1 Introduction

1.1 Formalization

Optimization problems arise in numerous applications. The user is confronted with the task to make some choice in order to minimize (maximize) some cost (performance) under some conditions. In its original form the problem can be formulated in various ways, such as

- design a wing shape which minimizes the drag while assuring a certain lift;
- find the path for a robot arm which consumes the minimal time while visiting given points;
- design an identification experiment on a plant which gives maximal information in given time.

In general, the user has to choose decision variables in order to minimize an objective function with respect to some constraints.

The first step to the solution of this problem is to *formalize* it, i.e., to bring it into a form which is amenable to a mathematical treatment. A formalized finite-dimensional optimization problem looks as follows:

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

In order to transform the problem to this form we have to identify

- the *decision variable* $x \in \mathbb{R}^n$
- the *objective function* $f : \mathbb{R}^n \to \mathbb{R}$
- the *feasible set* $X \subset \mathbb{R}^n$.

This is often itself a far from nontrivial task. When setting up the mathematical description we must keep in mind the solvability of the resulting problem. This may necessitate to deviate from the original problem formulation and to make *approximations*.

The feasible set $X \subset \mathbb{R}^n$ can, e.g., be represented by

- scalar equalities $g_i(x) = 0$
- scalar inequalities $h_j(x) \leq 0$
- matrix inequalities $A_k(x) \succeq 0$
- norm inequalities $||Ax|| \leq c$
- black-box oracles, ...

The cost function may be given

- analytically
- by a black-box oracle
- with a (sub-)gradient
- with a Hessian, ...

Some combinations of cost function an constraints lead to standard problem classes:

- Linear program (LP): linear cost function, linear equality and inequality constraints
- Quadratic program (QP): quadratic cost function, linear equality and inequality constraints
- Standard quadratic program (StQP): quadratic cost function, feasible set is the simplex $\{x \in \mathbb{R}^n \mid \langle \mathbf{1}, x \rangle = 1,\ x \geq 0\}$

- Mixed integer linear program (MILP): linear cost function, binary, integer, and linear equality and inequality constraints
- Quadratically constrained quadratic program (QCQP): quadratic cost function and constraints
- Semi-definite program (SDP): linear cost function, linear equality constraints, linear matrix inequality constraints

1.2 Examples

Uniform Approximation: We want to approximate a (complicated) function $g(x)$ uniformly on a domain $G \subset \mathbb{R}^m$ by a linear combination of basis functions $f_k : G \to \mathbb{R}$, $k = 1, \dots, n$. In order to make this task feasible, we approximate the domain G by a *discrete* set $D = \{x_1, \dots, x_N\} \subset G$, e.g., a dense grid. Then we may write the problem as follows:

$$\min_{c \in \mathbb{R}^n} \max_{j=1,\dots,N} \left| g(x_j) - \sum_{k=1}^{n} c_k f_k(x_j) \right|.$$

This is an unconstrained problem of the desired form (1), with $c = (c_1, \dots, c_n)$ being the decision variable, and the maximum of the absolute value being the cost function. Note that this function is considered here as a function of c. The other components of the function are known from the original formulation of the problem and constitute the *data*.

However, the cost function is a piece-wise linear function of c and is as such too complicated for being minimized straightforwardly. We shall therefore introduce an *auxiliary variable* $\tau \in \mathbb{R}$ and add it to the decision variables, and introduce constraints, as follows:

$$\min_{(\tau,c) \in \mathbb{R} \times \mathbb{R}^n} \tau : \qquad -\tau \leq g(x_j) - \sum_{k=1}^{n} c_k f_k(x_j) \leq \tau.$$

Now the cost function is *linear* in the decision variables, and the feasible set is given by *linear* inequalities. Such a problem is called a *linear program* (LP) and can be solved by standard optimization software.

The process of adding additional auxiliary variables to the problem is called *lifting*. The feasible set of the new augmented problem then projects to the feasible set of the original problem, i.e., it is a *lift* of the original feasible set.

