UFR des mathématiques

TER

Modélisation et simulation de la croissance des tumeurs

Fatimetou Mohamed El Hacen
Ahmed El Kerim

Responsable d'encadrement :
Mme Claire Chainais

Lille , Mai 2018

Nous voudrions présenter nos remerciements à notre responsable d'encadrement Mme Chainais. Nous voudrions également lui témoigner notre gratitude pour sa patience et son soutien qui nous a été précieux afin de mener à bien notre travail.

Table des matières

Chapitre 1

1.1 Introduction et Modélisation

1.1.1 Problématique(Modèle de croissance tumorale)

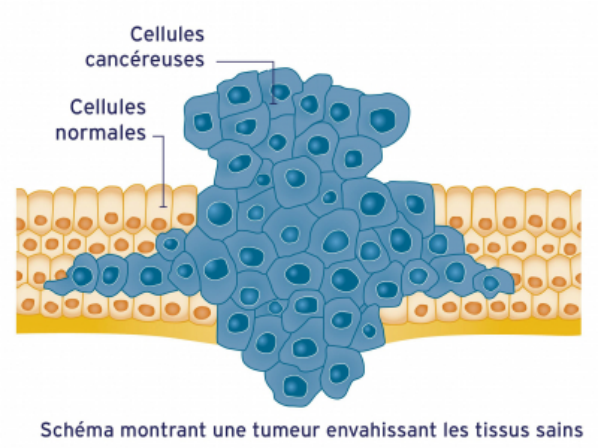


Schéma montrant une tumeur envahissant les tissus sains

Le cancer est l'une des plus grandes causes de mortalité de nos jours. Cette maladie désigne la croissance anarchique de cellules envahissant et asphyxiant les organes voisins.
Un très grand nombre de phénomènes interviennent dans le développement d'une tumeur (amas de cellules) qui n'est plus sensible à la régulation de la division cellulaire et dont la division nécessite beaucoup de nutriments et d'oxygène. Les différentes interactions sont encore mal comprises et la modélisation mathématique peut se révéler très utile pour l'étude et la compréhension de ces interactions.

La modélisation mathématique intègre les connaissances médicales et permet de réaliser des simulations numériques de croissance tumorale grâce à des méthodes numériques.
Ces simulations numériques peuvent aider à l'optimisation des traitements et au développement de nouvelles modalités pour les thérapies.
Ils existent différentes approches pour bâtir des modèles mathématiques décrivant la croissance tumorale à l'aide des équations aux dérivées partielles. Benoît Perthame, Min Tang et Nicolas Vauchelet dans leur article **"Traveling wave solution of the Hele-Shaw model of tumor growth with nutrient"** ont suggérée une modélisation mathématique pour la croissance tumorale avec nutriment. Le modelé qu'ils proposent conduit à un problème elliptique à frontière libre couplé à une équation de diffusion des nutriments qui peut être instable. Ils expliquent à l'aide de plusieurs simulations numériques que la concentration des cellules malades est soumise aux phénomènes de réaction-diffusion des nutriments.

1.1.2 Présentation du modèle

Les inconnus du modèle sont : $n(x,t)$ la densité des cellules malades et $c(x,t)$ la concentration des nutriments.
Le système couplé de Hele-Shaw s'écrit sous la forme :

$$\begin{cases} -\triangle c + \Psi(n,c) = 0. & (1) \\ \frac{\partial n}{\partial t} - div(n\nabla p(n)) = nG(c) & (2) \end{cases}$$

Où :

- $p(n) = n^\gamma$ représente la loi de pression
- γ est une constante positive
- $\Psi(n,c)$ représente les deux effets : la consommation des nutriments par les cellules, l'apport des nutriments aux cellules.
- G représente le taux d'accroissement de cellules malades.

Dans l'article, ils présentent deux modèles pour l'étude de la concentration des nutriments :

- Modèle in vivo : Dans ce cas on considère que le nutriment est apporté par le réseau vasculaire dans la région saine ($n = 0$) et diffusé vers le tissu. On choisit

$$\Psi(n,c) = \Psi(n)c + 1_{(n=0)}(c - C_B)$$

L'équation sur c s'écrit :

$$-\triangle c + \Psi(n)c = 1_{(n=0)}(c - C_B)$$

— Modèle in vitro : Dans ce cas on suppose que la tumeur est entourée d'un liquide dans lequel la diffusion des nutriments est si rapide que l'on suppose qu'elle est constante à l'intérieur de la région tumorale. La consommation est linéaire :

$$\Psi(n, c) = \Psi(n)c$$

l'équation de la consommation des nutriments s'écrit donc :

$$-\triangle c + \Psi(n)c = 0, x \in (n > 0)c = C_B pour x \in (n = 0).$$

1.1.3 Objectif du TER

Le travail présenté au cours de cette étude a pour objectif de faire un couplage entre les 2 équations modélisant la concentration en nutriments et la densité de cellules malades et faire des simulations numériques pour nos schémas. Dans le cadre de ce travail, notre référence principale était le modèle Hele-Shaw développé par Benoit Perthame, Min Tang et Nicolas Vauchelet. Plusieurs difficultés se présentent en traitant ce problème dont tout d'abord la non linéarité de l'équation (2) sur n ,dans le terme concernant la pression $p(n) = n^\gamma$.
La deuxième difficulté qui se présente est le couplage entre les équations modélisant les deux phénomènes pour mettre en oeuvre la dépendance entre les deux.
Les schémas utilisés sont des schémas de type différences finies. Notre stratégie pour résoudre ce problème consiste à résoudre (1) à n fixé avec la méthode des différences finies à une dimension en s'inspirant de nos codes déjà développés en Fortran dans des projets précédents pour des équations similaires. Dans un deuxième temps comme (2) est non linéaire on a programmé une méthode de Newton pour sa résolution avec c fixé . Enfin, après avoir validé les deux méthodes de résolution pour chaque équation on met en oeuvre le couplage entre les deux. On se limite dans cette étude au modèle in vitro.

Chapitre 2

2.1 Méthode numérique pour le calcul de c

2.1.1 Présentation du schéma

Dans un premier temps, nous allons donc étudier ce problème sur l'intervalle $\Omega = [-5, 5]$ sans couplage avec l'équation (2). Nous considérons les conditions aux limites de Dirichlet homogène. Ici n est une donnée . Notre problème s'écrit donc comme le suivant :

$$-\triangle c + \Psi(n)c = 0 \quad x \in (n > 0)$$

$$c = C_B \quad x \in (n = 0).$$

On discrétise notre domaine $\Omega = [-5, 5]$ à l'aide d'un maillage de pas $h = \frac{10}{L+1}$ avec $(L + 2)$ points. On note c_i la solution $i = 1, ..., L + 2$, l'approximation de $c(ih)$. Elle est donnée par le schéma :

$$\begin{cases} -\frac{c_{i+1}-2c_i+c_{i-1}}{h^2} + \Psi(n_i)c_i = 0, \quad \forall i \in J, \quad J = \{i/n_i \geq 0\} \\ c_i = C_B, \quad \forall i \in J^{'}, \quad J^{'} = \{i/n_i = 0\} \end{cases}$$

On peut remarquer que la solution de ce problème n'est variable que à l'intérieur de la croissance du tumeur et donc la difficulté par rapport à ce que on a déjà fait dans les projets passés est de déterminer les indices pour lesquels $n_i > 0$. Notamment dans le couplage, l'ensemble $J = [i_1, i_2]$ change à chaque itération. Le problème précèdent se simplifie au système discrétisé suivant sur J :

$$\begin{cases} -\frac{c_{i+1}-2c_i+c_{i-1}}{h^2} + \Psi(n_i)c_i = 0, \quad \forall i \in J \\ c_{i_1-1} = C_B, \quad c_{i_2-1} = C_B \end{cases}$$

2.1.2 Mise sous forme matricielle

Pour passer à la résolution de notre problème on écrit le système sous la forme matricielle suivante : $Ac = b$ avec

$$A = \begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \vdots \\ \vdots & \ddots & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \vdots \\ \vdots & 0 & \frac{-c_B}{h^2} & \frac{2}{h^2} + \Psi(n_{i_1}) & \frac{-1}{h^2} & 0 & \cdots & \cdots & \cdots & \vdots \\ \vdots & \vdots & \ddots & \frac{-1}{h^2} & \ddots & \ddots & \ddots & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \frac{-1}{h^2} & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & 0 & \cdots & \frac{-1}{h^2} & \frac{2}{h^2} + \Psi(n_{i_2}) & \frac{-c_B}{h^2} & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \cdots & \cdots & 0 & 1 & 0 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \cdots & \cdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ i_1 \\ \vdots \\ \vdots \\ i_2 \\ \\ \\ \end{matrix}$$

$$b = \begin{pmatrix} c_B \\ \vdots \\ c_B \\ 0 \\ \vdots \\ \vdots \\ 0 \\ c_B \\ \vdots \\ c_B \end{pmatrix} \begin{matrix} \\ \\ \\ i_1 \\ \vdots \\ \vdots \\ i_2 \\ \\ \\ \end{matrix}$$

Et donc la résolution de ce système revient à résoudre celui compris entre les indices i_1, i_2.
i_1 : Le premier indice pour lequel $n_i > 0$.
i_2 : Le dernier indice pour lequel $n_i > 0$.
Et donc notre système sera de taille $i_2 - i_1 + 1$
On a mis donc en oeuvre une procédure **Assemblage** qui, étant donné n_i, permet de trouver les indices i_1, i_2 et fait un stockage profil pour la matrice du système compris entre $[i_1, i_2]$ et aussi calcule le second membre du même

système.
La nouvelle matrice A et le nouveau second membre b s'écrivent :

$$A = \frac{1}{h^2}\begin{pmatrix} 2+4h^2n_{i_1} & -1 & 0 & \cdots & \cdots & 0 \\ -1 & \ddots & \ddots & 0 & \cdots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & \cdots & 0 & -1 & 2+4h^2n_{i_2} \end{pmatrix}$$

$$b = \begin{pmatrix} \frac{c_B}{h^2} \\ 0 \\ \vdots \\ \vdots \\ 0 \\ \frac{c_B}{h^2} \end{pmatrix}$$

On peut constater que la matrice A est symétrique à diagonale strictement dominante. On déduit que A est symétrique définie positive.
On utilise un stockage profil qui est le plus adapté pour notre matrice afin de résoudre notre système linéaire.

