

Dynamique de réseaux multi-échelles complexes sous contraintes: Modélisation et Analyse

Liam Toran

Stage de fin de M2A 2019 au Laboratoire J.A. Dieudonné de l'Université de Nice
sous la supervision de Yves D'Angelo, Rémi Catellier, Laurent Monasse

Thèmes : Mathématiques et leurs Interactions, Modélisation, Analyse, Processus Stochastiques, Équations aux dérivées partielles et ordinaires, Stabilité, Réaction-Diffusion, Ondes progressives, Simulation Numérique.

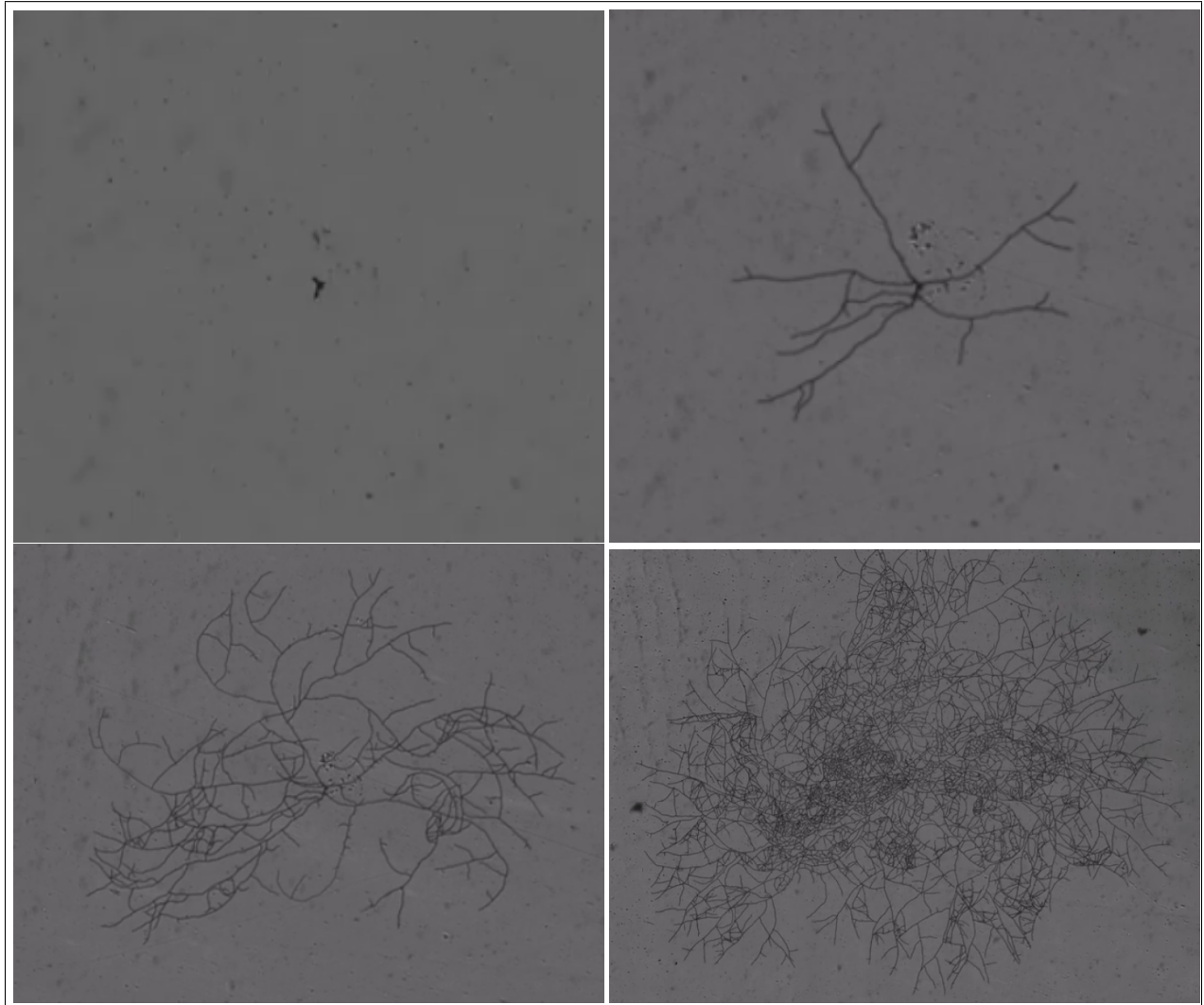


FIGURE 1 – Capture d'un réseau de champignon en expansion, par

Table des matières

1	L'équation de Fisher ou KPP	3
1.1	Preliminaire	3
1.2	Réaction	3
1.3	Réaction-Diffusion	3
1.4	Solutions en onde plane stationnaire / onde progressive	4
2	Dynamique de Réseaux en Croissance	5
2.1	Explication des équations du système (6)	5
2.2	Dérivation de l'équation "KPP avec mémoire"	6
3	Recherche de la vitesse d'onde des solutions progressives de l'Équation KPP avec Mémoire	8
3.1	Au voisinage de $(0, 0, C_0)$	8
3.1.1	Première condition : P' a deux annulations :	8
3.1.2	Deuxième condition : $\Delta > 0$:	8
3.1.3	Signe des racines au voisinage de $(0, 0, C_0)$	9
3.2	Au voisinage de $(0, \rho_\infty, 0)$	9
4	Schémas et Positivité	10
4.1	Pour l'équation différentielle ordinaire	10
4.1.1	Schéma semi-implicite I pour l'EDO	10
4.1.2	Schéma semi-implicite II pour l'EDO	10
4.2	Pour l'équation aux dérivées partielles	11
4.2.1	Schéma semi-implicite I pour l'EDP	11
5	Résolution numérique	12
5.1	Résolution de l'EDO	12
5.1.1	Résultat de la simulation de l'EDO	12
5.2	Résolution de l'EDP en 1D	13
5.2.1	Résultat de la simulation de l'EDP en 1D	13
	Appendices	14

1 L'équation de Fisher ou KPP

1.1 Préliminaire

Notre point de départ est l'équation de diffusion :

$$\partial_t u = \Delta u \quad (1)$$

En plus de la diffusion, considérons des modèles où le taux d'accroissement de u dépend aussi de la densité u .

Ceci donne les équations de reaction-diffusion :

$$\partial_t u = \Delta u + F(u) \quad (2)$$

où F est assez lisse.

