MA 580; Numerical Analysis I

C. T. Kelley
NC State University
tim_kelley@ncsu.edu
Version of November 28, 2016

NCSU, Fall 2016
Part VIe: Residual Correction

Residual Correction

Suppose you have a linear solver: $\mathbf{S}$ where

$$\mathbf{S}(\mathbf{b}) \approx \mathbf{A}^{-1}\mathbf{b}.$$

Residual correction methods try to overcome problems with the solver by using it iteratively.

Algorithmic Sketch

$\mathbf{x} = \mathbf{x}_0$
$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$
while $\|\mathbf{r}\| > \tau$ **do**
  $\mathbf{e}_s = \mathbf{S}(\mathbf{r})$
  $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{e}_s$
  $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$
end while

When $\mathbf{S}(\mathbf{b}) = \mathbf{A}^{-1}\mathbf{b}$ you converge in one iteration because

$$\mathbf{e}_s = \mathbf{A}^{-1}\mathbf{r} = \mathbf{A}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}) = \mathbf{x}^* - \mathbf{x} = \mathbf{e}.$$

Classic Iterative Improvement

Suppose **S** is a single precision solve?

$$\mathbf{S}(\mathbf{b}) = \mathbf{U}_s^{-1}\mathbf{L}_s^{-1}\mathbf{b}$$

where $\mathbf{L}_s$ and $\mathbf{U}_s$ are the **single precision** LU factors of **A**.

```
AS=single(A);
[LS, US]=lu(AS);
```

will do the job.

The rules and the facts

- Rule: computed residuals in full precision
- Fact: relative error from solve will be single precision at best
- Fact: cost of LU should be reduced by 1/8
- Fact + Rule: you have to pay attention to the language-dependent conversions from single to double and back.

Precision Management

To compute $\mathbf{S}(\mathbf{b})$ you need get prepared

- Convert $\mathbf{A}$ to single to get $\mathbf{A}_s$
- Compute the single precision LU factorization $\mathbf{L}_s\mathbf{U}_s = \mathbf{A}$

You use the preparation by

- Compute $\mathbf{x}_s = \mathbf{U}_s^{-1}\mathbf{L}_s^{-1}\mathbf{b}$ in single.
- Convert $\mathbf{x}_s$ to double to get $\mathbf{x}$.
- Compute $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$ in double
- Compute $\mathbf{e}_s = \mathbf{U}_s^{-1}\mathbf{L}_s^{-1}\mathbf{r}$ in single
  You will probably not have to convert $\mathbf{r}$ to single to do this
- Convert $\mathbf{e}_s$ to double to get $\mathbf{e}_d$
- Compute $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{e}_d$ in double.

Some Matlab

Preparation:

```
AS=single(A);
[LS,US]=lu(AS);
```

Computing the correction:

```
xs=US\(LS\ b); xd=double(xs);
rd=b-A*xd;
es=US\(LS\rd); ed=double(es);
xd=xd+ed;
```

The iteration

```
AS=single(A);
[LS,US]=lu(AS);
xs=US\(LS\b);
xd=double(xs); rd=b-A*xd;
while norm(rd,inf) > 1.d-13
   es=US\(LS\rd);
   ed=double(es);
   xd=xd+ed;
   rd=b-A*xd;
end
```

Some Results: Method of Synthetic Solutions

Begin with

```
n=100;
A=rand(n,n); xe=rand(n,1); b=A*xe;
```

$\kappa_2(\mathbf{A}) \approx 1900.$

relative error	relative residual
1.77e-05	1.52e-07
1.61e-11	2.25e-13
2.02e-14	1.02e-16

Analysis via Back-of-the-Envelope:I

Suppose δ_s is the relative error the single precision computations.

- So $\mathbf{x} = \mathbf{U}_s^{-1}\mathbf{L}_s^{-1}\mathbf{b} = \mathbf{x}^* - \mathbf{e}$,
- $\mathbf{e} = O(\delta_s)$,
- and $\mathbf{e}_d = \mathbf{U}_s^{-1}\mathbf{L}_s^{-1}\mathbf{r} = \mathbf{e} + \|\mathbf{e}\| O(\delta_s) = \mathbf{e} + O(\delta_s^2)$.
- Which tells me that the correction is

$$\mathbf{x} = \mathbf{x}_d + \mathbf{e}_d = \mathbf{x} + \mathbf{e} + O(\delta_s^2) = \mathbf{x}^* + O(\delta_s^2)$$

So if $\delta_s = 10^{-k}$, you get k figures with every iteration.

Analysis via Back-of-the-Envelope:II

What's δ_s?

- $\epsilon_s \approx 10^{-8}$, single precision roundoff?
- $\kappa(\mathbf{A})\epsilon_s \approx 1900 \times 10^{-8} \approx 10^{-5}$?

Looks like $\delta_s = \kappa(\mathbf{A})\epsilon_s$.

A different perspective

Is this a stationary iterative method? Is the fixed point map

$$\mathbf{K}(\mathbf{x}) = \mathbf{x} + \mathbf{S}(\mathbf{x}) \equiv \mathbf{x} + I_s^d \mathbf{U}_s^{-1} \mathbf{L}_s^{-1} I_d^s (\mathbf{b} - \mathbf{A}\mathbf{x})$$

So $\mathbf{K}$ is linear if the maps

- I_s^d conversion from single to double
  No problem: puts the same number is a bigger bucket
- I_d^s conversion from double to single
  This rounding, just as linear as floating point addition

Bottom line: it's close enough to linear.

What's the iteration matrix?

The map is

$$\begin{aligned}\mathbf{K}(\mathbf{x}) &= \mathbf{x} + I_s^d \mathbf{U}_s^{-1}\mathbf{L}_s^{-1} I_d^s(\mathbf{b} - \mathbf{A}\mathbf{x}) \\ &= (\mathbf{I} - I_s^d \mathbf{U}_s^{-1}\mathbf{L}_s^{-1} I_d^s \mathbf{A})\mathbf{x} + I_s^d \mathbf{U}_s^{-1}\mathbf{L}_s^{-1} I_d^s \mathbf{b} \equiv \mathbf{M}_s\mathbf{x} + \mathbf{x}_s\end{aligned}$$

So it's preconditioned Richardson iteration and the convergence rate is

$$\rho(\mathbf{M}_s) = \rho\left(\mathbf{I} - I_s^d \mathbf{U}_s^{-1}\mathbf{L}_s^{-1} I_d^s \mathbf{A}\right).$$

How might we estimate that?

Estimating $\|\mathbf{M}_s\|$

How about evaluating $\|\mathbf{M}_s\mathbf{x} - \mathbf{x}\|$ for a random $\mathbf{x}$?
We already did that and got

relative error	relative residual
1.77e-05	1.52e-07
1.61e-11	2.25e-13
2.02e-14	1.02e-16

So things look consistent, and $I_s^d \mathbf{U}_s^{-1} \mathbf{L}_s^{-1} I_d^s$ is a good approximate inverse of $\mathbf{A}$.
Warning! Matlab does not support sparse single precision arrays.

There's more!

