Solveur Elliptique Multi-Grille [1]

Implémentation Python/PyTorch et Applications à la Mécanique des Fluides

Gauvain Thomas[12]

[1]ENS Rennes [2]INRIA - Équipe ODYSSEY

22 juillet 2024

1. Stage de L3 encadré par Louis Thiry

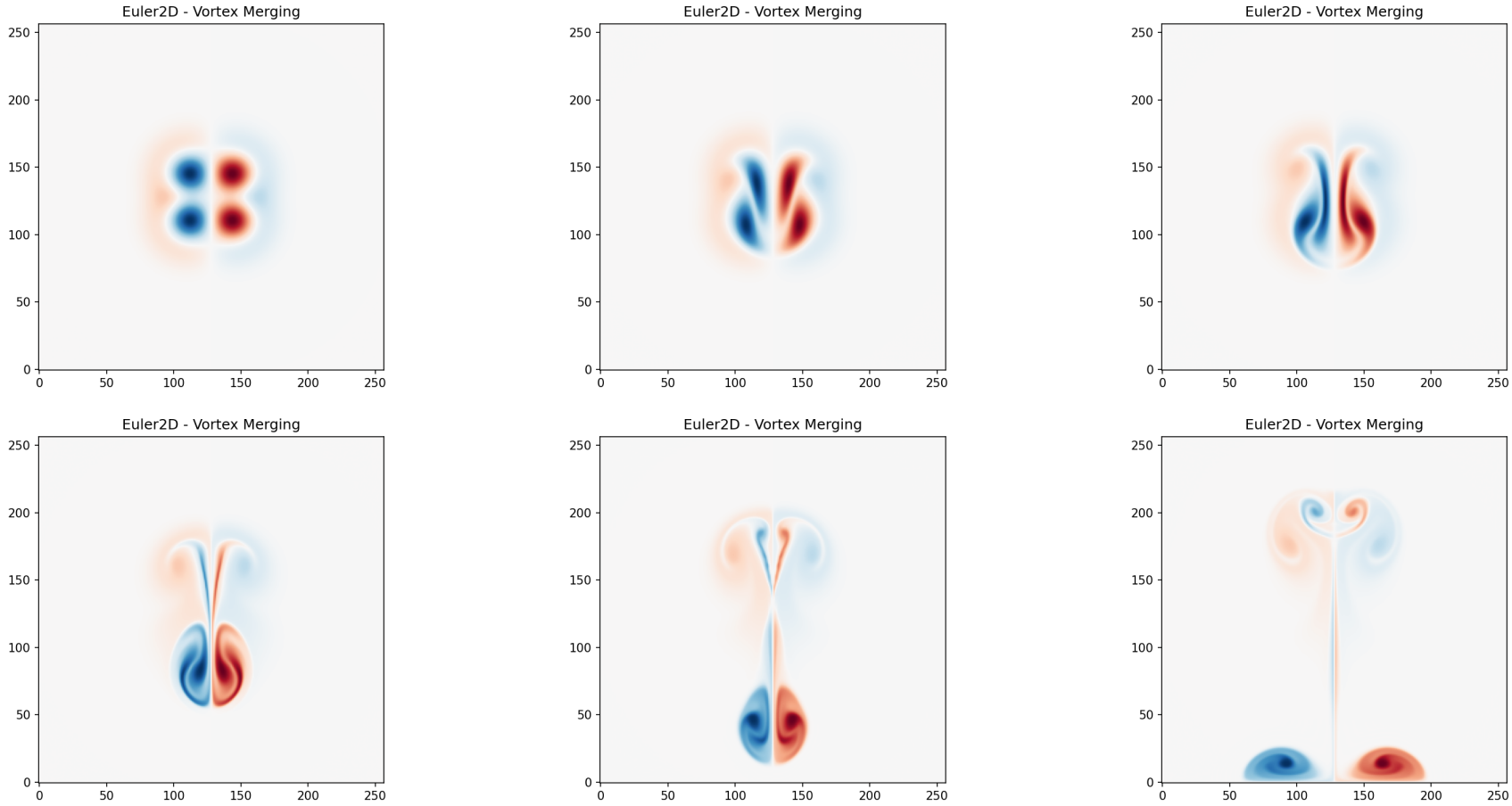


Figure – Vortex merging : snapshots de la vorticité pour $t \in \{1, 20, 40, 90, 150, 240\}$

