SIM202 : Le modèle de compétition-diffusion de Lotka-Volterra

Joshua Wolff, Nathan Seeleuthner et Pascal Nguyen

Mars 2020

1 Etablissement du modèle

1.1 Les équations du modèle

Le modèle de Lotka-Volterra est utilisée pour représenter des systèmes de type prédateur -proie , à savoir l'équilibre entre plusieurs espèces en compétition. Utilisée dans divers domaines tels que le comportement des animaux (d'où son nom), les équilibres chimiques, la dynamique de l'Univers...

Dans le cadre de ce projet, nous allons nous limiter à 2 entités et étudier ce modèle avec une composante de diffusion.

Soit donc u et v les tailles des populations des 2 entités U et V considérés. L évolution de la population se traduit par une équation de diffusion du type :

$$\frac{du}{dt} = \frac{\partial u}{\partial t} - c\Delta u = S$$

Où S est un terme source de couplage lié aux paramètres d'évolution (mortalité, prédation) et c le coefficient de diffusion. On va donc définir c_u et c_v les coefficients de diffusion des espèces U et V.

On est donc face à un problème du type :

$$\begin{cases} \frac{\partial u}{\partial t} - c_u\Delta u = S_u \\ \frac{\partial v}{\partial t} - c_v\Delta v = S_v \end{cases}$$

1.2 Couplage des équations

On va définir les termes sources de la façon suivante, en normalisant les dimensions :

$$\begin{cases} S_u = u(1 - u - cv) \\ S_v = v(a - v - bu) \\ c_u = 1 \\ c_v = d \\ \Omega = \mathbb{R}^2 \end{cases}$$

On restreindra par la suite le domaine de résolution spatial à $\Omega = [0, 2]$x$[0, 2]$.

1.3 Conditions aux limites et conditions initiales

Nous restreindrons le domaine d'étude à l'intervalle borné $\Omega = [0,2]$x$[0,2]$ et nous imposerons des conditions limite de Dirichlet. Les conditions initiales seront appliquées au cas par cas.

2 L'objet Vecteur

2.1 Architecture

L'objet Vecteur est constitué d'un entier spécifiant sa dimension est d'un pointeur sur double pour stocker ses éléments.

Les surcharges d'opérateurs usuelles ont été imlémentées (avec notamment le produit scalaire entre deux vecteurs "—")

3 L'objet Matrice

L'objet matrice est composé d'une classe mère Matrice et de deux classes filles : Matrice-pleine et Matrice-creuse.

3.1 Matrice pleine

3.1.1 Opérations

La classe Matrice_pleine contient les opérations :
- multiplication par un vecteur, un scalaire, une matrice
- addition de matrices
- transposée
- inversion par Gauss Jordan
- résolution d'une équation matricielle par méthode itérative de descente de gradient conjugué

Inversion par Gauss Jordan

Pour inverser la matrice-pleine, nous avons utilisé un algorithme de type Gauss Jordan de complexité O(n^3) (la recherche du pivot représente au maximum n(n+1)/2 opérations, puis pour chaque pivot on effectue les operations sur les ligne donc on multiplie par n). *Comme la matrice est symetrique définie positive, nous avons pensé à une décomposition M=LU et une décomposition de Cholesky* $M = LL^T$*, mais nous n'avons pas réussi à l'implémenter.*

Résolution itérative par méthode de gradient conjugué

Au lieu de chercher une inversion directe pour une matrice_creuse, nous avons pensé à une méthode itérative pour obtenir une approximation précise du vecteur x dans l'équation Ax=b. Nous avons choisi la méthode du gradient conjugué car notre matrice solution A est bien symetrique définie positive.

Initialisation:

$$r_0 = b - Ax_0 \ (x_0 \ est \ le \ vecteur \ nul)$$

$$p_0 = r_0$$

$$k = 0$$

Tant que la norme de $r_{k+1} > \epsilon$ qu'on a défini au préalable et $k < N$ le nombre maximal d'itération:

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$$

$$x_{k+1} = x_k + \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k A p_k$$

$$\beta_k = \frac{r_{k+1}^T r_k}{r_k^T r_k}$$

$$p_{k+1} = r_{k+1} + \beta_k p_k$$

$$k+ = 1$$

Résultat:

$$\text{retourner } x_{k+1}$$

La convergence se fait en au plus n opérations.En restreignant le nombre d'iteration à $k << n$ (la dimension du probleme), la complexité est au plus $O(kn^2)$ (un produit matrice*vecteur est de complexité $O(n^2)$ et on fait au plus k itérations) .

