Projet de fin de semestre

PELTIER Hugo / GLUCINA Candice

25 mai 2025

ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

Module : Équation aux dérivées partielles

Équation d'Allen-Cahn

Remerciements

Nous tenons à remercier notre chargé de matière M.YAKOUBI et notre chargé de matière M.ANDRIANTSITOHANINA pour leurs précieux conseils et leur accompagnement tout au long de ce semestre.

Table des matières

1 Introduction

1.1 Références bibliographiques

Références

[1] Wikipédia – *Équation d'Allen-Cahn*,
https://fr.wikipedia.org/wiki/%C3%89quation_d%27Allen-Cahn

[2] YouTube – *Simulation de l'équation d'Allen-Cahn*,
https://www.youtube.com/watch?v=yX0EAxZHNCQ

[3] 123dok.net – *Équation Allen-Cahn, modèles champs de phase*,
https://123dok.net/article/%C3%A9quation-allen-cahn-mod%C3%A8les-champs-phase-%C3%A9tude-cristallisation.yr2wn27z

[4] Université de La Rochelle – *Équations de champ de phase*,
https://lasie.univ-larochelle.fr/01-c-Equations-de-champ-de-phase

[5] ResearchGate – *Simulations of phase separation processes of Allen-Cahn equation on Torus*,
https://www.researchgate.net/figure/Simulations-of-phase-separation-processes-of-Allen-Cahn-equat
fig4_349964142

1.2 Présentation du modèle

L'équation d'Allen-Cahn étudiée est :

$$\frac{\partial u}{\partial t} - \epsilon \Delta u + f(u) = 0, \quad (x,t) \in \Omega \times (0,T) \tag{1}$$

Avec les conditions aux limites :

$$\begin{cases} u(x,0) = u_0(x), & x \in \Omega \\ u(x,t) = 0, & x \in \partial\Omega \end{cases}$$

Et :

$$f(u) = F'(u) \quad \text{et} \quad F(u) = \frac{1}{4}(u^2-1)^2$$

$$\frac{\partial u}{\partial t} = \epsilon \Delta u - f(u), \qquad \text{avec } f(u) = u^3 - u. \tag{2}$$

L'équation d'Allen-Cahn est une équation aux dérivées partielles (EDP) non linéaire du second ordre. Cette équation modélise souvent la séparation de phases.
où :

- $\epsilon > 0$ est un petit paramètre de diffusion,
- Δu désigne le Laplacien de u,
- $f(u)$ est un terme de réaction doublement stable :

$$\Rightarrow \quad f(u) = u^3 - u$$

2 Developpement

2.1 Analyse mathématique du problème

Pour mieux comprendre le comportement de l'équation d'Allen-Cahn, nous considérons une linéarisation autour de $u \approx 0$. Sans cette approximation, f(u) nous donne un terme non séparable qui empêchera la sépartaion des variables. En développant $f(u) = u^3 - u$ en série de Taylor à l'ordre 1, on obtient :

$$f(u) \approx -u$$

L'équation devient alors :

$$\frac{\partial u}{\partial t} = \epsilon \Delta u + u$$

Nous supposons une solution sous forme séparée :

$$u(x,t) = X(x)T(t) \quad (x,t) \in (0,L) \times (0,T)$$

En applicant la méthode de séparation des variables vue en TD, avec les conditions de Dirichlet, qui impliquent $X(0) = X(L) = 0$ on obtient :

$$u(x,t) = e^{-\lambda_n t} \sin\left(\frac{n\pi x}{L}\right)$$

où la valeur de λ_n est :

$$\lambda_n = \varepsilon \left(\frac{n\pi}{L}\right)^2 - 1$$

D'après cette solution en 1D, on observe que le système évolue de façon sinusoïdale avec le terme sinus et il décroit rapidement de manière exponentielle dans le temps. Le paramètre ε joue un rôle essentiel car si sa valeur est élevée, cela amplifiera la diffusion et donc lissera rapidement les variations spatiales. A l'inverse, si sa valeur est faible cet effet sera ralenti et des instabilités pourront apparaitre.

2.2 Présentation et anaylse des méthodes choisies

2.2.1 Schéma semi-implicite

Nous choisissons le schéma semi-implicite pour avoir une meilleure stabilité numérique sans que cette méthode soit trop difficile à implémenter. En traitant la diffusion de manière implicite et la non-linéarité de manière explicite, il est possible d'utiliser des pas de temps très grands sans que la stabilité soit impactée.

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = \varepsilon \Delta u_{i,j}^{n+1} - f(u_{i,j}^{n})$$

où :
- le terme de diffusion est traité implicitement (calculé à l'instant $n+1$),
- le terme de réaction est traité explicitement (calculé à l'instant n).

