MA 580; Numerical Analysis I

C. T. Kelley
NC State University
tim_kelley@ncsu.edu
Version of November 30, 2016

NCSU, Fall 2016
Part IXb: Multigrid

Multigrid V-cycle

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

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

$v^h \leftarrow V^h(v^h, f^h)$

if $h = H$ **then**

Solve $\mathbf{A}^h\mathbf{v}^h = \mathbf{f}^h$ exactly.

else

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

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

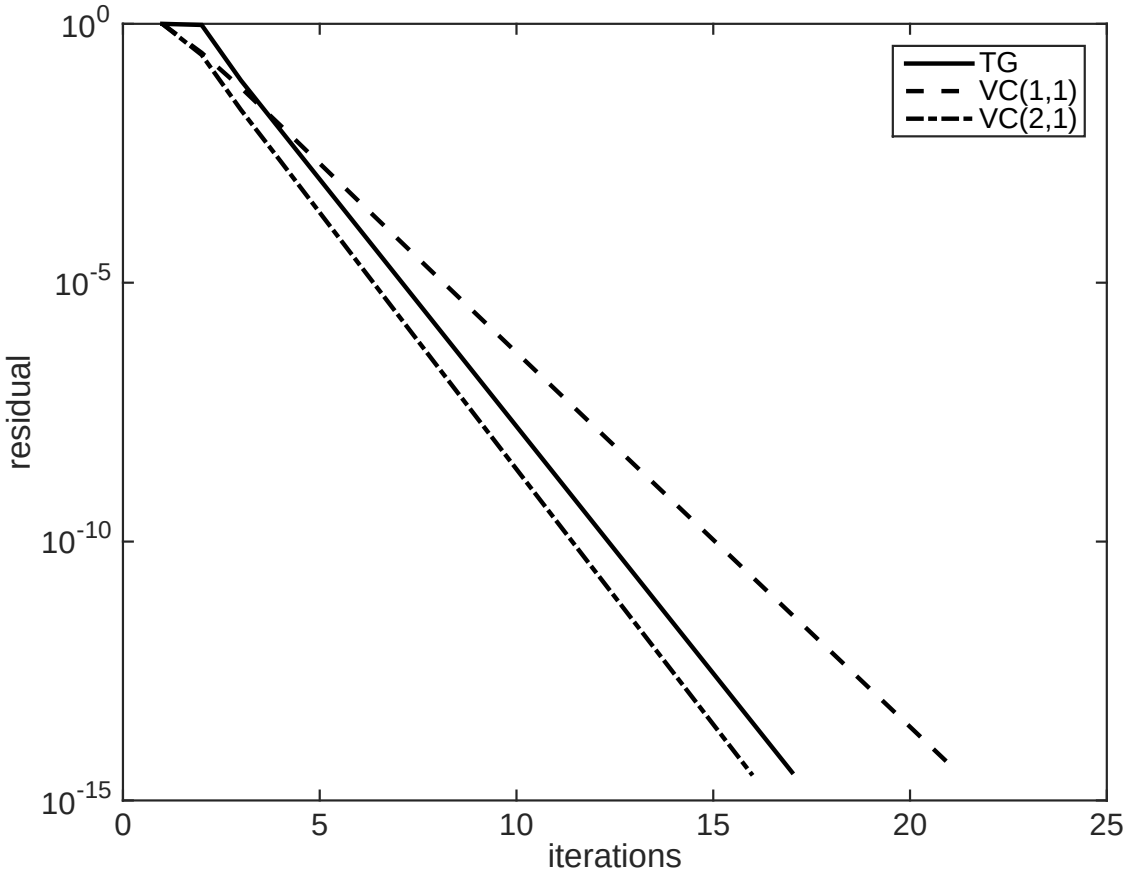
$\mathbf{e}^{2h} = V^{2h}(0, I_h^{2h}\mathbf{r}^h)$

$\mathbf{v}^h = \mathbf{v}^h + I_{2h}^h\mathbf{e}^{2h}$

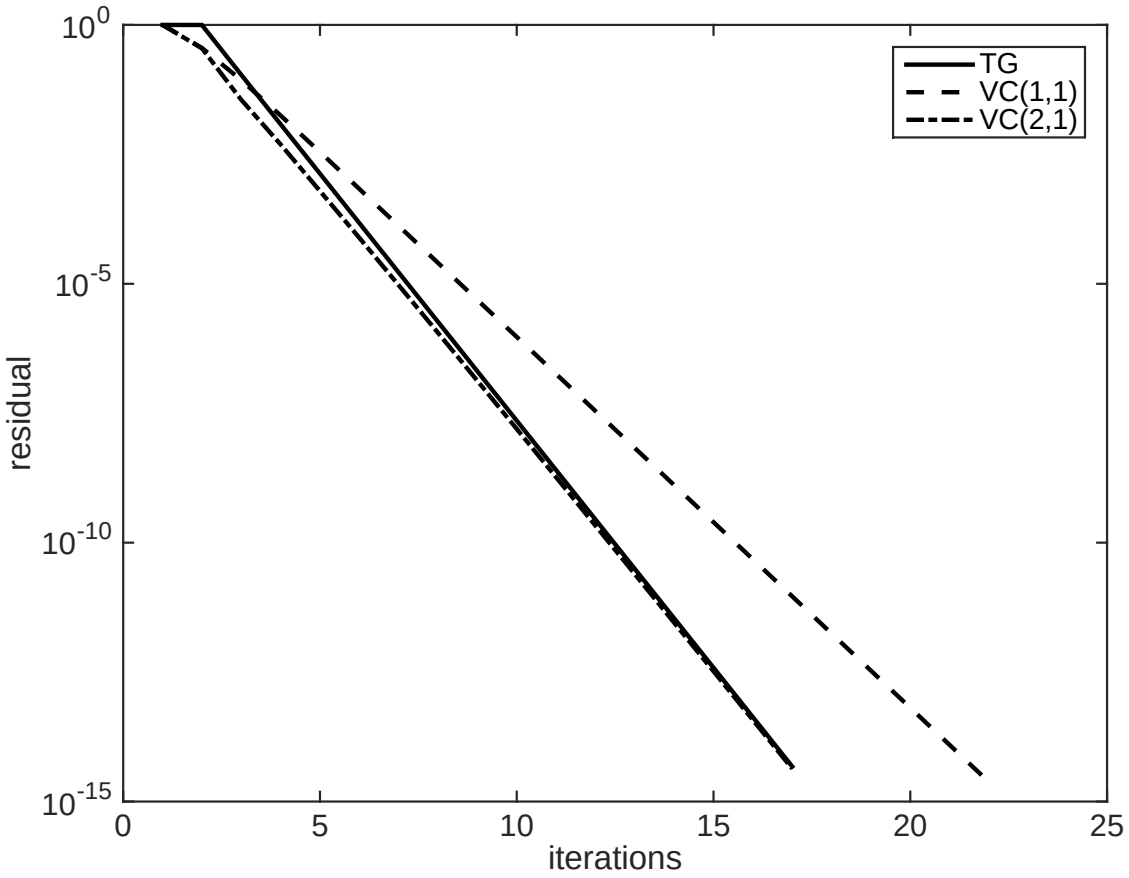
$\mathbf{v}^h \leftarrow \mathbf{S}^{\nu_2}(\mathbf{v}^h, \mathbf{f}^h)$

