ORIE 6326 Homework 11
due 5/8/17

1. In this problem we will show that gradient descent with backtracking line search on an α-strongly convex, β-smooth function f converges, using an operator point of view.

   Recall the backtracking line search for deciding a step size in the direction Δx starting from the iterate x. Fix parameters $a \in (0, 1/2]$ and $b \in (0, 1)$. Starting with an initial stepsize $t = 1$, check the sufficient decrease condition

   $$f(x + t\Delta x) < f(x) + at\nabla f(x)^\top \Delta x$$

   and accept the stepsize if it is satisfied. Otherwise, update $t := bt$ and repeat until the stepsize t satisfies the sufficient decrease condition.

   Denote the k-th iterate as $x^{(k)}$, the k-th search direction $\Delta x^{(k)} = -\nabla f(x^{(k)})$, and the step size at the k-th iteration as $t^{(k)}$. Let us view gradient descent from an operator perspective. We write the iteration as

   $$x^{(k+1)} = F(x^{(k)}) = x^{(k)} - t_f(x^{(k)}, \nabla f(x^{(k)})),$$

   where we consider the step size as an operator t_f that computes its output $t^{(k)}\nabla f(x^{(k)})$ via backtracking linesearch.

   (a) Use the co + cocoercivity inequality (Lemma 3.11 from Bubeck)

   $$(\nabla f(x) - \nabla f(y))^T(x - y) \geq \frac{\alpha\beta}{\alpha + \beta}\|x - y\|^2 + \frac{1}{\alpha + \beta}\|\nabla f(x) - \nabla f(y)\|^2$$

   to show that the Lipshitz constant L for the gradient descent operator with fixed step size t satisfies

   $$L \leq \begin{cases} |1 - t\alpha| & t \leq \frac{2}{\alpha+\beta} \\ |1 - t\beta| & t \geq \frac{2}{\alpha+\beta} \end{cases}.$$

   This result strengthens the bound on the Lipschitz constant for the gradient descent operator that we proved in class.

   (b) Use the β-smoothness of f to show that the sufficient decrease condition above is satisfied when $0 \leq t \leq 1/\beta$. Conclude that the backtracking exits either when $t = 1$ or with a value of $t \geq b/\beta$.

   (c) Use the α-strong convexity of f to show that if the sufficient decrease condition above is satisfied, then $t \leq 2(1 - a)/\alpha$.

   (d) Assume $\kappa = \beta/\alpha \leq 2$ and $a = \frac{1}{2}$. Using the fact that $t^{(k)} \in (b/\beta, 2(1 - a)/\alpha)$, show that the gradient descent with linesearch operator F is a contraction. Use this to conclude that gradient descent with linesearch converges. (Gradient descent with linesearch converges even without a bound on the condition number; but a bound is required to ensure that the operator F is a contraction.)

2. Consider a system of equations $Ax = b$ where A is symmetric and positve semidefnite. The condition number of a system of linear equations is $\kappa(A) = \lambda_{\max}(A)/\lambda_{\min}(A)$. When the system is poorly conditioned $\kappa \gg 1$, iterative solvers like the conjugate gradient method can be very slow. Our goal is to create a new system of equations $\hat{A}\hat{x} = \hat{b}$ with two properties:

   - *Improved conditioning.* $\kappa(\hat{A}) < \kappa(A)$.
   - *Equivalent solutions.* It is easy to recover the solution x from the solution $\hat{x}$.

   One way to do so is to introduce a nonsingular matrix M, which we call the *preconditioner*, and to rewrite the system as

   $$M^{-1}Ax = M^{-1}b.$$

Then we see that x is a solution to $Ax = b$ if and only if it is a solution to $M^{-1}Ax = M^{-1}b$.

A second way to precondition is to introduce a nonsingular matrix D which is easy to invert. We re-write the system as

$$ADy = b.$$

Then we see that x is a solution to $Ax = b$ if and only if $y = D^{-1}x$ is a solution to $ADy = b$. We can solve the system of equations $ADy = b$ for y, and then invert D to find x.

