МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ «МОДЕЛИ И МЕТОДЫ ИССЛЕДОВАНИЯ ИНФОРМАЦИОННЫХ ПРОЦЕССОВ И СИСТЕМ»

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Направление подготовки 09.04.02 «Информационные системы и технологии»
Направленность (профиль) «Технологии работы с данными и знаниями, анализ информации»

Пятигорск, 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

1. Цель и задачи освоения дисциплины (модуля)

Целью освоения дисциплины «Модели и методы исследования информационных процессов и систем» является получение знаний, умений и навыков по использованию методов исследования и моделирования информационных процессов и систем при проведении научных исследований и решении практических задач, а также формирование на их основе компетенций будущего магистра по направлению подготовки 09.04.02. – Информационные системы и технологии и направленности (профилю) «Технологии работы с данными и знаниями, анализ информации». Освоение методологии и технологии моделирования, в первую очередь компьютерного, информационных процессов в различных системах. Приобретение навыков профессионального использования современных пакетов имитационного моделирования при разработке моделей задач из различных сфер деятельности.

Задачи дисциплины: использование методологии системного анализа для исследования информационных процессов, систем и технологий; моделирование информационных процессов и систем; изучение современных подходов к формализации информационных систем; использование информационных технологий при анализе и синтезе информационных систем и инструментальных средств моделирования информационных процессов и систем

2. Перечень планируемых результатов обучения по дисциплине (модулю), соотнесённых с планируемыми результатами освоения образовательной программы

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ОПК-7 Способен разрабатывать и применять математические модели процессов и объектов при решении задач анализа и синтеза	ОПК-7.1 Применяет методы научных исследований и математического моделирования при решении задач анализа и синтеза	Правильно использует: методы анализа и синтеза информационных систем; формальные модели систем; средства структурного анализа; методологию структурного системного анализа и проектирования. Грамотно применяет методы научных исследований и математического моделирования при решении задач анализа и синтеза

распределенных информационных систем и систем поддержки принятия решений;	распределенных информационных систем и систем поддержки принятия решений	распределенных информационных систем и систем поддержки принятия решений Выбирает адекватные методы научного поиска и интеллектуального анализа научной информации при решении новых задач, методы анализа и синтеза информационных систем, распределенных информационных систем и систем поддержки принятия решений. Адекватно использует математический аппарат для решения задач в области информационных систем и технологий
	ОПК-7.2 Разрабатывает математические модели процессов и объектов при решении задач анализа и синтеза распределенных информационных систем и систем поддержки принятия решений	Анализирует динамические оптимизационные модели, модели предметных областей информационных систем, математические модели информационных процессов Разрабатывает модели процессов и объектов информационной среды, аналитические и имитационные модели предметных областей. Выбирает адекватные модели и средства разработки архитектуры информационных систем. Использует адекватные основные приемы по исследованию информационных систем и технологий
ОПК-8 Способен осуществлять эффективное управление разработкой программных средств и проектов	ОПК-8.1: Осуществляет управление работами по выявлению и анализу требований к программным средствам и проектам	Анализирует: принципы выявления, разработки, документирования требований в ИТ проектах; принципы изменения и планирования требований к программным средствам и проектам. Выполняет: обзор прикладных и информационных процессов; анализ прикладных и информационных процессов; реинжиниринг

		прикладных и информационных процессов. Демонстрирует практические навыки: выбора необходимого программного обеспечения; планирования и разработки программного обеспечения
	ОПК-8.2: Проводит мониторинг и управляет работами проекта в ИТ области	Грамотно использует: понятия жизненного цикла проекта: инициацию, планирование, исполнение, мониторинг и контроль, закрытие; основную стратегию разработки программных средств и проектов, особенности процессного подхода к управлению прикладными ИС и современные ИКТ в процессном управлении Осуществляет: эффективное управление проектами ИС на всех стадиях жизненного цикла, оценивание эффективности и качества проекта; применение современных методов управления проектами и сервисами ИС; обеспечение эффективного контроля и регулирования работ, а также управление изменениями Демонстрирует практические навыки по анализу спецификации, мониторингу, управлению и контролю работами проекта в ИТ области

ЛАБОРАТОРНАЯ РАБОТА 1
Моделирование процессов и систем в пакете Simulink системы MATLAB

Цель и содержание: изучить основные приемы работы с системой компьютерной математики MATLAB приобрести навыки построения моделей систем в пакете Simulink

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций

Вопросы для обсуждения на лабораторном занятии: построение простейших моделей в пакете Simulink системы MATLAB.

Формируемые компетенции: ОПК-7.1 ОПК-7.2 ОПК-8.1 ОПК-8.2

Теоретическое обоснование

Пакет прикладных программ MATLAB относится к инструментальным средствам, которые предназначены для решения задач в различных областях человеческой деятельности (научных, экономических, инженерных, физических и т. д.), позволяющим производить моделирование, разработку и отладку различных систем и устройств. Его используют более миллиона инженерных и научных работников, он работает на большинстве современных операционных систем, включая Linux, macOS, и Windows.

MATLAB имеет большое количество функций для анализа данных, практически из всех областей математики. MATLAB как язык программирования был разработан Кливом Моулером (Cleve Moler) в конце 1970-х годов. Язык MATLAB является высокоуровневым интерпретируемым языком программирования, включающим основанные на матрицах структуры данных, широкий спектр функций, интегрированную среду разработки, объектно-ориентированные возможности и интерфейсы к программам, написанным на других языках программирования.

Программы, написанные на MATLAB, бывают двух типов — функции и скрипты. Функции имеют входные и выходные аргументы, а также собственное рабочее пространство для хранения промежуточных результатов вычислений и переменных. Скрипты же используют общее рабочее пространство. Как скрипты, так и функции сохраняются в виде текстовых файлов и компилируются в машинный код динамически. Существует также возможность сохранять так называемые *pre-parsed* программы — функции и скрипты, обработанные в вид, удобный для машинного исполнения. В общем случае такие программы выполняются быстрее обычных, особенно если функция содержит команды построения графиков.

Основной особенностью языка MATLAB являются его широкие возможности по работе с матрицами

Интерфейс системы MATLAB представлен 1.1.

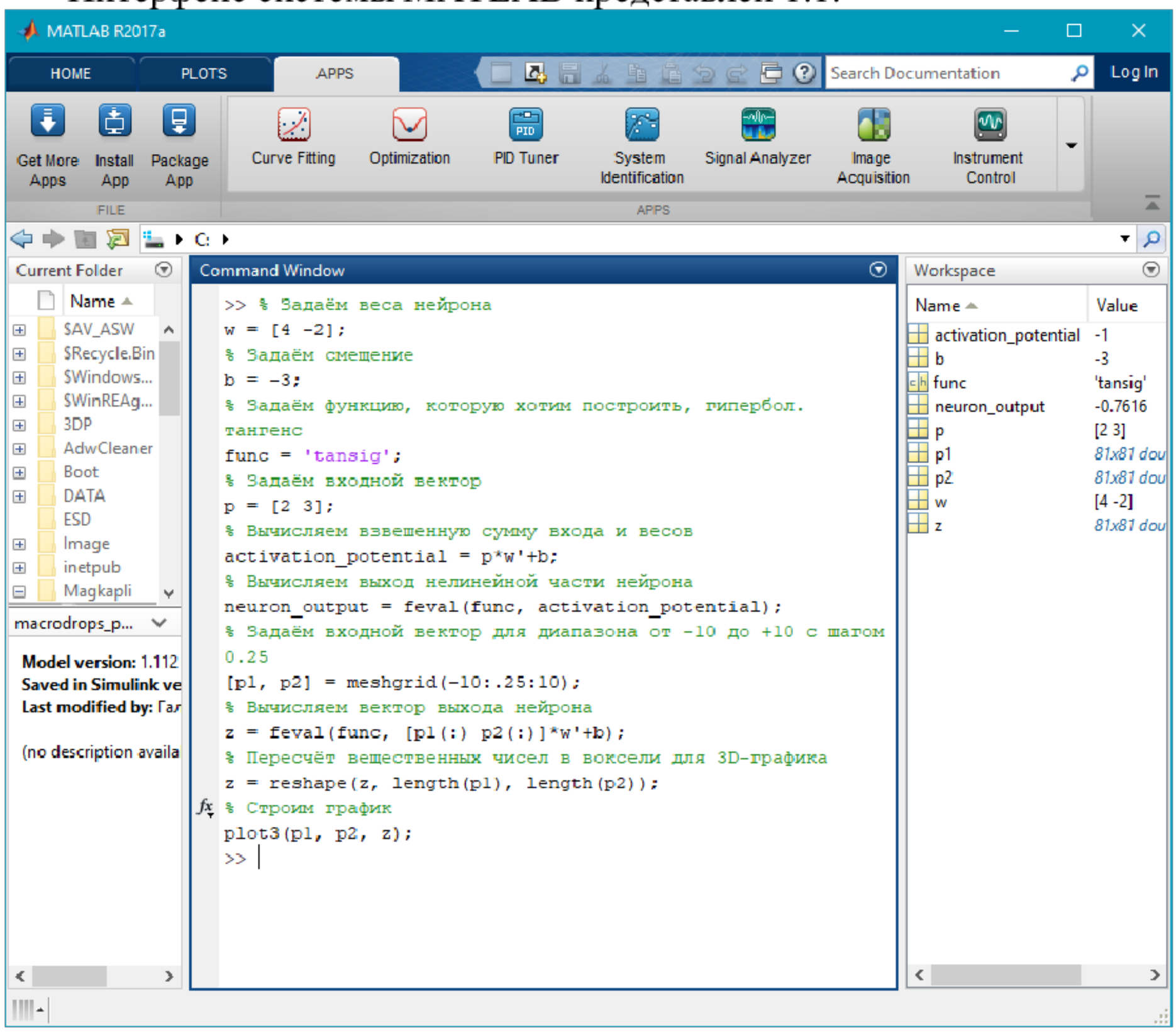


Рисунок 1.1 –Интерфейс системы MATLAB

Simulink – это подсистема имитационного математического моделирования динамических процессов. Simulink является составной частью пакета Matlab и полностью интегрирован с ним. Модели в Simulink состоят из набора графических блоков, представляющих компоненты объекта или какие-то функциональные элементы (источники сигналов различного вида, виртуальные регистрирующие приборы, средства анимации), и направленных связей между ними (линии распространения сигналов). Simulink позволяет осуществлять имитационное моделирование поведения различных динамических нелинейных систем.

Для начала работы необходимо создать новый файл Simulink model, в котором с помощью библиотек Simulink (рисунок 1.2, 1.3) и осуществить графическую сборку системы из отдельных блоков, хранящихся в библиотеках Simulink.

Рассмотрим некоторые блоки математических операций (рисунок 1.2):

- **Abs** предназначен для вычисления абсолютного значения входного сигнала;

- **Add** предназначен для вычисления алгебраической суммы двух (или более) сигналов;

Gain умножает входной сигнал на заданный коэффициент и т.д.

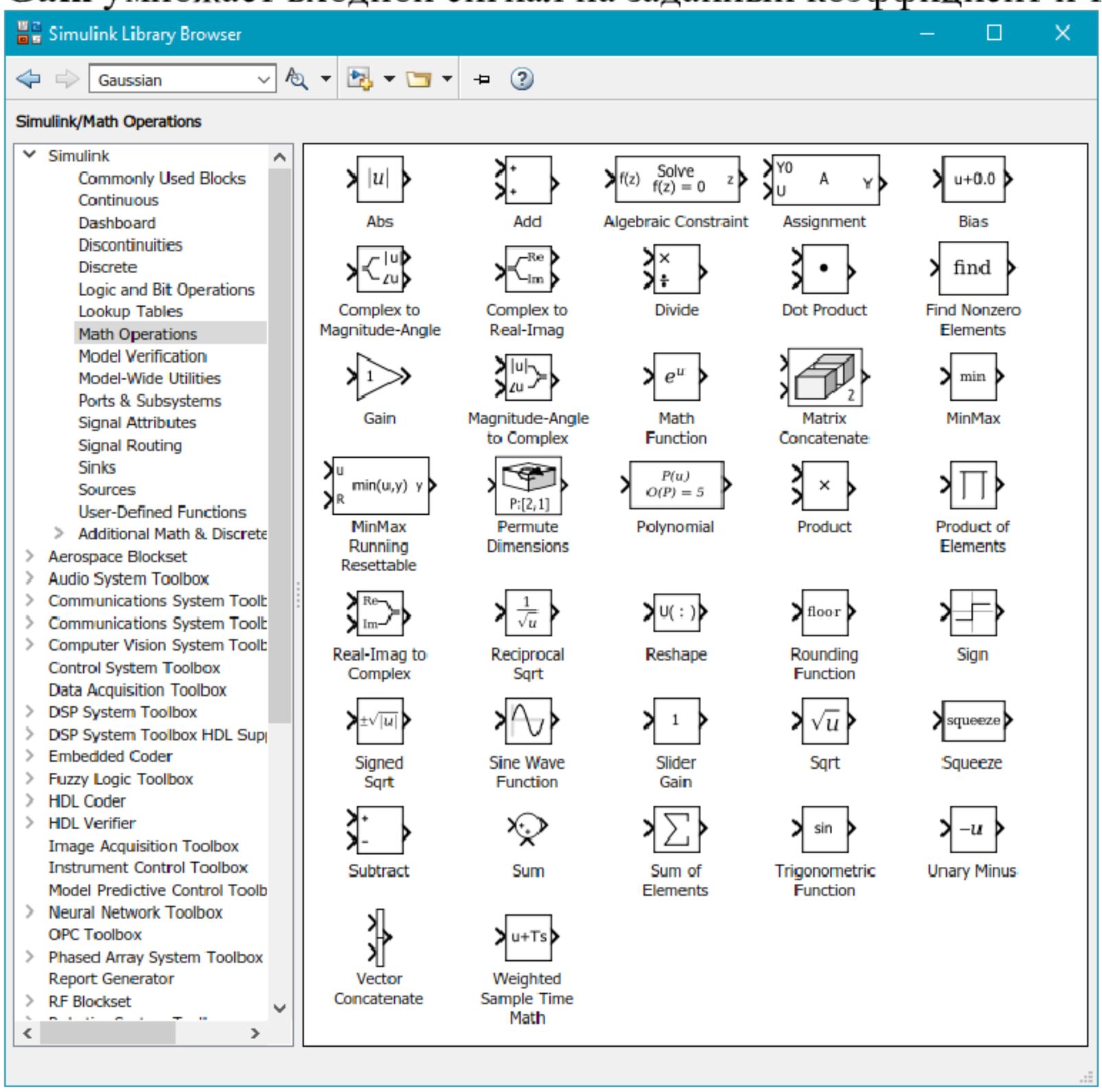


Рисунок 1.2 – Библиотеки Simulink. Блоки математических операций

Рассмотрим часто встречающиеся блоки (рисунок 1.3):

- **From File** – блок считывания данных из файла;
- **Sine Wave** формирует синусоидальный сигнал с заданной частотой, амплитудой, фазой и смещением;
- **Signal Generator** формирует один из четырех видов периодических сигналов (синусоидальный, прямоугольный, пилообразный, случайный сигналы);
- **Ramp** формирует линейный сигнал;
- **Step** формирует ступенчатый сигнал;
- **Random Number** формирует случайный сигнал с нормальным распределением уровня сигнала и т.д.

Аппаратура и материалы. Для выполнения лабораторной работы необходимо использовать следующее: аппаратное обеспечение: персональный компьютер и программное обеспечение: операционную систему Windows 10 и выше; Microsoft Office13 и выше, системы компьютерной математики Machcad и MATLAB R2017a и выше.

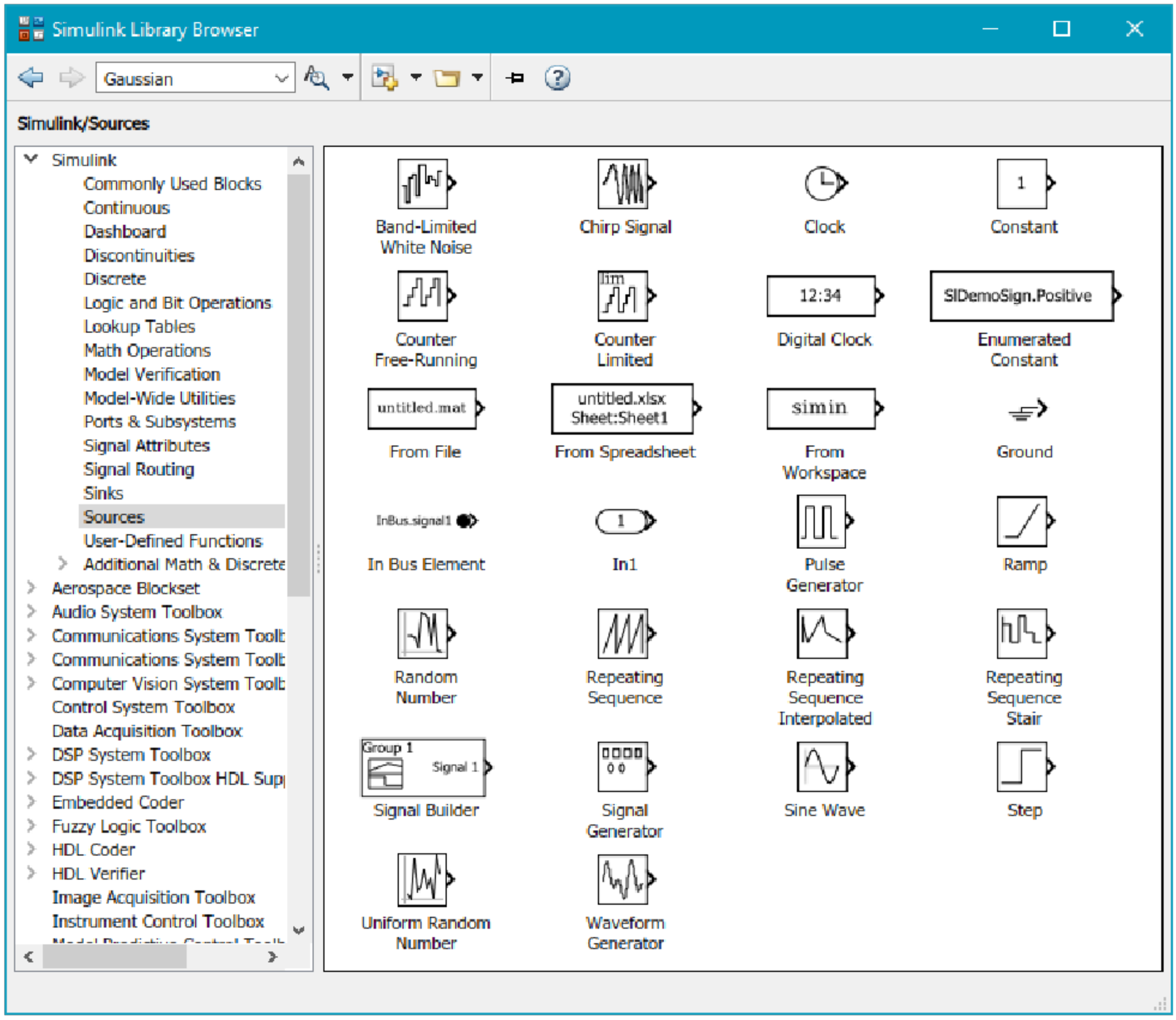


Рисунок 1.3 – Блоки библиотеки Simulink/Sources

Методика и порядок выполнения работы

Выполните предложенные задания, предварительно рассмотрев приведенные в примеры.

Задание 1.1. Построение простейших моделей

Построить модель сигнала вида $x(t) = A\sin(\omega t+\varphi)+t^m$ на интервале $[a; b]$, и результаты моделирования вывести на виртуальный осциллограф. Параметры сигналов приведены в таблице 1.1. Номер варианта соответствует номеру, под которым студент записан в списке группы.

На рисунке 1.4 приведен один из вариантов построения сигнала: $x(t)=1{,}5\sin(\pi t+\pi/3)+t^2$ с помощью функциональных блоков. Сигнал $1{,}5\sin(\pi t+\pi/3)$ задан в окне диалога параметров блока Sine Wave – Source Block Parameters: Sine Wave (рисунок 1.5).

Таблица 1.1 – Параметры сигнала

№ варианта	Амплитуда, A	Частота, ω	Фаза, φ	Степень, *m*	Интервал,

					[*a*; *b*]
1.	0,5	*2π*	*π*/3	2	[0; 3]
2.	2,5	3*π*/4	*π*/6	3	[0; 1]
3.	1,5	2*π*/3	*π* /8	2	[0; 5]
4.	4,5	*π*/4	5*π*/4	4	[0; 6]
5.	3,5	3*π*/2	3*π*/5	3	[0; 7]
6.	8,5	*π*/8	3*π*/4	1,5	[0; 4]
7.	7,5	5*π*/4	2*π*/3	3,5	[0; 1,5]
8.	9,5	3*π*/5	*π*/4	2	[0; 2,5]
9.	0,9	*π*/3	3*π*/2	4	[0; 1,8]
10.	0,45	*π*/6	*π*/12	3	[0; 3,2]

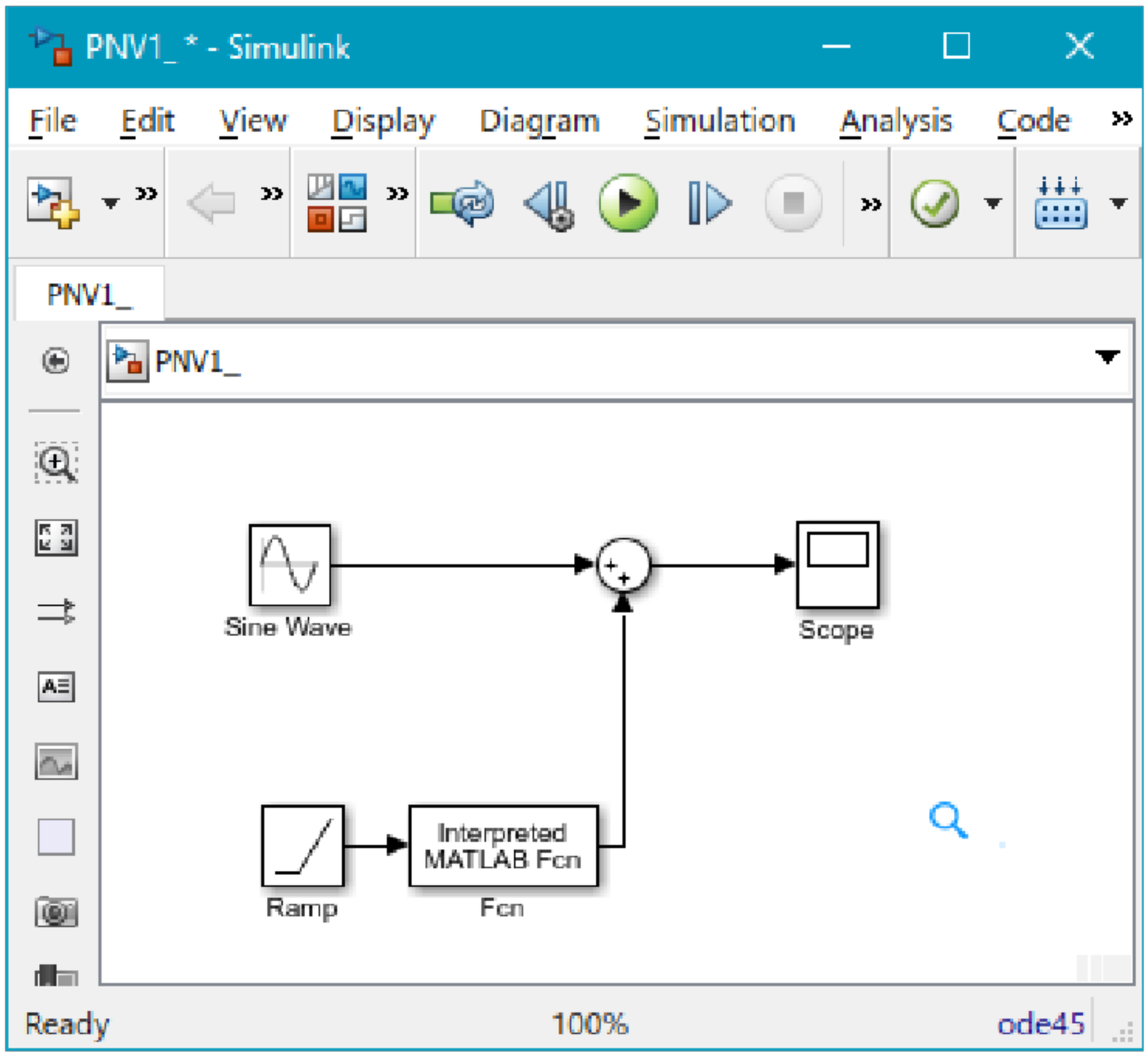


Рисунок 1.4 – Структурная схема модели сигнала $x(t)=1,5\sin(\pi t+\pi/3)+t^2$

Для сигнала t^2 использовались два блока – блок линейного сигнала и блок математической функции, для которого в соответствующем окне была введена функция в степени 2 (рисунок 1.6).

Интервал моделирования задан в пределах от 0 до 2 в диалоговом окне Configuration Parameters меню Simulation (рисунок 1.7).

Результаты моделирования работы выведены на экран виртуального осциллографа (рисунок 1.8).

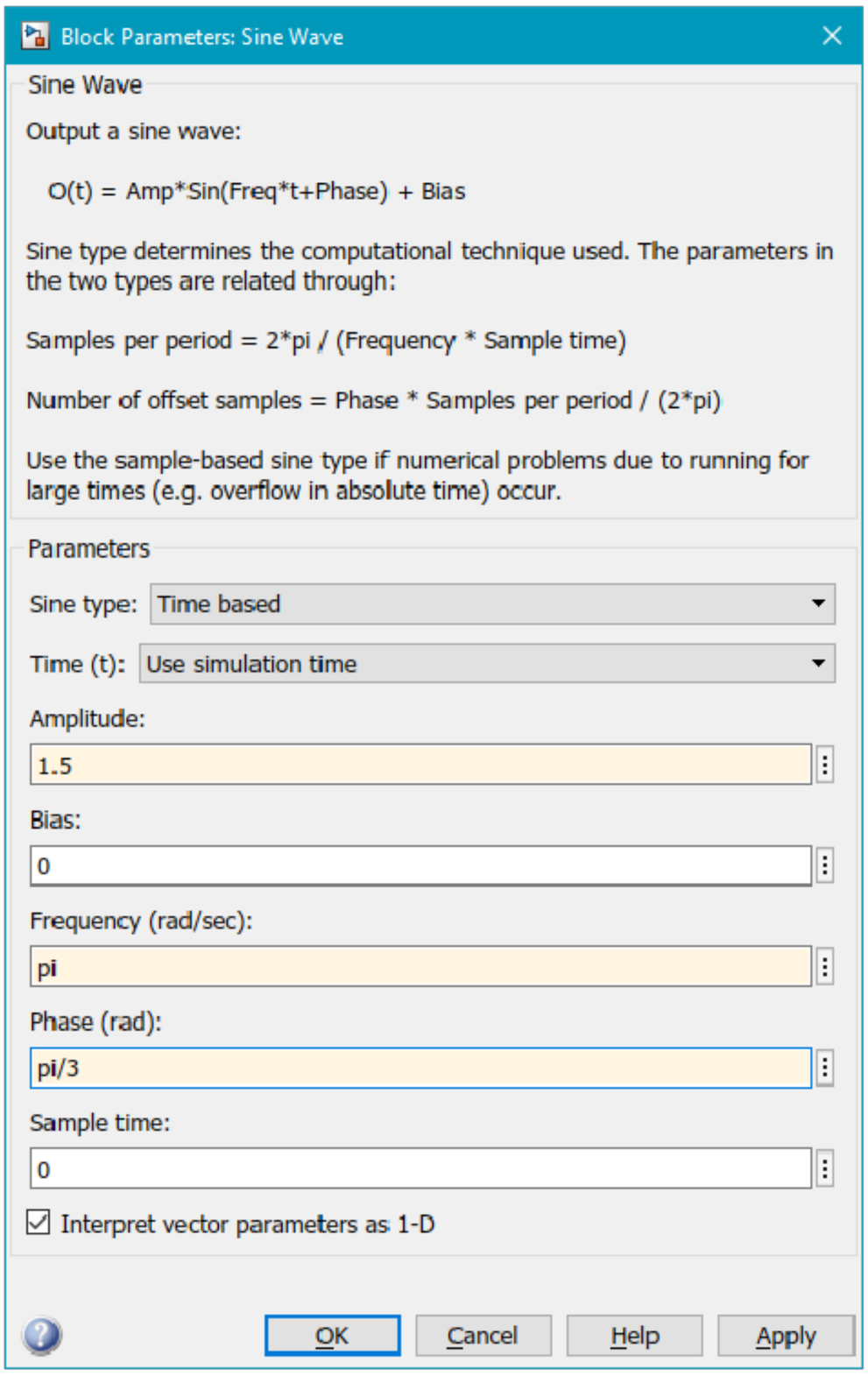


Рисунок 1.5– Окно диалога параметров блока Sine Wave с заданными параметрами сигнала

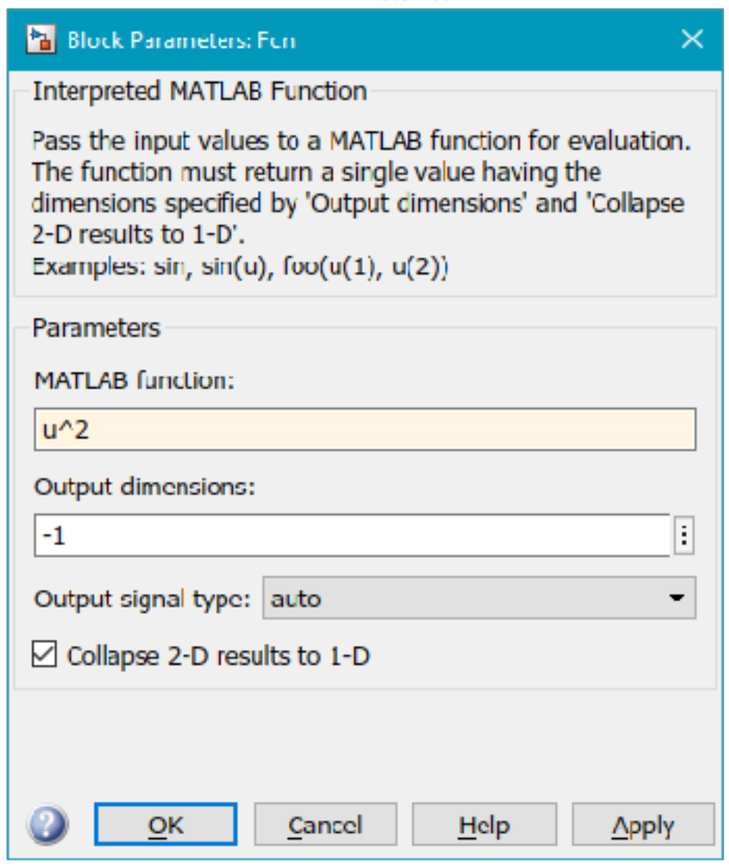


Рисунок 1.6 – Параметры функции t^2

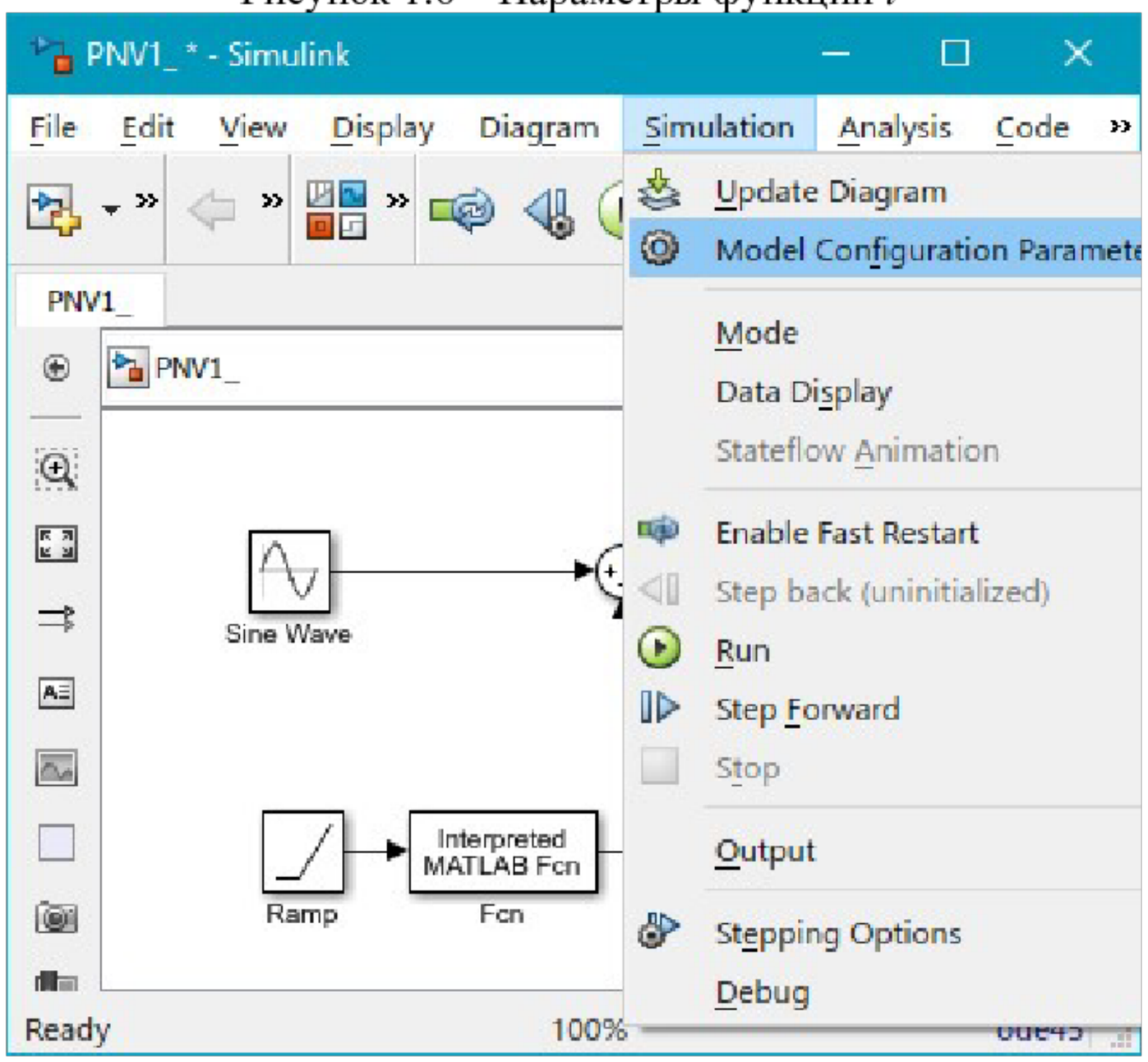


Рисунок 1.7 – Вызов окна диалога Configuration Parameters меню Simulation

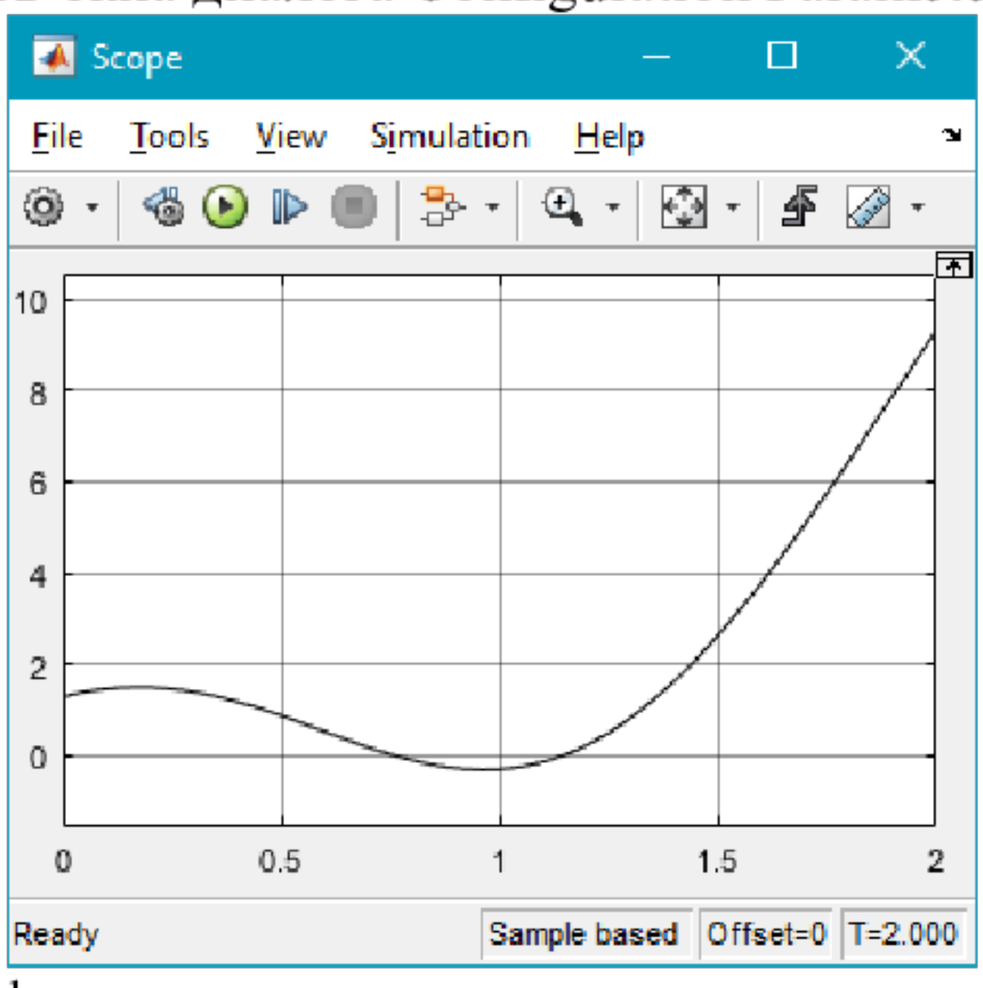


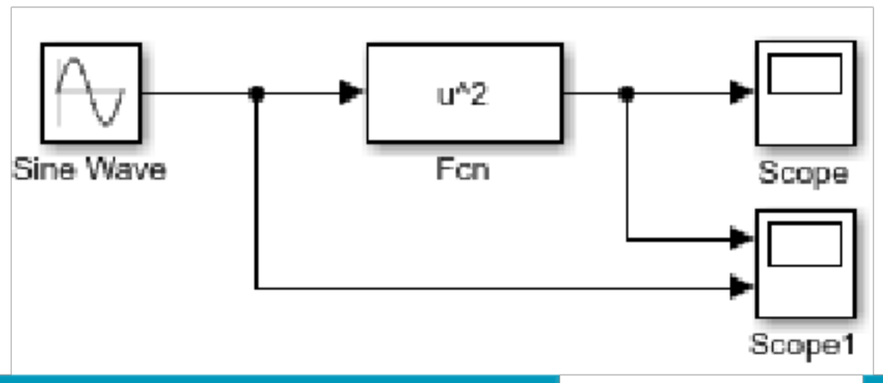
Рисунок 1.8 –График заданного сигнала на экране виртуального осциллографа