Différences finies centrales :

$$\Delta u_{i,j}^{n+1} \approx \frac{u_{i+1,j}^{n+1} + u_{i-1,j}^{n+1} + u_{i,j+1}^{n+1} + u_{i,j-1}^{n+1} - 4u_{i,j}^{n+1}}{\Delta x^2}$$

En réorganisant l'équation, on obtient la formule de mise à jour :

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \Delta t\, \varepsilon\, \Delta u_{i,j}^{n+1} - \Delta t\, f(u_{i,j}^{n})$$

2.2.2 Schéma de PICARD

Nous utilisons cette méthode pour traiter la non-linéarité présente dans l'équation d'Allen-Cahn. Elle consiste à effectuer des itérations successives pour approcher la solution du problème non linéaire, tout en gardant une approche simple et efficace, ce qui nous évite l'utilisation de méthodes plus lourdes comme celle de Newton.

Nous cherchons à résoudre l'équation discrétisée :

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = \varepsilon \Delta u_{i,j}^{n+1} - f(u_{i,j}^{n+1}) \tag{3}$$

La méthode de Picard consiste à linéariser le terme non linéaire $f(u_{i,j}^{n+1})$ en utilisant une approximation basée sur une valeur précédente $u_{i,j}^{(k)}$:

$$f(u_{i,j}^{n+1}) \approx f(u_{i,j}^{(k)}) \quad \text{où } k \text{ est l'indice d'itération de Picard.} \tag{4}$$

Le schéma devient alors :

$$u_{i,j}^{(k+1)} = u_{i,j}^{n} + \Delta t\, \varepsilon \Delta u_{i,j}^{(k+1)} - \Delta t\, f(u_{i,j}^{(k)}) \tag{5}$$

La boucle d'itération est répétée jusqu'à convergence, c'est-à-dire jusqu'à ce que :

$$\left\| u^{(k+1)} - u^{(k)} \right\| < \text{tolérance fixée} \tag{6}$$

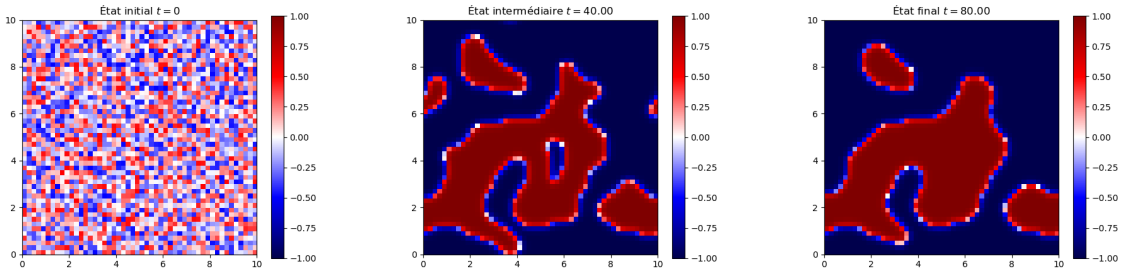
3 Résultat numériques

3.1 Présentation des simluations

Pour analyser le phénomène de séparation des phases, nous avons utiliser les deux schémas présentés pour faire différentes simulations. Pour chaque méthode, nous avons défini les mêmes conditions initiales et les mêmes paramètres afin de pouvoir comparer les résultats. Nous avons choisi d'observer l'évolution du champ $u(x, y, t)$ et de mesurer la stabilisation autour des deux états d'équilibre $+1$ et -1.

Le schèma semi-implicite donne une séparation de phases comme attendu mais légèrement moins stable et propre qu'avec l'utilisation de la méthode de Picard.

3.1.1 Simulation du schéma semi-implicite



(a) État initial ($t = 0$) (b) État intermédiaire ($t = T/2$) (c) État final ($t = T$)

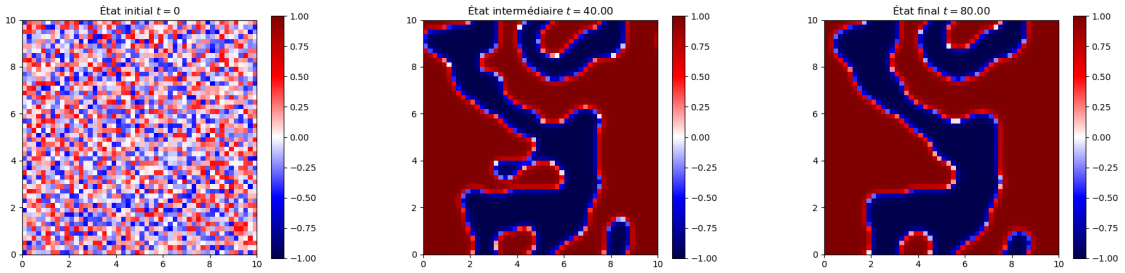
FIGURE 1 – Évolution au cours du temps simulé par l'équation d'Allen-Cahn.

