WORKING PAPER

IAC - DIDAS - N: A DYNAMIC INTERACTIVE DECISION ANALYSIS AND SUPPORT SYSTEM FOR MULTICRITERIA ANALYSIS OF NONLINEAR MODELS WITH NONLINEAR MODEL GENERATOR SUPPORTING MODEL ANALYSIS

T. Kreglewski
J. Paczynski
J. Granat
A.P. Wierzbicki

December 1988
WP-88-112

IIASA
International Institute
for Applied Systems Analysis

IAC - DIDAS - N: A DYNAMIC INTERACTIVE DECISION ANALYSIS AND SUPPORT SYSTEM FOR MULTICRITERIA ANALYSIS OF NONLINEAR MODELS WITH NONLINEAR MODEL GENERATOR SUPPORTING MODEL ANALYSIS

T. Kreglewski
J. Paczynski
J. Granat
A.P. Wierzbicki

December 1988
WP-88-112

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

Foreword

This paper is one of the series of 11 Working Papers presenting the software for interactive decision support and software tools for developing decision support systems. These products constitute the outcome of the contracted study agreement between the System and Decision Sciences Program at IIASA and several Polish scientific institutions. The theoretical part of these results is presented in the IIASA Working Paper WP-88-071 entitled *Theory, Software and Testing Examples in Decision Support Systems*. This volume contains the theoretical and methodological bacgrounds of the software systems developed within the project.

This paper presents the user documentation for decision analysis and support systems of DIDAS family designed for supporting decision problems when the model of the system under study can be formulated in terms of set of nonlinear equations. The program presented in the paper, called IAC-DIDAS-N is provided with a nonlinear model generator and editor that support definition, edition and symbolic differentiation of nonlinear models for multiobjective decision analysis. A specially introduced standard of defining nonlinear programming models for multiobjective optimization helps to connect the model generator with other parts of the system. Optimization runs involved in interactive, multiobjective decision analysis are performed by a new version of nonlinear programming algorithm specially adapted for multiobjective problems. This algorithm is based on shifted penalty functions and projected conjugate directions techniques.

An attachment to this paper presents user documentation for a pilot version of a nonlinear model generator with facilities for symbolic differentiation and other means of fundamental model analysis.

Alexander B. Kurzhanski
Chairman
System and Decision Sciences Program

I A C - D I D A S - N
A Dynamic Interactive Decision Analysis and Support System for Multicriteria Analysis of Nonlinear Models with Nonlinear Model Generator supporting model analysis

T. Kreglewski, J. Paczynski, J. Granat, A. P. Wierzbicki

Institute of Automatic Control, Warsaw University of Technology

Contents

1 Extended summary

In many complex decision problems involving economic, environmental and technological decisions as well as in complex engineering design, decision maker needs some help of an analyst or a team of them to learn about possible decision options and their predicted results. The team of analysts frequently summarizes its knowledge in the form of a *substantive model* of the decision situation that can be formalized mathematically and computerized. While such a model can never be perfect and cannot encompass all aspects of the problem, it is often a great help to the decision maker in the process of learning about novel aspects of the decision situation and gaining expertise in handling problems of a given class. Even if the final decisions are typically made judgementally — that is, are based on holistic, deliberative assessments of all available information without performing a calculative analysis of this information, see (Dreyfus, 1984) — the interaction of a decision maker with the team of analysts and substantive models prepared by them can be of great value when preparing such decisions.

In organizing such interaction, many techniques of optimization, multicriteria decision analysis and other tools of mathematical programming can be used. To be of value for a holistically thinking decision maker, however, all such techniques must be used as supporting tools of interactive analysis rather than as means for proposing unique optimal decisions and thus replacing the decision maker. The decision analysis and support systems of DIDAS family — that is, Dynamic Interactive Decision Analysis and Support systems, see e.g. (Lewandowski et al., 1983, 1987) — are specially designed to support interactive work with a substantive model while using multicriteria optimization tools, but they stress the learning aspects of such work, such as the right of a decision maker to change his priorities and preferences when learning new facts. DIDAS systems can be used either by analysts who want to analyze their substantive models, or by teams of analysts and decision makers, or even by decision makers working alone with a previously defined substantive model; in any case, we shall speak further about *the user* of the system.

There are several classes of substantive models that all require special technical means of support — see (Lewandowski et al., 1987). The IAC-DIDAS-N version is designed to support models of multiobjective nonlinear programming type. While some nonlinear DIDAS versions have been developed before, they did not follow any standards of defining such models, since such standards did not exist. In order to support the work with a user that is not necessarily a specialist in computer programming and nonlinear optimization programming, it has become necessary to introduce such standards.

Models of multiobjective nonlinear programming type specify, firstly, the following classes of variables: *input variables* that can be subdivided into *decision variables* (that is, means of multiobjective optimization) and *parametric variables* (that is, model parameters that are kept constant during multiobjective analysis but might be changed during parametric or sensitivity analysis) — and *outcome variables* that can be subdivided into *floating outcomes* (either used as model constraints or only used for the easiness of definition of the nonlinear model or having only informative importance for the user) and *optimized outcomes* or *objectives* (the ends of multiobjective optimization that can be either maximized or minimized or stabilized, that is, kept close to a desired level). Actually, the distinction between various types of outcome variables is not necessarily sharp and the user might change their classification and select his objectives among various outcome variables when defining the multiobjective analysis problem.

For all input and outcome variables, a reasonably defined nonlinear model should include

lower and *upper bounds*, that is, reasonable ranges of admissible changes of these variables. Moreover, an essential part of a nonlinear model definition are *model equations*, that is, nonlinear functions that define the dependence of all outcome variables on input variables. To make the model definition easier for the user, it is assumed that outcome variables are defined consecutively and that they can depend not only on input variables, but also on previously defined outcome variables. However, all outcome variables must be defined explicitly.

There are many examples of decision problems that can be analyzed while using a substantive model of multiobjective nonlinear programming type; for example, DIDAS-type systems with multiobjective nonlinear programming models have been used in analyzing various environmental or technological problems (see Kaden, 1985, Grauer et al., 1983). As a demonstrative or tutorial example, IAC-DIDAS-N uses a multiobjective nonlinear programming model of acid deposition in forest soil (see Hettelingh and Hordijk, 1987). The user can also define substantive models of multiobjective nonlinear programming type for his own problems and analyze them with the help of IAC-DIDAS-N.

A typical procedure of working with the IAC-DIDAS-N system consists of several phases.

In the first phase, a user — typically, an analyst — defines the substantive model and edits it on the computer. In earlier versions of nonlinear DIDAS-type systems (which were mostly implemented on bigger mainframe computers) this phase has not been explicitly supported in the system and the user had to separately prepare (define and edit) his nonlinear model, typically in the form of a FORTRAN procedure that contained also user-supplied formulae for the derivatives of all outcome functions with respect to decision variables. It is a known fact that most mistakes in applying nonlinear programming methods are made when determining derivatives analytically; thus, this form of preparation of a substantive model required rather much experience in applications of nonlinear programming.

The new features of IAC-DIDAS-N are, firstly, the definition and edition of substantive models in an easy but flexible standard format of a spreadsheet, where the input variables correspond to spreadsheet columns and the outcome variables — to spreadsheet rows; special cells are reserved for types of variables, lower and upper bounds for all variables, as well as reference levels (reservation levels for stabilized outcomes, aspiration and reservation levels for maximized and minimized outcomes), for results of various optimization computations, etc. However, another unique new feature of IAC-DIDAS-N is an automatic support of calculations of all needed derivatives by a symbolic differentiation program. The user does not need to laboriously calculate many derivatives and to check whether he did not make any mistakes; he must only define model equations or outcome functions (possibly in a recursive, but explicit form) and make sure that these functions are differentiable and admissible for the symbolic differentiation program — which admits functions from a rather wide class. The spreadsheet format currently implemented does limit somehow the size of substantive models that can be defined in it, but reasonable models of nonlinear programming type that can be usefully analyzed on microcomputers should not be too large anyway; on the other hand, the spreadsheet format allows also for display of computed values of automatically determined formulae for derivatives in appropriate cells. The user of IAC-DIDAS-N can also have several substantive models recorded in special model directories, use old models to speed up the definition of a new model, etc., while the system supports automatically the recording of all new or modified models in the appropriate directory.

In further phases of work with DIDAS-type systems, the user — here typically an analyst working together with the decision maker — specifies a multiobjective analysis problem related to his substantive model and participates in an initial analysis of this problem. There might be many multiobjective analysis problems related to the same substantive model: the

specification of a multiobjective problem consists in designating optimized outcomes (objectives) between outcome variables, defining whether an objective should be minimized, or maximized, or stabilized — kept close to a given level. Moreover, the user can also shift bounds on any outcome when specifying a multiobjective analysis problem.

For a given definition of the multiobjective analysis problem, the decisions and outcomes in the model are subdivided into two categories: those that are *efficient* with respect to the multiobjective problem (that is, such that no objective can be improved without deteriorating some other objective) and those that are inefficient. It is assumed that the user is interested only in efficient decisions and outcomes (this assumption is reasonable provided he has listed all objectives of his concern; if he has not, or if some objectives of his concern are not represented in the model, he can still modify the sense of efficiency by adding new objectives, or by requiring some objectives to be kept close to given levels, or by returning to the model definition phase and modifying the model).

One of main functions of a DIDAS-type systems is to compute efficient decisions and outcomes — following interactively various instructions of the user — and to present them for analysis. This is done by solving a special parametric nonlinear programming problem resulting from the specification of the multiobjective analysis problem; for this purpose, IAC-DIDAS-N contains a specialized nonlinear programming algorithm called *solver*. Following the experiences with previous versions of nonlinear DIDAS systems, a robust nonlinear programming algorithm based on shifted penalty functions and projected conjugate directions techniques was further developed for IAC-DIDAS-N.

However, a multiobjective problem definition admits usually many efficient decisions and outcomes; the user should first learn about *ranges* of changes of outcomes and *bounds* on *efficient outcomes*. This is the main function of IAC-DIDAS-N in the initial analysis phase. The user can request the system to optimize any objective separately; however, there is also special command in the system related to these functions.

The command "utopia" results in subsequent computations of the best possible outcomes for all objectives treated separately (such outcomes are practically never attainable jointly, hence the name *utopia point* for the point in outcome space composed of such outcomes). During "utopia" calculations some approximations of worst possible efficient values are also obtained. The point in outcome space composed of the worst efficient values is called *nadir point*, however its exact calculation is a very difficult computational task — for nonlinear models there is even no constructive method for such calculation. The approximation of nadir point components obtained during utopia point calculations is rather to optimistic. The decision maker or an analyst can change the nadir values obtained according to his knowledge.

The utopia and nadir points give important information to the user about reasonable ranges of (efficient) decision outcomes; in order to give him also information about a reasonable compromise efficient solution, a *neutral efficient solution* can be also computed in the initial analysis phase following a special command. The neutral solution is an efficient solution situated 'in the middle' of the range of efficient outcomes, while the precise meaning of being 'in the middle' is defined by the distances between the utopia and (the approximation of) the nadir point. After analyzing the utopia point, the nadir point and a neutral solution (which all can be represented graphically for the user), the initial analysis is completed and the user has already learned much about ranges of attainable efficient objectives and the possible tradeoffs between these objectives. Each change of the definition of the substantive model or of the multiobjective analysis problem, however, necessitates actually a repetition of the initial analysis phase.

The third phase of work with the IAC-DIDAS-N system consists in interactive scanning of efficient outcomes and decisions, guided by the user through specifying two *reference points* called *reservation point* and *aspiration point* in the objective space, i.e. *reservation levels* and *aspiration levels* for each objective; the system admits also a more simple option of specifying only one reference (aspiration or reservation) level for some or even for all objectives. The user has already reasonable knowledge about the range of possible outcomes and thus he can specify his reference levels: aspiration level that he would like to attain and reservation level that he would like to satisfy in any case. The utopia and the nadir points could be used as initial values for the aspiration point and the reservation point, respectively. If the neutral solution was calculated, then the system suggests to the user another, more adequate initial aspiration point: an unattainable outcome point closer to the efficient solutions than the utopia point, and more adequate initial reservation point: an attainable outcome closer to the efficient solutions than the nadir point.

IAC-DIDAS-N utilizes the aspiration and the reservation levels as parameters in a special achievement function coded in the system, uses its solver to compute the solution of a nonlinear programming problem equivalent to maximizing this achievement function, and responds to the user with an attainable, efficient solution and outcomes that strictly correspond to the user-specified references.

If the aspirations are not attainable and the reservations are attainable (which is a typical and recommended case), then the response of the system is a solution with attainable, efficient outcomes that are either between the aspiration and reservation points or uniformly as close as possible to the former one. If the aspirations are 'too low' (if they correspond to attainable but inefficient outcomes that can be improved), then the response of the system is a solution with outcomes that are uniformly better than the aspirations. If the reservations are 'too high' (if they correspond to outcomes that are not attainable), then the response of the system is an efficient solution with outcomes that are uniformly worse than the reservations. The precise meaning of the uniform approximation or improvement depends on *scaling units* for each objective that are defined automatically in the system basing on the differences between the utopia point, the current aspiration point and the current reservation point, therefore implicitly defined be the user. This automatic definition of scaling units has many advantages to the user who is not only relieved of specifying scaling units but also has a better control of the selection of efficient outcomes by changing reference levels in such a case.

After scanning several representative efficient solutions and outcomes controlled by changing references, the user learns typically enough either to select subjectively an actual decision (which needs not to correspond to the decisions proposed in the system, since even the best substantive model might differ from real decision situation) or to select an efficient decision and outcome proposed in the system as a basis for actual decisions. Rarely, the user might be still uncertain what decision to choose; for this case, several additional options can be included in a system of DIDAS type. Such options include two more sophisticated scanning options: a *multidimensional scanning*, resulting from perturbing current aspiration levels along each coordinate of objective space, and a *directional scanning*, resulting from perturbing current aspiration levels along a direction specified by the user (see Korhonen, 1985). Another option is *forced convergence*, that is, such changes of aspiration levels along subsequent directions specified by the user that the corresponding efficient decisions and outcomes converge to a final point that might represent the best solution for the preferences of the user. However, these additional options are not implemented in IAC-DIDAS-N, since the experience of working with DIDAS-type systems shows that these options are rarely used.

2 Theoretical manual

The standard form of a multiobjective nonlinear programming problem is defined as follows:

$$\textit{maximize} \quad [q = f(x)]; \quad X = \{\, x \in R^n : g'(x) = 0, g''(x) \leq 0 \,\} \tag{1}$$

where $x \in R^n$, $q \in R^p$, $f : R^n \to R^p$ is a given function (assumed to be differentiable), $g' : R^n \to R^{m'}$ and $g'' : R^n \to R^{m''}$ are also given functions (of the same class as f) and the maximization of the vector q of p objectives is understood in the Pareto sense: $\hat{x}, \hat{q}$ are solutions of (1) iff $\hat{q} = f(\hat{x})$, $\hat{x} \in X$ and there are no such x, q with $q = f(x)$, $x \in X$ that $q \geq \hat{q}$, $q \neq \hat{q}$. Such solutions $\hat{x}, \hat{q}$ of (1) are called, respectively, an *efficient decision* $\hat{x}$ and the corresponding *efficient outcome* $\hat{q}$. If, in this definition, it were only required that there would be no such x, q with $q = f(x)$, $x \in X$ that $q > \hat{q}$, then the solutions $\hat{x}, \hat{q}$ would be called *weakly efficient*. Equivalently, if the set of all attainable outcomes is denoted by

$$Q = \{\, q \in R^p : q = f(x), x \in X \,\} \tag{2}$$

and so called *positive cones*

$$D = R^p_+ = \{\, q \in R : q_i \geq 0, i = 1, \ldots, p \,\}, \quad \tilde{D} = R^p_+ \setminus \{\emptyset\}, \quad \tilde{\tilde{D}} = \mathrm{int} R^p_+ \tag{3}$$

are introduced (thus, $q \geq \hat{q}$ can be written as $q - \hat{q} \in D$, $q \geq \hat{q}$, $q \neq \hat{q}$ as $q - \hat{q} \in \tilde{D}$ and $q > \hat{q}$ as $q - \hat{q} \in \tilde{\tilde{D}}$), then the sets of efficient outcomes $\hat{Q}$ and of weakly efficient outcomes $\hat{Q}^w$ can be written as:

$$\hat{Q} = \{\, \hat{q} \in Q : (\hat{q} + \tilde{D}) \cap Q = \emptyset \,\} \tag{4}$$

$$\hat{Q}^w = \{\, \hat{q} \in Q : (\hat{q} + \tilde{\tilde{D}}) \cap Q = \emptyset \,\} \tag{5}$$

The set of weakly efficient outcomes is larger and contains the set of efficient outcomes; in many practical applications, however, the set of weakly efficient outcomes is decisively too large. Some efficient outcomes for multiobjective nonlinear programming problems might have unbounded *trade-off coefficients* that indicate how much an objective outcome should be deteriorated in order to improve another objective outcome by a unit; therefore, it is important to distinguish also a subset $\hat{Q}^p \subset \hat{Q}$ called the set of *properly efficient* outcomes, such that the corresponding trade-off coefficients are bounded.

The *abstract problem of multiobjective nonlinear programming* consists in determining the entire sets $\hat{Q}^p$ or $\hat{Q}$ or $\hat{Q}^w$. The *practical problem of multiobjective decision support* using nonlinear programming models is different and consists in computing and displaying for the decision maker (or, generally, for the user of the decision support system) some selected properly efficient decisions and outcomes. However, a properly efficient outcome with trade-off coefficients that are extremely high or extremely low does not practically differ from a weakly efficient outcome. Thus, some a priori bound on trade-off coefficients should be defined and properly efficient outcomes that do not satisfy this bound should be excluded. This can be done by defining a slightly broader positive cone:

$$D_\varepsilon = \{\, q \in R^p : \mathrm{dist}\,(q, D) \leq \varepsilon \|q\| \,\} \tag{6}$$

where any norm in R^p is used, also to define the distance between q and D. The corresponding, modified definition of D_ε-efficiency:

$$\hat{Q}^{p\varepsilon} = \{\, \hat{q} \in Q : (\hat{q} + \tilde{D}_\varepsilon) \cap Q = \emptyset \,\}; \quad \tilde{D}_\varepsilon = D_\varepsilon \setminus \{\emptyset\} \tag{7}$$

applies to properly efficient outcomes that have trade-off coefficients a priori bounded by approximately ε and $1/\varepsilon$; such outcomes are also called *properly efficient with* (a priori) *bound* (see Wierzbicki, 1986).

The selection of properly efficient outcomes with bound and the corresponding decisions should be easily controlled by the user and should result in any outcome in the set $\widehat{Q}^{p\varepsilon}$ might wish to attain. Before turning to some further theoretical problems resulting from these practical requirements, observe first that the standard formulation of multiobjective nonlinear programming is not the most convenient for the user. Although many other formulations can be rewritten to the standard form by shifting scales or introducing proxy variables, such reformulations should not bother the user and should be automatically performed in the decision support system. Therefore, we present here another basic formulation of the multiobjective nonlinear programming problem, more convenient for typical applications.

A *substantive model* of multiobjective nonlinear programming type consists of the specification of vectors of n decision variables $x \in R^n$ and of m outcome variables $y \in R^m$ together with nonlinear *model equations* defining the relations between the decision variables and the outcome variables and with *model bounds* defining the lower and upper bounds for all decision and outcome variables:

$$y = g(x); \quad x^{\mathrm{lo}} \leq x \leq x^{\mathrm{up}}; \quad y^{\mathrm{lo}} \leq y \leq y^{\mathrm{up}} \tag{8}$$

where $g : R^n \to R^m$ is a (differentiable) function that combines the functions f, g' and g'' from the standard formulation. Thus, $m = m' + m'' + p$; but the choice, which of the components of the outcome variable y correspond only to constraints and which correspond to objectives, is flexible and can be modified by the user. There are only inequality constraints in the definition of substantive model (9), but equality constraints for some outcomes could be easily explained as

$$y_i^{\mathrm{lo}} \leq y_i \leq y_i^{\mathrm{up}} \quad \text{with} \quad y_i^{\mathrm{lo}} = y_i^{\mathrm{up}} \quad \text{for some } i \tag{9}$$

Denote the vector of p objective outcomes by $q \in R^p \subset R^m$ (some of the objective variables might be originally not represented as outcomes of the model, but we can always add them by modifying this model) to write the corresponding objective equations in the form:

$$q = f(x) \tag{10}$$

where f is also composed of corresponding components of g. Thus, the set of attainable objective outcomes is again $Q = f(X)$, but the set of admissible decisions X is defined by:

$$X = \{ x \in R^n : x^{\mathrm{lo}} \leq x \leq x^{\mathrm{up}};\ y^{\mathrm{lo}} \leq q(x) \leq y^{\mathrm{up}} \} \tag{11}$$

Moreover, the objective outcomes are not necessarily maximized; some of them might be minimized, some maximized, some stabilized or kept close to given *stabilization levels* (that is, minimized if their value is above stabilization level and maximized if their value is below stabilization level). All these possibilities can be summarized by introducing a different definition of positive cone D:

$$\begin{aligned} D = \{ q \in R^p : \ & q_i \geq 0, && 1 \leq i \leq p' \ ; \\ & q_i \leq 0, && p'+1 \leq i \leq p'' \ ; \\ & q_i = 0, && p''+1 \leq i \leq p \ \} \end{aligned} \tag{12}$$

where the first p' objectives are to be maximized, the next from $p'+1$ until p'' — minimized, and the last from $p''+1$ until p — stabilized. The definition of the cone D_ε does not change

its analytical form (6), although the cone itself changes appropriately. Actually, the user needs only to define what to do with subsequent objectives; the concept of the positive cones D and D_ϵ is used here only in order to define comprehensively what are efficient and properly efficient outcomes for the multiobjective problem.

Given some stabilization levels q_i^s for stabilized objectives and the requirement that these objectives should be minimized above and maximized below stabilization levels, the set of efficient outcomes can be defined only relative to the stabilization levels. However, since the user can define stabilization levels arbitrarily, of interest here is the union of such relative sets of efficient outcomes. Let $\tilde{D} = D \setminus \{\emptyset\}$ and $\tilde{D}_\epsilon = D_\epsilon \setminus \{\emptyset\}$; then the outcomes that might be efficient or properly efficient with bound for arbitrary stabilization levels for stabilized objectives can be defined, as before, by the relations (4) or (7). The weakly efficient outcomes are of no practical interest in this case, since the cone D typically has empty interior which implies that weakly efficient outcomes coincide with all attainable outcomes.

The stabilized outcomes in the above definition of efficiency are, in a sense, similar to the outcomes with equality constraints (9); however, there is an important distinction between these two concepts. Equality constraints must be satisfied; if not, then there are no admissible solutions for the model. Stabilized objective outcomes should be kept close to stabilization levels, but they can differ from those levels if, through this difference, other objectives can be improved. The user of a decision support system should keep this distinction in mind and can, for example, modify the definition of the multiobjective analysis problem by removing equality constraints for some outcomes and putting these outcomes into the stabilized objective category. Outcomes with inequality constraints could be in the same way converted to either minimized or maximized outcomes.

