Multigrid

C. T. Kelley
NC State University
tim.kelley@ncsu.edu
Theory for Jacobi Smoother

We showed that a Jacobi iteration for the central difference discretization $A^h u^h = f^h$ of

$$-u'' = f, \quad 0 < x < 1$$

with $u(0) = u(1) = 0$

- damped the error in the high frequencies fast, and
- the error in the low frequencies very slowly,
- so the iteration is a smoother,

which explains the methods poor convergence properties. MG exploits this by attacking the smooth components on coarser grids.
Recall that

\[
A^h = \frac{1}{h^2} \begin{pmatrix}
2 & -1 & 0 & \ldots & 0, & 0 \\
-1 & 2 & -1 & 0 & \ldots & 0 \\
0 & -1 & 2 & -1 & \ldots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \ldots, & 0 & -1 & 2 & -1 \\
0 & \ldots, & \ldots, & 0 & -1 & 2
\end{pmatrix}
\]

and

\[
f^h = (f(x_1), \ldots, f(x_N))^T.
\]
The eigenvectors of $A^h$ are $\{u_n\}_{n=1}^N$ where

$$u_n = (\xi_1^n, \ldots, \xi_N^n)^T$$

and

$$\xi_i^n = \sqrt{2/h} \sin(ni \pi h),$$

with eigenvalues

$$\lambda_n = h^{-2} 2 (1 - \cos(\pi nh)) = \pi^2 n^2 + O(h^2) \text{ for small } n$$

We proved this with elementary trigonometry.
A better smoother

Consider the damped Jacobi iteration:

$$x_{n+1} = (1 - \omega)x_n + \omega D^{-1}(L + U)x_n + D^{-1}b$$

with iteration matrix $M^h_{DJ} = (1 - \omega)I + \omega M_{JAC}$, where

$$M^h_{JAC} = D^{-1}(L + U).$$

So $\omega = 1$ is Jacobi and $\omega = 0$ is nothing. What's the optimal $\omega$ if we want to damp the high-frequency terms?
Eigen-analysis

We showed that the eigenvalues of $M_{JAC}$ were

$$\mu_n = 1 - (h^2/2)\lambda_n$$

with the same eigenvectors as $A^h$. Similarly

$$\mu_n = (1 - \omega) + \omega(1 - (h^2/2)\lambda_n) = 1 - (h^2/2)\omega\lambda_n$$

$$1 - \omega(1 - \cos(\pi nh))$$
Now suppose $n \geq N/2$ (high frequency), then

$$
\mu_n = 1 - \omega(1 - \cos(\pi nh)) = 1 - \omega + \omega \cos(\pi nh)
$$

so

$$
1 - 2\omega \leq \mu_n \leq 1 - \omega.
$$

Minimize $|\mu_n|$ to see that the optimal value of $\omega$ is $2/3$. So

$$
|\mu_n| \leq 1/3 \text{ for } N/2 \leq n \leq N - 1
$$
Idealized Two-Grid Method: I

Notation:

- $h = 1/(N - 1)$, $N - 1$ even.
- $\Omega^h$: space of grid functions with step size $h$
  so $u^h \in \Omega^h$
- $I_h^{h'}$: intergrid transfer from $\Omega^h \rightarrow \Omega^{h'}$
- $P^h_H$ and $P^h_L$ project onto low and high frequencies.
Idealized Two-Grid Method: II

For Now: We will define $I_{2h}^h$ and $I_h^h$ by Fourier truncation.

\[ u = \sum_{n=1}^{N-1} \alpha_n u_n^h \in \Omega^h \]

\[ I_{2h}^h u = \sum_{n=1}^{(N-1)/2} \alpha_n u_n^{2h} \in \Omega^{2h} = I_{2h}^h P_L u \]

and

\[ w = \sum_{n=1}^{(N-1)/2} \alpha_n u_n^{2h} \in \Omega^{2h} \]

\[ I_{2h}^h w = \sum_{n=1}^{(N-1)/2} \alpha_n u_n^h \in \Omega^h \]

Note: $I_{2h}^h P_L A^u = A^{2h} I_{2h}^h P_L u$ and $I_{2h}^h I_{2h}^h = P_L$
Idealized Two-Grid Method: III

Simple method:

- Let $u_c \in \Omega^h$
- Take one weighted Jacobi iteration on $\Omega^h$ to obtain $u_{1/2}$
- Compute the residual $r^h = f^h - A^h u_{1/2}$
- Compute the coarse grid correction $w = (A^{2h})^{-1} l_h^{2h} r^h$
- $u_{3/4} = u_{1/2} + l_h^{1/2} w$
- Take one weighted Jacobi iteration on $\Omega^h$ to obtain $u_+$
Idealized Two-Grid Method: IV

If we did everything on $\Omega^h$, then

$$w = A^{-1}r = A^{-1}(f^h - A^h u) = A^{-1}f^h - u = u^h - u$$

so

$$u + w = u^h$$

solves the problem. We will show that the two grid method does pretty well.
Idealized Two-Grid Method: V

Apply weighted Jacobi and let $e = u^h - u$.

$$\|P_H e_{1/2}\| \leq \frac{1}{3}\|P_H e_0\|$$

so

$$r^h = f^h - A^h u_{1/2} = P_H(f^h - A^h u_{1/2}) + P_L(f^h - A^h u_{1/2})$$

$$= A^h P_H e_{1/2} + P_L A^h e_{1/2}.$$

We are just about done because ...
The coarse grid correction eliminates the low frequency errors.