end if

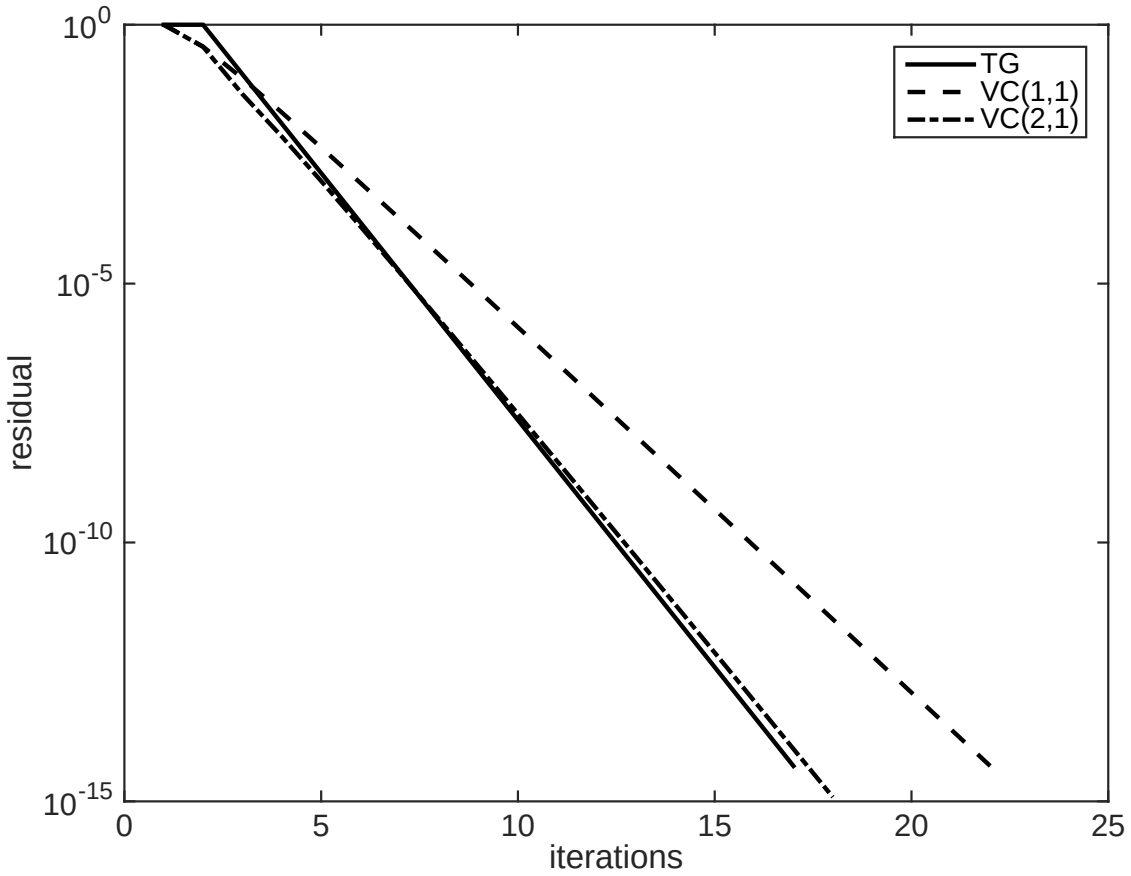
How'd I do? $h = 1/512$



How'd I do? $h = 1/2048$



How'd I do? $h = 1/4096$



Observations

- $\nu_1 = \nu_2 = 1$ is ok, convergence rate = .19
- $\nu_1 = 2, \nu_2 = 1$ is better, convergence rate = .115
- TG is faster, rate = .111, but more expensive.
- The rates seem to be independent of h.
  Theory says they are if H is sufficiently small.
- There are some opportunities for poor and inefficient programming.

Complexity: size of problem

Rules:

- We are solving $-\nabla^2 u = f$ with
  - Dirichlet boundary conditions, which means
  - Values of u specified on boundary.
- We are in $d = 1, 2, 3$ space dimensions
- with $N = n^d$ interior grid points
- $h = 1/(n+1) = O(N^{1/d})$
- $2d + 1$ nonzeros per row in $\mathbf{A}^h$, so
  - Cost of residual and cost of smoother are $(2d+1)n^d + O(n^{d-1}) = O(N)$.

Complexity: Cost of solve, iterations

Objective: solve to truncation error

$$\|\mathbf{e}_k^h\|/\|\mathbf{f}\| = O(h^2).$$

How large does k need to be?
Assume: $\|\mathbf{M}_{VC}\| = \rho$
Then you're there if $\rho^k \approx h^2$ or

$$k \approx 2\log(h)/\log(\rho) = (2/d)\log(N)/\log(1/\rho) = O(\log(N)).$$

Complexity: Cost of solve, V-cycle

- If $h = 2^{-P}$ and $H = 2^{-p}$, then we have $P - p + 1$ levels in the recursion.
- We visit each level twice.
- At level l, $h = 2^{-l}$ and the problem size is

$$N_l = (2^l - 1)^d = N\frac{2^l - 1}{2^P - 1} \approx N2^{l-P}.$$

- Work on each level is
  - $\nu_1 + \nu_2$ and
  - a residual computation.

Keeping score

At each level the cost of the smooths and residual is

$$(\nu_1 + \nu_2 + 1)(2d + 1)N_l = C_V N 2^{l-P}$$

where

$$C_V = (\nu_1 + \nu_2 + 1)(2d + 1)\frac{1 - 2^{-l}}{1 - 2^{-P}}.$$

If we neglect the cost of the "solve" at the base of the recursion, then ...

