GAMS TUTORIALS FOR BEGINNERS

Prepared by:
A. G. Savitsky
and
Daene C. McKinney

June 1999

For
Central Asian Regional Mission
U.S. Agency for International Development

Environmental Policy and Institutional Strengthening Indefinite Quantity Contract (EPIQ)
Partners: International Resources Group, Winrock International, and Harvard Institute for International Development

Subcontractors: PADCO; Management Systems International; and Development Alternatives, Inc.

Collaborating Institutions: Center for Naval Analysis Corporation; Conservation International; KBN Engineering and Applied Sciences, Inc.; Keller-Bliesner Engineering; Resource Management International, Inc.; Tellus Institute; Urban Institute; and World Resources Institute.

Task Order No. 813
Contract No. PCE-I-00-96-00002-00

GAMS TUTORIALS FOR BEGINNERS

Prepared by:
A. G. Savitsky
and
Daene C. McKinney

June 1999

For
Central Asian Regional Mission
U.S. Agency for International Development

Environmental Policy and Institutional Strengthening Indefinite Quantity Contract (EPIQ)
Partners: International Resources Group, Winrock International, and Harvard Institute for International Development
Subcontractors: PADCO; Management Systems International; and Development Alternatives, Inc.
Collaborating Institutions: Center for Naval Analysis Corporation; Conservation International; KBN Engineering and Applied Sciences, Inc.; Keller-Bliesner Engineering; Resource Management International, Inc.; Tellus Institute; Urban Institute; and World Resources Institute.

About the Authors

Daene McKinney is Team Leader and Senior Environmental Policy Advisor for the USAID Environmental Policy and Institutions for Central Asia (EPIC) Program in Almaty, Kazakhastan. He is responsible for the EPIC program implementation and USAID, regional and host country collaborations; overseeing all EPIC team activities. Dr. McKinney's permanent position is Associate Professor in the Environmental and Water Resources Engineering program of the Department of Civil Engineering at The University of Texas at Austin. He earned his Ph.D. in Civil Engineering with a major in Water resources Engineering at Cornell University in 1990. After working for the US EPA for one year as an environmental engineer and hydrogeologist, he joined the faculty at the University of Texas. Dr. McKinney has served as an Associate Editor of the ASCE Journal of Water Resource Planning and Management, the Chair of the ASCE Water Resource Systems Committee, a member of the Board of Directors and the International Committee of the Universities Council on Water Resources (UCOWR), and a Permanent Observer from the International Water Resources Association (IWRA) to the International Commission on Irrigation and Drainage (ICID), Aral Sea Study Program. Dr. McKinney teaches undergraduate courses in Fluid Mechanics, Numerical Methods, Hydrology, and graduate courses in Water Resources Planning and Management and Numerical Methods for Environmental Engineers. Dr. McKinney's research interests include developing and applying numerical methods for simulation, optimization, and uncertainty analysis of water resources management problems, and the development of laboratory and field experimental techniques for the characterization and remediation of aquifer and groundwater contamination.

Andrey Savitsky is a Water Resources Advisor for the USAID Environmental Policy and Institutions for Central Asia (EPIC) Program in the Tashkent office, Uzbekistan. He is responsible for mathematical modeling of optimal management of water and energy resources within the EPIC Program. Savitsky's permanent position is Lead Specialist in the Water Resources Department of the Scientific Information Center of the Interstate Coordination Water Commission (SIC ICWC). In 1986, he earned his degree of Candidate of Technical Sciences with a major in Hydraulics and Hydrological Engineering on the subject of Water and Salt Regimes of Reservoirs of the Central Asia. At the beginning of his scientific career he worked in the Hydroaerodynamics Laboratory at the Institute of Mechanics and Antiseismic Constructions for the Academy of Sciences of the Uzbek Soviet Socialist Republic. He was involved in studying dynamics of multiphase environments in industry (heat exchange systems in reactors; manufacturing of experimental samples for cumulative, non-stationary, high temperature flows of gas; mathematical modeling of multiphase interpenetrating motions of continuum). Then he joined the Scientific and Research Institute of Irrigation where he worked for a long period of time. The majority of his works and achievements are connected with mathematical simulation of dynamic and cinematic processes in rivers and reservoirs, dams in the process of demolition, in Soil – Plant – Atmosphere systems in agricultural cycles. In SIC ICWC, Savitsky's work is connected with optimization of water resources and development software programs for users. Savitsky's research interests include developing and applying mathematical methods for simulation, optimization of management of natural and artificial phenomena.

Abstract

Contents

Preface

Acronyms

This is the place to give an alphabetical list of all acronyms and abbreviations used in the document. Please note that USAID is the preferred abbreviation for the US Agency for International Development. This *EPIQ Writer's Guide for Deliverables* uses the following acronyms and abbreviations:

COTR	Contracting Officers' Technical Representative
EPIQ	Environmental Policy and Institutional Strengthening Indefinite Quantity Contract
SOW	Scope of Work
USAID	US Agency for International Development

1. INTRODUCTION

The GAMS technology is designed to facilitate the work of modelers, who create mathematical models of real processes around us. It relieves modelers of the necessity to design algorithms of models, in addition to the development of models. Thus, the major functional objective of GAMS is to design algorithms for mathematical models.

GAMS is the English abbreviation of "General Algebraic Modeling System." However, it does not mean that this system can be used only for mathematical models with algebraic equations. Those, who are acquainted with methods of solving differential equations in partial derivatives, know that the methods are always transformed to a system of algebraic equations. These algebraic equations are solved according to some iteration algorithm. That is, the applicability of GAMS is much wider. In the Tutorials, groups of small models and programs of the GAMS technology are presented. These groups allow users to develop their own models, using given examples as references.

The Tutorials is a supplement to the official GAMS Tutorials (Brooke et al., 1997), which was issued in English by GAMS Development Corporation. The Tutorials differ from the above official Tutorials only by a wider range of small user models with explanation of individual important peculiarities. These Tutorials cannot be considered even as a reduced substitute for the major GAMS Tutorials. These Tutorials can serve only as its supplement.

1.1 INSTALLATION OF GAMS ON YOUR COMPUTER

1.1.1 What Should You Have, Know and Be Good at to Use GAMS for Solving Scientific Problems

You should have:

A system capable of working in the MS-Dos mode and its characteristics should not be less than those of a standard IBM 386.

What should you know and be good at:

You should be able to state a mathematical problem correctly. You should formulate all logical interrelations of the problem and features of its behavior, specially in extreme cases. GAMS cannot think for you. It can only help you to solve some mathematical problem, stated correctly. If for some reason, the model you have established does not give corrent digital results, generally, the mistake is not a GAMS mistake. Try to find the mistake in your own model. If you have not managed to find the mistake, have a rest and try to find it again. Check your model for mistakes (the theoretical part plus presentation in the GAMS technology). However, in some cases, and in the Tutorials there is such an example, GAMS can provide you with a decision which is not optimal. That is, you should be ready for any contingencies. Although in most cases, you will get what you seek. You should know how to use the text editor in Windows or MS-Dos mode on your computer.

You can use any other editor in any other system, but the editor you use should be able to create files readable in MS-Dos environment. The editor may not have the ability to use any symbols, which characterize a font for printing, but are not displayed on the screen. For instance, the editor Lexicon is inadequate under the condition of using bold, underlined and other font. The internal editor Norton and the widespread text editor PE are very apt.

1.1.2 Sources of Information on the GAMS Models

There are two sources of obtaining such information:

1. Books and tutorials (the list is at the end of the Tutorials);
2. The internal library of the GAMS models supplied with the main GAMS package. It is in the subdirectory MODLIB. The library is created automatically as the package is installed on your computer. In the library, there are more than 150 models with necessary comments. These models are operable. All models are text files, which can be printed by any editor.

2. PRIMARY INFORMATION ON GAMS

2.1 Installing and Running GAMS for Modeling

You should install the GAMS package on your computer as follows:

You create a new directory on your hard disk to place GAMS there. Generally, this directory is called "GAMS". Therefore, all further instructions on installation will assume that you have done the same. In this directory, you copy the entire contents of your installation diskettes. Among the copied files there is (must be) the *.exe* file GAMSINST.EXE. After you have run it, you are offered a series of questions to choose SOLVER to solve your models. The authors recommends choosing MINOS. It is not the best, but it is used in the models of Central Asian specialists more often. If you have the licensed right to use the system, you answer "yes" to the corresponding question. Otherwise, you will be provided with the GAMS package, which is able only to solve small problems.

Then, you should open file AUTOEXEC.BAT and specify the path to the GAMS directory in the line PATH. If you have installed GAMS in the directory GAMS you created on the disk "C", you should add C:\GAMS in the line PATH. The model GAMS is a text file with description of a mathematical problem recorded according to the rules of GAMS technology. The text file with a GAMS model should have extension *GMS*. Running starts through the command line of DOS "GAMS name of text file". For example, GAMS model_1.gms. The models created and solved on machines with insufficient disk space and memory can be the more solved on machines of higher capacity with the same results. The GAMS package does not have (we do not know yet) any embedded specialized editors or graphical input-output. GAMS is an open system, where any user has absolute access to initial text of model. On the one hand, such openness allows scholars and modelers to develop their own models in the prompt and efficient way, freely orientating in achievements of other specialists. On the other hand, unauthorized borrowings are possible and easily feasible for back-alley people.

Actually, there are more advantages than drawbacks. To protect the copyright, publish your models and programs and present them at conferences more often. Unfortunately, you cannot use some symbols of pseudographics and Cyrillic letters in the text of models, which are supposed to be processed with GAMS. You cannot do it even in comments to the executed text of programs. Further, we shall show how you can get round this and some other constraints of GAMS.

2.2 Structure of models in GAMS

Each model in the GAMS language may have its own distinctions. It can comprise groups of alternating models, which are linked together only through a small number of common variables or parameters. However, most models have the following structure:

SETS	Structures consisting of a complex of indices or names
(indices)	Declaration of fact of existence
	Determination of relations between individual structures
DATA	PARAMETERS, TABLES, SCALARS (parameters, tables, scalars)
(data)	Declaration of fact of existence
	Determination of values of input parameters
	Preliminary computations
VARIABLES	Variables or arrays of variables
(variables)	Declaration with assigning a type of variable
	Declaration of limits for possible changes, initial level
EQUATIONS	Equations or complexes and arrays of equations
(equations)	Declaration with assigning a name
	Recording of equations on the GAMS technology
MODEL, SOLVERS	Model and methods of solution. The objective is to optimize structure
(model, solvers)	of model and method of solution.
OUTPUT	Output of information into a separate special file. Although it is not
(output)	necessary.

Besides, information is displayed on the screen under any solving circumstances. The compilation results of the model and computations are automatically sent to the special "LIST" file. This file has the same name as the file of the reference model, but a different extension LST. For example, you decided to compute a model in the file my_model.GMS. You order this in the command line GAMS my_model.GMS. Whatever happens, whether the computation is right or wrong, the file my_model.LST automatically appears. Depending on the situation, the contents of this file may differ from the given below, but the basis remains.

1. Reprinting the model
2. Map of interrelations of parameters and variables
3. Information on equations
4. Information on the solution status
5. Information on the obtained solution

Before describing basic rules of the GAMS technology, we will give brief information about some features of the language. No equation, variable or their interrelationship can be used until they have been declared in the zone of STATEMENTS. These statements can be formed in quite an arbitrary style. We will use this arbitrariness in the examples below. The GAMS technology does not distinguish between big and small letters of the alphabet. In this case, it is useful to observe the GAMS rules. Capitals are used for function words, small letters are used to form the model's own meaning. We only supplement this rule with the condition that comments are given in Russian in italics. These comments should not be available in the module carried out, but only on paper. GAMS accepts comments using only the English alphabet.

2.3 Comments in the Model and Program

There are three main methods of introducing comments and elucidation's in the body of a GAMS

model.

1. If line begins with the " * "symbol, all contents of the line are treated as a comment.
   For example, *plants (plants)
2. The program section beginning with the line $ONTEXT and ending with the line $OFFTEXT is a comment. The above functional words begin with the first symbol of the line.

   $ONTEXT;
   - - - - - - - - - -
   This guide is for a scientist who knows the Russian language. The tutorials supplement the major Tutorials for the users, who know Russian.
   + + + + + + + + + +
   $OFFTEXT;

The user can introduce comments in the bodies of GAMS declarations. In most examples, the comments are typed in a smaller font to remind the user about the necessity to change them in the working program for comments in English or any other language using the Latin alphabet. For instance, the use of State Language based on Latin in the Republic of Uzbekistan endlessly simplifies the record of comments in GAMS.

2.4 General Information on Declaration and Definition

For most structures of the GAMS technology, there are two types of statements:

a) DECLARATION – assignment of a certain value to a stated parameter
b) DEFINITION – method of its calculation through the parameters defined above or through structures of connections

Parameters and connections always require DECLARATIONS. Assignment of numerical value to a given parameter is not always compulsory. However, equations always require both the first and the second. However, DEFINITION for equations means the construction of their mathematical structures, and for parameters it means their direct computation. While fixing numerical interrelations in the body of an equation, the order of computation does not matter. There is no such notion as "first equation", "second equation" and so on. While calculating, there is no order of using an equation. In GAMS, for parameters and constants there is a method of simultaneous declaration and definition in one operator. Examples are given below.

For a series of GAMS structures, it is possible to use long names. However, it is better to limit yourself only with 8 letters for some or other notion. All notions and names of structures in the GAMS language begin with a letter and can contain digits or a limited number of other symbols (maximum possible number is shown in the examples).

2.5 Terms, Symbols and Reserved Words

Each programming language including GAMS has functional words, which have special meaning. Therefore, they cannot participate in the identification of variables and parameters. The user cannot

assign the name ASSIGN to any of the variables, because GAMS will read this word as a functional one, endure it with a special meaning, define it as improperly used and show that an error has occurred. Below, there is a list of words, which are reserved in GAMS as functional ones.

Table 1
Reserved Words in GAMS.

abort	acronym	acronyms	Alias	all	and
assign	binary	card	Display	eps	eq
equation	equations	ge	Gt	inf	integer
le	loop	lt	Maximizing	minimizing	model
models	na	ne		negative	not
options	or	ord	Parameter	parameters	positive
prod	scalar	scalars	Set	sets	smax
smin	sos1	sos2	Sum	system	table
using	variable	variables	Or	yes	repeat
until	while	if	Then	else	semicont
semiint	file	files	Putpage	puttl	free
no	option	solve	For		

Table 2
Reserved Special Combinations of Symbols and Their Meaning.

=l=	less	--	previous in cycle
=g=	more	++	next in cycle
=n=	not equal	**	involution
arithmetic	+,-,/,*		

Table 3
Symbols and Their Usage.

A to Z	alphabet	-	minus
a to z	alphabet	( )	parentheses
&	ampersand	[ ]	brackets
*	asterisk	{}	braces
@	at	%	percent
\	slash	From 0 to 9	digits
:	colon	#	number
,	comma	?	question mark
$	dollar	;	semicolon
.	period	'	quotation marks, single
+	plus	/	virgule
"	quotation marks, double_	Underlining	
=	equals	!	exclamation point
<	is less than	^	diacritic mark (over a vowel)
>	is more than		

3. ELEMENTS OF GAMS TECHNOLOGY

3.1 Sets, Indices and Constants Denominated as "SETS"

SETS is a major notion in the GAMS language. With their help, we construct all connections between variables and equations in most models. For those, who are acquainted with programming, indices can serve as equivalents of SETS. Usually, ordinals function as indices, for instance "first", "second", etc. In GAMS, indices have names, written through a combination of letters and digits, without spaces. On the one hand, these are ordinals, but on the other hand, they have names. Such duality provides for easy printing of calculation results.

In addition, SETS has a feature of the Boolean variables (truth, falsity). Boolean variables will be described in detail below. Here, we note that definition of each collection of SETS first means that the given index exists. Thus, the Boolean variable "truth" corresponds to this index. In subsets, a series of elements of the main set may not be present, and so the Boolean variable "falsity" corresponds to them. The element SET consists only of letters, digits and marks "+" and "-". It begins with either a letter or a digit, but the next symbol can be a letter, digit or marks "+" and "-". For example:

```
Jlobest         1999    1972-73
ROKETer         D6H83           Navruz-99
```

Study the next example of using SETS in the GAMS programs

```
SETS
      i  plants         / Tashkent, Almaty/
      j  consumers    / Tokyo, London, Moscow/;
```

GAMS remembers the sequence of given index notions. For instance here, Almaty is second among i, and Moscow is third among j. The group of SETS begins with the word SETS and ends with the symbol ";". It is possible to declare the same information in the model through two individual groups:

```
SETS      i  plants        / Tashkent, Almaty /;
SETS      j  consumers    / Tokyo, London , Moscow /;
```

Written declarations in the usual mathematical form look as follows:

```
i  =  { Tashkent,Almaty }
j  =  { Tokyo, London, Moscow }
```

Symbols "/" separate lists of certain elements of structures. Elements are divided by commas. It is possible to write a comment between the word, which defines structure, and the meanings of this word inside slashes. There is only one rule: all comments should fit in one row. Only values can be continued on the following rows. GAMS uses comments to form information on the model in

output forms. Therefore, comments should be short and clear. They should not contain ambiguity. Comments should be written only in English. Here is an example of a simultaneous declaration and definition of the I and J structures.

The GAMS language has great power. Therefore, models can be very big and contain an enormous number of structures. A GAMS option eases the definition of all elements of the structure. This option is shown in the example below.

```
SETS t  year    /1965, 1966, 1967,
                 1968, 1969, 1970,
                    ...   ...   ...
                    ...   ...   ...
                    ...   ...   ...
                       1998, 1999/;
```

Or a different way

```
SETS t  year    /1965*1999/;
```

For GAMS, both records are equal, but the second record is obviously shorter.

Another example:

```
SETS m machine  /mach1, mach 2, mach 3,
                  mach 4, mach 5,  ...,
                     ...   ...   ...
                     ...   ...   ...
                    ...   ...   …
                 mach 24/;
```

Or a different way:

```
SETS m machine   / mach 1* mach 24/;
```

Sometimes in models, we deal with the same collection of structures, independent of one another.

```
SETS      k  station      / Toktogul, Uchkurgan, Charvak, Chardara /;
SETS      j  station      / Toktogul, Uchkurgan, Charvak, Chardara /;
```

Simultaneous definition of the same structures under different names is important when the user wants to use their internal interrelations in the model, e.g. structures may be owned only by one country. To avoid repetition, this record can be presented in a different way, which is simple, but identical:

```
SETS      k station     / Toktogul, Uchkurgan, Charvak, Chardara  /;
ALIAS(k,j);
```

Repeating the same structures very often, you can make a mistake, which can be interpreted by GAMS as an independent structure. For example:

```
SETS      i  consumers   / Almaty , Tashkent, Fargana  /;
SETS      j  consumers   / Almaty , Tashkent, Fergana  /;
```

The user refers to three cities: Almaty, Tashkent, and Fargana. GAMS considers four cities: Almaty, Tashkent, Fergana, and Fargana. While solving big problems (hundreds of elements in structures), it is difficult to find such mistake. Therefore, GAMS has the operator ALIAS( ). This operator simplifies a record and it does not have the mistake similar to the above.

In SETS, it is possible to define subsets. For example:

```
SETS
nt nodes    /post1*post14, nurek, vahsh, pandj, tmgu,
              tald, div1, aral_sea, r_1*r_6, total /
ns(n)     supplies    / vahsh, pandj /
nr(n)     regions     / r_1*r_4, aral_sea /
nr1(n)    regions     / r_1*r_6 /
nl(n)     lakes       / nurek, tald, TMGU /
rlast(n) last user    / aral_sea /
```

In the above example, *nt* is a collection of names, which simply make up a list of objects to be computed. Although GAMS remembers the sequence of their appearance, but in this case it does not matter.

