International Institute for Applied Systems Analysis
IIASA www.iiasa.ac.at

Interactive Optimization Systems

Kelevedzhiev, E. and Kirov, N.

IIASA Working Paper

WP-89-007

March 1989

WORKING PAPER

INTERACTIVE OPTIMIZATION SYSTEMS

Emil Kelevedzhiev
Nikolaj Kirov

March 1989
WP-89-007

IIASA
International Institute
for Applied Systems Analysis

INTERACTIVE OPTIMIZATION SYSTEMS

Emil Kelevedzhiev
Nikolaj Kirov

March 1989
WP-89-007

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
A-2361 Laxenburg, Austria

Foreword

The existing software for nonlinear programming has been designed under two assumptions: that the user is a highly qualified programmer as well as a specialist in optimization, and the software will be utilized on the mainframe computer. Therefore, building application programs and solving real life problems is a rather complicated task. Due to growing popularity of microcomputers, it is necessary to make the optimization software available for users who are specialists in their application domain, but who are not well trained in software engineering.

This paper presents the software system consisting of several modules which allow easy access to a rich library of optimization modules. The basic components of the system constitute tools for definition of the optimization problem using simple mathematical notation as well as symbolic transformation and gradient computation (module SYSTRA), editing numerical data (module EDITA) and interactive control of optimization process (module DIALOG).

Alexander Kurzhanski
Chairman
System and Decision Sciences Program

INTERACTIVE OPTIMIZATION SYSTEMS

Emil Kelevedzhiev and Nikolaj Kirov

Institute of Mathematics
Sofia
Bulgaria

Introduction

The project "Interactive Optimization Systems" is developed by collaborators of the Institute of Mathematics with Computing Center at the Bulgarian Academy of Sciences, and the Department of Mathematics and Mechanics at Sofia University "Kliment Ochridski". As planned, the study will comprise three stages of development. So far, the contract signed between the Bulgarian National Committee for Applied System Analysis and Management (BNC ASAM) and the International Institute for Applied System Analysis (IIASA) concerns only stage I (1986 - 1987).

According to the working plan a project for a family of interactive optimization systems was developed. The project is implemented on a microcomputer IBM PC. It reflects a certain methodology which is based on both the world gained experience and the experience of our specialists in their long practice of applying mathematical methods and optimization software.

The main features of the projected family of dialogue systems are listed below:

- Unification of the man-machine interaction. This implies that all the dialogue systems from the family will maintain similar ways of interaction with respect to both program tools and performance. Our aim is to facilitate the non-experienced user in his work with this software - once accustomed to a particular system the user will need minimal efforts to start working with another system from the family.
- Orientation to user's professional background. This means that any dialogue system will be easily adapted to the set of practical problems and user's professional level. The goal again is a facilitation for the user - the dialogue will be led in terms of his

language and in a correspondence with his experience.

- Availability of the program tools which permit description and a thorough editing of the problems in a natural way to the user. In most of the existing interactive optimization systems such tools are either missing or insufficiently developed. The dialogue systems belonging to the family will offer different kinds of program means for description and thorough editing which corresponds to the specific practical problems.
- Process orientation, i.e. a possibility for thorough intracement and control in the process of problem solving (the computational process). This aspect seems to be the weakest one in the design of most of the existing interactive optimization systems. In this area not a lot of experience is gained although such a possibility (the complete control of the computational process) appears to be extremely necessary for many classes of optimization problems. The maintenance of that possibility will be one of the main advantages of the family of interactive optimization systems.
- Openness of the libraries. The library module of each dialogue system from the family will be open for including new numerical algorithms as well as for excluding certain ones due to the system adaptation to a specific practical set-up.
- Openness of the family. The family of dialogue systems could be (and will be) open for including new optimization systems according to demands from the practice.

The family of interactive optimization systems is implemented in a high level programming language, so that its transportation to other kinds of computers will be possible.

According to the needs of optimization software, the following interactive systems will be created during the three stages of the project:

- Dialogue system for unconstrained function minimization and function minimization with lower and/or upper bounds for the variables;
- Dialogue system for solving transportation problems;
- Dialogue system for solving linear programming problems;
- Dialogue system for solving discrete optimization problems.

The development of dialogue optimization systems and systems for decision support has its own history. One of the first such systems is described in [1]. Theory, software and testing examples for interactive decision support systems are given in [2].

1. Main Results

The project comprises the following activities:

A) Creation of the base software for the family of dialogue systems, i.e. the technological tools which are common for all the dialogue systems from the family. These tools concern the problem description, the problem editing and the control of the computational process.

Working out the basic part of the family of dialogue systems is an important work whose successful fulfillment will imply the further development of any particular dialogue system. The base software comprises the means and the features of the man-machine interaction, and the means and features stipulate the capacities of the dialogue systems themselves.

The base software includes:

- Symbol transformation system. The system design provides a description of the optimization problem by means of formulae, i.e. the respective functions are introduced in an analytical form. A project of the system was developed and according to it not only input of symbols and treatment of functions in an analytical form is allowed but also elementary transformation of expressions, finding out partial derivatives of functions etc. are possible. This project was realized and the created system bears the symbolic name SYSYTRA.
- Table editor of data. The editor provides a description of problems which are set as a finite sequence of data being from different (but fixed) types. Such problems arise in the linear programming, the discrete optimization programming, transport programming etc. A project of the editor with a whole complex of facilities concerning input, editing and data storage was developed. It is here named EDITA.
- Technological means maintaining the man-machine interaction. A dialogue language and the corresponding screen editor (DIALOG) were designed. The screen editor allows: control of the computational process, choice of numerical method, pausing the computation and analyzing the current situation, varying the control parameters and continuation of the process.

