DECISION SUPPORT SYSTEM MINE PROBLEM SOLVER FOR NONLINEAR MULTI-CRITERIA ANALYSIS

S. Kaden
T. Kreglewski

January 1986
CP-86-5

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
2361 Laxenburg, Austria

PREFACE

The *Regional Water Policies* project of IIASA was focused on intensively developed regions where the water resources are integrating elements of the environment. The research was directed towards the development of methods and models to support the resolution of conflicts within such socio-economic environmental systems. One of our case studies deals with open-pit lignite mining areas. The developed Decision Support System MINE has been implemented for a test region in the Lusatian Lignite District of the GDR.

The complex problems of such regional policy analysis are not tractable in one model using any of existing computational methods. That is why a heuristic two-level model approach has been applied. Simplified first-level models together with interactive procedures for multi-criteria analysis are used in the *Planning Model* for screening analysis of rational long-term policies. Second-level models serve for the verification and specification of the results of screening analysis.

In developing the system our major goal was to make it user-friendly, highly interactive and robust. For the planning model these features are determined above all by the effectivity of the problem solver for multi-criteria analysis. The given paper describes such a problem solver being developed for the DSS MINE. This research has been done within the framework of a collaborative agreement between IIASA and the Technical University of Warsaw, Institute of Automatic Control.

Sergei Orlovski
Project Leader
Regional Water Policies Project

ABSTRACT

The Decision Support System MINE has been developed for the analysis of regional water policies in open-pit lignite mining areas. It is based on a two-level model approach. The first-level *planning model* is used for the estimation of rational strategies of long-term development applying dynamic multi-criteria analysis. The second-level *management model* considers managerial/ operational aspects for shorter time steps (monthly and yearly).

The paper describes the problem solver for multi-criteria analysis in the planning model. This analysis is based on the *reference point approach*. For the solution of the resulting nonlinear programming problem the MSPN-algorithm, developed at the Institute of Automatic Control of the Technical University has been adopted. The solver considers the special characteristics of the mathematical model of the DSS MINE, as its non-linearity and the sparse character of the resulting Jacobian matrix.

Starting with the description of the general mathematical structure of the planning model within the DSS MINE the problem formulation for multi-criteria analysis based on the *Reference Point Approach* is given. Next, the non-linear problem solver MSPN is presented, including a program description. Finally the results of some computational tests are shown.

CONTENTS

DECISION SUPPORT SYSTEM MINE
PROBLEM SOLVER FOR NONLINEAR MULTI-CRITERIA ANALYSIS

S. Kaden[1] and T. Kreglewski[2]

1. Introduction

Regions with open-pit lignite mining are characterized by complex and strong interactions in the socio-economic environmental system with special regard to water resources. Caused by lignite mining, above all the necessary mine drainage, originate significant conflicts between different interest groups. For a detailed description of those problems see Kaden et al., 1985a.

Due to the complexity of the socio-economic environmental processes in mining areas, the design of regional water policies and water use technologies as well as mine drainage can only be done properly based on appropriate mathematical models. From a critical analysis of the state-of-the-art of modeling in lignite mining areas it has been concluded, that above all methods and models are required to support the analysis and implementation of *rational long-term regional water policies* in open-pit lignite mining areas, to achieve a proper balance between economic welfare and the state of the environment, Kaden et al. 1985b.

Towards that goal the research of the Regional Water Policies project of IIASA, in collaboration with research institutes in the GDR, and in Poland, in the period 1984-1985 was directed. One of its major products is the **Decision Support System MINE**, see Kaden et al. 1985a, Kaden 1986. The DSS MINE has been implemented for a test region in the Lusatian Lignite District in the GDR.

The analysis of regional water policies in mining regions is a problem of dynamic multi-criteria choice. An advanced system of decision aids is needed which allows, Kaden et al. 1986:

[1] International Institute for Applied Systems Analysis Laxenburg, Austria
[2] Institute of Automatic Control, Technical University of Warsaw, Poland

- to consider the controversy among different water users and interest groups,
- to include multiple criteria some of which can not be evaluated quantitatively,
- to take into the account the uncertainty and the stochastic character of the system inputs as well as the limited possibilities to analyze all the decisive natural and socio-economic processes and impacts,
- to offer a set of decision alternatives, demonstrating the necessary trade-offs between different water users and interest groups.

At present no mathematical methods are available or practical applicable considering all these problems in one single model. Only time-discrete hierarchical model systems can satisfy all requirements. Frequently already a two-level model hierarchy satisfies most requirements. For the DSS MINE such a two-level system has been realized.

The first-level model is a **Planning Model** for the dynamic multi-criteria analysis for a relatively small number of *planning periods* , $j=1,...,J$ as the time step for principal management/technological decisions. Variable time steps are used starting with one year and increasing with time up to 15 years.

The planning model serves for the estimation of rational strategies of long-term systems development. These strategies are selected by multi-criteria analysis.

The second-level **Management Model** is applied for the simulation of systems behavior for a larger number of smaller *management periods* (monthly and yearly time steps). It is used to analyze managerial decisions by the help of stochastic simulation and to verify results obtained with the planning model.

The DSS MINE is intended to be highly interactively, user-friendly and robust. The realization of these goals depends above all on the effectivity of the basic mathematical methods and models. One of the fundamental algorithms is the algorithm for non-linear multi-criteria analysis in the planning model.

The given paper describes the solver for non-linear multi-criteria analysis of the DSS MINE. It has been developed in collaboration between IIASA and the Institute of Automatic Control of the Technical University Warsaw, Poland.

2. Multi-Criteria Analysis

2.1. Structure of the planning model of the DSS MINE

The planning model covers a *planning horizon* of 50 years divided into maximum 10 planning periods, see Figure 1.
The figure illustrates that the highest accuracy is achieved for the first planning periods. The later planning periods give rough estimates of future systems development. Their consideration ensures a rational systems development in the long-term run.

The planning model of the DSS MINE serves for the estimation of rational strategies of long-term systems development. These strategies are selected by multi-criteria analysis considering a number of *criteria*. The criteria have to be chosen from a given set of *indicators*, e.g. cost of water supply, cost of mine drainage, satisfaction of water demand and environmental requirements. These indicators are assumed to be integral values over the whole planning horizon. In Figure 2 a block scheme of the planning model is given.

With the purpose of a unified model being independent on the chosen criteria it is assumed that for all indicators bounds are given and all indicators are treated as constraints. Based on that the following multi-criteria problem for a subset O_l $l \in L_0$ of the indicators $\mathbf{O}(O_l , l=1,...,L)$ is defined:

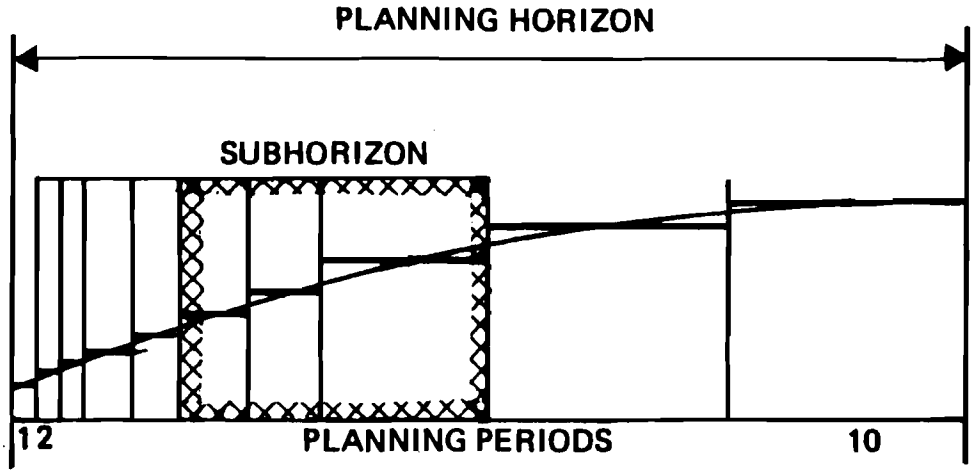

Figure 1: Time discretization for the planning model

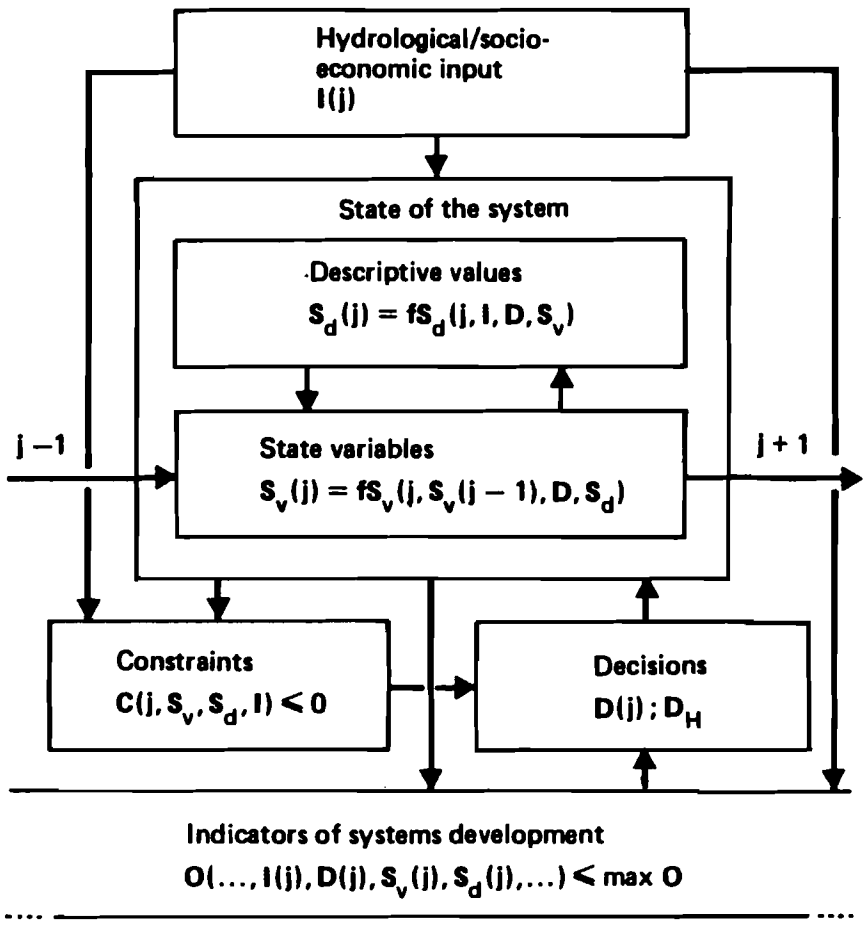

Figure 2: Block schema of the planning model

$$O_l = Minimum\ !\ l \in L_0 \tag{2.1}$$

subject to inequality constraints

$$\mathbf{O} \leq \max \mathbf{O} \tag{2.2}$$

$$\mathbf{C}_{in}(j) \leq 0\ ,\ j=1,\ldots,J$$

equality constraints

$$\mathbf{C}_{eq}(j) = 0\ ,\ j=1,\ldots,J \tag{2.3}$$

$$\mathbf{S}_v(j) - \mathbf{fS}_v(j) = 0\ ,\ j=1,\ldots,J$$

bounds

$$\min \mathbf{D}(j) \leq \mathbf{D}(j) \leq \max \mathbf{D}(j)\ ,\ j=1,\ldots,J \tag{2.4}$$

