TP SNHPC : Résolution d'équation aux dérivées partielles

Boris Baudel - Ecole Normale Supérieure Paris-Saclay - ArteQ

Professeur : Fikri Hafid

Contents

1. Résolution analytique

1.1 Résolution analytique $u(x,0) = \sin(2\pi x)$

Nous cherchons à résoudre l'équation aux dérivées partielles suivante :

$$u_t(x,t) = u_{xx}(x,t)$$

avec les conditions aux limites et initiales suivantes :

$$u(0,t) = u(1,t) = 0, \quad u(x,0) = f_1(x) = \sin(2\pi x), \quad \text{pour } x \in]0,1[\text{ et } t > 0.$$

Nous utilisons la méthode de séparation des variables en supposant que $u(x,t) = X(x)T(t)$. En substituant dans l'EDP, nous avons :

$$X(x)T'(t) = X''(x)T(t)$$

En divisant les deux côtés par $X(x)T(t)$:

$$\frac{T'(t)}{T(t)} = \frac{X''(x)}{X(x)} = -\lambda$$

où λ est une constante de séparation. Nous obtenons deux équations ordinaires :

$$T'(t) + \lambda T(t) = 0$$

$$X''(x) + \lambda X(x) = 0$$

En appliquant les conditions aux limites $X(0) = X(1) = 0$, nous cherchons des solutions sous la forme trigonométrique. La solution générale de $X''(x) + \lambda X(x) = 0$ est :

$$X(x) = A\cos(\sqrt{\lambda}x) + B\sin(\sqrt{\lambda}x)$$

Avec les conditions aux limites, nous trouvons que $A = 0$ et $\lambda = n^2\pi^2$ pour n entier non nul. Pour $n = 2$:

$$\lambda = 4\pi^2$$

$$X(x) = B\sin(2\pi x)$$

Pour $T(t)$ avec $\lambda = 4\pi^2$, la solution est :

$$T(t) = Ce^{-4\pi^2 t}$$

La solution complète avec la condition initiale donnée est :

$$u(x,t) = C\sin(2\pi x)e^{-4\pi^2 t}$$

En substituant $u(x,0) = \sin(2\pi x)$, nous trouvons que $C = 1$.

La solution finale est :

$$u(x,t) = \sin(2\pi x)e^{-4\pi^2 t}$$

1.2 Solution analytique de l'EDP pour une condition initiale par morceaux

Nous cherchons à résoudre l'équation aux dérivées partielles :

$$u_t(x,t) = u_{xx}(x,t)$$

avec les conditions aux limites :

$$u(0,t) = u(1,t) = 0$$

et la condition initiale :

$$u(x,0) = f_2(x) = \begin{cases} 2x & \text{si } x \in \left]0, \frac{1}{2}\right] \\ 2(1-x) & \text{si } x \in \left]\frac{1}{2}, 1\right[\end{cases}$$

La fonction $f_2(x)$ est développée en série de Fourier en sinus, car la solution doit satisfaire les conditions de Dirichlet homogènes aux bords. La série de Fourier est de la forme :

$$f_2(x) = \sum_{n=1}^{\infty} b_n \sin(n\pi x)$$

où les coefficients b_n sont donnés par :

$$b_n = 2\int_0^1 f_2(x)\sin(n\pi x)\,dx$$

Pour notre fonction par morceaux, les coefficients deviennent :

$$b_n = 2\left(\int_0^{1/2} 2x\sin(n\pi x)\,dx + \int_{1/2}^1 2(1-x)\sin(n\pi x)\,dx\right)$$

Les calculs des intégrales peuvent être réalisés par intégration par parties.

$$\begin{aligned}\int_0^{1/2} 2x\sin(n\pi x)\,dx &= \left[-\frac{2x\cos(n\pi x)}{n\pi}\right]_0^{1/2} + \int_0^{1/2}\frac{2\cos(n\pi x)}{n\pi}\,dx \\ &= -\frac{\cos(\frac{n\pi}{2})}{n\pi} + \left[\frac{2\sin(n\pi x)}{n^2\pi^2}\right]_0^{1/2} \\ &= -\frac{\cos(\frac{n\pi}{2})}{n\pi} + \frac{2\sin(\frac{n\pi}{2})}{n^2\pi^2}\end{aligned}$$

