Scientific Computing II

Towards Multigrid Methods

Michael Bader

Summer term 2012

Part I: Smoothing Property of Relaxation Methods

Part II: Multigrid Methods

Part II: Components of Multigrid Methods

Part I

Smoothing Property of Relaxation Methods

Convergence of Relaxation Methods

Observation

- slow convergence
- high frequency error components are damped more efficiently
- smooth error components are reduced very slowly

Convergence Analysis

- remember iteration scheme: $x^{(i+1)} = Mx^{(i)} + Nb$
- derive iterative scheme for the error $e^{(i)} := x - x^{(i)}$:

$$e^{(i+1)} = x - x^{(i+1)} = x - Mx^{(i)} - Nb$$

- for **consistent** scheme, x is a fixpoint of the iteration ($x = Mx - Nb$)
- hence:

$$\begin{aligned} e^{(i+1)} &= Mx + Nb - Mx^{(i)} - Nb = Me^{(i)} \\ e^{(i)} &= M^i e^{(0)}. \end{aligned}$$

Convergence Analysis (2)

- iteration equation for error: $e^{(i)} = M^i e^{(0)}$
- consider eigenvalues λ_j and eigenvectors v_j of iteration matrix M:

$$Mv_j = \lambda_j v_j \quad \Rightarrow \quad M\underbrace{\Big(\sum_j \alpha_j v_j\Big)}_{=:e^{(0)}} = \sum_j \lambda_j \alpha_j v_j$$

$$\Rightarrow \quad M^i e^{(0)} = M^i\Big(\sum_j \alpha_j v_j\Big) = \sum_j \lambda_j^i \alpha_j v_j$$

- convergence, if all $|\lambda_j| < 1$
- speed of convergence dominated by largest λ_j

The Model Problem

1D Poisson equation:

- $-u''(x) = f(x)$ on $\Omega = (0,1)$
- $u = 0$ on $\partial\Omega$ (hom. Dirichlet boundaries)
- discretised on a uniform grid of mesh size $h = \frac{1}{n}$
- compute approximate values $u_j \approx u(x_j)$
  at grid points $x_j := jh$, with $j = 1, \ldots, (n-1)$
- system matrix A_h built from 3-point stencil:

$$\frac{1}{h^2}\begin{bmatrix} -1 & 2 & -1 \end{bmatrix}$$

- A_h a tridiagonal $(n-1) \times (n-1)$-matrix

The Smoothing Property

Eigenvalues and -vectors of A_h:

- eigenvalues: $\lambda_k = \frac{4}{h^2}\sin^2\left(\frac{k\pi}{2n}\right) = \frac{4}{h^2}\sin^2\left(\frac{k\pi h}{2}\right)$
- eigenvectors: $v^{(k)} = \big(\sin(k\pi j/n)\big)_{j=1,\ldots,n-1}$
  – both for $k = 1, \ldots, (n-1)$

For Jacobi relaxation:

- iteration matrix $M = I - D_A^{-1}A = I - \frac{h^2}{2}A$
- eigenvalues of M: $\mu_k := 1 - 2\sin^2\left(\frac{k\pi h}{2}\right)$
- $|\mu_k| < 1$ for all k, but $|\mu_k| \approx 1$ if $k = 1$ or $k = n-1$
- $\mu_1 \in \mathcal{O}(1 - h^2)$: slow convergence of smooth errors
- $\mu_{n-1} \approx -1$: "sign-flip" (but slow reduction) of "zig-zag" error components
- convergence factor determined by $\mathcal{O}(1 - h^2)$

The Smoothing Property

Eigenvalues and -vectors of A_h:

- eigenvalues: $\lambda_k = \frac{4}{h^2}\sin^2\left(\frac{k\pi}{2n}\right) = \frac{4}{h^2}\sin^2\left(\frac{k\pi h}{2}\right)$
- eigenvectors: $v^{(k)} = \left(\sin(k\pi j/n)\right)_{j=1,\ldots,n-1}$
  – both for $k = 1, \ldots, (n-1)$

For weighted Jacobi relaxation:

- iteration matrix $M = I - \omega D_A^{-1} A = I - \frac{h^2}{2}\omega A$
- eigenvalues of M: $1 - 2\omega \sin^2\left(\frac{k\pi h}{2}\right)$
- $\mu_1 \in \mathcal{O}(1 - h^2)$: slow convergence of smooth errors
- $\mu_{n-1} \approx 0$ for $\omega = \frac{1}{2}$; $\mu_{n-1} \approx -\frac{1}{3}$ for $\omega = \frac{2}{3}$
  thus quick reduction of high-frequency errors
- convergence determined by $\mathcal{O}(1 - n^{-2})$
  (slower than normal Jacobi due to ω)

The Smoothing Property (2)

"Fourier mode analysis"

- decompose the error $e^{(i)}$ into eigenvectors (for 1D Poisson: $\sin(k\pi x_j)$,)
- determine convergence factors for "eigenmodes"

Observation for weighted Jacobi and Gauß-Seidel:

- 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.

$\Rightarrow$ **"smoothing property"**

The Smoothing Property (2)

“Fourier mode analysis”

- decompose the error $e^{(i)}$ into eigenvectors (for 1D Poisson: $\sin(k\pi x_j)$,)
- determine convergence factors for “eigenmodes”

Another Observation:

- the smoothest (slowest converging) component corresponds to the smallest eigenvalue of A ($k = 1$)
- remember residual equation: $Ae = r$:
  if $e = v^{(1)}$, then $r = \lambda_1 v^{(1)}$

$\Rightarrow$ **“small residual, but large error”**

Part II

Multigrid Methods

Multigrid Idea No. 1

- additional result from convergence analysis: "high-frequency error" is relative to mesh size
- on a sufficiently coarse grid, even very low frequencies can be "high-frequency" (if the mesh size is big)

"Multigrid":

- use multiple grids to solve the system of equations
- on each grid, a certain range of error frequencies will be reduced efficiently

Nested Iteration

Solve the problem on a coarser grid:

- will be comparably (very) fast
- can give us a good initial guess:
- **nested iteration**/ "poor man's multigrid"

Algorithm:

1. Start on a very coarse grid with mesh size $h = h_0$; guess an initial solution x_h
2. Iterate over $A_h x_h = b_h$ using **relaxation** method $\Rightarrow$ approximate solution x_h
3. **interpolate** the solution x_h to a finer grid $\Omega_{h/2}$
4. proceed with step 2 (now with mesh size $h := h/2$) using interpolated $x_{h/2}$ as initial solution

Multigrid Idea No. 2

Observation for nested iteration:

- error in interpolated initial guess also includes low frequencies
- relaxation therefore still slow
- can we go "back" to a coarser grid later in the algorithm?

$\Rightarrow$ **Idea No. 2: use the residual equation**:

- solve $Ae = r$ on a coarser grid
- leads to an approximation of the error e
- add this approximation to the fine-grid solution

A Two-Grid Method

Algorithm:

1. **relaxation/smoothing** on the fine level system $\Rightarrow$ solution x_h
2. compute the **residual** $r_h = b_h - A_h x_h$
3. **restriction** of r_h to the coarse grid Ω_H
4. compute a **solution** to $A_H e_H = r_H$
5. **interpolate** the coarse grid solution e_H to the fine grid Ω_h
6. add the resulting **correction** to x_h
7. again, **relaxation/smoothing** on the fine grid

Correction Scheme – Components

smoother: reduce the high-frequency error components, and get a smooth error

restriction: transfer residual from fine grid to coarse grid, for example by
- injection
- (full) weighting

coarse grid equation: (acts as) discretisation of the PDE on the coarse grid

interpolation: transfer coarse grid solution/correction from coarse grid to fine grid

The Multigrid V-Cycle

1. smoothing on the fine level system
   $\Rightarrow$ solution x_l
2. compute the residual $r_l = b_l - A_l x_l$
3. restriction of r_l to the coarse grid Ω_{l-1}
4. solve coarse grid system $A_{l-1} e_{l-1} = r_{l-1}$ by a **recursive call to the V-cycle algorithm**
5. interpolate the coarse grid solution e_{l-1} to the fine grid Ω_l
6. add the resulting correction to x_l
7. **post-smoothing** on the fine grid

