MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY
"KHARKIV POLYTECHNIC INSTITUTE"

O. Zakovorotniy

O. Lipchanska

FUNDAMENTALS OF COMPUTATIONAL INTELLIGENCE

Part 2

Laboratory workshop
for carrying out laboratory work
for full-time and part-time students
by speciality Computer Engineering and Computer Science

Approved by
editorial and publishing
council of the NTU "KhPI" University,
protocol № 12 dated 29.12.2021.

Kharkiv
NTU "KhPI"
2022

UDC 004.89
ББК 32.973-02
З 19

Reviewers:
V. D. Kovalov, Doctor in Technical Science, Professor, Rector of Donbass State Machine-building Academy, Laureate of the State Prize of Ukraine in the field of science and technology;
G. F. Kryvulia, Doctor in Technical Science, Professor, professor of the Computer Engineering Design Automation department, Kharkiv National University of Radio Electronics.

Наведено теоретичні відомості про пакет MATLAB та способи побудови в ньому нечітких множин, алгоритмів нечіткого виведення й нечіткої кластерізації, нечіткого контролера, генетичних алгоритмів і нейронечітких гібридних мереж. Теоретичний матеріал підкріплений великою кількістю прикладів із використання описаних нечітких систем.

Призначено для студентів денної та заочної форм навчання за напрямом "Комп'ютерна інженерія" й "Комп'ютерні науки".

Іл. 64. Табл. 12. Бібліог. 12 назв.

Zakovorotniy O. Fundamentals of Computational Intelligence / O. Zakovorotniy, O. Lipchanska: Laboratory workshop. – Kharkiv: NTU "KhPI", 2022. – 152 p. – In engl. language.

З 19

ISBN 978-617-578-129-6

Theoretical information about the MATLAB package and methods for constructing fuzzy sets, fuzzy inference and fuzzy clustering algorithms, a fuzzy controller, genetic algorithms and neuro-fuzzy hybrid networks in it is given. The theoretical material is supported by a large number of examples on the use of the described fuzzy systems.

Designed for full-time and part-time students in the direction of "Computer Engineering" and "Computer Science".

Pict. 64. Tabl. 12. Referenc. 12 names.

UDC 004.89
ББК 32.973-02

ISBN 978-617-578-129-6

INTRODUCTION

The *Fuzzy Sets Theory* was developed in 1965 when professor Lotfi Zadeh of the University of Berkeley published the fundamental work Fuzzy Sets in the Information and Control journal. The adjective "fuzzy" was included into the title of the new theory in order to distance itself from classical crisp mathematics and Aristotelian logic, which operate with crisp concepts: "belongs – does not belong", "true – false". The concept of a fuzzy set was born by L. Zadeh as dissatisfaction with the mathematical methods of classical systems theory, which forced his to achieve artificial accuracy, inappropriate in many systems of the real world, especially in the so-called humanistic systems that include people"

The beginning of the practical application of fuzzy sets theory can be considered 1975, when Mamdani and Assilian built the first fuzzy controller to control a simple steam engine. In 1982, Holmblad and Ostergad developed the first industrial fuzzy controller, which was implemented in the control of the cement firing process at a plant in Denmark. The success of the first industrial controller based on fuzzy linguistic rules "If – then" led to a surge of interest in fuzzy set theory among mathematicians and engineers. Somewhat later, Bartholomew Kosko proved the *Fuzzy Approximation Theorem*, according to which any mathematical system can be approximated by a system based on fuzzy logic. In other words, with the help of natural language statements-rules "If – then", with their subsequent formalization by means of fuzzy set theory, it

is possible to accurately reflect an arbitrary relationship "inputs - output" without using the complex apparatus of differential and integral calculus, traditionally used in management and identification.

Systems based on fuzzy sets have been developed and successfully implemented in such areas as process control, transport management, medical diagnostics, technical diagnostics, financial management, stock forecasting, pattern recognition, etc. The range of applications is very wide - from video cameras and household washing machines to air defense missile guidance and control of combat helicopters. Practical experience in the development of fuzzy logical inference systems shows that the time and cost of their design is much less than when using the traditional mathematical apparatus. This ensures the required level of robustness and transparency of models.

Laboratory work 9

FUZZY CONTROL OF DYNAMIC PROCESSES

The purpose of the laboratory work is to obtain and to consolidate knowledge about methods of designing fuzzy controllers, to form practical skills in managing complex objects or dynamic processes.

9.1. Summary of theory

9.1.1. Introduction to the theory of fuzzy control

Fuzzy inference methods, in addition to the tasks of constructing expert systems, are widely used in the development of fuzzy controllers. The main purpose of a controller is to control an object, in which the behavior of a controled object is described by fuzzy rules.

Let Y be the main parameter of the object;

Y^* is desired parameter value;

$E = [Y^* - Y]$ is current state error;

U is control action generated by the controller;

$U(K)$ is value of control action;

$$U(K) = F(U(K-1), ..., U(K-\tau), E(K-1), ..., E(K-\tau));$$

τ is the depth of the background consideration.

Usually fuzzy controller works with depth $\tau = 1$:

$$U(K) = F(U(K-1), E(K-1)); \text{ or } U(K) = F(U(K-1), \Delta E(K-1));$$

where ΔE is change of the error, and the rule for generating the control action is set using the fuzzy rule base. Usually the control action takes into account the

previous action and the last change of the error.

The control process is described by a set of variables belonging to the class of linguistic variables. The entities of a controlled process can be classified as "input variables" and "output variables" of a process. In general, input variables define time-dependent streams, while output variables describe instantaneous process states.

Input variables that can be controlled by humans or computers are called process controlled variables. Control in a broad sense means manipulating a process in a feedback loop in order to obtain the best possible result, i.e. obtaining the optimal value of certain output variables.

For management it is necessary:

1) determine the current values of the significant output variables;

2) compare current output values with given target values;

3) select the values of the controlled variables so as to achieve the target values;

4) repeat step 1 – 3 until the target values are obtained.

Consider a fuzzy controller. Moreover, the controlled process has three input variables with two controlled variables (input 2 and input 3) and one random variable (input 1), as well as three output variables. The fuzzy controller, according to a certain algorithm, converts three output variables into two control variables, which are then returned to the control object. Comparison of output values with target values is done by the controller itself.

Unlike most standard controllers that use mathematical calculations, a fuzzy controller applies so-called rules, like a human operator.

Example 9.1. Control rules:

– IF the pressure is low, THEN open the tap

(chemical production process).

– IF the loan interest is high, THEN reduce borrowing

(financial planning process).

– IF the power consumption is low, THEN reduce the generator output

power (power management process).

The terms in roman are the names of object variables, the words in italics are the fuzzy values of the corresponding variables. Similar rules are used by humans in the management process. An experienced operator understands the meaning of fuzzy terms: "low pressure" or "slightly open the tap", while the automatic control device works with clear values: the output variables of many technical processes are processed by analog-to-digital converters, control variables are processed by digital-to-analog converters; both types of devices are reduced to processing digital values.

So, to implement a fuzzy controller, you need to::

– convert the crisp values of the output process variables into fuzzy ones;

– use the rules for converting fuzzy output variables into fuzzy control values;

– convert fuzzy values of control variables to crisp ones.

For each variable A of the process, we define the membership function $\mu(a, f)$, which associates any crisp value of variable *A* with a fuzzy one.

9.1.2. Rules and implication

Rules

The rules in the fuzzy controller system have the form IF f_0, THEN g_0, where f_0 and g_0 are any fixed fuzzy values of some output variable *A* and some control variable *B*, respectively.

Example 9.2. Fuzzy controller.

Let the process contain:

– three output variables with fuzzy values:

1) pressure (low, medium, high);

2) temperature (cold, rather warm, warm, hot);

3) humidity (dry, wet, damp);

– two control variables with fuzzy values:

1) water supply (absent, average, maximum);

2) heating intensity (absent, low, medium, maximum);

– два two rules:

1) IF the pressure is low THEN the valve is half open.

The f_0 value is pressure low means that the fuzzy value of the output variable "pressure" is low. Value g_0 is valve half open that means that the "valve" is half open.

2) IF the temperature is cold AND the humidity is damp THEN the heater, the maximum.

The premise of the f_0 rule is temperature is cold AND humidity is wet combines the fuzzy values of the output variables "temperature" and "humidity". The value of g_0 is a heater, maximum means that the fuzzy value of the control variable "heater" is maximum.

The f_0 value in the conditional part of the rule can be either a simple fuzzy value of one output variable, or a combination of fuzzy values of several output variables. The fuzzy value of g_0 in the final part of the rule almost always represents the simple fuzzy value of only one control variable.

Implication

Rules like "IF f_0, THEN g_0" are called implications in Boolean logic, where f_0 and g_0 are Boolean entities with truth values of 0 or 1. Formally, the implication is written in the form $f_0 \rightarrow g_0$ each pair of values; it returns the result according to Table 9.1.

Table 9.1 – Truth table

f_0	g_0	$f_0 \rightarrow g_0$	HE $f_0 \vee g_0$
0	0	1	1
0	1	1	1
1	1	1	1
1	0	0	0

It is seen from table 9.1 that the operation $f_0 \rightarrow g_0$ is equivalent to (NOT f_0) $\vee$ g_0.

In fuzzy logic, the implication of fuzzy values f_0 and g_0, $f_0 \rightarrow g_0$ is defined as a new combined fuzzy value.

For example, fuzzy implication: (NOT f_0) OR g_0.

Membership function:

$$\mu(a, b, f_0 \rightarrow g_0) = \mu[a, b, (\text{NOT } f_0) \text{ OR } g_0] = S[1 - \mu(a, f_0), n(b, g_0)],$$

where a and b are any two base values of object variables A and B, respectively $f_0 \in A$ and $g_0 \in B$, $\mu(a, f_0)$ and $n(b, g_0)$ are partial membership functions A and B.

Mapping process state and fuzzy controller rules

Consider a process with an output variable A, a control variable B and a rule linking the fuzzy values f_0 of entity$A$ and g_0 of entity$B$ (Table 9.2).

Table 9.2 – Characteristics of the controlled process

Value	Input variable A	Control variable B
Base value	$\{a\}$	$\{b\}$
Fuzzy value	$\{f\}$	$\{g\}$
Membership function	$\mu(a, f)$	$n(b, g)$
Selected fuzzy value	$f_0 \in \{f\}$	$b_0 \in \{g\}$
Representing crisp values	$\alpha(t) \in \{a\}$	$\beta \in \{b\}$
The rule	IF f_0 THEN b_0	Does not exist

At any time t the crisp value $\alpha(t)$ of the output variable A is compared with the fuzzy value f_0, i.e. with the membership function $\mu(\alpha, f_0)$, and at any time t, and for any crisp value b of the control variable B, the pair (α, b) is compared with the rule $f_0 \rightarrow g_0$ = (NOT f_0) OR g_0 with membership

$$\mu(\alpha, b, f_0 \rightarrow g_0) = S[1 - \mu(\alpha, f_0), n(b, g_0)].$$

According to the condition, it is necessary to consider the conjunction of a fuzzy meaning and a rule, namely f_0 AND $[f_0 \rightarrow g_0] = f_0$ AND [(NOT f_0) OR g_0].

At any time t and for any crisp value b from the set B, the pair (α, b) belongs to a new fuzzy set with membership

$$\rho(\alpha, b, f_0 \text{ AND } [(\text{NOT } f_0) \text{ OR } g_0]) = T\{\mu(\alpha, f_0), S[1 - \mu(\alpha, f_0), n(b, g_0)]\}.$$

For a fixed time t and a fixed value $\alpha(t)$ this function depends on b.

Choosing a crisp value for the control variable

For any fixed $t = t_0$ the crisp value is $b = \beta(t_0)$. The control variable B is chosen so that the membership value of the pair $[\alpha(t_0), b]$, that is, the value of the expression

$$\rho(b) = f_0 \text{ AND } [(\text{NOT } f_0) \text{ OR } g_0] = T\{\mu(\alpha, f_0), S[1 - \mu(\alpha, f_0), n(b, g_0)]\}$$

was the maximum for $b = \beta(t) = b_{\max}$.

Consider the use of function $\rho(b)$ in a situation where the process output corresponds to the conditional part of the rule.

Let us assume that the output variable has the required value with high confidence. This means that $\mu \approx 1$, and therefore, $\rho(b) \approx\approx T\{1, S[0, n(b, g_0)]\} = T\{1, n(b, g_0)]) = n(b, g_0$According to the control law, the value of b should be chosen so that the membership $n(b, g_0)$ is maximal.

Consider a situation in which the output of a process more or less conforms to the rule.

Suppose that the values of the output process variables are more or less acceptable ($\mu \leq 0{,}5$), then

$$S[1 - \mu, n(b, g_0)] \geq \max[1 - \mu, n(b, g_0)] \geq 0{,}5 \geq \mu$$

and therefore

$$\rho(b) = T\{\mu, S[1 - \mu, n(b, g_0)]\} \le min\{\mu, S[1 - \mu, n(b, g_0)]\} \le \mu.$$

The smaller the value μ, the lower the degree of membership $\rho(b)$ for any b; therefore, the choice of b is less justified, therefore, the rule for determining the control variable is only more or less acceptable.

Consider the situation when the output of the process is unsatisfactory ($\mu \approx 0$).

The function $\rho(b) \approx T\{0, S[1, n(b, g_0)]\} = T\{0, 1\} = 0$ regardless of b, and the choice of b is not suitable for any value. According to daily experience, rules whose conditions are not met do not use.

9.1.3. Combining conditions

Rules with combined conditions are similar to the following

IF the pressure is low OR the pressure is medium THEN the valve is open

or

IF the temperature is cold AND the humidity is damp THEN the heater, maximum

are handled similarly. It does not matter whether conditions that reference one or more output variables are combined. In all cases, for crisp output values $\alpha_1(t)$, ..., $\alpha_N(t)$ at time t the degree of membership $\mu(\alpha_1(t), \ldots, \alpha_N(t), f_0)$ to the combined fuzzy value f_0 is determined. Further calculations are carried out as follows: the function $\rho(b) = T\{\mu(\alpha_1(t), \ldots, \alpha_N(t), f_0), n(b, g)\}$ is determined, and the optimal value b.

9.1.4. Accumulation of results and dephasing

Aggregating the results of multiple rules

Consider many rules for driving a car.

Example 9.3. Aggregating rule results:

IF the obstacle is close, THEN the speed is slow.

IF time is short THEN speed is fast.

IF the date is the end of the week THEN the speed is fast.

IF fuel is empty THEN stop speed.

IF funds are not enough THEN speed economy.

All rules contain an output variable "speed" with fuzzy values "stop", "slow", "economy" and "fast".

All rules contain an output variable "speed" with fuzzy values "stop", "slow", "economy" and "fast".

Compliance with the rules can lead to inconsistent or conflicting conclusions, depending on which conditions worked in a particular situation. If, for example, time and money are both small, should the car go "fast" and "economically"? If you use several rules at the same time, you need to decide which one is true. The OR operation in this context means "exclusive OR".

A rule with the condition f_0 and the conclusion g_0 is equivalent to a fuzzy value (f_0 AND g_0). At any time t and for a certain b, a crisp output value of the process is obtained from the membership functions $\rho(b) = T\{\mu(\alpha_1(t), \ldots, \alpha_N(t), f_0), n(b, g)\}$ for f_0 AND g_0. Here are some rules that apply together and are equivalent to a combination of fuzzy values:

$$(f_1 \text{ AND } g_1) \text{ OR } (f_2 \text{ AND } g_2) \text{ OR } \ldots \text{ OR } (f_N \text{ AND } g_N)$$

with membership functions $\rho(b) = S\{\rho_1(b), \rho_2(b), \ldots \rho_N(b)\}$, where

$$\rho_1(b) = T\{\mu(\alpha_1(t), \ldots, \alpha_N(t), f_0), n(b, g_1)\};$$
$$\rho_2(b) = T\{\mu(\alpha_1(t), \ldots, \alpha_N(t), f_0), n(b, g_2)\};$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$\rho_N(b) = T\{\mu(\alpha_1(t), \ldots, \alpha_N(t), f_0), n(b, g_N)\}.$$

In the above expression $\rho(b) = S\{\rho_1(b), \rho_2(b), ... \rho_N(b)\}$ at any fixed time t, the function b differs from the optimal value $b = \beta$.

Defuzzification

Next, we will consider how to obtain a crisp optimal value β of the control variable b. This procedure is called defuzzification and is the last step of each control cycle in a fuzzy system. In general, the variable $b = \beta$ has a partial membership function

$$\rho(b) = S\{\rho_1(b), \rho_2(b), ... \rho_N(b)\} = S\{T[\mu(\alpha_1(t), n(b, g_1)], T[\mu(\alpha_2(t), n(b, g_2)], \ldots, T[\mu(\alpha_N(t), n(b, g_N)]\}.$$

For defuzzification of a fuzzy result, in practice, three methods are used, which we will consider below.

Maximum height method. It is applied if the function $\rho(b)$ has a certain absolute maximum, like some value $b = b_{max}$, at which $\rho(b_{max}) > \rho(b)$ for all b. The advantage of the method is its simplicity of application, and the disadvantage is that the function $\rho(b)$ is not considered in the entire domain of definition, but only in maximum positions, which can lead to unacceptable results.

Average maximum method. If the function $\rho(b)$ has several relative maxima, then a weighted average can be taken. Considering the general form of the expression $\rho(b)$, this method is better than the previous one. It is also applicable if $\rho(b)$ is constant over certain intervals b (which often happens in practice), since one can take the centers of each such interval as "relative local maxima" and then average these values.

Center of gravity method. The graphical representation of the function $\rho(b)$ in Cartesian coordinates is a curve that limits a flat area by the upper and lower boundaries of b and with the X-axis. The abscissa β of such a figure gravity center is taken as the desired optimum

$$M = \int_c^d b \cdot \rho(b)\, db, \quad F = \int_c^d \rho(b)\, db, \quad b_{opt} = M/F.$$

If $\rho(b)$ is determined only for discrete values of b (the so-called singletons), then the calculation of the center of gravity is modified as follows:

$$M = \sum_n b_n \cdot \rho(b_n)\, db, \quad F = \sum_n \rho(b_n)\, db, \quad b_{opt}=M/F.$$

9.2. Fuzzy control systems for dynamic processes

9.2.1. Simulation of a ball rolling on a seesaw

Entering the *slbb* command in the *Command Window* of the MATLAB package results in a block diagram of the fuzzy control system appearing in the *Simulink* window (Figure 9.1). The task of control in this case is to maintain such a state of the ball rolling on a seesaw, in which it could not roll off them.

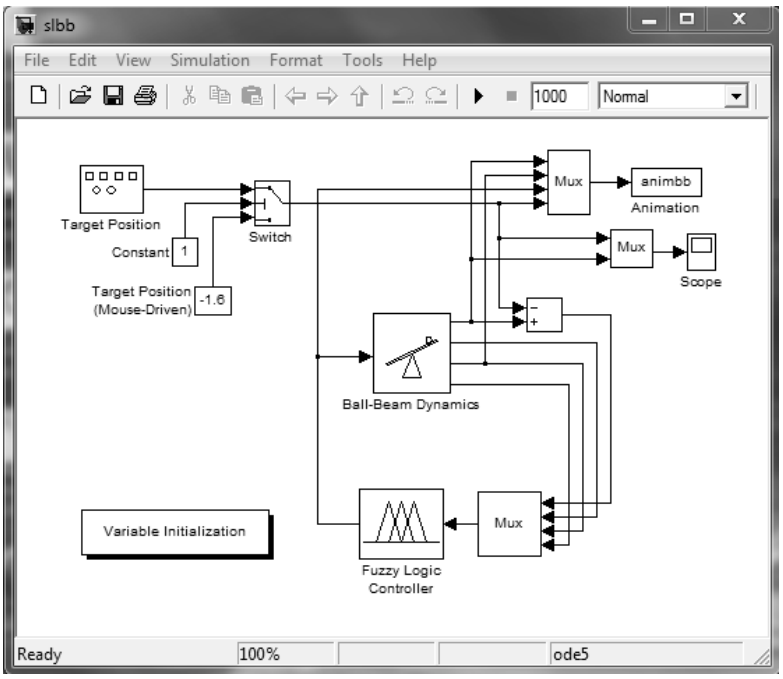

Figure 9.1. Block diagram of a fuzzy control system

Starting the simulation leads to the appearance of an animation picture (Figure 9.2) illustrating how the presented system, generating control actions, "pushing" either the left or the right half of the seesaw, does not allow the ball to

roll off them. The push point is represented by the apex of a triangle moving along the seesaw arm.

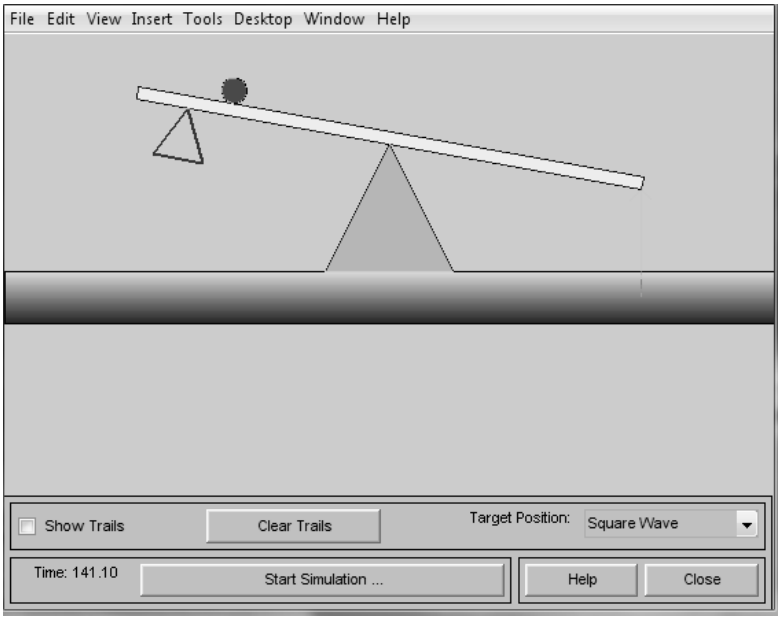

Figure 9.2. Dynamic system "ball on seesaw"

If desired, the user can view the main signals of the system (Figure 9.3).

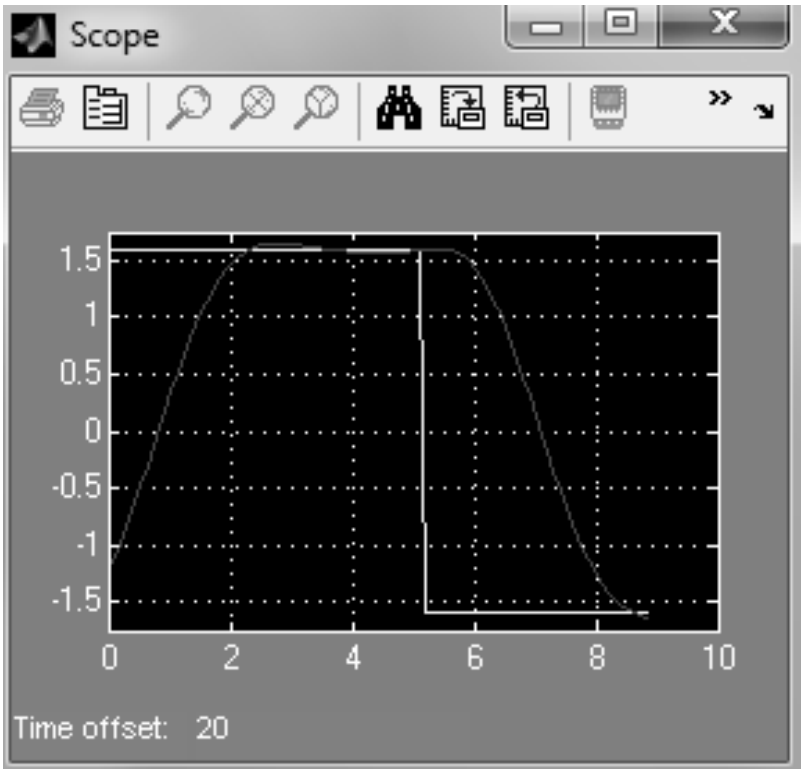

Figure 9.3. The main signals of the system are the value of the required and current position of the ball on the seesaw

Double-clicking on a block of the model diagram allows you to display a window with the parameters of the corresponding block. So, Figure 9.4 shows such a window for a fuzzy logic block. It is easy to see that it is characterized by a single parameter *the file name* that specifies the structure of the block.

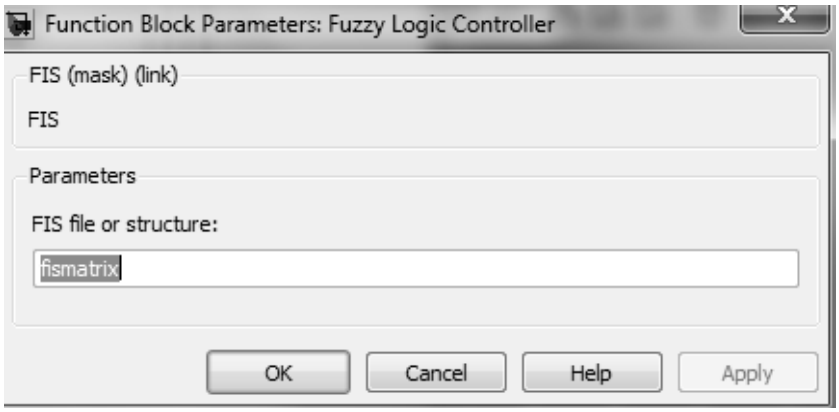

Figure 9.4. Окно блока нечеткой логики

The *Target Position* list (Figure 9.2) allows you to select one of four types of actions:

– *Sinusoid Wave* – sine wave;
– *Square Wave* – square wave;
– *Saw Wave* – triangle wave;
– *Mouse-Driver* – action set by the mouse.

Show Trails option allows you to accumulate changes in the position of the ball, seesaw arms and the point of impact on them. Thanks to this, you can get a clearer idea of the dynamics of the analyzed system. An example of the accumulation of information about the position of the ball, seesaw arm and points of influence is shown in Figure 9.5.

9.2.2. Simulation of ball bounces from a seesaw

Similar to the above example named *juggler* simulates more complex ball movement (Figure 9.6).

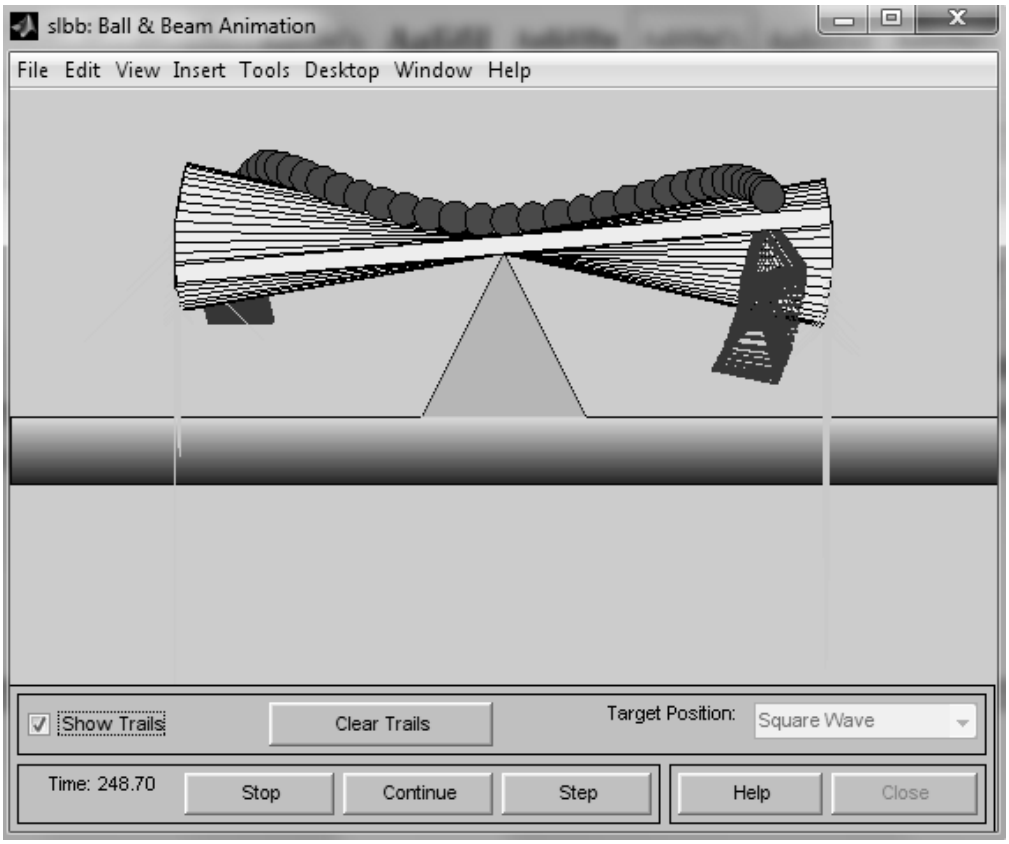

Figure 9.5. Illustration of a controled object in accumulation mode

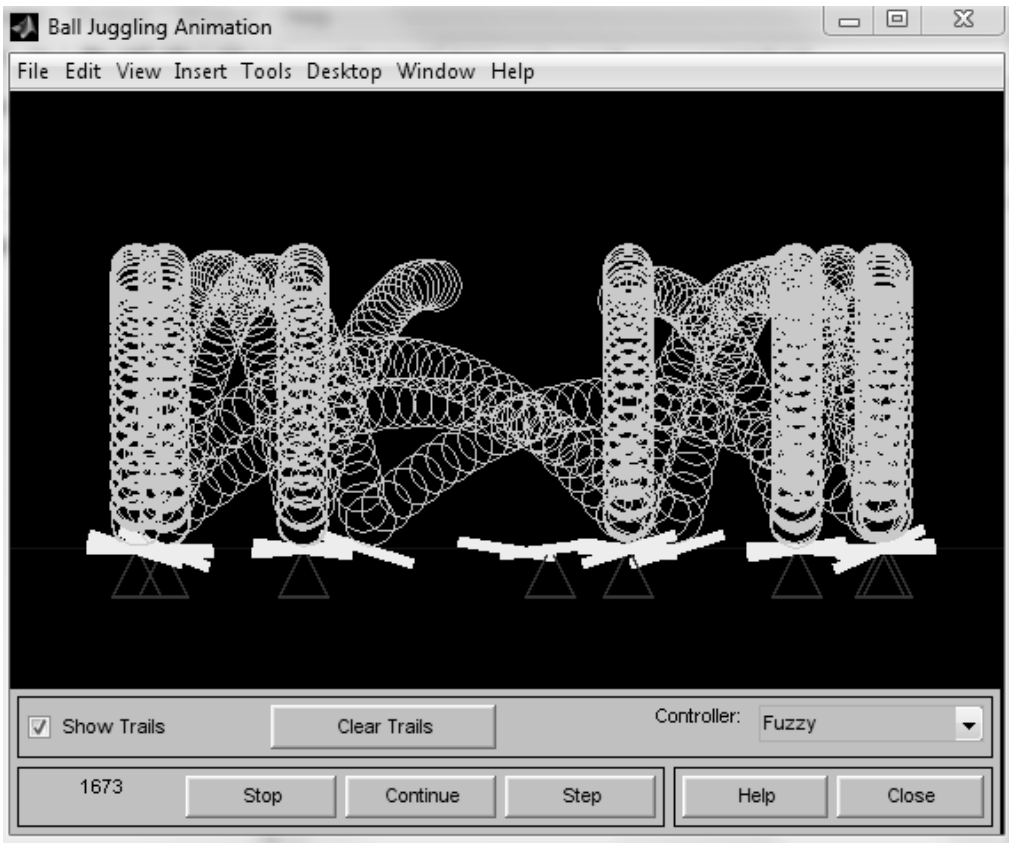

Figure 9.6. Ball bounce dynamics when using fuzzy logic

Here the ball bounces off the arms of the seesaw, which moves to the point where the ball falls. In this case, the initial position of the arm changes in such a way that the ball, if possible, bounces vertically upwards. Ejection of the ball outside the range of movement of the seesaw is excluded taking into account bounces from the left or right walls.

In this example, the ability to simulate the ball rebound from the seesaw arm in two modes using fuzzy logic and a mathematical model is especially valuable. The latter case is shown in Figure 9.7. It is easy to see that in this case it is generally not possible to achieve the ball bouncing strictly vertically upwards because the rule "the angle of reflection is equal to the angle of incidence" applies.

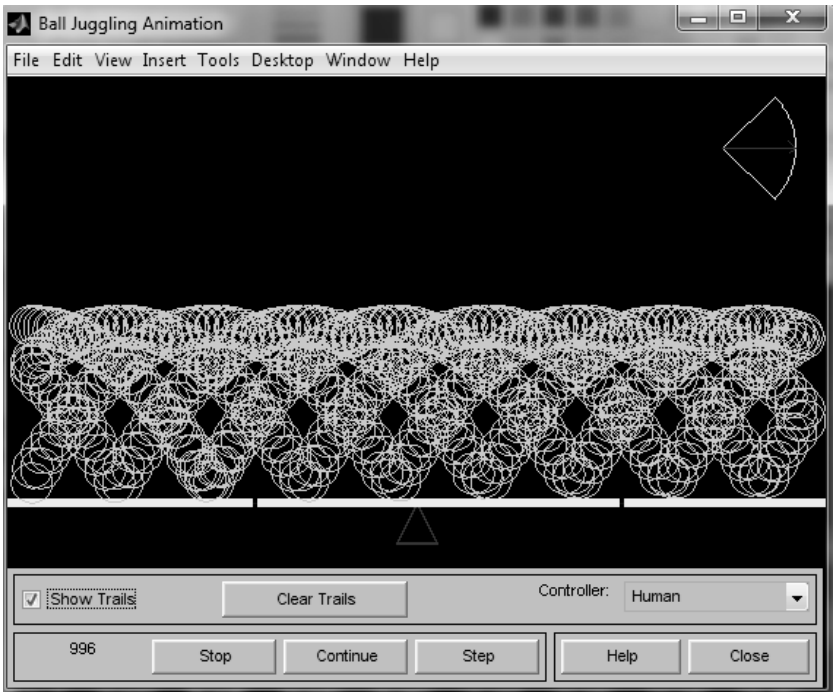

Figure 9.7. The dynamics of ball bounces using a mathematical model

Along with the model window, this example displays the *Rule Viewer*, shown in Figure 9.8. It allows you to specify the parameters of the ball bounces.

This example was implemented without using *Simulink*. The MATLAB code of the program that implements it can be viewed by activating the *View*

code for juggler hyperlink in the demo window of this example with the mouse. The program takes a little more than 700 lines and is viewed in the program editor of the MATLAB system.

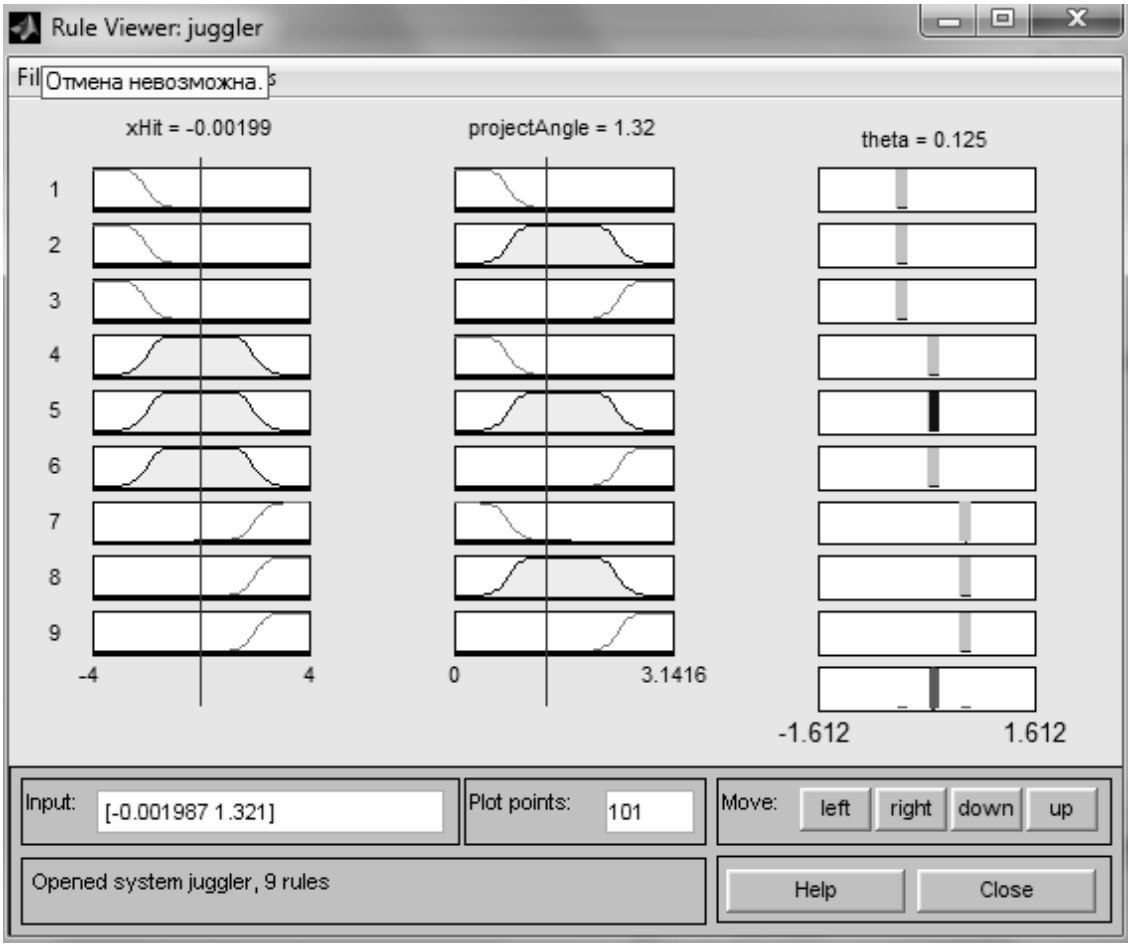

Figure 9.8. *Rule Viewer*

9.2.3. Water mixer control system

Entering the shower command in the *Command Window* of the MATLAB package results in the appearance in the *Simulink* window of a block diagram of a fuzzy control system for a cold and hot water mixer.