This collection of names belongs to the group of manifold physical objects. Among them, there are water sources (supplies), regions (regions) and reservoirs (lakes). There is also a set, which is a subset of a subset. The Aral Sea (*aral_sea*) is one of flow consumers. Its capacities will be underlined in the model. So, *ns(n)* – is a subset of flow sources. Elements of the common set /*vahsh,pandj*/, included in the given subset, are listed within slashes. The same should be said about the list of consumers *nr(n)* and reservoirs *nl(n)*. The Aral Sea is a component of the subset of consumers *nr(n).* At the same time, it is included in the individual special subset *rlast(n).* The last record can be substituted for the equal one.

```
rlast(nr) last user  / aral_sea /
```

Besides, it should be clear that the subset nr( ) does not include all elements of the common set n, but only a part of them. Therefore in Boolean variables

```
nr('aral_sea')    = YES ;   (truth)
nr('nurek')       = NO  ;   (falsity)
```

Reference of the SET name in single quotation marks in some subset is applying for a specific element of the set.

3.2 Operations with SETS

With SETS, it is possible to perform different operations using symbols

"+", "-", "*", "not"

Symbol " + " performs the operation of a uniting set. Consider the following example:

subitem3(item) = subitem1(item) + subitem2(item);

The set subitem3(item) is the set equal to all elements subitem1(item) and all elements subitem2(item). The above operation is equal to the following method of using Boolean variables. This method is longer.

SUBITEM3(ITEM)=NO; SUBITEM3(SUBITEM2)=YES; SUBITEM3(SUBITEM1)=YES;

For our case, all water objects can be united in the structure of water objects by the following operation:

```
SETS   water(n);
       water(n) = nl(n) + ns(n);
```

For all sources and all lakes, SETS water(n) will exist and be defined by the Boolean variable YES. In the rest of the cases, SETS will not be available, and the Boolean variable NO will correspond to it.

Symbol "*" performs the operation of set intersection; only the elements included in both A and B, belong to the intersection of the sets A and B. Consider the following example:

Subitem3(item) = subitem1(item) * subitem2(item);

Membership of subitem3(item) is a set consisting of the elements, which are both in subitem1(item) and in subitem2(item).

In our case, it is possible to distinguish a subset consisting only of the elements "r1", "r2", "r3", and "r4".

```
SETS      simple(n);
simple(n) =nr(n) * nr1(n);
```

The symbol "NOT" performs the operation of a set complement. Consider the following example:

subitem3(item) = NOT subitem1(item);

Membership subitem3(item) is a set consisting of elements included in item, but not available in

subitem1(item). That is, all elements in subitem1(item) are deleted in subitem3(item). The above operation is equal to a longer method:

```
subitem3(item) = yes; subitem3(subitem1) = no;
```

In our case:

```
SETS real_reg(n);   real_reg(n) = nr(n) not nlast(n);
```

and real_reg(n) will consist of "r1", "r2", "r3", "r4"; only for them, SET will be defined by the Boolean variable YES.

With the help of symbol "-" we can form a complement of sets or a set, which is a difference between the sets A and B. This set consists of elements, which belong to A but not B. Operator "-" performs the operation of difference of sets. Consider the following example:

```
subitem3(item) = subitem1(item) - subitem2(item);
```

The subset subitem3(item) is a set consisting of elements, which are members of subitem 1 (item), but they are not members of subitem 2 (item). The above operation is equal to a longer method:

```
subitem3(item) = yes$(subitem1(item); Subitem3(subitem2) = no;
```

The symbol "$" here should be considered as a logical operator, which states the condition on the right-hand side as a condition of the equality. This symbol is not chosen at random. The rules of using this symbol will be described below in the paragraph “Dollar Operations in GAMS”. For our example:

```
SETS  pool(n);
        pool(n) = nr1(n) -nr(n);
        end('r5') = yes;
        end('r6') = yes;     and for all other "n",  end(n) = no;
```

Subsets can be defined with the help of Boolean variables as well. This is shown by the example below.

```
SET item        /cat, eagle, tiger, lion, fly, man /
     subitem1(item)  (first  subset of item)
     subitem2(item)  (second subset of item);
     subitem1('fly') = yes;  subitem1('eagle') = yes;
     subitem2(item)  = yes;  subitem2('man')   = no;
     display subitem1, subitem2;
```

The output of the program can be read at the end of listing *.LST, which appears automatically.

```
COMPILATION TIME     =        0.010 SECONDS        VERID WAT-25-091
GAMS 2.25.091  DOS Extended/C       03/16/99 12:54:12  PAGE      3
```

```
  General   Algebraic   Modeling   System   Execution
----     8 SET       SUBITEM1      (first  subset of item)
EAGLE,   FLY
----     8 SET       SUBITEM2      (second subset of item)
CAT ,    EAGLE,    TIGER,    LION ,    FLY
EXECUTION TIME      =        0.010 SECONDS        VERID WAT-25-091
```

3.3 Defining Multivariable Indices

Multidimensional SETS differ from the unidimensional ones only by the following:
Elements in multidimensional SETS are defined by indices (unidimensional SETS) divided by commas. The example below illustrates the use of two-dimensional SETS.

```
SETS n  /
     SIM_N__1,OUT_N__1,CTR_N__1,USE_N__1,
     SIM_N__2,OUT_N__2,CTR_N__2,USE_N__2,
     SIM_N__3,OUT_N__3,CTR_N__3,USE_N__3,
     SIM_N__4,OUT_N__4,CTR_N__4,USE_N__4,
     SIM_N__5,OUT_N__5,CTR_N__5,USE_N__5/;
ALIAS(n,n1);
SETS n_to_n(n,n1) subsystem of DIVERSIONS /
    SIM_N__1. USE_N__1,  SIM_N__2. USE_N__2,
    SIM_N__3. USE_N__3,  SIM_N__4. USE_N__4 /;
SETS n_n_n(n,n1) subsystem of out_flow /
    CTR_N__3. OUT_N__2, SIM_N__4.
    OUT_N__1, SIM_N__5. OUT_N__3/;
SETS n_to_nr(n,n1);
    n_to_nr(n,n1)=n_to_n(n,n1)+n_n_n(n,n1);
```

Consider a multidimensional SET. It can include not more than 10 identifiers-elementSET. In GAMS, it is recorded as follows:

We have unidimensional SET for each nameSET, where all elements SET are specified (see above). Then, an additional SET is made up (in GAMS this is called a “MANY-TO-MANY MAPPING”). In this SET, are specified what elements SET of each nameSET constitute the multidimensional SET. Syntax SET “MANY-TO-MANY MAPPING” complies with the rules presented in the table. The rules allow achieving a considerably reduced introduction of multidimensional SETS.

Structure	Result
(f,b).c.d	f.b.c, f.b.d
(f,b).(c,d).e	f.c.e, b.c.e, f.d.e, b.d.e
1*3.1*3.1*3	1.1.1, 1.1.2, 1.1.3, ..., 3.3.3
(f.1*3).c	(f.1, f.2, f.3).c or f.1.c, f.2.c, f.3.c

Example:

```
SETS i  /f, b /
     j  /c,d,e /
     ij1  /f.c, f.d /
     ij2  /f.c, b.c /                  or  ij2  /(a,b).c     /
     ij3  /f.c, b.c, f.d, b.d /;       or  ij3  /(f,b).(c,d) /;
```

4. INITIAL DATA: SCALARS, PARAMETERS, TABLES

4.1 General Rules Applicable to Scalars, Parameters and Tables

Major working elements of the GAMS technology are identifiers: parameters and marks.

Table 4
The rules of making up identifiers and marks.

	Identifiers	UnquotedLabels	Quoted Labels
Number of Characters	10	10	10
Must Begin With	A letter	A letter or a number	Any character
Permitted Special Characters	None	+or - characters	Any but the starting quote

GAMS parameters are introduced based on free format. Operators can be placed anywhere on the row. On one row, you can also place several operators. On the other hand, an operator can take several rows, as it is shown below.

```
Parameter;
Parameter;
Parameter; parameter; parameter;
```

The words, you are reading now, are a long parameter, which takes two rows. An example of inadmissible identifiers:

```
15 $rhsoie moi          ewoifkoioiiio  ueru&wc
```

Digital data are contained in arrays (zero, scalar, or multidimensional indices). The SETs, described above, play the role of indices. As an index, SET is present everywhere except zero arrays.

To declare an array, we use functional words: SCALAR (zero-dimensional), PARAMETERS (unidimensional), and TABLE (multidimensional). Assignment of numerical values can be carried out in two ways:

1) Assignment
2) Computation.

4.2 Data Entry Through Assignment at the Moment of Definition

In the following example, we show a way to determine numerical values in a GAMS model. Parameter L is supposed to be used in the model as a zero array. The operator SCALAR with simultaneous assignment can introduce its value.

```
SCALAR  L  size   /0.99/;
```

The value 0.99 is assigned simultaneously with declaration (statement of fact of existence) of the L value. The sequence of declaration and definition of numerical values for unidimensional arrays is a bit different. The example below illustrates this brightly:

```
SETS       i   consumers   / Almaty, Tashkent, Fergana   /;
PARAMETERS  a(i)  orders    /  Almaty  7350
                               Tashkent  7250
                               Fergana  7170  /
                   b(i)  supplies  / Almaty 7250, Tashkent 7180
                               Fergana 770  /;
PARAMETERS  c(i)  price  / Almaty 5, Tashkent  21.80, Fergana 1.70 /;
```

In this example, declaration and definition are available simultaneously

The following example illustrates data entry in the tabular form at the moment of definition.

```
SETS p  types of crops / cotton, wheat, rice /;
SETS t  season          / winter, spring, summer, fall /;
Table w(p,t)  comments on the variable w
            cotton   wheat  rice
winter                 1
spring       20        23
summer      60        10       70
fall           0.01
```

If some cells are missing in the table, it means that the value of the array for the corresponding complex of indices is not defined. However, if this element of array is used in further computations, it will be considered as equal to zero.

GAMS precisely determines which digit belongs to which element of an array. If the maker of the program can mentally draw horizontal and vertical lines to create precisely expressed cells containing only one digit or a name of one symbol, (s)he will have the same notions and views GAMS has. For example, GAMS can read a table of the following type incorrectly.

```
Table w(p,t)  comments
cotton   wheat  rice
winter                 1
spring       20        23
summer      60        10       70
fall           0.01
```

Consider the following example. OLA is a four-dimensional array. It has four index positions. It is initialized when the format of special type shown below is used.

```
SET   i  first index          /first, second /
        j  second index       /one, two, three /
```

```
      k  third index          /m, n /
PARAMETER OLA(i,j,k)   a 3 - dimensional structure  /
first.one.m          320.0,   first.one.n           96.0
second.one.m          40.0,   second.one.n         967.0
first.two.m           25.0,   first.two.n          679.0
second.two.m         420.0,   second.two.n          67.0
first.three.m        725.0,   first.three.n        560.0
second.three.m       426.0,   second.three.n       645.0   /;
```

4.3 Computation of Parameters

Parameters are computed under the condition that declaration occurred above based on the text of programs. Determination of data through direct computation or assignment following the initial definition (it may follow some other calculations) is possible and sometimes useful for the solution of some interim models. The assignment is most clear from the following example:

```
PARAMETERS c(i,j) price of transporting a unit of products;
c(i,j) = f * d(i,j)*a.L;
```

In the example, we consider the scalar f, two-dimensional array d(i,j) and even the variable a (special notion described in the next paragraph) as known and previously defined.

It is assumed that the value of *a* is calculated based on some previous optimization model. You should pay attention to the lack of any instructions for what indices of i and j, these computations should be carried out. GAMS will carry out these computations for all indices i and j, which are defined above in the body of the program in the operators SETS. If you need to assign or compute a specific value of one or a small group of elements of the array, you should write this computation as follows.

```
c( 'Tashkent', 'Fergana' )  =  f * 0.001 ;
c( 'Tashkent', 'Chinaz' )  = 20 * 0.001 ;
```

To organize computations within some limits of the variables i and j, you should use subsets. Methods of defining subsets are described in Section 3.

Besides, you can use the special logical operator "$", which is conditional. Features of this operator and rules of using it are described below. Computations of parameters (of all dimensions) should be consistent, as it is accepted in ordinary programming languages. A collection of standard functions, shown in Tables 5, 6, and 7 below, is at your disposal in the GAMS compiler. Table 6 reflects the results of using logical operators. Classification of all GAMS Functions is given in Table 7

Table 5

Logical Operators.

Operators	Description
Not	not
And	and
Or	including or
Xor	excluding or

Table 6
Generalized Results of Using Logical Operators.

Operands		Results			
A	B	A and B	A or B	A or B	not A
0	0	0	0	0	1
0	not zero	0	1	1	1
not zero	0	0	1	1	0
not zero	not zero	1	1	0	0

Table 7
Functions of GAMS.

Functions	Description	Classification	Exogenous	Endogenous
abs	absolute	heterogeneous	legal	DNLP
arctan	arc tangent	homogenous	legal	NLP
ceil	homogenous	legal	illegal	
cos	cosine	discontinuous	legal	NLP
errorf	error	homogenous	legal	NLP
exp	exponent	homogenous	legal	NLP
floor	minimal	discontinuous	legal	illegal
log	natural	homogenous	legal	NLP
log10	decimal	homogenous	legal	NLP
mapval	function	discontinuous	legal	illegal
max	maximum	heterogeneous	legal	DNLP
min	minimum	heterogeneous	legal	DNLP
mod	residue	discontinuous	legal	illegal
normal	normal	illegal	illegal	illegal
power	whole	homogenous	legal	NLP
round	rounding-off	discontinuous	legal	illegal
sign		discontinuous	legal	illegal
sin	sine	homogenous	legal	NLP
sgr	squaring	homogenous	legal	NLP
sqrt	square root	homogenous	legal	NLP
trunc	truncation	discontinuous	legal	illegal
uniform	uniform	illegal	illegal	illegal

** If there are several arguments, classification is given only for the first argument. The second argument should be integer or usually constant.

In GAMS program calculations, the user can use all functions shown in Tables 5, 6, and 7.

Examples of using assignment operators are given below

```
ckoren = SQRT(c);
dkhvny = m * Ckoren;
w          = SIN(l)/EXP(-lambda);
ecw(i)   = SQRT( 2 * cotton(i) + woda(i) /zemla(i) );
t(i)        = MIN( p(i)*qq(i)*SIN(r(i)), LOG(s(i)) );
enver(i,j) = SQRT(SQR(x1(i) - x1(j)) + SQR(x2(i)-x2(j)) );
baht(j)    = non(j) * EXP( -tinchlik*vakt(j)*soglik );
```

Besides those listed above, two special functions are available for the user: SUM(..., ...) and PROD(..., ...).

```
e = SUM(i,a(i));
```

This record means that the value of "e" will be equal to the algebraic sum of all elements of the unidimensional array "a( )" on the entire range of indices "i".

```
t(i) = PROD(j,b(i,j));
```

This record means that in the unidimensional array "t( )" each "i" element will be defined as a product of elements of the two-dimensional array "b( )" on "j" with the fixed related index “i”.

The description of how functions act is given in the examples below.

5. VARIABLES

5.1 Types of Variables and Their Declaration

Variables are the major object, with which GAMS operates and which it searches and defines. To define the list of variables we use the functional word VARIABLES.
There are five types of defining variables:

VARIABLES (free type)	Variables belong to the entire set of real numbers from "minus" infinity to "plus" infinity
POSITIVE VARIABLE	Variables belong to the entire set of positive real numbers from zero to "plus" infinity
NEGATIVE VARIABLES	Variables belong to the entire set of negative real numbers from zero to "minus" infinity
BINARY VARIABLES	Variables can take a value of 1 or 0
INTEGER VARIABLES	Variables belong to the entire set of positive integers (0, 1, 2, 3, 4 …..)

Variables (as well as arrays of initial data defined in Section 4 with the help of functional words SCALAR, PARAMETERS, TABLE) can be zero-dimensional and multidimensional. Their dimensions are determined by the availability and number of indices. The declaration of variables is based on the following principle. For example:

```
POSITIVE VARIABLES
        BOCHKA ;  zero-dimensional positive variable

VARIABLES
        OBJ, STAKAN    zero-dimensional variables
        KVAS(i)        unidimensional variables
        COST(i,j,k)    multidimensional variables;
```

Variables are included in equations as well as both parameters of structure and constants, forming with them model equations. Variables are always defined in the solution process. However, users and modelers are twice faced with variables, when they make up a model. For the first time, they face with variables while declaring them (it is described above) and for the second time after the model block of equations.

To avoid breaking the description of variables, stop on the block of defining initial and boundary conditions for variables. This block is always located between the block of equations and a solver (SOLVERs in GAMS).

5.2 Boundaries of Variables, Initial and Fixed Values

Considering types of variables, the user sees, that some variables have the allowed boundaries of their existence. For instance, if some variables are declared as positive, then "zero" is the lower boundary of their possible existence. The same boundary remains for negative variables, but "zero" is their maximum possible value. In addition, the user has a possibility to specialize a possible area of search of variables with the help of special records. These records can be treated as peculiar suffixes of variables.

Table 8
Main Suffixes.

Suffix	Description
.L	Level or a boundary (edge) value
.LO	Lower boundary
.UP	Upper boundary
.FX	Fixed value
Additional suffixes	(you can read about them in the major GAMS Tutorials)
.m	Boundary (edge) or dual value
.prior	Priority
.scale	Scaling

No suffix, which defines boundaries, is compulsory. The lack of this suffix in the model will be treated by GAMS as its presence, but within default sizes for each type of variables. For example, it is clear that the lower boundary of positive variables is zero. It means that if c(i,j) is declared as a positive variable, the record c.lo(i,j)=0 is just redundant.
The search of variables always starts from a boundary of a modification area, most often from zero. However, users and modelers should remember that zero is a dangerous value, which should be avoided, if possible. The record c.lo(i,j)=0.0000001; will not distort the solution very much, but will avoid some difficulties. Very often in the models one can see degenerated equations for zero values of variables, or solutions "adhered" to a boundary, and GAMS SOLVERS failed to search optimality in the model.

Examples show best how suffixes act.

```
a.LO(i,j) =0.1;    a.UP(i,j) =7.7;  a.L(i,j) =3;
```

Lower possible boundary in the search of variables a(i,j) is 0.1. Upper boundary of variables a(i,j) equals 7.7. The search of variables for all indices i and j will begin from 3.

```
b.fx(i) =9.1;
```

Although variable b(i) is preset for the model as a variable, its role in the systems of equations will be similar to the role of parameter, i.e. it will be fixed and always identical to 9.1.

The more the user constricts boundaries of possible variable modification, the sooner (s)he will

obtain a solution. However, you should not constrict boundaries of search too much, because in complicated models you may have the situation, where there is no area of searching solutions at all. Namely, there may be systems of equations, in which one variable is within the boundaries allowed by the user, but the second variable is certain to be in the unallowed area. Usually, the consequences of calculating such systems are grave.

6. EQUATIONS

The block of equations in the GAMS models consists of two parts:

- Declaration of names of equations
- Equations

6.1 Declaration of Equation Names

Declaration of equation names is similar to the declaration of SET or PARAMETER. The similarity is in the fact that the list and comments are allowed and recommended. Several equations can be declared in one row separated by commas. The syntax of declaring names of equations is as follows.

```
EQUATION(S)   nameEQN  comment ;
```

EQUATION(S) – a functional word, which can be found each time before the name of an equation if there is semicolon at the end. Or it can be found only one time at the beginning of the list of equation names. Commas separate different names of equations in the row or one row is assigned for each name.