Cette méthode permet d'avoir une bonne approximation de la solution en moins de calculs. Pour un maillage grossier et avec des matrices pleines, le calcul de la solution pouvait prendre jusqu'a 20 minutes avec Gauss-Jordan . Avec la méthode de gradient conjugé, la solution est calculée quasi instantanément.

L'implémentation du calcul de la solution avec des matrice_creuses n'est plus necessaire pour obtenir une solution avec un maillage fin, mais elle permettrait d'optimiser un peu plus le calcul.

3.2 Matrice creuse

3.2.1 Intérêt

L'interet de la Matrice-creuse est d'utiliser moins de mémoire pour stocker la matrice et d'effectuer moins de calcul lors du produit matriciel. En effet, La complexité du produit matricielle pour une Matrice_pleine est constante en $O(n^2)$ tandis qu'elle est **au maximum** en $O(n^2)$ pour une Matrice_creuse.

3.2.2 Architecture

La classe fille Matrice_creuse utilise une map pour stocker ses valeurs. La map permet d'associer une paire d'entier (les indices où se trouvent l'élément non nul) et la valeur non nulle correspondante. Pour construire une Matrice_creuse, on part d'une d'un objet map vide, puis on utilise la fonction **add** pour ajouter dans la map les indices des coefficients non nuls ainsi que leur valeur.

3.2.3 Opérations

Pour tirer partie de la structure de la Matrice_creuse nous avons implémenté les opérations permettant de transformer une matrie_pleine en matrice-creuse (**PtoC**) et la transformation inverse (**CtoP**). En effet une transformation de matrice_pleine en matrice_creuse a une complexité maximale en $O(n^2)$.

4 L'objet Maillage

4.1 Architecture

Le maillage est constitué des éléments :

- Nbpt : nombre de noeuds
- Nbtri : nombre de triangles
- Coorneu : Coordonnées des noeuds
- Refneu : référence des noeuds
- Numarete : références des points constituant les arêtes
- Refarete : référence des arêtes
- Numtri : références des points constituant les triangles
- Reftri : référence des triangles

Le maillage est crée en amont grâce au logiciel gmsh. L'implémentation de la méthode de descente de gradient permet de faire un choix de pas de maillage assez fin tout en limitant la complexité temporelle de l'algorithme. Le principal enjeu autour de cette classe est de permettre un lien simple et robuste entre l'interface gmsh et notre code en C++ : on doit être en mesure de changer des paramètres du maillage et de compiler rapidement le code avec ces nouvelles données.

On récupère le maillage obtenu avec un constructeur par défaut qui lit le fichier texte (*maillageSIM202.txt*) contenant les informations relatives au maillage. Une fonction permettant de séparer les chaînes de caractère au niveau d'un séparateur a été implémentée pour pouvoir récupérer les éléments du maillage *(split-str)*.

Une fonction membre de cette classe permet en outre de récupérer les points situés sur le bord du domaine (pour la pseudo-élimination). On utilise pour cela le fait que les couples de points du bord du domaine forment des arêtes qui n'appartiennent qu'à un seul triangle à la fois contrairement aux autres arêtes du maillage.

5 L'objet Problème

5.1 Eléments algorithmiques

Par des considérations de stabilité et de précision des résultats, le pas de discrétisation spatiale h sera pris égal à $0,05$ et le pas de discrétisation temporelle que l'on fera varier.

5.1.1 Discrétisation temporelle

Nous allons discrétiser la dérivée temporelle en utilisant le schéma ci-dessous :

$$\frac{\partial U}{\partial t} = \frac{U^{n+1} - U^n}{\Delta t}$$

On s'intéessera donc maintenant au système suivant :

$$\begin{cases} \frac{U^{n+1}-U^n}{\Delta t} - c_u \Delta U^{n+1} = S_u^n \\ \frac{V^{n+1}-V^n}{\Delta t} - c_v \Delta V^{n+1} = S_v^n \end{cases}$$

On va maintenant chercher à trouver une expression explicite de ΔU^{n+1} et ΔV^{n+1} de manière à pouvoir exprimer les termes au temps $n+1$ en fonction des termes au temps n.

5.1.2 Discrétisation spatiale

Pour discrétiser les derivées d'ordre 2 isssues du calcul du laplacien, nous allons utiliser la méthode des éléments finis de Lagrange d'ordre 1.

