Notes for Ma221 Lecture 12, Nov 12, 2024

Splitting Methods

Goal: given an initial guess x_0 for a solution
of A*x=b, cheaply compute a sequence x_i converging to A^(-1)*b.

Def: A splitting of A is a decomposition A = M - K with M nonsingular

This yields the following iterative method: Write A*x = M*x - K*x = b,
so M*x = K*x + b. Then compute x_(i+1) from x_i by solving M*x_{i+1} =
K*x_i + b,
or x_{i+1} = M^(-1)*K*x_i + M^(-1)*b = R*x_i + c.
  (*) x_{i+1} = R*x_i + c
For this to work well, we need two conditions:
(1) It should converge.
(2) Multiplying by R (i.e. solving M*x_(i+1) = K*x_i+c for x_(i+1))
and
    computing c = M^{-1}*b should be much cheaper than solving with A
itself.

Lemma: Let ||.|| be any operator norm. Then if ||R||<1, (*) converges
to A^(-1)*b for any x_0.
Proof: Subtract x = R*x + c from x_{i+1} = R*x_i+c to get
x_(i+1) - x = R*(x_i-x) or
  ||x_(i+1)-x|| <= ||R||*||x_i-x|| <= ... <= ||R||^(i+1)*||x_0-x||
which converges to zero for any x_0 if ||R||<1.

Def: The spectral radius of R is rho(R) = max |lambda|, the largest
absolute
value of any eigenvalue of R

Lemma: For all operator norms, rho(R) <= ||R||. For all R and all
eps>0,
there exists an operator norm ||.||_* such that ||R||_* <= rho(R) +
eps

Proof: To show rho(R) <= ||R||, note that ||R|| = max_x ||R*x||/||x||
By picking x to be an eigenvector, we see that ||R|| >= |lambda| for
any
eigenvalue lambda of R. To construct ||.||_*, we use the Jordan Form
of R:
Let S^{-1}*R*S = J be in Jordan form, i.e. J is block diagonal with
Jordan
blocks. Let D = diag(1,eps,eps^2,...,eps^(n-1)). Then
J_eps = D^{-1}*J*D leaves the diagonal (eigenvalues) unchanged, and
multiplies
each superdiagonal entry of J by eps, so J has eigenvalues on the
diagonal,
and eps above the diagonal in any Jordan block. We now define a new

vector
norm by ||x||_* = ||(S*D)^{-1}*x||_inf. (See Question 1.5 for the
proof that
this defines a vector norm.) Then

```
  ||R||_* = max_x ||R*x||_* / ||x||_*
          = max_x ||(S*D)^(-1)*R*x||_inf / ||(S*D)^(-1)*x||_inf
          = max_y ||(S*D)^(-1)*R*(S*D)*y||_inf / ||y||_inf
                       where y = (S*D)^(-1)*x
          = max_y ||D^(-1)*S^(-1)*R*S*D*y||_inf / ||y||_inf
          = max_y ||J_eps*y||_inf / ||y||_inf
          <= max |lambda| + eps
          = rho(R) + eps
```

Theorem: The iteration x_{i+1} = R*x_i + c converges to the solution
of A*x=b
for all starting vectors x_0 if and only if rho(R)<1.

Proof: If rho(R) >= 1, chose x_0 so that x_0-x is an eigenvector of R
for
its largest eigenvalue, call it lambda. Then x_i-x = R^i*(x_0-x) =
lambda^i*(x_0-x)
does not converge to zero since |lambda| >= 1.
If rho(R) < 1, use the last lemma to conclude that there is an
operator norm ||R||_*
and an eps such that ||R||_* <= rho(R)+eps < 1, so that
x_i-x = R^i*(x_0-x) converges to zero for all x_0.

Obviously, the smaller is rho(R), the faster is convergence. Our goals
is to pick the
splitting A = M-K so that multiplying by R = inv(M)*K is easy, and
rho(R) is small. Choosing M = I and K = I-A makes multiplying by R
easy, but
will not generally make rho(R) small. Choosing K = 0 and so R = 0
makes convergence
immediate, but requires having the solution c = A^{-1}*b.

