Numerical Techniques in Fluid Dynamics: Session 2

Steady-state Diffusion

Frederik Aerts
Turbulent Flow Simulation and Optimization Group
AY 25-26

1 Outline

1 Governing equations

Transport equation for passive scalar φ

$$\frac{\partial \varphi}{\partial t} + (\mathbf{u} \cdot \boldsymbol{\nabla})\varphi = \boldsymbol{\nabla} \cdot (\kappa \boldsymbol{\nabla} \varphi) + S_{\varphi}$$

1 Governing equations

Transport equation for passive scalar φ

$$\frac{\partial \varphi}{\partial t} + (\mathbf{u} \cdot \boldsymbol{\nabla})\varphi = \boldsymbol{\nabla} \cdot (\kappa \boldsymbol{\nabla} \varphi) + S_\varphi$$

Simplify

- steady state: $\frac{\partial \varphi}{\partial t} = 0$
- no advection: $(\mathbf{u} \cdot \boldsymbol{\nabla})\varphi = 0$
- uniform diffusivity κ [$\mathrm{m}^2\,\mathrm{s}^{-1}$]

$$-\boldsymbol{\nabla} \cdot (\kappa \boldsymbol{\nabla} \varphi) = S_\varphi$$

1 Derivation of discrete equations

Integrate governing equation over control volumes and apply Gauss divergence theorem:

$$-\boldsymbol{\nabla} \cdot (\kappa \boldsymbol{\nabla} \varphi) = S_\varphi$$

1 Derivation of discrete equations

Integrate governing equation over control volumes and apply Gauss divergence theorem:

$$-\boldsymbol{\nabla} \cdot (\kappa \boldsymbol{\nabla} \varphi) = S_\varphi$$

$$-\int_{CV} \boldsymbol{\nabla} \cdot (\kappa \boldsymbol{\nabla} \varphi) \mathrm{d}\Omega = \int_{CV} S_\varphi \mathrm{d}\Omega$$

1 Derivation of discrete equations

Integrate governing equation over control volumes and apply Gauss divergence theorem:

$$
\begin{aligned}
-\boldsymbol{\nabla} \cdot (\kappa \boldsymbol{\nabla} \varphi) &= S_\varphi \\
-\int_{CV} \boldsymbol{\nabla} \cdot (\kappa \boldsymbol{\nabla} \varphi) \mathrm{d}\Omega &= \int_{CV} S_\varphi \mathrm{d}\Omega \\
-\int_{\Gamma} \kappa \boldsymbol{\nabla} \varphi \cdot \mathbf{n} \mathrm{d}s &= \int_{CV} S_\varphi \mathrm{d}\Omega
\end{aligned}
$$

1 Derivation of discrete equations

Integrate governing equation over control volumes and apply Gauss divergence theorem:

$$
\begin{aligned}
-\boldsymbol{\nabla}\cdot(\kappa\boldsymbol{\nabla}\varphi) &= S_\varphi \\
-\int_{CV}\boldsymbol{\nabla}\cdot(\kappa\boldsymbol{\nabla}\varphi)\mathrm{d}\Omega &= \int_{CV} S_\varphi \mathrm{d}\Omega \\
-\int_{\Gamma}\kappa\boldsymbol{\nabla}\varphi\cdot\mathbf{n}\mathrm{d}s &= \int_{CV} S_\varphi \mathrm{d}\Omega \\
-\sum_{f=1}^{N_f}\kappa\left.\frac{\partial\varphi}{\partial n}\right|_f A_f &= S_\varphi\Omega
\end{aligned}
$$

1 Derivation of discrete equations

Integrate governing equation over control volumes and apply Gauss divergence theorem:

$$
\begin{aligned}
-\boldsymbol{\nabla} \cdot (\kappa \boldsymbol{\nabla} \varphi) &= S_\varphi \\
-\int_{CV} \boldsymbol{\nabla} \cdot (\kappa \boldsymbol{\nabla} \varphi) \mathrm{d}\Omega &= \int_{CV} S_\varphi \mathrm{d}\Omega \\
-\int_{\Gamma} \kappa \boldsymbol{\nabla} \varphi \cdot \mathbf{n} \mathrm{d}s &= \int_{CV} S_\varphi \mathrm{d}\Omega \\
-\sum_{f=1}^{N_f} \kappa \left.\frac{\partial \varphi}{\partial n}\right|_f A_f &= S_\varphi \Omega \\
-\sum_{f=1}^{N_f} \kappa \frac{\varphi_{NB1} - \varphi_{NB2}}{||\boldsymbol{\xi}_f||} A_f &= S_\varphi \Omega
\end{aligned}
$$