Cela consiste à créer, à partir du maillage générée, une nouvelle base discrète de l'espace. Ainsi, on prendra comme base $(\omega_1, \omega_2, ..., \omega_N)$ avec N le nombre de sommets du maillage, et $\omega_i(M_j) = \delta_{i,j}$.

On recherche donc la solution sous la forme d'un couple de vecteur $(U^n, V^n)_{n\in\mathbb{N}}$ chacun de dimension le nombre de sommets et dont la $i^{ème}$ composante correspond à la population présente sur le sommet i à l'instant n.

En mettant le système sous la forme d'une formulation variationnelle, cette méthode nous permet d'exprimer les termes au temps $n+1$ en fonction des termes au temps n :

$$\begin{cases} \mathbb{A}_u U^{n+1} = S_u^n + \frac{1}{\Delta t}\mathbb{M}U^n \\ \mathbb{A}_v V^{n+1} = S_v^n + \frac{1}{\Delta t}\mathbb{M}V^n \end{cases}$$

Avec $\mathbb{A}_i = \frac{1}{\Delta t}\mathbb{M} + c_i\mathbb{K}, \quad i = u, v$ et $\begin{cases} \mathbb{M}_{i,j} = \int_\Omega \omega_i\omega_j d\Omega \\ \mathbb{K}_{i,j} = \int_\Omega \nabla\omega_i\nabla\omega_j d\Omega \end{cases}$

On remarque donc qu'il suffit de calculer, puis d'inverser les matrices $\mathbb{A}_u$ et $\mathbb{A}_v$ pour trouver une expression explicite de U^{n+1} et V^{n+1} en fonction de U^n et V^n.

5.2 Architecture

5.2.1 Assemblage des matrices solutions

Matrices élémentaires

Pour calculer plus facilement, les matrice de raideur et de masse (respectivement $\mathbb{K}$ et $\mathbb{M}$), on commence par calculer les matrices élémentaires de raideur et de masse (respectivement $\mathbb{K}^0$ et $\mathbb{M}^0$), restreintes au 1^{er} triangle du maillage.

Le triangle élémentaire $\hat{T}$ est composé des sommets suivants :

$$\hat{M}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \ \hat{M}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \ \hat{M}_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Et la base élémentaire $(\hat{\omega}_1, \hat{\omega}_2, \hat{\omega}_3)$ est défini de la manière suivante :

$$\hat{\omega}_1 = 1 - x - y$$

$$\hat{\omega}_2 = x$$

$$\hat{\omega}_3 = y$$

Or, on sait que les sommets du triangle T_l sont liés au sommets du triangle élémentaire par la relation suivante :

$$M_i = F_l(\hat{M}_i) = B_l\hat{M}_i + S_l, \quad i = 1, 2, 3$$

$$S_l = M_1 \ \ et \ \ B_l = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix}$$

En effectuant un changement de variable $\hat{\omega}_i = F_l^{-1}(\omega_i)$, on obtient pour la matrice de masse :

$$\int_{T_l} \omega_i(M)\omega_j(M)d\Omega = \int_{T_{elem}} \hat{\omega}_i(\hat{M})\hat{\omega}_j(\hat{M})|det(B_l)|d\hat{\Omega}$$

Et pour la matrice de rigidité :

$$\int_{T_l} \nabla\omega_i(M)\nabla\omega_j(M)d\Omega = \int_{T_{elem}} [(B_l^T)^{-1}\nabla\hat{\omega}_i(\hat{M})]^T.[(B_l^T)^{-1}\nabla\hat{\omega}_j(\hat{M})]|det(B_l)|d\hat{\Omega}$$

On trouve finalement :

$$\begin{cases} \mathbb{M}^0_{i,j} = \frac{1}{24}|det(B_l)| & i \neq j \\ \mathbb{M}^0_{i,i} = \frac{1}{12}|det(B_l)| \end{cases}$$

$$\begin{cases} \mathbb{K}^0_{i,j} = \frac{1}{2}[(B_l^T)^{-1}\nabla\hat{\omega}_i(\hat{M})]^T.[(B_l^T)^{-1}\nabla\hat{\omega}_j(\hat{M})]|det(B_l)| \\ Avec \;\; \nabla\hat{\omega}_1 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \; \nabla\hat{\omega}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \; \nabla\hat{\omega}_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{cases}$$

Matrices de raideur et de masse

Il nous suffit maintenant de construire $\mathbb{M}$ à partir $\mathbb{M}^0$ et $\mathbb{K}$ à partir de $\mathbb{K}^0$ et des coordonnées des sommets des triangles.