Algorithm 1 algorithme de résolution de Ac = b avec décomposition LdL^t

Stockage profil avec la procédure **Assemblage**
Calculer la décomposition de Crout LDL^t avec la procédure **Gauss sprof**
Résolution du système avec la procédure **Résout sprof**
résoudre $Lz = b$
résoudre $Dy = z$
résoudre $L^t x = y$

2.1.3 Simulations numériques (à n fixé)

On fixe les paramètres suivants pour les simulations :
$c_B = 1$
$\Psi(n) = \begin{array}{ll} 2 & n \geq 1 \\ 1 & n \leq 1 \end{array}$
$L = 100$
On prend plusieurs choix de n pour faire plusieurs simulations. Pour fabriquer ces cas tests on s'est inspiré des courbes données dans le papier de Benoit Perthame, Min Tang et Nicolas Vauchelet.

1. Cas :
   $$n(x) = \begin{pmatrix} 0 & -5 \leq x \leq -0.5 \\ 0.8 & -0.5 \leq x \leq 0.5 \\ 0 & 0.5 \leq x \leq 5 \end{pmatrix}$$
   Les résultats obtenus sont présentés dans la figure 2.1 :

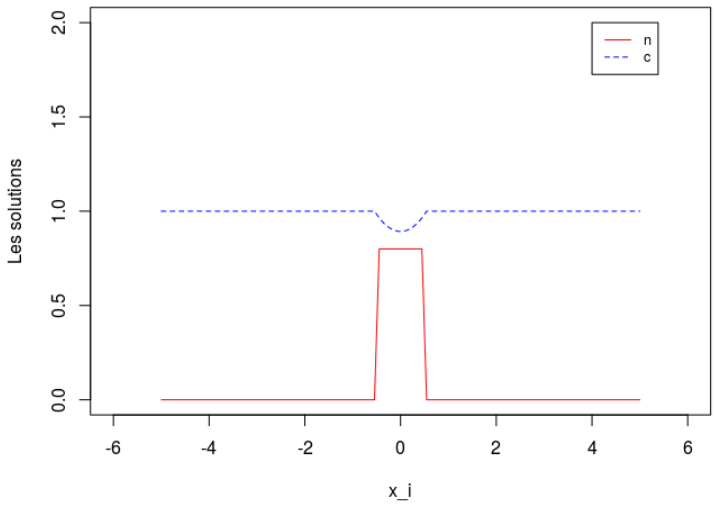


FIGURE 2.1 – La donnée n et la Solution associée c pour le cas 1

2. Cas : $n(x) = \begin{pmatrix} 0 & -5 \leq x \leq -0.9 \\ 1 & -0.9 \leq x \leq 0.9 \\ 0 & 0.9 \leq x \leq 5 \end{pmatrix}$

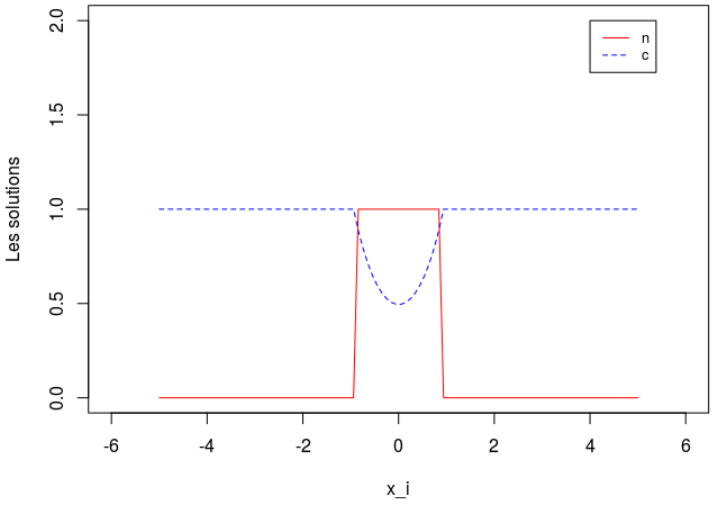


FIGURE 2.2 – La donnée n et la Solution associée c pour le cas 2

Les résultats obtenus sont présentés dans la figure 2.2 :

3. Cas :

$$n(x) = \begin{pmatrix} 1 & -1.5 \leq x \leq -0.8 \\ 1 & 0.8 \leq x \leq 1.5 \\ 1 - 0.9cos(\frac{\pi}{1.6}x) & sinon \end{pmatrix}$$

Les résultats obtenus sont présentés dans la figure 2.3 :

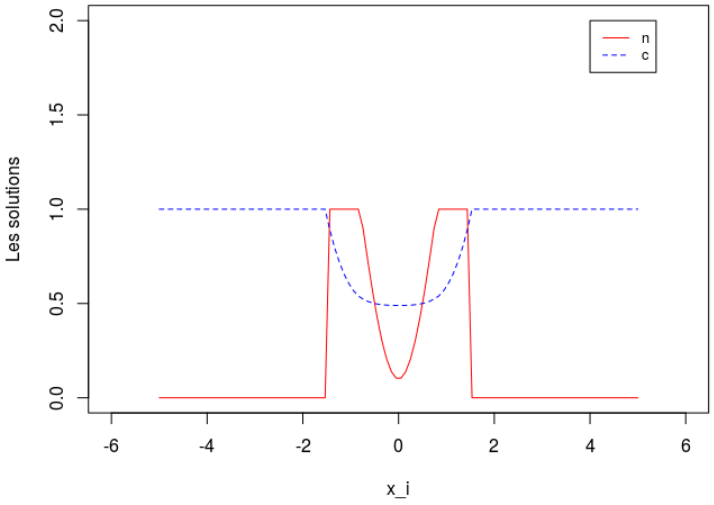


FIGURE 2.3 – La donnée n et la Solution associée c pour le cas 3

Le schéma utilisé est le schéma étudié en cours.
On se contente ici de comparer les résultats obtenus pour différentes valeurs de n avec ceux du papier de Benoit Pethame, Min Tang et Nicolas Vauchelet.
On obtient des courbes similaires.

Chapitre 3

3.1 Méthode numérique pour le calcul de n

3.1.1 Présentation du schéma

On rappelle que l'équation sur n s'écrit :

$$\frac{\partial n}{\partial t} - div(n\nabla p(n)) = nG(c)$$

On va réécrire l'équation pour que la pression p n'intervienne pas .
On a :

$$\nabla p = \partial x(p) = \partial_x(n^\gamma) = \gamma n^{\gamma-1}\partial_x n$$

$$\Rightarrow n\partial_x p(n) = \gamma n^\gamma \partial_x n = \frac{\gamma}{\gamma+1}\partial_x(n^{\gamma+1})$$

$$\Rightarrow div(\nabla p) = \partial_{xx}(n^{\gamma+1})$$

Et donc notre équation s'écrit :

$$\partial_t(n) - \partial_{xx}(n^{\gamma+1}) = nG(c) \ (2)$$

Elle dépend à la fois du temps et de l'espace, on discrétise tout d'abord notre intervalle d'espace $\Omega = [-5, 5]$ en L+2 points et avec un pas $h = \frac{10}{L+1}$ et notre intervalle de temps $T = [0, 1]$ avec un pas Δt.
On note n_i^k, $i = 1, ..., L$ l'approximation de $n(ih, kDt)$.
On étudie cette équation en la découplant de celle de c et on utilise des conditions aux bords de type Dirichlet homogène, tout en conservant sa non linéarité.

On a une équation de diffusion parabolique. On veut éviter d'avoir une condition contraignante sur le pas de temps, d'où vient le choix du schéma Euler implicite. Et donc notre schéma s'écrit sous la forme suivante :

$$\frac{n_i^{k+1} - n_i^k}{\Delta t} - \frac{\gamma}{\gamma+1}\frac{(n_{i+1}^{k+1})^{\gamma+1} - 2(n_i^{k+1})^{\gamma+1} + (n_{i-1}^{k+1})^{\gamma+1}}{h^2} = n_i^{k+1} G(c_i^{k+1}) (3)$$

$$n_0 = 0 \qquad \text{et} \qquad n_{L+1} = 0$$

Contrairement au système d'équations linéaires de l'équation sur c on peut remarquer que ce schéma numérique conduit à un système d'équations non linéaires :$F_k(n_i^{k+1}) = 0$, avec :

$$F_k(n_i^{k+1}) = \frac{n_i^{k+1} - n_i^k}{\Delta t} - \frac{\gamma}{\gamma+1}\frac{(n_{i+1}^{k+1})^{\gamma+1} - 2(n_i^{k+1})^{\gamma+1}(n_{i-1}^{k+1})^{\gamma+1}}{h^2} - n_i^{k+1} G(c_i^{k+1})$$

la résolution de ce système d'équations non-linéaires (2) semble beaucoup plus délicate, et se fait à l'aide d'une méthode de Newton.