By adding shifting scales, adding a number of proxy variables and changing the interpretation of the function g, the substantive model formulation (8), (9), (10), (11) together with its positive cone (12) and the related concept of efficiency could be equivalently rewritten to the standard form of multiobjective nonlinear programming (1); this, however, does not concern the user. More important is the way of user-controlled selection of an efficient decision and outcome from the set (4) or (7). For stabilized objective outcomes, the user can change the related stabilization levels in order to influence this selection; *it is assumed here that he will do so for all objective outcomes*, that is, *use the corresponding reference levels in order to influence the selection of efficient decisions.*

For minimized and maximized objectives the user can specify two kinds of reference levels: *aspiration levels* denoted here $\bar{q}_i$ or $\bar{q}$ as a vector called *aspiration point* and *reservation levels* denoted $\bar{\bar{q}}_i$ or $\bar{\bar{q}}$ as a vector called *reservation point*. The aspiration levels represent the levels that the user would like to attain (although the aspiration point as whole is not attainable in most cases), whereas the reservation levels could be interpreted as 'soft' lower limits for objectives (for maximized objectives; upper limits for minimized objectives). Reservation levels $\bar{\bar{q}}_i$ for maximized objectives should be 'below' the aspiration levels $\bar{q}_i$ $(\bar{\bar{q}}_i < \bar{q}_i,$ $i = 1, \ldots, p')$, whereas reservation levels $\bar{\bar{q}}_i$ for minimized objectives should be 'above' the aspiration levels $\bar{q}_i$ $(\bar{\bar{q}}_i > \bar{q}_i,$ $i = p'+1, \ldots, p'')$. If these conditions are not satisfied for some objectives, system automatically changes $\bar{q}_i$ or $\bar{\bar{q}}_i$.

For each stabilized objective q_i the user can specify the *lower reservation level* denoted $\bar{\bar{q}}_i^l$ and the *upper reservation level* denoted $\bar{\bar{q}}_i^u$. It is assumed that the stabilization level q_i^s is given implicitly as the mean value of two reservation levels $q_i^s = (\bar{\bar{q}}_i^l + \bar{\bar{q}}_i^u)/2$, thus the user defines the *reservation range* around the stabilization level. Moreover the system defines internally the *lower aspiration level* $\bar{q}_i^u = q_i^s - \delta(\bar{\bar{q}}_i^u - \bar{\bar{q}}_i^l)/2$ and the *upper aspiration level* $\bar{q}_i^l = q_i^s + \delta(\bar{\bar{q}}_i^u - \bar{\bar{q}}_i^l)/2$, thus the *aspiration range* is δ times narrower than the reservation

range with q_i^s being the center of both ranges. The coefficient δ has the default value 0.1 and can be changed by the user during the interactive process.

The aspiration and reservation points, called jointly *reference points*, are both user-selectable parameters (for minimized and maximized objectives; for stabilized objectives two reservation levels are user-selectable). A special way of parametric scalarization of the multiobjective analysis problem is utilized for the purpose of influencing the selection of efficient outcomes by changing reference points. This parametric scalarization is obtained through maximizing an *order-approximating achievement function* (see Wierzbicki 1983, 1986). There are several forms of such functions; properly efficient outcomes with approximate bound $\varepsilon, 1/\varepsilon$ are obtained when maximizing a function of the following form:

$$\begin{aligned} s(q,\bar{q},\bar{\bar{q}}) \;=\; & \min\left(\min_{1\le i\le p''} z_i(q_i,\bar{q}_i,\bar{\bar{q}}_i),\ \min_{p''+1\le i\le p} z_i(q_i,\bar{\bar{q}}_i^l,\bar{\bar{q}}_i^u)\right) + \\ & + \varepsilon\left(\sum_{i=1}^{p''} z_i(q_i,\bar{q}_i,\bar{\bar{q}}_i) + \sum_{i=p''+1}^{p} z_i(q_i,\bar{\bar{q}}_i^l,\bar{\bar{q}}_i^u)\right), \end{aligned} \tag{13}$$

where the parameter ε should be positive, even if very small; if this parameter would be equal zero, then the above function would not be order-approximating any more, but *order-representing*, and its maximal points could correspond to weakly efficient outcomes.

The functions $z_i(q_i,\bar{q}_i,\bar{\bar{q}}_i)$ for maximized objectives $(i=1,\ldots,p')$ are defined by:

$$z_i(q_i,\bar{q}_i,\bar{\bar{q}}_i) = \min\left((q_i-\bar{\bar{q}}_i)/s_i', 1+(q_i-\bar{q}_i)/s_i''\right) \tag{14}$$

and the functions $z_i(q_i,\bar{q}_i,\bar{\bar{q}}_i)$ for minimized objectives $(i=p'+1,\ldots,p'')$ are defined by:

$$z_i(q_i,\bar{q}_i,\bar{\bar{q}}_i) = \min\left((\bar{\bar{q}}_i-q_i)/s_i', 1+(\bar{q}_i-q_i)/s_i''\right) \tag{15}$$

while the functions $z_i(q_i,\bar{\bar{q}}_i^l,\bar{\bar{q}}_i^u)$ for stabilized objectives $(i=p''+1,\ldots,p)$ are defined by:

$$\begin{aligned} & z_i(q_i,\bar{\bar{q}}_i^l,\bar{\bar{q}}_i^u) = \min\left(z_i^l, z_i^u\right) \\ & z_i^l = \min\left((q_i-\bar{\bar{q}}_i^l)/s_i', 1+(q_i-\bar{q}_i^l)/s_i''\right) \\ & z_i^u = \min\left((\bar{\bar{q}}_i^u-q_i)/s_i', 1+(\bar{q}_i^u-q_i)/s_i''\right) \end{aligned} \tag{16}$$

where

$$\begin{aligned} & q_i^l = q_i^s - \delta(q_i^s-\bar{\bar{q}}_i^l), \\ & q_i^u = q_i^s + \delta(\bar{\bar{q}}_i^u-q_i^s), \\ & q_i^s = (\bar{\bar{q}}_i^u-\bar{\bar{q}}_i^l)/2. \end{aligned} \tag{17}$$

The coefficients $s_i' > 0,\ s_i'' > 0$ in (14), (15) and (16) are scaling units for all objectives and are determined automatically in the IAC-DIDAS-N system to obtain the following common *absolute achievement measure* for all *individual criterion achievement functions* $z_i(q_i,\cdot,\cdot)$:

$$z_i = \begin{cases} 1+\eta & \text{if } q_i = q_i^{\text{best}} \quad (q_i^s \text{ for stabilized objectives}) \\ 1 & \text{if } q_i = \bar{q}_i \quad (\bar{q}_i^l \text{ or } \bar{q}_i^u \text{ for stabilized objectives}) \\ 0 & \text{if } q_i = \bar{\bar{q}}_i \quad (\bar{\bar{q}}_i^l \text{ or } \bar{\bar{q}}_i^u \text{ for stabilized objectives}) \end{cases} \tag{18}$$

where q_i^{best} is the upper limit (for maximized objectives; lower limit for minimized objectives) of all attainable efficient values of objective q_i and $\eta > 0$ is an arbitrary coefficient.

For minimized or maximized objectives ($i = 1, \ldots, p''$), scaling coefficients s_i' and s_i'' depend on relations between aspiration level $\bar{q}_i$, reservation level $\bar{\bar{q}}_i$ and upper limit q_i^{max} (for maximized objectives; lower limit q_i^{min} for minimized objectives) of all attainable efficient values of objective q_i :

$$\begin{aligned} s_i' &= \bar{q}_i - \bar{\bar{q}}_i, \quad & s_i'' &= (q_i^{max} - \bar{q}_i)/\eta, \quad & \text{if} \quad & 1 \le i \le p', \\ s_i' &= \bar{\bar{q}}_i - \bar{q}_i, \quad & s_i'' &= (\bar{q}_i - q_i^{min})/\eta, \quad & \text{if} \quad & p' + 1 \le i \le p''. \end{aligned} \tag{19}$$

For stabilized objectives ($i = p'' + 1, \ldots, p$), scaling coefficients s_i' and s_i'' depend on the distance between $\bar{\bar{q}}_i^l$ and $\bar{\bar{q}}_i^u$ (i.e. reservation range) and on the user defined coefficient δ (i.e. on relations between aspiration and reservation ranges):

$$\left.\begin{aligned} s_i' &= (1 - \delta)(\bar{\bar{q}}_i^u - \bar{\bar{q}}_i^l)/2 \\ s_i'' &= \delta(\bar{\bar{q}}_i^u - \bar{\bar{q}}_i^l)/(2\eta) \end{aligned}\right\} \quad \text{if} \quad p'' + 1 \le i \le p. \tag{20}$$

Parameter η in (18), (19) and (20) is selected according to current relations between $\bar{q}_i$, $\bar{\bar{q}}_i$, q_i^{max}, q_i^{min} and the value of coefficient δ :

$$\eta = \min \left(\min_{1 \le i \le p'} \frac{q_i^{max} - \bar{q}_i}{\bar{q}_i - \bar{\bar{q}}_i} \, , \min_{p'+1 \le i \le p''} \frac{\bar{q}_i - q_i^{min}}{\bar{\bar{q}}_i - \bar{q}_i} \, , \frac{\delta}{1 - \delta} \right) \tag{21}$$

The system checks and does necessary projections for three sets of conditions that must hold for this selection of s_i' and s_i'' :

$$\begin{aligned} \bar{\bar{q}}_i < \bar{q}_i < q_i^{max} \quad & , \text{if} \quad 1 \le i \le p', \\ q_i^{min} < \bar{q}_i < \bar{\bar{q}}_i \quad & , \text{if} \quad p' + 1 \le i \le p'', \\ \bar{\bar{q}}_i^l < \bar{\bar{q}}_i^u \quad & , \text{if} \quad p'' + 1 \le i \le p. \end{aligned} \tag{22}$$

The achievement function $s(q, \bar{q}, \bar{\bar{q}})$ can be maximized with $q = f(x)$ over $x \in X$; however, the function (13) is nondifferentiable (for example, if $q = \bar{q}$). On the other hand, if the function $g(x)$ (and thus also $f(x)$) is differentiable, then the maximization of function (13) in the system can be converted automatically to an equivalent differentiable nonlinear programming problem by introducing proxy variables and substituting the min operation in (13) by a number of additional inequalities. If the coefficient $\varepsilon > 0$, then the achievement function has the following properties (see Wierzbicki, 1986):

a) For any arbitrary aspiration and reservation points satisfying conditions (22), not necessarily restricted to be attainable ($\bar{q} \in Q$, $\bar{\bar{q}} \in Q$) or not attainable ($\bar{q} \notin Q$, $\bar{\bar{q}} \notin Q$), each maximal point $\hat{q}$ of the achievement function $s(q, \bar{q}, \bar{\bar{q}})$ with $q = f(x)$ over $x \in X$ is a D_ε-efficient solution, that is, a properly efficient solution with trade-off coefficients bounded approximately by ε and $1/\varepsilon$.

b) For any properly efficient outcome $\hat{q}$ with trade-off coefficients bounded by ε and $1/\varepsilon$, there exist such aspiration $\bar{q}$ and reservation $\bar{\bar{q}}$ points that the maximum of the achievement function $s(q, \bar{q}, \bar{\bar{q}})$ is attained at the properly efficient outcome $\hat{q}$. In particular, if

the user (either by chance or as a result of a learning process) specifies some attainable but not efficient reservation point $\bar{\bar{q}}$ and an aspiration point $\bar{q}$ that in itself is such properly efficient outcome, $\bar{q} = \hat{q}$, and if conditions (22) are satisfied then the maximum of the achievement function $s(q, \bar{q}, \bar{\bar{q}})$, equal one, is attained precisely at this point.

c) If the aspiration point $\bar{q}$ is 'too high' (for maximized outcomes; 'too low' for minimized outcomes), then the maximum of the achievement function, smaller than one, is attained at an efficient outcome $\hat{q}$ that best approximates uniformly, in the sense of scaling units s_i', the aspiration point. If the aspiration point $\bar{q}$ is 'too low' (for maximized outcomes; 'too high' for minimized outcomes), then the maximum of the achievement function, larger than one, is attained at an efficient outcome $\hat{q}$ that is uniformly, in the sense of scaling units s_i'', 'higher' than the aspiration point.

d) By changing his aspiration $\bar{q}$ and reservation $\bar{\bar{q}}$ points, the user can continuously influence the selection of the corresponding efficient outcomes $\hat{q}$ that maximize the achievement function, provided the maximum is unique and the set $\hat{Q}^{p\varepsilon}$ is connected.

The parameter ε in the achievement function determines bounds on trade-off coefficients: if an efficient solution has trade-off coefficients that are too large or too small (say, lower than 10^{-6} or higher than 10^{6}) than it does not differ for the decision maker from weakly efficient outcomes — some of its components could be improved without practically deteriorating other components. Another interpretation of this parameter is that it indicates how much an average overachievement (or underachievement) of aspiration levels should correct the minimal overachievement (or maximal underachievement) in the function (13).

The achievement function (13) can be transformed to an equivalent form if taking into account the scaling coefficients determined by (19) and (20) and assuming that the parameter $\varepsilon = 0$:

$$s(q, \bar{q}, \bar{\bar{q}}) = 1 + \eta - \max\left(\max_{1 \le i \le p''} \tilde{z}_i(q_i, \bar{q}_i, \bar{\bar{q}}_i), \max_{p''+1 \le i \le p} \tilde{z}_i(q_i, \bar{\bar{q}}_i^l, \bar{\bar{q}}_i^u) \right) \tag{23}$$

with

$$\begin{aligned} \tilde{z}_i(q_i, \bar{q}_i, \bar{\bar{q}}_i) &= \max\left(w_i', w_i''\right), \qquad 1 \le i \le p'', \\ \tilde{z}_i(q_i, \bar{\bar{q}}_i^l, \bar{\bar{q}}_i^u) &= \max\left(w_i^{-\prime}, w_i^{-\prime\prime}, w_i^{+\prime\prime}, w_i^{+\prime}\right), \qquad p'' + 1 \le i \le p, \end{aligned} \tag{24}$$

where

$$w_i' = \begin{cases} \eta + \dfrac{\bar{q}_i - q_i}{\bar{q}_i - \bar{\bar{q}}_i}, & \text{if} \quad 1 \le i \le p', \\ \eta + \dfrac{q_i - \bar{q}_i}{\bar{\bar{q}}_i - \bar{q}_i}, & \text{if} \quad p' + 1 \le i \le p'', \end{cases} \tag{25}$$

$$w_i'' = \begin{cases} \eta \dfrac{q_i^{\max} - q_i}{q_i^{\max} - \bar{q}_i}, & \text{if} \quad 1 \le i \le p', \\ \eta \dfrac{q_i - q_i^{\min}}{\bar{q}_i - q_i^{\min}}, & \text{if} \quad p' + 1 \le i \le p'', \end{cases} \tag{26}$$

$$\left.\begin{aligned} w_i^{-\prime} &= \eta + \frac{\bar{q}_i^l - q_i}{\bar{q}_i^l - \bar{\bar{q}}_i^l} \\ w_i^{-\prime\prime} &= 2\eta\,\frac{q_i^s - q_i}{\bar{q}_i^u - \bar{q}_i^l} \\ w_i^{+\prime\prime} &= 2\eta\,\frac{q_i - q_i^s}{\bar{q}_i^u - \bar{q}_i^l} \\ w_i^{+\prime} &= \eta + \frac{q_i - \bar{q}_i^u}{\bar{\bar{q}}_i^u - \bar{q}_i^u} \end{aligned}\right\} \quad \text{if } p'' + 1 \le i \le p, \tag{27}$$

with q_i^s, $\bar{q}_i^l$ and $\bar{q}_i^u$ given by (17).

The maximization of an achievement function in IAC-DIDAS-N is performed by a specially developed nonlinear optimization algorithm, called *solver*. Since this maximization is performed repetitively, at least once for each interaction with the user that changes the parameters $\bar{q}$ or $\bar{\bar{q}}$, there are special requirements for the solver that distinguish this algorithm from typical nonlinear optimization algorithms: it should be robust, adaptable and efficient, that is, it should compute reasonably fast an optimal solution for optimization problems of a broad class (for various differentiable functions $g(x)$ and $f(x)$) without requiring from the user that he adjusts special parameters of the algorithm in order to obtain a solution. The experience in applying nonlinear optimization algorithms in decision support systems (see Kreglewski and Lewandowski, 1983, Kaden and Kreglewski, 1986) has led to the choice of an algorithm based on penalty shifting technique and projected conjugate gradient method. Since a penalty shifting technique anyway approximates nonlinear constraints by penalty terms, an appropriate form of an achievement function that differentiably approximates function (23) has been also developed and is actually used in IAC-DIDAS-N. This *smooth order-approximating achievement function* has the form:

$$\begin{aligned} s(q, \bar{q}, \bar{\bar{q}}) = 1 + \eta - \Bigg\{ & \sum_{i=1}^{p''} \left[(\max(0, w_i'))^\alpha + (w_i'')^\alpha \right] + \\ & + \sum_{i=p''+1}^{p} \left[\left(\max(0, w_i^{-\prime})\right)^\alpha + \left(\max(w_i^{-\prime\prime}, w_i^{+\prime\prime})\right)^\alpha + \left(\max(0, w_i^{+\prime})\right)^\alpha \right] \Bigg\}^{1/\alpha} \end{aligned} \tag{28}$$

where w_i', w_i'', $w_i^{-\prime}$, $w_i^{-\prime\prime}$, $w_i^{+\prime\prime}$ and $w_i^{+\prime}$ are given by (25), (26) and (27).

The parameter $\alpha \ge 2$ is responsible for the approximation of the function (13) or (23) by the function (28): if $\alpha \to \infty$ and $\varepsilon \to 0$, then these functions converge to each other (if taking into account the specific definition of scaling coefficients in (13)). However, the use of too large parameters α results in badly conditioned problems when maximizing function (28), hence $\alpha = 4 \div 10$ are suggested to be used, the default value is $\alpha = 10$. During numerical computations a slightly simpler *scalarizing function* is used and minimized:

$$\begin{aligned} \tilde{s}(q, \bar{q}, \bar{\bar{q}}) = & \sum_{i=1}^{p''} \left[(\max(0, w_i'))^\alpha + (w_i'')^\alpha \right] + \\ & + \sum_{i=p''+1}^{p} \left[\left(\max(0, w_i^{-\prime})\right)^\alpha + \left(\max(w_i^{-\prime\prime}, w_i^{+\prime\prime})\right)^\alpha + \left(\max(0, w_i^{+\prime})\right)^\alpha \right] \end{aligned} \tag{29}$$

The function (29) must be minimized with $q = f(x)$ over $x \in X$, while X is determined by simple bounds $x^{lo} \leq x \leq x^{up}$ as well as by inequality constraints $y^{lo} \leq g(x) \leq y^{up}$ (or equality constraints for some i such that $y_i^{lo} = y_i^{up}$). In the shifted penalty technique, the following function is minimized instead:

$$p(x, \xi', \xi'', u', u'') = \tilde{s}(f(x), \bar{q}, \bar{\bar{q}}) + \\ + \frac{1}{2} \sum_{i=1}^{p} \xi_i' \left(\max(0, g_i(x) - y_i^{up} + u_i') \right)^2 + \\ + \frac{1}{2} \sum_{i=1}^{p} \xi_i'' \left(\max(0, y_i^{lo} - g_i(x) + u_i'') \right)^2 \tag{30}$$

where ξ', ξ'' are penalty coefficients and u', u'' are penalty shifts. This function is minimized over x such that $x^{lo} \leq x \leq x^{up}$ while applying conjugate gradient directions, projected on these simple bounds if one of the bounds becomes active. When a minimum of this penalty function with given penalty coefficients and given penalty shifts (the latter are initially equal zero) is found, the violations of all outcome constraints are computed, the penalty shifts and coefficients are modified according to the shifted-increased penalty technique (see, e.g., Wierzbicki, 1984), and the penalty function is minimized again until the violations of outcome constraints are admissibly small. The results are then equivalent to the outcomes obtained by minimizing the scalarizing function (29) under all constraints. This technique, though it might seem cumbersome, is according to our experience one of the most robust nonlinear optimization methods; the user of the system is not bothered with its details, since the adjustment of penalty shifts and coefficients is done automatically in this technique.

Another advantage for the user is that he is not bothered with the definition of derivatives of penalty function (30), needed in the conjugate gradient method, nor even with the definition of the derivatives of constraints functions $g_i(x)$ and outcome functions $f(x)$. This is unique feature of IAC-DIDAS-N system: all needed derivatives are automatically (symbolically) determined and computed either in the nonlinear model generator that supports the model definition phase or in the solver algorithm using shifted penalty technique.

The only parameter that might influence the interaction of the system with the user is the parameter α in the smooth scalarizing function (29). Thus, the user can select this parameter; if this parameter is very large, his control of efficient outcomes obtained by minimizing (29) is somewhat easier, but the solver might take long time or produce not quite robust results in this case. The user has also access to some other parameters of the optimization procedure, which is needed in cases of especially difficult optimization problems.

The minimization of an scalarizing function is a convenient way of organizing the interaction between the model and the user. Before the interactive analysis phase, however, the user must firstly define the substantive model, then define the multiobjective analysis problem by specifying outcome variables that should be maximized, minimized, stabilized, or ***floating*** (that is, displayed for users' information only, but not included as optimized objectives; various decision variables of interest to the user can be also included into one of these categories).

The scalarizing function of the form (29) uses two kinds of additional information:

- bounds of efficient outcomes: 'upper' bounds for maximized outcomes, 'lower' bounds for minimized outcomes. These bounds must be determined once for the given multiobjective analysis problem.

- user-supplied reference levels: aspiration level and reservation level for each minimized or maximized outcome, two reservation levels for each stabilized outcome. The user changes reference levels (aspiration, reservation or both) several times during the interactive analysis of the multiobjective problem, however some initial values should be determined in the system.

In the initial analysis phase of the work with the IAC-DIDAS-N system the bounds for efficient outcomes are calculated: the 'upper' (in the meaning of the 'best' attainable) and the 'lower' (in the meaning of the 'worst' attainable and efficient). The former is determined exactly (with given numerical accuracy), whereas the latter is only approximated, because there is no constructive way for determining it exactly for nonlinear multicriteria problems.

The 'upper' bound for efficient solutions is obtained through maximizing each objective separately (or minimizing, in case of minimized objectives; in the case of stabilized objectives, the user should know their entire attainable range, hence they should be both maximized and minimized), while all others objectives (including stabilized ones) should be considered as floating or free. The scalarizing function (29) is not used during these calculations, objective functions $q_i = f_i(x)$ are used in the penalty function instead of $\tilde{s}$ (directly if the objective under consideration should be minimized or with the minus sign if it should be maximized). If there are no stabilized outcomes, the results of such optimizations form a point that limits from 'above' (for maximized outcomes; from 'below' for minimized outcomes) the set of efficient outcomes $\widehat{Q}$, but this point almost never (except in degenerate cases) is in itself an attainable outcome; therefore, it is called the *utopia point*. The entire number of optimization runs in utopia point computations is $p'' + 2(p - p'')$.

During all these computations, the 'lower' bound for efficient outcomes can be also estimated, just by recording the lowest (for maximized objectives; highest for minimized objectives) efficient outcomes that occur in subsequent optimizations (there is no need to record them for stabilized objectives, where the entire attainable range is anyway estimated). However, such a procedure results in the accurate, tight 'lower' bound for efficient outcomes — called *nadir point* $\hat{q}^{\mathrm{nad}}$ — only if $p'' = 2$; for larger numbers of maximized and minimized objectives, particularly for nonlinear models, this procedure can give misleading results. In further computations appropriate components of $\hat{q}^{\mathrm{uto}}$ and $\hat{q}^{\mathrm{nad}}$ are used as components of q^{max} and q^{min} in the scalarizing function (29).

