# CMDA 3606 · MATHEMATICAL MODELING II

## Problem Set 11

Posted 24 April 2019. Due at 12pm on Friday, 3 May 2019. Late work due 11:59pm on Friday, 3 May 2019.

Basic guidelines: Students may discuss the problems on this assignment, but each student must submit his or her individual writeup and code. (In particular, you *must write up your own individual MATLAB code.*) Students may consult class notes and other online resources for general information; cite all your sources and list those with whom you have discussed the problems.

1. [32 points: 8 points per part]

   This problem introduces another approach for solving large linear systems of equations, the *conjugate gradient method* (CG). CG is also a "Krylov subspace method" (as is GMRES) although CG is restricted to linear systems that have a coefficient matrix, $\mathbf{A}$, that is symmetric and positive definite. However, it is able to capitalize on the symmetry and positive definiteness of $\mathbf{A}$ and so can be much faster than GMRES for such problems.[†]

   Assume that $\mathbf{A}$ is symmetric positive definite and return to the objective function

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

   which we had introduced before in the first question of Problem Set 10.

   A set of (nonzero) direction vectors $\mathbf{p}_1, \ldots, \mathbf{p}_n$ are called *conjugate* (or $\mathbf{A}$-*orthogonal*) provided

   $$\mathbf{p}_j^T\mathbf{A}\mathbf{p}_k = 0, \qquad j \neq k.$$

   The conjugate gradient method builds a set of conjugate directions that are often much more effective than the steepest descent directions, because the conjugacy requirement prevents the method from cycling between a small number of directions and indeed, it guarantees convergence in a finite number of steps.

   (a) Suppose the nonzero vectors $\mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbb{R}^n$ are conjugate.
   **Show** that $\mathbf{p}_1, \ldots, \mathbf{p}_n$ are *linearly independent.*
   (Hint: Suppose a linear combination $\sum_{i=1}^n \alpha_i\mathbf{p}_i = 0$. By expanding $(\sum_{i=1}^n \alpha_i\mathbf{p}_i)^T\mathbf{A}(\sum_{j=1}^n \alpha_j\mathbf{p}_j) = 0$, show that it must be that $\alpha_1 = \alpha_2 = \cdots = \alpha_n = 0$.)

   (b) Suppose that at the $k$th step, we want to continue from the current iterate, $\mathbf{x}_k$, and search along a (fixed) direction $\mathbf{p}_k$ in order to minimize $\phi(\mathbf{x}_k + \alpha\mathbf{p}_k)$. If $\alpha_k$ is the value of $\alpha$ that accomplishes this, then we assign the value of the next iterate as $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$.
   **Show** that $\phi(\mathbf{x}_k + \alpha\mathbf{p}_k)$ is minimized among all $\alpha \in \mathbb{R}$ by

   $$\alpha_k = \frac{\mathbf{p}_k^T\mathbf{r}_k}{\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k}.$$

[†]Section 5.1 of the text by Nocedal and Wright (linked from the class website) will give you some helpful background information about the conjugate gradient algorithm. Nocedal and Wright define the residual as $\mathbf{r}_k = \mathbf{A}\mathbf{x}_k - \mathbf{b}$; we continue to use $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$, so some of our formulas differ.

(c) The conjugate gradient method begins by setting $\mathbf{p}_0 = \mathbf{r}_0$, and then for $k > 0$, one sets

$$\mathbf{p}_k = \mathbf{r}_k - \beta_k \mathbf{p}_{k-1},$$

where the scalar $\beta_k$ is chosen to ensure $\mathbf{p}_k$ and $\mathbf{p}_{k-1}$ are conjugate. **Show** this requirement implies

$$\beta_k = \frac{\mathbf{p}_{k-1}^T \mathbf{A} \mathbf{r}_k}{\mathbf{p}_{k-1}^T \mathbf{A} \mathbf{p}_{k-1}}.$$

(Remarkably, this is enough to ensure that $\mathbf{p}_k$ is conjugate to *all* the previous directions. You do not have to prove this; see Nocedal and Wright for details.)

(d) Refer back to Problem 1 on Problem Set 10. For the two pairs of $\mathbf{A}$ and $\mathbf{b}$ considered in parts (e) and (f) of that question, run the Conjugate Gradient iteration for 30 iterations. To the `semilogy` plots of $\|\mathbf{r}_k\|$ versus $k$ that you produced for these questions on Problem Set 10, add your Conjugate Gradient convergence curve. (You should see a significant improvement for the second problem.)

You may base your implementation of the Conjugate Gradient method on the following pseudocode (adapted from Nocedal and Wright, Algorithm 5.1).

$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{p}_0 = \mathbf{r}_0$
for $k = 0, 1, \ldots,$ maxit
    $\alpha_k = (\mathbf{p}_k^T \mathbf{r}_k)/(\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k)$
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
    $\mathbf{r}_{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{k+1}$
    $\beta_{k+1} = (\mathbf{p}_k^T \mathbf{A} \mathbf{r}_{k+1})/(\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k)$
    $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} - \beta_{k+1} \mathbf{p}_k$
end

2. [32 points: 4 points for parts (a) and (e); 8 points each for parts (b), (c), and (d).]

