Министерство сельского хозяйства Российской Федерации
ФГБОУ ВО «Красноярский государственный аграрный университет»

А. Ф. Семенов

МОДЕЛИРОВАНИЕ В АГРОИНЖЕНЕРИИ

Рекомендовано Учебно-методическим советом федерального государственного бюджетного образовательного учреждения высшего образования «Красноярский государственный аграрный университет» для внутривузовского использования в качестве учебного пособия для студентов, обучающихся по направлению 35.04.06 «Агроинженерия» направленности (профиль) «Электрооборудование и электротехнологии в АПК»

Электронное издание

Красноярск 2024

ББК 40.7с51
С 30

Рецензенты:

В. Б. Белый, кандидат технических наук, доцент кафедры электрификации и автоматизации сельского хозяйства ФГБОУ ВО «Алтайский государственный аграрный университет»

В. Д. Очиров, кандидат технических наук, доцент, заведующий кафедрой энергообеспечения и теплотехники ФГБОУ ВО «Иркутский государственный аграрный университет»

С 30 **Семенов, А. Ф.**
Моделирование в агроинженерии [Электронный ресурс]: учебное пособие / А. Ф. Семенов; Красноярский государственный аграрный университет. – Красноярск, 2024. – 213 с.

Представлен теоретический материал, необходимый для освоения навыков моделирования с помощью программы Matlab. Приведены пошаговые методики самостоятельного выполнения лабораторных работ.

Предназначено для студентов, обучающихся по направлению подготовки 35.04.06 «Агроинженерия» направленность (профиль) «Электрооборудование и электротехнологии в АПК».

ББК 40.7с51

ВВЕДЕНИЕ

Дисциплина «Моделирование в агроинженерии» является частью цикла естественно-научных дисциплин по направлению подготовки магистров. Содержание дисциплины охватывает круг вопросов, связанных с решением инженерно-технических задач.

Преподавание дисциплины предусматривает следующие формы организации учебного процесса: лабораторные работы, самостоятельная работа студента, консультации. Программой дисциплины предусмотрены следующие виды контроля: текущий контроль успеваемости в форме защиты лабораторных работ, проверки домашних заданий и промежуточный контроль в форме аттестации студентов.

Моделирование в агроинженерии – комплексное научное направление, имеющее междисциплинарный характер, активно содействующее развитию других научных направлений и тем самым выполняющее и интегративную функцию в системе наук.

Цели преподавания дисциплины:

– дать целостное представление об информатике и ее роли в развитии общества;

– раскрыть суть и возможности технических и программных средств информатики;

– сформировать понимание, с какой целью и каким образом можно использовать информационные системы и технологии.

В результате изучения дисциплины студент должен **знать** законы и методы математики, естественных, гуманитарных и экономических наук для решения стандартных и нестандартных профессиональных задач.

Также студент должен **уметь** использовать основы алгоритмизации и программирования для решения задач на современных ЭВМ; рассчитывать и оценивать условия и последствия (в том числе экологические) принимаемых организационно-управленческих решений в области технического и энергетического обеспечения высокоточных технологий производства сельскохозяйственной продукции; применять знания о современных методах исследований.

Кроме того, студент должен **владеть** логическими методами и приемами научного исследования; способностью анализировать современные проблемы науки и производства в агроинженерии, а также вести поиск их решения.

ИНСТРУКЦИЯ ПО ОХРАНЕ ТРУДА ПРИ ЭКСПЛУАТАЦИИ ПЭВМ, КРАСНОЯРСКИЙ ГАУ-СМК-ИОТ-7.1.4-0.0-03-2018

1. Общие требования безопасности

1.1. Настоящая инструкция разработана для работников, занятых эксплуатацией и использованием персонально-вычислительных машин (далее ПЭВМ):

– пользователей ПЭВМ, совмещающих работу оператора с основной работой и занятых работой с компьютером основную часть своего рабочего времени;

– работников, трудовая деятельность которых связана с приемом и вводом информации, наблюдением и корректировкой решаемых задач по готовым программам, программистов, занятых на компьютерах разработкой, проверкой, отладкой программ;

– инженеров и техников по компьютерам, производящих профилактические и ремонтные работы, устанавливающих причины сбоев, работающих со схемами и другой документацией.

1.2. К самостоятельной работе с компьютерами допускаются пользователи, прошедшие при приеме на работу психиатрическое освидетельствование, медицинский осмотр, вводный инструктаж в службе охраны труда, аттестованные на I квалификационную группу по электробезопасности, которая подтверждается ежегодно в установленном порядке, и прошедшие обучение по охране труда и оказанию первой помощи пострадавшим, а также ознакомленные с картой специальной оценки условий труда на рабочем месте.

1.3. Женщины со времени установления беременности и в период кормления ребенка грудью к выполнению всех видов работ, связанных с использованием ПЭВМ, не допускаются или для них ограничивается время работы с ПЭВМ (не более 3 часов за рабочий день (смену)).

1.4. Первичный инструктаж на рабочем месте проводит непосредственный руководитель при приеме на работу, а затем каждые шесть месяцев должен проводиться повторный инструктаж. Все виды инструктажей регистрируются в журнале регистрации инструктажа на рабочем месте.

1.5. В процессе работы пользователи ПЭВМ и оргтехникой должны соблюдать правила внутреннего трудового распорядка.

1.6. Работник должен:

– знать инструкции по эксплуатации ПЭВМ и другой оргтехники;

– знать места подключения токоприемников, коммутирующих устройств, а также уметь определять их исправное состояние;

– знать пути эвакуации персонала и действия в случае возникновения аварийных ситуаций;

– знать места нахождения средств пожаротушения и уметь их применять;

– уметь оказывать первую помощь пострадавшим.

1.7. При эксплуатации ПЭВМ и другой оргтехники на работника могут оказывать действие следующие опасные производственные факторы:

– повышенный или пониженный уровень освещенности;

– повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека;

– напряжение зрения, внимания, длительные статические нагрузки.

1.8. Первичными средствами пожаротушения, доступными работнику, являются песок, кошма и ручные огнетушители. Углекислотные огнетушители позволяют тушить легковоспламеняющиеся жидкости и электрооборудование напряжением до 1 кВ (1000 В) без снятия напряжения.

1.9. Пользователи ПЭВМ должны соблюдать правила электро- и пожаробезопасности.

1.10. Лица, допустившие нарушение инструкций по охране труда, подвергаются дисциплинарному взысканию в соответствии с правилами внутреннего трудового распорядка и при необходимости внеочередной проверке знаний требований охраны труда.

2. Требования безопасности перед началом работы

2.1. Работнику запрещается приступать к работе при обнаружении неисправности оборудования.

2.2. Каждый работник перед началом работы обязан:

2.2.1. При необходимости проветрить помещение, устранить повышенную подвижность воздуха (сквозняки) и т. д.

2.2.2. Осмотреть и привести в порядок рабочее место:

– отрегулировать освещенность на рабочем месте и убедиться в ее достаточности, проверить правильность установок стола, стула, при необходимости произвести их регулировку.

2.3. Дополнительные требования при работе с любыми токоприемниками:

2.3.1. Убедиться в наличии защитного заземления.

2.3.2. Проверить правильность подключения используемых токоприемников в электросеть.

2.3.3. Проверить исправность проводящих проводов и отсутствие оголенных участков проводов.

2.3.4. Производить включение электрооборудования в сеть путем вставки исправной вилки в исправную розетку.

2.3.5. Запрещается производить протирание влажной или мокрой салфеткой электрооборудования, которое находится под напряжением (вилка вставлена в розетку). Влажную или любую другую уборку производить только при отключенном оборудовании.

2.3.6. Работник должен убедиться, что включенное им оборудование никого не подвергает опасности.

2.4. Дополнительные требования при работе с ПЭВМ:

2.4.2. Проверить правильность угла наклона экрана, положение клавиатуры, положение компьютерной мыши на специальном коврике, расположение элементов компьютера в соответствии с требованиями эргономики и в целях исключения неудобных поз и длительного напряжения тела.

2.5. Работник обязан соблюдать последовательность включения ПК:

– включить блок питания;

– включить периферийные устройства (принтер, монитор, сканер и др.);

– включить системный блок.

2.6. Работник обязан сообщить руководителю подразделения, службы или участка об обнаруженной неисправности оборудования и приступить к работе после устранения нарушений в работе или неисправностей оборудования.

2.7. Следует иметь в виду, что монтаж сетей 36, 220 и 380 В для подключения электрооборудования производит электротехнический персонал (электрослесарь, электротехник).

3. Требования безопасности во время работы

3.1. Работник во время работы ОБЯЗАН:

– выполнять только ту работу, которая ему была поручена и по которой он был проинструктирован, не допуская при этом спешки с учетом безопасных приемов и методов работы;

– в течение всего рабочего времени содержать в порядке и чистоте рабочее место;

– соблюдать правила эксплуатации ПЭВМ и оргтехники в соответствии с инструкциями по эксплуатации;

– держать открытыми вентиляционные отверстия, которыми оборудованы приборы.

3.2. Работнику во время работы ЗАПРЕЩАЕТСЯ:

– прикасаться к задней панели системного блока (процессора) при включенном питании (переключать разъемы интерфейсных кабелей периферийных устройств при включенном питании);

– загромождать верхние панели устройств бумагами и посторонними предметами;

– допускать захламленность рабочего места бумагой;

– производить частые переключения питания;

– допускать попадание влаги на поверхность системного блока, монитора, рабочую поверхность клавиатуры, дисководов, принтеров и других устройств;

– включать сильно охлажденное (принесенное с улицы в зимнее время) оборудование;

– самостоятельно производить вскрытие и ремонт оборудования.

3.3. Работник обязан отключить компьютер от электросети при возникновении следующих обстоятельств:

– при обнаружении неисправности;

– внезапном снятии напряжения электросети;

– во время чистки и уборки оборудования;

– во время работы на компьютере (если работник почувствует хотя бы слабое действие электрического тока, работа должна быть немедленно прекращена и неисправное оборудование должно быть сдано для проверки или ремонта).

3.4. Дополнительные требования во время работы с компьютером:

3.4.1. При любых случаях сбоя в работе технического оборудования или программного обеспечения немедленно сообщить руководителю и обслуживающему инженерно-техническому персоналу.

3.4.2. При необходимости прекращения работы на некоторое время корректно закрыть все активные задачи.

4. Требования безопасности в аварийных ситуациях

4.1. О каждом несчастном случае, связанном с производством, пострадавший или очевидец должны:

– немедленно сообщить соответствующему руководителю и в службу охраны труда университета (тел.: 8 (391) 246-41-54);

– оказать первую помощь пострадавшему, организовать его доставку в лечебное учреждение;

– сохранить для расследования обстановку на рабочем месте и состояние оборудования такими, какими они были в момент происшествия, если это не угрожает здоровью и жизни окружающих работников и не приведет к аварии;

– при расследовании причин несчастного случая работник ОБЯЗАН сообщить известные ему обстоятельства происшедшего случая, свидетелем которого он явился;

– при обнаружении человека, попавшего под напряжение, немедленно освободить его от действия электрического тока путем отключения электропитания и до прибытия врача оказать потерпевшему первую помощь;

– при любых случаях сбоя в работе технического оборудования или программного обеспечения немедленно вызвать представителя инженерно-технической службы эксплуатации ПЭВМ;

– в случае появления рези в глазах, резкого ухудшения зрения, невозможности сфокусировать взгляд или навести его на резкость, появления боли в пальцах и кистях рук, усиления сердцебиения немедленно покинуть рабочее место, сообщить о происшедшем руководителю работ и обратиться к врачу;

– при возгорании оборудования отключить питание и принять меры к тушению очага пожара при помощи углекислого огнетушителя, сообщить о происшествии непосредственному руководителю, при необходимости вызвать пожарную команду.

4.2. Работник обязан во всех случаях обнаружения обрыва проводов питания, неисправности заземления, других повреждений электрооборудования, появления запаха гари; в случае появления резких ухудшений самочувствия, а также в любых других ситуациях, которые, по мнению работника, создают непосредственную угрозу его здоровью и жизни (здоровью) людей, он обязан немедленно прекратить работу, отключить питание и сообщить об аварийной ситуации непосредственному руководителю и в отдел главного энергетика (тел.: 8 (391) 246-41-54).

5. Требования безопасности после окончания работы

5.1. По окончании работ работник обязан соблюсти следующую последовательность выключения ПЭВМ и оргтехники:

– произвести закрытие всех активных задач;

– выключить приборы и оборудование, отключить питание системного блока и других периферийных устройств, за исключением работающих в ждущем режиме (факс, сигнализация и т. п.);

5.2. Осмотреть и привести в порядок рабочее место.

5.3. О замеченных во время работы неисправностях и неполадках доложить непосредственному руководителю.

5.4. Вымыть руки с мылом.

Лабораторная работа № 1
НАЧАЛО РАБОТЫ В СРЕДЕ MATLAB

Цель работы: получение практических навыков, необходимых для решения задач с помощью пакета Matlab.

1.1. История и основные возможности пакета Matlab

Система Matlab – язык программирования высокого уровня, предназначенный для технических вычислений.

Систему Matlab разработал Клив Молер (Cleve Moler), и с конца 70-х гг. XX в. она широко использовалась на больших ЭВМ. В начале 80-х гг. XX в. Дж. Литл (John Little) разработал версию системы PC Matlab для компьютеров класса IBM PC, VAX и Macintosh.

В последнее время в научных и инженерно-технических кругах получила широкое распространение система Matlab. Более того, в настоящее время она принята в качестве официального средства оформления инженерной документации и научных публикаций. В чем же причина ее популярности?

Система Matlab специально создана для проведения инженерных расчетов: математический аппарат, который используется в ней, предельно приближен к современному математическому аппарату инженера и ученого и опирается на вычисления, производимые с матрицами, векторами и комплексными числами; графическое представление функциональных зависимостей здесь организовано в форме, которую требует именно инженерная документация.

Язык программирования системы Matlab весьма прост, он содержит лишь несколько десятков операторов; незначительное количество операторов здесь компенсируется большим числом процедур и функций, содержание которых понятно пользователю, имеющему соответствующую математическую и инженерную подготовку.

В отличие от большинства математических систем Matlab является открытой системой: практически все ее процедуры и функции доступны не только для использования, но и для модификации. Почти все вычислительные возможности системы можно применять в режиме чрезвычайно мощного научного калькулятора, а также составлять собственные программы, предназначенные для многоразового применения; это делает Matlab незаменимым средством при проведении научных исследований. По скорости выполнения задач

Matlab опережает многие другие подобные системы. Все эти особенности делают ее весьма привлекательной для использования.

Система Matlab разработана специалистами компании MathWork Inc. (г. Нейтик, штат Массачусетс, США). Хотя впервые эта система начала использоваться в конце 1970-х гг, широкое распространение она получила в 1980-х, в особенности после появления на рынке версии 4.0. Последние версии Matlab – это системы, которые содержат множество процедур и функций, необходимых инженеру и научному работнику для осуществления сложных численных расчетов, моделирования технических и физических систем и оформления результатов этих расчетов. Matlab (сокращение от MATrix LABoratory – матричная лаборатория) представляет собой интерактивную систему, предназначенную для выполнения инженерных и научных расчетов и ориентированную на работу с массивами данных. Система обеспечивает возможность обращения к программам, которые написаны на языках Fortran, C и C++.

Привлекательной особенностью системы Matlab является наличие встроенной матричной и комплексной арифметики. Система поддерживает выполнение операций с векторами, матрицами и массивами данных, реализует сингулярное и спектральное разложение, расчет ранга и чисел обусловленности матриц, поддерживает работу с алгебраическими полиномами, решение нелинейных уравнений и задач оптимизации, интегрирование функций в квадратурах, численное интегрирование дифференциальных и разностных уравнений, построение различных графиков, трехмерных поверхностей и линий уровня.

Система Matlab обеспечивает выполнение операций с векторами и матрицами даже в режиме непосредственных вычислений. Ею можно пользоваться как мощнейшим калькулятором, в котором наряду с обычными арифметическими и алгебраическими действиями могут использоваться также сложные операции, такие как обращение матрицы, вычисление ее собственных значений и векторов, решение систем линейных алгебраических уравнений и многие другие. Характерной особенностью системы является ее открытость, т. е. возможность ее модификации и адаптации к конкретным задачам пользователя.

Matlab представляет широкие возможности для работы с сигналами, для расчета и проектирования аналоговых и цифровых фильтров, включая построение их частотных, импульсных и переходных характеристик. Имеются в системе и средства выполнения спек-

трального анализа и синтеза, в частности реализация прямого и обратного преобразования Фурье, благодаря этому ее довольно удобно использовать для проектирования электронных устройств.

С системой Matlab поставляется свыше ста подробно прокомментированных М-файлов, которые содержат демонстрационные примеры и определения новых операторов и функций. Наличие этих примеров и возможность работать в режиме непосредственных вычислений значительно облегчает изучение системы пользователями, заинтересованными в выполнении математических расчетов.

Система Matlab использует собственный М-язык, который сочетает в себе положительные свойства различных известных языков программирования высокого уровня. С языком Basic систему Matlab роднит то, что она представляет собой интерпретатор (осуществляет пооператорное компилирование и выполнение программы, не образуя отдельного исполнительного файла), М-язык имеет незначительное количество операторов, в нем отсутствует необходимость объявлять типы и размеры переменных. От языка Pascal система Matlab позаимствовала объектно-ориентированную направленность, т. е. такое построение языка, которое обеспечивает образование новых типов вычислительных объектов на основе типов объектов, уже существующих в языке. Новые типы объектов (в Matlab они называются классами) могут иметь собственные процедуры их преобразования (они определяют методы этого класса), причем новые процедуры могут быть вызваны с помощью обычных знаков арифметических операций и некоторых специальных знаков, которые применяются в математике.

Принципы сохранения значений переменных в Matlab наиболее близки к тем, которые присущи языку Fortran, а именно: все переменные являются локальными, действуют лишь в границах той программной единицы (процедуры, функции или главной, управляющей программы), где им присвоены конкретные значения. При переходе к выполнению другой программной единицы значения переменных предыдущей программной единицы либо теряются (в случае, если выполненная программная единица представляла собой процедуру или функцию), либо становятся недосягаемыми (если выполненная программа является управляющей). В отличие от языков Basic и Pascal в языке Matlab нет глобальных переменных, действие которых распространялось бы на все программные единицы. Но при этом язык Matlab обладает возможностью, которая отсутствует в других языках. Интерпретатор Matlab позволяет в одном и том же сеансе работы вы-

полнять несколько самостоятельных программ, причем все переменные, используемые в этих программах, являются для них общими и образуют единое рабочее пространство. Это дает возможность более рационально организовать сложные (громоздкие) вычисления по типу оверлейных структур.

Вышеуказанные особенности Matlab делают ее весьма гибкой и удобной в использовании вычислительной системой.

С помощью языка Matlab можно решать следующие задачи:

- производить вычисления;
- создавать программы;
- моделировать процессы;
- анализировать, исследовать и визуализировать данные;
- разрабатывать приложения, включающие графический интерфейс.

1.2. Основные объекты системы Matlab

Рабочая область

Математическое выражение, которое должно быть вычислено в численном или символьном виде, строится на основе следующих *объектов* системы Matlab:

- констант;
- переменных;
- операторов и специальных знаков;
- функций.

Все данные, а именно константы и переменные, хранятся в специально отведенной области памяти, называемой *рабочей областью***.** После применения команд присвоения, например:

⟩⟩ **x** = [1 2 3; 4 1 6; 4 5 1];

⟩⟩ **apath** = 'c:\windows';

⟩⟩ **x1** = 2.3

⟩⟩ **xa** = '**s**'

в рабочую область были записаны данные **x**, **apath**, **x1** и **xa.**

Содержимое рабочей области можно просмотреть и очистить. Рассмотрим команды для работы с данными в рабочей области.

Просмотр списка данных

⟩⟩ **who**

В результате в командном окне появится сообщение, содержащее список объектов в рабочей области:

Your variables are:

apath x x1 xa

Для просмотра одного объекта рабочей области можно использовать команду

〉〉 **who x**

или функцию

〉〉 **who('x')**

В результате в командном окне появится сообщение, содержащее список, состоящий из одного объекта, заданного командой или функцией в рабочей области:

Your variables are: x

Для просмотра нескольких объектов рабочей области также можно использовать эту команду:

〉〉 **who x apath**

или функцию

〉〉 **who('x', 'apath')**

В результате в командном окне появится сообщение, содержащее список, состоящий из нескольких объектов, заданных командой или функцией, содержащихся в рабочей области:

Your variables are: apath x

Можно просмотреть таблицу данных, содержащихся в рабочей области. В таблице выводится: имя переменной или константы – **Name**, размерность массива, который описывает данная переменная (в частном случае массив состоит из одного элемента) – **Size**, объем занимаемой памяти – **Bytes** и тип данных в массиве – **Class**:

〉〉 **whos**

В результате в командном окне появится таблица с информацией обо всех объектах, содержащихся в рабочей области, а также количество объектов и память в байтах:

Name	Size	Bytes	Class
apath	1x10	20	char array
x	3x3	72	double array
x1	1x1	8	double array
xa	1x1	2	char array
Grand total is 27 elements using 294 bytes			

Эти команду или функцию можно применить к одной или нескольким переменным по аналогии с командой или функцией **who**, например:

〉〉 **whos x**

или

〉〉 **whos('x')**

Результат:

Name	Size	Bytes	Class
x	3x3	72	double array
Grand total is 9 elements using 72 bytes			

А также:

〉〉 **whos x apath**

или

〉〉 **whos('x', 'apath')**

Результат:

Name	Size	Bytes	Class
apath	1x10	20	char array
x	3x3	72	double array
Grand total is 19 elements using 92 bytes			

В рассмотренных командах и функциях в имени операнда или параметра соответственно можно применять маску для замены одного или нескольких символов, например:

〉〉 **who apath x***

или

〉〉 **who('apath', 'x*')**

В результате, кроме переменной **apath**, список будет содержать также переменные, имена которых начинаются на **x**:

Your variables are:

apath x x1 xa

Также можно применить маску для нескольких символов:

〉〉 **whos a*h**

или

〉〉 **whos('a*h')**

Результат:

Name	Size	Bytes	Class
apath	1x10	20	char array
Grand total is 1 elements using 20 bytes			

Для очистки рабочей области можно использовать команду

〉〉 **clear**

Применение такой команды приведет к удалению из памяти всех данных.

Эту команду или функцию можно применить к одной переменной:

〉〉 **clear x**

или

〉〉 **clear('x')**

Произойдет удаление из памяти только переменной **x.**

А также для списка переменных:

〉〉 **clear x apath**

или

〉〉 **clear('x', 'apath')**

Результат – удаление из памяти переменных **x** и **apath**.

В имени операнда или параметра данной команды и функции соответственно также можно применять маску для замены одного или нескольких символов, например:

〉〉 **clear apath x***

или

〉〉 **clear('apath', 'x*')**

Результат – удаление из памяти переменных **apath**, **x**, **x1** и **xa**.

Дефрагментация рабочей области необходима после удаления из памяти переменных. Она производится с помощью команды **pack.**

Другой способ работы с объектами рабочей области предоставляет окно «Workspace», которое можно активизировать с помощью пункта главного меню Matlab «Desktop/Workspace». Данное окно содержит таблицу со списком всех объектов рабочей области с указанием объема занимаемой объектами памяти и их типом. Функции окна «Workspace» позволяют изменять имена переменных, инициализировать новые переменные, считывать данные с диска, сохранять данные на диске, удалять переменные, распечатывать их значения на принтере, строить графики, а также инициализировать окно «Array Editor», с помощью которого можно редактировать значения численных массивов так, как это делается с помощью электронной таблицы «Excel».

Константы – в системе Matlab применяется два типа констант:

– численные;

– символьные.

Численные константы содержат численные данные.

Вещественные числа в системе Matlab представлены с двойной точностью, занимают 8 байт памяти компьютера.

Комплексные числа хранятся в памяти в форме вектора, состоящего из двух вещественных чисел, первое из которых интерпретируется как вещественная часть мнимого числа, вторая – как коэффициент при мнимой единице. Таким образом каждая комплексная константа занимает 16 байт. Мнимая единица обозначается через **i** или **j**.

Примеры ввода комплексных чисел:

〉〉 **3i**

〉〉 **2 + 3i**

〉〉 **-3.123j**

〉〉 **-123.456 + 7.8i.**

Для вывода в зону просмотра командного окна предусмотрены различные форматы чисел. Назначение формата отражается только на форме вывода чисел. Объем памяти, занимаемый числом, не изменяется. Некоторые из предусмотренных в системе Matlab форматов вывода чисел и примеры вывода числа π в различных форматах:

– **bank** – десятичное число с двумя разрядами после точки. Вид числа π, выведенного в данном формате: 3.14;

– **hex** – шестнадцатеричное число. Число π в данном формате выглядит: 400921fb54442d18;

– **long** – пятнадцатиразрядное число с фиксированной точкой. Число π: 3.14159265358979;

– **long e** – пятнадцатиразрядное число с плавающей точкой. Число π: 3.141592653589793e + 000;

– **long g** – улучшенный вывод 15 знаков числа с плавающей или фиксированной точкой. Число π: 3.14159265358979;

– **rat** – отношение наименьших целых чисел. Число π: 355/113;

– **short** – пятиразрядное число с фиксированной точкой. Число π: 3.1416;

– **short e** – пятиразрядное число с плавающей точкой. Число π: 3.1416e + 000;

– **short g** – улучшенный вывод 5 знаков числа с плавающей или фиксированной точкой. Число π: 3.1416;

– без параметра – установка формата вывода чисел, принятого по умолчанию (**short**). Число π: 3.1416;

– **+** – только «+», если число положительное, или «–», если число отрицательное. Число **π**: +.

Установить формат вывода чисел можно двумя способами:

– с помощью следующей команды:

〉〉 **forma**t <указывается выбранный формат>

Например:

〉〉 **format short e**

С помощью пункта «File/References» главного меню открыть окно с заголовком «References», в котором, установив курсор на строку «Command Window», из раскрывающегося списка можно выбрать необходимый формат вывода численных данных.

Символьные константы. Значение символьной константы необходимо задавать в апострофах, например:

〉〉 **'z'**

или

〉〉 **simbol = 'z'**

Одна символьная константа занимает 2 байта памяти компьютера.

Системные переменные – константы, которые задаются системой при ее загрузке, но могут переопределяться. Примеры системных переменных:

– **i** и **j** – мнимая единица;

– **pi** – число π = 3,1415926…;

– **Eps** – погрешность для операций над числами с плавающей точкой 2.2204×10-016;

– **Realmin** – наибольшее число с плавающей точкой 2.2251×10^{-308};

– **Realmax** – наименьшее число с плавающей точкой $1.7977\times10^{+308}$;

– **Inf** – значение машинной бесконечности;

– **ans** – переменная, хранящая результат последней операции. Например, в результате команды

〉〉 **'z'**

переменная «**Ans**» примет '**z**', но в результате команды

〉〉 **simbol = 'z'**

переменная «**Ans**» сохранит свое предыдущее значение

– **NaN** – указание на нечисловой характер данных.

Переменные – объекты, имеющие имена, способные хранить разные по значению и типу данные.

Типы переменных заранее не задаются, а определяются выражением, значение которого присваивается переменной. Знак присваивания «**=**».

Идентификатор переменной может состоять из любого количества символов, но идентифицируется только 31 начальный символ. Идентификатор должен начинаться с буквы, может содержать буквы, цифры и символ «_».

Примеры:

>> **x = 7.7**

>> **x1 = 'w'**

>> **x_input = 33.3**

Система Matlab предполагает, что каждая переменная есть *массив*, в частном случае состоящая из одного элемента.

Операторы – специальное обозначение для определенной операции над данными – *операндами*. В системе Matlab определены следующие виды операторов:

– арифметические операторы;

– операторы отношения;

– логические операторы;

– поразрядные операторы.

Операторы отношения, логические и поразрядные операторы будут рассмотрены в следующих лекциях.

Рассмотрим некоторые арифметические операции и соответствующие операторы системы Matlab.

Сложение: **M1 + M2**, например:

>> **x1+7.7;**

>> **x2+x1;**

Вычитание: **M1 – M2**, например:

>> **x1-7.7;**

>> **x2-x1;**

Умножение **M1 * M2**, например:

>> **x1*7.7;**

>> **x2*x1;**

Возведение в степень **M1 ^ x**, например:

>> **x1^3;**

>> **x2^x1;**

Деление **M1 / M2**, например:

>> **x1/7.7;**

>> **x2/x1;**

Среди рассмотренных арифметических операций наивысшим приоритетом обладает операция возведения в степень, т. е., если выражение состоит из нескольких операций, то в первую очередь будет выполнено возведение в степень. Далее умножение и деление. Наименьший приоритет у операций сложение и вычитание.

Функции – объекты, имеющие уникальные имена, выполняющие определенные преобразования над своими аргументами и возвращающие результат этих преобразований.

Функции бывают:

– встроенные, хранящиеся в откомпилированном ядре системы Matlab, поэтому они выполняются быстро;

– внешние, хранящиеся в файлах с расширением 'm'. Пользователи системы Matlab могут также создавать **M**-файлы, в которых могут быть запрограммированы функции (function) или сценарии (script).

Функции могут иметь один или несколько параметров (аргументов). При вызове функции список параметров необходимо заключать в круглые скобки, разделителем в списке параметров функции служит запятая. Параметром (аргументом) функции может быть константа или переменная, значение которой к моменту вызова функции определено. При этом константа или значение переменной должно быть допустимого для данного случая типа, например у функции, вычисляющей логарифм, аргумент может быть только численного типа. Рассмотрим для примера функцию, предназначенную для построения графиков, она может иметь от одного до нескольких параметров. Если параметр только один, то вызов функции выглядит следующим образом:

plot(x)

При количестве параметров равном двум вызов функции:

plot(x,y)

Если число параметров три, при этом третий параметр, определяющий формат линии графика, задан в идее символьной константы, то вызов функции

plot(x,y,’ -.or’)

Пусть теперь третий параметр заранее задан, как значение переменной:

s = ’ -.or’

Теперь можно записать

plot(x,y,s)

В данной работе мы рассмотрим только несколько примеров встроенных арифметических функций системы Matlab.

Функция **log(Z)** вычисляет натуральный логарифм значения аргумента, например:

Z = 12

V = log(Z)

Результат: **V = 2.4849**.

Функция **log10(Z)** вычисляет десятичный логарифм значения аргумента:

V = log10(Z)

Результат: **V = 1.0792**.

Функция **log2(Z)** вычисляет двоичный логарифм значения аргумента, например:

V = log2(Z)

Результат: **V = 3.5850**.

Функция **exp(Z)** вычисляет экспоненты значений аргумента, например:

V = exp(Z)

Результат: **V = 162754.79.**

Функция **abs(Z)** вычисляет абсолютную величину значения аргумента, например:

V = abs(-Z)

Результат: **V = 12.**

Функция **sqrt(Z)** вычисляет квадратный корень аргумента, например:

V = sqrt(Z)

Результат: **V = 3.46**.

Рассмотрим тригонометрические функции. Параметрами функций **sin**, **cos**, **tan**, **cot** и других тригонометрических функций системы Matlab являются углы, измеряемые в радианах. Допускаются комплексные значения аргументов данных функций.

Функция **sin(Z)** вычисляет синус значения аргумента, например:

V = sin (Z)

Результат: **V = -0.54**.

Функция **cos (Z)** вычисляет косинус значения аргумента, например:

V = cos (Z)

Результат: **V = 0.84**.

Функция **tan (Z)** вычисляет тангенс значения аргумента, например:

V = tan (Z)

Результат: **V = -0.64**.

Функция **cot (Z)** вычисляет котангенс значения аргумента, например:

V = cot (Z)

Результат: **V = -1.57**.

Обратные тригонометрические функции системы Matlab **asin**, **acos**, **atan**, **acot** и другие возвращают углы, измеряемые в радианах.

Функция **asin(Z)** вычисляет арксинус значения аргумента, например:

V = asin (Z)

Результат: **V = 1.57**.

Функция **acos (Z)** вычисляет арккосинус значения аргумента, например:

V = acos (Z)

Результат: **V = 0**.

Функция **atan (Z)** вычисляет арктангенс значения аргумента, например:

V = atan (Z)

Результат: **V = 1.49**.

Функция **acot (Z)** вычисляет арккотангенс значения аргумента, например:

V = acot (Z)

Результат: **V = 0.08**.

Рассмотрим функции, предназначенные для округления чисел до целых значений.

Функция **round (X)** округляет аргумент **X** до ближайшего целого числа, например:

y=round(2.2)

Результат: **y=2**.

z=round(2.8)

Результат: **z=3**.

Для комплексного **X** мнимые и вещественные части округляются независимо друг от друга, например:

y=round(2.2+2.8i)

Результат: **y=2+3i**.

z=round(2.8+2.2i)

Результат: **z=3+2i.**

Функция **ceil (X)** округляет аргумент **X** до ближайшего целого числа, большего или равного **X**, например:

y=ceil(2.2)

Результат: **y=3.**

Для комплексного **X** мнимые и вещественные части округляются независимо друг от друга.

Функция **floor (X)** округляет аргумент **X** до ближайшего целого числа, меньшего или равного **X**, например:

y=floor(2.8)

Результат: **y=2.**

Для комплексного **X** мнимые и вещественные части округляются независимо друг от друга.

Функция **fix (X)** округляет аргумент **X** до ближайшего целого числа, меньшего или равного **X**, если **X>0**, например:

y=fix(2.8)

Результат: **y=2.**

Функция **fix (X)** округляет аргумент **X** до ближайшего целого числа, большего или равного **X**, если **X<0**, например:

y=fix(-2.8)

Результат: **y=-2.**

Для комплексного **X** мнимые и вещественные части округляются независимо друг от друга.

1.3. Возможности пакета Matlab

С помощью языка Matlab можно:

– производить вычисления;

– создавать программы;

– моделировать процессы;

– анализировать, исследовать и визуализировать данные;

– разрабатывать приложения, включающие графический интерфейс.

Для того чтобы начать работу в среде Matlab, необходимо запустить приложение Matlab.*exe* нажатием клавиши «Enter» или двойным щелчком левой кнопкой мыши по пиктограмме, изображенной на рисунке 1.1, а, и расположенной на рабочем столе, или с помощью меню «Пуск» («Start»), как показано на рисунке 1.1, б.

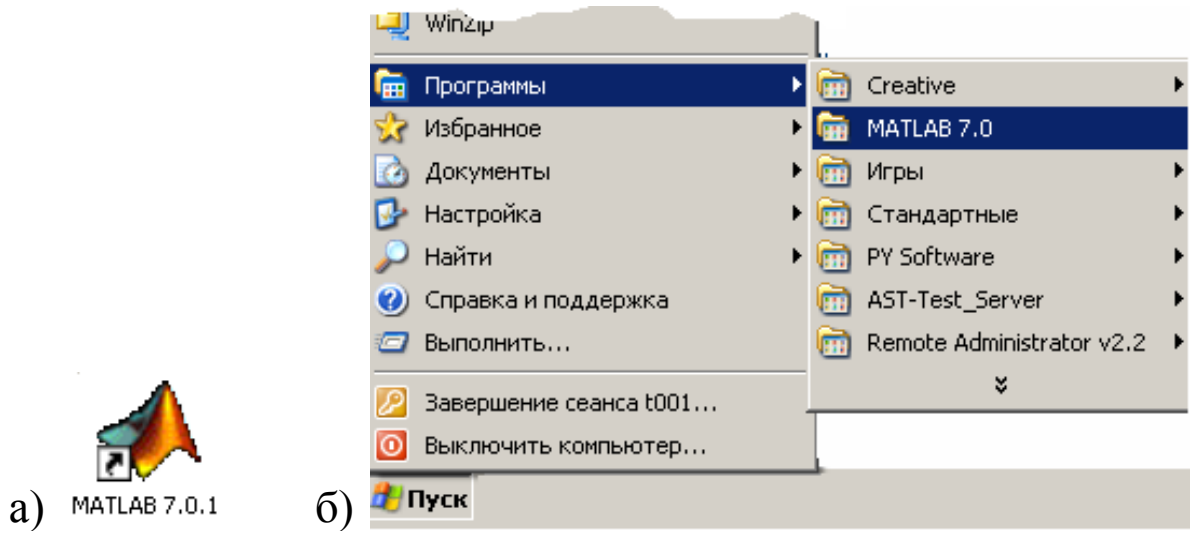


Рисунок 1.1 – Запуск приложения Matlab

1.4. Работа в командном окне

Сеанс работы в системе Matlab начинается с работы в командном окне. На рисунке 1.2 показан вид командного окна Matlab версии 7 при первой загрузке системы, когда пользователь еще не внес изменения в конфигурацию окон.

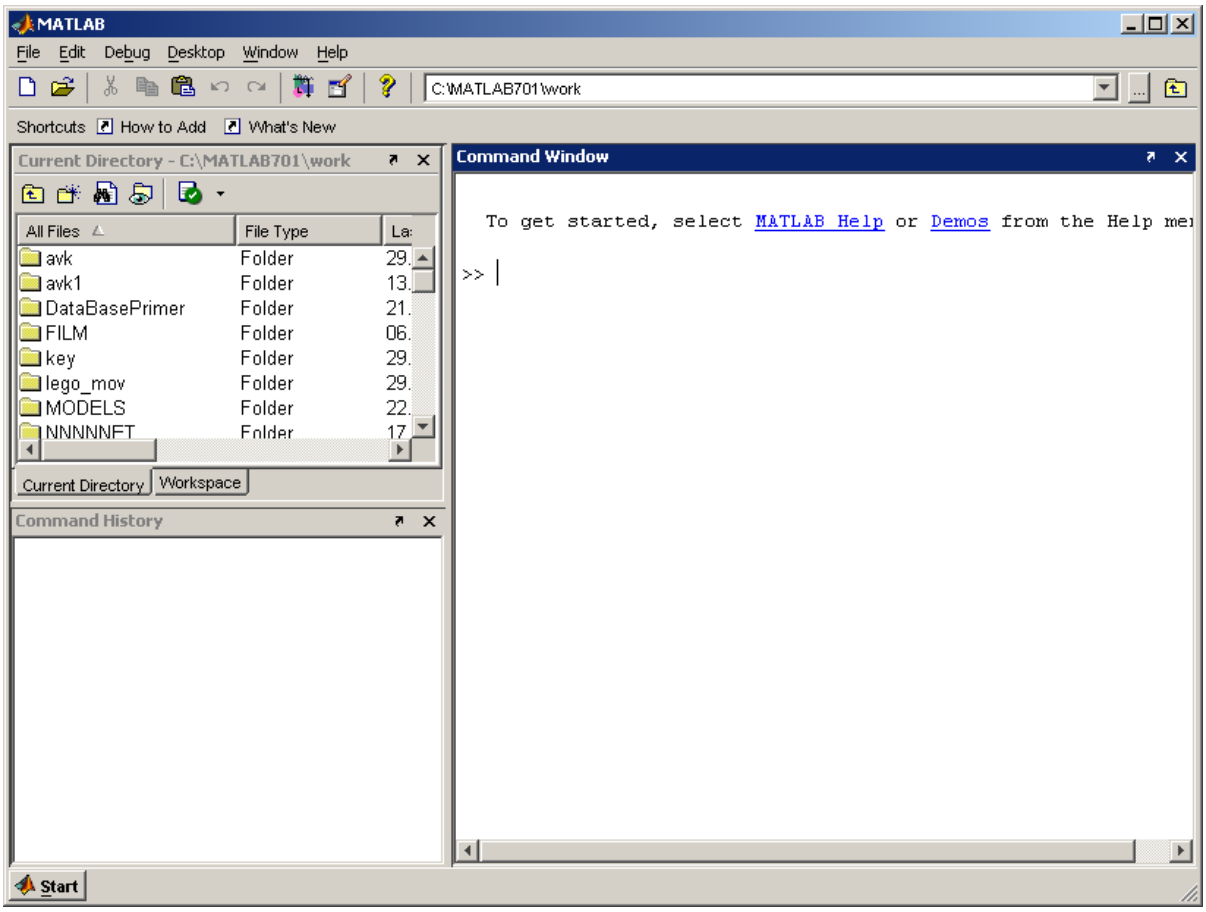


Рисунок 1.2 – Вид окон Matlab при первой загрузке системы

Одновременно с «Command Window» загружаются окна «Workspace», «Command History» и «Current Directory», их функции будут рассмотрены в данной работе и в следующих лабораторных работах. Для начала эти окна лучше закрыть щелчком левой кнопки мыши по крестику в верхнем правом углу каждого из перечисленных окон, тогда останется открытым только командное окно «Command Window».

В командном окне на позиции ввода находится мигающий курсор. Символ «⟩⟩» – приглашение для ввода команд, которые образуют строку ввода или командную строку. В строку ввода можно записать различные команды Matlab. Например, введем значение переменной *x* и вычислим следующую функцию от переменной **x**:

$$Q = \frac{0.2x^4 \cos|x|}{x - 77}.$$

На рисунке 1.3, а, показано, как можно задать формат вывода результатов (**format bank**), затем последовательно применить функции **cos** и **abs**, оператор «*****» для вычисления необходимой функции. На рисунке 1.3, б, видно, что данные вычисления можно задать в одной строке. На рисунке 1.3, в, применен специальный символ «**;**» для подавления вывода результата в командном окне.

Примечание. В данном примере командное окно системы Matlab использовано в качестве калькулятора. **x** и **y** являются идентификаторами переменных. Правила образования идентификаторов переменных, некоторые функции, операторы и специальные символы Matlab рассмотрены в пункте 1.2.

Возможен перенос строки ввода на следующую строку с помощью многоточия «…», системой Matlab одинаково интерпретируются команды:

⟩⟩ **x = [1 2 3;4 1 6;...**
4 5 1];

и

⟩⟩ **x = [1 2 3;4 1 6;4 5 1];**

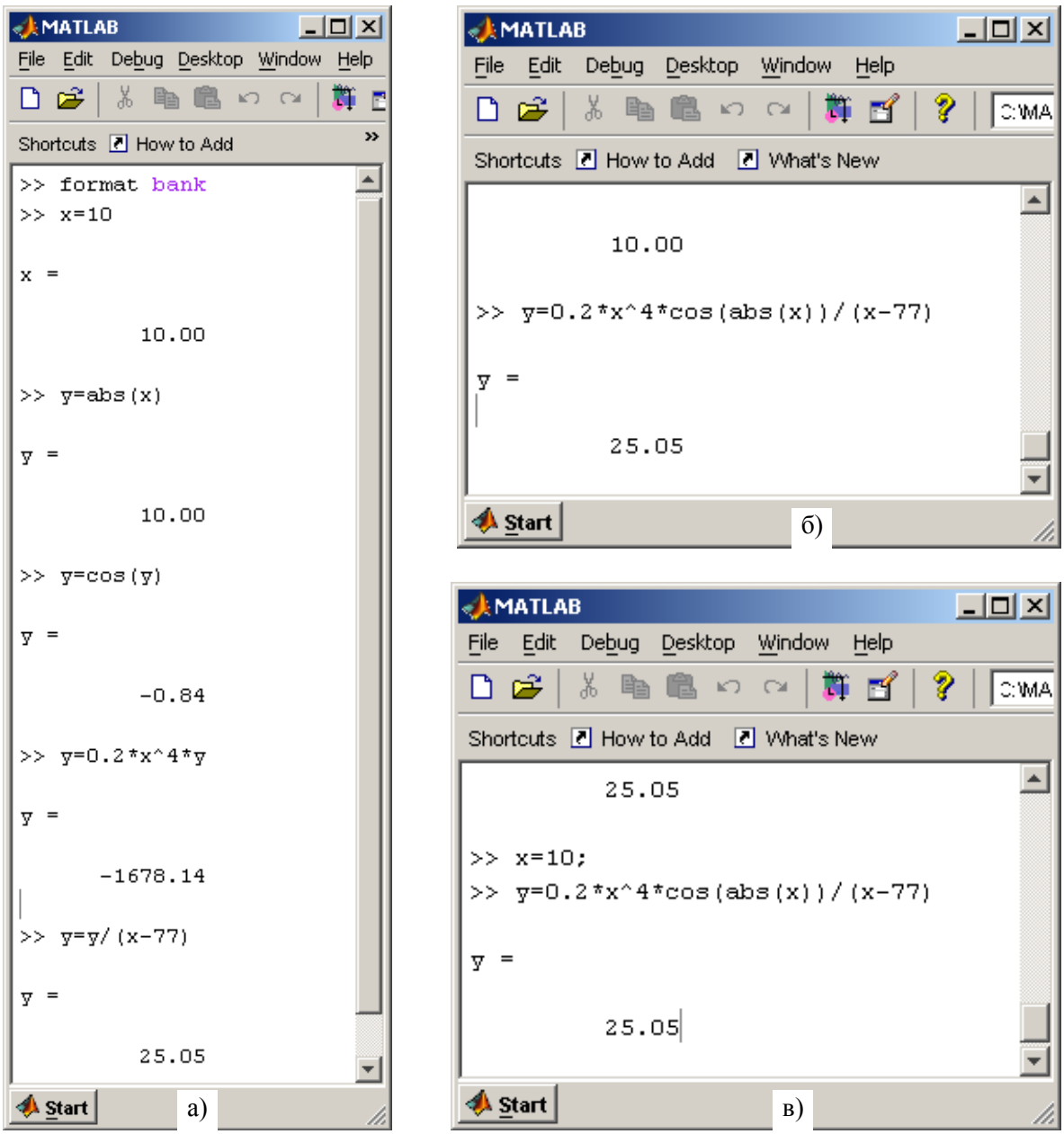


Рисунок 1.3 – форматы вывода результатов

Благодаря механизму переноса командной строки с помощью многоточия «…», одна командная строка может состоять из любого количества физических строк, но *зоной редактирования* является только последняя физическая строка командного окна Matlab. Таким образом, командное окно делится на зону редактирования и зону просмотра. Все действия, связанные с редактированием (удаление символов, ввод символов с клавиатуры, вставка содержимого буфера обмена и т. д.), возможны только в зоне редактирования.

Команды меню в пункте «Edit» главного меню окна Command Window:

– **Unddo** – отменить последнее действие;

– **Redo** – вернуть отмененное действие;

– **Gut** – вырезать и запомнить в буфере обмена выделенные символы (в зоне просмотра символы не удаляются, а только помещаются в буфер обмена);

– **Copy** – копировать в буфер обмена выделенные символы;

– **Paste** – вставить из буфера обмена символы;

– **Paste Special** – открытие окна выбора специальных вставок;

– **Select All** – выделить текст всего окна Command Window;

– **Delete** – удалить выделенные символы или символ после курсора;

– **Find** – поиск строки в тексте Command Window (откроется окно, в котором можно ввести необходимую для поиска строку);

– **Find Files** – поиск файла (открытие окна поиска файлов);

– **Clear Command Window** – удаление всего текста в Command Window, позиция ввода перемещается в верхний левый угол Command Window;

– **Clear Command History** – удаление всех строк в окне Command History;

– **Clear Workspace** – удаление всех данных из рабочей области Matlab и очистка окна просмотра переменных.

Кнопки панели инструментов, предназначенные для редактирования строки ввода:

–  – вырезать и запомнить в буфере обмена выделенные символы (данное действие можно производить не только в строке редактирования, но и в зоне просмотра, но символы в зоне просмотра удалены не будут);

–  – копировать в буфер обмена выделенные символы (данное действие можно производить не только в строке редактирования, но и в зоне просмотра);

–  – вставить символы из буфера обмена символы;

–  – отменить последнее действие;

–  – вернуть отмененное действие.

В таблице 1.1 перечислены специальные клавиши и комбинации клавиш, предназначенные для редактирования текста командной строки.

Таблица 1.1 – Комбинации клавиш для редактирования текста командной строки

Команда редактирования строки ввода	Комбинация клавиш	Назначение	Примечание
Режим вставки символа	Ins	Включение / выключение режима вставки символа	
Перемещение курсора	→ Ctrl+b	Вправо на один символ	
	← Ctrl+f	Влево на один символ	
	Ctrl+→ Ctrl+r	Вправо на одно слово	
	Ctrl+← Ctrl+l	Влево на одно слово	
	Home Ctrl+a	На начало строки	
	End Ctrl+e	В конец строки	
Выделение символов	Shift+→	Слева направо	Данные действия можно производить и в зоне просмотра, в случае действия «Ctrl+x» символы в зоне просмотра удалены не будут
	Shift+←	Справа налево	
Действия с буфером обмена	Ctrl+x	Вырезать и запомнить в буфере обмена выделенные символы	
	Ctrl+c	Копировать в буфер обмена выделенные символы	
	Ctrl+v	Вставить из буфера обмена символы	
	Delete	Удалить выделенные символы	
	Ctrl+z	Отменить последнее действие	
Подстановка команд	↑ Ctrl+p	Предыдущих	
	↓ Ctrl+n	Последующих	
Стирание символа	Delete Ctrl+d	Справа от курсора	
	Ctrl+h	Слева от курсора	
Очистка строки ввода	Ctrl+k	От текущей позиции до конца строки	
	Esc	всей строки	

1.5. Создание сценария

Для записи и последующего сохранения сценария команд («Script») необходимо создать **M**-файл одним из следующих способов:

– воспользоваться пунктом главного меню «File/New/M-File»;

– дважды щелкнуть левой кнопкой мыши по пиктограмме  с всплывающей подсказкой «New MFile»;

– в строке ввода команд ввести **edit.**

Любое из перечисленных действий приведет к открытию редактора **M**-файлов. Введем в поле редактора команды, которые ранее работали в режиме калькулятора, затем сохраним файл под именем «Prim1». После чего окно редактора **M**-файлов будет выглядеть, как показано на рисунке 1.4.

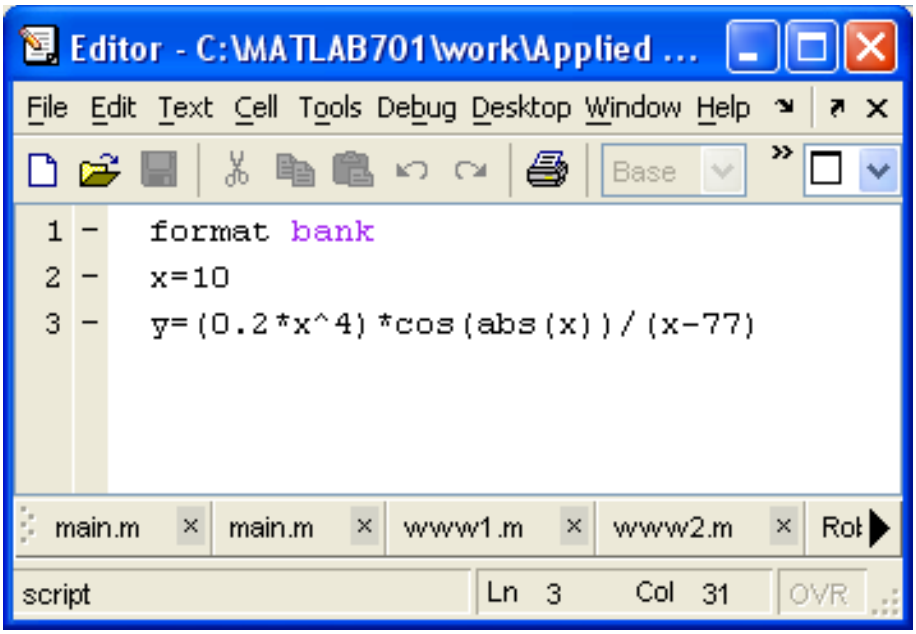


Рисунок 1.4 – Вид редактора M-файлов с введенными командами

Для выполнения команд, записанных в сценарии, можно воспользоваться одним из способов:

– активизировать пункт главного меню редактора «Debug/Run»;

– нажать клавишу **F5**;

– дважды щелкнуть левой кнопкой мыши по пиктограмме  с всплывающей подсказкой **Run.**

Результат выполнения сценария выводится в командном окне.

Примечание. Правила формирования имени **M**-файла аналогичны правилам формирования идентификаторов переменных.

1.6. Создание файла Simulink-модели

Файл **Simulink**-модели может быть создан одним из следующих способов:

– воспользоваться пунктом главного меню «File/New/Model», что приведет к открытию редактора **Simulink**-модели, после чего необходимо открыть также библиотеку **Simulink** с помощью пиктограммы с всплывающей подсказкой «Library Browser» или пункта главного меню «View/Library Browser»;

– из командного окна открыть библиотеку **Simulink** с помощью пиктограммы с всплывающей подсказкой «Simulink». Затем в открывшемся окне «Simulink Library Browser» активизировать пункт меню «File/New/Model» или воспользоваться сочетанием клавиш **Ctrl+n** или дважды щелкнуть левой кнопкой мыши по пиктограмме с всплывающей подсказкой «Create a new Model», что также приведет к открытию редактора **Simulink**-модели.

Блоки, расположенные в правом окне библиотеки **Simulink,** переносятся в окно редактора **Simulink**-модели с помощью стандартной технологии **Windows** «drag and drop» («перетащить и опустить»). На рисунке 1.5 показан пример переноса блока «Constant» группы блоков «**Sources**» в окно модели с помощью левой кнопки мыши. Для установки необходимого значения константы необходимо дважды щелкнуть левой кнопкой мыши по пиктограмме «Constant», расположенной в окне модели, затем в раскрывшемся окне с заголовком «Sources Bloc Parameters: Constant» установить требуемое значение в строке редактирования «Constant Value» (рис. 1.6).

