Chapter 19: Handling Equality Constraints in NLP

Equality constraints are the most difficult to handle in nonlinear programming: how do you generate a point that lies exactly on a curved surface in a high dimensional space? Even worse, how to you find a point that lies exactly on *multiple* curved surfaces *simultaneously* in a high dimensional space? There are three main approaches:

- Analytic substitution of variables to eliminate equality constraints.
- Lagrange Multipliers
- The Generalized Reduced Gradient (GRG) method.

We'll concern ourselves with problems that have already been analytically reduced, so we'll only cover the second and third methods in the list above. As an aside, note that a significant amount of model simplification is commonly done by the presolve function of modern modelling systems such as AMPL, GAMS, etc.

Lagrange Multipliers

The method of Lagrange multipliers is applicable when the (i) the objective function is differentiable, and (ii) the constraints are all of the equality type. It relies on the following insight: if you have a single equality constraint, then you are at a local optimum point for the model when the gradient of the constraint at that point is *parallel* to the gradient of the objective function at that same point.

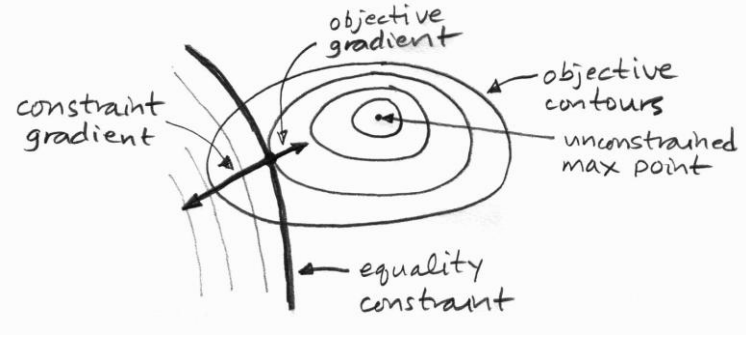

Figure 19.1: Parallel gradients in the method of Lagrange.

Fig. 19.1 illustrates this idea for a maximization objective. At the constrained optimum point on the equality constraint, the objective gradient points up the hill in the general direction of the unconstrained maximum point. At the same point, the gradient of the equality constraint points in exactly the opposite direction. Remember that an equality constraint is just a function like any other, hence you can also draw contour lines for it, and I have sketched a few in faintly. An equality constraint just fixes the function at a particular value, i.e. all solutions *must* be on one particular contour line. Recall also that gradient vectors always point in the direction of the maximum rate of *increase* of the function value. In Figure 19.1 it happens that the direction of increase in the LHS of the equality constraint is down and to the left in the diagram, so the gradient of the equality constraint points in that direction. In the diagram, the gradient of the objective function and the gradient of the equality constraint are parallel (or collinear). It happens that they have opposite directions, but

that doesn't matter. If the direction of increase of the equality function had been up and to the right in the diagram, then the two gradients would be on top of each other, with both pointing in the same direction.

As you can see in the diagram, if you move along the equality constraint even slightly you arrive at a worse value of the objective function. And at this new point, the objective and constraint gradients will no longer be parallel. The two are only "in balance" at a local optimum point. Think of the objective function as providing gravity in its gradient direction and the solution point as a marble rolling along a track provided by the equality constraint: the marble will come to rest at a local optimum point where the gradients are in balance. If the gradients aren't in balance, then the marble will roll some more.

Note that the two gradients at the local optimum can be in the same or opposite directions, as long as they are collinear, and can have different lengths. The relationship between the two gradients is expressed by a factor called the *Lagrange multiplier* (λ), which adjusts for direction (has positive or negative sign) and length.

When there are multiple equality constraints, a local optimum is found at a point where the objective function gradient is balanced by a linear combination of the gradients of the equality constraints. Suppose there are m independent equality constraints in n variables, where $m<n$ (otherwise the problem would have a single solution if $m=n$, or be overconstrained and have no solution if $m>n$):

$$\begin{gathered}\max or \min f(\boldsymbol{x}) \\ s.t.\ g_1(\boldsymbol{x}) = b_1 \\ g_2(\boldsymbol{x}) = b_2 \\ \vdots \\ g_m(\boldsymbol{x}) = b_m\end{gathered}$$

The first step is to convert the entire problem to the *Lagrange function* which incorporates the constraints:

$$h(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \sum_{i=1}^{m} \lambda_i [g_i(\boldsymbol{x}) - b_i]$$

where λ_i are the Lagrange multipliers, one for each constraint. Now we find the critical points of the Lagrange function by setting the partial derivatives to zero and solving simultaneously. The set of partial derivatives with respect first to the original variables in $\boldsymbol{x}$, and next with respect to the Lagrange multipliers $\boldsymbol{\lambda}$ is:

$$for\ j = 1 \ldots n\text{:}\ \frac{\delta h(\boldsymbol{x}, \boldsymbol{\lambda})}{\delta x_j} = \frac{\delta f(\boldsymbol{x})}{\delta x_j} - \sum\nolimits_{i=1}^{m} \lambda_i \frac{\delta g_i(\boldsymbol{x})}{\delta x_j} = 0 \quad (1)$$

$$for\ i = 1 \ldots m\text{:}\ \frac{\delta h(\boldsymbol{x}, \boldsymbol{\lambda})}{\delta \lambda_i} = -g_i(\boldsymbol{x}) + b_i = 0 \quad (2) \qquad (2)$$

Now here's the interesting thing: Equation (1) expresses the desired relationship that the gradient of the objective function is parallel to a linear combination of the gradients of the constraints,

where the λ_i are weighting factors related to the relative length and direction (given by the sign) of the equality constraint gradients. This is the condition we need to satisfy for local optimality. BUT we must also make sure to satisfy the original equality constraints, and this is exactly what Equation (2) expresses! So, if some point $(\boldsymbol{x},\boldsymbol{\lambda})$ is a critical point for the Lagrange function $h(\boldsymbol{x},\boldsymbol{\lambda})$, then it is a *feasible* critical point for the original model. But just a critical point: it could be a local maximum or a local minimum or even a saddle point for the original problem: you need to do some additional testing to determine which.

The three main steps in the method of Lagrange are:

1. Convert the model to the Lagrange function form.
2. Set all of the partial derivatives to zero and solve the resulting set of equations. Note that this can be a very difficult problem all by itself: the equations are likely to be nonlinear. You must apply a technique for solving sets of nonlinear equations, e.g. the Newton-Raphson method.
3. Test any critical points that are found to see if they are local maxima or minima for the original model. This may *also* be difficult. Recall that you can't simply check the Hessian of the objective function because this is a constrained problem, so you likely are not at an unconstrained local optimum of the objective function (which can be positively identified by the Hessian).

So there are several practical difficulties with the method of Lagrange: (i) you may not be able to solve the set of nonlinear equations in Step 2 because your numerical solution method fails, or because there is no solution, and (ii) deciding the optimality status of any point(s) output by Step 2.

Let's take a look at a simple example. I'm making it easy by using a quadratic objective function (so it's partial derivatives are linear) and a linear constraint. This makes Step 2 easy: all of the equations are linear. You won't normally be so lucky in real life. Here's the model:

$$\begin{gathered} maximize\ f(\boldsymbol{x}) = -2x_1^2 - x_2^2 + x_1x_2 + 8x_1 + 3x_2 \\ s.t.\ 3x_1 + x_2 = 10 \end{gathered}$$

So the Lagrange function is:

$$h(\boldsymbol{x},\lambda) = -2x_1^2 - x_2^2 + x_1x_2 + 8x_1 + 3x_2 - \lambda(3x_1 + x_2 - 10)$$

The resulting partial derivative equations are:

$$\begin{aligned} \frac{\delta h(\boldsymbol{x},\lambda)}{\delta x_1} &= -4x_1 + x_2 + 8 - 3\lambda = 0 \\ \frac{\delta h(\boldsymbol{x},\lambda)}{\delta x_2} &= -2x_2 + x_1 + 3 - \lambda = 0 \\ \frac{\delta h(\boldsymbol{x},\lambda)}{\delta \lambda} &= -3x_1 - x_2 + 10 = 0 \end{aligned}$$

So we have three equations in three unknowns to solve, and luckily they are linear, so it is straightforward to find the solution: (x_1, x_2, λ) = (2.464, 2.607, 0.250). So we have found a point

that is critical for $h(\boldsymbol{x},\lambda)$ and is both feasible and critical for the original constrained model. But how to test whether it is a local optimum for the original constrained model? In some cases it's obvious: if the objective function is concave and there is only a single Lagrange point returned in Step 2, then it must be a local maximum for the original constrained problem. In other cases you will need to find all of the Lagrange points and then choose the one that gives the largest value of $f(\boldsymbol{x})$ if maximizing, and the smallest value of $f(\boldsymbol{x})$ if minimizing.

Given all the difficulties associated with the method of Lagrange, why do we bother studying it? There are several reasons:

- It is a classical method that still finds frequent use because it is a good match for certain classes of problems.
- It actually applies to inequality constrained problems too: you just need to convert the inequalities to equality constraints by the addition of suitable slack and surplus variables.
- It is the basis of the Karush-Kuhn-Tucker conditions, which are vital for nonlinear programming. This is a set of conditions allowing for the identification of critical points in general nonlinear programming models, as we will see later.
- Finally, there is this theorem: if there is no $\boldsymbol{\lambda}$ at a point $\boldsymbol{x}$ such that $\nabla f(\boldsymbol{x}) - \sum_{i=1}^{m} \lambda_i \nabla g_i(\boldsymbol{x}) = 0$ then $\boldsymbol{x}$ cannot be a local optimum point. This means that if a point is proposed as a being a local optimum you can sometimes use this rule to prove that it is not.

The Generalized Reduced Gradient (GRG) Method

The main idea in the Generalized Reduced Gradient (GRG) method is that under some conditions it is possible to incorporate equality constraints directly into the objective function, thereby reducing the dimensionality of the model and converting it to an unconstrained form (and we know how to solve unconstrained problems: the gradient method comes to mind). Hence the problem has *reduced* dimensionality and can be solved by the *gradient* method. It is easiest to incorporate equality constraints into the objective function if they are linear. Here's a simple example:

$$\begin{aligned} min \quad & f(\boldsymbol{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 \\ s.t. \quad & g_1(\boldsymbol{x}) = 2x_1 + x_2 + 2x_3 + x_4 - 8 = 0 \\ & g_2(\boldsymbol{x}) = x_1 - x_2 + 4x_3 + x_4 + 2 = 0 \end{aligned}$$

Notice that the objective function is nonlinear while the two constraints are linear. We can use those two linear constraint equations to solve for two variables in terms of the other two:

$$x_2 = -0.5x_1 + x_3 + 5$$

$$x_4 = -1.5x_1 - 3x_3 + 3$$

Now we can substitute these two equations directly into the objective function to yield an equivalent unconstrained problem in only two dimensions (x_1 and x_3) instead of the original four:

$$minimize\ f(x_1, x_4) = x_1^2 + [-0.5x_1 + x_3 + 5]^2 + x_3^2 + [-1.5x_1 - 3x_3 + 3]^2$$

After we solve this unconstrained problem in x_1 and x_3 we just recover the values of x_2 and x_4 from the equations above.

But what do we do if the constraints are nonlinear? In that case we make linear approximations.

Making a Linear Approximation to a Nonlinear Function at a Given Point

Linear approximation is a standard and much-used technique in nonlinear programming. The linear approximation to some nonlinear function $g(\boldsymbol{x})$ is given by a truncated Taylor's Series expansion around the given point $\bar{\boldsymbol{x}}$:

$$g(\boldsymbol{x}) \approx g(\bar{\boldsymbol{x}}) + \nabla g(\bar{\boldsymbol{x}})^T(\boldsymbol{x} - \bar{\boldsymbol{x}})$$

Notice that the highlighted items $g(\bar{\boldsymbol{x}}), \nabla g(\bar{\boldsymbol{x}})^T$ and $\bar{\boldsymbol{x}}$ are fixed constants at the given point $\bar{\boldsymbol{x}}$. The accuracy of the approximation weakens as you move farther away from $\bar{\boldsymbol{x}}$ where the gradient of the function was evaluated. If the curve is highly nonlinear then the accuracy of the approximation degrades even faster.

Here is an example. Let's linearly approximate a function $g(\boldsymbol{x}) = x_1^2 + x_2 + 4x_3 + 4x_4 - 4$ around some point $\bar{\boldsymbol{x}}$. In expanded form, this is:

$$g(\boldsymbol{x}) \approx \bar{x}_1^2 + \bar{x}_2 + 4\bar{x}_3 + 4\bar{x}_4 - 4 + [2\bar{x}_1 \;\; 1 \;\; 4 \;\; 4] \begin{bmatrix} x_1 - \bar{x}_1 \\ x_2 - \bar{x}_2 \\ x_3 - \bar{x}_3 \\ x_4 - \bar{x}_4 \end{bmatrix}$$

$$\text{simplifies to } g(\boldsymbol{x}) \approx (2\bar{x}_1)x_1 + x_2 + 4x_3 + 4x_4 - (\bar{x}_1^2 + 4)$$

So at a particular point, say $\bar{\boldsymbol{x}} = (2,3,2,5)$, the linearization reduces to:

$$g(\boldsymbol{x}) \approx 4x_1 + x_2 + 4x_3 + 4x_4 - 8 = 0$$

Now suppose we want to find the value of the function at some nearby point (3,4,3,6). This particular version of the linearized function, we find that $g(\boldsymbol{x}) \approx 44$ where the exact value, using the original nonlinear function, is $g(\boldsymbol{x}) = 45$.

Steps in the Generalized Reduced Gradient Method

With that background, we can now take a look at the steps in the GRG algorithm. The main idea is to solve a *sequence* of subproblems, each using a new linearized approximation of the nonlinear constraints at the current point, and absorbing them into the objective function as per the algorithm.

0. Choose an initial point $\boldsymbol{x}$, tolerances and other parameter settings.
1. Linearly approximate the nonlinear constraints around the current point $\boldsymbol{x}$ and solve the resulting reduced unconstrained problem via a gradient method to find a new solution point $\boldsymbol{x}'$.
2. If $|\boldsymbol{x} - \boldsymbol{x}'|$ is small enough, then exit with $\boldsymbol{x}'$ as the solution.

3. $\boldsymbol{x} \leftarrow \boldsymbol{x}'$. Go to Step 1.

There are a few important algorithm details to discuss that must be clarified.

How do you numerically rewrite the linear (or linearly approximated) equality constraints to isolate a subset of the variables?

We don't want to have to do manual algebra to rewrite the linear equations in a form that isolates a subset of variables: we want this to be done algorithmically. This is actually easy to do using a technique borrowed from the simplex method for linear programming. The steps are as follows:

- Divide the variables into *basic* and *nonbasic* sets. It's the basic variables that will be substituted out of the objective function.
- Recall that there is 1 basic variable per constraint.
- Perform Gaussian elimination to get a +1 coefficient for the basic variable for each row, with zeros above and below it, just like putting the simplex tableau into proper form.
- This isolates the basic variables and allows them to be substituted into the objective function. Again we don't need to do this algebraically, this is taken care of algorithmically.

How do you solve the unconstrained subproblem?

Just like in the name of the method, it's usually some form of gradient method. But, technically, there's nothing to stop you from using, say, a pattern search method.

What if you don't have a feasible starting point?

In this case, you can set up and solve a phase 1 problem whose goal is to find a feasible point. It works like this:

- Choose a starting point. Of course a point that is as close as possible to feasible is best.
- Check which constraints are violated at this point. Satisfied constraints remain as phase 1 constraints. Violated constraints form the phase 1 objective, which may be to minimize the sum of the constraint violations, for example.
- Solve this problem via the GRG method.

When phase 1 terminates at a feasible point, switch to the usual full GRG for phase 2, which then retains feasibility.

How does GRG handle inequality constraints?

The first part is to convert all inequality constraints into equality constraints by the addition of suitable slack and surplus variables. The second part is a little harder. As the

algorithm proceeds, we must decide which inequalities are active (i.e. whose slack or surplus variables are equal to zero). There are heuristic *active set strategies* for making this decision, based on e.g. how close a constraint is to being satisfied at the current point.

How are variable bounds handled?

These are respected at all times. This isn't hard to do: you simply make sure not to exceed any variable bounds during the gradient method line searches.

There are a number of commercial implementations of the GRG method, including codes by the unsurprising names of GRG, GRG2 and LSGRG (for "large-scale" GRG), many by Lasdon and Waren. One of their GRG codes is actually embedded in Microsoft Excel as the nonlinear solver.

Linearization Schemes in Two Commercial Codes

Linear approximation is frequently a component of nonlinear solvers. It's interesting to see how it is used in completely different ways in two commercial NLP solvers.

LSGRG is (as might be expected) an implementation of the GRG algorithm for large-scale problems. It is a *feasible-path* method, meaning that it requires a feasible point (hence may employ a phase 1 to find a suitable starting point), and maintains feasibility as it proceeds. Technically it stays *close* to feasibility since it uses approximations to the nonlinear constraints which may violate true feasibility by a small amount. It works as described previously, by choosing an active set of constraints at each iteration, which are linearized and incorporated into the objective function. The subproblem is solved via a gradient method, but feasibility is retained by checking each new point. The cycle of choosing constraints, linearizing, solving, and checking is repeated until the stopping conditions are met.

MINOS stands for "Modular In-core Nonlinear Optimization System" and was quite a popular solver for many years, though it has since been superseded. It was extensively used for solving linear programming problems, as well as nonlinear problems, because of its excellent embedded LP solver. The LP solver is much used in its NLP solution approach, which had four stages:

- *Phase 1*: Linearize the nonlinear constraints at the current point and seek a feasible solution to the resulting LP. Since the nonlinear constraints curve, the linear approximations to them vary quite a bit depending on the linearization point, so there is no guarantee that the linearized version of the problem will have a feasible LP solution, even if there is a feasible solution for the nonlinear model. So MINOS incorporated a heuristic for improving the chances of finding a feasible solution for the LP: it would loosen the constraints a certain amount (by adjusting the right hand side constant to make it easier to solve) and try again. It would try this up to five times before giving up and declaring the model to be infeasible.
- *Phase 2*: Find the optimum point for LP established during Phase 1.
- *Phase 3*: Choose the active constraints at the current point, linearize them, and incorporate into the GRG objective function.

- *Phase 4*: Solve the GRG problem. If the stopping conditions are met, then exit with the current $\boldsymbol{x}$ as the solution, otherwise go to Phase 1.

So both solvers use the GRG algorithm, and hence must use linearization. But they do so in somewhat different ways. It's interesting to see how the basic ingredients of an algorithm can be combined to produced different solvers.