Il est souvent naturel dans les modèles de considérer $F(u)$ proportionnel à u pour u petit ("croissance"), et quand u devient proche de 1, l'accroissement $F(u)$ s'arrête : $F(1) = 0$ ("saturation"). Ces types de modèles ont été introduits et examinés par les travaux de Fisher[1] et Kolmogorov, Petrovsky et Piskounov[2] (abrégés KPP).

Un exemple d'une telle équation est :

$$\partial_t u = \Delta u + u(1 - u) \quad (3)$$

qui sera dans la suite étudiée dans le cas 1-dimensionnel en x : $u = u(x, t)$.

1.2 Réaction

En observant les solutions constantes en x : $u(x, t) = v(t)$ dans (3), l'équation différentielle ordinaire (EDO ou ODE) :

$$\partial_t v = v - v^2 = F(v) \quad (4)$$

est obtenue.

Il y a deux équilibres ($F(v) = 0$) pour $v = 0$ et $v = 1$. Par le théorème de stabilité de Lyapunov, $F'(0) > 0$ montre que $v = 0$ est instable et $F'(1) < 0$ montre $v = 1$ est asymptotiquement stable.

1.3 Réaction-Diffusion

Dans l'espace $X = C_{b,unif}^0(\mathbb{R}, \mathbb{R})$ des fonctions bornées et uniformément continues, il y a existence locale et unicité des solutions de l'équation de Fisher-KPP (2). Grâce à un principe du maximum, il y a aussi existence globale et unicité des solutions.

Théorème 1. :

Existence et Unicité de la solution dans X : Soit $U_0 \in X$. Il existe une unique solution de l'équation de Fisher-KPP (2) $U \in C([0, \infty[, X)$ avec condition initiale U_0 .

Théorème 2. :

Principe du Maximum : Soit u_1 et u_2 deux solutions de (2). Si il existe t_0 tel que $u_1(x, t_0) < u_2(x, t_0)$ $\forall x$ alors $u_1(x, t) < u_2(x, t)$ $\forall x$ et $\forall t > t_0$

1.4 Solutions en onde plane stationnaire / onde progressive

Rappelons la définition d'une solution en onde plane stationnaire / onde progressive :

Définition 1.1. Solutions en onde plane stationnaire

Une solution en onde plane stationnaire est une solution de la forme $u(x, t) = h(x - st)$ où $c \in \mathbb{R}$.

On fera parfois l'abus de notation $u(x, t) = u(x - st)$

Sous des hypothèses “faibles” sur F , l'équation (2) : $\partial_t u = \Delta u + F(u)$ a alors la propriété surprenante et importante de posséder des solutions en ondes planes stationnaires liant les états d'équilibre $u = 1$ (à $-\infty$) et $u = 0$ (à $+\infty$).

Les hypothèses sur F portent en partie sur le fait que (2) doit posséder :

- Deux états d'équilibre $u = 1$ et $u = 0$: $F(0) = F(1) = 0$:
- Un phénomène de “croissance” : $F'(0) > 0$
- Un phénomène de “saturation” : $F'(1) < 0$

Étude des solutions en ondes progressive de (2) :

En substituant $u(x, t) = h(x - st) = h(y)$ pour $y = x - st$ dans (2), les équations obtenues sur h sont :

$$\begin{cases} h''(y) + sh'(y) + F(h(y)) = 0 \\ h(-\infty) = 1 \\ h(+\infty) = 0 \end{cases} \quad (5)$$

qui est une équation elliptique non linéaire. Le problème est donc de trouver s et $h \in C^2$ tels que le système (5) soit vérifié. Le théorème obtenu est le suivant :

Théorème 3. :

Soit $F \in C^1([0, 1])$ tel $F(0) = F(1) = 0$ et $F \geq 0$.

Il existe une vitesse critique s_* telle que $s_*^2 \geq 4F'(0)$ et :

- i) $\forall s \geq s_*$, l'équation (5) a une solution $h_s : \mathbb{R} \rightarrow]0, 1[$ de classe C^3 .

Cette solution est unique à translation près.

- ii) $\forall c < c_*$ l'équation (5) n'a pas de solution $h : \mathbb{R} \rightarrow [0, 1]$

Remarque : Dans le second cas il existe des solutions mais elles ne sont pas confinées dans $[0, 1]$ ni dans \mathbb{R}^+ , ce qui ne fait pas de sens dans une étude de densité de population.

2 Dyamique de Réseaux en Croissance

Dans cette section et par la suite nous étudions le modèle sur la croissance de réseaux dynamiques branchant, par exemple un champignon, proposé par Rémi Catellier, Yves D'Angelo et Cristiano Ricci, avec rescaling adéquat :

$$\begin{cases} \partial_t \mu + \nabla(\mu v) = f(C)(\mu + \rho) - \mu \rho \\ \partial_t(\mu v) + \nabla(\mu v \times v) + T \nabla \mu = -\lambda \mu v + \mu \nabla C - \mu v \rho \\ \partial_t \rho = F(v) \mu \\ \partial_t C = -b \rho C \end{cases} \quad (6)$$

L'inconnue μ représente la densité des apex du champignon.

L'inconnue ρ représente la densité des hyphes/ du réseau.

L'inconnue v représente la vitesse des apex.

L'inconnue C représente la concentration des nutriments.

Les paramètres T , λ et b sont des scalaires représentant la température, l'amortissement fluide sur la vitesse des apex, et le taux de consommation des nutriments par le réseau.

La fonction f indique l'influence de la concentration de nutriments sur la croissance du champignon. Pour avoir un état stationnaire sur la croissance du champignon, $f(0) = 0$ et $f(x)/x$ dans L^1 proche de 0 sont imposés.

La fonction F représente l'inverse du temps moyen passé par les apex dans un point donné, et est donné par l'expression :

$$F(V) = \left(\frac{1}{2\pi T}\right)^{\frac{d}{2}} \int_{\mathbb{R}^d} |v| \exp\left(-\frac{|v - V|^2}{2T}\right) dv \quad (7)$$

où d est la dimension du problème. Ceci est souvent simplifié en substituant $F(V)$ par une constante : $F(V) = F_0$.

