KL

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

Research Supported by NSF, DOE, ARO, USACE

DTU ITMAN, 2011

Outline

KL: linear solvers

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

Getting Started: I

To solve $Ax = b$ with KL you

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

Getting Started: II

Make sure that `kl.m` and `kl_optset.m` are in your MATLAB path. We will solve the integral equation in Monday's exercises.

$$a_{ij} = \delta_{ij} - \sin(x_i - x_j) w_j / 2, \; 1 \le i, j \le N \text{ and } \beta_i = \cos(x_i).$$

Here $x_i = i * h, \; h = 1/(N+1)$, 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.

```
function atv = simple(x)
N=length(x); h=1/(N-1);
for i=1:N
    w(i) = h;
end
w(1)=.5*w(1); w(N)=.5*w(N);
for i=1:N
   for j=1:N
      a(i,j)=sin((i-j)*h)*w(j);
   end
end
atv=x - a*x;
```

Call kl to solve the problem

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

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.

```
function a = build_matrix(N)
h=1/(N-1);
for i=1:N
    w(i) = h;
end
w(1)=.5*w(1); w(N)=.5*w(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 {\tt 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);
[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.

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