Introduction to Iterative Methods

1 Iterative methods

We now look to solve linear systems of equations in a fundamentally different way than the direct methods which we are accustomed to. Take our canonical form of a linear system $A\vec{x} = \vec{b}$. We can rewrite this matrix-vector equation as

$$\vec{x} = (I - A)\vec{x} + b \tag{1}$$

Let us say that

$$x^{k+1} = (I - A)x^k + b$$

Start with a guess for x^0 and iterate to convergence. We define convergence when

$$||x^{k+1} - x^k|| < \epsilon \tag{2}$$

where $|| * ||$ is the norm of a vector. A few things stand out about this approach to solving our original system. First, we cannot say a priori how many operations will be required. Our direct solvers like Gaussian Elimination have a set, exact number of operations to get your answer. With iterative methods, you know how many operations per iteration are used but you have no way of knowing before runtime how many iterations are required.

Under what conditions will this converge? To figure this out, instead of solving for $\vec{x}$, let us solve for the error. Let $\tilde{x}$ be the true solution. Define the error at iteration k+1 as

$$e^{k+1} = \tilde{x} - x^{k+1}$$

Using our equation for iteration above for the true solution and the approximation, we arrive at

$$e^{k+1} = (I - A)e^k \tag{3}$$

We want the error term to be smaller after each iteration so our approximation approaches the true solution. The spectral radius of the matrix $(I - A)$ tells us whether this technique will converge. We know how to find the spectral radius of an arbitrary matrix; find the eigenvalues.

$$\det((I - A) - \lambda I) = 0$$

$$\det(-A + I(1 - \lambda)) = 0$$

$$\det(A + \eta I) = 0$$

where $\eta = (1 - \lambda)$. The ηs are just eigenvalues of A. Take, for example, the matrix A we had for the BECS (Backwards Euler) implicit discretization of the 1D heat equation.

$$A = \begin{bmatrix} 2c+1 & -c & 0 & \dots & & \\ -c & 2c+1 & -c & 0 & \dots & \\ 0 & -c & 2c+1 & -c & 0 & \dots \\ \vdots & \dots & \ddots & & & \end{bmatrix} \tag{4}$$

A check in MATLAB of the eigenvalues of $(I - A)$ for any value of c demonstrates that the spectral radius is greater than 1. So this will not converge!

This problem means that our new iterative method will not work in many cases. We should know before hand if our error term is diminished with each iteration. Even if the spectral radius is less than one, but is close to one, our approximation will converge too slowly to be useful. Instead, then, we try a small modification using a preconditioning matrix P.

$$\begin{aligned} P\tilde{x} &= (P - A)\tilde{x} + b \\ Px^{k+1} &= (P - A)x^k + b \\ x^{k+1} &= (I - P^{-1}A)x^k + P^{-1}b \text{ (solution)} \\ e^{k+1} &= (I - P^{-1}A)e^k \text{ (error)} \end{aligned} \tag{5}$$

Now, how our error term changes from iteration to iteration depends on the spectral radius of $(I - P^{-1}A)$. A good preconditioner is essential for good performance.

2 Some simple preconditioners

1. **Jacobi iteration**

   $P = \text{diag}(A) = D$

Our iterative equation becomes

$$x^{k+1} = (I - D^{-1}A)x^k + D^{-1}b$$

Let us consider the case of BECS discretization of the 1D heat equation (yet again). What is $(I - D^{-1}A)$? Multiplying by D^{-1} normalizes the diagonal by dividing all elements by $2c + 1$.

$$(I - D^{-1}A) = \begin{bmatrix} 0 & \frac{c}{2c+1} & 0 & \dots & & \\ \frac{c}{2c+1} & 0 & \frac{c}{2c+1} & 0 & \dots & \\ 0 & \frac{c}{2c+1} & 0 & \frac{c}{2c+1} & 0 & \dots \\ \vdots & \dots & \ddots & & & \end{bmatrix} \tag{6}$$

See the MATLAB code below which solves the BECS discretization with Jacboi iteration.

