SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

GRAPHICAL USER INTERFACE FOR SOLVING OF GLOBAL OPTIMIZATION PROBLEMS

DIPLOMA THESIS

FCHPT-5414-50943

2012 **Bc. Ján Rusnák**

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

GRAPHICAL USER INTERFACE FOR SOLVING OF GLOBAL OPTIMIZATION PROBLEMS

DIPLOMA THESIS

FCHPT-5414-50943

Study programme:	Automation and Informatization in Chemistry and Food Industry
Number of study field:	2621
Study field:	5.2.14 Automation
Workplace:	ÚIAM FCHPT STU v Bratislave
Thesis supervisor:	prof. Ing. Miroslav Fikar, DrSc.
Consultant:	Ing. Radoslav Paulen

Bratislava 2012 **Bc. Ján Rusnák**

DIPLOMA THESIS TOPIC

Registration number:	FCHPT-5414-50943
Student's ID:	50943
Author of thesis:	Bc. Ján Rusnák (50943)
Study programme:	Automation and Informatization in Chemistry and Food Industry
Study field:	5.2.14 automation
Thesis supervisor:	prof. Ing. Miroslav Fikar, DrSc.
Consultant:	Ing. Radoslav Paulen
Workplace:	ÚIAM FCHPT STU v Bratislave

Topic: **Graphical user interface for solving of global optimization problems**

Length of thesis: 60

Topic specifications:

Aim of this work is to create graphical user interface (*GUI*) for solving of global optimization problems. Graphical user interface will be created in MATLAB programming environment. Deterministic method alphaBB and chosen stochastic method will be used for solving of global optimization problems. Graphical user interface will further enable user for comfortable introduction of solved optimization problem. Moreover, user will be able to choose the desired approach from different methods and settings. Finally, graphical user interface will output a summary of obtained results.

Selected bibliography:

1. PARDALOS, P. -- FLOUDAS, C. *Frontiers in Global Optimization.* Dortrecht: Kluwer Academic Publishers, 2003. 597 s.

2. HIGHAM, D. -- HIGHAM, N. J. *MATLAB GUIDE*. Philadelphia: SIAM - Society for Industrial and Applied Mathematics, 2000. 283 s. ISBN 0-89871-469-9.
3. *MATLAB. High-Performance Numeric Computation and Visualization Software. Reference GUIDE.* Natick: Math Works, 1994. 548 s.

Diploma thesis topic submission date: **13. 02. 2012**

Deadline for submission of Diploma thesis: **19. 05. 2012**

Bc. Ján Rusnák
Student

prof. Ing. Miroslav Fikar, DrSc.
Head of office

prof. Ing. Miroslav Fikar, DrSc.
Study programme supervisor

ACKNOWLEDGEMENT

I would like to thank to my supervisor, Prof. Ing. Miroslov Fikar, DrSc., and to my consultant, Ing. Radoslav Paulen, for their help, patience and valuable advices throughout my studies. Also I would to thank to whole my family and friends for given me material and psychical support.

ABSTRACT

Graphic user interface belongs to those tools, which makes using of computers easier. For user it enables easy and effective solving of various difficult mathematical problems. Then it is no more necessary to swing himself among many commands and preferences, which solve this problems. There is a short description of global optimization solving method, which is used in this work and some common basic information about designing a graphic user interface (*GUI*). Next part of work is addressed to a *GUI* itself and running it. At the end there are few examples solved.

Keywords: Graphical User Interface, Global Optimization, Minimizer, Global Minimum

ABSTRAKT

Grafické užívateľské rozhranie je jedným z nástrojov, ktorý uľahčuje použitie výpočtovej techniky. Umožňuje ľahko a efektívne riešiť žložité matematické problémy bez toho, aby sa užívateľ musel vyznať v sieti rozličných príkazov a nastavení, ktoré tieto problémy riešia. V práci je stručne vysvetlená metóda, ktorou sa rieši optimalizačný problém. Ďalej sú v nej základné informácie o návrhu grafického užívateľského rozhrania (*GUI*). Na záver je vyriešených niekoľko príkladov.

Kľúčové slová: Grafické Užívateľské Rozhranie, Globálna Optimalizácia, Minimizer, Globálne Minimum

CONTENTS

LIST OF FIGURES

INTRODUCTION

How to obtain an optimum? This is the question that a lot of people are trying to answer. Finding the optimum is very important in many ways of today's modern world, where everyone is trying to decrease his costs and increase his profits. According to that there are lots of problems, which are waiting for its solution.

There exists one special branch of optimal problems, for which is characteristic presence of the nonconvexity in one or more of its subfunctions. Nonlinear theory of optimization is trying to eliminate these nonconvexities via performing the global optimum search and develop effective global optimization methods (Adjiman, 1998a).

In first part of this work we will discuss global optimization and its characteristic features. We will describe the main problem with reaching globally optimal solutions – the nonconvexity. It causes the presence of many local optima and we have to differentiate them and find the global optimum. Therefore there are developed various methods for obtaining it. We know two basic approaches of global optimization- deterministic and stochastic. We use one method from each to find a global optimum. First one is the deterministic spatial *Branch and Bound* method. Second one is the stochastic method called *Multistart*.

In next part we will simply describe *MATLAB's* Graphic User Interface (*GUI*). It is helpful tool in solving problems without knowing every detail of calculation, enables acquiring of result to bigger group of people. There are two basic ways of designing the *GUI*. First is through the *GUI Development Environment*, which is interactive in many ways and it is simple to create *GUI* through it. Second way is creating *GUI* programmatically, which is considered harder, but enables us to see into the *GUI* itself, understand processes in it and achieve some functionality, which are not supported in *GUIDE*.

We are designing *GUI* programmatically in very simple way, and this is the third part of our work. Here we will describe composition of our *GUI*, its functionality and types of problems, which we are solving. We have two basic classes of solving problems- deterministic and stochastic approach. Each one works on a totally different principle, because deterministic algorithm works systematically and stochastic method works on the principle of probabilistic search.

At the end we will solve few one-dimensional and multi-dimensional optimization problems by using both approaches. Results will be displayed in final figures in appropriate text boxes. Some function can be drawn in plots, where we can see proof of our right calculation.

1 GLOBAL OPTIMIZATION

In recent years global optimization becomes an important part of searching for global optimum. Its sphere of action is from process engineering, continues with economics and to the other industry and applied science. The presence of nonconvexity in many solved function causes that there are a lot of local optima. It means that we have to find just one optimum from the locals and we will call it global optimum. Global optimization computes this global optimum, which can be a global minimum or global maximum in a feasible region where constraints have to be fulfilled. Objective function is minimized regarding to the constraint function in equality or inequality form (Repčíková, 2009; Adjiman, 1998a).

The base division of global optimization methods is stochastic and deterministic methods. Stochastic methods such as *Multistart*, *clustering methods*, *variable neighbourhood* search and others cannot guarantee finding of global optimum since they are based on random search techniques (Paulen, 2008). In our case we use *Multistart* method, where the searching for global minimum is performed via random generation of starting points of local optimization.

On the other hand deterministic methods for example: *Branch and Bound* or *Interval analysis* method provides finite ε-convergence and a global optimality of the obtained solution (Paulen, 2008).

Deterministic global optimization solves these important tasks (Floudas, 2005):

- Identify a global minimum of the objective function (the lowest value of objective function) subject to the set of constraints.
- Identify lower and upper bounds of global minimum of objective function which are valid for the whole feasible region.
- Identify a set of good quality local solutions in the vicinity of the global solution
- All solutions satisfy the set of equality and inequality constraints.
- Prove that a constrained nonlinear problem is feasible or infeasible.

In recent few years there are discussed these classes of mathematical problems (Floudas, 2005):

- Twice continuously differentiable nonlinear optimization, NLPs,
- Mixed-integer nonlinear optimization, MINLPs,
- Differential-algebraic systems, DAEs,

- Grey-box and non-factorable problems,
- Bilevel nonlinear and mixed integer optimization.

One of the most used deterministic global optimization methods is the *Branch and Bound* (*BB*) method with its extension, spatial *BB.* Algorithm of *αBB*, which is one of the spatial *BB* methods, guarantees for convergence to a point near the global minimum subject to chosen accuracy criterion (Adjiman, 1998a).

1.1 BRANCH AND BOUND METHOD

1.1.1 Problem statement

Formulation of NLP problem of BB method:

$$\min_x J(x) \qquad (1.1)$$

$$h(x) = 0 \qquad (1.2)$$

$$g(x) \leq 0 \qquad (1.3)$$

$$x^L \leq x \leq x^U$$

where $x \in R^n$ is defined in n-dimensional Euklid's space. The function *J(x)* represents optimization criterion, *h(x)* is vector equality constrains and *g(x)* is vector of inequality constraints. All these functions belong to C^2, therefore they are twice differentiable. The x^L and x^U are bound constraints for variable *x*.

1.1.2 Algorithm of BB

The algorithm of BB minimizes the objective function *J(x)*, which represents optimization criterion with set of equality and inequality constraints. Problem solution begins with relaxation of nonconvex problem. Here we acquire a lower bound of solution using NLP algorithms designed for local extremes searching. Nonconvex objective function is underestimated by relaxed problem's objective function at the certain interval. Every local minimum of this problem is a global minimum as well. The upper bound is acquired as a solution of original nonconvex problem (Černá, 2010).

The relative distance between these two bounds, which is defined as ratio of difference between values in upper and lower bound and lower bound at a given intervals in numerical value, is compared with the chosen accuracy ε. When the distance is bigger

than accuracy, algorithm continues in calculation by dividing the original interval into two new subintervals. The problem is now solved on each subinterval separately. In case, when the lower bound on the certain subinterval is bigger than upper on the other, then there is no global minimum on this subinterval and it is eliminated from the calculation. This operation is called *fathoming*. The whole process of branching and eliminating the subintervals will continue until distance between upper and lower bound equals or is less than chosen accuracy ε. Fig. 1 represents scheme of *αBB* algorithm for calculation the global optimization problems.

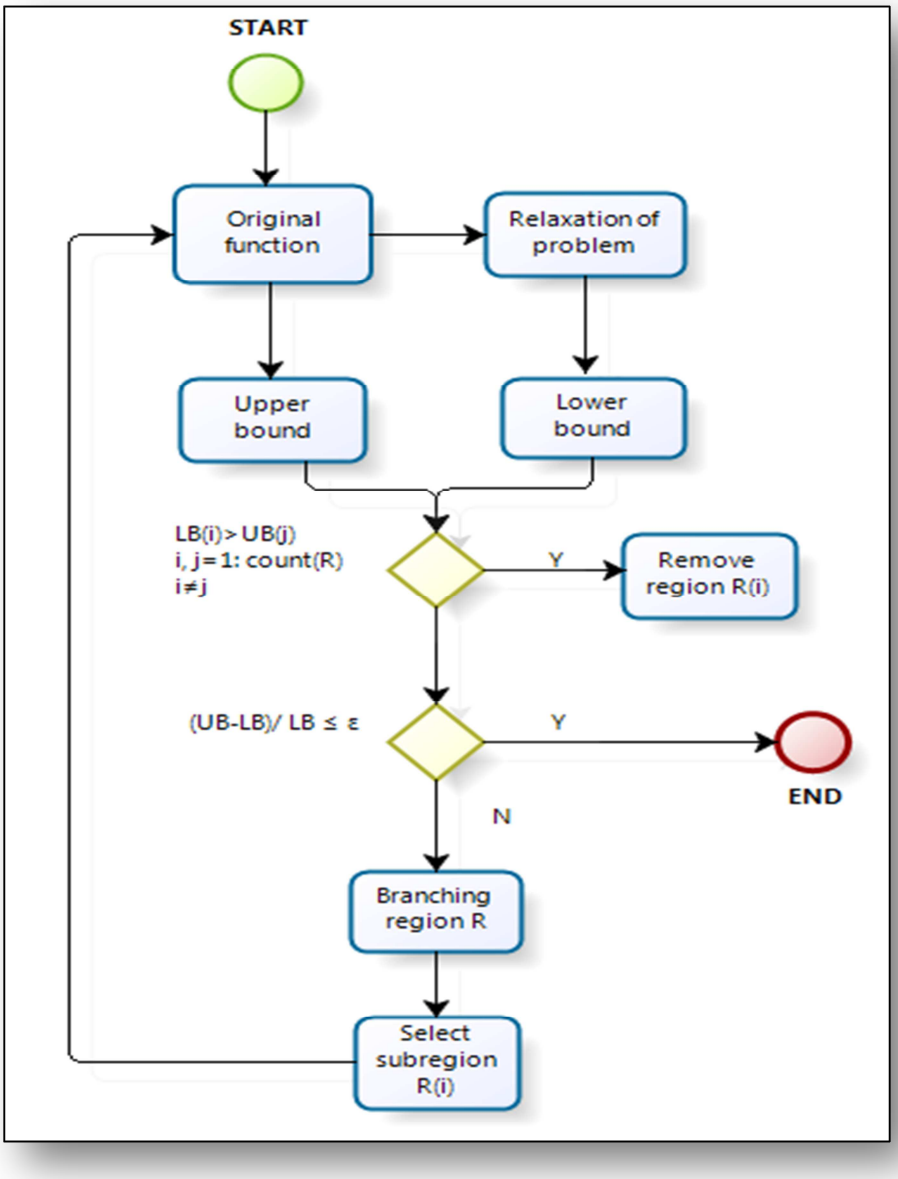

Fig. 1: Scheme of αBB algorithm

1.1.3 Convex relaxation

It is applied that the objective function and inequality constraint functions can be separated into the sum of special nonconvex terms (STNT) and general nonconvex terms (ATNT). Equality constraint functions are transformed into two inequality constraints.