Now we can describe Jacobi, Gauss-Seidel (GS) and Successive
Overrelaxation (SOR).
All share the notation A = D - L' - U' = D*(I - L - U), where D is the
diagonal of A,
L' is the negative of the strictly lower triangular part, and U' is
the negative of
the strictly upper triangular part of A.

```
Jacobi:
   In words, for j = 1 to n, pick x_(i+1)(j) to exactly satisfy
equation j
   As a loop:
       for j = 1 to n, x_(i+1)(j) = (b_j - sum_{k neq j}
A(j,k)*x_i(k))/A(j,j)
```

```
   As a splitting: A = D - (L'+U') = M - K, so
       R_J = M^{-1}*K = D^{-1}*(L'+U') = L+U
   For the 2D Poisson equation T_N*V+V*T_N = h^2*F, to get from V_i to
V_(i+1):
       for j = 1 to n, for k = 1 to n
              V_(i+1)(j,k) = (V_i(j-1,k) + V_i(j+1,k) + V_i(j,k-1) +
V_i(j,k+1)
                             + h^2*F(j,k))/4
                           = "average" of 4 nearest neighbors and
right-hand-side

Gauss-Seidel:
    In words, improve on Jacobi by using most recently updated values
of solution
    As a loop:
       for j = 1 to n, x_(i+1)(j) = (b_j - sum_{k < j} A(j,k)*x_(i+1)
(k)
                                                   ... updated x
components
                                         - sum_{k > j}
A(j,k)*x_i(k) ) / A(j,j)
                                                   ... older x
components
    As a splitting: A = (D-L') - U', so
      R_GS = (D-L')^{-1}*U' = (D*(I-D^{-1}*L'))^{-1}*U' = (I-L)^{-1}*U
    Note that the order in which we update entries matters in GS
(unlike Jacobi).
    For 2D Poisson there are two orders to consider:
      natural order (rowwise or columnwise update of V(i,j))
      Red-Black (RB) ordering: color the entries in the 2D mesh like a
checkerboard,
        and number the Reds before all the Blacks. Note that each Red
node only has
        Black neighbors, and vice-versa. Thus when updating Red nodes,
we can
        update them in any order, since all the Black neighbors have
old data.
        Then when we update the Black nodes, we can update them in any
order,
        because all the Red neighbors have new data.
     For all (j,k) nodes that are Red (j+k even)
              V_(i+1)(j,k) = (V_i(j-1,k) + V_i(j+1,k) + V_i(j,k-1) +
V_i(j,k+1)
                             + h^2*F(j,k))/4 ... only use old data
     For all (j,k) nodes that are Black (j+k odd)
              V_(i+1)(j,k) = (V_(i+1)(j-1,k) + V_(i+1)(j+1,k) +
V_(i+1)(j,k-1)
                              + V_(i+1)(j,k+1)
                              + h^2*F(j,k))/4 ... only use new data
```

```
SOR: In words: This depends on a parameter w: We let the result of SOR
be
       a weighted combination of the old (x_i) and new (x_(i+1))
solutions
       that GS would have computed:
          x^SOR(w)_(i+1)(j) = (1-w)*x_i(j) + w*x^GS_(i+1)(j)
       When w=1, SOR(1) is just GS. When w<1 we would call this
underrelaxation,
       and when w>1, we call it overrelaxation. We prefer to
"overrelax" with the
       intuition that if moving in the direction from x_i to x_(i+1)
was a good idea,
       moving even farther in the same direction is better.
       Later we will show how to pick w optimally for the model
problem.
     As a loop:
       for j = 1 to n, x_(i+1)(j) = (1-w)*x_i(j) +
                                    w*(b_j - sum_{k < j}
A(j,k)*x_(i+1)(k)
                                                   ... updated x
components
                                           - sum_{k > j}
A(j,k)*x_i(k) ) / A(j,j)
                                                   ... older x
components
