numerical optimization methods in economics

Karl Schmedders
From *The New Palgrave Dictionary of Economics*, Second Edition, 2008
Edited by Steven N. Durlauf and Lawrence E. Blume

Abstract

Optimization problems are ubiquitous in economics. Many of these problems are sufficiently complex that they cannot be solved analytically. Instead economists need to resort to numerical methods. This article presents the most commonly used methods for both unconstrained and constrained optimization problems in economics; it emphasizes the solid theoretical foundation of these methods, illustrating them with examples. The presentation includes a summary of the most popular software packages for numerical optimization used in economics, and closes with a description of the rapidly developing area of mathematical programs with equilibrium constraints, an area that shows great promise for numerous economic applications.

Keywords

central path; computable general equilibrium models; computational algebraic geometry; constrained optimization; global minimum; homotopy continuation methods; interior-point methods; Karush-Kuhn-Tucker conditions; Kuhn–Tucker conditions; Lagrange multipliers; line search methods; linear independence constraint qualification (LICQ); linear programming; local minimum; local solution; logarithmic barrier method; mathematical programs with equilibrium constraints; metaheuristics; multidimensional optimization problems; Newton's method; Newton–Raphson methods; nonlinear programming; numerical optimization methods; optimality conditions; optimization algorithms; penalty methods; polynomial functions; quadratic programs; sequential quadratic programming; simplex method for solving linear programs; simulated annealing; Smale's global Newton method; steepest descent method; Taylor's th; trust region methods; unconstrained optimization

JEL classifications

D5

Article

Optimizing agents are at the centre of most economic models. In our models we typically assume that consumers maximize utility or wealth, that players in a game maximize payoffs, that firms minimize costs or maximize profits, or that social planners maximize welfare. But it is not only the agents in our models that optimize. Econometricians maximize likelihood functions or minimize sums of squares. Clearly optimization is one of the key techniques of modern economic analysis.

The optimization problems that appear in economic analysis vary greatly in nature. We encounter finite-dimensional problems such as static utility maximization problems with a few goods. An optimal solution to such a problem is a finite-dimensional vector. We analyse infinite-dimensional problems such as infinite-horizon social planner models or continuous-time optimal control problems. Here the solution is an infinite-dimensional object, a vector with countably infinitely many elements or even a function over an interval. Our agents may face constraints such as budget equations, short-sale restrictions or incentive-compatibility constraints. There are also unconstrained problems such as nonlinear least-square problems. Decision variables may even be restricted to be discrete. Agents' objective functions may be linear or nonlinear, convex or nonconvex, many times differentiable or discontinuous. Finally, an economic optimization problem may be deterministic or stochastic.

Unless we consider stylized models in theoretical work or make very stringent and often quite unrealistic assumptions in applied models, the optimization problems that we encounter cannot be solved analytically. Instead we need to resort to numerical methods. The numerical methods that we employ to solve economic optimization models vary just as much as the optimization problems we encounter. It is therefore impossible to cover the wide variety of numerical optimization methods that are useful in economics in a short article. For the purpose of the exposition here we focus on deterministic finite-dimensional nonlinear optimization problems including linear programs. This is a natural choice because such problems are ubiquitous in economic analysis. Moreover, the techniques for these problems play also an important part in many other numerical methods such as those for solving economic equilibrium and infinite-dimensional problems. The interested reader should consult computation of general equilibria (new developments), computational methods in econometrics and dynamic programming.

We first indicate some of the fundamental technical difficulties that we need to be aware of when we apply numerical methods to our economic optimization problems. We then highlight the basic theoretical foundations for numerical optimization methods. The popular numerical optimization methods have strong theoretical foundations. Unfortunately, current textbooks in computational economics, with the partial exception of Judd (1998), neglect to emphasize these foundations. As a result some economists are rather sceptical about numerical methods and view them as rather ad hoc approaches. Instead, a good understanding of the theoretical foundations of the numerical solution methods gives us an appreciation of the capabilities and limitations of these methods and can guide our choice of suitable methods for a specific economic problem. We outline the most fundamental numerical strategies that form the basis for most algorithms. All presented numerical strategies are implemented in at least one of the those computer software packages for solving optimization problems that are most popular in economics. We close our discussion with a look at mathematical programs with equilibrium constraints (MPECs), a promising research area in numerical optimization that has useful applications in economics.

1 Newton's method in one dimension

We start with the one-dimensional unconstrained optimization problem

$$\min_{x \in \mathbb{R}} f(x). \tag{1}$$

Perhaps the first (if any) numerical method that most of us learnt in our calculus classes is Newton's method. Newton's method attempts to minimize successive quadratic approximations to the objective function f in the hope of eventually finding a minimum of f. To start the computations we need to provide an initial guess $x^{(0)}$. The quadratic approximation $q(x)$ of $f(x)$ at the point x^0 is

$$q(x) = f(x^0) + f'(x^{(0)})(x - x^{(0)}) + \frac{1}{2} f''(x^{(0)})(x - x^{(0)})^2$$

where f' and f'' denote the first and second derivative of the function f, respectively. Solving the first-order condition

$$q'(x) = f'(x^{(0)}) + f''(x^{(0)})(x - x^{(0)}) = 0$$

on the assumption that $f'(x^{(0)}) \neq 0$ yields the solution

$$x^{(1)} = x^{(0)} - \frac{f'(x^{(0)})}{f''(x^{(0)})}.$$

Now we repeat this process using a quadratic approximation to f at the point $x^{(1)}$. The result is a sequence of points, $\{x^{(k)}\} = x^{(0)}, x^{(1)}, x^{(2)}, \ldots, x^{(k)}, \ldots$, that we hope will converge to the solution of our minimization problem. This approach is based on the following theoretical result.

Theorem Suppose x^* is the solution to the minimization problem (1). Suppose further that f is three times continuously differentiable in a neighborhood of x^* and that $f'(x^*) \neq 0$. Then there exists some $\delta > 0$ such that if $|x^* - x(0)| < \delta$, then the sequence $\{x^{(k)}\}$ converges quadratically to x^*, that is,

$$\lim_{k \to \infty} \frac{|x^{(k+1)} - x^*|}{|x^{(k)} - x^*|^2} = \kappa$$

for some finite constant κ. □

We illustrate this theorem with a simple example.

Example 1 A consumer has a utility function $u(x, y) = \ln(x) + 2 \ln(y)$ over two goods. She can spend \$1 on buying quantities of these two goods, both of which have a price of \$1. After substituting the budget equation, $x + y = 1$, into the utility function the consumer wants to maximize $f(x) = \ln(x) + 2 \ln(1 - x)$. Setting the first order condition equal to 0 yields the solution $x^* = \frac{1}{3}$. (This quantity is globally optimal because the function f is strictly concave.)

Suppose we start Newton's method with the initial guess $x^{(0)} = 0.5$. Then the first Newton step yields

$$x^{(1)} = 0.5 - \frac{f'(0.5)}{f''(0.5)} = 0.5 - \frac{-2}{-12} = \frac{1}{3}.$$

Newton's method found the exact optimal solution in one step. This (almost) never happens in practice. Much more usual is the behaviour we observe when we start with $x^{(0)} = 0.8$. Then Newton's method delivers as its first five steps

$$0.63030303, 0.407373702, 0.328873379, 0.333302701, 0.333333332.$$

We observe that the sequence rapidly converges to the optimal solution. The corresponding errors $|x^{(k)} - x^*|$,

$$0.2969697, 0.07404037, 0.00445995, 3.0632 \cdot 10^{-5}, 1.4078 \cdot 10^{-9}$$

converge to but never exactly reach zero. The rate of convergence is called quadratic since $|x^{(k+1)} - x^*| < L|x^{(k)} - x^*|^2$ for some constant L once k is sufficiently large. □

But, of course, contrary to this simple example, we typically do not know x^* and so cannot compute the errors $|x^{(k)} - x^*|$. Instead, we need a stopping rule that indicates when the procedure terminates. The requirement that $f'(x^{(k)}) < \delta$ may appear to be an intuitive stopping rule. But that rule may be insufficient for functions that are very 'flat' near the optimum and have large ranges of non-optimal points satisfying this rule. Therefore, a safer stopping rule requires both $f'(x^{(k+1)}) < \delta$ and $|x^{(k+1)} - x^{(k)}| < \varepsilon\,(1 + |x^{(k)}|)$ for some pre-specified small error tolerance $\varepsilon, \delta > 0$. So the Newton method terminates once two subsequent iterates are close to each other and the first derivative almost vanishes.

Observe that Newton's method found a maximum, and not a minimum, of the utility function. The reason for this fact is that this method does not search directly for an optimizer. Note that the key step in the algorithm is finding a stationary point of the quadratic approximation $q(x)$, that is, a point satisfying $q'(x) = 0$. Before we can claim to have found a maximum or minimum of f we need to do more work. In this example the strict concavity of the utility function ensures that a stationary point of f yields a maximum. So an assumption of our economic model assures us that the numerical method indeed finds the desired maximum.

Example 2 Consider the simple polynomial function $f(x) = x(x-2)^2$. Starting with $x^{(0)} = 1$ leads to the sequence

$$0.5, 0.65, 0.666463415, 0.666666636, \ldots$$

converging to $\frac{2}{3}$. Starting with $x^{(0)} = 1.5$ leads to the sequence

$$2.75, 2.198529412, 2.022777454, 2.000376254$$

converging to 2. Neither of these two points yields a global optimum, the function f is actually unbounded above and below. The point $\frac{2}{3}$ is a local maximizer ($f''(2/3) = -4 < 0$) while 2 is a local minimizer ($f''(2) = 4 > 0$). The stationary point that we find greatly depends on our initial guess. □

Our simple observations about the behaviour of Newton's method for one-dimensional optimization problems apply in practice to higher-dimensional nonlinear optimization problems and to almost all optimization methods. We will almost always face these fundamental issues in our economic applications. First, most practical optimization methods for unconstrained problems search only for a

stationary point (with possibly additional favourable properties). They do not directly attempt to compute an optimizer. Second, as a result, most practical methods may terminate with a non-optimal point. To ensure global optimality we need to perform additional checks. Third, it is rather unusual in practice to explicitly solve for an exact solution. Usually we can only hope for a sequence of points $\{x^{(k)}\}$ generated by an iterative process that converges to a limit having some desired property. Therefore, we need a stopping rule that indicates when the iterative process stops. Fourth, the algorithm may not terminate and diverge even if a globally optimal solution exists.

Newton's method is a special instance of a family of methods for solving multidimensional optimization problems. Before we examine more general methods we provide some basic intuition for the theoretical underpinnings of these solution methods.

2 Theoretical foundation: Taylor's th

The gradient of the function f at a point $x = (x_1, x_2, \ldots, x_n)$ is the column vector

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_n}(x)\right)^\top$$

of partial derivatives of f with respect to the variables x_1,x_2,. . ., x_n. The Hessian of f at x is the $(n \times n)$-matrix

$$H(x) = \left(\frac{\partial^2 f}{\partial x_i \partial x_j}(x)\right)_{i,j=1}^{n}$$

of the second derivatives $\frac{\partial^2 f}{\partial x_i \partial x_j}(x)$ of f. The inner product of two (column) vectors $x, y \in \mathbb{R}^n$ is denoted by $x^\top y$.