$$h(x)=0 \quad \Rightarrow \begin{matrix} h(x)\le 0 \\ -h(x)\le 0 \end{matrix} \tag{1.4}$$

The objective function can be written in form:

$$J(x)=STNT(x)+ATNT(x) \tag{1.5}$$

where

$$STNT(x)=LT(x)+CT(x)+\sum_{i=1}^{n_{BT}} b_i x_{BTi,1} x_{BTi,2}+\sum_{i=1}^{n_{TT}} t_i x_{TTi,1} x_{TTi,2} x_{TTi,3}+$$
$$+\sum_{i=1}^{n_{FT}} f_i \frac{x_{FTi,1}}{x_{FTi,2}}+\sum_{i=1}^{n_{FTT}} ft_i \frac{x_{FTTi,1} x_{FTTi,2}}{x_{FTTi,3}}+\sum_{i=1}^{n_{UT}} UT_i\left(x_{UT_i}\right) \tag{1.6}$$

$$ATNT(x)=\sum_{i=1}^{n_{NT}} NT_i(x) \tag{1.7}$$

Here we assume $L(x)$ as linear;

$C(x)$ as convex;

n_{BT} as count of bilinear terms;

$x_{BTi,1}$ and $x_{BTi,2}$ represents two variables in i-*th* bilinear term;

b_i is its coefficient;

n_{TT} as count of trilinear terms;

$x_{TTi,1}$, $x_{TTi,2}$ and $x_{TTi,3}$ represents three variables in i-*th* trilinear term;

t_i is its coefficient;

n_{FT} as count of fractional terms:

$x_{FTi,1}$ and $x_{FTi,2}$ represents two variables in i-*th* fractional term;

f_i is its coefficient;

n_{FTT} as count of fractional trilinear terms;

$x_{FTTi,1}$, $x_{FTTi,2}$ and $x_{FTTi,3}$ represents three variables in i-*th* fractional trilinear term;

ft_i is its coefficient;

n_{UT} as count of one-dimensional concave terms;

$UT_i(x_{UTi})$ as one-dimensional concave term;

x_{UTi} represents its variable;

ft_i is its coefficient;

n_{NT} as count of general nonconvex terms and

$NT_i(x)$ as i-*th* nonconvex term (Adjiman, 1998a).

It is not necessary to underestimate the linear and convex terms. During underestimation we introduce new variables - w_{BT}, w_{TT}, w_{FT}, w_{FTT} for nonconvex (bilinear, trilinear, fractional, fractional trilinear) terms, where they have to satisfy certain constraints. Underestimation of one-dimensional concave terms does not need introducing of new variables, neither new constraints. It is simply carried out by linearization of them at the vicinity of a lower bound of each variable (Paulen, 2008, Repčíková, 2009).

Bilinear terms in form of *xy* are underestimated on region $\lfloor x^L, x^U \rfloor \times \lfloor x^L, x^U \rfloor$ by introducing new variable w_{BT} . This variable replaces all the terms in *xy* form in our problem. It is necessary to input four new linear inequality constraints (Repčíková, 2009).

$$\begin{aligned} w_{BT} &\geq x^L y + x y^L - x^L y^L \\ w_{BT} &\geq x^U y + x y^U - x^U y \\ w_{BT} &\leq x^L y + x y^U - x^L y \\ w_{BT} &\leq x^U y + x y^L - x^U y^L \end{aligned} \qquad (1.8)$$

Similarly, for trilinear terms *xyz* is introduced new variable w_{TT} with eight linear inequality constraints. For the fraction bilinear terms *x/y* we introduce variable w_{TT} with constraints depending on the sign of *x,* for fractional trilinear terms *xy/z* we introduce new variable w_{FTT} with constraints, where our lower bounds are bigger or equal to zero.

General nonconvex terms are underestimated at whole region $\lfloor x^L, x^U \rfloor$ by function $L_\alpha(x)$ defined as:

$$L_\alpha(x) = J(x) + \sum_{j=1}^{n} \alpha_j \left(x_j^U - x_j\right) \cdot \left(x_j^L - x_j\right) \qquad (1.9)$$

where for $j = \overline{1, n}$ it is assumed $\alpha_j \geq 0$. Every nonconvexity in original function $J(x)$ can be relaxed by convex quadratic term subject to the sufficiently big values of

parameters α_j. It is guaranteed that $L_\alpha(x)$ is an underestimator of $J(x)$, because there is a proof that the sum in (1.8) is negative at the whole region *C* (Adjiman, 1998a)

The final underestimator $L(x)$ to the original function $J(x)$ subject to (1.5, 1.9) acquires following form:

$$L(x) = LT(x) + CT(x) + \sum_{i=1}^{n_{BT}} b_i w_{BTi} + \sum_{i=1}^{n_{TT}} t_i w_{TTi} + \sum_{i=1}^{n_{FT}} f_i w_{FTi} + \sum_{i=1}^{n_{FTT}} ft_i w_{FTTi} +$$
$$+ \sum_{i=1}^{n_{UT}} \left[UT_i\left(x_{UTi}^L\right) + \frac{UT_i\left(x_{UTi}^U\right) - UT_i\left(x_{UTi}^L\right)}{x_{UTi}^U - x_{UTi}^L}\left(x_i - x_{UTi}^L\right) \right] + \qquad (1.10)$$
$$+ \sum_{i=1}^{n_{NT}} \left[NT_i(x) + \sum_{j=1}^{n} \alpha_{ij}\left(x_j^U - x_j\right)\left(x_j^L - x_j\right) \right]$$

where α_{ij} represents i-*th* general nonconvex term and its j-*th* variable and there have to be satisfied particular constraints for variables w_{BT}, w_{TT}, w_{FT}, w_{FTT} (Paulen 2008, Repčíková, 2009).

1.2 SPATIAL BRANCH AND BOUND METHOD

The algorithm of *αBB* method underestimates only general nonconvex terms of $J(x)$ in (1.5). Anyway by our approach we declare all terms of $J(x)$ as general nonconvex. The final underestimator $L(x)$ for general nonconvex terms is a convex function with positive semi-definite Hessian matrix $H_L(x)$ (Repčíková, 2009).

The final underestimator acquires form:

$$L(x) = \sum_{i=1}^{n_{NT}} \left[NT_i(x) + \sum_{j=1}^{n} \alpha_{ij}\left(x_j^U - x_j\right)\left(x_j^L - x_j\right) \right] \qquad (1.11)$$

The dependence of final underestimator's Hessian matrix $H_L(x)$ from original function's Hessian matrix $H_J(x)$ is described by following formula:

$$H_L(x) = H_J(x) + 2\Delta \qquad (1.12)$$

where Δ is diagonal shift matrix whose diagonal elements are the α_j's (Adjiman, 1998a).

For the identification of Δ it is necessary to transform underestimator to the form with only one value of α_i . It is proved that function $L(x)$ has correct convex underestimator of objective function $J(x)$ while:

$$\alpha \geq \max\left(0, -\frac{1}{2} \min_{i, x^L \leq x \leq x^U} \lambda_i(x)\right) \qquad (1.14)$$

where λ_i are eigenvalues of Hessian matrix $H_J(x)$ (Repčíková, 2009).

In *BB* algorithms rate of convergence often depends on the way of branching of original interval. In *αBB* algorithms it is even more important, because the quality of underestimator is directly proportional to variables' bounds. Now there are classified four main strategies of branching the intervals for spatial *BB*:

1-st strategy: Using k-section for all or some of the variables.

2-nd strategy: Using quality criterion measurement of underestimator for every term regarding to the maximal distance between concrete term and its underestimator.

3-rd strategy: Using quality criterion measurement of underestimator for every term regarding to the maximal distance in optimizer.

4-th strategy: Using the total influence measurement of every variable to the quality of relaxed problem (Repčíková, 2009).

In our work we use branching on least reduced axis, which is part of first strategy group.

2 GRAPHICAL USER INTERFACE

Powerful computing equipment brings new options in solving complicated mathematical problems. On the other side solvers, which were designed for calculation, are not often very user-friendly and their application is not so easy for everybody. In recent few years *MATLAB* becomes an important tool for solving various difficult problems. However, it offers option of creating a graphical user interface (*GUI*), which graphically displays one or more windows with control components. This allows user to perform interactive tasks without typing commands at the command line and obtain solution. There is no need for user to code program and understand details of the tasks. This is performed by using *GUI* components as menus, push buttons, radio buttons, sliders and few others. We can select size and position of these components if needed. User can make the components do what he needs by clicking or manipulating with them via keystrokes. *GUI* can also provide any type of computation, write and read files containing needed data, communicate with other *GUI*s or display results by plotting tables or plots (www.mathworks.com, 2012).

We recognize two types of building the *GUI* in *MATLAB*'s environment. First approach is called *GUIDE*, what means *GUI* Development Environment, and it is interactive *GUI* construction editor. It works simply by populating the generated figure with components from within a graphic layout editor. There are created associated code files, which contain callbacks for the *GUI* and its components. The figure, either the code file, are both saved and using one means using other for the purpose of running the *GUI*.

Second *GUI*-building approach is programmatic. We create a code file, where have to be defined all the necessary component properties of the figure and controls, as well as the callbacks. This causes that the codes in the files are generally longer than through *GUIDE*. Anyway, the figure does not have to be saved, because code creates a new one each time when user runs it.

There is option of combining them into one hybrid, although they look differently. We can create *GUI* with a *GUIDE* and then modify it programmatically to the some degree, but this does not stand in reverse (www.mathworks.com, 2012).

2.1 CREATING GUI WITH GUIDE

GUIDE, the Graphical User Interface Development Environment, represents kit of tools for creating *GUI*s in *MATLAB*. Using these tools we can simplify the process of designing and programming the *GUI*. *GUIDE* starts simply just by typing *GUIDE* in command line. The context menu will be displayed and we can choose between creating new figure or editing already existing figure. In Fig. 2 is displayed figure in *Layout Editor*, which enables us to populate a *GUI* with *GUI* components. We can add sliders, text fields, buttons, axes and some other components via clicking and dragging them into the layout area. Afterwards they could be resized to user's desired size. Other tools from editor enable us to:

- Create menus and context menus
- Create toolbars
- Modify the appearance of components
- Set tab order
- View a hierarchical list of the component objects

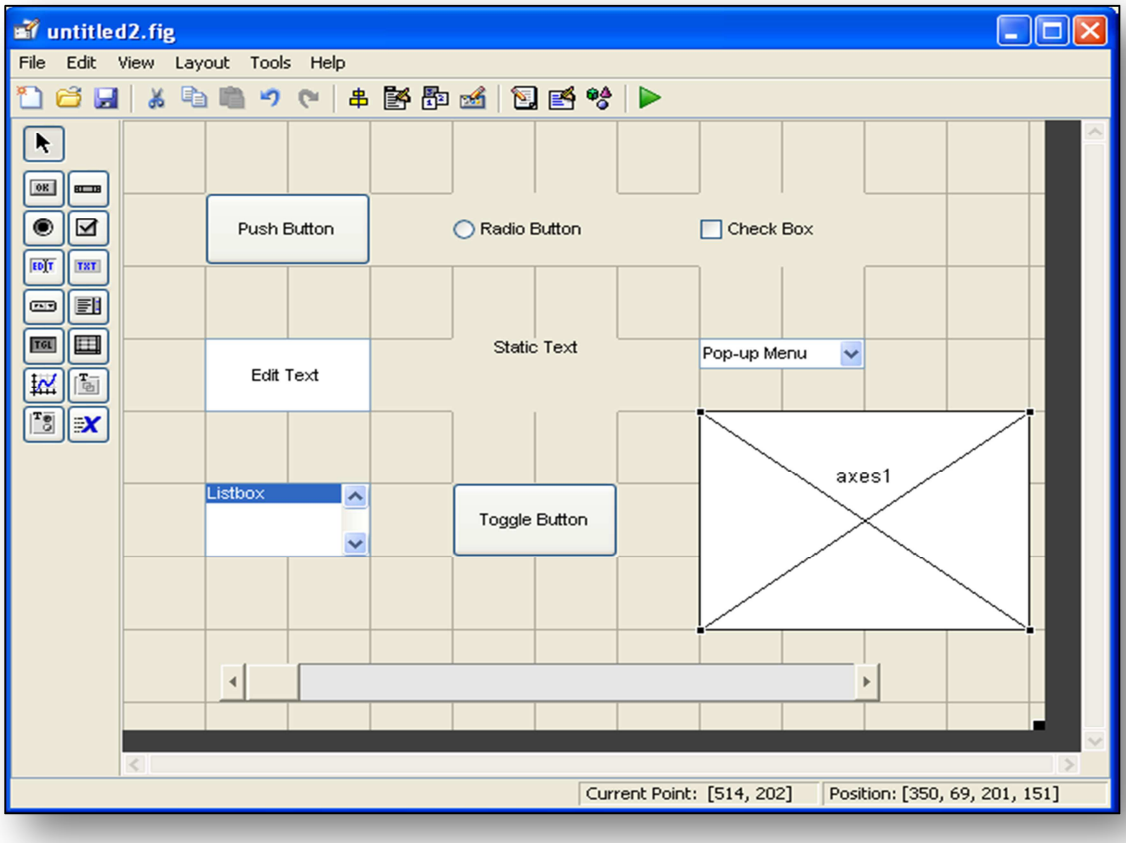

Fig. 2: *MATLAB's GUI Development environment* with examples of its components

When the designing of *GUI* components is finished, then we can save our *GUI* layout. Except our figure's file, there is automatically generated file with *MATLAB* code for controlling the way the *GUI* works. The *GUI* is initialized and organizes its *callback* by this code. *Callbacks* are functions that execute in response to user activity, such as mouse click. We can edit these files and add code to the *callbacks* to perform function we want or add any new functions, too.

