AMS526: Numerical Analysis I (Numerical Linear Algebra for Computational and Data Sciences)

Lecture 25: Overview of Multigrid Methods

Xiangmin Jiao

SUNY Stony Brook

Outline

Stationary Iterative Methods

- Stationary iterative methods can be interpreted as a fixed point iteration obtained by matrix splitting.
- Let $A = M - N$ and $r_k = b - Ax_k$ we can obtain

$$x_{k+1} = M^{-1}Nx_k + M^{-1}b \tag{1}$$

$$x_{k+1} = x_k + M^{-1}r_k \tag{2}$$

- Different choices of splitting lead to various schemes

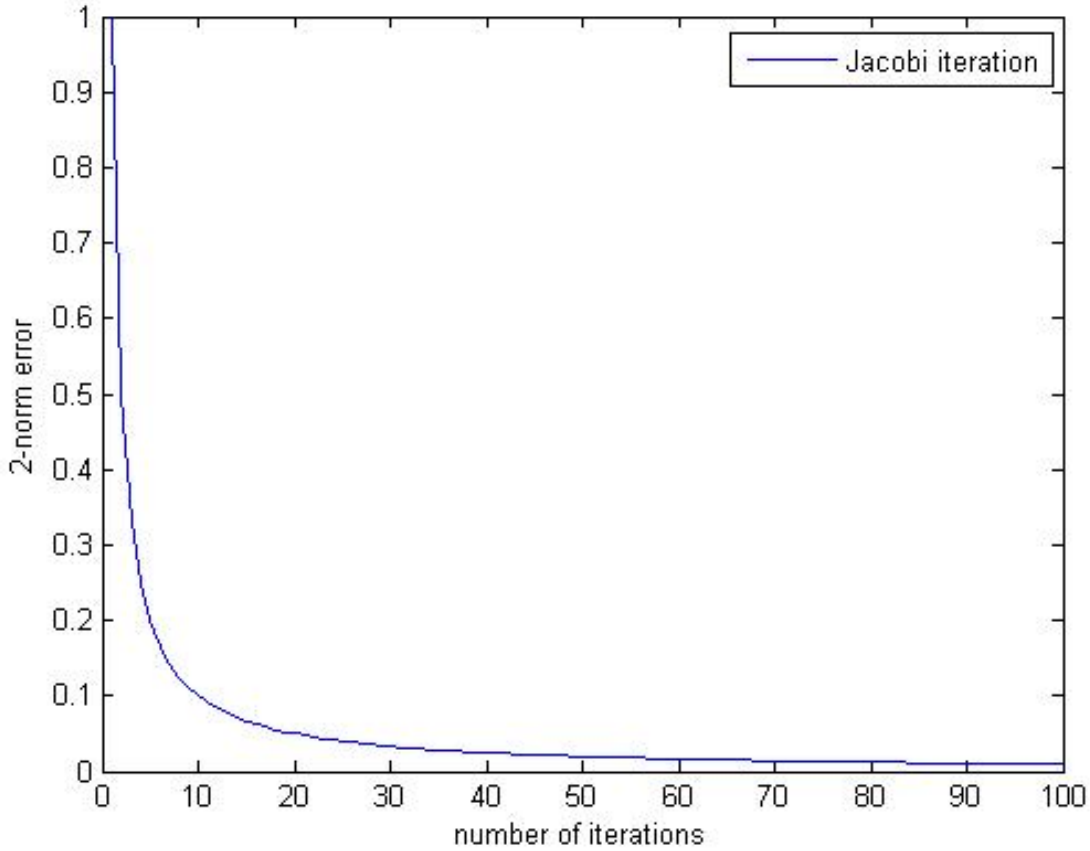
Stationary Iterative Methods

- These iteration schemes work for a wide range of problems
- They can often be implemented without forming the matrix explicitly.
- However, they have slow convergence

Example

For 2D Poisson equation,
Spectral radius of Jacobi iteration matrix is $\cos\left(\frac{\pi}{n}\right) \approx 1 - O\left(\frac{1}{n^2}\right)$.
Number of iterations required to achieve ϵ is $O(n^2 \ln \epsilon^{-1})$.

