Capacité numérique : évolution temporelle de la température d'un système en réaction chimique

Objectif et introduction

Tracer, à l'aide d'un langage de programmation, l'évolution temporelle de la température pour un système siège d'une transformation adiabatique modélisée par une seule réaction chimique dont les caractéristiques cinétiques et l'enthalpie standard de réaction sont données. **Durée = 2h**.

I. Hypothèses et données

$n_0 = 1$ mol de peroxyde d'hydrogène est placée dans $m_{eau} = 1$ kg d'eau à $T_0 = 298$ K contenu dans un calorimètre de capacité thermique nulle. Un volume négligeable d'une solution concentrée de chlorure ferreux est ajoutée de sorte à catalyser la réaction de dismutation :

$$H_2O_{2(aq)} = H_2O_{(\ell)} + \frac{1}{2}O_{2(g)} \qquad \Delta_r H^\circ = -94,6\ \text{kJ}\cdot\text{mol}^{-1} \text{ à } 298\ K$$

Cette réaction présente un ordre 1 par rapport à H_2O_2. On souhaite tracer l'évolution de la température du système jusqu'à ce que la transformation soit terminée à 99 %.

Données :

- *capacité thermique massique de l'eau liquide : $c_{eau} = 4,18\ \text{J}\cdot\text{K}^{-1}\cdot\text{g}^{-1}$;*
- *énergie d'activation de la réaction : $E_a = 72,4\ \text{kJ}\cdot\text{mol}^{-1}$;*
- *paramètre pré-exponentiel : $A = 1,3.10^{11}\ \text{min}^{-1}$.*

Données

R = 8,314 S.I.

n_0 = 1 mole.

$\Delta rH°$ = - 94,6 kJ.mol^{-1}.

E_a = 72,4 kJ.mol^{-1}.

A = 1,3.10^{11} min^{-1}.

T_0 = 298 K.

m_{eau} = 1000 g.

$C_P°(H_2O(l)) = C_P°$ = 4,18 J.K^{-1}.g^{-1}

Densité de l'eau d = 1.

On note dans le programme xi l'avancement molaire ξ de la réaction, T la température de la réaction et t le temps.

II- Programmation

Q1. Introduire dans le programme python les données de l'énoncé (annexe 1)

Q2. Ecrire la loi d'Arrhénius portant sur la constante de vitesse k(T) de la réaction et la coder en python.

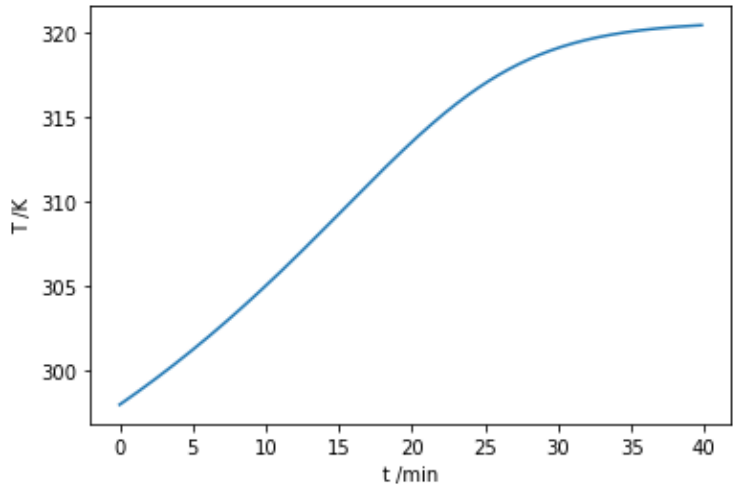
Q3. On choisira par la suite un pas de temps dt égal à 0,1 min en vue d'appliquer la méthode d'Euler. On prendra comme valeurs initiales : t_i = 0 ; T_i = 298 K et xi_i = 0. Compléter le programme en indiquant ces valeurs et rappeler le principe de la méthode d'Euler (annexe 2).

Q4. Calculer le temps de demi-réaction et justifier le choix de la valeur du pas dt.

Q5. Compléter les instructions dxi et dT dans la boucle permettant de remplir les listes ls_xi et ls_T contenant les valeurs d'avancement et de température.

Q6. Compléter le programme python pour tracer T(t).