Suppose you have two solvers $\mathbf{S}_1$ and $\mathbf{S}_2$ which are good at different things.
Suppose

- $\mathbf{x}_+ = \mathbf{S}_1(\mathbf{x}_c, \mathbf{b})$ is a few iterations of an iterative solver
  - for $\mathbf{Ax} = \mathbf{b}$,
  - with $\mathbf{x}_c$ as the incoming iterate, and
  - $\mathbf{x}_+$ as the new iterate.
- $\mathbf{S}_2(\mathbf{b})$ is a solver (iterative or direct) which returns a "converged" result for $\mathbf{Ae} = \mathbf{r}$.
- Given the transfer maps, these solvers could have different physics, precisions, grids, ...

Two-solver correction

You can let one correct the other via . . .

$\mathbf{x} = \mathbf{x}_0$
while Not happy **do**
  $\mathbf{x}_{1/2} = \mathbf{S}_1(\mathbf{x}, \mathbf{b})$
  $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}_{1/2}$
  $\mathbf{e} = \mathbf{S}_2(\mathbf{r})$
  $\mathbf{x} = \mathbf{x}_{1/2} + \mathbf{e}$
end while

Examples

- Iterative refinement: $\mathbf{S}_1 = \mathbf{I}$; $\mathbf{S}_2 = \mathbf{U}_s^{-1}\mathbf{L}_s^{-1}$.
- MCSA: $\mathbf{S}_1$ is one step of Richardson: $\mathbf{S}_2$ is Monte Carlo solve
- Two-grid: $\mathbf{S}_1$ is a couple steps of Jacobi, Gauss-Seidel, ... $\mathbf{S}_2$ is a solve on a coarser grid.
- Multigrid: Like two-grid, but $\mathbf{S}_2$ is a recursive call to MG.

Two grid for Laplacian in 1-D

Recall our sad story with Jacobi for this problem.

- We had a simple example.
- Jacobi did ok for the first few iterations
- and then went very, very slowly.
- It seemed to kill the high frequency error fast,
- and get stuck on the low frequency stuff.

Let's look again.

Jacobi Example

Let's solve

$$-u'' = 0,\ u(0) = u(1) = 0.$$

with $h = 1/101$ and $N = 100$. The solution is $u = 0$. We will use as an initial iterate

$$u_0 = x(1-x) + \frac{1}{10}\sin(49\pi x)$$

We will take 100 Jacobi iterations.

Matrix Representation

$$\mathbf{Au} = \mathbf{f}$$

where A is tridiagonal and symmetric

$$\mathbf{A}^h = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & \dots & 0, & 0 \\ -1 & 2 & -1 & ,0 & \dots & 0 \\ 0 & -1 & 2 & -1, & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots, & ,0, & -1 & 2 & -1 \\ 0 & \dots, & \dots,, & 0 & -1 & 2 \end{pmatrix}$$

Eigenvalues and Eigenvectors

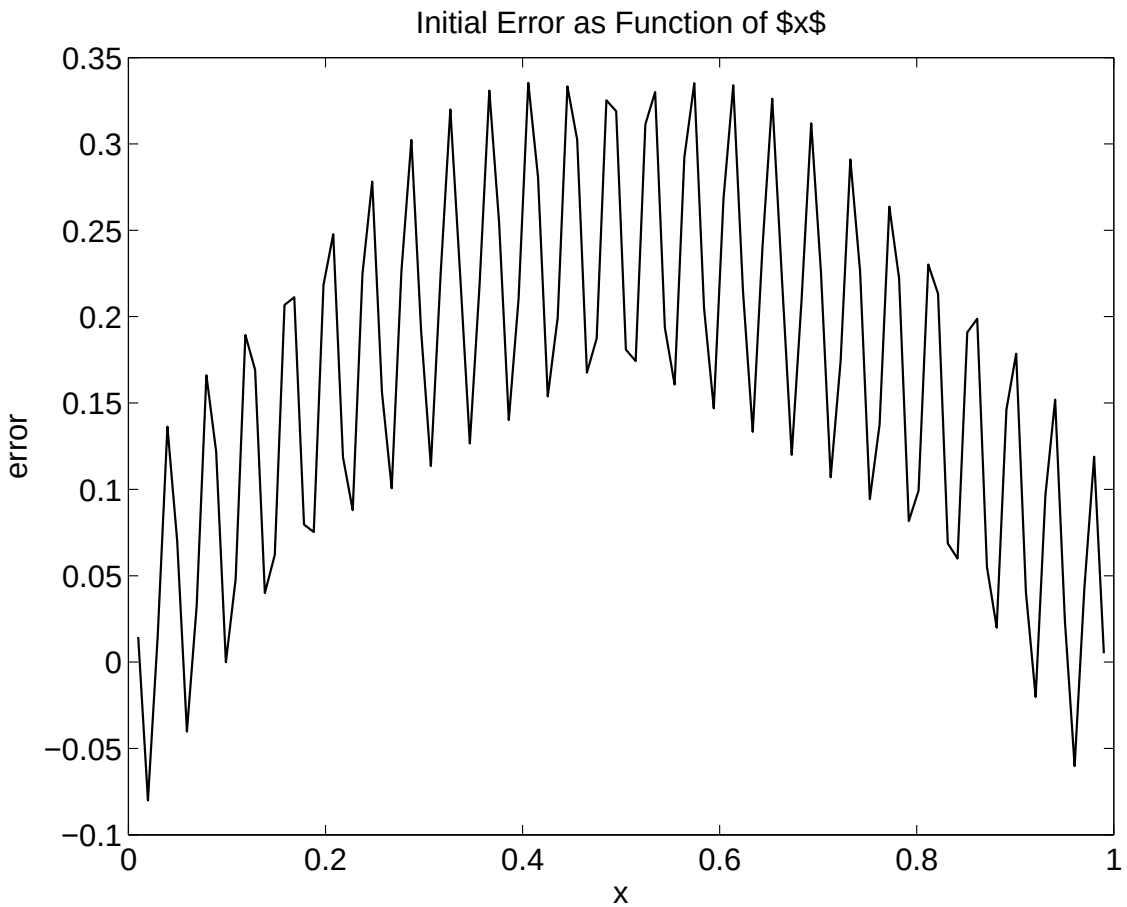
Theorem: A is symmetric positive definite. The eigenvalues are

