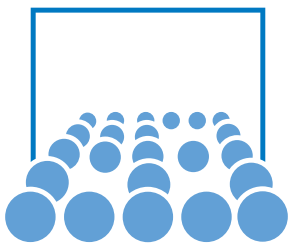
Algorithms of Scientific Computing II

3. Algebraic Multigrid Methods

Hans-Joachim Bungartz



3.1 Preliminary Remarks

3.2 Iterative Solvers

3.3 The Multigrid Principle

3.4 AMG

3.1 Preliminary Remarks

3.2 Iterative Solvers

3.3 The Multigrid Principle

3.4 AMG

3.1 Preliminary Remarks

- **Multigrid methods** and their many relatives (multi-scale, multi-resolution, multi-something) are the most efficient algorithms for the iterative solution of large sparse systems of linear equations. Here, in contrast to classical iterative schemes, the number of iterative steps that are needed to obtain a certain error reduction does not depend on the system size n.
- The system matrices typically stem from the discretization of partial differential equations – hence, there is a geometric context (neighbouring grid points, coarse and fine grids, high and low error frequencies). Therefore, the original multigrid solvers developed in that context are also called **geometric multigrid methods**.

Preliminary Remarks (2)

- However, what to do if such a context is missing? If there is just a system $Ax = b$, but no information on the origin of A? If there may have never been gird points etc.? For that, **algebraic multigrid methods (AMG)** have been developed – as highly efficient black box solvers, which do not know anything more than the matrix entries. This third chapter shall provide a brief introduction to AMG.
- We start with a concise overview of
  - the discretization of PDE (where do the matrices come from?),
  - classical iterative solvers (how did people solve these systems in pre-multigrid times?), and
  - the (geometric) multigrid principle.

3.1 Preliminary Remarks

3.2 Iterative Solvers

3.3 The Multigrid Principle

3.4 AMG

3.2 Iterative Solvers

Finite Difference Methods for PDE

- given: domain
$$\Omega \subseteq \mathbb{R}^d,\ d \in \{1, 2, 3\}$$
- on Ω, we define: (regular) **grid** Ω_h
- grid resolution or **mesh width**
$$h = (h_x, h_y, h_z)$$
- replace derivatives with difference quotients:
  - first derivatives: **forward**-, **backward**-, or **central** differences
$$\frac{\partial u}{\partial x}(\xi) \doteq \frac{u(\xi + h_x) - u(\xi)}{h_x}, \frac{u(\xi) - u(\xi - h_x)}{h_x}, \frac{u(\xi + h_x) - u(\xi - h_x)}{2h_x}$$
  - second derivatives: standard **3-point stencil**
$$\frac{\partial^2 u}{\partial x^2} \doteq \frac{u(\xi + h_x) - 2u(\xi) + u(\xi - h_x)}{h_x^2}$$
  - Laplace operator in 2D or 3D: 5-point or 7-point stencil
  - There are broader stencils, too (involving more neighbours).

Finite Difference Methods (2)

- in each inner grid point:
  - construct a **difference equation**
  - unknowns (**degrees of freedom**): discrete approximation to the function values in the grid points
  - one degree of freedom per grid point and per (scalar) unknown quantity
- in points on or close to the boundary:
  - concrete equation depends on boundary conditions:
    - **Dirichlet**: no difference equation in boundary points (there, the function values are given, hence no unknowns needed)
    - **Neumann**: special appearance of the difference equation at the boundary (implement the boundary conditions)
- discretization leads to a system of linear equations (SLE)
  - sparsely populated, in general
  - fast (iterative) solvers are vital to scientific computing

Finite Difference Methods (3)

- simple example: Poisson equation on the unit square

$$-\Delta u = f \quad \text{on} \quad]0,1[^2$$

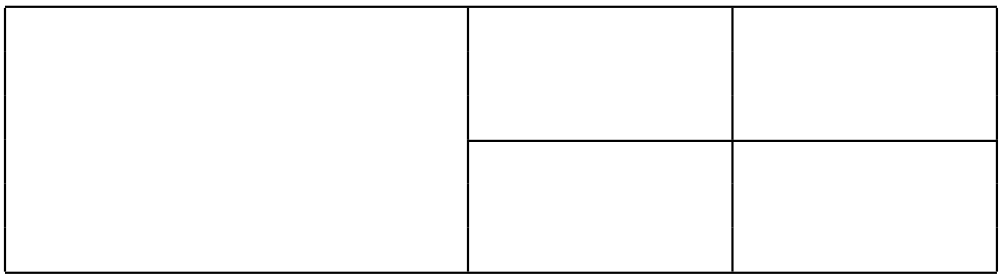
- equidistant square grid: $h = h_x = h_y = 1/N$
- number of degrees of freedom (unknowns): $M = (N-1)^2$
- standard 5-point stencil results in linear system $Ax = b$ with
  - $M \times M$-Matrix A (the pointwise difference equations)
  - M-vectors b (right-hand side) and x (unknown values of u)
- A is a **sparse matrix** with band structure; after multiplication with h^2 :
  - **Dirichlet boundary conditions:**
    - all diagonal elements are 4, per row 2 to 4 non-zeroes (-1)
    - boundary values to the right-hand side
  - **Neumann boundary conditions** (various possibilities):
    - diagonal elements are 2 or 3 next to the boundary and 4 elsewhere; a corresponding number of -1 entries per row
    - pairs $(1, -1)$ along the boundary to the right-hand side

Finite Difference Methods (4)

- typically, accuracy of quadratic order:

$$\|u^{\text{computed}} - u^{\text{exact}}\| = O(h^2) = O(N^{-2})$$

- **curse of dimensionality**:
  - $O(N^d)$ points are needed in case of d dimensions
- starting points for improvements:
  - stencils of higher order (cubic, quartic, ...):
    - take more than two neighbouring points into account
    - problem: matrix gets denser (less non-zeroes)
  - being more economic with grid points:
    - use locally refined grids (adaptive grids)
    - problem: what to do where two resolutions meet – which values?



Standard Iterative Schemes for SLE

- **iterative** solution of large (sparse) SLE:
  - one of the central tasks in numerical simulation
  - they occur at the discretization of ODE (BVP) and PDE
- frequently, **direct** solvers are not competitive:
  - too large number of unknowns (cf. PDE in 3D)
  - classical elimination approaches may destroy the (sparse) structure of the matrix
  - why exact solution in case of approximations? (this holds, esp., for the nonlinear case, where an SLE has to be solved in each iterative step to treat the nonlinearity)
  - goal: performance of the kind ”‘for 3 digits we need 10 steps”’ – independent of the number of unknowns
- however, the typical reality of classical iterative schemes:
  - speed of convergence deteriorates with increasing problem size!