B) Creation of one dialogue system from the family. As a first implementation, the dialogue system for unconstrained function minimization and function minimization with lower and/or upper bounds for the variables has been chosen. The system is meant for solving optimization problems from the following classes:

- minimization along a given direction (one dimensional minimization);
- multidimensional non-constrained minimization;
- multidimensional minimization with lower and/or upper bounds for the variables.

The answer to the question which system had to be implemented first was dictated by the following considerations:

- The unconstrained minimization problems are typical representatives from the class of optimization problems whose successful solving depends each time on the control of the computational process (choice and combination of methods, choice of values of the methods' parameters etc.), i.e. they constitute a "nice" class of problems for approbation and demonstration of the base software;
- Many methods for other classes of optimization problems (for instance: constrained function minimization, optimal control etc.) in part employing one-dimensional and/or unconstrained function minimization problems.

Creation of libraries with a sufficiently large choice of methods is a base for further development of the other dialogue systems from the family.

2. SYSYTRA

2.1. Introduction

SYSYTRA is a package of procedures that perform analytic differentiation and other simple symbolic manipulations. These procedures are written in high-level language PASCAL/TURBO for IBM PC/XT microcomputer, and have certain advantages over existing and more comprehensive packages. The SYSYTRA procedures are easily interfaced with the user's programs. The library of SYSYTRA procedures is described. These procedures may be implemented in computer programs for numerical computations and optimization. Ideas on how the procedures might be embedded in other programs to create a new one are described.

2.2. Purpose

One of the main questions concerning the problem of finding out a minimum of a given function is how to describe this function. Moreover, there are many instances where it is necessary to differentiate a function $f:R \rightarrow R$ one or more times. If a package that

performs analytical differentiation is not available, the user must either carry out the differentiation by hand, a tedious and error-prone process, or compute a numerical approximation to the derivatives. In many instances both alternatives are unacceptable.

In most of the existing systems the user have to write a subroutine (in some programming language as PASCAL, FORTRAN, etc.) for calculation of the function values and the values of its partial derivatives. In the general case such a work is difficult - it requires certain mathematical skillfulness and practical knowledge of writing computer programs.

The efficiency of minimization methods is essentially dependent on the time of calculation of the function value and its partial derivatives. It is important to ensure that the function and its partial derivatives have possibly simplest forms. In most cases the simplification of arithmetic expressions is difficult to be done by the user.

The SYSYTRA package removes all these inconveniences. It might be used in any dialogue system as an admissible alternative for function description.

2.3. History

The development of systems for symbolic (analytical) transformations has a 30 year history. Up to date there are a lot of such systems, a part of which is described in the book [3]. It is known that the present day systems for symbolic transformation can find out the derivatives of functions, simplify the expressions as well as compute symbolically the integrals, factorize terms etc. There are systems for solving symbolically algebraic and differential equations as well as systems for special symbolic computation for electronic circuits, as an example.

A symbolic manipulation system is implemented in Nonlinear Model Generator designed by IIASA project [4].

Such systems allow to increase the number of users of computer programs. Moreover, the system for symbolic transformation grants the user an absolute precision of produced symbolic representations (derivatives, etc.).

The time of computation is naturally growing up when the symbolic transformation is used. However, the time of finding out manually the derivatives, its simplification and writing a program for function computation is obviously greater than the time of running the symbolic transformation package.

Several packages for performing symbolic computation are well known. MACSYMA and REDUCE are probably the best of them. However, these packages are, to a large de-

gree, "isolated" in the sense that they do not readily interface with programs written by the user. The common feature of most of the systems is its closed-loop design. They require a large operating memory and high-speed computers. They are not intended for personal computers. Most of them do not work as dialogue systems. No parts of them might be used as a part of user programs.

2.4. Main Features

The system for symbolic transformation SYSYTRA has the following features:

1. All procedures are written in the high-level language PASCAL, version known as TURBO-PASCAL, implemented by Borland International in [5] (v.3.1 for microcomputer IBM PC/XT). The SYSYTRA needs a minimal configuration of this computer.
2. The system SYSYTRA is composed by procedures, such that every one of them might be used independently, and is easy to be incorporated in other programs.
3. The main possibilities included in SYSYTRA are those for symbolic differentiation and simplifications of functions, defined by algebraic expressions. The syntax of expressions is closely related to that commonly used in such a language as PASCAL. Briefly speaking the input expressions for SYSYTRA are a combination of functions (including sin, cos, exp etc) of one, two, or more variables connected by algebraic operations: addition, subtraction, multiplication, division, taking a power. The arguments of the functions may be expressions too.

2.5. Description of the set of processing functions

The SYSYTRA package manipulates the set F of functions f that have the following definition:

Definition:

Let X be a given set, the elements of which may be represented by a computer, and on which the binary operations $+:X\times X\rightarrow X$; $-:X\times X\rightarrow X$, $*:X\times X\rightarrow X$,/ $:X\times X\rightarrow X$, $\hat{}:X\ X\rightarrow X$ are defined. A function $f:X^h \rightarrow X$ belongs to F if and only if it can be represented as the last in a finite sequence of functions $\{f_\alpha\}$ that are such that:

1^0 each $f_\alpha \in F$

2^0 if $x = (x,\ldots,x_n) \in X^n$, then there exist $f_{\alpha j}$, such $f_{\alpha j}(x) = x_j$, $j=1,\ldots,n$;

