Multiscale Finite Volume Method

Louis Glänzer / Jonathan Ulmer

August 19, 2025

Outline

Outline

Numerical Problem

Diffusion Equation

- We solve the diffusion equation (1)

$$\nabla \cdot (D(x)\nabla c) = f(x) \qquad \text{in } \Omega \tag{1}$$

$$c(x) = 0 \qquad \text{on } \partial\Omega \tag{2}$$

- with Dirichlet boundary (2)
- We use constant source $f(x) = 1, x \in \Omega$
- We use a Finite Volume solver

Derivation of the 1D Finite Volume Method

Discretization

- Divide Ω into a uniform grid $\{Q_i\}$
  - cells Q_i are disjunct
  - $Q_i \cap Q_j = \emptyset, i \neq j$
  - we assume constant solution c_i on Q_i
  - we assume constant source $f(x) = f_i$, $x \in Q_i$

Derivation of the 1D Finite Volume Method

Integral Form

- since the Diffusion EQ.(1) holds on Ω, it holds for any integral.
- $\implies$ (1) holds for the integral over Q_i

$$\begin{aligned} \nabla \cdot (D(x)\nabla c) &= f(x) && \text{in}\,\Omega \\ \int_{Q_i} \nabla \cdot (D(x)\nabla c) &= \int_{Q_i} f(x)\,\mathrm{d}x && i = 1, \dots, N \end{aligned} \tag{3}$$

- We use the constant source to evaluate the right hand side
- We transform (3) with the Stokes theorem
  - reduces the derivative degree on c and D

$$\int_{\partial Q_i} D(x)\nabla c \cdot \vec{n}\,\mathrm{d}S = |Q_i| f_i \qquad i = 1, \dots, N \tag{4}$$

Derivation of the 1D Finite Volume Method

Numerical Flux

- constant solution
- c is discontinuous on ∂Q
- ∇c is not defined on the cell boundary
- we replace $D(x)\nabla c$ with numerical flux $\vec{g}$
- $\vec{g}$ describes flux of the solution between neighbouring cells

$$\int_{\partial Q_i} D(x)\nabla c \cdot \vec{n}\,\mathrm{d}S = \int_{Q_i} f(x)\,\mathrm{d}x \qquad i = 1, \dots, N \tag{5}$$

$$\int_{\partial Q_i} \vec{g}(c^+, c^-) \cdot \vec{n}\,\mathrm{d}S = \int_{Q_i} f(x)\,\mathrm{d}x \qquad i = 1, \dots, N \tag{6}$$

Numerical flux approximation

1D Flux

We employ the flux approximation introduced in the MMM Lecture. Since we only investigated diffusion terms with an analytical representation, we are able to calculate this value directly.

$$g(c^+, c^-) = -D(x^{\frac{1}{2}+})\frac{c^+ - c^-}{h} \tag{7}$$

Furthermore, we introduce transmissitivities $T_\pm$ between both cells.

$$g(c^+, c^-) = T_\pm * \left(c^+ - c^-\right)$$
$$T_\pm = -D(x^{\frac{1}{2}+})\frac{1}{h}$$

Numerical flux approximation

2D Flux

We define the flux term $\vec{g} := (g_x, g_y)^T$ in 2 Dimensions very similar to those in one dimension.

$$g_x(c_{i+1,j}, c_{ij}) = -\Delta_y D(x_{i+\frac{1}{2},j}) \frac{c_{i+1,j} - c_{ij}}{\Delta_x} \tag{8}$$

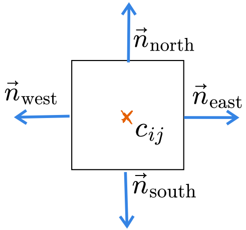
$$g_y(c_{i,j+1}, c_{ij}) = -\Delta_x D(x_{i,j+\frac{1}{2}}) \frac{c_{i,j+1} - c_{ij}}{\Delta_y} \tag{9}$$

and in the same manner we introduce 2D transmissions T^x_{i+1j}, T^y_{ij+1}

$$g_x(c_{i+1j}, c_{ij}) = T^x_{i+1j}\,(c_{i+1j} - c_{ij})$$
$$g_y(c_{ij+1}, c_{ij}) = T^y_{ij+1}\,(c_{i+1j} - c_{ij})$$