Stationary Iterative Methods



After 5 Jacobi iterations on a Poisson equation, error decreases very slowly.

Smoothing Effect

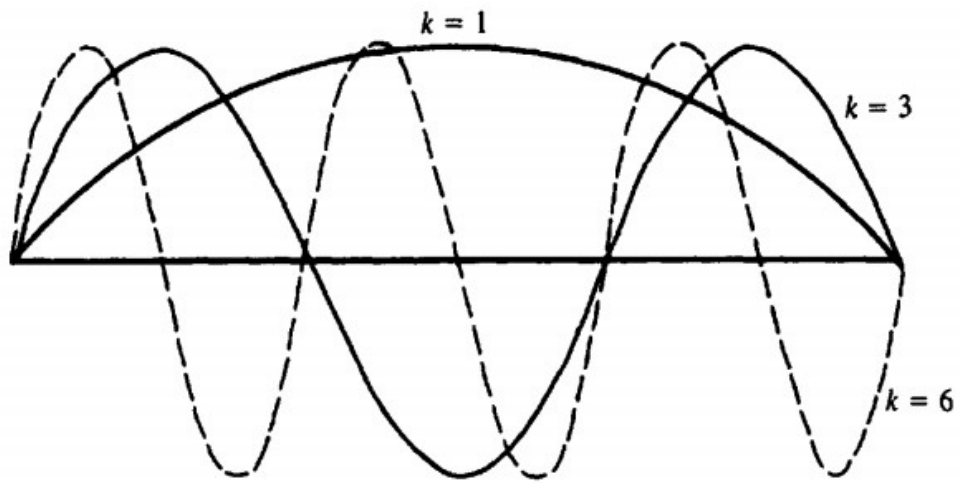
- The reason behind this behavior is the smoothing property of stationary iterative methods
- This property is one of the theoretical foundations of multigrid methods
- To illustrate the idea we apply iterative methods to the homogeneous system with initial guess v_k