Bottomline: V-cycle iteration

Since we visit each level twice, the total cost of the V-cycle is

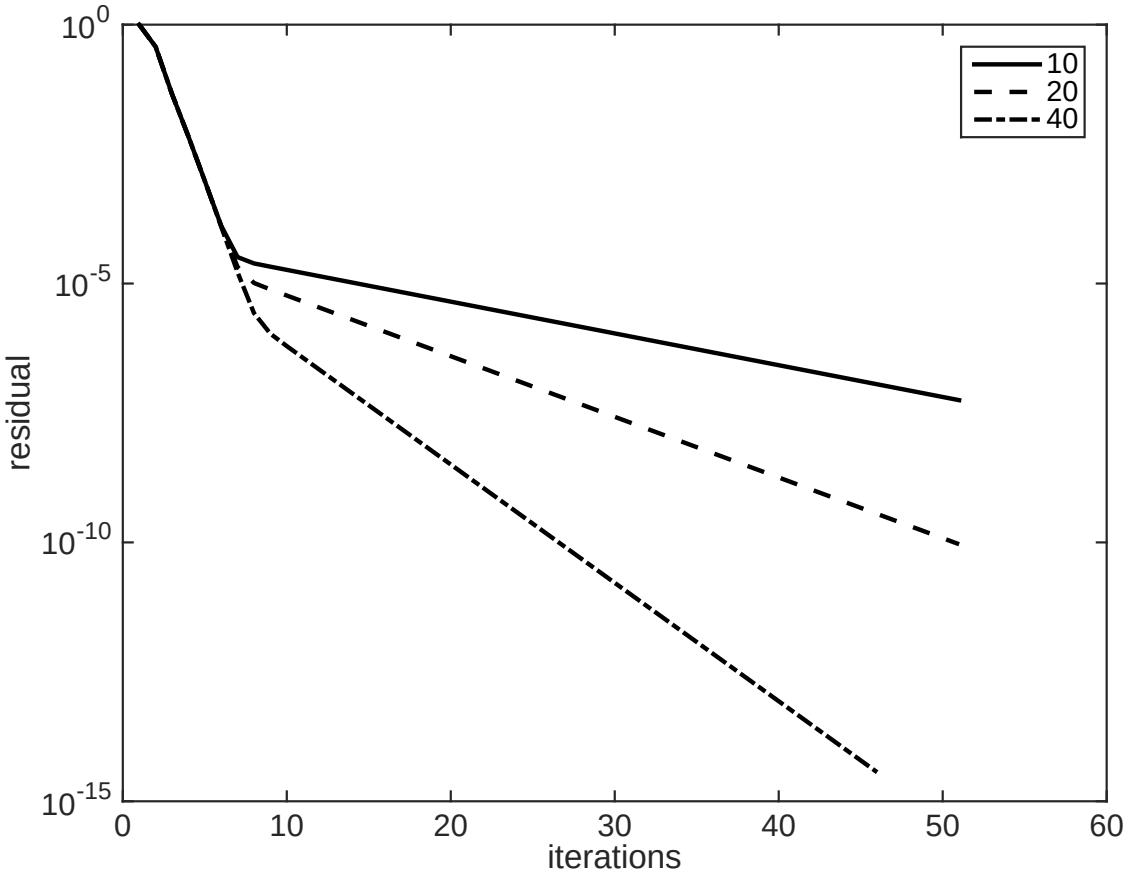
$$C_V N \sum_{l=p}^{P} 2^{l-P} \leq 2 C_V N = O(N)$$

and we need $O(\log(N))$ iterations to solve to truncation error. So the cost of the solve is

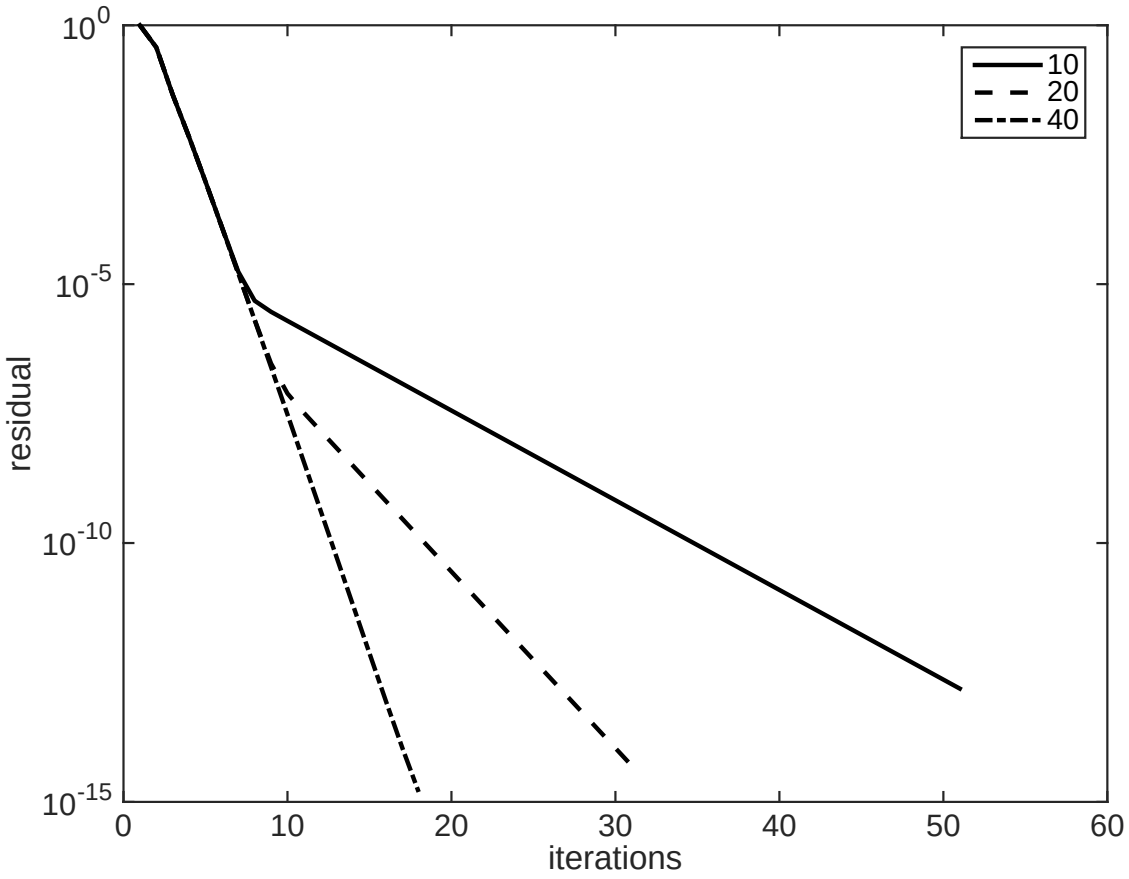
$$O(N \log(N))$$

which is fast!

Cheating on the coarse grid solve? Jacobi?



Cheating on the coarse grid solve? Gauss-Seidel?



What does a "fast solver" have to do

For a problem of size N,

- get an acceptable solution in
- $O(N)$ or $O(N \log(N))$ work.

So, are we finished? Is the V-cycle the end?

Nested Iteration or Full Multigrid (FMG): $h = 2^{-(P-p)}H$

Solve $\mathbf{A}^H\mathbf{v}^H = \mathbf{f}^H$ on Ω^H; $h = H$.
for $il = 1 : p$ **do**
  $h = h/2$
  $\mathbf{v}^h = I_{2h}^h \mathbf{v}^{2h}$
  $\mathbf{v}^h \leftarrow V^h(\mathbf{v}^h, \mathbf{f}^h)$
end for

What is this in English?

- Start on the coarse mesh and do a highly accurate solve.
- Move up one mesh at a time with
  - the result from the previous mesh as the initial iterate
  - for **a single** V-cycle,
  - after which, it's time for the next finer mesh.

Any reason why this should work?

Assume $\|\mathbf{M}_{VC}\| = \rho < 1/4$, then

- The discretization is second order accurate.
  - The incoming error is second order on the $2h$ mesh.
  - Reduce error by $1/4$ and it's time for the next mesh.
- So at each level, I'm accurate to truncation error after the V-cycle.