2.1 Explication des équations du système (6)

Le champignon est un réseau branchant dynamique qui peut être étudié en deux parties : les apex (pointes du réseau) représentés par leur densité μ et les hyphes (branches du réseau) représentés par leur densité ρ

Les lignes du système (6) représentent :

i) La première ligne du système est le bilan de masse sur les apex avec le terme gauche classique $\partial_t \mu + \nabla(\mu v)$. Le terme de droite est composé de : - $f(C)(\mu + \rho)$ correspondant à une croissance proportionnelle à la concentration de nutriments du réseau et la masse existante d'apex et d'hyphes, - et un terme $-\mu \rho$ qui correspond à l'anastomose : une pointe qui rencontre une branche va fusionner avec elle et être détruite. Il y a un terme de croissance et un terme de saturation comme pour le modèle KPP.

ii) La deuxième ligne est le bilan de vitesse avec le terme de gauche classique $\partial_t(\mu v) + \nabla(\mu v \times v)$. Le terme $T \nabla \mu$ représente le mouvement brownien suivi par les apex. Le terme $-\lambda \mu v$ représente un amortissement fluide dans la physique du problème. Le terme $+\mu \nabla C$ représente la tendance des apex à aller vers les milieux de forte concentration. Le terme $-\mu v \rho$ représente la perte de vitesse du à l'anastomose.

iii) La troisième ligne correspond à la relation entre les branches et les pointes : la trace laissée par les apex sont les branches.

iv) La quatrième ligne décrit l'évolution de la concentration de nutriments : ils sont consommés par les hyphe avec un taux bC où b est une constante positive.

2.2 Dérivation de l'équation "KPP avec mémoire"

En faisant tendre T et λ vers $+\infty$, avec $\frac{T}{\lambda} = K$ constant, la deuxième ligne de (6) donne :

$$+K\nabla\mu = -\mu v \quad (8)$$

En injectant ceci dans la ligne 1 du système, on obtient le système de 3 inconnues suivant :

$$\begin{cases} \partial_t\mu = K\Delta\mu + f(C)(\mu + \rho) - \mu\rho \\ \partial_t\rho = F_0\mu \\ \partial_tC = -b\rho C \end{cases} \quad (9)$$

dit "KPP avec mémoire".

Soit (μ, ρ, C) vérifiant le système d'équations suivant :

$$\begin{cases} \partial_t \mu = f(C)(\mu + \rho) - \mu \rho \\ \partial_t \rho = F_0 \mu \\ \partial_t C = -b \rho C \end{cases} \quad (10)$$

Lemme 1. C est de signe constant.

En effet on a $C(t) = C(0) \exp(-b \int_0^t \rho(s) ds)$.

Lemme 2. Soit (μ, ρ, C) tel que $\mu(0) \geq 0$, $\rho(0) > 0$, $C(0) > 0$. Alors $\mu(t) > 0 \forall t$

Démonstration. Supposons par l'absurde que μ devient négatif alors soit $t^* = \min t > 0 / \mu(t) < 0$. Alors :

- $\mu(t) \geq 0 \forall t < t^*$

- $\partial_t \mu(t^*) \leq 0$ par définition de t^* . (Sinon $\mu(t^* + \epsilon) > 0 \forall \epsilon << 1$)

- $\rho(t) > 0 \forall t < t^*$ car $\partial_t \rho = F_0 \mu$ et $F_0 > 0$

- $\partial_t \mu(t^*) = f(C(t^*))\rho(t^*) > 0$ ce qui est en contradiction avec la deuxième affirmation. □

3 Recherche de la vitesse d'onde des solutions progressives de l'Équation KPP avec Mémoire

On a le modèle suivant :

$$\begin{cases} \partial_t \mu - \frac{T}{\lambda} \Delta \mu = f(C)(\mu + \rho) - \mu \rho \\ \partial_t \rho = F_0 \mu \\ \partial_t C = -b \rho C \end{cases} \quad (11)$$

où $f(0) = 0$ et f est positive. Typiquement, $f(C) = C$:

Dans la suite on pose $K = \frac{T}{\lambda}$

On recherche des solutions en onde plane, on pose s la vitesse d'onde et $\xi = x - st$.

$$\begin{cases} -s\mu' - K\mu'' = f(C)(\mu + \rho) - \mu \rho \\ -s\rho' = F_0 \mu \\ C' = \frac{b\rho C}{s} \end{cases} \quad (12)$$

Nos états stationnaires sont $(\mu, \rho, C) = \begin{cases} (0, 0, C_0) \\ (0, \rho_\infty, 0), \rho_i nfty > 0 \end{cases}$

3.1 Au voisinage de $(0, 0, C_0)$

Au voisinage de $(0, 0, C_0)$ on a, en posant $f(C_0) = f_0$:

$$\begin{cases} -s\mu' - K\mu'' = f_0(\mu + \rho) \\ -s\rho' = F_0 \mu \end{cases} \quad (13)$$

ce qui devient

$$\rho''' + \frac{s}{K}\rho'' + \frac{f_0}{K}\rho' - \frac{F_0 f_0}{Ks}\rho = 0 \quad (14)$$

de polynôme caractéristique

$$P(X) = X^3 + \frac{s}{K}X^2 + \frac{f_0}{K}X - \frac{F_0 f_0}{Ks} \quad (15)$$

Pour $s < 0$, $P(0) > 0$ donc P a une racine négative r_1 .

Pour que P ait deux autres racines réelles $r_3 > r_2 > r_1$ il faut (condition nécessaire et suffisante) que P' s'annule deux fois et que le discriminant Δ de P soit positif.

3.1.1 Première condition : P' a deux annulations :

$P'(X) = 3X^2 + 2\frac{s}{K}X + \frac{f_0}{K}$ a pour discriminant $\Delta' = 4\frac{1}{K^2}(s^2 - 3Kf_0)$ ce qui donne la condition

$$\boxed{s^2 > 3Kf_0} \quad (16)$$

3.1.2 Deuxième condition : $\Delta > 0$:

Pour $P = aX^3 + bX^2 + cX + d$ on a $\Delta = b^2c^2 + 18abcd - 27a^2d^2 - 4ac^3 - 4b^3d$ ce qui dans notre cas donne

$$\begin{aligned} \Delta &= \frac{1}{K^4}f_0^2s^2 - 18\frac{f_0^2F_0}{K^3} - 27\frac{F_0^2f_0^2}{K^2s^2} - 4\frac{f_0^3}{K^3} + 4\frac{F_0f_0s^2}{K^4} \\ &= s^2\frac{f_0(f_0 + 4F_0)}{K^4} - \frac{f_0^2(18F_0 + 4)}{K^3} - \frac{27F_0^2f_0^2}{K^2}\frac{1}{s^2} \\ &= \frac{f_0}{K^4s^2}[(f_0 + 4F_0)s^4 - Kf_0(18F_0 + 4)s^2 - 27K^2F_0^2f_0] \end{aligned}$$