We are faced with the operation of a hot and cold water mixer almost every day. If the water pressure is constant, then there is no special need to operate this simple device because it is enough to set the mixing knob to the desired position and control the water temperature by hand. But, if the water pressure is constantly changing, then you shoul use an automatic temperature

controller. The Demo example shows a closed-loop temperature control system using fuzzy logic. *Simulink* system diagrams are shown in Figure 9.9.

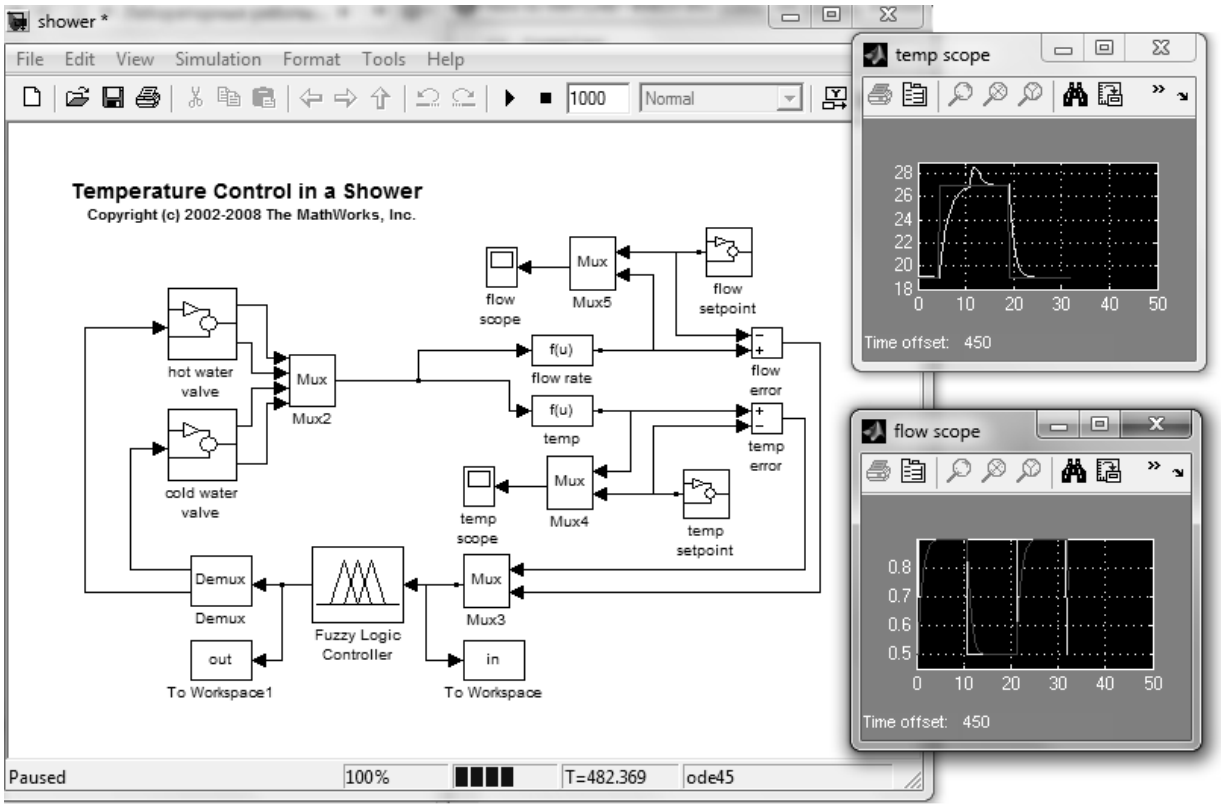

Figure 9.9. *Simulink* Diagrams of a Water Mixer Control System

The system provides for water temperature control, both with manual control and under conditions of changing the pressure of hot and cold water after a given manual setting. Water flow and temperature diagrams are displayed on virtual oscilloscopes.

9.2.4. Inverted pendulum control system

Entering the *slcp1* command in the *Command Window* of the MATLAB package results in a block diagram of the fuzzy inverted pendulum control system appearing in the *Simulink* window. (Figure 9.10).

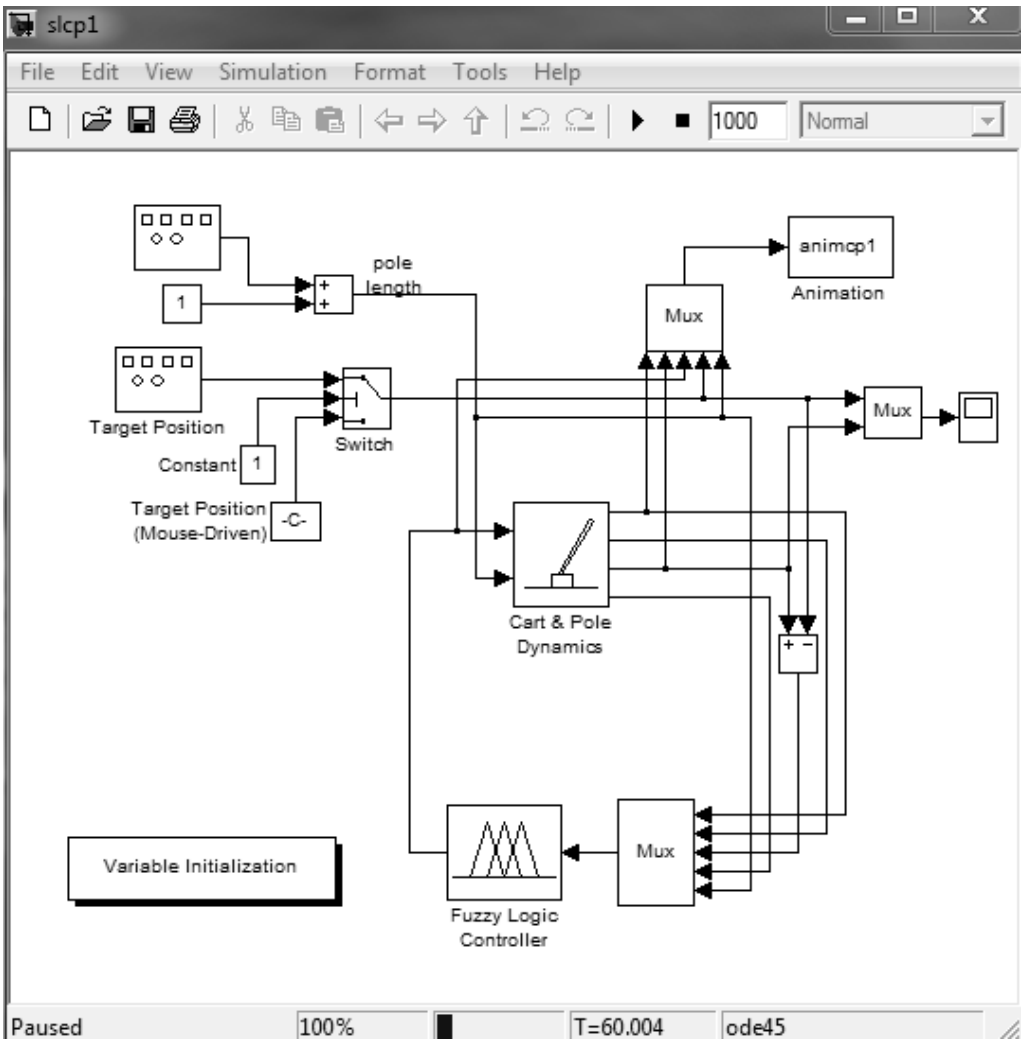

Figure 9.10. Block diagram of a fuzzy control system "inverted pendulum"

Starting the simulation leads to the appearance of an animation picture (Figure 9.11) illustrating the movement in space of the dynamic system "inverted pendulum".

9.2.5. Control system for two inverted pendulums

Entering the *slcpp1* command in the *Command Window* of the MATLAB package results in the appearance of a block diagram of a fuzzy control system for two inverted pendulums in the *Simulink* window (Figure 9.12).

Starting the simulation leads to the appearance of an animation picture (Figure 9.13), illustrating the movement in space of the dynamic system "inverted pendulums".

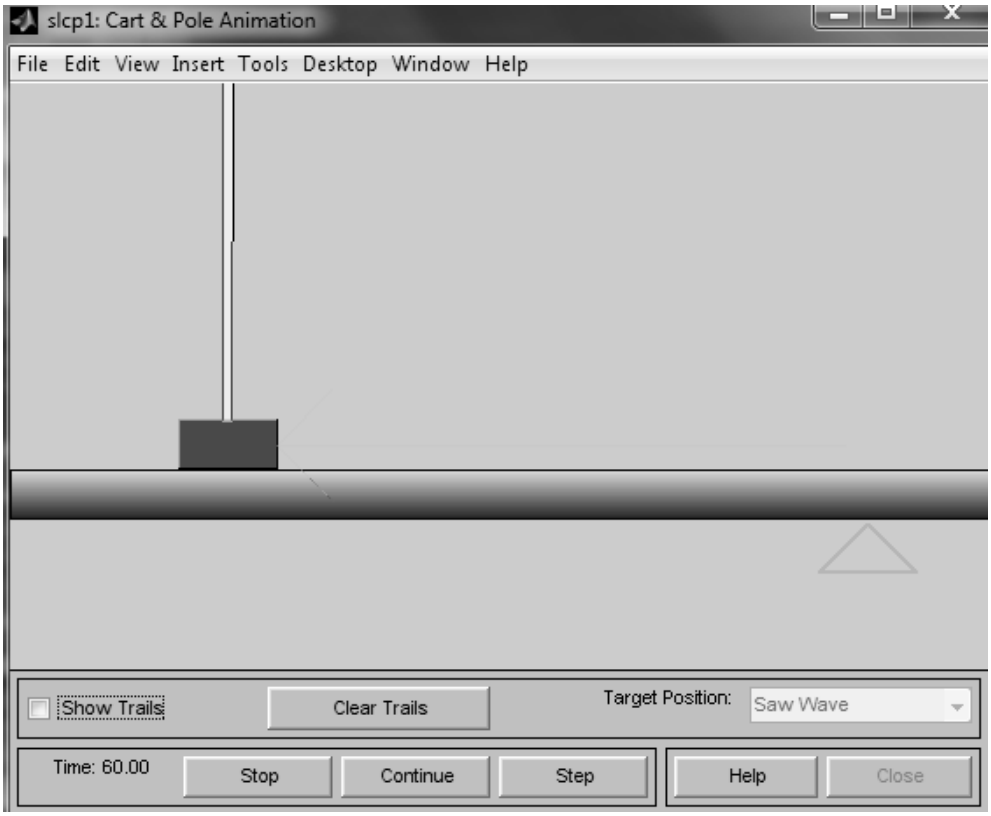

Figure 9.11. Dynamic system "inverted pendulum"

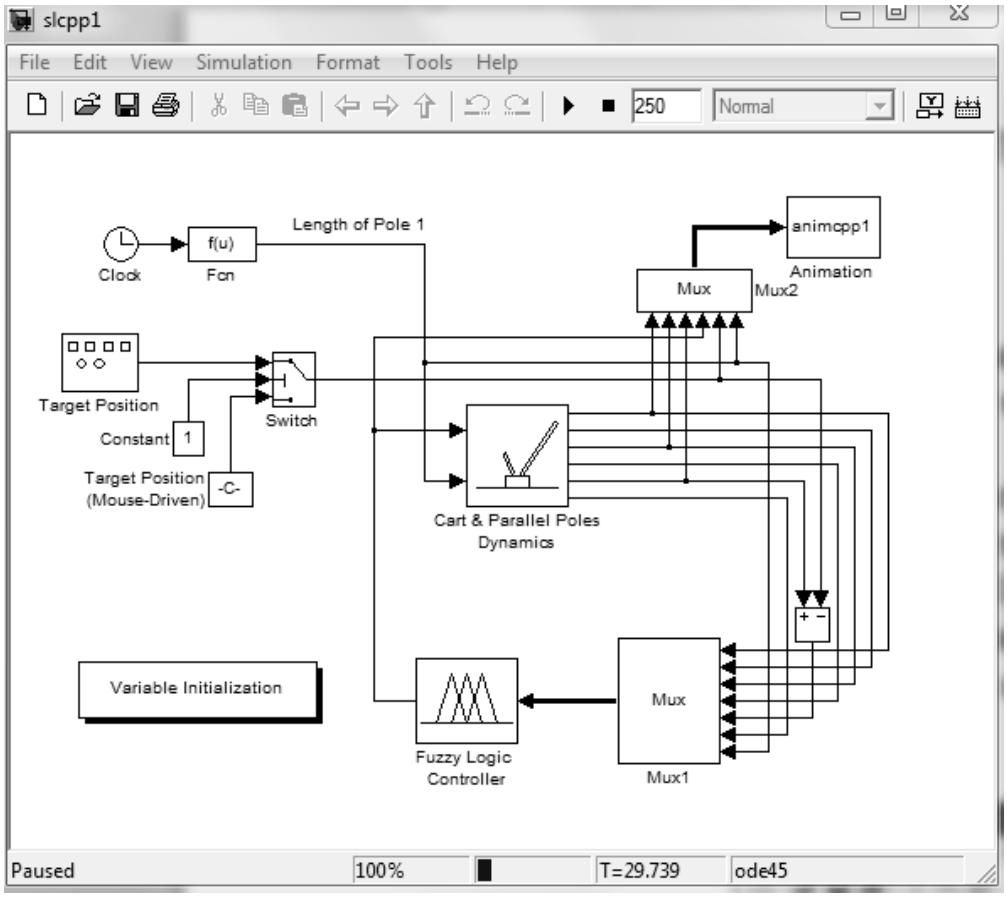

Figure 9.12. Block diagram of a fuzzy control system

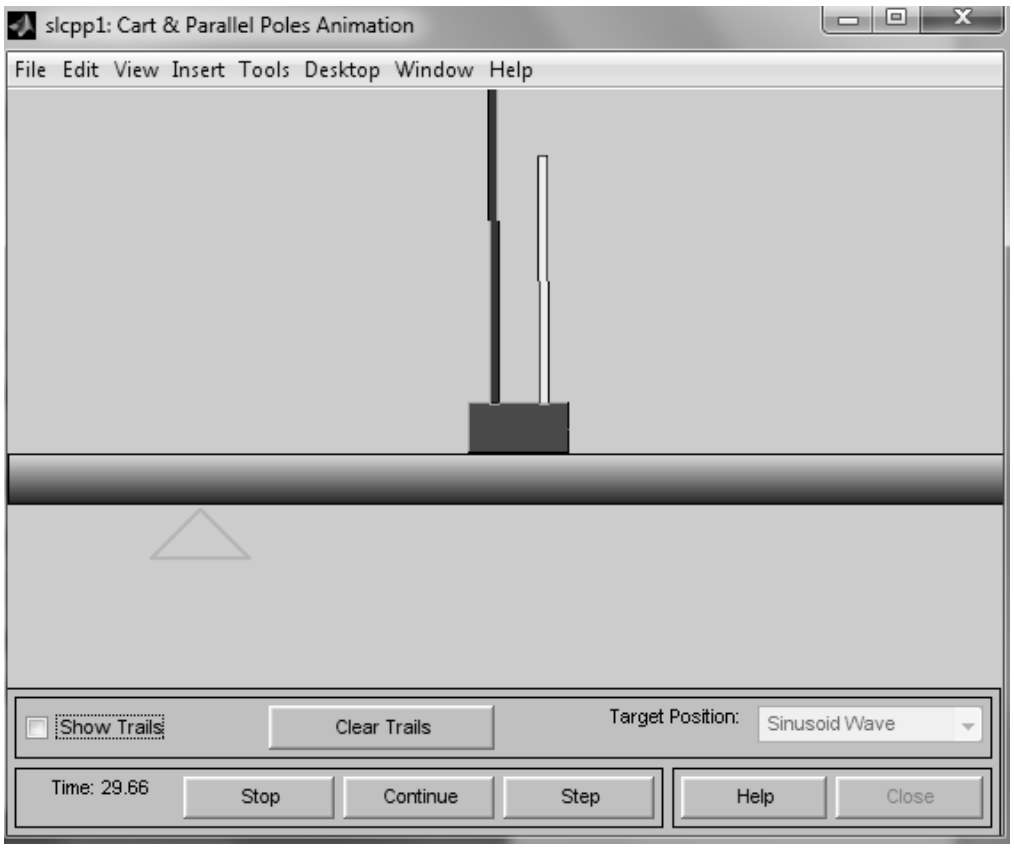

Figure 9.13. "Inverted pendulums" dynamic system

9.3. Individual tasks

1. It is necessary to study methods of constructing fuzzy controllers by means of fuzzy logic and *Simulink* simulation blocks.

2. Simulate all five fuzzy controllers that control various dynamic processes in *Simulink*.

3. Receive and provide in the report a verbal description and a graphical display of the fuzzy inference rules of the considered fuzzy regulators.

4. Describe the conditions and determine the parameters of the system under which there is a "breakdown" of the control of all five fuzzy controllers that control various dynamic processes.

5. Design the lab’s report.

Laboratory work 10

REGULATION USING FUZZY CONTROLLER

The purpose of the laboratory work is to obtain and consolidate knowledge, to form practical skills in working with methods of constructing fuzzy controllers using *Fuzzy Logic* tools and *Simulink* modeling blocks of the MATLAB package.

10.1. Summary of theory

The fuzzy logic package for the MATLAB system is a package of applied programs related to the theory of fuzzy sets and allowing to design the so-called fuzzy expert and control systems. Basic package features:

1. Construction of fuzzy inference systems (expert systems, controllers, dependency approximators).

2. Construction of adaptive fuzzy systems (hybrid neural networks).

3. Interactive dynamic simulation in *Simulink.*

The *Fuzzy Logic Toolbox* software includes the following main programs that have a graphical interface, the *Fuzzy Inference System Editor* (*FIS* editor) together with auxiliary programs: the Membership Function Editor, the *Rule editor*, a *Rule Viewer*; and a *Surface Viewer*. The *FIS* editor is launched from the command line by the *Fuzzy* command.

Fuzzy inference systems created in one way or another using the *Fuzzy Logic Toolbox* can be integrated with *Simulink* tools, which allows you to simulate systems within the latter.

Let's consider an example of building a fuzzy controller using *Fuzzy Logic tools* and simulation blocks of *Simulink*, which monitors the water level in the tank. Figure 10.1 shows a control object in the form of a water tank. Two

pipes fit to the control object: water flows into the tank through one pipe equipped with a tap, and water flows out the tank through the other pipe.

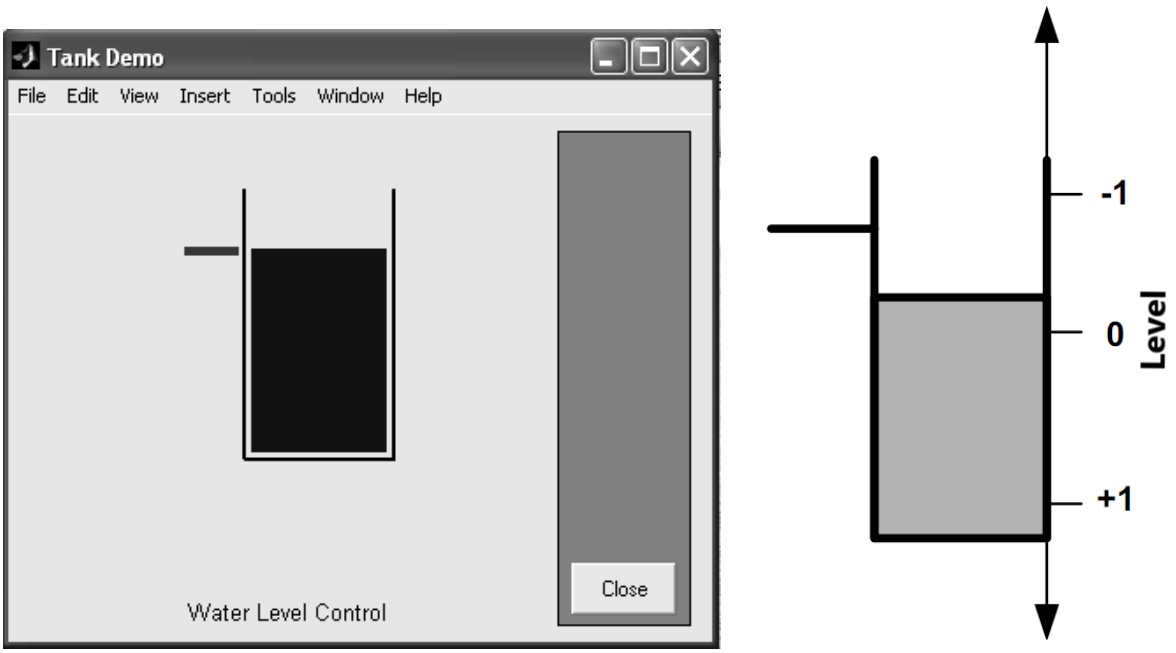

Figure 10.1. Schematic representation of the control object (water tank)

The water supply to the tank can be adjusted by opening the tap more or less. The water flow is uncontrollable and depends on the diameter of the outlet pipe (it is fixed) and on the current water level in the tank. If we understand the water level as the output (adjustable) variable, and the tap as the regulating element, then it can be noted that such a control object is dynamic and non-linear from the point of view of its mathematical description.

Let us define the purpose of control here as setting the water level in the tank at the required (changing) level and try to solve the corresponding control problem by means of fuzzy logic.

Obviously, the regulator, which ensures the achievement of the control goal, must receive information about the discrepancy (difference) between the required water level and actual water level. In this case, this regulator must generate a control signal to the regulating element (valve).

Fuzzy controller construction order:

1. Set membership functions for each of the input and output variables.

2. Develop a rule base for an implemented fuzzy system.

3. Select fuzzy inference algorithm (Mamdani or Sugeno) and analyze the results of the created system.

As a first approximation, the functioning of the regulator can be described by a set of the following rules:

1. *If* (*level is okay*) *then* (*valve is no_change*);

2. *If* (*level is low*) *then* (*valve is open_fast*);

3. *If* (*level is high*) *then* (*valve is close_fast*);

4. *If* (*level is okay*) *and* (*rate is positive*) *then* (*valve is close_slow*);

5. *If* (*level is okay and* (*rate is negative*) *then* (*valve is open_slow*),

which in translation means:

1. IF the level is correct, THEN the tap remains unchanged;

2. IF the level is low, THEN quickly open the tap;

3. IF the level is high, THEN close the valve quickly;

4. IF the level corresponds to the given one AND its growth is positive, THEN the tap must be closed slowly;

5. IF the level corresponds to the given one AND its growth is negative, THEN the tap must be opened slowly.

10.1.1. Formation of the membership function

Launch the *FIS* editor using the *Fuzzy* command. By default, an inference algorithm of the Mamdani type is proposed. Since the system must have two inputs, you need to add a second input to the system through the *Edit / Add input* menu item (a second yellow block named *input2* appears in the editor window).

Next, make a single click on the *input1* block, change its name to "*level*", complete the input of the new name by pressing the <*Enter*> key. Set the name "*rate*" to the *input2* block in the same way and set the name "*valve*" to the output block (top right) *output1*. Assign a name to the entire system, for example "*LevelControl*", by doing this via the *File / Save to Workspace* menu item. The view of the editor window after these actions is shown in Figure 10.2.

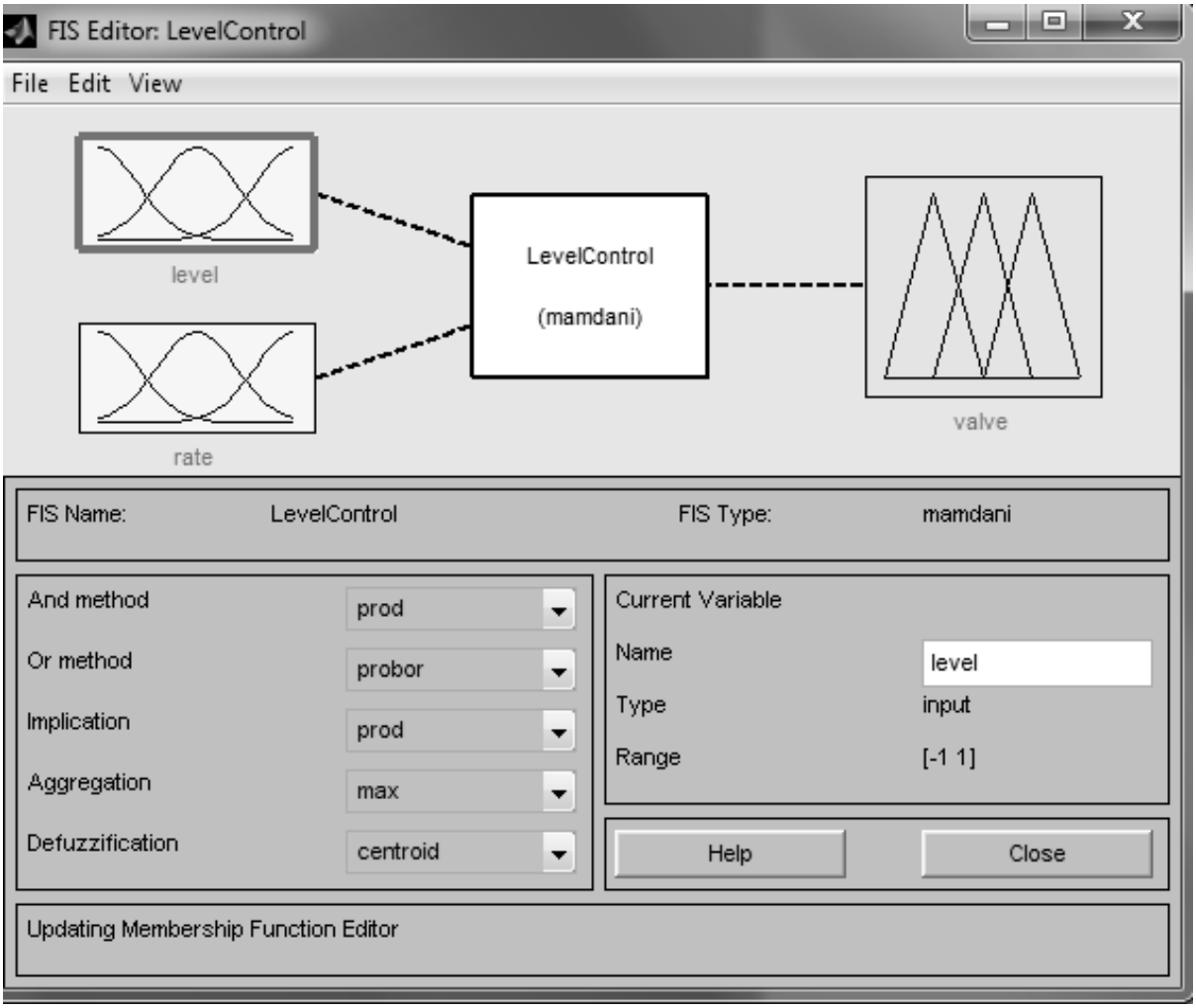

Figure 10.2. View of the FIS-editor window after defining the system structure

In order to generate membership functions for variables, select the *View / Edit membership functions* menu item and set the system parameters in accordance with table 10.1. The following functions are used here. The Gaussian function *Gaussmf* has the form

$$y = Gaussmf(x, [c, \sigma]) = \exp(-(x - c)^2 / (2\sigma)^2).$$

The triangular shape function *Trimf* is defined as follows:

$$f(x,a,b,c) = \begin{cases} 0, & x < a, \\ \dfrac{x-a}{b-a}, & a \le x \le b, \\ \dfrac{c-x}{c-b}, & b \le x \le c, \\ 0, & x > c. \end{cases}$$

Table 10.1 – Parameters of membership functions for variables *level*, *rate*, *valve*

Input variable names	Membership function name	Membership function type
level	*High*	*Gaussmf* [0,3; –1]
	Okay	*Gaussmf* [0,3; 0]
	Low	*Gaussmf* [0,3; 1]
rate	*Negative*	*Gaussmf* [0,03; –0,1]
	None	*Gaussmf* [0,03; 0]
	Positive	*Gaussmf* [0,03; 0,1]
valve	*Close_fast*	*Trimf* [–1; –0,9; –0,8]
	Close_low	*Trimf* [–0,6; –0,5; –0,4]
	No_change	*Trimf* [–0,1; 0; 0,1]
	Open_low	*Trimf* [0,2; 0,3; 0,4]
	Open_fast	*Trimf* [0,8; 0,9; 1]

Generate inference rules using the *Rule Editor*. To do this, you should enter the appropriate rules from subsection 10.1, as shown in Figure 10.3.

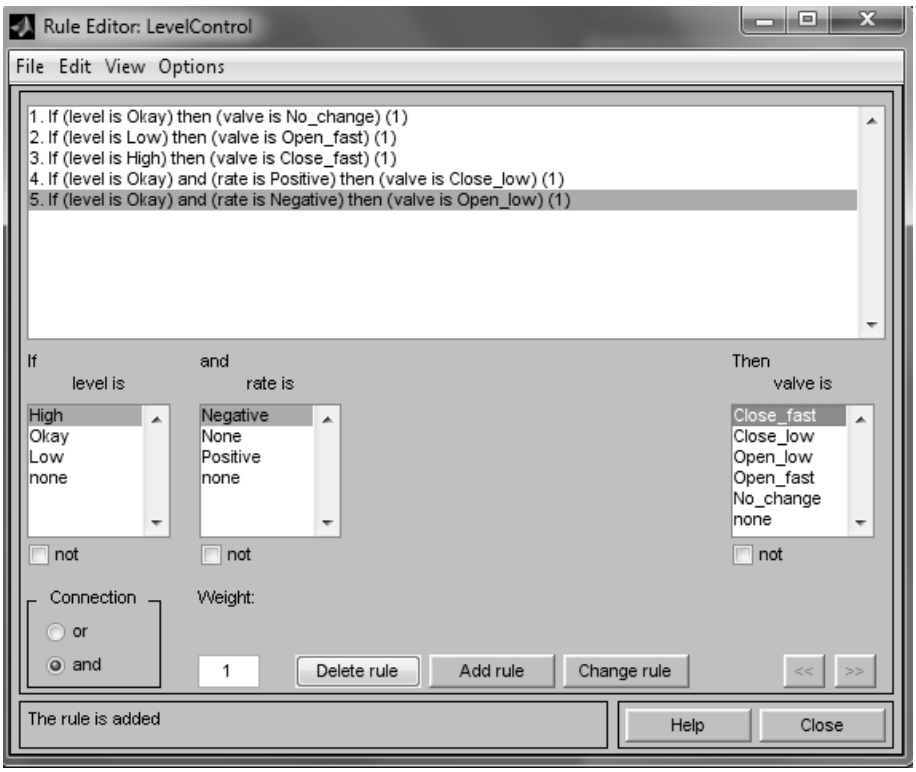

Figure 10.3. Rule editor window

Open the rules viewer window through the *View / View rules* menu item and set the values of the *level* and *rate* variables. The result for the *valve* output should be shown the same as in Figure 10.4.

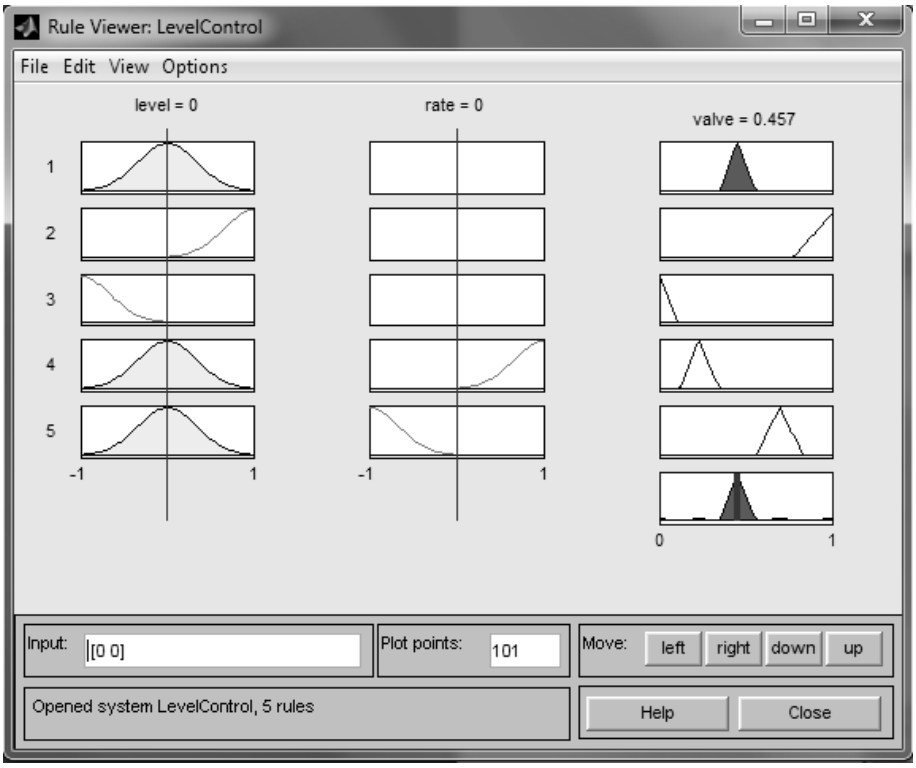

Figure 10.4. Rules viewer window for a task when *level* = 0 and *rate* = 0

Open the window for viewing the graphical dependence of the output variable on the input variables through the *View / View surface* menu item.

You can export and import the results if necessary. When using the menu items *File / Save to disk*, a text (*ASCII*) file with the *fis* extension is created on the disk. It can be viewed, edited if necessary outside the MATLAB system, and also reused in subsequent sessions with the system.

10.1.2. Creating custom membership functions

If none of the built-in membership functions are suitable for the problem being solved, then you can define and use your own function. Such a function must be created as an M-file, return a value in the range from 0 to 1, and have no more than 16 arguments. The function declaration (for example, *custmf*) is executed in the following sequence:

1. Create file named *cutmf.m.*

2... Select the *Edit / Add custom MF* menu item in the membership function editor menu.

3. In the *M-File function* field of the *Add customized membership function* dialog box that appears, enter the name of the created M-file (*custmf*).

4. In the *Parameter List* field of the window enter the required numerical parameters.

5. In the *MF name* field enter some unique name of the function being specified, for example, *polymf*.

6. Completion of the function definition is confirmed by pressing the button. <*ok*> (Figure 10.5).

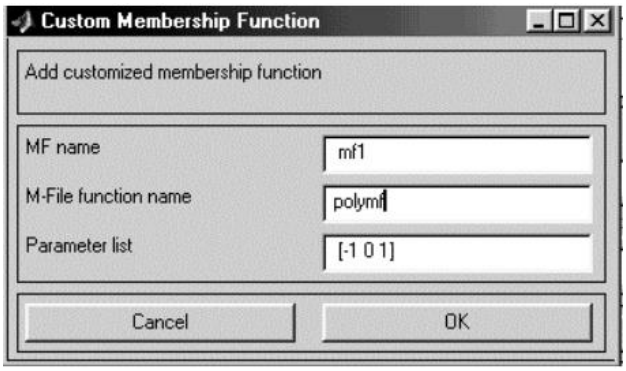

Figure 10.5. User membership function setting window

10.1.3. Model of the water level control system in the tank

A model of a water level control system in a tank with a fuzzy regulator is shown in Figure 10.6.

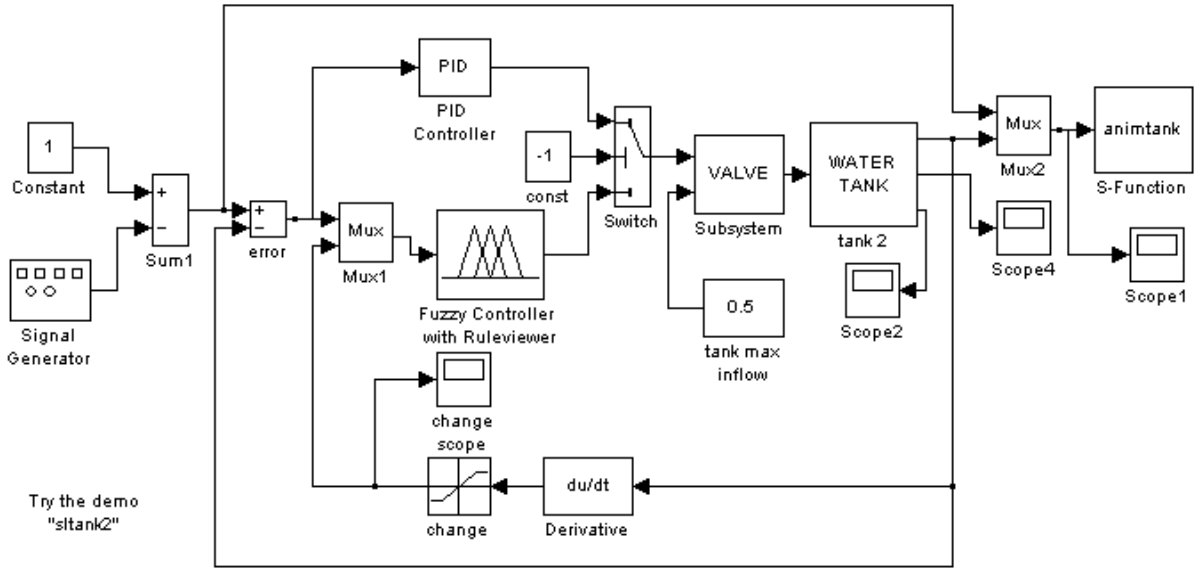

Figure 10.6. Block diagram of a model of a water level control system in a tank with a fuzzy controller

In this case, the adjustable parameters of the main blocks of the water level control system in a tank with a fuzzy controller (Figure 10.6) have the values shown in Figure 10.7.