In very rare and rather degenerate cases some components $\hat{q}_i^{\mathrm{nad}}$ of the estimation of the nadir point and components $\hat{q}_i^{\mathrm{uto}}$ of the utopia point could have the same value — it may happen if, for example, the structure of the substantive model results in the set (2) with empty interior. In such a case the user can update manually these nadir point components according to his knowledge, otherwise the IAC-DIDAS-N system assumes such outcomes to be floating (it is not included in the scalarizing function (29) regardless of its type — maximized, minimized or stabilized) but checks its values at each efficient solution whether it is equal to the value $\hat{q}_i^{\mathrm{nad}} = \hat{q}_i^{\mathrm{uto}}$.

Once the approximate bounds $\hat{q}^{\mathrm{uto}}$ and $\hat{q}^{\mathrm{nad}}$ are computed and known to the user, they can be utilized in various ways. First, their appropriate components are used as components of q^{max} and q^{min} in the scalarizing function (29). Second way consists in computing a *neutral efficient solution*, with objectives situated approximately 'in the middle' of the efficient set. For this purpose, the aspiration point $\bar{q}$ is set very close to the utopia point $\hat{q}^{\mathrm{uto}}$ (only for maximized or minimized outcomes; for stabilized outcomes upper and lower limits of efficient outcomes are used as appropriate reservation levels $\bar{\bar{q}}_i^{\mathrm{l}} = \hat{q}_i^{\mathrm{min}}$ and $\bar{\bar{q}}_i^{\mathrm{u}} = \hat{q}_i^{\mathrm{max}}$) and the reservation point $\bar{\bar{q}}$ is set very close to the nadir point $\hat{q}^{\mathrm{nad}}$ (only for maximized and minimized objectives). By minimizing the scalarizing function $\tilde{s}(q, \bar{q}, \bar{\bar{q}})$ with such data, the

neutral efficient solution is obtained and can be utilized by the user as a starting point for further interactive analysis of efficient solutions. Basing on the neutral efficient solution $\hat{q}^{\text{neu}}$ and bounds on efficient objectives $\hat{q}^{\text{uto}}$ and $\hat{q}^{\text{nad}}$, system proposes to the user initial values for aspiration levels $\bar{q}_i$ and reservation levels $\bar{\bar{q}}_i$ for maximized and minimized objectives:

$$\left.\begin{aligned} \bar{q}_i &= \hat{q}_i^{\text{neu}} - (\hat{q}_i^{\text{neu}} - \hat{q}_i^{\text{uto}})/3 \\ \bar{\bar{q}}_i &= \hat{q}_i^{\text{neu}} + (\hat{q}_i^{\text{neu}} - \hat{q}_i^{\text{uto}})/3 \end{aligned}\right\} \quad \text{if} \quad 1 \le i \le p'' \tag{31}$$

and initial values for lower $\bar{\bar{q}}_i^l$ and upper $\bar{\bar{q}}_i^u$ reservation levels for stabilized outcomes:

$$\left.\begin{aligned} \bar{\bar{q}}_i^l &= \hat{q}_i^{\text{neu}} - \Delta_i \\ \bar{\bar{q}}_i^u &= \hat{q}_i^{\text{neu}} + \Delta_i \end{aligned}\right\} \quad \text{if} \quad p'' + 1 \le i \le p \tag{32}$$

where

$$\Delta_i = 0.5 \min \left(\hat{q}_i^{\text{neu}} - q_i^{\min}, q_i^{\max} - \hat{q}_i^{\text{neu}} \right).$$

These values, although rather arbitrary, constitute a good starting point for further interaction.

In further interactive analysis, an important consideration is that the user should be able to easily influence the selection of the efficient outcomes $\hat{q}$ by changing the aspiration point $\bar{q}$ (and, optionally, the reservation point $\bar{\bar{q}}$) for maximized and minimized objectives and reservation levels $\hat{q}^l$ and $\hat{q}^u$ for stabilized objectives in the minimized scalarizing function $\tilde{s}(q, \bar{q}, \bar{\bar{q}})$. It can be shown (see Wierzbicki, 1986) that the best suited for this purpose is the choice of scaling units s_i' and s_i'' that are not constant, but are changed implicitly by the user and depend on differences between current values of aspiration and reservation levels and utopia point components according to (19) and (20) or, equivalently, using the scalarizing function (29) with (25), (26) and (27) provided that conditions (22) hold. The interpretation of such a way of setting scaling units is that the user attaches implicitly more importance to reaching an aspiration component $\bar{q}_i$ if he places it close to the known utopia component; in such a case, the corresponding scaling unit becomes smaller and the corresponding objective component is weighted stronger in the scalarizing function $\tilde{s}(q, \bar{q}, \bar{\bar{q}})$. Thus, this way of *scaling relative to utopia-reference difference* is taking into account the implicit information given by the user in the relative position of the aspiration point. The only drawback of the described choice of scaling units are strong inequalities in conditions (22), not convenient for the user and for the numerical application, hence $q_i^{\max}$ and $q_i^{\min}$ in (25) and (26) are not taken directly as appropriate components of $\hat{q}^{\text{uto}}$ and $\hat{q}^{\text{nad}}$, but slightly displaced utopia and nadir points are used instead in the current system implementation:

$$\begin{aligned} q_i^{\max} &= \hat{q}_i^{\text{uto}} + 0.01(\hat{q}_i^{\text{uto}} - \hat{q}_i^{\text{nad}}), \qquad \text{if} \qquad 1 \le i \le p', \\ q_i^{\min} &= \hat{q}_i^{\text{uto}} - 0.01(\hat{q}_i^{\text{nad}} - \hat{q}_i^{\text{uto}}), \qquad \text{if} \quad p' + 1 \le i \le p''. \end{aligned} \tag{33}$$

It is assumed now that the user selects the aspiration and reservation components satisfying $\hat{q}_i^{\text{nad}} \le \bar{\bar{q}}_i < \bar{q}_i \le \hat{q}_i^{\text{uto}}$ for maximized outcomes or $\hat{q}_i^{\text{uto}} \le \bar{q}_i < \bar{\bar{q}}_i \le \hat{q}_i^{\text{nad}}$ for minimized outcomes or $\bar{\bar{q}}_i^l < \bar{\bar{q}}_i^u$ for stabilized outcomes (if he does not, the system automatically does necessary projections). If the user specifies only one reference value for some objective, then

the system determines internally the second value, thus the same two reference levels scalarizing function can be used. For maximized and minimized objectives missing reservation levels are calculated using formulae:

$$\bar{\bar{q}}_i = \begin{cases} \bar{q}_i - (q_i^{\max} - \bar{q}_i), & \text{if} \quad 1 \leq i \leq p', \\ \bar{q}_i + (\bar{q}_i - q_i^{\min}), & \text{if} \quad p'+1 \leq i \leq p''. \end{cases} \tag{34}$$

whereas missing aspiration levels are calculated using formulae:

$$\bar{q}_i = \begin{cases} 0.05(q_i^{\max} + \bar{\bar{q}}_i), & \text{if} \quad 1 \leq i \leq p', \\ 0.05(\bar{\bar{q}}_i + q_i^{\min}), & \text{if} \quad p'+1 \leq i \leq p''. \end{cases} \tag{35}$$

When the relative scaling is applied, the user can easily obtain — by suitably moving reference points — efficient outcomes that are either situated close to the neutral solution, in the middle of efficient outcome set $\hat{Q}$, or in some remote parts of the set $\hat{Q}$, say, close to various extreme solutions. Typically, several experiments of computing such efficient outcomes give enough information for the user to select an actual decision — either some efficient decision suggested by the system, or even a different one, since even the best substantive model cannot encompass all aspects of a decision situation. However, there might be some cases in which the user would like to receive further support — either in analyzing the sensitivity of a selected efficient outcome, or in converging to some best preferred solution and outcome.

For analyzing the sensitivity of an efficient solution to changes in the proportions of outcomes, a *multidimensional scan* of efficient outcomes can be applied in IAC-DIDAS-N. This operation consists in selecting an efficient outcome, accepting it as a base $\bar{q}^{\text{bas}}$ for aspiration points, and performing p (or p'') additional optimization runs with the aspiration points determined by:

$$\bar{q}_j = \bar{q}_j^{\text{bas}} + \beta \left(\hat{q}_j^{\text{uto}} - \hat{q}_j^{\text{nad}} \right), \quad \bar{q}_i = \bar{q}_i^{\text{bas}}, \quad i \neq j, \quad 1 \leq i \leq p \tag{36}$$

where β is a coefficient determined by the user, $-1 \leq \beta \leq 1$; if the aspiration components determined by (36) are outside the range $\hat{q}_j^{\text{nad}}$, $\hat{q}_j^{\text{uto}}$, they are projected automatically on this range; the reservation point is kept constant ($\bar{\bar{q}} = \hat{q}^{\text{nad}}$) during this procedure. The aspiration components for stabilized outcomes may or may not be perturbed in this operation. The efficient outcomes resulting from the minimization of the scalarizing function $\tilde{s}(q, \bar{q}, \bar{\bar{q}})$ with such perturbed aspiration points are typically also perturbed mostly along their respective components, although other their components might also change.

For analyzing the sensitivity of an efficient solution when moving along a direction in the outcome space — and also as a help in converging to a most preferred solution — a *directional scan* of efficient outcomes can be implemented in IAC-DIDAS-N. This operation consists again in selecting an efficient outcome, accepting it as a base $\bar{q}^{\text{bas}}$ for aspiration points, selecting another aspiration point $\bar{q}$, and performing a user-specified number K of additional optimizations with aspiration points determined by:

$$\bar{q}(k) = \bar{q}^{\text{bas}} + \frac{k}{K} \left(\bar{q} - \bar{q}^{\text{bas}} \right), \quad 1 \leq k \leq K \tag{37}$$

The efficient solutions $\hat{q}(k)$ obtained through minimizing the scalarizing function $\tilde{s}(q, \bar{q}(k), \bar{\bar{q}})$ with such aspiration points (and constant reservation point $\bar{\bar{q}} = \hat{q}^{\text{nad}}$) constitute a cut through the efficient set $\tilde{Q}$ when moving approximately in the direction $\bar{q} - \bar{q}^{\text{bas}}$.

If the user selects one of these efficient solutions, accepts as a new $\bar{q}^{bas}$ and performs next directional scans along some new directions of improvement, he can converge eventually to his most preferred solution (see Korhonen, 1985). Even if he does not wish the help in such convergence, directional scans can give him valuable information.

Another possible way of helping in convergence to the most preferred solution is choosing aspiration points as in (37) but using a harmonically decreasing sequence of coefficients (such as $1/j$, where j is the iteration number) instead of user-selected coefficients k/K. This results in convergence even if the user makes stochastic errors in determining next directions of improvement of aspiration points, or even if he is not sure about his preferences and learns about them during this analysis (see Michalevich, 1986). Such a convergence, called here forced convergence, is rather slow. Neither the forced convergence nor multidimensional nor directional scan are yet implemented in the current version of IAC-DIDAS-N, but they will be implemented in later versions.

3 Short user manual

3.1 Introduction

The IAC-DIDAS-N system (Institute of Automatic Control, Dynamic Interactive Decision Analysis and Support, Nonlinear version) is decision support system designed to help in the analysis of decision situations where a mathematical model of substantive aspects of the situation can be formulated in the form of a multiobjective nonlinear programming problem.

The system can be run on an IBM-PC-XT, AT or a compatible computer with Hercules Graphics Card, Color Graphic Adapter or Enhanced Graphics Adapter and, preferably, with a numeric coprocessor and a hard disk. If a numeric coprocessor is available then the coprocessor version called `DIDASN` of the IAC-DIDAS-N system can be used taking advantage of the coprocessor computational capacity, otherwise only the emulation version called `DIDASNE` of the IAC-DIDAS-N system can only be used with less computational capabilities. The system is recorded on two diskettes. Each diskette contains the compiled code of one version of the program together with some data files with demonstrative examples of nonlinear models. While the installation of the selected version of the system in the user directory on a hard disk (or less preferably on a working diskette) is done (using `INSTALL` batch file contained on both diskettes — see the installation guide in the Appendix A), the program can be activated by the command `DIDASN` or `DIDASNE` at the DOS prompt.

System supports the following general functions:

- definition and edition of a substantive model of the decision situation in a user-friendly format of a spreadsheet and a screen window editor.
- specification of a multiobjective decision analysis problem related to the substantive model. This is performed by specific features of spreadsheet edition.
- initial multiobjective analysis of the problem, resulting in estimating bounds on efficient outcomes of decisions and in learning about some extreme and some neutral decisions. These functions are supported by some specific commands and the results are presented to the user in the spreadsheet and graphical form.
- interactive analysis of the problem with the stress on learning by the user of possible efficient decisions and outcomes, organized through system's response to user-specified aspiration and reservation levels for objective outcomes. The IAC-DIDAS-N system

responds with efficient solutions and objective outcomes obtained through the maximization of an achievement function that is parameterized by the user-specified aspiration and reservation points. The maximization is performed through a nonlinear programming algorithm called solver. The interactive analysis is supported by entering user data into specific cells in the spreadsheet, executing commands from the menu and using graphical representation of results.

The menus of IAC-DIDAS-N are organized as pull-down tree-structured menus and performs various functions used in several phases of the interactive analysis process. Most of the functions of model edition phase as well as specification of a decision problem and it's initial analysis phase are specific commands in the spreadsheet edition (the decision variables are defined as columns of the spreadsheet, the outcome variables are defined as rows, outcome formulae are entered in the corresponding cells, there are special rows and columns for lower and upper bounds, for defining user names of objective outcomes and their types, reference points, utopia point, for solutions corresponding to the reference points). The functions of the interactive analysis phase are executed by macrocommands using menus and various function keys; the user can get various help displays that suggest in an easy fashion the commands useful in a current phase of work with the system.

IAC-DIDAS-N system has been developed in the Institute of Automatic Control, Warsaw University of Technology, Warsaw, Poland which has the authorship rights, under the contracted study agreement "Theory, Software and Testing Examples for Decision Support Systems" with the Systems and Decision Sciences Program of the International Institute for Applied Systems Analysis, Laxenburg near Vienna, Austria, which has the copyright for this system in international distribution. Please contact Methodology of Decision Analysis Project of SDS Program at IIASA, A–2361 Laxenburg, Austria.

3.2 Phases of the work

The work with a IAC-DIDAS-N system consists of three phases:

1. model edition phase
2. problem definition and initial analysis phase
3. interactive analysis and comparison of results phase

All these phases are supported in the system and an explicit command is required to move from one phase to another. Moreover system checks, whether required move is possible and gives appropriate error messages or asks for additional confirmation. There are two logical spreadsheets: the model editing spreadsheet and the interactive analysis spreadsheet. The former is used mostly to perform all the system functions in the first phase of the work, whereas the latter performs all the system functions during two other phases.

System invoked without arguments always starts with phase 1 and permits of the move to phase 2 only if the model definition is complete. A complete model consists of three groups of obligatory data:

- valid formulae for all defined outcomes (rows of the spreadsheet),
- lower and upper bounds (that do not contradict) for all variables,
- values for all used parameters.

Optionally, model can contain some other, user-supplied data:

- lower and upper bounds (that do not contradict) for outcomes,
- names of variables, parameters and outcomes (user names override standard system names of the form x1, x2, ...for variables, z1, z2, ...for parameters and y1, y2, ...for outcomes). The names must be unique within the model.
- units for variables, parameters and outcomes. This information can be included to improve the understanding of the model in the spreadsheet, but it is not used by the system. The only exception is the use of units for outcomes for special scaling method in graphical representation.
- lower and upper bounds for parameters; not used in the current system implementation but planned to be used in parametric analysis in future system implementations.
- short (up to 30 characters) model description; it is displayed in the spreadsheet and printed together with the print-out of the model. It may be used as an extension to the model name, that is too short (up to 8 characters) to be meaningful.
- five parameters that are used to tune the nonlinear solver (see the next section). If some of them are not given, then current default system values are stored together with other model data.

To store the edited model on a disk, the name of the model must be supplied by the user. Therefore, while using < F2 > (Save) or < Alt M F > (Model selection — Fix and save) commands (see section 3.6), system asks for the name (up to 8 characters, it must be a valid DOS file name). However, there is an important distinction between these two commands. The former just saves the model as it is, whereas the latter first checks for completeness of the model and either displays an error message if the model is not complete or changes the status of the model to 'fixed' and stores the complete model. Both commands check, whether the given name is not the same as the model file name existing already in the current directory on a disk. If it is, then system asks for additional confirmation. Answering 'yes' means, that the system deletes the file with the same name containing the previous model definition together with all related problem definition files and result data files.

Successful 'Fix and save' command for the model moves automatically to the second phase of the work. Obligatory elements of the model definition and bounds for outcomes (if given) can not be changed now. The command < Alt M N > (Model selection — New) must be used to move back to the first phase to allow any change in this part of the model definition, but it will be treated as a definition of a new model. The command < Alt M R > (Model selection — Reset) can also be used to move back to the first phase, but it deletes all the model data and starts a new model definition from the scratch. Optional parts of the model definition (except the bounds for outcomes) can be changed even if the model is fixed, because they do not affect computational characteristics of the model. Such changes in the model definition can be stored on a disk using the command < F2 > (Save). It also holds for all changes in the rows and columns order done with the < Alt F R M > (Format — Rows — Move) and < Alt F C M > (Format — Columns — Move) commands.

Model stored on a disk can be restored using the command < Alt M G > (Model selection — Get from disk). The model is restored together with the information, whether it was fixed or not. Thus, after restoring not fixed model the system is still in the phase 1, whereas after restoring a fixed model the system moves automatically to the phase 2. It is also possible

to restore a model immediately while invoking the IAC-DIDAS-N system — the name of the model must be put as the first argument in the DOS command line (for example to restore automatically the model DEMO, invoke the system with the command DIDASN DEMO or DIDASNE DEMO).

Edited model (not necessarily fixed) can be printed using the command < Alt M P > (Model selection — Print). Model stored on a disk, but actually not interesting to the user can be deleted from a disk together with all related problem definition files and result data files using the command < Alt M E > (Model selection — Erase).

Second phase of the work with the IAC-DIDAS-N system is for a specification of the decision problem to be analyzed using already defined and 'fixed' model. The complete definition of a decision problem consists of two parts: user supplied part and system supplied part. The user supplied part must contain two kinds of information: the selection of outcomes to be minimized, maximized or stabilized objectives (the contents of the 'Stat' column in the spreadsheet) and the values of lower and upper bounds for outcomes, if not given in the model definition. Moreover, the user can add some other data:

- lower and upper bounds for outcomes, even if already given in the model definition; the new values override correspondent data in the model definition,
- short (up to 30 characters) problem description; it is displayed in the spreadsheet and printed together with the print-out of the problem. It may be used as an extension to the problem name, that is too short (up to 8 characters) to be meaningful.
- five parameters that are used to tune the nonlinear solver (see the next section); these values override correspondent data either in the model definition or default system values.
- new bounds (updates) of the system supplied approximation of the nadir point.

For a given set of selected objectives (and optional bounds redefinitions and some solver parameters) system performs initial analysis of the decision problem: calculation of ranges of efficient solutions (utopia and nadir points) and calculation of neutral solution being the starting point for further interaction during the third phase of the work (see the theoretical manual). Two parts of this analysis can be performed jointly using the command < F3 > (Calculate) or separately using first the command < Alt P U > (Problem selection — Utopia) and next the command < Alt P T > (Problem selection — neuTral). The system supplied part of a complete problem consists of results of these calculations. If the calculations are performed separately then after calculating the utopia point (but prior to the neutral point calculation) the user can modify values of the nadir point approximation using the command < Alt P A @> (Problem selection — nAdir).

To store the edited problem on a disk, the name of the problem must be supplied by the user. Therefore, while using < F2 > (Save) or < Alt P F > (Problem selection — Fix and save) commands, system asks for the name (up to 8 characters, it must be a valid DOS file name). However, there is again an important distinction between these two commands. The former just saves the problem as it is, whereas the latter first checks for completeness of the problem and either displays an error message if the problem is not complete or changes the status of the problem to 'fixed' and stores the complete problem. The file containing the model definition is automatically updated if any changes were made in its optional part. Both commands check, whether the given name is not the same as the problem file name existing already on a disk for the currently used model. If it is, then system asks for additional

confirmation. Answering 'yes' means, that the system deletes the file with the same name containing previous problem definition together with all related result data files.

Successful 'Fix and save' command for the problem moves automatically to the third phase of the work. None of the problem elements can be changed now. The command < Alt P N > (Problem selection — New) must be used to move back to the second phase to allow any change in the problem definition, but it will be treated as a definition of a new problem. The command < Alt P R > (Problem selection — Reset) can also be used to move back to the second phase, but it deletes all the problem data and starts new problem definition from the scratch.

Problem stored on a disk can be restored using < Alt P G > (Problem selection — Get from disk). The problem is restored together with the information, whether it was fixed or not. Thus, after restoring not fixed problem the system is still in the phase 2, whereas after restoring a fixed problem the system moves automatically to the phase 3. It is also possible to restore a problem immediately while invoking the IAC-DIDAS-N system — the name of the model must be put as the first argument in the DOS command line and the name of the problem must be put as the second argument in the DOS command line (for example to restore automatically problem `FIRST` defined for the model `DEMO`, invoke the system with the command `DIDASN DEMO FIRST` or `DIDASNE DEMO FIRST`).

The result of neutral solution calculation is stored in a separate file as a result file with a system-defined name 'Neutral'. It is stored on a disk and then restored each time a fixed problem related to it is stored and restored.

Edited problem (not necessarily fixed) can be printed using the command < Alt P P > (Problem selection — Print). Moreover, if the problem is fixed then the neutral result can also be printed using the command < Alt R P > (Result selection — Print). Problem stored on a disk, but actually not interesting to the user can be deleted from a disk together with all related result data files using the command < Alt P E > (Problem selection — Erase).

Third phase of the work with the IAC-DIDAS-N system is for an interactive analysis of the decision problem already defined and 'fixed', for defined and 'fixed' model. The only values that are changed by the user during this phase are aspirations and reservations for minimized or maximized objectives and lower and upper reservations for stabilized objectives (see the theoretical manual). The complete result consists of these user-supplied data together with the system response — an efficient solution calculated using scalarizing function parameterized with these data. Either < F3 > (Calculate) or < Alt R C > (Result selection — Calculate) can be used to start the calculations.

Moreover, the user can add short (up to 30 characters) result description; it is displayed in the spreadsheet and printed together with the print-out of the result. It may be used as an extension to the result name, that is too short (up to 8 characters) to be meaningful.

To store the calculated result on a disk, the name of the result must be supplied by the user. Therefore, while using < F2 > (Save) or < Alt R S > (Result selection — Save and new) commands, system asks for the name (up to 8 characters, it must be a valid DOS file name). There is no difference between these two commands. Both commands first checks for completeness of the result and either displays a message if the result is not calculated or stores the calculated result. The file containing the model definition is automatically updated if any changes were made in its optional part. Both commands check, whether the given name is not the same as the result file name existing already on a disk for the currently used problem and model. If it is, then system asks for additional confirmation. Answering 'yes' means, that the system deletes the file with the same name containing previous result data. Moreover, the name 'Neutral' is reserved and can not be used.

