Matlab Barrier Method Report

Matthew Broussard

Dependencies: Yalmip, SDP3

OptimizationTest.m

The top level of my function for my implementation of the barrier method is OptimizationTest.m. OptimizationTest begins with several user-contributed parameters influencing various functions.The inputs used in OptimizationTest are: n gives the number of variables in the primal problem, m gives the number of constraints, and $condnum$ gives an approximate condition number for the Hessian matrix. These are all integers.

The next level lower, Barrier.m, will use s_0 and μ. They control the number and difficulty of outer iterations (inversely proportional, alas). Next in the chain, ConsNewton.m uses ϵ and $maxiter$ to determine stopping criteria. Finally Linesearch.m uses α and β as parameters controlling the gains it requires and the speed it shrinks step length, respectively. Their domains are described below.

We use the code given in the assignment to generate a problem, which automatically generates F, $GradF$, $HessF$, A, c, and Q. These will be inputs for Barrier, the next function in the chain. Their domains are described below.

OptimizationTest gives the optimal x the function comes up with, the optimal function value, the number of outer iterations, and the total number of inner iterations as outputs. They are labeled in the generated text file.

Running this script runs my full algorithm on the generated problem.

Barrier

OptimizationTest calls Barrier with the following arguments:

- F is the function to be optimized, in the form $F(x) = x^T * Q * x + c^T * x \; s.t. \; A * x = b$.
- $GradF$ is the gradient of F.
- $HessF$ is the Hessian of F.
- $xfeas$ is a strictly feasible point. It must solve $A * x = b$ and be positive in all dimensions.

- A is the constraint matrix, size $m \times n$.
- s_0 and μ are the initial accuracy of the approximation Barrier will construct and the speed at which that accuracy improves for each iteration. Generally, high s_0 front loads the processing demands, while a high μ makes each iteration take longer but requires fewer iterations overall. The function stops once $\frac{m}{s_0 * \mu^i} < \epsilon$. $s_0 > 0$, $\mu > 1$.
- ϵ is the tolerance of the problem. Small ϵ values will give accurate but computationally expensive answers. Large values give cheap but less accurate answers. $\epsilon > 0$.
- α, β, and $maxiter$ will all be passed on without being used.
- Q is an $n \times n$ positive semidefinite symmetric matrix. Defined by the function to be optimized.
- c is a vector, again defined by the given problem.
- fid is used if you wish to write your output to a .txt file rather than the command window.

This layer sets up the centering steps of the barrier method algorithm. Each iteration uses ConsNewton to optimize the approximation function $s * f(x) + \Sigma log(x_i)$ using **x** as the initial vector, then updates **x** with the returned optimal x and s with $s * \mu$, thus setting up the conditions for the next iteration. The function also checks the ending condition (i.e. that $m/s < \epsilon$), returning the following outputs once finished:

- $Solutionx$, the optimal x.
- $Solutionfval$, the optimal function value.
- $Solutioniter$, the number of outer iterations.
- $Inneriter$, the total number of inner iterations as outputs.

ConsNewton

Barrier calls ConsNewton with the following arguments

- F, $GradF$, $HessF$, A, $epsilon$, and fid have been described already. Likewise, α and β will be passed along without being used.

- $xinit$ is the starting point for the algorithm, and must be feasible.
- $maxiter$ is the maximum number of iterations you will allow. This is a positive integer. Set too low, the function may give up too soon, print an error message, and give an incorrect answer.
- s is a parameter constructed in Barrier. A high s will return a more accurate estimate of the solution, but will be difficult computationally without a sufficiently accurate $xinit$.

ConsNewton optimizes a convex quadratic form with positive semidefinite Hessian over the positive orthant via Newton's method. It solves the system of equations described on page 545 of Boyd & Vandenburghe to determine the search direction d_{nt}, calculates the decrement, then checks a stopping condition ($decrement^2/2 < \epsilon$). If the condition fails, the function performs a line search at its current x in the direction d_{nt}, then updates x with the returned improved value. The next iteration begins so long as the maximum number of iterations has not been achieved.

If the stopping condition is met or the maximum number of iterations is achieved, ConsNewton returns the following:

- $Outputstatus$, which tracks whether ConsNewton reached a good approximation before it reached the maximum number of iterations.
- 'Converged' means the function returned valid outputs, while 'All is lost' indicates the function ran out of iterations.
- $Outputiter$, which counts the iterations used.
- $Outputfval$, the optimal function value for the approximation function.
- $Outputx$, the optimal x value for the approximation function.

Linesearch

Linesearch takes inputs as follows:

- F is the function for which the line search is being invoked.
- x is the location where the search is to be performed length n vector.

- d is the search direction (length n vector).
- df is the derivative in the search direction $(GradF * d)$.
- α controls how much the line search must improve the function (proportionate to distance). High α values cause ConsNewton to use small step sizes. $\alpha \in [0, .5)$.
- β controls how quickly you reduce the length of your search vector. Small β makes Linesearch faster, but forces more iterations of ConsNewton because of the short steps it causes. $\beta \in [0, 1)$.
- $tinit$ is an initial value for t. If left blank, $t = 1$.

Linesearch follows the provided code closely. It takes a step length of 1 in the search direction, checks for the percentage of the linear model improvement required by α, then either returns t or updates it with $t * \beta$A and reiterates. However, this version of Linsearch also need to keep the search confined to the positive orthant. I achieved this by taking the vector $x + t * d$ and looping $t = \beta * t$ until $x + t * d$ was positive in all dimensions.

Linesearch returns only t, a length along the direction of steepest descent which leaves $x + t * d$ feasible and reduces the function value sufficiently. t is a positive scalar.

Test

Test.m runs one instance each of Barrier.m, ConsNewton.m, and Linesearch.m. A simple example problem is preloaded, but other feasible problems could be used.

Alphatest and Further Exploration

This code tests the effect each input (save ϵ) has on the number of iterations required. It creates graphs showing

- α from .005 to .5
- β from .009999 to .9999
- s_0 from 1/5 to 20
- μ from 11 to 1001

- n from 11 to 1001
- m from 1 to 50
- $condnum$ from 1 to 10

As suggested in the book, these tests suggest that a relatively large α (close to .5) converges much more quickly than smaller α, a small β takes a long time to converge but too large a β can also slow the process somewhat (near .5 was the fastest for this test), and larger s_0 tended to be more efficient than smaller, although that could well change at higher values. μ took its optimal value around 40, which is a little higher than I expected from the book, and the data were far noisier in this test than the previous ones.

The later tests were perforce run on different problems in each iteration, so the interpretation is somewhat less clear. However, as we might expect the iterations increased roughly logarithmically with n. Increasing m seemed much harder on the system, at least in its reduced range. Given that this script already takes a while to run, I didn't choose to look at extremely high values of m.

Changing $condnum$ surprised me. I expected a much clearer correlation between the condition number and the number of iterations for convergence, but I found quite a bit of variance. $condnum = 5$ was, for instance, one of the highest results, while $condnum = 7$ was amongst the lowest. Nonetheless, the slope does seem positive, so increasing the condition number generally increases the number of iterations.

The graphs for these tests are included as the appropriately named .fig files.

The .zip file also contains OptimizationOutput.txt, a record of the algorithm's run against that of SDPT3.