The nameEQN can include two parts: identifier of the name of equation and indices in parentheses. The identifier of names includes not more than 10 symbols and always begins with a letter. Indices can be either letters or digits. They represent SETS. Comments may not have more than 80 symbols and should be at the row where the identifier is located. For example:

```
SETS  q  clever         / vasia, kolia, misha, sasha /
      s   very clever  / first, second /;

EQUATIONS
   differ           discrepancy level of income and abilities in general
   silno(q)         qualification
   nadbavka(s)   bonus
   slabo(q)         salary ;
```

Declaration of the first equation goes after the key word EQUATION. Declaration of the equation begins with the name of the equation (differ), then the text follows (level of discrepancy). Equation differ is a scalar equation.

Equation slabo is declared through SET q (4 terms). Equation nadbavka(s) is declared through SET s (2 terms). This gives 6 different equations, proceeding from independent combinations s and q. You should remember that in the given case 6 equations have been reserved.

The role of indices in enumeration of names of equations is a bit different than in definition of elements of arrays. Here, it would be more correct to call them the indices of the managing

computations, or the definers of the number of computed equations. Moreover, earlier all indices were already defined as SETS.

However, the most important thing is that the clear coordinated relationship should exist between isolated items of equations and indices in the names of equations. This relationship will be described below.

6.2 Definition of Equations Structures

Definition of equations structures is a mathematical peculiarity of record equations in the GAMS language. The syntax for defining equations in GAMS is as follows.
nameEQN..

algebraic expression // special relation character //algebraic expression;

First, we have the nameEQN (name of the equation) was shown in the declaration of equation names, then two periods ".." follow which always separate name of the equation and the algebraic expression. Expression of equation structure is a record of algebraic equation using allowed algebraic operations. It is possible to use the embedded functions like SIN( ), EXP( ), shown in Tables 5, 6, and 7. The user should carefully follow the equality of the number of open and closed brackets and other usual details, with which any program engineer, using FORTRAN or similar languages while recording equations with the complexes of assignment operators, may run across.

However, the difference is that the body of the equation should contain a special relation character, such as:

=E=	right part is equal to the left
=G=	right part is greater or equal to the left
=L=	right part is less or equal to the left
=N=	there is no forced relationship between left and right parts (This type of equations is rarely used)

Equations can occupy an unlimited number of rows. Spaces can be used for more obvious and convenient comprehension. The equations once defined cannot be changed or redefined. If such necessity arises, it is essential to introduce a new equation with a new name. This name should be in the declaration part of names of equations.

However, it is possible to change the number of estimated elements of the equation through the change of data, which the equation uses. On the other hand, you can exclude a part of equations on logical condition of relationship between SETS and PARAMETER.

For the last action, we use an special mechanism. Detailed description of the mechanism is given in a chapter below. This is the HANDING MECHANISM ($ operator) introduced in the name of equation or existing in the computation part of the equation.

Here is an example elucidating the description of scalar equations.

```
VARIABLES   phi, phipsi, philam, phipi, phieps;
EQUATIONS  obj;
obj..  phi  =e= phipsi + philam + phipi - phieps;
```

Other mathematicaly equivalent equations may be used:

```
obj..  phi - phipsi =e=  philam + phipi - phieps;  variant 1
obj..  phi - phipsi - philam =e=  phipi - phieps;  variant 2
obj..  phi - phipsi - philam - phipi =e=  phieps;  variant 3
```

The order of elements in the equation is not important, but practice shows that often one identical record of the same equation excels another identical record in the speed with which a solution may be found (more often it is encountered in nonlinear problems).

Scalar equations are most often encountered while making up optimization models. The examples written above are only scalar equations. They contain only scalar variables. However, scalar equations can contain index variables as well, where index operators of peculiar convolution by index are used. For example:

```
pulat..   pontij =E=  SUM(j,x(j));
```

We can see, that the name of the equation is scalar, the left part is scalar, and the right part has the index SET "j". Consequently, after summing the index disappears and eventually the right part becomes scalar as well. You should observe correspondence between the dimensions of the name and left and right parts of equations. The record of the following form is deeply erroneous

```
pulat(m)..   pontij(k) =e=  SUM(j,x(j))+ y(i);
```

Whatever addition you would delete, you will get the message that there is the uncontrolled index (case k, i), or that the equation does not depend on the controlled index (case m).

Index equations differ from scalar equations only by a closer correspondence between dimensions of equations of left and right parts.

Indexes SET, which take the position to the left of ".." and immediately follow the name of an index equation, can be considered as control indexes, because the index equation standing to the right of ".." will be solved for all possible variants of indexes.

Consider the following example of a simple index equation, which indicates the collection of kindred equations

```
sag(k)=e=mag(k)+dag(k)*nag(k)
```

having the name dg in the complex. It will be solved for the whole collection of SETS "k"

```
dg(k)..  sag(k)  =E=  mag(k) + dag(k)*nag(k);
```

Let k have three terms, so it is possible to input three constraints on each of four variables

```
sag.LO(k)=1.0;   sag.UP(k)=6.0;   dag.LO(k)=5.0;   dag.UP(k)=8.0;
mag.LO(k)=3.0;   mag.UP(k)=7.0;   nag.LO(k)=2.0;   nag. UP(k)=9.0;
```

GAMS allows the use of names and structures of big equations. However, the fundamental rule is the existence in the model of at least one scalar variable and one scalar equation. The GAMS SOLVER allows optimizing only on a scalar variable. In the following example, this variable is ddd in the equation fff.

```
SETS  j  / a1, a2, a3 /;
SETS  h / b1, b2 /;
ALIAS(i,j);
PARAMETERS  maxim(j,h);
PARAMETERS  minim(j);
        maxim(j,h)= 5;  minim(j)= 1;
VARIABLES
        mig(j,h), big(j,h), dog(j,h), man(j,h),
        sensay(i,h), bird(i,j)
        ddd
        ;
EQUATIONS
        fig(j,h), dan(j,h)
        fff;
fig(j,h)..  mig(j,h)   =E= big(j,h) - dog(j,h) + minim(j) ;
dan(j,h).. man(j,h)  =L= PROD(i,sensay(i,h)*bird(i,j)) - maxim(j,h);
fff..         ddd        =E= SUM(j,(SUM(h,   (mig(j,h)+man(j,h)))));

$ONTEXT;
If we substitute the related lines in the model for the lines below, we also obtain a solution,
but it is difficult to apprehend its physical sense.

 fff(j);
 fff(j)..   minim(j) =E= SUM(h,(ddd*mig(j,h)+man(j,h)));

$OFFTEXT;

mig.LO(j,h)      =  minim(j);     mig.UP(j,h)     =  maxim(j,h);
big.LO(j,h)      =  minim(j);     big.UP(j,h)     =  maxim(j,h);
dog.LO(j,h)      =  minim(j);     dog.UP(j,h)     =  maxim(j,h);
man.LO(j,h)     =  minim(j);     man.UP(j,h)    =  maxim(j,h);
sensay.LO(i,h) =  minim(i);     sensay.UP(i,h) =  maxim(i,h);
bird.LO(i,j)      =  minim(i);     bird.UP(i,j)      = SUM(h, maxim(i,h));
MODEL andre /ALL/;
SOLVE andre USING NLP  MINIMASING ddd.
```

6.3 Explicit Use of Symbols of Indexes in Equations

Very often, it is necessary to use symbols in equations. This can be done using quotation marks, as in PARAMETER. Consider the following example:

```
dz.. tz =E= y('jan') + y('feb') + y('mar') +y('dec') +y('nov') +y('oct');
```

We can see that the equation will be performed only for a nonvegetation season. In this instance, SET in the variable y(t) is defined for all twelve months. However, GAMS is a very flexible system and the same system can be recorded if a subset for the index SET has been created.

```
SET tt       month /jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec/;
SET to(tt)  month /jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec/;
SET t(tt)    month non vegetation /jan,feb,mar,oct,nov,dec/;
SET v(tt)   month vegetation;
      v(tt) = to(tt) - t(tt);
... ... ... ... ... ... ...
VARIABLE
... ... ... ... ... ... ...
EQUATION
... ... ... ... ... ... ...
dz.. tz =e=SUM(t, y(t));
... ... ... ... ... ... ...
... ... ... ... ... ... ...
```

6.4 Prevention of Situations Related to the Areas of Functions Existence

Making up equations, the modeler or the user should carefully follow whether in equations of models there are structures, which do not exist under some possible values of variables. For instance, the function LOG does not exist if its argument equals 0. Dividing by 0 is another example, you can see it in the expression x = E = 1/y. Usually, the attempt to solve such equations causes an error and calculations are automatically reset.

One of the ways of preventing such situations is artificial liquidation of dangerous situations through setting the boundaries of possible change of variables. Consider the following example:

```
boss..  main =E= SUM(y,alfa(y)*LOG(c(y)));
c.LO(y) = 0.1;
```

Constraint for c(y) will not allow emerging the situation where the function LOG(c(y)) cannot be defined.

6.5 Assembling Model from a Set of Equations

After the modeler or the user has declared SETS, PARAMETERS, VARIABLES, EQUATIONS, (s)he can form a specific system from a collection of equations and give it a specific name.

Two situations are possible:
1) All created equations are included in the model
2) Only a part of the total number of equations is included in the model

In the first case, the syntax of declaring the model is as follows:

```
MODEL    nameModel (comment in English letters) /ALL/;
```

E.g.:

```
MODEL  traffic     "transport system"  /ALL/;
```

The model is called "traffic", and the key word ALL means that the model includes all (declared and created) equations.

For the second case, the user can create numerous variants of equations and unite them in groups, forming different models with distinct names and they may have various optimization objectives.

The following example contains the solution of three versions: linear, nonlinear and a special version named “expenses”. The models are distinguished by different collections of equations.

```
MODEL   nortonl  "linear version"          /far, terc, wdfl, qsc, obj/
        nortonn  "nonlinear version"     /far, terc, wdfn, qsc, obj/
        nortone  "expenditure version" /far, terc, wdfe, qsc, obj/ ;
```

where "far, terc, wdfl, ..." are names of equations, a part of which may be index equations.

6.6 Setting Goals in the Solution of Models

A goal is set by the following composite operator, which sets a goal and defines a way to achieve it.

```
SOLVE andre USING DNLP MINIMIZING o;     or
SOLVE andre USING DNLP MAXIMiZING o;
```

The first word is unchangeable. It is compulsory in the expression. The second word is the name of the model, which is given by the user or the modeler. The way to input the model name is in the MODEL operator and this is described in Section 6.5. The third word is unchangeable and compulsory. It precedes the appearance of the fourth word, which defines the way of solving the model. The fourth word can be MINIMIZING (minimizing) or MAXIMIZING (maximizing). The

fifth word is the name of a scalar variable, in relation to which either maximizing or minimizing is carried out. The fourth and fifth words may be missing, as in the model wallmcp.127, included in the internal library of GAMS in the directory MODLIB (see line Solve wall using mcp;). The third word, which defines a way of solving the model, has the widest variety. Below, we present the ways of solving models, which are built in the GAMS technology.

Table 9
Methods of Solving Optimization Problems.

Ways to Solve Models	Description
LP	Linear programming. The model cannot contain nonlinear or discrete (binary and integer) variables.
NLP	Nonlinear programming. In the model major nonlinear forms are only continuous functions. However, the model does not contain discrete variables.
DNLP	Nonlinear programming with discontinuous derivatives. This model can contain heterogeneous functions. The solution of this problem is more complicated than NLP. We recommend users to avoid such type of solving models.
RMIP	Relaxed mixed integer programming. This way can contain discrete variables, but discrete requirements are not stringent. Integer and binary variables can take any values within boundaries.
MIP	Mixed integer programming. It is similar to RMIP, but the requirements to discreteness of variables and equations are stringent. Discrete variables should take discrete values within boundaries.
RMINLP	Relaxed mixed integer nonlinear programming. The model can contain both discrete variables and major nonlinear forms. Discrete requirements are not stringent. This class of problems as for solution complexity is like NLP
MINLP	Mixed integer nonlinear programming. The same characteristics as for RMINLP, but the requirements to discreteness are very stringent.
MCP	Mixed Complementary Problem
CNS	Constrained Nonlinear System

6.7 Requirements of the User to the Model

The following example demonstrates the use of suffixes in models to specify some additional actions.

```
MODEL  transport /ALL/;
```

```
transport.RESLIM + 60;
```

In this example, the time limit to solve the optimization problem is 60 seconds. Based on the operator SOLVE, GAMS calls one of available ways to solve individual models (Table 9). There are benefits from the use of several SOLVE statements in the program. Several SOLVE statements can be used in one program and executed successively. Sometimes, thereby it is possible to solve complicated problems, which cannot be solved with one operator SOLVE. The best solutions found in the first SOLVE are used in the next SOLVE. Very likely, the following one uses some new trajectory, and it leads to the solution.

```
SOLVE andre USING DNLP MAXIMIZING o;
SOLVE andre USING DNLP MAXIMIZING o;
SOLVE andre USING DNLP MAXIMIZING o.
```

7. DISPLAY OF INFORMATION AS TEXT FILES

7.1 Declaration of File Name for Output Information

GAMS allows displaying the information about the model and its solution as text files. We should remind you, that the text output file my_model.LST is always present. File *.LST is created automatically. For purposeful display of information in this file, the special operator DISPLAY is used. You can best appreciate this operator by studying it in examples (Example 4 in the Appendix). You should use this operator only when adjusting the program, because in this case it is possible to display the information you are interested in the quickest way. However, there is an opportunity to create additional output files. The following is the full syntax for defining output files:

FILE nameFail comments / internal namefile /

where: FILE is a keyword used to define files;
nameFile is a internal filename used in models of GAMS to direct output to the internal text file.

In a declaration of a file, internal filename and comments are not compulsory. In case of lack of a filename, GAMS provides an internal filename by default and this filename is FNAME.PUT. GAMS allows working with a set of output files, but you should read about it in the main GAMS Tutorials of the GAMS Development Corporation.

Consider the following example:

FILE out_file
out_common is a declaration of the special external file named /out.txt/
CON the monitor screen is treated as an output file;

The first output file is named out_file in the model. It directs information by default into the file out_file.put. The second output file is named out_common. GAMS directs the information to the certain internal file out.txt. The last case: the special internal name CON defines the monitor screen as an output file, to record output on the display in the DOS system. The record on the display may be useful, because it informs the user about different details of computation of the created model.

7.2 Output of Information (Transcription Mode and Addition Mode)

The operator PUT is used for both assigning the name of current file to some declared file and recording data elements in this file. Full syntax of the operator PUT is as follows:

put nameFile_1 elements nameFile_2 elements ... nameFile_3 elements;

As it is shown in the above syntax, the set of output files can be successively recorded in one operator PUT. However, you should not forget that the current output file is only one file. After the

elements have been sent to the output file, the next internal filename and the name of the current (working) file is reassigned. This reassignment is based on the next internal filename in the operator PUT. In the operator PUT, the last internal name remains as a current name, until the following operator PUT uses a new internal name. When the program completes its work, all files are closed by default. Consider the example:

```
FILE res /result.dat/
PUT res;
PUT " ATTENTION !!!              "/;
PUT n.TL /;
PUT x.L:8:2 /;
PUTCLOSE  res;  (details are specified in paragraph 7.4)
```

This example shows how the file result.dat was correlated with the internal name res. In the second line, the output information was directed into the file res, which was made as a working file. In the third line, we can see the way of outputting text comments into the file. In the fourth line, we can see the way of outputting SET in the form of literal names. The fifth line contains the output of digital value of the variable x in the format of 8 digits, 2 of which are fractional. The last line closes the file res to take the information out. This example is given to show the reader two new important peculiarities of the information output, namely the suffixes TL and L. Suffix TL is used for indices SET. Literal names of indices will be taken out into the file. Suffix L is used for variables, which were defined in the process of solving the model with the help of the GAMS compiler. For output, the file is re-recorded every time and the old information in it is deleted (if there were no special directions described in paragraph 7.3).

7.3 Adding New Information to an Already Existing File

PUT is a multiobjective operator. It can add or rerecord an existing file. The functional suffix of the filename.AP determines, what operation takes place. By default, SUFFIX takes the value of 0. This means re-recording of the existing file, whereas the value of 1 means an addition to the file. Consider our already existing file out.txt. The use of the next operator adds output information to it without deleting the information this file already contains.

```
out_common.AP = 1;
```

However, if the file does not exist, it should be created in advance.

7.4 Closing Output Files

The keyword PUTCLOSE is used to close the file before completion of the GAMS program (When the program is completed all output files are closed automatically.). The syntax of closing files is as follows:

PUTCLOSE filename elements;

where: filename is an internal name of the file, which should be closed;
elements is the last output of information in the file, before it is closed.

If the internal filename is missing in the operator PUTCLOSE, the current file will be closed. It is necessary to note that after the use of the command PUTCLOSE, the file cannot be newly redefined to use it again.

8. CONDITIONS FOR DOLLAR OPERATIONS ($)

8.1 General Statements

This section is about the dollar operator ($), which is one of the strongest features of the GAMS compiling program. Dollar operator provides operations with logical conditions. The term $ (condition) can be read " if the condition is fulfilled", where the condition is of logical character capable of acquiring Boolean values described in the text above.

Logical dollar conditions should not contain variables (absolutely forbidden). But the usage of suffixes of variables, like L. and .M, allows using dollar operations. But it should be noted that the usage of the suffixes transfers the variables into constant values. It means that it is impossible to use logical dollar operators in optimization modeling.

These Tutorials will show how to pass over these constraints. The authors of the Tutorial managed to find a way of resolving the problem that first seems to have no solution.

Dollar operator is used in the model for conditional assignations, expressions and equations. The subsections below will show examples of using them. The following paragraph deals with the themes of dollar conditions in conditional assignations, expressions and equations.

The essence of $ operator can be explained by analyzing the following example.

The following simple condition:

If (b>1.5), then a = 2;

can be written in GAMS with the help of the dollar operator as follows:

a$(b > 1.5) = 2;

If the condition is not fulfilled then the assignation is not executed.

8.2 Embedded Dollar Conditions

Dollar conditions can be embedded. The term $(condition1$(condition2)) can be read as $(condition1 and condition2). That is, simultaneous fulfillment of the first and second conditions. In case with the embedded dollar conditions all expressions following the dollar must be inserted into a parenthesis.

Consider the following example:

d(i)$(w(i)$q(i)) = v(i);

where i, d(i), w(i) are sets while q(i) and v(i) are parameters. The assignation will be executed for element d(i) which simultaneously are elements of w and q. Note the position of parenthesis in dollar conditions. The operator given above can be rewritten as follows:

```
d(i)$( w(i) AND q(i) ) = v(i);
```

In order to provide readability of the operator it is recommended to use logical operator AND instead of embedded dollar operators.

8.3 Conditional Assignations

The operator given above was connected with conditional assignation. In this example, the dollar condition was on the left side of assignation. But the effect of the dollar condition mostly depends on its position regarding the equal sign.

This operator uses dollar condition on the left side of the equal sign. The assignation of the operator with dollar condition on the left side is not fulfilled, if logical conditions are not satisfied. It means that values of parameters on the left will remain unchanged for those indexes, where conditions are not fulfilled.

If the parameters of the left side of assignation are not numerically determined, their values will be equal to 0, for logical conditions are not satisfied. The $ mark is very useful for the cases when it is necessary to avoid dividing by zero. For example:

```
result(i)$( znamen(i) NE 0 ) = (1./znamen(i)) -1.;
```

The znamen(i) parameter was defined in the model and the operator is used for calculating result(i) without making mistakes. Dollar values can be used to prevent dividing by 0.

If any value associated with znamen(i) is equal to 0, the assignation will not be executed and there will be the preceding value for result(i). If result(i) was not initiated before (by assigning some value), then all indexes with znamen(i) equaling 0, will be assigned 0 value.