1 Derivation of discrete equations

For orthogonal case (cfr. blackboard):

$$-\sum_{f=1}^{N_f} \kappa \frac{\varphi_{NB1} - \varphi_{NB2}}{||\boldsymbol{\xi}_f||} A_f = S_\varphi \Omega$$

$$-\kappa \left\{ \frac{\varphi_E - \varphi_P}{||\boldsymbol{\xi}_e||} A_e + \frac{\varphi_N - \varphi_P}{||\boldsymbol{\xi}_n||} A_n + \frac{\varphi_W - \varphi_P}{||\boldsymbol{\xi}_w||} A_w + \frac{\varphi_S - \varphi_P}{||\boldsymbol{\xi}_s||} A_s \right\} = S_\varphi \Omega_P$$

1 Derivation of discrete equations

For orthogonal case (cfr. blackboard):

$$-\sum_{f=1}^{N_f} \kappa \frac{\varphi_{NB1} - \varphi_{NB2}}{||\boldsymbol{\xi}_f||} A_f = S_\varphi \Omega$$

$$-\kappa \left\{ \frac{\varphi_E - \varphi_P}{||\boldsymbol{\xi}_e||} A_e + \frac{\varphi_N - \varphi_P}{||\boldsymbol{\xi}_n||} A_n + \frac{\varphi_W - \varphi_P}{||\boldsymbol{\xi}_w||} A_w + \frac{\varphi_S - \varphi_P}{||\boldsymbol{\xi}_s||} A_s \right\} = S_\varphi \Omega_P$$

Simplified equation for cell P:

$$a_P \varphi_P + \sum_{NB} a_{NB} \varphi_{NB} = S_P \Omega_P$$

with

$$a_{NB} = -\kappa A_f / ||\boldsymbol{\xi}_f|| \qquad a_P = -\sum_{NB} a_{NB}$$

$\rightarrow$ Cast into linear system $\mathrm{A}\boldsymbol{\varphi} = \mathbf{b}$, solve for $\boldsymbol{\varphi}$.

1 Boundary conditions

Boundary conditions should be imposed on the **faces** between physical and ghost cells

2nd order PDE: 1 BC per boundary suffices

Dirichlet	Neumann
$\varphi_f = \varphi^*$	$\frac{\partial \varphi}{\partial n}\Big\rvert_f = \varphi'^*$
$\lambda \varphi_{PC} + (1-\lambda)\varphi_{GC} = \varphi^*$	$\pm \frac{\varphi_{GC} - \varphi_{PC}}{\lvert\lvert \boldsymbol{\xi}_f \rvert\rvert} = \varphi'^*$

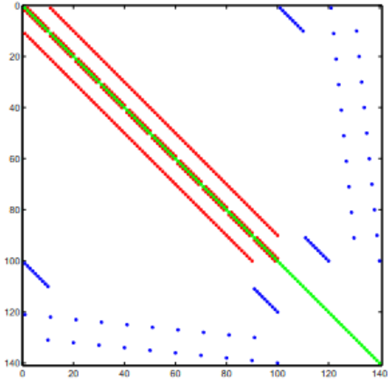
$\rightarrow$ leads to governing equation for φ in ghost cells.

2 Outline

2 FVMLab Matrix Storage

Matrix A from linear system $\mathrm{A}\boldsymbol{\varphi} = \mathbf{b}$ is very sparse (less than 3% non-zero entries)

- diagonal elements: a_P
- off-diagonal elements: a_{nb} (internal faces)
- off-diagonal elements: a_{nb} (boundary faces)

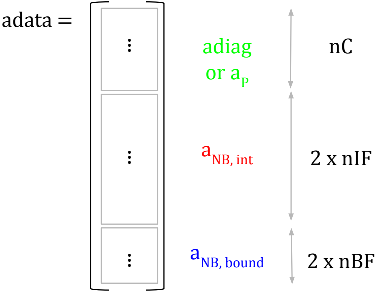


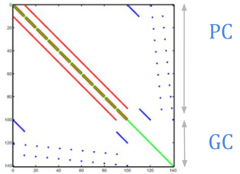
Stored in vector 'adata' (cfr. next slide & ScalarFvEqn2)

2 FVMLab Matrix Storage

Creation sparse matrix:

```
eqn = ScalarFvEqn2(dom);
eqn.adata = [adiag; anb_internal; anb_boundary];
eqn.bdata = bdata;
[A, b] = to_msparse(eqn);
```





