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 d'Yves D'Angelo, Rémi Catellier et Laurent Monasse.

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

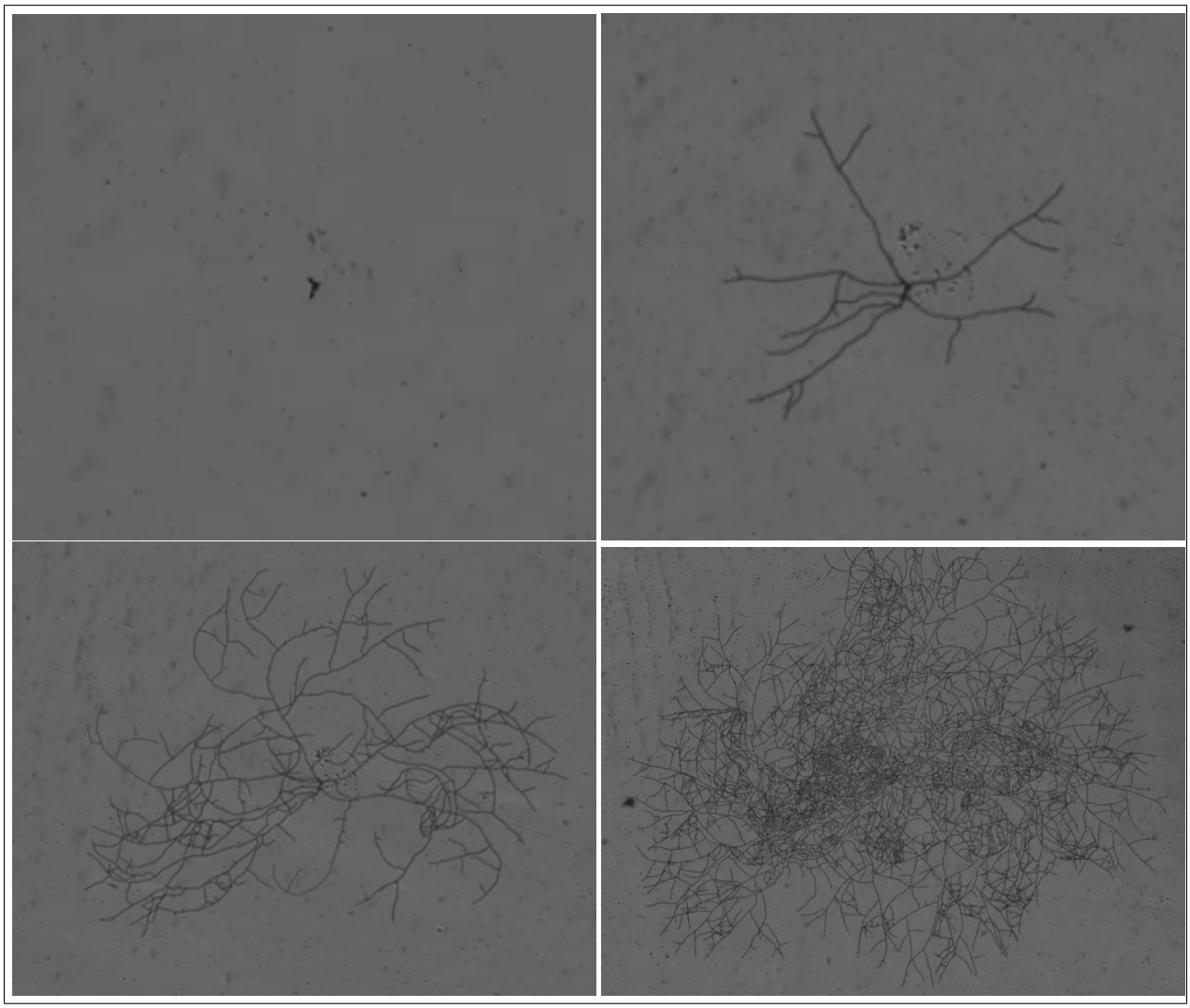

FIGURE 1 – Capture d'un réseau de champignon en expansion,
par Éric Herbert, Gwenaël Ruprich-Robert et Florence Leclerc.

Table des matières

1 L'équation de Fisher ou KPP

1.1 Préliminaire

Notre point de départ est l'équation de diffusion :

$$\partial_t u = \Delta u \tag{1}$$

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 réaction-diffusion :

$$\partial_t u = \Delta u + F(u) \tag{2}$$

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 étés introduits et examinés par les travaux de Fisher[1] et Kolmogorov, Petrovsky et Piscounuv (abrégés KPP).
Un exemple d'une telle équation est :

$$\partial_t u = \Delta u + ru(1-u) \tag{3}$$

où $r > 0$, 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) suivante est obtenue :

$$\partial_t v = r(v - v^2) = F(v) \tag{4}$$

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 $X = C^0_{b,unif}(\mathbb{R}, \mathbb{R})$ 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 de Fisher-KPP 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 d'ondes plane stationnaire / onde progressive

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

Définition 1.1. Solutions en onde plane stationnaires.
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 :

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

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. Existence de solutions en onde progressive pour les équations de réaction-diffusion :
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 $h_s : \mathbb{R} \to]0,1[$ de classe C^3.
Cette solution est unique à translation près.
- ii) $\forall s < s_*$ l'équation (5) n'a pas de solution $h : \mathbb{R} \to [0,1]$

Remarques :
Dans le cas ii) il existe des solutions en ondes planes 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.
Dans le cas de l'équation de Fisher-KPP, c'est à dire pour $F(u) = r(u - u^2)$, on a $s_*^2 = 4F'(0) = 4r$: la vitesse minimale de propagation est $s^* = 2\sqrt{r}$.

1.5 Dans l'exemple de Fisher-KPP

Considérons l'équation de Fisher-KPP (3) : $\partial_t u = \Delta u + ru(1-u)$.
Comme $u \equiv 0$ et $u \equiv 1$ sont des solutions particulières de (3), si $0 \leq u_0(x) \leq 1 \ \forall x$, alors par le principe du maximum on a $0 \leq u(x,t) \leq 1 \ \forall x, t$.
Soit h une solution en onde plane de (5) avec $0 \leq h \leq 1 \ \forall y$, i.e. $h''(y) + sh'(y) + rh(y) - rh^2(y) = 0$.
En linéarisant autour de l'état $h = 0$ on obtient :

$$h''(y) + sh'(y) + rh(y) = 0 \tag{6}$$

de polynôme caractéristique $X^2 + sX + r = 0$ et de discriminant $\Delta = s^2 - 4r$.
On voit alors que la condition $s^2 \geq 4r$ est nécessaire pour que $0 \leq h \leq 1$: c'est la condition d'amortissement fort de l'oscillateur autour de l'état $h = 0$.

1.6 Théorèmes de sélection de la vitesse pour KPP

Le théorème important suivant est du aux travaux de Kolmogorov, Petrovsky et Piscounuv de 1937. C'est l'article et le résultat fondateur de la théorie des ondes planes dans les systèmes de réaction-diffusion.

Théorème 4. Convergence vers une solution d'onde à vitesse minimale pour les solutions de l'équation de Fisher-KPP avec une donnée initiale à support compact
Soit $u_0 \to]0,1[$ une donnée initiale à support compact. Soit u la solution de l'équation de Fisher-KPP (3) avec $r = 1$ et de donnée initiale u_0. Alors quand $t \to \infty$, u converge uniformément en x vers une solution d'onde h_{s^*} de (5) qui se de déplace à vitesse minimale $s^* = 2$:

$$\sup_{y \in \mathbb{R}} |u(y + m(t), t) - h_{s^*}(y)| \to_{t \to \infty} 0$$

où $m(t) = 2t - (3/2)\log(t) + y_0$.

Remarque : La vitesse du front est alors $s(t) = \partial_t m(t) = 2 - \frac{3}{2t} \to_{t \to \infty} 2$.

Ce résultat a été raffiné par la suite par Uchiyama, Bramson et Lau. Leurs travaux apportent plus d'informations sur comment la vitesse du front se sélectionne en fonction de la donnée initiale, et comment il est possible d'obtenir d'autres vitesses de fronts que la vitesse minimale en fonction de la donnée initiale.

Théorème 5. Sélection de la vitesse pour les solutions de l'équation de Fisher-KPP en fonction de la donnée initiale
Si $u_0 \to]0,1[$ vérifie $\liminf_{x \to -\infty} u_0(x) > 0$ et $\int_0^{+\infty} xe^x u_0(x)/dx < \infty$
alors il existe $y_0 \in \mathbb{R}$ tel que la solution de (3) avec données initiales u_0 vérifie

$$\sup_{y \in \mathbb{R}} |u(y + m(t), t) - h_{s^*}(y)| \to_{t \to \infty} 0$$

où $m(t) = 2t - (3/2)\log(t) + y_0$.

D'autres vitesses peuvent être sélectionnées : Si la donnée initiale vérifie $u_0(x) \approx e^{-\lambda_-(s)x}$ quand $x \to +\infty$, où $\lambda_-(s)$ est la plus petite racine du polynôme caractéristique $X^2 + sX + r = 0$, alors la solution converge vers une onde progressive de vitesse s.