If znamen(i) was not defined for cases when it equaled 0, then the assignation above can be written down as follows:

```
result(i)$znamen(i)  =  (1./znamen(i)) -1.;
```

It is clear that znamen(i) will be associated with Boolean variable FALSE and assignation will be executed automatically avoiding cases of dividing by 0.

8.4 Dollar On The Right Side Of The Equation Mark

Assignation is always executed with the assignation operator with a dollar condition on the right side. If the logical condition is not fulfilled, then the parameter or the variable will be transformed into 0, even if it was equal to some number. For example:

```
x = 20$(y > 7.5);
```

If expressed by words it will look like as follows:

```
IF (y > 7.5)  THEN  (x=20),  ELSE  (x=0)
```

The type of construction expressed by (IF-THEN-ELSE) will work out as the condition with the symbol of dollar. One more variant of achieving the same result is given bellow (when GT and LE operators are used):

```
x = 20$(y  GT  7.5) + 0$(y  LE  7.5);
```

8.5. Conditional Index Operations

Another useful application of the dollar condition is the control of index operations. The procedure is nearly the same as with the dollar on the left side. For example:

```
summa  =  SUM(i$(nechto(i)),product(i));
```

The operator calculates the sum of *product(i)* for all SET i, but under the condition that parameter *nechto(i)* is defined for the current index i.

Another typical example of using $ operator is the correlation of (SET-to-SET).

```
SETS  r  / west, east, north, south /
s  / andijan, fergana, navoi, uchcuduk, tokio /
sootvet(r,s)  /north.( navoi, andijan )
south.(tokio, uchcuduk )/
PARAMETER  v(r)
intake(s)  "income of each state"
/ andijan    75.8,  navoi          57.5
  fergana    67.5,  uchcuduk    87.4/;
```

The set *sootvet* provides the congruence and shows the relationships between the two independent SETS. For example, let the *intake* parameter be an intake for each city. The v(r) value can be defined by assignation operator with an embedded $ operator.

```
v(r)  =  SUM(s$sootvet(r,s),  intake(s));
```

For every case of *r*, the summation with the help of *s* is possible only through pairs (r,s) where *sootvet(r,s)* exists. Schematic existence of these paired sets of contacts between the cities is similar to logical value of (true) or arithmetical value of "no zero". The summation above can also be written down as follows:

SUM(s, intake(s)$sootvet(r,s)).

However, this form is also not very easy for understanding.

The use of dollar operator $ for multidimensional SETS connected with power calculations in models is shown in the Appendix of this Tutorial.

8.6 Conditional Equations

Dollar operator is also used for managing the calculation process in equations. In the subsections below, two main types of dollar operators for equations are discussed.

1) In the body of the equation, and;
2) For managing the calculation of the equation.

Dollar operator in the body of the equation is similar to dollar control on the right side assignation as shown above. And if "right " is represented as a value on the right from '..', the similarity will be even closer. The operation of (IF-ELSE) is assumed to be the same as in the case with assignation, in the paragraph for calculation of parameters. For example:

maba(i).. x(i) =LE= y(i) + (e(i) - m(i))$f(i);

In this expression, value (e(i) - m(i)) is added to the right side of the equation for elements *i* referred to f(i).

The control of index operations with the use of dollar conditions can be accomplished as in case with assignation. Consider the following inequation under the name (sb):

sb(i).. SUM(j$nn(i,j), x(i,j)) =L= s(i);

In this example, we have a summation of all elements x(i,j) of the index (SET) j for cases with double SET nn(i,j). In cases, where j does not have a corresponding value nn(i,j), the adding of the value x(i,j) is not executed.

8.7 Dollar Control over Calculation of Equations

This control is similar to dollar control on the left side of the assignation (see above). If left is understood as being on the left of '..', then the similarity is even closer. The objective of the dollar control through equations is to restrict the number of constraints to be less than was determined by sets. Consider the following example dealing with the optimal management of water resources:

```
ou_flow_nl(n,m)$(nl(n)$(NOT mm(m)))..
ou_flow(n,m) =E=  -vol(n,m)+vol(n,m-1)+in_flow(n,m);
```

where:

- n - is a set of river nodes of the model;
- m - set of estimated time intervals of the model;
- m-1 - previous estimated time interval;
- nl(n) - SETS for reservoirs;
- mm(m) - SETS for estimated time intervals of the first month of the whole period. They are introduced for avoiding calculation problems when addressing the previous month (m-1) which calculation is not defined. ou_flow(n,m), vol(n,m), in_flow(n,m) are the variables defined in the optimization process.

The first line contains the name of the equation ou_flow_nl and the condition $(nl(n)$(not mm(m))). The computation is accomplished according to the following scheme:

> The equation is solved if the calculation node refers to *nl(n)* set of reservoirs, and the time interval is the first in the list of sets *m* characteristic for estimated time intervals and is not associated with subsets of the initial time interval *mm(m)*.

8.8 Special Constructions for Conditional Variables

One of the main constraints of dollar operators is that they are prohibited for use with variables defined through optimization. For example, we have to solve a problem with variables determining the computation of equations:

> Y = X**2, if X < 0 otherwise Y =X*X*X;

where Y, and X are variables. We have the following structure of computation:

```
VARIABLES y, x;
EQUATION
first;
first.. y  =E=  (x*x)$(x < 0) + (x*x*x)$(x > 0);
x.LO  =  -10;
x.UP  =  10;
MODEL andre /ALL/;
SOLVE andre USING DNLP MINIMIZING y;
```

The GAMS compiler will inform you about the error made by using the dollar operator with variables.

```
**** THE FOLLOWING DNLP ERRORS WERE DETECTED IN MODEL ANDR:
**** 53 IN EQUATION FIRST      .. ENDOG $ OPERATION
```

**** 53 IN EQUATION FIRST .. ENDOG $ OPERATION

However, the necessary result can be achieved by using the following construction of equations:

```
first.. y =e= (x*x)*
(0.5*(sqrt(x*x)-x)/(sqrt(x*x)+0.000001))+
(x*x*x)*
(0.5*(-sqrt(x*x)+x)/(sqrt(x*x)+0.000001));
```

The feature of the computer to define a square root as positive is used here. In principle, the answer from the three arithmetic constructions given below can form an ARITHMETIC switch for the element or subset of the equation based on variables. This method allows bypassing the compiler constraint on the use of conditional operations for equation variables in optimization models:

1) z/ ABS(z) = 1, if z>0
   z/ ABS(z) =-1, if z<0

2) -0.5*(z-SQRT(z*z)) = 0, if z>0
   -0.5*(z-SQRT(z*z)) = z, if z<0

3) (ABS(z)*z)/(z*z+0.00000001) = +1, if z>0
   (ABS(z)*z)/(z*z+0.00000001) = -1, if z<0

In this way, there is no risk of being divided by zero, however, there are some distortions around z = 0. There is a possibility of using conditional operators for computation (on the right side of symbols).

Truncation of equations is done by conditional operators $ for index parameters and sets (see examples above and below). The way of solving the complex problem of using conditional operators with variables is shown below:

> We have two reservoirs. The first reservoir (called a flow-type reservoir) takes water in (according to a defined hydrograph of inflows) and releases it (according to a defined hydrograph of demands). The second reservoir (called an intake-type reservoir) is connected with the first reservoir through a valve that opens when the amount of water of the first reservoir exceeds the water amount of the second reservoir.
>
> Besides, the valve opens (in the other direction) when the water volume of the first reservoir is not enough for the demanded releases. In this case, water from the intake reservoir will cover water deficit of the flow-type reservoir through the open valve.

This is not an abstract example. It is taken from real life. Big Tuyamuyun reservoir on the Amu Darya River is comprised of two parts: onstream and offstream parts. Water comes to the offstream part of the reservoir by gravity flow when the water level of the onstream part exceeds the water level of the offstream part. On the other hand, water of the offstream reservoir comes to the onstream section when there is a shortage of irrigation water for agriculture. Water storage in the

offstream Tuyamuyun reservoir is more preferable than in the onstream part, due to its structure. Hence, preferential storage of water is achieved in the offstream part of the reservoir.
Consider the mathematical aspect of the problem connected with the two reservoirs:

1) $$\frac{dW_p}{dt} = Q_{pr} - Q_{ot} - Q_{in} + Q_{ret}$$

2) $$\frac{dW_n}{dt} = Q_{in} - Q_{ret}$$

3) $W_p(t) = W_n(t)$ if $Q_{in}(t) > 0$, $Q_{ret} = 0$

$W_{n,min} < W_n < W_{n,max}$
$W_{p,min} < W_p < W_{p,max}$

$0 < Q_{in} < Q_{in,max}$
$0 < Q_{ret} < Q_{in,max}$

$Q_{pr} = F_1(t)$, $W_n = W_{no}$ if $t = 0$
$Q_{ot} = F_2(t)$, $W_p = W_{po}$ if $t = 0$

4) $$\sum_{t \geq \min} Q_{ret}(t)$$

It is obvious that the third equation gets involved in the calculation when water storages in both reservoirs are equal. Equation 4 provides for a constant trend of $Q_{ret}(t)$ to decrease down to 0. It will never be equal to zero, if there is not sufficient water in the first reservoir. It will automatically mean that $Q_{in} = 0$ (equation 3 is not taken into account).

How can we model this situation in GAMS? Since we do not have a possibility of taking out the equation from the computation process using the conditional operator, we eliminate the impact of the equation. It can be attained by isolating the elements of the equation by transforming it into an identity.

```
SETS
        n capacity  /tmgu,sult/          determination of two capacities
                                          (onstream and offstream);
SETS
        m  time   / jan, feb, mar, apr, may,     determination of estimated time
                    jun, jul, aug, sep, oct,     interval sets;
                    nov, dec, eenndd /
        mm(m) /jan/                              selection of the first estimated time interval;

TABLE in_flow(n,m)          inflow in the stream-flow reservoir(mln.cub.m)
        jan feb mar apr may jun jul aug sep oct nov dec  eenndd
        tmgu       128 125 234 360 541 645 807 512 267 210 981 928   250
```

```
TABLE ou_flow(N,m)                    water requirements (mln.cub.m)
                                      releases from the stream-flow reservoir;
      jan feb mar apr may jun jul aug sep oct nov dec  eenndd
      tmgu      111 111 111 500 222 700 333 333 333 333 333 333   200

VARIABLES
      intake(m),              intake to the offstream reservoir;
      sbros(m),               discharge from offstream reservoir;
      vol(n,m),               storage for each reservouir and each time interval;
      obj;                    criterion function;

EQUATION
      balanc1(n,m),           equation for water balance of the stream-flow reservoir;
      balanc2(n,m),           equation for water balance of the offstream reservoir;
      rules1(n,m),            equation for rules of filling the reservoirs;
      ben;                    equation containing the criterion function;

      balanc1(n,m)$(not mm(m))..                  equation for the stream-flow reservoir;
      vol('tmgu',m)-vol('tmgu',m-1) =e=           volume change;
      +in_flow('tmgu',m)-ou_flow('tmgu',m)        inflow minus outflow;
       -intake(m)+sbros(m);                       relationship with the offstream reservoir;

       balanc2(n,m)$(not mm(m))..          equation for the offstream reservoir;
       vol('sult',m)-vol('sult',m-1)=e=    volume change;
       intake(m)-sbros(m);                 relationship with the offstream reservoir;

      rules1(n,m)$(not mm(m))..                   equation for rules of reservoir functioning;
      (vol('sult',m)-vol('tmgu',m))     -                    1 – line
      (vol('sult',m)-vol('tmgu',m))     *                    2 – line
       (1-intake(m) /(intake(m)+0.0000000000000000001))      3 – line
       =E= 0.0;
```

Consider the relationships between all lines of the equation. The third line can be equal either to 1 or 0 depending on the value of the variable Qin. It is a logical operator formed in accordance with arithmetical rules.

If line 3 was missing (the third line is equal to 1), then lines 1 and 2 would form an identity. In other words, the equation does not affect the calculation and can be considered as non-existent in the model. To see this, we can add to the model an equation where the difference of values is equal to zero (nothing will change).

If the third line is equal to zero, it means reduction of the initial equation:

```
vol('sult',m) =E= vol('tmgu',m);
```

which means that volumes are equal in both reservoirs. That is has to be provided in the calculation.

```
ben.. obj =e= SUM(M$(not mm(m)),sbros(m));      criterion function that provides
                                                storage for the offstream reservoir;
vol.LO('tmgu '  ,m    ) = 1150;                 lower level of the stream-flow reservoir;
vol.UP('tmgu '  ,m    ) = 4590;                 upper level of the stream-flow reservoir;
vol.FX('tmgu '  ,'jan') = 1200;                 initial level of the stream-flow reservoir;
vol.LO('sult '  ,m    ) = 100 ;                 lower level of the offstream reservoir;
vol.UP('sult '  ,m    ) = 4590;                 upper level of the offstream reservoir;
vol.FX('sult '  ,'jan') = 1200;                 initial level of the offstream reservoir;
sbros.UP(m)= 1500;                              possible limits of intensity;
sbros.LO(m)= 0.0;                                     drawdown of the offstream reservoir;
intake.UP(m)= 1500;                             possible limits of intensity;
intake.LO(m)= 0.0;                              drawdown of the stream-flow reservoir;
intake.L(m)= 0.00001;                           initial stage of finding a solution;

OPTION ITERLIM= 20000;                          not more than 20000 iterations
OPTION LIMROW =12;                              list all estimated time intervals;
OPTION OPTCR  =0.000001;                        accuracy of optimal solution;

MODEL tmgu   /all/;

SOLVE tmgu   USING  DNLP MINIMIZING obj;
SOLVE tmgu   USING  DNLP MINIMIZING obj;

PARAMETER A(M);
A(M)=(1-(intake.l(m)/(abs(intake.l(m))+0.00000001)));
FILE res /amu_TMG6.dat/
PUT res;
PUT obj.l:10:5;
PUT "  =========balance ====================  "/;
PUT "                                  "/;
PUT "  ====================================  "/;
PUT "                                  "/;
PUT "          sbros  intake  sult  tmgu                "/;
LOOP((m)$(ord(m) ne card(m)),
 PUT   A(M):5:2,sbros.L(M):7:1,intake.L(M):7:1,vol.L('sult',m):7:1,vol.L('tmgu',M):7:1,
      in_flow('tmgu',m ):7:1,ou_flow('tmgu',m ):7:1,(in_flow('tmgu',m )-ou_flow('tmgu',M )):8:2,
      (vol.L('tmgu',m)-vol.L('tmgu',m-1)):8:2,(vol.L('sult',m)-vol.L('sult',m-1)):8:2,
      ((vol.L('tmgu ',m)-vol.L('tmgu',m-1))+( vol.L('sult ',m)-vol.L('sult ',m-1))):8:2 /;);
 PUT " ==================================== "/;
  The result of the calculation will be represented in text file:
====================================
1200.0 1200.0
 1 0.00   0.0    7.0   1207.0 1207.0 125.0 111.0   14.00    7.00     7.00
 2 0.00   0.0   61.5   1268.5 1268.5 234.0 111.0  123.00   61.50    61.50
```

3	1.00	1.5	0.0	1247.0	1150.0	360.0	500.0	-140.00	-118.50	-21.50
4	0.00	0.0	111.0	1358.0	1358.0	541.0	222.0	319.00	208.00	111.00
5	1.00	0.0	0.0	1358.0	1303.0	645.0	700.0	-55.00	-55.00	0.00
6	0.00	0.0	209.5	1567.5	1567.5	807.0	333.0	474.00	264.50	209.50
7	0.00	0.0	89.5	1657.0	1657.0	512.0	333.0	179.00	89.50	89.50
8	1.00	0.0	0.0	1657.0	1591.0	267.0	333.0	-66.00	-66.00	0.00
9	1.00	0.0	0.0	1657.0	1468.0	210.0	333.0	-123.00	-123.00	0.00
10	0.00	0.0	229.5	1886.5	1886.5	981.0	333.0	648.00	418.50	229.50
11	0.00	0.0	297.5	2184.0	2184.0	928.0	333.0	595.00	297.50	297.50
12	0.00	0.0	25.0	2209.0	2209.0	250.0	200.0	50.00	25.00	25.00

==========

The Table above contains the obtained results. It is obvious that the offstream reservoir was used during the third time interval, when the stream-flow reservoir was incapable of covering the deficit. In the case with time intervals 3,5,8 and 9 the equation, that providing the equality of volumes, did not work. In the rest of the cases, it worked providing the equality of reservoir volumes.

In other words, the result of the truncation of the SET m is provided by relationship between the variables. This concluding section of the Tutorials shows that GAMS is very sensitive to the construction of logical relations between equations. If GAMS does not free the users from the routine of constructing calculation algorithms, it sets higher requirements to logical behavior of equation sets in models.

The user will obtain a proper answer to a correct objective, properly formulated.

9. ADDITIONAL PROGRAMMING OPERATORS IN GAMS

GAMS has a number of embedded operators frequently used for other programming languages. This section is about the operators and their use. There are four embedded GAMS operators:

1) LOOP - cycle operator.
2) IF-ELSE - conditional operator.
3) FOR - cycle operator.
4) WHILE - conditional cycle operator.

9.1. Loop

The LOOP operator is very frequently used with the PUT operator for printing out and displaying the information on the screen.

The LOOP operator has the following syntax:

```
LOOP (control_ area [$(condition)] operator {; operator});
```

If control_area contains more than one set, then parentheses are needed. The LOOP operator provides performance of the operator within the cycle limits for each index of the set in turn. The sequence of work is determined by the sequence of indices. Therefore, LOOP(cycle) is a more general type of index operation. The cycle set can be controlled by the $ operator, but it must be static or embedded. The cycles can be controlled by more than one set. However, it is impossible to declare or solve equations within the LOOP operator. It is not allowed to transform control sets within the body of the cycle.

Consider the following hypothetical case of the integration model:

```
SET  x  / d15*d25 /
PARAMETER  nachalo(x)  / d15 423 /
differ(x)  / d15  58.4, d16  47.7, d17  68.8
             d18  46.5, d19  57.9,  d20  46.9
             d21  64.5, d22  65.4,  d23  34.7
             d24  76.4, d25  34.4/;
```

The LOOP operator can be used for computing increasing sums:

```
LOOP(x, nachalo(x+1)  =  nachalo(x)  +  differ(x));
```

Only one operator and one control set - x is presented here within the cycle operation area

```
nachalo(x+1) = nachalo(x) + differ(x)
```

Consider the following example used in the algorithm of the Newton method for extracting a square root. In practice, the imbedded function SQRT() is used. However, in this case, we have to show the functioning of LOOP operator on the basis of the program taken from the GMAS Language Guide (Brooke et al., 1997). All comments are given in Russian, but it must be kept in mind it is done for training reasons only. Attempts at inserting comments with Cyrillic characters will be followed by a message about errors in the program.

So, we have an approximate equality specifying that:
if *x* – is an approximated square root of *v*, then

```
(x+v/x) - 2*x = 0

SET I                          "number of iterations in 100" /i-1*i-100/;
PARAMETER  value(i)            "value defined as a square root";
SCALARS
    target        "the number for extraction of the square root"  /23.456/;
    sqrtval       "ultimate value of approximated extraction of the square root";
    curacc        "accuracy of calculation in accordance with Newton methodology"
    reltol        "required relative inaccuracy = 0.000001" /1.0e-06/;
ABORT$(target <= 0)  "testing of the positive value of the initial figure ",
                          target; value("i-1")  =  target/2 ;
curacc   =  1;
loop(i$(curacc > reltol),
        value(i+1) = 0.5*(value(i) + target/value(i));
        sqrtval = value(i+1);
        curacc  = ABS (value(i+1) - value(i))/(1+abs(value(i+1)))
     );
ABORT$(curacc>reltol) "the square root is not extracted with the required accuracy "
nachalo(x+1)  =  nachalo(x)  +  differ(x) option decimals = 8;
DISPLAY "square root is extracted with the required accuracy ",sqrtval, value;
```

The conclusion is:

```
----  21 square root is extracted with the required accuracy;
----  21  PARAMETER SQRTVAL   =  4.84313948 ultimate value of the
                                      approximated root extraction;
----  21  PARAMETER VALUE    the value defined as a square root;
I-1  11.7280000, I-2  6.8640000, I-3  5.1406247, I-4  4.8517471
I-5   4.8431471, I-6  4.8431394, I-7  4.8431394
```

In this example the user can see a new operator, ABORT, which is used for fail-safe canceling of computation.