$$Au = 0$$

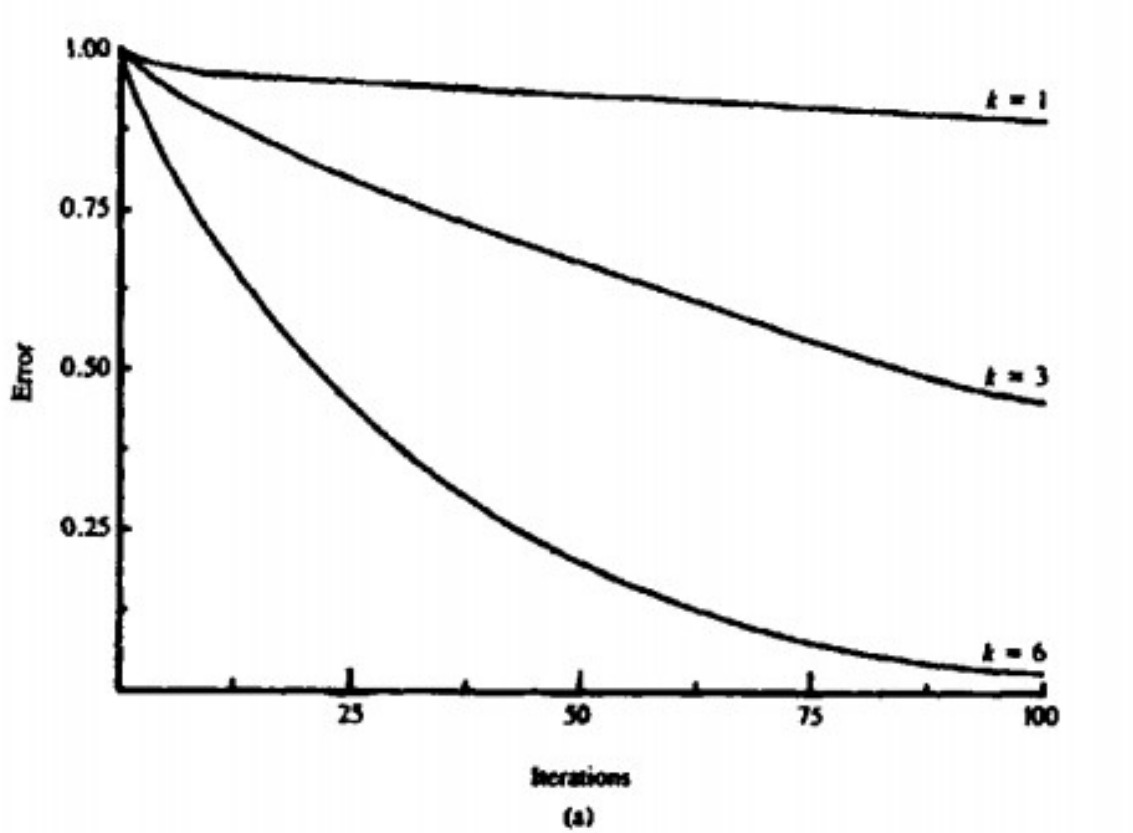
- v_k is chosen as $(v_k)_j = \sin\left(\frac{jk\pi}{n}\right),\ 1 \leqslant k \leqslant n-1,\ 1 \leqslant j \leqslant n-1$ *(Fourier modes)*

Smoothing Effect

- The modes in the lower half of the spectrum, with wavenumbers in the range $1 \leqslant k < \frac{n}{2}$ are called *low frequency* or smooth modes.
- The modes in the upper half of the spectrum, with $\frac{n}{2} \leqslant k \leqslant n-1$ are called *high frequency* modes or *oscillatory* modes.

