# rapport-modele

November 12, 2025

# 1 Simulation numérique de l'advection bidimensionnelle

---

## 1.1 Projet de Méthodes Numériques – Rapport de validation

### 1.1.1 Étudiant : *[MEZING THERESE CLAUDIA]*

### 1.1.2 Encadrant : *[Pierre Archambeau]*

### 1.1.3 Date : *[12/11/2025]*

---

**Université / École :** *[Université de Liège]*
**Filière :** *[Génie civil des constructions]*
**Année universitaire :** *2025 – 2026*

---

## 1.2 1. Introduction

Le transport d'une grandeur scalaire (comme la température, la concentration ou un polluant) dans un fluide en mouvement est un phénomène fondamental en mécanique des fluides et en physique numérique. Ce processus est modélisé par l'équation d'advection, qui décrit comment une quantité se déplace sous l'effet d'un champ de vitesse donné.

Dans ce rapport, nous étudions la résolution numérique de l'équation d'advection bidimensionnelle sur un domaine périodique. L'objectif est double :mettre en œuvre un schéma numérique de type volumes finis pour simuler le déplacement d'une distribution gaussienne initiale;et évaluer la convergence et la précision du schéma à l'aide d'une solution analytique de référence.

Pour ce faire, nous utiliserons :une vitesse uniforme connue,une condition initiale gaussienne centrée,et une intégration temporelle explicite de type Runge–Kutta d'ordre 2 (Heun).

La comparaison entre la solution numérique et la solution analytique permettra de calculer l'erreur en norme  2 et d'estimer l'ordre de convergence du schéma.

## 1.3 2. Méthodologie

### 1.3.1 2.1. Modèle mathématique

Équation considérée :

$$\frac{\partial q}{\partial t} + u\frac{\partial q}{\partial x} + v\frac{\partial q}{\partial y} = 0$$

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from fv2d import Grid2D, AdvectionSolver2D
from time_integrators import rk2_heun_step
from initial_conditions import gaussian

def uniform_velocity(u0=1.0, v0=0.0):
    def vel(X, Y):
        return np.full_like(X, u0), np.full_like(X, v0)
    return vel

grid = Grid2D(128, 128, 1.0, 1.0)
vel = uniform_velocity(1.0, 0.0)
solver = AdvectionSolver2D(grid, vel, recon='linear')

q0 = gaussian(grid, x0=0.25, y0=0.5, sigma=0.05)
q = q0.copy()

dt = solver.cfl_dt(0.4)
t, tmax = 0.0, 1.0

while t < tmax - 1e-12:
    if t + dt > tmax:
        dt = tmax - t
    q = rk2_heun_step(solver, q, dt)
    t += dt

plt.figure(figsize=(6,6))
plt.imshow(q, origin='lower', cmap='viridis', extent=[0,1,0,1])
plt.colorbar(label='Concentration')
plt.title(f'Champ advecté à t = {t:.2f}')
plt.xlabel('x'); plt.ylabel('y')
plt.show()
```

### 1.3.2 2.2. Méthode numerique

La résolution de l'équation d'advection bidimensionnelle repose sur la méthode des volumes finis, combinée à une intégration temporelle de type Runge–Kutta d'ordre 2 (Heun). Cette approche permet d'assurer la conservation de la quantité advectée et une précision d'ordre 2 en espace et en temps.

**2.2.1. Equation d'advection** L'équation d'advection 2D s'écrit sous la forme conservative suivante :

+ ( ) + ( ) = 0 t q

- x (uq)
- y (vq)

=0

où :

( , , ) q(x,y,t) est la grandeur transportée,

, u,v sont les composantes du champ de vitesse = ( , ) u=(u,v)

**2.2.2. Discrétisation spatiale** Le domaine est discrétisé en × N x

×N y

cellules régulières, de tailles Δ Δx et Δ Δy. En intégrant l'équation sur la cellule ( , ) (i,j), on obtient :

, = − 1 Δ ( + 1 2 , − − 1 2 , ) − 1 Δ ( , + 1 2 − , − 1 2 ) dt dq i,j

=− Δx 1

(F i+ 2 1

,j

−F i− 2 1

,j

)− Δy 1

(G i,j+ 2 1

−G i,j− 2 1

)

avec :

+ 1 2 , = + 1 2 , + 1 2 , , , + 1 2 = , + 1 2 , + 1 2 F i+ 2 1

,j

=u i+ 2 1

,j

q i+ 2 1

,j

,G i,j+ 2 1

=v i,j+ 2 1

q i,j+ 2 1

Les valeurs aux interfaces + 1 2 , q i+ 2 1

,j

sont obtenues par reconstruction linéaire :

+ 1 2 , = , + 1 2 , Δ q i+ 2 1

,j

=q i,j

- 2 1

i,j

Δx

où , i,j

représente la pente locale limitée pour éviter les oscillations.

[ ]:
```
import numpy as np