9.2. IF-ELSE Operator

The IF-ELSE operator is useful for transferring from one operator to another. In some cases, it can be written down as a set of dollar conditions. The IF operator can be used for making the GAMS code more understandable.

The optional part of "ELSE" allows formulating the traditional construction "IF-THEN-ELSE". The following syntax is for the "IF-THEN-ELSE" operator:

```
if (condition,
        operators;
{condition ELSEIF, operators}
        [operators ELSE;]
);
```

Note, the braces and brackets are not required, but can be used. "Condition" means logical conditions described in the paragraph on conditional operations of GAMS.

Declaration or definition of equations can not be performed within the IF operator. Consider the following set of operators taken from the GAMS Language Guide (Brooke et al., 1997):

```
p(i)$(f <= 0) = -1;
p(i)$(f >  0)  and (f < 1)) = p(i)**2;
p(i)$(f >  1) = p(i)**3;
q(j)$(f <= 0) = -1;
q(j)$(f >  0)  and (f < 1)) = q(j)**2;
q(j)$(f >  1) = q(j)**3;
```

the same can be expressed through the IF-ELSEIF-ELSE operators:

```
IF (f <= 0,
        p(i) = -1;
        q(j) = -1;
ELSEIF ((f > 0) AND (f < 1)),
        p(i) = p(i)**2;
        q(j) = q(j)**2;
ELSE
        p(i) = p(i)**3;
        q(j) = q(j)**3;
);
```

The body of the IF operator can contain the SOLVE operator. Consider the following example of the model from the GAMS Language Guide (Brooke et al., 1997):

```
IF  (ml.MODELSTAT EQ 4),
*     the model ml is undefined;
*     change the conditions on the boundaries of x and solve it again;
```

```
        x.up(j) = 2*x.up(j) ;
        SOLVE ml USING LP MINIMIZING  lop;
    ELSE IF (ml.MODELSTAT NE 1), ABORT "error occurred in solving the model ml";););
```

The user can see a new suffix ".MODELSTAT" here. Its meaning is quite transparent and its numerical value is relevant to situations raised in the process of solving the problem. A complete list of situations is given in the GAMS Language Guide.

9.3 WHILE Operator

WHILE operator is used for cycling the process of calculation before the execution of some logical condition. The syntax of the WHILE operator is the following:

```
while (operator,
        operators;
);
```

As with the FOR operator, declaration and definition of equations can not be done in the WHILE operator. However, WHILE operator can be used for control of the SOLVE operator. Consider the following part of the GAMS program that is randomly looking for the global optimum of the non-convex function. The example is taken from the GAMS Language Guide:

```
SCALAR  count;  count = 1;
SCALAR  globmin;  globmin = INF;
OPTION  BRATIO = 1;
WHILE ((count  LE  1000),
        x.l(j) = UNIFORM(0,1);
        SOLVE ml  USING  lp  MINIMIZING  obj;
        IF  (obj.l  LE  globmin,
                globmin = obj.l;
                globinit(j)  = x.l(j);
        );
        count = count +1;
);
```

In this example, the non-convex model is solved on 1000 randomly taken numbers, while the global solution is found through constant comparison.

Model [PRIME] from the GAMS library of models illustrates the use of WHILE operator in the example with simple generation of numbers less than 200.

9.4 FOR Operator

Operator FOR is used for cycling the bloc of operators. It has the following syntax:

```
FOR  (i = initial value, TO finite value [BY step]
        operators;
);
```

Note that *i* is not a set, but a numerical parameter. The step defines the growth depending on the alteration of *i* after each cycle of FOR. The initial and final values of the *i* parameter and of the step do not have to be integer-valued. The initial and final values can be positive or negative real numbers. The step must be a positive real number.

Operator FOR can be used for control of the SOLVE operator. Consider the following example from the GAMS program randomly studying a non-convex function for finding a global optimum. The example is taken from the GAMS Language Guide:

```
SCALAR i;
SCALAR globmin; globmin = inf;
OPTION  BRATIO = 1;
FOR (i = 1 TO 1000,
        x.l(j)  =  UNIFORM(0,1);
        SOLVE ml  USING  NLP  MINIMIZING  obj ;
        IF  (obj.l  LE globmin,
                globmin = obj.L ;
                globinit(j)  =  x.L(j);
        );
);
```

In this example, the non-convex function is examined for 1000 randomly taken points, while the global solution is found through comparison. The use of real numbers as initial and final values of the cycle and steps, can be shown in the following example:

```
FOR (s  =  -18.4  TO  10.3  BY 6.1, DISPLAY s; );
```

Resulting printout of the file will contain the following lines:

```
----   1 PARAMETER  S       =  -18.400
----   2 PARAMETER  S       =  -12.300
----   3 PARAMETER  S       =  -6.200
----   4 PARAMETER  S       =  -0.100
----   5 PARAMETER  S       =  +6.000
----   6 PARAMETER  S       =  -12.100
```

Note, that S value was increased by the GAMS compiler 6.1 during each cycle until it exceeded 10.3.

10. Operators for Defining a Serial Number of Index in Sets

10.1 Sequence of Indexes in the Set

When describing methods of declaration of sets and subsets, it was shown how elements of sets are defined. But the GAMS compiler permits the use of not only integer, but literal values as well. It was shown above that the points do not possess numerical values. The index '1986' does not have a numerical value 1986, and therefore, the index '01' is not only a number but is not even identical to the index with the name '1'.

However, as it was described above, the GAMS compiler memorizes index names, as well as their sequences. The user also has access to special operators ORD( ) and CARD( ). Section 10 describes two operators ORD( ) and CARD( ) for restoring the values when sets are addressed. If the sequence for sets of indexes is once arranged, it will remain for other cases.

In order to see the sequence of indexes taken by the GAMS compiler, $ONUELLIST command must be used before the first declaration. The process of arranging indexes into sequence is seen best from the $ONUELLIST program operation.

```
SET  f1  /a487, a488, a489, a490, a491/
     f2  /a483, a484, a485, a486, a487/
     f3  /a485, a489, a491, a483, a487/ ;
PARAMETER v1(f1) /a487   1,a488   2,a489   3,a490   4,a491   5 /;
PARAMETER v2(f2) /a483  21,a484  22,a485  23,a486  24,a487  25 /;
PARAMETER v3(f3) /a485  31,a489  32,a491  33,a483  34,a487  35 /;
FILE res /dasa.dat/
PUT res;
PUT /;LOOP((f1),PUT v1(f1):10:2  /;);
PUT /;LOOP((f2),PUT v2(f2):10:2  /;);
PUT /;LOOP((f3),PUT v3(f3):10:2  /;);
```

OUTPUT FILE dasa.dat

```
      1.00        25.00        35.00
      2.00        21.00        32.00
      3.00        22.00        33.00
      4.00        23.00        34.00
      5.00        24.00        31.00
```

DEFAULT ENTRY FILE
Unique Elements in Entry Order
(sequence of indexes according to the entry)

```
1  A487     A488     A489     A490     A491     A483
7  A484     A485     A486
```

Unique Elements in Sorted Order

```
1  A483     A484     A485     A486     A487     A488
7  A489     A490     A491
```

The interested modeler or user can use the program for examining all possible combinations of index names and study consequences of their sorting. Due to the time constraints it is impossible to describe all options. But it should be noted that letter indexes are sorted according to the English alphabet. You can check and see it yourself.

10.2 Operators ORD( ) and CARD( )

The ORD operator gives the appropriate index positions of the set and is equal to an ordinal number. ORD can be used with one-dimensional static and sequential sets. Here are some examples showing how the ORD operator is used.

```
SET t  vakt  time   /100*120/
PARAMETER  val(t);
val(t) = ORD(t);
```

The result of the operator will be the following:

- val('100') will be 1;
- val('101') will be 2, etc.

The functioning of the operator is best shown in the examples of the Appendix. CARD gives the number of elements of the set as an ordinal number. CARD can be used with any set, even with a dynamic and non-arranged one. For example:

```
SET  t  time  period  /45*68/
PARAMETER  s;  s = CARD(t);
```

As a result, the CARD( ) operator will be assigned the value 24. The CARD( ) operator is frequently used for initiating and canceling computation at the final stage. The example below shows the fixation of the variable of the optimization model, so that the value of the variable is equal to the demand(t) parameter:

```
c.fx(t)$( ORD(t) = CARD(t) ) = demand(t);
```

The operator is not functional for values of the t index that are not the final one in the set.

10.3 LAG and LEAD Operators

Operators LAG and LEAD are used for correlating the "estimated" element of a set with the "next" or "preceding" element of the set. These operators are applicable for arranged sets only. GAMS has two variants of LAG and LEAD operators:

- linear operators LAG and LEAD (+, -)
- circular operators LAG and LEAD (++, --).

The difference between these two types of operators is in the method of processing at the initial and final points of the sequence. In circular operators after the final element of the sequence there comes the first, while in linear operators sets are broken. The risk is connected with an attempt at making the GAMS compiler process those parameters or variables that are not defined by appropriate indexes.

Linear operators LAG and LEAD (+, -) are very convenient for modeling time periods that are not cycled. Particular year (design steps: - one month and less) is an example if the user does not want the initial values to be identical with the final ones. For example, it is unlikely that the initial storage of a long-term reservoir will be identical with its end storage.

GAMS can differentiate linear operators (+, -) from arithmetic operators according to the context. When LAG and LEAD operators are used in the circular context it is assumed that the initial and final values of sets are adjacent, i.e. "the first-1" is "final", "the last +2" is "the first +1", etc. All indexes for providing appropriate assignations are defined.

The circular form of operators LAG and LEAD (++, --) is convenient for modeling cycled periods of time. A particular year (design steps: - one month and less) is an example, if the user does not want the initial values to be identical with the final ones. For example, often the initial storage of a seasonal reservoir is to be identical with its end storage. Naturally, January is the month following December despite changing years. Production processes of cyclic character, where separate stages have to be defined, are very convenient for using LEAD and LAG operators.

The operators LEAD and LAG can be used in assignations. Use of operators LAG and LEAD on the left side of the assignation is called an "index" while the use of it on the left side is called "assignation" and includes definition of the area of assignation.

The index for unreal elements is called a default element (zero) and the computation process does not stop. However, attempts of assigning computation results to unreal elements are not processed by the GAMS compiler. Consider the following example that illustrates application of linear operators LAG and LEAD for indexing:

```
SET  t  vakt  time    /j-1987*j-1991/;
PARAMETER a(t), b(t);
a(t)  =  1986 + ord(t);          Parameter a(t) for each 19??
                                 An integer value equal to a year is assigned
                                 to the parameter a(t) for  j-19??.
b(t)  =  -1;                     Assignation to the unidimensional array
```

b(t) values –1 for increasing the information area
of the follow-up computation.

```
b(t) = a(t-1);
OPTION DECIMALS = 0; DISPLAY a, b; operator OPTION DECIMALS
```

Discards decimal numbers for demonstration through DISPLAY operator (described below).

The calculation results will be as follows:

```
----   6 PARAMETERS A
Y-19U87 1987, Y-1988 1988, Y-1989 1989, Y-1990 1990, Y-1991 1991
----   6 PARAMETERS B
Y-1988 1987, Y-1989 1988, Y-1990 1989, Y-1991 1990
```

Note that a( ), as well as b( ) have 5 indexes, but the first index belongs to b( ), because during the computation it has addressed the non-existent element a( ). But the preceding value b(1) existed and was equal to -1.

Consider the following example where two parameters *a* and *c* are used for illustrating assignation of linear operators LAG and LEAD.

```
SET  t  vakt   /j-1987*j-1991/;
PARAMETER a(t), b(t);
 a(t)  =  1986 + ord(t);
 b(t)  =  -1;
 b(t+2) = a(t);
OPTION DECIMALS = 0; DISPLAY a, b;
```

The fragment of the printout is given below.

```
----     14 PARAMETER A
J-1987 1987,    J-1988 1988,    J-1989 1989,    J-1990 1990,    J-1991 1991
----     14 PARAMETER B
J-1987   -1,    J-1988   -1,    J-1989 1987,    J-1990 1988,    J-1991 1989
```

It can be seen that all elements of the set b() are present this time, though two of them are not defined and have not been processed by an appropriate operator LAG or LEAD.

```
ziro(t) = ziro(t - ord(t));
```

It is obvious that this operator is identical to

```
ziro(t) = 0;
```

Another example illustrates the use of the circular operators LAG and LEAD.

```
SET          s   kvartals  /spring, summer, fall , winter/;
```

```
PARAMETERS      value(s)   /spring  701, summer  702, fall 703, winter  704/;
PARAMETERS      lagval__1(s)
PARAMETERS      leadval_2(s) ;

lagval__1(s) = -7; lagval__1(s)  = value(s--3);
leadval_2(s) = -7; leadval_2(s++2)  = value(s);

OPTION DECIMALS = 0;  DISPLAY value, lagval__1, leadval_2;
```

The result of the calculation is given below.

```
----      9 PARAMETER LAGVAL__1
SPRING 702,    SUMMER 703,    FALL   704,    WINTER 701
----      9 PARAMETER LEADVAL_2
SPRING 703,    SUMMER 704,    FALL   701,    WINTER 702
```

It can be seen that all calculations are moving along the cycle of four seasons of the year:

```
lagval__1(s)     = value(s--3);
leadval_2(s++2)  = value(s);
```

equivalent to:

```
lagval__1(s++3)  = value(s);
leadval_2(s)     = value(s--2);
```

From the analysis of the last equivalent records it is clear that there is no difference in using "indicator" and "assignation" in cyclic operators LAG and LEAD.

Operators LAG and LEAD can be used in equations. Principles defined for these operators are true for equations. The location of operators LAG and LEAD in the body of the equation is very important. They become indicators if located on the right side of the symbol "..". If their conditions are not met, they disappear.

LAG and LEAD operators located on the left side of the symbol ".." are modifications of the area for defining equations. But it must be kept in mind, that a linear form can be a reason and that the variables of boundaries of arranged sets fall outside the limits of the definition. In each particular case, the user will find not very pleasant results.

Consider the examples with linear operator forms of LAG and LEAD in equations used for reducing the sphere of their determination.

This is an adapted example from [RAMSEY]:

```
SET  t  time  periods  /1990*2000/
tfirst(t)  first period
tlast(t)   last period;
```

```
tfirst(t)  = yes$(ord(t)  eq1);
tlast(t)   = yes$(ord(t)  eq (card(t));
DISPLAY  tfirst,  tlast;
VARIABLES  k(t)  capital stock (trillion rupes)
                 i(t)  investment (trillion rupes per year);
EQUATIONS  kk(t)  capital balans (trillion rupes)
                  tc(t)  terminal condition (for post-term growth);
                  kk(t+1)..      k(t+1) =e= k(t) +i(t);
                  tc(tlast)..    g*k(tlast)  =l= i(tlast);
```

Declaration t is turned on as a set for operating the first and last indexes through the subsets.

```
kk(t)$(not  tfirst(t)).. k(t+1) =e=  k(t)  +  i(t);
```

Conditional operator $ is the reason for eliminating the calculation of the first index of the set t. It is clear that the use of operators LAG and LEAD in the area of the names of equations (before symbols ..) will lead to canceling of one or more equations during the computation process. This can cause undesirable consequences for other equations of the model. On the whole, the choice between LAG and EAD operators as an indicator or for control of computation depends on the taste of the user and is only restricted by the internal logic of the model.

In case of cycle operators LAG and LEAD, differences between these uses are not important. For example:

```
SET   t  season  /spring, summer, fall, winter/;
PARAMETERS demand(t)   /spring 100, summer 120, fall 123, winter 213/;
PARAMETERS capacity(t) /spring 121, summer 45, fall 300, winter 214/;
POSITIVE VARIABLE
        produce(t)
        sale(t)
        available(t);
VARIABLE
        result    ;
EQUATION
        matbal(t)
        mat;
        matbal(t)..  availbl(t++1) =E=  produce(t)  -  sale(t) + availbl(t);
        mat..     result =E=  sum(t,sale(t));
sale.UP(t)    =   demand(t);
produce.UP(t) =   capacity(t);
availbl.UP(t) =   1000;
MODEL andre /ALL/;
SOLVE andre USING NLP  MAXIMiZING result;
FILE res /result.dat/
PUT res;
PUT " season      sale produce available demand capacity ";PUT /;
PUT "_____ ";PUT /;
```

```
LOOP((t),PUT t.tl,sale.l(t):8:2,produce.l(t):8:2,
availbl.l(t):8:2,demand(t):8:2,capacity(t):8:2 /;);
PUT "_____ ";
PUT result.l:10:2;
```

There will be the following result:

```
Season sale produce available demand capacity
_____
SPRING       100.00  121.00   54.00  100.00  121.00
SUMMER        120.00   45.00   75.00  120.00   45.00
FALL         123.00  176.00    0.00  123.00  300.00
WINTER        213.00  214.00   53.00  213.00  214.00
_____
556.00
_____
```

In this example, four equations are calculated. They are given below:

```
availbl(summer) =e=  produce(spring) - sale(spring)+availbl(spring);
availbl(fall  )      =e=  produce(summer) - sale(summer)+availbl(summer);
availbl(winter)   =e=  produce(fall  ) - sale(fall  )+availbl(fall  );
availbl(spring)   =e=  produce(winter) - sale(winter)+availbl(winter);
```

It is obvious that the calculation is done in the closed cycle of seasons ignoring the transition from one season into another. Besides, it is clear that the variable *availbl()* has been calculated very accurately and the resulting value will not be affected by adding a constant number of seasons. However, the logic of the problem is important here.

11. ELEMENTARY NONLINEAR ALGEBRAIC EQUATIONS

11.1 Solution of Algebraic Equations (G. Kovalenko)

GAMS is very well suited to solving algebraic equations. The only difficulty is connected with defining all roots if there are many. The principle of finding a solution is the following:

The system of equations is written down first with variables that have to be defined included into the list of VARIABLES. Then, any additional variable or any variable from the system of equations is optimized. The result obtained will be in the form of sets of variables representing roots of the system of equations. See below an example of solving a system of equations.

Consider a simple example of a GAMS application for revealing typical structures and interactions of its components described above. For example, we have to solve two equations:

```
A * x^2  + B * x  +  C  =  y
M * x    + N              =  y
```

It can be rewritten in the form:

```
(A * x^2  + B * x  +  C ) - (M * x    + N ) = y' - y''
```

It is obvious that if the absolute value of (y' - y'') is equal to 0, then this equality is true when the sought-for roots are present in the system of equations. Let us make GAMS minimize the value (y' - y'') and define *x* for these minimum values.

```
SCALAR          zone for defining constants, parameters;
A  /  10/       entry of specific coefficients of the system;
B  /  10/       the equation to be solved;
C  /  10/       A * x^2  + B * x  +  C  =  y
M  / -10/       M * x    + N          =  y
N  / 100/;
VARIABLES       zone for defining the variables;
x               variable we are interested in;
o               variable characterized by the square difference between the functions obtained
                for the first and second equation;
EQUATIONS       zone for defining the names of equations;
alone;          there will be only one equation with the name “ALONE";
                zone for equations themselves and their arithmetic structures;

alone.. ((a*x*x+b*x+c)-(m*x+n))*((a*x*x+b*x+c)-(m*x+n)) =E= o;

x.LO =  -500;   low boundary for search of the variable x;
x.UP =   500;   upper boundary for search of the variable x;
```

x.L = -7.5; initial value of the variable x;

MODEL an /ALL/ the model is given the name AN and is comprised of all equations;

SOLVE an USING NLP MINIMIZING o; the command is given to solve the model AN by using nonlinear programming methods for minimizing the variable "o";

FILE res /result.dat/ Name the result file and correlate it with the internal name *res.*

PUT res; Make the internal file *res* a work file for results.

