KOYTCHA ALIBAY Mustapha
CORDONNIER Adrien
Avril 2023
A3PSB

PROJET – Modélisation des réactions chimiques

Enseignants : H. El-Otmany, C. Peschard, L. Bletzacker, P. Geoffroy

Table des matières

INTRODUCTION

Le projet de modélisation des réactions chimiques dans le cadre du cours MA322 - Quadrature & résolution numérique d'EDO a pour objectif d'expliquer les phénomènes oscillatoires de concentrations d'éléments chimiques au cours du temps. Les réactions chimiques sont décrites par des équations chimiques impliquant des réactifs et des produits. Les réactions complexes peuvent impliquer une série d'étapes au cours desquelles certains éléments, qui agissent comme réactifs, deviennent des produits dans des processus ultérieurs, entraînant des fluctuations périodiques des concentrations de ces éléments chimiques. Ce projet se concentre sur un modèle Brusselator simple. Dans ce modèle, les concentrations de deux éléments chimiques x et y varient au temps $t > 0$ selon un système d'équations différentielles ordinaires (ODE). Le système est instable pour une valeur particulière du paramètre, provoquant des oscillations constantes autour du point d'équilibre. Ce projet nous permettra de mettre en pratique des méthodes d'intégration numérique pour la résolution d'ODE et de comprendre les phénomènes oscillatoires et les phénomènes d'instabilité dans les réactions chimiques.

Vous trouverez dans la suite de ce rapport les réponses aux questions demandées avec un apport personnel et des observations. Certaines questions ont été répondu par un paragraphe ou à l'aide de tableaux contenant des graphiques. Bonne lecture !!

1. Contexte et problème de CAUCHY

Dans notre vie, des réactions chimiques se produisent autour de nous, et même en nous, tous les jours. Il y tellement de réactions chimiques différentes que les scientifiques trouvent utile de les regrouper en catégories (dosage, photosynthèse, décomposition, acidobasique, combustion, ...) pour les classer afin de les maitriser en produisant des éléments essentiels (Eau H_2O, Sel de table $NaCl$, Carbonate de sodium Na_2CO_3, Hydrogène H_2, Oxygène O_2,...) pour nous et notre planète. Ces réactions chimiques sont décrites par des équations chimiques qui s'écrivent en général sous la forme :

$$A + B \rightleftharpoons C + D$$

Où les éléments A et B appelés les réactifs qui réagissent entre eux pour donner naissance aux éléments C et D, appelés les produits de la réaction chimique.

L'étude de processus chimiques complexes fait intervenir très souvent des réactions qui s'enchaînent. Il peut arriver que certains éléments jouant le rôle de réactifs à une certaine étape soient aussi le produit dans une étape ultérieure. Ce phénomène, appelé autocatalyse, donne lieu à des changements périodiques dans la concentration de ces éléments chimiques. Dans ce cadre de réactions oscillantes, nous trouvons les réactions suivantes que nous pourrons consulter sur YouTube en cliquant sur les liens associés :

i. Cœur battant de mercure [1], [2]
ii. Réaction de Beloussov-Jabotinski [3] : $3HOOC - CH - COOH + 4B_rO \rightarrow 4Br^- + 9CO + 6HO$
iii. Réaction de Briggs-Rauscher[4] : $IO_3^- + 2H_2O_2 + CH_2(CO_2H)_2 + H + ICH(CO_2H)_2 + 2O_2 + 3H_2O$

Dans ce projet, il s'agit d'illustrer ce phénomène à travers le modèle simple du Brusselator, ou les concentrations x et y de deux éléments chimiques évoluent au cours du temps $t > 0$ suivant le système d'équations différentielles ordinaires (EDO) :

$$(1) \qquad \begin{cases} x' = a + x^2y - (b+1)x \\ y' = bx - x^2y \end{cases}$$

Où a et b sont deux réels et x' et y' sont les dérivées en temps des concentrations x et y associées respectivement aux éléments chimiques A et B. Le système part d'un état initial $(x(0), y(0)) = (x_0, y_0) \in \mathbb{R}^2$ donné et tend vers une position d'équilibre lorsque $b \leq 1 + a^2$ et on écrit ainsi :

$$\lim_{t \to +\infty} (x(t), y(t)) = (a, \frac{b}{a})$$

En revanche, le système suivant est instable pour $b > 1 + a^2$ et donne lieu à des oscillations perpétuelles dans la concentration des deux éléments autour du point $(a, \frac{b}{a})$.

1. Présenter le système (1) sous la forme d'un problème de Cauchy suivant :

$$(2) \qquad \begin{cases} Y' = f(y), & \forall t \in I, \\ Y(0) = Y_0 = (x_0, y_0) & donné \end{cases}$$

[1] . https://www.youtube.com/watch?v=INMHbMSOLb8
[2] . https://www.youtube.com/watch?v=MW9PHH2wI5o
[3] . https://www.youtube.com/watch?v=IBa4kgXI4Cg
[4] . https://www.youtube.com/watch?v=O4T0D2E9Bj4

On pose :

$$Y(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$$

$$\Leftrightarrow Y'(t) = \begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix} = \begin{pmatrix} a + x^2 y - (b+1)x \\ bx - x^2 y \end{pmatrix}$$

Ainsi, on peut poser le problème de Cauchy suivant :

$$\begin{cases} Y' = f(y), & \forall t \in I, \\ Y(0) = Y_0 = (x_0, y_0) \end{cases}$$

On retrouve donc bien l'équation le problème de Cauchy (2).

2. Indiquer quel $Y(t)$ à poser et quelle fonction f en précisant le domaine de départ et d'arrivée. On peut démontrer que ce problème de Cauchy admet une et une seule solution (bien que, au sens strict, F n'est pas une fonction lipschitzienne par rapport à Y).

On doit poser :

$$f(y) = \begin{pmatrix} a + x^2 y - (b+1)x \\ bx - x^2 y \end{pmatrix}$$

$$Départ = \{(x, y) \in R^2 \mid b > 1 + a^2 \text{ ou } x^2 y < bx\}$$

$$Arrivée = \{(x, y) \in R^2 \mid b \leq 1 + a^2 \text{ et } x = a, y = \frac{b}{a}\}$$

Montrons que f(y) est 2 lipschitzienne :

$$|f(y1) - f(y2)| = \begin{pmatrix} |a + x^2 y1 - (b+1)x - (a + x^2 y2 - (b+1)x| \\ |bx - x^2 y1 - (bx - x^2 y2)| \end{pmatrix}$$

$$\leq \begin{pmatrix} x^2 |(y1 - y2)| \\ x^2 |(y1 - y2)| \end{pmatrix}$$

$$\leq \begin{pmatrix} x^2 (|y1| - |y2|) \\ x^2 (|y1| - |y2|) \end{pmatrix}$$

$$\leq L \begin{pmatrix} |y1| - |y2| \\ |y1| - |y2| \end{pmatrix} \text{ avec } L = x^2$$

Ainsi, x^2 n'est pas une constante donc $f(y)$ est lipschitzienne seulement sur un intervalle particulier. On peut alors dire que comme $f(y)$ est continue sur cet intervalle et est 2-Lipschitzienne sur celui-ci, le problème de Cauchy possède donc une et une seule solution.

2. Méthode d'EULER explicite

Pour $T > 0$ fixe, nous cherchons une approximation numérique de $Y(t)$ solution du problème de Cauchy (2) pour tout instant $t \in I = [0,\ T]$. On souhaite approcher $Y(t)''$ au mieux possible en $N + 1$ instants de temps $t \in I$, que l'on notera t_n $(avec\ 0 \leq n \leq N)$. On considère le cas où les instants sont uniformément repartis sur l'espace temporel I de sorte que, en posant $\tau = \frac{T-0}{N}$, on aura $t_n = n\tau$.

