Lecture 1

Introduction

(General formulation of the problem; Important examples; Black Box and Iterative Methods; Analytical and Arithmetical Complexity; Uniform Grid Method; Lower complexity bounds; Lower bounds for Global Optimization; Rules of the Game.)

1.1 General formulation of the problem

Let us start with fixing the mathematical form of our main problem and the standard terminology.

1.1.1 Problem formulation and terminology

Let $x$ be an $n$-dimensional real vector: $x = (x^{(1)}, \ldots, x^{(n)}) \in \mathbb{R}^n$, $G$ be a subset of $\mathbb{R}^n$, and functions $f(x), g_1 \ldots g_m(x)$ are some real-valued function of $x$.

In the entire course we deal with some variants of the following minimization problem:

$$\min f(x),$$

subject to: $g_j(x) \& 0, \ j = 1 \ldots m,$

$x \in G,$

where $\&$ could be $\leq$, $\geq$ or $=$.

We call $f(x)$ the objective function, the vector function $g(x) = (g_1(x), \ldots, g_m(x))$ is called the functional constraint, the set $G$ is called the basic feasible set, and the set

$$Q = \{x \in G \mid g_j(x) \& 0, \ j = 1 \ldots m\},$$

is called the feasible set of the problem (1.1.1).\footnote{That is just a convention to consider the minimization problems. Instead, we could consider a maximization problem with the objective function $-f(x)$.}
• **Constrained problems**: $Q \subset \mathbb{R}^n$.

• **Unconstrained problems**: $Q \equiv \mathbb{R}^n$.

• **Smooth problems**: all $g_j(x)$ are differentiable.

• **Nonsmooth problems**: there is a nondifferentiable component $g_k(x)$.

• **Linearly constrained problems**: all functional constraints are linear:

$$g_j(x) = \sum_{i=1}^{n} a_{i,j} x_i + b_j \equiv \langle a_j, x \rangle + b_j, \; j = 1 \ldots m,$$

(here $\langle \cdot, \cdot \rangle$ stands for the *inner product* in $\mathbb{R}^n$), and $G$ is a polyhedron.

If $f(x)$ is also linear then (1.1.1) is a *Linear Programming Problem*. If $f(x)$ is quadratic then (1.1.1) is a *Quadratic Programming Problem*.

There is also some classification in accordance to the properties of the feasible set.

• Problem (1.1.1) is called *feasible* if $Q \neq \emptyset$.

• Problem (1.1.1) is called *strictly feasible* if $\exists x \in \text{int} \; Q$ such that $g_j(x) < 0$ (or $> 0$) for all inequality constraints and $g_j(x) = 0$ for all equality constraints.

Finally, we distinguish different types of solutions to (1.1.1):

• $x^*$ is called the optimal *global solution* to (1.1.1) if

$$f(x^*) \leq f(x) \; \text{for all} \; x \in Q$$

(*global minimum*). Then $f(x^*)$ is called the *optimal value* of the problem.

• $x^*$ is called a *local solution* to (1.1.1) if

$$f(x^*) \leq f(x) \; \text{for all} \; x \in \text{int} \; \bar{Q} \subset Q$$

(*local minimum*).

Let us consider now several examples demonstrating the origin of the optimization problems.

**Example 1.1.1** Let $x^{(1)}, \ldots, x^{(n)}$ be our *design* or *decision variables*. Then we can fix some functional *characteristics* of our decision: $f(x), \; g_1(x), \ldots, g_m(x)$. That could be the price of the project, the amount of the required resources, the reliability of the system, and many others.

We fix the most important characteristics, $f(x)$, as our *objective*. For all others we impose some bounds: $a_j \leq g_j(x) \leq b_j$. 

1.1. GENERAL FORMULATION OF THE PROBLEM

Thus, we come up with the problem:

\[ \min f(x), \]
\[ \text{s.t.: } a_j \leq g_j(x) \leq b_j, \quad j = 1 \ldots m, \]
\[ x \in G, \]

where \( G \) stands for the structural constraints, like positiveness or boundedness of some variables, etc. ■

**Example 1.1.2** Let our initial problem be as follows: Find \( x \in \mathbb{R}^n \) such that

\[ g_1(x) = a_1, \]
\[ \ldots \]
\[ g_m(x) = a_m. \]  \hspace{1cm} (1.1.2)

Then we can consider the problem:

\[ \min_x \sum_{j=1}^{m} (g_j(x) - a_j)^2 \]

(may be with some additional constraints on \( x \)).