$$\min \mathbf{D}_H \leq \mathbf{D}_H \leq \max \mathbf{D}_H$$

This model describes a non-linear dynamic multi-criteria problem. For the given problem in mining areas it can be assumed that the systems dynamic is determined above all by the externally fixed mine drainage. The internal systems dynamic is relatively slight, that means the influence of the state variables on the result (indicators) is less important. Consequently the problem Eq.(2.1)-(2.4) may be divided into subproblems for a few *subhorizons* m, $m=1,\ldots,M$, see Figure 1.

$$\mathbf{o}^*(m) => Minimum\ !\ ,\ m=1,\ldots,M \tag{2.5}$$

subject to Eq.(2.2)-(2.4) for subhorizon m. This approach reduces the computational effort due to the smaller dimension of the non-linear programming problem.

In Figure 3 the structure of the Jacobian matrix is depicted for a subhorizon with two planning periods. The numbers give the actual size of the problem for the GDR test area.
The Figure illustrates the sparse character of the matrix. With the increasing number of planning periods per subhorizon the matrix is getting more sparse. The algorithm for non-linear programming has to consider this property in order to reduce storage consumption and computational effort.

2.2. Problem formulation for multi-criteria analysis

Instead of the problem oriented model formulation above for simplicity and convenience in the following a more compact mathematical formulation of the multi-criteria problem Eq.(2.1)-(2.5) is used.

We consider a nonlinear optimization problems of the form:

minimize set of functions

$$f_i(x)\ ,\quad i \in I_0 \tag{2.6}$$

subject to:

nonlinear inequality constraints

$$f_i(x) \leq b_i\ ,\quad i=1,\ldots,n_g \tag{2.7}$$

nonlinear equality constraints

$$f_i(x) = b_i\ ,\quad i=n_g+1,\ldots,n_h \tag{2.8}$$

and bounds for all variables

$$l_j \leq x_j \leq u_j\ ,\quad j=1,\ldots,n \tag{2.9}$$

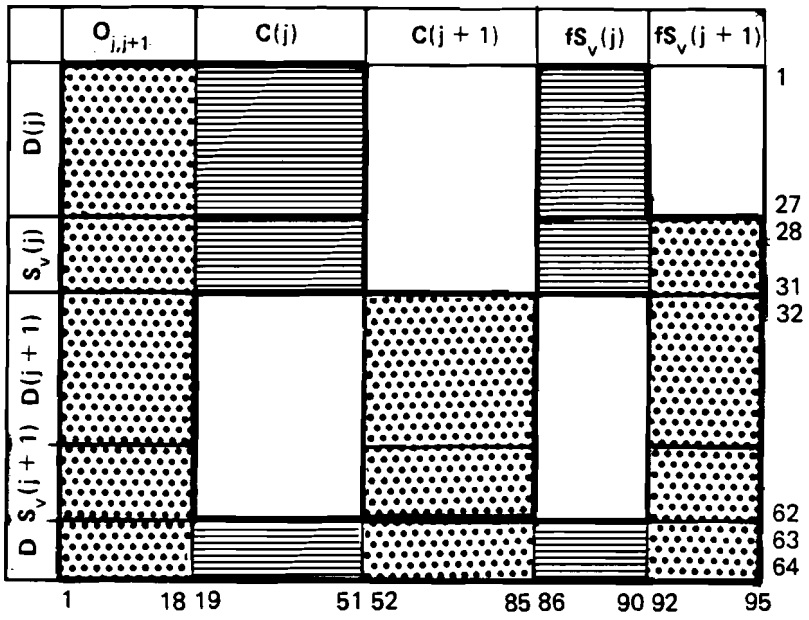

Figure 3: Structure of the Jacobian matrix

The nonlinear functions $f_i(x)$, $i=1,\ldots,n_h$ are assumed to be differentiable and their gradients must be known in the analytical form. These functions, together with the right hand sides b_i , $i=1,\ldots,n_h$, the lower bounds l_j , $j=1,\ldots,n$, and the upper bounds u_j , $j=1,\ldots,n$, constitute the model of systems behavior.

As explained above some of the values $f_i(x)$, $i=1,\ldots,n_g$ calculated in the model are defined as indicators of systems development. The set of indices I_0 contains the numbers of functions $f_i(x)$ selected as objectives, being of interest for the decision maker applying the DSS MINE. This set can be changed at any time.

Most frequently the set I_0 has more than one element and in such a case problem (2.6) - (2.9) is the problem of ***multiobjective optimization***. To solve such a problem, one must *scalarize* it, i.e. reduce it to single criteria equivalent using a *scalarizing function*.

2.3. Scalarizing method - the Reference Point Approach

For the DSS MINE the Reference Point Approach (Wierzbicki, 1983) is applied. In this method the reduction of the multiobjective optimization problem to a single objective one must be interactively defined by the decision maker (the model user). The preferences among several criteria are unknown a priori and are determined during the interactive procedure. For this purpose, the decision maker defines a reference value r_i for each selected objective function $f_i(x)$. These values should reflect in some sense desired values of the objectives.

If all functions of the mathematical model are linear, then the scalarizing function using reference values can be given as follows:

$$s(w) = \max_{i \in I_0} w_i + \varepsilon \sum_{i \in I_0} w_i \qquad (2.10)$$

with

$w_i = f_i(x) - r_i$ - distance between current values $f_i(x)$ and desired (reference) values r_i ;

ε - small positive number.

The second term of the scalarizing function is added to guarantee Pareto optimality of the solution. (If $\varepsilon = 0$ only weak Pareto optimality is guaranteed).

If $f_i(x) \geq r_i$ for all objectives, then the first term of (2.10) is simply a Chebyshev norm of the distance between solution $f(x)$ and reference point r ; in a more general case, this scalarizing function is not necessarily equivalent to a norm, but is strictly monotonous and thus guarantees Pareto optimality of its minimas. Unfortunately, this function is non-differentiable and cannot be used in the case when the mathematical model is non-linear, because non-linear non-differentiable optimization methods are neither robust nor efficient enough for use in interactive systems. In the non-linear case only a smooth approximation of function (2.10) can be used.

The approximation used in this package has the form:

$$s(w) = \left[\frac{1}{n_0} \sum_{i \in I_0} \left[s_i w_i \right]^{\rho} \right]^{1/\rho} \tag{2.11}$$

with

n_0 - number of objectives;

s_i - scaling factor for i-th objective;

$w_i = \dfrac{f_i(x) - \overline{f_i}}{r_i - \overline{f_i}}$ - measure of distance between $f_i(x)$ and r_i ;

$\overline{f_i}$ - lower bound for $f_i(x)$ and r_i ;

ρ - positive (even) integer.

For the lower bound $\overline{f_i}$ the *utopia (ideal) value* may be used minimizing objective i separately.

If ρ is very large, then the approximation of the function (2.10) by (2.11) is good. Unfortunately, the problem of minimizing (2.11) becomes badly conditioned in such a case. Therefore, $\rho = 4$ or 6 is used in this package.

3. Non-Linear Problem Solver MSPN

3.1. Theoretical background

3.1.1. General description of the algorithm

For a given fixed set of indices I_0 , fixed reference point values r_i and scaling factors s_i , the resulting optimization problem takes the form:

$$\min_{x \in X} \left[s_w(x) = = s(w(x)) \right] \tag{3.1}$$

The feasible set X is determined as a intersection of two other sets:

$$X = X_N \cap X_L \tag{3.2}$$

X_N is the set described by nonlinear inequalities (2.7) and equalities (2.8).

$$X_N = \left\{ x \in R^n : f_i(x) \leq b_i , i = 1,\ldots,n_g , f_i(x) = b_i , i = n_g + 1,\ldots,n_h \right\} \tag{3.3}$$

X_L is the set described by lower and upper bounds (linear inequalities) (2.9).

$$X_L = \left\{ x \in R^n : l_j \leq x_j \leq u_j , j = 1,\ldots,n \right\} \tag{3.4}$$

Thus, the problem (3.1) is a standard nonlinear constrained optimization problem. In this package a ***double iterative penalty algorithm*** is used for the problem solver.

The lower level algorithm solves the problem:

$$\min_{x \in X_L} \left[F(x) = s_w(x) + p(x, v, k) \right] \tag{3.5}$$

The objective function used here, called penalty function, consists of the sum of the original objective function and a penalty term $p(x, v, k)$ (the precise form of this term is given later, see (3.6)). Linear constraints are satisfied at each step of the algorithm because a special method of reduced gradient is used for this purpose. Nonlinear constraints, however, are violated and the penalty term in (3.5) is related to this violation.

The upper level algorithm adjusts parameters v and k in the penalty function to satisfy nonlinear constraints. At each step of this algorithm, the lower level problem (3.5) is solved. However, the required accuracy of its solution depends on the violation of nonlinear constraints. Nonlinear constraints are strongly violated in very first iterations of the upper level algorithm and, therefore, the lower level problem can be solved very roughly.