Il vient :

$$bn = 2\left[\left[-\frac{2x}{n\pi}\cos(n\pi x)\right]_0^{1/2} + \left[\frac{2}{(n\pi)^2}\sin(n\pi x)\right]_0^{1/2} + \left[-\frac{2(1-x)}{(n\pi)^2}\cos(n\pi x)\right]_{1/2}^{1} + \left[-\frac{2}{(n\pi)^2}\sin(n\pi x)\right]_{1/2}^{1}\right]$$

En simplifiant les termes, on a :

$$bn = 2\left[-\frac{1}{n\pi}\cos\left(\frac{n\pi}{2}\right) + \frac{2}{(n\pi)^2}\sin\left(\frac{n\pi}{2}\right) + \frac{1}{n\pi}\cos\left(\frac{n\pi}{2}\right) + \frac{2}{(n\pi)^2}\sin\left(\frac{n\pi}{2}\right)\right]$$

$$bn = \frac{8\sin\left(\frac{\pi n}{2}\right))}{\pi^2 n^2}.$$

Une fois ces coefficients déterminés, la solution complète de l'EDP est :

La solution finale est :

$$u(x,t) = \sum_{n=1}^{\infty} b_n \sin(n\pi x)e^{-n^2\pi^2 t}$$

Chaque terme de cette série représente un mode de vibration de la barre thermique, et chaque mode décroît exponentiellement au cours du temps.

2. Solution numérique : Schéma explicite

2.1 Schéma explicite pour l'équation de la chaleur

- Divisons l'intervalle spatial $[0,1]$ en $N+1$ points avec un pas $h = \frac{1}{N+1}$, et notons $x_i = ih$ pour $i = 0, 1, \ldots, N+1$.
- Le temps est discrétisé avec un pas $\Delta t = k$, et $t^n = nk$ pour $n = 0, 1, 2, \ldots$.

Approximons les dérivées comme suit :

- Dérivée temporelle : $\frac{\partial u}{\partial t} \approx \frac{u_i^{n+1} - u_i^n}{k}$,
- Dérivée seconde spatiale : $\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{h^2}$.

En insérant ces approximations dans l'équation $u_t = u_{xx}$, on obtient :

$$\frac{u_i^{n+1} - u_i^n}{k} = \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{h^2}.$$

Cela donne le schéma explicite :

$$u_i^{n+1} = u_i^n + \lambda \left(u_{i-1}^n - 2u_i^n + u_{i+1}^n \right),$$

où $\lambda = \frac{k}{h^2}$.

Pour garantir la stabilité du schéma explicite, la condition CFL (Courant–Friedrichs–Lewy) doit être satisfaite :

$$\lambda = \frac{k}{h^2} \leq \frac{1}{2}.$$

2.2 Formulation Matricielle du Schéma Explicite pour l'Équation de la Chaleur

Nous considérons l'équation aux dérivées partielles de la forme :

$$u_t(x,t) = u_{xx}(x,t)$$

avec des conditions aux limites de Dirichlet homogènes. Pour une résolution numérique par le schéma en différences finies, nous adoptons un schéma explicite en temps et centré en espace, d'ordre 1 en temps et d'ordre 2 en espace. Le domaine spatial est discrétisé en points $x_i = ih$ pour $i = 0, 1, \ldots, M$ et le temps est discrétisé en instants $t^n = nk$ pour $n = 0, 1, \ldots, N$. La mise à jour temporelle s'exprime par :

$$u_i^{n+1} = u_i^n + \lambda(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

où $\lambda = \frac{k}{h^2}$ est le paramètre de stabilité. Pour formuler ce schéma en termes matriciels, nous définissons le vecteur des températures à l'instant t^n par :

$$\mathbf{u}^n = \begin{bmatrix} u_1^n \\ u_2^n \\ \vdots \\ u_{M-1}^n \end{bmatrix}$$

La matrice tridiagonale A, représentant l'opérateur discret Laplacien avec conditions aux limites de Dirichlet, est :

$$A = \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -2 \end{bmatrix}$$

La formulation matricielle complète pour le schéma explicite est donc :

$$\mathbf{u}^{n+1} = (I + \lambda A)\mathbf{u}^n$$

2.3 Implémentation sur Python

Pour faire l'implémentation sur python nous allons utiliser les résultats et le schéma explicite que nous avons fait précédemment. Voici le code de cette implémenation :