3^0 each $f_\alpha \neq f_{\alpha j}$ has one of the forms

$$f_{\alpha_1} + f_{\alpha_2}$$

$$f_{\alpha_1} - f_{\alpha_2}$$

$$f_{\alpha_1} * f_{\alpha_2}$$

$$f_{\alpha_1} / f_{\alpha_2}$$

$$f_{\alpha_1} \hat{} f_{\alpha_2}$$

$$g_1(f_{\alpha_1}(x))$$

$$g_2(f_{\alpha_1}(x), f_{\alpha_2}(x))$$

$$g_3(f_{\alpha_1}(x), f_{\alpha_2}(x), f_{\alpha_3}(x))$$

where

$$f_{\alpha_1}, f_{\alpha_2}, f_{\alpha_3} \in F,\ g_1 \in E_1,\ g_2 \in E_2,\ g_3 \in E_3\ .$$

The sets E_1,E_2,E_3 are given sets of functions by one, two or three variables respectively. For examples

$$E = \{-(.), sqrt\ (.), exp\ (.), \ln\ (.), sin\ (.), cos\ (.),\ arctan\ (.)\}.$$

The sets E_1,E_2 and E_3 may be extended to include any other function from X to X (or from $X\times X$ to X, or from $X\times X\times X$ to X) that may be evaluated in a particular computing environment provided that its derivative is itself in F.

Some remarks about definition of the set F

1. The similar definition is given in the paper [6], where the set F is called "a set of computable functions".
2. The elements of base set X are divided into two classes: The first one is the so called "constants", and the second one "independent variables". To evaluate a function $f \in F$, we have to know current values of this "independent variables".

3. The set of function F is obviously closed with respect to the operations: addition, subtraction, multiplication, division and taking a power. Moreover F is a closed relative to taking an arbitrary superposition of function of F.

4. Taking a power follows the definition:

$$x \wedge y = \begin{cases} \exp(y \times ln x) & , \ x>0 \\ \underbrace{x \times \cdots \times x}_{y \ times} & , \ x<0 \text{ and } y \textit{ is an integer} . \end{cases}$$

5. Owing to the nature of the differentiation operator, the partial derivative of a function of F with respect to any of the variables is itself a function of F.

An example

Consider the function $f:X^5 \rightarrow X$ defined by

$$f(x,p,a,b,c) = \sin(x+p\hat{}(ifgt(a,b,c)) ,$$

where the function ifgt $\in E$:

$$\text{ifgt}(a,b,c) = \begin{cases} b, & \text{if } a>0 \\ c, & \text{otherwise} \end{cases}$$

The function f belongs to F, since we may write:

$$f_1 = a, \ f_2 = b, \ f_3 = c, \ f_4 = p, \ f_5 = x ,$$

$$f_6 = \text{ifgt}(a,b,c)$$

$$f_7 = p\hat{}\text{ifgt}(a,b,c) = f_4 \wedge f_6$$

$$f_8 = x+p\hat{}\text{ifgt}(a,b,c) = f_5 + f_7$$

$$f_9 = \sin(x+p\hat{}\text{ifgt}(a,b,c)) = \sin(f_8) .$$

Clearly f is equal to the last in a finite sequence of functions that satisfies the conditions of the definition.

To evaluate the function f let suppose that current values of "independent variables" are $x=-1,p=5,a=2,b=0,c=3$. Then

$$\sin(-1+5\hat{}\text{ifgt}(2,0,3)) = \sin(-1+5\hat{}0) = \sin(0) = 0.$$

2.6. Data Structures (Internal representation)

Given an expression that defines a function of F, SYSYTRA generates the sequence of functions $\{f\}$ that make up its form according to definition and then stores this sequence in the computer memory. The function once stored in this way, may then be differentiated, evaluated, output as a string, or composed or combined with other functions that are similarly stored. This section describes the data structures used to store the finite sequence $\{f\}$. Any term in the sequence is one of the following:

a constant (i.e. of the form of integer or real number);

a variable (i.e. of the form of x_1,x_2,...);

a ternary term (i.e. of the function ifgt(a,b,c));

a binary term (i.e. x_1+x_2 , or a function of two variables);

a unary term (i.e. $\sin(x)$).

To store a constant we need only to store the value of this constant. To store a variable we need to store its name and current value. Storing unary, binary or ternary terms is slightly more complicated. An unary term contains an argument and an operator, where the argument is another term in the sequence. We could therefore store an unary term in a data structure consisting of a string and pointer; the string represents the unary opera- tor, and the pointer points to the argument. Similarly, a binary or ternary term could be stored in a structure composed of the operator in a string, and pointers to each of the subterms. This gives rise to a binary tree (or more correctly an acyclic graph) where each node represents a term in the sequence $\{f_\alpha\}$ and each leaf node is a constant or a variable. The head of the tree represents the last term in the sequence $\{f_\alpha\}$. The function of the example is represented by the tree structure shown in Figure 1.

2.7. Type Declarations

The type declarations necessary in a Pascal implementation of the SYSYTRA for real-valued function are given below.

```
const maxvar=30;
type  funstrType=string[255];
      string6=string[6];
      varvalType=array[1..maxvar] of real;
      varnameType=array[1..maxvar] of string6;
      tpnode = -1..3;
      mix = record
```