Two kinds of data are stored and restored as results, these are efficient values of objectives together with values of other, non-objective outcomes and values of variables related to the efficient solution. Values of variables are loaded into the spreadsheet (row Values in the model editing spreadsheet) following each successful calculation of an efficient solution and while restoring result from a disk. These obtained or restored values are used as a starting point for calculation of next efficient solution.

If the user finds, that calculated result is not interesting, it is possible to clear it either using the command < Alt R N > (Result selection — New) that clears only the system response or using the command < Alt R R > (Result selection — Reset) that clears also all user-supplied part of the problem definition. However, only results stored on a disk can be compared using graphical representation.

Result stored on a disk can be restored using the command < Alt R G > (Result selection — Get from disk). It is also possible to restore a result immediately while invoking the IAC-DIDAS-N system — the name of the model must be put as the first argument in the DOS command line, the name of the problem must be put as the second argument in the DOS command line and the name of the problem must be put as the third argument in the DOS command line (for example to restore automatically the result `R1` obtained for the problem `FIRST` defined for the model `DEMO`, invoke the system with the command `DIDASN DEMO FIRST R1` or `DIDASNE DEMO FIRST R1`).

Calculated result can be printed using the command < Alt R P > (Result selection — Print). Result stored on a disk, but actually not interesting to the user can be deleted from a disk using the command < Alt R E > (Result selection — Erase).

3.3 Editing with the spreadsheet

Two spreadsheets used in the IAC-DIDAS-N system are rather specialized. They differ from the standard ones (like Lotus 1-2-3) in two aspects. First, in the implemented spreadsheets there are predefined types of contents of all cells: there are dedicated cells for storing text, other cells for storing numbers and other cells for storing formulae. Secondly, the IAC-DIDAS-N has an integrated compiler with a symbolic differentiation facility, that compiles the formulae and produces binary codes for calculations of formula value and for calculations of all derivatives. Therefore two kinds of operation are defined for spreadsheet cells with formulae: compilation and calculation.

The top line in both spreadsheet contains pull-down menu entries and the bottom line contains the function keys meanings.

Both spreadsheets are built from two partially independent parts. First three columns from the left are common for both spreadsheets and have as many rows as there are outcomes in the model. If the model has more than 13 outcomes then only 13 are displayed and the spreadsheet is scrolled up and down according to moves of the spreadsheet marker. These three columns are used to define and display outcome definition: name, unit and status, from left to right, respectively. However, the status is the element of problem definition, therefore it can be accessed only from the interactive analysis spreadsheet. The fourth column of the model editing spreadsheet contains outcome formulae, but only their values are displayed, to display and edit the formula one cell from this column must be selected with the spreadsheet marker and then the < Enter > key causes the display of formula in a window.

In the upper left corner of the spreadsheet the status of the model, problem and result are displayed together with the amount of free memory in relation to the amount of free memory available while the system was loaded. For example FreeMem 22 % means that only

22% of the available memory were used for current model, problem and result definition. It is not recommended to use the system if the value displayed is below 20%, because of large amount of memory required temporarily during compilation of formula, optimization and graphical representation. The behavior of the system is not completely predicable in some out of memory situations. The last status value is the flag Auto ON/OFF toggled with the function key < F8 > or with the command < Alt S A > (Switches — Auto ON/OFF). This flag reflects the state of the automatic spreadsheet recalculation feature. If it is ON then the spreadsheet is recalculated after each change of any cell, otherwise the recalculation is only on explicit request command < F3 > (Calculate).

The second part of the model editing spreadsheet has as many columns as there are variables, parameters and outcomes; at the top of each column a type designator is displayed: 'var', 'par' or 'out', respectively. Each column contains: name, unit, upper bound, value, lower bound and values of derivatives, from top to bottom, respectively. First five items are elements of model definition, except values for variables that are accessible and can be changed in any phase of the work. Cells with values of outcomes are only for display and their contents can not be edited. Only up to four columns are displayed at a time. If there are more than four columns the spreadsheet is scrolled left and right according to moves of the spreadsheet marker. Just above the area with derivative values the texts 'Partial derivative values' or 'Total derivative values' are alternatively displayed. They are toggled with the function key < F7 > or with the command < Alt S T > (Switches — Totals ON/OFF) and reflect the type of derivatives that are calculated and displayed.

The second part of the interactive analysis spreadsheet has seven columns, but only five of them are displayed at a time, thus it is scrolled one column left or one column right according to the moves of the spreadsheet marker. Contents of these seven columns depends of the type of outcome and can be different for different rows of the spreadsheet. Descriptions of the columns change accordingly to the type of outcome where the spreadsheet marker currently is. For outcomes not selected as objectives the spreadsheet columns contain (from left to right): upper bound, next two columns are blank, last calculated or loaded value of solution, next two columns are blank, lower bound. For maximized objectives there are there: upper bound, utopia value, aspiration level, last solution value, reservation level, nadir value and lower bound. For minimized objectives there are there: lower bound, utopia value, aspiration level, last solution value, reservation level, nadir value and upper bound. At last, for stabilized objectives there are there: upper bound, upper utopia value, upper reservation level, last solution value, lower reservation level, lower utopia value and lower bound. At a first look this arrangement seems to be very complicated, but one can easily find that all values that should be compared each other are placed side to side and that in most cases all values at each row either decrease or increase while moving from left to right. If the problem is not fixed then the columns with aspirations and reservations (for minimized and maximized objectives; lower and upper reservations for stabilized objectives) are not displayed, thus the remaining five columns are displayed all the time and there is no need to scroll the spreadsheet horizontally. At the top of this part of the interactive analysis spreadsheet names and descriptions of current model, problem and result are displayed.

To move the spreadsheet marker eight cursor keys can be used to perform four regular and four oblique directions of move (for example < Home > cursor key moves the marker in the upper-left direction). Moreover, it is also possible to perform fast jumps from one part of the spreadsheet to another (only in the model editing spreadsheet). Fast jumps from the right part to the left part (namely to the formulae column) and back without change of the row and without scrolling the right part are performed with the keys combinations < Ctrl

Left > and < Ctrl Right >, whereas fast jumps from the bottom part to the top part (namely to the value row) and back without change of the column and without scrolling the bottom part are performed with the keys combinations < Ctrl PgUp > and < Ctrl PgDn >.

The way data are entered into a particular cell depends on a type of data the cell is destined to store. To edit the contents of particular cell one must move the spreadsheet marker to this cell. The edition of contents of a spreadsheet cell is performed with the use of two editors: formula editor for edition of formula (only the formulae column in the model edition spreadsheet) and the cell editor for all the other cells. In the latter case there are two possibilities: the modification of the previous contents or the input of a new contents. The < Enter > key causes the entry into the cell editor with the previous contents of the cell, any other key is treated as the first character input of the new contents, with the previous contents deleted. The cell editor allows the use of standard editing keys (< Backspace >, < Del >, < Left >, < Right >, < Home >, < End >) in two modes: insert and replace that are toggled with the < Ins > key. Current mode is indicated with the shape of the cursor — block cursor means insert mode, underline cursor means replace mode. There are two keys that ends the cell edition: < Enter > key means storing of a new data, < Esc > key means restoring of the previous contents.

The < Enter > key must always be pressed to start the edition of a formula, thus it is always the edition of the previous contents of the cell. The formula editor is a full featured window editor. Standard editing keys (< Backspace >, < Del >, < Left >, < Right >, < Up >, < Down >, < Home >, < End >, < PgUp >, < PgDn >) can be used in two modes exactly like in the cell editor. The function of two terminating keys is also the same. Moreover, to facilitate the edition of several formulae that are very similar or even with some identical parts, the concept of a buffer was implemented.

Any part of currently displayed formula can be marked (displayed in a reverse video mode): the beginning is marked with < F7 > function key, the end is marked with < F8 > function key. Marked area can be copied (duplicated) into the current cursor position with the use of < F10 > function key. Marked area can be deleted with the use of < F6 > function key, however to avoid unintentional deletes the cursor must be at the position just following the end of the marked block, otherwise an appropriate message is displayed and the block is not deleted. Marked area can be copied (duplicated) into the buffer using the function key < F5 > and then restored in the current cursor position using the function key < F10 >. The contents of the buffer is displayed in a window at the bottom of the screen. This window is closed (without deleting the contents of the buffer) using the function key < F9 > and while the formula editor is left, and opened again using the function key < F9 >. If the buffer contents is too long to fit the size of the window then < Up > and < Down > cursor keys are used to scroll it up and down, one line at a time.

3.4 Usage of the nonlinear solver

The nonlinear solver used in the IAC-DIDAS-N system is rather fast and robust. It's operation, however, depends on some parameters, that should be sometimes adjusted to the properties of particular model and problem. For small models (not more than ten variables and outcomes) the default system values can be successfully used. There are four ways of changing values of these parameters: permanent, for a model, for a problem and for a result. Parameters are changed using menu entry Options in either spreadsheet. If any of the parameters was changed then while leaving the menu system asks 'Do you want to make these changes permanent (Y/N) ?'. If the answer is yes, then the program updates itself on a disk

in such a way, that the new values will be default values, otherwise the changes will be only temporary, valid till the end of a current run of the system. Values of the parameters are stored on a disk in the model definition file and in the problem definition file, these values are restored while the model or problem file are loaded.

The first three parameters are used in stop tests of the solver:

Accuracy — the norm of gradient of the penalty function at the solution point must be not greater than the Accuracy value. The default value is 10^{-4}, for large, highly nonlinear models this value should be changed to 10^{-2} or even more.

Violation — the nonlinear constraints (bounds on outcomes) at the solution point must not be violated more than Violation value. The default value is 10^{-4}, but this value should also be increased for large models with many nonlinear constraints.

Iterations — the limit of iterations (recalculations of the spreadsheet). The default value is 1000.

The last two parameters define the shape of the scalarizing function:

Scaling exp. — It is the parameter α in the scalarizing function (see theoretical manual). The default value is 10, it should be decreased to 4 or 6 for large models (it must be an even number).

Ratio Asp/Res — It is the parameter δ in the scalarizing function (see theoretical manual). The default value is 0.1 and may be changed in a range 0.01–0.9. The selection of this value is rather a matter of taste and does not depend on the size of the problem.

Although there are two independent stop tests (Accuracy check together with Violation check and Iterations limit check) the user can interrupt the running optimization process by pressing < Ctrl Break > key combination. The best point obtained till this moment is then displayed as a solution, but the system does not treat this point as a solution.

Optimization runs are sometimes very time consuming, therefore to give the user an information that the system did not crashed, a sequence of letters is displayed. Each time the solver stops the move in a current direction a next letter is displayed. The letters are displayed in the right half of the second to the last line on the screen starting with the letter 'A' in center of this line. If all 40 positions are filled with 'A', they are cleared and letters 'B' ('C','D' etc.) are displayed.

There are five possible messages displayed at the end of optimization run. These are:

Optimal solution found — it is the most desirable message.

Required accuracy not attainable — it means, that the solution was found with the full numerical accuracy, but due to round-off errors the required accuracy was not obtained. The accuracy and violation parameters should be increased.

Iterations limit — solution not found — the iterations parameter should be increased.

Interrupted with < Ctrl-Break >	— the user pressed the < Ctrl Break > key combination.
Solver error NNN	— this message should never appear. [1]

3.5 Graphical representation of results

Several objectives and several results can be displayed simultaneously in a graphical bar form. The system has some internal rules of selecting results and objectives to be displayed. These rules can be summarized in the form: display of all objectives and of as many results as possible. The < F9 > (Graphics) command is used to display bar representation using these rules. However, the command < Alt G O > (Graphics — Objectives selection) can be used to select manually objectives to be displayed — up to ten objectives can be marked and next displayed. The command < Alt G R > (Graphics — Results selection) can be used to select manually results to be selected — up to ten results can be marked and next displayed. The user selection overrides the system rules, thus subsequent executions of < F9 > (Graphics) command cause the display of user selected objectives and results.

Two scaling methods are implemented in the graphical representation. In the first scaling method (Normal scale `ON`) each objective is scaled independently: the maximal level (top of the bar picture) is equal to the upper bound for displayed objective and the minimal level (bottom of the bar picture) is equal to the lower bound for displayed objective. This scaling method is very simple, but has two important drawbacks. First, very often the bounds range is much wider than the range of efficient solutions, therefore all solutions together with utopia and nadir points can be represented as a one point of he bar. Secondly, the objectives that form trajectories (in dynamic cases) are scaled independently and can not be compared.

In the second scaling method (Normal scale `OFF`) the range of efficient changes is determined for each objective independently:

- for minimized objectives the maximal level is equal to the value of the reservation level or the value of the solution, whichever is greater; the minimal level is equal to the value of the utopia level.
- for maximized objectives the maximal level is equal to the value of the utopia level; the minimal level is equal to the value of the reservation level or the value of the solution whichever is less.
- for stabilized objectives the maximal level is equal to the value of upper reservation level or the value of the solution, whichever is greater; the minimal value is equal to the value of lower reservation level or the value of the solution, whichever is less.

Next, for all groups of objectives with the same units the common scale is determined that is the distance from the lowest minimal level to the highest maximal level of all objectives within the group.

3.6 Menu and function keys description

Most commands are executed by entering a pull-down menu, moreover some most frequently used commands are also accessible through function keys. There are two sets of pull-down menu entries and two sets of function keys commands for two existing spreadsheets — model

[1] Please let us know if it does.

edition spreadsheet and multiobjective analysis spreadsheet; in particular, function keys are used to select one of the spreadsheet.

Pull-down menu commands could be invoked in two ways — through < Alt > key or through < Esc > key. In the former way the user must press the key related to the first (highlighted) letter of the pull-down menu entry (eg. **s** key for Switches menu entry) while holding down the < Alt > key — such action will be denoted as < Alt X > where **X** means the letter (eg. < Alt S >). In the latter way the user can press < Esc > key and the last used pull-down menu is entered — thus this way is useful for repeated commands.

In both ways, < Right > and < Left > cursor keys could be used to move from one menu entry to another. Commands within current menu entry could be selected either by moving the menu marker with < Up > and < Down > cursor keys and pressing the < Enter > key or by pressing the key related to the first (highlighted) upper case letter of the command name (in most cases it is the first letter of the command). Selection means the execution of the selected command or entry to the next level menu. Pressing the < Esc > key while in a menu causes exit from the current level of the menu either to the previous level menu or to the spreadsheet. If the menu window is too small to display all items then up and down arrows appear in the bottom line of the menu window and the menu is scrolled vertically according to the moves of the marking bar.

In the following description menu entries are terminated with a colon whereas commands are terminated with a dot. In the former case related menu items are listed below the description of the menu entry.

3.6.1 Menu for model edition

< Alt M > **M**odel selection: — commands for operations on models as whole entities

- **G**et from disk: — displays menu of currently defined models, selected model is loaded from disk into spreadsheet
- **F**ix and save. — checks for completeness of the model, fixes it and saves on disk (asks for model name if not named previously)
- **D**escription. — asks for 30 characters brief description of the model
- **N**ew. — un-fixes the model, resets the name and comment, the model in the spreadsheet remains unchanged
- **R**eset. — un-fixes the model, resets the name and comment, deletes all data in the spreadsheet, resizes the spreadsheet to one-row and one-column
- **E**rase: — displays menu of currently defined models, selected model is deleted together with all related problems and results
- **P**rint. — prints current model

< Alt R > **F**ormat: — operations on rows or columns of the spreadsheet

Rows: — changes the number or the order of rows in the spreadsheet

Insert. — inserts blank, highlighted row in the spreadsheet, this row can be moved up and down within a spreadsheet using < Up > and < Down > cursor keys, < Alt I > makes insertion permanent, < Esc > key cancels it

Delete. — highlights a row, it can be moved up and down within a spreadsheet using < Up > and < Down > cursor keys, < Alt D > deletes highlighted row (if not referenced in other rows) and related column, < Esc > key cancels the command

Move. — highlights a row, the highlighting bar can be moved up and down within a spreadsheet using < Up > and < Down > cursor keys, < Alt S > selects highlighted row. The selected row can then be moved within a spreadsheet, < Alt P > places the highlighted row in a current place, < Esc > key cancels the command

Columns: — changes the number, order or type of columns in the spreadsheet

Insert. — inserts blank, highlighted column in the spreadsheet, this column can be moved left and right within a spreadsheet using < Left > and < Right > cursor keys, < Alt V > makes insertion permanent as a column with a variable, < Alt P > makes insertion permanent as a column with a parameter (system asks for its initial value), < Esc > key cancels it

Delete. — highlights a column, it can be moved left and right within a spreadsheet using < Left > and < Right > cursor keys, < Alt D > deletes selected column (if not referenced in the spreadsheet, columns related to outcomes cannot be deleted), < Esc > key cancels the command

Move. — highlights a column, it can be moved left and right within a spreadsheet using < Left > and < Right > cursor keys, < Alt S > selects highlighted column. The selected column can then be moved within a spreadsheet, < Alt P > places the highlighted column in a current place, < Esc > key cancels the command

to **Var.** — highlights a column, it can be moved left and right within a spreadsheet using < Left > and < Right > cursor keys, < Alt C > changes the type of column from a previous type to a variable type (if it was an outcome then related row is deleted), < Esc > key cancels the command

to **Par.** — highlights a column, it can be moved left and right within a spreadsheet using < Left > and < Right > cursor keys, < Alt C > changes the type of column from a previous type to a parameter type (if it was an outcome then related row is deleted), < Esc > key cancels the command

to **Out.** — highlights a column, it can be moved left and right within a spreadsheet using < Left > and < Right > cursor keys, < Alt C > changes the type of column from a previous type to a outcome type (related row is created), < Esc > key cancels the command

< Alt S > **Switches:** — influence numerical calculations in the spreadsheet

Totals on/off. — toggles calculations and display of values of either partial or total derivatives

Auto on/off. — switches on and off the automatic recalculation of the spreadsheet following each change of its contents

< Alt C > **Calculate.** — recalculates the whole spreadsheet

< Alt L > **List:** — displays the menu of models stored on the disk, for the selected model displays the menu of problems stored on the disk, for the selected problem displays the list of results stored on the disk, < Enter > key selects, < Esc > key moves back

< Alt O > **O**ptions: — changes colors and some problem-dependent parameters in the nonlinear programming solver

Colors. — enables changes of foreground and background colors or attributes of all items displayed on the screen during the work of the program in an easy, interactive way, all changes are immediately visible on the screen, see Appendix B for details

Accuracy. — accuracy of optimization, if this value is too high then results could be misleading, if this value is too low then optimization time may be too large

Violation. — acceptable violation of bounds for outcomes, if this value is too high then results could be misleading, if this value is too low then optimization time may be too large

Iterations. — limit of iterations (recalculations of the model) during each optimization run

Scaling exponent. — coefficient α in the scalarizing function

Ratio Asp./Res. — ratio of the with of aspiration and reservation ranges for stabilized objectives

3.6.2 Function keys for model edition

< F1 > — context sensitive help.

< F2 > — save. — saves model (if changed), problem (if changed) and result (if changed), asks for names if not named previously)

< F3 > — calculate. — same as < Alt C >

< F4 > — list. — same as < Alt L >

< F6 > — go to multiobjective analysis spreadsheet.

< F7 > — totals on/off. — same as < Alt S T >

< F8 > — auto on/off. — same as < Alt S A >

< F10 > — exit to DOS.

3.6.3 Menu for interactive analysis

< Alt M > **M**odel selection: — same as in the model edition spreadsheet

< Alt P > **P**roblem selection: — commands for operation on problems as whole entities

	Get from disk.	— displays menu of currently defined problems (for current model), selected problem is loaded from disk into spreadsheet
	Fix and save.	— check for completeness of the problem, fixes it and saves on the disk (asks for problem name if not named previously)
	Description.	— asks for 30 characters brief description of the problem
	New.	— un-fixes the problem, resets the name and comment, the problem in the spreadsheet remains unchanged
	Reset.	— un-fixes the problem, resets the name and comment, deletes all problem data in the spreadsheet
	Utopia.	— checks for completeness of the problem and calculates the utopia point, approximates the nadir point (see theoretical manual)
	n**A**dir.	— enters special spreadsheet editing mode that enables user updates of the nadir point values, < Esc > key exits this mode
	neu**T**ral.	— checks for completeness of the problem and calculates the neutral solution (see theoretical manual)
	Erase.	— displays menu of currently defined problems (for current model), selected problem is deleted together with all related results
	Print.	— prints current problem
< Alt R >	**R**esult selection:	— commands for operations on results as whole entities
	Get from disk.	— displays menu of currently defined results (for current model and current problem), selected result is loaded from disk into spreadsheet
	Save and new.	— checks for completeness of the result, saves it on disk (asks for result name if not named previously) and resets the name and comment
	Description.	— asks for 30 characters brief description of the result
	New.	— resets the name and comment, the result data in the spreadsheet remains unchanged
	Reset.	— resets the name and comment, clears all result data in the spreadsheet

Calculate. — checks for completeness of the result data and calculates the efficient solution (see theoretical manual)

Variables. — displays a window with values of variables related to current efficient solution

Erase. — displays menu of currently defined results (for current model and for current problem), selected result is deleted

Print. — prints current result

< Alt G > **Graphics:** — selects the results and objectives which will be displayed,switches the scaling method and starts the display

Display. — displays the graphical representation of result

Result selection — displays menu of currently defined results (for current model and current problem), selected results are displayed on the screen, < Enter > key selects, up to 10 results can be selects, < Esc > key moves back to the Graphics menu

Objectives selection — displays menu of objectives, selected objectives are displayed on the screen, < Enter > key selects, < Esc > key moves back to the Graphics menu

Normal scale — toggles the scaling method

< Alt O > **Options:** — same as in model edition spreadsheet

3.6.4 Function keys for interactive analysis

< F1 > — context sensitive help.

< F2 > — save. — same as in model edition spreadsheet

< F3 > — calculate. — calculates utopia point (if not calculated), neutral solution (if not calculated) and efficient solution (if not calculated), performs all necessary checks for completeness of model, problem and result data

< F4 > — list. — same as < F4 > and < Alt L > in model edition spreadsheet

< F5 > — go to model editing spreadsheet.

< F9 > — graphics: — displays menu of currently defined results (for current model and current problem), up to 10 results can be selected and displayed in the bar form

< F10 > — exit to DOS.

3.7 Syntax of formulae

Outcome formulae entered into the spreadsheet are standard arithmetic expressions with some possible extensions. Five binary arithmetical operators can be used: addition '+', subtraction '-', multiplication '*', division '/' and power '^' , moreover an unary minus can be used, having higher precedence than binary operators. Standard arithmetical rules are used for operator precedence and calculation order, parenthesis can be used to imply specific order of calculations. There is only one restriction of the use of these operators: a sequence of two power operators x^{y^z} is not allowed, either operator together with its arguments must by enclosed in parenthesis $(x^y)^z$ or $x^{(y^z)}$ to explicitly define the order of calculations.

There are several built-in functions that can be used in outcome formulae, ten functions with one argument *abs*, *arctan*, *cos*, *exp*, *ln*, *log*, *signum*, *sin*, *sqr*, *sqrt* and two functions with two arguments *min*, *max*. Moreover there is a predefined constant *Pi*. Functions *abs*, *signum*, *min*, *max* should be used with caution because they are nondifferentiable. Logical structures of the form **if** *logical expression* **then** *arithmetic expression* **else** *arithmetic expression* or **if** *logical expression* **then** *arithmetic expression* **elsif** *logical expression* **then** *arithmetic expression* **else** *arithmetic expression* should be used also with caution for the same reason. Up to ten levels of **elsif** are allowed. Two arguments relation operators '<' ' <=' '=' '<>' '>=' '>' and three arguments membership operator 'in [.,.]' and logical operators 'and' 'or' 'xor' 'not' can be used in logical expressions.

