MA 580; Iterative Methods for Linear Equations

C. T. Kelley
NC State University
tim_kelley@ncsu.edu

Version of October 10, 2016

Read Chapters 2 and 3 of the Red book.

NCSU, Fall 2016

Part VIc: GMRES Examples

Using GMRES

- Do not write your own.
- Matlab has one.
- I have two.
  - Red Book Solvers
    http://www4.ncsu.edu/~ctk/matlab_roots.html
  - KL: new linear solver codes; alpha-testing
    http://www4.ncsu.edu/~ctk/knl.html
- I will use KL for the examples I do.
- Any of the three will do fine for MA 580.

What a GMRES code needs from you.

- The function for the matrix-vector product $\mathbf{Ax}$
  - Matlab will accept $\mathbf{A}$ and do the right thing.
  - It is better practice to write it up as a function.
- The initial iterate $\mathbf{x}_0$.
- The right hand side $\mathbf{b}$
- The termination tolerance.
  Terminate when $\|\mathbf{r}_k\| \leq \tau \|\mathbf{b}\|$
- Storage limits and maximum iterations.

Example I: Very easy integral equation

We will use the trapezoid rule to discretize

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

You'll get $\mathbf{Au} = \mathbf{b}$ where

$$a_{ij} = \delta_{ij} - \frac{\sin(x_i - x_j) w_j}{1 + x_i + x_j},\ 1 \le i, j \le N \text{ and } b_i = \cos(x_i).$$

The trapezoid rule says $x_i = (i-1) * h$, $h = 1/(N-1)$, and

$$w_1 = w_N = h/2; w_i = h,\ 2 \le i \le N-1.$$

The matvec function

Here's a very simple (and inefficient) matvec. simple.m

```
function atv = simple(u)
N=length(u); h=1/(N-1); w=h*ones(N,1);
w(1)=.5*w(1); w(N)=.5*w(N);
a=zeros(N,N); % preallocate storage!!
for i=1:N
   for j=1:N
      a(i,j)=sin((i-j)*h)*w(j)/(1 + (i+j-2)*h);
   end
end
atv=u - a*u;
```

Better version with comments on Moodle page.

Why is this an easy problem?

If you write $\mathbf{A} = \mathbf{I} - \mathbf{M}$ then

$$m_{ij} = sin((i-j) * h) * w_j/(1 + (i+j-2) * h)$$

so

$$|m_{ij}| \leq \sin(1) * w_j < .9 * w_j.$$

Which means that $\|\mathbf{M}\|_\infty < .9$.
In fact, for $N = 101$,

$$\|\mathbf{M}\|_\infty < .3.$$

Do fancier calculus and you'll see why.
$\mathbf{A}$ is also diagonalizable, $\kappa(\mathbf{V}) < 300$.

So how many Krylovs do you need . . .

to make $\|\mathbf{r}_k\|/\|\mathbf{r}_0\| < 10^{-6}$?
I know that $\sigma(\mathbf{A}) \subset (.7, 1.3)$. Using

$$p_k(z) = (1-z)^k \in \mathcal{P}_k$$

in the standard way, I'm there if

$$\kappa(\mathbf{V})10^{-k} = 3 \times 10^{2-k} \leq 10^{-6}.$$

So $k = 9$ does the work.
For 10^{-3} I'd need $k = 6$, which is very conservative.

Querying the Matrix

I got $\kappa(\mathbf{V})$ and the eigenvalues with

```
[V,D]=eig(a);
cond(V)
```

If `cond(V)` is REALLY BIG, **A** is probably not diagonalizable.

KL: linear solvers

- `kl.m` is a single code that lets you use many linear solvers. CG, GMRES, ...
- You control KL with the `kl_optset.m` program.
- If you have already figured out how to use the codes from the red book or the ones built into Matlab, that is ok. If not, use KL.

Getting Started

To solve $Ax = b$ with KL you

- Write a matrix-vector product function for A
- Write a matrix-vector product function for your preconditioner if you need one.
- Organize the data the matvecs need into a data structure to pass to KL
- Set the options and run.

Call kl to solve the problem

kl_simple.m

```
N=101;
h=1/(N-1);
u0=rand(N,1);
%
% Make x a column vector.
%
x=h:h:1-h; x=x';
b=cos(x);
[u, error] = kl(u0, b, @simple);
error
norm(simple(u) - b)
```

