Description

In this assignment, you will implement Newton’s method.
You should turn in an archive containing all of your source code, plots, and a document containing a brief description of your code (how to run it, what parameters it takes, etc), and answers to all questions. Email this archive to your TA, at cotter@tti-c.org.

Please remember to turn in a document, and to answer the questions (in particular, talk about what your plots mean)!

1 Newton’s Method

1.1 Implementation

Implement Newton’s method descent (algorithm 9.5 of Boyd and Vandenberghe) with a backtracking line search (algorithm 9.2). Your function should terminate once the stopping criterion $\lambda^2(x) \leq 2\epsilon$ is satisfied.

Your function should take seven parameters: $f$ (the function to minimize); $\nabla f$ (the gradient); $\nabla^2 f$ (the Hessian); $\epsilon$ (the stopping threshold); $\alpha$ and $\beta$ (the parameters to the backtracking line search); $x$ (the starting point). It should return a list of triples $(x, f(x), t)$, with one entry per iteration, containing: points $x$; objective function values $f(x)$; line search scale $t$.

As in the previous assignments, code defensively, and comment:

1. On entry, check that $\epsilon > 0$, $0 < \alpha < \frac{1}{2}$ and $0 < \beta < 1$, raising an error if any of these conditions is not satisfied
2. Check that you do not enter an infinite loop, by raising an error if some ridiculously large number of iterations are performed
3. Do the same for the backtracking line search loop
4. Try to evaluate the function $f$, gradient $\nabla f$ and Hessian $\nabla^2 f$ as few times as possible per iteration
5. Comment any portion of your code of which the interpretation is not obvious

1.2 Logistic regression

1.2.1 Hessian

Recall that, in the logistic regression problem solved on the previous programming assignment, you solved the following:

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

You also, on the previous programming assignment, proved convexity, calculated the gradient, and wrote implementations of $f$ and $\nabla f$. Calculate the Hessian $\nabla^2 f$ of the above objective function, and implement a function to calculate it.
1.2.2 Optimizing

Using the data provided for the previous programming assignment (http://bleu.uchicago.edu/CMSC34500/pa2/data.csv), optimize the above objective function using Newton’s method, with initial point \( w = 0 \).

Optimize the above objective function using steepest descent with the quadratic norm defined by the Hessian at the initial point \( \nabla^2 f(\vec{0}) \). Using your estimate of \( w^* \) from the previous problem set, also run steepest descent with quadratic norm defined by the Hessian at the optimum \( \nabla^2 f(w^*) \).

Create a plot of the log-error \( \log |f(x) - p^*| \) versus iteration number for: gradient descent with backtracking line search (which you did on the previous programming assignment); Newton’s method; both versions of steepest descent. All four curves should be on the same plot (in different colors or line styles). Discuss the relative performance of these algorithms.

1.3 Comparisons

On the last programming assignment, you optimized the convex function:

\[
f_A(x) = \ln \left( 1 + e^{x^T A x} \right) + 1^T x
\]

Where \( x \in \mathbb{R}^n \), \( A \in \mathbb{S}^n_+ \), and \( 1 \in \mathbb{R}^n \) is the all-ones vector. In this problem, we will again optimize this function, but in a higher dimensional space, with:

\[
A = \text{diag} \left( 1, \kappa^{-\frac{1}{n-1}}, \kappa^{-\frac{2}{n-1}}, \ldots, \kappa^{-1} \right)
\]

Calculate the gradient and Hessian of the above objective function. Write functions to calculate \( f_A(x) \), \( \nabla f_A(x) \) and \( \nabla^2 f_A(x) \) (your work on the previous programming assignment should come in handy here). Note that, since \( A \) is a diagonal matrix, actually constructing it and multiplying by it would be wasteful, since only \( n \) of its \( n^2 \) elements are nonzero. If we let \( a \) be the vector of diagonal elements of \( A \), then we may quickly calculate \( A x \) in matlab as “\( a \cdot x \)”, and in python (with numpy) as “\( \text{multiply}(a, x) \)”.

1.3.1 Comparison of Newton’s method with gradient descent

Modify your gradient descent implementation to return the same list of 4-tuples as your Newton’s method implementation, \((x, f(x), t)\), containing: points \( x \); objective function values \( f(x) \); line search scale \( t \).

For \( n = 2 \) and \( \kappa \in \{1, 2, 4\} \), use gradient descent and Newton’s method, with \( \eta = 10^{-4} \), \( \epsilon = \frac{1}{10} \cdot 10^{-8} \) (it’s not important that our stopping criteria for gradient descent and Newton’s method are not directly comparable), \( \alpha = 0.4 \), \( \beta = 0.8 \), and initial point \( x = 1\), to optimize this function, and create the following plots:

- Log error \( \log |f_A(x) - p^*| \) versus iteration number, for both gradient descent and Newton’s method (on the same plot)
- Step size \( t \) versus iteration number, for both gradient descent and Newton’s method (on the same plot)

As in the previous problem set, we’ll need good good estimates of the optimal point \( x^* \) and function value \( p^* = f_A(x^*) \) for each \( \kappa \), in order to create plots. Find these values by running one of your algorithms until it performs a null update (that is, until \( f(x) \) is left completely unchanged by an iteration, within the precision of the arithmetic).

Discuss what these plots show you. In each plot, can you identify the point at which Newton’s method begins to exhibit quadratic convergence?

1.3.2 Effect of problem dimension

For fixed \( \kappa = 10 \), use gradient descent and Newton’s method to optimize this function, for \( n = 2^k \), where \( k \in \{1, 2, \ldots, 10\} \), calculating the runtime of each algorithm, for each \( n \) (in matlab, use the “tic” and “toc” functions;
in Python, use the “time.clock()” function). Repeat each run 10 times, and calculate the mean runtime for each algorithm, for each $n$. Use $\eta = 10^{-4}$, $\epsilon = \frac{1}{2} \cdot 10^{-8}$ (as in problem 1.3.1) the two curves which you will eventually plot are not directly comparable, due to the difference in stopping condition—however, these curves will have such dramatically different shapes that this issue will be irrelevant), $\alpha = 0.4$, $\beta = 0.8$, and initial point $x = \vec{1}$. Create a plot of mean runtime versus $k = \log_2 n$ for both algorithms (on the same plot). Describe what you see, and explain why it is happening.

When calculating $e^{x^T A x}$ inside your functions for $f_A$, $\nabla f_A$ and $\nabla^2 f_A$, when $k$ (and therefore $n$) is large, you may experience overflows. You should handle these by using the following identities:

$$\log (1 + e^y) = \begin{cases} 
 y + \log (1 + e^{-y}) & \text{if } y < 0 \\
 \log (1 + e^y) & \text{otherwise}
\end{cases}$$

$$\frac{e^y}{1 + e^y} = \begin{cases} 
 \frac{1}{1 + e^{-y}} & \text{if } y < 0 \\
 \frac{1}{1 + e^y} & \text{otherwise}
\end{cases}$$

In addition to removing the possibility of overflow, calculating these quantities as above should increase the precision of your calculations.