The animation is started by the *Start* command of the *Simulation* menu. After that, an animation window will appear (the *Tank Demo* window in

Figure 10.1). The current water level is shown in black. The required water level is marked with a black line. The animation starts automatically when the window is opened.

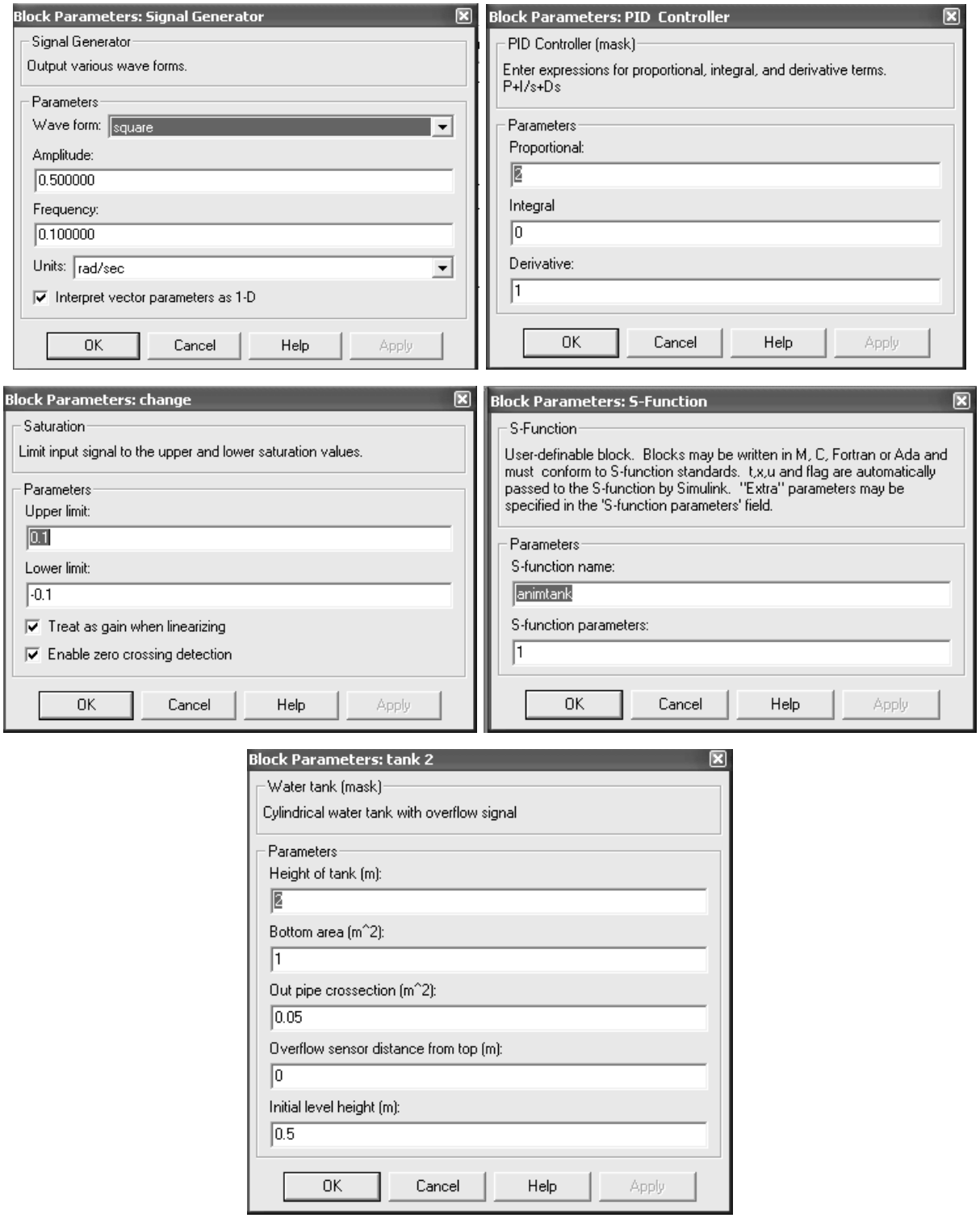

Figure 10.7. Configurable parameters of the main blocks of the water level control system in the tank

The fuzzy controller is implemented by a fuzzy inference system with two inputs: the difference between the required and current water levels and the rate of change of this difference. Timing diagrams of the required and current water levels are shown in the *Scope1* window (Figure 10.8) with light and dark lines, respectively. The *change scope* window (Figure 10.9) shows a timing diagram of the rate of change of the difference, the required water level and the current water level.

These windows are opened by clicking the mouse button on the *Scope1* and *change scope* icons in the *Simulink* model of the "Water tank with a fuzzy controller" system. The user can similarly display the time charts for water flow (*Scope4*) and tank overflow alarm (*Scope2*).

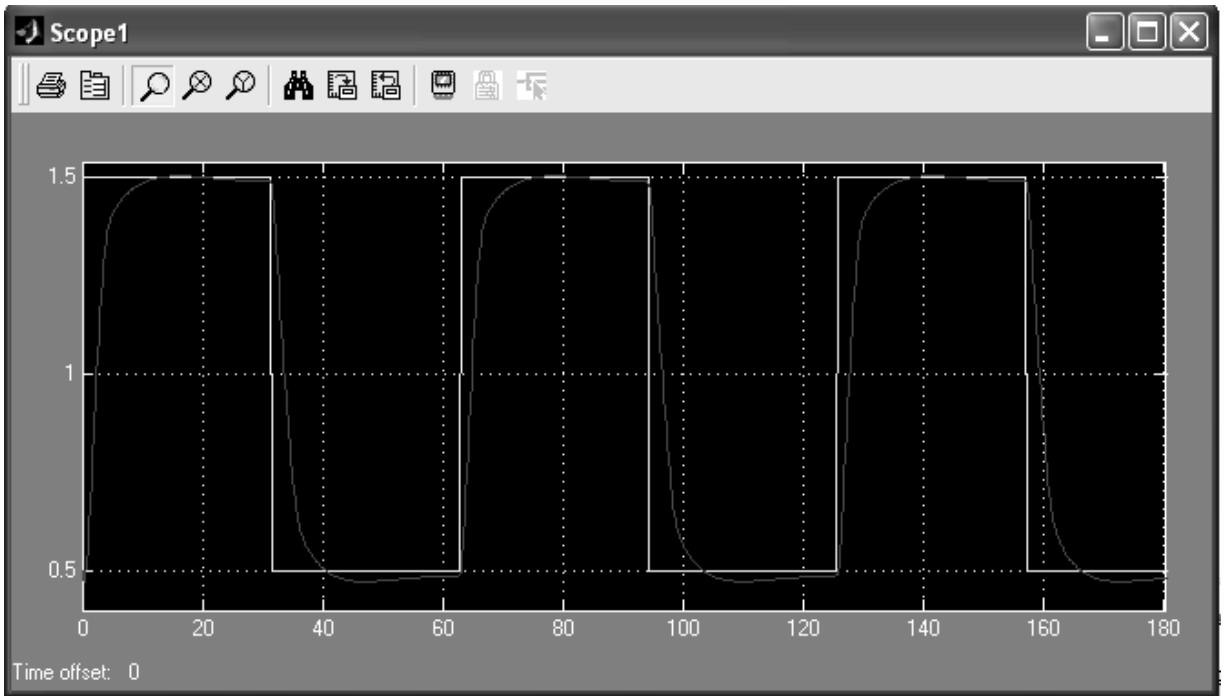

Figure 10.8. Diagram of required and current water levels

As can be seen from the figure, the transient process in the system has an aperiodic form and ends quickly enough, i.e. the quality of regulation should be recognized as good.

The fuzzy controller uses five rules to calculate the control action (Figure 10.10).

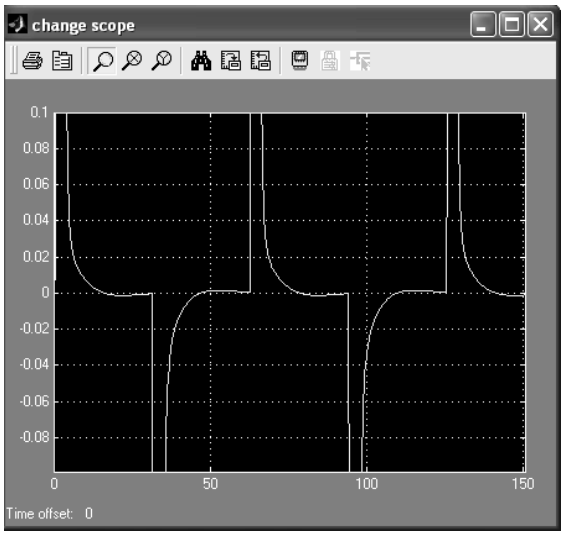

Figure 10.9. Timing diagram of the change rate of the difference between the required and current water levels

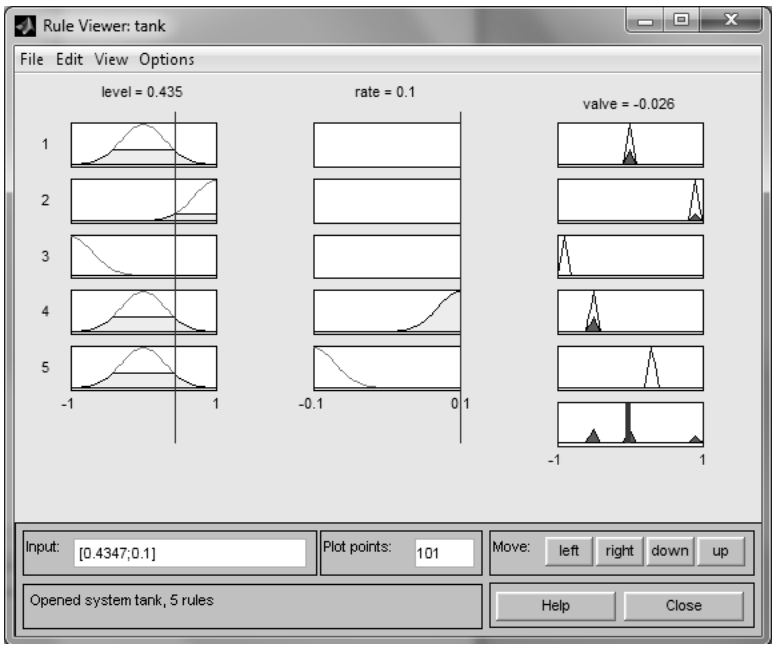

Figure 10.10. Fuzzy rules for the calculation of the control action

For the linguistic assessment of the input variable *level* (the difference between the required and the current water levels), three terms with Gaussian membership functions are used. To estimate the input variable *rate* (the rate of change of the difference between the required and the current water levels), two terms with Gaussian membership functions are used. For the output variable

valve (valve position change) five terms with triangular membership functions are used.

To view the fuzzy controller in *Simulink* format (Figure 10.11), right-click on the *Fuzzy Controller* block (see Figure 10.6) is needed. In the menu that appears, select the *Look under mask* command. Then, in the new *Link: sltankrule / Fuzzy Logic Controller with Ruleviewer* (Figure 10.12) graphics window, right-click on the *Fuzzy Logic Controller* block is needed. In the menu that appears, select the *Look under mask* command. Then, in the new *Link graphic window: sltankrule / Fuzzy Logic Controller with Ruleviewer / Fuzzy Logic Controller*, right-click on the *FIS Wizard* block and again select the *Look under mask* command from the menu that appears.

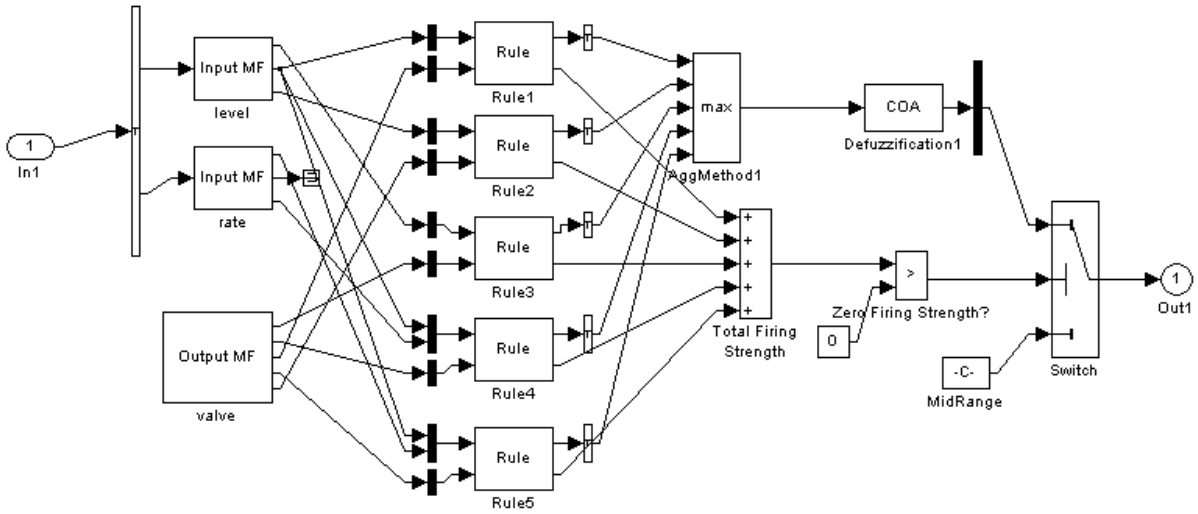

Figure 10.11. Fuzzy controller of water level in a tank in *Simulink* format

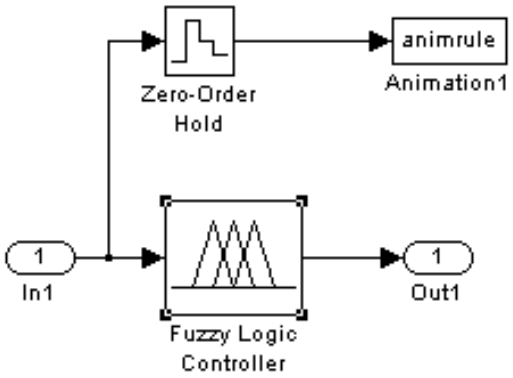

Figure 10.12. *Fuzzy Logic Controller with Ruleviewer* block structure

You can try to change the way the system functions. For example, you can make certain changes to the system defined by the *tank.fis* file (by changing the rules, membership functions, rendering method, etc.).

This can be done with the help of the considered programs with a graphical interface (such as the *FIS editor*, etc.). Of course, all this is best done by stopping the modeling process.

You can go to the block diagram of the same control system, but with a *Fuzzy Logic Controller with Ruleviewer* by executing (in command line mode) the *sltankrule* command.

10.2. Individual tasks

1. It is necessary to study methods of constructing fuzzy controllers using fuzzy logic toolkit and *Simulink* simulation blocks.

2. Simulate a fuzzy controller that controls the water level in a tank in *Simulink.*

3. Get and provide a verbal description and a graphical display of fuzzy inference rules of the constructed regulator in the report.

4. Describe the principle of the circuit operation and explain the results obtained.

5. Make changes to the fuzzy inference rules and explain how the changes will affect the operation of the entire device.

The following parameters are common to all options (Table 10.2).

Individual options for assignments for membership functions of the variable *level* are given in table. 10.3.

It uses the following notation:

$$polymf\,(x,A)=\frac{\exp(a_0+\ldots+a_n x^n)}{1+\exp(a_0+\ldots+a_n x^n)};\ \ A=[a_0,a_1,\ldots,a_n].$$

Table 10.2 – Parameters of membership functions for variables *valve* and *ratre*

Input variable names	Membership function name	Membership function type
valve	*Close_fast*	*Trimf* [–1; –0,9; –0,8]
	Close_low	*Trimf* [–0,6; –0,5; –0,4]
	No_change	*Trimf* [–0,1; 0; 0,1]
	Open_low	*Trimf* [0,2; 0,3; 0,4]
	Open_fast	*Trimf* [0,8; 0,9; 1]
rate	*Negative*	*Gaussmf* [0,03; –0,1]
	None	*Gaussmf* [0,03; 0]
	Positive	*Gaussmf* [0,03; 0,1]

Table 10.3 – Parameters of the membership function for a variable *level*

№	*High* *polymf* (x,A)	*Okay* *polymf* (x,A)	*Low* *polymf* (x,A)
1	2	3	4
1	[–3,535; –2,925; 4,293]	[1,604; –1,723; -9,023]	[–2,908; 2,732; 3,256]
2	[–2,591; –2,978; 3,049]	[2,284; 0,697; –8,499]	[–3,724; 3,836; 4,227]
3	[–3,252; –3,353; 3,829]	[2,104; 0,967; –9,367]	[–3,238; 1,977; 3,823]
4	[–4,149; –2,777; 5,561]	[1,638; 0,4058; –8,782]	[–2,559; 2,687; 2,4026]
5	[–3,553; –3,939; 5,266]	[1,443; –0,218; –6,373]	[–3,323; 3,4241; 2,833]
6	[–2,916; –2,883; 2,781]	[1,736; –0,234; –6,048]	[–2,696; 3,001; 2,9796]
7	[–2,241; –2,435; 2,177]	[2,094; 0,2477; –9,948]	[–1,733; 2,522; 2,3055]
8	[–3,9789; –2,825; 4,47]	[1,941; –0,189; –8,253]	[–2,441; 2,131; 2,3544]
9	[–2,503; –2,979; 2,271]	[1,968; 1,24; –10,8512]	[–2,441; 2,131; 2,3543]
10	[–2,526; –2,787; 2,974]	[2,444; 0,9028; –15,06]	[–3,326; 3,185; 4,0171]
11	[–3,691; –3,016; 3,816]	[1,038; 0,8638; –8,122]	[–2,742; 2,483; 3,4183]
12	[–2,952; –3,167; 3,316]	[1,706; –1,023; –6,489]	[–2,764; 2,392; 2,5585]
13	[–3,124; –3,482; 3,972]	[0,941; –0,3953; –5,63]	[–3,134; 2,372; 3,4613]
14	[–3,662; –3,621; 5,183]	[1,869; –0,291; –8,634]	[–2,95; 3,132; 3,24231]
15	[–3,113; –2,659; 2,983]	[2,038; –2,138; –11,24]	[–3,407; 2,763; 3,8331]

6. Design the lab’s report.

Laboratory work 11

SYMBOLIC CALCULATIONS IN THE MATLAB PACKAGE

The purpose of the laboratory work is to obtain and consolidate knowledge, to form practical skills in working with the MATLAB package in symbolic calculations.

11.1. Summary of theory

11.1.1. Symbolic variables and functions

MATLAB includes *ToolBox Symbolic Math*, designed for symbolic calculations. Conversion of expressions, determination of analytical solutions to problems of linear algebra, differential and integral calculus, obtaining numerical results with any accuracy is not a complete list of the possibilities provided by the *ToolBox*.

Symbolic variables and functions are objects of the *sym object* class, as opposed to numeric variables, which are contained in *double arrays*. The symbolic object is created using the *syms* function. The command

```
» syms x a b
```

creates three symbolic variables *x, a, b*. The construction of symbolic functions from variables of the *sym object* class is done using the usual arithmetic operations and notation for built-in mathematical functions, for example:

```
» f=(sin(x)+a)^2*(cos(x)+b)^2/sqrt(abs(a+b))
f=
     (sin(x)+a)^2*(cos(x)+b)^2/abs(a+b)^(1/2)
```

Writing a formula in one line is not always convenient. So the *pretty* function allows you to get a more natural form of the fraction:

```
» pretty(f)
                2                2
  (b + cos(x))   (a + sin(x))
  ---------------------------
                  1/2
          |a + b|
```

The defined function *f* is also symbolic, as you can easily verify with the *whos* command.

Existing symbolic variables and functions allow the creation of new symbolic expressions:

```
» syms y
» g=(exp(-y)+1)/exp(y)
g =
    (exp(-y)+1)/exp(y)
» h=f*g
h =
    (sin(x)+a)^2*(cos(x)+b)^2/abs(a+b)^(1/2)*(exp(-y)
+1)/exp(y)

» pretty(h)
                2 /   1         \                2
  (b + cos(x))   | ------ + 1 | (a + sin(x))
                 \ exp(y)      /
  ----------------------------------------
                              1/2
                exp(y) |a + b|
```

You can create a symbolic function without first declaring variables using the *sym* function. Its input argument is a string with an expression enclosed in apostrophes:

```
» z=sym('c^2/(d+1)')
z =
    c^2/(d+1)
» pretty(z)
              2
             c
            -----
            d + 1
```

Comment 1

The *sym* function can be used to declare symbolic variables. The *syms a b c* command is equivalent to the command sequence *a* = *sym*('*a*'), *b* = *sym*('*b*'), *c* = *sym*('*c*'):

```
» syms a b c
» whos a b c
Name   Size    Bytes Class
 a     1x1     126 sym object
 b     1x1     126 sym object
 c     1x1     126 sym object
Grand total is 6 elements using 378 bytes
» a1=sym('a1'), b1=sym('b1'), c1=sym('c1')
a1 =
    a1
b1 =
    b1
c1 =
    c1
```

```
» whos a1 b1 c1
 Name   Size     Bytes Class
 a1     1x1      128 sym object
 b1     1x1      128 sym object
 c1     1x1      128 sym object
Grand total is 9 elements using 384 bytes
```

When working in the field of complex numbers, it should be noted that the defined variables are generally complex. Complex symbolic variables are set with the *syms* command with the *unreal* option. The *real* option means that variables are treated as real

Consider an example, where the result of symbolic computation depends on which symbolic variables are used: real or complex ones. Let's declare two real variables *a* and *b* and form a complex number, assuming that *a* is the real part and *b* is the imaginary part. Then find its conjugate number using *conj*:

```
» syms a b real
» p=conj(a+i*b)
p = a-i*b
```

Let's perform similar actions, having previously declared *a* and *b* as complex symbolic variables

```
» syms a b unreal
» q=conj(a+i*b)
q = conj(a+i*b)
```

Thus, in the general case $p \neq q$.

Symbolic variables can be elements of vectors and matrices. Matrix row items are specified with spaces or commas as they are entered. Column items are specified with semicolons, just like when entering regular matrices. The result is

symbolic matrices and vectors to which matrix and element-wise operations and built-in functions are applicable. Consider an example of input and multiplication of two symbolic matrices:

```
» syms a b c d e f g h
» A=[a b;c d]
A =
    [ a, b]
    [ c, d]
» B=[e f; g h]
B =
    [ e, f]
    [ g, h]
» C=A*B
C =
    [ a*e+b*g, a*f+b*h]
    [ c*e+d*g, c*f+d*h]
```

The *sym* function allows you to convert the values of numeric variables to symbolic ones. Enter numeric matrix *A* and convert it to symbolic matrix *B*:

```
» A=[1.3 -2.1 4.9; 6.9 3.7 8.5]
A =
    1.3000  -2.1000  4.9000
    6.9000  3.7000  8.5000
» B=sym(A)
B =
    [ 13/10, -21/10, 49/10]
    [ 69/10, 37/10,  17/2]
```

Writing a number in the form of a rational fraction is used in the transition from numerical expressions to symbolic ones. The use of rational fractions when performing symbolic calculations allows you to always get an accurate result that does not contain rounding errors. You can verify this with the following simple example. Set the format to *long* to display as many significant digits as possible for the values of numeric variables and find the sum of the numbers 10^{10} and 10^{-10}

```
» format long
» 1.0e10+1.0e-10
ans =
    1.000000000000000e+010
```

Now convert the numbers to symbolic variables and calculate the sum again:

```
» large=sym(1.0e10);
» small=sym(1.0e-10);
» s=large+small
s = 10000000000000000001/10000000000
```

The rational fraction is the exact value of the sum. It is clear that symbolic calculations are time consuming compared to conventional computations.

Calculation of symbolic expressions is done using the *vpa* function, for example

```
» c=sym('sqrt(2)');
» cn=vpa(c)
cn =
    1.414213562373095048801688724209 7
```

Thirty-two significant digits are saved by default. The second input parameter of the *vpa* function allows you to specify calculations with a given number of bits, for example, with 45 bits

```
» cn=vpa(c,45)
cn =
    1.41421356237309504880168872420969807856967187
```

The output of the *vpa* function is a symbolic variable, but it can be used in normal calculations, for example

```
» cn=vpa(c,45)
cn =
    1.41421356237309504880168872420969807856967187
» cn+2
ans =
    3.41421356237309504880168872420969807856967187
» cn*2
ans =
    2.82842712474619009760337744841939615713934375
```

The result of arithmetic operations in these cases is obtained in symbolic variables. To convert symbolic variables to numeric ones, i.e. variables of the *double array* type, the *double* function is used:

```
» cnd=double(cn)
cnd = 1.41421356237310
```

The visualization of symbolic functions of one variable is done using *ezplot*. The simplest way to use *ezplot* is to specify a symbolic function as the only input argument, while the graph of the function on the segment $[-2\pi, 2\pi]$ is displayed in the graphics window. For example:

```
» f=sym('x*sin(x^2)^3');
» ezplot(f)
```

Note (Figure 11.1) that a corresponding title is automatically generated.

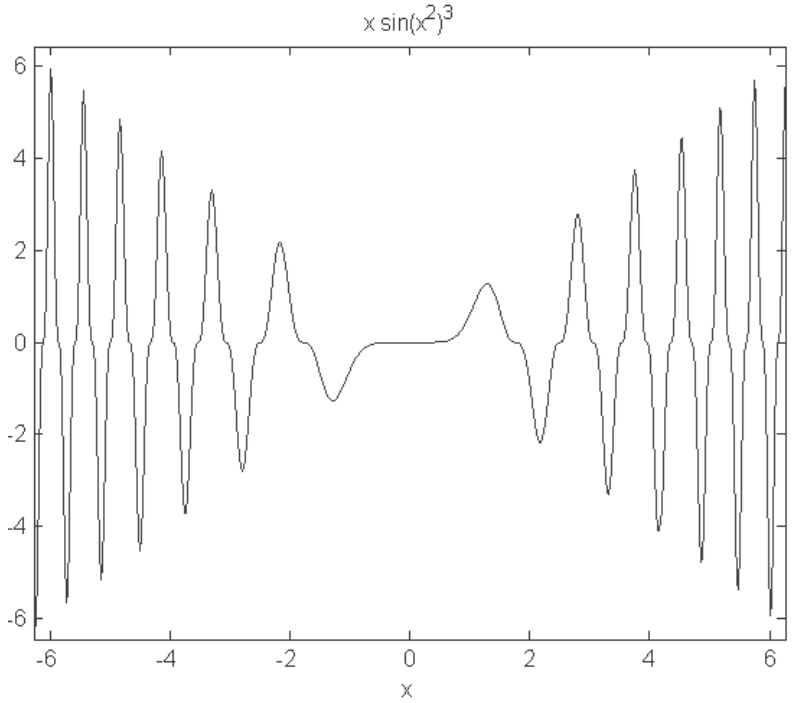

Figure 11.1. Symbolic function graph

The second argument can be a vector with the boundaries of the segment on which you want to plot the graph of the function:

```
» ezplot(f,[-9 7])
```

The *ezplot* function has some differences from its counterpart, the *fplot* function, which is applied to numeric functions. In particular, it is possible to specify a symbolic function depending on two arguments:

```
» z=sym('x^2+a^3');
» ezplot(z,[-2 1 -3 4])
```

The limits of argument change are determined by their names. The first two numbers correspond to the first alphabetically, and the last numbers correspond to the second alphabetically. In the example under consideration, *a* changes from –2 to 1, and *x* changes from –3 to 4. A line is displayed in the graphics window that satisfies the expression `x^2+a^3=0`.

ToolBox Symbolic Math provides the user with a whole set of tools for visualizing symbolic functions [1, 2].

11.1.2. Simplifying and Converting of Expressions

Complex algebraic and trigonometric expressions can often be reduced to equivalent ones by simplification. *ToolBox Symbolic Math* has a number of utility functions for various symbolic transformations. Polynomial operations are implemented in four functions: *collect*, *expand*, *factor*, *horner*.

The coefficients of the powers of the independent variable are calculated using the *collect* function. Enter the polynomial and display it in the command window with *pretty*:

```
» p=sym('(x+a)^4+(x-1)^3-(x-a)^2-a*x+x-3');
» pretty(p)
             3                   4           2
 x + (x - 1)  - a x + (a + x)  - (a - x)  - 3
```

Then apply the *collect* function to the polynomial:

```
» pc=collect(p);
» pretty(pc)
  4               3        2        2
 x  + (4 a + 1) x  + (6 a  - 4) x  +
     3                4    2
```

```
+ (4 a  + a + 4) x + a  - a  - 4
```

By default, *x* is selected as the independent variable in the polynomial. However, we can assume that *a* is an independent variable, and *x* is included in the coefficients of a polynomial depending on a. The second argument of the *collect* function is intended for specifying the variable, at the degrees of which the coefficients should be found:

```
» pca=collect(p,'a');
» pretty(pca)
   4           3        2        2        3
  a  + (4 x) a  + (6 x  - 1) a  + (4 x  + x) a + x +
            3    2    4
+ (x - 1)  - x  + x  - 3
```

The *expand* function represents a polynomial as a sum of monomials:

```
» pe=expand(p);
» pretty(pe)
   4      3         2  2    2         3
  a  + 4 a  x + 6 a  x  - a  + 4 a x  +
          4    3      2
+ a x + x  + x  - 4 x  + 4 x - 4
```

The *expand* argument can be not only a polynomial, but also a symbolic expression containing trigonometric, exponential, and logarithmic functions:

```
» pk=sym('(sin(x)+cos(x))^3+(sin(x)+cos(x))^2+
(sin(x)+cos(x))')
pk =
    (sin(x)+cos(x))^3+(sin(x)+cos(x))^2+(sin(x)+cos(x
))
» pk2=expand(pk);
» pretty(pk2)
```

```
      3           2                2                2
cos(x)  + 3 cos(x)  sin(x) + cos(x)  + 3 cos(x) sin(x) +
                                  3         2
+ 2 cos(x) sin(x) + cos(x) + sin(x)  + sin(x) + sin(x)
```

The arguments of the *expand*, *collect* functions can be not only separately polynomials or trigonometric, or exponential, or logarithmic functions, but also their combinations, for example:

```
» p=sym('(x+a)^4+(x-1)^3+(sin(x)+cos(x))^4');
» pretty(p)
         3                      4           4
  (x - 1)  + (cos(x) + sin(x))  + (a + x)
» p1=collect(p)
p1 =
    x^4+(1+4*a)*x^3+(-3+6*a^2)*x^2+(3+4*a^3)*x+a^4-
-1+(sin(x)+cos(x))^4
» p2=expand(p1)



p2 =
    x^4+x^3+4*x^3*a-3*x^2+6*x^2*a^2+3*x+4*x*a^3+a^4-
1+sin(x)^4+4*sin(x)^3*cos(x)+6*sin(x)^2*cos(x)^2+4*si
n(x)*cos(x)^3+cos(x)^4
» pretty(p2)
   4      3        2  2         3    4    3      2
  a  + 4 a  x + 6 a  x  + 4 a x  + x  + x  - 3 x  + 3 x +
       4           3                 2       2
 + cos(x)  + 4 cos(x)  sin(x) + 6 cos(x)  sin(x)  + 4*
               3         4
 * cos(x) sin(x)  + sin(x)  - 1
```

Symbolic polynomials are factorized by the *factor* function if the resulting factors have rational coefficients:

```
» p=sym('7*x^5-56*x^4+105*x^3+140*x^2-532*x+336');
» p1=factor(p)
p1 =
     7*(x-2)*(x-3)*(x-4)*(x+2)*(x-1)
```

The *factor* function can also represent numbers as the product of prime numbers.

```
» syms a
» a=sym('2738470');
» a1=factor(a)
a1 = (2)*(5)*(7)*(19)*(29)*(71)
```

Note that the appeal

```
» a2=factor(2738470)
a2 = 2   5   7  19  29  71
```

outputs a similar result to the command window, but the variable *a1* is symbolic, and the variable *a2* is real. It is easy to verify using *whos*:

```
» whos a1 a2
    Name  Size    Bytes Class
     a1   1x1     176 sym object
     a2   1x6     48 double array
Grand total is 33 elements using 224 bytes
```

MATLAB is an object-oriented environment, *double array* numeric variables form a class with their *factor* function, and the *factor* function for symbolic variables is implemented in another file function. MATLAB determines the corresponding class by the type of the argument, and then it determines the required function.

The *horner* function allows you to represent a polynomial according to the Horner scheme (*Horner polynomial representation*):

```
» horner(p1)
ans =
     336 +(-532+(140+(105+(-56+7*x)*x)*x)*x)*x
```

Simplification of general expressions is performed using the *simple*, *simplify* functions, which are based on different approaches. The *simplify* function implements a powerful algorithm for simplifying expressions containing both trigonometric, exponential and logarithmic functions, as well as special functions.

In addition, the *simplify* function is capable of transforming expressions containing symbolic exponentiation, summation, and integration. The algorithm in *simple* tries to get an expression that is represented by fewer characters than the original by consistently applying all of the *ToolBox* simplification functions. When simplifying complex expressions, it is advisable to use both functions, since either of them can produce better results than the other. Consider the following example

```
»  v=sym('(((2+sqrt(3))/(sqrt(2)+sqrt(2+sqrt(3))))+(2-
sqrt(3))/(sqrt(2)-sqrt(2-sqrt(3))))^2')
v =
((3^(1/2)  -  2)/(2^(1/2)  -  (2  -  3^(1/2))^(1/2))  -
-(3^(1/2) + 2)/((3^(1/2) + 2)^(1/2) + 2^(1/2)))^2
```

```
» pretty(v)
  /        1/2                    1/2           \2
  |       3    - 2               3    + 2       |
  | ------------------- - ------------------- |
  |  1/2         1/2 1/2     1/2     1/2    1/2 |
  \ 2    - (2 - 3   )       (3    + 2)    + 2    /

» v2=simplify(v)
» pretty(v2)

  /        1/2                    1/2           \2
  |       3    - 2               3    + 2       |
  | ------------------- - ------------------- |
  |  1/2         1/2 1/2     1/2     1/2    1/2 |
  \ 2    - (2 - 3   )       (3    + 2)    + 2    /

» v1=simple(v)

v1 = 2
```

Thus, the best result is obtained with the *simple* function in this example.

The *subs* function allows you to substitute one expression into another. In general, a function is called with three input arguments: the name of the symbolic function, the variable to be replaced, and the expression to be substituted into the expression. The *subs* function, in particular, makes it easier to enter cumbersome symbolic expressions. For example:

```
»  f=sym('((N^2-x^2)/(N+x)^2)+(sin(2*x)/A)*(sqrt(B*C))
+(A/B)^2+(cos(x)/C)^2')
» pretty(f)
   2    2          2    2                   1/2
  N  - x     cos(x)     A    sin(2 x) (B C)
  -------- + ------- + -- + -----------------
         2       2        2            A
  (N + x)       C        B

» f=subs(f,'A','sin(x)');
```

```
» f=subs(f,'B','tan(x)');
» f=subs(f,'C','cot(x)');
» pretty(f)
   2    2          2         2                              1/2
  N  - x     cos(x)    sin(x)    sin(2 x) (cot(x) tan(x))
  -------- + ------- + ------- + -------------------------
         2         2         2             sin(x)
  (N + x)    cot(x)    tan(x)
```

Let's simplify the resulting expression using the *simple* function:

```
» f2=simple(f)
f2 =
    (2*sin(x)*N+sin(2*x)*N+sin(2*x)*x)/(N+x)/sin(x)
» pretty(f2)

      2 sin(x) N + sin(2 x) N + sin(2 x) x
      ------------------------------------
           (N + x) sin(x)
```

Substitution of its numeric value instead of a variable leads to the computation of a symbolic function from the value of the argument, for example

```
»  f=sym('((N^2-x^2)/(N+x)^2)+(sin(2*x)/A)*(sqrt(B*C))
+(A/B)^2+(cos(x)/C)^2')
» pretty(f)
   2    2          2    2                   1/2
  N  - x     cos(x)    A    sin(2 x) (B C)
  -------- + ------- + -- + -----------------
         2         2    2           A
  (N + x)         C    B

» q=subs(f,'x',0)
q =
    1+A^2/B^2+1/C^2
```

```
» f3=sym('sin(x)+exp(x)+tan(x)');
» q1=subs(f3,'x',0)
q1 =
     1
```

11.1.3. Taylor series expansion and definition of symbolic expressions for sums

The *taylor* function allows you to expand mathematical functions in a Taylor series for example:

```
» f=sym('sin(x)');
» tf=taylor(f);
» pretty(tf)
   5     3
  x     x
  --- - -- + x
  120   6
```

By default, six terms of the expansion series are displayed in the vicinity of the zero point. The number of terms in the expansion can be specified in the second optional parameter of the *taylor* function. The third parameter indicates which of the variables should be used for the expansion in the case when the symbolic function is defined in several variables. The point in the vicinity of which the decomposition is carried out is indicated in the fourth input argument of the *taylor* function, for example

```
» syms x;
» f=sym('sin(x)');
» tf=taylor(f, 5, x, pi/4);
» pretty(tf)
   1/2 / pi     \3    1/2 / pi     \2    1/2 / pi     \4
  2    | -- - x |    2    | -- - x |    2    | -- - x |
       \ 4      /         \ 4      /         \ 4      /
```

```
2           \ 4      /
  ---------------- - ---------------- + ---------------- +
         12                   4                   48

            1/2 / pi      \
    1/2   2     | -- - x |
  2             \ 4      /
+ ---- - ---------------
  2             2
```

The *symsum* function allows you to find symbolic expressions for sums, including infinite ones. In general, referring to *symsum* involves specifying four arguments: a term in symbolic form, depending on the index, the index itself, and the upper and lower limits of the sum. If the terms include a factorial, then the *sym* function should be applied to the expression for the factorial. Find the value of the infinite sum, which is the series expansion of the *sin(x)* function

$$s = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

```
» syms k x
» s=symsum((-1)^(k)*x^(2*k+1)/sym('(2*k+1)!'),
k,0,inf)
s =
    sin(x)
```

11.1.4. Determination of limits, differentiation and integration

The *limit* function finds the limit of a function at some point, including plus or minus infinity. The first input argument to *limit* is a symbolic expression, the second input argument is a variable, and the third input argument is the point at which the limit is defined. Let, for example, it is required to calculate

$$\lim_{x \to \infty} (1 + \frac{1}{x})^{ax}$$

```
» syms a x
» limit((1+1/x)^(x*a),x,Inf)
```

```
ans = exp(a)
```

The limit function allows you to find one-sided limits. To find the limit on the right, specify the fourth optional argument *'right'*, and to find the limit on the left specify the fourth optional argument *'left'*. Find a solution to the following two problems

$$\lim_{x\to 0+}(10+x)^{1/x};\quad \lim_{x\to 0-}(10+x)^{1/x}.$$

```
» syms x
» limit((10+x)^(1/x),x,0, 'left')
ans = 0
» limit((10+x)^(1/x),x,0, 'right')
ans = inf
```

Note that the usual limit at zero does not exist:

```
» limit((10+x)^(1/x),x,0)
ans =
     NaN
```

Defining the derivative in terms of the limit allows you to use limit to differentiate functions. For example, let's find the first derivative of a arctg(x) function using the equality

$$\frac{d}{dx}\text{arctg}(x)=\lim_{h\to 0}\frac{\text{arctg}(x+h)-\text{arctg}(x)}{h}$$

```
» syms h x
» L=limit((atan(x+h)-atan(x))/h, h, 0);
» pretty(L)
```

```
    1
------
     2
1 + x
```

Computing derivatives of any order is easier with the diff function. The symbolic notation of the function is indicated in the first input argument, the variable by which the differentiation is made is indicated in the second input argument, and the order of the derivative is indicated in the third input argument.

