Solving the linear system

Numerical Flow Simulation

Edouard Boujo

Fall 2022

École
polytechnique
fédérale
de Lausanne

Linear system

- After discretization (and linearization if needed), the conservation equation(s) yield a linear system to be solved:

$$\mathbf{A}\boldsymbol{\phi} = \mathbf{b}$$

$\mathbf{A}$: *n* x *n* matrix

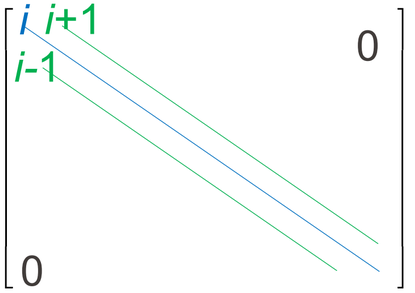
$\boldsymbol{\phi}, \mathbf{b}$: *n* x 1 vectors

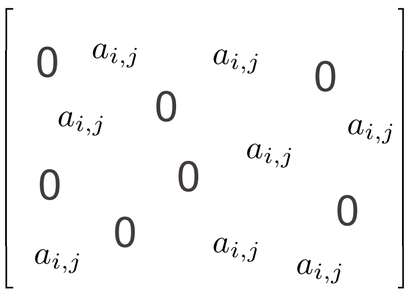
$a_{i,j} \neq 0$ if ϕ_i depends on ϕ_j

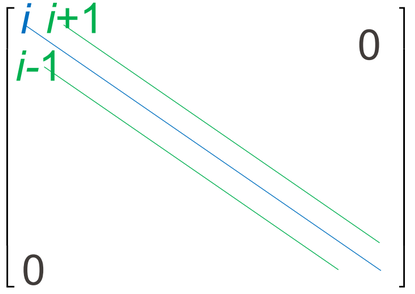
- The size *n* depends on:
  - the mesh: number of control volumes
  - the physical dimension: 1D/2D/3D
  - the equations: single vs. coupled (1 unknown vs. several unknowns)

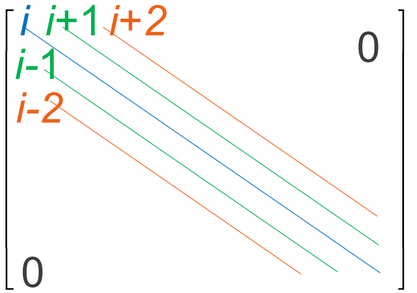
Linear system

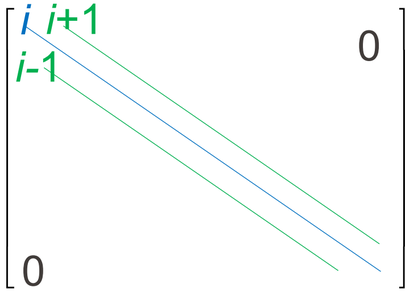
- The structure of the matrix $\mathbf{A}$ depends on:
  - The mesh: structured vs. unstructured
  - The equations that are solved (the order of the derivatives)
  - The discretization method: different schemes involve a different number of neighbors (e.g. UD vs. CD vs. QUICK, see week 2)
  - The physical dimension: 1D vs. 2D vs. 3D
- Solution methods depend on this structure.

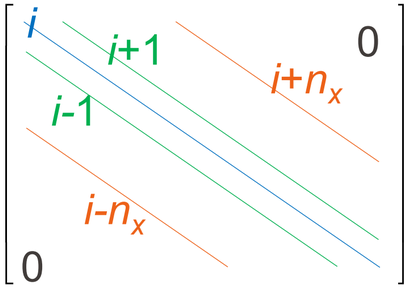












Direct vs. iterative methods

- Direct methods:
  - Build a series of equivalent **exact** systems
  - Compute the solution in a **finite** number of operations
  - Overall complexity generally $O(n^3)$

- Iterative methods:
  - Build a series of **approximate** solutions
  - May converge after an **infinite** number of steps
  - Complexity at each step generally $O(n^2)$
    → useful if practical number of steps is $< n$

Direct methods: Gaussian elimination

$$\mathbf{A}\boldsymbol{\phi} = \mathbf{b} \qquad \begin{cases} a_{1,1}\phi_1 + a_{1,2}\phi_2 + \ldots + a_{1,n}\phi_n = b_1 \\ a_{2,1}\phi_1 + a_{2,2}\phi_2 + \ldots + a_{2,n}\phi_n = b_2 \\ \vdots \\ a_{n,1}\phi_1 + a_{n,2}\phi_2 + \ldots + a_{n,n}\phi_n = b_n \end{cases}$$

- Naively:
  - Use eq. 1 to express ϕ_1 as function of $\phi_2 \ldots \phi_n$
  - Substitute in other eqs. (2… *n)*
  - Repeat with $\phi_2, \phi_3 \ldots$ until ϕ_n is known explicitly
  - Substitute back into eq. *n*-1 to get ϕ_{n-1}, eq. *n*-2 to get ϕ_{n-2},… eq. 1 to get ϕ_1

- Systematically: find suitable linear combinations of rows to eliminate variables successively
  - Step 1: “forward elimination” to make the system triangular
  - Step 2: “backward substitution” to solve the triangular system

Direct methods: Gaussian elimination

$$\mathbf{A}\boldsymbol{\phi} = \mathbf{b} \qquad \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n} \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

- Step 1: “forward elimination”
  1. Replace each row *i*>1 of $\mathbf{A}$ by $(\text{row } i) - \dfrac{a_{i,1}}{a_{1,1}}(\text{row } 1)$

  i.e. define new coefficients and rhs: $a_{i,j}^{(1)} = a_{i,j} - \dfrac{a_{i,1}}{a_{1,1}} a_{1,j}$ $\qquad b_i^{(1)} = b_i - \dfrac{a_{i,1}}{a_{1,1}} b_1$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ 0 & a_{2,2} - \frac{a_{2,1}}{a_{1,1}} a_{1,2} & a_{2,3} - \frac{a_{2,1}}{a_{1,1}} a_{1,3} & \cdots & a_{2,n} - \frac{a_{2,1}}{a_{1,1}} a_{1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & a_{n,2} - \frac{a_{n,1}}{a_{1,1}} a_{1,2} & a_{n,3} - \frac{a_{n,1}}{a_{1,1}} a_{1,3} & \cdots & a_{n,n} - \frac{a_{n,1}}{a_{1,1}} a_{1,n} \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 - \frac{a_{n,1}}{a_{1,1}} b_1 \\ \vdots \\ b_n - \frac{a_{2,1}}{a_{1,1}} b_1 \end{pmatrix}$$