$$\lambda_n = h^{-2} 2 \left(1 - \cos(\pi n h)\right).$$

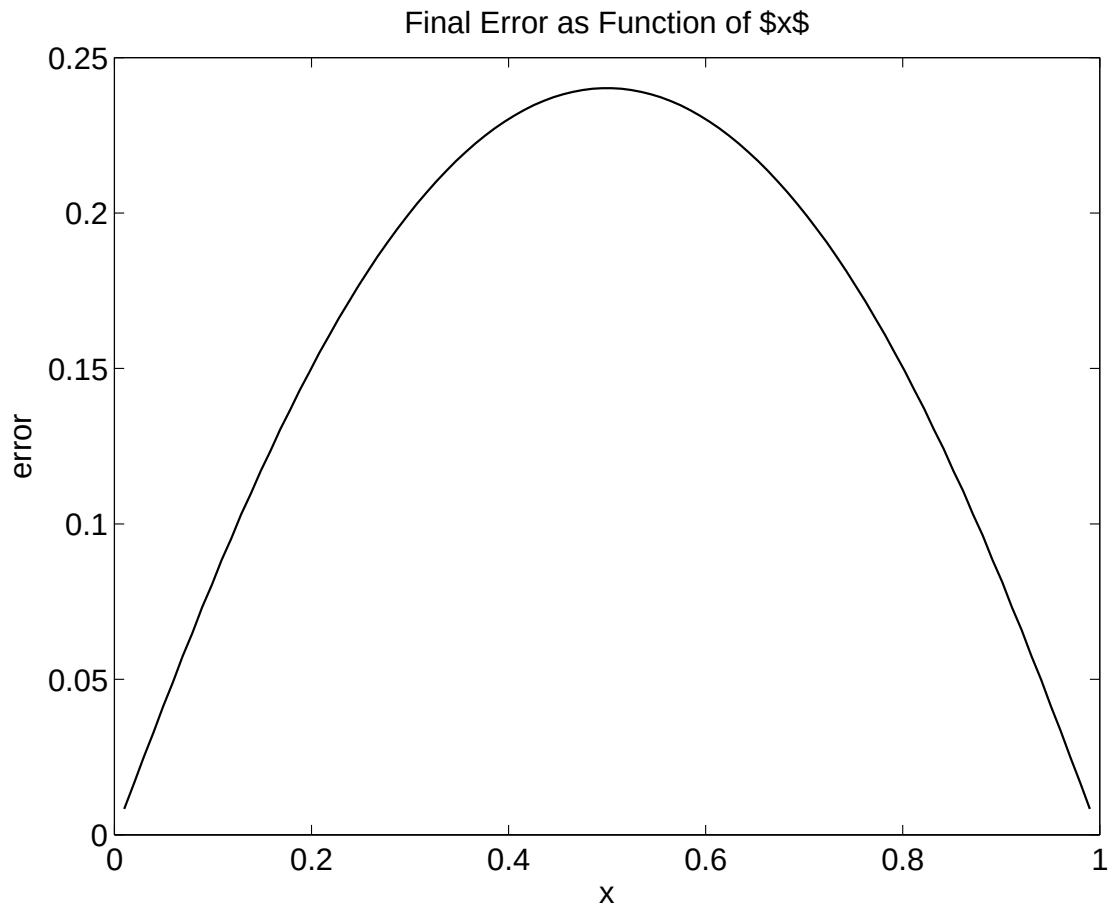
The eigenvectors $\mathbf{u}_n = (\xi_1^n, \ldots, \xi_N^n)^T$ are given by

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

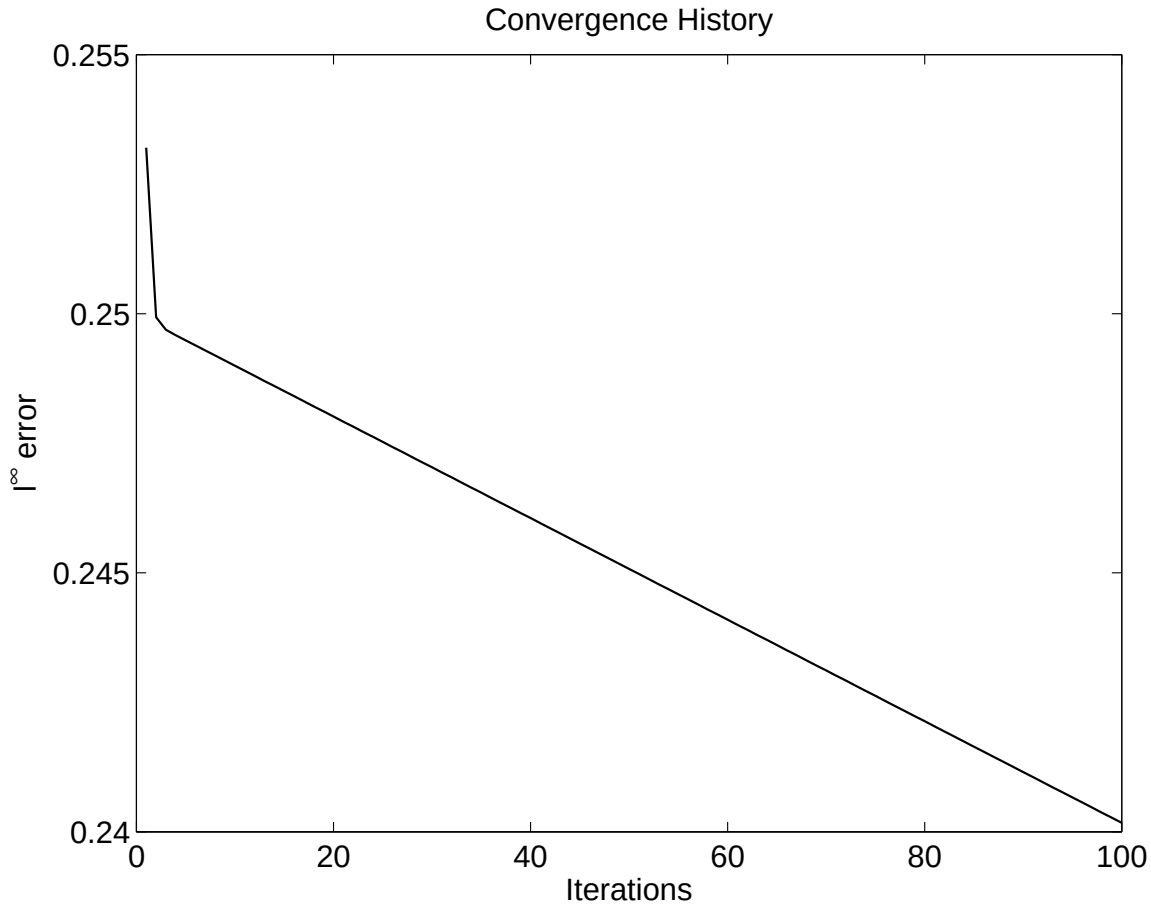
Initial Error as Function of x



Final Error as Function of x



Final Error as Function of x



How's this for a great idea?

- Start at mesh size h.
- Do a few Jacobi's (one?)
  We'll call Jacobi (or whatever) the **smoother**.
- Transfer residual to mesh $2h$.
- Do an exact solve of $\mathbf{D}_2^{2h}\mathbf{e}_{2h} = \mathbf{r}_{2h}$
- Send $\mathbf{e}_{2h}$ to h mesh and correct $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{e}_h$

And the details?

- Exactly how many Jacobi's do I need here?
- Is Jacobi the best tool for the job?
  Gauss-Seidel? SOR? Something else?
- Can I do the obvious thing for intergrid transfer?
- Is this really likely to work?!?

Two-grid algorithm: notation

- Ω^h: vectors corresponding to the h-mesh R^{n^d} where there are n points in each direction and $d = 1, 2, 3$.
- $\mathbf{A}^h$: discrete differential operator on Ω^h.
- I_h^{2h}: fine-to-coarse intergrid transfer.
- I_{2h}^h: coarse-to-fine intergrid transfer.
- $TG(\mathbf{v}^h, \mathbf{f}^h)$: a single two-grid iteration for $\mathbf{A}^h\mathbf{v}^h = \mathbf{f}^h$.
- $\mathbf{S}^\nu(\mathbf{v}^h, \mathbf{f}^h)$: ν iterations of the **smoother**.