Resource Allocation: Suppose we may fabricate a number of products which we can sell at prices $p_1, \dots, p_n$, respectively. The production of a unit of product l consumes a_{kl} units of raw material k, $k = 1, \dots, K$, of which a total quantity of r_k units is available. We wish to choose the quantities $x_1, \dots, x_n$ of each product to be produced in order to maximize the revenue.

The problem can be formalized as follows:

$$\min_x -\langle p, x \rangle : \qquad Ax \leq r, \ x \geq 0,$$

where A is the $K \times n$ matrix made up of the coefficients a_{kl}, and x, r, p are the vectors made up of the corresponding elements. The constraint $x \geq 0$ is necessary to prevent the conversion of products back to raw materials, which would correspond to a negative quantity x_l.

Again the cost function and the constraints are linear in the decision variables, and the problem has been formalized as an LP.

Max-Cut: We are given a weighted graph $G = (V, E)$ with vertex set $V = \{v_1, \dots, v_n\}$ and edge set $E = \{e_1, \dots, e_m\}$, where each edge e_k has been attached a nonnegative weight w_k. The Max-Cut problem consists in separating (cutting) the vertex set into a disjoint union $S \cup T$ of two subsets such that the sum of the edge weights between the two subsets is maximized.

We shall represent a cut by a vector $x \in \{-1, +1\}^n$, i.e., a vertex of a hyper-cube, where the indices of elements $x_j = -1$ correspond to the vertices in S and the indices of elements $x_j = 1$ to vertices in T. Note that $-x$ and x represent the same cut, as the transformation $x \mapsto -x$ corresponds to an exchange of the sets S, T.

Consider the real symmetric matrix $A(x) = \frac{1}{4}(\mathbf{1} - xx^T)$, where $\mathbf{1}$ denotes the all-ones matrix. Then $A_{ij} = 0$ if v_i, v_j are in the same subset S or T, and $A_{ij} = \frac{1}{2}$ otherwise.

Construct a real symmetric $n \times n$ matrix W such that the element W_{ij} equals the edge weight w_k if the vertices v_i, v_j are linked by edge e_k, and zero if v_i, v_j are not linked. Then the sum of the edge weights of the cut is given by the expression $\langle A, W\rangle = \sum_{i,j=1}^n A_{ij}W_{ij}$. We thus arrive at the formulation

$$\min_{x\in\{-1,+1\}^n} (-\langle A(x), W\rangle) = \min_{X\in\mathcal{MC}} -\frac{1}{4}\langle \mathbf{1} - X, W\rangle,$$

where $\mathcal{MC}$ is the *MaxCut polytope*, which is defined as the convex hull of the set of matrices $\{xx^T \,|\, x \in \{-1,+1\}^n\}$.

In the first formulation the cost function is quadratic, in the second formulation it is linear, but the feasible set consists of an exponential number of points in the first case and has a complicated structure in the second case. It can actually be proven that Max-Cut is an NP-hard problem.

Min-Cut: Here were are confronted with the same problem as for Max-Cut, but we want to *minimize* the weight of the cut. The weights w_k are assumed to be nonnegative. The cut is to be chosen to separate two given vertices s, t. In contrast to Max-Cut this problem can be reduced to a polynomially sized LP. We shall show below that Min-Cut is equivalent to Max-Flow.

Max-Flow: Let G be a directed graph with vertex set $V = \{v_1 = s, v_2, \ldots, v_{n-1}, v_n = t\}$ and edge set $E = \{e_1, \ldots, e_m\}$. The distinguished vertices s, t are called the source and the sink. To each edge e_k there are attached weights $w_k^{\pm} \geq 0$. These are interpreted as flow capacities in and against the direction of the edge. As in the MaxCut problem we build a matrix W, with $W_{ij} = w_k^+$ if edge e_k is from vertex v_i to v_j, $W_{ij} = w_k^-$ if e_k is from v_j to v_i, and $W_{ij} = 0$ if v_i, v_j are not linked by an edge. Note that W is not necessarily symmetric, as $w_k^{\pm}$ do not need to coincide. The problem consists in finding the maximal flow from the source to the sink through the network.

