Notes for Ma221 Lecture 14, Nov 29, 2023

Krylov Subspace Methods: GMRES and CG

Now we return to Chapter 6, on solving A*x=b.
Given the orthogonal basis of the Krylov subspace Q_k = [q_1,...,q_k],
our goal is to find the "best" approximate solution x_k in span(Q_k).
To make progress, we need to define "best"; depending on what we choose,
we end up with different algorithms:

(1) Choose x_k to minimize || x_k - x ||_2, where x is the true solution.
Unfortunately, we don't have enough information in Q_k and H_k = Q_k^T*A*Q^k
(or T_k when A is symmetric) to compute this.

(2) Choose x_k to minimize the norm || r_k ||_2 of the residual r_k = b-
A*x_k.
We can do this, and the algorithm is called MINRES (for "minimum residual")
when A is symmetric, and GMRES (for "generalized minimum residual") when it
is not.

(3) Choose x_k so that r_k is perpendicular to {script K}_k, i.e. r_k^T * Q_k
= 0.
This is called the orthogonal residual property, or a Galerkin condition, by
analogy to finite elements. When A is symmetric, the algorithm is called
SYMMLQ. When A is nonsymmetric, a variant of GMRES works.

(4) When A is spd, it defines a norm || r ||_{A^(-1)} = (r^T * A^(-1) *
r)^(1/2).
We say the best solution minimizes || r_k ||_{A^(-1)}. Note that
   || r_k ||_A^(-1)^2 = r_k^T*A^(-1)*r_k
                      = (b-A*x_k)^T*A^{-1)*(b-A*x_k)
                      = (A*x-A*x_k)^T*A^{-1)*(A*x-A*x_k)
                      = (x-x_k)^T*A*(x-x_k)
                      = || x - x_k ||_A^2
This algorithm is called the Conjugate Gradient Algorithm.

Thm: When A is spd, definitions (3) and (4) of "best" are equivalent, and
using the Conjugate Gradient algorithm, it is possible to compute x_k
from previous iterates for the cost of one multiplication by A, and
a small number of dot products and saxpys (y = alpha*x + y), keeping
only 3 vectors in memory.

More generally, the choice of algorithm depends on the properties of A:
Decision Tree: Figure 6.8 (IterativeTreeAx=b.ps)
See also pointer on class web page to the on-line book
   Templates for the Solution of Linear Systems
for an expanded version of this tree, and pointers to software.

We now discuss GMRES, which uses the least structure of the matrix,
and then CG, which uses the most.

In GMRES, at each step we choose x_k to minimize
   || r_k ||_2 = || b - A*x_k ||_2
where x_k = Q_k*y_k, i.e. x_k is in {script K}_k(A,b).
Thus we choose y_k to minimize
   || r_k ||_2 = || b - A*Q_k*y_k ||_2
               = || b - A*[Q_k,Q_u]*[y_k; 0] ||_2
               = || Q^T*(b - A*[Q_k,Q_u]*[y_k; 0]) ||_2
               = || Q^T*b - H*[y_k; 0]) ||_2
               = || e_1 * ||b||_2 - [H_k H_uk] * [y_k] ||_2

```
                                  [ H_ku H_u ]   [  0  ]
              = || e_1 * ||b||_2 - [ H_k  ] * y_k ||_2
                                  [ H_ku ]
```
Since only the first row of H_ku is nonzero, we just need to solve the
(k+1)-by-k upper Hessenberg least squares problem
```
    || r_k ||_2 = || e_1 * ||b||_2 - [ H_k           ] * y_k ||_2
                  ||                 [ 0...0 H(k+1,k) ]
```
which can be done inexpensively with k Givens rotations, exploiting the
upper Hessenberg structure, costing just O(k^2) or O(k) instead of O(k^3).

Now we return to CG, and begin by showing that when A is spd,
it is "best" in two senses:

Lemma: When A is spd, (3) and (4) are equivalent:
   (3) Choose x_k so that r_k^T * Q_k = 0.
   (4) Choose x_k to minimize || r_k ||_A^(-1) = || x_k - x ||_A
       which are solved by
       (*) x_k = Q_k*T_k^(-1)*Q_k^T*b = Q_k*T_k^(-1)*e_1*||b||_2 ,
       where T_k is the tridiagonal matrix computed by Lanczos.
       Also r_k = +- ||r_k||_2*q_(k+1) .