2 Dynamique de Réseaux en Croissance

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

$$\begin{cases} \partial_t \mu + \nabla(\mu v) = f(C)(\mu + \rho) - \mu\rho \\ \partial_t(\mu v) + \nabla(\mu v \times v) + T\nabla\mu = -\lambda\mu v + \mu\nabla C - \mu v\rho \\ \partial_t \rho = F(v)\mu \\ \partial_t C = -b\rho C \end{cases} \tag{7}$$

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 représentants 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 :

$$F(V) = (\frac{1}{2\pi T})^{\frac{d}{2}} \int_{\mathbb{R}^d} |v| \exp(-\frac{|v-V|^2}{2T}) dv \tag{8}$$

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 (7)

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 (7) 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 a 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 due 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 hyphes 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 (7) donne :

$$+K\nabla\mu = -\mu v \tag{9}$$

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

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

dit "KPP avec mémoire".

2.3 Propriétés de l'équation de réaction associée à "KPP avec mémoire"

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

$$\begin{cases} \partial_t \mu = f(C)(\mu+\rho) - \mu\rho \\ \partial_t \rho = F_0\mu \\ \partial_t C = -b\rho C \end{cases} \tag{11}$$

avec $f(0) = 0$.
Ce système correspond au systême "KPP avec mémoire" sans le terme de diffusion. On s'intéresse au comportement de (μ, ρ, C) sur $\mathbb{R}^+$:

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 $(\mu(0), \rho(0)) > (0,0)$ (les deux positifs, au moins un non nul), $C(0) > 0$.
Alors $\mu(t) \geq 0\ \forall t > 0$.

Démonstration. Supposons par l'absurde que μ devient négatif alors soit $t^* = \inf(t > 0/\mu(t) < 0)$.
Alors :
$\mu(t) \geq 0\ \forall t \leq t^*$
$\partial_t\mu(t^*) \leq 0$ par définition de t^*. (Sinon $\mu(t^* + \epsilon) > 0\ \forall \epsilon << 1$)
$\rho(t) > 0\ \forall t \leq t^*$ 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. □

Dans la suite on se place dans le cas où $(\mu(0), \rho(0)) > (0,0)$, $C(0) > 0$:

Lemme 3. ρ est croissante car $\partial_t\rho = F_0\mu \geq 0$. En particulier ρ est positive.

Lemme 4. C est décroissante et $\lim_{t\to+\infty} C(t) = 0$

Démonstration. ρ est positive donc C est décroissante.
$(\mu(0), \rho(0)) > (0,0)$ et $\partial_t\rho = F_0\mu$ impliquent qu'il existe un t_0 tel que $\rho(t_0) > 0$.
Comme ρ est croissante $\forall t \geq t_0$, $\rho(t) \geq \rho(t_0)$.
Donc $\forall t \geq t_0$, $0 < C(t) = C(0)\exp(-b\int_0^t \rho(s)ds) \leq C_{ste}e^{-b\rho(t_0)t} \underset{t\to+\infty}{\longrightarrow} 0$
Donc $\lim_{t\to+\infty} C(t) = 0$. □

Lemme 5. Si f est croissante et $\int_0^1 \frac{f(x)}{x}dx < \infty$ alors μ est bornée.

Démonstration. On a $\partial_t \mu = f(C)(\mu+\rho) - \mu\rho \leq f(C)\mu + f(C)\rho$.
Montrons que $f(C)$ est intégrable :
$C(t) \leq C_{ste}e^{-b\rho(t_0)t}$ et f est croissante donc $\int_0^\infty f(C)dt \leq \int_0^\infty f(C_{ste}e^{-b\rho(t_0)t})dt$.
Soit le changement de variable $u = C_{ste}e^{-b\rho(t_0)t}$, $du = -b\rho(t_0)u\ dt$:
$\int_0^\infty f(C_{ste}e^{-b\rho(t_0)t})dt = \frac{1}{b\rho(t_0)}\int_0^1 \frac{f(u)}{u}du < \infty$ car $\int_0^{C_{ste}} \frac{f(x)}{x}dx < \infty$ donc $f(C)$ est intégrable.
Montrons que $\phi = f(C)\rho$ est intégrable :
Effectuons le changement de variable $u = C$, $du = -b\rho u\ dt$ dans $\int_0^\infty f(C)\rho\ dt$:
$\int_0^\infty f(C)\rho\ dt = \frac{1}{b\rho(t_0)}\int_0^{C_{ste}} \frac{f(u)}{u}du < \infty$ car $\int_0^{C_{ste}} \frac{f(x)}{x}dx < \infty$ donc $\phi = f(C)\rho$ est intégrable.
On a $\partial_t \mu \leq f(C)\mu + \phi$.
Par le lemme de Gronwall :
$\mu(t) \leq \mu(0) + \int_0^t \phi(s)\ ds + \int_0^t \phi(s)f(C)(s)\exp(\int_s^t f(C)(u)du)\ ds$
$\leq \mu(0) + \int_0^{+\infty} \phi(s)\ ds + \int_0^t \phi(s)f(C)(s)\exp(\int_0^{+\infty} f(C)(u)du)\ ds$
$\leq \mu(0) + \int_0^{+\infty} \phi(s)\ ds + \exp(\int_0^{+\infty} f(C)(u)du)\int_0^t \phi(s)f(C)(s)\ ds$
$f(C)$ est bornée et ϕ est intégrable donc $f(C)\phi$ est intégrable.
On a donc : $\mu(t) \leq \mu(0) + \int_0^{+\infty} \phi(s)\ ds + \exp(\int_0^{+\infty} f(C)(u)du)\int_0^{+\infty} \phi(s)f(C)(s)\ ds\ \forall t$ □

Dans la suite on se place dans le cas où f est croissante et $\int_0^1 \frac{f(x)}{x}dx < \infty$

Lemme 6. $\lim_{t\to+\infty} \mu = 0$ et $\lim_{t\to+\infty} \rho = \rho_\infty < +\infty$

Démonstration. μ est bornée, soit μ_n une suite extraite de la fonction μ qui tend vers ℓ.
On a $\ell - \mu(t) = \lim_{n\to+\infty} \int_t^{t_n} \partial_t\mu\ ds = \lim_{n\to+\infty} \int_t^{t_n} f(C)(\mu+\rho) - \mu\rho\ ds$.
Or $f(C)$ est intégrable (c.f. preuve du lemme 5) et μ est bornée donc $f(C)\mu$ est intégrable.
De même $f(C)\rho$ est intégrable (c.f. preuve du lemme 5).
On a donc $\lim_{n\to+\infty} \int_t^{t_n} \mu\rho = \int_t^{+\infty}(f(C)\mu + f(C)\rho)\ dt\ + \ell - \mu(t)$
Or $\mu\rho = F_0\rho\partial_t\rho = \frac{F_0}{2}\partial_t\rho^2$ donc $\int_t^{t_n} \mu\rho\ dt = \frac{F_0}{2}(\rho(t_n)^2 - \rho(t)^2)$.
Or ρ est croissante donc a une limite dans $[0, +\infty]$.
Ainsi ℓ est déterminée entièrement par la limite de ρ et ne dépend pas de la suite extraite.
Par critère séquentiel μ a une limite ℓ qui est finie car μ est bornée.
Mais alors $\lim_{t\to+\infty} \frac{F_0}{2}(\rho(t)^2 - \rho(T)^2) = \int_T^\infty (f(C)\mu + f(C)\rho)\ dt\ + \ell - \mu(T) < \infty$.
Donc ρ^2 a une limite finie et donc ρ aussi.
Comme $\mu = \frac{\partial_t\rho}{F_0}$ et ρ a une limite finie et μ aussi, μ tend nécessairement vers 0. □

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

On a le modèle suivant :

$$\begin{cases} \partial_t \mu - K\Delta\mu = f(C)(\mu+\rho) - \mu\rho \\ \partial_t \rho = F_0 \mu \\ \partial_t C = -b\rho C \end{cases} \tag{12}$$

où $f(0) = 0$ et f est positive. Typiquement, $f(C) = C$:
On recherche des solutions en onde plane, on pose s la vitesse d'onde et $\xi = x - st$.
Par abus de notation, on pose $\mu(\xi) = \mu(x,t)$, $\rho(\xi) = \rho(x,t)$, etc...
On a alors :

$$\begin{cases} -s\mu' - K\mu'' = f(C)(\mu+\rho) - \mu\rho \\ -s\rho' = F_0\mu \\ C' = \frac{b\rho C}{s} \end{cases} \tag{13}$$

Nos états stationnaires (i.e. qui correspondent à des derivées nulles) sont :

$$(\mu,\rho,C) = \begin{cases} (0,0,C_0) \\ (0,\rho_\infty,0), \rho_\infty > 0 \end{cases} \tag{14}$$

