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 $A\mathbf{x}$
  - Matlab will accept $A$ 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 $\|r_k\| \leq \tau\|\mathbf{b}\|$
- 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 \( Au = b \) where

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

The trapezoid rule says \( x_i = (i - 1) \times h, \quad h = 1/(N - 1), \) and

\[ w_1 = w_N = h/2; \quad w_i = h, \quad 2 \leq i \leq N - 1. \]
The matvec function

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

```matlab
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 $A = I - M$ then

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

so

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

Which means that $\|M\|_{\infty} < .9$.  

In fact, for $N = 101$,

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

Do fancier calculus and you’ll see why.  

$A$ is also diagonalizable, $\kappa(V) < 300$.  

So how many Krylovs do you need . . .

to make $\|r_k\|/\|r_0\| < 10^{-6}$?
I know that $\sigma(A) \subset (.7, 1.3)$. Using

$$p_k(z) = (1 - z)^k \in P_k$$

in the standard way, I’m there if

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

So $k = 9$ does the work.
For $10^{-3}$ I’d need $k = 6$, which is very conservative.
I got $\kappa(V)$ and the eigenvalues with

$$[V,D]=\text{eig}(a);$$

$\text{cond}(V)$

If $\text{cond}(V)$ 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

```matlab
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?

```matlab
>> 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

```matlab
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) = 0.5*w(1); w(N) = 0.5*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;

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

\[
N = 101; \quad h = 1/(N-1); \quad u0 = \text{zeros}(N,1);
\]

\[
\% \% \text{Make } x \text{ a column vector.}
\%
\%
\]

\[
x = 0:h:1; \quad x = x';
\]

\[
b = \cos(x);
\]

\[
a = \text{build\_matrix}(N);
\]

\[
\text{options} = \text{kl\_optset}('\text{matvec\_data}', a, 'ltol', 1.d -10);
\]

\[
[u, \text{error}] = \text{kl}(u0, b, @\text{matvec\_v2}, \text{options});
\]

\[
\text{error norm}(a*u - b)
\]

\[
\text{plot}(x, u)
\]
The matvec

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

```matlab
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 \( \Omega \) 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

$$\|r_k\| < \tau \|b\|.$$  

What do you do if you want

$$\|r_k\| < \tau \|r_0\|?$$

Answer, residual correction which is . . .
Residual Correction

- Compute $r_0 = b - Ax$
- Solve $Ae = r_0$ with GMRES
  terminate when $\|r_0 - Ae_k\| \leq \tau \|r_0\|$
- Set $x_k = x + e_k$
- Then

$$r_k = b - Ax_k = b - Ax - Ae_k = r_0 - Ae_k$$

- So $\|r_k\| \leq \tau \|r_0\|
Options in KL

You set the options with \texttt{kl\_optset}. You may set several options at once with

\begin{verbatim}
options = kl_optset('option1', value, 'option2', value);
\end{verbatim}

and change an existing options structure like this

\begin{verbatim}
options = kl_optset('option1', value, 'option2', value, options);
\end{verbatim}
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.

```matlab
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.

```matlab
options = kl_optset('matvec_data',data_m,'p_data',data_p);
```
Options: 1

- \( ltol \): 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 precondition? default = 'left'
- `orthog`: 'cgs' or 'mgs', default 'cgs' (but think!)
Boundary value problem

Problem 3.9.6 on page 60 of the Red Book.

\[-u'' + u' + u = 1, \quad u(0) = u(1) = 0.\]

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

\[
D_2 = \frac{1}{h^2} \begin{pmatrix}
2 & -1 & 0 & \ldots & 0, & 0 \\
-1 & 2 & -1 & \ldots & 0 & 0 \\
0 & -1 & 2 & -1 & \ldots & 0 \\
\vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\
0 & \ldots, & 0 & -1 & 2 & -1 \\
0 & \ldots, & \ldots, & 0 & -1 & 2
\end{pmatrix}
\]
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

\[ D_1^T = -D_1 \]

with purely imaginary eigenvalues.
First Derivative Matrix Representation

\[ D_1 = \frac{1}{2h} \begin{pmatrix}
0 & 1 & 0 & \ldots & 0, & 0 \\
-1 & 0 & 1 & \ldots & 0 & 0 \\
0 & -1 & 0 & 1 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \ddots & \ddots \\
0 & \ldots & 0 & -1 & 0 & 1 \\
0 & \ldots & \ldots & \ldots & 0 & -1 & 0
\end{pmatrix} \]
Matrix Representation for the BVP

\[-u'' + u' + u = 1, \quad u(0) = u(1) = 0.\]

\[D_2 + D_1 + I = (1, 1, \ldots, 1)^T.\]