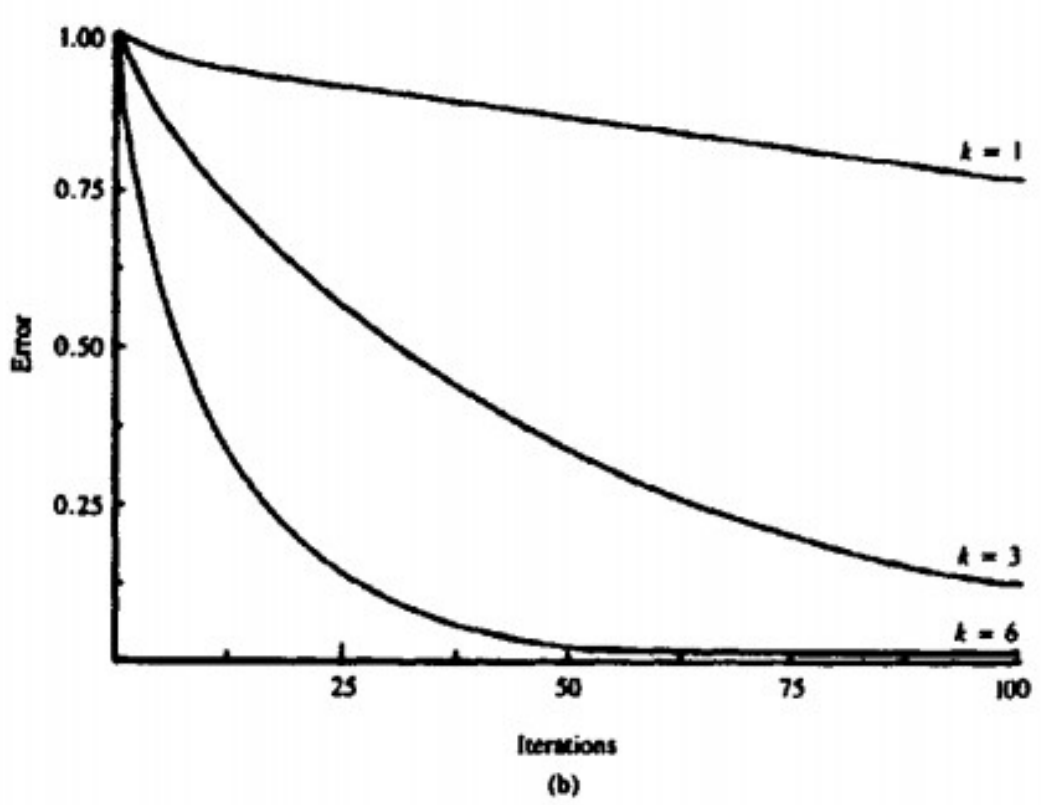


Smoothing Effect



Weighted Jacobi applied on 1-D model problem with 64 points with initial guess v_1, v_3 and v_6

Smoothing Effect

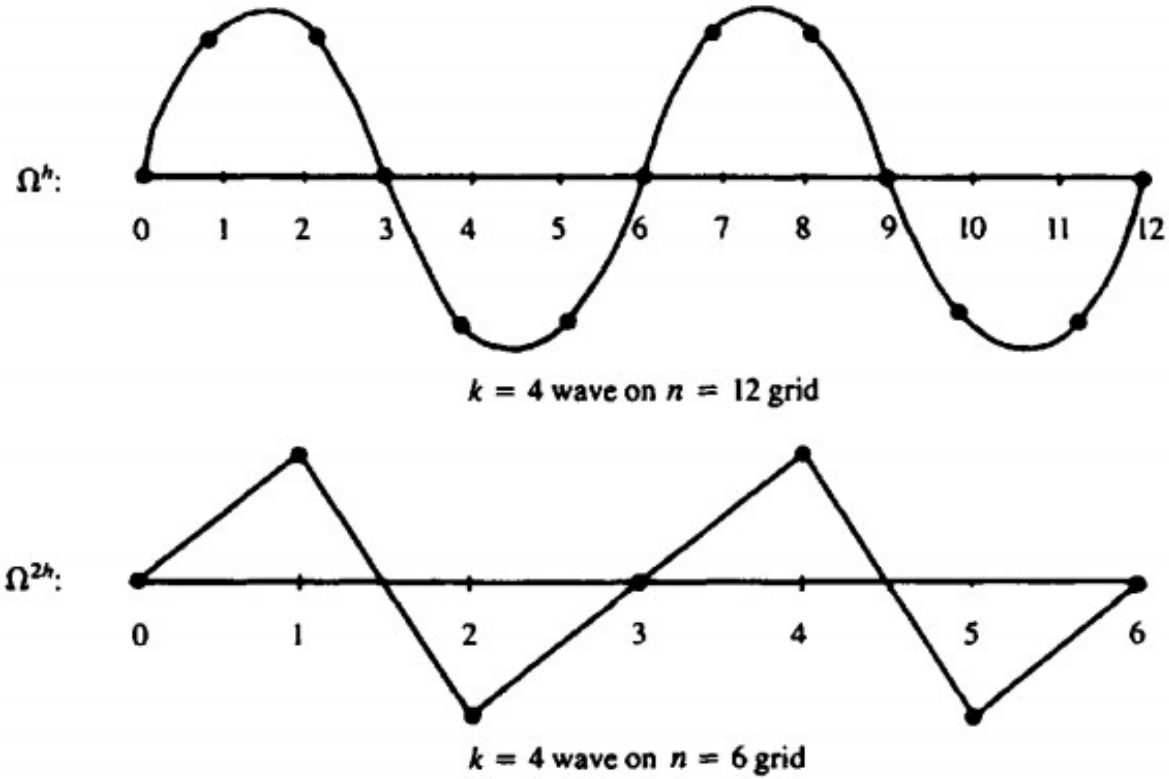


Gauss Seidel applied on 1-D model problem with 64 points with initial guess v_1, v_3 and v_6