4 Illustrative examples

4.1 Testing Example

This example was chosen because its multiobjective analysis is simple and can be performed analytically. It serves to test the correctness of the installation of the program and to check whether the hardware and software IBM PC compatibility of your computer is sufficient to use the DIDASN (or DIDASNE) program.

The model has four variables (xa xb xc xd), two parameters (za zb) and three outcomes (obj1 obj2 wrk).

The model is defined as follows:

Outcome equations:

$$\begin{aligned} \texttt{obj1} &= (\texttt{xa}-1)\texttt{\^{}}2 + \texttt{za} * (\texttt{xb}-1)\texttt{\^{}}2 + (\texttt{xc}-1)\texttt{\^{}}2 + \\ &\quad (\texttt{xd}-1)\texttt{\^{}}2 + \texttt{wrk} \\ \texttt{obj2} &= \texttt{xa\^{}}2 + \texttt{za} * \texttt{xb\^{}}2 + \texttt{xc\^{}}2 + \texttt{xd\^{}}2 + \texttt{wrk} \\ \texttt{wrk} &= \texttt{zb} * (\texttt{xa}-\texttt{xb})\texttt{\^{}}2 + (\texttt{zb}-\texttt{za}) * (\texttt{xc}-\texttt{xd})\texttt{\^{}}2 \end{aligned}$$

Bounds on variables and outcomes:

$$\begin{aligned} -10 \le \texttt{xa} \le 10 \\ -10 \le \texttt{xa} \le 10 \\ -10 \le \texttt{xa} \le 10 \\ -10 \le \texttt{xa} \le 10 \end{aligned}$$

$$-1 \leq \texttt{obj1} \leq 12$$
$$-1 \leq \texttt{obj2} \leq 12$$
$$0 \leq \texttt{wrk} \leq 100$$

Values of parameters:

$$\texttt{za} = 7$$
$$\texttt{zb} = 100$$

Initial values of variables:

$$\texttt{xa} = 0$$
$$\texttt{xb} = 0$$
$$\texttt{xc} = 0$$
$$\texttt{xd} = 0$$

The multiobjective nonlinear programming problem is to minimize objectives `obj1` and `obj2`, while the outcome `wrk` is floating (free).

The Pareto frontier in the objective space for this example can be determined analytically and has the form:

$$\sqrt{\texttt{obj1}/10} + \sqrt{\texttt{obj2}/10} = 1$$

with the utopia point `( 0.0, 0.0 )`, nadir point `( 10.0, 10.0 )`, and the neutral solution point `( 2.5, 2.5 )`.

Numerical results obtained during computation will be slightly different because of numerical errors and finite accuracy of calculations. Calculations in two versions of the system are performed using different hardware and software, therefore small differences between results obtained using these two versions could be observed. In the following example session expected results will be given for coprocessor version. If results for emulation version are different they will be given in braces { }. Attached figures are for coprocessor version only.

To go through the testing example, we will perform the following actions:

1. We execute the program `DIDASN` (or `DIDASNE`) at the DOS prompt.
2. We get initial banner with program name, version number and information about the authors (Fig. 1).
3. We press any key and get initial, smallest possible model editing spreadsheet (Fig. 2).
4. We load the model pressing keys:

< Alt M >	[model selection menu appears in the upper left corner of the screen]
< G >	[list of accessible models appears in the small window with the `DEMO` model name being the first, if it is not the first then select the `DEMO` name moving the marking bar with < Up > and < Down > cursor keys]

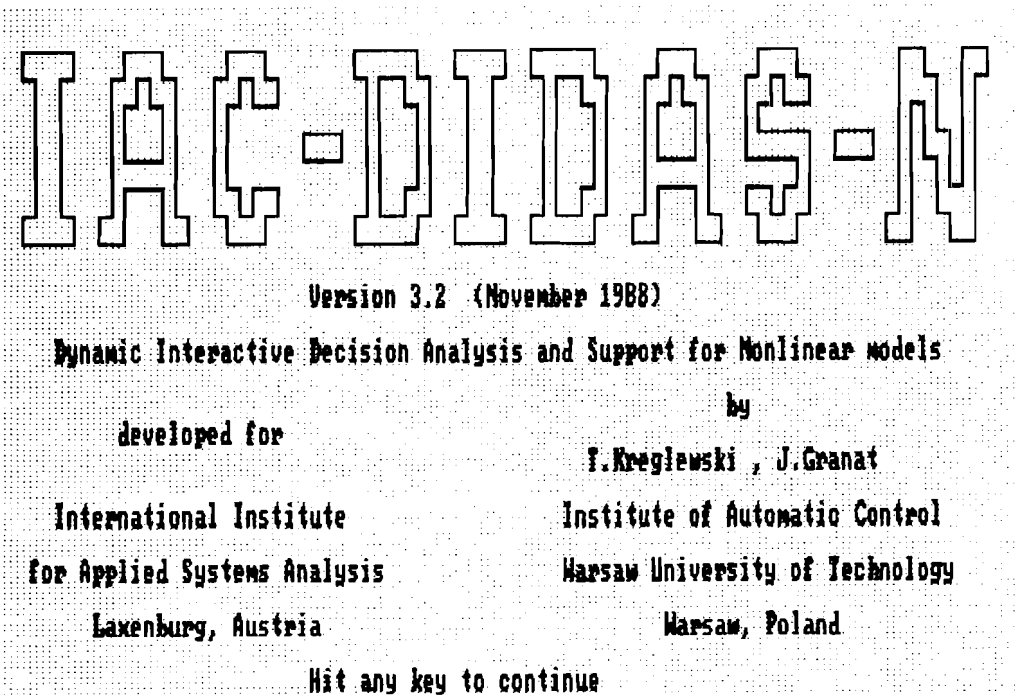


Figure 1: Initial screen

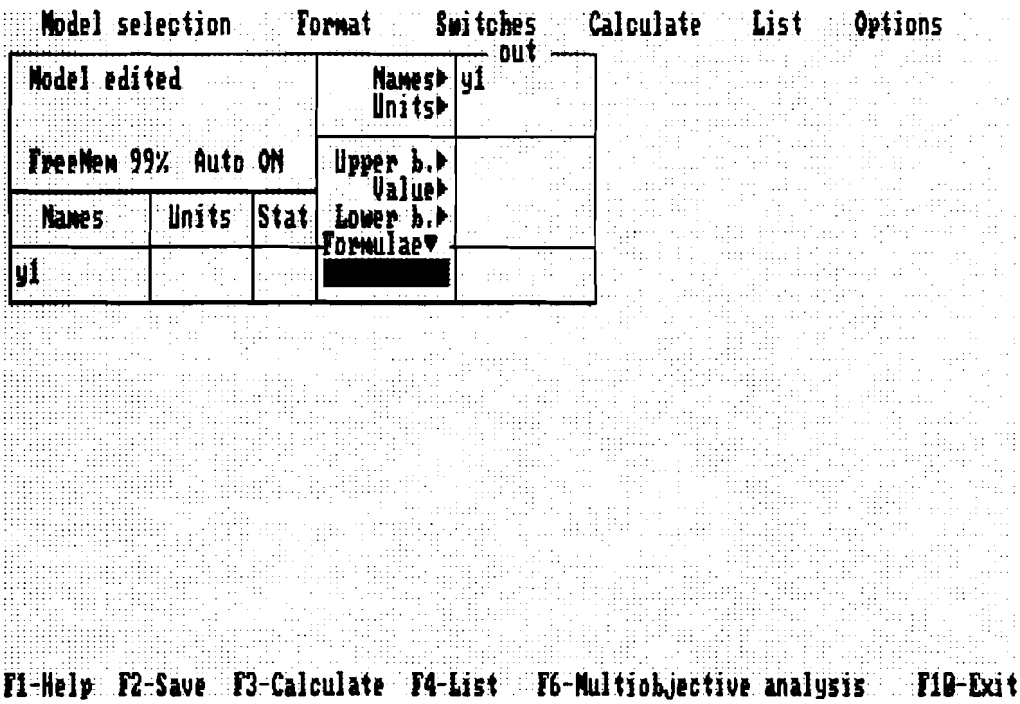


Figure 2: Initial spreadsheet

< Enter >	[DEMO model is loaded and displayed]
< F5 >	[DEMO model is stored on a disk with the status 'fixed', thus interactive analysis spreadsheet is automatically selected, we switch to model editing spreadsheet (Fig. 3)]

5. Using the cursor movement keys we move to the row Values and to the column xa. The marked cell contains the current value of the variable xa — this value is 1.0. We enter a new value, just typing 2 and pressing the < Enter > key. The spreadsheet is immediately recalculated (Fig. 4).

6. Now we switch to the multiobjective analysis spreadsheet pressing the < F6 > key and load example problem definition DEMO1 for the model DEMO pressing < Alt P > < G > < Enter >. The problem definition together with utopia and nadir points, neutral solution and some proposed aspiration and reservation levels are displayed (Fig. 5).

7. Now we will try to recalculate the problem and obtain the same results. First, we start the definition of a new problem pressing < Alt P N >. Now, we press < Alt P U > and the system determines the utopia point (analytical solution is (obj1, obj2) = (0.0, 0.0), obtained values are (2.714E-21, 1.351E-21 { 5.863E-18, 1.557E-17 }) and the nadir point (analytical solution is (obj1, obj2) = (10.0, 10.0), obtained results are the same).

8. Let the system determine the neutral solution — we press < Alt P T > (It is also possible to press single function key < F3 > instead of < Alt P N > and < Alt P T > to calculate in sequence utopia point, nadir point and neutral solution). Analytical solution is x = (0.5, 0.5, 0.5, 0.5), (wrk, obj1, obj2) = (0.0, 2.5, 2.5), obtained results for obj1 and obj2 are exactly the same, only value for outcome wrk is slightly different 1.564E-10. We check results for variables using the command < Alt R V > they are (0.499999, 0.5, 0.500004, 0.500005). Now we fix the problem (with the command < Alt P F > and giving it any valid name — e.g. demo2). Before doing it we can submit some problem description (with < Alt P D >).

9. Let the system determine an efficient solution corresponding to aspiration level of objective obj1 changed from 1.6667 to 1.0 and reservation level of objective obj2 changed from 3.3333 to 4.0; obtained results are 2.124 and 2.906 for objectives obj1 and obj2, respectively, and 6.982E-20 { 4.530E-13 } for outcome wrk. We check results for variables — they are 0.539114, 0.539114, 0.539114, 0.539114. We save obtained result (with < Alt R S > and giving it any valid name — e.g. myfirst). Before doing it we can submit some result description (< Alt R D >).

10. Now we can compare two results already obtained using graphical representation — for this purpose it is enough to press < F9 > function key. We obtain a screen with bars representing our results, it is better to change the scaling method pressing the key < F2 > (Fig. 6).

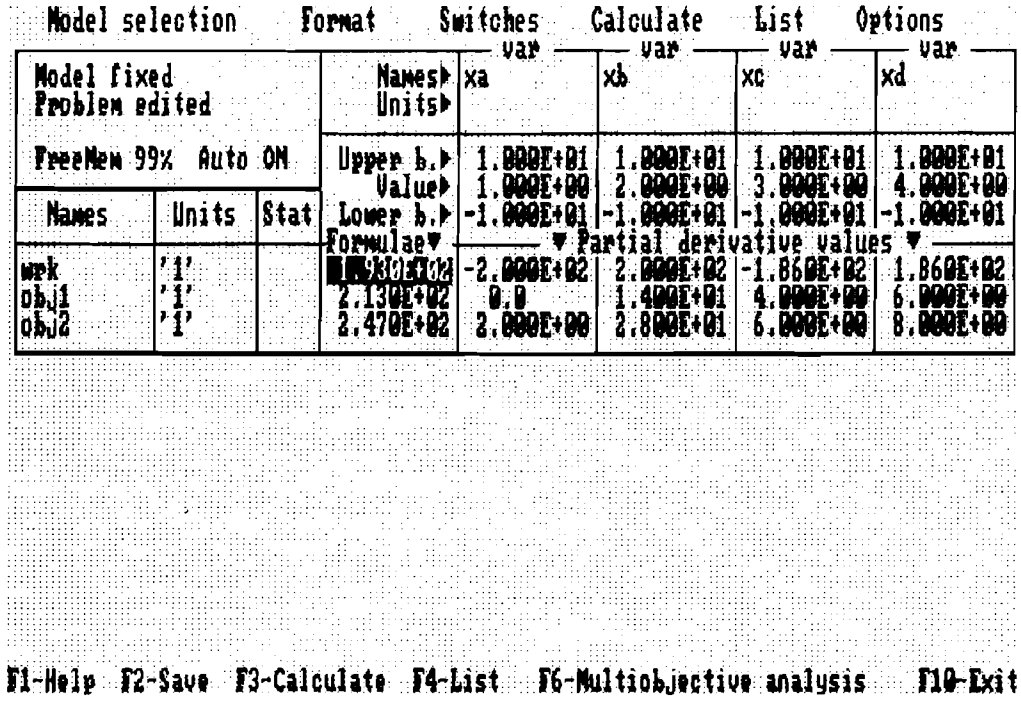


Figure 3: DEMO model loaded

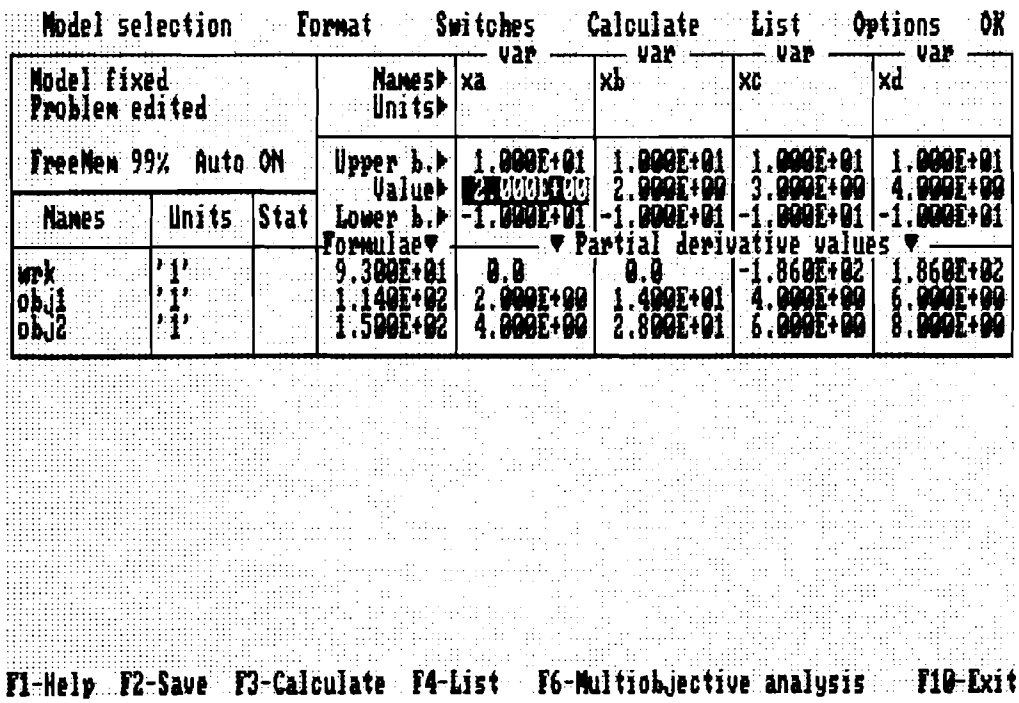


Figure 4: DEMO model recalculated

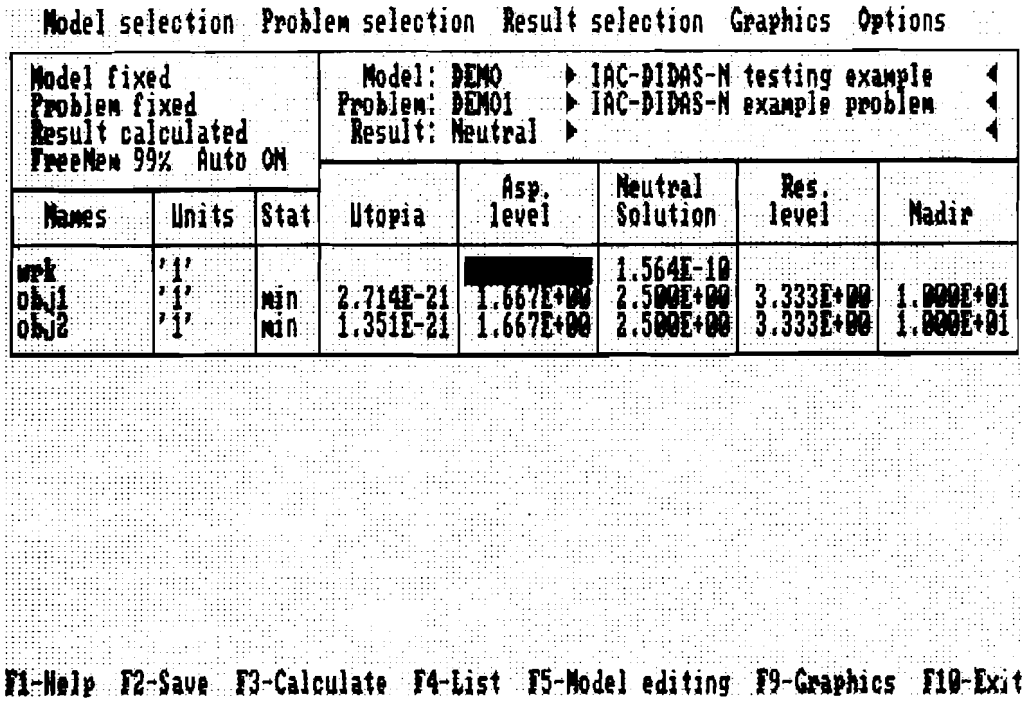


Figure 5: DEMO1 problem loaded

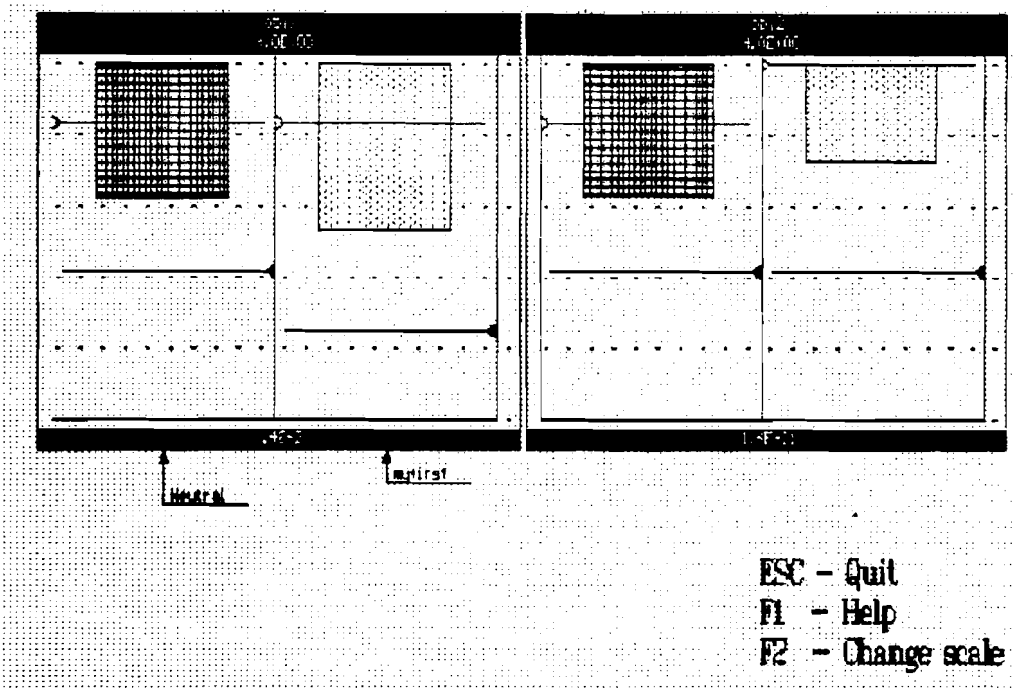


Figure 6: Graphical comparison

4.2 Tutorial example

Typical procedure of working with the DIDASN program and various accepts of use of several program commands are discussed in this section. This discussion is done using a real-life example specially designed for this purpose. The model used in this example is a very rough approximation of the much more complicated model of acid deposition in forest soil described by Hettelingh and Hordijk (1987).

4.2.1 Description of the model

We consider two regions (denoted by the index $k = 1, 2$) burning one type of fuel (say, coal) and emitting sulphur dioxide. The problem is, in fact, a dynamic one and should be considered in many time periods of one year duration; here we simplify it by considering only three periods each of five years duration (denoted by $t = 1, 2, 3$).

The sulphur dioxide emission in each region and time period is determined by:

$$S_{k,t} = S^p_{k,t}(1 - p_{k,t}) \tag{38}$$

where $S^p_{k,t}$ is the potential emission, specified exogenously. It may be described in the form:

$$S^p_{k,t} = E_{k,t}\frac{z_{k,t}}{h_{k,t}}(1 - r_{k,t}) \tag{39}$$

where $E_{k,t}$ is the total energy production in region k and in time period t, $h_{k,t}$ is the heat content of the fuel, $z_{k,t}$ is the sulphur content of the fuel, $r_{k,t}$ is the reduction coefficient through sulphur remaining in ashes. However, for the purpose of this simplified model, $S^p_{k,t}$ are assumed to be given as model parameters (for $k = 1, 2$ and $t = 1, 2, 3$). In the computerized model $S^p_{k,t}$ are denoted as parameter names Sktp and $S_{k,t}$ are denoted as outcome names Skt where k are digits $1, 2$ representing two regions and t are digits $1, 2, 3$ representing three time periods.

The reduction coefficients $p_{k,t}$ in (38) describe the effects of the pollution control measures. These coefficients serve as the main decision variables, therefore, there are actually six decision variables $p_{k,t}$ $(k = 1, 2,\ t = 1, 2, 3)$. In the computerized model they are denoted as variable names pkt.

It is assumed that the decision maker in k-th region is interested in:

- the costs of pollution control measures $C_{k,t}$ for each period;
- the level of pH (denoted here as $pH_{k,t}$ and denoted as outcome names pHkt in the model) in forest soil;

or together in two objective outcome trajectories each of three periods length. In the DIDAS methodology, however, we investigate cooperative actions of both decision makers jointly, therefore, the joint "decision maker" is interested in four outcome trajectories — two cost trajectories and two pH trajectories each of three periods length, a total of twelve objective outcomes.

The cost $C_{k,t}$ (denoted in the model as outcome names Ckt) is function of both potential emission $S_{k,t}$ and of the reduction coefficient $p_{k,t}$. Actually, the situation is more complicated, since the costs have also dynamic character: there is a high investment cost of pollution control devices, but they are not so expensive in maintenance; on the other hand, once installed, the devices give a defined coefficient $p_{k,t}$. However, when considering only five-year periods we can apply a much simpler model of the pollution control costs, understood as

joint cost of investments and maintenance in five-year period and depending on the reduction coefficient achieved in average in this period:

$$C_{k,t} = c_k S^p_{k,t} \frac{p_{k,t}}{2(1 - p_{k,t})} \tag{40}$$

where c_k is the cost of reducing the potential emission by half per one unit of potential emission. This is a very simple approximation of actual cost curves and can be replaced by any other more exact approximation. The form of this approximation express, however, the fact that it becomes increasingly costly to obtain reduction coefficients close to 1 . Because of numerical reasons, the reduction coefficient must be bounded away from 1 and constrained in a range, say, $0 \leq p_{k,t} \leq 0.99$ (in the computerized model reduction coefficients are measured in percents and bounded from 0% to 99%).