Exemple : Si on considère le triangle T_l composé des sommets référencés i, j, k. La matrice de masse élémentaire de masse correspondante s'écrit :

$$\mathbb{M}^0 = \begin{pmatrix} \int_{T_l} \omega_i\omega_i d\Omega & \int_{T_l} \omega_i\omega_j d\Omega & \int_{T_l} \omega_i\omega_k d\Omega \\ \int_{T_l} \omega_j\omega_i d\Omega & \int_{T_l} \omega_j\omega_j d\Omega & \int_{T_l} \omega_j\omega_k d\Omega \\ \int_{T_l} \omega_k\omega_i d\Omega & \int_{T_l} \omega_k\omega_j d\Omega & \int_{T_l} \omega_k\omega_k d\Omega \end{pmatrix}$$

On aura alors, par exemple :

$$\mathbb{M}_{i,i} = n\mathbb{M}^0_{1,1}$$

$$\mathbb{M}_{i,j} = 2\mathbb{M}^0_{1,2}$$

$$\mathbb{M}_{j,k} = 2\mathbb{M}^0_{2,3}$$

Avec n le nombre de triangles auxquel appartient le sommet i. Le coefficient 2 correspond au fait que deux triangles ont au plus 1 arrête en commun.

5.2.2 Pseudo-élimination

Etant donné que l'on travaille avec les conditions limite de Dirichlet, on souhaite imposer une condition de nullité sur les noeuds du bord.
Pour cela on utilise les fonctions *elimine_M* et *elimine_V* (resp. Matrice et

Vecteur), qui prennent en argument la matrice solution $\mathbb{A}$ (resp. le Vecteur S^n associé), le nombre de sommets du maillage et un Vecteur contenant les références des noeuds du bords, puis procède à la pseudo-élimination, qui mets égaux à 0 les coefficients correspondants à un noeuds situé sur le bord du maillage.

5.2.3 Assemblage des seconds membres

L'assemblage des seconds membre se fait de manière évidente, en calculant à partir de U^n et V^n

$$\begin{cases} S_u^n = U^n(c\mathbf{1} - dV^n) \\ S_v^n = V^n(-a\mathbf{1} + bU^n) \end{cases}$$

ou

$$\begin{cases} S_u^n = U^n(\mathbf{1} - U^n - cV^n) \\ S_v^n = U^n(a\mathbf{1} - V^n - bU^n) \end{cases}$$

selon si on s'intéresse à la partie validation ou à la partie résolution du problème.

6 Main

6.1 Architecture

L'architecture de main.cpp se décompose en 4 parties :

- La récupération des données du maillage
- Le choix de l'exécutable et son calcul
- La résolution du problème
- La récupération des données calculées et leur affichage

6.2 Récupération des données du maillage

Après avoir initialisé le maillage grâce au logiciel gmsh (on choisit un pas de maillage h =0.05 pour toutes les résolutions) on récupère les éléments nécessaires aux calculs de la solution en faisant appel au constructeur de la classe Maillage.

On crée par la suite le problème (initialisation des coefficients des différents problèmes) et on récupère les points du bords du domaine grâce à la fonction dédiée de la classe Maillage en prévision de la phase de pseudo-élimination.

6.3 Validation du code et Résolution du problème

6.3.1 Validation du code

Equation de Poisson

Nous avons commencé par tester la résolution de l'équation de Poisson avec condition aux limites de type Dirichlet : -$\Delta u = f$ avec $u(x, y) = sin(\pi x)sin(\pi y)$ dont on peut calculer facilement le Laplacien $f(x, y) = \pi^2 sin(\pi x)sin(\pi y)$ et ainsi vérifier la justesse de notre simulation du point de vue spatial. Nous avons rencontré un problème de scaling quadratique en $\frac{1}{h}$, finalement corrigé en multipliant nos fonctions de base par $\frac{1}{h}$ (h étant la hauteur du triangle issue du noeud associé au degré de liberté).