2.2 CREATING GUI PROGRAMMATICALLY

Creating *GUI* is realized programmatically in *MATLAB* editor by writing code into the file. *GUI*s support almost all of *MATLAB* functions. We can use them to add components to *GUI*s, modify its behavior, change layout or manage data within the *GUI*. In *MATLAB*, a *GUI* is a figure, so we have to create the figure first and obtain a handle for it. We set some of the figure properties as a *position* of the figure on the screen, its *visibility*, *name*, *color* etc. Then we start writing the statements that add components of the *GUI* with specific functions. Using the *uicontrol* function and setting the specific property value we can add these components:

- Check boxes
- Editable text fields
- Frames
- List boxes
- Pop-up menus
- Push buttons
- Radio buttons
- Sliders
- Static text labels
- Toggle buttons

Often used function is *uimenu* function, which specifies the menu of the figure, then *uipanel* function or *uibuttongroup*, which groups components, *axe* function and many others.

Adding the *callback* function to each component enables *MATLAB* to execute it and associate it with particular event through its specified name. If it is everything set correctly, the *GUI* can run a saved code in *MATLAB*'s m-file.

We use programmatic approach of creating the *GUI* in our work. Creating the figure window is realized via running command *figure();* in Fig. 3. We set its color and position via commands *color* and *position.* Property *name* enables us to define name of the figure. If we set value of properties *resize* and *numbertitle* to "*off*", user cannot change size of the figure respectively number of the figure is not displayed in its title. As we can see in Fig. 5, via property *name* we can display text "*GLobal OPtimization start*" in figure's title. *GUI* As we can see in Fig. 4. is an example of creating the pushbutton, because it has set *push* as *style* option. Size and weight of the font is set via *fontsize* and *fontweight* properties. Property *string* represents, which text we want to display in *pushbutton*. *Backgroundcolor* property sets color of background and *foregroundcolor* sets color of the text *"Quit"*. *Callback* function executes commands *close all* and *clear*, so this pushbutton is assigned for closing the whole *GUI*.

```
f_1 = figure('Color',[0.2 0.6 0.8],...
    'Resize','off',...
    'Numbertitle','off',...
    'name','GLobal OPtimization start', ...
    'position',[s_width/4 s_heigth/4 s_width/2 s_heigth/2]);

% ScreenSize is a four-element vector: [left, bottom, width, height]
```

Fig. 3: Code for creating figure and setting its properties

```
push_3 = uicontrol(gcf,...
    'Style','push',...
    'Position',[3*f_width/7 f_heigth/8 f_width/7 f_heigth/13],...
    'String','Quit',...
    'Fontsize',f_heigth/(13-2)/4,...
    'Fontweight','bold',...
    'BackgroundColor',[1 0 0],...
    'ForegroundColor','black',...
    'CallBack','close all,clear');
```

Fig. 4: Code for creating pushbutton *QUIT* and setting its properties

3 GLOP – GRAPHICAL USER INTERFACE FOR GLOBAL OPTIMIZATION

Graphical user interface for solving problems of global optimization (*GLOP*), this work deals with, was designed for using deterministic *αBB* method algorithm and stochastic *Multistart* method. We use programmatic approach of creating the *GUI* files. Its main purpose is to decrease complexity of obtaining the global minimum and to open it for new users not so familiar with its calculation. The problem input is comfortable, and after few clicks on the control tools, problem calculation can start. Solution of problem is displayed with parameters such as:

- Global minimum
- Minimizer
- Number of iterations (for deterministic method)
- Runtime in seconds

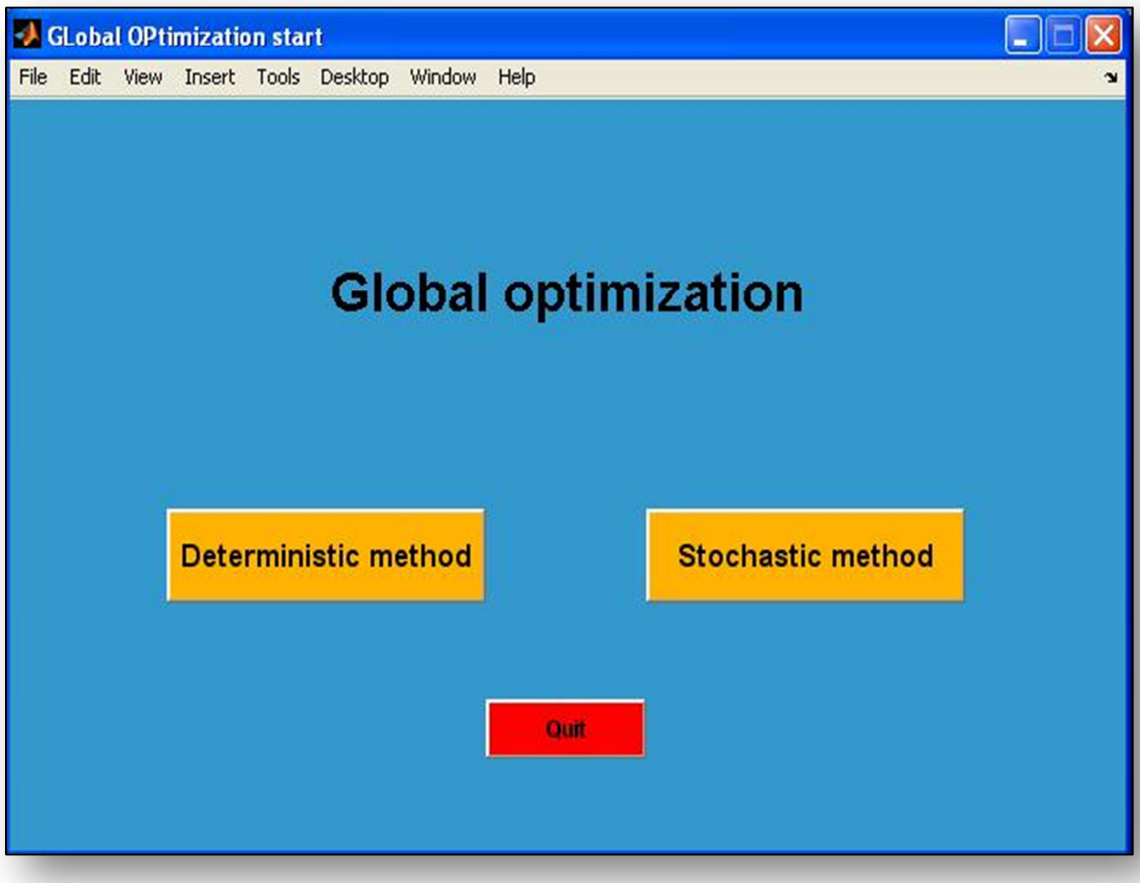

Fig. 5: Starting window of *GLOP*

Running the *GLOP* begins by entering the *glopstart* command in the *MATLAB* command line. As the code in file *glopstart.m* begins to execute, sets of commands draws the figure of *GUI*'s opening window in Fig. 5. We can see three pushbuttons with different functionality and text describing name of the *GUI*. In this part we will discuss about how we obtained this output.

In Fig. 6 we use command *scrsz()* to detect screen size and store it in *global* variables *s_width* and *s_height*. We need these variables to draw figure proportionally to the screen size. Other parameters of figure are set such as it was said in part 2.2. After that we detect size of the figure with help of the *get()* function, because we need to use it later. We store its values in variables *f_width* and *f_height*.

```
global s_width s_heigth
scrsz=get(0,'screensize');
s_width=scrsz(3);
s_heigth=scrsz(4);

f_1 = figure('Color',[0.2 0.6 0.8],...
    'Resize','off',...
    'Numbertitle','off',...
    'name','GLobal OPtimization start', ...
    'position',[s_width/4 s_heigth/4 s_width/2 s_heigth/2]);

fsz=get(f_1,'position');
f_width=fsz(3);
f_heigth=fsz(4);
% ScreenSize is a four-element vector: [left, bottom, width, height]
```

Fig. 6: Code for creating new figure with setting of its position

```
text_1 = uicontrol(gcf,...
    'Style','text',...
    'Position',[f_width/4 4*f_heigth/6 f_width/2 f_heigth/8],...
    'String','Global optimization',...
    'Fontsize',f_heigth/(8-2)/3,...
    'Fontweight','bold',...
    'BackgroundColor',[0.2 0.6 0.8],...
    'ForegroundColor','black');
```

Fig. 7: Code for creating static text with setting of its position

The static text is drawn by *uicontrol function* with *style* option *text* (Fig. 7). Code draws text “Global optimization”, sets its properties such as position, color, size and weight of the font. As we can see, it uses variables *f_width* and *f_height* from Fig. 6 to proportional setting of font and position. The pushbuttons, which are representing deterministic or stochastic method of solving the problem, also use *uicontrol function*, but with *style* option *push* (Fig. 8). We set similarly their *position* and *fontweight* parameters such as in previous Fig. 7. At the top of code is displayed value of property *string*, which informs user about intended action of *pushbutton*. The difference between them is executed via their *callback* function, where each one opens different corresponding submenu window.

```
push_1 = uicontrol(gcf,...
    'Style','push',...
    'Position',[f_width/7 2*f_heigth/6 2*f_width/7 f_heigth/8],...
    'String','Deterministic method',...
    'Fontsize',f_heigth/(8-2)/5,...
    'Fontweight','bold',...
    'Fontweight','bold',...
    'BackgroundColor',[1 0.7 0],...
    'ForegroundColor','black',...
    'CallBack','close all,glop_deterministic');


push_2 = uicontrol(gcf,...
    'Style','push',...
    'Position',[4*f_width/7 2*f_heigth/6 2*f_width/7 f_heigth/8],...
    'String','Stochastic method',...
    'Fontsize',f_heigth/(8-2)/5,...
    'Fontweight','bold',...
    'BackgroundColor',[1 0.7 0],...
    'ForegroundColor','black',...
    'CallBack','close all,glop_multistart');
```

Fig. 8: Code for creating static the two pushbuttons

3.1 DETERMINISTIC GLOBAL OPTIMIZATION

While designing the *GLOP*, we decided to distinguish between one and multi-dimensional problems (Fig. 9). It is because we can plot simply the figure of one-dimensional problem with the marked solution as confirmation of correct calculation,

while by multi-dimensional problems it could not be so easy. As NLP's problems solver we use *MATLAB*'s *fmincon* function.

3.1.1 Deterministic one-dimensional global optimization

Clicking on the push button *One-dimensional* in Fig.9, we activate dialog window in Fig. 10, where problems of one-dimensional global optimization are solved. Into the editable text fields we enter the corresponding information as problem's objective function; the lower bound x^L and upper bound x^U ; if exist, then a number of equality and inequality constraints and value of accuracy criterion, if it is different from the default value. Depending on the number of corresponding constraints there will be drawn dialog windows with the same number of editable text fields. We can see window for entering inequality constraints in Fig. 11. There are three editable text fields, where constraints. If we enter four as a number of inequality constraints, four editable text fields will be drawn. Window for entering equality constraints is generated similarly as Fig. 11.

