Multiscale Finite Volume Method

Karl Louis Glänzer / Jonathan Ulmer

August 19, 2025

Outline

Numerical Problem I

Diffusion Equation

- We solve the diffusion equation (1)

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

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

- 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 I

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 II

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

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

Derivation of the 1D Finite Volume Method III

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 I

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.

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

2D Flux

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

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

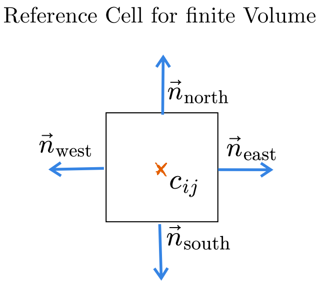
$$g_y(c_{i,j+1}, c_{ij}) = -\Delta_x D(x_{i,j+\frac{1}{2}}) \frac{c_{i,j+1} - c_{ij}}{\Delta_y} \tag{9}$$

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

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

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

Linearization I

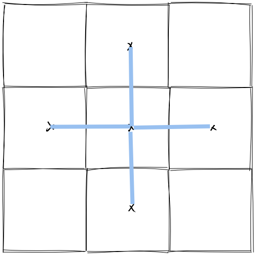


- only 4 normals in 2D (2 in 1D)
- integral simplifies

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

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

Multiscale I



- Adds microscale simulation
  - one dimensional
  - for each normal
  - n subgrid cells
- Calculates Transmissions T_{ij}
  - with microscale solution ϕ

$$T_{\pm} = -\int_Q D(x)(\phi'_{\pm}(x))^2 \, \mathrm{d}x$$

Diffusion I

We investigate of single and multiscale solvers with different Diffusion functions, that we introduce in the following sections

1D I

Since the Aim of multiscale Finite Volume, is to improve the results for highly fluctuating diffusivities, we test with the following oscillating function

Code

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

1D II

Diffusivity

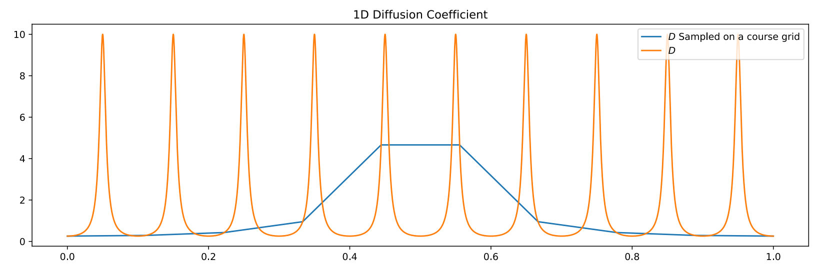


Figure: Oscillating Diffusivity for 1D finite volume method

2D Oscillation I

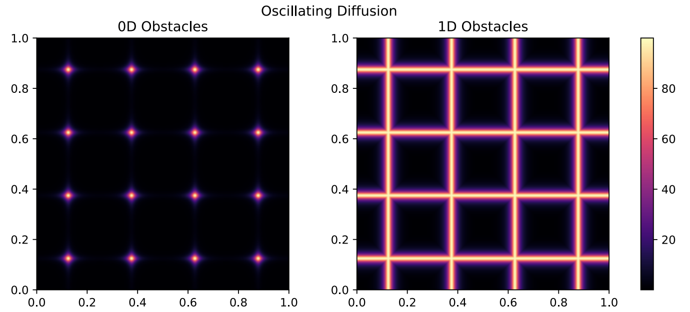
Code

```
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))
```

