Université de Strasbourg

UFR de Math et Info

Classification Cahn-Hilliard.

KERMEZLI Ikram

Master 1 Calcul scientifique et mathématique de l'information

supervisé par :

M. J. Aghili

M. C. Prudhomm

Introduction

Le modèle de Cahn-Hilliard est un modèle d'équation aux dérivées partielles parabolique du quatrième ordre, introduit en 1958 par Cahn et Hilliard.

L'utilisation du modèle de Cahn-Hilliard est étendue à divers domaines de la chimie, de la physique, de la biologie et au secteur de l'ingénierie.

L'objectif de ce stage est de résoudre l'équation de Cahn-Hilliard en cherchant le meilleur contrôle pour tenter d'obtenir un état final souhaité.

Le modèle de Cahn-Hilliard

Le modèle de Cahn-Hilliard est défini comme suit:

$$\begin{cases} \partial_t u & = & \Delta(\phi'(u) + \gamma \Delta u) \\ u(t=0,.) & = & u_0(x) \end{cases} \quad (P)$$

avec u et u_0 deux fonctions de $\Omega \subset \mathbb{R}^2 \times [0, T]$ dans $\mathbb{R}$, $\gamma > 0$.

On effectue un changement de variable:

$$\begin{cases} \partial_t u & = & \Delta w & & \\ w & = & \phi'(u) + \gamma \Delta u & & (P') \\ u(0,.) & = & u_0(x) & \forall x \in \Omega & \end{cases}$$

Schéma implicite d'Euler

On effectue une discrétisation implicite par différence finis de (P'):

$$\begin{cases} \Omega & = & [0,1] \times [0,1] \\ \text{le nombre de points} & = & N \\ \Delta x = \Delta y & = & \frac{1}{N-1} \\ \Delta t & = & cfl \times \Delta x^2 \end{cases}$$

On pose:

$$\begin{aligned} U_{i,j}^n &\approx u(t_n, x_{i,j}) \\ W_{i,j}^n &\approx w(t_n, x_{i,j}) \end{aligned}$$

Où :

$$\begin{aligned} x_{i,j} &= (i\Delta x, j\Delta y) \\ t_n &= n\Delta t \end{aligned}$$

Schéma implicite d'Euler

On obtient après calcul :

$$BX^{n+1} + G(X^{n+1}) - F^n = 0$$

Où :

$$B = \begin{pmatrix} Id & \frac{\Delta t}{\Delta x^2} A \\ \frac{-\gamma}{\Delta x^2} A & Id \end{pmatrix} \in \mathbb{R}^{\mathbb{N},\mathbb{N}}$$

$$F = \begin{pmatrix} U_{1,1}^n \\ \vdots \\ U_{N,N}^n \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad X^{n+1} = \begin{pmatrix} U_{1,1}^{n+1} \\ \vdots \\ U_{N,N}^{n+1} \\ W_{1,1}^{n+1} \\ \vdots \\ W_{N,N}^{n+1} \end{pmatrix} \quad G(X^{n+1}) = \begin{pmatrix} 0 \\ \phi'(U^{n+1}) \end{pmatrix}.$$

Schéma de Crank-Nicolson

La θ-méthode consiste à approcher la dérivée spatiale au temps $n+1$ à l'aide de termes évalués à n (partie explicite) et de termes évalués à $n+1$ (partie implicite):

$$\begin{cases} \frac{U^{n+1}-U^n}{\Delta t} &= \theta\mathcal{L}(W^{n+1}) + (1-\theta)\mathcal{L}(W^n) \\ W^{n+1} &= \phi'(U^{n+1}) + \gamma\mathcal{L}(U^{n+1}) \end{cases}$$

Où $\mathcal{L}$ représente la discrétisation du laplacien

Ce schéma est d'ordre 2 en temps($O(\Delta t^2)$)

Outils utilisés dans l'implémentation

- Le programme est écrit en Python.
- numpy, matplotlib, torch.
- Le logiciel utilisé est Visual Studio Code.
- Une partie de l'exécution du programme s'est faite sur le cluster Atlas

Algorithme de Cahn-Hilliard

La fonction J est définie d'après le system d'équations (P'') par :

$$J(X) = BX^{n+1} + G(X^{n+1}) - F^n$$

Pour simplifier l'implémentation, on calcul la fonction J en deux partie: $res(U)$ et la deuxième en $res(W)$ qu'on concatène pour retourner un seul vecteur $J(X)$.

Pour obtenir le gradient de la fonction, on utilise pour cela la bibliothèque autograd.numpy, la fonction jacobian() retourne le gradient, on fait appel à cette fonction dans $dJ(X)$ qu'on utilise directement dans l'algorithme de Newton.

Algorithme de Cahn-Hilliard

1) On effectue l'appel à de l'algorithme de Newton,
2) On test la convergence de ce dernier:
   - Si l'algorithme converge on augmente le pas de temps Δt d'un certain pourcentage fixé :

     $$\Delta t = \alpha \times \Delta t$$

     et on met à jour les compteurs de temps t et du nombre d'itérations.
   - Si l'algorithme ne converge pas, on met à jour le pas de temps seulement :

     $$\Delta t = \beta \times \Delta t$$

     et on revient à l'étape 1).

Passage au Pytorch

Pour pouvoir résoudre l'EDP en introduisant le contrôle on a besoin de trouver le contrôle optimal V qui minimise la quantité :

$$\| u(T, x, V) - \hat{u} \|^2$$

et pour se faire, on propose d'utiliser la bibliothèque Pytorch qui permet de créer un graphe et donc de faire la différenciation de l'optimisation de graphe.

Algorithme de descente du gradient

On passe à la dernière étape: l'algorithme de la descente du gradient qui résout l'EDP en trouvant le contrôle V qui permet d'obtenir un résultat attendu.

En introduisant le contrôle, l'EDP s'écrit comme suit:

$$\begin{cases} \partial_t u & = & \Delta w - v \\ w & = & \phi'(u) + \gamma \Delta u \\ u(0,.) & = & u_0(x) \qquad \forall x \in \Omega \end{cases}$$

Algorithme de descente du gradient

On propose de définir le contrôle V par 3 constantes, c'est a dire qu'on cherche le contrôle dans un espace tel que:

$$H = \left\{ V = v_0 f_0(x) + v_1 f_1(x) + v_2 f(t) / (v_0, v_1, v_2) \in \mathbb{R}^3 \right\}$$

Où f_0 et f_1 sont des fonctions constantes par morceaux, de $\mathbb{R}^2$ dans $\mathbb{R}$; on les appelle aussi base de Haar et f une fonction qui dépend du temps.