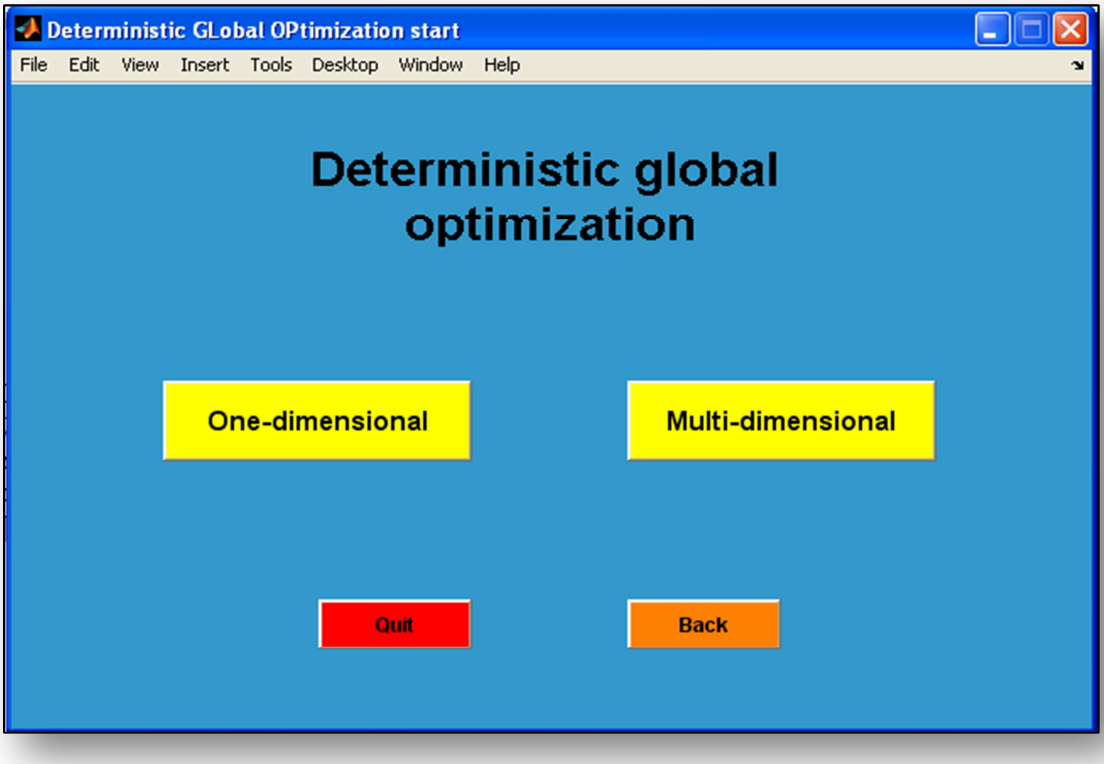

Fig. 9: Starting window with selection of deterministic global optimization's style

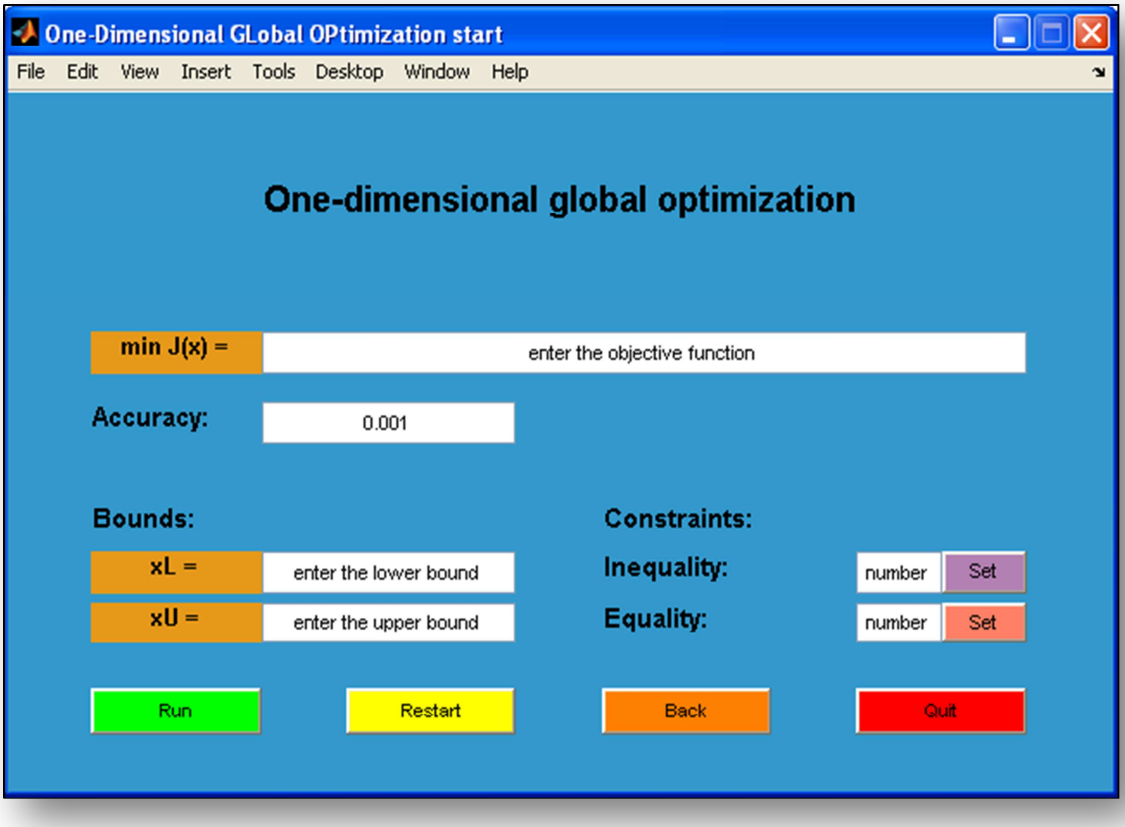

Fig. 10: Dialog window for entering one-dimensional problem

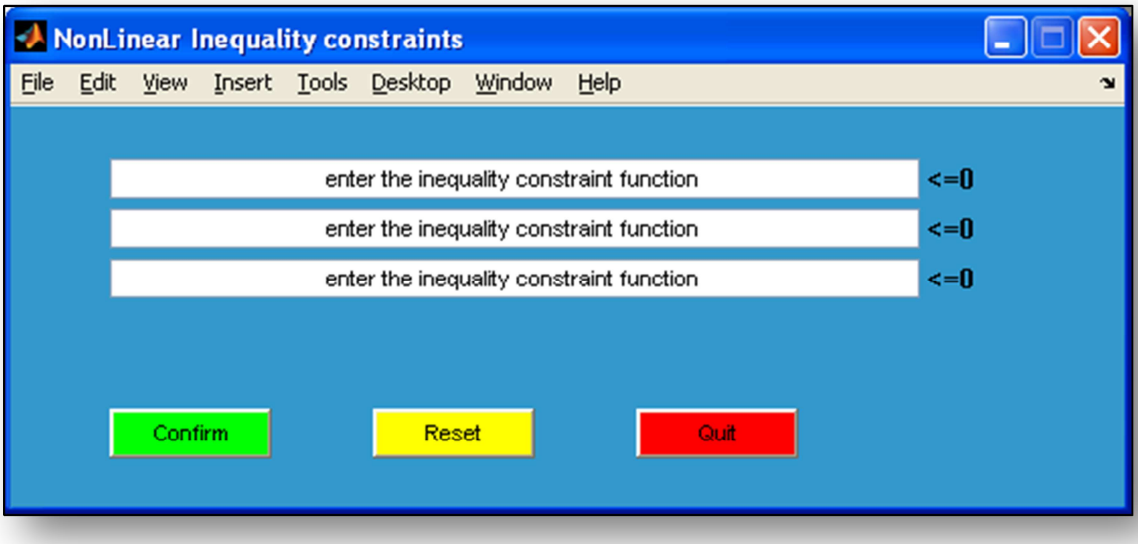

Fig. 11: Window for entering inequality constraints

The objective and constraint functions are saved as strings of characters in a variable of that name. For the purposes of *αBB* method we need to transform these variables into

symbolic form. Symbolic toolbox enables us to obtain functions' first and second derivative. There is a simple example of obtaining these derivatives in Fig. 12. We declare *x* as a symbolic variable and then declare *v* as symbolic variable, too. In two-dimensional case, *v* would be declared as vector of two symbolic variables. This allows us to obtain derivative for each variable.

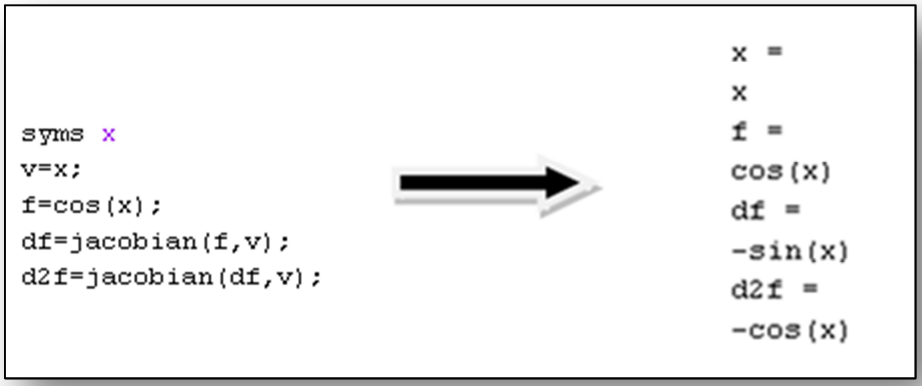

Fig. 12: Example of obtaining the derivatives

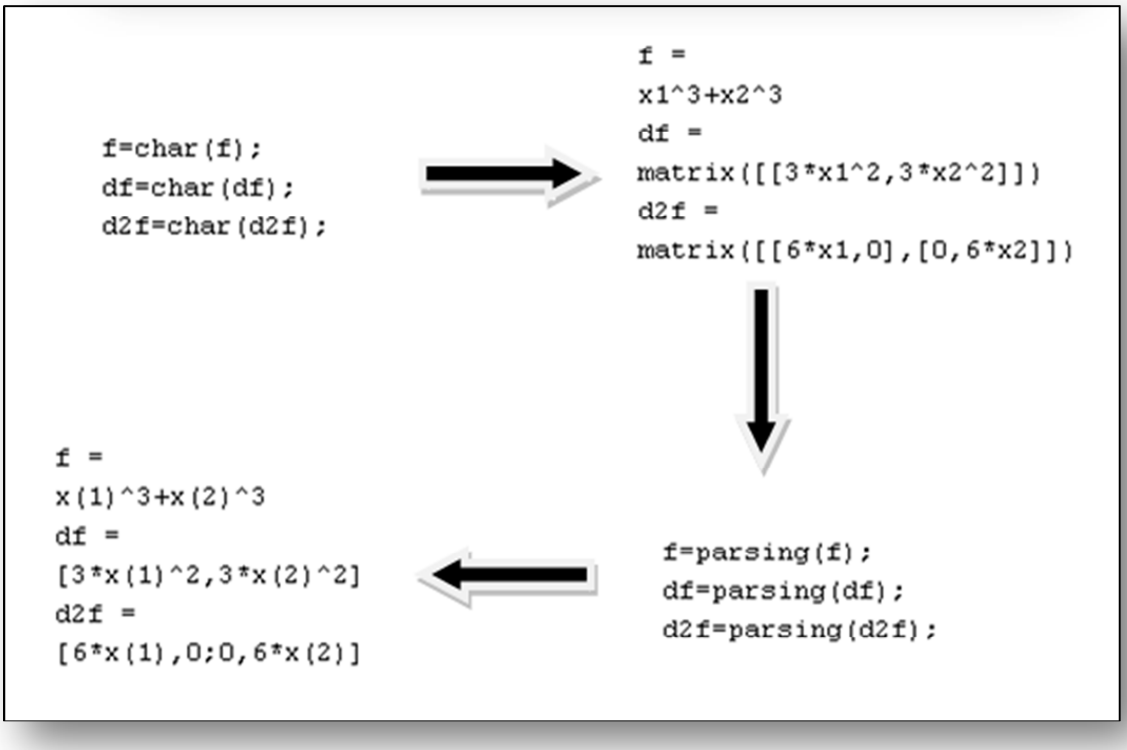

Fig. 13: Example of transformation of variables to the desired output

After that we declare variables of functions and its' derivatives as a strings again. Next example in Fig. 13 describes transformation of two-variable objective function through our *parsing.m* file in Fig. 14.

```
function string=parsing(string)

string=regexprep(string,'matrix','');
string=regexprep(string,'],[',';');
string=regexprep(string,'([','');
string=regexprep(string,'])','');
string=regexprep(string,'x(\d+)','x($1)');
```

Fig. 14: Function *parsing.m* , which transforms our strings

When we get requested output, we can create objective and constraint functions' files dynamically by using *fopen*, *fprintf* and *fclose* commands in Fig. 15. Function as in Fig. 16, which *fmincon* is used in calculation, is generated automatically. We consider this option more effective than slow evaluation of symbolic variables in functions, and thus the runtime of calculation is shorter.

```
global f
delete objfun.m
df=diff(f);
fid=fopen('objfun.m','wt');
fprintf(fid,'function [y,dy] = objfun(x)\n');
fprintf(fid,'\n');
f=char(f);
df=char(df);

fprintf(fid,'y=%s;\n',parsing(f));
fprintf(fid,'dy=%s;\n',parsing(df));
fclose(fid);
```

Fig. 15: File generating objective function's file *objfun.m*

```
function [y,dy] = objfun(x)

y=cos(x);
dy=-sin(x);
```

Fig. 16: Dynamically generated file of objective function

Calculation starts by clicking the pushbutton *Run*. There is generated figure, as in Fig. 17, with two subplots, which are generated during each iteration. First one represents dependence of accuracy according to the number of iterations, the other dependence of bounds size according to the number of iterations. It changes during the calculation and user can see the progress, which is made. Algorithm terminates when relative error equals or is less than chosen accuracy or the number of maximum iterations in achieved. Accuracy is defined as a ratio of difference between values in upper and lower bound and lower bound in numerical value. Function with global minimum is drawn and the final results – global minimum, minimize, number of iteration and runtime are displayed in new figures (Fig. 25).