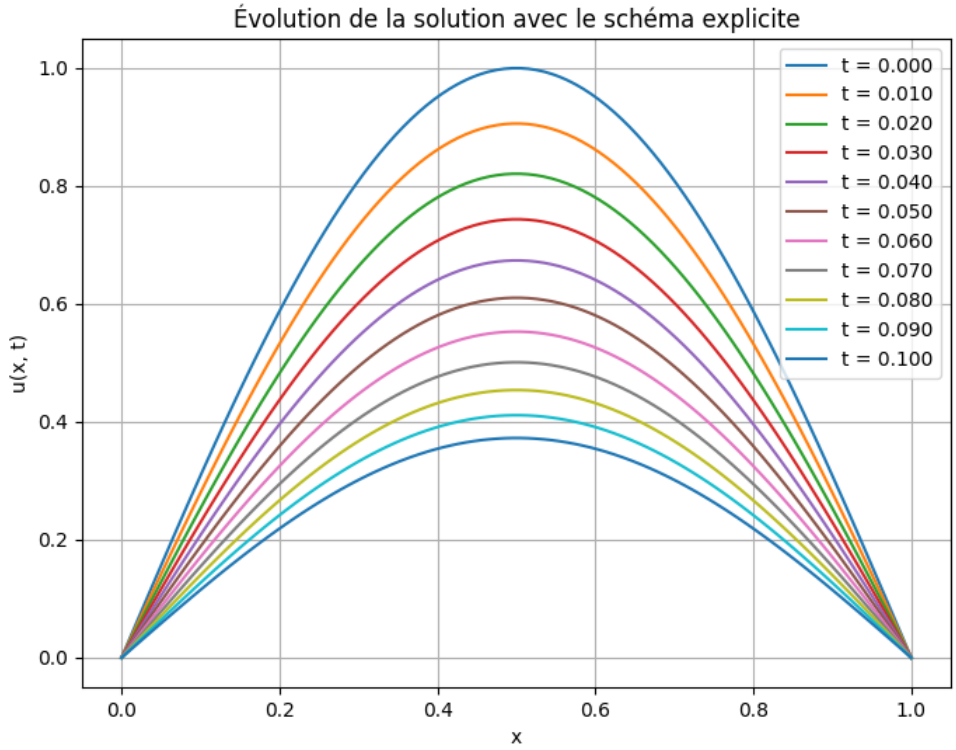


Figure 1: Solution schéma explicite

```python
L = 1.0
T = 0.1
N = 100
M = 5000
h = L / (N + 1)
k = T / M
lambda_ = k / h**2
if lambda_ > 0.5:
    print("CFL non respectee (lambda > 0.5).")
else:
    print("CFL respectee (lambda <= 0.5).")
diagonal = (1 - 2 * lambda_) * np.ones(N)
off_diagonal = lambda_ * np.ones(N - 1)
A = np.diag(diagonal) + np.diag(off_diagonal, k=1) + np.diag(off_diagonal, k
    =-1)
# Initialisation
x = np.linspace(0, L, N + 2)
t = np.linspace(0, T, M + 1)
u = np.zeros((M + 1, N + 2))
f = lambda x: np.sin(np.pi * x)  # Exemple : u(x,0) = sin(pi*x)
u[0, 1:-1] = f(x[1:-1])
# Schema explicite
for n in range(0, M):
    u[n + 1, 1:-1] = A @ u[n, 1:-1]# u[n+1, 0] = 0 conditions limites
plt.figure(figsize=(8, 6))
for i in range(0, M + 1, M // 10):
    plt.plot(x, u[i, :], label=f"t = {t[i]:.3f}")
plt.title("Evolution de la solution avec le schema explicite")
plt.xlabel("x")
plt.ylabel("u(x, t)")
plt.legend()
plt.grid()
plt.show()
```

3. Schéma Implicite pour l'Équation de la Chaleur

Nous cherchons à résoudre l'équation de la chaleur avec un schéma implicite, qui est stable sans condition sur le pas de temps.

3.1 Formulation du Schéma implicite

Le schéma d'Euler implicite en temps combiné avec une approximation centrée pour la dérivée seconde en espace est défini par la relation suivante :

$$\frac{u_i^{n+1} - u_i^n}{k} = \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{h^2}$$

où h est le pas spatial et k le pas temporel.

Cette équation peut être réarrangée en :

$$-\lambda u_{i-1}^{n+1} + (1 + 2\lambda)u_i^{n+1} - \lambda u_{i+1}^{n+1} = u_i^n$$

où $\lambda = \frac{k}{h^2}$.