Two-Grid Method

$\mathbf{v}^h \leftarrow TG(\mathbf{v}^h, \mathbf{f}^h)$

$\quad \mathbf{v}^h \leftarrow \mathbf{S}^{\nu_1}(\mathbf{v}^h, \mathbf{f}^h)$

$\quad \mathbf{r}^h = \mathbf{f}^h - \mathbf{A}^h \mathbf{v}^h$

$\quad \mathbf{r}^{2h} = I_h^{2h} \mathbf{r}^h$

$\quad$ Solve(?) $\mathbf{A}^{2h} \mathbf{e}^{2h} = \mathbf{r}^{2h}$

$\quad \mathbf{e}^h = I_{2h}^h \mathbf{e}^{2h}$

$\quad \mathbf{v}^h = v^h + e^h$

$\quad \mathbf{v}^h \leftarrow \mathbf{S}^{\nu_2}(\mathbf{v}^h, \mathbf{f}^h)$

Testing . . .

Let's make some questionable decisions.

- One Jacobi presmooth ($\nu_1 = 1$, $\nu_2 = 0$)
- I_{2h}^{h}: piecewise linear interpolation.
- I_{h}^{2h}: restriction by injection.

Jacobi in Matlab

```
function us=jacobi(u,b)
n=length(u); h=1/(n+1); h2=h*h; us=zeros(n
    ,1);
us(2:n-1)=h2*b(2:n-1) + u(3:n) + us(1:n-2);
us(1) = h2*b(1) + u(2); us(n) = h2*b(n) + u(
    n-1);
us=us*.5;
```

Restriction by Injection

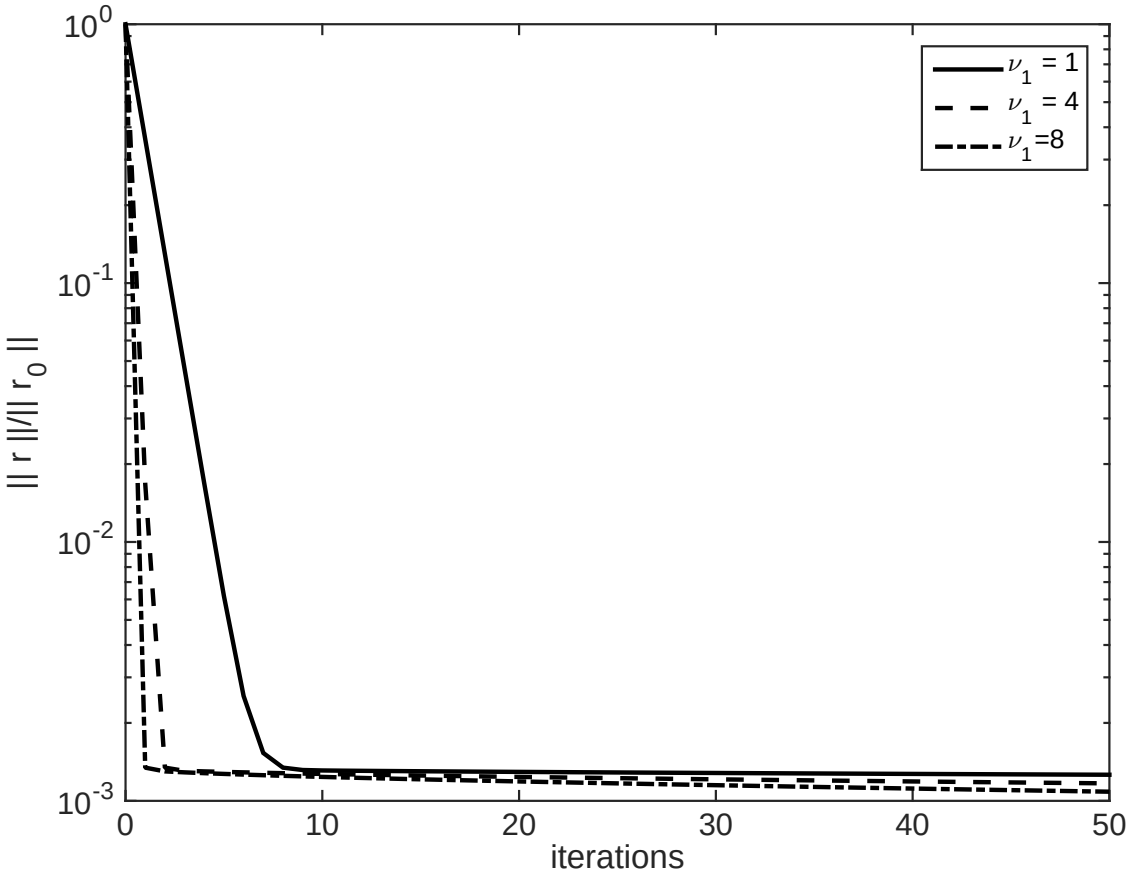
```
function uc=inject(uf)
% INJECT
% fine to coarse intergrid transfer by
   injection
% function uc=inject(uf)
%
nf=length(uf);
uc=uf(2:2:nf-1);
```

Piecewise Linear Interpolation

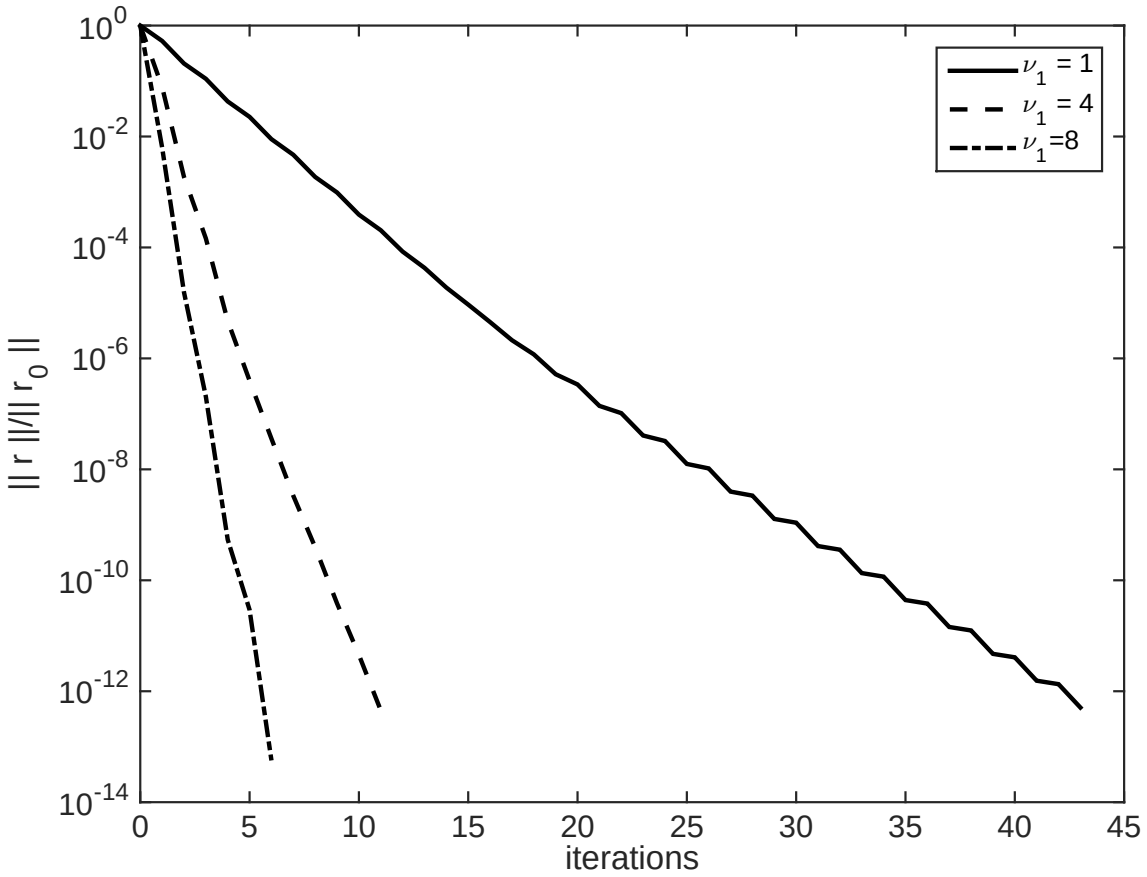
```
function uf = ctof(uc)
% CTOF
% function uf = ctof(uc)
% Coarse to fine intergrid transfer
%
nc=length(uc); nf=2*(nc+1) - 1; uf=zeros(nf,1);
% Use zero boundary conditions
uf(1)=.5*uc(1); uf(nf)=.5*uc(nc);
% Vectorize the rest.
uf(3:2:nf-1)=.5*(uc(1:nc-1)+uc(2:nc));
uf(2:2:nf)=uc;
```