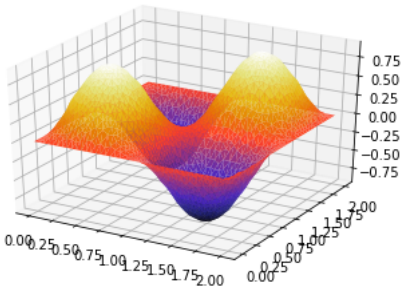

Figure 1: Simulation du problème de Poisson

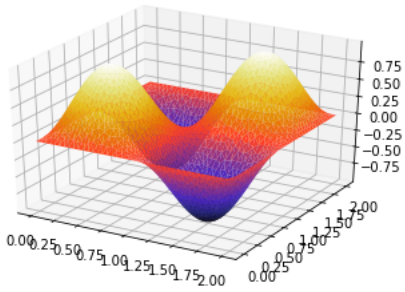

Figure 2: Solution réelle du problème de Poisson

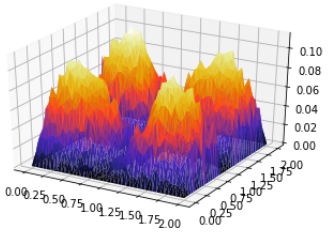

Figure 3: Ecart absolu entre la solution réelle et simulée du problème de Poisson

On constate une erreur maximale de $0,1$, soit une erreur relative d'environ 10%. Nous ne savons pas interpréter cette erreur : l'augmentation de la finesse du maillage ne diminue pas davantage l'erreur (contrairement à ce qu'affirme le théorème d'Aubin-Nitsche). Nous sommes cependant assuré de la cohérence du code concernant le maillage.

Modèle sans diffusion

Pour valider le comportement temporel de notre solution, nous allons étudier un modèle sans diffusion, qui correspond aux équations "classiques" de Lotka-Volterra qui modélisent une intéraction proie/prédateur dans le temps :

$$\begin{cases} S_u = u(c - dv) \\ S_v = v(-a + bu) \\ c_u = 0 \\ c_v = 0 \\ \Omega = \mathbb{R} \end{cases}$$

On va donc chercher à résoudre :

$$\begin{cases} \frac{\partial u}{\partial t} = u(c - dv) \\ \frac{\partial v}{\partial t} = v(-a + bu) \\ \Omega = \mathbb{R} \end{cases}$$

On cherche notamment à mettre en évidence l'inluence des paramètres a, b, c et d sur l'évolution des populations. On rappelle la signification des coefficients et des fonctions du problème :

- u : population de proies
- v : population de prédateurs

- a : taux de mortalité des prédateurs
- b : taux de naissance des prédateurs lié à la capture des proies
- c : taux de naissance des proies
- d : taux de mortalité des proies lié à la prédation

Avant d'entrer dans l'interprétation de chaque figure, remarquons que l'on observe bien sur chacune d'entre elles les oscillations qui caractérisent le lien entre les deux espèces. En initialisant les populations dans des proportions identiques, la quantité de proies chute vivement tandis que celle de prédateurs augmente. Une fois que le vivier de nourriture se réduit, la quantité de prédateur décroît, ce qui permet aux proies de se reproduire sans menace. Les prédateurs pourront alors profiter de cette augmentation de vivier de nourriture pour se reproduire, et ainsi de suite...

Fig 4 (a=2,b=1,c=4,d=1) : configuration qui permet aux proies et aux prédateurs de cohabiter dans des proportions semblables (léger avantage pour les prédateurs).

Fig 5 (a=4,b=1,c=2,d=1) : La modification des coefficients donne désormais un clair avantage biologique aux proies, donc les pics de population sont presque deux fois plus importants que ceux de leurs prédateurs.

Fig 6 (a=4,b=2,c=2,d=1) : Les coefficients sont identiques à la situation précédente, à l'exception du taux de naissance des prédateurs lié à la capture des proies, ce qui leur permet de reprendre le dessus sur le plan démographique.

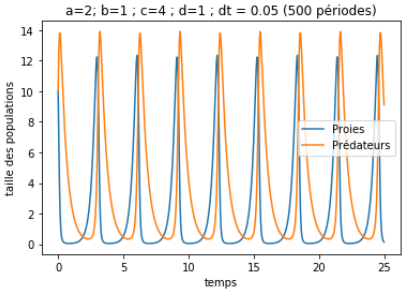

Figure 4: Simulation du problème temporel, sans diffusion

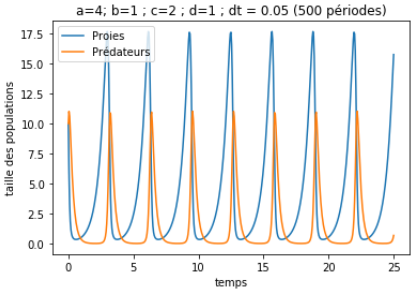

Figure 5: Simulation du problème temporel, sans diffusion

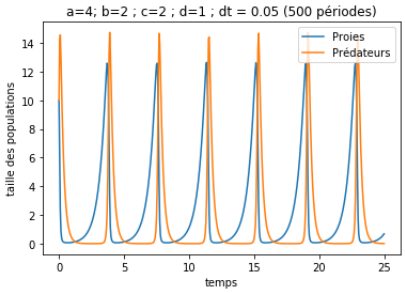

Figure 6: Simulation du problème temporel, sans diffusion

Equation de diffusion non couplée

La dernière étape de la validation consiste à simuler l'évolution d'une population seule. Il s'agit donc d'un modèle temporel et spatial contrairement aux deux tests précédents, mais sans concurrence. L'équation de diffusion seule avec les conditions de Dirichlet homogène est la suivante :

$$\begin{cases} \frac{\partial u}{\partial t} - c\Delta u = f(x) \quad sur\ \Omega = [0,2]^2 \\ u(t=0) = 1 \quad sur\ \Omega \\ u = 0 \quad sur\ \partial\Omega \end{cases}$$

Avec $f(x,y) = \pi^2 sin(\pi x) sin(\pi y)$ (nous reprenons la condition à l'intérieur du domaine que nous avons employé dans le premier test pour vérifier si on converge bien vers la même solution).

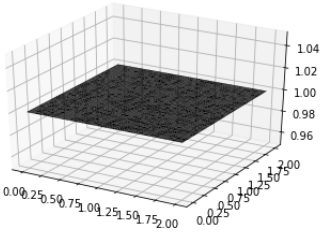

Figure 7: Visualisation de l'état initial de la population (T=0)

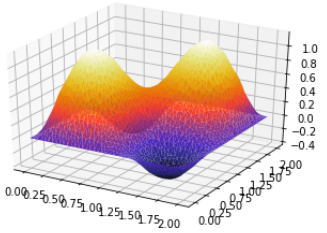

Figure 8: Visualitation de l'état courant de la population (T=0,3)

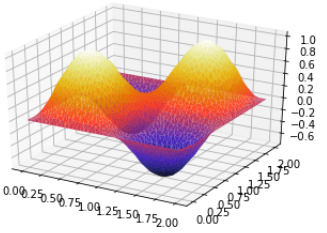

Figure 9: Visualitation de l'état courant de la population (T=0,6)

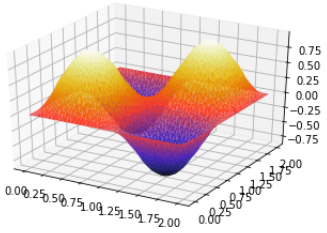

Figure 10: Visualitation de l'état final de la population (T=0,9)

On observe bien une convergence vers la solution du premier test pour un temps suffisament grand, ie convergence vers un état stationnaire.

6.3.2 Résolution du problème

Compte tenu des résultats observés précédemment, nous sommes désormais rassurés vis à vis de la cohérence des simulations à venir. Il ne nous reste qu'à ajouter le couplage au système précédent. La résolution se fait donc selon les étapes suivantes :

1) Initialisation de la répartition des population
2) Construction des matrices de raideur et de masse
3) Assemblage, pseudo élimination et inversion des matrices solutions
4) Mise à jour de l'état de chaque population (boucle temporelle)

Le choix des coefficients est capital : en effet, on doit veiller à respecter des conditions sur ces derniers pour respecter la notion de capacité biotique (la taille maximale de la population d'un organisme qu'un milieu donné peut supporter).

Le choix que nous faisons ici fixe une capacité biotique de 1 pour chaque espèce (coefficients : $a = 1, b = 1/3, c = 1/4, d = 1, c_u = 1, c_v = 1$).

On remarque ainsi qu'en partant de populations réparties de manière constante sur le territoire (les premiers résultats sont obtenus en initialisant la première espèce (u) à 0.9 et la seconde (v) à 0.1) ces dernières ne subissent pas de variation importantes du point de vue spatial (il n'y a pas de pic démographie en des endroits particuliers). Les deux groupes augmentent progressivement et de manière homogène dans l'espace pour tendre vers un équilibre situé à la limite de la capacité biotique imposée.

On propose dans un second temps une initialisation plus originale avec une population de départ homogène sur tout le territoire pour la seconde espèce (v), mais une répartition "sinusoïdale" pour la première (u) (on initialise avec la fonction suivante : $g(x, y) = 0.4sin(\pi x)sin(\pi y)+0.5$) afin d'observer si une condition initiale à gradient non nul entraîne des grandes variations démographiques sur le territoire dans la populations concurrente.

Après une phase d'homogénéisation de la population de la première espèce on observe le même schéma que précédemment : une augmentation progressive et homogène des deux groupes pour tendre vers la capacité biotique.

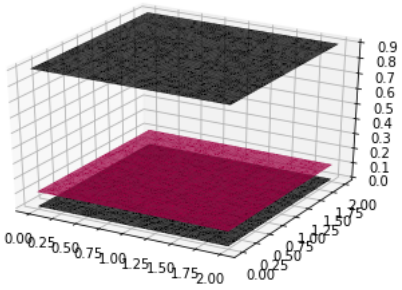

Figure 11: Visualisation de l'état initial des populations (T=0)

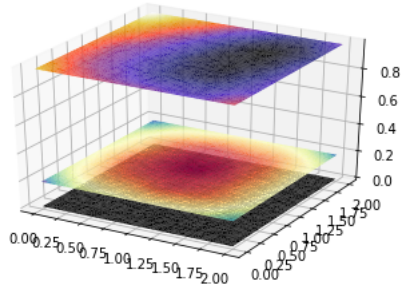

Figure 12: Visualitation de l'état courant des populations (T=1)

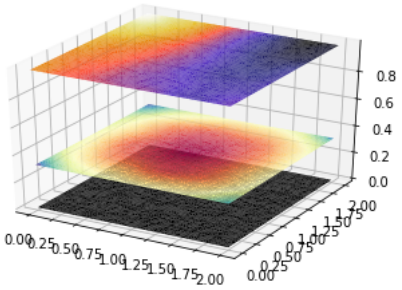

Figure 13: Visualitation de l'état courant des populations (T=2)

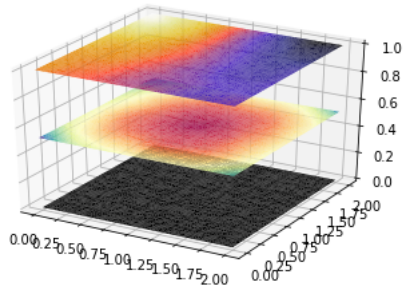

Figure 14: Visualitation de l'état courant des populations (T=3)

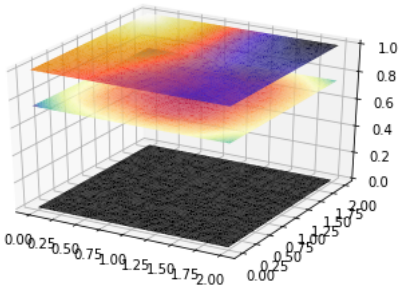

Figure 15: Visualitation de l'état courant des populations (T=4)

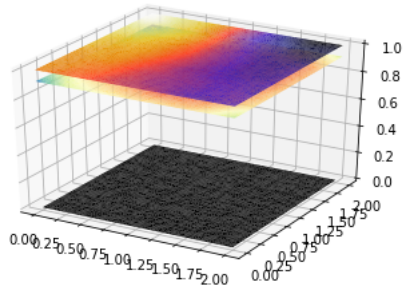

Figure 16: Visualitation de l'état final des populations (T=5)

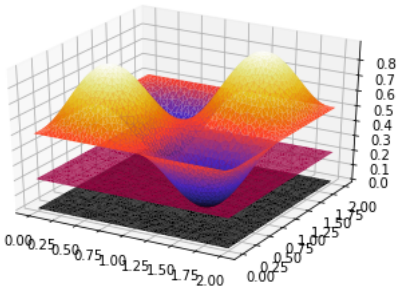

Figure 17: Visualisation de l'état initial des populations (T=0)

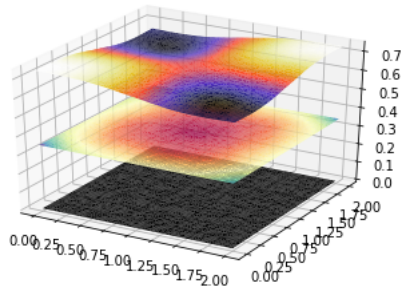

Figure 18: Visualitation de l'état courant des populations (T=1)

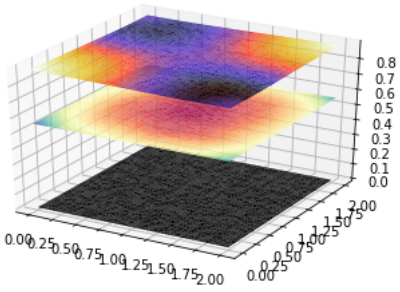

Figure 19: Visualitation de l'état courant des populations (T=2)

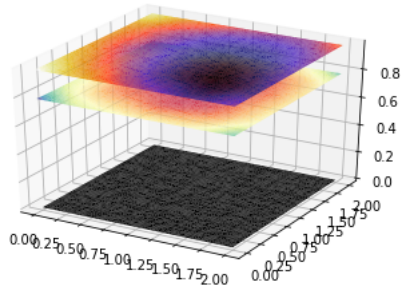

Figure 20: Visualitation de l'état courant des populations (T=3)

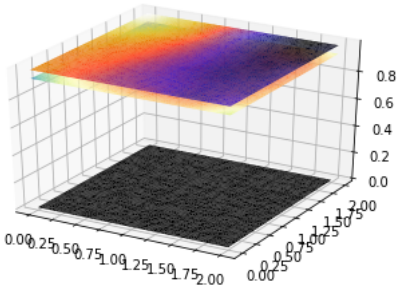

Figure 21: Visualitation de l'état courant des populations (T=4)

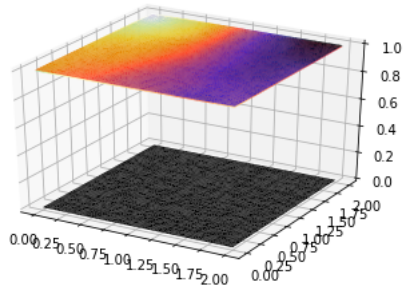

Figure 22: Visualitation de l'état final des populations (T=5)

6.4 Affichage

Les affichages précédemment observés on été réalisé en python via les modules *matplotlib.pyplot* pour le test en temporel uniquement et *mpltoolkits.mplot3d* pour tous les problèmes incluant des maillages.

La récupération des données se fait en deux temps :

- récupération des noeuds du maillage en début de main
- récupération du (ou des) vecteurs contenant les valeurs des solutions aux noeuds du maillage pendant et à la fin la résolution du problème

Le problème qui se pose est alors de gérer les formats de sortie de code pour qu'ils soient adaptés aux fonctions d'affichage évoluées des librairies disponibles en python. De manière générale, l'extraction des données se fait par écriture directe dans un fichier texte (affichage.txt) en utilisant des retour à la ligne pour séparer les objets (coordonnées des noeuds du maillage en x, en y, et valeurs de la solution aux noeuds du maillage) et des espaces pour séparer les valeurs numériques contenues dans ces objets.

Une fonction dédiée à la lecture de fichier texte (*numpy.loadtxt* de la librairie *numpy*) permet d'extraire nos résultats et des les convertir en tableaux *numpy* qui sont gérés par la fonction d'affichage plot_trisurf de *matplotlib.pyplot*. On peut alors afficher et superposer nos résultats pour comparer graphiquement l'évolution des populations en compétition.

7 Conclusion

L'étude de ces problèmes fut pour nous l'occasion d'appronfondir les notions de C++ abordées en SIM201. Nous avons du associer les compétences acquises en SIM201, ANN201 et OPT201 pour proposer une solution au problème final. Nous devions en outre faire face au problème de la complexité temporelle de l'algorithme du fait de la finesse du maillage. L'emploi de la méthode de descente de gradient a été très importante pour faire tourner l'algorithme rapidement et trouver les bugs dans le code (l'algorithme se termine en deux minutes pour le problème final, qui est le plus coûteux en temps avec un pas de maillage h=0.05 et T_max=5).

Pour améliorer davantage notre code, nous pourrions chercher à utiliser des matrices creuses (la classe a été crée, mais pas utilisée dans le code car nous n'avions pas de moyen de les inverser initialement) et chercher à tester davantage de valeurs de coefficients dans nos problèmes pour bien mettre en évidence leur influence sur le comportement des populations.