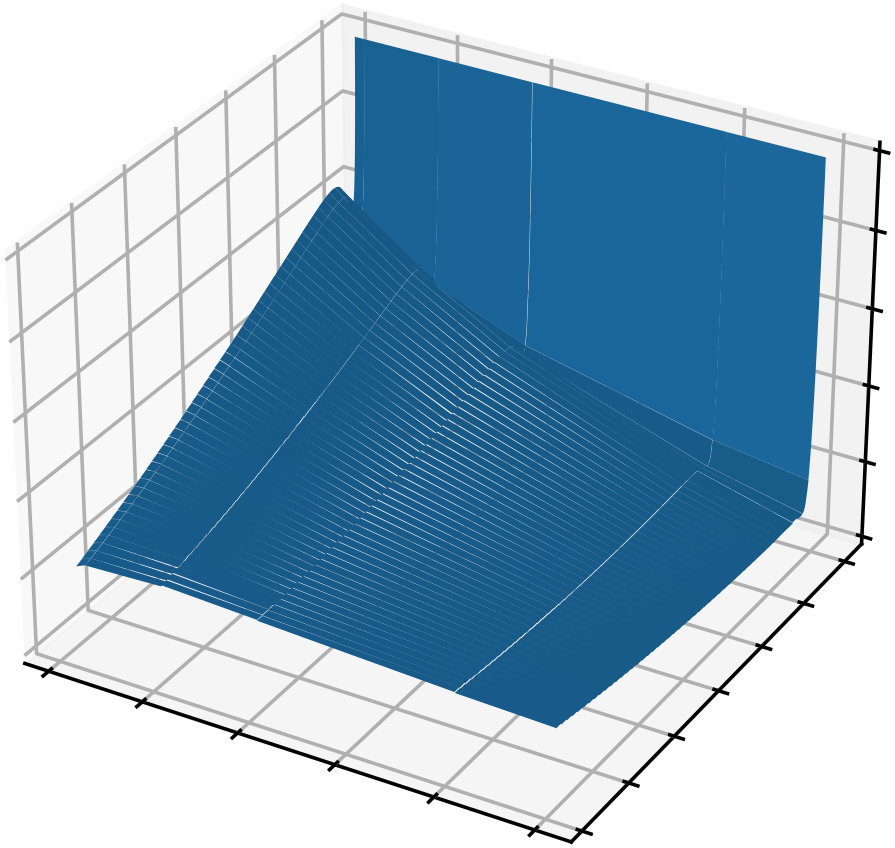
Année académique 2023-2024

Production d'H_2 par reformage catalytique de CH_4 et captage de CO_2 dans un réacteur.



LIÈGE université Sciences Appliquées

4 avril 2024

Moreci Raffaele
S2304531

Odding Luca
S2303933

Lindsey Alexandre
S2302371

Table des matières

Question 1

1.1 Méthode de la sécante

Notre fonction sécante requiert trois arguments nécessaires au bon fonctionnement de la méthode qui sont :

— fun, la fonction dont on recherche les racines,
— x, un tableau contenant les deux valeurs de x de départ
— tol, la tolérance qui permet de gérer la précision des résultats (la boucle s'arrête lorsque $abs(x0 - x1) >= tol$)

Logiquement nous devons donc compromettrefonction la précision pour limiter les calculs et donc ne pas mettre une tolérance trop faible. Nous avons opté, après une multitude d'essais, pour une valeur de 5×10^{-4} qui nous permet de garder une bonne précision pour un temps d'exécution faible.

Nous avons implémenté la fonction en s'assurant tout d'abord que toutes les conditions initiales soient bonnes et, pour que la fonction n'itère pas à l'infini, nous avons introduit max_i en arguments $kwargs$ avec une valeur par défaut élevée qui restreint la fonction à ne pas faire plus d'itérations que ce nombre tout en laissant une marge afin de ne pas arrêter trop vite l'exécution. Après avoir pris nos précautions, une boucle va itérer tant que l'évaluation de la fonction en $x1$ est plus petite ou égale à notre tol (nous utilisons la tolérance car il est quasiment impossible que la valeur soit exactement égale à 0 par les méthodes numériques).