Restriction by Injection, two-grid, Jacobi

That would be great, but it seems to get stuck.



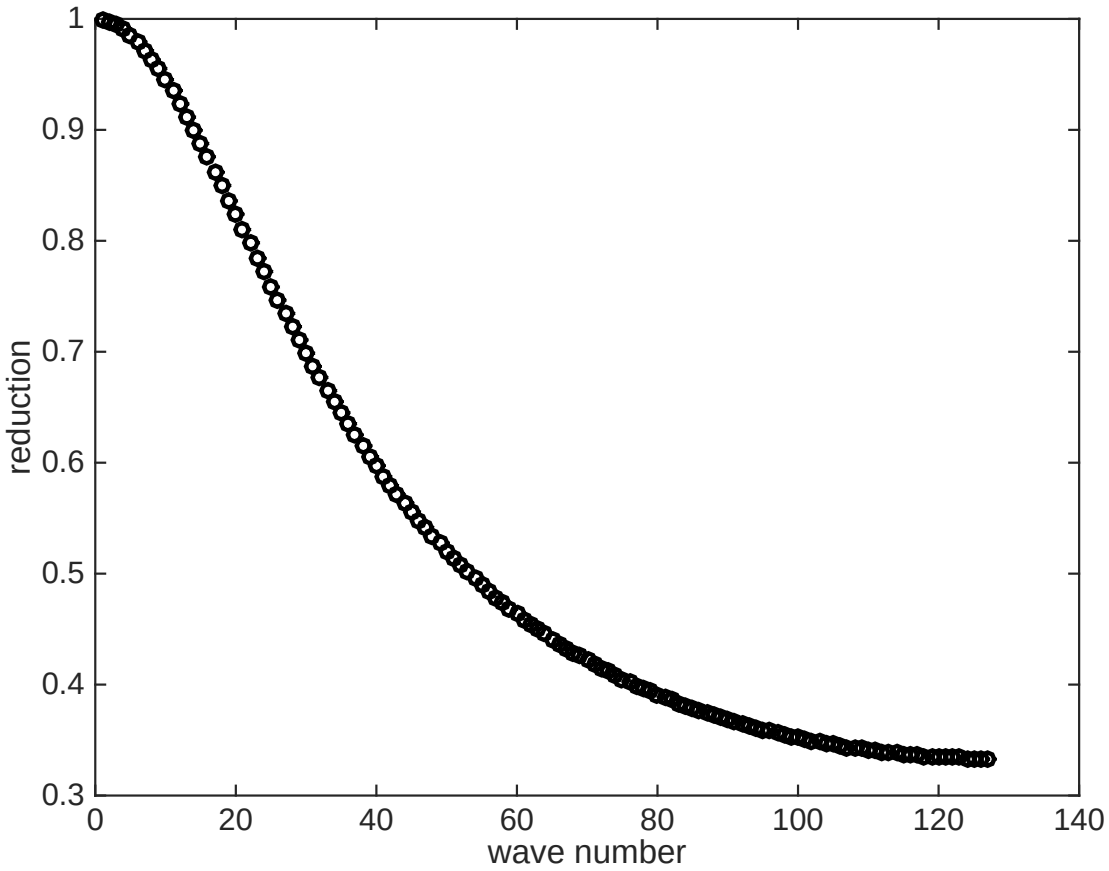
Gauss-Seidel is better. Why?



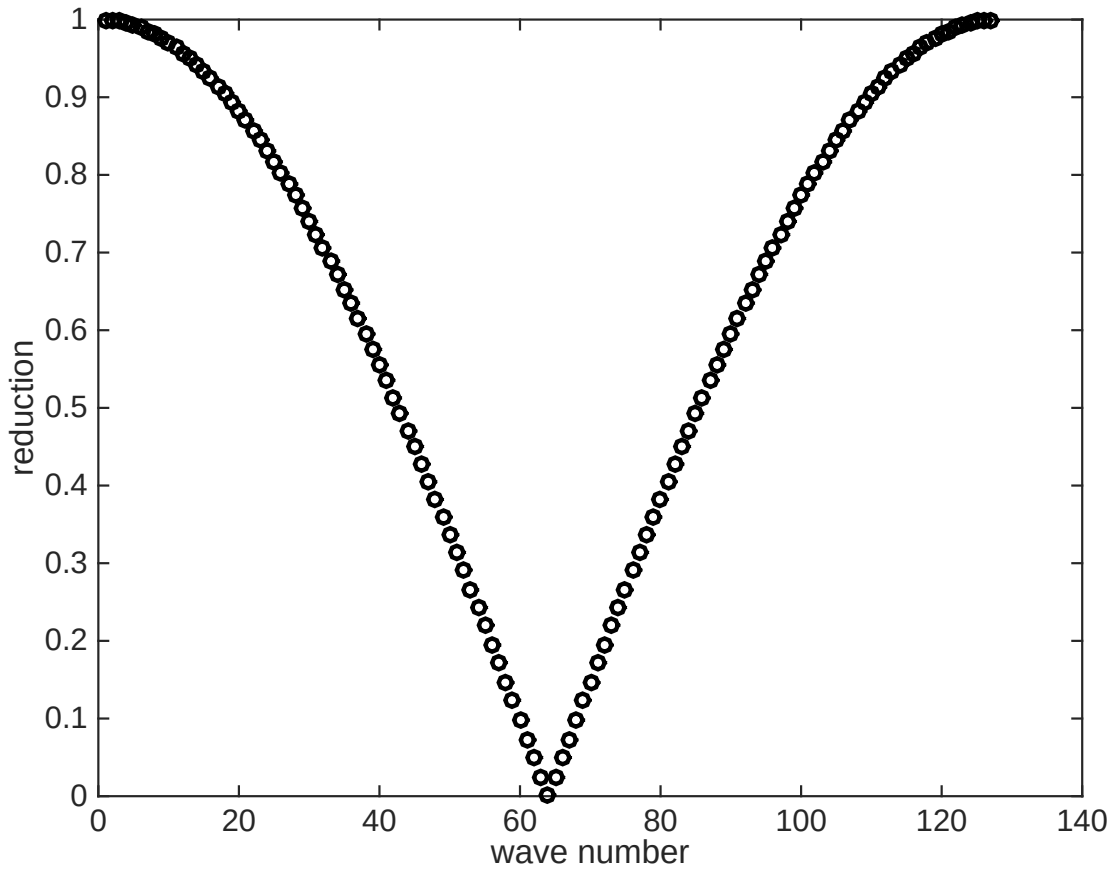
What happened?