None

2D Oscillation II

Diffusion



2D Box Condition I

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

Diffusivity

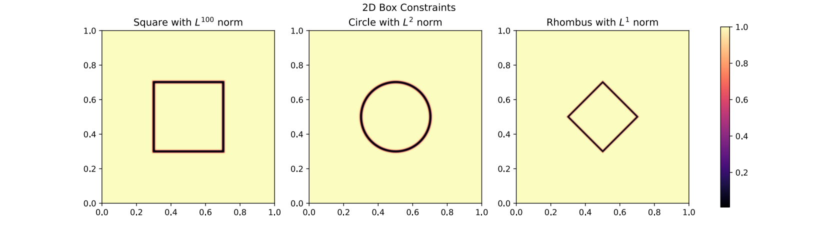


Figure: Constraints restricting flow from the center of the Domain

Program Structure I

For convenience in Explanation and Execution, we bundle all required information for solving a 1D system into a python class, which is structured as follows

Program Structure II

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 III

Initialization

```python
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)
```

Solving

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

Program Structure IV

Boundary

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

Assembly of the linear system

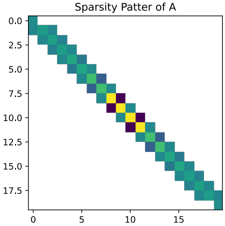