Let's apply *diff* to calculate the first and second derivatives of the function $arctg(x)$:

```
» P=diff('atan(x)',x,1);
» pretty(P)
    1
------
 2
x  + 1

» P=diff('atan(x)',x,2);
» pretty(P)
      2 x
- ---------
    2     2
   (x  + 1)
```

Symbolic integration is much more difficult than differentiation. *ToolBox Symbolic Math* allows you to work with both indefinite and definite integrals. Indefinite integrals of symbolic functions are calculated using the *int* function. The symbolic function and the variable over which the integration is performed are indicated as input arguments, for example, let it be necessary to calculate the indefinite integral $f = \int \exp(2x)dx$, then we get:

```
» syms x
» f=sym('exp(2*x)');
» I=int(f,x)
» pretty(I)
    1/2 exp(2 x)
```

Of course, the *int* function does not allow obtaining an indefinite integral of an arbitrary function.

To find a definite integral in symbolic form, you should set the lower and upper limits of integration, respectively, in the third and fourth arguments of *int*

```
» syms x a b
» f=sym('exp(2*x)');
» I=int(f,x,a,b);
» pretty(I)
    1/2 exp(2 b) - 1/2 exp(2 a)
```

Double integrals are calculated by applying the *int* function twice. For example, let it be necessary to calculate the integral $\int_c^d \int_a^b y\sin(x)dxdy$, then to define it, you need to set symbolic variables *a*, *b*, *c*, *d*, *x*, *y*, integrand function *f* from *x* and *y* and integrate first over one variable, and then over another

```
» syms a b c d x y
» f=sym('y*sin(x)');
» Ix=int(f,x,a,b)
Ix =
    -y*cos(b)+y*cos(a)
» Iy=int(Ix,y,c,d)
Iy =
```

```
    1/2*(-cos(b)+cos(a))*(d^2-c^2)
» pretty(Iy)
      2    2
    (c  - d ) (cos(a) - cos(b))
  - --------------------------
                 2
```

Any multiple integrals are calculated in a similar way in symbolic form.

11.2. Individual tasks

1. Specify symbolic variables to define two symbolic matrice *A* and *B*, size 3×3. Determine the product of matrices *A* and *B*.

2. Specify a numeric matrix *D*, size 3×3, containing numbers *N*, *N*+1, *N*–1, 0.9*N*, 2.4, 3.5 etc. as elements where *N* is your group journal number. Get Symbolic Matrix from Numeric Matrix.

3. Determine the sum of the numbers $10^{10}+10^{-10}+10^{N}+10^{-N}$, using symbolic calculations where *N* is your group journal number.

4. Use symbolic calculations to determine the square root of a number *N*.*N* has forty-two significant digits.

5. Plot Symbolic Functions:

$$f = N^2 \sin(Nx),\ -2\pi < x < 2\pi;$$

$$f_1 = (x-N)^2 + (y+N)^3,\ -10 \le x \le 10;\ 10 \le y \le 10,$$

where *N* is your group journal number.

6. Here is a polynomial

$$y = (x+N)^5 - (x+2N)^3 + (x+N)^2 + x - N, \qquad (11.1)$$

where N is your group journal number. Use the *pretty* command to display the polynomial in a command window, then convert it to powers x with appropriate coefficients.

7. Determine the coefficients of the polynomial (11.1) at the variable N.

8. Represent the polynomial (11.1) as a sum of monomials.

9. Factor the polynomial (11.1).

10. Apply the *horner* function to the polynomial (11.1).

11. Present the number 1000 N+2 as a product of prime numbers.

12. Simplify the expression

$$y = \sin(a)\sin(b-c)\cos(b+c-a) + \sin(b)\sin(c-a)\cos(c+a-b) + \\ + \sin(c)\sin(a-b)\cos(a+b-c).$$

13. Get seven terms of the Taylor series in the vicinity of the zero point for the function $y = N(\sin(x) + \cos(x))$.

14. Determine the sum of the members of the series $s = \sum_{k=0}^{\infty} N(-1)^k \frac{x^{2k}}{(2k)!}$.

15. Determine the limit $\lim_{x \to 0} N\frac{\sin(x)}{x}$, where N is your group journal number.

16. Define the limits $\lim_{x \to 0+} \operatorname{arctg}(\frac{1}{x})$ and $\lim_{x \to 0-} \operatorname{arctg}(\frac{1}{x})$.

17. Determine the first three derivatives of a function $y = N(\sin(x))^2 + N^2 x^3 + e^{Nx}$, where N is your group journal number.

18. Calculate the integral $\int_0^b \int_0^a (\frac{x^2}{2p} + \frac{y^2}{2q}) dxdy$.

19. Design the lab's report.

Laboratory work 12

APPLICATION OF GENETIC ALGORITHMS IN DETERMINING THE EXTREMA OF FUNCTIONS

The purpose of the laboratory work is to obtain and consolidate knowledge, to form practical skills for determining the extrema of functions using genetic algorithms in the MATLAB package.

12.1. Summary of theory

Genetic algorithms (GA) refer to heuristic search algorithms that are used to solve optimization and modeling problems by random selection, combination and variation of the desired parameters using mechanisms that resemble biological evolution. GA is a type of evolutionary computation that solves optimization problems using natural evolution methods such as inheritance, mutation, selection, and crossing over. A distinctive feature of GA is the use of the "crossing" operator, which performs the operation of recombination of candidate solutions, the role of which is similar to the role of crossing in wildlife.

Typical applications of GA are as follows: search for extremums of functions; optimization of queries in databases; graph problems (traveling salesman problem, coloring, finding matchings); setting up and training an artificial neural network; layout tasks; scheduling; game strategies; approximation theory, etc.

The main idea of GA is the organization of the "struggle for existence" and "natural selection" among the trial (approximate) solutions to the problem. Since GAs use biological analogies, the terminology used is reminiscent of biological.

As you know, the evolutionary theory asserts that life on our planet arose at first only in its simplest forms - in the form of unicellular organisms. These forms gradually became more sophisticated, adapting to the environment and giving rise to new species. Only many millions of years later did the first animals and humans appear. We can say that each biological species over time improves its qualities so as to most effectively cope with the most important tasks of survival, self-defense, reproduction, etc. This is how a protective coloration arose in many fish and insects, a tortoise's shell, poison in a scorpion, and many other useful devices.

With the help of evolution, nature is constantly optimizing all living things, sometimes finding the most extraordinary solutions. At first glance, it is unclear why this progress occurs, but there is a scientific explanation for it. This explanation can be given based on only two biological mechanisms - natural selection and genetic inheritance.

Natural selection plays a key role in evolutionary theory. Its essence lies in the fact that the fittest individuals survive better and bear more offspring than the less fit ones. Note that natural selection by itself does not yet ensure the development of a biological species. Indeed, if we assume that all descendants are born approximately the same, then different generations will differ only in number, but not in fitness. Therefore, it is very important to study how inheritance occurs, i.e. how the properties of the child depend on the properties of the parents.

The basic law of inheritance is intuitive to everyone. It consists in the fact that descendants are similar to their parents. In particular, the descendants of more fit parents are likely to be some of the fittest in their generation. To understand what this similarity is based on, we need to delve a little deeper into the structure of the animal's cell - into the world of genes and chromosomes.

In the cells of any animal there is a set of chromosomes that carry information about this animal. The main part of the chromosome is a DNA strand (a molecule of deoxyribonucleic acid), which consists of four types of

special compounds - nucleotides, going in a specific sequence. Nucleotides are designated by the letters A, T, C and G, and it is the order in which they appear that encodes all the genetic properties of a given organism. More precisely, DNA determines what chemical reactions will take place in a given cell, how it will develop and what functions it will perform.

A gene is a segment of the DNA chain responsible for a specific property of an individual, for example, for eye color, hair type, skin color, etc. The entire set of human genetic traits is encoded by approximately 60 thousand genes, the total length of which is more than 90 million nucleotides.

There are two types of cells: sex and somatic. Each human somatic cell contains 46 chromosomes. These 46 chromosomes are actually 23 pairs, and in each pair one of the chromosomes is received from the father, and the second of the chromosomes is received from the mother. Paired chromosomes are responsible for the same traits - for example, the paternal chromosome may contain the gene for black eyes, and the paired maternal chromosome may contain the gene for blue eyes. There are certain laws governing the participation of certain genes in the development of an individual. In particular, in our example, the descendantwill be black-eyed, since the blue-eye gene is "weak" (recessive) and is suppressed by any other color gene.

There are only 23 chromosomes in the sex cells, and they are unpaired. During fertilization, the fusion of the male and female sex cells occurs and the embryo cell is formed, containing just 46 chromosomes. What properties will the descendant receive from the father, and what properties will the descendant receive from the mother?

It depends on what kind of sex cells were involved in fertilization. The fact is that the process of production of sex cells (the so-called meiosis) in the body is subject to accidents, thanks to which the offspring still differ in many ways from their parents. In meiosis, in particular, the following happens: the paired chromosomes of a somatic cell come close together, then their DNA strands break in several random places and the chromosomes exchange

their parts (Figure 12.1).

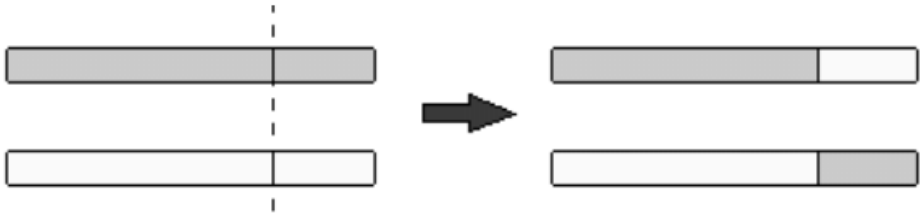

Figure 12.1. Conventional crossover scheme

This process provides the emergence of new variants of chromosomes and is called "crossover". Each of the newly appeared chromosomes will then be inside one of the sex cells, and its genetic information can be realized in the descendants of this individual.

The second important factor affecting heredity is mutations, which are expressed in changes in some parts of DNA. Mutations are also random and can be caused by various external factors such as radiation exposure. If the mutation has occurred in the sex cell, then the changed gene can be passed on to the descendant and manifest itself in the form of a hereditary disease or in other new properties of the descendant. It is believed that it is mutations that are the cause of the emergence of new biological species, and the crossover already determines the variability within the species (for example, genetic differences between people).

As noted above, evolution is a process of continuous optimization of biological species. Natural selection guarantees that the fittest individuals will produce a sufficiently large offspring, and thanks to genetic inheritance, we can be sure that some of these offspring will not only retain the high fitness of their parents, but will also have some new properties. If these new properties turn out to be useful, then with a high probability they will be passed on to the next generation. Thus, there is an accumulation of useful qualities and a gradual increase in the adaptability of the biological species as a whole. Knowing how

the problem of optimizing species in nature is solved, we now apply a similar method to solve various real-life problems.

Let's introduce some notation and give some classic examples. As a rule, in the optimization problem, we can control several parameters (we denote their values by x_1, x_2, ..., x_n, and our goal is to maximize (or minimize) some function $f(x_1, x_2, ..., x_n)$, depending on these parameters. The function f is called the objective function. For example, if you want to maximize the objective function "company income", then the controlled parameters are the number of employees in the company, production volume, advertising costs, prices for final products, etc. It is important to note that these parameters are interrelated: in particular, with a decrease in the number of employees, the volume of production is likely to fall.

Several solution methods have been developed for this kind of problem. If the objective function is sufficiently smooth and has only one local maximum (unimodal), then the optimal solution can be obtained by the gradient descent method. The idea behind this method is that the optimal solution is obtained by iteration. A random starting point is taken, and then, in the cycle, this point is shifted by a small step, and the step is made in the direction in which the objective function grows the fastest. The disadvantage of the gradient algorithm is that the requirements for the function are too high - in practice, unimodality is extremely rare, and for a non-unimodal function, the gradient method often leads to a non-optimal response.

Similar problems arise with other mathematical methods. In many important problems, parameters can only take certain values, and the objective function is not defined at all other points. Of course, in this case there can be no question of its smoothness and fundamentally different approaches are required.

Imagine an artificial world inhabited by many creatures (individuals), and each creature is a solution to our problem. We will consider an individual to be the more adapted, the better the corresponding solution (the greater the value of the objective function it gives). Then the task of maximizing the objective

function is reduced to finding the most fit creature. Of course, we cannot put all creatures in our virtual world at once, since there are a lot of them. Instead, we will consider many successive generations.

Now, if we manage to put into action natural selection and genetic inheritance, then the resulting world will obey the laws of evolution. Note that in accordance with our definition of fitness, the goal of this artificial evolution will be precisely to create the best solutions. Evidently, evolution is an endless process in the course of which the fitness of individuals gradually increases. By forcibly stopping this process long enough after its start and choosing the most fit individual in the current generation, we can get an answer close to the optimal one, but we may not get such an answer, since this is a random search. This, in short, is the idea of a genetic algorithm. Let us now turn to precise definitions and describe the operation of the genetic algorithm in more detail.

In order to talk about genetic inheritance, we need to supply our creatures with chromosomes. In the genetic algorithm, a chromosome is a certain numerical vector corresponding to the selected parameter, and the set of chromosomes of a given individual determines the solution to the problem. What kind of vectors should be considered in a specific task is up to the user himself. Each of the positions of the chromosome vector is called a gene.

Let us now define the concepts corresponding to mutation and crossover in the genetic algorithm.

A mutation is a transformation of a chromosome that randomly changes one or more of its positions (genes). The most common type of mutation is a random change in only one of the genes on a chromosome.

Crossover (in the GA literature, the name crossing over and crossing is also used) is an operation in which one or more new chromosomes are generated from two chromosomes. In the simplest case, the crossover in the genetic algorithm is implemented in the same way as in biology (см. Figure 12.1). In this case, the chromosomes are cut at a random point and exchange parts with each other. For example, if the chromosomes (1, 2, 3, 4, 5) and (0, 0, 0, 0, 0) are

cut between the third and fourth genes and exchange their parts, then you get descendants (1, 2, 3, 0, 0) and (0, 0, 0, 4, 5).

The genetic algorithm diagram is shown in Figure 12.2.

Initially, an initial population of individuals (individuals) is generated, i.e. some set of solutions to the problem. This is usually done randomly. Then we have to simulate reproduction within this population. For this, several pairs of individuals are randomly selected, a cross is made between the chromosomes in each pair, and the resulting new chromosomes are placed in the population of a new generation.

The genetic algorithm retains the basic principle of natural selection. It lies in the fact that the more adapted the individual (the greater the value of the objective function corresponding to him), the more likely he will participate in crossing. Now mutations are simulated: in several randomly selected individuals of the new generation, some genes are changed. Then the old population is partially or completely destroyed, and we move on to consider the next generation. The population of the next generation in most implementations of GA contains the same number of individuals as the initial one, but due to selection, the fitness in it is on average higher. Now the described processes of selection, crossing and mutation are repeated for this population, etc.

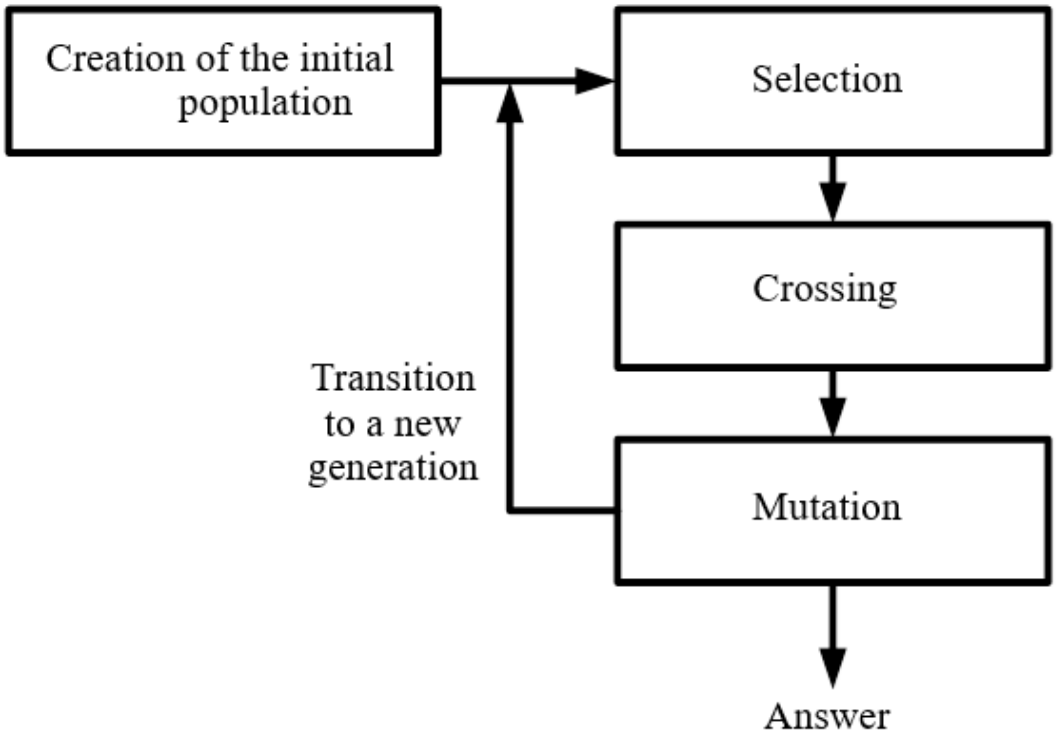

Figure 12.2. Block diagram of a genetic algorithm

We will observe the emergence of completely new solutions to our problem in each next generation. Among them there will be both bad and good, but thanks to selection, the number of good solutions will increase. Note that there are no absolute guarantees in nature, and even the most fit tiger can die from a rifle shot without leaving offspring. By simulating evolution on a computer, we can avoid such undesirable events and always keep the best individual of the current generation alive. This technique is called the "strategy of elitism".

12.2. Implementation of Genetic Algorithms Using MATLAB Console

In MATLAB, the ability to use GA for computations is implemented using the *Genetic Algorithm and Direct Sarch Toolbox* tab, which extends the capabilities of the *Optimization Toolbox* with genetic algorithms. Such algorithms are most often used in the case when the desired objective function is

discontinuous, essentially nonlinear, stochastic and has no derivatives, or these derivatives are not sufficiently defined. You can now work with genetic algorithms in two toolboxes.

Genetic algorithms belong to the *Genetic Algorithm* section and are called from the command line using *gatool* or *ga*. Genetic algorithms and their combinations with other optimization methods can be found in the *Direct Search Toolbox* section. To do this, type *psearchtool* on the command line. Let's consider the first option of working with GA. There are 4 main functions for working with the algorithm:

ga is function for finding the minimum of the objective function;

gaoptimget returns the parameters of the used genetic algorithm;

gaoptimset sets the parameters of the genetic algorithm;

gatool opens the *Genetic Algorithm Tool* window.

In order to apply GA to the set goal function, it is necessary, first of all, to write it to the M-file and save it in the current folder.

12.2.1. Ga function

The *ga* function is called on the command line according to the syntax below:

$$[x\ fval] = ga(@fitnessfun, nvars, options),$$

where *fitnessfun* is the name of the M-file containing the supplied objective function, *nvars* is the number of independent variables in the objective function, *options* is the structure containing the parameters of the used GA. If the parameters are not changed, then their values will be taken by default. The calculation results will be saved in following variables: *fval* is final objective function value, *x* is the point at which the optimum value is reached.

There are other options for calling the *ga* function:

```
x = ga(fitnessfun, nvars)
x = ga(fitnessfun, nvars, options)
x = ga(problem)
[x, fval]=ga(...)
[x, fval, reason]=ga(...)
[x, fval, reason, output]=ga(...)
[x, fval, reason, output, population]=ga(...)
[x, fval, reason, output, population, scores]=ga(...)
```

The description *x* = *ga*(*fitnessfun*, *nvars*) is used to solve the optimization problem, *fitnessfun* is the objective function to be minimized, and *nvars* is the length of the decision vector *x* corresponding to the best individual.

x = *ga*(*fitnessfun*, *nvars*, *options*) is used to solve the optimization problem using the parameters (options) of the algorithm.

x = *ga*(*problem*) finds the minimum of a function whose structure is described by three fields:

fitnessfcn is an objective function;

nvars is the number of independent variables of the objective function;

options are the parameters of the GA structure specified by the *gaoptimset* function.

[*x*, *fval*] = *ga*(...) returns *fval*, the value of the objective function by *x*.

[*x*, *fval*, *reason*] = *ga*(...) returns *reason* that is a string containing the parameters for stopping the algorithm.

[*x*, *fval*, *reason*, *output*] = *ga*(...) returns *output* a set of information about each generation and other information about the implementation of the algorithm. The *output* structure consists of the following fields:

Randstate or (*randnstate*) is the initial state of the population, randomly generated (the difference in functions is in different conclusions);

generations is the number of generations to be calculated;

funccount is the number of function calculations;

message are parameters for stopping the algorithm. This message prints out multiple arguments for the completion of the algorithm.

[*x*, *fval*, *reason*, *output*, *population*] = *ga*(...) returns a population matrix whose rows correspond to an individual in the final population.

[*x*, *fval*, *reason*, *output*, *population*, *scores*] = *ga*(...) returns the final population calculations.

Comment 1

The population must be represented as real numbers for all optimization problems. The *ga* function does not work for functions with complex variables. To solve problems involving complex numbers, you need to write the objective function in the form of an admissible real vector, separating the real and complex parts. Such an objective function might look like the following:

```
[x fval, reason] = ga(@rastriginsFcn, 10)
x =
    Columns 1 through 7
    0.9977 0.9598 0.0085 0.0097 −0.0274 −0.0173 0.965
    Columns 8 through 10
    −0.0021 −0.0210 0.0065
fval =
    3.7456
reason =
    generations
```

12.2.2. Gaoptimset function

The *gaoptimset* function is used to configure the genetic algorithm. It allows you to build a GA by combining operators as desired by the user. The syntax for this function is as follows:

```
options = gaoptimset
```

```
gaoptimset
options = gaoptimset('param1', value1, 'param2',
value2,...)
options = gaoptimset(oldopts,'param1',value1,...)
options = gaoptimset(oldopts,newopts)
```

Function Description:

options = *gaoptimset* (no arguments are introduced here) with the help of this construction, the structure of the GA is specified, which differs from the structure of the GA by default.

gaoptimset does not require input or output of arguments. As a result, it will form a list of parameters and their actual values.

options = *gaoptimset*('*param*1',*value*1,'*param*2',*value*2,...) generates a structure with many parameters and their values. The default values can be used for a few non-special parameters.

options = *gaoptimset*(*oldopts*,'*param*1',*value*1,...) creates a copy of *oldopts*, modified with the selected custom options and their values.

options = *gaoptimset*(*oldopts*, *newopts*) combines the parameters of the existing structure *oldopts*, with the parameters of the new structures *newopts*.

Some parameters with non-zero values in *newopts* can be overwritten or can be assigned to old parameters in *oldopts*.

12.3. Implementation of Genetic Algorithms Using the MATLAB Dialog Box

Another more intuitive way to work with GA is to call the window using the gatool function. The dialog box is shown in Figure 12.3.

As you can see from the figure, the window is accompanied by help on all components, and the user's work consists in setting the parameters and pressing the *"start"* button. The result will be the same as if the *gaoptimset* and *ga*

functions were used sequentially. In the *plots* section, you can select variables whose changes will be displayed graphically.

Vectorization of functions is provided in the GA dialog box, due to which the calculations are much faster. The meaning of this method is that vectors are used as function parameters, then for the current population the target function will be called only once, calculating the fitness of all individuals. For example, consider the function $f(x_1, x_2) = x_{21} – 2x_1x_2 + 6x_1 + x_2^2 – 6x_2$.

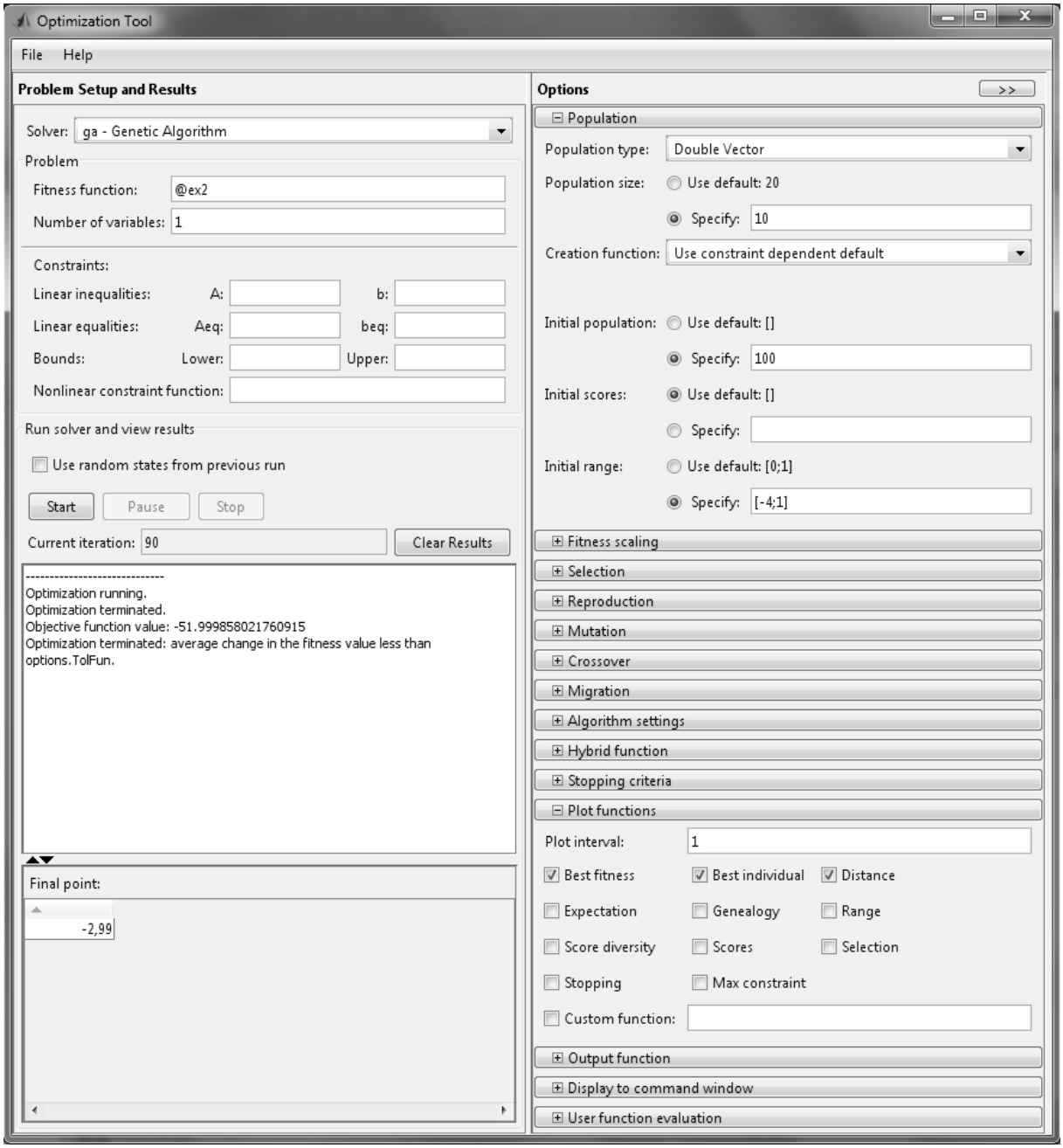

Figure 12.3. Genetic Algorithms Dialog Box

Let's write an M-file for it using the following code:

```
» z =x(:,1)./ 2 – 2*x(:,1).*x(:,2) + 6*x(:,1) +
x(:,2)./ 2–6*x(:,2);
```

Here *x*(:, 1) is a vector, a ./ и .* are operations of elementwise exponentiation and multiplication, respectively.

In the toolbox window, in the *Vectorize option* list, set the *On* value.

You can verify the efficiency of vectorization using the example of the Rastrigin function. Let's enter the following expression on the command line:

```
» tic;
» ga(@rastriginsfcn,20);
» toc;
```

As a result, you can find out the time spent on calculations:

```
Elapsed time is 4.366073 seconds.
```

Let's do the same, but using the vectorization of the Rastrigin function:

```
» options=gaoptimset('Vectorize','on');
» tic;
» ga(@rastriginsfcn,20,options);
» toc;
```

Find out the time spent on vectorization:

```
Elapsed time is 0.581 seconds.
```

As you can see, the vectorization method is much faster.

12.4. Application of genetic algorithms to find the minimum of a function

Let us find the minimum of one variable function of the form:

$$f(x) = 8x - 16 - 12\sqrt[3]{(x+4)^2}\,.$$

Let's write an M-file for this function and save it in the current folder under the name *ex2.m*.

```
function y = ex2(x)
y=8*x-16-12*((x+4)^(2/3));
```

Let's call the toolbox window with *gatool*.

In the *fitness function* field, enter the name of the objective function *@ex*2.

Set the values of the GA parameters: the number of individuals in the population is 10, the number of generations is 100 (in the window of the algorithm stopping criterion), the initial segment is [–4; 1]. In the *plots* section, check the boxes for *best fitness, best individual, distance* and press the *start* button.

As a result of the process completion, the value of the variable *x* corresponding to the minimum of the function will appear in the *final point* window. And in the *status and result* window, you can see the found minimum value of the objective function.

For this task, the results are as follows:

– the minimum of the function is attained at the point *x* = –2.99;

– function value $f(-2.99) = -51.999858021760915$.

Figure 12.4 shows the change in the value of the objective function, the best individual and the distance between individuals in generations. From the data obtained in Figure 12.4, it can be seen that starting from about 80 populations, the algorithm converges to a solution. Individuals become the same (Hamming distance is 0) in the last 18 generations.

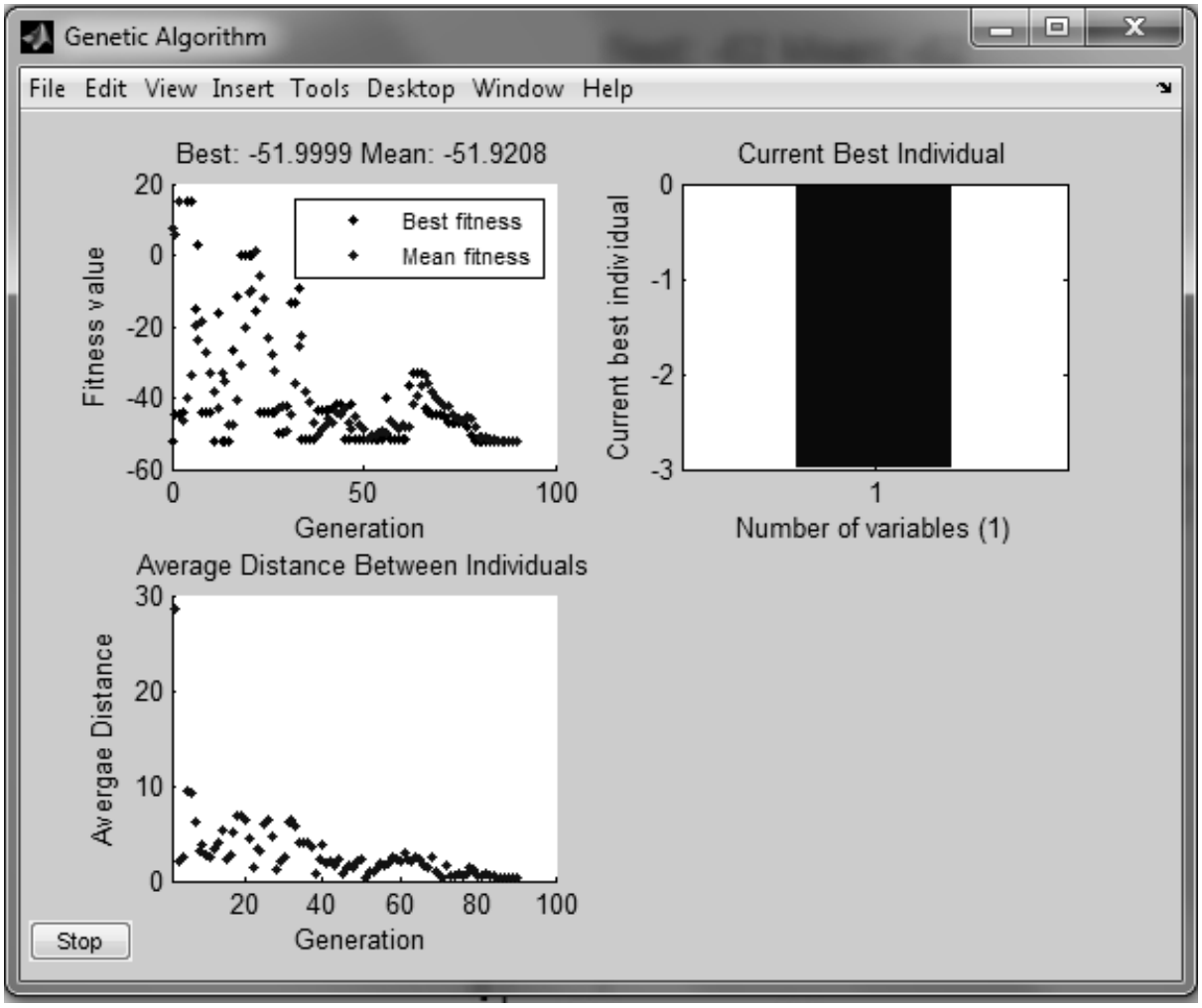

Figure 12.4. Graphical analysis of the solution

GA needs to be run several times, and then the optimal solution should be chosen. This is because the initial population is formed using a random number generator. You can verify the correctness of the solution by plotting the function (Figure 12.5).

The same could be achieved using the *gaoptimset* and *ga* functions. To view the M-file, select the *Generate M-file* command from the *File* menu of the *Genetic Algoritm Tool* window, save the file under a different name and view the code.

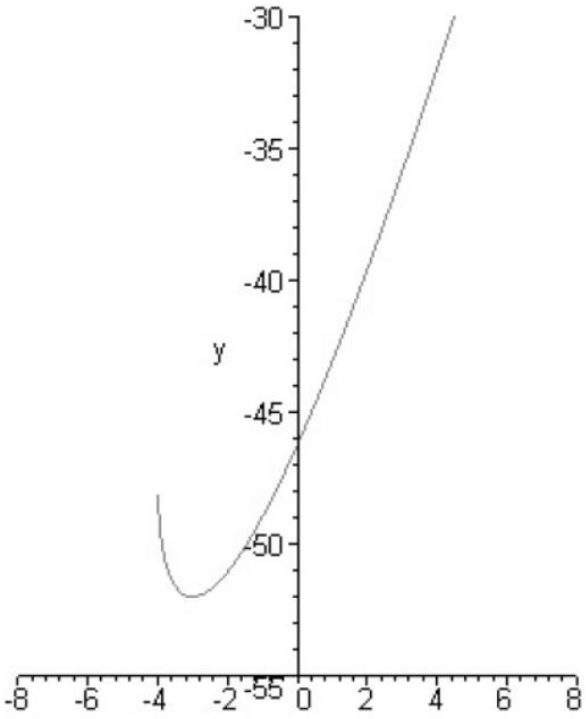

Figure 12.5. Function graph

For this task, we got:

```
function [x,fval,exitflag,output,population,score] =
ex2q(nvars,PopInitRange_Data,PopulationSize_Data,InitialPo
pulation_Data)
% This is an auto generated M-file from Optimization Tool.
% Start with the default options
options = gaoptimset;
% Modify options setting
options = gaoptimset(options,'PopInitRange',
PopInitRange_Data);
options = gaoptimset(options,'PopulationSize',
PopulationSize_Data);
```

```
options = gaoptimset(options,'InitialPopulation',
InitialPopulation_Data);
options = gaoptimset(options,'Display', 'off');
options = gaoptimset(options,'PlotFcns', {  @gaplotbestf
@gaplotbestindiv @gaplotdistance });
[x,fval,exitflag,output,population,score] = ...
ga(@ex2,nvars,[],[],[],[],[],[],[],options);
```

12.5. Application of genetic algorithms to find the maximum of a function

Let us find the maximum of a function of two variables of the form:

$$z(x,y) = \exp(-x^2 - y^2) + \sin(x + y).$$

Only minimization problems can be solved in the GA dialog box. To find the maximum of the function *f(x)*, the function *-f(x)* should be minimized. This is because the minimum point of *-f(x)* is some point of *f(x)* at which the maximum is reached.

