également un mode **debug** pour `print` certaines variables d'une fonction en cas de souci.

Nous avons tenté par curiosité de trouver le pas limite pour lequel la méthode de Euler finissait par diverger, et il se situe aux alentours de 0.085. Une estimation avec ce pas nous montre des courbes intéressantes qui commencent à avoir plein d'erreurs au niveau du calcul de la dérivée (cf figure 13).

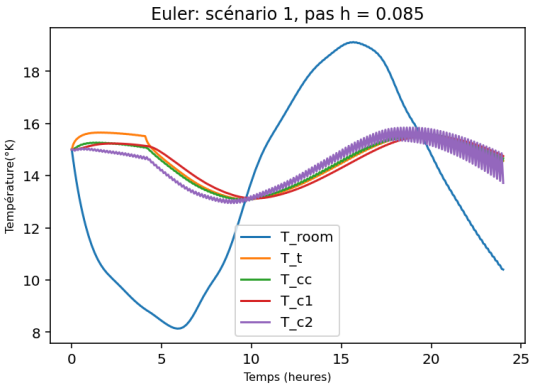

FIGURE 13 – Euler avec un h à la limite de la stabilité.

Nous avons également voulu voir comment se comportait la température à stabilisation en fonction du Δt choisi. Pour cela nous avons discrétisé la plage de 20h de Δt en 40 parties et nous avons ensuite évalué la température maximale à l'état stationnaire. Cela nous a donné un joli graphique comme montré en figure 14.

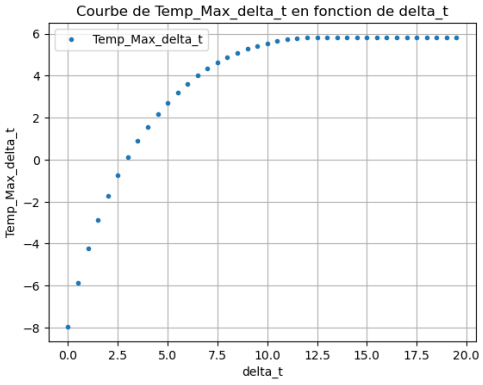

FIGURE 14 – Variation de la température de stabilisation en fontion du Δt.