Université Montpellier
Faculté des Sciences Montpellier
Département de Mathématiques et Informatique

Module : Estimation à Posteriori

Master 2 : Modélisation et Analyse Numérique (MANU)

Rapport du séance 4 et 5

Réalisé par : NAOUAL Arkhouch

Année universitaire
2024/2025

Table des matières

Introduction

Le but de ce projet est de développer un code capable de résoudre une équation aux dérivées partielles instationnaire, avec un terme source dépendant du temps, tout en optimisant l'adaptation du maillage pour améliorer la précision de la solution. Le projet vise à :
1) Résoudre une EDP avec un terme source variable.
2)Comprendre et gérer la non-convergence due à l'instationnarité.
3) Optimiser le maillage pour capturer efficacement les détails de la solution.
4) Introduire des critères d'arrêt mixtes basés sur le nombre de points de maillage et l'erreurL_2

0.1 La définition de L'équation aux dérivées partielles (EDP)

L'équation aux dérivées partielles (EDP) donnée par

$$\begin{cases} u_t + V(t,s)u_s - \nu u_{ss} = -\lambda u + f(t,s), \;\; s \in \Omega =]0, L[\;\; t \geq 0, \\ u(t,0) = u_g, \;\; u(t,L) = u_d \;\; \text{conditions aux limites de Dirichlet homogènes} \\ u(0,s) = u_0(s) \;\; \text{donnée : condition initiale} \end{cases} \tag{1}$$

On définit la solution exacte par :

$$U_{\text{ex}}(t.s) = U(t)Vs)$$

telque :

$$U(t)Vs) = sin(4\pi t)v(s)$$

Alors :

