CENTRAL EUROPEAN UNIVERSITY

Department of Mathematics and its Applications

Master of Science Thesis

in Applied Mathematics

Siarhei Charnyi

Solving maximin problem by decomposition to smooth and nonsmooth problems

Scientific Adviser

Prof. Istvan Maros

University of Pannonia

CONTENTS

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor Professor Istvan Maros for fruitful and helpful discussions on the topic. I would like to thank the department of Mathematics for the financial support for traveling expenses. I had also an opportunity to order a number of books on my topic via the department of Mathematics, which made my work on the thesis more effective. I am also thankful to Renyi Institute of Mathematics, and to Professor Ervin Győri in particular, who made us possible to use the electronic subscriptions together with the excellent library of the Institute. It allowed me to collect a number of books and over 200 articles on my topic.

NOTATIONS

c, k	lowercase italic letters: scalar variables
$\mathbf{v}$	lowercase boldface letters: vectors
$\mathbf{A}$	uppercase letters: matrices
$\mathbb{R}^n$	n-dimensional Euclidean space
$\langle \mathbf{a}, \mathbf{b} \rangle$	scalar product of vectors $\mathbf{a}$ and $\mathbf{b}$

INTRODUCTION

Study of different conflict situations within operations research and game theory resulted in various optimization problems, which are much more complex than traditional problems of mathematical programming.

Among optimization problems, the so-called minimax (or maximin) problems occupy an important place. Mathematical modeling of conflict situations and study of problems under uncertainty conditions using the principle of guaranteed result lead to minimax problems [4]. Among a wide range of minimax problems, for example, one can distinguish bilinear minimax problems with linear constraints. The necessity of studying this class of problems is justified by its wide range of applications and practical importance [1]. As it is mentioned in [1], bilinear minimax problems often appear in matrix game theory, within development of numerical methods for nonlinear minimax problems, etc.

One of the first maximin problems was described by P.L. Chebyshev. It is the problem of the best uniform approximation of a function by polynomials, and it was intensively studied by a number of authors [4]. This problem is a special case of a maximin problem with independent variables, namely:

find

$$\sup_{\mathbf{x}\in X}\inf_{\mathbf{y}\in Y} f(\mathbf{x},\mathbf{y}). \tag{0.1}$$

A more general case is represented by the minimax problem with coupled constraints:

$$\sup_{\mathbf{x}\in X}\inf_{\mathbf{y}\in F(\mathbf{x})} f(\mathbf{x},\mathbf{y}) \tag{0.2}$$

According to V.F. Demyanov, the above problem is the main one in the game theory of two players with an information transfer [4]. Let the first player have a payoff function $f(\mathbf{x},\mathbf{y})$ and the admissible set of strategies X. The second player has a payoff function $g(\mathbf{x},\mathbf{y})$ and the admissible set of strategies Y. The first player informs the second player about his decision, i.e. the strategy $\mathbf{x}\in X$. Then the formula (0.2) gives the best guaranteed result of the first player, where $F(\mathbf{x})$ is the mapping:

$$F(\mathbf{x}) = \left\{\mathbf{y}\in Y \mid g(\mathbf{x},\mathbf{y}) = \max_{\mathbf{z}\in Y} g(\mathbf{x},\mathbf{z})\right\}.$$

Any minimax problem can be considered as a result of the application of the principle of the best guaranteed result (or other principle of optimality) for a game with a particular strategy

of players [4]. As there are new systems and strategies of interaction of players being developed constantly, new minimax problems will appear.

Berc Rustem in [10] compares the minimax approach with the approach based on expectation maximization to decisions under uncertainty. He mentions that traditionally decisions under uncertainty are made by optimizing the expected performance of the system approach. However, he recognizes its limitations in that such an approach does not take into account the worst-case situation. The decisions based on expected value optimization are acceptable in many cases, but they need to be estimated for a case of the worst-case scenario [10]. As he shows, this is especially important if the worst case scenario may lead to a failure of the system. But at the same time, he acknowledges, the probability of the worst-case scenario might be so negligible that decisions based on it may result in an unnecessarily pessimistic decision. The minimax strategy provides an estimation which gives an opportunity to compare the performance of the worst-case with the expected value performance [10]. The performance of the minimax decision does not decrease for any scenario. Therefore the minimax design represents a robust strategy.

Following this line of thought, Rustem shows that minimax design is sometimes preferable to expected value optimization approach for two reasons. First, the minimax strategy provides the tool for analysis of the risk of the worst-case, which the expected value optimization approach does not take into account. Second, the worst-case scenario may cause a failure of the system, even when the expected value optimization appears to be the most suitable approach. Therefore every decision should be based both on minimax strategy and on the expected performance approach [10].

However Rustem highlights that the minimax strategy is a pessimistic one. Therefore it may result in a decision with an unnecessary low performance. At the same time, he point out that in the worst-case scenario the decision based on expected value optimization may lead to an unacceptable decrease of performance. Therefore, the author argues, neither minimax nor expected value optimization approaches can be taken for granted. Both of them can be used only for analyzing the risks of uncertain effects. In the presence of uncertainty, Rustem shows, the decision maker may desire to evaluate his optimal strategy for a worst-case realization of uncertain parameters. Usually this leads to minimax formulations [10].

In [10] Rustem gives a detailed overview of existing algorithms for a continuous maximin problem. He also shows there a number of applications of the minimax problem to optimal pricing of options, optimal portfolio, decisions under uncertainty, etc.

It is necessary to say that not enough algorithms have been developed for minimax (maximin) problems. Among different types of maximin problems the problem with two sets of

continuous variables occupies an important place. But existing algorithms can deal only with the continuous minimax problem with *independent constraints*. For example, Rustem and Howe's book "Algorithms for Worst-Case Design and Applications to Risk Management", which is devoted mainly to the algorithms for minimax/maximin problem and its applications, describes the algorithms that deal with minimax problems with *independent constraints* only. A number of practical applications, for example the game of 2 players with transfer of information, lead to the minimax/maximin problems with *coupled constraints*. Therefore it is important to develop an algorithm which can handle the minimax problem with coupled constraints as well. Within the thesis, I consider a continuous maximin problem with coupled constraints. The proposed method is based on decomposition of the original problem to two subsequently solved problems. One of the problems is a nonsmooth problem.

Nonsmooth optimization methods may be applied not only to problems with nonsmooth variables, but also in many other cases. An example is a nonlinear problem which is difficult to solve due to some properties, such as special structure of the problem, large dimension or large number of constraints. For example, for the nonlinear problem:

$$\begin{aligned} & f(\mathbf{x},\mathbf{y}) \to \min \\ & g_i(\mathbf{x},\mathbf{y}) \le 0, i = 1...r \\ & h_j(\mathbf{x},\mathbf{y}) = 0, j = 1...p \\ & \mathbf{x} \in X \subset \mathbb{R}^n \end{aligned}, \tag{0.3}$$

(where $X \subset \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m$) when dimension is high, a number of experts (A. Ben-Tal, J. Outrata, J. Zowe) consider it reasonable to solve the problem (0.3) by decomposition to two subsequently solved problems of low dimension. One of the problems is a nonsmooth problem. That is the original problem is replaced with

$$P(\mathbf{x}): \quad \Phi(\mathbf{x}) \to \min, \tag{0.4}$$

where

$$\Phi(\mathbf{x}) = \min_{\mathbf{y}} \{ f(\mathbf{x},\mathbf{y}), \mathbf{y} \in F(\mathbf{x}) \},$$

$$F(\mathbf{x}) = \begin{Bmatrix} \mathbf{y} \in \mathbb{R}^m \mid g_i(\mathbf{x},\mathbf{y}) \le 0, i = 1...r; \\ h_j(\mathbf{x},\mathbf{y}) = 0, \ j = 1...p \end{Bmatrix}$$

is a multivalued mapping.

It is suggested to use so called "Bundle methods" for the nonsmooth problem (0.4). Bundle methods are rather universal and are widely used. But one of the obvious shortcomings of applying bundle methods to the problem (0.4) is that they do not take into account the special properties of the problem.

Within the thesis I am considering the following continuous maximin problem with coupled constraints:

$$\max_{\mathbf{x}\in\mathbb{R}^n} \min_{\mathbf{y}\in F(\mathbf{x})} f(\mathbf{x},\mathbf{y}) \tag{0.5}$$

where

$$F(\mathbf{x}) = \left\{ \begin{matrix} \mathbf{y} \in \mathbb{R}^m \mid g_i(\mathbf{x},\mathbf{y}) \le 0, i = 1...r; \\ h_j(\mathbf{x},\mathbf{y}) = 0, j = 1...p \end{matrix} \right\}$$

is a multivalued mapping.

The goal of the present work is to develop and implement an algorithm for the case of objective function $f(\mathbf{x},\mathbf{y})$ linear in variable $\mathbf{y}$. The method is based on obtaining the steepest ascend direction of the minimum function $\Phi(\mathbf{x})$. It is also based on the theory of optimization of nonsmooth functions.

The proposed decomposition for problem (0.5) is promising due to the following:

1) it reduces the dimension of the original problem
2) it makes it possible to handle the maximin problem with coupled constraints
3) it allows the use of existing nonsmooth optimization methods.

The present thesis begins with an introductory chapter, followed by the 3 chapters, the conclusion, and the references and appendix section. Chapter 1 presents basic concepts of convex analysis and reviews existing methods of nonsmooth optimization. In chapter 2 I have developed a method of decomposition of the maximin problem with coupled constraints, an algorithm for the problem, and I have studied the arising problems and methods of their solution. In chapter 2 I have also obtained and proved the necessary condition of local maximum of the function $\Phi(\mathbf{x})$. Chapter 3 contains a computational study of the developed algorithm and analysis of obtained results. The appendix contains the code listing of the developed program in the system of computer algebra Maple.

CHAPTER 1.
BASIC CONCEPTS AND REVIEW OF EXISTING METHODS OF NONSMOOTH OPTIMIZATION

In the present chapter the necessary information from convex and nonsmooth analysis is given (see [9]). For simplicity the space X is assumed to be finite-dimensional. We assume that $X = \mathbb{R}^n$. Let $\langle \mathbf{x}^*, \mathbf{x} \rangle$ denotes the scalar product of two vectors $\mathbf{x}^*, \mathbf{x} \in X, |\mathbf{x}|$ is Euclidean norm of vector **x**. Let B be an open unit ball with the centre at 0, i.e. $B = \{\mathbf{x} \in X, |\mathbf{x}| < 1\}$.

1.1. The basic concepts of convex analysis

Convex sets.

Def A set $C \subset X$ is called *convex,* if together with any two points $x_1, x_2 \in C$ it contains a segment connecting them, i.e. $\lambda x_1 + (1 - \lambda) x_2 \in C$ for any $\lambda \in [0,1]$. Empty set $\varnothing$ is assumed convex by definition. If C is a convex set then its closure $\mathrm{cl}\, C$, an interior $\mathrm{int}\, C$ and the set

$$\lambda C = \{\lambda x \mid x \in C\}, \lambda \in \mathbb{R}$$

are also convex [9].

If sets C_1 and C_2 are convex, then their intersection $C_1 \cap C_2$ and their algebraic sum

$$C_1 + C_2 = \{x_1 + x_2 \mid x_1 \in C_1, x_2 \in C_2\}$$

are also convex [9].

Def Let $M \subset X$. The intersection of all convex (convex closed) sets in *X*, containing *M*, is called a *convex hull (convex closure)* of set *M* and is denoted by co*M* ($\overline{\mathrm{co}}M$).

Notice that co*M* and $\overline{\mathrm{co}}M$ are convex sets. Besides, $\mathrm{co}\, M \subset \overline{\mathrm{co}}M$ and hence $\mathrm{cl}\,\mathrm{co}\, M \subset \overline{\mathrm{co}}M$. Inversely, $\overline{\mathrm{co}}M \subset \mathrm{cl}\,\mathrm{co}\, M$. Therefore $\overline{\mathrm{co}}M = \mathrm{cl}\,\mathrm{co}M$ [9].

Def Linear combination $\lambda_1 \mathbf{x}_1 + \lambda_2 \mathbf{x}_2 + \ldots + \lambda_m \mathbf{x}_m$ is called a *convex combination* of points $\mathbf{x}_1, \ldots, \mathbf{x}_m$, if $\lambda_1 + \lambda_2 + \ldots + \lambda_m = 1, \lambda_i \geq 0, i = \overline{1,m}.$

Theorem 1.1. [9] In the space $X = \mathbb{R}^n$, any point from $\mathrm{co}\, M$ can be presented as a convex combination of at most $n+1$ points from M.

It directly follows from the theorem 1.1 that the set *M* is convex if and only if $\mathrm{co}\, M = M$ [9].

The important role in convex analysis and applications is played by the following

The separability theorem. [9] Let C_1 and C_2 are nonempty closed convex sets and one of them is bounded. Then, if $C_1 \cap C_2 \neq \varnothing$, then there exists a vector $\mathbf{x}^*$ and a number $\varepsilon > 0$ such that $\langle \mathbf{x}^*, \mathbf{x}_1 \rangle \leq \langle \mathbf{x}^*, \mathbf{x}_2 \rangle - \varepsilon$ for all $\mathbf{x}_1 \in C_1, \mathbf{x}_2 \in C_2$.

Def A set $K \subset X$ is called a *cone*, if from $x \in K$ it follows that $\lambda x \in K$ for all $\lambda > 0$.
The cone *K* is convex if and only if from $x_1, x_2 \in K$ it follows that $x_1 + x_2 \in K$ [9].

Def Let *C* be a nonempty convex set in *X*. The set

$$0^+ C \triangleq \left\{ \bar{x} \in X \mid x + \lambda \bar{x} \in C, \forall \lambda > 0 \right\}.$$

is called a *recession cone* of set *C*.

As $0 \in 0^+ C$ therefore $0^+ C \neq \varnothing$. The recession cone $0^+ C$ is a convex cone [9] and

$$0^+ C = \left\{ \bar{x} \in X \mid C + \bar{x} \subset C \right\}.$$

A special role among convex sets is played by so-called polyhedral sets.

Def The set of points $\mathbf{x} \in C$ is called *polyhedral*, if it satisfies the system of linear inequalities

$$\langle \mathbf{x}_i^*, \mathbf{x} \rangle \leq \alpha_i, i = \overline{1,m}, \tag{1.1}$$

where $\alpha_i \in \mathbb{R}, \mathbf{x}_i^* \in X, i = \overline{1,m}$ are fixed.

A special case of a polyhedral set is a convex polyhedron, i.e. a bounded set described by the system (1.1). A convex polyhedron is a convex hull of a finite number of points.

Def A cone *K* is called polyhedral, if there is a finite set of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$, such that

$$K = \left\{ \mathbf{x} \in X \mid \mathbf{x} = \sum_{i=1}^{m} \lambda_i \mathbf{x}_i, \lambda_1 \geq 0, i = \overline{1,m} \right\}.$$

It is known that a polyhedral cone can be always described by a finite system of linear homogeneous inequalities

$$\langle \mathbf{x}_i^*, \mathbf{x} \rangle \leq 0, i = \overline{1,m}. \tag{1.2}$$

Inversely, the set of solutions of the system (1.2) is a polyhedral cone.

Among the properties of polyhedral cones it is necessary to mention the following: a polyhedral cone is always closed, the sum and intersection of polyhedral cones are again polyhedral cones [9].

Convex functions.

Def Let's consider a function $f : X \to \mathbb{R} \cup \{\pm\infty\}$. One may connect the following sets with the function *f*:

$$\operatorname{dom} f = \{x \in X \mid f(x) < +\infty\},$$
$$\operatorname{epi} f = \{(x,\lambda) \in X \times \mathbb{R} \mid f(x) \le \lambda\}.$$

They are called an *effective set* and a *function supergraph* of *f* accordingly. The supergraph completely defines function *f* [9], i.e.

$$f(x) = \inf_{\lambda} \{\lambda \mid (x,\lambda) \in \operatorname{epi} f\},$$

Def Function *f* is called *convex* if the set $\operatorname{epi} f$ is convex in the space $X \times \mathbb{R}$.

Def If $\operatorname{dom} f \neq \varnothing$ and $f(x) > -\infty$ for all *x,* then the function of *f* is called a *proper function*.

A proper function *f* is convex if and only if $f(\lambda x_1 + (1-\lambda)x_2) \le \lambda f(x_1) + (1-\lambda) f(x_2)$ for all $x_1, x_2 \in X, \lambda \in [0,1]$ [9].

The convexity of a function (not necessarily proper function) is equivalent to the following inequality [9]

$$f(\lambda x_1 + (1-\lambda)x_2) \le \lambda f(x_1) + (1-\lambda) f(x_2)$$

for all $x_1, x_2 \in X, \lambda \in [0,1]$.

Examples of convex functions are:

1) affine function

$$f(x) = \langle x^*, x \rangle + \alpha, \text{ where } x^* \in X, \alpha \in \mathbb{R};$$

2) support function of a convex set $C \subset X$

$$S_c(x^*) = \sup\{\langle x^*, x \rangle \mid x \in C\}.$$

Let's consider some operations over convex functions.

The sum $f + g$ of proper convex functions of *f* and *g* is a convex function [9].

Def Function *f* is called *closed* if the set $\operatorname{epi} f$ is closed in $X \times \mathbb{R}$.

The function *f* is closed if and only if one the following holds [9]:

1) *f* is lower semicontinuos

2) level sets $\{x \mid f(x) \le \alpha\}$ are closed in *X* for all $\alpha \in \mathbb{R}$.

Def Every function *f* can be assigned a function $\operatorname{cl} f = \inf\{\lambda \mid (x,\lambda) \in \operatorname{epi} f\}$ called *closure of function f*. As the closure of a convex set is convex, therefore the closure of a convex function is always a convex function [9].

Def Function $f(x) = \sup\{f_i(x) \mid i \in I\}$ is called *the least upper bound of a set of functions* $f_i(x), i \in I$.

Def Function *f* is called *positive homogeneous*, if $f(\lambda x) = \lambda f(x)$ for all $\lambda > 0, x \in X$. A function *f* is convex and positive homogeneous if and only if its supergraph epi *f* is a convex cone [9].

The following statement holds.

Theorem 1.2. [9] Let *f* be a proper convex function. Then the following are equivalent:

1. *f* is bounded from above in a neighbourhood of point *x*;
2. *f* is continuous in *x*;
3. *f* is Lipschitz in a neighbourhood of *x*, i.e. there exists a constant $l > 0$ such that $|f(x_1) - f(x_2)| \le l|x_1 - x_2|$ for all x_1, x_2 from a neighbourhood of point *x*.

Def A *directional derivative* of function *f* in direction $\bar{x} \in X$ at point *x* is

$$f'(x;\bar{x}) = \lim_{\varepsilon \downarrow 0} \varepsilon^{-1}\left[f(x+\varepsilon\bar{x}) - f(x)\right],$$

if the limit in the right hand side of the equality exists (finite or infinite). For a proper convex function *f* there exists a derivative $f'(x;\bar{x})$ in each point $x \in \operatorname{dom} f$ in any direction $\bar{x} \in X$. This derivative is a convex positive homogeneous function in variable $\bar{x}$.

Def A *subdifferential* of a convex function *f* at point $x_0 \in \operatorname{dom} f$ is the following set

$$\partial f(x_0) \triangleq \left\{x^* \in X \,\middle|\, f(x) - f(x_0) \ge \left\langle x^*, x - x_0\right\rangle \forall x \in X\right\}.$$

A subdifferential $\partial f(x)$ is a convex closed set in *X* [9]. Its support function is $f'(x;\bar{x})$, where the closure is taken with respect to $\bar{x}$. Thus $\partial f(x) = \partial_{\bar{x}} f'(x;0)$. If *f* is a convex function, continuous at point *x*, then $\partial f(x)$ is a nonempty compact set [9] and

$$f'(x;\bar{x}) = \max\left\{\left\langle x^*, \bar{x}\right\rangle \middle| x^* \in \partial f(x)\right\}.$$

In particular, if function *f* is convex and differentiable at point *x*, its subdifferential $\partial f(x)$ contains one element - a gradient $\nabla f(x)$ [9].