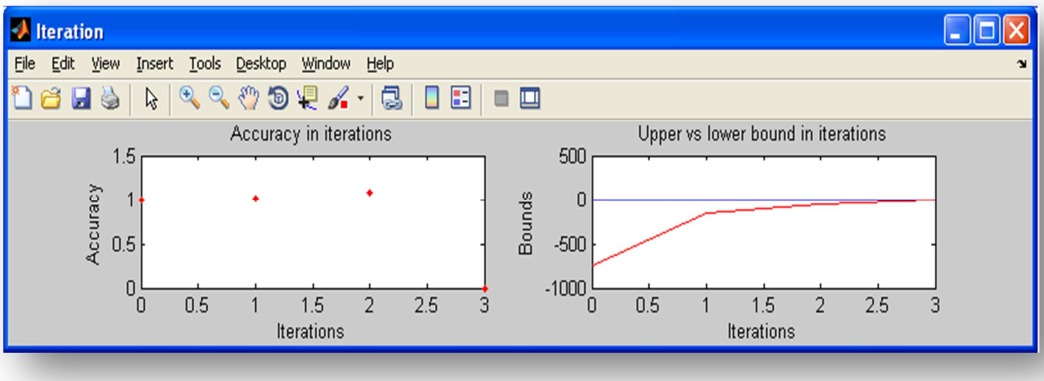

Fig. 17: Figure of dependence of accuracy and distance between the bounds

3.1.2 Deterministic multi-dimensional global optimization

Problems of multi-dimensional global optimization are more complex according to the matrix form of function derivatives. The more variables we have, the bigger derivative matrices are. Our function *parsing.m* transforms variables from symbolic variable to string

as it was in Fig. 13. The problem's input in Fig. 18 is similar as in one-dimensional case, only upper and lower bounds have to be entered in vector form.

Before the algorithm starts searching for minimum, we can choose from various ways of α calculation (Szakálová, 2012) as in Fig. 19 and in Fig. 20. *GLOP* automatically computes first values of α for each method and displays the one with the smallest value of maximum separation distance measurement term μ_α as a recommended. For μ_α holds (Adjiman, 1998b):

$$\mu_\alpha = \frac{1}{4}\sum_i \alpha_i \left(x_i^U - x_i^L\right)^2 \qquad (3.1)$$

After submitting the chosen method, calculation starts. Two subplots in figure display dependence of accuracy and bounds from number of iterations during calculation. At the end of calculation the final results are displayed such as global minimum, minimizer, number of iterations and the runtime in seconds. According to the dimensionality of the problem, every dimension of the minimizer is displayed separately. We achieved this by using the same principle as generating objective and constraint functions' files (Fig. 15).

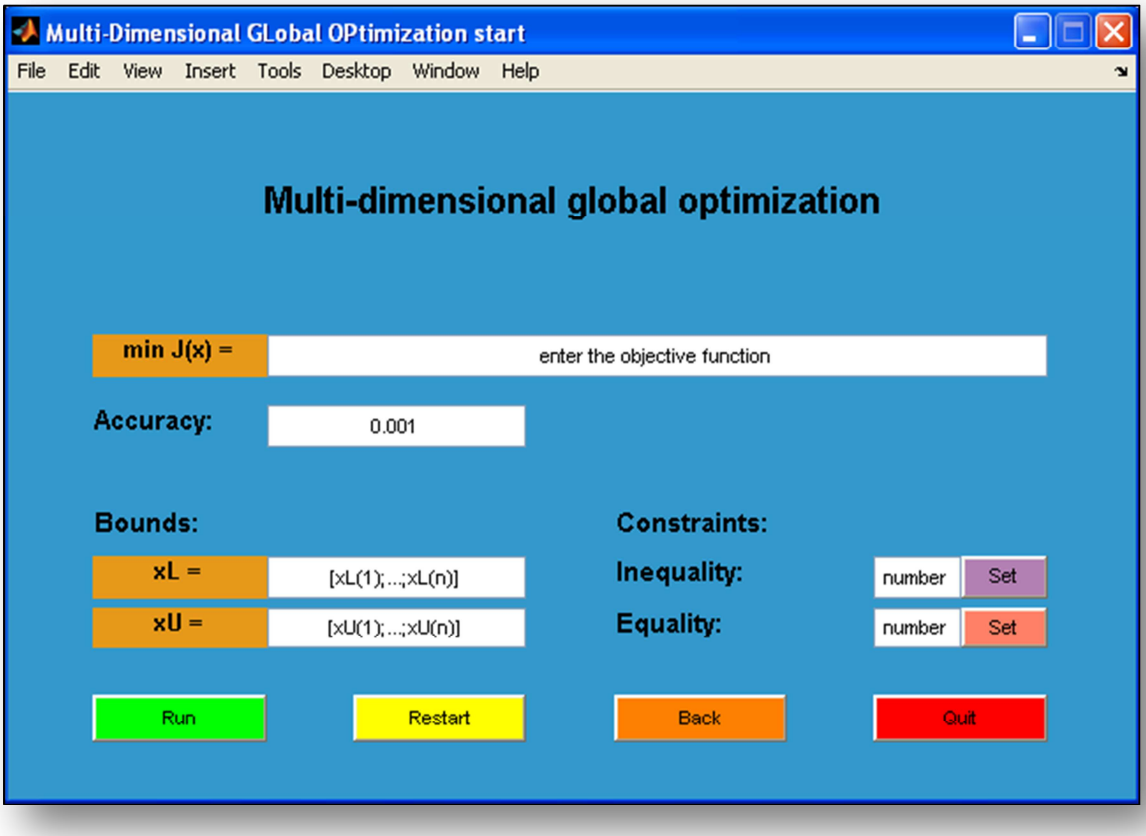

Fig. 18: Dialog window for entering multi-dimensional problem

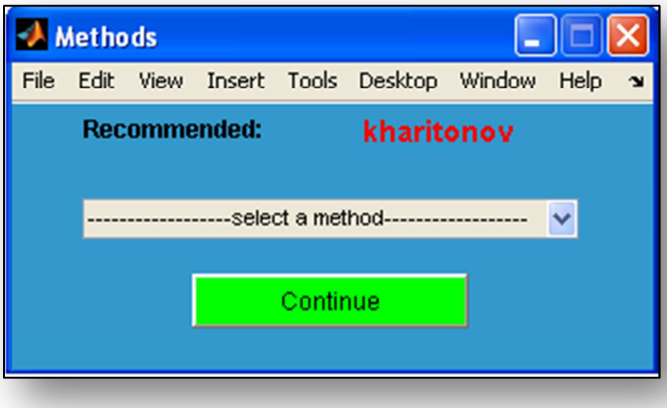

Fig. 19: Dialog window for choosing method of calculation α parameter

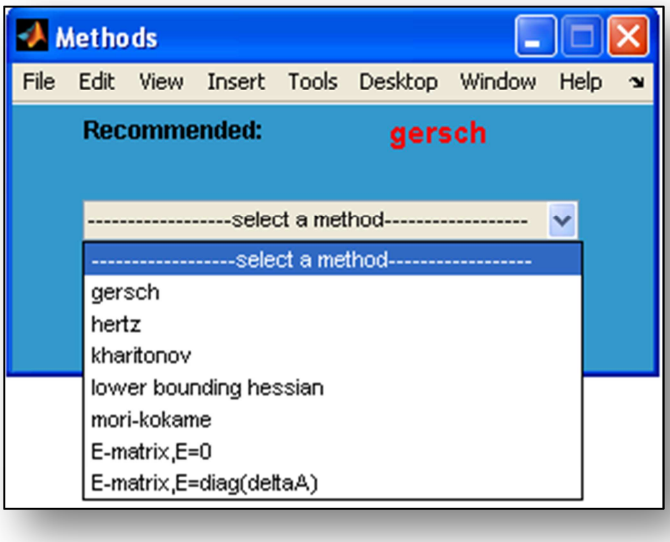

Fig. 20: Dialog window for choosing method of calculation α parameter, all methods

3.2 STOCHASTIC GLOBAL OPTIMIZATION

As stochastic algorithm, which is searching for global minimum, we chose *Multistart* method. Its main principle is, that it obtains a sequence of randomly generated starting points for NLP algorithm. In our case we have to obtain a sequence of random starting points from inputted interval constraint- $\left[x^L, x^u\right]$. An example how we do it is in Fig. 21, where variable *mul_count* represents user-defined number of starting points and is input to the *rand()* function generating random numbers from interval $[0,1]$. Then we modify it to

random numbers from our $\left[x^L, x^u\right]$ interval. For each starting point this method finds a local minimum in its vicinity. Therefore, it is a random-based method, sometimes it cannot find a global minimum.

```
x0=[];
for i=1:length(xL)
    x0_help=rand(mul_count,1)*(xU(i)-xL(i))+xL(i);
    x0=[x0,x0_help];
end
```

Fig. 21: Source code for generating starting points

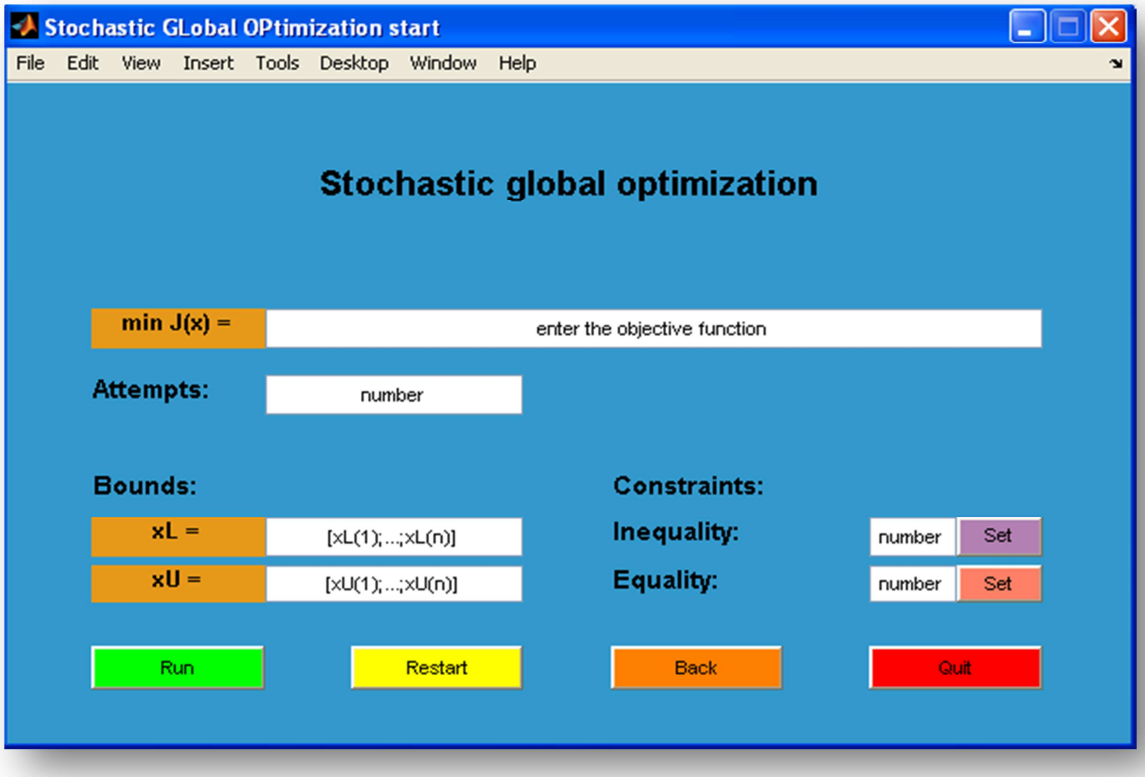

Fig. 22: Dialog window for stochastic global optimization

Menu designed for searching a global minimum by *Multistart* method in Fig. 22 displays after the selecting *Stochastic method* pushbutton at the start menu. Principle of inputting the problem, bounds and constraints is the same as in deterministic methods, even more the way of creating the objective and constraint functions' files and displaying the final results is same. The only difference is, that we have to input number of starting

points instead of accuracy. The final results are displayed in figure with parameters: global minimum, minimizer in each variable and runtime in seconds (Fig. 30). In addition to that we can see percentage representation of found local minima in histogram (Fig. 26).