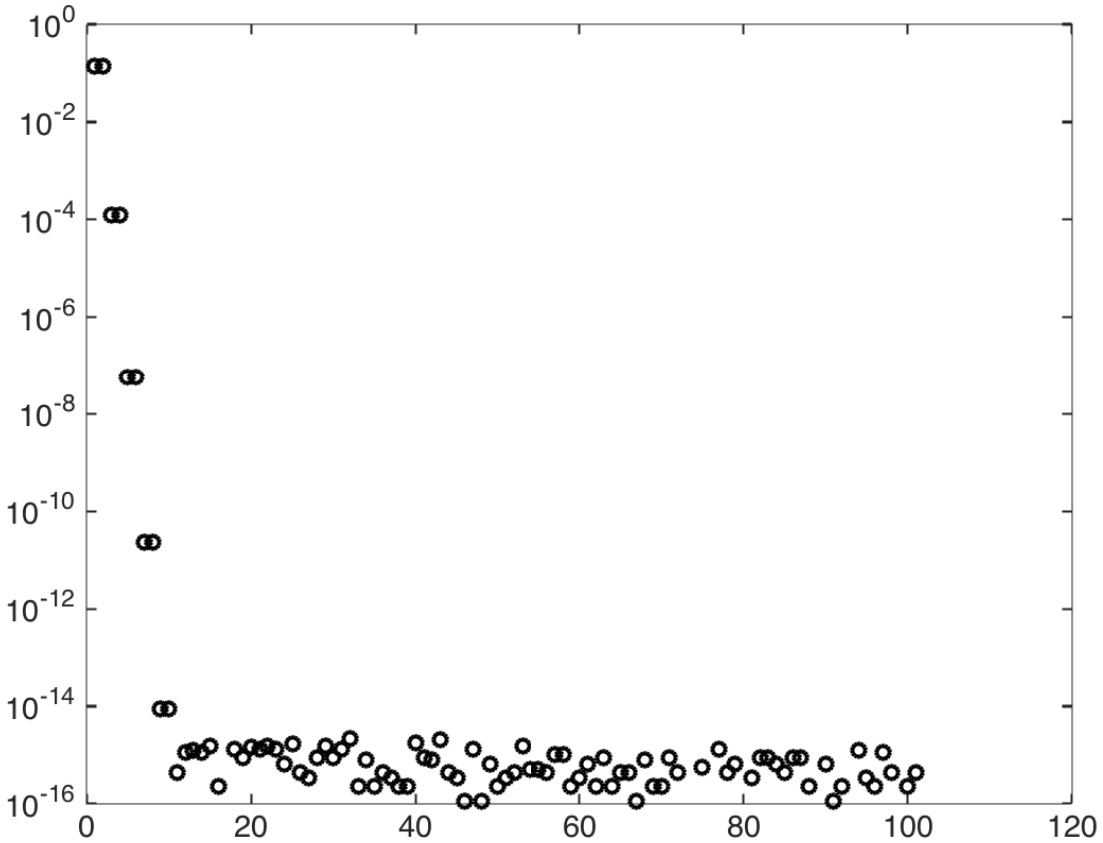
So, what happens?

```
>> kl_simple

error =
   8.5661e+00    1.1903e+00    1.0543e-03
ans =
   1.0543e-03
```

Why did I only need two iterations to drive the relative residual to below 10^{-3}?
Answer: **clustering**! The norm never tells the whole story for integral equations problems.

Eigenvalues of $\mathbf{I} - \mathbf{A}$ for $N = 101$

I'm plotting $|1 - \lambda_i|$.



Perhaps we need more iterations?

Default relative residual tolerance in `kl` is 10^{-3}. Change it to 10^{-10} with

```
options=kl_optset('ltol',1.d-10);
[u,error]=kl(u0,b,@simple,options);
semilogy(error,'-','LineWidth',2);
set(gca,'FontSize',14);
```

The error vector is

```
error =
   8.5661e+00    1.1903e+00    1.0543e-03    1.2780e-07    2.1541e-12
```

What's missing?

We are using the default options, which is rarely the best choice. And ...

- `simple.m` is very inefficient
  - It computes A at every iteration
  - It allocates storage for at every iteration.
- We should do this once in the main program.
- So how do we pass that stuff to the matvec?

Passing data to KL: I

Put the data the matvec needs in a structure or array and pass the structure or array to KL.
Write a function to build the matrix first. build_matrix.m

```
function a = build_matrix(N)
h=1/(N-1); w=h*ones(N,1);
w(1)=.5*w(1); w(N)=.5*w(N);
a=zeros(N,N);
for i=1:N
   for j=1:N
      a(i,j)=sin((i-j)*h)*w(j);
   end
end
a = eye(N) - a;
```

Passing data to KL: II

You use the kl_optset function to pass the data to kl.

```
N=101; h=1/(N-1); u0=zeros(N,1);
%
% Make x a column vector.
%
x=0:h:1; x=x';
b=cos(x);
a = build_matrix(N);
options=kl_optset('matvec_data',a,'ltol',1.d
   -10);
[u, error] = kl(u0, b, @matvec_v2, options);
error
norm(a*u - b)
plot(x,u)
```

The matvec

KL knows about the 'matvec_data' and does the right thing. You write the matvec with the data as a second argument.

```
function atv=matvec_v2(x,a)
atv=a*x;
```

This is very handy if your matvec needs things like angles (transport) or physical parameters.
Do not use global variables for this. Matlab's parallel toolbox does not like global variables.