Basics of Iterative Schemes

- consider iterative methods, start from $x^{(0)} \in \mathbb{R}^n$ and end (hopefully) close to the solution x of $Ax = b$:

$$x^{(0)} \to x^{(1)} \to \ldots \to x^{(i+1)} \to \ldots \to \lim_{i \to \infty} x^{(i)} = x$$

- speed of convergence:

$$\|x - x^{(i+1)}\| < \gamma \cdot \|x - x^{(i)}\|^s$$

  for a $0 < \gamma < 1$, order of convergence s
- typical behaviour of simple iterative methods for SLE:

$$s = 1, \quad \gamma = O(1 - n^{-k}), \quad k \in \{0, 1, 2, ...\}$$

  (n : number of grid points)
- strategy: look for methods with
  - only $O(n)$ arithmetic operations per iterative step (cost; that's the minimum effort)
  - convergence behaviour according to $\gamma < 1 -$ const. (benefit)
- two big families: **relaxation** and **Krylov subspace** methods

Relaxation Methods

- sometimes also called **smoothers** :
  - **Richardson** iteration
  - **Jacobi** iteration
  - **Gauß-Seidel** iteration
  - **successive over-relaxation (SOR)** or **damped** methods
- starting points:
  - **error** e of the current approximation (cause): unknown
  - **residual** r of the current approximation (effect): easily accessible

$$r^{(i)} = b - Ax^{(i)} = Ax - Ax^{(i)} = A(x - x^{(i)}) = -Ae^{(i)}$$

  (used as error indicator, since nothing better is available)
- how to use the residual for better approximations?
  - **Richardson**: use the residual directly as a correction
  - **Jacobi/Gauß-Seidel**: make one component of r to zero
  - **SOR/damped**: as before, but correction may be to big / small

Important Relaxation Schemes

- **Richardson iteration**:

```
for i = 0,1,...
        for k = 1,...,n:
```

$$x_k^{(i+1)} := x_k^{(i)} + r_k^{(i)}$$

Here, simply the residual $r^{(i)}$ is taken as correction to that current approximation $x^{(i)}$ (component-wise).

Important Relaxation Schemes (2)

- **Jacobi iteration**:

```
for i = 0,1,...
            for k = 1,...,n:     y_k := 1/a_kk · r_k^(i)
            for k = 1,...,n:     x_k^(i+1) := x_k^(i) + y_k
```

  - In each sub-step k of a step i, a correction y_k is computed and stored.
  - Applied immediately, this would make the k-component of the residual $r^{(i)}$ to vanish (for the moment – to be verified easily by inserting).
  - Thus, equation k would be exactly solved with this approximation to x – a progress, which would be immediately destroyed in the following sub-step $k+1$, of course.
  - However, these component updates are not applied at once, but at the end of each complete iterative step only (second k-loop).

Important Relaxation Schemes (3)

- **Gauß-Seidel Iteration**:

```
for i = 0,1,...
    for k = 1,...,n:
```

$$r_k^{(i)} := b_k - \sum_{j=1}^{k-1} a_{kj} x_j^{(i+1)} - \sum_{j=k}^{n} a_{kj} x_j^{(i)}$$

$$y_k := \frac{1}{a_{kk}} \cdot r_k^{(i)}, \quad x_k^{(i+1)} := x_k^{(i)} + y_k$$

  - Here, again the Jacobi correction is computed, but now, and in contrast to Jacobi's method, each update is applied immediately, as soon as it is available, and not at the end of each step of iteration.
  - Hence, when updating component k, we already know and use the modified new values of components 1 to $k-1$.

Important Relaxation Schemes (4)

- Sometimes, a **damping** (multiplication of the correction term with some factor $0 < \alpha < 1$) or an **over-relaxation** (factor $1 < \alpha < 2$) improves the convergence behaviour of the schemes discussed above:

$$x_k^{(i+1)} := x_k^{(i)} + \alpha y_k \, .$$

  - For Gauß-Seidel, the variant with $\alpha > 1$ is widespread, and we then speak of **SOR methods** (**Successive Over-Relaxation**).
  - For Jacobi, however, typically damping is used.

Discussion: Additive Splitting of the System Matrix

- For a short convergence analysis of the relaxation schemes introduced above, we need an algebraic formulation instead of the algorithmic one.
- All four schemes are based on the simple idea to write matrix A as a sum $A = M + (A - M)$, where $Mx = b$ is very easy to solve and where the difference $A - M$ should be small with respect to some matrix norm.
- With the help of such a suitable M, Richardson, Jacobi, Gauß-Seidel, and SOR methods can be written as

$$Mx^{(i+1)} + (A - M)x^{(i)} = b$$

or, solved for $x^{(i+1)}$,

$$x^{(i+1)} := M^{-1}b - M^{-1}(A-M)x^{(i)} = M^{-1}b - (M^{-1}A - I)x^{(i)} = x^{(i)} + M^{-1}r^{(i)}$$

- Furthermore, we split A in another way, using its diagonal part D_A, its strictly lower triangular part L_A, and its strictly upper triangular part U_A as summands:

$$A =: L_A + D_A + U_A \,.$$

Discussion: Additive Splitting of the System Matrix (2)

With that, we can show the following relations:

- Richardson: $M := I$,
- Jacobi: $M := D_A$,
- Gauß-Seidel: $M := D_A + L_A$,
- SOR: $M := \frac{1}{\alpha} D_A + L_A$.

- When we consider the algorithmic formulations for Richardson's and Jacobi's methods, the first two relations are obvious:
  - With Richardson, the residual is taken as correction directly, which leads to the identity I as pre-factor.
  - With Jacobi, the residual is divided by the diagonal matrix element, which leads to the inverse of the diagonal part D_A as pre-factor.

Discussion: Additive Splitting of the System Matrix (3)

- Since the Gauß-Seidel iteration is a special case of the SOR scheme ($\alpha = 1$), it suffices to prove the above formula for M for the general SOR case. The algorithm directly provides

$$
\begin{aligned}
& x_k^{(i+1)} := x_k^{(i)} + \alpha \left(b_k - \sum_{j=1}^{k-1} a_{kj} x_j^{(i+1)} - \sum_{j=k}^{n} a_{kj} x_j^{(i)} \right) / a_{kk} \\
\Leftrightarrow \quad & x^{(i+1)} := x^{(i)} + \alpha D_A^{-1} \left(b - L_A x^{(i+1)} - (D_A + U_A) x^{(i)} \right) \\
\Leftrightarrow \quad & \frac{1}{\alpha} D_A x^{(i+1)} = \frac{1}{\alpha} D_A x^{(i)} + b - L_A x^{(i+1)} - (D_A + U_A) x^{(i)} \\
\Leftrightarrow \quad & \left(\frac{1}{\alpha} D_A + L_A \right) x^{(i+1)} + \left((1 - \frac{1}{\alpha}) D_A + U_A \right) x^{(i)} = b \\
\Leftrightarrow \quad & M x^{(i+1)} + (A - M) x^{(i)} = b \,,
\end{aligned}
$$

  which proves the assumption for the SOR method.