On est revenu à étudier le signe du polynôme en s^2

$$D(s^2) = (f_0 + 4F_0)s^4 - Kf_0(18F_0 + 4)s^2 - 27K^2F_0^2f_0 \quad (17)$$

de discriminant d :

$$\begin{aligned} d &= \left(Kf_0(18F_0 + 4)\right)^2 + 108(f_0 + 4F_0)K^2F_0^2f_0 \\ &= K^2f_0(f_0(18F_0 + 4)^2 + 108(f_0 + 4F_0)F_0^2) > 0 \end{aligned}$$

On obtient donc la condition sur la positivité de Δ :

$$s^2 > K \frac{f_0(18F_0 + 4) + \sqrt{f_0(f_0(18F_0 + 4)^2 + 108(f_0 + 4F_0)F_0^2)}}{2(f_0 + 4F_0)} \quad (18)$$

3.1.3 Signe des racines au voisinage de $(0, 0, C_0)$

On sait déjà que $r_3 < 0$. Comme $r_1 r_2 r_3 < 0$, on remarque que r_2 et r_1 sont du même signe. De plus P' a un axe de symétrie $X = -\frac{s}{3K} > 0$ car $s < 0$ donc P atteint un minimum local (forcement négatif) en un point positif donc P a une racine positive.

On en déduit $r_1 > r_2 > 0$:

Sous les conditions (16) et (18), P a deux racines positives et une négative.

3.2 Au voisinage de $(0, \rho_\infty, 0)$

Autour de $(0, \rho_\infty, 0)$: Posons $(\mu, \rho, C) = (\mu, \rho_\infty + \epsilon, C)$. On a

$$\begin{cases} -s\mu' - K\mu'' = f(C)\rho_\infty - \mu\rho_\infty \\ C' = \frac{b\rho_\infty C}{s} \\ -s\epsilon' = F_0\mu \end{cases} \quad (19)$$

la deuxième ligne donne

$$C(y) = \Lambda \exp\left(\frac{b\rho_\infty}{s}y\right) \quad (20)$$

et la réunion de la première et la deuxième se traduit sur ϵ par :

$$s^2\epsilon'' + Ks\epsilon''' = f(C)F_0\rho_\infty + s\epsilon'\rho_\infty \quad (21)$$

est une EDO d'ordre trois en ϵ avec terme source $\frac{F_0f(C)}{Ks}\rho_\infty$ de polynôme caractéristique :

$$Q(X) = X^3 + \frac{s}{K}X^2 - \frac{\rho_\infty}{K}X \quad (22)$$

qui possède toujours trois racines : 0, une négative et une positive : $X = -\frac{1}{2K}(s \pm \sqrt{s^2 + 4\rho_\infty Ks})$
Sur μ on a :

$$-s\mu' - Ks\mu'' = f(C)\rho_\infty - \mu\rho_\infty \quad (23)$$

Dans le cas $f(C) = C$:

μ a pour polynôme caractéristique homogène $M(X) = X^2 + \frac{1}{K}X - \frac{\rho_\infty}{Ks}$ de racines :

$$r_{1,2} = -\frac{1}{2K}(1 \pm \sqrt{1 + 4\frac{\rho_\infty K}{s}})$$

donc $\mu_H = Ae^{r_1 y} + Be^{r_2 y}$ (On choisit $r_1 > r_2$).

En cherchant une solution particulière de la forme $\mu_p = M \exp(\frac{b\rho_\infty}{s}y)$ on obtient $M = -\frac{\Lambda}{b^2\rho_\infty K + b - 1}$

et donc $\mu = Ae^{r_1 y} + Be^{r_2 y} + Me^{\frac{b\rho_\infty}{s}y}$ et donc $\rho = \rho_\infty + \alpha e^{r_1 y} + \beta e^{r_2 y} + \Gamma \exp(\frac{b\rho_\infty}{s}y)$ pour $\Gamma = \frac{Ms}{b\rho_\infty}$

4 Schémas et Positivité

On a le modèle suivant (“KPP avec mémoire”) :

$$\begin{cases} \partial_t \mu = K \Delta \mu + C(\mu + \rho) - \mu \rho \\ \partial_t \rho = F_0 \mu \\ \partial_t C = -b \rho C \end{cases} \quad (24)$$

4.1 Pour l'équation différentielle ordinaire

Sans dépendance spatiale :

$$\begin{cases} \partial_t \mu = C(\mu + \rho) - \mu \rho \\ \partial_t \rho = F_0 \mu \\ \partial_t C = -b \rho C \end{cases} \quad (25)$$

4.1.1 Schéma semi-implicite I pour l'EDO

Soit le schéma semi-implicite I pour l'EDO :

$$\boxed{\begin{cases} \mu^{n+1} = \mu^n + \Delta t (C^n(\mu^{n+1} + \rho^{n+1}) - \mu^{n+1} \rho^n) \\ \rho^{n+1} = \rho^n + \Delta t (F_0 \mu^{n+1}) \\ C^{n+1} = C^n - \Delta t (b \rho^{n+1} C^{n+1}) \end{cases}} \quad (26)$$

Ce schéma donne :

$$\begin{cases} \mu^{n+1} (1 - \Delta t (C^n(1 + \Delta t F_0)) + \rho^n) = \mu^n + \Delta t C^n \rho^n \\ \rho^{n+1} = \rho^n + \Delta t (F_0 \mu^{n+1}) \\ C^{n+1} = C^n \frac{1}{1 + \Delta t b \rho^{n+1}} \end{cases}$$

Pour conserver la positivité il suffit que le terme $(1 - \Delta t (C^n(1 + \Delta t F_0)) + \rho^n)$ reste positif :

Par exemple :

$$\boxed{C^0 < \frac{1}{\Delta t (1 + F_0 \Delta t)}} \quad (27)$$

4.1.2 Schéma semi-implicite II pour l'EDO

Soit le schéma semi-implicite II pour l'EDO :

$$\boxed{\begin{cases} \mu^{n+1} = \mu^n + \Delta t (C^n(\mu^{n+1} + \rho^{n+1}) - \mu^n \rho^n) \\ \rho^{n+1} = \rho^n + \Delta t (F_0 \mu^{n+1}) \\ C^{n+1} = C^n - \Delta t (b \rho^{n+1} C^{n+1}) \end{cases}} \quad (28)$$

Ce schéma donne :

$$\begin{cases} \mu^{n+1} (1 - \Delta t (C^n(1 + \Delta t F_0))) = \mu^n + \Delta t \rho^n (C^n - \mu^n) \\ \rho^{n+1} = \rho^n + \Delta t (F_0 \mu^{n+1}) \\ C^{n+1} = C^n \frac{1}{1 + \Delta t b \rho^{n+1}} \end{cases}$$

Pour conserver la positivité il suffit que les terme $(1 - \Delta t (C^n(1 + \Delta t F_0)))$ et $\mu^n + \Delta t \rho^n (C^n - \mu^n)$ restent positif :

Par exemple :

$$\boxed{C^0 < \frac{1}{\Delta t (1 + F_0 \Delta t)}} \quad (29)$$

et

$$\boxed{\rho^n < \frac{1}{\Delta t}} \quad (30)$$

On obtient une condition de plus que le schéma semi-implicite I.