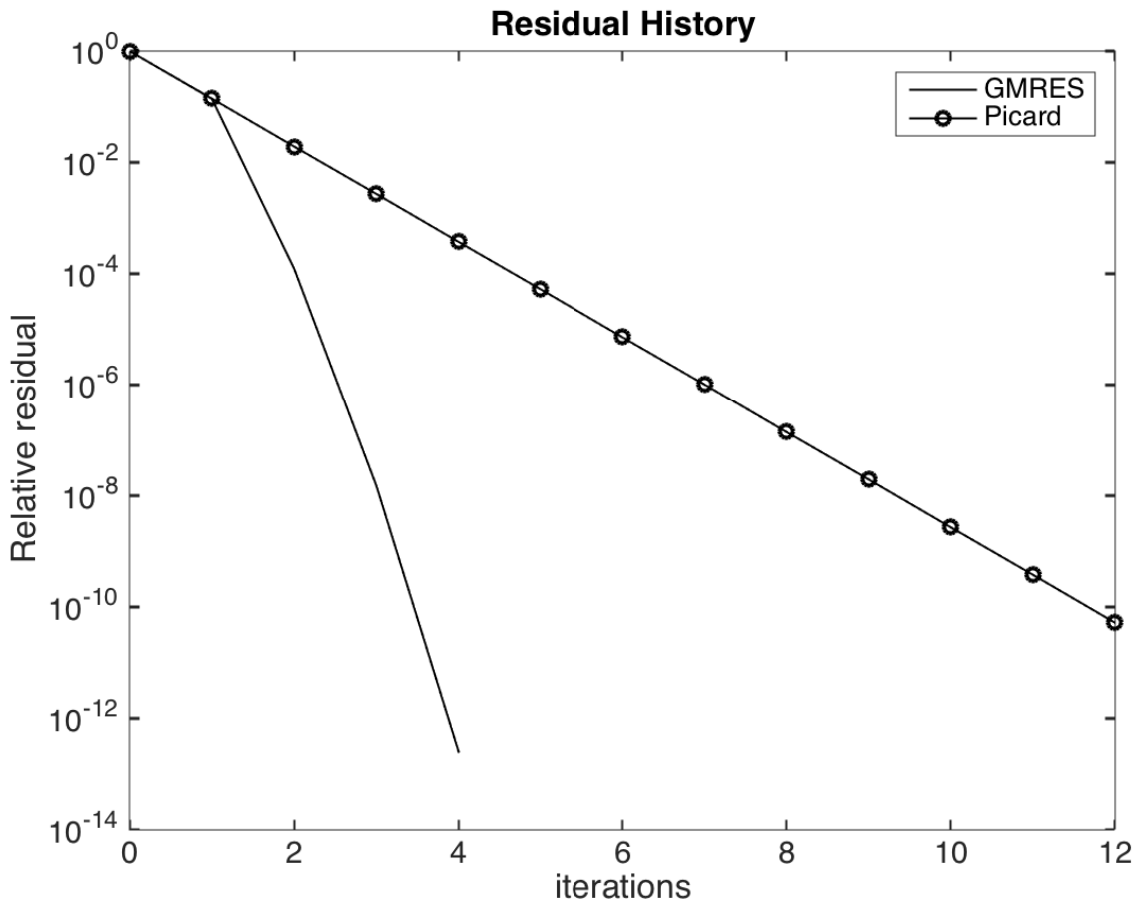
kl_ieq.m

Example on Moodle page.

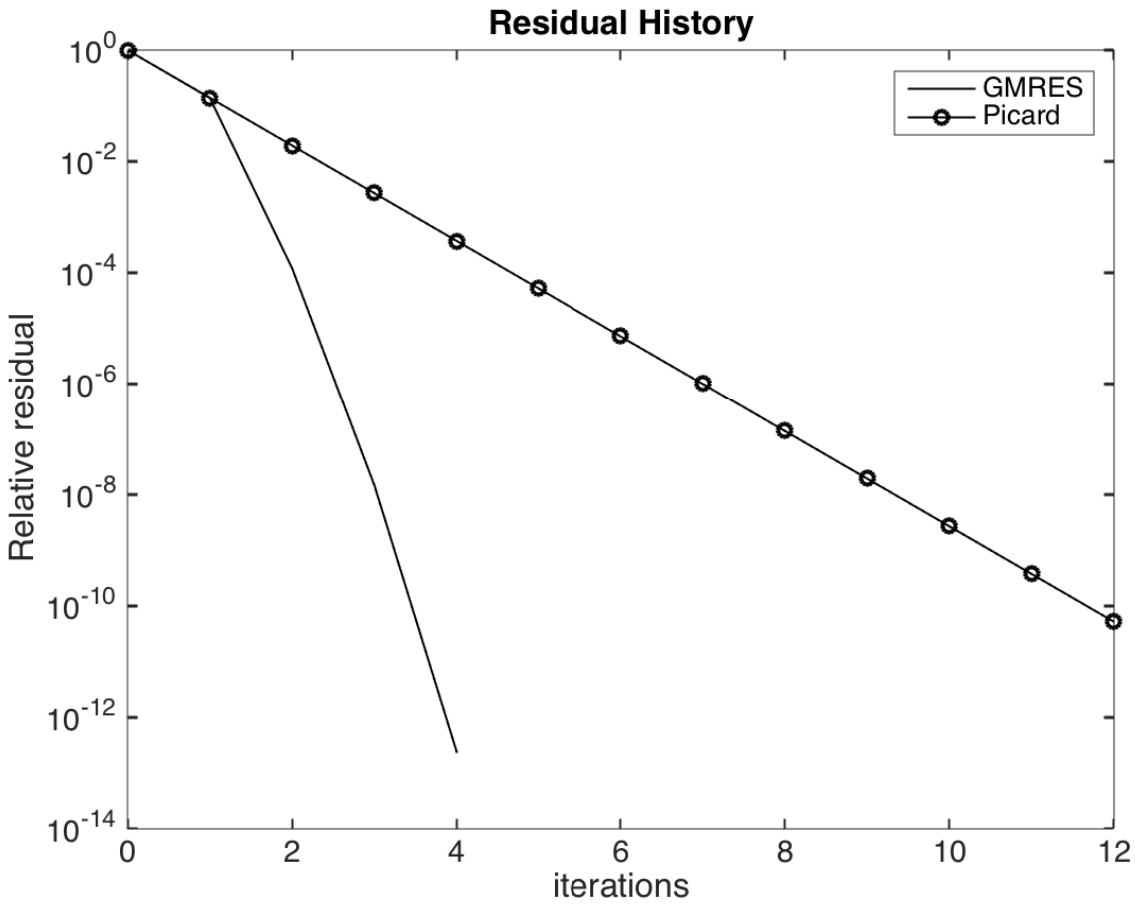
- Runs `kl.m` for the integral equations problem.
- Self-contained
- Compares GGMRES to Picard (Richardson) iteration.
- Makes nice plot with labels for everything.
- Could be a template for your own work.
- Type `help kl_optset` to understand what I did.