The level of pH in forest soil is assumed to have more long-time dynamic aspects and thus it is modeled by dynamic equations. When approximating more complicated relations described in (Hettelingh and Hordijk, 1987), we must take into account that acid absorption and reduction capacities of forest soil are nonlinear, that they are strongest in the carbonate range (pH=8.0–6.2, but we will take pH=7 as an upper bound), quite different and not so strong in the silicate range (pH=6.2–5.0) and again stronger in the cation exchange range (pH=5.0–4.2) while any pH level below 4.0 might be considered as catastrophic. Therefore, instead of including more realistic and complicated models that might be considered in further variants of this application, in the tutorial example we consider only a nonlinear dynamic model for the pH range 7–4 of the approximate form:

$$pH_{k,t} = \left(pH_{k,t-1} + \alpha_k \left(7 - pH_{k,t-1}\right)\right) \left(1 - \frac{3}{7} \Phi \left(\frac{D_{k,t}}{CAP_k}\right)\right) \tag{41}$$

where $D_{k,t}$ (denoted as outcome names DktR in the model) are sulphur deposits in given region and period, CAP_k are the five-year carrying absorption capacity (if $D_{k,t} \geq CAP_k$ then it is assumed that $pH_{k,t}$ drops to 4 or below), and the function Φ express the essential nonlinearity of absorption and reduction of acids by forest soil. A convenient form of this function is:

$$\Phi(x) = \begin{cases} 0, & \text{if} \quad x \leq 0 \\ 3x^2 - 2x^3, & \text{if} \quad 0 \leq x \leq 1 \\ 1, & \text{if} \quad x \geq 1 \end{cases} \tag{42}$$

This is a twice-differentiable (except at $x = 0$ and $x = 1$, where it is only once-differentiable) spline function.

If $x = D_{k,t}/CAP_k = 0$, then the dynamic part of (41) illustrates the self-regeneration of forest soil with a regeneration coefficient α_k (during a five-year period); this coefficient characterizes, what part of the distance between pH=7 and the actual pH level will be restored in five years.

The initial value $pH_{k,0}$ (for $t = 0$) is given as a parameter for both regions $k = 1, 2$. It should be stressed again that the nonlinear dynamic model (41) is only a very rough approximation of actual forest soil chemistry and must be updated by specialists for more realistic policy analysis for other than only tutorial example purposes.

The sulphur deposits $D_{k,t}$ are the results of sulphur-emissions $S_{k,t}$ as determined by a deposition model, which in this simplified case is again assumed in the simplest possible form:

$$D_{k,t} = a_{k,1} S_{1,t} + a_{k,2} S_{2,t}, \qquad k = 1, 2 \tag{43}$$

where $a_{k,j}$ are transfer coefficients from region j to region k (whereas $0 \leq a_{k,j} \leq 1$, and, for simplicity, we assume $a_{k,1} + a_{k,2} = 1, \quad k = 1, 2$).

On such a simplified model, we can illustrate the issues of multiobjective dynamic and nonlinear analysis of the effects of pollution control. The analysts or the decision makers can jointly analyze in this model:

- what would be the maximal pollution reduction rates, if they have limited founds for pollution control in each of time periods, and what would be the corresponding effects on forest soil acidity;
- what are the possibilities of multiobjective dynamic compromises between the trajectories of costs in all periods and trajectories of forest soil acidity.

For both purposes, the DIDAS methodology can be applied. The multiobjective analysis can be performed by specifying reference (aspiration, or aspiration and reservation) trajectories for costs and for the pH levels, while the DIDASN system will compute multiobjectively optimal (effective) trajectories for these variables that are consistent with the model (feasible) and in a sense best attuned to the reference trajectories.

4.2.2 Sample session

The model described in the previous section is already prepared as a disk file RAIN and can be loaded into the IAC-DIDAS-N spreadsheet using the command < Alt M G > (Model selection — Get from disk). It is stored as a 'fixed' model, thus to make some experiments with the model we must use the command < Alt M N > (Model selection — New). Now we can change all upper and lower bounds, values of parameters and outcome formula. Following each change of variable or parameter value the spreadsheet is automatically recalculated. After some play with the model we load again the original one and start the second phase of the work.

We define now the decision problem. We select outcomes to be minimized or maximized. First we get take into account only one region performances: we mark costs C11 C12 C13 as minimized and pH levels pH11 pH12 pH13 as maximized. Bounds for all outcomes are defined in the model and we don't redefine them. We ask the system to calculate Utopia and Nadir points using the command < Alt P U > (Problem selection - Utopia) and after a while we get the results. Next we ask the system to calculate a compromise solution, so called neutral solution, being the starting point for further interaction. We enter the command < Alt P T > (Problem selection — neuTral) and again wait a while. When the neutral solution is calculated and displayed the problem definition is finished and we can 'fix' the problem using the command < Alt P F > (Problem selection — Fix and save). We are asked to give a name of the problem, it may be ONEREG. Basing on values of Utopia, Nadir and Neutral points the system proposes us initial values of aspiration and reservation levels. Further interaction consists in a sequence of three or four actions:

- modification of aspiration and or reservation levels (it is enough to change only one value); it is obtained through the edition of appropriate spreadsheet cells.
- calculation of the efficient solution corresponding to current levels of aspirations and reservations. Optimization process is initiated with the < F3 > (Calculate) command.
- if the result (efficient solution) is not satisfactory, we can discard it with the command < Alt R N > (Result selection — New) and go back to the first action. Otherwise we save the result with the command < Alt R S > (Result selection — Save and new).

- optionally, we can compare several results obtained for current problem using graphical representation, directly with the command < F9 > or with some selections of objectives and results to be displayed within < Alt G > (Graphics) menu.

Because all interesting results are stored on a disk, interaction session can be stopped at any time and next resumed.

Now we continue the interaction, but for the problem previously defined. We load a problem `RAIN1` using the command < Alt P G > (Problem selection — Get from disk). The problem definition with calculated utopia and nadir values together with the neutral solution are loaded. There are twelve objectives now: costs `C11 C12 C13` and pH levels `pH11 pH12 pH13` for the first region and costs `C21 C22 C23` and pH levels `pH21 pH22 pH23` for the second region. Please observe, that neutral solution values for the first region are worse than in the previous problem. It is because now the neutral solution is a compromise between interests of both regions.

We find, that costs in the second region are decisively to large, but pH in both regions can be accepted. Thus we try to decrease costs in the second region decreasing reservation levels for costs `C21 C22 C23` from 2180 to 1500. We press < F3 > ,wait for result and save the result with th command < F2 >. To compare new result with the neutral solution we use graphical representation. First we select objectives to be displayed - only ten can be displayed simultaneously. We enter the command < Alt G O > (Graphics — Objectives selection). System display the list of all twelve objectives with first ten marked. More important for us are changes of pH in the last period than in the first. Therefore we 'unmark' objectives `pH11 pH21` and 'mark' objectives `pH22 pH23` - both operations are performed moving the marking bar with the < Up > and < Up > cursor keys and pressing the < Enter > key. We don't need to enter the Graphics — Results selection menu because currently there are only two results, thus both are automatically selected. We execute now the display command in the graphics menu and obtain bar representation of results. We can press the < F1 > (Help) function key to get help on the meaning of several elements of the picture, but we find the picture rather not legible. The standard scaling method (Normal scale ON) is based on the distances from lower to upper bounds. In our case the changes of efficient values are much smaller than these distances. It seems that the second (Normal scale OFF) scaling method will be useful – it is based on distances between utopia and reservation values (or solution values if they are worse than reservations). The < F2 > key during the graphical representation toggles both scaling methods, we press this key once.

Now the picture is legible and we can see that in fact costs in the second region are decreased, but simultaneously in the first region costs are increased and pH levels are decreased. Now we can either increase back the reservations for costs in the second region or increase aspirations and/or reservations for pH levels in both regions. We try to explore the second possibility. We increase reservations for `pH12 pH13` from 5.983 to 5.990 and from 6.265 to 6.270, respectively, and again calculate efficient solution, save it and look on graphical representation – we press in sequence three function keys: < F3 > (Calculate) < F2 > (Save) and < F9 > (Graphics). The previous selection of objectives is still active and now three results are displayed.

pH levels are now acceptable, but the costs in the first region are very high. To balance the costs in both regions we slightly increase reservations in the second (from 1500 to 1700) region and decrease reservations in the first region (from 1090 to 900). The fourth obtained result seems to be close to the acceptable solution of multiobjective decision problem.

5 References

Dreyfus, S. (1984). Beyond rationality. In M. Grauer, M. Thompson, A. P. Wierzbicki (eds), Plural Rationality and Interactive Decision Processes, Proceedings Sopron 1984. Springer Verlag, Berlin Heidelberg New York Tokyo (Lecture Notes in Economic and Mathematical Systems 248).

Hettelingh, J. P. and L. Hordijk (1987). Environmental Conflicts: The Case of Acid Rain in Europe. RR-87-9, International Institute for Applied Systems Analysis, Laxenburg, Austria.

Kaden, S. (1985). Decision support system for long-term water management in open-pit lignite mining areas. In G. Fandel, M. Grauer, A. Kurzhanski and A. P. Wierzbicki (eds), Large Scale Modeling and Interactive Decision Analysis, Proceedings Eisenach 1985. Springer Verlag, Berlin Heidelberg New York Tokyo (Lecture Notes in Economic and Mathematical Systems 273).

Kaden, S. and T. Kreglewski (1986). Decision support system MINE — problem solver for nonlinear multi-criteria analysis. CP-86-5, International Institute for Applied Systems Analysis, Laxenburg, Austria.

Kreglewski, T. and A. Lewandowski (1983). MM-MINOS — an integrated decision support system. CP-83-63. International Institute for Applied Systems Analysis, Laxenburg, Austria.

Korhonen, P. (1985). Solving discrete multiple criteria decision problems by using visual interaction. In G. Fandel, M. Grauer, A. Kurzhanski and A. P. Wierzbicki (eds), Large Scale Modeling and Interactive Decision Analysis, Proceedings Eisenach 1985. Springer Verlag, Berlin Heidelberg New York Tokyo (Lecture Notes in Economic and Mathematical Systems 273).

Lewandowski, A., M. Grauer, A. P. Wierzbicki (1983). DIDAS: theory, implementation. In M. Grauer, A. P. Wierzbicki (eds), Interactive Decision Analysis, Proceedings Laxenburg 1983. Springer Verlag, Berlin Heidelberg New York Tokyo (Lecture Notes in Economic and Mathematical Systems 229).

Lewandowski, A., T. Kreglewski, T. Rogowski, A. P. Wierzbicki (1987). Decision Support Systems of DIDAS Family. In A. Lewandowski, A. P. Wierzbicki (eds), Theory, Software and Testing Examples for Decision Support Systems. WP-87-26, International Institute for Applied Systems Analysis, Laxenburg, Austria.

Lewandowski, A., A. P. Wierzbicki (1988). Aspiration Based Decision Analysis and Support, Part I: Theoretical and Methodological Backgrounds. WP-88-3, International Institute for Applied Systems Analysis, Laxenburg, Austria.

Makowski, M., and J. Sosnowski (1984). A decision support system for planning and controlling agricultural production with a decentralized management structure. In M. Grauer, M. Thompson, A. P. Wierzbicki (eds), Plural Rationality and Interactive Decision Processes, Proceedings Sopron 1984. Springer Verlag, Berlin Heidelberg New York Tokyo (Lecture Notes in Economic and Mathematical Systems 248).

Messner, S. (1985). Natural gas trade in Europe and interactive decision analysis. In G. Fandel, M. Grauer, A. Kurzhanski and A. P. Wierzbicki (eds), Large Scale Modeling and Interactive Decision Analysis, Proceedings Eisenach 1985. Springer Verlag, Berlin Heidelberg New York Tokyo (Lecture Notes in Economic and Mathematical Systems 273).

Michalevich, M. (1986). Stochastic approaches to interactive multicriteria optimization problems. WP-86-10. International Institute for Applied Systems Analysis, Laxenburg, Austria.

Wierzbicki, A. P. (1983). A mathematical basis for satisficing decision making. *Mathematical Modeling* 3, 391–405.

Wierzbicki, A. P (1984). Models and Sensitivity of Control Systems. Elsevier, Amsterdam.

Wierzbicki, A. P. (1986). On the completeness and constructiveness of parametric characterizations to vector optimization problems. *OR Spektrum* 8, 73–87.

A Installation guide

The distribution diskette 1 contains the following files:

DIDASN.EXE	— compiled code of the coprocessor version of the program
DEMO.MOD	— simple nonlinear model (testing example)
RAIN.MOD	— nonlinear model used in Tutorial Example
READ.ME	— last time notes and corrections
INSTALL.BAT	— batch command file to install coprocessor version of the program and both models on a hard disk or on a working diskette

Moreover, there are examples of multiobjective problem definitions for both model examples in two diskette subdirectories (DEMO and RAIN).

The distribution diskette 2 contains the following files:

DIDASNE.EXE	— complied code of the emulation version of the program
DEMO.MOD	— simple nonlinear model (testing example)
RAIN.MOD	— nonlinear model used in Tutorial Example
READ.ME	— last time notes and corrections
INSTALL.BAT	— batch command file to install emulation version of the program and both models on a hard disk or on a working diskette

Moreover, there are examples of multiobjective problem definitions for both model examples in two diskette subdirectories (DEMO and RAIN).

To install the selected version of the program and models:

- insert the distribution diskette 1 or 2 in floppy disk drive,
- change current drive and current directory to the drive and directory where you want to install the program,
- enter the command

  `a:install a:`

  where a is a drive letter of the drive where the distribution diskette is inserted (typically it is just a drive A)

Installation procedure makes the sub directory DIDASN, writes there the program code (DIDASN.EXE or DIDASNE.EXE), next makes subdirectory DIDASN\MODELS, creates there some more subdirectories and writes into them two demonstrative nonlinear models together with examples of problems and results.

B Selection of colors

Both versions of the program can work on IBM PC/XT/AT or compatible computers with Hercules Graphics Card (HGC), Color Graphics Adapter (CGA) and Enhanced Graphics Adapter (EGA or VGA). The user can select (independently for each particular graphics card) his own set of colors/attributes for display several items on the screen using model editing spreadsheet menu entry Options — command Colors.

There are 9 items with some restrictions of selections of their colors:

Spreadsheet cells
Spreadsheet lines
Spreadsheet labels
Spreadsheet marker
Editing windows
Message windows
Warning messages
Error messages
Screen background

First three items form spreadsheets and therefore have the same background color, the spreadsheet marker should have colors that make it visible in all spreadsheet cells (empty and not empty).

Highlighted double-triangles marker selects one of 9 items marker and can be moved using < Up > and < Down > cursor keys. For selected item < Home > and < End > cursor keys change the foreground color, whereas < PgUp > and < PgDn > cursor keys change background color. Selected item is displayed in the colors/attributes currently selected for it. Simultaneously names of the colors are displayed on the right.

< Esc > key cancels all colors changes and causes return to the spreadsheet. < Enter > key makes the changes effective and, additionally, program asks for making these changes permanent. In case of answer `Y` (or y) the changes are recorded on disk and will also be effective during subsequent runs of the program. Selections of colors for two versions of the program (`DIDASN.EXE` and `DIDASNE.EXE`) are independent.

Attachment

Nonlinear Model Generator IAC-DIDAS-G

1 Extended summary

In many situations a decision maker needs help of an analyst or a team of analysts to learn about possible decision options and their predicted results. Such situations include both purely engineering complex design and mixed social economical and environmental problems. A team of analysts frequently summarizes its knowledge in the form of a substantive model of the decision problem that can be formalized mathematically and computerized. A mathematical model can never be perfect but, nevertheless, it can often be of great help to a decision maker in the process of learning about novel aspects of a decision situation.

The learning process needs interaction of a decision maker with a team of analysts and substantive models prepared by them. In organizing such interaction, many techniques of optimization, multicriteria decision analysis and other tools of mathematical programming can be used. However, all such techniques must be used as supporting tools of interactive analysis rather than as means for proposing unique optimal decisions and thus replacing the decision maker.

Such considerations led to the construction of the decision analysis and support systems of DIDAS family — that is, Dynamic Interactive Decision Analysis and Support systems, see e.g. (Lewandowski at al., 1985). DIDAS systems are addressed to analysts or teams of analysts who want to analyze their substantive models and, if the system is user-friendly, even to decision makers working alone.

The most general classification of models is into linear and nonlinear ones. Problems described by mathematical models of linear structure were investigated by many authors and several approaches to defining such problems interactively were proposed. The situation is much more complicated in the case of nonlinear models. A short review (see: Lewandowski, 1985) of problem interfaces in existing implementations of nonlinear DIDAS systems will underline the main points. Before the fulfillment of the present contract there existed three versions of nonlinear DIDAS: DIDAS-N developed by Grauer and Kaden (1983), a specialized system developed by Kaden and Kreglewski (1985) and general purpose DIDAS implemented by Kreglewski and others (Kreglewski at al., 1985). In the Grauer and Kaden's version, the equations describing objective and constrains functions must be programmed in FORTRAN. The authors supply a "skeleton" FORTRAN subroutine with empty "holes" where the user must locate his FORTRAN code. This is rather a complicated task — separate parts of the problem definition must be located in various places of the code, and the code itself must be written taking into account the variable names and structures used in this skeleton subroutine. Next, the user must calculate analytically the derivatives of the objective and the constraints functions. This is needed in each version of DIDAS, since practically only differentiable optimization methods are sufficiently efficient and robust to be applied in interactive decision support systems. This task is time consuming and can be a source of errors. A warning for persons who believe in their error-free analytic calculations can be found in (Pavelle at al., 1981), in a slightly different context. They state that a computer verification (by means of symbolic computation) of eight widely employed tables of indefinite integrals discovered that about 10 percent of the formulae were erroneous; one of the tables was found to have an error rate of 25 percent. Errors in gradients are typically difficult to detect — they can only be detected when the behavior of an optimization process is observed by a user qualified

in optimization techniques. After completing this analytical task, the user must properly augment his FORTRAN formulae with the penalty function terms and their derivatives. This is conceptually rather difficult for a user who is not familiar with mathematical programming algorithms, and can lead to numerous errors. The conflict of variable names is also probable.

The specialized system of Kaden and Kreglewski is of no interest in the present context, since its model was programmed once for ever and the user interacts only on the level of input data and reference point selection. The general purpose version developed by Kreglewski and others (1985) also needs a FORTRAN subroutine containing the problem description, but there are less programming constrains put on the user. He must preserve only the general structure of the subroutine header (formal parameter declaration) and the COMMON block. No variable conflict can occur, and the standards according to which the body of the subroutine must be composed are straightforward. The task of analytical computation of derivatives remains but the errors can most probably be detected aposteriori by the method of Wolfe (1982).

Concluding, the following basic disadvantages of problem generation are found in the past implementations of nonlinear DIDAS:

- the user must compute analytically all derivatives of objective and constraint functions,
- all objective and constraint functions as well as their derivatives must be programmed in FORTRAN according to the specifications supplied by the manual of the system; those specifications can be difficult to understand for a non-experienced user,
- the user must be familiar with the details of the computing environment of the computer on which he is working, such as editor, compiler, linker, operating system command language etc.
- any changes of the model — which might often occur during the process of interactive work with the system — cannot be performed within the system, but must go through the chain: program editor — FORTRAN compiler — linker — operating system. This slows down the interaction process, makes it difficult and inefficient.

The new model generator is aimed at a user who is neither an experienced computer specialist nor an expert in optimization techniques. The work done can be divided into the following parts:

- a proposal of a standard for nonlinear model formulation (thus extracting a concrete subclass from the wide universe of nonlinear problems that is characterized rather too widely by: "a nonlinear problem is a problem, which is not linear"),
- imbedding this model in an easy and user-friendly format of a spreadsheet,
- automatic computing of the necessary derivatives by symbolic differentiation procedures,
- providing safe environment for numerical calculations,
- providing means for convenient model simulation.

The proposed standard for model formulation helps to classify thinking about model in the user's mind. All variables of the model are divided into input and output variables or outcomes. The input variables can be further subdivided into decision variables and

parametric variables. The model equations are of explicit type: outcome variables are defined consecutively, depending on input variables and previously defined outcome variables (which serve then as intermediate outcomes). Those dependencies are checked automatically by the program.

In contrast with the previous versions of the model generator, this program has two packages for automatic differentiation. The first one produces code for calculation of the derivative's value while the second presents derivatives in symbolic form. This solution results from our experience from earlier versions. Efficient computations of values should repeatedly use values of common subexpressions. Such code, even after transformation to some symbolic form would not be very human-readable and would bear no resemblance to a formula produced "by hand". But the goal in symbolic differentiation is to produce a formula similar to that obtained by manual differentiation. The coding of such a formula would not necessary produce efficient numerical code since there may be many repetitions of similar blocks of the code.

The domain of differentiation is extended in comparison with general purpose symbolic computation packages. In optimization there is a long tradition of using nondifferentiable functions e.g. `MAX`, `MIN`. In such cases program calculates arbitrarily selected element of their subdifferentials. Their correct interpretation remains the responsibility of the user.

Properties of all but "academic" nonlinear models can contain many "mysteries". One cannot hope that the first optimization run will yield a solution of the problem. Most likely, the procedures of an optimization solver will encounter numerical problems. The user can investigate analytical and numerical properties of the model by inspecting formulae and their derivatives and performing numerical simulations. After gaining some insight he should be able to improve the formulation of the model.

In particular, the algorithm of shifted penalty functions used by the optimization solver in nonlinear DIDAS systems can drive the model formulae outside of their domains of definition. Such situations can be easily dealt with thanks to properties of the program: the syntax of formulae allows for an easy extension of the model's definition domain. Corrections can be introduced interactively because of general user-friendliness of the model generator.

The model generator was designed as a part of a new nonlinear version of the DIDAS system, it is available however as a separate program too. In the latter case it forms an extension to IAC-DIDAS-N, called IAC-DIDAS-G. Model files of both programs are identical.

2 Theoretical manual

2.1 General layout

Spreadsheet structure of the model generator suggests the general layout of the user's screen. One row of cells, say in horizontal direction, corresponds to input variables. Then a natural choice is to define output variables in a column of cells. The rectangular area of cells thus defined can then be interpreted as (partial or total) derivatives of outcomes with respect to inputs. An outcome can depend on previously defined outcomes too. Therefore the "input" row is augmented to include outputs. In the additional rectangular area only cells in the lower triangle are interpreted as derivatives, the cells on the diagonal and in in the upper triangle remain void. An important problem remains however open — how to define an outcome variable?

In one solution an output cell can correspond to a program in some universal language e.g. FORTRAN. Such solution would inherit the majority of drawbacks of past DIDAS implementations, the whole expressive power of a programming language would however be at

disposal.

The other solution, accepted in the presented program, is to constrain the allowable language constructs to expressions. The class of expressions is augmented in comparison with typical statement-oriented programming languages. Such approach removes all drawbacks of "classical" implementations. The users need very little training with such expression-oriented "language" to use it correctly and efficiently. However the price of simplicity is the limited expressive power of the language.