     As a splitting multiply through by D to get
         (D-w*L')*x_(i+1) = ((1-w)*D + w*U')*x_i + w*b
     and then divide by w to get the splitting
         A = (D/w - L') - (D/w-D + U')
     or
         R_SOR(w) = (D/w - L')^(-1) * (D/w - D + U') = (I-
w*L)^(-1)*((1-w)*I + w*U)
     For 2D Poisson with Red-Black Ordering:
        For all (j,k) nodes that are Red
               V_(i+1)(j,k) =  (1-w)*V_i(j,k) + w*
                              (V_i(j-1,k) + V_i(j+1,k) + V_i(j,k-1) +
V_i(j,k+1)
                              + h^2*F(j,k))/4 ... only use old data
        For all (j,k) nodes that are Black
               V_(i+1)(j,k) = (1-w)*V_i(j,k) + w*
                              (V_(i+1)(j-1,k) + V_(i+1)(j+1,k) +
V_(i+1)(j,k-1)
                              + V_(i+1)(j,k+1)
                              + h^2*F(j,k))/4 ... use new data

Now we analyze the convergence of these basic methods, in general and
for the
Model Problem (2D Poisson) in particular.

Jacobi's method for the Model problem is easy to analyze, since in the
```

```
splitting
T_nxn = M - K = 4*I - (4*I - T_nxn), so R = M^(-1)*K = I - T_nxn/4,
so the eigenvalues of R are 1 - (lambda_i+lambda_j)/4 for all pairs of
eigenvalues lambda_i = 2*(1-cos(i*pi/(n+1))) and lambda_j of T_n,
and so the spectral radius rho(R) of R is
  1 - lambda_min/2 = 1 - (1 - cos(pi/(n+1))) = cos(pi/(n+1)) ~ 1 -
pi^2/2(n+1)^2
which gets closer to 1 as n grows, i.e. Jacobi converges more slowly.

Since the error after m steps is multiplied by rho(R)^m, we estimate
the speed of convergence by computing how many steps m are needed
to reduce the error by a constant factor. Setting rho(R) = 1 - x
and solving (1-x)^m = exp(-1) for simplicity, we get
   (1 - x)^m = (1-x)^[(1/x)*m*x] ~ exp(-m*x) = exp(-1)
or m = 1/x = 2*(n+1)^2/pi^2 = O(n^2) = O(N) steps, where N=n^2 is the
number of unknowns. Note that cond(T_nxn)~4*N/pi^2 is also O(N).
So the number of iterations grows proportionally to the dimension N,
and to the condition number. So to reduce the error by any constant
factor costs O(N) iterations * O(N) flops_per_iteration, or O(N^2)
overall, explaining the entry in the table above. This is typical
for many iterative methods: the number of iterations grows with the
condition number (multigrid is an important exception!).

Provided the variables are updated in the appropriate order for 2D
Poisson, we get
rho(R_GS) = rho(R_J)^2, i.e. one step of Gauss-Seidel reduces the
error as much as
two Jacobi steps; this is better but the overall complexity O(N^2) is
the same.

Then, again with the right update order, and the right choice of
overrelaxation parameter w, SOR(w) for 2D Poisson is much faster with
     rho(R_SOR(w_opt)) ~ 1 - 2*pi/n
for large n, which is close to 1 but much farther away than rho(R_J),
and takes only O(n) = O(sqrt(N)) steps to reduce the error by a
constant
factor, for an overall cost of O(N^(3/2)) as opposed to O(N^2).

Now we present (and prove some of!) the general theory of these
three methods applied to Ax=b.

Thm 1: If A is strictly row diagonally dominant:
          |A_ii| > sum_{j neq i} |A_ij|
       then both Jacobi and GS converge, and GS is at least as fast
       as Jacobi in the sense that ||R_GS||_inf <= ||R_J||_inf < 1
Proof: (just for Jacobi; see Thm 6.2 in text for full proof):
       Split A = D - (D - A) so R_J = D^(-1)*(D-A) = I -D^(-1)*A
       So sum_i |R_J(j,i)|
            = | 1 - A(j,j)/A(j,j)| +  sum_{i neq j} |A(j,i)|/|A(j,j)|
            = ( sum_{i neq j} |A(j,i)| ) / |A(j,j)|
```