Def A function $f : X \to \mathbb{R} \cup \{\pm\infty\}$ is called *concave* if the function $-f$ is convex.

Def A function $f : X \times Y \to \mathbb{R} \cup \{\pm\infty\}$ is called *convex-concave* if function $x \mapsto f(x,y)$ is convex for all *y*, and function $y \mapsto f(x,y)$ is concave for all *x*.

The following statement takes place.

The theorem of minimax [9]. Let $X_0 \subset X$ and $Y_0 \subset Y$ be convex closed compact sets. Let function $f(\mathbf{x},\mathbf{y})$ be convex-concave function on the given sets. Then

$$\inf_{\mathbf{x}\in X_0}\sup_{\mathbf{y}\in Y_0} f(\mathbf{x},\mathbf{y}) = \sup_{\mathbf{y}\in Y_0}\inf_{\mathbf{x}\in X_0} f(\mathbf{x},\mathbf{y})$$

Also the following inequality holds

$$\inf_{\mathbf{x}\in X_0}\sup_{\mathbf{y}\in Y_0} f(\mathbf{x},\mathbf{y}) \geq \sup_{\mathbf{y}\in Y_0}\inf_{\mathbf{x}\in X_0} f(\mathbf{x},\mathbf{y})$$

for arbitrary sets X_0 and Y_0 and any function $f(\mathbf{x},\mathbf{y})$.

1.2. Multivalued mappings

In the present work we will need a concept of a multivalued mapping [9].

Def Let $X = \mathbb{R}^n$, $Y = \mathbb{R}^m$. A mapping $F : X \to 2^Y$ is called a *multivalued mapping* if for every $x \in X$ it assigns a set $F(x) \subset Y$.

The following sets are associated with a multivalued mapping F

$$\operatorname{gr} F = \{(x, y) | x \in X, y \in F(x)\}$$

and

$$\operatorname{dom} F = \{x \in X | F(x) \neq \varnothing\},$$

which are called the graph and effective set of mapping F respectively. It is obvious that

$$y \in F(x) \Leftrightarrow (x, y) \in \operatorname{gr} F,$$

i.e. a mapping F is completely determined by its graph.

The evaluation of different types of derivatives of minimum (maximum) functions defined on values of multivalued mappings, is an actual problem for quasidifferential calculus, the analysis of sensitivity of parametrical extremal problems, theory of necessary conditions of an extremum and other branches of optimization theory. In particular, it is important to define a class of multivalued mappings for which a minimum function is directionally differentiable [9].

1.3. The review of existing methods of nonsmooth optimization

One can solve a maximin problem by its decomposition to two optimization problems one of which is a nonsmooth problem. Therefore the review of existing methods of nonsmooth optimization is of interest.

The current state of optimization theory is characterized by intensive development of so-called nonsmooth analysis, i.e. a generalized differential calculus for nondifferentiable in the usual sense functions. The concept of a subdifferential for convex functions was the first generalization of a classical gradient of a function in the space $\mathbb{R}^n$. But unlike a gradient, a subdifferential of a convex function $f:\mathbb{R}^n \to \mathbb{R}$ is in general, a set $\partial f(x)$ in the space $\mathbb{R}^n$. Its element $\xi \in \partial f(x)$ is called a subgradient of a function *f* at point *x*. In this case the classical Fermat theorem for a minimum point *x* takes the form $0 \in \partial f(x)$. A point *x*, in which the last condition is satisfied, is called a stationary point. If a convex function *f* is differentiable in the usual sense, then $\partial f(x) = \nabla f(x)$ and Fermat's theorem is reduced to its usual form $\nabla f(x) = 0$.

The generalization of the notion of a subdifferential for non-convex functions has led to the notion of generalized gradient (F. Clark). A generalized gradient for a function *f* is also denoted by $\partial f(x)$, and in the case of a smooth function *f* it coincides with its gradient.

Another direction in the development of nonsmooth analysis is connected with the use of directional derivative and its generalizations (V.F. Demjanov [4]) and the development on its basis of subdifferential (or quasidifferential) calculus of nonsmooth functions.

The development of generalized differential calculus for nonsmooth and nondifferentiable in the usual sense functions makes it possible to construct methods of nonsmooth optimization.

1.3.1. Subgradient methods

Subgradient methods are among the first methods developed for optimization of nonsmooth convex functions.

At iteration x_k, these methods make a step of a given size along a negative subgradient:

$$x_{k+1} = x_k + \lambda_k d_k ,$$

where $d_k = -g_k / |g_k|$ and $g_k \in \partial f(x_k)$.

Due to its simple structure subgradient methods are still widely used. But they have a number of serious shortcomings. Subgradient methods do not guarantee a decrease of an objective

function at each iteration, there is no easily implementable stopping criterion, and the rate of convergence of these methods is slow (usually it is less then linear).

1.3.2. Bundle-methods

There is a wide range of algorithms called bundle-methods. But all of them have a number of common features. All of them collect a set of subgradients or approximated subgradients associated with points in which they were calculated. On its basis one can obtain an interior approximation of a subdifferential. Then these methods calculate an approximation of the quickest descent direction. Using the obtained direction, these methods calculate the next potential point, using a search along the direction, or a trust region method. If the obtained point provides a sufficient decrease of an objective function, then the algorithm moves to it (serious step). Otherwise it undertakes a so-called null step at which the current point remains the same, and the subgradient obtained at the current iteration is added to the bundle, thus improving the approximation of a subdifferential.

One of the essential differences of different bundle methods is the method which they use to update a bundle of subgradients. In general they use either a convex combination of all previous subgradients or choose only some of them (all subgradients that do not describe well the behaviour of a function at the current point are removed from the bundle).

The first bundle methods obtained a new candidate point, using the search along the obtained direction. Later modifications (H.Shramm, I.Tsove) use the concept of a trust region which has a number of advantages.

Al bundle methods have the following two main properties:

- At iteration x_k they use an information *bundle*, $(f(x_k), g_k)$ $(f(x_{k-1}), g_{k-1})$ …, collected prior to the current iteration to construct a model of a function *f*;
- if the model is insufficiently adequate due to the nonsmooth structure of the function, then the additional information on subgradients in a neighbourhood of x_k should be collected.

Using the collected information, an approximation of *f* at point x_k is developed by the method of cutting planes:

$$\max_{1 \le i \le k} \{g_i^T (x - x_i) + f(x_i)\}.$$

The last expression is a piecewise linear approximation of a convex function *f* from below, which coincides with *f* at every x_i. Thus we obtain:

$$f_{CP}(x_k;d) = \max_{1\le i\le k}\{g_i^T d + g_i^T(x_k - x_i) + f(x_i)\},$$

where $d = x - x_k$, $d \in \mathbb{R}^n$.

This model poorly describes a function far from a point x_k. Therefore one adds a stabilizing term $(1/2t_k)d^T d$ for $t_k > 0$ to f_{CP} while searching for a minimum of the model. If f_{CP} describes the behaviour of *f* in a neighbourhood of x_k well enough, then the value of d_k which minimizes the expression

$$f_{CP}(x_k;d) + \frac{1}{2t_k}d^T d,$$

is a descent direction for the function *f*. A search along the direction d_k gives a point x_{k+1} such that $f(x_{k+1}) < f(x_k)$. For a nonsmooth function f it can happen that f_{CP} is such a bad approximation of f, that d_k is not a descent direction for *f* (or a search in this direction gives only insignificant decrease of f). In this case the second part of the method should be applied. Clearly f_{CP} does not have the same behaviour as *f* on the ray $x_k + \lambda d_k, \lambda \ge 0$. To compensate this lack of the information, the method remains in the current point x_k and adds a subgradient from $\partial f(x_k + \lambda d_k)$ for a small $\lambda > 0$ to the model.

Thus, the algorithm of the bundle method has the following form:

Iteration $x_k \to x_{k+1}$:

4. Calculate $d_k = d(t_k) = \arg\min\{f_{CP}(x_k;d) + (\frac{1}{2t_k})d^T d \mid d \in R^n\}$.

5. Carry out a search in the direction $x_k + \lambda d_k$ for $\lambda \ge 0$.

If a search gives essential decrease of function *f*, then make a serious step: $x_{k+1} = x_k + \lambda_k d_k$, $\lambda_k \in \arg\min_{\lambda>0} f(x_k + \lambda d_k)$ and calculate $g_{k+1} \in \partial f(x_{k+1})$.

If a search leads to an insufficient decrease, then make a null step: $x_{k+1} = x_k$ and calculate $g_{k+1} \in \partial f(x_k + \lambda d_k)$ for a small enough $\lambda > 0$.

Unlike the subgradient method, the described above procedure guarantees a decrease of an objective function at each iteration. Furthermore, there is a stopping criterion: x is an optimum, if d_k is close enough to 0. And as the search procedure calculates a step size, the rate of convergence of the algorithm is significantly faster.

Computing a value of an objective function and its subgradient often involves considerable computational complexity. Therefore the following modification of a bundle-method (H. Schramm, J. Zowe), which does not use a search along the obtained direction is of interest.

Using this approach, the expression from the first step of the previous algorithm is replaced with

$$d(\rho_k) = \arg\min\{f_{CP}(x_k;d) \mid \frac{1}{2}d^T d \le \rho_k\}.$$

Now, instead of using some chosen in advance and more or less random ρ_k (or respectively t_k) the method uses a trust region approach: that is it increases/decreases ρ_k in a systematic way.

Such approach has the following two advantages. It provides a method of choosing *t*, and at the same time there is no need to carry out a search along the obtained direction. Clearly, the same approach can be applied also to $\frac{1}{2t_k}d^T d$ by adjusting t_k. It allows to reduce the problem to a problem of quadratic programming for which there is a number of reliable and effective algorithms, and software. Thus, we obtain the following iteration scheme :

Iteration $x_k \to x_{k+1}$:

1. Calculate $d_k = d(t_k) = \arg\min\{f_{CP}(x_k;d) + (\frac{1}{2t_k})d^T d \mid d \in \mathbb{R}^n\}$.
2. If $f(x_k + d_k)$ is "sufficiently" less than $f(x_k)$, then

   increase t_k and return to (1), or

   make a serious step, let $x_{k+1} = x_k + d_k$, and calculate $g_{k+1} \in \partial f(x_{k+1})$
3. If $f(x_k + d_k)$ is "insufficiently" less than $f(x_k)$, then

   decrease t_k and return to (1), or

   make a Null step and let $x_{k+1} = x_k$ and calculate $g_{k+1} \in \partial f(x_k + d_k)$.

Thus, the second algorithm uses a piecewise linear approximation of an objective function not just to obtain a quickest descent direction, but also to calculate a candidate point for the next iteration.

1.3.3. Method of cutting hyperplanes

These methods generate a sequence of points, which converges to a solution of the problem. Similar to the described above method, these points together with subgradients calculated in these points, describe a piecewise linear approximation of the problem called a localization set. Each subsequent point defines a new cutting hyperplane and improves the localization set until a satisfactory solution is obtained.

There are several modifications of the described method. The algorithm of choosing of a next "test" point, which is used for construction of a new plane, is the most essential difference between them. The most robust results are shown by so-called central methods, which calculate the next point as a centre of points, obtained at previous iterations.

1.3.4. Efficiency of nonsmooth optimization methods

By decomposition of the initial smooth problem we obtain two optimization problems one of which is a nonsmooth problem. Its solution often requires a lot of computations to find a value of function and its subgradient at each iteration of an algorithm of nonsmooth optimization. A typical example is a decomposition of maximin problem to two optimization problems one of which is a nonsmooth problem. To compute a value of the objective function one has to solve a number of optimization subproblems of the same type. It makes up a significant part of running time of the algorithm.

CHAPTER 2.
DEVELOPMENT OF THE ALGORITHM FOR THE MAXIMIN PROBLEM

2.1. Problem statement. Decomposition of the maximin problem to smooth and nonsmooth problems.

Methods of nonsmooth optimization can be applied not only to problems with nonsmooth functional parameters, but also in a variety of cases when a usual problem of nonlinear programming is difficult to solve due to its properties (a special structure of the problem and a large number of variables and constraints). Consider, for example, a nonlinear programming problem of the following form:

$$\begin{aligned} & f(\mathbf{x},\mathbf{y}) \to \min \\ & h_i(\mathbf{x},\mathbf{y}) \le 0, i = 1...r \\ & h_i(\mathbf{x},\mathbf{y}) = 0, i = r+1...p \\ & \mathbf{x} \in X \subset \mathbb{R}^n \end{aligned},$$

where $X \subset \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m$ are optimized variables, f and h_i are smooth functions. If the dimension of the problem is high, a number of experts (for example, A.Ben-Tal, J.Outrata, J.Zowe) consider it reasonable in certain cases to decompose the problem to two sequentially solved problems of lower dimension, one of which is nonsmooth. That is the original problem is replaced with the following problems:

$$\begin{aligned} & P(\mathbf{x}): \\ & \Phi(\mathbf{x}) = \min_{\mathbf{y}} \{ f(\mathbf{x},\mathbf{y}), \mathbf{y} \in F(\mathbf{x}) \}, \end{aligned} \tag{2.1}$$

where

$$F(\mathbf{x}) = \left\{ \mathbf{y} \in \mathbb{R}^m \mid h_i(\mathbf{x},\mathbf{y}) \le 0, i = 1...r; h_i(\mathbf{x},\mathbf{y}) = 0, i = r+1...p \right\}$$

is a multivalued mapping.

And then, one has to solve the following problem of nonsmooth optimization

$$\Phi(\mathbf{x}) \to \min_{\mathbf{x}}, \mathbf{x} \in X. \tag{2.2}$$

A number of experts in nonsmooth optimization (for example, A.Ben-Tal, J.Outrata, J.Zowe) suggest to use so-called bundle methods for the nonsmooth problem (2.2). Indeed, bundle methods are universal enough and are widely applied recently. On the other hand, one of the obvious shortcomings of applying bundle methods to the problem (2.2) is that these methods do not take into account the specificity of the problem (2.2).

Indeed, it is known [9] that under certain regularity conditions, there exists a derivative of the function $\Phi(x)$ at point x in direction $\mathbf{v}$ in the problem (2.1), which has the following form:

$$\Phi'(\mathbf{x};\mathbf{v}) = \min_{\mathbf{y}\in\omega(\mathbf{x})} \max_{\boldsymbol{\lambda}\in\Lambda(\mathbf{x},\mathbf{y})} \left\langle \nabla_{\mathbf{x}} L(\mathbf{x},\mathbf{y},\boldsymbol{\lambda});\mathbf{v}\right\rangle, \tag{2.3}$$

where

$$\omega(\mathbf{x}) = \left\{\mathbf{y}\in F(\mathbf{x}) \mid f(\mathbf{x},\mathbf{y}) = \Phi(\mathbf{x})\right\}$$

is a set of solutions of the problem (2.1),

$$\boldsymbol{\lambda} = \left(\lambda_1,\ldots,\lambda_p\right),$$

$$\mathrm{I}(\mathbf{x},\mathbf{y}) = \left\{i = 1...r : h_i(\mathbf{x},\mathbf{y}) = 0\right\},$$

is the set of active constraints,

$$L(\mathbf{x},\mathbf{y},\boldsymbol{\lambda}) = f(\mathbf{x},\mathbf{y}) + \sum_{i=1}^{p} \lambda_i h_i(\mathbf{x},\mathbf{y})$$

is a Lagrange function for the problem (2.1),

$$\Lambda(\mathbf{x},\mathbf{y}) = \left\{\boldsymbol{\lambda}\in\mathbb{R}^p \mid \nabla_{\mathbf{y}} L(\mathbf{x},\mathbf{y},\boldsymbol{\lambda}) = 0, \lambda_i \geq 0, i = 1...r; \lambda_i = 0, i\in I(\mathbf{x},\mathbf{y})\right\}$$

is the set of Lagrange multipliers for the problem (2.1) for a fixed $\mathbf{x}$ at a point $\mathbf{y}\in F(\mathbf{x})$.

Def A (RLI) *regularity condition* is said to be satisfied at point $\mathbf{z}_0 \in \mathrm{gr}\, F$ (or that a multivalued mapping is *RLI*-regular at point $\mathbf{z}_0$), if the system of elements $\nabla_{\mathbf{y}} h_i(\mathbf{z}_0), i\in I(\mathbf{z}_0)$ is linearly independent.

Theorem 2.1 [9].

Let mapping F be RLI-regular at point $\mathbf{z}_0 \in \mathrm{gr}\, F$. Then the set $\Lambda(\mathbf{z}_0)$ consists of a single point, i.e. $\Lambda(\mathbf{z}_0) = \{\boldsymbol{\lambda}\}$.

In particular, if the problem (2.1) has a unique solution, i.e. $\omega(\mathbf{x}) = \{\mathbf{y}\}$ then the formula (2.3) takes the following simple form:

$$\Phi'(\mathbf{x};\mathbf{v}) = \max_{\boldsymbol{\lambda}\in\Lambda(\mathbf{x},\mathbf{y})} \left\langle \nabla_{\mathbf{x}} L(\mathbf{x},\mathbf{y},\boldsymbol{\lambda}),\mathbf{v}\right\rangle \tag{2.4}$$

On the other hand, in any case the function

$$\Phi'(\mathbf{x};\mathbf{v}) = \max_{\boldsymbol{\lambda}\in\Lambda(\mathbf{x},\mathbf{y})} \left\langle \nabla_{\mathbf{x}} L(\mathbf{x},\mathbf{y},\boldsymbol{\lambda}),\mathbf{v}\right\rangle \tag{2.5}$$

is the upper convex approximation of the function Φ at point $\mathbf{x}$ for any $\mathbf{y}\in\omega(\mathbf{x})$.

A directional derivative of the function Φ allows to express a necessary condition of a minimum for the problem (2.2) in the simple form:

$$\Phi'(\mathbf{x};\mathbf{v}) \geq 0$$

for any vector $\mathbf{v}$ from a cone of possible directions $K(x,X)$ of set X at point x.

Thus, using formula (2.4), we can obtain a steepest descent direction v of the function $\Phi(x)$ and on its basis develop numerical algorithms of the quickest descent.

Consider now the following maximin problem

$$\max_{\mathbf{x}\in\mathbb{R}^n} \min_{\mathbf{y}\in F(\mathbf{x})} f(\mathbf{x},\mathbf{y}), \tag{2.6}$$

where

$$F(\mathbf{x}) = \left\{\mathbf{y}\in\mathbb{R}^m \mid h_i(\mathbf{x},\mathbf{y}) \leq 0, i = 1...p\right\},$$

is a multivalued mapping, $\mathbf{x}\in\mathbb{R}^n, \mathbf{y}\in\mathbb{R}^m$ are optimized variables, f and h_i are smooth functions. In the same way, the necessary condition of optimality can be expressed in the form

$$\Phi'(\mathbf{x};\mathbf{v}) \leq 0$$

And an algorithm of a steepest ascent for the given maximin problem can be developed using the necessary condition of optimality.

In the present work an algorithm for the maximin problem with coupled variables (2.6) is developed.