3.1.2 Méthode de Newton

Présentation :

La méthode de Newton pour la résolution d'un système d'équations non linéaires est une généralisation de l'algorithme de Newton pour la recherche d'un zéro d'une fonction unidimensionnelle. Le fait qu'un algorithme pour un problème unidimensionnel puisse efficacement se généraliser au cas multidimensionnel est une situation exceptionnelle. Cette méthode est souvent appelée aussi méthode de NEWTON-RAPHSON.

Soit $F : R^n \longrightarrow R^n$ une application dérivable. Pour déterminer une approximation numérique des solutions de $F(t) = 0$, la méthode de Newton part d'une solution approchée x et remplace l'équation $F(t) = 0$ par l'équation approchée :

$$F(x) + (t - x)JF(x) = 0$$

d'où la solution :

$$t = x - JF(x)^{-1}F(x)$$

Cette formule n'a un sens que si $JF(x)^{-1}$(la jacobienne de F) existe et qu'elle soit non nulle pour tout x.

Dans un contexte plus général si $t \in I$ on peut encore recommencer notre

algorithme en partant de la valeur initiale donnée par t d'où une suite définie par la relation de récurrence :

$$x_{n+1} = x_n - JF(x_n)^{-1}F(x_{n+1})$$

$$x_0 \in I$$

Le choix de x_0 étant fondamentale dans la méthode pour assurer sa convergence, elle doit être proche de la solution envisagée.

Convergence :

Le taux de convergence d'une méthode itérative est définie comme :

$$\lim_{k \longrightarrow \infty} \frac{|| \ e^{k+1} \ ||}{|| \ e^k \ ||^r}$$

- $r = 1$ et $c < 1$: Convergence linéaire ceci correspond à un gain de précision par itération d'un nombre de digits constant.
- $r > 1$: Convergence super-linéaire ceci implique que l'accroissement de la précision par itération va en augmentant.
- $r = 2$: Convergence quadratique la précision double à chaque itération.

La méthode de Newton converge quadratiquement sous certaines conditions et elle vérifie :

$$|| \ x_{k+1} - \overline{x} \ || \leq \beta \ || \ x_k - \overline{x} \ ||^2, \beta > 0$$

Algorithm 2 Algorithme de Newton

Données : x_0 : solution initiale
ϵ : tolérance donnée
Nmax : nombre maximal d'itérations
k = 0
while ($k \leq Nmax$ et $|| \ x_{k+1} - x_k \ || > \epsilon$) **do**
  $k = k + 1$
  résoudre $JF(x_k)z_k = -F(x_k)$
  $x_{k+1} = x_k + z_k$
end while

La complexité algorithmique de la méthode de Newton est déterminée par la résolution du système $JF(x_k)z_k = -F(x_k)$ et du calcul de l'inverse de

la matrice Jacobienne $JF(x_k)^{-1}$. Ceci peut apparaître comme un désavantage de la méthode de Newton. Cependant dans la pratique si on utilise une méthode de résolution directe pour $JF(x_k)z_k = -F(x_k)$ et lorsqu'on utilise les dérivées analytiques, la méthode de Newton a une complexité comparable à celle des méthodes itératives pour la résolution des systèmes linéaires. Sachant que le nombre d'itérations de la méthode de Newton est nettement inférieur que le nombre d'itérations pour celles-ci.
Pour résoudre ce système on utilise **Algorithm 1**

Résolution de notre système :

La méthode de Newton consiste à résoudre à chaque itération un système non linéaire.
Notre système (3) s'écrit sous la forme :

$$\frac{n_i^{k+1} - n_i^k}{\Delta t} - \frac{\gamma}{\gamma+1}\frac{(n_{i+1}^{k+1})^{\gamma+1} - 2(n_i^{k+1})^{\gamma+1}(n_{i-1}^{k+1})^{\gamma+1}}{h^2} - n_i G(c^{k+}) = 0$$

Celui-ci conduit à un système : $F_k(n_i^{k+1}) = 0$
avec :

$$\begin{aligned} F_k(x) &= x - n^k + \Delta t \frac{\gamma}{\gamma+1} A x^{\gamma+1} - \Delta t x G \\ &= (I_d - \Delta t G)x + \Delta t \frac{\gamma}{\gamma+1} A x^{\gamma+1} - n^k \end{aligned}$$

I_d : La matrice identité de taille L

$$G = \begin{pmatrix} Gc(c_1^k) & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & Gc(c_L^k) \end{pmatrix}$$

$$A = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & \cdots & \cdots & 0 \\ -1 & \ddots & \ddots & \cdots & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & \cdots & -1 & 2 \end{pmatrix}$$

$$n^k = \begin{pmatrix} n_1^k \\ \vdots \\ n_i^k \\ \vdots \\ n_L^k \end{pmatrix}, x^{\gamma+1} = \begin{pmatrix} x_1^{\gamma+1} \\ \vdots \\ x_i^{\gamma+1} \\ \vdots \\ x_L^{\gamma+1} \end{pmatrix}$$

Notre système est donc de taille L Pour le résoudre on utilise la méthode de Newton pour résoudre : $F_k(x) = 0$

$$F_k : R^L \longrightarrow R^L$$

$$F_k(x) = (I_d - G)x + \Delta t \frac{\gamma}{\gamma+1} A x^{\gamma+1} - n^k$$

On calcule en suite La jacobienne de notre système qui s'écrit :

$$JF_k = I_d - G + \Delta t \frac{\gamma}{\gamma+1} AD$$

Avec :

$$D = \begin{pmatrix} x_1^\gamma & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & x_L^\gamma \end{pmatrix}$$

Notre Jacobienne s'ecrit donc :

$$JG_k = \begin{pmatrix} 1 - Gc(c_1^k) & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & Gc(c_L^k) - 1 \end{pmatrix} +$$

$$\frac{\Delta t \gamma}{\gamma+1} \begin{pmatrix} \frac{2x_1^\gamma}{h^2} & \frac{-x_1^\gamma}{h^2} & \cdots & \cdots & 0 \\ \frac{-x_1^\gamma}{h^2} & \ddots & \ddots & \cdots & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \frac{-x_{L-1}^\gamma}{h^2} \\ 0 & \cdots & \cdots & \frac{-x_{L-1}^\gamma}{h^2} & \frac{2x_L^\gamma}{h^2} \end{pmatrix}$$

Donc notre matrice Jacobienne JG_k est la somme de deux matrices symétriques et à diagonales strictement dominantes donc elle est symétrique définie positive donc elle est inversible d'où la validité de notre méthode de Newton pour résoudre ce système.
Nous testons ainsi la validité de notre méthode de Newton

3.1.3 Simulations numériques

Pour vérifier la convergence de notre méthode de Newton et pour réaliser quelques simulations numériques on a modifié notre problème (3) en choisissant les valeurs de quelques paramètres tout en préservant la nature et la non-linéarité de notre problème.
On a tout d'abord commencé avec les choix suivants :
$G = 0$, $\gamma = 1$
Et donc le problème s'écrit sous la forme :

$$\partial_t(n) - \frac{1}{2}\partial_{xx}(n^2) = 0$$

Une solution exacte est donnée par :

$$n(t,x) = max(t-x, 0)$$

Pour les conditions au bords et initiales données par :

$$n(t,5) = 0 \text{ et } n(t,-5) = 5 + t$$

$$n(0,x) = max(-x, 0)$$

Pour vérifier notre schéma numérique nous adaptons notre code à ce système et nous ajoutons la condition aux bords. Pour le moment on n'avait utilisé que des conditions aux bords Dirichlet homogène. Ici nos conditions de Dirichlet ne sont pas homogènes.
Sur les figures suivantes on observe la solution numérique ainsi que la solution exacte nous pouvons noter que les deux solutions sont quasiment confondues partout.
On a choisi comme paramètres pour la représentation : $\Delta t = 0.001$, $L = 1000$.
Sur les figures $3.1, 3.2, 3.3$ et 3.4 suivants on observe l'évolution de la solution numérique et la solution exacte à plusieurs instants.