3.2 Formulation Matricielle

Le système peut être représenté sous forme matricielle pour faciliter le calcul numérique. Soit $\mathbf{u}^n$ le vecteur des températures à l'instant t^n et $\mathbf{u}^{n+1}$ à l'instant t^{n+1}, alors :

$$A\mathbf{u}^{n+1} = \mathbf{u}^n$$

où A est une matrice tridiagonale définie comme suit :

$$A = \begin{bmatrix} 1+2\lambda & -\lambda & 0 & \cdots & 0 \\ -\lambda & 1+2\lambda & -\lambda & \cdots & 0 \\ 0 & -\lambda & 1+2\lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1+2\lambda \end{bmatrix}$$

La résolution de ce système à chaque pas de temps permet de mettre à jour le vecteur de température selon :

$$\mathbf{u}^{n+1} = A^{-1}\mathbf{u}^n$$

3.3 Implémentation sur Python

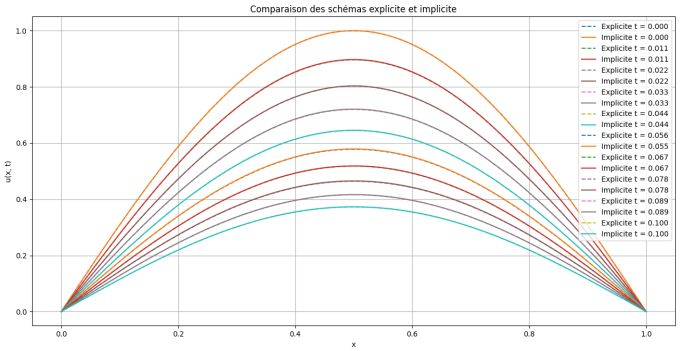


Figure 2: Solution schéma implicite

Le résultat de l'implémentation est visible dans la figure 2, nous avons un résultat similaire a celui de la première figure. Nous pouvons dresser le tableau suivant qui compare les deux méthodes.

Aspect	**Méthode explicite**	**Méthode implicite**
Formulation	$\mathbf{U}^{n+1} = \mathbf{A}\mathbf{U}^n$	$\mathbf{A}\mathbf{U}^{n+1} = \mathbf{U}^n$
Stabilité	Condition de stabilité stricte ($\lambda \leq 0.5$)	Stable pour tout λ (inconditionnelle)
Résolution	Directe, sans résolution de système	Résolution d'un système linéaire à chaque pas
Coût de calcul	Bas pour des petits pas temporels k	Plus élevé à chaque pas de temps
Précision pour T long	Faible (petits k)	Bonne (possibilité de grands k)
Applications	Cas où k est petit (problèmes rapides)	Cas où k est grand (temps longs)

Table 1: Comparaison des méthodes explicite et implicite pour l'équation de la chaleur.

```python
from scipy.linalg import solve_banded
L = 1.0
T = 0.1
N = 100
M = 500
h = L / (N + 1)
k = T / M
lambda_ = k / h**2
if lambda_ > 0.5:
    print("lambda > 0.5,schema  stable, moins precis")
else:
    print("Schema implicite stable.")
diagonal = (1 + 2 * lambda_) * np.ones(N)
off_diagonal = -lambda_ * np.ones(N - 1)
A_banded = np.zeros((3, N))  # Matrice tridiagonale
A_banded[0, 1:] = off_diagonal  # Diagonale superieure
A_banded[1, :] = diagonal       # Diagonale principale
A_banded[2, :-1] = off_diagonal # Diagonale inferieure
x = np.linspace(0, L, N + 2)
t = np.linspace(0, T, M + 1)
u = np.zeros((M + 1, N + 2))
f = lambda x: np.sin(np.pi * x)  # Exemple : u(x,0) = sin(pi*x)
u[0, 1:-1] = f(x[1:-1])
# Schema implicite
for n in range(0, M):
    # Resolution du systeme lineaire A * U^{n+1} = U^n
    u_next = solve_banded((1, 1), A_banded, u[n, 1:-1])
    u[n + 1, 1:-1] = u_next  # Mise a jour de la solution
plt.figure(figsize=(8, 6))
for i in range(0, M + 1, M // 10):
    plt.plot(x, u[i, :], label=f"t = {t[i]:.3f}")
plt.title("evolution de la solution avec le schema implicite (matricielle)")
plt.xlabel("x")
plt.ylabel("u(x, t)")
plt.legend()
plt.grid()
plt.show()
```

