Chapter 18: Constrained Nonlinear Programming

Unconstrained nonlinear programming is hard enough, but adding constraints makes it even more difficult. Any point in an unconstrained problem is feasible (though probably not optimal), but in constrained NLP a random point may not even be feasible because it violates one or more constraints! Look back at Reasons 3, 5, and 6 of Chapter 16 for specifics on how adding constraints complicates a nonlinear programming problem.

A recurring theme in NLP is trying to convert a difficult problem into something we already know how to solve. We saw this with gradient search where we turned the multi-dimensional unconstrained problem into a series of one-dimension unconstrained problems that we already knew how to solve. It's no different for handling constraints: the first thing we'll try is converting constrained problems to unconstrained problems that we already know how to solve. We'll need *penalty* and *barrier* methods for this.

It's important not to confuse these two. *Penalty methods* are used to find a feasible point (i.e. a point that satisfies all of the constraints simultaneously), and work by applying a penalty related to some measure of the amount of infeasibility. In other words they try to reduce the infeasibility to zero. *Barrier methods*, on the other hand, are used to prevent the infeasibility measure from growing, and are typically used when we already have a feasible point (at least relative to the inequality constraints). Think of it this way: penalty methods drive you into a feasible region, and barrier methods prevent you from leaving it (roughly speaking).

Penalty Methods to Find a Feasible Point

Penalty methods are normally used as a kind of nonlinear phase 1 whose goal is simply to find a feasible point, any feasible point. Bear in mind that the feasible regions may be disconnected, so a successful run of a penalty method will simply find a point in just one of the possibly many feasible regions in the NLP.

Penalty methods often ignore the objective function $f(\boldsymbol{x})$ entirely by replacing it with a new objective function that focuses entirely on violations of the constraints (both regular constraints and variable bounds). A definition of a *constraint violation* looks like this:

Constraint type	*Violation*
$g_i(\boldsymbol{x}) \leq b$	$max(0, g_i(\boldsymbol{x}) - b)$
$g_i(\boldsymbol{x}) \geq b$	$max\ (0, b - g_i(\boldsymbol{x}))$
$g_i(\boldsymbol{x}) = b$	$\lvert g_i(\boldsymbol{x}) - b \rvert$
Variable bound: $x_j \leq b$	$max\ (0, x_j - b)$
Variable bound: $x_j \geq b$	$max\ (0, b - x_j)$
Variable bound: $x_j = b$	$\lvert x_j - b \rvert$

Now if there are I constraints $f_i(\boldsymbol{x})$ for i=1...I and J variables x_j for j=1...J, and we define the violation as $v(g_i(\boldsymbol{x}))$ for some constraint i, and as $v(x_j)$ for the bound on some variable x_j, then some common penalty functions $p(\boldsymbol{x})$ are:

- Minimize the sum of the constraint violations:

$$Minimize\ p(\boldsymbol{x}) = \sum_{i=1}^{I} v\big(g_i(\boldsymbol{x})\big) + \sum_{j=1}^{J} v(x_j)$$

- Minimize the sum of the squared constraint violations:

$$Minimize\ p(\boldsymbol{x}) = \sum_{i=1}^{I} \big[v\big(g_i(\boldsymbol{x})\big)\big]^2 + \sum_{j=1}^{J} \big[v(x_j)\big]^2$$

Now we solve an unconstrained problem that has only the selected penalty function as the objective function. However to evaluate this unconstrained objective function value at a point we still have to inspect the constraint functions at that point to determine their contribution to the objective function if they are violated. The minimum value for either version of the penalty function is zero, and this is achieved only at a feasible point (where all constraints are satisfied).

It is common to add a nonnegative "strength" constant multiplier μ to the penalty function, so that it becomes, for example $Minimize\ \mu p(\boldsymbol{x}) = \mu \sum_{i=1}^{I} \big[v\big(g_i(\boldsymbol{x})\big)\big]^2 + \mu \sum_{j=1}^{J} \big[v(x_j)\big]^2$. It is also possible to combine the original objective function with the penalty function so that we seek both feasibility and optimality simultaneously. For example, if the original objective function $f(\boldsymbol{x})$ seeks to maximize, then the combined objective function is: *maximize* $F(\boldsymbol{x}) = f(\boldsymbol{x}) - \mu p(\boldsymbol{x})$. As you can see, any constraint violations worsen the value of $f(\boldsymbol{x})$ achieved at any particular point. The problem with the combined objective function is that it can lead you to a point that has a locally optimum value of $F(\boldsymbol{x})$, but which is not actually a feasible point. This happens where the value of $f(\boldsymbol{x})$ is especially attractive, but it is just a little infeasible, so the penalty does not sufficiently outweigh the attractiveness of the point. *Exact penalty methods* try to handle this problem by adjusting the μ multiplier so that any optimum point of F(x) is also an optimum point for the original constrained problem.