Complexity

- Neglect the coarse mesh solve.
- One V-cycle at level l with cost of $C_V N_l$
- Sum this mess to get a cost of

$$C_V \sum_{l=p}^{P} N 2^{P-l} \leq 2 C_V N = O(N)!$$

Simple example

$$-u'' = f, u(0) = a, u(1) = b$$

Where $a = 1, b = e\cos(1)$,

$$u^*(x) = e^x \cos(x) \text{ and } f(x) = 2e^x \sin(x)$$

How does this change the discretization?

Change in discretization

$(D2u)_1 = (2u_1 - u_2 - a)/h^2$ and $(D2u)_N = (2u_N - u_{N-1} - b)/h^2$.

So the matrix does not change, but f_1^h and f_N^h get the boundary conditions.

$$f_i^h = f(x_i) \; 2 \le i \le N-1,$$

$$f_1^h = f(x_1) + a/h^2, \text{ and } f_N^h = f(x_N) + b/h^2.$$

Change in I_{2h}^h

The coarse-to-fine intergrid transfer for f needs to know about boundary conditions.

```
% Use boundary conditions in the
    interpolation.
uf(1)=.5*(a + uc(1)); uf(nf)=.5*(b + uc(nc))
    ;
% Vectorize the rest.
uf(3:2:nf-1)=.5*(uc(1:nc-1)+uc(2:nc));
uf(2:2:nf)=uc;
```

Evaluating the results

- For this problem $h = 1/4096$ and $H = 1/32$.
- We should see errors decrease by $1/4$ at each level.
  - We can extract the errors from FMG for this artificial problem.
  - We can also compare $I_{2h}^{h} v^{2h}$ to v^{h} at level h.
- So we'll look at both.

Errors and Differences

$\|v^h - v^*\|_\infty$	ratio	$\|I_{2h}^h v^{2h} - v^h\|_\infty$	ratio
4.11e-05	0.00	5.27e-04	0.00
1.50e-05	0.36	1.55e-04	0.30
4.77e-06	0.32	4.27e-05	0.27
1.35e-06	0.28	1.15e-05	0.27
3.80e-07	0.28	3.04e-06	0.26
1.00e-07	0.26	7.95e-07	0.26
2.64e-08	0.26	2.04e-07	0.26

I can't get clean timings. The solve is too fast.

How hard can this be?

Solve $\mathbf{A}^H \mathbf{v}^H = \mathbf{f}^H$ on Ω^H; $h = H$.
for $il = 1 : p$ **do**
  $h = h/2$
  $\mathbf{v}^h = I_{2h}^h \mathbf{v}^{2h}$
  $\mathbf{v}^h \leftarrow V^h(\mathbf{v}^h, \mathbf{f}^h)$
end for

Reality

- If I do what I've been doing, I'm in trouble.
  - Building $\mathbf{A}^h$, $\mathbf{f}^h$, $\mathbf{e}^h$ every pass through V^h costs way too much.
  - I need to pay attention to storage if I'm to do 2-D or 3-D.
  - So, I need to think about data structures (coming up).
- A plain-vanilla tridiagonal solver is faster for this problem.
  - So this is not very interesting in 1-D.

Data Structures: Solutions and Right Side

- At a minimum,
  - you need to allocate storage $\mathbf{v}^h$ and $\mathbf{f}^h$ for all levels.
  - You'll use this storage for both $\mathbf{e}^h$ and $\mathbf{r}^h$.
- You may also want to store $\mathbf{A}^h$ for all levels.
  - That gets painful in 2-D and 3-D, so perhaps ...
  - you'll pass around a function for the mat-vec.

Preconditioning via Stationary Iterative Methods

Recall that if you have a stationary iterative method for

$$\mathbf{Ax} = \mathbf{b}$$

that looks like

$$\mathbf{x} \leftarrow \mathbf{K}(\mathbf{x}, \mathbf{f}) \equiv \mathbf{Mx} + \mathbf{c} \equiv \mathbf{Mx} + \mathbf{Lb}$$

You get a preconditioner $\mathbf{P}$ when you express the iteration as

$$\mathbf{x} \leftarrow (\mathbf{I} - \mathbf{PA})\mathbf{x} + \mathbf{Pb}$$

which means $\mathbf{P} = \mathbf{L}$.

Recall Jacobi precondtioning

So, the preconditioner-vector product can be realized as

$$\mathbf{Pv} = \mathbf{K}(0, \mathbf{v}).$$

Let's verify with Jacobi

$$\mathbf{K}_{jac}(\mathbf{x}, \mathbf{f}) = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x} + \mathbf{D}^{-1}\mathbf{f}.$$

So

$$\mathbf{P}_{jac}\mathbf{v} = \mathbf{K}_{jac}(0, \mathbf{v}) = \mathbf{D}^{-1}\mathbf{v}.$$

So if $\mathbf{K}$ can't be cleanly expressed in matrices I can manipulate, I can still build a preconditioner.

V-cycle preconditioning

$$\mathbf{Pv} \equiv V^h(0, \mathbf{v})$$

- Implementation, easy
  - you'll need to package the V-cycle for the Krylov solver,
  - which is a different task that packaging for FMG.
  - Then the PC-vector product is just a call to $V^h(0, \mathbf{v})$.
- For analysis, you'll have to compute the preconditioner.

Example: Two-grid iteration ...

Getting it to work.

Remember this?

$$-u''(x) + \int_0^1 k(x,y)u(y)\,dy = f(x), \;\; u(0) = u(1) = 0.$$

where the integral operator is the one we've been using

$$k(x,y) = \frac{\sin(x-y)}{1+x+y}$$

and $f(x) = e^{10x}\sin(10x)$.

What's different?

Very little,

- Swap V-cycle for $D2^{-1}$
- Different things in data structure
- Files in `Integral_Differential_MG_PC` on Moodle page

Setting things up

Call the V-cycle from the pvec function.

```
function pvec=mgpc(v,my_stuff)
% MGPC apply V-cycle for Laplacian as a
   preconditioner.
%
n=length(v);
pvec=PC_VC(zeros(n,1),v,my_stuff,0,0);
```