4. Solution numérique : Schéma de Crank Nicholson

4.1 Schéma de Crank Nicholson : Définition

L'équation à résoudre est :

$$u_t(x,t) = u_{xx}(x,t), \quad x \in]0,1[,\ t > 0,$$

avec les conditions :

$$u(0,t) = 0, \quad u(1,t) = 0 \quad \text{pour tout } t > 0,$$

et une condition initiale :

$$u(x,0) = f(x).$$

Nous allons appliquer le schéma de Crank-Nicholson, qui combine les schémas explicite et implicite pour obtenir un schéma numérique stable et précis. Le schéma de Crank-Nicholson s'obtient en faisant la moyenne des termes explicites et implicites pour les dérivées spatiales à l'instant t^n et t^{n+1}. Pour le terme explicite (à l'instant t^n) :

$$u_{xx} \approx \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{h^2}.$$

Pour le terme implicite (à l'instant t^{n+1}) :

$$u_{xx} \approx \frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{h^2}.$$

4.1.1 Moyennage des termes explicite et implicite

Le schéma de Crank-Nicholson pour $u_t = u_{xx}$ s'écrit :

$$\frac{u_i^{n+1} - u_i^n}{k} = \frac{1}{2}\left[\frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{h^2} + \frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{h^2}\right].$$

En réarrangeant cette équation, nous obtenons :

$$-\lambda u_{i-1}^{n+1} + (1+2\lambda)u_i^{n+1} - \lambda u_{i+1}^{n+1} = \lambda u_{i-1}^n + (1-2\lambda)u_i^n + \lambda u_{i+1}^n,$$

où $\lambda = \frac{k}{2h^2}$.

4.2 Ordre : Crank Nicholson

- Le terme u_{xx} est approximé par une différence centrale spatiale :

$$u_{xx} \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}.$$

- Cette approximation introduit une erreur de troncature de l'ordre $O(h^2)$, ce qui donne un schéma d'ordre **2 en espace**.

- La dérivée temporelle u_t est approximée par une différence centrée temporelle via une moyenne des termes explicites (au temps t^n) et implicites (au temps t^{n+1}) :

$$\frac{u_i^{n+1} - u_i^n}{k} \approx \frac{1}{2}\left(u_{xx}^n + u_{xx}^{n+1}\right).$$

- Cette approximation introduit une erreur de troncature de l'ordre $O(k^2)$, ce qui donne un schéma d'ordre **2 en temps**. Le schéma de Crank-Nicholson est d'ordre **2 en espace** ($O(h^2)$) Et d'ordre **2 en temps** ($O(k^2)$).

4.3 Formulation matricielle : Crank Nicholson

Sous forme matricielle, cela devient :

$$\mathbf{A}\mathbf{U}^{n+1} = \mathbf{B}\mathbf{U}^n,$$

avec :

Matrice tridiagonale pour les termes implicites (t^{n+1}) :

$$\mathbf{A} = \begin{bmatrix} 1+2\lambda & -\lambda & 0 & \cdots & 0 \\ -\lambda & 1+2\lambda & -\lambda & \cdots & 0 \\ 0 & -\lambda & 1+2\lambda & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & -\lambda \\ 0 & 0 & \cdots & -\lambda & 1+2\lambda \end{bmatrix}.$$

Matrice tridiagonale pour les termes explicites (t^n) :

$$\mathbf{B} = \begin{bmatrix} 1-2\lambda & \lambda & 0 & \cdots & 0 \\ \lambda & 1-2\lambda & \lambda & \cdots & 0 \\ 0 & \lambda & 1-2\lambda & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \lambda \\ 0 & 0 & \cdots & \lambda & 1-2\lambda \end{bmatrix}.$$

Les vecteurs $\mathbf{U}^n$ et $\mathbf{U}^{n+1}$ représentent les solutions aux instants t^n et t^{n+1}, respectivement :

$$\mathbf{U}^n = [u_1^n, u_2^n, \ldots, u_N^n]^T, \quad \mathbf{U}^{n+1} = [u_1^{n+1}, u_2^{n+1}, \ldots, u_N^{n+1}]^T.$$