L'animation de l'évolution de $u(x, y, t)$ montre le processus classique de séparation de phases modélisé par l'équation d'Allen-Cahn.
À partir d'un état initial complètement désordonné, on observe la formation progressive de domaines où u se stabilise autour de $+1$ et -1. Ces domaines grandissent au fur et à mesure de l'évolution temporelle.

Les interfaces séparant les deux phases deviennent de plus en plus lisses sous l'effet de la diffusion, tandis que les petites structures disparaissent.
Avec le temps, le système évolue pour former de plus grands domaines rouge et bleu, avec de moins en moins de frontières entre eux. Cette évolution montre que le système cherche naturellement à réduire le nombre de zones de transition, pour devenir plus stable.

3.1.2 Simulation du schéma de PICARD

Nous avons fait la même simulation avec le schéma de Picard, le résultat est assez similaire, mais en animant la simulation au cours du temps la séparation se fait légèrement plus rapidement et avec une meilleure qualité.
La stabilisation observée confirme l'efficacité de la méthode de Picard pour simuler la séparation de phases.



(a) État initial ($t = 0$) (b) État intermédiaire ($t = T/2$) (c) État final ($t = T$)

FIGURE 2 – Évolution au cours du temps simulé par l'équation d'Allen-Cahn avec la méthode de Picard.

Nous avons donc choisi de présenter les résultats de nos autres simulations réalisés à partir de ce schéma.

Pour étudier la stabilité du système, nous traçons l'évolution de l'énergie libre en fonction du temps avec la méthode de Picard.

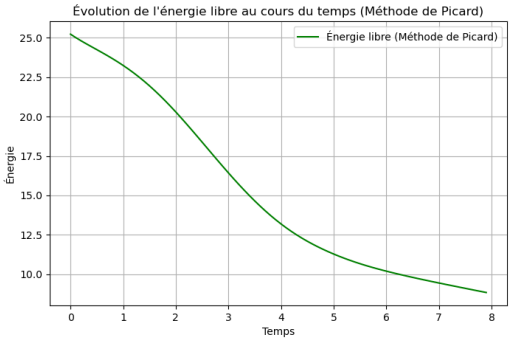


FIGURE 3 – Évolution de l'énergie libre au cours du temps pour la méthode de Picard.

La courbe montre que l'énergie libre décroît progressivement au cours du temps, ce qui correspond bien à la tendance naturelle du système à évoluer vers des états plus stables.
Nous observons que la baisse de l'énergie est relativement régulière tout au long de la simulation.
Par rapport au schéma semi-implicite, nous avons une stabilité qui s'obtient légèrement plus rapidement et de manière plus régulière.

La stabilisation observée nous confirme l'efficacité de la méthode de Picard pour simuler la séparation de phases.

Nous présentons maintenant l'évolution de la proportion de points proches de $+1$ et -1 au cours du temps.

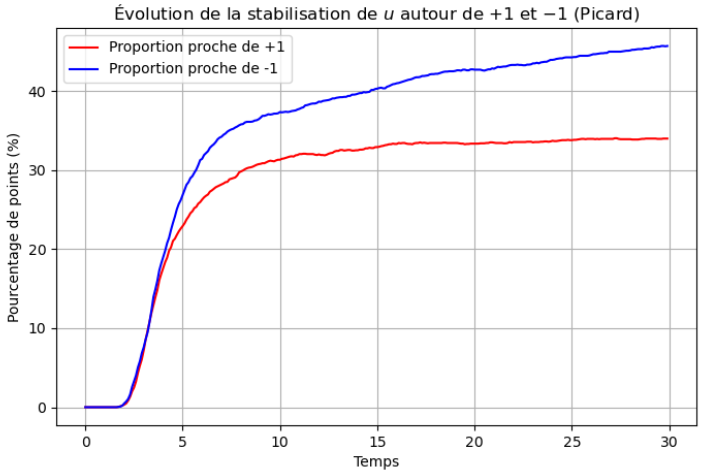


FIGURE 4 – Évolution de la proportion de points proches de $+1$ et -1 au cours du temps.

La courbe montre que, dès $t \approx 5$, une grande partie des points converge vers $+1$ ou -1. À la fin de la simulation, environ 47% des points sont proches de -1 et 34% proches de $+1$, la séparation des phases est bien marquée.