We shall represent a flow through the network by a skew-symmetric $n \times n$ matrix F. The element F_{ij} designates the actual flow from vertex v_i to vertex v_j. If there is no flow between v_i and v_j, then $F_{ij} = 0$, if the flow is from v_j to v_i, then $F_{ij} = -F_{ji} < 0$.

The Max-Flow problem can then be formalized as follows:

$$\min_{F=-F^T} \sum_{i=1}^{n-1} F_{i1} : \quad F \leq W, \quad \sum_{i=1}^{n} F_{ij} = 0 \quad \forall\, j = 2, \ldots, n-1.$$

Here the decision variable is the skew-symmetric matrix F of flows. The flow from s to t is given by the sum of outflows from the source minus the sum of inflows in the source, which is equal to $-\sum_{i=1}^{n-1} F_{i1}$. The inequality $F \leq W$ has to be interpreted element-wise and ensures that the flows remain bounded by the corresponding capacities, $|F_{ij}| \leq W_{ij}$ for all i, j. The equalities are balance equations that ensure the sum of inflows into an intermediate vertex equals the sum of outflows.

The cost function and the constraints are linear in the $\frac{n(n-1)}{2}$ decision variables F_{ij}, $i < j$, and the problem reduces to an LP.

Equivalence with Min-Cut: To the Min-Cut problem we associate a Max-Flow problem as follows. The vertex set in the Max-Flow problem is the same as in the Min-Cut problem, and to any undirected edge e_k in the Min-Cut problem there corresponds a directed edge in the Max-Flow problem, with weights $w_k^{\pm} = w_k$. An undirected edge in the Min-Cut problem is hence interpreted as a tube allowing a flow in both directions and bounded by the edge weight in absolute value.

Clearly every flow from s to t is bounded from above by the minimum cut. We shall now show that the maximal flow is actually equal to the minimum cut.

Let F^* be the flow matrix corresponding to the maximal flow. We then define the *residual network* as the network allowing flows F bounded by the inequalities $F \leq W - F^*$, i.e., flows F such that $F + F^*$ is a valid flow through the original network. Note that $W - F^* \geq 0$, because F^* represents a feasible flow. Since F^* is the maximal flow, the residual network does not allow any positive flow from s to t. We then define S as the

set of vertices which can be reached from s through the residual network (i.e., vertices v such that there exists a positive flow from s to v), and T as the set of remaining nodes. The maximal flow F^* then flows through the cut defined by S and T. Suppose that the value of this cut is strictly larger than the value of F^*. Then there exists an edge e_k in the cut, linking $v_i \in S$ and $v_j \in T$, such that $w_k > F^*_{ij}$. But then v_j is reachable from v_i through the residual network, contradicting the definition of S and T. Hence the value of the cut must equal the value of the flow F^*.

Since the value of F^* is a lower bound to every cut separating s and t, the value of the cut defined by S and T must be minimal.

Ford-Fulkerson method: For the Max-Flow / Min-Cut problem there exist several algorithms that are more efficient than solving the LP. The conceptually simplest class are the Ford-Fulkerson methods. They use the fact that as long as a given flow is not maximal, the residual network will allow to link the source node to the target node. However, the sub-problem of finding a linking path is solvable in a number of steps which is proportional to the number of edges e. Having found a path connecting s to t in the residual network, we may add the flow along this path to the current flow and obtain the next iterate with a strictly larger capacity.

Whether the method terminates in a finite number of steps depends on the specific algorithm looking for the linking path. The version of Edmonds-Karp finds the shortest linking path by performing a breadth-first search. It has a $O(ne^2)$ overall complexity.

1.3 Convexity

The difference between Min-Cut and Max-Cut comes from the fact that the former can be reduced to a convex problem with polynomial size in the data, while the latter cannot.

Definition 1.1. A set $X \subset \mathbb{R}^n$ is called *convex* if for every $x, y \in X$ and $\lambda \in (0,1)$ we have $\lambda x + (1-\lambda)y \in X$.