Solveurs actuels

- Fortran
  Codes longs et peu adaptables
- Solveurs spectraux FFT, DCT, DST
  Rapides et optimisés, mais limités aux géométries rectangulaires

Solveurs actuels

- Fortran
  Codes longs et peu adaptables
- Solveurs spectraux FFT, DCT, DST
  Rapides et optimisés, mais limités aux géométries rectangulaires

Objectifs

- Solveur lisible et modulable, en tirant profit des nouveaux modèles de résolution itératifs ainsi que des architecures : Python/PyTorch
- Comparaison des performances.

Table of Contents

Table of Contents

Problème - Version continue

$\Omega \in \mathbb{R}^d$ un ouvert, $\Gamma = \partial\Omega$

L un opérateur linéaire elliptique

$$\begin{cases} L^{\Omega} u = f & (\Omega) \\ L^{\Gamma} u = f & (\Gamma) \end{cases}$$

Problème - Version continue

$\Omega \in \mathbb{R}^d$ un ouvert, $\Gamma = \partial\Omega$
L un opérateur linéaire elliptique

$$\begin{cases} L^{\Omega} u = f & (\Omega) \\ L^{\Gamma} u = f & (\Gamma) \end{cases}$$

Exemples (sans conditions aux limites)

- Équation de Poisson :
  $\Delta u = f \quad (\Omega)$
  $L = \Delta$
- Équation de Helmholtz :
  $\Delta u - \lambda u = f, \quad \lambda > 0 \quad (\Omega)$
  $L = \Delta - \lambda I$

Avec $\Delta = \sum_{i=1}^{d} \frac{\partial}{\partial x_i}$ l'opérateur laplacien.

Conditions aux limites

Conditions de Dirichlet homogènes

$$u(\boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \Gamma \tag{1}$$

Conditions aux limites

Conditions de Dirichlet homogènes

$$u(\boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \Gamma \tag{1}$$

Conditions de Neumann homogènes

$$\frac{\partial u}{\partial n}(\boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \Gamma \tag{2}$$

Conditions aux limites

Conditions de Dirichlet homogènes

$$u(\boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \Gamma \tag{1}$$

Conditions de Neumann homogènes

$$\frac{\partial u}{\partial n}(\boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \Gamma \tag{2}$$

Unicité

Les solutions du problème est définie à une constante près dans un cas avec conditions de Neumann.

Discrétisation

Laplacien 5 points en 2D

Pour $(x, y) \in \Omega_h \setminus \Gamma_h$

$$\begin{aligned}\Delta_h u_h(x,y) &= \frac{u_h(x+dx,y)+u_h(x-dx,y)-2u_h(x,y)}{dx^2} \\ &+ \frac{u_h(x,y+dy)+u_h(x,y-dy)-2u_h(x,y)}{dy^2} \\ &= \frac{1}{h^2}[u_h(x+h,y)+u_h(x-h,y)+u_h(x,y+h) \\ &+ u_h(x,y-h)-4u_h(x,y)]\end{aligned} \tag{3}$$