Here is some intuition for (*):
   Multiplying Q_k^T*b = e_1*||b||_2 projects b onto the Krylov subspace
      spanned by Q_k
   T_k^(-1) is the inverse of the projection of A onto the Krylov subspace
   Multiplying by Q_k maps back from the Krylov subspace to R^n

Proof: Drop the subscript k for simpler notation, so Q=Q_k, T=T_k,
x = Q*T^(-1)*e_1*||b||_2, r = b-A*x and T = Q^T*A*Q is spd (and so
nonsingular), since A is. Then we need to confirm that
```
   Q^T * r = Q^T * (b - A*x)
           = Q^T*b - Q^T*A*x
           = e_1*||b||_2 - Q^T*A*Q*T^(-1)*e_1*||b||_2
           = e_1*||b||_2 - T*T^(-1)*e_1*||b||_2
           = 0  as desired
```
Now we need to show that this choice of x minimizes || r ||_A^{-1}^2,
so consider a different x' = x + Q*z and r' = b - A*x' = r - A*Q*z
```
   || r' ||_A^(-1)^2 = r'^T * A^(-1) * r'
                     = (r - A*Q*z)^T*A^(-1)*(r - A*Q*z)
                     = r^T*A^(-1)*r - 2(A*Q*z)^T*A^(-1)*r +
(A*Q*z)^T*A^(-1)*(A*Q*z)
                     = r^T*A^(-1)*r - 2*z^T*Q^T*A*A^(-1)*r +
(A*Q*z)^T*A^(-1)*(A*Q*z)
                     = r^T*A^(-1)*r - 2*z^T*Q^T*r + (A*Q*z)^T*A^(-1)*(A*Q*z)
                     = r^T*A^(-1)*r               + (A*Q*z)^T*A^(-1)*(A*Q*z)
                           since Q^T*r = 0
                     = || r ||_A^(-1)^2           + || A*Q*z ||_A^(-1)^2
```
and this is minimized by choosing z = 0, so x' = x as desired.

Finally note that r_k = b - A*x_k must be in {script K}_(k+1), so
that r_k is a linear combination of columns of Q_{k+1} = [q_1,...,q_(k+1)].
But since r_k is perpendicular to the columns of Q_k = [q_1,...,q_k],
r_k must be parallel to q_(k+1), so r_k = +-||r_k||_2 * q_(k+1)

There are several ways to derive CG. We will take the most "direct" way
from the formula (*) above, deriving recurrences for 3 sets of vectors,
of which we only need to keep the most recent ones: x_k, r_k, and
so-called conjugate gradient vectors p_k.
(1) The p_k's are called gradients because each step of CG can be thought
of as moving x_(k-1) along the gradient direction p_k

(i.e. x_k = x_(k-1) + nu * p_k) until it minimizes || r_k ||_A^(-1)
over all choices of the scalar nu.
(2) The p_k's are called conjugate, or more precisely A-conjugate, because
they are orthogonal in the A inner product: p_k^T*A*p_j = 0 if j neq k.

CG is sometimes derived by figuring out recurrences for x_k, r_k and
p_k that satisfy properties (1) and (2), and then showing they satisfy
the optimality properties in the Lemma. A nice presentation is found
in Shewchuk's writeup on the class web page. Instead, we will
start with the formula for x_k = Q_k*T_k^(-1)*e_1*||b||_2 , from
the lemma, and derive the recurrences from there.

Since T_k = Q_k^T*A*Q_k is spd, we can do Cholesky to get
T_k = L'_k*L'_k^T where L'_k is lower bidiagonal, and
L'_k*L'_k^T = L_k*D_k*L_k^T where L_k has unit diagonal
and D_k is diagonal, with D_k(i,i) = L'_k(i,i)^2. Then
from the Lemma

$$
\begin{aligned}
x_k &= Q_k*T_k^{(-1)}*e_1*||b||_2 \\
&= Q_k*(L_k*D_k*L_k^T)^{(-1)}*e_1*||b||_2 \\
&= Q_k*L_k^{(-T)} * (D_k^{(-1)}*L_k^{(-1)}*e_1*||b||_2) \\
&= P'_k * y_k
\end{aligned}
$$

where we write P'_k = [p'_1,...,p'_k]. The eventual conjugate gradients
p_k will turn out to be scalar multiples of the p'_k. So we
know enough to prove property (2) above:

Lemma: The p'_k's are A-conjugate, i.e. P'_k^T * A * P'_k is diagonal.
Proof:

$$
\begin{aligned}
P'_k{}^T * A * P'_k &= (Q_k*L_k^{(-T)})^T * A * (Q_k*L_k^{(-T)}) \\
&= L_k^{(-1)}* Q_k^T*A*Q_k * L_k^{(-T)} \\
&= L_k^{(-1)}* T_k * L_k^{(-T)} \\
&= L_k^{(-1)}* (L_k * D_k * L_k^T) * L_k^{(-T)} \\
&= D_k
\end{aligned}
$$

Now we derive simple recurrences for the column p'_k of P'_k, and the
components of y_k, which in turn give us a recurrence for x_k.
We will show that P'_(k-1) is identical to the leading k-1 columns of P'_k:

$$P'_k = [P'_{(k-1)}, p'_k]$$

and similarly for y_(k-1) = [s_1; ... ;s_(k-1)] and y_k =
[s_1;...;s_(k-1);s_k].
Assuming these are true for a moment, they will give us the recurrence for
x_k:
(Rx)

$$
\begin{aligned}
x_k = P'_k*y_k &= [P'_{(k-1)}, p'_k]*[s_1;...;s_k] \\
&= P'_{(k-1)}*[s_1;...;s_{(k-1)}] + p'_k*s_k \\
&= x_{(k-1)} + p'_k*s_k
\end{aligned}
$$

assuming we can also get recurrences for p'_k and s_k.