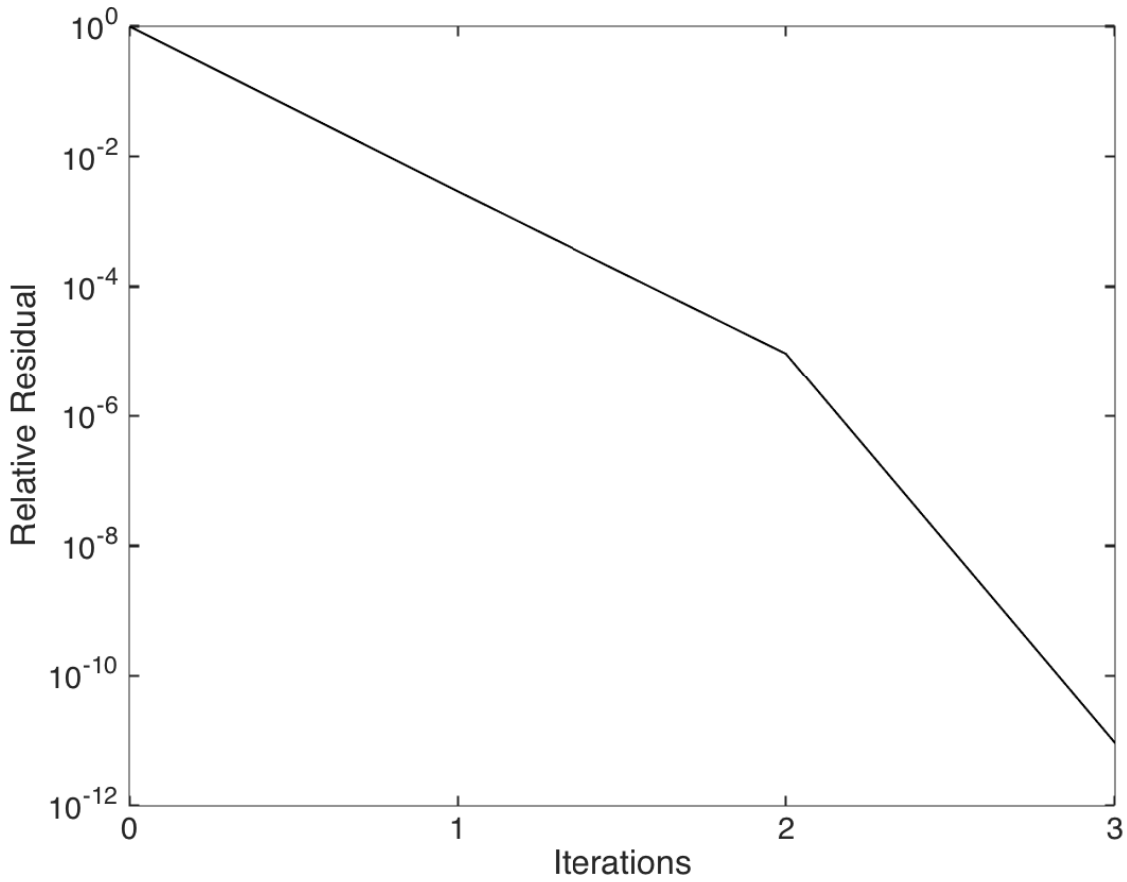
Building the structure

```
function my_stuff = build_pc_data(nf,nc,nu1,nu2)
%
lmax=1 + log2((nf+1)/(nc+1));
n = ((nf+1)/2^(lmax-1))-1; h=1/(n+1);
e=ones(n,1);
AC=spdiags([-e 2*e -e], -1:1, n, n);
AC=AC/(h*h);
my_stuff=struct('nf',nf,'nc',nc,'lmax',lmax,...
        'nu1',nu1,'nu2',nu2,'AC',AC);
```

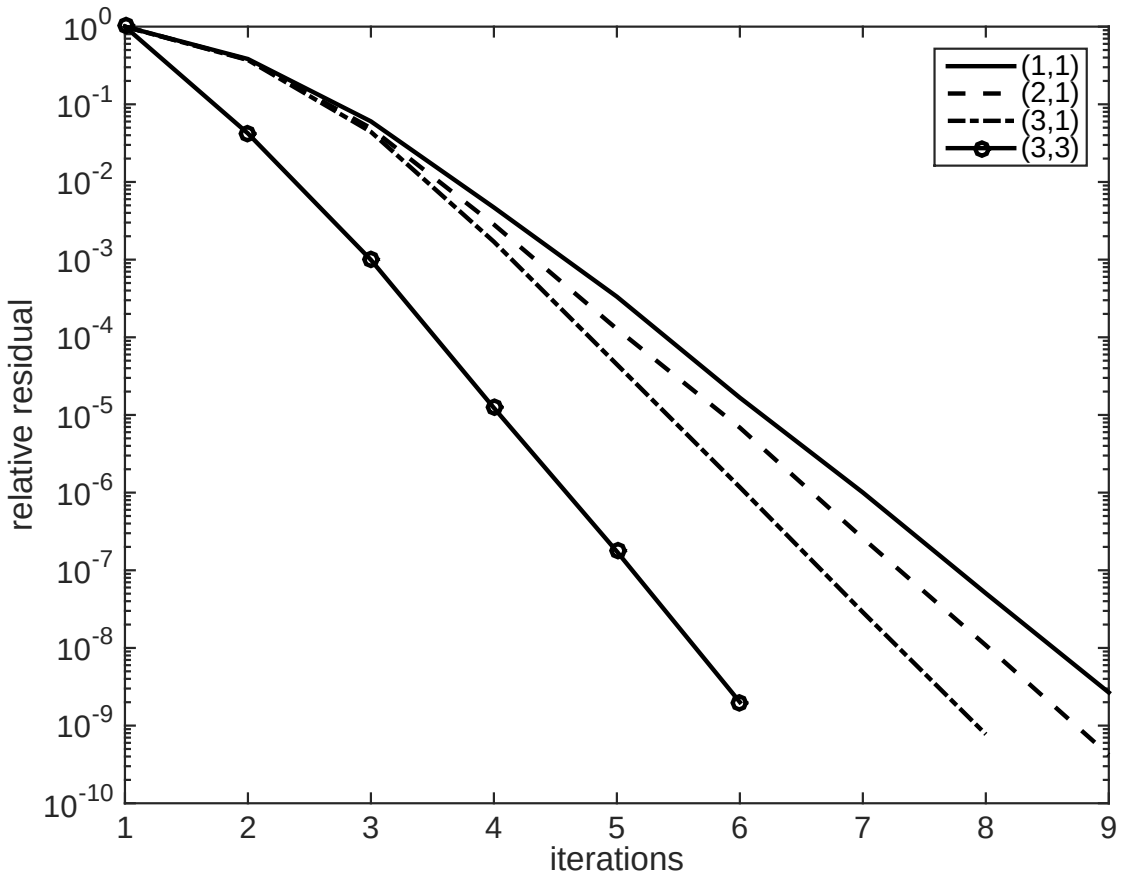
And that's it.

```
%
% set up the options for kl
%
options=kl_optset('ltol',1.d-8,'matvec_data',A,...
     'ptv',@mgpc,'p_data',p_data,'p_side','right');
```

Residual History from Laplacian solve PC



Residual History from V-cycle PC



Two Space Dimensions

$$-\nabla^2 u(x,y) = f(x,y) \text{ for } 0 < x, y < 1$$

subject to

$$\begin{aligned} u(x,0) &= b_{down}(x); \\ u(x,1) &= b_{up}(x); \\ u(0,y) &= b_{left}(y); \\ u(1,y) &= b_{right}(y); \end{aligned}$$

Discretization

$$(\mathbf{A}^h\mathbf{v})_{i,j} = h^{-2}(4v_{i,j} - v_{i-1,j} - v_{i+1,j} - v_{i,j-1} - v_{i,j+1}) = f^h_{i,j}$$

understanding that

$$v_{0,j} = v_{i,0} = v_{(n+1),j} = v_{i,(n+1)} = 0.$$

Boundary conditions in right side

$$f^h_{i,j} = f(x^h_i, x^h_j) \text{ for } 2 \leq i, \leq n-1$$

and

$$\begin{aligned}
f^h_{1,j} &= f(x^h_1, x^h_j) + b_{left}(x^h_j)/h^2, \\
f^h_{n,j} &= f(x^h_n, x^h_j) + b_{right}(x^h_j)/h^2, \\
f^h_{i,1} &= f(x^h_i, x^h_1) + b_{down}(x^h_i)/h^2, \\
f^h_{i,n} &= f(x^h_i, x^h_n) + b_{up}(x^h_i)/h^2.
\end{aligned}$$

Damped Jacobi

$$v_{ij}^{Jacobi} = h^2 f_{ij} + v_{i-1,j} + v_{i+1,j} + v_{i,j-1} + v_{i,j+1}$$

$$\mathbf{v}^{DJ} = (1-\omega)\mathbf{v} + \omega \mathbf{v}^{Jacobi}.$$

"Optimal": $\omega = 4/5$, $\rho \approx .6$.