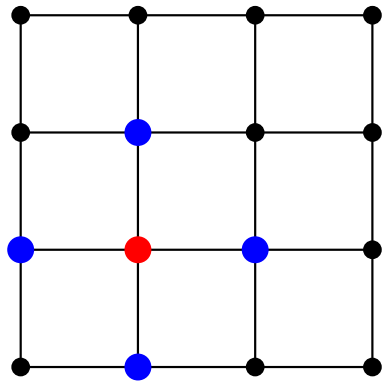


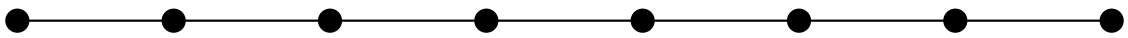
Table of Contents

Jacobi

En 1D

$$\Delta_h u = \frac{u(x+h) + u(x-h) - 2u(x)}{h^2} = f(x)$$

$$u(x) = \frac{u(x+h) + u(x-h) - h^2 f(x)}{2} := z(x)$$

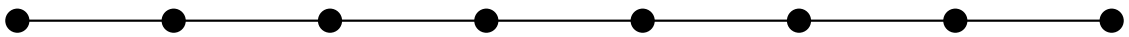


Jacobi

En 1D

$$\Delta_h u = \frac{u(x+h) + u(x-h) - 2u(x)}{h^2} = f(x)$$

$$u(x) = \frac{u(x+h) + u(x-h) - h^2 f(x)}{2} := z(x)$$



Jacobi iteration

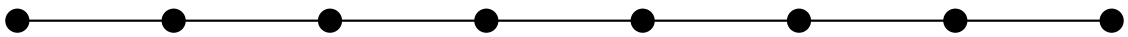
$$u^{m+1} = z^{m+1} \tag{4}$$

Jacobi

En 1D

$$\Delta_h u = \frac{u(x+h) + u(x-h) - 2u(x)}{h^2} = f(x)$$

$$u(x) = \frac{u(x+h) + u(x-h) - h^2 f(x)}{2} := z(x)$$



Jacobi iteration

$$u^{m+1} = z^{m+1} \tag{4}$$

ω-relaxed Jacobi iteration

$$u^{m+1} = u^m + \omega(z^{m+1} - u^m) = (1-\omega)u^m + \omega z^{m+1} \tag{5}$$

Code

```
def jacobi_smoothing(u, f, dx, dy, omega=.8, lambd=0.):
    """Jacobi relaxation method for vertex-centered grid
    with Dirichlet boundary conditions"""

    # Dirichlet BC
    u[..., [0,-1], :] = 0
    u[..., [0,-1]] = 0

    Ax=1.0/dx**2; Ay=1.0/dy**2
    Ap=1.0/(2.0*(Ax+Ay) + lambd)
    u[...,1:-1,1:-1] = (
            omega*Ap*(Ax*(u[...,2:,1:-1] + u[...,:-2,1:-1])
            + Ay*(u[...,1:-1,2:] + u[...,1:-1,:-2])
            - f[...,1:-1,1:-1])
            + (1-omega)*u[...,1:-1,1:-1])
    return u
```