4.2 Pour l'équation aux dérivées partielles

4.2.1 Schéma semi-implicite I pour l'EDP

Soit le schéma semi-implicite I pour l'EDP :

$$\boxed{\begin{cases} \mu_i^{n+1} = \mu_i^n + K \Delta t \frac{\mu_{i+1}^{n+1} - 2\mu_i^{n+1} + \mu_{i-1}^{n+1}}{\Delta x^2} + \Delta t (C_i^n (\mu_i^{n+1} + \rho_i^{n+1}) - \mu_i^{n+1} \rho_i^n) \\ \rho_i^{n+1} = \rho_i^n + \Delta t (F_0 \mu_i^{n+1}) \\ C_i^{n+1} = C_i^n - \Delta t (b \rho_i^{n+1} C_i^{n+1}) \end{cases}} \quad (31)$$

Ce schéma donne :

$$\begin{cases} (1 + \frac{K \Delta t}{\Delta x^2} A - \Delta t (C^n (1 + \Delta t F_0)) + \rho^n) \mu^{n+1} = \mu^n + \Delta t C^n \rho^n \\ \rho^{n+1} = \rho^n + \Delta t (F_0 \mu^{n+1}) \\ C^{n+1} = C^n \frac{1}{1 + \Delta t b \rho^{n+1}} \end{cases}$$

où A est la matrice de $-\Delta$:

$$A = \begin{bmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{bmatrix} \quad (32)$$

A étant symétrique définie positive, afin de préserver la positivité, on obtient la même condition (suffisante) que pour l'EDO :