$$\mathbf{A}^{(1)}\boldsymbol{\phi} = \mathbf{b}^{(1)}$$

Direct methods: Gaussian elimination

$$\mathbf{A}^{(1)}\boldsymbol{\phi} = \mathbf{b}^{(1)} \qquad \begin{bmatrix} a_{1,1}^{(1)} & a_{1,2}^{(1)} & \cdots & a_{1,n}^{(1)} \\ 0 & a_{2,2}^{(1)} & \cdots & a_{2,n}^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & a_{n,2}^{(1)} & \cdots & a_{n,n}^{(1)} \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{pmatrix} = \begin{pmatrix} b_1^{(2)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{pmatrix}$$

- Step 1: “forward elimination”
  2. Replace each row *i*>2 of $\mathbf{A}^{(1)}$ by $(\text{row } i) - \dfrac{a_{i,2}^{(1)}}{a_{2,2}^{(1)}}(\text{row } 2)$

  i.e. define new coefficients and rhs: $a_{i,j}^{(2)} = a_{i,j}^{(1)} - \dfrac{a_{i,2}^{(1)}}{a_{2,2}^{(1)}} a_{2,j}^{(1)} \qquad b_i^{(2)} = b_i^{(1)} - \dfrac{a_{i,2}^{(1)}}{a_{2,2}^{(1)}} b_2^{(1)}$

$$\begin{bmatrix} a_{1,1}^{(2)} & a_{1,2}^{(2)} & a_{1,3}^{(2)} & \cdots & a_{1,n}^{(2)} \\ 0 & a_{2,2}^{(2)} & a_{2,3}^{(2)} & \cdots & a_{2,n}^{(2)} \\ \vdots & 0 & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & a_{n,3}^{(2)} & \cdots & a_{n,n}^{(2)} \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{pmatrix} = \begin{pmatrix} b_1^{(2)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{pmatrix}$$

$$\mathbf{A}^{(2)}\boldsymbol{\phi} = \mathbf{b}^{(2)}$$

Direct methods: Gaussian elimination

$$\mathbf{A}^{(k-1)}\boldsymbol{\phi} = \mathbf{b}^{(k-1)}$$

- Step 1: “forward elimination”

  k. Replace each row *i*>k of $\mathbf{A}^{(k-1)}$ by $(\text{row } i) - \dfrac{a_{i,k}^{(k-1)}}{a_{k,k}^{(k-1)}}(\text{row } k)$

  i.e. define: $a_{i,j}^{(k)} = a_{i,j}^{(k-1)} - \dfrac{a_{i,k}^{(k-1)}}{a_{k,k}^{(k-1)}} a_{k,j}^{(k-1)} \qquad b_i^{(k)} = b_i^{(k-1)} - \dfrac{a_{i,k}^{(k-1)}}{a_{k,k}^{(k-1)}} b_k^{(k-1)}$

  $$\mathbf{A}^{(k)}\boldsymbol{\phi} = \mathbf{b}^{(k)}$$

  …

  …

  n-1. Finally:

$$\begin{bmatrix} a_{1,1}^{(n-1)} & a_{1,2}^{(n-1)} & a_{1,3}^{(n-1)} & \cdots & a_{1,n}^{(n-1)} \\ 0 & a_{2,2}^{(n-1)} & a_{2,3}^{(n-1)} & \cdots & a_{2,n}^{(n-1)} \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & a_{n,n}^{(n-1)} \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{pmatrix} = \begin{pmatrix} b_1^{(n-1)} \\ b_2^{(n-1)} \\ \vdots \\ b_n^{(n-1)} \end{pmatrix}$$

$$\mathbf{A}^{(n-1)}\boldsymbol{\phi} = \mathbf{b}^{(n-1)}$$

Direct methods: Gaussian elimination

$$\mathbf{A}^{(n-1)}\boldsymbol{\phi} = \mathbf{b}^{(n-1)} \qquad \begin{bmatrix} a_{1,1}^{(n-1)} & a_{1,2}^{(n-1)} & a_{1,3}^{(n-1)} & \cdots & a_{1,n}^{(n-1)} \\ 0 & a_{2,2}^{(n-1)} & a_{2,3}^{(n-1)} & \cdots & a_{2,n}^{(n-1)} \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & a_{n,n}^{(n-1)} \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{pmatrix} = \begin{pmatrix} b_1^{(n-1)} \\ b_2^{(n-1)} \\ \vdots \\ b_n^{(n-1)} \end{pmatrix}$$

- Step 2: "backward substitution" to solve the triangular system
  - Row *n*: $\phi_n = \dfrac{b_n}{a_{n,n}}$ (omit superscript $^{(n-1)}$)
  - Row *n*-1: $\phi_{n-1} = \dfrac{b_{n-1} - a_{n-1,n}\phi_n}{a_{n-1,n-1}}$
  - Row *i*: $\phi_i = \dfrac{1}{a_{i,i}}\left(b_i - \sum_{k=i+1}^{n} a_{i,k}\phi_k\right)$
- Complexity $O(n^3)$ → not practical for large systems.

Direct methods: TDMA algorithm

- “TriDiagonal Matrix Algorithm” (also called Thomas algorithm)
- Particular case of Gaussian elimination for **tridiagonal** systems (suitable for 1D problems)

$$\begin{bmatrix} a_{1,1} & a_{1,2} & 0 & \cdots & 0 \\ a_{2,1} & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & a_{n-1} \\ 0 & \cdots & 0 & a_{n,n-1} & a_{n,n} \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{n-1} \\ \phi_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}$$

- Complexity $O(n)$

Direct methods: TDMA algorithm

$$\begin{bmatrix} a_{1,1} & a_{1,2} & 0 & \cdots & 0 \\ a_{2,1} & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & a_{n-1} \\ 0 & \cdots & 0 & a_{n,n-1} & a_{n,n} \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{n-1} \\ \phi_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}$$