PUT " ATTENTION !!! "/; Enter a comment ATTENTION !!! and reset the line with the symbol /.

PUT x.L:8:2/; The result of the found variable *x* must be in 8:2 format. Symbol "/" means resetting.

Figure 1 shows three lines of special significance for understanding how GAMS works. The first two lines are functions with points of intersection to be found. The third line is a distance between the functions depending on x. In reality, this function characterizes the variable o, subject to minimization. It can be easily established that this system has two roots:

x' =-4.162; x" = 2.162;

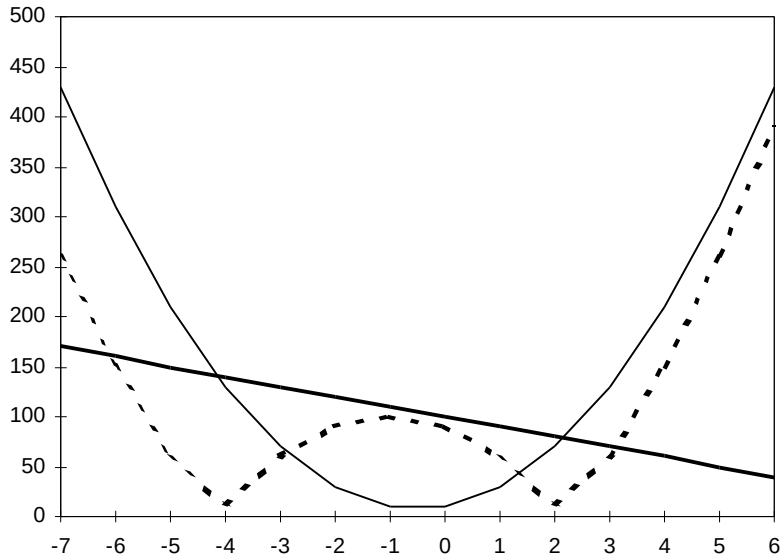

Fig.1. Chart for the Task of Solution of Equations

If the model is processed by GAMS, the following answer can be obtained: x" = 2.162. Where is x' =-4.162? You can obtain the answers by using the boundaries only:

x. LO = -500; Lower boundary for search of the variable x;

UP = -1.500; Upper boundary for search of the variable x;

x.L = -7.5; Initial value of the variable õ.

But if x.UP = -0.500 is chosen, then a very poor result will be obtained. In other words, there is no solution of the problem since the function does not take the same value in the area where argument *x* changes. The best convergence is noted on the boundary of the argument change. This very value is suggested for solution of the system of equations.

By considering this example we intended to show the GAMS users, that GAMS has its deficiencies, but they can be eliminated. Any solution found on boundaries is a clear sign of a faulty solution. By assessing the third line of Picture 1, you can find out why a poor result is obtained when the upper boundary is equal to -0.5, as well as why good results are obtained, if the boundary equals -1.5.

However, broader boundaries do not prevent making mistakes, since other complexities arise in this case. Now, lets consider other important elements of entering the listing of the model.

*.LST.

```
  1  SCALAR
  2  a  /10/
  3  b  /10/
  4  c  / 10/
  5  m  /-10/
  6  n /100/;
  7  VARIABLES
  8  x
  9  o
 10  EQUATIONS
 11  alone;
 14  alone..  ((a*x*x+b*x+c)-(m*x+n))*((a*x*x+b*x+c)-(m*x+n))=E= o;
 16  x.LO =  -500;
 17  x.UP =   500;
 18  MODEL an /ALL/
 19  SOLVE an USING DNLP MIMIMAZING o;
 22  FILE res /result.dat/
 23  PUT res;
 24  PUT "     ATTENTION !!!             "/;
 25  PUT x.L:8:2/;
```

Symbol Listing

```
A         PARAM  DECLARED       2  DEFINED       2      REF     2*14
ALONE       EQU    DECLARED      11  DEFINED      14 IMPL-ASN      19
                   REF       18
AN          MODEL  DECLARED      18  DEFINED      18 IMPL-ASN      19
                   REF       19
B         PARAM  DECLARED       3  DEFINED       3      REF     2*14
```

```
C          PARAM  DECLARED       4  DEFINED       4     REF    2*14
M          PARAM  DECLARED       5  DEFINED       5     REF    2*14
N          PARAM  DECLARED       6  DEFINED       6     REF    2*14
O          VAR    DECLARED       9 IMPL-ASN      19     REF      14
                  19
RES         FILE  DECLARED      22  DEFINED      22     REF      23
X          VAR    DECLARED       8 IMPL-ASN      19 ASSIGNED      16
                  17     REF     8*14      25
```

Consider, for example, M. The variable M is declared in line 5 (see listing above) and also defined in line 5. It is mentioned two times in line 14. The same goes for the rest parameters, variables, names and characteristics.

```
PARAMETERS      A,B,C,M,N
VARIABLES       O,X
EQUATIONS       ALONE
MODELS          AN
FILES           RES
```

You can observe that the system of the developed model is presented in the listing automatically with enumeration of its elements;

Then, a very important information comes for understanding of the text. However, we shall skip it for the purposes of simplification, leaving only the most necessary part of it.

```
                LOWER    LEVEL    UPPER   MARGINAL
---- VAR X     -500.000    2.162  500.000     EPS
---- VAR O       -INF        .      +INF       .
```

It can be easily observed that there is a figure between -500 and 500 in the column LEVEL. In the column MARGINAL we can see EPS (very small number). This indicates that the solution found is true. We know how the second solution can be found in this case. It depends on the experience of the user what should be done in other cases. Presence of some other symbols in the zone of MARGINAL is a bad sign. It is specially unpleasant if the word UNFES appears. It means that there is no optimum in the zone of search.

11.2 Method of Least Squares for Arbitrary Functions (Tikhonova, O.N.)

For example, consider the following problem:

Three values (dep1, dep2, dep3) are connected through a complex function with the fourth value (result). A function is defined by the user on the basis of the physical phenomenon under study. For example, there have been 8 simultaneous measurements of four values during 8 different time intervals. These 4 values are interconnected by the function:

$$a(dep1)^2 - \frac{d}{dep3} - \frac{b}{dep2} + e^{-(calc)^2} = calc$$

where a,d,b – unknown coefficients;
calc - the value that, if possible, should not differ from the value result.

Numerical values of the coefficients must be determined in the way best providing the connection between the four measured values or the coincidence of calc and result for 8 measurements. The last statement can be written down as a criterion function:

$$\sum_t (calc(t) - result(t))^2$$

Consider the model developed by GAMS technology:

Specify time periods of measurements (8, in this case):

```
SETS tn     time  / jan1,jan2,jan3,jan4,jan5,jan6,jan7,jan8 /;
SETS t(tn) time  / jan1,jan2,jan3,jan4,jan5,jan6,jan7,jan8 /;
```

Specify 8 series of measurements of the 3 values taken by the user as arguments of the unknown functional:

```
PARAMETERS
 dep1(t)   results /jan1  2, jan2  3, jan3  3, jan4  3, jan5  5, jan6  5, jan7  6, jan8  7/
 dep2(t)   results /jan1 30, jan2 60, jan3 70, jan4 60, jan5 80, jan6 90, jan7 100, jan8 100/
 dep3(t)   results /jan1  1, jan2  6, jan3  7, jan4  3, jan5  5, jan6  9, jan7   8, jan8  17/;
```

Specify 8 series of measurements of the 3 values taken by the user as a function.

```
PARAMETERS
 result(t) results /jan1 10,jan2 20,jan3 30,jan4 20,jan5 40,jan6 50,jan7  60,jan8  70/;
```

```
VARIABLES         declare 4 variables of free type
a,b,d,calc(t),obj;     and series calc(t) containing 8 elements.
```

```
EQUATION          declare names for equations (the first represents
olga1(t),          the system of 8 single-type of equations
olga2;             calculated for each concrete time period)
```

First equation:

```
olga1(t)..      a*dep1(t)*dep1(t)-d/dep3(T)-b/dep2(T) +
            EXP(-calc(t)*calc(t))  =E= calc(t);
```

Second equation:

```
olga2..        obj=E=sum(t,(SQR((calc(t)-result(t)))));
a.Lo = -100; a.UP =  100; b.lo = -100; d.LO = -101; d.up = -100;

MODEL OLGA / ALL /;                formulation of the model

SOLVE olga USING NLP MINIMIZING obj;  command for solving the minimization of
                                        deviation of calc and result  for all time periods.

FILE res /olga.out/                                          organization of the text
PUT "  ================================== "/;   file  olga.out
PUT "  CALCULATION a,b,d                  "/;   coment on structure
PUT "  a*X1*X1-d/X3-b/X2 = Y              "/;   of the design formula
PUT "  ================================== " ;
LOOP((t),PUT "   ",    organization of the display table by the LOOP operator;
PUT t.TL,                display of the name for each time period;
PUT  dep1(t):10:1,       output of the first argument;
PUT  dep2(t):10:1,       output of the second argument;
PUT  dep3(t):10:1,       output of the third argument;
PUT calc.L(t):10:1,      output of the computed value of the function;
PUT  result(t):10:1      output of the measured value of the function;
 /;);                    resetting of the line of the Table, and  ");" -  end LOOP.
PUT (a.L)      ;  output of the 3 known coefficients that provide
PUT (b.L)      ;  minimization of square deviations of the design calc  from
PUT (c.L)      ;  the measured result  for all 8 measurements.
```

12. ELEMENTARY PROBLEMS OF POWER NETWORKS

12.1 Problem of Power Generation, Distribution, and Consumption

Consider a power network comprised of two groups: the first group with power generation facilities and the second group for power consumption. The power transmissin lines can be used to transfer power to consumers. Figure 2 shows a network with power generation and power consumption nodes.

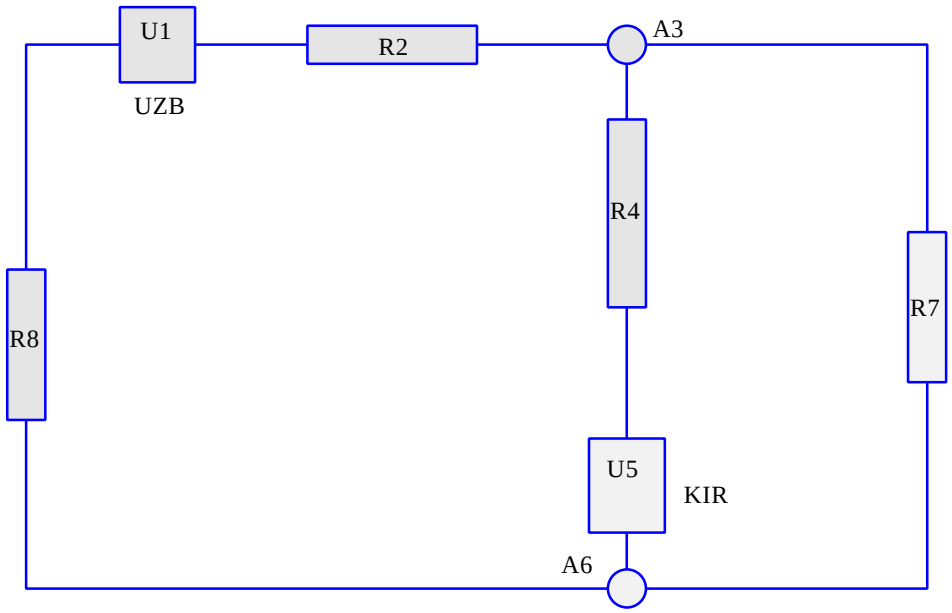

Fig. 2. Energy and Electricity Tasks Schematic

The value of power generation at node a5 is 100.
The value of power consumption at node a4 is 80.
The value of power consumption at node a8 is 30.
The value of power consumption at node a2 is 10.
The value of power transferred from node a8 to node a6 is 100.
The value of power transferred from node a2 to node a3 is 50.

The value of the power transfer in all lines of the network and the value of power generation at the node a1 must be determined. It is known that the power generation at this node is less than 900.

This problem is similar to the problem connected with power generation and power transfer between the nodes of the network. The essence of the problem becomes clear assuming that "a1" refers to

the set UZB, and "a5" to another set KIR. Besides, there is a condition on specified values of power transfer through power transmission lines of the network.

Consider the laws characteristic for the model:

1) The net flow through a power generating node (not through the power consumption node) is equal to 0, if the direction of power transfer is marked by + or -.

   Total value I(m,n) = 0;
   through n;

2) Flow from the first node in the direction of a second node of the circuit does not change. However, it has a "-" symbol for the first node, and "+" for the second node:

   I(m,n) = -I(n,m);

3) Flow transferred through a power generating node is increased by the value of the generated power:

   I(m,n)-I(n,k) = E(n);

4) Flow transferred through a power consumption node is decreased by the value of the consumed power:

   I(m,n)-I(n,k) = -R(n);

In the equations:

I(m,n)	flow from node "m" to node "n",
E(j)	power generation at node "j";
R(j)	power consumption at node "j";
m,n,k,j	refers to many nodes of the network.

On the basis of the equations 1) - 4) we can show that the total value of power generation is equal to the total value of the power consumption.

```
SETS
mt    /A1,A2,A3,A4,A5,a6,a7,a8/          names for all network nodes mt;
m(mt) /A1,A2,A3,A4,A5,a6,a7,a8/          identical sybset m;
ALIAS(M,M1);                             identical set m1;
SETS
mr(m)    /a2,a8,a4,a7/                   subset for power consumers;
me(m)    /a1,A5/                         subset for power generation facilities; m0(m)    /a3,a6/
                                         subset of transfer  nodes;
```

```
$ONTEXT;
      ---A1(uzb) -------A2------A3
      1                  1
      A8            A4      A7
      1                  1
      1            A5(kir)  1
      1                  1
      ------------------A6--------
$OFFTEXT;

M_and_M1(M,M1) /
a1.a2, a1.a8, a2.a1, a2.a3, a3.a2, a3.a4,    all nodes with their connections;
a3.a7, a4.a3, a4.a5, a5.a4, a5.a6, a6.a5,
a6.a8, a6.a7, a7.a3, a7.a6, a8.a1, a8.a6 /;

VARIABLES
e(m),          specified power generation area;
r(m),          specified power consumption area;
iii(m,m1),     specified power generation area;
obj;           criteria function not connected with the calculation;

EQUATION       equation names;
coni(m,m1),    regularities – 1;
conu0(m),      regularities – 2;
conue(m),      regularities – 3;
conur(m),      regularities – 4;
ben;           equation that does not affect the calculation.
                It is given only as a tribute to the GAMS rules
                for construction of models.
```

The reader can easily find out the regularities given in the theoretical description of the problem.

```
coni(m,m1)..          iii(m,m1) =E= -iii(m1,m);
conu0(m)$(m0(m)).. sum(m1$(M_and_M1(m,m1)),iii(m,m1)) =E=   0      ;
conue(m)$(me(m)).. sum(m1$(M_and_M1(m,m1)),iii(m,m1)) =E=   e(m)   ;
conur(m)$(mr(m)).. sum(m1$(M_and_M1(m,m1)),iii(m,m1)) =E=  -r(m)   ;
```

We will clarify only the following parts of equations:

```
$(m0(m))              - for energy transit nodes only;
$(me(m))              - for energy generation nodes only;
$(mr(m))              - for energy consumption nodes only;
$(M_and_M1(m,m1))- to be taken into account only in case there is a link between "m" and "m1"
                        nodes;

ben.. obj =E= 1;      The significance of the equation for the whole model
                        is quite clear;
```

```
e.UP('a1')      =900;   Less than 900 units of power can be generated at a1
                                node;
e.LO('a1')      =  0;
e.FX('a5')      =100;   100 units of power has been generated at a5 node;
iii.FX('a8','a6')=100;  100 units of power has been transferred in the
                                direction of a8-a6 ;
iii.FX('a2','a3')= 50;  50 units of energy has been transferred in the
                                direction of a2-a3 ;
r.FX('a4')      = 80;   80 units of power has been consumed at a4  node;
r.FX('a8')      = 30;   80 units of power has been consumed at a4  node;
r.FX('a2')      = 10;   80 units of power has been consumed at a4  node;
MODEL   an /ALL/;               definition of the model;
SOLVE   an USING NLP MINIMIZING obj;   command for calculation; FILE res /  den.out/
        organization of the printout command in the file den.out;
PUT res;                        embedded to the internal name res.
PUT "          ---A1(uzb) -------A2------A3        " /;
PUT "          1                       1       " /;
PUT "          A8              A4      A7      " /;
PUT "          1                       1       " /;
PUT "          1               A5(kir)  1       " /;
PUT "          1                       1       " /;
PUT "          ------------------A6--------        " /;
PUT " I  " /; PUT "          ";                     display of the comment;
LOOP(m, PUT m.TL:7; );              display of the table for energy transfers between the nodes;
PUT /;LOOP(m,  PUT m.TL:6;
LOOP(m1,PUT iii.L(M,m1):8:3;);    0 – no transfer, + transfer from the
                                        line lement
PUT /;);                                to the element of the column,
                                        (acceptance of the energy flow);
PUT /;  PUT "      Produce  " /;        display of the comment;
LOOP(m, PUT m.TL:6; );    PUT /;   display of names of nodes in the page
                                        header of the table;
LOOP(m, PUT e.L(m):6:2; );PUT /;        display of energy production values;
PUT /;  PUT "       Usage  " /;         display of the comment;
LOOP(m, PUT m.TL:6; );    PUT /;   display of names of nodes in the page
                                        header of the table;
LOOP(m, PUT r.L(m):6:2; );PUT /;   display of energy consumption values.
```

12.2 Defining Voltage and Strength of Current in Direct-Current Circuit

The problem is to define through specified elements the unknown variables of the d.c. circuit of any random configuration comprised of power sources and resistances. Consider the scheme similar to the scheme shown on Figure 2.

```
SETS
MT    /A1,A2,A3,A4,A5,a6,a7,a8 /    names for all network nodes mt;
M(MT) /A1,A2,A3,A4,A5,a6,a7,a8 /          identical sybset m;
ALIAS(M,M1);                              set m1; identical to “m”
ALIAS(M,M2);                        set m2; identical to “m”

$ONTEXT;
            ---A1--A2-----A3------
            1               1
            A8          A4     A7
            1               1
            1          A5     1
               1               1
            --------------A6------.
$OFFTEXT;

SETS
mr(m)  resisters       /a2,a8,a4,a7/      location of resistances;
me(m)  electro supplies   /a1,A5/      location of current sources;
m0(m)  knots           /a3,a6/           knots of the nodes;
M_and_M1(M,M1)              connection
 /a1.a2,a1.a8,      a2.a1,a2.a3,     A3.A2,A3.A4,A3.A7,
  a4.a3,a4.a5,      a5.a4,a5.a6,     a6.a5,a6.a8,a6.a7,
 a7.a3,a7.a6,       a8.a1,a8.a6  /         two-dimensional set defining the
                                           relationship between the nodes;
m1_m_m2(m1,m,m2);                         declaration of the three-dimensional set;
m1_m_m2(m1,m,m2)=YES$(m_and_m1(m1,m) AND m_and_m1(m,m2));
```

If there is a simultanous connection between the "m", "m1" and "m2", then there is a simalteunous connection between "m", "i1" and "m2".

```
m1_m_m2(m1,m,m1)$M1_m_M2(M1,m,M1)= NO;
```

