Appendix

Guide to MP

A1. Mathematical Programming

We mentioned in §1, that a mathematical program is to minimize (or maximize) a function f of n variables $x_1, \ldots, x_n$ over a set S :

$$f(x) \to \min,\ x \in S, \qquad \text{(A1.1)}.$$

The decision variables $x_1, \ldots, x_n$ are also called control, planning, or strategic variables. We write them in a column x, so the feasible region S is a subset of R^n, all columns of n real entries. The set S is often given by a system of constraints of the following three types: $g(x) = 0, g(x) \geq 0$, or $g(x) \leq 0$, where $g(x)$ is a function on R^n. Note that the same set S can be given by different systems of constraints. Theoretically, it can always be given by one constraint of the type $g(x) = 0$ as well as one constraint of the type $g(x) \leq 0$. Using a penalty method, all constraints can be eliminated.

The real-valued function f is called the objective function or *minimand*. It always can be arranged to be linear. Namely, given program (A1.1), we can introduce a new variable z and consider the equivalent problem

$$z \to \min,\ x \in S,\ f(x) - z \leq 0 \qquad \text{(A1.2)}$$

with $n + 1$ variables and the objective function being a linear form. The optimal values for (A1.1) and (A1.2) are the same.

In this generality, we cannot develop much of theory, but we can give a couple of definitions. We endow R^n with the Euclidean norm

$$|x| = \|x\|_2 = (x^T x)^{1/2}.$$

Definition A1.3. A point a in S is called a *local* (or *relative*) *optimum point*, or *local optimizer*, or a *locally optimal solution* if there is $\varepsilon > 0$ such that a is an optimal solution after we add the constraint $|x - a| \leq \varepsilon$.

Remark. If we used the $\| \ \|_1$- or $\| \ \|_\infty$-norm instead of the $\| \ \|_2$-norm, then the constraint $|x - a| \leq \varepsilon$ would be equivalent to a system of linear constraints. An advantage of the $\| \ \|_2$-norm is that it is smooth (all partial derivatives of every order exist). The choice of norm in Definition A1.3 does not matter because

$$\|e\|_2/n^{1/2} \leq \|e\|_\infty \leq \|e\|_1 \leq \|e\|_2 n^{1/2}$$

for all vectors e in R^n. ■

For example, any optimal solution (which can be called a global optimum point) is also a local optimum point. In linear programming, the converse is true, see A3 below.

Definition A1.4. A *feasible direction* at a point a in S is a nonzero vector b with the following property: there is $\varepsilon_0 > 0$ such that $a + b\varepsilon$ belongs to S for all ε in the interval $0 \leq \varepsilon \leq \varepsilon_0$. ■

Note that this concept is independent of the objective function. When $S = R^n$, every direction at every point is feasible.

Many methods in mathematical programming involve choosing a feasible direction at a point such that small movement away from the point in this direction improves the objective function. Then there is a line search to decide how far we should move until we get the next point in our iterative procedure. The choice of a feasible direction at a point x may depend on the values of the objective function and the functions in the constraints at x as well as on the values of derivatives. Some iterative methods also use the values at previous points, in which case the method starts with more than one initial point.

In parallel and genetic programming, "time" could be more complicated than the sequence $0, 1, \ldots$. In some cases, an arbitrary or random choice is involved in finding the next point; otherwise we have a deterministic algorithm.

Notice that iterative procedures in mathematical programming usually do not give exact optimal solutions. Moreover, the final answer is not always even feasible. One reason is that the exact answer could involve irrational numbers even if all data are rational.

Linear programs are exceptional in this respect, because theoretically the simplex method is guaranteed to give an exact optimal solution in finitely many pivot steps. But suppose the answer involves integers with 10,000,000,000 digits. Can we really compute all these digits? If so, can we use such an answer and is it worth our time to compute the exact answer? Does it make sense to compute many digits of the answer if the data are not precise?

The answer with 10^{10} digits may look somewhat far-fetched, because an optimal solution for a LP is a unique solution for a system of linear equations (once you know the basis), and such systems have been solved for many years. However, in practice they are usually solved approximately.

Some iterative methods produce an infinite sequence

$$x^{(t)} = [x_1^{(t)}, \ldots, x_n^{(t)}], \quad t = 0, 1, \ldots$$

in R^n, and others are augmented by various stopping rules. The simplest stopping rules are as follows: Stop after a certain number of iterations or a certain amount of time. A natural termination occurs when $x^{(t)} = x_s$ for all $s > t$. If it is easy to recognize an optimal solution (or a local optimizer), a good stopping rule is to stop when $x^{(t)}$ is optimal (respectively, locally optimal). Other rules involve a small positive number ε, called *tolerance* [e.g., stop when $f(x^{(t)})$ is within ε from the optimal value or stop when $x^{(t)}$ is within ε from an optimal solution (assuming that the closeness can be easily judged)]. Some methods stop when $|x^{(t+1)} - x^{(t)}| \leq \varepsilon$ or when $|f(x^{(t+1)}) - f(x^{(t)})| \leq \varepsilon$.

An iterative method usually works on a class of mathematical programs. For example, methods using derivatives assume their existence. If we know nothing about the function $f(x)$ besides its values $f(x^{(t)})$ and the sequence $x^{(t)}$ does not exhaust S, we do not know how close we come to the optimal value. To guarantee the convergence of a method or to get estimates of how close it comes to the optimum, further restrictions on the class of mathematical programs could be needed. A typical condition on $f(x)$ is the *Lipschitz condition* with a *Lipschitz constant* K:

$$|f(x) - f(y)| \leq K|x - y| \text{ for all } x, y. \tag{A1.5}$$

This condition implies that $f(x)$ is continuous. When the gradient

$$f'(x) = \nabla f(x) = [\partial f(x)/\partial x_1, \ldots, \partial f(x)/\partial x_n] \tag{A1.6}$$

exists and is continuous for all x, (A1.5) is equivalent to the following: $|f'(x)| \leq K$ for all x.

Another useful condition is the convexity of f which guarantees that the function is continuous and that a local optimizer is optimal. Any affine function $f(x) = cx + d$ is convex, smooth, and admits Lipschitz constant $K = |f'(x)| = |c|$.

A desirable property of an iteration method for solving (A1.1) is the *descent property*

$$f(x^{(t+1)}) < f(x^{(t)}) \text{ unless } x^{(t+1)} = x^{(t)}$$

or the *strong descent property* for $f(x)$: $f((1-\alpha)x^{(t)} + \alpha x^{(t+1)})$ is a decreasing function of α when $0 < \alpha \leq 1$ unless $x^{(t+1)} = x^{(t)}$.

The simplex method has the strong descent property.

Now we discuss conditions for a point x to be an optimal solution in the case when the objective function $f(x)$ and the constraint functions are differentiable. First we consider the unconstrained optimization (i.e., $S = R^n$). Then a point $x = x^* = [x_1^*, \ldots, x_n^*]^T$ is locally optimal only if it is *critical* or *stationary* [i.e., the gradient vanishes: $f'(x^*) = 0$].

Consider now the case when S is given by a system of constraints $g_i(x) = 0,\ i = 1, \ldots, k$ with differentiable functions g_i. In this case it is well-known that if x^* is a local optimizer, then the gradient vectors

$$\nabla f(x^*), \nabla g_1(x^*), \ldots, \nabla g_k(x^*) \text{ are linearly dependent.} \quad \text{(A1.7)}$$

In the *regular* case, when $\nabla g_1(x^*), \ldots, \nabla g_k(x^*)$ are linearly independent, the condition (A1.7) can be written as

$$\nabla f(x^*) = \sum_{i=1}^{k} \lambda_i \nabla g_i(x^*) \quad \text{(A1.8)}$$

where the coefficients λ_i are known as *Lagrange multipliers.*

Suppose now that S is given by a system of constraints

$$g_i(x) = 0 \text{ for } i = 1, \ldots, k \text{ and } g_i(x) \leq 0 \text{ for } i = k+1, \ldots, l \quad \text{(A1.9)}$$

with differentiable functions g_i. A feasible solution x is called *regular* if the gradients $\nabla g_i(x)$ with $g_i(x) = 0$ (i.e., *active* gradients) are linearly independent. Here are the Karush-Kuhn-Tucker (KKT) conditions for a regular feasible solution x^* to be locally minimal:

$$-\nabla f(x^*) = \sum_{i=1}^{l} \lambda_i \nabla g_i(x),\ \lambda_i g_i(z^*) = 0\ \forall i,\ \lambda_i \geq 0 \text{ for } i > k.$$

The condition $\lambda_i g_i(z^*) = 0\ \forall i$ means that only the active gradients are involved. The KKT conditions are a system of linear constraints for λ_i, which becomes a system of linear equations (A1.8) in the case $l = k$. The fact that the KKT conditions are necessary follows easily from the duality in linear programming combined with an implicit function theorem. In the the case when the constraints are linear, the condition of regularity can be dropped. (This is also true in the case $l \leq 1$.) Moreover, in this case the KKT conditions are sufficient for x^* to be optimal provided that $f(x)$ is convex, cf. A3 below.

A2. Univariate Programming

This is mathematical programming with a single decision variable ($n = 1$ in notations of A1). The feasible region most often is an interval, a ray, or the whole line R (unconstrained optimization). By various methods, mathematical programs with several variables (multivariate programs) are reduced to or use univariate programs. For instance, we can try to approximate the feasible region S in R^n by a "space-filling" curve. Or we can choose a feasible direction at a point and then do a line search along this direction to find the next point.

Direct Search Methods

These methods are used when derivatives are expensive to calculate or do not exist. However, there is no sharp distinction from methods using derivatives, because derivatives can be approximated using values of $f(x)$.

A simple-minded way to minimize a function $f(x)$ on an interval $a \le x \le b$ is to choose a large natural number N, compute

$$f(a + i(b - a)/N) \text{ for } i = 0, 1, \dots, N,$$

and find the minimal value. When $f(x)$ admits a Lipschitz constant K, this value differs from the optimal value not more than by $K/(2N)$. More refined methods, *grid-based methods*, involve subdivision some of the intervals $a+i(b-a)/N \le x \le a+(i+1)(b-a)/N$ into smaller intervals in search for better solutions.

As an alternative, we can compute $f(x)$ at N points chosen at random in the interval. When $f(x)$ satisfies a Lipschitz condition, the minimal value of $f(x)$ at N points converges to the optimal value with probability 1 as $N \to \infty$.

We call a function $f(x)$ on an interval $a \le x \le b$ *unimodal* if it is continuous and has exactly one locally optimal solution x^*.

Fibonacci search for the minimum of a unimodal function uses the Fibonacci sequence F_t, see Exercise 13 in §22. We fix a number N and we want to restrict the unknown optimal solution x^* to the smallest possible interval by evaluating the objective function at N judicially chosen points. When $N = 0$ or 1, we cannot restrict x^* to a smaller interval. When $N = 2$ we can restrict x^* to the interval $a \le x \le (a + b)/2 + \varepsilon$ or $(a + b)/2 - \varepsilon \le x \le b$ by comparing $f((a+b)/2+\varepsilon)$ and $f((a+b)/2-\varepsilon)$ where $0 < \varepsilon < (a-b)/2$. [When $f'((a + b)/2)$ exists, we can do bisection instead, saving ε.]

For any $N > 2$, we compare $f(a + (1 - F_{N-1}/F_N)(b - a))$ and $f(a + (b - a)F_{N-1}/F_N)$ and reduce the interval $a \le x \le b$ to either the subinterval $a \le x \le a + (b - a)F_{N-1}/F_N$ or the subinterval $a + (1 - F_{N-1}/F_N)(b - a) \le x \le b$. The length of the subinterval is $(b - a)F_{N-1}/F_N$ in both cases. After $N - 2$ steps, we restrict x^* to a subinterval of length $2/F_n$. The last two evaluations reduce the length to $(1 + \varepsilon)/F_n$ with arbitrarily small positive ε.

When $N \to \infty$, the Fibonacci search becomes the *search by golden section*, when we compare $f(a\alpha+(1-\alpha)b)$ and $f((1-\alpha)a+\alpha b)$ and reduce the interval $a \le x \le b$ to either $a \le x \le (1 - \alpha)a + \alpha b)$ or $a\alpha + (1 - \alpha)b \le x \le b$, where $\alpha = 2/(1 + \sqrt{5}) \approx 0.618$. The number $1/\alpha$ is known as *golden section ratio.* After k evaluations, the uncertainty interval for x^* is reduced to a subinterval of length $(b - a)\alpha^{k-1}$.