Program Structure V

Matrix Assembly

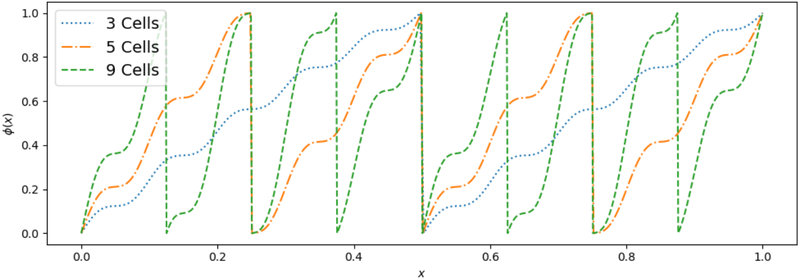
```
def assemble_matrix(self)-> None:
    diagp1 = np.zeros(self.N)
    diagp1[2:] =  self._T[1:]
    diagm1 = np.zeros(self.N)
    diagm1[:-2] =  self._T[:-1]
    diag0 = np.ones(self.N)
    diag0[1:-1] = -1 * (self._T[1:] + self._T[:-1])
    self._A = spdiags([diagm1 , diag0 , diagp1] , np.array( [-1,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

Program Structure VI

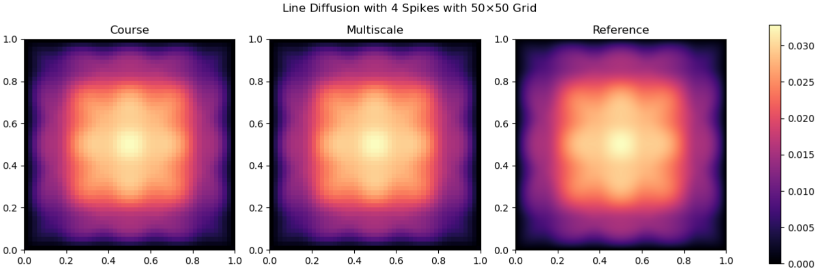
Sparsity Pattern of the linear system

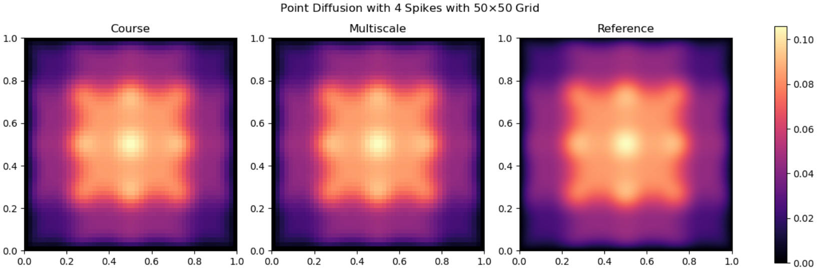


Microscale Basis

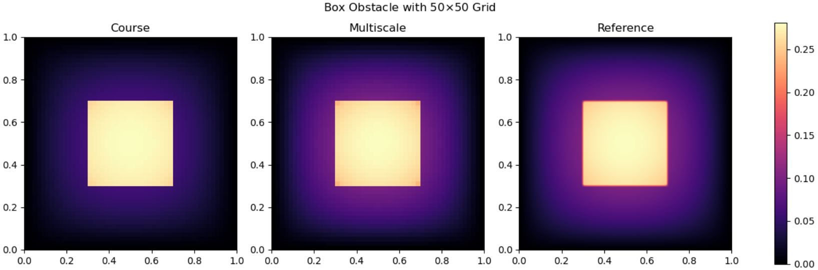


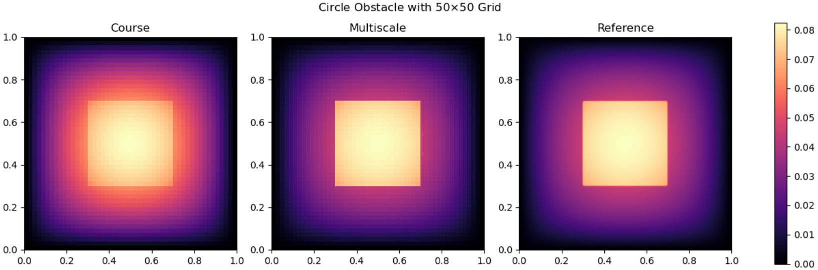
Oscillations I



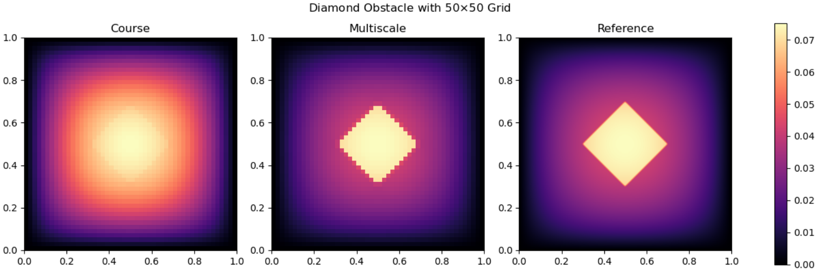


Box Conditions I

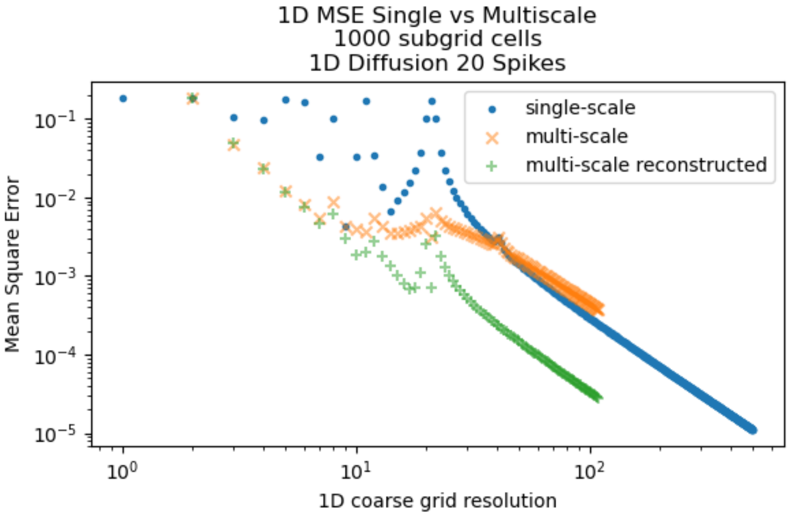