Let's write an M-file for the function $z(x) = -f(x)$ and save it in the current folder with the name *ex*13.*m*:

```
function z = ex13(x)
z=-(exp(-x(1)^2-x(2)^2)+sin(x(1)+x(2)));
```

Let's call the dialog with *gatool*.

In the *fitness function* field, enter the name of the objective function @*ex*13. Set the values of the GA parameters: the number of variables equal to 2, the number of individuals in the population equal to 10, the number of generations equal to 100 (in the window of the algorithm stopping criterion), the

initial segment equal to [–1; 3]. To build graphs in the plots section, set the checkboxes for *best fitness*, *best individual*, *distance* and press the *start* button.

As a result of the completion of the process, the value of the variable *x* will appear in the *final point* window, corresponding to the minimum of the function, and in the *status* and *result* window, you can see the found minimum value of the objective function *z (x).*

For this task, the results are as follows:

– the maximum of the function is reached at the point $x = 0.46419$, $y = 0.42406$;

– function value $f(0.46419; 0.42406) = 1.449$.

You can also construct other GAs by modeling operators (selection of parental pairs, crossing over, mutation, migration, selection of individuals to a new population, criteria for completing the algorithm) and algorithm parameters.

You can further explore GA optimization by looking at the *Direct Search Toolbox.* To call a window, you can simply write *psearchtool* on the command line.

12.6. Individual tasks

1. According to the number in the group journal from Table 12.1, select the function for determining the extrema using GA in the MATLAB package.

2. Search for the minimum and maximum of the selected function using the command line and the GA graphical interface of the MATLAB package, using the functions *gaoptimset* and *ga*. At the same time, substantiate the choice of all parameters and criteria for stopping the operation of the GA.

3. Calculate the time spent on finding the extrema of a function when implementing GA using the command line of the MATLAB package.

4. Plot the function of the function in order to make sure that the extrema found using GA are correct.

5. Generate M-file codes for each problem of finding extremums of a function using GA.

6. Explain the difference between the results obtained with multiple launches of the GA during the search for the optimal solution.

7. Design the lab's report.

Table 12.1 – Functions for determining extrema using a genetic algorithm

Number in the group journal	The name and type of the function for determining the extrema
1, 8, 15	Sinusoidal function: $f(x) = x_1 \cdot \sin 4x_1 + 1{,}1 \cdot x_2 \cdot \sin 2x_2.$
2, 9, 16	Rastrigin function: $f(x) = 20 + x_1^2 + x_2^2 - 10 \cdot (\cos 2\pi x_1 + \cos 2\pi x_2).$
3, 10, 17	Modified Rastrigin function: $f(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10 \cdot \cos 2\pi x_i),$ for $n = 2.$
4, 11, 18	Schweffel function: $f(x) = 418{,}9829n - \sum_{i=1}^{n}(x_i \cdot \sin\sqrt{\lvert x_i \rvert}),$ for $n = 2.$
5, 12, 19	Eckley function ($n = 2$): $f(x) = 20 + e - 20 \cdot \exp\left(-0{,}2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos 2\pi x_i\right).$
6, 13, 20	Mikhalevich function: $f(x) = -\sum_{i=1}^{n}(\sin x_i \cdot \sin\frac{i x_i^2}{\pi}),$ for $n = 2.$
7, 14, 21	Negnevitsky function: $f(x) = (x_1 - x_1^3 - x_2^3)\exp(-x_1^2 - x_2^2) - (1 - x_1)^2 \exp(-x_1^2 - (x_2 + 1)^2).$

Laboratory work 13

APPLICATION OF GENETIC ALGORITHMS IN OPTIMIZATION PROBLEMS

The purpose of the laboratory work is to obtain and consolidate knowledge, to form practical skills in the application of genetic algorithms to various optimization problems.

13.1. Summary of theory

13.1.1. Application of genetic algorithms to the problem of computer network optimization

An extensive computer network (CN) will serve as an object of genetic optimization. The backbone of the network is a support ring, or backbone, which is a series-connected hub. Hosts are connected to the hubs, which, in turn, can be routers for the second level subnets. Each host can be connected to any router. The optimization task is to find such a host connection scheme in which the traffic on the backbone will be minimal. Backbone traffic is calculated on the basis of statistical data accumulated over a sufficiently long period of time, which makes it possible to estimate the volume of traffic between two hosts.

To optimize traffic, it is necessary to solve the following engineering problems:

- develop and implement a system for collecting traffic statistics on a backbone, consisting of hubs;
- develop a system for presenting input data for a genetic algorithm (GA);
- develop and implement a genetic algorithm.

Organization computer network model

Let us describe the current state of the CN from the optimization point of view. We believe that at some stage in the development of the CN, a decision

was made to lay the central line, purchase and install communication equipment. The list of workstations (nodes) of the CN is known, and for the nodes the location problem has been solved, i.e. the coordinates of the nodes are known. The placement of nodes in the conditions of the institution is subject to the existing structure of divisions, therefore, it is advisable to limit the technical optimization to the framework of the placement of communication equipment. Communication equipment is represented by routers, hubs and / or switches. Each node (workstation, host) can only be connected to one switch / hub, so each switch / hub defines a CN segment. We consider the partitioning of nodes by switch / hub to be known.

Let there be n switch / hub and m nodes in CN. Denote nodes as u_i, $i = 1, ..., m$, and switches/hubs as q_j, $j = 1, ..., n$. The communication channels between q_l and q_k will be denoted as S_{lk}. Restrictions from the point of view of the placement problem are represented by restrictions on the allowable lengths of communication channels. S_{lk} channels are connected in series in most real CNs (Figure 13.1). The channels that close the ring (the dotted line in Figure 13.1) are necessary to improve reliability by quickly forming a new trunk in case of damage to communication channels.

There are two types of communication channels in the context of the task:

– <node> – < switch/hub>;

– < switch/hub> – < switch/hub>.

Any CN has m channels of the first type (according to the number of nodes) and no more than $n \cdot (n – 1)/2$ channels of the second type.

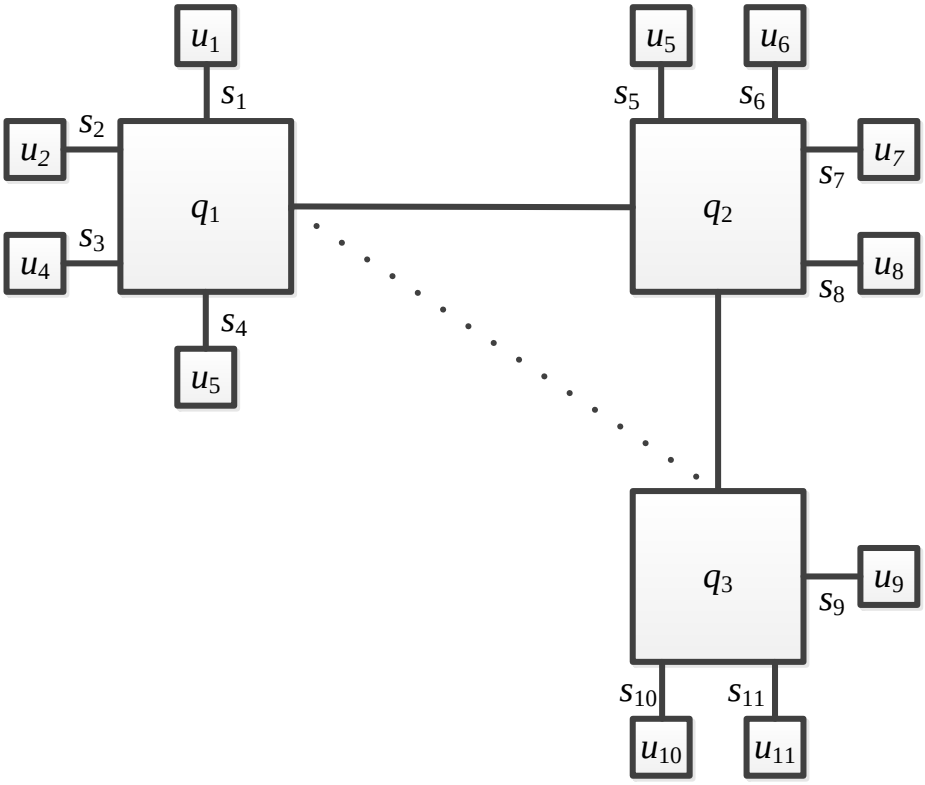

Figure 13.1. Example of CN with switching

During the CN upgrade process, topology change is represented by the following actions: reconnecting nodes to other switches/hubs or laying new communication channels. The effectiveness of each new communication channel is evaluated by the impact on the total traffic of all communication channels. The efficiency function can be recalculated many times for different variants of CN.

$$F^l = \min \| T_k - P_k \|, \quad \text{for} \; \forall \; k = 1, \ldots, K$$

The variable T_k expresses the traffic of channel k, and the variable P_k expresses the maximum channel capacity, K is the number of communication channels. The task of reconnecting nodes to other switches/hubs is the task of dividing m workstations into n groups. There is a limit on the number of connected stations $n_i^{\max}$ for each i-th group.

Solving the problem of reconnecting workstations is a process of directed enumeration of connection options for the purpose of optimization. The dimension of the problem is large even for a medium-sized computer network (up to 1000 nodes), so it seems appropriate to solve the problem using a genetic algorithm.

The coding of the problem solution (chromosome) can be as follows. A variant of splitting nodes into segments, i.e. it is convenient to represent the solution as a series of integers. Let all nodes have unique numbers from 1 to *m* and be ordered according to these numbers. Position *i* contains the number of the switch/hub (from 1 to *n*) to which node *i* is connected. An example of encoding the CN topology shown in Figure 13.1 is shown in Table 13.1.

Table 13.1 – Example of encoding for connecting CN nodes

Switch / Hub	1	1	1	1	2	2	2	2	3	3	3
Node number	1	2	3	4	5	6	7	8	9	10	11

It is needed $[\log_2 n]$ bits for binary encoding of the number q_i of the switch / hub, where [] is the rounding operation upwards.

For the above example, a chromosome using binary coding is shown in Table. 13.2.

Table 13.2 – Binary coded chromosome

Chromosome	01	01	01	01	10	10	10	10	11	11	11
Gene number	1	2	3	4	5	6	7	8	9	10	11

With this coding method and with a free mutation mechanism, it is theoretically possible for forbidden combinations to appear, so the mutation operator needs to be clarified when using a genetic algorithm. The result of mutation of an individual gene (position) is a sequence of operations:

- random selection of a gene (probability 0,001);
- random bit selection;

– inversion of the selected bit;

– operation mod n on the gene containing the result of the inversion $mod_n(x)$.

The recombination operator is performed traditionally, but the border for cutting the parental chromosomes must be the border of the genes. The determination of the chromosome optimality function has the greatest importance, since the determination affects the convergence of evolution. The following function can be taken as the optimality function:

$$\max_{l} \min_{k} \|T_k - P_k\|, \quad \text{for } \forall\ l,\ l \in\ L,\ \forall\ k,\ k \in\ K,$$

where T_k is channel k traffic; P_k is channel capacity; L is the set of all options for the choice of communication equipment; K is the number of communication channels.

The type of communication equipment significantly affects the workload of communication channels. The real CN often contains both switches and hubs. You can improve the capacity of communication channels through the optimal choice of switches or hubs. The problem of choosing communication equipment is set at the level of communication channels, each channel k is characterized by a capacity that is real P_k and maximum P_k^{max} (bit/s).

The intensity of interaction (message transmission) of any pair of nodes is the value of B_{ij} (bit/s). The B_{ij} value is measured over a long period of time and averaged. Averaging can be represented either by calculating the average value B_{ij}^{cp}, or by plotting a probability distribution based on a histogram.

The total traffic per communication channel depends on the channel type. We will consider only channels of the type:

– < switch/hub > – < switch/hub >.

Then the following subtypes of channels can be distinguished:

– < switch > – < switch >;

– <hub > – < hub >;

– <switch > – < hub >.

The total traffic is expressed differently for the three subtypes of communication channels. Consider the total traffic of a channel of type <switch> – <switch>:

$$T_k = \sum_{j \in M_1}^{n} \sum_{j \in M_2}^{n} B_{ij}.$$

The set of vertices M_1 and the set of vertices M_2 are the sets of nodes on one and the other side of the communication channel. The total traffic of the channel <hub> – <hub> is measured differently, since the channel formed by the hubs constitutes a common trunk:

$$T_k = \sum_{\substack{j \in 1 \\ i \neq j}}^{n} \sum_{\substack{j \in 1 \\ i \neq j}}^{n} B_{ij}.$$

The total traffic of the channel <hub> – <switch> can be calculated as follows:

$$T_k = \sum_{\substack{j \in 1 \\ i \neq j}}^{n} \sum_{j \in M_2}^{n} B_{ij}.$$

The need for CN modernization is determined by the degree of proximity of the total traffic and the capacity of the communication channels. The following genetic algorithm is used to find the optimal connection.

Step 1. A population of chromosomes is generated randomly with a check for correctness, i.e. connectivity options encoded in the chromosome.

Step 2. For each chromosome, the value of the optimality function is calculated, i.e. the amount of traffic on the backbone when connecting encoded in the chromosome.

Step 3. The chromosomes are ranked in accordance with the value of the optimality function, the chromosomes are ordered as efficiency decreases.

Step 4. The selection probability is calculated for each chromosome,

$$P_s(a_i^t) = \frac{f(a_i^t)}{\sum_{j=1}^{\lambda} f(a_j^t)},$$

where P_s is the probability of selection; a_i^t is the *i*-th chromosome in the *t*-th generation; *f* is the optimality function; *t* is the generation number; λ is the population size.

Step 5. The selection probabilities are scaled, the highest probability is taken as one, the smallest one is taken as zero, the rest of them are proportionally scaled:

$$P_{sl} = \frac{P_{s_i} - P_{min}}{P_{max} - P_{min}},$$

where P_{sl} is the scaled probability; P_{s_i} is the probability of selection; P_{min} is the minimum probability in the population; P_{max} is the maximum probability in the population.

Step 6. The formation of a new population begins: an elite selection is made, i.e. a certain number of "old" chromosomes are transferred to a new population.

Step 7. A crossing over of randomly selected chromosomes is performed. The crossing over point is chosen randomly. The process continues until the size of the new population is equal to the size of the original.

Step 8. Go to step 2 and the loop repeats the specified number of times.

Step 9. Stop.

13.1.2. Application of genetic algorithms to the problem of placing radioelements in the device case

The structure of the problem of placing different-sized elements in space is given as follows:

– space limitations of the accommodation volume (for example, the dimensions of a particular building, determined by the enterprise standard);

– placement elements specified by their dimensions (volumes).

In practice, the task of placement in space is often replaced by placement on an installation area, which is called the mounting field. Each element intended to be placed is then represented by its installation area, which is roughly a rectangle, i.e. two dimensions: length and width.

Let's assume that the artboard is also a rectangle. According to the degree of complexity, a simpler task of placing identical elements on a field with multiple dimensions and a more complex task of placing elements with different installation areas are distinguished. We will solve the problem of placing different-sized elements on a limited mounting field.

The initial data are: a, b are the dimensions of the installation space; $\{(a_1, b_1), ..., (a_i, b_i), ..., (a_n, b_n))$ is the set of placement elements; C is a matrix of connections of accommodation elements, which is a matrix of connectivity. You need to find an option for placing elements on the mounting space

$$Z = ((x_1, y_1), ..., (x_i, y_i), ..., (x_n, y_n)),$$

where x_i, y_i are the coordinates of the gravity center of the installation area of the placement element i such that the overlap area of the areas of the placed elements is equal to zero, and the total length of the connections is minimal.

The placement problem is posed as a problem of optimizing a function that expresses a normalized estimate of the sum of the penalty for overlapping the areas of the placed electronic radio elements (ERE) and the total length of connections:

$$F = \min_{z_j \in Z}(k \cdot O(L(z_j)) + P(S_{\text{total}}(z_j)),$$

where k is the weight coefficient; $O(L(z_j))$ is an estimate of the total length of connections, reduced to the interval [0, 1]; z_j is a placement option; $P(S_{\text{общ}}(z_j))$ is a penalty function for overlapping areas, taking values from the interval [0,1]; $S_{\text{общ}}$ is the total overlapping area of the areas of the placed elements.

The total length of the connections is calculated by the formula:

$$L = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij} \cdot c_{ij},$$

where d_{ij} is the distance between installation positions of elements $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$; c_{ij} is the number of connections between elements i and j from the original adjacency matrix C.

Normalization of the total length of connections can be carried out by calculating the ratio $L(z_j)$ to $L_{\max}$, where

$$L_{\max} = n^2 \cdot \sqrt{a^2 + b^2};$$

$$O(L(z_j)) = L(z_j) / L_{\max}.$$

The total area of the overlap is calculated using the following formula:

$$S_{\text{пер}} = \sum_{i=1}^{n} \sum_{j=1}^{n} S_{ij},$$

where S_{ij} is the overlap area of the elements z_i and z_j,

$$S_{ij} = [0{,}5(a_2 + a_1) - |x_2 - x_1|][0{,}5(b_2 + b_1) - |y_2 - y_1|].$$

Area Overlapping Penalty Function is

$$P(S_{\text{total}}) = S_{\text{total}}/nab.$$

13.1.3. Study of the genetic algorithms efficiency for the problem of placing radioelements in the device case

We study the effectiveness of the genetic algorithm in solving the problem of placing elements on a plane. A characteristic feature of using a standard genetic algorithm for solving practical problems is the need to refine its parameters. To create a software implementation of the genetic algorithm for the placement of different-sized radio-electronic elements, the following modification of the GA is required.

1. The chromosome is a linked list of pairs of coordinates of the centers of gravity of the placement elements. Each coordinate is described by a real number.

2. As a mutation operator, a probabilistic change in a random position of a chromosome is used. The single-point version of the crossover is used as the recombination operator.

3. The condition for the completion of the evolution is set in the form of determining the number of generations or in the form of a condition for achieving zero overlap of elements.

The results of the analysis of the behavior of the genetic algorithm depending on the values of the probability of mutation are shown in Figure 13.2.

The experiment was carried out for mutation probabilities in the range from 0.1 to 0.6 with a step of 0.1. A probability value greater than 0.6 violates the convergence of the GA optimality function.

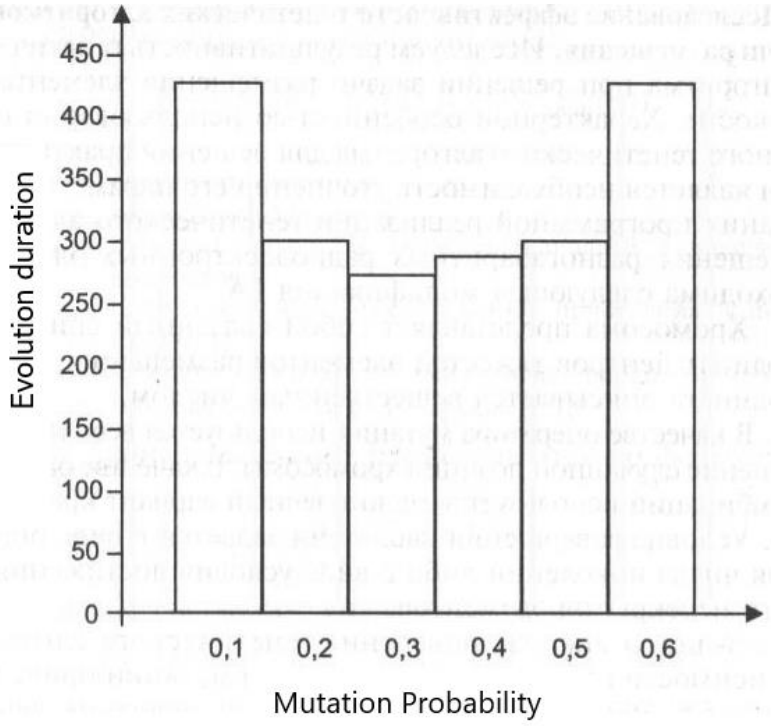

Figure 13.2. Evolution duration depending on the value of the mutation probability diagram

In addition to the standard GA, evolutionary strategies can be used to solve the placement problem: the "mutation only" strategy with a proportional selection operator; recombination strategy (m, k). Such a recombination operator assumes m parents and k descendants. The parameters m and k are set by the user. Theoretically, the constraints on the parameters have the following form: $m = 1, ..., l - 1$; $k = 1, ..., m$. The parameter l is the length of the chromosome (or in our case, the number of placement elements). The following combinations of parents and descendants were experimentally studied: (2, 2), (3, 1), (3, 2), (3, 3), (4, 1), (4, 2), (4, 3), (4, 4). It should be noted that when combining (2, 1), we get the usual single-point crossing-over as the recombination operator, in which two parents and one randomly selected descendant participate.

Figures 13.3 and 13.4 show the types of graphs of the genetic algorithm optimality for comparison.

A number of experiments with an algorithm that uses a strategy with an alternative recombination operator revealed a special performance of the recombination operator (3, 3). Although, in general, the results of this type of algorithm are worse than in the case of a standard genetic algorithm with one-point crossover. First of all, this is due to the fact that a significant increase in chromosome cut points leads to the destruction of good subregions of potential solutions. This is noticeable by the numerous "bursts" on the graph in Figure 13.4.

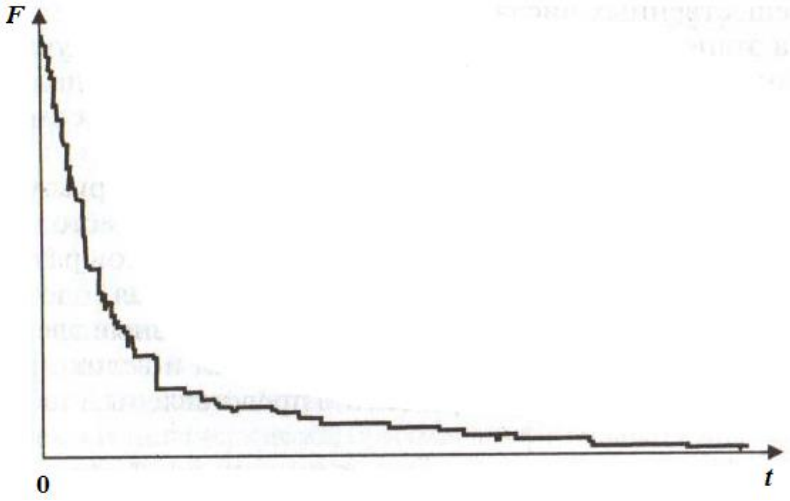

Figure 13.3. Graph of the GA optimality function for a single-point crossover

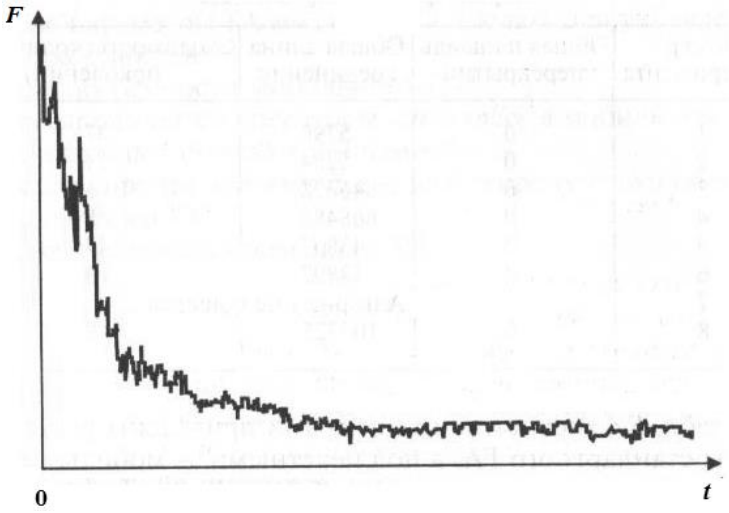

Figure 13.4. Graph of the GA optimality function for a strategy with recombination (m, k)

When using a mobile genetic algorithm in practice, difficulties arise both with the formulation of the initial population of solutions and with the optimality function. The structure of a chromosome is a list of pairs of coordinates ordered by element number: $((1, x_1, y_1), ..., (i, x_i, y_i), ..., (v, x_v, y_v))$.

In the mobile genetic algorithm, a chromosome is encoded by a list of <gene number, gene value> pairs. When solving the placement problem, the number of the gene coincides with the number of the placed element, and the value of the gene coincides with a pair of coordinates. Thus, the gene number is an integer, and the coordinates are two real numbers.

At the initialization stage, λ strings are generated randomly. The parameter λ is the population size, it is set by the user. The optimality function for each chromosome is calculated as the overlap area of the elements.

The effectiveness of mobile GA for structural synthesis problems can be tested on the following four test sets: "large mounting field and few placement elements"; "many placement elements" (high packing density); "low packing density, but different placement elements"; "High packing density and great variety." The results of the experiments are presented in Table 13.3.

Table 13.3 – Standard GA parameters

Experiment number	Total overlap area	Total length of connections	Convergence (number of generations)
1	0	5796	37
2	0	5224	11
3	0	646628	6
4	0	668482	8
5	0	43835	47
6	0	38897	13
7	Algorithm didn't converge		
8	0	103325	8

Table 13.3 shows the results of the standard GA under even numbers, and the results of the mobile GA under odd numbers. It can be concluded on the effectiveness of the mobile genetic algorithm based on the experiments.

13.2. Individual tasks

1. For students with even numbers in the journal of the group: it is necessary to write a program in any high-level language that implements the solution of the formulated problem of optimizing a computer network using the described genetic algorithm. For students with odd numbers in the group journal: it is necessary to write a program that implements the solution of the formulated problem of placing radio elements in the device case.

2. Justify the choice of all parameters and criteria for stopping the operation of the genetic algorithm.

3. Design the lab's report.

Laboratory work 14

MATLAB APPLICATION FOR MODELING A HEBB NEURAL NETWORK

The purpose of the laboratory work is to obtain and consolidate knowledge, to form practical skills in working with the MATLAB package when using M-files and developing programs for solving artificial intelligence problems.

14.1. Summary of theory

14.1.1. Formal neurons of artificial neural networks

When modeling neural networks, a simple processor element is usually used as artificial neurons, shown in Fig. 14.1. Its inputs receive a vector $X = (x_1, \ldots, x_n)$ of input signals that are output signals of other neurons, as well as a single bias signal. All input signals, including the bias signal, are multiplied by the weights of their connections and summed:

$$S = \sum_{i=1}^{n} x_i w_i + w_0, \qquad (14.1)$$

where S is the total input signal; w_i $(i = \overline{1,n})$ are weight coefficients of connections of input signals $x_1, \ldots, x_n$; w_0 is a weight coefficient of connection of bias signal.

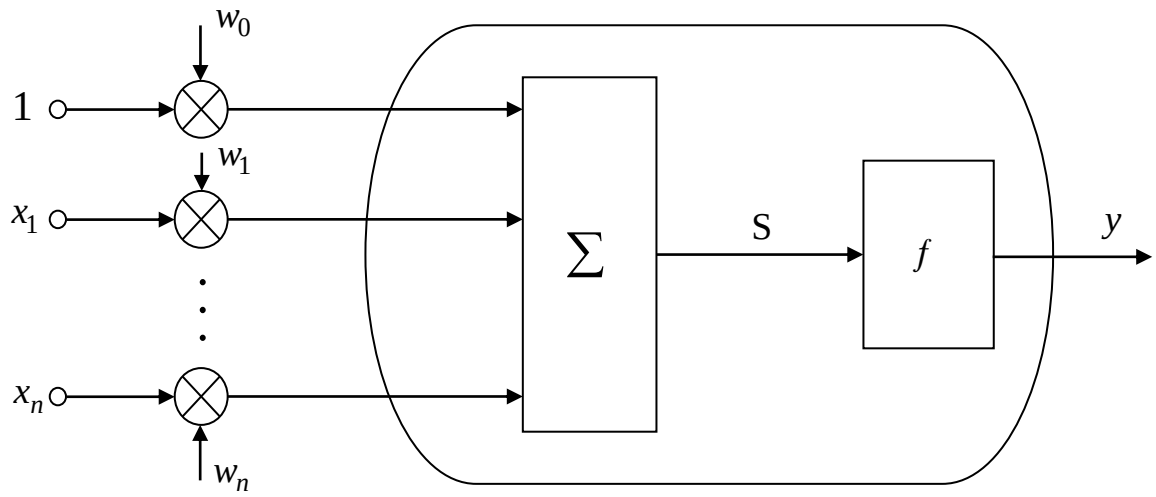

Figure 14.1. Processing element used in conventional neural networks

The received signal S is fed to the input of the block that implements the neuron activation function f. Typical activation functions are binary

$$y = \begin{cases} 1, & \text{if} \quad S > 0, \\ 0, & \text{if} \quad S \le 0 \end{cases} \tag{14.2}$$

or bipolar

$$y = \begin{cases} 1, & \text{if} \quad S > 0, \\ -1, & \text{if} \quad S \le 0 \end{cases} \tag{14.3}$$

Many authors, when describing a neuron model, do not use a bias signal, but a neuron θ threshold, which leads to an equivalent element model. In this case, expressions (14.2) and (14.3) take the form, respectively:

$$y = \begin{cases} 1, & \text{if} \quad S > \theta, \\ 0, & \text{if} \quad S \le \theta, \end{cases} \tag{14.4}$$

$$y = \begin{cases} 1, & \text{if} \quad S > \theta, \\ -1, & \text{if} \quad S \le \theta, \end{cases} \tag{14.5}$$

where

$$S = \sum_{i=1}^{n} w_i x_i \,. \qquad (14.6)$$

A graphic representation of the binary and bipolar activation functions for this case is shown in Figure 14.2, *a* and 14.2, *b*.

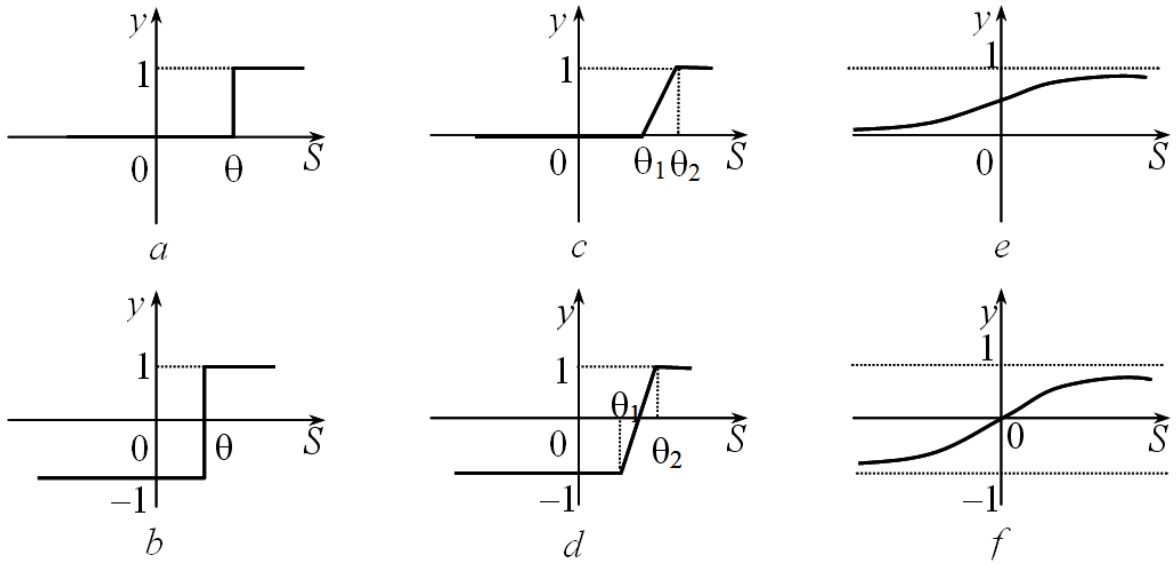

Figure 14.2. Neuron activation functions

From the comparison of expressions (14.1) – (14.3) and (14.4) – (14.6) it follows that each neuron threshold value θ can be assigned a weight coefficient w_0 of the bias signal connection and vice versa.

Linear binary or bipolar activation functions are used less frequently. (Figure 14.2, *c* and 14.2, *d*):

$$y = \begin{cases} -a, \text{ when } S < \theta_1, \\ kS + a_0, \text{ when } \theta_1 \le S \le \theta_2, \\ 1, \text{ when } S > \theta_2, \end{cases} \qquad (14.7)$$

where a is equal to zero for binary output signals of neurons and a is equal to minus one for bipolar signals; k, a_0 are the constant coefficients.

In addition to those given in the theory of neural networks, the following nonlinear activation functions are also often used:

- binary sigmoid (Figure 14.2, *e*):

$$y = \frac{1}{1 + e^{-\tau S}}, \tag{14.8}$$

where τ is the constant coefficient;

- bipolar sigmoid (Figure 14.2, *f*):

$$y = \frac{2}{1 + e^{-\tau S}} - 1. \tag{14.9}$$

The given models of artificial neurons ignore many known properties of biological prototypes. For example, they do not take into account the time delays of neurons, the effects of frequency modulation, local excitation, and related phenomena of subthreshold temporal and spatial summation, when the cell is excited not by simultaneously arriving impulses, but by sequences of excitatory signals arriving at short intervals. The periods of absolute refractoriness are also not taken into account, during which nerve cells cannot be excited, i.e. as if they have an infinitely high excitation threshold, which then decreases to a normal level in a few milliseconds after the passage of the signal. It is easy to continue this list of differences, which many biologists consider decisive, but artificial neural networks still reveal a number of interesting properties that are characteristic of biological prototypes.

14.1.2. Solving recognition problems based on individual neurons (Hebb's rule)

Artificial neural networks designed to solve a variety of specific tasks can contain from a few neurons to thousands and even millions of elements.

However, even a single neuron (Figure 14.1) with a bipolar or binary activation function can be used to solve simple problems of image recognition and classification. The choice of bipolar (1, –1) or binary (1, 0) representation of signals in neural networks is carried out based on the problem being solved, and in many cases it is equivalent. There is a range of tasks in which binary coding of signals is more convenient, however, in general, bipolar representation of information is more preferable.

Since the output signal of a binary neuron (Figure 14.1) takes only two values, the neuron can be used to classify the presented images into two classes.

Let there be a set M of images for which the correct classification into two classes is known: $X^1 = \{X^{11}, X^{12}, ..., X^{1q}\}$, $X^2 = \{X^{21}, X^{22}, ..., X^{2p}\}$, $X^1 \cup X^2 = M$, $X^1 \cap X^2 = \varnothing$. And let the output signal $y = 1$ corresponds to the first class X^1, and the signal $y = -1$ corresponds to the first class X^2. If, for example, some image $X^\alpha = (X_1^\alpha, ..., X_n^\alpha)$, $X^\alpha \in M$ is presented and its weighted sum of input signals exceeds zero:

$$S = \sum_{i=1}^{n} x_i^\alpha w_i + w_0 > 0,$$

then the output signal $y = 1$ and, therefore, the input image X^α belongs to the class X^1. If $S \le 0$, then $y = -1$ and the presented image belongs to the second class.

It is also possible to use a separate neuron to select from the set of classes $M = \{X^1 = \{X^{11}, \ldots, X^{1k}\}, \ldots, X^i = \{X^{i1}, \ldots, X^{iq}\}, \ldots, X^p = \{X^{p1}, \ldots, X^{pm}\}\}$ images of a single class X^i. In this case, it is assumed that one of the two possible output signals of the neuron (for example, 1) corresponds to the class X^i, and the second output signal corresponds to all other classes. Therefore, if the input image X^α leads to the appearance of a signal $y = 1$, then $X^\alpha \in X^i$, if $y = -1$ (or $y = 0$, if binary coding is used), then this means that the presented image does not belong to allocated class.

A recognition system based on a single neuron divides the entire space of possible solutions into two regions using a hyperplane

$$x_1w_1 + x_2w_2 + \ldots + x_nw_n + w_0 = 0.$$

For two-dimensional input vectors, a straight line is the boundary between the two image classes: the input vectors which are above this line belong to one class, and the input vectors which are below it belong to another class.

Several methods can be used to adapt, tune, or train neuron connection weights. Let's consider one of them, called "Hebb's rule". Hebb, exploring the mechanisms of functioning of the central nervous system, suggested that learning occurs by strengthening connections between neurons whose activity coincides in time. Although this assumption is far from being always fulfilled in biological systems and does not exhaust all types of training, it is very effective when training single-layer neural networks with bipolar signals.

