Algorithms for constrained optimization

There are many, many constrained optimization algorithms, each tuned to the particulars of different classes of problems. We will look at the basics that underlie some of the more modern techniques. We will see that the concept of duality both helps us understand how these algorithms work, and gives us a way of determining when we are close to the solution. We will describe several techniques. Nearly all of these ultimately work by replacing the constrained program with an unconstrained program (or a series of unconstrained programs).

Eliminating equality constraints

The first approach is not so much an algorithm as a "trick" that lets us sometimes avoid even thinking about the constraints. Convex optimization problems with equality constraints[1] can always be written as optimization problems without equality constraints, and if there are no inequality constraints then the new program is unconstrained. To see this, suppose we are solving

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}.$$

Note that we can decompose any feasible $\boldsymbol{x}$ as $\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{h}$, where $\boldsymbol{x}_0$ satisfies $\boldsymbol{A}\boldsymbol{x}_0 = \boldsymbol{b}$ and $\boldsymbol{h} \in \text{Null}(\boldsymbol{A})$. Suppose that $\text{Null}(\boldsymbol{A})$ is K-dimensional and has basis $\boldsymbol{Q}$. Then we can write any $\boldsymbol{h} \in \text{Null}(\boldsymbol{A})$ as $\boldsymbol{h} = \boldsymbol{Q}\boldsymbol{w}$ and can re-write the constrained problem as

$$\underset{\boldsymbol{w}\in\mathbb{R}^K}{\text{minimize}} \;\; f(\boldsymbol{x}_0 + \boldsymbol{Q}\boldsymbol{w}).$$

Sometimes this method can be very helpful, but note that computing $\boldsymbol{x}_0$ and $\boldsymbol{Q}$ could potentially be expensive.

[1] Note that, in order for the optimization problem to be convex, any equality constraints must be *affine*, i.e., of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.

Projected gradient descent

Now suppose that we wish to solve the

$$\underset{\boldsymbol{x}\in\mathcal{C}}{\text{minimize}} \;\; f(\boldsymbol{x})$$

where f is a differentiable convex function and $\mathcal{C}$ is a convex set in $\mathbb{R}^N$. Another way to express this is as the unconstrained problem

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) + I_{\mathcal{C}}(\boldsymbol{x}), \tag{1}$$

where

$$I_{\mathcal{C}} = \begin{cases} 0 & \text{if } \boldsymbol{x} \in \mathcal{C}, \\ \infty & \text{otherwise}, \end{cases}$$

denotes the **indicator function** for the set $\mathcal{C}$. This wouldn't seem to do very much for us – since $I_{\mathcal{C}}(\boldsymbol{x})$ is non-differentiable, we cannot apply gradient-based methods to solving (1).

However, we have encountered some algorithms for minimizing nonsmooth convex functions. One that is particularly well-aligned with (1) is the **proximal gradient method**. Recall that proximal gradient method applies when our objective function consists of the sum of a smooth term (in this case, f) and a nonsmooth term (in this case, $I_{\mathcal{C}}$), resulting in the core iteration of

$$\boldsymbol{x}_{k+1} = \text{prox}_{\alpha_k I_{\mathcal{C}}}(\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)).$$

This would yield a tractable algorithm with provable guarantees *if* we can compute $\text{prox}_{\alpha_k I_{\mathcal{C}}}$. So what does $\text{prox}_{\alpha_k I_{\mathcal{C}}}$ look like? Note that

$$\begin{aligned} \text{prox}_{\alpha_k I_{\mathcal{C}}}(\boldsymbol{z}) &= \underset{\boldsymbol{x}\in\mathbb{R}^N}{\arg\min} \left(I_{\mathcal{C}}(\boldsymbol{x}) + \frac{1}{2\alpha_k}\|\boldsymbol{x}-\boldsymbol{z}\|_2^2 \right) \\ &= \underset{\boldsymbol{x}\in\mathcal{C}}{\arg\min} \;\; \|\boldsymbol{x}-\boldsymbol{z}\|_2^2 \\ &= \mathcal{P}_{\mathcal{C}}(\boldsymbol{z}), \end{aligned}$$

where $\mathcal{P}_{\mathcal{C}}(\boldsymbol{z})$ denotes the **projection** of $\boldsymbol{z}$ onto the set $\mathcal{C}$. Note that this holds for any $\alpha_k > 0$.

Thus, the core iteration of the proximal gradient method is equivalent to

$$\boldsymbol{x}_{k+1} = \mathcal{P}_{\mathcal{C}}(\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)).$$

That is, at each iteration we take a gradient step on f and then re-project onto the constraint set. Notice that for any $k \geq 1$, we are always guaranteed that $\boldsymbol{x}_k$ is feasible.