Задание 1.2. Моделирование сигнала заданной формы:

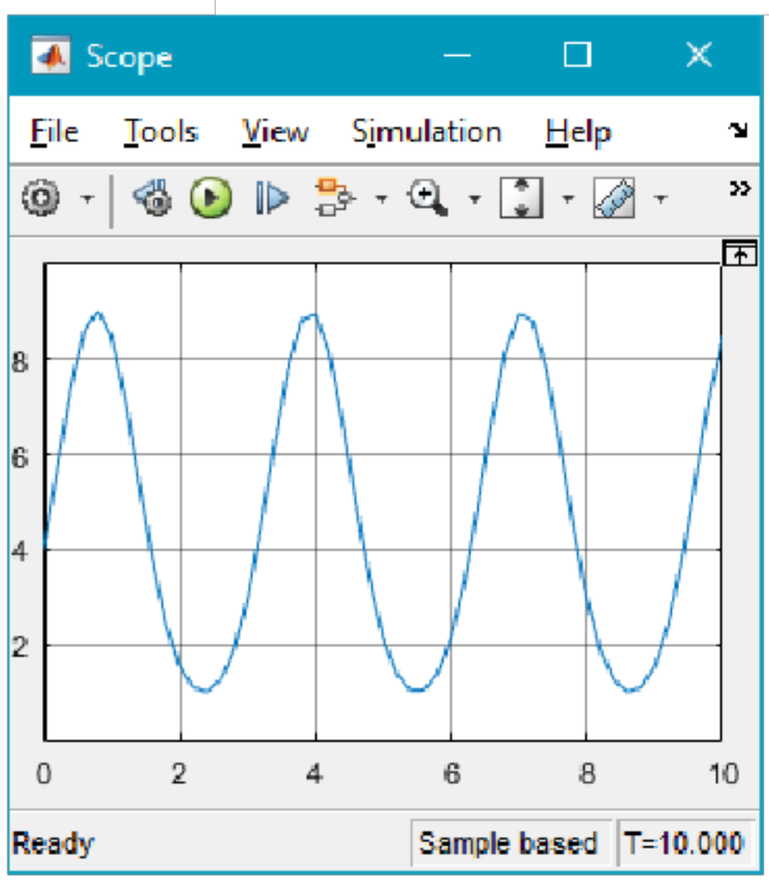
$x(t) = (2+\sin(2t))^2$; $x(t) = (2+0{,}5\sin(2t))^{1/2}$; $x(t) = te^{-t}\cos(2\pi t+\pi/4)$;

$x(t) = 1.5 + 0.7t - 0.05t^2 - (e^{-3t}+t^3)^{1/3}$

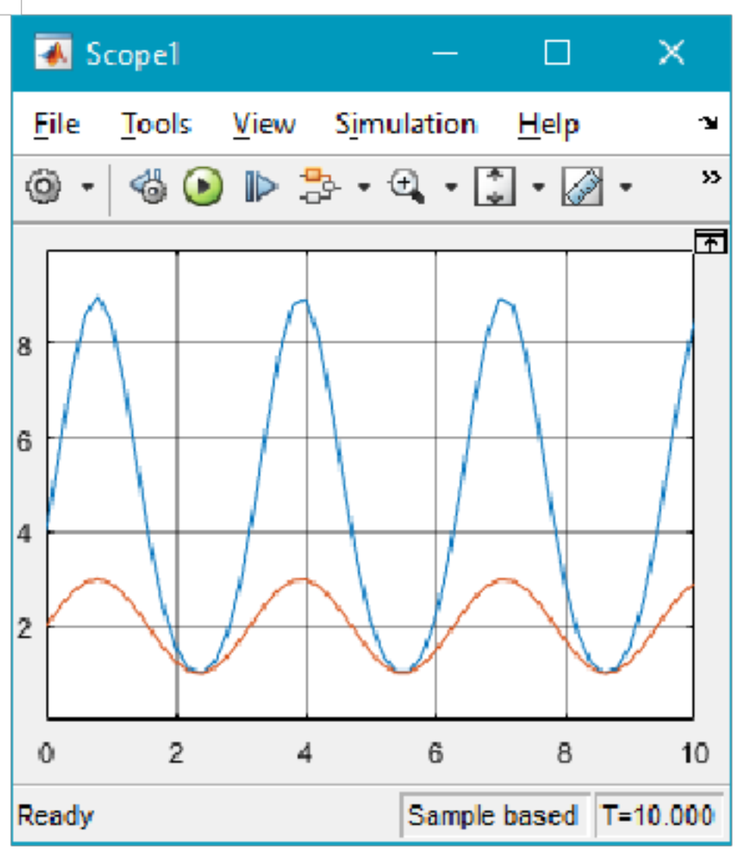
На рисунке 1.9 приведен один из вариантов построения сигнала: $x(t) = (2+sin(2t))^2$ с помощью функциональных блоков и результаты.



а



б



с

Рисунок 1.9 – Структурная схема модели сигнала $x(t) = (2+sin(2t))^2$ – *а* и графики двух сигналов на экране виртуального осциллографа: $2+sin(2t)$ –*б*, и $x(t) = 2+\sin(2t)$ – *с*

Задание 1.3. Моделирование интегрированного сигнала

В системах связи помехи и замирания, воздействующие на сигнал при его прохождении по каналу связи, имеют статистический характер и могут быть описаны при помощи различных законов распределений. В частности, замирания в канале связи при отсутствии прямой видимости между абонентом и базовой станцией имеют рэлеевский закон распределения; аддитивные помехи (шумы) часто описываются нормальным (гауссовским) законом распределения; временные интервалы между вызовами в сетях связи обычно имеют экспоненциальный закон распределения и.т.д.

Построить модель для имитации смеси сигналов:

- синусоидального сигнала и белого шума (рисунок 1.10);
- синусоидального сигнала и гауссова шума (с математическим ожиданием m=0,и дисперсией σ^2 =0.01);
- синусоидального сигнала и шума с равномерным распределением.

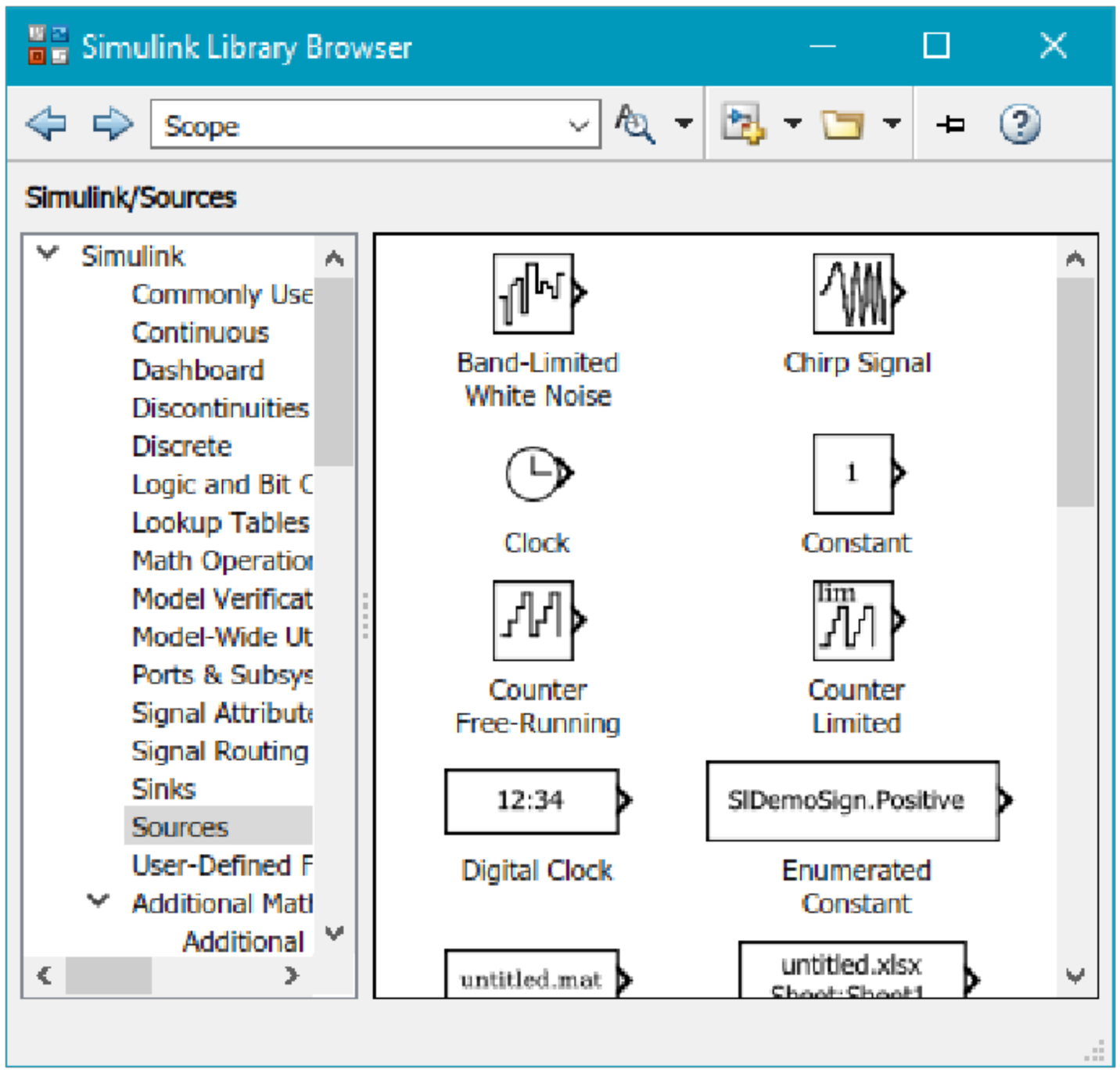


Рисунок 1.10 – Блоки библиотеки Sources

Пример построения модели для имитации смеси синусоидального сигнала белого шума и график полученного в результате имитации результирующего сигнала представлены на рисунках 1.11, 1.12.

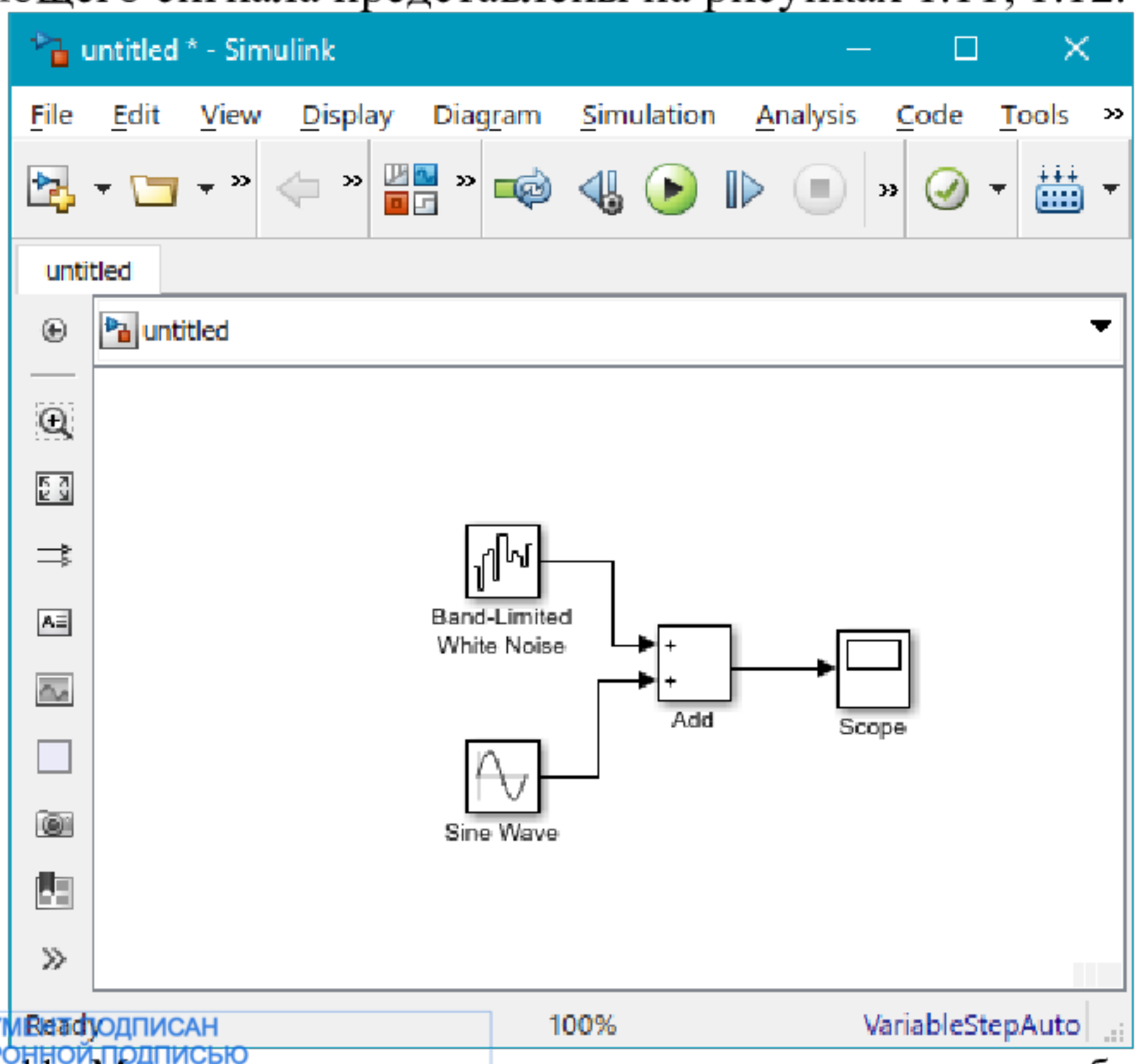


Рисунок 1.11– Модель имитации смеси сигналов синусоидального и белого шума

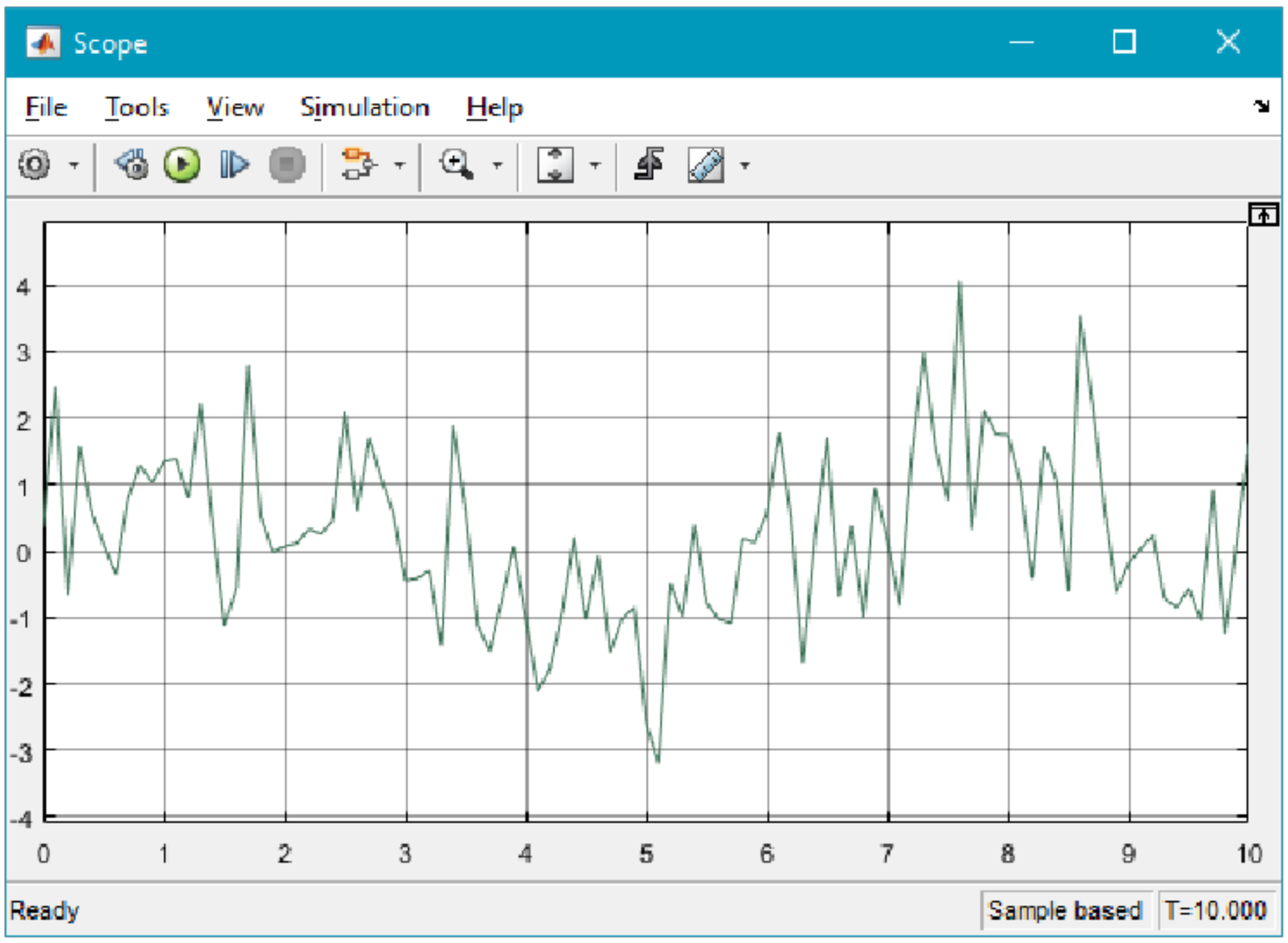


Рисунок 1.12 –Результат имитации смеси сигналов синусоидального и белого шума

Содержание отчета и его форма

Подготовьте отчет, в котором приведите технологию выполнения заданий.

Отчет по лабораторной работе должен содержать:

1) название работы;
2) цель лабораторной работы;
3) формулировку задания и технологию его выполнения;
4) ответы на контрольные вопросы;
5) приложение – файлы выполненных заданий.

Вопросы для защиты работы

1. Для чего используется система MATLAB?
2. Как осуществляется программирование в среде MATLAB?
3. Для чего используется пакет Simulink?
4. Как с помощью Simulink можно моделировать поведение сложных систем?
5. На какой технологии основана разработка моделей средствами SIMULINK (S-модели)?

ЛАБОРАТОРНАЯ РАБОТА 2
Моделирование объектов, описываемых системами уравнений

Цель и содержание: использование классических методов аналоговой вычислительной техники для моделирования объектов, описываемых системами алгебраических уравнений, приобретение навыков построения таких моделей в системе компьютерной математики MATLAB.

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций

Вопросы для обсуждения на лабораторном занятии: построение моделей объектов, описываемых системами алгебраических уравнений.

Формируемые компетенции: ОПК-7.1 ОПК-7.2 ОПК-8.1 ОПК-8.2

Теоретическое обоснование

Приближенное решение широкого круга вычислительных задач осуществляется с помощью методов решения систем линейных уравнений, если удается свести задачу к системе уравнений. Эти методы обычно делят на две большие группы. К первой группе относятся точные методы, которые позволяют для любых систем найти точные значения неизвестных после конечного числа точно выполняемых арифметических операций.

Ко второй группе относят приближенные методы, которые являются итерационными, так как решения в них получают в результате процесса приближений. Точные методы применяются для задач небольших размерностей (~10^2), а для задач большой размерности используют итерационные методы.

Особое место среди них занимают вероятностные методы, которые полезны лишь в случаях очень высокой размерности систем.

Для моделирования объектов или процессов, протекание которых можно описать с помощью системы линейных уравнений, применяют метод сущность которого заключается в замене системы линейных уравнений (2.1) системой дифференциальных уравнений (2.2) ей эквивалентной.

Пусть система линейных уравнений, описываемая исследуемый объект или процесс имеет вид:

$$\begin{aligned} a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_n&=b_1\\ a_{21}x_1+a_{22}x_2+\ldots+a_{2n}x_n&=b_2\\ &\ldots\\ a_{n1}x_1+a_{n2}x_2+\ldots+a_{nn}x_n&=b_n \end{aligned} \qquad (2.1)$$

или в матричном виде: $\boldsymbol{A}x=\boldsymbol{b}$, где $\boldsymbol{A}$–квадратная матрица размером $n \times n$, $\boldsymbol{b}$ и x – векторы размером n (n– размерность системы).

Заменим данную систему алгебраических уравнений эквивалентной системой дифференциальных уравнений:

$$\begin{aligned}
&\frac{dx_1}{dt}+a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_n-b_1=0\\
&\frac{dx_2}{dt}+a_{21}x_1+a_{22}x_2+\ldots+a_{2n}x_n-b_2=0\\
&\ldots\\
&\frac{dx_n}{dt}+a_{n1}x_1+a_{n2}x_2+\ldots+a_{nn}x_n-b_n=0
\end{aligned} \qquad (2.2)$$

Доказательством эквивалентности систем 2.1 и 2.2 является наличие затухающего решения системы дифференциальных уравнений 2.2, т.е. как только все производные станут равны $\left(\frac{dx_i}{dt}=0\right)$, решение системы уравнений 2.1 будет найдено.

Для обеспечения затухающего решения достаточным условием является положительное значение определенности матрицы коэффициентов системы уравнений. Это возможно, в частности, при условии, когда

$$a_{ij}\geq\sum_{j=1}^{n}a_{ij}, i\neq j. \qquad (2.3)$$

Рассмотрим пример решения системы линейных алгебраических уравнений:

$$\begin{cases}4x_1+2x_2=14\\2x_1+5x_2=-5\end{cases} \qquad (2.4)$$

с помощью инструментальных средств табличного процессора Excel; систем Mathcad и Matlab, пакета Simulink.

Рассмотрит технологию решения системы (2.4) с помощью выше указанных программных средств.

Решение системы линейных алгебраических уравнений в табличном процессоре Excel можно получить, используя матричную форму записи системы линейных уравнений в: $\boldsymbol{Ax = b}$, следовательно, вектор решения: $x = \boldsymbol{A^{-1}\cdot b}$. На рисунке 2.1. представлено решение системы уравнений 2.4 с помощью Excel. Алгоритм решения приведен работе №4

Решение систем линейных уравнений в MathCAD можно получить с помощью встроенной функции **lsolve (*A*,*b*)**, которая возвращает вектор ***x*** для системы линейных уравнений при условии, если задана матрица коэффициентов и вектор свободных членов. Также для решения системы линейных уравнений используются формулы Крамера и метод Гаусса.

На рисунке 2.2 представлено решение системы линейных уравнений методом обратной матрицы и с помощью функции **lsolve(*A*,*b*)**.

Для решения системы уравнений по формулам Крамера необходимо:

1. Переменной ORIGIN присвоить значение равное единице.
2. Ввести матрицу системы и столбец правых частей системы уравнений.
3. Вычислить определитель матрицы системы, учитывая, что система имеет единственное решение, если определитель отличен от нуля.

4. Вычислить определители новых матриц, полученных путем замены соответствующего столбца столбцом свободных членов.

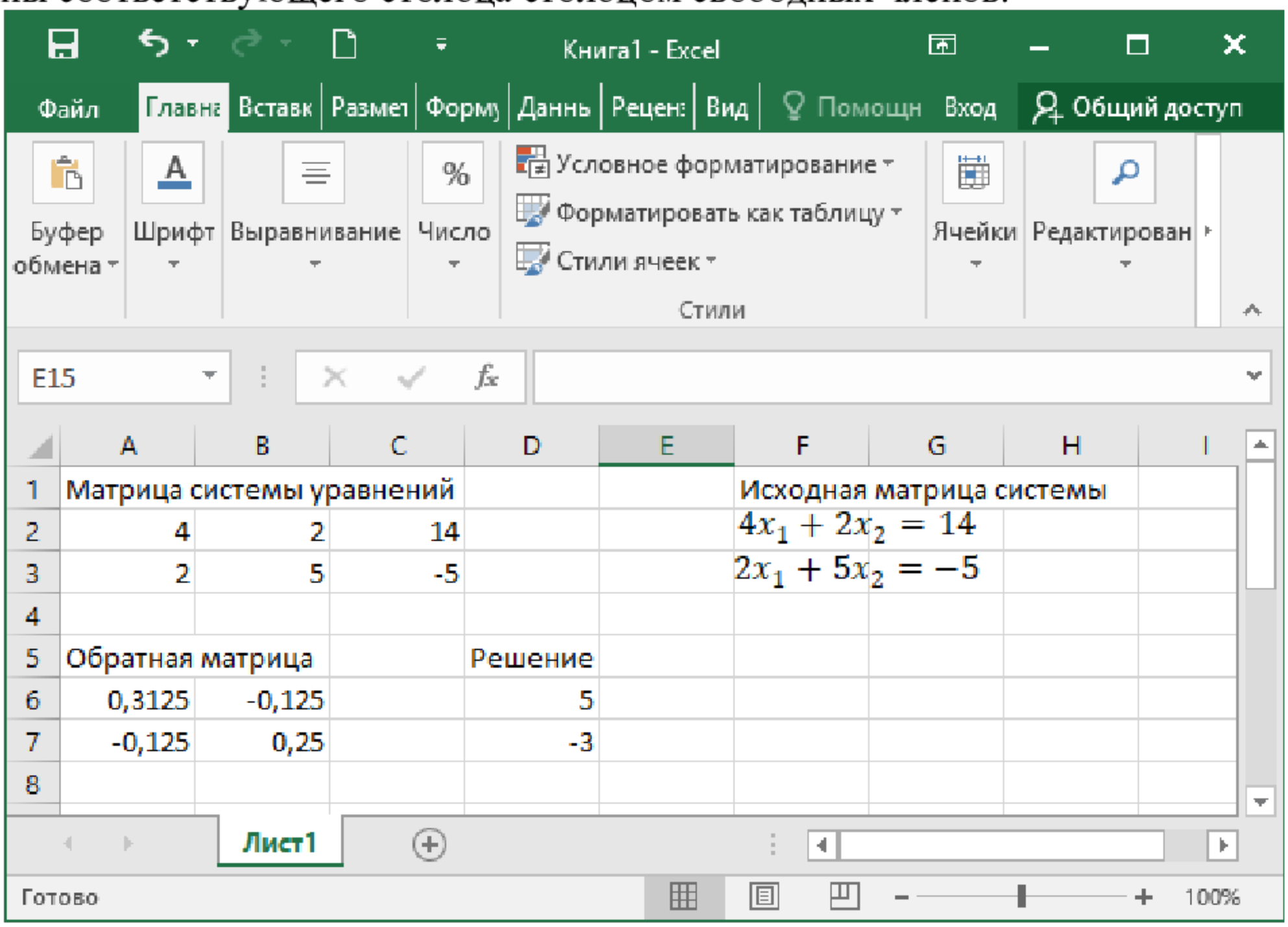


Рисунок 2.1 – Рабочий лист с решением системы линейных алгебраических уравнений (2.4)

$$4\cdot x_1 + 2\cdot x_2 = 14$$
$$2\cdot x_1 + 5\cdot x_2 = -5$$

Запись системы линейных уравнений в матричном виде

$$A := \begin{pmatrix} 4 & 2 \\ 2 & 5 \end{pmatrix} \qquad b := \begin{pmatrix} 14 \\ -5 \end{pmatrix}$$

Решение системы линейных уравнений

$$x := A^{-1}\cdot b \qquad x = \begin{pmatrix} 5 \\ -3 \end{pmatrix}$$

Решение системы линейных уравнений с помощью функции lsolve(A,b)

$$x := \mathrm{lsolve}(A,b) \qquad x = \begin{pmatrix} 5 \\ -3 \end{pmatrix}$$

Рисунок 2.2 – Решение системы линейных уравнений 2.4 методом обратной матрицы и с помощью функции **lsolve (A,b)**

ORING := 1

$$A := \begin{pmatrix} 4 & 2 \\ 2 & 5 \end{pmatrix} \qquad b := \begin{pmatrix} 14 \\ -5 \end{pmatrix} \qquad \Delta := |A| \qquad \Delta = 16$$

$$\Delta 1 := \begin{pmatrix} 14 & 2 \\ -5 & 5 \end{pmatrix} \qquad \Delta 2 := \begin{pmatrix} 4 & 14 \\ 2 & -5 \end{pmatrix}$$

$$|\Delta 1| = 80 \qquad |\Delta 2| = -48$$

$$x_1 := \frac{|\Delta 1|}{\Delta} \qquad x_2 := \frac{|\Delta 2|}{\Delta}$$

$$x_1 = 5 \qquad x_2 = -3$$

Рисунок 2.3 – Рабочий лист Mathcad с решением системы уравнений 2.4 методом Крамера

5. Определить решение системы с помощью формул Крамера: $x_i = \frac{\Delta_i}{\Delta}$, где Δ – определитель матрицы системы, Δ_i – определитель матрицы *n*-го порядка, полученный из матрицы системы заменой *i*-го столбца столбцом правых частей системы уравнений.

Решение системы линейных уравнений с помощью формул Крамера представлено на рисунке 2.3.

Для решения системы уравнений методом Гаусса необходимо:

1. Переменной **ORIGIN** присвоить значение равное единице.
2. Ввести матрицу системы и столбец правых частей системы уравнений.
3. Сформировать расширенную матрицы системы с помощью функции **augment**(***A***,***b***).
4. Привести расширенную матрицу системы к ступенчатому виду с помощью функции **rref**(***Ar***).
5. Вывести решение системы, выделив последний столбец полученной матрицы для чего сформировать столбец решения системы с помощью функции **submatrix** (***Ag***, *ir*, *jr*, *ic* ,*jc*) – блок матрицы ***Ag***, состоящий из всех элементов, содержащихся в строках от *ir* до *jr* и столбцах от *ic* до *j*c.
6. Проверить правильность решения вычислив ***Ax– b***.

Выполните решение системы уравнений методом Гаусса самостоятельно.

На рисунке 2.4 представлен результат решения системы уравнений (2.4), в MATLAB.

```
Command Window
>> A=[4 2;2 5];
>> B=[14 -5];
>> X=B/A

X =

     5    -3

>>
```

Рисунок 2.4 – Решение системы уравнений (2.4) в MATLAB

Для решения данной системы в пакете Simulink необходимо перейти к эквивалентной системе дифференциальных уравнений в соответствии с системой (2.2):

$$\begin{cases} \frac{dx_1}{dt} = 14 - 4x_1 - 2x_2 \\ \frac{dx_2}{dt} = -5 - 2x_1 - 5x_2 \end{cases} \quad (2.5)$$

Структурная схема модели данной системы приведена на рисунке 2.5. Переходный процесс установления решения изображен на экране виртуального осциллографа (рисунок 2.6).

На рисунке 2.6 видно, что после $t = 2$ на выходах виртуальных интеграторов устанавливаются сигналы, соответствующие решению системы линейных алгебраических уравнений:

$$x_1 = 5,$$
$$x_2 = -3.$$

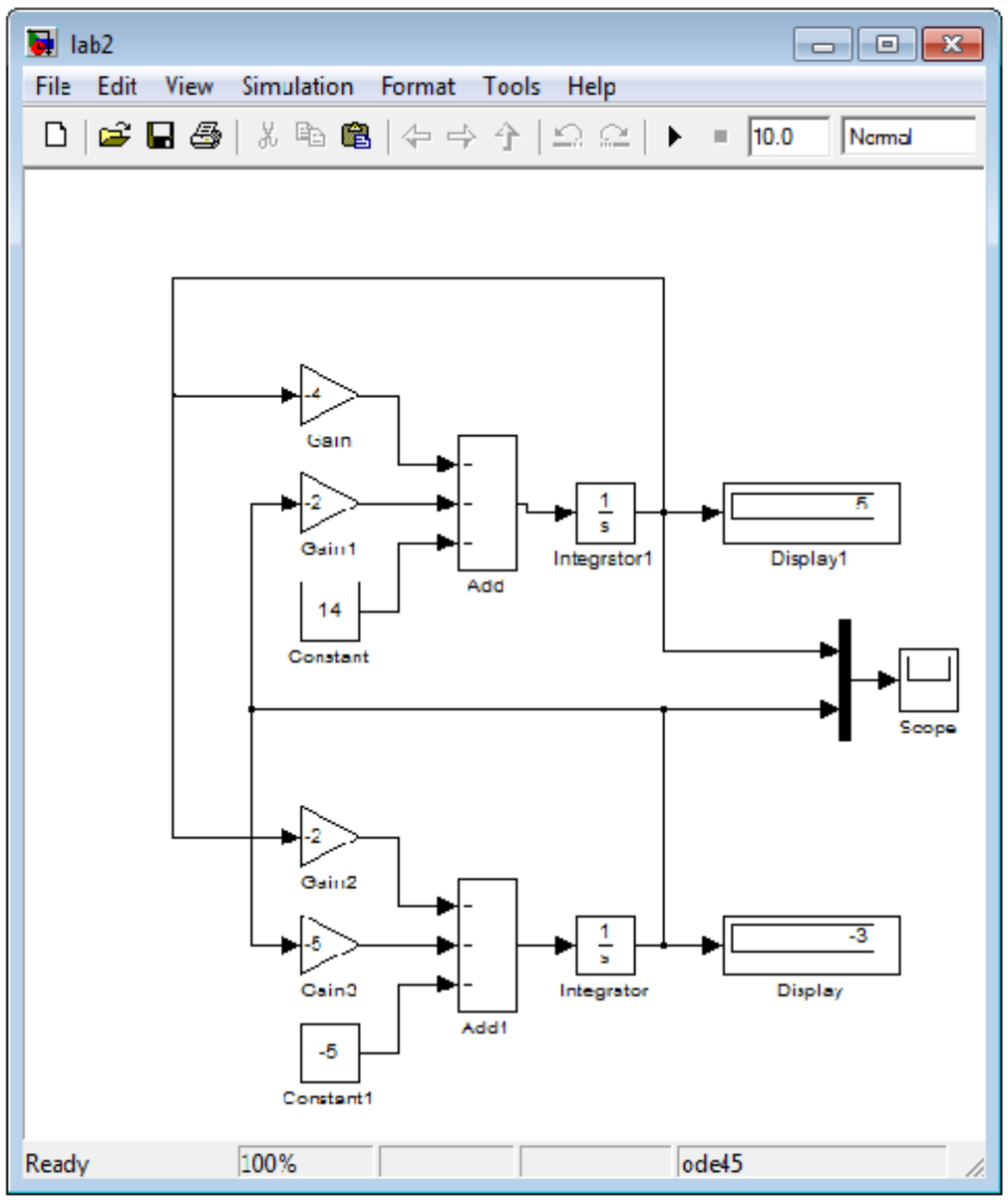


Рисунок 2.5 – Структурная схема модели системы дифференциальных уравнений 2. 6

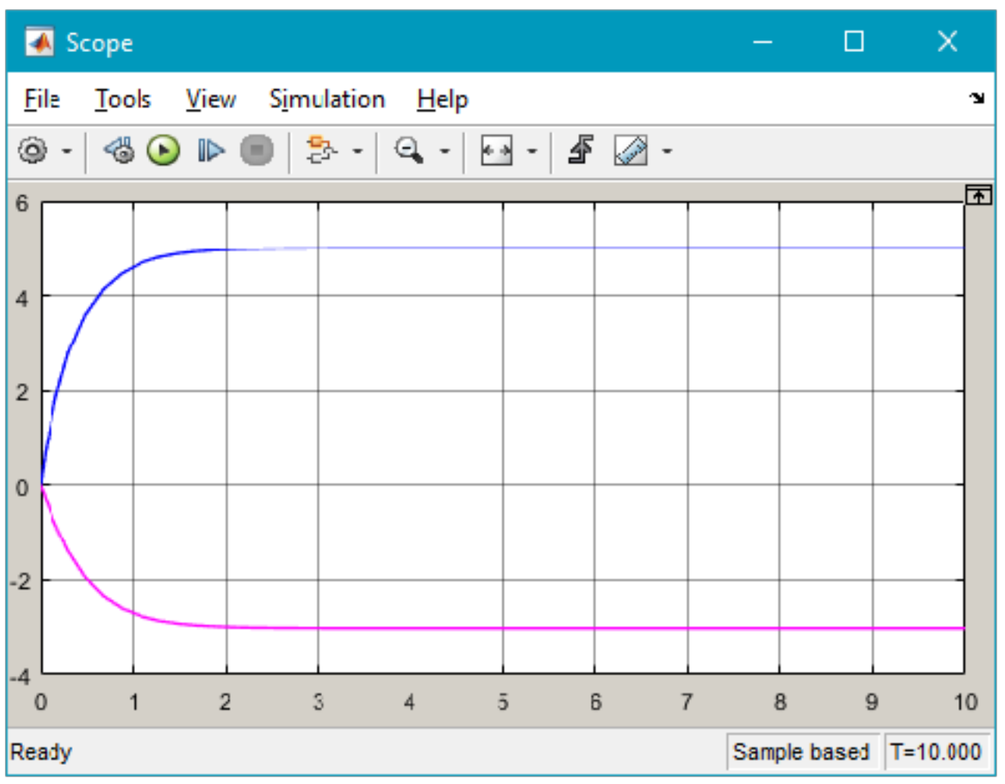


Рисунок 2.6 – Переходный процесс установления решения системы линейных алгебраических уравнений [2]

Аппаратура и материалы. Для выполнения лабораторной работы необходимо использовать следующее: аппаратное обеспечение: персональный компьютер и программное обеспечение: операционную систему Windows 10 и выше; Microsoft Office13 и выше, системы компьютерной математики Machcad и MATLAB R2017a и выше.