3.1.2. Reduced gradient algorithm

The algorithm described here and applied in the software package uses gradient reduction, that is, an elimination of some gradient components. If, at some point, the value of a particular variable x_i is between its bounds l_i and u_i, then this variable can be either increased or decreased. However, if this value is equal to one of the bounds, say, to the upper bound u_i, then this variable can be only decreased. In such a case, negative values of the objective gradient component can not be accepted and are set to zero; this variable will remain unchanged in the next direction of search. This modified gradient is called reduced because some of its components are set to zero and it acts only in some subspace of the space of all variables.

The algorithm begins by calculating the gradient of the penalty function (3.5) at some starting point x^0. This gradient is then reduced in such a way that a nonzero step in the direction of search can be performed inside the set X_L. The step-size in this direction is calculated using quadratic approximations in the line search method. In the resulting point, the gradient is calculated and reduced again. First direction is just opposite to the reduced gradient (minus gradient), the next directions are conjugate directions constructed on reduced gradients. After some number of iterations the algorithm resets itself and uses minus gradient direction again.

Following notation and symbols will be used in the detailed description of the algorithm :

- ε - accuracy parameter given from the upper level algorithm;
- k - iteration number;
- m - number of conjugate direction.

The algorithm can be characterized by the following steps:

1° Initialize: Set $k = 0$ and $m = 0$

2° Calculate gradient: $g^k = \nabla F(x^k)$

3° Gradient reduction for each $i = 1, \ldots, n$:

if $g_i^k > 0$ and $x_i^k = l_i$ or $g_i^k < 0$ and $x_i^k = u_i$ then set $g_i^k = 0$.

If the resulting subspace is different than this obtained in last preceding iteration, set $m = 0$.

4° Stop test: if $|| g^k || \leq \varepsilon$ then stop .

5° Calculation of new direction: if $m = 0$ or m is greater than the number of nonzero elements in g^k then get a simple direction

$$d^k = -g^k$$

otherwise calculate conjugate direction using Polak-Ribiere algorithm:

$$d^k = -g^k + \beta^k d^{k-1}$$

with

$$\beta^k = \frac{< g^k , (g^k - g^{k-1}) >}{< g^{k-1}, g^{k-1} >}$$

6° Direction check: if $< d^k , g^k > \geq 0$ then set $m = 0$ and go back to step 5°

7° Step-size limit:

$$\tau_M = \min_{\substack{i=1,\ldots,n \\ d_i^k \neq 0}} \max \left\{ \frac{u_i - x_i^k}{d_i^k} , \frac{x_i^k - l_i}{-d_i^k} \right\}$$

8° Line search: find step-size $\hat{\tau}$ such that

$$f(x^k + \hat{\tau} d^k) = \min_{[0, \tau_M]} f(x^k + \tau d^k)$$

If it fails ,i.e. $\hat{\tau} = 0$ then set $m = 0$ and go back to step 5°

9° Step:

$$x^{k+1} = x^k + \hat{\tau} d^k \quad , \quad k = k + 1 \quad , \quad m = m + 1$$

go to step 2°.

The actual algorithm implemented in FORTRAN is much more complicated. It includes many safeguards and it takes into account round off errors and finite accuracy of computations.

3.1.3. Penalty shift algorithm

The penalty term in (3.5) has the following form:

$$p(x, v, k) = \sum_{i=1}^{n_g} k_i \left[f_i(x) - b_i + v_i \right]^* \max \left[0 , \left[f_i(x) - b_i + v_i \right] \right] + \sum_{i=n_g+1}^{n_h} k_i \left[f_i(x) - b_i + v_i \right]^2 \qquad (3.6)$$

with

- k_i - positive penalty coefficients,
- v_i - penalty shifts,
  v_i non-negative for $i = 1, \ldots, n_g$,
  v_i unconstrained for $i = n_g + 1, \ldots, n_h$.

Standard penalty algorithms use any method of unconstrained optimization to solve (3.5). The penalty coefficients are then increased according to the violation of constraints obtained and (3.5) is solved again. This procedure is repeated until the solution of (3.5) is forced by the penalty term to be sufficiently close to the feasible set. Most frequently, the penalty coefficients become very large which makes problem (3.5) ill conditioned.

In the *shifted penalty function algorithm* there is no need to increase penalty coefficients as strongly as in standard penalty algorithms; penalty shifts are used instead to increase the penalty effect. The penalty function is shifted in the direction opposite to the constraint violation. In the case of inequality constraints, this leads to a shift "inside" the admissible set X_N ; to shift a constraint $f_i(x) \le b_i$ inside, one must decrease the right hand side b_i or, equivalently, set a positive value of the shift parameter v_i in (3.6). The penalty term becomes then active in a band measured by v_i along the corresponding boundary of the set X_N . In the case of equality constraints, penalty shifts can be either positive or negative; the adequate shift is just in the direction opposite to the current violation of constraints when solving (3.5). In both cases, penalty shifts increase the related penalty term in (3.6). Additional safeguards are employed to avoid stopping the algorithm inside the feasible set with respect to the active constraint.

The algorithm is characterized by the following steps:

1° Set initial $k_i = k_i^0$ and $v_i = 0$, $i = 1,\ldots,n_h$

2° Solve problem (3.5) using the reduced conjugate gradient algorithm; calculate maximal violation of constraints at the solution point:

$$q = \max(gv, gf, hv),$$

with

gv - norm of violation of inequality constraints:

$$gv = \max_{i=1,\ldots,n_g} \max(0, (f_i(x) - b_i))$$

gf - norm of inequality constraints forced inside the feasible set:

$$gf = \max_{\substack{i=1,\ldots,n_g \\ i: v_i > 0}} (b_i - f_i(x))$$

hv - is the norm of violation of equality constraints:

$$hv = \max_{i=n_g+1,\ldots,n_h} | f_i(x) - b_i |$$

3° Stop test: if q is less than a given accuracy coefficient η then stop.

4° If it is a first iteration then set $c = q$ and go to step 6° .

5° If $q > d$ then set $p = d$ and go to step 9° .

6° Penalty shifts:

$$v_i = \max(0, v_i + f_i(x) - b_i), \; i = 1, \ldots, n_g$$

$$v_i = v_i + f_i(x) - b_i, \; i = n_g + 1, \ldots, n_h$$

7° If $q > c$ then set $p = c$ and go to step 9° .

8° Set $d = q$, $c = 0.4 * q$ and go back to step 2° .

9° For each constraint violated more than p , increase the penalty coefficients $k_i = 2 * k_i$ and decrease the penalty shifts $v_i = 0.5 * v_i$. Go to step 8° .

In most cases changes of penalty shifts in step 6° are sufficient to get convergence. However, if the rate of convergence in not satisfactory, penalty coefficients are also changed in step 9° .

3.1.4. Verification of gradients

Depending on the number of time periods taken into account in the model, compare Section 2.1, the number of functions $f_i(x)$ in the general form (2.6) - (2.9) of the model changes from about thirty to several hundreds. The number of nonzero gradient elements changes from several hundreds to several thousands. The model of the system itself is rather complicated: it is not a single FORTRAN subroutine, but rather large set of interconnected subroutines and data blocks, see Kaden 1986. Thus it is very easy to make a mistake calculating analytical forms of the derivatives of complex expressions in the model.

Unfortunately, optimization algorithms are very sensitive to such mistakes and become inefficient or even fail if the changes of objective values and constraints are inconsistent with their gradients.

Therefore it is necessary to check the consistency of all gradients after each modification of the model. For this purpose, a special numerical algorithm was included in the optimization package. In this algorithm gradients are checked numerically by applying a finite difference method.

According to the Lagrange theorem in a single dimensional case the term:

$$\frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

is equal to the derivative of the function $f(x)$ at some point between x_1 and x_2. Typical algorithms of gradient estimation assume that this term is the best approximation of the gradient in the center of this interval. However, such approach requires $2 * n$ points to estimate a gradient in the n dimensional case .

A regular simplex method can be applied to minimize the number of points where the functions $f_i(x)$ have to be calculated. At a neighborhood of such a point x_0 a regular simplex of $n+1$ vertices $x_1, x_2, \ldots, x_{n+1}$ is constructed with x_0 in the center:

$$x_0 = \frac{1}{n+1} \sum_{j=1}^{n+1} x_j$$

In a regular simplex all distances $r = ||x_0 - x_j||$, $j = 1, \ldots, n+1$ are equal and value r is called the radius of the simplex. The problem is to find the directions $d_j = x_j - x_0$ that span the regular simplex.

The algorithm of gradient verification consist of two parts: construction of a simplex (steps $1° - 4°$) and calculation of gradient estimate (steps $5° - 7°$).

1° Calculate scalars:

$$\delta_1 = -r \sqrt{\frac{n+1}{2n}} \quad , \quad \delta_j = \delta_{j-1} \sqrt{\frac{2}{j*(j+1)}}, \quad j = 2, \ldots, n$$

2° Calculate first direction:

$$d_1 = \begin{vmatrix} \delta_1 \\ \delta_2 \\ \cdot \\ \cdot \\ \delta_n \end{vmatrix}$$

3° Calculate next directions recursively for $j = 2,\ldots,n+1$

$$d_j = \frac{1}{j-1} \sum_{i=1}^{j-1} d_i$$

but for each d_j additionally changing only component number $j-1$

$$d_j^{j-1} = -(j-1)\, d_j^{j-1}$$

4° Calculate vertices of the simplex as:

$$x_j = x_0 + d_j , \quad j = 1, \ldots, n+1$$

5° At each point x_j, $j = 1,\ldots,n+1$, calculate values

$$f_i^j = f_i(x_j)$$

6° For each functions $f_i(x)$ calculate an estimate of its gradient component:

$$\frac{\partial f_i(x_0)}{\partial x_j} = \frac{f_i^{j+1} - f_i^j}{-2 * \delta_j} , \quad j = 1, \ldots, n$$

7° Compare these numerical estimates with the gradients calculated analytically at point x_0.

3.2. Program description

3.2.1. Program structure

The problem solver MSPN for non-linear multi-criteria analysis is embedded in the complex DSS MINE as it is illustrated in Figure 4. A detailed description of the model system is given in Kaden, 1986.
The MSPN package is a set of 21 interlinked FORTRAN subroutines and functions. In Figure 5 the structure of the MSPN package is depicted.
A detailed description of the used COMMON blocks and of all subroutine and functions is given in the Appendices A and B.

