The method of conjugate gradients

The conjugate gradient iteration is as follows:

Start with $k = 0$, $x_0 = 0$, $r_0 = b$ and $p_0 = r_0$

Repeat

1. $k = k + 1$
2. $q = Ap_{k-1}$
3. $\alpha_k = r_{k-1}^* r_{k-1} / (p_{k-1}^* q)$
4. $x_k = x_{k-1} + \alpha_k p_{k-1}$
5. $r_k = r_{k-1} - \alpha_k q$
6. $\beta_k = r_k^* r_k / r_{k-1}^* r_{k-1}$
7. $p_k = r_k + \beta_k p_{k-1}$

until convergence

Main cost per step: $q = Ap_{k-1}$ which is the cost of applying A

We present an implementation of the CG method for solving $Ax = b$ where A is assumed to be a positive definite matrix.

```
function x = myCG(A,b,x0,MaxIts,ErrTol)
% Solve the positive definite system Ax = b by the CG method.

% ErrTol is the desired accuracy
% MaxIts is the maximum number of CG steps
% x0     is the initial guess, the default value is x0 = 0

if nargin < 5,
ErrTol = 1.e-9;
end
if nargin < 4,
MaxIts = 100;
end
if nargin < 3,
        x0 = zeros(length(b),1);
end

% Initialization
k = 0; x = x0;                          % Initialize x
r = b - A*x; p = r;                     % Initialize residuals and
                                        % conjugate direction
delta.new = r'*r; delta0 = delta.new;

while (k < MaxIts) && (delta.new > ErrTol^2*delta0)
  q = A*p;
  alpha = delta.new/(p'*q);
  x = x+alpha*p;                        % Update x

  if mod(k+1,50) == 0                   % Update r
    r = b - A*x;                        % If iteration count is divisible
  else                                  % by 50, recompute the residuals to
    r = r - alpha*q;                    % prevent accumulation of numerical
  end                                   % errors

  delta.old = delta.new;
  delta.new = r'*r;                     % Update delta
  p = r + (delta.new/delta.old)*p;      % Update d

  k = k+1;                              % Increment iteration count
end
```

We are now going to test this routine:

```
n = 400;                                % Problem size
[Q,R] = qr(randn(n));                   % Make random orthonormal matrix
lmin = 1; lmax = 10;
lambda = 1 + (lmax-lmin).*rand(n,1);  % Choose eigenvalues in [1,10]
A = Q*diag(lambda)*Q';                  % Make A with given eigenvalues
b = randn(n,1);                         % Make b

x = A\b;
xx = myCG(A,b);
norm(x - xx)/norm(x)                    % Check accuracy

ans =

   1.1482e-09
```

The number of CG steps is 31 in the above example.

We now perform a large scale computation

```
n = 6000;                               % Problem size
[Q,R] = qr(randn(n));                   % Make random orthonormal matrix
lmin = 1; lmax = 10;
lambda = 1 + (lmax-lmin).*rand(n,1);  % Choose eigenvalues in [1,10]
A = Q*diag(lambda)*Q';                  % Make A with given eigenvalues
b = randn(n,1);                         % Make b

tic, x = A\b; toc
Elapsed time is 30.087873 seconds.

tic, xx = myCG(A,b); toc
Elapsed time is 5.266198 seconds.

norm(x - xx)/norm(x)                    % Check accuracy

ans =

   1.0785e-09
```

On this example, CG is about 6 times faster and is accurate. The number of CG steps is 32! Seems independent of problem size (more later). With that being said, one needs to be cautious here. For instance, some of the routines in matlab may be hardcoded in C.

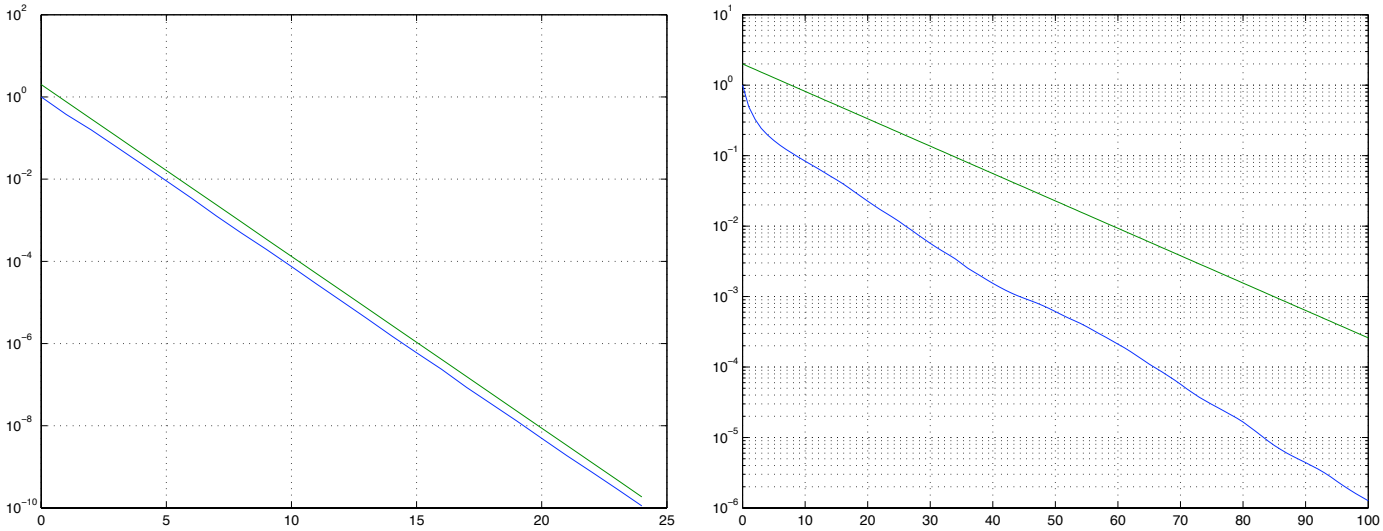

Figure 1: The blue curve is the achieved accuracy and the green curve is the theoretical upper bound

Check convergence of CG for various condition numbers

```
% Test CG convergence
n = 2000;                                % Problem size
[Q,R] = qr(randn(n));                    % Make random orthonormal matrix
lambda = 1 + (lmax-lmin).*rand(n,1);     % Choose eigenvalues in [lmin, lmax]
A = Q*diag(lambda)*Q';                   % Make A with given eigenvalues
x = randn(n,1);
b = A*x;
```

Two experiments with lmax/lmin = 5 and with lmax/lmin = 500.