And the plot ...

Residual History Plot: $N = 101$



Residual History Plot: $N = 1001$



Conclusions

- Performance is mesh-independent.
- Faster than Picard or LU for this problem:
  - Cost of LU is $N^3/3$
  - Cost of k GMRES is $(k+1)N^2 + O(kN)$
  - Cost of k Picards is kN^2

Why 2nd kind integral equations are Krylov-friendly

$$u(x) - \int_\Omega k(x,y)u(y)\ dy = f(x)$$

If k is reasonable (L^2, continuous) and Ω is bounded then

- Eigenvalues of integral operator cluster at zero.
- Same story for any sensible discretization.
- So only finite many eigenvalues of **A** outside your favorite ball about $z = 1$.
- So the convergence gets faster as the iteration progresses.

Termination: Tim's error correction

Krylov codes terminate when

$$\|\mathbf{r}_k\| < \tau\|\mathbf{b}\|.$$

What do you do if you want

$$\|\mathbf{r}_k\| < \tau\|\mathbf{r}_0\|?$$

Answer, **residual correction** which is . . .

Residual Correction

- Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}$
- Solve $\mathbf{A}\mathbf{e} = \mathbf{r}_0$ with GMRES
  terminate when $\|\mathbf{r}_0 - \mathbf{A}\mathbf{e}_k\| \leq \tau\|\mathbf{r}_0\|$
- Set $\mathbf{x}_k = \mathbf{x} + \mathbf{e}_k$
- Then

$$\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k = \mathbf{b} - \mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{e}_k = \mathbf{r}_0 - \mathbf{A}\mathbf{e}_k$$

- So $\|\mathbf{r}_k\| \leq \tau\|\mathbf{r}_0\|$

Options in KL

You set the options with `kl_optset`. You may set several options at once with

```
options = kl_optset('option1',value,'option2',
    value);
```

and change an existing options structure like this

```
options = kl_optset('option1',value,'option2',value,
    options);
```

Examples

Set the termination tolerance to 10^{-12}; use CG for the solver; set the maximum iterations to 100, and `my_precond.m` for the preconditioner.

```
options = kl_optset('ltol',1.d-12,'lmethod','cg
    ',...
          'maxitl',100,'ptv',@my_precond);
```

Change the options to send `data_m` (matvec) and `data_p` (precond) to the matvec and preconditioner.

```
options = kl_optset('matvec_data',data_m,'p_data',data
    _p);
```

Options: I

- `ltol`: relative residual termination limit; default is 10^{-3}.
- `maxitl`: iteration limit; default 40
- `lmethod`: iterative method; default = 'gmres' (quotes required)
- `matvec_data`: data structure for the matvec
- `p_data`: data structure for the preconditioner
- `ptv`: preconditioner-vector product function

Options: II

- `p_side`: left or right precond? default = 'left'
- `orthog`: 'cgs' or 'mgs', default 'cgs' (but think!)

Boundary value problem

Problem 3.9.6 on page 60 of the Red Book.

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

We will discretize this with central differences. The second deriviative is something we've done before

$$\mathbf{D}_2 = \frac{1}{h^2} \begin{pmatrix} 2 & -1 & 0 & \dots & 0, & 0 \\ -1 & 2 & -1 & ,0 & \dots & 0 \\ 0 & -1 & 2 & -1, & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots, & ,0, & -1 & 2 & -1 \\ 0 & \dots, & \dots,, & 0 & -1 & 2 \end{pmatrix}$$

Second-order Central Difference Approximation of u'

$$\frac{u(x+h) - u(x-h)}{2h} = u'(x) + O(h^2)$$

which you can get directly with Taylor's theorem.
So the matrix representation will be tridiagonal and skew-symmetric

$$\mathbf{D}_1^T = -\mathbf{D}_1$$

with purely imaginary eigenvalues.

First Derivative Matrix Representation

$$\mathbf{D}_1 = \frac{1}{2h}\begin{pmatrix} 0 & 1 & 0 & \dots & 0, & 0 \\ -1 & 0 & 1 & ,0 & \dots & 0 \\ 0 & -1 & 0 & 1, & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots, & ,0, & -1 & 0 & 1 \\ 0 & \dots, & \dots,, & 0 & -1 & 0 \end{pmatrix}$$

Matrix Representation for the BVP

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

$$\mathbf{D}_2 + \mathbf{D}_1 + \mathbf{I} = (1, 1, \ldots, 1)^T.$$

Best way to solve this is $O(N)$ tridiagonal solver.
We'll use this example as a way to see what preconditioning will do for you.

No preconditioning?