Another solution is to introduce some functional language as means for definition of the outcome variables. Such definition can be concise and the expressive power is not restricted. The only problem is the ability and willingness of the user to master another programming language, which is conceptually completely different from languages used in the field of numerical calculations.

2.2 Syntax of formulae

The syntax of formulae used in the spreadsheet should be as "natural" as possible, i.e. near to one's notational habits. These habits are created by mathematical education and by experience in programming languages. However, a closer look reveals that those sources are completely inconsistent.

Formulae are composed usually of numerical constants, cell names, parenthesis, some set of standard functions and operators denoted by the characters `+`, `-`, `*`, `/`, `^`. Two of them, `+` and `-`, have nonunique meaning — they denote both the binary operators of addition and subtraction and the unary operators of plus and minus (change of sign). The operator properties are usually described by their precedence and associativity. It seems however that school mathematics establishes accepted conventions only for binary operators. Further patterns can be sought in popular programming languages and tools. Such review depicts only the total lack of commital conventions.

The most exotic rules are used in APL. All operators have the same precedence level and are right-associative. This contradicts even the "school" rules. Leaving this case out of considerations, most problems arise from the treatment of unary minus. Aho and Ullman (1977) give a dramatic warning: "Beware the treatment of unary minus!". Three different syntaxes can be considered as eligible candidates for the spreadsheet. They will be presented below using the notation of Modified Backus-Naur Form. The meaning of meta-symbols is as follows:

`=`	—	denotes the definition,
`|`	—	separates alternative options within the clause,
`"..."`	—	terminal symbols are quoted,
`(...)`	—	exactly one of the enclosed alternatives must be selected,
`[...]`	—	denotes zero or one occurrence of the enclosed subclause,
`{...}`	—	denotes zero or any number of occurrence of the enclosed subclause.

The syntax definitions below are left uncompleted. Lacking definition of *factor* will be presented later in two variants, any of them can be composed with previous syntaxes giving the total number of six variants.

Syntax S1

```
expression = ["+"] simple_expression { ("+" | "-") simple_expression }
simple_expression = term { ("*" | "/") term }
term = signed_factor { "^" signed_factor }
signed_factor = ["-"] factor
```

Syntax S2

```
expression = ["+"] simple_expression { ("+" | "-") simple_expression }
simple_expression = signed_term { ("*" | "/") signed_term }
signed_term = ["-"] term
term = factor { "^" factor }
```

Syntax S3

```
expression = ["+" | "-"] simple_expression
             { ("+" | "-") simple_expression }
simple_expression = term { ("*" | "/") term }
term = factor { "^" factor }
```

The S1 syntax is used e.g. in ALGOL 68, SNOBOL, MICROCALC (a demonstration spreadsheet program supplied by Borland together with TURBO-PASCAL). From the definition, the following operator precedences can be read:

$$\text{- (unary)} \quad > \quad \hat{} \quad > \quad *\,,\,/ \quad > \quad +\,,\,\text{- (binary)}.$$

The S2 syntax is used e.g. in FORTRAN, BASIC, PL/I, Lotus 1–2–3 by Lotus (1983), MUSIMP (the implementation language for MUMATH symbolic computation system by Microsoft (1983)). It has the following precedences:

$$\hat{} \quad > \quad \text{- (unary)} \quad > \quad *\,,\,/ \quad > \quad +\,,\,\text{- (binary)}.$$

The S3 syntax is that of PASCAL extended with the power operator. Its precedences are:

$$\hat{} \quad > \quad *\,,\,/ \quad > \quad +\,,\,\text{- (unary and binary)}.$$

Each syntax has its peculiarities, e.g.

$$\begin{array}{ll} \text{in S1:} & -2\hat{}2 = 4, \qquad 4 - 2\hat{}2 = 0; \\ \text{in S2, S3:} & -2\hat{}2 = -4. \end{array}$$

The syntax rules must be supplemented by associativity rules. In our context they are essential only for the power operator. The problem is whether it is right-associative, i.e. `a^b^c` evaluates as `a^(b^c)`, or left-associative: `(a^b)^c`, as typically tacitly assumed in elementary schools. Both cases lead to completely different values, e.g. `2^(2^3)=256` while `(2^2)^3=64`. Thus the semantics of a formula depends on the chosen rule. Associativity affects not only the process of formula evaluation, but even the process of computation of derivatives. For the right-associative power operator

```
d(e^x^2)/dx = 2xe^x^2,
```

(e denotes here the base of natural logarithms), while for the left-associative one

```
d(e^x^2)/dx = 2e^(2x).
```

It seems that courses in differential calculus do not explicitly declare associativity but tacitly use the first variant, see e.g. examples in the classical russian textbook (Fichtenholz, 1966), vol. I, par. 99. MUMATH assumes right associativity (under normal setting, properties of all operators can be freely modified by the user). Similarly does MACSYMA, perhaps the world's largest computer algebra system, see a differentiation example in (Pavelle, 1985).

Associativity rules used in programming languages are nonunique. Aho and Ullman (1977) state on p. 47: "ALGOL evaluates all binary operators left-associatively. FORTRAN lets the compiler designer choose the associativity, and PL/I evaluates all binary operators left-associatively, except for exponentiation, which is right-associative". In implementations of BASIC used by the author the power operator was left-associative, and that of FORTRAN — right-associative. In hand calculators it is left associative, perhaps due to hardware and software limitations.

In the presented implementation the power operator in the input language is nonassociative, e.g. the formula a^b^c is not grammatically correct and must be supplemented with parenthesis — as in previous examples. However the software for symbolic differentiation is more general, it assumes the right — associativity of the power operator and has no limit as to the number of consecutive powers. All other operators are left — associative.

The lacking part of syntax definition can have two forms:

Syntax S4

```
factor = constant | variable | "(" expression ")" |
         standard_function  factor.
```

Syntax S5

```
factor = constant | variable | "(" expression ")" |
         standard_function "(" expression ")".
```

The difference between S4 and S5 is in the allowed forms of arguments of standard functions. Syntax S5 is conservative; each argument must be put into parenthesis, e.g.

```
SIN(x),  ARCTAN(25.6),  EXP(SIN(LOG(y))).
```

Syntax S4 is more flexible, it allows for simplified forms for "simple" arguments but includes also the forms from S5. Thus in S4 the following formulas are equivalent:

```
SIN x,  SIN(x),
COS 55, COS(55),
EXP SIN LOG y,  EXP(SIN(LOG(y))).
```

Arguments which are not factors must of course be put into parenthesis e.g.

```
SIN(a+b),   LOG(2*x).
```

A search for inspiration gives as usual no results. Handbooks of mathematics do not use parenthesis, majority of computer languages do use but some of them do not. Some BASIC dialects even differ between themselves in this point.

Of course any syntax convention will do, as long as formula evaluation and differentiation are consistent and the user is aware of its properties. The present implementation uses the syntax S1+S5 although the software for symbolic differentiation can use the S4 syntax.

To increase the expressing power of the spreadsheet, the syntax of formulae is extended and the new notion of a *conditional_expression* is introduced:

Syntax S6

```
conditional_expression = expression | "IF" logical_expression
                         "THEN" expression { "ELSIF" logical_expression
                         "THEN" expression } "ELSE" expression
logical_expression = logical_term { ["OR" | "XOR"] logical_term }
logical_term = logical_factor { "AND" logical_factor }
logical_factor = relation | "(" logical_expression ")" |
                 "NOT" logical_factor
relation = expression ["=" | "<>" | "<" | ">" | "<=" | ">="] expression |
           expression "IN" [ "[" | "(" ] expression ","
           expression [ "]" | ")" ]
```

This syntax allows for a formula such as

```
IF x<0 AND SIN(x-y) IN [0.7,y/2) THEN x^2 ELSE 44
```

The next change, although formally minor, has great practical consequences.

Syntax S7

```
factor = constant | variable | "(" conditional_expression ")" |
         standard_function "(" conditional_expression ")"
```

This modification allows e.g. for a formula

```
x^(IF x<>y THEN x ELSE 2*y)
```

The present implementation uses the syntax S1+S6+S7. "Classical" standard functions are augmented by the `MAX`, `MIN` functions. In the input language they can have two arguments, while the software for symbolic differentiation can handle any number of arguments (in practice it is limited by a suitably large constant — say 9 — which can be of any value).

2.3 Symbolic differentiation of formulae

2.3.1 General remarks

Programmers recognized very early that computers can work not only with numbers themselves but also with more abstract symbols. Computer programs for symbolic manipulation have been appearing since late fifties. However the popularity of symbolic computing is orders of magnitude less then that of numeric computing. Many people are even not aware of their existence. There are numerous reasons of this situation. Restricted availability — at first at some university centers and only some types of computers and/or operating systems, high cost — large computers, high demand of operating memory and usually long (and unpredictable) computation time. The situation has changed at first with the introduction of

computer networks and then with the era of microcomputers. However, a professional system for symbolic computation needs a powerful workstation with the operating memory of some Megabytes. In the IBM-PC class there is a MUMATH system by Microsoft. (It was the only available system when the contract have been started. The situation has changed by now but conclusions remain valid). The use of it in this work was excluded for two reasons:

- license problem,
- interface with the rest of program would be possible only at the operating system level via exchange of files.

The automatic differentiation can be understood in two ways. The "classical" symbolic systems produce a formula of a derivative. Another, somewhat simpler systems produce a value of a derivative. Examples of latter systems can be found e.g. in series of papers by Rall; the book by Rall (1981) is the good and most detailed representative.

The classical scope of differentiation is not sufficiently wide in the presented application. In optimization it is customary to use some potentially nondifferentiable functions as e.g. `MAX` and `MIN`. The problem of their differentiation is resolved by calculating an element of their subdifferentials. This element is chosen in the following way:

i) the program calculates derivatives of each argument,

ii) the function `MAX` is replaced by `SMX` (SelectMaXimum) and `MIN` is replaced by `SMN` (SelectMiNimum). Derivatives of the arguments of `MAX`, `MIN` are the arguments of the `SMX` and `SMN` functions,

iii) semantics of the `SMX` and `SMN` functions is defined as follows. Evaluation of a formula and its derivatives starts from the `MAX`, `MIN` function and selects some argument. Arguments with the same ordering number are selected in the `SMX` and `SMN` functions.

The process of symbolic differentiation can be outlined as follows. An input formula is converted to its internal representation — a binary tree. This structure is actually differentiated, i.e. another binary tree is created which represents the derivative. This tree is simplified and finally converted into the form of a formula.

A program for any symbolic computations resembles strongly the compiler of a programming language. Therefore some knowledge of elements of compiler-writing problems would be of great help in understanding the following. An excellent source of information is (Wirth, 1976), Chap. 5.

2.3.2 Parsing

It is customary to partition the compilation process into a series of subprocesses called phases. At first the input language must be specified. In our case it is very simple, it consists only of formulae and is defined by means of syntax MBNF clauses or syntax diagrams. The first phase of compilation, called the ***lexical analysis***, separates characters of the source language (the formula) into groups that logically belong together. They are called ***symbols*** or ***tokens***, they are e.g. identifiers of variables, identifiers of standard functions, symbols of operators and punctuation symbols. The output of the lexical analyzer is a stream of symbols, which is passed to the next phase, to the ***syntax analyzer*** or ***parser***.

The parser checks whether the symbols appearing at its input form a legal sequence of the input language (defined by its syntax rules). Besides, it does some other functions. In

the model generator there are four parsers differing by these functions. During edition of a formula parser detects errors (if any) and interacts with the user on their correction. During "compilation" of a formula it produces code for a stack machine. During "numerical" differentiation it produces a code for calculation of derivatives. During "symbolic" differentiation it produces a binary tree — an internal representation of a formula. Of course only the "front-end" parser does the checking. The later ones assume the correctness of their input and "concentrate" on their particular job, their structure remains however the same.

Many parsing methods have been implemented in compilers and/or published. In this work a *recursive-descent* parsing was implemented (*top-down parsing without backtracking*). It is easy to implement by hand and enables to express the generation of output directly in terms of the syntactic structure of the source language. Presentation of syntax in the form of a diagram gives immediately the block scheme of the parser. Each occurrence of a terminal symbol corresponds to the instruction, which recognizes this symbol and reads the next symbol from the input. Each occurrence of a nonterminal symbol corresponds to the call of a procedure, whose structure is given by its own diagram.

This phase is present in both differentiation packages, but the later are different. In the "numerical" version no special internal representation of a formula is needed and a stack structure is sufficient. The parser recognizes parts of expressions, suspected of being common subexpressions, and after code for their evaluation is created, pushes their values on the stack for subsequent use. The following description is concerned with the "symbolic" variant which is more exotic to an average computer user.

2.3.3 Internal representation of formulae

The majority of languages, in which systems for symbolic computation are implemented, are of list-processing type, i.e. a *list* is their primal informational structure. An external representation of a list can be e.g. `(A,B,C,D,E)`, while the internal representation consists of chained *nodes*. Each node contains some information (in the above example — a letter character) or the pointer (an address) to "own" information, and the pointer to the next element of the list. There is a pointer constant, usually called `NIL`, which means "pointer to nowhere" and is used to denote the end of a list. The simplest "units" of information stored in a node (or pointed to from a node) are called *atoms*. Usually they are numbers, identifiers and pointers. The simplest and most commonly used list representation of formulae corresponds to their prefix form, e.g. `x+y` becomes `(+,x,y)`. Each element of a list can be a list themself, e.g. the expression `x+2*y+3` can correspond to the list `(+,x,(*,2,y),3)`.

Thus a tree of large complexity can be created. In PASCAL (which is the implementation language of the program) nodes will be represented by *records* and connections between them by *pointers*. There are no recognized rules for choosing a particular representation of a node; it depends on habits, experience and preferences of the implementer. Many decisions must be taken in connection with the usual trade-off between the range of used memory and the speed of execution. E.g. some redundant information can be stored in nodes thus being immediately available (but more memory is used for data). In another variant, it can be extracted from the tree structures when needed — thus more memory is used for program code for additional procedures, and the execution is slower.

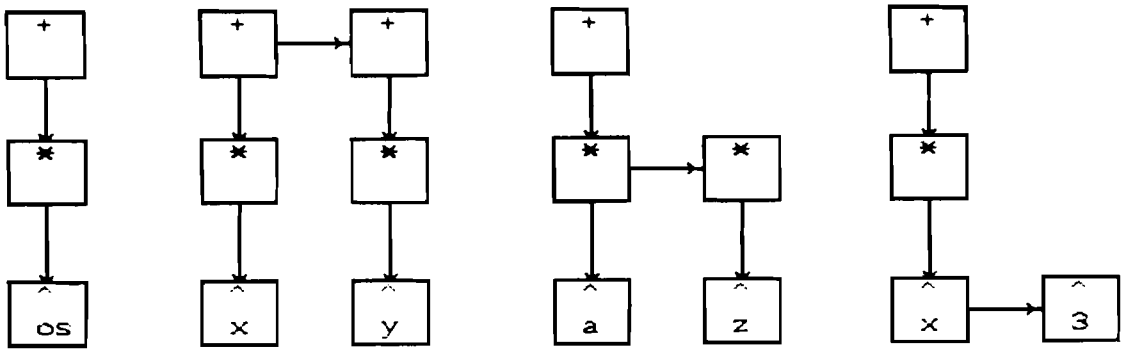
In the presented implementation, there are two pointers in each node. It means that the structure formed with the use of them is a *binary tree*. The usual form of visualization of such structure is with the *root* being most upper and leaves dangling down. In the present context it is, however, more convenient to imagine, that the binary tree is turned counter-clockwise

so that the right branches are horizontal and the left — vertical. To limit the number of procedures needed for operations on structures, the internal representation of structures does not use the binary minus and the division. Instead the addition and multiplication is used with the right arguments multiplied by -1 or taken to power -1, respectively.

The above-mentioned list representation is used in MUMATH symbolic programming system. Its code is known and some comparative remarks can be deduced, although it is written in a specialized MUSIMP programming language and therefore any comparison with a PASCAL implementation must be rather rough. The main advantage of the MUSIMP representation is the memory economy. The price is the relative complexity of arithmetic operations, since a list contains only very condensed syntactic information. E.g. the process of adding two formulae presented by lists `(+,x,y)` and `(*,2,y)` must at first reconstruct their syntactic relations. Arithmetic and algebraic packages contain about 100 functions (not to mention "primitive" MUSIMP functions) and from the programming point of view are the most complicated part of the whole system. Functions are highly recursive and the number of their calls is rather large too. An experiment with tracing (of main algebraic functions only) was performed for two simple expressions `e1=x+y` and `e2=2*x-y`. In the process of their addition there was 18 function calls, in the subtraction — 62 calls. The solution is computed (and simplified) recursively e.g. by adding successive simple expressions. It means that simplification is repeated many times, but when the final result is obtained it is already in a simplified form.

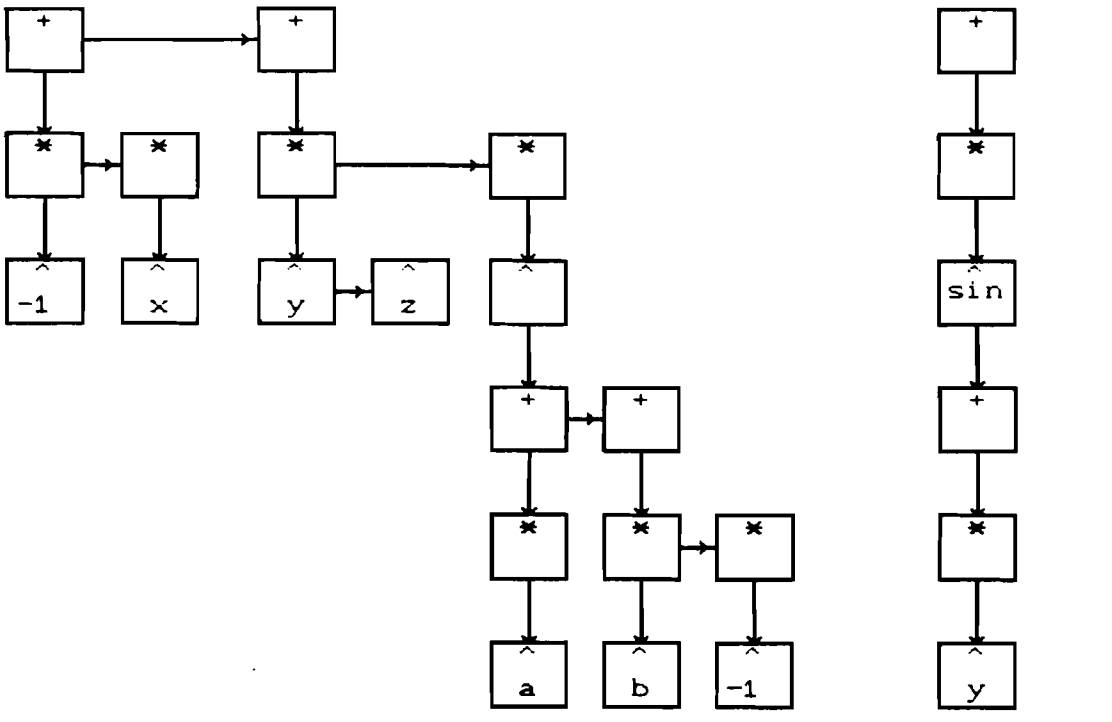
In the presented implementation quite reverse principles were chosen, basing on the following considerations. At a time there exist only two structures, that of a differentiated formula and that of a derivative. After the derivative is compressed into the formula form, the memory used for representation of structures is released. Thus the data storage presents no special limitation. PASCAL is rather talkative for such type of problems, program grows large — so it was thought desirable to limit the number of procedures. The chosen structure expresses the syntactic properties of a formula in an explicit form, at the cost of larger number of nodes. In MUMATH a n-argument sum, product or power expression is represented by a $(n+1)$-element list. In the presented implementation the number of nodes is equal to $3n$, $2n+1$ and $n+2$ respectively (the more complicated is the formula — the better is the comparison). As the result, there are only about 20 "algebraic" procedures. Once the formulae are converted into internal form, their addition is achieved in 1 procedure call. Compression of structures into formulae is divorced from the arithmetic and differentiating operations, i.e. analytical operations are performed on structures and only the final result is transformed into a formula form (and simplified). To improve speed further, the iterative methods are used whenever possible.

Internal representation of formulae is illustrated on figures 1–5. Fundamental structures are presented on Fig. 1. Characters +, *, ^ denote consecutive levels of the graph. Simple-expressions are linked together via nodes at the + level — Fig. 1.b, terms — at the * level — Fig. 1.c, signed-factors — at the ^ level — Fig. 1.d. Structures corresponding to formulae without parentheses are three nodes "deep". The highest level, denoted by + sign, corresponds to simple-expressions. Addition of two formulae corresponds to linking of two structures at this level — e.g. Fig. 1.b. In the expression presented on this figure there are two simple-expressions, each consisting of a simple term, each of them being a single signed-factor, each being a factor (without sign). The middle level, denoted by * sign, corresponds to terms. Linking of structures at that level corresponds to multiplication — e.g. in Fig. 1.c. The low level, denoted by ^ sign, corresponds to signed-factors. Constants and names of variables are contained only in nodes of the ^ level, nodes from levels + and * are used only



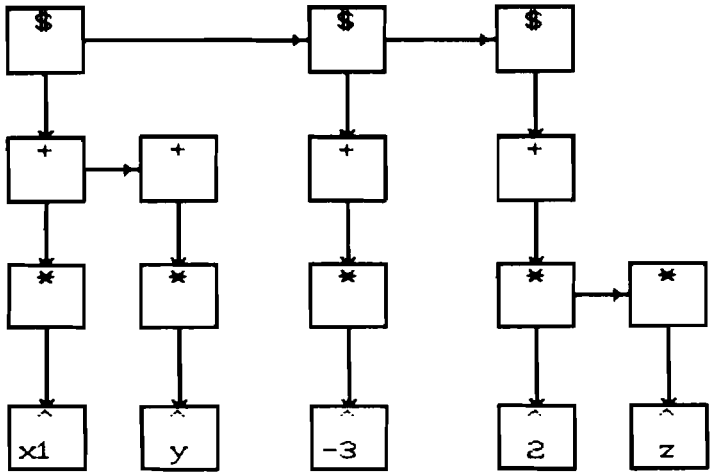
a) os b) x+y c) a*z d) x^3

Figure 1: Elementary structures



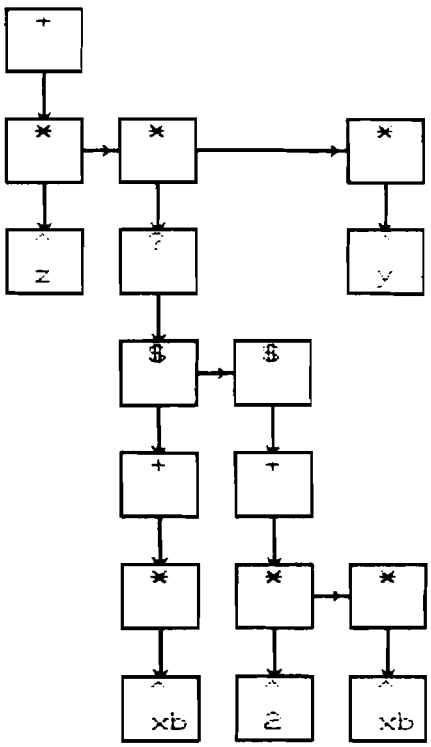
a) -x+y^z*(a-b) b) SIN(y)

Figure 2: More complicated examples



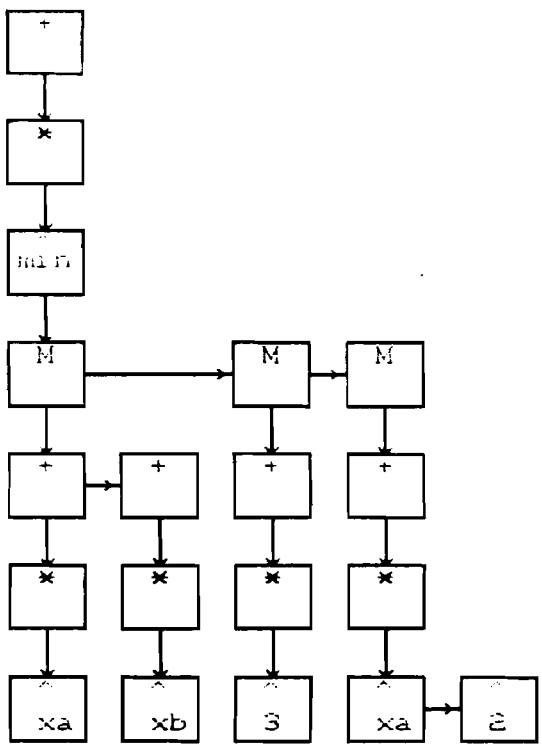

```
IF w1<0 THEN x1+y ELSIF x IN [-2.34, 0.15) THEN -3 ELSE 2*z
```

Figure 3: The structure of a conditional expression




```
z*(IF x<0 THEN xb ELSE 2*xb)*y
```

Figure 4: A conditional expression used as factor




```
MIN(xa+xb,3,xa^2)
```

Figure 5: The structure of a n-argument function

for structuring. The expression in Fig. 1.c consists of one simple-expression, which has two terms. The expression in Fig. 1.d has one term consisting of two signed-factors, each of them being a factor.