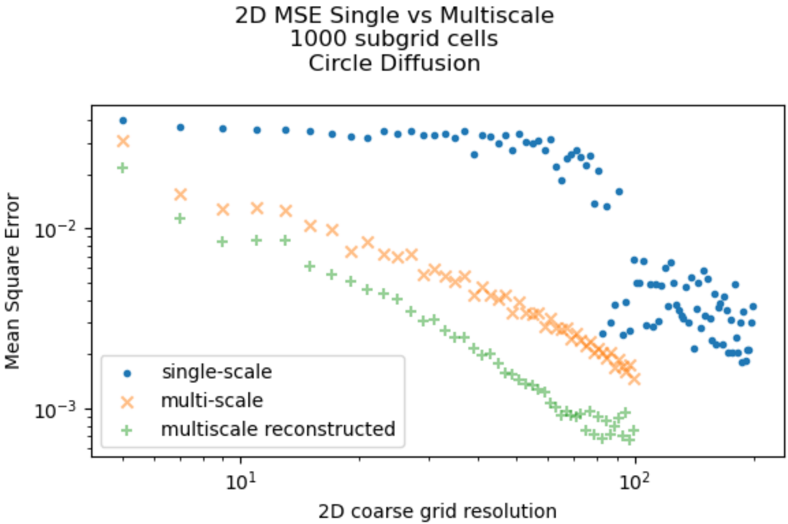
Box Conditions II



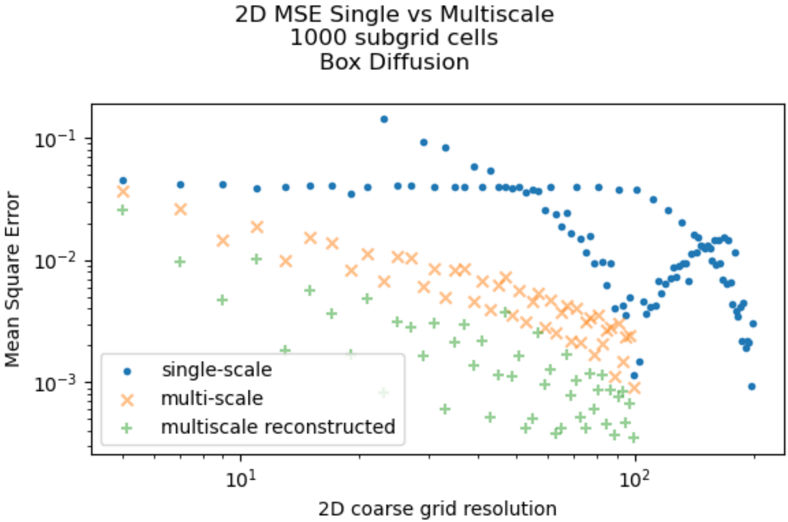
1D Error I


1D MSE Single vs Multiscale
1000 subgrid cells
1D Diffusion 20 Spikes
single-scale
multi-scale
multi-scale reconstructed
Mean Square Error
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{0}
10^{1}
10^{2}
1D coarse grid resolution

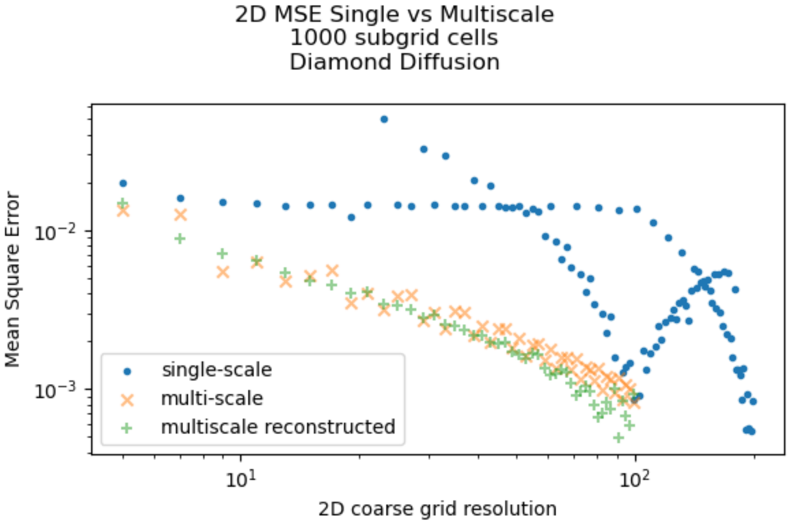
2D Error I


2D MSE Single vs Multiscale
1000 subgrid cells
Circle Diffusion
Mean Square Error
10^{-2}
10^{-3}
10^{1}
10^{2}
2D coarse grid resolution
single-scale
multi-scale
multiscale reconstructed

2D Error II

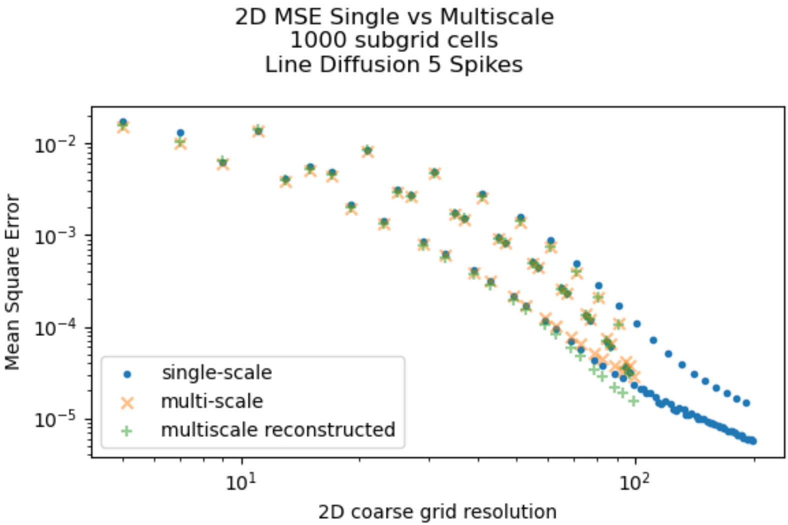

2D MSE Single vs Multiscale
1000 subgrid cells
Box Diffusion
Mean Square Error
10^{-1}
10^{-2}
10^{-3}
10^{1}
10^{2}
2D coarse grid resolution