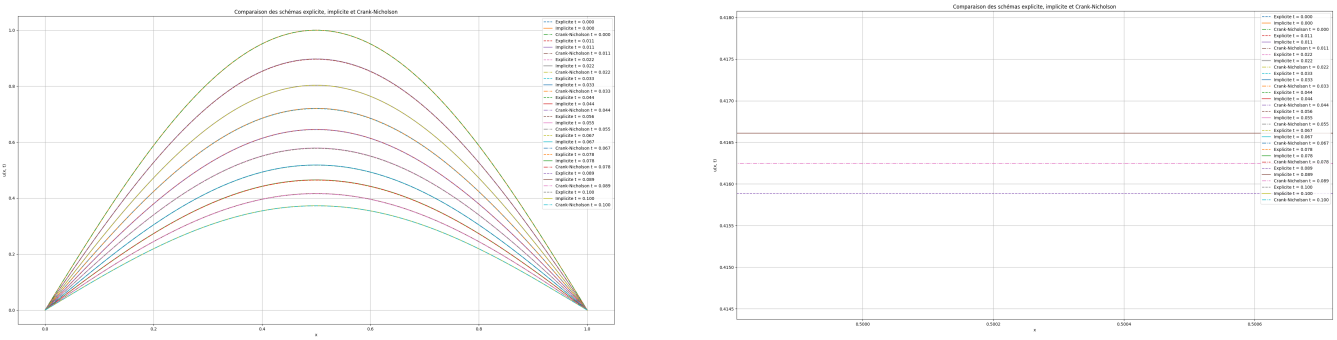
```
L = 1.0
T = 0.1
N = 100
M_explicit = 5000
M_implicit = 500
M_crank = 500
h = L / (N + 1)  # Pas spatial
x = np.linspace(0, L, N + 2)  # Points spatiaux (inclut les bords)
f = lambda x: np.sin(np.pi * x)  # Exemple : u(x,0) = sin(pi*x)
# Sch ma explicite
# ----------------------------------------
k_explicit = T / M_explicit  # Pas temporel explicite
lambda_explicit = k_explicit / h**2  # Constante explicite
if lambda_explicit > 0.5:
    print(" CFL non respect e (lambda > 0.5).")
else:
    print("CFL respect e pour le sch ma explicite (lambda <= 0.5).")
diagonal_explicit = (1 - 2 * lambda_explicit) * np.ones(N)
off_diagonal_explicit = lambda_explicit * np.ones(N - 1)
A_explicit = np.diag(diagonal_explicit) + np.diag(off_diagonal_explicit, k=1) +
    np.diag(off_diagonal_explicit, k=-1)
u_explicit = np.zeros((M_explicit + 1, N + 2))
u_explicit[0, 1:-1] = f(x[1:-1])  # Condition initiale
for n in range(0, M_explicit):
    u_explicit[n + 1, 1:-1] = A_explicit @ u_explicit[n, 1:-1]
# ----------------------------------------
# Sch ma implicite
# ----------------------------------------
k_implicit = T / M_implicit  # Pas temporel implicite
lambda_implicit = k_implicit / h**2  # Constante implicite
if lambda_implicit > 0.5:
    print("lambda > 0.5.")
else:
    print("Sch ma implicite stable.")
diagonal_implicit = (1 + 2 * lambda_implicit) * np.ones(N)
off_diagonal_implicit = -lambda_implicit * np.ones(N - 1)
A_banded_implicit = np.zeros((3, N))
A_banded_implicit[0, 1:] = off_diagonal_implicit
A_banded_implicit[1, :] = diagonal_implicit
A_banded_implicit[2, :-1] = off_diagonal_implicit
u_implicit = np.zeros((M_implicit + 1, N + 2))
u_implicit[0, 1:-1] = f(x[1:-1])  # Condition initiale
for n in range(0, M_implicit):
    u_next = solve_banded((1, 1), A_banded_implicit, u_implicit[n, 1:-1])
    u_implicit[n + 1, 1:-1] = u_next
# ----------------------------------------
# Schema de Crank-Nicholson
# ----------------------------------------
k_crank = T / M_crank  # Pas temporel pour Crank-Nicholson
lambda_crank = k_crank / (2 * h**2)
diagonal_crank_A = (1 + 2 * lambda_crank) * np.ones(N)
off_diagonal_crank_A = -lambda_crank * np.ones(N - 1)
A_banded_crank = np.zeros((3, N))
A_banded_crank[0, 1:] = off_diagonal_crank_A
A_banded_crank[1, :] = diagonal_crank_A
A_banded_crank[2, :-1] = off_diagonal_crank_A
diagonal_crank_B = (1 - 2 * lambda_crank) * np.ones(N)
off_diagonal_crank_B = lambda_crank * np.ones(N - 1)
B_crank = np.diag(diagonal_crank_B) + np.diag(off_diagonal_crank_B, k=1) + np.
    diag(off_diagonal_crank_B, k=-1)
u_crank = np.zeros((M_crank + 1, N + 2))
u_crank[0, 1:-1] = f(x[1:-1])  # Condition initiale
for n in range(0, M_crank):
    rhs = B_crank @ u_crank[n, 1:-1]  # Second membre B * U^n
    u_next = solve_banded((1, 1), A_banded_crank, rhs)
    u_crank[n + 1, 1:-1] = u_next
```

```
# ----------------------------------------
# Visualisation compareE
# ----------------------------------------
plt.figure(figsize=(10, 6))
time_indices_explicit = np.linspace(0, M_explicit, 10, dtype=int)
time_indices_implicit = np.linspace(0, M_implicit, 10, dtype=int)
time_indices_crank = np.linspace(0, M_crank, 10, dtype=int)
for idx_e, idx_i, idx_c in zip(time_indices_explicit, time_indices_implicit,
    time_indices_crank):
    plt.plot(x, u_explicit[idx_e, :], '--', label=f"Explicite t = {idx_e *
    k_explicit:.3f}")
    plt.plot(x, u_implicit[idx_i, :], '-', label=f"Implicite t = {idx_i *
    k_implicit:.3f}")
    plt.plot(x, u_crank[idx_c, :], '-.', label=f"Crank-Nicholson t = {idx_c *
    k_crank:.3f}")
plt.title("Comparaison des sch mas explicite, implicite et Crank-Nicholson")
plt.xlabel("x")
plt.ylabel("u(x, t)")
plt.legend()
plt.grid()
plt.show()
```