Smoothing Effect

- Oscillatory modes are eliminated quickly
- Smooth modes remain relatively unchanged
- Errors for the model problem can be decomposed using these Fourier modes
- After several iterations, high frequency components will disappear and the error becomes smooth

Smoothing Effect



If we project a smooth wave directly onto a coarser grid, it becomes more oscillatory.

Smoothing Effect

- What does it imply?
- If we can move the error to a coarser grid, iterations will be more effective!
- Even if the error does not become more oscillatory, relaxing on the coarse grid is simply cheaper
- We may consider using coarse grids

Outline

Nested Iteration

We can solve problems on coarse grids to obtain better initial guesses:

- Relax $Au = f$ on a very coarse grid Ω^{8h} to obtain an initial guess for the next finer grid Ω^{4h}.
- Relax $Au = f$ on grid Ω^{4h} to obtain an initial guess for Ω^{2h}.
- Relax $Au = f$ on grid Ω^{2h} to obtain an initial guess for Ω^{h}.
- Relax $Au = f$ on Ω^{h} to obtain a final approximation to the solution.

Correction Scheme

- From our previous observations, error becomes smooth after relaxations.
- If we move the error to a coarser grid, it becomes oscillatory and iterations are effective
- What problem should we be solving then?
- The residual equation $Ae = f - Av = r$, where v is approximate solution of u

Correction Scheme

- Why residual equation?
- We want to relax the error directly since it becomes oscillatory on coarse level
- If we can solve the residual equation accurately then the real solution u can be obtained by $u = v + e$.
- Relaxation on the original equation $Au = f$ with arbitrary initial guess v is equivalent to relaxing on the residual equation $Ae = r$ with specific initial guess $e = 0$

Two-Grid Correction Scheme

The basic form of the multigrid method is defined as the following two-grid correction scheme [Briggs et al., Multigrid Tutorial]:

$$v^h \leftarrow MG\left(v^h, f^h\right)$$

1. **Presmoothing:** relax μ_1 times on $A^h u^h = f^h$ on Ω^h with initial guess v^h.
2. **Restriction:** compute the fine-grid residual $r^h = f^h - A^h v^h$ and restrict it to the coarse grid by $r^{2h} = Rr^h$.
3. **Coarse Grid Solving:** either solve $A^{2h} e^{2h} = r^{2h}$ or relax μ_3 times with initial guess 0 on Ω^{2h}.
4. **Prolongation:** interpolate the coarse-grid error to the fine grid by $e^h = Pe^{2h}$ and correct the fine-grid approximation by $v^h \leftarrow v^h + e^h$.
5. **Postsmoothing:** Relax μ_2 times on $A^h u^h = f^h$ on Ω^h with initial guess v^h.

Presmoothing

Presmoothing: relax μ_1 times on $A^h u^h = f^h$ on Ω^h with initial guess v^h.

- We apply μ_1 steps of iterations on the original linear system
- This step is known as the presmoothing step
- After iterations errors e^h will become smooth and it will appear oscillatory on Ω^{2h}
- We then approximate the residual equation on Ω^{2h}

Restriction

Restriction: compute the fine-grid residual $r^h = f^h - A^h v^h$ and restrict it to the coarse grid by $r^{2h} = Rr^h$.