# Exemple de grille régulière
nx, ny = 64, 64
Lx, Ly = 1.0, 1.0
dx, dy = Lx / nx, Ly / ny
x = np.linspace(dx/2, Lx - dx/2, nx)
y = np.linspace(dy/2, Ly - dy/2, ny)
X, Y = np.meshgrid(x, y, indexing='xy')

# Vitesse uniforme
u, v = 1.0, 0.0

# Fonction gaussienne initiale
def gaussian(x, y, x0=0.25, y0=0.5, sigma=0.05):
    return np.exp(-((x-x0)**2 + (y-y0)**2)/(2*sigma**2))

q = gaussian(X, Y)

# Exemple : calcul du flux dans la direction x
```

```
F = u * q
```

**2.2.3. Intégration temporelle-Schéma de Heun(RK2)** L'évolution temporelle est assurée par le schéma de Heun, un Runge–Kutta explicite d'ordre 2 :

{ ( 1 ) = + Δ ( )

+ 1 = 1 2 ( + ( 1 ) + Δ ( ( 1 ) ) ) { q (1) =q n +ΔtL(q n ) q n+1 = 2 1

(q n +q (1) +ΔtL(q (1) ))

où ( ) L(q) est l'opérateur spatial représentant les flux d'advection.

[ ]:
```
def heun_step(q, L, dt):
    """
    Effectue une étape de Runge-Kutta d'ordre 2 (Heun).
    L : fonction qui calcule les flux d'advection (opérateur spatial)
    """
    q1 = q + dt * L(q)          # prédiction (Euler)
    return 0.5 * (q + q1 + dt * L(q1))  # correction moyenne
```

**2.2.4. Condition de stabilité(CFL)** Le pas de temps doit respecter la condition de Courant–Friedrichs–Lewy (CFL) :

## 2

max ( Δ Δ + Δ Δ ) max C=max( Δx u Δt

- Δy v Δt

) C max

Dans la pratique, on choisit max = 0.4 C max

=0.4.

[ ]:
```
def compute_dt(u, v, dx, dy, cfl=0.4):
    """Calcule le pas de temps à partir du critère CFL."""
    return cfl / (abs(u)/dx + abs(v)/dy)

dt = compute_dt(u, v, dx, dy)
print(f"Pas de temps choisi : dt = {dt:.5f}")
```

**2.2.5. Conditions aux limites** Le domaine est périodique : la valeur quittant le bord droit réapparaît à gauche, et inversement.

( + , , ) = ( , , ) q(x+L x

,y,t)=q(x,y,t)

Cette propriété simplifie la comparaison avec la solution analytique, puisqu'après un temps =
/ T=L x

/u, la forme initiale retrouve sa position d'origine.

### 2.0.1 2.3. Etude de convergence et validation

L'objectif de cette partie est de vérifier la précision du schéma numérique en comparant la solution numérique obtenue à la solution analytique pour un champ de vitesse uniforme.

**2.3.1. Solution analytique** Pour un champ de vitesse constant ( , ) (u,v), la solution de l'équation d'advection est simplement un déplacement du profil initial :

( , , ) = 0 ( − , − ) q(x,y,t)=q 0

(x−ut,y−vt)

Ainsi, après un temps t, la distribution initiale 0 q 0

est simplement translatée de ( , ) (ut,vt).

**2.3.2. Norme d'erreur L2** L'erreur entre la solution numérique num q num

et la solution de référence ref q ref

est mesurée à l'aide de la norme 2

:

2 = , ( , num − , ref ) 2 Δ Δ E L 2

= i,j

(q i,j num

−q i,j ref

) 2 ΔxΔy

**2.3.3. Calcul de l'ordre de convergence** L'ordre de convergence observé est déterminé entre deux maillages successifs :

## 3

log ( 2 ( 1 ) / 2 ( 2 ) ) log ( 2 / 1 ) p= log(N 2

/N 1

) log(E L 2

(N 1

)/E L 2

(N 2

))

où 1 , 2 N 1

,N 2

sont les résolutions spatiales.

[ ]:
```
import numpy as np
import math
from fv2d import Grid2D, AdvectionSolver2D
from time_integrators import rk2_heun_step
from initial_conditions import gaussian

def uniform_velocity(u0=1.0, v0=0.0):
    def vel(X, Y):
        return np.full_like(X, u0), np.full_like(X, v0)
    return vel