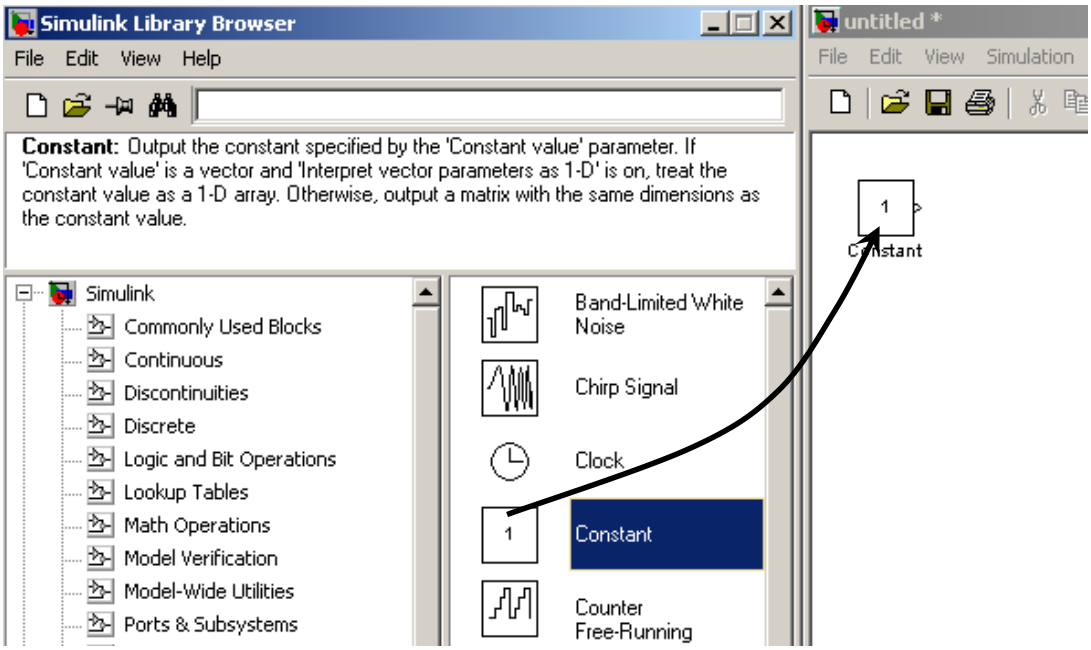


Рисунок 1.5 – Перенос блока в окно модели

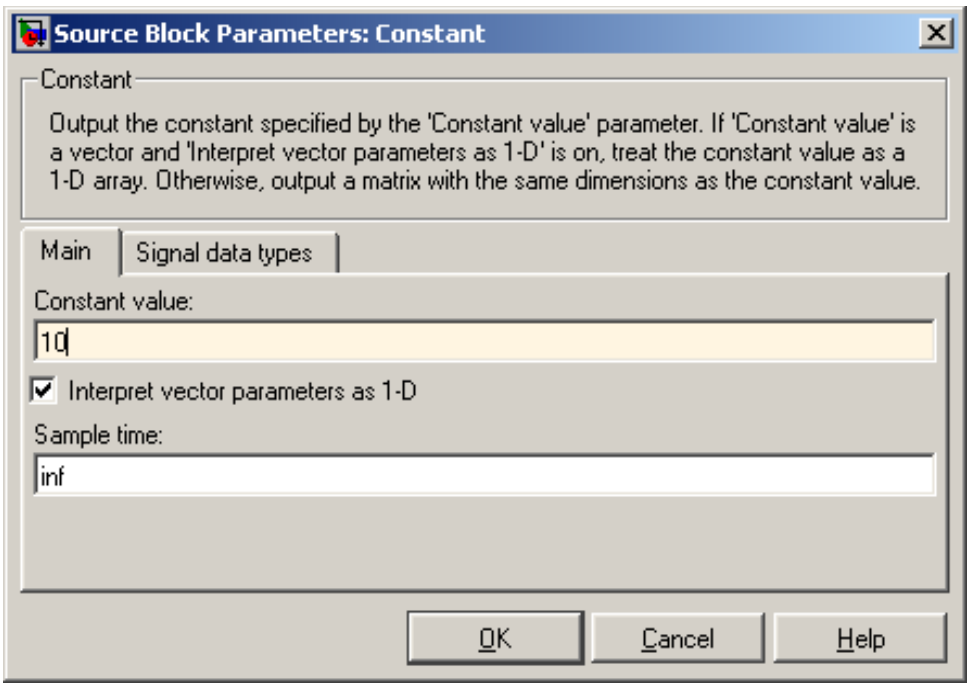


Рисунок 1.6 – Установка значения в строку редактирования

Пусть требуется составить модель для вычисления функции от переменной $\mathbf{x}\,\mathrm{Q} = \dfrac{0.2x^4 \cos|x|}{x-77}$. Ранее эта функция была вычислена с помощью сценария при **x** =10. Зададим значение константы, равное 10. **Simulink**-модель для вычисления функции, сохраненная в файле с именем «SimPrim1.mdl», приведена на рисунке 1.7.

Соединение блоков в направлении передачи сигнала осуществляется с помощью левой кнопки мыши. Для разветвления сигнала необходимо начать прорисовывать линию от места разветвления, нажав правую кнопку мыши.

Примечание. Правила формирования имени файла **Simulink**-модели аналогичны правилам формирования имен **M**-файлов. Не рекомендуется присваивать одинаковые имена **M**-файлу и файлу **Simulink**-модели.

Рассмотрим порядок построение Simulink-модели. Сначала возведем число 10 в степень 4. Для этого используем блок «Math Function» раздела «Math Operation» (рис. 1.8). После переноса данного блока в окно **Simulink**-модели двойным щелчком по пиктограмме блока откроем окно «Function Bloc Parameters: Math Operation», с помощью выпадающего меню «Function» выберем функцию **pow**, а степень 4 установим, присоединив ко второму входу блока «Math

Function» сигнал от константы, имеющей значение, равное 4. Метод моделирования константы рассмотрен выше.

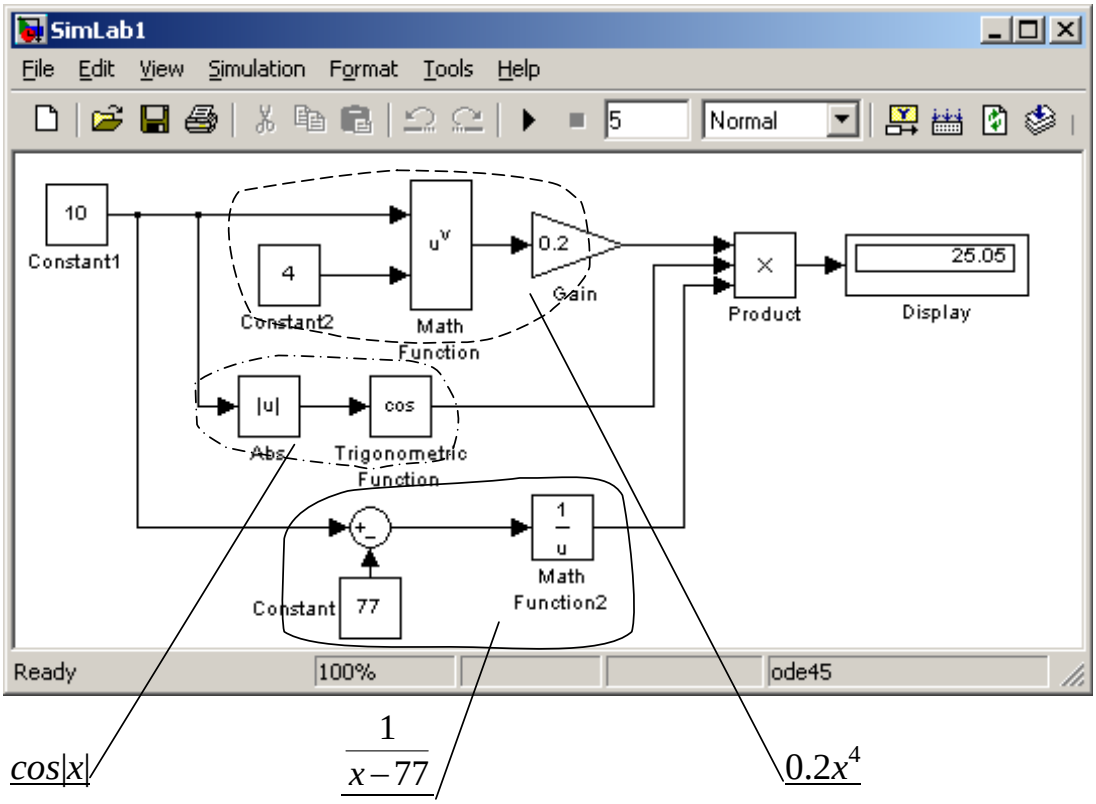


Рисунок 1.7 – Вычисление функции в файле .mdl

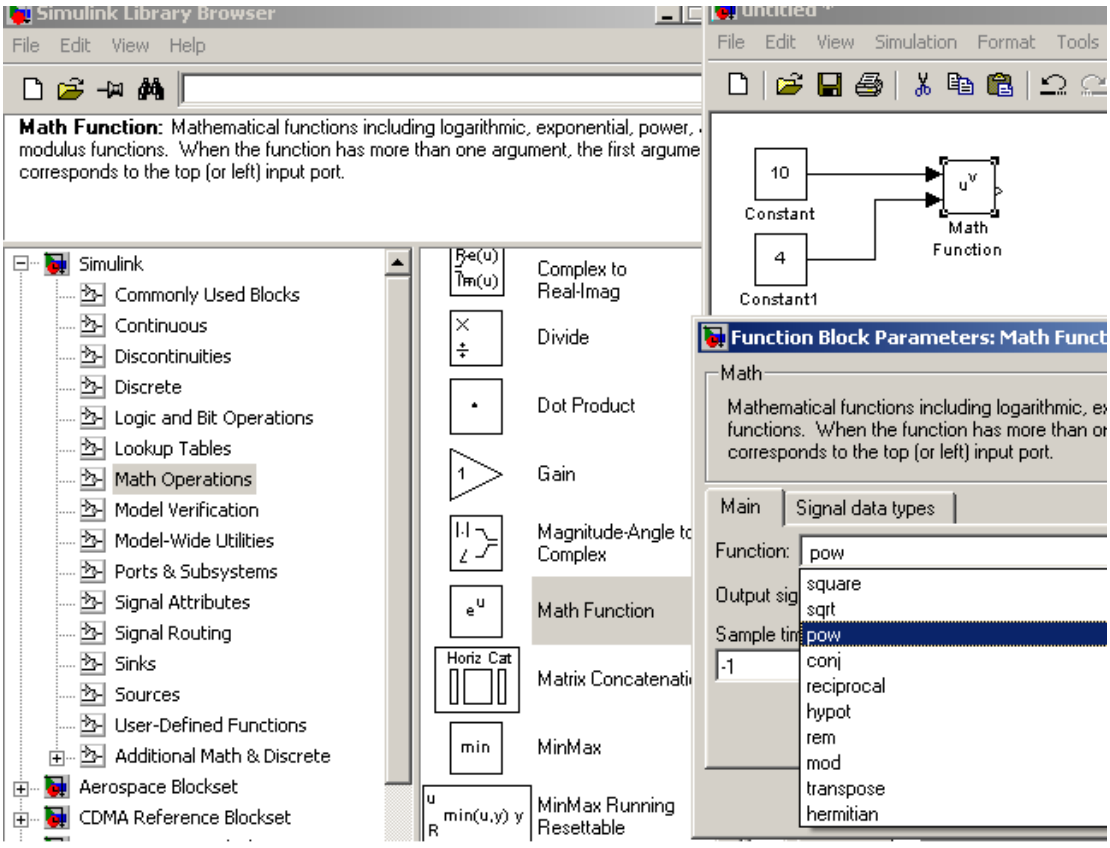


Рисунок 1.8 – Использование блока «Math Function»

С помощью блока «Math Function» можно моделировать математические функции, приведенные в таблице 1.2.

Таблица 1.2 – Функции, доступные для вычисления с помощью блока «Math Function»

Обозначение в меню «Function» блока «Math Function»	Функция
exp	e^x (экспонента)
10^u	Степенная функция
log	Натуральный логарифм
log10	Десятичный логарифм
magnitude^2	Модуль комплексного числа в квадрате
square	Квадрат
sqrt	Квадратный корень
pow	Возведение в степень
conj	Комплексно сопряженное число
reciprocal	Обратная величина
transpose	Транспонирование

Результат действия 10^4 необходимо умножить на 0.2. Используем блок «Gain» из раздела «Math Operation». Для того чтобы установить необходимый множитель (0.2), откроем окно настройки блока двойным щелчком мыши и изменим число 1 в редакторе «Gain» на число 0.2 (рис. 1.9).

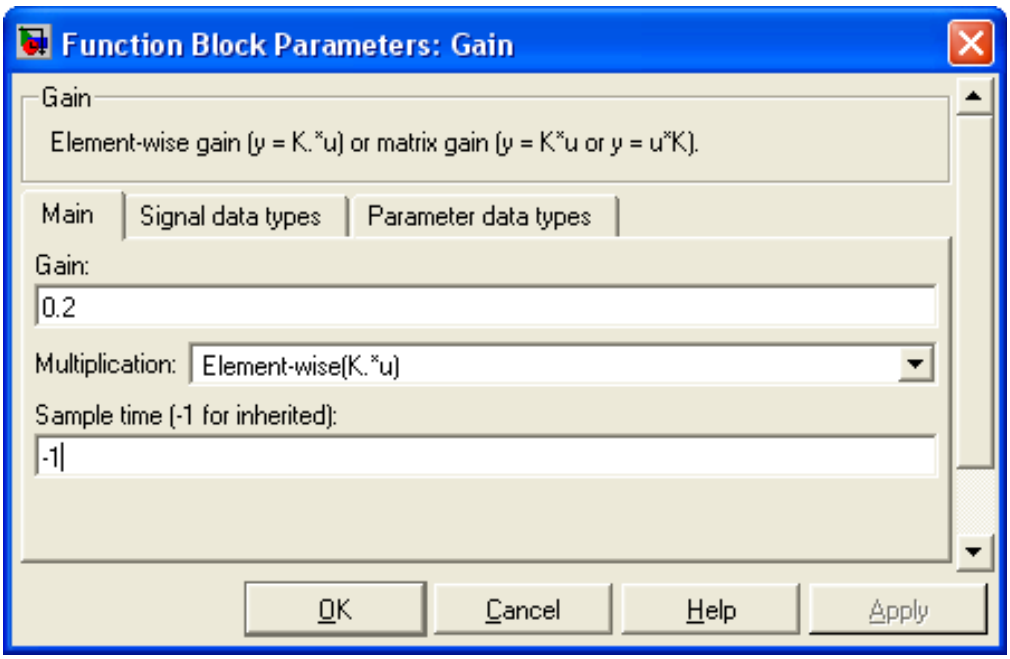


Рисунок 1.9 – Установка множителя в блоке «Gain»

Таким образом, выполнены действия $0.2x^4$ (x=10). На рисунке 1.7 соответствующий участок обведен пунктирной линией.

Для расчета ***cos|x|*** применены блоки «Abs» и «Trigonometric Function» из раздела «Math Operation», при этом функция **cos** установлена с помощью выпадающего меню в окне настройки блока «Trigonometric Function». На рисунке 1.7 соответствующий участок обведен штрихпунктирной линией.

На рисунке 1.7 сплошной линией обведен участок для вычисления выражения $\frac{1}{x-77}$. Использованы блоки «Constant» из раздела «Sources» и «Sum» из раздела «Math Operation». Настройка блока «Sum» производится с помощью окна настройки, которое можно открыть двойным щелчком левой кнопки мыши по пиктограмме блока (в редакторе «List of signs» вместо строки «|++» ввести строку «|+-», как на рисунке 1.10). Затем блок «Math Function» с функцией **reciprocal** (рис. 1.11) превращает ***x*77** в $\frac{1}{x-77}$.

Наконец перемножаем $\mathbf{0.2x^4}$**,** ***cos|x|*** и $\frac{1}{x-77}$. Блок «Product» находится в разделе «Math Operation», количество входных сигналов можно установить в окне настройки (рис. 1.12).

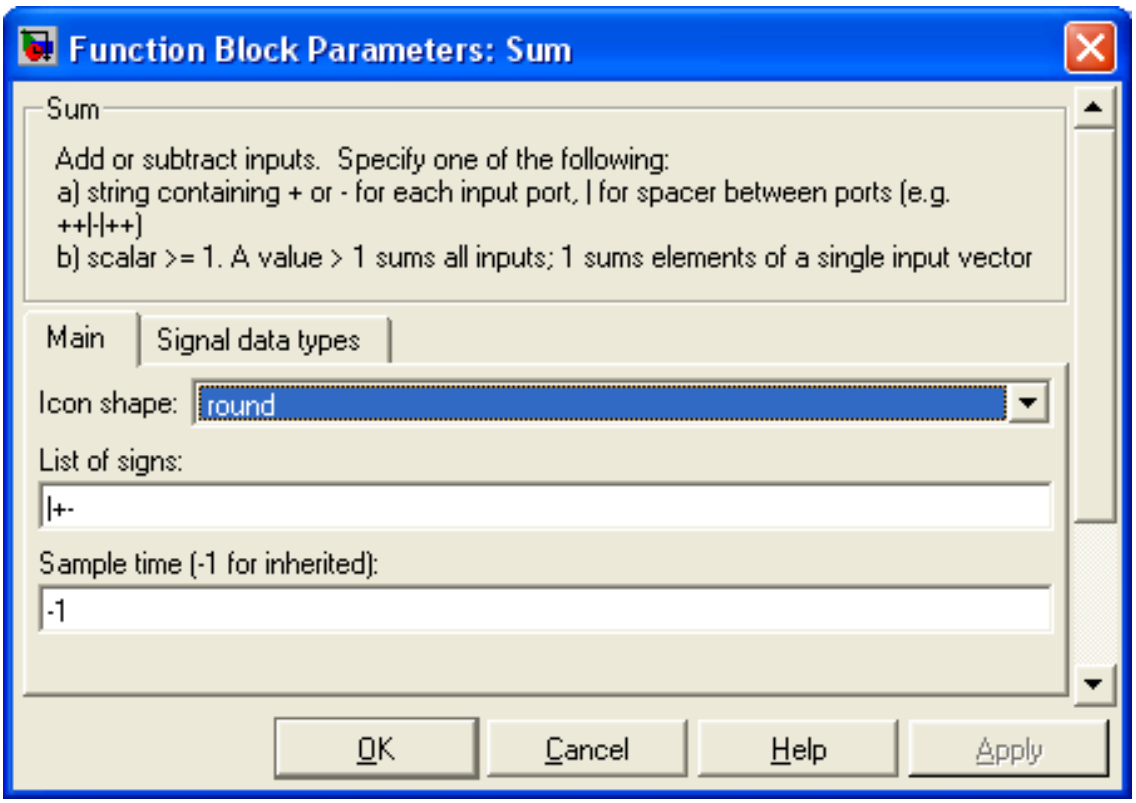


Рисунок 1.10 – Настройка блока «Sum»

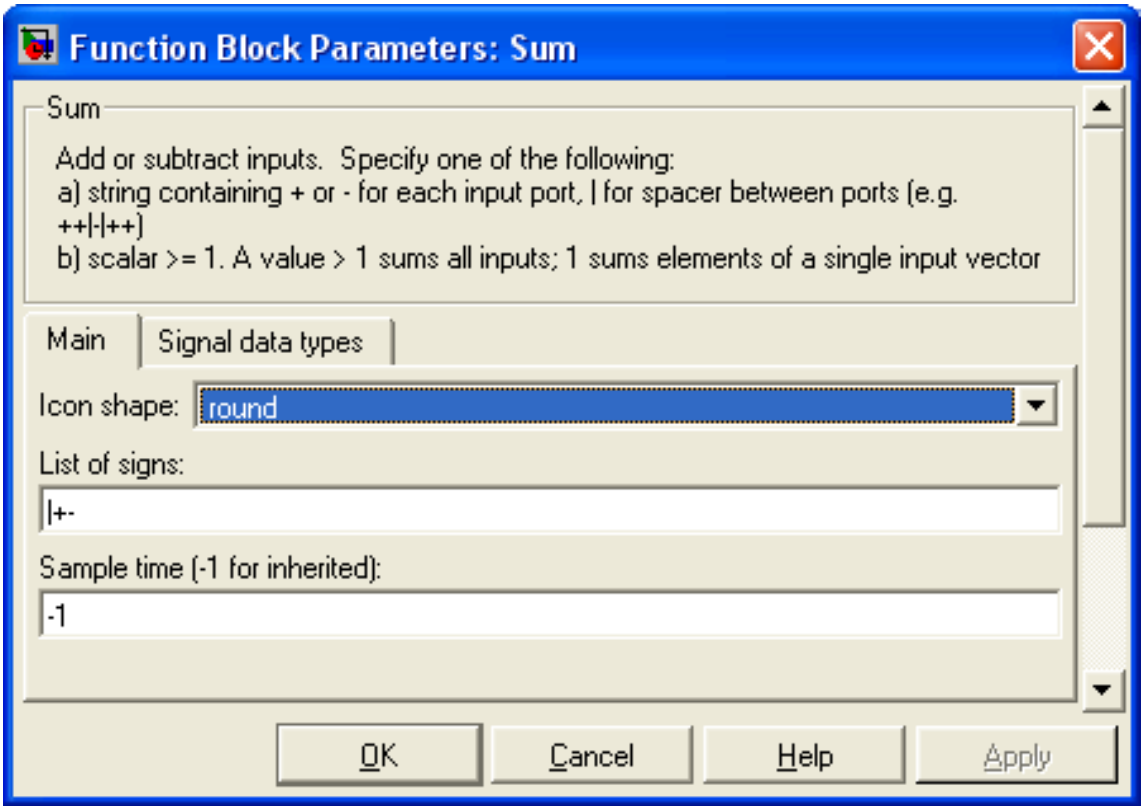


Рисунок 1.11 – Функция reciprocal

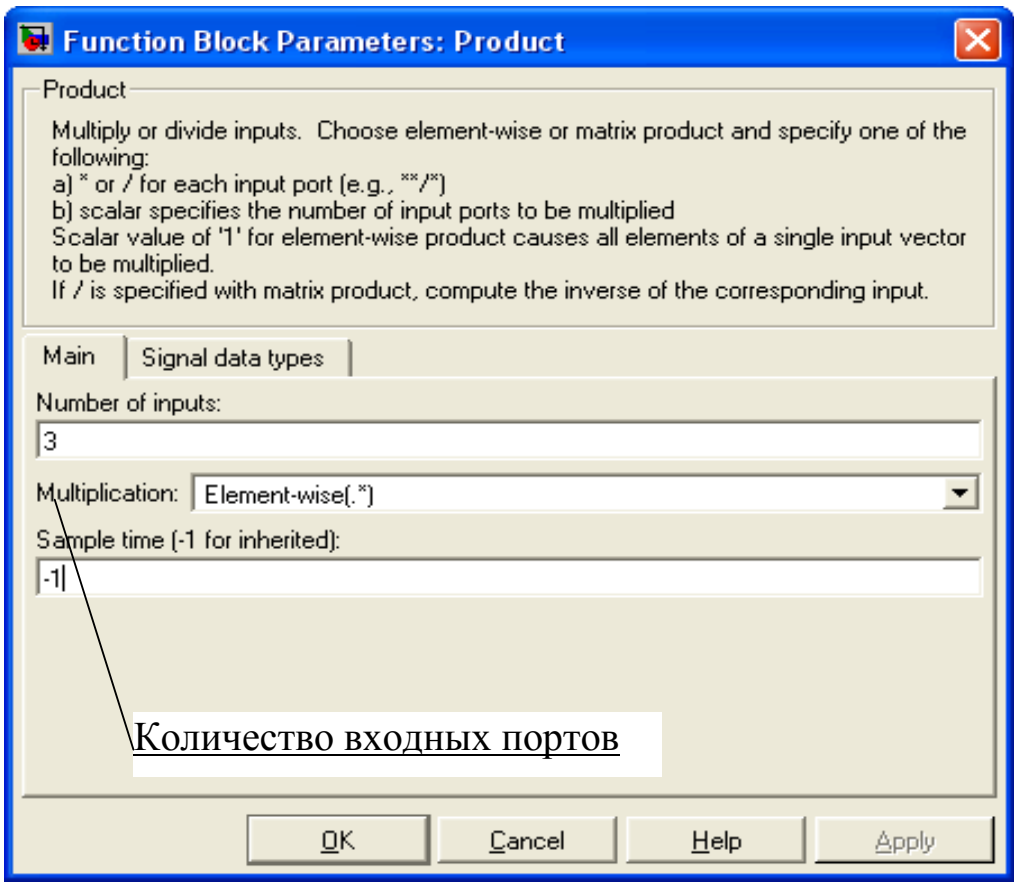


Рисунок 1.12 – Установка количества входных сигналов

Результат расчета модели будет выведен в блоке «Display», находящемся в разделе «Sinks».

Запуск модели производится щелчком левой кнопкой мыши по пиктограмме ▶ с всплывающей подсказкой «Start simulation». После запуска модели в блоке «Display» отразится результат 26.05.

1.7. Задание для лабораторной работы

Для выполнения работы потребуется персональный компьютер с требованиями: любой процессор Intel или AMD (x86-64); 2 Гб свободного места на диске (только Matlab), 4–6 Гб для стандартной установки; 2 Гб оперативной памяти (с Simulink рекомендуется 4 Гб, с Polyspace рекомендуется 4 Гб на каждое ядро); графический ускоритель с поддержкой OpenGL 3.3 и 1 Гб графической памяти.

Ход работы:

1. В командном окне присвоить переменной любое допустимое значение – действительное число. Недопустимым может быть значение переменной, приводящее к делению на нуль, вычислению логарифма нуля и т. д. Можно выбрать значение из интервала, указанного в таблице 1.3.
2. Запрограммировать функцию согласно заданному варианту.

3. Открыть редактор **M**-файлов. В сценарии повторить действия пунктов 1 и 2.

4. Сохранить сценарий как **M**-файл.

5. Выполнить **M**-файл.

6. Сравнить результат расчета в режиме калькулятора и сценарии.

7. Создать **Simulink**-модель.

8. В **Simulink**-модели запрограммировать ту же функцию.

9. Сравнить результат расчета в режиме калькулятора, сценарии и **Simulink**-модели.

Таблица 1.3 – Варианты заданий

Номер варианта	Задание	Интервал значений x
1	$Q=2^x-e^x$	[–3; 0]
2	$Q=x^4-14x^3+60x^2-70x$	[–2; 1]
3	$Q=\frac{x^2-30}{x^4+7.7x^2+3}$	[7; 9]
4	$Q=2x^2+3e^{-x}$	[0.1; 1]
5	$Q=2+x-x^2$	[0; 1]
6	$Q=(1-x)^4$	[0; 2]
7	$Q=\cos(x)+x$	[1; 1,6]
8	$Q=x^{1/2}+(1-x)^{2/3}$	[8; 10]
9	$Q=x^3-6x^2+9x+4$	[2; 4]
10	$Q=2x^2-x^4$	[0.5; 1.5]
11	$Q=x+1/\lvert x+1\rvert$	[–0.9; 1]
12	$Q=x^3\sqrt{x-1}$	[0.9; 1.1]
13	$Q=x\times e^x$	[–2; 0]
14	$Q=\ln(x/(x-20)^{3/2})$	[0; 19]
15	$Q=\frac{x^2-3x+2}{x^2+2x-1}$	[1.3; 1.4]
16	$Q=\operatorname{arctg}(x)-1/2\times\ln(1+2x)$	[–0.5; 1]
17	$Q=\lvert x\rvert\times e^{-\lvert x-1\rvert}$	[–0.5; 0.5]
18	$Q=x^2+e^{1.1x+3}$	[–3; –1]
19	$Q=x^4-1.5\operatorname{arctg}(x)$	[0; 1]
20	$Q=-0.4x+e^{\lvert x-2\rvert}$	[1; 3]
21	$Q=\operatorname{tg}(x^2)+2x$	[–1; 1]
22	$Q=2*x+\sin(x^2)$	[1; 2]
23	$Q=-e^x\times\ln(x)$	[–3; –1]
24	$Q=\sqrt{x+5.5}+x^4$	[–1; 1]

Контрольные вопросы

1. Когда и кем разработана система Matlab?
2. Какие задачи можно решать с помощью языка Matlab?
3. Назвать основные объекты системы Matlab.
4. Что такое рабочая область?
5. Как просмотреть содержимое рабочей области?
6. Как очистить рабочую область?
7. Как редактировать содержимое рабочей области?
8. Численные константы, типы чисел в Matlab.
9. Какие форматы чисел используются в Matlab?
10. Символьные константы в Matlab.
11. Что такое переменные? Идентификаторы переменных.
12. Что такое операторы, операнды? Виды операторов.
13. Перечислить арифметические операторы. Привести примеры.
14. Встроенные функции в Matlab. Примеры встроенных арифметических функций.
15. Встроенные функции в Matlab. Примеры встроенных тригонометрических функций.
16. Встроенные функции в Matlab. Примеры встроенных функций, предназначенных для округления чисел до целых значений.
17. Как осуществить перенос строки ввода на следующую строку?
18. Перечислить команды меню в пункте «Edit» главного меню окна Command Window.
19. Перечислить кнопки панели инструментов, предназначенные для редактирования строки ввода.
20. Перечислить специальные клавиши и комбинации клавиш, предназначенные для редактирования текста командной строки.
21. Как можно создать сценарий?
22. Способы выполнения команд, записанных в сценарии.
23. Как можно создать Simulink-модели?
24. Технология переноса блоков в окно Simulink-модели.
25. Как установить значение константы?
26. Как настроить параметры блока «Math Function»?
27. Как установить множитель блока «Gain»?
28. Как настроить параметры блока «Abs»?
29. Как настроить параметры блока «Trigonometric Function»?
30. Как настроить параметры блока «Product»?
31. Запуска модели.
32. Как можно посмотреть результат вычисления модели?

Лабораторная работа № 2
РАБОТА С МАССИВАМИ В MATLAB

Цель работы: получение практических навыков, необходимых для работы с массивами данных с помощью Matlab.

2.1. Классификация массивов в Matlab

В лабораторной работе № 1 было отмечено, что Matlab (сокращение от MATrix LABoratory – матричная лаборатория) представляет собой интерактивную систему, предназначенную для выполнения инженерных и научных расчетов и ориентированную на работу с массивами данных. Система поддерживает выполнение операций с векторами, матрицами и массивами данных. Matlab обеспечивает выполнение операций с векторами и матрицами. Системой Matlab можно пользоваться как мощнейшим калькулятором, в котором наряду с обычными арифметическими и алгебраическими действиями могут использоваться такие сложные операции, как обращение матрицы, вычисление ее собственных значений и векторов, решение систем линейных алгебраических уравнений, и много других.

Система Matlab предполагает, что каждая переменная есть массив, в частном случае состоящий из одного элемента.

Matlab оперирует массивами следующих типов:

numeric – виртуальный тип, от которого образуются следующие типы массивов: **uint8**, **uint16**, **uint32**, **uint64**, **int8**, **int16**, **int32**, **double** (**int64**):

uint8 – массивы 8-разрядных чисел, значения элементов данных массивов могут лежать в интервале $[0; 2^8–1]$;

uint16 – массивы 8-разрядных чисел, значения элементов данных массивов могут лежать в интервале $[0; 2^{16}–1]$;

uint32 – массивы 8-разрядных чисел, значения элементов данных массивов могут лежать в интервале $[0; 2^{32}–1]$;

uint64 – массивы 8-разрядных чисел, значения элементов данных массивов могут лежать в интервале $[0; 2^{64}–1]$;

int8 – массивы 8-разрядных чисел, значения элементов данных массивов могут лежать в интервале $[–2^7; 2^7–1]$;

int16 – массивы 8-разрядных чисел, значения элементов данных массивов могут лежать в интервале $[–2^{15}; 2^{15}–1]$;

int32 – массивы 8-разрядных чисел, значения элементов данных массивов могут лежать в интервале $[-2^{31}; 2^{31}-1]$;

int64 – массивы 8-разрядных чисел, значения элементов данных массивов могут лежать в интервале $[-2^{63}; 2^{63}-1]$, данный тип также называется **double**, а также массивом чисел удвоенной точности;

sparse – разреженные матрицы чисел удвоенной точности;

char – вектор, элементы которого – символы;

cell – массивы ячеек, каждая ячейка содержит массив любого типа;

struct – массивы структур, где все элементы массива могут содержать массивы разных типов.

В данной лекции рассмотрим массивы типа **double**. При присваивании переменной какого-либо действительного значения под данную переменную автоматически выделяется 8 байт оперативной памяти компьютера, таким образом, по умолчанию инициализируется массив данных типа **double**, например, пусть произведены следующие действия:

⟩⟩ **x =** [1 2 3;4 1 6;4 5 1];

⟩⟩ **x1 =** 2.3

В рабочую область были записаны данные **x** и **x1.** При просмотре содержимого рабочей области любым из приведенных в лабораторной работе № 1 способов увидим, что переменная **x1** – массив размером **1×1**, занимает **8** байт памяти, имеет тип **double**, переменная **x** – массив размером **3×3**, занимает **72** байта памяти, имеет тип **double.**

Нумерация элементов массивов начинается с **1.**

2.2. Преобразование типов массивов

Часто необходимо преобразовать символьные данные в численные. Функция **str2num** позволяет преобразовать символьную строку, которая содержит только цифры, точки, пробелы, запятые и точки с запятой, в массив типа **double**. Пусть имеется строка символов:

MyString = ‘1.1, 1.2;12 13’

Преобразуем строку **MyString** в массив чисел:

Mass= str2num(MyString)

Полученный массив:

$$\mathbf{Mass}=\begin{bmatrix} 1.1 & 1.2 \\ 12.0 & 13.0 \end{bmatrix}$$

Массив символов, состоящий из двух строк, получится из массива типа **double** с помощью функции **num2str**:

NewString= num2str(Mass)

2.3. Организация массивов переменных

Подробно рассмотрим работу с наиболее интересующими нас одномерными и двумерными массивами.

Пусть массив состоит из одного элемента, организовать такой массив можно следующим образом:

db = 77;

или

db = [77];

или

db(1) = 77;

В рабочей области появится массив типа **double**, размером **1×1**, он будет занимать **8** байт оперативной памяти компьютера.

Если массив состоит более чем из одного элемента, то при присваивании значений элементам он ограничивается квадратными скобками. При присваивании значений элементам массива используются разделители:

– пробел или запятая – отделяет друг от друга элементы вектора;

– точка с запятой – переход к следующей строке матрицы.

Для того чтобы задать вектор (одномерный массив), необходимо применить одну из следующих операций присваивания:

пусть разделителем между элементами вектора является пробел, например:

vector = [1 2 3 1.1 2.2];

или (элементы вектора разделены запятой)

vector = [1, 2, 3, 1.1, 2.2];

или присвоим значение каждому элементу массива отдельно

vector(1) = 1;

vector(2) = 2;

vector(3) = 3;

vector(4) = 1.1;

vector(5) = 2.2;

или с использованием символа двоеточие «**:**» для объединения номеров элементов массива как на рисунке 2.1.

Рисунок 2.1 – Использование символа двоеточие для объединения номеров элементов массива

Результатом любой из приведенных операций явится появление в рабочей области массива типа **double**, размером **1×5**, он будет занимать **40** байт оперативной памяти компьютера.

Доступ к каждому элементу вектора возможен по его индексу. Например, для просмотра значения первого элемента вектора можно в командную строку ввести

>>**vector(1)**

Для того чтобы третий элемент вектора умножить на константу **5**, необходимо записать:

X = vector(3)*5;

В результате данной операции переменной **X** будет присвоено значение 15 (3×5 = 15).

С помощью двоеточия «**:**» можно объединить последовательно расположенные элементы вектора. Следующая запись позволяет просмотреть третий, четвертый и пятый элементы вектора:

>>**vector**(3:5)

Для того чтобы задать вектор, элементы которого члены ряда **Xi+1 = Xi+h**, где значение **h** постоянно, можно воспользоваться записью:

X = -2:0.5:1;

где **-2** – значение первого элемента вектора; **1** – значение последнего элемента вектора; **0.5** – шаг (разность между текущим и предыдущим элементами вектора). Таким образом, задан массив **X=**|-2, -1.5, -1, -0.5, 0, 0.5, 1|.

Если значение шага не указано, то по умолчанию устанавливается шаг, равный **1**, например, введем строку:

X = -2:1;

Тогда элементы вектора **X**: -2, -1, 0, 1.

Шаг может быть отрицательным, например:

X = 1:-0.5:-2

В этом случае элементы вектора **X**: 1, 0.5, 0, -0.5, -1, -1.5, -2.

При организации двумерного массива (матрицы) для разделения строк применяется точка с запятой «**;**», например, (разделитель между элементами вектора – пробел). Пример – на рисунке 2.2.

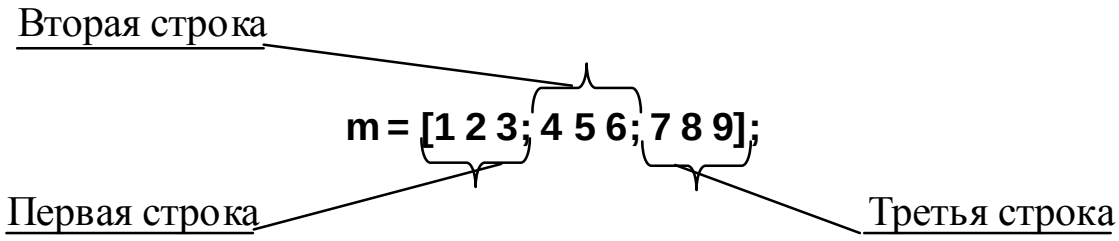


Рисунок 2.2 – Использование символа точка с запятой для разделения строк массива

Или то же самое, но элементы векторов разделены запятой:

m = [1, 2, 3; 4, 5, 6; 7, 8, 9];

или присвоим значение каждому элементу массива отдельно:

m(1,1) = 1;
m(2,1) = 4;
m(3,1) = 7;
m(1,2) = 2;
m(2,2) = 5;
m(3,2) = 8;
m(1,3) = 3;
m(2,3) = 6;
m(3,3) = 9

В любом из этих случаев получается матрица

$$\mathbf{m} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

Массив **m** имеет размерность **3×3**, тип **double**, занимает **72** байта оперативной памяти компьютера.

При обращении к элементу матрицы необходимо после идентификатора массива в круглых скобках указать сначала номер строки, в которой находится элемент, затем через запятую – номер столбца,

например, элемент **m(3,2)**, равный **8**, находится в третьей строке и втором столбце матрицы **m**.

Зададим первую строку матрицы **matrix** как члены возрастающего ряда, а третью – как члены убывающего ряда (рис. 2.3), или зададим сначала значения векторов, затем создадим из полученных векторов матрицу (рис. 2.4).

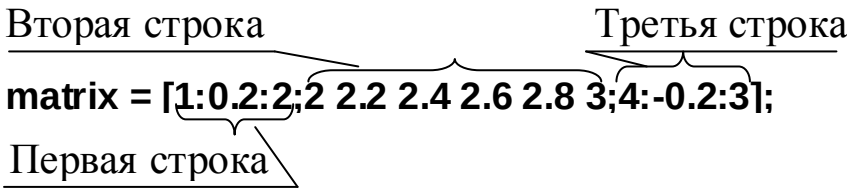


Рисунок 2.3 – Строки матрицы как члены возрастающего и убывающего ряда

```
vec = 1:0.2:2;
vector = [2 2.2 2.4 2.6 2.8 3];
v = 4:-0.2:3;
```

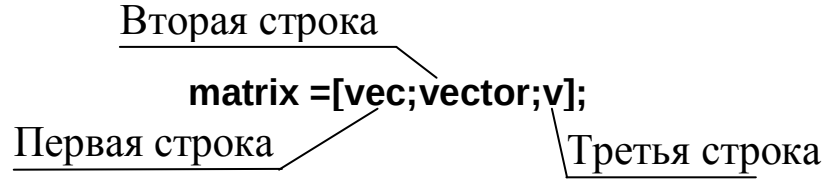


Рисунок 2.4 – Создание матрицы из векторов

Возможны и другие варианты задания матрицы.

Примечание: необходимо, чтобы все векторы состояли из одного и того же числа элементов.

Математическое изображение полученной матрицы:

$$\mathbf{matrix}=\begin{bmatrix} 1 & 1.2 & 1.4 & 1.6 & 1.8 & 2 \\ 2 & 2.2 & 2.4 & 2.6 & 2.8 & 3 \\ 4 & 3.8 & 3.6 & 3.4 & 3.2 & 3 \end{bmatrix}$$

При обращении к элементам матрицы также возможно объединение индексов с помощью двоеточия «**:**». Примеры:

matrix(:,2) – обращение ко второму столбцу и всем строкам матрицы, т. е. ко всему вектору **1.2**, **2.2** и **3.8**.

matrix(1,2:5) – обращение к первой строке и элементам со второго по пятый, т. е. **1.2**, **1.4**, **1.6** и **1.8**.

Исходными элементами при создании массивов могут быть не только константы, но и переменные и математические выражения. Например, как на рисунке 2.5:

x=7.7
y=-2.4
z=2+3*i

Создадим вектор: **vector_x_y_z=[z, z^2, z*y, -z+x, x*y]**

Получится массив:

vector_x_y_z=|2-3i, -5-12i, -4.8+7.2i, 5.7+3i, -18.48|

Рисунок 2.5 – Создание массивов из констант, переменных и математических выражений

Матрицу можно представить как один вектор, собранный из столбцов матрицы, например, полученный массив **matrix**:

1, 1.2, 1.3, 1.4, 1.6, 1.8, 2, 2, 2.2, 2.4, 2.6, 2.8, 3, 4, 3.8, 3.6, 3.4, 3.2, 3

Тогда возможен доступ к ее элементам по номеру элемента этого вектора **k**, который можно определить:

$k = n \times (j - 1) + i,$

где **n** – число строк матрицы; **j** – номер столбца; **i** – номер строки.

В массиве **matrix** число строк равно **3**. Например, **matrix(5)** и **matrix(2,2)** (строка №2, столбец №2) – это один и тот же элемент матрицы (**k = 3×(2-1)+2**), равный **2.2**.

matrix(5:7) – это список следующих элементом массива **matrix**:

matrix(2,2) (строка №2, столбец №2 **k =** 3×(2-1)+2 = 5), равен 2.2;

matrix(3,2) (строка №3, столбец №2 **k =** 3×(2-1)+3 = 6), равен 3.8;

matrix(1,3) (строка №1, столбец №3 **k =** 3×(3-1)+1 = 7), равен 1.4.

2.4. Определение размеров массивов

Для определения числа столбцов матрицы **matrix** воспользуемся функцией **length(matrix)**. Единственным аргументом данной функции является матрица, длину которой необходимо измерить. Функция возвращает количество столбцов матрицы. Например:

matrix = [1:0.2:2;2 2.2 2.4 2.6 2.8 3;4:-0.2:3];

Получится массив, как на рисунке 2.6.

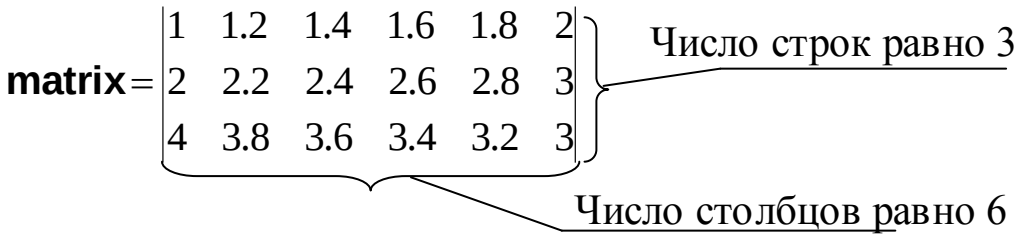


Рисунок 2.6 – Матрица matrix

len = length(matrix)

Переменная **len** получит значение **6** (число столбцов матрицы **matrix** равно **6**).

Если аргументом функции **length** является вектор, то результатом выполнения данной функции будет число элементов вектора:

vector = [2 2.2 2.4 2.6 2.8 3];

vlen = length(vector)

Переменная **vlen** получит значение **6** (число элементов вектора **vector** равно **6**).

Для определения всех размеров массивов служит функция **size(matrix)**. Если аргумент функции – двумерный массив, то функция возвращает вектор, состоящий из двух элементов, первый элемент – число строк матрицы, второй – число столбцов. Определим размеры массива **matrix**:

msize = size(matrix)

Значение переменной **msize** – вектор **(3,6)**, т. е. число строк массива **matrix** равно **3**, число столбцов равно **6**.

vsize = size(vector)

В данном случае **vsize** переменная получит значение **(1,6)**, то есть число строк массива **vector** равно **1**, число столбцов равно **6**.

Если аргументом функции **size** является трехмерный массив, данная функция вычислит вектор, состоящий из трех элементов: первый элемент – число страниц массива, второй элемент – число строк в страницах, третий – число столбцов в страницах. Для примера создадим трехмерный массив, используя двухмерный массив **matrix**.

Создадим первую страницу трехмерного массива **y**:

y(1,:,:) = matrix

Здесь первый индекс указывает на страницу, второй и третий – на строки и столбцы соответственно. Двоеточие на месте второго и третьего индексов означает соответственно все строки и все столбцы указанной страницы. Таким образом, первая страница массива **y** –

матрица **matrix**. На место второй и третьей страниц также поставим матрицу **matrix**:

y(2,:,:) = matrix

y(3,:,:) = matrix

Получился массив, как на рисунке 2.7.

Рисунок 2.7 – Трехмерный массив

Функция:

sy = size(y)

вернет значение **[3 3 6]** в вектор **sy**.

2.5. Изменение размеров массивов

Удаление строк. Пусть имеется двумерный массив **matrix**. Данный массив состоит из трех строк и шести столбцов. Следующая операция позволяет удалить из **matrix** строку номер **n**:

n = 2

matrix(n,:) = [ ]

Так как **n =** 2, то из массива **matrix** удалена вторая строка, теперь **matrix** имеет вид:

$$\mathbf{matrix} = \begin{vmatrix} 1 & 1.2 & 1.4 & 1.6 & 1.8 & 2 \\ 4 & 3.8 & 3.6 & 3.4 & 3.2 & 3 \end{vmatrix}$$

Теперь массив **matrix** состоит из двух строк и шести столбцов.

Удаление столбцов. Следующая операция позволяет удалить из **matrix** столбец:

n = 2

matrix(:,n) = [ ]

Так как **n =** 2, то из массива **matrix** удален второй столбец, теперь **matrix** имеет вид:

$$\mathbf{matrix} = \begin{vmatrix} 1 & 1.4 & 1.6 & 1.8 & 2 \\ 4 & 3.6 & 3.4 & 3.2 & 3 \end{vmatrix}$$

Теперь массив **matrix** состоит из двух строк и пяти столбцов.

Для удаления сразу нескольких строк или столбцов можно применять специальный знак объединения двоеточие «**:**»:

matrix(:,2:4) = [ ]

Теперь из массива **matrix** удалены столбцы со второго по четвертый, и массив **matrix** имеет вид:

$$\mathbf{matrix} = \begin{vmatrix} 1 & 2 \\ 4 & 3 \end{vmatrix}$$

Объединение массивов. Массив можно сформировать уже описанным в данной лекции методом с использованием квадратных скобок и разделителей (пробел, запятая, точка с запятой), используя в качестве элементов нового массива не скаляры, а векторы и матрицы. Таким способом получены матрица **matrix** из векторов **vec**, **vector** и **v** и трехмерный массив **y** из матриц **matrix**.

Для объединения массивов также служит функция

cat(n, m, m1, m2, …),

где **m, m1, m2, …** – массивы, которые будут объединяться; **n** – параметр, управляющий способом объединения: при **n=2** – горизонтальная конкатенация, при **n=1** – вертикальная конкатенация, а при **n=3** – создается трехмерный массив, при **n=4** – четырехмерный и т. д. Получим массив **matrix** из векторов **vec**, **vector** с помощью данной функции:

vec = 1:0.2:2;

vector = [2 2.2 2.4 2.6 2.8 3];

v = 4:-0.2:3;

matrix=cat(1,vec,vector,v)

Так как **n=1**, в результате получилась такая же матрица, как и при организации массива из векторов **vec**, **vector** и **v** с помощью квадратных скобок и точек с запятой:

$$\mathbf{matrix} = \begin{vmatrix} 1 & 1.2 & 1.4 & 1.6 & 1.8 & 2 \\ 2 & 2.2 & 2.4 & 2.6 & 2.8 & 3 \\ 4 & 3.8 & 3.6 & 3.4 & 3.2 & 3 \end{vmatrix}$$

Трехмерный массив **y** из матриц **matrix**:

y=cat(3, matrix, matrix, matrix)

Результат такого объединения такой же, как и в случае организации трехмерного массива с помощью квадратных скобок.

Создадим две матрицы **m1** и **m2**:

m1=[1.1 1.2 1.3;1.4 1.5 1.6]

m2=[2.1,2.2;2.3,2.4]

Их математическое изображение:

$$\mathbf{m1}=\begin{vmatrix}1.1 & 1.2 & 1.3\\ 1.4 & 1.5 & 1.6\end{vmatrix} \text{ и } \mathbf{m2}=\begin{vmatrix}2.1 & 2.2\\ 2.3 & 2.4\end{vmatrix}$$

Так как матрицы имеют одинаковое количество строк, из них можно составить матрицу с тем же количеством строк, соединив в один массив с помощью пробела или запятой, создавая строку матриц:

m=[m1 m2]

или с использованием запятой вместо пробела

m=[m1,m2]

или с помощью функции **cat**, первый параметр равен **2**, т. е. конкатенация горизонтальная

m=cat(2,m1,m2)

Число столбцов будет равно сумме числа столбцов исходных матриц. Получившаяся матрица представлена на рисунке 2.8.

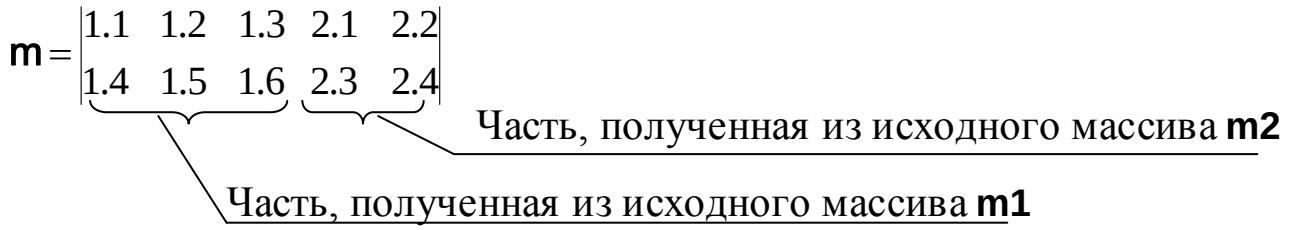


Рисунок 2.8 – Конкатенация горизонтальная

Таким способом можно объединить любое количество матриц при условии, что все исходные матрицы имеют одинаковое число строк.

Для того чтобы объединить матрицы в столбец, необходимо, чтобы все исходные матрицы состояли из одного и того же количества столбцов. Тогда полученный массив будет иметь то же число столбцов, что и исходные массивы, а число строк будет равно сумме числа строк исходных массивов. Создадим массивы, состоящие из двух столбцов и различного числа строк:

m3=[3.1,3.2;3.3,3.4]

m4=[4.1,4.2;4.3,4.4;4.5,4.6]

m5=[5.1,5.2;5.3,5.4;5.5,5.6;5.7,5.8]

Их математическое изображение:

$$m3 = \begin{vmatrix} 3.1 & 3.2 \\ 3.3 & 3.4 \end{vmatrix},\ m4 = \begin{vmatrix} 4.1 & 4.2 \\ 4.3 & 4.4 \\ 4.5 & 4.6 \end{vmatrix} \text{ и } m5 = \begin{vmatrix} 5.1 & 5.2 \\ 5.3 & 5.4 \\ 5.5 & 5.6 \\ 5.7 & 5.8 \end{vmatrix}$$

Теперь объединим полученные матрицы, применяя точку с запятой «**;**»:

mv=[m3;m4;m5]

или с помощью функции **cat**, первый параметр равен **1**, т. е. конкатенация вертикальная

mv=cat(1,m3,m4,m5)

Результат представлен на рисунке 2.9.

$$mv = \begin{vmatrix} 3.1 & 3.2 \\ 3.3 & 3.4 \\ 4.1 & 4.2 \\ 4.3 & 4.4 \\ 4.5 & 4.6 \\ 5.1 & 5.2 \\ 5.3 & 5.4 \\ 5.5 & 5.6 \\ 5.7 & 5.8 \end{vmatrix}$$

Часть матрицы, полученная из исходного массива **m3**

Часть матрицы, полученная из исходного массива **m4**

Часть матрицы, полученная из исходного массива **m5**

Рисунок 2.9 – Конкатенация вертикальная

К полученному массиву **mv** присоединим справа еще один столбец, в котором, как и в **mv**, должно быть **9** строк (рис. 2.10).

y=[mv [m3(:,1); m3(:,2); m5(:,1); 2]]

$$y = \begin{vmatrix} 3.1 & 3.2 & 3.1 \\ 3.3 & 3.4 & 3.3 \\ 4.1 & 4.2 & 3.2 \\ 4.3 & 4.4 & 3.4 \\ 4.5 & 4.6 & 5.1 \\ 5.1 & 5.2 & 5.3 \\ 5.3 & 5.4 & 5.5 \\ 5.5 & 5.6 & 5.7 \\ 5.7 & 5.8 & 2 \end{vmatrix}$$

Рисунок 2.10 – Присоединение столбца

Тот же результат получится, если применить функцию **cat**, первый параметр равен **1**, т. е. конкатенация вертикальная (рис. 2.11).

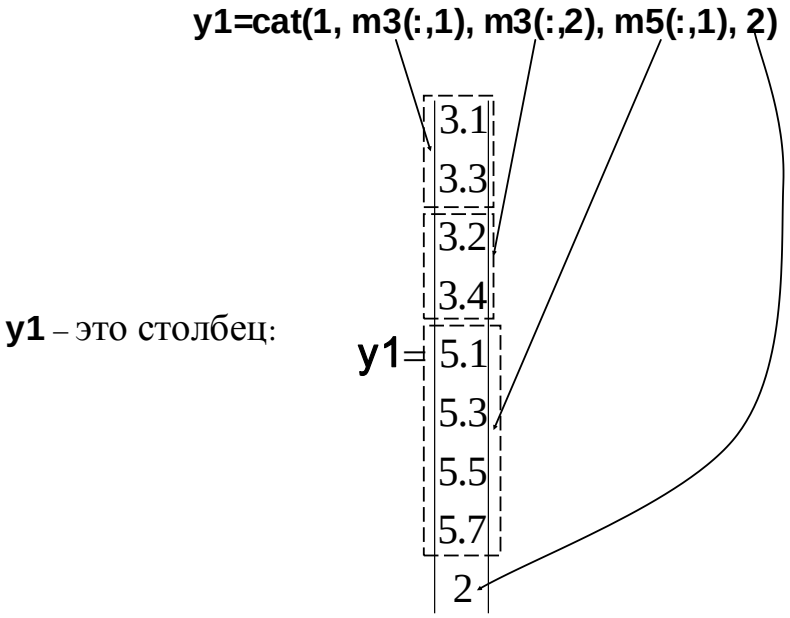


Рисунок 2.11 – Присоединение строк

Теперь объединим массив **mv** с массивом **y1** с помощью функции **cat**, первый параметр равен **2**, т. е. конкатенация горизонтальная:

y=cat(2, mv, y1)

То же можно сделать с помощью одной строки, если вместо идентификатора массива **y1** подставить функцию, с помощью которой получили массив **y1** (рис. 2.12).

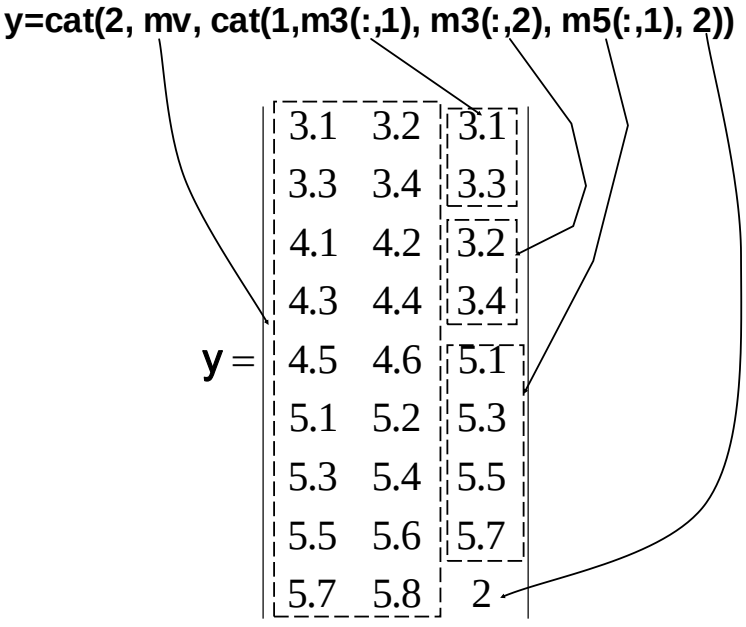


Рисунок 2.12 – Присоединение массивов

2.6. Применение арифметических операторов к массивам

Система Matlab позволяет применять арифметические операторы к массивам типа **double**.

Сложение. Для того чтобы к каждому элементу массива, например матрицы **m3**, прибавить скаляр, например константу **1**, достаточно записать **m3+1**.