Nous avons décidé de séparer le calcul du numérateur et du dénominateur : si le numérateur $y1 \times (x1 - x0)$ est égal à 0, $x0$ et $x1$ doivent avoir la même valeur du coup la fonction retourne -1 pour indiquer que la fonction ne converge pas. Si le dénominateur $y1 - y0$ est égal à 0, la fonction ne converge pas vu la méthode de la sécante : la droite passe par deux points qui forment une droite parallèle à l'abscisse. La boucle se finira soit par le non-respect de sa condition ou bien si la boucle fait un nombre anormalement grand d'itérations.

1.2 Méthode de la bissection

Les arguments sont les mêmes que dans la méthode de la sécante avec la même tolérance de 5×10^{-4} mais sans introduire max_i car si les conditions initiales sont respectées, la fonction convergera toujours. Nous nous assurons donc que les conditions initiales sont respectées en posant l'hypothèse que la fonctions traitée est continue.

Pour se rapprocher de la racine nous allons itérer pendant k itérations égal à $log_2(\frac{x1-x0}{2 \times tol})$ comme vu dans le cours. Dans cette boucle, on calcule la moyenne des abscisses des deux points qui sera celle de notre nouveau point. Si la valeur de la fonction en ce point est positive, on remplace $x1$ (à image positive) par cette moyenne et inversement. Après toutes les itérations, la valeur de xi sera la valeur de notre racine.

Question 2

2.1 Création d'odefunction

Notre fonction *odefunction* est une fonction qui reçois trois arguments :

— z, la distance axiale du réacteur

— $C0$, le tableau avec les valeurs initiales des huit variables d'état : les concentrations initiales en CH_4, H_2O, H_2, CO, CO_2 la conversion fractionnaire X, la température T et la pression P

— *mode* et *param*, qui nous serviront plus tard

La fonction odefunction calcule la dérivée du tableau $\frac{dC}{dz}$ et retourne ces valeurs. Pour plus de clarté et de flexibilité, les constantes ont été placées dans le fichier *constants.py*. Certaines constantes sont implémentées sous forme de fonction, cela n'est pas optimisé pour le temps mais nous avons opté pour cette option par soucis de lisibilité et de flexibilité, ce qui nous a aidé pour la question 3.

2.2 Résolution d'équation différentielle

Notre fonction *calculConcentrationEuler* est une fonction qui suit la méthode d'Euler explicite au premier ordre pour résoudre approximativement l'équation contenue dans odefunction. Cette fonction prend cinq arguments en entrée :

— fun, la fonction à résoudre

— x, un tableau contenant les bornes pour lesquels l'équation doit être résolue

— $y0$, les valeurs initiales de la fonction

— *step*, le pas qui est par défault à 5e-8

— *mode* et *param*, qui nous serviront plus tard.

La fonction renvoie une approximation de la résolution de la fonction.

La fonction *solve_ivp* prend les mêmes arguments que *calculConcentrationEuler*. Nous avons aussi modifier les valeurs de la tolérance relative *rtol* et de la taille maximum du pas *max_step* qui ont tous deux pour but d'augmenter la précision de notre approximation. Nous avons choisi notre *rtol* égale à 5×10^{-7} et notre max-step égale à 1×10^{-6}, ces valeurs sont les meilleurs compromis que nous avons trouvé entre une précision correcte et une durée d'exécution minimale.

(((Justification du pas dans *calculConcentrationEuler*)))

Après plusieurs tests, nous pouvons conclure que *solve_ivp* est beaucoup plus efficace que notre fonction. Notre fonction prend environs (((x minutes))) pour résoudre l'équation tandis que *solve_ivp* ne prend que (((x secondes))), *solve_ivp* va donc (((x))) fois plus vite. Ceci est probablement dû au fait que *solve_ivp* utilise un pas variable mais pas notre fonction. *solve_ivp* est également beaucoup plus précis puisqu'elle utilise une méthode d'ordre suprérieur. Nous concluons que *solve_ivp* est beaucoup plus efficace, que ce soit pour le temps d'exécution que pour la précision.

Question 3

Question 4