- We had this idea that Jacobi did great on high-frequency and poorly on low, so
- I assumed that Gauss-Seidel did the same.
- I was half right.
- It's time to do an experiment.
  - Take one Jacobi/Gauss-Seidel iteration for
  - $\mathbf{Au} = 0$ with $\mathbf{u}_0 = \sin(i\pi x)$ for $1 \leq i \leq n$
  - and see what the reduction rate is.

Results for Gauss-Seidel



Results for Jacobi



I'd like to save Jacobi because

it parallelizes and vectorizes well.
One fix: damped Jacobi

$$\mathbf{u}^{new} = (1-\omega)\mathbf{u}^{old} + \omega\mathbf{u}^{jacobi}$$

For our problem it's

$$u_i^{new} = (1-\omega)u_i^{old} + \omega(h^2 b_i + u_{i-1}^{old} + u_{i+1}^{old})/2.$$

So what's ω?

- $\omega = 1$ is Jacobi.
- $\omega = 0$ is do nothing.
- The winner is $\omega = 2/3$ because ...

Analysis of Damped Jacobi

Damped Jacobi is

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

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

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

Eigen-analysis

We showed that the eigenvalues of $\mathbf{M}_{JAC}$ were

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

with the same eigenvectors as $\mathbf{A}^h$. Similarly

$$\begin{aligned} \mu_n &= (1-\omega) + \omega(1 - (h^2/2)\lambda_n) = 1 - (h^2/2)\omega\lambda_n \\ &= 1 - \omega(1 - \cos(\pi n h)) \end{aligned}$$

Optimal ω

Now suppose $n \geq N/2$ (high frequency), then

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

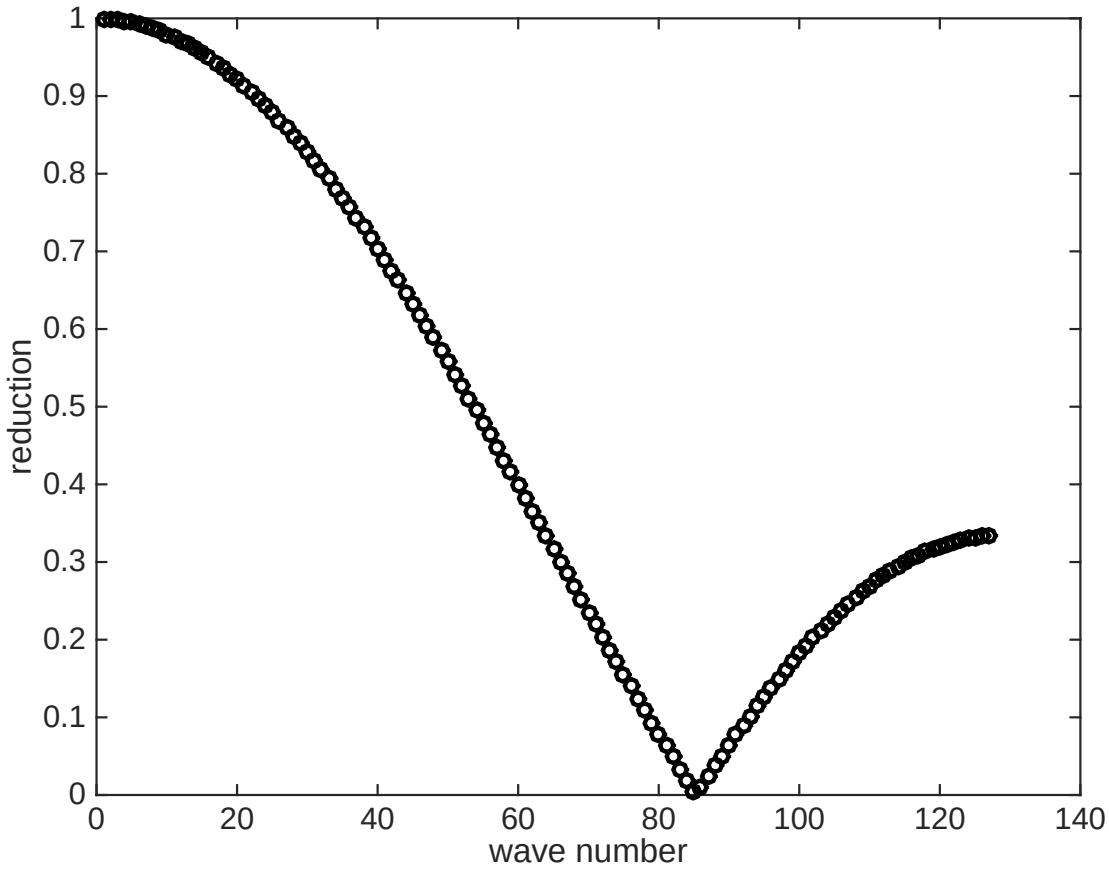
because $\cos(\pi n h) \leq \cos(\pi/2) = 0$, so