Результат сложения

$$\begin{vmatrix} 4.1 & 4.2 \\ 4.3 & 4.4 \end{vmatrix}$$

Можно сложить два массива, имеющих одинаковое число строк и столбцов, поэлементно, таким образом, что

$$\mathbf{m_{ij}=m1_{ij}+m2_{ij},\ i=1\ldots n;\ j=1\ldots m,}$$

где **m** – массив, являющийся результатом поэлементного сложения массивов **m1** и **m2**; **i** и **j** – индексы, указывающие на номера строк и номера столбцов соответственно элементов массивов **m**, **m1** и **m2**; **n** и **m** – число строк и столбцов массивов **m**, **m1** и **m2**.

То есть результатом сложения будет массив той же размерности, что и массивы-слагаемые, каждый элемент массива-суммы будет равен сумме элементов массивов-слагаемых, имеющих одинаковые индексы. Например, создадим еще одну матрицу размерами **2×2**:

m6=[11 12;21 22]

Получилась матрица

$$\mathbf{m6}=\begin{vmatrix} 11 & 12 \\ 21 & 22 \end{vmatrix}$$

Сложение матриц:

msum=m3+m6

Формирование элементов массива **msum** следующим образом:

$$\mathbf{msum}=\mathbf{m6}+\mathbf{m3}=\begin{vmatrix} 3.1 & 3.2 \\ 3.3 & 3.4 \end{vmatrix}+\begin{vmatrix} 11 & 12 \\ 21 & 22 \end{vmatrix}=\begin{vmatrix} 3.1+11 & 3.2+12 \\ 3.3+21 & 3.4+22 \end{vmatrix}=\begin{vmatrix} 14.1 & 15.2 \\ 24.3 & 25.4 \end{vmatrix}$$

Вычитание. Применение операторов вычитания скаляра из всех элементов массива и поэлементного вычитания массивов, имеющих одинаковое число строк и столбцов, происходит аналогично.

Умножение. Также работает и оператор умножения массива на скаляр, умножим, например, матрицу **m6** на **2**:

m62=m6*2

Результат:

$$\mathbf{m62}=\mathbf{m6}\times\mathbf{2}=\begin{vmatrix} 11\times 2 & 12\times 2 \\ 21\times 2 & 22\times 2 \end{vmatrix}=\begin{vmatrix} 22 & 24 \\ 42 & 44 \end{vmatrix}$$

Для поэлементного перемножения массивов необходимо, чтобы массивы-сомножители имели одинаковое число строк и столбцов, перед знаком умножения необходимо ставить точку, оператор умножения будет выглядеть: «**.***» тогда

$$\mathbf{m_{ij}=m1_{ij}\times m2_{ij},\ i=1\ldots n;\ j=1\ldots m},$$

где **m** – массив, являющийся результатом поэлементного умножения массивов **m1** и **m2**; **i** и **j** – индексы, указывающие на номера строк и номера столбцов соответственно элементов массивов **m**, **m1** и **m2**, **n** и **m** – число строк и столбцов массивов **m**, **m1** и **m2**.

То есть результатом умножения будет массив той же размерности, что и массивы-сомножители, а каждый элемент полученного массива будет равен произведению элементов массивов-сомножителей, имеющих одинаковые индексы. Например, перемножим поэлементно массивы **m6** и **m3**:

prodm=m6.*m3

Результат

$$\mathrm{prodm}=\begin{vmatrix} 11\times 3.1 & 12\times 3.2 \\ 21\times 3.3 & 22\times 3.4 \end{vmatrix}=\begin{vmatrix} 34.1 & 38.4 \\ 69.3 & 74.8 \end{vmatrix}$$

Система Matlab позволяет произвести матричное перемножение матриц и векторов **C=A*B**, при этом оператор умножение «*» без точки, умножение системой Matlab производится по формуле

$$\mathbf{c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{nj},\ (i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n)},$$

где **c** – элемент матрицы **C**; **a** и **b** – элементы матриц **A** и **B**; **m** – число столбцов матрицы **A** и число строк матрицы **B**; **n** – число столбцов матрицы **B** и число строк матрицы **A**.

Таким образом, необходимо, чтобы число столбцов первой матрицы и число строк второй были одинаковыми, также одинаковыми должны быть число строк первой матрицы и число столбцов второй, а переместительный закон при матричном умножении не действует. Этот же принцип умножения матриц иллюстрирует данный алгоритм (рис. 2.13), в котором символы **a**, **b**, **m** и **n** имеют те же значения, что и в приведенной выше формуле.

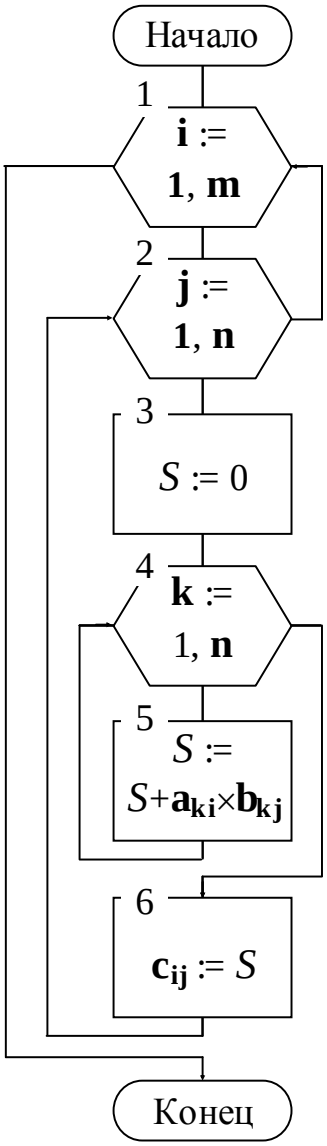


Рисунок 2.13 – Алгоритм матричного перемножения матриц и векторов

Для примера организуем две матрицы **A** размером **2×3** (**2** строки и **3** столбца) и матрицу **B** размером **3×2** (**3** строки и **2** столбца):

A=[11 12 13;21 22 23]

B=[1.1 1.2;2.1 2.2;3.1 3.2]

Математическое изображение этих матриц:

$$\mathbf{A} = \begin{vmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{vmatrix}; \mathbf{B} = \begin{vmatrix} 1.1 & 1.2 \\ 2.1 & 2.2 \\ 3.1 & 3.2 \end{vmatrix}$$

Найдем произведение матриц **A** и **B**:

prodAB=A*B

Произведение матриц **A** и **B**

$$\mathbf{prodAB}=$$

$$=\begin{vmatrix} a_{11}\times b_{11}+a_{12}\times b_{21}+a_{13}\times b_{31} & a_{11}\times b_{12}+a_{12}\times b_{22}+a_{13}\times b_{32} \\ a_{21}\times b_{11}+a_{22}\times b_{21}+a_{23}\times b_{31} & a_{21}\times b_{12}+a_{22}\times b_{22}+a_{23}\times b_{32} \end{vmatrix}=$$

$$=\begin{vmatrix} 11\times 1.1+12\times 2.1+13\times 3.1 & 11\times 1.2+12\times 2.2+13\times 3.2 \\ 21\times 1.1+22\times 2.1+23\times 3.1 & 21\times 1.2+22\times 2.2+23\times 3.2 \end{vmatrix}=$$

$$=\begin{vmatrix} 77.6 & 81.2 \\ 140.6 & 147.2 \end{vmatrix}$$

Деление. Рассмотрим несколько случаев из тех, что могут возникнуть при делении массивов данных. Для примеров создадим две квадратные матрицы **A** и **B** и скаляр **x**:

A=[1 6 2;9 5 7;3 2 1];

B=[1 2 3; 8 3 6;4 2 9];

x=2;

Получились матрицы:

$$\mathbf{A}=\begin{bmatrix} 1 & 6 & 2 \\ 9 & 5 & 7 \\ 3 & 2 & 1 \end{bmatrix} \text{ и } \mathbf{B}=\begin{bmatrix} 1 & 2 & 3 \\ 8 & 3 & 6 \\ 4 & 2 & 9 \end{bmatrix}$$

При правом делении матрицы на скаляр, которое может быть запрограммировано следующим образом:

C=A / x;

или

C=A ./ x;

Результатом является матрица той же размерности, что и матрица-делимое, элементы полученной матрицы – частное от деления элементов матрицы-делимого на скаляр-делитель, т. е. деление матрицы на скаляр происходит поэлементно. Результат:

$$\mathbf{C}=\frac{\mathbf{A}}{\mathbf{x}}=\begin{bmatrix} 1 & 6 & 2 \\ 9 & 5 & 7 \\ 3 & 2 & 1 \end{bmatrix} \Big/ 2=\begin{bmatrix} 1/2 & 6/2 & 2/2 \\ 9/2 & 5/2 & 7/2 \\ 3/2 & 2/2 & 1/2 \end{bmatrix}=\begin{bmatrix} 0.5 & 3 & 1 \\ 4.5 & 2.5 & 3.5 \\ 1.5 & 1 & 0.5 \end{bmatrix}$$

При правом делении скаляра на матрицу необходимо указывать на поэлементный способ деления, т. е. ставить точку перед знаком деления:

C1=x ./ A;

Результат: матрица той же размерности, что и матрица-делитель; элементы полученной матрицы – частное от деления скаляра-делимого на элементы матрицы-делителя:

$$\mathbf{C1}=\frac{\mathbf{x}}{\mathbf{A}}=2\Big/\begin{bmatrix}1&6&2\\9&5&7\\3&2&1\end{bmatrix}=\begin{bmatrix}2/1&2/6&2/2\\2/9&2/5&2/7\\2/3&2/2&2/1\end{bmatrix}=\begin{bmatrix}2&0.33&1\\0.22&0.4&0.29\\0.67&1&2\end{bmatrix}$$

К такому же результату приведет левое поэлементное деление матрицы на скаляр:

C1=A .\ 2

$$\mathbf{C1}=2\Big/\begin{bmatrix}1&6&2\\9&5&7\\3&2&1\end{bmatrix}=\begin{bmatrix}2/1&2/6&2/2\\2/9&2/5&2/7\\2/3&2/2&2/1\end{bmatrix}=\begin{bmatrix}2&0.33&1\\0.22&0.4&0.29\\0.67&1&2\end{bmatrix}$$

Если две матрицы имеют одинаковую размерность, то возможно поэлементное деление одной матрицы на другую. Частное от деления – матрица той же размерности, элементы которой – результат деления элементов делимого на элементы делителя, имеющих одинаковые индексы. Правое поэлементное деление:

D=A ./ B

$$\mathbf{D}=\frac{\begin{bmatrix}1&6&2\\9&5&7\\3&2&1\end{bmatrix}}{\begin{bmatrix}1&2&3\\8&3&6\\4&2&9\end{bmatrix}}=\begin{bmatrix}1/1&6/2&2/3\\9/8&5/3&7/6\\3/4&2/2&1/2\,9\end{bmatrix}=\begin{bmatrix}1&3&0.67\\1.13&1.67&1.17\\0.75&1&0.11\end{bmatrix}$$

Левое поэлементное деление:

D1=A .\ B

$$\mathbf{D1}=\frac{\begin{bmatrix}1&2&3\\8&3&6\\4&2&9\end{bmatrix}}{\begin{bmatrix}1&6&2\\9&5&7\\3&2&1\end{bmatrix}}=\begin{bmatrix}1/1&2/6&3/2\\8/9&3/5&6/7\\4/3&2/2&9/1\end{bmatrix}=\begin{bmatrix}1&0.33&1.5\\0.89&0.6&0.86\\1.33&1&9\end{bmatrix}$$

Если матрицы квадратные и имеют одинаковую размерность, то матричное деление одной матрицы на другую программируется следующей строкой:

M=A / B;

Данное действие дает тот же результат, что и матричное произведение первой матрицы на матрицу обратную второй: **M=A/B=A×B-1**, таким образом, необходимо, чтобы матрица-делитель была невырожденной матрицей, т. е. ее определитель отличался от нуля.

При левом матричном делении квадратных матриц:

M1=A \ B;

Результат: как при обратном матричном умножении первой матрицы на вторую: **M1=A\B=A 1× B**, тогда, необходимо, чтобы матрица-делимое была невырожденной, т. е. ее определитель отличался от нуля.

Если матрицы **A** и **B** не квадратные, то деление работает как решатели систем линейных уравнений: действие **X=A \ B** находит решение системы уравнений вида **AX=B**, где **A** – прямоугольная матрица размера **m×n** и **B** – матрица размера **n×k**, а **X=B / A** находит решение системы уравнений вида **XA=B**, где **A** – прямоугольная матрица размера **n×m** и **B** – матрица размера **m×k**.

Возведение в степень. При поэлементном возведении в степень массива, когда показатель степени – скаляр, результатом будет массив той же размерности, каждый элемент которого равен элементу исходного массива, возведенного в соответствующую степень, например:

M2=A.^x;

Результат:

$$\mathbf{M2} = \begin{bmatrix} 1^2 & 6^2 & 2^2 \\ 9^2 & 5^2 & 7^2 \\ 3^2 & 2^2 & 1^2 \end{bmatrix} = \begin{bmatrix} 1 & 36 & 4 \\ 81 & 25 & 49 \\ 9 & 4 & 1 \end{bmatrix}$$

Если показатель степени – массив, то при поэлементном возведении в степень он должен иметь ту же размерность, что и возводимый в степень массив, тогда каждый элемент первого массива будет возведен в степень, показатель которой – элемент второго массива, имеющий тот же самый индекс:

M3=A.^[2 3 4; 1 3 2; 4 5 6];

Результат:

$$\mathbf{M3} = \begin{bmatrix} 1^2 & 6^3 & 2^4 \\ 9^1 & 5^3 & 7^2 \\ 3^3 & 2^5 & 1^6 \end{bmatrix} = \begin{bmatrix} 1 & 216 & 16 \\ 9 & 125 & 49 \\ 81 & 32 & 1 \end{bmatrix}$$

Матричное возведение в степень возможно только квадратной матрицы, а показатель степени должен быть скаляром, тогда данное действие аналогично матричному умножению, когда всеми сомножителями является одна и та же матрица: **A^3=A*A*A**.

2.7. Использование встроенных функций Matlab при работе с массивами

Все встроенные математические функции Matlab могут применяться к массивам любых размеров. Для примера вычислим квадратный корень всех элементов матрицы **A**:

V = sqrt(A);

Результат:

$$\mathbf{V} = \begin{bmatrix} \sqrt{1} & \sqrt{6} & \sqrt{2} \\ \sqrt{9} & \sqrt{5} & \sqrt{7} \\ \sqrt{3} & \sqrt{2} & \sqrt{1} \end{bmatrix} = \begin{bmatrix} 1 & 2.45 & 1.41 \\ 3 & 2.24 & 2.65 \\ 1.73 & 1.41 & 1 \end{bmatrix}$$

2.8. Задания для лабораторной работы

2.8.1. Формирование вектора, просмотр элементов вектора, определение длины вектора

Для начала работы необходимо создать **M**-файл сценария. Выполнить следующие задания:

Сформировать вектор, используя квадратные скобки, состоящий из 10–15 элементов.

Вывести на экран значение отдельных двух элементов вектора.

Вывести на экран значение нескольких элементов, расположенных подряд.

Определить длину вектора.

Для выполнения задания необходимо использовать данные для вариантов заданий из пункта 2.9.

Пример выполнения задания – на рисунке 2.14.

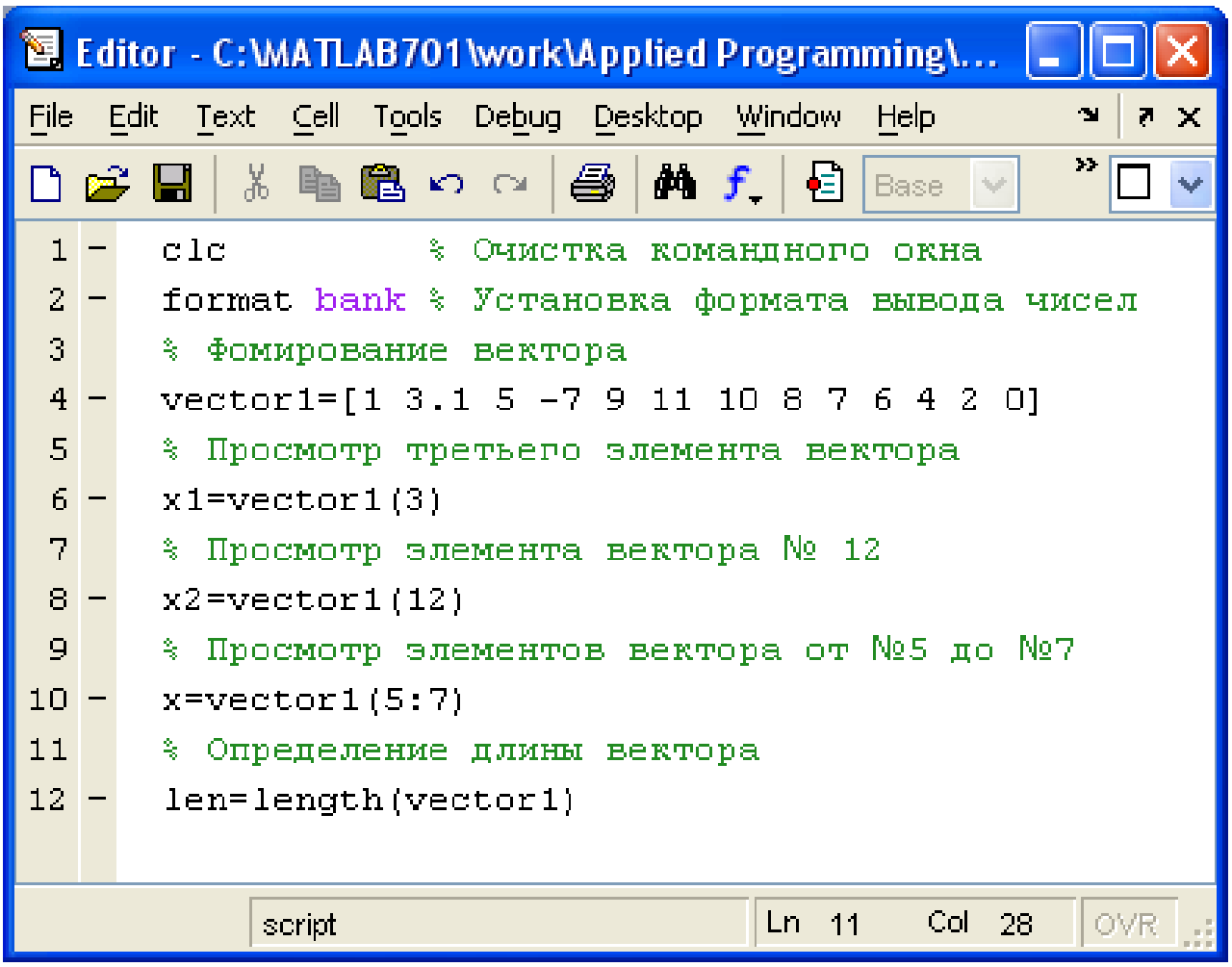


Рисунок 2.14 – Пример создания m-сценария

Запустить **m**-сценарий, просмотреть результат работы.

Результат работы **m**-сценария приведен на рисунке 2.15. В окне «**Command Window**» отображены результаты расчетов. Объяснить результат.

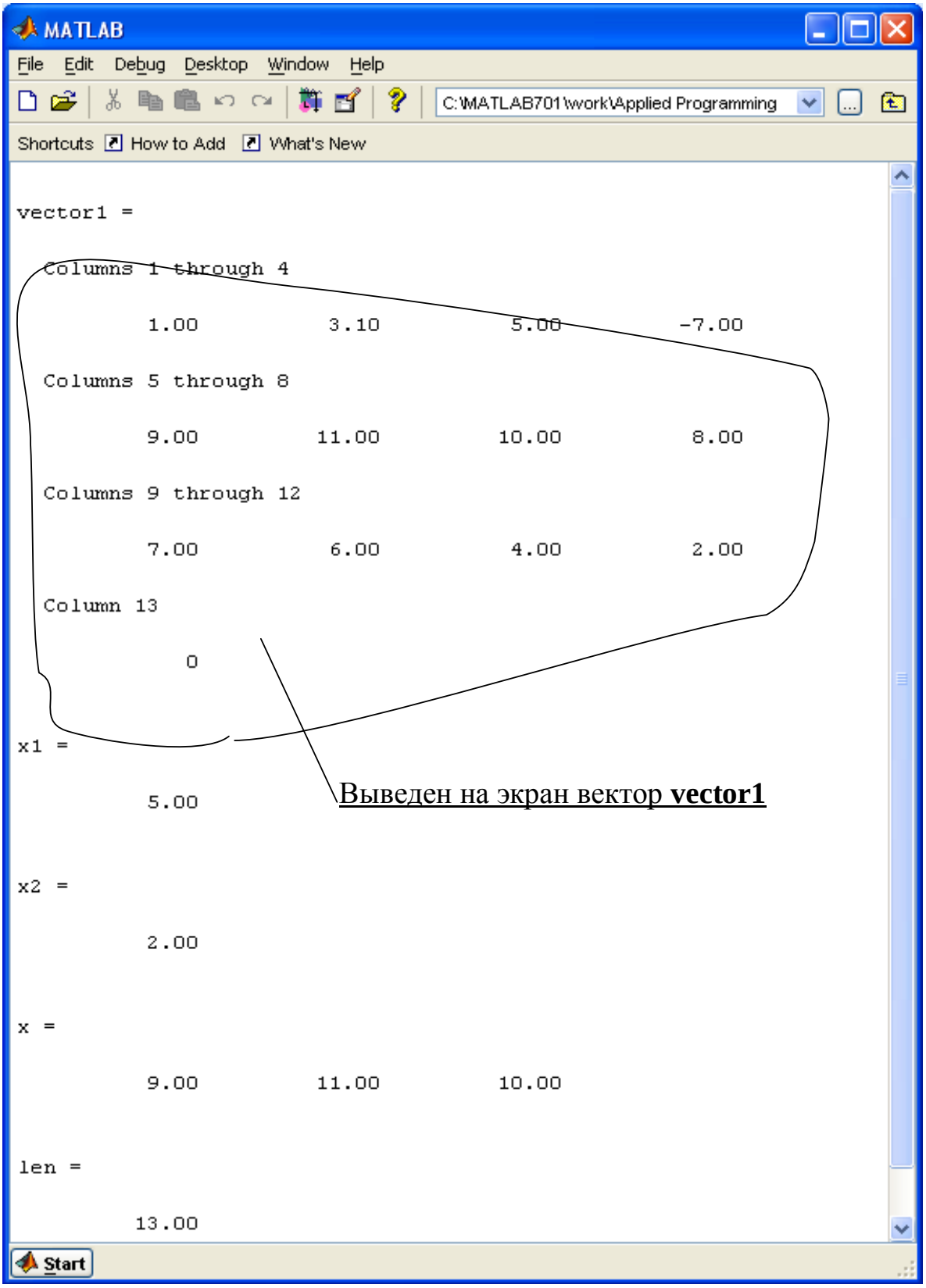


Рисунок 2.15 – Результат работы m-сценария

2.8.2. Вычисление математического выражения

Сформировать вектор, задав минимальное и максимальное значения и шаг изменения элементов вектора. Для выполнения задания необходимо использовать данные для вариантов заданий пункта 0.

Вычислить функцию от вектора (см. вариант задания в п. 2.9).

Построить график функции с помощью встроенной функции **plot(x,y)**, где **x** – вектор, элементы которого откладываются по оси абсцисс, где **y** – вектор, элементы которого откладываются по оси ординат.

Пример выполнения задания смотри на рисунке 2.16.

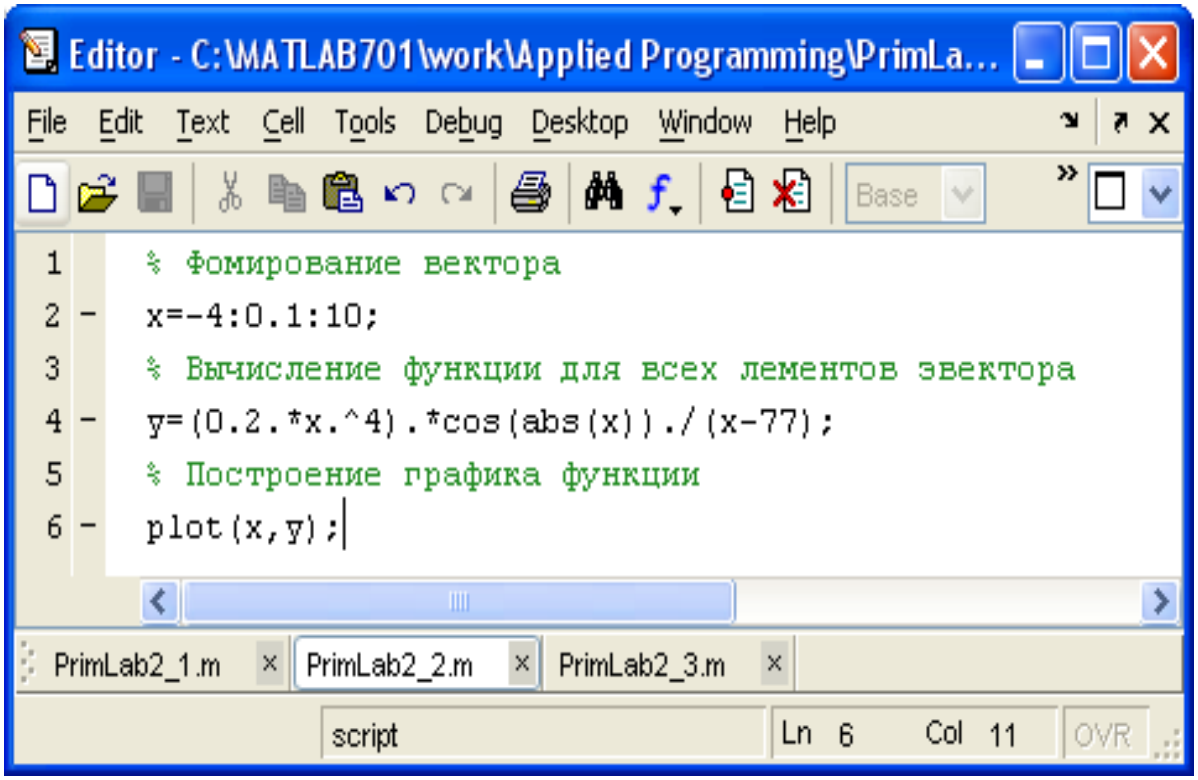


Рисунок 2.16 – m-сценарий вычисления математического выражения и построения графика функции

При работе приведенного на рисунке 2.16 **m**-сценария в окне «**Command Window**» не будут отражаться результаты вычислений, так как после математических выражений и функций стоят символы «;», подавляющие вывод результатов вычислений.

Будет построен график функции, как на рисунке 2.17. Объяснить результат.

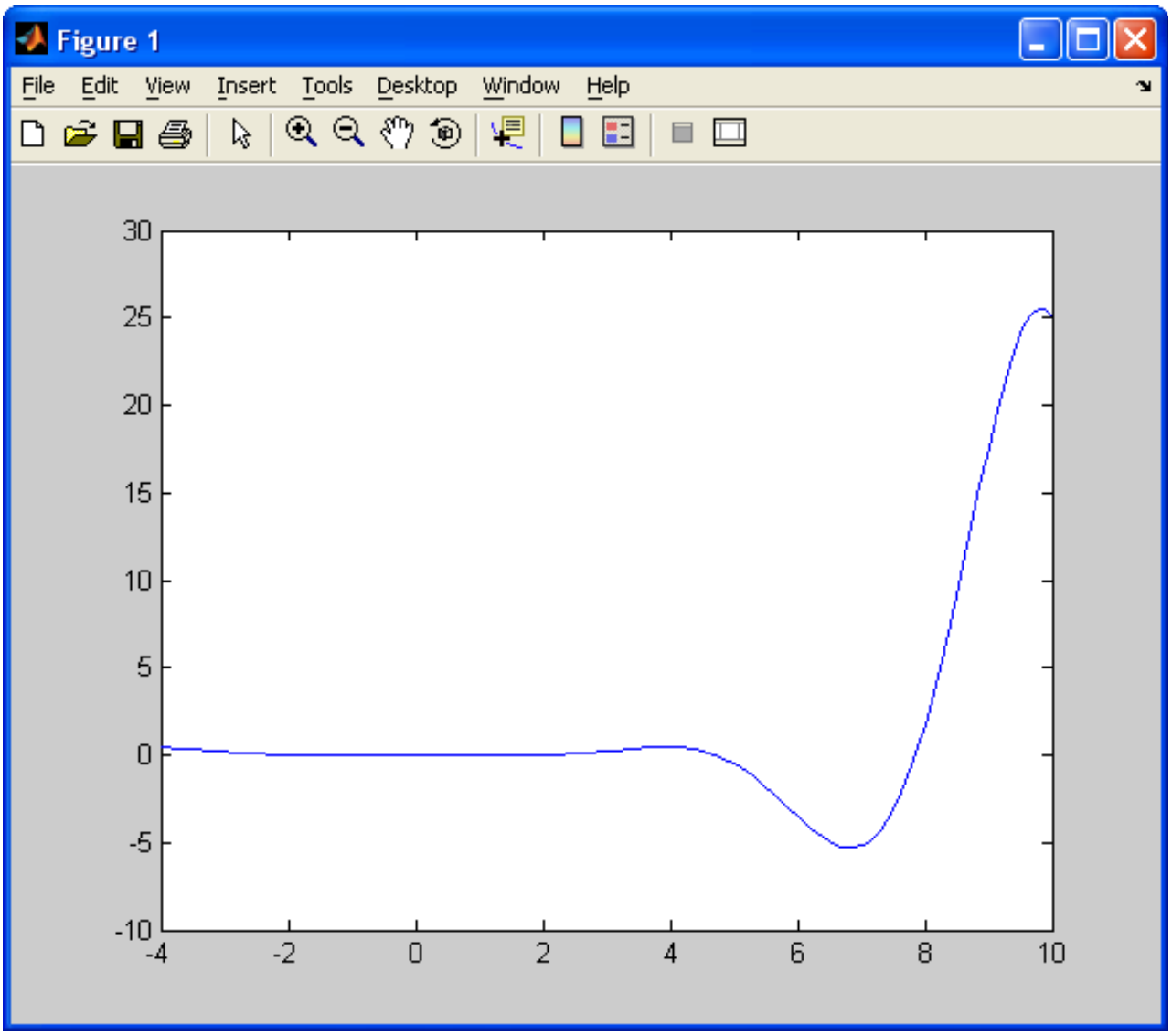


Рисунок 2.17 – График функции

2.8.3. Формирование матрицы и доступ к элементам матрицы

Сформировать матрицу. Для выполнения задания необходимо использовать данные для вариантов заданий из пункта 2.9.

Определить размеры матрицы.

Организовать просмотр отдельных элементов, целых строк и прямоугольных фрагментов матрицы.

Пример выполнения задания – на рисунке 2.18. Объяснить результат.

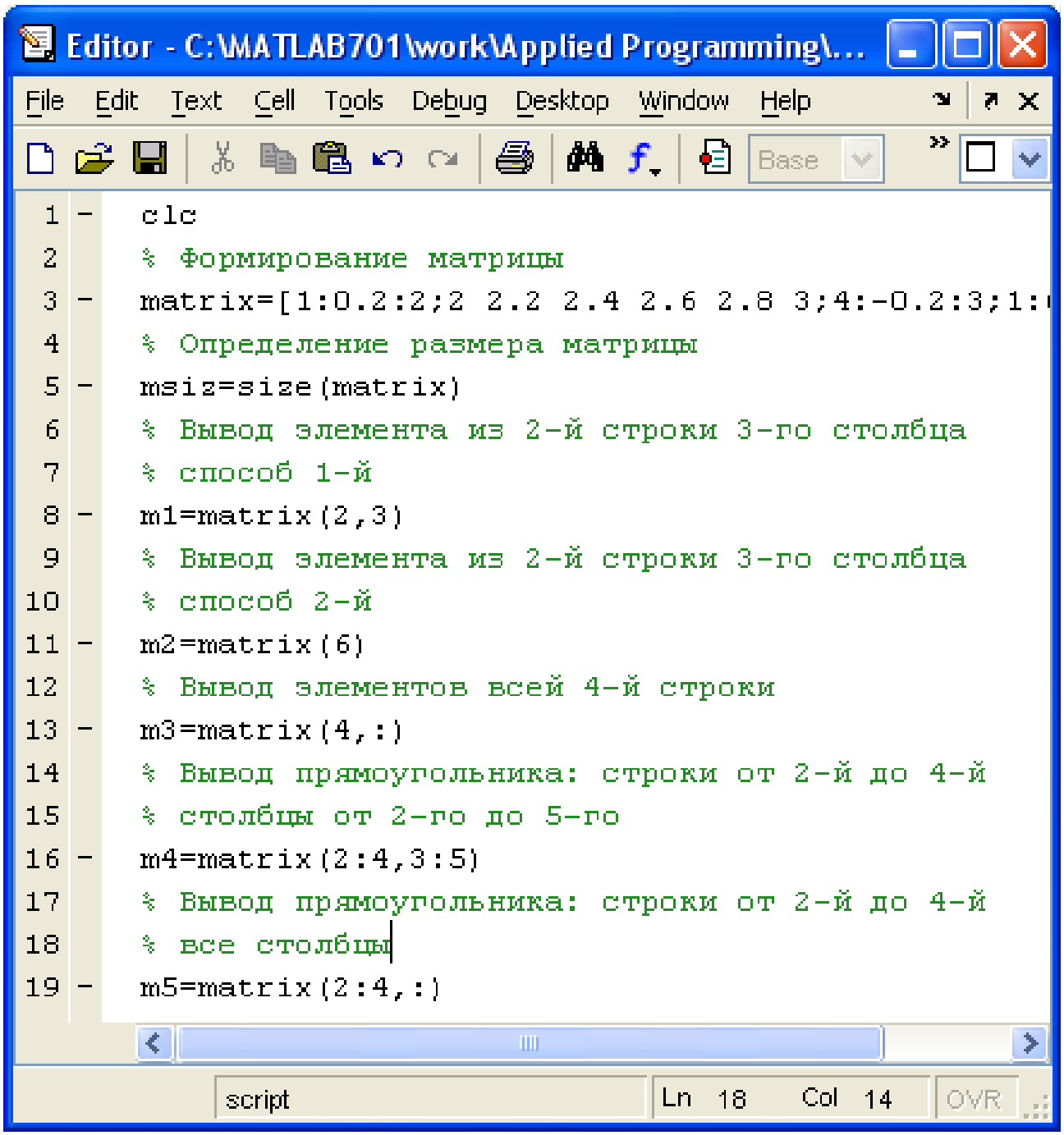


Рисунок 2.18 – m-сценарий формирования и вывода элементов матрицы

Результат работы **m**-сценария – на рисунке 2.19. Объяснить результат.

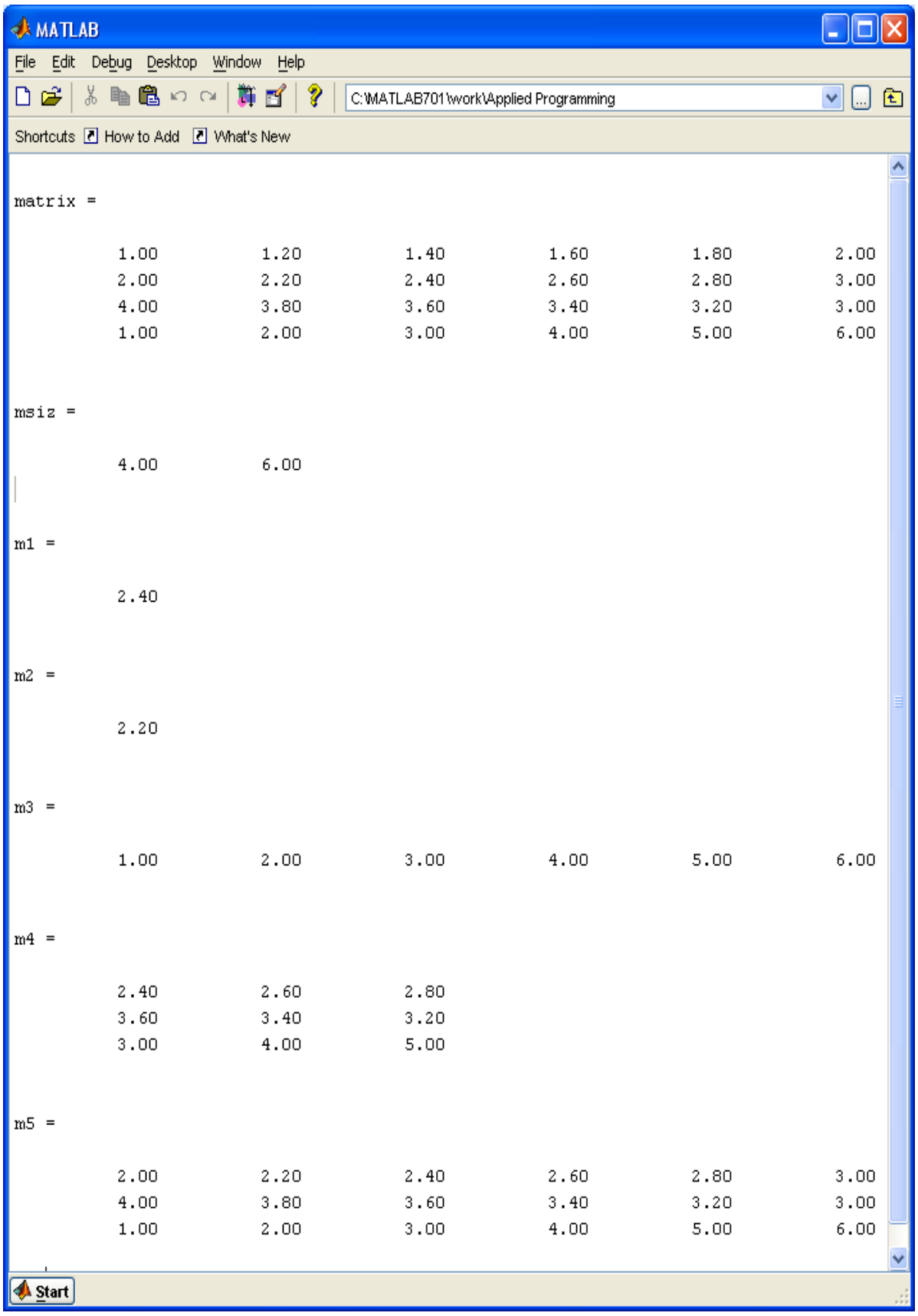


Рисунок 2.19 – Результат работы m-сценария формирования и вывода элементов матрицы

2.8.4. Удаление строк и столбцов матрицы

Пусть необходимо сделать следующие действия:
– сформировать матрицу;
– определить размеры матрицы;
– удалить одну строку из матрицы;
– определить новые размеры матрицы;
– удалить один столбец из матрицы;
– определить новые размеры матрицы.
Пример выполнения задания – на рисунке 2.20.

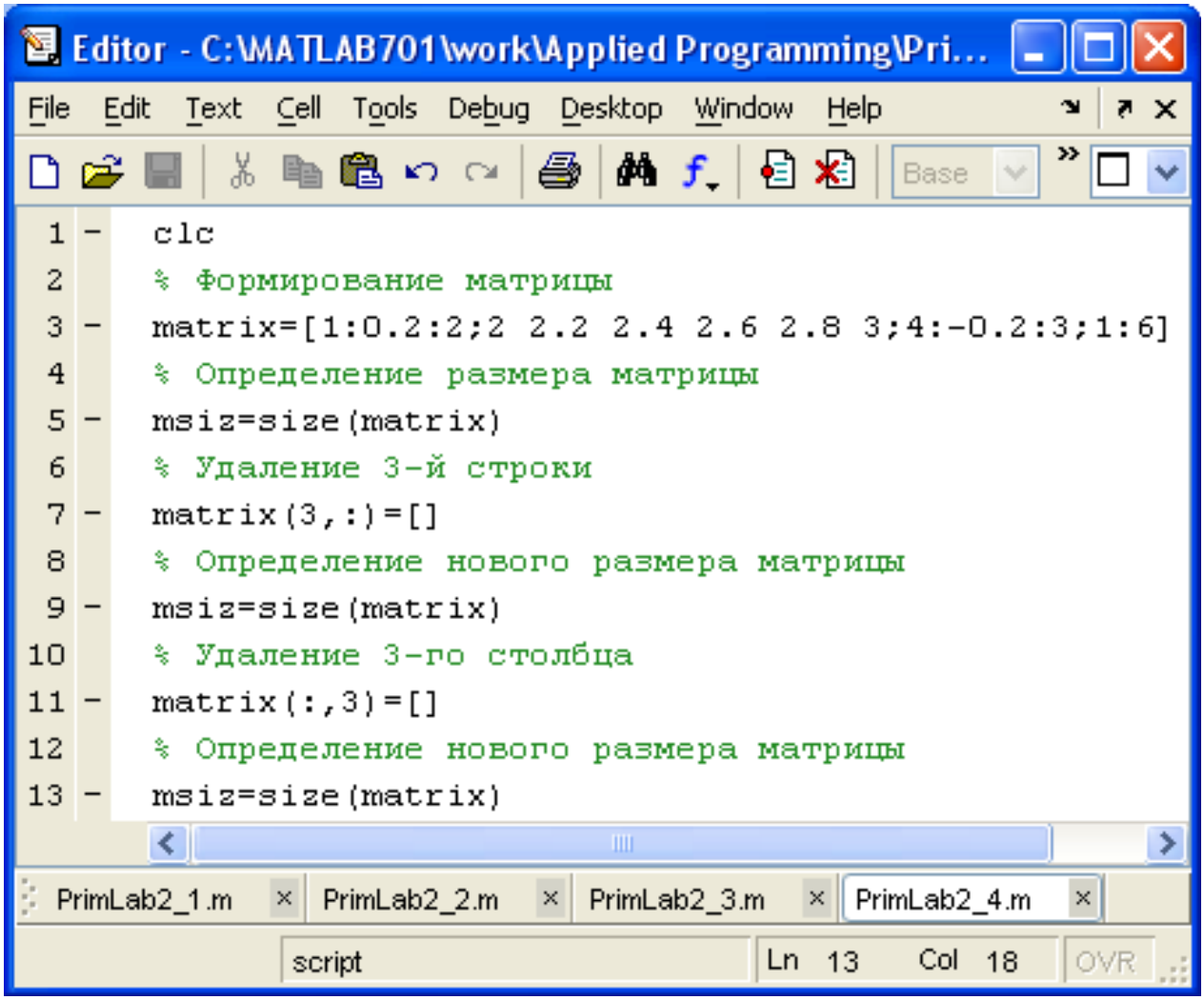


Рисунок 2.20 – m-сценарий удаления элементов матрицы

Результат работы **m**-сценария – на рисунке 2.21. Объяснить результат.

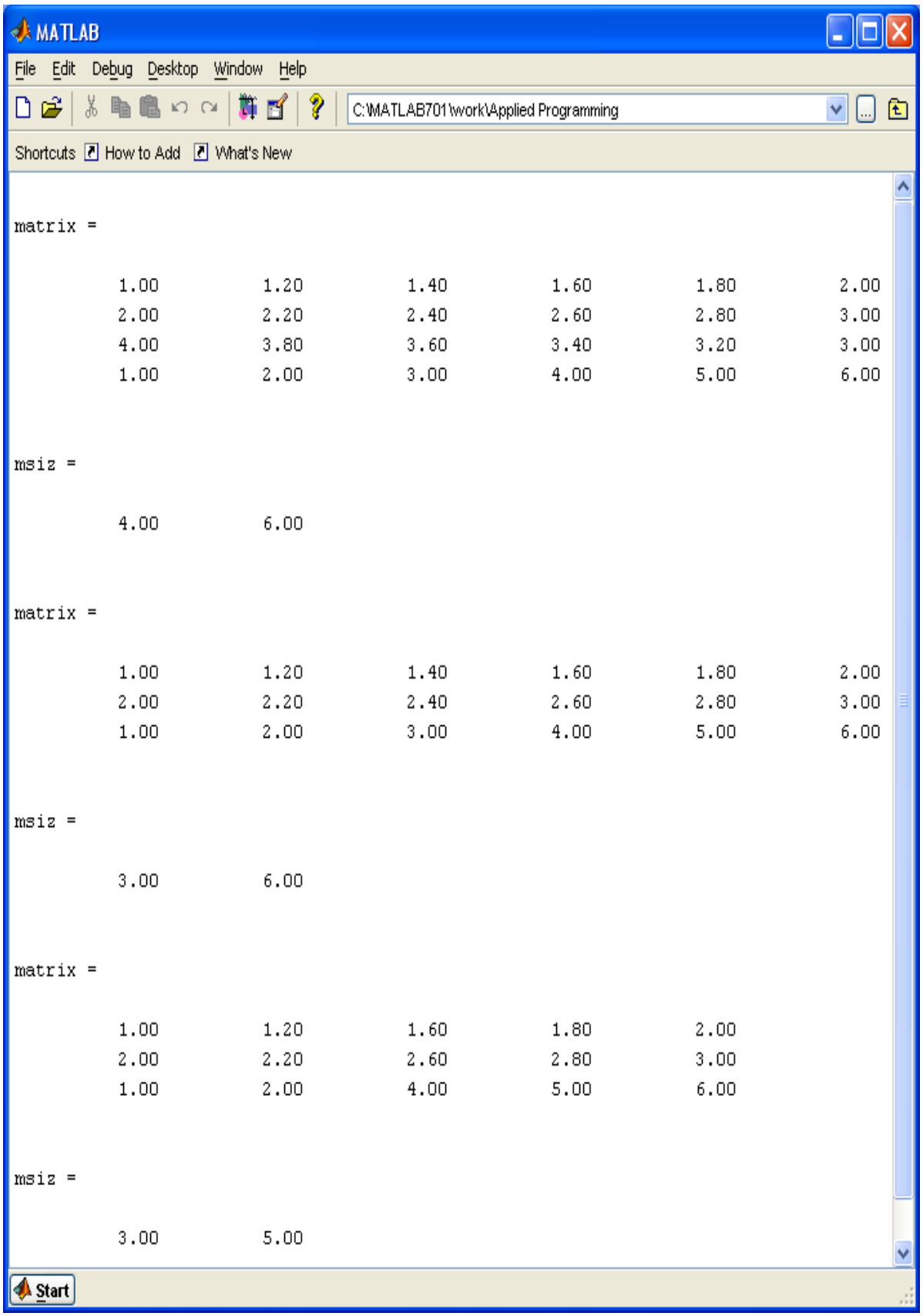


Рисунок 2.21 – Результат работы m-сценария удаления элементов матрицы

2.8.5. Объединение массивов

Сформировать три массива, имеющих одно и то же количество столбцов, например три вектора одинаковой длины.

Объединить массивы в одну матрицу с помощью вертикальной конкатенации.

Сформировать три массива, имеющих одно и то же количество строк.

Объединить массивы в одну матрицу с помощью горизонтальной конкатенации.

Пример выполнения задания – на рисунке 2.22.

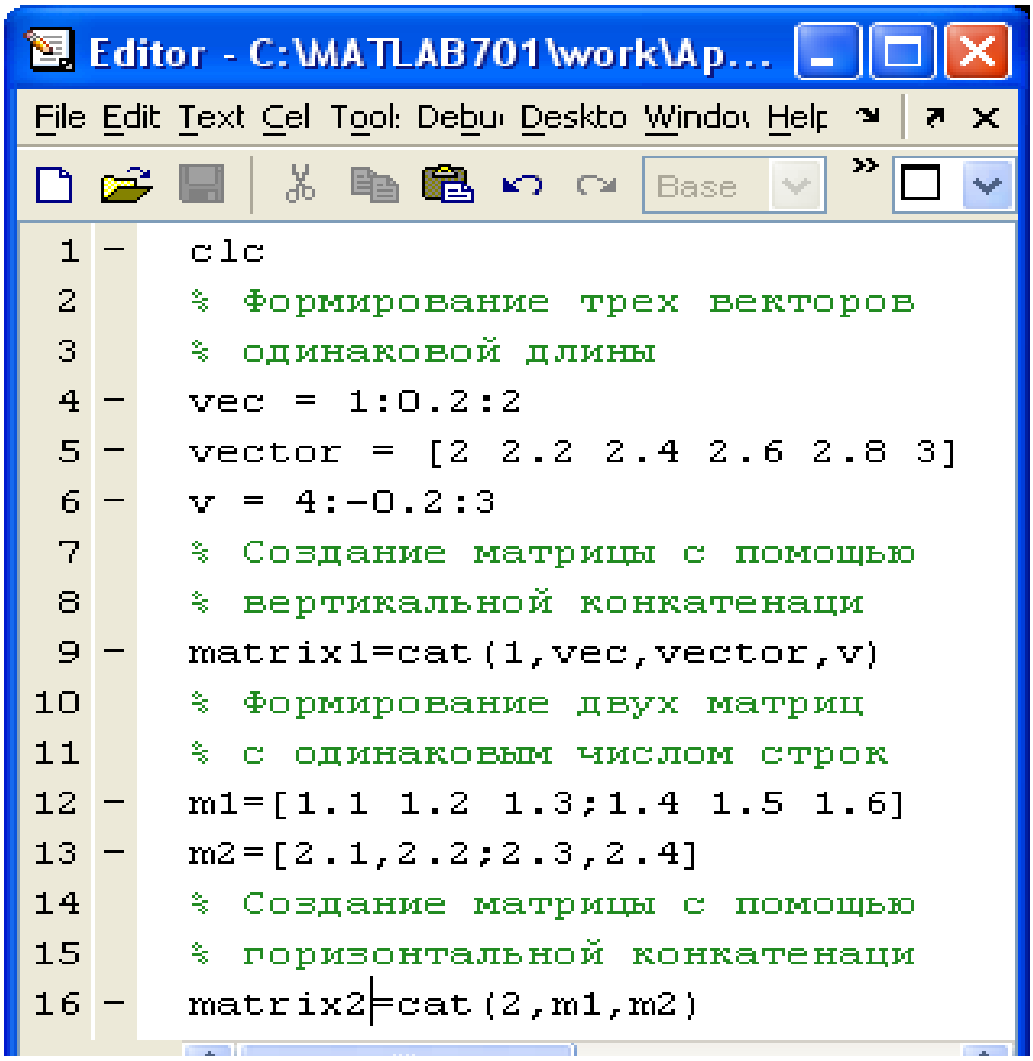


Рисунок 2.22 – m-сценарий применения вертикальной и горизонтальной конкатенации

Результат работы **m**-сценария – на рисунке 2.23. Объяснить результат.

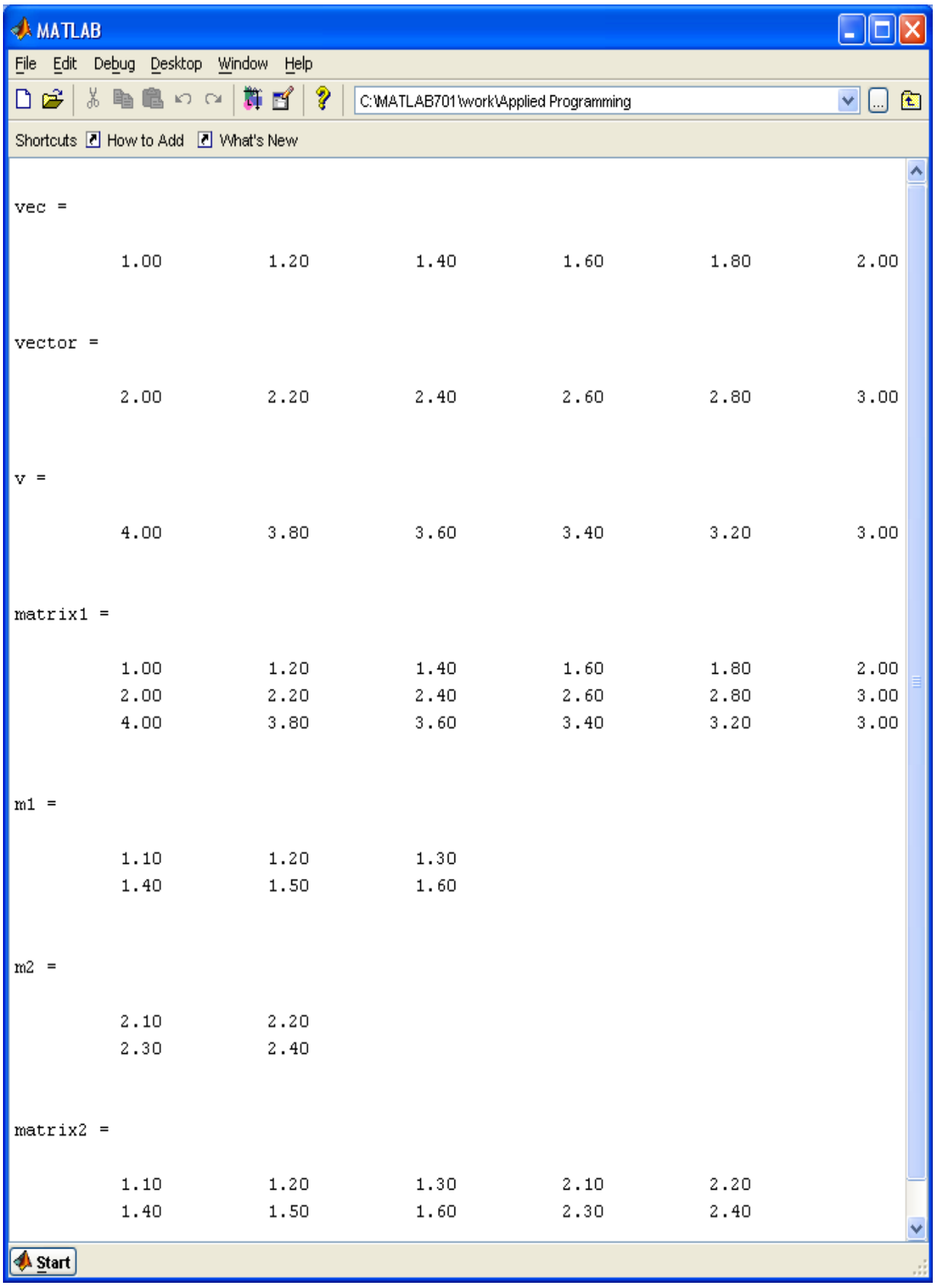


Рисунок 2.23 – Результат работы m-сценария вертикальной и горизонтальной конкатенации

2.8.6. Сложение и умножение матриц

Сформировать два массива, имеющих одно и то же количество столбцов и строк.

Получить сумму двух массивов. Перемножить массивы поэлементно.

Сформировать два массива, таких, где число строк первого равно числу столбцов второго, а число столбцов первого равно числу строк второго.

Получить матричное произведение двух массивов.

Пример выполнения задания – на рисунке 2.24.