2.2. Development of a method for solving the maximin problem

In order to reduce technical difficulties at the current phase of research, I assume the problem $P(\mathbf{x})$ to be linear in variable $\mathbf{y}$ and containing only inequality constraints, i.e.

$$\begin{aligned} &f(\mathbf{x},\mathbf{y}) = < c(\mathbf{x}), \mathbf{y} >, \\ &h_i(\mathbf{x},\mathbf{y}) = < a_i(\mathbf{x}), \mathbf{y} > + b_i(\mathbf{x}) \leq 0, i = 1,..., p, \\ &X = \mathbb{R}^n \end{aligned} \tag{2.7}$$

where $a_i(x), b_i(x), c(x)$ are continuously differentiable functions.

It is known [9], that for the problem (2.7) $\Lambda(\mathbf{x}_0,\mathbf{y}_0) = \Lambda(\mathbf{x}_0)$ for all $\mathbf{y}_0 \in \omega(\mathbf{x}_0)$. That is the set $\Lambda(\mathbf{x}_0,\mathbf{y}_0)$ does not depend on $\mathbf{y}_0$.

And therefore

$$\Phi'(\mathbf{x}_0;\overline{\mathbf{x}}) = \min_{\mathbf{y}_0\in\omega(\mathbf{x}_0)} \max_{\boldsymbol{\lambda}\in\Lambda(\mathbf{x}_0)} \left\langle \nabla_{\mathbf{x}} L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda}), \overline{\mathbf{x}} \right\rangle.$$

At the same time it follows from [4] that for the function $\Phi(\mathbf{x})$ to reach a maximum (local) at point $\mathbf{x}_0$ it is necessary that

$$\Phi'(\mathbf{x}_0;\bar{\mathbf{x}}) \le 0$$

for all $\bar{\mathbf{x}}$.

As functions $f(\mathbf{x},\mathbf{y})$ and $h_i(\mathbf{x},\mathbf{y})$ are linear in variable $\mathbf{y}$, therefore $L(\mathbf{x},\mathbf{y},\boldsymbol{\lambda}) = f(\mathbf{x},\mathbf{y}) + \sum_{i=1}^{p} \lambda_i h_i(\mathbf{x},\mathbf{y})$ is also linear in $\mathbf{y}$. Hence $\nabla_{\mathbf{x}} L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda})$ is linear in $\mathbf{y}$ and obviously in $\boldsymbol{\lambda}$. As scalar product is linear in both arguments, therefore $\langle \nabla_{\mathbf{x}} L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda}),\bar{\mathbf{x}}\rangle$ is linear in $\boldsymbol{\lambda}$, $\bar{\mathbf{x}}$ and *y*. Hence $\langle \nabla_{\mathbf{x}} L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda}),\bar{\mathbf{x}}\rangle$ is a convex-concave function and we can apply the minimax theorem. We can show that the necessary condition of a maximum of function $\Phi(\mathbf{x})$ at a point $\mathbf{x}_0$ has the following form:

$$0 \ge \max_{|\bar{\mathbf{x}}|\le 1} \Phi'(\mathbf{x}_0;\bar{\mathbf{x}}) = \max_{|\bar{\mathbf{x}}|\le 1} \min_{\mathbf{y}_0\in\omega(\mathbf{x}_0)} \max_{\lambda\in\Lambda(\mathbf{x}_0)} \langle \nabla_{\mathbf{x}} L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda}),\bar{\mathbf{x}}\rangle =$$

$$= \max_{|\bar{\mathbf{x}}|\le 1} \max_{\lambda\in\Lambda(\mathbf{x}_0)} \min_{\mathbf{y}_0\in\omega(\mathbf{x}_0)} \langle \nabla_{\mathbf{x}} L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda}),\bar{\mathbf{x}}\rangle = \max_{\lambda\in\Lambda(x_0)} \left(\max_{|\bar{x}|\le 1} \min_{y_0\in\omega(x_0)} \langle \nabla_{\mathbf{x}} L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda}),\bar{\mathbf{x}}\rangle \right)$$

Theorem 2.2.

Let $\mathbf{a}$ and $\mathbf{y}$ be vectors and $Y \subset \mathbb{R}^n$ be an arbitrary subset. Then the following holds

$$\max_{|\mathbf{a}|\le 1} \min_{\mathbf{y}\in Y} \langle \mathbf{a},\mathbf{y}\rangle = \min_{\mathbf{y}\in Y} |\mathbf{y}|$$

proof:

Let α denotes the angle between vectors a and y. We have

$$\max_{|\mathbf{a}|\le 1} \min_{\mathbf{y}\in Y} \langle \mathbf{a},\mathbf{y}\rangle = \max_{|\mathbf{a}|\le 1} \min_{\mathbf{y}\in Y} |\mathbf{a}||\mathbf{y}|\cos\alpha =$$

$$= \max_{|\mathbf{a}|\le 1} \left(|\mathbf{a}| \min_{\mathbf{y}\in Y} \left(|\mathbf{y}|\cos\alpha \right) \right) = \max_{|\mathbf{a}|=1} \left(\min_{\mathbf{y}\in Y} |\mathbf{y}|\cos\alpha \right)$$

As we can always choose vector $\mathbf{a}$ such that $\cos\alpha = 1$, therefore

$$\max_{|\mathbf{a}|=1} \left(\min_{\mathbf{y}\in Y} |\mathbf{y}|\cos\alpha \right) = \min_{\mathbf{y}\in Y} |\mathbf{y}|$$

q.e.d.

By the last theorem we have that

$$\max_{|\bar{\mathbf{x}}|\le 1} \min_{\mathbf{y}_0\in\omega(\mathbf{x}_0)} \langle \nabla_{\mathbf{x}} L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda}),\bar{\mathbf{x}}\rangle = \min_{\mathbf{y}_0\in\omega(\mathbf{x}_0)} |\nabla_{\mathbf{x}} L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda})|$$

hence

$$0 \geq \max_{\boldsymbol{\lambda}\in\Lambda(\mathbf{x}_0)}\left(\max_{|\overline{\mathbf{x}}|\leq 1}\min_{\mathbf{y}_0\in\omega(\mathbf{x}_0)}\left\langle\nabla_{\mathbf{x}}L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda}),\overline{\mathbf{x}}\right\rangle\right)=\max_{\boldsymbol{\lambda}\in\Lambda(\mathbf{x}_0)}\min_{\mathbf{y}_0\in\omega(\mathbf{x}_0)}\left|\nabla_{\mathbf{x}}L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda})\right|.$$

That is it is necessary, that

$$0 \geq \max_{\boldsymbol{\lambda}\in\Lambda(\mathbf{x}_0)}\min_{\mathbf{y}_0\in\omega(\mathbf{x}_0)}\left|\nabla_{\mathbf{x}}L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda})\right|,$$

The last is equivalent to that the inequality

$$\min_{\mathbf{y}_0\in\omega(\mathbf{x}_0)}\left|\nabla_{\mathbf{x}}L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda})\right|\leq 0$$

is satisfied for every $\boldsymbol{\lambda}\in\Lambda(\mathbf{x}_0)$.

Which is equivalent to that for any $\boldsymbol{\lambda}\in\Lambda(\mathbf{x}_0)$ there exists a point $\mathbf{y}_0\in\omega(\mathbf{x}_0)$ such that

$$\left|\nabla_{\mathbf{x}}L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda})\right|\leq 0.$$

Obviously, the condition $\left|\nabla_{\mathbf{x}}L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda})\right|\leq 0$ is equivalent to $\nabla_{\mathbf{x}}L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda})=0$.

Thus, I have proved the following result providing a necessary condition of a maximum point.

Theorem 2.3.

The function $\Phi(\mathbf{x})$ has a maximum (local) at a point $\mathbf{x}_0$, if the following condition is satisfied:

for any $\boldsymbol{\lambda}\in\Lambda(\mathbf{x}_0)=\Lambda(\mathbf{x}_0,\mathbf{y}_0)$ there exists such $\mathbf{y}_0\in\omega(\mathbf{x}_0)$ that $\nabla_{\mathbf{x}}L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda})=0$.

As the steepest ascent direction $\overline{\mathbf{x}}_0$ of the function $\Phi(\mathbf{x})$ at point $\mathbf{x}_0$ must satisfy the condition

$$\max_{|\overline{\mathbf{x}}|\leq 1}\Phi'(\mathbf{x}_0;\overline{\mathbf{x}})=\Phi'(\mathbf{x}_0;\overline{\mathbf{x}}_0),$$

and accordingly

$$\max_{|\overline{\mathbf{x}}|\leq 1}\Phi'(\mathbf{x}_0;\overline{\mathbf{x}})=\max_{|\overline{\mathbf{x}}|\leq 1}\min_{\mathbf{y}_0\in\omega(\mathbf{x}_0)}\max_{\boldsymbol{\lambda}\in\Lambda(\mathbf{x}_0)}\left\langle\nabla_{\mathbf{x}}L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda}),\overline{\mathbf{x}}\right\rangle.$$

Or if $\omega(\mathbf{x}_0)=\{\mathbf{y}_0\}$ then

$$\max_{|\overline{\mathbf{x}}|\leq 1}\Phi'(x_0;\overline{x})=\max_{|\overline{\mathbf{x}}|\leq 1}\max_{\boldsymbol{\lambda}\in\Lambda(\mathbf{x}_0)}\left\langle\nabla_{\mathbf{x}}L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda}),\overline{\mathbf{x}}\right\rangle=$$
$$=\max_{\boldsymbol{\lambda}\in\Lambda(\mathbf{x}_0)}\max_{|\overline{\mathbf{x}}|\leq 1}\left\langle\nabla_{\mathbf{x}}L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda}),\overline{\mathbf{x}}\right\rangle=\max_{\boldsymbol{\lambda}\in\Lambda(\mathbf{x}_0)}\left|\nabla_{\mathbf{x}}L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda})\right|.$$

Hence

$$\Phi'(\mathbf{x}_0;\overline{\mathbf{x}}_0)=\max_{\boldsymbol{\lambda}\in\Lambda(\mathbf{x}_0)}\left|\nabla_{\mathbf{x}}L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda})\right|.$$

The directional derivative will be maximal when the direction $\bar{\mathbf{x}}$ coincides with the direction of the steepest ascent of the function $\Phi(\mathbf{x})$ at point $\mathbf{x}_0$. Therefore we have proved the following statement.

Theorem 2.4

Let $\omega(\mathbf{x}_0) = \{\mathbf{y}_0\}$. The steepest ascent direction of function $\Phi(\mathbf{x})$ at point $\mathbf{x}_0$ coincides with $\bar{x} = \dfrac{\nabla_{\mathbf{x}} L(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\lambda}_{\max})}{\left|\nabla_{\mathbf{x}} L(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\lambda}_{\max})\right|}$, where $\boldsymbol{\lambda}_{\max}$ is the solution of the problem $\left|\nabla_{\mathbf{x}} L(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\lambda})\right| \to \max\limits_{\lambda \in \Lambda(\mathbf{x}_0)}$. The last problem is equivalent to the problem $2 < \mathbf{p}, \mathrm{H}\boldsymbol{\lambda} > + < \boldsymbol{\lambda}, (\mathrm{H}^T \mathrm{H})\, \boldsymbol{\lambda} > \to \max\limits_{\lambda \in \Lambda(\mathbf{x}_0)}$.

This is a difficult problem of maximization of a convex quadratic function over a convex polyhedron (see for example [11]). However, as it will be shown further, the set $\Lambda(\mathbf{x}_0)$ consists of a single point under quite general conditions of *RLI-regularity* of the problem.

2.3. Finding the steepest ascent direction for the problem of unconditional maximization for the function $\Phi(\mathbf{x})$

Let's assume that in the main problem (2.6) functions f and h_i are linear in variable $\mathbf{y}$. Thus the original problem can be presented in the following form:

$$\begin{aligned} &\max_{\mathbf{x} \in \mathbb{R}^n} \min_{\mathbf{y} \in F(\mathbf{x})} f(\mathbf{x}, \mathbf{y}), \\ &f(\mathbf{x}, \mathbf{y}) = < c(\mathbf{x}), \mathbf{y} >, \\ &F(\mathbf{x}) = \left\{ \mathbf{y} \in \mathbb{R}^m \mid h_i(\mathbf{x}, \mathbf{y}) \le 0, i = 1 \ldots p \right\}, \\ &h_i(\mathbf{x}, \mathbf{y}) = < a_i(\mathbf{x}), \mathbf{y} > + b_i(\mathbf{x}) \le 0, i = 1, \ldots, p. \end{aligned} \tag{2.8}$$

We need to find the steepest ascent direction $\mathbf{v} = \mathbf{v}_0$ at a point $\mathbf{x} = \mathbf{x}_0$ given the value of $\mathbf{y}_0 \in \omega(\mathbf{x})$ for the problem $P(\mathbf{x}_0)$. Therefore we have to solve the problem (2.9) with a normed direction v:

$$\max_{\|\mathbf{v}\|=1} \max_{\lambda \in \Lambda} \left\langle \nabla_{\mathbf{x}} L(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\lambda}), \mathbf{v} \right\rangle, \tag{2.9}$$

where

$$\Lambda = \Lambda(\mathbf{x}_0, \mathbf{y}_0) = \left\{ \boldsymbol{\lambda} \in \mathbb{R}^r \mid \nabla_{\mathbf{y}} L(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\lambda}) = 0, \lambda_i \ge 0, i = 1 \ldots r; \lambda_i = 0, i \notin I(\mathbf{x}_0, \mathbf{y}_0) \right\},$$

$$I(\mathbf{x}_0, \mathbf{y}_0) = \left\{ i = 1 \ldots r \mid h_i(\mathbf{x}_0, \mathbf{y}_0) = 0 \right\}$$

is the set of active constraints at the point $(\mathbf{x}_0, \mathbf{y}_0)$.

Let's transform the expression (2.9):

$$\max_{\|\mathbf{v}\|=1}\max_{\lambda\in\Lambda}\langle\nabla_{\mathbf{x}}L(\mathbf{x}_0,\mathbf{y}_0,\boldsymbol{\lambda}),\mathbf{v}\rangle=\max_{\|\mathbf{v}\|=1}\max_{\lambda\in\Lambda}\left\langle\nabla_{\mathbf{x}}(f(\mathbf{x}_0,\mathbf{y}_0)+\sum_{i=1}^{r}\lambda_i h_i(\mathbf{x}_0,\mathbf{y}_0)),\mathbf{v}\right\rangle=$$

$$\max_{\|\mathbf{v}\|=1}\max_{\lambda\in\Lambda}\left\langle\mathbf{p}+\sum_{i=1}^{r}\lambda_i\nabla_x h_i(\mathbf{x}_0,\mathbf{y}_0),\mathbf{v}\right\rangle=\max_{\|\mathbf{v}\|=1}\max_{\lambda\in\Lambda}\langle\mathbf{p}+\mathbf{H}\boldsymbol{\lambda},\mathbf{v}\rangle=\max_{\|\mathbf{v}\|=1}\max_{\lambda\in\Lambda}\{\langle\mathbf{v},\mathbf{H}\boldsymbol{\lambda}\rangle+\langle\mathbf{p},\mathbf{v}\rangle\}.$$

Therefore the problem (2.9) is reduced to the problem

$$\max_{\|\mathbf{v}\|=1}\max_{\lambda\in\Lambda}\{\langle\mathbf{v},\mathbf{H}\boldsymbol{\lambda}\rangle+\langle\mathbf{p},\mathbf{v}\rangle\},\tag{2.10}$$

where the matrix $\mathbf{H}=\begin{bmatrix}\frac{\partial h_1}{\partial x_1}\cdots\frac{\partial h_r}{\partial x_1}\\ \cdots\cdots\cdots\\ \frac{\partial h_1}{\partial x_n}\cdots\frac{\partial h_r}{\partial x_n}\end{bmatrix}$ is evaluated at the point $(\mathbf{x}_0,\mathbf{y}_0)$, $\mathbf{p}=\nabla_{\mathbf{x}}f(\mathbf{x}_0,\mathbf{y}_0)$.

As scalar product reaches maximum when directions of vectors $\mathbf{v}$ and $\mathbf{H}\boldsymbol{\lambda}+\mathbf{p}$ coincide, and as $|\mathbf{v}|=1$ then $\mathbf{v}=\frac{\mathbf{H}\boldsymbol{\lambda}+\mathbf{p}}{|\mathbf{H}\boldsymbol{\lambda}+\mathbf{p}|}$. Therefore we obtain from (2.10)

$$\max_{\|\mathbf{v}\|=1}\max_{\lambda\in\Lambda}\{\langle\mathbf{v},\mathbf{H}\boldsymbol{\lambda}\rangle+\langle\mathbf{p},\mathbf{v}\rangle\}=\max_{\lambda\in\Lambda}\max_{\|\mathbf{v}\|=1}\langle\mathbf{H}\boldsymbol{\lambda}+\mathbf{p},\mathbf{v}\rangle=\max_{\lambda\in\Lambda}\{|\mathbf{H}\boldsymbol{\lambda}+\mathbf{p}|\}=\max_{\lambda\in\Lambda}|\mathbf{H}\boldsymbol{\lambda}+\mathbf{p}|.$$

The solution $\boldsymbol{\lambda}\in\Lambda$ of the last problem, obviously, coincides with the solution of the problem

$$\max_{\lambda\in\Lambda}\langle\mathbf{H}\boldsymbol{\lambda}+\mathbf{p},\mathbf{H}\boldsymbol{\lambda}+\mathbf{p}\rangle=\max_{\lambda\in\Lambda}\{\langle\boldsymbol{\lambda},\mathrm{H}^T\mathbf{H}\boldsymbol{\lambda}\rangle+2\langle\mathbf{p},\mathbf{H}\boldsymbol{\lambda}\rangle+|\mathbf{p}|^2\},$$

Or if we omit a constant term $|\mathbf{p}|^2$, it coincides with the solution of the problem

$$\begin{cases}\langle\boldsymbol{\lambda},\mathrm{H}^T\mathbf{H}\boldsymbol{\lambda}\rangle+2\langle\mathbf{p},\mathbf{H}\boldsymbol{\lambda}\rangle\to\max\limits_{\lambda}\\ \boldsymbol{\lambda}\in\Lambda\end{cases}\tag{2.11}$$

Solving the problem of maximization of a convex quadratic function over a convex polyhedron (2.11) we obtain $\boldsymbol{\lambda}$.

In the sequel, in order to avoid this quite a difficult problem, I assume that a general enough *RLI-regularity* condition holds for the problem. By the theorem 2.1, under *RLI-regularity* condition, the set Λ consists of the single point. Therefore, it is enough to check *RLI-regularity* condition for the problem and find a unique point of the set Λ. Thus, if we have that $\mathbf{H}\boldsymbol{\lambda}+\mathbf{p}=0$, then the point $\mathbf{x}_0$ satisfies the necessary condition of optimality and it is a point of local maximum. The problem is solved. Its solution is $(\mathbf{x}_0,\mathbf{y}_0)$. Otherwise (if $\mathbf{H}\boldsymbol{\lambda}+\mathbf{p}\neq 0$) using the obtained λ, we obtain a steepest ascent direction:

$$\mathbf{v} = \frac{\mathbf{H}\boldsymbol{\lambda} + \mathbf{p}}{|\mathbf{H}\boldsymbol{\lambda} + \mathbf{p}|} = \frac{\nabla_{\mathbf{x}} L(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\lambda})}{|\nabla_{\mathbf{x}} L(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\lambda})|}. \tag{2.12}$$

On its basis, I will obtain the next approximation of a vector $\mathbf{x}_0$ by maximization of the function $\Phi(\mathbf{x}_0 + \beta\mathbf{v})$ in β.

The proposed method has the following advantages. First, the dimension of the obtained problem is low. Secondly, under the general enough condition of RLI-regularity (i.e. linear independence of gradients of constraints h_i), the admissible set $\Lambda(\mathbf{x}_0)$ consists of the unique point, and the solution of the last problem is trivial.