3.1 Linéarisation au voisinage de $(0,0,C_0)$

Au voisinage de $(0,0,C_0)$ on a, en posant $f(C_0) = f_0$, la linéarisation de 12 :

$$\begin{cases} -s\mu' - K\mu'' = f_0(\mu+\rho) \\ -s\rho' = F_0\mu \end{cases} \tag{15}$$

ce qui devient :

$$\rho''' + \frac{s}{K}\rho'' + \frac{f_0}{K}\rho' - \frac{F_0 f_0}{Ks}\rho = 0 \tag{16}$$

de polynôme caractéristique :

$$P(X) = X^3 + \frac{s}{K}X^2 + \frac{f_0}{K}X - \frac{F_0 f_0}{Ks}. \tag{17}$$

Le signe de s correspondant à la direction de propagation, il y'a symétrie en s : on prend içi $s < 0$ ce qui correspond a une propagation vers la gauche.
Pour $s < 0$, P est de degŕe 3 et $P(0) > 0$ donc P a une racine négative r_1 .
Pour conserver la positivité autour de l'état $(0,0,C_0)$ il faut que les racines de P soit réelles : sinon on obtient osillations autour de 0.
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

$$\boxed{s^2 > 3Kf_0.} \tag{18}$$

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 :

$$D(s^2) = (f_0+4F_0)s^4 - Kf_0(18F_0+4)s^2 - 27K^2F_0^2f_0 \tag{19}$$

de discriminant d :

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

On obtient donc la condition sur la positivité de Δ :

$$\boxed{s^2 > K\frac{f_0(18F_0+4) + \sqrt{f_0(f_0(18F_0+4)^2 + 108(f_0+4F_0)F_0^2)}}{2(f_0+4F_0)}.} \tag{20}$$

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

On sait déjà que $r_3 < 0$. Comme $r_1r_2r_3 < 0$, 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 (18) et (20), P a deux racines positives et une négative.

Conclusion Comme pour l'équation de KPP, la linéarisation autour de l'état $(0, 0, C_0)$ fait apparaitre une condition sur s nécessaire pour preserver la positivité.

3.2 Linéarisation au voisinage de $(0, \rho_\infty, 0)$

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

$$\begin{cases} -s\mu' - K\mu'' = f(C)\rho_\infty - \mu\rho_\infty \\ C' = \frac{b\rho_\infty C}{s} \\ -s\epsilon' = F_0\mu. \end{cases} \tag{21}$$

La deuxième ligne donne

$$C(y) = \Lambda\exp(\frac{b\rho_\infty}{s}y) \tag{22}$$

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

$$s^2\epsilon'' + Ks\epsilon''' = f(C)F_0\rho_\infty + s\epsilon'\rho_\infty \tag{23}$$

qui est une EDO d'ordre trois en ϵ avec terme source $\frac{F_0 f(C)}{Ks}\rho_\infty$ de polynôme caracteristique :

$$Q(X) = X^3 + \frac{s}{K}X^2 - \frac{\rho_\infty}{K}X \tag{24}$$

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

$$-s\mu' - Ks\mu'' = f(C)\rho_\infty - \mu\rho_\infty. \tag{25}$$

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 :
$r_{+,-} = -\frac{1}{2K}(1 \pm \sqrt{1 + 4\frac{\rho_\infty K}{s}})$
donc $\mu_H = Ae^{r_+ y} + Be^{r_- y}$ (On choisit $r_+ > 0, r_- < 0$).
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 $\mu = Ae^{r_+ y} + Be^{r_- y} + Me^{\frac{b\rho_\infty}{s}y}$ et donc $\rho = \rho_\infty + \alpha e^{r_+ y} + \beta e^{r_- y} + \frac{Ms}{b\rho_\infty}\exp(\frac{b\rho_\infty}{s}y)$.

Conclusion Comme pour l'équation de KPP, on obtient à priori pas de condition sur s suite à la linéarisation autour de l'état $(0, \rho_\infty, 0)$ mais seulement des informations sur la dynamique autour de ces états.

4 Schémas Numériques

On a le modèle suivant ("KPP avec mémoire") :

$$\begin{cases} \partial_t \mu = K\Delta\mu + C(\mu+\rho) - \mu\rho \\ \partial_t \rho = F_0 \mu \\ \partial_t C = -b\rho C \end{cases} \tag{26}$$

que l'on souhaite simuler par différences finies.
Il y a plusieurs difficultés : capture de fronts raides, exigence de positivité, difficulté calculatoire du schéma entièrement implicite... C'est pour cela que l'on choisira un schéma semi-implicite.

4.1 Pour l'équation différentielle ordinaire

Sans dépendance spatiale (équation de réaction) :

$$\begin{cases} \partial_t \mu = C(\mu+\rho) - \mu\rho \\ \partial_t \rho = F_0 \mu \\ \partial_t C = -b\rho C. \end{cases} \tag{27}$$

4.1.1 Schéma semi-implicite I pour l'EDO

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

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

Ce schéma peut se résoudre efficacement avec la reformulation suivante :

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

Positivité du schéma 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 :

$$\boxed{\Delta t(1+F_0\Delta t) < \frac{1}{C_0}}. \tag{29}$$

4.1.2 Schéma semi-implicite II pour l'EDO

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

$$\boxed{\begin{cases} \mu^{n+1} = \mu^n + \Delta t(C^n(\mu^{n+1}+\rho^{n+1}) - \mu^n\rho^n) \\ \rho^{n+1} = \rho^n + \Delta t(F_0\mu^{n+1}) \\ C^{n+1} = C^n - \Delta t(b\rho^{n+1}C^{n+1}). \end{cases}} \tag{30}$$

Ce schéma peut se résoudre efficacement avec la reformulation suivante :

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

Positivité du schéma 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 :

$$\boxed{C^0 < \frac{1}{\Delta t(1+F_0\Delta t)}} \tag{31}$$

et

$$\boxed{\rho^n < \frac{1}{\Delta t}}. \tag{32}$$

En explicitant un terme de plus que le schéma I, on obtient une condition de plus sur la positivité que le schéma semi-implicite I, condition qui dépend en plus du temps ! La schéma I étant déjà relativement aussi difficile à inverser, on choisira dans la suite de simuler le schéma I.

4.2 Pour l'équation aux dérivées partielles

4.2.1 Schéma semi-implicite I pour l'EDP en 1D

Soit le schéma semi-implicite I pour l'EDP en 1D :

$$\boxed{\begin{cases} \mu_i^{n+1} = \mu_i^n + K\Delta t \frac{\mu_{i+1}^{n+1} - 2\mu_i^{n+1} + \mu_{i-1}^{n+1}}{\Delta x^2} + \Delta t(C_i^n(\mu_i^{n+1} + \rho_i^{n+1}) - \mu_i^{n+1}\rho_i^n) \\ \rho_i^{n+1} = \rho_i^n + \Delta t(F_0 \mu_i^{n+1}) \\ C_i^{n+1} = C_i^n - \Delta t(b\rho_i^{n+1} C_i^{n+1}) \end{cases}} \tag{33}$$

Ce schéma a été construit pour donner une équation linéaire en μ^{n+1} :

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

où A est la matrice de discrétisation par différences finies de $-\Delta$ en 1D :

$$A = \begin{bmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{bmatrix} \tag{34}$$

Remarque : pour le schéma en dimension $n \geq 2$, il suffit de remplacer A par la matrice de discrétisation par différences finies de $-\Delta$ en dimension n.

Positivité du schéma : Afin de préserver la positivité, on obtient la même condition (suffisante) que pour l'EDO :

$$\boxed{C^0 < \frac{1}{\Delta t(1+F_0\Delta t)}} \tag{35}$$

Démonstration. Supposons $\mu^0 > 0$.
Raisonnons par l'absurde et supposons que $n = \min n \mid \exists j \mid \mu_j^{n+1} < 0$ existe. Soit $j = \arg\min \mu_i^{n+1}$.

On a $(1 - \Delta t(C_j^n(1+\Delta t F_0)) + \rho_j^n)\mu_j^{n+1} = \mu^n + \Delta t C^n + \frac{K\Delta t}{\Delta x^2}(\mu_{j+1}^{n+1} - 2\mu_j^{n+1} + \mu_{j-1}^{n+1})$.
Or par définition de n et comme $C^0 < \frac{1}{\Delta t(1+F_0\Delta t)}$ et $C^n < C^0$:

Donc $\mu^n + \Delta t C^n > 0$ et $1 - \Delta t(C_j^n(1 + \Delta t F_0)) + \rho_j^n > 0$.
Et par définition de j : $\mu_{j+1}^{n+1} - 2\mu_j^{n+1} + \mu_{j-1}^{n+1} \geq 0$.
On a donc $\mu_j^{n+1} > 0$ mais $\mu_j^{n+1} = \min(\mu_i^{n+1}) < 0$ par définition de j et n : Contradiction. $\square$