4 EXAMPLES

For the testing of functionality of *GLOP*, there were solved three examples.

4.1 EXAMPLE 1

First example represents minimization of one-dimensional test function without constraints:

$$\min J(x) = 0{,}05(x - x^*)^2 + \sin^2\left[(x - x^*)^2 + (x - x^*)\right] + \sin^2\left[(x - x^*)\right]$$
$$x \in \langle -20{,}30 \rangle \tag{4.1}$$

where x^* is chosen as 1. If the *GLOP* works right, then obtained minimizer should be in x=1. Both methods, deterministic *αBB* (Fig. 23) and stochastic *Multistart* (Fig. 24) were used.

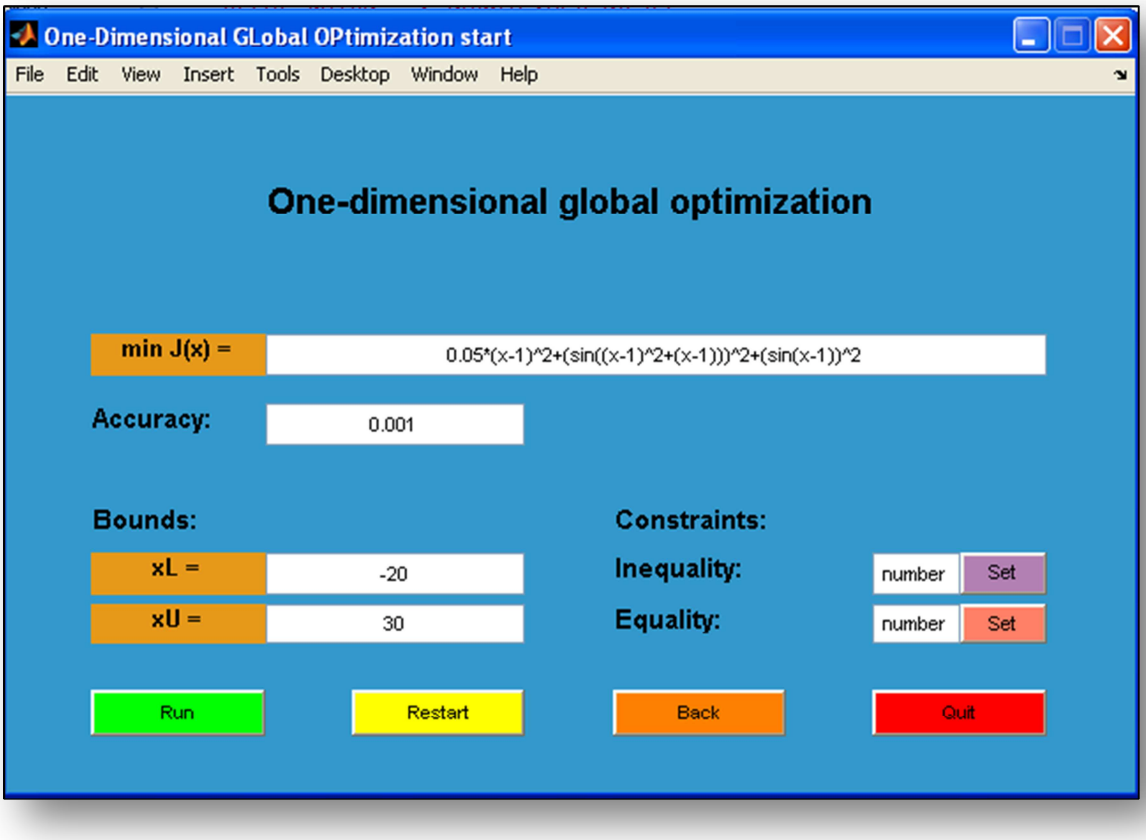

Fig. 23: One-dimensional problem without constraints, approach deterministic

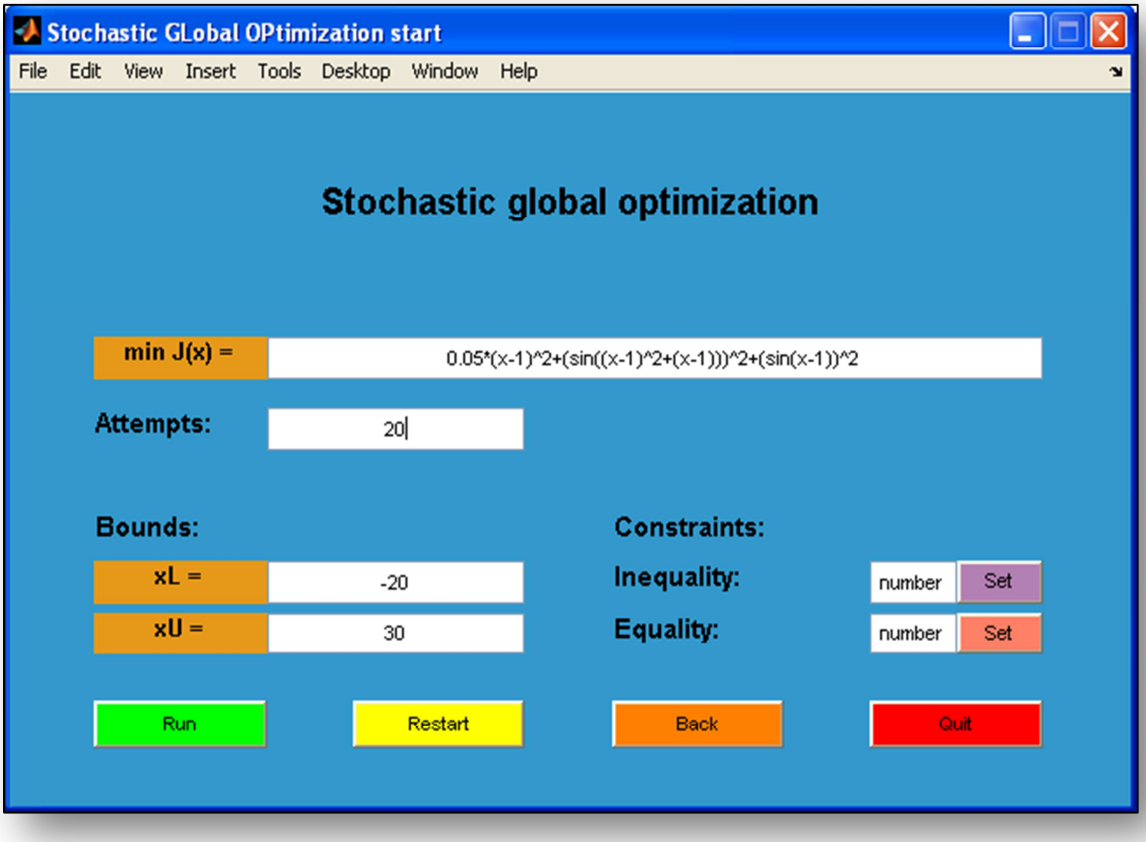

Fig. 24: One-dimensional problem without constraints, stochastic approach

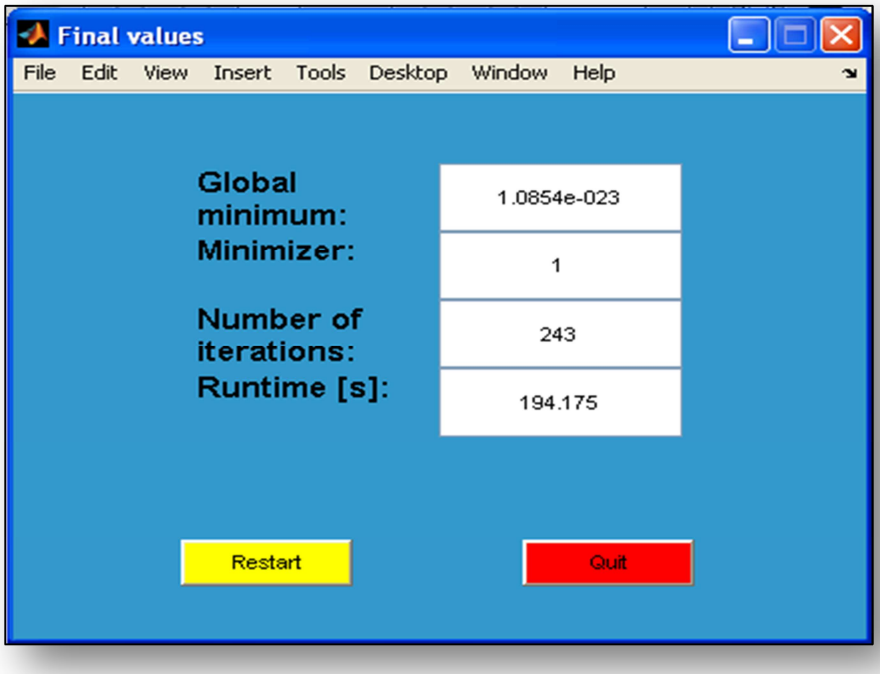

Fig. 25: Results of one-dimensional problem, stochastic approach

Because of large interval: $x^L = -20$ and $x^U = 30$ and function's nonconvexity, *αBB* method provides great values of calculation time and iterations as seen in Fig. 25. We are sure, that we acquired right solution, because our minimizer is really x=1. This result is also supported by Fig.26, where we can see whole function in chosen bounds with marked global minimum.

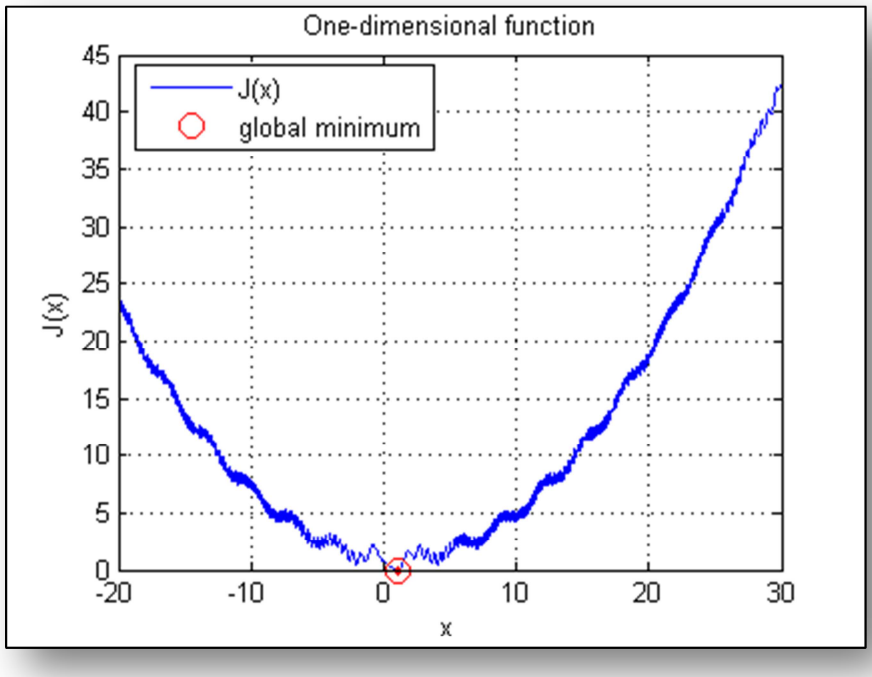

Fig. 26: Process diagram of test function, example 1

Multistart method acquired good results, too. We tested it with various values of starting points. Method provided good results, even by twenty starting points we obtained our global minimum in ratio 35% to all found local minims as seen in Fig. 26. The results are displayed in Table 1. We can say, that the increasing number of starting points causes increasing of calculation time and probability of finding the global minimum.