Discussion: General Convergence Behaviour

- Concerning convergence, there are two direct consequences of the approach
$$Mx^{(i+1)} + (A - M)x^{(i)} = b :$$
- If the sequence $\left(x^{(i)}\right)$ converges at all, then the exact solution x of our system $Ax = b$ is the limit.
- For an analysis, we assume that the matrix $-M^{-1}(A - M)$ (i. e. the matrix which is applied to $e^{(i)}$ to get $e^{(i+1)}$; see below) is symmetric. Then, its spectral radius ρ (i.e. the largest (modulus!) eigenvalue) is the quantity that characterizes the convergence behaviour:
$$\left(\forall x^{(0)} \in \mathbb{R}^n : \lim_{i\to\infty} x^{(i)} = x = A^{-1}b\right) \quad \Leftrightarrow \quad \rho < 1 \,.$$
To see that, subtract $Mx + (A - M)x = b$ from the equation at the slide's top:
$$Me^{(i+1)} + (A - M)e^{(i)} = 0 \quad \Leftrightarrow \quad e^{(i+1)} = -M^{-1}(A - M)e^{(i)} \,.$$

Discussion: General Convergence Behaviour (2)

- If all eigenvalues have a modulus smaller than 1 and if, hence, $\rho < 1$, then all error components are reduced in each step of the iteration. If $\rho > 1$, at least one error component will/may grow.
- When constructing iterative schemes, our goal must, of course, be a spectral radius that is as small as possible (as close to zero as possible).

Discussion: Convergence Statements

- There are a couple of results on the methods' convergence. We mention some important ones:
  - For the convergence of the SOR method, a necessary condition is $0 < \alpha < 2$.
  - If A is positive definite, then both the SOR (for $0 < \alpha < 2$) and the Gauß-Seidel iteration converge.
  - If A and $2D_A - A$ are both positive definite, then Jacobi's method converges.
  - If A is strictly diagonally dominant, (i. e. if $a_{ii} > \sum_{j \neq i} \mid a_{ij} \mid$ for all i), then both the Jacobi and the Gauß-Seidel method converge.
  - In certain cases, the optimal parameter α can be determined, such that ρ gets minimal and, hence, the error reduction per step of iteration gets maximal.
- The Gauß-Seidel iteration is not generally better than Jacobi's, as one might suppose due to the immediate updates. There are examples where the first converges but the latter diverges, and vice versa. In many cases, however, Gauß-Seidel needs only half the number of steps Jacobi needs.

On the Spectral Radius of Typical Iteration Matrices

- Obviously, ρ is not only crucial for the question whether the iteration scheme converges at all, but also for its quality, that is its speed of convergence: The smaller ρ is, the faster all components of the error $e^{(i)}$ are reduced in each iterative step.
- In practice, unfortunately, the above results on convergence are of a more theoretical value only, since ρ is frequently that close to 1 that – despite convergence – the number of steps to obtain convergence is far too big.
- An important scenario is the dicretization of partial differential equations:
  - Typically, ρ depends on the problem size n and, hence, on the resolution h of the underlying grid, for example

$$\rho \;=\; O(1-h_l^2) \;=\; O\left(1-\frac{1}{4^l}\right)$$

    with a mesh width $h_l = 2^{-l}$.
  - This is a huge drawback: The finer and more accurate our grid is, the poorer gets the convergence behaviour of our relaxation methods. Hence, better iterative solvers are a must!

3.1 Preliminary Remarks

3.2 Iterative Solvers

3.3 The Multigrid Principle

3.4 AMG

3.3 The Multigrid Principle

- starting point: **Fourier mode analysis** of the errors
  - decompose the error $e^{(i)} = x^{(i)} - x$ into its Fourier components (Fourier transform)
  - observe how they change/decrease under a standard relaxation like Jacobi or Gau-Seidel (in a two-band sense):
    - The *high* frequency part (with respect to the underlying grid) is reduced quite quickly.
    - The *low* frequency part (w.r.t. the grid) decreases only very slowly; actually the slower, the finer the grid is.
  - This behaviour is annoying
    - the low frequencies are not expected to make troubles, but we can hardly get rid of them on a fine grid;
  - but also encouraging
    - the low frequencies can be represented and, hopefully, tackled on a coarser grid – there is no need for the fine resolution.

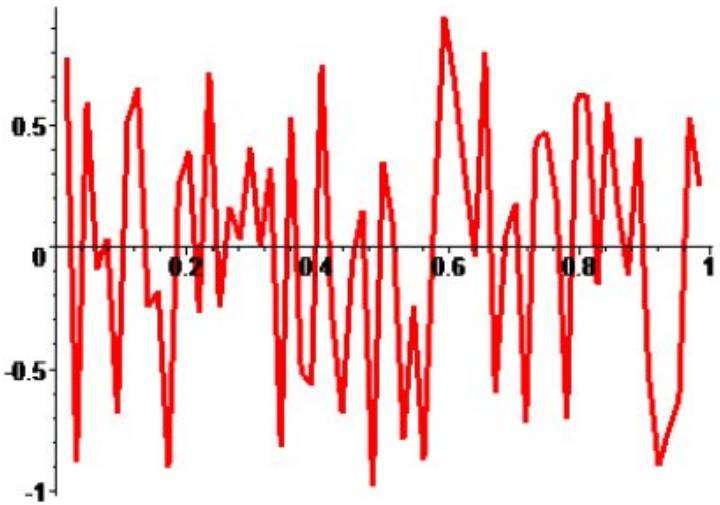
A Simple Example

- 1D Laplace equation, $u(0) = u(1) = 0$ (exact solution 0)
- equidistant grid, 65 points, 3-point stencil, damped Jacobi method with damping parameter 0.5
- start with random values in $[0, 1]$ for u in the grid points
- After 100 (!) steps, there is still a maximum error bigger than 0.1 due to low-frequency components!
- therefore the name **smoothers** for relaxation schemes:
  - They reduce the strongly oscillating parts of the error quite efficiently.
  - They, thus, produce a **smooth** error which is very resistent.
- the idea: work on grids of different resolution and combine the effects in an appropriate way

A Simple Example (2)

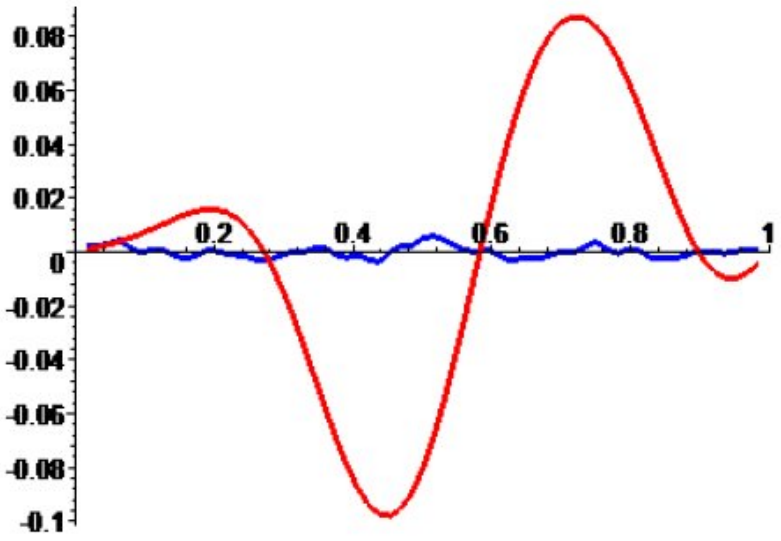
Random-Startfehler

zum Beispielproblem



Fehler nach 100 Jacobi-Schritten (rot)