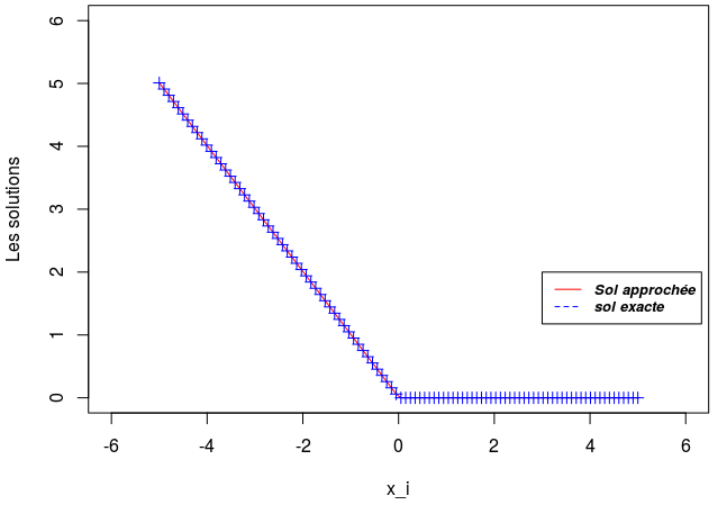


FIGURE 3.1 – à l'instant t=0.01

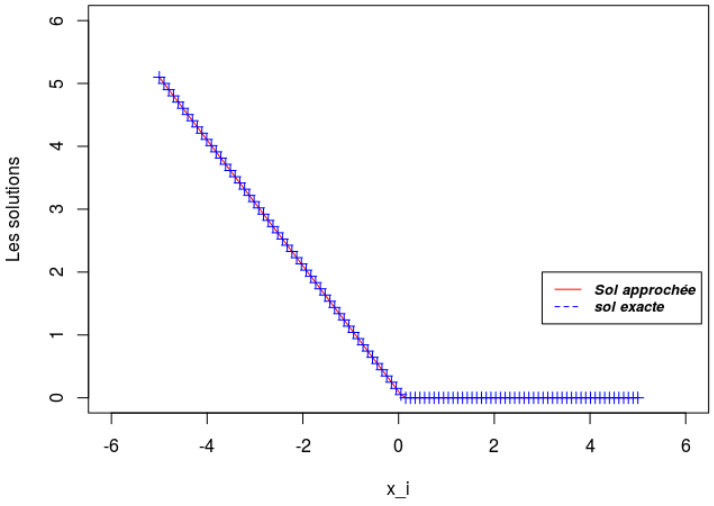


FIGURE 3.2 – à l'instant t=0.1

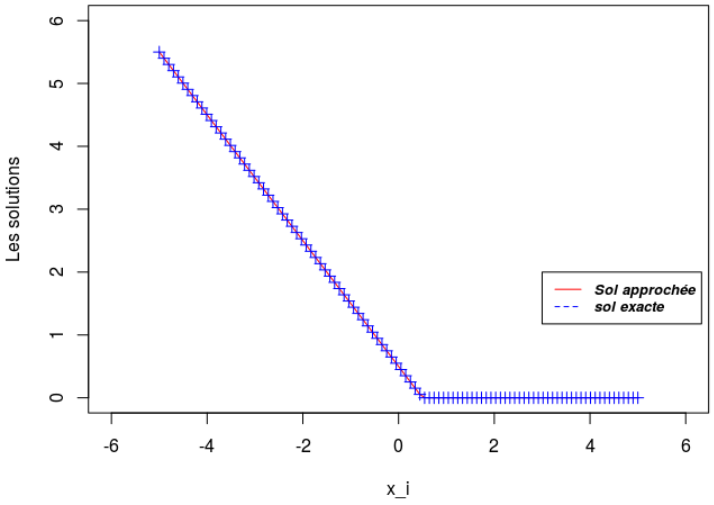


FIGURE 3.3 – à l'instant t=0.5

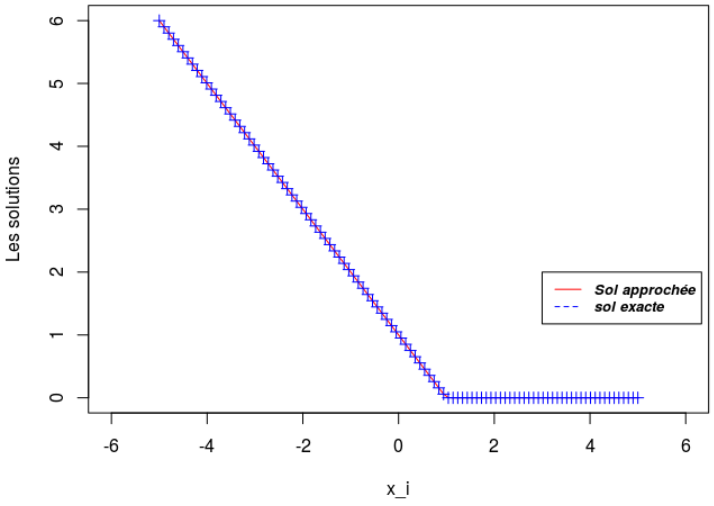


FIGURE 3.4 – à l'instant final

Nous avons également étudié l'ordre de convergence de la méthode de Newton.
Soit n^k la solution à la kième itération $k \in N$ de la méthode de Newton et $\overline{n}$ solution exacte choisie pour notre problème, ainsi on vérifie la convergence quadratique de notre méthode de Newton en calculant le rapport

$$\frac{|| \, n^{k+1} - \bar{n} \, ||}{|| \, n^k - \bar{n} \, ||^2} \leq \beta, \quad \beta > 0$$

Nous traçons ainsi ce rapport en fonction du nombre d'itérations de la méthode de Newton.

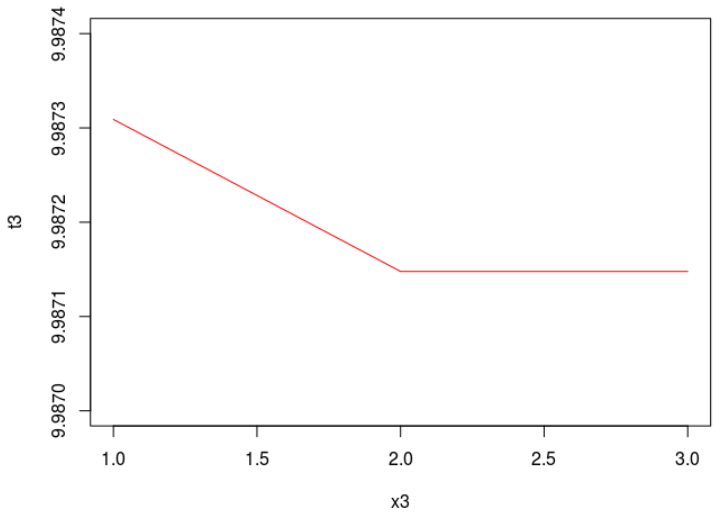
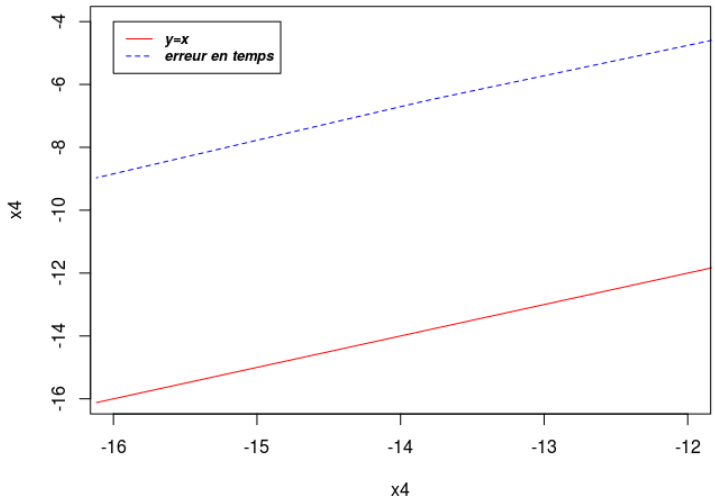


FIGURE 3.5 – schema montrant l'ordre de convergence

On observe donc bien que le rapport $\frac{||n^{k+1}-\overline{n}||}{||n^k-\overline{n}||^2}$ est donc bien fini.
Nous déduisons donc la convergence quadratique de la méthode de Newton.
On conclut donc la bonne implémentation de notre méthode de Newton.

Maintenant, on cherche à calculer l'erreur en calculant la différence entre la solution exacte et la solution numérique, pour vérifier la convergence d'ordre 1 en temps de notre schéma numérique. Pour cela on calcule la norme infinie de notre erreur avec la procédure **norm inf** qu'on déjà implémenté sur Fortran , en fixant notre pas d'espace et en multipliant chaque fois notre pas de temps par 10^{-1}.
La figure 3.6 représente en échelle logarithmique l'erreur en fonction du pas de temps, aussi on représente sur la même figure la fonction $y = x$.
On observe bien que l'ordre de convergence est 1 en temps $O(\Delta t)$.