Bad, bad idea. Start off with

```
h=1/(N+1); x=h:h:1-h; x=x'; b=ones(N,1);
e=ones(N,1); z=zeros(N,1);
D2 = spdiags([-e 2*e -e],-1:1,N,N); D2=D2/(h*h);
D1=spdiags([-e z e],-1:1,N,N); D1=D1/(2*h);
A=D2+D1+speye(N,N):
u0=zeros(N,1);
```

Call `kl` for $N = 100, 400, 1600$

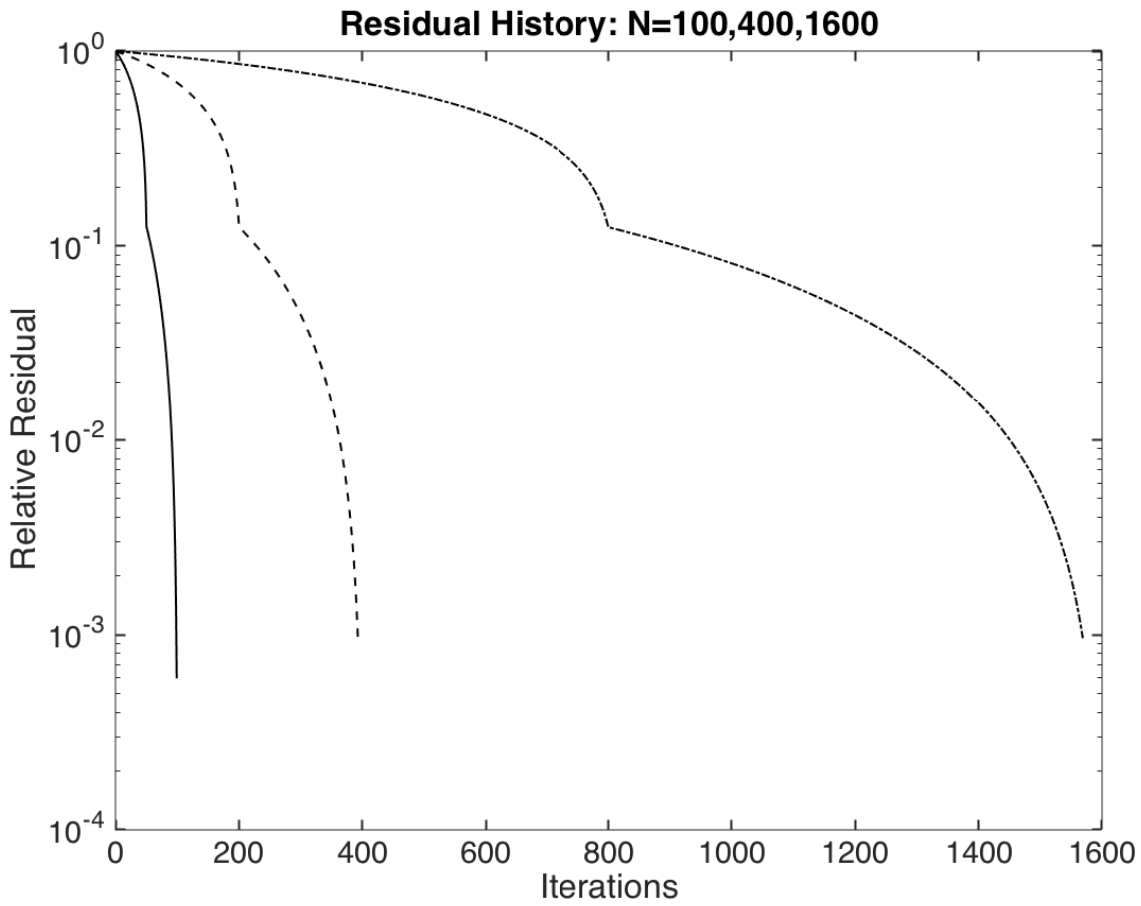
This will be slow, so I'm giving `kl` N iterations.

```
options=kl_optset('ltol',1.d-3,'maxitl',N,'matvec_data
    ',A);
[u,hist]=kl(u0,b,@bvp,options);
```

where

```
function atv=bvp(x,A)
atv=A*x;
```

Residual History Plot: No preconditioning



Preconditioning with $\mathbf{D}_2^{-1}$

My preconditioner-vector product will be

$$\mathbf{Mv} = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{v}$$

where $\mathbf{LU} = \mathbf{D}_2$ is the sparse LU factorization of $\mathbf{D}_2$
In matlab you factor outside of the loop

```
[L,U]=lu(D2);
```

and then pass $\mathbf{L}, \mathbf{U}$ to the function for the preconditioner.

Two ways to precondition.

You can wire the preconditioner into the matvec. For the continuous problem and left preconditioning, the preconditioned problem is

$$u + D_2^{-1}(u' + u) = u + D_2^{-1}(D_1 + I)u.$$

and you'd write that as a matvec.

```
function atv = pbvp(u,L,U,D_1)
atv= u+U\(L\(D_1*u + u));
```

Somehow you have to pass $\mathbf{D}_1$, $\mathbf{L}$, and $\mathbf{U}$ to the function.
The Red Book did it this way and used (GAG!) global variables.

Better way, stand-alone preconditioner

- A wired preconditioned matvec has to be rewritten to go from left to right.
- It's hard to swap preconditioners without doing extra work.
- It's error-prone.