In accordance with the Hebb rule, if an incorrect output signal y corresponds to the presented bipolar image $X = (x_1, \ldots, x_n)$, then the weights w_i $(i = \overline{1,n}.)$ of neuron connections are adapted according to the formula

$$w_i\,(t + 1) = w_i\,(t) + x_i y, \quad i = \overline{0,n}, \tag{14.10}$$

where $w_i\,(t)$, $w_i\,(t + 1)$ are respectively, the weight of the i-th connection of the neuron before and after adaptation; x_i $(i = \overline{1,n},)$ are input image components; $x_0 \equiv 1$ is bias signal; y is the output signal of the neuron.

In a more complete and strict form, the algorithm for adjusting the weights of neuron connections using the Hebb rule is as follows:

Step 1. The set $M = \{(X^1, t^1), \ldots, (X^m, t^m)\}$, is set. It consists of pairs (input image $X^k = \left(x_1^k, \ldots, x_n^k\right)$, required neuron output signal t^k, $k = \overline{1,m}$). Neuron connection weights are initiated:

$$w_i = 0, \quad i = \overline{0, n}.$$

Step 2. For each pair (X^k, t^k), $k = \overline{1, m}$ the steps 3 – 5 are performed until the stop conditions are met.

Step 3. A set of neuron inputs is initiated:

$$x_0 = 1, \; x_i = x_i^k, \quad i = \overline{1, n}.$$

Step 4. The output signal of the neuron is initiated: $y = t^k$.

Step 5. The weights of neuron connections are adjusted according to the rule

$$w_i\,(new) = w_i\,(old) + x_i\,y, \quad i = \overline{0, n}\,.$$

Step 6. Stop conditions checking.

The corresponding output signal y^k is calculated for each input image X^k.

$$y^k = \begin{cases} 1, \text{ if } S^k > 0, \\ -1, \text{ if } S^k \le 0, \end{cases} \quad k = \overline{1, m},$$

where

$$S^k = \sum_{i=1}^{n} x_i^k w_i + w_0\,.$$

If the vector (y^1, ..., y^m) of calculated output signals is equal to the vector (t^1, ..., t^m) of given neuron signals, i.e. given output signal corresponds to each input image, then the calculations stop (go to step 7), if (y^1, ..., y^m) $\neq$ (t^1, ..., t^m), then go to step 2 of the algorithm.

Step 7. Stop.

Example 1. Let it be required to train a bipolar neuron to recognize the images X^1 and X^2, shown in Figure 14.3.

Figure 14.3. Input images

In this case, we require that the output signal of the neuron "+1" corresponds to the image X^1 and the output signal of the neuron "–1" corresponds to the image X^2.

Applying the Hebb algorithm gives the following results:

Step 1. The set is set:

$$M = \{(X^1 = (1, -1, 1, 1, 1, 1, -1, -1, 1), 1),$$
$$(X^2 = (1, 1, 1, 1, -1, 1, 1, -1, 1), -1)\};$$

and neuron connection weights are initiated: $w_i = 0,\ i = \overline{0, 9}$.

Step 2. Steps 3 – 5 of the algorithm are performed for each of two pairs $(X^1, 1), (X^2, -1)$

Step 3. A set of neuron inputs is initiated for the image of the first pair: $x_0 = 1, x_i = x_i^1,\ i = \overline{0, 9}$.

Step 4. The output signal of the neuron is initiated for the image of the first pair: $y = t^1 = 1$.

Step 5. Neuron connection weights are adjusted according to Hebb's rule $w_i = w_i + x_i^1 y \quad (i = \overline{0, n})$:

$$w_0 = w_0 + x_0 y = 0 + 1 \cdot 1 = 1;\quad w_1 = w_1 + x_1^1 y = 0 + 1 \cdot 1 = 1;$$

$$w_1 = w_3 = w_4 = w_5 = w_6 = w_9 = 1; \;\; w_2 = w_2 + x_2^1 y = 0 + (-1)\cdot 1 = -1;$$
$$w_2 = w_7 = w_8 = -1.$$

Step 3. A set of neuron inputs is initiated for the image X^2 of the second pair:

$$x_0 = 1,\ x_i = x_i^2,\ i = \overline{0, 9}.$$

Step 4. The output signal of the neuron is initiated for the image of the second pair (X^2, t^2):

$$y = t^2 = -1.$$

Step 5. Neuron connection weights are adjusted:

$$w_0 = w_0 + x_0 y = 1 + 1\cdot(-1) = 0;$$
$$w_1 = w_1 + x_1^2 y = 1 + 1\cdot(-1) = 0;$$
$$w_1 = w_3 = w_4 = w_6 = w_9 = 0;$$
$$w_2 = w_2 + x_2^2 y = -1 + 1\cdot(-1) = -2;$$
$$w_2 = w_7 = -2;$$
$$w_5 = w_5 + x_5^2 y = 1 + (-1)\cdot(-1) = 2;$$
$$w_8 = w_8 + x_8^2 y = -1 + (-1)\cdot(-1) = 0.$$

Step 6. Checking the stop conditions.

The input and output signals of the neuron are calculated upon presentation of the image X^1:

$$S^1 = \sum_{i=1}^{9} x_i^1 w_i + w_0 = 1\cdot 0 + (-1)\cdot(-2) + 1\cdot 0 + 1\cdot 0 + 1\cdot 2 + 1\cdot 0 + (-1)\cdot(-2) +$$
$$+ (-1)\cdot 0 + 1\cdot 0 + 0 = 6,$$
$$y^1 = 1, \text{ since } S^1 > 0.$$

The input and output signals of the neuron are calculated upon presentation of the image X^2:

$$S^2 = \sum_{i=1}^{9} x_i^2 w_i + w_0 = 1\cdot 0 + 1\cdot(-2) + 1\cdot 0 + 1\cdot 0 + (-1)\cdot 2 + 1\cdot 0 + 1\cdot(-2) +$$
$$+ (-1)\cdot 0 + 1\cdot 0 + 0 = -6,$$
$$y^2 = -1, \text{ since } S^2 < 0.$$

Since the vector $(y^1, y^2) = (1, -1)$ is equal to the vector (t^1, t^2), then the calculations stop, since the goal is achieved - the neuron correctly recognizes the given images.

Step 7. Stop.

14.1.3. Hebb neural network

The use of a group of m bipolar or binary neurons $A_1, \ldots, A_m$ (Figure 14.4) makes it possible to significantly expand the capabilities of the neural network and recognize up to 2^m different images. The use of this network for recognizing 2^m (or close to 2^m numbers) different images can lead to unsolvable problems of adapting the neural network connection weights. Therefore, it is often recommended to use this architecture to recognize only m different images, giving each of them a single output only at the output of one A-element (the outputs of the rest of them should take the value "–1" for bipolar neurons or "0" for binary neurons).

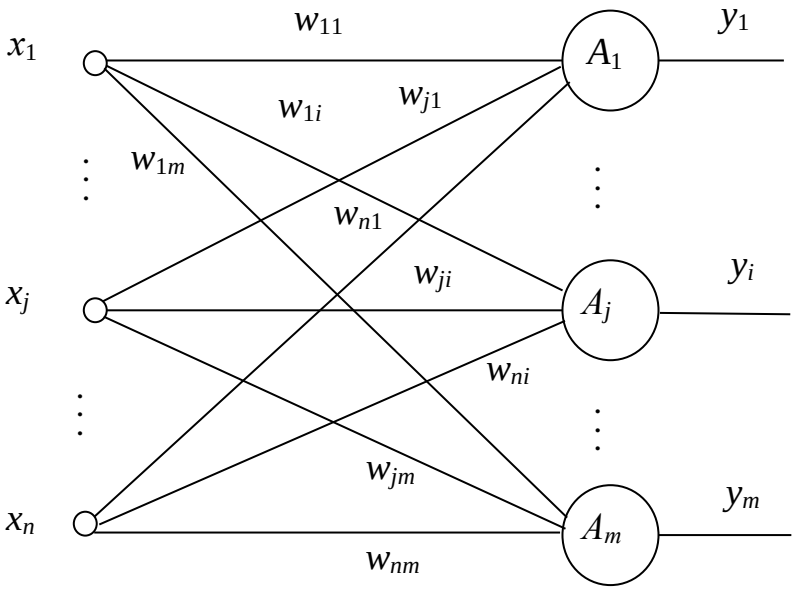

Figure 14.4. Neural network of *m* elements

A single-layer neural network with binary neurons, shown in Figure 14.4, can be trained using an algorithm based on the Hebb rule. In this case, it is called the Hebb network. The use of other learning algorithms for the same network also leads to a change in the name of the neural network. The use of their learning algorithms in the names of networks is typical for the theory of neural networks. For a bipolar representation of signals, it is possible to train a neural network using the following algorithm::

Step 1. Set $M = \{(X^1, t^1), \ldots, (X^m, t^m)\}$, is set. It consists of pairs (input image $X^k = \left(x_1^k,\ldots,x_n^k\right)$, required neuron output signal t^k, $k = \overline{1,m}$). Neuron connection weights are initiated:

$$w_{ji} = 0, \quad j = \overline{0,n}, \quad i = \overline{1,m}.$$

Step 2. Each pair (X^k, t^k) is checked for the correctness of the reaction of the neural network to the input image. If the resulting network output vector

(y_1^k, …, y_m^k) differs from the given one $t^1 = (t_1^k, \ldots, t_m^k)$, then steps 3 – 5 are performed.

Step 3. A set of neurons inputs is initiated: $x_0 = 1$, $x_j = x_{jk}$, $j = \overline{1,n}$.

Step 4. Output signals of neurons are initiated: $y_i = t_i^k$, $i = \overline{0,m}$.

Step 5. The weights of neuron connections are adjusted according to the rule:

$$w_{ji}(new) = w_{ji}(old) + x_j y_i, \quad j = \overline{0,n}, \quad i = \overline{0,m}.$$

Step 6. Stop conditions are checked, i.e. the correct functioning of the network upon presentation of each input image. If the conditions are not met, then go to step 2 of the algorithm, otherwise, terminate the calculations (go to step 7).

Step 7. Stop.

14.2. Individual tasks

1. Design a Hebb network structure capable of recognizing four different letters of your first or last name.

2. Develop an algorithm and a program using M-files that simulates the Hebb network. It is obligatory to provide for the possibility of situations with unsolvable problems of adapting the weights of the neural network connections in the algorithm.

3. Train a Hebb Neural Network to recognize four given letters of your first or last name.

4. Demonstrate the performance of the network when presented with training images and images containing errors.

5. Design the lab's report.

Laboratory work 15

NEURO-FUZZY MODELING IN THE MATLAB ENVIRONMENT

The purpose of the laboratory work is to obtain and consolidate knowledge about modeling methods and principles of functioning of neuro-fuzzy systems, as well as the formation of practical skills in designing neuro-fuzzy networks in the MATLAB package.

15.1. Summary of theory

As a rule, in a state of unstable economy, characterized by frequent changes in economic conditions, managerial decision-making is carried out in conditions of uncertainty, as a result of which the task of planning economic activity and prognosis its results is one of the most complex and controversial.

There are a number of classical methods for prognosis economic indicators based on the apparatus of mathematical statistics. The methods of analysis and modeling of time series, methods of multivariate regression analysis are among them. A feature of these methods is the need for a clear specification of the constructed models, in addition, additional difficulties for the use of these methods are created by the non-stationarity of the studied economic processes.

A promising direction in the field of solving prognosis problems is the use of the apparatus of artificial neural networks and neuro-fuzzy networks.

The principle of operation of a neural network is as follows. Based on the existing data set, a certain relationship is constructed between the input and output variables of the system. In this case, in the process of training the network, the parameters (weights) of the resulting functional relations are adjusted. (As a rule, the identification and definition of this dependence in an explicit form is not possible due to the above reasons). The neural network

model is positioned as a "black box" due to the fact that the internal algorithm for its tuning is not "transparent", and the results and relationships are difficult to interpret.

Fuzzy neural networks or hybrid networks are designed to combine the advantages of neural networks and fuzzy inference systems. On the one hand, they allow developing and presenting system models in the form of fuzzy production rules, which are visual and easy to interpret. On the other hand, neural network methods are used to build fuzzy production rules, which is a more convenient and less time-consuming process for system analysts.

These guidelines discuss the basic concepts of neuro-fuzzy networks, as well as the tools for their development provided by the MATLAB system.

For forecasting, we use the *TSK* fuzzy network. The generalized inference scheme of the *TSK* model using M rules and N variables x_j can be represented as:

$$IF\ (x_1\ IS\ A_1^{(1)})\ AND\ (x_2\ IS\ A_2^{(1)})\ AND\ \ldots\ AND\ (x_n\ IS\ A_n^{(1)}),$$
$$THEN\ y_1 = p_{10} + \sum_{j=1}^{N} p_{1j} x_j$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$IF\ (x_1\ IS\ A_1^{(M)})\ AND\ (x_2\ IS\ A_2^{(M)})\ \ AND\ \ldots\ AND\ (x_n\ IS\ A_n^{(M)}),$$
$$THEN\ y_M = p_{M0} + \sum_{j=1}^{N} p_{Mj} x_j\ .$$

The $IF\ (x_i\ IS\ A_i)$ condition is implemented by the fuzzification function, which is represented by the generalized Gaussian function separately for each variable x_i:

$$\mu_A(x_i) = \frac{1}{1 + \left(\frac{x_i - c_i}{\sigma_i} \right)^{2b_i}}, \qquad (15.1)$$

where $\mu_A(x_i)$ represents the operator A_i. In fuzzy networks, it is advisable to set this condition in the form of an algebraic product, from which it follows that for the *k*-th inference rule

$$\mu_A^{(k)}(x) = \prod_{j=1}^{N} \left[\frac{1}{1 + \left(\frac{x_i - c_j^{(k)}}{\sigma_j^{(k)}} \right)^{2b_j^{(k)}}} \right]. \qquad (15.2)$$

With *M* inference rules, the aggregation of the output result of the network is performed according to the formula

$$y = \sum_{i=1}^{M} \frac{w_i}{\sum_{j=1}^{N} w_j} \left(p_{i0} + \sum_{j=1}^{N} p_{ij} x_j \right), \qquad (15.3)$$

which can be represented in the form

$$y(x) = \frac{1}{\sum_{k=1}^{N} w_k} \left(\sum_{k=1}^{M} w_k y_k(x) \right), \qquad (15.4)$$

where $y_k(x) = p_{k0} + \sum_{j=1}^{N} p_{kj} x_j$.

The weights w_k present in this expression are interpreted as the significance of the components $\mu_A^{(k)}(x)$, defined by formula (15.1). Under this condition, formula (15.4) can be associated with a multilayer network structure (Figure 15.1). This network has five layers:

– the first layer performs separate fuzzification of each variable x_i (i = 1, 2, …, N), determining for each *k*-th inference rule the value of the

membership coefficient $\mu_A^{(k)}(x_i)$ in accordance with the applied fuzzification function. This is a parametric layer with parameters $c_j^{(k)}$, $\sigma_j^{(k)}$, $b_j^{(k)}$, to be adapted during the learning process;

– the second layer (non-parametric) performs aggregation of individual variables x_i, determining the resulting value of the membership coefficient $w_k = \mu_A^{(k)}(x)$ for the vector *x* (inference rule activation level) in accordance with formula (15.2);

– the third layer is a *TSK* function generator that calculates the values of $y_k(x) = p_{k0} + \sum_{j=1}^{N} p_{kj} x_j$. In this layer, the signals $y_k(x)$ are also multiplied by the values w_k, formed on the previous layer. This is a parametric layer in which linear weights p_{kj} for $k = 1, 2, \ldots, M$ and $j = 1, 2, \ldots, N$, are to be adapted and determine the consequence function of the *TSK* model;

– the fourth layer (non-parametric) consists of two adder neurons, one of which calculates the weighted sum of signals $y_k(x)$, and the second neuron determines the sum of weights $\sum_{k=1}^{M} w_k$;

– the last, fifth layer (normalizing), consists of a single output neuron, in which the weights are normalized in accordance with formula (15.4). The output signal *y(x)* is determined by the expression corresponding to dependence (15.3),

$$y(x) = f(x) = \frac{f_1}{f_2}. \qquad (15.5)$$

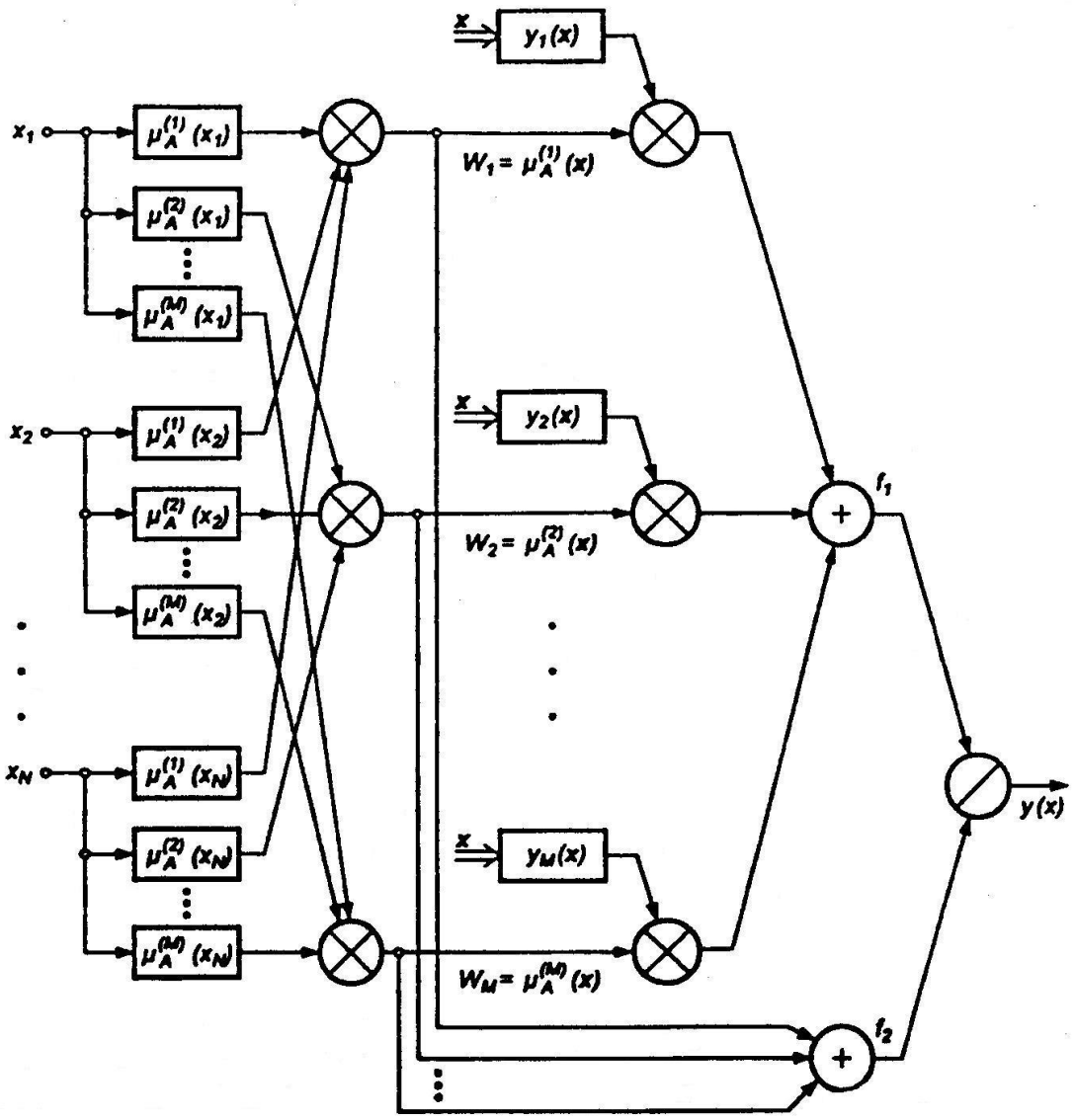

Figure 15.1. The structure of the *TSK* fuzzy neural network

It follows from the above description that the *TSK* fuzzy network contains only two parametric layers (first and third), the parameters of which are refined in the learning process. The parameters of the first layer will be called nonlinear parameters, since they refer to the nonlinear function (15.1), and the parameters of the third layer will be called linear weights, since they refer to the parameters p_{kj} of the *TSK* linear function.

Refining the functional dependence (15.4) for the *TSK* network, we obtain:

$$y(x) = \frac{1}{\sum_{k=1}^{M}\left[\prod_{j=1}^{N}\mu_A^{(k)}(x_j)\right]}\sum_{k=1}^{M}\left[\prod_{j=1}^{N}\mu_A^{(k)}(x_j)\right]\left[p_{k0} + \sum_{j=1}^{N} p_{kj}x_j\right]. \qquad (15.6)$$

If we accept that the condition parameters are fixed at a particular moment in time, then the function *y(x)* is linear regarding variables x_i (i = 1, 2, …, N).

Given N input variables, each rule generates N+1 variables $p_j^{(k)}$ of the linear dependence *TSK*. With M inference rules, this gives $M(N+1)$ linear network parameters. In turn, each membership function uses three parameters (c, s, b) to be adapted. If we accept that each variable x_i is characterized by its own membership function, then with M inference rules we get $3MN$ non-linear parameters. In sum, this gives $M(4N+1)$ linear and non-linear parameters, the values of which must be selected in the process of network training.

In practice, fewer independent membership functions for individual variables are used to reduce the number of adaptable parameters. In this case, they are guided by rules in which the membership functions of various variables are combined. If we accept that each variable x_i has m different membership functions, then the maximum number of rules that can be created by combining them is: $M = m^N$ (with three membership functions spread over two variables, this is $3^2 = 9$ inference rules). Thus, the total number of nonlinear network parameters with M inference rules decreases from $3MN$ in the general case to $3NM^{1/N}$. The number of linear parameters with such a modification remains unchanged, i.e. $M(N+1)$.

Hybrid network as an adaptive system of neuro-fuzzy inference. A hybrid network is a multilayer neural network of a special structure without feedback, which uses ordinary (not fuzzy) signals, weights and activation functions, and the summation operation is based on the use of a fixed T-norm, S-norm, or some other continuous operation. In this case, the values of the inputs, outputs and weights of the hybrid neural network are real numbers from the segment [0, 1].

The main idea underlying the hybrid network model is to use the existing data sample to determine the parameters of membership functions that best fit some fuzzy inference system. In this case, well-known procedures for training neural networks are used to find the parameters of membership functions.

In the *Fuzzy Logic Toolbox* package of the MATLAB system, hybrid networks are implemented in the form of the so-called adaptive system of neuro-fuzzy inference *ANFIS*. On the one hand, the *ANFIS* hybrid network is a neural network with a single output and multiple inputs, which are fuzzy linguistic variables. In this case, the terms of the input linguistic variables are described by the membership functions standard for the MATLAB system, and the terms of the output variable are represented by a linear or constant membership function.

On the other hand, the *ANFIS* hybrid network is a zero or first order Sugeno-type *FIS* fuzzy inference system in which each of the fuzzy production rules has a constant weight equal to 1.

15.2. Modeling and implementation of a neuro-fuzzy network in the MATLAB environment

In the *Fuzzy Logic Toolbox* package of the MATLAB system, hybrid networks are implemented in the form of adaptive systems of neuro-fuzzy inference *ANFIS*. At the same time, the development and research of hybrid networks is possible:

– in interactive mode using a special graphic editor of adaptive networks, called the editor *ANFIS*;

– in command line mode by entering the names of the corresponding functions with the necessary arguments directly into the command window of the MATLAB system.

The *ANFIS* editor allows you to create or load a specific model of an adaptive fuzzy inference system, train it, visualize its structure, change and

configure its parameters, and use the configured network to obtain fuzzy inference results.

15.2.1 ANFIS editor description

ANFIS is an acronym for *Adaptive Neuro-Fuzzy Inference System*. The *ANFIS* editor allows you to synthesize neuro-fuzzy networks from experimental data automatically. A neural fuzzy network can be considered as one of the varieties of Sugeno-type fuzzy inference systems. At the same time, the membership functions of the synthesized systems are tuned (trained) in such a way as to minimize deviations between the results of fuzzy modeling and experimental data. The *ANFIS* editor is loaded using the *anfisedit* command. A graphic window will appear (Figure 15.2), which contains the functional areas of the *ANFIS* editor as a result of executing the command.

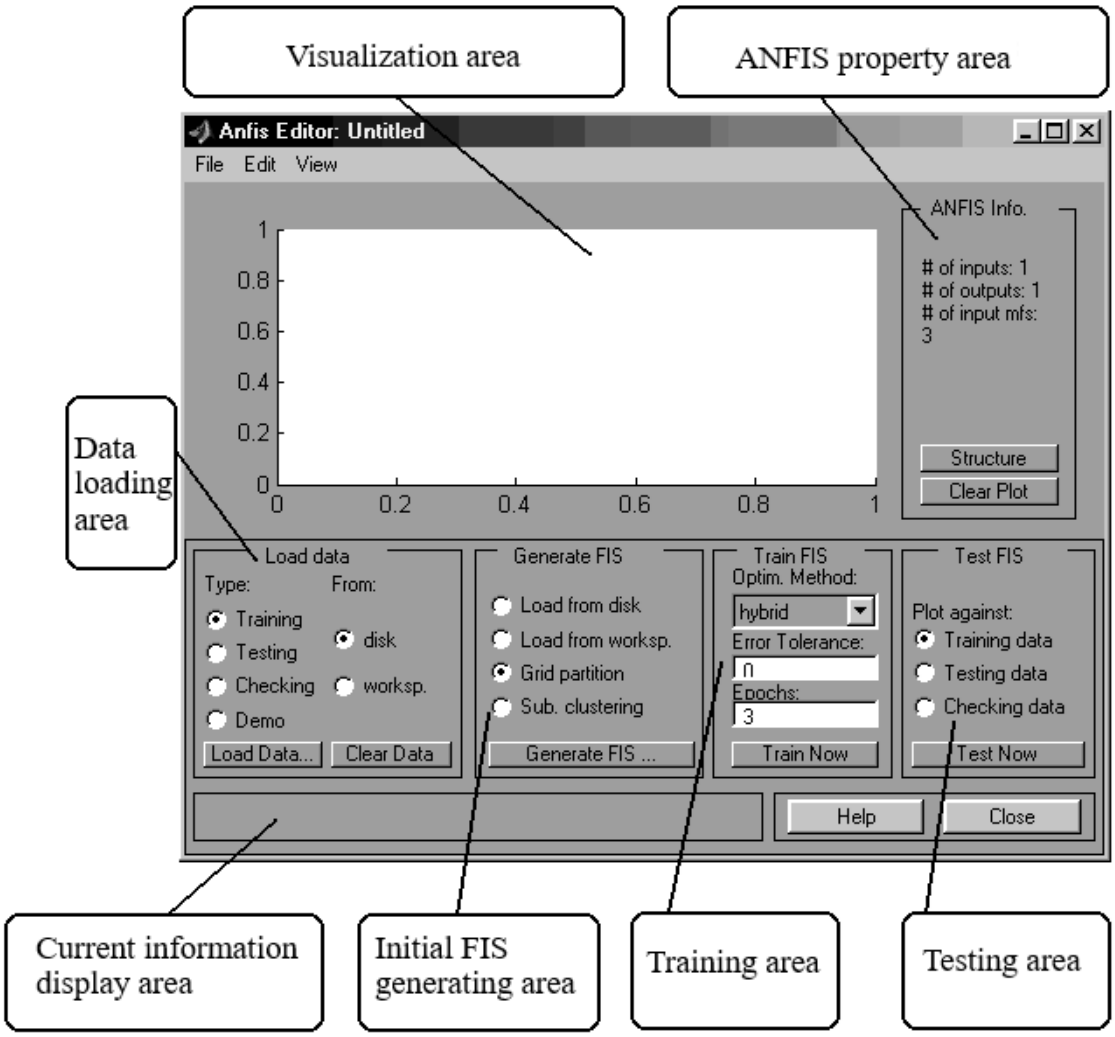

Figure 15.2. The main window of the *ANFIS* Editor

ANFIS-редактор содержит 3 верхних меню – *File*, *Edit* и *View*, область визуализации, область свойств *ANFIS*, область загрузки данных, область генерирования исходной системы нечеткого логического вывода, область обучения, область тестирования, область вывода текущей информации, а также кнопки *Help* и *Close*, которые позволяют вызвать окно справки и закрыть *ANFIS*-редактор, соответственно.

The *File* and *View* Menu are the same for all *GUI* modules used with fuzzy inference systems.

Edit – Menu. The general view of the menu is shown in Figure 15.3.

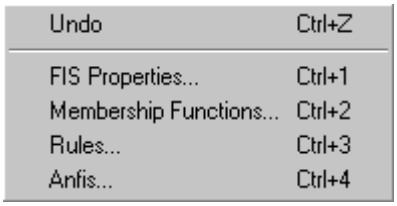

Figure 15.3. *Edit* Menu

The *Undo* command cancels a previous action. It is also performed by pressing <*Ctrl*+*Z*>.

The *FIS Properties…* command opens the FIS editor. This command can also be executed by pressing <*Ctrl*+1>.

The *Membership Functions…* command opens the membership function editor. This command can also be executed by pressing <*Ctrl*+2>.

The *Rules…* command opens the knowledge base editor. This command can also be executed by pressing <*Ctrl*+3>.

The *Anfis…* command opens *ANFIS*-editor. This command can also be executed by pressing <*Ctrl*+3>. Note that this command, launched from the *ANFIS* editor, does not lead to any action, since this editor is already open. However, the *Anfis…* command is added to the *Edit* menu of other *GUI* modules used with fuzzy inference systems, which allows you to open the ANFIS editor from these modules.

Visualization area. This area displays two types of information:

– when training the system – the training curve is in the form of the graph which displays the dependence of the training error on the ordinal number of the iteration graph;

– when loading data and testing the system – experimental data and simulation results.

Experimental data and simulation results are displayed as a set of points in two-dimensional space. In this case, the serial number of the data line in the sample (training, testing or control) is plotted along the abscissa, and the value of the output variable for this line of the sample is plotted along the ordinate. The following markers are used:

– blue dot (.) is a testing sample;

– blue circle (*o*) is a training sample;

– blue plus (+) is a control sample;

– red asterisk (*) is the simulation results.

ANFIS property area. The *ANFIS* properties area (*ANFIS info*) displays information about the number of input and output variables, the number of membership functions for each input variable, and the number of rows in the samples. There are two buttons in this area*: Structure* и *Clear Plot*.

Pressing the *Structure* button opens a new graphical window in which the fuzzy inference system is represented as a neuro-fuzzy network. As an illustration, a neuro-fuzzy network containing four input variables and one output variable is shown (Figure 15.4). In this system, three linguistic terms are used to evaluate each of the input variables and four terms are used to evaluate the output one.

Pressing the *Clear Plot* button clears the visualization area.

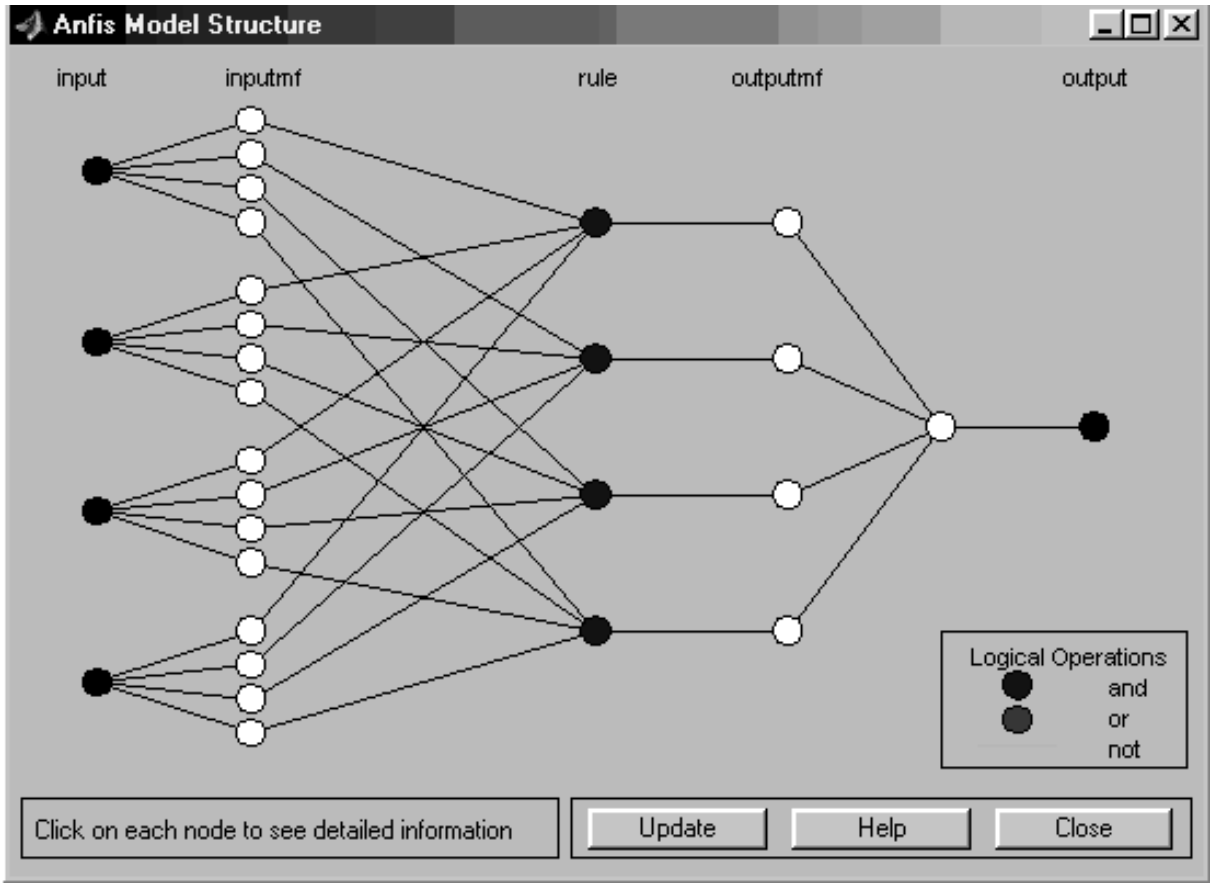

Figure 15.4. An example of the structure of a neuro-fuzzy network

Data loading area. The Load data area contains:

– data type selection menu (*Type*) containing alternatives (*Traning* – training sample; *Testing* – testing sample; *Checking* – control sample; *Demo* – demo example);

– data source selection menu (*From*) containing alternatives (*disk* – disk; *worksp* – MATLAB workspace);

– *Load Data…* button, by pressing which a file selection dialog box appears if the data is loaded from the disk, or a window for entering the selection identifier if the data is loaded from the workspace;

– *Clear Data* button.

During one session of the *ANFIS* editor, you can load data of one format, i.e. the number of input variables in the samples must be the same.

The area of the fuzzy logical inference initial system generation. In the area of generation (*Generate FIS*) there is a menu for choosing a method for

creating an initial fuzzy logical inference system. The menu contains the following alternatives:

– *Load from disk* – loading system from disk;

– *Load from worksp* – loading system from MATLAB workspace;

– *Grid partition* – system generation by the grid method (without clustering);

– *Sub*. *Clustering* – system generation by subclustering method.

The area also contains the *Generate* button, by pressing which the initial fuzzy inference system is generated.

When you select *Load from disk*, the standard file open dialog box appears.

When you select *Load from worksp* the standard fuzzy logical inference system identifier entry dialog box appears.

When you select *Grid partition* the grid method parameter input window (Figure 15.5) appears, in which you need to specify the number of terms for each input variable and the type of membership functions for the input and output variables.

When you select *Sub. clustering* a window for entering the following parameters of the subclustering method appears (Figure 15.6):

– *Range of influence* – levels of input variables influence;

– *Squash factor* – suppression coefficient;

– *Accept ratio* – a coefficient that sets how many times the potential of a given point must be higher than the potential of the center of the first cluster in order for the considered point to be the center of one of the clusters;

– *Reject ratio* – a coefficient that sets how many times the potential of a given point must be lower than the potential of the center of the first cluster in order for the point under consideration to be excluded from the possible centers of clusters.