(1).png

FIGURE 3.6 – schéma montrant l'ordre de convergence en temps

Chapitre 4

4.1 Le modelé couplé

4.1.1 Structure du code

Après avoir vérifié la bonne implémentation de chacune des nos 2 équations , nous allons implémenter maintenant le couplage des deux équations qui s'écrit :

$$\begin{cases} -\triangle c + \Psi(n,c) = 0. & (1) \\ \frac{\partial n}{\partial t} - div(n\nabla p(n)) = nG(c) & (2) \end{cases}$$

. Autrement notre système couplé s'écrit :

$$\begin{cases} -\triangle c + \Psi(n)c = 0 \quad x \in (n > 0) \\ c = C_B \quad x \in (n = 0) \\ \partial_t(n) - \partial_{xx}(n^{\gamma+1}) = nG(c) \quad x \in]-5,5[\\ n(t,-5) = 0 \quad n(t,5) = 0. \end{cases}$$

On discrétise tout d'abord notre intervalle d'espace $\Omega = [-5,5]$ en L+2 points , avec un pas $h = \frac{10}{L+1}$ pour les 2 équations. Aussi on discrétise notre intervalle de temps $T = [0,1]$ avec un pas Δt pour la deuxième équation et donc nos deux schémas numériques couplés s'écrit :

$$\begin{cases} -\frac{c_{i+1}^{k+1} - 2c_i^{k+1} + c_{i-1}k+1}{h^2} + \Psi(n_i^k)c_i^{k+1} = 0, \quad \forall i \in J, \quad J = \{i/n_i \geq 0\} \\ c_i = C_B, \quad \forall i \in J^{'}, \quad J^{'} = \{i/n_i = 0\} \\ \frac{n_i^{k+1} - n_i^k}{\Delta t} - \frac{\gamma}{\gamma+1}\frac{(n_{i+1}^{k+1})^{\gamma+1} - 2(n_i^{k+1})^{\gamma+1} + (n_{i-1}^{k+1})^{\gamma+1}}{h^2} = n_i^{k+1}G(c_i^{k+1}) (3) \\ n_0 = 0 \quad , n_{L+1} = 0 \end{cases}$$

On lance notre calcul avec la donnée initiale n_i^0. À chaque étape on commence par résoudre un système linéaire pour trouver la concentration c_i^{k+1}, à l'instant $(k+1)$, comme dans le chapitre (2) avec comme donnée n_i^k. Puis on résout un système non linéaire pour trouver la densité des cellules malades n_i^{k+1}, au même instant $(k+1)$, en utilisant la méthode démontrée dans le chapitre (3) avec les données c_i^{k+1} et n_i^k calculées précédemment.

4.1.2 Simulation numérique

Pour la simulation numérique de notre couplage On a pris les paramètres suivants :

$$G(c) = \begin{cases} 21 & c \geq 0.6 \\ -30 & c < 0.6 \end{cases}$$

$$\Psi(n, c) = \begin{cases} 2 & n \geq 1 \\ 1 & n < 1 \end{cases}$$

$$\gamma = 2$$

$$\Delta t = 0.0001$$

tolerance de la méthode de newton : $tol = 10^{-10}$

$$L = 100$$

On prend comme solution initiale pour (2) :

$$n(x) = \begin{pmatrix} 0 & -5 \leq x \leq -0.5 \\ 0.8 & -0.5 \leq x \leq 0.5 \\ 0 & 0.5 \leq x \leq 5 \end{pmatrix}$$

On observe l'évolution de notre couplage à plusieurs instants Sur les figures 4.1 jusqu'à 4.6 suivants :

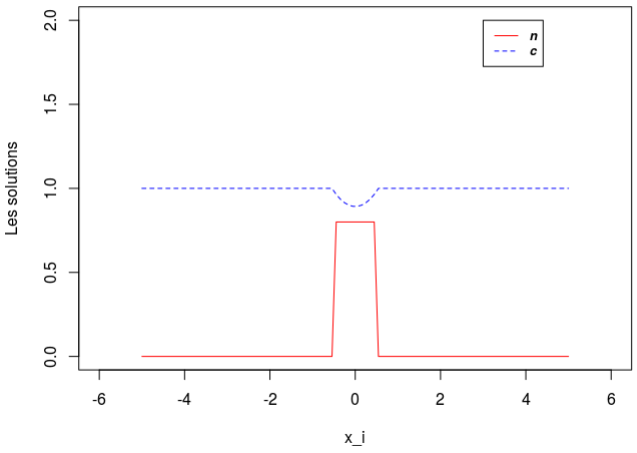


FIGURE 4.1 – Couplage à l'état initial

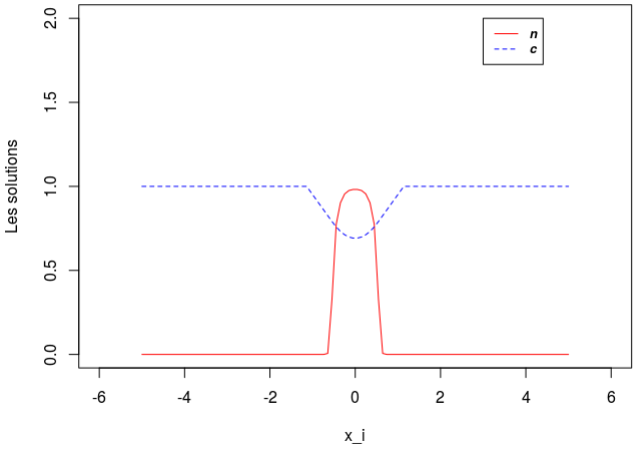


FIGURE 4.2 – Couplage à l'instant t =0.01

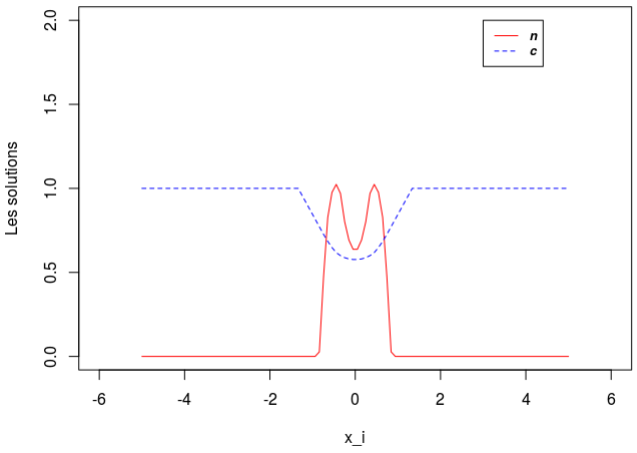


FIGURE 4.3 – Couplage à l'instant t =0.05

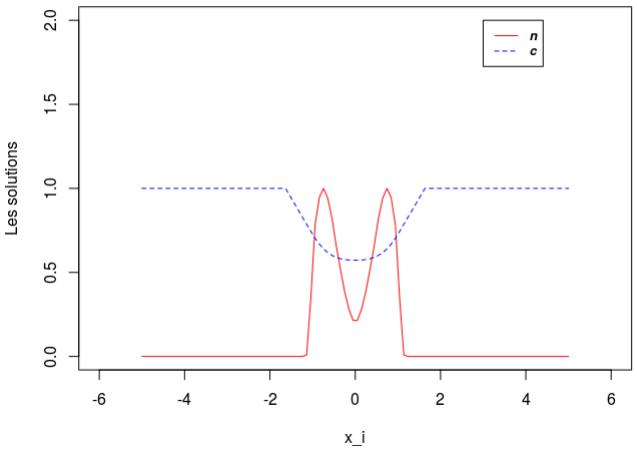


FIGURE 4.4 – Couplage à l'instant t =0.1

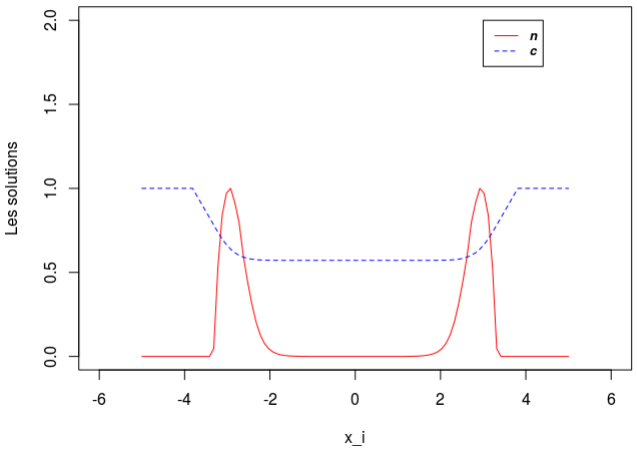


FIGURE 4.5 – Couplage à l'instant t =0.5

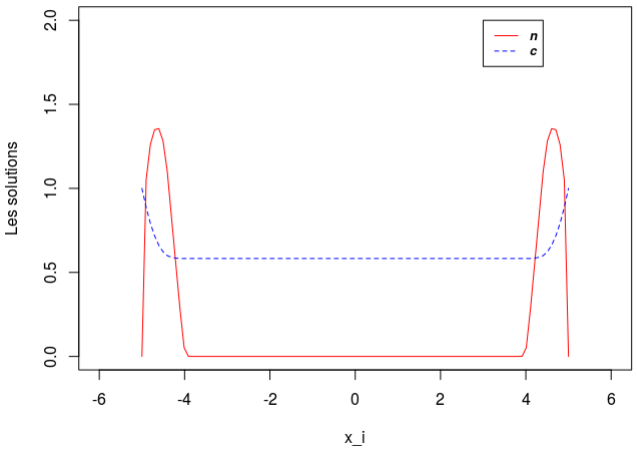


FIGURE 4.6 – Couplage à l'état final

Pour vérifier la convergence de la méthode de Newton dans ce cas, on

a représenté le nombre d'itérations de la méthode à chaque instant dans la figure 4.7.

