10-725: Optimization **Fall 2012**

Lecture 1: August 28

Lecturer: Geoff Gordon/Ryan Tibshirani *Scribes: Lu Xie and Zeyu Jin*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

1.1 Administrivia

This class has no particular prerequisites, but life will be hard if you don't have experience in any of the following fields:

- Algorithms
- Programming
- Linear algebra
- Machine learning and/or statistics
- *Most important: formal thinking*

Working together is encouraged but you need to be able to solve problems by your own.

Our course website is: http://www.cs.cmu.edu/ ggordon/10725-f12/

And our google group is: http://groups.google.com/group/10725-f12/

Work hard! Have fun!

1.2 An optimization example

Here is a simple economy example. Suppose there are m agents and n goods. For each agent i, the production of goods is $p_i \in R^n$, and the consumption is $c_i \in R^n$. The cost of producing p_i for agent i is $s_i(p_i)$, and the utility of consuming c_i is $d_i(c_i)$. Now we want to maximize the total utility subtracted by the total cost, given the constraint that the production and consumption are balanced:

$$\max_{p_i,c_i} \sum_i [d_i(c_1) - s_i(p_i)] \text{ s.t. } \sum_i p_i = \sum_i c_i \tag{1.1}$$

The solution of this problem is called *Walrasian equilibrium*. One idea to solve this problem is to introduce a price term λ_j to good j and let the agents optimize production/consumption independently:

$$\max_{p_i,c_i} \sum_i [(d_i(c_i) - \lambda c_i) - (s_i(p_i) - \lambda p_i)] = \sum_i \max_{p_i,c_i} (d_i(c_i) - \lambda c_i) - (s_i(p_i) - \lambda p_i). \tag{1.2}$$

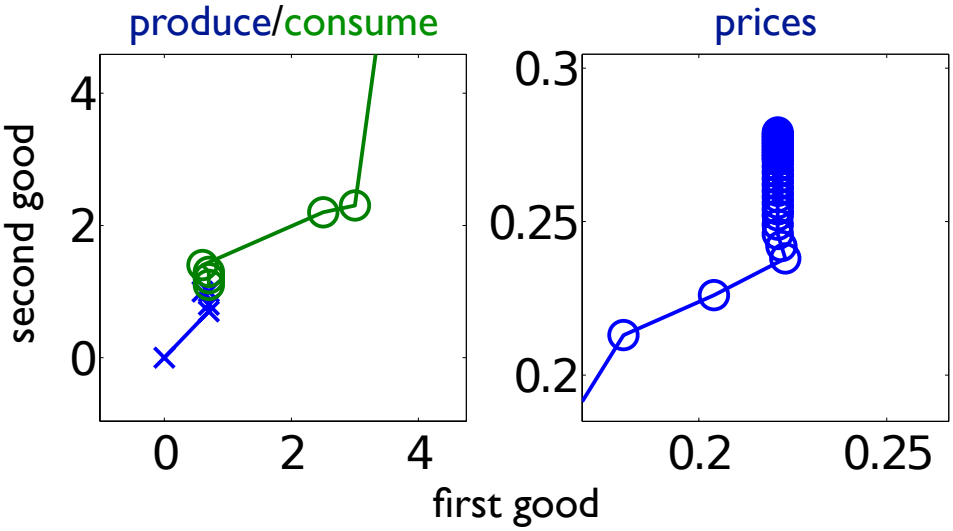

Figure 1.1: Results for a random market

The idea behind this is that the price term will penalize the utility of consumption and also compensate the cost of production. Intuitively the production goes up and the consumption goes down with a higher price and vice versa. Presumably the "just right" prices will satisfy the constraints. This is called the *tâtonnement* algorithm. Here is the pseudo code:

$\lambda \leftarrow [0, 0, 0, ...]^T$
for k = 1,2,...
    each agent solves for p_i and c_i at price λ according to (1.2)
    $\lambda \leftarrow \lambda + t_k(\sum_i c_i - \sum_i p_i)$ where t_k is a positive value called learning rate.
    if the $\sum_i c_i$ almost equal to $\sum_i p_i$, exit the loop
endfor

We usually pick a small positive number for t_k. Please refer to Fig. 1.1 for the resulting plots for a random market with 3 agents and 2 goods and note how fast it converges to equilibrium.

So why is *tâtonnement* so cool? First this algorithm is nearly obvious once the problem is setup. Leon Walras (1874) came out with this algorithm based on the ideas of Antoine Augustin Cournot (1838), but interestingly he didn't know about the convergence. The analysis of convergence (1950s; Arrow and Debrau) is subtle; it only needs concepts from later in this course, including duality, dual decomposition and convergence rates of gradient descent. The variants of this problem, however, need even more subtlety.

1.3 Formalizing optimization problems

A "typical" optimization problem has the following form:

$$\min_x f(x) \text{ s.t.}$$

$$g_i(x) \leq 0, i \in \text{INEQ}$$

$$h_j(x) = 0, j \in \text{EQ}$$

Different set of constraints requires different algorithm. For example, linear programming can be used when $f()$ and $g_i()$ are all linear functions, convex programming can handle the situation that $f()$ and $g_i()$ are all convex functions, and in a more complicated case where $f()$ is a linear function and $g_i(x) = -\min(\text{eig}(\text{reshape}(x, k, k)))$, we will use semi-definite programming.

There are some cool and fun facts about linear programming (LP), which is at least as old as Fourier. First practical LP algorithm is the famous simplex method (Dantzig, 1947). For a long time, the best runtime bounds were exponential while practical runtime were observed much better. Many people thought LPs were NP-hard, but Kachiyan (1979) and Karmarkar (1984) proved that LP can be solved in polynomial time. Later Spielman and Teng (2002) proved that simplex method solves "most" LPs in polynomial time. The other thing is that LPs are P-complete, which indicates the "hardest" polynomial time problem.

Optimization exists in many machine learning (ML) and statistics problems. For example, optimization can be found in regressions, PCA, MLE, SVM and (PO)MDP. Optimization has certain advantages. It's a fast and generic algorithm, it has an expressive form, it connects objective to statistics and it has standard transformations. There are also some exceptions like integration over posterior, non-parametric statistics and belief propagation.

When setting up an optimization problem, this first step is to translate informal thinking into formal expressions. The second step is to decide whether a transformation, including duality, relaxations or approximations will make a hard problem easier. The last step is to pick an algorithm to solve the problem. There many algorithms available, such as first order, interior point, ellipsoid, cutting plane, eigensystems, message passing/relaxation, etc. Some algorithms are smooth while some are not, and some algorithms are a combination of others.

There are many things to consider when picking an algorithm. Comparing the runtime of first order algorithms (gradient descent, FISTA, Nesterov's method) versus higher order algorithms (Newton, log barrier, ellipsoid, affine scaling), the latter ones can cut the number of iterations from the order of $1/\epsilon$ to $\log(1/\epsilon)$, but paying approximately $O(n^3)$ time on each iteration as oppose to $O(n)$ or less.

The problem itself can also affect the choice of algorithms. For example, is this problem balanced (i.e. #constraints $\approx$ #variables) or not? Does the constraint matrix A have any special structure like sparsity or locality? How fast can we compute Ax? What's the degree of "niceness" (e.g. differentiable, strong convex, self-concordant, submodular) of $f()$? Can we split $f(x) = g(x) + h(x)$? Is $f(x)$ "close to" a smooth function? Do we care more about practical implementation or analysis?

1.4 Game time

Please feel free to play a simple online poker game at Geoff's webpage:

http://www.cs.cmu.edu/$\tilde{g}$gordon/poker/

Now think about the best strategy to win!