‘

5. Comparaison des méthodes

Les avantages et inconvénients de toutes les méthodes sont données ci-dessous :



(a) Solutions pour $f_1(x)$.

(b) Solutions pour $f_1(x)$.

Figure 3: Comparaison des schémas pour toutes les méthodes

Schéma	Ordre en temps	Ordre en espace
Explicite	1 ($O(k)$)	2 ($O(h^2)$)
Implicite	1 ($O(k)$)	2 ($O(h^2)$)
Crank-Nicholson	2 ($O(k^2)$)	2 ($O(h^2)$)

Table 2: Ordre de précision des schémas numériques.

Avantage Crank-Nicholson : Il offre une meilleure précision temporelle ($O(k^2)$) par rapport aux deux autres schémas.
Avantage Crank-Nicholson : Comme le schéma implicite, il est inconditionnellement stable, mais avec une meilleure précision que le schéma implicite.

Schéma	Stabilité
Explicite	Conditionnellement stable ($k/h^2 \leq 0.5$)
Implicite	Inconditionnellement stable
Crank-Nicholson	Inconditionnellement stable

Table 3: Stabilité des schémas numériques.

Schéma	Précision dans le temps
Explicite	Faible pour de grands T (petits k nécessaires pour stabilité).
Implicite	Bonne pour de grands T, mais erreurs de diffusion possibles.
Crank-Nicholson	Excellente pour de grands T, sans erreurs de phase.

Table 4: Précision dans les simulations à long terme.

Avantage Crank-Nicholson : Il offre une meilleure précision que le schéma implicite, en particulier sur le long terme, sans nécessiter de petits pas temporels comme le schéma explicite.

Schéma	Coût
Explicite	Faible (pas de système linéaire à résoudre).
Implicite	Modéré (système linéaire à chaque pas).
Crank-Nicholson	Modéré (même coût que le schéma implicite).

Table 5: Coût de calcul des schémas numériques.

Avantage Crank-Nicholson : Le coût est similaire à celui du schéma implicite, mais avec une meilleure précision.

Schéma	Complexité
Explicite	Facile
Implicite	Moyenne (nécessite un solveur pour un système linéaire).
Crank-Nicholson	Moyenne (nécessite aussi un solveur pour un système linéaire).