Linearization

Reference Cell for finite Volume

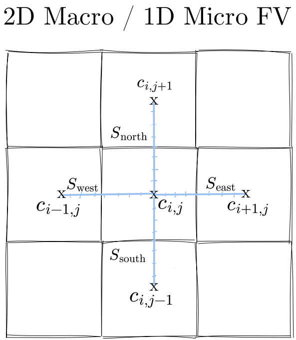


- only 4 normals in 2D (2 in 1D)
- integral simplifies

$$\int_{\partial Q_i} \vec{g}(c^+, c^-) \cdot \vec{n} \, \mathrm{d}S = |Q_i| f_i$$

$$\sum_{n \in \partial Q} \vec{g}(c_{ij+\vec{n}}, c_{ij}) \cdot \vec{n} = |Q_i| f_i$$

Multiscale



- Adds microscale simulation
  - one dimensional
  - for each normal
  - n subgrid cells
- Calculates Transmissions T_S
  - with microscale solution ϕ_S

$$T_S = -\int_{S_0}^{S_1} D(x)(\phi_S'(\vec{x}))^2 \,\mathrm{d}S$$

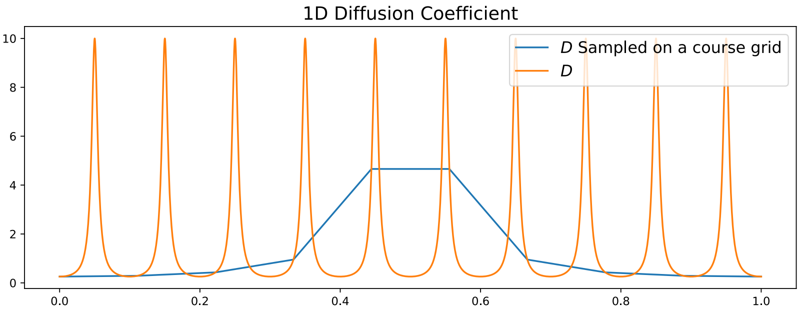
Outline

1D

Since the aim of multiscale Finite Volume is to improve the results for highly fluctuating diffusivities.

Code

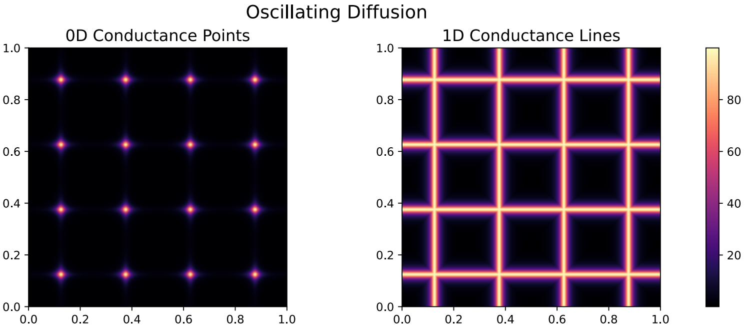
```python
def oscillation(x, eps = 0.1):
    return 1 / (2+1.9 * np.cos(2 * np.pi* x / eps))
```



2D Oscillation

Code

```
def osc2D_point(x,y , eps = 0.25):
    return oscillation(x, eps=eps) * oscillation(y, eps=eps)
def osc2D_line(x,y , eps = 0.25):
    return np.maximum(oscillation(x, eps=eps) , oscillation(y,
    ↪  eps=eps))
```



2D Box Condition

To test numerical stability of our methods, we introduce a box constraint that traps some concentration in the center.