In the example presented in Fig. 2.a there are two simple-expressions. The first one consists of two terms. Each of these terms is a single factor. The second simple-expression has two terms. The first one consists of two factors, the second has one factor, which is an expression in parentheses. Each pair of nested parentheses increases the depth of a structure by three nodes. Standard functions are just another form of factors. Their names appear in nodes of the ^ level. Arguments in parentheses are expressed by a lower "layer" of structure, three nodes deep — see Fig. 2.b.

Conditional expressions are presented on Fig. 3. The roots of the consecutive "forks" are denoted by the $ character. Logical-expressions are not subject of differentiation; therefore they are not represented in structures of conditional-expressions. An example of a factor in form of a conditional-expression is presented in Fig. 4. Its appearance is marked by the ? character.

The structure of a function, presented in Fig. 2.b, is valid for one-argument functions only. The MAX and MIN functions, which can have arbitrary number of arguments, use the special structure presented in Fig. 5. The structures of consecutive arguments are marked with the M character.

2.3.4 Arithmetic operations and differentiation of structures

All mathematical operations are performed on structures — in fact on copies of their arguments. The addition is the simplest, the corresponding structures are linked together at the + level (see the structure on Fig. 1.b). This structure can be treated as the result of "addition" of the two structures corresponding to expressions x and y. In the multiplication procedure a skeleton structure is produced as in Fig. 1.c, but without the variables a, z in the ^ nodes. The copies of arguments are linked under those nodes, instead. The power procedure acts similarly using the structure presented in Fig. 1.d. Subtraction and division appear only in the "external" formula representation and never — in the "internal" structural form, so they need not be implemented. Operations of standard functions are performed according to patterns presented in Fig. 2.b and Fig. 5.

Differentiating procedures take structures as their input and produce structures representing derivatives, basing on rules of differentiation and using the library of mathematical operations. The derivative of an expression is formed as the sum of derivatives of simple-expressions. A simple-expression has the form of

$$t_1 * t_2 * \ldots * t_n ,$$

where t_i denote a term. Its derivative if formed as the sum of all products of the form

$$t_1 * \ldots * dt_i/dx * \ldots * t_n .$$

Differentiation of these both syntactic constructs can (and is) be implemented iteratively. A term has the general form

$$f_1 \hat{} f_2 \hat{} \ldots \hat{} f_n ,$$

where f_i denote signed-factors. The derivative of a term is computed according to the formula

$$d(f_1 \hat{} R)/dx = f_1 \hat{} R * \ln f_1 * dR/dx + R * f_1 \hat{} (R - 1) * df_1/dx ,$$

where f_1 denote the first factor and R — the rest of a term. It is a recursive definition and must be implemented recursively. The derivative is produced by creating all necessary factors and linking them into two simple-expressions and finally into one expression. The derivative of a signed-factor (or a factor) is computing according to its form. A derivative of a constant or a variable is the structure corresponding to the constant 0 or 1. When a factor has the form of an expression enclosed in parentheses, its derivative is computed by the recursive call of the expression differentiating procedure. When a factor is a standard function, its derivative is computed by creating a structure corresponding to the derivative of this function and multiplying it by the derivative of its argument.

2.3.5 Simplification and compression of structures

Differentiating procedures give the result in the form of a structure, which is highly redundant, e.g. the derivative of `2*x` is computed as `0*x+2*1`, so the compression must be accompanied by the simplification.

In this implementation the general rules of differentiation, presented in the preceding section, are somewhat modified. This spoils slightly their conciseness and mathematical elegance but reduces much the redundancy of produced derivatives. Nevertheless, simplification of structures remains the most complicated part of the whole package.

The process of simplification presents real problems in any system for symbolic computations. This is due to the large number of interconnected simplifying rules, used by humans, and to the nonexistence of the "simplest" canonical form of a formula. In large systems some forms are used parallel and then one of the results is selected. Such approach was naturally out of question in the described program. E.g. several strategies can be adopted for extracting common terms from simple expressions. The expression:

i) `c*a*b*d + a*b + a`

can be simplified to:

ii) `a*(b*c*d + b + 1)`,

iii) `a*b*(c*d + 1) + a`,

iv) `a*(b*(c*d + 1) + 1)`.

In the first case only the terms which are common to all simple expressions are extracted. In the second case the extraction of a common term affects only those simple expressions, which yield the most strongly modified result. The third case is a recursive version of the first one. Which strategy is the best? One can easily find examples in favor of each of them and counterexamples against each of them.

Similar problems arise when dealing with exponents. The following transformation:

v) $x^5 + x^2 \longrightarrow x^2 * (x^3 + 1)$

seems reasonable, but neither of the following two:

vi) $x^5 + x^{-2} \longrightarrow x^5 * (1 + x^{-7})$,

vii) $x^5 + x^{-2} \longrightarrow x^{-2} * (x^7 + 1)$

seems not. Similarly the following transformation

viii) $x^y + x^2 \longrightarrow x^y * (1 + x^{2-y})$

seems unreasonable too. But counterexamples against those considerations can be easily constructed too.

In general purpose systems there are control parameters, which can modify the default strategy of simplification. An experienced user can tune the system in order to obtain the most satisfactory result for any particular expression. Such approach was out of question in this program and some permanent strategy had to be chosen, e.g. transformations illustrated by ii) and v) were chosen. The authors would welcome any comments about this approach and suggestions about eventual changes.

Technically, the simplification is done as follows. There is a library of procedures in the system, which recognize trees with specific structural properties and transform them into another trees representing "simpler" structures. E.g. exponential expressions are served by the following transformations:

- power of powers
$$(x^a)^b \longrightarrow x^{ab},$$
- distribution of the exponent on the base
$$(xy)^a \longrightarrow x^a y^a,$$
- distribution of the base on the exponent
$$x^{a+b} \longrightarrow x^a x^b,$$
- factorization of the exponent from the base
$$x^a y^a \longrightarrow (xy)^a,$$
- factorization of the base from the exponent
$$x^a x^b \longrightarrow x^{a+b}.$$

The simplifying procedures interact with each other in iterative and recursive way. The pattern of interaction is an experimentally tuned compromise between the resulting form of a formula and the simplification time. This point is illustrated by the following example, comparing the result obtained with model generator with the results obtained with the use of two professional symbolic systems.

The calculation of
$$\frac{d}{dx} x^{x^x}$$
yields the following results (rewritten in mathematical notation):

- in this program
$$(x^x \ln x + x^x) x^{x^x} \ln x + x^{(-1+x+x^x)},$$
- in the MUMATH system
$$x^{(x+x^x)} (\ln x)^2 + x^{(x+x^x)} \ln x + x^{(-1+x+x^x)},$$

- in the MACSYMA system

$$x^{x^x}(x^x \ln x(\ln x + 1) + x^{(x-1)}).$$

Every simplification rule, necessary to transform our result to the MACSYMA's form is inherent in the system. However, to obtain this form one more simplification run would be necessary — it would improve the form of this particular formula but in other cases it could be only a waste of computing time. Finally, the choice of the "nicest" result is somehow a matter of taste.

The scope of simplification is not as wide as in general purpose symbolic systems, since the symbolic part forms here only a fragment of the whole program. Most care was devoted to treating expressions with power operators while e.g. fractional and trigonometric simplifications are not implemented at all. Besides, any numerical calculations which sometimes appear (e.g. when calculating numeric coefficients in symbolic expressions) are calculated in the floating — point domain. In general packages numbers are represented in the rational domain and numerical calculations are performed "symbolic" too with the absolute accuracy.

The preceding discussion get ahead of the last step in the calculation of a derivative — the compression of a binary tree into a formula. It is implemented as a recursive procedure. The structure is searched in *post-order*. For a given node there is a formula or a value corresponding to the (part of original) structure pointed to by the "low" pointer of this node and a formula or a value corresponding to the structure pointed to by the "right" pointer. These two arguments are combined together into a formula or a value, according to the "contents" of the node, i.e. they are added, multiplied, exponentiated, the sign is changed or a standard function is applied. The process of conversion is accompanied by additional simplification, in fact quite powerful but "local" in scope, i.e. only interactions between the contents of a given node, its "lower" subexpression and its "right" subexpression play their role here.

2.4 Evaluation of formulae

Cells of the spreadsheet are evaluated from the right to the left and from the top to the bottom. A stack machine is defined and formulae are compiled into the code for that virtual computer. This code is stored in the spreadsheet and is used for the evaluation of formulae. All run-time errors are handled by the program itself. In the spreadsheet there is no mechanism for deciding whether any of the formulae was changed. Such device would be very costly since it had to remember the old and the new versions, perform syntactic and semantic analysis of both and compare results. To keep the spreadsheet consistent it is simply recalculated in situations, when the user had chance to modify any of the formulae.

2.5 Symbolic differentiation in the spreadsheet

The above considerations apply to symbolic differentiation too. In the spreadsheet only partial derivatives are calculated, and they are calculated each time anew. There are no limits (except available memory) on the length of input formulae and resulting derivatives. If there is no sufficient amount of memory, the symbolic calculations are stopped and the previous state of the spreadsheet is reconstructed.

3 Short user manual

The described variant of the nonlinear model generators — IAC-DIDAS-G is an extension of the IAC-DIDAS-N, equipped additionally with a symbolic differentiation package. The user manual for the DIDAS-N remains valid, the differences are minor. The installation procedure for both systems is identical.

The symbolic differentiation is an additional option in the model editing phase. To make room on the screen the LIST command (i.e. invoked by pressing < Alt L >) was deleted from the upper menu line. It is however not a drawback since previously the List could be invoked both by pressing < Alt L > and pressing the function key < F4 > — and this later option remains. In the gained place another menu item was put: ***sYmbolic*** (i.e. invoked by the < Alt Y > key). The formulae of derivatives can be viewed on the screen or printed. In the latter case the user can chose between derivatives of one outcome or of all outcomes in the model.

The following description lists the additional menu entries, presented according to the style of the DIDAS-N user manual.

< Alt Y >	sYmbolic:	— switches the symbolic mode on,
	Differentiate.	— asks for the selection of an outcome to be differentiated. The results can only be viewed on the screen.
	diff & Print:	— results will be printed. This command invokes the Goal submenu.
	Goal:	— selects the goal for printing:
	One outcome.	— asks for the selection of an outcome to be differentiated and printed,
	All outcomes.	— derivatives of all outcomes are printed.

Context sensitive help information can be obtained by pressing the < F1 > key. Exit to the operating system — by pressing the < F10 > key.

The model generator has the form of a specialized spreadsheet. Rows represent outcomes (dependent variables), whereas columns represent inputs, parameters (independent variables) and outcomes. A model is defined as mathematical formulae for outcomes in the "Solution" column.

Formulae of partial derivatives are calculated in cells to the right of a "Solution" cell, with respect to the variables which label the corresponding columns of the spreadsheet. They can be viewed the same way as outcome formulae.

User names can be defined in suitable spreadsheet cells. Derivatives can be referred to in the form e.g. `{Ya/Xb}`. Such names can appear only in logic expressions (see later explanation of syntax). Letters in upper and lower cases are not distinguished. Formulae must be explicit, i.e. only names of previously defined outcomes can be used in an outcome formula (this is checked by the formula editor).

After placing the cell cursor in a cell, it can be "entered" by pressing the < Enter > key. The blinking one-character cursor appears and the cell can be edited. To end the edition press the < Enter > key again.

3.1 Syntax of formulae

The most general form of a formula is called a *conditional expression*. For the ease of presentation it will be temporarily simplified to a "conventional" expression. The smallest syntax unit of an expression is a *factor*. It can be:

- a variable name, e.g. `Xb`,
- a numerical constant, e.g. `3`, `1.234`,
- PI (a predeclared constant),
- a derivative name, e.g. `{Yc/Xa}`.

Further, it can be an expression enclosed in parenthesis or a standard function of an expression enclosed in parenthesis:

```
( <expression> ),
<standard_function> ( <expression> ) .
```

Expressions (of "conventional" form) are formed from factors with the use of operators `+`, `-` (unary and binary), `*`, `/`, `^`. The unary minus operator has higher precedence than power operator. The power operator is non-associative while all other binary operators are left-associative.

Standard functions include: one parameter functions `ABS`, `ARCTAN`, `COS`, `EXP`, `LN`, `LOG`, `SIGNUM`, `SIN`, `SQR` (square), `SQRT` (square root); and two parameters functions `MAX`, `MIN`.

More complicated syntactic construct is a *conditional expression*. It can be:

- a "usual" expression,
- `IF <logical expression> THEN <expression> ELSE <expression>` ,
- ```
  IF <logical expression> THEN <expression>
  ELSIF <logical expression> THEN <expression>
  ELSIF ...  ELSE <expression> .
  ```

There may be up to nine `ELSIF ... THEN ...` clauses. The smallest syntax unit of a *logical expression* is a *relation*. It can have the following forms:

```
<expression> <relational operator> <expression>
```

where the *relational operators* are `=`, `<>`, `<`, `>`, `<=`, `>=`,

```
<expression> IN [ <expression> , <expression> ) .
```

In this relation of belonging to a given interval, the interval can have open and closed ends in any combination (denoted by `(`, `)`, and `[`, `]` correspondingly).
Examples of relations:

```
xa-xb >= SIN(ya) ,
EXP(xa) IN (1,SQRT(2+xb)] .
```

The next higher structure is a *logical factor*, which can be:

- `<relation>` ,
```

- (<logical expression>) ,
- NOT <logical factor> .

Logical expressions are formed from logical factors using AND, OR, XOR operators, e.g.

```
xa>0 OR SIN(xb) IN [0.3, xa/2] ,
NOT ( xa<0 AND xb IN (1,3) ) .
```

Now, we can present other extensions of syntax beyond its initial, simplified form:
A *factor* can be:

```
( <conditional expression> ) ,
<standard function> ( <conditional expression> ) .
```

The other forms of a factor remain the same. Thus e.g. the following expression is legal:

```
(IF xa-xb/4 IN (EXP(zb),-1+SQRT(xa/2)) THEN 5 ELSE xa^2) +
  xa*(IF xb<0 THEN SIN(xb) ELSE MAX(xa/2,xa^2-xb,5))^3 .
```

3.2 Differentiation

The spreadsheet calculates and displays analytical formulae of partial derivatives. To give more flexibility in model formulation some nondifferentiable functions are included. Their derivatives calculated and displayed in the spreadsheet are arbitrarily selected elements of their subdifferentials and thus the user is responsible for their correct interpretation. Note:

- the derivative of ABS is taken as SIGNUM,
- the derivative of SIGNUM is taken as 0,
- the derivatives of MAX, MIN are calculated in the following sense:
  - i) derivatives of each argument are calculated,
  - ii) a shorthand notation is used: SMX(..., derivative ,...) – SelectMaXimum or SMN — SelectMiNimum.
  - iii) during calculation some argument is selected in the original expression. Arguments with the same order number are selected in derivatives.

The derivatives of conditional expressions are calculated in the following sense:

- only "true" expressions, i.e. not embedded in any IF (ELSIF) ... THEN brackets are differentiated,
- conditions are copied from the input formula,
- a derivative expression for evaluating is selected according to actual values of conditional expressions.

4 Illustrative examples

The Model Generator is an extension of the DIDAS-N. To avoid repetitions examples presented here illustrate only the problems not highlighted in the documentation of the latter, i.e. they are connected with symbolic differentiation. In the following the names x1, x2 denote variables and the names y1, y2 denote outcomes.

Example 1.

The simplification is performed only when considered necessary. E.g. it is not applied to input formulae (i.e. before differentiating). When calculating the derivative of

```
y1 = x1*x2/x1
```

with respect to x1, the dependence of y1 on x1 is assumed and the following result is *calculated*:

```
{y1/x1} = 0.
```

When the equivalent formula

```
y1 = x2
```

is given as input, the program knows the answer (trivial 0) *without differentiating*.

Similarly with the input formula

```
y1 = LN(SQRT(ABS(SQR(EXP(x1)))))
```

the program will laboriously calculate the derivative of a compound function instead of differentiating the equivalent formula

```
y1 = x1.
```

Simplification of the input formulae was thought of as a waste of time, the user of a baroque style will take consequences by himself.

Example 2.

The rational arithmetic is not implemented, thus the derivative of

```
y1 = 2/3*x1*x2
```

is calculated as

```
{y1/x1} = 0.6666667*x2 .
```

Example 3.

A function of a numerical argument is not calculated in order to preserve symbolic information. Thus the derivative of

```
y1 = 5^x2
```

is presented in the form

```
{y1/x2} = 5^x2*LN(5)
```

and not as

```
{y1/x2} = 1.60944*5^x2 .
```

There are obvious exceptions to this rule — when the value of a function equals 0 or 1. This remark does not apply to trigonometric functions — rules for their simplifications are not implemented at all.

Example 4.

A common factor is extracted only when it appears in each simple-expression of an expression. Thus the derivative of

```
y1 = x1*x2*(2 + x1 + x2)
```

is presented as

```
{y1/x1} = x2*(2 + x1*x2 + x1*x2)
```

i.e. common factors x1 and x2 are not extracted since they do not appear in the first simple-expression of the expression in parenthesis. Another situation arises while differentiating the formula

```
y2 = x1*x2*(SIN(x1*x2) + SIN(x1*x2)) .
```

Its derivative is presented as

```
{y2/x2} = 2*x1*(SIN(x1*x2) + x2*x1*COS(x1*x2)) .
```

In this case common factors are extracted and the result is equal to that of differentiating of simplified input formula:

```
y2 = 2*x1*x2*SIN(x1*x2) .
```

Example 5.

The speed of symbolic computations was one of the main goals in the described implementation. Therefore only a number of simplification actions, in a prescribed order and with limited recurrence is applied to an internal representation of a derivative. In the consequence the derivatives of different but algebraically equivalent formulae can sometimes appear in different (algebraically equivalent too) forms. E.g. the derivative of

```
y1 = 2^(x1^2)
```

is equal to

```
{y1/x1} = 2^(1 + x1^2)*x1*LN(2) ,
```

while the derivative of an equivalent formula

```
y2 = (2^x1)^x1
```

is presented in the form

```
{y2/x1} = x1*2*2^(x1^2)*LN(2) .
```

Another possible strategy is to continue simplifying of a formula until it cannot be simplified any further. E.g. the {y2/x1} would be transformed to the form of {y1/x1} in one more recursive call of the simplification procedure.

This example illustrates another point of interest. For technical reasons it is necessary to introduce some canonical form of a formula. E.g. in a product the numerical coefficients should come on the first place, next there should be variables, next ...and functions at the end. Transformations of internal structures into canonical forms is necessary before any comparison of structures and only then. However it could be applied "on exit" of the simplification process for aesthetical reasons. In the system it is not applied there — just for the speed of calculations. Effects of such approach are visible on results presented in

Examples 4 and 5. The design principles illustrated by this example are perhaps the most controversial. The author would appreciate any comments and suggestions concerning them.

Now comes the problem of using this additional option. No explicit examples are given as they would be naive to every user but a student just learning fundamentals of mathematical analysis. The view of formulae of derivatives can be an auxiliary tool in the process of model formulation, can increase the analytical insight into properties of the model and improve the understanding of nonlinear phenomena. The symbolic differentiation package can do nothing more than a skilled person can do, but it will save the user time and protect him from analytical errors.

5 Hardware requirements

The hardware requirements for the Nonlinear Model Generator are identical to those for the IAC-DIDAS-N program. It runs on a microcomputer compatible with IBM-XT or AT (with Hercules Graphic Card, Color Graphics Adapter or Enhanced Graphics Adapter and, preferably, with a numerical coprocessor and a hard disk) and requires 512 kilobytes of memory. However, an extra amount of memory is needed for symbolic computations. This memory is allocated on entry to the symbolic processing phase and deallocated on exit. The size of this extra memory depends on the level of complication of a formula, the value of about 30 kilobytes seems to be rather safe upper bound. Besides, the program is compiled with an enlarged stack since the procedures, which transform a structure into a formula, are highly recursive. When there is not enough memory, symbolic computations are abandoned or even not started at all and the corresponding message is issued.

The installation procedure of IAC-DIDAS-G is identical as for the IAC-DIDAS-N.

References

Aho A.V. and D. Ullman (1977). Principles of Compiler Design. Addison Wesley.

Fichtenholz G.M. (1966). Handbook of Differential and Integral Calculus. Nauka. (in russian).

Kaden S. and M. Grauer (1984). A Nonlinear Dynamic Interactive Decision Analysis and Support System (DIDAS-N). User's Guide. WP-84-23. IIASA.

Kreglewski T. and S. Kaden (1985). Decision Support System MINE. Problem Solver for Nonlinear Multicriteria Analysis. IIASA.

Lewandowski A., T. Kreglewski and T. Rogowski (1985). DIDAS-NL — the Nonlinear Version of DIDAS System. In A. Lewandowski and A. Wierzbicki, eds., Theory, Software and Test Examples for Decision Support Systems. IIASA.

Lewandowski A. (1985). Problem Interface for Nonlinear DIDAS. Draft.

Pavelle R. (1985). MACSYMA: Capabilities and Applications to Problems in Engineering and the Science. In B. Buchberger, ed., EUROCAL'85, European Conference on Computer Algebra. Lecture Notes in Computer Science, Vol. 203. Springer.

Pavelle R., M. Rothstein and J. Fitch (1981). Computer Algebra. *Scientific American*, December 1981.

Rall L.B. (1981). Automatic Differentiation: Techniques and Applications. Lecture Notes in Computer Science, Vol. 120. Springer.

Wirth N. (1976). Algorithms + Data Structures = Programs. Prentice Hall.

Wolfe P. (1982). Checking the Calculation of Gradients. *ACM Trans. Math. Software*. Vol. 8, December 1982.