1. En utilisant le développement de Taylor de $Y(t_n + \tau)$ autour de t_n, montrer que

$$Y(t_n + 1) = Y(t_n + \tau) = Y(t_n) + \tau f\big(Y(t_n)\big) + O(\tau^2), \qquad 0 \leq n \leq N - 1$$

En négligeant les termes d'ordre supérieur à τ, rappeler le principe de la méthode d'Euler explicite.

En utilisant le développement de Taylor de $Y(t_n + \tau)$ autour de t_n, on a :

$$Y(t_n + \tau) = Y(t_n) + \tau * Y'(t_n) + \left(\frac{\tau^2}{2}\right) * Y''(t_n) + O(\tau^3)$$

En négligeant les termes d'ordre supérieur à τ (c'est-à-dire $O(\tau^3)$ et supérieur), on obtient :

$$Y(t_n + \tau) \approx Y(t_n) + \tau * Y'(t_n)$$

En utilisant la définition de la dérivée $Y'(t_n) = f(Y(t_n))$ (puisque $Y'(t_n) = dY(t_n) / dt$), on peut écrire :

$$Y(t_n + \tau) \approx Y(t_n) + \tau * f(Y(t_n))$$

Ce qui correspond au principe de la méthode d'Euler explicite :

À chaque étape n, on calcule $Y(t_n + \tau)$ en utilisant l'approximation :

$$Y(t_n + \tau) \approx Y(t_n) + \tau * f(Y(t_n))$$

où $f(Y(t_n))$ est la fonction qui définit le système d'équations différentielles ordinaires à résoudre.

Appliquer la méthode d'Euler explicite pour donner une solution approchée de $Y(t)$. Pour cela, on pourra :

a) Implémenter une fonction $f(t,\ y)$ qui, étant donné le vecteur Y, retourne le vecteur $f(Y)$.

```
def Brusselator(a, b):
    def f(Y):
        x, y = Y
        dxdt = a + x**2 * y - (b+1)* x
        dydt = b*x - x**2*y
        return np.array([dxdt, dydt])
    return f
```

La fonction Brusselator prend comme paramètre a et b puis va retourner la fonction f(Y) de taille (1, 2). f prend comme paramètre un vecteur Y.

b) Implémenter un schéma d'Euler **Euler Explicit** qui, partant de l'approximation Y_n a un certain instant t_n, renvoie l'approximation Y_{n+1} à l'instant suivant $t_{n+1} = t_n + \tau$.

```
def euler_explicit(f, t, y, h):
    y_n = y + h * f(t, y)
    t_n = t + h
    return t_n, y_n
```

La fonction euler_explicit prend pour paramètre une fonction f, un temps t, un vecteur (1, 2) y et un pas de temps h. La fonction va calculer et retourner y_{n+1} et t_{n+1} grâce aux paramètres entrés.

c) En utilisant la condition initiale Y_0, faire une boucle sur tous les temps permettant d'obtenir l'approximation Y_n à tous les instants $n = 0, \ldots, N$. Stocker la trajectoire, il s'agit plus précisément de stocker l'ensemble des approximations Y_n pour tous les instants t_n dans une liste.

```
def Euler_explicit(f, ic, T0, Tf, N):
    h = (Tf - T0) / N
    Lt = np.arange(T0, Tf + h, h)
    Ly = np.zeros((len(Lt), len(ic)))
    Ly[0] = ic

    for n in range(len(Lt) - 1):
        Ly[n+1] = Ly[n] + h * f(Ly[n])

    return Lt, Ly
```

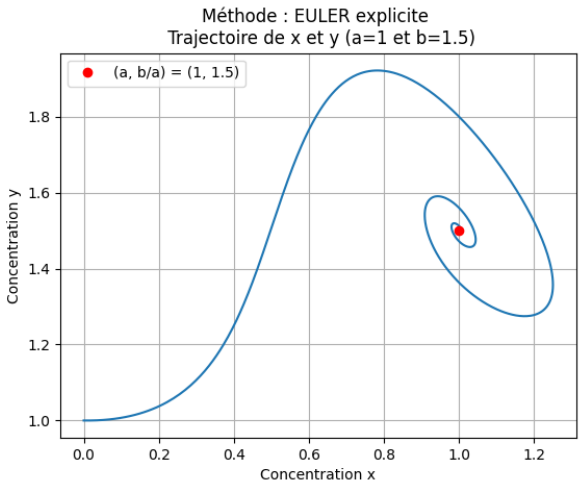
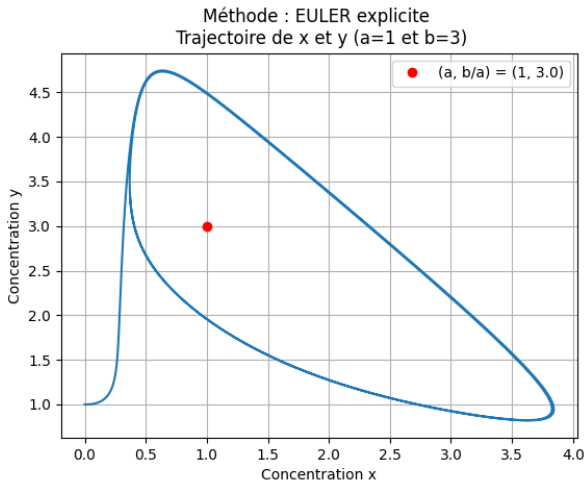
La fonction Euler_explicit reprend le même processus que la fonction définit précédemment. On vient calculer le pas h tel que $h = \frac{T_f - T_0}{N}$, N étant le nombre de subdivision, T_f le temps final et T_0 le temps initial. On prend ic qui est un vecteur (1, 2) qui contient les valeurs initiales du système. Par le même procédé que dans la fonction précédente, on vient calculer y_{n+1} dans une boucle for.

d) Utiliser le code pour obtenir la trajectoire de 0 à T pour

$$T = 18, \quad a = 1, \quad b = 3, \quad x(0) = 0, \quad y(0) = 1, \quad N = 1000$$

Au vu des valeurs de a et b, prédire le comportement du système.

e) Même question pour $a = 1$ et $b = 1{,}5$

$a = 1, \quad b = 1.5$	$a = 1, \quad b = 3$
Ici, $b < 1 + a^2$, donc le système va tendre vers une position d'équilibre au point $\left(a, \frac{b}{a}\right)$.	Ici, $b > 1 + a^2$, donc le système va osciller autour du point $\left(a, \frac{b}{a}\right)$.
	

2. Pour visualiser les résultats, nous conseillons de suivre les étapes ci-dessous :

i. Coder une fonction **concentrationPlotting** qui représente graphiquement l'évolution temporelle des concentrations x et y pour $t = t_n$, $n = 0, \ldots, N$. Vous pouvez stocker la figure avec la commande **plt. savefig('evol-concentration.pdf')**.

```
# Fonction pour afficher les graphiques des concentrations x et y en fonction du temps
def concentration_plotting(t, y, NomMethode, a, b):
    plt.plot(t, y[:, 0], label="x")
    plt.plot(t, y[:, 1], label="y")
    plt.xlabel("Temps")
    plt.ylabel("Concentration")
    plt.title("Méthode : {t} \n Evolution temporelle des concentrations x et y (a={a}
et b={b})".format(t=NomMethode, a=a,
b=b))
    plt.savefig("Evol-concentration.pdf")
    plt.legend()
    plt.grid()
    plt.show()
```