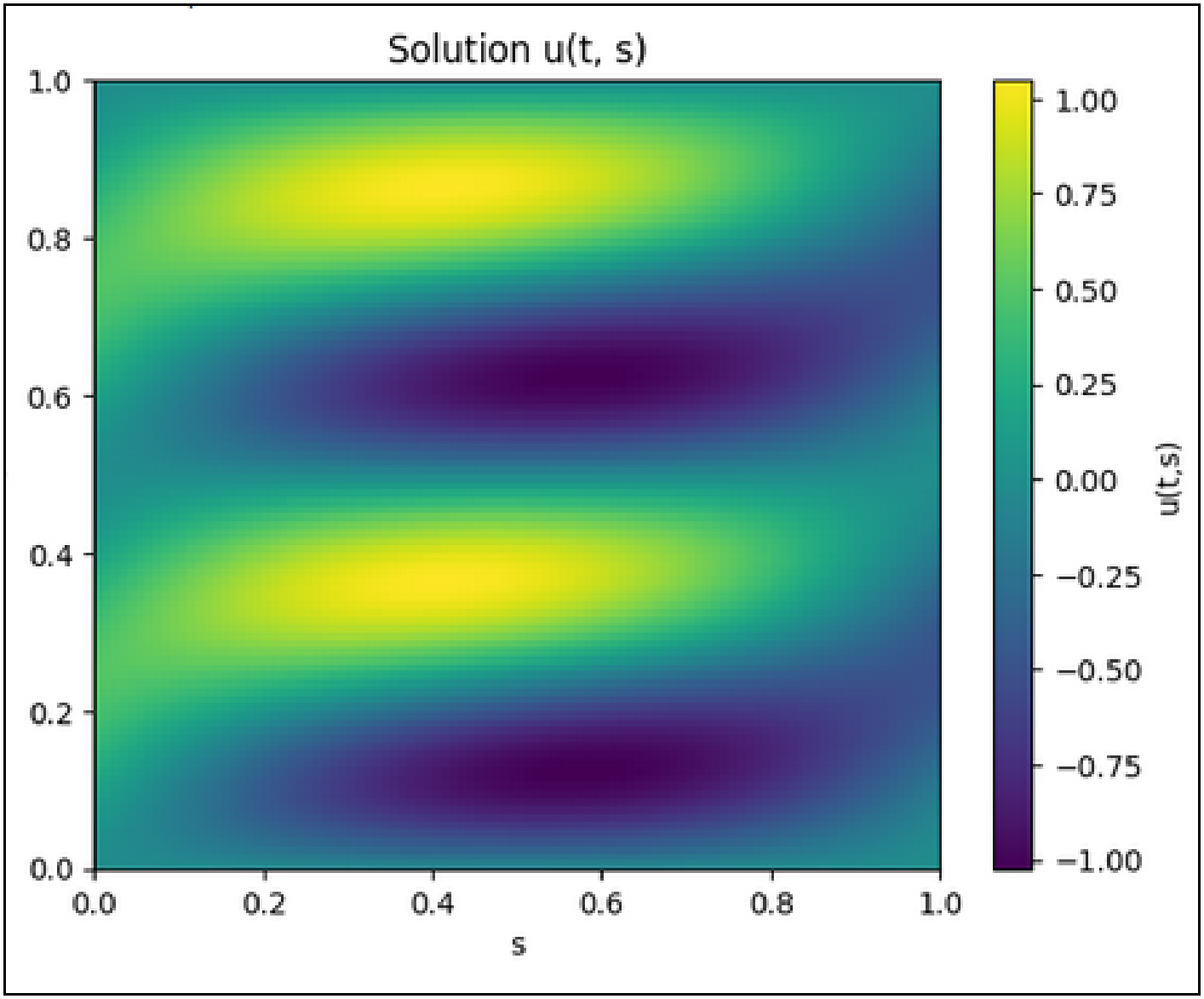
$$U'(t)Vs) = 4\pi cos(4\pi t)v(s)$$

0.2 Code Python pour la Solution

Dans la figure 1, elle représente le Code Python pour la solution .

```python
import numpy as np
import matplotlib.pyplot as plt

# Paramètres
nu, lambda_, time_steps, dt = 0.01, 1.0, 100, 0.01
mesh = np.linspace(0, 1, 100)

# Calcul de la solution
def solve_adrs():
    u_solution = np.zeros((time_steps, len(mesh)))
    for j, s in enumerate(mesh):
        u_solution[0, j] = np.sin(4 * np.pi * 0) * np.sin(np.pi * s)

    for t_step in range(1, time_steps):
        t = t_step * dt
        for j, s in enumerate(mesh):
            u_t = 4 * np.pi * np.cos(4 * np.pi * t)
            u_solution[t_step, j] = (u_solution[t_step - 1, j] +
                                     dt * (u_t * np.sin(np.pi * s) +
                                           np.sin(4 * np.pi * t) *
                                           np.pi * np.cos(np.pi * s) +
                                           nu * np.sin(4 * np.pi * t) *
                                           (-np.pi**2 * np.sin(np.pi * s)) +
                                           lambda_ * np.sin(4 * np.pi * t) *
                                           np.sin(np.pi * s)))
        print(f"Residue at step {t_step}: {np.linalg.norm(u_solution[t_step] - u_solution[t_step - 1])}")

    return u_solution

# Exécution et visualisation
plt.imshow(solve_adrs(), aspect='auto', extent=[0, 1, 0, time_steps * dt])
plt.colorbar(label='u(t,s)')
plt.xlabel('s')
plt.ylabel('t')
plt.title('Solution u(t, s)')
```

Dans la figure 2 , elle représente la solution.