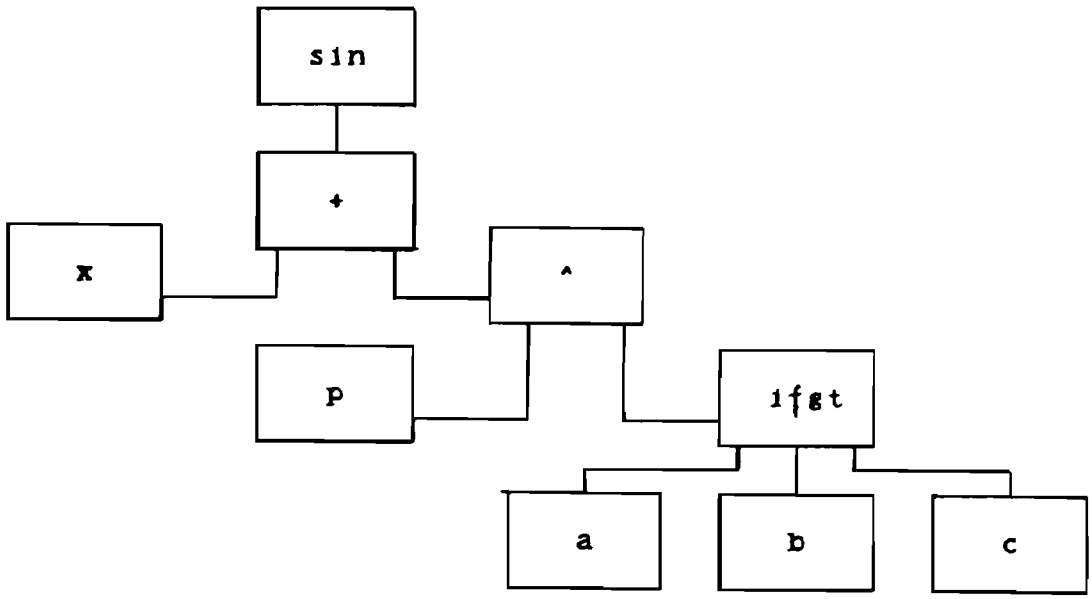

Figure 1.

```
case tpnode of
-1: (i: string6);
0: (n: real);
1,2,3: (o: string6);
end
pointerType = ^ node;
node = record
flag : byte;
tp   : tpnode;
name : mix
arg1 : pointerType;
arg2 : pointerType;
arg3 : pointerType;
end;
```

In the type declaration it is assumed that no more than 30 variables will be used and that all variable names will be six characters or less. This, of course, could be altered to

meet specific requirements. The names and values of variables are stored in arrays of type varnameType and varvalType respectively. The tree representation of a given function is defined by type node. This type is a record, whose fields have the following meaning: *flag* is needed for storing a temporary information; *name* contains a numerical value or a name according to *tp*. On *tp* = 0 *name* is a real constant. On *tp* = -1 *name* is a name of an independent variable. On *tp* = 1,2,3 *name* is a name of a function of 1,2,3 variables respectively. The fields *arg1*, *arg2*, *arg3* points to the arguments.

2.8. Description of Procedures

The procedures that make up the SYSYTRA package are described in this section. The type declaration given in the preceding section, and the appropriate type declaration for any variables, are assumed. The procedure *functiontree*, which has a heading of the form

functiontree (funstr: funstrType; var pointer: pointerType; var err:integer);

takes as input a string representation of a given function stored in *funstr*. The tree representation of the function is created, and a pointer *pointer* to it is returned. The output variable *err* contains an error code. If *err* $\neq$ 0, then there is a syntax error in string representation of a function and a tree structure is not created.

The procedure *writefun*, which has a heading of the form

writefun (pointer: pointerType; var outfunfile: text);

in some sense reverses the process of *functiontree* and converts a function held in its tree representation and *pointed* at by pointer into a string stored in *outfunfile*. Bracketing is used to avoid ambiguity. The procedure *newtree*, which heading is

newtree (oldpointer: pointerType; var newpointer: pointerType);

makes a copy of the tree representation of the function pointed at by *oldpointer*. The output variable *newpointer* stores the pointer to new tree representation. This procedure is needed for some of the following procedures that change the input tree representation.

Once the tree representation of a function has been created, it may be evaluated by the procedure *funvalue*, which has a heading of the form

funvalue (pointer: pointerType, varname: varnameType; varval: varvalType; dim: integer; var: err: integer): real;

This procedure takes as input a pointer *pointer* to the three representation and arrays *varname* and *varval*, which store the names and current values at the variables. *dim* is an

actual number of function arguments. The value of the function at the given point *varval* is returned as*funvalue*. The output variable *err* contains an error code. If *err* $\neq$0 then there is no errors.

The procedure differfun (pointer: pointerType; resp: string6; var err: integer);

takes a pointer *pointer* to the tree representation of a function and creates a new tree representation of the partial derivative of the function with respect to variable that name is stored in *resp*. The *pointer* becomes the pointer to the tree representation of the partial derivative. The output variable *err* contains an error code. If err $\neq$0 then there is no errors. Thus the statements

resp = 'x'; differfun (pointer, resp, err);

would create the tree structure representing ∂ f/∂ x and return a pointer to it.

The procedure funsimple (varpoint: pointerType; depth: integer);

takes a pointer to the tree representation of the function, simplifies this function, and returns a pointer to new tree representation. The simplification might be done on different levels according to an input parameter *depth* = 0,1,2,3.

3. EDITA

3.1. Introduction

EDITA is a software product for creation, observation and correction of data introduced in form of two-dimensional tables. It is a flexible and easy to use editor for creation and modification of problem data bases.

The procedures of EDITA are written in high-level language PASCAL/TURBO for IBM PC/XT microcomputer.

EDITA is suitable for implementation in computer program for solving linear programming problems, quadratic programming problems etc., and any other problems that use two-dimensional tables as input data.

3.2. Purpose

The great many of the optimization problems which arise in practice frequently have a fixed data structure - the data being composed of vectors and matrices in the respective dimensions and given in form of two- dimensional tables. The linear programming problems are of this type, as well as the problems in the quadratic programming, the discrete

optimization programming, the transportation problem etc. As a rule, the problems of the above types are big-sized and the input, correction and storage of data are time consuming actions for the user. It is therefore important that any dialogue system designed to solve such problems should possess tools allowing a program management of these actions. The table editor EDITA appears in this role.