2.4. Description of the conceptual algorithm

Input data for the problem of nonlinear programming (2.6) are: numbers *r*, *n*, *m*; smooth function $f(\mathbf{x}, \mathbf{y})$ linear in variable **y**; smooth functions $h_i(\mathbf{x}, \mathbf{y})$ also linear in **y**; an initial vector $\mathbf{x}_0$. Let $\mathbf{x}_k = \mathbf{x}_0$, $k = 0$.

Step 1. Given some fixed value $\mathbf{x} = \mathbf{x}_k$ from $X = \mathbb{R}^n$, we obtain a problem $P(\mathbf{x}_k)$. Solving it with simplex-method, we obtain a solution $\mathbf{y}_k = \omega(\mathbf{x}_k)$ of the problem $P(\mathbf{x}_k)$.

Step 2. Solving the problem of maximization of a convex quadratic function on a convex polyhedron (2.11), we obtain a vector $\boldsymbol{\lambda}_k$. As it has been shown above, at this step it is enough to check the condition of RLI-regularity for the problem and find a feasible point of the set Λ. It will be the vector $\boldsymbol{\lambda}_k$.

Step 3. If $\mathbf{H}_k\boldsymbol{\lambda}_k + \mathbf{p}_k = 0$, then the point $\mathbf{x}_0$ satisfies the necessary condition of optimality and it is a point of local maximum. The problem is solved. Its solution is $(\mathbf{x}_0, \mathbf{y}_0)$. Otherwise, if $\mathbf{H}_k\boldsymbol{\lambda}_k + \mathbf{p}_k \neq 0$ we move to step 4.

Step 4. Using the obtained at step 2 vector $\boldsymbol{\lambda}_k$, we obtain a steepest ascent direction $\mathbf{v}_k$ by formula (2.12).

Step 5. Determine a step size β_k.

Step 6. The next approximation of a vector $\mathbf{x}_0$ is obtained by the formula $\mathbf{x}_{k+1} = \mathbf{x}_k + \beta_k\mathbf{v}_k$. We move to the step 1, letting $k = k + 1$.

2.4.1. Finding an extremum point for the problem $P(\mathbf{x})$

As it was already mentioned earlier, we have to solve the problem $P(\mathbf{x}_k)$ to find a vector $\mathbf{y}_k$ given vector $\mathbf{x}_k$ at each iteration step of the problem (2.8). As functions f and h_i are linear in variable **y**, the problem $P(\mathbf{x}_k)$ is a linear programming problem with respect to *y*, and it can be solved with the simplex-method.

Let we are given a value $\mathbf{x} = \mathbf{x}_k$. The problem $P(\mathbf{x})$ for $X = \mathbb{R}^n$ can be written as follows:

$$\begin{aligned} &\Phi(\mathbf{x}_k) = \min_{\mathbf{y}}(f(\mathbf{x}_k, \mathbf{y})), \mathbf{y} \in F(\mathbf{x}_k), \\ &F(\mathbf{x}_k) = \left\{\mathbf{y} \in \mathbb{R}^m \mid h_i(\mathbf{x}_k, \mathbf{y}) \le 0,\ i = 1...r\right\} \end{aligned} \tag{2.13}$$

Denote by $\psi(\mathbf{y}) = f(\mathbf{x}_k, \mathbf{y}),\ g_i(\mathbf{y}) = h_i(\mathbf{x}_k, \mathbf{y})$. We can rewrite the problem (2.13) as follows:

$$\begin{aligned} &\psi(\mathbf{y}) \to \min \\ &g_i(\mathbf{y}) \le 0, i = 1...r \\ &\mathbf{y} \in \mathbb{R}^m \end{aligned} \tag{2.14}$$

It is a linear programming problem. Solving it, we obtain a vector $\mathbf{y}_k$.

2.4.2. Finding the steepest ascent direction

In an expanded form the problem (2.11) can be written as follows:

$$\begin{cases} \left\langle \boldsymbol{\lambda}, \mathbf{H}^T\mathbf{H}\boldsymbol{\lambda} \right\rangle + \left\langle 2\mathbf{p}, \mathbf{H}\boldsymbol{\lambda} \right\rangle \to \max \\ \boldsymbol{\lambda} \in \begin{cases} \boldsymbol{\lambda} \in \mathbb{R}^r \mid \nabla_{\mathbf{y}} L(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\lambda}) = 0;\ \lambda_i \ge 0,\ i = 1...r; \\ \lambda_i = 0,\ i \notin \left\{i = 1...r \mid h_i(\mathbf{x}_0, \mathbf{y}_0) = 0\right\} \end{cases} \end{cases} \tag{2.15}$$

The problem (2.15) is the problem of maximization of a convex quadratic function on a convex polyhedron, which in its general form can be written as follows:

$$\begin{cases} f(\mathbf{x}) = \frac{1}{2}\left\langle \mathbf{x}, \mathbf{D}\mathbf{x} \right\rangle + \left\langle \mathbf{c}, \mathbf{x} \right\rangle \to \max \\ \mathbf{A}\mathbf{x} = \mathbf{b} \\ \mathbf{x}(J_+) \ge 0 \end{cases},$$

where $\mathbf{D}_{r\times r} = 2\mathbf{H}^T\mathbf{H}, \mathbf{c}_{r\times 1} = (2\mathbf{p}^T\mathbf{H})^T, \mathbf{x}_{r\times 1} = \boldsymbol{\lambda}$. The constraints $\mathbf{x}(J_+) \ge 0$ in our case become $\lambda_i \ge 0, i = 1...r$. And constraints $\mathbf{A}\mathbf{x} = \mathbf{b}$ become $\nabla_{\mathbf{y}} L(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\lambda}) = 0, \lambda_i = 0, i \notin I(\mathbf{x}_0, \mathbf{y}_0)$.

Consider constraints $\nabla_{\mathbf{y}} L(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\lambda}) = 0, \lambda_i = 0, i \notin I(\mathbf{x}_0, \mathbf{y}_0)$,

$$\nabla_{\mathbf{y}} L(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\lambda}) = \nabla_{\mathbf{y}} f(\mathbf{x}_0, \mathbf{y}_0) + \sum_{i=1}^{r} \lambda_i \nabla_y h_i(\mathbf{x}_0, \mathbf{y}_0) .$$

Therefore $\mathbf{b} = -\nabla_{\mathbf{y}} f(\mathbf{x}_0, \mathbf{y}_0)$, $\mathbf{A} = \begin{bmatrix} \frac{\partial h_1}{\partial y_1} \cdots \frac{\partial h_r}{\partial y_1} \\ \cdots\cdots\cdots\cdots \\ \frac{\partial h_1}{\partial y_m} \cdots \frac{\partial h_r}{\partial y_m} \end{bmatrix}, \mathbf{x} = \boldsymbol{\lambda}.$

Taking into account conditions $\lambda_i = 0, i \notin I(\mathbf{x}_0, \mathbf{y}_0)$, we obtain that for $| I(\mathbf{x}_0, \mathbf{y}_0) | = k$, the number of λ_i satisfying the condition $\lambda_i = 0$ is at least $r - k$. Hence, the problem of maximisation of a convex quadratic function on a convex polyhedron (2.15) can be simplified by reduction of its dimension.

Let $\lambda_i = 0, i \in \{a_1, a_2, ..., a_{r-k}\}$. In this case it is possible to reduce the dimension of a vector $\boldsymbol{\lambda}$ by removing from it elements $\lambda_i, i \in \{a_1, a_2, ..., a_{r-k}\}$. The dimension of the vector $\boldsymbol{\lambda}$ thus becomes equal to $k \times 1$. By removing components of the vector $\boldsymbol{\lambda}$ we reduce the dimensions of matrixes **A**, **D** and a vector **c**. From the matrix **A** we delete columns which indexes are in the set $\{a_1, a_2, ..., a_{r-k}\}$. Since $\mathbf{D}_{r \times r} = 2\mathbf{H}^T\mathbf{H}$, we have to modify matrixes $\mathbf{H}$ and $\mathbf{H}^T$ to reduce the dimension of the matrix **D**. One should remove those columns from the matrix $\mathbf{H}$, and those rows from matrix $\mathbf{H}^T$, which indexes are in the set $\{a_1, a_2, ..., a_{r-k}\}$. In the same way we reduce the dimension of the vector $\mathbf{c}_{r \times 1} = (2\mathbf{p}^T\mathbf{H})^T$.

Therefore we have to solve the problem of maximization of a convex quadratic function on a convex polyhedron of lower dimension

$$\begin{cases} \overline{f}(\boldsymbol{\lambda}) = \frac{1}{2}\langle \boldsymbol{\lambda}, \overline{\mathrm{D}}\, \boldsymbol{\lambda} \rangle + \langle \overline{\mathrm{c}}, \boldsymbol{\lambda} \rangle \to \max \\ \overline{\mathbf{A}}\boldsymbol{\lambda} = \overline{\mathrm{b}} \\ \overline{\lambda_i} \geq 0, i = 1...k \end{cases} \qquad (2.16)$$

where $\overline{\mathbf{D}}, \overline{\mathbf{A}}, \overline{\mathbf{c}}, \overline{\mathbf{b}}$ are new vectors and matrixes obtained by removing redundant elements.

Thus, by obtaining a feasible point of a polyhedron we obtain a vector $\overline{\lambda_{k \times 1}}$. Then, in order to obtain a required vector $\boldsymbol{\lambda}$ it is necessary to expand a column vector $\overline{\lambda_{k \times 1}}$ by adding $r - k$ zeros to it, which where deleted during the reduction of the dimension of the problem (2.15).

2.4.3. Finding an iteration step size β_k

The method of choosing a step size along the obtained direction strongly influences the rate of convergence, as well as the convergence of the method in general.

In case of a smooth objective function, one-dimensional minimization methods show good results. However, in case of a nonsmooth objective function, this method does not guarantee the convergence to an optimum point at all. Another drawback of 1-dimensional minimization is the necessity of a large amount of function value calculations. As often happens, such calculations are quite computationally expensive as in particular in my problem.

I carried out numerical experiments with four methods of choosing a step size described below.

Method 1. Fixed step size.

Description. In this method, a step size is set as a constant by the user. This method of choosing a step size is the simplest one. At the same time, as it will be shown in a computational experiment, it is the least effective one. The greater the step size, the lower the precision of the obtained solution. If the step size is too small and the starting point is far from the solution, then the algorithm requires hundreds and thousands of iterations.

Advantages. It is the simplest method of choosing a step size in terms of algorithmic implementation. It does not require additional calculations of the objective function values.

Shortcomings. It requires a lot of iterations. The precision of the obtained solution depends on the step size. Also, in general, the error of the obtained solution can be as great as the length of the step. It is impossible to say in advance what step size will be optimal for a given function and a given starting point. The number of required iterations strongly depends on the starting point and the chosen step size.

Method 2. Choice of a step size using 1-dimensional maximization

If significant effort is necessary for calculating the direction v when minimizing a smooth function, then the step size β_k is usually calculated by minimizing (maximizing) the function $\psi(\beta) = \Phi\left(\mathbf{x}_k - \beta\mathbf{v}\right)$. The accuracy of calculating the minimum point of function $\psi(\beta)$ needs to be harmonized with the accuracy of calculation of values of function $\Phi(\mathbf{x})$. Besides, the

accuracy of calculating the one-dimensional minimum depends on the number of calculations of function values of $\Phi(\mathbf{x})$ along the direction $-\mathbf{v}$.

Theoretical estimations often indicate that the method is convergent at a rate for example $O(1/m)$. However, even while minimizing a smooth function, it is often discovered in the course of calculation that when the number of iterations increases, either the rate of convergence decreases, or the process ceases to converge at all. One of the causes of this may be the loss of precision in calculation of one-dimensional minimum. This undesirable situation can be sometimes corrected by increasing the accuracy of one-dimensional minimization. It is usually necessary to increase the accuracy of one-dimensional minimization or maximization in the neighborhood of the minimum point or when the point $\mathbf{x}_k$ is in a so-called ravine, i.e. when the derivative in some directions is close to zero.

I implemented the dichotomy method for one-dimensional maximization.

Advantages:

- In case of a smooth function, the method converges quite fast to the required solution. However, the minimum function

$$\Phi(\mathbf{x}) = \min_{\mathbf{y} \in F(\mathbf{x})} f(\mathbf{x}, \mathbf{y})$$

can be nonsmooth. As I will show during the computational experiment, when an objective function is nonsmooth, the method of one-dimensional minimization may not converge to an optimum point at all.

Shortcomings:

- As I will show in the next chapter, the iteration process gets into an infinite loop when the point $\mathbf{x}_k$ nears the kink of the objective function.
- Bad convergence in an area where the function is nonsmooth.
- It requires a lot of calculations of the objective function values along the obtained direction. This in turn requires solving a lot of problems of linear programming in my case.

Method 3. The step size β_k is obtained in the following form:

$$\beta_k = c\left|\mathbf{H}\boldsymbol{\lambda} + \mathbf{p}\right|,$$

where c is a constant,

$$|\mathbf{H}\boldsymbol{\lambda} + \mathbf{p}| = |\nabla_x L(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\lambda})|.$$

Advantages:

- This method does not require additional calculation of values of the minimum function in determining the step size.
- It does not require a large number of iterations.
- It does not get caught in an endless loop in the area where the function is nonsmooth.
- It takes into account the behavior of the objective function at the given point.

Shortcomings:

- The rate of convergence decreases in the area where the objective function is nonsmooth.
- In order to obtain the best result, this method requires choosing the value of constant c heuristically. Its value may depend on the objective function.

2.4.4. The modified bundle method for the construction of an ascent direction

The present method of choosing a step size and ascent direction is an improved modification of the previous method. Here I have applied the described below modification of the bundle method together with the developed algorithm of finding a steepest ascent direction. As it will be shown during the computational experiment, the rate of convergence strongly decreases when the iterative process nears a kink of the objective function. In the neighbourhood of a kink of the objective function its gradient has a rupture. Therefore even an arbitrary small change of argument of the objective function involves a finite change of the gradient.

In order to increase the stability of the iteration process, and hence the rate of convergence, I applied the analogy with a physical ball having a certain mass. The main difference (for our goal) of a physical ball from a mathematical point is that it has inertia. Therefore the physical ball cannot change its moving direction arbitrary fast. Due to inertia the physical ball has certain stability during its movement. In the present method of choosing a step size and ascent direction I have applied an idea of adding inertia to a mathematical point.

In order to add inertia while choosing an ascent direction, I have used the information about the ascent direction of two previous iterations. Namely:

$$\mathbf{v} = \frac{\mathbf{v}_k + \frac{1}{2}\mathbf{v}_{k-1} + \frac{1}{4}\mathbf{v}_{k-2}}{\left|\mathbf{v}_k + \frac{1}{2}\mathbf{v}_{k-1} + \frac{1}{4}\mathbf{v}_{k-2}\right|},$$

where $\mathbf{v}$ is the desired ascent direction,

$\mathbf{v}_k$ is an ascent direction obtained at the current step,

$\mathbf{v}_{k-1}$ is an ascent direction obtained at the previous iteration,

$\mathbf{v}_{k-2}$ is an ascent direction obtained two iteration before.

The step size is calculated as

$$\beta_k = c\left|\mathbf{H}\boldsymbol{\lambda} + \mathbf{p}\right|,$$

where c is a constant and

$$\left|\mathbf{H}\boldsymbol{\lambda} + \mathbf{p}\right| = \left|\nabla_{\mathbf{x}} L(\mathbf{x}_0, \mathbf{y}_0, \boldsymbol{\lambda})\right|.$$

Clearly the present modification of choosing of an ascent direction does not require additional calculations of function values. Taking into account distances to the previous points would further increase the rate of convergence of the iteration process. However, as it will be shown during the computational experiment, even without it, the rate of convergence of the present method essentially exceeds all results obtained earlier.

Advantages of the present method:

- It is stable in the area, where the objective function is nonsmooth.
- This method does not require additional calculation of values of a minimum function in determining the step size.
- It requires a small number of iterations.
- It does not get caught in an endless loop in the area where the function is nonsmooth
- It takes the behavior of the function into account at the given point.

Shortcomings:

- In order to obtain the best result, this method requires choosing the value of constant c heuristically. Its value may depend on the objective function.

During the computational experiment, I will investigate the efficiency and convergence of described methods of choosing of an ascent direction and an iteration step size.

CHAPTER 3.
COMPUTATIONAL STUDY

3.1. Description of the computational study

On the basis of the described algorithm, I have implemented a program for the maximin problem with coupled variables using the system of computer algebra Maple.

Using the developed program I have carried out a number of computational experiments. I have drawn the graph of the minimum function $\Phi(\mathbf{x})$, three-dimensional graph and level sets and shown the iterative process.

As a test objective function, I took a function $f(\mathbf{x},\mathbf{y})$ linear in variable y and containing quadratic terms in $\mathbf{x}$. Constraints h_i are linear in $\mathbf{x}$ and $\mathbf{y}$. That is they form a convex polyhedron:

$$
\begin{aligned}
&f(\mathbf{x},\mathbf{y}) = -x_1^2 y_1 - x_2^2 y_2,\\
&h_1 = -y_1,\\
&h_2 = y_1 - 5,\\
&h_3 = -y_2,\\
&h_4 = y_2 - 5,\\
&h_5 = x_1 + x_2 + y_1 + 2y_2 - 4,\\
&h_i \le 0, i = \overline{1,5}.
\end{aligned}
$$

Let's show that under the above constraints the following inequality holds:

$$\max_{\mathbf{x}\in\mathbb{R}^n} \min_{\mathbf{y}\in F(\mathbf{x})} f(\mathbf{x},\mathbf{y}) \le 0.$$

We have

$$
\begin{aligned}
&\min_{\mathbf{y}\in F(\mathbf{x})} f(\mathbf{x},\mathbf{y}) = \min_{y\in F(x)} \left[-x_1^2 y_1 - x_2^2 y_2\right] =\\
&= \min_{y\in F(x)} \left[-(x_1^2 y_1 + x_2^2 y_2)\right] = -\max_{\mathbf{y}\in F(\mathbf{x})} (x_1^2 y_1 + x_2^2 y_2)
\end{aligned}.
$$

As $x_i^2 \ge 0$ and $0 \le y_i \le 5$, $i = 1,2$, then $x_i^2 y_i \ge 0$. Therefore $x_1^2 y_1 + x_2^2 y_2 \ge 0$, hence it follows that $\max_{\mathbf{y}\in F(\mathbf{x})} (x_1^2 y_1 + x_2^2 y_2) \ge 0$, which is equivalent to

$$\min_{\mathbf{y}\in F(\mathbf{x})} f(\mathbf{x},\mathbf{y}) = -\max_{\mathbf{y}\in F(\mathbf{x})} (x_1^2 y_1 + x_2^2 y_2) \le 0$$

for any $\mathbf{x} \in \mathbb{R}^n$. The required statement follows.

I will need this estimation to analyze a precision of solutions obtained during the numerical experiments.

Now let's show that in a polyhedron of admissible solutions of the problem, the strict equality is attained

$$\max_{\mathbf{x}\in\mathbb{R}^n} \min_{\mathbf{y}\in F(\mathbf{x})} f(\mathbf{x},\mathbf{y}) = 0.$$

Let $x_1 = 0, x_2 = 0$. Then

$$f(\mathbf{x},\mathbf{y}) = -x_1^2 y_1 - x_2^2 y_2 = -0 y_1 - 0 y_2 = 0.$$

The constraints for $x_1 = 0, x_2 = 0$ have the following form:

$$\begin{cases} y_1 \geq 0, \\ y_1 \leq 5, \\ y_2 \geq 0, \\ y_2 \leq 5, \\ y_1 + 2y_2 \leq 4 \end{cases}.$$

The last system of inequalities has a solution, for example, at $y_1 = 0, y_2 = 0$.