Note that the problem (1.1.2) is almost universal. It covers ordinary differential equations, partial differential equations, problems, arising in Game Theory, and many others. ■

**Example 1.1.3** Sometimes our decision variable \( x^{(i)} \ldots x^{(n)} \) must be integer, say, we need \( x^{(i)} \in \{0,1\} \). That can be described by the constraint:

\[ x^{(i)}(x^{(i)} - 1) = 0, \quad i = 1 \ldots n. \]

Thus, we could treat also the \( \{0,1\} \) Programming Problems:

\[ \min f(x), \]
\[ \text{s.t.: } a_j \leq g_j(x) \leq b_j, \quad j = 1 \ldots m, \]
\[ x \in G, \]
\[ x^{(i)}(x^{(i)} - 1) = 0, \quad i = 1 \ldots n. \] ■
Looking at these examples, a reader can understand the enthusiasm of the pioneers of nonlinear programming, which can be easily recognized in the papers of 1950 – 1960. Thus, our first impression should be as follows:

*Nonlinear Optimization is very important and perspective application theory. It covers almost all fields of Numerical Analysis.*

However, just by looking at the same list, especially at Examples 1.1.2, 1.1.3, a more suspicious (or more experienced) reader should come to the following conjecture:

*In general, optimization problems are unsolvable.*

Indeed, the life is too complicated to believe in a universal tool for solving all problems at once.

However, conjectures are not so important in science; that is a question of the personal taste how much we can believe in them. The most important event in the optimization theory in the middle of 70s was that this conjecture was proved in some strict sense. The proof is so simple and remarkable, that we cannot avoid it in our course. But first of all, we should introduce some special language, which is necessary to speak about such serious things.

### 1.1.2 Performance of Numerical Methods

Let us consider some sample situation: We have a problem $\mathcal{P}$, which we are going to solve using a method $\mathcal{M}$. What is the performance of $\mathcal{M}$ on $\mathcal{P}$? Let us start with the general definition:

*Performance of $\mathcal{M}$ on $\mathcal{P}$ is the total amount of computational efforts, which is required by method $\mathcal{M}$ to solve the problem $\mathcal{P}$.*

In this definition there are several things to be specified. First, what does it mean: to solve the problem? In some fields it could mean to find the exact solution. However, in many areas of numerical analysis that is impossible (and optimization is definitely the case). Therefore, for us to solve the problem should mean:

*To find an approximate solution to $\mathcal{P}$ with a small accuracy $\epsilon > 0$.*

Now, we know that there are different numerical methods for doing that, and of course, we want to choose the scheme, which is the best for our $\mathcal{P}$. However, it appears that we are looking for something, what does not exist. In fact, it does, but it is too silly. Just imagine a method for solving (1.1.1), which always reports that $x^* = 0$. Of course, it does not work on all problems except those with $x^* = 0$. And for the latter problems its “performance” is better than that of all other schemes.

Thus, we cannot speak about the best method for a concrete problem $\mathcal{P}$, but we can do that for a class of problems $\mathcal{F} \supset \mathcal{P}$. Indeed, usually the numerical methods are developed
for solving many different problems with the similar characteristics. Therefore we can define the performance of $M$ on $F$ as its performance on the worst problem from $F$.

Since we are going to speak about the performance of $M$ on the whole class $F$, we should assume that $M$ does not have a complete information about a concrete problem $P$. It has only the description of the problem class $F$. In order to recognize $P$ (and solve it), the method should be able to collect the personal information about $P$ by parts. For modeling this situation, it is convenient to introduce the notion of oracle. Oracle $O$ is just a unit, which answers the successive questions of the method. The method $M$, collecting and handling the data, is trying to solve the problem $P$.

In general, each problem can be included in different problem classes. For each problem we can imagine also the different types of oracles. But if we fix $F$ and $O$, then we fix a model of our problem $P$. In this case, it is natural to define the performance of $M$ on $(F, O)$ as its performance on the worst $P_w$ from $F$.

Let us now consider the iterative process which naturally describes any method $M$ working with the oracle.

**General Iterative Scheme.**

**Input:**

A starting point $x_0$ and an accuracy $\epsilon > 0$.

