1 Constrained Convex Optimisation

We now have a slight diversion into constrained convex optimisation, the denouement of which will not be apparent until later. This section follows (REF), and only contains enough to make the optimisation algorithm on graphs

Say, for example, that you are taking a walk up a mountain and would like to calculate the highest point along your path. You may then be tempted to solve a problem which looked like this:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & Ax = b \end{aligned}$$

where f is a function in $\mathbb{R}$, $A \in \mathbb{R}^{n \times m}$ is a matrix, and $x \in X$, X is some convex set.

This is a problem of constrained optimisation, which can be solved by adding a Lagrange multiplier like term to the unconstrained objective (which coincides with the constrained optimisation problem):

$$L(x, y) = f(x) + y^T (Ax - b) \tag{1.0.1}$$

where y is the Lagrange multiplier, and L is referred to as the Lagrangian. The components of y can be interpreted as the rate of change of the objective function as a function of the constraint variable. For example, if we are required to find the maximum height along a path on the mountain, the Lagrange multiplier would represent the rate of change in height along that specific path.

We can find the solution of the original problem via the duality theorem of Fenchel:

$$g(y) = \inf_x L(x, y) = -f^* \left(-A^T y\right) - b^T y \tag{1.0.2}$$

where f^* is the convex conjugate (Legendre-Fenchel) transform of f. The problem is now to maximise g.

This maxima can be expressed in closed form as:

$$x^* = \underset{x}{\text{minimize}} L(x, y^*) \tag{1.0.3}$$

where y^* is the soln of the dual problem.

Together 1.0.2 and 1.0.3 suggest a iterative algorithm, which proceeds iteratively by finding a minima of the objective function (up to a convex transformation) and then updating the Lagrange multiplier until some convergence criterion is met:

$$x^{k+1} := \underset{x}{\text{minimize}} L(x, y^*) \tag{1.0.4}$$

$$y^{k+1} := y^k + \alpha^k \left(Ax^{k+1} - b\right) \tag{1.0.5}$$

This is a form of gradient ascent, with α as the step size and k is the iteration counter. Assuming that g is differentiable, the algorithm is monotone (i.e. $g\left(y^{k+1}\right) > g\left(y^k\right)$.

This algorithm is easily extended to the case where the objective function is linearly separable. That is scenarios where we can split $f(x)$ into a sum of functions $f_i(x_i)$, each over a disjoint domain

$x_i \in X_i$ (in our hill walking example, we can split our chosen path into a series of continuous intervals which lay next to each other). Extending the algorithm is simple: all that needs to be done is to perform the minimisation step in (1.0.5) in each of the functions f_i, and then jointly update the multiplier.

We have assumed the diferentiabilty g throughout, and it is this smoothness condition which allows the previous algorithm to terminate on a solution of the original constrained problem.

For problems with g not differentiable, the augmented Lagrangian is introduced as:

$$L_\rho(x, y) = f(x) + y^T(Ax - b) + \frac{\rho}{2}\|Ax - b\|_2^2 \tag{1.0.6}$$

where ρ is a penalty parameter. That is we're no longer seeking exact solutions, but instead seek solutions close in mean square error.

The associated dual function is $g_\rho(y) = \inf_x L_\rho$. By adding the penalty term, g_ρ is no differentiable, and the $\rho \to 0$ regime corresponds to the solution of the original problem.

The algorithm in this case can be found by first minimising over x and then evaluating the resulting equality constraint residual, i.e.:

$$x^{k+1} := \arg\min x L_\rho \tag{1.0.7}$$

$$y^{k+1} := y^k + \rho\left(Ax^{k+1} - b\right) \tag{1.0.8}$$

This algorithm is known as the method of multipliers for solving the problem (1). Note that even if f is separable, the augmented Lagrangian may not be.

1.1 ADMM

The alternating direction method of multipliers (ADMM), is an algorithm intended to blend the decomposability of the gradient ascent algorithm 1.0.5, with the more general method of multipliers 1.0.8, i.e. when the objective function is decomposable but the dual function is not differentiable. The algorithm solves problems of the form

$$\min_x f(x) + g(z) \tag{1.1.9}$$

$$\text{s.t } Ax + Bz = c \tag{1.1.10}$$

where f and g are assumed to be convex function with range in $\mathbb{R}$, $A \in \mathbb{R}^{p\times n}$ and $B \in \mathbb{R}^{p\times m}$ are matrices (not assumed to have full rank), and $c \in \mathbb{R}^p$. We begin by illustrating a few examples, relevant to the type of problems encountered in signal processing.

Example 1.1. *The Basis Pursuit problem*

$$\begin{aligned} \underset{x}{\textit{minimize}} \quad & \|x\|_1 \\ \textit{subject to} \quad & Ax = b \end{aligned}$$

can be written in the ADMM form (1.1.10):

$$\begin{aligned} \underset{x}{minimize} \quad & f(x) + \|z\|_1 \\ subject\ to \quad & x - z = 0 \end{aligned}$$

where f is the indicator function of $\{x \in \mathbb{R}^n : Ax = b\}$.