A function $f : X \to \mathbb{R}$ on a convex set X is called *convex* if for every $x, y \in X$ and $\lambda \in (0,1)$ we have $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$, and *concave* if the opposite inequality holds.

If the feasible set X and the cost function $f(x)$ in problem (1) are convex, then the problem is called convex. Convexity of the problem guarantees that every local minimum is also a global minimum.

Examples of simple convex problems

Convex quadratic function: We wish to minimize the function $f(x) = \frac{1}{2}x^T Ax + b^T x$ with A real symmetric and with positive eigenvalues. The minimizer can be found analytically via the gradient condition $f'(x) = Ax + b = 0$. This problem hence reduces to solving a linear system of equations $Ax = -b$.

Linear function over ellipsoid: We wish to solve the problem

$$\min_x c^T x : \qquad (x - x_0)^T A^{-1}(x - x_0) \leq 1,$$

with A real symmetric and with positive eigenvalues, and with $c \neq 0$. The feasible set of the problem is an ellipsoid centered on x_0.

The minimizer x^* has to lie on the boundary of the ellipsoid, with the level surface through the minimizer being tangent to the ellipsoid and the gradient pointing inside the ellipsoid. Therefore there exists $\lambda < 0$ such that $\lambda c = 2A^{-1}(x^* - x_0)$, yielding $x^* = x_0 + \frac{\lambda}{2}Ac$. We also get

$$1 = (x^* - x_0)^T A^{-1}(x^* - x_0) = \frac{\lambda^2}{4} c^T A A^{-1} A c = \frac{\lambda^2}{4} c^T A c,$$

yielding $\lambda = -\frac{2}{\sqrt{c^T A c}}$ and finally

$$x^* = x_0 - \frac{Ac}{\sqrt{c^T A c}}.$$

Line search with bisection: We wish to minimize a convex function $f : \mathbb{R} \supset I \to \mathbb{R}$, or more generally a function which is strictly monotonically decreasing for $x \leq x^*$ and strictly monotonically increasing for $x \geq x^*$, where x^* is the minimizer of the function, and I is a (not necessarily finite) interval. We assume that a minimizer