- Step 1: forward elimination
  - Express ϕ_1 as fct of ϕ_2: $\phi_1 = \dfrac{b_1 - a_{1,2}\phi_2}{a_{1,1}} = P_1\phi_2 + Q_1$

$$P_1 = -\frac{a_{1,2}}{a_{1,1}} \qquad Q_1 = \frac{b_1}{a_{1,1}}$$

  - Express ϕ_2 as fct of ϕ_3: $a_{2,2}\phi_2 = b_2 - a_{2,1}\phi_1 - a_{2,3}\phi_3$

$$\phi_2 = \frac{b_2 - a_{2,1}Q_1 - a_{2,3}\phi_3}{a_{2,2} + a_{2,1}P_1} = P_2\phi_3 + Q_2$$

$$P_2 = \frac{-a_{2,3}}{a_{2,2} + a_{2,1}P_1} \qquad Q_2 = \frac{b_2 - a_{2,1}Q_1}{a_{2,2} + a_{2,1}P_1}$$

  - Express ϕ_i: $\phi_i = P_i\phi_{i+1} + Q_i$

$$P_i = \frac{-a_{i,i+1}}{a_{i,i} + a_{i,i-1}P_{i-1}} \qquad Q_i = \frac{b_i - a_{i,i-1}Q_{i-1}}{a_{i,i} + a_{i,i-1}P_{i-1}}$$

  - Obtain finally: $\phi_n = Q_n = \dfrac{b_n - a_{n,n-1}Q_{n-1}}{a_{n,n} + a_{n,n-1}P_{n-1}}$

Direct methods: TDMA algorithm

$$\begin{bmatrix} a_{1,1} & a_{1,2} & 0 & \cdots & 0 \\ a_{2,1} & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & a_{n-1} \\ 0 & \cdots & 0 & a_{n,n-1} & a_{n,n} \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{n-1} \\ \phi_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}$$

- Step 2: backward substitution

$$\begin{aligned} \phi_n &= Q_n \\ \phi_{n-1} &= P_{n-1}\phi_n + Q_{n-1} \\ &\cdots \\ \phi_i &= P_i\phi_{i+1} + Q_i \\ &\cdots \\ \phi_1 &= P_1\phi_2 + Q_1 \end{aligned}$$

- Algorithm:
  - Compute P_1, Q_1, then P_2, Q_2, etc.
  - Set $\phi_n = Q_n$
  - Compute ϕ_{n-1}, then ϕ_{n-2}, etc.

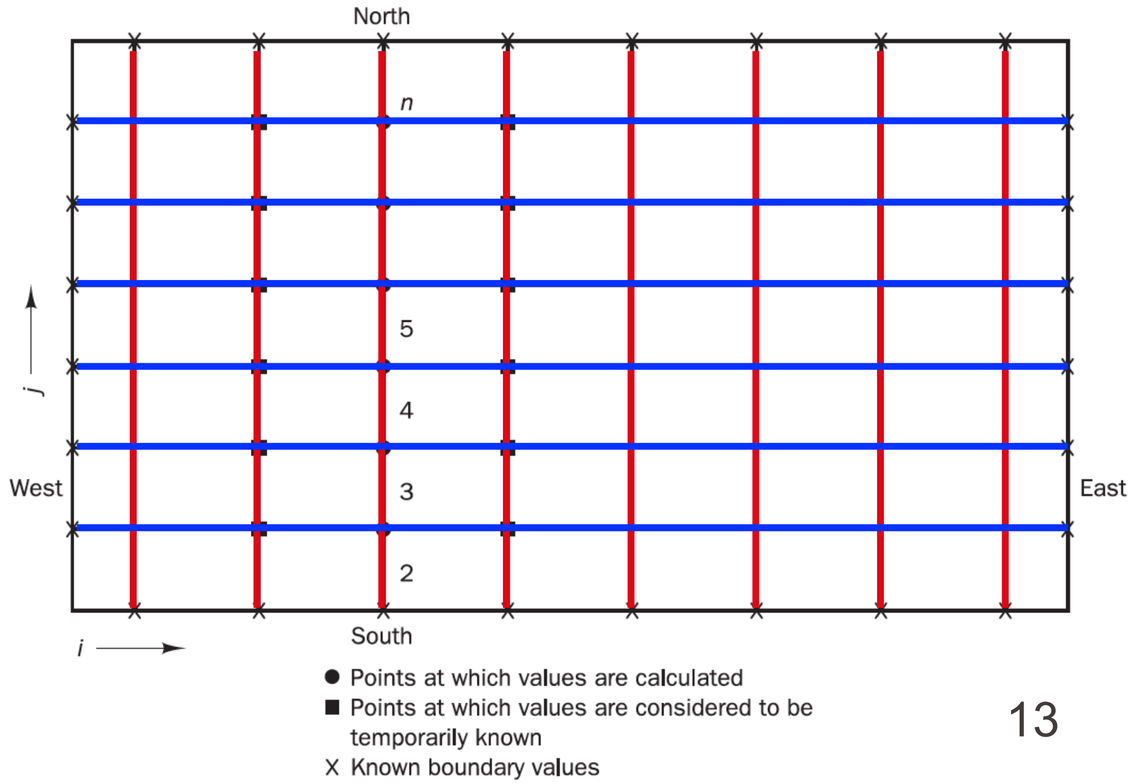
TDMA extended to 2D/3D problems

- TDMA can be applied iteratively to 2D/3D structured meshes
- First applied in 1D along one line, e.g. in the North-South direction:

rewrite $a_P\phi_P = a_W\phi_W + a_E\phi_E + a_S\phi_S + a_N\phi_N + b$

as $a_P\phi_P = a_S\phi_S + a_N\phi_N + (b + a_W\phi_W^* + a_E\phi_E^*)$ with guess values for W, E

- Repeat for all N-S lines, sweeping the domain from W to E.
- Repeat for all W-E lines, sweeping the domain from S to N.
- Iterate, alternating the line/sweep directions to improve the propagation of boundary conditions within the domain.



