Final Hand In · Week VIII

CCO · Constraint Continuous Optimization

Wednesday 21st October, 2009

Johan Sejr Brinch Nielsen

Email: zerrez@diku.dk
Cpr.: 260886-2547

Dept. of Computer Science,
University of Copenhagen

Contents

Introduction

Problem 1: Mandatory

I have filled out the Course Evaluation.

Problem 2: Taylor

Taylor Series

A Taylor series is a series expansion of a function $f(x) : \mathbb{R} \rightarrow \mathbb{R}$ in some point a. It is given by:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Taylor's Theorem

Taylor's Theorem states that any real function f(where we can derive the needed derivatives) can be expanded as ($a = 0$):

$$f(x) = f(0) + xf'(0) + \frac{x^2}{x!}f''(0) + \ldots + \frac{x^{n-1}}{(n-1)!}f^{n-1}(0) + \int_0^x \frac{(x-u)^{n-1}}{(n-1)!}f^{(n)}(u)du$$

Note the remainder denoted by the integral. Taylor series expansion are an important tool in optimization, since it can result in a simpler function (f.x. a quadratic function) which approximates the original objective function.

Problem 3: Line Search Methods

The principle behind line search methods is to minimize the objective function by first computing a descent direction and then an "appropriate" step size to jump. Examples of line search methods are steepest descent and Newton's method.

The Armijo and Wolfe conditions is a set of conditions that restrict the area the optimization method will consider as "good" choices. Line search methods can use the conditions to select a step size after the step direction has been chosen.

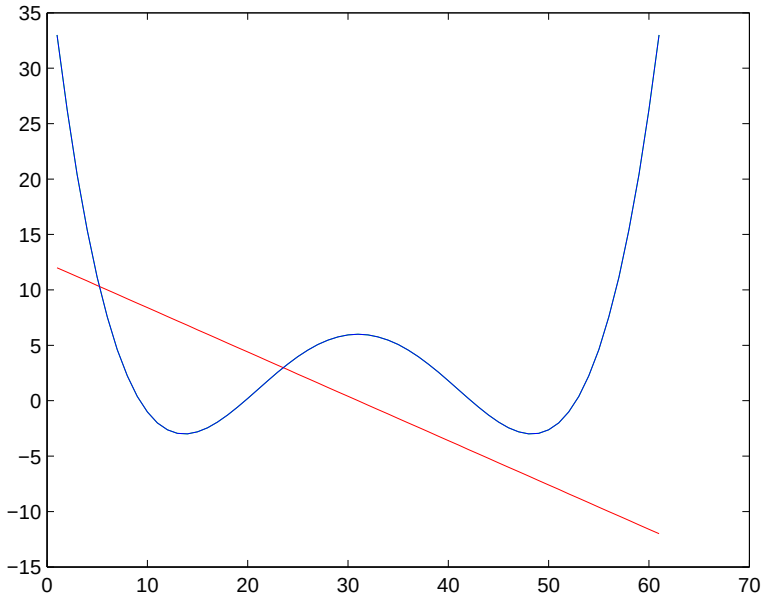
Armijo Conditions

The Armijo conditions are satisfied when:

$$f(x + \alpha p_k) \leq f(x) + c_1 \alpha p^T \nabla f(x)$$

Where x is the current point, p is the descent direction, α is the step size and $0 < c_1 < 1$ is a constant relaxing the condition. The result of this condition is that the taken step should lower the objective value at least as much as what we would have gotten following a straight line in the same direction. c_1 relaxes the gradient of this direction.

Figure 1: Example of Armijo Back-tracking



Armijo back-tracking visualized as a line (red) going through a 2D objection function (blue). Only values beneath the red line is permitted. The first of the two intersections (to the left) is the current solution point x, while the second denotes the maximum distance reachable under the conditions.

Wolfe Conditions

The Wolfe conditions are satisfied when the Armijo conditions are satisfied and:

$$p^T \nabla f(x + \alpha p) \geq c_2 p^T \nabla f(x)$$

Where x is the current point, p is the search direction, α is the step size and $0 < c_1 < c_2 < 1$ is a constant relaxing the condition.

The condition states that the slope of the function in the chosen point must at least that of the starting point.

Armijo Back-tracking

The projected Armijo back-tracking will, given a search direction p, find a step size α for which $f(x + \alpha p)$ satisfies the Armijo conditions.

The algorithm starts with a large step length (e.g. 1). The step length is then lowered until a point that meets the condition is found.