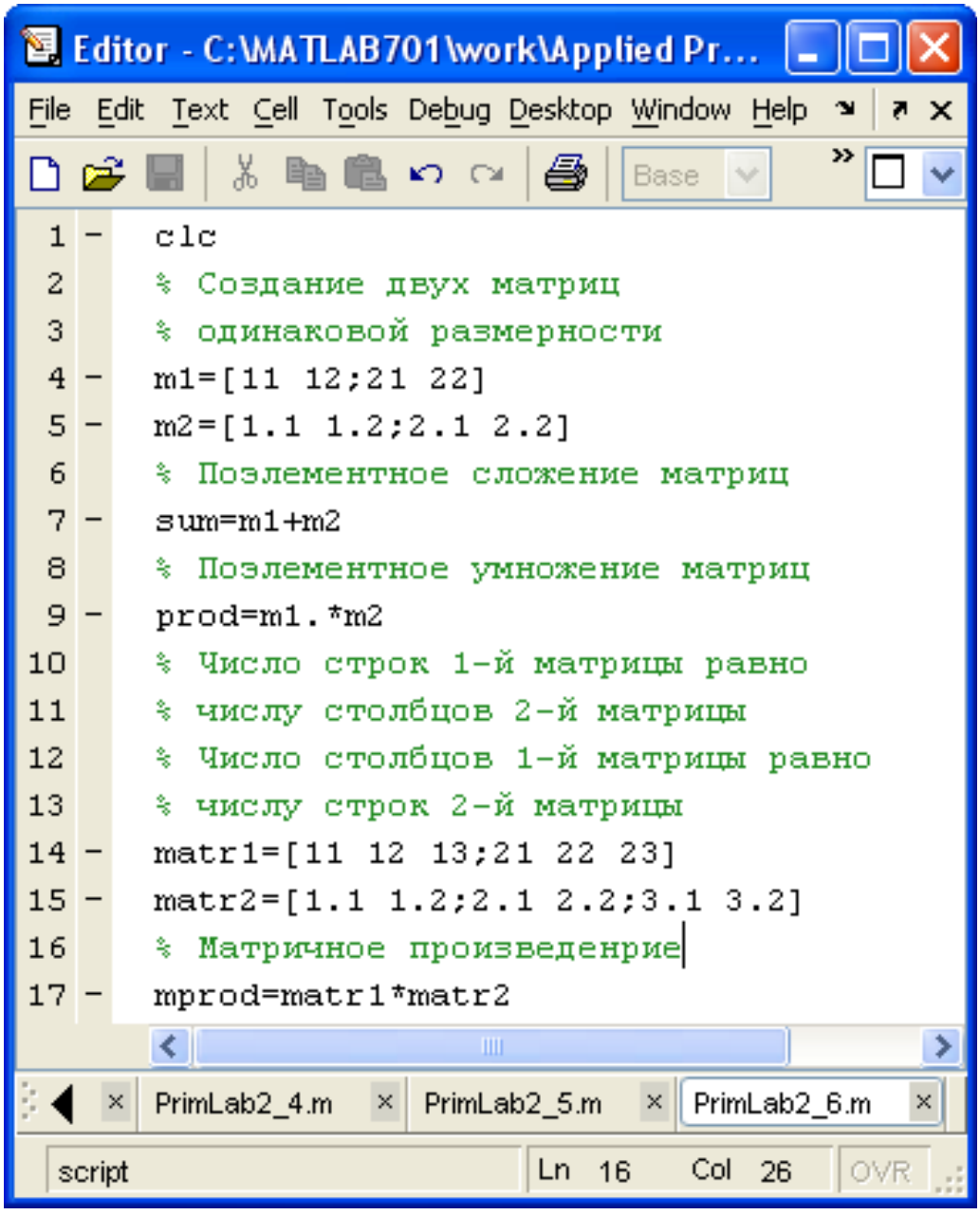


Рисунок 2.24 – m-сценарий сложения и умножения матриц

Результат работы **m**-сценария – на рисунке 2.25. Объяснить результат.

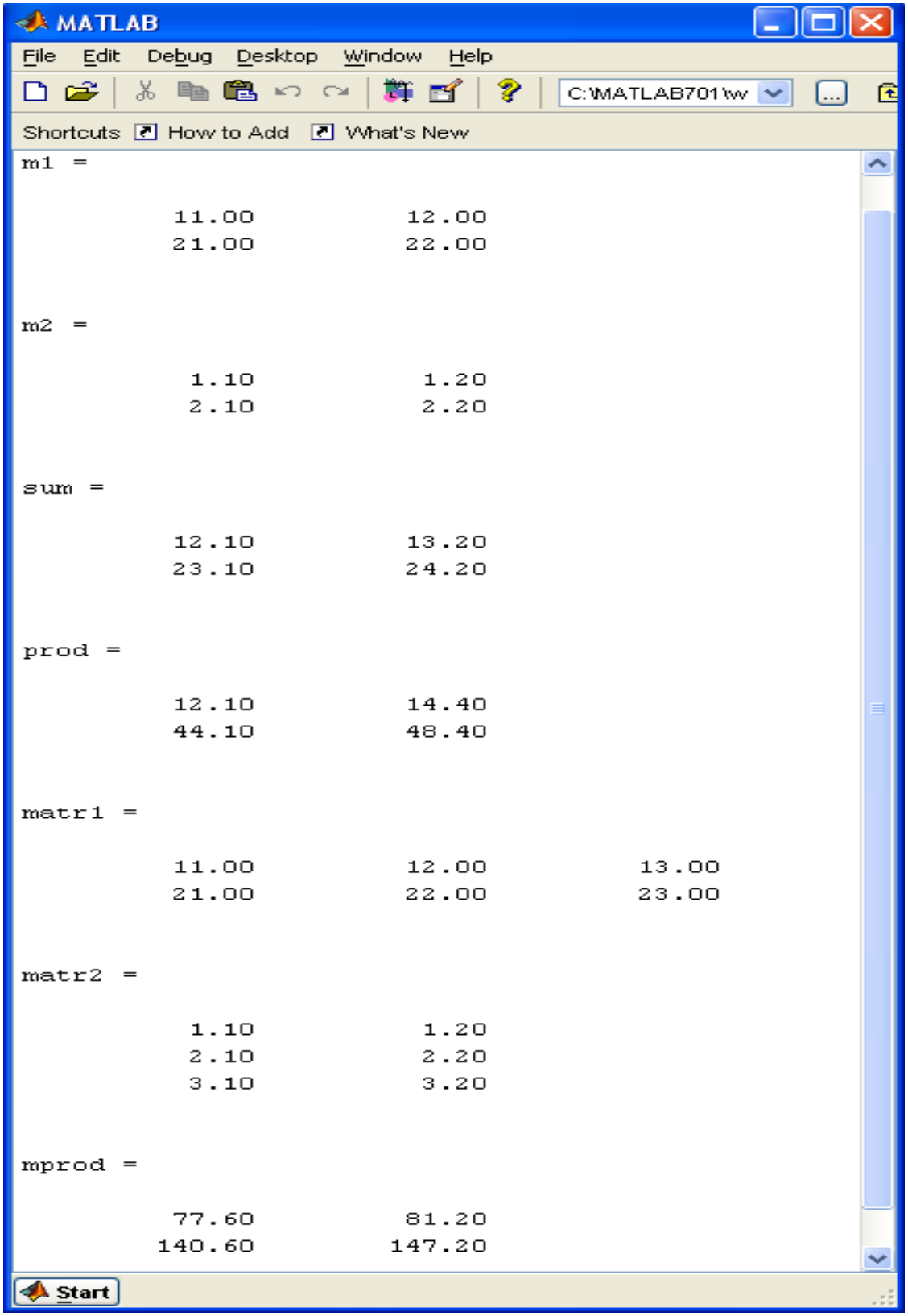


Рисунок 2.25 – Результат работы m-сценария сложения и умножения матриц

2.8.7. Деление матриц

Сформировать два массива, имеющих одно и то же количество столбцов и строк.

Разделить матрицу на скаляр.

Разделить скаляр на матрицу.

Организовать правое поэлементное деление одной матрицы на другую.

Организовать левое поэлементное деление одной матрицы на другую.

Пример выполнения задания – на рисунке 2.26.

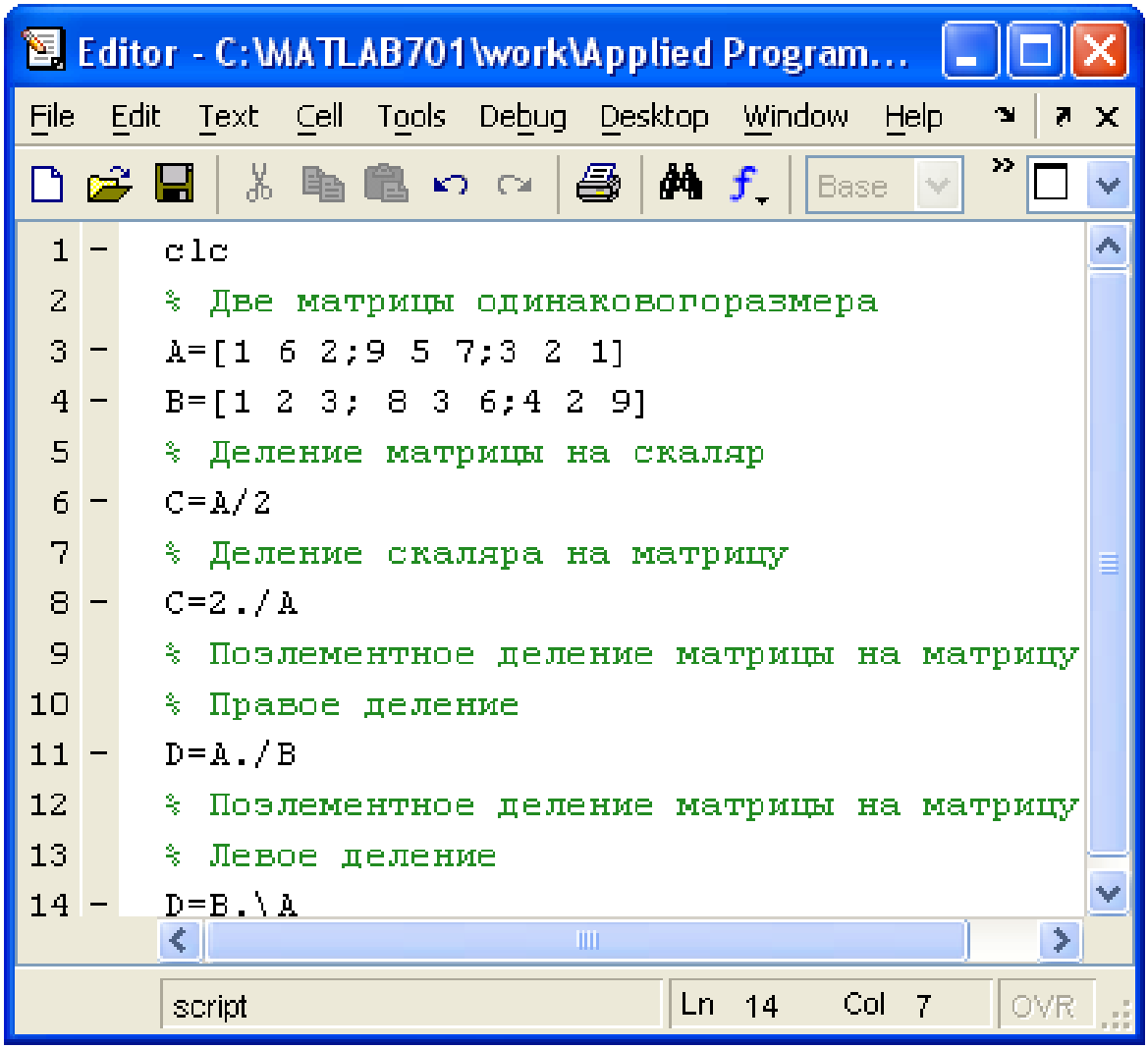


Рисунок 2.26 – m-сценарий деления матриц

Результат работы **m**-сценария – на рисунке 2.27. Объяснить результат.

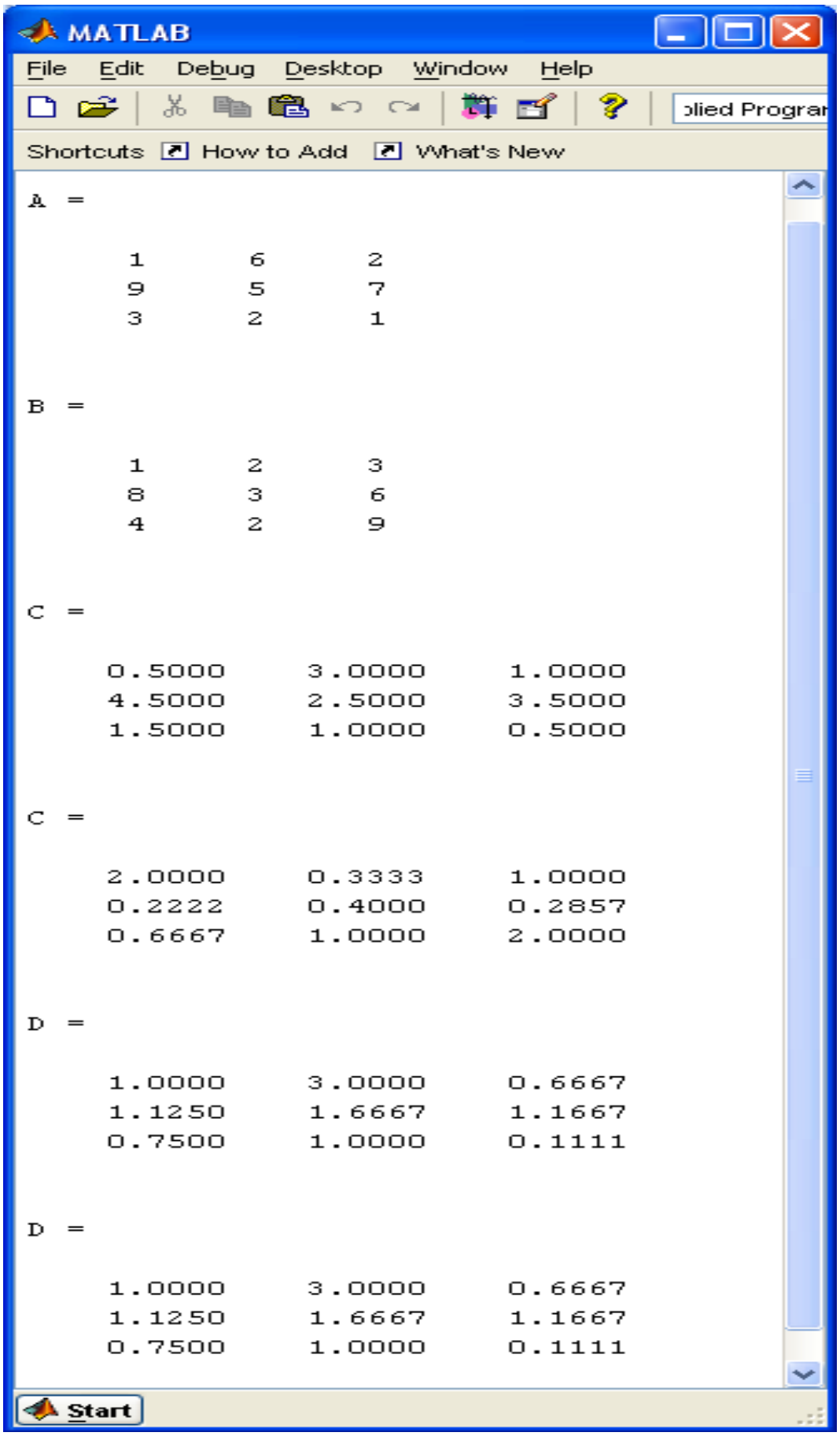


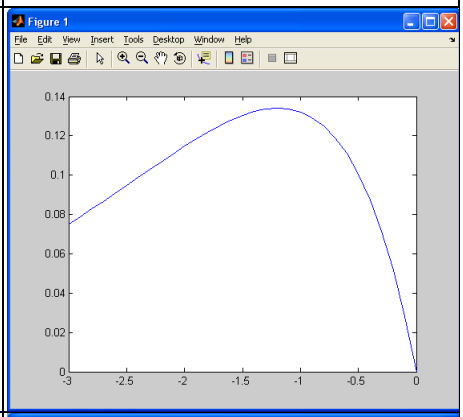
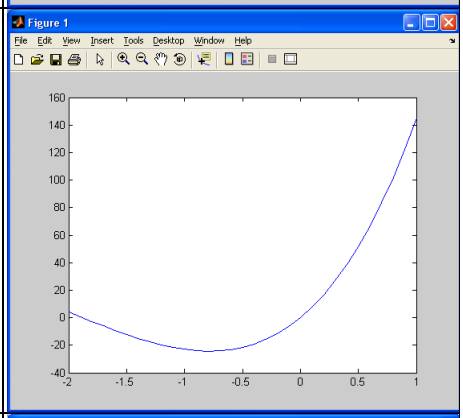
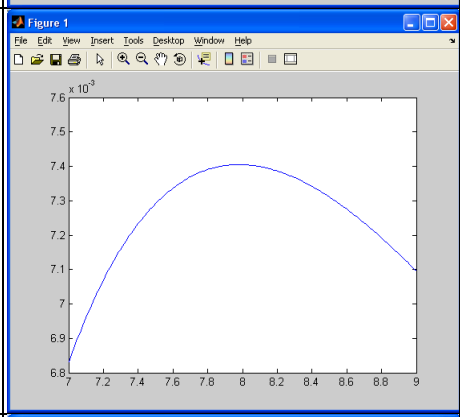
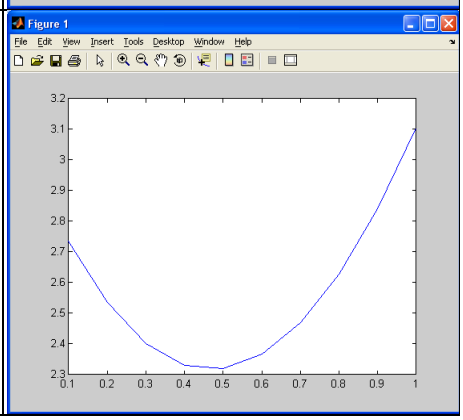
Рисунок 2.27 – Результат работы m-сценария деления матриц

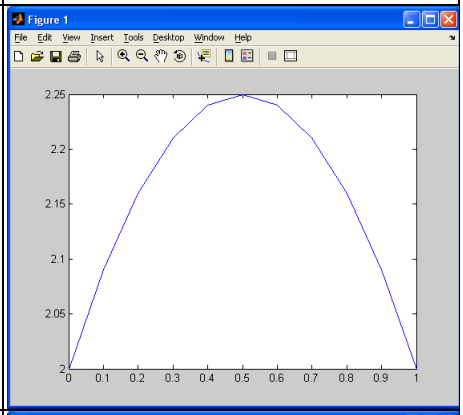
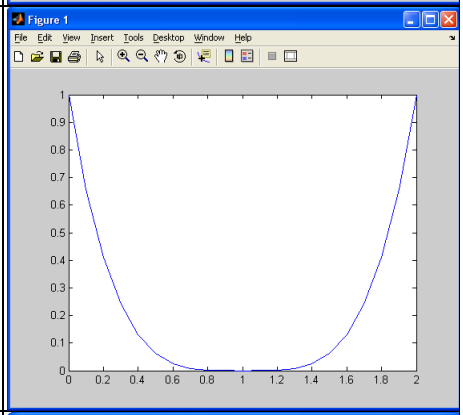
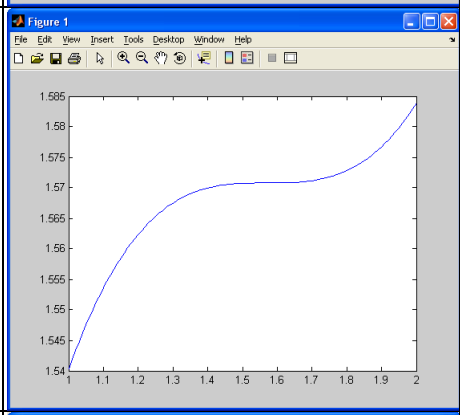
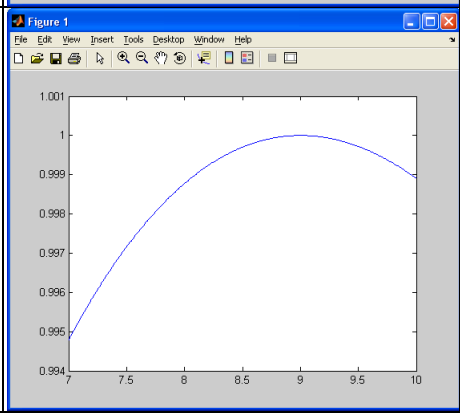
2.9. Варианты заданий

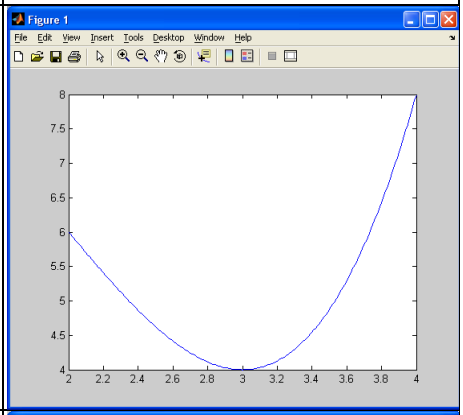
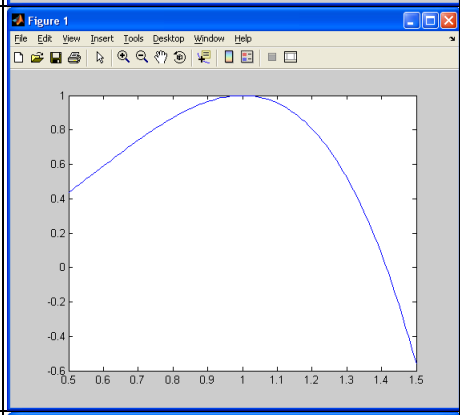
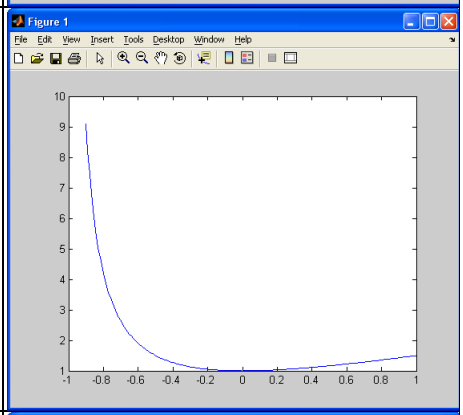
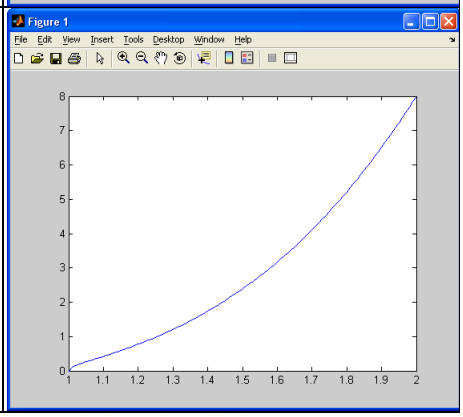
Формирование вектора, просмотр элементов вектора, определение длины вектора

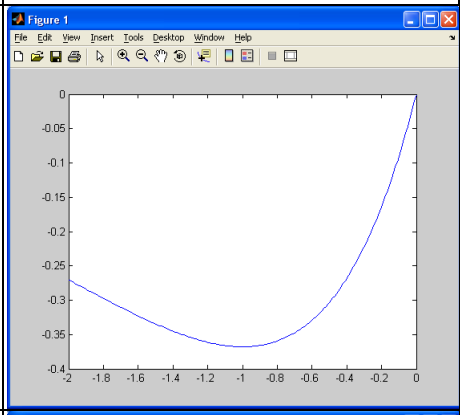
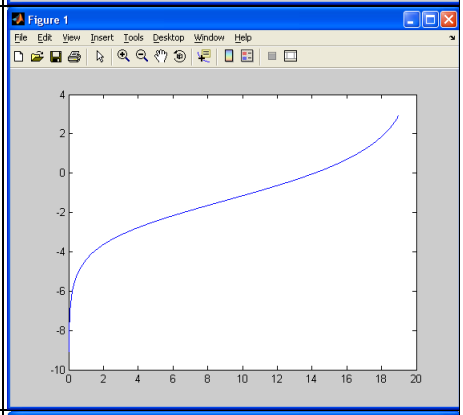
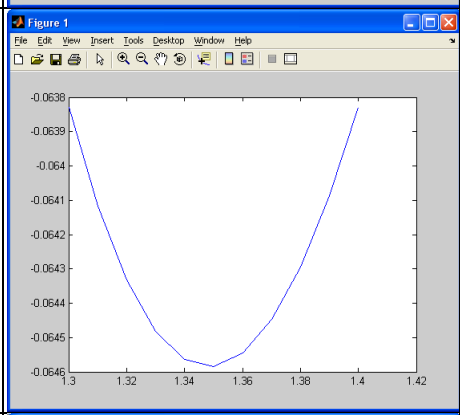
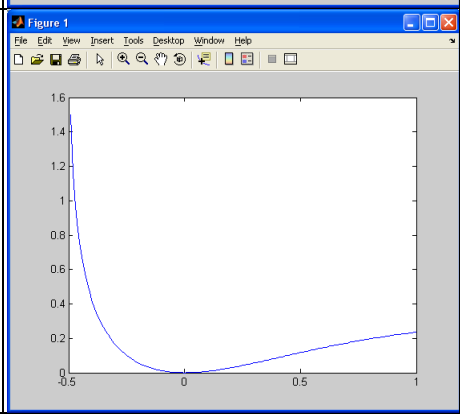
Номер варианта	Вектор	Номер элементов вектора, которые необходимо просмотреть
1	0,30 0,87 0,02 0,77 0,97 0,99 0,79 0,44 0,50 0,21 12	1, 6, с 7 до 11
2	0,63 0,72 0,69 0,08 0,45 0,44 0,35 0,15 0,68 0,70 0,73	2, 5, с 6 до 10
3	0,48 0,55 0,12 0,45 0,72 0,89 0,27 0,25 0,87 0,23 0,80	3, 4, с 7 до 10
4	0,91 0,23 0,24 0,05 0,08 0,64 0,19 0,84 0,17 0,17 0,99	4, 7, с 6 до 9
5	0,44 0,34 0,31 0,37 0,39 0,59 0,12 0,04 0,46 0,87 0,93	5, 3, с 5 до 8
6	0,26 0,16 0,87 0,24 0,65 0,97 0,66 0,87 0,01 0,14 0,82	6, 2, с 4 до 7
7	0,43 0,89 0,73 0,69 0,35 0,17 0,16 0,19 0,42 0,86 0,49	7, 1, с 3 до 6
8	0,82 0,46 0,46 0,45 0,41 0,90 0,01 0,30 0,05 0,69 0,65	8, 11, с 2 до 5
9	0,98 0,55 0,40 0,20 0,63 0,73 0,38 0,01 0,42 0,75 0,79	9, 10, с 1 до 4
10	0,92 0,84 0,37 0,62 0,73 0,19 0,90 0,57 0,63 0,23 0,55	10, 1, с 7 до 11
11	0,93 0,34 0,66 0,39 0,63 0,70 0,40 0,41 0,66 0,84 0,37	11, 9, с 6 до 10
12	0,43 0,59 0,57 0,72 0,51 0,78 0,49 0,19 0,70 0,98 0,81	1, 8, с 5 до 10
13	0,70 0,48 0,11 0,66 0,37 0,14 0,57 0,82 0,67 1,00 0,96	2, 7, с 4 до 8
14	0,06 0,36 0,55 0,26 0,60 0,05 0,57 0,70 0,96 0,75 0,74	3, 6, с 3 до 6
15	0,43 0,63 0,80 0,08 0,95 0,92 0,60 0,25 0,87 0,51 0,73	4, 3, с 2 до 7
16	0,42 0,96 0,07 0,55 0,29 0,86 0,34 0,68 0,05 0,36 0,50	5, 11, с 1 до 4
17	0,43 0,56 0,62 0,11 0,90 0,75 0,79 0,81 0,67 0,20 0,27	6, 5, с 7 до 11
18	0,63 0,54 0,06 0,09 0,27 0,41 0,47 0,91 0,60 0,33 0,48	7, 4, с 6 до 11
19	0,60 0,16 0,83 0,96 0,60 0,03 0,81 0,61 0,70 0,09 0,42	8, 3, с 5 до 10
20	0,38 0,17 0,83 0,84 0,45 0,96 0,15 0,87 0,77 0,44 0,62	9, 2, с 3 до 8
21	0,95 0,64 0,25 0,35 0,19 0,49 0,41 0,46 0,61 0,07 0,31	10, 1, с 2 до 6
22	0,61 0,18 0,62 0,25 0,59 0,51 0,46 0,54 0,94 0,34 0,40	11, 10, с 1 до 5
23	0,31 0,41 0,29 0,39 0,50 0,72 0,31 0,11 0,44 0,47 0,01	1, 11, с 4 до 9
24	0,66 0,72 0,28 0,26 0,71 0,78 0,99 0,47 0,90 0,45 0,80	2, 11, с 3 до 7

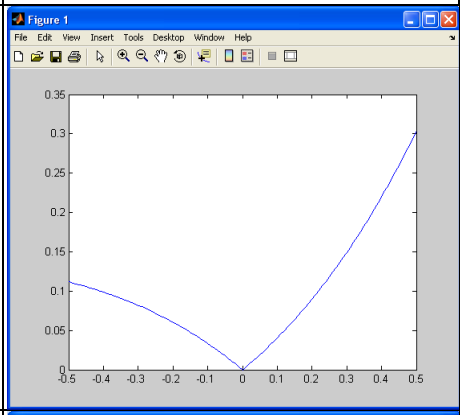
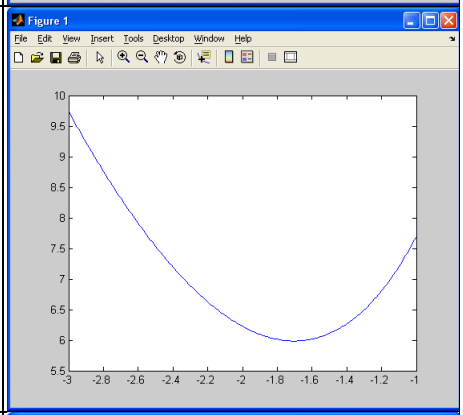
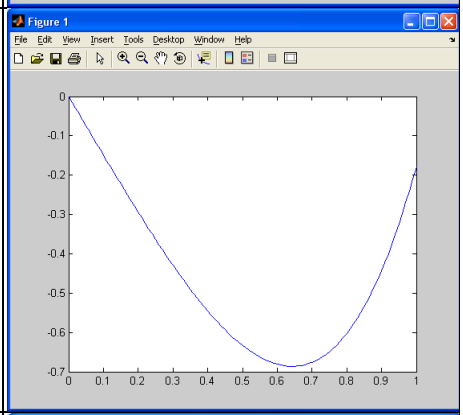
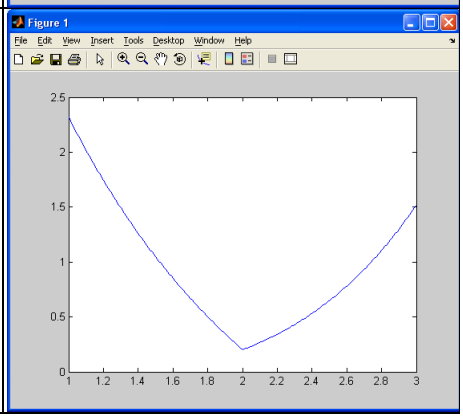
Вычисление математического выражения

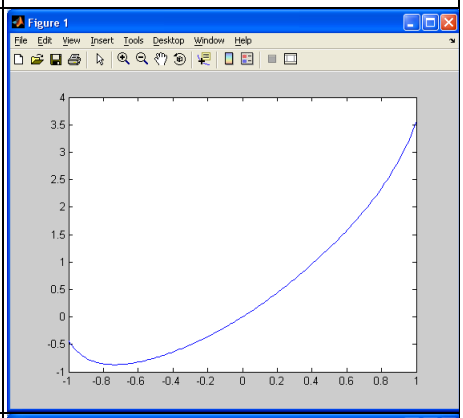
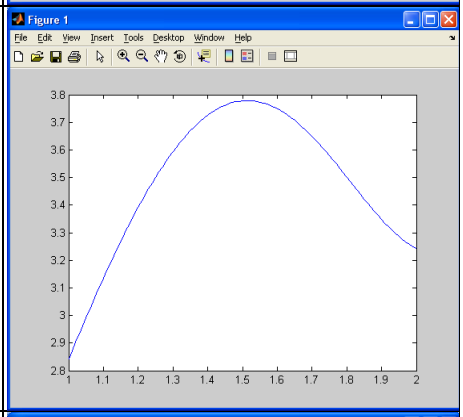
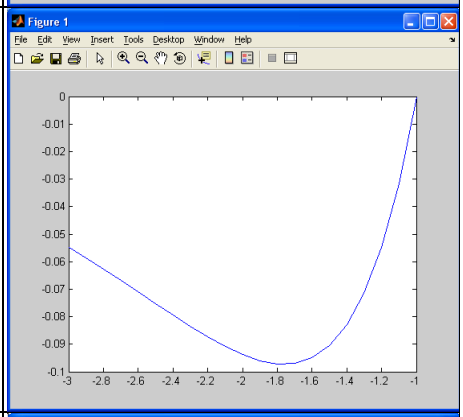
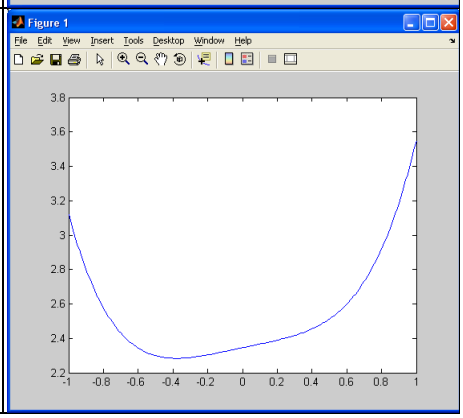
Номер варианта	Задание	Интервал значений x	Результат
1	$Q=2^x-e^x$	[-3; 0]	
2	$Q=x^4-14x^3+60x^2-70x$	[-2; 1]	
3	$Q=\frac{x^2-30}{x^4+7.7x^2+3}$	[7; 9]	
4	$Q=2x^2+3e^{-x}$	[0.1; 1]	

Номер варианта	Задание	Интервал значений x	Результат
5	$Q=2+x-x^2$	[0; 1]	
6	$Q= (1-x)^4$	[0; 2]	
7	$Q=\cos(x)+x$	[1; 2]	
8	$Q=x^{1/2}+(1-x)^{2/3}$	[8; 10]	

Номер варианта	Задание	Интервал значений x	Результат
9	$Q=x^3-6x^2+9x+4$	[2; 4]	
10	$Q=2x^2-x^4$	[0.5; 1.5]	
11	$Q=x+1/\lvert x+1\rvert$	[-0.9; 1]	
12	$Q=x^3\sqrt{x-1}$	[1; 2]	

Номер варианта	Задание	Интервал значений x	Результат
13	$Q=x\times e^{x}$	[-2; 0]	
14	$Q=\ln(x/(x-20)^{3/2})$	[0; 19]	
15	$Q=\frac{x^2-3x+2}{x^2+2x-1}$	[1.3; 1.4]	
16	$Q=arctg(x)-1/2\times\ln(1+2x)$	[-0.5; 1]	

Номер варианта	Задание	Интервал значений x	Результат
17	$Q= \|x\| \times e^{-\|x-1\|}$	[-0.5; 0.5]	
18	$Q=x^2+e^{1.1x+3}$	[-3; -1]	
19	$Q=x^4-1.5\text{arctg}(x)$	[0; 1]	
20	$Q=-0.4x+e^{\|x-2\|}$	[1; 3]	

Номер варианта	Задание	Интервал значений x	Результат
21	$Q=\mathrm{tg}(x^2)+2x$	[-1; 1]	
22	$Q=2*x+\sin(x^2)$	[1; 2]	
23	$Q=-e^x\times\ln(x)$	[-3; -1]	
24	$Q=\sqrt{x+5.5}+x^4$	[-1; 1]	

Формирование матрицы. Доступ к элементам матрицы. Удаление строк и столбцов матрицы

Номер варианта	Размер матрицы, которую необходимо сформировать	Вывод элемента массива с номерами строки и столбца	Вывод строки, номер	Вывод прямоугольника		Номера строки и столбца, которые необходимо удалить из массива
				номер строки и столбца верхнего левого угла	номер строки и столбца нижнего правого угла	
1	5×12	1, 8	1	2, 1	5, 7	1, 3
2	7×13	2, 10	2	3, 2	6, 8	2, 4
3	9×14	3, 12	3	5, 3	7, 7	3, 5
4	4×15	4, 13	4	1, 4	4, 10	4, 6
5	6×16	5, 11	5	3, 5	5, 15	5, 7
6	8×12	6, 9	6	4, 6	7, 9	6, 8
7	5×13	1, 7	1	3, 7	5, 12	2, 9
8	7×14	2, 5	3	4, 8	6, 13	4, 10
9	9×15	3, 3	5	5, 9	9, 14	6, 14
10	4×16	4, 1	2	1, 1	3, 7	1, 11
11	6×12	5, 2	4	3, 2	5, 6	3, 12
12	8×13	6, 3	6	4, 3	6, 7	5, 13
13	5×14	1, 4	1	2, 4	5, 11	2, 12
14	7×15	2, 5	7	3, 5	5, 8	4, 15
15	9×16	3, 6	2	4, 6	7, 14	9, 16
16	4×12	4, 7	4	1, 7	3, 11	1, 9
17	6×13	5, 8	3	3, 8	5, 12	3, 8
18	8×14	6, 9	5	2, 9	5, 14	7, 7
19	5×15	1, 10	2	3, 1	5, 5	5, 6
20	7×16	2, 11	3	4, 2	7, 7	6, 5
21	9×12	3, 12	9	1, 3	8, 7	8, 4
22	4×13	4, 13	3	2, 4	4, 7	3, 3
23	6×14	5, 14	5	3, 5	5, 10	5, 2
24	8×15	6, 15	8	4, 6	8, 9	4, 1

Объединение массивов

Номер варианта	Размер матрицы, которую необходимо сформировать с помощью вертикальной конкатенации	Размер матрицы, которую необходимо сформировать с помощью горизонтальной конкатенации
1	3×3	3×12
2	3×4	3×11
3	3×5	3×10
4	3×6	3×9
5	3×7	3×8
6	3×8	3×7
7	3×9	4×12
8	3×10	4×11
9	3×11	4×10
10	3×12	4×9
11	3×13	4×8
12	3×14	5×12
13	3×15	5×11
14	3×3	5×10
15	3×4	5×9
16	3×5	5×8
17	3×6	5×7
18	3×7	6×12
19	3×8	6×11
20	3×9	6×10
21	3×10	6×9
22	3×11	6×8
23	3×12	6×7
24	3×13	6×15

Сложение и умножение матриц. Деление матриц

Номер варианта	Размер матриц для поэлементного сложения, поэлементного умножения и деления	Размеры матриц для матричного умножения	
1	3×3	7×10	10×7
2	3×4	7×9	9×7
3	3×5	7×8	8×7
4	3×6	7×6	6×7
5	4×3	7×5	5×7
6	4×4	7×4	4×7
7	4×5	7×3	3×7
8	4×6	7×2	2×7
9	5×3	6×5	5×6
10	5×4	6×4	4×6
11	5×5	6×3	3×6
12	5×6	6×2	2×6
13	6×2	5×7	3×4
14	6×3	5×4	4×5
15	6×4	5×3	3×5
16	6×5	5×2	2×5
17	6×6	4×6	6×4
18	7×2	4×5	5×4
19	7×3	4×3	3×4
20	7×4	4×2	2×4
21	7×5	3×7	7×3
22	7×6	3×6	6×3
23	5×7	3×5	5×3
24	4×8	3×4	4×3

Контрольные вопросы

1. Какие типы массивов применяются в Matlab?
2. Преобразование типов массивов.
3. Какими способами можно организовать массив? Примеры массивов разных размерностей.
4. Доступ к элементам вектора.
5. Создание вектора, элементы которого – члены ряда Xi+1 = = Xi+h, где значение h постоянно.

6. Создание массива, элементы которого – члены других массивов и математические выражения. Привести примеры.

7. Представление матрицы, как вектора, собранного из столбцов матрицы.

8. Функции для определения размеров массивов.

9. Удаление строк и столбцов из массивов.

10. Объединение массивов.

11. Сложение и вычитание массивов.

12. Умножение массивов на скаляр, поэлементное умножение и возведение в степень массивов.

13. Матричное умножение и возведение в степень массивов.

14. Деление массивов.

15. Использование встроенных функций Matlab при работе с массивами.

16. Способы формирования вектора.

17. Просмотр элементов вектора.

18. Определение длины вектора.

19. Вычисление математического выражения.

20. Формирование матрицы.

21. Доступ к элементам матрицы.

22. Удаление строк и столбцов матрицы.

23. Объединение массивов.

24. Сложение матриц.

25. Умножение матриц.

26. Деление матриц.

Лабораторная работа № 3
ВВОД И ВЫВОД ДАННЫХ В MATLAB

Цель работы: получение практических навыков, необходимых для организации ввода и вывода данных с помощью Matlab.

3.1. Ввод и вывод данных в окне «Command Window»

Функция **disp** выводит текст или содержимое массивов в окно «**Command Window**». Например, в рабочей области Matlab содержится вектор **vector**, состоящий из шести элементов, тогда при заданном формате вывода численных данных «bank» в ответ на функцию

disp(vector)

в командном окне будет напечатано: «2.00 2.20 2.40 2.60 2.80 3.00».

Чтобы напечатать строку, функцию **disp** необходимо применить в форме

disp(‘Вектор’),

в командном окне будет напечатана строка: «Вектор».

С помощью функции **input** можно ввести данные в рабочую область Matlab. Приведем пример:

x=input('Ввести матрицу')

Когда в программе встречается такая функция, в окно «Command Window» выводится сообщение «Ввести матрицу», после чего воспроизведение программы останавливается до того момента, пока оператор не введет необходимые данные и не нажмет клавишу **Enter**. В результате переменной **x** будет присвоено значение, введенное оператором в окне «Command Window». Например, пусть оператор после сообщения «Ввести матрицу» ввел строку [2.00 2.20 2.40;2.60 2.80 3.00], после чего нажал клавишу **Enter**. Теперь переменная **x** имеет значение:

$$\mathbf{X} = \begin{vmatrix} 2 & 2.2 & 2.4 \\ 2.6 & 2.8 & 3 \end{vmatrix}.$$

Иногда в работе программы полезно установить паузы. Такую возможность дает функция **pause**. Если применить данную функцию без аргумента, то пауза в программе продлится до нажатия оператором клавиши **Enter**. Аргумент функции **pause** задается в секундах, например следующая функция остановит воспроизведение программы на 10 с:

pause(10).

Приведем фрагмент программы, в котором оператору предписывается ввести значения напряжения и сопротивления, затем вычисляется значение тока, результат вычисления сообщается оператору, причем сообщение выводится на 6 с, затем удаляется:

```
disp('Вычисление значения тока в цепи')
U=input('Ввести значение напряжения в цепи')
disp('Напряжение в цепи равно')
disp(U)
disp('вольт')
R=input('Ввести значение сопротивления цепи')
disp('Сопротивление цепи равно')
disp(R)
disp('ом')
I=U/R; % Расчет тока
disp('Ток в цепи равен')
disp(I)
disp('ампер')
pause(6)
clc
```

Последняя команда приведенного фрагмента программы **clc** очищает окно «Command Window».

3.2. Файловый ввод и вывод данных

Язык Matlab позволяет работать с дисковыми файлами различных форматов, в т. ч. с файлами изображений, звуковыми, архивными, текстовыми, бинарными файлами, видеофайлами, файлами электронной почты и др.

Команды (функции) **save** и **load** предназначены для записи данных из рабочей области в дисковый файл и загрузки данных из дискового файла в рабочую область.

Рассмотрим варианты применения команды (функции) **save**.

Все содержимое рабочей области можно записать в дисковый файл с помощью команды

save file

или то же в форме функции

save('file'),

где **file** – имя дискового файла, в котором будут сохранены данные. По умолчанию система Matlab допишет к имени файла расширение **mat**. Формат полученного таким образом двоичного файла предусматривает сохранение идентификаторов и типов, записанных в нем переменных.

Например, пусть рабочая область содержит следующие массивы матрицу matrix, вектор **vector,** скаляр **mk**, строку **ms**:

$$\textbf{matrix} = \begin{vmatrix} 1 & 1.2 & 1.4 & 1.6 & 1.8 & 2 \\ 2 & 2.2 & 2.4 & 2.6 & 2.8 & 3 \\ 4 & 3.8 & 3.6 & 3.4 & 3.2 & 3 \end{vmatrix};$$

vector = [2, 2.2, 2.4, 2.6, 2.8, 3];

mk=12;

ms=’String1’.

Команда **save file_all** или функция **save(‘file_all’)** создаст файл **file_all.mat** и запишет в него после служебной информации:

– идентификаторы **matrix**, **vector**, **mk** и **ms**;

– данные о типах всех данных (**double**, **double**, **double** и **char**);

– размерности всех массивов (**3×6**, **1×6**, **1×1** и **1×7**);

– все массивы.

Если файл «file_all.mat» уже существует, то все его содержимое файла будет заменено. Для того чтобы новые записи были добавлены, а уже существующие сохранены, данную команду или функцию следует применять с ключом «**-append**»:

save file_all -append

или

save ('file_all','-append')

Для сохранения в дисковом файле определенных массивов данных, например **matrix** и **vector**, необходимо указать идентификаторы сохраняемых массивов:

save file matrix vector

или

save ('file','matrix','vector').

Возможно применение маски в начале, середине или конце идентификатора массива, например для массивов **matrix**, **mk** и **ms**:

save file_m m*

или

save ('file_m ','m').

Вместо команды (функции) **save** можно воспользоваться диалоговым окном, предназначенным для сохранения всех переменных из

рабочей области в двоичном файле в формате Matlab. Такое окно открывается функцией **uisave**.

Для загрузки данных из двоичного дискового файла в рабочую область служит команда (функция) **load**:

load file

или то же в форме функции

load('file'),

в рабочую область будет переписано все содержимое файла **file_all.mat**.

Формат данной команды (функции) предусматривает возможность загрузки из дискового файла только перечисленных массивов, а также применение маски в идентификаторах, аналогично формату команды (функции) **save.**

Графическое диалоговое окно для выбора двоичного файла и последующей загрузки всех данных, содержащихся в выбранном файле, в рабочую область Matlab можно открыть с помощью функции **uiload**.

Для создания текстового файла с помощью команды (функции) служит ключ «**-ascii**». Создадим текстовый файл **filem.txt**, содержащий массив **matrix**:

save filem.txt matrix -ascii

или

save('filem.txt','matrix','-ascii').

Полученный текстовый файл содержит только матрицу данных размерностью 3×6, каждый элемент массива имеет длину восемь десятичных разрядов, разделителями между элементами в строках служат пробелы, отделяют одну строку от другой символы перевода каретки, в текстовом файле не сохраняется идентификатор массива. Содержимое такого файла можно просматривать и редактировать различными текстовыми редакторами Windows.

С помощью ключа «**-double**» получается тот же результат, но длина элементов массива шестнадцать десятичных разрядов:

save filem.txt matrix –ascii –double

или

save('filem.txt','matrix','–ascii','–double')

Ключ «**-tabs**» позволяет получить текстовый файл с разделителем между элементами в строках в виде знака табуляции:

save filem.txt matrix –ascii –tabs

или

save('filem.txt','matrix','–ascii','–tabs')

Чтобы получить матрицу **m** в рабочей области Matlab из текстового файла **file_all.mat**, напишем:

m=load('filem.txt').

Обмен данными с электронными таблицами Microsoft Excel

Встроенные функции Matlab позволяют прочитать данные из электронных таблиц Microsoft Excel, а также сохранить данные, например результаты вычислений, в файле формата Microsoft Excel.

Пусть в текущем каталоге существует файл электронной таблицы Microsoft Excel под именем «*Коэффициент пропускания по суткам.xls*». С помощью функции **xlsfinfo** можно определить, является ли данный файл электронной таблицей Microsoft Excel:

info=xlsfinfo('Коэффициент пропускания по суткам')

В **info** будет возвращена строка «Microsoft Excel Spreadsheet».

Можно выяснить и имена листов электронной книги, для списка имен листов введем еще один выходной параметр функции **xlsfinfo**:

[**info list]=xlsfinfo**('Коэффициент пропускания по суткам')

Массив **list** – список всех листов книги «Коэффициент пропускания по суткам.xls», содержащих числовые данные: 'частота1', 'коэф пропускания', 'частота', 'зависимости от тока'

При применении функции **xlsread** с единственным параметром – именем файла (расширение файла «**xls**» можно не указывать):

data=xlsread('Коэффициент пропускания по суткам')

В массив **data** будут записаны все числовые данные, находящиеся на первой страницы электронной книги. При этом нечисловые данные, расположенные в начале листа, будут пропущены, а остальные нечисловые данные будут заменены в массиве **data** системной константой **NaN**, указывающей на нечисловой характер данных.

С ключом "-1":

data=xlsread('Коэффициент пропускания по суткам', -1)

Функция «**xlsread**» открывает указанный файл в приложении Microsoft Excel, после выбора оператором необходимого листа и диапазона ячеек с помощью левой кнопки мыши приложение можно закрыть, в массив **data** будут записаны выбранные оператором данные.

Формат функции «**xlsread**» позволяет указать номер или имя листа для чтения данных

data=xlsread('Коэффициент пропускания по суткам',1)

или

data=xlsread('Коэффициент пропускания по суткам','частота1')

Для чтения данных определенного диапазона с листа электронной таблицы необходимо указать начальную и конечную ячейки:

data = xlsread('Коэффициент пропускания по суткам',1,'A1:A10')

С помощью данной функции возможно прочитать также и нечисловые данные, которые будут возвращены функцией во второй выходной параметр, например:

[data,txt]=xlsread('Коэффициент пропускания по суткам',1).

Массив **txt** – массив ячеек, его размер так же, как размер обычного массива можно определить с помощью функции **size**:

N=size(txt)

N – двумерный массив ячеек, адрес каждой ячейки состоит из номера строки и номера столбца, которые заключаются в фигурные скобки, например ячейка **txt{1,1}** содержит строку «*Значения коэффициента пропускания в диапазоне 350–800 нм, спектральной плотности и потока ФАР в зависимости от времени выращивания растений*», ячейка **txt{1,2}** – пустую строку и так далее.

Функция «**xlsread**» имеет и другие возможности, с которыми при необходимости можно познакомиться, используя справку пакета Matlab.

Функция «**xlswrite**» предназначена для сохранения данных в файле электронной таблицы Microsoft Excel.

Пусть в рабочей области Matlab содержится массив **Matrix**:

Matrix=

1	140.28	252.36	96.67	8.37
2	142.02	262.54	100.07	9.07
3	149.90	285.70	96.78	9.35
4	147.12	277.52	101.30	9.67
5	163.62	307.95	100.35	9.45
6	173.40	322.44	104.8	10.12
7	178.86	334.88	106.17	10.35
8	186.26	350.11	109.20	11.03
9	183.53	346.10	104.48	10.38
10	198.76	374.91	106.88	12.15
11	205.30	378.49	113.14	12.98
12	206.77	397.48	112.98	11.34
13	198.42	378.39	109.07	10.95
14	216.48	393.44	114.45	12.89
15	221.45	403.84	115.23	13.71

Сохраним данную матрицу в файле электронной таблицы:

xlswrite('Primer', Matrix)

Если файл **Primer.xls** не существует в текущем каталоге, то он будет создан, и на его первом листе будут располагаться данные, совпадающие с данными матрицы **Matrix**, начиная с ячейки **A1.**

Можно указать номер лист, на котором необходимо сохранить данные:

xlswrite('Primer', Matrix,2)

Если вместо номера листа в качестве второго параметра функции ввести его символьное имя, то в случае отсутствия листа с таким именем он будет вставлен в электронную книгу и на него будут помещены необходимые данные, например:

xlswrite('Primer', Matrix, 'Результаты эксперимента')

Для того чтобы расположить данные в определенных ячейках, необходимо указать диапазон ячеек или адрес верхней левой ячейки, например:

xlswrite('Primer', Matrix, 'Результаты эксперимента', 'A3')

или

xlswrite('Primer', Matrix, 'Результаты эксперимента', 'A3:E17')

Теперь на листе электронной книги есть место для надписей. Первый столбец – это номера проведенных экспериментов, второй – выход реакции, третий – количество вещества, четвертый – температура, пятый – количество катализатора. Перед сохранением текстовых данных в электронной книге создадим в рабочей области Matlab массив ячеек, содержащий необходимые данные:

txt{1,1}='№ эксперимента';
txt{2,1}='i';
txt{1,2}='Выход реакции';
txt{2,2}='y';
txt{1,3}='Количество вещества';
txt{2,3}='x1';
txt{1,4}='Температура';
txt{2,4}='x2';
txt{1,5}='Количество катализатора';
txt{2,5}='x3';

Получился массив ячеек

txt= [{№ эксперимента} {Выход реакции} {Количество вещества} {Температура} {Количество катализатора}]

{i} {y} {x1} {x2} {x3}

На листе электронной таблицы создадим надписи:

xlswrite('Primer', txt, 'Результаты эксперимента','A1')

Графический ввод данных. Всплывающее меню

Для ввода целого числа, имеющего значение больше нуля, в рабочую область Matlab можно воспользоваться функцией **menu**, ее синтаксис: **Choice=menu('Заголовок меню', 'Кнопка №1', 'Кнопка №2',… 'Кнопка №i'… 'Кнопка №N')**. Первый параметр – это строка заголовка меню, остальные параметры – надписи на кнопках меню. Данная функция работает следующим образом: на экране монитора появляется всплывающее меню, заголовок которого – первый параметр функции, число кнопок на единицу меньше количества параметров функции, на каждой кнопке расположена соответствующая надпись. Окно является модальным, т. е. с его появлением воспроизведение программы останавливается до нажатия оператором одной из кнопок. После нажатия кнопки окно с меню закрывается, а выходной параметр **Choice** приобретает значение, равное номеру выбранной кнопки.

Пусть необходимо выбрать вариант задания. Создадим меню (применим перенос командной строки или строки сценария с помощью знака «…»):

Choice=menu('Выбор вариантов',…

'Самый сложный №1','Сложный №2','Средний №3',…

'Слабый №4','Самый слабый №5','Не выбирать')

Появившееся меню будет иметь заголовок «*Выбор вариантов*» и шесть кнопок для выбора, надписи на кнопках: «*Самый сложный №1*», «*Сложный №2*», «*Средний №3*», «*Слабый №4*», «*Самый слабый №5*», «*Не выбирать*». Если выбрать щелчком левой кнопки мыши вторую кнопку с надписью: «*Сложный №2*», окно меню будет закрыто, **Choice=2,** при выборе кнопки с надписью: «*Не выбирать*» после закрытия окна меню значение **Choice** будет равно 6.