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

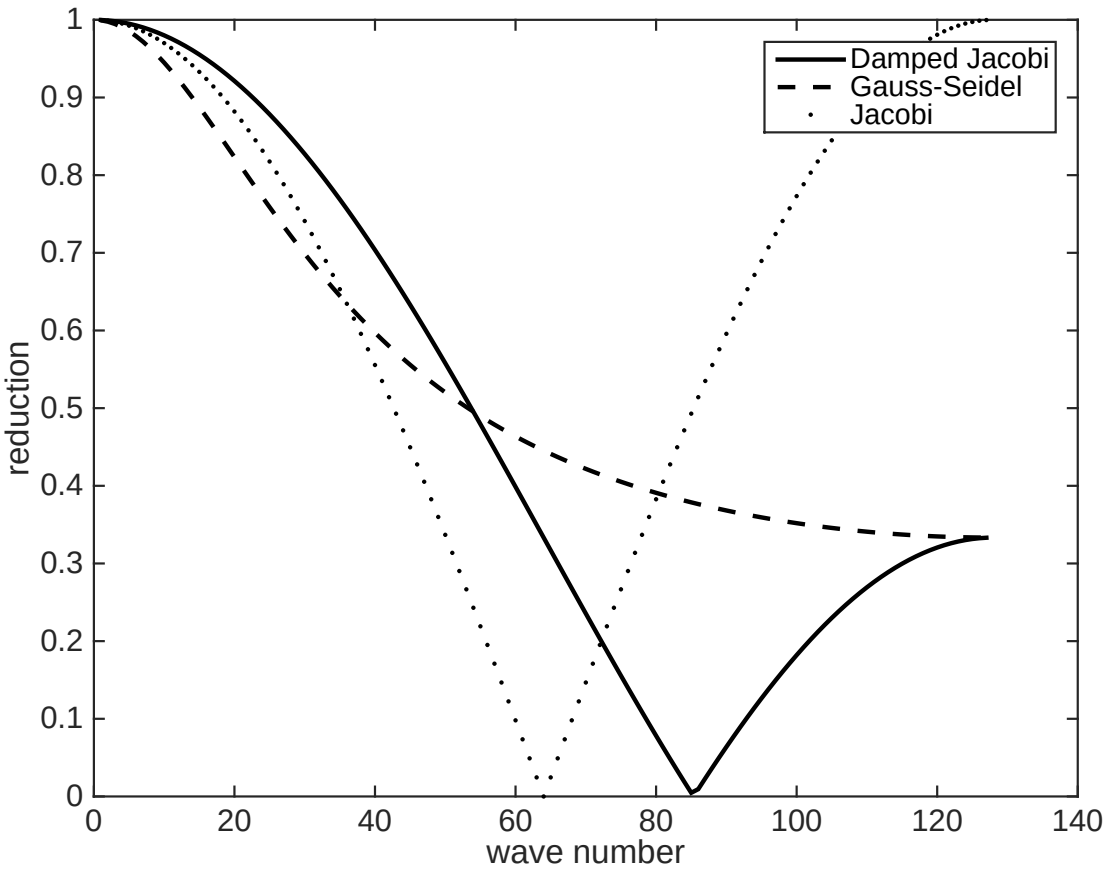
Minimize $|\mu_n|$ to see that the optimal value of ω is $2/3$. So

$$|\mu_n| \leq 1/3 \text{ for } N/2 \leq n \leq N - 1$$

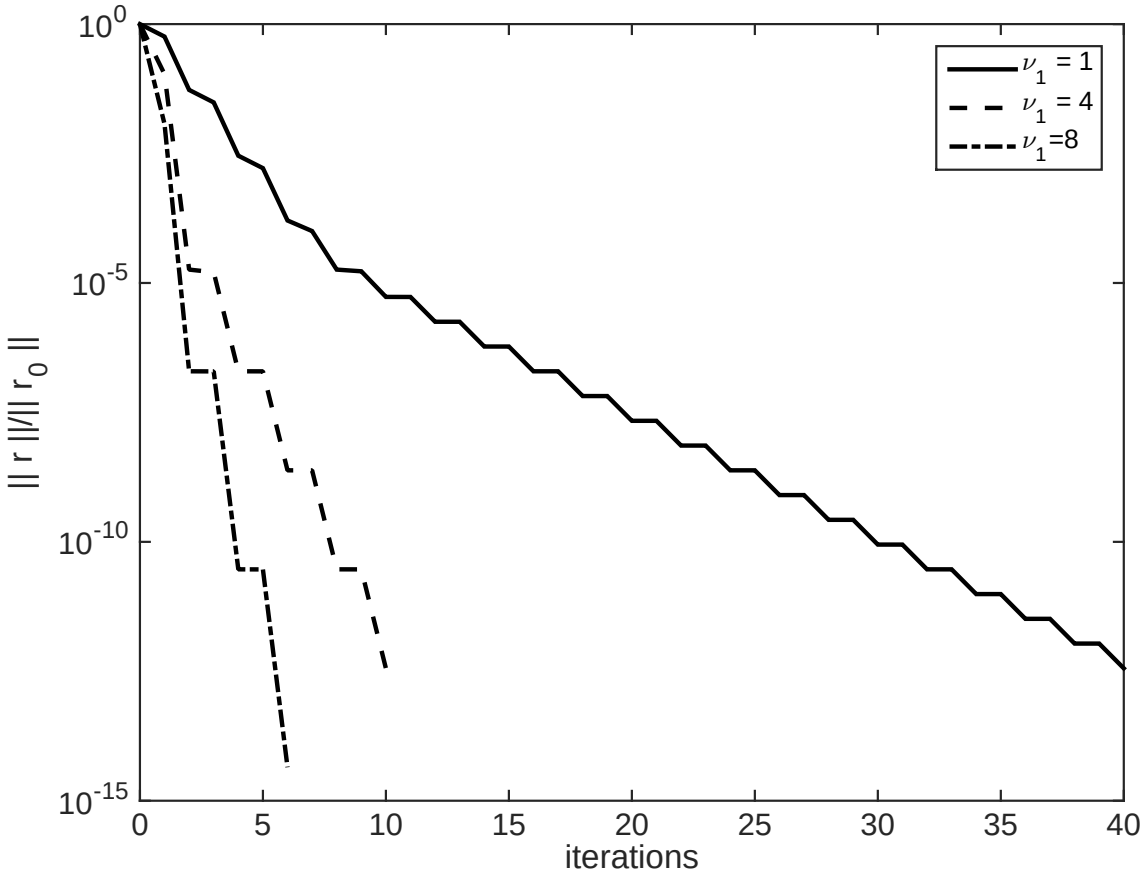
Results for Damped Jacobi



Comparison


Damped Jacobi
Gauss-Seidel
Jacobi
reduction
wave number
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
20
40
60
80
100
120
140

Damped Jacobi's the winner. Let's solve something.



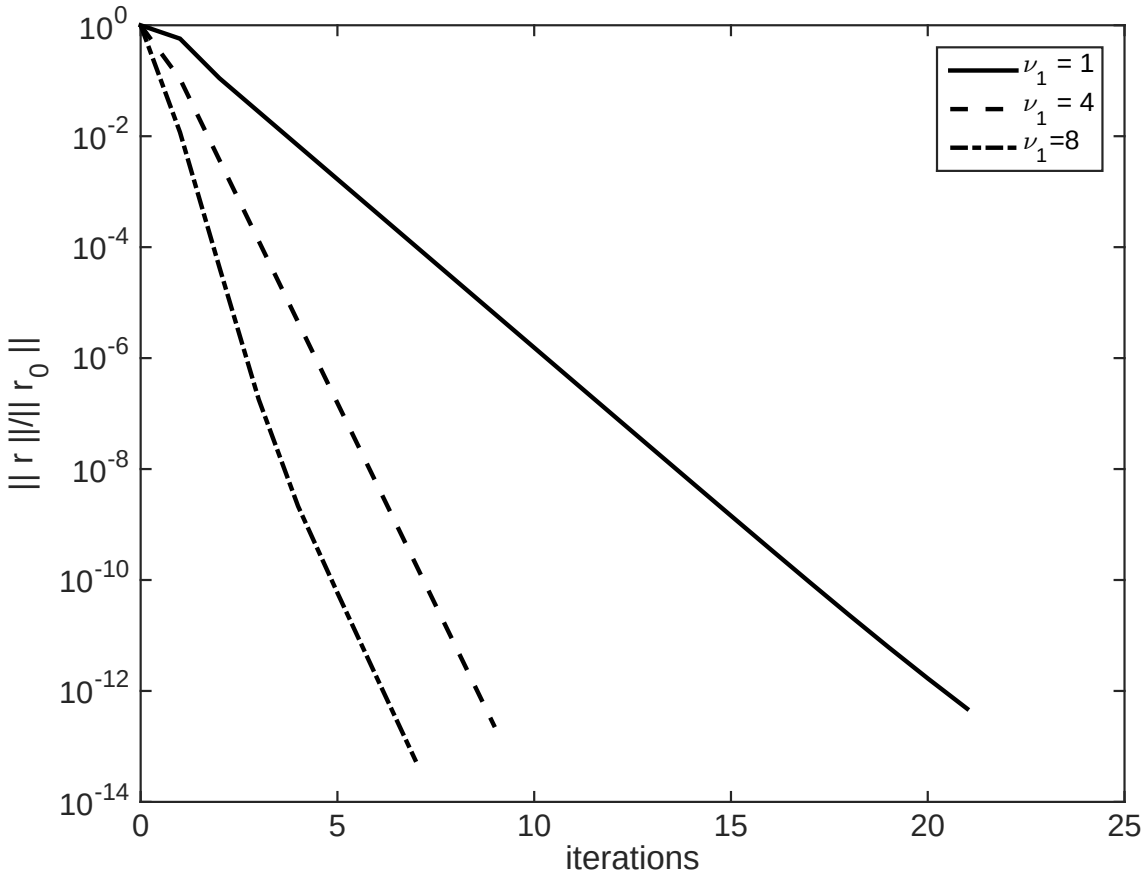
Can we do better?