def l2_error(q_num, q_ref, grid):
    diff = q_num - q_ref
    return math.sqrt(np.sum(diff**2) * grid.dx * grid.dy)

def reference_solution(grid, q0, vel, final_time):
    u = float(vel(grid.X, grid.Y)[0][0])
    v = float(vel(grid.X, grid.Y)[1][0])
    Xs = (grid.X - u*final_time) % grid.Lx
    Ys = (grid.Y - v*final_time) % grid.Ly
    xi = (Xs / grid.dx - 0.5).astype(int) % grid.nx
    yi = (Ys / grid.dy - 0.5).astype(int) % grid.ny
    return q0[yi, xi]

def run_convergence():
    ns = [32, 64, 128, 256]
    errors = []
    final_time = 1.0

    for n in ns:
        grid = Grid2D(n, n, 1.0, 1.0)
        vel = uniform_velocity(1.0, 0.0)
        solver = AdvectionSolver2D(grid, vel, recon='linear')
        q0 = gaussian(grid, x0=0.25, y0=0.5, sigma=0.05)
        q = q0.copy()

        dt = solver.cfl_dt(0.4)
        t = 0.0
```

```
        while t < final_time - 1e-12:
            if t + dt > final_time:
                dt = final_time - t
            q = rk2_heun_step(solver, q, dt)
            t += dt

        qref = reference_solution(grid, q0, vel, final_time)
        err = l2_error(q, qref, grid)
        errors.append(err)
        print(f"n = {n:3d}, L2 error = {err:.3e}")

    for i in range(1, len(ns)):
        order = np.log(errors[i-1]/errors[i]) / np.log(ns[i]/ns[i-1])
        print(f"Order between {ns[i-1]} and {ns[i]}: {order:.2f}")

run_convergence()
```

**2.3.4. Résultats et interprétation** Les résultats typiques montrent une erreur décroissante lorsque la résolution augmente, avec un ordre de convergence proche de 2, confirmant :

la cohérence du schéma de reconstruction linéaire,

et la précision d'ordre 2 en temps du schéma de Heun.

[ ]:
```
import matplotlib.pyplot as plt

ns = np.array([32, 64, 128, 256])
errors = np.array([1.2e-2, 3.0e-3, 7.5e-4, 1.8e-4])  # Exemple de valeurs
orders = np.log(errors[:-1]/errors[1:]) / np.log(ns[1:]/ns[:-1])

plt.loglog(ns, errors, 'o-', label='Erreur L2')
plt.title("Convergence du schéma d'advection 2D")
plt.xlabel("Résolution N")
plt.ylabel("Erreur L2")
plt.grid(True, which="both")
plt.legend()
plt.show()

for i in range(len(orders)):
    print(f"Entre {ns[i]} et {ns[i+1]} : ordre = {orders[i]:.2f}")
```

## 3.1 3. Résultats et discussion

Dans cette section, nous présentons les résultats de la simulation numérique de l'advection bidimensionnelle d'une bosse gaussienne, ainsi que l'analyse quantitative de la convergence du schéma utilisé.

### 3.1.1 3.1. Evolution temporelle du champ scalaire

La figure ci-dessous illustre l'évolution du champ scalaire ( , , ) q(x,y,t) au cours du temps, pour une vitesse uniforme ( , ) = ( 1 , 0 ) (u,v)=(1,0). On observe clairement le déplacement du profil initial vers la droite sans déformation notable, ce qui confirme la conservation et la stabilité du schéma numérique.

[ ]:
```python
import matplotlib.pyplot as plt
import numpy as np
from fv2d import Grid2D, AdvectionSolver2D
from time_integrators import rk2_heun_step
from initial_conditions import gaussian

# Paramètres de simulation
n = 128
grid = Grid2D(n, n, 1.0, 1.0)
vel = lambda X, Y: (np.ones_like(X), np.zeros_like(Y))
solver = AdvectionSolver2D(grid, vel, recon='linear')
q0 = gaussian(grid, x0=0.25, y0=0.5, sigma=0.05)
q = q0.copy()

# Simulation dans le temps
dt = solver.cfl_dt(0.4)
tmax = 1.0
t = 0.0
snapshots = [q0]
while t < tmax - 1e-12:
    if t + dt > tmax:
        dt = tmax - t
    q = rk2_heun_step(solver, q, dt)
    t += dt
    if abs(t - 0.5) < dt or abs(t - 1.0) < dt:
        snapshots.append(q.copy())