Hence for $x_1 = 0, x_2 = 0$ and as $F(\mathbf{x})$ is not an empty set we have that:

$$\max_{\mathbf{x}\in\mathbb{R}^n} \min_{\mathbf{y}\in F(\mathbf{x})} f(\mathbf{x},\mathbf{y}) = \max_{\mathbf{x}\in\mathbb{R}^n} \min_{\mathbf{y}\in F(\mathbf{x})} 0 = 0.$$

Thus, the point $x_1 = 0, x_2 = 0$ is a solution of the above maximin problem.

Let's check now that RLI-regularity holds. That is that the system of elements $\nabla_y h_i(z_0), i \in I(z_0) \cup I,$ is lineary independent:

$$\nabla_y h_i(z_0) = \begin{bmatrix} \frac{\partial h_1}{\partial y_1} & \frac{\partial h_2}{\partial y_1} & \frac{\partial h_3}{\partial y_1} & \frac{\partial h_4}{\partial y_1} & \frac{\partial h_5}{\partial y_1} \\ \frac{\partial h_1}{\partial y_2} & \frac{\partial h_2}{\partial y_2} & \frac{\partial h_3}{\partial y_2} & \frac{\partial h_4}{\partial y_2} & \frac{\partial h_5}{\partial y_2} \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 & 0 & 1 \\ 0 & 0 & -1 & 1 & 2 \end{bmatrix}.$$

Clearly, rows of the last matrix are linearly independent. Therefore *RLI*-regularity condition is satisfied.

The graph of the minimum function $\Phi(\mathbf{x}) = \min_{\mathbf{y}\in F(\mathbf{x})} f(\mathbf{x},\mathbf{y})$ below illustrates the location of a maximin point and the obtained estimation:

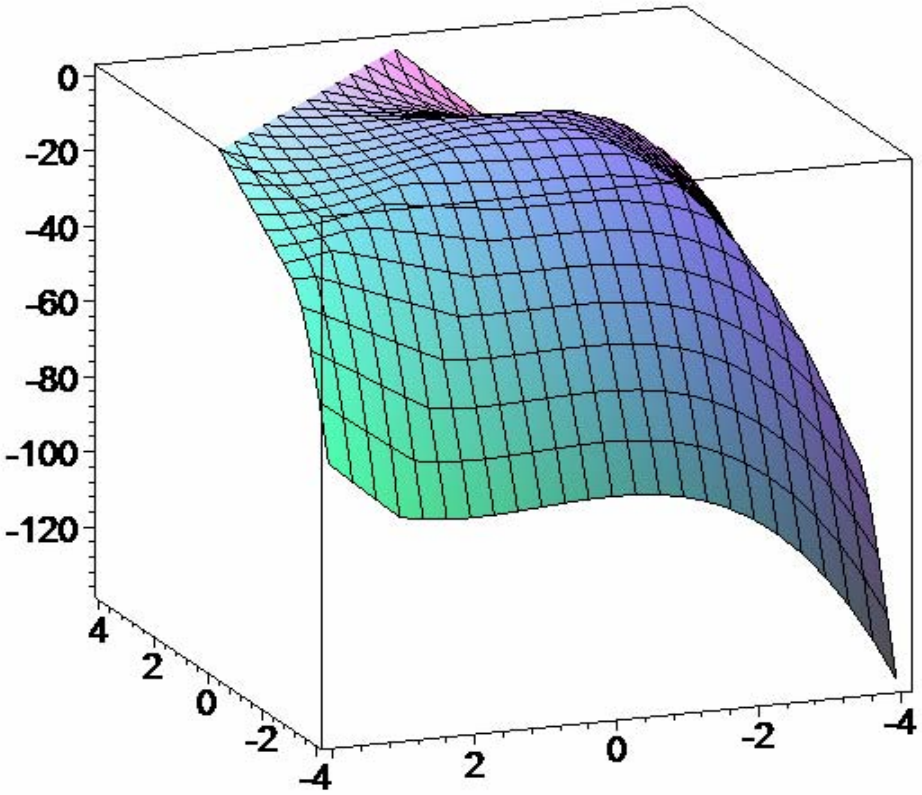

Fig. 1 The rear view of the function $\Phi(\mathbf{x})$ for $x_1 = -4.4,\ x_2 = -4.4$

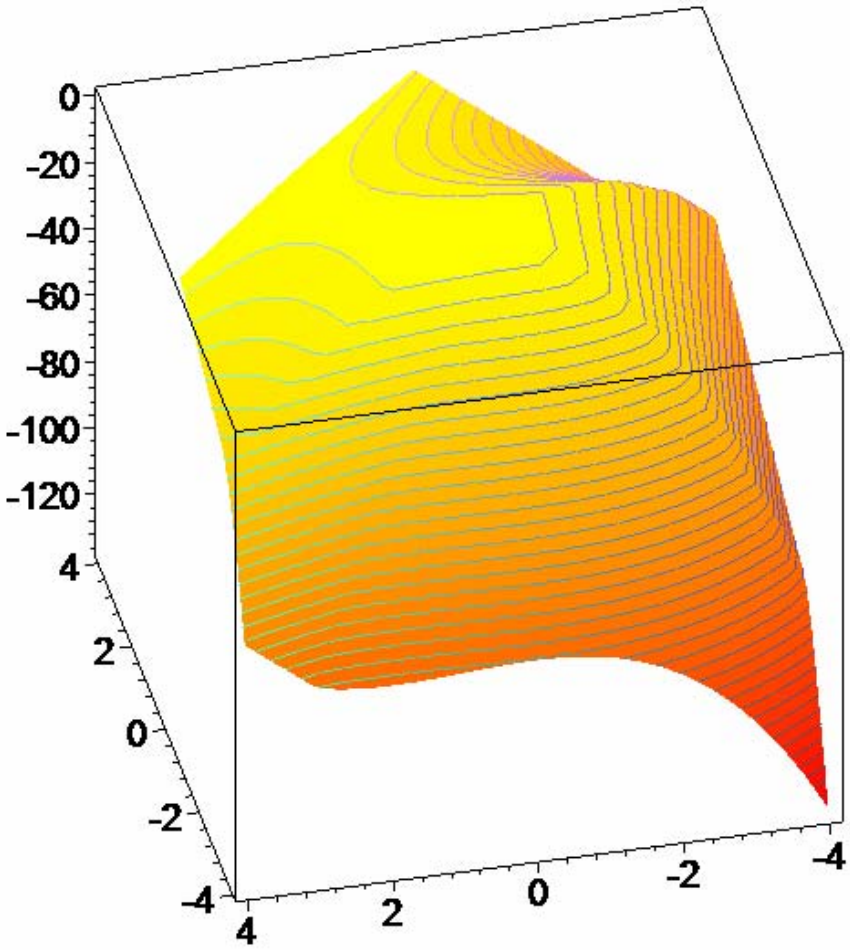

Fig. 2 The rear view of 3-dimensional graph of the function $\Phi(\mathbf{x})$ and its level sets for $x_1 = -4.4,\ x_2 = -4.4$

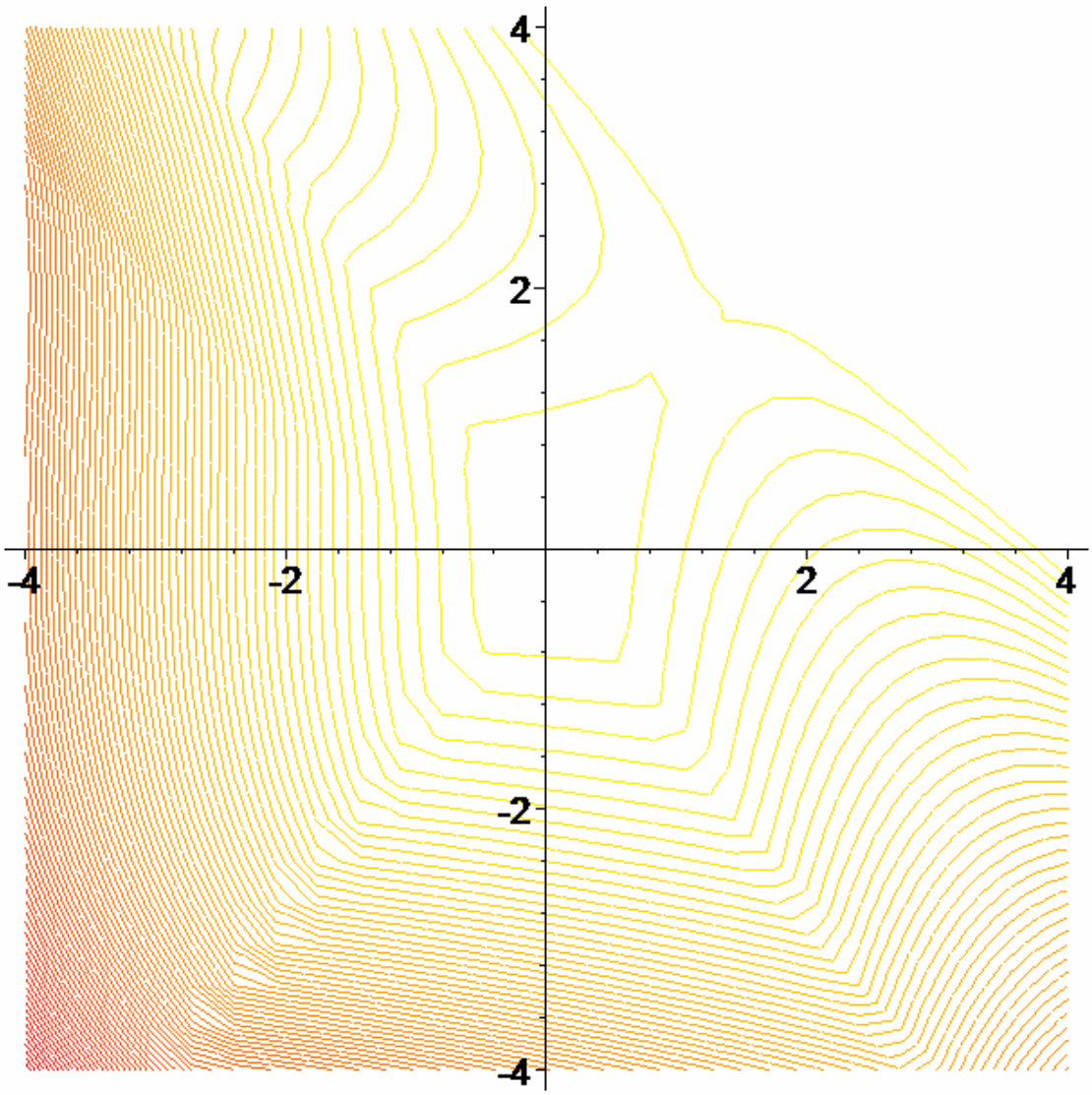

Fig. 3 Level sets of the function $\Phi(\mathbf{x})$ for $x_1 = -4.4,\ x_2 = -4.4$

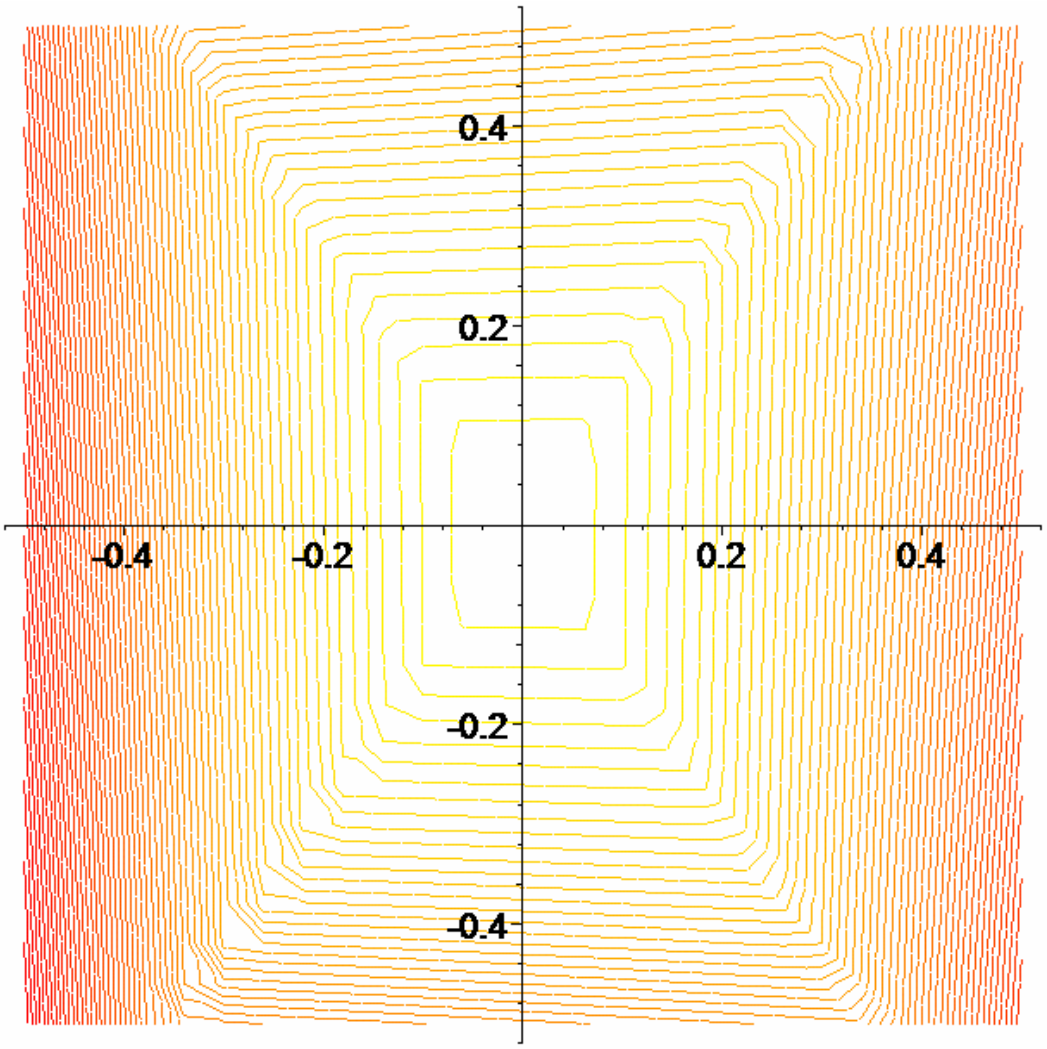

Fig. 4 Level sets of the function $\Phi(\mathbf{x})$ for $x_1 = -0.5..0.5,\ x_2 = -0.5..0.5$

The above graphs and level sets show that the optimum point of the function $\Phi(\mathbf{x})$ is in the neighbourhood of the point $(x_1, x_2) = (0,0)$.

The results of numerical experiments for the specified function are shown below.

Example 1.

In the present example, I have used the following expression for calculating a step size:

$$\beta = 0.1 \cdot |\mathbf{v}|$$

where v is the obtained steepest ascent direction at the current point. The constant 0.1 has been chosen heuristically.

Input data for the algorithm:

1. The required precision of solution

2. Maximum number of allowed iterations. After reaching the maximum number of iterations, the iterative process stops even if the required precision has not been attained.
3. Starting point for the iterative process.

In the present example:

Input data:

1. Initial starting point $(x_1, x_2) = (-12, 3)$.
2. The maximum number of allowed iterations is 50.
3. The required precision of solution $\varepsilon = 0.001$.

The obtained result:

1. The obtained solution $\mathbf{x} = (0.0000625,\ 0.0002395)$.
2. Number of iterations $k = 28$.
3. The obtained value of the objective function $f_k = -.39 \cdot 10^{-6}$.

Graphs below shows the iterative process of the computational experiment.

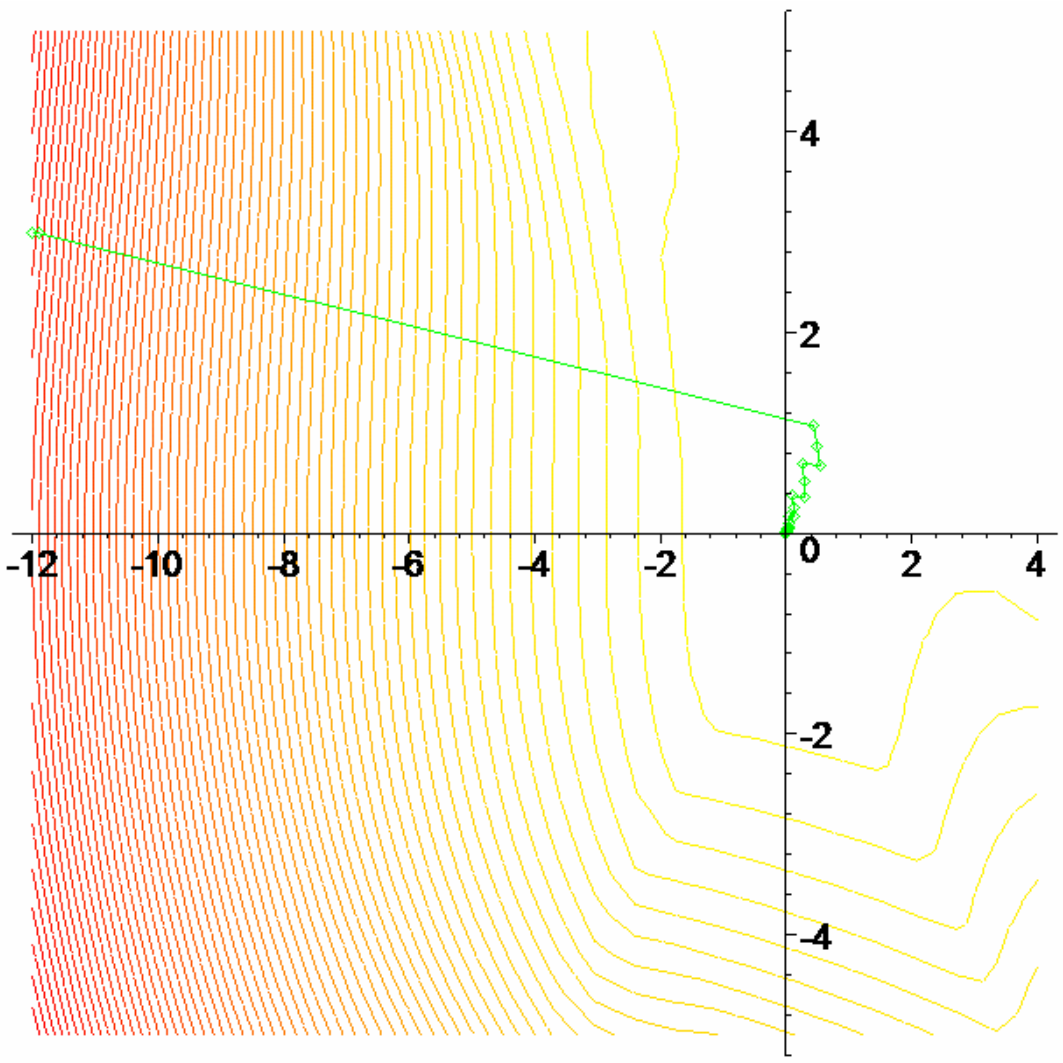

Fig. 5 Trajectory of iterations from the starting point (-12, 3)

Figure 5 shows, that after a few iterations the iterative process enters the neighbourhood of an optimum point. However then the iterative process gets on a kink of an objective function and the convergence decreases strongly. The iterative process in the neighbourhood of an optimum point is shown below.