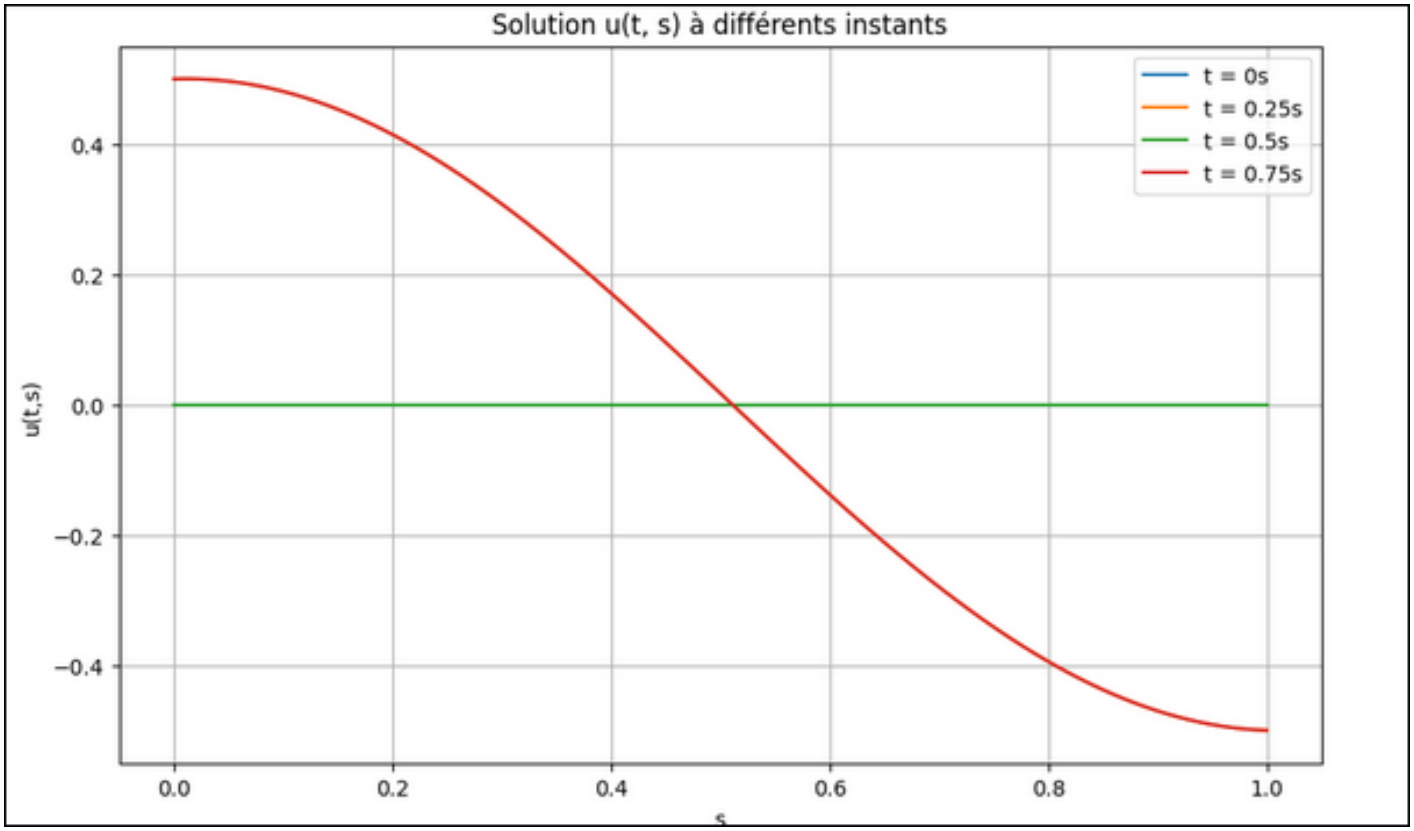


0.3 Visualisation à Différents Instants la solution

Dans la figure 3, elle représente le Code Résumé pour la Visualisation

```python
# Paramètres
nu, lambda_, time_steps, dt = 0.01, 1.0, 100, 0.01
mesh = np.linspace(0, 1, 100)

# Calcul de la solution
def solve_adrs():
    u_solution = np.zeros((time_steps, len(mesh)))
    for j, s in enumerate(mesh):
        u_solution[0, j] = np.sin(4 * np.pi * 0) * np.sin(np.pi * s)

    for t_step in range(1, time_steps):
        t = t_step * dt
        for j, s in enumerate(mesh):
            u_t = 4 * np.pi * np.cos(4 * np.pi * t)
            u_solution[t_step, j] = (u_solution[t_step - 1, j] +
                                     dt * (u_t * np.sin(np.pi * s) +
                                           np.sin(4 * np.pi * t) *
                                           np.pi * np.cos(np.pi * s) +
                                           nu * np.sin(4 * np.pi * t) *
                                           (-np.pi**2 * np.sin(np.pi * s)) +
                                           lambda_ * np.sin(4 * np.pi * t) *
                                           np.sin(np.pi * s)))
        print(f"Residue at step {t_step}: {np.linalg.norm(u_solution[t_step] - u_solution[t_step - 1])}")

    return u_solution

# Exécution de la simulation
u_solution = solve_adrs()

# Visualisation de la solution à différents instants
times_to_plot = [0, 0.25, 0.5, 0.75, 1.0]  # Temps spécifiques à visualiser
plt.figure(figsize=(10, 6))

for t in times_to_plot:
    t_step = int(t / dt)
    if t_step < time_steps:  # Vérifie que l'indice est valide
        plt.plot(mesh, u_solution[t_step], label=f't = {t}s')

plt.xlabel('s')
plt.ylabel('u(t,s)')
plt.title('Solution u(t, s) à différents instants')
plt.legend()
plt.grid()
plt.show()
```

Dans la figure 4 , elle représente la solution U (t, s) à Différents Instants



0.4 les criteres d'arret pour l'iteration d'adaptation

Les critères d'arrêt pour l'itération d'adaptation dans un problème de simulation numérique, comme celui que nous avons discuté, peuvent varier en fonction des objectifs spécifiques de l'étude. Voici quelques critères couramment utilisés :

1. Erreur L_2

Définition : Calcule l'erreur entre la solution actuelle et une solution de référence (ou une solution à un temps précédent) en utilisant la norme L_2.
Critère : Arrêter l'itération si l'erreurL_2 est inférieure à un seuil prédéfini, par exemple, ϵ .
2. Nombre de points de maillage

Définition : Suit le nombre total de points dans le maillage.
Critère : Arrêter l'itération si le nombre de points de maillage atteint un maximum prédéfini, ce qui limite la complexité du calcul.
3. Convergence des solutions

Définition : Évaluer si les solutions des itérations successives changent de manière significative.
Critère : Arrêter si la différence entre les solutions successives (en termes de normeL_2ou autre) est inférieure à un seuil.
4. Temps d'exécution

Définition : Surveiller le temps nécessaire pour effectuer les itérations.
Critère : Arrêter l'itération si un certain temps maximum d'exécution est atteint.

5. Stabilité numérique

Définition : Vérifier si la solution devient instable ou diverge.

Critère : Arrêter si des oscillations ou des comportements non physiques apparaissent dans la solution.
6. Critère mixte

Définition : Combiner plusieurs critères d'arrêt. Critère : Ne pas arrêter tant que l'erreurL_2 est supérieure à un seuil et que le

nombre de points de maillage n'a pas atteint un maximum.

que donne le maillage adaptatif avec controle de metrique stationnaire ?

Le maillage adaptatif avec contrôle de métrique stationnaire permet d'atteindre une convergence plus rapide et une meilleure précision en ajustant dynamiquement le maillage en fonction des caractéristiques de la solution. En fin de simulation, cela garantit que les résultats sont fiables et indépendants du maillage utilisé. Cette méthode est particulièrement utile dans les simulations complexes où des phénomènes variés peuvent se produire dans différentes régions de l'espace.