However, if "m1" coincides with "m2", it is not a double simultaneous connection< but the connection based on the principle "TO- FROM". It is excluded from the whole set of triple connections.

```
VARIABLES
e(m),r(m),uuu(m,m1),iii(m,m1),OBJ;
                        where e  is a current source set;
                        r – resistance set;
                        uuu – matrix of voltages between ì  and ì1;
                        uuu – matrix of currents in the direction from ì to ì1;
                        obj –the irrational and inalterable variable for filling the
                        equation BEN;
```

```
EQUATION
coni(m,m1),             equation for calculation of currents;
conu(m,m1),             equation for calculation of voltages;
ii(m),                  law of Kirchhoff;
uu_0(m1,m,m2),          law of Ohm for nodes of electrical circuit;
uu_e(m1,m,m2),          law of Ohm for the node with the source of power supply;
uu_r(m1,m,m2),          law of Ohm for the node with resistance;
sor;                    name of the equation of no significance which is not
                        connected with solution of this model;
ii(m)..         sum(m1$m_and_m1(m,m1),iii(m,m1)) =e= 0;
coni(m,m1)..    iii(m,m1) =e= -iii(m1,m)  ;
conu(m,m1)..    uuu(m,m1) =e=  uuu(m1,m)  ;
uu_0(m1,m,m2)$m0(m)..
   0  =e=  (uuu(m,m1)-uuu(m,m2))$m1_m_m2(m1,m,m2);
uu_e(m1,m,m2)$(me(m)$(m1_m_m2(m1,m,m2)$( ord(m1) gt  ord(m2))))..
  uuu(m,m1)-uuu(m,m2) =e= e(m);
uu_r(m1,m,m2)$(mr(m)$m1_m_m2(m1,m,m2))..
   uuu(m,m1)-uuu(m,m2) =e= -r(m)*iii(m,m1);
sor..   obj =E=iii('a1','a8')*iii('a1','a8');

$ONTEXT;
sor..   obj =E=1;
```

There is no difference between the equation *sor*.. in the comments or *sor*.. in the model. The type and structure of equations is different, but the result is the same.

```
e.UP(m)    = 100; e.LO(m)    = -100;          In this part, the upper boundary is
                                              specified as equal to 100,
r.UP(m)    = 100; r.LO(m)    = -100;
iii.UP(m,m1) = 100; iii.LO(m,m1) = -100;  The low boundary as equal to
                                              100 for all model variables participating in the model;
uuu.UP(m,m1) = 100; uuu.LO(m,m1) = -100;
iii.L(m,m1)$M_and_M1(M,M1)     = -1  ;  Declare initial value of current for all
                                        the power transmission lines different from 0;
 r.FX('a2')      = 10   ;        Declare initial resistances in ohms;
r.FX('a4')      = 20   ;
r.FX('a7')      = 40   ;
 e.FX('a1')      = 20   ;        emf of power supply sources;
e.FX('a5')      = 40   ;
 iii.fx('a1','a2') = 0.125;      Declare specified value of current transferred from
                                 "a1" to "a2";
MODEL IVAN /ALL/;       Operator for assembling of the model from the equations SOLVE
IVAN USING NLP MINIMIZING obj;  Command on
                                 solving by using the methods of non-linear programming for
                                 minimization of OBJ.
FILE res /ivan.out/             Display for the text file "ivan.out" through
```

```
                                connecting the internal file name res. to the outside file.  The res. file
                                becomes active.
PUT /;PUT " M_and_M1" /;                                comment;
LOOP(m,  PUT m.TL:6;
       LOOP(m1, PUT M_and_M1(m,m1):7; );PUT /;);  display of matrix for
                                                    interconnections between the nodes;
       PUT /;PUT " M1_m_M2" /;                      comment;
LOOP(m1,                                            display;
LOOP(m,LOOP(m2,PUT M1_m_M2(m1,M,m2):7;);PUT /;); serius of tables
                                                    characterizing triple links;
PUT /;
);
PUT "          ---U1-----R2------A3-------          " /;     display of the scheme
PUT "          !              !       !       " /;            as a comment
PUT "          R8               R4       R7       " /; for visual checking of links
                                                    between the nodes;
PUT "          !              !       !       " /;

PUT "          !              U5       !       " /;
PUT "          !              !       !       " /;
PUT "          ------------------A6--------        " /;
       PUT " I amper  " /;PUT "          ";            comment;
      LOOP(m, PUT m.TL:7; );PUT /       ;            display of names of nodes for
                                                    the heading of the lower table;
LOOP(m,PUT m.TL:6;LOOP(m1,PUT iii.L(M,m1):7:3;); PUT /;);
```

Display of the table of currents on the condition that each cell contains the current moving from one node located in the line to the node of the column.

```
PUT /;PUT " U volt " /;PUT "          ";
LOOP(m, PUT m.TL:7; ); PUT /;
LOOP(m,PUT m.TL:6;LOOP(m1,PUT uuu.L(M,m1):7:3;);PUT /;);
```

The same as above but for voltages. Display of information on emf.

```
PUT /;PUT " E volt on supplies " /;PUT "          ";
LOOP(m, PUT m.tl:6; );PUT /;
PUT "      ";LOOP(m, PUT e.l(m):6:2; );PUT /;
```

Display of information on resistance in ohms.

```
PUT /;PUT " R - resisters  " /;PUT "          ";
LOOP(m, PUT m.tl:6; );PUT /;
PUT "      ";LOOP(m, PUT r.l(m):6:2; );PUT /;
```

13. SIMPLE TASKS FOR WATER MANAGEMENT IN RIVER SYSTEMS

13.1 Optimal Management of a Monocapacitive Reservoir

The task unites two consistent mathematical models in which the design information from the first model gets through to the second model. The mathematical model main objective is an optimal limitation of raised demands for water releases under water deficit conditions. Optimality means a proportional reservoir water release limitation.

It is known that:

1) The curve demonstrating relationships of reservoir levels and storages given (even if fragmentary) in the tabular form. It will be needed for calculation of losses by evaporation, available water surfaces and levels.
2) A reservoir inflow forecast for all estimated time intervals
3) Water release demands for a lower reach for all estimated time untervals
4) Ivaporative capacity for all estimated time intervals
5) A reservoir initial filling and a desired filling in definite time intervals.

Define:

Mode of reservoir water level and storage changes, that corresponds to optimal proportional release limitations for a water deficit case.

The calculation objective for the first model: to find such aa,bb coefficients, that if to place them in function aa*(level)^bb, then the storage will be a result corresponding to this level. We would like to remind you that the computer knows both the reservoir level and storage, and it may define their correspondence by the label (r1,r2,r3 ...r10).

It is obvious, there is none of the guarantee that we will receive the needed storage. Thus, we will compute this storage, subtract it from the actual storage and memorize it under the symbol "îî", we will make the computer choose the àà, bb coefficients, so that"îî"be the least. It will be better if the difference is squared, otherwise, the differences will be minor, i.e. negative.

Calculation equations:

1) Wr(i) = aa* H(i) ^bb
2) LIM{ [Wr(i) - Wf(i)]^2} = 0 aa = ?, bb =?

where:

i series number for the level and storage measurements;
Wr(i) -a design storage corresponding to the given level;;
Wf(i) -actual storage corresponding to the given level.

The calculation objective for the second model:optimal (prescribed by a corresponding criteria) management of a monocapacitive reservoir.

Calculation equations:

1) Wk(j) = Wn(j) + Win(j) - Wout(j) - e(j)*S[H(j)]
   H(j) = F[{Wk(j)+Wn(j)}/2,aa,bb]
2) LIM{ [Wout(j) - Wdem(j)]^2} => 0 Wout(j) = ?
the latest criterion equation may be presented in a modified shape.

where:
Wk(j) – reservoir water storage for the end of the time
Wn(j) – reservoir water storage for the beginning of the estimated time interval
Win(j) – inflow volume
Wout(j) – water release volume (design)
e(j) – evaporativity
S[H(j)] – reservoir water storage as a water level function
Wdem(j) – water release demands for a lower reach
H(j) = F[{Wk(j)+Wn(j)}/2,aa,bb] – the function of a relationship of a water level in the reservoir and average water storage, proper for the given estimated time interval. The function is defined analytically, according to the function and coefficients from the first part of the model.

Both models are presented in one text file, and the bound (boundary) between them is just symbolic.

```
SCALAR vmax  /1000/;          the constant input:ìàõ reservoir storage
SCALAR vmin  /10/;   the constant input:ìin reservoir storage
SCALAR ho    /100/;  input of a number of estimated time intervals
SCALAR num   /18/;
SCALAR what /1 /;    the constant which will determine whether or not the releases and demands
                     should be converted to storages
SCALAR cek  /1/      number of seconds in the time interval
SETS                 identification of a number of variables in the array
rk                   they are ten in array rk, and they have names
                     r1 r2 r3 r4 r5 r6 r7 r8 r9 r10,
   / r1*r10/         but now they are mixed, and it is not clear which follows which;
rek(rk)              identification of a number of variables in the array
/ r1*r10/;           they are ten in the array rek, and they have names
                           r1 r2 r3 r4 r5 r6 r7 r8 r9 r10.
```

The computer understands which follows which, it is clear that the second variablefollows the first, and r8 follows r7. These are similar to the numbers of soldier symbols, the machine knows them by names, it knows they exist, and it knows who follows whom.

```
PARAMETER            Input of direct characteristics of the information file.
ur(rek) /
r1   0.1,            number 0.1 takes up the position under rl index
r2   1,         number 6 takes up the position under r4,
```

r3 0, etc.
r4 6, These numbers are levels which the machine now remembers,
r5 7, but each figure allegedly has umber **r1, ...,r10**
r6 10,
r7 15, r8 20, r9 25,
r10 30/;

PARAMETER Input of direct characteristics of the information file
vr(rek) /.
r1 0.1, number 1 takes up the position under **r2** index
r2 1, number 330 takes up the position under **r7**,
r3 0 etc.
r4 50, These numbers are storages which the machine now remembers
r5 110, but each figure allegedly has number **r1**, ...,**r10**
r6 220, moreover, levels and storages are allegedly combined
r7 330, into pairs.. They are unified by the labels **r1, r2, r3**, and etc.
r8 660,
r9 880, r10 1990/;

VARIABLES a function word indicating declaration of names of the variables;
aa,bb,oo; names of the variables subject to definition.
We would like to remind you that **aa** and **bb** are the constants by which according to the formula W = aa*H^bb storage W may be defined.
H - level.

EQUATIONS a function word indicating declaration of the equation names;
result; equation name
result.. oo =E= SUM(rek$((ur(rek)>0)and(vr(rek)>0)),
(((aa*ur(rek)**bb-vr(rek))*(aa*ur(rek)**bb-vr(rek)))**2));
This equation may be better explained by parts.
SUM(rek,ur(rek)) – calculate the sum for the case when the element of the array **ur(rek)** is more than zero (it is implied that the element is defined), and it is defined for the row rek, i.e for all r1,r2,r3, ... r10. $ - is similar to the word “if”
sum(rek$(condition), array): The sum of the array elements should be considered only if the condition in brackets has been satisfied.
sum(rek$((ur(rek)>0)and(vr(rek)>0)),array): only those cases (r1,r2,r3, .. r10) will be summed up for which both elements of the pairs = level (ur) and storage (vr)= are simultaneously (“AND”)above zero.
OPTION ITERLIM= 3000; It is allowed to make only 3000 assumptions;
OPTION OPTCR=0.000000001; Calculations should be advanced until the 0.000000001 accuracy is reached.
It is known that the variables sought for =(aa) and (bb)= are available within certain bounds, thus, we will roughly make:
LO - lower bound
UP – upper bound
aa.LO= 0.0001; aa.UP=100.1; bb.LO= 0.0001; bb.UP=100.01;

MODEL gog /result/; The operator who will form a model out of the equations.

SOLVE gog USING NLP MINIMAXING oo; A command to solve the model problem on the
FILE res /result.dat/ computer having displayed the method of the;

PUT res solution; the objective of the solution is the output of the computation
information into the text file *result* which has an internal name *res*,
the name has been made active for the output.
PUT " manner (1 or 0) 1-in runof 0-dicharge "/ ; comments
PUT what:16:0/; output of the parameter respective to the comments
PUT " cek in interval "/ comments
PUT cek:16:0/; output of the parameter respective to the comments
PUT " quantity of interval "/ ; comments
PUT num:16:0/; output of the parameter respective to the comments
PUT " dead level (ho) "/ comments
PUT ho:16:2/; output of the parameter respective to the comments
PUT " koefficients(a/b) in formula w = a * (H-ho)** b "/ ;
PUT aa.L:16:4/; output of the variables **aa** and **bb**. **L** index shows
PUT bb.L:16:4/; that the solutions are found.
PUT " Morfology "/ comments
PUT " level-ho volume(real) volume(calc) difference"/;
The table with the lines containing the prescribed level and storage are taken out for printing. In the columns you have the level and storage number values which you inserted. The computation storage and its deviations from the actual storage are also introduced into the columns.
LOOP(rek,PUT rek.TL:8, ur(rek):10:2, vr(rek):10:2,
((aa.L*ur(rek)**bb.L)$((ur(rek)>0)and(vr(rek)>0))):20:2,
((aa.L*ur(rek)**bb.L-vr(rek))$((ur(rek)>0)AND(vr(rek)>0))):10:2
/;);
SCALAR beg_hajm /100/; declaration and definition of the initial storage
SCALAR a /0.29/; declaration and definition of the scalar **a**
SCALAR b /0.01/; declaration and definition of the scalar **b**
a=aa.L*bb.L/(aa.L**((bb.L-1)/bb.L)); computation through the variables **aa** and **bb**, scalars
a and **b** of the new model which are found in the previous
model. They are made in such a way
b=(bb.L-1)/bb.L; that the available surface of the reservoir may be defined by equation S =
a*W^b. The formula
dW / dz = S(H) is used in which **z** is a vertical coordinate; and
d is a differential.
SETS The function word SET for introducing indices;
dk / h1*h18/ Definition of 18 time indices
dek(dk) / h1*h18/; (a year with the step of 12, or half a year with the step of 10 days
may be considered).
PARAMETER The function word PARAMETERS for the mass data input
Kelish(dek) / Input of a forecast inflow into the reservoir for 18 time intervals
h1 12, h2 10, h3 10, h4 8, h5 8, h6 8, h7 6, h8 6, h9 6,
h10 4, h11 4, h12 4, h13 8, h14 10, h15 10, h16 10, h17 10, h18 10/
Talab(dek) / Input of the reservoir release demands for 18 time intervals

```
    h1   6, h2  15, h3  14, h4  13, h5   12, h6  11, h7  10, h8   9, h9  8,
    h10  7, h11  6, h12  5, h13  4, h14   3, h15  2, h16  2, h17  2, h18 2/
  ee(dek) /       Input of evaporativity in meters for a time interval
    h1  0.0016, h2   0.0015, h3  0.0014, h4  0.0013, h5  0.0012, h6  0.0011,
    h7  0.0010, h8   0.009 , h9  0.008 , h10 0.007 , h11 0.006 , h12 0.005,
    h13 0.004 , h14  0.003 , h15 0.002 , h16 0.002 , h17 0.002 , h18 0.002/;
 Talab(dek) =Talab(dek)  *cek;        Do or don't we need to convert discharges into storages?
 Kelish(dek)=Kelish(dek) *cek;        If cek=1, then we do not.  If cek is equal to the number of
                                      seconds in the estimated time interval, then we do.
POSITIVE VARIABLES      Names of variables which may be only positive
hajm(dek),
chikish(dek);
VARIABLES               This scalar variable may be only of a free type (GAMS
Bajar;                  requirement)
EQUATIONS               Definition of equation names
baj,                    the first is scalar, and the second is 18 monotype
Balans(dek);            equations.
Balans(dek)..           Equation of a reservoir water budget
          0 =E=               The following mix of items may equal to zero:
  Hajm(dek-1)$(ord(DEK) GT 1)+ Former storage if the time interval is not the first;
    beg_hajm$(ord(DEK) EQ 1)- Initial storage if the time interval is the first;
          Hajm(dek)+          subtract the data of a reservoir final storage
  Kelish(dek)-chikish(dek)-   add inflow and subtract  release data, and subtract
                              evaporation losses, i.e.
  ee(dek)*                    evaporativity multiplied by the average available water
(a*((((Hajm(dek)+Hajm(dek-1))*0.5)$(ord(DEK) GT 1)+   surface amount for the estimated
 ((Hajm(dek)  + beg_hajm)*0.5)$(ord(DEK) eq 1))**b));  time interval.
 The $ operator defines either the first or not the first time interval to register the difference between
the initial and a former water storage in the reservoir.
Baj..   Bajar    =E=       the difference between a water release to a lower reach and a demand
                            for this release is computed in this equation.
SUM(DEK,(((chikish(dek)-Talab(DEK))*(chikish(dek)-Talab(DEK))+0.01)**0.3));
$ontext;        Two options of other criterion function constructions are given in this model as
                            comments.
Baj..   Bajar    =e=(0.00000001+
PROD(DEK,(((1.0001-chikish(dek)/Talab(DEK))** 4.0))))*
SUM(DEK,(((chikish(dek)-Talab(DEK))*(chikish(dek)-Talab(DEK))+0.01)**2.1));
Baj..   Bajar    =e=
   SUM(DEK,(((chikish(dek)-Talab(DEK))*(chikish(dek)-Talab(DEK)))**0.1));
$offtext;
  Hajm.UP(dek)=vmax;        Assignment of bounds for a possible flow regulation by the
  Hajm.LO(dek)=vmin;        reservoir.
* Hajm.FX('h1 ')=130;
* Hajm.FX('h2 ')=130;       Till the symbol "*" stands in the first position
* Hajm.FX('h3 ')=130;       the given line serves as a comment.
* Hajm.FX('h4 ')=130;
* Hajm.FX('h5 ')=130;       If to take away an asterisk from some lines, then the given
```

```
* Hajm.FX('h6 ')=130;          variables are not subject to computation and are not fixed
* Hajm.FX('h7 ')=130;          under optimization.  The remained variables will be computed
* Hajm.FX('h8 ')=130;          fully considering present circumstances.
*Hajm.FX('h9 ')=130;
* Hajm.FX('h10')=130;
* Hajm.FX('h11')=130;
* Hajm.FX('h12')=130;          If to raze out all asterisks, then the optimization model
* Hajm.FX('h13')=130;          enters the simulation mode.
* Hajm.FX('h14')=130;
* Hajm.FX('h15')=130;
* Hajm.FX('h16')=130;
* Hajm.FX('h17')=130;
 Hajm.FX('h18')=130;          The measured up reservoir storage after optimal amount of water has
                              been released, and which appears to be an additional constraint for flow
                              regulation.  Actually, it is a reservoir guaranteed final filling.
chikish.LO(dek)=0.0001;Upper, lower and finite (algorithmic) values of reservoir water releases.
chikish.L(dek)=1;
*chikish.UP('h1 ')=0.0;
*chikish.UP('h2 ')=0.0;          Full analogy with the fixed variable described above.
 ... ... ... ... ... ... ...
 ... ... ... ... ... ... ...
 ... ... ... ... ... ... ...
*chikish.UP('h4 ')=0.0;          ATTENTION!  There exists real riskiness not to gain the desired
                              results, but it is not stipulated in the model.  For example, total inflow is 100,
*chikish.UP('h16')=0.0;          initial filling is 10, final storage is 130.  The result will be
*chikish.UP('h17')=0.0;          a declaration that it is impossible to find a solution.
*chikish.up('h18')=0.0;
OPTION ITERLIM = 3000;   3000 iterations are allowed
OPTION OPTCR   =0.000000001;    Accuracy of optimum definitions
OPTION LIMROW  =   12;           Output into the listing of the information about 12 time
                              intervals;
MODEL gg / baj,balans/;          Definition of the model structure
SOLVE gg USING  NLP MINIMIZING bajar;        A command for the solution with selection of
                                          methods and an objective.
PUT Bajar.l:16:4/;      Criteria output.
PUT  "Balance from GAMS in English     "/;          Comment.
PUT  "      Max_volume       Min_level "/;          Table header
PUT Vmax:16:2,Vmin:16:2/;                Marginal storage
PRESENTATION OF THE TABLE HEADER AS COMMENTS
UT "Begin    volume    and    level      "/;
PUT  "          V              V        "/;
PUT beg_hajm:16:4,(ho+(beg_hajm/aa.L)**(1/bb.L)):16:2/;
PUT  "          Volume   Volume     Input" ;
PUT  "    Output     Needs     Evap   square   %    Level"/;
PUT  "          begin      end           " ;
PUT  "                           average      average "/;
PUT  "          mln.m3   mln.m3    mln.m3" ;
```

```
PUT "   mln.m3   mln.m3   mln.m3   mln.m2   %     metr"/;
 LOOP(dek,
PUT dek.TL                                  : 8,          Lead-out of
(hajm.L(dek-1)+beg_hajm$(ORD(dek) EQ 1))              : 9:2, ap
hajm.L(dek)                                 : 9:2, prop
kelish(dek)                                :10:2, riate
chikish.L(dek)                              :10:2, columns
Talab(dek)                                 :10:2,  of the table
(ee(dek)*(a*((((Hajm.L(dek)+
Hajm.L(dek-1))*0.5)$(ORD(dek) GT 1)+
((Hajm.L(dek)+beg_hajm)*0.5)$(ORD(dek) EQ 1))**b)))     : 9:3,
((a*((((Hajm.L(dek)+Hajm.L(dek-1))*0.5)$(ORD(dek) GT 1)+
((Hajm.L(dek)+beg_hajm)*0.5)$(ORD(dek) EQ 1))**b)))     : 9:2,
(chikish.l(dek)/Talab(dek))                       : 6:2,
(ho+((Hajm.L(dek)/aa.L)**(1/bb.L)))                    : 8:2
 /;);
```

13.2. Optimal Management of a River System (Sources, Reservoirs, Customers, River Mouths)

Figure 3 represents an elementary river system in which sources, reservoirs, consumers, and river mouths are available. It is required to re-allocate the available water resource to satisfy optimally (according to the criteria of optimality) water customers' demands.
The task is based on the graph theory. A conservative water transportation is made according to the flow sheets (graphs). In the nodes (depending upon their belonging to) modification of water resource occurs, compliant to particular laws.