Iterative methods

- Build a series of approximate solutions $\boldsymbol{\phi}^{(k)}$ to the actual system $\mathbf{A}\boldsymbol{\phi} = \mathbf{b}$
- Error vector: $\mathbf{e}^{(k)} = \boldsymbol{\phi} - \boldsymbol{\phi}^{(k)}$ (but generally the true solution is unknown)
- Residual vector: $\mathbf{r}^{(k)} = \mathbf{A}\boldsymbol{\phi}^{(k)} - \mathbf{b}$ (useful: can actually be computed)
- Convergence: $\lim_{k\to\infty} ||\mathbf{e}^{(k)}|| = \mathbf{0}$ $\lim_{k\to\infty} ||\mathbf{r}^{(k)}|| = 0$
- Convergence can be assessed with different criteria:
  - Absolute or relative iteration error: $||\boldsymbol{\phi}^{(k)} - \boldsymbol{\phi}^{(k-1)}||$ or $\dfrac{||\boldsymbol{\phi}^{(k)} - \boldsymbol{\phi}^{(k-1)}||}{||\boldsymbol{\phi}^{(k-1)}||} < \text{tol}$
  - Absolute residual: $||\mathbf{r}^{(k)}|| < \text{tol}$
  - Normalized residual (most common): $\dfrac{||\mathbf{r}^{(k)}||}{||\text{diag}(\mathbf{A})\boldsymbol{\phi}^{(k)}||}$ or $\dfrac{||\mathbf{r}^{(k)}||}{||\mathbf{r}^{(k_0)}||} < \text{tol}$

$2 \leq k_0 \lesssim 10$

Typically use L^1 norm $||\mathbf{x}|| = \sum_{i=1}^{n} |x_i|$ or $||\mathbf{x}|| = \frac{1}{n}\sum_{i=1}^{n} |x_i|$

Point-iterative methods

- At each iteration, solve each equation *i* for ϕ_i, using guess values ϕ_j^* for the other unknows:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n} \end{bmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad \begin{matrix} \rightarrow \\ \rightarrow \\ \cdots \\ \rightarrow \\ \cdots \end{matrix} \quad \begin{matrix} \phi_1 = \dfrac{1}{a_{1,1}} \left(b_1 - \sum_{j \neq 1} a_{1,j} \boxed{\phi_j^*} \right) \\ \phi_2 = \cdots \\ \\ \phi_i = \dfrac{1}{a_{i,i}} \left(b_i - \sum_{j \neq i} a_{i,j} \boxed{\phi_j^*} \right) \end{matrix}$$

- **Jacobi** method: guess value taken from previous iteration.

$$\phi_i^{(k)} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j \neq i} a_{i,j} \boxed{\phi_j^{(k-1)}} \right)$$

Point-iterative methods

- **Gauss-Seidel** method: guess value taken from previous or current iteration, using the most recently updated one.

$$\phi_i^{(k)} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j<i} a_{i,j} \phi_j^{(k)} - \sum_{j>i} a_{i,j} \phi_j^{(k-1)} \right)$$

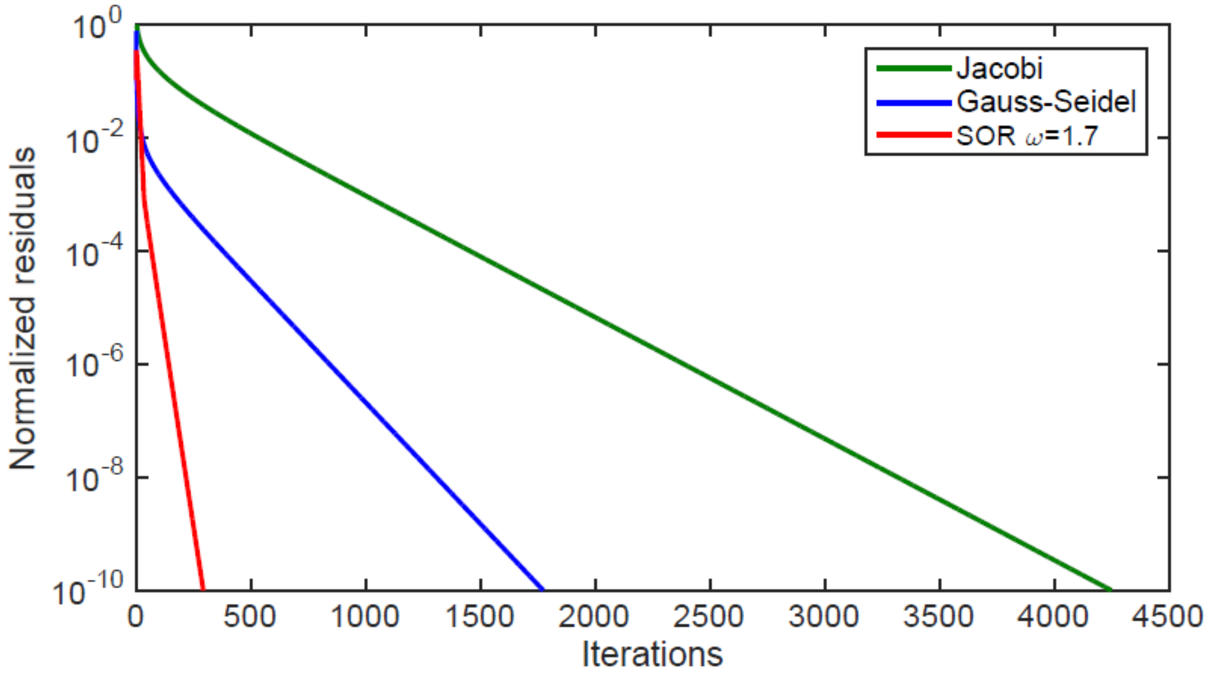
- Both Jacobi and GS require $O(n^2)$ iterations. GS often faster than Jacobi.
- **Successive Over-Relaxation (SOR)** method:

$$\phi_i^{(k)} = (1-\omega)\phi_i^{(k-1)} + \frac{\omega}{a_{i,i}} \left(b_i - \sum_{j<i} a_{i,j} \phi_j^{(k)} - \sum_{j>i} a_{i,j} \phi_j^{(k-1)} \right) \qquad 0 < \omega < 2$$

  - $\omega > 1$: can speed up convergence
  - $\omega < 1$: can stabilize the iterative procedure if needed
  - Optimal value of ω is case-dependent.