La méthode de Picard permet ainsi d'obtenir une stabilisation rapide et nette des domaines.

3.2 Validation par solution manufacturée

Afin de valider numériquement notre méthode, nous avons utilisé une solution exacte artificielle (dite "solution manufacturée") :

$$u_{\text{exact}}(x, y, t) = e^{-t} \sin(\pi x) \sin(\pi y)$$

Cette fonction satisfait une version modifiée de l'équation d'Allen-Cahn comportant un terme source :

$$f_{\text{source}}(x, y, t, u) = e^{-t} \sin(\pi x) \sin(\pi y)(1 + 2\pi^2 \epsilon) + u^3 - u$$

Ce terme a été ajouté dans notre schéma explicite pour s'assurer que u_{exact} est bien solution.

Nous avons comparé la solution approchée u_{num} à la solution exacte à l'instant $t = 1$. L'erreur quadratique L^2 obtenue via un code python en annexe est :

$$\text{Erreur}_{L^2} = 5.98 \times 10^{-1}$$

Ce résultat permet de vérifier que notre méthode est cohérente et reproduit correctement une solution connue lorsque les conditions sont bien contrôlées.

3.3 Effet d'un terme de transport

Nous avons étudié une version modifiée de l'équation d'Allen-Cahn avec un terme de transport :

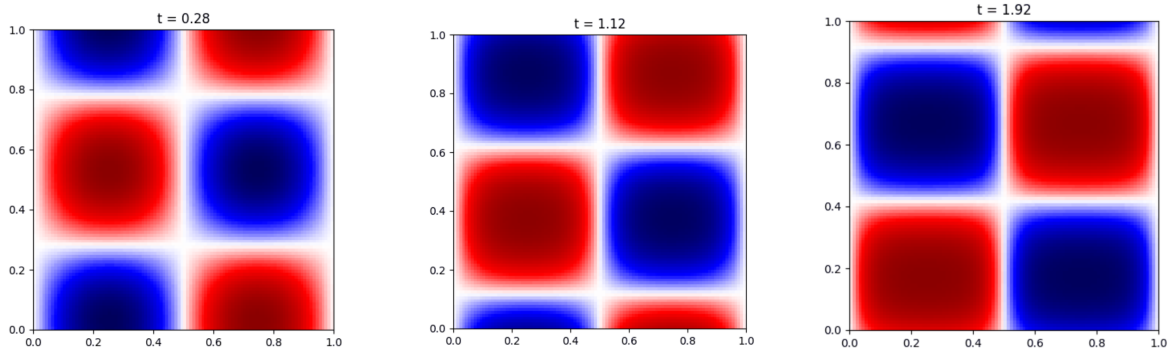
$$\frac{\partial u}{\partial t} + \vec{v} \cdot \nabla u = \epsilon \Delta u - f(u)$$

où $\vec{v} = (1, 0)$ représente un champ de vitesse constant horizontal.

Le transport est discrétisé à l'aide d'un *schéma amont* :

$$\vec{v} \cdot \nabla u \approx \frac{u_{i,j} - u_{i-1,j}}{\Delta x}$$

Pour observer visuellement l'effet du transport, nous avons simulé l'évolution du champ $u(x, y, t)$ avec une condition initiale structurée, une faible diffusion ($\epsilon = 0{,}001$), et un temps d'évolution court.
Les résultats montrent un décalage clair des motifs sinusoïdaux vers la droite, comme illustré ci-dessous à trois instants.



(a) $t = 0.28$ (b) $t = 1.12$ (c) $t = 1.92$

FIGURE 5 – Effet du transport horizontal sur la structure du champ $u(x, y, t)$ dans l'équation d'Allen-Cahn. Les motifs se déplacent progressivement vers la droite.

3.4 Conclusion

L'équation d'Allen-Cahn possède de nombreuses applications dans les domaines de la physique des matériaux, de la biologie et du traitement d'images.

Dans l'industrie des matériaux composites et de la métallurgie elle est utilisée pour modéliser la séparation de phases pour prédire la formation de microstructures et optimiser les propriétés mécaniques des matériaux.

Dans le domaine biologique, elle sert à décrire la dynamique de ségrégation dans les membranes cellulaires, où différentes phases lipidiques peuvent apparaître.

Enfin, en informatique, des méthodes issues du modèle d'Allen-Cahn sont exploitées pour la segmentation d'images, en identifiant de manière automatique les différentes régions d'une image selon leurs caractéristiques.

Dans cette étude, nous avons appliqué un schéma semi-implicite et la méthode de Picard pour simuler la dynamique de séparation de phases.