For a unimodal function, the search by golden section produces a sequence $x_t = x^{(t)}$ convergent to the optimal solution x^*. Applied to an arbitrary continuous function, the method produces a sequence x_t convergent to a point x^* such that

Either $x^* = a$ or there are infinitely many t such that $x_t < x^*$ and $f(x_t) \ge f(x^*)$;

Either $x^* = b$ or there are infinitely many t such that $x_t > x^*$ and $f(x_t) \ge f(x^*)$.

Such a point x^* is critical if $f'(x^*)$ exists and $x^* \ne a, b$.

Splitting the feasible interval into smaller intervals allows us to find more such points.

Better methods can be devised if we use additional information about the objective function. For example, it helps if the function is differentiable, in which case a is locally optimal if $f'(a) > 0$, b is locally optimal if $f'(b) < 0$, and a point c inside $(a < c < b)$ is locally optimal only if it is critical [i.e., $f'(c) = 0$]. A critical point c is locally minimal if $f(x)$ is convex or if $F''(c)$ exists and is positive.

Bisection Method

This is a method for finding a local minimizer using the derivative $f'(x)$. Assume that $f'(x)$ exists in the interval $a \le x \le b$ and that $f'(a) \le 0 \le f'(b)$ (otherwise, we have a local optimizer). We set $a_0 = a$, $b_0 = b$, $x_0 = (a_0 + b_0)/2$.

Given a_t, b_t, and $x_t = (a_t + b_t)/2$, we define a_{t+1}, b_{t+1}, and $x_{t+1} = (a_{t+1} + b_{t+1})/2$ as follows. If $f'(x_t) > 0$, then $a_{t+1} = a_t, b_{t+1} = x_t$. Otherwise, $a_{t+1} = x_t, b_{t+1} = b_t$.

Note that $f'(a_t) \le 0 \le f'(b_t)$, $a_{t+1} \le a_t < b_{t+1} \le b_t$, and $b_t - a_t = (b-a)/2^t$ for $t = 0, 1, \ldots$. It follows that the sequence $x_t = (a_t + b_t)/2$ converges, say $x_t \to x^*$. If $f'(x)$ is continuous at $x = x^*$ then $f'(x^*) = 0$, so x^* is a good candidate for a local optimizer. It is a local optimizer if, for example, $f'(x)(x - x^*) \ge 0$ for x close to x^*.

To look for more than one local optimizer, we can start with splitting the initial interval $a \le x \le b$ into many smaller intervals and apply the bisection method to each small interval.

If the derivatives do not exist or are hard to compute, *dichotomous search* can be used, where we replace computation of $f'(c)$ by evaluating $f(c-\varepsilon)$ and $f(c+\varepsilon)$ for a small positive ε. However, the search by golden section is more efficient because it reduces the interval of uncertainty $(1+\sqrt{5})^2/4 \approx 2.61803$ times using two evaluations of $f(x)$.

In the case of unconstrained program ($S = R$) we can decide that we do not care about solutions x with $|x| > M$ for some large M. Then we have a program where S is a bounded interval, so we can use the bisection method or other methods devised for optimization over intervals. We also can cover R by a sequence of finite intervals. However, there are methods that are simpler in the unconstrained case.

Gradient Methods

Let $S = R$, and let $f'(x)$ exist and be continuous for all x. We start with an initial point x_0. Given any point x_t, the next point in the gradient method is $x_{t+1} = x_t - f'(x_t)$. A modified gradient method is given by

$$x_{t+1} = x_t - \alpha f'(x_t) \tag{A2.1}$$

with some $\alpha > 0$.

The method terminates naturally at a critical point. If $f'(x)$ admits the Lipschitz constant $K < 2/\alpha$, then

$$f(x_t) - f(x_{t+1}) \ge \alpha(1 - K\alpha/2) f'(x_t)^2 > 0$$

so we have the descent property for $f(x)$.

Assume now that $f'(x)$ admits the Lipschitz constant $K \le 1/\alpha$. Then we have the strong descent property for $f(x)$ (see A1). Moreover, we have the strong descent property for $|f'(x)|$.

The last property implies that the sequence x_t is strictly monotone for all t or until we hit s such that $f'(x_s) = 0$. When $f'(x_0) < 0$ [respectively, $f'(x_0) > 0$], then either the sequence converges to the first critical point x^* on the right (respectively, on the left) of x_0 or there are no critical points on the right (respectively, left) and $x_t \to \infty$ (respectively, $x_t \to -\infty$).

Under the additional condition that

$$f'(x) - f'(y) \geq (x - y)K_2 \text{ for all } x, y$$

for some $K_2 > 0$ [e.g., $f''(x) \geq K_2$ for all x] the function $f(x)$ is convex (so every critical point is optimal) and we can estimate the rate of convergence:

$$f'(x_{t+1}) \leq f'(x_t)(K - K_2)/K \leq f'(x_0)(1 - K_2/K)^{t+1}$$

hence

$$|x_{t+1} - x_t| \leq \alpha |f'(x_0)|(1 - K_2/K)^t$$

and

$$|x_t - x^*| \leq \alpha |f'(x_0)|(1 - K_2/K)^t K/K_2.$$

If no Lipschitz constant for $f'(x)$ exists or is available, we can use the *damped gradient method*

$$x_{t+1} = x_t - \alpha_t f'(x_t) \tag{A2.2}$$

with a *damping sequence* $\alpha_1, \alpha_2, \ldots$ such that $\alpha_t > 0$ for all t, $\alpha_t \to 0$ as $t \to \infty$, and $\sum \alpha_t = \infty$. (See [V] for a discussion of damping sequences.) We get convergence of x_t to a critical point, ∞, or $-\infty$, provided that there are only finitely many critical points.

To find more than one critical point, we can try different initial points. For example, if the critical point x^* we found is not a local optimizer, we can start again with a point y_0 close to x^* such that $f(y_0) < f(x^*)$. More sophisticated approaches modify $f(x)$ in a way to exclude the critical points that are already found.

In the case when S is a finite interval $a \leq x \leq b$, any iterative method can be adjusted by replacing infeasible x_{t+1} by a or b whichever is closer.

To avoid calculation of $f'(x_t)$ in the gradient method, we can replace it by, say, $(f(x_t) - f(x_{t-1}))/(x_t - x_{t-1})$ and start with two distinct initial points x_0, x_1.

Newton Methods

More sophisticated methods than the gradient methods are Newton methods. They are designed to work in regions where $f(x)$ is convex. The standard version assumes that

$$f''(x) \text{ exists and is positive for all } x. \tag{A2.3}$$

However, there are versions that do not use the first and/or second derivatives. We start with a version which uses $g(x) = f'(x)$ but not $f''(x)$.

Assume that the derivative $g(x) = f'(x)$ exists and is continuous. We want to find a zero of this function $g(x)$. We start with two initial points $x_0 \neq x_1$. Then we draw the straight line through the points $(x_0, g(x_0)), (x_1, g(x_1))$ and intersect this line, which approximates the function $g(x)$ and whose slope is

$$s = (g(x_1) - g(x_0))/(x_1 - x_0),$$

with the horizontal axis. So we want $s = (0 - g(x_1))/(x_2 - x_1)$, hence $x_2 = x_1 - g(x_1)/s$ for our next point x_2 when $s \neq 0$. When $s = 0$, we take $x_2 = (x_0 + x_1)/2$. Then we repeat the procedure with x_1, x_2 instead of x_0, x_1, and so on. We obtain a sequence x_t that (we hope) converges to a zero of $g(x)$.

For the rest of this section, we assume the condition (A2.3). So the objective function $f(x)$ is convex. In its pure form, the Newton method for finding a minimizer of $f(x)$ [i.e., a zero of $g(x) = f'(x)$] starts with one initial point x_0 and defines the sequence x_t by

$$x_{t+1} = x_t - g(x_t)/g'(x_t). \tag{A2.4}$$

To guarantee the convergence of the Newton method (A2.4) even for a real analytic function, we need additional conditions.

In the case when $g'(x) = f''(x)$ admits a Lipschitz constant K_3 and $|g(x)/g'(x)^2| \leq K_1$ for all x, we have

$$|g(x_{t+1})| \leq (g(x)/g'(x))^2 K_3/2 \leq |g(x)| K_1 K_3/2;$$

hence (A2.4) converges if $K_1 K_3/2 = q < 1$ for all x. Namely, under this condition, $g(x_{t+1}) \leq g(x_0) q^{n+1}$. Also the method converges very fast [with the ascent property for $|g(x)|$] if $g'(x) \geq K_2 > 0$ for all x and we start with x_0 such that $|g(x_0)| K_3/(2K_2^2) = q_0 < 1$.

Namely, we have the descent property for $|g(x)|$ and $q_0|g(x_{t+1})| \leq (q_0|g(x_t)|)^2$; hence

$$q_0|g(x_t)| \leq q_0^{2^t}$$

for all t.

A modified Newton method is

$$x_{t+1} = x_t - \alpha g(x_t)/g'(x_t) \tag{A2.5}$$

with some $\alpha > 0$. It converges and has the descent property for $f(x)$ when $\alpha g'(y)/g'(x) \leq 1$ for all x, y (see the preceding discussion of the gradient method). There is a whole spectrum of quasi-Newton methods which generalize and fill up the gap between the gradient methods and the Newton methods.

For example, here is a damped Newton method:

$$x_{t+1} = x_t - \alpha_t g(x_t)/g'(x_t)$$

where α_t is a sequence such that $\alpha_t \to 0$ (e.g., $\alpha_t = 1/t$).

The Newton method is based on approximation of $f(x)$ by a quadric $a_0 + a_1 x + a_2 x^2$ using information at a single point or at two distinct points. It converges in one step for a strictly convex quadric ($a_2 > 0$). More complicated methods use approximation by a cubic

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

Self-Concordant Functions

An important analysis of the modified Newton method was done in [NN]. Instead of traditional bounds on $f''(x)$ to select α in (A2.5) and analyze convergence, we assume that the function

$$f''(x)^{-1/2} \text{ admits a Lipschitz constant } \kappa_2. \tag{A2.6}$$

In the case when $f'''(x)$ exists, this Lipschitz condition is equivalent to the following *self-concordant* condition which was introduced in [NN] and used in many subsequent publications:

$$|f'''(x)|/f''(x)^{3/2} \leq 2\kappa_2 \text{ for all } x. \tag{A2.6$'$}$$

In the case when $\kappa_2 = 0$, the Newton method terminates in one step with the optimal solution, so we can assume that $\kappa_2 > 0$.

On one hand, we want α in (A2.5) to be small so we have the strong descent property for $f(x)$ that prevents overshooting the local minimum. Since we assume that $f''(x) > 0$ for all x, the strong descent property for $f(x)$ is equivalent to that for $|g(x)|$. On the other hand, taking α smaller than necessary slows down the convergence. So what is the smallest possible α under the condition (A2.6) such that $g(x_{t+1}) = 0$?

The Lipschitz condition (A2.6) means that

$$|g'(x)^{-1/2} - g'(y)^{-1/2}| \le \kappa_2 |x - y| \text{ for all } x, y.$$

Taking here $x = x_t$ and $a = g'(x_t)^{-1/2}$, we obtain that

$$g'(y) \ge 1/(a + \kappa_2 |y - x_t|)^2 \text{ for all } y \tag{A2.7}$$

and

$$g'(y) \le 1/(a - \kappa_2 |y - x_t|)^2 \text{ when } |y - x_t| < a/\kappa_2. \tag{A2.8}$$

We set $b = |g(x_t)|$. Assume now that $g(x_t) < 0$ [the case $g(x_t) > 0$ is similar]. Then we integrate (A2.8) from x_t to $x_t + z$ with $0 \le z < a/\kappa_2$ and obtain

$$g(x_t + z) \le -b + 1/(\kappa_2(a - \kappa_2 z)) - 1/(\kappa_2 a) \text{ for } 0 \le x < a/\kappa_2 \tag{A2.9}$$

On the interval $0 \le z < a/\kappa_2$, the right hand side of (A2.8) increases from $-b$ to ∞ taking the zero value at $z^* = a^2 b/(1 + ab\kappa_2)$. So $g(x_t + z) \le 0$ for $0 \le z \le z^*$. From the equation

$$z^* = -\alpha g(x_t)/g'(x_t) = \alpha a^2 b,$$

we find

$$\alpha = \alpha_t = 1/(1 + ab\kappa_2) = 1/(1 + g'(x_t)^{-1/2} |g(x_t)| \kappa_2).$$

Thus, we have the strong descent property for (A2.5) with this α. [The same α works in the case $g(x_t) > 0$.]