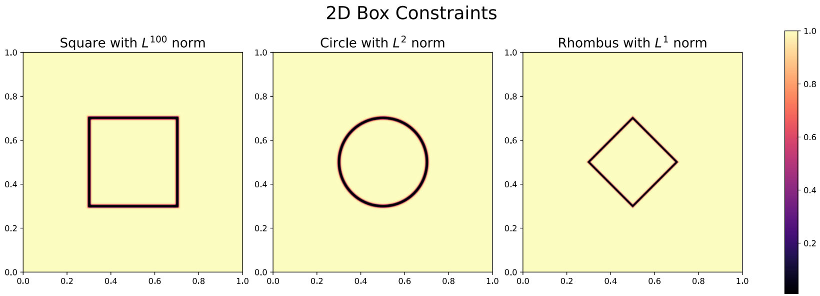


Figure: Constraints restricting flow from the center of the Domain

Outline

Program Structure

Class Structure

```
class FVSolver:
   N : int
   resolution : int
   h : np.float64
   x : NDArray[np.float64]
   D : Callable
   f : NDArray[np.float64]
   c : NDArray[np.float64]
   micro_basis : NDArray[np.float64]
   _T : NDArray[np.float64]

<<Init>>
<<Assemble Matrix>>
<<Boundary>>
<<Solve>>
<<Microscale Transmissions>>
<<Reconstruct Microscale Solution>>
```

Program Structure

Initialization

```
def __init__(self , N :int , D :Callable  , domain=(0.,1.))->None:
    self.h = (domain[1] - domain[0]) / (N-1)
    self.N = N
    self.D = D
    self.x = np.linspace(domain[0] , domain[1] , N)
    self._T =  -1/self.h * D((self.x[:-1] + self.x[1:])*0.5)
    self.f = self.h* np.ones(N)
```

Program Structure

Boundary

```
def set_boundary(self , bc=(0.,0.)):
    self.f[0] = bc[0]
    self.f[-1] = bc[1]
```

Solving

```
def solve(self):
    self.c = spsolve(self._A.tocsr() , self.f)
    return self.c
```

Program Structure

Matrix Assembly

```
def assemble_matrix(self)-> None:
    diagp1 = np.zeros(self.N)
    diagp1[2:] =  self._T[1:]
    diagm1 = np.zeros(self.N)
    diagm1[:-2] =  self._T[:-1]
    diag0 = np.ones(self.N)
    diag0[1:-1] = -1 * (self._T[1:] + self._T[:-1])
    self._A = spdiags([diagm1 , diag0 , diagp1] , np.array( [-1, 0,
    ↪   1] ))
```

Program Structure

Sparsity Pattern of the linear system

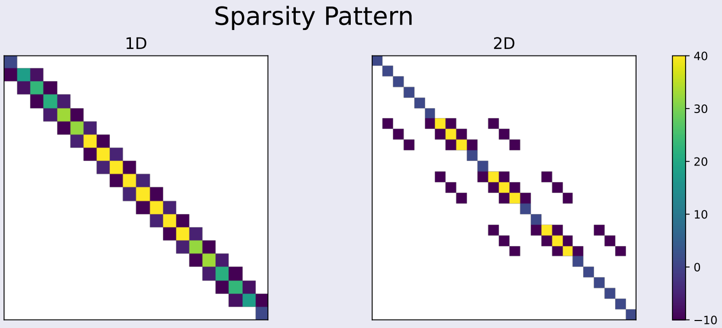
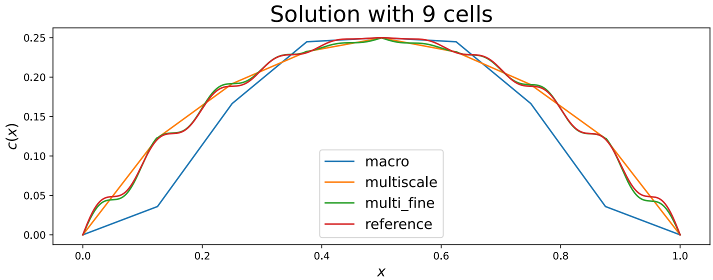
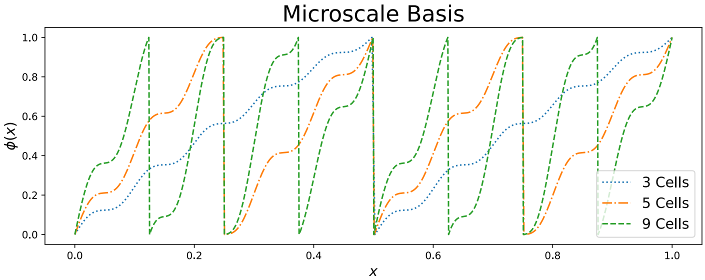


Figure: Non zeros values of the system matricies for a 20 cell 1D and a 5×5 cell 2D system.