A natural question is how to solve the unconstrained model consisting of either just the penalty function $p(\boldsymbol{x})$ or the combined objective function $F(\boldsymbol{x})$ since they have incorporate discontinuous violation measures like $max(0, g_i(\boldsymbol{x}) - b)$ that are not differentiable. Of course pattern search methods can be used, but it would be nice to be able to use the more efficient gradient methods. As it happens, minimizing the squared version of the penalty function is differentiable: just use the right version of the violation function for the current point.

Barrier Methods

Barrier methods work differently for inequality and equality constraints. For inequality constraints (and variable bounds) they require a starting point that oversatisfies the constraint, e.g. if the constraint is $g(\boldsymbol{x}) \leq 10$, then a point at which $g(\boldsymbol{x}) = 9$ oversatisfies the constraint. The effect of the barrier method is then to prevent the point from moving onto the limiting value of the constraint (i.e. moving to $g(\boldsymbol{x}) = 10$ in this example). It does this by applying an increasing repulsive effect as the point moves nearer and nearer to the limiting value of the inequality. Think of it as an electric fence to keep sheep inside a field (which is the feasible region) and that

pushes on the sheep harder and harder as they approach the fence. This is why they are named "barrier methods": they provide a barrier to prevent escape from the feasible region.

One common form of the barrier function for a constraint of the form $g_i(\boldsymbol{x}) \leq b_i$ is

$$B(\boldsymbol{x}, r) = \frac{r}{b_i - g_i(\boldsymbol{x})}$$

where r is a nonnegative parameter that adjusts the strength of the barrier "resistance". You can see that this function works well as long as $g_i(\boldsymbol{x}) \leq b_i$, but if $g_i(\boldsymbol{x})=b_i$ then the function will blow up, and if $g_i(\boldsymbol{x}) \geq b_i$ it will give the wrong sign. Further, as $g_i(\boldsymbol{x})$ approaches b_i the size of the barrier function increases rapidly, until the blow-up at $g_i(\boldsymbol{x})=b_i$. This form works equally well for $\geq$ inequalities: just multiply the inequality by -1 and treat as $\leq$ type.

Simple variable bounds are just a special case of inequality constraints. The standard nonnegativity constraint of the form $x_j \geq 0$ has this barrier form: $B(\boldsymbol{x}, r) = \frac{r}{x_j}$.

But how do we find an initial point that oversatisfies the inequality constraints and variable bounds? You can use a penalty method for that.

Barrier methods have a different effect for equality constraints. Since it's very hard to find an initial point that satisfies a set of inequalities, barrier methods instead try to draw the initial point closer to exactly satisfying the equality constraint. A common form of the barrier function for an equality constraint $g_i(\boldsymbol{x})=b_i$ is:

$$B(\boldsymbol{x}, r) = \frac{\left(b_i - g_i(x)\right)^2}{\sqrt{r}}$$

Here, the effect is to reduce the size of the barrier function as $g_i(\boldsymbol{x})$ approaches b_i. So instead of pushing the point away as for an inequality constraint, the barrier function draws the point toward an equality constraint. Note that this effect increases in strength as r decreases in size.

Barrier methods are similar to penalty methods in that they convert the original constrained problem into an unconstrained problem, usually by combining the original objective function with a barrier function that stands in for the constraints. Then we can use the methods that we already have for unconstrained problems to solve this new version of the original model. The standard way to do this follows next.

Sequential Unconstrained Minimization (Maximization) Technique: SUMT

The *Sequential Unconstrained Minimization Technique* (or *Maximization* if that is what you are doing) is the usual way in which constrained problems are converted to an unconstrained form and solved that way. It makes use of barrier methods, and may use penalty methods as well to find a suitable initial point that oversatisfies the inequality constraints.