The MSPN package contains only the algorithms for multiobjective optimization. All input data and the mathematical model (Eq.(2.6)-(2.9)) are prepared outside ouf MSPN in the DSS MINE. The same holds true for the storage, processing and output of optimization results. A detailed description is given in Kaden, 1986.

The only link between model users and the MSPN package takes place in the case of modification of optimization control parameters (default values are defined), and in case of numerical problems. In Section 3.2.3 and 3.2.4 some informations are given.

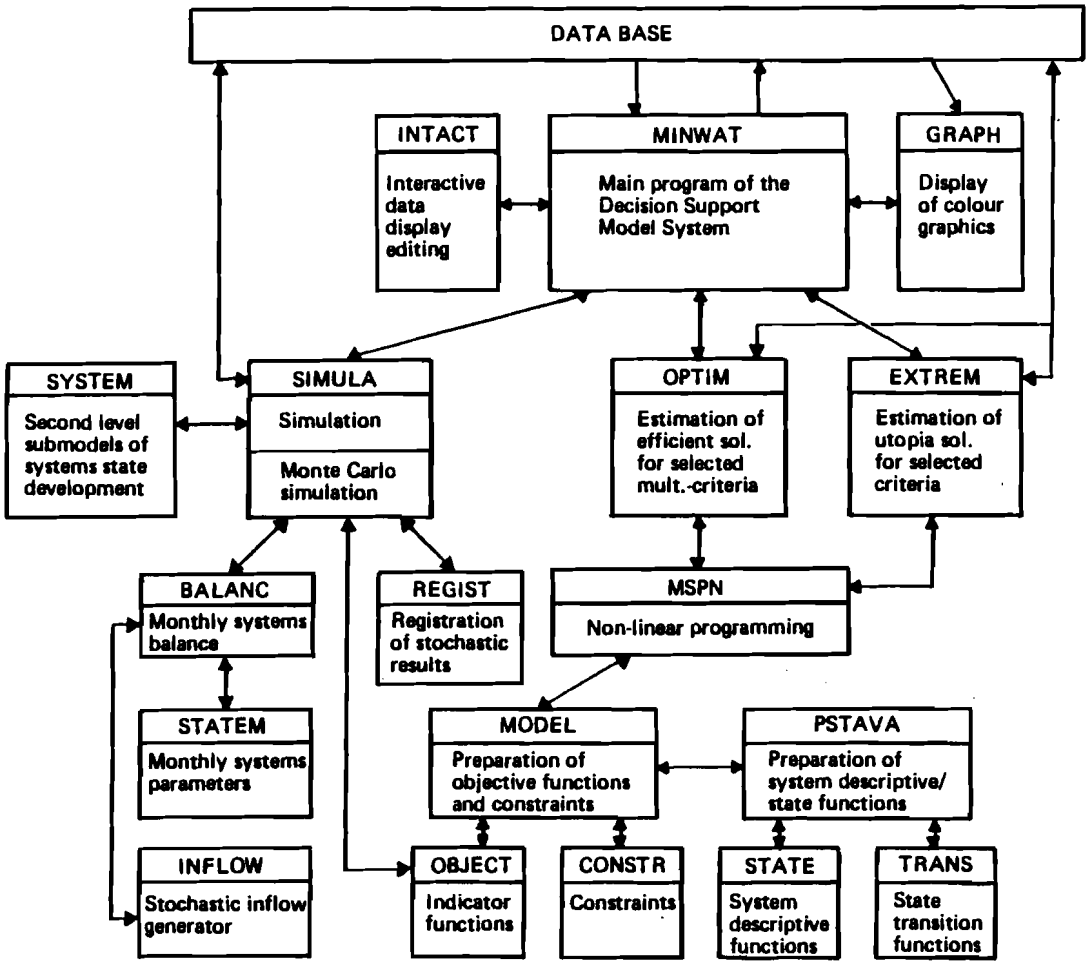

Figure 4: Structure of the Decision Support System MINE

3.2.2. Storage method for Jacobian matrix

Since the Jacobian matrix is sparse (compare Section 2.1) its columns (gradients of constraints) are stored in a special way using an indirect indexing method. The gradients of constraints are not stored as n-dimensional vectors (sequences of n elements) but rather as sequences of elements known to be 'active' i.e. such elements which can have nonzero values. The remaining elements ('non active') must be known to be equal to zero during the entire optimization process.

The active elements are assumed to be listed as sequences of elements ordered according to their place in the original n dimensional vector. This method of listing is not obligatory but the package is more efficient if active elements are ordered in such a way.

The beginning indices and the lengths of these sequences are stored in the matrix of indices icon(). The nonzero gradient elements themselves are stored in the Jacobian area of a general purpose storage matrix r() (see descriptions of COMMON blocks /optc/ and /opti/ in Appendix A).

The array icon() is used for addressing the array icon() itself and the storage array r(). It has two logical parts: the first part has ng + nh elements and contains first level indices for addressing the second part of icon(), the second part contains the descriptions of gradients of constraints. The length of the second part of icon() is equal to ng + nh plus twice the number of separate sequences of elements used for storing all the elements of the constraint gradients.

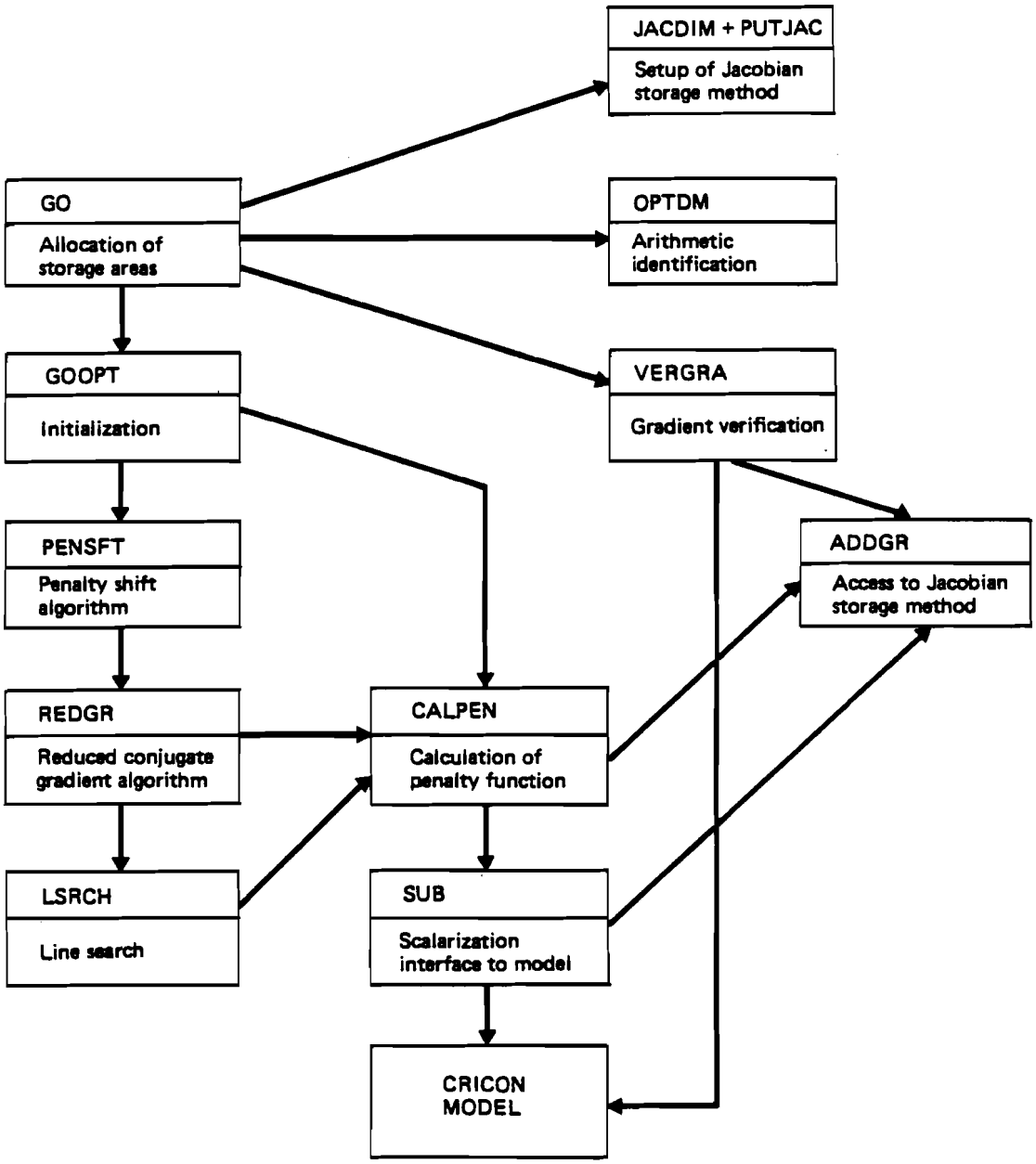

Figure 5: Structure of the MSPN package

For a given constraint number i :

icon(i) - contains the index of an element in the second part of icon(), where the gradient description of this particular constraint begins, define ki = icon (i);

icon(ki) - contains the minus index of the array r() where the first active element of this gradient is stored. All others active elements of this constraint are stored in the next consecutive elements of the array r();

icon(ki+1) - contains the number of the first active element of the gradient in the whole n dimensional vector; if it is non-positive, then this constraint has no active elements at all (in case of dummy constraint) and it is the end of the description of this constraint;

icon(ki+2) - contains the length of the sequence of active elements (it can be equal to one);

icon(ki+3) - contains the number of the beginning of the next sequence of active elements of the gradient; if it is positive then element icon(ki+4) performs the same role as icon(ki+2), otherwise it is the end of the description of this constraint.

Pairs number/length are repeated as many times as required to describe all separate nonzero sequences of elements of the gradient.

3.2.3. Optimization control parameters

According to the optimization algorithm described in Section 3.1 the following control parameters are needed:

Range (rk):

Roughly estimated range of changes of variables during the optimization process, scaling of variables is useful.

default: rk =1.