Fehler nach 2 Mehrgitter-Schritten (blau)



Coarse Grid Correction

- sequence of equidistant grids on our domain: $\Omega_l,\ l = 1, 2, \ldots, L$, with mesh width $h_l = 2^{-l}$
- let A_l, b_l denote corresponding matrix, right-hand side, ...
- combine work on two grids with a **correction scheme**:

smooth the current solution x_l ;

form the residual $r_l = b_l - A_l x_l$;

restrict r_l to the coarse grid Ω_{l-1} ;

provide a solution to $A_{l-1}\, e_{l-1} = r_{l-1}$;

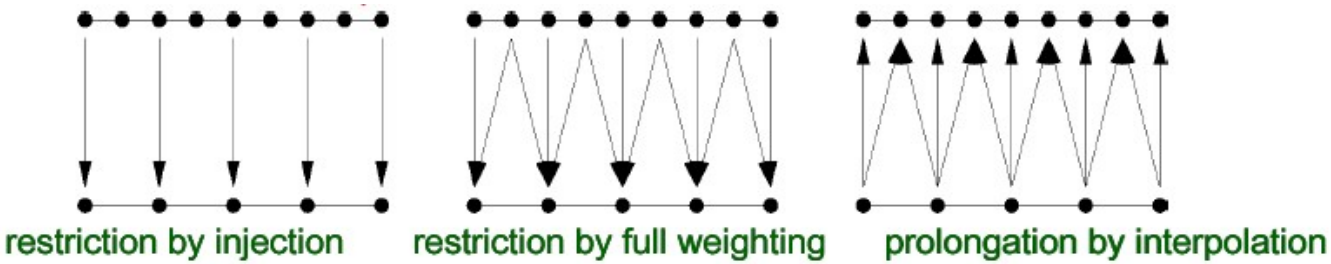
prolongate e_{l-1} to the fine grid Ω_l;

add the resulting correction to x_l;

if necessary, smooth again ;

Coarse Grid Correction (2)

- the different steps of this **2-grid algorithm** :
  - the **pre-smoothing**: reduce high-frequency error components, smooth error, and prepare residual for transfer to coarse grid
  - the **restriction**: transfer from fine grid to coarse grid
    - **injection** : inherit the coarse grid values and forget the others
    - **(full) weighting** : apply some averaging process
  - the **coarse grid correction**: provide an (approximate) solution on the coarse grid (direct, if coarse enough; some smoothing steps otherwise)
  - the **prolongation**: transfer from coarse grid to fine grid
    - usually some **interpolation** method



  - the **post-smoothing**: sometimes reasonable to avoid new high-frequency error components

The V-Cycle

- recursive application of 2-grid scheme leads to **multigrid methods**
- there, the coarse grid equation is solved by coarse grid correction, too; the resulting algorithmic scheme is called **V-cycle** :

smooth the current solution x_l ;

form the residual $r_l = b_l - A_l x_l$;

restrict r_l to the coarse grid Ω_{l-1} ;

solve $A_{l-1}\, e_{l-1} = r_{l-1}$ by coarse grid correction ;

prolongate e_{l-1} to the fine grid Ω_l;

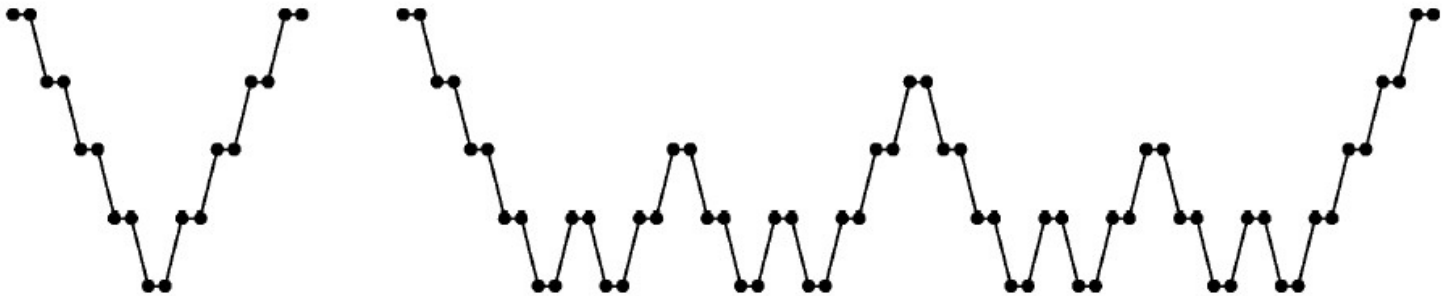
add the resulting correction to x_l;

if necessary, smooth again ;

- on the coarsest grid: direct solution
- number of smoothing steps: typically small (1 or 2)

Further Multigrid Cycles

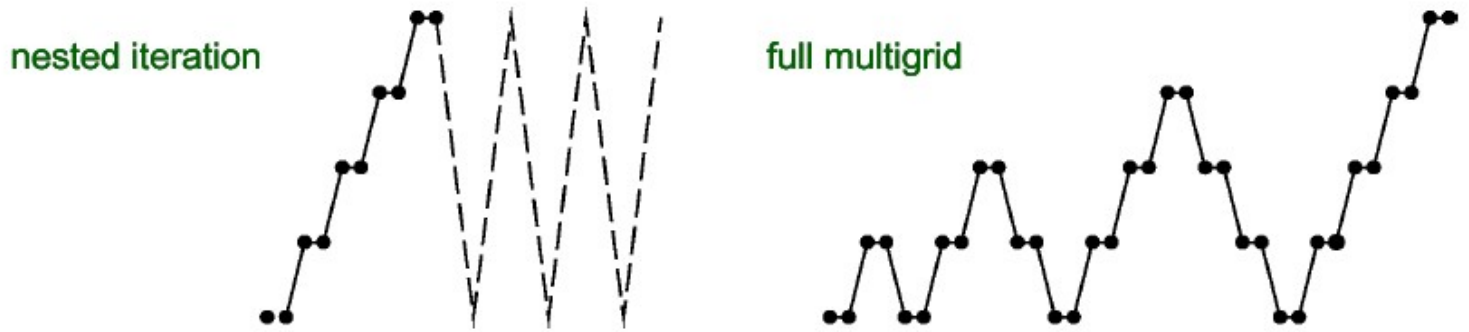
- the V-cycle is not the only multigrid scheme:
  - the **W-cycle**: after each prolongation, visit the coarse grid once more, before moving on to the next finer grid