$$\boxed{C^0 < \frac{1}{\Delta t (1 + F_0 \Delta t)}} \quad (33)$$

5 Résolution numérique

5.1 Résolution de l'EDO

5.1.1 Résultat de la simulation de l'EDO

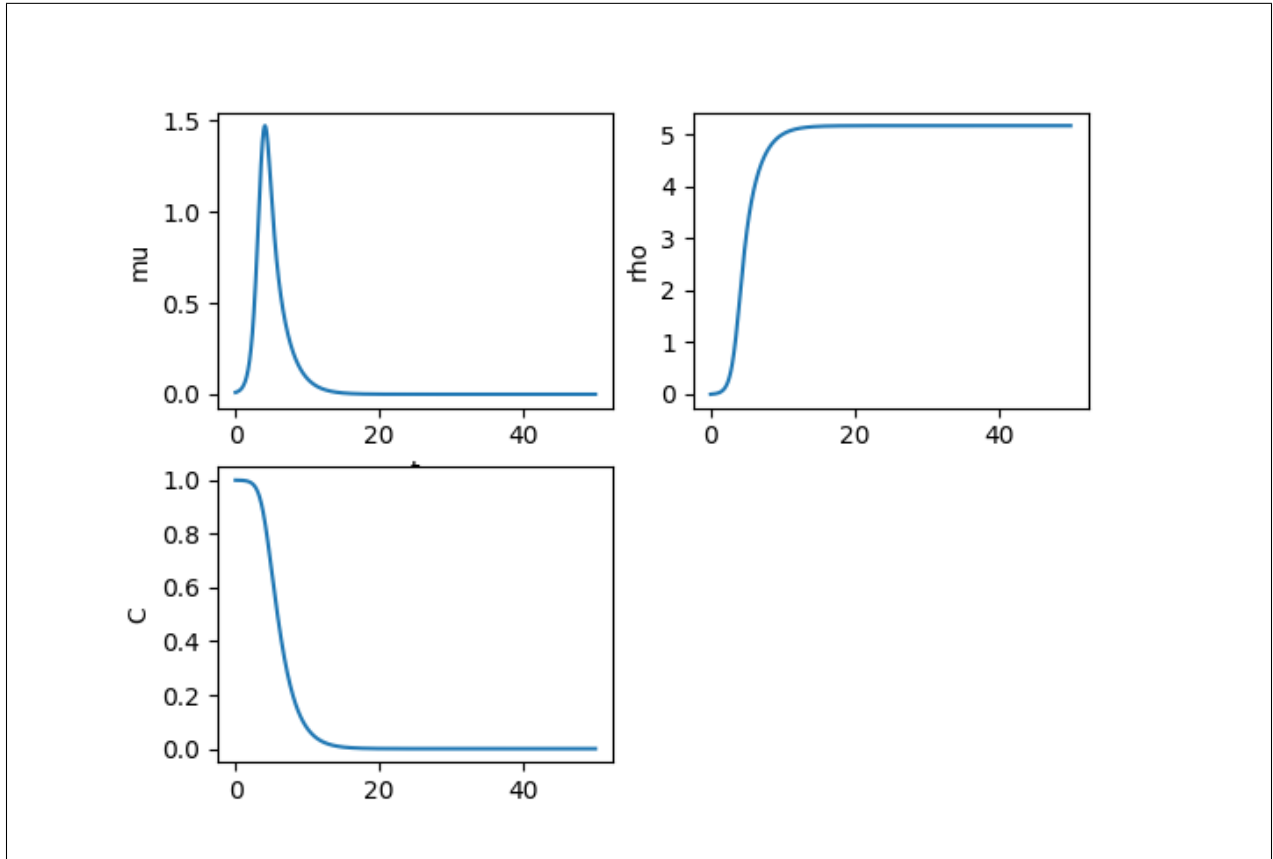


FIGURE 2 – Résolution du schéma implicite pour l'EDO

5.2 Résolution de l'EDP en 1D

5.2.1 Résultat de la simulation de l'EDP en 1D

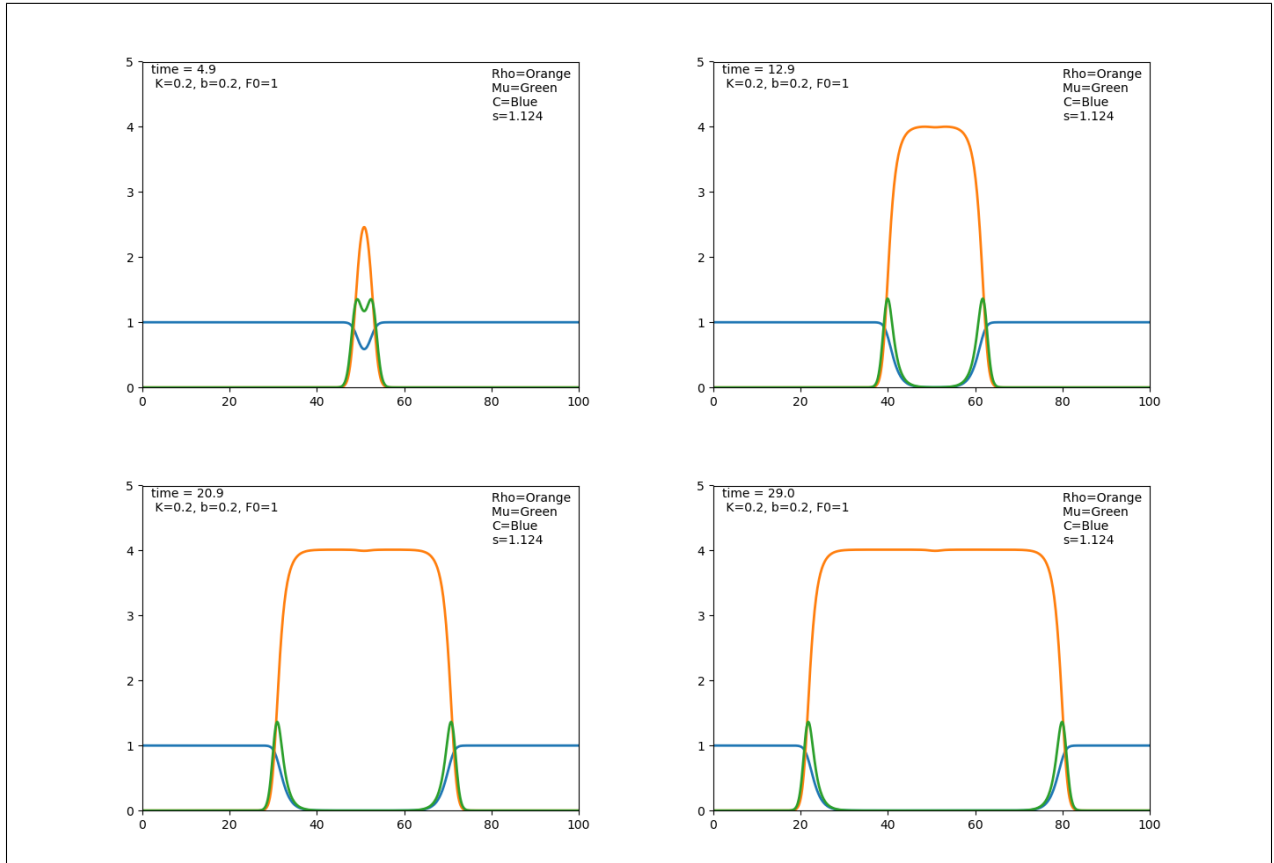


FIGURE 3 – Résolution du schéma semi implicite I pour l'EDP en 1D

Appendices

Code de résolution de l'EDO

```
1000 import matplotlib.pyplot as plt
1001 from scipy.integrate import ode
1002 import numpy as np

1004 b=.5 # dtC=-b*rho*C
1005 F0=1 # dtRho = Fo*Mu
1006 tf=50 # temps final de la simulation
1007 rho0=0 #rho initial
1008 mu0=.1 #mu initial
1009 c0=1 #concentration initiale
1010 n=1000 #nombre de pas de temps

1012 #Résolution du schéma explicite
1013 def euler_explicite_edo(b, F0, tf, rho0, mu0, c0, n):
1014     #t0<t1 , temps etudies ,
1015     #rho0, mu0,c0 reels positifs: condition initiale
1016     #n entier(nombre d'iterations)
1017     h=tf/n #pas Deltat
1018     rho=rho0
1019     mu=mu0
1020     c=c0
1021     t=0
1022     Rho=[rho0]
1023     Mu=[mu0]
1024     C=[c0]
1025     T=[t]
1026     for k in range(n):
1027         new_mu = mu + h*(c*(mu+rho)-mu*rho)
1028         new_rho = rho + h*F0*mu
1029         new_c = c - h*b*rho*c
1030         mu=new_mu
1031         rho=new_rho
1032         c=new_c
1033         t=t+h
1034         Mu.append(new_mu)
1035         Rho.append(new_rho)
1036         C.append(new_c)
1037         T.append(t)
1038     return T,Mu,Rho,C

1040 #Résolution du schéma semi- implicite I
1041 def euler_semi_I_edo(b, F0, tf, rho0, mu0, c0, n):
1042     #t0<t1 , temps etudies ,
1043     #rho0, mu0,c0 reels positifs: condition initiale
1044     #n entier(nombre d'iterations)
1045     h=tf/n #pas Deltat
1046     rho=rho0
1047     mu=mu0
1048     c=c0
1049     t=0
1050     Rho=[rho0]
1051     Mu=[mu0]
1052     C=[c0]
1053     T=[0]
```

```

1054     for k in range(n):
1055         new_mu = (mu + h*c*rho)/(1+h*rho-h*c*(1+h*F0))
1056         new_rho = rho + h*F0*new_mu
1057         new_c = c/(1 + b*h*new_rho)
1058         mu=new_mu
1059         rho=new_rho
1060         c=new_c
1061         t=t+h
1062         Mu.append(new_mu)
1063         Rho.append(new_rho)
1064         C.append(new_c)
1065         T.append(t)
1066     return T,Mu,Rho,C

1067 #Résolution du schéma semi- implicite II
1068 def euler_semi_II_edo(b, F0, tf, rho0, mu0, c0, n):
1069     #t0<t1 , temps etudies ,
1070     #rho0, mu0,c0 reels positifs: condition initiale
1071     #n entier(nombre d'iterations)
1072     t=0
1073     h=tf/n #pas Deltat
1074     rho=rho0
1075     mu=mu0
1076     c=c0
1077     Rho=[rho0]
1078     Mu=[mu0]
1079     C=[c0]
1080     T=[0]
1081     for k in range(n):
1082         new_mu = (mu + h*c*rho-h*rho*mu)/(1-h*c*(1+h*F0))
1083         new_rho = rho + h*F0*new_mu
1084         new_c = c/(1 + b*h*new_rho)
1085         mu=new_mu
1086         rho=new_rho
1087         c=new_c
1088         t=t+h
1089         Mu.append(new_mu)
1090         Rho.append(new_rho)
1091         C.append(new_c)
1092         T.append(t)
1093     return T,Mu,Rho,C

1094 #Programmation de la méthode de Newton-Raphson
1095 def newton(f, gradf, newton_steps, x0):
1096     x=x0
1097     for k in range(newton_steps):
1098         x=x-f(x)/gradf(x)
1099     return x

1100 #Résolution du schéma implicite
1101 def euler_implicite_edo(b, F0, tf, rho0, mu0, c0, n):
1102     #t0<t1 , temps étudiés ,
1103     #rho0, mu0,c0 reels positifs: conditions initiale
1104     #n entier(nombre d'itérations)
1105     t=0
1106     newton_steps=10 #nombre d'itérations de la méthode de Newton-Raphson pour le
1107     calcul implicite
1108     h=tf/n #pas deltat
1109     rho=rho0
1110     mu=mu0