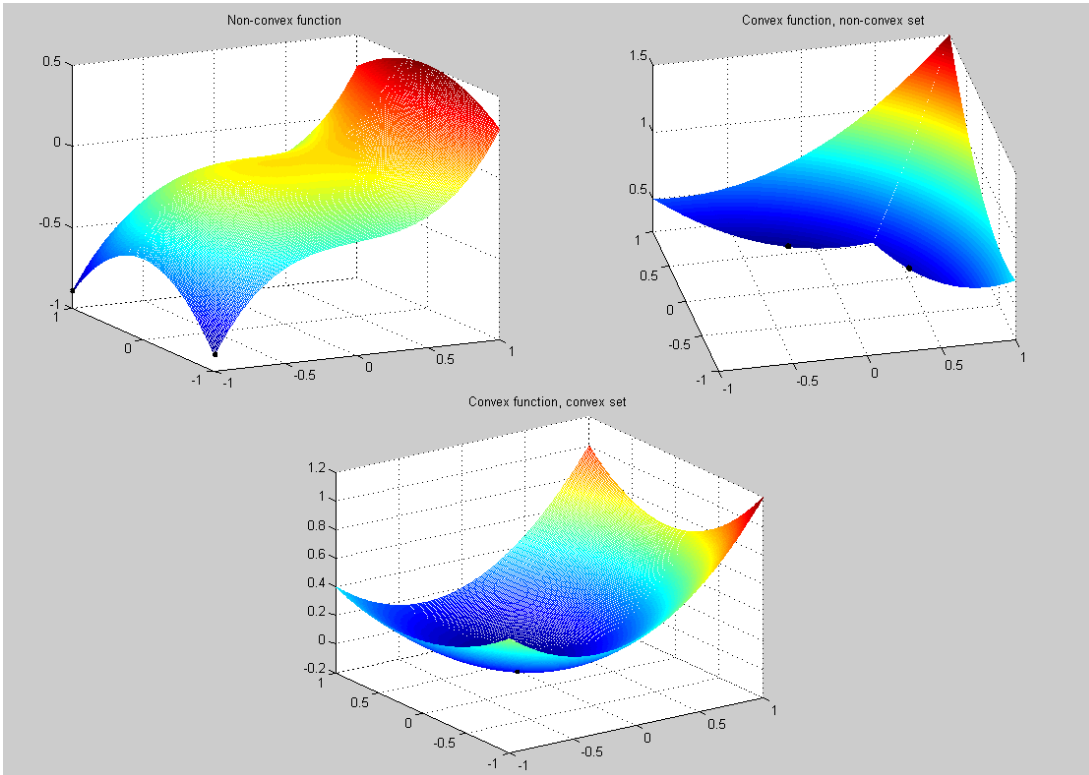

Figure 1: Uniqueness of minima in the convex and non-convex case

x^* exists and that given a point x, there is a means to determine whether $x \leq x^*$ or $x \geq x^*$, e.g., by computing the derivative $f'(x)$.

We commence with an initial guess x_0 and check whether x_0 is smaller or larger than x^*. Assume without loss of generality that $x_0 < x^*$. Then we choose $d > 0$ and $\lambda > 1$ and compute recursively a sequence of points $x_{k+1} = x_k + d \cdot \lambda^k$. After $n \approx \frac{\log(1+\frac{(x^*-x_0)(\lambda-1)}{d})}{\log \lambda}$ steps we arrive at a point x_n such that $x_{n-1} \leq x^* \leq x_n$, and $x_n - x_{n-1} < d \cdot \lambda^n$.

We now set $I_0 = [x_{n-1}, x_n]$ and construct recursively a sequence of nested intervals $I_k \subset I_{k-1}$ with mid-points y_k. Here I_k is either the left half or the right half of I_{k-1}, depending on whether $y_k \geq x^*$ or $y_k \leq x^*$. After $\log_2 \frac{x_n - x_{n-1}}{\epsilon} < \log_2 \frac{d \cdot \lambda^n}{\epsilon}$ steps we arrive at an interval of length ϵ which contains a minimizer.

Line search with golden ratio: We wish to solve the same problem as before, but this time given a point x we only have information on the value $f(x)$.

We commence with two initial points $x_0 < x_1$ and determine the corresponding function values. Suppose without loss of generality that $f(x_0) > f(x_1)$. Then there exists a minimizer $x^* > x_0$. We compute again a sequence of points $x_{k+1} = x_k + d \cdot \lambda^k$, until after $n = O(\log \frac{x^* - x_0}{d})$ steps we arrive at a point x_n such that $f(x_n) \geq f(x_{n-1})$. Thus we have three points $x_{n-2} < x_{n-1} < x_n$ such that $f(x_{n-1}) \leq \min(f(x_{n-2}), f(x_n))$. It follows that the minimizer lies in the interval $I_0 = [x_{n-2}, x_n]$.

We now construct recursively a sequence of nested intervals $I_{k+1} \subset I_k$ which contain the minimizer. At each step we have already computed the end-points x_{k-}, x_{k+} of the interval I_k and a point x_{km} in its interior. Assume now that $\frac{x_{k+} - x_{km}}{x_{k+} - x_{k-}} = \alpha = \frac{\sqrt{5}-1}{2} \approx 0.618$, i.e., x_{km} divides I_k at a golden ratio. We choose the next iterate $x_{k+1} = x_{k-} + x_{k+} - x_{km}$ such that it divides I_k also at a golden ratio, but lies in the other half of I_k. If now $f(x_{k+1}) > f(x_{km})$, then we discard the end-point x_{k+} of I_k which is closer to x_{k+1}, if $f(x_{k+1}) < f(x_{km})$, then we discard the other end-point x_{k-} which is closer to x_{km}. We again end up with three points $x_{k\pm}, x_{km}, x_{k+1}$, two of which are the end-points of the new interval I_{k+1} and one of which divides this interval at a golden ratio. At each step the length of the interval decreases by a factor α. After $\log_{\alpha^{-1}} \frac{x_n - x_{n-2}}{\epsilon}$ steps we arrive at an interval of length ϵ which contains a minimizer.