One problem is that restriction by injection ignores half of the fine grid points.
A better idea is restriction by **full weighting**

$$u_i^{2h} = (u_{2i-1}^h + 2u_{2i}^h + u_{2i+1}^h)/4$$

```
function vcoarse=ftoc(vfine)
% FTOC  fine to coarse intergrid transfer by full
    weighting
nf=length(vfine);
nc=.5*(nf+1)-1;
vtmp=zeros(nf+2,1);
vcoarse=.5*vfine(2:2:nf-1);
vcoarse=vcoarse+.25*(vfine(1:2:nf-2)+vfine(3:2:nf));
```

Restriction by full weighting, two-grid, Damped Jacobi



Something more to like. Matrix representations

$$I_{2h}^{h} = \frac{1}{2}\begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 \\ 2 & 0 & \dots & 0 & 0 & 0 \\ 1 & 1 & \dots & 0 & 0 & 0 \\ 0 & 2 & \dots & 0 & 0 & 0 \\ 0 & 1 & 1 & \dots & 0 & 0 \\ 0 & 0 & 2 & \dots & 0 & 0 \\ 0 & 0 & 1 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 1 \\ 0 & 0 & 0 & \dots & 0 & 2 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}, I_{h}^{2h} = \frac{1}{2}(I_{2h}^{h})^{T}(\text{full weighting})$$

Two-Grid is a stationary iterative method

Go through the loops and

$$\mathbf{M}_{TG} = \mathbf{M}_S^{\nu_2}(\mathbf{I} - I_{2h}^h(\mathbf{A}^{2h})^{-1} I_h^{2h}\mathbf{A}^h)\mathbf{M}_S^{\nu_1}$$

where $\mathbf{M}_S$ is the iteration matrix for the smoother.
One standard configuration is

- Damped Jacobi smoother
- $\nu_1 = \nu_2 = 1$

Estimate the spectral radius

- Look at ratio of relative residuals.
- The ratios should converge to the spectral radius
- unless something strange happens.
- Do this for various values of h to see if the spectral radius depends on h or not.

Feel-good Prediction

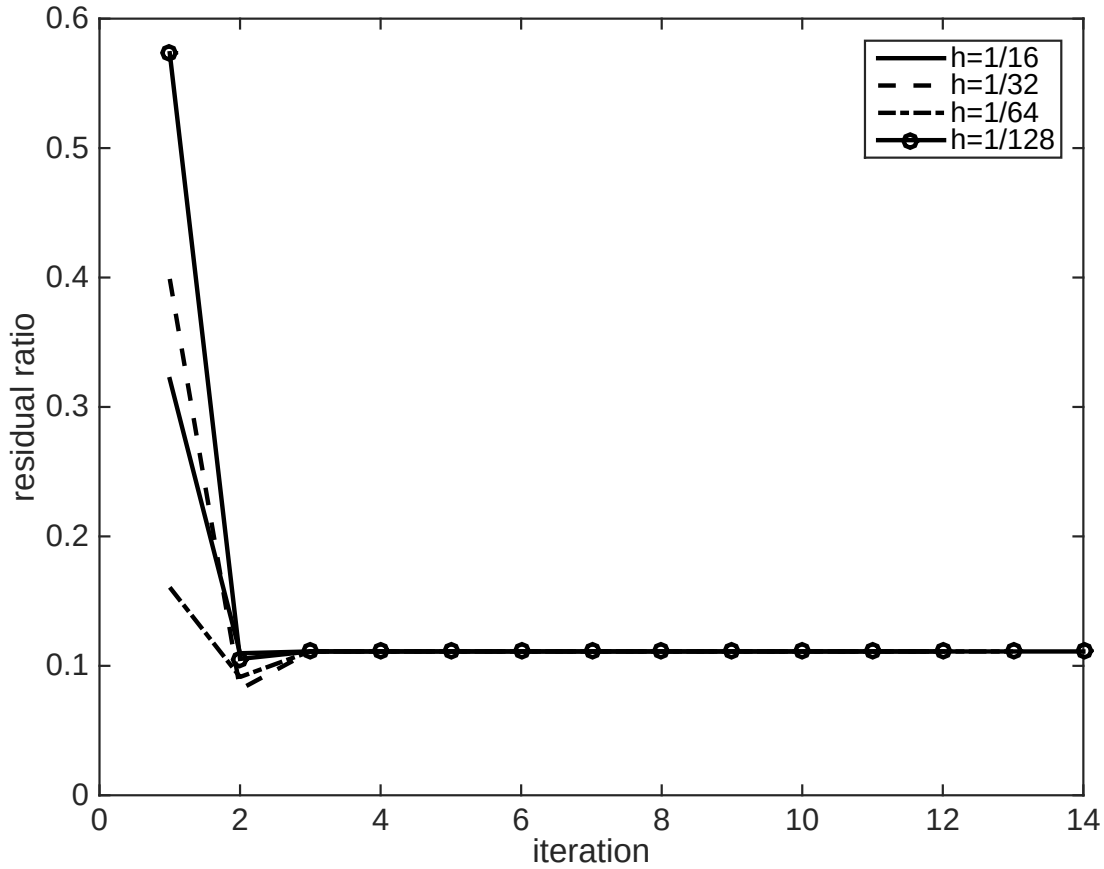
- Each application of damped Jacobi reduces high-frequency error by $1/3$
- So, if the coarse grid correction eliminates the low-frequency errors completely . . .

then

$$\rho(\mathbf{M}_{TG}) = 1/9.$$

Really?

We have a winner! Ratio = 1/9



Where are we?

- Good:
  - TG is a stationary iterative method.
  - Solver work moved to coarse grid.
  - Convergence rate independent of h.
- Bad:
  - Coarse mesh solver is still a matrix factorization.

Fix: replace the coarse mesh solver with another two-grid iteration and get . . .