Now we estimate the decrease rate for $|g(x_t)|$. Integrating (A2.7) we obtain

$$g(x_t + x) \ge -b - 1/(\kappa_2(a + \kappa_2 x)) + 1/(\kappa_2 a) \text{ for all } x \ge 0. \tag{A2.10}$$

Taking here

$$x = x^* = a^2 b/(1 + ab\kappa_2),$$

we obtain that

$$|g(x_{t+1})| \le 2ab^2\kappa_2/(1 + 2ab\kappa_2) = |g(x_t)|(1 - \alpha)/(1 - \alpha/2). \quad \text{(A2.11)}$$

[The same is true in the case $g(x_t) \ge 0$.]

Assume now that

$$ab = |g(x_t)|/g'(x_t)^{1/2}$$

is bounded, that is,

$$|f'(x)| \le \kappa_1 f''(x)^{1/2} \text{ for all } x. \quad \text{(A2.12)}$$

We set

$$\kappa = \kappa_1\kappa_2,\ \alpha^* = 1/(1 + \kappa),\ \beta^* = (1 - \alpha^*)(1 - \alpha^*/2) = 2\kappa/(1 + 2\kappa). \quad \text{(A2.13)}$$

Then $\alpha^* \le \alpha_t \le 1$ for all t and

$$|g(x_{t+1})| \le |g(x_t)|\beta \le |g(x_t)|\beta^*$$

for all t hence

$$|g(x_t)| \le |g(x_0)|\beta^{*t} \to 0$$

as $t \to \infty$.

We have the strong descent property for $|g(x)|$, which implies the monotone convergence $x_t \to x^*$ as $t \to \infty$, where x^* is a number or $\pm\infty$. When $|x^*| < \infty$, we have $g'(x) \ge K_2 > 0$ for all x between x_0 and the minimizer x^*. This low bound on $g'(x)$ implies that

$$|x_t - x^*| \le |g(x_t)|/K_2 \le |g(x_0)|\beta^t/K_2.$$

Moreover,

$$|g(x_{t+1})| \le |g(x_t)ab| \le cg(x_t)^2$$

with $c = \kappa_2/K_2^{1/2}$ for all t. Once $|cg(x_s)| = q < 1$ for some s, we have

$$|cg(x_{s+t})| \le q^{2^t}$$

for all t.

Example A2.14. Let $f(x) = -c_1 \log(x) - c_2 \log(1-x)$ with $c_1, c_2 > 0$. On the interval $0 < x < 1$ we have the bound

$$f''(x) \geq K_2 = (c_1^{1/3} + c_2^{1/3})^3$$

as well as (A2.6) with $\kappa_2 = (\min[c_1, c_2])^{-1/2}$ and (A2.12) with $\kappa_1 = (\max[c_1, c_2])^{1/2}$; hence

$$\kappa = \kappa_1 \kappa_2 = (\max[c_1, c_2] / \min[c_1, c_2])^{1/2}.$$

If we start with x_0 in the interval and use the modified Newton method with $\alpha = \alpha^* = 1/(1+\kappa)$, then we stay in the interval by the strong descent property and we have $|f'(x_t)| \leq |f'(x_t)|\beta^{*t}$ and $|x_t - x^*| \leq |f'(x_t)|\beta^t / K_2$ with

$$\beta^* = 2\kappa/(1+2\kappa) = (1-\alpha)/(1-\alpha/2) < 1.$$

Using the variable step size

$$\alpha = \alpha_t = 1/(1 + g'(x_t)^{-1/2} |g(x_t)| \kappa_2) \geq \alpha^* \tag{A2.15}$$

we obtain even faster convergence (of the type $|x_t - x^*| \leq cq^{2^t}$ with $0 \leq q < 1$) preserving the strong descent property for $f(x)$.

Example A2.16. This is a generalization of the previous example. Let

$$f(x) = -\sum_{i=1}^{m} c_i \log(b_i - a_i x)$$

with all $c_i > 0$. This function is defined on the set S given by the constraints $a_i x < b_i$. The set S can be given as follows: $a < x < b$ with a being a number or $-\infty$ and b being a number or ∞. Assume now that $a < b$ (i.e., S is not empty).

We have (A2.12) with $\kappa_1 = (\sum c_i)^{1/2}$ and (A2.6) with $\kappa_2 = \min(c_i)^{-1/2}$ for all $x \in S$.

In the particular case when all $c_i = 1$, we have $\kappa_1 = m^{1/2}$, $\kappa_2 = 1$ and $\kappa = m^{1/2}$. So we can take $\alpha^* = 1/(1+m^{1/2})$.

When S is bounded, we also have a bound $f''(x) \geq K_2 > 0$ for all $x \in S$. If S is unbounded, we do not have such a bound. When $a = -\infty$ and $f'(x_0) > 0$, we have min $= -\infty$ and $x_t \to -\infty$. When $b = \infty$ and $f'(x_0) < 0$, we have min $= -\infty$ and $x_t \to \infty$. In all other cases we have

$$|x_t - x^*| \leq c_1 (2m^{1/2}/(1+2m^{1/2}))^t$$

if we use the modified Newton method with fixed

$$\alpha = \alpha^* = 1/(1+m^{1/2}).$$

We have even faster convergence if we use the variable step size (A2.15).

A3. Convex and Quadratic Programming

This section is about minimizing a convex function $f(x)$ over a convex set S. Convex programming generalizes linear programming.

Theorem A3.1. For any convex program, any locally optimal solution is optimal.

Proof (cf. [B1], [FV]). Let x^* be a locally optimal solution and y is a feasible solution. We have to prove that $f(x^*) \leq f(y)$. Suppose that $f(x^*) > f(y)$.

Since S is convex, $z = (1-a)x^* + ay$ is a feasible solution for $0 \leq a \leq 1$. Since f is convex $f(z) \leq (1-a)f(x^*) + af(y) < f(x^*)$ for $0 < a \leq 1$. For a close to 0, z is close to x^*, which contradicts x^* being locally optimal. ∎

If desirable, by an easy trick (A1.2), $f(x)$ can be arranged to be a linear form while keeping the feasible set convex. The set S is often given by a constraint or constraints of the form $g(x) \leq 0$, with a convex function g. Besides being convex, such a set S is *closed* (i.e. contains its limit points). Conversely, any closed convex set can be given by a constraint $g(x) \leq 0$ with convex $g(x)$. In the case when S is given by a system of convex constraints $g_i(x) \leq 0$ for $i = 1, \ldots, m$, such a convex function $g(x)$ can be written as $g(x) = \max[g_1(x), \ldots, g_m(x)]$.

Some convex sets S are given by constraints involving *quasiconvex functions.*

Definition A3.2. A function $g(x)$ is called *quasiconvex* if the set $g(x) \leq c$ is convex for every number c. ∎

Equivalently, $g(ax+(1-a)y) \leq \max(f(x), f(y))$ for all x, y and all a in the interval $0 \leq a \leq 1$. Every convex function is quasiconvex. A function $g(x)$ of one variable x is quasiconvex if it is unimodal.

Note that several quasiconvex constraints $g_i(x) \leq 0$ for $i = 1, \ldots, m$ can be replaced by a single quasiconvex constraint $g(x) = \max[g_1(x), \ldots, g_m(x)]$.

In the case when $S = R^n$ and the function f is differentiable, x is optimal if and only if $\nabla f(x) = 0$. In general, we have the following result.

Theorem A3.3. Let x^* be a feasible solution for the convex program

$$f(x) \to \min, g_i(x) \leq 0 \text{ for } i = 1, \ldots, m, \tag{A3.4}$$

where $f(x)$ is convex and differentiable at x^* and all $g_i(x)$ are quasiconvex and differentiable at x^*. If x^* satisfies the KKT conditions (see A1 above), then x^* is optimal.

Proof. We write

$$-\nabla f(x^*) = \sum_{i=1}^{m} \lambda_i \nabla g_i(x)$$

with $\lambda_i g_i(z^*) \geq 0$ for all i and $\lambda_i = 0$ when $g_i(x^*) = 0$.

We have to prove that $f(x^*) \leq f(x')$ for any feasible solution x'. Since $g_i(x) \leq 0 = g_i(x^*)$ for any active constraint and the restriction of $g_i(x)$ on the line segment connecting x^* and x' is quasiconvex, we conclude that $\nabla g_i(x^*)(x' - x^*) \leq 0$. So

$$\nabla f(x^*)(x' - x^*) = -\sum_{i=1}^{m} \lambda_i \nabla g_i(x)(x' - x^*) \geq 0.$$

Since the restriction of $f(x)$ on the segment is convex, we conclude that $f(x^*) \leq f(x')$. ∎

Corollary A3.5. Let x^* be a regular feasible solution for the convex program $f(x) \to \min$, $g_i(x) \leq 0$ for $i = 1, \ldots, k$, where $f(x)$ is convex and differentiable at x^* and all $g_i(x)$ are quasiconvex and differentiable at x^*. Then x^* is optimal if and only if it satisfies the KKT conditions. ∎

In the case when a function $f(x)$ has continuous second derivatives, $f(x)$ is convex if and only if its Hessian, that is, the matrix

$$\nabla^2 f(x) = [\nabla^2_{i,j} f(x)] = f''(x) = [\frac{\partial^2 f(x)}{\partial x_i \partial x_j}] \qquad \text{(A3.6)}$$

$= Q$ is *positive semidefinite* (i.e., $y^T Q y \geq 0$ for all $y \in R^n$). Equivalently, the quadratic form $y^T Q y$ is a sum of squares of linear forms.

Traditionally, *quadratic programming* is a part of convex programming. It is about minimizing a function $f(x)$ that is a sum of an affine function and squares of linear forms, subject to a finite set of linear constraints. The objective function can be written in the form $f(x) = d + cx + x^T a x$, where x is a column of n decision variables, d a given number, c a row of n given numbers, and a is an $n \times n$ symmetric positive semidefinite matrix. As a sum of convex functions, our $f(x)$ is convex, so quadratic programming is a particular case of convex programming. When a quadratic program is bounded, it has an optimal solution. This is not true for convex programs, as the univariate unconstrained example $e^x \to \min$ shows. When $a = 0$, our quadratic program becomes a linear program.

When all constraints are equations, an optimal solution can be found by solving two systems of linear equations. Namely, we can solve our system of constraints and excluding variables to reduce our problem to a quadratic program without any constraints. Then we can find all optimal solutions by setting all partial derivatives to be zero. The most often used methods for unconstrained convex programs are Newton and quasi-Newton methods, see the next section. In the quadratic case, the Newton method gives the exact optimal solution in one step.

In the constrained case, the interior methods are most often used, see A5. Many methods of constrained optimization work particularly well in the case of a convex program, see A4. In fact, for nonconvex programs those methods are often *heuristic*, i.e., they seek a solution but do not guarantee they will find one. By contrast, while applied to convex programs some methods have good convergence properties. Moreover, under additional restrictions on the Hessian, we can estimate the convergence rate, cf., A2 and A4.

One approach to solving a convex program is to approximate the convex objective function by a piecewise linear function (i.e., by the maximum of a finite set of affine functions). Also, we approximate the feasible region S by a finite system of linear constraints. When S is given by constraints of the form $g_i(x) \le 0$ with convex $g_i(x)$, this can be achieved by approximating each $g_i(x)$ by a piecewise linear function. The point here is that the convex program (A3.4) with piecewise linear functions $f(x), g_i(x)$ is equivalent to a linear program. There are special modifications of the simplex method to handle linear program arising this way. Some of them work with tableaux of variable size, where columns and rows are added when necessary and redundant rows and columns are dropped.

Duality in linear programming has been extended to convex programming. For example, for a quadratic program

$$x^T Qx + cx \to \min, Ax \le b, x \le 0$$

with $Q = Q^T$, its dual according to [D] is

$$-y^t Qy + bu \to \max,\ u \ge 0,\ A^T u - 2Qy \le c.$$

Finally, there are tricks to reduce nonconvex programs to convex programs. For example, consider the program $f(x) \to \min$ with a twice differentiable $f(x)$. If $K(f'(x)u)^2 \ge -u^T f''(x)u$ (respectively, $K(f'(x)u)^2 \ge -u^T f''(x)uf(x)$) for some $K \ge 0$ and all $x, u \in R^n$ then our program has the same optimal solutions as the convex program $\operatorname{sign}(f(x))|f(x)|^{K+1} \to \min$ (respectively, $e^{Kf(x)} \to \min$).

A4. Multivariate Programming

Methods for univariate optimization (see A2) have been generalized or used in many ways for multivariate optimization. Some methods are simpler when the feasible region is the whole space R^n so we start with unconstrained optimization and discuss the constrained case later.

Coordinate Descent Method

We start with an initial point $x^{(0)} = [x_1^{(0)}, \ldots, x_n^{(0)}]$. To find the next point, we choose a *descent coordinate* and optimize $f(x)$ with respect to this variable keeping other variables fixed. Then we do this with other coordinates to get a sequence $x^{(0)}, x^{(1)}, \ldots$ with $f(x^{(t+1)}) \leq f(x^{(t)})$. A possible choice of descent coordinates is cyclic:

$$x_1, x_2, \ldots, x_n; x_1, x_2, \ldots.$$

If $f(x^{(t+n)}) = f(x^{(t)})$ for some t and $f'(x)$ exists and is continuous at $x = x^{(t)}$, then $f'(x^{(t)}) = 0$.

When the gradient $f'(x)$ of $f(x)$ is available, it can be used for a better choice of descent coordinate. This method is simpler than the gradient method (see below), but the convergence is usually poorer.

Grid-Based Methods

Grid-based methods involve a derivative-free search for a local optimizer over successively refined meshes. See [CP1].

Simplex-Based Methods

Simplex-based methods construct an evolving pattern of $n + 1$ distinct points in R^n that are viewed as the vertices of a simplex (in the case $n = 1$ we have an evolving pair of points and the simplex is an interval; cf. A2). In some methods, the next simplex is obtained by using simple rules such as reflecting away from a vertex with the largest value of $f(x)$ and contracting toward a vertex with the smallest value of $f(x)$.

In other methods, evolution is more complicated, with the main goal being strict improvement of the best objective function value at each iteration (cf. [K2], [SMY]).

Gradient Methods (a.k.a. Methods of Steepest Descent)

We want to find a critical point of a continuously differentiable function $f(x)$ (to be minimized) starting with an initial point $x_{0,}$. The next point $x^{(t+1)} = F(y, t)$ depends on the previous point $x^{(t)} = y$ as follows:

$$F(y, t) = y - \alpha_t f'(y)^T \qquad \text{(A4.1)}.$$

The numbers $\alpha_t > 0$ here are called *step sizes*, and $f'(x) = \nabla f(x)$ is the gradient [see (A1.6)].

The method terminates naturally when it hits a critical point y in which case $F(y,t) = y$. Until this happens, for sufficiently small α_t, we improve our objective function at each step (i.e., we have the strong descent property).

The *step length* is $b_t = |x^{(t+1)} - x^{(t)}| = |\alpha_t f'(y)|$. For a method to converge to a point we want $b_t \to 0$. (This corresponds to cooling in simulated annealing.) To reach an optimal solution that could be far away from the initial point, we would like to either have $\sum 1/b_t = \infty$ or try many different initial points. If $x^{(t)}$ converges to a point z, then $f'(z) = 0$ provided that $f'(x)$ is continuous.

The perfect choice for α_t would be the value α that is optimal for the univariate minimization $f(y - \alpha f'(y)^T) \to \min$, where $y = x^{(t)}$. The perfect line search can be done exactly in some cases [e.g., when $f(x)$ is a polynomial of total degree 2]. In general, line search is done approximately (see A2). For example, we can set $a_t = 1/t$ or $b_t = 1/t$ (cf., Brown's method).

When $f(x)$ is not differentiable or computation of its derivatives is not so easy, we replace the direction $-f'(y)$ in the gradient method by any direction taking $f(x)$ down (method of feasible directions). On the other hand, when the second derivatives of $f(x)$ are available, the Newton method (discussed subsequently) is preferable.

For instance, let $f(x) = x^T Q x + cx$ with a positive definite symmetric matrix Q (i.e., $z^T Q z > 0$ for all $0 \neq z \in R^n$). Then there is a unique optimal solution $x^* = -Q^{-1}c^T/2$ and the gradient method with perfect line search gives the global convergence $|x_{t,} - x^*| \leq \alpha\beta^t$ with some α and $0 < \beta < 1$. (A similar bound holds in the more general case when $f(x)$ is a convex function with continuous second derivatives and $y^T f''(x) y / z^T f''(x) z$ is bounded when $x, y, z \in R^n$ and $|y| = |z| = 1$, see [L].) The Newton method in this case gives the global convergence in one step: $x_{,1} = x^*$.

Trust Region Methods

Given a point w we find the next point $F(w)$ by minimizing a *merit* function $\tilde{f}(x$, which approximates $f(x)$, over a *trust region* S'. We *trust* our approximation in S'. See [CGT].

For example, the *linear approximation* of a differentiable function $f(x)$ at a point w is the affine function

$$f(w) + f'(w)(x - w) \tag{A4.2}$$

of x. Taking S' to be the ball $|x - w| \leq \alpha_t |f'(w)|$, we obtain the gradient method (A4.1) above.

Using the same linear approximation but $S' = S$ given by a finite set of linear constraints, we obtain a method for finding local optimizers in concave problems, see below.

In the unconstrained case, taking the linear approximation as $\tilde{f}(x)$, and S' given by $(x-w)^T Q(x-w) \leq 1$ with symmetric positive definite matrix Q, we obtain the classical *variable metric method.*

Using again the linear approximation but a different trust region, depending on constraints, we obtain the affine scaling method (see A5).

Assume now that the Hessian $f''(x)$ (see (A3.6)) exists. The *quadratic approximation*

$$\tilde{f}(x) = f(w) + f'(w)(x - w) + (x - w)^T f''(w)(x - w)/2 \quad \text{(A4.3)}$$

of $f(x)$ at w is a polynomial in x of total degree ≤ 2. Suppose that the matrix $Q = f''(w)$ is positive definite. Using the quadratic approximation and $S' = R^n$, we obtain the Newton method (A4.4) below.

Newton Methods

We assume that the matrix $f''(x)$ exists and is continuous and invertible for all x. The next point $F(w)$ in the Newton method is the critical point of the quadratic approximation $\tilde{f}(x)$ (see (A4.3)) which can be computed explicitly:

$$F(w) = w - f''(w)^{-1} f'(w)^T. \quad \text{(A4.5)}$$

Note that $F(w)$ is the minimizer for $\tilde{f}(x)$ if and only if the matrix $f''(w)$ is positive definite, i.e., $\tilde{f}(x)$ is convex.

To get convergence we usually imposes Lipschitz conditions on second derivatives or bounds on third derivatives of $f(x)$. For example, suppose that $f''_d(x) \geq K_0 > 0$ for all $x, d \in R^n$ with $|d| = 1$ and $f''_d(x)$ admits a Lipschitz constant K_3 for every $d \in R^n$ with $|d| = 1$. Here f''_d is the directional derivatives of f, i.e., the second derivative d^2/dz^2 of the restriction of f onto a line $x = w + dz$ with $d \in R^n$. Then

$$|f'(F(w))| \leq |f'(w)|^2 K_3/(2K_0^2). \quad \text{(A4.6)}$$

Thus, the Newton method converges well to a local minimizer x^* if $f''(x^*)$ is positive definite and we start sufficiently close to x^*.

We do not need to assume that $f(x)$ is convex everywhere in R^n. The damped version

$$F(w) = w - \alpha f''(w)^{-1} f'(w)^T \tag{A4.7}$$

with $0 < \alpha \leq 1$ can be used to increase the region of convergence.

For large n, the matrix $f''(w)^{-1}$ is usually computed approximately, using its previous value as a starting point. This leads to *the conjugate direction methods* and *quasi-Newton methods*, which fill the gap between the gradient methods and the Newton methods (see [L, Ch.8 and 9]).

Self-Concordant Functions

Assume that $f''(x)$ exists, is continuous, and is positive definite. Assume also that the restriction of $f(x)$ onto every line satisfies the condition (A2.6), that is,

$$(e^T f''(x)e)^{-1/2} u - (e^T f''(x+e)e)^{-1/2} \leq \kappa_2 \text{ for all } x, e \in R^n, e \neq 0. \tag{A4.8}$$

As we saw in A2, the modified Newton method (A4.7) has the strong descent property for $f(x)$ if

$$\alpha \leq \frac{1}{1 + \kappa_2 |f'(w)u||u|/(u^T f''(w)u)^{1/2}}$$

where $u = -f''(x)^{-1} f'(w)^T$ is the Newton direction at w hence $f'(w)u = -f'(w)f''(x)^{-1}f'(w)^T < 0$.

But to get a better decrease in $|f'(x)|$ [rather than in $|f'(x)u/|u|$] we want a smaller

$$\alpha \leq \alpha_t = \frac{1}{1 + \kappa_2 |f'(w)|^{1/2} |u|^{-1/2} (-f'(w)^T f''(w)u)^{-1/2}}, \tag{A4.9}$$

where

$$0 < -f'(w)^T f''(w)u = f'(w)^T f''(w) f'(w)$$
$$\leq (u^T f''(w)u)^{1/2} (f'(w)^T f''(w) f'(w))^{1/2}.$$

With such α we have (assuming that the third derivatives exist and continuous as in [NN])

$$|f'(F(w))| \leq |f'(w)|(1-\alpha)/(1-\alpha/2).$$

Now we introduce an n-dimensional version of (A2.12):

$$f'(x)f''(x)^{-1}f'(x)^T \le \kappa_1 |f'(x)|^{1/2} |f''(x)^{-1}f'(x)^T|^{1/2}| \ \forall\ x \in R^n. \tag{A4.10}$$

Under this condition, we set

$$\kappa = \kappa_1\kappa_2, \alpha = 1/(1+\kappa), \beta = 2\kappa/(1+2\kappa) = (1-\alpha)/(1-\alpha/2).$$

Then $\alpha^* \le \alpha_t$ for all t and the method (A4.7) has the strong descent property and $f'(x^{(t)}) \le f'(x^{(0)})\beta^t$ for all t where $x^{(t)} = F^t x^{(0)}$.

It follows that either $|x^{(t)}| \to \infty$ or $x^{(t)} \to x^* \in R^n$. Under the additional condition

$$u^T f''(x)u \ge K_2|u|^2 \ \forall\ u \in R^n \tag{A4.11}$$

for some $K_2 > 0$, we obtain that $|x^{(t)} - x^*| \le \beta^t/K_2$. Taking $\alpha = \alpha_t \ge \alpha^*$ as in (A4.9), we obtain faster convergence near x^*. ∎

Now we discuss the constrained optimization. Even finding a feasible solution in this case could be difficult.

In some cases, constraints defining S can be explicitly solved for some variables which allows us to eliminate those constraints and variables. For example, if all our constraints are linear equations, we can eliminate all constraints and obtain an unconstrained program.

Some iterative methods can be adapted to the constrained case by restricting, if necessary, the step size to stay in F. If we hit the boundary, more sophisticated methods like the gradient projection [finding a feasible direction closest to $-f'(x)$] are used to stay in S [see subsequent discussion and [L]]. So it could be a good idea to avoid the boundary using barrier methods (see the next section).

Penalty Methods

A widely used way to remove constraints in (A1.1) involves a *penalty function.* This function $p(x)$ should be zero on S and positive elsewhere. So the feasible solutions for (A1.1) are the optimal solutions for the unconstrained program $p(x) \to \min$.

The program (A1.1) is replaced by a sequence $f(x) + c_k p(x) \to \min$ of unconstrained programs P_k with a sequence $0 < c_k \to \infty$ of large penalties for violating the constraints.

Assuming that both $f(x)$ and $p(x)$ are continuous and that the program P_k has an optimal solution $x^{(k)}$ [which is automatic when $f(x)/p(x) \to 0$ as $|x| \to \infty$], every limit point of the sequence $x^{(k)}$ is an optimal solution for the program $f(x) \to \min,\ x \in S$.

When S is given by constraints (A1.8) a penalty function can be given as

$$p(x) = \sum_{i=1}^{k} g_i(x)^2 + \sum_{i=k+1}^{l} (\max[0, g_i(x)])^2. \qquad \text{(A4.12)}$$

When all $g_i(x)$ are differentiable, so is $p(x)$.

Note that the same S can be given by different systems of constraints, which results in different penalty functions. A good judgement should be exercised to get $p(x)$ such that minimization of $f(x) + c_k p(x)$ can be done efficiently by a chosen method (e.g., a Newton method). Another point to be decided on is how to coordinate the degree of precision for solving P_k with a choice of the sequence c_k.

When we have a program (A3.4) with convex $g_i(x)$ the function $p(x)$ becomes $p(x) = \sum_{i=1}^{m} (\max[0, g_i(x)])^2$ which is a convex function, so each P_k is a convex program. Another convex penalty function in this case is $p(x) = \max[0, g_1(x), \ldots, g_m(x)]$.

A penalty function $p(x)$ is called *exact* if for some number c a local minimizer of the unconstrained problem $f(x) + cp(x) \to \min$ is also a local minimizer for the original program (A1.1). Search for such functions leads to duality and the KKT conditions. See [L]. Some iterative methods bridge or combine the ideas of duality and penalty.

Linear Constraints

Suppose that S is given a finite system of linear constraints, like in linear programming. If all constraints are linear equations then we can solve this linear system. If there are no feasible solutions or there is only one feasible solution, we are done. Otherwise, we can exclude some variables and get an equivalent unconstrained optimization problem (with a smaller number of variables).

In the case of linear program, finding a feasible solution in general is as difficult as finding an optimal solution. One method is the simplex method. The ellipsoid method can be used for a linear or a convex program; see [H1]. Also, there are *infeasible interior methods*, which are similar to interior methods but are exterior methods working with infeasible solutions and intended to produce a feasible solution.

When S is given by linear constraints and $f'(x) = \nabla f(x)$ exists, the following method using linear programming has been suggested.

Given a feasible solution w, we minimize the linear approximation (A4.1) of $f(x)$ at a point w, which is an affine function of x, to find the next point $F(w)$. Note that we have the strong descent property in the case when $f(x)$ is convex. However, when no vertex is optimal, line search on the segment connecting w and $F(w)$ is needed for convergence. Here is an example of damping:

$$x^{(t+1)} = x^{(t)} + (F(x^{(t)}) - x^{(t)})/t.$$

To find an initial point $x^{(0)}$ (or to find that the program is infeasible) we can minimize an arbitrary linear form over S. We terminate the procedure when $f'(x^{(t)}) = 0$. In the convex case, it is clear that either $x^{(t)}$ converges to an optimal solution or $|x^{(t)}| \to \infty$. In general, every limit point of the sequence is critical.

The method with perfect line search was suggested in [FW] for quadratic programming. In this case the line search is accomplished by the Newton method in one iteration. Also in this case an optimal solution exists if S is not empty and $f(x)$ is bounded from below.

Here is what can be done if the direction d at a feasible solution y improves $f(x)$ but is not feasible. We consider the linear system $Ax = b$ corresponding to the active constraints. Then we project d onto the subspace $Ax = 0$, i.e., replace d by the closest vector pd in the subspace, see §23. The explicit formula for the matrix p in the case when y is regular (i.e., AA^T is invertible) is

$$p = 1_n - A^T(AA^T)^{-1}A.$$

The direction pd is feasible and improves $f(x)$ unless $pd = 0$. In the case when $d = -f'(x)$ we obtain a *gradient projection* method.

In the case when the objective function is *concave* [i.e., $-f(x)$ is convex], we have the corner principle, so no damping is necessary. Concave programs appear in some applications [VCS]. Once a local optimizer is found, an additional linear constraint can be used to exclude it (*cutting plane* methods, see [L, Chapter 13]). After this the procedure is repeated to get a new local minimizer. It is possible to obtain an optimal solution (if it exists) in finitely many steps. See [HT], [P].

Note that more general feasible regions, especially convex ones, can be approximated by regions given by linear constraints. This extends possible applications of methods mentioned in this subsection.

A5. Interior Methods

Given a set S in R^n, a point a in S is called *interior* if there is an $\varepsilon > 0$ such that the ball $|x - a| \leq \varepsilon$ is contained in S.

An interior (point) method for solving a mathematical program $f(x) \to \min, x \in S$ starts with an interior point $x^{(0)}$ and generates a sequence of interior points $x^{(0)}, x^{(1)}, \ldots$ convergent to an optimal solution (or finds that there are no optimal solutions). In some cases such global convergences is too much to hope, and we are satisfied with one of the following: $f(x^{(t)}) \to \min$; a subsequence of $x^{(t)}$ converges to a local optimizer.

Brown's fictitious play method is an interior point method when it starts with a mixed strategy $x^{(0)}$ with nonzero entries. Karmarkar [K1] made a breakthrough in mathematical programming when he suggested an interior point algorithm with good convergence for linear programming. Since that time many improvements and generalizations were suggested in thousands of publications. Most of these publications deal with convex programming.

The set $\mathrm{Int}(S)$ of interior points in S can be empty even for a nonempty S. The *boundary* of S is defined to be the set $a \in R^n$ such that for every $\varepsilon > 0$, the ball $|x - a| \leq \varepsilon$ contains a point in S and a point outside S. Note that when the objective function is affine but not constant, every optimal solution (if any exists) belongs to the boundary of the feasible region.

Some mathematical programs with empty $\mathrm{Int}(S)$ can be transformed into those with nonempty interiors and then solved by interior methods. For example, when S is given by constraints $g_i(x) \leq 0$ for $i = 1, \ldots, m$ with continuous functions, we can use an exterior method and obtain a point y such that $g_i(x) < \varepsilon$ for all i with a small $\varepsilon > 0$. Then we relax the constraints $g_i(x) \leq 0$ to $g_i(x) \leq \varepsilon$ to enlarge S to the set S_ε including this point into the $\mathrm{int}(S_\varepsilon)$.

Another way works for linear programs. Solving some linear equations and excluding some variables in a LP with the feasible region S, we can obtain an equivalent (by affine transformations) linear program with the feasible set S' such that either both programs are infeasible, or both problems have exactly one feasible solution, or $\mathrm{int}(S')$ is nonempty. A more sophisticated way, which does not require solving any equations, involves connection with matrix games. Solving any linear program can be reduced to solving a symmetric matrix game, which in its turn can be reduced to solving a linear program with a known optimal value (namely, 0) and a known feasible solution in the interior.

Many methods for unconstrained optimization or univariate optimization can be adjusted to become interior methods. For example, this is clear for the coordinate descent method. In the gradient method or the Newton method we decrease the stepsize, if necessary, to stay in int(S).

Even when int(S) is not empty it could be a difficult problem, the *feasibility problem* or Phase 1, to find an interior point. Different modifications of the interior method, called infeasible interior methods, are suggested to handle this problem.

Now we consider some methods that became particularly important after [K1].

Affine Scaling Methods

Let the feasible region S be convex and given as in (A3.4). Let int(S) be given by $g_i(x) < 0$ for $i = 1, \ldots, m$. We start with a point $x^{(0)} \in$ int(S). Given any point $w = x^{(t)} \in$ int(S), the constraints giving S can be rewritten as follows:

$$(g_i(w) - g_i(x))/g_i(w) \leq 1 \text{ for } i = 1, \ldots, m.$$

We define the region $S_t \subset S$ containing x_t by

$$\sum_{i=1}^{m} ((g_i(w) - g_i(x))/g_i(w))^2 \leq 1. \tag{A5.1}$$

Next we define $F(w) = x^{(t+1)}$ as a minimizer of $f(x)$ over S_t. In the unlikely case when $F(w)$ hits the boundary, we can do some damping [i.e., replace $F(w)$ by $w+\alpha(F(w)-w)$ with positive $\alpha < 1$.]

Actually the method is useless unless minimization of $f(x)$ over S_t is easier than that over S. In general, it is not easier. For example, when $S_t = S$ for all t when $m = 1$. When $m \geq 2$ the set S_t need not be convex. However, there is an important case when the minimization over S_t is easier. Namely, assume that the functions $f(x)$ and $g_i(x)$ are affine and that $f(x) = cx + d$ is not constant (i.e., $c \neq 0$). Then the constraint (A5.1) defining S_t has a polynomial $g(x)$ of total degree ≤ 2 on the left-hand side. Therefore, we can find $x^{(t+1)}$ easily. One way to do this is to make an affine change of variables and bring $g(x)$ to one of the following two standard forms: $g(x) = z_1^2 + \cdots + z_m^2 + d_0$ or $g(x) = z_1^2 + \cdots + z_k^2 + d_1 z_{k+1} + d_0$ with $k \leq m - 1$.

In the first case, S_t is the ball $|z|^2 \leq 1 - d_0$ in the new coordinate (an ellipsoid in the original coordinates), and it is easy to minimize $f(x) = cx + d = \tilde{c}z + \tilde{d}$ over S_t: The unique optimal solution is $z^{(t+1)} = -(1 - d_0 1)^{1/2}\tilde{c}/|c|$.

In the second case, since the program is bounded, $\tilde{c}_i = 0$ for $i > k$, so the first k components of an optimal solution $z^{(t+1)}$ are unique and given a similar formula, the k^{th} component is arbitrary when $d_1 = 0$ or is subject to a linear constraint, while the other components (if they exist) are arbitrary.

Instead of changing variables, we can just solve a system of linear equations (the KKT conditions; see A1).

Note that the condition that $f(x)$ is affine can be satisfied easily (see A1), and that any convex set S can be approximated by a system of linear constraints. So the method of affine scaling can be used for more general convex programs, at least in principle. For example, we can replace the constraints $g_i(x) \leq 0$ by linear constraints $g_i(x^{(t)}) + g_i'(x^{(t)})(x - x^{(t)}) \leq 0$. Damping could be used to stay in $\text{int}(S)$.

Practical computations showed that the method is sensitive to the choice of an initial interior point x_0. A good tip is to stay away from the boundary. There is some evidence [H2] to indicate good results for convex programs provided that we start close to an optimal solution or, more generally, to the *central path* (see subsequent discussion).

Barrier Methods

Let S be a subset of R^n with nonempty interior $\text{int}(S)$. A *barrier* function $B(x)$ for S is a continuous function on $\text{int}(S)$ such that $B(x) \geq 0$ for all $x \in \text{int}(S)$ and $B(z^{(k)}) \to \infty$ for every sequence $z^{(k)} \in \text{int}(S)$ that converges to a point outside $\text{int}(S)$. Note that unless $S = R^n$ a barrier function cannot be extended to a continuous function on R^n.

If S is given as in (A3.4), here are some barrier functions $B(x)$:

$$-\sum_i \log(-g_i(x)); -\sum_i -1/g_i(x); -1/\max_i[g_i(x)].$$

Keeping S intact, changes in $g_i(x)$ generate more examples and erase the difference between these three examples.

Given a bounded mathematical program (A1.1) with a continuous $f(x)$, a barrier function $B(x) \geq 0$ on nonempty $\text{Int}(S)$, and a sequence $\delta_t > 0$ such that $\delta_t \to 0$, we approximate the program by a sequence of programs

$$f(x) + \delta_t B(x) \to \min, x \in \text{ int}(S), \qquad \text{(A5.2)}.$$

If we start in $\text{int}(S)$ and use a method with the strong descent property to solve (A5.2) ignoring the constraint $x \in \text{int}(S)$, then we stay in $\text{int}(S)$.

Theorem A5.3. Suppose that $v = \inf_{x\in \mathrm{int}(S)}(f(x)) > -\infty$. Set $v_t = \inf_{x\in \mathrm{int}(S)}(f(x) + \delta_t B(x))$. Then $v_t \to v$ as $t \to \infty$.

Proof. Clearly $v_k \geq v_{k+1} \geq v$. For any $\varepsilon > 0$, we find $y \in \mathrm{int}(S)$ such that $f(y) - v \leq \varepsilon/2$. Next we find k such that $\delta_t B(y) \leq \varepsilon/2$ for $t \geq k$. Then

$$|v_k - v| = v_k - v \leq (f(y) + \delta_t B(y)) - v \leq \varepsilon/2 + \varepsilon/2 = \varepsilon$$

for $t \geq k$. ∎

Assume now that $f(x) = cx$ is linear, $B(x)$ is convex and that the program

$$f(x) + \delta B(x) \to \min, x \in \mathrm{int}(S)$$

has a unique optimal solution $x^*(\delta)$. The points $x^*(\delta)$ form the so called *central path,* so barrier methods are also known as central path methods. The sequence $x^{(t)} = x^*(\delta_t)$ in Theorem A5.3 follows this path, hence the term a *path-following method.*

The point $x^{(t+1)}$ is usually found by one or more steps of the modified Newton method starting from $w = x^{(t)}$. So first we find the Newton direction $u = -B''(w)^{-1}(c/\delta_t + B'(w))^T$ for $f(x) + \delta_t B(x)$. Then we set $x^{(t+1)} = w + \alpha u$ where the numbers α are chosen to improve $f_k(x)$ The strong descent property would keep $x^{(t+1)}$ interior automatically. According to A2, $\alpha = 1/(1 + \kappa)$ is a good choice where κ is an upper bound for $-(cu/\delta_t + b'(0)|b'''(0)|)/b''(0)^2$ and $b(s) = B(w + us)$.

Larger step size α may take us outside S. An alternative choice for finding α that does not require $f'''(x)$ is $\alpha = \beta_t\alpha_t$, where α_t is the maximal value for the univariate program $\alpha \to \max, x^{(t)} + \alpha z \in S$ and β_t is a damping sequence (say, $\beta_t = 1/t$).

For good convergence we want $B(x)$ to be self-concordant with $f''_d(x) \geq K_0 > 0$. Then $f_k(x)$ is also self-concordant. The existence of self-concordant barriers in convex programming was proved in [NN]. Moreover, for several classes of programs such barriers were constructed explicitly [after making the objective function linear as in (A1.2)].

For S given by a finite system of linear constraints $g_i(x) \leq 0$, a good self-concordant barrier is the logarithmic barrier $B(x) = -\sum_i \log(-g_i(x))$.

One purpose of the barrier method is to introduce a penalty for approaching the boundary and hence stay in the feasible region while using methods for unconstrained optimization.

For this, we usually require that $B(z^{(k)}) \to \infty$ for any sequence $z^{(k)}$ in int(B) which converges to a point at the boundary. However, this may inhibit our approach to optimal solutions at the boundary, so the parameter $\delta_t \to 0$ is used. In some cases we can find and use a barrier function such that $B(z^{(k)}) \to \infty$ for any sequence $z^{(k)}$ in int(S) that converges to any nonoptimal point at the boundary, with a fixed value δ_t.

Here is a version of the path following method that can be called the sliding objective method or the cutting plane method. Given a convex program (A1.1) with linear $f(x) = cx$, we use a convex barrier function $B(x)$ for S and a positive sequence $\alpha_t \to 0$.

As initial point, we take $x^{(0)}$ to be the minimizer of $B(x)$ over S. Given $x^{(t)}$, we find the next point $x^{(t+1)}$ by applying a step (or several steps) of the modified Newton method to the objective function $B(x) - \log(f(x^{(t)}) - f(x) + \alpha_t)$ which is a convex barrier function for $\{x \in S, f(x) \leq f(x^{(t)}) + \alpha_t\}$.

It was shown in [NN] that for any convex S with nonempty bounded int(S) there is a convex barrier function $B(x)$ which is self-concordant in the sense of (A4.8) with κ_2 depending on n and which also admits k_1 in the sense of (A4.10). When $f(x)$ is affine, $f(x) + cp(x)$ is self-concordant with the same κ_2 and admits a different κ_1.

For some S, self-concordant barriers are constructed explicitly in [NN], [H2]. In [NN] self-concordant functions are allowed to have noninvertible Hessian, which requires some changes in definitions and in the Newton method. However, after excluding the variables which do not effect $f(x)$, we are back in the case of an invertible Hessian.

Example A5.4. Let S be given by linear constraints

$$g_i(x) \leq 0, \ i = 1, \ldots, m.$$

We assume that int(S) is not empty. Then

$$B(x) = \sum_{i=1}^{m} \log(-g_i(x))$$

is a convex barrier function.

When int(S) contains a whole straight line, a variable can be excluded from our program. Otherwise, $f''(x)$ is invertible for all x. When S is bounded, the condition (A4.11) holds for some $K_2 > 0$.

By our computation in Example A2.16, $p(x)$ is self-concordant with $\kappa_2 = 1$ and the condition (A4.10) holds with $\kappa_1 = m^{1/2}$.

Potential Reduction Methods

Given a mathematical program (A1.1), a *potential function* $h(x)$ is a function on $\text{int}(S)$ satisfying the following condition: for any sequence $x^{(k)}$ in $\text{int}(S)$, $h(x^{(k)}) \to -\infty$ if and only if $f(x^{(k)}) \to f_0$, where f_0 is the optimal value. So minimization of $h(x)$ or $e^{h(x)}$ is equivalent to minimization of $f(x)$.

For example, let $B(x)$ be a barrier function. Assume that $B(x)+C_0 \log(f(x) - f_0) \to -\infty$ when x converges to an optimal solution. Then $C_0 \log(f(x) - f_0) + B(x)$ is a potential function.

Now we use the setting in (A3.4) with affine functions $f(x), g_i(x)$. In [K1], $e^{h(x)} = (f(x) - f_0)^m / \prod_{i=1}^m g_i(x)$ with such a function $h(x)$ was used to measure the progress of descent but it did not enter explicitly in the determination of the direction of movement. This is the traditional concept of *merit function*. However nothing is wrong with minimizing the potential or merit function instead of the objective function (except possible confusion with terminology).

However, replacement of $f(x)$ by $h(x)$ or $e^{h(x)}$ makes sense only if some optimization method works better for $h(x)$ or $e^{h(x)}$ than for $f(x)$. In [K1], $h(x)$ is not convex, but $e^{h(x)}$ is convex and self-concordant.

Linear Complementary Problem (LCP)

The problem is to find two vectors $x, y \in R^n$ such that

$$y = Ax + b,\ x \geq 0,\ y \geq 0,\ x_i y_i = 0 \text{ for all } i$$

with given $n \times n$ matrix A and a vector $b \in R^n$.

An LCP is *monotone* if the matrix A is positive semidefinite. By the KKT conditions, any quadratic program (in particular, any linear program) can be cast as a monotone LCP.

Interior methods were developed for solving LCP (cf. [FMP]).

Semidefinite Programming (SDP)

We consider the set $M_n(R)$ of all $n \times n$ real matrices. Given two matrices $A = [A_{i,j}], B = [B_{i,j}] \in M_n(R)$ we denote by $A \bullet B$ the number $\sum_{i=1}^n \sum_{j=1}^n A_{i,j} B_{i,j}$. A semidefinite program is a mathematical program of the form $C \bullet X \to \min$, subject to

$$A_{(i)} \bullet X = b_i \text{ for } i = 1, \ldots, m, X \text{ is positive semidefinite,}$$

where X is a symmetric $n \times n$ matrix of variables and the data consist of symmetric matrices $C, A_{(i)} \in M_n(R)$ and numbers b_i.

SDP has wide applications in convex programming, combinatorial optimization, and control theory. Interior methods are used in SDP, [W1], [K4], [R].

A6. Perturbation

Perturbation was the first and still is the easiest way to show that cycling can be avoided; that is, there is a simplex method that always works. It is not practical because it requires additional computations. However, the concept of perturbation is useful in other areas of mathematical programming (see A7). In Phase 2 of the simplex method (see §10), we replace the column $b = [b_1, \ldots, b_n]^T$ in the tableau

$$\begin{array}{cc} x & 1 \\ \left[\begin{array}{cc} A & b \\ c & d \end{array}\right] & \begin{array}{l} = u \\ = z \to \min, \quad x \geq 0, \quad u \geq 0 \end{array} \end{array}$$

by the column $b(\epsilon) = [b_1 + \epsilon, \ldots, b_n + \varepsilon^n]^T$. Formally, we work with *polynomials* in ε. Informally, $b(\varepsilon)$ is a small perturbation of b; ε is considered a small positive number. We compare polynomials in ε as follows:

$a_0 + a_1\,\varepsilon + a_2\,\varepsilon^2 + \cdots \geq f_0 + f_1\,\varepsilon + f_2\,\varepsilon^2 + \cdots$ if either
$a_0 > f_0$, or
$a_0 = f_0$, and $a_1 > f_1$, or
$a_0 = f_0$, $a_1 = f_1$, and , $a_2 > f_2$, or
. . .
. . .
. . . .

Now, when we apply the simplex method, stage 2, to the tableau

$$\begin{array}{cc} x & 1 \\ \left[\begin{array}{cc} A & b(\varepsilon) \\ c & d(\varepsilon) \end{array}\right] & \begin{array}{l} = u \\ = z \to \min, \quad x \geq 0, \quad u \geq 0 \end{array} \end{array}$$

where $d(\varepsilon) = d$ in the initial tableau, we obtain tableaux of the same form, and no entry of $b(\varepsilon)$ is ever 0. This is because the entries of $b(\varepsilon)$ are always linearly independent over the real numbers (they were linearly independent in the initial tableau, and then pivot steps result in addition and multiplication operations on these entries with real coefficients, so they remain linearly independent). So the entry $d(\varepsilon)$ decreases its value after each pivoting step; that is, $d(\varepsilon)$ cannot stay the same. This makes cycling impossible. If we obtain an optimal ε-tableau, then, by setting $\varepsilon = 0$, we obtain an optimal tableau for the original problem. If we obtain an ε-tableau with a bad column, then, by setting $\varepsilon = 0$, we obtain a tableau for the original problem with a bad column.

A7. Goal Programming

In real life, we can have several goals or objectives to optimize. It does not happen often that an optimal solution for one objective function is also optimal for another objective function. To apply mathematical programming, we need to combine those goals into one objective function and possibly take into account some goals in additional constraints. There are many ways to do this, so there are several books on goal, vector, or multiobjective programming (cf. [CVL], [K5], [S]). A similar problem occurs in game theory, where goals of different players could be different.

Suppose that we have several functions $f_1(x), \ldots, f_m(x)$ to minimize. We can set a numerical goal b_i for each objective and then minimize a convex combination of our functions:

$$f(x) = \sum_{i=1}^{m} c_i f_i(x) \to \min, f_i(x) \le b_i \text{ for all } i,\ x \in S$$

where $c_i > 0$ for all i.

Every optimal solution x^* of this program is *Pareto optimal* or *efficient* [i.e., x^* is feasible, and there is no other feasible point, y, such that $f_i(y) \le f_i(x^*)$ for all i and $f_i(y) < f_i(x*)$ for some i]. The set of such points is called the efficient frontier or the Pareto boundary. In some situations, especially when S is convex, every Pareto optimal solution can be obtained as an optimal solution for a convex linear combination of $f_i(x)$.

In the case when $k = n = 2$, S is convex, $f_1(x) = x_1, f_2(x) = x_2$, Nash suggested using

$$f(x) = -(b_1 - f_1(x))(b_2 - f_2(x))$$

instead of

$$f(x) = c_1 f_1(x) + c_2 f_2(x).$$

He did this in the contest of two-person cooperative games, when x_i was the payoff for player i.

The convexity of S gives uniqueness for optimal solution. The optimal solution x^* exists if there is a feasible solution and S is bounded and closed, which is automatic in the game theory context. Moreover, x^* is Pareto optimal if $f(x^*) \ne 0$.

A natural generalization of Nash's approach is the following program:

$$f(x) = -\sum_{i=1}^{m} \log(b_i - f_i(x)) \to \min,\ f_i(x) \le b_i \text{ for all } i,\ x \in S.$$

To get a convex program in the terms of new variables $y_i = f_i(x)$, we would like the image S' of S under the mapping

$$x \mapsto [f_1(x), \ldots, f_m(x)] \in R^m$$

to be convex. In the game theory context this is achieved by allowing the joint mixed strategies. This amounts to replacing S' by its convex hull. So our problem takes the form

$$g(y) = -\sum_{i=1}^{m} \log(b_i - y_i) \to \min,\ y_i \le b_i \ \text{ for all } i,\ y \in S'$$

with a convex self-concordant objective function, which makes the modified Newton method, with an appropriate barrier function, work well.

These approaches are used when the goals are of roughly comparable importance. In the case of *preemptive goal programming* the functions are partially ordered by priority. We can combine goals at the same priority level as before. However, goals at different levels are treated differently.

For example, let $f_i(x)$ be sorted by priority with $f_1(x)$ being most important. We can use the following sequential procedure to solve the overall problem by a sequence of k mathematical programs. First we minimize $f_1(x)$ and obtain the optimal value c_1^*. Then we minimize $f_2(x)$ with the additional constraint $f_1(x) = c_1^*$ (this makes sense only if the first program has more than one optimal solution). And so on. A nice way to state the overall problem is

$$f(x) = \sum_{i=1}^{k} \varepsilon^{i-1} f_i(x) \to \min, f_i(x) \le b_i \text{ for all } i,\ x \in S,$$

where the objective function is a polynomial in ε and the polynomials are compared as in A6. Note that the objective function now is not real-valued. Instead of polynomials, we can consider the rows with k entries that are ordered lexicographically.

Note also that any optimal solution in preemptive goal programming is also Pareto optimal.

A very general way to combine our m objectives $f_i(x)$ is by a function $h(y)$ of m variables that is nondecreasing with respect to every variable in the region $y \ge 0$ [in the differentiable case, this means that $g'(y) \ge 0$) in the region]. Then our program is

$$-h(b_1 - f_1(x), \ldots, b_m - f_m(x)) \to \min, f_i(x) \le b_i \ \text{ for all } i,\ x \in S.$$

Every optimal solution for this program is also Pareto optimal. In the above examples, $h(y) = cy$ with $c \ge 0$ and $g(y_1, y_2) = y_1 y_2$.

A8. Linear Programming in Small Dimension

Recent interior methods beat simplex methods for very large problems. On the other hand, it is known that for LPs with small number of variables (or, by duality, with a small number of constraints), there are faster methods than the simplex method, cf. [C2]. We will demonstrate this for linear programs with two variables, say x and y.

We write the program in canonical form

$$cx + c'y \to \min, ax + a'y \le b, x \ge 0, y \ge 0$$

where c, c' are given numbers and a, a', b are given columns of m entries each. The program can be written in the standard row tableau

$$\begin{array}{c} \begin{array}{ccc} x & y & 1 \end{array} \\ \left[\begin{array}{ccc} -a & -a' & b \\ c & c' & 0 \end{array}\right] \begin{array}{l} = * \ge 0 \\ \to \min \end{array} \end{array} \tag{A8.1}$$

The feasible region S has at most $m + 2$ vertices. Phase 2, starting with any vertex, terminates in at most $m + 1$ pivot steps.

At each pivot step, it takes at most two comparisons to check whether the tableau is optimal or to find a pivot column. Then in m sign checking, at most m divisions, and at most $m - 1$ comparisons we find a pivot entry or a bad column.

Next we pivot to compute the new $3m+3$ entries of the tableau. In one division we find the new entry in the pivot row that is not the last entry (the last entry was computed before) and in $2m$ multiplications and $2m$ additions we find the new entries outside the pivot row and column. Finally, we find the new entries in the pivot column in $m+1$ divisions. So a pivot step, including finding a pivot entry and pivoting, can be done in $8m + 3$ *operations*—arithmetic operations and comparisons.

Thus, Phase 2 can be done in

$$(m + 1)(8m + 3)$$

operations. While small savings in this number are possible [e.g., at the $(m + 1)$-th pivot step we need to compute only the last column of the tableau] it is unlikely that any substantial reduction of this number for any modification of the simplex method can be achieved (in the worst case).

Concerning the number of pivot steps, for any $m \geq 1$ it is clearly possible for S to be a bounded convex $(m+2)$-gon, in which case for any vertex there is a linear objective function such that the simplex method requires exactly $\lfloor 1+n/2 \rfloor$ pivot steps with only one choice of pivot entry at each step. It is also possible to construct an $(m+2)$-gon, an objective point, and an initial vertex in a way that m pivot steps with unique choice are required (or with two choices at the first step such that the first choice leads to the optimal solution while the second choice leads to m additional pivot steps with unique choice).

Now we outline a method to find an optimal solution, assuming that (A8.1) is feasible, into $\leq 100m + 100$ operations. We proceed by induction on m. For $m \leq 12$ we can use the simplex method and finish in

$$(m+1)(8m+3) < 100m + 100$$

operations. So we assume that $m \geq 13$.

Assume that the objective function is nonconstant (otherwise, we are done in two comparisons). We set $u = -cx - c'y$. Set $v = x$ when $c' \neq 0$ and $v = y$ otherwise.

We rewrite all $m+1$ conditions (not including $v \geq 0$) in the form

$$u \leq a_i v + b_i,\ i = 1, \ldots, l,$$

$$u \geq a_{-i} v + b_{-i},\ i = 1, \ldots, m+1-l.$$

This requires at most $3(m+1)$ operations. If $l = 0$, then the program is unbounded, and we are done in $\leq 3(m+1) \leq 100m + 100$ operations. So we assume that $l \geq 1$.

If the numbers $a_{2i-1} - a_{2i}$ and $b_{2i-1} - b_{2i}$ have different signs or both are zero for some integer i such that either $1 \leq i \leq l/2$ or $1 \leq -i \leq (m+1-l)/2$, then we can drop one of these two constraints. This involves at most $m+1$ subtractions and $0.5m+0.5$ sign comparisons. We denote by $l' \leq l$ and $m' - l' - 1$ the remaining numbers of constraints of type $\leq$ and $\geq$. Note that $l' \geq 1$. If $m' \leq 12$, then we are done in

$$3(m+1) + (m+1) + (0.5m + 0.5) + 100 \cdot 12 + 100$$

$$= 4.5m + 4.5 + 100 \cdot 12 + 12 < 100m + 100$$

operations. So we assume that $m' \geq 13$.

We write the remaining constraints as

$$u \leq a'_i v + b'_i,\ i = 1, \ldots, l',$$

$$u \geq a'_{-i}v + b'_{-i},\ i = 1, \ldots, m' + 1 - l'$$

and we remember the computed numbers $q_i = a_{2i-1} - a_{2i}, p_i = b'_{2i-1} - b'_{2i}$ with $\text{sign}(p_i) = \text{sign}(q_i) \neq 0$.

Now we compute $v_i = p_i/q_i$ for $1 \leq i \neq l'/2$ and $1 \leq -i \neq (m' + 1 - l')/2$. This gives

$$k = \lfloor l'/2 \rfloor + \lfloor (m + 1 - l')/2 \rfloor \leq (m' + 1)/2$$

numbers. The number of arithmetic operations is

$$k \leq (m' + 1)/2 \leq 0.5m + 0.5.$$

Then we compute a median v_0 of these $k \geq 0.5m' - 1.5$ numbers. It requires at most $18k + 18 \leq 9m + 27$ comparisons (see A10).

Next we find the maximum u'' of $a'_i v_0 + b'$ with $1 \leq -i \leq m' + 1 - l'$. This requires $m' - l'$ comparisons.

We also find the minimum u' of $a'_i v_0 + b'$ with $1 \leq l'$ computing at the same time the set Y of $\text{sign}(a'_i)$ with $a'_i x_i + b'_i = u'$. This requires at most $2l' - 1$ comparisons. The total number of operations is at most

$$(4.5m + 4.5) + (0.5m + 0.5) + (9m + 27) + (m' - l') + (2l' - 1)$$
$$\leq 16m + 31.$$

If $u'' \leq u'$ and Y contains either 0 or both 1 and -1, then v_0 is an optimal solution and u' is the optimal value. So we are done in $(16m + 31) + 1 < 100m + 100$ operations.

If either $u'' > u'$ or $Y = \{-1\}$, then $v^* < v_0$ for each optimal solution v^*. In the remaining case, when $u'' \leq u'$ and $Y = \{1\}$, then there is an optimal solution $v^* \leq v_0$. In both cases, we know what side of x_0 to search for v^*.

Now we can drop one constraint for every v_i (including v_0) that is on the other side of v_0 than v^*. This means that at least $k/2$ constraints drop, and at most

$$m' - k/2 \leq m' - (m' - 3)/4 = 0.75m' + 0.75 \leq 0.75m + 0.75$$

stay. By the induction hypothesis, we can finish in at most

$$100(0.75m + 1.75)$$

operations. Thus, in at most

$$(16m + 31) + 1 + 100(0.75m + 1.75) = 91m + 207 < 100m + 100$$

operations, we can solve our linear program.

The number $100m + 100$ can be improved by a more careful accounting.

A9. Integer Programming

This section is about linear programs with additional conditions that some variables are integers. A particular case, is boolean, binary, or combinatorial programming when all variables are required to be 0 or 1. The general case with all variables being bounded integers can be reduced to binary case by writing variables in base 2 (binary representation). In the case of combinatorial programming, a solution can be thought as a set of variables (taking value 1). If all variables are required to be integers, we have a *pure* integer program.

Example A9.1. Maximize $f(n)$ subject to $1 \leq n \leq N$, n an integer, where N is a given positive integer.

This is an univariate integer program. We want to find a maximal term in the finite sequence $f(1), f(2), \ldots, f(N)$. The problem can be solved by $N-1$ comparisons (see A10 below).

Example A9.2. *Job Assignment Problem* (see p. 90).

This is a boolean program. It can be reduced to a linear program, see p. 192.

Example A9.3. *Knapsack Problem:*

$$cx \to \max, ax \leq b, x = [x_1, \ldots, x_n]^T$$

with integers $x_i \geq 0$, where $a, c \geq 0$ are given rows and b is a given number.

This integer program models the maximum value of a knapsack that is limited in weight by b, where x_j is the number of items of type i with value c_i and weight a_i. In a bounded knapsack problem, we have additional constraints $x_i \ e_i$. In a 0-1 knapsack problem, all $e_i = 1$.

Example A9.4. *Traveling Salesman Problem* (TSP, cf. [GP]).
Given an $n \times n$ cost matrix $[c(i,j)]$, a *tour* is a permutation of $[\sigma(1), \ldots, \sigma(n)]$ of *cities* $1, 2, \ldots, n$. A tour means visiting each city exactly once, and then returning to the first city (called home). The cost of a tour is the total cost

$$c(\sigma(1), \sigma(2)) + c(\sigma(2), \sigma(3)) + \ldots + c(\sigma(n-1), \sigma(n)) + c(\sigma(n), \sigma(1)).$$

The TSP is to find a tour of minimum total cost. It can be stated as an integer program using (like the job assignment problem) the binary variables x_{ij}. ■

Using binary variables we can reduce logically complicated systems of constraints to usual systems where all the constraints are required to be satisfied. For example, the program

> minimize $f(x)$ satisfying any two of the following three constraints $g_1(x) \le 0, g_2(x) \le 0, g_3(x) \le 0$

can be written as

> $f(x) \to \min$, $y_1 g_1(x) \le 0$, $y_2 g_2(x) \le 0$, $y_3 g_3(x) \le 0$, $y_1 + y + 2 + y_3 = 2, y_i$ binary.

Now we discuss some methods of solving integer programs.

Rounding Linear Solutions

Dropping the conditions that the variables are integers, we obtain a linear program, the LP-*relaxation* of the integer program, IP. If an optimal solution of the LP satisfies the integer restriction (as for the job assignment problem), it is optimal for IP. In general, rounding the optimal solution for LP, we obtain a "solution" for IP. This solution need not be feasible, and if it is feasible, it need not be optimal. However, it is used sometimes in real life when better solutions are hard to find.

Exhaustive and Random Enumeration

When the variables restricted to be integers are bounded, the IP is reduced to a finite set of LPs by running over all possible values for those variables. This approach is practical only when the number of LPs is small. In general, we can choose the values at random. The more choices tried, the closer to 1 is the probability of finding an optimal value for IP.

Reduction of IP to an LP Using Convex Hull

Consider the feasible set S for our IP and its convex hull S' (i.e., the set of all convex combinations of all points in S). Then S' can be given by a finite set of linear constraints, and optimization of our objective function $f(x)$ over S is equivalent to optimization of $f(x)$ over S', which is a linear program. Moreover, every extreme point of S' that is optimal belongs to S and hence is optimal for the IP. Although theoretically this approach reduces any IP to a LP, this method works only when S' can be described by a small number of linear constraints. See §3 for examples.

Branch-and-Bound Method

We outline this method for integral programs (with binary variables $x_1, \ldots, x_n$) using the best-first branching rule although it can be

used for any mathematical program where some variables are integers, and there are different branching rules. We start with the LP-relaxation LP_0 of our IP minimization program IP_0 (i.e., the linear program obtained by replacing the conditions $x_i^2 = x_i$ in IP_0 by the conditions $0 \leq x_i \leq 1$). If an optimal solution of LP_0 is integral, we are done. In any case the optimal value for LP_0 is a lower bound for the optimal value of IP_0. Next we split IP_0 into two IPs, IP_1 and IP_2 fixing $x_1 = 0$ or 1. We solve the corresponding linear programs obtaining lower bounds v_1 and v_2 for the optimal values of IP_1 and IP_2. Next we branch the program with lower bound into two programs, setting $x_2 = 0$ and $x_2 = 1$. Continuing the process, we have at step t a tree with t nodes. At each node we have a lower bound for the corresponding IP obtained by solving the corresponding LP-relaxation. If no optimal solution for the nodes with minimal bound is integral, we branch one of these nodes into two. The process terminates in at most $2^n - 1$ branchings.

