Nonlinear Programming: Theory and Algorithms

Sirong Luo

Faculty of Statistics and Data Science
Shanghai University of Finance and Economics

Table of Contents

It is sometimes necessary to consider more general nonlinear programs than the ones we have already studied: linear, quadratic, or conic programs. We give a brief introduction to this vast topic, and we discuss an application to estimating a volatility surface.

Table of Contents

Consider a very general optimization problem of the form

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_j(\mathbf{x}) = b_j, \quad \text{for } j = 1, \ldots, m \\ & h_i(\mathbf{x}) \leq d_i, \quad \text{for } i = 1, \ldots, p, \end{aligned}$$

or the equivalent more concise form

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{x}) = \mathbf{b} \\ & \mathbf{h}(\mathbf{x}) \leq \mathbf{d}, \end{aligned} \tag{20.1}$$

where $f, g_j, h_i : \mathbb{R}^n \to \mathbb{R}$. In the special case when all functions f, g_j, h_i are linear, problem (20.1) is a linear program as discussed in Chapter 2. When some of the functions f, g_j, h_i are nonlinear, problem (20.1) is a *nonlinear program*.

Many practical problems are naturally formulated as nonlinear programs. We already saw quadratic programming and conic programming in earlier chapters. However, the family of problems that can be formulated as nonlinear programs is enormous, as many, if not most, imaginable kinds of constraints and objectives can be cast in terms of nonlinear functions. (For some noteworthy examples, see the exercises at the end of the chapter.)

The immense modeling power of nonlinear programming comes at a cost: unlike linear, quadratic, and conic programming, which have a solid theory and are solvable via a few algorithmic templates, both the theory and methods to solve general nonlinear programs are far more complicated. Different types of nonlinear programs, determined by structural properties of the objective and constraint functions, are amenable to different types of algorithms. The subsequent sections sketch the main theory and most popular algorithmic ideas. A comprehensive treatment of this vast topic is beyond the scope of this textbook. We refer the reader to the excellent references Bertsekas (1999), Güler (2010), and Nocedal and Wright (2006) for more details.

Table of Contents

There are numerous software packages for solving nonlinear programs. The following are some popular ones. We list them according to the class of algorithms (discussed in Section 20.4) that they are based on: (1) CONOPT, GRG2, Excel SOLVER. These solvers are based on the *generalized reduced-gradient method*. (2) MATLAB optimization toolbox, SNOPT, NLPQL. These solvers are based on *sequential quadratic programming*. (3) MINOS, LANCELOT. These solvers are based on an *augmented Lagrangian approach*. (4) MOSEK, LOQO, IPOPT. These solvers are based on *interior-point methods*.

Table of Contents

In this section we consider the class of nonlinear programs (20.1) where the functions f, g_i, h_j are once or twice continuously differentiable. The optimality conditions for linear and convex quadratic programs extend to this more general context, albeit some new technicalities arise. In particular, for a general nonlinear program the theory described below applies to *local minima*.

Let $\mathcal{X} := \{\mathbf{x} \in \mathbb{R}^n : g(\mathbf{x}) = b, h(\mathbf{x}) \leq d\}$ denote the feasible set of (20.1). A point $\mathbf{x}^* \in \mathcal{X}$ is a *local minima* of (20.1) if there exists $r > 0$ such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \text{ for all } \mathbf{x} \in B_r(\mathbf{x}^*) \cap \mathcal{X},$$

where $B_r(\mathbf{x}^*)$ denotes the ball of radius r around $\mathbf{x}^*$; that is,

$$B_r(\mathbf{x}^*) := \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{x}^*\| \leq r\}.$$

A point $\mathbf{x}^* \in \mathcal{X}$ is a *strict local minima* of (20.1) if there exists $r > 0$ such that

$$f(\mathbf{x}^*) < f(\mathbf{x}) \text{ for all } \mathbf{x} \in B_r(\mathbf{x}^*) \cap \mathcal{X}.$$

Unconstrained Case

For ease of exposition we first describe the optimality conditions for the simpler case without constraints. Consider the unconstrained optimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}), \tag{20.2}$$

where $f : \mathbb{R}^n \to \mathbb{R}$.

Theorem 20.1 (First-order necessary conditions) *Suppose f is continuously differentiable. If a point $\mathbf{x}^* \in \mathbb{R}^n$ is a local minimum of (20.2) then $\nabla f(\mathbf{x}^*) = \mathbf{0}$.*

The above necessary conditions can be sharpened when the objective function is twice differentiable.

Theorem 20.2 (Second-order necessary and sufficient conditions) *Suppose f is twice continuously differentiable.*
(a) *If a point $\mathbf{x}^* \in \mathbb{R}^n$ is a local minimum of (20.2) then $\nabla f(\mathbf{x}^*) = \mathbf{0}$ and $\nabla^2 f(\mathbf{x}^*) \succeq \mathbf{0}$.*
(b) *If $\mathbf{x}^* \in \mathbb{R}^n$ is such that $\nabla f(\mathbf{x}^*) = \mathbf{0}$ and $\nabla^2 f(\mathbf{x}^*) \succ \mathbf{0}$, then $\mathbf{x}^*$ is a strict local minimum of (20.2).*

Constrained Case

Consider now the general constrained problem (20.1). The optimality conditions for (20.1) rely on the following technical condition.

Definition 20.3 Let $\mathbf{x} \in \mathcal{X}$. Define $I(\mathbf{x}) := \{i : h_i(\mathbf{x}) = d_i\}$. The point $\mathbf{x}$ satisfies the *linear independence constraint qualification* if the set of gradient vectors

$$\{\nabla g_j(\mathbf{x}) : j = 1, \ldots, m\} \cup \{\nabla h_i(\mathbf{x}) : i \in I(\mathbf{x})\}$$

is linearly independent.

Theorem 20.4 (First-order necessary conditions) *Suppose f, g_i, h_j are continuously differentiable. If a point $\mathbf{x}^* \in \mathcal{X}$ is a local minimum of (20.1) and satisfies the linear independence constraint qualification, then there exist some Lagrange multipliers $\mathbf{y} \in \mathbb{R}^m$ and $\mathbf{s} \in \mathbb{R}^p$ such that*

$$\begin{aligned} \nabla f(\mathbf{x}^*) + \sum_{j=1}^{m} y_j \nabla g_j(\mathbf{x}^*) + \sum_{i=1}^{p} s_i \nabla h_i(\mathbf{x}^*) &= \mathbf{0}, \\ \mathbf{s} &\geq \mathbf{0}, \\ s_i(h_i(\mathbf{x}^*) - d_i) &= 0, \quad \text{for } i = 1, \ldots, p. \end{aligned} \tag{20.3}$$

Observe that the first block of equations in (20.3) can be written as

$$\nabla_{\mathbf{x}} L(\mathbf{x}^*, \mathbf{y}, \mathbf{s}) = \nabla f(\mathbf{x}^*) + \nabla \mathbf{g}(\mathbf{x}^*)\mathbf{y} + \nabla \mathbf{h}(\mathbf{x}^*)\mathbf{s} = \mathbf{0},$$

where $L(\mathbf{x}, \mathbf{y}, \mathbf{s})$ is the following *Lagrangian function* for (20.1):

$$L(\mathbf{x}, \mathbf{y}, \mathbf{s}) := f(\mathbf{x}) + \mathbf{y}^\top(\mathbf{g}(\mathbf{x}) - \mathbf{b}) + \mathbf{s}^\top(\mathbf{h}(\mathbf{x}) - \mathbf{d}),$$

and where

$$\nabla \mathbf{g}(\mathbf{x}) = \begin{bmatrix} \nabla g_1(\mathbf{x}) & \cdots & \nabla g_m(\mathbf{x}) \end{bmatrix} \text{ and } \nabla \mathbf{h}(\mathbf{x}) = \begin{bmatrix} \nabla h_1(\mathbf{x}) & \cdots \nabla h_p(\mathbf{x}) \end{bmatrix}.$$

Observe the nice analogy to the first-order conditions for the unconstrained case.

Theorem 20.5 (Second-order necessary conditions) *Suppose f is twice continuously differentiable. If a point $\mathbf{x}^* \in \mathbb{R}^n$ is a local minimum of (20.2) and satisfies the linear independence constraint qualification, then there exist $\mathbf{y} \in \mathbb{R}^m$ and $\mathbf{s} \in \mathbb{R}^p$ such that (20.3) holds and*

$$\mathbf{d}^\top \left(\nabla^2_{\mathbf{x}} L(\mathbf{x}^*, \mathbf{y}, \mathbf{s}) \right) \mathbf{d} \geq 0$$

for all $\mathbf{d} \in T(\mathbf{x}^*)$.

Theorem 20.6 (Second-order sufficient conditions) *Suppose* f *is twice continuously differentiable and* $\mathbf{x}^* \in \mathbb{R}^n$ *satisfies the linear independence constraint qualification. If there exist* $\mathbf{y} \in \mathbb{R}^m$ *and* $\mathbf{s} \in \mathbb{R}^p$ *such that (20.3) holds as well as*

$$h_i(\mathbf{x}^*) = d_i \Rightarrow s_i > 0, \text{ and}$$

$$\mathbf{d}^\top \left(\nabla^2_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) \right) \mathbf{d} > 0$$

for all non-zero $\mathbf{d} \in T(\mathbf{x}^*)$ *, then* $\mathbf{x}^*$ *is a local minimum of (20.1).*

Convex Case

As we detail next, the optimality conditions described above simplify and strengthen substantially when the underlying problem is convex.

Proposition 20.7 *Assume the objective function f in (20.2) is convex and differentiable. Then $\mathbf{x}^*$ is an optimal solution to (20.2) if and only if $\nabla f(\mathbf{x}^*) = \mathbf{0}$.*

Proposition 20.8 *Assume the objective function f in (20.1) is convex, the equality constraint functions g_i are linear, and the inequality constraint functions h_j are convex. Furthermore, assume all f, g_i, h_j are differentiable. Then $\mathbf{x}^*$ is an optimal solution to (20.1) if and only if there exist $\mathbf{y} \in \mathbb{R}^m, \mathbf{s} \in \mathbb{R}^p$ such that*

$$\nabla_{\mathbf{x}} L(\mathbf{x}^*, \mathbf{y}, \mathbf{s}) = \mathbf{0}, \quad g(\mathbf{x}^*) = \mathbf{b}, \quad h(\mathbf{x}^*) \leq \mathbf{d}, \quad \mathbf{s} \geq \mathbf{0}, \quad \mathbf{s}^\top (h(\mathbf{x}^*) - \mathbf{d}) = 0.$$

Table of Contents

Unconstrained Case

We next describe three main algorithmic approaches to solving an unconstrained optimization problem of the form (20.2), namely the *gradient descent method*, *Newton's method*, and the *subgradient method*. These methods in turn provide the foundation for the more elaborate methods for solving constrained optimization problems.

Gradient Descent

Suppose the objective function f in (20.2) is differentiable. In this case a simple method for solving (20.2) is based on going downhill on the graph of the function f. The gradient gives the direction of fastest initial increase and thus its negative is the direction of fastest initial decrease. This can also be motivated by the first-order Taylor approximation of f around a point $\mathbf{x}$: for $\mathbf{p}$ small we have

$$f(\mathbf{x}+\mathbf{p}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \mathbf{p}.$$

Among all $\mathbf{p}$ of fixed norm, the one pointing in the direction $-\nabla f(\mathbf{x})$ minimizes the right-hand side.
Algorithm 20.1 gives a formal description of the gradient descent method.

Algorithm 20.1 Gradient descent method

1: choose $\mathbf{x}^0 \in \mathbb{R}^n$
2: **for** $k = 0, 1, \ldots$ **do**
3: choose a step length $\alpha_k > 0$ and set $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \nabla f(\mathbf{x}^k)$
4: **end for**

The choice of step length α is a critical detail in the implementation of the gradient descent algorithm. If α is too large, the algorithm may fail to converge to a solution because the objective value could even increase after one iteration. On the other hand, if α is too small, the algorithm will be too slow. This issue applies not only to the gradient descent method but to any method that aims to move along a direction $\mathbf{p}$.

Suppose $\mathbf{p} \in \mathbb{R}^n$ is a *descent direction* at the current point $\mathbf{x}^k$; that is, $\nabla f(\mathbf{x}^k)^\top \mathbf{p} < 0$. A popular approach is to choose the step length large enough and perform *backtracking*; that is, shrink α_k by a multiplicative constant smaller than one until the following sufficient decrease condition holds for some predetermined $\mu \in (0, 1)$:

$$f(\mathbf{x}^k + \alpha_k \mathbf{p}) \leq f(\mathbf{x}^k) + \alpha_k \cdot \mu \cdot \nabla f(\mathbf{x}^k)^\top \mathbf{p}. \tag{20.4}$$

The sufficient decrease requirement in (20.4) is called the *Armijo-Goldstein* condition. The first-order Taylor approximation for f around $\mathbf{x}$ ensures that (20.4) holds for sufficiently small α_k provided f is differentiable and $\mathbf{d}$ is a descent direction. Algorithm 20.2 describes this kind of backtracking. This type of backtracking is also often called *line search*.

Algorithm 20.2 Backtracking to select the step length α_k

1: choose $\alpha_k > 0$ and $\beta, \mu \in (0, 1)$
2: **while** (20.4) fails **do** $\alpha_k = \beta \cdot \alpha_k$
3: **end while**

The latter update can be motivated by considering the second-order Taylor approximation to f around $\mathbf{x}$:

$$f(\mathbf{x}+\mathbf{p}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \mathbf{p} + \frac{1}{2}\mathbf{p}^\top \nabla^2 f(\mathbf{x})\mathbf{p}.$$

Observe that when $\nabla^2 f(\mathbf{x}) \succ 0$ the Newton step $\mathbf{p} := -\nabla^2 f(\mathbf{x})^{-1}\nabla f(\mathbf{x})$ minimizes the right-hand side.

Newton's method also applies to solving nonlinear equations. Consider the system of nonlinear equations

$$F(\mathbf{x}) = \mathbf{0} \tag{20.5}$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ is a differentiable function. Let $F'(\mathbf{x})$ denote the *Jacobian matrix* of F, that is, the $n \times n$ matrix with (i, j) component

$$F'(\mathbf{x})_{ij} = \frac{\partial f_i(\mathbf{x})}{\partial x_j},$$

where $f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})$ are the components of $F(\mathbf{x})$.

In its pure form, Newton's method for (20.5) updates a trial point $\mathbf{x}$ to the new point

$$\mathbf{x}^+ = \mathbf{x} - F'(\mathbf{x})^{-1}F(\mathbf{x}).$$

The latter update can be motivated by considering the first-order Taylor approximation to F around $\mathbf{x}$:

$$F(\mathbf{x} + \mathbf{p}) \approx F(\mathbf{x}) + F'(\mathbf{x})\mathbf{p}.$$

The Newton step $\mathbf{p} = -F'(\mathbf{x})^{-1}F(\mathbf{x})$ makes the above right-hand side equal to zero.

Observe that Newton's method for the unconstrained optimization problem (20.2) is exactly the same as Newton's method for solving the system of nonlinear equations $\nabla f(\mathbf{x}) = \mathbf{0}$.

Newton's method has a much faster rate of convergence than gradient descent provided the initial iterate is sufficiently close to the solution. On the other hand, when the initial iterate is far from the solution, the above pure form of Newton's method may fail to converge. The latter drawback can be rectified by performing some backtracking along the Newton step direction as described in Algorithm 20.3. The step length α_k can be chosen via the backtracking procedure described in Algorithm 20.2 to ensure the Armijo–Goldstein sufficient decrease condition (20.4) holds. For the Newton direction $\mathbf{d} = -\nabla^2 f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}^k)$, a natural and customary initial step length at each step is $\alpha_k = 1$.

Algorithm 20.3 Newton's method with backtracking

choose $\mathbf{x}^0 \in \mathbb{R}^n$
for $k = 0, 1, \ldots$ **do**
    choose a step length $\alpha_k \in (0, 1]$ via backtracking and set $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \nabla^2 f(\mathbf{x}^k)^{-1} \nabla f(\mathbf{x}^k)$
end for

Subgradient Method

In the special case when the objective function f is convex, the gradient descent method can be extended to *non-smooth* functions; that is, functions that are not necessarily differentiable. Non-smooth functions arise often in optimization. In particular, the Lagrangian relaxation heuristic for (8.9) described in Section 8.3.3 yields the minimization of a non-smooth convex function.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. A point $\mathbf{g} \in \mathbb{R}^n$ is a *subgradient* of f at $\mathbf{x} \in \mathbb{R}^n$ if for all $\mathbf{y} \in \mathbb{R}^n$,

$$f(\mathbf{y}) - f(\mathbf{x}) \geq \mathbf{g}^\top (\mathbf{y} - \mathbf{x}).$$

The *subdifferential* of f at $\mathbf{x}$, denoted $\partial f(\mathbf{x})$, is the set of *subgradients* of f at $\mathbf{x}$.

The subdifferential of a convex function is non-empty at every point. The following example illustrates the subdifferential of a simple non-smooth function. Consider the convex function $f : \mathbb{R} \to \mathbb{R}$ defined by $f(x) = |x|$. In this case we have

$$\partial f(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ [-1, 1] & \text{if } x = 0. \end{cases}$$

Algorithm 20.4 describes the subgradient method for (20.2) when f is a convex function. Observe that it is a natural extension of Algorithm 20.1.

Algorithm 20.4 Subgradient method

1: choose $\mathbf{x}^0 \in \mathbb{R}^n$
2: **for** $k = 0, 1, \ldots$ **do**
3: choose $\mathbf{g}_k \in \partial f(\mathbf{x}^k)$ and a step length $\alpha_k > 0$, and set $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \mathbf{g}_k$
4: **end for**

For non-smooth functions, the choice of step length α_k for the subgradient method cannot be chosen via a backtracking procedure as the Armijo–Goldstein condition (20.4) cannot be guaranteed in the absence of differentiability. Various choices have been proposed in the literature. The following two generic types of step lengths are particularly simple and popular. The first one is to choose fixed sizes $\alpha_k = \alpha > 0$ for all k. The second one is to choose slowly diminishing sizes such that

$$\sum_{k=0}^{\infty} \alpha_k^2 < \infty,$$

$$\sum_{k=0}^{\infty} \alpha_k = \infty.$$

Constrained Case-Generalized Reduced Gradient

The main idea behind the *generalized reduced gradient* method is to reduce a constrained problem to a sequence of unconstrained problems in a space of lower dimension. To illustrate this procedure, consider the special case when the equality constraints are linear:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{s.t.} \mathbf{A}\mathbf{x} = \mathbf{b} \end{aligned} \tag{20.6}$$

for some $\mathbf{A} \in \mathbb{R}^{m \times n}$. Without loss of generality we may assume that $\mathbf{A}$ has full row rank as otherwise either some constraints are redundant or the problem is infeasible.

Since $\mathbf{A}$ has full rank, we can partition both $\mathbf{A}$ and $\mathbf{x}$ as follows:

$\mathbf{A} = \begin{bmatrix} \mathbf{A}_B & \mathbf{A}_N \end{bmatrix}$, and $\mathbf{x} = \begin{bmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{bmatrix}$ for some subset $B \subseteq \{1, \ldots, n\}$ such that $\mathbf{A}_B$ is non-singular. Therefore

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad \Leftrightarrow \quad \mathbf{A}_B\mathbf{x}_B + \mathbf{A}_N\mathbf{x}_N = \mathbf{b} \quad \Leftrightarrow \quad \mathbf{x}_B = \mathbf{A}_B^{-1}(\mathbf{b} - \mathbf{A}_N\mathbf{x}_N).$$

Consequently, problem (20.6) is equivalent to the following reduced space unconstrained minimization problem:

$$\min_{\mathbf{x}_N} \hat{f}(\mathbf{x}_N)$$

where

$$\hat{f}(\mathbf{x}_N) = f(\mathbf{A}_B^{-1}(\mathbf{b} - \mathbf{A}_N\mathbf{x}_N), \mathbf{x}_N).$$

Consider a more general program with nonlinear equality constraints:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{s.t. } g(\mathbf{x}) = \mathbf{b}. \end{aligned} \tag{20.7}$$

We can extend the above approach by approximating the nonlinear equality constraints with their first-order Taylor approximation. More precisely, suppose the current point is $\mathbf{x}^k$. Consider the modification of (20.7) obtained by replacing $g(\mathbf{x}) = \mathbf{b}$ with its first-order Taylor approximation

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{s.t. } g(\mathbf{x}^k) + \nabla g(\mathbf{x}^k)^\top (\mathbf{x} - \mathbf{x}^k) = \mathbf{b}. \end{aligned} \tag{20.8}$$

Observe that the latter problem is of the form (20.6) and is thus amenable to the type of reduced space approach described above. Algorithm 20.5 describes a template for a generalized reduced gradient approach to problem (20.7). The step length α at each iteration is typically chosen to balance both goals of objective function reduction and constraint satisfaction.

Algorithm 20.5 Generalized reduced gradient

choose $\mathbf{x}^0$
for $k = 0, 1, \ldots$ **do**
    solve the linearized constraints problem (20.8) to find a search direction $\Delta\mathbf{x}^k$
    choose a step length $\alpha > 0$ and set $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha\Delta\mathbf{x}^k$
end for

The generalized reduced gradient approach can be extended to deal with inequality constraints as well via an *active-set approach* like that discussed in Chapter 5. The basic idea is that the active inequalities can be treated as equality constraints. The challenge of course is to determine the correct set of active inequalities at the optimal solution.

Sequential Quadratic Programming

The central idea of *sequential quadratic programming* is to capitalize on algorithms for quadratic programming to solve more general nonlinear programming problems of the form (20.1).

Given a current iterate $\mathbf{x}^k$, problem (20.1) can be approximated with the following quadratic program:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}^k) + \nabla f(\mathbf{x}^k)^\top (\mathbf{x} - \mathbf{x}^k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^k)^\top \mathbf{B}_k (\mathbf{x} - \mathbf{x}^k) \\ \text{s.t.} \quad & g(\mathbf{x}^k) + \nabla g(\mathbf{x}^k)^\top (\mathbf{x} - \mathbf{x}^k) = \mathbf{b} \\ & h(\mathbf{x}^k) + \nabla h(\mathbf{x}^k)^\top (\mathbf{x} - \mathbf{x}^k) \leq \mathbf{d}, \end{aligned} \tag{20.9}$$

where

$$\mathbf{B}_k = \nabla^2_{\mathbf{xx}} L(\mathbf{x}^k, \mathbf{y}^k, \mathbf{s}^k)$$

is the Hessian of the Lagrangian function with respect to the $\mathbf{x}$ variables and $(\mathbf{y}^k, \mathbf{s}^k)$ is the current estimate of the vector of Lagrange multipliers.

Algorithm 20.6 describes a template for a sequential quadratic programming approach to problem (20.7). Once again, the step length α at each iteration is typically chosen to balance both goals of objective function reduction and constraint satisfaction.

Algorithm 20.6 Sequential quadratic programming

choose $\mathbf{x}^0, \mathbf{y}^0, \mathbf{s}^0$
for $k = 0, 1, \ldots$ **do**
    solve the quadratic program (20.9) to find a search direction $(\Delta\mathbf{x}^k, \Delta\mathbf{y}^k, \Delta\mathbf{s}^k)$
    choose a step length $\alpha > 0$ and set $(\mathbf{x}^{k+1}, \mathbf{y}^{k+1}, \mathbf{s}^{k+1}) = (\mathbf{x}^k, \mathbf{y}^k, \mathbf{s}^k) + \alpha(\Delta\mathbf{x}^k, \Delta\mathbf{y}^k, \Delta\mathbf{s}^k)$
end for

Interior-Point Methods

Interior-point methods, formerly discussed in Chapters 2 and 5, can be extended to general nonlinear programming under suitable differentiability conditions. The gist of the method is to solve the optimality conditions (20.3).

Similar to the linear and quadratic programming cases, interior-point methods generate a sequence of iterates that satisfy some inequalities strictly and each iteration of the algorithm aims to make progress towards satisfying the optimality conditions (20.3). The algorithm inevitably becomes a bit more elaborate for nonlinear programs because of the nonlinearities in the constraints.

As before we use the following notational convention: given a vector $\mathbf{s} \in \mathbb{R}^p$, let $\mathbf{S} \in \mathbb{R}^{p \times p}$ denote the diagonal matrix defined by $S_{ii} = s_i$ for $i = 1, \ldots, n$, and let $\mathbf{1} \in \mathbb{R}^p$ denote the vector whose components are all 1s. The optimality conditions (20.3) can be restated as

$$\begin{bmatrix} \nabla f(\mathbf{x}) + \nabla g(\mathbf{x})\mathbf{y} + \nabla h(\mathbf{x})\mathbf{s} \\ g(\mathbf{x}) - \mathbf{b} \\ h(\mathbf{x}) + \mathbf{z} - \mathbf{d} \\ \mathbf{SZ1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{s}, \mathbf{z} \geq 0.$$

Given $\mu > 0$, let $(\mathbf{x}(\mu), \mathbf{y}(\mu), \mathbf{z}(\mu), \mathbf{s}(\mu))$ be the solution to the following perturbed version of the above optimality conditions:

$$\begin{bmatrix} \nabla f(\mathbf{x}) + \nabla g(\mathbf{x})\mathbf{y} + \nabla h(\mathbf{x})\mathbf{s} \\ g(\mathbf{x}) - \mathbf{b} \\ h(\mathbf{x}) + \mathbf{z} - \mathbf{d} \\ \mathbf{SZ1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \mu\mathbf{1} \end{bmatrix}, \quad \mathbf{s}, \mathbf{z} > 0.$$

The first condition above can be written as $\mathbf{r}_\mu(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{s}) = \mathbf{0}$ for the *residual vector*:

$$\mathbf{r}_\mu(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{s}) := \begin{bmatrix} \nabla f(\mathbf{x}) + \nabla g(\mathbf{x})\mathbf{y} + \nabla h(\mathbf{x})\mathbf{s} \\ g(\mathbf{x}) - \mathbf{b} \\ h(\mathbf{x}) + \mathbf{z} - \mathbf{d} \\ \mathbf{SZ1} - \mu\mathbf{1} \end{bmatrix}.$$

The *central path* is the set $\{(\mathbf{x}(\mu), \mathbf{y}(\mu), \mathbf{z}(\mu), \mathbf{s}(\mu)) : \mu > 0\}$. Under suitable assumptions $(\mathbf{x}(\mu), \mathbf{y}(\mu), \mathbf{z}(\mu), \mathbf{s}(\mu))$ converges to a local optimal solution to (20.3). This suggests the following algorithmic strategy: Suppose $(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{s})$ is "near" $(\mathbf{x}(\mu), \mathbf{y}(\mu), \mathbf{z}(\mu), \mathbf{s}(\mu))$ for some $\mu > 0$. Use $(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{s})$ to move to a better point $(\mathbf{x}^+, \mathbf{y}^+, \mathbf{z}^+, \mathbf{s}^+)$ "near" $(\mathbf{x}(\mu^+), \mathbf{y}(\mu^+), \mathbf{z}(\mu^+), \mathbf{s}(\mu^+))$ for some $\mu^+ < \mu$.

It can be shown that if a point $(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{s})$ is on the central path, then the corresponding value of μ satisfies $\mathbf{z}^\top \mathbf{s} = p\mu$. Likewise, given $\mathbf{z}, \mathbf{s} > 0$, define

$$\mu(\mathbf{z}, \mathbf{s}) := \frac{\mathbf{z}^\top \mathbf{s}}{p}.$$

To move from a current point $(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{s})$ to a new point, we use the Newton step for the nonlinear system of equations $r_\mu(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{s}) = 0$; that is,

$$(\Delta\mathbf{x}, \Delta\mathbf{y}, \Delta\mathbf{z}, \Delta\mathbf{s}) = -\mathbf{r}'_\mu(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{s})^{-1}\mathbf{r}_\mu(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{s}). \tag{20.10}$$

Algorithm 20.7 presents a template for an interior-point method.

Algorithm 20.7 Interior-point method for nonlinear programming

1: choose $\mathbf{x}^0, \mathbf{y}^0$ and $\mathbf{z}^0, \mathbf{s}^0 > 0$
2: **for** $k = 0, 1, \ldots$ **do**
3: solve the Newton system (20.10) for $(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{s}) = (\mathbf{x}^k, \mathbf{y}^k, \mathbf{z}^k, \mathbf{s}^k)$ and $\mu := 0.1\mu(\mathbf{z}^k, \mathbf{s}^k)$
4: choose a step length $\alpha \in (0, 1]$ and set $(\mathbf{x}^{k+1}, \mathbf{y}^{k+1}, \mathbf{z}^{k+1}, \mathbf{s}^{k+1}) = (\mathbf{x}^k, \mathbf{y}^k, \mathbf{z}^k, \mathbf{s}^k) + \alpha(\Delta\mathbf{x}, \Delta\mathbf{y}, \Delta\mathbf{z}, \Delta\mathbf{s})$
5: **end for**

The step length α in step 4 should be chosen via a backtracking procedure so that $\mathbf{z}^{k+1}, \mathbf{s}^{k+1} > 0$ and the size of $\mathbf{r}_\mu(\mathbf{x}^{k+1}, \mathbf{y}^{k+1}, \mathbf{z}^{k+1}, \mathbf{s}^{k+1})$ is sufficiently smaller than $\mathbf{r}_\mu(\mathbf{x}^k, \mathbf{y}^k, \mathbf{z}^k, \mathbf{s}^k)$.

Table of Contents

We conclude this chapter with a description of nonlinear programming to estimate the volatility surface. The discussion in this section is based on Coleman et al. (1999a,b).

The Black-Scholes-Merton (BSM) equation for pricing European options is based on a geometric Brownian motion model for the movements of the underlying security. Namely, one assumes that the underlying security price S_t at time t satisfies

$$\frac{dS_t}{S_t} = \mu dt + \sigma dW_t \tag{20.11}$$

where μ is the *drift*, σ is the (constant) volatility, and W_t is the standard Brownian motion.

Using this equation and some standard assumptions about the absence of frictions and arbitrage opportunities, one can derive the BSM partial differential equation for the value of a European option on this underlying security. Using the boundary conditions resulting from the payoff structure of the particular option, one determines the value function for the option. For example, for the European call and put options with strike K and maturity T,we obtain the following formulas:

$$C(K,T) = S_0\Phi(d_1) - Ke^{-rT}\Phi(d_2) \tag{20.12}$$

$$P(K,T) = Ke^{-rT}\Phi(-d_2) - S_0\Phi(-d_1) \tag{20.13}$$

where

$$d_1 = \frac{\log(S_0/K) + (r+\sigma^2/2)T}{\sigma\sqrt{T}},$$
$$d_2 = d_1 - \sigma\sqrt{T},$$

and $\Phi(\cdot)$ is the cumulative distribution function for the standard normal distribution.

In the formula r represents the continuously compounded risk-free and constant interest rate and σ is the volatility of the underlying security that is assumed to be constant.

The risk-free interest rate r, or a reasonably close approximation to it, is often available, for example from Treasury bill prices in US markets. Therefore, all one needs to determine the call or put price using these formulas is a reliable estimate of the volatility parameter σ. Conversely, given the market price for a particular European call or put, one can uniquely determine the implied volatility of the underlying security (implied by this option price) by solving the equations above with the unknown σ.

Empirical evidence against the appropriateness of (20.11) as a model for the movements of most securities is abundant. Most such studies refute the assumption of a volatility that does not depend on time or underlying price level.

Indeed, studying the prices of options with the same maturity but different strikes, researchers observed that the implied volatilities for such options exhibited a "smile" structure, i.e., higher implied volatilities away from the money in both directions, decreasing to a minimum level as one approaches the at-the-money option from up or down. This is clearly in contrast with the constant (flat) implied volatilities one would expect had (20.11) been an appropriate model for the underlying price process.

There are quite a few models that try to capture the volatility smile, including stochastic volatility models, jump diffusions, etc. Since these models introduce non-traded sources of risk, perfect replication via dynamic hedging as in the BSM approach becomes impossible and the pricing problem is more complicated.

An alternative that is explored in Coleman et al. (1999b) is the one-factor continuous diffusion model:

$$\frac{dS_t}{S_t} = \mu(S_t, t)dt + \sigma(S_t, t)dW_t, \quad t \in [0, T] \tag{20.14}$$

where the constant parameters μ and σ of (20.11) are replaced by continuous and differentiable functions $\mu(S_t, t)$ and $\sigma(S_t, t)$ of the underlying price S_t and time t. Here T denotes the end of the fixed time horizon. If the instantaneous risk-free interest rate r is assumed constant and the dividend rate is constant, given a function $\sigma(S, t)$, a European call option with maturity T and strike K has a unique price. Let us denote this price with $C(\sigma(S, t), K, T)$.

While an explicit solution for the price function $C(\sigma(S,t),K,T)$ as in (20.12) is no longer possible, the resulting pricing problem can be solved efficiently via numerical techniques. Since $\mu(S,t)$ does not appear in the generalized BSM partial differential equation, all one needs is the specification of the function $\sigma(S,t)$ and a good numerical scheme to determine the option prices in this generalized framework.

So, how does one specify the function $\sigma(S,t)$? First of all, this function should be consistent with the observed prices of currently or recently traded options on the same underlying security. If we assume that we are given market prices of m call options with strikes K_j and maturities T_j in the form of bid-ask pairs (β_j, α_j) for $j = 1, \ldots, n$, it would be reasonable to require that the volatility function $\sigma(S,t)$ is chosen so that

$$\beta_j \leq C(\sigma(S,t), K_j, T_j) \leq \alpha_j, \quad j = 1, \ldots, n. \tag{20.15}$$

To ensure that (20.15) is satisfied as closely as possible, one strategy is to minimize the violations of the inequalities in (20.15):

$$\min_{\sigma(S,t)\in\mathcal{H}} \sum_{j=1}^{n} [\beta_j - C(\sigma(S,t), K_j, T_j)]^+ + [C(\sigma(S,t), K_j, T_j) - \alpha_j]^+ \tag{20.16}$$

Above, $\mathcal{H}$ denotes the space of measurable functions $\sigma(S,t)$ with domain $\mathbb{R}_+ \times [0,T]$ and $[u]^+ = \max\{u, 0\}$. Alternatively, using the closing prices C_j for the options under consideration, or choosing the mid-market prices $C_j = (\beta_j + \alpha_j)/2$, we can solve the following nonlinear least-squares problem:

$$\min_{\sigma(S,t)\in\mathcal{H}} \sum_{j=1}^{n} (C(\sigma(S,t), K_j, T_j) - C_j)^2 \tag{20.17}$$

This is a nonlinear least-squares problem since the function $C(\sigma(S,t), K_j, T_j)$ depends nonlinearly on the variables, namely the local volatility function $\sigma(S,t)$.
While the calibration of the local volatility function to the observed prices using the objective functions in (20.16) and (20.17) is important and desirable, there are additional properties that are desirable in the local volatility function. The most common feature sought in existing models is regularity or smoothness.

For example, in Lagnado and Osher (1997) the authors try to achieve a smooth volatility function by modifying the objective function in (20.17) as follows:

$$\min_{\sigma(S,t)\in\mathcal{H}} \sum_{j=1}^{n} (C(\sigma(S,t), K_j, T_j) - C_j)^2 + \lambda \|\nabla\sigma(S,t)\|_2. \qquad (20.18)$$

Here, λ is a positive tradeoff parameter and $\|\cdot\|_2$ represents the L^2-norm in $\mathcal{H}$. Large deviations in the volatility function would result in a high value for the norm of the gradient function, and by penalizing such occurrences, the formulation above encourages a smoother solution to the problem. The most appropriate

The most appropriate value for the tradeoff parameter λ must be determined experimentally. To solve the resulting problem numerically, one must discretize the volatility function on the underlying price and time grid. Even for a relatively coarse discretization of the S and t spaces, one can easily end up with an optimization problem with many variables.

An alternative strategy is to build the smoothness into the volatility function by modeling it with spline functions. The use of the spline functions not only guarantees the smoothness of the resulting volatility function estimates but also reduces the degrees of freedom in the problem. As a consequence, the optimization problem to be solved has many fewer variables and is easier. This strategy is proposed in Coleman et al. (1999b) and we review it below.

We start by assuming that $\sigma(S, t)$ is a bi-cubic spline. While higher-order splines can also be used, cubic splines often offer a good balance between flexibility and complexity. Next we choose a set of spline knots at points $(\bar{S}_i, \bar{t}_i)$ for $i = 1, \ldots, k$. If the value of the volatility function at these points is given by $\bar{\sigma}_j := \sigma(\bar{S}_j, \bar{t}_j)$, the interpolating cubic spline that goes through these knots and satisfies a particular end condition (such as the natural spline end condition of linearity at the boundary knots) is uniquely determined.

In other words, to completely determine the volatility function as a natural bi-cubic spline (and therefore to determine the resulting call option prices) we have k degrees of freedom represented with the choices $\bar{\sigma} = (\sigma_1, \ldots, \sigma_k)$. Let $\Sigma(S, t, \sigma)$ be the bi-cubic spline local volatility function obtained by setting $\sigma(\bar{S}_j, \bar{t}_j) := \bar{\sigma}_j$. Let $C(\Sigma(S, t, \sigma), S, t)$ denote the resulting call price function. Then the analog of the objective function (20.17) is

$$\min_{\sigma \in \mathbb{R}^k} \sum_{j=1}^{n} (C(\Sigma(S, t, \sigma), K_j, T_j) - C_j)^2 \tag{20.19}$$

One can introduce positive weights w_j for each of the terms in the objective function above to address different accuracies or confidence in the call prices C_j. One can also introduce lower and upper bounds l_i and u_i for the volatilities at each knot to incorporate additional information that may be available from historical data, etc. This way, we form the following nonlinear least-squares problem with k variables:

$$\begin{aligned} \min_{\sigma \in \mathbb{R}^k} f(\sigma) &:= \sum_{j=1}^{n} w_j \left(C(\Sigma(S, t, \sigma), K_j, T_j) - C_j \right)^2 \\ \text{s.t.} \quad & l \leq \sigma \leq u. \end{aligned} \tag{20.20}$$

It should be noted that the formulation above will not be appropriate if there are many more knots than prices, that is, if k is much larger than n. In this case, the problem will be underdetermined and solutions may exhibit "overfitting". There should be fewer knots than available option prices.

The problem (20.20) is a standard nonlinear optimization problem except that the objective function $f(\sigma)$ and in particular the function $C(\Sigma(S, t, \sigma), K_j, T_j)$ depends on the decision variables σ in a complicated and non-explicit manner. Since most of the nonlinear optimization methods we discussed in the previous section require at least the gradient of the objective function (and sometimes its Hessian matrix as well), this may sound alarming. Without an explicit expression for f, its gradient must be estimated either using a finite difference scheme or using automatic differentiation.

Coleman et al. (1999b) implement both alternatives and report that local volatility functions can be estimated very accurately using these strategies. They also test the hedging accuracy of different delta-hedging strategies, one using a constant volatility estimation and another using the local volatility function produced by the strategy above. These tests indicate that the hedges obtained from the local volatility function are significantly more accurate.