Диалоговое окно для ввода данных

Для ввода данных оператором с использованием клавиатуры компьютера можно применить функцию **inputdlg**. Например, для ввода любого числа или нескольких чисел достаточно организовать диалоговый ввод следующим образом:

DataInput=inputdlg('To enter number')

Данный диалог также является модальным, т. е. воспроизведение программы останавливается до нажатия одной из кнопок диалогового окна: **Ok** или **Cancel**. В окне есть сообщение, текст которого «To enter number», и окно редактора, предназначенное для ввода данных с клавиатуры. Если необходимо ввести численные данные, то оператор должен придерживаться правил ввода массивов типа **double**. После нажатия кнопки **Ok** массив **DataInput** будет содержать массив символов из окна редактора. Затем с помощью функции преобразования можно получить массив данных типа **double**. Необходимо учесть, что придется иметь дело с массивом ячеек, поэтому для адресации ячеек следует применять фигурные скобки. Пусть оператор ввел строку: **12 22; 17.1, 18.2**. Теперь **DataInput=' 12 22; 17.1, 18.2'**. Преобразуем данную строку в массив чисел:

Data=str2num(DataInput{1})

Полученный массив:

$$\mathbf{Data}=\begin{bmatrix}12.0 & 22.0\\ 17.1 & 18.2\end{bmatrix}$$

К диалоговому окну можно добавить заголовок, например, «*Ввод данных*»:

DataInput=inputdlg('To enter number', 'Ввод данных')

Следующий параметр функции – количество строк в окне редактора, по умолчанию строка одна, но можно задать любое целое число строк, например две:

DataInput=inputdlg('To enter number', 'Ввод данных', 2)

Если задается строка данных по умолчанию при появлении окна, эти данные будут записаны автоматически в окне редактора, оператор может заменить их любыми другими (количество строк в редакторе – одна):

DataInput=inputdlg('To enter number', 'Ввод данных',1,…

{ '12.1 22.2'})

В одном диалоговом окне можно получить любое количество редакторов для ввода данных, например три редактора. Создадим диалоговое окно для ввода параметров проводников электрической

цепи, заголовок окна «*Параметры RCL*», сообщения для первого редактора – «*R*», для второго – «*C*», для третьего «*L*», каждый редактор имеет одну строку, значения по умолчанию $R=1$, $C=10^6$, $L=10^3$:

DataInput=inputdlg({'R','C','L'},' Параметры RCL ',1,…

{'1','10^-6', '10^-3'})

Для преобразования данных из второго редактора необходимо обратиться ко второй ячейке массива **DataInput**:

C= str2num(DataInput{2})

Чтобы получить численные значения из третьего редактора:

L = str2num(DataInput{3})

Диалоговое окно для выбора из списка

Функция **listdlg** создает окно со списком из заданных строк и кнопками для выбора **Ok** и отмены **Cancel**. Строки списка задаются массивом ячеек. Возможен выбор нескольких элементов списка. Выходной аргумент: **Selection** – номер или вектор с номерами выбранных строк. Например, создадим окно для выбора функции из списка:

Selection=…

listdlg(**'ListString'**,{'Линейная';'Квадратичная';'Кубическая'})

Если пользователь выбрал строку «*Квадратичная*» и нажал кнопку «**Ok**», то окно закрывается, выходной параметр **Selection** приобретает значение, равное **2**. Если выбрано «*Линейная*» и «*Кубическая*» и нажата кнопка «**Ok**», то **Selection =** [1 3].

3.3. Графический вывод данных

Matlab имеет возможности построения графиков функций и визуализации данных:

1. Высокоуровневые графические функции (**plot**, l**oglogx**, **semilogx**, **bar**, **hist**, **stairs**, **errorbar**, **stem**, **polar**, **rose**, **compass**, **feather**, **contour**, **surf**, **surfc**, **surfl**, **mesh**, **meshc**, **quiver**, **plot3**, **contour3**, **slice**, **waterfal** и др.).

2. Интерактивную среду **Plotting Tools**, компонента которой **Plot Editor**, также позволяет изменять свойства элементов графика.

3. Специализированные функции и средства **ToolBox** для графического отображения характеристик исследуемых объектов и результатов.

Этого набора оказывается недостаточно, если приложение должно осуществить вывод графических результатов в готовом виде, не предполагающем их дальнейшую правку в редакторе графиков или в ходе работы управлять элементами графиков: удалять поверхности,

изменять цвет и толщину линий, добавлять стрелки и поясняющие надписи и т. д. В этих случаях использование дескрипторной графики и низкоуровневых графических функций дает возможность полного контроля над элементами графиков.

Построение графиков отрезками прямых

Функция **plot(X,Y)** строит график функции **У(X)**, координаты точек **(x,y)** которой берутся из векторов одинакового размера **Y** и **X.** Если **X** или **Y** – матрица, то строится семейство графиков по данным, содержащимся в колонках матрицы. Например:

X=[0 1 2 3 4 5];

Y=[**sin(X)./cos(X)**];

plot(X,Y)

Будет построен график, состоящий из отрезков прямой, соединяющих точки, абсциссы которых являются элементами вектора **X**, ординаты – элементами вектора **Y**.

Функция **plot(Y)** строит график **Y(i)**, где значения ординат **y** берутся из вектора **Y**, а значения абсцисс **i** представляют собой индекс соответствующего элемента вектора **Y**. Если **Y** содержит комплексные элементы, то выполняется функция **plot(real(Y), imag(Y))**, т. е. по оси абсцисс будут откладываться действительные части элементов вектора **Y**, по оси ординат – коэффициенты при мнимой единице соответствующих элементов вектора **Y**. Пример такого графика:

```
x= -2*pi:0.02*pi:2*pi;
y=sin(x)+i*cos(3*x);
plot(y)
```

Во всех других случаях мнимая часть данных игнорируется, таким образом, график **plot(x,y)** в данном примере будет выглядеть иначе, чем график **plot(y)**.

Данная функция позволяет управлять цветом и стилем линий графика. Функция **plot(X,Y,S)** аналогична функции **plot(X.Y)**, но тип линии графика можно задавать с помощью строковой константы **S.** Например, нарисуем график штрихпунктирной линией, для этого введем в строку **S** символы «**-.**», состоящей из окружностей (символ «**O**»), цвет линии – желтый (символ «**Y**»):

```
S = '-.OY'
plot(X,Y,S)
```

или

```
plot(X,Y,'-.OY')
```

Возможно задание следующих *стилей линии графика*:

сплошная – символ «**-**»;

двойной пунктир – символ «**:**»;

штрих-пунктир – символы «**-.**»;

штриховая – символы «**--**».

Линию графика можно построить с помощью следующих элементов:

точка – символ «**.**»;

окружность – символ «**o**»;

крест – символ «**x**»;

плюс – символ «**+**»;

звездочка – символ «*****»;

квадрат – символ «s»;

ромб – символ «d»;

треугольник (вниз) – символ «v»;

треугольник (вверх) – символ «^»;

треугольник (влево) – символ «<»;

треугольник (вправо) – символ «>»;

пятиугольник – символ «**p**»;

шестиугольник – символ «**h**».

Цвет линии графика:

желтый – символ «**y**»;

голубой – символ «**b**»;

зеленый – символ «**g**»;

фиолетовый – символ «**g**»;

красный – символ «**r**»;

синий – символ «**b**»;

черный – символ «**k**».

Таким образом, с помощью строковой константы **S** можно изменять цвет линии, представлять узловые точки различными отметками (точка, окружность, крест, треугольник с разной ориентацией вершины и т. д.) и менять тип линии графика.

Функция **plot(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...)** строит на одном графике ряд линий, представленных данными вида **(Xi,Yi,Si)**, где **Xi** и **Yi** – векторы или матрицы, а **Si** – строки.

Для примера создадим вектор:

x=-2*pi:0.1*pi:2*pi;

Зададим три функции:

yl=sin(x);

y2=sin(x).^2;
y3=sin(x).^3;

Построим три графика полученных функций. В данном случае функция **plot** будет выглядеть как на рисунке 3.1.

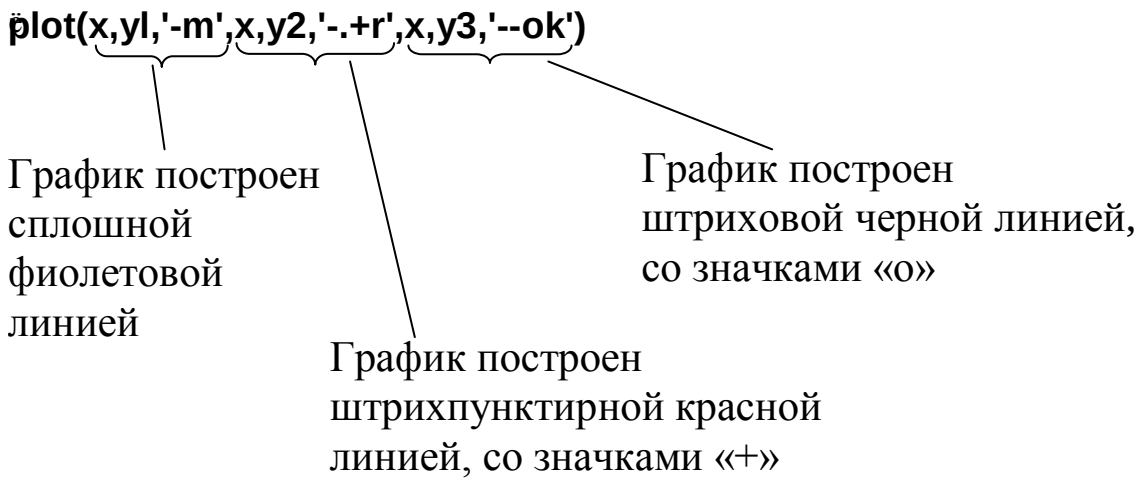


Рисунок 3.1 – Функция plot

При отсутствии указания на цвет линий и точек он выбирается автоматически из таблицы цветов (белый исключается). Если линий больше шести, то выбор цветов повторяется.

Возможно также применять функцию **plot** в формате:

x = 0:0.2:10;
y = cos(x);
plot(x, y, 'LineWidth', 2, 'Marker', 'o', 'MarkerSize', 10)

В данном примере при построении графика функцией **plot** заданы свойства линии:

LineWidth – толщина линии 2 пт.;
Marker – тип маркера (кружок);
MarkerSize – размер маркера 10 пт.

Создание массивов данных для трехмерных графиков

Трехмерные поверхности описываются функцией двух переменных **z(x,y)**. Функция **[X,Y] = meshgrid(x,y)** преобразует область, заданную векторами **x** и **y**, в массивы **X** и **Y**, которые могут быть использованы для вычисления функции двух переменных и построения трехмерных графиков. Строки выходного массива **X** являются копиями вектора **x**; а столбцы **Y** – копиями вектора **y**., например:

x=[1,2,3,4]
y=[13,14,15,16,17]
[X,Y]=meshgrid(x,y)

Результат:

$$X=\begin{bmatrix}1 & 2 & 3 & 4\\ 1 & 2 & 3 & 4\\ 1 & 2 & 3 & 4\\ 1 & 2 & 3 & 4\\ 1 & 2 & 3 & 4\end{bmatrix};\quad Y=\begin{bmatrix}13 & 13 & 13 & 13\\ 14 & 14 & 14 & 14\\ 15 & 15 & 15 & 15\\ 16 & 16 & 16 & 16\\ 17 & 17 & 17 & 17\end{bmatrix}$$

Другой пример:

[X,Y] = meshgrid(-2:0.2:2,-2:0.2:2);

Такой вызов функции позволяет задать опорную плоскость для построения трехмерной поверхности при изменении **X** и **Y** от **-2** до **2** с шагом **0.2**.

Функция **[X,Y,Z] = meshgrid(x,y,z)** возвращает трехмерные массивы для вычисления функций трех переменных и построения четырехмерных графиков.

Построение трехмерных графиков поверхностей

Вычислим функцию:

Z=X.^2+Y.^2

Следующие функции построят трехмерный график данной функции:

plot3(X,Y,Z)

surf(X,Y,Z)

mesh(X,Y,Z)

Существуют и другие функции для построения трехмерных графиков.

3.4. Ввод и вывод данных в Simulink

*Блоки библиотеки **Source***

При работе с моделями Simulink также возникает необходимость работать с внешними данными. Рассмотрим блоки Simulink, предназначенные для ввода и вывода данных.

Блок считывания данных из файла «**From File**», имеющий пиктограмму 

, позволяет получать данные из внешнего файла. Параметры блока **From File**:

File Name – имя файла с данными;

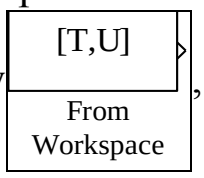
Sample time – шаг изменения выходного сигнала блока.

Структура данных в **mat**-файле – многомерный массив с переменным числом строк, которое определяется числом регистрируемых переменных. Элементы первой строки содержат последовательные значения модельного времени от **1** до **m**, элементы в других строках – соответствующие им значения переменных.

$$\begin{bmatrix} t_1 & t_2 & \dots & t_i & \dots & t_{m-1} & t_m \\ u1_1 & u1_2 & \dots & u1_i & \dots & u1_{m-1} & u1_m \\ u2_1 & u2_2 & \dots & u2_i & \dots & u2_{m-1} & u2_m \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ uj_1 & uj_2 & \dots & uj_i & \dots & uj_{m-1} & uj_m \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ un\text{-}1_1 & un\text{-}1_2 & \dots & un\text{-}1_i & \dots & un\text{-}1_{m-1} & un\text{-}1_m \\ un_1 & un_2 & \dots & un_i & \dots & un_{m-1} & u1_m \end{bmatrix}$$

Матрица должна состоять как минимум из двух строк. Значения времени записаны в первой строке матрицы, а в остальных строках находятся значения сигналов, соответствующие данным моментам времени. Значения времени должны быть записаны в возрастающем порядке. Выходной сигнал блока содержит только значения сигналов, а значения времени в нем отсутствуют. Если шаг расчета текущей модели не совпадает с отсчетами времени в файле данных, то Simulink выполняет линейную интерполяцию данных.

Файл данных – **mat**-файл, из которого считываются значения, не является текстовым. Пользователям Simulink удобнее всего создавать **mat**-файл с помощью блока «**To File**» (библиотека **Sinks**).

Блок считывания данных из рабочего пространства «**From Workspace**», имеющий пиктограмму 

, предназначен для получения данных из рабочей области Matlab.

Параметры блока **From Workspace**:

Data – имя переменной (матрицы или структуры), содержащей считываемые блоком данные;

Sample time – шаг изменения выходного сигнала блока;

Interpolate data – интерполяция данных для значений модельного времени, не совпадающих со значениями в переменной **Data.**

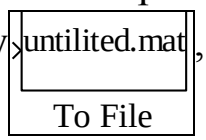
Form output after final data value by – вид выходного сигнала по окончании значений времени в переменной **Data**: **Extrapolate** – линейная экстраполяция сигналов; **SettingToZero** – нулевые значения сигналов; **HoldingFinalValue** – выходные значения сигналов равны последним значениям; **CyclicRepetition** – циклическое повторение значений сигналов. Данный вариант может использоваться, если переменная **Data** имеет формат **Structure without time**.

На пиктограмме блока обозначены:

T – вектор значений модельного времени tout (только целочисленные значения).

U – векторы значений величин, вводимых из рабочей области, длина каждого из векторов равна длине вектора **T**. Элементы векторов могут задаваться числовыми константами, переменными, вычисляемыми выражениями, а также можно задать имя вектора из рабочей области

Блоки библиотеки ***Sink***

Блок сохранения данных в файле «**To File**», имеющий пиктограмму 

, записывает данные, поступающие на его вход, в дисковый файл.

Параметры блока:

Filename – имя файла для записи. По умолчанию файл имеет имя **untitled.mat**. Если не указан полный путь файла, то файл сохраняется в текущей рабочей папке.

Variable name – имя переменной, содержащей записываемые данные.

Decimation – кратность записи в файл входного сигнала. При **Decimation** = 1 записывается каждое значение входного сигнала, при **Decimation** = 2 записывается каждое второе значение, при **Decimation** = 3 – каждое третье значение и так далее.

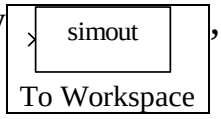
Sample time – шаг модельного времени. Определяет дискретность записи данных.

Данные в файле сохраняются в виде матрицы:

$$\begin{bmatrix} t_1 & t_2 & \dots & t_i & \dots & t_{m-1} & t_m \\ u1_1 & u1_2 & \dots & u1_i & \dots & u1_{m-1} & u1_m \\ u2_1 & u2_2 & \dots & u2_i & \dots & u2_{m-1} & u2_m \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ uj_1 & uj_2 & \dots & uj_i & \dots & uj_{m-1} & uj_m \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ un\text{-}1_1 & un\text{-}1_2 & \dots & un\text{-}1_i & \dots & un\text{-}1_{m-1} & un\text{-}1_m \\ un_1 & un_2 & \dots & un_i & \dots & un_{m-1} & u1_m \end{bmatrix}$$

Значения времени от **1** до **m** записываются в первой строке матрицы, а в остальных строках – значения сигналов, соответствующих данным моментам времени.

Файл данных (**mat-файл**), в который записываются данные, не является текстовым.

Блок сохранения данных в рабочей области «*To Workspace*», имеющий пиктограмму 

, записывает данные, поступающие на его вход, в рабочую область Matlab.

Параметры блока **To Workspace**:

Variable name – имя переменной, содержащей записываемые данные.

Limit data points to last – максимальное количество сохраняемых расчетных точек по времени (отсчет ведется от момента завершения моделирования). В том случае, если значение параметра **Limit data points to last** задано как **inf**, то в рабочей области будут сохранены все данные.

Decimation – кратность записи данных в рабочую область.

Sample time – шаг модельного времени. Определяет дискретность записи данных.

Save format – формат сохранения данных. Может принимать значения:

Matrix – матрица. Данные сохраняются как массив, в котором число строк определяется числом расчетных точек по времени, а число столбцов – размерностью вектора подаваемого на вход блока. Если на вход подается скалярный сигнал, то матрица будет содержать лишь один столбец.

Structure – структура. Данные сохраняются в виде структуры, имеющей три поля: **time** – время, **signals** – сохраняемые значения сигналов, **blockName** – имя модели и блока **To Workspace**. Поле **time** для данного формата остается не заполненным.

Structure with Time – структура с дополнительным полем (время). Для данного формата, в отличие от предыдущего, поле «time» заполняется значениями времени.

Цифровой дисплей «Display» отображает значение сигнала в виде числа.

Параметры цифрового дисплея:

Format – формат отображения данных. Параметр Format может принимать следующие значения:

short – 5 значащих десятичных цифр;

long – 15 значащих десятичных цифр;

short_e – 5 значащих десятичных цифр и 3 символа степени десяти;

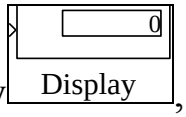
long_e – 15 значащих десятичных цифр и 3 символа степени десяти;

bank – "денежный" формат. Формат с фиксированной точкой и двумя десятичными цифрами в дробной части числа;

Decimation – кратность отображения входного сигнала. При **Decimation=1** отображается каждое значение входного сигнала, при **Decimation=2** отображается каждое второе значение, а при **Decimation=3** – каждое третье значение и т. д.

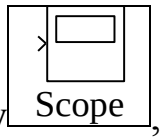
Sample time – шаг модельного времени. Определяет дискретность отображения данных.

Floating display (флажок) – перевод блока в «свободный» режим. В данном режиме входной порт блока отсутствует, а выбор сигнала для отображения выполняется щелчком левой клавиши мыши на соответствующей линии связи. В этом режиме для параметра расчета **Signal storage reuse** должно быть установлено значение **off** (вкладка **Advanced** в окне диалога **Simulation parameters**…).

Блок «Display», имеющий пиктограмму 

, может использоваться для отображения не только скалярных сигналов, но также векторных, матричных и комплексных. Если все отображаемые значения не могут поместиться в окне блока, в правом нижнем углу бло-

ка появляется символ, указывающий на необходимость увеличить размеры блока, что можно сделать с помощью левой кнопки мыши.

Осциллограф «Scope», имеющий пиктограмму 

, строит графики исследуемых сигналов в функции времени. Позволяет наблюдать за изменениями сигналов в процессе моделирования.

Для того чтобы открыть окно просмотра сигналов, необходимо выполнить двойной щелчок левой клавишей мыши на изображении блока. Это можно сделать на любом этапе расчета (как до начала расчета, так и после него, а также во время расчета). В том случае, если на вход блока поступает векторный сигнал, то кривая для каждого элемента вектора строится отдельным цветом. Настройка окна осциллографа выполняется с помощью панелей инструментов.

Панель инструментов содержит 11 кнопок:

Первая слева кнопка **Print** – печать содержимого окна осциллографа.

Вторая слева кнопка **Parameters** – доступ к окну настройки параметров.

Три расположенные рядом кнопки с изображением лупы: **Zoom** – увеличение масштаба по обеим осям; **Zoom X-axis** – увеличение масштаба по горизонтальной оси; **Zoom Y-axis** – увеличение масштаба по вертикальной оси. Если нажать соответствующую кнопку (**Zoom**, **Zoom X-axis** или **Zoom Y-axis**) и щелкнуть один раз левой клавишей мыши, в нужном месте графика произойдет 2,5-кратное увеличение масштаба. Если нажать соответствующую кнопку (**Zoom**, **Zoom X-axis** или **Zoom Y-axis**), то, нажав левую клавишу мыши, с помощью динамической рамки или отрезка можно указать область графика для увеличенного изображения.

Шестая кнопка **Autoscale** – автоматическая установка масштабов по обеим осям, на кнопке изображен бинокль.

Седьмая кнопка **Save current axes settings** – сохранение текущих настроек окна.

Восьмая кнопка **Restore saved axes settings** – установка ранее сохраненных настроек окна.

Девятая кнопка **Floating scope** – перевод осциллографа в свободный режим.

Десятая кнопка **Lock/Unlock axes selection** – закрепить/разорвать связь между текущей координатной системой окна и

отображаемым сигналом. Инструмент доступен, если включен режим **Floating scope**.

Одиннадцатая кнопка **Signal selection** – выбор сигналов для отображения. Инструмент доступен, если включен режим **Floating scope**.

Вторая кнопка **Parameters** открывает окно настройки параметров осциллографа, имеющего две закладки: **General** – общие параметры и **Data history** – параметры сохранения сигналов в рабочей области Matlab.

На вкладке **General** задаются следующие параметры:

Number of axes – число входов (систем координат) осциллографа. При изменении этого параметра на изображении блока появляются дополнительные входные порты.

Time range – величина временного интервала, для которого отображаются графики. Если время расчета модели превышает заданное параметром **Time range**, то вывод графика производится порциями, при этом интервал отображения каждой порции графика равен заданному значению **Time range.**

Tick labels – вывод/скрытие осей и меток осей. Может принимать три значения (выбираются из списка): **all** – подписи для всех осей, **none** – отсутствие всех осей и подписей к ним, **bottom axis only** – подписи горизонтальной оси только для нижнего графика.

Sampling – установка параметров вывода графиков в окне. Задает режим вывода расчетных точек на экран. При выборе **Decimation** кратность вывода устанавливается числом, задающим шаг выводимых расчетных точек, в том случае если режим вывода расчетных точек задается как **Sample time**, то его числовое значение определяет интервал квантования при отображении сигнала.

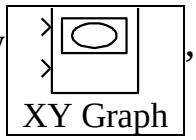
На вкладке **Data history** задаются следующие параметры:

Limit data points to last – максимальное количество отображаемых расчетных точек графика. При превышении этого числа начальная часть графика обрезается. В том случае если флажок параметра **Limit data points to last** не установлен, то Simulink автоматически увеличит значение этого параметра для отображения всех расчетных точек.

Save data to workspace – сохранение значений сигналов в рабочей области Matlab.

Variable name – имя переменной для сохранения сигналов в рабочей области Matlab.

Format – формат данных при сохранении в рабочей области Matlab. Может принимать значения: **Array** – массив, **Structure** – структура, **Structure with time** – структура с дополнительным полем **Время**.

Графопостроитель **XY Graph**, имеющий пиктограмму 

, строит график одного сигнала в функции другого (график вида Y(X)).

Параметры:

x-min – минимальное значение сигнала по оси **X**;

x-max – максимальное значение сигнала по оси **X**;

y-min – минимальное значение сигнала по оси **Y**;

y-max – максимальное значение сигнала по оси **Y**;

Sample time – шаг модельного времени.

Блок имеет два входа. Верхний вход предназначен для подачи сигнала, который является аргументом (X), нижний – для подачи значений функции (Y).

3.5. Задание для лабораторной работы

Организовать ввод элементов матрицы с помощью команды **input** и вывод матрицы с помощью команды **disp**. Пример **m**-сценария смотрите на рисунке 3.2.

На рисунке 3.3 показан результат работы программы после ввода оператором шести элементов матрицы.

Сохранить введенную матрицу в дисковом файле с помощью функции **save**, используя ключи таким образом, чтобы полученный файл имел текстовый формат.

Посмотреть содержимое файла с помощью встроенного в Windows текстового редактора, например, «Блокнот», «WordPad» и др. (рис. 3.4).

Сохранить введенную матрицу в файле формата Microsoft Excel.

Посмотреть содержимое файла с помощью Microsoft Excel (рис. 3.5).

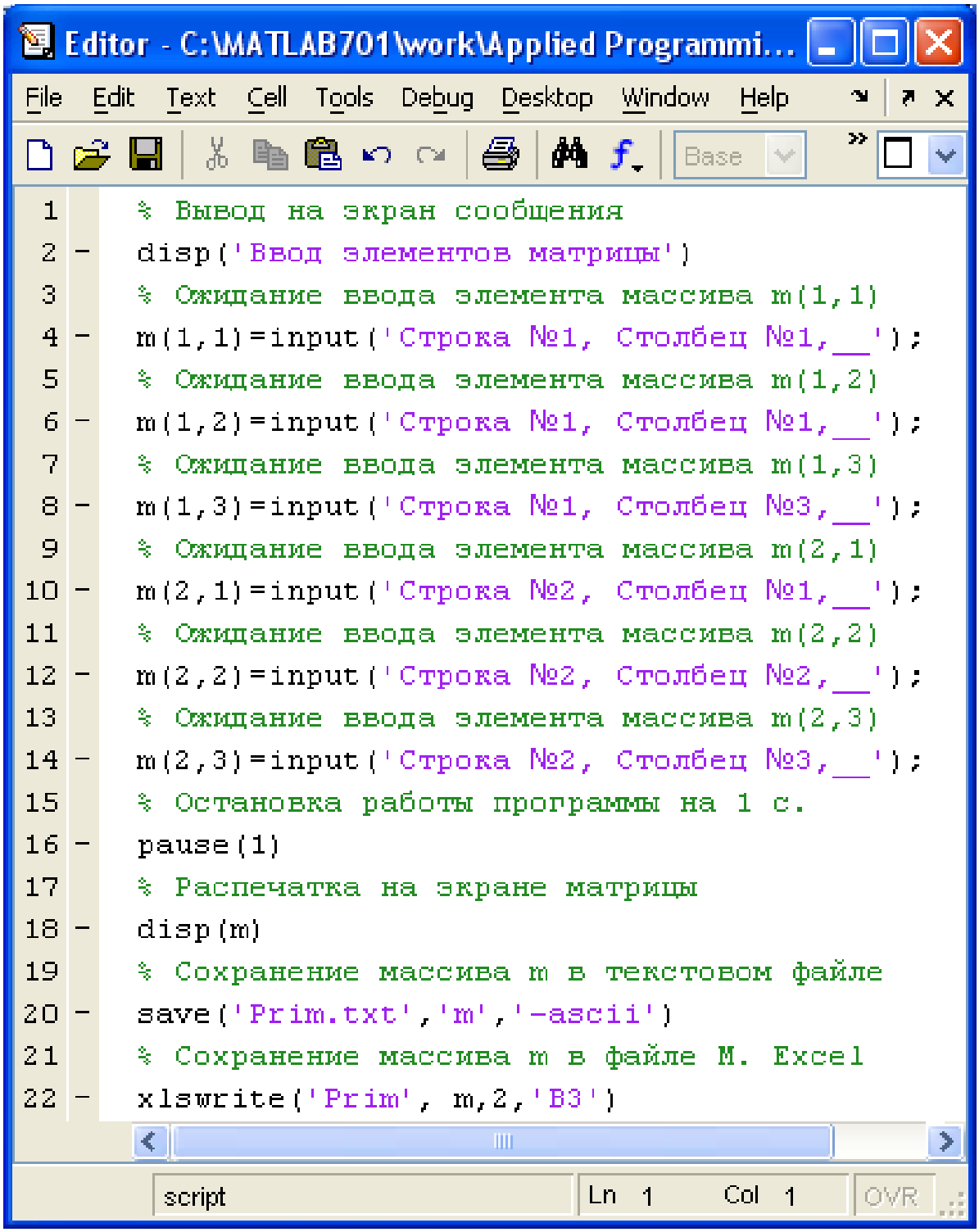


Рисунок 3.2 – Вывод матрицы с помощью команды disp

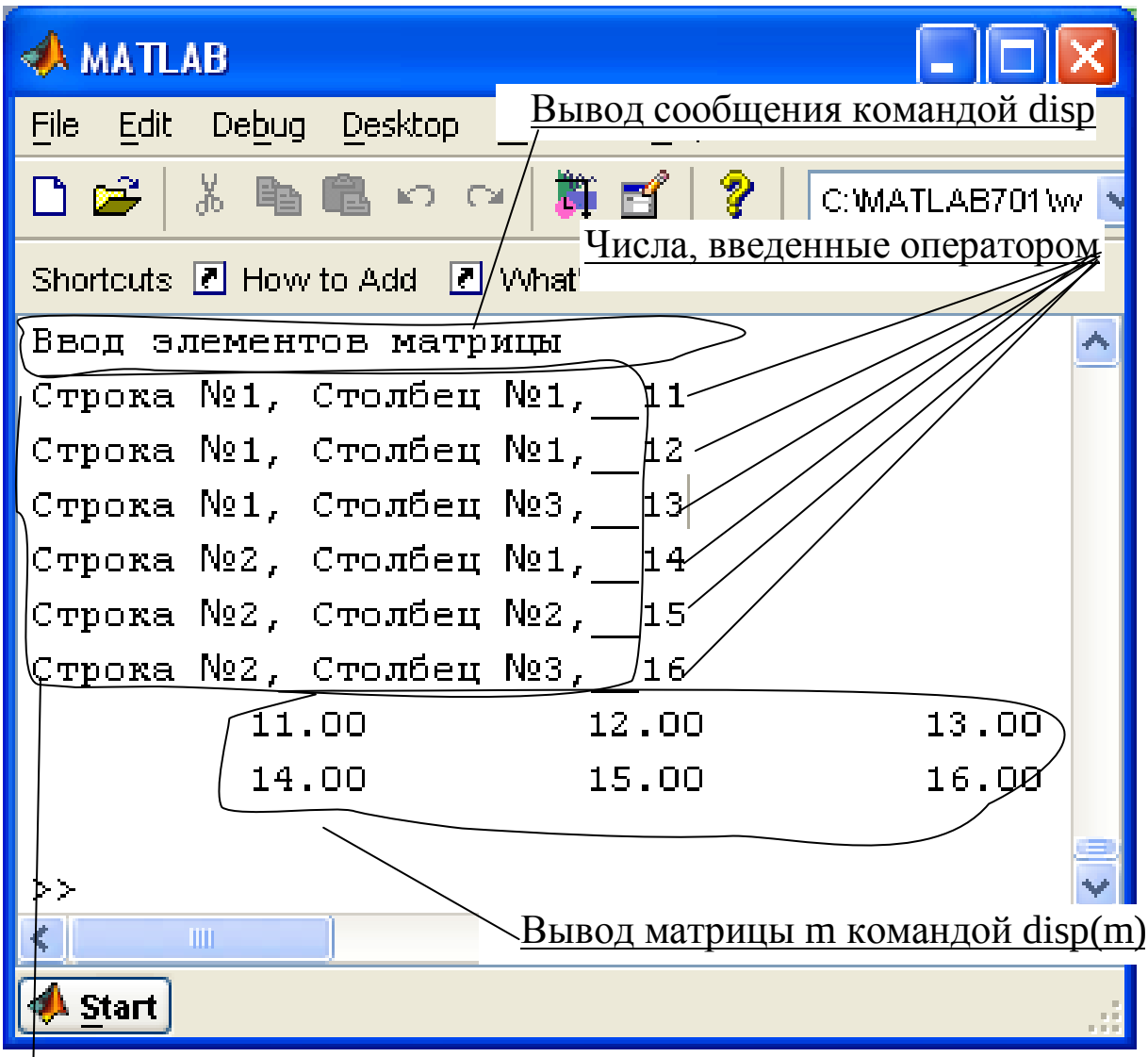


Рисунок 3.3 – Результат работы программы после ввода оператором шести элементов матрицы

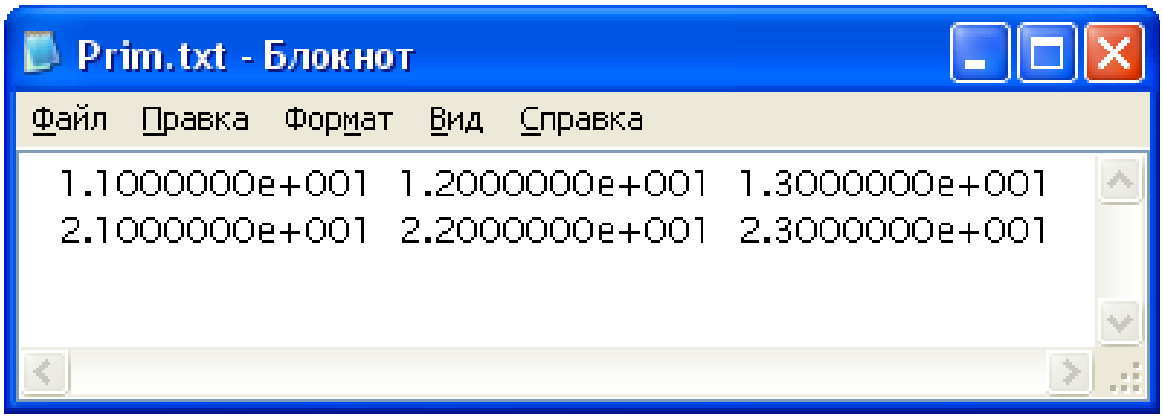


Рисунок 3.4 – Просмотр содержимого файла с помощью текстового редактора, например «Блокнот»

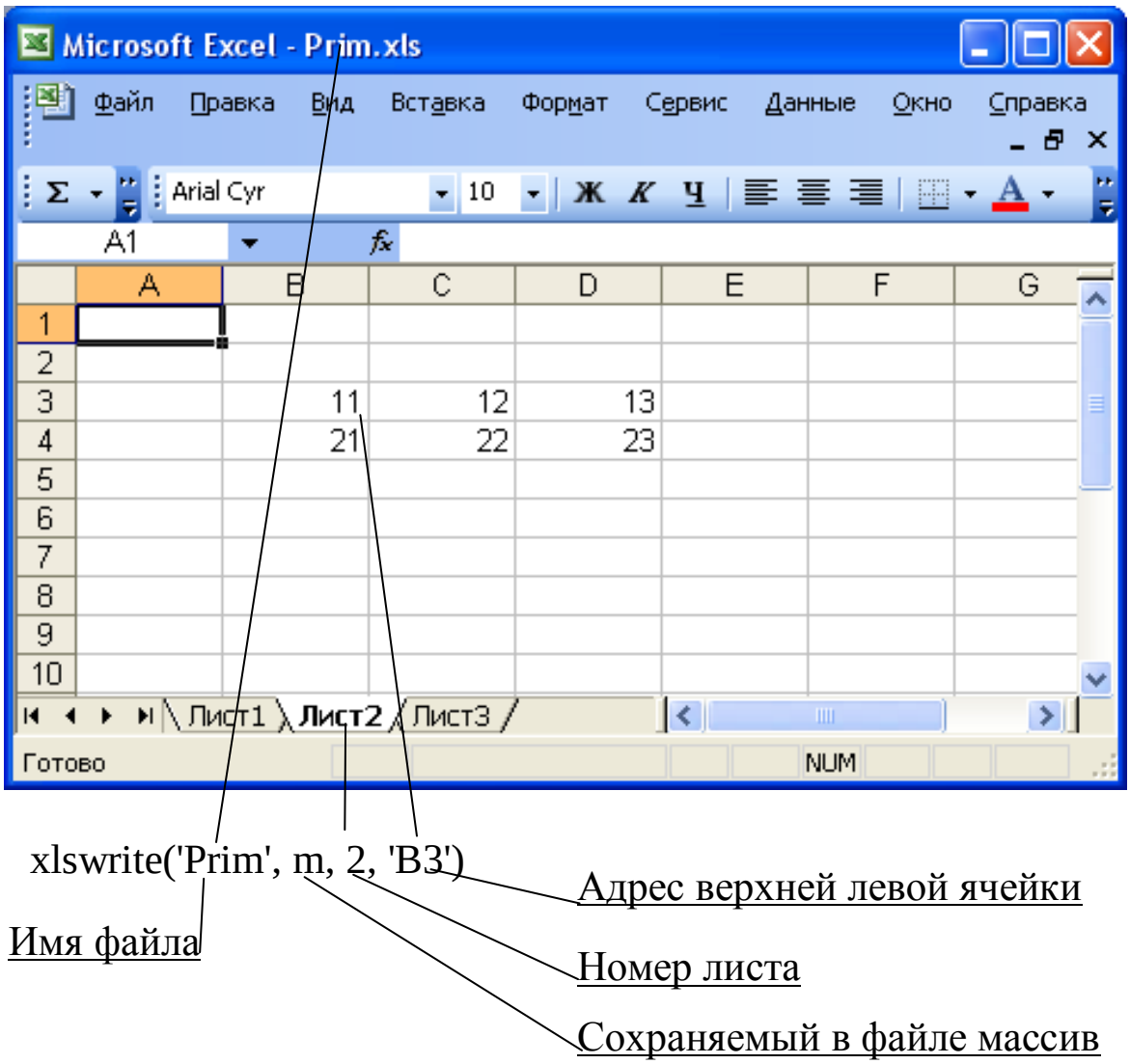
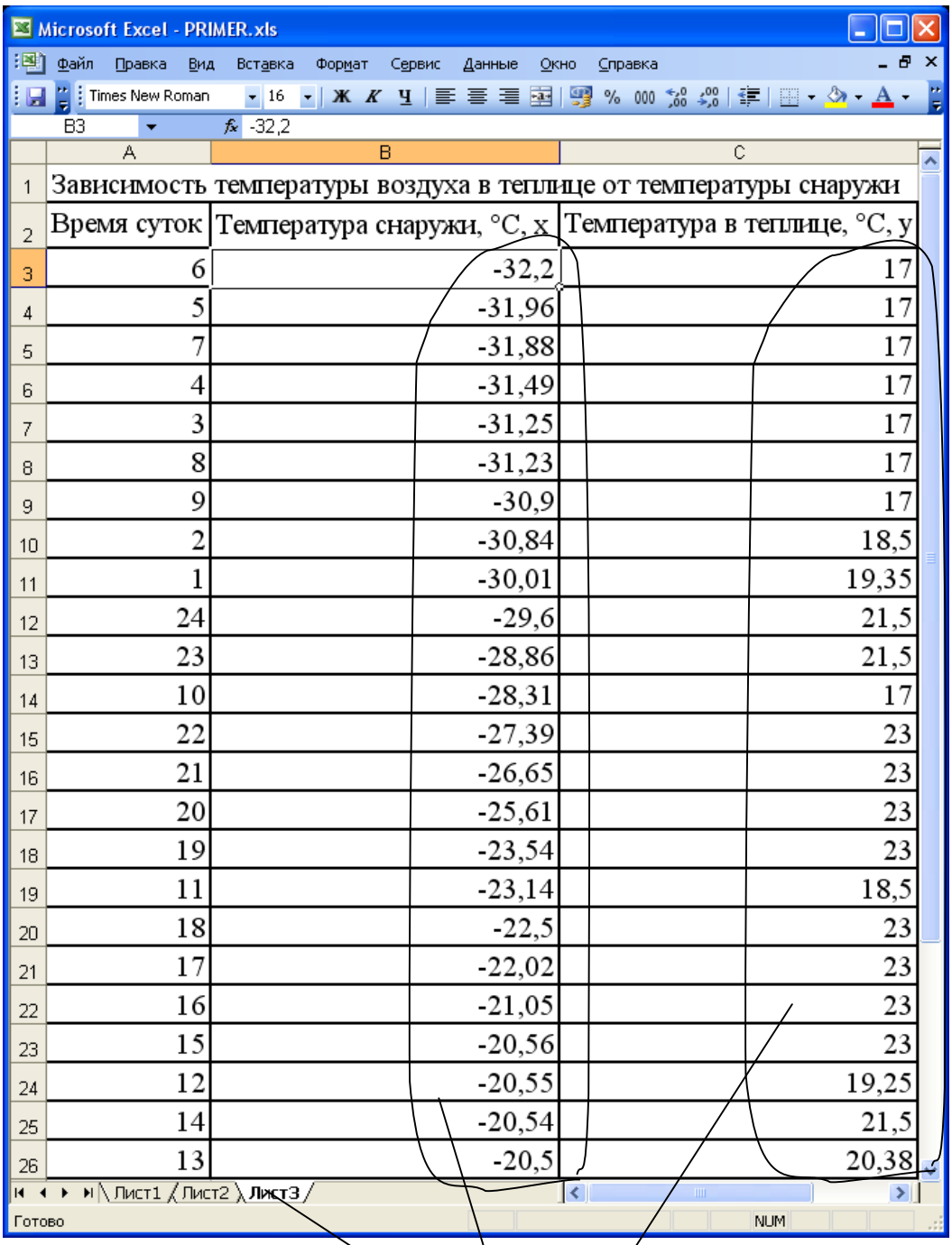


Рисунок 3.5 – Просмотр содержимого файла с помощью Microsoft Excel

Ввод данных в рабочую область Matlab из дискового файла

Необходимо ввести данные в рабочую область Matlab из дискового файла формата Microsoft Excel и текстового дискового файла. Построить графики зависимости **y** от **x** и вектора отношения **y** к **x**, который необходимо ввести из текстового файла, от **x**.

Создать файл Microsoft Excel (см. вариант задания в п. 3.6). Для примера используем файл, в котором приведены результаты наблюдений температуры воздуха в теплице и температуры снаружи в течение суток, данные отсортированы по столбцу **B** («Температура снаружи, °C, x»). На рисунке 3.6 приведен вид листа Microsoft Excel, содержащего данные и функции для чтения данных в массивы рабочей области Matlab.



	A	B	C
1	Зависимость температуры воздуха в теплице от температуры снаружи		
2	Время суток	Температура снаружи, °C, x	Температура в теплице, °C, y
3	6	-32,2	17
4	5	-31,96	17
5	7	-31,88	17
6	4	-31,49	17
7	3	-31,25	17
8	8	-31,23	17
9	9	-30,9	17
10	2	-30,84	18,5
11	1	-30,01	19,35
12	24	-29,6	21,5
13	23	-28,86	21,5
14	10	-28,31	17
15	22	-27,39	23
16	21	-26,65	23
17	20	-25,61	23
18	19	-23,54	23
19	11	-23,14	18,5
20	18	-22,5	23
21	17	-22,02	23
22	16	-21,05	23
23	15	-20,56	23
24	12	-20,55	19,25
25	14	-20,54	21,5
26	13	-20,5	20,38

```
% Чтение данных из столбца C в вектор y1
y1=xlsread('PRIMER',3,'C3:C26');

% Чтение данных из столбца B в вектор x
x=xlsread('PRIMER',3,'B3:B26');
```

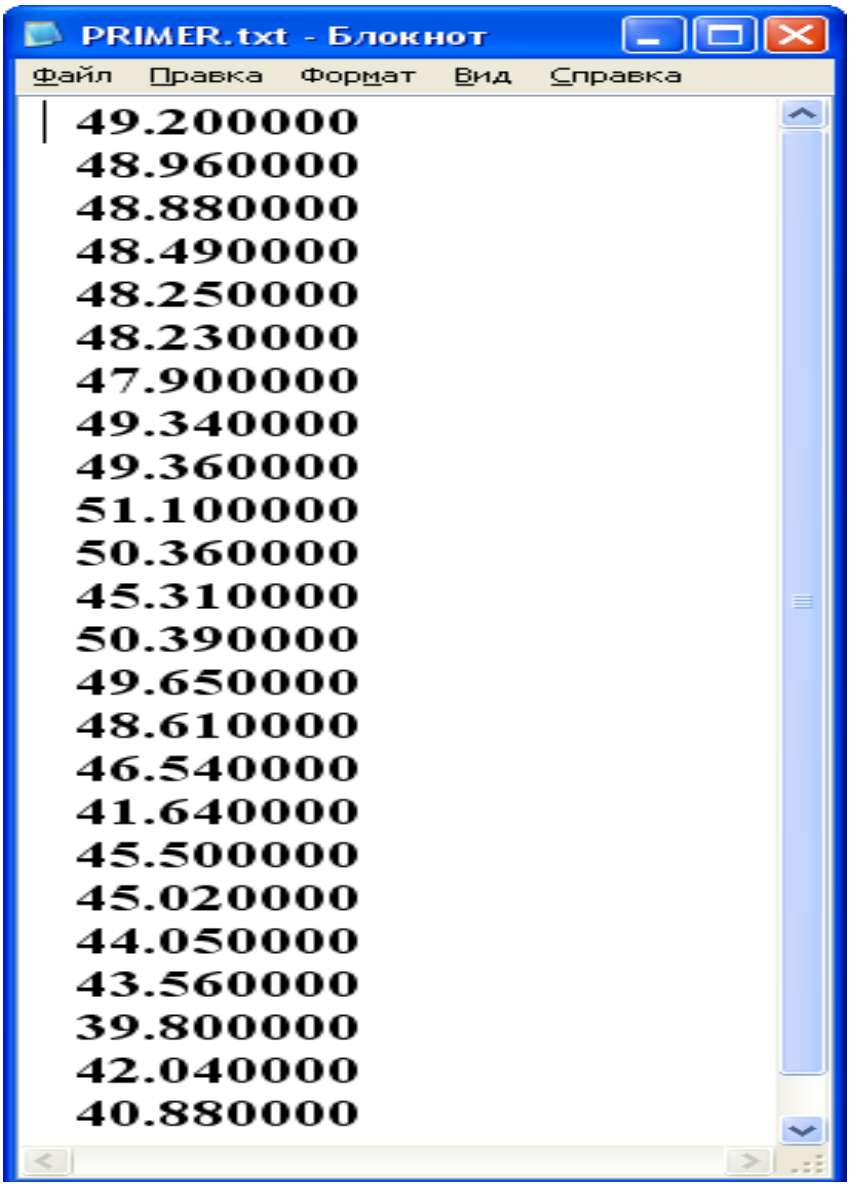
Номер листа

Адреса ячеек

Рисунок 3.6 – Лист Microsoft Excel, содержащий данные и функции для чтения данных в массивы рабочей области Matlab

Создать текстовый файл (см. вариант задания в п. 3.6).

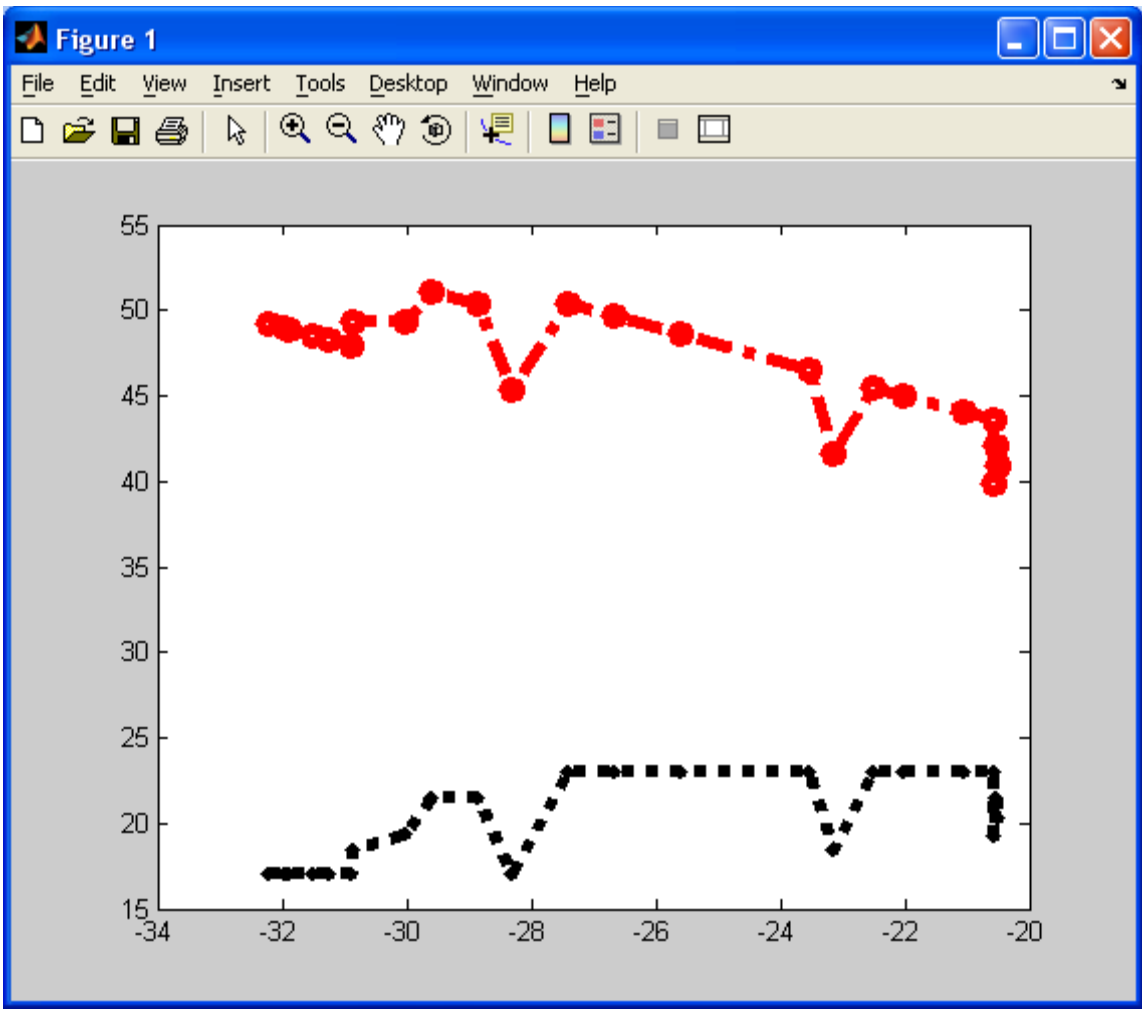
В примере используем текстовый файл, в котором сохранен вектор разности температуры воздуха в теплице и температуры снаружи. На рисунке 3.7 показан редактор «Блокнот», в окне которого можно просмотреть содержимое текстового файла, и функция для чтения данных из рабочей области Matlab.




```
% Чтение данных из файла "PRIMER.txt" в вектор y1
y2=load('PRIMER.txt');
```

Рисунок 3.7 – Редактор «Блокнот» с содержимым текстового файла

На рисунках 3.8 и 3.9 показаны графики и функции построения и форматирования графиков.

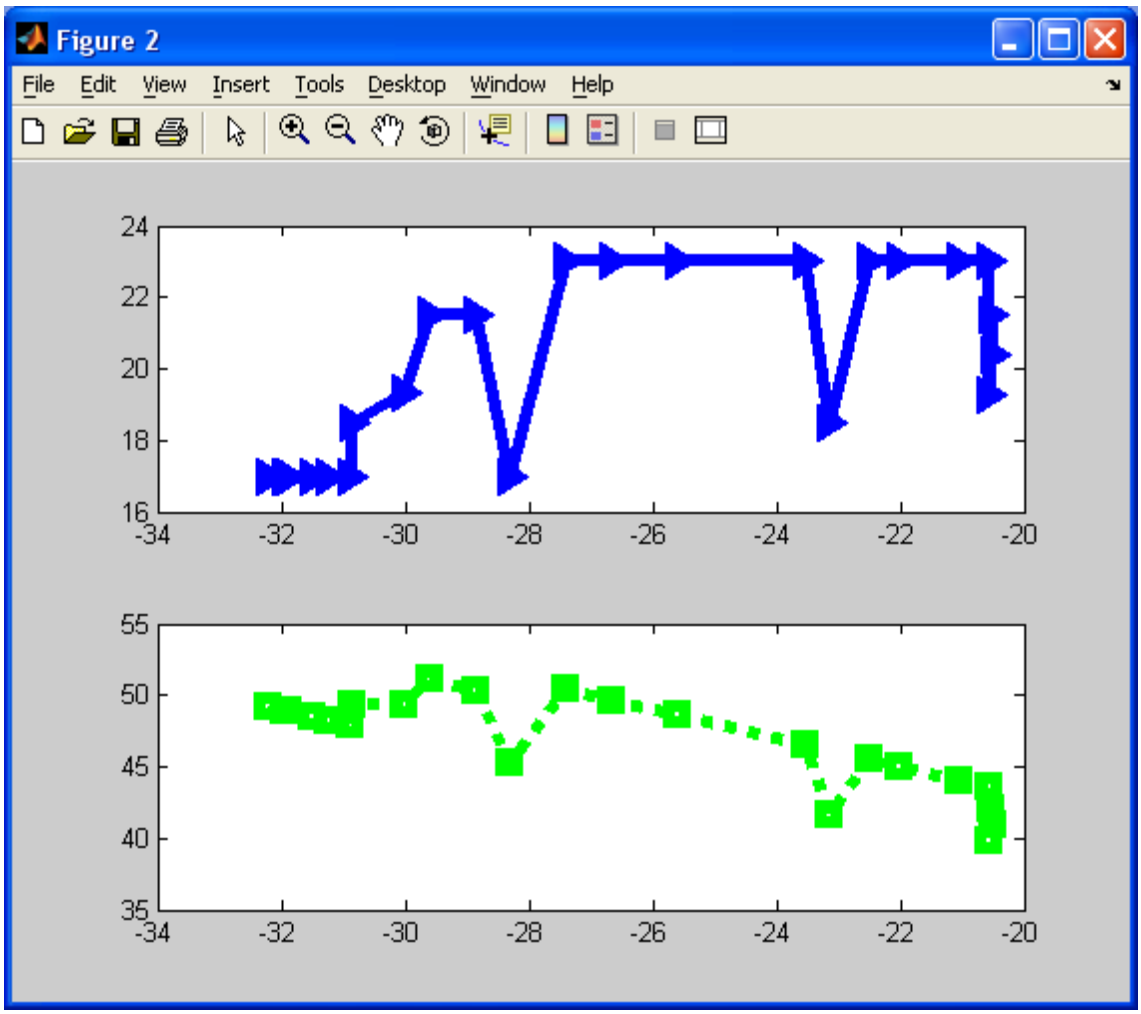


Построение первого графика:
x – вектор, отложенный по оси абсцисс;
y1 – вектор, отложенный по оси ординат.
Формат линии: «:» – стиль – двойной пунктир;
«x» – элемент крест;
«k» – цвет – черный

```
% Построение двух графиков в одних осях
plot(x,y1,':xk',x,y2,'-.or')
```

Построение второго графика:
x – вектор, отложенный по оси абсцисс;
y2 – вектор, отложенный по оси ординат.
Формат линии: «-.» – стиль – штрих-пунктир;
«o» – элемент окружность;
«r» – цвет – красный

Рисунок 3.8 – Графики и функции построения и форматирования графиков в одном графическом окне



Создание матрицы осей координат:
2 – число строк, 1 – число столбцов,
1 – выбор первых осей для построения графика

```
% Создание осей графическом окне
subplot(2,1,1)
% Построение графика
plot(x,y1,'->b');
% Создание осей графическом окне
subplot(2,1,2)
% Построение графика
plot(x,y2,':sg');
```

Создание матрицы осей координат:
2 – число строк, 1 – число столбцов,
2 – выбор вторых осей для построения графика

Рисунок 3.8 – Графики и функции построения и форматирования графиков в двух графических окнах

Весь сценарий чтения данных из дисковых файлов Microsoft Excel и текстового, построения графиков и форматирования графиков приведен на рисунке 3.10.

Функции **figure**(1) и **figure**(2) создают графические окна с заголовками «**Figure** 1» и «**Figure** 2» соответственно. После создания окна оно становится текущим, построение осей, графиков и так далее происходит в текущем окне.