Gauss-Seidel rouge-noir

Méthode de Jacobi successivement sur chaque couleur

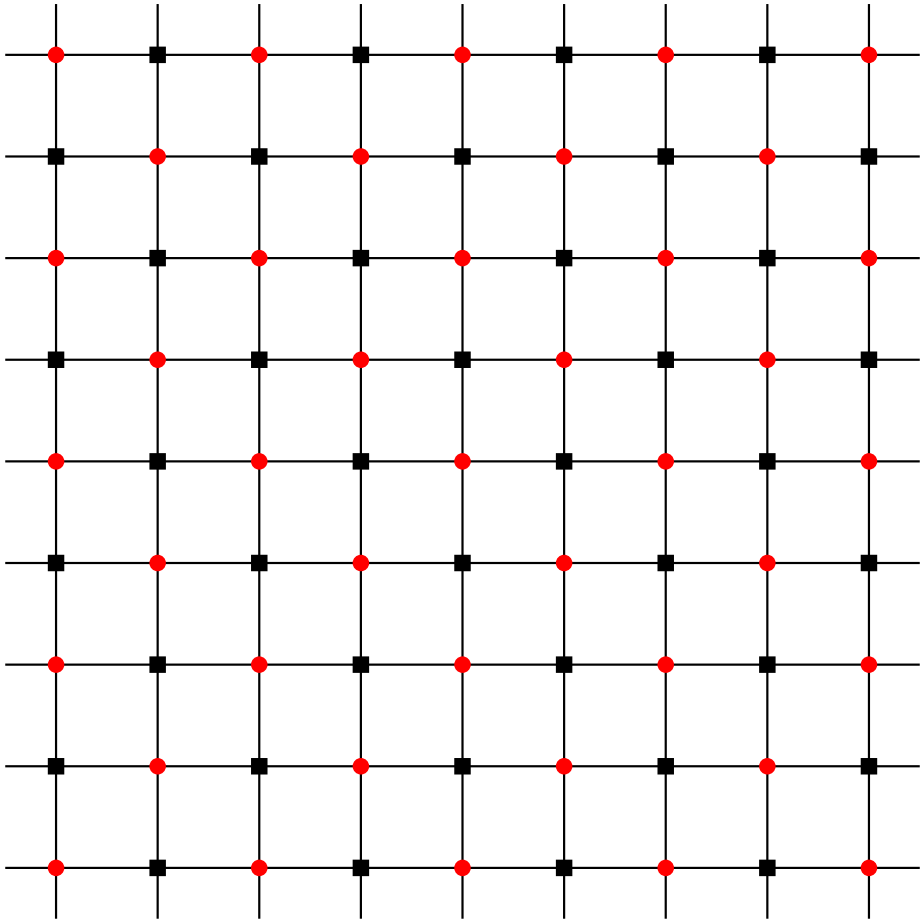


Figure – Découpage rouge/noir pour la méthode de Gauss-Seidel

Table of Contents

Méthodes itératives

Objectif : résoudre $Lu = f$

Définitions

- Solution approchée : u^m
- Erreur : $e^m = u - u^m$
- Résidu : $r^m = f - Lu^m$

Méthodes itératives

Objectif : résoudre $Lu = f$

Définitions

- Solution approchée : u^m
- Erreur : $e^m = u - u^m$
- Résidu : $r^m = f - Lu^m$

Équation du résidu

$$Le^m = r^m \tag{6}$$

Méthodes itératives

Objectif : résoudre $Lu = f$

Définitions

- Solution approchée : u^m
- Erreur : $e^m = u - u^m$
- Résidu : $r^m = f - Lu^m$