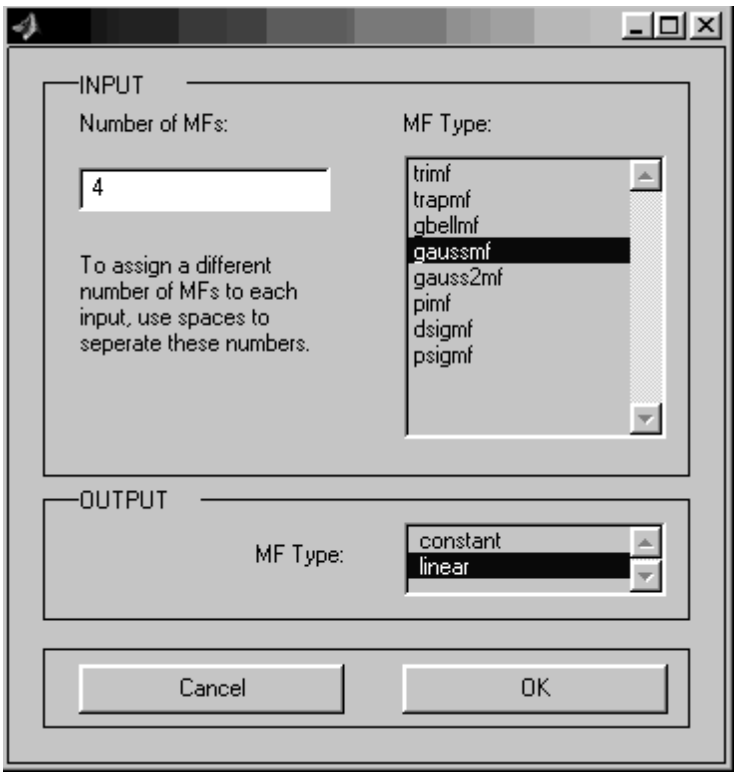

Figure 15.5. Parameter input window for the grid method

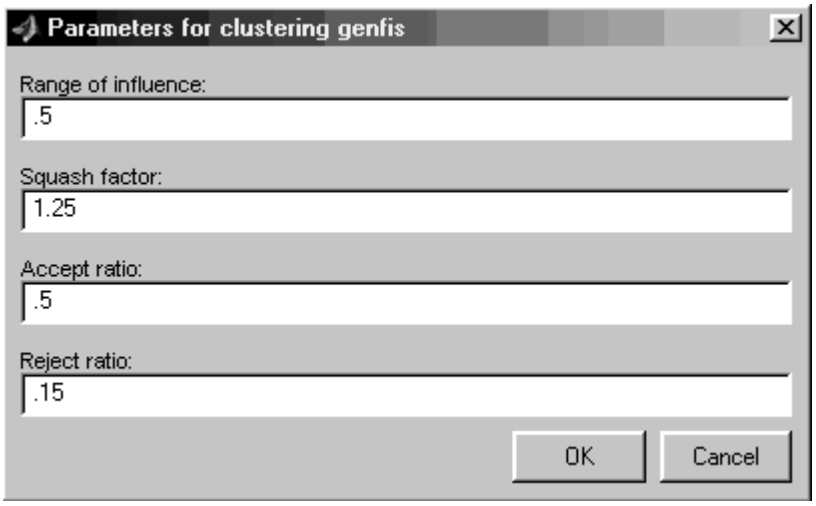

Figure 15.6. Parameter input window for subclustering method

The training area. Area of study. The training area (*Train FIS*) contains the menu for selecting the optimization method (*Optim. method*), the field for setting the required training accuracy (*Error tolerance*), the field for setting the number of training iterations (*Epochs*) and the *Train Now* button. Pressing this button starts the training mode. Intermediate learning outcomes are displayed in

the visualization area and in the MATLAB workspace. The *ANFIS* editor implements two training methods:

– *backpropa* – error backpropagation method based on the ideas of the fastest descent method;

– *hybrid* – a hybrid method that combines the backpropagation method with the least squares method.

Testing area. The testing area (*Test FIS*) contains the sample selection menu and the *Test Now* button, by pressing which the fuzzy system is tested and the results are displayed in the visualization area.

Current information display area. This area displays the most significant current information, for example, messages about the completion of operations, the value of the training or testing error, etc.

15.2.2. Synthesis of a neuro-fuzzy network in the MATLAB environment

Description of the problem: there are initial data on changes in the RTS financial index for the period from 03/01/2021 to 04/30/2021. It is required to build a neuro-fuzzy network and predict the value of the index for 1/05/2021.

The general sequence of the hybrid network model development process can be represented as follows.

1. Preparing a file with training data (Table 15.1). It is advisable to use the spreadsheet editor MS Excel. The training sample must be saved in an external file with **.dat* extension. The prediction algorithm implies that each subsequent value is calculated based on several previous ones.

Table 15.1 – Data set for neuro-fuzzy network training

First input variable	Second input variable	Third input variable	Output variable
688,72	686,21	667,27	669,26
686,21	667,27	669,26	673,25
667,27	669,26	673,25	688,68
669,26	673,25	688,68	680,86
673,25	688,68	680,86	671,33

688,68	680,86	671,33	669,55
680,86	671,33	669,55	676,20
671,33	669,55	676,20	680,57
669,55	676,20	680,57	698,70
676,20	680,57	698,70	708,59
680,57	698,70	708,59	721,81
698,70	708,59	721,81	712,88
708,59	721,81	712,88	713,15
721,81	712,88	713,14	705,16
712,88	713,14	705,16	706,71
713,14	705,16	706,71	716,55
705,16	706,71	716,55	721,35
706,71	716,55	721,35	736,28
688,72	686,21	667,27	669,26
686,21	667,27	669,26	673,25
667,27	669,26	673,25	688,68
669,26	673,25	688,68	680,86
673,25	688,68	680,86	671,33
688,68	680,86	671,33	669,55
680,86	671,33	669,55	676,20
671,33	669,55	676,20	680,57

2. Open the *ANFIS* editor. Upload the training data file. By pressing *Load Data* button, a file selection dialog box appears if the data is loaded from disk, or a window for entering the selection identifier if data is loaded from the workspace.

The ANFIS editor with training data loaded is shown in Figure 15.7.

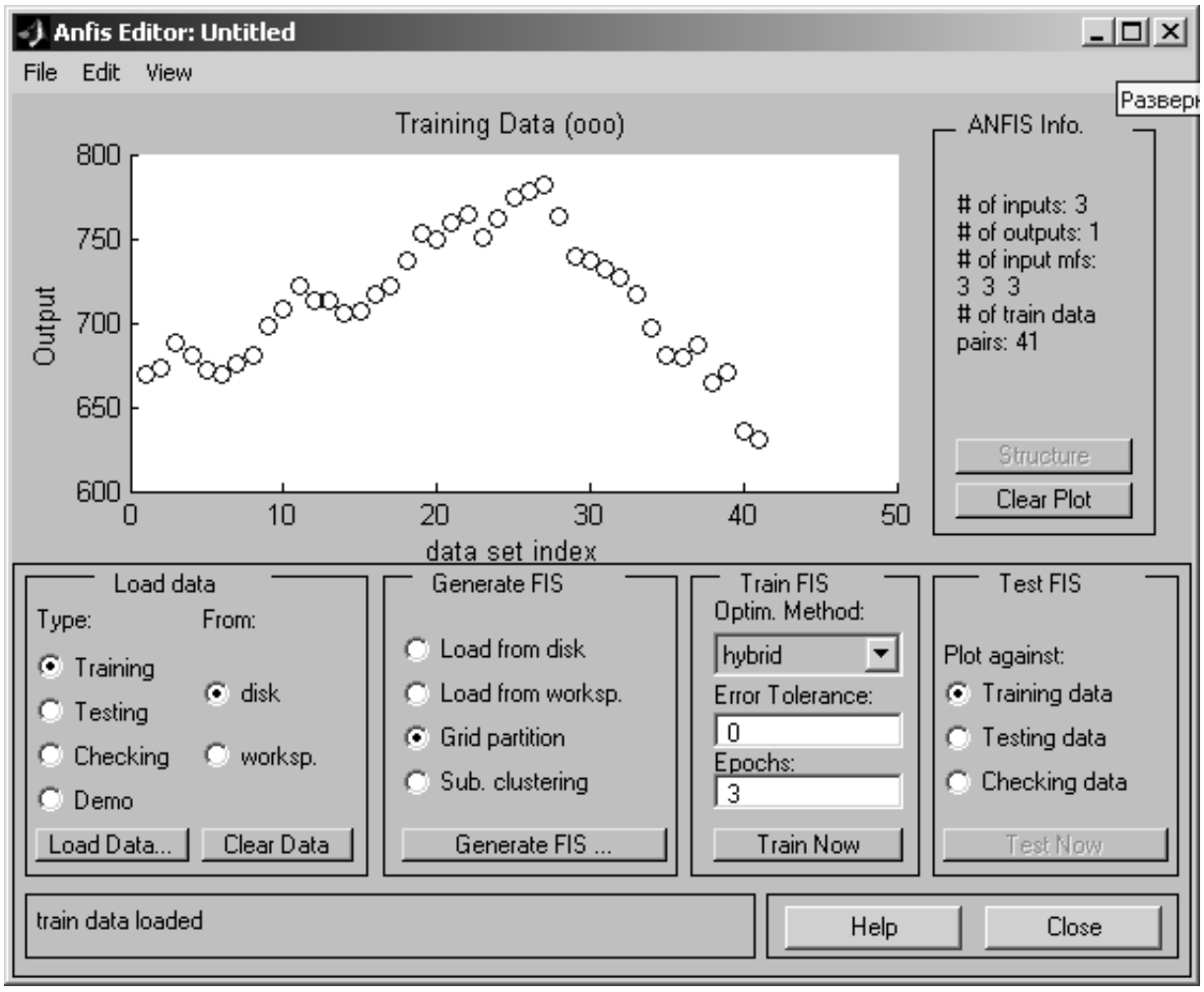

Figure 15.7. ANFIS editor GUI after loading training data

3. After preparing and loading the training data, you can generate the structure of the Sugeno-type *FIS*, which is a hybrid network model in the MATLAB system. For this purpose, use the *Generate FIS* button at the bottom of the editor's working window. In this case, the first two options refer to the previously created structure of the hybrid network, and the last two options refer to the form of splitting the input variables of the model.

Before generating the structure of a fuzzy inference system of the Sugeno type, after calling the properties dialog box, we set three linguistic terms for each of the input variables, and choose triangular functions as the type of their membership functions.

After clicking the *Generate FIS* button, a dialog box is called showing the number and type of membership functions for individual terms of the input variables and the output variable (Figure 15.8).

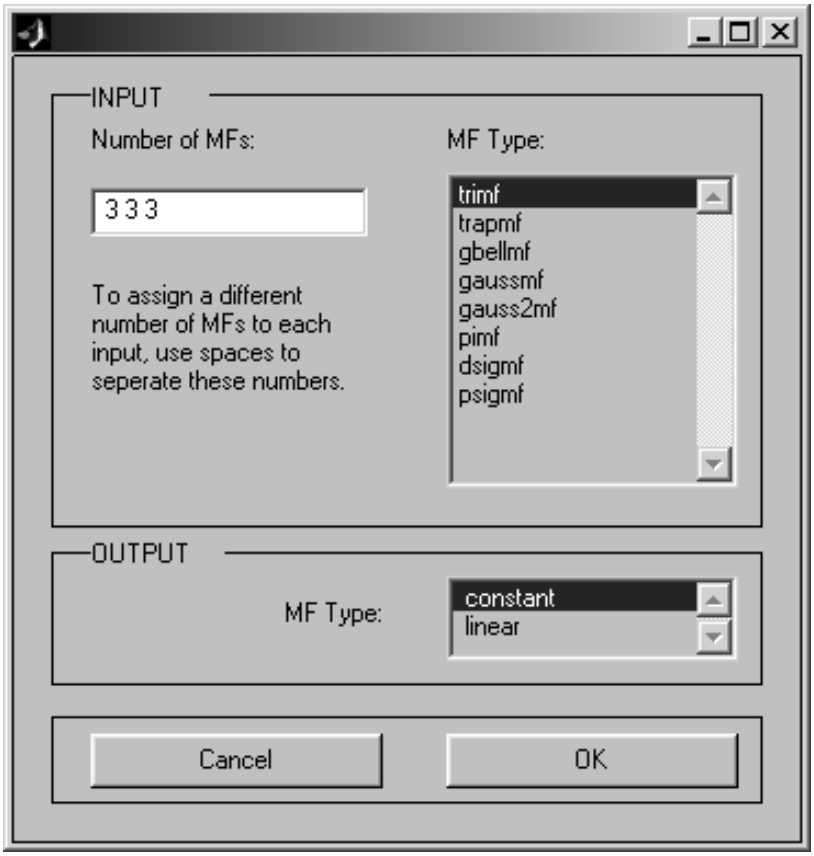

Figure 15.8. Dialog box for specifying the number and type of membership functions

4. After generating the hybrid network structure, you can visualize its structure by clicking the *Structure* button in the right part of the graphics window (Figure 15.9).

For the example under consideration, the fuzzy inference system contains three input variables with three terms each, 27 fuzzy production rules, one output variable with 27 terms.

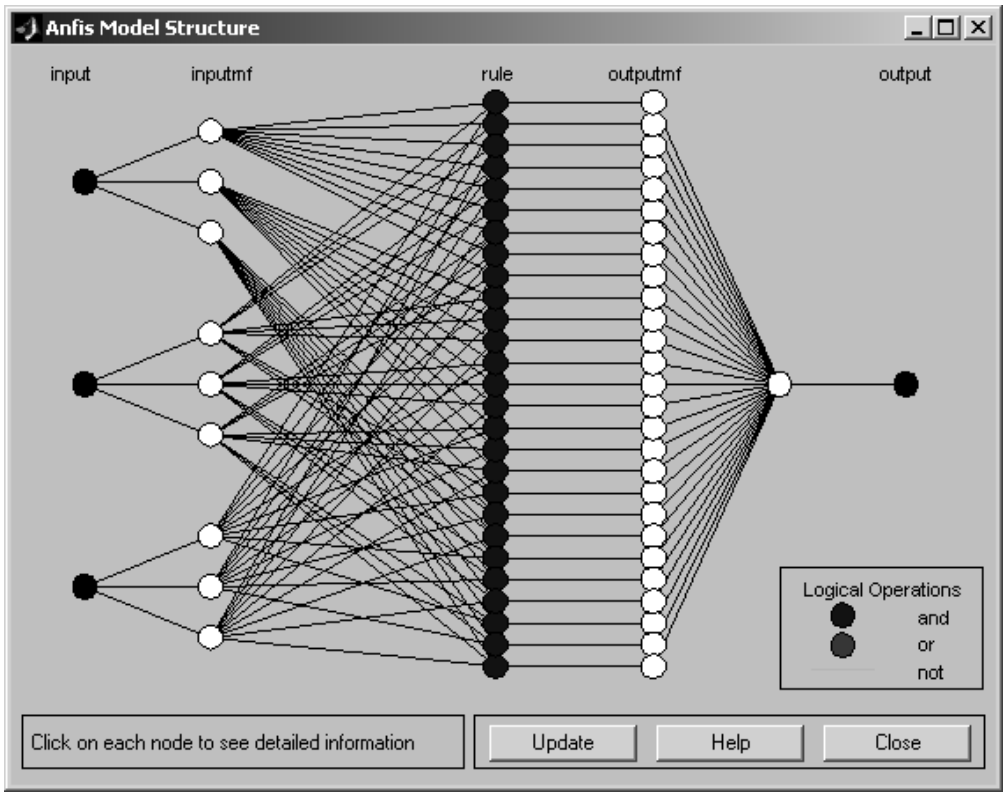

Figure 15.9. Structure of the generated fuzzy inference system

5. Before training a hybrid network, you must set the training parameters, for which you should use the following group of options in the lower right part of the working window:

1. Choose a hybrid network training method – backpropagation (*backpropo*) or hybrid (*hybrid*), which is a combination of the least squares method and the reverse gradient descending method.

2. Set the training error level (*Error Tolerance*) - the default value is 0 (it is not recommended to change).

3. Set the number of training cycles (*Epochs*) - the default value is 3 (it is recommended to increase for the example in question, set its value to 40).

To train the network, click the *Train now* button. At the same time, the progress of the training process is illustrated in the visualization window in the form of a graph that displays the dependence of the error on the number of training cycles (Figure 15.10).

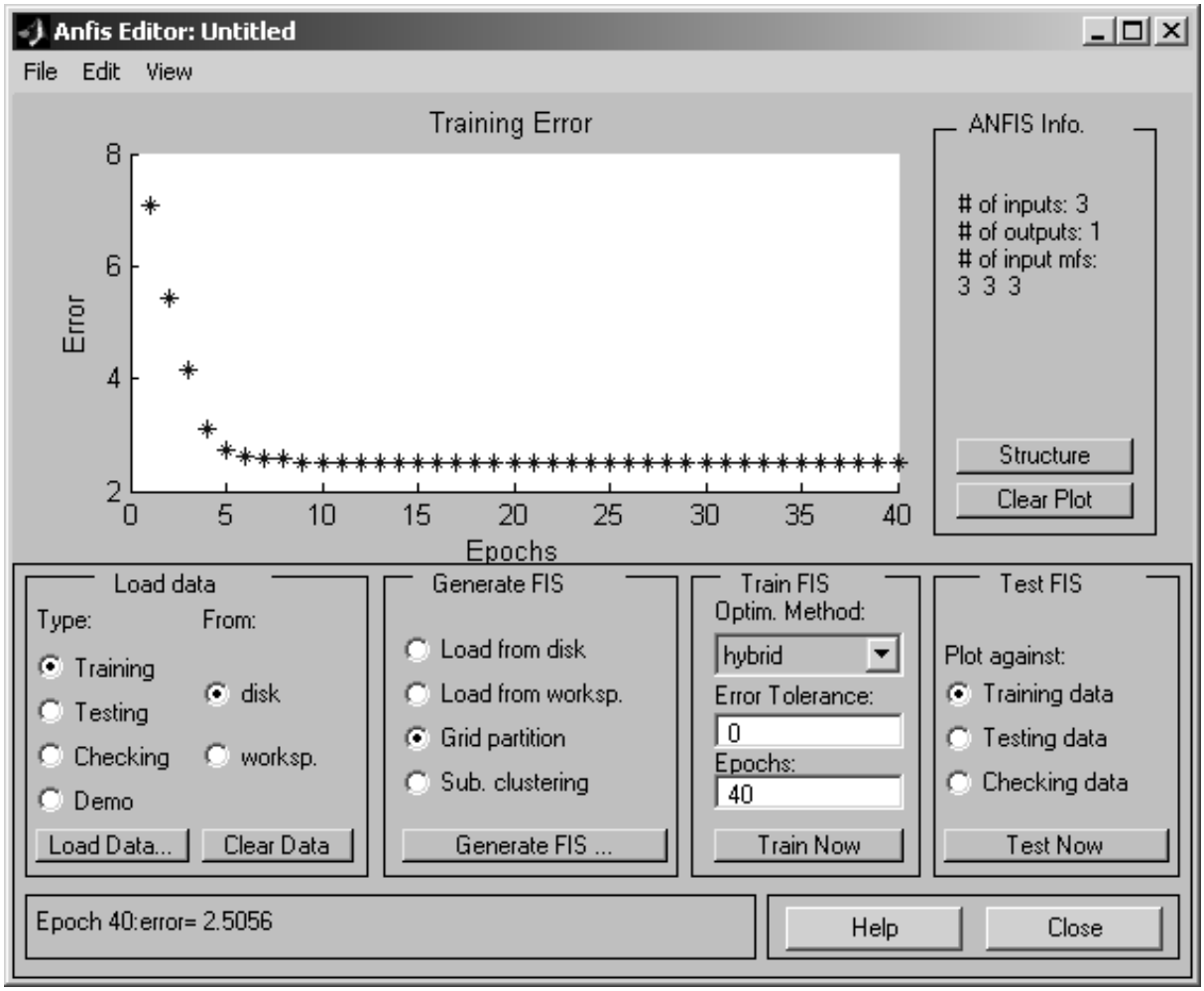

Figure 15.10. Graph of the dependence of the training error on the number of training cycles

Further configuration of the parameters of the constructed and trained hybrid network can be performed using standard graphical tools of the package *Fuzzy Logic Toolbox* (Figure 15.11 – 15.14). To do this, it is recommended to save the created fuzzy inference system in an external file with **.fis* extension. After that, this file should be loaded into the editor of fuzzy inference systems *FIS*.

6. Let's check the adequacy of the constructed fuzzy hybrid network model. To do this, you can predict the dollar exchange rate for a certain day. To solve this problem, you need to use the *evalfis* function.

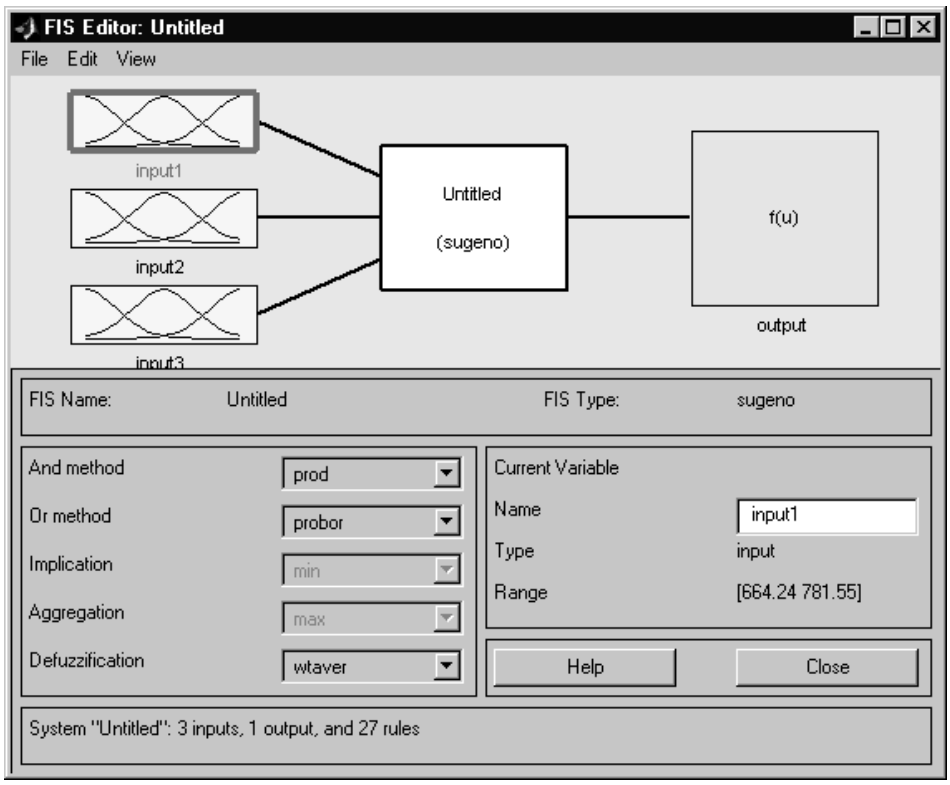

Figure 15.11. FIS editor GUI for generated fuzzy inference system

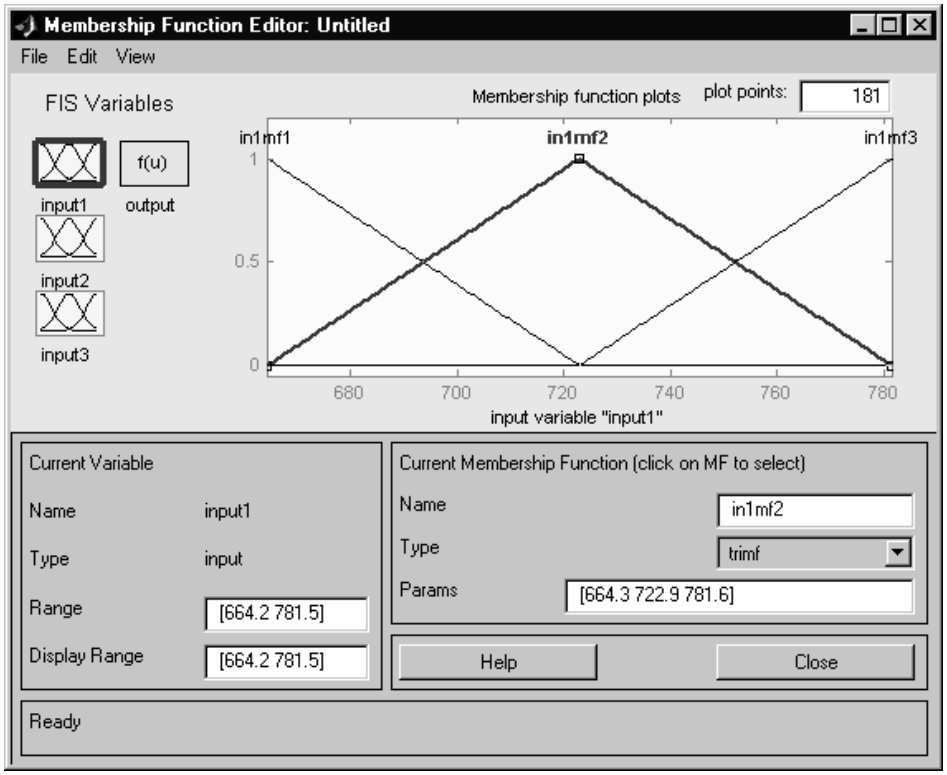

Figure 15.12. Graphical interface of the constructed fuzzy inference system's membership functions editor

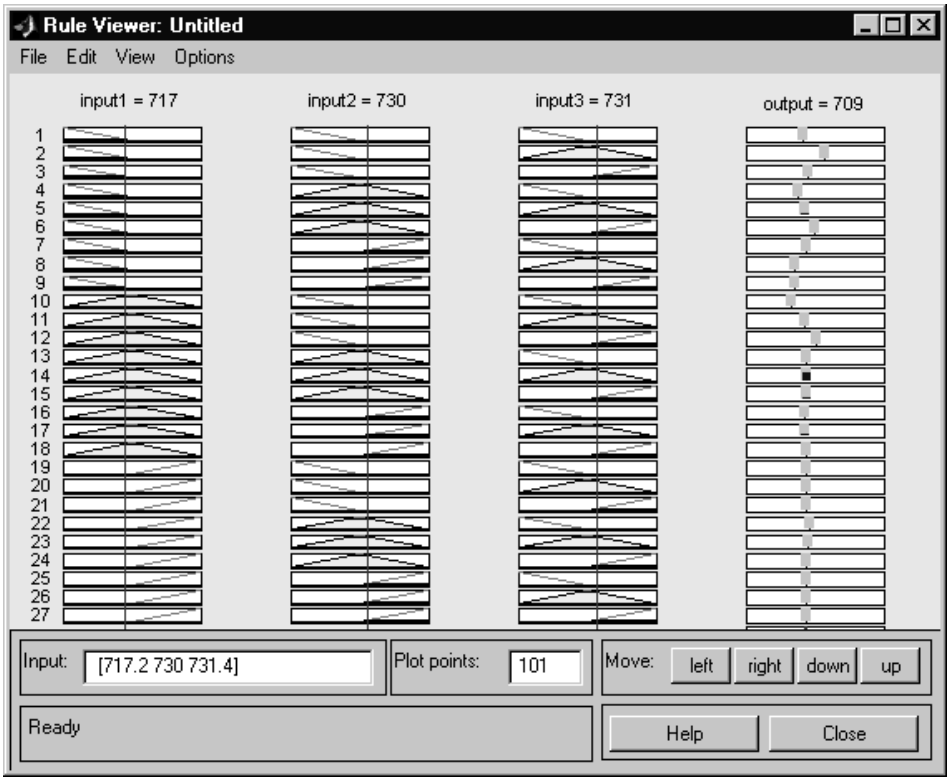

Figure 15.13. Graphical interface for viewing the rules of the generated fuzzy inference system

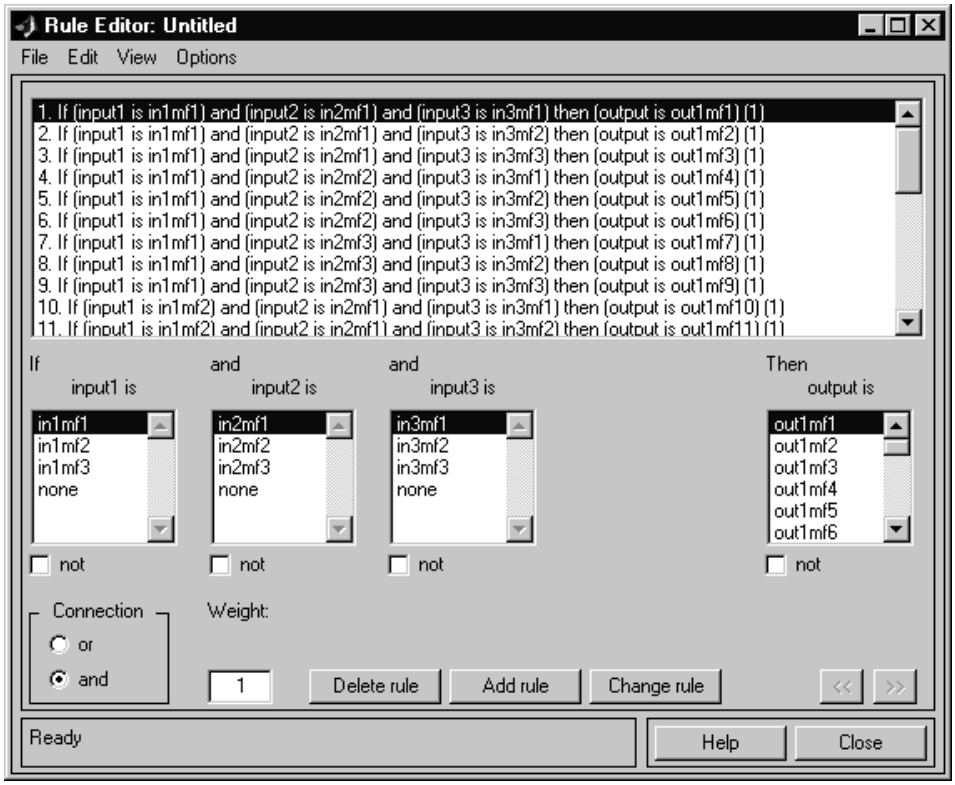

Figure 15.14. Fragment of fuzzy rule base

15.3. Individual tasks

1. Formulate a problem from the field of computer technology, for the solution of which the use of a hybrid neuro-fuzzy network would be justified.

2. Form a sample for training a hybrid neural network.

3. Generate and visualize hybrid neural network structure in MATLAB system.

4. Train a hybrid neural network. Set and justify the parameters of her training.

5. Build a fuzzy inference system for the resulting hybrid neural network.

6. Check the adequacy of the constructed fuzzy hybrid network model.

7. Design the lab’s report.

Laboratory work 16

PROGNOSIS PROBLEM SOLVING USING FUZZY NEURAL NETWORKS

The purpose of the laboratory work is to obtain and consolidate knowledge, to form practical skills in building neural networks and hybrid neural networks for prognosis in the MATLAB package.

16.1. Summary of theory

Neural networks and fuzzy logic are universal approximators of complex (nonlinear) functional dependencies in many intellectual problems of cybernetics: prognosis, diagnostics, pattern recognition, etc.

The advantage of fuzzy logic is the ability to use expert knowledge about the structure of an object in the form of linguistic statements: if <inputs>, then <output>. However, the apparatus of fuzzy logic does not contain learning mechanisms. Therefore, the results obtained with its help strongly depend on the type of membership functions that formalize fuzzy terms. In addition, the expert needs to define all the rules. Such an algorithm is largely static. The need to change it will entail a rather laborious expert procedure.

The main feature of neural networks is their ability to learn. To train a neural network, no a priori information about the structure of the desired functional dependence is required. All that is needed is a training sample in the form of <inputs – output> pairs.

Instead of determining all the membership functions of linguistic variables and rules, it is necessary to create a training sample of sufficient size. One of the criteria for "sufficiency" is the following: the number of training pairs <inputs – output> must exceed the number of configurable network parameters.

The advantage of this method is the ability to self-configure network parameters during operation.

Combining fuzzy logic with neural networks gives a fundamentally new quality. The resulting neuro-fuzzy network has two important human (intellectual) properties:

– linguistics, i.e. using natural language knowledge;

– training in real time.

In accordance with the analysis carried out in a number of works, for the class of tasks to which neuro-fuzzy networks are applicable, neural networks require significantly more iterations during training. Since any iteration takes a certain computer time, then the total learning time of the neural network is more.

There are two main types of neuro-fuzzy networks (NFN): Sugeno-Takagi-Kanga and Wang-Mendel. The structure of the Wang-Mendel network is shown in Figure 16.1.

Layer I consists of neurons-fasifiers and performs separate fuzzification of each variable by membership functions (MF). The outputs of this layer are calculated using the generalized Gaussian function:

$$FP[i \cdot M + j] = \exp\left[\left(-\frac{(x - C[i \cdot M + j])^2}{2D[i \cdot M + j]^2}\right)^B\right], \qquad (16.1)$$

presented in rational form:

$$FP[i \cdot M + j] = \frac{1}{1 + \left(\frac{X[i] - C[i \cdot M + j]}{D[i \cdot M + j]}\right)^{2B[i \cdot M + j]}}, \qquad (16.2)$$

where $i = \overline{0, N-1}$, $j = \overline{0, M-1}$; M is the number of membership functions for each variable; N is the number of input variables; $C[iM + j]$ (bias); $D[iM + j]$

(scale); $B[iM + j]$ (form) are non-linear network parameters describing the Gaussian function (to be adjusted during training).

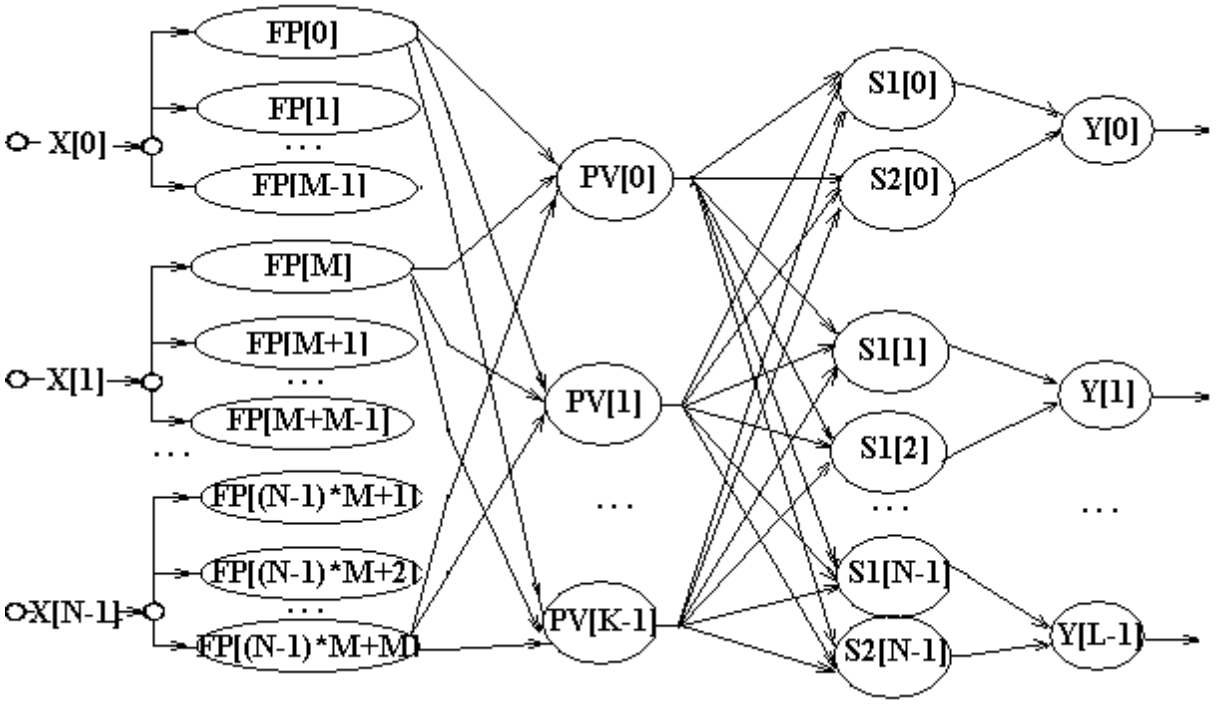

Figure 16.1. The structure of the NFN Wang-Mendel

Layer II consists of neurons–rules and determines the activation levels of inference rules (*PV*). The outputs of this layer are calculated by the formula:

$$PV[i] = \prod_{j=0}^{N-1} FP[j], \tag{16.3}$$

where $i = \overline{0,K-1}$, and $K = M^N$.

Layer III consists of two types of adder neurons, one of which (S_1) aggregates inference rules, and the second one (S_2) aggregates weighted inference rules. The outputs of neurons are calculated by the formulas:

$$S_1[j] = \sum_{i=0}^{K-1} W[j,i] \cdot PV[i], \tag{16.4}$$

$$S_2[j] = \sum_{i=0}^{K-1} PV[i], \qquad (16.5)$$

where $j = \overline{0, L-1}$, L is the number of network outputs; $W[j, i]$ are linear network parameters that determine the weight coefficients of the applied rule (to be adjusted during training).

Layer IV contains normalizer neurons and normalizes the output variables of the Y network. Its output is calculated by the formula:

$$Y[i] = S_1[i] / S_2[i], \qquad (16.6)$$

where $i = \overline{0, K-1}$.

This network contains two parametric layers - the first and third. In the first layer, each neuron stores three non-linear network parameters C, D and B, which define the Gaussian membership function. The third layer stores the linear parameters of the network W, which determine the weight coefficients of the rules and the values of the output variables.

16.2. Building Neural Networks and Hybrid Neural Networks for prognosis in MATLAB

One of the main features of neural networks (NN) is the ability to extrapolate. In the process of training on the available statistics (for example, about a certain time series), the NN is able to find dependencies (including those hidden even for an experienced expert) between data and situations, which, in the subsequent trained NN, make it possible to predict the behavior of some functional.

In particular, the NN is quite widely used to predict the exchange rate for a certain period. For example, to forecast the exchange rate on the fifth banking

day based on the available data on the previous four or a monthly prognosis based on data for six months.

For these purposes, it is possible to use both conventional NNs of direct propagation and fuzzy neural networks.

16.2.1. Neural network building

Let's build a neural network that, based on the data on the exchange rate for four banking days, predicts the exchange rate on the fifth day (extrapolates). Data from 01/01/2021 to 03/10/2021 on the US dollar exchange rate are used. Training data – from 01/01/2021 to 03/01/2021.

Create a matrix of training data:

```
» obych=[31.5673  31.6053  31.6117  31.6113  31.6123
         31.6053  31.6117  31.6113  31.6123  31.6268
         31.6117  31.6113  31.6123  31.6268  31.6302
         31.6113  31.6123  31.6268  31.6302  31.6409
         ..........................................................
         31.8382  31.8423  31.8445  31.8424  31.844];
```

Create a matrix of input values:

```
» vxod=obych(:,1:4);
```

Create a matrix of expected output values:

```
» vixod=obych(:,5);
```

Create a neural network for prognosis:

```
» prognoz=newff(minmax(vxod'),[15 1],{'logsig'
'purelin'});
```

Train the neural network:

```
» prognoz=train(prognoz,vxod',vixod');
```

As a result of training the neural network, we get the window shown in Figure 16.2.