- $I^h_{2h} P_L A^h e_{1/2} = A^{2h} P_L e_{1/2}$, so
- $(A^{2h})^{-1} I^h_{2h} P_L A^h e_{1/2} = I^h_{2h} P_L e_{1/2}$,
- and $w = I^h_{2h} I^h_{2h} P_L e_{1/2} = P_L e_{1/2}$.

Hence

$$e_{3/4} = e_{1/2} - P_L e_{1/2} = P_H e_{1/2}$$

and hence $\|e_+\| \leq (1/9)\|e_c\|$
Algorithmic Description

\[ v^h \leftarrow TG(v^h, f^h) \]
\[ v^h \leftarrow S(v^h, fh) \]
\[ r^h = f^h - A^h v^h_{1/2} \]

Solve \( A^{2h} e^{2h} = l_{2h}^h r^h \) exactly.

\[ u^h_+ = u^h_{1/2} + l_{2h}^h e^{2h} \]
\[ v^h \leftarrow S(v^h, fh) \]
Algorithmic Description: MG

Multigrid replaces the coarse mesh solve with two-grid iteration.

\( h \): desired (finest) grid; \( H \): coarsest grid.

\[ v^h \leftarrow V^h(v^h, f^h) \]

\textbf{if} \( h = H \) \textbf{then}

Solve \( A^h v^h = f^h \) exactly.

\textbf{else}

\[ v^h \leftarrow S(v^h, f^h) \]
\[ r^h = f^h - A^h v^h \]
\[ e^{2h} = V^{2h}(0, l_2^h r^{2h}) \]
\[ u_+^h = u_{1/2}^h + l_{2h}^h w \]

\[ v^h \leftarrow S(v^h, f^h) \]

\textbf{end if}

This is a \( V \)-cycle. The convergence rate remains unchanged.
Observations

- Intergrid transfers in frequency cost $O(N \log N)$ work (prove this with FFT).
- MG is faster than two-grid because the exact solves live only on the coarsest mesh.
- This method only works for the simplest problem where
  - we can connect the eigenfunctions of $A$ to those of $R_{TG}$
  - the intergrid transfers have the same eigenfunctions
  - $I_{2h}^h I_{2h}^{2h} = P_L^h$
- In general, we have none of those things.
Inhomogenous BC
Neumann BC
Intergrid Transfers: 1D

- Nested grids: $x_i^h = ih; x_{2i}^h = x_i^{2h}$
- $I_{2h}^h$: linear interpolation;

\[
(l_{2h}^{h}v^{2h})_{2i} = v_i^{2h} \quad \text{and} \quad (l_{2h}^{h}v^{2h})_{2i-1} = (v_i^{2h} + v_{i-1}^{2h})/2.
\]

- $I_{h}^{2h}$: full weighting

\[
(l_{h}^{2h}v^{h})_i = (v_{2i-i}^{h} + 2v_{2i}^{h} + v_{2i+1}^{h})/4.
\]

except on the boundary.
Two Space Dimensions

Fine-to-Coarse full weighting

\[(i^h_{2i,j}v^h_{2i,j}) = (v^h_{2i-1,j} + v^h_{2i+1,j}) + v^h_{2i,j-1} + v^h_{2i,j+1} + 4v^h_{2i,2j})/8\]

Coarse-to-fine: Linear interpolation, see board
Matrix Representation: 1

- $N = 2^p - 1$ internal grid points
- Internal grid points: $\{x^h_i\}_{i=1}^N$; $h = 1/(N + 1)$
- Boundary grid points: $x^h_0 = 0$, $x^h_{N+1} = 1$
- $I_{2h}^h$ is $1 + (N - 1)/2 \times N$
- Other option: restriction by injection: $(I_{2h}^h v)_i = v_{2i}$
Matrix Representation: II

\[ I_{2h}^h = \frac{1}{2} \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 & 0 \\ 2 & 0 & \ldots & 0 & 0 & 0 \\ 1 & 1 & \ldots & 0 & 0 & 0 \\ 0 & 2 & \ldots & 0 & 0 & 0 \\ 0 & 1 & 1 & \ldots & 0 & 0 \\ 0 & 0 & 2 & \ldots & 0 & 0 \\ 0 & 0 & 1 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & 1 \\ 0 & 0 & 0 & \ldots & 0 & 2 \\ 0 & 0 & 0 & \ldots & 0 & 1 \end{pmatrix}, \quad I_{h}^{2h} = \frac{1}{2} (I_{2h}^h)^T. \]
Interpolatory intergrid transfers have aliasing effects.