**Initialization.**

Set $k = 0$, $I_{-1} = \emptyset$. Here $k$ is the iteration counter and $I_k$ is the informational set accumulated after $k$ iterations.

**Main Loop.**

1. Call the oracle $O$ at $x_k$.
2. Update the informational set: $I_k = I_{k-1} \cup (x_k, O(x_k))$.
3. Apply the rules of method $M$ to $I_k$ and form the new test point $x_{k+1}$.
4. Check the stopping criterion. If yes then form an output $\bar{x}$. Otherwise set $k = k + 1$ and go to 1.

**End of the Loop.**

Now we can specify the term computational efforts in our definition of the performance. In the scheme (1.1.3) we can easily find two main sources of that. First one is in the Step 1, where we call the oracle, and the second one is in Step 3, where we form the next test point. We introduce two measures of the complexity of the problem $P$ for the method $M$:

1. **Analytical complexity:** The number of calls of the oracle, which is required to solve the problem $P$ up to the accuracy $\epsilon$.
2. **Arithmetical complexity:** The total number of the arithmetic operations (including the work of the oracle and the method), which is required to solve the problem $P$ up to the accuracy $\epsilon$.

Note that this $P_w$ can be bad only for $M$.
Thus, the only thing which is not clear now, is the meaning of the words *up to the accuracy* $\epsilon > 0$. Note, that this meaning is very important for our definitions of the complexity. However, it is too specific to speak about that here. We will make this meaning exact when we will consider the concrete problem classes.

Comparing the notions of analytical and arithmetical complexity, we can see that the second one is more realistic. However, for a concrete method $\mathcal{M}$, the arithmetical complexity usually can be easily obtained from the analytical complexity. Therefore, in this course we will speak mainly about some estimates of the analytical complexity of some problem classes.

There is one standard assumption about the oracle, which allows to obtain most of the results on the analytical complexity of the optimization methods. This assumption is called the *black box concept* and it looks as follows:

1. The only information available from the oracle is its answer. No intermediate results are available.
2. The oracle is local: A small variation of the problem far enough from the test point $x$ does not change the answer at $x$.

This concept is extremely popular in the numerical analysis. Of course, it looks as an artificial wall between the method and the oracle created by ourselves. It seems natural to allow the method to analyze the internal structure of the oracle. However, we will see that for some problems with complicated structure this analysis is almost useless. On the other hand, for some important problems it could help. If we have enough time, that will the subject of the last lecture of this course.

To conclude this section, let us present the main types of the oracles used in optimization. For all of them the input is a test point $x \in \mathbb{R}^n$, but the output is different:

- **Zero-order oracle**: the value $f(x)$.
- **First-order oracle**: the value $f(x)$ and the gradient $f'(x)$.
- **Second-order oracle**: the value $f(x)$, the gradient $f'(x)$ and the Hessian $f''(x)$.

### 1.2 Complexity bounds for Global Optimization

Let us practice in applying the formal language, introduced in the previous section, to a concrete problem. For that, let us consider

$$ \min_{x \in B_n} f(x). \quad (1.2.4) $$

In our terminology, that is an unconstrained minimization problem without functional constraints. The basic feasible set of this problem is $B_n$, an $n$-dimensional cube in $\mathbb{R}^n$:

$$ B_n = \{ x \in \mathbb{R}^n \mid 0 \leq x_i \leq 1, \ i = 1, \ldots, n \}. $$

In order to specify the problem class, let us make the following assumption:
The objective function \( f(x) \) is Lipshitz continuous on \( B_n \):

\[
\forall x, y \in B_n : \quad | f(x) - f(y) | \leq L \| x - y \|
\]

with some constant \( L \) (Lipshitz constant).

Here and in the sequel we use notation \( \| \cdot \| \) for the Euclidean norm on \( \mathbb{R}^n \):

\[
\| x \| = \langle x, x \rangle = \sqrt{\sum_{i=1}^{n} (x_i)^2}.
\]

Let us consider a trivial method for solving (1.2.4), which is called the Uniform Grid Method. This method, \( G(p) \), has one integer input parameter \( p \) and its scheme is as follows.