Указания по технике безопасности. Студенты должны следовать общепринятой технике безопасности для пользователей персональных компьютеров. Не следует самостоятельно производить ремонт технических средств, установку и удаление программного обеспечения. В случае обнаружения неисправностей необходимо сообщить об этом администратору компьютерного класса.

Методика и порядок выполнения работы

Выполните все примеры, приведенные в теоретической части и предложенные задания, предварительно ознакомившись с теоретической частью.

Задание 2.1. Построить модель решения задачи. Отряд туристов вышел в поход на нескольких байдарках ***L***, часть из которых ***A***-местные, а часть –***B***-местные. Сколько ***A***-местных и сколько ***B***-местных байдарок было в походе, если отряд состоит из ***M*** человек?

Переходный процесс установления решения выведите на экран виртуального осциллографа. Параметры сигналов приведены в таблице 2.1. Номер варианта соответствует номеру, под которым студент записан в списке группы.

Таблица 2.1– Параметры задачи

№ вар-та	A	B	L	M
1.	1	2	12	20
2.	1	3	7	17
3.	1	4	5	11
4.	1	5	8	16
5.	2	3	8	19
6.	2	4	10	37
7.	2	5	8	16
8.	3	4	8	26
9.	3	5	6	24
10.	4	5	7	33

Задание 2.2.Построить модель системы линейных алгебраических уравнений вида:

$$A x_1 + B x_2 + C x_3 = D$$
$$L x_1 + M x_2 + N x_3 = O$$
$$Q x_1 + R x_2 + S x_3 = T$$

Переходный процесс установления решения выведите на экран виртуального осциллографа. Параметры сигналов приведены в таблице 2.2. Номер варианта соответствует номеру, под которым студент записан в списке группы.

Таблица 2.2 – Параметры сигнала

№ вар-та	A	B	C	D	L	M	N	O	Q	R	S	T
1.	5	2	1	2.5	3	7	2	-1.5	4	0.5	5	11.5
2.	2	1.5	3.5	-16.25	2	3	4	-19.5	2	-0.5	5	-22.75
3.	0.5	-8	-9	-30.5	4	5	6	25.5	-1.5	2	3	8.5
4.	-1	2.5	9	-35.25	5	6	7	-22	1	2	-2.5	10
5.	3	-2	-7	-8.5	6	7	8	38	1	-3.5	6	32
6.	4.5	5.5	4	-3.25	7	8	9	-12	-4.5	5	1	38
7.	6.5	-5	-6	39	1	2	3	10.5	4	1	-5.5	7.25
8.	5	7.5	9.5	27.75	2	3	4	-4.5	1	-6.5	9	11.75
9.	-4	10	3	-67.5	3	4	5	15.5	-7.5	8	1	-90.5
10.	8	1	6	64.5	4	5	6	22.5	7	1	-8.5	70

Содержание отчета и его форма

Подготовьте отчет, в котором приведите технологию выполнения заданий.

Отчет по лабораторной работе должен содержать:

1) название работы;
2) цель лабораторной работы;
3) формулировку задания и технологию его выполнения;
4) ответы на контрольные вопросы;
5) приложение – файлы выполненных заданий.

Вопросы для защиты работы

1. Как с помощью Simulink можно моделировать объекты, описываемые системами алгебраических уравнений?
2. Какие основные математические блоки используются для моделирования объектов, описываемых системами алгебраических уравнений?
3. Что является достаточным условием, обеспечивающим затухающее решение системы дифференциальных уравнений?

ЛАБОРАТОРНАЯ РАБОТА 3
Моделирование с помощью нейронных сетей. Основы байесовских методов классификации

Цель и содержание работы: познакомить студента с основами байесовских методов классификации.

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций

Вопросы для обсуждения на лабораторном занятии: основные модели теории нейронных сетей, построение простейших моделей нейронных сетей в MATLAB.

Формируемые компетенции: ОПК-7.1 ОПК-7.2 ОПК-8.1 ОПК-8.2

Теоретическое обоснование

Изучите теоретическое обоснование, используя информацию представленную ниже и соответствующую литературу.

В настоящее время искусственные нейронные сети применяются для решения многих не формализуемых или трудно формализуемых задач, таких как:

Задача обработки сигналов и изображений.

При обработке изображения (сигнала) часто необходимо отделить полезную (информативную) компоненту сигнала от шума. Такие задачи получили название задач фильтрации. Задачи подобного рода важны, в частности, в полиграфии. Например, нужно выделить наиболее существенную для пользователя информацию и убрать все остальное. Это сложная задача теории искусственного интеллекта.

Задача распознавания образов. Распознавание речи.

Пусть, например, на конвейере движется продукция. Необходимо создать систему технического зрения, позволяющую автоматически различить бракованную и нормальную продукцию. Другой пример: курсы валют, акций обычно сильно колеблются. Эти колебания связаны с тем, что на рынке имеется много агентов, действующих, как правило, несогласованно. Трейдер решает, в частности, такую задачу — выделить среди этих колебаний так называемый основной тренд, чтобы понять, стоит ли вкладывать свои деньги в акции. Основной тренд представляет собой некую усредненную кривую, где мелкие колебания (шум) должны быть устранены в результате фильтрации. Еще одна популярная ныне и крайне важная задача о выделении основного тренда — это задача анализа климатических данных с целью выяснения, есть глобальное потепление или нет.

Еще несколько важных задач: – распознавание спама в Интернете; автоматический перевод, распознавание почерка и так далее.

Нейронные сети можно использовать при следующих условиях:

1. Если задачу может решить человек.

2. Если при решении задачи можно выделить множество входных факторов (сигналов, признаков, данных и т.п.) и множество выходных факторов.

3. Если изменения входных факторов приводит к изменению выходных.

В то же время применение нейронных сетей при решении некоторых задач может оказаться эффективнее использования разума человека. Это объясняется тем, что человеческий разум ориентирован на решение задач в трехмерном пространстве. Многомерные задачи для него характеризуются значительно большей трудоемкостью.

Искусственным нейронным сетям не свойственно такое ограничение. Им все равно решать трехмерную или 10-мерную задачу.

Подобно биологической нейронной системе ИНС является вычислительной системой с огромным числом параллельно функционирующих простых процессоров с множеством связей.

Модели ИНС в некоторой степени воспроизводят "организационные" принципы, свойственные мозгу человека. Моделирование биологической нейронной системы с использованием ИНС может также способствовать лучшему пониманию биологических функций. Такие технологии производства, как VLSI (сверхвысокий уровень интеграции) и оптические аппаратные средства, делают возможным подобное моделирование. [Богатиков, В.Н. Построение систем управления на основе нейронных сетей: Учебно–методическое пособие / В.Н. Богатиков, Л.В. Дранишников, А.Е. Пророков. - Апатиты: Изд-во КФ ПетрГУ, 2011. – 41 с]

Общие принципы функционирования нейронных сетей [Вакуленко С.А., Жихарева А.А. Практический курс по нейронным сетям – СПб: Университет ИТМО, 2018. – 71 с.]

В основе построения нейронных моделей лежит использование аналогии с мозгом и реальными нейронами. Мозг состоит из нейронов, нейроны связаны друг с другом и обмениваются сигналами. Сигналы имеют булевскую природу, а система связанных нейронов – это стохастическая динамическая система.

Таким образом, нейрон – это пороговая система, которая получает входные сигналы от других нейронов, суммирует их и генерирует выходной сигнал, если эта сумма превышает некий порог.

Идеализированная модель такой пороговой системы может быть построена при помощи сигмоидальных функций, например, функции:

$$¿\frac{1}{1+e^{ax}}, \tag{3.1}$$

где x – амплитуда входного сигнала, который получает нейрон от других нейронов; σ – выходной сигнал нейрона, представленной на рисунке 3.1, или функции Хевисайда:

$$¿\begin{cases}1, x>0\\0, x\le 0\end{cases}, \qquad (3.2)$$

представленной на рисунке 3.2, или кусочно-линейной функция $\sigma = \max(x,0)$.

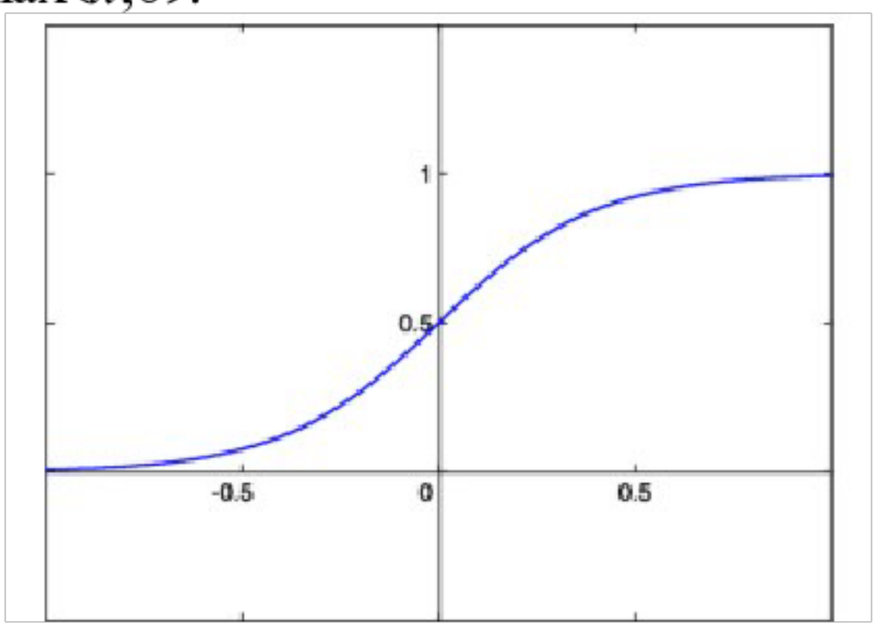


Рисунок 3.1 – График сигмоидальной функции $¿\dfrac{1}{1+e^{ax}}$

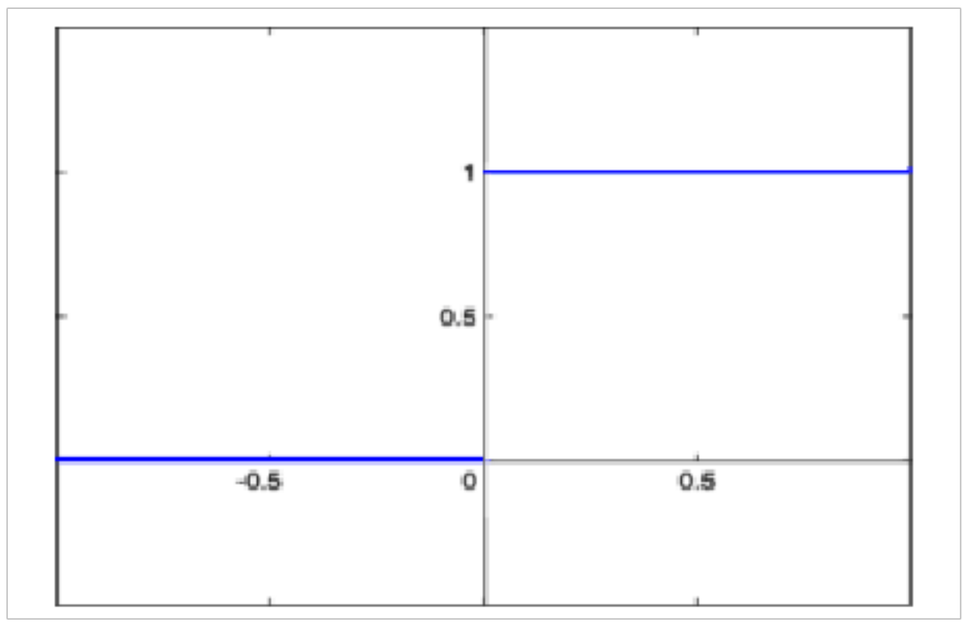


Рисунок 3.2 – График функции Хевисайда

Для решения задачи распознавания образов можно использовать рекуррентную нейросеть – многослойный персептрон. Силу связи между нейронами описывает матрица синаптических связей (W), в которой хранится вся долговременная фундаментальная информация. В процессе обучения мозга матрица синаптических связей медленно меняется со временем. Таким образом, можно разделить два динамических процесса, протекающих в мозге: быструю динамику нейронов, которая описывается динамической системой, и медленную, связанную с изменением силы связи между нейронами. Медленная динамика упрощенно описывается правилом Хебба (Дональд Хебб – канадский физиолог, экспериментально показавший, что если два нейрона одновременно активны, то сила связи между ними растет).

Рассмотрим общие принципы функционирования нейронных сетей на примере задачи классификации.

Предположим, мы хотим создать автоматическую систему, которая различает два типа объектов – A и B. Системы технического зрения позволяют записать данные об объектах (признаки объекта) в цифровом виде.

Предположим, что мы характеризуем объект с помощью набора признаков $(x_1, x_2, \ldots, x_n)$. Признаки могут быть выражены с помощью целых чисел или даже булевских переменных, то есть «есть признак» – «нет признака».

Совокупность признаков можно рассматривать как вектор с n компонентами, или точку k-мерного Эвклидова пространства $X=(x_1, x_2, \ldots, x_n)$. Тогда задача классификации сводится к следующей математической задаче: разделить два множества точек A и B n-мерного эвклидова пространства некоторой гиперповерхностью размерности n-1.

Сделаем некоторые важные комментарии. Выбор признаков для классификации является исключительно сложной задачей, которую мы

пока не рассматриваем. Ранее эта задача решалась вручную, в последнее время появились эффективные методы автоматического нахождения наиболее эффективных признаков (так называемый Deep Learning). Отметим, что выбор правильных признаков важен для успешности последующей обработки системы признаков методами, которые мы описываем ниже. Обучение нейронной сети в задачах классификации происходит на наборе обучающих примеров $X(1),X(2),\ldots,X(P)$, для которых принадлежность объекта к классу A или классу B известна. Чтобы математически формализовать этот факт, определим индикатор:

$$D\,X=\begin{cases}1, X\,A\\0, X\,B\end{cases}., \tag{3.3}$$

По накопленному в результате обучения «опыту» строим нейросеть (систему ИИ), которая проводит разделяющую поверхность.

Математически этот процесс может быть описан как поиск некоторой функции $y = F(X,W)$, где W – набор параметров нейронной сети (или другой системы ИИ). Для нейросетей эти параметры, в частности, задают силу связи между нейронами и подбираются так, чтобы ошибка обучения (*error training*) была бы минимальной (как можно ближе к нулю). В качестве ошибки обучения обычно рассматривают функцию

$$E_{train}(W)=\left|F\,X\,j\,W-D\,X\,j\right|, \tag{3.4}$$

где $X(j)$, $j\overline{1,P}$ берутся из обучающего множества.

Для проверки эффективности обучения нейронной сети берут тестовое множество объектов и вычисляют

$$E_{test}(W)=\left|F\,X\;j\,W-D\,X\,j\right|, \tag{3.5}$$

где $X(j)$ взяты из тестового множества.

После того, как система обучена (что иногда требует большого процессорного времени), она для любого поданного на вход системы объекта X автоматически решает, к какому классу он относится.

Рассмотрим основные модели теории нейронных сетей, выделив отдельно два разных класса сетей: ***Сети прямого распространения***: однослойный персептрон; многослойный персептрон; сети радиальных базисных функций (RBF);машины опорных векторов (SVM). ***Рекуррентные сети***: реккурентные сети; аттракторные нейронные сети и т.д.

Одна из первых моделей нейронной сети была предложена Ф. Розенблаттом в 50-е годы XX века и получила название однослойного персептрона. В этой модели входной сигнал проходил всего один слой нейронов и после этого формировался выходной сигнал при помощи сигмоидальной функции.

В дальнейшем выяснилось, что возможности таких простейших систем ограничены. Было предложено обобщение — модель многослойного персептрона, отличие которого от однослойного заключается в прохождении входного сигнала через несколько слоев.

Многослойные персептроны могут моделировать любую систему «вход-выход», то есть любой черный ящик. Однако они имеют ряд недостатков, в частности, выбор числа слоев, нейронов и связей между ними не так прост.

Такие системы, как сети радиальных базисных функций и машины опорных векторов, можно рассматривать как специальные модификации многослойных персептронов с целью преодоления недостатков последних.

Все перечисленные модели характеризуются наличием слоев, через которые проходит сигнал. В рекуррентных нейронных сетях невозможно выделить отдельные слои. Сигналы могут циркулировать по сети во всех направлениях, образуя сложную пространственно-временную структуру (*pattern*). Такие сети обладают колоссальными возможностями. Так, например, известно, что они могут моделировать любую динамическую систему или машину Тьюринга, то есть реализовать любой алгоритм.

Рассмотрим однослойный персептрон, который представляет собой простейшую модель нейронной сети. Однослойный персептрон получает на вход сигнал, заданный вектором X и на выходе выдает число $y = \sigma(S)$, где $Sw\sum_k w_k x_k h w_1 x_1 w_2 x_2 \ldots w_n x_n h$. Входной вектор X состоит из n компонент $X=(x_1, x_2, \ldots, x_n)$, каждая из которых представляет собой численную характеристику анализируемого нейронной сетью объекта. Через $w_i, i\overline{1,n}$ и h обозначены параметры персептрона — синаптические веса и порог (сдвиг) соответственно.

Синаптические веса указывают силу влияния конкретной характеристики на выходное значение. Значения этих параметров могут быть как положительными, так и отрицательными. Отметим, что переменные $x_i, i\overline{1,n}$ и могут быть булевскими, то есть принимать значения 0 или 1. Это означает, что объект обладает данным признаком, если $x_i = 1$, и не обладает, в случае, если $x_i = 0$. В этом случае функция $(S) = H S = \begin{cases} 1, S>0 \\ 0, S\le 0 \end{cases}$ – функция Хевисайда или функция скачка. На рисунок 3.3 приведена модель однослойного персептрона, цифрами 1-4 помечена последовательность действий при работе алгоритма.

Преимущество персептрона заключается в его простоте. Укажем и его недостаток — персептрон классифицирует только линейно разделимые объекты (то есть только такие множества векторов $X(t)$, между которыми можно провести разделяющую эти множества гиперплоскость).

Геометрическая интерпретация Однослойный персептрон работает как классификатор объектов, которые математически могут быть заданы как элементы двух разных множеств в многомерных линейных пространствах. Чтобы понять, как это происходит, рассмотрим случай, когда ситмоидальная функция есть функция скачка $\sigma = H(S)$. Предположим, требуется классифицировать объекты в два непересекающихся класса объектов A и B.

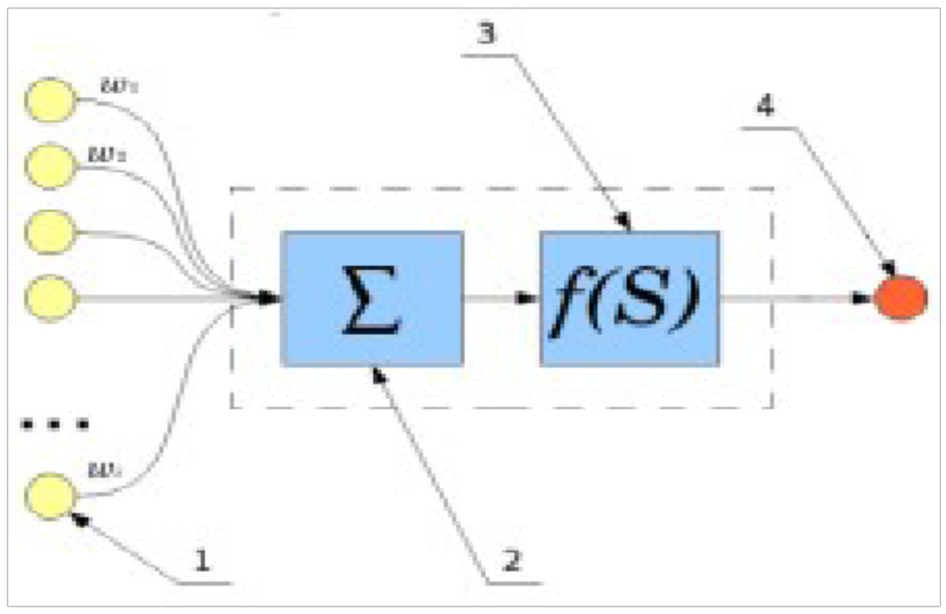


Рисунок 3.3 – Модель однослойного персептрона

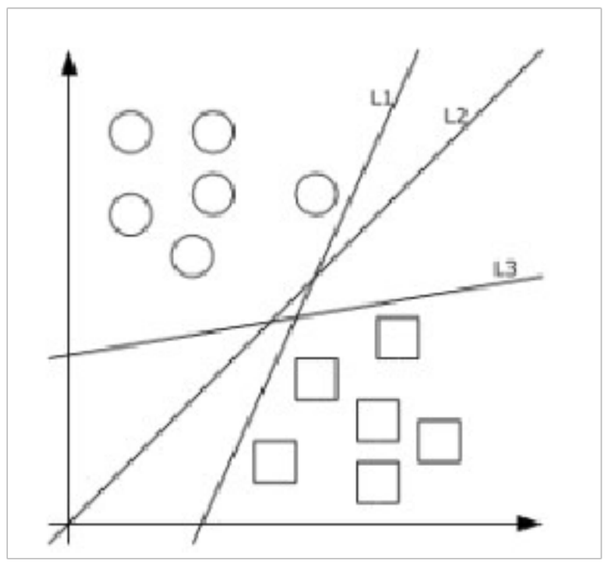


Рисунок 3.4. Геометрическая интерпретация однослойного персептрона

Для этого, определим выходные значения: положим $D = 1$, если объект принадлежит к классу A, $D = 0$, если объект принадлежит к классу B: $X = \begin{cases} 1, X A \\ 0, X B \end{cases}$. Тогда, из определения функции скачка $\sigma = H(S)$ следует, что объект лежит в классе A при условии $S > 0$ и объект лежит в классе B, если $S < 0$. Следовательно, уравнение $S = 0$ может быть рассмотрено как некий разделитель множеств объектов разных классов. С другой стороны, уравнение $S = 0$ равносильно уравнению $w_1 x_1 w_2 x_2 \ldots w_n x_n h = 0$, которое определяет гиперплоскость в n-мерном линейном пространстве, состоящем из векторов (строк) ( ) $X = (x_1, x_2, \ldots, x_n)$

Напомним, что в двумерном пространстве гиперплоскость — это прямая, а в трехмерном — обычная плоскость. Иллюстрация на рисунке 3.4 демонстрирует вышесказанное на примере игры в квадратики (класс A) и шарики (класс B) в случае n=2, то есть на плоскости. Прямые L_1, L_2, L_3 – примеры «разделителей» этих классов. Таким образом, однослойный персептрон есть линейный разделитель.

Моделирование логических функций однослойным персептроном Однослойный персептрон позволяет несложным образом реализовать ряд логических функций, таких как конъюнкция, дизъюнкция, отрицание, демократическое голосование.

Результат каждой из этих функций определяется значением двух булевских переменных x_1 и x_2.

Представим значения этих переменных как вектор входных значений $X = (x_1,x_2)$, а результат как значение функции скачка, то есть y= $H(S) =$ H($w\ x\ w\ x - h$) = = 1 1 + 2 2 - . Поясним сказанное на примере функции конъюнкции — булевской функции, определенной следующей таблицей:

X_1	0	1	0	1
X_2	0	0	1	1
$X_1 \wedge X_2$	0	0	0	1

Наша задача — представить эту логическую функцию с помощью однослойного персептрона. Рассмотрим вектор входных значений $X = (x_1,x_2)$ как точку плоскости, а результат логической операции – как форму

точки: квадрат, в случае, когда результат операции истинен ($y = 1$), и круг, если результат операции ложен ($y = 0$). Тогда наша задача становится эквивалентной такой задаче классификации: найти прямую, разделяющую два класса объектов — квадраты и круги.

Прямой, разделяющей эти множества объектов, является, например, прямая $x_1 + x_2 - 1.5 = 0$. Это означает, что синаптические веса $w_1 = 1$, $w_2 = 1$, а порог $h = 1.5$. Иллюстрация вышесказанного приведена на рисунке 3.5.

Однако оказывается, что иногда квадраты и круги неразделимы. Марвин Минский показал, что возможности персептрона ограничены, потому что он разделяет множества с помощью гиперплоскости (пример Марвин). В самом деле, есть такие явно различимые классы объектов, для которых прямая как их разделитель не подходит. Например, рассмотрим логическую функцию «исключающее ИЛИ» (*XOR*), определяемую формулой $XOR(x, y) = x + y \ (mod\ 2)$. Для лучшего понимания приведем таблицу значений функции *XOR* для различных значений переменных:

X_1	0	1	0	1
X_2	0	0	1	1
$XOR\ (X_1, X_2)$	0	1	1	0

Эта функция не реализуема однослойным персептроном, то есть не существует такой прямой, которая бы разделила эти два класса объектов. На рисунке 3.6 проиллюстрирован результат этой логической операции в виде квадратов, в случае, когда результат операции истинен ($y = 1$), и круга, если результат операции ложен ($y = 0$). Очевидно, что эти два множества (квадраты и круги) не разделимы никакой прямой.

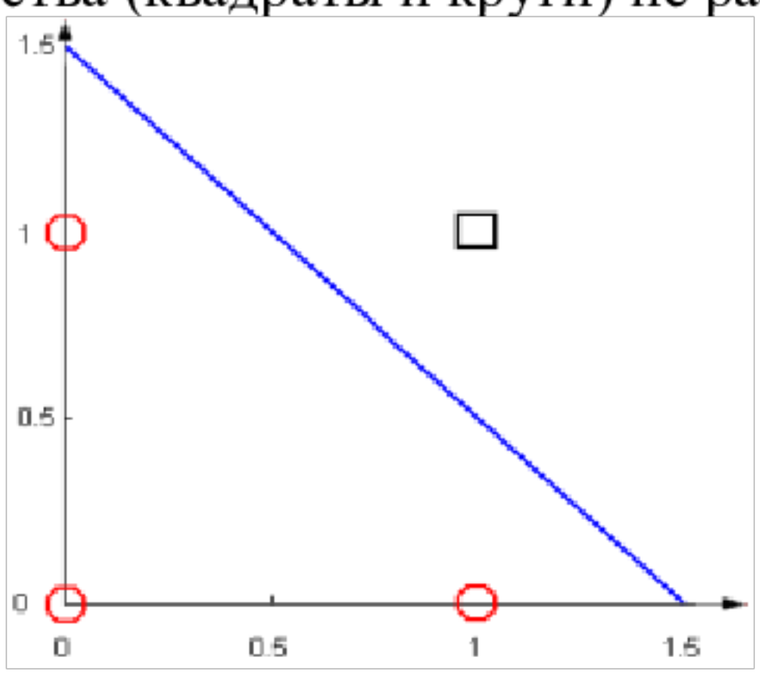


Рисунок 3.5 – Моделирование конъюнкции персептроном

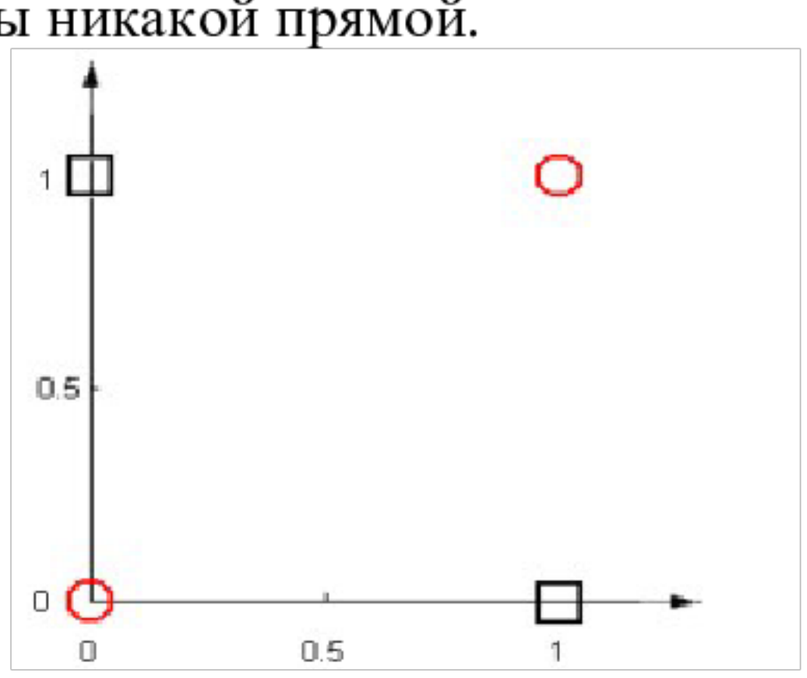


Рисунок 3.6 – Пример Минского однослойным

Алгоритм обучения однослойного персептрона

Имеются итерационные алгоритмы, которые находят параметры, если исходные объекты линейно разделимы.

Обучение нейронной сети в задачах классификации происходит на наборе обучающих примеров $X(1), X(2), \ldots, X(P)$, , в которых ответ – принадлежность к классу A или B, – известен. Определим индикатор D

следующим образом: положим $D(X) = 1$, если X из класса A, и положим $D(X) = 0$, если X из класса B, то есть

$$D(X)\begin{cases}1, X A,\\ 0, X B\end{cases} \quad (3.6)$$

где всякий вектор X состоит из n компонент: $X = (x_1, x_2, \ldots, x_n)$. Задача обучения персептрона состоит в нахождении таких параметров $w_1, w_2, \ldots, w_n$ и h , что на каждом обучающем примере персептрон выдавал бы правильный ответ, то есть

$$y X i D X i, i\overline{1, P} \ (2) \quad (3.7)$$

Интуитивно кажется очевидным, что если персептрон обучен на большом числе корректно подобранных примеров, и равенство (3.7) выполнено для почти всех $X i, i\overline{1, P}$, то в дальнейшем персептрон будет с близкой к единице вероятностью проводить правильную классификацию для остальных примеров. Этот интуитивно очевидный факт был впервые математически доказан (при некоторых предположениях) в основополагающей работе наших соотечественников В. Вапника и А. Червоненскиса еще в 1960-х годах. На практике, однако, оценки по теории Вапника–Червоненскиса иногда не очень удобны, особенно для сложных моделей нейронных сетей. Поэтому практически, чтобы оценить ошибку классификации, часто поступают следующим образом: множество обучающих примеров разбивают на два случайно выбранных подмножества, при этом обучение идет на одном множестве, а проверка обученного персептрона — на другом. Рассмотрим подробнее алгоритм обучения персептрона.

Шаг 1. Инициализация синаптических весов и смещения.

Значения всех синаптических весов модели полагают равными нулю: $w_i = 0, i\overline{1, n}$; смещение нейрона h устанавливают равным некоторому малому случайному числу. Ниже, из соображений удобства изложения и проведения операций будем пользоваться обозначением $w_0 = -h$.

Обозначим через $w_i(t), i=\overline{1, n}$вес связи от i-го элемента входного сигнала к нейрону в момент времени t.

Шаг 2. Предъявление сети нового входного и желаемого выходного сигналов.

Входной сигнал $X = (x_1, x_2, \ldots, x_n)$предъявляется нейрону вместе с желаемым выходным сигналом D.

Шаг 3. Адаптация (настройка) значений синаптических весов. Вычисление выходного сигнала нейрона.

Перенастройка (адаптация) синаптических весов проводится по следующей формуле: $w_i(t + 1) = w_i(t) + r \cdot (D(t) - y(t)) \cdot x_i(t)$, $i\overline{0, n1}$ где под $D(t)$ подразумевается индикатор, определенный равенством (3.6), а r – параметр обучения, принимающий значения меньшие 1.

Описанный выше алгоритм — это алгоритм градиентного спуска, который ищет параметры, чтобы минимизировать ошибку. Алгоритм

итеративный. Формула итераций выводится следующим образом. Введем риск

$$R(t)=\sum_{e=1}^{T}\left(D(t)-F(X(t),W(t))\right)^2, \qquad (3.8)$$

где суммирование идет по числу опытов (t – номер опыта), при этом задано максимальное число опытов – T. Подставим вместо F формулу для персептрона, вычислим градиент по w. В результате мы получим указанную выше формулу перенастройки весов.

Иллюстрация работы алгоритма

В процессе обучения вычисляется ошибка $(t)D(t)y(t)$. Построим график, показывающий, как меняется эта ошибка (рисунок 3.7) в ходе обучения сети и адаптации весов. На нем хорошо видно, что, начиная с некоторого шага, величина $\delta(t)$ равна нулю. Это означает, что персептрон обучен.

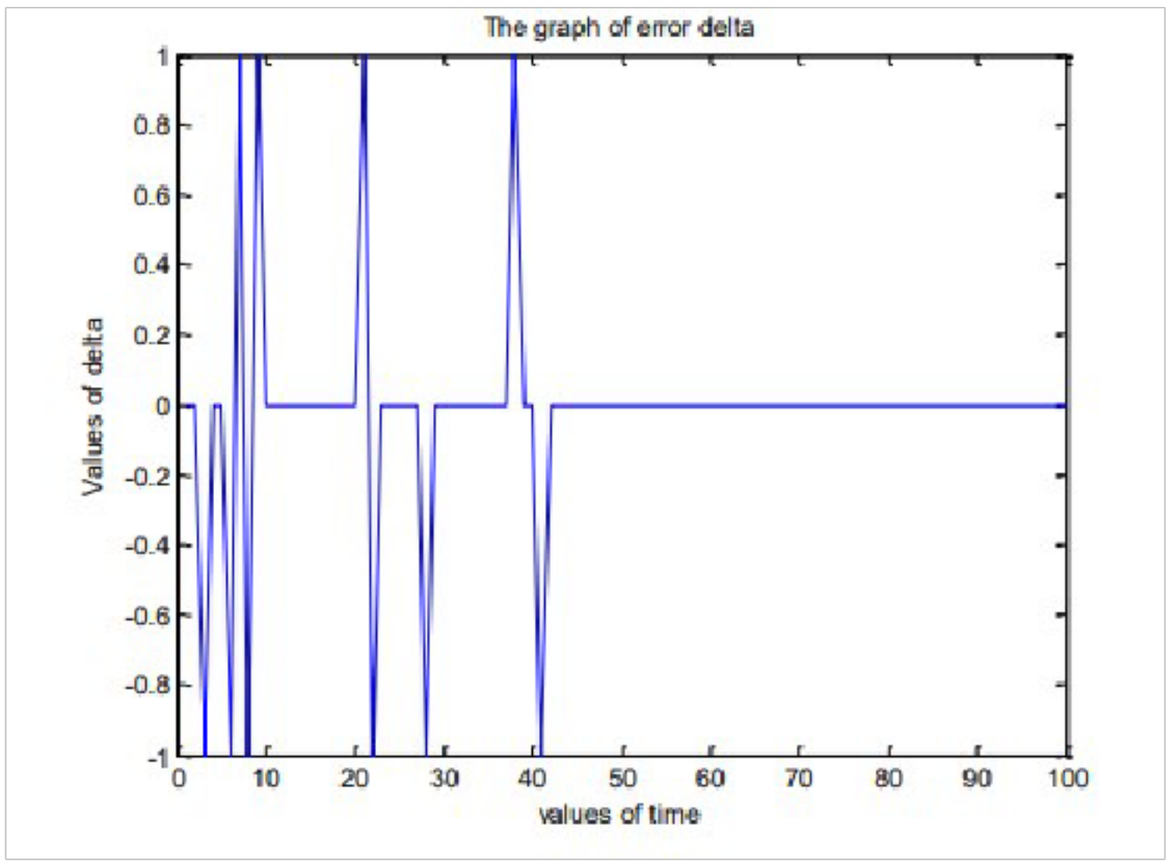


Рисунок 3.7 – График изменения ошибки в процессе обучения

Задача о моделировании черного ящика в общей постановке

Пусть известен реальный выход устройства $D(t)$ при входе $X(t)$, где $X(t)$ — вектор с компонентами $(x_1,x_2,\ldots,x_n)$, t – номер проводимого опыта $,t\overline{1,T}$ (максимальное число опытов T заранее определено). Необходимо найти параметры модели $w(w_0,w_1,\ldots,w_n)$, такие, что выход модели $F(X,w)$ и реальный выход устройства $D(t)$ были бы как можно ближе в среднеквадратичном смысле, то есть

$$R(t)=\sum_{e=1}^{T}\left(D(t)-F(X(t),W(t))\right)^2 \to min \qquad (3.8)$$

Функцию R(t) называют эмпирическим риском.

Аппаратура и материалы. Для выполнения лабораторной работы необходимо использовать следующее: аппаратное обеспечение: персональный компьютер и программное обеспечение: операционную систему Windows 10 и выше; Microsoft Office13 и выше, системы компьютерной математики Machcad и MATLAB R2017a и выше.

Указания по технике безопасности. Студенты должны следовать общепринятой технике безопасности для пользователей персональных компьютеров. Не следует самостоятельно производить ремонт технических средств, установку и удаление программного обеспечения. В случае обнаружения неисправностей необходимо сообщить об этом администратору компьютерного класса (обслуживающему персоналу лаборатории).