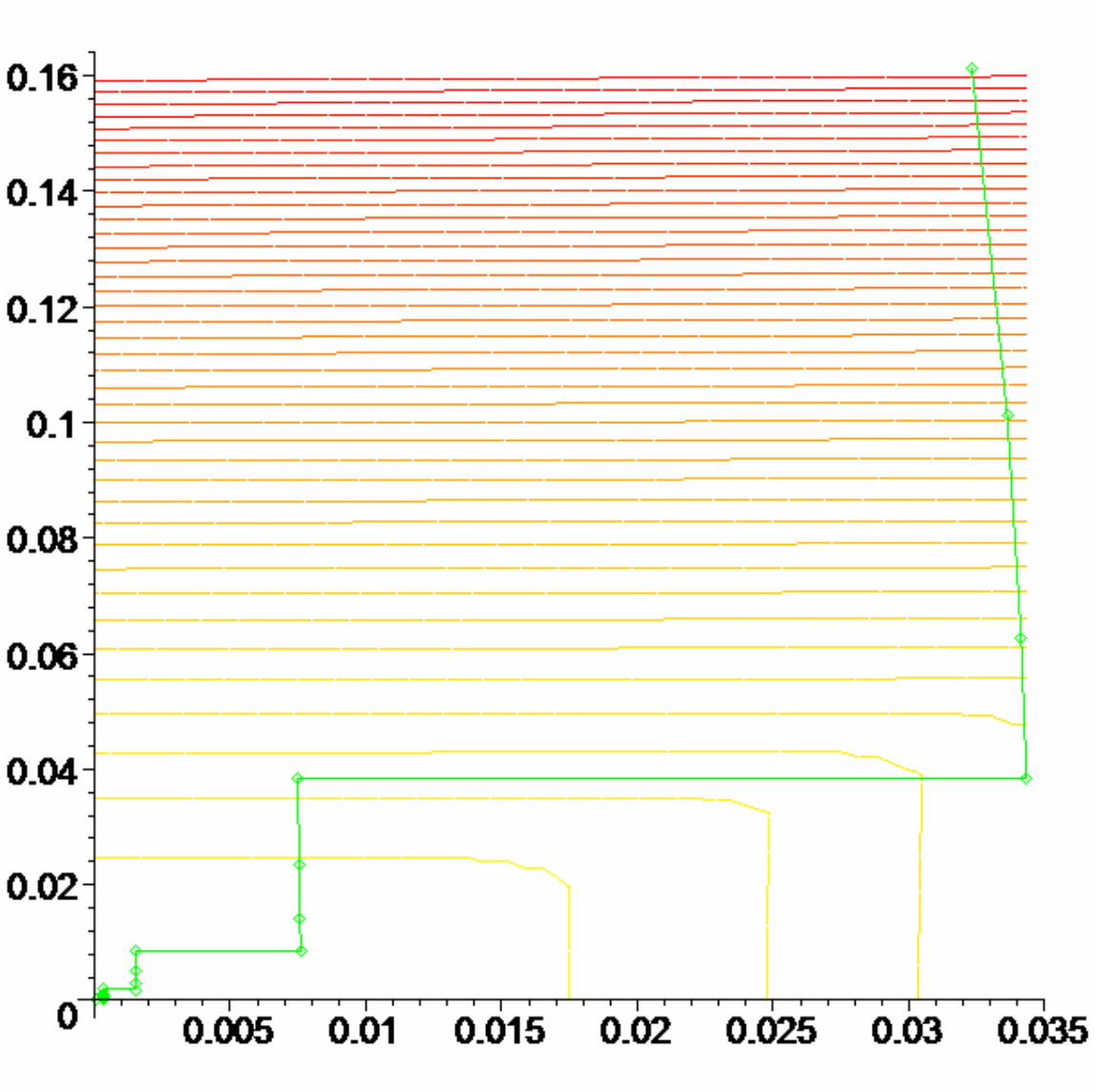

Fig. 6 Trajectory of iterations, starting from the 5th iteration

The above graphs of level sets show that the rate of convergence on a kink of a minimum function significantly decreases. It happens due to the nonsmoothness of the minimum function. In the place shown on the last graph, the minimum function is nonsmooth, and respectively its gradient is a discontinuous function. That is usual methods of smooth optimization are not suitable for the problem. Nevertheless, the algorithm has found an optimal point for a finite number of iteration (28 iterations). And, as I will show further, this result can be significantly improved, by application of a simple modification of a bundle method. The required precision of solution is achieved.

Example 2 (choosing a step size by the method of one-dimensional maximization)

In the present example, I have used a method of one-dimensional maximization described in the previous chapter (dichotomy method) for choosing a step size.

Input data:

1. Initial starting point $(x_1, x_2) = (-12, 3)$
2. Required precision $\varepsilon = 0.001$.
3. The maximum number of allowed iterations is 50.

The obtained result:

1. The obtained solution $\mathbf{x} = (.35007,\ .49507)$.
2. The number of iterations is 50.
3. The obtained value of the objective function $f_k = -.38662$.

Graphs below show the iterative process of the computational experiment.

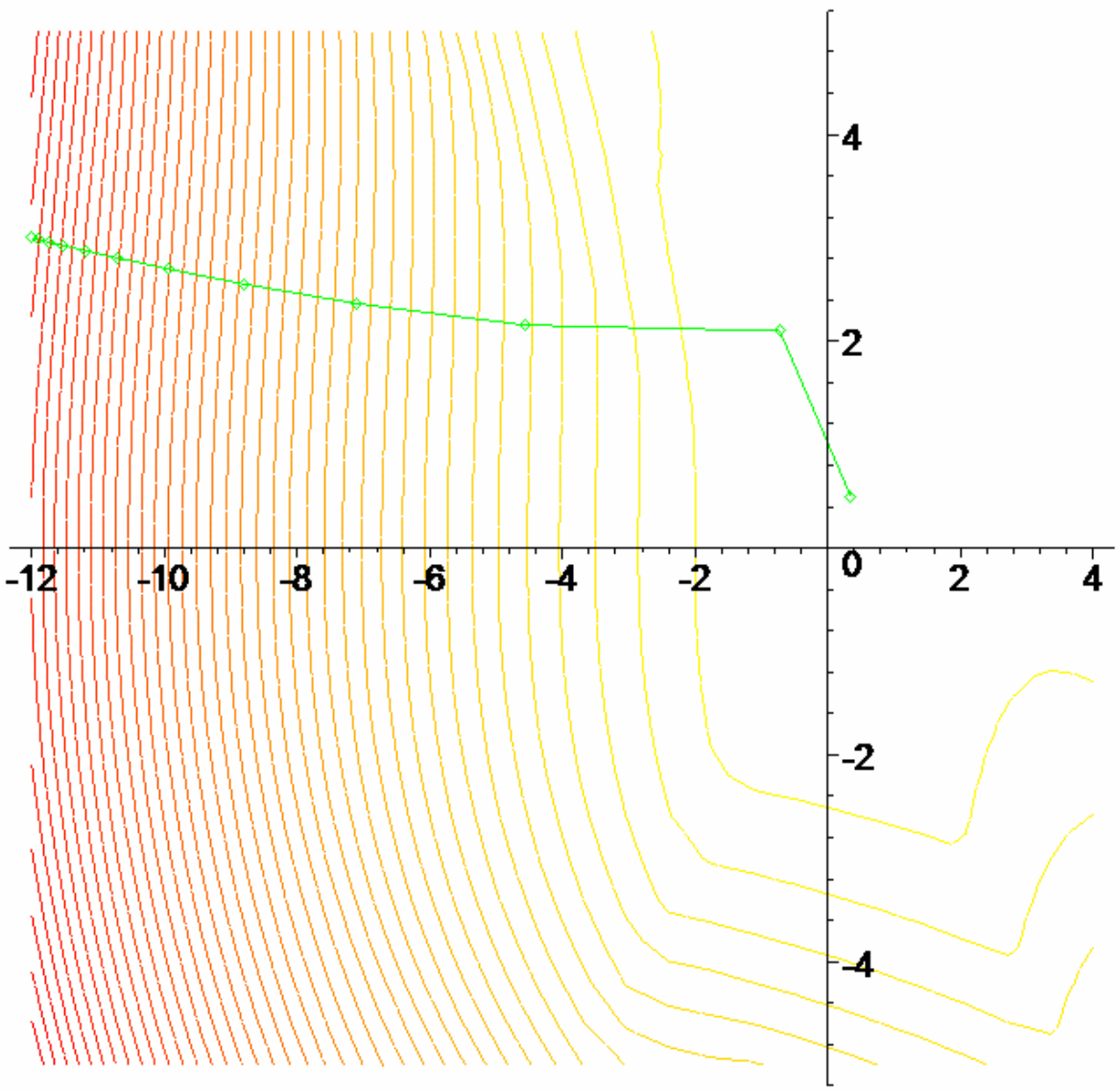

Fig. 7 The trajectory of iterations from the starting point (-12,3)

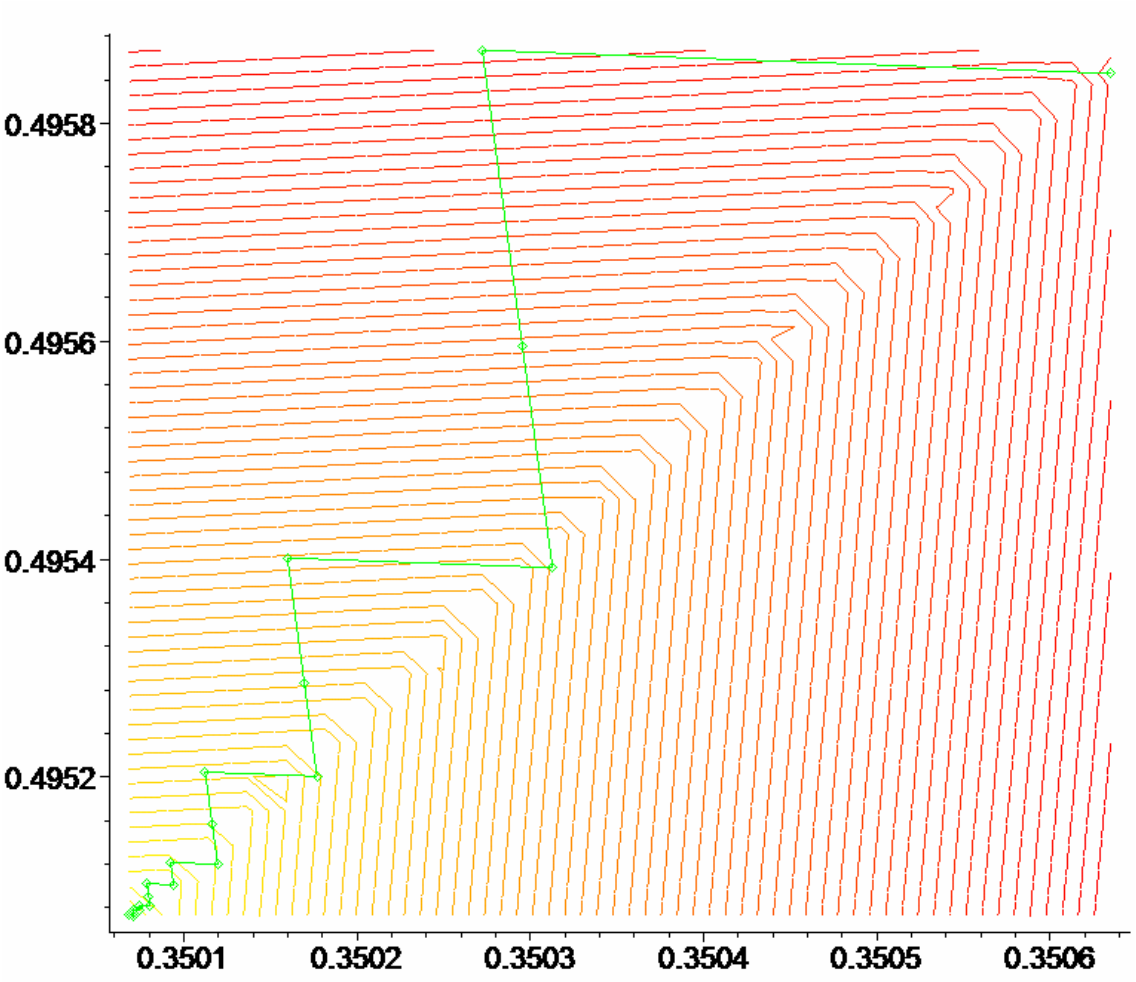

Fig. 8 Trajectory of iterations, from 11th iteration

The above graphs show that the extremum point has not been reached. The algorithm has got caught in an endless loop at point $\mathbf{x} = (.3501, .49507)$. Thus, the method of 1-dimensional maximization is not suitable for calculating an iteration step size for nonsmooth functions.

Thus using the one-dimensional maximisation method, the iterative process has stopped on a kink (in a place where function nonsmooth) and has got caught in an endless loop. The reason is that on a kink, in the direction of steepest ascent, we are already in the neighbourhood of an optimum point. Therefore the obtained step size is close to 0.

The above graphs of level sets show that convergence decreases far from an extremum point and an iterative process gets caught in an endless loop on a kink of the minimum function. It occurs due to the nonsmoothness of the minimum function. And respectively the gradient of the minimum function has a rupture in the current point.

Example 3.

In the present example I have used a constant step size.

Input data:

1. Initial starting point $(x_1, x_2) = (-12, 3)$
2. Required precision $\varepsilon = 0.001$.
3. The maximum number of allowed iterations is 50
4. Step size $\beta = 0.4$.

Results of the computational experiment:

1. The obtained solution $\mathbf{x} = (.074678, .098967)$.
2. The number of iterations $k = 50$.
3. The obtained value of the objective function $f_k = -.4483632057$.

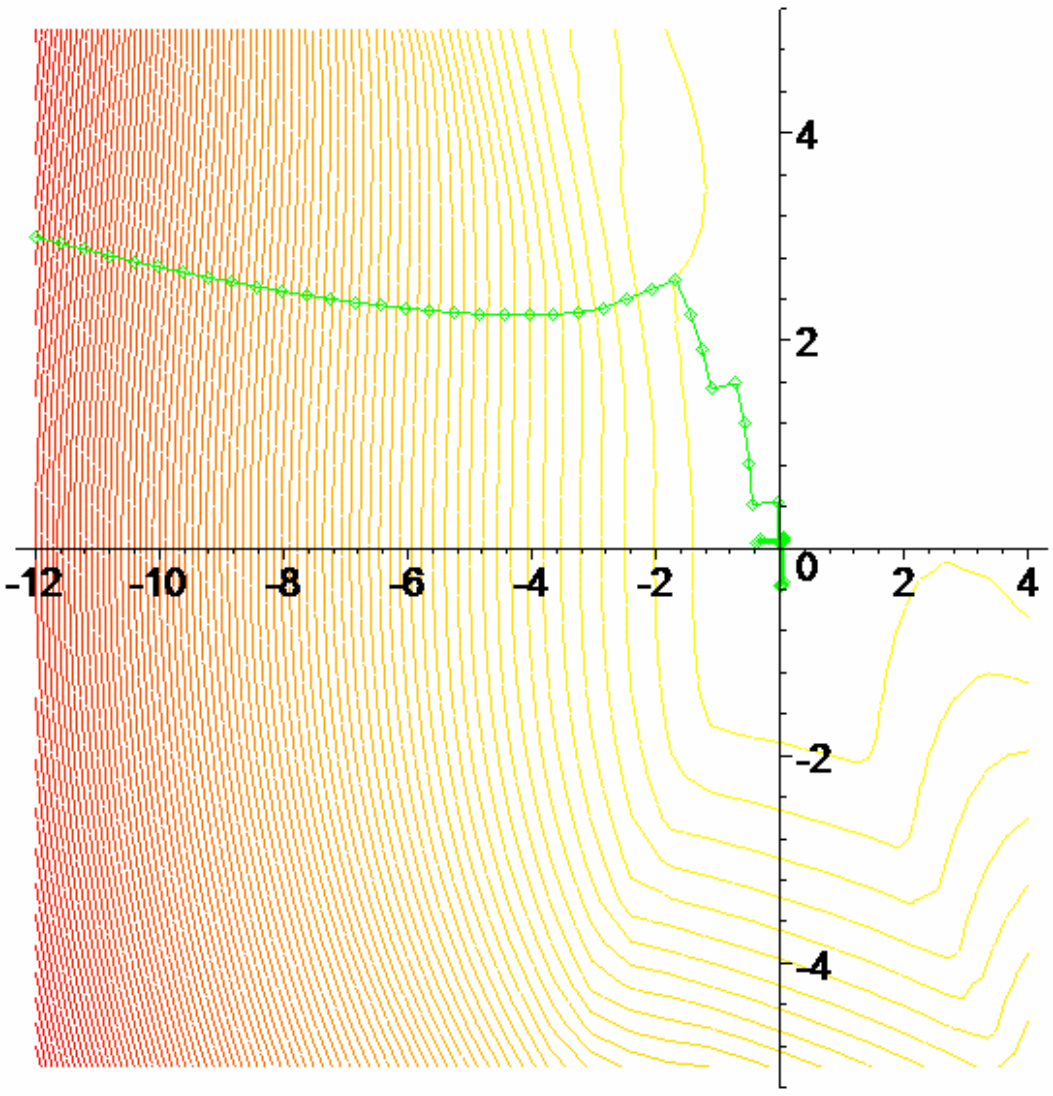

Fig. 9 The trajectory of iterations from the starting point (-12,3)

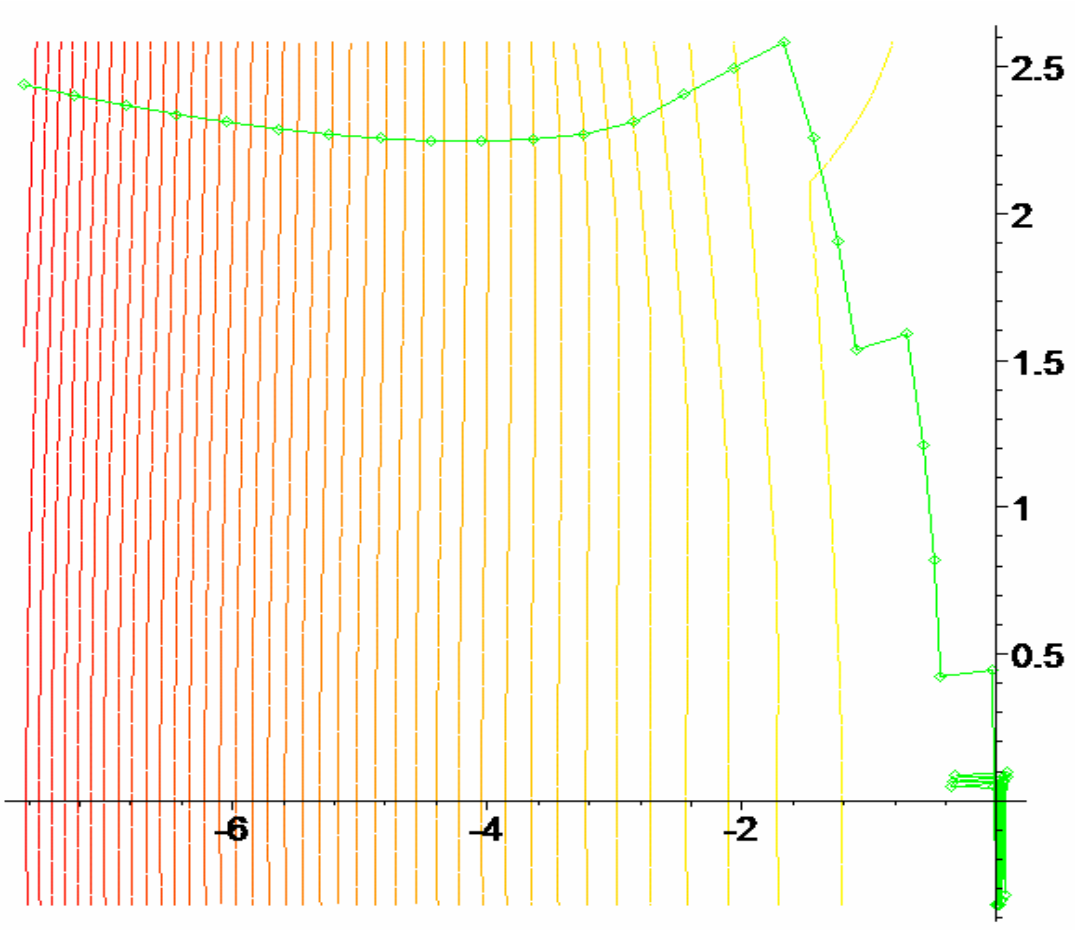

Fig. 10 Trajectory of iterations, starting from the 11th iteration

The iterative process has stopped because it reached the maximum number of iterations. After entering the neighbourhood of the optimum point, the iterative process has got caught in an endless loop.

