

Optimization, Programming Assignment #2

May 13, 2010

Description

In this assignment, you will experiment with the log-barrier 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.

1 Log-barrier function

The matlab files “objective_quadratic.m”, “algorithm_gd.m”, “algorithm_newton.m” and “line_search_backtracking.m” are (filled-in) versions of the quadratic objective, gradient descent, Newton’s method, and backtracking line search implementations of the previous programming assignment.

The function in “objective_log_barrier.m” takes as parameters an objective function f_0 , a constraint function ϕ (which has the same calling convention as the objective function), a vector x , and a scaling parameter t , and returns the value, gradient and Hessian of the log-barrier objective:

$$f(x, t) = tf_0(x) + \phi(x)$$

Make sure that you understand the contents of all of these files.

1.1 Implementation

The file “objective_quadratic_log_barrier.m” implements an objective function, and computes the value, gradient and Hessian of:

$$f(x, t) = t \left(\frac{1}{2} x^T Q x + v^T x \right) - \sum_i \log(a_{i,:} x + b)$$

this is the log-barrier objective function $f(x, t) = tf_0(x) + \phi(x)$ for a quadratic objective f_0 , and log-barrier function ϕ for the linear constraints $Ax + b \geq 0$.

Fill in this function.

1.2 Experiments

The matlab script “main_quadratic.m” minimizes a quadratic objective subject to three linear constraints, with $t = m$, m being the number of constraints, in this case 3 (this is an extremely small value of t), and plots the iterates of gradient descent and Newton’s method. What do you observe? Remember that you can zoom in on the plots!

The “main_quadratic2.m” script plots the number of iterations required by gradient descent and Newton’s method for t between $1/8$ and 2 (once more, these are extremely small values of t , but the performance of gradient descent should give you a clue as to why we didn’t use larger t). What do you observe?

2 Log barrier method

2.1 Implementation

One of the most appealing properties of the log-barrier method is that, with constraints of the form $f_i(x) \geq 0$, and given oracle access to each constraint function f_i and its first and second derivatives, we may calculate the value, gradient and Hessian of the log-barrier function $g_i(x) = -\log f_i(x)$. The matlab file “objective_scalar_constraints.m” implements this. It takes as parameters a *cell array* of constraint functions f_i and a point x , and returns $\sum_i g_i(x)$, $\sum_i \nabla g_i(x)$ and $\sum_i \nabla^2 g_i(x)$. Each constraint function takes x as a parameter, and returns $f_i(x)$, $\nabla f_i(x)$ and $\nabla^2 f_i(x)$.

A cell array is a matlab data structure which may be treated similarly to a matrix. The main difference is that it may contain any object (not just scalars), including, as we have already seen, function handles. The notation for accessing the elements of a cell array is the same as that for matrices, with curly braces substituted for parentheses: for a one-dimensional cell array a , the i th element is $a\{i\}$ (instead of $a(i)$, as it would be for a vector), and for a two-dimensional cell array, the i, j th element is $a\{i, j\}$.

Fill in “objective_scalar_constraints.m”.

The file “constraint_quadratic.m” implements a constraint function which returns the value, gradient and Hessian of:

$$f_i(x) = \frac{1}{2}x^T Qx + v^T x + c$$

which will be used to implement the constraint $f_i(x) \geq 0$. This constraint function is almost identical to the quadratic objective function—the only difference is the presence of a constant term c .

Fill in “constraint_quadratic.m”.

The file “algorithm_log_barrier.m” contains a mostly-complete implementation of the log-barrier method, algorithm 11.1 of Boyd and Vandenberghe. This function takes (among other things) a cell array of constraint functions as a parameter, and constructs the log-barrier objective $f(x, t) = tf_0(x) - \sum_i \log f_i(x)$ using “objective_log_barrier.m” and your filled-in version of “objective_scalar_constraints.m”.

Fill in “algorithm_log_barrier.m”.