You get the convergence rate $1/9$ you’d expect.
Intergrid Transfers for 1-D: $l^h_{2h}$

\[ w = l^h_{2h}(u) \]

\[ N = h^{-1} - 1; \quad h_2 = 2h; \quad N_2 = h_2^{-1} - 1 \]

\[ \text{for } i = 0 : N_2 \text{ do} \]

\[ w_{2i} = u_i \]

\[ w_{2i+1} = (u_i + u_{i+1})/2 \]

\[ \text{end for} \]
Intergrid Transfers for 1-D: $I_{h}^{2h}$

\[ w = I_{h}^{2h}(u) \]
\[ N = h^{-1} - 1; \quad h_2 = 2h; \quad N_2 = h_2^{-1} - 1 \]
\[ w_0 = u_0; \quad w_{N_2+1} = u_{N+1} \]
\[ \text{for } i = 1 : N_2 \text{ do} \]
\[ w_i = \frac{(u_{2i-1} + 2u_{2i} + u_{2i+1})}{4} \]
\[ \text{end for} \]
Notation: Smooth $\nu$ times at level $h$ with initial iterate $v^h$

$$v^h \leftarrow S^\nu(v^h, f^h)$$

for $is = 1 : \nu$ do

for $i = 1 : N$ do

$$w_i = h^2(f_i^h + v_{i-1}^h + v_{i+1}^h)/2$$

end for

for $i = 1 : N$ do

$$v_i^h = (1 - \omega)v_i^h + \omega w_i$$

end for

end for
The V-cycle

\[ v^h \leftarrow V^h(v^h, f^h) \]

**if** \( h = H \) **then**

Solve \( A^h v^h = f^h \) to high accuracy.

**else**

\[ v^h \leftarrow S^{\nu_1}(f^h, \nu_1) \]

Compute \( f^{2h} = I^{2h}_{h}(f^h - A^h v^h) \)

\[ v^{2h} = 0 \]

\[ v^{2h} \leftarrow V^{2h}(v^{2h}, f^{2h}) \]

\[ v^h \leftarrow v^h + I^{h}_{2h} v^{2h} \]

\[ v^h \leftarrow S^{\nu_2}(f^h, \nu_1) \]

**end if**
Typical choice $\nu_1 = \nu_2 = 1$.  
V-cycle for high-order term is a good preconditioner for PDEs.  
V-cycle reduces the error by fixed amount independent of $h$.  
We consider very easy problems and simple grids  
Solve step for $\Omega^H$ can be iterative.  
There’s more, much more  
Other smoothers (Gauss-Seidel, . . . )  
Algebraic multigrid (AMG)  
MG for Navier-Stokes, combustion . . .  
Non isotropic flows/grids
Matlab Example: Coarse Mesh Solve

You can avoid the exact solve on the coarse mesh with no loss. This example shows that:

- Convergence rate is independent of $h$
- You can simply do a few (24) smoothing steps at the coarse level
- You are not restricted to Poisson’s equation
Helmholtz in 1-D

\[-u'' + \sigma u = f; u(0) = u(1) = 1.\]

Arrange things so the exact solution is

\[u(x) = e^x \sin(\pi x).\]

Comparisons from 1-D codes

- Several fine levels: \( h = 2^{-l}, 5 \leq l \leq 10 \)
- Exact solve vs 24 smooths on Coarse mesh.
Nested Iteration or Full Multigrid (FMG): $h = 2^{-p}H$

Solve $A^Hv^H = f^H$ on $\Omega^H; h = H$.

for $il = 1 : p$ do
  $h = h/2$
  $v^h = I_{2h}^h v^{2h}$
  $v^h \leftarrow V^h(v^h, f^h)$
end for

Why is this the same as the nested iteration from last week?
Refer to pages 42 and 43.
You do not want to repeatedly allocate storage for the solutions on the various grids. You can avoid this by creating a large structure which preallocates the storage. Some of the examples in the 1-D codes do this.
Comments

- Optimal complexity (see next slide)
- Complexity leads to debugging tool
- FMG is a solver. If error is $O(h^2)$ then one V-cycle per level will suffice.
- A V-cycle can be a preconditioner
A^h \text{ discrete Laplacian in } R, N \text{ interior grid points}

- Cost of smoother, mat-vec, intergrid transfer = 3N
- Cost or residual, correction = N
- Cost of solve at level \( H \) is \( V_0 \)
- Then, cost of V-cycle at level \( h_j = 2^{-j}H \) is

\[
V_j = 3(N_j(\nu_1 + \nu_2) + 2N_j) + 2N_j + V_{j-1}
\]

where \( N_j = (1/h_j) - 1 \approx 2N_{j-1} \).
Complexity Example: II

So, if $\nu_1 = \nu_2 = 1$, then

$$V_j = 14N_j + V_{j-1} = 14 \sum_{i=1}^{j} N_l + V_0$$

$$\leq 28N_j + V_0$$

Cost of FMG for $h = 2^{-p}H; N = N_p$

$$FMG_i \leq \sum_{l=0}^{p} N_l + V_l \leq \sum_{l=0}^{p} 29N_l + pV_0$$

$$\leq 58N + pV_0 = O(N).$$
Testing Complexity

In one dimension, you should see the computing time double if $h \rightarrow h/2$.
In practice, MATLAB-sized problems take so little time that you will find this hard to measure.