Algorithme de descente du gradient

L'algorithme de la descente du gradient est défini comme ceci:

1) On fixe un V_0 initial et un nombre d'itérations maximal,
2) On défini l'optimiser:

   optimizer = torch.optim.Adam([V], lr=lr)

   où lr étant une constante fixé au début.
3) A chaque itération on résout l'équation de Cahn-Hilliard avec l'algorithme de CH, on calcul la fonction a minimiser:

   loss = torch.norm(U-y)**2

   où U est la solution de l'algorithme de CH et y est la solution cible qu'on veut approcher avec le V.Puis:

   loss.backward() et optimizer.step()

   La variable V est actualisée, ceci jusqu'à un certain nombre d'itérations.

Rappel du plan

Algorithme avec le pas de temps adaptatif

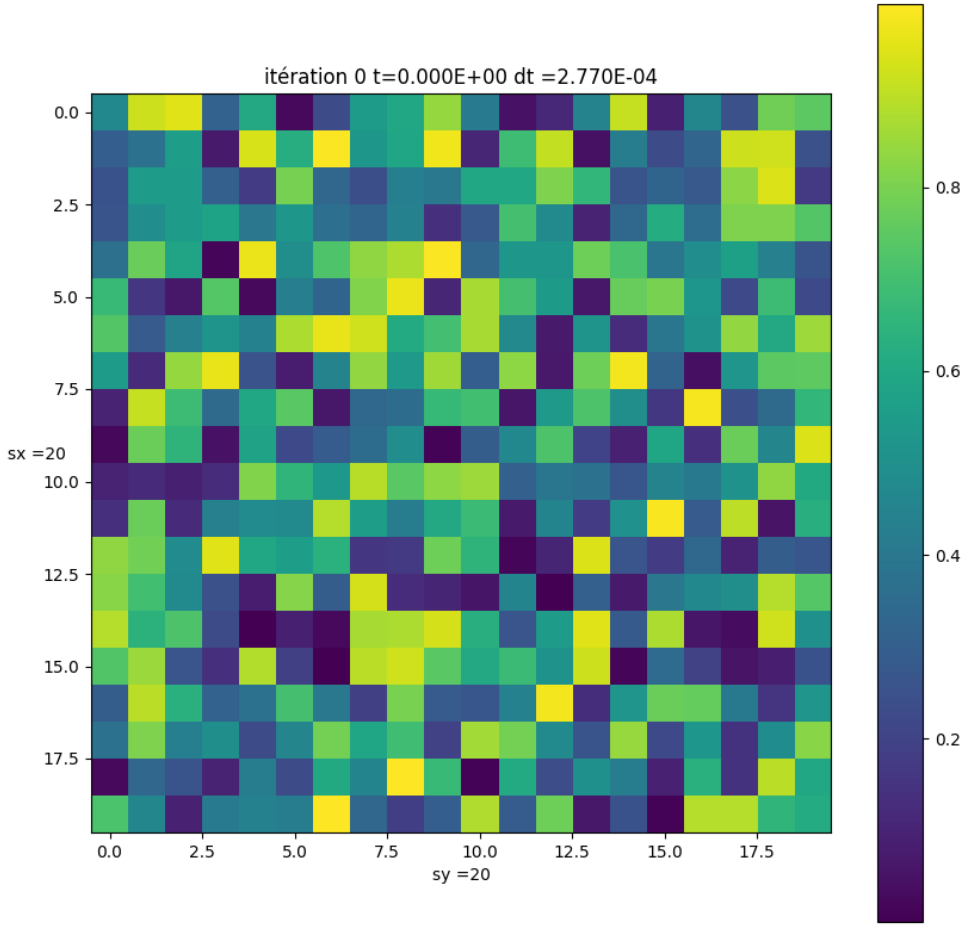
Pour illustrer la différence entre les deux version de l'algorithme de résolution de l'équation CH, on effectue deux résolution et on compare les résultats obtenus ci-dessous:

Premièrement, l'algorithme sans pas adaptatif, c'est à dire que le pas de temps Δt est constant pendant toute l'exécution de l'algorithme, on prend comme paramètres :

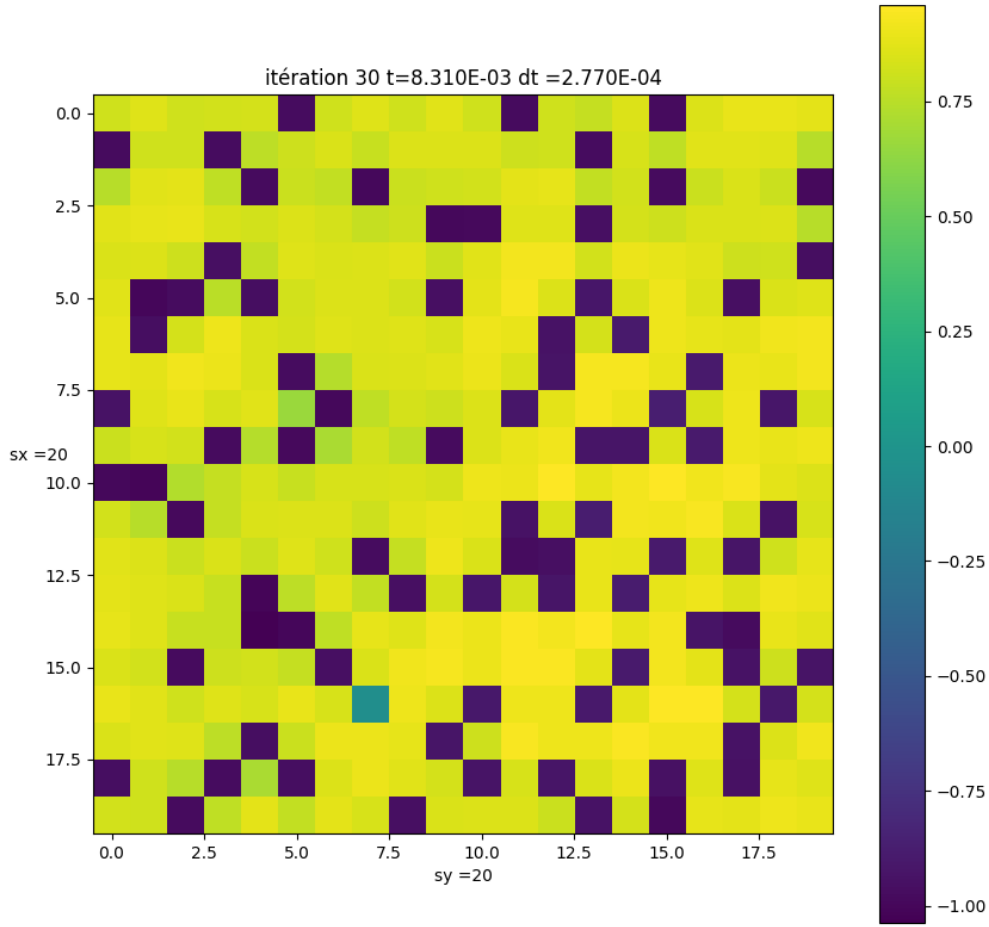
$$\begin{cases} \text{nombre de points} &= 20 \\ \Delta x = \Delta y &= 5.3\text{e-}2 \\ \gamma &= 1\text{e-}4 \\ \Delta t_0 &= 2.77\text{e-}4 \\ \text{T final} &= 0.1 \end{cases}$$