single-scale
multi-scale
multiscale reconstructed

2D Error III



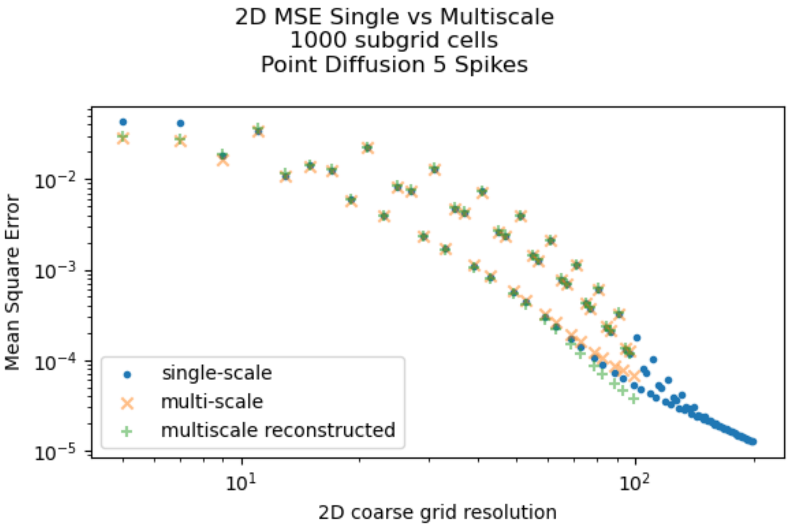
2D MSE Single vs Multiscale
1000 subgrid cells
Diamond Diffusion
Mean Square Error
10^{-2}
10^{-3}
10^{1}
10^{2}
2D coarse grid resolution
single-scale
multi-scale
multiscale reconstructed

2D Error IV



2D MSE Single vs Multiscale
1000 subgrid cells
Line Diffusion 5 Spikes
Mean Square Error
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{1}
10^{2}
2D coarse grid resolution
single-scale
multi-scale
multiscale reconstructed

2D Error V



2D MSE Single vs Multiscale
1000 subgrid cells
Point Diffusion 5 Spikes
Mean Square Error
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{1}
10^{2}
2D coarse grid resolution
single-scale
multi-scale
multiscale reconstructed