Point-iterative methods

- Example: 1D steady diffusion, Dirichlet BCs, n=33 (see week 2)

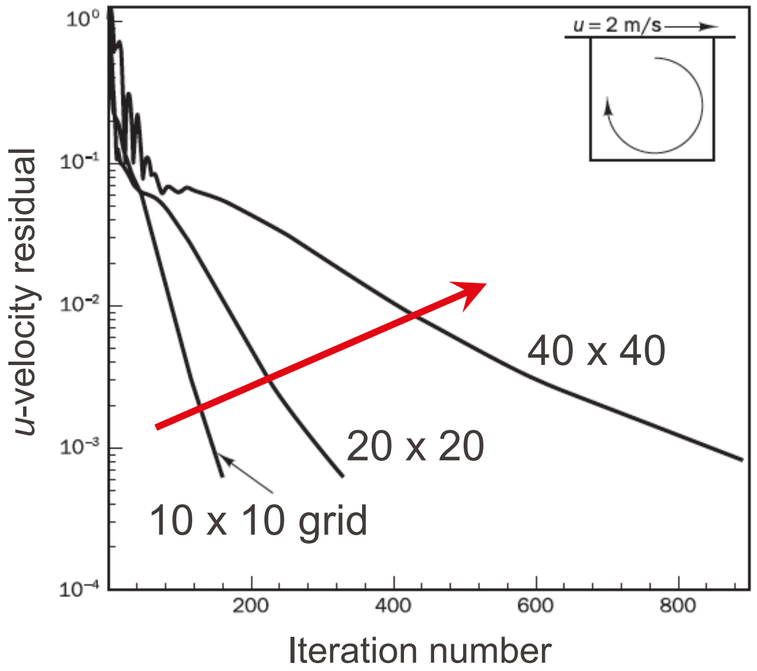


- Still need hundreds of iterations, for a small number of unknows… Can we do even faster?

Multigrid methods

Two observations about iterative methods:

1. Finer mesh: **more accurate** solution (smaller final error), but **slower convergence** (more iterations needed to decrease the residuals, and each iteration is more expensive).

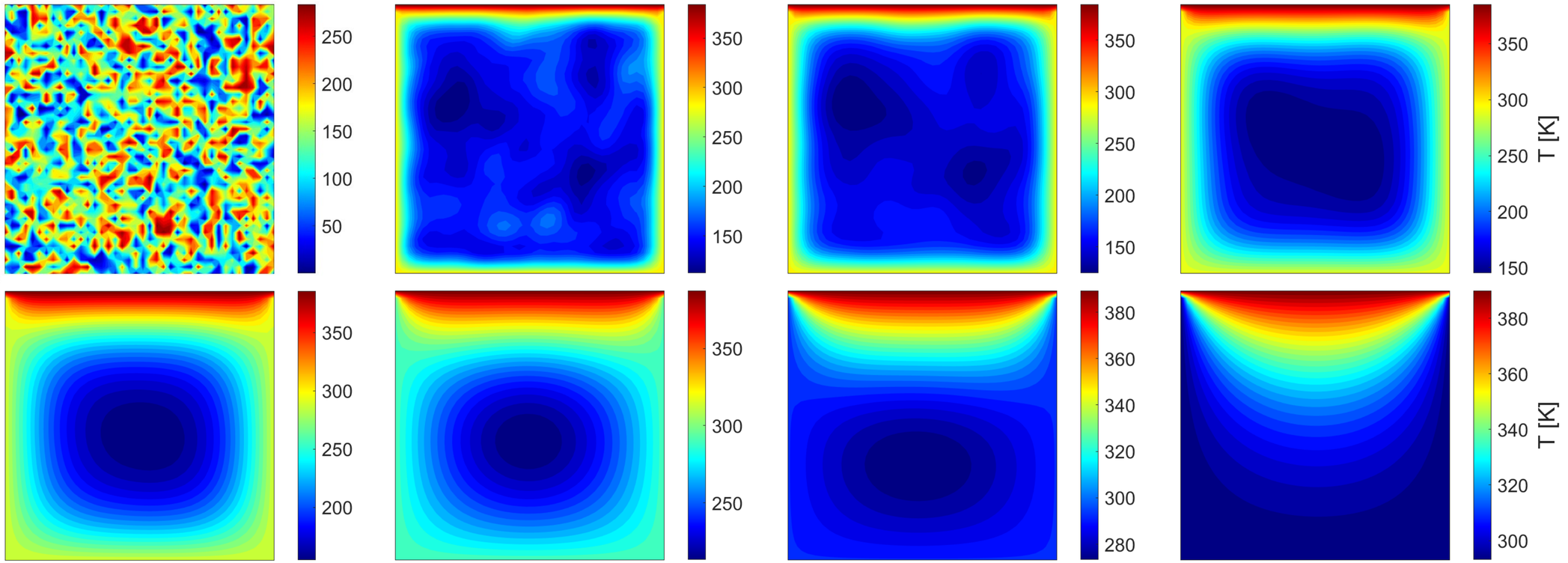


Example: flow in a lid-driven cavity

Multigrid methods

Two observations about iterative methods:

2. Consider this example (2D steady diffusion; see week 2). What do we observe?



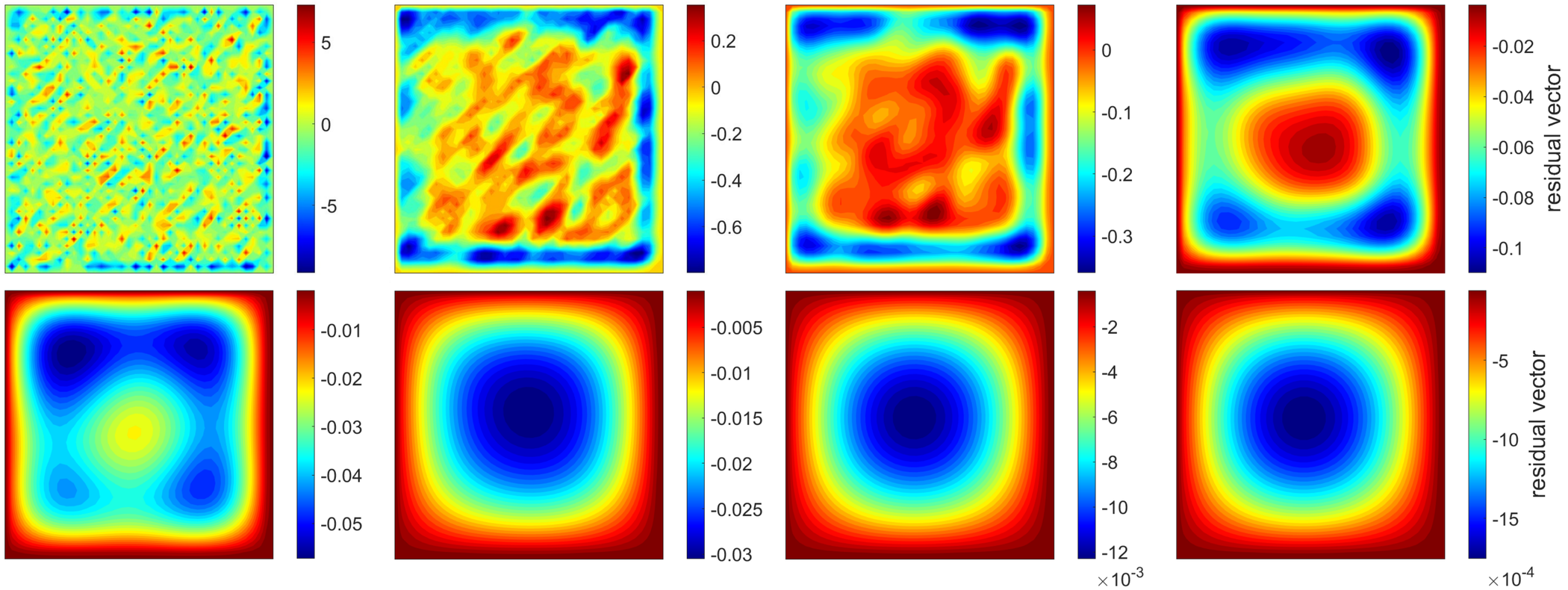
Approximate solution $T^{(k)}$

(40 x 40 grid, iterations k=0, 5, 10, 30, 60, 150, 300, 600)

Multigrid methods

Two observations about iterative methods:

2. Consider this example (2D steady diffusion; see week 2). What do we observe?



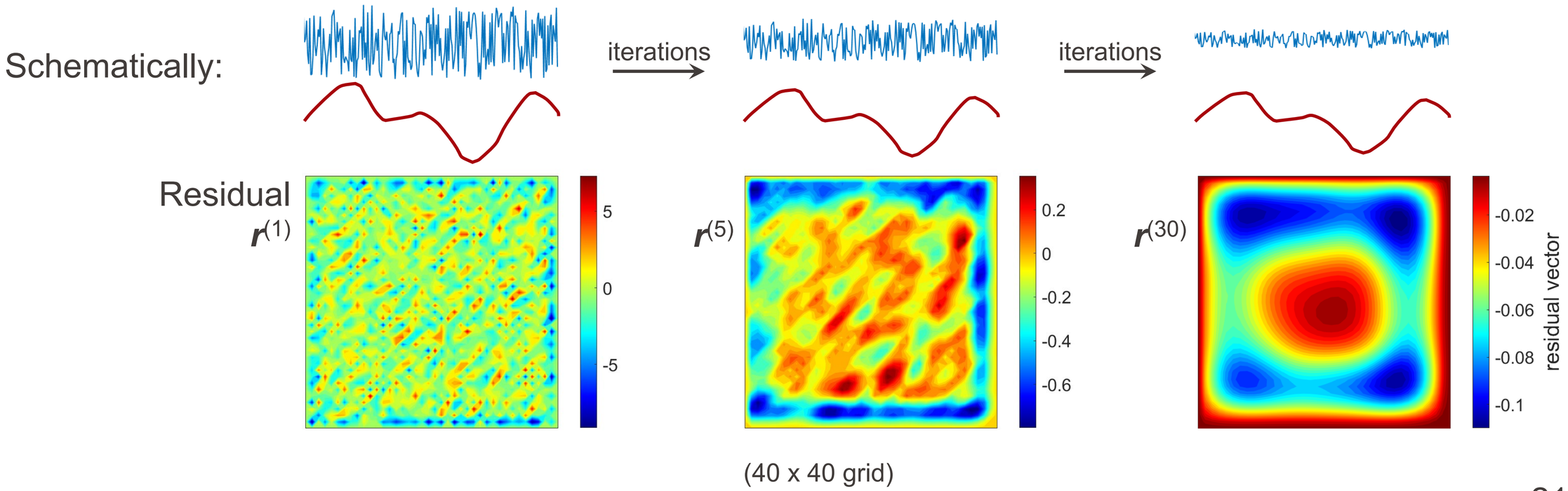
Residual vector $\boldsymbol{r}^{(k)}=\boldsymbol{A}\boldsymbol{T}^{(k)}-\boldsymbol{b}$

(40 x 40 grid, iterations k=1, 5, 10, 30, 60, 150, 300, 600)

Multigrid methods

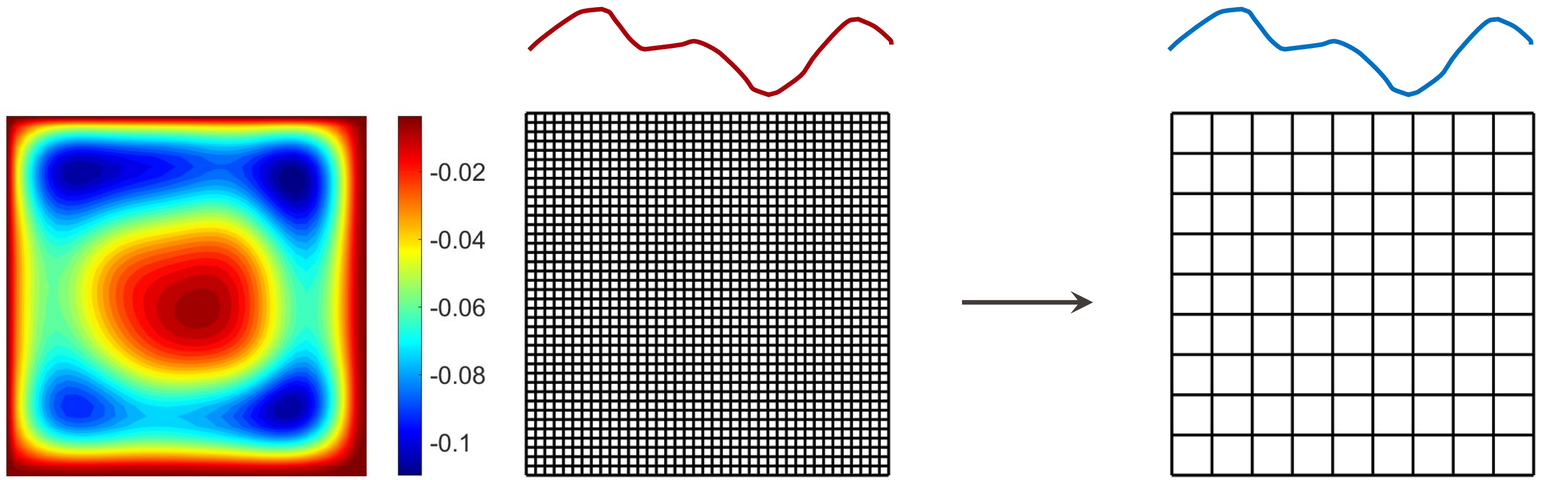
Two observations about iterative methods:

2. These methods only have a **local effect**: they quickly reduce the short-wavelength components of the error (wavelengths comparable to the mesh size), but they act slowly on long wavelengths.



Multigrid methods

- If we change to a **coarser** grid, the long-wavelength error becomes a "short" wavelength (comparable to the mesh size). *"Smooth becomes rough".*



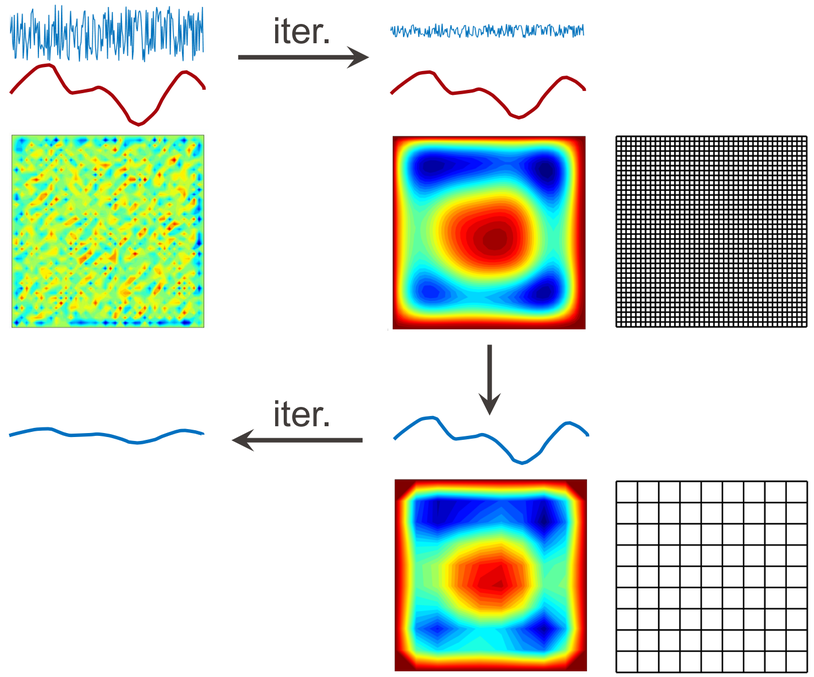
- Now, on this coarser grid, the short-wavelength error can be reduced very quickly.

Multigrid methods

- **Multigrid idea**: combine iterations on meshes of different sizes. Schematically:

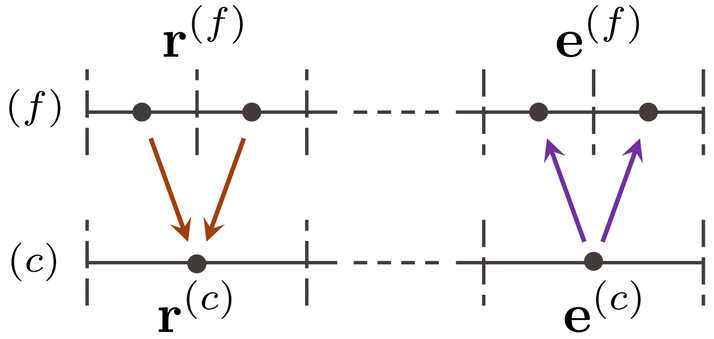
  - Initial solution on original **fine** mesh
    → quickly reduce short-wavelength error.

  - Propagate on **coarse** mesh → quickly reduce the original long-wavelength error (which is now a short wavelength).



  - Come back to original **fine** mesh → improve final accuracy.

Multigrid methods



System to be solved (on fine mesh): $\mathbf{A}^{(f)}\boldsymbol{\phi} = \mathbf{b}$

- Initial fine-mesh iterations:
  - A few steps only (e.g. GS) → approximate solution $\boldsymbol{\phi}^{(f)}$
  - Compute the residual from $\mathbf{A}^{(f)}\boldsymbol{\phi}^{(f)} = \mathbf{b} - \mathbf{r}^{(f)}$
  - Note that the error $\mathbf{e}^{(f)} = \boldsymbol{\phi} - \boldsymbol{\phi}^{(f)}$ satisfies $\mathbf{A}^{(f)}\mathbf{e}^{(f)} = \mathbf{r}^{(f)}$

- Restriction + coarse-mesh iterations:
  - Transfer the residual to the coarser mesh (interpolation): $\mathbf{r}^{(c)}$
  - Solve for the error, which satisfies $\mathbf{A}^{(c)}\mathbf{e}^{(c)} = \mathbf{r}^{(c)}$

- Prolongation:
  - Transfer back the error to the finer mesh (interpolation): $\mathbf{e}^{(f)}$

- Update $\boldsymbol{\phi}^{(f)}_{new} = \boldsymbol{\phi}^{(f)} + \mathbf{e}^{(f)}$; perform a few fine-mesh iterations $\mathbf{A}^{(f)}\boldsymbol{\phi} = \mathbf{b}$
- Repeat the whole process until convergence.