**Scheme of the method \( G(p) \).**

1. Form \((p + 1)^n\) points

\[
x_{(i_1, i_2, \ldots, i_n)} = \left( \frac{i_1}{p}, \frac{i_2}{p}, \ldots, \frac{i_n}{p} \right),
\]

where

\[
i_1 = 0, \ldots, p,
\]

\[
i_2 = 0, \ldots, p,
\]

\[\ldots\]

\[
i_n = 0, \ldots, p.
\]

2. Among all points \( x_{(\cdot)} \) find the point \( \bar{x} \) with the minimal value of the objective function.
3. Return the pair \( (\bar{x}, f(\bar{x})) \) as the result. \( \blacksquare \)

Thus, this method forms a uniform grid of the test points inside the cube \( B_n \), computes the minimal value of the objective over this grid and returns it as an approximate solution to the problem (1.2.4). In our terminology, this is a zero-order iterative method without any influence of the accumulated information on the sequence of test points. Let us find its efficiency estimate.

**Theorem 1.2.1** Let \( f^* \) be the global optimal value of problem (1.2.4). Then

\[
f(\bar{x}) - f^* \leq L \frac{\sqrt{n}}{2p}.
\]

**Proof:**

Let \( x^* \) be the global minimum of our problem. Then there exists a "number" \( (i_1, i_2, \ldots, i_n) \) such that

\[
x = x_{(i_1, i_2, \ldots, i_n)} \leq x^* \leq x_{(i_1+1, i_2+1, \ldots, i_n+1)} = y
\]
LECTURE 1. INTRODUCTION

(here and in the sequel we write \( x \leq y \) for \( x, y \in \mathbb{R}^n \) if and only if \( x_i \leq y_i, \ i = 1, \ldots, n \).

Note that \( y_i - x_i = 1/p, \ i = 1, \ldots, n \), and \( x_i^* \in [x_i, y_i], \ i = 1, \ldots, n \).

Denote \( \tilde{x} = (x + y)/2 \). Let us form a point \( \tilde{x} \) as follows:

\[
\tilde{x}_i = \begin{cases} 
y_i, & \text{if } x_i^* \geq \tilde{x}_i, \\
x_i, & \text{otherwise.}
\end{cases}
\]

It is clear that \(| \tilde{x}_i - x_i^* | \leq \frac{1}{2p}, \ i = 1, \ldots, n \). Therefore

\[
\| \tilde{x} - x^* \|^2 = \sum_{i=1}^{n} (\tilde{x}_i - x_i^*)^2 \leq \frac{n}{4p^2}.
\]

Since \( \tilde{x} \) belongs to our grid, we conclude that

\[
f(\tilde{x}) - f(x^*) \leq f(\tilde{x}) - f(x^*) \leq L \| \tilde{x} - x^* \| \leq L \frac{\sqrt{n}}{2p}.
\]

Note that now we still cannot say what is the complexity of this method on the problem (1.2.4). And the reason is that we did not define what should be the quality of the approximate solution we are looking for. Let us define our goal as follows:

\[
\text{Find } \bar{x} \in B_n : f(\bar{x}) - f^* \leq \epsilon.
\]

Then we immediately get the following result.

**Corollary 1.2.1** The analytical complexity of the method \( G \) is as follows:

\[
\mathcal{A}(G) = \left( \left\lfloor L \frac{\sqrt{n}}{2\epsilon} \right\rfloor + 1 \right)^n
\]

(here \( \lfloor a \rfloor \) is the integer part of \( a \)).

**Proof:**

Indeed, let us take \( p = \left\lceil L \frac{\sqrt{n}}{2\epsilon} \right\rceil + 1 \). Then \( p \geq L \frac{\sqrt{n}}{2\epsilon} \), and therefore, in view of Theorem 1.2.1, we have:

\[
f(\tilde{x}) - f^* \leq L \frac{\sqrt{n}}{2p} \leq \epsilon.
\]

This result is more informative, but we still have some questions. First, may be our proof is too rough and the real performance of \( G(p) \) is much better. Second, we cannot be sure that this is a reasonable method for solving (1.2.4). May be there are some methods with much higher performance.

In order to answer these questions, we need to derive for (1.2.4), (1.2.6) the lower complexity bounds. The main features of these bounds are as follows.
• They are based on the *Black Box* concept.
• They can be derived for a specific class of problems \( \mathcal{F} \) equipped by a local oracle \( \mathcal{O} \).
• These bounds are valid for all reasonable iterative schemes. Thus, they provide us with a lower bound for the *analytical complexity* on the problem class.
• They use the idea of *resisting oracle*.