Les résultats montrent que l'équation d'Allen-Cahn permet de modéliser efficacement la stabilisation d'un système en deux phases distinctes. On observe également que la méthode de Picard améliore la précision des interfaces et accélère la stabilisation par rapport au schéma semi-implicite.

Ces simulations confirment l'intérêt de ce type de méthode numérique pour représenter l'évolution de systèmes complexes, notamment dans le domaine des matériaux, de la biologie ou du traitement d'images.

4 Annexe

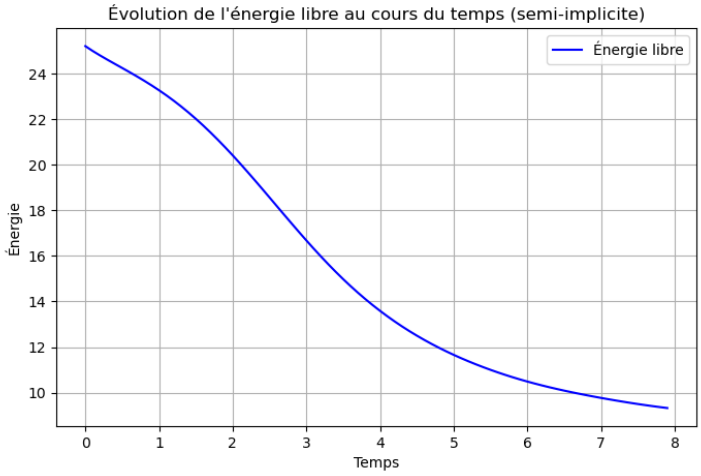


FIGURE 6 – Évolution de l'énergie libre au cours du temps pour le schéma semi-implicite.

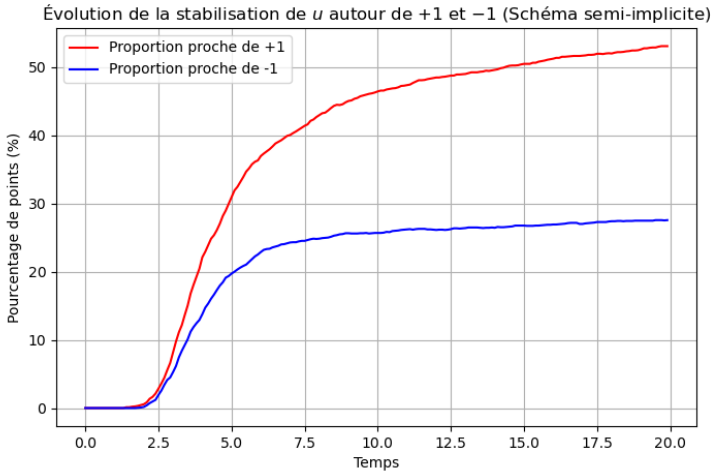


FIGURE 7 – Évolution de la proportion de points proches de $+1$ et -1 au cours du temps (Schéma semi-implicite).

Code Python - Simulation Allen-Cahn avec Picard et GIF

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
Lx, Ly = 10.0, 10.0
Nx, Ny = 50, 50
dx, dy = Lx / Nx, Ly / Ny
epsilon = 0.01
dt = 0.001
Tmax = 80.0
steps = int(Tmax / dt)
x = np.linspace(0, Lx, Nx)
y = np.linspace(0, Ly, Ny)
X, Y = np.meshgrid(x, y)
u = np.random.uniform(-0.5, 0.5, (Nx, Ny))
def laplacian(u):
```

```
    return (np.roll(u, 1, axis=0) + np.roll(u, -1, axis=0) +
            np.roll(u, 1, axis=1) + np.roll(u, -1, axis=1) - 4 * u) / dx**2
snapshots = []
capture_interval = 50
max_iter = 20
tolerance = 1e-4
for n in range(steps):
    u_old = u.copy()
    u_new = u.copy()
    for k in range(max_iter):
        lap_u = laplacian(u_new)
        f_u_old = u_old**3 - u_old
        u_next = u_old + dt * (epsilon * lap_u - f_u_old)
        if np.linalg.norm(u_next - u_new, ord=np.inf) < tolerance:
            break
        u_new = u_next
    u = u_new
    if n % capture_interval == 0:
        snapshots.append(u.copy())
fig, ax = plt.subplots(figsize=(6,5))
img = ax.imshow(snapshots[0], extent=[0, Lx, 0, Ly], origin='lower', cmap='seismic',
    ↪ vmin=-1, vmax=1)
def animate(i):
    img.set_data(snapshots[i])
    ax.set_title(f"Temps = {i * capture_interval * dt:.2f}")
    return [img]
ani = animation.FuncAnimation(fig, animate, frames=len(snapshots), interval=100, blit=
    ↪ True)
ani.save(r'C:\Users\hugop\OneDrive\Bureau\evolution_allen_cahn_picard.gif', writer='
    ↪ pillow', fps=10)
plt.close()
```

Listing 1 – Simulation avec méthode de Picard et sauvegarde GIF

Code Python - États intermédiaires (figures fixes)