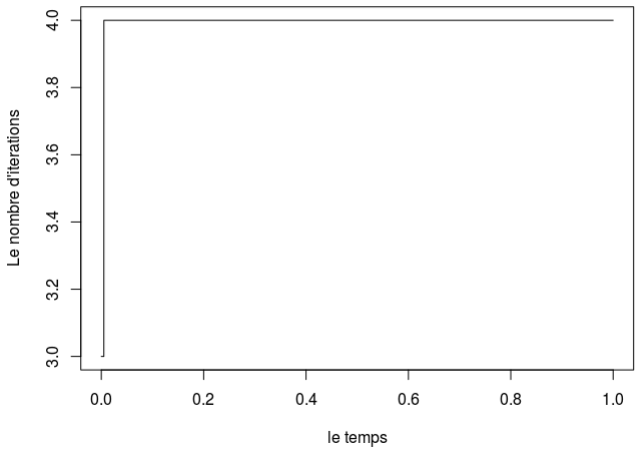


FIGURE 4.7 – Nombre d'itérations à chaque instant

On fait une autre simulation en conservant les mêmes paramètres que précédemment et en prenant $\gamma = 50$ On obtient les figures de 4.8 à 4.13 :

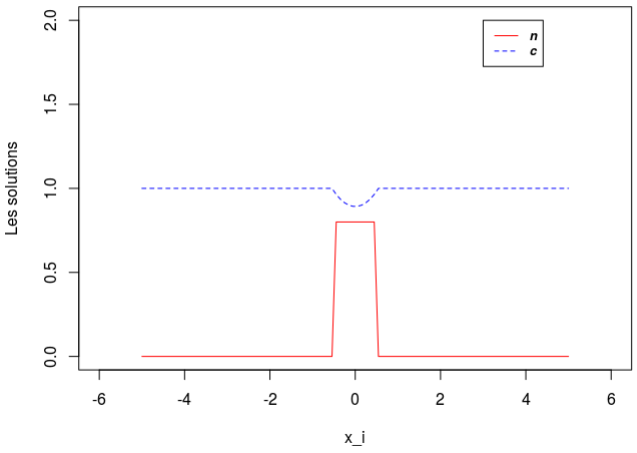


FIGURE 4.8 – Couplage à l'état initial

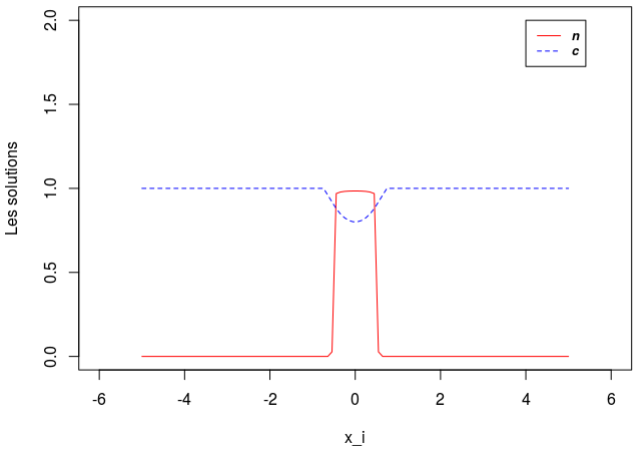


FIGURE 4.9 – Couplage à l'instant t =0.01

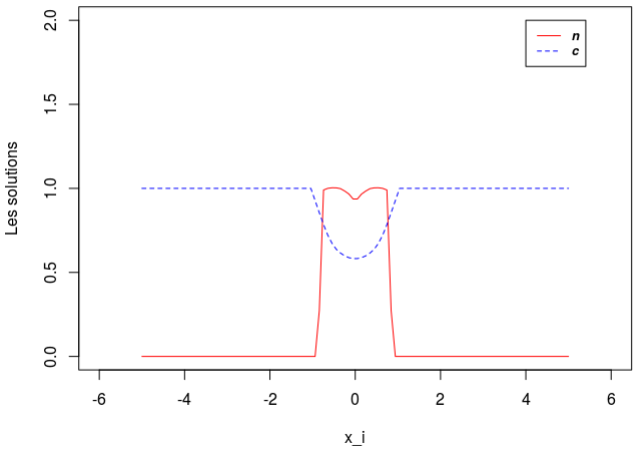


FIGURE 4.10 – Couplage à l'instant t =0.05

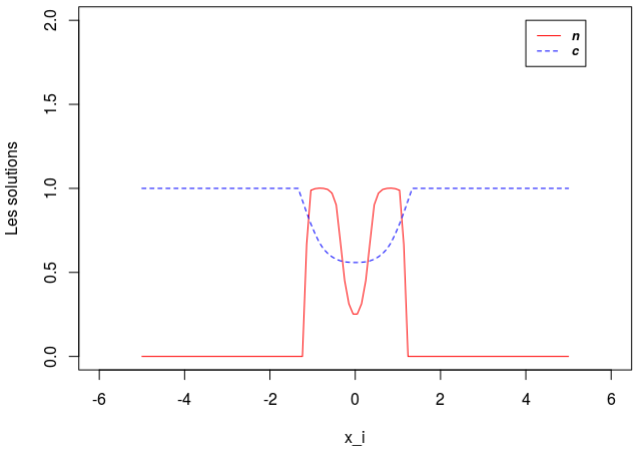


FIGURE 4.11 – Couplage à l'instant t =0.1

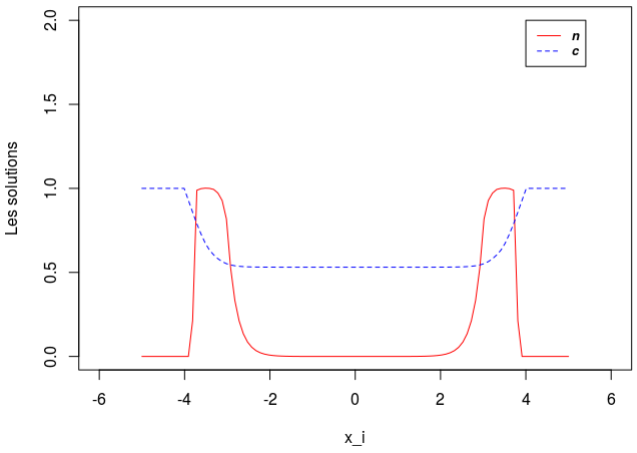


FIGURE 4.12 – Couplage à l'instant t =0.5

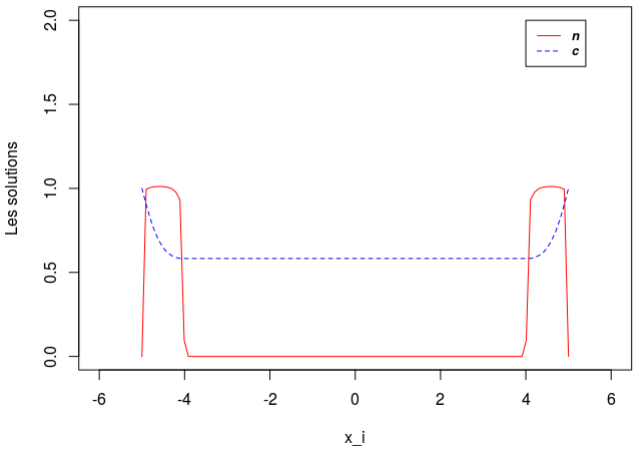
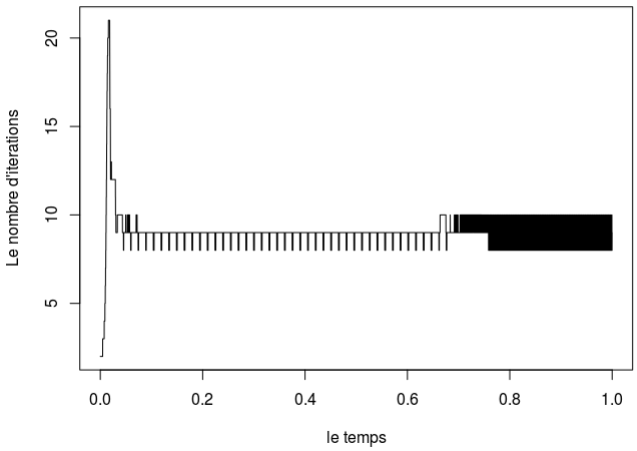


FIGURE 4.13 – Couplage à l'état final



iter.png

FIGURE 4.14 – Nombre d'itérations à chaque instant

L'évolution de notre couplage dans les deux simulations précédentes montre une diminution de la concentration des nutriments aux cours du temps, ce qui provoque une chute de la densité des cellules malades, ceci s'explique par le terme $G(c)$ d'accroissement des cellules malades qui se change en valeur négative avec la diminution de c pour provoquer cette chute qu'on observe sur les figures.