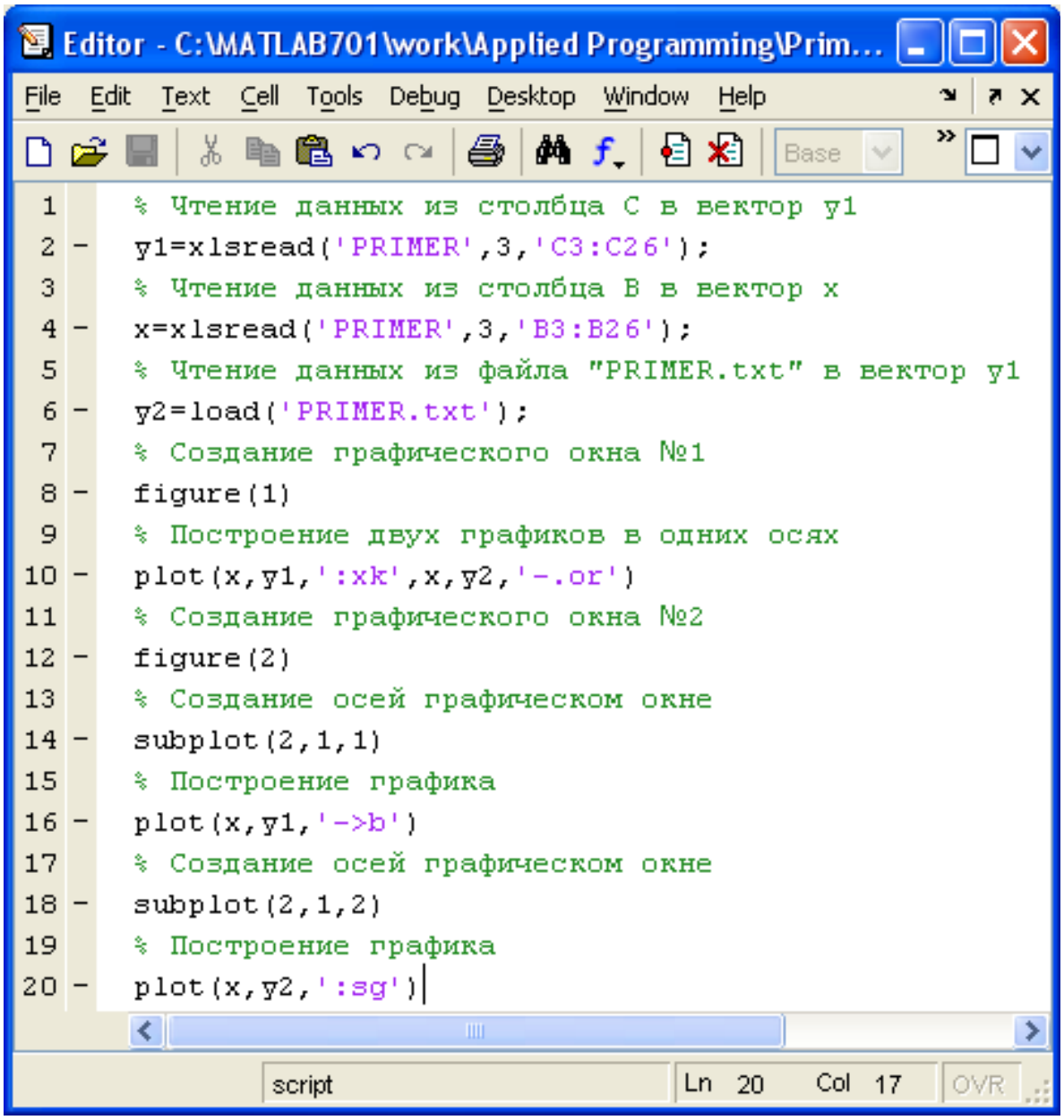


Рисунок 3.10 – Сценарий чтения данных из дисковых файлов Microsoft Excel и текстового, построения графиков и форматирования графиков

Построение трехмерных графиков

Необходимо построить график функции двух переменных (см. вариант задания в п. 3.6).

Для примера построим график функции

$$z = -\sqrt{\arcsin(x)^2 + \arcsin(y-7)^2}.$$

Текст сценария построения трехмерного графика функции от двух переменных приведен на рисунке 3.11.

Построенный в результате работы данного сценария график смотрите на рисунке 3.12.

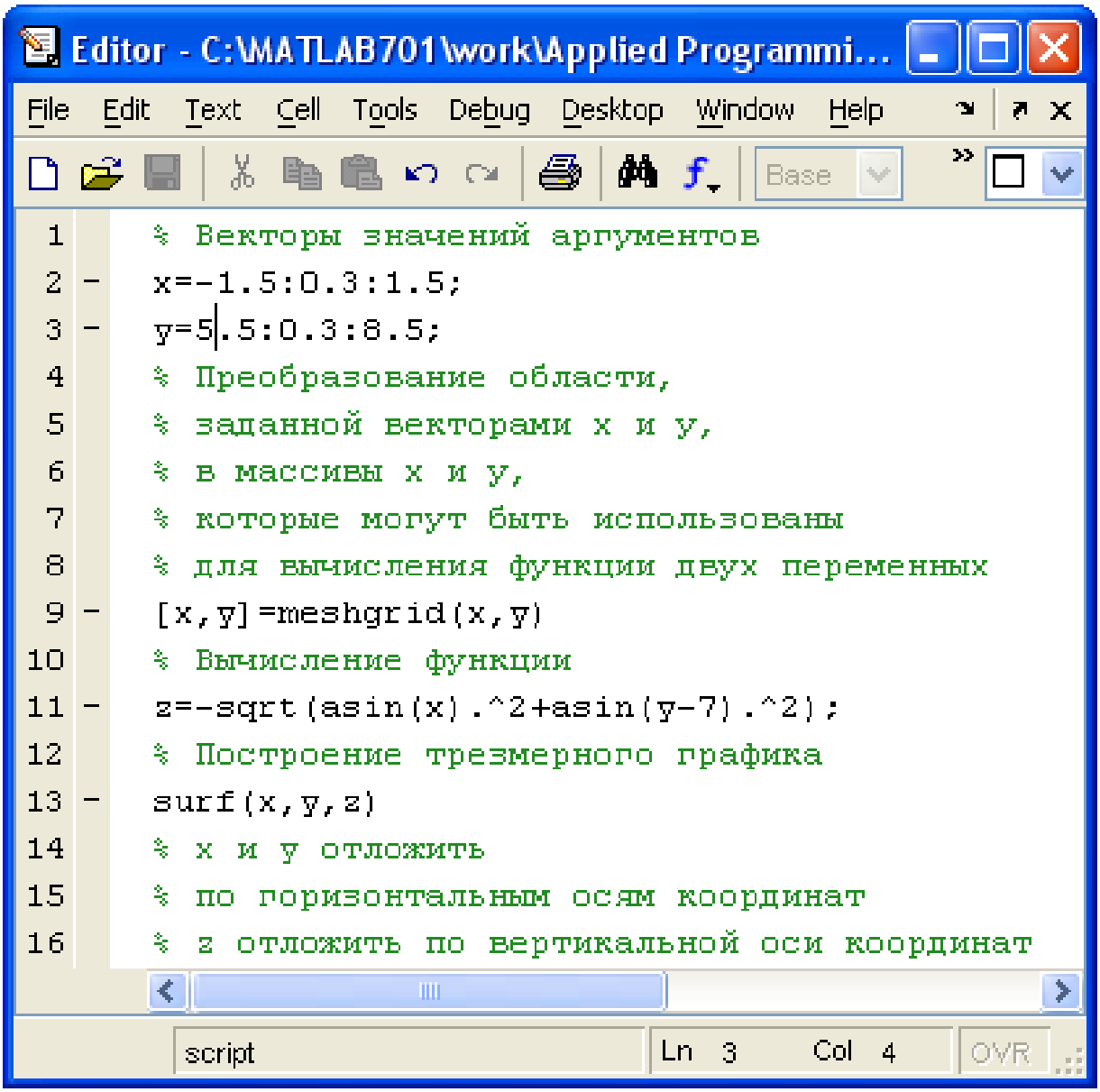


Рисунок 3.11 – Текст сценария построения трехмерного графика функции от двух переменных

Функция [**x**,**y**]=**meshgrig**(**x**,**y**) создает матрицы **x** и **y**.
Матрица **x** состоит из одиннадцати одинаковых строк:
-1.50 -1.20 -0.90 -0.60 -0.30 0 0.30 0.60 0.90 1.20 1.50
Матрица **y** состоит из одиннадцати одинаковых столбцов:
5.5
5.8
6.1
6.4
6.7
7.0
7.3
7.6
7.9
8.2
8.5

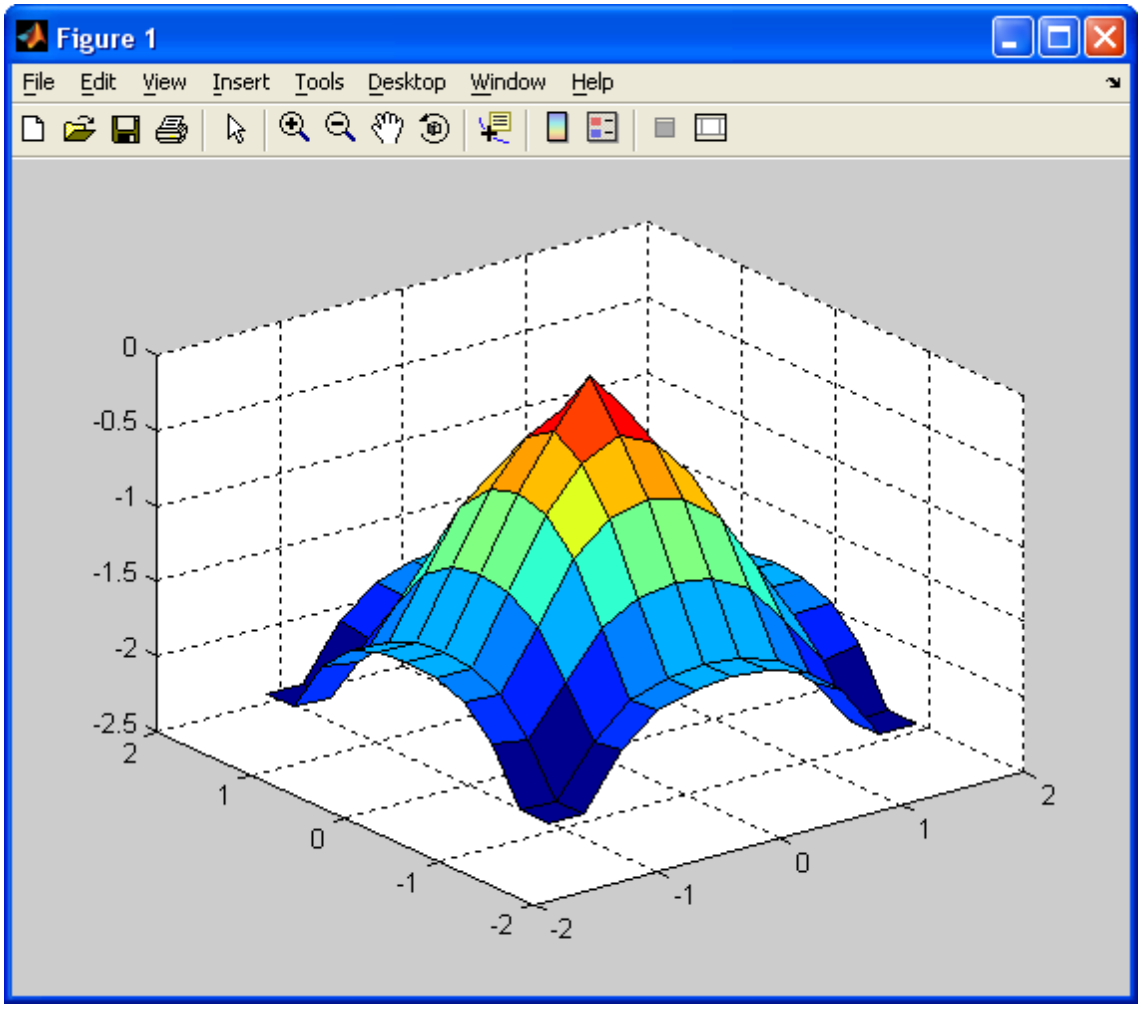


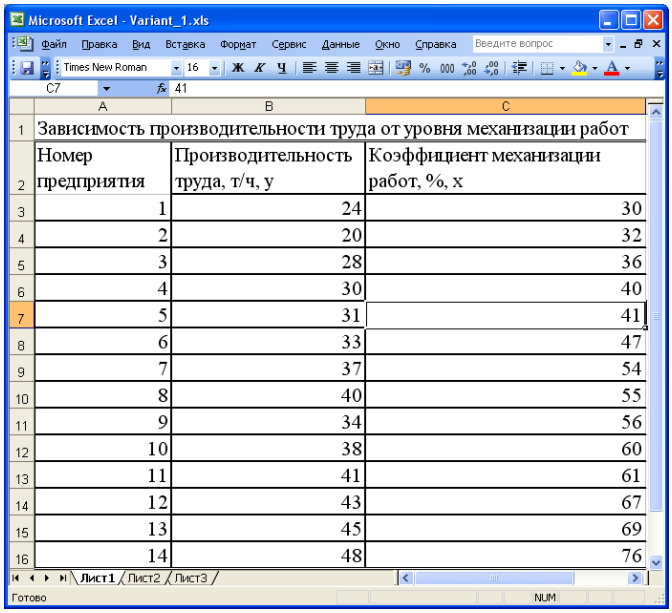
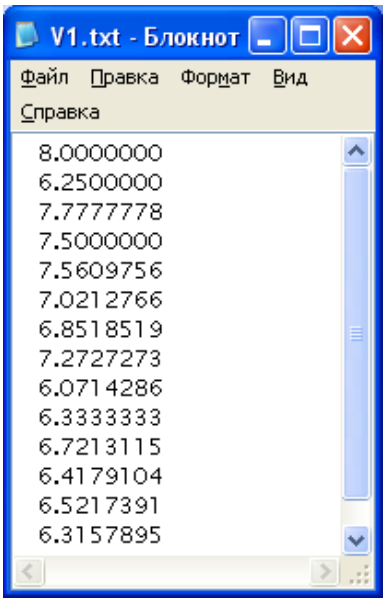
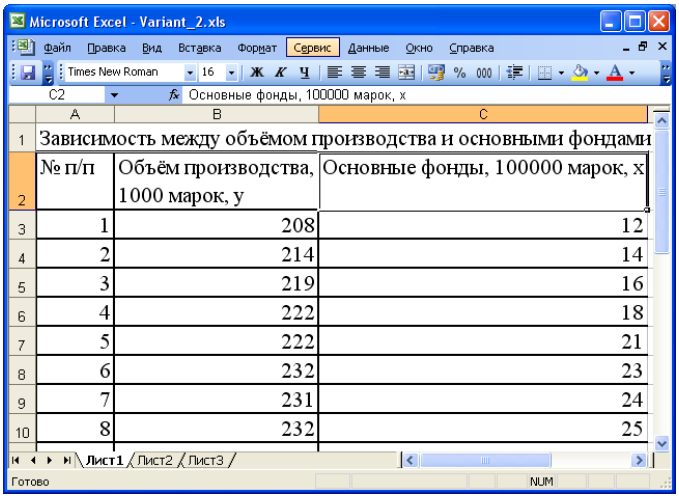
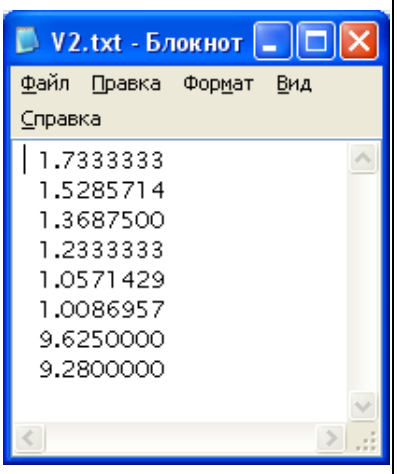
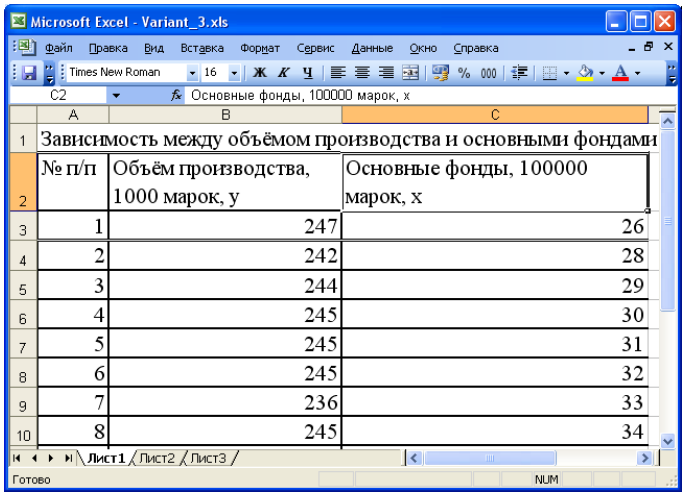
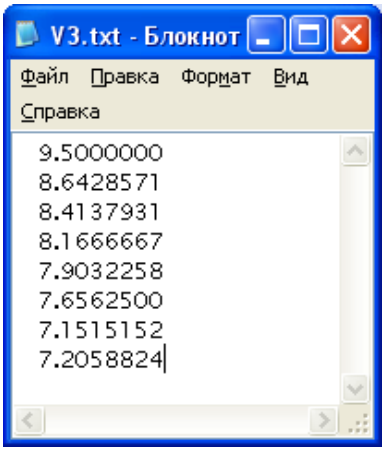
Рисунок 3.12 – Результат работы сценария построения трехмерного графика

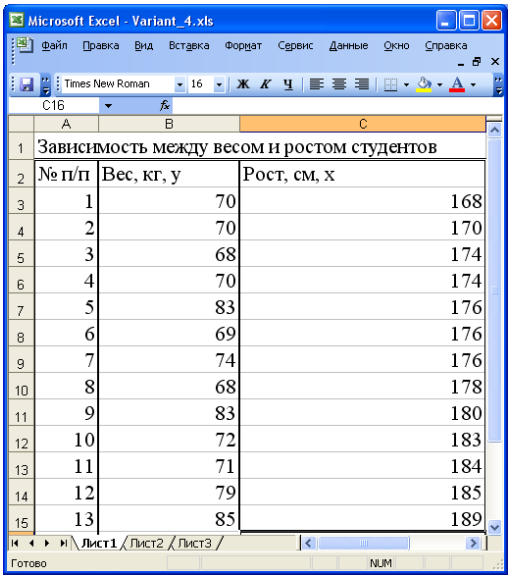
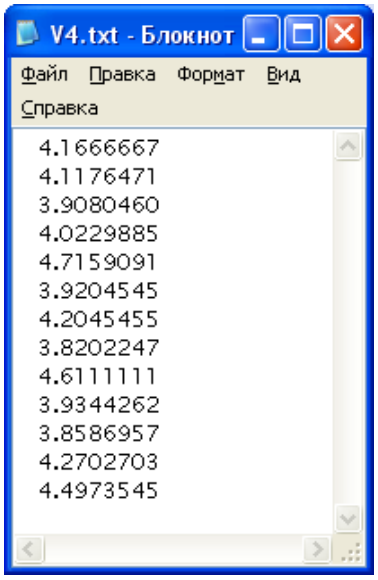
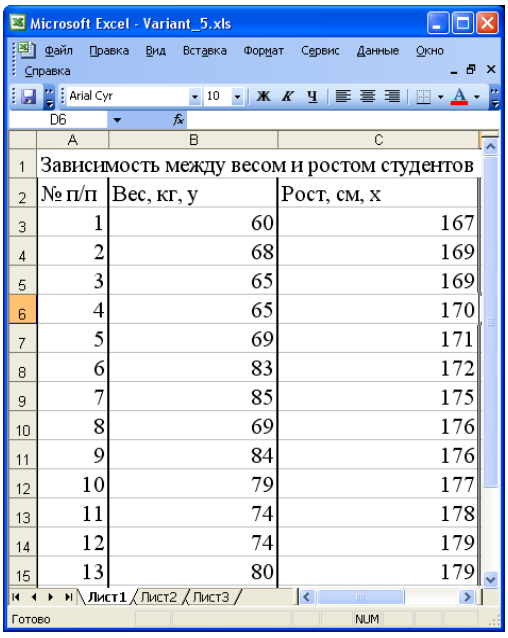
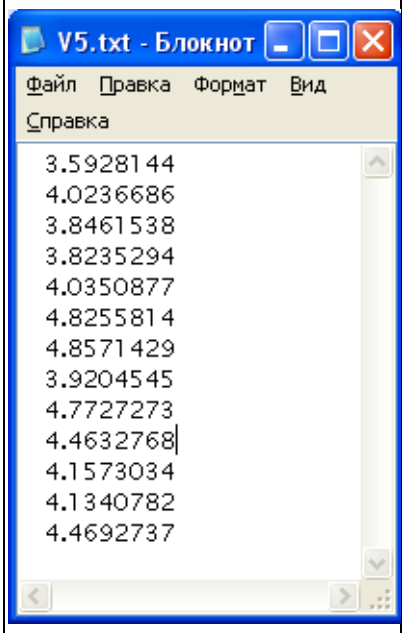
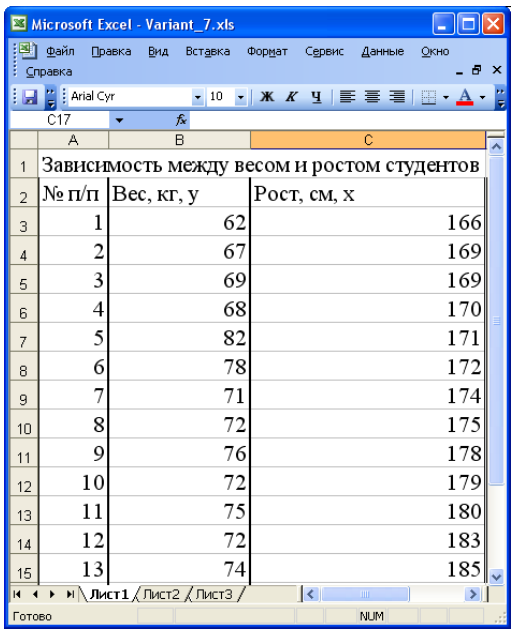
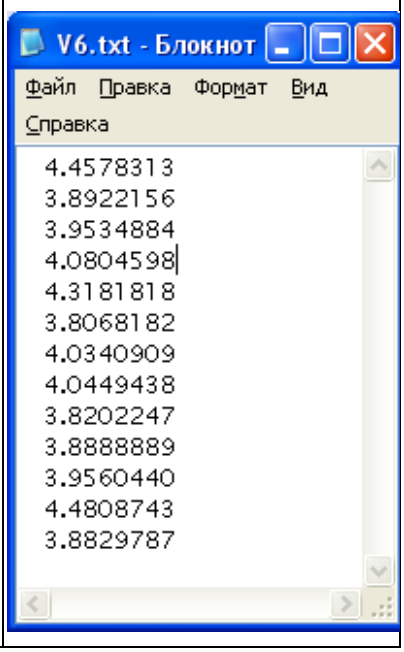
3.6. Варианты заданий

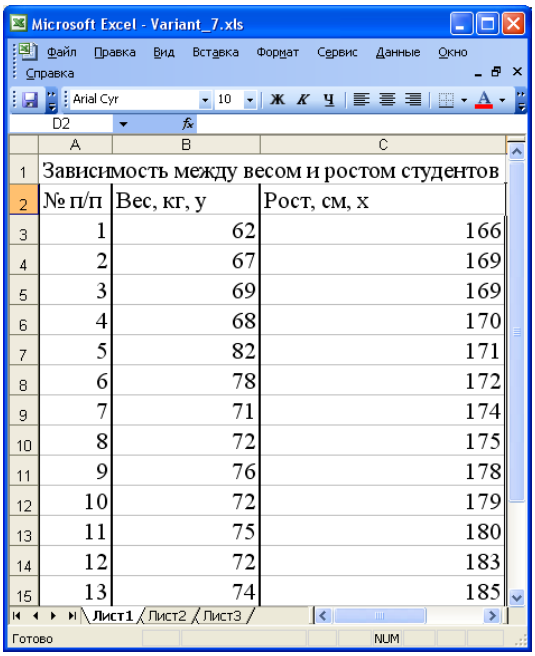
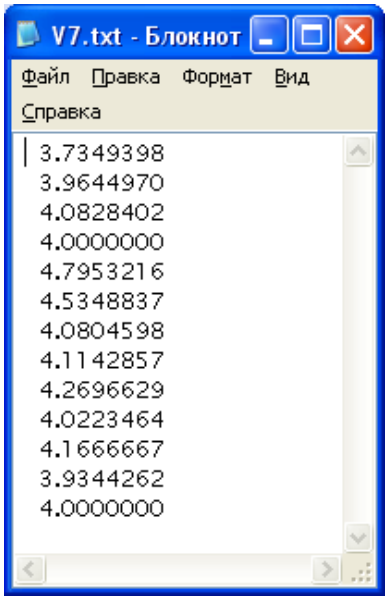
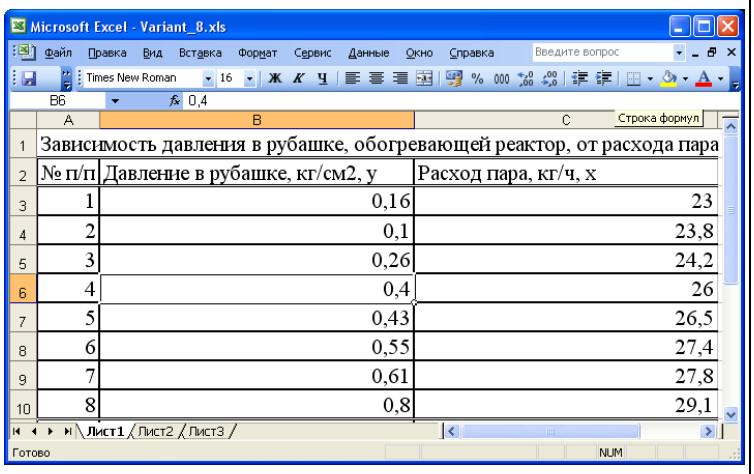
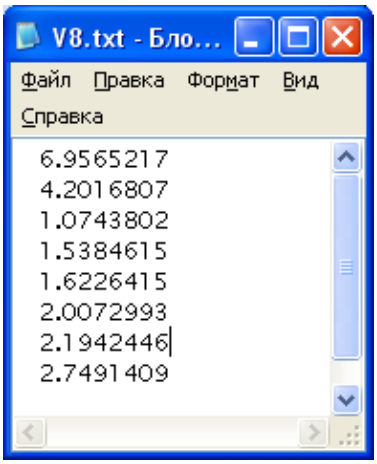
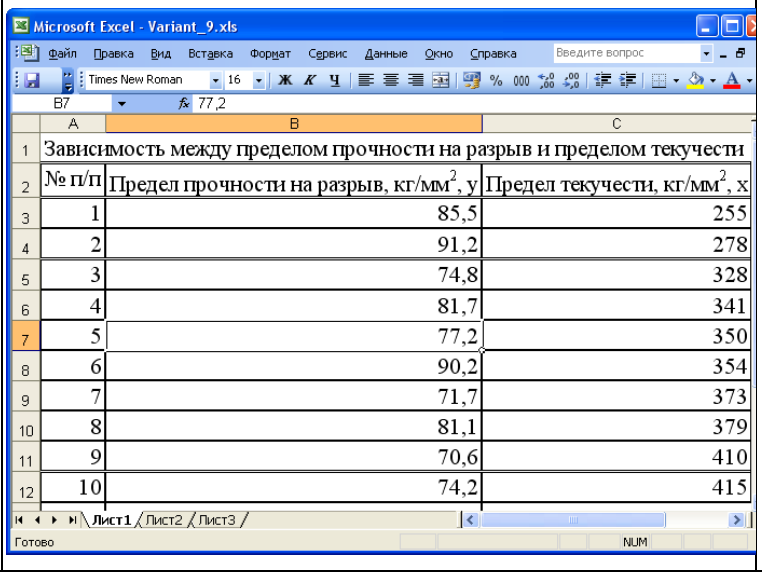
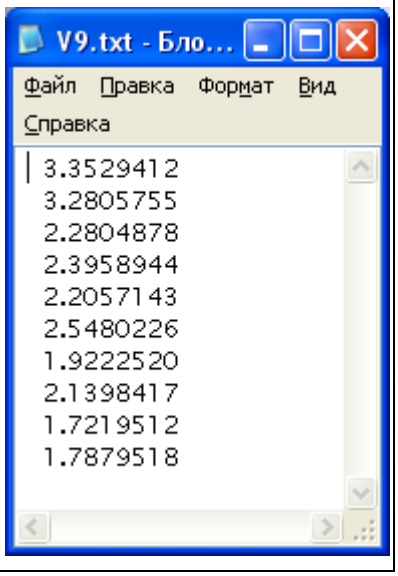
Ввод и вывод в «Command Window». Сохранение данных в текстовом файле и в файле формата Microsoft Excel

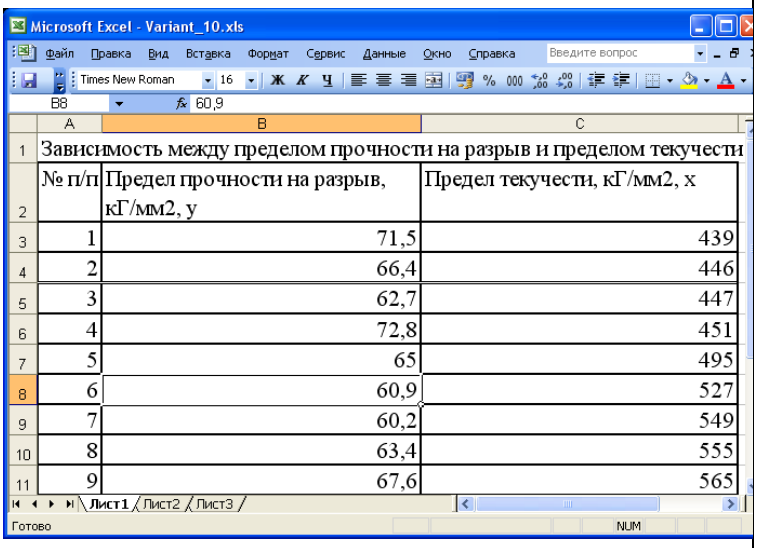
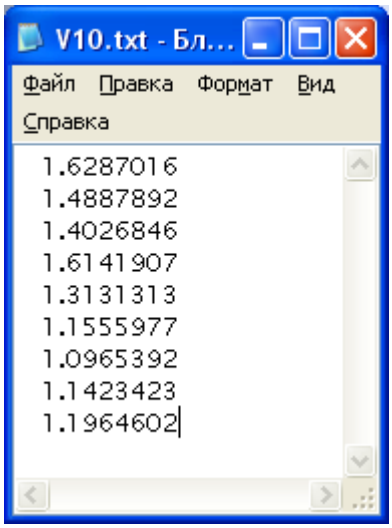
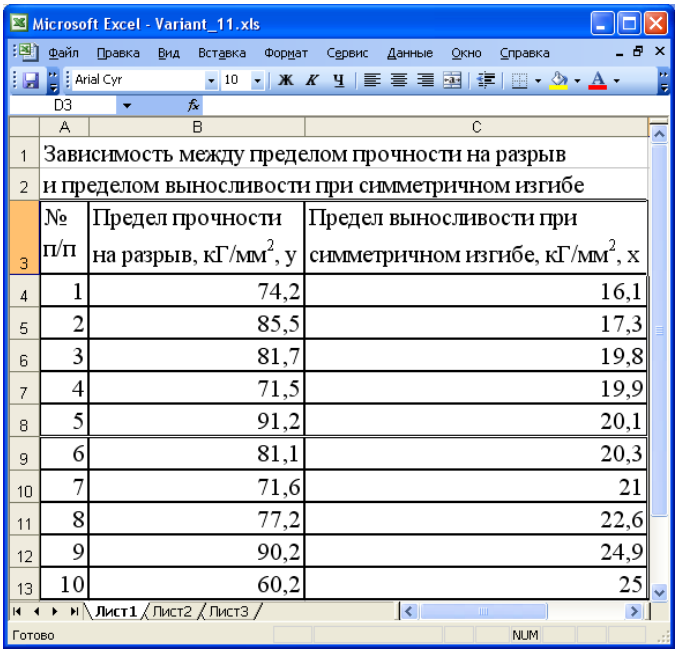
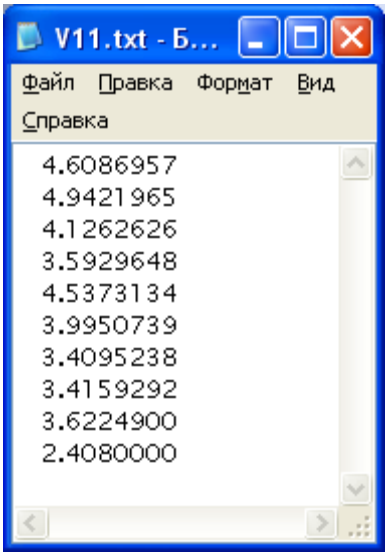
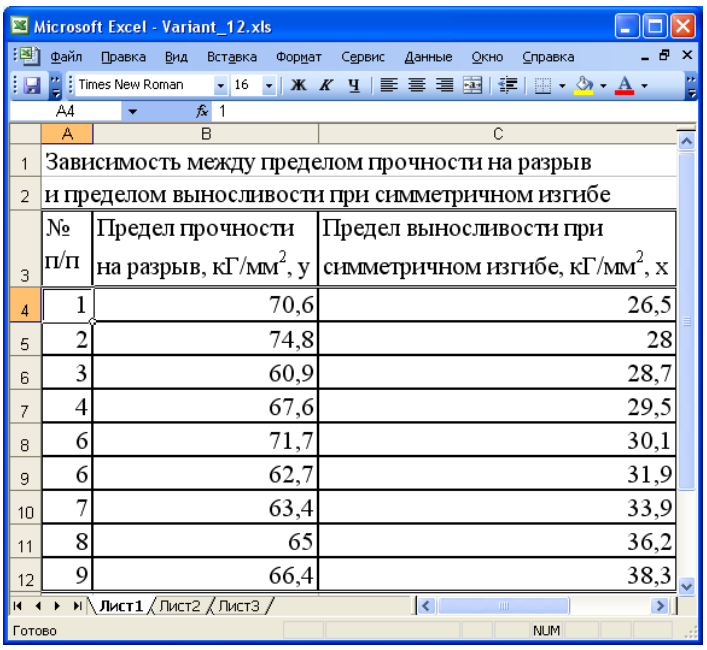
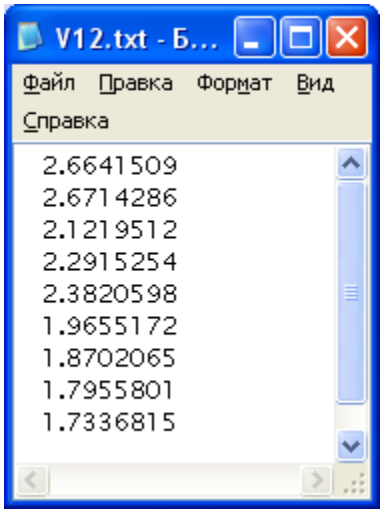
Номер варианта	Организовать ввод массива размерами	Сохранить массив в файле формата Microsoft Excel	
		Номер листа	Верхняя левая ячейка
1	3×3	3	C10
2	3×4	2	D9
3	3×5	1	E8
4	2×6	3	F7
5	4×3	2	G6
6	4×4	1	H5
7	4×2	3	I4
8	4×3	2	J3
9	2×3	1	K10
10	4×4	3	L9
11	5×3	2	C8
12	5×2	1	D7
13	6×2	3	E6
14	6×3	2	F5
15	2 ×4	1	G4
16	3×5	3	H3
17	4×3	2	I10
18	7×2	1	J9
19	4×3	3	K8
20	2×4	2	L7
21	3×5	1	C6
22	2×6	3	D5
23	5×3	2	E4
24	4×8	1	F3

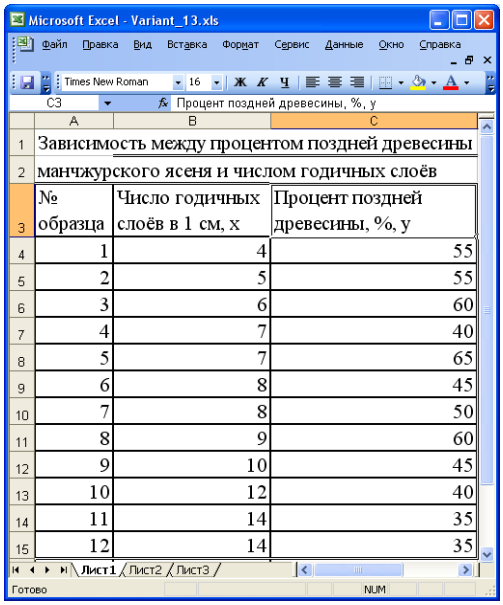
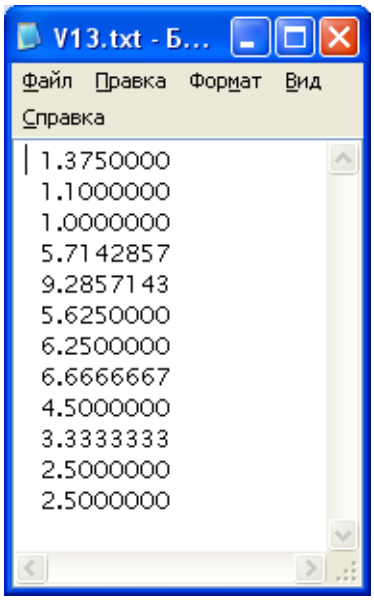
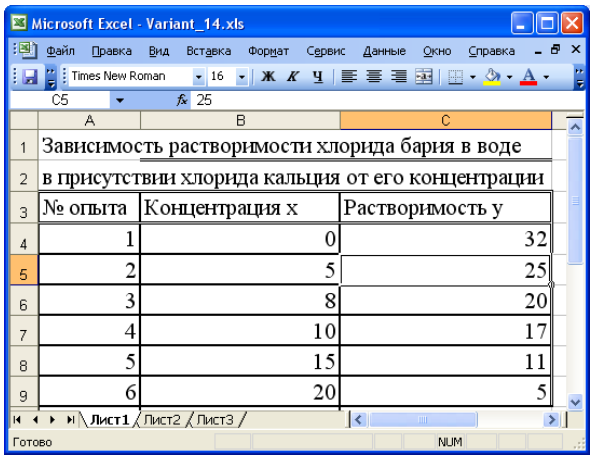
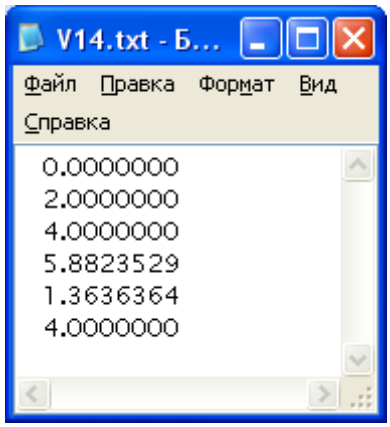
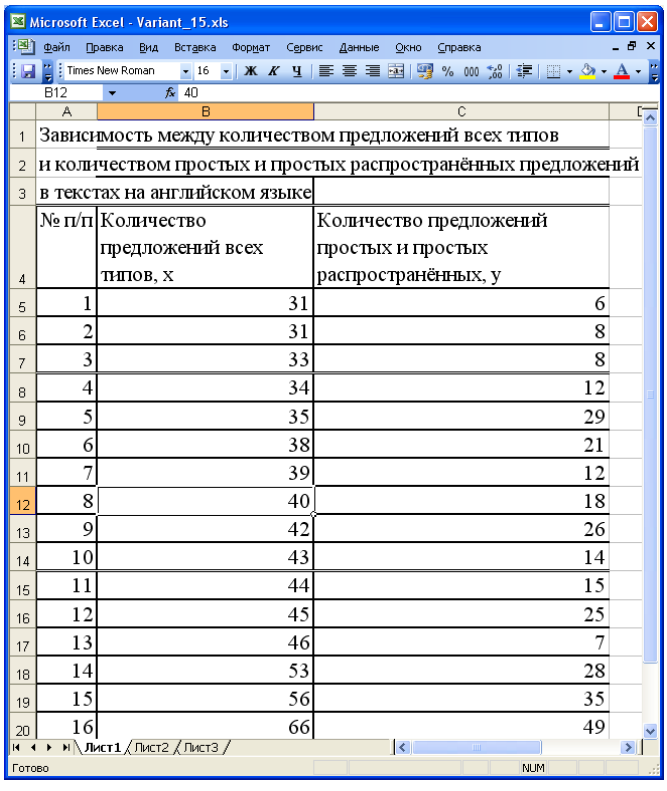
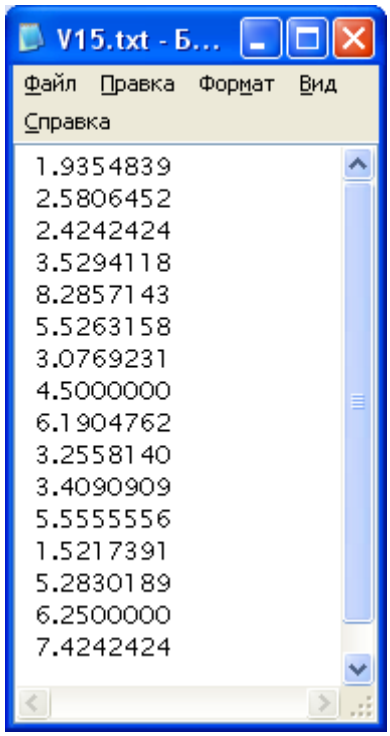
Ввод данных в рабочую область Matlab из дискового файла

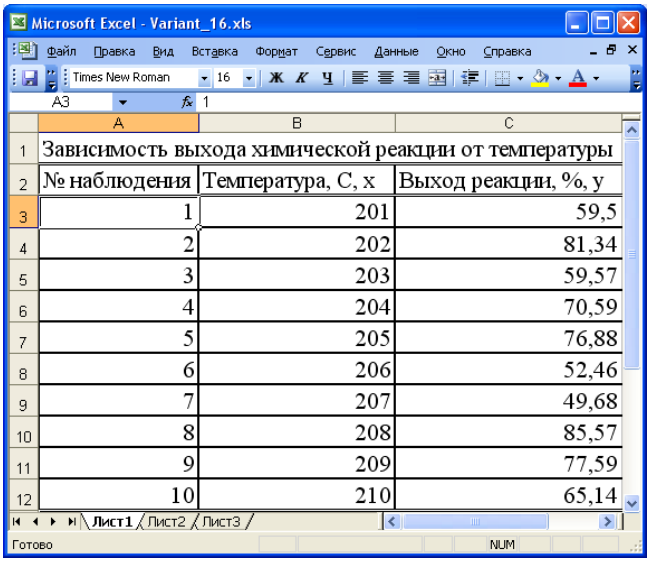
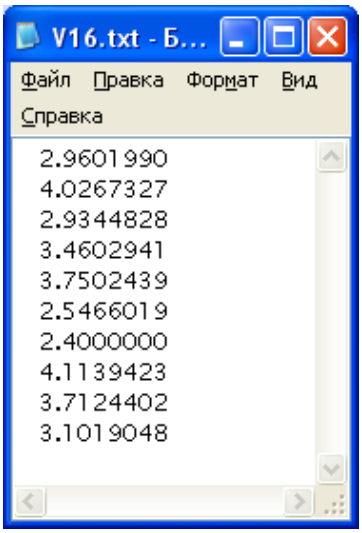
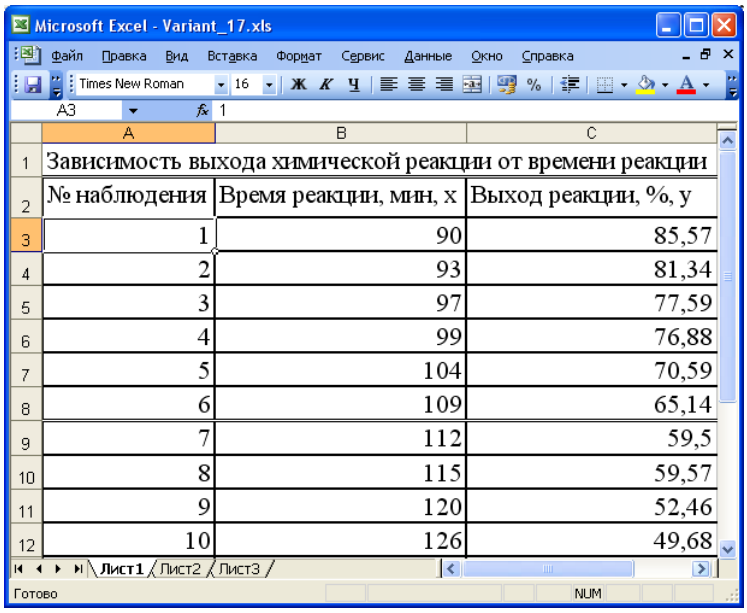
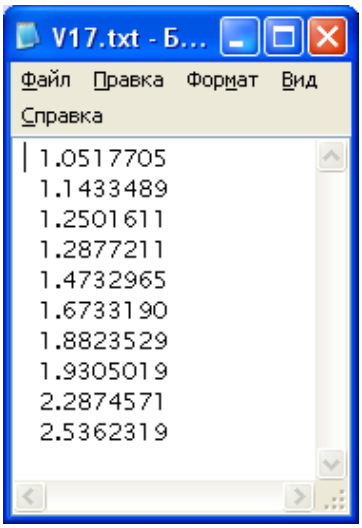
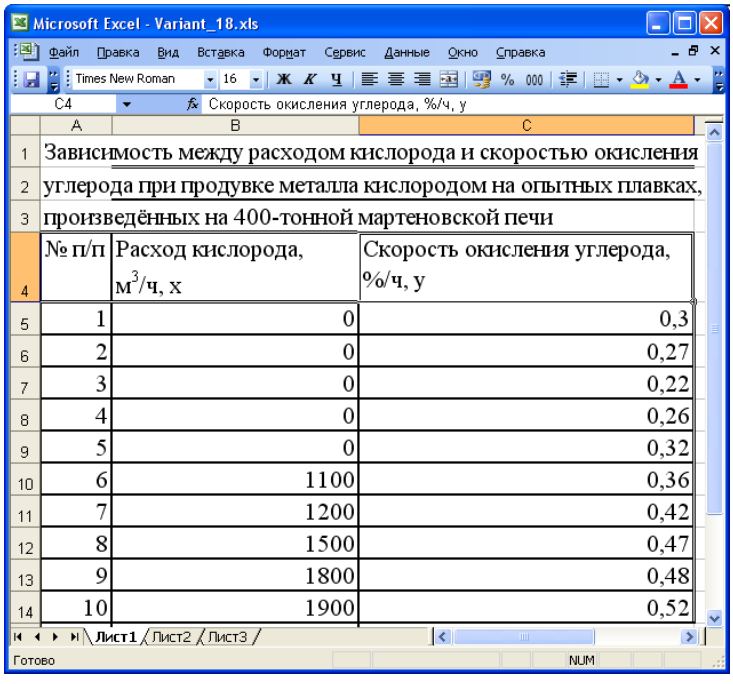
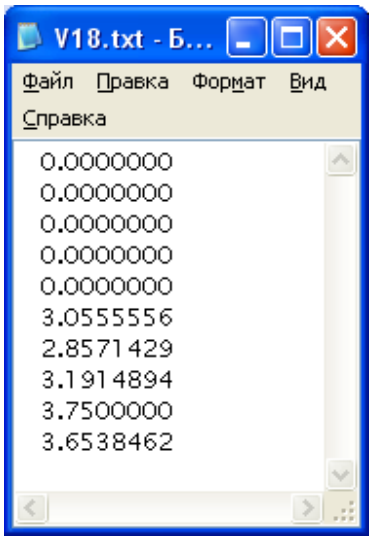
Вариант	Файл Microsoft Excel	Текстовый файл
1		
2		
3		

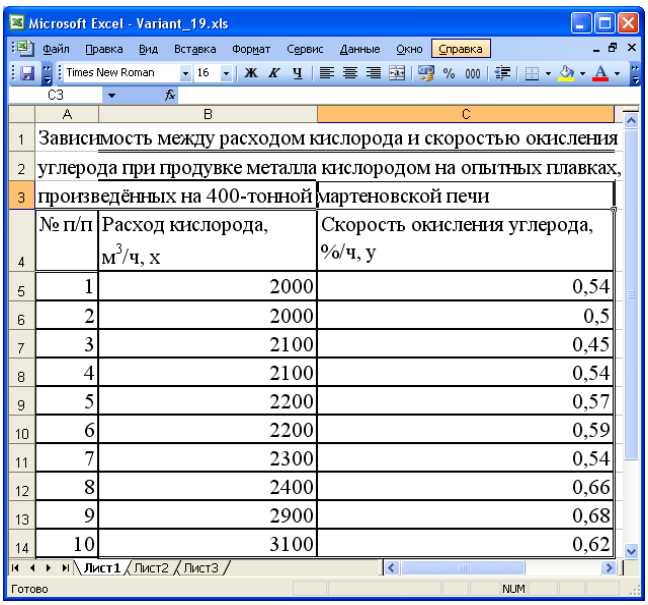
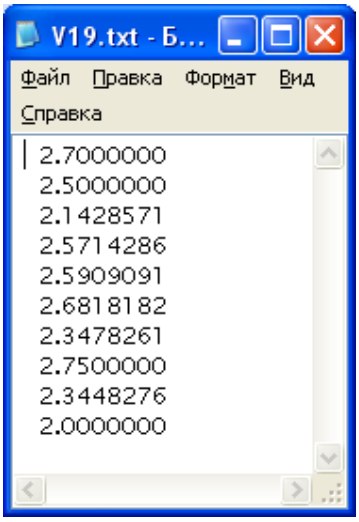
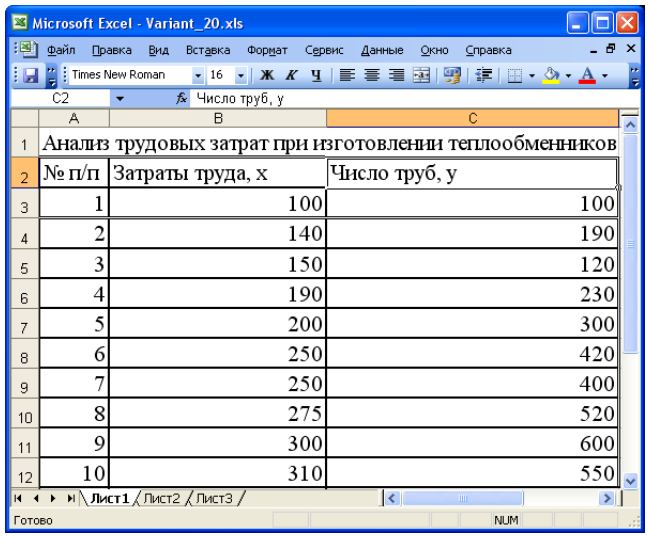
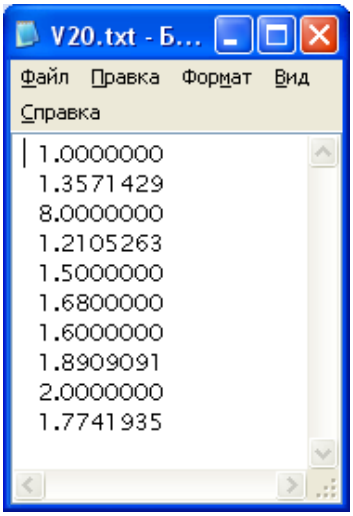
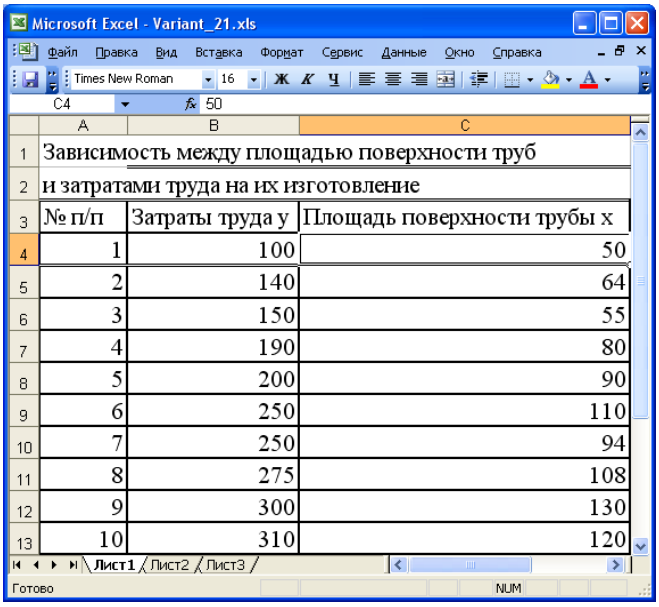
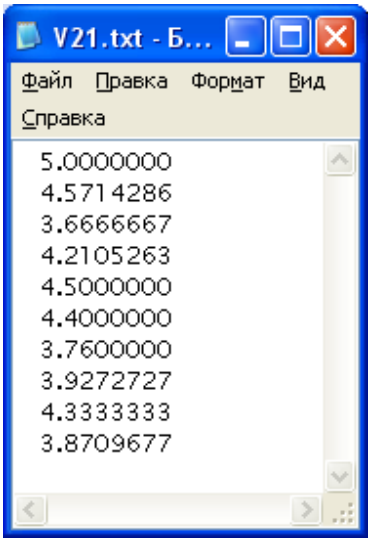
Вариант	Файл Microsoft Excel	Текстовый файл
4		
5		
6		