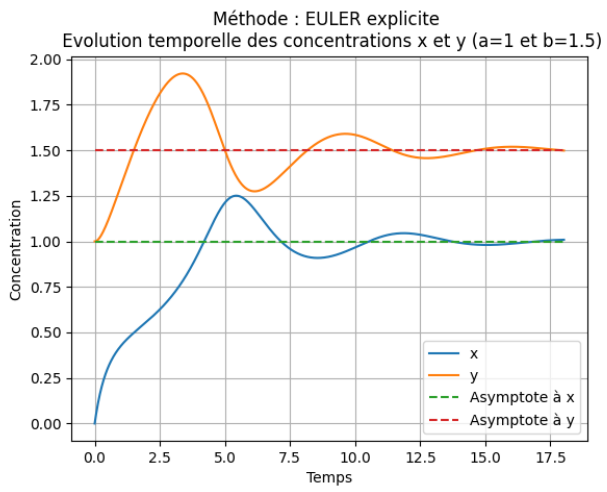
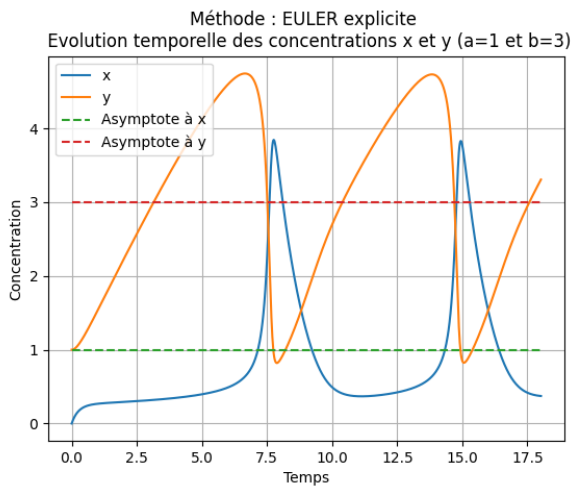
La liste Ly retourné par Euler_explicite contient des vecteurs (1, 2) où la première valeur du tableau correspond aux variations de x et la deuxième valeur du tbleau contient les variations de y. Il suffit de plot toutes les premières valeurs du tableau et ensuite toutes les deuxièmes valeurs de ce même tableau, le tout en fonctions de la liste Lt qui contient les variations du temps.

ii. Coder une fonction **trajectoryPlotting** qui représente graphiquement la trajectoire (x, y) des concentrations pour tout $t = t_n,\ n = 0, \ldots, N$. Stocker la figure avec le nom **'trajectoire.pdf'**.

```
# Fonction pour afficher le graphique de la trajectoire (x,y) en fonction du temps
def trajectory_plotting(y, NomMethode, a, b):
    plt.plot(y[:, 0], y[:, 1])
    plt.xlabel("Concentration x")
    plt.ylabel("Concentration y")
    plt.title("Méthode : {t} \n Trajectoire de x et y (a={a} et
b={b})".format(t=NomMethode, a=a,    b=b))
    plt.savefig("trajectoire.pdf")
    plt.grid()
    plt.show()
```

Ici on vient exprimer x en fonction de y.

iii. Utiliser les fonctions ci-dessus pour représenter graphiquement l'évolution temporelle en utilisant les données des questions (d) et (e).

$a = 1, \quad b = 1.5$	$a = 1, \quad b = 3$
	

```
Temps d'exécution pour la méthode Euler explicite : 0.004017829895019531 secondes
```

```
Temps d'exécution pour la méthode Euler explicite : 0.003961324691772461 secondes
```

On observe que lorsque les conditions de stabilité sont respectées ($b < 1 + a^2$), les concentrations x et y tendent respectivement vers a et b. Dans le second cas, les concentrations x et y oscillent autour de leurs asymptotes.

Les temps d'exécutions restent équivalents peu importe la valeur de b.

3. Une (autre) méthode RUNGE-KUTTA d'ordre 4

Si la fonction $t \to Y(t)$ est suffisamment régulière, l'erreur d'approximation avec la méthode d'Euler explicite au temps final $t = T$ est de l'ordre du pas temps τ, c'est-à-dire il existe une constante $C > 0$ telle que :

$$||Y(T) - Y_N|| \leq C_\tau$$

Donc, lorsque $\tau \to 0$, on a convergence de la solution numérique vers la solution du système Y. Cependant, dans la pratique la constante C de l'inégalité est souvent très grande et augmente avec T. Pour cette raison, il est en général nécessaire de prendre des pas de temps τ extrêmement petits pour arriver à une précision convenable. Cela augmente considérablement le nombre d'instants intermédiaires et rallonge les temps de calcul et motive la recherche d'approximations dont l'erreur est de la forme :

$$||Y(T) - Y_N|| \leq C_\tau^r$$

Où $r > 1$ et $C > 0$. On parle de schémas d'ordre supérieur. Pour les construire, une première possibilité serait de prendre des développements de Taylor d'ordre supérieur mais cela est en général très coûteux voire parfois même impossible en fonction du problème.

Une façon plus simple de construire des schémas d'ordre supérieur est le schéma Runge-Kutta. Dans cette approche, f est évaluée en un certain nombre de points et l'approximation se construit comme suit :
Pour tout $0 \leq n \leq N - 1$, on pose

$$Y_{n+1} := Y_n + \tau \sum_{j=1} \gamma_j K_n^j$$

où

$$k_n^i := \begin{cases} f(Y_n) \; si \; i = 1 \\ f\left(Y_n + \tau \sum_{j=1} \beta ij K_n^j\right) \; si \; 2 \leq i \leq M \end{cases}$$

Les coefficients $\beta_{ij} \; et \; \gamma_i$ sont usuellement donnes dans le tableau ci-dessous appelée tableau de Butcher :

α_i	β_{ij}
	γ_j

Où les indices i sont des indices de ligne et les indices j des indices de colonne. Le tableau comporte aussi des coefficients α_i dont on n'aura pas réellement besoin dans ce problème (la fonction f a une seule variable y). Dans ce type d'approche, étant donné un ordre d'approximation r souhaité, la question est de trouver des bons coefficients $\beta_{ij} \; et \; \gamma_i$ qui satisfont cet ordre d'approximation r. Afin de réduire le cout des calculs, il faut aussi que M soit le plus petit possible. Les questions tournant autour de la recherche des coefficients $\beta_{ij} \; et \; \gamma_i$ donnant un certain ordre r et avec le plus petit degré possible M est une question difficile et nous nous contenterons de considérer le tableau suivant qui donne un schéma d'ordre $r = 4$, que nous appellerons RKF4, pour lequel $M = 6$.

0						
$\frac{1}{4}$	$\frac{1}{4}$					
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				
$\frac{12}{13}$	$\frac{1932}{2197}$	$\frac{-7200}{2197}$	$\frac{7296}{2197}$			
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$\frac{-845}{4104}$		
$\frac{1}{2}$	$\frac{-8}{27}$	2	$\frac{-3544}{2565}$	$\frac{1859}{4104}$	$\frac{-11}{40}$	
γ_i	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$\frac{-9}{50}$	$\frac{2}{55}$

1. Implémenter un schéma **RKF4Butcher** qui, partant de l'approximation Y_n a un certain instant t_n renvoie l'approximation Y_{n+1} à l'instant $t_n + \tau$ avec la méthode Runge-Kutta d'ordre 4.

```
def RKF4Butcher(f, ic, T0, Tf, N):
    h = (Tf - T0) / N  # step size if h is constant
    Lt = np.arange(T0, Tf + h, h)
    Ly = np.empty((len(Lt), np.size(ic)), dtype=float)  # Matrice contenant les valeurs
de f(Y)
    Ly[0, :] = ic
    M = 6

    # Coefficients de Butcher
    beta = np.array([[0, 0, 0, 0, 0, 0],
                     [1 / 4, 0, 0, 0, 0, 0],
                     [3 / 32, 9 / 32, 0, 0, 0, 0],
                     [1932 / 2197, -7200 / 2197, 7296 / 2197, 0, 0, 0],
                     [439 / 216, -8, 3680 / 513, -845 / 4104, 0, 0],
                     [-8 / 27, 2, -3544 / 2565, 1859 / 4104, -11 / 40, 0]])

    gamma = np.array([16/135, 0, 6656/12825, 28561/56430, -9/50, 2/55])
    alpha = np.array([0, 1/4, 3/8, 12/13, 1, 1/2])  # Pas utile ici

    K = [0] * M  # Liste contenant des Kij

    # Boucle principale
    for i in range(len(Lt) - 1):
        for k_j in range(M):
            if k_j == 0:
                K[k_j] = f(Ly[i,:])
            else:
                s = sum([beta[k_j, var] * K[var] for var in range(k_j)])
                K[k_j] = f(Ly[i,:] + h * s)

        Ly[i+1, :] = Ly[i, :] + h * sum([gamma[k_j] * K[k_j] for k_j in range(len(K))])

    return Lt, Ly
```