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

\[
\begin{align*}
h &= 1/(N+1); \quad x = h:h:1-h; \quad x = x'; \quad b = \text{ones}(N,1); \\
e &= \text{ones}(N,1); \quad z = \text{zeros}(N,1); \\
D2 &= \text{spdiags}([-e \ 2*e \ -e],[-1:1,N,N]); \quad D2 = D2/(h*h); \\
D1 &= \text{spdiags}([-e \ z \ e],[-1:1,N,N]); \quad D1 = D1/(2*h); \\
A &= D2 + D1 + \text{speye}(N,N); \\
u0 &= \text{zeros}(N,1);
\end{align*}
\]
Call kl for \( N = 100, 400, 1600 \)

This will be slow, so I’m giving \( \text{kl} \) \( N \) iterations.

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

where

```matlab
function atv = bvp(x, A)
atv = A * x;
```
Residual History Plot: No preconditioning
Preconditioning with $D_2^{-1}$

My preconditioner-vector product will be

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

where $LU = D_2$ is the sparse LU factorization of $D_2$

In MATLAB you factor outside of the loop

$$[L,U] = \text{lu}(D2);$$

and then pass $L, U$ 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.

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

Somehow you have to pass $D_1$, $L$, and $U$ 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

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

You could also do this entirely matrix-free.
Writing the preconditioner

Write $D_2^{-1}v$ as a function of $v, L, U$
First, build a data structure for $L$ and $U$ outside of the solver.

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

then write the preconditioner-vector product to read the struct

```matlab
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 \texttt{kl} to precondition

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

- Setting the \texttt{ptv} tells \texttt{kl} that the preconditioner is \texttt{d2inv}.
- Setting \texttt{p_data} gives \texttt{d2inv} what it needs.
- Typing \texttt{help kl_optset} will tell you more.
- Code \texttt{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

\[-u''(x) + \int_0^1 k(x, y)u(y) \, dy = f(x), \quad u(0) = u(1) = 0.\]

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.
STAY AWAKE!!!
The unknowns 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 $1 \leq i \leq N$
All we need to do is combine what we had before, paying attention to the boundary conditions.

```matlab
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 vec are the same

```matlab
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

```matlab
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 \( \{x_i^j\}_{i=1}^{2^l-1} \)
- \( h_i = 2^{-l} \)
- 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 $I_{2h}^h$ (fine to coarse) and $I_{2h}^h$ (coarse to fine)
Restriction by injection

\[(I_{2h}^2u)_i = u_{2i}, 1 \leq i \leq 2^l - 1 \equiv 1/h - 1.\]

Prolongation by interpolation

\[(I_{2h}^h 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.
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:
- $A_h$ is the matrix for meshwidth $h$
- $f^h$ is the right side.
- $u^h \approx A_h^{-1} f^h$ 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 $x_i^h = i \times h$. In that case we should see
More Error Estimation

- $\|e^{2h}\|_\infty \approx E_{2h} \equiv \|u^h - l_{2h}^h u^{2h}\|_\infty = O(h^2)$,
- and $E_{2h}/E_h \approx 4$ as $h \to 0$.
- Using the $\ell^2$ norm is trickier since $\|1\|_2 = \sqrt{N}$ so
  
  $$\|u^h - l_{2h}^h u^{2h}\|_2 = O(h^2 N^{1/2}) = O(h^{1.5})$$

- Fix: use relative error and look at
  
  $$\|u^h - l_{2h}^h u^{2h}\|_2 / \|u^h\|_2 = O(h^2)$$

I like the $\ell^\infty$ norm to avoid confusion.
Example: IDE

The code itself error_study.m is pretty simple.

```matlab
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
```matlab
>> 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
```