If the chosen search direction is a descent direction, such a point must necessarily exist (the condition is satisfied when $\alpha = 0$). However, it may be very close to the initial x. If so, the step taken can be very short, nearly

nothing at all. It is important to ensure a minimum step size to avoid repeating computations in x.

The algorithm can be described with the following pseudo code:

```
while not (Armijo-condition of (x+a*c1*pk)) do
  a = a * c
end
```

Where $0 < c < 1$.

Problem 4: Trust Region Methods

Trust region methods works by first deciding on a step size and then compute the a decent direction within the step size (the "region").

The idea is to use a model function m that approximates the objective function f. Knowing the step size λ, the goal is to find the best search direction in the model function (which is easy to minimize) and then follow this direction in f. Of course one needs to verify that the model is in fact precise enough to yield a decent in f. If not, the λ is lowered. λ is a measure for the precision of the model function. The region containing points with a distance lower than λ away from x_k is called the "trust" region.

Levenberg-Marquardt

The Levenberg-Marquardt uses an interpolation between Gauss-Newton and steepest descent. Intuitively, it tries the direction of Gauss-Newton and then falls back to steepest descent when unsuccessful.

The Levenberg-Marquardt update value is defined as:

$$\Delta = -(J^T J - \lambda I)^{-1} J^T r$$

Where J is the jacobian matrix, λ is the step modifier, r is the residual vector and Δ is the next modification of the approximation. When the step modifier λ gets large, it will dominate the inverted matrix, such that:

$$\Delta \approx -(-\lambda I)^{-1} J^T r =$$

$$-\frac{1}{\lambda} I J^T r$$

It is now clear why λ becomes a step modifier and how Levenberg-Marquardt graduately switches to steepest descent. λ determines whether $J^T J$ or $-\lambda I$ should dominate the direction. The method adapts both step size and direction during the optimization, however the step size is always lowered. Hence, it makes sense to classify this method as a trust region method (bound by the starting λ value).

Dogleg

The Dog-Leg method is based on a combination of the Cauchy point and a Quasi-Newton method. It uses a second degree Taylor expansion as its model. It will choose its search direction based on the following three cases:

1. When the point suggested by Newton's method is within the trust region, this point is selected.

2. If this is not the case and the trust region requests a smaller step than that of the Cauchy point, the direction towards this point is followed as far as the region permits.

3. In the third case, when the region boundary is between the Cauchy point and the Newton point, the intersection between the limiting boundary and the line between these two points are chosen.

As can be seen, the possible directions is given by two joined lines (from p_k to the Cauchy point and further to Newton's point). This gives a bending line (a "dog leg"). Note that since the model function is quadratic, the Newton method will minimize it in a single iteration. So there is never a need to go further than the point found by Newton's method.

Specifically, p_k may take one of the following forms:

1.
$$p_{k+1} = p^B$$

2.
$$p_{k+1} = \frac{\Delta}{\mid p^U \mid} p^U$$

3.
$$p_{k+1} = p^U + \frac{\Delta - \mid p^U \mid}{\mid p^B \mid + 2p^{U\prime} p^B} p^B$$

Where Δ is the size of the trust region, p^B is the point suggested by Newton's method and p^U is the Cauchy point. The two latter points are defined as:

$$p^B = -B^{-1} \cdot g$$

$$p^C = \frac{g'g}{g'Bg} \cdot g$$

Where g is the gradient in the point x_k, B is an approximation of the Hessian and B^{-1} is an approximation of its inverse; both B and B^{-1} are positive definite matrices.

There is one detail in the algorithm yet to be discussed. That is what size to choose for λ and how to adjust this dynamically. Generally, one should lower λ when no decent direction is found and only raise λ when the step taken is as long as λ (the step is restricted by λ). One way to achieve this is by comparing the actual improvement in the objective function with that of the model (the expected improvement).

Problem 5: First Order Optimality Conditions

The Karush-Kuhn-Tucker (KKT) conditions are a set of *necessary* conditions that must be satisfied before x can be a minimum of the function f. If the KKT conditions are not satisfied, one can be sure that x is not a minimum. However, whenever they are satisfied, one only knows that x *might* be a minimum.

Consider the minimization problem:

$$\begin{array}{rl} \min_x & f(x) \\ \text{subject to:} & c(x) \geq 0 \end{array}$$

Where $x \in R^n$, c are the constraints and $f : \mathbb{R}^n \to \mathbb{R}^n$.