```

```

1112 c=c0
1113 Rho=[rho0]
1114 Mu=[mu0]
1115 C=[c0]
1116 T=[0]
1117 for k in range(n):
1118     #Calcul de new_mu par methode de Newton Raphson
1119     #coefficients du polynome d'ordre 3 en new_mu
1120     alpha = -h**4*F0**2*b
1121     beta= -F0*h**2*(b+1+2*rho*b*h)
1122     gamma= -(1+b*h*rho)+b*h**2*F0*mu+h*(c*(1+h*F0)-rho*(1+b*h*rho))
1123     delta=(1+b*h*rho)*mu + h*c*rho
1124     def P(X):
1125         return alpha*X**3+beta*X**2+gamma*X+delta
1126     def gradP(X):
1127         return 3*alpha*X**2+2*beta*X+gamma
1128     new_mu=newton(P,gradP,newton_steps,mu)
1129     new_rho = rho + h*F0*new_mu
1130     new_c = c/(1 + b*h*new_rho)
1131     mu=new_mu
1132     rho=new_rho
1133     c=new_c
1134     t=t+h
1135     Mu.append(new_mu)
1136     Rho.append(new_rho)
1137     C.append(new_c)
1138     T.append(t)
1139 return T,Mu,Rho,C
1140
1141 #Utilisation des libraries python (scipy) pour résoudre l'EDO
1142 def black_box_edo(b, F0, tf, rho0, mu0, c0, n):
1143     def f(t,y,arg1,arg2):
1144         mu=y[0]
1145         rho=y[1]
1146         c=y[2]
1147         return [c*(mu+rho)-mu*rho, F0*mu, -b*rho*c]
1148
1149     r = ode(f).set_integrator('zvode', method='adams')
1150     r.set_initial_value([mu0,rho0,c0],0).set_f_params(F0,b)
1151     dt=tf/(n-1)
1152     Rho=[rho0]
1153     Mu=[mu0]
1154     C=[c0]
1155     t=0
1156     T=[0]
1157     while r.t < tf:
1158         mu,rho,c = r.integrate(r.t+dt)
1159         Mu.append(mu)
1160         Rho.append(rho)
1161         C.append(c)
1162         T.append(r.t)
1163     return T,Mu,Rho,C
1164
1165 T,Mu,Rho,C = black_box_edo(b, F0, tf, rho0, mu0, c0, n)
1166 rho_inf_theorique = .5*(rho0+np.sqrt(rho0**2 + 4*F0*(mu0+(c0/b))))
1167 print(rho_inf_theorique)
1168 rho_inf= Rho[n-1]
1169 print(rho_inf)

```



```

A=[np.log(rho_inf-x) for x in Rho]
1172 B=[-b*y*rho_inf+np.log((F0)/(b*rho_inf)) for y in T]
#Tracé des solutions
1174 plt.subplot(221)
plt.plot(T,Mu)
1176 plt.ylabel('mu')
plt.xlabel('t')
1178 plt.subplot(222)
plt.plot(T,Rho)
1180 plt.ylabel('rho')
plt.subplot(223)
1182 plt.plot(T,C)
plt.ylabel('C')
1184 plt.subplot(224)
plt.plot(T,A)
1186 plt.plot(T,B)
plt.ylabel('log(rho_inf-rho), -b*rho_inf*t')
1188 plt.show()

```

edo.py

Code de la résolution de l'EDP en 1D

```

1000 # %load edp_1d.py
import matplotlib.pyplot as plt
1002 import numpy as np
import scipy.sparse as sp
1004 from scipy.sparse.linalg.dsolve import spsolve
import matplotlib.animation as animation
1006
#Coefficients physiques
1008 K=.4 #coefficient diffusion
b=.5 # dtC=-b*rho*C
1010 F0=1 # dtRho = Fo*Mu

#Paramètres numériques
1012 n_t=1001 #nombre de pas de temps
1014 tf=25 # temps final de la simulation
xf = 100 #longueur de la simulation
1016 n_x =500 #nombres de points de la simulation

#Données initiales
1018 rho0=np.zeros(n_x) #rho initial
1020 mu0=np.zeros(n_x) #mu initial
mu0[(n_x//2):(n_x//2 +10)]=.01
1022 c0=np.zeros(n_x)+1 #concentration initiale

1024 def edp_1d_explicite(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x):
    dt=tf/(n_t-1)
1026    dx=xf/(n_x-1)
    X=np.linspace(0,xf,n_x)
1028    T=np.linspace(0,tf,n_t)
    Mu=np.zeros((n_t,n_x))
1030    Rho=np.zeros((n_t,n_x))
    C=np.zeros((n_t,n_x))
1032    Mu[0]=mu0
    Rho[0]=rho0
1034    C[0]=c0
    #Résolution du schema explicite
1036    for n in range(0,n_t-1):