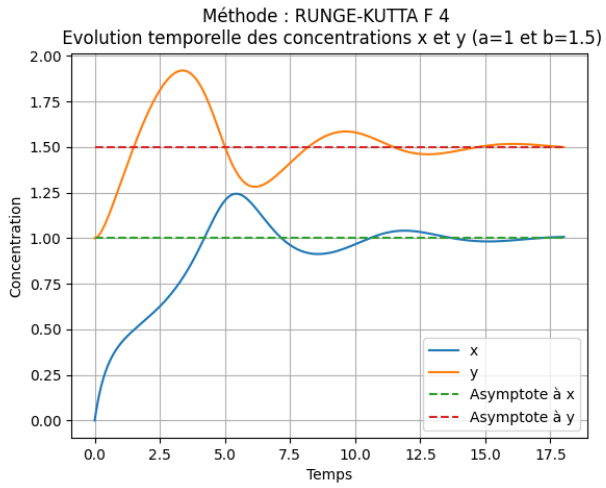
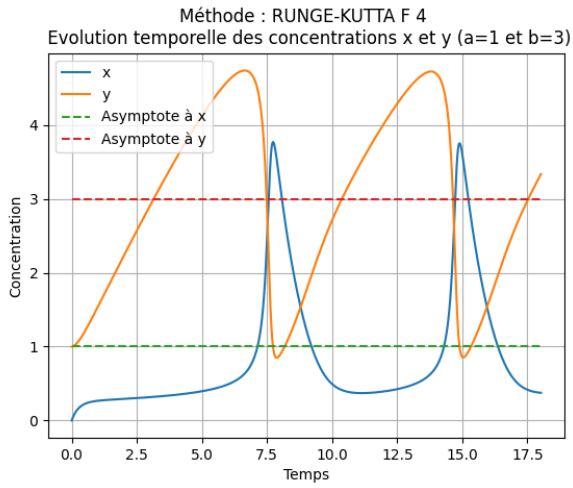
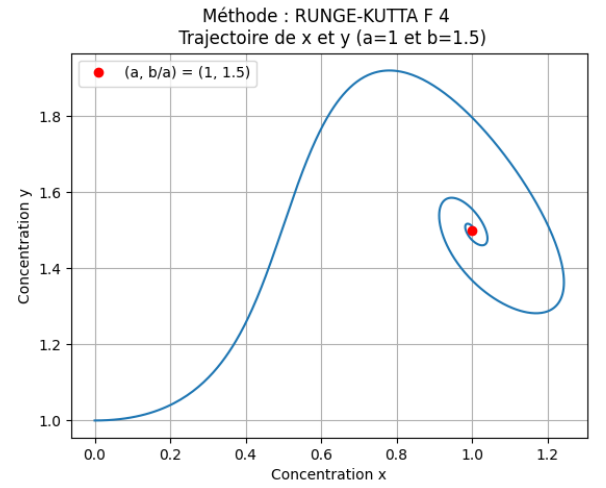
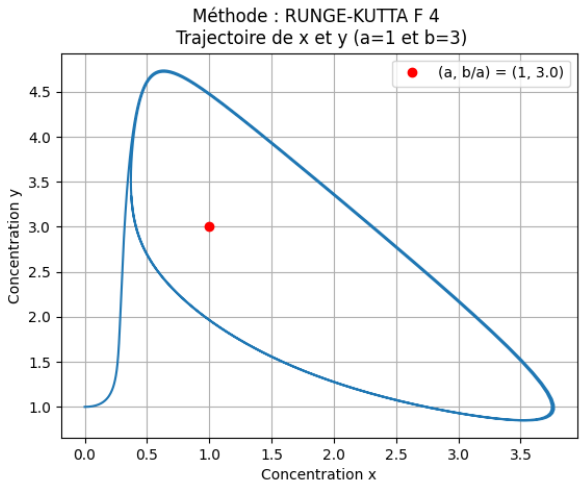
On définit la liste Lt qui contient tous temps espacés du pas h. On vient ensuite définir Ly, la liste contenant toutes les valeurs Y_k. Dans la boucle principale, on vient calculer :

$$k_n^i := \begin{cases} f(Y_n) \; si \; i = 1 \\ f\left(Y_n + h\sum_{j=1} \beta ij K_n^j\right) \; si \; 2 \leq \; i \leq M \end{cases}$$

Pour ensuite trouver :

$$Y_{n+1} := \; Y_n + h\sum_{j=1} \gamma_j K_n^j$$

2. Appliquer ce schéma pour les données en question (d) et (e) de la section 2 pour obtenir la trajectoire de 0 à T.
3. Tracer l'évolution temporelle des concentrations ($x \; et \; y$) et la trajectoire obtenue.

$a = 1, \quad b = 1.5$	$a = 1, \quad b = 3$
	
	
`Temps d'exécution pour la méthode de Runge-Kutta 4 Butcher : 0.06482577323913574 secondes`	`Temps d'exécution pour la méthode de Runge-Kutta 4 Butcher : 0.06681942939758301 secondes`

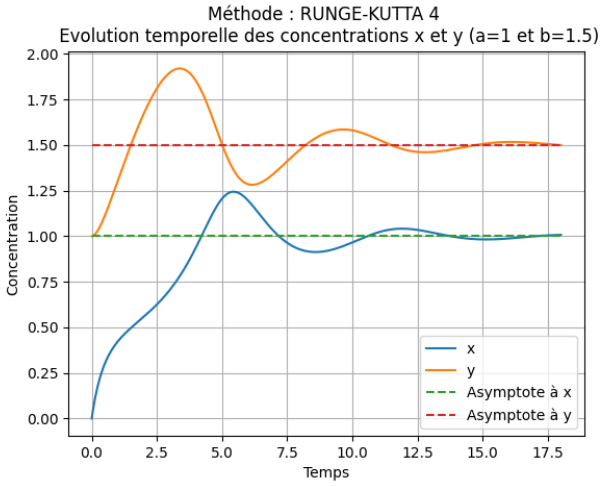
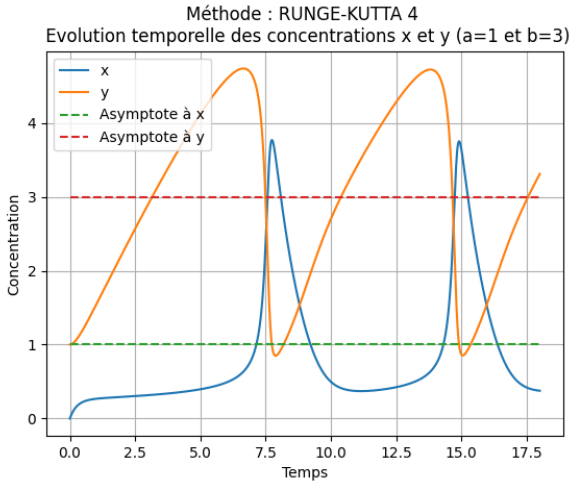
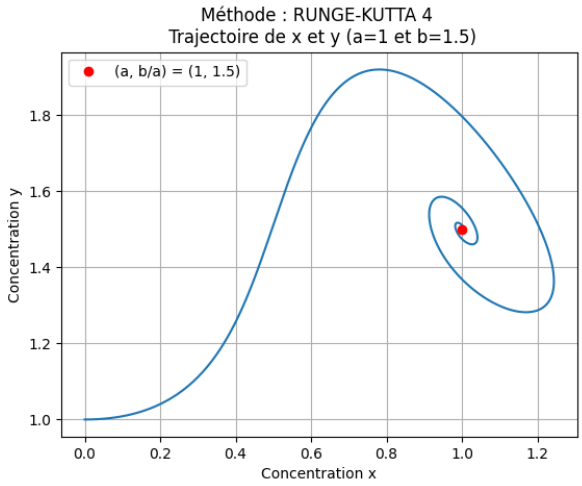
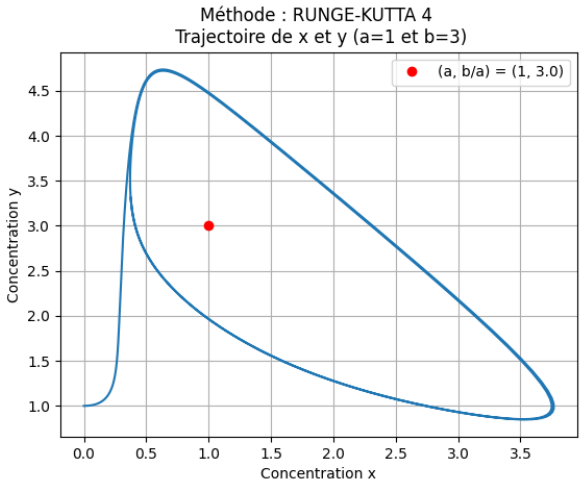
Ces courbes tracées à l'aide de la méthode de Runge-KuttaF4 sont identiques à celles de la méthode d'Euler explicite. Les différences peuvent avoir lieu au niveau du temps d'exécutions et de la précision de chacune de ces méthodes. On peut donc observer, après avoir calculé les différents temps de résolution des méthodes, que la méthode d'Euler explicit est plus rapide que la méthode de Runge-Kutta.