Fine-to-Coarse full weighting

$$
\begin{aligned}
(I_h^{2h} v^h)_{ij} &= (v^h_{2i-1,2j} + v^h_{2i+1,2j} \\
&\quad + v^h_{2i,2j-1} + v^h_{2i,2j+1} + 4v^h_{2i,2j})/8
\end{aligned}
$$

Coarse-to-fine linear interpolation

For the fine mesh points on or in between coarse mesh points.

$$
\begin{aligned}
(I_{2h}^{h}\mathbf{v}^{2h})_{2i,2j} &= \mathbf{v}_{i,j}^{2h} \\
(I_{2h}^{h}\mathbf{v}^{2h})_{2i+1,2j} &= (\mathbf{v}_{i,j}^{2h} + \mathbf{v}_{i+1,j}^{2h})/2 \\
(I_{2h}^{h}\mathbf{v}^{2h})_{2i,2j+1} &= (\mathbf{v}_{i,j}^{2h} + \mathbf{v}_{i,j+1}^{2h})/2 \\
(I_{2h}^{h}\mathbf{v}^{2h})_{2i+1,2j+1} &= (v_{i,j}^{2h} + v_{i+1,j}^{2h} + v_{i,j+1}^{2h} + v_{i+1,j+1}^{2h})/4
\end{aligned}
$$

Minimizing errors with a hack.

```
function uf=ctof(uc,bf);
% CTOF
% Interpolation in 2D
%
[nc,nc]=size(uc);
nf=2*(nc+1)-1;
%
% Here's an array large enough to hold the boundary data. This
% lets me vectorize everything.
%
ucpad=zeros(nc+2,nc+2); ucpad(2:nc+1,2:nc+1)=uc;
uf=zeros(nf,nf); erange=2:2:nf-1; orange=1:2:nf; lrange=1:2:nf-1;
uf(erange,erange)=uc;
uf(orange,erange)=.5*(ucpad(1:nc+1,2:nc+1)+ucpad(2:nc+2,2:nc+1));
uf(erange,orange)=.5*(ucpad(2:nc+1,1:nc+1)+ucpad(2:nc+1,2:nc+2));
uf(orange,orange)=.25*(ucpad(1:nc+1,1:nc+1)+ucpad(1:nc+1,2:nc+2)...
+ucpad(2:nc+2,2:nc+2)+ucpad(2:nc+2,1:nc+1));
```

Model Problem

$$-\nabla^2 u(x,y) = f(x,y)$$

with data cooked up so that

$$u^*(x,y) = e^{xy}\cos(x).$$

- Damped Jacobi smoother; $\nu_1 = \nu_2 = 3$
- Full weighting
- Residual evaluation with i, j loops
- Store fine mesh solution as 2D vector
- Direct solve on coarse mesh

Two grid: $nf = 1023$, over 10^6 unknowns

$\|\mathbf{r}\|_\infty$	ratio	$\|\mathbf{e}\|_\infty$	ratio	$\|\delta\mathbf{v}_n\|_\infty$	ratio
1.07e+09		1.16e+03			
4.31e+07	0.04	3.52e+01	0.03	1.16e+03	
1.95e+06	0.05	1.40e+00	0.04	3.38e+01	0.03
8.76e+04	0.04	5.99e-02	0.04	1.34e+00	0.04
3.99e+03	0.05	2.63e-03	0.04	5.73e-02	0.04
1.82e+02	0.05	1.17e-04	0.04	2.51e-03	0.04
8.34e+00	0.05	5.29e-06	0.05	1.12e-04	0.04
3.83e-01	0.05	2.26e-06	0.43	5.04e-06	0.05
1.76e-02	0.05	2.26e-06	1.00	2.29e-07	0.05
8.16e-04	0.05	2.26e-06	1.00	1.05e-08	0.05
3.78e-05	0.05	2.26e-06	1.00	4.85e-10	0.05

V-cycle

$\|\mathbf{r}\|_\infty$	ratio	$\|\mathbf{e}\|_\infty$	ratio	$\|\delta\mathbf{v}_n\|_\infty$	ratio
1.07e+09		1.16e+03			
4.82e+07	0.04	7.92e+01	0.07	1.13e+03	
2.55e+06	0.05	5.93e+00	0.07	7.33e+01	0.06
1.35e+05	0.05	4.40e-01	0.07	5.49e+00	0.07
7.34e+03	0.05	3.25e-02	0.07	4.08e-01	0.07
4.05e+02	0.06	2.42e-03	0.07	3.01e-02	0.07
2.28e+01	0.06	1.80e-04	0.07	2.24e-03	0.07
1.31e+00	0.06	1.36e-05	0.08	1.67e-04	0.07
7.60e-02	0.06	2.29e-06	0.17	1.24e-05	0.07
4.48e-03	0.06	2.26e-06	0.99	9.33e-07	0.08
2.69e-04	0.06	2.26e-06	1.00	7.09e-08	0.08

FMG: $nf = 2047, \nu_1 = \nu_2 = 10$

error	ratio	step	ratio
1.70e-05			
1.19e-05	0.70	1.95e-03	
4.31e-06	0.36	5.22e-04	0.27
1.23e-06	0.29	1.35e-04	0.26
3.23e-07	0.26	3.43e-05	0.25
8.38e-08	0.26	8.65e-06	0.25
2.14e-08	0.26	2.19e-06	0.25
5.45e-09	0.25	5.53e-07	0.25

Comparative Timings

- I tried to compare FMG with `chol(D2)`
  - but `chol(D2)` crashed my desktop for $n = 2047$
- $D2\backslash f$ was faster and did not crash my computer.

FMG is fast

1/h	FMG	ratio	Direct Solve	ratio
32	1.95e-03		8.03e-04	
64	4.83e-03	2.47	6.82e-03	8.49
128	1.30e-02	2.70	2.21e-02	3.24
256	6.93e-02	5.32	1.04e-01	4.73
512	2.03e-01	2.93	5.40e-01	5.17
1024	9.96e-01	4.90	2.52e+00	4.67
2048	4.50e+00	4.51	1.18e+01	4.70

V-cycle preconditioning

We'll do a nonlinear problem

$$-\nabla^2 u + \cos(u_x) = f$$

with f and the boundary conditions fixed so that

$$u^*(x, y) = e^{xy} \cos(x).$$

Jacobian

We have linear operators and composition operators. So if

$$F(u) = -\nabla^2 u + \cos(u_x) - f, \text{ with non-zero bc.}$$

then

$$F'(u)v = -\nabla^2 v - \sin(u_x)v_x, \text{ with zero bc .}$$

Therefore it's possible to do a Jacobian-vector product analytically.

