Homework/Programming 6. (Due Nov. 12)

Math. 609-600

The goal of this homework set is to implement and do some numerical calculations involving the conjugate gradient and preconditioned conjugate gradient algorithms. To make meaningful examples, we start by considing the finite difference approximation to Laplace's equation with a homogenous Dirichlet boundary condition using a uniformly spaced grid on the rectangle $\Omega := (0,1)^2$. Specifically, we consider approximating the function u defined on Ω and satisfying

$$\begin{aligned} -\Delta u(x) &= f, \qquad \text{for } x \in \Omega, \\ u(x) &= 0, \qquad \text{for } x \in \partial\Omega. \end{aligned} \tag{0.1}$$

Here $\Delta := (\partial/\partial x_1)^2 + (\partial/\partial x_2)^2$, $\partial\Omega$ denotes the boundary of Ω and f is a given real valued function.

This problem was already discussed in the 639d class notes 8, Example 1. For an integer $N > 2$, we set $h = 1/(N+1)$ and start with a two dimensional equally spaced grid indexed by j, k in $\{0, 1, \ldots, N, N+1\}$ with nodes $x_{j,k} = h(j,k) \in \overline{\Omega} := \Omega \cup \partial\Omega$. The finite difference approximation is an array $U_{j,k} \approx u(x_{j,k})$ defined on the nodes, $x_{j,k}$, $j, k = 0, \ldots, N+1$. As the value of the solution is zero on boundary points, we set $U_{j,k} = 0$ when $x_{j,k} \in \partial\Omega$. We are left with N^2 interior points. We obtain N^2 equations by using the finite difference approximation to the Laplacian,

$$\begin{aligned} &-\Delta u(x_{j,k}) = \\ &\qquad \frac{4u(x_{j,k}) - u(x_{j+1,k}) - u(x_{j-1,k}) - u(x_{j,k+1}) - u(x_{j,k-1})}{h^2} \\ &\qquad\qquad + O(h^2) = f(x_{j,k}). \end{aligned} \tag{0.2}$$

As usual, we throw away the error term and replace the unknown solution by its approximation and obtain

$$\begin{aligned} &\frac{4U_{j,k} - U_{j+1,k} - U_{j-1,k} - U_{j,k+1} - U_{j,k-1}}{h^2} = f(x_{j,k}), \\ &\qquad \text{for } j, k = 1, \ldots, N. \end{aligned} \tag{0.3}$$

Thus, we left with an $N^2 \times N^2$ linear system

$$AU = F. \tag{0.4}$$

It was shown that this system is symmetric and positive definite (in Example 1, of Class notes 8). To actually write down this system as a matrix vector problem, one would have to choose some ordering of the unknowns. Instead, we will think of A as a linear operator on the set of $N \times N$ matrices so that U and F are both $N \times N$ matrices in (0.4). Note that an $N \times N$ matrix is naturally represented as a two dimensional array in MATLAB. The implementation of conjugate gradient on this problem requires the following ingredients: (in the following C, V, W are $N \times N$ matrices)

(a) Vector operations, e.g., for a, b scalars, the computation of $C = a * V + b * W$ is simply coded as the matlab command:

$$C = a * V + b * W;$$

(b) Inner product operations:

$$(V, W) := \sum_{j,k=1}^{N} V(j,k) * W(j,k). \tag{0.5}$$

(c) A routine which evaluates the action of the operator A on a matrix W, i.e, AW (referred to as the operator A applied to the matrix W).

The following matlab functions will be needed for this homework.

(a) a5ev.m
(b) lapinv.m
(c) sinfft.m
(d) sinfft2.m

If you prefer to use Python, you will have to convert these codes. The codes are also available on the class home page.

The first routine a5ev provides the operator evaluator. Given an $N \times N$ vector W and N, this function computes the $N \times N$ matrix

AW:

$$function\ [AW] = a5ev(W, N).$$

If you look carefully at a5ev, you will see that it never sets up a vector ordering of the two dimension matlab array W. Instead, it uses the ordering that matlab assigns to two dimensional arrays. The equations use the same ordering which implies that $AW(j,k)$ corresponds to the difference equation which has $4W_{j,k}$ appearing in it. Also, a5ev avoids conditional statements which are generally slower than arthmetic statements.

This stresses the point that the conjugate gradient algorithm does not need to know anything about structure of the matrix (assuming that it is symmetric and positive definite). Actually, in this case, because of matlab's vectorization operations, your cg algorithm will not even appear to know that it is dealing with matrices of unknowns.

You should write a matrix inner product routine

$$function\ [s] = IP(V, W, N)$$

which returns the sum in (0.5).

Problem 1. Write a conjugate gradient routine following the code given on P.2 of 639d lecture notes 11 calling a5ev and IP when needed. Note that **you must follow the algorithm in the notes and not simply copy one from the web as there are other (mathematically equivalent?) ones based on different formulas.** You should pass as input arguments, x0, b, aeval, and N where aeval is the name of the function which evaluates A (with interface ax=aeval(x,N)). Your code must reuse memory (you should use only three extra vectors x,p,r, following the pseudo code of P.2 of lecture 11). You are only allowed two A evaluations per step after startup, the extra one to compute the normalized error (see below). For $N = 4, 8, 16, 32, 128, 256$, iterate for the solution

$$Ax = 0$$

with initial iterate x_0 equal to the vector of all ones. At startup, compute $xn := \|x_0\|_A = (Ax_0, x_0)^{1/2}$ for normalization. At each iteration,

compute the normalized error

$$\epsilon_i = \frac{\|x_i\|_A}{xn}$$

and iterate until

$$\epsilon_j \leq .01 \tag{1.1}$$

For each N, report the first index j satisfying (1.1) and the normalized error at that j.