Consider the Conjugate Gradient method for solving the system $Ax = b$:

Algorithm 1 Conjugate Gradient

```
1: Take $x_0 = 0$, $r_0 = b$, $p_0 = r_0$.
2: for $k = 1, 2, \dots$ do
3:     Compute a step length $\alpha_k = (r_{k-1}^\top r_{k-1})/(p_{k-1}^\top A p_{k-1})$;
4:     Update the approximate solution $x_k = x_{k-1} + \alpha_k p_{k-1}$;
5:     Update the residual $r_k = r_{k-1} - \alpha_k A p_{k-1}$;
6:     Compute a gradient correction factor $\beta_k = (r_k^\top r_k)/(r_{k-1}^\top r_{k-1})$;
7:     Set a new search direction $p_k = r_k + \beta_k p_{k-1}$;
8: End.
```

Here the residual at each iteration is $r_k = b - Ax_k$, and the search direction is p_k. To precondition this algorithm, we will take $M^{-1} = LL^\top$ for some non-singular matrix L. We will establish some properties of this preconditioned algorithm.

(a) Setting $\hat{A} = L^\top AL$, $\hat{x} = L^{-1}x$, and $b = L^\top b$, show that $\hat{A}\hat{x} = \hat{b}$ is equivalent to $Ax = b$. Further, the new residual at step k is $\hat{r}_k = \hat{b} - \hat{A}\hat{x}_k$. Show that $\hat{r}_k = L^\top r_k$.

(b) Now, define $\hat{p}_k = L^{-1}p_k$, $\tilde{r}_k = M^{-1}r_k$. The new step size is $\hat{\alpha}_k = (\hat{r}_{k-1}^\top \hat{r}_{k-1})/(\hat{p}_{k-1}^\top \hat{A}\hat{p}_{k-1})$. Show that $\hat{\alpha}_k = (r_{k-1}^\top \tilde{r}_{k-1})/(p_{k-1}^\top A p_{k-1})$.

(c) Similarly, the new gradient correction factor is $\hat{\beta}_k = (\hat{r}_k^\top \hat{r}_k)/(\hat{r}_{k-1}^\top \hat{r}_{k-1})$. Show that $\hat{\beta}_k = (r_k^\top \tilde{r}_k)/(r_{k-1}^\top \tilde{r}_{k-1})$.

(d) Lastly, show that the updates $\hat{x}_k = \hat{x}_{k-1} + \hat{\alpha}_k \hat{p}_{k-1}$, $\hat{r}_k = \hat{r}_{k-1} - \hat{\alpha}_k \hat{A}\hat{p}_{k-1}$, $\hat{p}_k = \hat{r}_k + \hat{\beta}_k \hat{p}_{k-1}$ are equivalent to $x_k = x_{k-1} + \hat{\alpha}_k p_{k-1}$, $r_k = r_{k-1} - \hat{\alpha}_k A p_{k-1}$, $p_k = \tilde{r}_k + \hat{\beta}_k p_{k-1}$, respectively.

(e) Based on the reasoning above, devise a new algorithm we will call the Preconditioned Conjugate Gradient, which should have something to do with the Conjugate Gradient algorithm.

(f) How would you suggest picking L? Why do you expect this choice to give improved performance?

3. This exercise explores the proximal gradient method for solving the optimization problem

$$\text{minimize} \quad f(x) + g(x),$$

with variable $x \in \mathbf{R}^n$, where $f : \mathbf{R}^n \to \mathbf{R}$ is a β-smooth convex function, and $g : \mathbf{R}^n \to \mathbf{R}$ is a closed proper convex function whose proximal operator is easy to compute. We will show the proximal gradient method converges using the monotone operator theory you learned in class, and you will implement it to solve a LASSO problem.

(a) Prove that z is a minimizer of $f + g$ if and only if z is a fixed point of the Forward-Backward Operator

$$(I + t\partial g)^{-1}(I - t\nabla f).$$

Here I is the identity map and $t > 0$ is a stepsize. Use the fact that a point z is a minimizer of $f(x) + g(x)$ if and only if $0 \in \nabla f(z) + \partial g(z)$.