For us only the notion of the resisting oracle is new. Therefore, let us discuss it in details. A resisting oracle is trying to create a *worst* problem for each concrete method. It starts from an ”empty” function and it tries to answer each call of the method in the worst possible way. However, the answers must be *coherent* with the previous answers and with the description of the problem class. Note that after the termination of the method it is possible to *reconstruct* the created problem. Moreover, if we launch the method on this problem, it will reproduce the same sequence of the test points since it will have the same answers of the oracle.

Let us show how it works for the problem (1.2.4). Consider the class of problems \( \mathcal{F} \) defined as follows:

**Problem Formulation:** \( \min_{x \in B_n} f(x) \).

**Problem Class:** \( f(x) \) is *Lipschitz continuous* on \( B_n \).

**Approximate solution:** Find \( \bar{x} \in B_n : f(\bar{x}) - f^* \leq \epsilon \).

**Theorem 1.2.2** *The analytical complexity of \( \mathcal{F} \) for the 0-order methods is at least \( \left( \left\lfloor \frac{L}{2\epsilon} \right\rfloor \right)^n - 1 \).*

**Proof:**
Assume that there exists a method, which needs less than

\[
p^n - 1, \quad p = \left\lfloor \frac{L}{2\epsilon} \right\rfloor \quad (\geq 1),
\]

calls of oracle to solve any problem of our class up to accuracy \( \epsilon > 0 \). Let us suppose that when the method finds its approximate solution \( \bar{x} \) we allow it to call the oracle one more time at \( \bar{x} \), which will not be counted in our complexity evaluation. So, the total number of calls to the oracle of the method is \( N < p^n \).

Let us apply this method to the following resisting oracle:

*It reports that \( f(x) = 0 \) at any test point.*

Therefore this method can find only \( \bar{x} \in B_n : f(\bar{x}) = 0 \).

Note that there exists \( \hat{x} \in B_n \) such that

\[
\hat{x} + \frac{1}{p} e \in B_n, \quad e = (1, \ldots, 1),
\]

and there were no test points inside the box

\[
B = \{ x \mid \hat{x} \leq x \leq \hat{x} + \frac{1}{p} e \}.
\]
Denote \( \tilde{x} = \hat{x} + \frac{1}{2p}\epsilon \) and consider the function
\[
\tilde{f}(x) = \min\{0, L \| x - \tilde{x} \|_\infty - \epsilon\},
\]
where \( \| a \|_\infty = \max_{1 \leq i \leq n} |a_i| \).

Note that the function \( \tilde{f}(x) \) is Lipshitz continuous (since \( \| a \|_\infty \leq \| a \| \)) and the optimal value of \( \tilde{f}(\cdot) \) is \( -\epsilon \). Moreover, \( \tilde{f}(x) \) differs from zero only inside the box
\[
B' = \{ x \| x - \tilde{x} \|_\infty \leq \frac{\epsilon}{L} \}.
\]

Since \( 2p \leq L/\epsilon \), we conclude that
\[
B' \subseteq B \equiv \{ x \| x - \tilde{x} \|_\infty \leq \frac{1}{2p} \}.
\]

Thus, \( \tilde{f}(x) \) is equal to zero at all test points of our method. Since the accuracy of the result of our method is \( \epsilon \), we come to the following conclusion: If the number of calls of the oracle is less than \( p^n \) then the accuracy of the result cannot be less than \( \epsilon \).

Now we can say much more about the performance of the uniform grid method. Let us compare its efficiency estimate with the lower bound:
\[
G : \left( L \sqrt{\frac{n}{2\epsilon}} \right)^n, \quad \text{Lower bound:} \left( \frac{L}{2\epsilon} \right)^n.
\]
Thus, we conclude that \( G \) has optimal dependence of its complexity in \( \epsilon \), but not in \( n \). Note that our conclusion depends on the problem class. If we consider the functions \( f \):
\[
\forall x, y \in B_n : \quad | f(x) - f(y) | \leq L \| x - y \|_\infty
\]
then the same reasoning as before proves that the uniform grid method is optimal with the efficiency estimate \( \left( \frac{L}{2\epsilon} \right)^n \).

Theorem 1.2.2 supports our initial claim that the general optimization problems are unsolvable. Let us look at the following example.