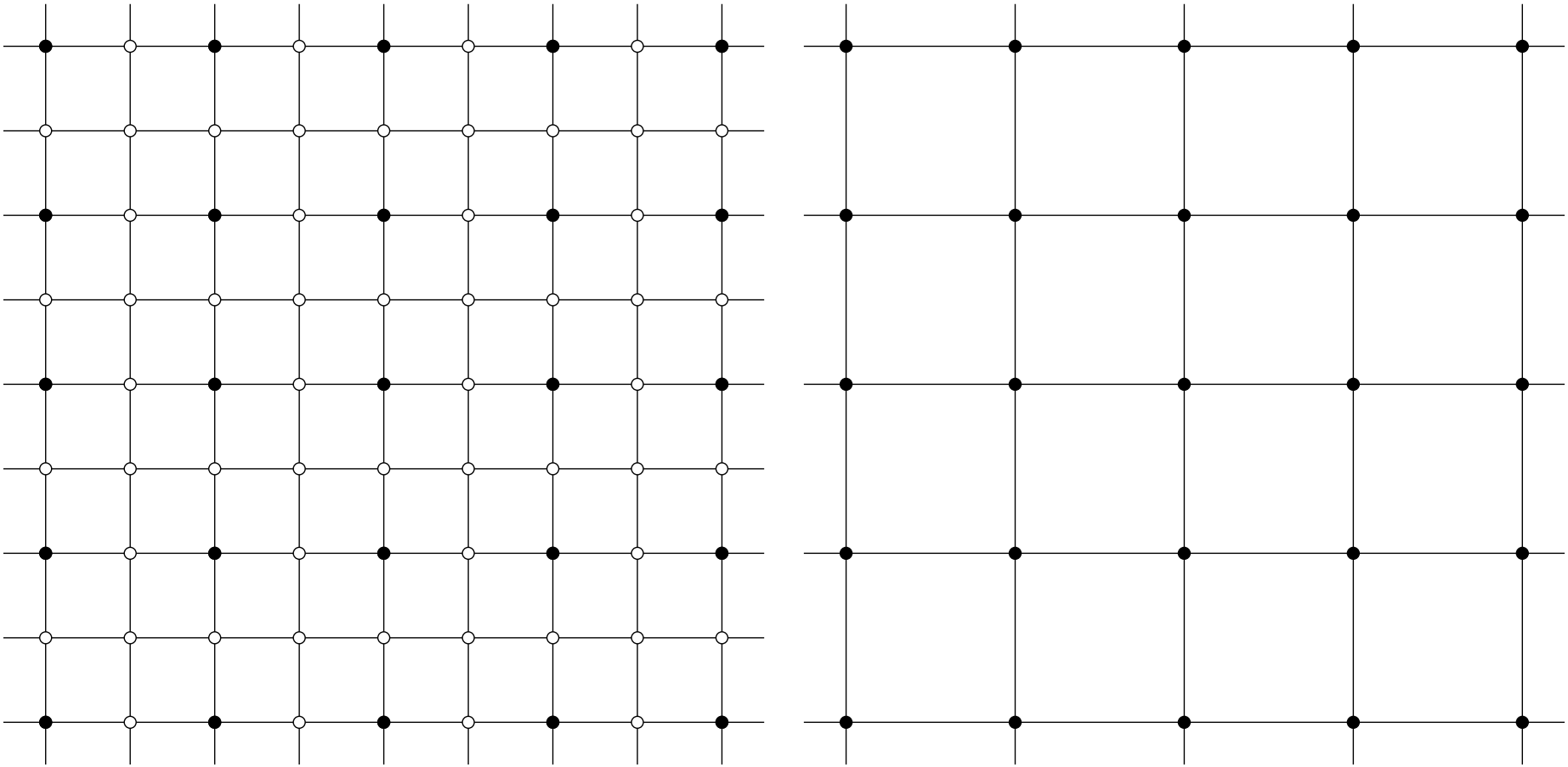
Équation du résidu

$$Le^m = r^m \tag{6}$$

Amélioration de l'approximation

$$u^{m+1} = u^m + e^m \tag{7}$$

Multi-grille



(a) Ω_h (b) Ω_{2h}

Figure – Une grille fine et une grille grossière

Two-Grid Cycle

Rappel : équation du résidu

$$Le^m = r^m \tag{6}$$

Résolution avec une approximation L_H de L_h (en général $H = 2h$)

- Calcul du résiduel $r_h^m = f_h - L_h u_h^m$
- Restriction du résiduel r_H sur la nouvelle grille Ω_H
- Résolution sur Ω_H : $L_H e_H^m = r_H^m$
- Interpolation de la correction (approximée) sur Ω_h : e_h
- Calcul de la nouvelle approximation : $u_h^{m+1} = u_h^m + e_h$

En réalité non convergent : il faut appliquer une méthode de relaxation avant et/ou après la correction.

V-cycle

Même idée que pour le Two-Grid Cycle, en résolvant récursivement sur la grille approchée.

Algorithme

- Presmoothing
- Calcul du résiduel $r_h^m = f_h - L_h u_h^m$
- Restriction du résiduel r_H sur la nouvelle grille Ω_H
- Résolution sur Ω_H : $L_H e_H^m = r_H^m$ récursivement (avec 0 en approximation initiale)
- Interpolation de la correction (approximée) sur Ω_h : e_h
- Calcul de la nouvelle approximation : $u_h^{m+1} = u_h^m + e_h$
- Postsmoothing

V-cycle

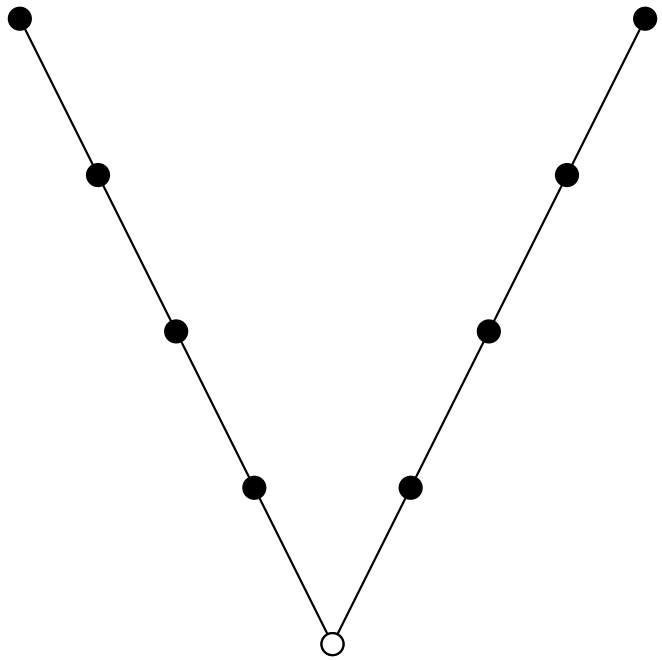


Figure – Structure d'un V-cycle | • : relaxation, ○ : solution exacte, \ : restriction, / : interpolation