  - W-cycle is sometimes advantageous with respect to speed of convergence

Nested Iteration and Full Multigrid

- two more famous multigrid schemes:
  - the **nested iteration**: start on coarsest grid Ω_1 , smooth, prolongate to Ω_2 , smooth, prolongate to Ω_3 , and so on, until finest grid is reached; now start V-cycle
  - **full multigrid**: replace ‘smooth steps above by ‘apply a V-cycle; combination of improved start solution and multigrid solver



- multigrid idea is not limited to rectangular or structured grids: we just need a hierarchy of nested grids (works for triangles or tetrahedra, too)
- also without underlying geometry: **algebraic** multigrid methods

Basic Convergence Results

- **Cost** (storage and computing time):
  - $1D:\quad c \cdot n + c \cdot n/2 + c \cdot n/4 + c \cdot n/8 + \ldots \leq 2c \cdot n = O(n)$
  - $2D:\quad c \cdot n + c \cdot n/4 + c \cdot n/16 + c \cdot n/64 + \ldots \leq 4/3c \cdot n = O(n)$
  - $3D:\quad c \cdot n + c \cdot n/8 + c \cdot n/64 + c \cdot n/512 + \ldots \leq 8/7c \cdot n = O(n)$
  - i.e.: work on coarse grids is negligible compared to finest grid
- **Benefit** (speed of convergence):
  - always significant acceleration compared with pure use of smoother (relaxation method)
  - in most cases even ideal behaviour $\gamma = O(1 - \text{const.})$
  - effect:
    - constant number of multigrid steps to obtain a given number of digits
    - overall computational work increases only linearly with n

3.1 Preliminary Remarks

3.2 Iterative Solvers

3.3 The Multigrid Principle

3.4 AMG

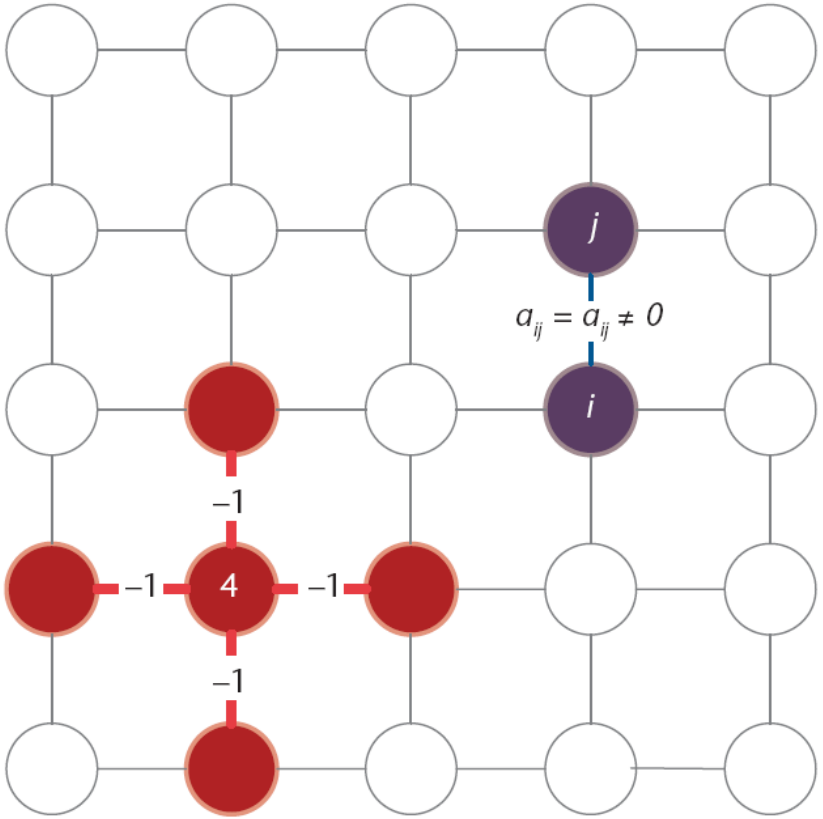
3.4 AMG

- As in the geometric case, we want to solve the system $Ax = b$ for some real, symmetric, and positive definite matrix A and a right-hand side b. We do not make any further assumptions on structure or origin of A.
- For a multigrid algorithm, we, again, need
  - a **smoother** or **relaxation scheme** and
  - a **coarse grid correction**, consisting of the components **restriction**, **coarse grid solver**, and **prolongation**.
- These notions are still geometry-related (“coarse“, etc.), the algorithms, however, won’t have any geometric context any more. In particular, we have to do without our frequency analogy from now on.
- AMG defines coarse grids, operators for grid transfer, and coarse grid equations solely based on the matrix entries $a_{i,j}$.
- Today, a lot of variants are available – for the following, we concentrate on the original AMG algorithm, C-AMG.
- The underlying algorithmic pattern (2-grid scheme, V-cycle) strongly resembles the one from geometric multigrid (see the previous section).

AMG and Geometry

- Attention: Though AMG does not need any geometric context, AMG is also frequently used for partial differential equations (which, of course, have a geometric context). That's why we will meet geometric analogies again and again.
- This gets apparent when we consider the adjacency graph of A:
  - nodes represent unknowns, i. e. components x_i of the solution vector x;
  - edges are present, if $a_{i,j} \neq 0$.
  - If A stems from a 2 D PDE grid, then grid and graph frequently look the same.

AMG and Geometry (2)



5x5 grid and adjacency graph for the Laplace equation

Algebraic Smoothness

- What does **smooth** mean, if there isn't any relation to geometry nor frequency?
  - **geometrically smooth**: the usual idea of smoothness (not oscillating etc.)
  - **algebraically smooth**: simply everything the smoother produces (in an AMG context, typically, Gauß-Seidel or related methods) – which might also imply something geometrically non-smooth
- An algebraically smooth error typically mainly consists of components of eigenvectors to small eigenvalues (so-called **small eigenmodes**):
  - This means that smoothing especially damps the largest eigenmodes.
  - The smallest eigenmodes – the "almost-kernel" of A – thus require the largest attention when doing the coarse grid correction.