This problem gives you the opportunity to experiment with CG using another large matrix from Tim Davis's sparse matrix collection. This matrix has dimension $n = 999,999$ and comes from a "landscape ecology problem." The matrix is symmetric and positive definite, so CG is appropriate to solve $\mathbf{Ax} = \mathbf{b}$. For details, see:
`http://www.cise.ufl.edu/research/sparse/matrices/McRae/ecology2.html`

(a) Download the matrix from `ecology2.mat` from this link:
`http://www.cise.ufl.edu/research/sparse/mat/McRae/ecology2.mat`
Load this file into MATLAB (`load ecology2`).
Extract the matrix $\mathbf{A}$ (`A = Problem.A;`) and generate a random $\mathbf{b}$ vector (`b = randn(999999,1);`).
What percentage of the entries in $\mathbf{A}$ are nonzero? (`nnz(A)` counts the nonzero entries.)

(b) Solve this system using the conjugate gradient method, up to 2000 iterations:
`[x,flag,relres,iter,resvec] = pcg(A,b,1e-6,2000);`
Produce a `semilogy` plot showing $\|\mathbf{r}_k\|$ versus $k$, as you did on the last problem set.
Use the `tic` and `toc` commands to time how long the PCG algorithm takes to run.
*This command might take a couple minutes to run.*

To accelerate convergence of slowly-converging iterations, one often *preconditions* the problem. For symmetric positive definite $\mathbf{A}$ like this matrix, a popular approach is *incomplete Cholesky preconditioning*. This procedure computes a lower triangular matrix $\mathbf{L}$ such that $\mathbf{LL}^T \approx \mathbf{A}$ (suggesting that $\mathbf{L}^{-1}\mathbf{AL}^{-T} \approx \mathbf{I}$), and then replaces the original equation $\mathbf{Ax} = \mathbf{b}$ with the equivalent linear system

$$\left(\mathbf{L}^{-1}\mathbf{AL}^{-T}\right)\mathbf{y} = \mathbf{L}^{-1}\mathbf{b},$$

where $\mathbf{y} = \mathbf{L}^T\mathbf{x}$. Since $\mathbf{L}$ is lower triangular and $\mathbf{L}^{-T}$ is upper triangular, it is easy to multiply by $\mathbf{L}^{-1}\mathbf{AL}^{-T}$, as performed at each iteration of the conjugate gradient algorithm.

(c) Compute the incomplete Cholesky factorization with zero fill-in:
`L = ichol(A);`
Use `tic` and `toc` to record the time needed to run this command, and also write down the memory used to store `L` (use the `whos` command). Then run preconditioned conjugate gradients:
`[x,flag,relres,iter,resvec2] = pcg(A,b,1e-6,2000,L,L');`
Plot $\|\mathbf{r}_k\|$ versus $k$ on the same plot you generated for part (b).
Use `tic` and `toc` to record the run times for the `pcg` command.

(d) Repeat that last experiment, but now using incomplete Cholesky with threshold fill-in:
`opts.type = 'ict'; opts.droptol = 1e-3; L = ichol(A,opts);`
Record the time required by `ichol`, the memory to store `L`, and the time required by `pcg`.
Plot $\|\mathbf{r}_k\|$ versus $k$ on the same plot you generated for part (b) and (c).

Repeat the experiment using `opts.droptol = 1e-5` (if your computer will let you; otherwise explain why this command fails for you).

(e) Summarize (in words) the conclusions you draw from these experiments. Is preconditioning worth it for this problem? Should you invest time in building `L`? Is memory a consideration?

3. [12 points: 6 points per part]

Suppose that Newton's method is applied to the objective function

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

from Problem 1, where $\mathbf{A}$ is a symmetric positive-definite matrix.

In Problem Set 10, you computed the gradient: $\nabla\phi(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$.

(a) Compute the Hessian of $\phi$, $\nabla^2\phi(\mathbf{x})$.

(b) Consider the application of Newton's method on this problem:

$$\begin{aligned}\mathbf{p}_k &= -\big(\nabla^2\phi(\mathbf{x}_k)\big)^{-1}(\nabla\phi(\mathbf{x}_k))\\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{p}_k.\end{aligned}$$

Explain clearly how many iterations Newton's method will take to converge for this problem. Will the convergence depend on the initial condition $\mathbf{x}_0$?

4. [24 points: 8 points per part]

Consider the famous *Rosenbrock function*, $f : \mathbb{R}^2 \to \mathbb{R}$ given by

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

(a) Compute $\nabla f(\mathbf{x})$, the gradient of $f$, and show that the only critical point occurs at $\mathbf{x}_\star = (1, 1)^T$.

(b) Compute $\nabla^2 f(\mathbf{x})$, the Hessian of $f$, and show (MATLAB is okay) it is positive definite at $x_\star$, i.e., show that the Hessian at $\mathbf{x}_\star$ is symmetric and has positive eigenvalues.

(This implies that the critical point from part (a) is a local minimum; since it is the only critical point, it must also be a global minimum.)

(c) Implement Newton's method to optimize this function:

$$\begin{aligned}\mathbf{p}_k &= -\big(\nabla^2 f(\mathbf{x}_k)\big)^{-1}(\nabla f(\mathbf{x}_k))\\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{p}_k.\end{aligned}$$

*Be very careful when you implement your formulas for $\nabla f(\mathbf{x})$ and $\nabla^2 f(\mathbf{x})$: a small typo will throw off the method.*

Test your method by running it on two initial guesses:

$$\mathbf{x}_0 = \begin{bmatrix} -0.25 \\ 1.25 \end{bmatrix}, \qquad \mathbf{x}_0 = \begin{bmatrix} 5 \\ 5 \end{bmatrix}.$$

In each case, report the values of $\mathbf{x}_k$ for $k = 0, \ldots, 7$. Use `format long` to show many digits.

Be on the look out for the *quadratic convergence* of Newton's method: once convergence begins, the method *doubles the number of correct digits* at each iteration.