Методика и порядок выполнения работы

Выполните предложенные задания, предварительно рассмотрев и выполнив приведенные примеры.

Пример 1. Построение нейронной сети для операции конъюнкции в пакете Matlab

Построим нейронную сеть для логической операции конъюнкции с помощью пакета Matlab.

Таблица данной логической операции приведена выше. В данной реализации массив входов P задает значения X_1, X_2 для всех обучающих примеров и представляет собой первые две строки этой таблицы.

Целевой вектор T содержит значения индикаторной функции $D(X)$ для всех примеров X (в данном случае 4 примера) и представляет собой последнюю строку этой таблицы. На рисунке 3.8 представлен программный код и результат его выполнения в командном окне Matlab.

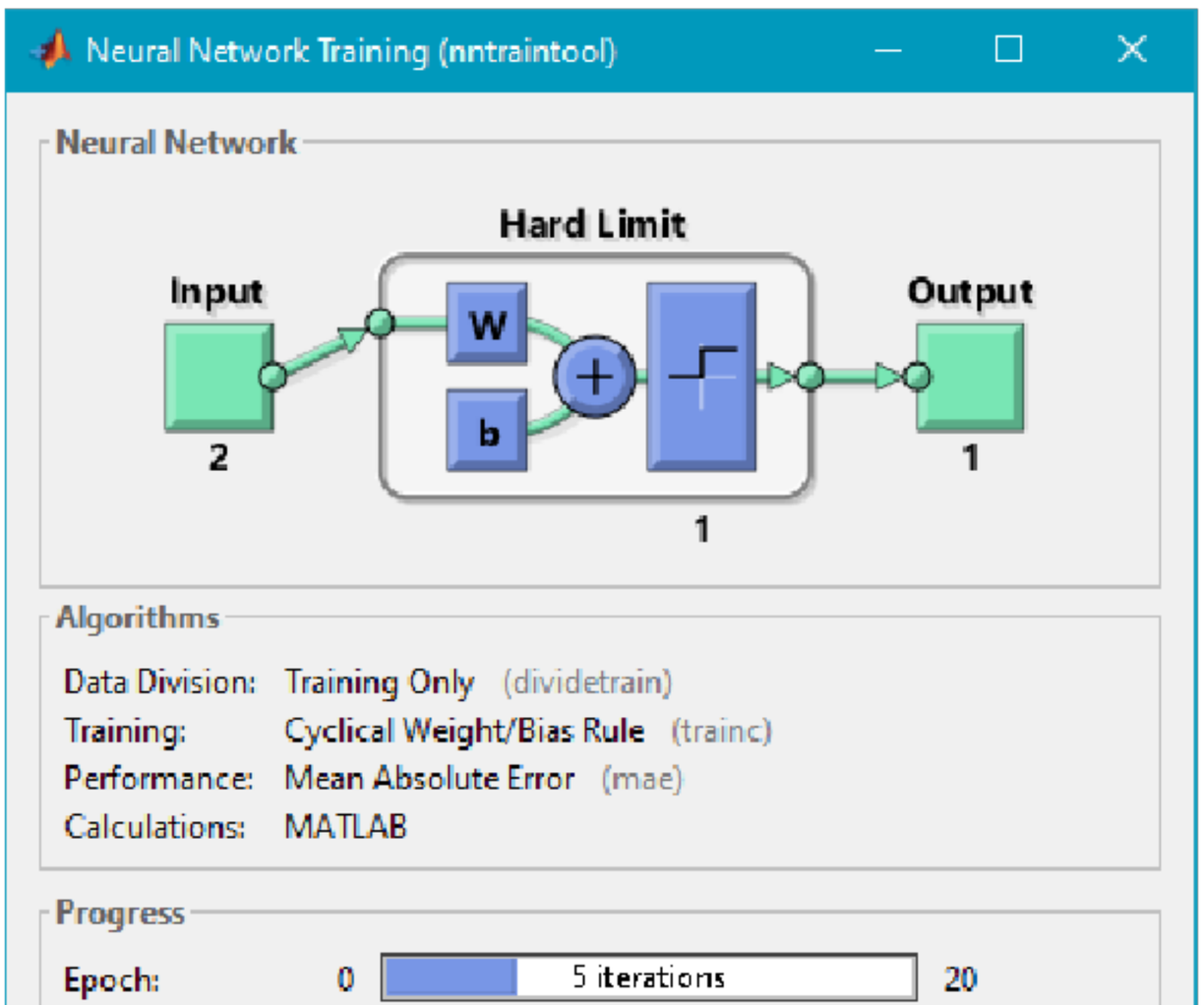
```
% аппроксимация конъюнкции однослойным персептроном
% построение функции y = H(w_1*X_1 + w_2*X_2 + w_0)
% (P,T) задают таблицу истинности
% зададим вектор входов
P = [ 0 1 0 1; 0 0 1 1 ];
% зададим вектор выходов
T = [ 0 0 0 1 ];
% встроенная функция построения сети с помощью персептрона,
% по умолчанию использована модель 'hardlim'
net = perceptron;
% определим максимальное число итераций для обучения сети
net.trainParam.epochs = 20;
% обучение сети
% в процессе обучения происходит коррекция весов w
net = train(net,P,T);
% симуляция
% вычисление значений аппроксимирующей функции в точках P
y = sim(net, P)
% синаптические веса после обучения сети
A = net.IW; celldisp(A)
% коэффициент сдвига b после обучения сети
w_0= net.b
```

```
Command Window

>> % аппроксимация конъюнкции однослойным персептроном
% построение функции y = H(w_1*X_1 + w_2*X_2 + w_0)
% (P,T) задают таблицу истинности
% зададим вектор входов
P = [ 0 1 0 1; 0 0 1 1 ];
% зададим вектор выходов
T = [ 0 0 0 1 ];
% встроенная функция построения сети с помощью персептрона,
% по умолчанию использована модель 'hardlim'
net = perceptron;
% определим максимальное число итераций для обучения сети
net.trainParam.epochs = 20;
% обучение сети
% в процессе обучения происходит коррекция весов w
net = train(net,P,T);
% симуляция
% вычисление значений аппроксимирующей функции в точках P
y = sim(net, P)
% синаптические веса после обучения сети
A = net.IW; celldisp(A)
% коэффициент сдвига b после обучения сети
w_0= net.b

y =

     0     0     0     1


A{1} =

     1     2



w_0 =

  cell

    [-3]
```

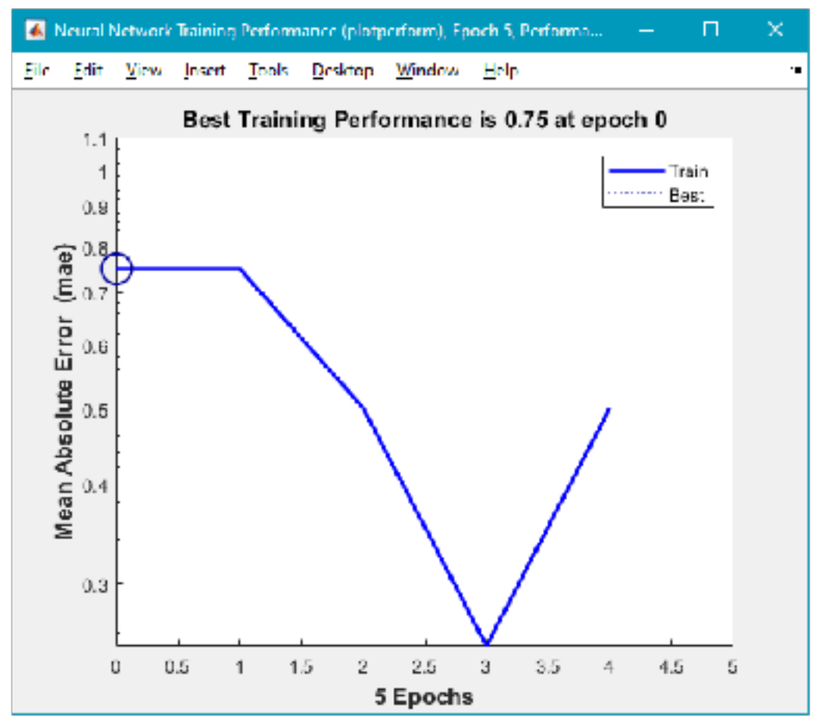
Рисунок 3.8 – Программный код и результат его работы в командном окне Matlab

В процессе работы программы Matlab отображает текущую информацию о модели и параметрах обучения в специальном окне Neural Network Training (рисунок 3.10). В частности, в этом окне показана структура нейронной сети — число входных нейронов (2), число уровней персептрона (1), число выходных нейронов (1), число итераций при обучении сети (в нашем случае — 5), время обучения сети.

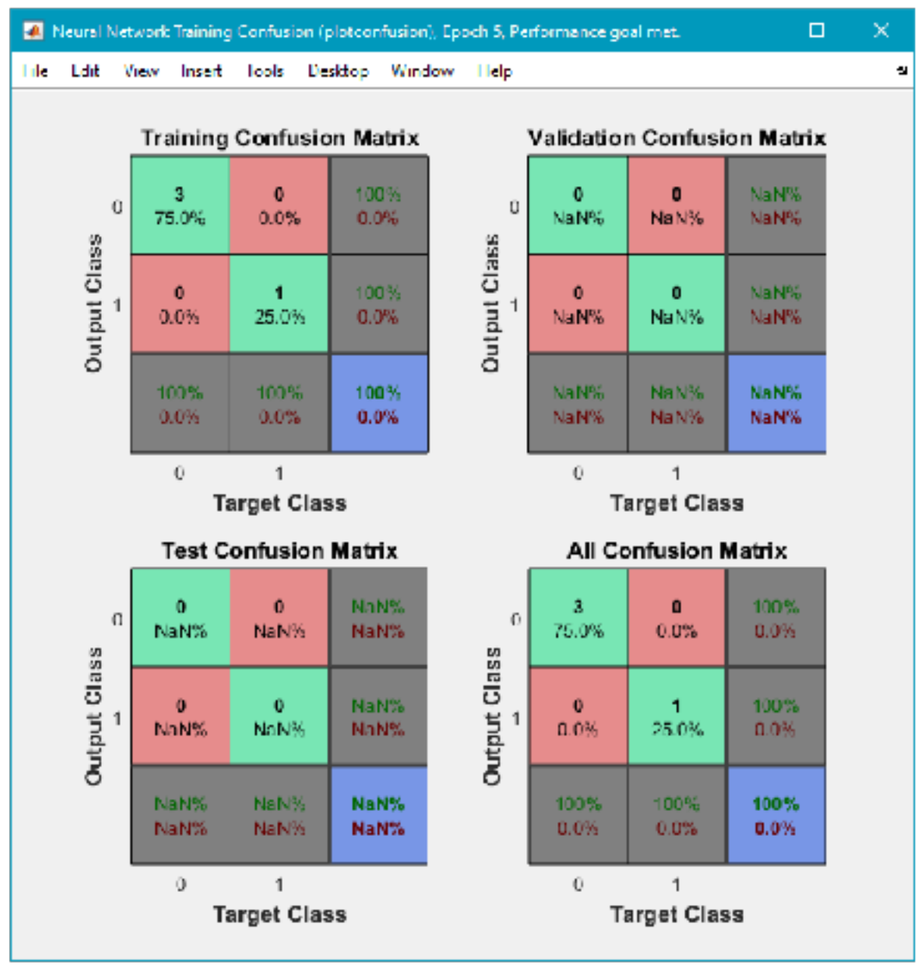
В то же время согласно инструкциям скрипта в окне команд будет выведена информация по результатам симуляции (вектор y) и значения весовых коэффициентов ($w_1 = 1$, $w_2 = 2$) и коэффициента сдвига ($w_0 = -3$):



а



б



с

Рисунок 3.10 – Нейронная сеть, реализующая конъюнкцию – а; настройка весов обучения – *б*; результат работы сети – с

Таким образом, получена модель персептрона, разделяющая классы объектов и определяемая уравнением $x_1 + 2x_2 - 3 = 0$, что геометрически будет означать разбиение двух множеств заданной прямой (рисунок 3.11).

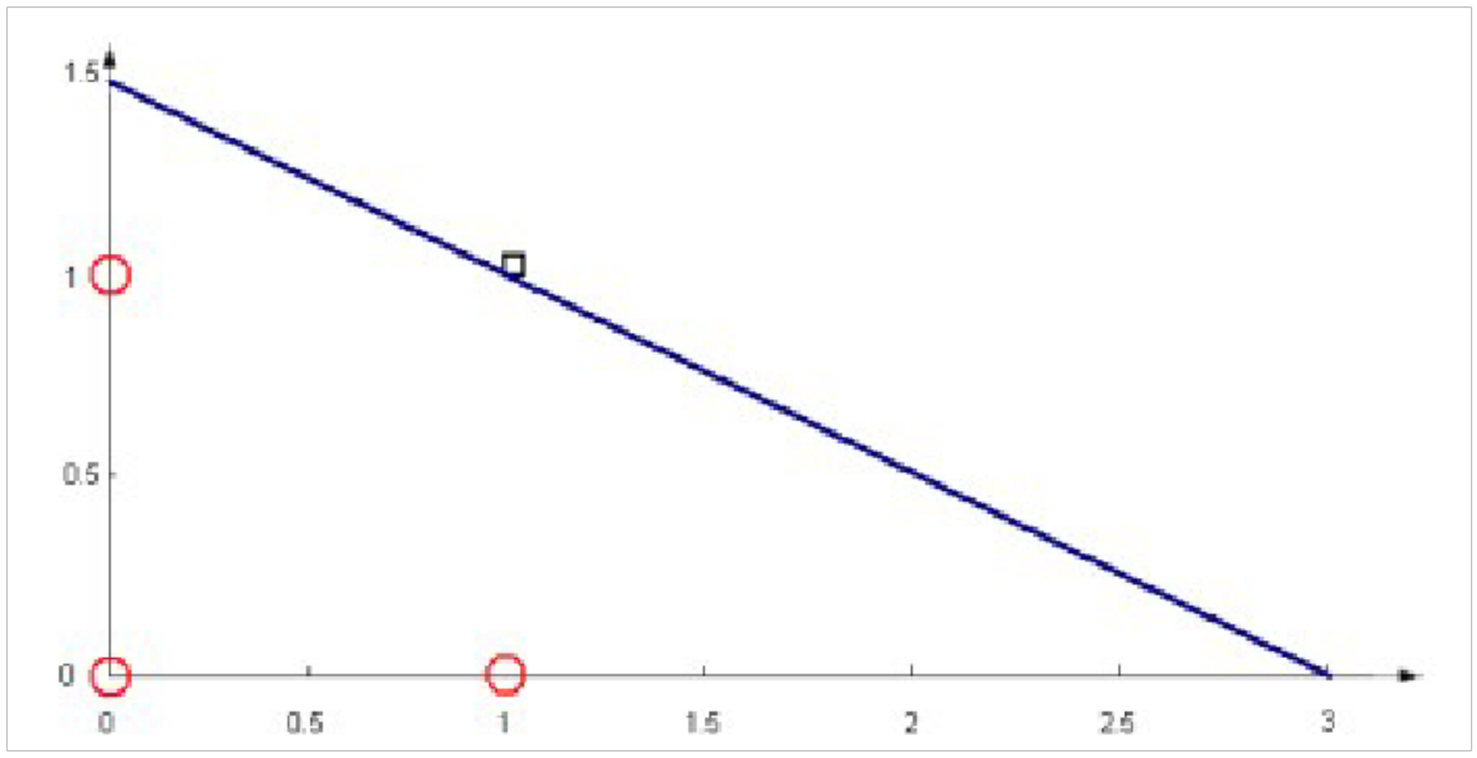


Рисунок 3.11– Моделирование конъюнкции однослойным персептроном в Matlab

Построение нейронной сети для классификации множеств точек плоскости в пакете Matlab

Пример 2. В качестве другого примера применения однослойного персептрона приведем пример классификации точек плоскости на два класса *A* и *B* – точки, находящиеся внутри эллипса, лежат в классе *A* и точки, находящихся вне эллипса, лежат в классе *B*. Эллипс определен общим уравнением кривой второго порядка вида:

$$a \cdot x^2 + b \cdot x \cdot y + c \cdot y^2 + d \cdot x + e \cdot y = 1 \qquad (3.9)$$

Ниже приведен программный код для эллипса, задаваемого уравнением:

$$1{,}5 \cdot x^2 + 2 \cdot x \cdot y + 1{,}1 \cdot y^2 + 2 \cdot x + 3 \cdot y = 1 \qquad (3.10)$$

Для наглядности приведены соответствующие построения на плоскости: исходная кривая, точки двух множеств, окрашенные соответственно в разные цвета.

Замечание. Этот пример показывает, что в исходном пространстве двух переменных однослойный персептрон не может разделить внешнюю и внутреннюю части эллипса. Однако можно сделать нелинейное преобразование данных, отобразив их в пространство высшей размерности. Здесь применяется так называемая теорема Ковера, которая утверждает, что после такого отображения данные могут стать линейно разделимыми.

Применим в нашем случае нелинейное отображение точки плоскости x, y в пространство размерности 5 следующим образом $: x, y = \left(a \cdot x^2 + b \cdot x \cdot y + c \cdot y^2 + d \cdot x + e \cdot y = 1 \right)$

```
% построение однослойного персептрона для классификации
% точек плоскости в соответствии с заданным эллипсом
% (внутри и вне эллипса (ax^2 + bxy + cy^2 + dx + ey = 1))
% Результат работы нейронной сети:
% вектор значений t:
% 0 - точка вне эллипса, 1 - точка внутри эллипса
N = 300; % число точек для обучения
% коэффициенты заданного эллипса
a = 1.5; b = 2; c = 1.1; d = -2; e = -3;
```

```
% построим график исходного эллипса
syms X Y;
f = a*X.^2 + b*X.*Y + c*Y.^2 + d*X + e*Y-1;
ellips = ezplot(f); set (ellips,'Color','b'); hold on
% сгенерируем исходные данные для обучения:
% точки плоскости (x,y), чьи координаты имеют
% нормальное распределение на интервалах [-2a;2a] и [-2c;2c]
x = randn(1,N)*(2*a) - d/(2*a); y = randn(1,N)*(2*c) - e/(2*c);
% определим отображение
% (x1,x2)->(a*x1^2,b*x2^2,c*x1*x2,d*x1,e*x2)
% матрица входов
P = [a*x.^2; b*x.*y; c*y.^2; d*x; e*y];
% сформируем вектор выходов T, приписывая значения 0 и 1
% для точек, лежащих вне и внутри эллипса соответственно
T = ones(1,N);
k = find( sum(P) >= 1);
T(k) = 0;
% функция построения сети с помощью персептрона,
% по умолчанию использована модель 'hardlim'
net = newp(P,T);
% определим максимальное число итераций для обучения сети
net.trainParam.epochs = 50;
% функция обучения сети; при этом происходит коррекция весов W
net = train(net,P,T);
% симуляция
% вычисление значений аппроксимирующей функции в точках P
t = sim(net, P)
% синаптические веса после обучения сети
A = net.IW;
celldisp(A)
% коэффициент сдвига b после обучения сети
b = net.b
% геометрические построения
% распознавание точек плоскости после обучения сети
k = find( t == 0); % индексы точек вне эллипса
x_out = x(k); y_out = y(k);
% построение точек вне эллипса
plot(x_out, y_out, '*r');
clear k;
k = find( t == 1); % индексы точек внутри эллипса
x_out = x(k); y_out = y(k);
% построение точек внутри эллипса
plot (x_out, y_out, '*g')
```

В процессе работы программа отображает текущую информацию о модели и параметрах обучения в специальном окне Neural Network Training (рисунок 3.12). В частности, в этом окне показана структура нейронной сети — число входных нейронов (5), число слоев персептрона (1) и используемые модели слоев, число выходных нейронов (1), число итераций при обучении сети (в нашем случае — 50), время обучения сети

(в нашем случае – 4 с). В окне команд будут выведены значения весовых коэффициентов
(w_1= –122.6519, w_2= –132.2919, w_3= –137.2794, w_4= – 121.0269, w_5= –127.4035) и коэффициента сдвига (w_0=151): A{1} –122.6519 –132.2919 –137.2794 –121.0269 –127.4035, b = [108], см. рисунок:

```
A{1} =

 -122.6519 -132.2919 -137.2794 -121.0269 -127.4035



b =

  cell

    [108]
```

а

б

Рисунок 3.12. Однослойный персептрон

Рисунок 3.13. Классификация точек однослойным персептроном (*а*), график настройки весовых коэффициентов персептрона(*б*)

Таким образом, построенная модель имеет следующий вид (у всех весовых коэффициентов указано только два десятичных знака):

$$y = H(S) = H(w_1 x_1 + w_2 x_2 + \ldots + w_5 x_5 + w_0) = H - 122.65\,x_1 - 132.29\,x_2 - 137.28\,x_3 - 121.03\,x_4 - 127.40\,x_5 + 10$$

Результат классификации множества точек после обучения персептрона представлен графически на рисунке 3.13 б; точки «внутри» и «вне» эллипса изображены зеленым и красным цветом соответственно.

Рассмотрим байесовские методы классификации. Байесовский подход является классическим в теории распознавания образов и лежит в основе многих методов. Он опирается на теорему о том, что если плотности распределения классов известны, то алгоритм классификации, имеющий минимальную вероятность ошибок, можно выписать в явном виде. Для оценивания плотностей классов по выборке применяются различные подходы.

Рассмотрим пример параметрического подхода, но сначала определим вероятностную постановку задачи классификации.

Пусть X – множество объектов, Y – конечное множество имён классов, множестве X×Y является вероятностным пространством с плотностью распределения p(x, y) = P(y)p(x|y). Вероятности появления объектов каждого из классов P_y = P(y) называются априорными вероятностями классов. Плотности распределения $p_y(x)$ = p(x|y) называются функциями правдоподобия классов. Вероятностная постановка задачи классификации разделяется на две независимые подзадачи.

1. Имеется простая выборка $X^l = (x_i, y_i)_{i=1}^l$ (l – размер выборки) из неизвестного распределения p(x, y) = $P_y p_y(x)$. Требуется построить эмпирические оценки априорных вероятностей $\hat{P}_y$ и функций правдоподобия $\hat{p}_y(x)$ для каждого из классов $y \in Y$.

2. По известным плотностям распределения $p_y(x)$ и априорным вероятностям P_y всех классов $y \in Y$ построить алгоритм a(x), минимизирующий вероятность ошибочной классификации.

Первая задача имеет множество решений, поскольку многие распределения p(x, y) могли бы породить одну и ту же выборку X^l. Приходится привлекать различные предположения о плотностях, что и приводит к большому разнообразию байесовских методов.

В параметрическом подходе предполагается, что плотность распределения выборки известна.

Нормальный дискриминантный анализ – это специальный случай байесовской классификации, когда предполагается, что плотности всех классов $p_y(x)$, $y \in Y$ являются многомерными нормальными.

Приведём формулу *n*-мерного (гауссова) распределения:

$$p(x) = \frac{1}{(2\pi)^{n/2}|S|^{1/2}} \exp\left(-\frac{1}{2}(x-m)^T S^{-1}(x-m)\right), \quad (3.11)$$

где *n* – количество числовых признаков, $m = E[x]$ – математическое ожидание (центр), *S* – ковариационная матрица ($S = E[(x-m)(x-m)^T]$), $|S|$ – детерминант *S*.

Пример 3. Вычислим, используя Matlab, значение гауссова распределения для $x_1 = [0.2,\ 1.3]^T$ и $x_2 = [2.2,\ -1.3]^T$, где $m = [0,\ 1]^T$, а $S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

Функцию нужно создать в отдельном файле Matlab (New → Function). Таким образом будет два файла Matlab, которые могут располагаться в одной папке. Чтобы Matlab понимал откуда ему загружать необходимую функцию, необходимо в окне Current Folder нажать правой кнопкой мыши на нужную папку, чтобы вызвать контекстное меню) и выбрать (Add to Path → Selected Folders and Subfolders), рисунок 3.14.

```
% Готовим Matlab к работе
close('all');
clear;
m=[0 1]'; S=eye(2);
x1=[0.2 1.3]'; x2=[2.2 -1.3]';
pg1=comp_gauss_dens_val(m,S,x1
)
pg2=comp_gauss_dens_val(m,S,x2
)
```

Где comp_gauss_dens_val (·) определим как

```
function
[z]=comp_gauss_dens_val(m,S,x)
[l,c]=size(m);
z=(1/(
(2*pi)^(l/2)*det(S)^0.5)      )*exp(-
0.5*(x-m)'*inv(S)*(x-m));
```

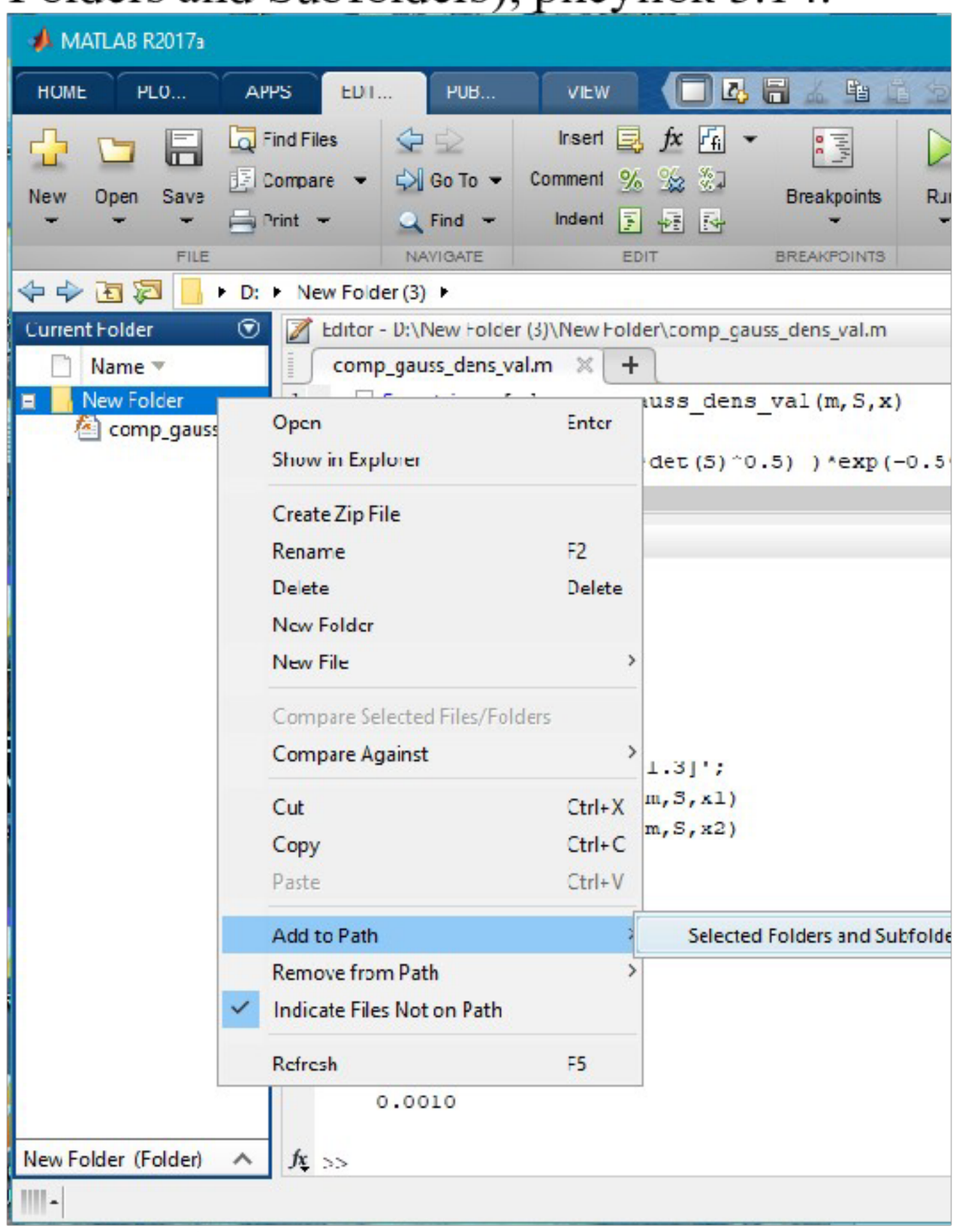


Рисунок 3.14 – Установка папки, откуда Matlab будет брать необходимые ему файлы

Функция **eye()** – создаёт единичную матрицу, запятая после квадратных скобок означает транспонирование вектора (**[]'**). Для получения справки по неизвестному объекту языка matlab, достаточно установить на него курсор и нажать F1 или написать команду help с именем объекта, допустим, **help eye**.

В результате выполнения кода получится, что pg1 = 0.1491, pg2 = 0.001.

На основании этого кода можно написать код байесовского классификатора, который определит к какому из двух классов относится входной вектор.

Например, пусть есть два класса w_1 и w_2, имеющих гауссовы

распределения $N(m_1, S_1)$, $N(m_2, S_2)$. $m_1 = [1, 1]^T$, $m_2 = [3, 3]^T$, $S_1 = S_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Примем во внимание, что вероятность появления первого и второго классов подсчитаны заранее и равны $P(w_1) = P(w_2) = ½$. Требуется узнать к какому классу принадлежит вход $x = [1.8, 1.8]^T$.

Пример 4. Напишем код байесовского классификатора, который решит эту задачу:

```
close('all');
clear;
 P1=0.5;
P2=0.5;
m1=[1 1]';
m2=[3 3]';
S=eye(2);
x=[1.8 1.8]';
p1=P1*comp_gauss_dens_val(m1,S,x)
p2=P2*comp_gauss_dens_val(m2,S,x)
```

```
>> close('all');
clear;
 P1=0.5;
P2=0.5;
m1=[1 1]';
m2=[3 3]';
S=eye(2);
x=[1.8 1.8]';
p1=P1*comp_gauss_dens_val(m1,S,x)
p2=P2*comp_gauss_dens_val(m2,S,x)

p1 =

    0.0420


p2 =

    0.0189
```

Получается, что $p_1 = 0.042$, $p_2 = 0.0189$, т.е. $p_1 > p_2$, соответственно двумерный вектор принадлежит первому классу.

Далее, проведём геометрический анализ нормальной плотности, её зависимость от матрицы ковариации.

Пример 5. Рассмотрим код, который генерирует 500 двумерных точек, распределенных в соответствие с двумерным нормальным распределением с центром $m = [0, 0]^T$ и матрицей ковариации $S = \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{bmatrix}$, где существует 8 типов этой матрицы:

1. $\sigma_1^2 = \sigma_2^2 = 1,\ \sigma_{12} = 0$;
2. $\sigma_1^2 = \sigma_2^2 = 0.2,\ \sigma_{12} = 0$
3. $\sigma_1^2 = \sigma_2^2 = 2,\ \sigma_{12} = 0$
4. $\sigma_1^2 = 0.2,\ \sigma_2^2 = 2,\ \sigma_{12} = 0$
5. $\sigma_1^2 = 2,\ \sigma_2^2 = 0.2,\ \sigma_{12} = 0$
6. $\sigma_1^2 = \sigma_2^2 = 1,\ \sigma_{12} = 0.5$;
7. $\sigma_1^2 = 0.3,\ \sigma_2^2 = 2,\ \sigma_{12} = 0.5$
8. $\sigma_1^2 = 0.3,\ \sigma_2^2 = 2,\ \sigma_{12} = -0.5$

```
close('all');
clear;
% Генерируем точки для первого случая
randn('seed',0); % используем старый вариант вызова генератора
% случайных чисел, слово 'seed' заменяет ряд параметров для такого
генератора
m=[0 0]'; % центр
S=[1 0;0 1]; % матрица ковариации
N=500; % количество точек
```

```
X = mvnrnd(m,S,N)'; % стандартная функция по генерации многомерного
% нормального случайного распределения
% Рисуем точки для первого варианта
figure(1), plot(X(1,:),X(2,:),'.');
figure(1), axis equal % устанавливаем тип стиля для осей
figure(1), axis([-7 7 -7 7])
% Рисуем точки для второго варианта
m=[0 0]';
S=[0.2 0;0 0.2];
N=500;
X = mvnrnd(m,S,N)';
figure(2), plot(X(1,:),X(2,:),'.');
figure(2), axis equal
figure(2), axis([-7 7 -7 7])
% Рисуем точки для третьего варианта
m=[0 0]';
S=[2 0;0 2];
N=500;
X = mvnrnd(m,S,N)';
figure(3), plot(X(1,:),X(2,:),'.');
figure(3), axis equal
figure(3), axis([-7 7 -7 7])
% Рисуем точки для четвертого варианта
m=[0 0]';
S=[0.2 0;0 2];
N=500;
X = mvnrnd(m,S,N)';
figure(4), plot(X(1,:),X(2,:),'.');
figure(4), axis equal
figure(4), axis([-7 7 -7 7])
% Рисуем точки для пятого варианта
m=[0 0]';
S=[2 0;0 0.2];
N=500;
X = mvnrnd(m,S,N)';
figure(5), plot(X(1,:),X(2,:),'.');
figure(5), axis equal
figure(5), axis([-7 7 -7 7])
% Рисуем точки для шестого варианта
m=[0 0]';
S=[1 0.5;0.5 1];
N=500;
X = mvnrnd(m,S,N)';
figure(6), plot(X(1,:),X(2,:),'.');
figure(6), axis equal
figure(6), axis([-7 7 -7 7])
% Рисуем точки для седьмого варианта
m=[0 0]';
S=[3 0.5;0.5 2];
N=500;
X = mvnrnd(m,S,N)';
```

```
figure(7), plot(X(1,:),X(2,:),'.');
figure(7), axis equal
figure(7), axis([-7 7 -7 7])
% Рисуем точки для восьмого варианта
m=[0 0]';
S=[.3 -0.5;-0.5 2];
N=500;
X = mvnrnd(m,S,N)';
figure(8), plot(X(1,:),X(2,:),'.');
figure(8), axis equal
figure(8), axis([-7 7 -7 7])
```

На рисунках 3/15 – 3.22 показано сгенерированное множество точек

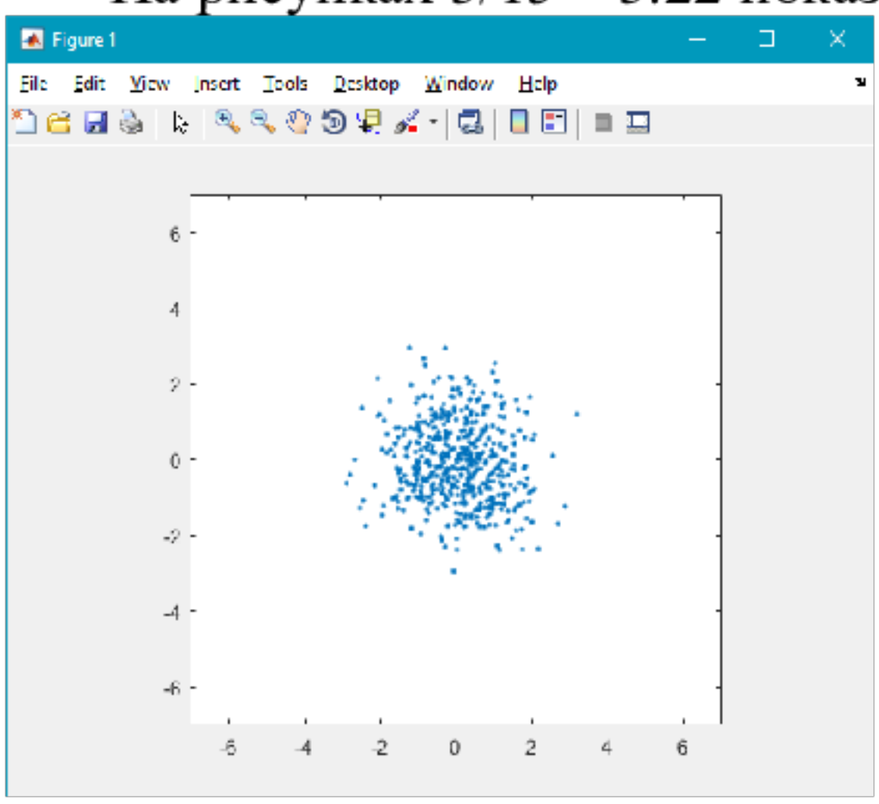


Рисунок 3.15 – Сгенерированное множество точек для случая $\sigma_1^2 = \sigma_2^2 = 1,\ \sigma_{12} = 0$

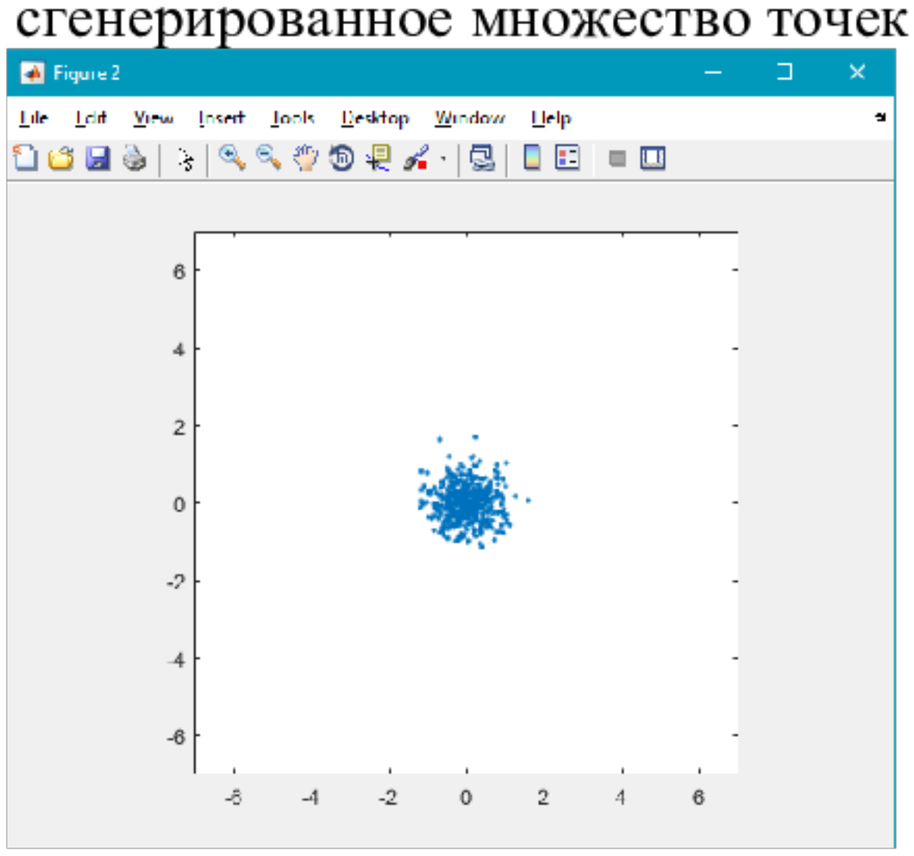


Рисунок 3.16 – Сгенерированное множество точек для случая $\sigma_1^2 = \sigma_2^2 = 0.2,\ \sigma_{12} = 0$