Many numerical methods rely on linear or quadratic approximations of the function f. Taylor's theorem provides a justification for this approach. Here we give a simple version of this theorem for functions with Lipschitz continuous derivatives. Consider a function $F : X \to Y$ for open sets $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^m$. Then F is Lipschitz continuous at $x \in X$ if there exists a constant $\gamma(x)$ such that

$$\|F(y) - F(x)\| \leq \gamma(x)\|y - x\|$$

for all $y \in X$, where $\|\cdot\|$ denotes the standard Euclidean norm.

Theorem Suppose the function $f : X \to \mathbb{R}$ is continuously differentiable on the open set $X \subset \mathbb{R}^n$ and that the gradient function ∇f is Lipschitz continuous at x with Lipschitz constant $\gamma^l(x)$. Also suppose that for $s \in \mathbb{R}^n$ the line segment $x + \theta s \in X$ for all $\theta \in [0,1]$. Then, the linear function l with $l(s) = f(x) + \nabla f(x)^\top s$ satisfies

$$|f(x+s) - l(s)| \leq \frac{1}{2}\gamma^l(x)\|s\|^2.$$

Moreover, if f is twice continuously differentiable on X and the Hessian H is Lipschitz continuous at x with Lipschitz constant $\gamma^q(x)$, then the quadratic function q with $q(s) = f(x) + \nabla f(x)^\top s + \frac{1}{2}s^\top H(x)s$ satisfies

$$|f(x+s) - q(s)| \leq \frac{1}{6}\gamma^q(x)\|s\|^3.$$

□

3 Unconstrained optimization

The multidimensional generalization of the unconstrained optimization problem (1) is given by

$$\min_{x \in \mathbb{R}^n} f(x). \tag{2}$$

Solving this optimization problem entails finding a global minimizer x^* satisfying $f(x^*) \leq f(x)$ for all $x \in \mathbb{R}^n$. With the exception of a few algorithms for problems that are either very small or have very special structure, there are no algorithms that are guaranteed to find a global minimum. Thus, we need to think in terms of local minima. A local minimizer is a point x^* that satisfies $f(x^*) \leq f(x)$ for all $x \in \mathcal{N}(x^*)$ where $\mathcal{N}(x^*)$ denotes a neighborhood of x^*. The point x^* is called an isolated local minimizer if it is the only local minimizer in $\mathcal{N}(x^*)$.

All these definitions by themselves are not all that helpful for finding a minimum. Instead, just as Newton's method in one dimension does, all practical numerical methods for unconstrained optimization problems rely on optimality conditions to find candidates for local minima. For functions with sufficient differentiability properties these are the following well-known conditions.

Theorem [Optimality conditions for unconstrained minimization]

1. If f is continuously differentiable and x^* is a local minimizer of f, then $\nabla f(x^*) = 0$.
2. If f is twice continuously differentiable and x^* is a local minimizer of f, then $\nabla f(x^*) = 0$ and $s^\top H(x^*)s \geq 0$ for all $s \in \mathbb{R}^n$.
3. If f is twice continuously differentiable and if x^* satisfies $\nabla f(x^*) = 0$ and $s^\top H(x^*)s > 0$ for all $s \in \mathbb{R}^n, s \neq 0$, then x^* is an isolated local minimizer of f. □

But when can we be assured that a local minimizer of f is actually a solution to the unconstrained optimization problem (2)? The perhaps easiest sufficient condition is that the function f is convex, that is, $s^\top H(x)s \geq 0$ for all $x \in \mathbb{R}^n$ if f is twice differentiable. Then any local minimizer x^* is a solution to problem (2), in fact, any stationary point x^* is a solution to (2).

The optimality conditions provide the foundation for all practical unconstrained optimization methods. The focus of all these algorithms is to find (actually, to approximate) a stationary point of f, that is, a solution to $\nabla f(x) = 0$. They do so by generating a sequence of iterates $\{x^{(k)}\}$ that ideally terminates once a stopping rule is satisfied indicating that an approximate solution has been found. The key step for these methods is to generate a new iterate $x^{(k+1)}$ from a current iterate $x^{(k)}$. A vast majority of optimization routines uses one of two basic strategies for moving from $x^{(k)}$ to $x^{(k+1)}$, a line search approach or a trust region method.

3.1 Line search methods

The general set-up of a line search method is as follows. From a point $x^{(k)}$ (with $\nabla f(x^{(k)}) \neq 0$) we look for a search direction $s^{(k)}$ that leads us to lower function values for f. Using the linear approximation l with $l(s) = f(x^{(k)}) + \nabla f(x^{(k)})^\top s$ we determine a descent direction $s^{(k)}$ satisfying

$$\nabla f(x)^\top s^{(k)} < 0,$$

which in turn implies $l(s^{(k)}) < f(x^{(k)})$. Because of Taylor's theorem we hope that along a step in the direction $s^{(k)}$ the function value $f(x)$ will be reduced. We calculate a suitable step length $\alpha_k > 0$ to ensure that $f(x^{(k+1)}) < f(x^{(k)})$ where

$$x^{(k+1)} = x^{(k)} + \alpha_k s^{(k)}.$$

Observe that at a given point $x^{(k)}$ and for a descent direction $s^{(k)}$ finding the optimal value of α_k requires us to solve a one-dimensional optimization problem. In principle we could apply Newton's method to this problem. In practice, however, this one-dimensional problem does not need to be solved exactly because repeatedly finding the optimal step length is both unnecessary for convergence of line search methods and computationally rather inefficient. Instead modern line search methods prefer to use inexact line searches that just pick a step length that leads to a sufficient decrease in the objective function value. One such approach is the backtracking Armijo line search, which requires that

$$f(x^{(k)} + \alpha_k s^{(k)}) \leq f(x^{(k)}) + \alpha_k \beta \nabla f(x^{(k)})^\top s^{(k)}$$

for some $\beta \in (0,1)$. The idea of this requirement is to link the step size α_k to the decrease in f. The longer the step the larger the decrease must be. Starting with an initial guess for α_k, say 1, we can now stepwise reduce the value of α_k until the above condition is satisfied. At that point we set $x^{(k+1)} = x^{(k)} + \alpha_k s^{(k)}$.

While the basic line search method seems very intuitive, it can fail if the search direction and the gradient tend to a point where they are orthogonal to each other, that is, the product $\nabla f(x^{(k)})^\top s^{(k)}$ tends to zero without the gradient itself approaching zero. This kind of failure can be avoided by a proper choice of search direction.

3.1.1 Method of steepest descent

The perhaps most intuitive choice for a descent direction is

$$s^{(k)} = -\nabla f(x^{(k)}),$$

because this search direction gives the greatest possible decrease in the linear approximation l (for a fixed step length). It is thus called the steepest descent direction. And indeed, a line search with the steepest descent direction has very nice theoretical properties.

Theorem Suppose that f is continuously differentiable and that ∇f is Lipschitz continuous on $\mathbb{R}^n$. Then for the sequence $\{x^{(k)}\}$ of iterates generated by a line search

method using the steepest descent direction and the backtracking Armijo line search one of the following three conditions must hold.

(C1) $\nabla f(x^{(k)}) = 0$ for some $k \geq 0$.
(C2) $\lim_{k \to \infty} \nabla f(x^{(k)}) = 0$.
(C3) $\lim_{k \to \infty} f(x^{(k)}) = -\infty$. □

The method of steepest descent has the global convergence property, that is, independent of the starting point the sequence of gradients will converge to a stationary point (but that does not mean that the sequence $x^{(k)}$ converges, think of $-\ln(x^{(k)})$!) or the function values diverge and indicate that no minimum exists.

Example 3 A consumer has a utility function $u(x_1, x_2, x_3) = \sqrt{x_1} + 2\sqrt{x_2} + 3\sqrt{x_3}$ over three goods. She can spend \$1 on buying quantities of these three goods, all of which have a price of \$1. After substituting the budget equation, $x_1 + x_2 + x_3 = 1$, into the utility function the consumer wants to maximize $\sqrt{x_1} + 2\sqrt{x_2} + 3\sqrt{1 - x_1 - x_2}$. (We can trivially solve this problem with pencil and paper and find the optimal solution $(\frac{1}{14}, \frac{4}{14}, \frac{9}{14})$.) We solve the consumer's optimization problem by minimizing the function $f(x_1, x_2) = -(\sqrt{x_1} + 2\sqrt{x_2} + 3\sqrt{1 - x_1 - x_2})$ with a steepest descent method (using the optimal step length in each step). Figure 1 indicates some of the early steps and Table 1 lists details of some of the steps. (To show convergence of variable values and the optimal function value we report six digits for these terms. The search direction and norm of the gradient are converging to zero and so for simplicity we report fewer and not always the same number of digits. We abbreviate numbers like $6.7 \cdot 10^{-8}$ by $6.7(-8)$.)

The steepest descent method makes good progress in the first few iterations but then slows down considerably. Note the comparatively little change in the values of $x^{(k)}$ during the last 10 to 15 iterations. The figure shows a lot of 'zigzagging' from iterate to iterate. □

Table 1 Steps of a steepest descent method

k	$x_1^{(k)}$	$x_2^{(k)}$	$s^{(k)}$		$\|\nabla f(x^{(k)})\|$	$f(x^{(k)})$
0	0.1	0.5	−0.7906	−0.9575	1.2417	−3.62781
1	0.0358229	0.422272	0.6041	−0.4988	0.7834	−3.69734
2	0.0867861	0.380194	−0.3573	−0.4328	0.5612	−3.71804
3	0.0528943	0.339146	0.2503	−0.2066	0.3245	−3.73387
4	0.0772951	0.318999	−0.1321	−0.1600	0.2075	−3.73858
5	0.0643195	0.303284	0.0853	−0.0704	0.1106	−3.74074
6	0.0732734	0.295891	−0.0414	−0.0502	0.0651	−3.74136
7	0.0691862	0.290940	0.0257	−0.0212	0.0334	−3.74157
⋮						⋮
10	0.0715805	0.286543	−0.0034	−0.0041	0.0054	−3.74166
⋮						⋮
15	0.0714140	0.285747	1.64 (−4)	−1.35 (−4)	2.12 (−4)	−3.74166
⋮						⋮
20	0.0714288	0.285716	−5.94 (−6)	−7.19 (−6)	9.33 (−6)	−3.74166

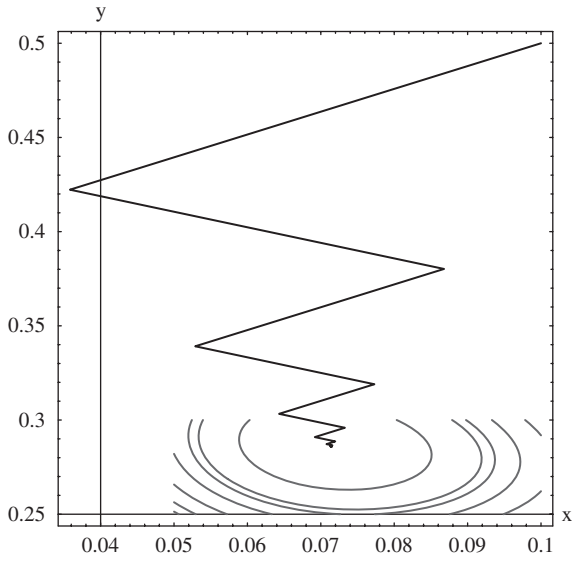

Figure 1 First steps of a steepest descent method

The behaviour of the steepest descent method in the example is quite typical. As a result the convergence of the method is rather slow. And so, despite having the global convergence property, it is useless in practice. The slow convergence (see Nocedal and Wright, 2006, ch. 3) of this method renders it impractical. The convergence problems are essentially due to the reliance on a first-order approximation, which ignores the curvature properties of *f*. Newton's method takes advantage of a second-order approximation.