V-Cycle – Implementation

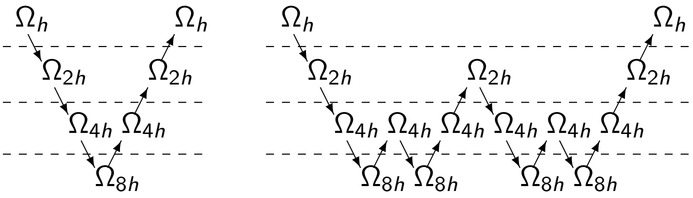
- on the coarsest grid: direct solution
- number of smoothing steps is typically very small (1 or 2)

Cost (storage and computing time):

- 1D: $c \cdot n + c \cdot n/2 + c \cdot n/4 + \ldots \leq 2c \cdot n$
- 2D: $c \cdot n + c \cdot n/4 + c \cdot n/16 + \ldots \leq 4/3c \cdot n$
- 3D: $c \cdot n + c \cdot n/8 + c \cdot n/64 + \ldots \leq 8/7c \cdot n$
- overall costs are dominated by the costs of the finest grid

The W-Cycle

- perform **two** coarse grid correction steps instead of one



(V-cycle and W-cycle)

- more expensive
- useful in situations where the coarse grid correction is not very accurate

The Full Multigrid V-Cycle (FMV)

Recursive algorithm:

- combines nested iteration and V-cycle
- (recursively!) perform an **FMV-cycle** on the next coarser grid to get a good initial solution
- interpolate this initial guess to the current grid
- perform a **V-cycle** to improve the solution

$$\begin{tikzcd}
 & & & & & & & & & \Omega_h \arrow[dr] & & & & & & \Omega_h \\
 & & & & \Omega_{2h} \arrow[dr] & & & & \Omega_{2h} \arrow[ur] & & \Omega_{2h} \arrow[dr] & & & & \Omega_{2h} \arrow[ur] & \\
 & \Omega_{4h} \arrow[dr] & & \Omega_{4h} \arrow[ur] & & \Omega_{4h} \arrow[dr] & & \Omega_{4h} \arrow[ur] & & & & \Omega_{4h} \arrow[dr] & & \Omega_{4h} \arrow[ur] & & \\
\Omega_{8h} \arrow[ur] & & \Omega_{8h} \arrow[ur] & & & & \Omega_{8h} \arrow[ur] & & & & & & \Omega_{8h} \arrow[ur] & & &
\end{tikzcd}$$

Speed of Convergence

- fastest method around
  (if all components are chosen carefully)
- **"textbook multigrid efficiency"**:

$$\left\| e^{(m+1)} \right\| \leq \gamma \left\| e^{(m)} \right\| ,$$

where convergence rate $\gamma < 1$ (esp. $\gamma << 1$) is independent of the number of unknowns

$\Rightarrow$ constant number of multigrid steps to obtain a given number of digits

$\Rightarrow$ overall computational work increases only linearly with the number of unknowns

Convergence Rates (2)

For the “Model Problem” (i.e., Poisson Problem):

- $\mathcal{O}(n)$ to solve up to “level of truncation”
- **“level of truncation”**: $\mathcal{O}(h^2)$ (discretisation error)
- $\mathcal{O}(n)$ is achieved by FMV-Cycle (1 or 2 cycles sufficient)

For Other Problems:

- OK for strongly elliptic problems
- multigrid variants for non-linear problems, parabolic/hyperbolic, . . .
- achieving “textbook efficiency” usually a demanding task

Part III

Components of Multigrid Methods

Smoothers

For the Poisson problem (see tutorials):

- Gauss-Seidel
- red-black Gauss-Seidel
- damped ($\omega = \frac{2}{3}$) Jacobi
- how about Jacobi (non-weighted) and SOR?
  $\rightarrow$ do not work well
  (do not smooth high frequencies efficiently)

Smoothers (2) – other problems

anisotropic Poisson eq.: $u_{xx} + \epsilon u_{yy} = f$

- Strong dependency in x-direction, weak dependency in y-direction
- Good smoothing of the error only in x-direction
- "semi-coarsening" (coarsen only in "smooth" direction) $\rightarrow$ see tutorials
- line smoothers: perform a column-wise Gauss-Seidel = solve each "column" (or row) simultaneously (direct, tridiagonal solver):

$$u_{i-1,j}^{(n+1)} - 4u_{ij}^{(n+1)} + u_{i+1,j}^{(n+1)} = f_{ij} - u_{i,j-1}^{(n)} - u_{i,j+1}^{(n)}$$