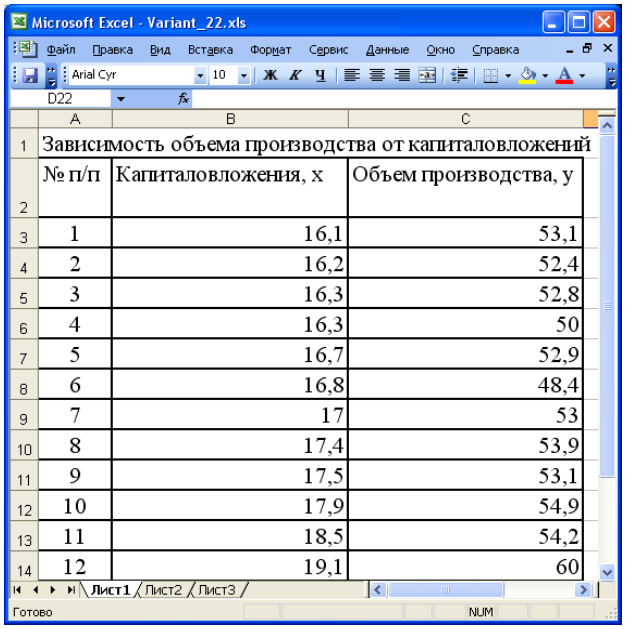
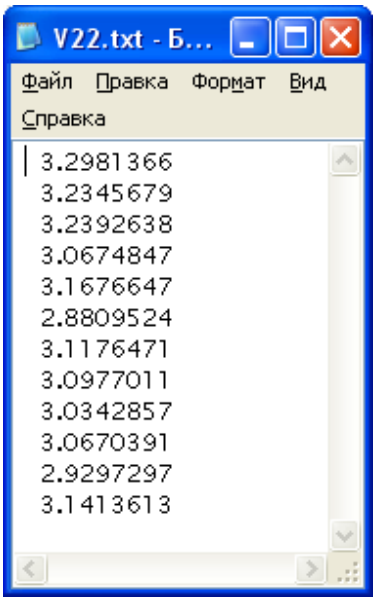
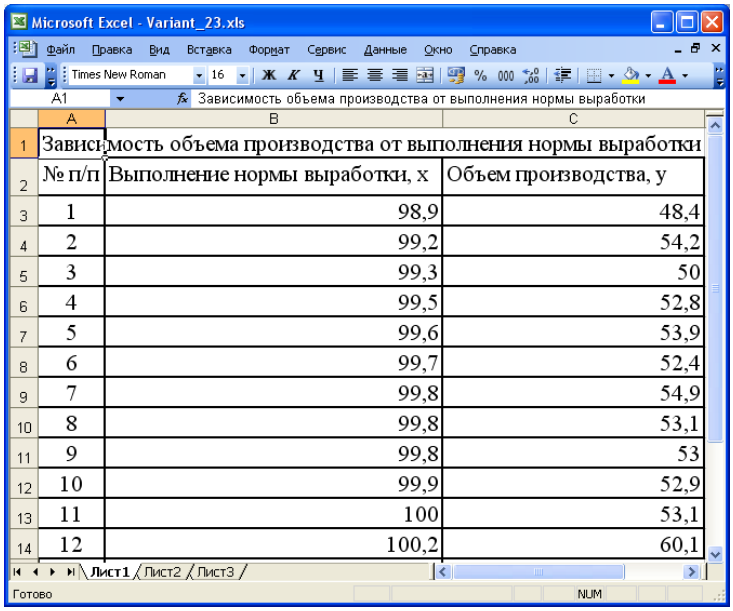
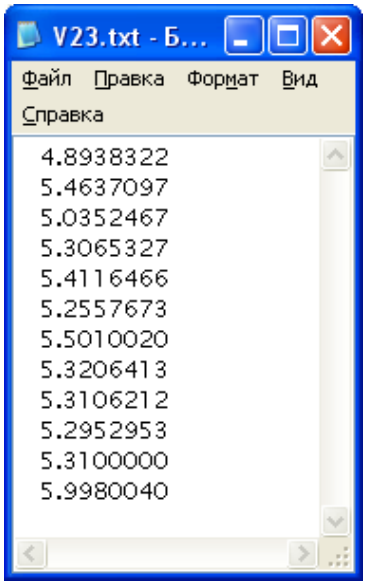
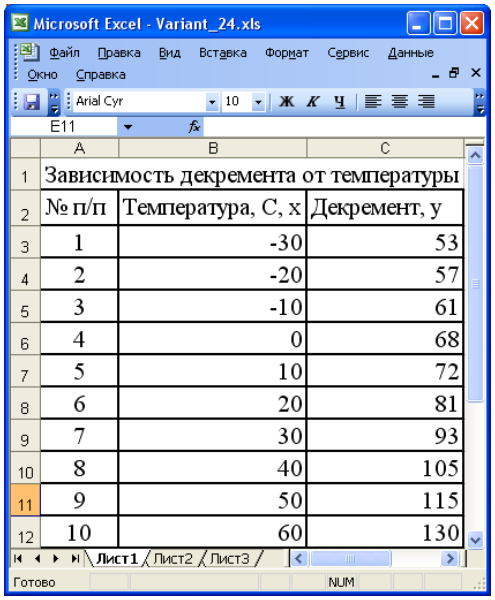
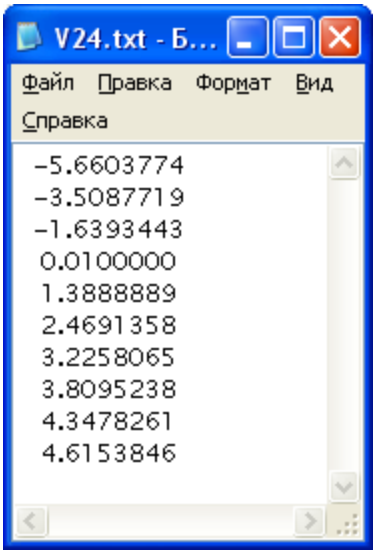
Вариант	Файл Microsoft Excel	Текстовый файл
7		
8		
9		

Вариант	Файл Microsoft Excel	Текстовый файл
10		
11		
12		

Вариант	Файл Microsoft Excel	Текстовый файл
13		
14		
15		

Вариант	Файл Microsoft Excel	Текстовый файл
16		
17		
18		

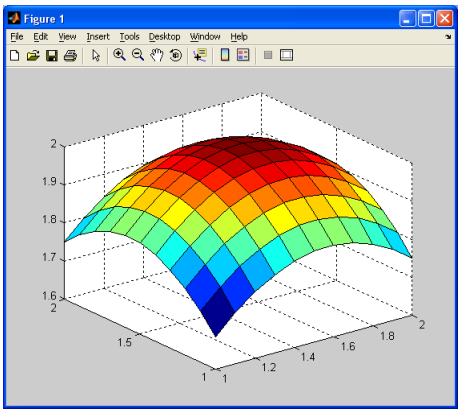
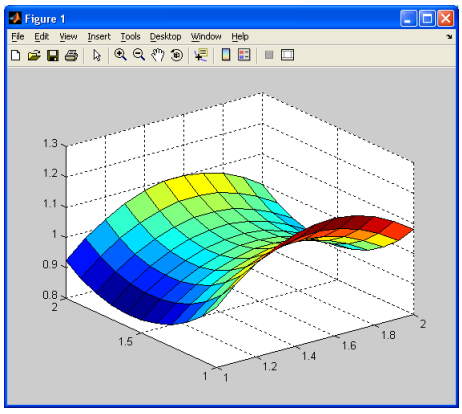
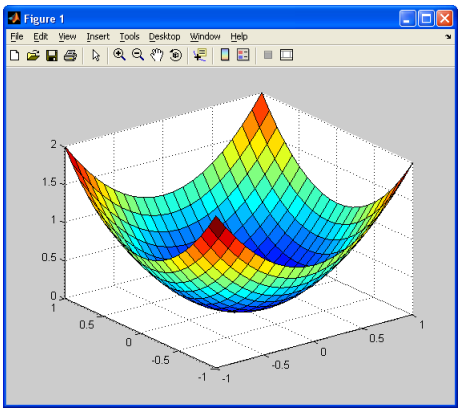
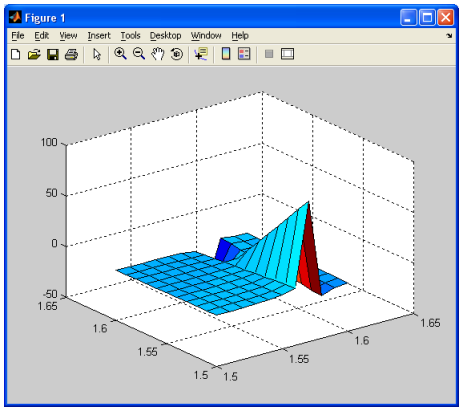
Вариант	Файл Microsoft Excel	Текстовый файл
19		
20		
21		

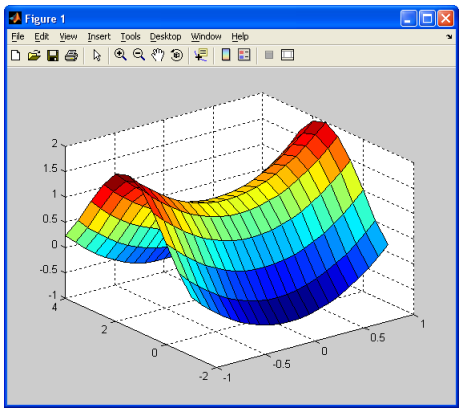
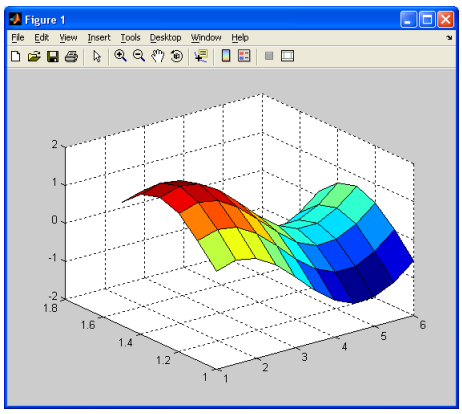
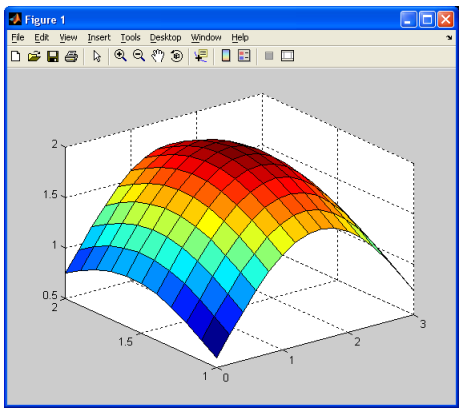
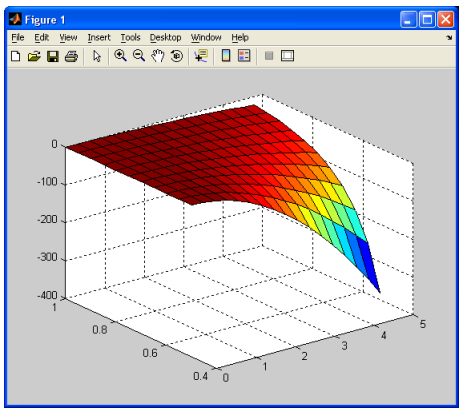
Вариант	Файл Microsoft Excel	Текстовый файл
22		
23		
24		

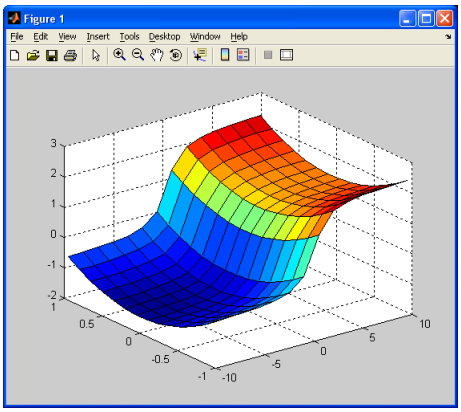
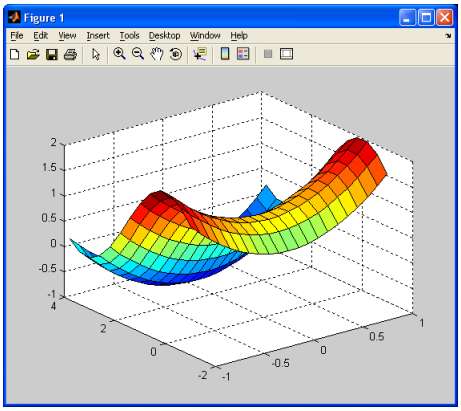
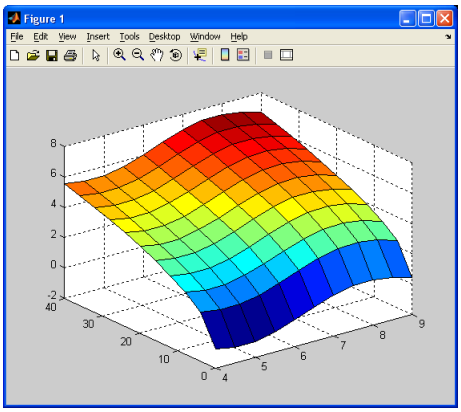
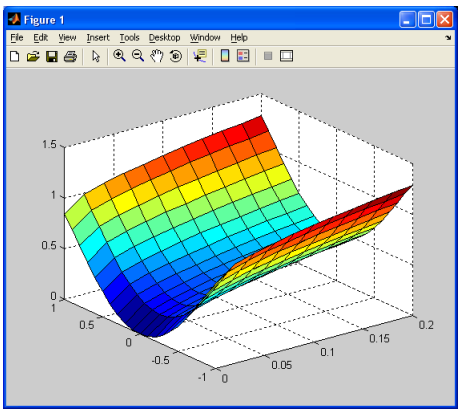
Ввод данных в рабочую область Matlab из дискового файла

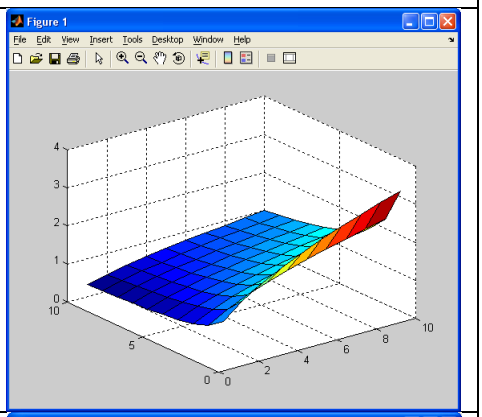
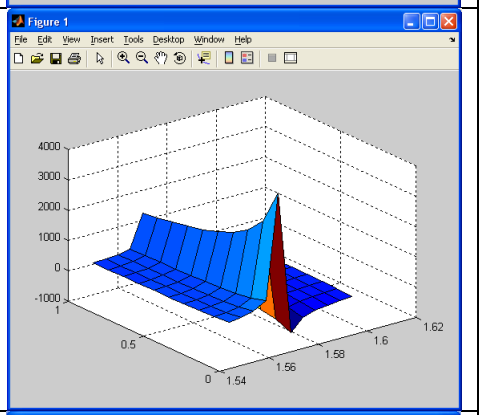
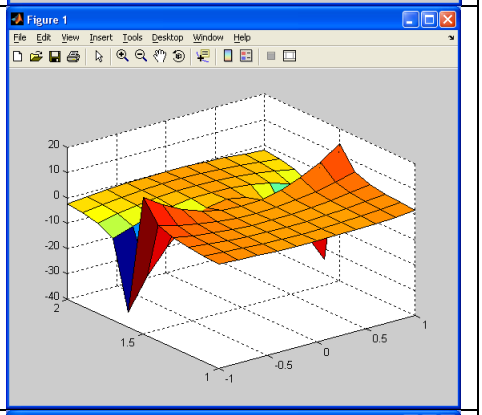
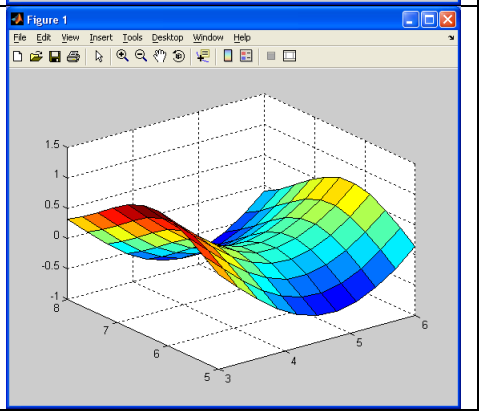
Номер варианта	Размер матриц для поэлементного сложения, поэлементного умножения и деления	Размеры матриц для матричного умножения	
1	3×3	7×10	10×7
2	3×4	7×9	9×7
3	3×5	7×8	8×7
4	3×6	7×6	6×7
5	4×3	7×5	5×7
6	4×4	7×4	4×7
7	4×5	7×3	3×7
8	4×6	7×2	2×7
9	5×3	6×5	5×6
10	5×4	6×4	4×6
11	5×5	6×3	3×6
12	5×6	6×2	2×6
13	6×2	5×7	3×4
14	6×3	5×4	4×5
15	6×4	5×3	3×5
16	6×5	5×2	2×5
17	6×6	4×6	6×4
18	7×2	4×5	5×4
19	7×3	4×3	3×4
20	7×4	4×2	2×4
21	7×5	3×7	7×3
22	7×6	3×6	6×3
23	5×7	3×5	5×3
24	4×8	3×4	4×3

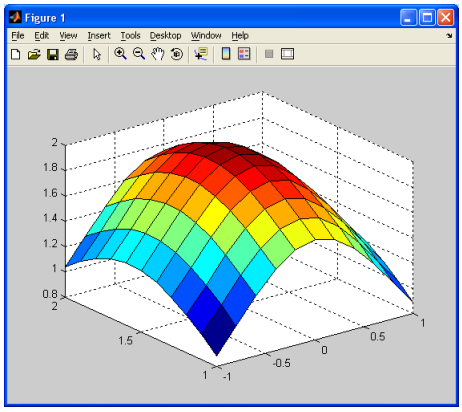
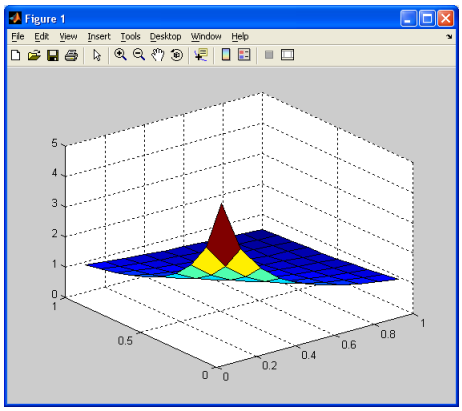
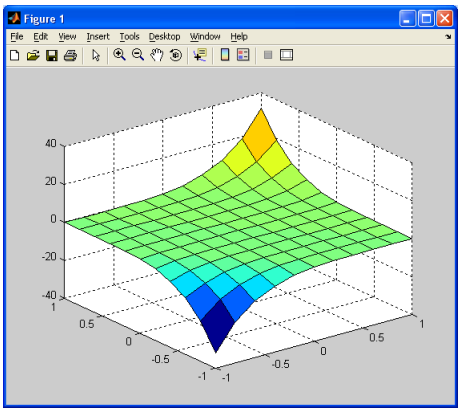
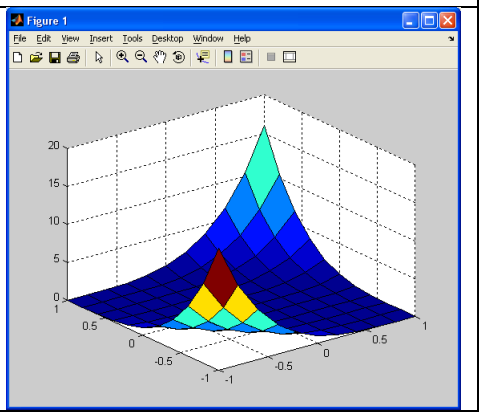
Построить график функции

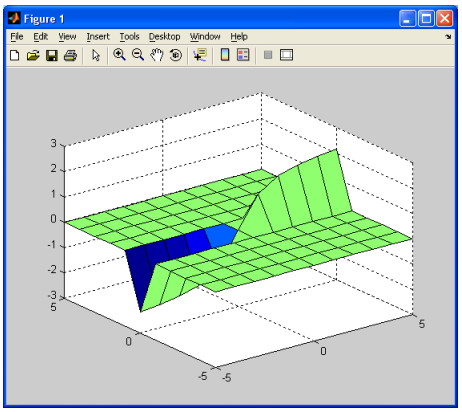
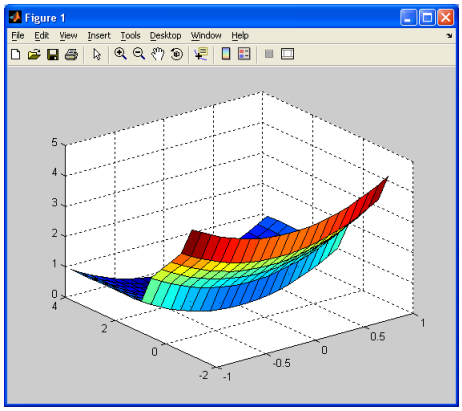
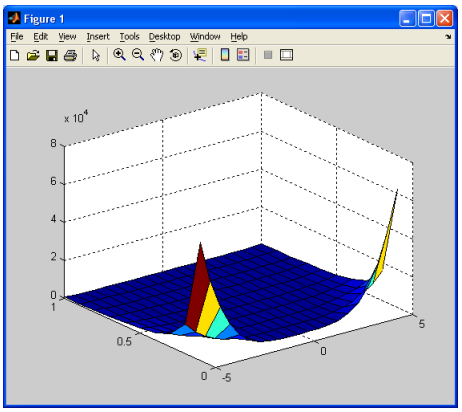
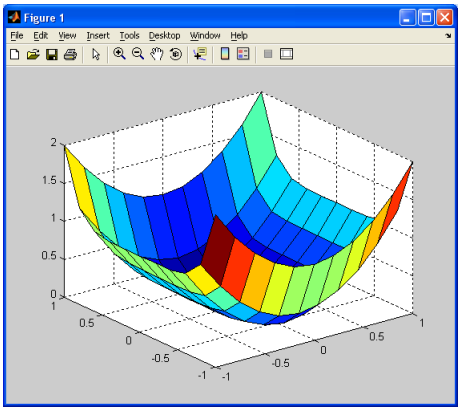
Номер варианта	Задание	Интервал значений X	Интервал значений Y	Результат
1	z=sin(X)+sin(Y)	[1; 2]	[1; 2]	
2	$z = \frac{\sin(X)}{\sin(Y)}$	[1; 2]	[1; 2]	
3	$z=X^2+Y^2$	[-1; 1]	[-1; 1]	
4	$z = \frac{tg(X)}{tg(Y)}$	[1,5; 1,6]	[1,5; 1,6]	

Номер варианта	Задание	Интервал значений X	Интервал значений Y	Результат
5	$z=X^2+\sin(Y)$	[-1; 1]	[-1; 4]	
6	$z=\sin(X)+\sin(6Y)$	[1; 6]	[1; 1,5]	
7	$z=\sin(X)+\sin(Y)^3$	[0; 3]	[1; 2]	
8	$z=-\frac{X^2}{Y^4}$	[0; 5]	[0,5; 1]	

Номер варианта	Задание	Интервал значений X	Интервал значений Y	Результат
9	$z=arct(X)+Y^2$	[-10; 10]	[-1; 1]	
10	$z=X^2+\cos(Y)$	[-1; 1]	[-1; 4]	
11	$z=\sin(X)+\sqrt{Y}$	[4; 9]	[0; 40]	
12	$z=\sqrt{X}+\sin(Y^2)$	[0; 0,2]	[-1; 1]	

Номер варианта	Задание	Интервал значений X	Интервал значений Y	Результат
13	$z = \frac{\sqrt{X}}{\sqrt{Y}}$	[1; 10]	[1; 10]	
14	$z = \frac{tg(X)}{\sqrt{Y}}$	[1,55; 1,6]	[0,1; 1]	
15	$z=X^2tg(Y)$	[-1; 1]	[1; 2]	
16	$z = \sin(X)+\cos\left(\frac{Y}{2}\right)^4$	[3; 6]	[5; 8]	

Номер варианта	Задание	Интервал значений X	Интервал значений Y	Результат
17	$z=\cos(X)^2+\sin(Y)^3$	[-1; 1]	[1; 2]	
18	$z=\frac{(Y+X)^2}{(Y+X)^3}$	[0,1; 1]	[0,1; 1]	
19	$z=(Y+X)^2(Y+X)^3$	[-1; 1]	[-1; 1]	
20	$z=(Y+X)^2(Y+X)^2$	[-1; 1]	[-1; 1]	

Номер варианта	Задание	Интервал значений X	Интервал значений Y	Результат
21	$z=\operatorname{atan}(X)\arccos(Y^2)$	[-5; 5]	[-5; 5]	
22	$z=X^2+\arccos(Y)$	[-1; 1]	[-1; 4]	
23	$z=\frac{X^4}{Y^2}$	[-5; 5]	[0,1; 1]	
24	$z=X^2+Y^6$	[-1; 1]	[-1; 1]	

Контрольные вопросы

1. Вывод в «Command Window».
2. Ввод в «Command Window».
3. Сохранение данных в файле формата Microsoft Excel.
4. Сохранение данных в текстовом файле.
5. Ввод данных в рабочую область Matlab из дискового файла.
6. Построение трехмерных графиков функций от двух переменных.
7. Ввод и вывод данных в окне «Command Window».
8. Файловый вывод данных.
9. Чтение данных из файлов формата Microsoft Excel.
10. Запись данных в файлы формата Microsoft Excel.
11. Всплывающее меню.
12. Диалоговое окно для ввода данных.
13. Диалоговое окно для выбора из списка.
14. Построение графиков отрезками прямых.
15. Построение трехмерных графиков поверхностей.
16. Блок Simulink From File.
17. Блок Simulink To File.
18. Блок Simulink From Workspace.
19. Блок Simulink To Workspace.
20. Блок Simulink Display.
21. Блок Simulink Scope.
22. Блок Simulink XY Graph.

Лабораторная работа № 4
ПРОГРАММИРОВАНИЕ С ПОМОЩЬЮ ЯЗЫКА ВЫСОКОГО УРОВНЯ MATLAB

Цель работы: получение практических навыков, необходимых при программировании с помощью языка высокого уровня Matlab.

Необходимо освоить программирование циклов **while (условное выражение)…end** и **for…end** и управление ходом воспроизведения программы с помощью операторов ветвления **switch… case … otherwise…end** и **if…elseif…else**.

Логическая структура любого алгоритма может быть представлена комбинацией трех логических структур:

– следование;

– разветвление;

– цикл.

Характерной особенностью этих трех структур является наличие в них одного входа и одного выхода.

Принцип следования. В процессе выполнения алгоритма производятся различные преобразования информации, осуществляемые заданной последовательностью операторов. Величины, которые при выполнении алгоритма изменяют свои значения, называются переменными.

Порядок выполнения операторов в алгоритме должен отвечать принципу следования и принципу обеспеченности переменных, в основе которых лежит обеспеченность значений переменных на каждом шаге выполнения алгоритма.

4.1. Разветвление

Эта структура обеспечивает в зависимости от результата проверки условия (истина или ложь) выбор одного из альтернативных путей работы алгоритма. Каждый из путей ведет к общему выходу, так что работа программы продолжается независимо от того, какой путь выбран. Возможные пути выполнения алгоритма помечаются соответствующими метками: **истина/ложь, да/нет, 1/0**. Для организации условных выражений используются **4** оператора: **if, else, elseif, end**, операторы отношения и логические операторы.

Операторы отношения

В Matlab операторы отношения могут возвращать одно из двух значений: **1** или **0**. Применяются следующие операторы отношения:

– оператор "= =", например, выражение **x= =y** возвращает значение **1**, если **x** и **y** имеют одинаковые значения, иначе – **0**, в случае если **x** и **y** являются массивами (обязательно одинаковой размерности), то возвращается массив той же размерности, в котором на месте элементов, которые равны между собой, стоят единицы, остальные элементы массива равны нулю. Соответствующая функция **eq(x,y)**;

– оператор "~=", например, выражение **x~=y** возвращает значение **1**, если x и **y** имеют различные значения, иначе – **0**, в случае если **x** и **y** являются массивами (обязательно одинаковой размерности), то возвращается массив той же размерности, в котором на месте элементов, которые неравны между собой, стоят единицы, остальные элементы массива равны нулю. Соответствующая функция **ne(x,y)**;

– оператор "<", например, выражение **x<y** возвращает значение **1**, если x меньше **y**, иначе – **0**, в случае если **x** и **y** являются массивами (обязательно одинаковой размерности), то возвращается массив той же размерности, в котором на месте элементов, для которых выполняется условие **x(i)<y(i)**, стоят единицы, остальные элементы массива равны нулю. Соответствующая функция **lt(x,y)**;

– оператор ">", например, выражение **x>y** возвращает значение **1**, если **x** больше **y**, иначе – **0**, в случае если **x** и **y** являются массивами (обязательно одинаковой размерности), то возвращается массив той же размерности, в котором на месте элементов, для которых выполняется условие **x(i)>y(i)**, стоят единицы, остальные элементы массива равны нулю. Соответствующая функция **gt(x,y)**;

– оператор "<=", например, выражение **x<=y** возвращает значение **1**, если **x** меньше или равно **y**, иначе – **0**, в случае если **x** и **y** являются массивами (обязательно одинаковой размерности), то возвращается массив той же размерности, в котором на месте элементов, для которых выполняется условие **x(i)<=y(i)**, стоят единицы, остальные элементы массива равны нулю. Соответствующая функция **le(x,y)**;

– оператор ">=", например, выражение **x>=y** возвращает значение **1**, если **x** больше или равно **y**, иначе – **0**, в случае если **x** и **y** являются массивами (обязательно одинаковой размерности), то возвращается массив той же размерности, в котором на месте элементов, для которых вы-

полняется условие **x(i)>=y(i)**, стоят единицы, остальные элементы массива равны нулю. Соответствующая функция **ge(x,y)**.

Логические операторы

Некоторые логические операторы и функции Matlab:

– оператор "**&**", например, **(x>2)&(y<0)** возвращает значение **1**, если оба выражения (слева и справа от **&**) равны **1**, иначе – **0,** соответствующая функция **and((x>2),(y<0))**;

– оператор "|", например, **(x>2)|(y<0)** возвращает значение **1**, если хотя бы выражения (слева и справа от |) равны **1**, иначе – **0,** соответствующая функция **or((x>2),(y<0))**;

– оператор "**~**", например, **~(x>2)** изменяет значение выражения на противоположное, т. е., если **(x>2)=0**, то **~(x>2)=1** и наоборот, соответствующая функция **not(x>2)**.

*Оператор **if***

Оператор условия **if…end** вычисляет некоторое логическое выражение и выполняет соответствующую группу инструкций в зависимости от значения этого выражения. Если логическое выражение истинно, то Matlab выполняет все инструкции между **if** и **end**, а затем продолжает выполнение программы в строке после **end**. Если условие ложно, то Matlab пропускает все инструкции между **if** и **end** и продолжает выполнение программы со строки, следующей после ключевого слова **end**. Элемент алгоритма в виде блок-схемы, соответствующий конструкции **if…end**, смотри на рисунке 4.1.

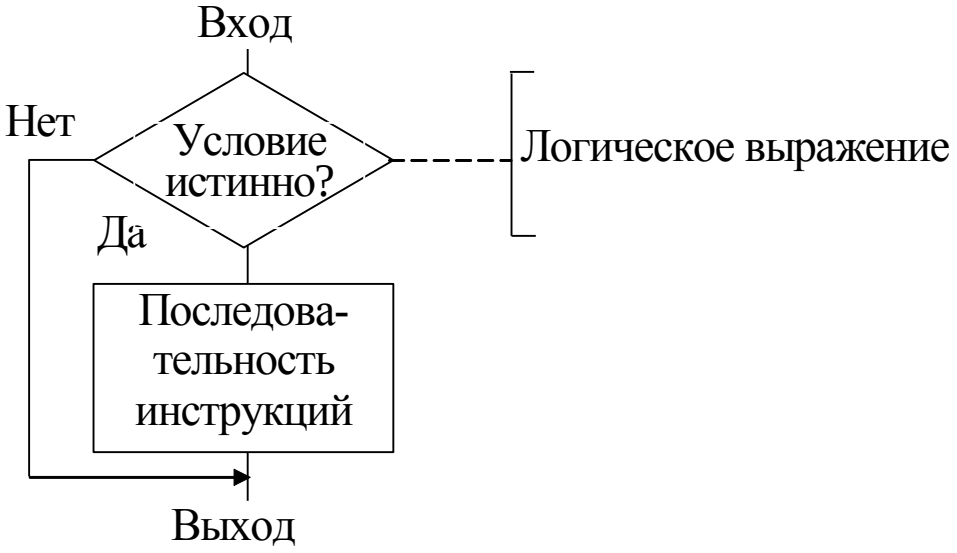


Рисунок 4.1 – Блок-схема конструкции if…end

При составлении программы с помощью языка Matlab синтаксис данной конструкции: **if** оператор сравнения.

Последовательность инструкций **end.**

Приведем пример. Необходимо вычислить натуральный логарифм числа, введенного оператором с клавиатуры, при этом необходимо проверить, не ввел ли оператор значение, меньшее или равное 0, так как в этом случае логарифм вычислить нельзя, но результат, который мы хотим получить, не должен превышать нуля, поэтому будем проверять, не ввел ли оператор значение больше единицы. Алгоритм этого вычисления смотри на рисунке 4.2.

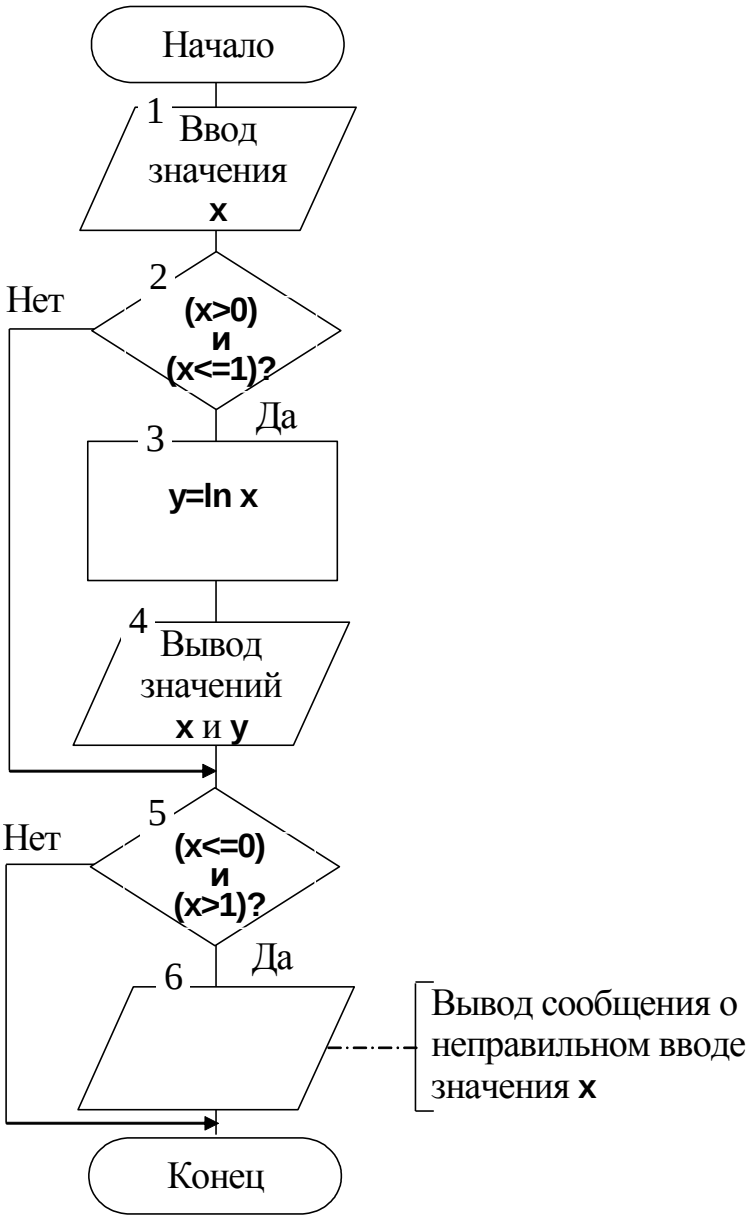


Рисунок 4.2 – Вычисление натурального логарифма числа, введенного оператором с клавиатуры

Текст программы:

```
x=input('Ввести значение аргумента')
if (x>0)&(x<=1)
   y=log(x);
  disp('x=')
  disp(x)
  disp('y=')
  disp(y)
end
if (x<=0)&(x>1)
   disp('Введенное значение должно быть больше нуля')
   disp('и не должно превышать значения 1')
end
```

Если оператор введет **0** или **-1** или другое значение **x**, при котором оператор **x>0** примет значение **0**, а также значение **x**, превышающее **1**, тогда оператор **x<=1** примет значение **0**, выражение **y=log(x)** не будет вычислено.

*Операторы **if…else…end** и **if…elseif…end***

Операторы **if…else…end** и **if…elseif…end** создают дополнительные ветвления внутри тела оператора **if**:

Оператор **else** не содержит логического условия. Инструкции, связанные с ним, выполняются, если предшествующий оператор **if** (и возможно **elseif**) ложны. Элемент алгоритма в виде блок-схемы, соответствующий конструкции **if…else…end**, смотри на рисунке 4.3.

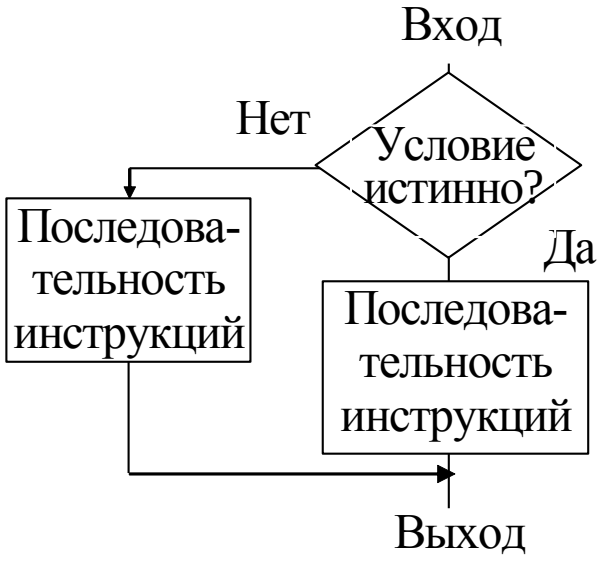


Рисунок 4.3 – Блок-схема, соответствующая конструкции if…else…end

В программе в общем виде данная конструкция: **if** оператор сравнения

Последовательность инструкций **else**

Последовательность инструкций **end**

Изменим нашу программу так, чтобы при верном вводе оператором аргумента программа вычисляла необходимую функцию и выводила результат, а при неправильном вводе – сообщала об ошибке ввода (рис. 4.4).

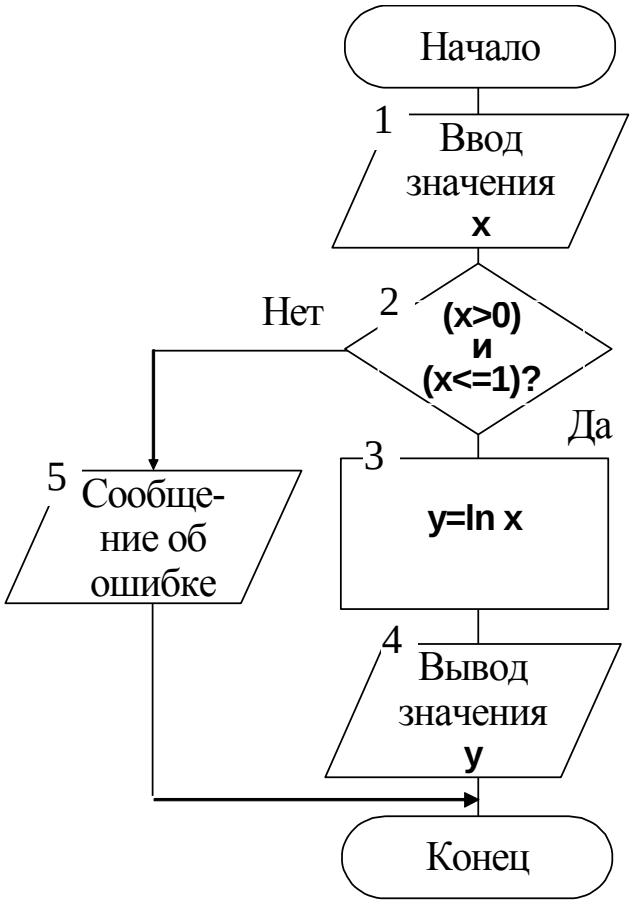


Рисунок 4.4 – Программа с возможностью при неправильном вводе сообщать об ошибке

Текст программы:

```
x=input('Ввести значение аргумента')
if (x>0)&(x<=1)
   y=log(x)
   disp('y=')
   disp(y)
else
   disp('Ошибочный ввод')
end
```

*Оператор **if…elseif…else…end***

Оператор **elseif** содержит логическое условие, которое выполняется, если предшествующий оператор **if** (и возможно **elseif**) ложны. Инструкции, связанные с оператором **elseif**, выполняются, если соответствующее условие истинно. Элемент алгоритма в виде блок-схемы, соответствующий конструкции **if…elseif…else…end**, смотри на рисунке 4.5.

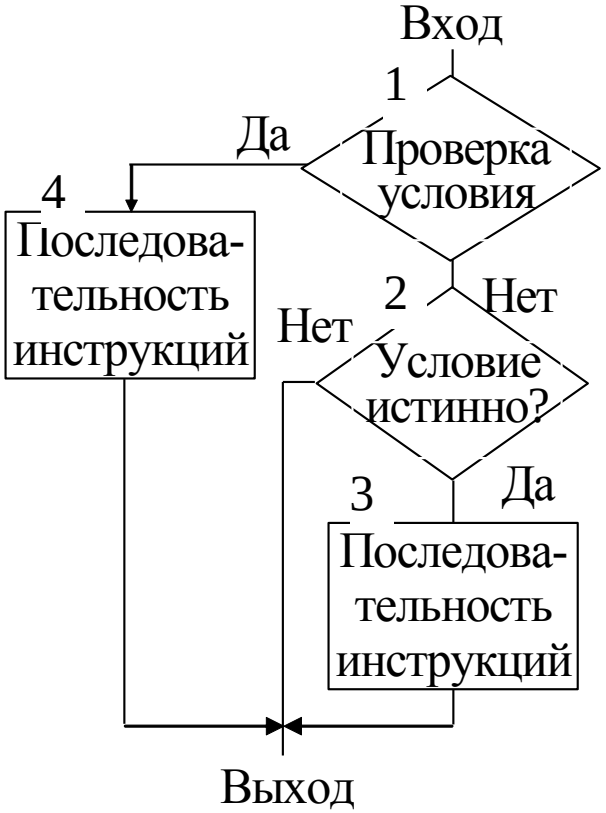


Рисунок 4.5 – Блок-схема,
соответствующая конструкции if…elseif…else…end

Оператор **elseif** может многократно использоваться внутри оператора условия **if**.

В программе в общем виде данная конструкция:

if оператор сравнения

Последовательность инструкций **elseif** оператор сравнения

Последовательность инструкций… **elseif** оператор сравнения

Последовательность инструкций **else**

Последовательность инструкций **end.**

С помощью такой конструкции легко организовать анализ выбора кнопки всплывающего меню:

```
k=menu('Выбрать вариант задания',…
'Задание по математике', 'Задание по физике',…
'Задание по химии', 'Не выбирать')
if k==1
   disp('Получите задание по математике')
elseif k==2
   disp('Получите задание по физике')
elseif k==3
   disp('Получите задание по химии')
else
   disp('Задание не выбрано')
end
```

Оператор переключения ***Switch***

Конструкция с переключателем **Switch…case…otherwise…end** используется для осуществления множественного выбора (или ветвления), в зависимости от значений некоторой переменной или выражения.

Оператор переключения содержит:

– заголовок **switch**, за которым следует вычисляемое выражение (скаляр или строка);

– произвольное количество групп **case**. Заголовки групп состоят из слова **case**, за которым следует возможное значение выражения, расположенное в той же строке. Последующие строки содержат инструкции, которые выполняются для данного значения выражения. Выполнение продолжается до тех пор, пока не встретится следующий оператор **case**, или оператор **otherwise**, или оператор **end**. На этом выполнение блока **switch** завершается;

– группу **otherwise** (не обязательно). Заголовок включает только слово **otherwise**. Начиная со следующей строки размещаются инструкции, которые выполняются, если значение выражения оказалось не обработанным ни одной из групп **case**. Выполнение завершает оператор **end**. Оператор **end** является последним в блоке переключателя.

Элемент алгоритма в виде блок-схемы, соответствующий конструкции **switch**, смотри на рисунке 4.6.

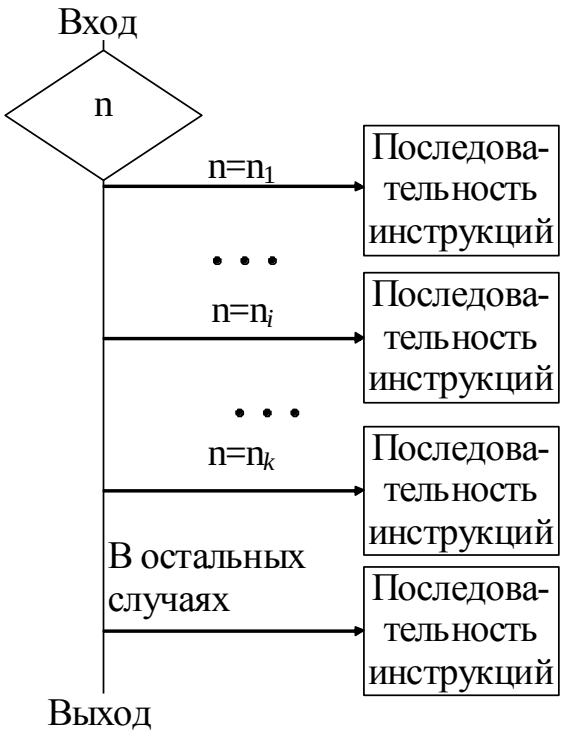


Рисунок 4.6 – Блок-схема, соответствующая конструкции switch

В программе в общем виде данная конструкция:

switch n
case n_1
Последовательность инструкций……..**case n_i**
Последовательность инструкций……**case n_k**
Последовательность инструкций **otherwise**
Последовательность инструкций **end.**

Переменная **n** должна быть скаляром или строкой.

С помощью такой конструкции также можно организовать анализ выбора кнопки всплывающего меню:

```
k=menu('Выбрать вариант задания',…
'Задание по математике', 'Задание по физике',…
'Задание по химии', 'Не выбирать')
switch k
   case 1
      disp('Получите задание по математике')
   case 2
      disp('Получите задание по физике')
   case 3
      disp('Получите задание по химии')
   otherwise
      disp('Задание не выбрано')
end
```

Используя вложенные выбирающие инструкции, преобразуем программу так, чтобы сначала выводилось сообщение о том, выбрано или нет задание, а потом, если задание выбрано, то приглашение его получить.

```
k=menu('Выбрать вариант задания',…
'Задание по математике', 'Задание по физике',…
'Задание по химии', 'Не выбирать')
if (k==1)|(k==2)|(k==3)
   disp('Выбрано задание')
   switch k
   case 1
      disp('Получите задание по математике')
   case 2
      disp('Получите задание по физике')
   case 3
      disp('Получите задание по химии')
   end
else
   disp('Задание не выбрано')
end
```

Теперь, если оператор выберет первую кнопку, получит сообщение:

Выбрано задание

Получите задание по математике

Теперь, если оператор выберет вторую кнопку, получит сообщение:

Выбрано задание

Получите задание по физике

Теперь, если оператор выберет третью кнопку, получит сообщение:

Выбрано задание

Получите задание по химии

Теперь, если оператор выберет четвертую кнопку, получит сообщение:

Задание не выбрано

4.2. Повторение, или цикл

Базовая структура повторение, или цикл, обеспечивает повторное выполнение или, другими словами, циклическую работу операторов. Оператор или группа операторов, повторяющихся в цикле, называется телом цикла.

Цикл с заранее заданным числом итераций

Цикл типа **for…end** используют для организации вычислений с заданным числом повторяющихся итераций.

Элемент алгоритма в виде блок-схемы, соответствующий циклу **for…end**, смотри на рисунке 4.7.

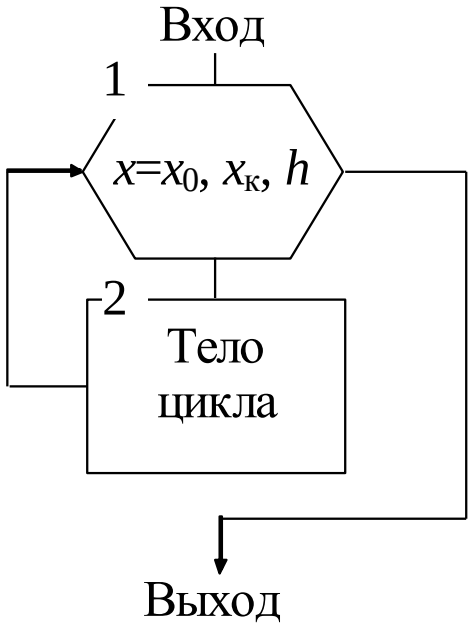


Рисунок 4.7 – Блок-схема, соответствующая циклу for…end

В программе в общем виде данная конструкция:

for $\mathbf{k=k_1:\Delta:k_n}$

инструкции **end**

где **k** – переменная цикла; $\mathbf{k_1}$ – начальное значение переменной цикла; **Δ** – приращение; $\mathbf{k_n}$ – конечное значение переменной цикла. По умолчанию приращение=**1.**

Вычислим число Фибоначчи: $\mathbf{F_N=F_{N-1}+F_{N-2}}$, при $\mathbf{N\geq 2}$, $\mathbf{F_0=F_1=1}$.

Числа Фибоначчи от нулевого до пятнадцатого: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987 (рис. 4.8).

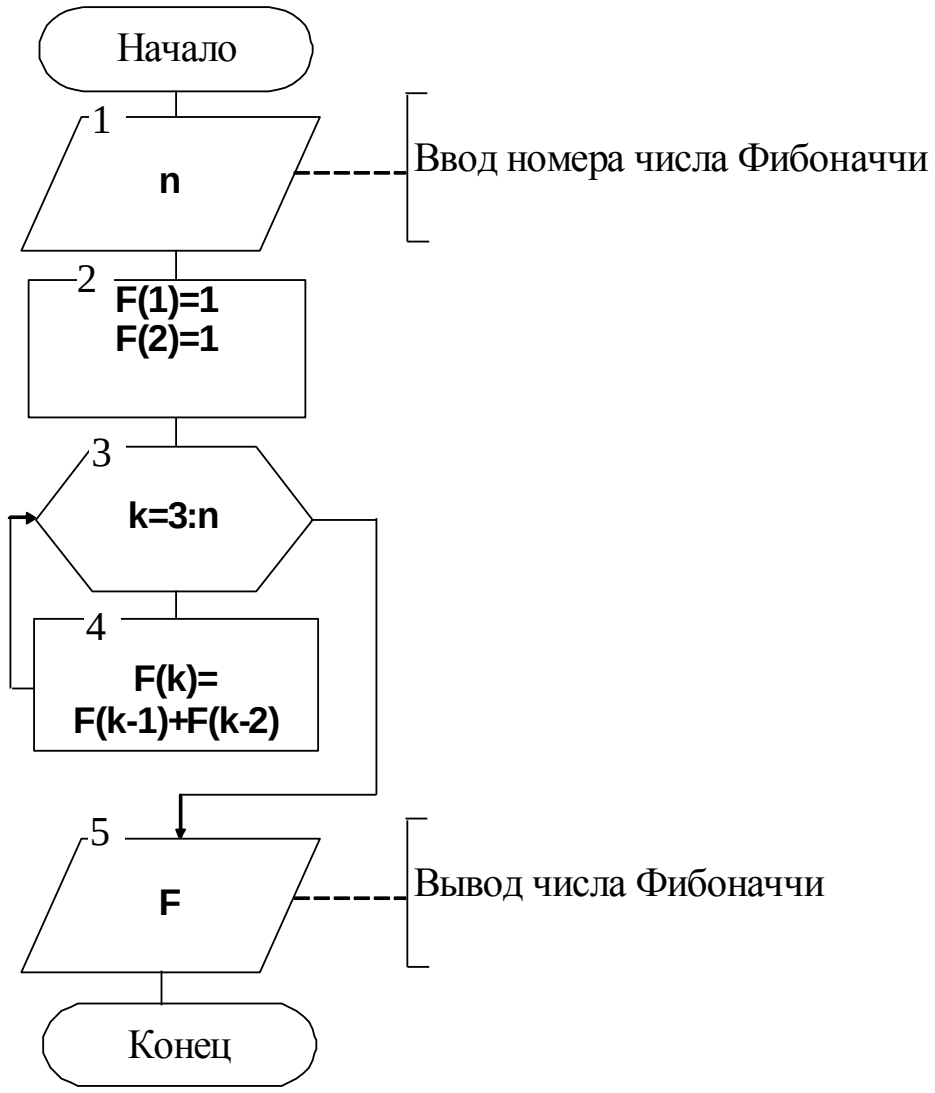


Рисунок 4.8 – Программа с возможностью вывода числа Фибоначчи от нулевого до пятнадцатого

```
n=input('Ввести номер числа Фибоначчи')
F(1)=1;
F(2)=2;
for k=3:n
    F(k)=F(k-1)+F(k-2)
end
disp('Число Фибоначчи номер')
disp(n)
disp('равно')
disp(F(n))
```

В данном примере начальное значение переменой цикла **k** равно **3**, ее конечное значение – **n**, приращение равно **1**.

Пусть массив из **n** чисел Фибоначчи сформирован, теперь сложим все четные числа этого массива. Для этого организуем цикл с

начальным значением переменой цикла **k**, равным **2**, ее конечным значением – **n**, приращением, равным **2** (рис. 4.9).

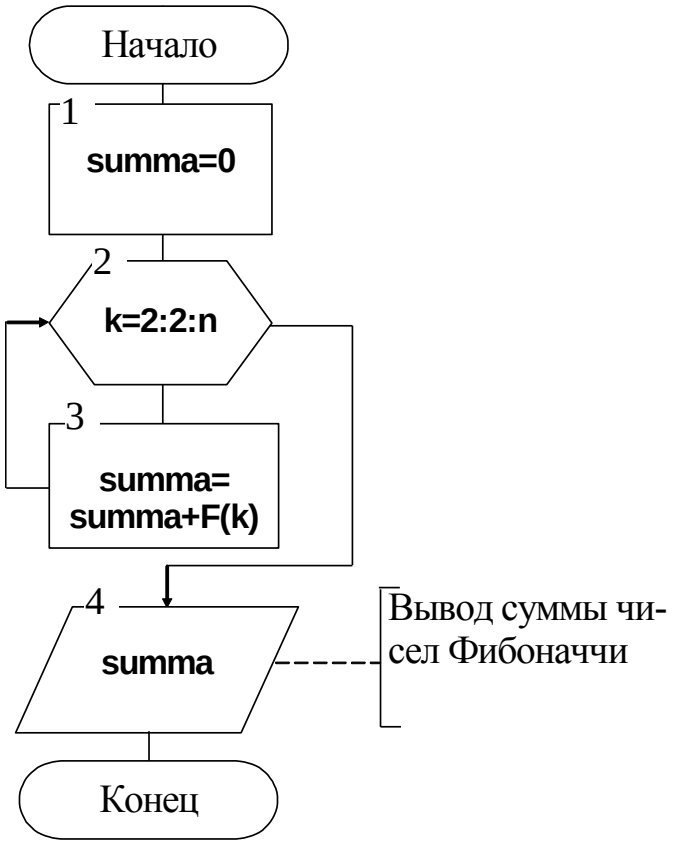


Рисунок 4.9 – Цикл с начальным значением переменой цикла k, равным 2, ее конечным значением – n, приращением, равным 2

```
summa=0
for k=2:2:n
   summa=summa+F(k)
end
disp('Сумма')
disp(n)
disp('четных чисел Фибоначчи')
disp('равна')
disp(summa)
```

В следующем примере оператор выбирает номера чисел Фибоначчи, сумму которых необходимо вычислить.

```
vec=input('Ввести номера чисел для вычисления их суммы')
summa=0
for k=vec
   summa=summa+F(k);
end
```

disp('Сумма')
disp(n)
disp('четных чисел Фибоначчи')
disp('равна')
disp(summa)

Оператор введет вектор **vec**, который содержит номера элементов массива чисел Фибоначчи. Оператор цикла **k** будет последовательно принимать значения элементов вектора **vec**. Элементы **vec** вектора должны удовлетворять условию: $\mathbf{vec_i \leq vec_{i-1}}$ для всех значений **i** или $\mathbf{vec_i \geq vec_{i-1}}$ для всех значений **i.**

Рассмотрим пример, в котором начальное, конечное значение и шаг изменения оператора цикла не являются целыми числами, значение оператора цикла не возрастает, а убывает на каждой следующей итерации цикла. Необходимо сформировать вектор из синусов чисел, изменяющихся от **6.1** до **0.1** с шагом **-0.05** (рис. 4.10).

