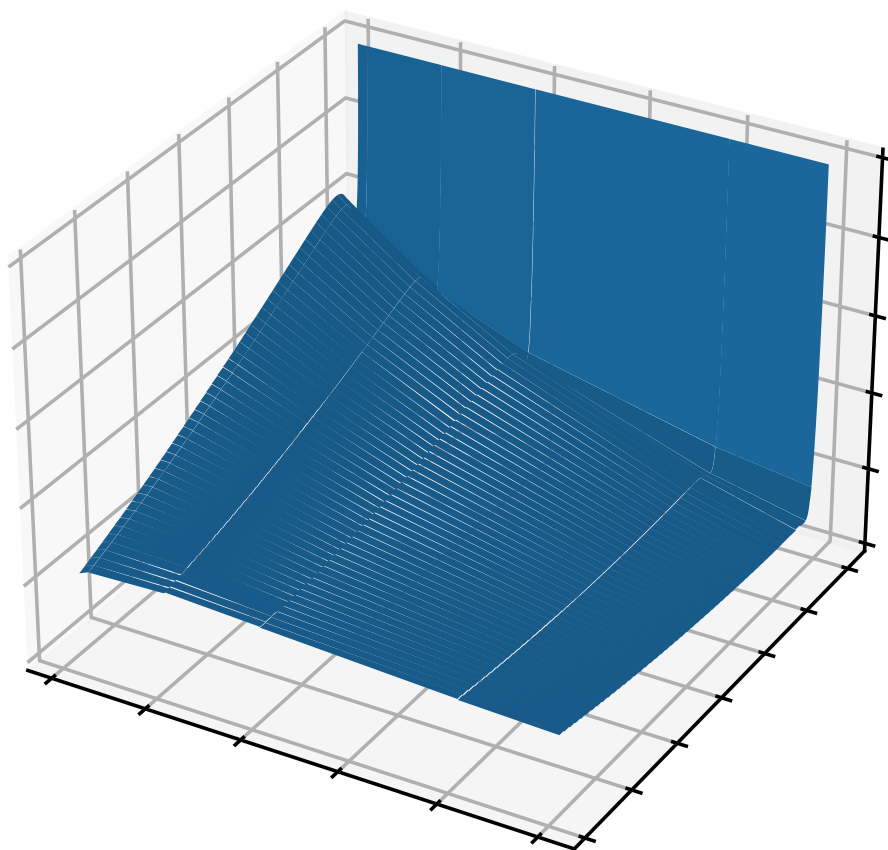


# Production d' $H_2$ par reformage catalytique de $CH_4$ et captage de $CO_2$ dans un réacteur.



# Table des matières

<b>Question 1</b>	<b>2</b>
1.1 Méthode de la sécante . . . . .	2
1.2 Méthode de la bisection . . . . .	2
<b>Question 2</b>	<b>3</b>
2.1 Création d'odefunction . . . . .	3
2.2 Résolution d'équation différentielle . . . . .	3
<b>Question 3</b>	<b>4</b>
3.1 Modèle sans capture de $CO_2$ . . . . .	4
3.2 Effet du flux de CaO sur le modèle . . . . .	4
3.3 Effet du flux des gaz sur le modèles . . . . .	4
3.4 Effet de la composition du gaz d'entrer sur le modèle . . . . .	4
<b>Question 4</b>	<b>5</b>

# 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  $abs(x_0 - x_1) \geq tol$ )

Logiquement nous devons donc compromettre 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  $5 \times 10^{-4}$  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  $y_1 \times (x_1 - x_0)$  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  $y_1 - y_0$  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 bisection

Les arguments sont les mêmes que dans la méthode de la sécante avec la même tolérance de  $5 \times 10^{-4}$  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 fonction traitée est continue.

Pour se rapprocher de la racine nous allons itérer pendant *k* itérations égal à  $\log_2(\frac{x_1 - x_0}{2 \times tol})$  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 reçoit 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 défaut 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  $5 \times 10^{-7}$  et notre max-step égale à  $1 \times 10^{-6}$ , 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 supérieur. Nous concluons que *solve\_ivp* est beaucoup plus efficace, que ce soit pour le temps d'exécution que pour la précision.

# Question 3

Pour cette question 3, nous avons introduit les arguments optionnels *mode* et *param* à odefunction. *mode* nous permet de tester différentes conditions de notre réacteur. Lorsque *mode* est à 0, le système fonctionne avec les paramètres de base, lorsqu'il est à 1, le modèle ne prend pas en compte la carbonatation, lorsqu'il est à 2, on remplace *us* par la valeur de *param* et lorsqu'il est à 3, on remplace *ug* par la valeur de *param*.

## 3.1 Modèle sans capture de $CO_2$

Afin de comparer un modèle de réacteur avec et sans capture de  $CO_2$ , nous avons mis le mode de odefunction à 3, soit un modèle sans capture de  $CO_2$ . Pour simuler ceci, nous avons remplacé la conversion fractionnaire ( $X$ ), le taux de consommation de  $CO_2$  par carbonatation ( $r_{cbn}$ ) et la vitesse d'entrée du  $CaO$  dans le réacteur ( $us$ ) par 0 car ils n'interviennent plus dans le modèle sans capture de  $CO_2$ .

((EXPLIQUER GRAPHIQUES)))

## 3.2 Effet du flux de $CaO$ sur le modèle

Pour analyser l'effet du flux d'entrée de  $CaO$ , nous avons fait varier la valeur de  $us$  à l'aide du mode 2. Nous l'avons fait varier de 0.001 à 0.025  $m.s^{-1}$ .

((EXPLIQUER GRAPHIQUES)))

## 3.3 Effet du flux des gaz sur le modèles

Pour analyser le flux des gaz, nous avons fait varier  $ug$  à l'aide du mode 3 en changeant la valeur de 1 à 3  $m.s^{-1}$  avec un pas de 0.5.

((EXPLIQUER GRAPHIQUES)))

## 3.4 Effet de la composition du gaz d'entrer sur le modèle

Dans le but de modifier les compositions initiales de gaz, nous avons fait varier le rapport des concentrations initiales en  $H_2O$  et  $CH_4$ .

((EXPLIQUER GRAPHIQUES)))

## Question 4