The main idea is to solve a *sequence* of unconstrained problems that differ only in the value of the resistance factor *r*. This factor will gradually be reduced, which has the effect of weakening the repulsive effect of the barrier function for the inequalities and strengthening its attractive effect for the equality constraints. Why do this? At first we want to make sure that the current

point does not stray out of the feasible region by crossing over any of the inequality constraints or variable bounds, so we start with a high repulsive force (i.e. a larger value of r). But on the other hand it is possible that the optimum solution is exactly on the limiting value of an inequality constraint, so we gradually reduce the repulsive effect of the inequalities to see if the solution point will move towards the inequality boundaries. At the same time, reducing the value of r increases the attractive effect of the equality constraints because we want to make sure that they are satisfied in the end. The rate of reduction in the value of r is controlled by a parameter θ which has a value between 0 and 1.

The steps of the SUMT algorithm are as follows:

0. Choose values for the parameters: (i) initial value of r, (ii) θ to control the rate of reduction of r, and (iii) ε, the acceptable error tolerance. Let k represent the overall iteration number, initially set at 0.
1. Find a suitable initial point $\boldsymbol{x}_0$ that oversatisfies the inequality constraints and variable bounds and is as close as possible to satisfying the equality constraints. Could use a penalty method for this.
2. Construct the barrier function $B(\boldsymbol{x},r)$ having one term for each constraint and variable bound. Combine this with the original objective function, e.g. minimize $f(\boldsymbol{x})$, to create the unconstrained combined objective function, e.g. *minimize* $F(\boldsymbol{x}) = f(\boldsymbol{x}) + B(\boldsymbol{x},r)$.
3. Using the starting point $\boldsymbol{x}_k$, find the optimum point $\boldsymbol{x}_{k+1}$ for $F(\boldsymbol{x})$ using a technique for unconstrained NLP, e.g. the gradient method.
4. Check the stopping conditions: if $|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k| \leq \boldsymbol{\varepsilon}$ then exit with $\boldsymbol{x}_{k+1}$ as the solution.
5. Reduce the value of r: $r \leftarrow r\theta$. Increment k: $k \leftarrow k+1$. Go to Step 3.

Let's look at an example reformulation for solution via SUMT. Here is the original constrained NLP model:

$$maximize\ f(\boldsymbol{x}) = x_1^4 - 3x_1x_2^3 + x_3^2 - 8$$

$$subject\ to: x_1 + x_2^2 \leq 18$$

$$x_3^2 + x_1 = 25$$

$$x_1 \geq 0, x_2 \geq 0, x_3\ unrestricted$$

The resulting combined objective will be to *maximize* $F(\boldsymbol{x},r) = f(\boldsymbol{x}) - B(\boldsymbol{x},r)$. Note that we subtract the barrier function in this case so it acts against the required maximization. The barrier function has terms for the inequality constraint, the equality constraint, and the two nonnegative variable bounds (note that x_3 is unrestricted), as follows:

$$B(\boldsymbol{x},r) = \frac{r}{18 - x_1 - x_2^2} + \frac{(25 - x_3^2 - x_1)^2}{\sqrt{r}} + \frac{r}{x_1} + \frac{r}{x_2}$$

So the resulting unconstrained combined objective function to use in the SUMT algorithm is:

$$maximize\ F(\boldsymbol{x}, r) = x_1^4 - 3x_1x_2^3 + x_3^2 - 8 - \left[\frac{r}{18 - x_1 - x_2^2} + \frac{(25 - x_3^2 - x_1)^2}{\sqrt{r}} + \frac{r}{x_1} + \frac{r}{x_2}\right]$$

Now let's look at a complete solution for the following model:

$$maximize\ f(x) = -x_1^2 - 4x_2^2 + 8x_1 + 16x_2$$
$$subject\ to: x_1 + x_2 \leq 5$$
$$x_1 \leq 3$$
$$x_1 \geq 0, x_2 \geq 0$$

Note that the objective function is identical to the unconstrained model that we solved on page 12 of Chapter 16. As we found there, the optimum solution when there are no constraints is *f*(4,2) = 32. But this solution is infeasible for the constrained version of the problem above: the point (4,2) violates the first two constraints. So now let's solve the constrained version of the problem using the SUMT method.

Step 0: Set the stopping error tolerance at ε=0.01, which is fairly large, but we choose it so that the calculations do not run through too many iterations in this example. We'll also choose the initial value of the repulsion factor as *r*=1.0, and the modification factor for *r* as θ=0.1, both typical values.