3.3. Main Features

The editor is meant for creation, observation and correction of data introduced in form of two-dimensional tables. Any place of the table might be filled with numeric or symbolic data, according to the user's desire, and any filled place could be banned for further corrections. Filling in the table might be done in the course of calculation of a certain function defined by the user. For large tables only a part of the table could be displayed and then the user decides which part to make visual. It is possible to enlarge or to reduce the size of the table (the number of rows and columns). Another possibility allows to find a place by its name. All contents of the table are easily written in a file or loaded from a file. The data transfer to a computational procedure and back is possible too.

EDITA is suitable for use on IBM-PC, IBM-XT, IBM-AT and IBM compatible microcomputers. The minimal amount of memory required is 256 K. A minimum of one floppy disk drive is required, although it is certainly easier to use EDITA (as is true of most large packages) if you have two floppy drives or a fixed disk drive. DOS 2.0 or higher is required for the operating system to support the editor.

3.4. Structure

The editor EDITA consists of one program module. The user may expand it with program modules for solving a chosen class of optimization problems. For this purpose, the user might set the editor to work in terms of the specific class of optimization problems. This may be done by writing appropriate procedures in PASCAL.

The file system of EDITA is quite simple. The all data created by EDITA may be saved to a file with name extension '.tbl'. The structure of this file is mixed: The real numbers are stored in binary representation (in the same way as internal representation in TURBO PASCAL: one byte for exponent, and fifth following bytes for mantissa), while the symbolic data are stored as strings, each symbol according to ASCII table of codes.

The '.tbl' file may be loaded from a disk and edited by EDITA, or converted by a specific procedure in order to transfer it to a chosen module for finding a solution of a given optimization problem.

3.5. The Menu System

The main concept for control of EDITA is controlled by menus. The user may choose some function from the main menu or go to the next menu. The whole menu system is presented below. The menu windows appear in the upper right corner of the screen. The user may select an item from the menu by the following method: The first is to use the Up and/or Down Arrow keys located in the Numeric Keypad area to move the pointer (red arrow) in the menu window to point to a selected item. Once the pointer is on the selected item, it is enough to press the Enter key to make that selection.

In each mode of operation the editing table is on the screen. One of the table areas appears in reverse video. This reverse video box is called the **Current Location Pointer**. Its function is to identify the active area of the table for user.

Main Menu

When the Main menu has appeared, the user may select the following possibilities:

RArr
LArr
DArr
UArr
<any char key>
F1 - help
F2 - redraw
F3 - rotation
F4 - page
F5 - define
F6 - change input
F7 - compute function
F8 - compute function in memory
F9 - input/output menu
F10 - change table

Description

EDITA starts with the Current Location Pointer in the upper left corner of the table. For entering the Problem data you may move the Pointer using the arrow keys (Up, Down, Right, Left) whose abbreviation are: UArr, DArr, RArr, LArr) to choose an area and then press any character or number key, according to the type of the area. At this moment you are in **INPUT** mode (See INPUT mode). The abbreviation F1..F10 stand for function keys. When solving a large problem, not all the data will fit on the screen simultaneously. The amount that can be displayed in one screen is called a **page**. The extra columns and rows can be imagined as being in their proper positions off screen, and can be reached by continuing to move in the appropriate direction using the Arrows. For example the right Arrow which is used repeatedly, will carry you to the edge of the screen. Press the Right Arrow key again, and the screen will scroll sideways, so that your view shifts one column to the right. All columns move one place left, with the leftmost column disappearing from the view. Vertical movement is similarly available. With large data arrays, it may be desirable at time to move the Pointer more rapidly than one cell at a time. To provide this press **F4 key**. Then Arrow keys will cause the display to scroll one complete page up, down, left or right. To return the usual function of Arrow keys press **F4 key** once more.

Pressing **F3 key** will cause changing of a function of Arrow keys in a similar way but slightly differently: Pressing an Arrow key will cause a "rotation" in all columns (rows) that follow Current Position Pointer. In this way you may see on the screen no necessary successive columns (or rows).

The F2 key redraws the start position of the table.
For using F5, F7, F8 see **Computing Functions**.
The F5 key enters in **Define Menu**.
The F6 key enters in **Change Input Menu**.
The F9 key enters in **Input/Output Menu**.
The F10 key enters in **Change Table Menu**.

INPUT mode (menu)
<any char>
BSP
DEL
ENTER
RArr
LArr

DArr
UArr

Description

Having typed number (or string) there are several ways to transfer it to table. Use the Enter key. This will automatically advance the **Pointer** to the next cell down or right, or cause no moving, depending on setting Input mode (see change input mode menu). Another way is to use the Arrow keys, which now will act to enter and shift one position in the indicated direction. Using them you can work down one column and up the next, back and forth across successive rows, or more among the cells in any convenient order. If you press the wrong key while typing data and notice it right away, press the Back space key (BSP) to wipe out the last character. If the whole entry is beyond repair, the DEL key will clean out the Current Position and put back the old Value.

Change Input menu
input
input to right
input to down

Define menu
define function
autofun table
autofun memory
find
repeat find
How to use Computing functions see section 3.6.
How to use Find see section 3.7.

Change table menu
type of a field
make a banned field
new row
new column
delete row
delete column

Description:

"type of a field" option allows you to change the field type: numeric or symbolic. "make a banned field" option allows you to lock the field for further corrections.

The next four options allow you to insert and delete additional rows or columns. The problem dimension, such as the number of variables and constraints in a linear program is most important. Very often we will want to consider the effect of adding a source and so on. EDITA permits you to do this easily, by adding or removing a row or column in the Data table.

To add a column, place the Current position Pointer anywhere in the column immediately left of the desired location. To enter the **Change table menu**, select the "new column" and press ENTER. This will cause all data on the right to move one cell further right, making way for the new column and scrolling if necessary to bring it on to the screen. On the other hand "delete column" will, after a warning, eliminate the column on which the **Current Location Pointer** is located. The corresponding row operations "new row" and "delete row" are exactly analogous.