Thus, for each kind of a node entrance Win is defined and exit Wout is computed.

Besides, the fifth special equation calculates Win as a sum of Wouts for all node outputs, those nodes have an input connection with the design node.

Potential water diversion for consumer's needs is considered by subtraction, the consumer has an output tie with the design node.

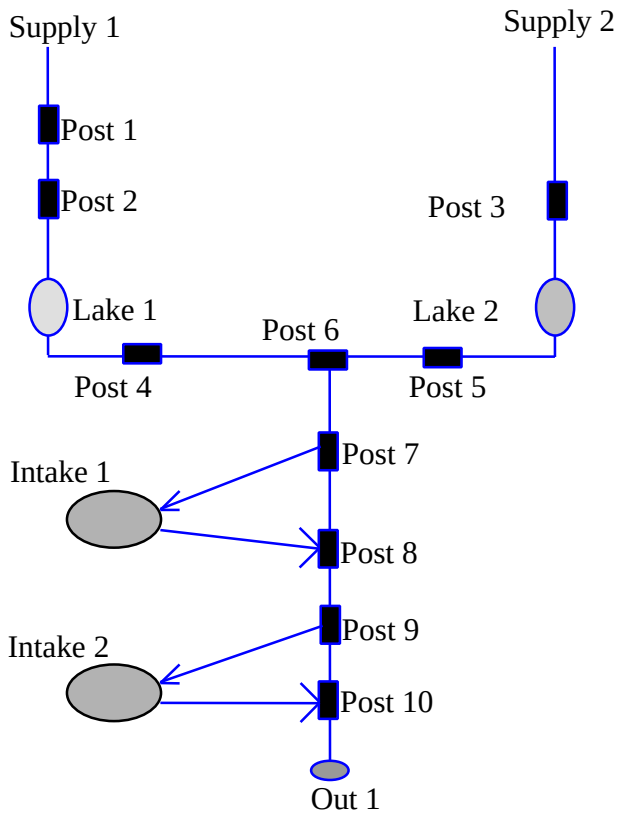

Fig. 3. Structure of a River Network for the Water Management Task

Main calculation equations for each time interval are as follows:

1) Wout = Win' Win' for the sources, for which the Win' logon is preassigned;
2) Wout = r* Win for customers, for flow, and river mouth (r - coefficient);
3) Wout = Win'+V(t+1)-V(t) for reservoirs;
4) Wout = ñóìià Wout for ordinary nodes the output is equal to the sum of outputs from the nodes having a log-in connection with the design node.
5) Win = ñóìià Wout - Win

CRITERION FUNCTION

6) lim [Win - Wdem] => min Win It is necessary to define Win for a river mouth and users for all **t**.
It is necessary to define Wout for reservoirs for all **t**.

In the fourth equation indices **t+l** and **t** are shown, in other equations index **t** is missing for simplicity purposes. Besides, constraints for maximum and minimum storages in the reservoirs, and in some cases for transit water discharges, were predetermined.

SETS
nt nodes /post_1*post_11,supply_1,supply_2,lake_1, input of a node set
lake_2,out_1,intake_1,intake_2,total /
n(nt) nodes /post_1*post_11,supply_1,supply_2,lake_1, identical
lake_2,out_1,intake_1,intake_2 / subset of nodes

```
 ns(n)    supplies  /supply_1,supply_2       / subset of sources
 nn(n)    simple    /post_1*post_11          / subset of simple nodes
 nr(n)    regions   /intake_1,intake_2,out_1 / subset of all users
 nrr(n)   regions   /intake_1,intake_2       / subset of users
 nl(n)    lakes     /lake_1,lake_2           / subset of reservoirs
 rlast(n) last user /out_1                   /; subset of river mouths

  ALIAS(n,n1);ALIAS(n,n2);            Input of identical subsets n1 and n2

SETS              Definitions and logistics of the subset connections entering
n_from_n(n,n1)    /the node n from the nodes n1 are easy to see in Fig.3.
post_1.supply_1,  post_2.supply_2,  post_3.post_1,    lake_1.post_3,
lake_2.post_2,    post_4.lake_1,    post_5.lake_2,    post_6.post_4,
post_6.post_5,    post_7.post_6,    intake_1.post_7,  post_8.post_7,
post_8.intake_1,  post_9.post_8,    intake_2.post_9,  post_10.post_9,
post_10.intake_2, out_1.post_10/;

SETS              Definitions and logistics of the subset connections outgoing
n_to_nr(n,n1) /             from the design node n to users n1.
 post_7.intake_1, post_8.intake_2,  post_10.out_1/;

SET     Definitions and logistics of a subset for design time intervals
  m  month  /eenndd,jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec /
 mm(m) /eenndd/  isolation of the first time interval into a subset
          Declaration and input of hydrographs for water content of sources

TABLE supp(N,m) water supplies     (mln cub.m.)
         jan   feb   mar   apr  may  jun  jul  aug   sep   oct   nov   dec
supply_1  128   125   234   360  541  645  807  512   267   210   181   128
supply_2  39.0  39.0  52.0  121  168  144  105  78.0  49.0  44.0  45.0  39.0

Declaration and input of hydrographs for the declared supply limits.

TABLE rreq(N,m) water requirements (mln cub.m.)
         jan  feb  mar   apr   may   jun    jul   aug    sep   oct  nov  dec
intake_1  0.1  0.1  0.1  64.5  109.8 184.4  243.7  200.9  99.5  0.1  0.1  0.1
intake_2  0.1  0.1  0.4  13.5   15.0  22.1   26.0   24.9  13.0  3.1  0.1  0.1
out_1    1000 1000 1000  1000  1000  1000   1000   1000  1000  1000 1000 1000

PARAMETER           Declaration and input of coefficients for a return flow.
 ret(n)     koeffic_of_returns /intake_1 0.5,intake_2 0.5,out_1 0.0/

POSITIVE VARIABLES       Declaration of names of variables for the model
                         provided that they are positive..
       intake(n,m)       water intake from a node
       in_flow(n,m)      inflow in a node
       ou_flow(n,m)      outflow from a node
```

```
      vol(N,M);                reservoir storage

VARIABLES          Function word preceeding the name of free variables;
obj;               Name of a free variable, this variable is also a criterion one;

EQUATION           Declaration of equation names
  ou_flow_no(n,m)  for calculation of total inflow into the node
  ou_flow_ns(n,m)  for sources
  ou_flow_nr(n,m)  for users
  ou_flow_nl(n,m)  for reservoirs
  ou_flow_nn(n,m)  for ordinary nodes
  ben;             for computation of a criterion function.

$ONTEXT;
-------------------------------------------------------------------------
$OFFTEXT;

ou_flow_no(n,m)$(nn(n)$(NOT mm(m)))   Equation for a subset of common nodes
 ou_flow(n,m) =E= in_flow(n,m);       and for none of the first time interval
ou_flow_ns(n,m)$(ns(n)$(NOT mm(m))).. Eguation for a subset of sources
 ou_flow(n,m) =E= supP(n,m);          and for none of the first time interval
ou_flow_nr(n,m)$(nr(n)$(NOT mm(m))).. Equation for a subset of users
 ou_flow(n,m) =E= ret(n)*intake(n,m); and for none of the first time interval
ou_flow_nl(n,m)$(nl(n)$(NOT mm(m))).. Equation for a subset of reservoirs
ou_flow(n,m)  =E=-vol(n,m)          and for none of the first time interval
+vol(n,m-1)+in_flow(n,m);
ou_flow_nn(n,m)$(not mm(m)).. Equation for computation of total inflow into the node
in_flow(n,m)  =E=                   considering water diversion
+sum(n1$(n_from_n(n,n1)),ou_flow(n1,m)) and for none of the first time interval
-sum(n1$(n_to_nr(n,n1)),intake(n1,m ));
ben..  Equation for computation of a criterion function built up according to the principle
       for calculation of total relatively satisfied declared demands
obj=E= sum(m,sum(n$nrr(n),(intake(n,m)/(rreq(N,m)+0.00000001))))+
sum(m,sum(n$rlast(n),(intake(n,m)/(rreq(N,m)+0.00000001))))/1000;
intake.LO(n,m)          = 0.0;       Input of lower water diversion boundary data
intake.UP(n,m)          = rreq(N,m); Input of upper water didersion boundary data
vol.LO('lake_1' ,m   )   = 100;  Input of the data of a reservoir storage lower boundary
vol.UP('lake_1' ,m   )   = 1590; Input of the data of a reservoir storage upper boundary
vol.FX('lake_1' ,'eenndd')  = 1200;  Input of reservoir initial filling data
vol.L( 'lake_1' ,m   )   = 1100;  Input of the initial point for finding decisions
vol.LO('lake_2' ,m   )   = 100; Input of the data of a reservoir storage lower boundary
vol.UP('lake_2' ,m   )   = 1590; Input of the data of a reservoir storage upper boundary
vol.L( 'lake_2' ,m   )   = 1100; Input of reservoir initial filling data
vol.FX('lake_2' ,'eenndd')  = 1200; Input of the initial point for finding decisions
OPTIOM ITERLIM= 1000;  Setting of the marginal number of iterations
OPTION LIMROW =12;     Number of outputs into the listing file of all 12, but not four, by
                       default, design time intervals
```

```
MODEL dan    /ALL/;                    Formation of the model out of equations

SOLVE dan    USING  NLP MAXIMiZING obj;  A command for calculation, indicating the
                                         calculation method and optimization objectives.

FILE res /result.dat/              Organization of the calculation information output into
PUT res;          the text file result.dat, which has the internal name res, active for the output.
Logistics for output into a table of the information mix relating to the amount of water which flew
in and flew out from the node.
LOOP((m),PUT
in_flow.L('post_5 ', m):6:1, ou_flow.L('post_5 ', m):6:1,
in_flow.L('post_6 ', m):6:1, ou_flow.L('post_6 ', m):6:1,
in_flow.L('post_7 ', m):6:1, ou_flow.L('post_7 ', m):6:1,
in_flow.L('post_8 ', m):6:1, ou_flow.L('post_8 ', m):6:1,
in_flow.L('post_9 ', m):8:3, ou_flow.L('post_9 ', m):8:3,
in_flow.L('post_10', m):7:2, ou_flow.L('post_10', m):6:2,
 intake.L('out_1   ',m):7:1                          /;);
PUT /; line reset
Logistics for output into a table of a water budget for both reservoirs
LOOP((m),PUT
in_flow.L('lake_1',m):7:1,vol.L('lake_1',m):7:1,ou_flow.L('lake_1',m):7:1,
in_flow.L('lake_2',m):7:1,vol.L('lake_2',m):7:1,ou_flow.L('lake_2',m):7:1/;);
PUT /; line reset
Logistics for output into a table of water intake budget components
LOOP((m),PUT
 intake.L('intake_1',m):7:1,ou_flow.L('intake_1',m):7:1,
 intake.L('intake_2',m):7:1,ou_flow.L('intake_2',m):7:1
/;);
```

14. OPTIMAL SELECTION OF AGRICULTURAL CROPS (Alex Meuraus)

The task is valuable as it is composed by the GAMS originator, it is the first time that the task is presented for demonstartion.

Task:
Assigned:
1) Types of agricultural crops
2) Computation period (a year, using a step of one month)
3) Required irrigation rates for each crop for each specific month
4) Stress coefficient for each crop and for each month.
   (The physical sense of this coefficient is to what degree the yield of a given crop will decrease in case of a relative poor irrigation in each specific time interval.)
5) Price for a product unit of each crop
6) All yield for each crop
7) A proposed area to allocate the given crops
8) Water hydrograph

It is required:
1) To determine, how much land should be allocated for each crop within the resources available
2) To identify irrigation hydrographs for each crop within the resources available

Objective: Maximize proceeds from the yield sales for each crop

The mathematical model is represented by the equations:

$$1)\ R = \text{ñóìì}\left(\tilde{N}(p)*S(p)*Y(p)*\text{ïðîèç}\left(\left(\frac{Wr(p,t)}{Wd(p,t)}\right)\right)\right);$$

$$2)\ \text{ñóìì}\ S(p) < So$$

$$3)\ \text{ñóìì}\ Wr(p,t) < Wo(t)$$

Where:

p,t	- crop and a time step index;
S(p)	- land area for the given crop;
Wr(p,t)	- reception of water for each crop in each moment;
Wd(p,t),l(p,t)	demands and a stress coefficient for each crop in each moment;
So	area of land bresource;
Wo(t)	hydrograph of water resource;
Ñ(p)	price for a crop yield unit;
S(p)	area for each crop;

Y(p) all yield for each crop.

```
SETS p  plants / cotton, rice, wheat, maize, others/  crops
    m  month  / jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec / time

                Table that presets water requirements
TABLE wreq(p,m) water requirements (mill m3 per ha)
        jan  feb  mar  apr  may   jun   jul  aug  sep  oct  nov  dec
cotton                    1.2   1.2   1.2  1.2
rice                     2.9   2.8   2.9  2.8  2.6
wheat                     0.8   0.8   0.9  0.8  0.7
maize                     1.8   1.9   1.9  1.8  1.8
others                    1.1   1.1   1.1  1.1  1.1

                Table that presets a stress coefficient
TABLE l(p,m) stress coeffs      ( - )
        jan  feb  mar  apr   may   jun   jul  aug  sep  oct  nov  dec
cotton                    1.0   1.0   1.0  0.9
rice                     0.9   0.8   0.9  0.8  0.6
wheat                     0.8   0.8   0.9  0.8  0.7
maize                     0.8   0.9   0.9  0.8  0.8
others                    1.0   1.0   1.0  1.0  1.0

after the function word "PARAMETERS"
   The first line defines the price for one centner of each crop;
   The second line defines maximum yield for each crop;
   The third line declares water hydrograph availability, but does not define it.
PARAMETERS
  price(p) sales prices  (money per c) / cotton 3, rice 5, wheat 7, maize 11,others 60/
  yield(p) optimal yield (c per ha)    / cotton 2, rice 3, wheat 4, maize  5, others 3/
  watersup(m) water supply (mill m3 per month);
  watersup(m) = UNIFORM(900,1100); The water hydrograph, declared just now,
                        is computed for each time interval as a variate
                        between numbers 900 and 1100.

SCALAR land  available land (ha) / 1000 /;  land area is defined and assigned;

                Three variables are subject to computation:
VARIABLES  obj  beneifit  (money) benefit from sales
        s(p) land cultivated (ha)         land cultivated for each crop
        walloc(p,m) water allocation (m3 per ha);      water resource and its allocation
                                                       among the crops during certain time.
        Three equations will be presented:
EQUATION   ben      benefit definition        equation calculating benefit
        landbal  land balance (ha)            land balance
        waterbal(m) waterbalance (ha);        water balance
```

Benefit from the yield sales is computed by formula-1
from the model built below according to the GAMS technology rules..

```
ben.. obj =E= sum(p,price(p)*s(p)*yield(p)*
         prod(m$wreq(p,m),(walloc(p,m)/wreq(p,m))**l(p,m)));
```

Total area of cultivated lands is always less than the available land resource.

```
landbal.. sum(p, s(p)) =L= land;
```

Amount of water used for irrigation is always less than the water available.

```
waterbal(m).. sum(p, s(p)*walloc(p,m)) =L= watersup(m);
```

```
MODEL andrei / ALL /; All represented equations participate in the model

s.L(p) = land/CARD(p);  Initial allocation of land among crops is equal
s.LO(p) = 0.1; Let a little land (lower bound) be cultivated initially for any crop.
walloc.L(p,m)  = watersup(m)/s.l(p); Let water be initially shared among crops.
walloc.LO(p,m) = wreq(p,m)*0.1;    Let 90 % of demand be a maximum constraint for actual
                                   irrigation.
walloc.UP(p,m) = wreq(p,m);Let water be delivered in the appropriate amount,
                           not more than it is required.
OPTION DOMLIM=1000;

SOLVE andrei USING NLP MAXIMiZING obj; Solve the model using unlinear programming
                                       methods to maximize the variable OBJ.

PARAMETER stress(p,m) water stress;     Introduce a new parameter named stress(p,m)
  stress(p,m)$(wreq(p,m) AND l(p,m)) =  Compute the parameter stress(p,m)
        walloc.L(p,m)/wreq(p,m);        by the formula which correlates delivered water with
                                   the declared demand.  These stresses actually provide
                                   maximum profit.
```

Calculations are made only for the cases, in which water demands and a stress coefficient are defined simultaneously (See condition $).

```
DISPLAY stress,s.L;  Output of the obtained results into the listing
```

Listing is a file with extension LST.
Each time the model will produce new results due to a casual formation of water resource.

15. References

1. Brooke, A., D. Kendrick, A. Meeraus, and R. Raman, GAMS Language Guide, RELEASE 2.25, Version 92, GAMS Development Corporation, Washington D.C., 1997

2. McKinney, D.C. and X. Cai, Multiobjective Water Resource Allocation Model for Toktogul Reservoir, Technical Report, US Agency for International Development, Environmental Policy and Technology Project, Central Asia Regional EPT Office, Almaty, Kazakstan, April, 1997.

3. Savitsky A.G. "Optimal control of water, land and hydropower resources in Central Asia region by modern modelling technologies help". Material ICID, Portigal, Lisbon, September 1998. (13 p.)