Cutting Plane Method

We start by solving the LP-relaxation of our IP problem. If the optimal solution satisfies the integrability constraints, we are done. Otherwise, an additional linear constraint is constricted, which cuts out the optimal solution but is satisfied by all feasible solutions (or at least by all optimal solutions) of IP. The process is repeated, so we obtain a sequence of linear programs with a decreasing sequence of the feasible regions. Many ways to construct the cutting constraint were suggested for which it was proved that the process terminates in finitely many steps. However, it is common that the number of steps is too large. *Polyhedral annexation* is a cutting plane approach to finding an optimal solution known to lie at an extreme point of a polyhedron, P. The general algorithm is to start at some extreme point and solve the polyhedral annexation problem. This will result in ascertaining that the extreme point is (globally) optimal, or it will generate a recession direction from which a convexity cut is used to exclude the extreme point. The approach generates a sequence of shrinking polyhedra by eliminating the cut region. Its name comes from the idea of annexing a new polyhedron to exclude from the original, homing in on the extreme point solution.

Many textbooks on linear programming contain chapters on integer programming. Also, there are books on integer programming (cf. [ES], [W2], [KV], [S2]).

A10. Sorting and Order Statistics

A lot of sorting is done by humans and computers. Sorting is a significant part of data processing. So finding efficient ways to sort is important. Sorting here means ordering items in a linear list, like making a list of students in class. More precisely, given $n \geq 1$ numbers $a_1, \ldots, a_n$, let $b_1 \leq \cdots \leq b_n$ be the same numbers sorted.

We want to find these numbers b_i as fast as possible. Usually, any method of sorting also finds a permutation σ such that $b_{\sigma(i)} = a_i$.

Sorting involves comparison of numbers and possibly moving them around. We will count only the number of comparisons, which we call steps for short. So the problem is how to sort a given set of numbers in a smallest number of steps.

Thus, we are interested only in the cost of collecting information (sufficient to order the numbers) about the relative size of numbers and ignoring costs of storing and using this information as well as the complexity of the algorithm.

Since there are $n!$ permutations of n numbers, it is well known that any sorting method requires at least $\lceil \log_2 n! \rceil$ steps, where $\lceil x \rceil$ denotes the least integer $t \geq x$. Here we prove the following well-known result (cf. [K3]).

Theorem A10.1. We can sort n numbers in

$$F_0(n) = \sum_{i=1}^{n} \lceil \log_2 i \rceil = m\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1$$

steps. ∎.

Proceeding by induction on n, it suffices to prove the following.

Lemma A10.2. Given a sorted list of numbers $b_1 \leq \cdots \leq b_{n-1}$ and another number a_n, it takes at most $\lceil \log_2 n \rceil$ steps to sort all n numbers.

Proof. In other words, we have to prove that k steps are sufficient to insert a new number b_n into an ordered list of $n = 2^k - 1$ numbers. The case $n = k = 1$ requiring one step is trivial. Let $k \geq 2$. We proceed by induction on k, using a bisection method that is similar to that in A2. Namely, we compare a_n with the median $b_{(n+1)/2}$. After this we have the task of inserting a_n into a sorted list of $(n-1)/2 = 2^{k-1} - 1$ numbers, which can be done in $k-1$ steps by the induction hypothesis. ∎

It is clear that $k-1$ steps are necessary and sufficient to find $\min = \min[a_1, \ldots, a_n]$. The same is true for the maximal number.

If we have some additional information on the sequence a_i this can be used to find min faster. For example, suppose that the sequence is *unimodal* (i.e., there is i^* such that a_i strictly decreases from $i \le i^*$ and strictly increases from $i \ge i^*$). A method for finding $\min = a_{i^*}$ using the minimal number of evaluations, similar to Fibonacci search, is called *lattice search.* It is more efficient than the bisection method which finds the minimum in $\lceil \log_2 n \rceil$ steps.

Suppose again that we know nothing about a_i. While it takes about $\log_2 n!$ steps to find the ordered list b_i it turns out that any particular b_i (an order statistic), including the median $b_{\lceil n/2 \rceil}$, may be computed faster—namely, in Cn steps with C bounded over all n, k; see [K3].

The known proofs with small C are quite long, so we give a couple of examples and then sketch a simple proof with $C = 18$.

For instance, $b_1 = \min(a_i)$ can be computed by $n-1$ comparisons.

As another example, consider the particular case $n = 5$. Sorting takes at least $\lceil \log_2(5!) \rceil = 8$ steps. On the other hand, as mentioned previously it takes $n-1 = 4$ steps (comparisons) to find b_1. After this, it takes three more steps to find b_2, the minimum of the remaining numbers. So b_2 can be found in seven steps. Similarly, b_5 and b_4 can be found in four or seven steps. Finally, the median b_3 can be found in seven steps as follows.

First we sort a_1, a_2, a_3, a_4 by the insertion method in $1+2+2 = 5$ steps and obtain $c_1 \le c_2 \le c_3 \le c_4$. Then we compare a_5 with the medians c_2 and c_3 (two more steps). If $c_2 \le a_5 \le c_3$, then $b_3 = a_5$. If $a_5 \le c_2$, then $b_3 = a_2$. If $a_5 \ge c_3$, then $b_3 = c_3$. Thus, each b_i can be found in seven steps.

It is known [DZ1] that any statistics b_i can be found in at most $2.95n + o(n)$ steps and that at least $2.01n + o(1)$ steps are required in the worst case [DZ2]. Here $o(n)$ [respectively, $o(1)$] stands for a sequence such that $o(n)/n \to 0$ [respectively, $o(1) \to 0$] as $t \to \infty$.

Now we sketch a proof that $18(n-1)$ steps are sufficient to find the k^{th} smallest number b_k. We proceed by induction on n. The cases $n \le 34$ are trivial because then we can sort n numbers in $18(n-1)$ steps. So let $n \ge 35$.

We write $n = 10l + 5 + r$ with $0 \le r \le 9$. We partition the first $10l + 5$ numbers a_i into $2l + 1$ 5-tuples and find the medians $c_1, \dots, c_{2l+1}$ in each 5-tuple. This can be done in $7(2l + 1)$ steps. By the induction hypothesis, we can find the median d of c_j in $36l$ steps.

Now we have $3l + 2$ numbers a_i on the right of d and $3l + 2$ numbers a_i on the left of d. In

$$n - 6l - 5 = 4l + 5$$

steps we place the remaining $n - 6l - 5$ numbers a_i on the right or left of d.

Now we have to find a certain statistic among $n' \le n - 3l - 3$ numbers on the right of d or among $n'' \le n - 3l - 3$ numbers on the left of d. By the induction hypothesis, we can do this in

$$18(n - 3l - 4)$$

steps.

Thus, the total number of steps is at most

$$7(2l + 1) + 36l + 4l + 5 + 18(n - 3l - 4) = 18n - 60 \le 18n - 18.\blacksquare$$

Finally we discuss the problem of finding saddle points in a given $m \times n$ matrix $[a_{i,j}]$ (or proving that they do not exist). This problem appears when we try to solve a matrix game, and it can be stated as an integer program.

Given i, j it takes $m+n-2$ steps to check whether this position is a saddle point. On the other hand it takes $n(m - 1)$ steps to find all maximal entries in every column. Similarly, it takes $m(n - 1)$ steps to find all minimal entries in every row. So after $2mn - m - n$ steps we are done (the positions selected twice are the saddle points). No faster method is known.

However, there is a faster method for finding a *strict* saddle point [i.e., (i, j) such that $a_{i,k} > a_{i,j} > a_{l,j}$ for all $k \ne j$ and $l \ne i$ (or proving nonexistence)]. The method [BV] starts with sorting the numbers $a_{i,i}$ on the main diagonal in $F_0(m)$ steps (assuming that $m \le n$), and terminates in

$$F_0(m) + F_0(m - 1) + n + m - 3 + (n - m)\lceil \log_2(m + 1) \rceil$$

steps.

A11. Other Topics and Recent Developments

We have not mentioned several important topics in mathematical programming including special classes of mathematical programs and special methods devised for solving those programs.

Besides explicitly given numbers, the data may include parameters or external variables (e.g., coefficients of linear functions) and/or random data. In §14, we mentioned parametric programming in the context of linear programming. *Stochastic programming* (cf. [BL], [KW]) deals in particular with uncertainty in data which is also the main concern in *fuzzy programming* (cf. [C1], [RI], [RV]). Fuzzy sets are also used to represent preferences which is connected with goal programming.

To solve large problems, special sophisticated tricks are needed to handle data, and methods are modified to obtain a good solution in a reasonable time. Here are some recent books on large-scale optimization: [B2], [T]. For large linear programs there are revised versions of the simplex method, where special attention is paid to handling data. For example, in the case of a very large number of columns in tableaux, they are generated during pivoting. Column generation is dual to the cutting plane method. Also, the simplex method has been modified to handle upper bounds on variables in a special way.

Some large linear programs can be split into subprograms that are weakly related between themselves. This leads to decomposition methods (nested programming) like Dantzig-Wolfe methods. Similarly, aggregation methods reduce solving a large LP to solving a sequence of smaller LPs.

In *fractional programming*, the objective function $f(x)$ and the constraint functions are sums of ratios of the form $a(x)/b(x)$ with affine functions $a(x), b(x)$ such that $b(x) > 0$ over S. In the one-term case, the problem looks like

$$a_0(x)/b_0(x) \to \min, a_i(x)/b_i(x) \le d_i \text{ for } i = 1, \dots, m$$

with affine functions $a_i(x), b_i(x)$ and constants f_i. This case is special because then the program is equivalent to a linear program [assuming that all $b_i(x) \le 0$ in the feasible region]. See [C-M].

In *separable programming*, the objective function $f(x)$ and every constraint function is a sum of univariate functions $f_i(x_i)$. Piecewise approximations and linear programming are used to solve such programs (cf. [S3]).

In polynomial programming, the objective function $f(x)$ and the constraint functions $g_i(x)$ in (A1.9) are polynomials. The linear programming correspond to the case of total degree 1. It is easy to operate polynomial functions (e.g., to compute their derivatives), and some questions, like the existence of feasible or optimal solutions, can be answered theoretically in finitely many arithmetic operations with rational data (but we do not know how to do it efficiently in higher-degree multivariate cases). However, solving polynomial programs cannot be much easier than solving programs with continuous functions because continuous functions over bounded regions can be approximated by polynomials.

The transportation problem (see Chapter 6) is a particular case of various *network problems.* Some of them can be reduced to transportation problem, and some share the property that integral data result in integral optimal solutions. Many textbooks on linear programming treat network problems, and there are special books on network problems, including nonlinear ones (cf. [ES]). Interior point methods are used nowadays for solving large network problems.

Dynamical programming (cf. [DL]) is concerned with optimal decisions over time. For continuous time, it is used in optimal control and variational calculus. For discrete time, we have multistage (multiperiod) models. The main idea in a multiperiod decision process is solving the problem from the end, going back in time. Position games (a.k.a. games in extensive form) use this approach, too (cf. [FSS], [M]).

One popular application of linear programming is *data envelopment analysis;* see

http://www.banxia.com/, http://www.deazone.com/.

Neural networks, which imitate biological neural systems, are used to solve some optimization problems ("learning" algorithms), and mathematical programming is used for designing efficient neural networks.

An iterative method may depend on several parameters. Choosing parameters for a mathematical program with an objective function which is difficult to compute or/and we know little about or/and a program with a complicated feasible region could be a daunting task. One approach is *genetic* or *evolutionary programming.* We start with several algorithms, called strings, with parameters and initial points chosen at random. After a few iterations for each string, strings are sorted according to improvement in the objective function. Bad strings are eliminated. Good strings are paired

up, and their "offsprings" appear with some values of parameters exchanged (crossover), changed a little bit at random (mutations), or combined in some ways. This approach is particularly attractive combined with parallel computing (many CPUs). (See [CVL], [LP].)

In general, progress in hardware (such as advances in parallel computers, quantum computers, and DNA computers) stimulates new approaches in mathematical programming (cf. [CP2], [DPW], [G], [NC], [LP], [MCC]).