On obtient:

Algorithme avec le pas de temps adaptatif



Algorithme avec le pas de temps adaptatif



Algorithme avec le pas de temps adaptatif

résultats :

$$\begin{cases} \text{temps de calcul} &= 3 \text{ minutes et } 10,17 \text{ secondes} \\ \text{temps final t} &= 8.31\text{e-}3 \\ \text{nombre d'itérations} &= 30 \\ \text{minimum de la solution} &= -1.037 \\ \text{maximum de la solution} &= 9.592\text{e-}1 \end{cases}$$

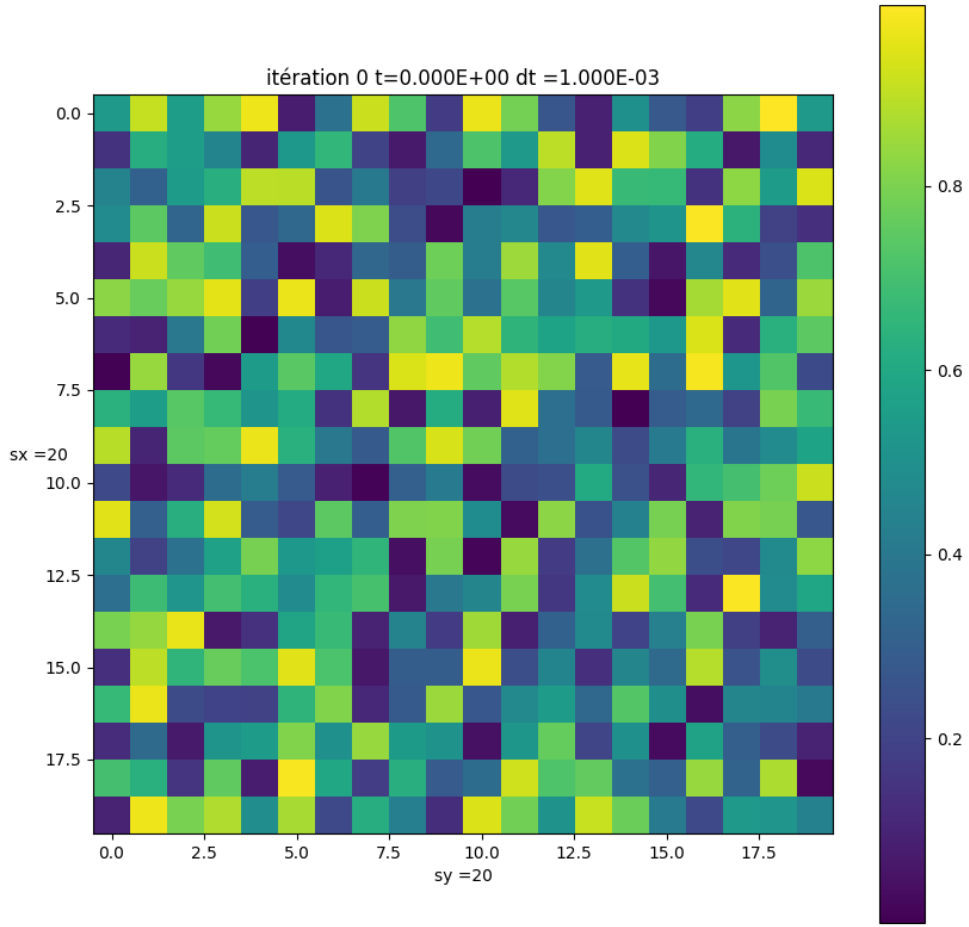
Algorithme avec le pas de temps adaptatif

Puis, l'algorithme avec pas adaptatif où le pas de temps Δt augmente ou diminue à chaque itération,selon la convergence ou non de l'algorithme de Newton, on prend les paramètres :

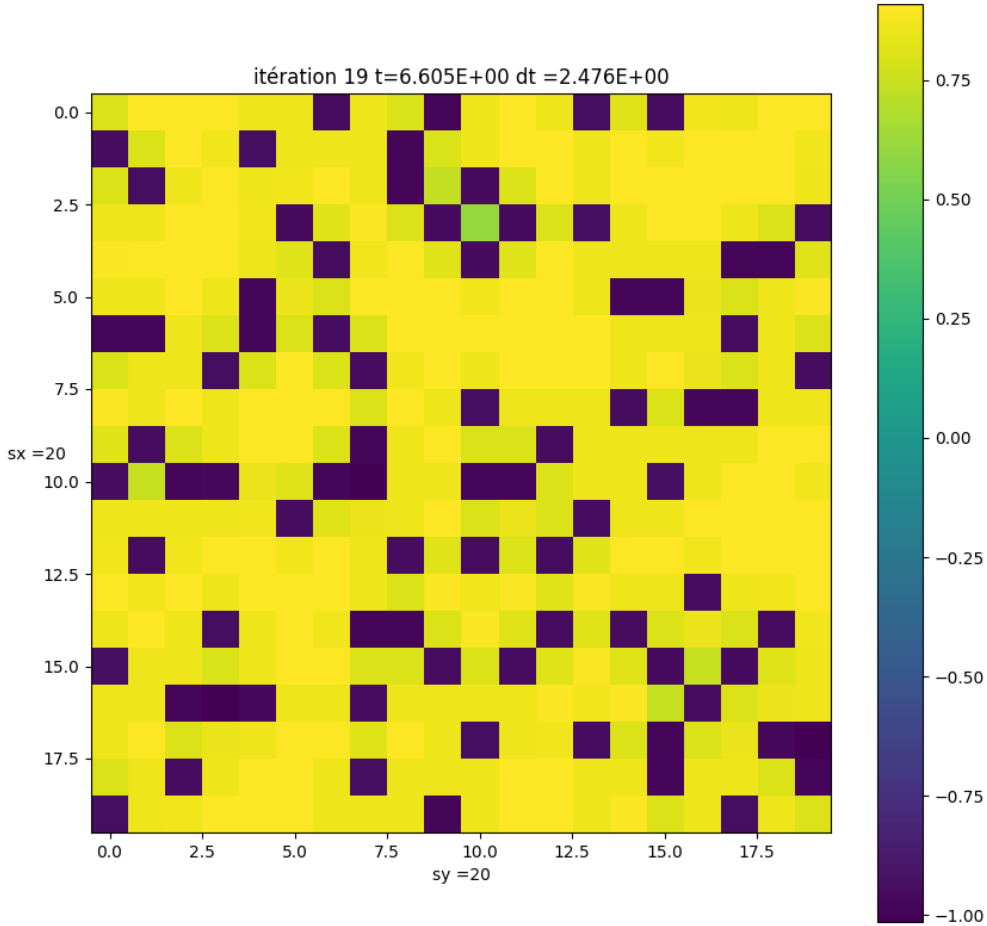
$$\begin{cases} \text{nombre de points} &= 20 \\ \Delta x = \Delta y &= 5.3\text{e-}2 \\ \gamma &= 1\text{e-}4 \\ \Delta t_0 &= 1\text{e-}3 \\ \text{T final} &= 5 \end{cases}$$