Outline

1D Oscillating Diffusion





Line Diffusion

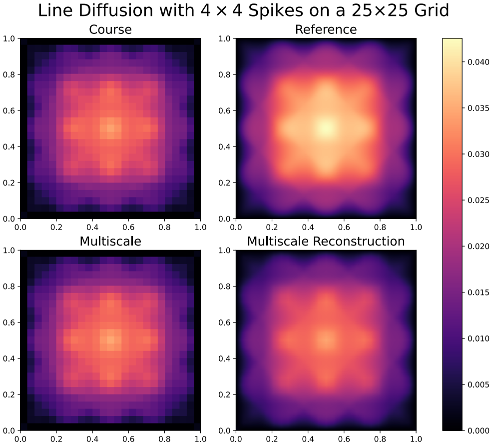


Figure: Comparison of different 2D solutions with a 1000×1000 reference solution.

Point Diffusion

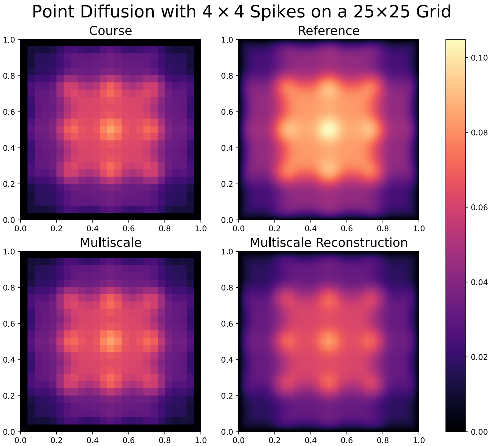


Figure: Comparison of different 2D solutions with a 1000×1000 reference solution.

2D Box

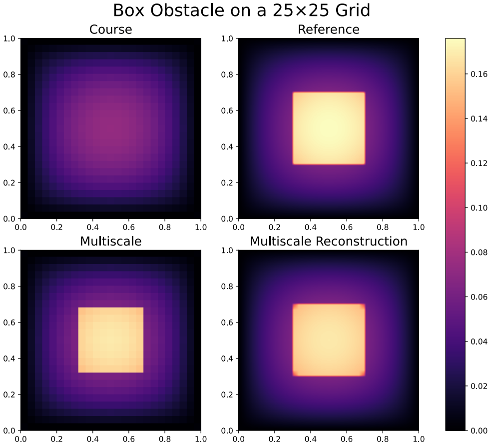


Figure: Comparison of different 2D solutions with a 1000×1000 reference solution.

2D Circle

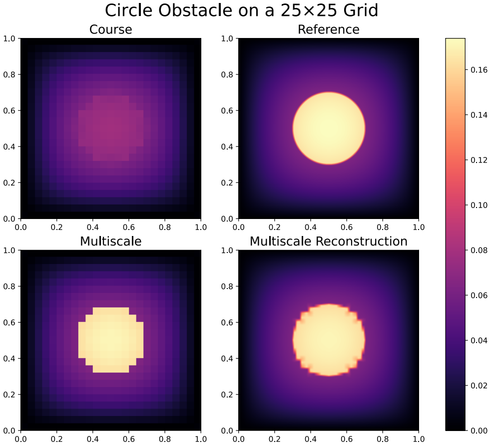


Figure: Comparison of different 2D solutions with a 1000×1000 reference solution.

2D Diamond

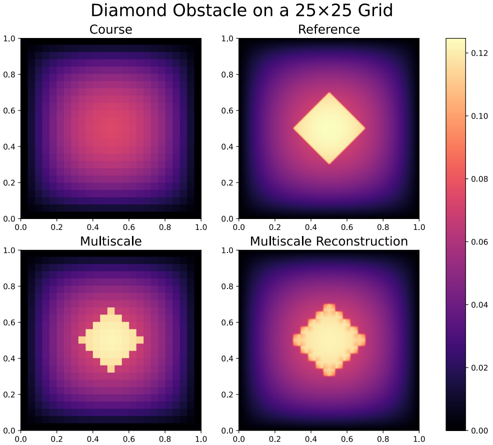


Figure: Comparison of different 2D solutions with a 1000×1000 reference solution.