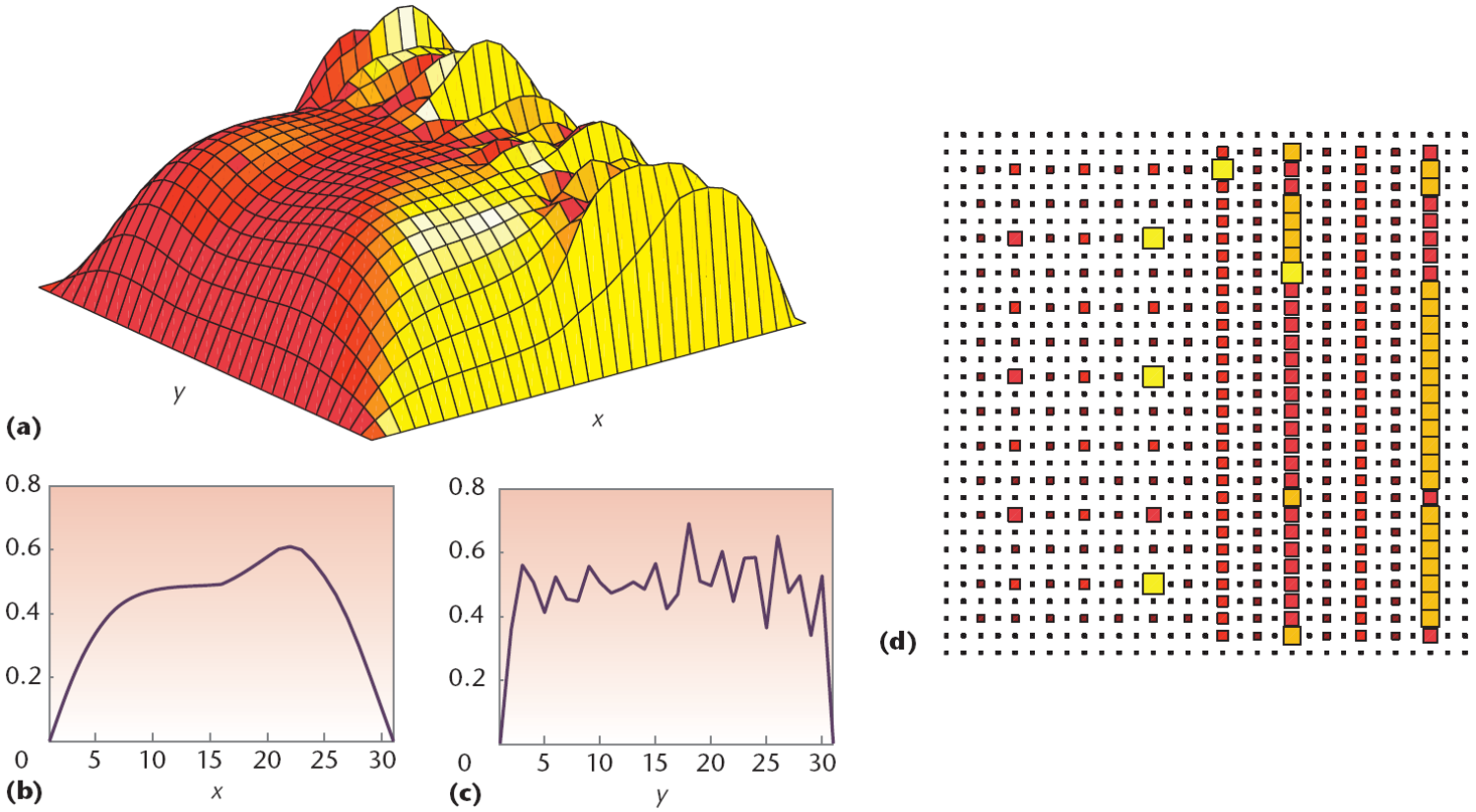
Algebraic Smoothness (2)

- A simple example problem:

$$-au_{xx} - bu_{yy} = f \quad \text{on} \quad [0,1]^2, \quad \text{Dirichlet boundary},$$

  where $a = b$ in the left half ($x < 0.5$) and $a \gg b$ elsewhere
  - Constant functions lie in the almost-kernel of A (matrix structure!).
  - Hence, geometric smoothness and almost-kernel coincide in this example.
  - Therefore, we expect AMG to coarsen in the direction of geometric smoothness (see the next slide).

An Example for Algebraic Smoothness



Algebraic smoothness. (a) Error for our sample problem, after seven Gauß-Seidel sweeps. (b) The error is geometrically smooth in x-direction, (c) but oscillates in y-direction in the right half. (d) In this example, AMG coarsens the grid in the direction of geometric smoothness. Bigger squares correspond to coarser grids.

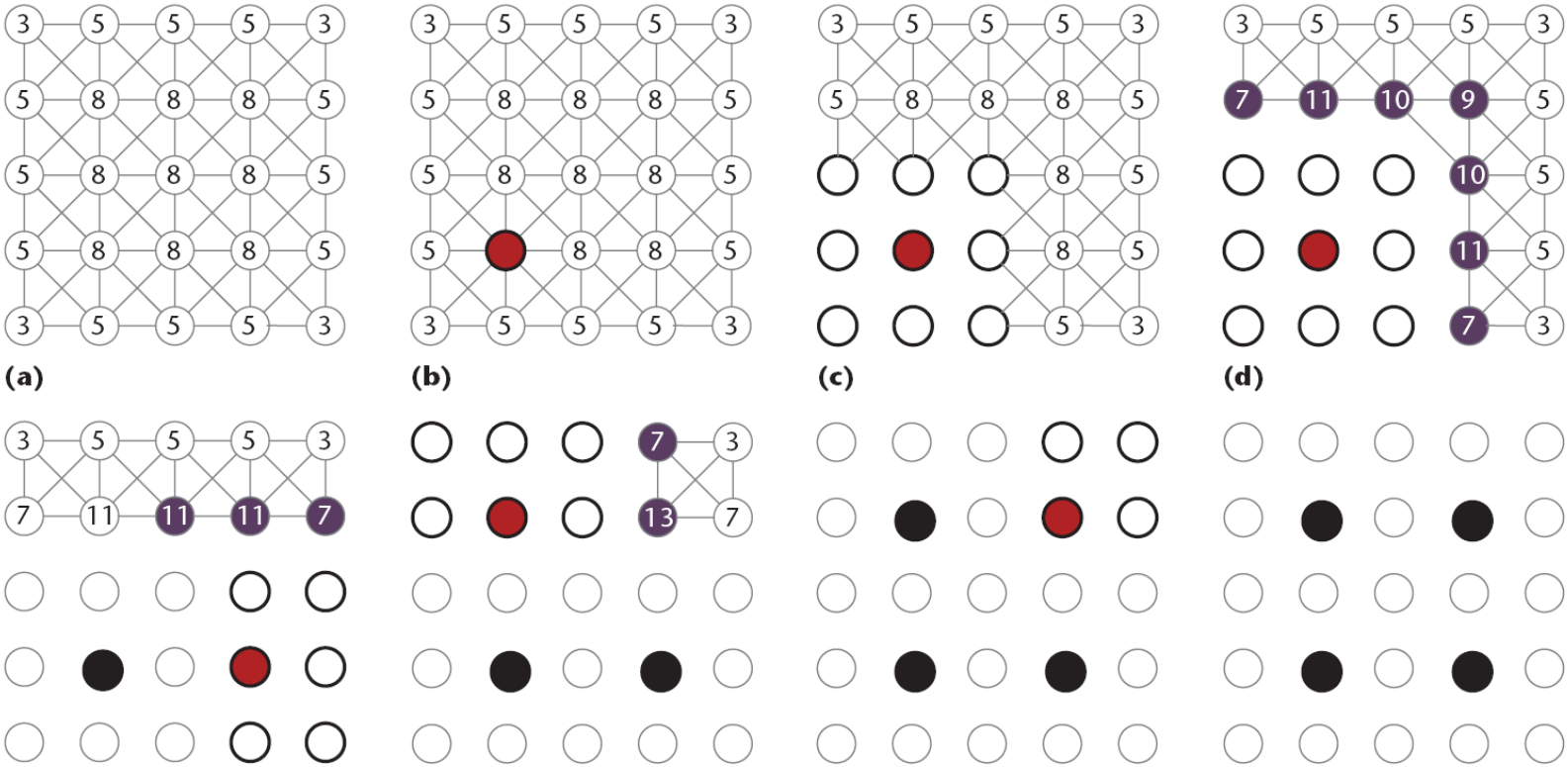
Basic Concepts in Classical AMG (C-AMG)