On obtient ceci :

Algorithme avec le pas de temps adaptatif



Algorithme avec le pas de temps adaptatif



Algorithme avec le pas de temps adaptatif

résultats :

$$\begin{cases} \text{temps de calcul} &= 3 \text{ minutes et } 11,67 \text{ secondes} \\ \text{temps final t} &= 6.61 \\ \text{nombre d'itérations} &= 19 \\ \text{minimum de la solution} &= -1.014 \\ \text{maximum de la solution} &= 9.089\text{e-}1 \end{cases}$$

Paramètre de pénalisation γ

C'est un paramètre qui pénalise la condition de $\Delta u = 0$ dans l'équation:

$$\partial_t u = \Delta(\phi'(u) + \gamma \Delta u)$$

L'algorithme qu'on met en place vas devoir minimiser la quantité :

$$\int_\Omega \phi'(u) + \gamma |\nabla u|^2 \, dx$$

Quand γ est assez grand (par exemple 1 ou 10^{-1}), l'algorithme va chercher des solutions $u(x)$ tel que :

$$\begin{cases} \phi(u) & \to 0 \\ |\nabla u| & \to 0 \end{cases}$$

Paramètre de pénalisation γ

On prend deux exemples pour illustrer le rôle du paramètre, on utilise le potentiel de Cahn-Hilliard, avec en premier un γ assez petit:

$$\begin{cases} \text{nombre de points} &= 10 \\ \Delta x = \Delta y &= \text{1.11e-1} \\ \gamma &= \text{1e-4} \\ \Delta t_0 &= \text{1e-3} \\ \alpha &= 1.6 \\ \beta &= 0.8 \\ \text{T final} &= 0.1 \end{cases}$$

On obtient:

Paramètre de pénalisation γ

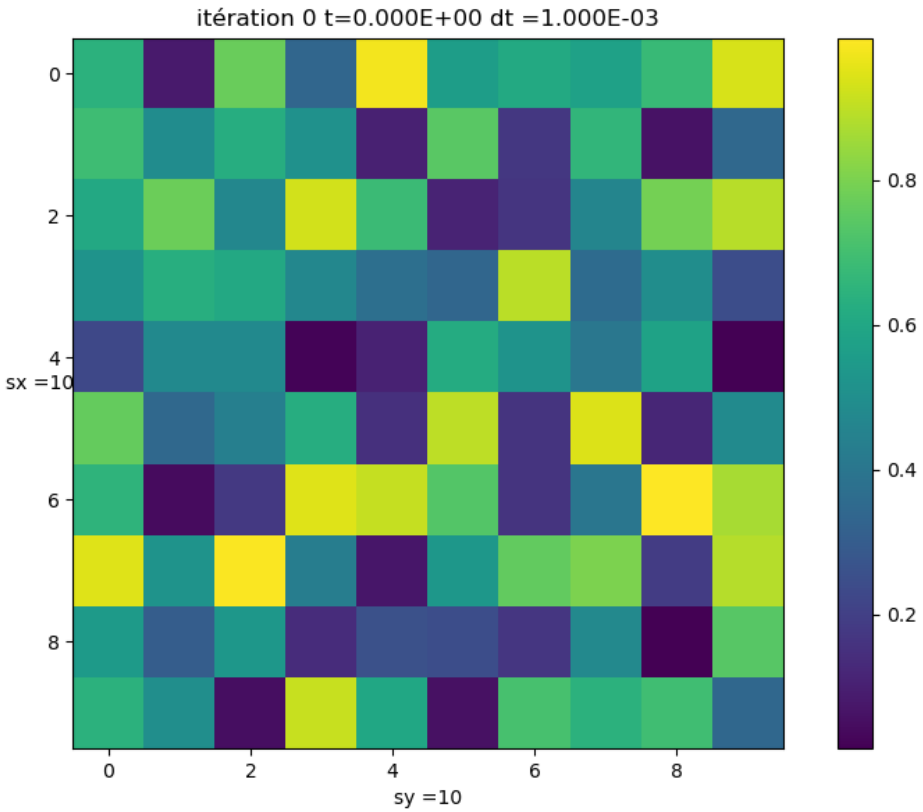


Figure: simulation avec $\gamma = 1e\text{-}4$

Paramètre de pénalisation γ

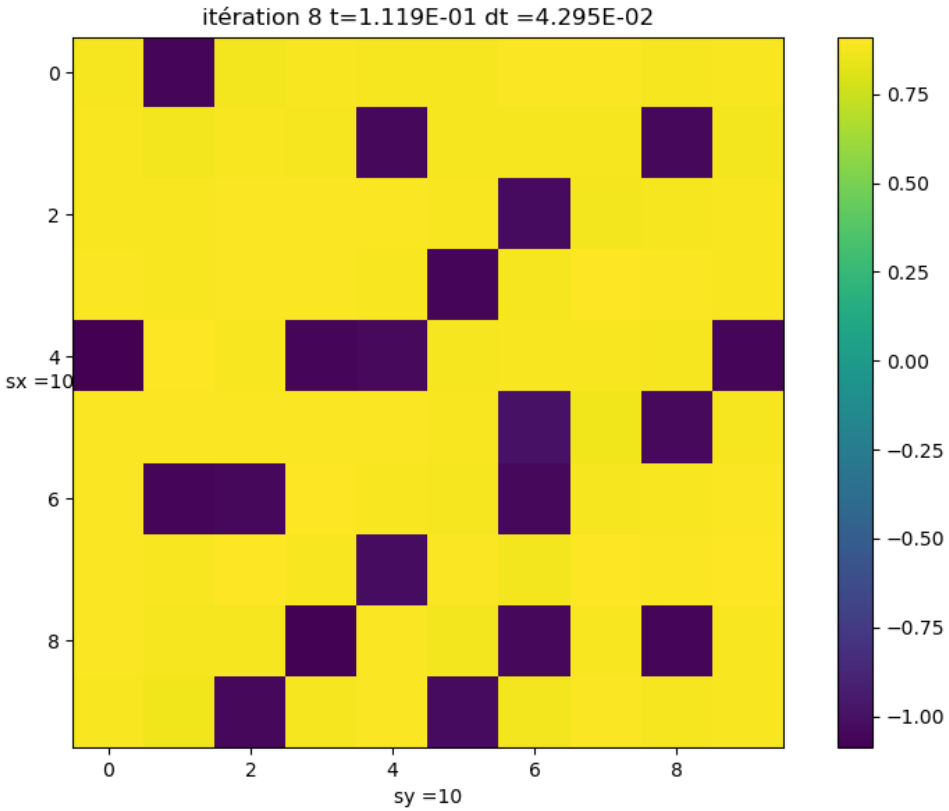


Figure: simulation avec $\gamma = 1e\text{-}4$

Paramètre de pénalisation γ

résultats :

$$\begin{cases} \text{temps de calcul} & = & 7.23 \text{ secondes} \\ \text{temps final t} & = & 1.12\text{e-}1 \\ \text{nombre d'itérations} & = & 7 \\ \text{minimum de la solution} & = & -1.08 \\ \text{maximum de la solution} & = & 9.113\text{e-}1 \end{cases}$$

