Notes for 2016-11-09

1 General relaxations

So far, we have mostly discussed stationary methods in we think of sweeping through all the variables in some fixed order and updating a variable or block of variables at a time. There is nothing that say that the order must be *fixed*, though, if we are willing to forgo the analytical framework of splittings. There are essentially two reasons that we might think to do this:

1. We decide which variable(s) to update next based on some adaptive policy, such as which equations have the largest residual. This leads to the *Gauss-Southwell* method. Various methods for fast (sublinear time) personalized PageRank use this strategy.

2. We update variable(s) on multiple processors, communicating the changes opportunistically. In this case, there may be no real rhyme or reason to the order in which we see updates. These methods are called *chaotic relaxation* or *asynchronous relaxation* approaches, and they have seen a great deal of renewed interest over the past several years for both classical scientific computing problems (e.g. PDE solvers) and for machine learning applications.

2 Alternating Direction Implicit

The *alternating direction implicit* approach to the model problem began life as an operator-splitting approach to solving a time-domain diffusion problem. At each step of an ordinary implicit time stepper for the heat equation, one would solve a system of the form

$$(I + \Delta t T)x = b,$$

where Δt is small and T is the 2D Laplacian operator. But note that if $T = T_x + T_y$, then

$$(I + \Delta t/2T_x)(I + \Delta t/2T_y) = (I + \Delta t T + O(\Delta t)^2);$$

hence, we commit only a small amount of error if instead of solving one system with T we solve two half-step systems involving T_x and T_y, respectively,

where T_x and T_y are the discretizations of the derivative operator in the x and y directions. This is known as the *alternating direction* method. The implementation is relatively straightforward:

```
% [U] = sweep_adi(U, F)
%
% Run one ADI sweep for 2D Poisson (single shift)
%
function [U] = sweep_adi(U, F);
  n = size(U,1)-2;
  I = 2:n+1;
  h2 = 1/(n+1)^2;
  dt = 1/(n+1);
  Ts = spdiags(ones(n,1)*[-1, 2+dt, -1], [-1, 0, 1], n, n);
  II = speye(n);

  % Iterate on Ts*U + U*Ts = h^2*F + 2*dt*U where Ts = T+dt*I:
  %  Ts*U = h^2*F + 2*dt*U - U*Ts
  %  U*Ts = h^2*F + 2*dt*U - Ts*U
  U(I,I) = Ts\( h2*F(I,I) - U(I,I)*Ts + 2*dt*U(I,I) );
  U(I,I) = ( h2*F(I,I) - Ts*U(I,I) + 2*dt*U(I,I) )/Ts;

end
```

In practice, cycling between several different versions of the shift parameter (interpreted above as a time-step) can lead to very rapid convergence of the ADI iteration. This beautiful classical result, which has deep connections to the Zolotarev problem from approximation theory, has taken on renewed usefulness in modern control theory and model reduction, where recent work has connected ADI-type methods for Sylvester equations to various rational Krylov methods.

The ADI method and its relations have also garnered many citations over the past 5–10 years because of their role as prior art for various optimization methods, such as the ADMM method.

3 Approximation from a subspace

Our workhorse methods for solving large-scale systems involve two key ideas: *relaxation* to produce a sequence of ever-better approximations to a problem, and *approximation from a subspace* assumed to contain a good estimate to the solution (e.g. the subspace spanned by iterates of some relaxation method). Having dealt with the former, we now deal with the latter.

Suppose we wish to estimate the solution to a linear system $Ax^{(*)} = b$ by an approximate solution $\hat{x} \in \mathcal{V}$, where $\mathcal{V}$ is some approximation subspace. How should we choose $\hat{x}$? There are three standard answers:

- *Least squares*: Minimize $\|A\hat{x} - b\|_M^2$ for some M.
- *Optimization*: If A is SPD, minimize $\phi(x) = \frac{1}{2}x^T Ax - x^T b$ over $\mathcal{V}$.
- *Galerkin*: Choose $A\hat{x} - b \perp \mathcal{W}$ for some test space $\mathcal{W}$. In *Bubnov-Galerkin*, $\mathcal{W} = \mathcal{V}$; otherwise we have a *Petrov-Galerkin* method.

These three methods are the standard approaches used in all the methods we will consider. Of course, they are not the only possibilities. For example, we might choose $\hat{x}$ to minimize the residual in some non-Euclidean norm, or we might more generally choose $\hat{x}$ by optimizing some non-quadratic loss function. But these approaches lead to optimization problems that cannot be immediately solved by linear algebra methods.

The three approaches are closely connected in many ways:

- Suppose $\hat{x}$ is the least squares solution. Then the normal equations give that $A\hat{x} - b \perp MA\mathcal{V}$; this is a (Petrov-)Galerkin condition.
- Similarly, suppose $\hat{x}$ minimizes $\phi(x)$ over the space $\mathcal{V}$. Then for any $\delta x \in \mathcal{V}$ we must have

  $$\delta\phi = \delta x^T (Ax - b) = 0,$$

  i.e. $Ax - b \perp \mathcal{V}$. This is a (Bubnov-)Galerkin condition.
- If x is the least squares solution, then by definition we minimize

  $$\frac{1}{2}\|Ax - b\|_M^2 = \frac{1}{2}x^T A^T MAx - x^T A^T Mb + \frac{1}{2}b^T Mb,$$

  i.e. we have the optimization objective for the normal equation SPD system $A^T MAx - A^T Mb = 0$, plus a constant.
- Note that if A is SPD, then we can express ϕ with respect to the A^{-1} norm as

  $$\phi(x) = \frac{1}{2}\|Ax - b\|_{A^{-1}}^2 - \frac{1}{2}b^T A^{-1} b,$$

  so choosing $\hat{x}$ by minimizing $\phi(x)$ is equivalent to minimizing the A^{-1} norm of the residual.

- Alternately, write $\phi(x)$ as

$$\phi(x) = \frac{1}{2}\|x - A^{-1}b\|_A^2 - \frac{1}{2}b^T A^{-1} b,$$

  and so choosing $\hat{x}$ by minimizing $\phi(x)$ is also equivalent to minimizing the A norm of the error.

When deriving methods, it is frequently convenient to turn to one or the other of these characterizations. But for computation and analysis, we will generally turn to the Galerkin formalism.

In order for any of these methods to produce accurate results, we need two properties to hold:

- *Consistency*: Does the space contain a good approximation to x?
- *Stability*: Will our scheme find something close to the best approximation possible from the space?

We leave the consistency and the choice of subspaces to later; for now, we deal with the problem of method stability.