On obtient le graphe ci-dessous pour l'évolution de la température en fonction du temps :



Annexe 1 : programme Python à compléter

```
import numpy as np
import matplotlib.pyplot as plt


# Données relatives au problème A COMPLETER
R =  #J/mol/K
n0 =  #mol
Delta_rH0 = #J/mol
Ea =  #J/mol/K
A = #min-1
T0 =  #K
Ceau =  #J/g/K
meau =  #g

# Fonction constante de vitesse k(Temp)
def k(Temp):
    A COMPLETER

# Pas de temps
dt = A COMPLETER # min

# Méthode d'Euler
t_i = A COMPLETER # min instant initial
T_i = A COMPLETER # K température initale
xi_i = A COMPLETER # mol avancement initial

ls_t = [t_i] # liste des instants
ls_T = [T_i] # liste des températures
ls_xi = [xi_i] # liste des avancements

xi = xi_i # initialisation de xi
t = t_i # initialisation de t
T = T_i  # initialisation de T

while xi < 0.99*n0 :
    t = t + dt
    ls_t.append(t)
    dxi = A COMPLETER
    xi = xi + dxi
    ls_xi.append(xi)
    dT = A COMPLETER
    T = T + dT
    ls_T.append(T)


# Tracé de T = f(t)

A COMPLETER
```

Annexe 2 : rappel sur la méthode d'Euler

Introduction :

Les équations différentielles rencontrées dans le cadre du cours ou des exercices de physique en 1ère année ne sont pas toutes intégrables « à la main ». Cela veut dire qu'il n'existe pas de solution analytique à l'équation différentielle traitée. Il faut alors faire appel à une **intégration numérique** pour obtenir par **approximation** la solution numérique de l'équation différentielle. Cette solution est présentée sous forme d'un tableau de valeurs approchées ou, et c'est le cas usuel, sous forme d'un graphique.

Il existe différentes méthodes permettant de réaliser cette intégration numérique. Parmi celles-ci, la **méthode d'Euler** se distingue par sa simplicité de mise en oeuvre et les bons résultats qu'elle donne sur les problèmes rencontrés usuellement en physique.

Principe de la méthode d'Euler

- **principe général :**

La méthode d'Euler est une méthode itérative qui permet de déterminer numériquement une solution approchée d'une équation différentielle **du premier ordre**, **la condition initiale étant connue**. Cela correspond à résoudre ce qu'on appelle **problème de Cauchy** :

→ équation différentielle du 1er ordre : $y'(t) = F(y(t), t)$ sur l'intervalle $[t_0 ; t_f]$
→ condition initiale connue : $y(t_0) = y_0$

- **procédure algorithmique :**

La procédure algorithmique est la suivante :

a) on commence par réaliser une subdivision régulière de l'intervalle $[t_0 ; t_f]$ en sous-intervalles de largeur dt appelée **pas de résolution**. Cela revient à générer un ensemble discret de points d'abscisses t_k telles que $t_k = t_0 + k\,dt$ (i.e. l'ensemble des points $\{t_0, t_0 + dt, t_0 + 2dt, t_0 + 3dt, ..., t_f\}$).

b) on cherche ensuite une **valeur numérique approchée** de $y(t_k)$. Cette valeur approchée est notée y_k. La valeur y_0 étant connue grâce à la condition initiale, on détermine les valeurs y_k en procédant de proche en proche grâce à une **relation de récurrence** (appelée encore schéma numérique) obtenue par approximation du nombre dérivé :

$$\frac{dy}{dt}(t_k) \approx \frac{y(t_{k+1}) - y(t_k)}{t_{k+1} - t_k} = \frac{y(t_{k+1}) - y(t_k)}{dt} \quad \text{d'où} \quad y(t_{k+1}) \approx y(t_k) + \frac{dy}{dt}(t_k)\,dt \quad \text{soit} \quad \boxed{y_{k+1} = y_k + F(y_k, t_k)\,dt}$$

- **influence de la valeur du pas d'itération choisi :**

La proximité de la solution numérique approchée avec la solution exacte de l'équation différentielle dépend fortement de la valeur du pas dt choisi pour discrétiser l'intervalle de résolution. Ce comportement est illustré ci-contre.