Number of starting points	$J(x_{min})$	x_{min}	**CPU [s]**	**Global minima [%]**
20	0	1	0,4106	35
50	0	1	0,8312	30
100	0	1	2,073	42
200	0	1	3,9457	42

Table 1: Results for the stochastic method, example 1

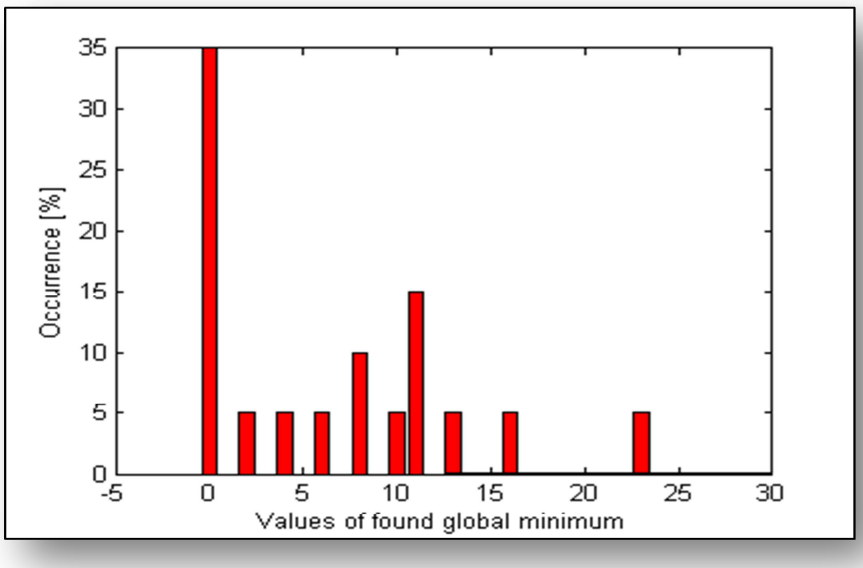

Fig. 27: Percentage of found local minims, example 1

4.2 EXAMPLE 2

As a second example, we were trying to find a minimum of Rosenbrock's two-dimensional function:

$$\min J(x) = 100\left(x_2 - x_1^{\,2}\right)^2 + (1 - x_1)^2$$
$$x_{1,2} \in \langle -2;2 \rangle \qquad (4.2)$$

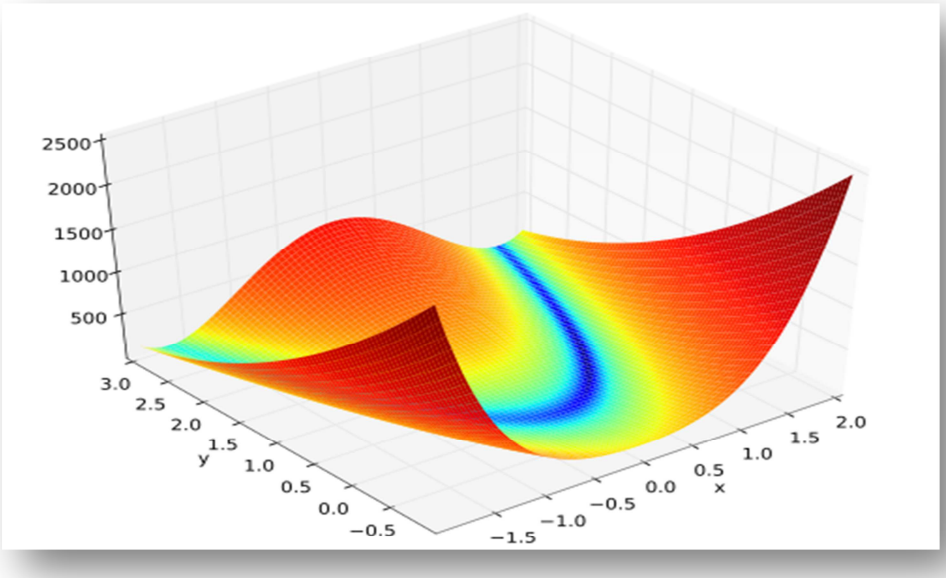

Fig. 28: Rosenbrock's function, example 2

We tested here two-dimensional optimization problem without constraint (Fig. 28), where we chose different methods of α parameter calculation. Fig. 29 shows, how we inputted data, which means entering the object function with two variables and interval constraints.

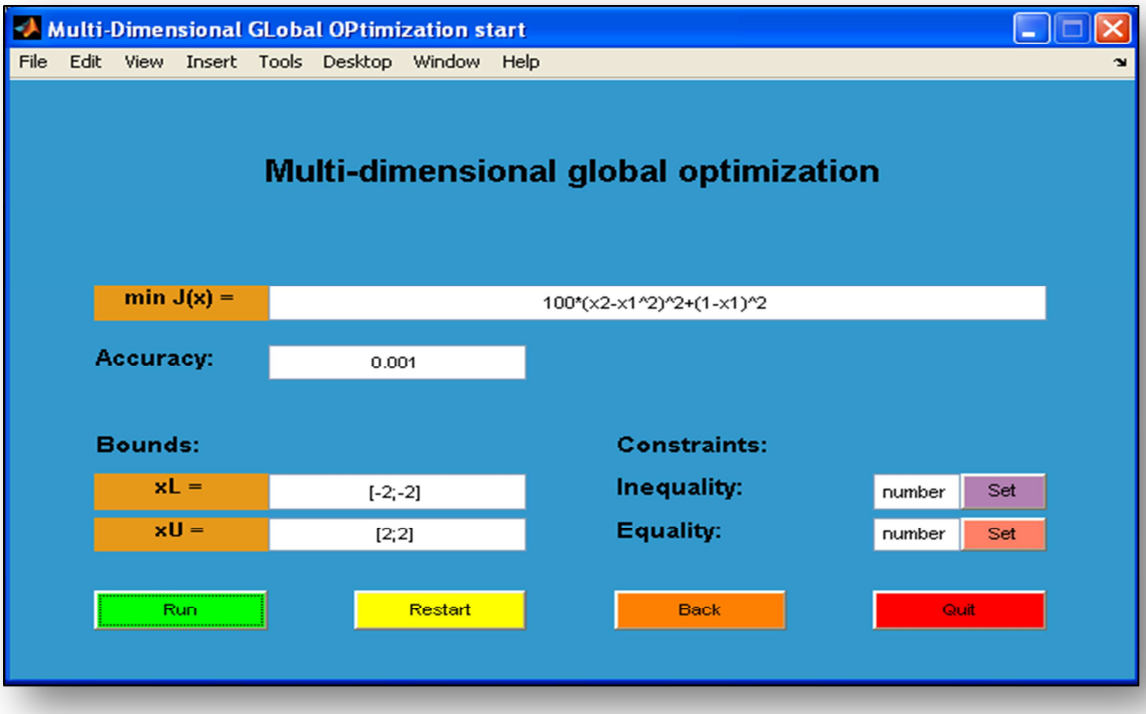

Fig. 29: Entering of two-dimensional problem without constraints, example 2

After submitting the correct data and choosing method of α calculation, we acquired results, which are presented in following Table 2. Almost all methods except *gerschogin* were able to converge to the global optimum, which is in $J(\boldsymbol{x}) = 0$ for $x = [1, 1]$.

Method	$J(x_{min})$	$x_{1,min}$	$x_{2,min}$	**CPU [s]**	**Iterations**
Gerschgorin	-	-	-	-	-
Hertz	0	1	1	66,5657	192
Kharitonov	0	1	1	292,7009	153
Lower bounding Hessian	0	1	1	65,7946	196
Mori- Kokame	0	1	1	143,5564	419
E-matrix, E=0	0	1	1	283,9082	153
E-matrix, E=diag(ΔA)	0	1	1	286,4219	153

Table2: Results for the deterministic method, example 2

We tested this function via stochastic approach, too. As we can see in Fig. 30, it obtained the same results as deterministic approach. We chose fifty starting points and the results were displayed in a very short time. According to the calculation time, we can say that this method is very effective and may be very useful in special cases.

Fig. 30: Results for the stochastic approach, example 2

4.3 EXAMPLE 3

We tested in third example solving of two-dimensional problem with equality and inequality constraints.

$$\begin{aligned} &\min J(x) = x_1^{\,3} + x_2^{\,3} \\ &h(x): x_1 + x_2 - 2 = 0 \\ &h(x): 10x_1 - x_2 + 12 \le 0 \\ &x_{1,2} \in \langle -1;2 \rangle \end{aligned} \tag{4.3}$$

Entering of the problem was similar to the two previous examples. We added two constraint functions to the corresponding windows, which were generated. In Table 3 are displayed results for the deterministic approach. Three of the deterministic methods found global minimum in one second. In Fig. 31 is displayed function diagram for this example, where is marked global minimum $J(x_{min})$. =2,0741. Results are correct, because marked point lays on the equality constraint and satisfy inequality constraint, too.

Method	$J(x_{min})$	$x_{1,min}$	$x_{2,min}$	**CPU [s]**	**Iterations**
Gerschgorin	2,0741	1,1111	0,88889	0,43062	3
Hertz	2,0741	1,1111	0,88889	0,48069	3
Kharitonov	2,0741	1,1111	0,88889	3,9557	2
Lower bounding Hessian	2,0741	1,1111	0,88889	0,46066	3
Mori- Kokame	2,0741	1,1111	0,88889	4,6523	4
E-matrix, E=0	2,0741	1,1111	0,88889	3,7354	2
E-matrix, E=diag(ΔA)	2,0741	1,1111	0,88889	3,7654	2

Table 3: Results for the deterministic method, example 3

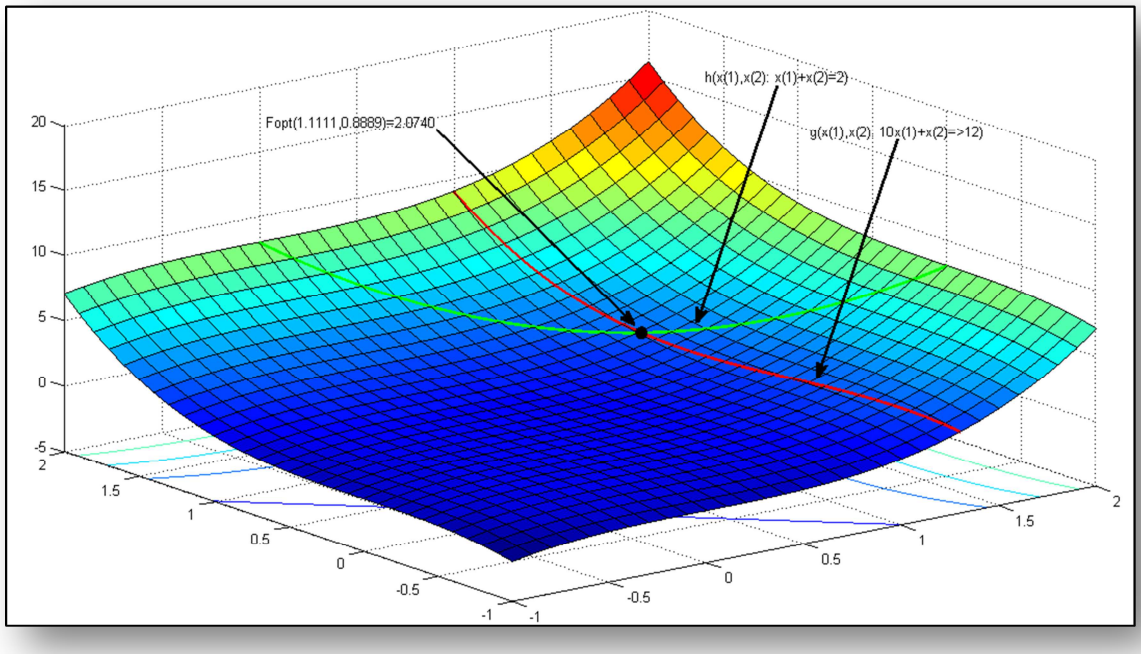

Fig. 31: Graph of test function, example 3

We tested this example by stochastic method, too. As number of starting points, we set fifty. *Multistart* algorithm found global minimum in two seconds, the results of this searching are displayed in Fig. 32 and they confirm deterministic results.

Fig. 32: Results for the stochastic approach, example 3

CONCLUSION

The main goal of this work was to create a graphical user interface in *MATLAB*'s environment for solving global optimization problems. We designed *GUI* which uses deterministic *αBB* method of global optimization for solving one-dimensional and multi-dimensional problems with constraints in certain interval.

We can easily enter the problem, constraints and bounds into the editable text fields. Then the calculation may begin. By the multivariable problems, we could select ways of computation of the parameter α. We had totally six available methods of its computation. For every α computation, we solved the problems and the results were written in tables. During calculation we could see figure, on which were two subplots. One draws dependence of accuracy from number of iterations and the other dependence of lower and upper bound from iterations, too. After the terminating of calculation, *GUI* displayed its final results in dynamically generated dialog windows, which size depended on number of variables. Than we could compare every single method subject to the acquired value of minimized objective function- global minimum, value of its minimizer, number of iterations needed for calculation and total computational time in seconds. By one-dimensional problems, there was generated a figure, which displayed function with marked global minimum, so user could confirm it by sight.

As stochastic method of searching for global minimum is the *Multistart* method. It can beat deterministic method by computational time, which is its main advantage. On the other side it does not guarantee that the obtained minimum is global. The used examples are for providing information about functionality of each approach.