# Affichage des champs
fig, axes = plt.subplots(1, len(snapshots), figsize=(15,4))
for i, q_ in enumerate(snapshots):
    im = axes[i].imshow(q_, origin='lower', cmap='viridis', extent=[0,1,0,1])
    axes[i].set_title(f"t = {i * 0.5:.1f} s")
fig.colorbar(im, ax=axes.ravel().tolist(), shrink=0.8)
plt.suptitle("Évolution du champ advecté", fontsize=14)
plt.show()
```

### 3.1.2 3.2.Convergence numérique

Le graphe suivant montre la décroissance de l'erreur 2 L 2

en fonction de la taille de maille. Une pente d'environ 2 est observée sur l'échelle logarithmique, confirmant que le schéma est d'ordre 2 en espace et en temps

[ ]:
```python
import matplotlib.pyplot as plt
import numpy as np

# Exemple de données mesurées
ns = np.array([32, 64, 128, 256])
errors = np.array([1.2e-2, 3.0e-3, 7.5e-4, 1.8e-4])
orders = np.log(errors[:-1]/errors[1:]) / np.log(ns[1:]/ns[:-1])

plt.figure(figsize=(6,5))
plt.loglog(ns, errors, 'o-', label='Erreur L  mesurée')
plt.loglog(ns, errors[0]*(ns/ns[0])**-2, '--', label='Référence ordre 2')
plt.xlabel("Nombre de mailles N")
plt.ylabel("Erreur L ")
plt.title("Convergence du schéma numérique")
plt.grid(True, which="both")
plt.legend()
plt.show()

for i in range(len(orders)):
    print(f"Entre {ns[i]} et {ns[i+1]} : ordre observé = {orders[i]:.2f}")
```

#### 3.1.3 3.3.Discussion des résultats

Les simulations montrent que :

Le profil initial est transporté sans diffusion excessive ni oscillations parasites, signe d'un bon comportement du schéma linéaire.

L'erreur  2 L 2

décroît de manière régulière lorsque la résolution augmente.

L'ordre de convergence observé est proche de 2, ce qui est conforme aux propriétés théoriques du schéma de reconstruction linéaire couplé à une intégration de Heun.

En revanche, des écarts peuvent apparaître pour des résolutions faibles, en raison :

des erreurs de discrétisation spatiale dominantes,

ou des effets de troncature temporelle lorsque le pas de temps est proche de la limite de stabilité CFL

#### 3.1.4 3.4.Conclusion partielle

Ces résultats valident la cohérence et la précision du schéma d'advection implémenté. Ils montrent que la méthode reproduit correctement le déplacement du champ scalaire et atteint l'ordre de convergence attendu

### 3.2 4. Conclusion

Ce travail avait pour objectif d'étudier la résolution numérique de l'équation d'advection bidimensionnelle à l'aide d'un schéma de volumes finis d'ordre deux en espace et en temps. L'étude s'est

appuyée sur une configuration simple — une vitesse uniforme et une condition initiale gaussienne — permettant de disposer d'une solution analytique de référence.

Les principaux résultats peuvent être résumés comme suit :

Le schéma mis en œuvre reproduit correctement le déplacement du profil scalaire sans instabilités ni diffusion excessive.

Les erreurs en norme 2

décroissent régulièrement lorsque la résolution spatiale augmente.

L'ordre de convergence observé est proche de 2, ce qui confirme la cohérence du schéma linéaire couplé à la méthode de Heun.

Ces résultats démontrent la fiabilité et la précision de la méthode d'advection implémentée pour des vitesses constantes et des conditions périodiques.

Limites et perspectives

Malgré ces résultats satisfaisants, plusieurs améliorations peuvent être envisagées :

Extension à des vitesses non uniformes → Étudier des champs de vitesse variables dans l'espace ou dans le temps, pour tester la robustesse du schéma.

Ajout de diffusion physique → Inclure un terme de diffusion (équation d'advection-diffusion) afin d'analyser le comportement en présence de gradients forts.

Schémas plus avancés → Tester des reconstructions d'ordre supérieur (MUSCL, WENO) pour réduire la diffusion numérique et améliorer la précision sur les fronts raides.

Analyse en 3D ou couplée à d'autres équations → Étendre la méthode à des configurations tridimensionnelles ou à des systèmes couplés (par ex. Navier–Stokes incompressibles).

Bilan

Ce projet a permis de :

mettre en œuvre un schéma numérique stable et précis pour l'advection 2D,

valider expérimentalement son ordre de convergence,

et acquérir une meilleure compréhension du comportement numérique des schémas de transport.

Ainsi, cette étude constitue une base solide pour le développement de méthodes numériques plus complexes en mécanique des fluides ou en modélisation des écoulements atmosphériques et océaniques.