I will now derive the KKS conditions for this minimization problem. Let us first examine the first-order Taylor expansion of the objective and constraint function. To maintain feasibility we require that $c(x + d) = 0$, where d is the direction. Hence,

$$0 = c(x + d) \approx c(x) + \nabla c(x)^T d = c(x)^T d$$

The last step is due to $c(x) = 0$. When this condition is satisfied, we maintain feasibility after the step in direction d.

To ensure an improvement in the objective function, we seek to satisfy the condition:

$$0 > f(x + d) - f(x) \approx \nabla f(x)^T d$$

That is the objective value should drop when moving in the direction of d (a descent direction was chosen).

Let $x^\star$ be a local minimizer of f. Then there exists a Lagrange multiplier $\lambda^\star$ such that the KKS conditions are met:

$$\nabla_x L(x^\star, \lambda^\star) = 0$$

$$c_i(x^\star) = 0$$

$$c_i(x^\star) \geq 0$$

$$\lambda^\star \geq 0$$

$$\lambda^\star c_i(x^\star) = 0$$

Problem 8: Splitting Method

The Splitting method solves a LCP by first splitting the matrix A in two. Hereafter, it defines a series of sub-problems which also belongs to the LCP class. However, hopefully these are easy to solve.

The first step is to split A into to new matrices M and N such that:

$$A = M - N$$

The sub-problem is now defined as:

$$\begin{aligned} M\lambda^{k+1} + c^k &\geq 0 \\ \lambda^{k+1} &\geq 0 \\ (\lambda^{k+1})^T(M\lambda^{k+1} + c^k) &= 0 \end{aligned}$$

Where $c^k = b - N\lambda^k$. Letting $M\lambda^{k+1}$ be $M\lambda^k$ makes the left-hand side of the first inequality become:

$$M\lambda^k + c^k = M\lambda^k + b - N\lambda^k$$

$$= A\lambda^k + b$$

And the sub-problem is back to the original LCP. Hence, the only difference lies in using λ^{k+1} together with M and λ^k together with N. Since λ^k is the approximation of $\lambda^\star$ at iteration k, and λ^{k+1} that at iteration $k+1$, this is a fixed point computation yielding a solution to the original LPC.

Not surprisingly, this fixed point iteration can be solved iteratively. A bit more surprising is, that we need to apply the minimum map reformulation. This reformulation yields the following root-search problem:

$$\min(\lambda^{k+1}, M\lambda^{k+1} + c^k) = 0$$

This formulation ensures the inequality constraints are fulfilled, since neither can be negative (the smallest of the two must be 0). This is equivalent to the equality constraint, that ensures the product of the two is 0. Hence, the new formulation is the same as the sub-problem.

We can now work on the new formulation:

$$\begin{array}{lll} \min & (\lambda^{k+1}, M\lambda^{k+1} + c^k) & = 0 \quad \Rightarrow \\ \min & (0, M\lambda^{k+1} + c^k - \lambda^{k+1}) & = -\lambda^{k+1} \quad \Rightarrow \\ \max & (0, -M\lambda^{k+1} - c^k + \lambda^{k+1}) & = \lambda^{k+1} \end{array}$$

The result is a fixed point problem (it contains λ^{k+1} on both sides). However, there is one more trick up the sleeve: case analysis on the sign of the ith element of $S = -M\lambda^{k+1} - c^k + \lambda^{k+1}$:

Case 1: $S_i = (-M\lambda^{k+1} - c^k + \lambda^{k+1})_i < 0$
If this is the case $\lambda_i^{k+1} = 0$, since $0 > S_i$.

Case 2: $S_i = (-M\lambda^{k+1} - c^k + \lambda^{k+1})_i \geq 0$
Clearly $S_i = \lambda_i^{k+1}$, since $S_i > 0$. That is:

$$(-M\lambda^{k+1} - c^k + \lambda^{k+1})_i = \lambda_i^{k+1} \Rightarrow$$

$$(-M\lambda^{k+1} - c^k)_i = 0 \Rightarrow$$

$$(-M\lambda^{k+1})_i = c_i^k$$

This situation is interesting. λ_i^{k+1} is now present on the left hand side only. Assuming that M is reversible, we can compute λ^{k+1} as:

$$\lambda_i^{k+1} = (-M^{-1}c^k)_i = (-M^{-1}(b - N\lambda^k))_i$$

Combining the two cases yields the following closed term for λ^{k+1}:

$$\lambda^{k+1} = \max(0, M^{-1}N\lambda^k - b)$$

In order to solve the LCP, one can approximate on $\lambda^\star$ using above equation until an acceptable error has been reached.

Conclusion

Source Code