- We restrict the residual onto Ω^{2h}
- Restriction operator can be chosen as injection $v_j^{2h} = v_{2j}^h$ or full-weighting $v_j^{2h} = \frac{1}{4}(v_{2j-1}^h + 2v_{2j}^h + v_{2j+1}^h)$

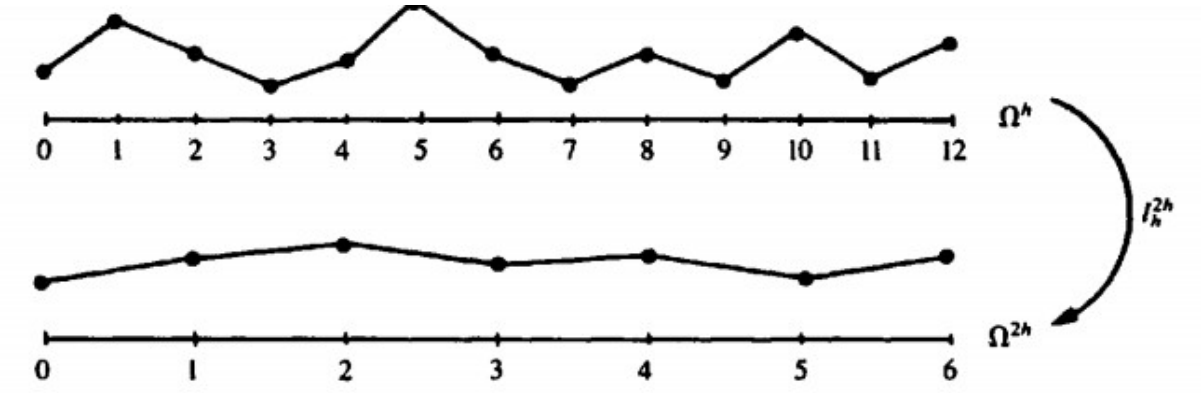


Figure: Restriction by full weighting

Solving on coarse level

Coarse Grid Solving: either solve $A^{2h}e^{2h} = r^{2h}$ or relax μ_3 times with initial guess 0 on Ω^{2h}.

- We obtain A^{2h} by rediscretizing the PDE on Ω^{2h}
- $A^{2h}e^{2h} = r^{2h}$ is an approximation of $A^h e^h = r^h$ on Ω^{2h}
- Iterative methods are effective as e^{2h} becomes oscillatory
- Iterations are also cheaper as there are less grid points

Prolongation

Prolongation: interpolate the coarse-grid error to the fine grid by $e^h = Pe^{2h}$ and correct the fine-grid approximation by $v^h \leftarrow v^h + e^h$.

- After e^{2h} is obtained, we interpolate it back to Ω^h and update error.
- Prolongation operator can be chosen as linear interpolation
  $v^h_{2j} = v^{2h}_j,\ v^h_{2j+1} = \frac{1}{2}(v^{2h}_j + v^{2h}_{j+1})$