Example 1.2. *Similarly the LASSO:*

$$minimise\ \frac{1}{2}\|Ax-b\|_2^2 + \lambda\|x\|_1 \tag{1.1.11}$$

$\lambda > 0$, can be solved by the ADMM with $f = \frac{1}{2}\|Ax-b\|_2^2$ and $g = \lambda\|x\|_1$.

The solution of (1.1.10) is:

$$x* = \inf\{f(x) + g(z) : Ax + Bz = c\} \tag{1.1.12}$$

As previously we form the Lagrangian:

$$L_\rho(x, y) = f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2 \tag{1.1.13}$$

ADMM consists of the iterations:

$$x^{k+1} := \arg\min x L_\rho\left(x, z^k, y^k\right) \tag{1.1.14}$$

$$z^{k+1} := \arg\min z L_\rho\left(x^{k+1}, z, y^k\right) \tag{1.1.15}$$

$$y^{k+1} := y^k + \rho\left(Ax^{k+1} + Bz^{k+1} - c\right) \tag{1.1.16}$$

Note that this algorithm is similar to both the gradient ascent and method of multipliers from the last section: it has an x minimisation step, an z minimisation step and a dual variable update. I.e. the Lagrangian is updated simultaneously with respect to the two primal variables. Separating minimisation into two steps (as opposed to joint minimisation) is what allows for decomposition if f or g (or both) are separable.

2 Constrained Optimisation on Graphs

We model the network as an undirected graph $G = (V, E)$, where $V = \{1 \dots J\}$ is the set of vertices, and $E = V \times V$ is the set of edges. An edge between nodes i and j implies that the two nodes can communicate. The number of nodes node i can communicate is written $\mathcal{N}_i$ and the degree of node i is $D_i = |\mathcal{N}_i|$.

We assume that a proper (or approximate) colouring of the graph is available: that is each node is assigned a number from a set $C = \{1 \dots c\}$, and no node shares a colour with any neighbour.

Given the measurement model from the section (SECTION), we can represent the linear system by concatenating the measurements of each node into a single system. Where, each row of the matrix represents the measurements of a single node. I.e we are trying to solve the following problem:

$$\min||x||_1 \text{ subject to } A_p x = b_p \tag{2.0.17}$$

The problem is coupled at each node by the variable x. To ease this issue, at each node create a local copy, denoted x_j (so there will be J copies of this variable).

Then, for consistency, we constrain the problem, so that each copy is identical: $x_1 = x_2 = \dots x_J$. Equivalently, given the graphical structure of the problem, we require that $x_i = x_j$ for each edge of the network.

So now we are solving the problem

$$\min \frac{1}{J} \sum_J ||x_j||_1 \text{ subject to } A_p x = b_p \text{ and } x_i = x_j \{i, j\} \in E \tag{2.0.18}$$

This is exactly the kind of problem that can be attacked by the Alternating Direction Method of Multipliers, described previously. We present a short example for bipartite graphs, which can easily be generalised to graphs with many colours.

2.1 Example: bipartite graphs

We are here considering graphs which are connected, and 2-colourable, that is nodes 1 to c have colour 1 (red say), and nodes $c+1$ to J have colour 2 (blue). We will demonstrate that we can decompose problem (2.0.18) into c problems which can be executed in parallel by minimising x at the 1st set of nodes, and $J - c$ problems which also can be executed in parallel by minimising at the second set of nodes.

Firstly, for compactness, we introduced the arc-incidence matrix of the graph B. This is the $J \times E$ matrix where each column corresponds to an edge $\{i, j\} \in E$, with the ith and jth entries equal to 1 and -1 respectively. With this notation, we can rewrite (2.0.18) as:

$$\min \frac{1}{J} \sum_J ||x_j||_1 \text{ subject to } A_p x = b_p \text{ and } \left(B^T \circ I\right) \bar{x} = 0 \tag{2.1.19}$$

where $\bar{x} = (x_1, x_2, \dots x_J)$, and $\circ$ is the Kronecker product.

In this form, the Lagrangian of the problem (2.1.19) can be written in the following way:

$$\begin{aligned} L\left(\bar{x}_1, \bar{x}_2; \lambda\right) = \frac{1}{J} \sum_{j \in C_1} ||x_j||_1 + \frac{1}{J} \sum_{j \in C_2} ||x_j||_1 + & \qquad (2.1.20) \\ \phi_1\left(\bar{x_1}, \lambda\right) + \phi_2\left(\bar{x_2}, \lambda\right) + & \qquad (2.1.21) \\ \rho \bar{x}_1^T \left(B_1 B_2^T \circ I_n\right) \bar{x}_2 & \qquad (2.1.22) \end{aligned}$$

where

$$\begin{aligned} \phi_i\left(\bar{x}_i, \lambda\right) &= \lambda^T \left(B_i^T \circ I_n\right) \bar{x}_i + \frac{\rho}{2} \left|\left|\left(B_i^T \circ I_n\right) \bar{x}_i\right|\right|^2 & (2.1.23) \\ &= \left(\left(B_i^T \circ I_n\right) \lambda\right)^T \bar{x}_i + \frac{\rho}{2} \bar{x}_i^T \left(B_1 B_2^T \circ I_n\right) \bar{x}_i & (2.1.24) \end{aligned}$$