Input/Output Menu
Save and Solve
rename
load
save
quit
This menu is for saving, loading or renaming the data file,
for solving the problem or for exiting EDITA.

3.6. Editing Data using Functions

Special features of EDITA have been designed to facilitate altering a large array of data in certain ways. Entering the **Define Menu** you may select "define function" option to input a function string (see section 2 for more details). The prompt line will ask you to enter the function. When you press the Enter key an internal representation of the function is created and stored for using. There are four ways to compute the function.

1. Pressing F7 key the numeric value of computation is written in **Current Location**. If you want to move the **Current Location Pointer** using an Arrow, each cell visited will automatically be fulfilled with the computational value, after pressing F7 key

2. Pressing F8 key the numeric value of computation is written in special location called **memory**.
3. Entering the "Define Menu", then select "autofun Table" option. This cause automatically all the cells to be visited and each of the cells to be filled with the computational value.
4. Entering the "Define Menu", then select "auto fun in memory" option. This will automatically cause all the cells to be visited and every time the special location "memory" to be filled with the computational value. Using the last two ways of computation you may decrement all cells of the table by a constant, or compute the sum of the all numeric cells, etc.

List of the special variables and function utilized in function definition

i - number of row
j - number of column
s - value of the Current Location Position
p - value of the previously visited Cell
m - value of the special location "memory"
rnd - value of a random number in [0,1), uniformly distributed
~ - logical negation
& - logical AND
| - logical OR
^ - taking a power
if(c,x,y) - function that returns x if c=1 and returns y otherwise.

The next functions have its usual meaning (as in language PASCAL): +(), -(), sqr(), sqrt(), sin(), cos(), exp(), ln(), atn(), abs(), int(), +, -, *, /, =, <, >.

3.7. Find command

This command allows the user to find a symbolic field by its name. The Current Location Pointer is moved at this field. When this field is off screen this command will automatically cause to scroll complete pages up, down, left, or right in such way that the

Current Location Pointer is on screen.

To activate this command enter the Define menu, then select "find" option. The prompt line ask you to enter the name and then the field with this name is found. If such a field does not exist a "beep" sound is heard. The "repeat last find" option reactivates the command.

4. DIALOG

4.1. Introduction

The screen editor DIALOG is a software product which provides program implementation of a number of menu and commands (they constitute the dialogue language) concerning the information output on the screen and the management of the computational process.

The procedures of DIALOG are written in high-level language PASCAL/ TURBO for IBM/XT microcomputer.

DIALOG is intended for implementation in computer program for solving nonlinear optimization problems with or without constrains, as well as for solving other problems that have no fixed structure of input and output data.

4.2. Purpose

Usually the input/output information for various classes of optimization problems differs on its type and quantity and might not always be displayed on the screen. For this reason it is important to give the user an opportunity to select and arrange the information needed by type and quantity. The DIALOG responds to the above requirements. It serves as a convenient tool for displaying and changing the current variables and parameters involving the computational process, as well as for control of this process (e.g. algorithms for minimization).

4.3. Main Features

The screen editor is meant for creation, observation and correction of data that take part in computational process. The main capacities are:

1. Distribution of places field with different data by type and number of different pages (screens) according to the user's choice.
2. Displacement of data on the screen according to the user's choice.
3. Change of names and contents of places.
4. Writing in and loading from a file the names, values and disposition of places.

The DIALOG permits starting and pausing of the computational process in order to facilitate display of the current values of the variables. In case the user has not enough qualification and professional skill he may load a standard files with selected places for input/output information about displaying process.

DIALOG is suitable for use on IBM-PC, IBM-XT, IBM-AT and IBM compatible microcomputers. The minimal amount of memory required is 256 K. A minimum of one floppy disk drive is required, although it is certainly easier to use DIALOG (as is true with most large packages) if you have two floppy drives or a fixed disk drive. DOS 2.0 or higher is required for the operating system to support the screen editor.

4.4. Structure

The screen editor DIALOG consists of one program module. The user may expand it with program modules for solving a chosen class of optimization problems. For this purpose, the user might be make a setting the DIALOG to work in term of the specific class of optimization problems. This may be done by making an inventory of all data needed for the specific class.

The current version of DIALOG consists of the following principal procedures:

1. KeybInput - procedure for input and processing keyboard information for control and change a computational process.
2. ScreenOut - procedure for output the screen information during the time when computational process is active.
3. MainInput - procedure for reading and processing main input file, with initial data for computational process.
4. SymbMan - procedure for symbolic manipulation of input string for function description. Needed for finding out a gradient of the function.

5. A family of procedures for processing the data on auxiliary files (see section 4.5). For each of the auxiliary files there is a procedure for creating a standard data, for editing this data by user, for writing in and for reading from the file.
6. CalcContr - procedure for control process of computation.
7. Some number of auxiliarily procedures, e.g. timer etc.

4.5. Files

The screen editor DIALOG file system consists of a main data file, which describes the problem of unconstrained optimization and five auxiliary files. The main data file might be created by every text editor. To start the interaction with DIALOG, the user has only to supply this file. The auxiliary files contain an information about the disposition of the fields on a screen, the names of these fields, etc. and they must be created by the screen editor DIALOG.

Main input file

The main input file is the only obligatory amount of information the user has to supply before he starts the interactions with DIALOG. During the session the user may be given some more details about the problem. The data that are not specified by Main input file are loaded from auxiliary files. If the auxiliary files do not exist, the data needed are setting to its default values. Creating the main input file, the following sections has to be present:

1. Information about the name and the type of the problem: Reserved word "name" followed by any string of up to 6 characters, and reserved word "type" followed by a similar string.
2. Information about the names and initial values of the variables, parameters and constants involving the problem. Reserved words "var", "par" or "const" followed by the name and initial value.
3. Description of function in the following form: identifier of a function - string representation.