`kl` lets you put the preconditioner in as a separate option.
`kl` needs no global variables.

The matvec

The matvec only needs to know about **A**, so you could use

```
function atv=bvp(x,A)
atv=A*x;
```

You could also do this entirely matrix-free.

Writing the preconditioner

Write $\mathbf{D}_2^{-1}\mathbf{v}$ as a function of $\mathbf{v}, \mathbf{L}, \mathbf{U}$
First, build a data structure for $\mathbf{L}$ and $\mathbf{U}$ outside of the solver.

```
[L,U]=lu(D2);
p_data=struct('L',L,'U',U);
```

then write the preconditioner-vector product to read the struct

```
function ptv=d2inv(v,p_data)
L=p_data.L;
U=p_data.U;
ptv=U\(L\v);
```

and, finally, tell `kl` about it ...

Telling kl to precondition

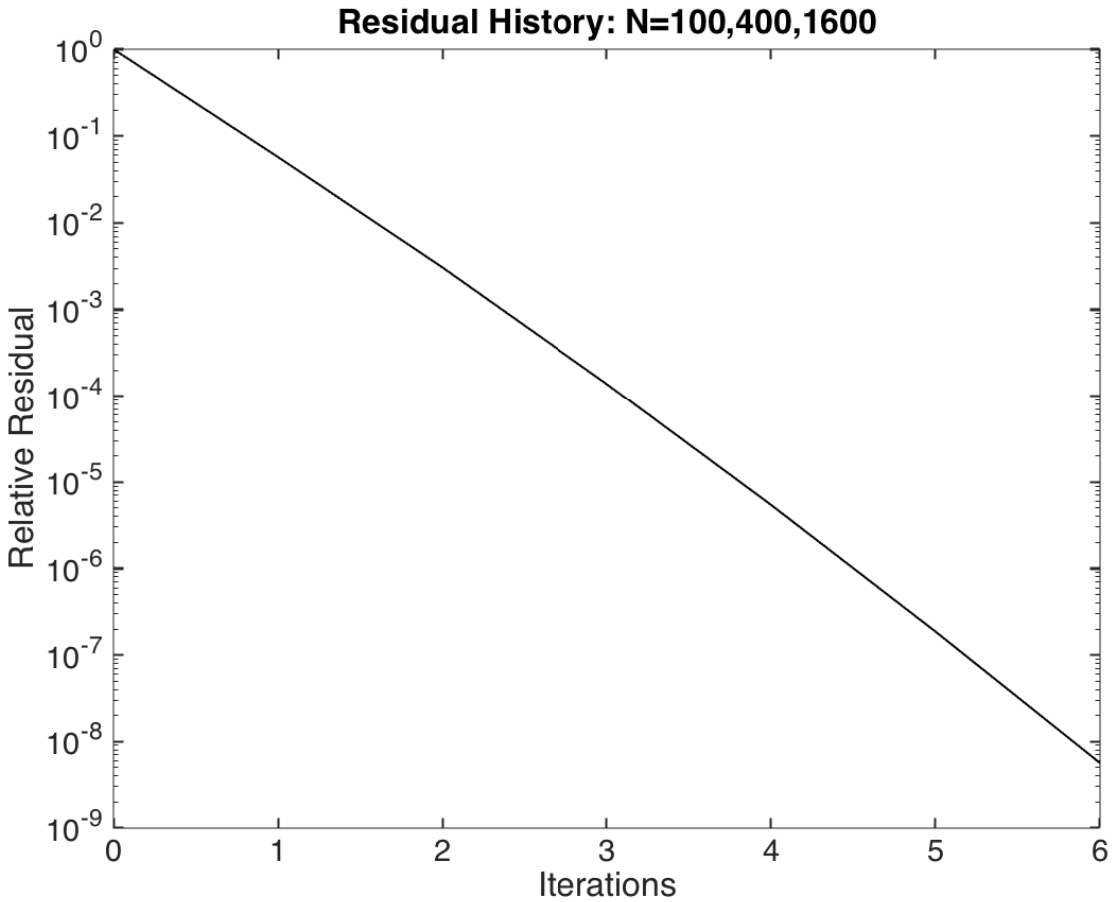
```
options=kl_optset('ltol',1.d-8,'matvec_data',A
    ,...
      'ptv','d2inv','p_data',p_data);
[u,hist]=kl(u0,b,@bvp,options);
```

- Setting the ptv tells kl that the preconditioner is d2inv.
- Setting p_data gives d2inv what it needs.
- Typing help kl_optset will tell you more.
- Code kl_bvp.m lives on the moodle page.

And the results . . .

Residual History Plot: left preconditioning

Where are the three lines?



Right preconditioning: the p_side option.

What? It's different!

Last Example, because we can.

Let's solve

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

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

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

and $f(x) = e^{10x}\sin(10x)$.
You're insane to do this with no preconditioning.
Objective: reduce the residual by a factor of 10^{-8}, so we must precondition from the **right**.

Trapezoid rule and boundary conditions

STAY AWAKE!!!