This algorithm is usually called **projected gradient descent**. It is a very simple (but often effective) method for solving constrained optimization problems when the projection onto the constraint set $\mathcal{C}$ can be computed efficiently. Note, however, that this is not always the case – sometimes computing this projection itself requires solving a challenging optimization problem.

Example: Least-squares with positivity constraints

Suppose we want to solve

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \ \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 \quad \text{subject to} \quad \boldsymbol{x} \geq \boldsymbol{0}.$$

This is a case where the projection onto the constraint set is relatively simple. It is easy to argue that the projection onto the set of all positive vectors is to simply just set all of the negative entries to zero. The projected gradient descent iteration is then

$$\boldsymbol{x}_{k+1} = \Big(\boldsymbol{x}_k + \alpha_k \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_k)\Big)_+,$$

where

$$(\boldsymbol{z})_+[i] = \begin{cases} z[i], & z[i] \geq 0, \\ 0, & z[i] < 0. \end{cases}$$

Barrier methods

Another popular and even more flexible approach are **barrier methods**. These can be thought of as again replacing our constrained problem with the unconstrained one in (1), but rather than attempting to minimize (1) directly, we instead solve a slight perturbation of this problem. In particular, we replace the indicator function $I_{\mathcal{C}}$ with some function b such that $b(\boldsymbol{x})$ is finite for any $\boldsymbol{x} \in \mathcal{C}$ and $b(\boldsymbol{x}) \to \infty$ as $\boldsymbol{x}$ approaches the boundary of $\mathcal{C}$.

To make this concrete, consider the constrained program

$$\underset{\boldsymbol{x}}{\text{minimize}} \;\; f(\boldsymbol{x}) \quad \text{subject to} \quad g_m(\boldsymbol{x}) \leq 0, \quad m = 1, \ldots, M.$$

In a barrier method we replace this with the unconstrained program

$$\underset{\boldsymbol{x}}{\text{minimize}} \;\; f(\boldsymbol{x}) + \sum_{m=1}^{M} B(g_m(\boldsymbol{x})),$$

where $B(x)$ is finite for $x < 0$ and $B(x) \to \infty$ as $x \to 0$ from the left. Again, unless B is the indicator function, the new program is an approximation to the original.

An interesting choice is $B(x) = -\frac{1}{\tau}\log(-x)$. This particular barrier function has the properties:

- You can analyze the number of Newton iterations needed for convergence for many f of interest.
- The solution[2] $\boldsymbol{x}^\star(\tau)$ of

$$\underset{\boldsymbol{x}}{\text{minimize}} \;\; \tau f(\boldsymbol{x}) - \sum_{m=1}^{M} \log(-g_m(\boldsymbol{x}))$$

[2] We have multiplied the objective by τ to make some of what follows a little easier to express.

can be used to generate a dual-feasible point, and hence a bound on how close we are to the solution.

To appreciate the second point above, we start by taking the gradient of the objective function above and setting it equal to zero. We see that

$$\tau \nabla f(\boldsymbol{x}^\star(\tau)) + \sum_{m=1}^{M} -\frac{1}{g_m(\boldsymbol{x}^\star(\tau))} \nabla g_m(\boldsymbol{x}^\star(\tau)) = \mathbf{0}.$$

So if we take

$$\lambda_m^\star(\tau) = -\frac{1}{\tau g_m(\boldsymbol{x}^\star(\tau))}, \ m = 1, \ldots, M,$$

we have $\lambda_m^\star(\tau) \geq 0$ and

$$f(\boldsymbol{x}^\star(\tau)) + \sum_{m=1}^{M} \lambda_m^\star(\tau) \nabla g_m(\boldsymbol{x}^\star(\tau)) = \mathbf{0}.$$

Since $\boldsymbol{x}^\star(\tau)$ is primal feasible, the only KKT condition we are missing is complementary slackness – we have replaced the condition

$$\lambda_m^\star g_m(\boldsymbol{x}^\star) = 0, \quad \text{with} \quad \lambda_m^\star(\tau) g_m(\boldsymbol{x}^\star(\tau)) = -1/\tau.$$

So if we set τ to be increasingly large, we obtain points that satisfy an increasingly tight approximation to the KKT conditions.

With this choice of $\boldsymbol{\lambda}^\star(\tau)$, we can also easily compute the dual of the original program:

$$\begin{aligned} d(\boldsymbol{\lambda}^\star(\tau)) &= \min_{\boldsymbol{x} \in \mathbb{R}^N} \left(f(\boldsymbol{x}) + \sum_{m=1}^{M} \lambda_m^\star(\tau) g_m(\boldsymbol{x}) \right) \\ &= f(\boldsymbol{x}^\star(\tau)) + \sum_{m=1}^{M} \lambda_m^\star(\tau) g_m(\boldsymbol{x}^\star(\tau)) \\ &= f(\boldsymbol{x}^\star(\tau)) - M/\tau. \end{aligned}$$

Hence, we know that if $\boldsymbol{x}^\star$ is a solution to the original program, then

$$f(\boldsymbol{x}^\star(\tau)) - f(\boldsymbol{x}^\star) \;\leq\; f(\boldsymbol{x}^\star(\tau)) - d(\boldsymbol{\lambda}^\star(\tau)) \;\leq\; M/\tau.$$

Thus, we know that solving the log-barrier problem gets us within M/τ of the optimal of the original primal objective.