Create a test sample to check training outcomes:

```
test =
    31.8400  31.8400  31.8424  31.8547  31.8584
    31.8400  31.8424  31.8547  31.8584  31.8596
    31.8424  31.8547  31.8584  31.8596  31.8578
    31.8547  31.8584  31.8596  31.8578  31.8600
» test_vxod=test(:,1:4);
» test_vixod=test(:,5);
```

Let's simulate the trained network on test data:

```
» sim(prognoz,test_vxod)
ans =
    31.8455  31.8467  31.8467  31.8463
» test_vixod
test_vixod=
    31.8584
    31.8596
    31.8578
    31.8600
```

The simulation results and the true values of the exchange rate differ sufficiently.

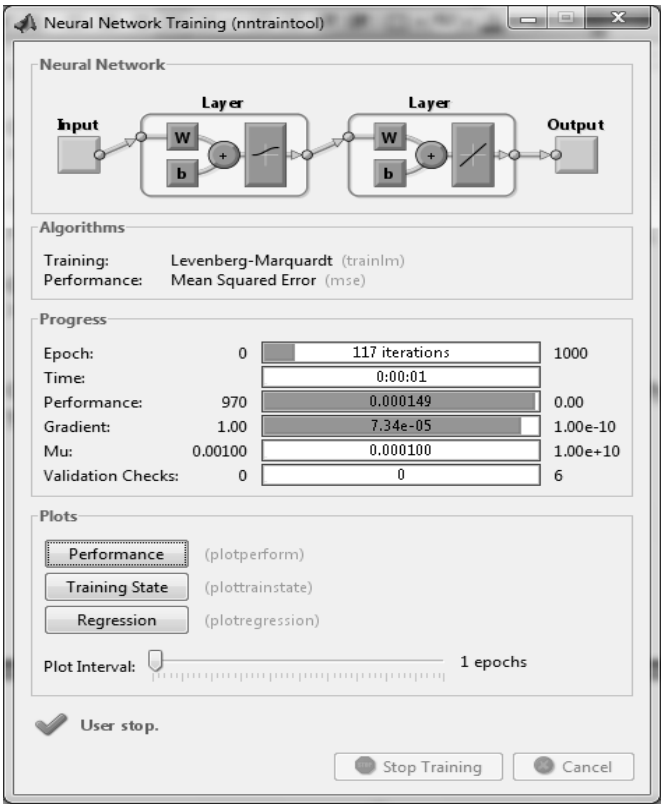

Figure 16.2. NN's training outcomes

16.2.2. Hybrid neural network building

Let's build a hybrid neural network that, based on the data on the exchange rate for four banking days, predicts the exchange rate on the fifth day (extrapolates). Data from 01/01/2021 to 03/10/2021 on the US dollar exchange rate are used. Training data – from 01/01/2021 to 03/01/2021, testing data – from 01/01/2021 to 01/09/2021, checking data – to 01/10/2021.

Let's create data files *.dat*: *training.gat*, *testing.dat*, *checking.dat*. They will contain the data needed to train and check the neural network. The data in them is a matrix. Each matrix has 5 columns - 4 banking days (input) and 1 day (output). The first file has 40 lines, the second file has 4 lines, and the third file has 1 line.

File *Training.dat*

```
31.5673  31.6053  31.6117  31.6113  31.6123
31.6053  31.6117  31.6113  31.6123  31.6268
31.6117  31.6113  31.6123  31.6268  31.6302
31.6113  31.6123  31.6268  31.6302  31.6409
.....................................................................
31.8382  31.8423  31.8445  31.8424  31.8424
```

File *Testing.dat*

```
31.8422  31.8445  31.8424  31.8547  31.8584
31.8423  31.8424  31.8547  31.8584  31.8596
31.8424  31.8547  31.8584  31.8596  31.8578
31.8547  31.8584  31.8596  31.8578  31.8634
```

File *Checking.dat*

```
31.8584  31.8596  31.8578  31.8623  31.8597
```

To build fuzzy neural networks in MATLAB, we use *Anfis Editor* (Figure 16.3).

Launch the editor with the *anfisedit* command.

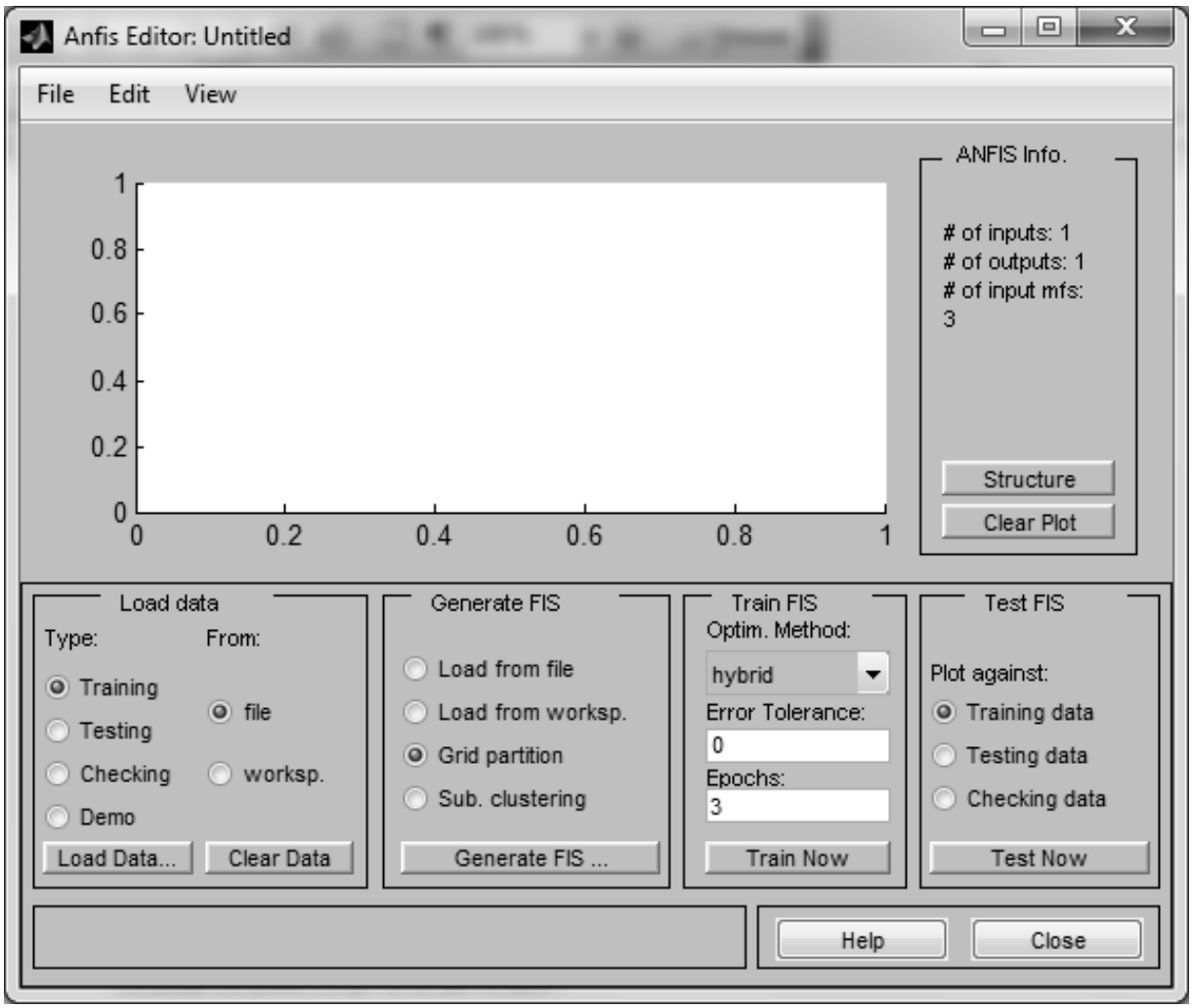

Figure 16.3. *Anfis editor* main window

In the *Load data* menu, select *Training* and *From disk*, press the *load data* button. In the window that appears, select the previously created *training.dat*.

In the *Load data* menu, select *Testing* and *From disk*, press the *load data* button. In the window that appears, select the previously created *testing.dat*.

In the *Load data* menu, select *Cheking* and *From disk*, press the *load data* button. In the window that appears, select the previously created *cheking.dat.*

Training and checking data is loaded (Figure 16.4).

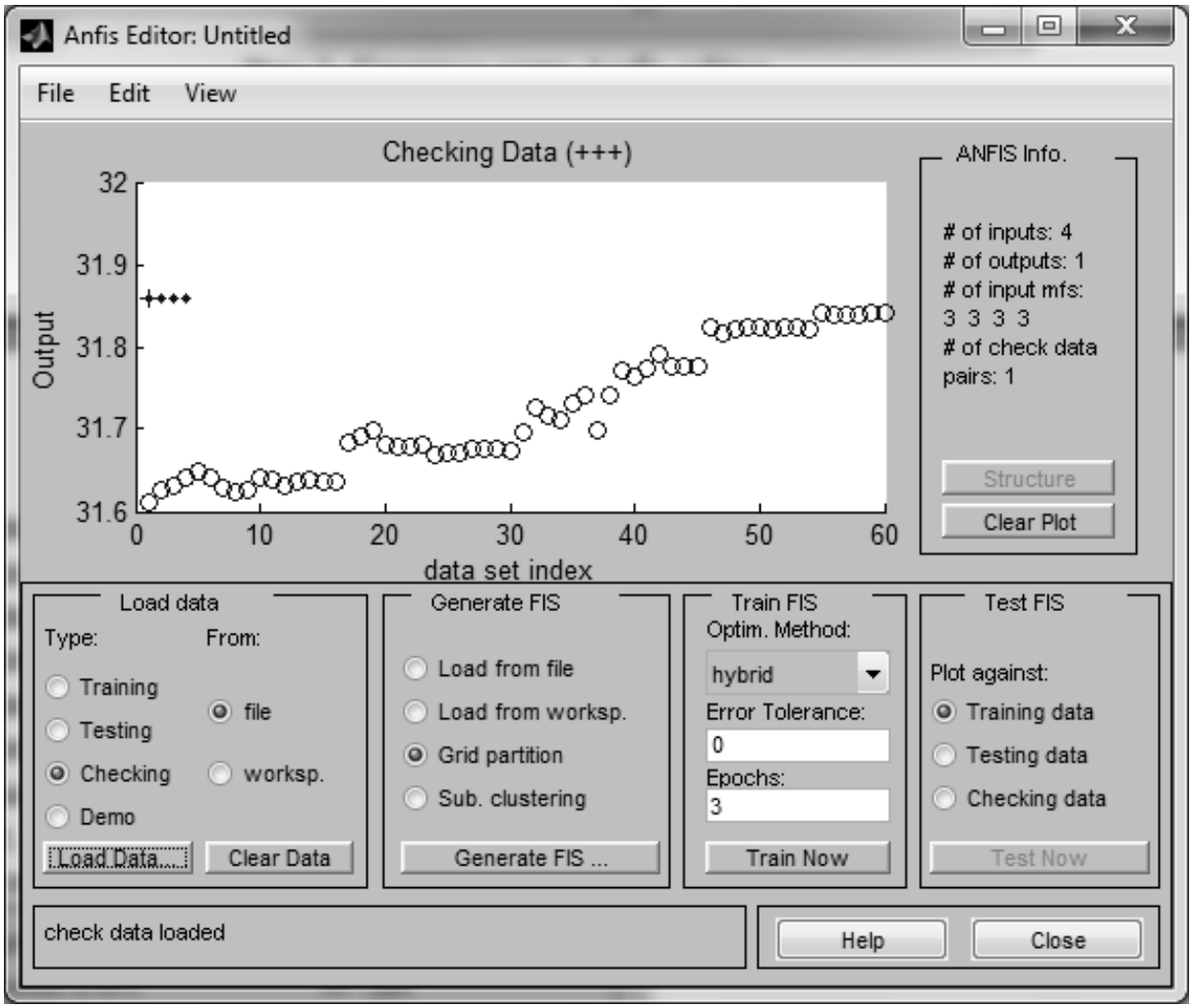

Figure 16.4. View of the main editor window after loading the training data

Further, having set the *Generate FIS* menu switch to the *Grid partition* position, you should press the *Generate FIS* button (Figure 16.5).

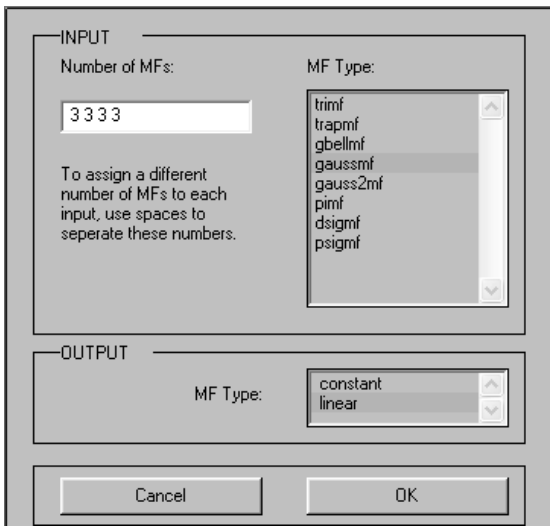

Figure 16.5. Parameters of the generated network

In this case, the model has 4 input variables, each of which corresponds to 3 terms of the *gaussmf* type. The output variable is given by a linear function.

By clicking on the *Structure* button of the *Anfis info* menu, you can see the structure of the network (Figure 16.6).

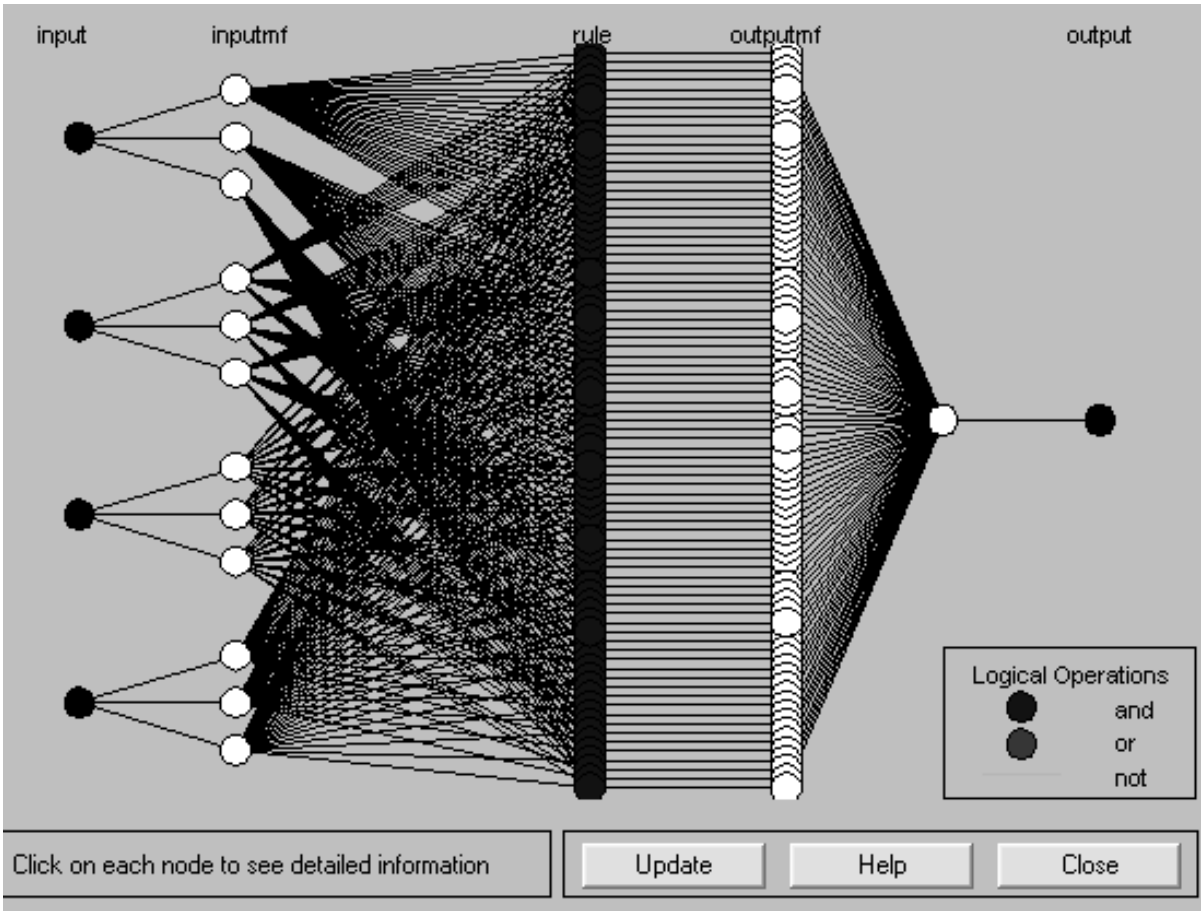

Figure 16.6. Structure of fuzzy neural network

The next step is to select a hybrid training method, the required training error is 0, and the number of training cycles is 10, as shown in Figure 16.7. Then click the *Train Now* button.

The learning error has settled at 0.00010501.

Let's test the network. To do this, in the *Test FIS* section of the main window, select the appropriate sample and click the *Nest Now* button. As a result, for the training data, the error was fixed at 0.00014247 (Figure 16.8), for the testing data, the error was fixed at 0.020429 (Figure 16.9), and for the test (control) data, the error was fixed at 0.002094 (Figure 16.10).

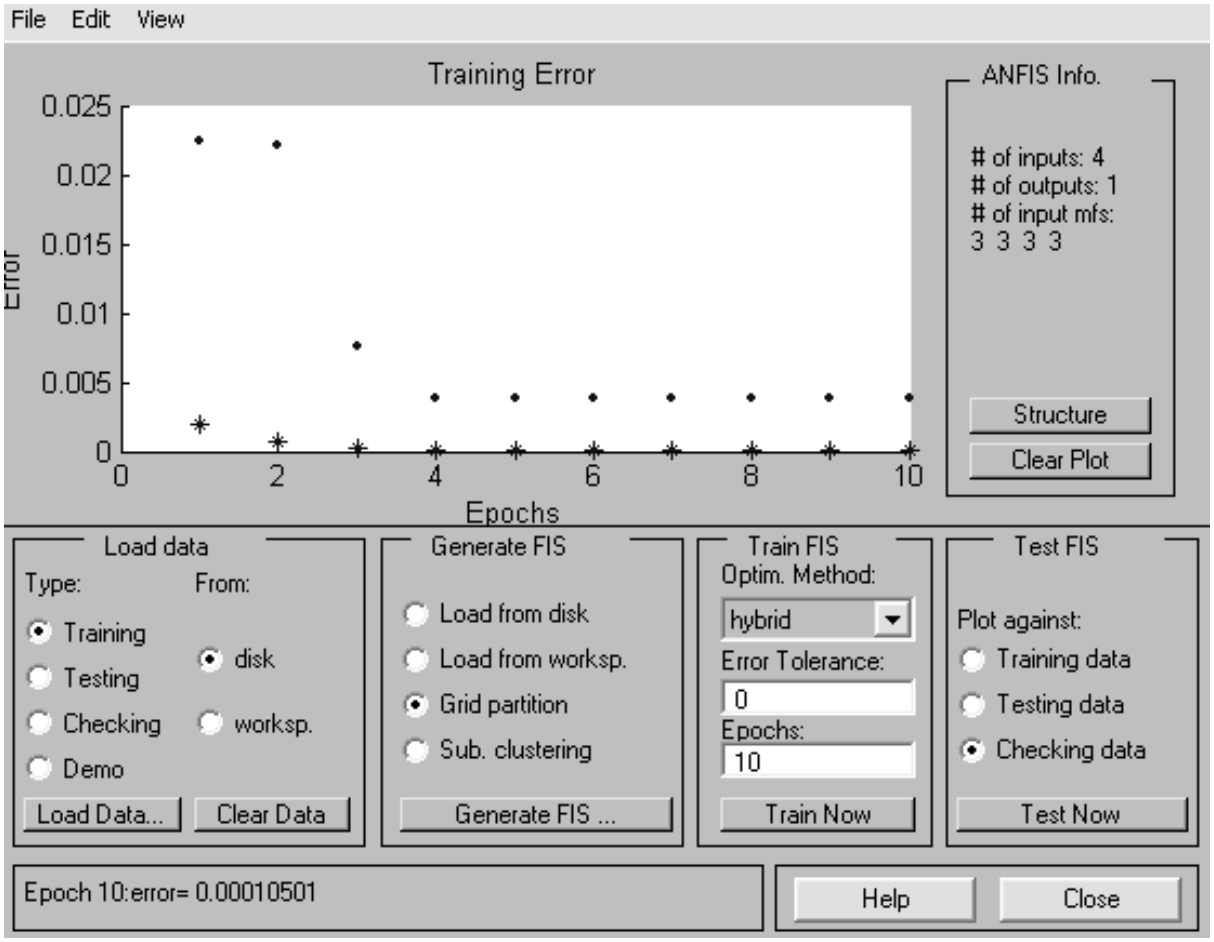

Figure 16.7. NFN teaching outcomes

Using the FIS Properties command of the Edit menu, it is possible to view the resulting fuzzy neural network as a fuzzy logical inference system.

Next, we export the results to the workspace *Export->To workspace*.

Let's use the *evalfis* command to accurately determine the value of the prognosis:

```
» out=evalfis([31.8584 31.8596 31.8578 31.86], anfis)
out= 31.8568
```

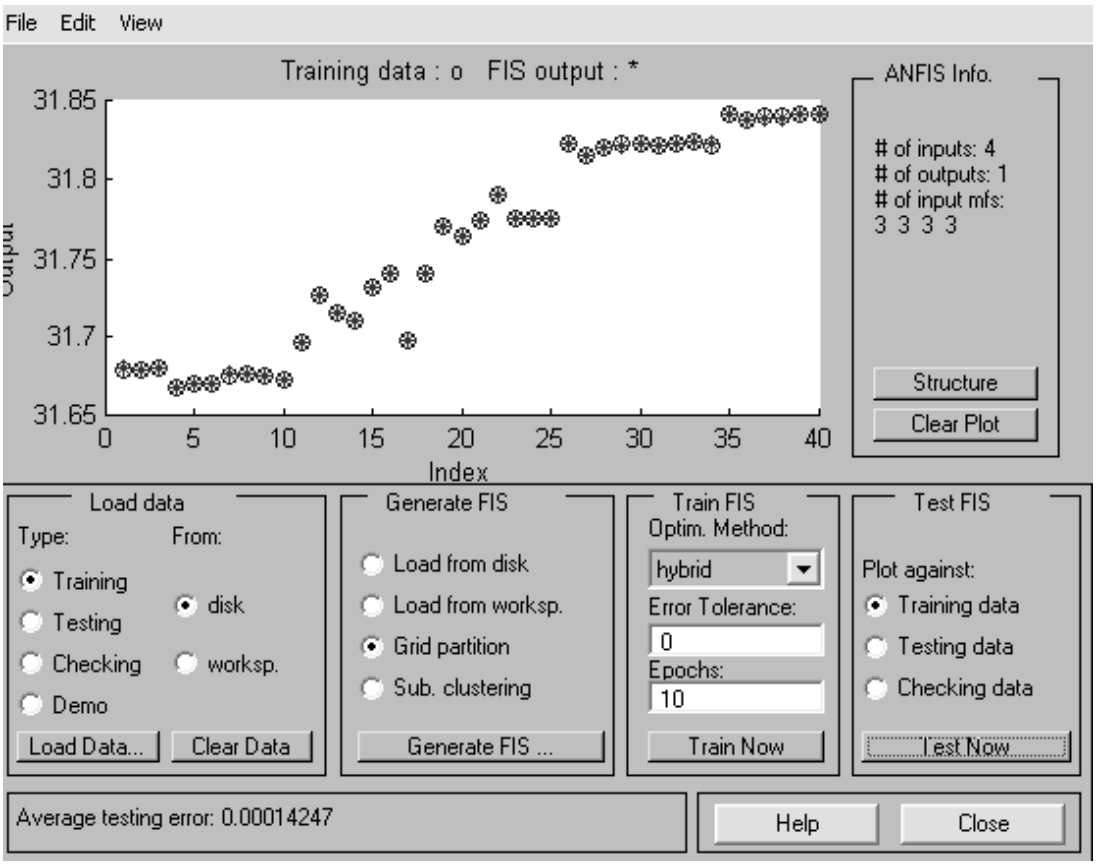

Figure 16.8. NFN simulation results for training data

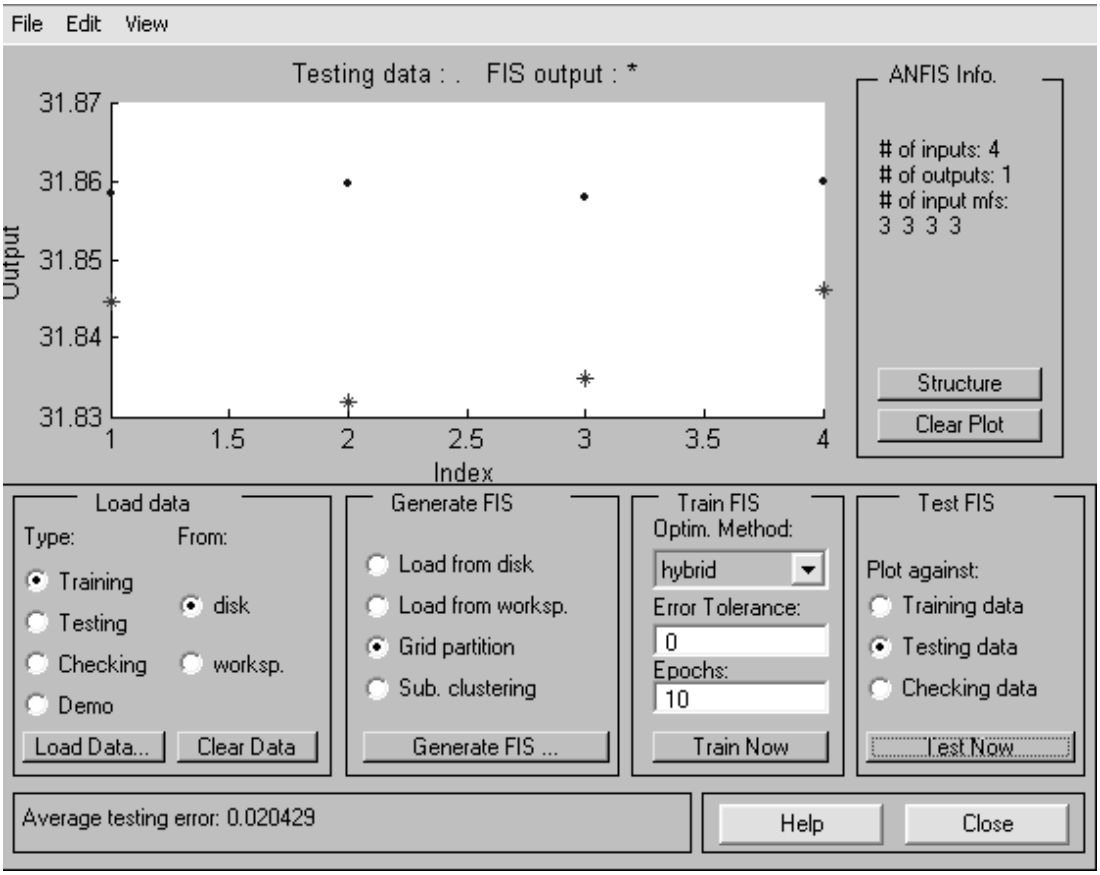

Figure 16.9. NFN simulation results for testing data

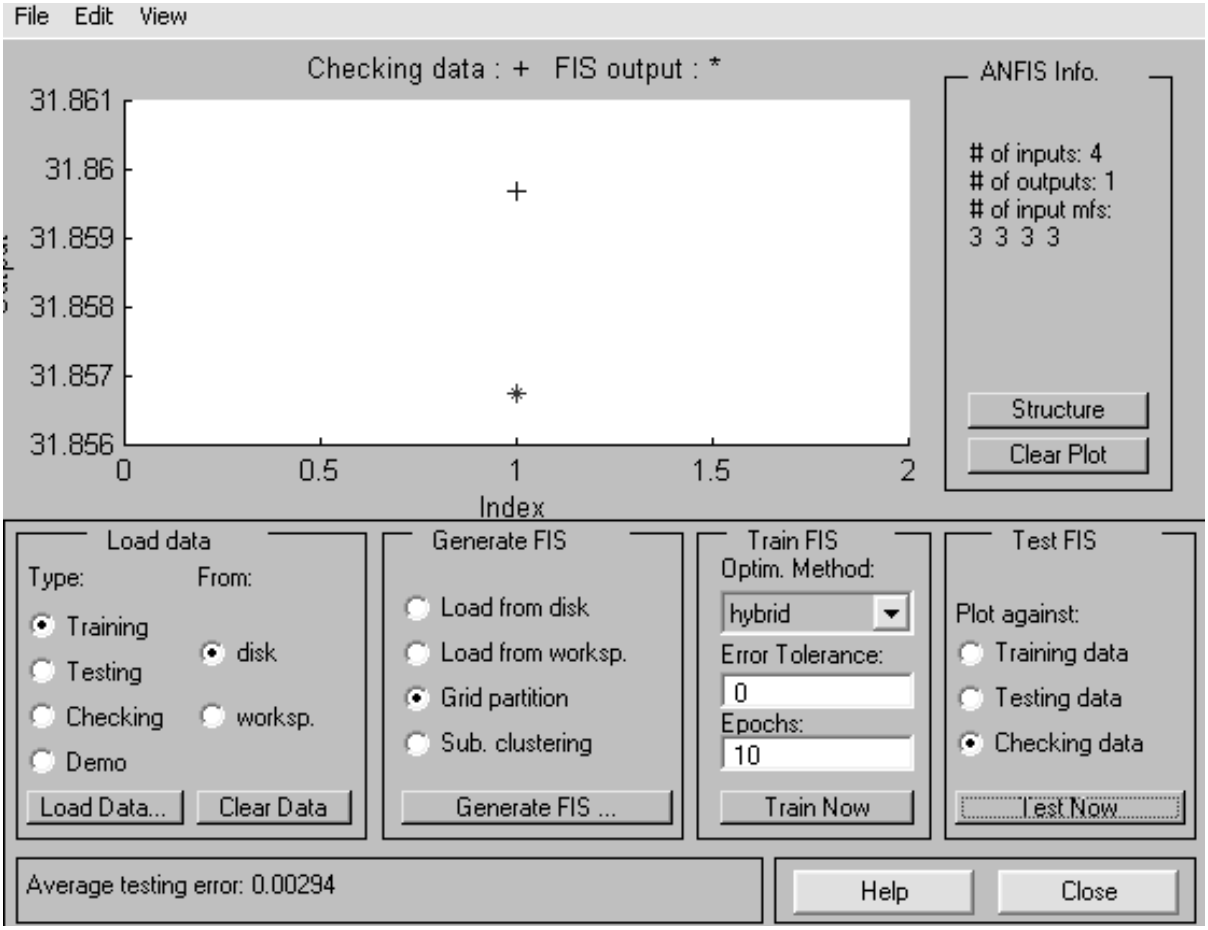

Figure 16.10. NFN simulation results for checking data

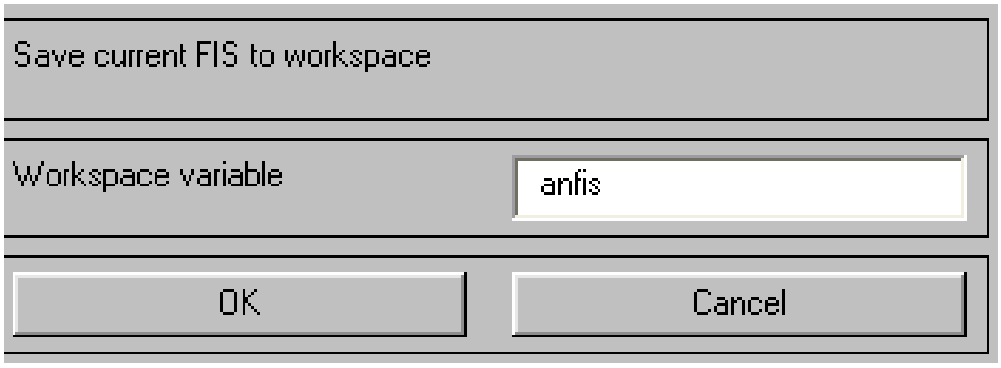

Figure 16.11. Results export

The actual value of the prognosis is 31.8597.

In this case, compared to a conventional feed-forward neural network, the result obtained is closer to the real value of the exchange rate.

16.3. Individual tasks

1. Formulate a prognosis problem from the field of computer technology. The use of NN and NFN should be justified for solving of this problem.

2. Form a training sample for NN and NFN.

3. Build a neural network and, experimenting with the number of neurons in the input and hidden layers, activation functions, training methods, achieve the best prediction result.

4. Visualize the resulting NN structure in the MATLAB system.

5. Build an NFN and, experimenting with learning methods, the number of membership functions in the input layer, achieve the best prediction result. Take the number of inputs the same as in paragraph 3 of the individual task.

6. Visualize the resulting NFN structure in the MATLAB system.

7. Build a fuzzy logical inference system for the resulting NFN.

8. Compare the results obtained with the help of NN and NFN (numerically) and draw conclusions.

9. Design the lab's report.

REPORTS DESIGN REQUIREMENTS

Reports are drawn up in English using *Microsoft Word* and *Times New Roman* font without hyphenation. Line spacing is one and a half, paragraph indentation is 1.25 cm. Report volume is no more than 10 printed pages. The report is prepared on A4 sheets using portrait orientation. Top, bottom and right margins are 2.0 cm, left one is 3.0 cm. Headers size is 0 cm, and footers size is 2.0 cm. Numbering is performed at the bottom of the page, font size is 14 *pt*, alignment is center.

The first line contains the last name and initials of the student, as well as the number of the academic group. Font size is 12 *pt*, bold, italic, right alignment.

The second line contains the work number printed in uppercase (font is straight, bold, 14 *pt*, center alignment).

Next, one blank line after the work number the topic of the work is printed in uppercase, (font is straight, bold, 14 *pt*, center alignment).

Then one blank line after the topic of the work the goal of the work is printed. Font size is 14 *pt*, indent is 1.25 *pt*, alignment is justified.

Further, the results of the execution of all individual task points are printed. The font of the text is 14 *pt*, paragraph indentation is 1.25 cm, alignment is justified.

The conclusions of the work are given at the end. They are printed one blank line after the results of the work. Font size is14 *pt*, indent is 1.25 *pt*, justified alignment.

REFERENCES

1. Zimmermann H.J. Fuzzy Set Theory-and Its Applications,/ H. J. Zimmermann. – New York: Springer Science+Business Media, 2001. – 525 p.

2. Hooda D.S. Fuzzy Logic Models and Fuzzy Control. An Introduction/ D.S. Hooda, V. Raich– UK: Alpfa Science, 2017. – 408 p.

3. Bassanezi R.C. A First Course in Fuzzy Logic, Fuzzy Dynamical Systems, and Biomathematics: Theory and Applications / R.C. Bassanezi. – New York: Springer Science, 2017. – 304 p.

4. Fuzzy Logic Toolbox: User's Guide / [Internet resource]. The MathWorks – Natick, MA: MathWorks, 2018. – 528 p. – Access mode: https://person.dibris.unige.it/masulli-francesco/lectures/ML-CI/lectures/MAT LAB% 20fuzzy%20toolbox.pdf.

5. Alavala Ch. R. Fuzzy Logic and Neural Networks Basic Concepts and Applications/ Ch. R. Alavala. – Delhi: New Age international publishers, 2007. – 276 p.

6. Tan C. An Artificial Neural Networks Primer with Financial Applications: Examples in Financial Distress Predictions and Foreign Exchange Hybrid Trading System / Clarence N W Tan. – Bond University, 1997. – 123 p.

7. Mettenheim H.-J. H. Advanced Neural Networks: Finance, Forecast, and other applications / Hans-Jörg Henri von Mettenheim. – Hanover: Computer Science, 2010. – 267 p.

8. Wasserman P.D. Neural Computing: Theory and Practice / Philip D. Wasserman. – Coriolis Group, 2006. – 230 p.

9. Ruan D. Intelligent Hybrid Systems: Fuzzy Logic, Neural Networks, and Genetic Algorithms / D. Ruan. – Springer, 1997.– 373 p.

10. Yang W.Y. Applied numerical methods using MATLAB / W. Y. Yang, W. Cao, T. Chung, J. Morris. – A John Wiley & sons, inc., publication, 2005. – 511 p.

11. Feng G. Analysis and Synthesis of Fuzzy Control Systems: A Model-Based Approach / G. Feng. – CRC Press, 2018. – 299 p.

12. Rajashekaran S. Neural Networks, Fuzzy Logic and Genetic Algorithms / S. Rajashekaran, G.A. Vijayalksmi. – Prentice-Hall of India Pvt.Ltd, 2004. – 456 p.

CONTENTS

NOTES

NOTES

NOTES

Educational edition

ZAKOVOROTNIY Oleksandr
LIPCHANSKA Oksana

FUNDAMENTALS OF COMPUTATIONAL INTELLIGENCE

Part 2

Laboratory workshop
for full-time and part-time students
by Computer Engineering and Computer Science direction
(Engl. language)

The work for the publication was recommended by *V. D. Dmitrienko*
Responsible person for the release is *S. Yu. Leonov*
In the author's edition

Signed for publication 29.12.2021. Format 60 × 84 $^{1}/_{16}$. Paper Copy Paper. Printing – risography. Times New Roman Font. Accounting-publishing sheet 6,2. Contractual price.

Publishing house NTU "KhPI".
61002, Kharkiv, Kirpichova str., 2