4. Exposer le schéma RK4 classique vu dans le cours pour le système (2), implémenter RK4 et tracer l'évolution temporelle des concentrations (x *et* y) et la trajectoire obtenue. Commenter les résultats.

Le code pour RK4 vu en cours est le suivant :

```
#Runge-Kutta fourth order for iniline function & vector function
def RK4(f, ic, T0, Tf, N):
      h = (Tf - T0) / N      #step size if h is constant
      Lt = np.linspace(T0, Tf, N)
      Ly = np.empty((N, np.size(ic)),dtype = float)
      Ly[0,:] = ic
```

```
    for i in range(N-1):
        #if h isn't constant, we use h=t[i+1]-t[i]
        k1 = h*f(Ly[i,:])
        y1 = Ly[i,:] + 1/2*k1
        k2 = h* f(y1)
        y2 = Ly[i,:] + 1/2*k2
        k3 = h* f(y2)
        y3 = Ly[i,:] + k3
        k4 = h* f(y3)
        k = (k1+2*k2+2*k3+k4)/6
        Ly[i+1,:] = Ly[i,:] + k
    return Lt, Ly
```

$a = 1, \quad b = 1{,}5$	$a = 1, \quad b = 3$
	
	
`Temps d'exécution pour la méthode de Runge-Kutta d'ordre 4 : 0.021940708160400 39 secondes`	`Temps d'exécution pour la méthode de Runge-Kutta d'ordre 4 : 0.021940469741821 29 secondes`

Le schéma RK4 vu en cours est plus rapide au niveau de l'exécution que RKF4. Cela peut s'expliquer par le nombre important de calculs effectués par la fonction RKF4.

4. Méthode de RUNGE-KUTTA d'ordre 4-5 et pas de temps adaptatif

En utilisant deux méthodes de Runge-Kutta d'ordre différents, il est possible de calculer la différence des solutions pour estimer l'erreur d'approximation à chaque pas de temps τ. Il est possible de garantir que l'erreur ne dépasse pas un certain seuil $\varepsilon_{max} > 0$ en adaptant le pas de temps τ au problème étudié. Pour mettre en œuvre cette stratégie, nous considèrerons deux méthodes de Runge-Kutta construits de sorte à ne différer que dans la valeur des coefficients γ_i : la méthode RKF4 d'ordre 4 présentée dans le début de la section 3 et un schéma d'ordre 5 dont les coefficients $\widetilde{Y}_i$ sont définis dans la dernière ligne du tableau de Butcher étendu :

0						
$\frac{1}{4}$	$\frac{1}{4}$					
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				
$\frac{12}{13}$	$\frac{1932}{2197}$	$\frac{-7200}{2197}$	$\frac{7296}{2197}$			
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$\frac{-845}{4104}$		
$\frac{1}{2}$	$\frac{-8}{27}$	2	$\frac{-3544}{2565}$	$\frac{1859}{4104}$	$\frac{-11}{40}$	
γ_i	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$\frac{-9}{50}$	$\frac{2}{55}$
$\bar{\gamma}_i$	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$\frac{-1}{5}$	0

En notant $\delta_i = \gamma_i - \bar{y}_i$, l'estimation de l'erreur d'approximation entre deux points consécutifs $t_n \; et \; t_n + \tau$ s'écrit ainsi

$$E(t_n + \tau) = \tilde{Y}_{n+1} - \bar{Y}_{n+1} = \tau \sum_{i=1}^{M} (y_i - \bar{y}_i) \cdot K^i = \tau \sum_{i=1}^{M} \delta_i K^i$$

Posons $\varepsilon(t_n + \tau) := ||E(t_n + \tau)||\infty$, l'algorithme adaptatif fonctionne comme suit. Supposons que l'on ait calculé la solution a l'instant $t = t_n$. On cherche de façon itérative :

i. Le pas de temps τ_{next} qui garantisse que $\varepsilon(t_n + \tau_{next}) < \varepsilon_{max}$. Ce pas de temps définit l'instant suivant $t_{n+1} = t_n + \tau_{next}$ où la solution $\tilde{Y}_{n+1}$ est calculée avec RKF4.

ii. Le pas de temps τ_{init} qui sera utilisé à l'instant suivant pour initialiser l'algorithme d'actualisation des pas de temps. Cet algorithme se décrit ainsi :
   - Pour $k = 0$, on fixe τ_0 a la valeur τ_{init} trouvée à l'instant précèdent.
   - Pour $k \geq 0$, on définit un nouveau pas de temps suivant la règle

$$\tau_{k+1} = \tau_k * \begin{cases} 0.1, & si \; e < 0.1 \\ 5, & si \; e > 5 \\ e, & sinon \end{cases} \quad avec \; e = 0.9 \left(\frac{\varepsilon_{max}}{\varepsilon(t_n + \tau_k)} \right)^{\frac{1}{5}}$$

Si $\varepsilon(t_n + \tau_k) > \varepsilon_{max}$, on passe à l'itération suivante.
Si $\varepsilon(t_n + \tau_k) \leq \varepsilon_{max}$, alors on arrête les itérations et on pose $\tau_{next} = \tau_k$, $\tau_{init} = \tau_{k+1}$. On calcule alors la solution a $t_{n+1} = t_n + \tau_{next}$, et on passe au temps suivant. Pour traiter ce nouvel instant, on utilisera la valeur de τ_{init} obtenue pour initialiser l'algorithme d'actualisation du pas de temps.