2.2 Experiments

The script “main_linear.m” optimizes a linear program:

$$\begin{aligned} \text{minimize} & : v^T x \\ \text{subject to} & : a_i x \geq b_i \end{aligned}$$

The particular LP which you solve is the “personnel scheduling” example of:

- Hillier and Lieberman. “Operations Research”. McGraw-Hill. 2005

A description of (and motivation for) this LP may be found on pages 56-58 (eighth edition).

The parameter μ controls how quickly the log-barrier scaling parameter t changes between inner Newton optimizations. The generated plot shows the total number of Newton iterations performed for a run of the log-barrier method for various values of μ , and $t_0 = 1$.

Describe the dependence which you expect to see here.

Does the plot meet your expectations?

What is the solution which is found for this LP?

3 Matrix constraints (optional)

3.1 Implementation

You will now extend the implementation of the log-barrier method to handle matrix constraints of the form $f_i(x) \succeq 0$, which constrain the matrix $f_i(x)$ to be positive semidefinite, rather than constraining the scalar

$f_i(x)$ to be nonnegative (for scalars, these notions are equivalent, so in particular your filled-in version of “constraint_quadratic.m” will continue to work fine, once you’ve completed this problem).

Because f_i is in general matrix-valued, the “gradient” and “Hessian” which are returned by matrix constraint functions must return *matrix* derivatives. We will represent these using three- and four-dimensional matrices, with the indices chosen such that, for g the 3D matrix of first derivatives and H the 4D matrix of second derivatives, $g_{i,j,k} = \frac{\partial}{\partial x_i} f_{j,k}$ and $H_{i,j,k,\ell} = \frac{\partial^2}{\partial x_i \partial x_j} f_{k,\ell}$.

The file “objective_matrix_constraints.m”, analogously to “objective_scalar_constraints.m”, calculates the value, gradient and Hessian of the log-barrier function $g_i(x) = -\log|f_i(x)|$ (here, $|\cdot|$ denotes the determinant). The difference from the function which you filled-in earlier is that this one must handle the 3D and 4D matrices of derivatives returned by matrix constraint functions.

The code makes use of the matlab “permute” function, which simply permutes the indices of a matrix. For example, if $y = \text{permute}(x, [4, 3, 2, 1])$, then $y_{i,j,k,l} = x_{\ell,k,j,i}$. You might want to read the matlab documentation of this function (or anything else with which you are unfamiliar).

Fill in “objective_matrix_constraints.m”.

You may find it helpful to use the product rule for matrix differentiation:

$$\frac{d}{dx} AB = \frac{dA}{dx} B + A \frac{dB}{dx}$$

as well as the following identities:

$$\begin{aligned} \frac{d}{dx} \log |A| &= \text{tr} \left(A^{-1} \frac{dA}{dx} \right) \\ \frac{d}{dx} \text{tr} (A) &= \text{tr} \left(\frac{dA}{dx} \right) \\ \frac{d}{dx} A^{-1} &= -A^{-1} \frac{dA}{dx} A^{-1} \end{aligned}$$

The file “constraint_sdp.m” implements a constraint function which returns the value, 3D matrix of partial first derivatives, and 4D matrix of partial second derivatives, of:

$$f(x) = A + \text{diag}(x)$$

Fill in “constraint_sdp.m”.

In order to make use of matrix constraints in the log-barrier implementation, you must modify “algorithm_log_barrier.m” by replacing the call to “objective_scalar_constraints” with a call to “objective_matrix_constraints”. Make this change.

3.2 Experiments

Re-run “main_linear.m”, using the new version of “algorithm_log_barrier.m” (which now handles matrix constraints), and ensure that the results are the same as they were when you used the scalar-constraint code path. Include the plot, and the solution to the LP, in your submission.

You will now optimize an instance of the SDP contained in equation 11.47 (page 603) of Boyd and Vandenberghe:

$$\begin{aligned} \text{minimize} & : 1^T x \\ \text{subject to} & : A + \text{diag}(x) \succeq 0 \end{aligned}$$

The matrix A is contained in “matrix.csv”.

Run “main_sdp.m”, and report the solution which is found for this SDP.