We now consider the variable coefficient problem approximated on P.6 of 639d lecture notes 8 (we leave the h^2 in the denominator on the left hand side of (8.12) of 639d Lecture Notes 8 although this is not critical).

Problem 2. Write a evaluation routine similar to a5ev but for the variable coefficient difference operator given in (8.12). You may code in the names of the coefficient routines for a_1 and a_2 into your evaluation routine. This routine will be used to evaluate the operator A in the preconditioned CG algorithm of Problem 3.

The next problem involves implementing the preconditioned conjugate gradient method (Algorithm 2 of 639d class notes 12). Let A_0 denote the matrix of Problem 1 above. You will use $B = A_0^{-1}$ as your preconditioner. As seen above, A_0 maps an $N \times N$ matlab array into another $N \times N$ matlab array.

In what follows, I will explain how $BV := A_0^{-1}V$ for a $N \times N$ array V can be efficiently computed and leads to the matlab code lapinv. It turns out that the eigenvectors of A_0 are of the form

$$\phi_{j,k}(l,m) = \sin(jl\pi/(N+1))\sin(kl\pi/(N+1)).$$

Note that for each $j,k \in 1,\ldots,N$, $\phi_{j,k}$ is an $N \times N$ matrix and its corresponding eigenvalue is given by:

$$\lambda(j,k) = h^{-2}(4 - 2\cos(j\pi/(N+1)) - 2\cos(k\pi/(N+1))).$$

For a $N \times N$ array W, we consider the two dimensional discrete sine transform,

$$SW(j,k) = \sum_{l,m=1}^{N} W(l,m) \sin(jl\pi/(N+1)) \sin(km\pi/(N+1)),$$
$$\text{for } j,k = 1,2,\ldots,N.$$

Clearly S is a linear operator on the space of $N \times N$ matrices and it turns out that

$$\frac{4}{(N+1)^2} S^2 = I \tag{2.1}$$

with I denoting the identity transformation (on $N \times N$ matrices).

Now, suppose that F is an $N \times N$ matrix and that C is such that

$$F(l,m) = \sum_{j,k=1}^{N} C(j,k)\phi_{j,k}(l,m) = SC(l,m), \qquad \text{for all } l,m = 1,\ldots,N.$$

By (2.1),

$$C = \frac{4}{(N+1)^2} SF.$$

Thus, since $\phi_{j,k}$ is an eigenvector of A_0,

$$\begin{aligned} A_0^{-1}F &= \sum_{j,k=1}^{N} C(j,k) A_0^{-1}\phi_{j,k} \\ &= \sum_{j,k=1}^{N} [(\lambda(j,k))^{-1} C(j,k)]\phi_{j,k} \\ &= \frac{4}{(N+1)^2} \sum_{j,k=1}^{N} [(\lambda(j,k))^{-1} (SF(j,k))]\phi_{j,k} \\ &= \frac{4}{(N+1)^2} S(\lambda^{-1} * (SF)) \end{aligned} \tag{2.2}$$

with $\lambda^{-1}(j,k) := 1/\lambda(j,k)$ and the above $*$ denoting the componentwise product two $N \times N$ arrays. This what is implemented in the code lapinv.m.

Note that the lapinv uses the routine sinfft2 to efficiently implement the two dimensional sine transform in terms on one dimensional sine transforms, each of which is subsequently reduced to two one dimensional FFTs of size $2N+2$ by doing an odd reflection. Thus, the overall computational cost of the evaluation of our preconditioner B is $O(N^2 \log(N))$. This just fails to be of optimal order by a logarithm.

Problem 3. Write a preconditioned conjugate gradient routine following Algorithm 2 of the 639 class notes 12. You should pass as input arguments, x0, b, aeval, beval, and N where aeval is the name of the function which you created in Problem 2 (with interface ax=aeval(x,N)) and beval is the name of the routine which evaluates the precondition (lapinv in this case). Your code must reuse memory (only four extra vectors x,p,r and tr. You are only allowed one beval evaluation and two aeval evaluations per step after startup, with the extra aeval evaluation used to compute the normalized error as in Problem 1. We start with $a_1(x_1,x_2) = 1/2 + x_1$ and $a_2(x_1,x_2) = 1/2 + x_2$. For $N = 4, 8, 16, 32, 128, 256$, iterate for the solution

$$Ax = 0$$

with initial iterate x_0 equal to the vector of all ones. Compute $xn := \|x_0\|_A = (Ax_0, x_0)^{1/2}$ for normalization. At each iteration, compute the normalized error

$$\epsilon_i = \frac{\|x_i\|_A}{xn}$$

and iterate until

$$\epsilon_j \leq .01 \tag{3.1}$$

For each N, report the first index j satisfying (3.1) and the normalized error at that j. Repeat these computations but for the 1/2s in a_1 and a_2 replaced by 1/10.