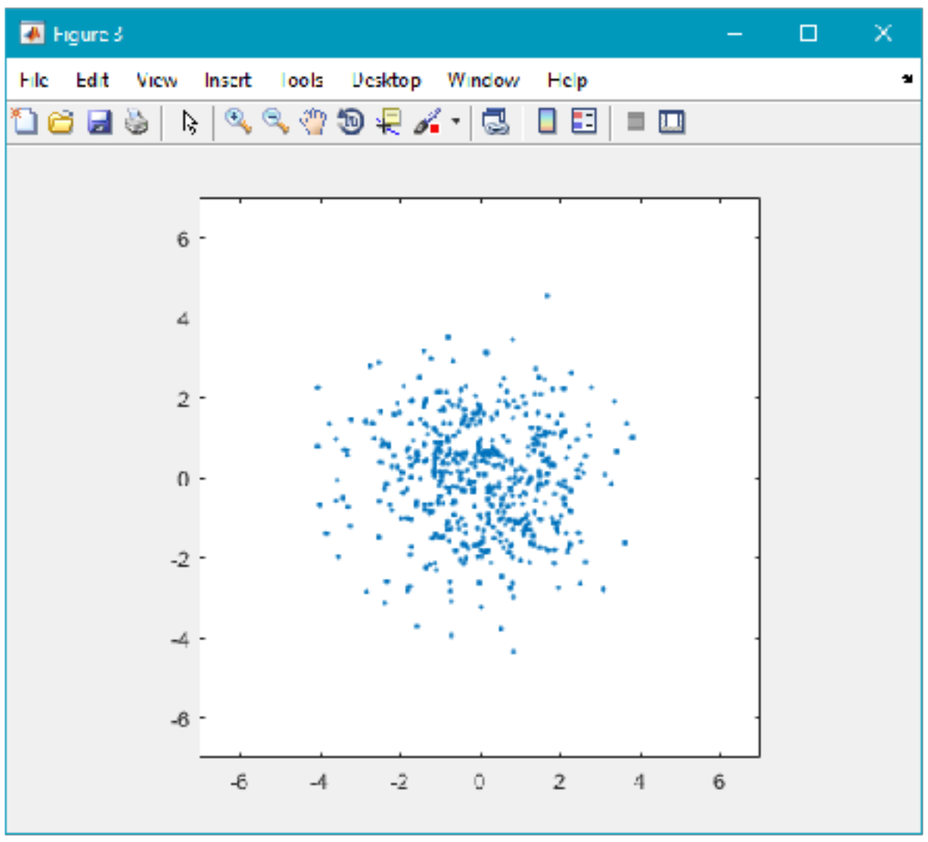


Рисунок 3.17 – Сгенерированное множество точек для случая $\sigma_1^2 = \sigma_2^2 = 2,\ \sigma_{12} = 0$

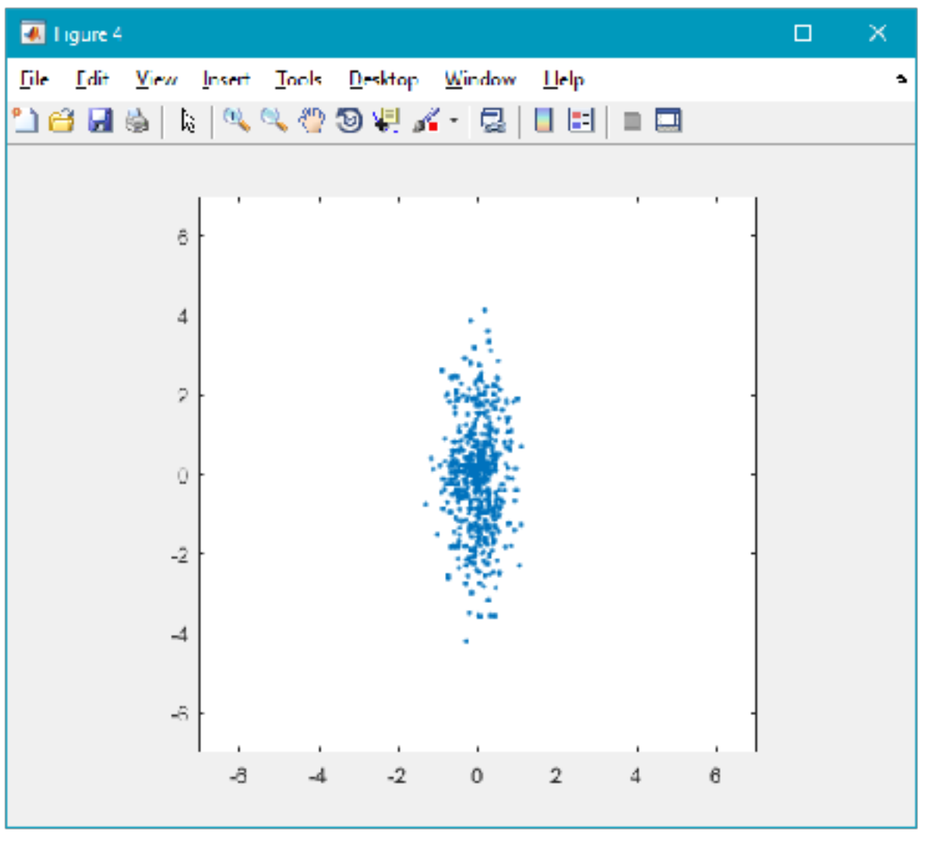


Рисунок 3.18 – Сгенерированное множество точек для случая $\sigma_1^2 = 0.2,\ \sigma_2^2 = 2,\ \sigma_{12} = 0$

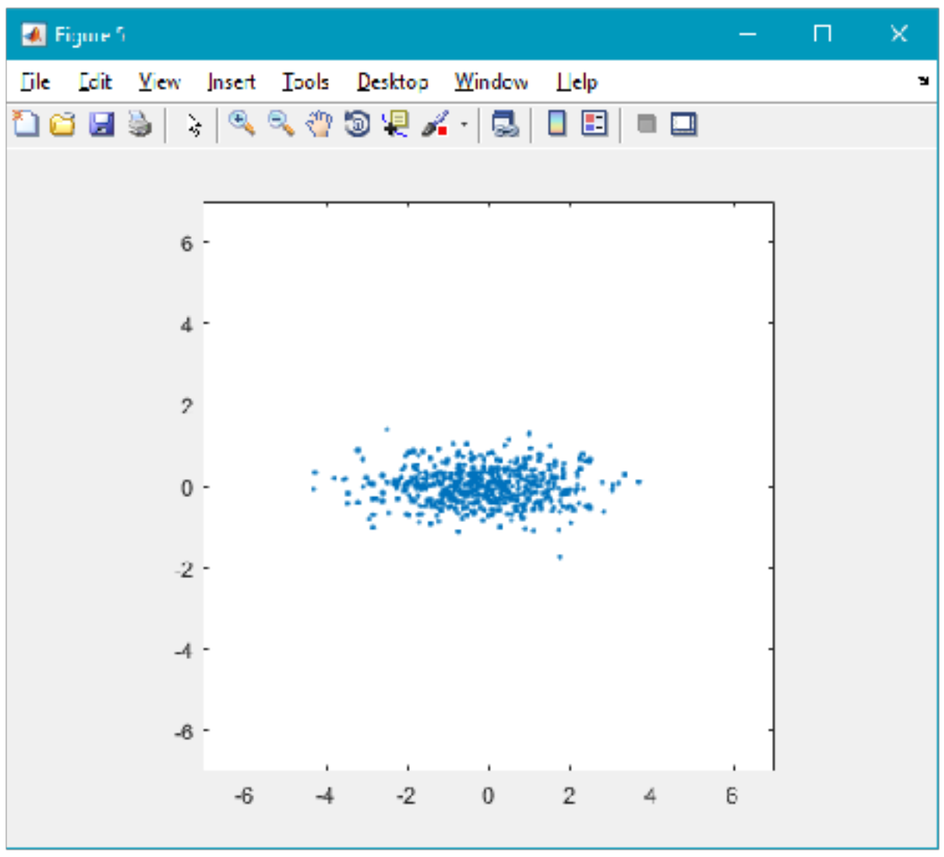


Рисунок 3.19 – Сгенерированное множество точек для случая $\sigma_1^2 = 2,\ \sigma_2^2 = 0.2,\ \sigma_{12} = 0$

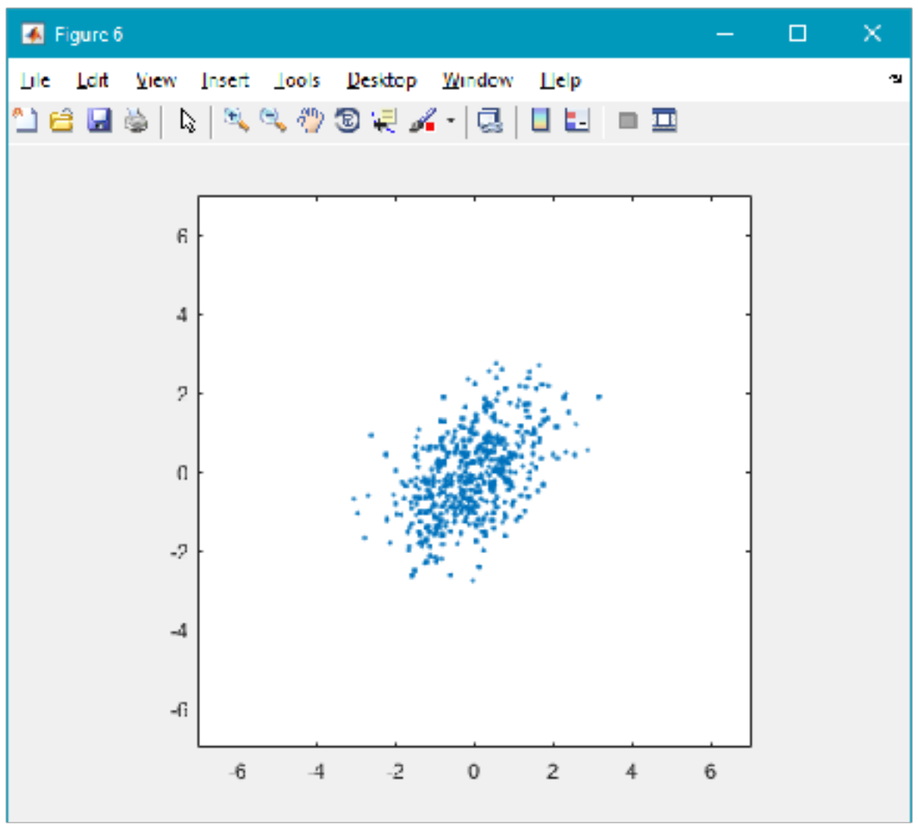


Рисунок 3.20– Сгенерированное множество точек для случая $\sigma_1^2 = \sigma_2^2 = 1,\ \sigma_{12} = 0.5$

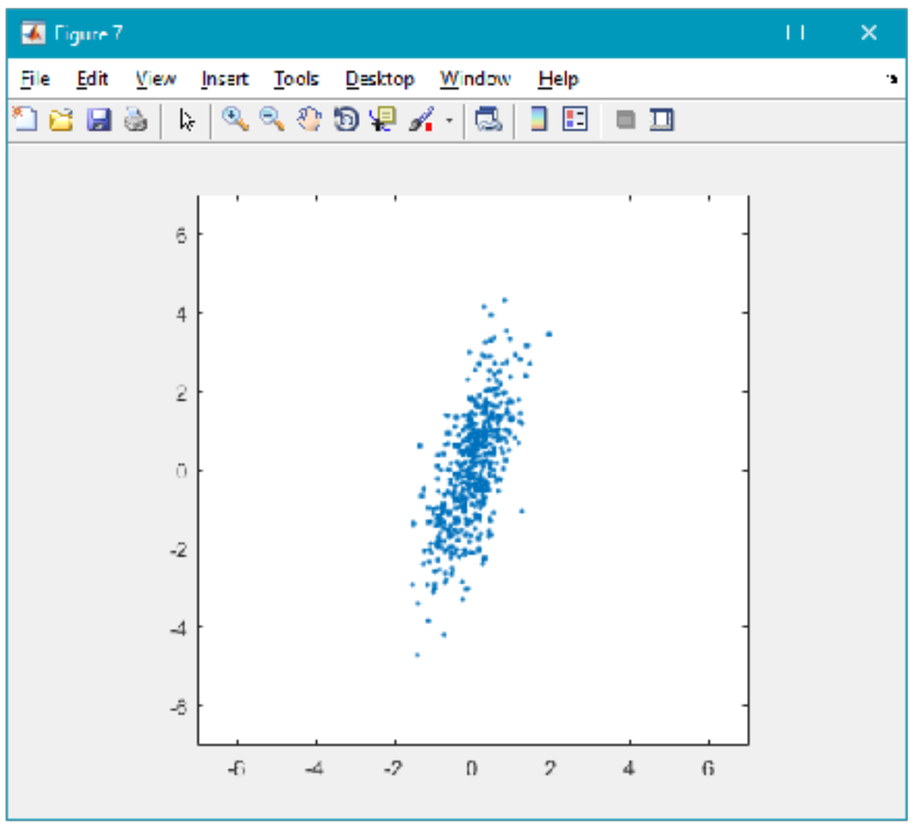


Рисунок 3.21– Сгенерированное множество точек для случая $\sigma_1^2 = 0.3,\ \sigma_2^2 = 2,\ \sigma_{12} = 0.5$

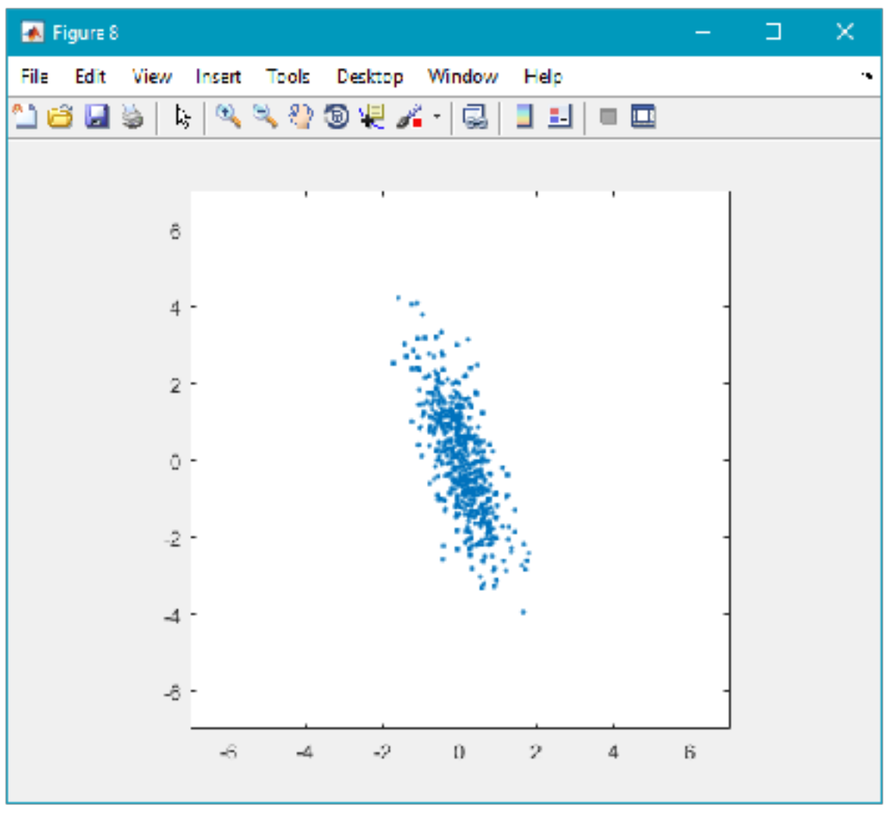


Рисунок 3.22– Сгенерированное множество точек для случая $\sigma_1^2 = 0.3,\ \sigma_2^2 = 2,\ \sigma_{12} = -0.5$

Из этих графиков можно сделать следующий вывод, если признаки некоррелированы, $S = diag(\sigma_1^2, \ldots, \sigma_n^2)$, то линии уровня плотности распределения имеют форму эллипсоидов с центром m и осями, параллельными линиям координат (рисунок 3.18 и 3.19). Если признаки имеют одинаковые дисперсии, то эллипсоиды являются сферами (рисунок 3.15 –3.17). Если признаки коррелированны, то матрица S не диагональная и линии уровня имеют форму эллипсоидов, оси которых повернуты относительно исходной системы координат (рисунок 3.20 – 3.22).

Итак, в коде байесовского классификатора мы определяли класс по формуле из теории вероятности: умножали вероятность соответствующего класса на плотность распределения, причем предполагали, что плотность – нормальная, а вероятности известны.

Но можно решать задачу классификации и путём вычисления

расстояний.

Допустим, у нас есть некая точка в виде *n*-мерного вектора, и есть выборка, которую тоже можно отразить в виде набора точек в *n*-мерном пространстве. Тогда, если мы сначала получим некие характеристики выборки, допустим, узнаем её центр-масс, то можно определить расстояние от точки до центра масс. Если выборок много, то какое расстояние меньше, тому классу и принадлежит точка.

Пример 6. Сначала используем формулу из школьного курса: евклидово расстояние.

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} = \sqrt{\sum_{k=1}^{n} (p_k - q_k)^2}, \qquad (3.12)$$

где p и q, n-мерные вектора.

```
close('all');
clear;
% x – входной вектор
x=[0.1 0.5 0.1]';
% m1 и m2 играют роль оценки двух неизвестных множеств: их центры масс
m1=[0 0 0]';
m2=[0.5 0.5 0.5]';
m=[m1 m2];
% передача информации в функцию
z=euclidean_classifier(m,X)

% функия в отдельном файле
function [z]=euclidean_classifier(m,X)
% вернёт индекс класса 1 или 2 к которому принадлежит точка X
% X – это матрица 1x3
% m – это матрица 3x2
% size() – возвращает координаты матрицы
[l,c]=size(m); % (l, c) = (3, 2)
[l,N]=size(X); % (l, N) = (3, 1)
for i=1:N % цикл по строкам X от 1 до 3
   for j=1:c % цикл по столбцам m от 1 до 2
      % высчитываем два евклидовых расстояния
      de(j)=sqrt((X(:,i)-m(:,j))'*(X(:,i)-m(:,j)));
   end
   % de(1) = 0.5196
   % de(2) = 0.5657
   % в пит будет содержаться минимальное значение, т.е. de(1)
   % z(i) – будет содержать его индекс, т.е. 1
   [num,z(i)]=min(de);
end
```

```
>> close('all');
clear;
% x - входной вектор
X=[0.1 0.5 0.1]';
% m1 и m2 играют роль оценки двух н
масс
m1=[0 0 0]';
m2=[0.5 0.5 0.5]';
m=[m1 m2];
% передача информации в функцию
z=euclidean_classifier(m,X)

z =

     1
```


Однако можно использовать и более сложную формулу расстояния: расстояние Махаланобиса, которое использует не только центр масс, но и матрицу корреляции.

Если мы оценили множества и нашли его среднее значение $m = [m_1, m_2, \ldots, m_n]^T$, а также матрицу ковариации S, то расстояние определится следующим образом от точки $x = [x_1, x_2, \ldots, x_n]^T$ до m.

$$D_M(x) = \sqrt{(x - m)^T S^{-1} (x - m)}. \quad (3.13)$$

Тогда точка будет принадлежать тому классу для которого выполняется условие:

$$\sqrt{(x - m_i)^T S^{-1} (x - m_i)} < \sqrt{(x - m_j)^T S^{-1} (x - m_j)}, \forall j \neq i, \quad (3.14)$$

т.е. для которого расстояние Махаланобиса минимально.

Пример 7. Напишем код для соответствующей классификации.

```
close('all');
clear;
x=[0.1 0.5 0.1]';
m1=[0 0 0]';
m2=[0.5 0.5 0.5]';
m=[m1 m2];
% для двух множеств одна и таже матрица ковариации S
S=[0.8 0.01 0.01;
0.01 0.2 0.01;
0.01 0.01 0.2];
z=mahalanobis_classifier(m,S,x)

% функция для расстояния Махаланобиса
function z=mahalanobis_classifier(m,S,X)
[l,c]=size(m);
[l,N]=size(X);
for i=1:N
   for j=1:c
      dm(j)=sqrt((X(:,i)-m(:,j))'*inv(S)*(X(:,i)-m(:,j)));
   end
% dm(1) = 1.1334
% dm(2) = 0.9918
   [num,z(i)]=min(dm);
end
% z = 2
```

```
>> close('all');
clear;
x=[0.1 0.5 0.1]';
m1=[0 0 0]';
m2=[0.5 0.5 0.5]';
m=[m1 m2];
% для двух множеств одна и таже
S=[0.8 0.01 0.01;
   0.01 0.2 0.01;
   0.01 0.01 0.2];
z=mahalanobis_classifier(m,S,x)

z =

     2
```

Мы видим, что в этом случае точка принадлежит классу 2.

Получается противоречие: евклидово расстояние показывает принадлежность к классу 1, а расстояние Махаланобиса – наоборот. В таком случае нужно принять ответ классификатора Махаланобиса, т.к. за счёт матрицы ковариации расстояние Махаланобиса учитывает ещё и форму распределения точек вокруг центра масс. По сути расстояние Махаланобиса – это просто расстояние между заданной точкой и центром масс, делённое на ширину эллипсоида в направлении заданной точки.

Рассмотрим далее принцип максимума правдоподобия, который

составляет основу параметрического подхода. Пусть задано множество объектов $X^m = |x_1, ..., x_m|$, выбранных независимо друг от друга из вероятностного распределения с плотностью φ(x;θ). Функцией правдоподобия называется совместная плотность распределения всех объектов выборки:

$$p(X^m;\theta) = p(x_1, ..., x_m;\theta) = \prod_{i=1}^{m} \varphi(x_i;\theta) \quad . \tag{3.15}$$

Значение параметра θ, при котором выборка максимально правдоподобна, то есть функция $p(X^m;\theta)$ принимает максимальное значение, называется оценкой максимума правдоподобия.

В случае гауссовой плотности с параметрами $\theta = (m, S)$ задача максимизации правдоподобия имеет аналитическое решение:

$$m_{ML} = \frac{1}{N}\sum_{i=1}^{N} x_i \quad , \tag{3.16}$$

$$S_{ML} = \frac{1}{N}\sum_{i=1}^{N} (x_i - m_{ML})(x_i - m_{ML})^T \quad . \tag{3.17}$$

Пример 8. Рассмотрим на примере как работает алгоритм максимума правдоподобия. Сгенерируем 50 точек, имеющих по две координаты каждая, используя гауссово распределение с центром $m = [2, -2]^T$, $S = \begin{bmatrix} 0.9 & 0.2 \\ 0.2 & 0.3 \end{bmatrix}$. Потом оценим параметры этого распределения по формулам (3.16) и (3.17), сравним результаты.

```
close('all');
clear;
% Генерируем 50 точек с заданным распределением гаусса
randn('seed',0);
m = [2 -2]; S = [0.9 0.2; 0.2 .3];
X = mvnrnd(m,S,50)';
% Оцениваем параметры распределения по выборке
[m_hat,
S_hat]=Gaussian_ML_estimate(X)
```

```
>> close('all');
clear;
% Генерируем 50 точек с заданным распре
randn('seed',0);
m = [2 -2]; S = [0.9 0.2; 0.2 .3];
X = mvnrnd(m,S,50)';
% Оцениваем параметры распределения по
[m_hat, S_hat]=Gaussian_ML_estimate(X)

m_hat =

    2.0495
   -1.9418


S_hat =

    0.8082    0.0885
    0.0885    0.2298
```

Функцию можно создать в отдельном файле Matlab.

```
function [m_hat,S_hat]=Gaussian_ML_estimate(X)
% Входной параметр:
%   X:      lxN  матрица, чьи колонки есть наши данные.
% Выходной аргумент:
%   m_hat:  l-размерная оценка среднего вектора распределения.
%   S_hat:  lxl оценка ковариационной матрицы распределения.
 [l,N]=size(X); % l = 2, N = 50
 m_hat=(1/N)*sum(X')'; % получаем среднеарифметический центр масс
% создаём и инициализируем 0 квадратную матрицу 2x2
```

```
S_hat=zeros(l);
for k=1:N
   S_hat=S_hat+(X(:,k)-m_hat)*(X(:,k)-m_hat)';
end
% Высчитываем среднее
S_hat=(1/N)*S_hat;
```

По результатам получился вектор m_hat = $[2.0495, -1.9418]^T$, $S\_hat = \begin{bmatrix} 0.8082 & 0.0885 \\ 0.0885 & 0.2298 \end{bmatrix}$.

Как видно, получившиеся оценки близки к оригиналу, однако, нужно учесть, что они проводились на выборке, состоящей из 50 элементов, что очень мало, поэтому эти оценки не надёжны. Для увеличения надёжности оценок нужно увеличить выборку.

Пример 9. Теперь рассмотрим последний пример, в котором обобщим весь предыдущий материал. Сгенерируем два множества X и X_1 каждое содержит 999 трехмерных точек, X – обучающее множество, X_1 – тестовое. Каждое множество включает три равновероятных класса: w_1, w_2, w_3; классы создаются с помощью распределения Гаусса со средними $m_1 = [0, 0, 0]^T$, $m_2 = [1, 2, 2]^T$, $m_3 = [3, 3, 4]^T$ и матрицами ковариации

$$S_1 = S_2 = S_3 = \begin{bmatrix} 0.8 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 0.8 \end{bmatrix} = \sigma^2 I$$

Используя X с помощью принципа максимального правдоподобия оценим такие параметры как среднее m и матрицы ковариации S_1, S_2, S_3, т.к. у нас 10^3 точек, то эти оценки будут надёжными. Далее, используя эти оценки проведём классификацию с помощью расстояния Махаланобиса, Евклида, а также байесовским классификатором. Для каждого способа вычислим ошибку вероятности и сравним результаты.

Сначала сгенерируем обучающее множество X.

```
close('all');
clear;
% Матрица m 3x3
m=[0 0 0; 1 2 2; 3 3 4]';
% Создаём матрицу S1 3x3 (единичную матрицу умножаем на 0.8)
S1=0.8*eye(3);
% Как переменная m содержит три вектора-центра масс, так и тут
% трехмерный массив S содержит три одинаковых матрицы ковариации
S(:,:,1)=S1;S(:,:,2)=S1;S(:,:,3)=S1;
% вероятность появления каждого из трех классов
P=[1/3 1/3 1/3]';
% количество элементов в обучающем множестве
N=1000;
% устанавливаем датчик случайных чисел с параметрами seed и 0.
randn('seed',0);
% Генерируем множество X[3;999] – сами точки, y[1, 999] – их метки
% от 1 до 333 – метка 1, от 334 до 666 – метка 2, от 667 до 999 – метка 3.
```

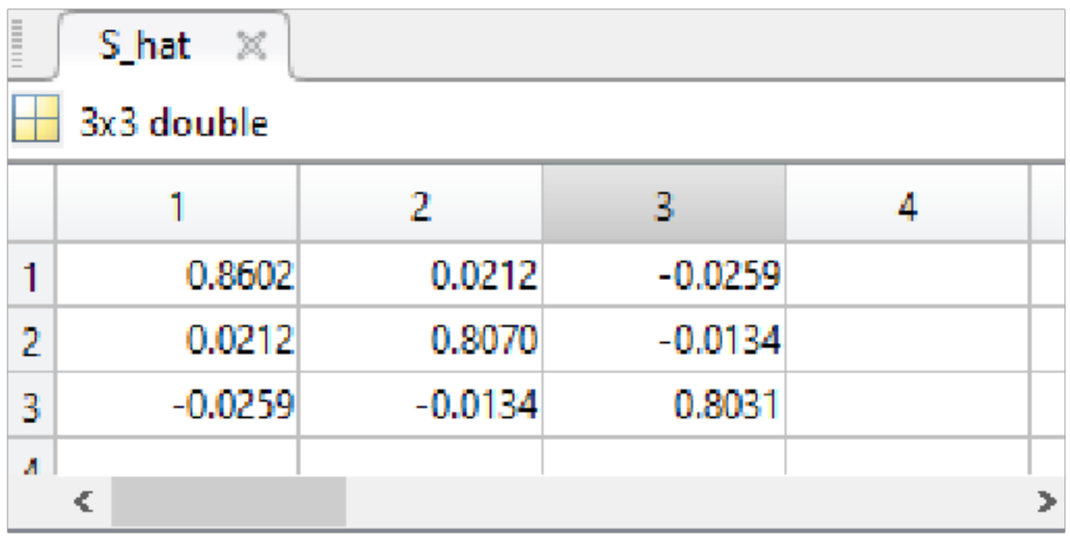
```
[X,y]=generate_gauss_classes(m,S,P,N);

% Функцию пишем в отдельном файле
function [X,y]=generate_gauss_classes(m,S,P,N)
[l,c]=size(m); % (l, c) = (3, 3)
X=[]; % Инициализация множеств
y=[];
for j=1:c % цикл от 1 до 3
% Генерируем трехмерные точки со средним m(:, j), где ":" – для всех строк,
% матрицей ковариации S(:, :, j) и количеством 1/3*1000, ограниченным снизу
  t=mvnrnd(m(:,j),S(:,:,j),fix(P(j)*N))';
  % сгенерированные точки присваиваем вектору X причем так, чтобы
  % они добавлялись в конец уже существующего вектора X. Сгенерировали
  % точки для j = 1, добавили их к пустому вектору, затем сгенерировали
  % точки для j = 2, добавили их в конец вектора, который уже содержит
  % точки для j = 1 и т.д.
  X=[X t]; % генерируем метки
  y=[y ones(1,fix(P(j)*N))*j];
end
```

Далее аналогично сгенерируем множество X_1, но уже с другими параметрами для **randn()**.

```
% Генерируем тестовую выборку X1
randn('seed',100);
[X1,y1]=generate_gauss_classes(m,S,P,N);
```

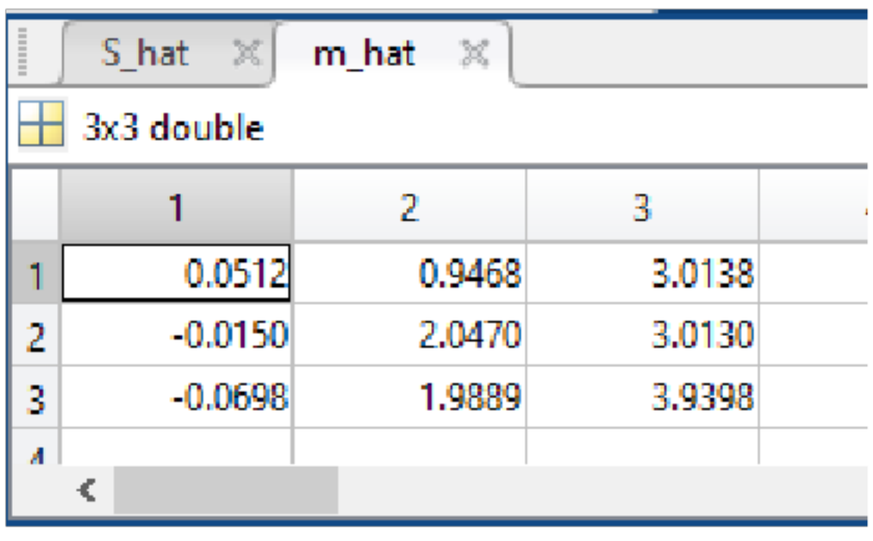
Теперь по принципу максимума правдоподобия вычислим оценки для распределения, которое использовалось при создании множества X.

```
% извлекаем из X только те точки, которые имеют метку 1
class1_data=X(:,find(y==1));
% вычисляем центр масс и ковариационную матрицу
% функция для вычисления была разобрана ранее
[m1_hat, S1_hat]=Gaussian_ML_estimate(class1_data);
% Аналогично
class2_data=X(:,find(y==2));
[m2_hat, S2_hat]=Gaussian_ML_estimate(class2_data);
% Аналогично
class3_data=X(:,find(y==3));
[m3_hat, S3_hat]=Gaussian_ML_estimate(class3_data);
% Получаем общую оценку единой ковариационной матрицы,
% как среднеарифметическое трех уже вычисленных матриц
S_hat=(1/3)*(S1_hat+S2_hat+S3_hat);
m_hat=[m1_hat m2_hat m3_hat];
```


S_hat

3x3 double

	1	2	3	4
1	0.8602	0.0212	-0.0259	
2	0.0212	0.8070	-0.0134	
3	-0.0259	-0.0134	0.8031	


S_hat m_hat

3x3 double

	1	2	3
1	0.0512	0.9468	3.0138
2	-0.0150	2.0470	3.0130
3	-0.0698	1.9889	3.9398

В итоге получаем:

$$S\_hat = \begin{bmatrix} 0.8602 & 0.0212 & -0.0259 \\ 0.0212 & 0.8070 & -0.0134 \\ -0.0259 & -0.0134 & 0.8031 \end{bmatrix} \text{ и } m\_hat = \begin{bmatrix} 0.0512 & 0.9468 & 3.0138 \\ -0.0150 & 2.0470 & 3.0130 \\ -0.0698 & 1.9889 & 3.9398 \end{bmatrix}.$$

Видно, что получившиеся оценки близки к тем, которые мы задавали изначально. В реальности *S* и *m* из условия задачи нам нужны были только для того, чтобы создать выборки. Предполагается, что выборки *X* и X_1 мы получаем откуда-то со стороны и делаем предположение, что они сформированы посредством нормального распределения. Такое предположение мы можем сделать, т.к. рисунки 3.15 – 3.22 демонстрируют, что гауссово распределение может моделировать широкий класс данных.

Теперь, зная характеристики выборки, точнее трёх классов, входящих в неё, можно провести классификацию точек из тестового множества X_1.

```
% высчитываем расстояние от точки X1[i], i=1..999 до точки-центра
% соответствующего класса, z_euclidean будет содержать 999 индексов классов
% с самым минимальным расстоянием
z_euclidean=euclidean_classifier(m_hat,X1);
% Аналогично
z_mahalanobis=mahalanobis_classifier(m_hat,S_hat,X1);
% Используем байесовский классификатор, однако с математической
% точки зрения, чтобы он работал корректно, мы должны подавать ему
% не вычисленные оценки множества X, а данные в условии, т.е. точные
z_bayesian=bayes_classifier(m,S,P,X1);
% Находим процент несовпадений по классам
err_euclidean = (1-length(find(y1==z_euclidean))/length(y1))
err_mahalanobis = (1-length(find(y1==z_mahalanobis))/length(y1))
err_bayesian = (1-length(find(y1==z_bayesian))/length(y1))
```

```
% Функция для классификатора байеса
function [z]=bayes_classifier(m,S,P,X)
[l,c]=size(m);
[l,N]=size(X);
for i=1:N % цикл от 1 до 999
    for j=1:c % цикл от 1 до 3
        t(j)=P(j)*comp_gauss_dens_val(m(:,j),S(:,:,j),X(:,i));
    end
    [num,z(i)]=max(t);
```

end

Получается, что во всех трех случаях ошибки похожи: 7.61%, 7.71%, 7.61%, рисунок 3.23.