```

```

1038     RHS=np.zeros(n_x)
1039     alpha=-C[n]*dt*(1+dt*F0)+dt*Rho[n]+1
1040     RHS[1:-1]= dt*((K/(dx**2))*(Mu[n,:-2]-2*Mu[n,1:-1]+Mu[n,2:])+C[n,1:-1]*Rho[n
1041 ,1:-1])
1042     RHS[0]= dt*((K/(dx**2))*(-2*Mu[n,0]+Mu[n,1])+C[n,0]*Rho[n,0])
1043     RHS[-1]=dt*((K/(dx**2))*(-2*Mu[n,-1]+Mu[n,-2])+C[n,-1]*Rho[n,-1])
1044     Mu[n+1]=(1/alpha)*(Mu[n]+RHS)
1045     Rho[n+1]=Rho[n]+dt*F0*Mu[n+1]
1046     C[n+1]=C[n]/(1 + b*dt*Rho[n])
1047     return X,T,Mu,Rho,C
1048
1049 def edp_1d_semi_implicite_I(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x):
1050     #Détermination des paramètres numériques deltat et deltax
1051     dt=tf/(n_t-1)
1052     dx=xf/(n_x-1)
1053     #Représentation de l'espace et du temps
1054     X=np.linspace(0,xf,n_x)
1055     T=np.linspace(0,tf,n_t)
1056     #Initialisation
1057     Mu=np.zeros((n_t,n_x))
1058     Rho=np.zeros((n_t,n_x))
1059     C=np.zeros((n_t,n_x))
1060     Mu[0]=mu0
1061     Rho[0]=rho0
1062     C[0]=c0
1063     #Résolution du schéma implicite-explicite I
1064     for n in range(0,n_t-1):
1065         #Matrice du Laplacien
1066         A=np.diag(-np.ones(n_x-1),-1)+np.diag(2*np.ones(n_x),0)+np.diag(-np.ones(n_x
1067 -1),1)
1068         #Laplacien Numerique
1069         A=A*K*dt/(dx**2)
1070         #Ajout des termes implicites
1071         alpha=-C[n]*dt*(1+dt*F0)+dt*Rho[n]+1
1072         A+=np.diag(alpha,0)
1073         A=sp.csc_matrix(A)
1074         #Résolution du système implicite
1075         Mu[n+1]= spsolve(A, Mu[n]+dt*C[n]*Rho[n])
1076         Rho[n+1]=Rho[n]+dt*F0*Mu[n+1]
1077         C[n+1]=C[n]/(1 + b*dt*Rho[n])
1078     return X,T,Mu,Rho,C
1079
1080 def edp_1d_semi_implicite_II(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x):
1081     #Détermination des paramètres numériques deltat et deltax
1082     dt=tf/(n_t-1)
1083     dx=xf/(n_x-1)
1084     #Représentation de l'espace et du temps
1085     X=np.linspace(0,xf,n_x)
1086     T=np.linspace(0,tf,n_t)
1087     #Initialisation
1088     Mu=np.zeros((n_t,n_x))
1089     Rho=np.zeros((n_t,n_x))
1090     C=np.zeros((n_t,n_x))
1091     Mu[0]=mu0
1092     Rho[0]=rho0
1093     C[0]=c0
1094     #Résolution du schéma implicite-explicite II
1095     for n in range(0,n_t-1):

```

```

1094     #Matrice du Laplacien
    A=np.diag(-np.ones(n_x-1),-1)+np.diag(2*np.ones(n_x),0)+np.diag(-np.ones(n_x
1096     -1),1)
    A=A*K*dt/(dx**2) #Laplacien Numerique
    #Ajout des termes implicites
1098     alpha=-C[n]*dt*(1+dt*F0)+1
    A+=np.diag(alpha,0)
1100     A= sp.csc_matrix(A)
    #Résolution du système implicite
1102     Mu[n+1]= spsolve(A, Mu[n]+dt*C[n]*Rho[n]-dt*Mu[n]*Rho[n])
    Rho[n+1]=Rho[n]+dt*F0*Mu[n+1]
1104     C[n+1]=C[n]/(1 + b*dt*Rho[n])
    return X,T,Mu,Rho,C
1106
X,T,Mu,Rho,C= edp_1d_semi_implicite_I(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x)
1108
#Valeur de rho a l'infini
1110 rho_inf = Rho[n_t-1,(n_x//2)]
print(rho_inf)
1112

1114 def speed(X,Rho):
    #Position du front
1116     argmed=np.zeros(n_t)
    for i in range(n_t):
1118         argmed[i]= X[(n_x//2)+np.min(np.where(np.append(Rho[i,(n_x//2):],[0])<
            rho_inf/2))]
    #Vitesse du front
1120     s = ((n_t-1)/tf)*(argmed[-1]-argmed[(n_t//2)])/(n_t//2)
    return s
1122 s=0
s = speed(X,Rho)
1124 print('La vitesse de propagation de la simulation est s=',s)
s_theorique = np.sqrt(K*((18*F0+4)+np.sqrt(((18*F0+4)**2)+108*(1+4*F0)*(F0**2))))
    /(2*(1+4*F0))
1126 #Attention, ceci est pour C0=1
print('La vitesse théorique de propagation est s_theorique=', s_theorique)
1128
#Animation
1130 fig = plt.figure()

1132 ax = plt.axes(xlim=(0, xf), ylim=(0, rho_inf+1))
line, = ax.plot([], [], lw=2)
1134 line2, = ax.plot([], [], lw=2)
line3, = ax.plot([], [], lw=2)
1136 line4, = ax.plot([], [], lw=2)
time_text = ax.text(0.02, 0.92, '', transform=ax.transAxes)
1138 legend_text = ax.text(0.80, 0.82, '', transform=ax.transAxes)

1140 def init():
    line.set_data([], [])
1142     line2.set_data([], [])
    line3.set_data([], [])
1144     line4.set_data([], [])
    time_text.set_text('')
1146     legend_text.set_text('')
    return line,line2,line3,line4, time_text, legend_text
1148

```

```

1150 def animate(i):
    line.set_data(X, C[i])
1152 line2.set_data(X, Rho[i])
    line3.set_data(X, Mu[i])
1154 line4.set_data(50+((i*s)*tf/(n_t-1)),np.linspace(0,rho_inf+1,10))
    time_text.set_text('time = {0:.1f}\n K={1}, b={2}, F0={3} '.format(T[i],K,b,F0))
1156 legend_text.set_text('Rho=Orange \nMu=Green \nC=Blue\ns={0:.3f} '.format(s))
    return line, line2, line3, line4, time_text, legend_text
1158

1160 anim = animation.FuncAnimation(fig, animate, init_func=init,
                                frames=(n_t-1), interval=(tf*200)/(n_t-1), blit=True)
1162
1164 #anim.save('EDP_1D.gif',writer='imagemagick', fps=30)
1166 plt.show()

```

edp_1d.py