The uknowns are the **interior grid points** This means that

- The integration rule does not include the endpoints. **The sum in the trapezoid rule is over the interior gridpoints!!!**
- So N unknowns means $h = 1/(N+1)$ and $w(i) = h$ for all $1 \leq i \leq N$

The matvec

All we need to do is combine what we had before, paying attention to the boundary conditions.

```
h=1/(N+1);
w=h*ones(N,1);
M=zeros(N,N);
for i=1:N
    for j=1:N
        M(i,j)=sin((i-j)*h)*w(j)/(1 + (i+j)*h);
    end
end
e=ones(N,1);
D2 = spdiags([-e 2*e -e],-1:1,N,N); D2=D2/(h*h);
A = D2 + M;
```

The matvec and the pvec are the same

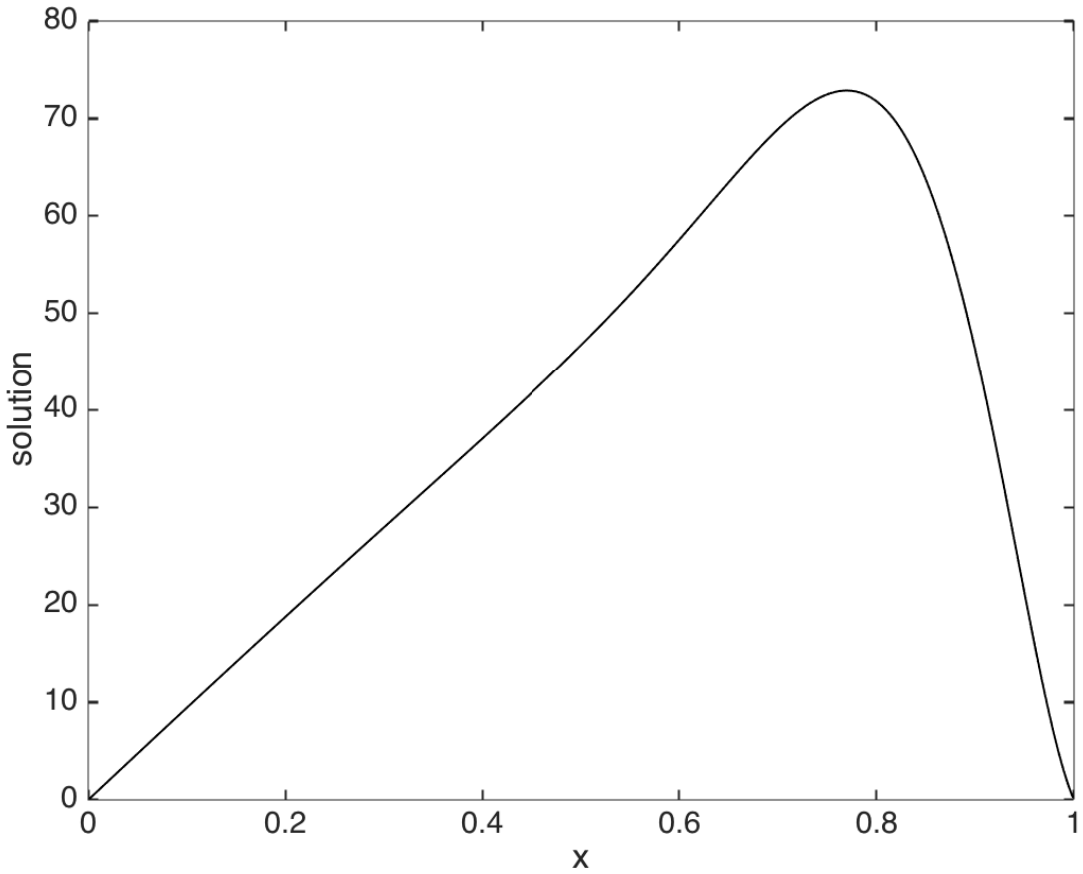
```
function atv=ide(x,A)
atv=A*x;

function ptv=d2inv(v,p_data)
L=p_data.L;
U=p_data.U;
ptv=U\(L\v);
```