Annexe A

A.1 Programme principale du couplage

```
program TER3
  use TER_module
  implicit none
  double precision , allocatable , dimension(:,:) :: Con ,Newt
  double precision , allocatable , dimension(:) :: Mtab, C,NA,b,U0,NB,C1,y
  integer , allocatable , dimension(:) :: indiag
  double precision :: cb,Dt,tol,valag,valad
  integer ::i,L,i1,i2,choix,gamma,j,Nmax
  tol = 0.0000000001
  gamma =50
  valag = 0.0                        ! valag et valad représente les condition
  valad = 0.0                         !          au bords pour n
  print*,"donner la valeur de L "  ! Le nombre de points
  read*,L
  print*,"donner la valeur de cb " !  La valeur de  c sur les bords
  read*,cb
  print*,"donner la valeur de choix " ! Pour choisir la solution n initiale
  read*,choix
  print*,"donner la valeur de Dt "   ! Le pas de temps
  read*,Dt
  Nmax =(1./Dt)                      !  Nombre d'itération à faire
  allocate(NA(L))
  allocate(y(L+2))
  allocate(Newt(Nmax+1,L+2))   ! Matrice pour stocker la solution de n
                                    à différents instants.
```

```
  allocate(Con(Nmax,L+2))      ! Matrice pour stocker la solution de c
                                    à différents instants.
  y = 0
  Newt = 0
  Con  = 1
  do i = 1,L
     NA(i) = n(-5+(i*10.)/dble(L+1),choix)      !solution initiale pour n
  end do
 Newt(1,2:L+1)=NA
  do i = 1,Nmax       ! mise en oeuvre du couplage
     call assemblage(choix,mtab,indiag,L,Newt(i,:),cb,b,i1,i2)
     print*,i1 ,i2
     call Gauss_sprof(indiag , MTAB)
     call resout_sprof(b , MTAB , INDIAG)  !Résolution de l'équation sur C
     do j = 1,L+2
        if(j >= i1 .and. j <= i2 ) then
         con(i,j) = b(j-i1+1)
        else
           con(i,j) = Cb
        end if
    end do
     Newt(i+1,2:L+1) = newton(L,Dt,Newt(i,2:L+1),tol,Con(i,2:L+1),y,gamma,valag,va
      end do
 ! call affiche(con)
 ! call affiche(Newt)
 call ecrire("concentration.txt",Con)
 call ecrire("densite1.txt",Newt)

end program TER3
```

A.2 Module du TER

```
module TER_module
  implicit none
  interface saisir
     module procedure saisir_m
     module procedure saisir_v
```

```
  end interface saisir

  interface affiche
     module procedure affiche_m
     module procedure affiche_v
     module procedure affiche1_v
  end interface affiche

  interface lire
     module procedure lire_m
     module procedure lire_v
  end interface lire

  interface ecrire
     module procedure ecrire_m
     module procedure ecrire_v
  end interface ecrire




contains
  !#########################################################################!


  subroutine saisir_m(matrice)
    integer , dimension(:,:),allocatable , intent(out):: matrice
    integer                                            :: i ,j,n,m

    print*, "donnez n et m"
    read*, n,m
    allocate(matrice(n,m))
    do j=1,m
       do i=1,n
          print*,  i,j
          read*, matrice(i,j)
       end do
    end do
```

```
  end subroutine saisir_m



  !#########################################################################!



  subroutine affiche_m(matrice)
    double precision, dimension(:,:) , intent(in) :: matrice
    integer                                       :: k,n
    n = size(matrice,1)
    do k=1,n
       print*, matrice(k,:)
    end do
  end subroutine affiche_m

!#########################################################################!

  subroutine ecrire_m(nom_du_fichier,matrice)
    double precision   , dimension(:,:) , intent(in):: matrice
    integer                                       :: i , n

    character(len=*)                              :: nom_du_fichier
    n = size(matrice,1)

    open (unit=1 , file = nom_du_fichier ,status = "replace", action="write")
    ! write(1,*)shape(matrice)
    do i=1,n
       write(1,*)matrice(i,:)
    end do
  end subroutine ecrire_m

  !#########################################################################!
 subroutine saisir_v(vecteur)
    double precision, dimension(:),allocatable , intent(out)::vecteur
    integer                                                  :: i ,n
```

```
  print*, "donnez n  "
  read*, n
  allocate(vecteur(n))

  do i=1,n
     print*,  i
     read*, vecteur(i)

  end do

end subroutine saisir_v

!#########################################################################!


subroutine affiche1_v(vecteur)
  integer , dimension(:) , intent(in) :: vecteur
  integer                             :: k,n
  n = size(vecteur)
  do k=1,n
     print*, vecteur(k)
  end do
end subroutine affiche1_v

!#########################################################################!


subroutine affiche_v(vecteur)
  double precision , dimension(:) ,allocatable, intent(in) :: vecteur
  integer                                                  :: k,n
  n = size(vecteur)
  do k=1,n
     print*, vecteur(k)
  end do
end subroutine affiche_v

!#########################################################################
```

```
subroutine lire_v(nom_du_fichier , vecteur)
  double precision , dimension(:),allocatable , intent(out):: vecteur

  character(len=*)                                          :: nom_du_fichier
  integer                                                   :: taille

  open (unit = 1 , file = nom_du_fichier ,action = "read")
  read(1,*) taille
  allocate( vecteur(taille))
  read(1,*)vecteur


end subroutine lire_v

!#############################################################################

subroutine ecrire_v(nom_du_fichier,vecteur)
  double precision , dimension(:) , intent(in):: vecteur
  integer                                     :: i , n

  character(len=*)                            :: nom_du_fichier
  n = size(vecteur,1)

  open (unit=1 , file = nom_du_fichier ,status = "replace", action="write")
  write(1,*)shape(vecteur)
  do i=1,n
     write(1,*)vecteur(i)
  end do
end subroutine ecrire_v

!#############################################################################

function norme1(V)

  double precision   , dimension(:)                         :: V
  double precision                                          ::  norme1
  norme1 =  sum(abs(V))
```

```
end function norme1

!#############################################################################

function norme_inf(V)
  implicit none
  integer                                    :: i ,n
  double precision, dimension(:)             :: V
  double precision                           :: norme_inf
  double precision                           :: max
  n= size(V)
  max =abs( V(1))
  do i=2,n,1
     if( abs(max)< abs(V(i))) then
        max =abs( V(i))
     else
        max = max
     end if
  end do
  norme_inf = max
end function norme_inf

!#############################################################################

subroutine normeinf_m(A,norminf)
  integer                                                 :: i ,j,m, n
  integer  ,allocatable , dimension(:,:) , intent(in) :: A
  double precision                                        ::  norminf
  double precision                                        :: max
  max = sum(abs(A(:,1)))
  do j=2,n
     if(max < sum(abs(A(:,j))))then
        norminf  =  sum(abs(A(:,j)))
     else
        max = max
     end if
  end do
```

```
end subroutine normeinf_m

!################################## Stokcage profil####################

SUBROUTINE profil_sprof(matrice,Mtab,INDIAG)
  IMPLICIT NONE
  INTEGER                                                          :: i, j, n
  DOUBLE PRECISION, DIMENSION(:,:), INTENT(IN)                    :: matrice
  DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE, INTENT(OUT)         :: Mtab
  INTEGER, DIMENSION(:), ALLOCATABLE, INTENT(OUT)                 :: INDIAG

  n = SIZE(matrice(:,1))
  ALLOCATE(INDIAG(n))
  INDIAG(1)=1
  DO i=2,n                          !Itération sur les lignes
     j = 1
     DO WHILE (matrice(i,j)==0)     !Test si le coeff (i,j) est nul
        j = j + 1                   !Le prochain tour de boucle va vérifier
                                         la colonne suivante
     END DO
     INDIAG(i)=INDIAG(i-1)+i-j+1    !On définit INDIAG de manière récursive
  END DO
  ALLOCATE(Mtab(INDIAG(n)))
  Mtab(1) = matrice(1,1)
  DO i=2,n
     Mtab(INDIAG(i-1)+1:INDIAG(i)) = matrice(i,i+1-(INDIAG(i)-INDIAG(i-1)):i)
  END DO

END SUBROUTINE profil_sprof
!################## Fonction pour choisir n ######################

function  n(x,choix)
  implicit none
  integer  :: choix
  double precision     :: n,x
  select case(choix)
  case(1)
```

```
      if ( -0.5 < x  .and. x < 0.5)then
         n = 0.8
      else
         n = 0
      end if
   case(2)
      if(-5.0 <= x  .and. x <= -0.9)then
         n = 0
      else if ( -0.9 < x  .and. x < 0.9)then
         n = 1
      else
         n = 0
      end if

   case(3)
      if(-5.0 <= x  .and. x <= -1.5)then
         n = 0
      else if ( -1.5 < x  .and. x < -0.8)then
         n= 1
      else if ( -0.8 < x  .and. x < 0.8)then
         n= 1 - 0.9*cos(22*x/dble(14*0.8))
      else if ( 0.8 < x  .and. x < 1.5)then
         n= 1
      else
         n = 0
      end if

   case(4)
      n = 0
   end select
end function n

!############################## Decomposition de Crout LDL^t#################!
subroutine Gauss_sprof(indiag , MATAB)

   integer , allocatable , dimension (:)                        :: Front
   integer , allocatable , dimension(:), INTENT(IN)             ::  indiag
   double precision , allocatable , dimension(:),INTENT(INOUT) :: MATAB
```

```
  integer                                                    :: i , n ,j , k
  double precision                                           :: s
  n = size(indiag)
  allocate(Front(n))
  front(1)=1
  do i = 2,n
     Front(i)=i-indiag(i)+indiag(i-1)+1
     do j = front(i), i-1

        do k = max(FRONT(j),Front(i)),j-1
           MATAB(indiag(i)-i+j)=MATAB(indiag(i)-i+j)-&&
           Matab(indiag(i)-i+k)*MATAB(indiag(j)-j+k)
        end do
     end do
     do j= Front(i),i-1
        S=MATAB(indiag(i)-i+j)/MATAB(indiag(j))
        MATAB(indiag(i))= MATAB(indiag(i)) - s*MATAB(indiag(i)-i+j)
        MATAB(indiag(i)-i+j)=S
     end do
  end do


end subroutine gauss_sprof


!#####################################################!
! Stockage en profil d'une matrice symetrique definie positive
subroutine resout_sprof(b , MATAB , INDIAG)

  integer , allocatable , dimension (:)                      :: Front
  integer , allocatable , dimension(:), INTENT(IN)           ::  indiag
  double precision , allocatable , dimension(:),INTENT(INOUT) :: MATAB , b
  integer                                                    :: i , n ,j
  n=size(indiag)
  allocate(Front(n))
  front(1)=1
  do i=2,n
     Front(i)=i-indiag(i)+indiag(i-1)+1
```

```
      do j= front(i),i-1
         b(i) = b(i)-(matab(indiag(i)-i+j))*b(j)
      end do
   end do
   do i = 1,n
      b(i)=b(i)/matab(indiag(i))
   end do

   do i = n,2,-1

      do j = Front(i),i-1
         b(j)=b(j)-Matab(indiag(i)-i+j)*b(i)
      end do
   end do

end subroutine resout_sprof
!################ La densite n avec euler explicite###################!
function  Densite(Gamma,L,U0,Dt)
   implicit none
   integer                                          :: Gamma , L
   double precision , dimension(:)                  :: U0
   double precision , allocatable , dimension(:,:) :: Densite
   double precision                                 :: Dt
   double precision                                 :: h
   integer                                          :: i , t
   h = 10./(L+1)
   allocate(Densite(int(10./Dt)+1,L+2))
   do i=2,L
      Densite(1,i) = U0(i)
   end do
   Densite(1,L+2) = 0
   Densite(1,1) = 0
   do t=2,int(10./Dt)+1
      Densite(t,L+2) = 0
      Densite(t,1) = 0
      do i=2,L+1
         Densite(t,i) = Densite(t-1,i) + (Gamma*Dt*1./(Gamma+1))*1./(h**2)*&
         ((Densite(t-1,i+1))**(gamma+1)- &
```

```
            2*(Densite(t-1,i))**(gamma+1) + &
            (Densite(t-1,i-1))**(gamma+1))+&
            Dt*densite(t-1,i)
     end do
  end do
end function Densite


!###################La densité n avec la méthode de Newton################!
! Fonction pour calculer X^gamma , avec X un vecteur
function puissance(x,gamma)
  double precision , dimension(:) :: x
  double precision , dimension(:) , allocatable:: puissance
  integer                       :: i , gamma
  allocate(puissance(size(x)))
  do i=1,size(x)
     puissance(i) = (x(i))**(gamma)
  end do
end function puissance
!#####################################################!
! Fonction pour calculer le seconde membre "G"
function Gc(x)
  implicit none
  double precision     :: Gc ,x
  if( x .LT. dble(0.6)) then
     Gc = -30
  else
     Gc = 21
  end if
end function Gc

!######## La fonction G_n #######!
function  G_n(x,x0,gamma,Dt,c,L,valag,valad)
  implicit none
  double precision  ,allocatable , dimension(:)     :: G_n
  double precision  , dimension(:)                  :: x , x0 ,c
  double precision  , dimension(:), allocatable    :: G1
  double precision  , allocatable , dimension(:,:)  :: Id , A ,G
```

```
  double precision                                        :: Dt, h,valag,valad
  integer                                                 :: L,gamma,i,j
  h = 10./(L+1)
  allocate(A(L,L))
  allocate(Id(L,L))
  allocate(G_n(L))
  allocate(G1(L))
  allocate(G(L,L))
  A = 0
  Id = 0
  G = 0
  G1 = 0
  do i=1,L
     do j=1,L
        if(i == j) then
           A(i,j) = 2./(h**2)
           Id(i,j) = 1
           !  G(i,i) = Gc(c(i))
        end if
     end do
  end do
  do i=1,(L-1)
     A(i,i+1) = -1./h**2
     A(i+1,i) = -1./h**2
  end do
  G_n =  matmul((Id-Dt*G),x)-x0+(Gamma*Dt*1./(Gamma+1))*
  &matmul(A,puissance(x,(Gamma+1)))
  G_n(1) = G_n(1) - (Gamma*Dt*1./(Gamma+1))*(valag**(gamma+1))/h**2
  G_n(L) = G_n(L) - (Gamma*Dt*1./(Gamma+1))*(valad**(gamma+1))/h**2
end function G_n
!#########################################################!
! Le gradient de G_n
function  Grad_G_n(x,gamma,Dt,c,L)
  implicit none
  double precision  ,allocatable , dimension(:,:)   :: Grad_G_n
  double precision  , dimension(:)                   :: x ,c
  double precision  , allocatable , dimension(:,:)  :: Id , A , B,D,G
  double precision                                   :: Dt, h
```

```
  integer                                                  :: L,gamma,i,j
  h = 10./(L+1)
  allocate(A(L,L))
  allocate(Id(L,L))
  allocate(D(L,L))
  allocate(G(L,L))
  G = 0
  A  = 0
  Id = 0
  D  = 0
  allocate(Grad_G_n(L,L))
  do i=1,L
     do j=1,L
        if(i == j) then
           A(i,j) = 2./(h**2)
           Id(i,j) = 1.0
           D(i,j) =(x(i))**gamma
           !   G(i,i) = Gc(c(i))
        end if
     end do
  END DO
  do i=1,(L-1)
     A(i,i+1) = -1./h**2
     A(i+1,i) = -1./h**2
  end do
  Grad_G_n = Id-Dt*G  + (gamma)*Dt*matmul(A,D)
end function Grad_G_n
!#################################################################!
! La méthode de Newton pour la résolution
function newton(L,Dt,x0,tol,c,xbar,gamma,valag,valad)
  implicit none
  double precision  ,dimension(:)                  :: c,x0
  double precision  ,allocatable , dimension(:)    :: b,x,x1,newton,mtab
  integer           ,allocatable , dimension(:)    ::indiag
  double precision  ,allocatable , dimension(:,:)  :: Grad
  double precision                                 :: Dt ,tol,valag,valad
  integer                                          :: L,gamma,i,j,t
  integer                                          :: iter
```

```
  double precision  , dimension(:)                  :: xbar !La solution exacte
  allocate(newton(L))
  allocate(x(L))
  allocate(x1(L))
  allocate(b(L))
  grad = Grad_G_n(x0,gamma,Dt,c,L)
  b =  - G_n(x0,x0,gamma,Dt,c,L,valag,valad)
  call profil_sprof(grad,Mtab,Indiag)
  call Gauss_sprof(indiag, Mtab)
  call resout_sprof(b , Mtab , Indiag)
  x = x0+b
  x1 = x0
  iter = 0
  do while((norme_inf(x - x1) > tol) .and.(iter < 50))
     iter = iter +1                              ! Nombre d'iterations
     x1 = x
     grad  =  Grad_G_n(x1,gamma,Dt,c,L)
     b   = - G_n(x1,x0,gamma,Dt,c,L,valag,valad)
     call profil_sprof(grad,Mtab,Indiag)
     call Gauss_sprof(indiag,mtab)
     call resout_sprof(b , Mtab , Indiag)
     x = b+x1
     ! print*,norme_inf(x-xbar)/((norme_inf(x1-xbar))**2)
  end do
  ! print*,"Nombre d'iterations =",iter
  do i = 1,L
     newton(i) =x(i)
  end do
end function newton

!#########################################################
!L'operateur psi(n,c)
function psi(x)
  implicit none
  double precision :: x,psi
  if(x < 1) then
     psi = 1
  else
```

```
      psi = 2
    end if
  end function psi


  !################################################!
  !Procedure pour calculer le second membre b , solution c et les indices i1 ,i2
  subroutine assemblage(choix,atab,indiag,L,NA,cb,b,i1,i2)
    integer ,intent(in)                                          :: L,choix
    integer ,intent(out)                                         ::i1,i2
    double precision ,intent(in)                                 :: cb
    integer                                                      :: i ,j,k,size
    double precision                                             :: h ,h1
    integer , allocatable , dimension(:), intent(out)            :: indiag
    double precision , allocatable , dimension(:), intent(out) :: atab,b
    double precision ,dimension(:),  intent(in)                  :: NA
    h1 = 10./(L+1)
!!!!!!!!!!!! calcul de l'indice i1 !!!!!!!!!!!!!!!!!!!
    i1=0
    i = 1
    do while ( NA(i) .EQ. 0 )
       i = i +1
       i1 = i1 + 1
    end do
    i1 = i1+1
!!!!!!!!!!!!!!!calcul de l'indice i2 !!!!!!!!!!!!!!!!!!
    i= i1
    do while ( NA(i) .NE. 0 )
       i = i + 1
    end do
    i2 = i-1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    size=i2-i1+1
    xi1 = i1*h1
    xi2 = i2*h1
    allocate(indiag(size))
    allocate(Atab(2*(size)-1))
    atab=0.
```

```
    do i=1,size
       indiag(1)=1
       if(i>1) then
          indiag(i)=indiag(i-1)+2
       end if
       atab(indiag(1))=2./((h1)**2)+ psi(NA(i1-1+i))*NA(i1-1+i)
       if(i>1) then
          atab(indiag(i))=2./((h1)**2)+psi(NA(i1-1+i))*NA(i1-1+i)
          atab(indiag(i)-1) = -1./((h1)**2)
       end if
    end do
    allocate(b(size))      ! Le second membre
    do i=1,size
       if(i==1 .or. i==size) then
          b(i) = Cb/((h1)**2)
       else
          b(i) = 0
       end if
    end do

  end subroutine assemblage
end module TER_module
```