Optimization, Programming Assignment #1

Due Monday October 28th

October 15, 2013

Description

In this assignment, you will experiment with gradient descent, conjugate gradient, BFGS and Newton's method. The included archive contains partial matlab code, which you must complete. Areas that you will fill in are marked with "TODO" comments.

You should turn in an archive containing all of your source code, and a document containing all plots, and answers to underlined questions.

Please remember to turn in a complete and readable document which contains your plots, and in which you answer the questions (in particular, talk about what your plots mean)! Your grade will be based far more on this document than on your source code.

This problem set draws heavily from the following text, although everything you need should also be present in your class notes:

- Jorge Nocedal and Stephen Wright. "Numerical Optimization". Springer, 1999.

1 Algorithms

The BFGS algorithm (algorithm 8.1 of Nocedal and Wright), and an "exact" line search (using the bisection method) are implemented in "algorithm_bfgs.m" and "line_search_bisection.m", respectively. You should understand the contents of these files (refer to your class notes on BFGS).

1.1 Backtracking line search

Complete the implementation of the backtracking line search in "line_search_backtracking.m". This is algorithm 9.2 of Boyd and Vandenberghe.

1.2 Gradient descent and Newton's method

Complete the implementations of gradient descent and Newton's method in "algorithm_gd.m" and "algorithm_newton.m". Gradient descent is algorithm 9.3 of Boyd and Vandenberghe, and Newton's method is algorithm 9.5. Your implementation of gradient descent should terminate once $\|\nabla f(x)\|_2^2 \leq \epsilon$, and your implementation of Newton's method should terminate once the squared Newton decrement $\lambda^2(x) = (\nabla f(x))^T \left(\nabla^2 f(x)\right)^{-1} (\nabla f(x))$ satisfies $\lambda^2(x) \leq \epsilon$.

1.3 Conjugate gradient

There are several different ways in which the linear conjugate gradient algorithm can be modified to work on a non-quadratic function, all of which are equivalent for a quadratic objective. Algorithm 1 contains pseudocode for the Fletcher-Reeves method. In your written assignment, you encountered the Polak-Ribière method, which differs from Fletcher-Reeves in the calculation of β, where $\beta_{PR}^{(k+1)} = \frac{\left(\nabla f\left(x^{(k+1)}\right)\right)^T\left(\nabla f\left(x^{(k+1)}\right)-\nabla f\left(x^{(k)}\right)\right)}{\left(\nabla f\left(x^{(k)}\right)\right)^T \nabla f\left(x^{(k)}\right)}$.

Pay special attention to lines 8 and 9. These lines "restart" the conjugate gradient algorithm every m steps (where m is the dimension of the problem), since by setting $\beta_{FR} = 0$, the search direction Δx will be set to the negative

Algorithm 1 Pseudocode for the Fletcher-Reeves conjugate gradient method (algorithm 5.4 of Nocedal and Wright)

```
ConjugateGradient$\left(x^{(0)}, \epsilon\right)$

    $m \leftarrow \dim\left(x^{(0)}\right)$
    $\Delta x^{(0)} \leftarrow -\nabla f\left(x^{(0)}\right)$
    $k \leftarrow 0$
    Loop
        $t^{(k)} \leftarrow$ line_search $\left(f, x^{(k)}, \Delta x^{(k)}\right)$
        $x^{(k+1)} \leftarrow x^{(k)} + t^{(k)} \Delta x^{(k)}$
        If $\left\|\nabla f\left(x^{(k+1)}\right)\right\|_2^2 \leq \epsilon$ then terminate
        If $k+1 \equiv 0$ mod $m$ then
            $\beta_{FR}^{(k+1)} \leftarrow 0$
        Else
            $\beta_{FR}^{(k+1)} \leftarrow \frac{\left(\nabla f\left(x^{(k+1)}\right)\right)^T \nabla f\left(x^{(k+1)}\right)}{\left(\nabla f\left(x^{(k)}\right)\right)^T \nabla f\left(x^{(k)}\right)}$
        $\Delta x^{(k+1)} \leftarrow -\nabla f\left(x^{(k+1)}\right) + \beta_{FR}^{(k+1)} \Delta x^{(k)}$
        $k \leftarrow k+1$
```

gradient on line 12, effectively causing the algorithm to start over from the current location. Nocedal and Wright motiviate restarting with the following intuition: because the conjugate gradient method converges in m iterations for a quadratic objective, if we assume that there is some neighborhood of the optimum in which the objective is very close to being quadratic, then provided that we "start over" at some point once we've entered this region, we will converge to the optimum m iterations later.

It is also worth noting that it is not obvious that Δx will always be a descent direction. In fact, while it is possible to guarantee this property by placing certain conditions (called the "strong Wolfe conditions") on the line search, backtracking line search does not satisfy them (algorithm 3.2 of Nocedal and Wright is an example of a line search which does). In practice, at least on this homework, this is not an issue, but it's something to keep in mind.