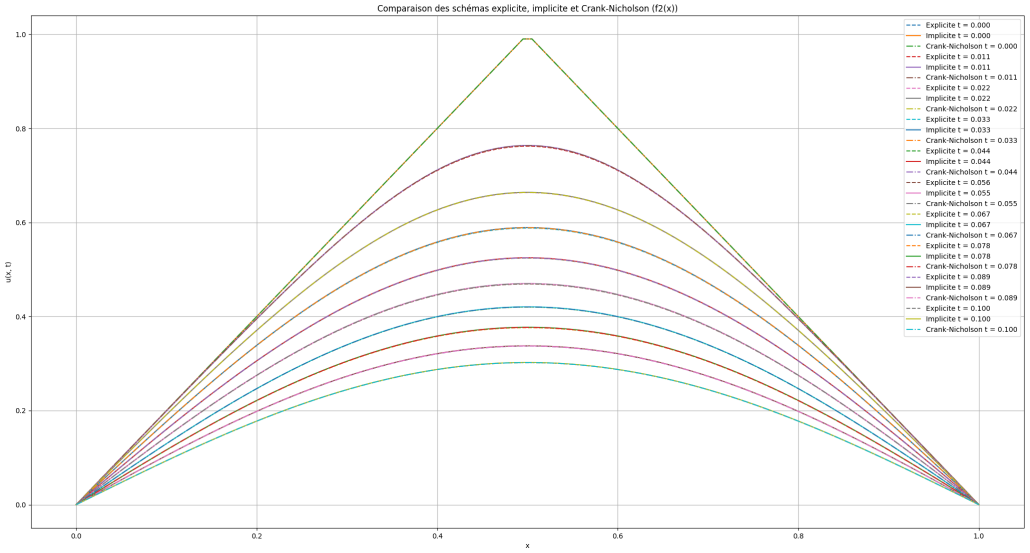
Table 6: Complexité d'implémentation.

Avantage Crank-Nicholson : Légèrement plus complexe que le schéma explicite, mais pas plus compliqué que le schéma implicite.

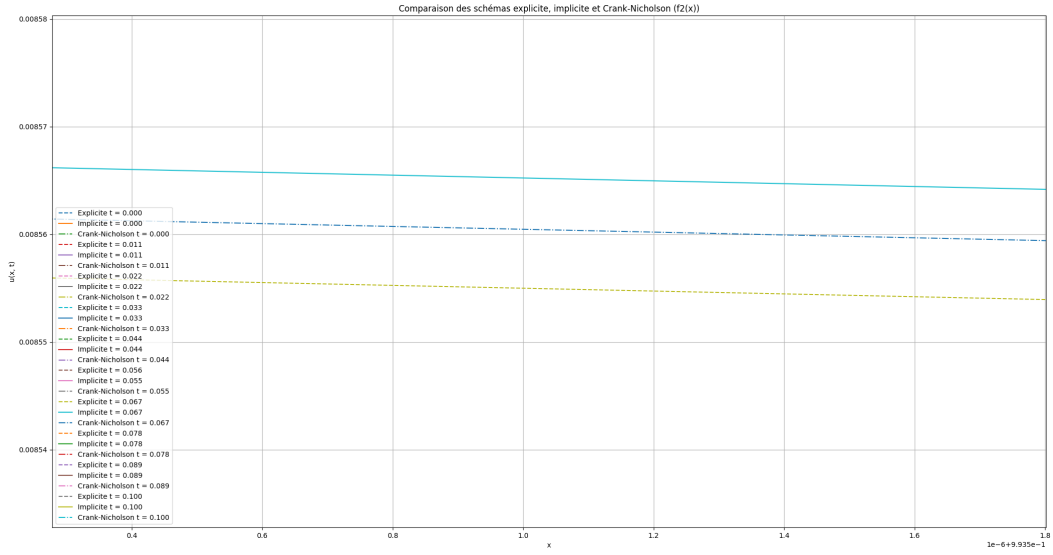
Schéma	Condition CFL
Explicite	$k/h^2 \leq 0.5$ pour stabilité.
Implicite	Pas de contrainte (stable pour tout k).
Crank-Nicholson	Pas de contrainte (stable pour tout k).

Table 7: Condition CFL des schémas numériques.

Avantage Crank-Nicholson : Comme le schéma implicite, il n'est pas limité par la condition CFL.



(a) Solutions pour $f_2(x)$.



(b) Solutions pour $f_2(x)$.

Figure 4: Comparaison des schémas pour les méthodes