Thus, using a fixed step size method, the required precision has not been reached. In general, a precision of solution directly depends on the step size in this method. Also the given method of choosing a step size requires a lot of iterations, and at the same time it does not guarantee that the required precision of a solution will be reached. However, the obvious advantage of the given method of choosing a step size is its simplicity. Also there is no need to calculate additional values of the objective function.

Example 4.

In the present example I have applied the modification of bundle method described in the last chapter together with the developed algorithm of finding a steepest ascent direction. As I have shown above, the rate of convergence strongly decreases when the iterative process nears a kink of the objective function. In the neighbourhood of a kink, a gradient of the objective function has

a rupture. Therefore even an arbitrary small change of coordinates involves a finite change of the gradient. The numerical experiment has shown that the given modification of choosing an ascent direction has allowed to improve the stability of iterative process considerably.

Input data:

1. Initial starting point $(x_1, x_2) = (-12, 3)$.
2. Required precision of solution $\varepsilon = 0.001$
3. The maximum number of allowed iterations is 50

Results of computational experiment:

1. The number of iterations $k = 14$.
2. The obtained solution $\mathbf{x} = (.213 \cdot 10^{-4}, .5485 \cdot 10^{-5})$.
3. The obtained value of the objective function $f_k = -.10602 \cdot 10^{-6}$.

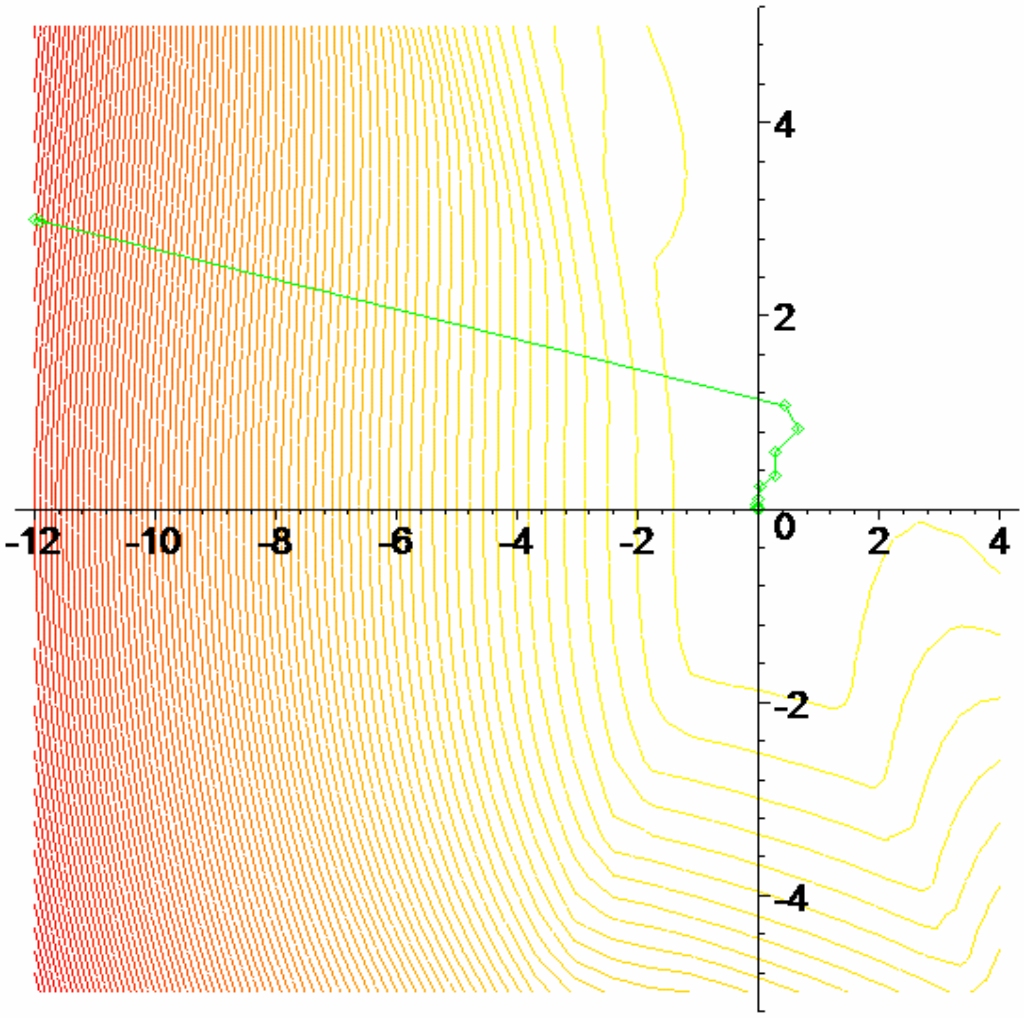

Fig. 2.11 The trajectory of iterations from the starting point (-12, 3)

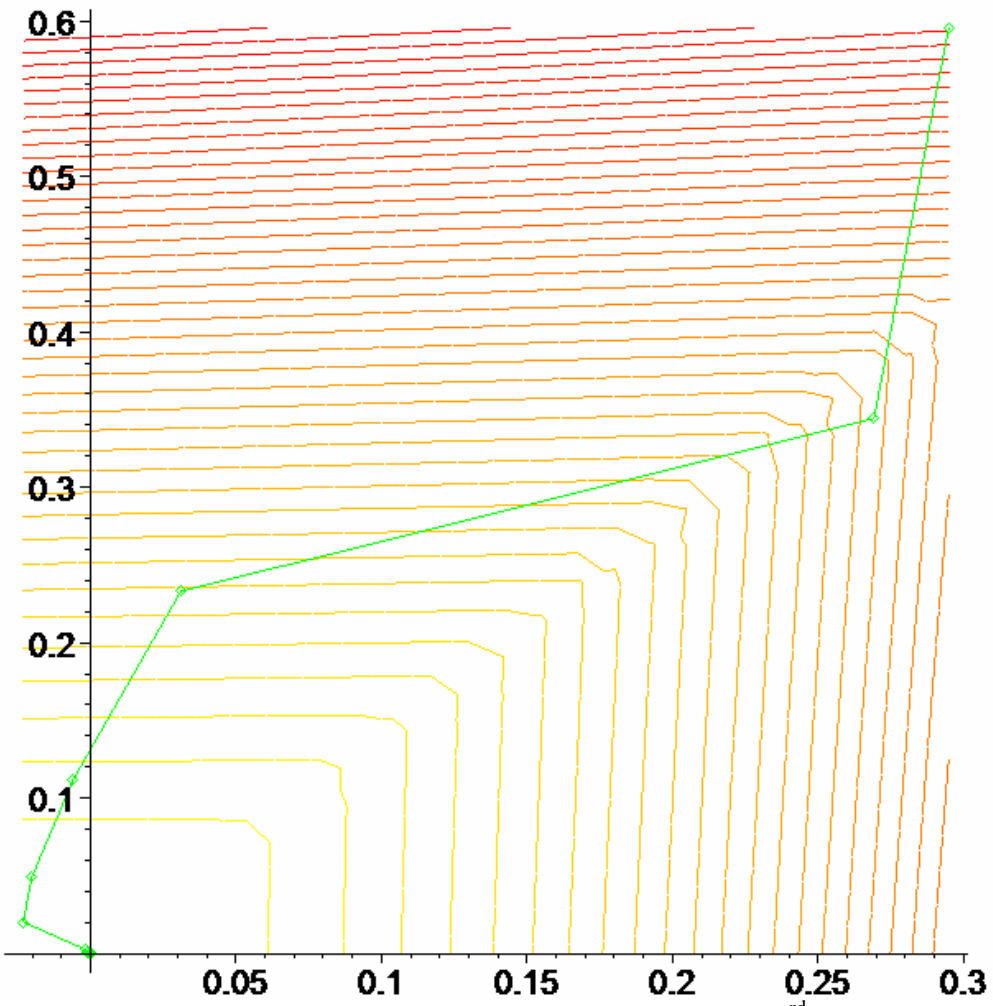

Fig. 2.12 The trajectory of iterations, starting from the 3rd iteration

In the present example the algorithm has stopped after 14 iterations. The required precision has been reached.

Using the given method of choosing a step size and an ascent direction, the algorithm required the least number of iterations (14 iterations). The above graphs show that the application of ideas of a bundle method has allowed reaching a much better stability and rate of convergence, than in the previous computational experiments. It is achieved by taking into account the information of the ascent directions of two previous iterations.

The number of required iterations is 2 times less, than in the best of obtained above results (28 iterations for a similar method without application of a bundle method). Further I intend to implement an advanced version of this method, which will also take into account a distance to the previous points and use the information of the previous directions of a greater number of iterations.

3.2. Analysis of results of the computational study

In chapter 2 I developed an algorithm, and a program that implements the method described above. The computational study in this chapter shows that the developed program allows us to calculate a maximin point in the specified examples.

In the first method of choosing a step size the algorithm stopped after 28 iterations and the required precision of the solution has been reached. Moreover, the value of the objective function in the obtained optimal point differs from the optimum by less than 10^{-5}. However, its rate of convergence strongly decreases when the iterative process nears the kink of the objective function.

A modification of the bundle method together with the developed algorithm of finding a direction of the steepest ascent showed the best result. The required precision of the solution has been reached after 14 iterations only. This result is twice as good as the best result of other methods of choosing a step size and an ascent direction. Moreover, the value of the objective function at the obtained point differs from the optimum only by 10^{-6}. The computational study has shown that the given method is stable in the area where the objective function is nonsmooth.

The iterative process with a constant step size stopped because it exceeded the maximum number of iterations. After entering the neighbourhood of an optimum point, the iterative process got caught in an endless loop. Thus, using this method of choosing a step size, the required precision was not reached, even though the limit of iterations was exceeded. Moreover, in this method the precision of the solution directly depends on the step size. The given method of choosing a step size also requires too many iterations.

The worst result was shown by the method of one-dimensional minimization. The iterative process got caught in an endless loop rather far from an optimum point and the allowed number of iterations was exceeded. Thus the obtained value of the objective function significantly differs from the optimum value of the objective function.

CONCLUSION

The goal of the present work was to develop an algorithm for the maximin problem with coupled variables. To this end I have studied methods of nonsmooth optimization in chapter 1. In chapter 2 I have developed a method of decomposition of the maximin problem with coupled constraints, studied the arising problems and methods of their solution. In chapter 2 I have also obtained and proved the necessary condition of local maximum of function $\Phi(\mathbf{x})$.

The practical results of the present work are the implementation and testing of the algorithm for the maximin problem with coupled constraints using the system of computer algebra Maple. In the course of the computational experiment, I have shown that the developed algorithm makes it possible to calculate maximin points with a given precision.

The obtained computational results correspond to the graph and level curves of the objective function. I have developed several methods of choosing a step size and methods of choosing of ascent direction. I have also analyzed and compared the advantages and shortcomings of each method, showing that the modification of bundle method of choosing an ascent direction gives the best result.

For most of the developed methods of choosing of a step size I have shown their efficiency and ability to calculate maximin points with a given precision in the numerical experiment.

I have justified the necessity of using of methods of nonsmooth optimization in the numerical experiment. I have also developed and tested an original synthesis of the developed algorithm with a modification of bundle method. The goal of the present work is fulfilled.

In future I am going to generalize the method to an objective function *f* nonlinear in both variables and nonlinear constraints. I will also apply the existing nonsmooth optimization methods to increase the speed of convergence, such as the bundle method, and will investigate the convergence of the algorithm for different types of functions. Another potential topic is to generalize the method for a more general Bilevel programming problems [3].

LIST OF PUBLICATIONS

1. Charnyi, S.G. “On Maximin problem with coupled constraints”. X Belorussian mathematical conference: Reports of international scientific conference, Minsk, November 3-7, 2008. – Volume 3. – Institute of Mathematics of National Academy of Sciences of Belarus, 2008, p. 122.
2. Charnyi, S.G., L.I. Minchenko “Solving maximin problem by decomposition to smooth and nonsmooth problems”. Neural Networks and Artificial Intelligence (ICNNAI-2008) //Proceedings of the Fifth International Conference (27-30 May, 2008, Minsk, Belarus). – Minsk: Propilei, 2008, pp. 298-301.
3. Charnyi, S.G. “Solving Maximin problem by decomposition to smooth and nonsmooth problems”. New mathematical methods and computer techniques in design, production and scientific research, X republic scientific conference of post-graduate students. – Gomel State University, 2007, pp. 181-182.
4. Minchenko, L.I., S.G. Charnyi “Solving Maximin problem by decomposition to smooth and nonsmooth problems”. Hardware for information security: reports of IV Belarusian-Russian scientific and technical conference, BSUIR, Minsk, May 28 – June 1, 2007, pp. 39-40.

REFERENCES

[1] Astrovski, A.I. and S.K. Korzhenevich. Study of Bilinear Minimax Problems: Theory and Computing Experiment. 1990. - 60 p. – (Preprint / Academy of Sciences, BSSR. Institute of Mathematics; №43 (443)).

[2] Danskin, John M.. The Theory of Max-Min. Springer-Verlag New York, Inc., 1967.

[3] Dempe, S. Foundations of Bilevel Programming, Springer, 2002.

[4] Demyanov, V.F. and V.N. Malozemov. Introduction to Minimax. M.:Science, 1972.

[5] Du, Ding-Zhu and Panos M. Pardalos (editors). Minimax and Applications. Kluwer Academic Publishers, 1995.

[6] Hiriart-Urruty, J.-B. and C. Lemarechal. Convex Analysis and Minimization Algorithms, volume I, Springer, Berlin, Heidelberg, New York, 1993.

[7] Hiriart-Urruty, J.-B. and C. Lemarechal. Convex Analysis and Minimization Algorithms, volume II, Springer, Berlin, Heidelberg, New York, 1993.

[8] Karmanov, V.G. Matematicheskoe programmirovanie. M: Science, 1975 (in Russian).

[9] Minchenko, L.I. O.F. Borisenko and S.P. Grishai. Multivalued Analysis and Perturbed Problems of Nonlinear Programming. Science and Engineering, 1993. p. 197. (in Russian)

[10] Rustem, B. and M. Howe. Algorithms for Worst-Case Design and Applications to Risk Management, Princeton University Press, 2002.

[11] Strekalovskiy, A.S. Elements of Nonconvex Optimization, Novosibirsk, 2003.

APPENDIX. CODE OF THE ALGORITHM

```
> restart;
debug level
> printlevel:=1;

maximum number of iterations
> iterNum:=50;

methods of choosing a step size
1)fixed step size. Requires a lot of iterations.
2)linear search along the direction v. It converges fast, but requires a lot of computations of
function value along the direction v.
3)step = c*abs(v[k]), where c - constant (currently the best value is about 0,1)
4)step beta[k]=c*beta[k-1], where c-constant. 1<c<2
4)method of unival minimization. implemented (function univalMinimization). it is used in the
3rd  case(function stepByDoubling).

> stepMethod:=3;
precision
> epsilon:=0.01;
initial step size
> beta_init:=0.1;
initial point
> xx[0] := <-12,3>;
>
> #creating bounderes for level and limit functions graphs;
> dimX:=2;#dimension of vector x
> dimY:=2;#dimension of vector  y
> constrNum:=5;#number of constraints
objective function f(x,y), linear in y
> f:=-x[1]^2*y[1]-x[2]^2*y[2];
> #f:=-x[1]^4*y[1]-x[2]^4*y[2];
> #f:=-x[1]*y[1]-x[2]*y[2];
```

```
creating vectors x and y
> xvector:=seq(x[i],i=1..dimX);yvector:=seq(y[i],i=1..dimY);
constraints h_i
> h[1]:=-y[1]:h[2]:=y[1]-5:
> h[3]:=-y[2]:h[4]:=y[2]-5:
> h[5]:=x[1]+x[2]+y[1]+2*y[2]-4:
>
> if xx[0][1]>0 then leftBound:=0;rightBound:=xx[0][1] end if;
>
> with(linalg):
> with(Optimization):
> with(simplex,feasible):

Execute the commands below to declare the external subroutines from GANSO library.
> #GANSOPATH:="mwrap_ganso.dll":
>    # if you need to provide full path, use
GANSOPATH:="c:/work/ganso/maple/mwrap_ganso.dll":
> #Mindfbm0 := define_external('MWRAP_MinimizeDFBM_0','MAPLE',
LIB=GANSOPATH):
> #Mindfbm := define_external('MWRAP_MinimizeDFBM','MAPLE', LIB=GANSOPATH):
> #Minecam0 := define_external('MWRAP_MinimizeECAM_0','MAPLE',
LIB=GANSOPATH):
> #Minecam := define_external('MWRAP_MinimizeECAM','MAPLE', LIB=GANSOPATH):
> #Minecamdfbm0 := define_external('MWRAP_MinimizeECAMDFBM_0','MAPLE',
LIB=GANSOPATH):
> #Minecamdfbm := define_external('MWRAP_MinimizeECAMDFBM','MAPLE',
LIB=GANSOPATH):
> #Mindfbmecam0 := define_external('MWRAP_MinimizeDFBMECAM_0','MAPLE',
LIB=GANSOPATH):
> #Mindfbmecam := define_external('MWRAP_MinimizeDFBMECAM','MAPLE',
LIB=GANSOPATH):
> #Mindso0 := define_external('MWRAP_MinimizeDSO_0','MAPLE', LIB=GANSOPATH):
> #Mindso := define_external('MWRAP_MinimizeDSO','MAPLE', LIB=GANSOPATH):
```

```
> #Minecamdso0 := define_external('MWRAP_MinimizeECAMDSO_0','MAPLE',
LIB=GANSOPATH):
> #Minecamdso := define_external('MWRAP_MinimizeECAMDSO','MAPLE',
LIB=GANSOPATH):
> #Miniterativedso0 := define_external('MWRAP_MinimizeIterativeDSO_0','MAPLE',
LIB=GANSOPATH):
> #Miniterativedso := define_external('MWRAP_MinimizeIterativeDSO','MAPLE',
LIB=GANSOPATH):
> #Minrandomstart0 := define_external('MWRAP_MinimizeRandomStart_0','MAPLE',
LIB=GANSOPATH):
> #Minrandomstart := define_external('MWRAP_MinimizeRandomStart','MAPLE',
LIB=GANSOPATH):
>
> #MindfbmHF := define_external('MWRAP_HFMinimizeDFBM','MAPLE',
LIB=GANSOPATH):
> #Minecam0HF := define_external('MWRAP_HFMinimizeECAM_0','MAPLE',
LIB=GANSOPATH):
> #MinecamHF := define_external('MWRAP_HFMinimizeECAM','MAPLE',
LIB=GANSOPATH):
> #Minecamdfbm0HF := define_external('MWRAP_HFMinimizeECAMDFBM_0','MAPLE',
LIB=GANSOPATH):
> #MinecamdfbmHF := define_external('MWRAP_HFMinimizeECAMDFBM','MAPLE',
LIB=GANSOPATH):
> #Mindfbmecam0HF := define_external('MWRAP_HFMinimizeDFBMECAM_0','MAPLE',
LIB=GANSOPATH):
> #MindfbmecamHF := define_external('MWRAP_HFMinimizeDFBMECAM','MAPLE',
LIB=GANSOPATH):
> #Mindso0HF := define_external('MWRAP_HFMinimizeDSO_0','MAPLE',
LIB=GANSOPATH):
> #MindsoHF := define_external('MWRAP_HFMinimizeDSO','MAPLE', LIB=GANSOPATH):
> #Minecamdso0HF := define_external('MWRAP_HFMinimizeECAMDSO_0','MAPLE',
LIB=GANSOPATH):
> #MinecamdsoHF := define_external('MWRAP_HFMinimizeECAMDSO','MAPLE',
LIB=GANSOPATH):
```

```
> #Miniterativedso0HF := define_external('MWRAP_HFMinimizeIterativeDSO_0','MAPLE',
LIB=GANSOPATH):
> #MiniterativedsoHF := define_external('MWRAP_HFMinimizeIterativeDSO','MAPLE',
LIB=GANSOPATH):
> #Minrandomstart0HF := define_external('MWRAP_HFMinimizeRandomStart_0','MAPLE',
LIB=GANSOPATH):
> #MinrandomstartHF := define_external('MWRAP_HFMinimizeRandomStart','MAPLE',
LIB=GANSOPATH):
>
> with(VectorCalculus):
> #generating the sequence of constraints
> constr:=seq(h[i],i=1..constrNum);
> H := Jacobian([constr],[xvector]); %?;
creating the constraints matrix of the polyhedron of Lagrange multipliers A. Its a Jacobian of
constraints in y
> A:=Jacobian([constr],[yvector]);
>
> #with(Student[MultivariateCalculus]):
> #p:=Gradient(x^2+y^2,[x,y]=[0,1])[1];
> #p:=Gradient(f,[x[1],x[2]]);
> p:=Gradient(f,[xvector]);
>
> with(LinearAlgebra):
>
creating the objective quadratic function and constraints Ax=b

> b:=-Gradient(f,[yvector]);
> Transpose(H); B := Multiply(H,Transpose(H));
> Lambda := `<,>`(seq(lambda[i],i = 1 .. constrNum)); %?;
> HX:=Multiply(Transpose(H),Lambda);
obtain the linear part of the objective quadratic function 2p*H*Lambda
> linearPart := 2*Multiply(Transpose(p), HX);
creating the maximizing quadratic function Lambda*(H*)*H*Lambda
> quadraticF := Multiply(Multiply(Transpose(Lambda), B), Lambda)+linearPart; 1;
> quadraticF;
```

```
>
```