Outline

1D Oscillating Diffusion

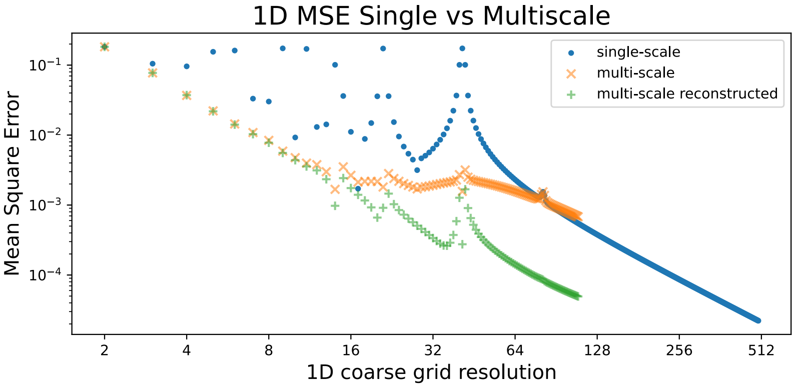


Figure: Mean Square Error (MSE) against a reference solution with a 1000 cell reference grid. For a 1D diffusion with 20 spikes and a 1000 cell microscale resolution.

2D Circle

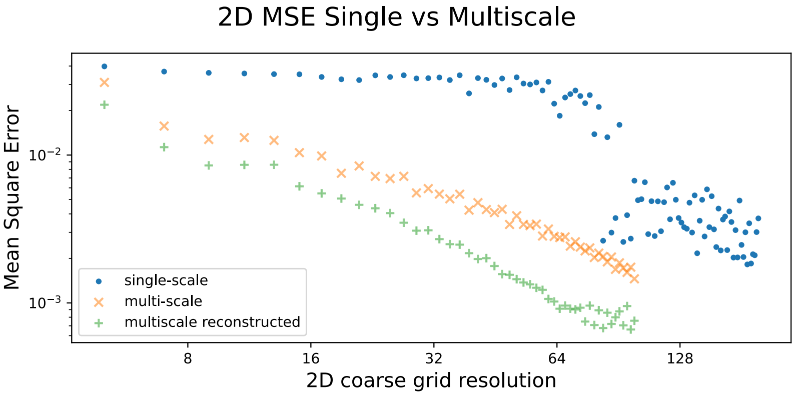


Figure: Mean Square Error (MSE) against a reference solution with a 1000×1000 cell reference grid. For a circular Obstacle and a 100 cell microscale resolution.

2D Box

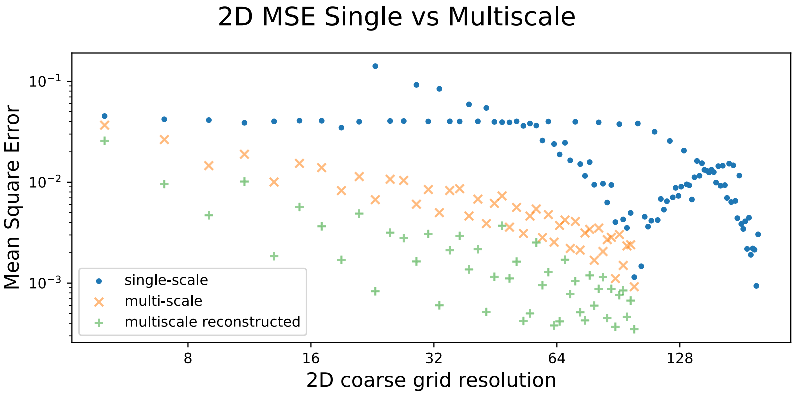


Figure: Mean Square Error (MSE) against a reference solution with a 1000×1000 cell reference grid. For a box Obstacle and a 100 cell microscale resolution.