Smoothers (3) – other problems

1D Convection-Diffusion eq.: $\epsilon u_{xx} + u_x = f$, $\epsilon \ll 1$

- “upwind discretisaton”:
  $\frac{\epsilon}{h^2}(u_{n-1} - 2u_n + u_{n+1}) + \frac{1}{h}(u_n - u_{n-1}) = f_n$
- (weighted) Jacobi and red-black Gauss-Seidel: no smoothing, basically updates one grid point per iteration
- Gauss-Seidel (relaxation from “left to right”): almost an exact solver
- in general: Gauss-Seidel smoothing in “downwind” order
  $\rightarrow$ difficult to do in 2D and 3D

Interpolation (aka "Prolongation")

For Poisson problem:

- (bi-)linear interpolation:
  in 1D: resembles homogeneous ($f = 0$) solution
- constant (in general too small approximation order):
  sometimes used for cell-based coarsening (unknowns located in cell centers)
- quadratic, cubic, etc.:
  often too costly, more smoothing steps are cheaper and can eliminate the disadvantage of a lower order interpolation
- **but:** in FMV-cycle interpolation to finer grid (after a completed V-cycle) should be higher-order

Interpolation – Matrix Notation

For linear interpolation (1D):

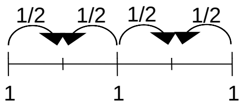
$$\begin{pmatrix} \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \\ 0 & 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}(0 + x_1) \\ x_1 \\ \frac{1}{2}(x_1 + x_2) \\ x_2 \\ \frac{1}{2}(x_2 + x_3) \\ x_3 \\ \frac{1}{2}(x_3 + 0) \end{pmatrix}$$

Notation: $I_{2h}^{h} x_{2h} = x_h$ or $P_{2h}^{h} x_{2h} = x_h$

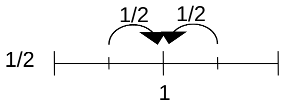
Restriction

For Poisson problem:

- "injection": pick values at corresp. coarse grid points
- "full weighting" = transpose of bilinear interpolation (safer, more robust convergence), see illustration below for the 1D case



linear interpolation



full weighting

Restriction – Matrix Notation

For full weighting (1D):

$$\begin{pmatrix} \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}(x_1 + 2x_2 + x_3) \\ \frac{1}{2}(x_3 + 2x_4 + x_5) \\ \frac{1}{2}(x_5 + 2x_6 + x_7) \end{pmatrix}$$

Notation: $I_h^{2h} x_h = x_{2h}$ or $R_h^{2h} x_h = x_{2h}$

Coarse Grid Operator

Two main options:

1. discretise PDE on grid Ω_h to obtain A_h
2. "Galerkin approach": $A_{2h} := R_h^{2h} A_h P_{2h}^h$
   $\rightarrow$ compare effect on vector x_{2h}:

$$A_{2h} x_{2h} := R_h^{2h} A_h P_{2h}^h x_{2h}$$

   $\rightarrow$ evaluate from right to left:
   - interpolate $x_2 h$ to $\hat{x}_h := P_{2h}^h x_{2h}$
   - apply fine-grid operator A_h to interpolated $\hat{x}_h$
   - restrict resulting matrix-vector product to Ω_{2h}

Exercise:

- Compute $A_{2h} := R_h^{2h} A_h P_{2h}^h$ for
  $A_h := \frac{1}{h^2} \text{tridiag}(-1, 2, -1)$

Literature

General:

- Gander, Hrebicek: *Solving Problems in Scientific Computing Using Maple and MATLAB*.
- Golub, Ortega: *Scientific Computing and Differential Equations*.
- Dongarra, et. al.: *Numerical linear algebra for high-performance computers*.

Literature (2)

Multigrid:

- Briggs, Henson, McCormick: *A Multigrid Tutorial* (2nd ed.).

Conjugate Gradients:

- Shewchuk: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*.