Now we want to write the Jacobi iteration as a scalar algorithm for our heat equation.

$$x^{k+1} = \begin{bmatrix} 0 & \frac{c}{2c+1} & 0 & \dots & & \\ \frac{c}{2c+1} & 0 & \frac{c}{2c+1} & 0 & \dots & \\ 0 & \frac{c}{2c+1} & 0 & \frac{c}{2c+1} & 0 & \dots \\ \vdots & \dots & \ddots & & & \end{bmatrix} \begin{bmatrix} x_1^k \\ x_2^k \\ \vdots \\ x_n^k \end{bmatrix} + \begin{bmatrix} \frac{1}{2c+1} & 0 & 0 & \dots & & \\ 0 & \frac{1}{2c+1} & 0 & 0 & \dots & \\ 0 & 0 & \frac{1}{2c+1} & 0 & 0 & \dots \\ \vdots & \dots & \ddots & & & \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{7}$$

Looking at each element, we arrive at a scalar formula

$$(x_i^{new})^{k+1} = \frac{c}{2c+1}((x_{i-1}^{new})^k + (x_{i+1}^{new})^k) + \frac{1}{2c+1}x_i^{old}$$

In 2D this becomes:

$$(x_{i,j}^{new})^{k+1} = \frac{c}{4c+1}((x_{i-1,j}^{new})^k + (x_{i,j-1}^{new})^k + (x_{i+1}^{new})^k + x_{i,j+1}^{new})^k) + \frac{1}{4c+1}x_{i,j}^{old}$$

The transition to three dimensions is equally simple and is required for HW3. The Wikipedia article on `Jacobi method` derives the general scalar formula from the general matrix formula and is a good reference.

2. **Gauss-Seidel iteration**

   $P = \text{tril}(A) = L$

```
hold off;
dt = 2.e-5
alpha = 1.
n=1000;
dx = 4/n;
C = alpha*dt/dx^2;

c1=-C*ones(n,1);
c2=(2*C+1*ones(n,1);

A=spdiags([c1 c2 c1],-1:1,n,n);

%since so easy to invert D, do it explicitly once
D= diag(diag(A));

invD = diag(1./diag(D));

B = eye(n) - invD*A

z=linspace(-2,2,n);
xold = exp(-2*z.^2);

% max number of iterations within a timestep before quitting
MAX_ITER = 1000;
%time steps
nsteps =100;

for i=1:nsteps
  %first guess can be anything but old solution is a good choice
  x=xold;
  niter = 0;
  for j=1:MAX_ITER
    xnew = B*x + invD*xold;
    if (mean(abs(x-new)) < 1.e-6)
      break;
    end
    x=xnew;
  end
  if (j==MAX_ITER)
    sprintf('%s\n', 'WARNING: iteration failed to converge')
  end
  sprintf('iterations: %d\n, error: %f\n', j, mean(x-xnew))
  if (i ==1)
    hold on;
  end
  plot(z,x);
  drawnow;
  % here is where you have finished a timestep and the old
      solution becomes the new RHS (b)
  xold = x;
end
```


Figure 1: `jacobi1dBE.m`– discretizes the one dimensional heat equation with Backwards Euler, Center Space and solves the linear system with Jacobi iteration

where tril() gives the lower triangular part of A. The convergence is twice as fast as Jacobi. Our iterative equation becomes

$$x^{k+1} = (I - L^{-1}A)x^k + L^{-1}b$$

This is the MATLAB-type approach for this iterative method. Noting that $(L - A)$ is upper triangular, it is usually easier to solve

$$Lx^{k+1} = (L - A)x^k + b$$

Once again, let's apply this to our BECS 1D case.

$$L - A = \begin{bmatrix} 0 & c & 0 & \dots & & \\ 0 & 0 & c & 0 & \dots & \\ 0 & 0 & 0 & c & 0 & \dots \\ \vdots & \dots & \ddots & & & \end{bmatrix} \tag{8}$$

To get a scalar formula, we follow a similar derivation as in Jacobi.

$$\begin{aligned} L(x_i^{new})^{k+1} &= c(x_i^{new})^k + b_i \\ &\vdots \\ (x_i^{new})^{k+1} &= \frac{c}{2c+1}((x_{i+1}^{new})^k + (x_{i-1}^{new})^{k+1}) + \frac{1}{2c+1}x_i^{old} \end{aligned} \tag{9}$$

The only difference between Gauss-Seidel and Jacobi is that the algorithm uses the x^{k+1} as you update it.

3. **Successive over-relaxation (SOR)**

$$P = D + \omega L$$

where $0 < \omega < 2$ for convergence.