**Example 1.2.1** Consider the problem class \( F \) defined by the following parameters:
\[
L = 2, \quad n = 10, \quad \epsilon = 0.01.
\]
Note that the size of the problem is very small and we ask only for 1% accuracy.

The lower complexity bound for this class is \( \left( \frac{L}{2\epsilon} \right)^n \). Let us compute what does it mean:

- Lower bound: \( 10^{20} \) calls of oracle,
- Complexity of the oracle: \( n \) a.o.,
- Total complexity: \( 10^{21} \) a.o.,
- Intel Quad Core Processor: \( 10^9 \) a.o. per second,
- Total time: \( 10^{12} \) seconds,
- 1 year: less than \( 3.2 \cdot 10^7 \) sec.

**We need:** \( 32 \,000 \) years.
This estimate is so disappointing that we cannot believe that such problems may become solvable even in the future. Indeed, suppose we believe into the Moore low, i.e. that the processor power is to be multiplied by 3 every 2 years. We can hope that a PC of 2030 will solve the problem in only 1 year, and in 2070 it will only take 1 second. However, let us just play with the parameters of the class.

- If we change $n$ for $n + 1$ then we have to multiply our estimate by 100. Thus, for $n = 11$ our time estimate is valid for the fastest available computer.

- But if we multiply $\epsilon$ by two, we reduce the complexity by the factor of 1000. For example, if $\epsilon = 8\%$ then we need only 2 days to solve the problem.

We should note, that the lower complexity bounds for problems with smooth functions, or for the high-order methods is not much better than that of Theorem 1.2.2. This can be proved using the same arguments and we leave the proof as an exercise for the reader. An advanced reader can compare our results with the upper bound for NP-hard problems, which are considered as the examples of very difficult problems in combinatorial optimization. It is $2^n \text{ a.o. only!}$

To conclude this section, let us compare our situation with some other fields of numerical analysis. It is well-known, that the uniform grid approach is a standard tool for many of them. For example, if we need to compute numerically the value of the integral

$$\mathcal{I} = \int_0^1 f(x)dx,$$

then we have to form the discrete sum

$$S_n = \frac{1}{N} \sum_{i=1}^{n} f(x_i), \quad x_i = \frac{i}{N}, \quad i = 1, \ldots, N.$$

If $f(x)$ is Lipshitz continuous then the result can be used as an approximation to $I$:

$$N = L/\epsilon \quad \Rightarrow \quad |\mathcal{I} - S_N| \leq \epsilon.$$

Note that in our terminology it is exactly the uniform grid approach. Moreover, it is a standard way for approximating the integrals. The reason why it works here is in the dimension of the problem. For integration the standard sizes are 1 – 3, and in optimization sometimes we need to solve problems with several million variables.

1.3 Rules of the game

Some of the statements in the Course (theorems, propositions, lemmas, examples (if the latter contain certain statement) are marked by superscripts * or +. The unmarked statements
are **obligatory**: you are required to know both the statement and its proof. The statements marked by * are **semi-obligatory**: you are expected to know the statement itself and may skip its proof (the latter normally accompanies the statement), although you are welcome, of course, to read the proof as well. The proofs of the statements marked by + are omitted; you are expected to be able to prove these statements by yourself, and these statements are parts of the assignments.

The majority of Lectures are accompanied by the ”Exercise” sections. In several cases, the exercises are devoted to the lecture where they are placed; sometimes they prepare the reader to the next lecture. Exercises marked by # are closely related to the lecture where they are placed or to the following one; it would be a good thing to solve such an exercise or at least to become acquainted with its solution (if any is given). Exercises which I find difficult are marked with $>$.

### 1.4 Suggested reading

It could be helpful to have a good book on analysis at hand.

If you want to improve your background on the basic mathematical notions involved, consider the reference


The main drawback of the “little blue book” by C. Lemarechal: *Méthodes numériques d’optimisation*, Notes de cours, Université Paris IX-Dauphine, INRIA, Rocquencourt, 1989, is that it is too small.

As far as the main body of the course is concerned, for Chapter 6 I would suggest the reference


- P.J. Laurent: *Approximation et Optimisation*, Hermann (1972)

All these books also possesses that important quality of being written in French. If you decide that you are interested in Convex Optimization, the following reading would is extremely gratifying


The latter book is available on the S. Boyd’s page: [http://www.stanford.edu/~boyd/index.html](http://www.stanford.edu/~boyd/index.html)