The function is used for global optimization. Returns the value of the maximizing quadratic function at point Lambda. n - dimension of vector lambda

```
> #ff := proc( n, lambda )
> ff := proc( lambda )
>  local s;
>
> #print(`pInXY=`,pInXY);
> #creating the objective quadratic function
> HX:=Multiply(Transpose(Hsimple),Transpose(lambda) );
> #print(`HX calculation passed`);
> linearPart := 2*Multiply(Transpose(pInXY), HX);
>
> #print(`linearPart calculation passed`);
>
> #quadraticFValue := Multiply(Multiply(Transpose(lambda), B), lambda)+linearPart;
> t1:=Multiply(lambda, B);
> #print(`t1 passed, t1=`,t1);
>
> #print(`lambda=`,lambda);
> t2:=Multiply(t1, Transpose(lambda));
>
> #print(`t2 passed, t2=`,t2);
> qFValue := t2+linearPart;
>
> #print(`Entering ff, lambda[1]=`,lambda[1],`lambda[2]=`,lambda[2]);
> #print(`in ff, quadraticFValue=`,qFValue);
>
> s:=eval(qFValue);
> #print(`s=`,s);
>   s  #this last value is what is returned
> end proc:
>
>
> #given x=(x1,x2) the function returns minimum of f in vector y
```

```
> f_min_y:= proc(x1,x2)
>    f_in_X:=eval(f,{x[1]=x1,x[2]=x2});
>    #print(`in f_min_y. f_in_X=`,f_in_X);
>    yy:=LPSolve(f_in_X,{seq(eval(h[i],{x[1]=x1,x[2]=x2})<=0,i=1..constrNum)} );
>    #print(`yy[2]=`,yy[2]);
>    f_k:=eval(f_in_X,{y[1]=rhs(yy[2][1]),y[2]=rhs(yy[2][2])});
>    f_k:=eval(f_in_X,yy[2]);
>  end proc:
>
> #auxiliary unival function used in some functions for step calculating
> #returns f( x_k+step*v, y(x_k+step*v) ) given step size
> linearF := proc(step)
>   #print(`in linearF, xx[k]=`,xx[k],`v[k]=`,v[k],`xx[k][2]=`,xx[k][2]);
>   #x1:=xx[k][1]+v[k][1]*step;#!change to the n dimentional case here #(x:=xx[k]+v[k]*step)
>   #x2:=xx[k][2]+v[k][2]*step;
>   trialPoint:=xx[k]+step*v[k];
>  #print(`in linearF, x1=`,x1,`x2=`,x2);
>  #print(`in linearF, trialPoint=`,trialPoint);
>
>   #f:=-f_min_y(x1,x2);
>   f:=-f_min_y(seq(trialPoint[i],i=1..dimX));
>
>  #print(`in linearF. f=`,f);
>  f;
>  end proc:
>
```

test function for unival minimizaiton procedure

```
> testF := proc(x)
>   -x^2;
> end proc:
> testF(-1);
```

the function does one-dimensional maximization given the interval [a,b]. We suppost that there is only 1 maximum point in the given interval. Implemented dichotomy algorithm. For maximizaiton it uses the auxiliary funciton linearF(step). Returns the step size, whic maximizes the objective function.

```
> univalMaximization := proc(aa,bb)
>   delta:=epsilon/100;#the distance from new points to the middle of the interval
>   print(`In univalMaximization a=`,aa,`b=`,bb);
>   #searching for the middle of the interval
> a:=aa;b:=bb;
> i:=1; length:=abs(b-a);
> while (length>epsilon) do
>   middle:=(a+b)/2; #print(`middle=`,middle);
>   l[i]:=middle-delta; #print(`l[i]=`,l[i]);#point to the left of the middle of interval
>   r[i]:=middle+delta; #print(`r[i]=`,r[i]);#point to the right of the middle of interval
>
>   #comparing the function values in obtained point and choosing the next interval
>   #print(`testF(l[i])=`,testF(l[i]),`testF(r[i])=`,testF(r[i]));
>   #print(testF(l[i])>=testF(r[i]));
> #if ( testF(l[i])>=testF(r[i]) ) then b:=r[i] else a:=l[i]; end if;#for testing
> if ( linearF(l[i])<=linearF(r[i]) ) then b:=r[i] else a:=l[i]; end if;
>
>   print(`the maximum point is inside the interval [`,a,`,`,b,`]`);
>   length:= abs(b-a);
>   print(`the length of the interval = `, length);
>   i:=i+1;#next iteration
> end do;
>  optimalPoint:=(b+a)/2; print(`optimalPoint=`,optimalPoint);
>  optimalPoint;
> end proc:#end of 1-dimensional maximization
> opt:=univalMaximization(-102,111):
> opt;


> #function determines the best step size in the given direction;
> #v - is the found descent direction
> #implemented the following methods
> #1) method of 1-dimensional maximization
> #2) linear search along the direction v (implemented here)
> #3) doubling the the step size and diving by 2
```

```
> #4) step size=0,1*lenght of vector v
> stepSize := proc(v)
>   currentX:=xx[k];
>   initStep:=0.1;
>   print(`Entering stepSize. currentX=`,currentX,`v=`,v);
>   i:=0;
>   #checking whether there is an increase in function value at the last iteration
>   while (i<2) or (f_step[i-2]>f_step[i-1]) do
>     #obtaing the next point (just for debug)
>     #x_v[i]:=currentX+initStep*i*v;
>     #print(`x_v[i]=`,x_v[i]);
>     step:=initStep*i;#obtaining the next step size
>     #print(`step=`,step);
>     f_step[i]:=linearF(step);#function value given the step size
>     #print(`f_step[i]=`,f_step[i],`i=`,i);
>     i:=i+1;
>   end do;#end of linear search
>   print(`step found. i=`,i,`f_step[i-2]=`,f_step[i-2],`f_step[i-1]=`,f_step[i-1]);
>
>   steps:=i-2;#number of steps till the maximum function value
>   #avoiding looping
>   if steps=0 then steps:=1 end if;
>   beta:=initStep*steps;#obtained step size
> end proc:
> #the method of doubling a step size depending on the function value
> #multiplying the previous step size by c>1.
> #if the function value in the next point is less then currect, then try 1/2 step size etc
> #at the final step (obtained interval) we have to run 1-dimensional search
> #searching step size as beta[k]=c*beta[k-1], where c-constanst. 1<c<2
> stepSizeByDoubling := proc()
>   print(`In stepSizeByDoubling`);
>   #multiplying the last step size by c=const
>   step:=1.5*beta[k-1]; #original.tested
>   print(`just before while. linearF(0)=`,linearF(0));
>   print(`linearF(step)=`,linearF(step));
```

```
>
> if k>6 then step:=univalMaximization(0,step); end if;#break;#trying 1-dimensional
maximization
>
> #  if k>0 then
> #   i:=0;
>    #checking whether there is an increase of f(x,y) at  the current step. If not, dividing the step
size by 2
> #   while(linearF(0)<linearF(step)) do
>      #print(`step=`,step,`linearF(step)=`,linearF(step));
>                                                                                #i:=i+1;
>      #step:=step/2; #instead of division by 2 we can use 1-dimensional maximization
> #   end do;
> #   print(`after while.`,`linearF(step)=`,linearF(step));
>    print(`step=`,step,`i=`,i);
> #  end if;#! here we need to run linear search!
>   step;
> end proc:
>
>
> # test the objective function: declare some vector x and evaluate ff(x)
> xtest:=Vector(1..2,[5,6]):
> eval(ff(<1,4>));
> Lo:=Vector(1..2,datatype=float[8],[seq(0,i=1..2)]):  # large values (greater 1e20) mean no box
constraints
> Up:=Vector(1..2,datatype=float[8],[seq(2e30,i=1..2)]):
> dim:=2; iter:=100: val:=1.0:
> basic:=Vector(1..constrNum,datatype=integer[4],[0,1]): # these are needed for constrained
optimization with
>         # linear equality constraints. These values refer to "basic" variables
>         # if basic is null then basic variables will be calclated automtically. They are 0 based. See
GANSO manual.
>
the main program
> k:=0;while k<iterNum+1 do
```

```
> #step 1. Solving the problem P(x_k) and obtain y_k by simplex-method
> #as the constraints and objective function are linear in y, the matrices A and b are constants
and y_k are not used currently
> print(`in step 1`);
>
> #substuting the current point x_k in the constraints
> for i from 1 to constrNum do hInXk[i]:=eval(h[i],{x=xx[k]}) end do;
> constrInXk:={seq(hInXk[i]<=0,i=1..constrNum)};
>
> #Solving the problem P(x_k) and obtain y_k by simplex-method
> yy[k]:=LPSolve(eval(f,{x=xx[k]}), constrInXk);
> yy[k]:=convert(yy[k][2],set);
> #substituting x_k and y_k into p
> pInX:=eval(p,{x=xx[k]}):
> pInXY:=eval(pInX,yy[k]):
>
> #Step 2. Solving the quadratic programming problem (10), we obtain vector lambda.
> bValue:=eval(b,{x=xx[k]}):#substituting x_k into the column of the RHS of equalities of
уравнений Lagrang set
>
> #checking for non-zero constraints. That is which constraints are active. and creating the list of
indeces of active constraints.
>
> active:=seq(i,i=1..0);#creating the empty sequence
> for i from 1 to constrNum do
>  h_kValue[i]:=eval(hInXk[i],yy[k]); #print(hInXk[i],h_kValue[i]);
>  if abs(h_kValue[i])<0.05 then #print(`i`,"th constrain is active");
>                         active:=active,i;
>  end if;
> end do;
>
> #list of indeces of active constraints
> activeList:=[active];
> dimActiveConstr:=nops(activeList);#the dim-n of the quadratic programming problem
> #print(`active=`,active);
```

```
>
> #getting the rows that correspond to active constraints from matrices A, H
> Asimple:=Transpose(A[activeList,1..2]);
> Hsimple:=H[activeList,1..2];
>
> #creating the quadratic function and substituting the values x_k, y_k
> Lambda := <seq(lambda[i],i = 1 .. dimActiveConstr)>;
> B := Multiply(Hsimple, Transpose(Hsimple));
> HX:=Multiply(Transpose(Hsimple),Lambda);
> linearPart := 2*Multiply(Transpose(pInXY), HX);
> quadraticF := Multiply(Multiply(Transpose(Lambda), B), Lambda)+linearPart;
> quadraticFValue:=eval(quadraticF,{x=xx[k]});
> quadraticFValue:=eval(quadraticFValue,yy[k]);
>
> #obtaing vactor lambda (if the constraints are RLI-regular), then the allowed set of Langrange
multipliers contains only a unique point!
> #lambda:=<linsolve(Asimple, bValue)>;
>
> #using NLPSolve to obtain lambda
> lambdaMax:=Optimization[NLPSolve](dimActiveConstr, ff, [NoUserValue,
NoUserValue,Asimple,bValue])[2];
> #NLPSolve(dimActiveConstr,ff,[Asimple, bValue] , maximize);
>
> #x0:=Vector(1..dimActive,datatype=float[8],[seq(0.5,i=1..dimActive)]):      # initial point for
local minimzation
> #print(`Setting up the equality constraints. The matrix of constraints and the right hand side
are`);
> #Eq:=Matrix(1..dimY,1..dimActive,datatype=float[8],Asimple):
> #RHSE:=Vector(1..dimY,datatype=float[8],bValue):
> #print(`Executing DFBM. The solution is `);
> #Mindfbm(dimActive,x0,val,ff,dimY,0,Eq,Lo,RHSE,Lo,Lo,Up,basic,300);
> x0;
> # lambda:=x0;
>
```

```
> #Step 3. If H[k]lambda[k]+p[k]=0, the the point satisfies the necessary condition of optimality and it is the point of local minimum. And the problem is solved. If not, then going to step 4.
> #if H[k]lambda[k]+p[k]=0 then exit end if;
> #print(H);
>
> #lambda:=<0,0,0,0,bValue[1]>;#for debuging
>
> res:=Multiply(Transpose(lambdaMax),Hsimple);
> #res:=Multiply(lambda,Hsimple);
> res:=Add(Transpose(res),pInXY):
> #res:=pInXY:
> normRes:=Norm(res,2);
> print(`res=`,normRes);
> if normRes<=epsilon then print(`The solution has been found. The number of interations k=`,k, `xx[k]=`, xx[k]); break end if;
>
>
> #Step 4. Using the obtained at step 2 direction by formula (11) we obtain the steepest ascend direction.
> v[k]:=res/Norm(res,2);#norm of vector res
>
> #the following is something like bundle method. increses convergence greatly!
> if k>=2 then v_temp:=v[k]+v[k-1]/2+v[k-2]/4;
> v_temp:=v[k]+v[k-1]*Norm(res,2)/Norm(xx[k]-xx[k-1],2)/8+v[k-2]/8;
> v[k]:=v_temp/Norm(v_temp,2);res:=res*1.5; end if;
>
> #Step 5. Searching for a step size beta to get a decrease along the direction v.
> print(`in step 5. calculating step`,`xx[k]=`,xx[k]);
>
>   #implemented the following methods of choosing a step size
>   #1)fixed step size. Requires a lot of iterations
>   if stepMethod=1 then beta[k]:=0.1; end if;
>
>   #2)linear search along the direction v. It converges fast, but requires a lot of computations of #function value along the direction v.
```

```
>   if stepMethod=2 then beta[k]:=stepSize(v[k]); end if;
>
>   #3)step = c*abs(v[k]), where c - constant (currently the best value is about 0,1)
>   if stepMethod=3 then beta[k]:=beta_init*Norm(res,2); end if; #experimental
>
>   #4)step beta[k]=c*beta[k-1], where c-constant. 1<c<2
>   if (stepMethod=4) and (k>0) then beta[k]:=stepSizeByDoubling() else beta[0]:=beta_init end
if;
>
>   #checking whether the future point is feasible. Adjusting step accordingly. If not, dividing it
by 2
>   i:=0; isFeasible:=false;
>    print(`checking whether the future point is feasible`);
>    while not(isFeasible) do #checking whether the point is feasible
>      nextPoint:=VectorAdd(xx[k],beta[k]*v[k]);#obtaning the next point
>      #checking whether the obtained point is feasible
>      isFeasible:=feasible({seq(eval(h[i],{x=nextPoint})<=0,i=1..constrNum)} );
>      #If not, then dividing the step size by 2
>      if not isFeasible then beta[k]:=beta[k]/2;print(`not feasible!`) end if;
>    end do;
>
>   print(`the obtained step size is beta[k]=`,beta[k]);
>
> #Step 6. The next approximation of vector x  we obtain by formular  x_k+1=x_k+beta_k*v[k].
And going to step 1, letting k = k+1.
>
> #print(`xx[`,k,`]=`,xx[k]);
> #print(`yy[`,k,`]=`,yy[k]);
> #f_k:=eval(f,{x=xx[k]});
> #fValue[k]:=eval(f_k,yy[k]);
> #print(`f[`,k,`]=`,fValue[k]);
>
> xx[k+1]:=VectorAdd(xx[k],beta[k]*v[k]);
>
> k:=k+1;#next iteration
```

```
>
> end do;#end of the main program
>
> lambda;
> #lambda:=linsolve(Asimple, bValue);

creating the list of iteration points and its trajectory
> seq([xx[i][1],xx[i][2],fValue[i]],i=1..iterNum):
> i_0:=11;#the initial interation number for trajectory
> tempSeq1:=seq(xx[i][1],i=i_0..k):
> leftBound:=min(tempSeq1);
> rightBound:=max(tempSeq1);
> tempSeq2:=seq(xx[i][2],i=i_0..k):
> upperBound:=max(tempSeq2);
> lowerBound:=min(tempSeq2);
> with(plots):
> seq1:=[seq(xx[i],i=i_0..k)]:
> seq2:=[seq(xx[i],i=0..k)]:
> points:=pointplot(seq1, symbolsize=13, colour=green,connect=false):
> points2:=pointplot(seq1, symbolsize=23, colour=green,connect=true):
> points3:=pointplot(seq2, symbolsize=13, colour=green,connect=false):
> points4:=pointplot(seq2, symbolsize=23, colour=green,connect=true):
> points3d:=pointplot3d([seq([xx[i][1],xx[i][2],fValue[i]],i=0..k-1)], symbolsize=13,
colour=black,connect=true):
>
> f_min_y(0,0.59);
>
> #building level sets
> #p2:=contourplot(f_min_y,0.34..rightBound,0.46..upperBound,contours=40):
> #p2:=contourplot(f_min_y,leftBound..rightBound,lowerBound..upperBound,contours=40):
> p21:=contourplot(f_min_y,-12..15,-0.5..5,contours=60):
> display(p21);
> display(points,points2,p2);
> display(points3,points4,p21);
> p3:=contourplot3d(f_min_y,-4..4,-4..4,contours=40,filled=true):
```

```
> display(p3);#display(points3d,p3);
> plot3d(f_min_y,-4..4,-4..4,grid = [25, 25]);
> #display(points3d,graph);
```