In the line search methods considered above, the number of known digits of the minimizer increases linearly with the index k of the iteration. Such a convergence behaviour is called *linear*.

1.4 Ellipsoid method

If a *separating oracle* is available, then there exists a simple polynomial-time iterative method to solve general convex optimization problems, the *ellipsoid method*. Here a separating oracle, given a point $x \in \mathbb{R}^n$, outputs a non-zero vector g such that a minimizer x^* of the problem is guaranteed to satisfy the condition $\langle g, x^* - x\rangle \leq 0$. This may be, e.g., the gradient $f'(x)$ of the cost function at a feasible point x.

The initial data are an ellipsoid $E_0 = \{x \in \mathbb{R}^n \mid (x - x_0)^T P_0^{-1}(x - x_0) \leq 1\}$ centered at the initial iterate x_0 and containing a minimizer. At iteration step k we call the separation oracle at the center x_k of the ellipsoid E_k to obtain a vector g_{k+1} such that $\langle g_{k+1}, x_k - x^*\rangle \geq 0$. Then we compute the new ellipsoid E_{k+1} centered on the new iterate x_{k+1} by

$$x_{k+1} = x_k - \frac{1}{n+1}\frac{P_k g_{k+1}}{\sqrt{g_{k+1}^T P_k g_{k+1}}},$$

$$P_{k+1} = \frac{n^2}{n^2-1}\left(P_k - \frac{2}{n+1}\frac{P_k g_{k+1}(P_k g_{k+1})^T}{g_{k+1}^T P_k g_{k+1}}\right).$$

The volume of the ellipsoids E_k decreases exponentially and each ellipsoid is guaranteed to contain a minimizer of the problem.

The method is impractical in most situations due to its slow convergence and numerical instabilities. It can be used for small-dimensional convex problems if little information other than the separation oracle is available.

1.5 Non-convex problems

Non-convex problems are in general much more difficult to solve than convex problems. When using local descent methods one can hope only for convergence to a local minimum.

Below we present a number of methods for unconstrained minimization.

Line search: At each iteration, we choose a *descent direction* d_k at the current iterate x_k. Then we solve the one-dimensional problem

$$\min_{\alpha>0} f(x_k + \alpha d_k)$$

as a sub-problem. Note that we do not need to solve the sub-problem to high accuracy. A coarse approximation of the minimum will suffice for serving as the next iterate x_{k+1}.

If the function is of class C^1, then we may choose $d_k = -f'(x_k)$ and obtain a gradient descent algorithm. Note that we implicitly assume the presence of a Euclidean structure on the underlying space, since we identify the co-vector $-f'(x_k)$ with the vector d_k.

If the function is of class C^2 and with positive definite Hessian, then we may choose $d_k = -(f''(x_k))^{-1}f'(x_k)$. This will be a Newton-type method.

There exist schemes which combine directions from several previous steps and achieve an acceleration of convergence. Quasi-Newton schemes build an approximation of the Hessian based on previously computed values of the gradient.

Trust region methods: At each iteration, we construct an *approximation* of the cost function around the current iterate x_k. Then we construct a sub-problem from the original problem with the approximation instead of f and the additional constraint $||x - x_k|| \leq c$. The feasible set of this constraint is the *trust region* where we are confident in our approximation of f. If the minimizer of the sub-problem gives a lower as expected decrease of the original cost function, then we decrease the constant c.

Regularization: At each iteration we add a *regularizing term* to the objective function, usually proportional to $||x - x_k||^p$ for some $p \geq 1$, and approximate the objective function by, e.g., a second order Taylor polynomial. The regularizing term has the effect of holding the next iterate close to the previous one, where the approximation is valid.

Constrained optimization problems can be converted to unconstrained ones by adding barriers or penalty functions. While the former are defined on the feasible set and tend to $+\infty$ at its boundary, the latter are zero on the feasible set and penalize constraint violations by an increasing value outside of the feasible set.

Our strategy, however, will consist in approximating the original problem globally by a convex one with polynomial complexity.