- **heuristics for smoothness**: smooth errors only change slowly in the direction of relatively large (negative) matrix entries.
  - For small eigenmodes v (length normalized to 1), it can be shown that
$$v^T A v \;=\; \sum_{i<j} (-a_{i,j}) \cdot (v_i - v_j)^2 \;\ll\; 1\,.$$
  - If $-a_{i,j}$ is large, v_i and v_j can not differ strongly.
- **Connection strength**: For a given threshold $\theta, 0 < \theta \leq 1$, the variable x_i strongly depends on x_j, if
$$-a_{i,j} \;\geq\; \theta \cdot \max_{k \neq i} \{-a_{i,k}\}\,.$$
  - That means that the connection strength is measured relatively to the largest negative off-diagonal element in the row; positive off-diagonal elements are always considered as weak connections.
  - Despite $A = A^T$, it is possible that x_i strongly depends on x_j, but x_j only depends weakly on x_i (there are AMG variants where this is not possible; here, we assume symmetric connection strength).

C-AMG: Coarse Grid Selection

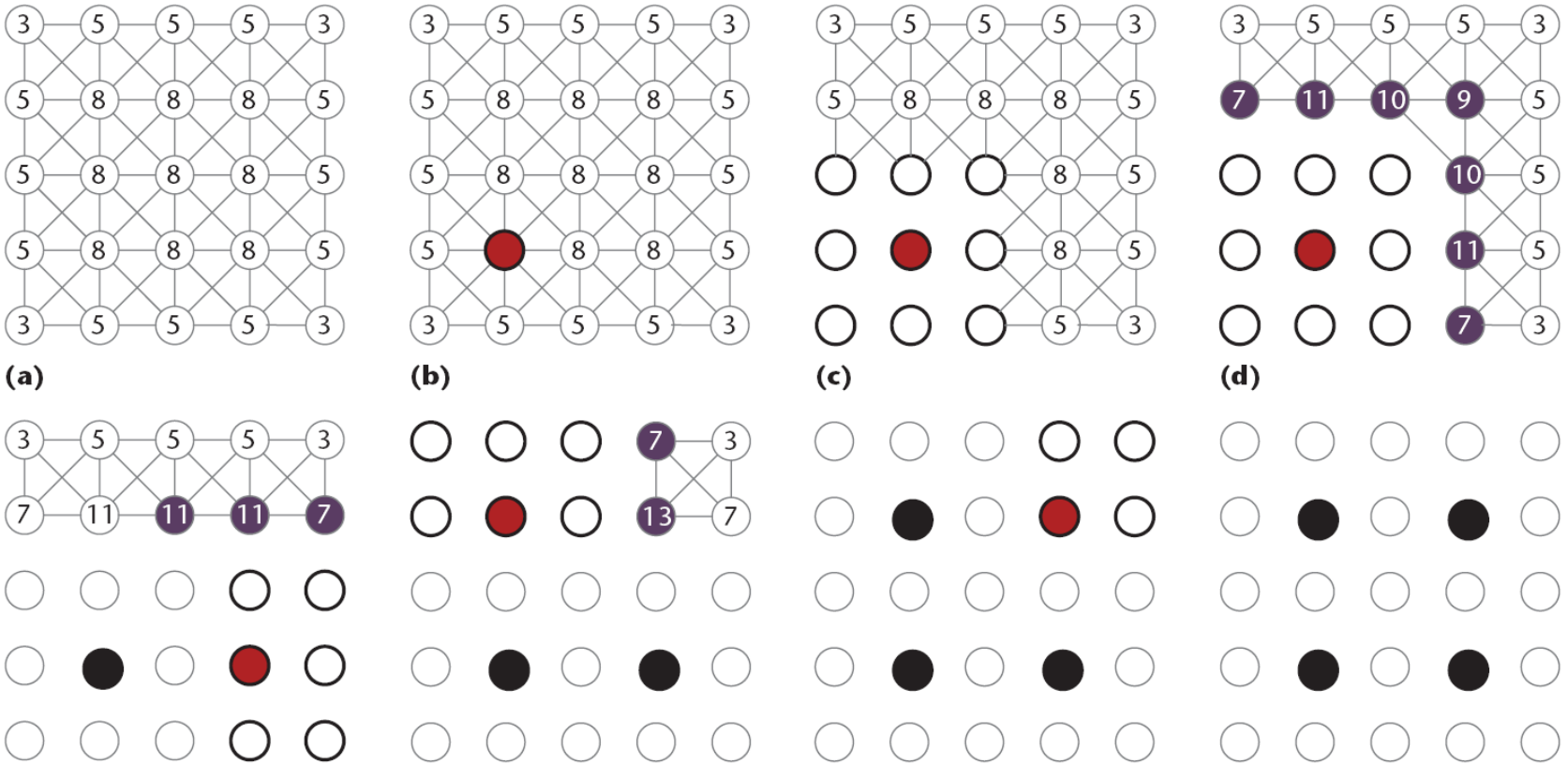
- The coarse grid always is a subset of the fine grid.
- Main principle underlying the coarse grid selection: Coarsen in the direction of strong matrix connections.
- Structure of the algorithm:
  1. get A_s from A by removing all weak connections or corresponding entries, resp.
  2. pass 1: determine a set C of coarse grid points and a set F (all other variables), starting from A_s
  3. pass 2: tune and optimize, if the interpolation should require that
- example:
  - 2 D Laplace equation
  - finite element discretizazion on uniform grids
  - standard 9-point stencil (8 in the diagonal, -1 off-diagonal)
  - independent of θ, all connections are of the same strength, hence $A_s = A$

Illustration of the C-AMG Coarse Grid Selection (Pass 1)



(a) each node gets the number of its off-diagonal connections as weight (number of off-diagonal non-zeroes, after removing small (and irrelevant) entries)

(b) a point of maximum weight is selected as a C-point



(c) all neighbours of the new C-point that have not yet been assigned to either F or C become F-points

(d) each new F-point causes an additional weight of 1 to all neighbours that have not yet been assigned