1. Implémenter le schéma de Runge-Kutta adaptatif **stepRK45** qui, partant de l'approximation Y_n à un certain instant $t = t_n$ et d'une valeur τ_{init} pour actualiser le pas de temps, renvoie la solution approchée Y_{n+1} avec RKF4 pour la nouvelle valeur $t_{n+1} = t_n + \tau_{next}$ et qui calcule τ_{init} pour l'instant suivant en fixant

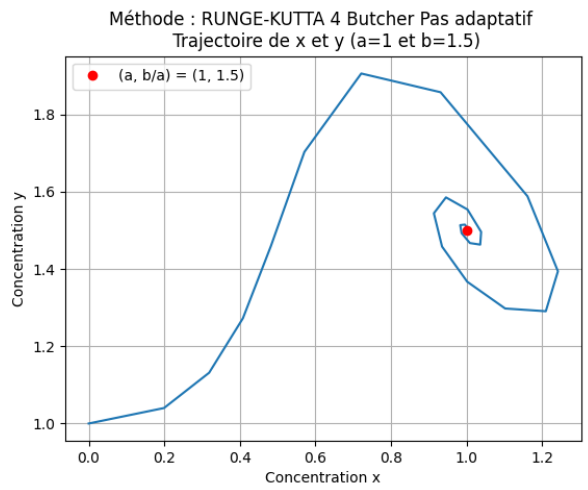
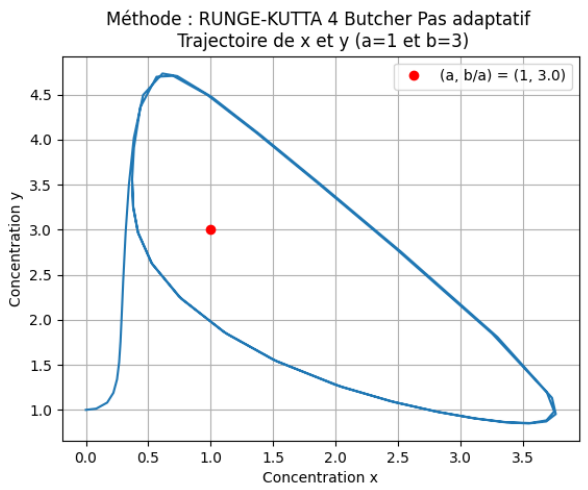
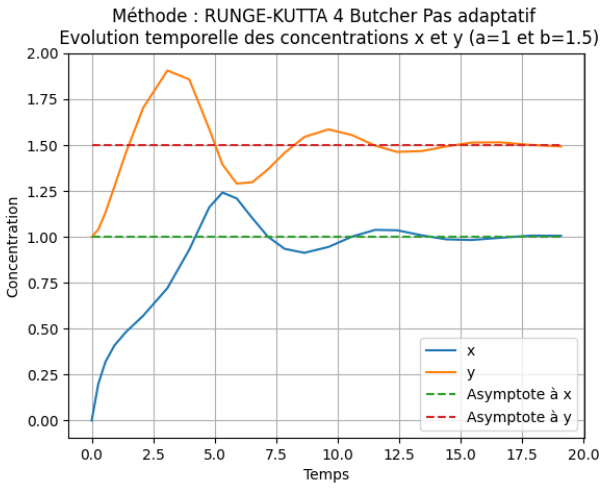
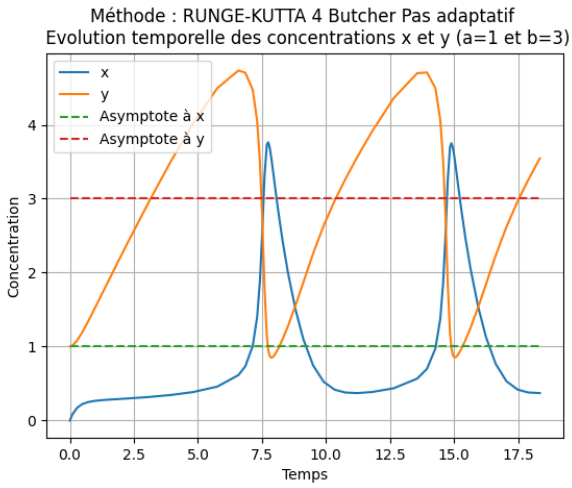
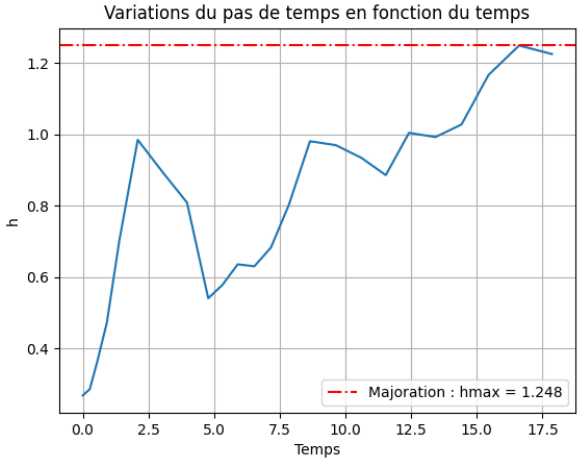
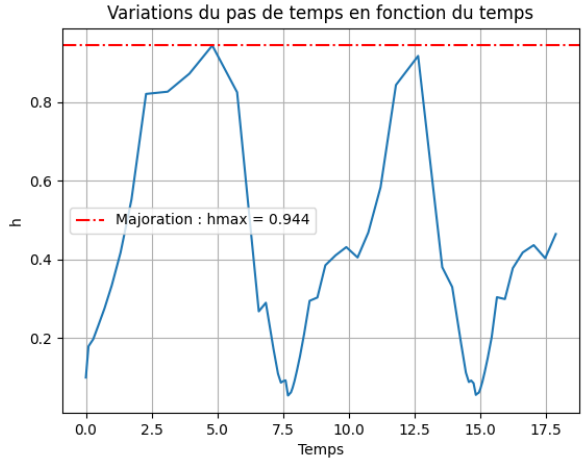
$$\varepsilon_{max} = 10^{-4}$$

```python
def stepRK45(f, ic, T0, Tf, epsilonmax=10**-4):
    t0 = time.time()
    h = 10  # Pas initial peu important
    Lt = [T0]
    Ly = np.zeros((1, 2))  # Matrice contenant les valeurs de f(Y)
    Ly[0, :] = ic # Définition des C.I
    M = 6

    # Coefficients de Butcher
    beta = np.array([[0, 0, 0, 0, 0, 0],
                     [1 / 4, 0, 0, 0, 0, 0],
                     [3 / 32, 9 / 32, 0, 0, 0, 0],
                     [1932 / 2197, -7200 / 2197, 7296 / 2197, 0, 0, 0],
                     [439 / 216, -8, 3680 / 513, -845 / 4104, 0, 0],
                     [-8 / 27, 2, -3544 / 2565, 1859 / 4104, -11 / 40, 0]])

    gamma = np.array([16 / 135, 0, 6656 / 12825, 28561 / 56430, -9 / 50, 2 / 55])
    gamma_bar = np.array([25 / 216, 0, 1408 / 2565, 2197 / 4104, -1 / 5, 0])
    delta = gamma - gamma_bar

    alpha = np.array([0, 1 / 4, 3 / 8, 12 / 13, 1, 1 / 2])  # Pas utile ici

    K = [0] * M  # Liste contenant des Kij
    i = 0
    l_h = []

    while Lt[-1] < Tf:
        for k_j in range(M):
            if k_j == 0:
                K[k_j] = f(Ly[i, :])
            else:
                s = sum([beta[k_j, var] * K[var] for var in range(k_j)])
                K[k_j] = f(Ly[i, :] + h * s)

        E = h * sum([delta[k_j] * K[k_j] for k_j in range(M)])  # Calcul de l'erreur
        epsilon = np.linalg.norm(E, np.inf)  # Calcul de la norme infinie de l'erreur

        if epsilon < epsilonmax:
            Lt.append(Lt[-1] + h)
            Ly = np.vstack([Ly, Ly[i, :] + h * sum([gamma[k_j] * K[k_j] for k_j in
range(M)])]) # Ajout d'une nouvelle ligne dans Ly

            l_h.append(h)
            i += 1

        e = 0.9 * (epsilonmax / epsilon) ** (1 / 5)

        if e < 0.1:
            h *= 0.1
        elif e > 5:
            h *= 5
        else:
            h *= e

    t1 = time.time()
    print("Temps méthode RUNGE-KUTTA 4 Pas adaptatif : ", t1-t0, "s")
    return Lt, Ly, l_h
```

Cet algorithme reproduit le pseudo-code énoncé ci-dessus. On reprend l'algorithme de RF4Butcher et on vient modifier le pas à chaque itération, on vient ensuite calculer l'erreur et epsilon afin de déterminer si on prend le pas ou non. Si ce n'est pas le cas on passe à l'itération suivante.