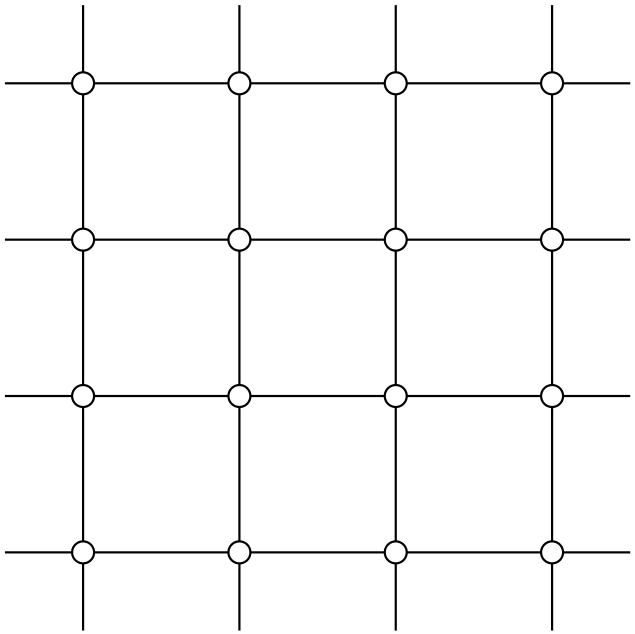
Performances

Table – Performances d'un V-cycle pour une grille $N \times N$ (en temps moyen par point de grille)

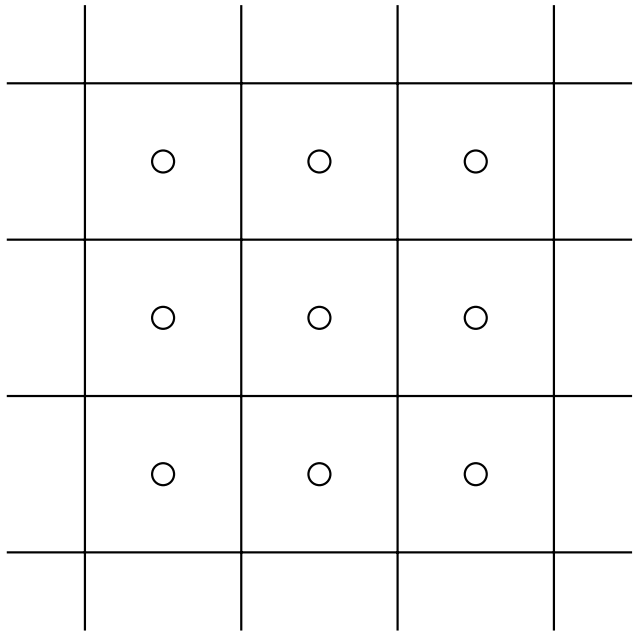
Solveur	N=256	N=512	N=1024
Fortran	3.25e-8 s	3.52e-8 s	4.85e-8 s
Python PyTorch non compilé	7.54e-7 s	3.47e-7 s	2.95e-7 s
Python PyTorch (CPU)	3.83e-7 s	1.69e-7 s	1.50e-7 s
Python PyTorch (GPU)	3.87e-7 s	1.04e-7 s	3.83e-8 s