$A[i,i] = adiag[i]$ $\quad i = 1..nC$

for nIF & nBF

$A[fNbC_1(i), fNbC_2(i)] = a_{NB}[2i-1]$ $\quad i = 1..nXF$

$A[fNbC_2(i), fNbC_1(i)] = a_{NB}[2i]$ $\quad i = 1..nXF$

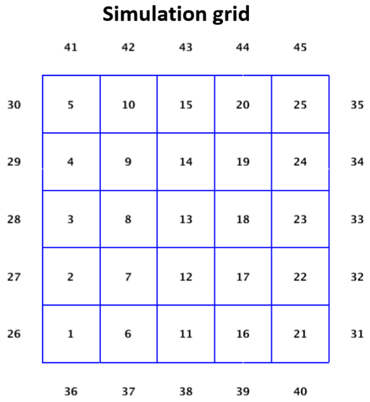
2 Program structure

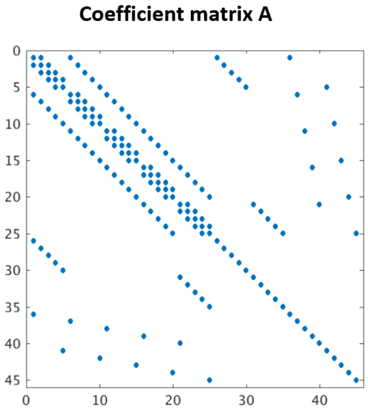
Initialize `adiag`, `bdata`, `anb_internal`, `anb_boundary`

- Loop internal faces
  - Add diagonal coefficients: 2 $\times$ PC
  - Add off-diagonal coefficients: 2 $\times$ PC
- Loop boundary faces
  - Add diagonal coefficients: 1 $\times$ PC (PDE) + 1 $\times$ GC (BC)
  - Add off-diagonal coefficients: 1 $\times$ PC (PDE) + 1 $\times$ GC (BC)

Tip: Use `spy` command to observe matrix structure!

2 Example: coefficient matrix for a 5 $\times$ 5 mesh





2 Boundary condition implementation

`casedef` structure: holds all information defining simulation case

- `casedef.BC` substructure: boundary conditions
  - 4 boundary conditions: `casedef.BC{1}`, ..., `casedef.BC{4}`
  - `casedef.BC{i}.zoneID`: name, e.g. 'Oostrand'
  - `casedef.BC{i}.kind`: type, 'Dirichlet' or 'Neumann'
  - `casedef.BC{i}.data`: value, φ^* or φ'^*
- `casedef.dom` substructure: domain
  - function `dom.getzone(casedef.BC{i}.zoneID)` $\rightarrow$ `BfaceZone` type
    - `elcount`: #faces for this BC
    - `range`: starting and stopping face for this BC

2 Boundary condition implementation

`casedef` structure: holds all information defining simulation case

- `casedef.BC` substructure: boundary conditions
  - 4 boundary conditions: `casedef.BC{1}`, ..., `casedef.BC{4}`
  - `casedef.BC{i}.zoneID`: name, e.g. 'Oostrand'
  - `casedef.BC{i}.kind`: type, 'Dirichlet' or 'Neumann'
  - `casedef.BC{i}.data`: value, φ^* or φ'^*
- `casedef.dom` substructure: domain
  - function `dom.getzone(casedef.BC{i}.zoneID)` $\rightarrow$ `BfaceZone` type
    - `elcount`: #faces for this BC
    - `range`: starting and stopping face for this BC

$\rightarrow$ For each BC in `casedef.BC`

1. get the ID (name)
2. get the zone
3. get the range

2 Mesh generation

Command to generate one-directional mesh vector:

```
seedX =
LineSeed.lineSeedOneWayBias(originV,displV,nParts,expFac,'o');
```

- `originV`: 2-D position of origin point of mesh vector (e.g. [0, 0])
- `displV`: 2-D position of ending point defining mesh vector (e.g. [1, 0])
- `nParts`: number of grid cells along mesh vector (e.g. 10)
- `expFac`: relative length of next cell to previous cell, $\frac{\Delta x_{i+1}}{\Delta x_i}$ (e.g. 1.10)

$\rightarrow$ Allows simulation of Cartesian grids, rotated grids, stretched grids, skewed grids, ... (try it yourself!)

3 Outline

3 Homework

Finish implementation of diffusion equation for next week.

- Dirichlet + Neumann boundary conditions
- Verify implementation

Next week's topic: advection - diffusion equation with known constant advection $\mathbf{u}$

$$(\mathbf{u} \cdot \boldsymbol{\nabla})\varphi = \boldsymbol{\nabla} \cdot (\kappa \boldsymbol{\nabla} \varphi) + S_\varphi$$