5 Résolution numérique

5.1 Résolution de l'EDO

5.1.1 Résultat de la simulation de l'EDO

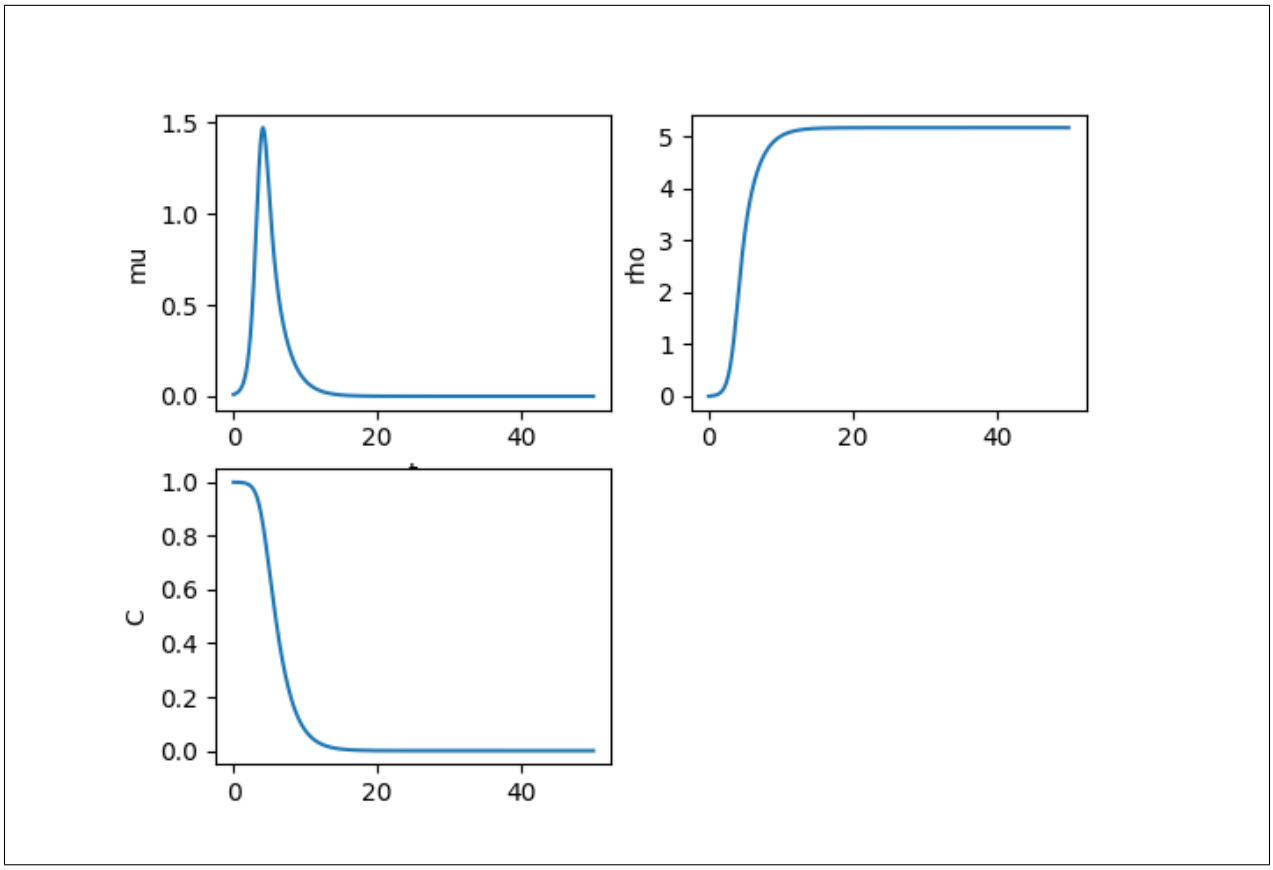

FIGURE 2 – Résolution du schéma semi-implicite I pour l'EDO (équation de réaction)

5.1.2 Observations de la simulation de l'EDO

On observe les phénomènes attendus sur l'EDO :
-i) μ est bornée et tend vers 0.
-ii) ρ est croissante et bornée.
-iii) C décroît vers 0.

-iv) Les solutions ont un comportement exponentiel autour des états stationnaires et ce comportement est bien prédit par la linéarisation de l'EDO autour de ces états :

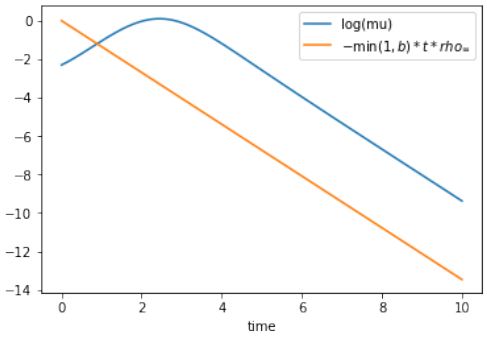

FIGURE 3 – Comportement de $\log(\mu)$, en particulier autour de $(\mu, \rho, C) = (0, \rho_\infty, 0)$. $\log(\mu)$ est bien linéaire autour des états stationnaires et sa pente (le facteur dans l'exponentielle) correspond exactement à $-\min(1, b)\rho_\infty$ ce qui est un résultat obtenu dans la partie Linéarisation

-v) L'ordre de convergence de l'EDO observé est de 1 :

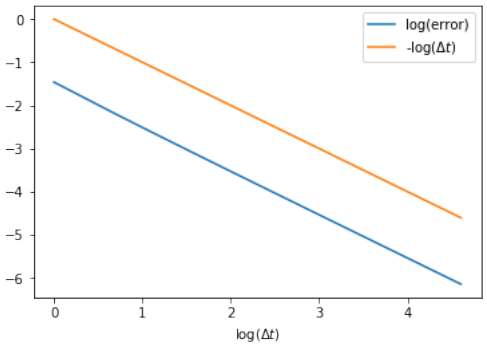

FIGURE 4 – Comportement de l'erreur en norme infinie quand $\Delta t \to 0$. La solution du schéma semi implicite I est comparée à différents pas de temps Δt à la solution numérique du schéma de Runge-Kutta à l'ordre 4 (RK4) avec $\Delta t = 2 * 10^{-7}$. On observe bien que la convergence est d'ordre 1.

5.2 Résolution de l'EDP en 1D

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

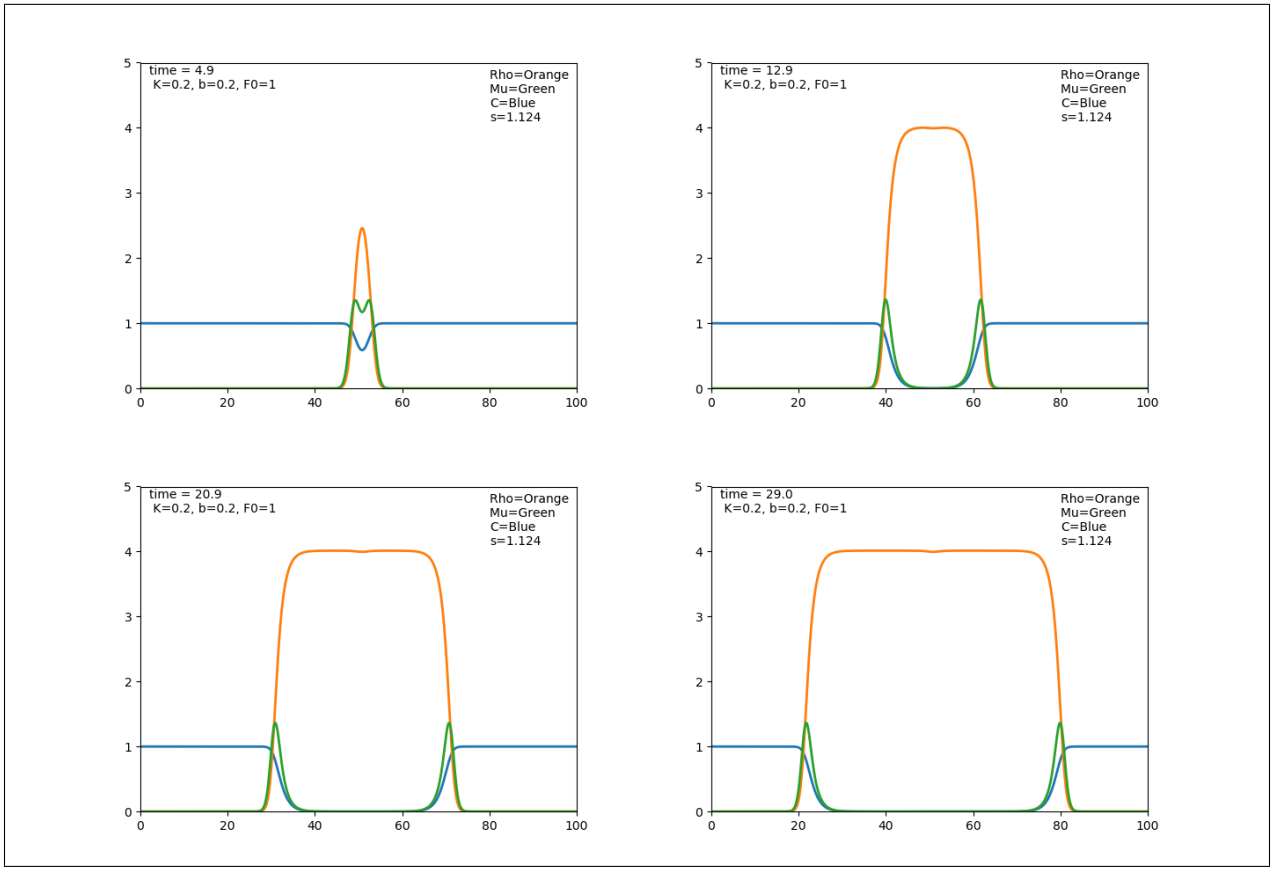

FIGURE 5 – Résolution du schéma semi implicite I pour l'EDP en 1D

On voit sur les simulations que la solution tend vers une solution de type onde plane stationnaire. Il est possible de calculer cette vitesse et de la comparer avec la vitesse théorique minimale obtenue dans la partie 3 :