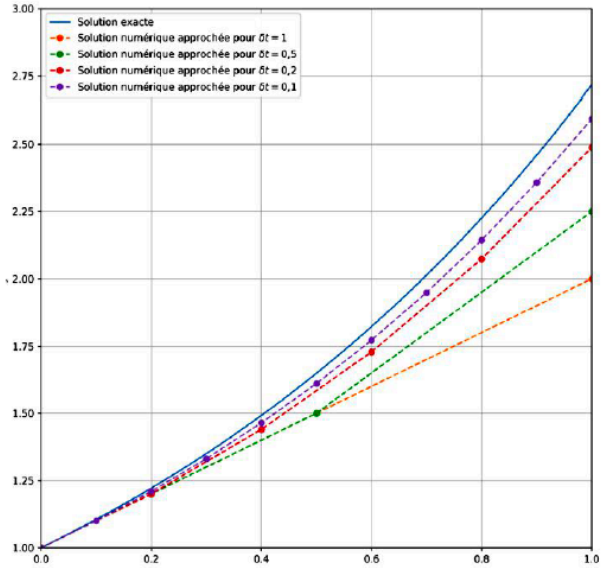


Figure 1 : résolution de l'équation différentielle y' (t) = y (t) sur l'intervalle [0 ; 1] avec la condition initiale y (0) = 1 et confrontation de la solution exacte avec les solutions numériques approchées fournies par la méthode d'Euler pour différentes valeurs du pas dt.

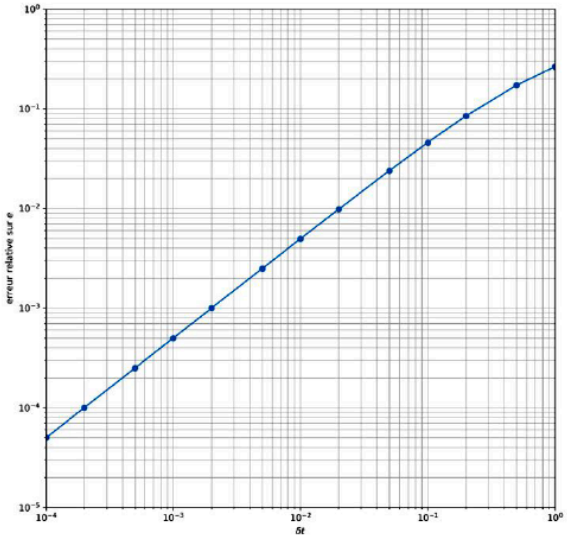


Figure 2 : erreur relative commise sur l'évaluation du nombre e par résolution de l'équation différentielle y' (t) = y (t) sur l'intervalle [0 ; 1] avec la condition initiale y (0) = 1 par la méthode d'Euler pour différentes valeurs du pas dt.

Par ailleurs, la complexité de la méthode d'Euler (qui caractérise le nombre d'opérations élémentaires réalisées par la machine lors de l'exécution de l'algorithme) est inversement proportionnelle au pas dt. Toutes choses égales par ailleurs, **le temps de calcul est donc d'autant plus long que le pas dt est petit**.

À la lumière des deux points discutés précédemment, il apparaît que **le choix du pas de discrétisation dt doit résulter d'un compromis entre précision et rapidité d'obtention de la solution numérique**. En Physique, la modélisation du problème considéré fait généralement apparaître au moins un temps caractéristique d'évolution τ (ou une distance caractéristique ℓ) et il est judicieux de choisir le pas de discrétisation dt (ou dx dans le domaine spatial) relativement à ce paramètre caractéristique : une valeur du pas comprise entre 0,01 τ et 0,1 τ s'avère souvent adaptée.

Corrigé : programme Python complété

```
import numpy as np
import matplotlib.pyplot as plt


# Données relatives au problème
R = 8.314 #J/mol/K
n0 = 1 #mol
Delta_rH0 = -94.6e3 #J/mol
Ea = 72.4e3 #J/mol/K
A = 1.3e11 #min-1
T0 = 298 #K
Ceau = 4.18 #J/g/K
meau = 1000 #g

# Fonction constante de vitesse k(Temp)
def k(Temp):
    return A*np.exp(-Ea/(R*Temp))

# Pas de temps
dt = 0.1 # min

# Méthode d'Euler
t_i = 0 # min instant initial
T_i = 298 # K température initale
xi_i = 0 # mol avancement initial

ls_t = [t_i] # liste des instants
ls_T = [T_i] # liste des températures
ls_xi = [xi_i] # liste des avancements

xi = xi_i # initialisation de xi
t = t_i # initialisation de t
T = T_i  # initialisation de T

while xi < 0.99*n0 :
    t = t + dt
    ls_t.append(t)
    dxi = k(T)*(n0-xi)*dt
    xi = xi + dxi
    ls_xi.append(xi)
    dT = -dxi*Delta_rH0/(Ceau*meau)
    T = T + dT
    ls_T.append(T)


# Tracé de T = f(t)
plt.plot(ls_t,ls_T)
plt.xlabel('t /min')
plt.ylabel('T /K')
plt.show()
```