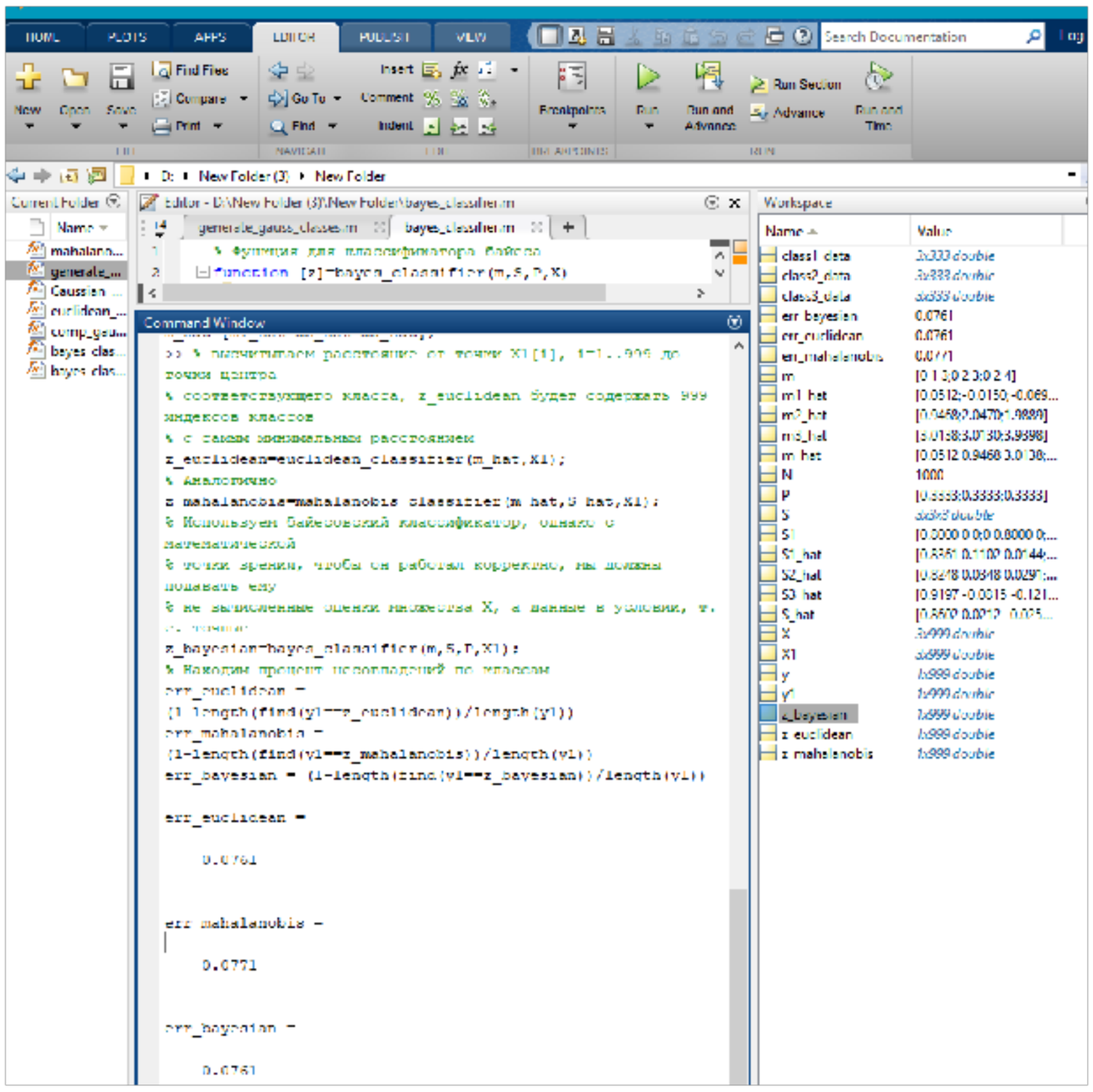


Рисунок 3.23 – Рабочее окно Matlab с результатами классификации с помощью расстояния Махаланобиса, Евклида и байесовским классификатором

Это не случайно, если выполняются 4 следующих условия, то все три метода классификации эквивалентны:

1. Все классы равновероятны.
2. Данные во всех классах имеют гауссовы распределения.
3. Матрицы ковариации одни и те же.
4. Ковариационные матрицы не просто равны, но и диагональные, причем все элементы на главной диагонали равны

Задание 3.1. Проверьте работу программного кода из второго примера для классификации точек плоскости на множества точек «вне» и «внутри» эллипса, для чего внесите следующие изменения в код и проанализируйте полученные результаты:

1) в качестве третьего параметра функции newp укажите другие модели персептрона (сигмоидальные функции): 'tansig' — гиперболический тангенс; 'logsig' — логистический сигмоид; 'purelin' — линейный сигмоид;

2) измените максимальное число итераций для проведения обучения сети;

3) проверьте корректность работы построенной нейронной сети, для чего проведите классификацию нескольких тестовых точек плоскости; для наглядности выполните построение этих точек на той же плоскости;

координаты тестовых точек задайте самостоятельно;

4) измените параметры исходного эллипса (коэффициенты кривой).

Задание 3.2. Рассмотрите, как в примере 3 параболу, заданную в виде общего уравнения кривой второго порядка. Сгенерируйте случайное множество на плоскости (100 точек). Используя нелинейное отображение $\psi(x,y)$ в 5-мерное пространство и однослойный персептрон в 5-мерном пространстве, разделите точки множества на два класса – точки, лежащие «вне» и «внутри» параболы.

Задание 3.2. Вычислите как в примере 4 значения p_1 и p_2, но для следующих двух случаев: ($P(w_1)$=1/6, $P(w_2)$=5/6), ($P(w_1)$=5/6, $P(w_2)$=1/6). Дайте объяснения зависимости результатов классификации от априорных вероятностей.

Задание 3.2. Повторите пример 8, но для N = 500 и N = 5000 точек. Прокомментируйте результат.

В таблице 3.1 приведены индивидуальные задания.

Таблица 3.1 – Индивидуальные варианты

1	Повторите пример 9, но где $S_1 = S_2 = S_3 = \begin{bmatrix} 0.8 & 0.2 & 0.1 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.2 & 0.8 \end{bmatrix} \neq \sigma^2 I$. Напишите вывод по получившемуся ответу.
2	Повторите пример 9, но где для генерации X и X_1 будут использованы следующие вероятности классов P_1 = 1/2, P_2 = P_3 = 1/4. Для этого случая байесовский классификатор должен показать лучший результат. Почему?
3	Повторите пример 9, но где $P(w1)$ = P($w2$) = P($w3$) = 1/3 и $S_1 = \begin{bmatrix} 0.8 & 0.2 & 0.1 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.2 & 0.8 \end{bmatrix}$, $S_2 = \begin{bmatrix} 0.6 & 0.01 & 0.01 \\ 0.01 & 0.8 & 0.01 \\ 0.01 & 0.01 & 0.6 \end{bmatrix}$, $S_3 = \begin{bmatrix} 0.6 & 0.1 & 0.1 \\ 0.1 & 0.6 & 0.1 \\ 0.1 & 0.1 & 0.6 \end{bmatrix}$. Объясните результат.
4	Повторите пример 9, но где $S_1 = S_2 = S_3 = \begin{bmatrix} 0.8 & 0.3 & 0.1 \\ 0.3 & 0.8 & 0.3 \\ 0.1 & 0.3 & 0.8 \end{bmatrix} \neq \sigma^2 I$. Напишите вывод по получившемуся ответу.
5	Повторите пример 9, но где для генерации X и X_1 будут использованы следующие вероятности классов P_1 = 0.8, P_2 = 0.15, P_3 = 0.05. Прокомментируйте результат.
6	Повторите пример 9, но где $P(w1)$ = $P(w2)$ = $P(w3)$ = 1/3 и $S_1 = \begin{bmatrix} 0.9 & 0.2 & 0.1 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.2 & 0.9 \end{bmatrix}$

	, $S_2 = \begin{bmatrix} 0.6 & 0.01 & 0.01 \\ 0.01 & 0.8 & 0.01 \\ 0.01 & 0.01 & 0.6 \end{bmatrix}$, $S_3 = \begin{bmatrix} 0.6 & 0.2 & 0.1 \\ 0.1 & 0.6 & 0.1 \\ 0.1 & 0.2 & 0.6 \end{bmatrix}$. Объясните результат.
7	$S_1 = S_2 = S_3 = \begin{bmatrix} 0.7 & 0.2 & 0.4 \\ 0.2 & 0.7 & 0.2 \\ 0.4 & 0.2 & 0.7 \end{bmatrix} \neq \sigma^2 I$ Повторите пример 9, но где . Напишите вывод по получившемуся ответу.
8	Повторите пример 9, но где для генерации X и X_1 будут использованы следующие вероятности классов $P_1 = 0.6$, $P_2 = P_3 = 0.2$ Для этого случая байесовский классификатор должен показать лучший результат. Почему?
9	Повторите пример 9, но где $P(w1) = P(w2) = 0.2$, $P(w3) = 0.6$ и $S_1 = \begin{bmatrix} 0.8 & 0.2 & 0.1 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.2 & 0.8 \end{bmatrix}$, $S_2 = \begin{bmatrix} 0.6 & 0.01 & 0.01 \\ 0.01 & 0.8 & 0.01 \\ 0.01 & 0.01 & 0.6 \end{bmatrix}$, $S_3 = \begin{bmatrix} 0.6 & 0.1 & 0.1 \\ 0.1 & 0.6 & 0.1 \\ 0.1 & 0.1 & 0.6 \end{bmatrix}$. Объясните результат.
10	Повторите пример 9, но где для генерации X и X_1 будут использованы следующие вероятности классов $P_1 = 0.4$, $P_2 = P_3 = 0.3$. Для этого случая байесовский классификатор должен показать лучший результат. Почему?

Содержание отчета и его форма

Подготовьте отчет, в котором приведите технологию выполнения заданий.

Отчет по лабораторной работе должен содержать:

1) название работы;
2) цель лабораторной работы;
3) формулировку задания и технологию его выполнения;
4) ответы на контрольные вопросы;
5) приложение – файлы выполненных заданий.

Вопросы для защиты работы

1. Что такое матрица ковариации?
2. Что такое дисперсия и математическое ожидание?
3. Чем отличается классификация на основе расстояния Махаланобиса от классификации на основе расстояния Евклида?
4. Что такое байесовский классификатор?
5. Что такое принцип максимума правдоподобия?

ЛАБОРАТОРНАЯ РАБОТА № 4
Классификаторы, основанные на оценки функции оптимизации

Цель и содержание: изучить основные приемы работы с классификаторами, основанными на оценки функции оптимизации.

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций.

Вопросы для обсуждения на лабораторном занятии: принцип действия и алгоритм работы персептрона, алгоритм минимизации среднеквадратической ошибки (LMS). построение классификаторов и их использование в MATLAB.

Формируемые компетенции: ОПК-7.1 ОПК-7.2 ОПК-8.1 ОПК-8.2

Теоретическое обоснование

Техника получения оптимального байесовского классификатора основывается на оценке функции распределения Гаусса для обучающих данных. Однако, это сложная задача особенно для многомерных данных.

Альтернативное решение заключается в отделении классов в многомерном пространстве с помощью разделяющих гиперплоскостей. Разделяющая гиперплоскость может быть записана в виде

$$w^T x + w_0 = 0 \quad (4.1)$$

Также (4.1) можно переписать в виде

$$w^T x' \equiv [w^T, w_0] \begin{bmatrix} x \\ 1 \end{bmatrix} = 0, \quad (4.2)$$

где $\boldsymbol{w}$ – вектор настраиваемых параметров, w_0 – свободный член, $\boldsymbol{x}$ – входной вектор.

С помощью таких гиперплоскостей можно отделить линейно отделимые классы.

Алгоритм адаптации вектора весовых коэффициентов элементарного персептрона можно сформулировать следующим образом.

Если n-ый элемент $x(n)$ обучающего множества корректно классифицирован с помощью весовых коэффициентов $w(n)$, вычисленных на n-ом шаге алгоритма, то вектор весов не корректируется, т.е. действует следующее правило:

$$\begin{aligned} w(n+1) &= w(n), \text{ если } w^T x(n) > 0 \text{ и } x(n) \in C_1 \\ w(n+1) &= w(n), \text{ если } w^T x(n) \le 0 \text{ и } x(n) \in C_2 \end{aligned} \quad (4.3)$$

В противном случае вектор весов персептрона подвергается коррекции в соответствии со следующим правилом:

$$\begin{aligned} w(n+1) &= w(n) - \eta(n)\, x(n), \text{ если } w^T(n) x(n) > 0 \text{ и } x(n) \in C_2 \\ w(n+1) &= w(n) + \eta(n)\, x(n), \text{ если } w^T(n) x(n) \le 0 \text{ и } x(n) \in C_1 \end{aligned}, \quad (4.4)$$

где C_1 и C_2 – линейно разделимые классы, а $\eta(n)$ – параметр скорости обучения.

Создадим функцию, которая будет обучать наш линейный нейрон (настраиваемую гиперплоскость). Сигнатура функции будет следующей: **[w, iter, mis_clas] = perce(X, y, w_ini. rho)**, где *X* – это матрица размером (l+1)xN (l – размер входного вектора, N – количество паттернов для обучения), y – вектор меток для класса C_1 (+1) и C_2 (-1), w_ini – вектор начальных параметров, которые нужно настроить в процессе обучения так, чтобы гиперплоскость отделяла C_1 от C_2, rho – η=const, т.е. скорость обучения, w – вектор, вычисленный алгоритмом, iter – количество итераций за которые был вычислен w, mis_clas – количество неправильно классифицированных векторов.

Рассмотрим примеры.

Пример 1. Необходимо сгенерировать 4 множества X_i, каждое – это набор точек в двумерном пространстве. Каждая точка из X_i имеет метку -1, точки случайно разбросаны в квадрате [3, 5]x[3, 5], [2, 4]x[2, 4], [0, 2]x[2, 4], [1, 3]x[1, 3]. Точки, распложенные в квадрате [0, 2]x[0,2] имеют метку класса +1. Требуется визуально отобразить классы, обучить персептрон для η=0.01 и η=0.05, и начальном векторе параметров $[1, 1, -0.5]^T$.

Листинг функции, реализующий персептрон, приведён ниже.

```
function [w,iter,mis_clas]=perce(X,y,w_ini,rho)

[l,N]=size(X);
max_iter=20000; % Максимальное количество итераций,
% которые будет работать персептрон, после этого будет
% считаться, что решение не найдено.

w=w_ini;       % Инициализируем вектор параметров
iter=0;        % Счётчик итераций

mis_clas=N;    % Инициализируем количество неправильно
% классифицированных паттернов

% цикл while по двум условиям
while(mis_clas>0)&&(iter<max_iter)
   iter=iter+1;
   mis_clas=0;

   gradi=zeros(l,1); % Вычисление корректирующего компонента для вектора
   for i=1:N
      if((X(:,i)'*w)*y(i)<0)
         mis_clas=mis_clas+1;
         gradi=gradi+rho*(-y(i)*X(:,i));
      end
   end

   if(iter==1)
      fprintf('\n First Iteration: # Misclassified points = %g \n',mis_clas);
   end
```

```
   w=w-rho*gradi; % Обновление вектора параметров
end
```

Листинг основного кода приведён ниже.

```
close('all');
clear
rand('seed',0);
% Генерируем класс Xi
N=[100 100]; % 100 векторов для каждого класса
l=2; % Размерность входа каждого вектора
 % для генерирования паттернов из X2, X3, X4 нужно вместо
% нижней строчки подставить одну из закомментированных строк
x=[3 3]';
% x=[2 2]'; для X2
% x=[0 2]'; для X3
% x=[1 1]'; для X4
% получаем 200 точек для Xi и для точек квадрата [0, 2]x[0, 2]
X1=[2*rand(l,N(1)) 2*rand(l,N(2))+x*ones(1,N(2))];
X1=[X1; ones(1,sum(N))];
y1=[-ones(1,N(1)) ones(1,N(2))]; % получаем метки
1. Выводим множество Xi
figure(1), plot(X1(1,y1==1),X1(2,y1==1),'bo',...
X1(1,y1==-1),X1(2,y1==-1),'r.')
figure(1), axis equal
% 2. Запускаем алгоритм обучения персептрна для эта=0.01
rho=0.01; % Скорость обучения
w_ini=[1 1 -0.5]'; % начальные веса
[w,iter,mis_clas]=perce(X1,y1,w_ini,rho)
```

Имеем 4 варианта X_i, i = 1..4, которые нужно отделить с помощью персептрона (рисунок 4.1 – 4.4).

Видно из рисунка X_4 (рисунок 4.4) классы линейно не разделимы.

Результаты обучения приведены в таблице 4.1, 4.2. В таблице 4.1 представлено количество итераций, за которое персептрон находит решение в зависимости от η, а в таблице 4.2 количество неправильно распознанных паттернов.

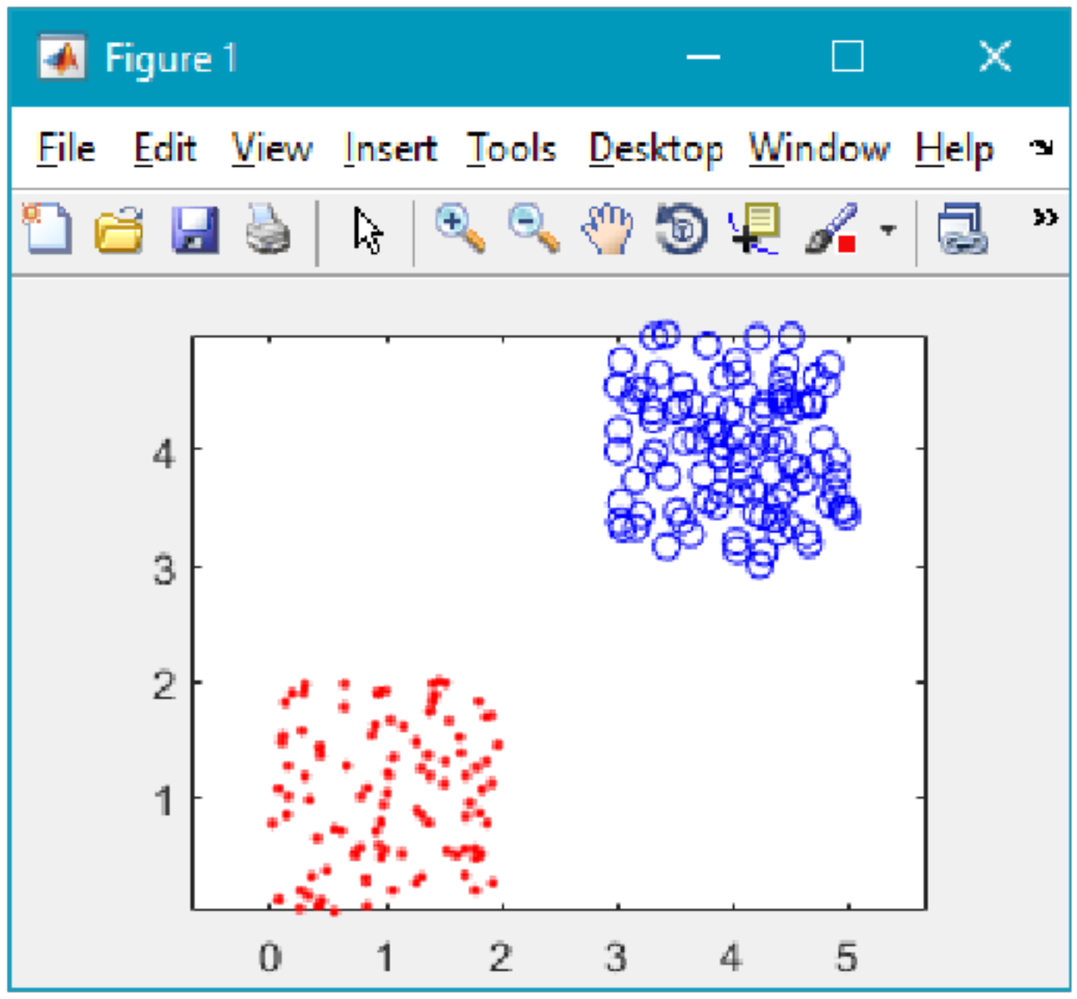


Рисунок 4.1 – Два класса для X_1

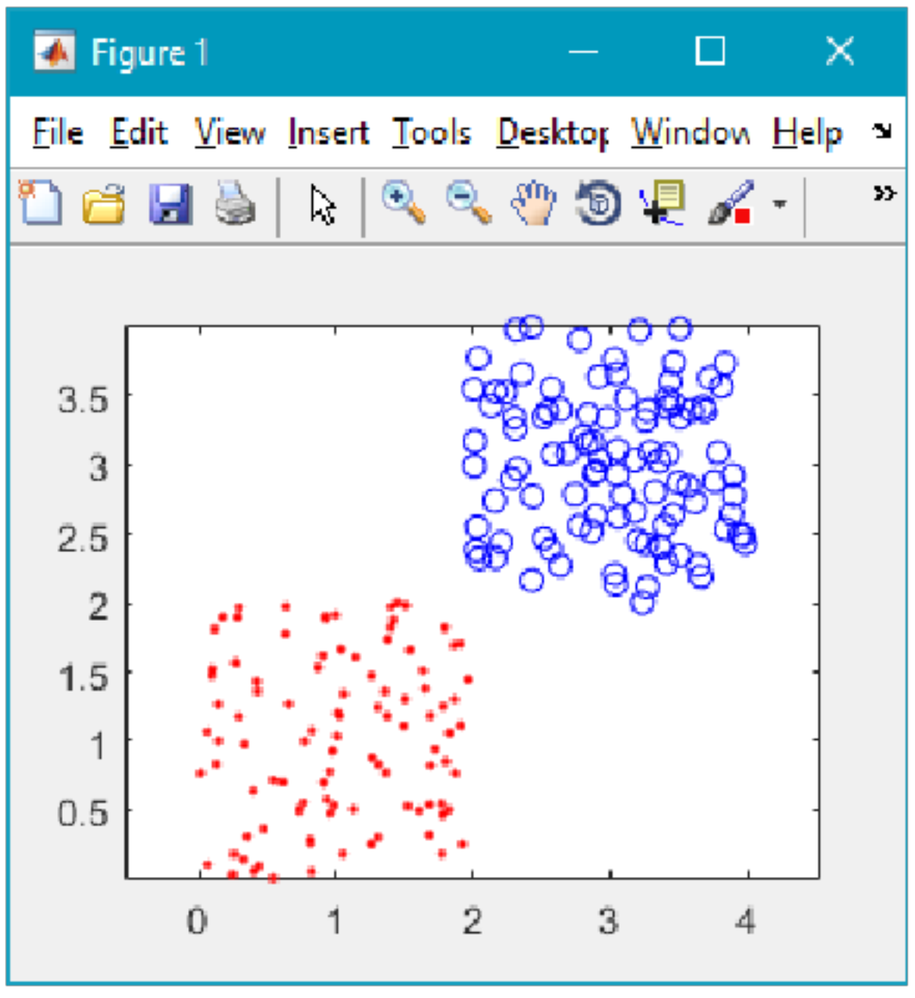


Рисунок 4.2 – Два класса для X_2

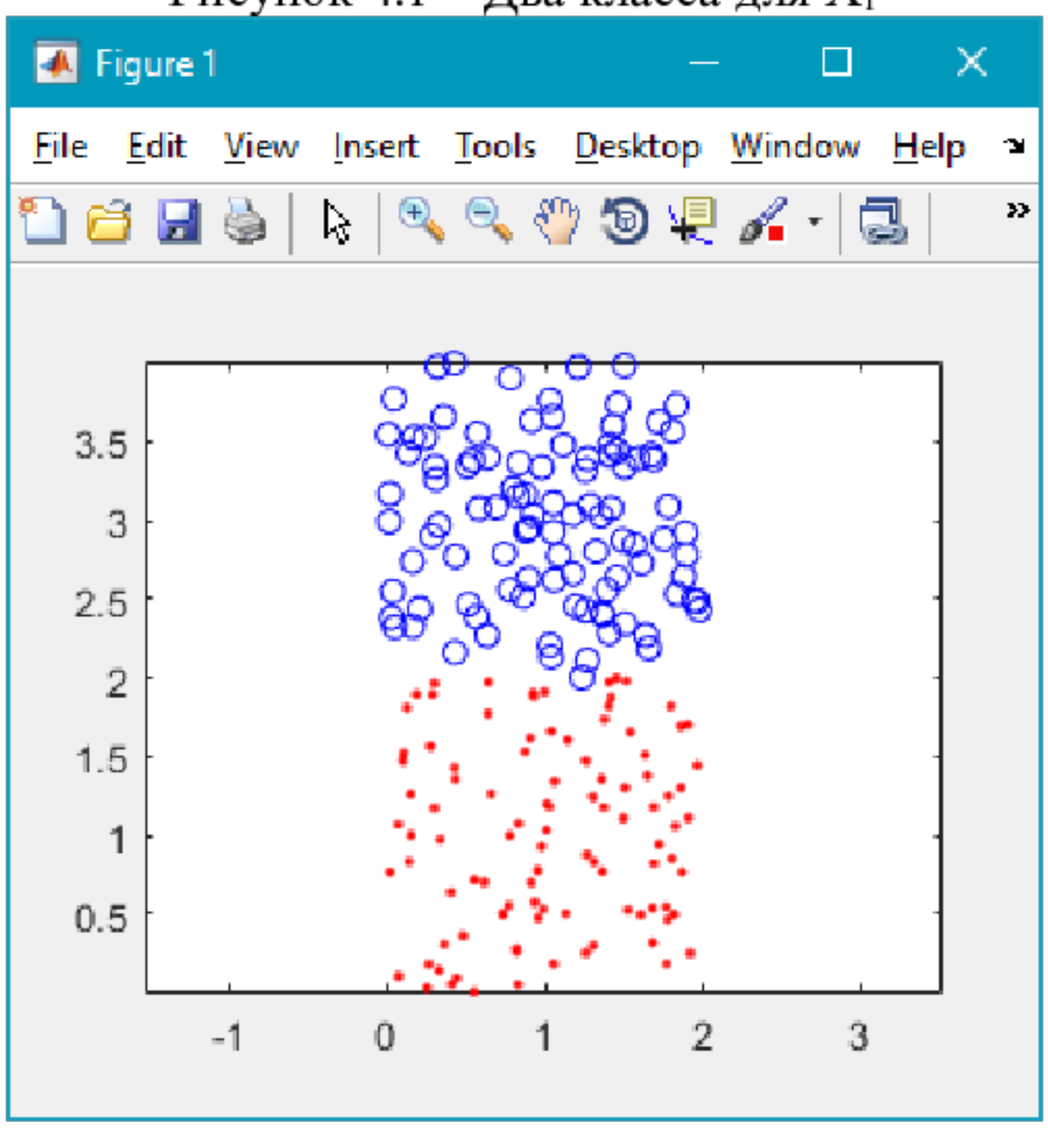


Рисунок 4.3 – Два класса для X_3

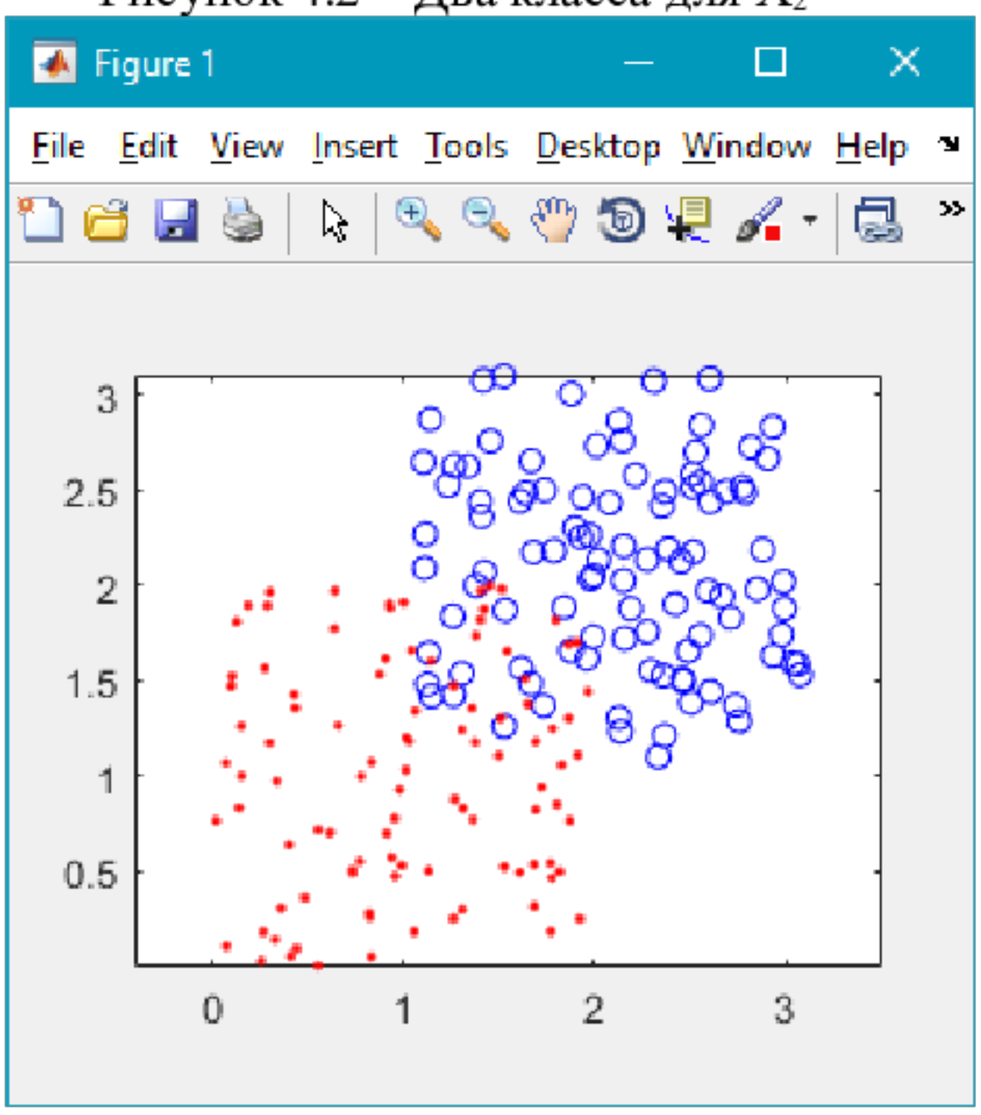


Рисунок 4.4 – Два класса для X_4

Таблица 4.1 – Количество итераций, за которые персептрон находит решение в зависимости от η

	X_1	X_2	X_3	X_4
$\eta = 0.01$	134	134	5441	Оптимального решения не найдено
$\eta = 0.05$	5	5	252	Оптимального решения не найдено

Таблица 4.2 – Количество неправильно распознанных паттернов

	X_1	X_2	X_3	X_4
$\eta = 0.01$, $\eta = 0.05$	0	0	0	25

При увеличении η увеличивается скорость обучения персептрона. В

последнем случае оптимального решения персептрон не находит, поэтому проведённая прямая отделяет не все паттерны класса.

Функция **perce(X,y,w_ini,rho)** запрограммирована для пакетного режима, т.е. вектор **w** корректируется один раз после вычисления поправок для каждого примера.

Пример 2. Обучение персептрона в онлайн-режиме. Ниже приведён листинг функции, где персептрон обучается в онлайн-режиме, где корректировки вычисляются для каждого примера, но далее, они не накапливаются, а сразу применяются к вектору весов.

```
function [w,iter,mis_clas]=perce_online(X,y,w_ini,rho)

[l,N]=size(X);
max_iter=10000000; % максимальное количество итераций
% требуется, если классы расположены близко друг к другу

w=w_ini;
iter=0;

mis_clas=N;
while(mis_clas>0)&&(iter<max_iter)
   mis_clas=0;

   for i=1:N
      if((X(:,i)'*w)*y(i)<0)
         mis_clas=mis_clas+1;
         w=w+rho*y(i)*X(:,i);  % Обновляем вектор весов
      end
      iter=iter+1;
   end

   if(iter==1)
      mis_clas
   end
end
```

Вызвать эту функцию можно с помощью следующей сигнатуры: **[w,iter,mis_clas]=perce_online(X1,y1,w_ini,rho).**

Если повторить предыдущий эксперимент, используя выше указанную функцию, при применении которой персептрон обучается в онлайн-режиме, то получим следующие результаты, таблица 4.3, 4.4.

Таблица 4.3 – Количество итераций, за которые персептрон находит решение в зависимости от η

	X_1	X_2	X_3	X_4
$\eta = 0.01$	600	600	6589400	Оптимального решения не найдено
$\eta = 0.05$	400	400	7729200	Оптимального решения не найдено

Таблица 2.4 – Количество неправильно распознанных паттернов

	X_1	X_2	X_3	X_4
$\eta = 0.01$, $\eta = 0.05$	0	0	0	6

Требуется значительно больше итераций, онлайн-обучение хуже сходится, зато в практическом плане более экономное, т.к. не требует специальных структур для хранения и накопления поправок.

Аппаратура и материалы. 64-разрядный (x64) персональный компьютер, процессор с тактовой частотой 1 ГГц и выше, оперативная память 1 Гб и выше, свободное дисковое пространство не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Matlab (R2013) и выше.

Указание по технике безопасности. Самостоятельно не производить: установку и удаление программного обеспечения; ремонт персонального компьютера. Соблюдать правила технической эксплуатации и техники безопасности при работе с электрооборудованием.

Методика и порядок выполнения работы

Выполните предложенные задания, предварительно рассмотрев приведенные в теоретической части примеры.

Задание 4.1 Создать два частично пересекающихся класса точек (100 точек для каждого класса) как показано в таблице 4.5.

Задание 4.2 Разделить их насколько это возможно с помощью персептрона. Тип обучения персептрона онлайн или пакетный указан в таблице.

Задание 4.3 Нарисовать найденное решение.

Таблица 4.5 – Индивидуальные варианты

№ Варианта	Задание	№ Варианта	Задание
1	$\begin{pmatrix} 0 & A \\ B & 0 \end{pmatrix}$ Онлайн обучение.	6	$\begin{pmatrix} A & 0 \\ B & 0 \end{pmatrix}$ Пакетное обучение.
2	$\begin{pmatrix} A & 0 \\ B & 0 \end{pmatrix}$ Пакетное обучение.	7	$\begin{pmatrix} 0 & A \\ B & 0 \end{pmatrix}$ Онлайн обучение.
3	$\begin{pmatrix} 0 & 0 \\ A & B \end{pmatrix}$ Пакетное обучение.	8	$\begin{pmatrix} A & 0 \\ B & 0 \end{pmatrix}$ Онлайн обучение.
4	$\begin{pmatrix} 0 & A \\ B & 0 \end{pmatrix}$ Онлайн обучение.	9	$\begin{pmatrix} 0 & A \\ B & 0 \end{pmatrix}$ Пакетное обучение.
5	$\begin{pmatrix} 0 & 0 \\ A & B \end{pmatrix}$ Пакетное обучение.	10	$\begin{pmatrix} 0 & 0 \\ A & B \end{pmatrix}$ Онлайн обучение.

Содержание отчета и его форма

Подготовьте отчет, в котором приведите технологию выполнения заданий.

Отчет по лабораторной работе должен содержать:

1) название работы;
2) цель лабораторной работы;
3) формулировку задания и технологию его выполнения;
4) ответы на контрольные вопросы;
5) приложение – файлы выполненных заданий.

Вопросы для защиты работы

1. Что такое персептрон?
2. Чем персептрон отличается от сети прямого распространения?
3. Какие задачи можно решать с помощью персептрона?

ЛАБОРАТОРНАЯ РАБОТА 5
Работа с нейронными сетями в MatLab

Цель и содержание: создать и обучить несколько нейронных сетей, решающих задачи линейного и нелинейного разделения классов, используя встроенный язык программирования Matlab

Организационная форма занятий: решение проблемных задач, разбор конкретных ситуаций

Вопросы для обсуждения на лабораторном занятии: освоить базовые команды для создания и обучения простых сетей в Matlab; научиться создавать и обучать сети для линейного и нелинейного разделения классов; научиться строить графики, отражающие результаты работы сетей.

Формируемые компетенции: ОПК-7.1 ОПК-7.2 ОПК-8.1 ОПК-8.2

Теоретическое обоснование

Самый гибкий способ по работе с ИНС в Matlab – это использовать встроенный язык программирования для создания и обучения сетей. В данной лабораторной работе на примере линейного разделения классов и задачи XOR будет рассмотрена техника создания и обучения разных нейронных сетей.

Код Matlab будем выделять жирным шрифтом. Комментарий к коду записывается после символа «%» и идёт до конца строчки. При желании, можно получить более подробную справку о любой функции, установив курсор на нужной и нажав клавишу «F1».

Прежде, чем программировать персептрон, рассмотрим пример построения поверхности отклика для обычного нелинейного нейрона с функцией активации – гиперболический тангенс.

Пример 1. Построение поверхности отклика для нелинейного нейрона с функцией активации – гиперболический тангенс

```
% Готовим Matlab к работе
close all, clear all, clc, format compact;
% Задаём веса нейрона
w = [4 -2];
% Задаём смещение
b = -3;
% Задаём функцию, которую хотим построить, гипербол. тангенс
func = 'tansig';
% Задаём входной вектор
p = [2 3];
% Вычисляем взвешенную сумму входа и весов
activation_potential = p*w'+b;
% Вычисляем выход нелинейной части нейрона
neuron_output = feval(func, activation_potential);
% Задаём входной вектор для диапазона от -10 до +10 с шагом 0.25
[p1, p2] = meshgrid(-10:.25:10);
% Вычисляем вектор выхода нейрона
```

```
z = feval(func, [p1(:) p2(:)]*w'+b);
% Пересчёт вещественных чисел в воксели для 3D-графика
z = reshape(z, length(p1), length(p2));
% Строим график
plot3(p1, p2, z);
% Включаем сетку
grid on;
% Подписываем оси графика
xlabel('Input 1');
ylabel('Input 2');
zlabel('Neuron output');
```

Разберём код. Команда **close all** закрывает все вспомогательные окна, которые могли быть открыты (окна различных мастеров, вроде nntool, не закрываются). **Clear all** удаляет все переменные из области workspace, **clc** – очищает область окна Command window, где будет набираться код. **Format compact** устанавливает формат отображения для чисел, так для вещественных числе будет отображаться 4 точки после запятой.

Веса задаются обычным вектором **w = [4 -2]**, как видно, значения отделяются друг от друга пробелом, указывать тип не обязательно. Функция активации задаётся строкой **'tansig'**. Далее идёт вычисление взвешенной суммы: скалярное произведение входа на веса и прибавка смещения. Вектора **p** и **w** нельзя просто так взять и перемножить, т.к. по правилам линейной алгебры один из них должен быть транспонирован, что и делается с вектором **w** с помощью **w'**. Можно посмотреть значение переменной, оно должно быть -1. Далее вычисляем функцию активации при входе равном -1. Делаем это через **feval()**. Как понятно из названия, функция вычисляет значение некоторой функции (первый параметр) от вектора входов (второй параметр). Причём, как и многие другие фунции Matlab, она перегружена, т.е. её вызов может происходить при разных способах записи второго параметра. В первом случае вектор редуцируется к скалярной величине, выход равен -0.7616.