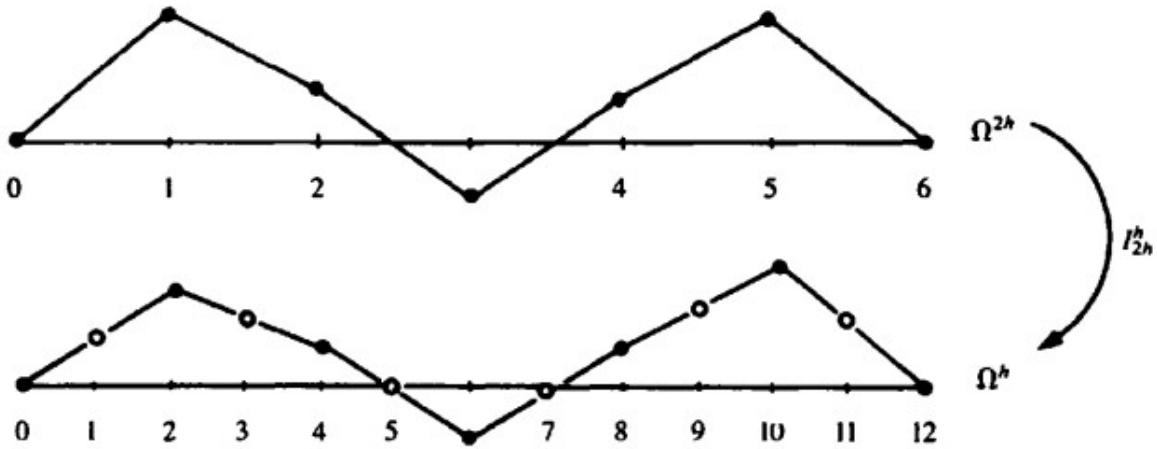


Figure: Prolongation by linear interpolation

Postsmoothing

Postsmoothing: relax μ_2 times on $A^h u^h = f^h$ on Ω^h with initial guess v^h.

- We apply μ_2 steps of iterations on the original linear system
- This step is known as the postmoothing step
- Errors will be further reduced

Two-Grid Correction Scheme

- Iteration on fine grid leaves smooth errors e^h and they appear to be oscillatory on coarse grid as e^{2h}.
- Iteration on coarse grid then solve e^{2h} effectively and e^{2h} will become a good approximation of e^h after interpolation.
- Finally with the correction step $v^h \leftarrow v^h + e^h$, we will obtain a solution very close to u.
- Errors which cannot be eliminated effectively by iterations are removed by coarse grid correction.

Toward Multigrids

- In our description, we assume e^{2h} on coarse level is solved accurately
- Practically a few steps of iterations can not guarantee sufficient accuracy of e^{2h}
- We may apply two-grid idea recursively on subsequent levels
- We can recursively solve problems on coarser levels and use them as initial guesses on fine levels