Multigrid methods

- Examples of restriction/prolongation operators on uniform 1D meshes:

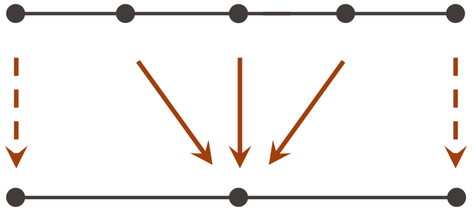
5-CV and 9-CV meshes

$$R^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

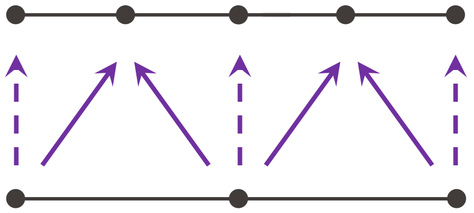
$$T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

3-CV and 5-CV meshes

$$R^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$



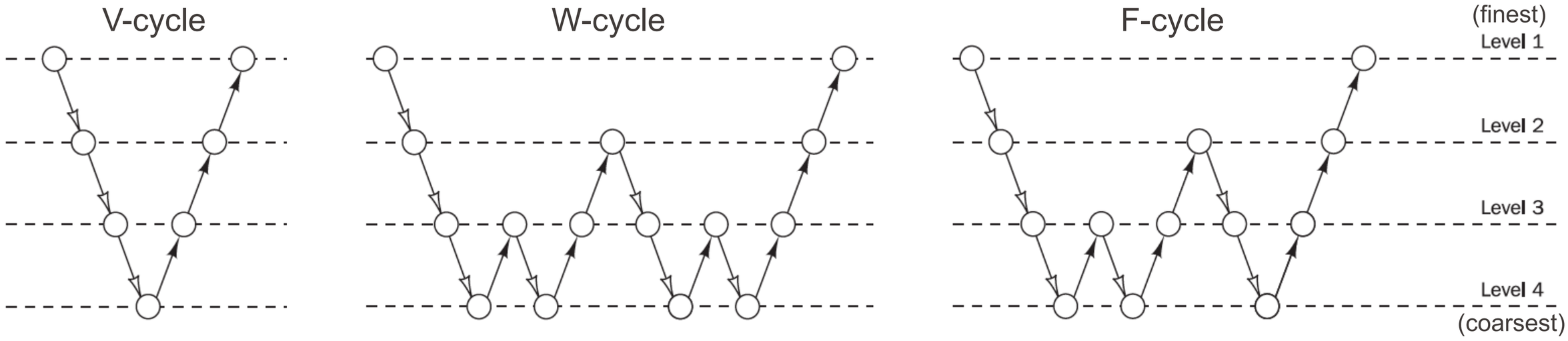
$$T^{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{pmatrix}$$



Multigrid methods

- Can use more than 2 meshes:
  - At each level *i*, solve for the error of the equation solved at level *i*-1.
  - Use only a few iterations, except on the coarsest mesh (full solution).

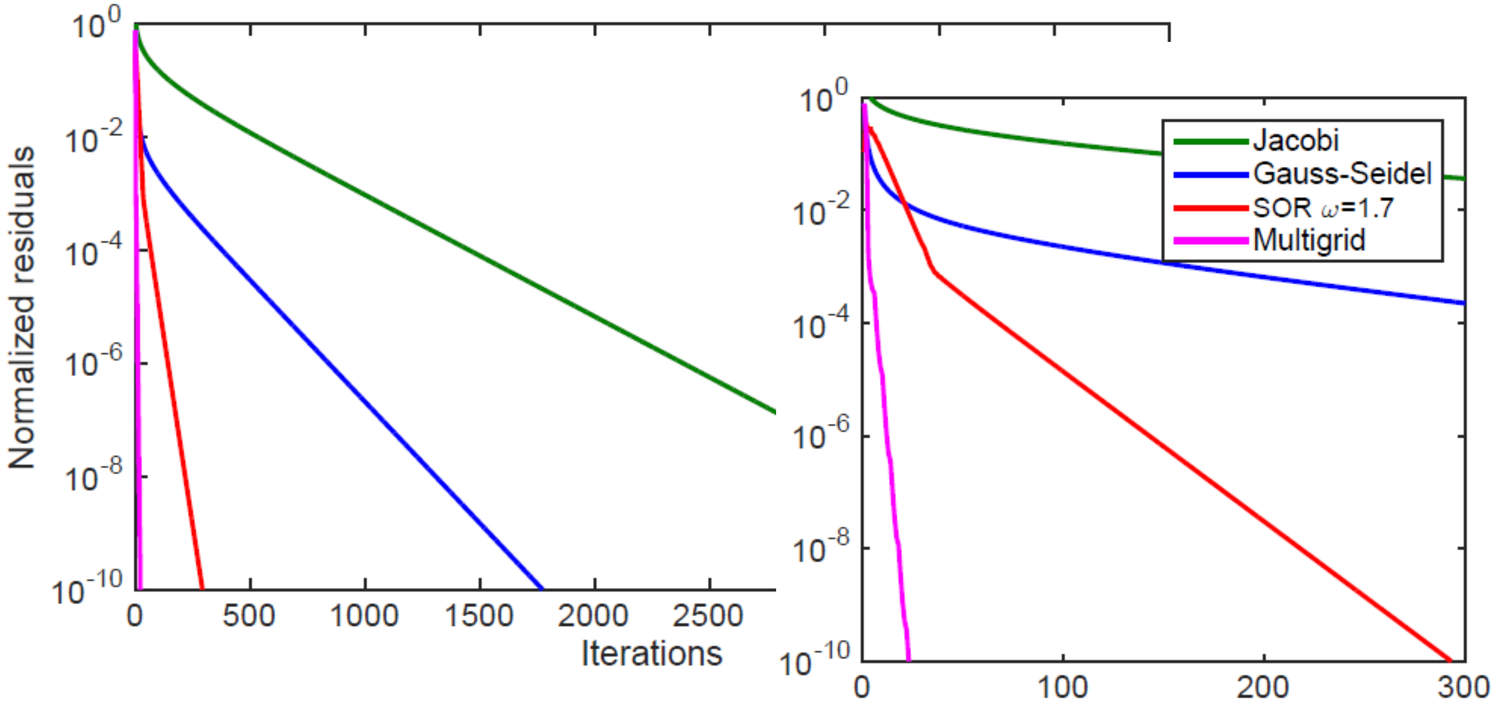
- Can use different types of cycles:



- Coarse-grid iterations are comparatively very cheap.

Multigrid methods

- Example: 1D steady diffusion, Dirichlet BCs, n=33 (see week 2)



- Multigrid methods require much less fine-mesh iterations than single-grid iterative methods. Widely used in CFD, including in Fluent.

Multigrid methods

- In Fluent: