Chapter 20: Function and Region Shapes, the Karush-Kuhn-Tucker (KKT) Conditions, and Quadratic Programming

Function and Region Shapes

As we saw in Chapter 16, nonlinear programming is much harder than linear programming because the functions can take many different shapes: the objective function may have many hills and valleys, and the constraints can interact in ways that give disconnected feasible regions. There are some fundamental definitions for nonlinear function and region shapes; we'll look at these next.

First, definitions that apply to an individual function, whether it is an objective function or a constraint. Take any two points and connect them via an interpolating line. A function is convex if every point on every interpolating line has an interpolated value that is greater than or equal to the function value at the same point. A function is concave if every point on every interpolating line has an interpolated value that is less than or equal to the function value at the same point. These ideas are illustrated in Figure 20.1. Note that a linear function is both convex and concave.

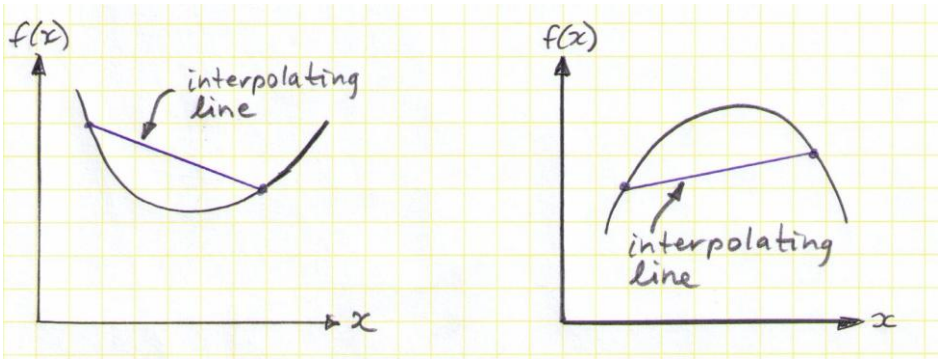

Figure 20.2: A convex function (left) and a concave function (right).

As you can see from the figure, a convex function will have a single local minimum and a concave function will have a single local maximum. This concept extends into multiple dimensions as well, so it corresponds directly with the ideas of positive definiteness of the Hessian matrix (convex function), and negative definiteness (concave function).

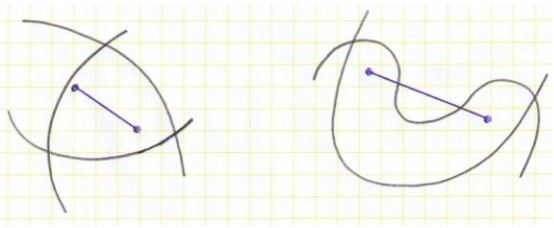

Figure 10.2: A convex set of points (left), and a nonconvex set (right).

We also need definitions that apply to the spaces defined by interacting sets of constraints. The most important concept is a *convex set of points*, defined as follows: all points on any line connecting any two points in the convex set also lie in the convex set. The opposite of a convex set of points is called a *nonconvex set* in which some of the points on the line segment lie outside the set of points. These ideas are illustrated in Fig. 20.2. In a convex set, every point can "see" every other point, but in a nonconvex set there are places where a given point cannot see every other point in the set. This can seriously affect how well a solver works. For example, a steepest descent gradient method

may try to move from the current point in the gradient direction, but encounter a constraint which prevents further improvement, even though it could reach a better feasible point if it could only "get around the corner" in the nonconvex set. This doesn't happen when the constraints form a convex set.

Knowledge of the shapes of the functions and the point sets created by the constraints is very useful in selecting the solution algorithm to use. We'll have more to say on this later, but for now observe that when minimizing a convex objective function subject to a convex set of constraints, there is exactly one local minimum, and there is exactly one local maximum when maximizing a concave function subject to a convex set of constraints. In both cases, this single local optimum is by definition also the global optimum.

It's difficult to analytically determine the shapes of functions beyond quadratics (which have a Hessian vector filled with constants), and there are very few software tools for this task either. One approach is to try many random line segments and look at the differences between the interpolated values of the function at the end points and the actual function value at some intermediate points. This is the approach taken by the MProbe software (http://www.sce.carleton.ca/faculty/chinneck/mprobe.html) mentioned in Chapter 16. Another approach is to try to break down a complex function into elemental parts for which the convexity/concavity is known or easily found. There are rules governing the shape of a function composed of convex or concave parts (e.g. the sum of convex functions is itself a convex function), so you can often reason out the shape of the complete function. This is the approach taken by the Dr. AMPL software (http://www.gerad.ca/~orban/drampl/) which operates on nonlinear models expressed in the AMPL modelling language. The final approach is the opposite: building functions from elemental parts in such a way that the resulting function is guaranteed to be convex. This is the approach taken in the CVX software (http://cvxr.com/cvx/).

The Karush-Kuhn-Tucker (KKT) conditions

Thus far we have a small set of methods to use for solving constrained or unconstrained nonlinear programming problems to find local optima. In practice these will always be implemented in computer software which will read a model in some form, calculate for a while, and eventually produce some output. But there is wide variety of possible outputs, some of which are hard to interpret. What does it mean if the solver says that it has made no progress for 25 iterations? Does that mean that it really is at a local optimum? Or does it mean that it is not looking in a promising area?

The best possible output message is one that definitely declares that a local optimum has been found. But how does the solver know this? Typically it is because it has checked the Karush-Kuhn-Tucker (KKT) Conditions. The KKT conditions are used to test a point to determine whether or not it is a critical point in a constrained nonlinear program. They don't actually determine whether the point is a local *optimum*, just that it is a *critical point* (could be a local maximum, a local minimum, or a saddle point). However since the solver is working towards a particular objective (maximization or minimization), it's a good bet that a point that satisfies the KKT conditions is a local optimum of the type sought by the solver.

The KKT conditions recognize that there are two different possibilities for a local optimum point:

A. *No constraints are active at the local optimum point.* In other words there is a local "hill" maximum (or "valley" minimum) in the objective function that is not near the limiting value of any constraint. In a case like this, the gradient will be zero ($\nabla f(\boldsymbol{x}) = \mathbf{0}$) at the local optimum point, and in fact the Hessian can be used to determine whether it is a local maximum, local minimum, or saddle point.

B. *At least one constraint is active at the local optimum point.* In this case some constraint is preventing the search method from improving the value of the objective function. At a point like this, the gradient of the objective function is *not* zero (i.e. $\nabla f(\boldsymbol{x}) \neq \mathbf{0}$) and the Hessian *cannot* be used to determine the optimality status and type of optimum.

The genius of the KKT conditions is that they handle *both* possibilities simultaneously. We'll get started with a simple observation.

Observation: In case B, we know how to use the method of Lagrange for equality constraints, so that will work if there are only equality constraints, but how do we deal with inequalities? This is actually pretty simple: given a point to test, you can determine whether an inequality is active (i.e. the point lies right on the limiting value of the inequality), in which case you can just treat it as if it were an equality constraint. If the inequality is inactive (i.e. it is oversatisfied), then you can just ignore it. To ignore an inactive inequality, just set its Lagrange multiplier to zero (i.e. $\lambda = 0$).

To make use of this observation, we proceed as follows:

- Always write the constraint equations so that the RHS constant is zero. For example a constraint such as $x_1^2 + 3x_2 \leq 5$ becomes $g(\boldsymbol{x}) = x_1^2 + 3x_2 - 5 \leq 0$.
- Now active constraints have value equal to zero, e.g. $g(\boldsymbol{x}) = 0$.
- Inactive constraints have a function value that is *not* equal to zero, e.g. $g(\boldsymbol{x}) = -4$ in the example above. But you can set the associated $\lambda = 0$ because the inequality is inactive.
- Therefore we always have $\lambda_i g_i(\boldsymbol{x}) = 0$ for i=1...m where there are m constraints. This covers both the active and the inactive constraints in one tidy package. Either $g_i(\boldsymbol{x}) = 0$ because the constraint is active, or $\lambda_i = 0$ because the inequality is inactive.

This is the *orthogonality condition*, expressed in matrix form as $\boldsymbol{\lambda}^T \boldsymbol{g}(\boldsymbol{x}) = \mathbf{0}$.

Let's now see if we can use these three elements to identify whether a given point is a critical point in an inequality-constrained NLP: (i) the Lagrange gradient condition, (ii) the orthogonality condition, and (iii) the original inequality constraints. We'll add equality constraints to the mix later. Let's express those three conditions for a system of m inequalities in n variables:

- *Lagrange gradient condition*: $\frac{\partial h(\boldsymbol{x},\boldsymbol{\lambda})}{\partial x_j} = 0$ for j=1...n.
- *Orthogonality condition*: $\lambda_i g_i(\boldsymbol{x}) = 0$ for i=1...m.
- *Original inequalities*: $g_i(\boldsymbol{x}) \leq 0$ for i=1...m.

At this point, recall that the Lagrange multiplier λ_i can be positive or negative for an equality constraint. This is because you cannot move off an equality constraint anyway, so it doesn't matter whether the increasing direction of the constraint gradient is in the same direction as the objective gradient (gives a positive multiplier), or in the opposite direction (gives a negative multiplier).

But the sign of λ_i *does* make a difference for active inequality constraints. But why? It's like this: just because the point you are testing makes the inequality active, it doesn't mean that the inequality is preventing you from improving the value of the objective function: it may be that moving towards the interior of the constraint will improve the objective function value. Let's imagine a situation in which you are standing right by a fence (the constraint) on the side of a hill (the objective function), and you would like to get to the top of the hill. The fence constraint is active because you are right by it. There are two possibilities: (*i*) The fence keeps you on the uphill side, in which case there is nothing to prevent you walking right to the top of the hill. In other words, you're at a point where the constraint is active, but it isn't preventing you from improving the value of the objective function. (*ii*) The fence constraint keeps you on the downhill side of the hill, in which case it is both active and *is* preventing you from going higher on the hill. We need to distinguish these two cases to determine whether a given point is a local optimum. In case (*i*) you are *not* at a local optimum, but in case (*ii*) you are. The *sign* of the Lagrange multiplier tells us whether we are in case (*i*) or case (*ii*).

How does this work? First, we require that all inequality constraints be put in the less-than-or-equal form, i.e. $g_i(\boldsymbol{x}) \leq 0$ for i=1...m. As always, the gradient of a function points in the increasing direction. So if a constraint of this form is active, then moving in its gradient direction is prevented: the constraint prevents $g_i(\boldsymbol{x})$ from getting any bigger. Now suppose we wish to maximize the objective function: this means we want to move in the objective function gradient direction. Hence if you are at some point and attempting to move in the objective gradient direction, but an inequality is active, and it's gradient is in the same direction as the objective gradient, then the Lagrange multiplier will be positive, and you will be in case (*ii*) above, i.e. at a local optimum. This concept is sketched in Fig. 20.3.

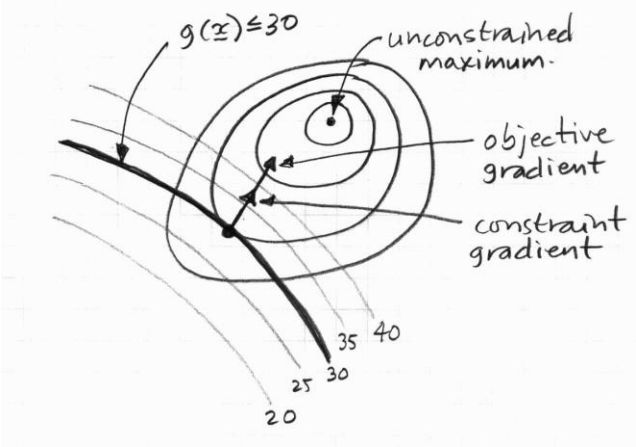

Figure 20.3: Lagrange multiplier nonnegativity condition.

Some gradient lines for the constraint are lightly sketched in Fig. 20.3, along with the value of the function at each of the lines. As you can see, moving in the constraint gradient direction is prevented by the constraint: it disallows values of $g(\boldsymbol{x})$ greater than 30. But it happens that the objective gradient is in the same direction: if we treat the inequality as an equality we will find that the Lagrange multiplier has a positive value.

If the gradient of the constraint pointed in the opposite direction to that shown in Fig. 20.3, moving down and to the left would be the prevented direction of

movement, so we could move up the objective hill to the unconstrained maximum point. In this case, treating the inequality as an equality yields a negative Lagrange multiplier.

So we see that an inequality constraint is only tight and "holding" the point when the Lagrange multiplier is positive (for an inequality of the form $g_i(\boldsymbol{x}) \leq 0$). This brings us to the *Lagrange multiplier nonnegativity condition* for inequality constraints (recall that the multiplier can be zero if the inequality is inactive).

The nonnegativity condition works as long as the inequality constraints satisfy a *constraint qualification*: the gradients of the constraints must be linearly independent at the point that is being tested.

Now we can summarize all four of the KKT conditions, as follows:

For a nonlinear program of the form:

$$\text{Max } f(\boldsymbol{x}) \text{ subject to } g_i(\boldsymbol{x}) \leq 0 \text{ for } i = 1 \ldots m$$

Where the Lagrangian function is:

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

The KKT conditions are:

$$\text{parallel gradients conditions:} \quad \frac{\partial h(\boldsymbol{x}, \boldsymbol{\lambda})}{\partial x_j} = 0 \text{ for } j = 1 \ldots n$$

$$\text{orthogonality conditions:} \quad \lambda_i g_i(\boldsymbol{x}) = 0 \text{ for } i = 1 \ldots m$$

$$\text{satisfaction of original constraints:} \quad g_i(\boldsymbol{x}) \leq 0 \text{ for } i = 1 \ldots m$$

$$\text{Lagrange multiplier nonnegativity:} \quad \lambda_i \geq 0 \text{ for } i = 1 \ldots m$$

But what if equality constraints are also included in the model? This is easily handled: now we don't care what sign the Lagrange multiplier is because you can't move off an equality constraint in either the gradient or the anti-gradient direction, so just remove the requirement that $\lambda_i \geq 0$ for each equality constraint. The orthogonality constraint is also not needed for an equality constraint because we already have $g_i(\boldsymbol{x}) = 0$, so the orthogonality condition is always satisfied.

What if you are dealing with a minimization instead of maximization? The easy route is to just multiply the objective function by -1 to convert it to a maximization and then apply the conditions listed above. As an alternative, you can keep the minimization objective and replace the Lagrange multiplier nonnegativity conditions by *nonpositivity* conditions.

Example: You have the following NLP model to solve:

$$\text{maximize } f(\boldsymbol{x}) = 10x_1^2 - x_2^2 + x_1 x_2$$
$$\text{subject to: } x_1 - x_2^2 \geq 5$$
$$x_1, x_2 \text{ unrestricted}$$

Your solver has returned the point (9,2), and you want to use the KKT conditions to determine whether this is likely a local maximum point.

The first thing to do is to convert the constraint to the required form, in this case:

$$-x_1 + x_2^2 + 5 \leq 0$$

So the Lagrange function is $h(\boldsymbol{x}, \lambda) = 10x_1^2 - x_2^2 + x_1x_2 - \lambda[-x_1 + x_2^2 + 5]$.

So the KKT conditions for this model are:

Parallel gradients: $\frac{\partial h(\boldsymbol{x},\boldsymbol{\lambda})}{\partial x_1} = 20x_1 + x_2 + \lambda = 0$ and $\frac{\partial h(\boldsymbol{x},\boldsymbol{\lambda})}{\partial x_2} = -2x_2 + x_1 - 2\lambda x_2 = 0$

Orthogonality: $\lambda(-x_1 + x_2^2 + 5) = 0$

Constraint satisfaction: $-x_1 + x_2^2 + 5 \leq 0$

Multiplier nonnegativity: $\lambda \geq 0$

Now let's check these conditions at the point (9,2) returned by the solver. We see immediately that the first parallel gradient condition is only satisfied if $\lambda < 0$, but this contradicts the multiplier nonnegativity condition. Thus we must conclude that the point is *not* a local maximum point. Note though that the constraint is tight at (9,2).

It is sometimes difficult to determine whether the conditions can be satisfied at a given point. The easy part is checking the original constraints to determine whether they are tight (in which case the Lagrange multiplier values must be found) or oversatisfied (in which case we know that the multipliers are zero because of the orthogonality condition). But then, just as for the method of Lagrange, you may need to solve a simultaneous set of possibly nonlinear equations, and that can be difficult. Still, the KKT conditions can be extremely helpful in assessing potential local optima.

Recall that satisfying the KKT conditions only indicates that the point is a critical point for the constrained NLP: there is no general guarantee that it is a maximum or minimum point. However we *can* guarantee that it is the global optimum point in two special cases, provided we know something about the shape of the objective function and the shape of the feasible region:

- *KKT point is a global maximum.* This is guaranteed if the objective function is globally concave and the constrained region is convex.
- *KKT point is a global minimum*. This is guaranteed if the objective function is globally convex and the constrained region is convex.

This happens because there is exactly one KKT point in each case, and it must be of the given type because that is the only type of critical point in the model.

Even better, they are used as the basis for quadratic programming, which allows a quadratic objective function subject to linear constraints, and for quadratically-constrained quadratic programming. This is because the derivatives of the Lagrange function (i.e. the parallel gradients

conditions) result in linear equations for quadratic functions. Hence we can essentially set up a linear set of equations to solve a quadratic program. More on this below.

An aside: The KKT conditions were known as the Kuhn-Tucker conditions for many years, following the publication of a paper by Kuhn and Tucker describing them in 1951. Many years later it was noticed that Karush had actually developed the same conditions much earlier in his master's thesis in 1939. But Karush did not publish any papers from his master's work, and so it was unnoticed for decades. The moral of the story for graduate students: publish your work!

Quadratic Programming

A quadratic program (QP) has a quadratic objective function (e.g. $Z = 12x_1^2 - 4x_1x_2 + 5x_3$) subject to linear constraints. Note that the quadratic nature of the objective function means that it can include squared terms, or terms that are the product of two variables, or linear terms. The simplest quadratic program is usually represented as follows:

$$\text{maximize } Z = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x}$$
$$\text{subject to } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$$
$$\boldsymbol{x} \geq \boldsymbol{0}$$

$\boldsymbol{A}$ is the usual $m \times n$ matrix for the linear constraints. $\boldsymbol{Q}$ is a symmetric $n \times n$ matrix. The $\boldsymbol{Q}$ matrix makes it easy to represent the second-order terms, which could involve any combination of two variables or the squaring of a single variable. Note that it is entirely possible to formulate the quadratic program without the ½ coefficient and with a $\boldsymbol{Q}$ matrix that is not symmetric: these just simplify some of the notation.

The ½ constant is required because of the way we construct $\boldsymbol{Q}$. In our example objective function $Z = 12x_1^2 - 4x_1x_2 + 5x_3$, we set $\boldsymbol{Q} = \begin{bmatrix} 24 & -4 & 0 \\ -4 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, constructed as follows:

- For squared terms like cx_i^2, write $2c$ in the ii position of $\boldsymbol{Q}$. So the $12x_1^2$ term in our example objective function appears as a 24 in the 1-1 position in $\boldsymbol{Q}$.
- For quadratic terms involving a pair of variables like cx_ix_j, write c in both the ij position and the ji position in $\boldsymbol{Q}$. So the $-4x_1x_2$ term in our example objective function appears as two terms: −4 in the 1-2 position and −4 in the 2-1 position in $\boldsymbol{Q}$.
- Purely linear terms (like the $5x_3$ in our example) do not appear in $\boldsymbol{Q}$ at all: they appear in the standard linear part of the objective function: $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$.

As mentioned above, the Lagrange conditions will yield linear functions since they are the derivatives of a quadratic function, so we have an *almost* completely linear system to solve to find a solution to this nonlinear problem. We also know the linear constraints form a convex set. Finally, we know that if we are minimizing a convex objective or maximizing a concave objective, subject to a convex set of constraints, then a point satisfying the KKT conditions must be the global optimum point.

So let's set up the KKT conditions for our minimization problem. First we define the Lagrange function a little differently. We use different symbols for the Lagrange multipliers for the *m* row constraints (λ_i, i=1...m) and for the *n* nonnegativity constraints (u_j, j=1...n). And we also need to convert the constraints to the $g_i(\boldsymbol{x}) \leq 0$ form needed for the KKT conditions: $\boldsymbol{A}_i(\boldsymbol{x})$ is the ith row of the $\boldsymbol{A}$ matrix, so the ith row constraint has the form $\boldsymbol{A}_i(\boldsymbol{x}) - b_i \leq 0$, and the jth nonnegativity constraint has the form $-x_j \leq 0$. With those adjustments, the Lagrange function can be written as:

$$h(\boldsymbol{x}, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} - \sum_{i=1}^{m} \lambda_i \left[\boldsymbol{A}_i(\boldsymbol{x}) - b_i\right] - \sum_{j=1}^{n} u_j[-x_j]$$

Now we can list the KKT conditions:

$$\text{parallel gradients conditions: } \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^T - \boldsymbol{\lambda}^T\boldsymbol{A} + \boldsymbol{u}^T = 0$$

$$\text{orthogonality conditions: } \begin{cases} \lambda_i[\boldsymbol{A}_i(\boldsymbol{x}) - b_i] = 0 \text{ for } i = 1 \ldots m \\ u_j x_j = 0 \text{ for } j = 1 \ldots n \end{cases}$$

$$\text{satisfaction of original constraints: } \begin{cases} \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ \boldsymbol{x} \geq \boldsymbol{0} \end{cases}$$

$$\text{Lagrange multiplier nonegativity: } \begin{cases} \boldsymbol{\lambda} \geq \boldsymbol{0} \\ \boldsymbol{u} \geq \boldsymbol{0} \end{cases}$$

As you can see, all of the conditions are linear *except* the orthogonality conditions, so it's *almost* linear, temptingly close to a possible solution via phase 1 linear programming where the variables are $\boldsymbol{x}$, $\boldsymbol{\lambda}$, and $\boldsymbol{u}$. Fortunately, there is a simple workaround to deal with the nonlinear constraints: adjust the LP pivoting rules so that we always satisfy the orthogonality conditions. We just make our pivots in a way that makes sure that (*i*) the slack variable for $\boldsymbol{A}_i(\boldsymbol{x}) \leq b_i$ is not basic at the same time as λ_i, and (*ii*) x_j is not basic at the same time as u_i. So we end up using something very much like linear programming to solve quadratic programs. In fact, most commercial LP solvers include the ability to solve quadratic programs since much of the main linear algebra machinery can be put to this purpose as well. However this LP-like approach does not apply to all QP forms: see below.

As is usual for the KKT conditions, things are a little easier when there are equality constraints: the corresponding nonnegativity constraint on the Lagrange multiplier is omitted, as is the orthogonality condition. So if row constraint *i* is an equality, just omit $\lambda_i \geq 0$ and omit $\lambda_i\boldsymbol{A}_i(\boldsymbol{x}) = 0$.

It is straightforward to check the shape of the quadratic objective function to determine whether it is concave, convex, etc. This is because the Hessian matrix (of second partial derivatives) is just equal to $\frac{1}{2}\boldsymbol{Q}$, which consists entirely of constants in a quadratic model. If it is negative definite or negative semidefinite, then the objective function is concave and so it is guaranteed that any point satisfying the KKT conditions is a global maximum point. This is the form of QP that can be solved by commercial LP solvers.

However things can be more complicated if $\frac{1}{2}\boldsymbol{Q}$ is not negative definite or negative semidefinite. One approach is to make small modifications to $\boldsymbol{Q}$ which make it negative definite so that the

solution method described above can still be applied. But that doesn't always work, in which case you are back to solving a general nonlinear programming problem, albeit with some special structure. Since QPs that have only equality constraints are simpler (no orthogonality constraints, no nonnegativity constraints on the row Lagrange multipliers) one common approach is *active set methods*. Just as for GRG these methods try to decide which inequality constraints should be active at a given point, and then treats those as equalities, creating a version of the original QP consisting solely of equality constraints that is consequently easier to solve. The difficulty arises in deciding which inequalities are active at a particular point.

In general, very good solvers are now commercially available for quadratic programs of all types, including quadratically constrained quadratic programs.