Norm (eps):

The stop test in the reduced conjugate gradient algorithm checks whether the norm of gradients of the penalty function is less then the value of *eps* (denoted as ε in the step 4° of the algorithm in the Section 3.1.2), if *eps* is to small stop with ip=4.

default: *eps* =0.1

Violation (eta):

The stop test in the penalty shift algorithm checks whether all constraints are violated less then η (denoted as η in the step 3° of the algorithm in the Section 3.1.3)

default: $eta = 10^{-3}$

Penalty (penco):

The initial value of penalty coefficients in the penalty shift algorithm (denoted as k_i^0 in the step 1° of the penalty shift algorithm - Section 3.1.3). As an estimate the ratio of gradients of the objective function to gradients of constraints should be used.

default: *penco* =1.

Iterations (lsm):

Maximal number of iterations (calculations of model values)

default: lsm =1000

rho:

Parameter for scalarizing function of the reference point method, see ρ in Section 2.3, Eq.(2.11)

default: *rho*=4

In Section 4.1 the influence of control parameters on the optimization procedure is analyzed.

3.2.4. Error handling

The basic presumption for a successful optimization is the correct model formulation, above all the analytical gradients. The latter can be checked using the verification algorithm, see Section 3.1.4.

From the MSPN package the following interrupts are realized in case of possible errors:

- Optimal solution not found because of the limit of iteration number (ip=3).
- Optimal solution not found because of numerical errors - required accuracy not attainable. In this case the control parameter *eps* might be increased. But, usually this interrupt indicates that analytical formulas for functions or their gradients are wrong (ip=4).
- Optimization algorithms can not start because of too small storage area reserved in the array r() in the COMMON block /**optc**/ (ip=5). For storage allocation see Kaden, 1986.
- Optimization fails because of the empty feasible set (ip=6). This might be caused by to strong constraints and bounds or by model formulation errors.

In case of the interrupts (ip=3,4,6) the model output can be used to localize possible errors. This output includes:

deps and eta	-	values compared on the stop test with eps and eta parameters, respectively; deps is the current norm of the penalty function, deta is the current violation of constraints
ip	-	error condition parameter; ip=2 means "optimal solution found"
iter	-	total number of model calculations (calls for subroutine cricon).

Lines with numbers at the beginning describe active constraints with:

** - their number (column in the Jacobian matrix),

w - the current value of constraint,

v - the current penalty shift,

penal- the current coefficient.

The value $l=(v+w)*penal$ is an approximation of a Lagrange multiplier with respect to the scalarized objective function. It may be used for a post-optimal analysis.

4. Computational Tests

4.1. Robustness of MSPN solver

In order to analyze the robustness of the MSPN algorithm with respect to the optimization control parameters, see Section 3.2.3, a series of numerical tests with the DSS MINE have been performed.

The tests have been done for a planning horizon of 7 planning periods. As criteria the following had been selected:

dev-m	-	Deviation municipal water demand/supply,
dev-i	-	Deviation industrial water demand/supply,
cost-mi	-	Total mine drainage cost,
cost-m	-	Cost for municipal water supply,
cost-i	-	Cost for industrial water supply.

For each criteria the utopia point was selected as reference point. As the starting point one with significant deviation to the expected solution was used in order to realize a large number of iterations.

In Figure 6 some results are depicted illustrating the influence of control parameters on the results. Only those criteria are shown which are strongly affected by the parameters. For the criteria *dev-m, dev-i, cost-m* the influence is almost negligible.
From these tests the following conclusions can be drawn:

range (rk):

The influence of this parameter on the numerical results is negligible. The variations are less then 1%. But a to small value affects the number of iterations significantly. According to Figure 6a values between 0.1 and 5 are reasonable.

violation (eta):

The influence of this parameter is again small, less then 1%. Smaller numbers of *eta* increase the number of iterations. As a compromise $eta=10^{-3}$ should be chosen.

norm (eps):

As expected this parameter stronger effects numerical results and number of iterations. Only values greater/equal 0.05 could be chosen. For the value *eps* =0.01 the required accuracy has not been attainable. The results deviate in a range between maximum 5 and 10% - quite acceptable from the practical point of view. Furthermore in each case Pareto-optimality (at least locally) was achieved, compare results for *cost-mi* and *cost-i* in Figure 6c. As a good compromise between accuracy and number of iterations *eps* =0.1 should be chosen.

penalty (penco):

The initial penalty coefficient has been varied between 0.5 and 10. It does practically not effect the numerical results. The influence on the number of iterations is small for values between 0.5 and 5. Only in the case of *penco* =10 the number of iterations increased. As a good compromise *penco* =1. is proposed.

Above the influence of optimization parameters on the criteria as integral parameters has been analyzed. Another interesting question is their influence on the variables (decisions).

The same parameter combinations as depicted in Figure 6 have been analyzed with respect to their impact on the variables. Analogously to the criteria the MSPN-algorithm is also very robust with respect to the variables. The influence of varying **range** and **violation** is almost negligible. The deviations are less then 5%, in most cases even less then 1% (related to the value of the variable). Only in one case the deviations are stronger. This is depicted in Figure 7 for the variables $q_{b,s}$ and $q_{b,ez}$ as decisions on water allocation (water quantity). For details on the meaning of the variables see Kaden et al., 1985a.
The solution for period 1 divers significantly in the case of **range** *rk* =0.1 from the

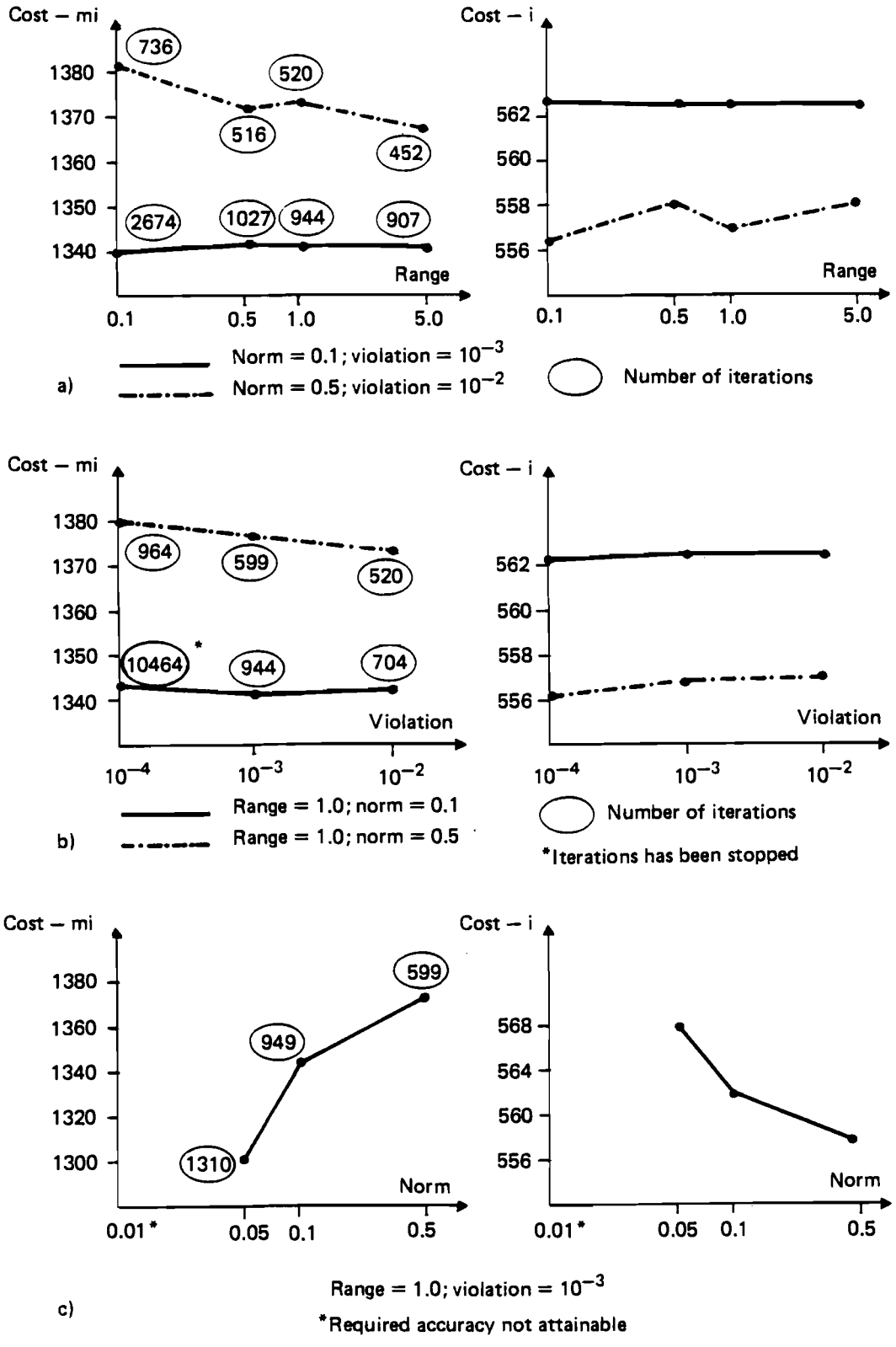

Figure 6: Influence of optimization control parameters

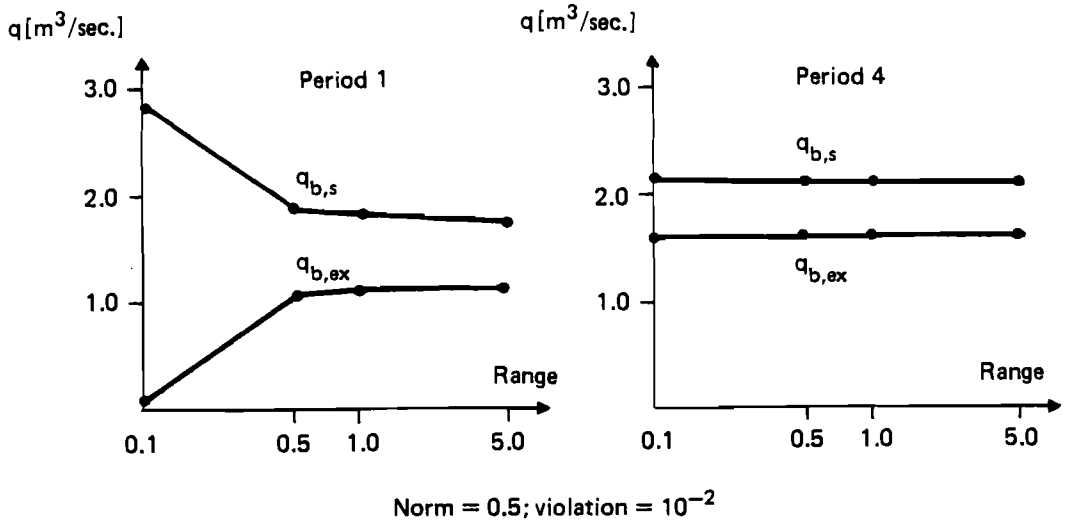

Figure 7: Influence of optimization parameters on variables

other solutions. For that two reasons are seen:

- the optimization is performed with respect to criteria as integral values over the whole planning horizon. Due to the increasing time steps of planning periods later planning periods represents a larger part of the criteria, get a higher weight.
- the parameter combination of a small **norm** esp=0.1 with a rough **violation** η=0.01 is not very reasonable.