< 1 by strict row diagonal dominance
so ||R_J||_inf < 1 and Jacobi converges

The model problem is not strictly row diagonally dominant,
since the diagonal equals the negative sum of the off diagonals
in most rows. We call a matrix where
|A_ii| >= sum_{j neq i} |A_ij|
in all rows, with strict inequality at least once, weakly row
diagonally
dominant. This alone is not enough for convergence: consider
A = [1 -1 0; 1 1 0; 0 0 1] and
R = [0 1 0; -1 0 0; 0 0 0], whose powers do not converge.
So we need one more condition:

Def: A matrix is A is irreducible if there is no permutation P
such that P*A*P^T is block triangular. Equivalently, the (directed)
graph corresponding to entries of A is "strongly connected",
i.e. there is a path from every vertex to every other vertex.

The model problem satisfies this definition, since a mesh is
strongly connected. And for the vertices next to a boundary of
the mesh, |A_ii| > sum_{j neq i} |A_ij|, yielding

Thm 2: If A is weakly row diagonally dominant and irreducible (like
the model problem) then rho(R_GS) < rho(R_J) < 1, so both
Jacobi and GS converge, and GS is faster. (Thm 6.3 in text; see
reference [249] in text for proof)

Now we come to SOR(w):

Thm 3: If A is spd, then SOR(w) converges if and only if 0 < w < 2.
In particular SOR(1) = GS converges.
(see Thm 6.5 in text for proof)

The previous results describe the most general situations in which
we can prove convergence of splitting methods.
To analyze the convergence of SOR(w) on the model problem we need
to use another graph theoretic property of the matrix:

Def: A matrix has Property A if there is a permutation P such that
P*A*P^T = [A11 A12 ; A21 A22] has the property that A11 and A22
are diagonal. In graph theoretic terms, the vertices V of the graph
can be broken into two disjoint sets V = V1 U V2, where there are
no edges between pairs of vertices in V1, or between pairs of
vertices in V2. Such a graph is called bipartite.

Ex: For a model problem on a 1D mesh, let the odd numbered nodes
be V1 and the even numbered be V2; since there are only edges
connecting even and odd nodes, the matrix has property A.

Ex: For a model problem on a 2D mesh, think of the vertices
forming a checkerboard so they are either Red or Black.
Then the edges only connect Red to Black, so the matrix has
Property A. Another way to say this is to put vertices
v_ij with i+j odd in V1 and i+j even in V2. This works
for the 3D model problem and in higher dimensions.

Thm 4: Suppose matrix A has Property A, and we do SOR(w)
updating all the vertices in V1 before all the vertices in V2.
Then the eigenvalues mu of R_J and lambda of R_SOR(w) are
related by
(*)   (lambda+w-1)^2 = lambda*w^2*mu^2
In particular, if w=1, then lambda=mu^2, so
rho(R_SOR(1)) = rho(R_GS) = rho(R_J)^2, and GS converges
twice as fast as Jacobi. (This appears as Thm 6.6 in the text.)

Proof: Assuming we number all vertices in V1 before the vertices in
V2, matrix A
will look like A = [A11 , A12 ; A21 , A22] where A11 and A22 are
diagonal, since
property A tells us there are no edges connected vertices in V1 to V1
or V2 to V2.
For Poisson this means T = 4*I + [0, 0 ; A21, 0] + [0, A12 ; 0, 0]
= D - L' - U' .
Now since lambda is an eigenvalue of R_SOR(w) we get

```
 0 = det( lambda*I - R_SOR(w) )
   = det( lambda*I - (I-w*L)^(-1)*((1-w)*I + w*U) )
   = det((I-w*L)*( " ) )
   = det( lambda*I - lambda*w*L - (1-w)*I - w*U )
   = det( (lambda-1+w)*I - lambda*w*L - w*U )
   = det( \sqrt(lambda)*w
          * [(lambda-1+w)/(sqrt(lambda)*w)*I - sqrt(lambda)*L - U/
sqrt(lambda) ] )
   = (\sqrt(lambda)*w)^n *
          det( (lambda-1+w)/(sqrt(lambda)*w)*I - sqrt(lambda)*L - U/
sqrt(lambda) )
```

Because of the sparsity of L and U, we can let D = [I, 0; 0, I/
sqrt(lambda)]
and note that D*(sqrt(lambda)*L)*inv(D) = L, and D*(U/
sqrt(lambda))*inv(D) = U.
so we can premultiply the matrix inside det( ) by D and postmultiply
by inv(D),
without changing the determinant, to get

```
 0 = det( (lambda-1+w)/(sqrt(lambda)*w)*I - L - U)
   = det( (lambda-1+w)/(sqrt(lambda)*w)*I - R_J)
```

i.e. (lambda-1+w)/(sqrt(lambda)*w) is an eigenvalue mu of the Jacobi
matrix R_J.

More generally, since we know all eigenvalues mu of R_J for the model
problem, we

can use (*) to figure out all eigenvalues lambda of R_SOR(w), then
then pick w_opt
to minimize rho(R_SOR(w_opt)). This yields

Thm 5: Suppose A has property A, we do SOR(w) updating vertices in V1
before V2 as
before, all eigenvalues of R_J are real, and mu= rho(R_J)<1 (as in the
model problem).
Then
w_opt = 2/(1+sqrt(1-mu^2))
rho(R_SOR(w_opt)) = w_opt - 1 = mu^2 / (1+sqrt(1-mu^2))^2
In particular, for the 2D model problem on an n x n grid
w_opt = 2/(1+sin(pi/(n+1))) ~ 2
rho(R_SOR(w_opt)) = cos^2(pi/(n+1))/(1+sin(pi/(n+1)))^2 ~ 1 -
2*pi/(n+1)
The proof follows by solving (*) for lambda (see Thm 6.7 in the text).