Since Lanczos constructs T_k row by row so T_(k-1) is the
leading k-1 by k-1 submatrix of T_k, and Cholesky also
computes row by row, we get that L_(k-1) and D_(k-1)
are the leading k-1 by k-1 submatrices of L_k and D_k, resp:

$$
\begin{aligned}
T_k &= L_k * D_k * L_k^T \\
&= \begin{bmatrix} L_{(k-1)} & 0 \\ 0 \ldots 0\ l_{(k-1)} & 1 \end{bmatrix} * \begin{bmatrix} D_{(k-1)} & 0 \\ 0 & d_k \end{bmatrix} * \begin{bmatrix} L_{(k-1)} & 0 \\ 0 \ldots 0\ l_{(k-1)} & 1 \end{bmatrix}^T
\end{aligned}
$$

$$
L_k^{(-1)} = \begin{bmatrix} L_{(k-1)}^{(-1)} & 0 \\ stuff & 1 \end{bmatrix}
$$

$$
\begin{aligned}
y_k &= D_k^{(-1)}*L_k^{(-1)}*e_1*||b||_2 \\
&= \begin{bmatrix} D_{(k-1)}^{(-1)} & 0 \\ 0 & d_k^{(-1)} \end{bmatrix} * \begin{bmatrix} L_{(k-1)}^{(-1)} & 0 \\ stuff & 1 \end{bmatrix} * e_1 * ||b||_2 \\
&= \begin{bmatrix} D_{(k-1)}^{(-1)} * L_{(k-1)}^{(-1)} * e_1 * ||b||_2 \\ s_k \end{bmatrix} \\
&= \begin{bmatrix} y_{(k-1)} \end{bmatrix}
\end{aligned}
$$

```
          [ s_k     ]
as the desired recurrence for y_k. Similarly
   P'_k = Q_k * L_k^(-T)
        = [ Q_(k-1) , q_k ] * [ L_(k-1)^(-T) stuff ]
                              [     0          1   ]
        = [ Q_(k-1)*L_(k-1)^(-T) , p'_k ]
        = [ P'_(k-1) , p'_k ]
so to get a formula for p'_k, write
   Q_k = P'_k * L_k^T,
and equating the last columns we get
   q_k = p'_k + p'_(k-1)*L_k(k,k-1)
or
(Rp)   p'_k = q_k - l_(k-1)*p'_(k-1)
as the desired recurrence for p'_k.

Finally we need a recurrence for r_k from (Rx):
(Rr)   r_k = b - A*x_k
           = b - A*(x_(k-1) + p'_k*s_k)
           = r_(k-1) - A*p'_k*s_k
Putting these vector recurrences together we get

(Rr)  r_k  = r_(k-1) - A*p'_k*s_k
(Rx)  x_k  = x_(k-1) + p'_k*s_k
(Rp)  p'_k = q_k - l_(k-1)*p'_(k-1)

To get closer to the final recurrences, substitute
q_k = r_(k-1)/||r_(k-1)||_2 and p_k = ||r_(k-1)||_2 * p'_k to get

(Rr')  r_k  = r_(k-1) - A*p_k*(s_k/||r_(k-1)||_2)
            = r_(k-1) - A*p_k*(nu_k)
(Rx')  x_k  = x_(k-1) + p_k*nu_k
(Rp')  p_k = r_(k-1) - (||r_(k-1)||_2 * l_(k-1) / ||r_(k-2)||_2) * p_(k-1)
           = r_(k-1) + mu_k * p_(k-1)

We still need formulas for the scalars mu_k and nu_k. There are
several choices, some more numerically stable than others.
See the text for the algebra, we just write here

    nu_k = r_(k-1)^T*r_(k-1) / p_k^T*A*p_k
    mu_k = r_(k-1)^T*r_(k-1) / r_(k-2)^T*r_(k-2)

Putting it all together, we get

Conjugate Gradient Method for solving Ax=b:
   k= 0 , x_0 = 0, r_0 = b, p_1 = b
   repeat
     k = k+1
     z = A*p_k
     nu_k = r_(k-1)^T*r_(k-1) / p_k^T*z
     x_k  = x_(k-1) + nu_k*p_k
     r_k  = r_(k-1) - nu_k*z
     mu_(k+1) = r_k^T*r_k / r_(k-1)^T*r_(k-1)
     p_(k+1)  = r_k + mu_(k+1) * p_k
   until || r_k ||_2 small enough

Note that minimization property (1) above follows from the fact that
since x_k minimizes || r_k ||_A^(-1) over all possible x_k in Q_k,
it must certainly also satisfy the lower dimensional minimization
property (1).
```

As with most other iterative methods (besides multigrid), the convergence rate of CG depends on the condition number cond(A):

Thm: || r_k ||_{inv(A)} / || r_0 ||_{inv(A)}}
     <= 2/(1 + 2*k/(sqrt(cond(A) - 1)))

So when cond(A) is large, one needs to take O(sqrt(cond(A))) steps to converge.

For d-dimensional Poisson's equation on a grid with n mesh points on a side, cond(A) is about n^2, so CG take O(n) steps to converge, for any dimension d.