Calculé sur l'exemple d'un bruit gaussien

Types de grilles



(a) Grille vertex-centered



(b) Grille cell-centered

Figure – Exemple d'une grille vertex-centered et d'une grille cell-centered

Full Multi-Grid

FMG : Calcul d'une approximation initiale avant V-cycle

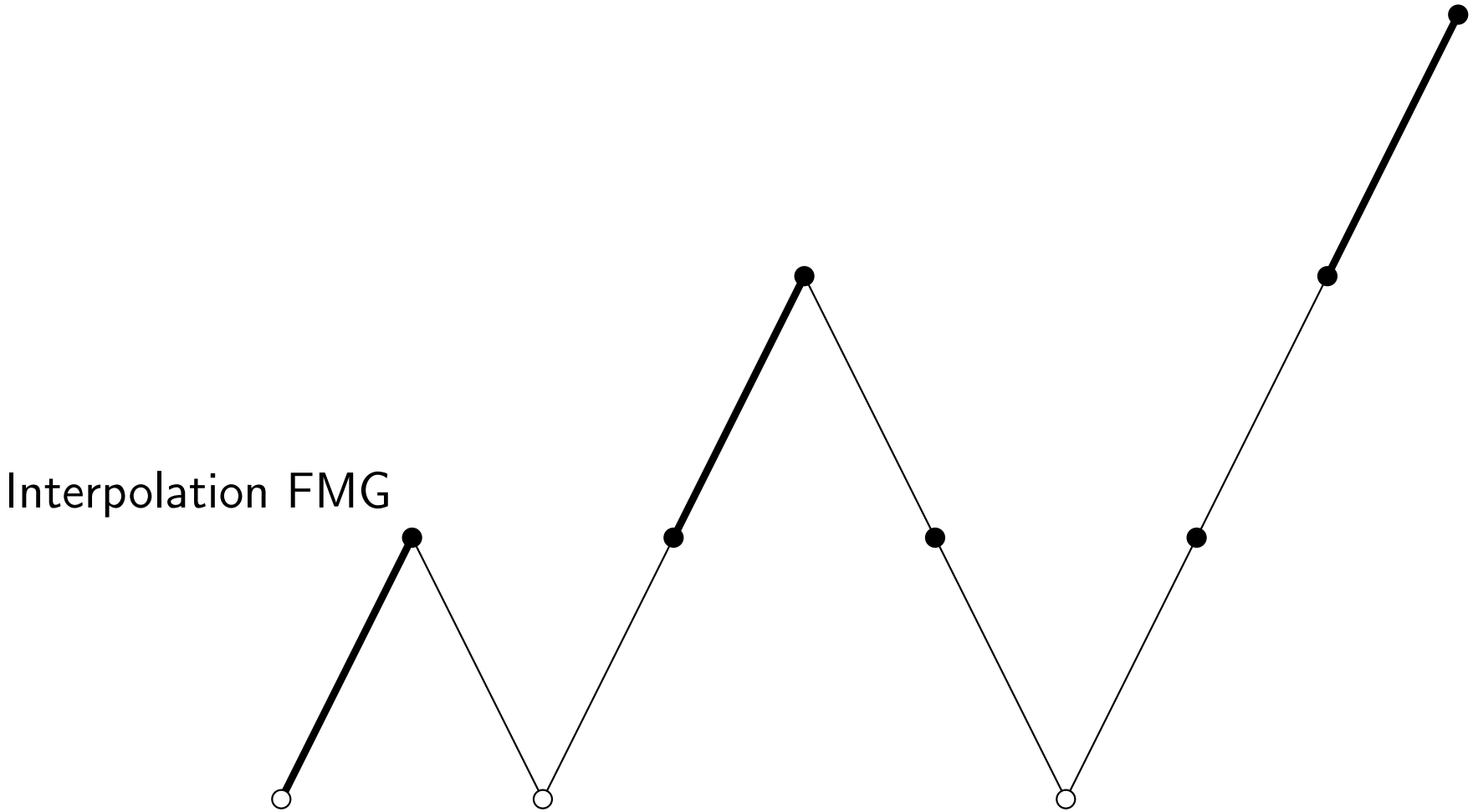


Figure – Structure d'un FMG

Table of Contents

Conclusion

Une implémentation simple et adaptable, avec des performances correctes grâce à des outils performants et une architecture CPU/GPU.
Permet une facilité de travail et d'amélioration des fonctionnalités.

Extensions

- Multi-grille 3D
- Interface avec appel à des routines optimisées
- Géométries complexes / Grilles non-uniformes
- Gestion de types de grilles différentes
- Résolution directe à basse résolution - optimisation CPU/GPU
- Pré-conditionneurs