Complete the implementation of the Fletcher-Reeves conjugate gradient method by filling in "algorithm_cg.m".

1.4 Quadratic objective

The matlab script "main_quadratic.m" creates a contour plot for the following objective (contained in "objective_quadratic.m"):

$$f(x) = \frac{1}{2} x^T Q x + v^T x$$

for:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 30 \end{bmatrix}$$
$$v = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

and overlays this contour plot with the iterates of each of our four algorithms, using *exact* line search, starting from:

$$x_0 = \begin{bmatrix} 4 \\ 0.3 \end{bmatrix}$$

How many iterations does each algorithm take to converge?

Why do conjugate gradient, BFGS and Newton take this number of iterations?

What is the relationship between consecutive directions in gradient descent, conjugate gradient and BFGS?

Play around with different values of Q and v, and explore the impact of various choices on the performance of gradient descent. Describe your results.

1.5 Sum-exp objective

The matlab script "main_sum_exp.m" creates a contour plot for the following objective (contained in "objective_sum_exp.m"):

$$f(x) = \sum_i \exp\left(a_i^T x + b_i\right)$$

for a_i and b_i chosen as in equation 9.20 (pg 470) of Boyd and Vandenberghe. It then overlays this contour plot with the iterates of each of our four algorithms, using *backtracking* line search with $\alpha = 0.4$ and $\beta = 0.9$. Describe the performance of the four algorithms (it may be helpful to zoom in on the region of the plot containing the optimum).

2 Logistic regression

Suppose that we have a list of feature vectors $x_i \in \mathbb{R}^n$ and corresponding class labels $y_i \in \{-1, 1\}$. We will assume that the log-odds of a sample being in each of the two classes are a linear function of X. That is, that:

$$\log \frac{P(Y=1 \mid X)}{P(Y=-1 \mid X)} = w^T X \tag{1}$$

This assumption, combined with the fact that probabilities must sum to one, gives that:

$$P(Y \mid X) = \frac{e^{Y w^T X}}{1 + e^{Y w^T X}}$$

We will find a weight vector w which maximizes the log-likelihood of the data under the assumption that the labels are conditionally independent given the features:

$$\begin{aligned}\mathcal{L}(w \mid x, y) &= \log \prod_{i=1}^{n} P(Y = y_i \mid X = x_i) \\ &= \sum_{i=1}^{n} \left(y_i w^T x_i - \log\left(1 + e^{y_i w^T x_i}\right)\right) \end{aligned} \tag{2}$$

You may wish to derive equation 2 from equation 1, but this is optional.

For more on logistic regression, refer to:

- Trevor Hastie, Robert Tibshirani and Jerome Friedman. "The Elements of Statistical Learning". Springer, 2001.

2.1 Convexity, gradient and Hessian

Maximizing the log-likelihood in the logistic regression model is clearly equivalent to the following:

$$\text{minimize} \; : \; \sum_{i=1}^{n} \left(-y_i w^T x_i + \ln\left(1 + e^{y_i w^T x_i}\right)\right)$$

Prove that this function is convex (hint: start out by proving that $\ln(1 + e^z)$ is convex as a function of z, and then use the results from section 3.2 of Boyd and Vandenberghe).

Calculate the gradient and Hessian (with respect to w) of this objective function.

Fill in "objective_logistic.m" with this objective function, and the gradient and Hessian which you just calculated.

2.2 Experiments

You are provided with two synthetic datasets for logistic regression: "logistic_small.csv", which contains one thousand 10-dimensional examples, and "logistic_large.csv", which contains one thousand 50-dimensional examples. The matlab script “main_logistic.m” loads one of these files, and plots the log-suboptimality versus the elapsed runtime for each of the four algorithms, using backtracking line search with $\alpha = 0.4$ and $\beta = 0.9$. Plot the performance of the algorithms for both datasets.

Which algorithms perform better on the low-dimensional dataset than on the high?

Can you explain this behavior?

Modify the script to plot the iteration number, rather than elapsed time, on the horizontal axis. Plot the performance of the algorithms for both datasets.

What happens to the apparent relative performance of the algorithms?

Recall the definitions of linear and quadratic convergence. Imagine that we plot log-suboptimality against iteration number for an “ideal” algorithm which exhibits linear convergence.

What should be the shape of the plot?

What about for quadratic convergence?

What does this tell you about the empirical rates of convergence of the four algorithms, on “logistic_large.csv”?

3 Play around (optional)

Experiment on these objectives (or any others you are interested in) with: different settings of the line search parameters, different line search algorithms (algorithm 3.2 of Nocedal and Wright would be a good choice), or different variants of the nonlinear conjugate gradient algorithm (Polak-Ribière, for example). Try repeating problems 1.4 and 1.5 with different choices of parameters (Q, v, a_i and b_i) for the objective functions. Report your results and conclusions.