Далее присваиваем переменным **p1** и **p2** значения от -10 до +10 с шагом 0.25. Делаем это через **meshgrid()**, которая создаёт прямоугольную сетку в специальном формате пригодном для построения графиков. Но для построения самого графика нам нужно знать значения не только **p1** и **p2**, но и выхода нейрона. Теперь вычисляем эти значения для вектора **z** с помощью **feval()**, видно, что теперь второй параметр напрямую записывается в виде скалярного произведения двух векторов плюс смещение. Двоеточие означает перевод значения в формат double (используется для научных и инженерных вычислений. Никогда не используйте float для этих задач в таких языках как C/C++, чтобы избежать переполнения переменной).

Функция **reshape()** пересчитывает числовые значения в воксели для отображения на графике. **Plot3()** строит трёхмерный график, рисунок 5.1. **Grid on** включает сетку на этом графике, рисунок 5.2, 5.3. **Xlabel()**,

ylabel(), **zlabel()** – подписывают оси у графика.

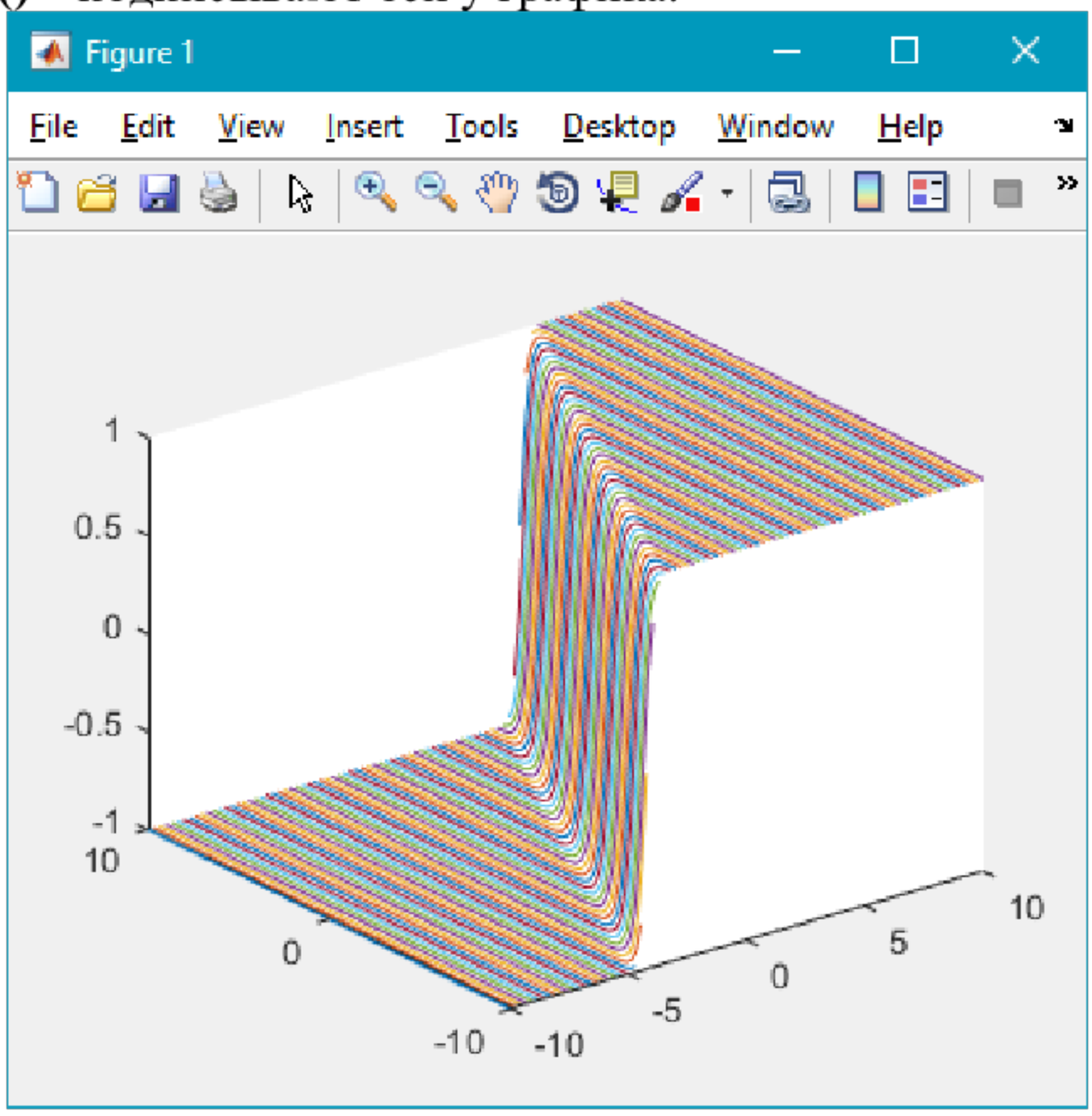


Рисунок 5.1 – Получившийся 3D-гарфик без сетки и подписей осей

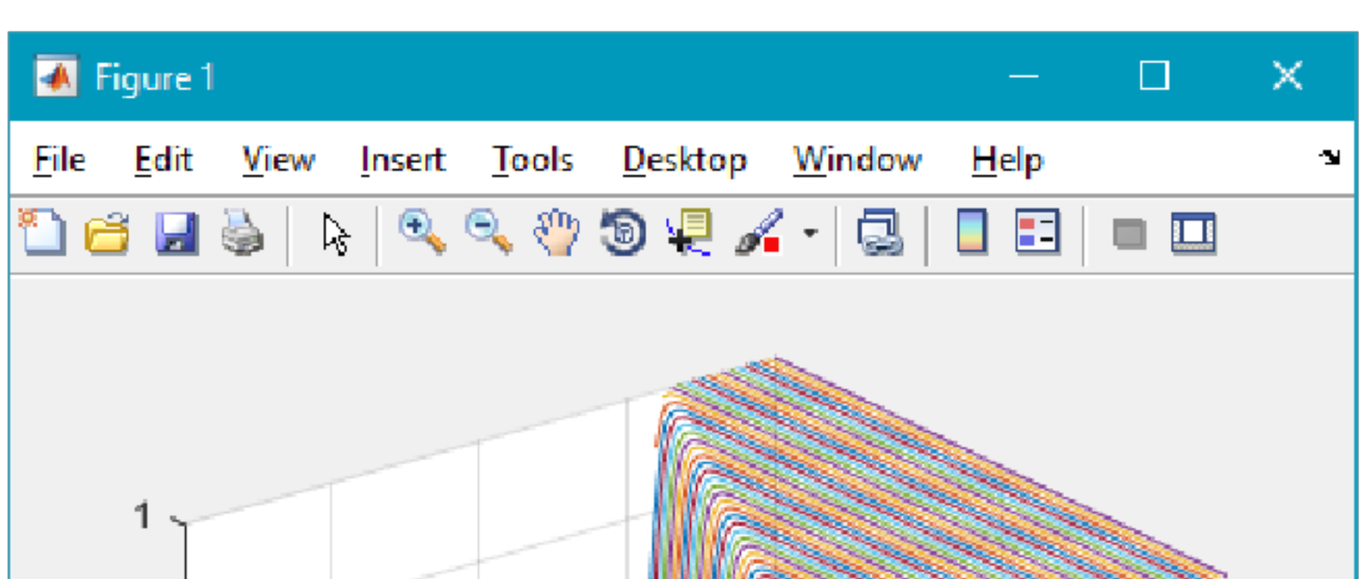


Рисунок 5.2 – Добавили сетку на фон

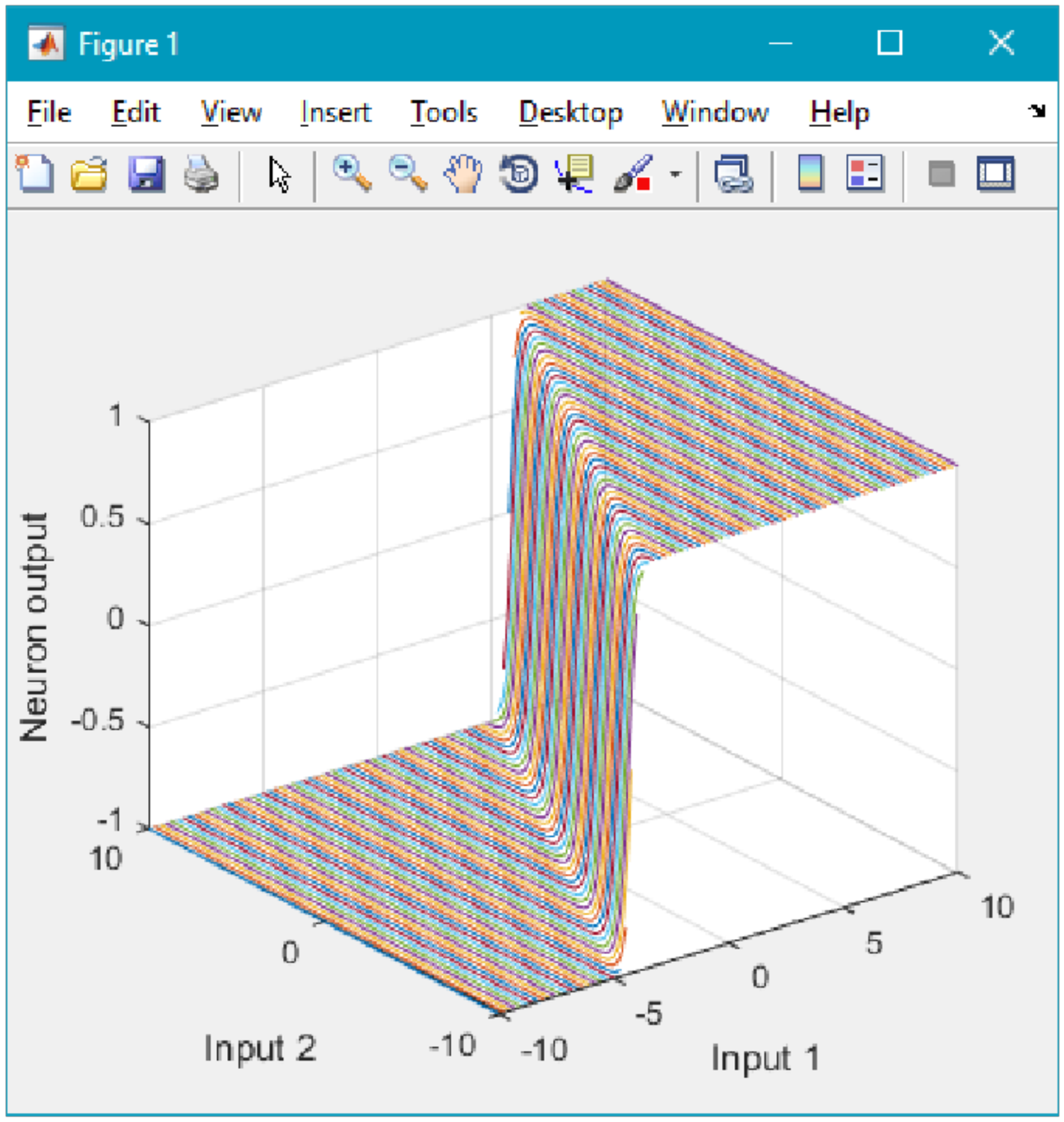


Рисунок 5.3 – Добавили подписи для осей *x*, *y*, *z*

Теперь создадим и обучим двухслойную сеть на одном примере.

Пример 2. Создание и обучение двухслойной сети.

```
close all, clear all, clc, format compact
% Входной транспонированный вектор
inputs = [1:6]'
% Выходной вектор, который должна промоделировать сеть
outputs = [1 2]';
% Задаём общую структуру сети
net = network (1, 2, [1; 0], [1; 0], [0 0; 1 0], [0 1]);
% Показываем её
view(net);
% Задаём количество промежуточных слоёв
net.layers{1}.size = 5;
% Задаём функции активации на них
net.layers{1}.transferFcn = 'logsig';
% Ещё раз отображаем сеть
view(net);
% Устанавливаем размерность входа и выхода
net = configure(net, inputs, outputs);
% Смотрим окончательный вариант сети
view(net);
% Получаем изначальный выход сети без обучения
initial_output = net(inputs);
% Устанавливаем метод обучения
```

```
net.trainFcn = 'trainlm';
% Устанавливаем функцию ошибки
net.performFcn = 'mse';
% Обучаем сеть на одном примере
net = train(net, inputs, outputs);
% Опять подаём вход, но уже на обученную сеть,
% сеть должна промоделировать выход [1 2]
final_output = net(inputs);
```

Входной вектор будет равен массиву из шести элементов от 1 до 6, т.к. диапазон записывается через «:». Функция **network()** задаёт общую структуру сети. Рассмотрим более подробно, что означает каждый параметр.

Первый параметр означает количество входов (не в смысле размерность входного вектора, а в смысле, – сколько векторов будет подано на вход сети. Допустим, в сверточных сетях обычное дело, что на вход подаётся не один, а два или три массива, каждый из которых как-то связан с дальнейшим слоем). У нас этот параметр равен 1. Второй параметр означает количество слоёв в сети. В данном случае имеем двухслойную сеть. Третий параметр – булев вектор, который означает какие нейроны будут иметь смещение, а какие – нет. Т.к. вектор [1; 0], то очевидно, что все нейроны первого слоя будут иметь смещение, а нейроны второго слоя – нет. Четвёртый параметр – это также булев вектор, который означает, – с какими слоями связан вход. В данном случае используется классическая схема, где вход связна только с последующим слоем, а, следовательно, вектор [1; 0]. Пятый параметр – это матрица связей между слоями, которая означает, какой слой с каким связан. Она записывается в виде вектора. Имеем [0 0; 1 0], если записать в виде матрицы, то получится $\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$. Тогда понятно, что первый слой связан со вторым слоем, т.к. единица располагается в позиции (1, 2), где 1 – это номер столбца, 2 – номер строки. Шестой параметр [0 1] – булев вектор, который показывает, с какими слоями связан выход. При таких значениях выход связан с предпоследним слоем.

При такой конфигурации получим структуру сети как на рисунке 5.4.

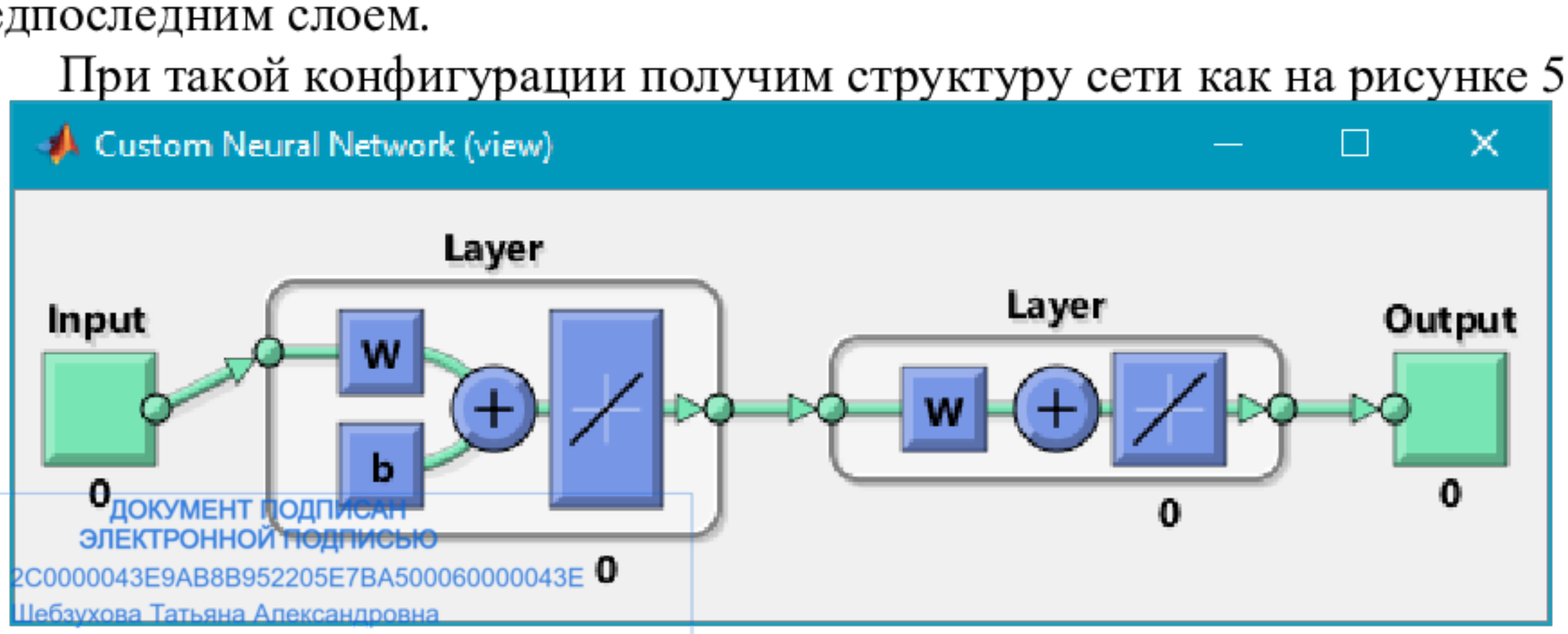


Рисунок 5.4 – Изначальная структура сети

Изменим третий параметр с [1;0] на [1;1], получим структуру как на рисунке 5.5.

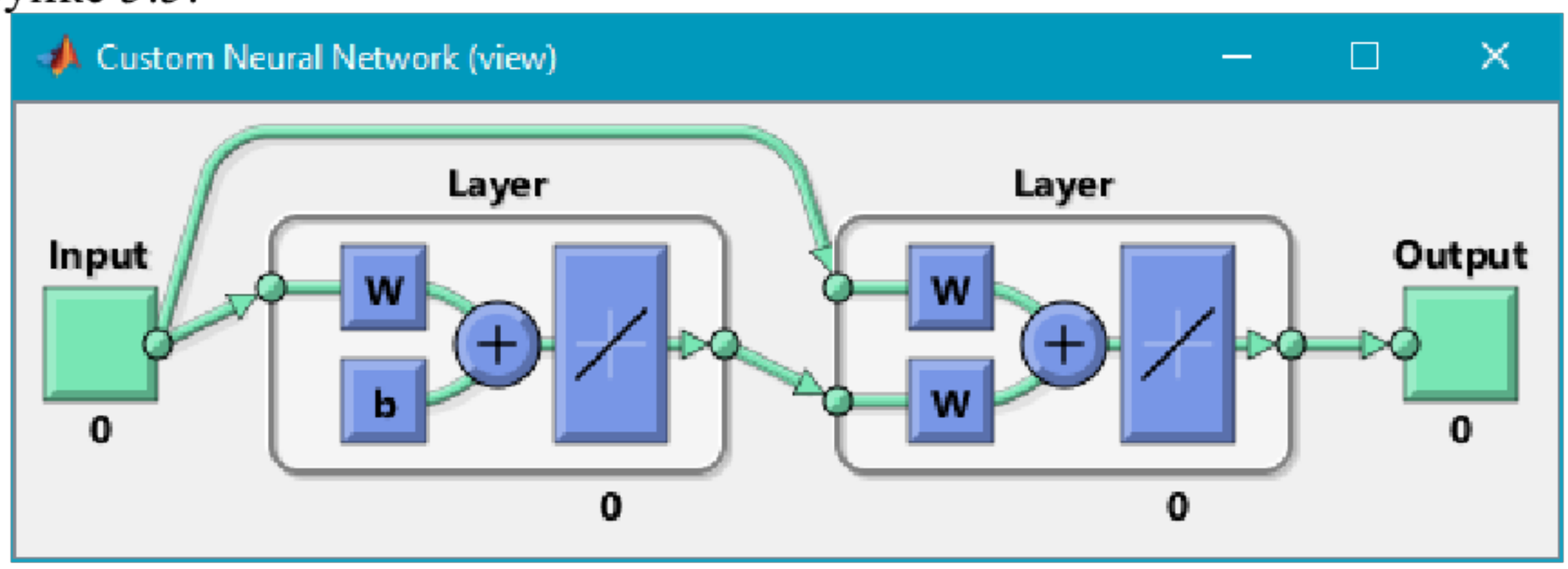


Рисунок 5.5 – Видно, что у второго слоя тоже появилось смещение

Изменим теперь и четвёртый параметр на [1; 1], получим структуру сети как на рисунке 5.6.

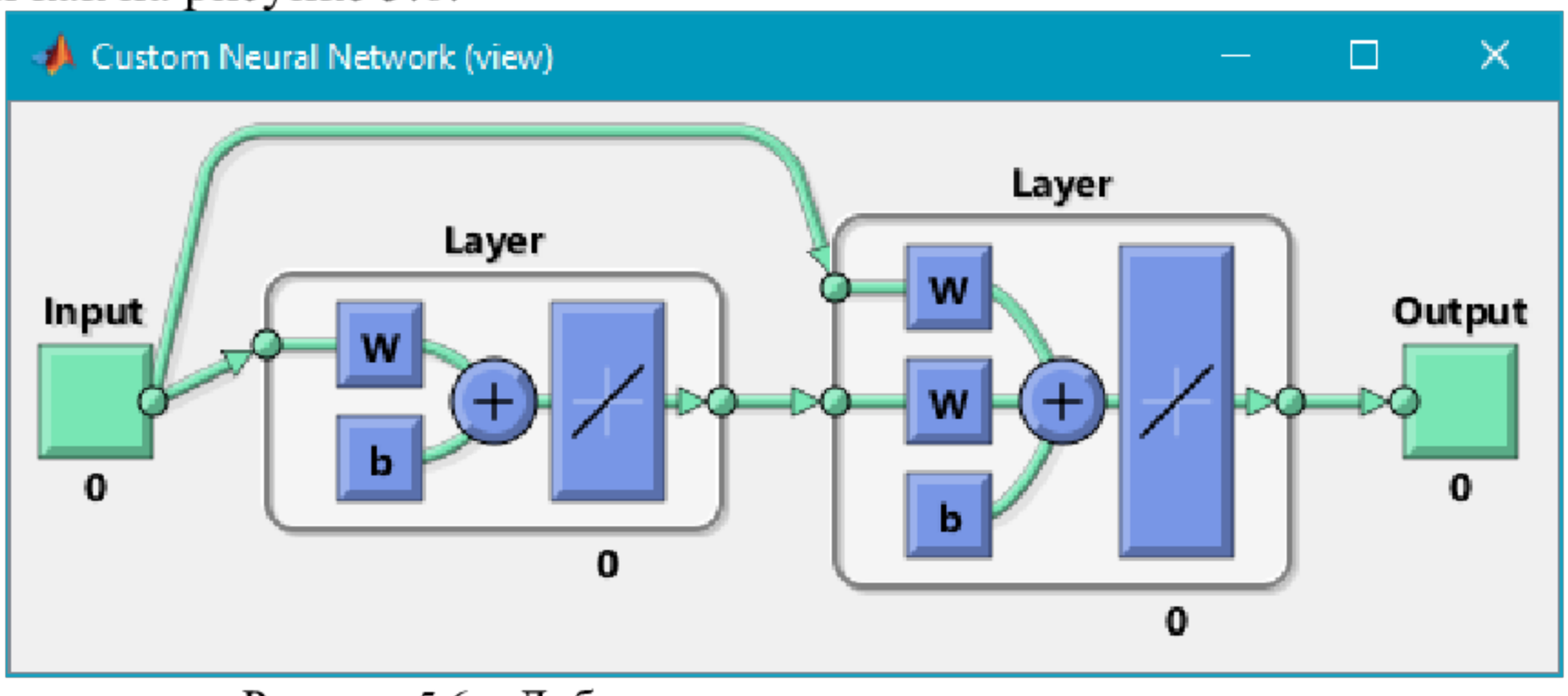


Рисунок 5.6 – Добавилась связь входа со вторым слоем

Изменим остальные два вектора на единичные, получим структуру как на рисунке 5.7.

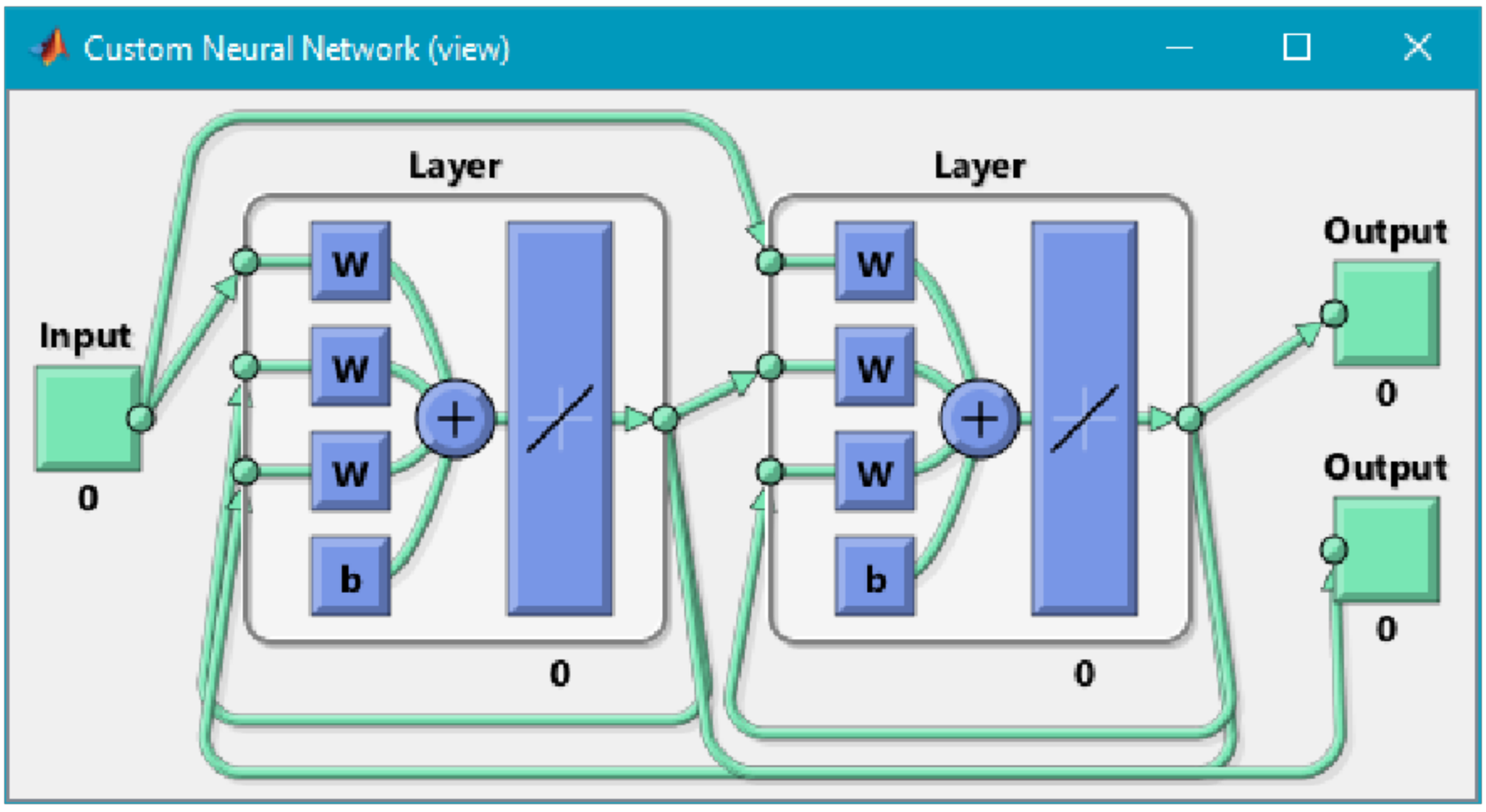


Рисунок 5.7 – Появился второй выход, связанный с первым слоем, а также каждый слой связан с самим собой и второй слой связан с первым (подаёт свои сигналы на вход первого слоя)

Net.layers{1}.size = 5 – присваиваем количество нейронов первому слою сети (индекс записывается в фигурных скобках). Видно, что обращение к данным параметрам осуществляется классическим образом для объектов: через точку. Т.е. сеть в Matlab – это объект (по терминологии объектно-ориентированного программирования (ООП)).

Небольшое примечание. Концепция ООП напрашивается для описания ИНС, т.к. вполне логично, что объект «сеть» включает в себя объекты – «слой», а те – «нейрон». Однако, нужно помнить, что при таком подходе оперативная память выделяется и группируется под конкретные объекты и в случае попыток быстрой реализации ИНС, возможно, с использованием ассемблера, не получится быстро реализовывать матричные операции. Короче, если требуется быстрая реализация сети, то от ООП-описания придётся отказаться.

Net.layers{1}.transferFcn = 'logsig' – присваивает слоям первого слоя функцию активации сигмойд (без отрицательных значений), рисунок 3.8.

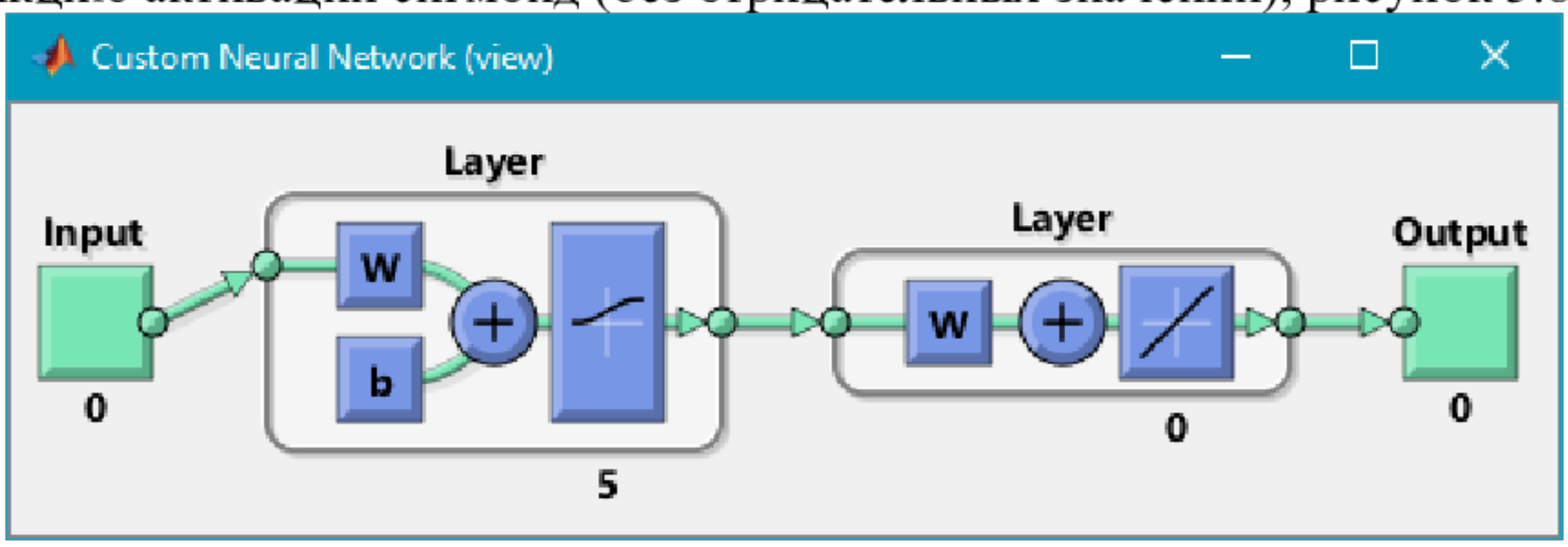


Рисунок 5.8 – Меняем функцию активации у нейронов первого слоя

Для установки входного и выходного слоя можно сконфигурировать сеть по отношению к вектору входа и выхода: **net = configure(net, inputs,**

outputs), рисунок 5.9.

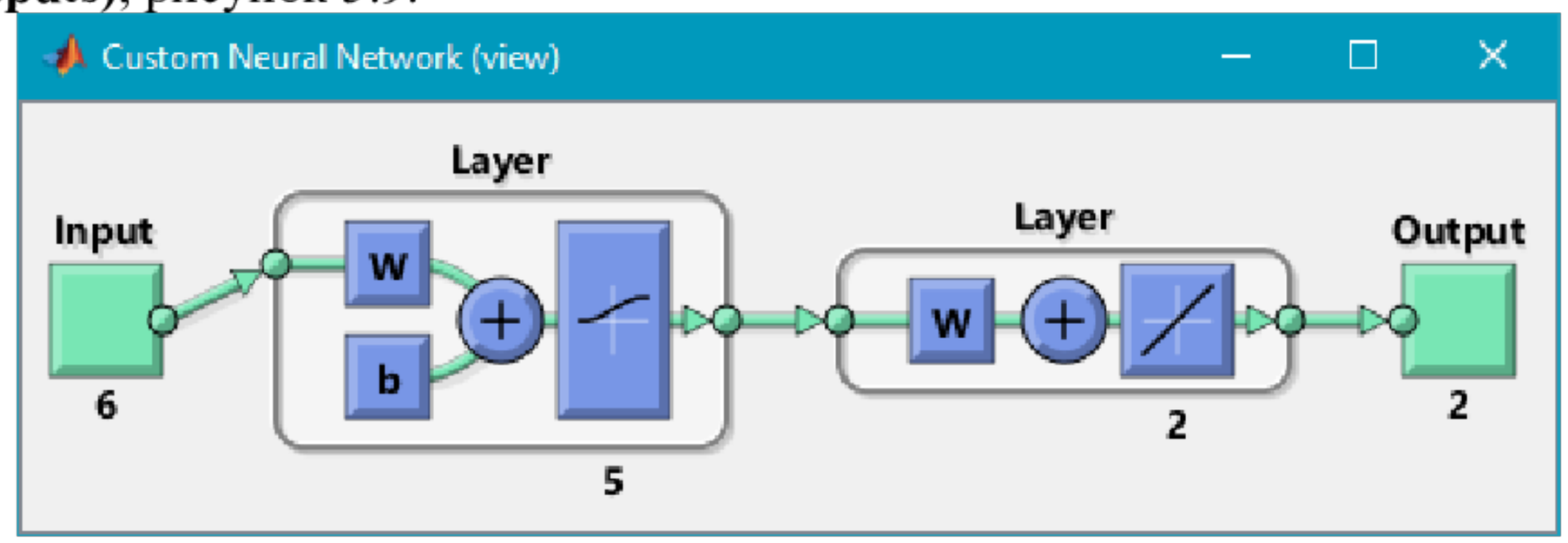


Рисунок 5.9 – Окончательно получаем общую структуру сети, где размер входа – 6, а выхода – 2

Вектор **initial_output** получит значения $[0\ \ 0]^T$, т.к. веса не инициализированы, т.е. сеть ничего делать не умеет. Для начала обучения нужно как минимум задать метод обучения и функцию ошибки, что и делается с помощью **net.trainFcn = 'trainlm'** и **net.performFcn = 'mse'**. **Trainlm** – это обучение по методу Левенберга-Марквардта, а **MSE** (mean square error) – это среднеквадратическая ошибка. **Train()** – запускает обучение сети и возвращает объект – обученную сеть. Процесс обучения показан на рисунке 5.10.

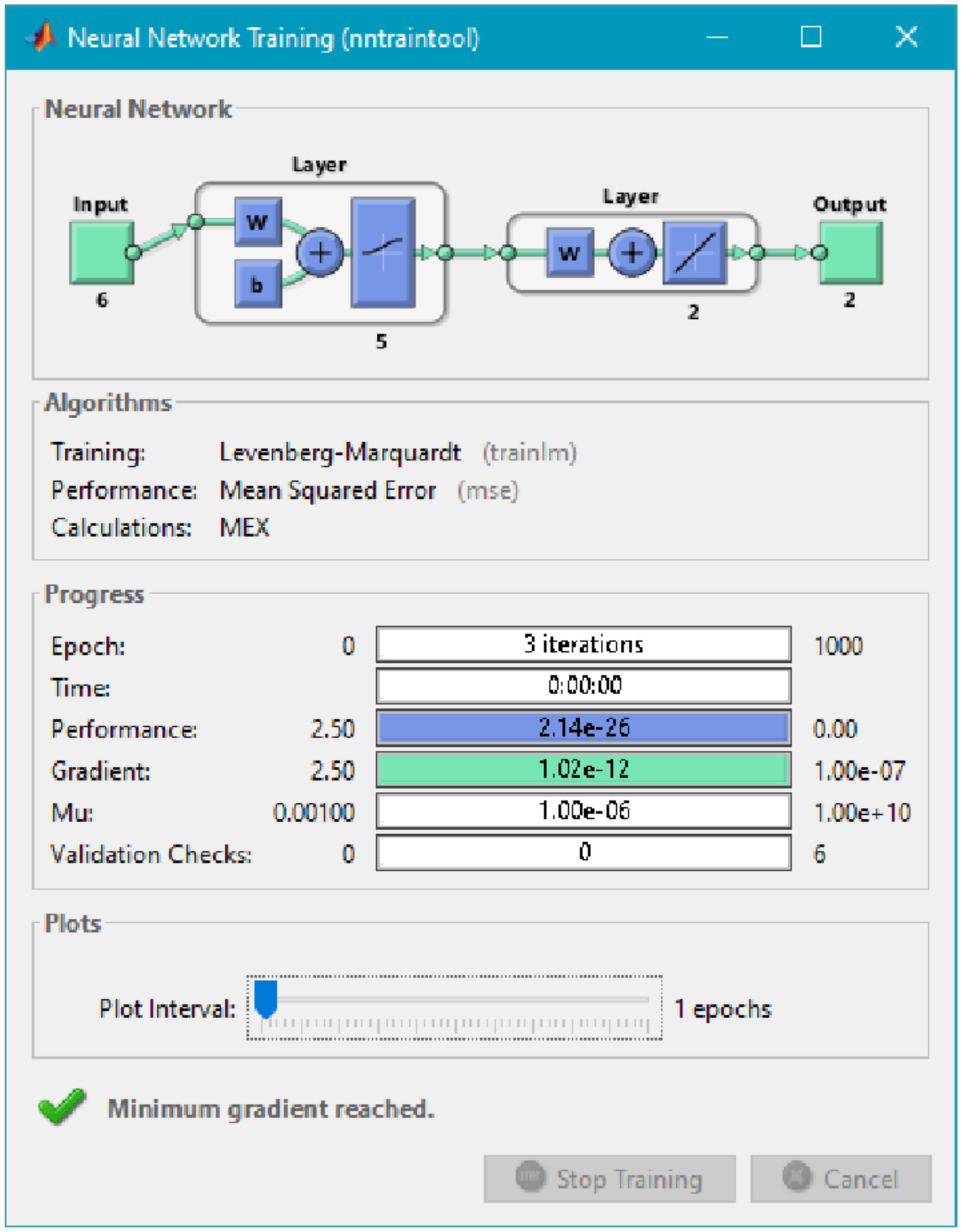


Рисунок 5.10 – Обучение сети

Позже формируем вектор **final_output**, который содержит ответ сети на вход, т.е. 1.0000 и 2.0000 (помним про формат **compact**).