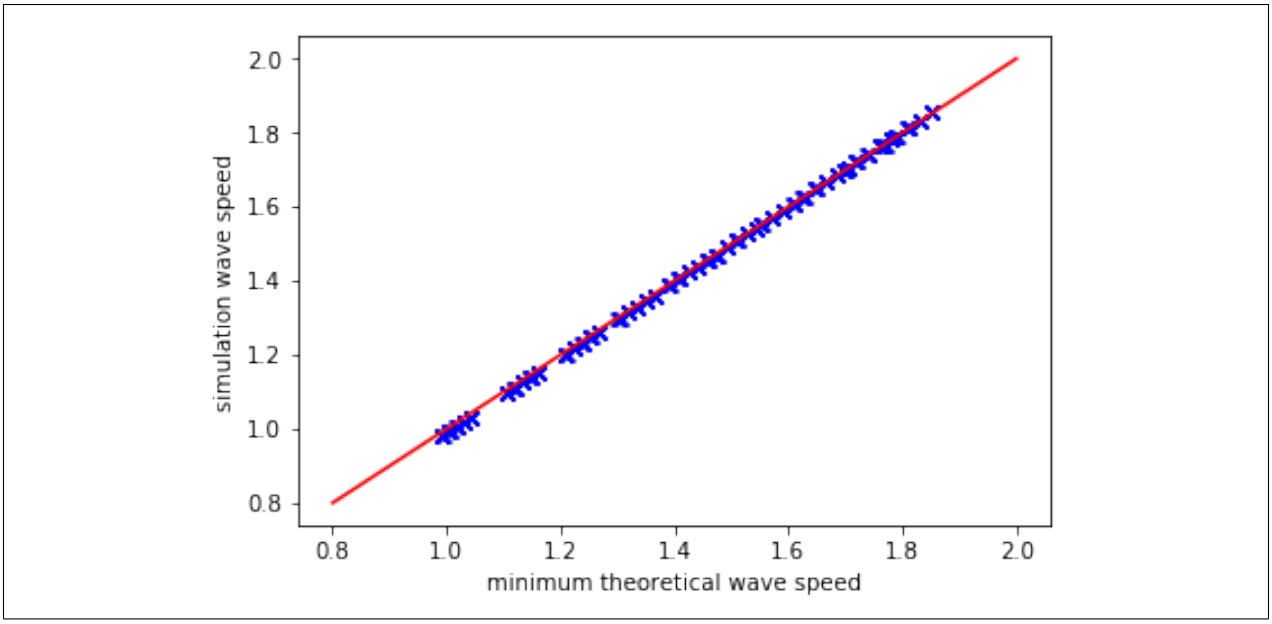

FIGURE 6 – Vitesse du front observée numériquement en fonction de la vitesse minimale théorique

Soit

$$s^*_{theorique} = K\frac{f_0(18F_0+4)+\sqrt{f_0(f_0(18F_0+4)^2+108(f_0+4F_0)F_0^2)}}{2(f_0+4F_0)} \tag{36}$$

la vitesse minimale théorique obtenue dans la partie 3.
Soit $\rho_\infty = \sup(\rho)$ et

$$X(t) = \inf(x/\rho(x,t) > \frac{\rho_\infty}{2}). \tag{37}$$

$X(t)$ est alors une approximation de la position du front à l'instant t.
La vitesse observée numériquement est alors choisie comme étant :

$$s_{simulation} = \frac{X(t_1+t_2)-X(t_1)}{t_2} \tag{38}$$

où t_1 et t_2 sont deux temps arbitraires où le front est déjà établi.
Le graphe ci dessus représente par les points bleus la vitesse du front observée numériquement pour différentes simulations (différents K, F_0 et f_0 et données initiales) en fonction de la vitesse minimale théorique associée à cette simulation. La droite rouge est la droite $s_{simu} = s^*_{theorique}$.
On remarque que la vitesse du front observée numériquement est très proche de la vitesse minimale théorique : ce phénomène est similaire à celui de l'équation de Fisher-KPP : pour une donnée initiale à support compact, **le front se propage asymptotiquement à la vitesse minimale de l'équation d'onde associée à l'EDP, obtenu par linéarisation autour de l'état** $(0, 0, C_0)$.

5.3 Résolution de l'EDP en 2D

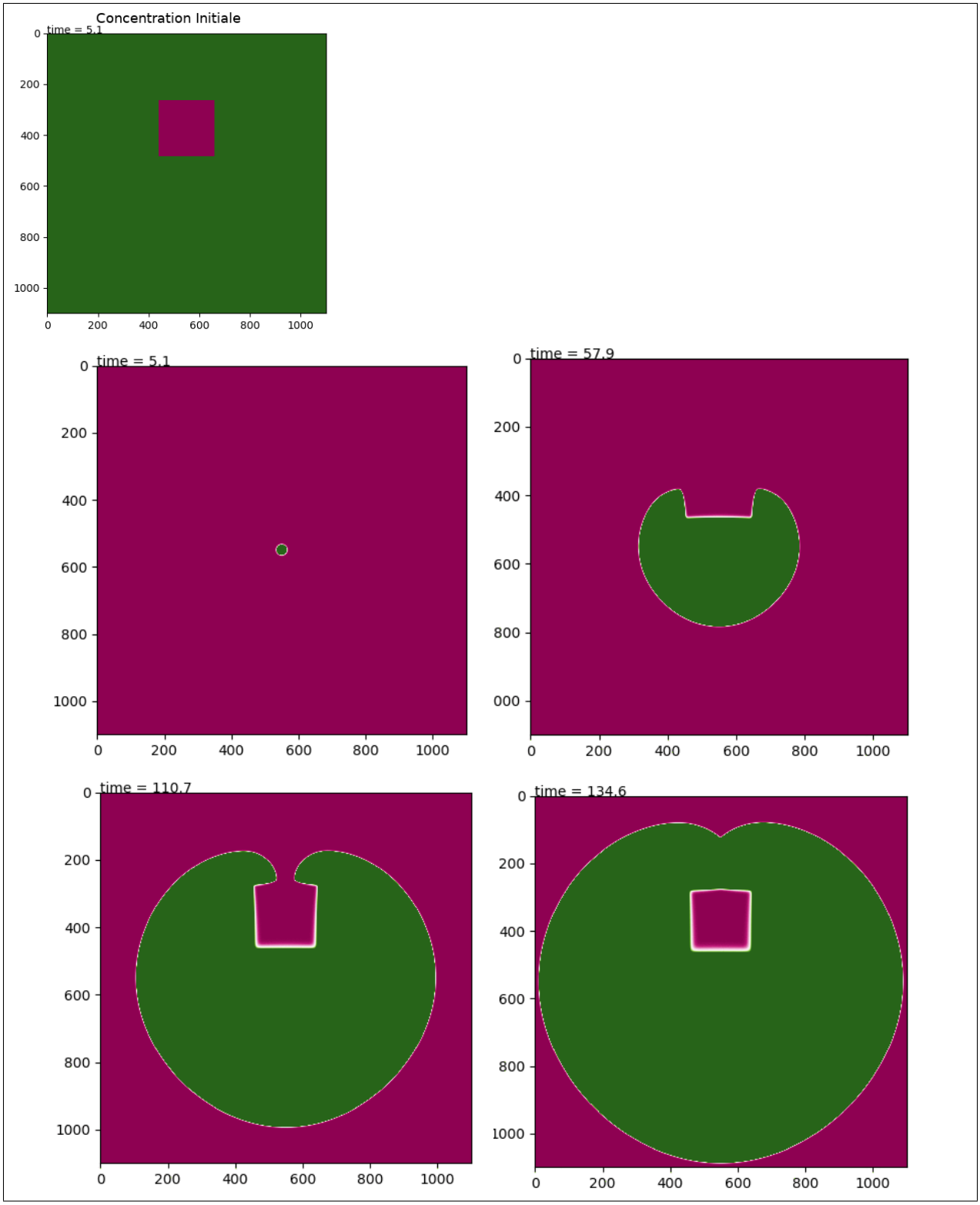

FIGURE 7 – Évolution de ρ pour la résolution du schéma semi implicite I pour l'EDP en 2D avec un trou de concentration. Ici la grille est de taille 1100x100, avec 1000 pas de temps.