Example

To solve the unconstrained optimization problem:

find a minimum of f(x,y) = (x-a)^2 +(y-b)^2,

where a,b are some parameters (let a=1,b=2), start point x=3, y=4, the user has to write the following Main input file:

```
name          Problem Name;
type          UCM;
var           x=3, y=4;
par           a=1, b=2;
f=(x-a)^2+(y-b)^2;
find min      f;
```

Auxiliary files:

1. File for storing the names and values of each field. Its name has extension ".dat".
2. File for storing the information about disposition of the fields on pages and its location on every page. Extension :".scr".
3. File for storing the help information. Its name extension is ".hlp".

4.6. Data Description

The information about the computational process are displaying on the screen as fields. Every field has two parts: Name of the field and Value. According to the data type the fields might represent reals, integers or symbol strings. According to the purpose of data the fields might be:

1. Names of methods running at the current time;
2. Parameters of these methods;
3. Information concerning the process management and its observation on the screen;
4. Data concerning the computational process (current values of the variables, the object function, its gradient etc).

Using the data display or stop the process of computation in order to change some of the parameters of the computational methods (initial point, tolerances, etc.). The main purpose is to speed up the convergence.

4.7. The menu system

The main concept for control of DIALOG is control by menus. The user may choose some function from the main menu or go to the next menu. The whole menu system is briefly presented below.

In each mode of operation the data fields are displayed on the screen. One of the data fields is marked out with a pointer (red arrow). Its function is to identify the active field of the data for correction by user.

Main menu bar

Calculation

Rename

Load

Save

Arrangement

Quit

Control keys

DArr, UArr, RArr, LArr, PgUp, PgDn,

F1, F2, F9, F10

Description

DIALOG starts with Pointer in the first datum field. For entering the new datum you may move the Pointer using the Up Arrow and the Down Arrow key to choose a field and the press on Left Arrow or Right Arrow keys to select the name or the value part of the field. At this moment you are in INPUT mode (see INPUT mode). To select a page you may use PgUp or PgDn Keys.

The F1 key gives help about the system.

The F2 key gives help about the chosen datum field.

The user may select an item from the Main menu bar by the following method: The first is to use the F9 and/or F10 keys to move the light box in the Main menu bar to point to a selected item. Then it is enough to press the '+' key located in the right down side on the keyboard to make the selection.

The "Calculation" item starts the computational Process. This Process may be stopped by pressing Alt key.

Choosing the "Rename" option you are prompted to input a new name for auxiliary files. Under this name these files will be loading or saving.

The "Load" and "Save" items serve for loading and saving the information concerning the names, values, displacement and disposition of the fields. This information is writing in and loading from files with extension '.dat' and '.scr'.

The "Arrangement" item is meant for displacement of data on the screen, using the RArr, LArr, UArr, DArr, End keys.

The "Quit" exits the DIALOG.

INPUT mode

<any char>

BSP

DEL

ENTER

Description

Having typed a number or a string you may transfer it to data fields pressing the ENTER key. If you press the wrong key while typing data and notice it right away, press the Backspace key (BSP) to wipe out the last character. If the whole entry is beyond repair, the DEL key will clean out the current position and put back the old value.

5. Library of Methods for Unconstrained Minimization and Minimization with Lower and/or Upper Bounds for the Variables

The dialogue system consists of a symbol transformation system (SYSYTRA)) - see 2.), screen editor (DIALOG - see 2.) and a library of procedures implementing methods for solving those classes of optimization problems which the system is designed for. The main feature of these classes of problems is the lack of universal solving methods. For this reason the system library includes various methods. Often, in the process of solving a particular problem, it is necessary to replace one method by another, or, to change the values of the same method's parameters.

The library contains programs that realize methods for solving the following problems:

A) One - dimensional minimization.

$$f(x) \longrightarrow \min, a \leq x \leq b,$$

where

$$a,b \in [-\infty,+\infty], f:R^1 \longrightarrow R^1$$

The following methods are included:

A1) Method of golden section.

The algorithm realizes the well-known golden section line search, see e.g. [14]. First, the minimum point x^* is localized, then a sequence x_k is constructed that converges to the point x^* on the basis of the golden section rule.

A2) Method of the one-sided parabolic interpolation.

The algorithm makes a parabolic approximation of the objective function and uses its minimum for generating a better point.

First, the minimum is localized in an interval [a,b], in which a "parabolic triple" (a, x, b) is known. Then a sequence of intervals $[a_k,b_k]$, is constructed, such that $x^* \in [a_k,b_k]$.

A3) Method of the stabilized parabolic interpolation.

The algorithm uses a parabolic approximation of the objective function, combined with a number of stabilizing rules for choosing the next approximation of the solution (in case of unsuccessful parabolic step). It is extracted from [16].

A4) Method of the cubic interpolation.

The algorithm is based on a cubic interpolation of the objective function using the values of this function and its derivatives at two points. First, the minimum point x^* is localized in an interval [a,b], which satisfies one of the conditions:

(i) $f'(a) < 0, f'(b) > 0;$

(ii) $f(a)<0, f(b) \geq f(a).$

Then a sequence of intervals $[a_k,b_k]$ is constructed, such that b_k-a_k tends to zero with k and $x^* \in [a_k,b_k]$.

B) Unconstrained minimization.

$$f(x) \longrightarrow \min, x \in R^n ,$$

where $f:R^n \longrightarrow R^1, R^n$ is the n-dimensional Euclidean space.