```
plt.figure()
plt.imshow(u, extent=[0, Lx, 0, Ly], origin='lower', cmap='seismic', vmin=-1, vmax=1)
plt.title("etat initial $t=0$")
plt.colorbar()
plt.savefig("etat_initial_picard.png")
plt.close()
snapshot_intermediate = steps // 2
snapshot_final = steps - 1
for n in range(steps):
    u_old = u.copy()
    u_new = u.copy()
    for k in range(max_iter):
        lap_u = laplacian(u_new)
        f_u_old = u_old**3 - u_old
        u_next = u_old + dt * (epsilon * lap_u - f_u_old)
        if np.linalg.norm(u_next - u_new, ord=np.inf) < tolerance:
            break
        u_new = u_next
    u = u_new
    if n == snapshot_intermediate:
        plt.figure()
        plt.imshow(u, extent=[0, Lx, 0, Ly], origin='lower', cmap='seismic', vmin=-1,
            ↪ vmax=1)
        plt.title(f"etat intermediaire $t={n*dt:.2f}$")
        plt.colorbar()
        plt.savefig("etat_intermediaire_picard.png")
        plt.close()
    if n == snapshot_final:
        plt.figure()
        plt.imshow(u, extent=[0, Lx, 0, Ly], origin='lower', cmap='seismic', vmin=-1,
            ↪ vmax=1)
        plt.title(f"etat final $t={n*dt:.2f}$")
        plt.colorbar()
```

```
        plt.savefig("etat_final_picard.png")
        plt.close()
```

Listing 2 – Sauvegarde des états initial, intermediaire et final

Code Python - Solution manufacturée et erreur L^2

```
import numpy as np
import matplotlib.pyplot as plt
epsilon = 0.01
Lx = Ly = 1.0
Nx = Ny = 50
dx = Lx / Nx
dt = 0.001
T = 1.0
Nt = int(T / dt)
x = np.linspace(0, Lx, Nx)
y = np.linspace(0, Ly, Ny)
X, Y = np.meshgrid(x, y, indexing='ij')
def u_exact(x, y, t):
    return np.exp(-t) * np.sin(np.pi * x) * np.sin(np.pi * y)
def f_source(x, y, t, u_val):
    return np.exp(-t) * np.sin(np.pi * x) * np.sin(np.pi * y) * (1 + 2 * np.pi**2 *
        ↪ epsilon) + u_val**3 - u_val
u = u_exact(X, Y, 0)
for n in range(Nt):
    t = n * dt
    lap_u = (np.roll(u, 1, axis=0) + np.roll(u, -1, axis=0) +
             np.roll(u, 1, axis=1) + np.roll(u, -1, axis=1) - 4 * u) / dx**2
    src = f_source(X, Y, t, u)
    u += dt * (epsilon * lap_u - u**3 + u + src)
u_ref = u_exact(X, Y, T)
error = np.sqrt(np.sum((u - u_ref)**2) * dx * dx)
print(f"Erreur L2 entre u_num et u_exact : {error:.5e}")
```

Listing 3 – Validation par solution exacte manufacturée

Code Python – Simulation Allen-Cahn avec schéma semi-implicite

```
import numpy as np
import matplotlib.pyplot as plt
Lx, Ly = 10.0, 10.0
Nx, Ny = 50, 50
dx, dy = Lx / Nx, Ly / Ny
epsilon = 0.01
dt = 0.001
Tmax = 80.0
steps = int(Tmax / dt)
x = np.linspace(0, Lx, Nx)
y = np.linspace(0, Ly, Ny)
X, Y = np.meshgrid(x, y)
u = np.random.uniform(-0.5, 0.5, (Nx, Ny))
def laplacian(u):
    return (
        np.roll(u, 1, axis=0) + np.roll(u, -1, axis=0) +
        np.roll(u, 1, axis=1) + np.roll(u, -1, axis=1) - 4 * u
    ) / dx**2
plt.figure(figsize=(6,5))
plt.imshow(u, extent=[0, Lx, 0, Ly], origin='lower', cmap='seismic', vmin=-1, vmax=1)
plt.title("etat initial $t=0$")
plt.colorbar()
plt.savefig("etat_initial.png")
plt.close()
snapshot_intermediate = steps // 2
snapshot_final = steps - 1
for n in range(steps):
    lap_u = laplacian(u)
```