Step 1: We need an initial point $\boldsymbol{x}_0$ that oversatisfies the inequality constraints (i.e. satisfies them, but not right at their limiting values), and that is as close as possible to satisfying the equality constraints exactly. Sometimes it is easy to choose an initial point by inspection, e.g. in this case $\boldsymbol{x}_0$ = (1,1) will do. Otherwise you may need to run a penalty method on the constraints.

Step 2: The combined objective function is:

$$maximize\ F(\boldsymbol{x}, r) = -x_1^2 - 4x_2^2 + 8x_1 + 16x_2 - \left[\frac{r}{5 - x_1 - x_2} + \frac{r}{3 - x_1} + \frac{r}{x_1} + \frac{r}{x_2}\right]$$

Steps 3-5: Now we solve the unconstrained NLP specified in Step 2 using a gradient search algorithm. The results of several of the steps are summarized in the following table:

Iteration	*r*	*Start point*	*Gradient search solution*	*f(**x**)*	*F(**x**,r)*
1	1.0	$\boldsymbol{x}_0$=(1.00, 1.00)	$\boldsymbol{x}_1$=(2.31, 1.85)	29.054	25.44
2	0.1	$\boldsymbol{x}_1$=(2.31, 1.85)	$\boldsymbol{x}_2$= (2.75, 1.90)	30.398	29.62
...					
6	0.00001		$\boldsymbol{x}_6$= (~3.00, 1.97)	30.996	30.99
7	0.000001	$\boldsymbol{x}_6$= (~3.00, 1.97)	$\boldsymbol{x}_7$= (~3.00, 1.97)	30.996	30.99

As we see in the table, the gradient search finds different solutions as the strength parameter *r* is reduced. Eventually it terminates when the gradient search solutions are sufficiently similar in two successive solutions, in this case at around (3.00, 1.97) because of our relatively large stopping tolerance. This is close to the true optimum solution of (3.00, 2.00) that we would reach with a different choice of the stopping tolerance. Note that this is at the intersection of the two

constraints $x_1 + x_2 \leq 5$ and $x_1 \leq 3$. These are the two constraints that are violated when we optimize only the objective function without considering the constraints, as on page 12 of Chapter 16.

Note how much work this solution requires. We solve 7 unconstrained versions of the problem (at different values of r). Each of those unconstrained gradient solutions requires the solution of multiple one-dimensional optimization problems. Figure 18.1 shows the location of the first few points in the SUMT solution trajectory. You can see how the trajectory of the solution points will lead to the intersection of the two constraints.

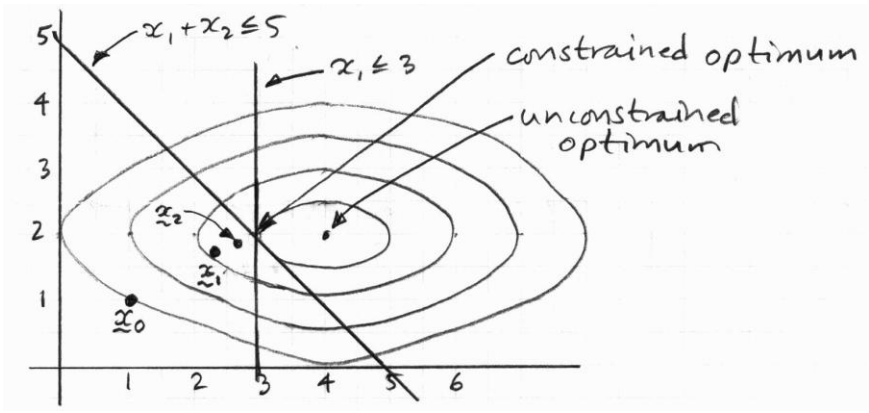

Figure 18.1: Some of the gradient solution end points in the overall SUMT solution.

Dealing with constraints in this way inevitably introduces some distortions, so it is not always possible to get a clean solution as we do in this small example. There are better ways to deal with constraints, as we'll see in the next few chapters.

Remember also that the SUMT method still only returns a local optimum. If you need to find a global optimum to a constrained NLP, then the only thing that you can do at this stage is to try a multistart approach and then take the best solution that you get as your estimated global optimum solution. But multistart is harder with a basic SUMT approach because of the special requirements of the initial $\boldsymbol{x}_0$ point, so you will likely have to couple it with a penalty method in a phase 1 procedure that will find a suitable $\boldsymbol{x}_0$.