Nevertheless the water balance is satisfied. The increased $q_{b,s}$-value is compensated by a reduced $q_{b,ex}$.

The effect of the **norm** is more significant as it should be expected from the results for criteria, see Figure 6c. In Figure 8 the results for four variables are depicted. These are the variables with the strongest deviations. For all other variables the deviations are less then 5%.
The deviations are hardly to be explained. Probably they are above all caused by flat objective functions with respect to the given variables. Small numerical deviations due to different accuracy could lead to different solutions.

4.2. Influence of starting point values

The optimization problem to be solved is non-linear in most of its parts. For such a complicated mathematical model as it is given for the DSS MINE it is practically impossible to analyze analytically the properties of the objective function with respect to convexity and extremal values. The existence of local optima has to be expected. Or, with other words, the estimated solution is not necessarily an global optimal solution.

Two principle possibilities are available to check the solution behavior with respect to local/global optima:

- Application of an optimization procedure resulting per definition in a solution being a global optimum. Such property posses some random search methods, e.g. ASTOP, Born 1985. The numerical effort of such methods is extremely high, their applicability for the given problem is still under study.

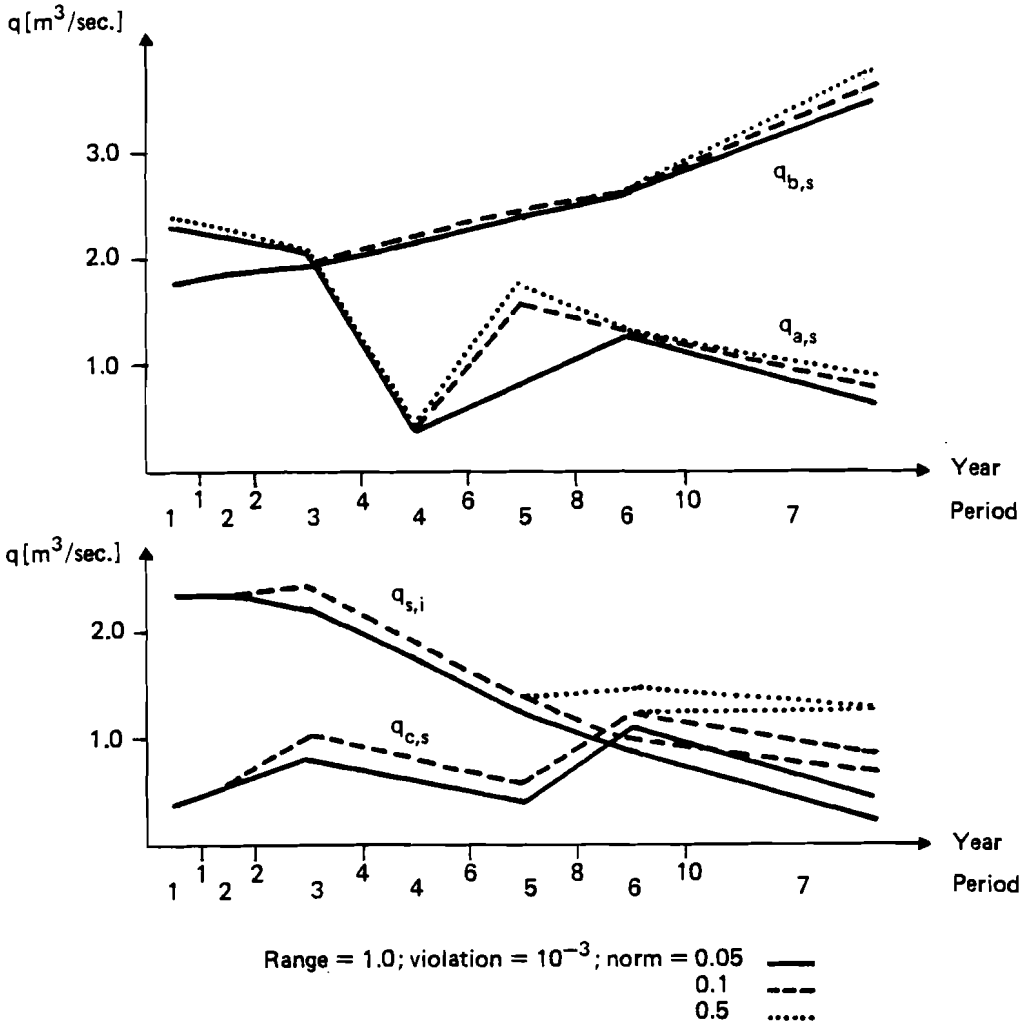

Figure 8: Influence of the **norm** on variables

- Experimental analysis varying the starting points for optimization.

In the following a few results for an experimental analysis are given.

The most logical way to analyze the influence of starting point values is the random selection of the starting points between upper and lower bound. This has been done using a random generator for uniform distributed random numbers. For the tests the same criteria as described in Section 4.1 have been considered. Results for selected criteria are listed in Table 1.
The Table illustrates that only the criteria *cost −mi* and *cost −i* significantly depend on starting point values. The maximal deviation is in the range of 10%.

An interesting question is, whether the different starting points result in different local optima, or the results are simply different Pareto-optimal solutions. In Figure 9 the results for two criteria are graphically illustrated.
It is out of question that only 11 tests are statistically not sufficiently for generalization. Nevertheless from the Figure 9 could be concluded that some of the solutions (connected by the dashed line) are global Pareto-optimal, a few others only local. But even for the "worst solution" the distance to the next hypothetic global

Table 1: Solutions for randomly selected starting point values

criteria	ref.-point	utopia-point	nadir-point	solutions										
dev –m [*l / sec.*]	0	0	31	7	8	8	8	8	8	8	8	8	8	8
dev –i [*l / sec.*]	0	0	106	3	2	4	4	1	2	1	1	1	1	1
cost –mi [*Mill.M*]	1151	1151	1476	1261	1261	1466	1315	1415	1411	1359	1443	1381	1446	1459
cost –m [*Mill.M*]	12	12	38	13	13	13	13	13	14	13	13	13	13	13
cost –i [*Mill.M*]	509	509	581	561	562	534	553	551	550	545	532	540	538	551

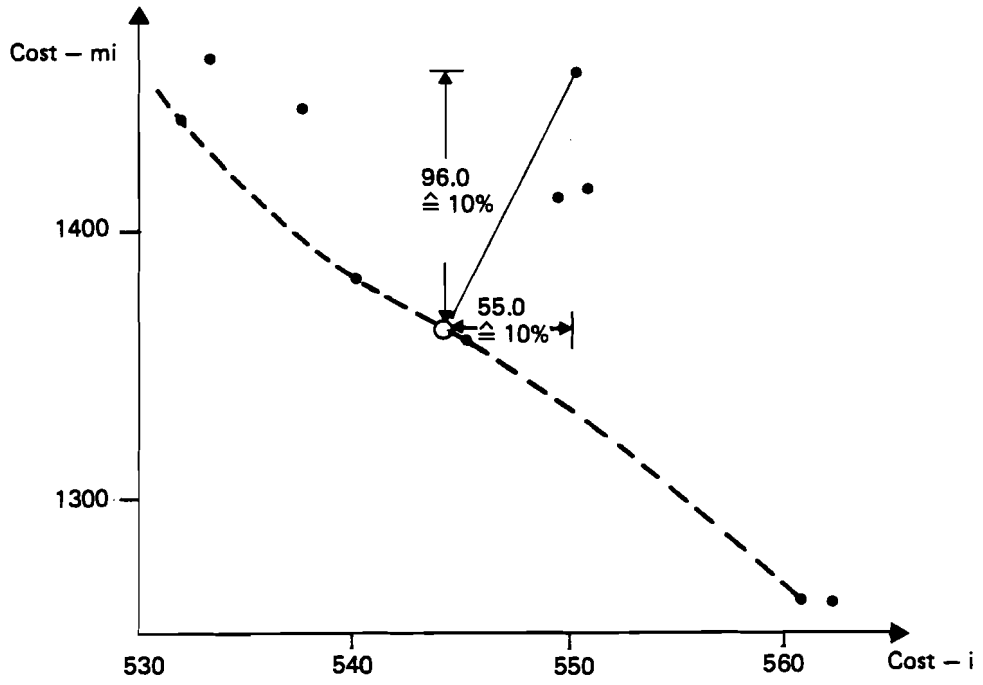

Figure 9: Solutions for different starting point values

Pareto-optimal point is less then 7 - 10% of its value.

Another series of numerical tests has been done chosing lower and upper bounds of variables as starting point values. For these tests only one planning period has been analyzed. For each variable one run was made with lower and upper bound, the results were compared with an arbitrary "nominal" solution. Fourteen variables have been varied (28 runs). In Table 2 some results are depicted.

The results illustrate the small influence of variations of single starting points for one period. The effect is accumulating if more planning periods are under

Table 2: Statistical analysis of the influence of starting point on criteria (first period)

criteria	nominal value	mean value	standard deviation
dev –m [*l / sec.*]	4	4	0.2
dev –i [*l / sec.*]	2	2	0.8
cost –mi [*Mill.Mark*]	63.6	64.2	1.2
cost –m [*Mill.Mark*]	1.1	1.1	0.0
cost –i [*Mill.Mark*]	35.6	35.5	0.7

consideration. In all cases for reasonable starting values the deviations are in the range of practical acceptance.

4.3. Conclusions

A series of numerical tests has been performed in order to analyze the robustness of the MSPN-algorithm with respect to optimization parameters and to check the influence of starting point values. From the results can be concluded that the MSPN-algorithm is well suited for the given problem. Variations in optimization parameters do not effect the solutions significantly.