The following unconstrained minimization methods are included:

B1) Nongradient methods:

B1.1) Rosenbrock method [15].

Let $S_1^k,\ldots,S_n^k$ be a set of orthogonal directions in R^n and x^k be the point obtained at the k-th step (x^0 given, S_i^0 the unit vectors $e_i, i=1,\ldots,n$). Doing steps along each direction a new point x^{k+1} is generated, such that $f(x^{k+1}) < f(x^k)$. Then the new directions are constructed by Gram-Schmidt procedure applied to the vectors $d_1^k = x^{k+1} - x^k = \lambda_1^{k*} s_1^k + \cdots \lambda_n^{k*} s_n^k$, $d_2^k = \lambda_2^{k*} s_2^k + \ldots + \lambda_2^{k*} s_n^k, \ldots$, $d_n^k = \lambda_n^{k*} s_n^k$, where λ_i^k is the sum of all successful steps along $s_i^k, i=1,\ldots,n$.

B1.2) Rosenbrock method with line search [14].

This is a modification of the above algorithm in which the steps $\lambda_i, i=1,\ldots,n$ along the directions $s_1^k,\ldots,s_n^k$ are obtained by a line search method

B1.3) Powell method.

The method uses the following basic idea [13]: Let $S_1^k,..,S_n^k$ be n linearly independent vectors in R^n and x^k be the point obtained in the k-th step (x^0 is given). One minimizes sequentially the function in the directions $S_n^k, S_1^k, \ldots, S_n^k$ and $S_{n+1}^k = (x_n^k - x_0^k) \; / ||x_n^k - x_0^k||$. The obtained point is x^{k+1}. If $||x_{n+1} - x_0|| < \epsilon_x$ then stop, else the basis is changed by taking a new direction S_{n+1}^k and dropping the direction in which the line search step is maximal.

B2) Gradient methods:

B2.1) Fletcher - Reeves conjugate gradient method [7].

B2.2) Polak - Ribiere conjugate gradient method [8].

B3) Quasi-Newton methods:

B3.1) Wolf - Broyden - Devidon variable metric method [10].

B3.2) Fletcher - Powell - Devidon variable metric method [9].

C) The above methods are modified to also solve problems with lower and/or upper bound for the variables

$$f(x) \longrightarrow \min, x \in R^n,$$

$$a_i \leq x_i, i \in J_1,$$

$$x_i \leq b_i, i \in J_2,$$

where J_1 and J_2 are subsets of the set of indices $1,\ldots,n$. It is possible to have $a_i = b_i$ for some $i \in J^*$, i.e. in the case when the constraining set has an empty interior and lies in a subspace (determined by the set of indices $\{1,\ldots,n\} J^*$. In

this case the minimizing sequences generated by the algorithms do not leave this subspace.

The constraints are taken into account in one of the following two ways:

C1) Projection of the direction on the domain given by the bounds for the variables.

The idea of the method is to generate the direction d and the new point x as in the unconstrained minimization and then project it on the constraining set [11].

C2) Projection of the point obtained by unconstrained optimization on the domain defined by the variables.

The idea is to generate at every step a descent direction (as in the unconstrained case) and then to project it on the constraining set. This is an extension of the Rosen projection gradient method [12].

All the gradient and Quasi-Newton methods are adapted to solve constrained problems by both projection of the point and of the direction. The non-gradient methods use a projection of the point only.

6. Conclusion

This paper presents a software system for interactive optimization. This system includes: A system for unconstrained function minimization and function minimization with lower and upper bounds for the variables. A system for solving linear programming problems. A symbol transformation system provides a description of the problem by means of formulae and finding out partial derivatives. A table editor of data provides a description of data being from fixed types and a screen editor for control of computational process.

REFERENCES

[1] N.N. Moiseev and others, (1978). Optimization Methods (in Russian), Moscow, Nauka publ.

[2] A. Lewandowski, A. Wierzbicki Eds., Theory, Software and testing examples for decision support systems, IIASA Working paper WP-87-26, Laxenburg, Austria, pp.163-176.

[3] (1983) Computer Algebra. Symbolic and Algebraic Computation. Edited by B.Buchberger, G.E.Colins and R.Loas, Springer-Verlag Wien New York.

[4] J.Paczynski, T.Kreglewski, Nonlinear Model Generator, In: A. Lewandowski, A. Wierzbicki Eds., Theory, Software and testing examples for decision support systems, IIASA Working paper WP-87-26, Laxenburg, Austria, pp.163-176.

[5] (1985). Turbo Pascal Reference Manual. Borland International Inc.

[6] J.M. Sheaker and M.A. Wolfe. ALGLIB, (August, 1985). A simple symbol-manipulation package. Communication of the ACM Volume 28 Number 8.

[7] Fletcher R., Reeves C.M., (1964). Computer J. 7, 149.

[8] Polak E., Ribiere G. Rev. (1969). Fr. Infolrm. Rech. Operations 16-R1(1969) 35-43.

[9] Fletcher R., Powell M.J.D., (1963). Computer J., 6, 163.

[10] G.G.Broyden, Math. Computation, 21(1967), 368-381.

[11] Levitin E.S., Polyak B.T., (1966). Journal Comp. Math. and Math. Physics, 6, 787-823.

[12] Rosen J.B., SIAM J., (1960). Appl. Math. 8, 181-217.

[13] Powell M.J.D., Computer J., 7, 155 (1964); 7, 303 (1965).

[14] Bazaraa M.S., Shetty C.M., (1979). Nonlinear programming. Theory and Algorithms, John Wiley.

[15] Rosenbrock H.H., (1960). Computer Journal, 3, 175-184.

[16] Gill P.E., Murray W., Wright M.H., (1981). Practical Optimization, Academic Press.