Differentiating BCs

- The type of BC are part of the operator.
  - They can be linear or nonlinear.
- The values of the BC are like constant terms in functions.
- Their derivative is zero.

Example: Linear problem

$$Lu = -u'', \; u(0) = a, u(1) = b.$$

Use the formula

$$L'v = \left. \frac{dL(u + \epsilon v)}{d\epsilon} \right|_{\epsilon=0}$$

to see that

$$L'v = -v'', \; v(0) = v(1) = 0.$$

1D example

$$F(u) = -u'' + e^{-u} - f = 0,\ u(0) = a, u(1) = b;$$

The discrete form with N unknowns is

$$\mathbf{F}(\mathbf{u}) = \mathbf{D}_2\mathbf{u} + \exp(\mathbf{u}) - \mathbf{f}$$

where

$$f_i = f(x_i)\ 2 \le i \le N-1$$

$$f_1 = f(x_1) + a/h^2, f_N = f(x_N) + b/h^2.$$

Nonlinear BC

$$F(u) = -u'' + \cos(u) + f,\ \phi(u(0)) = 0, \psi(u(1)) = 0.$$

$$F'(u)v = -v'' - \sin(u)v,\ \phi'(u(0))v(0) = 0, \psi'(u(1))v(1) = 0.$$

Is this right? Does the discrete form have the correct number of unknowns?

DE holds in the interior, BC on the boundary, so it's right.

So what are the equations

The unknowns are

$$\mathbf{u} = \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_N \\ u_{N+1} \end{pmatrix},$$

$$\mathbf{F}(\mathbf{u}) = \begin{pmatrix} \phi(u_0) \\ \mathbf{D}_2\mathbf{u}(1:N) \\ \phi(u_{N+1}) \end{pmatrix} - \mathbf{f} = 0,$$

and $\mathbf{f}$ ...

Equations: II

$$\mathbf{f} = \begin{pmatrix} 0 \\ f(x_1) + u_0/h^2 \\ u_2 \\ \vdots \\ u_N + u_{N+1}/h^2 \\ 0 \end{pmatrix}$$

Nonlinear 2D problem

$$-\nabla^2 u + \cos(u_x) = f\,.$$

where f and the boundary conditions are cooked up so taht

$$u^*(x,y) = e^{xy}\cos(x)$$

Discrete problem

$$\mathbf{F}(\mathbf{u}) \equiv \mathbf{D}_2\mathbf{u} + \cos(\mathbf{D}_x\mathbf{u}) - \mathbf{f} = 0$$

Where

$$(\mathbf{D}_x\mathbf{u})_{i,j} = \frac{u_{i+1,j} - u_{i-1,j}}{2h}.$$

Evaluate $u(1,j)$ and $u(n,j)$ using the boundary conditions.
WARNING: BC differ if you're evaluating $\mathbf{F}$ or $\mathbf{F}'$!

Setting this up.

- Solvers speak to one-dimensional vectors, so . . .
  - You evaluate the residual and apply the V-cycle on 2D vectors
  - and use `reshape` to go to/from the solvers.
    `v1 = reshape(v2,n*n,1); v2=reshape(v1,n,n);`
- Send the boundary conditions to
  - **f** and
  - the discrete first derivative.
- Regular mesh with $N = nf^2$ points; $nf = 2047$;

The residual is nothing we haven't seen before

```
function nlf = nl_residual(u1,problem_data)
% Nonlinear residual for the 2D V-cycle
    nonlinear problem
n2=length(u1); n=sqrt(n2);
u=reshape(u1,n,n);
fe=problem_data.fe;
bc=problem_data.bc;
ux=dxmult(u,bc);
nlf2=lapmult(u)+cos(ux)-fe;
nlf=reshape(nlf2,n2,1);
```

The x-partial knows about the boundary conditions

```
function ux=dxmult(u,bc);
[n,n]=size(u); h=1/(n+1); hi=.5/h;
bleft=bc(2:n+1,1);
bright=bc(2:n+1,2);
ux(1,:)=(u(2,:)-bleft')*hi;
ux(n,:)=(bright'-u(n-1,:))*hi;
ux(2:n-1,:)=(u(3:n,:)-u(1:n-2,:))*hi;
```

Preconditioner-vector product: Direct solve

```
function pxv=pmult(v,problem_data)
D2=problem_data.D2;
pxv=D2\v;
```

Preconditioner-vector product: V-cycle

```
function pxv=pvc(v2,problem_data)
n2=length(v2); n=sqrt(n2);
v=reshape(v2,n,n);
zz=zeros(n,n);
pv2=VC_2D(zz,v,problem_data);
pxv=reshape(pv2,n2,1);
```

KNL options

```
knlopts=knl_optset('atol',1.d-12,...
'rtol',1.d-12,'etamax',-.1,'p_side','right',...
'orthog','cgs','ptv',ptv,'p_data',problem_data);
```

Call KNL

```
[usol1,histknl]=...
    knl(u0,@nl_residual,knlopts,problem_data);
```