2D Diamond

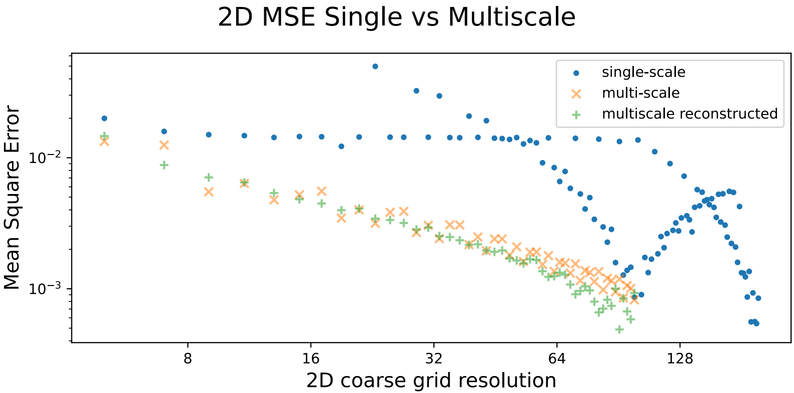


Figure: Mean Square Error (MSE) against a reference solution with a 1000×1000 cell reference grid. For a diamond obstacle and a 100 cell microscale resolution.

Line Diffusion

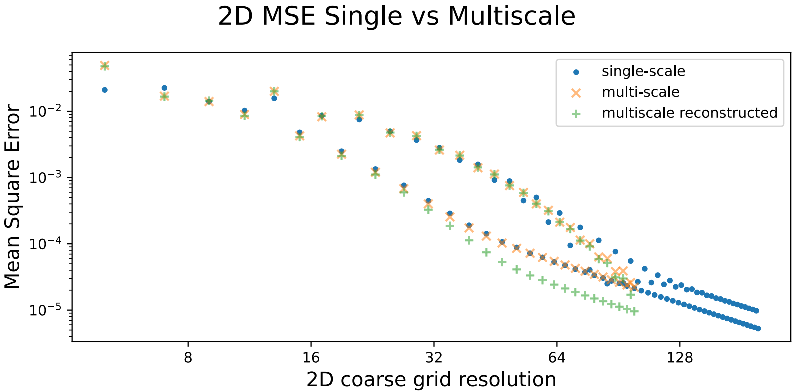


Figure: Mean Square Error (MSE) against a reference solution with a 1000×1000 cell reference grid. For a line diffusion with 4 spikes and a 100 cell microscale resolution.

Point Diffusion

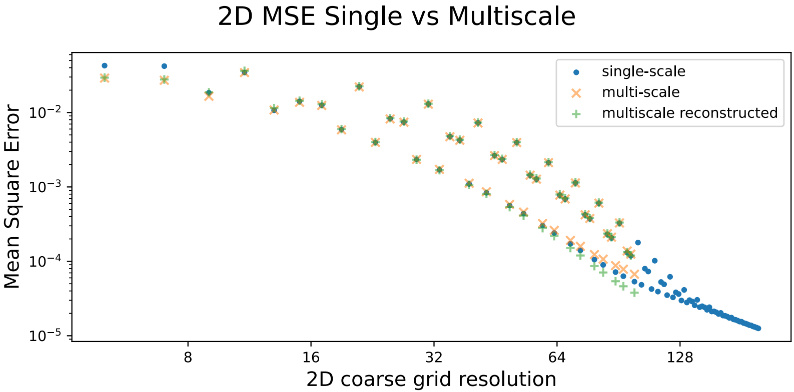


Figure: Mean Square Error (MSE) against a reference solution with a 1000×1000 cell reference grid. For a point diffusion with 4 spikes and a 100 cell microscale resolution.

Outline

Conclusion

- works well for 1D
  - no performance benefit
  - expensive microscale simulations
  - worse if single scale can resolve the details
  - reconstruction works very well
- works well for some 2D cases
  - can resolve thin walls
  - can reconstruct only orthogonal detail
- no benefit in other cases
  - no benefit for point and line diffusion
- requires some differentiability conditions on $D(\vec{x})$

Further Considerations

- multipoint flux approximation
- higher order 2D reconstruction
- realistic diffusion with interpolated high res data
- 3D
- irregular grid
- parallelization to make use of independent microscale simulations