```
    u = u + dt * epsilon * lap_u - dt * (u**3 - u)
    if n == snapshot_intermediate:
        plt.figure(figsize=(6,5))
        plt.imshow(u, extent=[0, Lx, 0, Ly], origin='lower', cmap='seismic', vmin=-1,
            ↪ vmax=1)
        plt.title(f"etat intermediaire $t={n*dt:.2f}$")
        plt.colorbar()
        plt.savefig("etat_intermediaire.png")
        plt.close()
    if n == snapshot_final:
        plt.figure(figsize=(6,5))
        plt.imshow(u, extent=[0, Lx, 0, Ly], origin='lower', cmap='seismic', vmin=-1,
            ↪ vmax=1)
        plt.title(f"etat final $t={n*dt:.2f}$")
        plt.colorbar()
        plt.savefig("etat_final.png")
        plt.close()
```

Listing 4 – Simulation avec schéma semi-implicite

Code Python – Énergie libre (Schéma semi-implicite)

```
import numpy as np
import matplotlib.pyplot as plt
Lx, Ly = 10.0, 10.0
Nx, Ny = 50, 50
dx, dy = Lx / Nx, Ly / Ny
epsilon = 0.01
dt = 0.01
Tmax = 8.0
steps = int(Tmax / dt)
x = np.linspace(0, Lx, Nx)
y = np.linspace(0, Ly, Ny)
X, Y = np.meshgrid(x, y)
u = np.random.uniform(-0.5, 0.5, (Nx, Ny))
def laplacian(u):
    return (
        np.roll(u, 1, axis=0) + np.roll(u, -1, axis=0) +
        np.roll(u, 1, axis=1) + np.roll(u, -1, axis=1) - 4 * u
    ) / dx**2
def energy(u):
    grad_u_x = (np.roll(u, -1, axis=0) - u) / dx
    grad_u_y = (np.roll(u, -1, axis=1) - u) / dy
    grad_norm_sq = grad_u_x**2 + grad_u_y**2
    F_u = 0.25 * (u**2 - 1)**2
    return np.sum(0.5 * epsilon * grad_norm_sq + F_u) * dx * dy
energies = []
times = []
for n in range(steps):
    lap_u = laplacian(u)
    u = u + dt * (epsilon * lap_u - (u**3 - u))
    if n % 10 == 0:
        energies.append(energy(u))
        times.append(n * dt)
plt.figure(figsize=(8,5))
plt.plot(times, energies, label="energie libre", color="blue")
plt.xlabel("Temps")
plt.ylabel("energie")
plt.title("evolution de l'energie libre au cours du temps (semi-implicite)")
plt.grid(True)
plt.legend()
plt.savefig("courbe_energie_semi_implicite.png")
plt.show()
```

Listing 5 – Évolution de l'énergie libre (schéma semi-implicite)

Code Python – Énergie libre (Méthode de Picard)

```python
import numpy as np
import matplotlib.pyplot as plt
Lx, Ly = 10.0, 10.0
Nx, Ny = 50, 50
dx, dy = Lx / Nx, Ly / Ny
epsilon = 0.01
dt = 0.01
Tmax = 8.0
steps = int(Tmax / dt)
x = np.linspace(0, Lx, Nx)
y = np.linspace(0, Ly, Ny)
X, Y = np.meshgrid(x, y)
u = np.random.uniform(-0.5, 0.5, (Nx, Ny))
def laplacian(u):
    return (
        np.roll(u, 1, axis=0) + np.roll(u, -1, axis=0) +
        np.roll(u, 1, axis=1) + np.roll(u, -1, axis=1) - 4 * u
    ) / dx**2
def energy(u):
    grad_u_x = (np.roll(u, -1, axis=0) - u) / dx
    grad_u_y = (np.roll(u, -1, axis=1) - u) / dy
    grad_norm_sq = grad_u_x**2 + grad_u_y**2
    F_u = 0.25 * (u**2 - 1)**2
    return np.sum(0.5 * epsilon * grad_norm_sq + F_u) * dx * dy
max_iter = 20
tolerance = 1e-4
energies = []
times = []
for n in range(steps):
    u_old = u.copy()
    u_new = u.copy()
    for k in range(max_iter):
        lap_u = laplacian(u_new)
        f_u_old = u_old**3 - u_old
        u_next = u_old + dt * (epsilon * lap_u - f_u_old)
        if np.linalg.norm(u_next - u_new, ord=np.inf) < tolerance:
            break
        u_new = u_next
    u = u_new
    if n % 10 == 0:
        energies.append(energy(u))
        times.append(n * dt)
plt.figure(figsize=(8,5))
plt.plot(times, energies, label="energie libre (Picard)", color="green")
plt.xlabel("Temps")
plt.ylabel("energie")
plt.title("evolution de l'energie libre au cours du temps (Methode de Picard)")
plt.grid(True)
plt.legend()
plt.savefig("courbe_energie_picard.png")
plt.show()
```

Listing 6 – Évolution de l'énergie libre (Picard)

Code Python – Stabilisation autour de ± 1 (Picard)