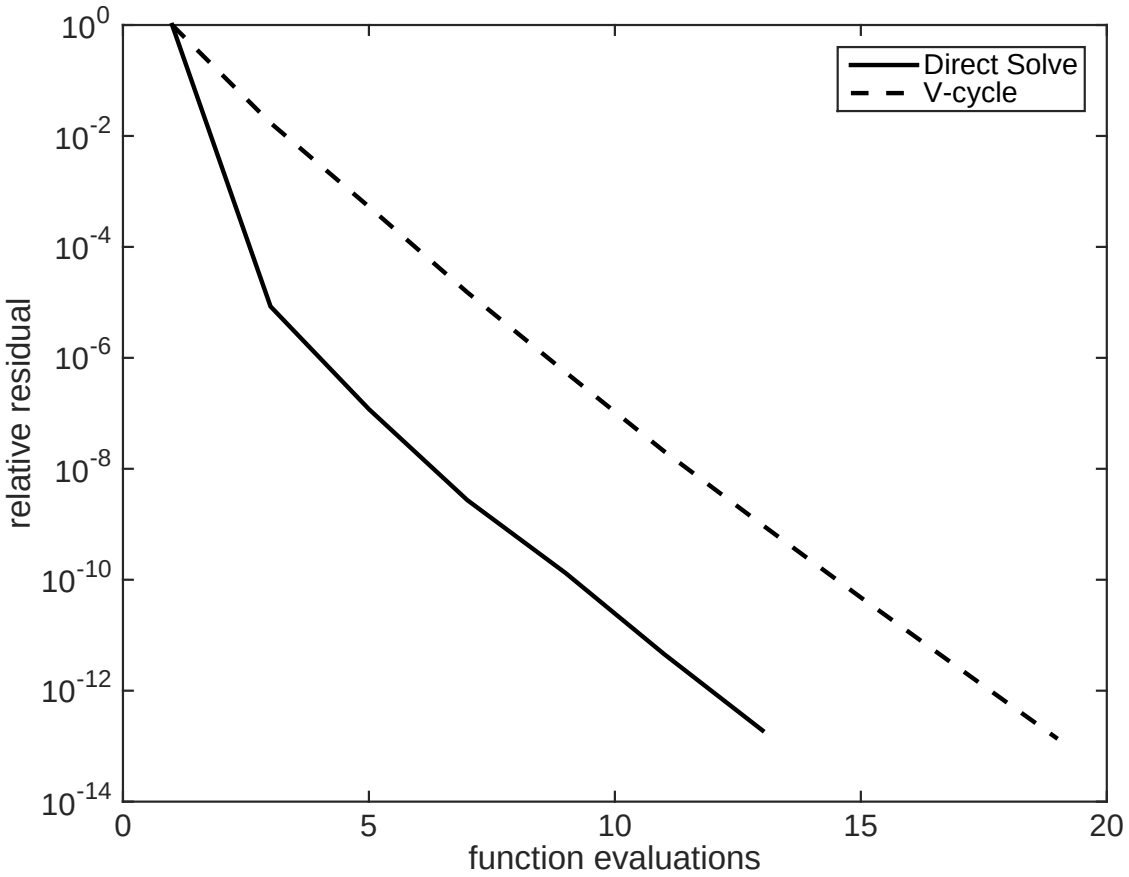
Algorithmic settings

- I used `knl.m` with
  - GMRES with right preconditioning, CGS twice
  - $\eta = .1$, $\tau_r = \tau_a = 10^{-12}$
- V-cycle: $\nu_1 = \nu_2 = 3, nc = 15$

Results

- Performance results independent of h.
- Compare V-cycle with $\mathbf{D}_2$ direct solve
- Timings(secs):
  - V-cycle: 42
  - Direct solver: 160

Residual History



Timings

1/h	VC	ratio	Direct Solve	ratio
32	3.09e-02		2.77e-02	
64	5.34e-02	1.72	1.01e-01	3.66
128	1.19e-01	2.23	3.23e-01	3.18
256	4.66e-01	3.92	1.68e+00	5.22
512	1.97e+00	4.22	7.35e+00	4.36
1024	9.72e+00	4.94	3.41e+01	4.64
2048	4.17e+01	4.29	1.70e+02	4.99

MG for Integral Equations

Fredholm 2nd kind integral equation on $C[0,1]$,

$$u(x) = (Ku)(x) + g(x) = \int_0^1 k(x,y)u(y)\,dy + g(x).$$

Here

- k and g are given continuous functions.
- $u \in C[0,1]$ is the unknown.

We assume that $I - K$ is nonsingular.

Atkinson (73) - Brakhage (60) Method

Notation:

- Sequence of quadrature rules: nodes $\{x_j^m\}_{j=1}^{N_m}$ and weights $\{w_j^m\}_{j=1}^{N_m}$.
- Operators: $K_m(u)(x) = \sum_{j=1}^{N_m} k(x, x_j^m) u(x_j^m) w_j^m$
- Note that K_m is defined on $C[0,1]$.

The sequence $\{K_m\}$ is collectively compact and converges strongly to K.

Defining all operators on $C[0,1]$ is easier than thinking of sequences of problems and operators.

Solving $u - K_m u = f$

First solve the finite-dimensional system for the function values at the nodes

$$u(x_i^m) - \sum_{j=1}^{N_m} k(x_i^m, x_j^m) u(x_j^m) w_j^m = f(x_i^m)$$

(with GMRES, for example).
Then recover u with the Nyström interpolation

$$u(x) = f(x) + \sum_{j=1}^{N_m} k(x, x_j^m) u(x_j^m) w_j^m.$$

Some Facts from Functional Analysis

Given $\rho > 0$ there is l_0 such that if $L \geq l \geq l_0$ the operator

$$B_l^L = I + (I - K_l)^{-1} K_L$$

satisfies

$$\|I - B_l^L (I - K_L)\| \leq \rho.$$

NOTE!! $(I - K_l)^{-1}$ won't work! We will fix that by changing K_l.

Atkinson-Brakhage iteration

Solve $u - K_L u = g$ on a fine mesh by using B_l^L as a preconditioner to fixed point iteration.

$$u_+ = u_c - B_l^L(u_c - K_L u_c - g).$$

Cost: Two fine-mesh calls to K_L.
$r_c = g - (I - K_L)u_c$ and $B_l^L r_c = r_c + (I - K_l)^{-1} K_L r_c$.
Evaluate $(I - K_l)^{-1} w$ via GMRES.

Implementation

You have seen how to compute K_L. The last part is the action of B_l^L on a function u.

- Evaluate $K_L u$.
- Solve $w - K_l w = K_L u$ as above.

A More Efficient Way

- Suppose you have a compact operator K and
- can compute operator-function products.
- Suppose you have a projection P_l onto a finite dimensional subspace V_l and
- P_l converges strongly to I.
- Then KP_L converges to K in the operator norm and
- $(I - KP_L)^{-1} \to (I - K)^{-1}$ in norm.

Faster Two-Grid Method

$$u_+ = u_c - (I - KP_l)^{-1}((I - K)u_c - g)$$

How to evaluate $(I - KP_l)^{-1}w$.

- Solve the finite dimensional problem

$$w_l - P_l K P_l w_l = P_l g$$

- Let

$$w = g - KP_l w_l.$$

Nested Iteration: $K_i = KP_i$; $K = K_L$

Algorithm `gridnest`$(g, u, \{K_i\}, l, L)$

Solve $u^l - \mathcal{K}_l u^l = g$ on the coarse level.

Set $u = u^l$.

for $m = l+1, \ldots L$ **do**

$u = u - (I - KP_l)^{-1}((I - K)u - g)$

end for

Remarks

- Analysis shows that $\|K_l - K_L\|$ is small for sufficiently large l and all $L > l$ (including $L = \infty$, the continuous problem.
- The algorithm can be realized by using piecewise linear interpolation, evaluation at grid points, and full weighting for the coarse-to-fine transfer.

Easy example

$$u(x) - \frac{1}{2}\int_0^1 \sin(x-y)u(y)\,dy = \cos(x)$$

I have already discretized this into the form $(I-K)u = f$, so the hard job is the intergrid transfers.

- Coarse to fine: simple interpolation.
- Fine to coarse: repeated applications of full weighting.
- And here's some MATLAB ...

Nested Iteration

Opimal approach

- Start on coarse grid l; solve to high accuracy.
- Interpolate to fine grid. Solve with multilevel method.
- Keep the coarse level at l for the entire solve!
- You'll need one iteration/level to maintain accuracy to truncation error if the coarse mesh is fine enough.

Example: Source Iteration in Transport Theory

- Let $\phi - K_L\phi = S$ be the fully discrete problem at level L.
- Map from level L to level $l < L$ by full weighting.
- That map has the same properties as a continuous projection.
- Map from l to L by interpolation.
- So $K_l = I_L^l K_L I_l^L$ will do the job.