3.1.2 Newton methods

The quadratic approximation q of the objective function f at an iterate $x^{(k)}$ is given by

$$q(s) = f(x^{(k)}) + \nabla f(x^{(k)})^{\top} s + \frac{1}{2} s^{\top} H(x^{(k)}) s.$$

The first-order condition $q'(s) = 0$ yields the search direction

$$s^{(k)} = -H(x^{(k)})^{-1} \nabla f(x^{(k)}).$$

Only under very strong conditions is Newton's method globally convergent.

Theorem Suppose that f is continuously differentiable and that ∇f is Lipschitz continuous on $\mathbb{R}^n$. If for the sequence $\{x^{(k)}\}$ of iterates generated by a line search method using the Newton direction and the backtracking Armijo line search the Hessian matrices $H(x^{(k)})$ are positive definite with eigenvalues that are uniformly bounded away from zero, then one of the conditions (C1), (C2), (C3) must hold. □

Example 4 We revisit the consumer's optimization problem from Example 3 and minimize the function $f(x_1, x_2) = -(\sqrt{x_1} + 2\sqrt{x_2} + 3\sqrt{1 - x_1 - x_2})$ with a Newton

method (using the optimal step length in each step). Table 2 lists all the steps of this method and Figure 2 displays some of the early steps.

Newton's method converges very rapidly. Unlike the steepest descent method it does not slow down near the solution, instead we see a quadratic rate of convergence just like in the one-dimensional problem in Example 1. □

Table 2 Steps of a Newton method

k	$x_1^{(k)}$	$x_2^{(k)}$	$s^{(k)}$		$\|\nabla f(x^{(k)})\|$	$f(x^{(k)})$
0	0.1	0.5	−0.0161	−0.2078	1.2417	−3.62781
1	0.0829896	0.280	−0.0128	0.0062	0.1440	−3.74077
2	0.0714128	0.285450	1.58 (−5)	2.64 (−4)	0.0014	−3.74166
3	0.0714286	0.285714	−2.27 (−8)	1.10 (−8)	3.15 (−7)	−3.74166
4	0.0714286	0.285714			5.46 (−15)	−3.74166

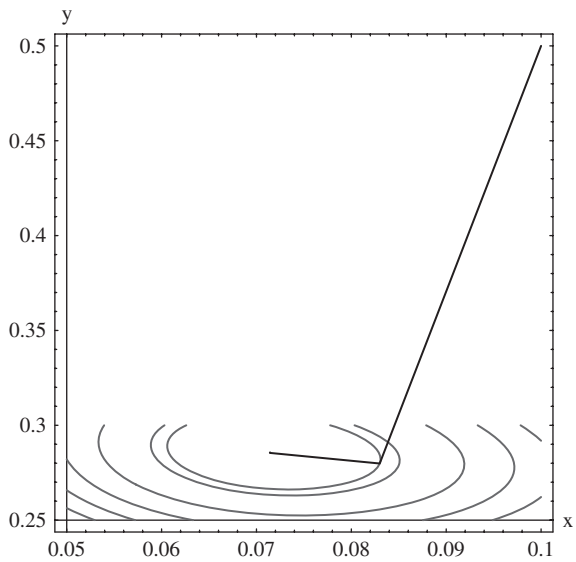

Figure 2 First steps of a Newton method

The condition that the Hessian matrix $H(x^{(k)})$ is positive definite for the entire sequence $\{x^{(k)}\}$ is rarely satisfied for general problems. But if the Hessian is not positive definite then the search direction $s^{(k)}$ may be an ascent instead of a descent direction. The modified Newton methods address this problem by modifying the Hessian matrix $H(x^{(k)})$. These methods choose a search direction

$$s^{(k)} = -(H(x^{(k)}) + M(x^{(k)}))^{-1}\nabla f(x^{(k)}),$$

where the matrix $M(x^{(k)})$ is chosen so that $H(x^{(k)}) + M(x^{(k)})$ is 'sufficiently' positive definite. If $H(x^{(k)})$ is sufficiently positive definite itself then, of course, $M(x^{(k)}) = 0$. A proper choice of $M(x^{(k)})$ is crucial for the effectiveness of this approach; see Gould and Leyffer (2002) and Nocedal and Wright (2006) for many more details.

The most tedious task in Newton's method is the computation of the Hessian matrix $H(x^{(k)})$. Therefore, for decades it was fashionable to develop methods, the so-called quasi-Newton methods, that rely on approximations of the exact Hessian

matrix. Interest in these methods has somewhat diminished due to the development of automatic differentiation techniques. These techniques allow a very fast and reliable computation of derivatives and so make the task of calculating the Hessian feasible even for large problems. Nocedal and Wright (2006, ch. 6) discuss quasi-Newton methods in detail.

Before we continue our discussion of optimization algorithms we pause for a quick comment on some potential name confusion. In addition to Newton methods for unconstrained optimization there is also a Newton method for solving nonlinear systems of equations. To avoid confusion and for historical reasons the root-finding methods for nonlinear systems of equations are sometimes called Newton–Raphson methods; see Judd (1998) and references therein. In particular, Newton methods for solving unconstrained optimization problems should not be confused with so-called global Newton methods. In economic theory the term 'Smale's global Newton method' appears to be well known. This term refers to a solution method for solving nonlinear systems of equations (see Smale, 1976) which is closely related to homotopy continuation methods. Clearly, we could use methods for nonlinear equations to solve the first-order conditions $\nabla f(x) = 0$. This approach, however, does not use other information from the underlying optimization problem and thus is often inefficient. Here we do not discuss methods for solving nonlinear equations, and refer to Allgower and Georg (1979), Judd (1998) and Miranda and Fackler (2002).

3.2 Trust region methods

Line search methods use an approximation of the objective function f to generate a search direction. Subsequently they determine a suitable step length along this direction. Trust region methods also rely on an approximation of f, but they first define a region around the current iterate in which they trust the approximation to be adequate. Then they simultaneously choose the direction and step length.

For the purpose of our discussion here we consider a quadratic approximation of f around $x^{(k)}$,

$$q_k(s) = f(x^{(k)}) + \nabla f(x^{(k)})^\top s + \frac{1}{2} s^\top B(x^{(k)}) s,$$

where $B(x^{(k)})$ is a symmetric approximation of the Hessian matrix $H(x^{(k)})$. Trust region methods do not require the Hessian matrix of the function q_k to be positive definite. Therefore, we could use $B(x^{(k)}) = H(x^{(k)})$. In that case, the algorithm is called a trust region Newton method. Given a trust region radius $\Delta_k > 0$ in each iteration, the algorithm seeks an (approximate) solution to the trust region sub-problem

$$\min_{s \in \mathbb{R}^n} q_k(s) \text{ subject to } \|s\| \leq \Delta_k.$$

Before we discuss how we may solve this sub-problem we need to decide on a proper choice for the trust region radius. Note that $q_k(0) - q_k(s^{(k)})$ is the predicted reduction for a step $s^{(k)}$. Similarly, $f(x^{(k)}) - f(x^{(k)} + s^{(k)})$ is the actual decrease in the objective. The ratio

$$\rho_k = \frac{f(x^{(k)}) - f(x^{(k)} + s^{(k)})}{q_k(0) - q_k(s^{(k)})}$$

gives an indication on how well the quadratic approximation predicts the reduction in the function value. Ideally we would like the step $s^{(k)}$ to yield a value of ρ_k of close to or larger than 1. In that case we accept the step and may possibly increase the radius for the next iteration. If, however, ρ_k is close to zero or even negative, then we would decrease the trust region radius, set $x^{(k+1)} = x^{(k)}$, and attempt to solve the sub-problem again.

Recall that line search methods do not require the step length to be chosen optimally in order to be globally convergent. Similarly, it is unnecessary and in fact computationally inefficient to solve the trust region sub-problem exactly. Instead, it suffices to search for a step giving a sufficient reduction in q_k. Such a sufficient reduction is achieved by requiring a decrease that is at least as large at that obtained by a step in the direction of steepest descent. The solution to

$$\min_{\alpha \in \mathbb{R}} q_k(-\alpha \nabla f(x^{(k)})) \text{ subject to } \| -\alpha \nabla f(x^{(k)}) \| \leq \Delta_k$$

yields the Cauchy point

$$s_k^C = -\tau_k \Delta_k \frac{\nabla f(x^{(k)})}{\| \nabla f(x^{(k)}) \|}$$

where the constant $\tau_k \in (0,1]$ depends on the curvature of q_k and the radius Δ_k; see Nocedal and Wright (2006) for a closed-form solution. The approximate solution $s^{(k)}$ of the trust region subproblem must now satisfy $q_k(s^{(k)}) \leq q_k(s_k^C)$.

Theorem Let q_k be the second-order approximation of the objective function f at $x^{(k)}$ and let s_k^C be its Cauchy point in the trust region defined by $\|s\| \leq \Delta_k$. Then

$$q_k(0) - q_k(s_k^C) = f(x^{(k)}) - q_k(s_k^C) \geq \frac{1}{2} \| \nabla f(x^{(k)}) \| \min\left\{ \frac{\| \nabla f(x^{(k)}) \|}{1 + \| B(x^{(k)}) \|}, \Delta_k \right\}.$$

□

The theorem has the typical flavour of results on trust region methods. It relates the reduction in the quadratic approximation, $q_k(0) - q_k(s_k^C)$, to $\|\nabla f(x^{(k)})\|$, which is a measure for the distance to optimality. Once again a global convergence result holds.

Theorem Consider the sequence $\{x^{(k)}\}$ of iterates generated by the described trust region method. Suppose that f is twice continuously differentiable and both the Hessian of f and the quadratic approximation q_k are bounded for all k. Then one of the conditions (C1), (C2), (C3) must hold. □

The trust region method based on the Cauchy point is effectively a steepest descent (line search) method where the choice of the step length is bounded by the trust region radius. Therefore, this method also suffers from very poor convergence in practice. Better algorithms start from the Cauchy point and try to improve upon it. There is a variety of such methods that take advantage of additional properties of f;

see Gould and Leyffer (2002) and Nocedal and Wright (2006). For a comprehensive treatment of trust region methods, see Conn, Gould and Toint (2000).

4 Constrained optimization

Now we consider the constrained optimization problem

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} & f(x) \\ \text{s.t.} & g_i(x) \geq 0 i \in I \\ & h_j(x) = 0 j \in E. \end{array} \qquad \text{(NLP)}$$

We define the feasible region $\mathscr{F}$ of this optimization problem to be the set of all points that satisfy the constraints, so

$$\mathscr{F} = \left\{ x \in \mathbb{R}^n \mid g_i(x) \geq 0, \;\; i \in I; \;\; h_j(x) = 0, \;\; j \in E \right\}.$$

Just as for the unconstrained optimization problem, we can define global and local solution. Of course, a desired optimal solution x^* to this optimization problem satisfies $f(x^*) \leq f(x)$ for all $x \in \mathscr{F}$. A point x^* is a local minimizer if it satisfies $f(x^*) \leq f(x)$ for all $x \in \mathscr{N}(x^*) \cap \mathscr{F}$ for some neighbourhood $\mathscr{N}(x^*)$ of x^*. The vector x^* is an isolated local minimizer if there exists a neighbourhood $\mathscr{N}(x^*)$ in which it is the only local minimizer.

The conditions of these definitions, just like their counterparts for unconstrained optimization problems, are pretty much useless for the computation of optimal solutions – with one major exception. The simplex method for solving linear programming problems relies on the comparison of objective function values at some special points in the feasible region. Most other practical numerical methods, however, rely again on optimality conditions. Penalty methods transform the problem (NLP) into (a sequence of) unconstrained optimization problems and then rely on their respective first-order conditions. Many methods rely directly on optimality conditions for constrained optimization. These optimality conditions require that certain degenerate behaviour does not occur at potential minimizers. Conditions that rule out such degenerate points are called 'constraint qualifications'. These conditions are important but do not always get the proper attention in economics, but see Simon and Blume (1994) for a rigorous treatment. Numerous such constraint qualifications exist; here we just mention one such condition.

The set of constraints that hold with equality at a feasible point $x \in \mathscr{F}$ is called the active set $\mathscr{A}(x)$. Formally,

$$\mathscr{A}(x) = \{ i \in I | g_i(x) = 0 \} \cup E.$$

The linear independence constraint qualification (LICQ) holds at a point $x \in \mathscr{F}$ if the gradients of all active constraints are linearly independent. Now we can state the well-known first-order necessary conditions, which most of us learnt as Kuhn–Tucker or Karush-Kuhn-Tucker (KKT) conditions.

Theorem Suppose x^* is a local solution of the problem (NLP) that satisfies the (LICQ). Then there exist unique Lagrange multipliers ν_i^*, $i \in I$, and λ_j^*, $j \in E$, such that the following conditions are satisfied.

$$\nabla f(x^*) - \sum_{i \in I} \nu_i^* \nabla g_i(x^*) - \sum_{j \in E} \lambda_j^* \nabla h_j(x^*) = 0, \tag{3}$$

$$g_i(x^*) \geq 0, \quad \text{for all } i \in I, \tag{4}$$

$$h_j(x^*) = 0, \quad \text{for all } j \in E, \tag{5}$$

$$\nu_i^* g_i(x^*) = 0, \quad \text{for all } i \in I, \tag{6}$$

$$\nu_i^* \geq 0, \quad \text{for all } i \in I. \tag{7}$$

Again we may ask when we can be assured that a solution to the KKT conditions is actually a solution to the nonlinear optimization problem (NLP). If the feasible region $\mathscr{F}$ is a convex set (see Simon and Blume, 1994), and the objective function f is convex on $\mathscr{F}$, then the problem (NLP) is called a convex programming problem, and any local solution is also a (global) solution of (NLP). For example, if the functions $h_j, j \in E$, are all linear and the functions $-g_i, i \in I$, are all convex, then $\mathscr{F}$ is a convex set. In this case, if f is convex, too, indeed any solution to the KKT conditions is a solution to (NLP).

Many of the most popular numerical methods for solving nonlinear constrained optimization problems take advantage of the KKT conditions in one form or another. First, however, we describe the basic version of the simplex method for linear programming which does not rely on first-order conditions.

4.1 The simplex method

When the objective function f and the constraint functions g_i, $i \in I$, and h_j, $j \in E$, are all linear functions in the variables $x \in \mathbb{R}^n$, then the constrained optimization problem is a linear programming problem, or 'linear program' for short. Linear programs have a standard form,

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} & c^\top x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array} \quad \text{(LP)}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and A is an $m \times n$ matrix. We can easily transform any linear programming problem with arbitrary linear inequalities and unbounded variables into this standard form.

The development of the simplex method in the late 1940s (Dantzig, 1949) for solving linear programs is generally regarded as the beginning of the modern era of optimization (Nocedal and Wright, 2006). The simplex method is, however, not only of historical importance but to this day the perhaps most widely used tool in optimization outside economics. Here we describe the fundamental idea of the basic version of the simplex method.

The system of equality constraints, $Ax = b$, has m equations in the n decision variables. For the LP to be an interesting optimization problem it must be the case that $m < n$. If $m > n$ then either the linear system is overdetermined and so the feasible region is empty and the LP has no solution, or the system can be simplified

so that the number of equations does not exceed the number of variables. The same conclusions apply for the case $m = n$ if the matrix A is singular. If $m = n$ and A has full rank, then the feasible region consists of at most one point and the LP is trivial. We can therefore assume that the system of equality constraints is underdetermined, that is, it has fewer equations than variables. Modern computer implementations of the simplex method start with a pre-processing phase, which transforms a given linear programming problem by removing redundancies and possibly even also eliminating some variables.

We can easily calculate some of the solutions to the system $Ax = b$. If we choose m of the n variables and set the remaining $n - m$ variables to zero, then the system reduces to a square system of m linear equations, which can be solved via Gaussian elimination. The chosen variables for which we solve the system are called `basic variables', while those variables that we set to zero are called 'non-basic variables'. Solving the m linear equations in the m basic variables can lead to three possible outcomes. First, we may detect that the system has no solution. Second, a solution, called basic solution, may exist and it also satisfies the remaining constraints of the LP, namely the sign restrictions $x \geq 0$. In this case the solution is called a 'basic feasible solution'. Third, the solution to the linear system may entail a negative value for at least one variable and thus violate the sign restriction. Such a solution is called 'basic infeasible'. Two basic solutions are called adjacent if their respective sets of basic variables have all but one element in common. The next theorem explains why the basic feasible solutions are of central importance to the linear program.

Theorem If the problem (LP) has a non-empty feasible region, then there is at least one basic feasible solution. If the problem (LP) has an optimal solution then it has the following properties.

1. At least one optimal solution is a basic feasible solution.
2. If (LP) has a unique solution, then this optimal solution is basic feasible.
3. If a basic feasible solution x^* has an objective function value that is not larger than the objective function values at all its adjacent basic feasible solutions, then x^* is a solution of (LP).
4. If the feasible region is bounded and a basic feasible solution x^* has an objective function value that is strictly less than the objective function value at all its adjacent basic feasible solutions, then x^* is the unique solution of (LP). □

This theorem provides the foundation for the basic approach of the simplex method. According to the first statement of the theorem, if an optimal solution exists then there must be a basic feasible solution that is optimal. Thus, for solving the problem (LP) it suffices to only examine basic feasible solutions. In principle we could now find a solution to the problem (LP) by simply calculating all its basic solutions and then choosing a basic feasible solution with the smallest objective function value. We would not want to do this in practice, however, since the number of possible basic solutions is $\binom{n}{m}$ and thus is huge for many applications. The simplex method prescribes a smart way of searching through the basic feasible solutions. Starting from some basic feasible solution, the simplex searches for another basic feasible solution with a lower objective function value. From a computational standpoint it is much quicker to examine only adjacent basic feasible

solutions. The information we have from having solved a linear system in, for example, the variables x_1, x_2, x_3, greatly simplifies finding a solution in the variables x_2, x_3, x_5. Therefore, the simplex considers only adjacent feasible solutions and chooses one of them by exchanging one basic variable against one non-basic variable and solving the resulting system of linear equations. On most (but not all) steps of the method the objective function value decreases. This process repeats itself until the method reaches a basic feasible solution without any adjacent basic feasible solutions having a lower objective function value. The third statement of the theorem (which is a special version of the convex programming property for linear programs) then ensures that the simplex method has found an optimal solution.

We illustrate the basic ideas underlying the simplex method in the following example.

Example 5 Consider the following linear programming problem.

$$\begin{array}{ll} \max_{x_1,x_2} & 3x_1 + 4x_2 \\ \text{s.t.} & x_1 + 2x_2 \leq 10 \\ & x_1 + x_2 \leq 8 \\ & x_2 \leq 4 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{array}$$

Linear programming problems with two variables allow a beautiful graphical representation, which greatly helps us to gain some intuition for the simplex method. Figure 3 shows the feasible region of this linear programming problem.

This problem has three inequality constraints and two sign restrictions. The introduction of three so-called slack variables transforms the inequalities into equations. This introduction of new variables is just one of several simple transformations that allow us to rewrite any linear programming problem into a linear program in standard form; see Dantzig (1963) or many other linear programming books. Here we obtain the following linear program.

$$\begin{array}{llccccccc} \min_{x_1,x_2,x_3,x_4,x_5} & -3x_1 & - & 4x_2 & & & & & \\ \text{s.t.} & x_1 & + & 2x_2 & + & x_3 & & & = 10 \\ & x_1 & + & x_2 & + & & x_4 & & = 8 \\ & & & x_2 & + & & & x_5 & = 4 \\ & & & x_1, & x_2, & x_3, & x_4, & x_5 & \geq 0 \end{array}$$

This linear program has $n = 5$ variables and $m = 3$ constraints.

Table 3 lists all $\binom{5}{3} = 10$ possible combinations of basic variables, the corresponding basic solution (if it exists), whether this solution is feasible, and the objective function values $z = -3x_1 - 4x_2$ for the basic feasible solutions. For example, the basic solution (4, 4, −2, 0, 0) is obtained by setting $x_4 = x_5 = 0$ and then solving the remaining three equations

$$x_1 + 2x_2 + x_3 = 10, \;\; x_1 + x_2 = 8, \;\; x_2 = 4,$$

in the three basic variables x_1, x_2, x_3. This basic solution is infeasible since $x_3 = -2$ violates the non-negativity constraint on this variable. The basic variables x_1, x_3, x_4 lead to the three equations

$$x_1 + x_3 = 10, \;\; x_1 + x_4 = 8, \;\; 0 = 4,$$

which obviously have no solution. We can relate the nine basic solutions to points in the graph of the feasible region in Figure 3. The five feasible solutions are represented by disks while the four infeasible solutions are given by circles. We can easily identify the coordinates of the nine indicated points with the values of the original variables x_1 and x_2 in the nine basic solutions. But where are the later introduced slack variables? The values of these variables at a basic solution show us where the corresponding point in the figure is in relation to the three constraints. The basic solution (4,4,−2,0,0) is represented by the point (4,4) in the graph. This point lies on the lines representing the second and third constraints, since $x_4 = x_5 = 0$, and outside the first constraint, since $x_3 < 0$.

The simplex method quickly solves this problem. Starting from the basic feasible solution that corresponds to the origin in Figure 3 it moves through the basic feasible solutions ('BFS') listed in Table 4 to find the optimal basic feasible solution (6,2,0,0,2). Figure 4 illustrates the steps of the simplex method. Starting from the point (0,0) it moves upwards to the point (0,4) with an objective function value of $z = -16$, then to (2,4) with $z = -22$ and finally to (6,2) with $z = -26$. The basic feasible solution corresponding to this last point has a strictly lower objective function value than both its adjacent basic feasible solutions at (2,4) and (8,0) and hence it must be the unique optimal solution.In Figure 4 only the visited points are indicated by disks and the iso-objective function lines for the values $-z$ of the original objective function (from the maximization problem) at these points. □

Table 3 All basic solutions

Basic variables	Basic solution	Property	z
x_1, x_2, x_3	(4, 4, −2, 0, 0)	Infeasible	–
x_1, x_2, x_4	(2, 4, 0, 2, 0)	Feasible	−22
x_1, x_2, x_5	(6, 2, 0, 0, 2)	Feasible	−26
x_1, x_3, x_4	–	No Solution	–
x_1, x_3, x_5	(8, 0, 2, 0, 4)	Feasible	−24
x_1, x_4, x_5	(10, 0, 0, −2, 4)	Infeasible	–
x_2, x_3, x_4	(0, 4, 2, 4, 0)	Feasible	−16
x_2, x_3, x_5	(0, 8, −6, 0, −4)	Infeasible	–
x_2, x_4, x_5	(0, 5, 0, 3, −1)	Infeasible	–
x_3, x_4, x_5	(0, 0, 10, 8, 4)	Feasible	0

Table 4 Iterates of the simplex method

Basic Variables	BFS	z
x_3, x_4, x_5	(0, 0, 10, 8, 4)	0
x_2, x_3, x_4	(0, 4, 2, 4, 0)	−16
x_1, x_2, x_4	(2, 4, 0, 2, 0)	−22
x_1, x_2, x_5	(6, 2, 0, 0, 2)	−26

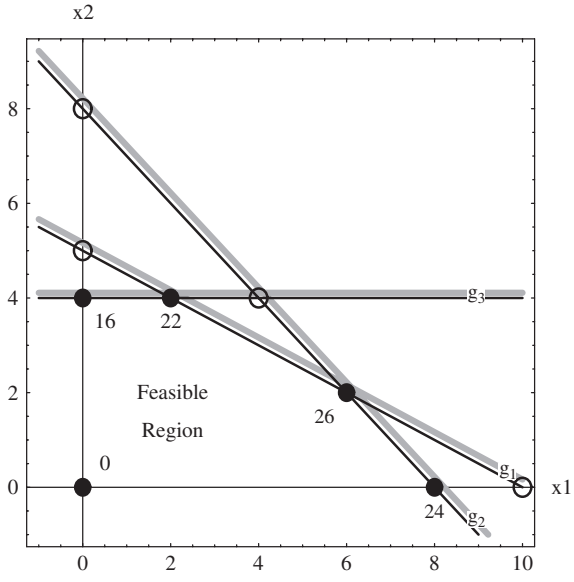

Figure 3 Feasible region of the (LP)

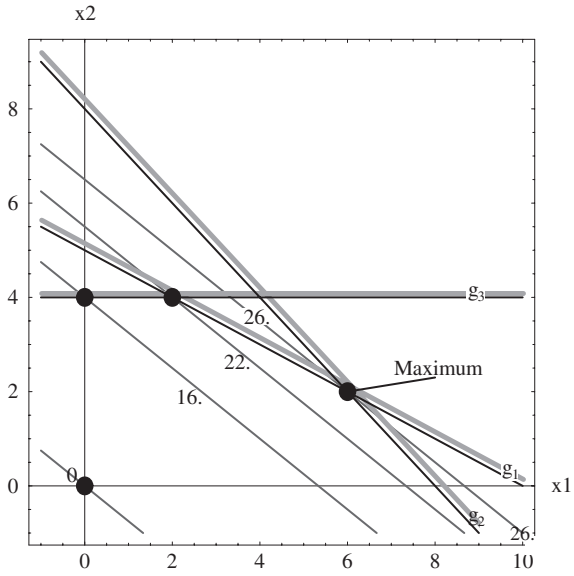

Figure 4 Solving the (LP)

We have conveyed only the basic principle of the simplex method for solving linear programming problems. Of course, an efficient and robust implementation of the simplex algorithm must address many technical details; see Fletcher (1987) or once again Nocedal and Wright (2006). The classical reference for the theory of the simplex method is the book by Dantzig (1963).

The simplex method is highly efficient on virtually all practical problems, but there do exist pathological problems on which it shows very poor performance. In these worst-case problems the running time of the simplex method grows exponentially in the dimension of the problems. (In a nutshell, the method visits far too many basic feasible solutions until it finds the optimal one.) Therefore, the

simplex method is of exponential complexity. Although these examples are irrelevant for practical applications, they generated interest in the development of different algorithms that would show better worst-case running times, in particular, that would have running times that grow only polynomially in the size of the problems. The first linear programming algorithm with polynomial complexity was the ellipsoid method of Khachiyan (1979). Although this method has polynomial complexity it is useless for actual computations, and apparently there has never been a serious practical implementation. The projective algorithm of Karmarkar (1984) started what is nowadays called the 'interior-point revolution'. This algorithm both has polynomial complexity and is of practical use, although the initial claims about its supposedly stellar practical performance were shown to be outrageous. The projective algorithm has long been superseded by more efficient methods, and the field of interior-point methods remains an active area of research to this day.

4.2 The idea of interior-point methods

Primal-dual methods are an important subclass of interior-point methods for solving constrained optimization problems. Here we give a basic outline of such a method for solving linear programs. The Karush–Kuhn–Tucker conditions for a linear programming problem in standard form are as follows.

$$A^\top \lambda + s = c \tag{8}$$

$$Ax = b \tag{9}$$

$$x_i s_i = 0, \quad i = 1, 2, \ldots, n \tag{10}$$

$$x \geq 0 \tag{11}$$

$$s \geq 0 \tag{12}$$

These first-order conditions characterize both the optimal solution of the given linear program and of its dual. (See Dantzig, 1963, or any book on linear programming for the definition of the dual of a linear program.) That fact motivates the name 'primal-dual' method.

Interior-point methods (approximately) solve a sequence of perturbed problems. Consider the following perturbation of the first-order conditions.

$$A^\top \lambda + s = c \tag{13}$$

$$Ax = b \tag{14}$$

$$x_i s_i = \mu, i = 1, 2, \ldots, n \tag{15}$$

$$x > 0 \tag{16}$$

$$s > 0 \tag{17}$$

Observe that the complementarity condition (10) has been replaced by the equations (15) for some positive scalar $\mu > 0$. Assuming that a solution $(x^{(0)},\lambda^{(0)},s^{(0)})$ to this system is given for some initial value of $\mu^{(0)} > 0$, interior-point methods decrease the parameter μ and thereby generate a sequence of points $(x^{(k)}, \lambda^{(k)}, s^{(k)})$ that satisfy the non-negativity constraints on the variables strictly, $x^{(k)} > 0$ and $s^{(k)} > 0$. This property led to the name 'interior-point' method. In the limit, as μ is decreased to zero, a point satisfying the original first-order conditions is reached. The set of solutions to the perturbed system,

$$C = \{x(\mu), \lambda(\mu), s(\mu) | \mu > 0\}$$

is called the central path.

The method is rather intuitive at this point. Given an iterate $(x^{(k)}, \lambda^{(k)}, s^{(k)})$ for some parameter value $\mu^{(k)}$ decrease the parameter to $\mu^{(k+1)} < \mu^{(k)}$ and determine the next iterate $(x^{(k+1)}, \lambda^{(k+1)}, s^{(k+1)})$. Implementing this method requires handling of many details. For example, it is often difficult to find a feasible starting point $(x^{(0)},\lambda^{(0)},s^{(0)})$ of the perturbed system. The most important step in the method is to solve the system (13)–(15) in each iteration (while maintaining the inequalities (16, 17)). Observe that this system consists of $2n + m$ linear and bilinear equations in as many variables. We can apply a nonlinear equations solver to this model. A popular approach is to use Newton's method for solving nonlinear systems of equations; see Judd (1998) or Miranda and Fackler (2002). The difficulty is to maintain the strict non-negativity constraints on the variables $x^{(k+1)}$ and $s^{(k+1)}$. An alternative approach for solving the parameterized system of equations is the application of path-following methods; see Nocedal and Wright (2006). Intuitively we can think of interior-point methods to be closely related to homotopy continuation methods for solving nonlinear systems of equations; see Allgower and Georg (1979).

Example 6 We revisit the linear program from Example 5. The perturbed first-order conditions (13)–(17) for this (LP) are as follows.

$$\begin{aligned}
y_1 + y_2 + s_1 + 3 &= 0 \\
2y_1 + y_2 + y_3 + s_2 + 4 &= 0 \\
y_1 + s_3 &= 0 \\
y_2 + s_4 &= 0 \\
y_3 + s_5 &= 0 \\
x_1 + 2x_2 + x_3 - 10 &= 0 \\
x_1 + x_2 + x_4 - 8 &= 0 \\
x_2 + x_5 - 4 &= 0 \\
x_i \cdot s_i = \mu, &\quad i = 1, 2, \ldots, n \\
x_1, x_2, \ldots, x_5 &> 0 \\
s_1, s_2, \ldots, s_5 &> 0
\end{aligned}$$

Table 5 displays the values for the variables $x_1,x_2,\ldots, x_5$ at some points on the central path for small values of μ. We observe how the central path moves through the interior of the feasible region, see Figure 3, and converges to the optimal solution as $\mu \to 0$. □

Table 5 Solutions $x^*(\mu)$ for small μ

μ	$x_1(\mu)$	$x_2(\mu)$	$x_3(\mu)$	$x_4(\mu)$	$x_5(\mu)$
1	5.9775	1.5451	0.9323	0.4774	2.4549
0.5	6.0305	1.7311	0.5073	0.2384	2.2689
0.1	6.0029	1.9478	0.1014	0.0492	2.0522
0.01	6.0000	1.9950	0.0100	0.0050	2.0050
0	6	2	0	0	2

By now the conceptual differences between the simplex method and interior-point methods are transparent. In geometric terms, the simplex method moves on specific points around the boundary of the feasible region until it finds a corner point corresponding to an optimal basic feasible solution. Interior-point methods move through the interior (or some methods even through the exterior) of the feasible region but they do not move within the boundary. Instead they approach the boundary only in the limit. In computational terms, the typical iteration of an interior-point method is relatively expensive to compute but can make significant progress towards the solution. Conversely, an iteration of the simplex method is relatively inexpensive but the method often requires a larger number of iterations.

Obviously the question arises of which of these two basic approaches is better for solving linear programs in practice. The answer depends very much on the nature of the problem. Currently the best available computer programs are efficient implementations of the dual simplex method (a special variant of the described standard simplex method) and primal–dual interior-point methods. Simplex method computer programs are usually faster on problems of small or medium size (say, of fewer than a million variables and constraints) while interior-point methods tend to do better on many but certainly not all large problems. If the user has significant prior information about the optimal solution, such as a good initial guess for an optimal basic feasible solution, then the simplex method is often much faster. The reason for this is that the simplex method is much easier to 'warm-start' than interior-point methods. In summary, interior-point methods and the simplex method are both important and useful algorithms for solving linear programs in practice.

Before we turn to interior-point methods for nonlinear optimization problems, we outline the basic concepts of another class of optimization algorithms. Penalty methods are quite intuitive and some of their ideas are relevant for interior-point methods but they are also of interest on their own.

4.3 Penalty methods

The basic idea of penalty methods is to replace the constrained optimization problem (NLP) by an unconstrained optimization problem and to solve the new problem instead. The objective function for the new unconstrained problem is the original objective plus a new term for each constraint. The new term is zero when the original constraint is satisfied but is positive if the original constraint is violated. The simplest and perhaps most intuitive penalty function is the quadratic penalty function.

To start we consider a nonlinear optimization problem with only equality but no inequality constraints,

$$\min_{x \in \mathbb{R}^n} f(x)$$
$$\text{s.t.} h_j(x) = 0 \;\; j \in E.$$

For such a problem we can define a penalty function

$$Q(x;\mu) = f(x) + \mu\left(\sum_{j \in E} h_j^2(x)\right)$$

with a penalty parameter $\mu > 0$. The idea of the penalty function method is to minimize the function Q for increasing values of μ. Observe that the function Q inherits its differentiable properties from the functions f and h_j, $j \in E$, of the original problem, and so we can use unconstrained optimization methods for minimizing $Q(x;\mu)$. In addition, as we generate a sequence $\mu^{(k)}$, $k = 0,1,2,\ldots$, we can use the previously calculated minimizer $x^{(k)}(\mu^{(k)})$ as initial guesses for the problem with $\mu^{(k+1)}$. This intuitive approach has a strong theoretical foundation, as the following theorem reveals; see Nocedal and Wright (2006).

Theorem Consider a sequence $\{\mu^{(k)}\}$ of penalty parameters with $\mu^{(k)} \to \infty$. Suppose that $x^{(k)}$ is the exact global minimizer of $Q(x;\mu_k) = f(x) + \mu(\sum_{j \in E} h_j^2(x))$. Then every limit point x^* of the sequence $\{x^{(k)}\}$ is a global solution of the (NLP). □

Although this result is nice from a theoretical viewpoint, it does not directly apply to practical applications. Of course, we typically cannot determine the exact minimizer of the penalty function and have to account for errors in the numerical approximation. The discussion in Nocedal and Wright (2006) shows that things get more complicated in practice once we allow for approximation errors. In addition, the penalty function may have many other stationary points that are not global or even local minimizers. The penalty function may even be unbounded if the penalty parameter μ is too small. At the other extreme, for very large values of μ the unconstrained minimization problem becomes more difficult, and the Hessian of Q gets ill-conditioned. All kinds of numerical problems arise that need to be carefully addressed in robust computer implementations of the quadratic penalty method; see Nocedal and Wright (2006).

For the general problem (NLP) with inequality and equality constraints we can define the penalty function as

$$Q(x;\mu) = f(x) + \mu\left(\sum_{i \in I} (\max(-g_i(x), 0))^2 + \sum_{j \in E} h_j^2(x)\right).$$

Now, however, things get more complicated since Q will typically not be twice differentiable. As a result the new unconstrained problem becomes more difficult to solve.

In addition to the quadratic penalty method several other such approaches exist and are used in practice. Nocedal and Wright (2006) describe non-differentiable penalty functions and the augmented Lagrangian method. Here we finish our discussion with an illustration of the quadratic approach.

Example 7 Consider a simple example of the classical portfolio optimization problem (Markowitz, 1952). An investor wants to allocate her entire wealth across

three securities with respective expected returns of 4 per cent, 8 per cent and 12 per cent. If she invests the respective portions x_1,x_2,x_3 in the three assets, then the variance of such a portfolio is $x_1^2 + 5x_2^2 + 3x_2x_3 + 10x_3^2$. The investors wants to minimize this variance under the condition that the expected return of her portfolio is at least 9 per cent. To simplify this illustration of the quadratic penalty method we exploit the fact that at the optimal solution the lower bound on the expected return is binding and thus write the investor's portfolio allocation problem as a nonlinear optimization problem with only equality constraints.

$$\begin{aligned} \min_{x_1,x_2,x_3} \; & x_1^2 + 5x_2^2 + 3x_2x_3 + 10x_3^2 \\ \text{s.t.} \; & x_1 + x_2 + x_3 - 1 = 0 \\ & 4x_1 + 8x_2 + 12x_3 - 9 = 0 \end{aligned}$$

The quadratic penalty function for the investor's portfolio optimization problem is

$$\begin{aligned} Q(x;\mu) = {} & x_1^2 + 5x_2^2 + 3x_2x_3 + 10x_3^2 \\ & + \mu\left((x_1 + x_2 + x_3 - 1)^2 + (4x_1 + 8x_2 + 12x_3 - 9)^2\right). \end{aligned}$$

We can easily solve the unconstrained problem with the basic Newton method as described in Section 3.1.2. Table 6 shows the solution to the unconstrained minimization of the penalty function for increasing values of μ. □

Table 6 Solutions $x^*(\mu)$ for large μ

μ	$x_1^*(\mu)$	$x_2^*(\mu)$	$x_3^*(\mu)$
1	0.78547	0.30659	0.26008
10	0.42484	0.35361	0.36870
100	0.21880	0.37632	0.42571
1000	0.18852	0.37962	0.43403
10000	0.18535	0.37996	0.43490
∞	0.185	0.38	0.435

Observe that the nonlinear optimization problem in this example has a quadratic objective function and linear constraints. Such optimization problems constitute a special and important subclass of problems called quadratic programs. Their special properties give rise to efficient solution methods, and we would not want to solve large quadratic programs with a penalty method. Nocedal and Wright (2006) present several algorithms for quadratic programming. Since solving quadratic programs is comparatively easy, an integral part of some algorithms for more general nonlinear optimization problems, such as the sequential quadratic programming methods, is to repeatedly solve quadratic programs that are derived as approximations for the more general problem.

4.4 The logarithmic barrier method

Logarithmic barrier methods are a particular type of interior-point methods for the solution of nonlinear optimization problems. We illustrate the basic idea of these methods for an inequality-constrained minimization problem.

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} & f(x) \\ \text{s.t.} & g_i(x) \geq 0 \quad i \in I. \end{array}$$

We can combine the objective function and the constraints and define a penalty function for this optimization problem by

$$P(x;\mu) = f(x) - \mu \sum_{i \in I} \ln g_i(x),$$

where $\mu > 0$ is called the barrier parameter and the expression $\sum_{i \in I} \ln g_i(x)$ is called a logarithmic barrier function. Each logarithmic term $-\ln g_i(x)$ tends to infinity as x approaches the boundary given by $g_i(x) \geq 0$ from the interior of the feasible region. This effect of the logarithmic terms will decrease as the barrier parameter μ becomes smaller. The idea of the logarithmic barrier method is now to let the parameter μ converge to zero. Under some conditions the optimal solution $x^*(\mu)$ of the unconstrained optimization problem $\min_{x \in \mathbb{R}^n} P(x;\mu)$ converges to the optimal solution of the original constrained optimization problem as μ tends to zero. Note that the logarithm ensures that $g_i(x^*(\mu)) > 0$ for all $\mu > 0$, that is, the solution to the unconstrained minimization problem is in the strict interior of the original constraints. This property represents a crucial distinction between this variant of an interior-point method and an active-set method such as the simplex method, which always tracks the set of binding constraints at a given iterate.

Observe that the first-order conditions for the penalty function problem are given by

$$\nabla_x P(x;\mu) = \nabla f(x) - \sum_{i \in I} \frac{\mu}{g_i(x)} \nabla g_i(x) = 0.$$

Now define for all $i \in I$

$$\nu_i(\mu) := \frac{\mu}{g_i(x)}.$$

Note that since $\mu > 0$ by definition we have that $\nu_i(\mu) > 0$. Thus, at a stationary point of the penalty function the following conditions hold.

$$\nabla f(x) - \sum_{i \in I} \nu_i \nabla g_i(x) = 0 \;\; g_i(x) - s_i = 0 \;\; \text{for all} \;\; i \in I$$
$$\nu_i s_i = \mu \;\; \text{for all} \;\; i \in I \nu_i > 0 \;\; \text{for all} \;\; i \in I s_i > 0 \;\; \text{for all} \;\; i \in I.$$

This set of conditions is just the primal-dual interior-point conditions for our original problem. We see that conditions (13)–(17) are just the specialization of these conditions for the linear programming model. And just like in the illustration of the section 4.2 we are interested in taking the parameter μ to zero. Unfortunately we do not have the space here to properly state a formal theorem. To make a long story short, under a few additional technical conditions, most notably second-order conditions of optimality, the following statements hold for a local solution x^* at which the KKT conditions are satisfied for some Lagrange multipliers ν^*.

1. The local minimizer $x^*(\mu)$ of $P(x;\mu)$ in some neighbourhood of x^* with $\lim_{\mu \downarrow 0} x^*(\mu) = x^*$ uniquely defines a continuously differentiable vector function $x^*(\mu)$ for all sufficiently small μ.

2. The function $x^*(\mu)$ yields Lagrange multipliers $\nu(\mu)$ satisfying $\lim_{\mu \downarrow 0} \nu(\mu) = \nu^*$ where $\nu^* g_i(x^*) = 0$.

An algorithm for solving the constrained problem is apparent now. For a given value of μ solve the unconstrained optimization problem with the objective function P. Then reduce μ stepwise to zero and follow the path of solutions $x^*(\mu)$. In the limit we can find the local solution x^* of the original problem. While this approach works in principle, it entails various difficulties. For example, the Hessian matrix of P becomes ill-conditioned for small values of μ. For this and many other technical issues see Gould and Leyffer (2002). Here we just illustrate the fundamental idea with an example.

Example 8 We revisit the consumer's utility maximization problem from Example 3 once again. The consumer has a utility function $u(x_1, x_2, x_3) = \sqrt{x_1} + 2\sqrt{x_2} + 3\sqrt{x_3}$ over three goods and faces the budget constraint $x_1 + x_2 + x_3 \leq 1$. We formulate the consumer's problem as the constrained minimization problem

$$\begin{array}{ll} \min_{x_1, x_2, x_3} & -\left(\sqrt{x_1} + 2\sqrt{x_2} + 3\sqrt{x_3}\right) \\ \text{s.t.} & 1 - x_1 - x_2 - x_3 \geq 0. \end{array}$$

The unconstrained function including a logarithmic barrier function for this minimization problem is

$$P(x_1, x_2, x_3; \mu) = -(\sqrt{x_1} + 2\sqrt{x_2} + 3\sqrt{x_3}) - \mu \ \ln \ (1 - x_1 - x_2 - x_3).$$

Table 7 displays solutions to this unconstrained problem for a few values of μ. Note that as $\mu \to 0$ the optimal solution approaches the optimal solution of the original utility maximization problem.

In all our examples so far we ignored the sign restriction of the variables. We could do that since the utility functions exhibit an Inada property, that is, $\lim_{x_i \to 0} \frac{\partial u}{\partial x_i} = +\infty$, and so we hope that a solver starting at a strictly positive solution will only iterate through such solutions (although we have to be careful in practice). But, of course, we can easily take the non-negativity constraints explicitly into account and consider the following problem.

$$\begin{array}{ll} \min_{x_1, x_2, x_3} & -\left(\sqrt{x_1} + 2\sqrt{x_2} + 3\sqrt{x_3}\right) \\ \text{s.t.} & 1 - x_1 - x_2 - x_3 \geq 0 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \\ & x_3 \geq 0 \end{array}$$

Note that the condition (LICQ) is always satisfied since not all four constraints can be satisfied simultaneously. As long as three constraints are binding (LICQ) holds. The unconstrained function including a logarithmic barrier function for this minimization problem is

$$\begin{aligned} P(x_1, x_2, x_3; \mu) = & -(\sqrt{x_1} + 2\sqrt{x_2} + 3\sqrt{x_3}) \\ & - \mu(\ln(1 - x_1 - x_2 - x_3) + \ln(x_1) + \ln(x_2) + \ln(x_3)). \end{aligned}$$

Table 8 displays solutions to this unconstrained problem for a few values of μ. Again we observe that $x^*(\mu) \to x^*$ as $\mu \to 0$. □

Table 7 Solutions $x^*(\mu)$ for small μ

μ	$x_1^*(\mu)$	$x_2^*(\mu)$	$x_3^*(\mu)$
1	0.0421124	0.168450	0.379012
0.5	0.0547198	0.218879	0.492478
0.1	0.0677112	0.270845	0.609401
0.01	0.0710478	0.284191	0.639430
0.005	0.0712379	0.284952	0.641141

Table 8 Solutions $x^*(\mu)$ for small μ

μ	$x_1^*(\mu)$	$x_2^*(\mu)$	$x_3^*(\mu)$
1	0.219696	0.270563	0.331757
0.5	0.195664	0.278956	0.389721
0.1	0.124696	0.286370	0.543846
0.01	0.0789861	0.285758	0.630009
0.005	0.0753165	0.285727	0.636309

Strangely enough, some of the modern and best interior-point algorithms are based on work predating Karmarkar (1984). For example, Frisch (1955) had already proposed an interior-point method based on logarithmic barrier functions for solving linear programs. A full early history with many results on barrier functions is Fiacco and McCormick (1968).

4.5 Sequential quadratic programming

Sequential quadratic programming (SQP) methods are among the most effective constrained optimization techniques, particularly when nonlinear constraints are present. These algorithms belong to the class of active-set methods that keep track of the binding constraints at each step. For a description of the basic ideas we consider a minimization problem with only equality constraints. (But these methods are much more widely applicable.)

$$\min_{x \in \mathbb{R}^n} f(x) \text{ s.t. } h_j(x) = 0, \ j \in E. \tag{18}$$

The KKT conditions for this problem are as follows.

$$\nabla f(x) - \sum_{j \in E} \lambda_j \nabla h_j(x) = 0, \tag{19}$$

$$h_j(x) = 0, \ j \in E. \tag{20}$$

These conditions are a system of $n + m$ nonlinear equations in the n variables x and the m Lagrange multipliers λ. Newton's method for solving nonlinear equations is now a natural approach for solving this system. The Jacobian of the left-hand side of the system (19)–(20) is given by

$$\begin{bmatrix} H(x) - \sum_{j \in E} \lambda_j H_j(x) & -A(x)^\top \\ A(x) & 0 \end{bmatrix},$$

where the matrix $A(x)^\top = [\nabla h_1(x), \ldots, \nabla h_J(x)]$ is the collection of the gradient vectors of all constraints $h(x) = (h_j(x))_{j \in E = \{1,2,\ldots,J\}}$. The matrix $H_j(x)$ denotes the Hessian matrix of the constraint function h_j at the point x. For a given point $(x^{(k)},\lambda^{(k)})$ the Newton step is then determined by the linear system

$$\begin{bmatrix} H(x^{(k)}) - \sum_{j \in E} \lambda_j^{(k)} H_j(x^{(k)}) & -A(x^{(k)})^\top \\ A(x^{(k)}) & 0 \end{bmatrix} \begin{bmatrix} s_x^{(k)} \\ s_\lambda^{(k)} \end{bmatrix} = -\begin{bmatrix} \nabla f(x^{(k)}) - \sum_{j \in E} \lambda_j^{(k)} \nabla h_j(x^{(k)}) \\ h(x^{(k)}) \end{bmatrix}$$

resulting in the new iterate $(x^{(k+1)}, \lambda^{(k+1)}) = (x^{(k)} + s_x^{(k)}, \lambda^{(k)} + s_\lambda^{(k)})$. Note that this last system is equivalent to the following linear system.

$$\begin{bmatrix} H(x^{(k)}) - \sum_{j \in E} \lambda_j^{(k)} H_j(x^{(k)}) & A(x^{(k)})^\top \\ A(x^{(k)}) & 0 \end{bmatrix} \begin{bmatrix} s_x^{(k)} \\ -\lambda^{(k+1)} \end{bmatrix} = -\begin{bmatrix} \nabla f(x^{(k)}) \\ h(x^{(k)}) \end{bmatrix} \tag{21}$$

Now consider the following quadratic optimization problem (QP).

$$\begin{aligned} &\min_{s \in \mathbb{R}^n} \ \frac{1}{2} s^\top \left(H(x) - \sum_{j \in E} \lambda_j H_j(x) \right) s + \nabla f^\top(x) s \\ &\text{s.t.} \quad A(x)s + h(x) = 0 \end{aligned}$$

The left-hand side of the constraints are a first-order (Taylor) approximation of the constraint function h of the original optimization problem. The objective function of (QP) is a second-order approximation of the difference $f(x + s) - f(x)$. The KKT conditions for the problem (QP) are as follows.

$$\left(H(x) - \sum_{j \in E} \lambda_j H_j(x) \right) s + \nabla f(x) - A^{\mathrm{T}}(x)\nu = 0 \tag{22}$$

$$A(x)s + h(x) = 0 \tag{23}$$

Observe that these KKT conditions at a point $(x^{(k)}, \lambda^{(k)})$ are equivalent to (21). Solving the first-order conditions of the original optimization problem with Newton's method is, under some technical conditions, equivalent to solving the quadratic optimization problem (QP). A Newton step at a given point $(x^{(k)}, \lambda^{(k)})$ is the same as solving the (QP) at this point. The idea of SQP methods is now to repeatedly solve this quadratic problem to generate a sequence of iterates that converges to a local solution of the original problem. Various good methods for solving quadratic optimization problems exist and can be applied to the problem (QP). Moreover, when combined with line search or trust region methods the approach has useful global convergence properties. Gould and Leyffer (2002) and Nocedal and Wright (2006) discuss details of line search and trust region SQP methods.

5 Global optimization

We emphasized repeatedly that most practical algorithms for solving nonlinear optimization problems search for a solution only to the (necessary) first-order conditions, that is, they search for a local solution. Unless we are solving a convex programming problem or an unconstrained minimization problem of a convex function, we often cannot be sure that a computed local solution is indeed an approximate solution to the problem at hand; recall Example 2. Only occasionally other additional knowledge, perhaps some particular property of an underlying economic model, may assure us that we found an optimal solution. Obviously it would be helpful to have methods for general non-convex problems that may not, or are at least less likely to, get stuck in only locally optimal solutions. Here we lay out two approaches for global optimization. We describe the basic ideas of some popular metaheuristics and, subsequently, the very promising area of research in polynomial optimization, which is likely going to produce powerful tools for economic problems.

5.1 Metaheuristics

Metaheuristics provide a general framework and basic guidelines for developing specific heuristics for solving optimization problems. While the underlying principles are very general, typically a method must be carefully tailored in order to obtain an effective algorithm for the special problem at hand. Most metaheuristics were originally developed for solving discrete optimization problems, such as integer or combinatorial problems. Their principal ideas can also be applied to come up with heuristics for continuous nonlinear optimization problems.

The central problem of most nonlinear optimization methods is the possibility of getting stuck at a locally optimal solution. Many methods allow only for iterative steps that lead to an improvement in the objective function value, but, for an exception, see the discussion on nonmonotone techniques in Conn, Gould and Toint (2000) and Nocedal and Wright (2006). Such methods cannot get away from a locally optimal solution. In order to escape from such a local solution we must allow our search procedure, at least sometimes, to move into a non-improving search directions; that is, temporarily the objective function value of the sequence of iterates may increase (in a minimization problem). Three metaheuristics that are supposed to escape local solution are tabu search, simulated annealing, and genetic algorithms. The latter two methods are examples of stochastic approaches for optimization. Here we give a description of the basic ideas underlying these three methods and refer to Brandimarte (2006), Judd (1998) and the citations in those books for details.

5.1.1 Tabu search

The choice of non-improving search directions must be carefully managed to avoid repeatedly returning to a previously found optimal solution. Such cycling may occur if, after a non-improving step away from a local solution, the algorithm takes an improving step and immediately returns to the previously found local solution. A tabu search procedure imposes at every iteration a list of search directions that the algorithm is not allowed to pursue. For example, if the method just took a step in the direction $s^{(k)}$ then it may not be allowed to examine a neighbourhood of search

directions around $-s^{(k)}$ for the next few iterations. In order to avoid memory problems in practical implementations, the tabu list usually consists only of the most recent steps taken. Of course, many technical issues need to be addressed to obtain a robust and efficient algorithm. The treatment of these issues usually depends greatly on the specific problem.

5.1.2 Simulated annealing

Simulated annealing is another metaheuristic that helps an algorithm to escape from locally optimal solutions. Instead of choosing only iterates that decrease the objective function in a minimization problem, simulated annealing methods also accept with some probability new iterates that increase the objective. The probability of accepting an iterate $x^{(k+1)}$ if $f(x^{(k+1)}) > f(x^{(k)})$ is

$$e^{-\frac{f(x^{(k+1)})-f(x^{(k)})}{T}}$$

with a parameter $T>0$. Simulated annealing methods typically start out with a fairly large value for T and then decrease it to 0. Observe that for large values of T the heuristic is likely to accept non-decreasing iterates, and so it allows the method to explore the feasible region. As T decreases the probability of acceptance of non-decreasing iterates of a fixed size also decreases. In the limit $T \to 0$ the method allows only iterates that decrease the objective function value. The perhaps simplest rule for reducing T is to start from a high value T_0 and then to set

$$T_{l+1} = \alpha T_l \text{ for some } 0 < \alpha < 1.$$

The basic ideas of simulated annealing are derived from an analogy of minimization with the physical annealing process of slowly cooling metals in order to reach a strong low-energy solid state. This analogy motivates the particular probability function for accepting increasing iterates and explains why the parameter T is called the temperature of the process. The rule of decreasing T is analogously called the cooling schedule. The earliest applications of simulated annealing were combinatorial problems; see Kirkpatrick, Gelatt and Vecchi (1983) as well as Cerny (1985).

5.1.3 Genetic algorithms

Genetic algorithms are derived from the analogy of finding better and better solutions with the theory of biological evolution of selecting fitter and fitter members of a species. As a result the literature on genetic algorithms uses terminology from evolutionary biology. Iterates in tabu search and simulated annealing algorithms are a single point. Contrary to that, genetic algorithms work with a set ('generation') of several current points. A genetic algorithm constructs a sequence of such sets. In a given iteration the objective function is evaluated at the points in the set ('fitness of a member'). The method then chooses elements of the set in a probabilistic fashion in order to build new elements for the next set. Usually the probability of an element being chosen is the higher the better its objective function value. Several ways to construct new elements exist. A standard operation is the so-called crossover. Given two elements ('parents') $x^{(k)}$ and $y^{(k)}$ in the set the crossover operation leads to

$$x^{(k+1)} = \left(x_1^{(k)}, \ldots, x_l^{(k)}, y_{l+1}^{(k)}, \ldots, y_n^{(k)}\right), \tag{24}$$

$$y^{(k+1)} = \left(y_1^{(k)}, \ldots, y_l^{(k)}, x_{l+1}^{(k)}, \ldots, x_n^{(k)}\right), \tag{25}$$

where the method chooses some arbitrary break point l in the n-dimensional vectors. The idea behind crossover is to preserve some parts of the original elements and at the same time generate quite arbitrarily new elements ('children') that are far away from the original ones, and thereby to escape local solution. Another type of operation aimed at achieving this goal is to randomly exchange an element in a member $x^{(k)}$ by another value ('mutation'). While these approaches have proven useful in combinatorial optimization, it is quite apparent that they may run into severe difficulties for constrained problems. Many technical details must, therefore, be resolved before these ideas yield a useful heuristic approach for solving an optimization problem.

The monograph by Holland (1975) popularized genetic algorithms. The basic ideas of computer simulations of evolution are much older.

Any heuristic method derived from a metaheuristic will always be an ad hoc approach to the problem at hand. Just like the standard methods of nonlinear optimization presented in this article, they are not guaranteed to find the solution of a problem. And, while such heuristics have proven useful in discrete optimization, they are generally regarded as inferior to the modern standard optimization techniques for continuous optimization. An economist's first choice of a solution method for a continuous optimization problem, particularly when nonlinear constraints are present, should always be one of the standard methods.