(b) Show that for all $t < \frac{2}{\beta}$, the forward operator $(I - t\nabla f)$ is non-expansive.

(c) In lecture, we showed that if F is a monotone operator, then the resolvent $(I+\lambda F)^{-1}$ is averaged, for any $\lambda > 0$. (It is an average of identity and the Cayley operator). Conclude that $(I + t\partial g)^{-1}$ is averaged.

(d) Suppose we have two operators $F = (1-\theta_1)I + \theta_1 F_0$ and $G = (1-\theta_2)I + \theta_2 G_0$ where F_0, G_0 are nonexpansive operator and $\theta_1, \theta_2 \in (0,1)$. Show that the resulting composition $G \circ F = \theta I + (1-\theta)H$ for some nonexpansive operator H and $\theta \in (0,1)$. This shows that the composition of averaged operator is also averaged.

(e) Conclude that the forward-backward operator is an averaged operator. Therefore the (damped) fixed point iteration will converges to a fixed point whenever it exists.

(f) Let's implement the forward and backward splitting method for the optimization problem

$$\text{minimize} \quad \|Ax-b\|_2^2 + \lambda\|x\|_1 \tag{1}$$

with variable $x \in \mathbf{R}^n$. Here $A \in \mathbf{R}^{m\times n}$, $b \in \mathbf{R}^m$ and $x \in \mathbf{R}^n$ and $\lambda > 0$. Identify 1) the function f, 2) the operator that you will use for forward step, *i.e.*, $I - t\nabla f$, 3) the function g, and 4) the operator you will use for the backward step, *i.e.*, $(I + t\partial g)^{-1}$. In this context, the forward backward splitting scheme is also called the proximal gradient method. The proximal gradient operator is $F = (I + t\partial g)^{-1}(I - t\nabla f)$

(g) Now implement the proximal gradient method using the data in "A.csv", "b.csv" and setting $\lambda = \frac{10^3}{50}$. Plot the fixed point residual $\|F(x^{(k)}) - x^{(k)}|_2$, the objective value $f(x^{(k)}) + g(x^{(k)})$, the smooth part $f(x^{(k)})$, and the nonsmooth part $g(x^{(k)})$ as a function of the iteration counter k.

(h) The condition number κ of an α-strongly convex β-smooth function is $\kappa = \frac{\beta}{\alpha}$. What is the condition number of the function f? Give a formula and compute its value using the data above. Use this to derive a bound on the Lipschitz constant of the operator $F = (I + t\partial g)^{-1}(I - t\nabla f)$.

(i) Can you find a good preconditioner for the optimization problem (1)? In other words, find a matrix $D \in \mathbf{R}^{n\times n}$ that is easy to invert, and so that the forward and backward splitting method for the optimization problem

$$\underset{y\in\mathbf{R}^n}{\text{minimize}} \quad \|ADy-b\|_2^2 + \lambda\|Dy\|_1 \tag{2}$$

converges faster than (1) with $\lambda = \frac{10^3}{50}$ still. You may plot the objective value of (2) or the fixed point residual defined by the proximal gradient operator for (2).

4. Consider the linear program

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array}$$

with $c \in \mathbf{R}^n$, $A \in \mathbf{R}^{m\times n}$, $b \in \mathbf{R}^m$.

In this problem, we will consider the shortest known LP solver. For a history of short LP solvers and a few other examples, we refer the interested reader to Jacob Mattingly's collection, available at `https://jemnz.com/lp75.html`. The LP solver we consider is adapted from one developed by Borja Peleato.

(a) Show that the following (Julia) code implements ADMM for the LP above, starting from any initial iterates x and $z \in \mathbf{R}^n$.

```
for i=1:99;z=max(z-x,x-c);x=z+A\(b-A*z);end
```

(b) Implement this LP solver in your language of choice, and try it on a few randomly generated LPs. For convenience, here is some code you can use to generate a random LP and initialize the method.

```
m,n = 20,10;
A = randn(m,n);
b = A*rand(n);
c = randn(n);
x = zeros(n);
z = zeros(n);
```

(c) Extra credit: find an LP solver whose code is fewer than 44 characters.