Теперь решим задачу о разделении классов с помощью персептрона, подобную той, которая была решена в лабораторной работе 3.

Пример 3. Разделение классов с помощью персептрона.

```
close all, clear all, clc, format compact
% Задаём количество паттернов каждого класса
N = 20;
% Задаём смещение
offset = 5;
% Создаём входные значения каждого класса
x = [randn(2, N) randn(2, N)+offset];
% Создаём выходные значения для этих классов
y = [zeros(1, N) ones(1, N)];
% Открываем окно для рисования
figure(1);
```

```
% Наносим точки (паттерны)
plotpv(x, y);
% Создаём объект-сеть типа Персептрон
net = perceptron;
% Обучаем персептрон
net = train(net, x, y);
% Рисуем общую структуру сети
view(net);
% Возвращаемся в окно для рисования и рисуем прямую линию,
% которая разделит два класса
figure(1);
plotpc(net.IW{1}, net.b{1});
```

rand(2, N) – создаёт массив размером 2xN, числа которого являются случайные величины, распределённые по нормальному закону с математическим ожиданием 0 и среднеквадратическим отклонением 1. Таким образом, будут созданы две матрицы 2xN, содержащие координаты точек-паттернов в двумерном пространстве, причём координаты второй матрицы будут смещены на 5 позиций, т.е. заранее гарантируется, что классы будут линейно разделимые. Обратим внимание, что эти две матрицы на самом деле записаны в одну матрицу размером 2x2*N. **Zeros(1, N)** и **ones(1, N)** создаёт массив нулей и единиц размером N, это метки для входных паттернов.

Figure(1) – вызывает окно, содержащее канву для рисования, рисунок 5.11.

Plotpv(x, y) – наносит точки на канву с координатами x и метками y, рисунок 5.12. Причём нули отображаются кружками, а единицы плюсами.

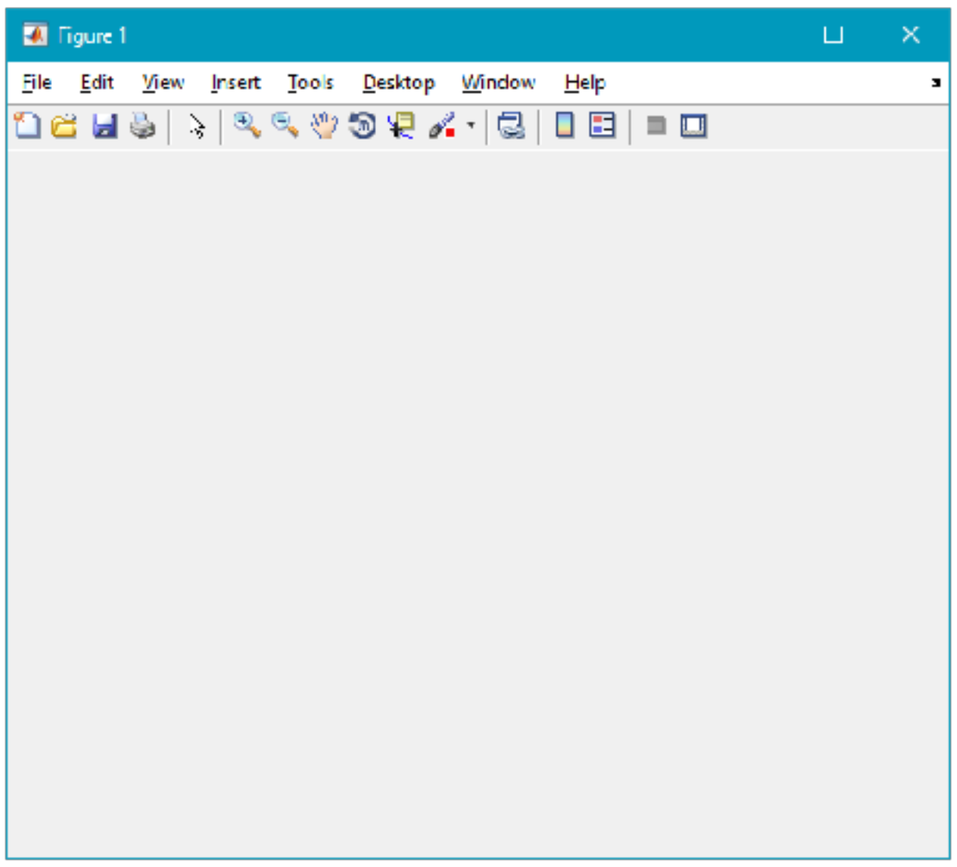


Рисунок 5.11 – Окно для рисования

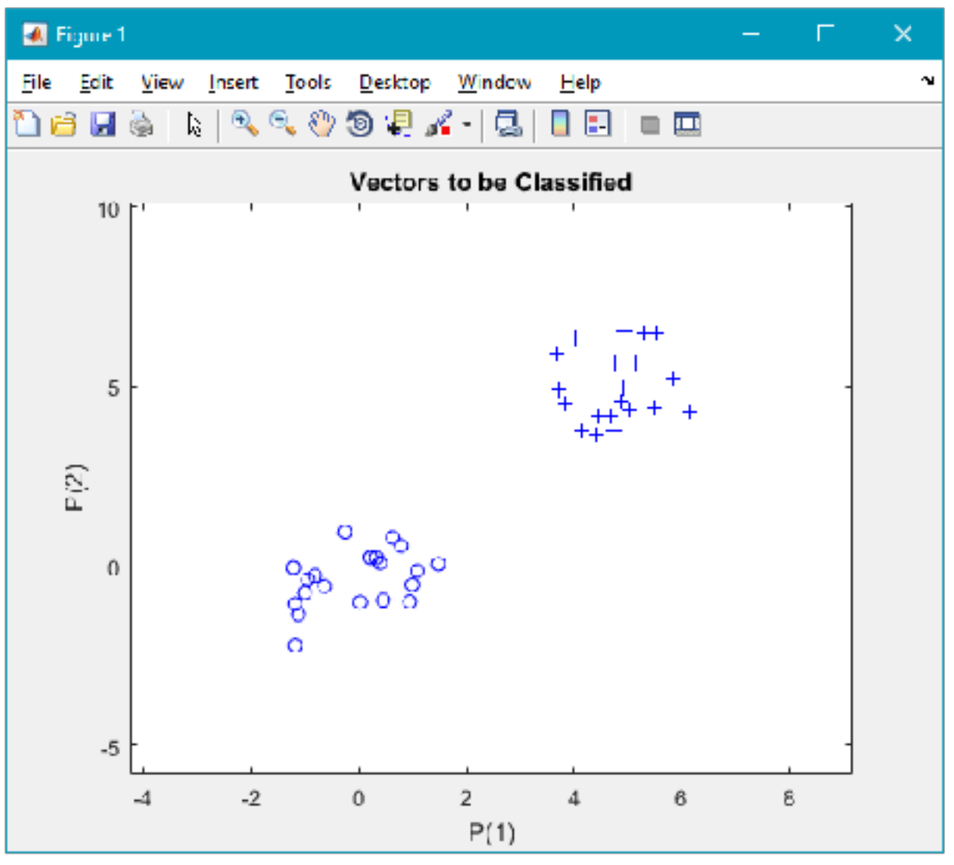


Рисунок 5.12 – Два созданных нами класса, видно, что они линейно разделимы

Net = perceptron – создаём сеть типа Персептрон, если после записи команды не ставить точку с запятой, то среда развернёт все установленные по умолчанию параметры, рисунок 5.13.

```
Command Window
>> net=perceptron
net =
    Neural Network
              name: 'Perceptron'
          userdata: (your custom info)
    dimensions:
         numInputs: 1
         numLayers: 1
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 0
        sampleTime: 1
    connections:
       biasConnect: true
     inputConnect: true
     layerConnect: false
    outputConnect: true
    subobjects:
             input: Equivalent to inputs{1}
            output: Equivalent to outputs{1}
            inputs: {1x1 cell array of 1 input}
            layers: {1x1 cell array of 1 layer}
           outputs: {1x1 cell array of 1 output}
            biases: {1x1 cell array of 1 bias}
      inputWeights: {1x1 cell array of 1 weight}
      layerWeights: {1x1 cell array of 0 weights}
    functions:
          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividetrain'
       divideParam: (none)
        divideMode: 'sample'
           initFcn: 'initlay'
```

Рисунок 5.13 – Методы и свойства объекта perceptron

Далее происходит обучение сети. Общая структура сети представлена на рисунке 5.14.

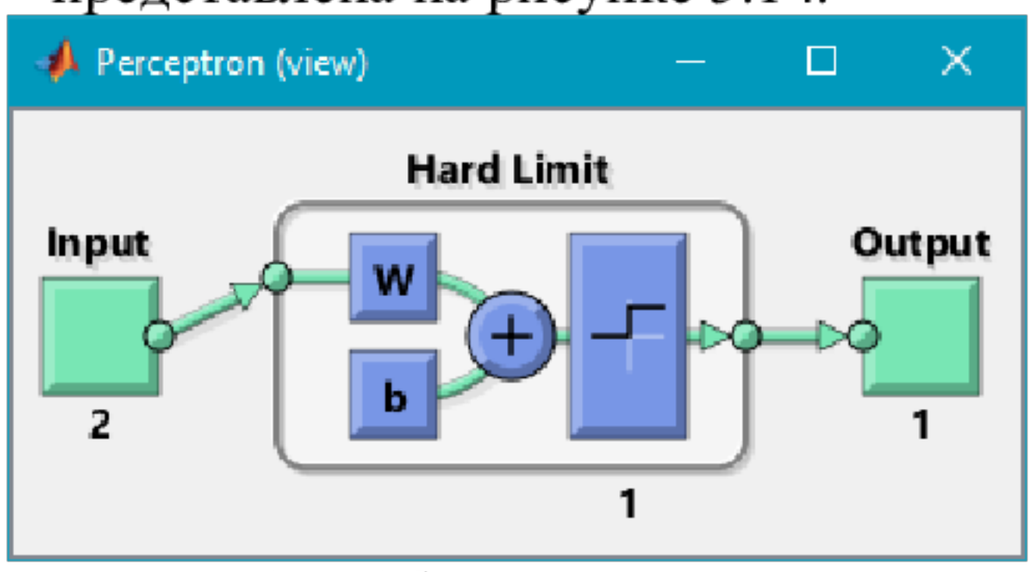


Рисунок 5.14 – Общая структура сети

Результаты обучения представлены на рисунке 5.15. Видно, что потребовалось 29 итераций.

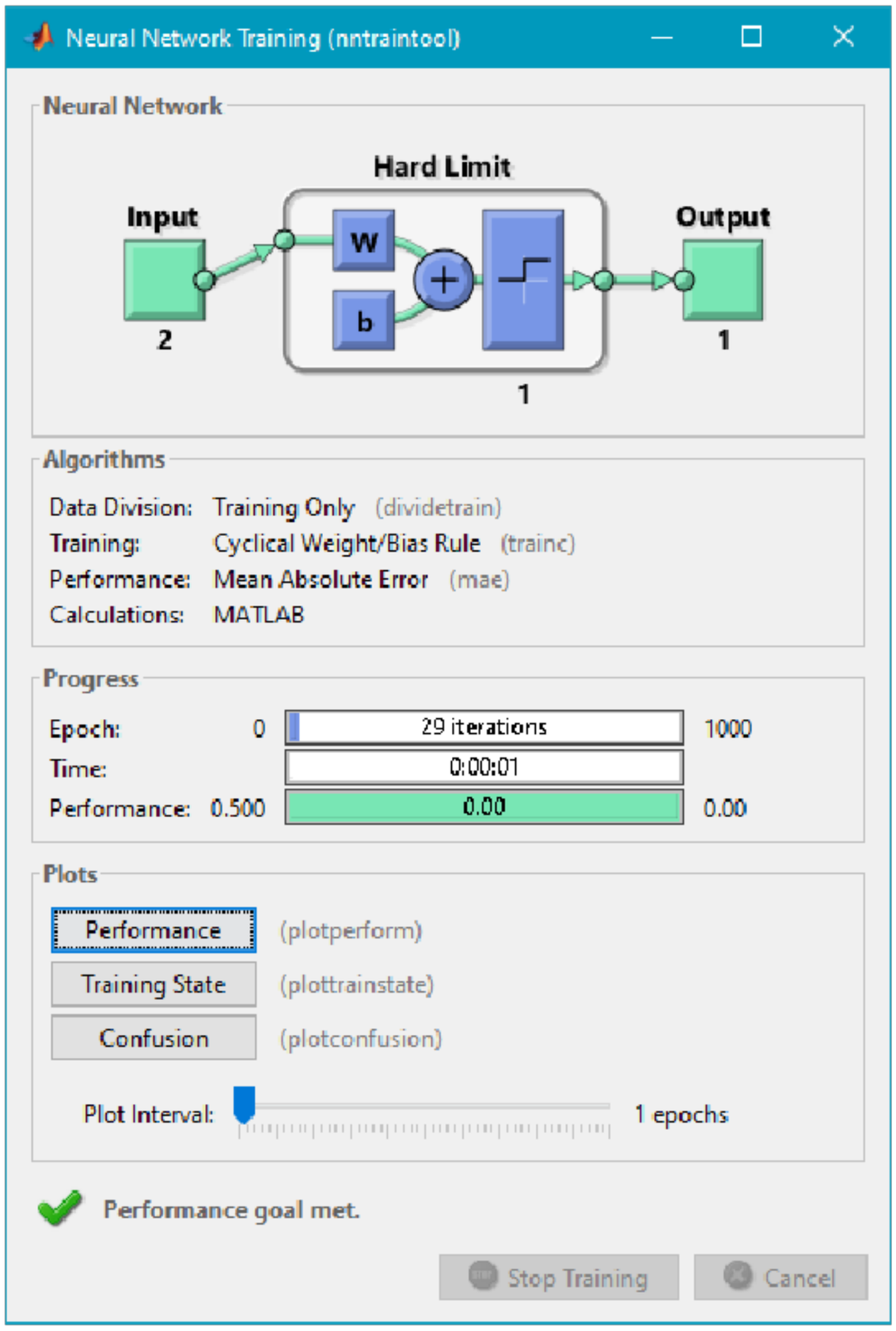


Рисунок 5.15 – Результаты обучения персептрона

На рисунке 5.16 представлен график настройки персептрона. График носит характерную зубчатую форму. После каждой эпохи высчитывается MSE между предполагаемыми ответами и реальными. Напомним, что при адаптации не используется спуск по поверхности ошибки, поэтому ждать постепенного снижения ошибки обучения не следует. Обучение идёт до тех пор, пока не будет найдена первая прямая, отделяющая один класс от другого.

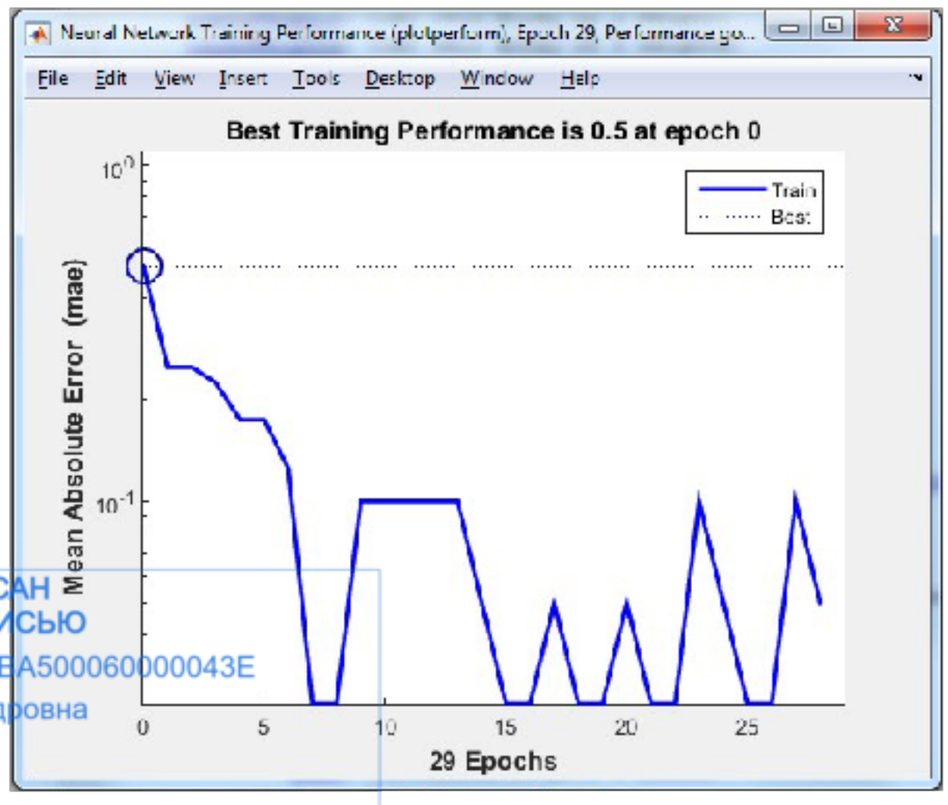


Рисунок 5.16 – График настройки весовых коэффициентов персептрона

Plotpc(net.IW{1}, net.b{1}) – строит прямую линию с соответствующими коэффициентами. **IW** – обозначает слой коэффициентов между входом и первым слоем, **b{1}** – смещение первого слоя. Итоговая прямая показана на рисунке 5.17.

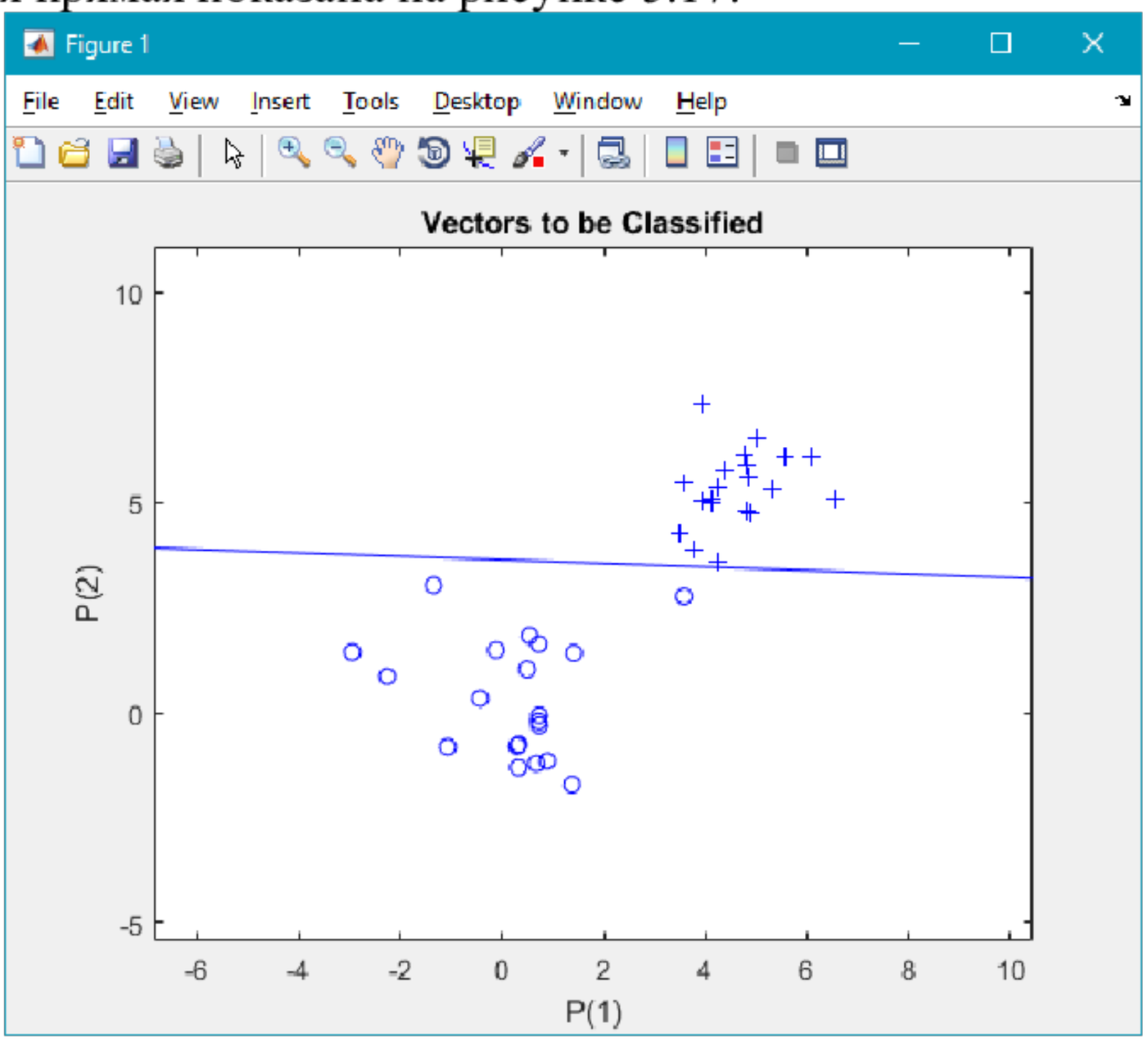


Рисунок 5.17 – Полученное решение по разделению двух классов

Теперь рассмотрим решение задачи разделения на 4 класса с помощью персептрона и обычной многослойной сети прямого распространения. Расположение классов оставим таким же, как и в задаче XOR (по углам «квадрата»), но теперь у нас не два класса, а 4, а также каждый класс представлен 30 паттернами.

Пример 4. Решение задачи разделения на 4 класса с помощью персептрона и обычной многослойной сети прямого распространения.

```
close all, clear all, clc, format compact
    % Каждый класс имеет 30 паттернов
    K = 30;
    % Смещение для классов 0.6
    q = .6;
    % Задаём 4 класса в виде векторов A, B, C, D (2 строки и 30 столбцов)
    A = [rand(1, K)-q; rand(1, K)+q];
    B = [rand(1, K)+q; rand(1, K)+q];
    C = [rand(1, K)+q; rand(1, K)-q];
    D = [rand(1, K)-q; rand(1, K)-q];
    % Рисуем на канве точки A определённого типа (третий параметр)
    plot(A(1, :), A(2, :), 'bs');
```

```
% Фиксируем канву
hold on;
grid on;
% На той же канве рисуем остальные классы
plot(B(1, :), B(2, :), 'r+');
plot(C(1, :), C(2, :), 'go');
plot(D(1, :), D(2, :), 'm*');
text(.5-q, .5+2*q, 'Class A');
text(.5+q, .5+2*q, 'Class B');
text(.5+q, .5-2*q, 'Class C');
text(.5-q, .5-2*q, 'Class D');
        % Задаём кодировку классов (метки)
        a = [0 1]';
        b = [1 1]';
        c = [1 0]';
        d = [0 0]';
        % Получаем общий вектор входных значений
        P = [A B C D];
        % Получаем общий вектор меток
        T = [repmat(a, 1, length(A)) repmat(b, 1, length(B)) repmat(c, 1, length(B))
repmat(d, 1, length(D))];
        % Сеть типа «персептрон»
        net = perceptron;
        % Пусть среднеквадратическая ошибка не равна 0
        E = 1;
        % Установка параметра о том, что будет минимум один прогон
net.adaptParam.passes = 1;
% Построили изначально ответ персептрона, до обучения
linehandle = plotpc(net.IW{1}, net.b{1});
% Счётчик по циклам
n = 0;
% Цикл будет идти пока среднеквадратическая ошибка не
% станет равной нулю или пока не достигнем значения итераций в 1000
while (sse(E) & n<1000)
        n = n + 1;
% Обучить персептрон
        [net, Y, E] = adapt(net, P, T);
% Нарисовать линии
linehandle = plotpc(net.IW{1}, net.b{1}, linehandle);
drawnow;
end
view(net);
% Проверка работы персептрона на входном векторе
p = [0.7; 1.2];
% Выдаст класс (1; 1)
y = net(p);
```

rand(1, K) – генерирует числа из равномерного распределения в виде вектора (первый параметр) размером K, в итоге **A = [rand(1, K)-q; rand(1, K)+q]** генерируется матрица 2xK, что соответствует координатам точек.

Plot(A(1, :), A(2, :), 'bs') – рисует на канве точки из матрицы A, переводя их в вещественный формат. Параметр **'bs'** – это цвет и форма

точек: b – синий и s – квадрат. **'r+'** – красный (r) крест (+), **'go'** – зелёный (g) кружок (o), **'m*'** – пурпурная (m) звездочка (*).

Text(.5-q, .5+2*q, 'Class A') – наносит текст на канву с координатами, записанными как два первых параметра.

Общий вид отображения классов представлен на рисунке 5.18.

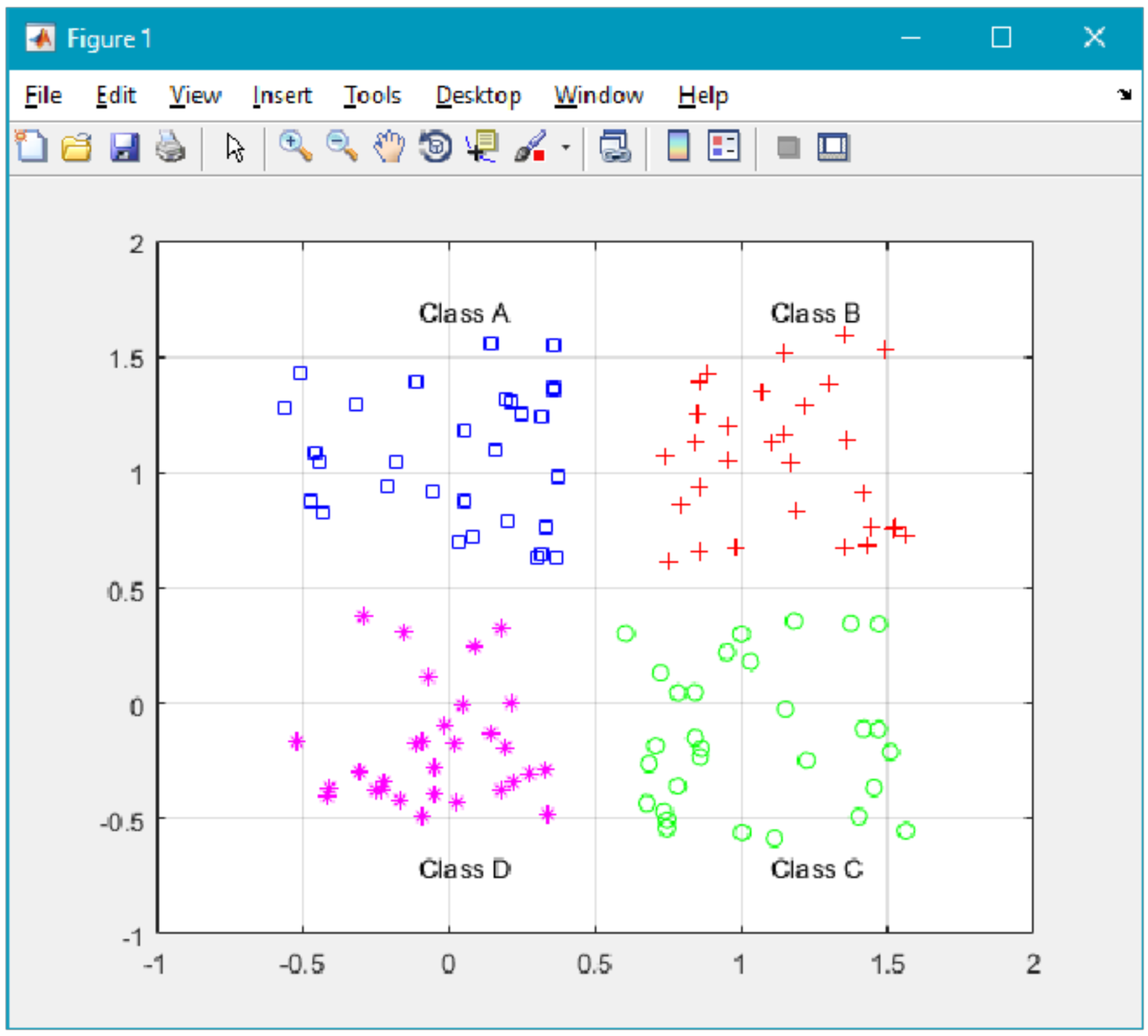


Рисунок 5.18 – Классы, которые предстоит разделить

Для получения вектора T необходимо раскопировать транспонированные вектора **a**, **b**, **c**, **d**. Для этой цели используется **repmat()**: **repmat (что копируем, копий по строкам, копий по столбцам)**. В результате вектор **T** примет вид как на рисунке 3.19: матрица из двух строк и 30 * 4 = 120 колонок.

```
Command Window
>> T

T =

  Columns 1 through 13

     0     0     0     0     0     0     0     0     0     0     0     0     0
     1     1     1     1     1     1     1     1     1     1     1     1     1

  Columns 14 through 26

     0     0     0     0     0     0     0     0     0     0     0     0     0
     1     1     1     1     1     1     1     1     1     1     1     1     1

  Columns 27 through 39

     0     0     0     0     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1     1     1

  Columns 40 through 52

     1     1     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1     1     1     1

  Columns 53 through 65

     1     1     1     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     0     0     0     0     0

  Columns 66 through 78

     1     1     1     1     1     1     1     1     1     1     1     1     1
     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 79 through 91
```

Рисунок 5.19 – Общий вид вектора меток T

Цикл **while()** – это цикл с предусловие, поэтому он может не выполниться ни разу, а следовательно, перед ним необходимо построить возможное решение (т.к. при создании сети **net = perceptron**) веса получают случайные значения.

[net, Y, E] = adapt(net, P, T) – процесс адаптации весов. **Net** – получит обученную сеть, **Y** – вектор ответов сети для входа **P**, **E** – ошибки сети для меток **T** и входов **P**. **Sse(E)** – сумма квадратов этих ошибок.

На рисунке 3.20 показана структура сети, видно, что единственный слой сети содержит два нейрона, что соответствует двум прямым линиям для разделения 4-х классов.

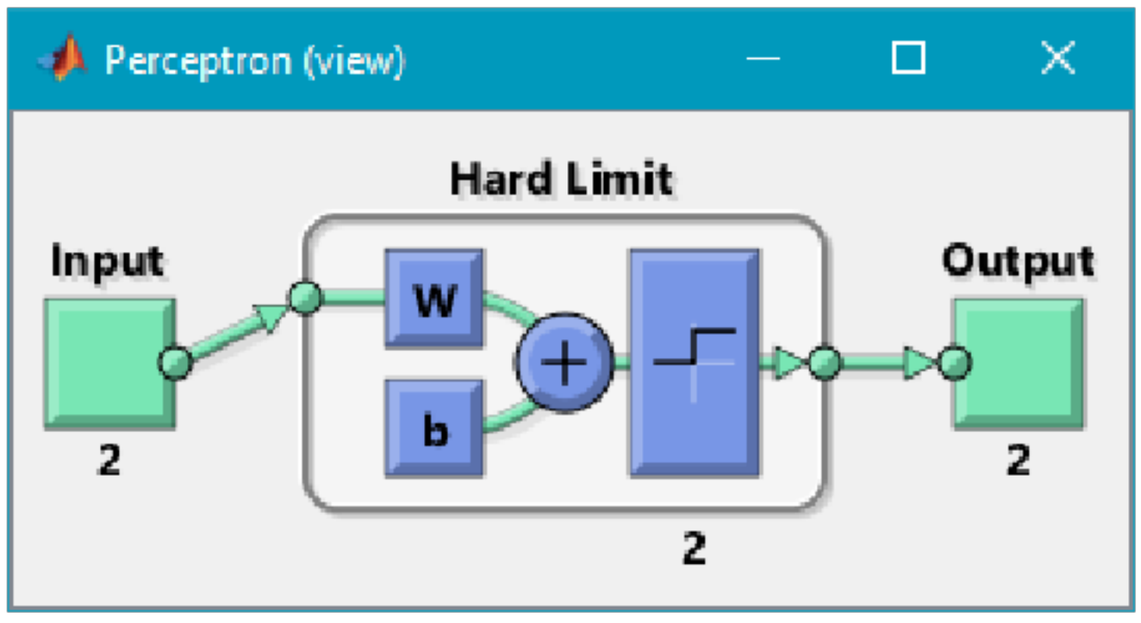


Рисунок 5.20 – Общая структура персептрона

Возможный ответ сети приведён на рисунке 5.21.

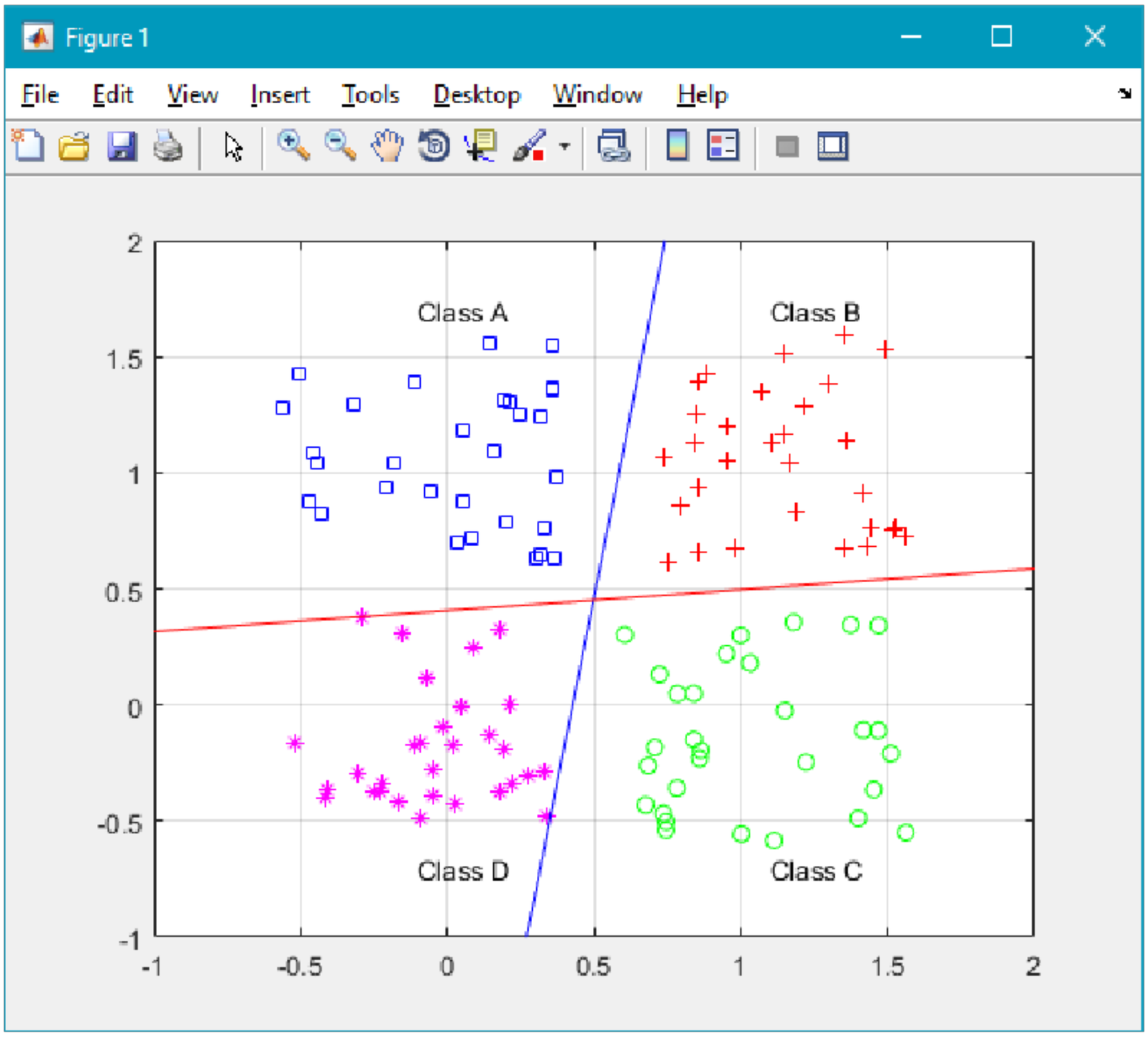


Рисунок 5.21 – Решения задачи разбития на 4 класса

Рассмотрим решение XOR-проблемы с помощью многослойной сети прямого распространения, но расширим представление каждого класса до 100 паттернов.

Пример 5. Решение примера 4 с помощью многослойной сети прямого распространения**.**

```
close all, clear all, clc, format compact
K = 100;
q = .6;
% Данные для обучения
%Обратите внимание на то, что randn() отделяются с помощью
%";". Если опустить этот символ, то A будет не 2x100, а 1x200.
A = [rand(1, K)-q; rand(1, K)+q];
B = [rand(1, K)+q; rand(1, K)+q];
C = [rand(1, K)+q; rand(1, K)-q];
D = [rand(1, K)-q; rand(1, K)-q];
figure(1);
% Наносим точки на канву
plot(A(1, :), A(2, :), 'k+');
hold on;
grid on;
plot(B(1, :), B(2, :), 'bd');
plot(C(1, :), C(2, :), 'k+');
plot(D(1, :), D(2, :), 'bd');
% Наносим названия классов
```