5.2 Polynomial functions

A substantial number of prominent economic models involves only polynomial functions, equations or inequalities. Even problems that at first appear to be non-polynomial can sometimes be transformed into having only polynomial expression. For example, the first-order conditions for the standard log-utility maximization problem

$$\max_{x \in \mathbb{R}^n} \sum_{i=1}^{n} \ln(x_i) \quad \text{s.t.} \quad \sum_{i=1}^{n} p_i(x_i - \omega_i) = 0$$

for prices p_i and endowments ω_i, $i = 1,\ldots, n$, can be written in polynomial form,

$$1 - \lambda p_i x_i = 0, \quad i \in \{1, \ldots, n\}, \tag{26}$$

$$\sum_{i=1}^{n} p_i(x_i - \omega_i) = 0, \tag{27}$$

where λ denotes the Lagrange multiplier. Polynomial functions and equations can be analysed using tools from computational algebraic geometry (Cox, Little and O'shea, 1997). Global optimization with polynomials is an active field of research in mathematics; see, for example Lasserre (2001), Parrilo and Sturmfels (2003), and the book by Sturmfels (2002) and the citations therein. It is possible (at least in theory)

to compute all local minima of polynomial functions. Similarly, it is possible to compute all solutions to a polynomial system of equations. With further expected advances in the theory of polynomial optimization and ever increasing speed of modern computers, these tools will soon have an impact in economics. For first results see computation of general equilibria (new developments).

6 Popular optimization software in economics

This section covers software packages and modelling languages that are frequently used in economics to solve optimization problems. This list is by no means exhaustive, and many other software products for solving optimization problems exist.

Perhaps the most popular software for numerical work in economics is MATLAB (MATLAB is a registered trademark of The MathWorks, Inc.). Computational economics and finance textbooks such as Brandimarte (2006), Kendrick, Mercado and Amman (2006) and Miranda and Fackler (2002) use MATLAB to solve economic problems. Other popular packages include GAUSS (GAUSS is a registered trademark of Aptech Systems, Inc.) and Mathematica (Mathematica is a registered trademark of Wolfram Research, Inc.) All three languages offer solvers for nonlinear optimization problems, which are continuously enhanced to solve larger and more difficult problems. Here we just list a few features of these software packages.

MATLAB has an optimization toolbox containing routines for solving both unconstrained and constrained nonlinear optimization problems. Methods for unconstrained problems include quasi-Newton and trust region techniques. The solvers for constrained optimization include an SQP method. MATLAB also has specialized methods for nonlinear least square problems; however, most of these solvers are considered to be of only mediocre quality. Much better solvers in MATLAB are available through the NAG toolbox (NAG is a registered trademark of The Numerical Algorithms Group, Inc.) The NAG Foundation Toolbox provides access to the large set of numerical routines contained in the Fortran-based NAG Foundation Library, which contains routines for constrained and unconstrained optimization.

The high-level matrix programming language GAUSS includes an applications module for constrained optimization that uses an SQP method in conjunction with several line search methods or a trust region method. GAUSS has some specialized modules for constrained maximum likelihood problems. For Mathematica there exists a global optimization package, which contains various functions for optimization. These functions are designed to search for global optima for problems with hundreds of variables. The monograph by Bhatti (2000) comes with an optimization toolbox for Mathematica that includes all the methods presented in this article.

These high-level languages are popular in economics because they are easy to learn and quickly facilitate solving problems of moderate size. For larger problems with thousands or even hundreds of thousands of variables, however, they are not reliable and certainly too slow. Economists interested in solving large problems need to use alternative software. An excellent alternative is the use of algebraic modelling languages.

The General Algebraic Modeling System (GAMS) is a high-level modelling language designed for mathematical programming and optimization; see Rosenthal (2006) for a user's guide. GAMS consists of a language compiler and a family of integrated high-performance solvers. GAMS is tailored for complex, large-scale modelling applications, and allows the user to build large models. It has a long history of successful applications in economics, particularly in solving large-scale computable general equilibrium (CGE) models. AMPL (Fourer, Gay and Kernighan, 2003) is an algebraic modelling language for mathematical programming, which allows users to set up and solve a great variety of optimization problems. The user has access to many popular and sophisticated solvers.

An exciting environment for solving optimization problems is the Network-Enabled Optimization System (NEOS); see Czyzyk, Mesnierand and Moré (1998) and Ferris, Mesnier and Moré (2000). NEOS is an optimization site that allows users to submit optimization problems over the Internet. The user does not need to download any solver but can just send optimization problems to NEOS and choose from a list of solvers. NEOS has access to many of the most current and powerful optimization routines. NEOS returns a solution and some runtime statistics to the user. Unfortunately, NEOS has been largely ignored by many economists.

7 Mathematical programs with equilibrium constraints

Mathematical programs with equilibrium constraints (MPECs) are currently at the frontier of numerical analysis. Economic models that can be classified as 'leader-follower' games are examples of MPECs. Suppose that the economic variables can be partitioned into x, those chosen by the 'leader' (government, employer, market maker, mechanism designer, and so on), and y, those chosen by the 'followers' (taxpayers, employees, traders, and so on) or determined in equilibrium (such as price). Suppose that the leader's payoff is $f(x, y)$ and that the equilibrium value y given x is represented by a combination of inequality conditions, $c(x,y) \geq 0$, and complementarity constraints, $0 \leq y \perp F(x,y) \geq 0$, where $0 \leq y \perp F(x,y) \geq 0$ if and only if $0 \leq y$, $F(x,y) \geq 0$, and $y^T F(x,y) = 0$. Equality constraints can be added without difficulty. The constraints correspond to, for example, budget and incentive constraints, and the complementarity constraints model the optimality conditions of the followers including any Lagrange multipliers. Then the leader's problem and the corresponding equilibrium are given by the solution to the MPEC

$$\begin{array}{ll} \max_{x,y} & f(x,y) \\ \text{s.t.} & c(x,y) \geq 0 \\ & 0 \leq y \perp F(x,y) \geq 0. \end{array}$$

MPECs present many mathematical challenges; the constraints are non-convex and reformulations as standard nonlinear optimization problems violate fundamental stability assumptions. Despite these facts, nonlinear optimization methods applied to such reformulations have been successful at solving some MPECs. For example, Chen et al. (2006) solve MPECs derived from some large-scale electricity market models. But they also show the limitations of the nonlinear optimization approach, and advocate the development of robust algorithms for solving MPECs that directly exploit the structure of the complementarity constraints. The development of such methods is under way. The ability to solve large and complicated MPECs will

greatly enhance economic modelling in many areas and will likely make MPECs a key tool of computational economic analysis in the future.

See Also

- computation of general equilibria
- computation of general equilibria (new developments)
- computational methods in econometrics
- dynamic programming
- linear programming
- nonlinear programming
- operations research
- simplex method for solving linear programs

I am grateful for helpful discussions with Sven Leyffer and am indebted to Ken Judd, Annette Krauss, and in particular Che-Lin Su for detailed comments on earlier drafts. I also thank the editors Larry Blume and Steven Durlauf for a careful review of my initial submission.

Bibliography

Allgower, E.L. and Georg, K. 1979. *Introduction to Numerical Continuation Methods*. New York: John Wiley & Sons. Reprinted by SIAM Publications, 2003.

Bhatti, M.A. 2000. *Practical Optimization Methods: With Mathematica Applications*. New York: Springer-Verlag.

Brandimarte, P. 2006. *Numerical Methods in Finance and Economics: A MATLAB-Based Introduction*. New York: John Wiley & Sons.

Cerny, V. 1985. A thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications* 45, 41–51.

Chen, Y., Hobbs, B.F., Leyffer, S. and Munson, T.S. 2006. Leader–follower equilibria for electric power and NO_x allowances markets. *Computational Management Science* 4, 307–30.

Conn, A.R., Gould, N.I.M. and Toint, P.L. 2000. *Trust-Region Methods*. Philadelphia: SIAM.

Cox, D., Little, J. and O'shea, D. 1997. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. New York: Springer-Verlag.

Czyzyk, J., Mesnier, M.P. and Morè, J. 1998. The NEOS server. *IEEE Journal on Computational Science and Engineering* 5, 68–75.

Dantzig, G.B. 1949. Programming of inter-dependent activities II, mathematical model. *Econometrica* 17, 200–211. Also in Koopmans, T.C., ed. *Activity Analysis of Production and Allocation*. New York: John Wiley & Sons, 1951.

Dantzig, G.B. 1963. *Linear Programming and Extensions*. Princeton: Princeton University Press.

Ferris, M.C., Mesnier, M.P. and Moré, J. 2000. NEOS and Condor: solving nonlinear optimization problems over the Internet. *ACM Transactions on Mathematical Software* 26, 1–18.

Fiacco, A.V. and McCormick, G.P. 1968. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. New York: John Wiley & Sons, Inc. Reprinted by SIAM Publications, 1990.

Fletcher, R. 1987. *Practical Methods of Optimization*. Chichester: John Wiley & Sons.

Fourer, R., Gay, D.M. and Kernighan, B.W. 2003. *AMPL: A Modeling Language For Mathematical Programming*. Pacific Grove, CA: Brooks/Cole–Thomson Learning.

Frisch, R.A.K. 1955. The logarithmic potential method of convex programming. Technical Report, University Institute of Economics, University of Oslo, Norway.

Gould, N.I.M. and Leyffer, S. 2002. An introduction to algorithms for nonlinear optimization. In *Frontiers in Numerical Analysis*, ed. J.F. Blowey, A.W. Craig and T. Shardlow. Berlin, Heidelberg: Springer-Verlag.

Holland, J.H. 1975. *Adaptation in Natural and Artifical Systems*. Ann Arbor: University of Michigan Press.

Judd, K.L. 1998. *Numerical Methods in Economics*. Cambridge, MA: MIT Press.

Karmarkar, N. 1984. A new polynomial-time algorithm for linear programming. *Combinatorics* 4, 373–95.

Kendrick, D.A., Mercado, P.R. and Amman, H.M. 2006. *Computational Economics*. Princeton: Princeton University Press.

Khachiyan, L.G. 1979. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady* 20, 191–4.

Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. 1983. Optimization by simulated annealing. *Science* 220, 671–80.

Lasserre, J.B. 2001. Global optimization with polynomials and the problem of moments. *SIAM Journal of Optimization* 11, 796–817.

Markowitz, H.M. 1952. Portfolio selection. *Journal of Finance* 7, 77–91.

Miranda, M.J. and Fackler, P. 2002. *Applied Computational Economics and Finance*. Cambridge, MA: MIT Press.

Nocedal, J. and Wright, S.J. 2006. *Numerical Optimization*. New York: Springer.

Parrilo, P.A. and Sturmfels, B. 2003. Minimizing polynomial functions. *Algorithmic and Quantitative Real Algebraic Geometry, DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 60, 83–99.

Rosenthal, R.E. 2006. *GAMS – A User's Guide*. Washington, DC: GAMS Development Corporation. Online. Available at http://www.gams.com/docs/gams/GAMSUsersGuide.pdf, accessed 7 February 2007.

Simon, C.P. and Blume, L. 1994. *Mathematics for Economists*. New York: W. W. Norton.

Smale, S. 1976. A convergent process of price adjustment and global Newton methods. *Journal of Mathematical Economics* 3, 107–20.

Sturmfels, B. 2002. *Solving Systems of Polynomial Equations*, vol. 97, CBMS Regional Conference Series in Mathematics, Providence, RI: American Mathematical Society.