Deux challenges numériques ont étés rencontrés :

- 1) Temps d'exécution : le schéma consiste principalement à résoudre à chaque pas de temps une équation matricielle $AX = B$, où A est de taille $n = 1100 * 1100 = 1.21 * 10^6$. Une idée naive serait alors de calculer A^{-1} par exemple par pivot de gauss, qui a un coût $\mathcal{O}(n^3)$, ou par l'algorithme de Strassen qui est un peu plus efficace $\mathcal{O}(n^{2.8})$. Cependant ces stratégies ne sont pas efficaces car il n'est pas nécessaire de calculer l'inverse de A : On cherche seulement l'antécédent d'une image. La stratégie implémentée ici est de trouver X par méthode des minimums résiduels, c'est à dire minimiser la fonction $x \to ||Ax - B||^2$ par méthode des moindres carrées, méthode qui fonctionne pour les matrices symétriques et qui est très rapide pour les matrices creuses (notre cas).
On peut encore accélérer la convergence de la méthode des minimums résiduels en précisant pour μ^{n+1} le guess μ^n.

- 2) Coût en mémoire : Une matrice carrée de taille $1.21 * 10^6$ pèse 10GB de RAM sous Python... Heureusement, dans notre cas la matrice A est creuse et il existe des méthodes et librairies adaptées pour représenter de telles matrices efficacement. De plus chaque ρ^n, μ^n et C^n est représenté par une matrice carrée de taille 1100, ce qui pèse 10MB. Ces matrices ne sont pas creuses donc il est plus dur de les compresser. Si l'on veut stocker ces quantités pour ensuite retracer l'évolution de ρ, μ, C pour 1000 pas de temps, ceci coûterait 30GB de RAM... Il faut donc faire des compromis, ce qui est classique : par exemple, on ne sort pas toutes les solutions mais uniquement à certains pas de temps bien choisis.

6 Recherche de la vitesse d'onde des solutions progressives de l'Équation fluide complète du champignon.

Le but de cette section est de montrer comment l'on peut obtenir la vitesse d'onde pour l'équation du champignon complète :

$$\begin{cases} \partial_t \mu + \nabla(\mu v) = f(C)(\mu + \rho) - \mu\rho \\ \partial_t(\mu v) + \nabla(\mu v \times v) + T\nabla\mu = -\lambda\mu v + \mu\nabla C - \mu v\rho \\ \partial_t \rho = F(v)\mu \\ \partial_t C = -b\rho C. \end{cases} \tag{39}$$

En effet dans les sections précédentes nous avons travaillés dans le cas λ et T très grands ce qui simplifiait les équations.
Cependant nous avons pu voir que, comme pour l'équation de Fisher KPP, pour une donnée initiale à support compact, la solution tend vers une solution d'onde et que la vitesse d'onde est déterminée par la plus petite vitesse (en valeur absolue) donnée par la condition d'amortissement fort autour de l'état initial.
Nous allons alors adopter la même stratégie pour l'équation complète.
Ici nous allons linéariser autour de l'état $(\mu, \rho, C, v) = (0, 0, C_0, v)$.
Equation d'onde pour le fluide :
Soit s la vitesse d'onde, $y = x - st$, par le même argument de symétrie que pour l'équation de "KPP avec mémoire", nous allons se placer dans le cas $s < 0$. Avec abus de notation : $(x, t) =: (y)$, où $y = x - st$, l'équation d'onde pour le fluide est :

$$\begin{cases} -s\mu' + (\mu v)' = f(C)(\mu + \rho) - \mu\rho \\ -s(\mu v)' + (\mu v \times v)' + T\mu' = -\lambda\mu v + \mu C' - \mu v\rho \\ -s\rho' = F(v)\mu \\ -sC' = -b\rho C. \end{cases} \tag{40}$$

Linéarisation autour de l'état $(\mu, \rho, C, v) = (0, 0, C_0, v)$:
On pose $f(C_0) = f_0$, et on cherche des solutions de la forme $\rho = \rho_0 \exp(Xy)$ autour de $\rho = 0$.
On obtient en intégrant la quatrième ligne $C = C_0 - \frac{b\rho_0 C_0}{X} \exp(Xy)$.
La troisième ligne donne : $-sX\rho_0 \exp(Xy) = F_0\mu$ donc $\mu = \frac{-sX\rho_0}{F_0} \exp(Xy)$.
Autour de $(\mu, \rho, C, v) = (0, 0, C_0, v)$, la premiere ligne donne : $-s\mu' + (\mu v)' = f_0(\mu + \rho)$ car on peut negliger $\mu\rho$ devant μ et ρ . On a donc

$$(\mu v)' = (f_0\rho_0 - \frac{sXf_0\rho_0}{F_0} - \frac{s^2X^2\rho_0}{F_0}) \exp(Xy) \tag{41}$$

donc

$$\mu v = (\frac{f_0\rho_0}{X} - \frac{f_0 s\rho_0}{F_0} - \frac{s^2X\rho_0}{F_0}) \exp(Xy). \tag{42}$$

Ainsi

$$v = \frac{\mu v}{\mu} = v_0 = s + \frac{f_0}{X} - \frac{f_0F_0}{X^2 s}. \tag{43}$$

La deuxième ligne donne $-s(\mu v)' + (\mu v \times v)' + T\mu' = -\lambda\mu v + \mu C' - \mu v\rho$.
On peut ici négliger $\mu v\rho$ et $\mu C'$ devant μv. On a donc :

$$(-v_0 sX\mu_0 + v_0^2 X\mu_0 + T\mu_0 X) \exp(Xy) = -\lambda\mu_0 v_0 \exp(Xy) \tag{44}$$

donc

$$-sXv_0 + v_0^2 X + TX = -\lambda v_0. \tag{45}$$

En multipliant par X^2 et en substituant $v_0 = s + \frac{f_0}{X} - \frac{f_0 F_0}{X^2 s}$ on a alors l'équation caracteristique :

$$X^4(Ts^2) + X^3(\lambda + f_0)s^3 + X^2(f_0^2 s^2 + \lambda f_0 s^2 - f_0 F_0 s^2) + X(-sF_0 f_0)(\lambda + 2f_0) + f_0^2 F_0^2 = 0. \tag{46}$$

En posant $Y = sX$ on obtient :

$$Y^4 + s^2 P_3(Y) = 0 \tag{47}$$

où $P_3(Y) \equiv \frac{1}{T}(Y^3(\lambda + f_0) + Y^2(f_0^2 + \lambda f_0 - f_0 F_0) - Y(F_0 f_0(\lambda + 2f_0)) + f_0^2 F_0^2)$ est un polynôme de degré 3 dont les coefficients ne dépendent que des données f_0, F_0, λ et T.
Soit Y une racine de l'équation 47, s est déterminé par Y par l'équation :

$$s^2 = -\frac{Y^4}{P_3(Y)}. \tag{48}$$

On cherche le plus grand s négatif tel que les racines de 47 soit toutes réelles (condition d'amortissement fort), donc nécessairement

$$\frac{\partial s}{\partial Y} = 0 \tag{49}$$

Ainsi

$$\Big(\frac{Y^4}{P_3(Y)}\Big)' = 0 \tag{50}$$

i.e.

$$Q(Y) \equiv 4P_3(Y) - YP_3'(Y) = 0 \tag{51}$$

Le polynome Q ne dépend pas de s, on peut donc calculer ses racines réelles (il en a une ou trois). On obtient alors un ou trois candidats négatifs pour s par la formule :

$$s^2 = -\frac{Y^4}{P_3(Y)}.$$

Le s recherché est alors le plus grand de ces candidats : en effet, ce s est le plus grand s tel que les racines de 47 soient toutes réelles.

Appendices

Code de résolution de l'EDO

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

b=.5 # dtC=-b*rho*C
F0=1 # dtRho = Fo*Mu
tf=20 # temps final de la simulation
rho0=0 #rho initial
mu0=.1 #mu initial
c0=1 #concentration initiale
n=2000 #nombre de pas de temps

#Résolution du schéma éxplicite
def euler_explicite_edo(b, F0, tf, rho0, mu0, c0, n):
  #t0<t1 , temps etudies,
  #rho0, mu0,c0 reels positifs: condition initiale
  #n entier(nombre d'iterations)
  h=tf/n #pas Deltat
  rho=rho0
  mu=mu0
  c=c0
  t=0
  Rho=[rho0]
  Mu=[mu0]
  C=[c0]
  T=[t]
  for k in range(n):
    new_mu = mu + h*(c*(mu+rho)-mu*rho)
    new_rho = rho + h*F0*mu
    new_c = c - h*b*rho*c
    mu=new_mu
    rho=new_rho
    c=new_c
    t=t+h
    Mu.append(new_mu)
    Rho.append(new_rho)
    C.append(new_c)
    T.append(t)
  return T,Mu,Rho,C

#Résolution du schéma semi- implicite I
def euler_semi_I_edo(b, F0, tf, rho0, mu0, c0, n):
  #t0<t1 , temps etudies,
  #rho0, mu0,c0 reels positifs: condition initiale
  #n entier(nombre d'iterations)
  h=tf/n #pas Deltat
  rho=rho0
  mu=mu0
  c=c0
  t=0
  Rho=[rho0]
  Mu=[mu0]
  C=[c0]
  T=[0]
```

```
    for k in range(n):
      new_mu = (mu + h*c*rho)/(1+h*rho-h*c*(1+h*F0))
      new_rho = rho + h*F0*new_mu
      new_c = c/(1 + b*h*new_rho)
      mu=new_mu
      rho=new_rho
      c=new_c
      t=t+h
      Mu.append(new_mu)
      Rho.append(new_rho)
      C.append(new_c)
      T.append(t)
    return T,Mu,Rho,C