For confirmation of right functionality of *GUI*, we tested it on three examples. First example was one-dimensional function without constraints with a lot of local minims. Second one was two dimensional Rosenbrock's function without constraints, where we used few methods of calculating α-parameter for deterministic optimization. Last one was two-dimensional function with two constraints. All the examples were tested by stochastic *Multistart* method, which gave same results as deterministic *αBB* method and by that the functionality of *GUI* was confirmed.

RESUMÉ

Práca je venovaná návrhu grafického užívateľského rozhrania pre riešenie problémov globálnej optimalizácie. V prvej časti opisujeme globálnu optimalizáciu, charakterizujeme jednu z jej metód a spôsoby získavania riešenia pomocou tejto metódy. V ďalšej časti uvádzame základný prehľad spôsobov návrhu grafického užívateľského rozhrania *(GUI)* v *MATLAB*-e. V tretej časti práce sa venujeme samotnému návrhu *GUI*, opisujeme jeho vlastnosti a základné funkcie. V poslednej časti riešime prostredníctvom *GUI* niekoľko vzorových príkladov.

Globálna optimalizácia hľadá extrémy funkcií v dovolenom priestore vzhľadom na ohraničenia v tvare rovností alebo nerovností. Nájdenie týchto extrémov často komplikuje prítomnosť nekonvexností, ktoré sa globálna optimalizácia snaží eliminovať. Základné delenie metód globálnej optimalizácie je na metódy stochastické a deterministické. Zo stochastických metód sa v práci venujeme metóde *Multistart*, z deterministických si podrobnejšie rozoberáme metódu vetiev a hraníc (*BB*).

Pri *BB* sa snažíme minimalizovať funkciu (1.1) vzhľadom na obmedzenia v tvare rovností (1.2) a nerovností (1.3), pričom platí, že všetky tieto funkcie musia mať spojité druhé derivácie. Platia tam intervalové ohraničenia pre premennú *x*. Na danom intervale zostrojíme podhodnocujúcu funkciu pôvodnej účelovej funkcie. Cez *NLP* algoritmy získame hornú a dolnú hranicu riešenia týchto funkcií a porovnáme vzájomnú vzdialenosť medzi nimi. V prípade toho, že je táto vzdialenosť väčšia ako je potrebná, rozdelíme pôvodný interval na dva podintervaly. Na každom podintervale opakujeme postup s podhodnocovaním a hľadáme hornú a dolnú hranicu. Ak nastane situácia, keď je horná hranica na jednom intervale menšia ako dolná hranica na inom intervale, tak interval s dolnou hranicou z výpočtov vynecháme. Intervaly delíme dovtedy, až kým sa vzdialenosť medzi hranicami nepriblíži na žiadanú hodnotu. Vtedy sa dopracujeme k riešeniu v podobe globálneho minima.

Účelovú funkciu a funkcie obmedzení v tvare nerovnosti môžeme rozdeliť podľa vzťahu (1.5) na sumu špeciálnych nekonvexných členov a všeobecných nekonvexných členov. Obmedzenie v tvare rovností podľa vzťahu (1.4) prevedieme na obmedzenia v tvare nerovností. Všetky tieto členy funkcie podhodnocujeme, pričom výsledný podhodnotiteľ pôvodnej funkcie nadobúda tvar (1.10). Pre priestorovú metódu vetiev

a hraníc platí, že podhodnocujeme len všeobecné nekonvexné členy, pričom všetky členy sa považujú za všeobecné nekonvexné. Podhodnotiteľ potom nadobúda tvar (1.11).

Závislosť Hessovej matice výsledného podhodnotiteľa je opísaná v (1.12) a závisí od súčtu Hessovej matice pôvodnej funkcie a dvojnásobku matice diagonálneho posunutia. Na identifikáciu matice diagonálneho posunutia potrebujeme poznať hodnotu parametra α, pre ktorú platí vzťah (1.14).

Grafické užívateľské rozhranie sa v prostredí *MATLAB*-u môže vytvárať dvomi spôsobmi. Prvým je vytvorenie prostredníctvom aplikácie *GUIDE*, ktorá nám umožňuje umiestňovať jednotlivé prvky *GUI* cez interaktívny editor. Kliknutím alebo potiahnutím prvkov ich umiestnime na požadovanú pozíciu, prípadne upravíme ich rozmery (Fig. 2). Výsledkom uloženia sú dva súbory, ktoré sa pri aktivácii *GUI* spustia. Druhým spôsobom tvorby *GUI* je programovacia tvorba. To znamená, že v editore *MATLAB*-u manuálne vytvoríme pre každé okno (Fig. 3) a každý ovládací prvok (Fig. 4) kód, ktorý obsahuje údaje ako pozícia, farba, písmo ,či *callback* funkcia niektorého z nich.

V našej práci tvoríme *GUI* pre riešenie problémov globálnej optimalizácie deterministickou *αBB* metódou a stochastickou metódou *Multistart* (*GLOP*). Po vyriešení problému sú zobrazené výsledky v podobe globálneho minima funkcie, bodov, v ktorých toto globálne minimum dosahuje, čas výpočtu a pri deterministickej metóde aj počet iterácií (Fig. 25, Fig. 30 a Fig. 32). Využívame pritom programovací spôsob tvorby rozhrania.

Po zadaní príkazu *glopstart* sa spustí úvodné okno umožňujúce si vybrať spôsob riešenia problému. Tvorí ho niekoľko rôznych prvkov. Prvým je už samotné okno grafu (Fig. 5), ktorého veľkosť vykresľujeme pomerovo k veľkosti našej obrazovky. Veľkosť obrazovky sa zisťuje cez funkciu *scrz(),* získané údaje potom priradíme do premenných *s_width* a *s_heigth* a použijeme ich vo funkcii *position()* upravujúcej umiestnenie okna (Fig. 6). Takisto zistíme cez funkciu *get()* rozmery okna, pretože ich potrebujeme pri umiestnení jednotlivých prvkov zobrazovaných v ňom. Prvok so statickým textom – „*Global optimization*“ (Fig. 7), informuje o účele *GLOP*. Tri tlačidlá (Fig. 4 a Fig. 8) zas predstavujú možnosti, ktoré si môžeme zvoliť ich stlačením. Všetky majú nastavenú pozíciu a písmo v pomere k rozmerom okna. Rozdiel medzi nimi je v ich *callback* funkcii, ktorá obsahuje rozdielne príkazy vykonávajúce sa po ich stlačení.

Menu *GLOP* umožňuje vybrať medzi dvoma prístupmi riešenia. Pri deterministickom prístupe si ešte vyberáme v submenu podľa typu problému, či sa jedná o jednorozmerný alebo viacrozmerný problém. Dôvodom je, že pri jednorozmernom

prípade dokážeme bez väčších problémov vykresliť priebeh účelovej funkcie s vyznačeným globálnym minimom (Fig. 26). Zvolením možnosti *One-dimensional* alebo *Multi-dimensional* si vyberieme typ problému (Fig. 9). Aktivuje sa okno (Fig. 10 alebo Fig. 18), v ktorom zadávame optimalizačný problém, teda účelovú funkciu, intervalové obmedzenia, prípadne môžeme zmeniť presnosť. Taktiež zadáme počet príslušných obmedzení, ich potvrdením sa vygeneruje okno s rovnakým počtom editovacích okien pre daný typ (Fig. 11).

Pre potreby algoritmu potrebujeme určiť prvé a druhé derivácie účelovej funkcie a funkcií obmedzení s využitím symbolického toolboxu. Príklad toho je na Fig. 12, kde si zadefinujeme symbolickú premennú *x*, následne si vytvoríme premennú *v*, podľa ktorej budeme funkcie derivovať. Pri dvojrozmernom probléme by sme mali dve symbolické premenné *x1* a *x2,* a teda premenná *v* by bola vektor. Ďalej potrebujeme funkciu a jej derivácie zmeniť zo symbolickej premennej na reťazce. Zmenou typu premennej sa však do reťazcov dostanú aj neželané znaky, ktoré odstránime využitím funkcie *parsuj()* na Fig.14, ktorú sme vytvorili za týmto účelom. Príklad takejto transformácie je na Fig. 13. Keď získame reťazce v požadovanom tvare, môžeme pristúpiť k tvorbe súborov prostredníctvom príkazov *fopen*, *fprintf* a *fclose* (Fig. 15). Fig. 16 zobrazuje vloženú účelovú funkciu alebo funkciu obmedzení zapísanú v požadovanom tvare. Toto všetko vykonávame kvôli tomu, aby sme znížili výpočtovú náročnosť tým, že minimalizujeme využitie symbolického toolboxu.

Po spustení programu sa v priebehu výpočtu zobrazuje graf závislosti presnosti od počtu iterácii a graf závislosti veľkosti hornej a dolnej hranice riešenia (Fig. 17). Presnosť si definujeme ako rozdiel hornej a dolnej hranice riešenia vydelený dolnou hranicou. Pri viacrozmerných úlohách sa nám zobrazí okno, v ktorom zvolíme metódu výpočtu parametra α (Fig. 19 a Fig. 20). Na základe kritéria zo vzťahu (3.1) sa zobrazuje najvhodnejšia metóda ako odporúčaná. Po ukončení výpočtu sa zobrazia výsledky ako v Fig. 25 a ukazujú globálne minimum, body v ktorých funkcia dosahuje globálne minimum, počet iterácií a čas výpočtu.

Pri stochastickej metóde *Multistart* využívame rovnaké zadanie údajov ako v deterministickom prípade. Rozdiel je len v tom, že namiesto presnosti zadávame počet štartovacích bodov, v okolí ktorých hľadáme lokálne minimum (Fig. 22). Na generovanie štartovacích bodov využívame funkciu *rand()*, pričom premenná *mul_count* predstavuje ich počet. Matematickými operáciami upravíme náhodné čísla tak, aby boli zo zvoleného intervalu (Fig. 21). Výsledky riešenia sa taktiež zobrazujú vo vytvorenom okne (Fig. 30

a Fig. 32). Tiež sa zobrazí histogram vyjadrujúci percentuálne zastúpenie jednotlivých lokálnych miním k ich celkovému počtu (Fig. 27).

V závere riešime tri vzorové príklady, kde demonštrujeme funkčnosť *GLOP*. V prvom prípade riešime nekonvexnú jednorozmernú úlohu (4.1). Obe metódy dokázali nájsť globálne minimum. Výsledok pre deterministickú metódu je zobrazený na Fig. 25 resp. Fig. 26., pričom je zrejmé, že výpočet nie je jednoduchý vzhľadom na zložitosť problému a veľké rozmedzie intervalového ohraničenia. Pri stochastickom spôsobe riešenia je dĺžka výpočtu priamo ovplyvnená počtom štartovacích bodov, čo však na druhej strane zvyšuje pravdepodobnosť nájdenia globálneho minima (Table 1). Druhý príklad predstavuje riešenie dvojrozmernej úlohy (4.2) bez obmedzení. Tu ukazujeme použitie jednotlivých metód získania parametra α a ich vplyv na celkový výsledok (Table 2). Pre porovnanie uvádzame aj zhodný výsledok riešenia stochastickou metódou (Fig. 30), čo potvrdzuje funkcionalitu. Tretím príkladom je dvojrozmerný problém s obmedzeniami oboch typov (4.3). Riešili sme ho rovnakým postupom ako predchádzajúci príklad. Table 3 zobrazuje riešenie deterministickými metódami, pre porovnanie takisto uvádzame riešenie stochastickou metódou (Fig. 32). Vo všetkých prípadoch výsledky potvrdzujú správnu funkcionalitu *GLOP*-u.

BIBLIOGRAPHY

Paulen, R. 2008. Global Optimization of Processes, Diploma Thesis, Bratislava: FCHPT STU in Bratislava, 2008.

Repčíková, I. 2009. Úvod do globálnej optimalizácie, Semestrálny projekt II, Bratislava: FCHPT STU in Bratislava, 2010

Černá, K. 2010. Tvorba grafického užívateľského rozhrania pre globálnu optimalizáciu, Diplomová práca, Bratislava: FCHPT STU in Bratislava, 2010

Szakálová, A. 2012. Globálna optimalizácia pomocou metódy alphaBB, Diplomová práca, Bratislava: FCHPT STU in Bratislava, 2012

Adjiman, C. S., Androulakis, I. P. Floudas, C. A. 1998a. A global optimization method, αBB, for general twice-deffernentiable constrained NLPs –II. Implementation and computational results. Computers and Chemical Engineering, 22(9):1159-1179, 1998a

Adjiman, C. S., Androulakis, I. P. Floudas, C. A. 1998b. A global optimization method, αBB, for general twice-deffernentiable constrained NLPs –I. Theoretical advances. Computers and Chemical Engineering, 22(9):1137-1158, 1998b.

Floudas, C. A., Akrotirianakis, I. G., Caratzoulas, S. Meyer, C. A., Kallrath, J. 2005. Global optimization in the 21st century: Advances and challenges. Computers and Chemical Engineering, 29(2005)1185-1202, 1998b.

www.mathworks.com. 2000-2012. Creating graphical user interfaces. 2012