```python
import numpy as np
import matplotlib.pyplot as plt
Lx, Ly = 10.0, 10.0
Nx, Ny = 50, 50
dx, dy = Lx / Nx, Ly / Ny
epsilon = 0.01
dt = 0.01
Tmax = 30.0
steps = int(Tmax / dt)
x = np.linspace(0, Lx, Nx)
y = np.linspace(0, Ly, Ny)
X, Y = np.meshgrid(x, y)
u = np.random.uniform(-0.5, 0.5, (Nx, Ny))
```

```
def laplacian(u):
    return (
        np.roll(u, 1, axis=0) + np.roll(u, -1, axis=0) +
        np.roll(u, 1, axis=1) + np.roll(u, -1, axis=1) - 4 * u
    ) / dx**2
max_iter = 20
tolerance = 1e-4
times = []
percent_plus1 = []
percent_minus1 = []
for n in range(steps):
    u_old = u.copy()
    u_new = u.copy()
    for k in range(max_iter):
        lap_u = laplacian(u_new)
        f_u_old = u_old**3 - u_old
        u_next = u_old + dt * (epsilon * lap_u - f_u_old)
        if np.linalg.norm(u_next - u_new, ord=np.inf) < tolerance:
            break
        u_new = u_next
    u = u_new
    if n % 10 == 0:
        times.append(n * dt)
        plus1 = np.sum(u > 0.8) / (Nx * Ny) * 100
        minus1 = np.sum(u < -0.8) / (Nx * Ny) * 100
        percent_plus1.append(plus1)
        percent_minus1.append(minus1)
plt.figure(figsize=(8,5))
plt.plot(times, percent_plus1, label="Proche␣de␣+1", color='red')
plt.plot(times, percent_minus1, label="Proche␣de␣-1", color='blue')
plt.xlabel("Temps")
plt.ylabel("Pourcentage␣de␣points␣(%)")
plt.title("Stabilisation␣autour␣de␣$+1$␣et␣$-1$␣(Methode␣de␣Picard)")
plt.legend()
plt.grid(True)
plt.savefig("courbe_stabilisation_picard.png")
plt.show()
```

Listing 7 – Évolution de la stabilisation (Picard)

Code Python – Stabilisation autour de ± 1 (Semi-implicite)

```
import numpy as np
import matplotlib.pyplot as plt
Lx, Ly = 10.0, 10.0
Nx, Ny = 50, 50
dx, dy = Lx / Nx, Ly / Ny
epsilon = 0.01
dt = 0.01
Tmax = 20.0
steps = int(Tmax / dt)
x = np.linspace(0, Lx, Nx)
y = np.linspace(0, Ly, Ny)
X, Y = np.meshgrid(x, y)
u = np.random.uniform(-0.5, 0.5, (Nx, Ny))
def laplacian(u):
    return (
        np.roll(u, 1, axis=0) + np.roll(u, -1, axis=0) +
        np.roll(u, 1, axis=1) + np.roll(u, -1, axis=1) - 4 * u
    ) / dx**2
times = []
percent_plus1 = []
percent_minus1 = []
for n in range(steps):
    lap_u = laplacian(u)
    u = u + dt * (epsilon * lap_u - (u**3 - u))
    if n % 10 == 0:
        times.append(n * dt)
        plus1 = np.sum(u > 0.8) / (Nx * Ny) * 100
```

```
        minus1 = np.sum(u < -0.8) / (Nx * Ny) * 100
        percent_plus1.append(plus1)
        percent_minus1.append(minus1)

plt.figure(figsize=(8,5))
plt.plot(times, percent_plus1, label="Proche de +1", color='red')
plt.plot(times, percent_minus1, label="Proche de -1", color='blue')
plt.xlabel("Temps")
plt.ylabel("Pourcentage de points (%)")
plt.title("Stabilisation autour de $+1$ et $-1$ (Schema semi-implicite)")
plt.legend()
plt.grid(True)
plt.savefig("courbe_stabilisation_semi_implicite.png")
plt.show()
```

Listing 8 – Évolution de la stabilisation (semi-implicite)