#Résolution du schéma semi- implicite II
def euler_semi_II_edo(b, F0, tf, rho0, mu0, c0, n):
    #t0<t1 , temps etudies,
    #rho0, mu0,c0 reels positifs: condition initiale
    #n entier(nombre d'iterations)
    t=0
    h=tf/n #pas Deltat
    rho=rho0
    mu=mu0
    c=c0
    Rho=[rho0]
    Mu=[mu0]
    C=[c0]
    T=[0]
    for k in range(n):
      new_mu = (mu + h*c*rho-h*rho*mu)/(1-h*c*(1+h*F0))
      new_rho = rho + h*F0*new_mu
      new_c = c/(1 + b*h*new_rho)
      mu=new_mu
      rho=new_rho
      c=new_c
      t=t+h
      Mu.append(new_mu)
      Rho.append(new_rho)
      C.append(new_c)
      T.append(t)
    return T,Mu,Rho,C

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

#Résolution du schéma implicite
def euler_implicite_edo(b, F0, tf, rho0, mu0, c0, n):
    #t0<t1 , temps etudiés,
    #rho0, mu0,c0 reels positifs: conditions initiale
    #n entier(nombre d'itérations)
    t=0
    newton_steps=10 #nombre d'itérations de la méthode de Newton-Raphson pour le
      calcul implicite
    h=tf/n #pas deltat
    rho=rho0
```

```
    mu=mu0
    c=c0
    Rho=[rho0]
    Mu=[mu0]
    C=[c0]
    T=[0]
    for k in range(n):
      #Calcul de new_mu par methode de Newton Raphson
      #coefficients du polynome d'ordre 3 en new_mu
      alpha = -h**4*F0**2*b
      beta= -F0*h**2*(b+1+2*rho*b*h)
      gamma= -(1+b*h*rho)+b*h**2*F0*mu+h*(c*(1+h*F0)-rho*(1+b*h*rho))
      delta=(1+b*h*rho)*mu + h*c*rho
      def P(X):
        return alpha*X**3+beta*X**2+gamma*X+delta
      def gradP(X):
        return 3*alpha*X**2+2*beta*X+gamma
      new_mu=newton(P,gradP,newton_steps,mu)
      new_rho = rho + h*F0*new_mu
      new_c = c/(1 + b*h*new_rho)
      mu=new_mu
      rho=new_rho
      c=new_c
      t=t+h
      Mu.append(new_mu)
      Rho.append(new_rho)
      C.append(new_c)
      T.append(t)
    return T,Mu,Rho,C

#Utilisation des libraries python (scipy) pour résoudre l'EDO
def black_box_edo(b, F0, tf, rho0, mu0, c0, n):
      def f(t,y,arg1,arg2):
          mu=y[0]
          rho=y[1]
          c=y[2]
          return [c*(rho+mu)-mu*rho, F0*mu, -b*rho*c]

      r = ode(f).set_integrator('zvode', method='adams')
      r.set_initial_value([mu0,rho0,c0],0).set_f_params(F0,b)
      dt=tf/(n-1)
      Rho=[rho0]
      Mu=[mu0]
      C=[c0]
      t=0
      T=[0]
      while r.t < tf:
          mu,rho,c = r.integrate(r.t+dt)
          Mu.append(mu)
          Rho.append(rho)
          C.append(c)
          T.append(r.t)
      return T,Mu,Rho,C

#Résolution
T,Mu,Rho,C = euler_semi_I_edo(b, F0, tf, rho0, mu0, c0, n)


rho_inf= Rho[n-1]
```

```
#Étude Asymptotique
A=[np.log(mu) for mu in Mu]
B=[-min(1,b)*y*rho_inf for y in T]

#Tracé des solutions et de l'étude asymptotique
plt.subplot(221)
plt.plot(T,Mu)
plt.ylabel('mu')
plt.xlabel('t')
plt.subplot(222)
plt.plot(T,Rho)
plt.ylabel('rho')
plt.subplot(223)
plt.plot(T,C)
plt.ylabel('C')
plt.subplot(224)
plt.plot(T,A)
plt.plot(T,B)
plt.ylabel('log(mu), -b*rho_inf*t')
plt.show()
```

edo.py

Code de la résolution de l'EDP en 1D

```
# %load edp_1d.py
import matplotlib.pyplot as plt
import numpy as np
import scipy.sparse as sp
from scipy.sparse.linalg.dsolve import spsolve
import matplotlib.animation as animation

#Coéfficients physiques
K=.5 #coefficient diffusion
b=.2 # dtC=-b*rho*C
F0= 1 # dtRho = Fo*Mu

#Paramêtres numériques
n_t=3001 #nombre de pas de temps
tf=20 # temps final de la simulation
xf = 150 #longeur de la simulation
n_x =601 #nombres de points de la simulation

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

def edp_1d_explicite(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x):
    dt=tf/(n_t-1)
    dx=xf/(n_x-1)
    X=np.linspace(0,xf,n_x)
    T=np.linspace(0,tf,n_t)
    Mu=np.zeros((n_t,n_x))
    Rho=np.zeros((n_t,n_x))
    C=np.zeros((n_t,n_x))
    Mu[0]=mu0
    Rho[0]=rho0
    C[0]=c0
```

```
    #Résolution du schema éxplicite
    for n in range(0,n_t-1):
        RHS=np.zeros(n_x)
        alpha=-C[n]*dt*(1+dt*F0)+dt*Rho[n]+1
        RHS[1:-1]= dt*((K/(dx**2))*(Mu[n,:-2]-2*Mu[n,1:-1]+Mu[n,2:])+C[n,1:-1]*Rho[n
    ,1:-1])
        RHS[0]= dt*((K/(dx**2))*(-2*Mu[n,0]+Mu[n,1])+C[n,0]*Rho[n,0])
        RHS[-1]=dt*((K/(dx**2))*(-2*Mu[n,-1]+Mu[n,-2])+C[n,-1]*Rho[n,-1])
        Mu[n+1]=(1/alpha)*(Mu[n]+RHS)
        Rho[n+1]=Rho[n]+dt*F0*Mu[n+1]
        C[n+1]=C[n]/(1 + b*dt*Rho[n+1])
    return X,T,Mu,Rho,C

def edp_1d_semi_implicite_I(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x):
    #Détermination des paramêtres numeriques deltat et deltax
    dt=tf/(n_t-1)
    dx=xf/(n_x-1)
    #Représentation de l'éspace et du temps
    X=np.linspace(0,xf,n_x)
    T=np.linspace(0,tf,n_t)
    #Initialisation
    Mu=np.zeros((n_t,n_x))
    Rho=np.zeros((n_t,n_x))
    C=np.zeros((n_t,n_x))
    Mu[0]=mu0
    Rho[0]=rho0
    C[0]=c0
    #Résolution du schéma implicite-explicite I
    for n in range(0,n_t-1):
        #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
    -1),1)
        #Laplacien Numerique
        A=A*K*dt/(dx**2)
        #Ajout des termes implicites
        alpha=-C[n]*dt*(1+dt*F0)+dt*Rho[n]+1
        A+=np.diag(alpha,0)
        A=sp.csc_matrix(A)
        #Résolution du systême implicite
        Mu[n+1]= spsolve(A, Mu[n]+dt*C[n]*Rho[n])
        Rho[n+1]=Rho[n]+dt*F0*Mu[n+1]
        C[n+1]=C[n]/(1 + b*dt*Rho[n+1])
    return X,T,Mu,Rho,C


def edp_1d_semi_implicite_II(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x):
    #Détermination des paramêtres numériques deltat et deltax
    dt=tf/(n_t-1)
    dx=xf/(n_x-1)
    #Représentation de l'éspace et du temps
    X=np.linspace(0,xf,n_x)
    T=np.linspace(0,tf,n_t)
    #Initialisation
    Mu=np.zeros((n_t,n_x))
    Rho=np.zeros((n_t,n_x))
    C=np.zeros((n_t,n_x))
    Mu[0]=mu0
    Rho[0]=rho0
    C[0]=c0
```

```
    #Résolution du schéma implicite-explicite II
    for n in range(0,n_t-1):
        #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-1),1)
        A=A*K*dt/(dx**2) #Laplacien Numerique
        #Ajout des termes implicites
        alpha=-C[n]*dt*(1+dt*F0)+1
        A+=np.diag(alpha,0)
        A= sp.csc_matrix(A)
        #Résolution du systême implicite
        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]
        C[n+1]=C[n]/(1 + b*dt*Rho[n+1])
    return X,T,Mu,Rho,C
#X,T,Mu,Rho,C= edp_1d_semi_implicite_I(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x)

X,T,Mu,Rho,C= edp_1d_semi_implicite_I(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x)


def speed(X,Rho,rho_inf):
    #Position du front
    argmed=np.zeros(n_t)
    for i in range(n_t):
        argmed[i]= X[(n_x//2)+ \
            np.min(np.where(np.append(Rho[i,(n_x//2):],[0])<rho_inf/2))]
    #Vitesse du front
    s = ((n_t-1)/tf)*(argmed[(n_t//2)+150]-argmed[(n_t//2)])/(150)
    return s

rho_inf = Rho[n_t-1,(n_x//2)]
s = speed(X,Rho,rho_inf)
print('La vitesse de propagation de la simulation est s=',s)