The mathematical model of the DSS-MINE behaves robust with respect to the selection of starting points. It can not be excluded that local optima are estimated, but the local optima are expected close to global optima.

For all tests deviations in criteria values have been found less then about 10%. Taking into the account the accuracy of input data, the simplified mathematical models of environmental and socio-economic processes this deviation is fully acceptable.

Some of the tests are from the practical point of view not realistically, especially the random generation of starting point values. For the practical problem the starting point values are in most cases known quite well and a consistency of variables in time has to be considered. (It does not make much sense to change variables as water allocation drastically between planning periods). Consequently the problem of starting point selection and local/global optima is from the practical point of view less significant. Furthermore the results of the multi-criteria analysis within the *planning model* serves only as a guideline for a more detailed analysis with the second level *management model.*

5. References

Born,J. 1985. Adaptively controlled random search - a variance function approach. Systems analysis, Modelling, Simulation, Vol. 2, No. 2, pp.109-112, Akademie-Verlag Berlin.

Kaden,S., Hummel,J., Luckner,L., Peukert,D., Tiemer,K. 1985a. Water Policies: regions with open-pit lignite mining (Introduction to the IIASA study), IIASA, WP-85-4, p.67.

Kaden,S., Luckner,L., Peukert,D., Tiemer,K. 1985b. Decision support model system for regional water policies in open-pit lignite mining areas. International Journal of Mine Water, Vol. 4, No.1, pp.1-16.

Kaden,S, Michels,I., Tiemer,K. 1986. Decision Support System MINE; the Management Model, IIASA, CP-86, forthcoming.

Kaden,S. 1986. Decision Support System MINE; Description of the Model System, IIASA, WP-86, forthcoming.

Wierzbicki,A.P. 1983. A Mathematical Basis for Satisficing Decision Making. Mathematical Modeling USA 3:391-405 (Report IIASA RR-83-7).

APPENDIX A: COMMON blocks

All COMMON blocks used in the solver part of the program are described here in lexicographical order. For each block, the full list of elements including a short description is given after its name. The types of elements are described using FORTRAN keywords preceding each element name.

/consti/

integer lbgh - the number of Jacobian matrix elements known to be nonzero and stored;

integer kbgh - the index of the beginning of an area in the array r(.) where the Jacobian matrix is stored;

integer icon() - the array of indices organizing storage of Jacobian matrix in the array r(.) (see the description in Section 3.2.2); the total size of this array is defined outside the package (see the description of subroutine **go**);

/opta/

integer k1 - the index of the beginning of an area in the array r() where current values of the scalarized objective function and constraints are stored;

integer k2 - the index of the beginning of an area in the array r() where the penalty shifts are stored;

integer k3 - the index of the beginning of an area in the array r() where the penalty coefficients are stored;

integer k5 - the index of the beginning of an area in the array r() where the best feasible point x is stored;

integer k6 - the index of the beginning of an area in the array r() where the values of scalarized objective function and constraints at the best feasible point x are stored;

/optc/

real*8 r() - a general purpose array used for storage of several kinds of data in most subroutines; areas of any size in this array are allocated using indices in the COMMON block /opti/ (see below); the total size of this array is defined outside the package (see the description of subroutine **go**);

/optd/

real*8 db - the absolute accuracy of real*8 computations, related to the smallest real*8 positive number distinguishable from real*8 zero;

real*8 dw - the relative accuracy of real*8 computations, related to the smallest real*8 positive number which added to real*8 one is distinguishable from real*8 one;

/optf/

logical ws - a logical variable which is true if for some reasons the value and/or gradient of penalty function are recalculated using previous data (the values and/or gradients of objectives and constraints); otherwise it is false;

logical tp - a logical variable which is false if during current line search at least one step with improvement was performed, otherwise it is true;

/optg/

integer kb - the index of the beginning of an area in the array r() where the gradient of the penalty function is stored;

/opti/

integer i1 - the index of the first element of the array r() free for use, elements r(1) - r(i1-1) being already allocated. In order to use a data area of the size M subroutine uses i1 as the beginning index of this area and sets i1=i1+M to inform all other subroutines that this area is already allocated;

integer i2 - the index of the last element of the array r() increased by one; i1 must always be less than i2. This condition is checked in all subroutines which allocate areas in array r(). If this condition is violated, then the value of IP is set to IP=5 and this value is returned from the package;

/optk/

integer kf - the index of the place in the array r() where the value of the scalarized objective function is stored;

integer kgh - the index of the beginning of an area in the array r() where values of constraints are stored;

integer kbf - the index of the beginning of an area in the array r() where the gradient of the scalarized objective function is stored;

integer kbgh - the index of the beginning of an area in the array r() where the Jacobian matrix is stored (this is the same value as kbgh in COMMON block /consti/ - it is repeated here for convenience);

integer krhs - the index of the beginning of an area in the array r() where the right hand sides of constraints are stored;

/optk1/

integer kx - the index of the beginning of an area in the array r() where the current point is stored;

integer kxl - the index of the beginning of an area in the array r() where lower bounds are stored;

integer kxu - the index of the beginning of an area in the array r() where upper bounds are stored;

/optn/

integer n - the number of independent variables x;

integer ng - the number of nonlinear inequality constraints;

integer nh - the number of nonlinear equality constraints;

/opto/

real*8 eta - an accuracy parameter set by the user (see an2 below);

logical logn - a logical variable which is true if at least one feasible point was found, otherwise it is false;

/opts/

integer ls - the number of calls to subroutine **cricon** calculating the model (see Kaden 1986) that remains to the end of computations if the iteration number limit will become active. At the beginning, this number is set to the maximal number of iterations defined by the user, and then subsequently decreased;

real*8 an1 - the value of the norm of penalty function minimized in the reduced gradient algorithm. The stop test checks whether this value is less than the given accuracy parameter eps;

real*8 an2 - the value of current maximal violation of constraints. The stop test of the shifted penalty algorithm checks whether this value is less than the given accuracy parameter eta;

APPENDIX B: Subroutines and functions

All subroutines and functions of the MSPN package are described here in the alphabetical order. Each formal parameter and COMMON block element is preceded by its function code and FORTRAN type. Possible function codes are:

(i) - input item, not changed inside the routine;

(o) - output item, the input value does not influence the calculations inside the routine;

(-) - item not used, given for alignment purposes only;

(i/o) - input and output item.

The COMMON block elements are already described in Appendix A. In the following only the role of formal parameters of subroutines and functions is described.

*** subroutine **addgr**

Adds weighted gradient of constraint 'lk' to the 'n'-dimensional gradient 'gl' of the penalty function, 'rho' is the weight (approximation of a Lagrange multiplier) of this constraint. The Jacobian matrix is sparse and so it is stored in a special way using indirect indexing method (see **jacdim** function description and Section 3.2.2).

Parameters:

(i/o)	real*8 gl	-	n dimensional array of gradient of penalty function
(i)	real*8 rho	-	weight of added constraint
(i)	integer lk	-	number of added constraint
(i)	integer n	-	number of gradient elements

COMMON blocks:

/consti/

(-) integer lbgh,kbgh

(i) integer icon()

/optc/

(i) real*8 r()

*** function real*8 **anorm**

Calculates and returns square of Euclidean norm of 'n'-dimensional vector 'a'. Returns zero if 'n' is not positive.

Parameters:

(i)	real*8 a()	-	'n' dimensional array
(i)	integer n	-	dimension of array 'a'

*** function real*8 **calpen**

Calculates and returns the value of the penalty function and/or calculates the gradient of the penalty function according to the value of parameter lb:.

lb= 0 calpen calculates only value of the penalty function

lb=	-1	calpen calculates only gradient of the penalty function
lb=	+1	calpen calculates both, value and gradient of the penalty function
lb=	+2	initialization of internal data

Parameters:

(i)	real*8 x	-	the 'n' dimensional array containing a point where value and/or gradient have to be calculated
(i)	integer lb	-	determines the required function (see above)

COMMON blocks:

/opta/

(i) integer k1,k2,k3,k5,k6

/optc/

(i/o) real*8 r()

/optf/

(i/o) logical ws
(o) logical tp

/optg/

(i) integer kb

/optk/

(i) integer kf,kgh,kbf,kbgh,krhs

/optn/

(i) integer n,ng,nh

/opto/

(i) real*8 eta
(i/o) logical logn

/opts/

(o) integer ls

*** subroutine **go**

It is the entry to the optimization system. It performs several functions:

- initialization of the whole optimization system;
- reservation of main storage areas checking whether the required space is available;
- call for **specsi** subroutine for dimension informations and control parameters from the data base;

- call for **jacdim** subroutine to initialize the storage algorithm for the Jacobian matrix;
- call for **datiou** subroutine for initial data from data base;
- if the value of 'iver' variable is non zero call for gradient verification subroutine;
- call the subroutine **go** for starting the optimization process;
- call for **datiou** subroutine for saving results of optimization;
- output error messages if optimization fails.

The error condition is detected using the value of parameter ip returned from optimization subroutines, possible values and their meanings are:

ip=	1	Used always as initial value when optimization subroutines are called.
ip=	2	Optimal solution with required accuracy (ε and η) was found after no more than lsm iterations.
ip=	3	Optimal solution not found because of the limit of iteration number.
ip=	4	Optimal solution not found because of numerical errors - required accuracy not attainable (May be analytical formulas for functions or their gradients are wrong ?).
ip=	5	Optimization algorithms can not start because of too small storage area reserved in the array r() in the COMMON block /**optc**/.
ip=	6	Optimization fails because of the empty feasible set.

Parameters:

(i/o)	integer is	-	declared size of the real*8 array r() in the COMMON block /**optc**/
(i)	integer isi	-	declared size of the integer array icon() in the COMMON block /**consti**/

COMMON blocks:

/opta/

(-) integer k1,k2,k3
(o) integer k5,k6

/optc/

(i/o) real*8 r()

/opti/

(o) integer i1

/optk/

(o) integer kf,kgh,kbf,kbgh,krhs

/optk1/

(o) integer kx,kxl,kxu

/optn/

(o) integer n,ng,nh

/opts/

(i) integer ls

(i) real*8 an1,an2

/verify/

(i) integer iver

*** subroutine **goopt**

Performs some more initialization and calls the subroutine **pensft** which implements the penalty shift optimization algorithm.