One interesting theoretical result that we will not prove here is that, with a reasonable way of adjusting τ (multiplying it by 10 at every iteration, for example), the number of log-barrier iterations to make the value of the barrier functional $f(\boldsymbol{x}^\star(\tau))$ agree with the minimal value $f(\boldsymbol{x}^\star)$ to the original constrained problem to some precision. The upshot is that there is a very close match after $\sim \sqrt{M}$ iterations. This means that in theory, solving a constrained problem is roughly as expensive as solving $\sqrt{M}$ unconstrained problems. In practice, it is actually much cheaper – standard log barrier iterations take maybe 20–50 iterations to produce good results.

Primal dual interior point methods

These are closely related to log barrier algorithms, but they take a more direct approach towards "solving" the KKT conditions. The general idea is to treat the KKT conditions like a set of nonlinear equations, and solve them using Newton's method.

We start with the same set of relaxed KKT conditions[3] we used with

[3] We are again only considering inequality constraints; it is straightforward to modify everything we say here to include linear equality constraints.

log barrier:

$$\boldsymbol{r}_\tau(\boldsymbol{x}, \boldsymbol{\lambda}) = \begin{bmatrix} \nabla f(\boldsymbol{x}) + \sum_m \lambda_m \nabla g_m(\boldsymbol{x}) \\ -\lambda_1 g_1(\boldsymbol{x}) - 1/\tau \\ \vdots \\ -\lambda_m g_m(\boldsymbol{x}) - 1/\tau \end{bmatrix}$$

If we find $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ such that the $N+M$-vector $\boldsymbol{r}_\tau(\boldsymbol{x}, \boldsymbol{\lambda}) = \mathbf{0}$, then we know we have found the same $\boldsymbol{x}^\star(\tau)$, $\boldsymbol{\lambda}^\star(\tau)$ that solve the log barrier problem.

Primal-dual interior point methods take Newton steps to try to make $\boldsymbol{r}_\tau = \mathbf{0}$, but they adjust τ at every step. The Newton step is characterized by

$$\boldsymbol{r}_\tau(\boldsymbol{x} + \delta\boldsymbol{x}, \boldsymbol{\lambda} + \delta\boldsymbol{\lambda}) \;\approx\; \boldsymbol{r}_\tau(\boldsymbol{x}, \boldsymbol{\lambda}) + \boldsymbol{J}_{\boldsymbol{r}_\tau}(\boldsymbol{x}, \boldsymbol{\lambda}) \begin{bmatrix} \delta\boldsymbol{x} \\ \delta\boldsymbol{\lambda} \end{bmatrix} = \mathbf{0},$$

where $\boldsymbol{J}_{\boldsymbol{r}_\tau}(\boldsymbol{x}, \boldsymbol{\lambda})$ is the **Jacobian** matrix for the vector-valued function $\boldsymbol{r}_\tau$ given by the approximation

$$\begin{bmatrix} \nabla^2 f(\boldsymbol{x}) + \sum_m \lambda_m \nabla^2 g_m(\boldsymbol{x}) & \nabla g_1(\boldsymbol{x}) & \nabla g_2(\boldsymbol{x}) & \cdots & \nabla g_M(\boldsymbol{x}) \\ -\lambda_1 \nabla g_1(\boldsymbol{x})^{\mathrm{T}} & -\,g_1(\boldsymbol{x}) & 0 & & 0 \\ -\lambda_2 \nabla g_2(\boldsymbol{x})^{\mathrm{T}} & 0 & -g_2(\boldsymbol{x}) & & 0 \\ \vdots & & & \ddots & \vdots \\ -\lambda_M \nabla g_M(\boldsymbol{x})^{\mathrm{T}} & 0 & 0 & \cdots & -g_M(\boldsymbol{x}) \end{bmatrix}.$$

The update direction is

$$\begin{bmatrix} \delta\boldsymbol{x} \\ \delta\boldsymbol{\lambda} \end{bmatrix} = -\boldsymbol{J}_{\boldsymbol{r}_\tau}^{-1} \boldsymbol{r}_\tau(\boldsymbol{x}, \boldsymbol{\lambda}).$$

With this direction, we can perform a line search. The parameter τ is updated at every step (getting larger) based on an estimate of the duality gap.

A key feature of this type of primal dual method is that the iterates $\boldsymbol{x}_k$ and $\boldsymbol{\lambda}_k$ do not have to be feasible (although they of course become feasible in the limit).