Setting up kl

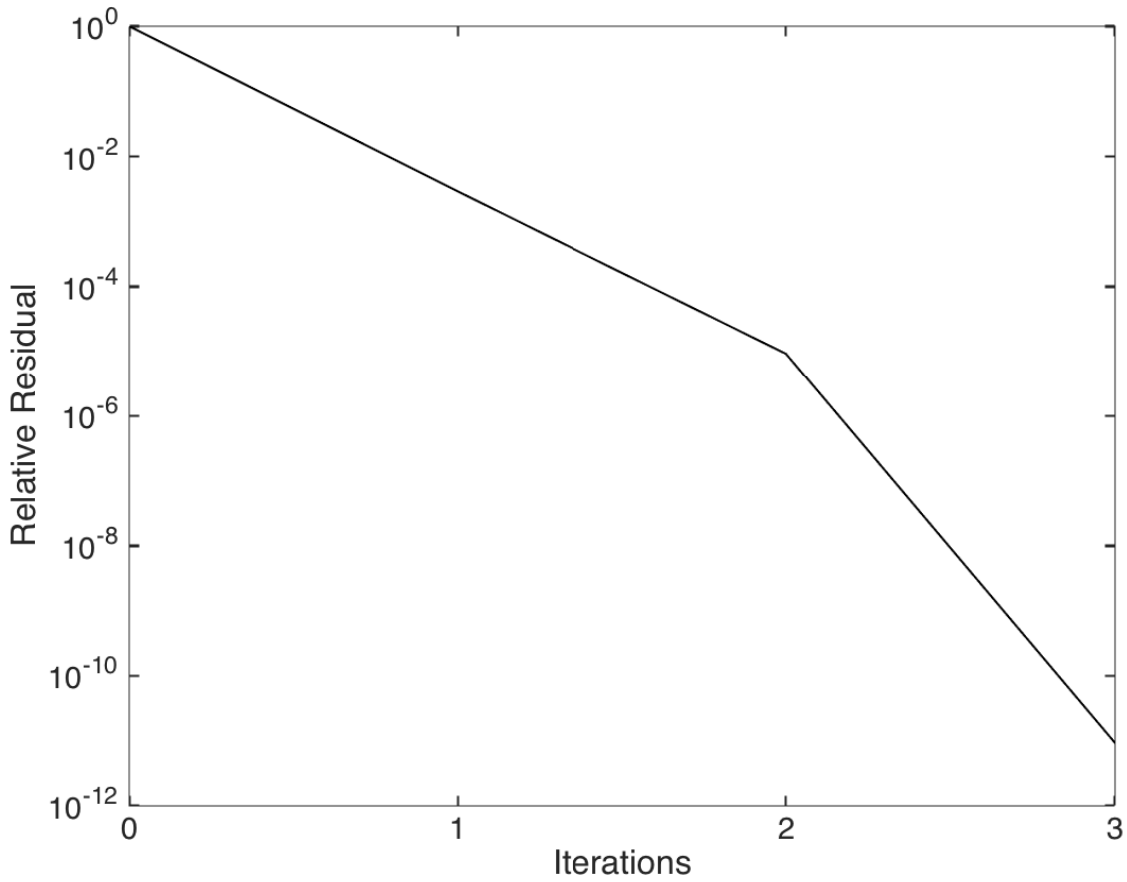
```
options=kl_optset('ltol',1.d-8,'matvec_data',A
    ,...
      'ptv',@d2inv,'p_data',p_data,'p_side','
         right');
```

Code: kl_ide.m on Moodle page.

Solution $N = 999, h = 10^{-3}$



Residuals $N = 999$. Easy problem.



Grid Refinement Study

- Use a sequence of grids $\{x_i^l\}_{i=1}^{2^l-1}$
- $h_l = 2^{-l}$
- Expect errors to drop by factor of 4

Questions:

- How to move from one grid to the next?
- How to compare results from different grids?
- How to estimate errors?

Intergrid Transfers: I

- I_s^t: Transfer between $h = s$ (source) to $h = t$ (target).
- We use only I_h^{2h} (fine to coarse) and I_{2h}^h (coarse to fine)

Intergrid Transfers: II

Restriction by injection

$$(I_h^{2h}\mathbf{u})_i = u_{2i}, 1 \le i \le 2^l - 1 \equiv 1/h - 1.$$

Prolongation by interpolation

$$(I_{2h}^{h}\mathbf{u})_i = \begin{cases} u_{i/2} & \text{if } i \text{ is even} \\ (u_{(i-1)/2} + u_{(i+1)})/2 & \text{if } i \text{ is odd} \end{cases}$$

Here we use the boundary conditions to make $v_0 = v_{N+1} = 0$.
Here's a picture.

Some Matlab: only works for zero bc

```
function uf = ctof(uc)
nc=length(uc); nf=2*(nc+1) - 1;
uf=zeros(nf,1);
%
% Use zero boundary conditions in the
    interpolation.
%
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;
```

Fine to coarse is easier

```
function uc=ftoc(uf)
nf=length(uf); nc=.5*(nf-1)-1;
uc=uf(2:2:nf-1);
```

Error Estimation

Notation:

- $\mathbf{A}_h$ is the matrix for meshwidth h
- $\mathbf{f}^h$ is the right side.
- $\mathbf{u}^h \approx \mathbf{A}_h^{-1}\mathbf{f}^h$ is the solution we get from GMRES.

If u^h is accurate to truncation error then

$$e_i^h = u_i^h - u(x_i^h) = O(h^2)$$

where $x_i^h = i * h$. In that case we should see

More Error Estimation

- $\|e^{2h}\|_\infty \approx E_{2h} \equiv \|\mathbf{u}^h - I_{2h}^h \mathbf{u}^{2h}\|_\infty = O(h^2)$,
- and $E_{2h}/E_h \approx 4$ as $h \to 0$.
- Using the ℓ^2 norm is trickier since $\|\mathbf{1}\|_2 = \sqrt{N}$ so

$$\|\mathbf{u}^h - I_{2h}^h \mathbf{u}^{2h}\|_2 = O(h^2 N^{1/2}) = O(h^{1.5})$$

- Fix: use relative error and look at

$$\|\mathbf{u}^h - I_{2h}^h \mathbf{u}^{2h}\|_2 / \|\mathbf{u}^h\|_2 = O(h^2).$$

I like the ℓ^∞ norm to avoid confusion.

Example: IDE

The code itself `error_study.m` is pretty simple.

```
for p=1:levels
    N=2*(N+1)-1;
    uc=ctof(u);
    [u,hist]=kl_ide(N);
    grid_data(p,1)=norm(uc-u,inf);
    if p > 1
       grid_data(p,2)=grid_data(p-1,1)/grid_data
           (p,1);
    end
end
```

As is the output

```
>> grid_data=error_study

grid_data =

   1.1618e+00             0
   3.2811e-01   3.5410e+00
   8.6743e-02   3.7826e+00
   2.2275e-02   3.8941e+00
   5.6426e-03   3.9477e+00
```