Parameters:

(i)	integer n	-	the number of variables 'x'
(i)	integer ng	-	the number of inequality constraints
(i)	integer nh	-	the number of equality constraints
(i/o)	real*8 x	-	array of variables 'x', starting point and optimal point are set here
(i)	real*8 xl	-	lower bounds for variables 'x'
(i)	real*8 xu	-	upper bounds for variables 'x'
(i)	real*8 rk	-	roughly estimated range of changes of variables
(i)	real*8 eps	-	the stop test in the reduced conjugate gradient algorithm checks whether the norm of gradient of penalty function is less then value of eps (denoted as ε in the step 4° of the algorithm in the Section 3.1.2)
(i)	real*8 eta	-	the stop test in the penalty shift algorithm checks whether all constraints are violated less the eta (denoted as η in the step 3° of the algorithm in the Section 3.1.3)
(i)	real*8 penco	-	the initial value of penalty coefficients in the penalty shift algorithm (denoted as k_i^0 in the step 1° of the penalty shift algorithm - Section 3.1.3)
(i)	integer lsm	-	maximal number of iterations (calculations of model values)
(i)	integer is	-	total size of the real*8 array r() in the COMMON block /**optc**/
(i/o)	integer ip	-	error code parameter (see the description of **go** subroutine)

COMMON blocks:

/opti/

(-) integer i1

(o) integer i2

/opts/

(o) integer ls
(o) real*8 an1,an2

*** function integer **jacdim**

Fills the COMMON block /consti/ according to the structure of the Jacobian matrix. Calculates and returns the size of storage area in the array icon().

Parameters:

(i)	integer isi	-	the reserved size of the array icon()
(i)	integer n	-	the number of variables
(i)	integer ng	-	the number of inequality constraints
(i)	integer nh	-	the number of equality constraints
(i)	integer kbgh1	-	the index of the beginning of an area in the array r() where the Jacobian matrix will be stored

COMMON blocks

/consti/

(o) integer lbgh,kbgh,icon()

*** subroutine **putjac**

Reserves area for the next sequence of Jacobian elements and describes it in the array icon(). See Section 3.2.2 for description of the storage method.

Parameters:

(i/o)	integer k	-	current position in icon()
(i/o)	integer l	-	reserved area in r()
(i)	integer np	-	index of first variable of sequence of active elements
(i)	integer lp	-	length of this sequence

COMMON blocks:

/consti/

(-) integer lbgh,kbgh
(o) integer icon()

*** subroutine **krok**

Adds the 'n' dimensional array 'b' to the another 'n' dimensional array 'd' multiplied by the scalar 'c' and puts the result into the 'n' dimensional array 'a'. It does nothing if 'n' is not positive.

Parameters:

(o)	real*8 a	-	n dimensional array of result
(i)	real*8 b	-	first n dimensional array of data
(i)	real*8 c	-	weight of the second added array
(i)	real*8 d	-	second n dimensional array of data

(i)	integer n	-	dimension of all arrays

*** subroutine **krok1**

Adds the 'n' dimensional array 'a' to the another 'n' dimensional array 'c' multiplied by the scalar 'b' and puts the result back into the array 'a'. It does nothing if 'n' is not positive.

Parameters:

(i/o)	real*8 a	-	n dimensional array of result and first added array
(i)	real*8 b	-	weight of the second added array
(i)	real*8 c	-	second n dimensional array of data
(i)	integer n	-	dimension of all arrays

*** subroutine **lsrch**

Minimizes the penalty function on the current direction (Section 3.1.2 - step 8° of the algorithm).

Parameters:

(i)	integer n	-	the number of variables 'x'
(o)	real*8 x	-	the array containing optimal point 'x'
(i)	real*8 xo	-	the array containing initial values for 'x'
(i)	real*8 d	-	the array containing direction of changes of 'x'
(i/o)	real*8 yb	-	on entry - value of the minimized function in the starting point; on return - value of the minimized function in the optimal point
(i)	real*8 pp	-	the value of the directional derivative of the minimized function in the direction 'd'
(i/o)	real*8 zb	-	on entry - initial step-size in the direction 'd'; on exit - optimal step-size in the direction 'd'
(i/o)	real*8 zm	-	on entry - step-size limit; on exit - if it is non zero then optimal solution is equal to its limit given on entry
(i)	real*8 delta	-	relative accuracy of the directional minimization
(i)	integer lpm	-	maximal number of improvement steps
(i)	integer lcm	-	maximal number of all steps
(i/o)	integer ip	-	on entry - must be set to 1; on exit - equals 2 if any improvement found, otherwise equals 4

COMMON blocks:

/optc/

(i) real*8 r()

/optd/

(i) real*8 db

/optg/

(i) integer kb

*** subroutine **mnoz1**

Multiplies the 'n' dimensional vector 'a' times scalar 'b' and puts the result back into vector 'a'. It does nothing if 'n' is not positive.

Parameters:

(i/o)	real*8 a	-	data and result vector
(i)	real*8 b	-	multiplier
(i)	integer n	-	dimension of all vectors

*** subroutine **move**

Moves data from the 'n' dimensional array 'b' into the 'n' dimensional array 'a'. It does nothing if 'n' is not positive.

Parameters:

(i/o)	real*8 a	-	destination array
(i)	real*8 b	-	source array
(i)	integer n	-	dimension of all arrays

*** subroutine **optdm**

Determines absolute and relative accuracy of computations in the real*8 arithmetic (see the description of the COMMON block /**optd**/).

Parameters: none

COMMON blocks:

/optd/

(o) real*8 db,dw

*** subroutine **pensft**

This subroutine implements the penalty shift algorithm described in the Section 3.1.3.

Parameters:

(i/o)	integer ip	-	on entry - must be set to 1; on return - indicates error condition according to the description of subroutine **go**
(i)	integer n	-	the number of variables 'x'
(i/o)	real*8 x	-	the array of variables 'x'; contain starting point on entry and solution point on exit
(i)	real*8 xl	-	the lower bounds for variables 'x'
(i)	real*8 xu	-	the upper bounds for variables 'x'
(i)	real*8 zo	-	the initial step-size for reduced gradient algorithm
(i)	real*8 eps	-	the required accuracy of minimization
(i)	real*8 eta	-	the required accuracy of satisfying constraints
(i)	real*8 penco	-	the initial value of the penalty coefficients

COMMON blocks:

/opta/

(o) integer k1,k2,k3
(i) integer k5,k6

/optc/

(i/o) real*8 r()

/optd/

(i) real*8 db,dw

/optf/

(o) logical ws,tp

/optn/

(-) integer n1
(i) integer ng,nh

/opto/

(o) real*8 etac
(o) logical logn
(i) integer

/opts/

(i/o) integer ls
(-) real*8 an1
(o) real*8 q

*** subroutine **redgr**

This subroutine implements the reduced conjugate gradient algorithm described in the Section 3.1.2.

Parameters:

(i)	integer n	-	the number of variables 'x'
(i/o)	real*8 x	-	the array of variables 'x', contains starting point on entry and optimal point on exit
(i)	real*8 xlb	-	the lower bounds for variables 'x'
(i)	real*8 xub	-	the upper bounds for variables 'x'
(i)	real*8 zo	-	the initial step-size of the algorithm
(i)	real*8 eps	-	the accuracy of minimization
(i/o)	integer ip	-	the same meaning as the ip parameter in the **pensft** subroutine

COMMON blocks:

/optc/

(i/o) real*8 r()

/optd/

(i) real*8 db,dw

/optg/

(i) integer kb

/opti/

(i/o) integer i1
(i) integer i2

/opts/

(i/o) integer ls
(o) real*8 bn

*** subroutine **rozn1**

Subtracts the 'n' dimensional vector 'b' from the another 'n' dimensional vector 'a' and puts the result back into the vector 'a'. It does nothing if 'n' is not positive.

Parameters:

(i/o)	real*8 a	-	data and result vector
(i)	real*8 b	-	subtracted vector
(i)	integer n	-	dimension of all vectors

*** function real*8 **skal**

Calculates the scalar product of the two 'n' dimensional vectors 'a' and 'b'. Returns zero if 'n' is not positive.

Parameters:

(i)	real*8 a	-	first vector
(i)	real*8 b	-	second vector
(i)	integer n	-	dimension of all vectors

*** subroutine **sub**

This subroutine works as an interface between optimization subroutines and the model. It calculates value and/or gradient of the scalarized objective function. The input parameter lb selects one of the following possible functions of the subroutine:

lb=	0	calculates the value of the scalarized function
lb=	-1	calculates the gradient of the scalarized function
lb=	+1	calculates both the value and the gradient of the scalarized function
lb=	+2	calls the model and calculates lower limits for individual objectives and reference point values

lb= +3 only calls the model (it is the last call - after optimization)

Parameters:

(i) real*8 x - the array containing the point where the model has to be calculated

(i) integer lb - the value of this parameter selects a function of the subroutine (see above)

COMMON blocks:

/optk/

(i) integer kf,kgh,kbf

/optn/

(i) integer n,ng,nh

*** subroutine **vergra**

Verifies analytically calculated gradients using the algorithm described in Section 3.1.4.

Parameters:

(i) integer n - number of variables x

(i) integer ng - number of inequality constraints

(i) integer nh - number of equality constraints

(i) real*8 x - point where gradients have to be verified

(i) real*8 rk - range of changes of variables x (see the description of the subroutine **goopt**)

COMMON blocks:

/optc/

(i/o) real*8 r()

/optd/

(i) real* db,dw

/opti/

(i) integer i1

/optk/

(-) integer kf

(i) integer kgh,kbf

*** subroutine **zero**

Resets to zero value all elements of the 'n' dimensional array 'a'. It does nothing if 'n' is not positive.

Parameters:

(o)	real*8 a	-	reseted array
(i)	integer n	-	dimension of the vector to be reseted

*** subroutine **znak**

Moves data from the 'n' dimensional array 'b' into the another 'n' dimensional array 'a' changing sign of each element. It does nothing if 'n' is not positive.

Parameters:

(o)	real*8 a	-	destination array
(i)	real*8 b	-	source array
(i)	integer n	-	dimension of all arrays