Since, no nodes within C_i are neighbours $B_i B_i^T$ is a diagonal matrix (with the degree of node i as the ith entry). So, we can finally re-write

$$\phi_i\left(\bar{x}_i,\lambda\right)=\sum_{j\in C_i}\left(\left(\sum_{k\in\mathcal{N}_i} sign\left(k-i\right)\lambda_{\{i,k\}}\right)^T x_i+\frac{\rho}{2}D_i\left|\left|x_i\right|\right|^2\right) \tag{2.1.25}$$

where the dual variable λ has been decomposed edge-wise so that $\lambda_{\{i,j\}}=\lambda_{\{j,i\}}$ and is associated with edge $\{i,j\}$ and the constraint $x_i=x_j$.

3 Consensus

Suppose we want to solve a problem such as:

$$\underset{x}{\text{minimize}} \quad \sum_i f_i\left(x\right)$$

this could arise in statistical computing where f_i would be the loss function for the $i^t h$ block of training data. We can write the problem for distributed optimisation as:

$$\begin{aligned} \underset{x}{\text{minimize}} &\quad \sum_i f_i\left(x_i\right) \\ \text{subject to} &\quad x_i-z=0 \end{aligned}$$

where x_i are local variables (for example local to each node in a spectrum sensing) and $x_i-z=0$ are the consensus constraints. Consensus and regularisation can be achieved by adding a regularisation term $g\left(z\right)$ - for example $g\left(z\right)=\lambda||x||_1$ corresponds to the LASSO, and the f_i would be $f_i=||A_ix_i-b||_2^2$.

As per the previous sections, we form the Augmented Lagrangian:

$$L_\rho\left(x,y\right)=\sum_i^n\left(f_i\left(x_i\right)+y_i^T\left(x_i-z\right)+\frac{\rho}{2}\|x_i-z\|_2^2\right) \tag{3.0.26}$$

The ADMM iterations for this Lagrangian are:

$$x_i^{k+1}:=\arg\min x_i\left(f_i\left(x_i\right)+y_i^{kT}\left(x_i-z\right)+\frac{\rho}{2}\|x_i-z\|_2^2\right) \tag{3.0.27}$$

$$z^{k+1}:=\frac{1}{n}\sum_i^n\left(x_i^{k+1}+\left(1\rho\right)y_i^k\right) \tag{3.0.28}$$

$$y_i^{k+1}:=y_i^k+\rho\left(x_i^{k+1}-z^{k+1}\right) \tag{3.0.29}$$

The z^{k+1} iteration is analytic as we're minimising the squared norm of x_i-z - so we average. With $\|x\|_1$ regularisation we perform soft-thresholding after the z update.

At each iteration the sum of the dual variables y_i is zero, so the algorithm can be simplified to:

$$x_i^{k+1}:=\arg\min x_i\left(f_i\left(x_i\right)+y_i^{kT}\left(x_i-\bar{x}^k\right)+\frac{\rho}{2}\|x_i-\bar{x}^k\|_2^2\right) \tag{3.0.30}$$

$$y_i^{k+1}:=y_i^k+\rho\left(x_i^{k+1}-z^{k+1}\right) \tag{3.0.31}$$

where

$$\bar{x}^k = \frac{1}{n}\sum_{i}^{n} x_i^k \tag{3.0.32}$$

This algorithm can be summarised as follows: in each iteration

- gather x^k and average to get $\bar{x}^k$
- scatter the average to nodes
- update y_i^k locally
- update x_i locally

Each agent is minimising it's own function, plus a quadratic term (the squared norm) which penalises the agent from moving too far from the previous average.

Note that the 'gather' stage doesn't require a central processor - this can be done in a distributed manner also.

3.1 statistical interpretation

At each step k of the algorithm each agent is minimising it's own loss function, plus a quadratic. This has a simple interpretation: we're doing MAP estimation under the prior $\mathcal{N}\left(\bar{x}^k + (1\rho)\, y_i^k, \rho I\right)$. I.e. the prior mean is the previous iteration's consensus shifted by node i disagreeing with the previous consensus.

I think what's going on is that after some (large) number of steps $k*$ we can replace the prior iteration mean with the 'optimal' mean i.e. $|x_i^{k*} - \bar{x}*| \leq \varepsilon$, and taking a convex combination of these $x_j^{k*} \forall j \in \{1 \dots n\}$ we get a slightly better estimate because for any finite k large enough not all the x_j will be the same.

I think that when we add regularisation, we change the prior to a Laplace prior, with parameters corresponding to the previous iteration.