(...) bottom row: go on until all points are either C-points or F-points

Interpolation

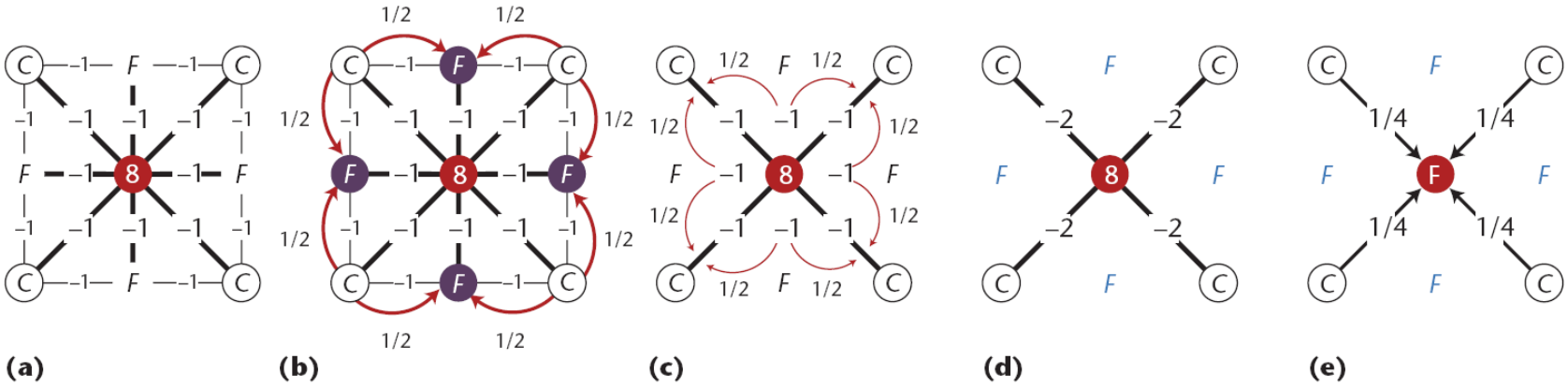
- starting point again: algebraically smooth error means dominance of small eigenmodes v
- because of $r^T r = e^T A^2 e \approx v^T A^2 v = \lambda^2 \ll 1$ (length of v normed to 1!), this also implies small residuals
- To understand interpolation, assume $r = Av = 0$ or $r = Ae = 0$, resp. (with an error e essentially consisting of small eigenmodes). For an F-point i, we get

$$a_{i,i}e_i = -\sum_{j \in C_i} a_{i,j}e_j - \sum_{j \in F_i^s} a_{i,j}e_j - \sum_{j \in F_i^w} a_{i,j}e_j \,,$$

  where
  - C_i denotes the C-points with a strong relation to i (interpolation in i will be based on these only),
  - F_i^s denotes the F-points with strong connection to i, and
  - F_i^w denotes all points with weak connection to i.
- The crucial point is now to add the error components e_j from the second and third partial sum above either to the points from C_i or directly to point i.
- 2 examples:

Prolongation in C-AMG – Example 1

standard 9-point finite element stencil



(a) standard 9-point finite element stencil (or matrix relations, resp.)

(b) strongly connected F-points are interpolated from the neighbouring C-points (weights according to the matrix entries)

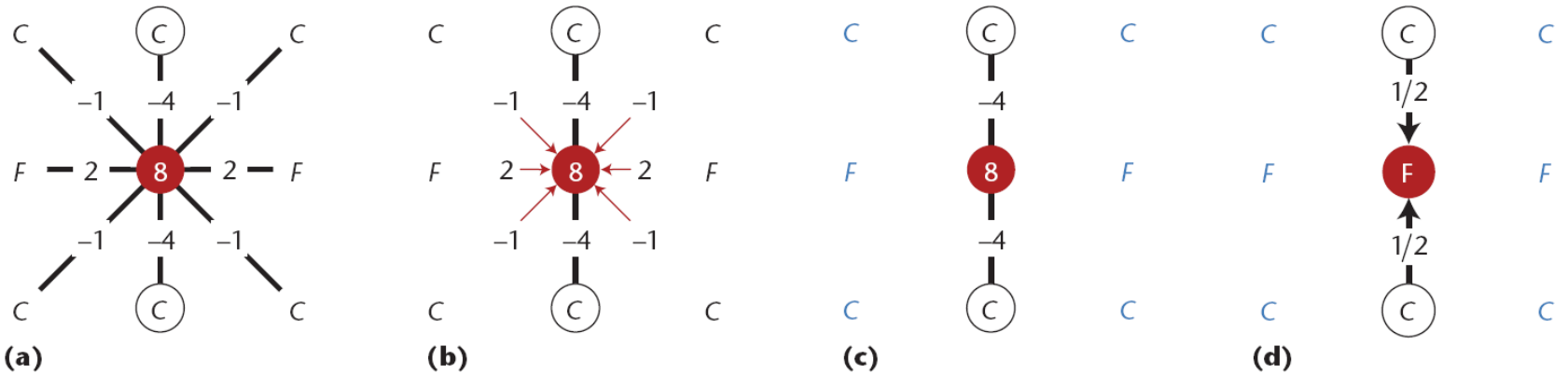
(c) then, the strong connections to the F-points are added to the relations to the C-points (according to the interpolation weights)

(d) from that, the collpased stencil results ...

(e) ... and from that the interpolation rule

Prolongation in C-AMG – Example 2

anisotropic 9-point finite element stencil



(a) anisotropic 9-point finite element stencil (or matrix relations, resp.)

(b) weak relations are included into the diagonal element

(c) from that, the collpased stencil results ...

(d) ... and from that the interpolation rule

Restriction and Coarse Grid Operator in C-AMG

Here, not that much has to be done:

- **restriction**: is defined as transposed matrix of the prolongation operator in C-AMG, i. e.

$$R := P^T$$

- **coarse grid operator**:
  - Here, there are less options than in the geometric multigrid case, where the given PDE can just be discretized on the coarse grid again. In contrast to that, we do not know the origin of A and, therefore, cannot transfer its generation process to other resolutions.
  - That's why, in C-AMG, always the so-called **Galerkin operator** is used, i. e.

$$A_c := P^T A P \,.$$

With that, the set of algorithmic components is complete, and we can construct different multigrid cycles – as before in the geometric case.

Performance Features

Fine grid	Iterations	Convergence factor	Coarse grids	Grid complexity*	Operator complexity†	Setup time‡	Solve time
31×31	9	0.19	5	1.6	1.7	—	—
61×61	10	0.23	6	1.6	1.6	0.01	0.02
121×121	9	0.23	8	1.6	1.7	0.05	0.07
241×241	9	0.23	9	1.6	1.7	0.25	0.32
481×481	9	0.23	12	1.7	1.7	1.02	1.27
961×961	11	0.29	13	1.7	1.7	4.42	6.28

Note: *The code used strength threshold $\theta = 0.4$ with $\nu_1 = \nu_2 = 1$ step of C-F Gauss-Seidel, and iterated until the relative residual was below 10^{-9}.* *Grid complexity *is the total number of grid points on all grids divided by the number of grid points on the fine grid.* †Operator complexity *is the total number of nonzeroes in the linear operators on all grids divided by the number of nonzeroes in the fine grid operator.* ‡Setup time *is the time required to choose coarse grids and build interpolation, restriction, and coarse-grid operators.*

References

This chapter's figures and tables have been taken from:
Robert D. Falgout. An introduction to algebraic multigrid computing. *Computing in Science and Engineering*, 8(6):24-33, 2006.