2. Appliquer ce schéma pour les données en question (d) et (e) de la section 2 pour obtenir la trajectoire de 0 à T et prédire le comportement de système.
3. Tracer l'évolution temporelle des concentrations (x, y) et la trajectoire obtenue.
4. Tracer la variation de la valeur des pas de temps en fonction du temps.

Pour une tolérance à $\varepsilon_{max} = 10^{-4}$.

$a = 1, \quad b = 1{,}5$	$a = 1, \quad b = 3$
	
	
	

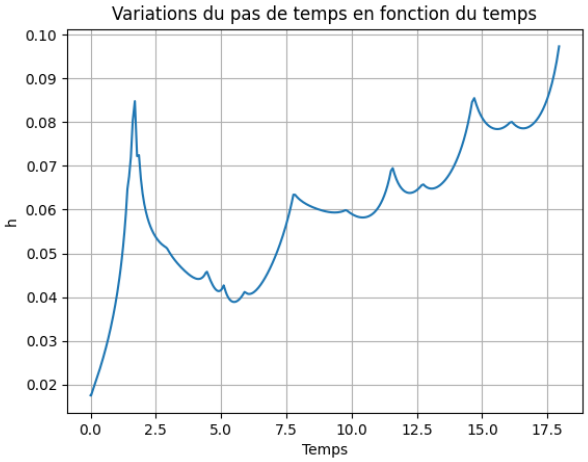
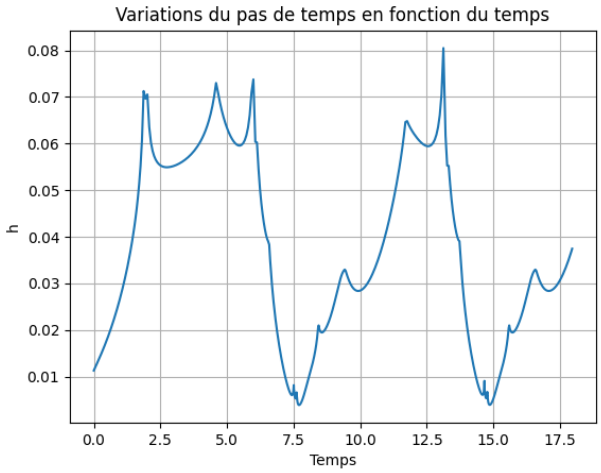
`Temps méthode EULER explicite :`
`0.00802922248840332 s`
 	
`Temps méthode RKF4Butcher :`
`0.14247632026672363 s`
	`Temps méthode EULER explicite :`
	`0.008238554000854492 s`
	
	`Temps méthode RKF4Butcher :`
	`0.14147281646728516 s`

```
Temps méthode RUNGE-KUTTA 4 :
0.04819679260253906 s

Temps méthode RUNGE-KUTTA4 Pas adaptatif:
0.004974126815795898 4 s
```

```
Temps méthode RUNGE-KUTTA 4 :
0.0466008186340332 s

Temps méthode RUNGE-KUTTA4 Pas adaptatif:
0.015146017074584961 s
```

Sur les courbes donnant le pas h en fonction du temps on remarque que celle-ci est majorée. Cette majoration est dû à l'erreur entrée par l'utilisateur. Si le pas est au-dessus de cette limite, on ne respecte plus la tolérance maximale. Cependant, si le pas est infinitésimal, le temps d'exécution serait trop long pour nos appareils, c'est pour cela qu'on vient multiplier le pas par 5 si ε est trop petit par rapport à ε_{max}.

Plus on diminue la tolérance, plus les courbes sont « lisses », on en déduit que le but du pas adaptatif est de converger le plus rapidement vers la solution du système tout en gardant une marge d'erreur que l'on maîtrise via le paramètre ε_{max}.

$a = 1, \quad b = 1{,}5$	$a = 1, \quad b = 3$
	

Ici on a lancé RF4Butcher à pas adaptatif avec $\varepsilon_{max} = 10^{-10}$, comme prévu les courbes sont beaucoup plus lisses car il y a plus de valeurs dans les listes. On observe aussi que le pas h évolue de la même manière mais est beaucoup plus petit qu'avant.

5. Comparaison des méthodes étudiées

Comparons maintenant la précision des différentes méthodes vues précédemment entre elles en prenant a = 1 et b = 3.

Le code utilisé est le suivant :