V-cycle and Full Multigrid scheme

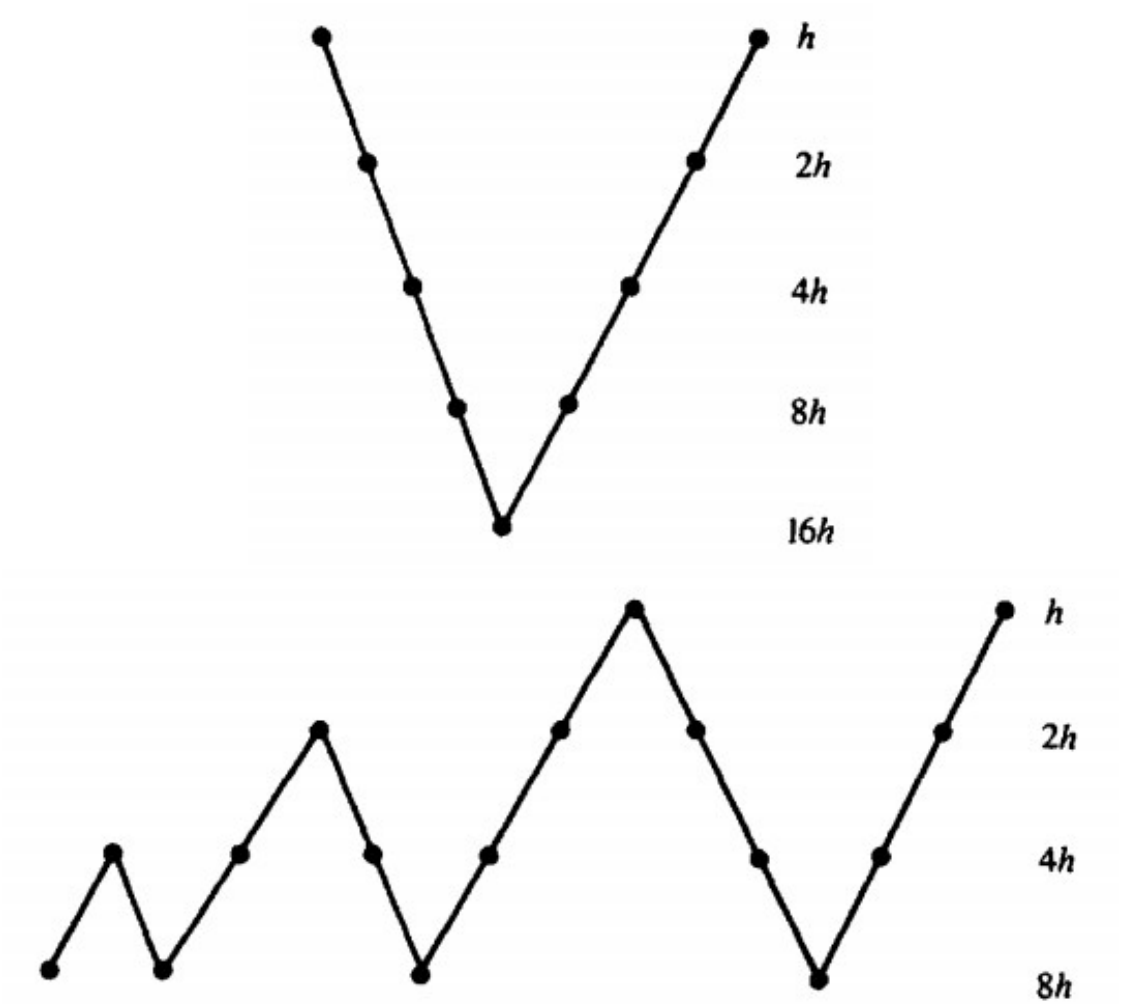


Figure: V-cycle and FMG scheme

Classical Algebraic Multigrid Method

- Smooth errors are defined algebraically as $e_{k+1} \approx e_k$ which leads us to $Ae \approx 0$
- Define interpolation operator P
- Coarsening is performed by a greedy maximum independent set algorithm on weighted graph
- Restriction is chosen as the transpose of interpolation and $A_{2h} = P^T A_h P$
- Current AMG development focuses on improving coarsening strategy and interpolation formula

Numerical Results: Poisson Equation

- Poisson equation discretized using third-order generalized finite difference method. The resulting matrix is asymmetric
- Comparison of algebraic multigrid method, MATLAB's built in direct solver and GMRES to problems of various sizes

	AMG $\left(TOL = 10^{-10}\right)$	Direct Solver	GMRES(10)
$10 \times 10 \times 10$	0.0081 seconds	0.0093 seconds	0.020 seconds
$20 \times 20 \times 20$	0.081 seconds	0.21 seconds	0.28 seconds
$40 \times 40 \times 40$	0.95 seconds	8.14 seconds	7.15 seconds
$80 \times 80 \times 80$	11.79 seconds	out of memory	470.88 seconds

Multigrid Software

- **hypre**: A library of high-performance preconditioners with a focus on multigrid methods. Developed by Lawrence Livermore National Laboratory. [Website: `https://computing.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods`]
- **ML (Multi-Level)**: Part of the Trilinos project, ML is a scalable preconditioning library for solving large sparse linear systems, with a focus on algebraic multigrid methods. [Website: `https://trilinos.github.io/ml.html`]
- **PyAMG (Python Algebraic Multigrid)**: A Python library implementing various algebraic multigrid solvers and tools. Great for integration with Python-based scientific computing stacks. [Website: `https://github.com/pyamg/pyamg`]

Further Reading

- Briggs, W. L., Henson, V. E., & McCormick, S. F. (2000). *A Multigrid Tutorial* (2nd ed.). SIAM. [An excellent resource for understanding multigrid methods in detail.]
- Trottenberg, U., Oosterlee, C. W., & Schüller, A. (2001). *Multigrid*. Academic Press. [Provides a comprehensive overview of multigrid techniques.]
- Hackbusch, W. (2013). *Multi-Grid Methods and Applications*. Springer-Verlag. [Focuses on both the practical and theoretical aspects of multigrid methods.]