Paramètre de pénalisation γ

On prend un deuxième exemple avec un γ plus grand:

$$\begin{cases} \text{nombre de points} &= 10 \\ \Delta x = \Delta y &= 1.11\text{e-}1 \\ \gamma &= 1\text{e-}2 \\ \Delta t_0 &= 1\text{e-}3 \\ \alpha &= 1.6 \\ \beta &= 0.8 \\ \text{T final} &= 2 \end{cases}$$

On obtient:

Paramètre de pénalisation γ

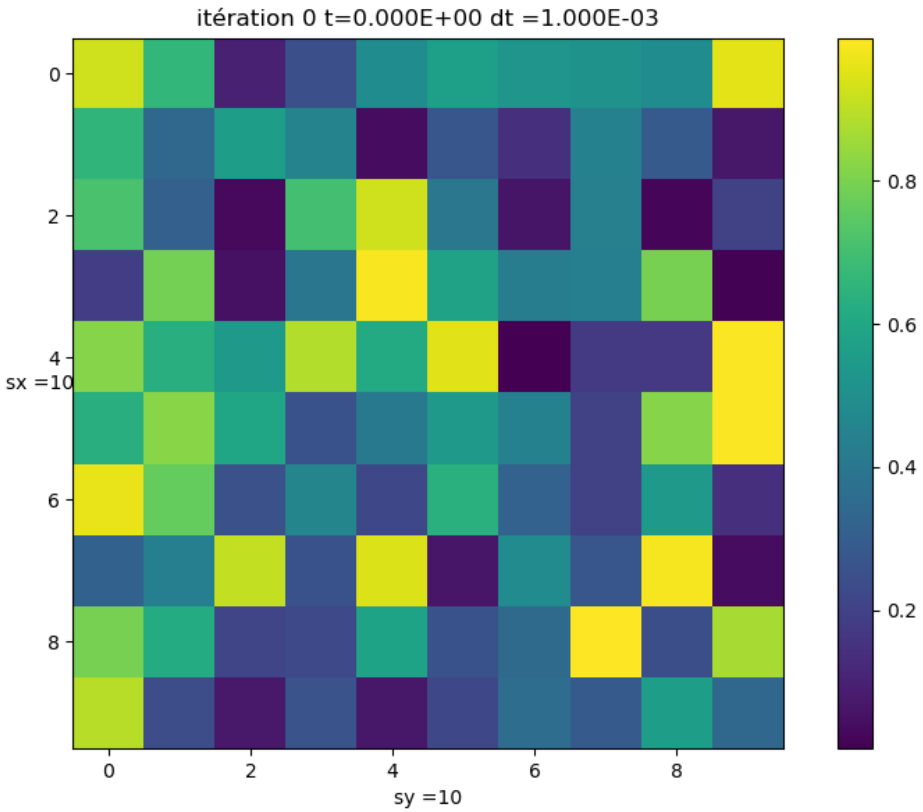


Figure: simulation avec $\gamma = 1e\text{-}2$

Paramètre de pénalisation γ

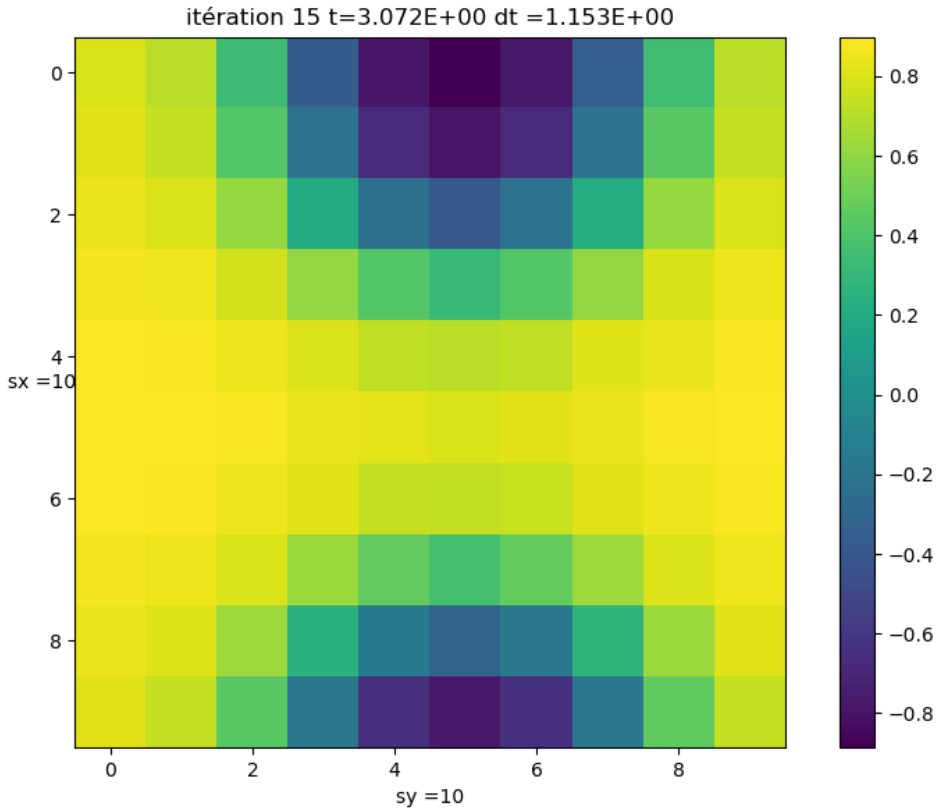


Figure: simulation avec $\gamma = 1e\text{-}2$

Paramètre de pénalisation γ

résultats :

$$\begin{cases} \text{temps de calcul} &= 10.84 \text{ secondes} \\ \text{temps final t} &= 3.07 \\ \text{nombre d'itérations} &= 14 \\ \text{minimum de la solution} &= -1.023 \\ \text{maximum de la solution} &= 9.94\text{e-}1 \end{cases}$$

Résolution de l'équation avec le contrôle V

On exécute le programme pour resoudre le système d'EDP en prenant un $u_0(x)$ qui soit un peu proche de $\hat{u}$, on fixe les paramètres comme suit:

$$\begin{cases} \text{nombre de points} &= 8 \\ \Delta x = \Delta y &= \text{1.43e-1} \\ \gamma &= \text{1e-4} \\ \Delta t_0 &= \text{1e-3} \\ \alpha &= 1.6 \\ \beta &= 0.8 \\ \text{T final} &= 0.1 \end{cases}$$

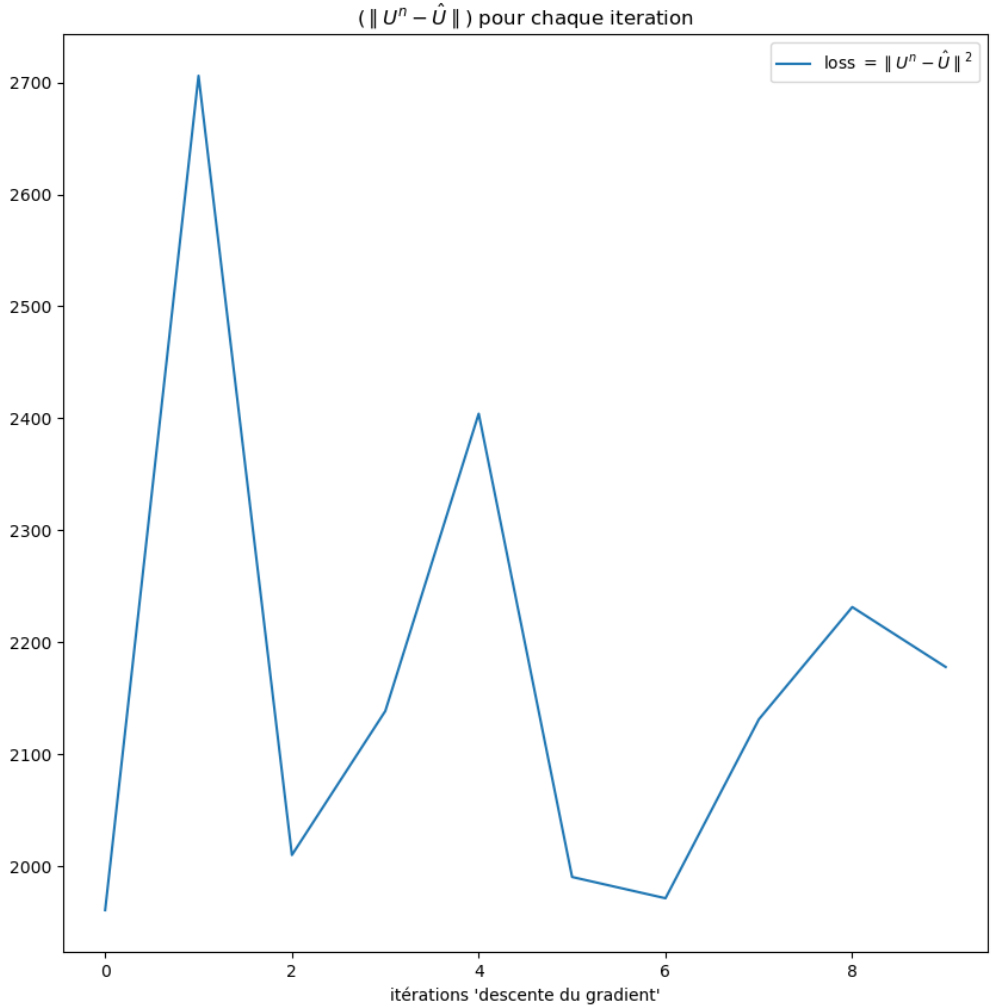
On obtient:

Résolution de l'équation avec le contrôle V

- D'abord le résultat de l'algorithme du gradient pour obtenir le $Vopt$:

A la fin de ce dernier, on a le graphe de la loss en fonction des itérations, dans cet exemple le nombre d'itérations est de 10:

Résolution de l'équation avec le contrôle V



Résolution de l'équation avec le contrôle V

On obtient le V optimal:

$$V = (0.0409, -0.6593, 0.1405)$$

avec lequel on passe à la deuxième étape,

- l'exécution du solveur CH :

En 8 itérations, on peut avoir la comparaison suivante :

Résolution de l'équation avec le contrôle V

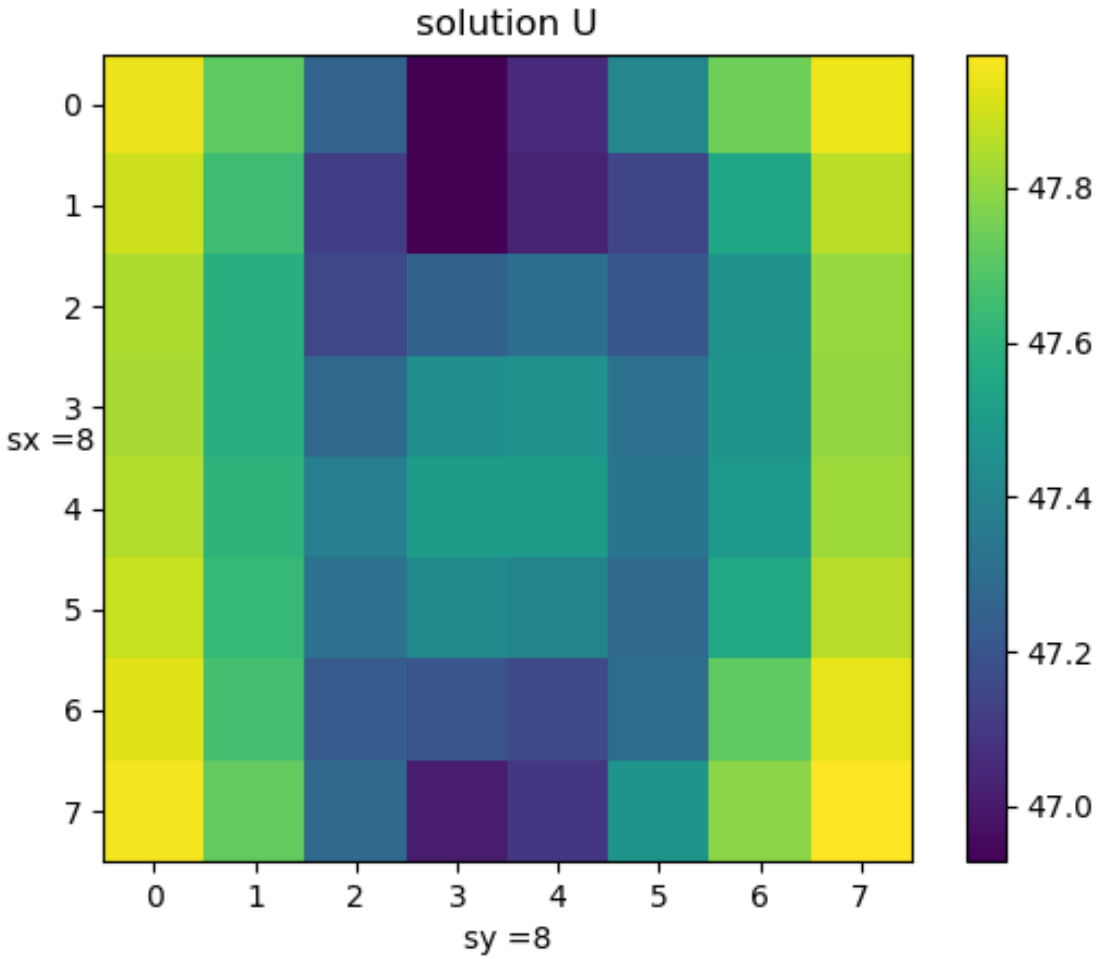
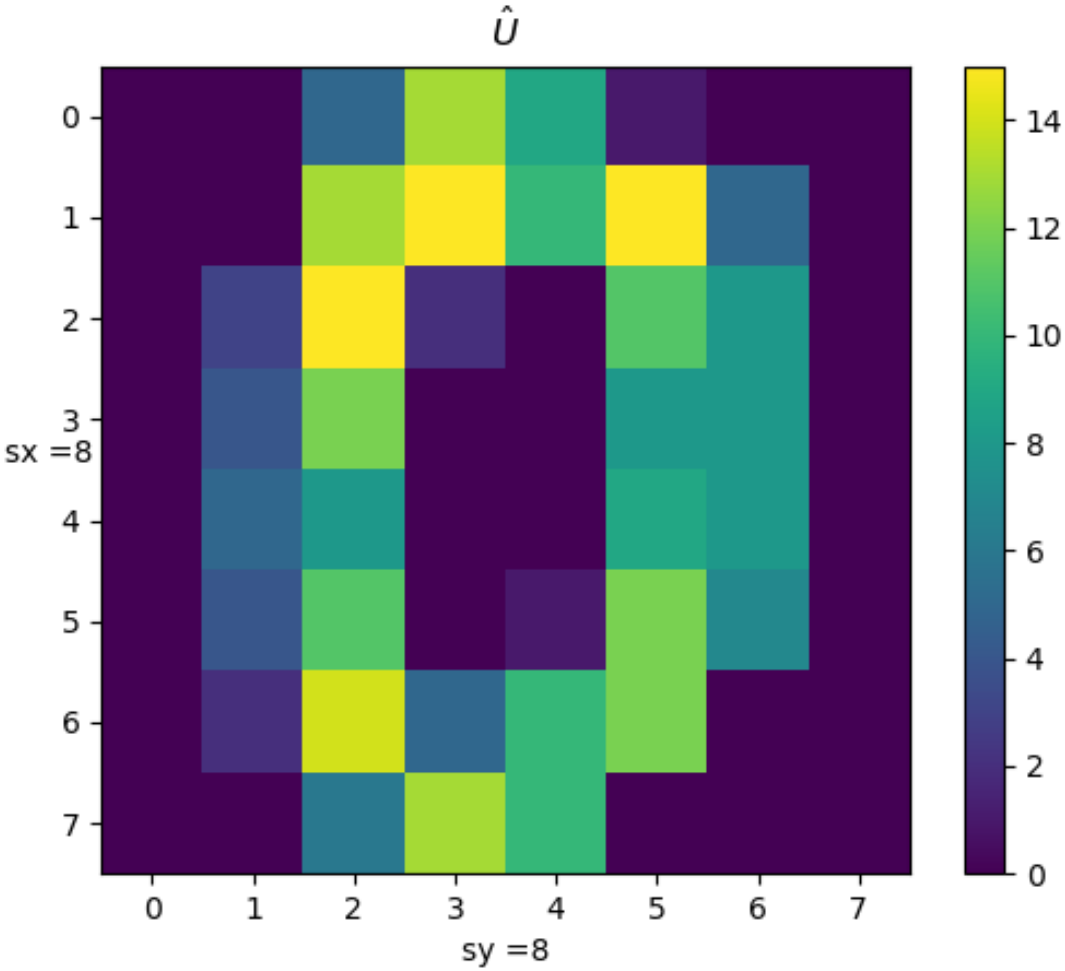


Figure: simulation avec le V optimal

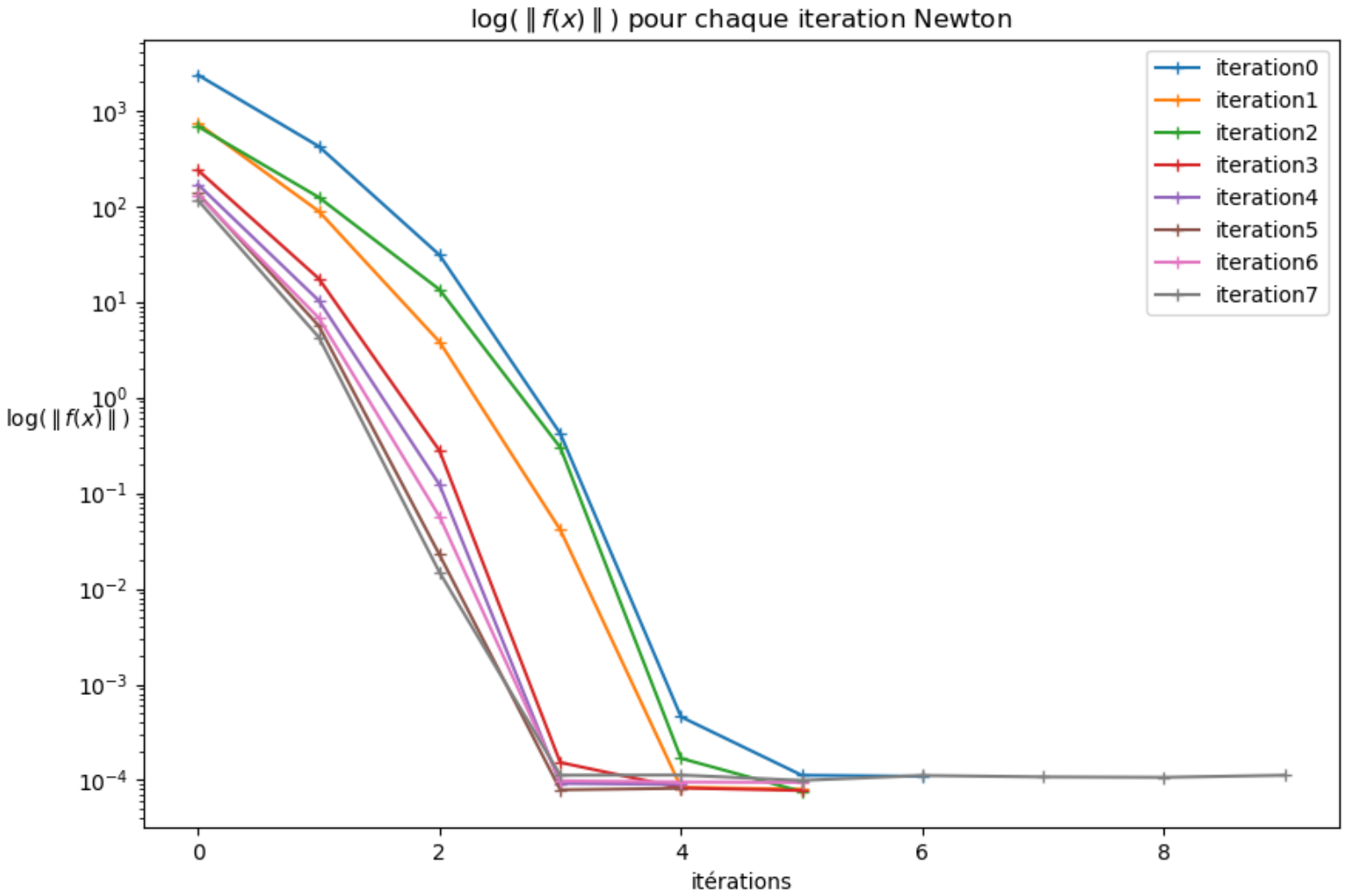
Résolution de l'équation avec le contrôle V



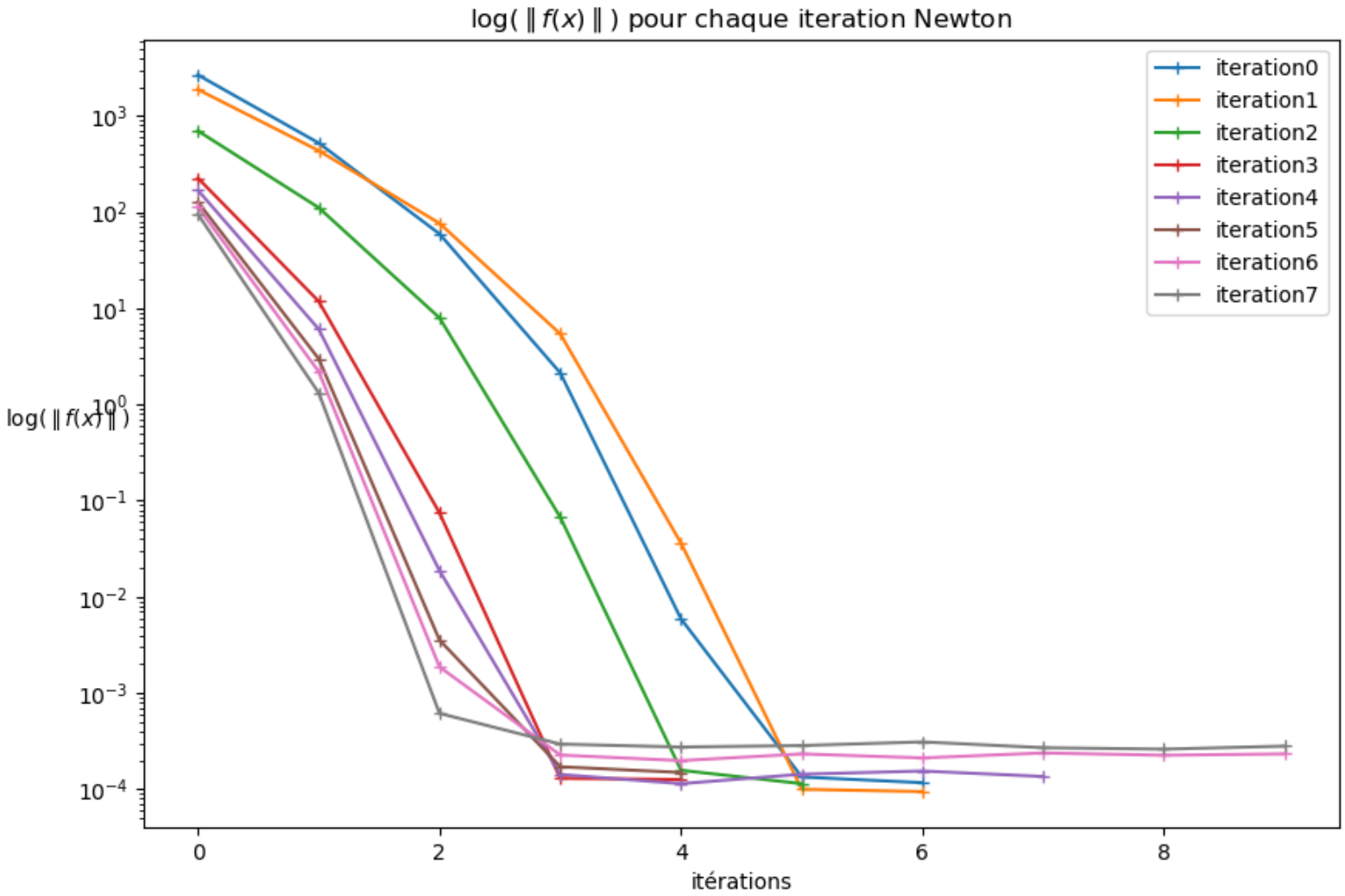
Résolution de l'équation avec le contrôle V

Pour s'assurer que la solution est convergente, on regarde les graphes en semilog de $J(X)$ en fonction des itérations de l'algorithme de Newton:

Résolution de l'équation avec le contrôle V



Résolution de l'équation avec le contrôle V



Conclusion

Même si la dernière étape n'a pas totalement abouti, le stage a permis d'approcher le modèle de Cahn-Hilliard sous d'une manière intéressante et qui peut être développée et utilisée dans la classification, ou dans le filtrage d'image par exemple.

Dans un futur travail, il serait prenant d'approfondir l'étude en abordant d'une autre façon le modèle de Cahn-Hilliars et d'essayer d'introduire et de définir le contrôle de plusieurs manières.

- C.M. Elliott, D.A. French and F.A. Milner, *A Second Order Splitting Method for the Cahn-Hiiliard Equation* , Numerische Mathematik, volume 54,pages: 575-590, 1989.
- Junseok Kim, Seunggyu Lee, Yongho Choi, Seok-Min Lee and Darae Jeong, *Basic Principles and Practical Applications of the Cahn–Hilliard Equation*, Hindawi Publishing Corporation, volume 2016, 27 Oct 2016.
- J.W. Cahn and J. E. Hilliard, *Free energy of a nonuniform system I, Interfacial free energy*, J. Chem. Phys. 28 (1958), 258-267.