# Comparaison de s theorique et numerique pour plusieurs données initiales
# ~ memory =[]
# ~ for i in range(5):
    # ~ for j in range(5):
        # ~ K = .2 + .2*i
        # ~ b = .1 + .1*j
        # ~ X,T,Mu,Rho,C= edp_1d_semi_implicite_I(K, b, F0, rho0, mu0, c0, n_t , tf, xf, n_x)
         # ~ #Valeur de rho a l'infini
        # ~ rho_inf = Rho[n_t-1,(n_x//2)]
        # ~ s = speed(X,Rho,rho_inf)
        # ~ s_theorique = np.sqrt(K*((18*F0+4)+np.sqrt(((18*F0+4)**2)+108*(1+4*F0)*(F0**2)))/(2*(1+4*F0)))
        # ~ memory += [K,b,s,s_theorique]
# ~ np.savetxt('memory_data3.dat', memory)


s_theorique = np.sqrt(K*((18*F0+4)+np.sqrt(((18*F0+4)**2)+108*(1+4*F0)*(F0**2)))/(2*(1+4*F0)))
#Attention, ceci est pour C0=1
print('La vitesse théorique de propagation est s_theorique=', s_theorique)

#Animation
fig = plt.figure()
```

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

def init():
    line.set_data([], [])
    line2.set_data([], [])
    line3.set_data([], [])
    line4.set_data([], [])
    time_text.set_text('')
    legend_text.set_text('')
    return line,line2,line3,line4, time_text, legend_text


def animate(i):
    line.set_data(X, C[i])
    line2.set_data(X, Rho[i])
    line3.set_data(X, Mu[i])
    #line4.set_data(xf/2+((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))
    legend_text.set_text('Rho=Orange \nMu=Green \nC=Blue\ns={0:.3f}'.format(s))
    return line,line2, line3,line4, time_text, legend_text


anim = animation.FuncAnimation(fig, animate, init_func=init,
                               frames=(n_t-1), interval=(tf*200)/(n_t-1), blit=True)



#anim.save('EDP_1D.gif',writer='imagemagick', fps=30)
plt.show()
```

edp_1d.py

Code de la résolution de l'EDP en 2D

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.animation as animation
import numpy as np
import scipy.sparse as sp
from scipy.sparse.linalg.dsolve import spsolve
from scipy.sparse.linalg import bicgstab, bicg, cg, cgs, gmres, lgmres, minres, qmr,
    gcrotmk
from IPython.display import HTML
import time
start_time = time.time()

#Coéfficients physiques
K=.2 #coefficient diffusion
b=.2# dtC=-b*rho*C
F0= 1 # dtRho = Fo*Mu

physique = [K,b,F0]
```

```
#Paramêtres numériques
n_t=500 #nombre de pas de temps
tf=170 # temps final de la simulation
xf = 500 #longueur de la simulation
n_x = 1100 #nombres de points de la simulation
yf = xf
n_y = n_x
n_xy = n_x * n_y
numerique = [n_t, tf, xf, n_x, yf, n_y]

params = physique, numerique


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

xm = 200
xM = 300
ym = 120
yM = 220
im = ((xm*n_x)//xf)
iM = ((xM*n_x)//xf)
jm = ((ym*n_y)//xf)
jM = ((yM*n_y)//xf)
for i in range(im,iM):
    for j in range(jm,jM):
        c0[i+j*n_x] = 0.

class EDP():
    def __init__(self,params):
        self.physique, self.numerique = params
        self.K, self.b, self.F0 = self.physique
        self.n_t, self.tf, self.xf, self.n_x, self.yf, self.n_y = self.numerique


        self.n_xy = self.n_x*self.n_y
        self.dt = self.tf/(self.n_t-1)
        self.dx = self.xf/(self.n_x-1)
        self.dy = self.yf/(self.n_y-1)

        #self.X = np.linspace(0,self.xf,self.n_x)
        #self.Y = np.linspace(0,self.yf,self.n_y)
        #self.T = np.linspace(0,self.tf,self.n_t)

        #Matrice du Laplacien
        self.Lapl = sp.diags(-4*np.ones(self.n_xy),0)
        #Lapl += sp.diags(np.ones(n_xy-1),1)+sp.diags(np.ones(n_xy-1),-1)
        diagmod = np.ones(self.n_xy-1)
        diagmod[np.arange(self.n_y-1,self.n_xy-1,self.n_y)] = np.zeros(self.n_y-1)
        self.Lapl += sp.diags(diagmod,1) + sp.diags(diagmod,-1)
        self.Lapl += sp.diags(np.ones(self.n_xy-self.n_y),self.n_y)+sp.diags(np.ones
    (self.n_xy-self.n_y),-self.n_y)
        self.Lapl = -self.K*self.dt/(self.dx**2)*self.Lapl
        self.Cond = sp.identity(self.n_xy)
    def array_to_2D(n_x,vect):
        return np.array(np.split(vect,n_x))
```

```
1076
    def integrate(self,initial):
1078        mu,rho,c = initial
            alpha=-c*self.dt*(1+self.dt*self.F0)+self.dt*rho+1
1080        A = self.Lapl + sp.diags(alpha,0)
            Target =  mu+self.dt*c*rho
1082
            #next_mu = spsolve(A,Target) #95.28 secondes d'execution
1084        #next_mu,check = bicg(A,Target) #3.38 secondes d'execution
            #next_mu,check = bicgstab(A,Target, x0=mu) #2.15 secondes d'execution
1086        #next_mu,check = cg(A,Target) #2.29 secondes d'execution
            #next_mu,check = cgs(A,Target) #2.36 secondes d'execution
1088        #next_mu,check = gmres(A,Target) #2.72 secondes d'execution
            #next_mu,check = lgmres(A,Target) #2.62 secondes d'execution
1090        next_mu,check = minres(A,Target, x0=mu, M=self.Cond) #2.15 secondes d'
    execution
            #next_mu,check = qmr(A,Target) #3.70 secondes d'execution
1092        #next_mu,check = gcrotmk(A,Target) #2.62 secondes d'execution
            next_rho = rho + self.dt*self.F0*next_mu
1094        next_c = c/(1+self.b*self.dt*next_rho)
            return next_mu, next_rho, next_c
1096

1098 Agent = EDP(params)
  dt = Agent.dt
1100
  mu= mu0
1102 rho= rho0
  c= c0
1104 Mu=[mu0]
  Rho=[rho0]
1106 C=[c0]
  T=[0]
1108 n = 0
  step = 5
1110 while n<n_t:
      mu,rho,c = Agent.integrate((mu,rho,c))
1112  if n % step ==0 :
          Mu.append(mu)
1114      Rho.append(rho)
          C.append(c)
1116      T.append(n*dt)
      if n % 25 ==0 :
1118      print(n, (time.time() - start_time))
      n+=1
1120
  print("--- %s seconds ---" % (time.time() - start_time))
1122
  tot = len(Mu)
1124
  Draw = 'C'
1126 f = C

1128 fig = plt.figure()
  im = plt.imshow(EDP.array_to_2D(n_x,f[2]), animated=True, cmap ='PiYG')
1130 time_text = plt.text(0, 0, '')

1132 i = 2
  def updatefig(*args):
```

```
    global i
    i+=1
    if i< tot -1:
        im.set_array(EDP.array_to_2D(n_x,f[i]))
        time_text.set_text('time = {0:.1f}'.format(T[i]))
    return im, time_text

ani = animation.FuncAnimation(fig, updatefig, interval=100, blit=True, repeat=True)
ani.save('EDP_2D_'+Draw+'.gif',writer='imagemagick', fps=30)

Draw = 'Mu'
f = Mu

fig = plt.figure()
im = plt.imshow(EDP.array_to_2D(n_x,f[2]), animated=True, cmap ='PiYG')
time_text = plt.text(0, 0, 'test')
i = 2
def updatefig(*args):
    global i
    i+=1
    if i< tot -1:
        im.set_array(EDP.array_to_2D(n_x,f[i]))
        time_text.set_text('time = {0:.1f}'.format(T[i]))
    return im, time_text

ani = animation.FuncAnimation(fig, updatefig, interval=100, blit=True, repeat=True)
ani.save('EDP_2D_'+Draw+'.gif',writer='imagemagick', fps=30)


Draw = 'Rho'
f = Rho

fig = plt.figure()
im = plt.imshow(EDP.array_to_2D(n_x,f[2]), animated=True, cmap ='PiYG')
time_text = plt.text(0, 0, '')
i = 2
def updatefig(*args):
    global i
    i+=1
    if i< tot -1:
        im.set_array(EDP.array_to_2D(n_x,f[i]))
        time_text.set_text('time = {0:.1f}'.format(T[i]))
    return im, time_text

ani = animation.FuncAnimation(fig, updatefig, interval=100, blit=True, repeat=True)
ani.save('EDP_2D_'+Draw+'.gif',writer='imagemagick', fps=30)


plt.show()
```

edp_2d.py