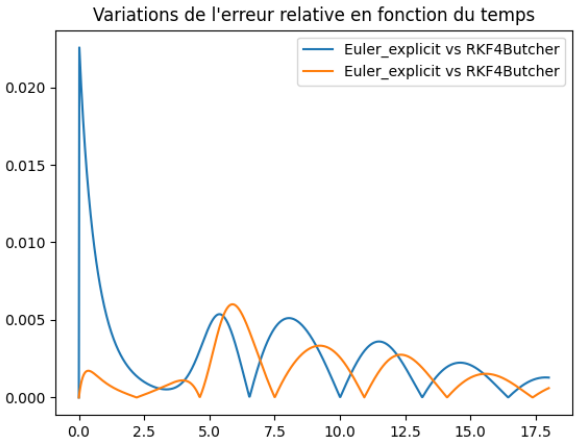
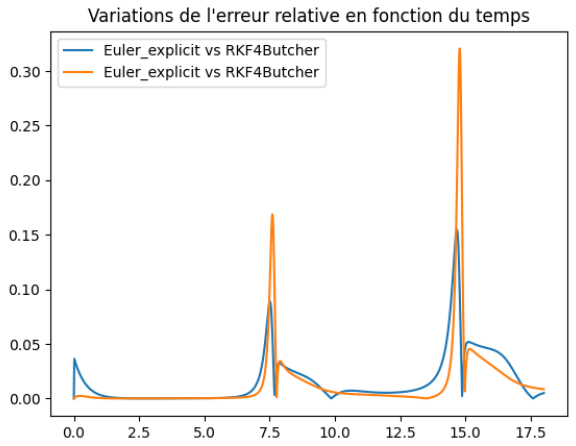
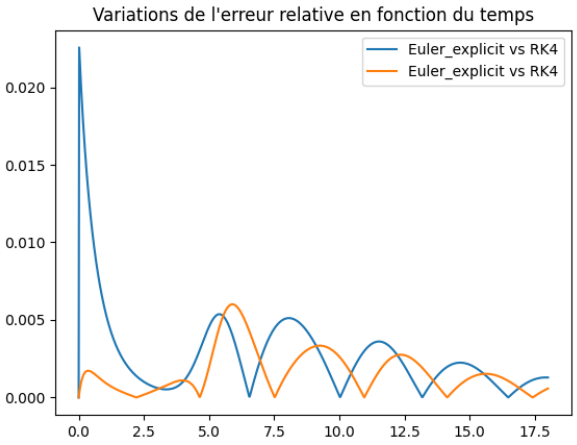
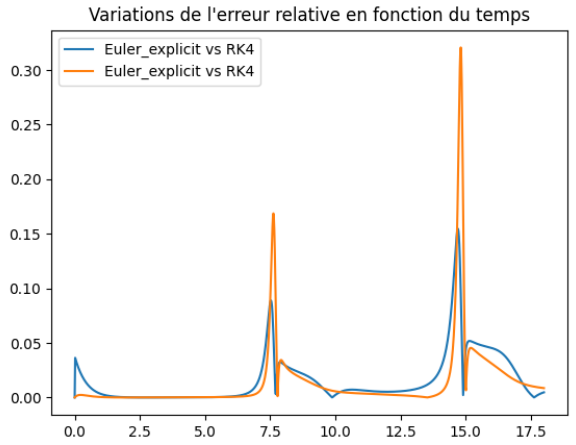
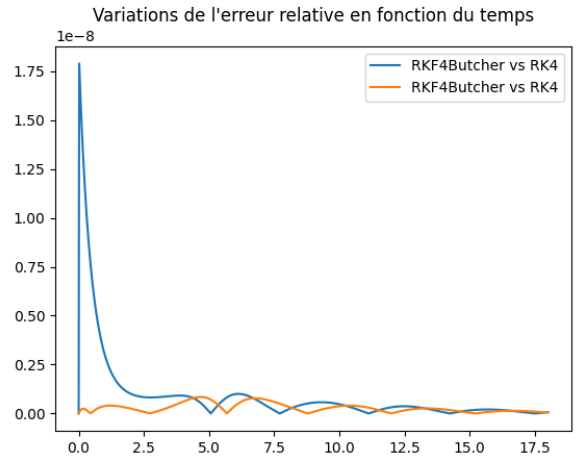
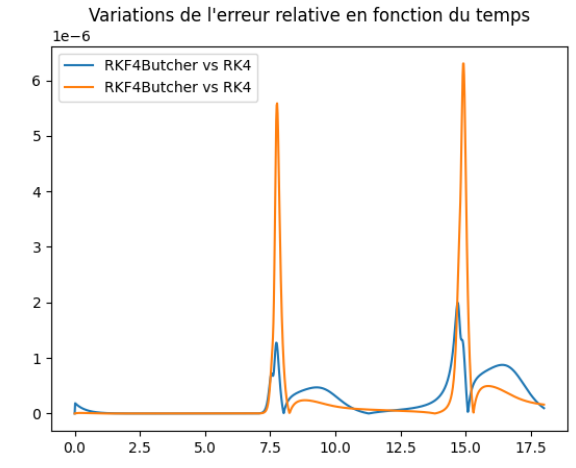
```
# Fonction pour calculer l'erreur relative maximale entre deux solutions
def erreur_relative(y1, y2):
    return np.max(np.abs(y1 - y2) / np.maximum(1e-10, np.abs(y2)))

a, b = 1, 3
f = Brusselator(a, b)
ic = np.array([0, 1])
T0, Tf = 0, 18
N = 1000
epsilonmax = 1e-6
# Euler_explicit vs RKF4Butcher
Lt_euler, Ly_euler = Euler_explicit(f, ic, T0, Tf, N)
Lt_rkf4, Ly_rkf4 = RKF4Butcher(f, ic, T0, Tf, N)
erreur_rel_euler_rkf4 = erreur_relative(Ly_euler, Ly_rkf4)
print("-------------------------------------------------------------------------------")
print("Erreur relative maximale entre Euler_explicit et RKF4Butcher : ",
erreur_rel_euler_rkf4)
print("-------------------------------------------------------------------------------")

# Euler_explicit vs RK4
Lt_euler, Ly_euler = Euler_explicit(f, ic, T0, Tf, N)
Lt_rk4, Ly_rk4 = RK4(f, ic, T0, Tf, N)
Ly_euler = np.delete(Ly_euler, -1, axis=0)  # enlève la dernière ligne de y1
Ly_euler = np.delete(Ly_euler, -1, axis=0)
erreur_rel_euler_rk4 = erreur_relative(Ly_euler, Ly_rk4)
print("-------------------------------------------------------------------------------")
print("Erreur relative maximale entre Euler_explicit et RK4 : ", erreur_rel_euler_rk4)
print("-------------------------------------------------------------------------------")

# RKF4Butcher vs RK4
Lt_rkf4, Ly_rkf4 = RKF4Butcher(f, ic, T0, Tf, N)
Lt_rk4, Ly_rk4 = RK4(f, ic, T0, Tf, N)
Ly_rkf4 = np.delete(Ly_rkf4, -1, axis=0)  # enlève la dernière ligne de y1
Ly_rkf4 = np.delete(Ly_rkf4, -1, axis=0)
erreur_rel_rkf4_rk4 = erreur_relative(Ly_rkf4, Ly_rk4)
print("-------------------------------------------------------------------------------")
print("Erreur relative maximale entre RKF4Butcher et RK4 : ", erreur_rel_rkf4_rk4)
print("-------------------------------------------------------------------------------")
```

Les résultats obtenus sont les suivants :

```
Temps méthode EULER explicite :  0.004988670349121094 s
Temps méthode RKF4Butcher :  0.06479454040527344 s
-------------------------------------------------------------------------------
Erreur relative maximale entre Euler_explicit et RKF4Butcher :  0.022560733751784315
-------------------------------------------------------------------------------
Temps méthode EULER explicite :  0.003989219665527344 s
Temps méthode RUNGE-KUTTA 4 :  0.02293848991394043 s
-------------------------------------------------------------------------------
Erreur relative maximale entre Euler_explicit et RK4 :  0.022560752044747872
-------------------------------------------------------------------------------
Temps méthode RKF4Butcher :  0.06781911849975586 s
Temps méthode RUNGE-KUTTA 4 :  0.02293872833251953 s
-------------------------------------------------------------------------------
Erreur relative maximale entre RKF4Butcher et RK4 :  1.7889366328527634e-08
-------------------------------------------------------------------------------
```

On remarque que la précision des méthodes de RK4 et RK4 Butcher est très similaire car l'erreur relative maximale entre les deux méthodes est de l'ordre de $1.79 * 10^{-8}$ (on peut considérer cette valeur comme 0). Si on compare maintenant les erreurs relatives maximales des méthodes de RK4 et la méthode d'Euler-Explicit on remarque que la méthode de RK4 est plus précise de l'ordre de 2%. Ainsi, Euler explicite est plus rapide mais RK4 est plus précise.

Voici les graphiques de l'erreur relative entre les méthodes

$a = 1, \quad b = 1{,}5$	$a = 1, \quad b = 3$
	
	
	

CONCLUSION

Dans ce projet, nous avons étudié un système dynamique non linéaire, caractérisé par deux équations différentielles couplées. Nous avons analysé l'évolution temporelle des concentrations des trajectoires, et nous avons constaté que le système présente un comportement cyclique, avec des fluctuations périodiques. Lorsque les conditions de stabilités sont respectées, les concentrations x et y tendent respectivement vers a et $\frac{b}{a}$ lorsque $t \to +\infty$.

Pour cela, nous avons premièrement utilisé la méthode d'Euler explicite pour discrétiser les équations et obtenir une solution numérique. Nous avons alors constaté que cette méthode est simple à implémenter mais nécessite des pas de temps très petits pour obtenir des résultats précis, ce qui augmente considérablement le temps de calcul.

Nous avons ensuite utilisé le schéma Runge-Kutta d'ordre 4 (RKF4) Butcher pour améliorer la précision de la solution numérique. Nous avons implémenté ce schéma en utilisant le tableau de Butcher fourni dans la partie 1, et avons obtenu des résultats beaucoup plus précis avec un temps de calcul raisonnable mais tout de même plus long que la méthode d'Euler Explicite. Nous avons par la suite utilisé la méthode Runge-Kutta 4 normale et avons trouvé un temps de calcul situé entre les deux méthodes d'avant.

La dernière méthode que nous avons utilisée est la méthode de Runge-Kutta 45 à pas adaptatif. En termes de précision, la méthode de RK45 était la plus précise lorsque $\varepsilon_{max} = 10^{-10}$, suivie de près par la méthode RK4.

En conclusion, l'étude de ce système dynamique nous a permis de mieux comprendre la dynamique des populations dans un écosystème et l'importance de l'interaction entre les différentes espèces. Les méthodes numériques que nous avons utilisées pour résoudre les équations différentielles nous ont permis d'obtenir des résultats précis et d'analyser le comportement du système à différents moments dans le temps.