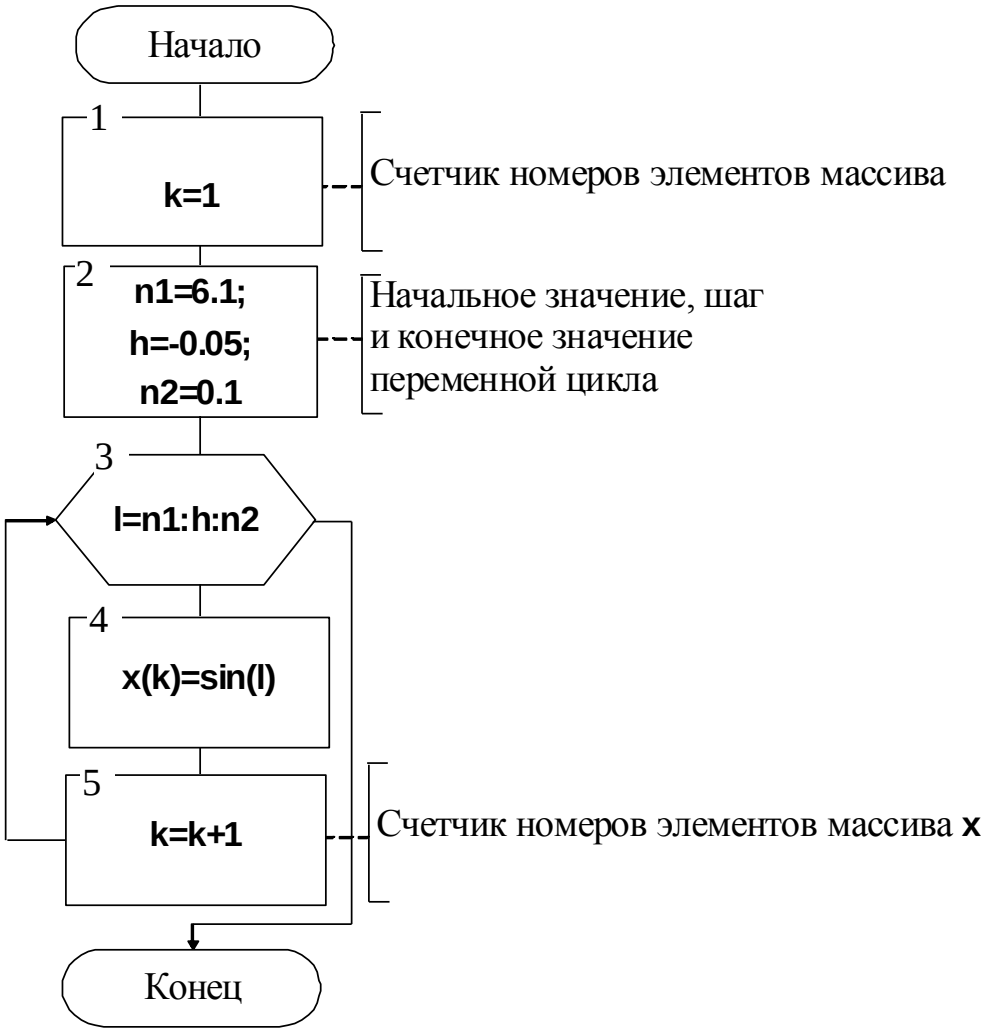


Рисунок 4.10 – Формирование вектора из синусов чисел, изменяющихся от 6.1 до 0.1 с шагом -0.05

n1=6.1;
h=-0.05;
n2=0.1
k=1
for l=n1:h:n2
x(k)=sin(l);
k=k+1;
end

Оператор ***while***

Оператор цикла **while…end** обеспечивает выполнение группы инструкций неопределенное число раз в соответствии с некоторым логическим условием завершения. Элемент алгоритма в виде блок-схемы, соответствующий циклу **while…end**, смотри на рисунке 4.11.

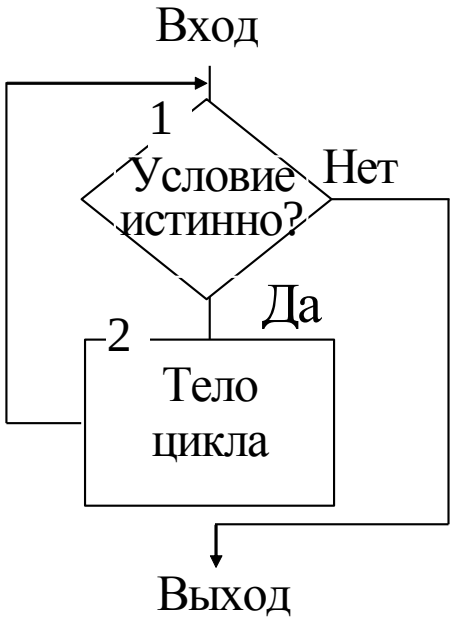


Рисунок 4.11 – Блок-схема, соответствующая циклу while…end

В программе в общем виде данная конструкция:

while логическое выражение

тело цикла

end

Выполнение операторов тела цикла повторяется до тех пор, пока логическое выражение *равно истине*.

В предыдущем примере решить задачу можно также с помощью оператора цикла **while** (рис. 4.12).

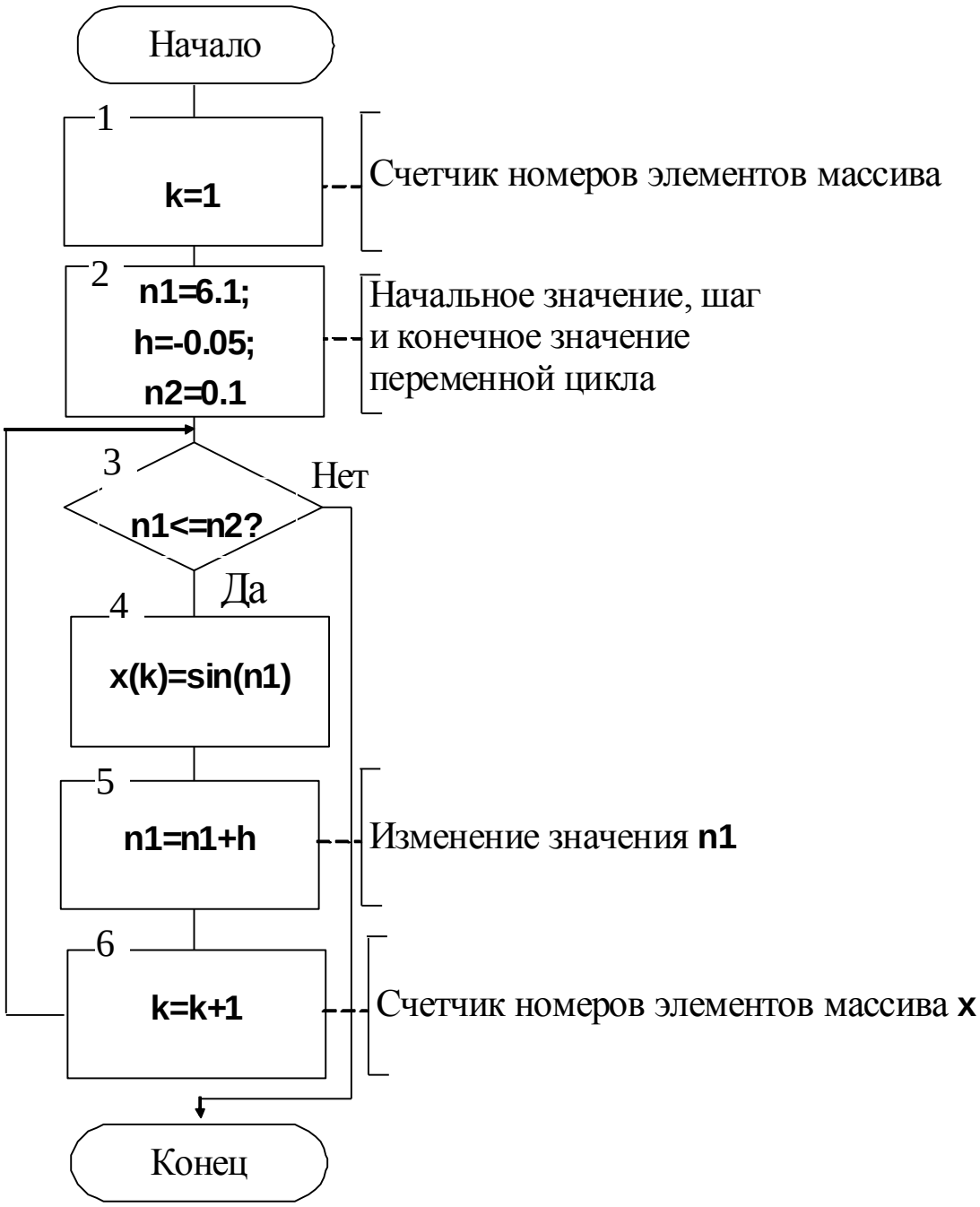


Рисунок 4.12 – Решение задачи с помощью оператора цикла while

```
k=1
n1=6.1;
h=-0.05;
n2=0.1
while n1<=n2
   x(k)=sin(l);
   k=k+1;
   n1=n1+h;
end
```

Оператор ***break***

Функция **break** прерывает выполнение циклов **for** и **while**. В случае вложенных циклов прерывание возможно только из самого внутреннего цикла.

Изменим программу анализа выбора кнопки всплывающего меню так, чтобы меню работало до тех пор, пока оператор не нажмет кнопку с надписью «*Не выбирать*», этот выбор приведет к окончанию работы. Организуем бесконечный цикл, используя логическое выражение, которое всегда равно «*истина*», а при анализе значения переменной **k** предусмотрим прерывание цикла с помощью оператора **break**, если **k** примет значение, не равное **1, 2** или **3**:

```
n=1
while n==1
    k=menu('Выбрать вариант задания', …
    'Задание по математике', …
    'Задание по физике',…
    'Задание по химии',…
    'Не выбирать')
    if k==1
       disp('Получите задание по математике')
    elseif k==2
       disp('Получите задание по физике')
    elseif k==3
       disp('Получите задание по химии')
    else
       disp('Задание не выбрано')
       break
   end
end
disp('Конец работы')
```

Оператор ***continue***

Оператор **continue** передает управление на следующую итерацию цикла, пропуская следующие за ним инструкции в теле цикла.

Например, необходимо найти сумму отрицательных элементов массива и сумму положительных элементов массива (рис. 4.13).

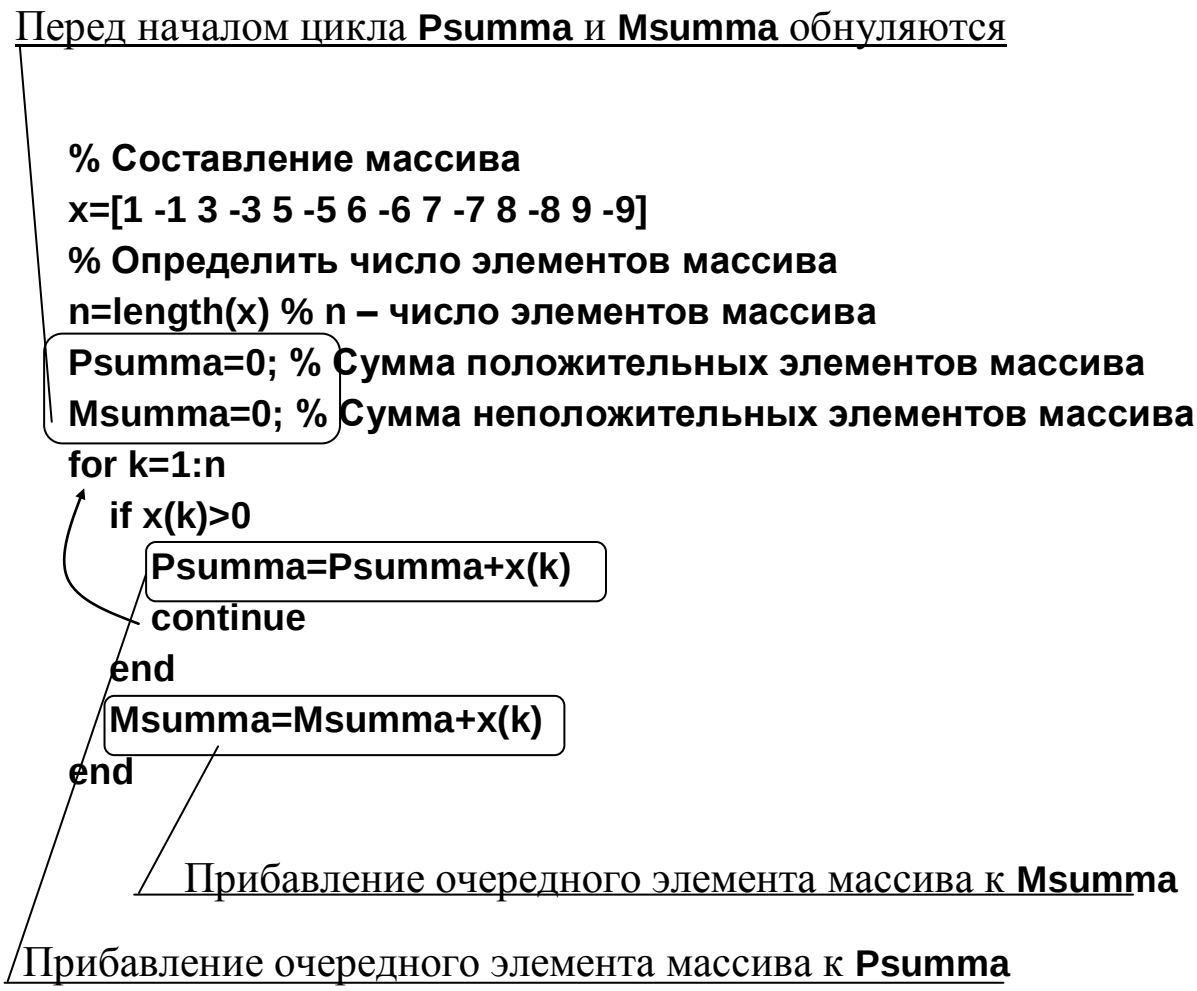


Рисунок 4.13 – Нахождение суммы отрицательных и положительных элементов массива

В программе при выполнении очередной итерации цикла в случае, если встретится положительный элемент массива, он прибавится к сумме **psumma**, после чего оператор **continue** передаст управление программой в начало цикла, т. е. оператору **for**. Если элемент массива не положительный, то он прибавится к сумме **msumma**.

Пример. Вычисление минимального значения функции методом равномерного поиска

Вычислить минимальное значение функции $\mathbf{y=x^4-14x^3+60x^2-70x}$ в интервале от **5** до **7** методом равномерного поиска. Рассмотрим алгоритм (рис. 4.14).

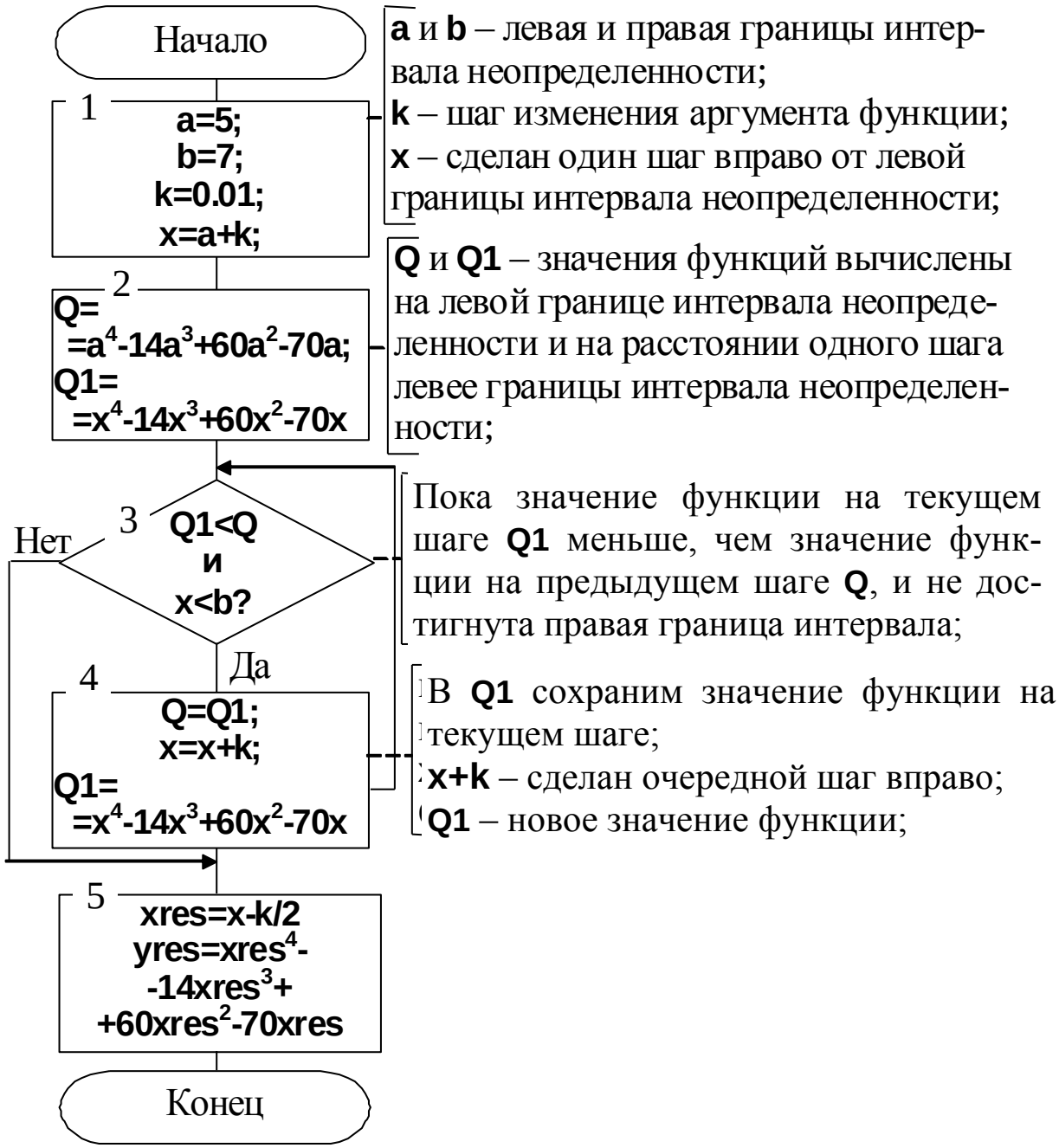


Рисунок 4.14 – Алгоритм метода равномерного поиска

Функция $y=x^4-14x^3+60x^2-70x$ в интервале от **5** до **7** имеет минимум. В начале поиска интервал неопределенности равен всему интервалу от **5** до **7**. Находим значение аргумента на расстоянии одного шага изменения аргумента **x=a+k** (блок **1**). Вычислим значение функции **Q** и **Q1** с аргументами **a** и **x** соответственно (блок **2**). В блоке **3** значения функции в двух точках сравниваются, если значение функции в точке **x** меньше, чем в точке **a**, то это значит, что при первом шаге минимальное значение функции не достигнуто, поэтому (см. блок 4), сохранив значение **Q1** в **Q**, увеличим аргумент еще на

один шаг: **x=x+k** и вычислим значение функции в текущей точке **x**: **Q1=x^4-14x^3+60x^2-70x**. Затем управление опять передается в блок **3**, и до тех пор, пока значение функции **Q** на текущем шаге будет меньше, чем значение функции **Q1** на предыдущем шаге, будет повторяться процесс в блоке **4**, так как это значит, что минимальная точка не достигнута. Если результат сравнения в блоке **3** окажется равным «**нет**», то управление перейдет к блоку **5**, где будет вычислена середина интервала между последним и предпоследним значениями аргумента **x** и вычислено значение функции в этой точке. Это минимальное значение функции в заданном интервале поиска.

Текст программы, реализующей приведенный алгоритм:

```
a=5;
b=7;
k=0.01;
x=a+k;
Q=a^4-14*a^3+60*a^2-70*a;
Q1=x^4-14*x^3+60*x^2-70*x;
while Q1<Q;
   Q=Q1;
   x=x+k;
   Q1=x^4-14*x^3+60*x^2-70*x;
end
xres=x-k/2
yres=xres^4-14*xres^3+60*xres^2-70*xres
```

В данном алгоритме ради простоты и наглядности опущен контроль правой границы интервала поиска. Устраним этот недостаток, введем проверку: не превышает ли значение текущей координаты значения «право» границы интервала неопределенности:

```
a=5;
b=7;
k=0.01;
x=a+k;
Q=a^4-14*a^3+60*a^2-70*a;
Q1=x^4-14*x^3+60*x^2-70*x;
while Q1<Q;
   Q=Q1;
   x=x+k;
   if x>b % перешагнули правую границу
```

```
        disp('Дальнейший поиск невозможен')
        k=0
        break % Прерывание цикла
    end
    Q1=x^4-14*x^3+60*x^2-70*x;
end
xres=x-k/2
yres=xres^4-14*xres^3+60*xres^2-70*xres
```

Тоже сделаем с помощью цикла **for**:

```
a=5;
b=5.7;
k=0.01;
Q=a^4-14*a^3+60*a^2-70*a;
for x=a+k:k:b
    Q1=x^4-14*x^3+60*x^2-70*x;
    if Q1>=Q
        disp('Переход чрез точку минимума')
        break
    end
    Q=Q1;
    x=x+k;
    if x>b
        disp('Дальнейший поиск невозможен')
        k=0
        break
    end
    Q1=x^4-14*x^3+60*x^2-70*x;
end
xres=x-k/2
yres=xres^4-14*xres^3+60*xres^2-70*xres
```

Вычисление максимального значения функции методом дихотомии

Необходимо вычислить максимальное значение функции $\mathbf{y=x^3-6x^2+9x+4}$ в интервале от **0,2** до **1,5** методом дихотомии (рис. 4.15).

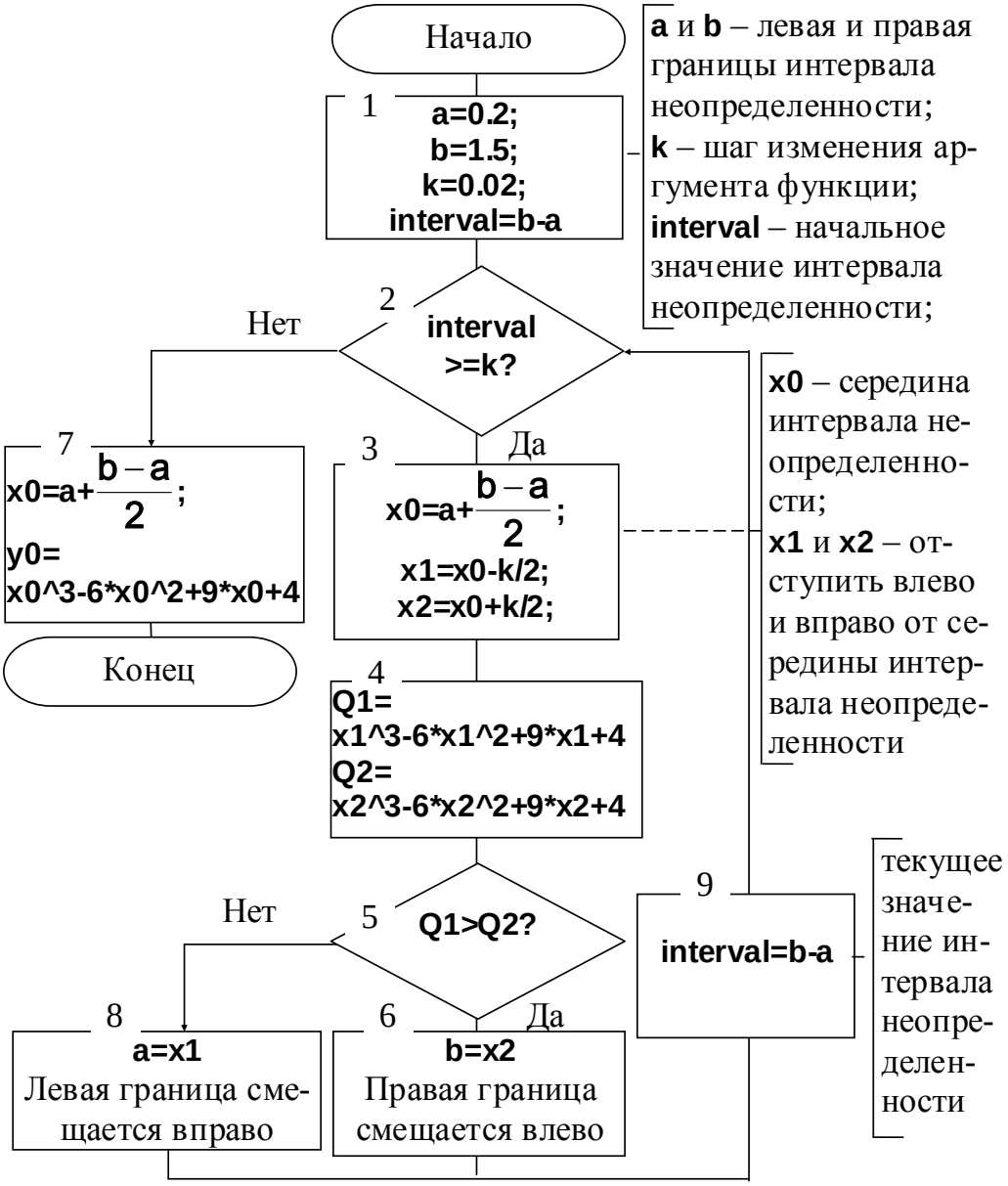


Рисунок 4.15 – Алгоритм метода дихотомии

Работа данного алгоритма начинается с назначения интервала неопределенности равным заданному в задаче интервалу изменения функции от **a** до **b** (блок **1**). Блок **2** предназначен для проверки, не превышает текущий интервал значения заданного приращения аргумента, так как на каждой итерации работы алгоритма интервал неопределенности уменьшается в 2 раза. Если результат блока **2** равен «истина» («да»), то в блоке **3** находится середина интервала неопределенности, затем точки **x1** и **x2**, находящиеся справа и слева на оси абсцисс от середины интервала на расстоянии половины шага изменения функции. В блоке **4** вычисляется значение функции **Q1** и **Q2** в точках **x1** и **x2**. В блоке **5** определяется, больше ли **Q1**, чем **Q2**, если «да», то под интервал, находящийся правее середины (точки **x**) не-

перспективен для поиска максимальной точки, поэтому правая граница **b** интервала неопределенности смещается в точку **x2** (блок **6**). В ином случае неперспективен левый под интервал, тогда левая граница **a** интервала неопределенности смещается в точку **x1** (блок **8**). В любом случае интервала неопределенности уменьшается в 2 раза. В блоке **9** вычисляется длина текущего интервала. На одной из итераций алгоритма проверка в блоке **2** вернет результат «**нет**», тогда в блоке **7** будет вычислено максимальное значение заданной функции в интервале от **a** до **b**.

Текст программы:

```
a=0.2;
b=1.5;
k=0.02;
interval=abs(a-b);
while interval>=k*2;
  x0=a+(b-a)/2;
  x1=x0-k/2;
  x2=x0+k/2;
  Q1=x1.^3-6*x1.^2+9*x1+4;
  Q2=x2.^3-6*x2.^2+9*x2+4;
  if Q1>Q2;
     b=x2;
  else
     a=x1;
  end
  interval=abs(b-a);
end
x0=a+(b-a)/2
y0=x0^3-6*x0^2+9*x0+4
```

4.3. Задание для лабораторной работы

С помощью цикла **while** организуем ввод массива данных. Проверку вводимых данных осуществить с помощью ветвления **switch**. Затем вычислить функцию от всех элементов массива (см. варианты задания в п. 4.4).

Для примера запрограммируем ввод 100 элементов массива.

Сначала инициализируем счетчик элементов **k**, присвоив ему начальное значение, равное 1 (блок 1 блок-схемы на рисунке 4.16).

Затем организуем цикл с предусловием, в теле которого запрограммируем ввод элементов массива с помощью клавиатуры (блок 3 блок-схемы на рис. 4.16) и увеличение счетчика (блок 4 блок-схемы на рис. 4.16), что позволит перейти к вводу следующего элемента массива на следующей итерации цикла, и обеспечит достижение счетчиком **k** значения **>100** и, таким образом, возможность завершения цикла **while k<=100…end**.

В условном выражении в заголовке цикла будет проверяться, не достиг ли счетчик значения 100 (блок 2 блок-схемы на рис. 4.16).

После ввода всего массива вычислим функцию $y = \frac{0.2x^4 \cos|x|}{x-77}$ (блок 5 блок-схемы на рис. 4.16) от всех элементов введенного массива.

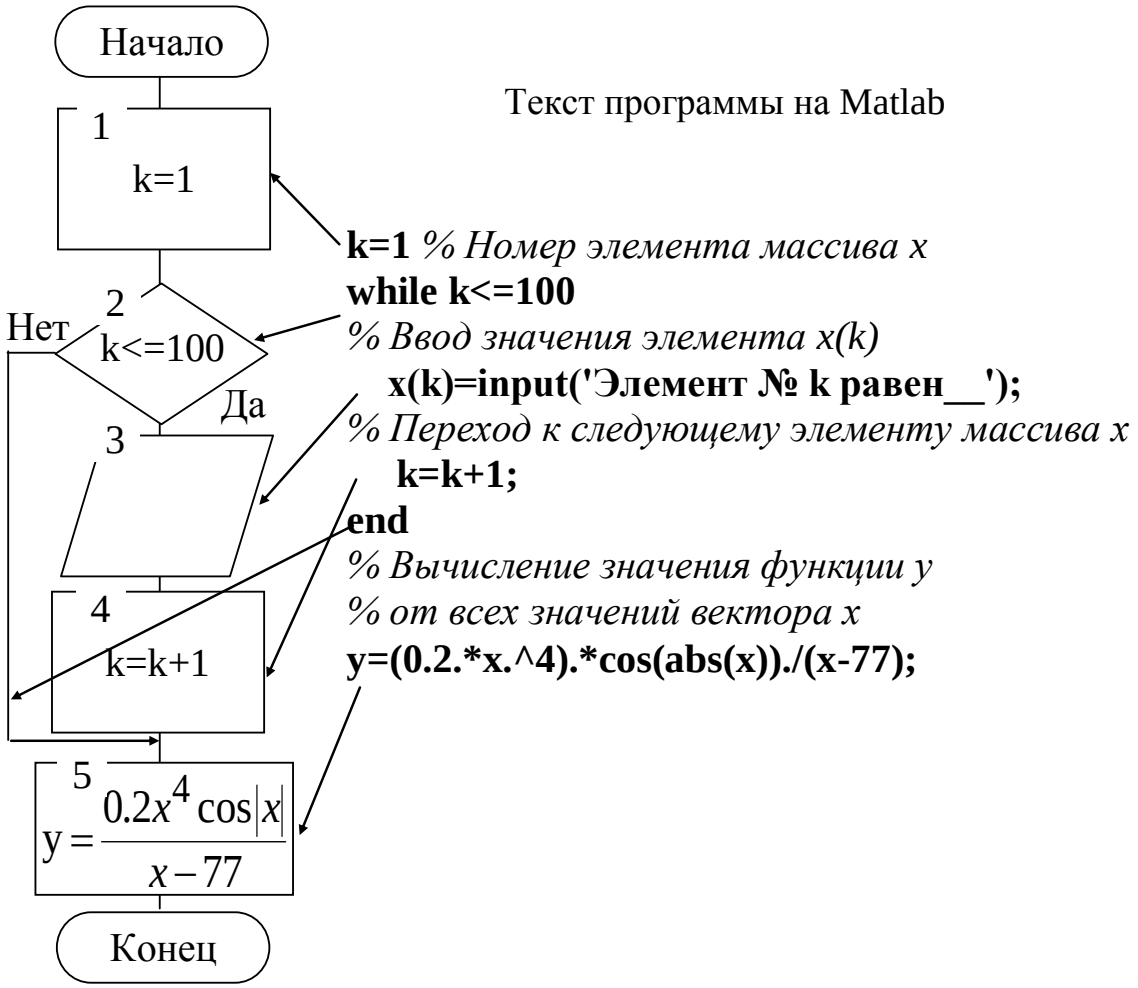


Рисунок 4.16 – Блок-схема применения цикла while и ветвления switch

При вычислении функции недопустимо, чтобы знаменатель принимал значение **0**, поэтому оператор не должен вводить значение **x(k)**

равное **77**. С помощью оператора ветвления **switch** исключим эту ситуацию: при проверке введенных значений, если **x(k)** равно **77**, то программа выведет сообщение «**Получится деление на 0**» и передаст управление на начало цикла с помощью оператора **continue**:

```
switch x(k)
    case 77
       disp('Получится деление на 0')
       continue
end
```

Условимся, что можно ввести массив не из **100**, а из меньшего числа элементов, если оператор введет значение **0**, то ввод прекратится и будет вычислена функция. С этой целью добавим к телу оператора **switch** проверку, если **x(k)** равно нулю, то цикл должен преждевременно завершиться, выведя сообщение «**Преждевременное прекращение ввода**», оператор **break** передаст управление программой на строку, идущую после цикла:

```
switch x(k)
    case 77
       disp('Получится деление на 0')
       continue
    case 0
       disp('Преждевременное прекращение ввода')
       break
end
```

Во всех остальных случаях можно переходить к вводу следующего элемента массива **x**. Поэтому добавим к телу оператора **switch** строки

```
otherwise
        disp('Все в порядке. Продолжение ввода')
% Переход к следующему элементу массива x
        k=k+1;
```

Тогда математическое выражение **k=k+1**, обеспечивающее увеличение счетчика элементов массива и, таким образом, переход к следующему элементу массива на следующей итерации цикла будет выполняться только, если не происходит преждевременное прекращение цикла или переход на начало цикла:

```
switch x(k)
    case 77
       disp('Получится деление на 0')
```

```
        continue
    case 0
        disp('Преждевременное прекращение ввода')
        break
    otherwise
        disp('Все в порядке. Продолжение ввода')
% Переход к следующему элементу массива x
        k=k+1;
end
```

Окончательно получится программа, блок-схема которой представлена на рисунке 4.17, сценарий – на рисунке 4.18.

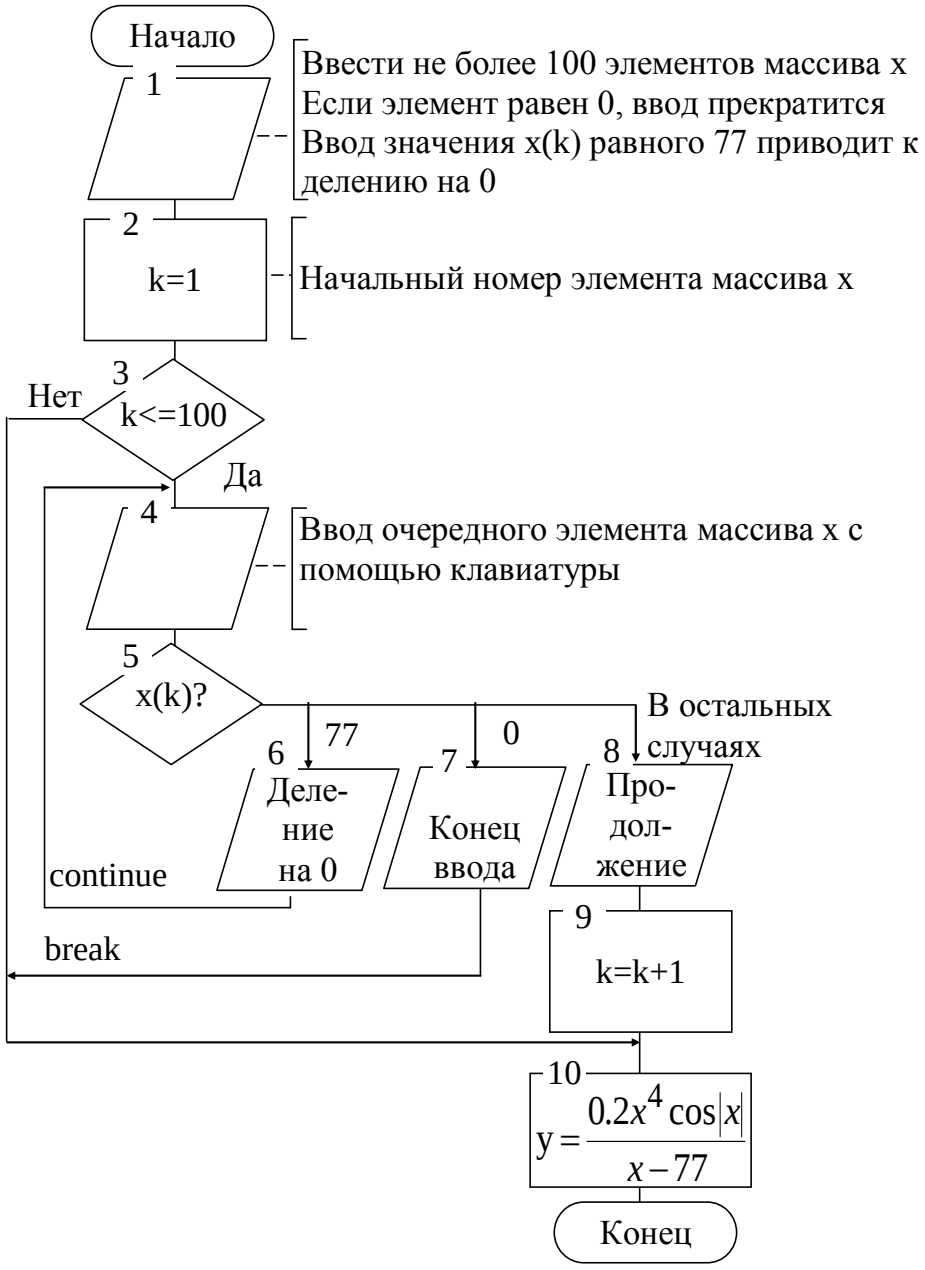


Рисунок 4.17 – Окончательная блок-схема программы

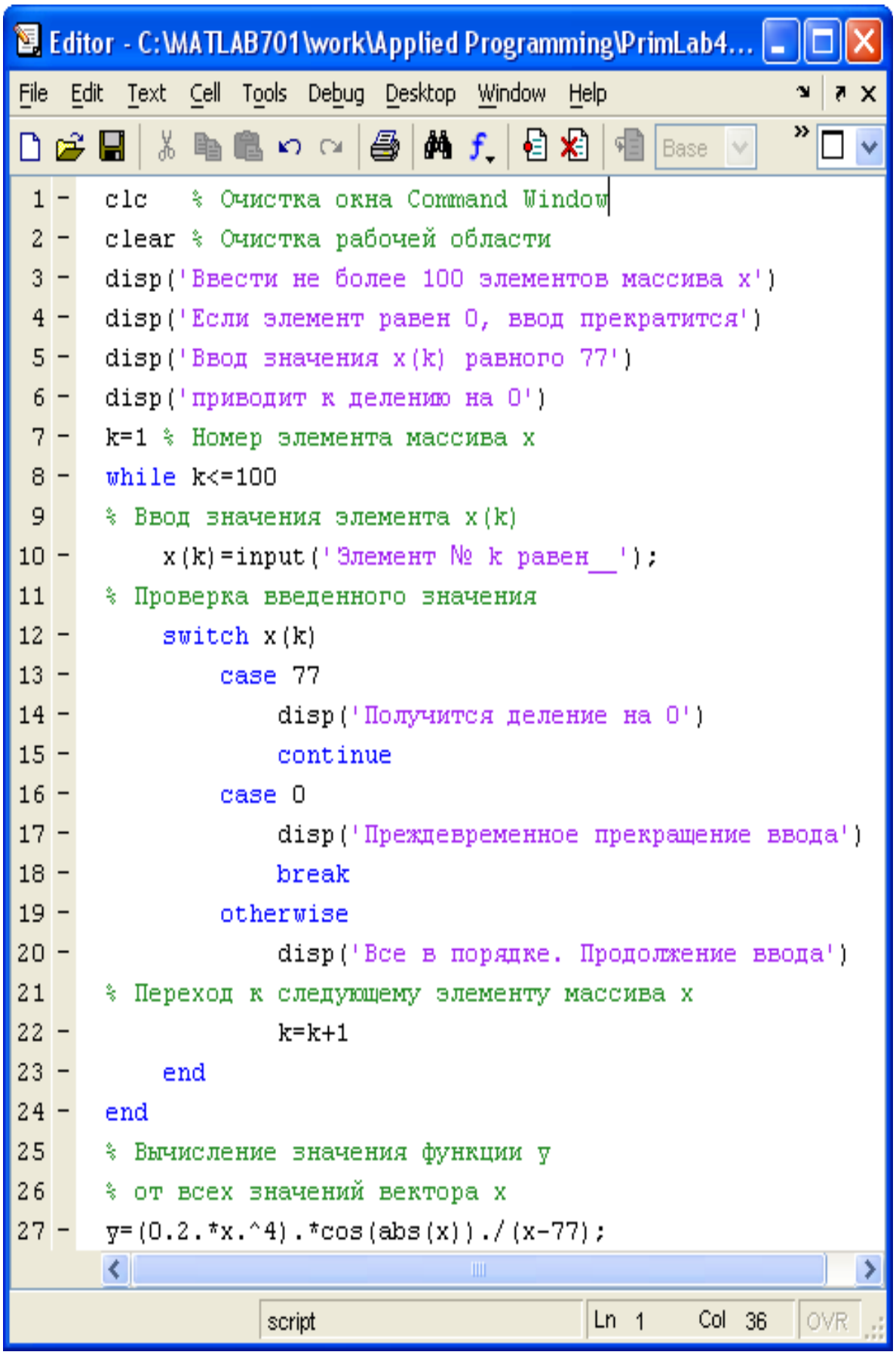


Рисунок 4.18 – Сценарий программы

При выполнении программы сначала будет выведено сообщение, запрограммированное четырьмя функциями **disp**. Затем значение **k**, равное **1**, так как после выражения **k=1** не стоит символ «**;**», подавляющий вывод в окно «Command Windows» результатов вычислений. Затем начнется первая итерация цикла. Первый оператор, находящийся в теле цикла – функция **input** выведет сообщение «*Элемент № k равен__*» и остановит воспроизведение программы до ввода в позиции курсора любого численного значения – значения **x(1)** и нажатия клавиши **Enter** (рис. 4.19).

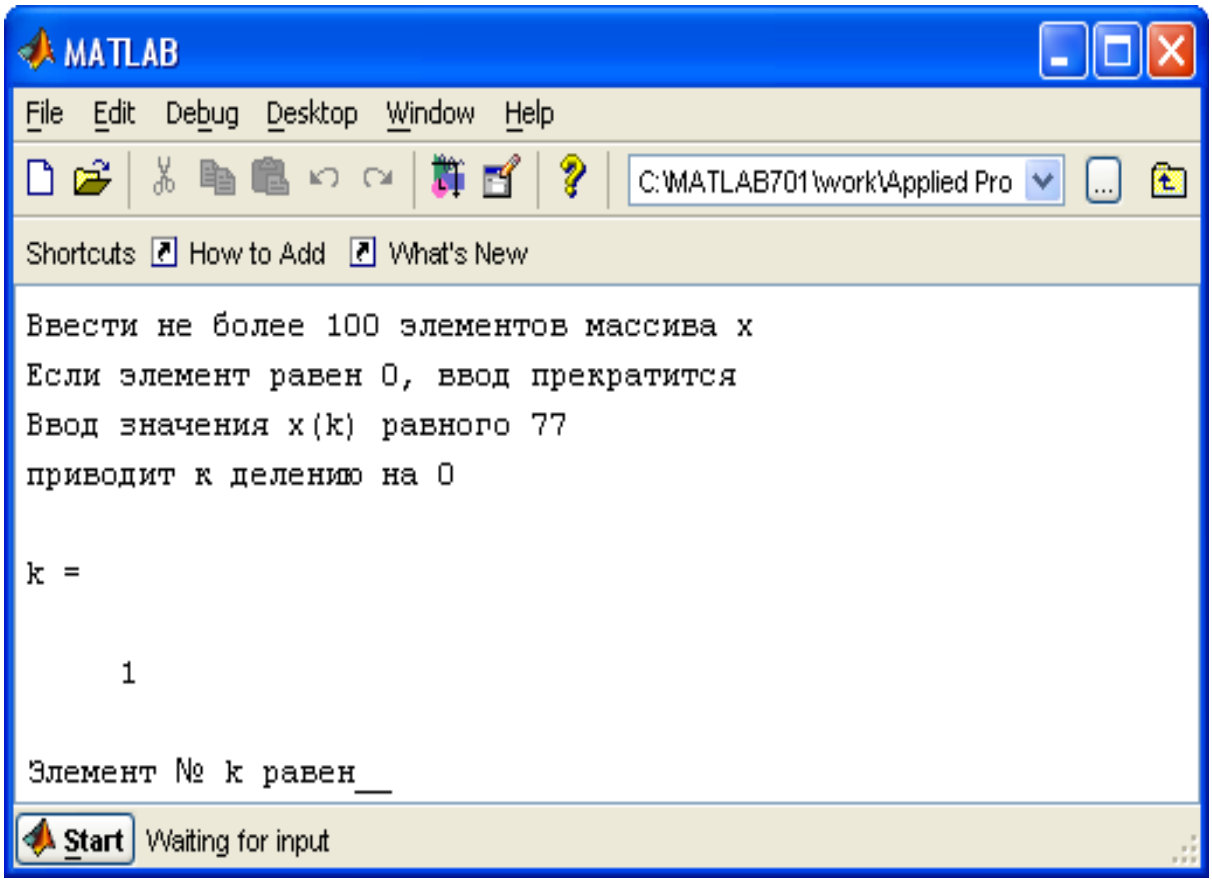


Рисунок 4.19 – Сообщение, запрограммированное четырьмя функциями disp

Следующий за **input** оператор **switch x(k)** сравнит введенное значение **x(k)** с константой **77**, так как первым в теле оператора **switch** встречается строка «case 77». Если данное сравнение вернет результат **истина**, то будут работать операторы, следующие после строки «case 77» до следующей строки «case»: появится сообщение «*Получится деление на 0*», оператор **continue** передаст управление программой первому оператору цикла, т. е. функции **input**. Нужно будет ввести опять значение **x(1)** (рис. 4.20).

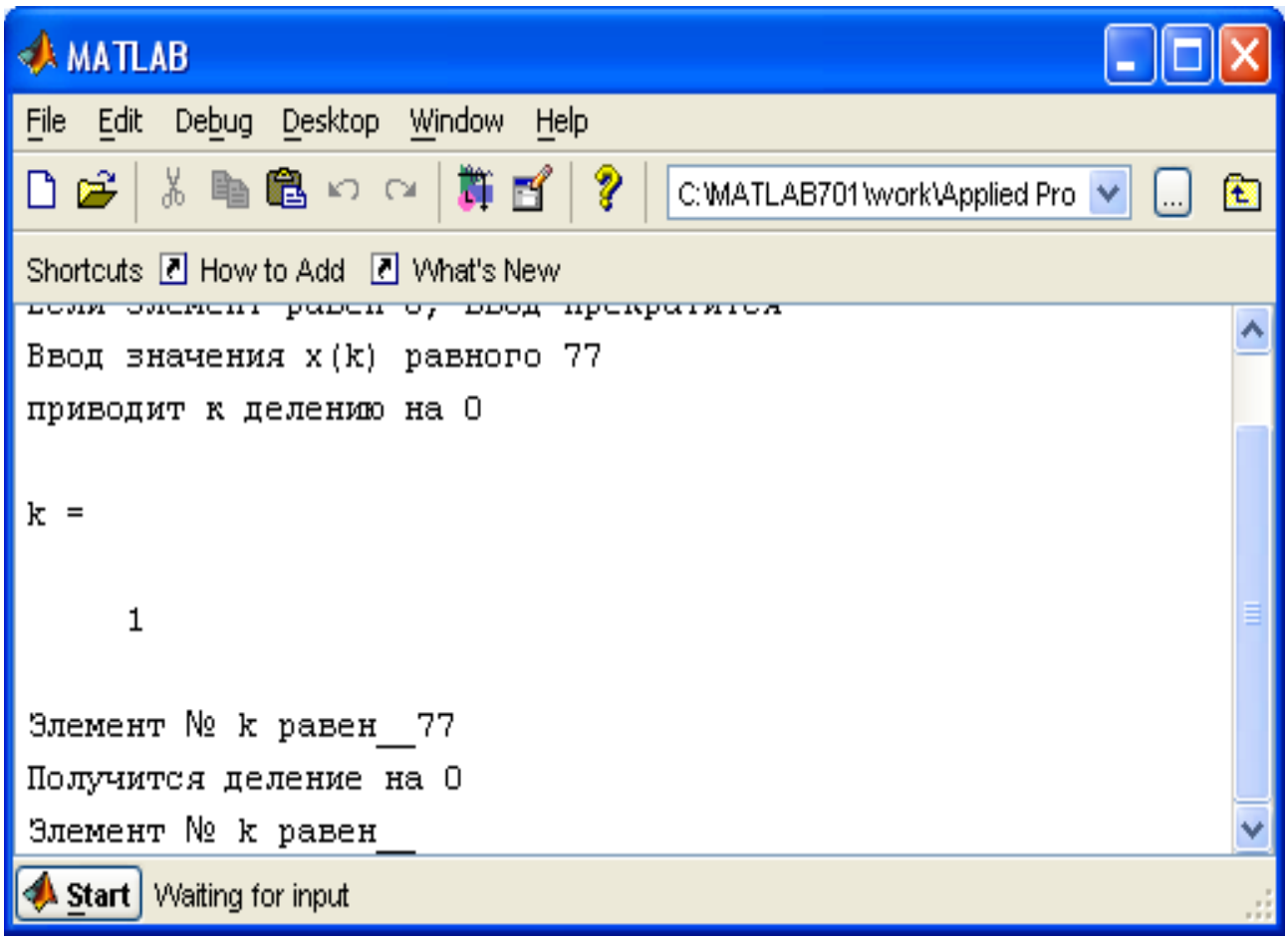


Рисунок 4.20 – Оператор switch x(k) сравнивает введенное значение x(k)

Если сравнение с константой **77** окажется неверно, то управление программой перейдет к строке «case 0», значение **x(1)** будет сравниваться с константой **0**. При результате сравнения **истина**, будут работать операторы, расположенные сразу после «case 0» до строки «otherwise»: появится сообщение «*Преждевременное прекращение ввода*», функция **break** прервет цикл (рис. 4.21), выполнится оператор, следующий сразу за строкой «end», завершающей цикл, т. е. будет вычислена функция $y = \dfrac{0.2x^4 \cos|x|}{x-77}$.

В случае ввода значения **x(k)**, равного **0**, при первой итерации цикла получится массив **x**, состоящий из одного элемента, равного **0**. Если значение **0** будет введено при одной из следующих итераций цикла, длина массива **x** будет равна текущему значению **k**, последний элемент массива будет равен **0**.

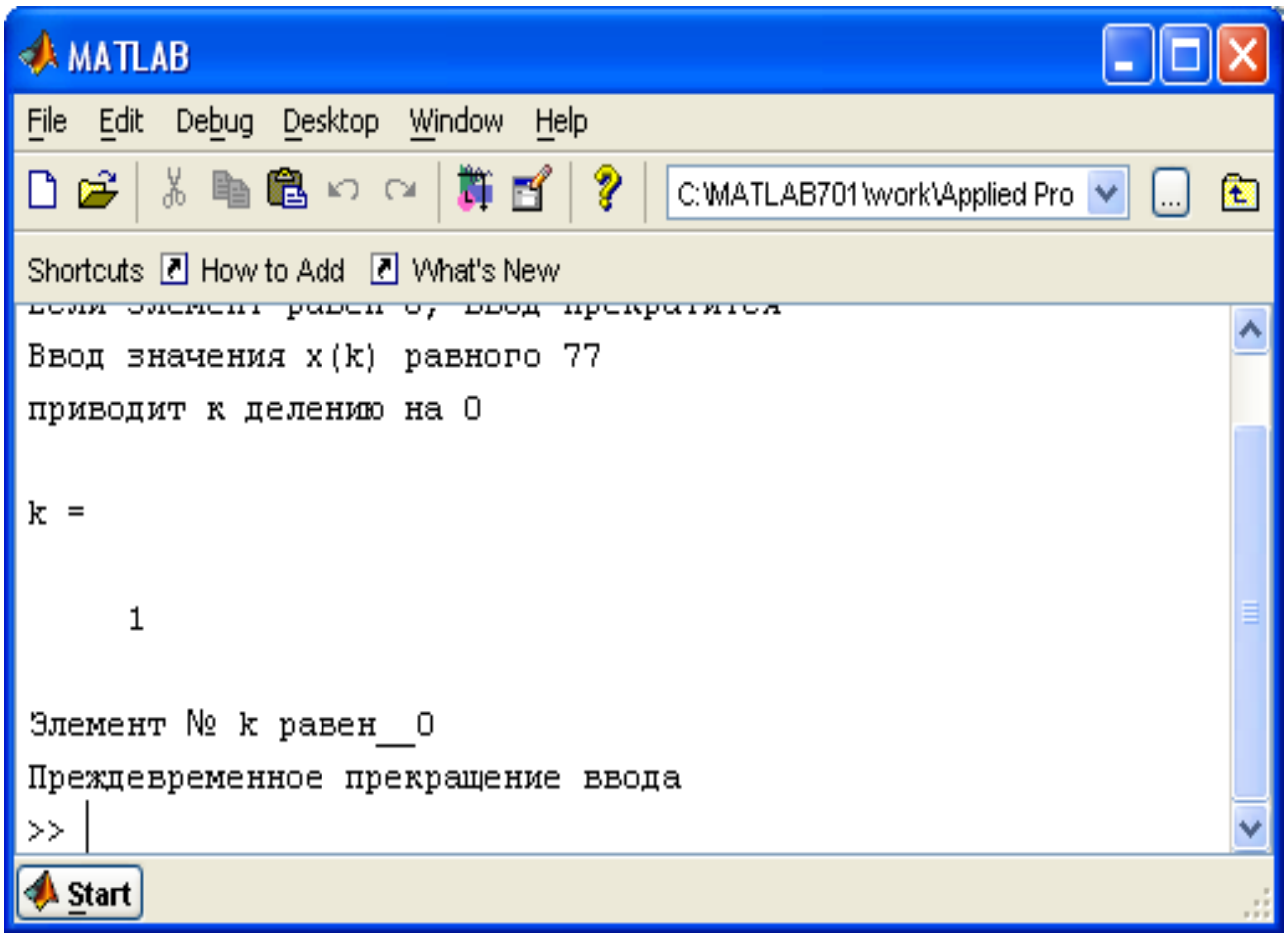


Рисунок 4.21 – Сообщение «Преждевременное прекращение ввода»

При результате сравнения *ложь* сработают операторы, расположенные после строки «**otherwise**» до строки «**end**», завершающей тело оператора **switch**: сообщение «*Все в порядке. Продолжение ввода*» и увеличение счетчика элементов **k=k+1**. Так как после данного выражения не стоит точка с запятой, на каждой итерации цикла будет выводиться на экран текущее значение счетчика **k** (рис. 4.22).

Цикл переходит к новой (второй) итерации, можно вводить значение следующего (второго **k=2**) **x(k)**.

Действия будут повторяться при каждой итерации цикла до его завершения, которое возможно в двух случаях:

– условное выражение **k<=100** будет равно *ложь*: значение **k** увеличивается на 1 на каждой итерации цикла и на одной из итераций примет значение 101;

– при вводе значения 0 и преждевременном завершении цикла оператором **break**.

После окончания работы цикла будет вычислено математическое выражение **y=(0.2.*x.^4).*cos(abs(x))./(x-77)** для всех значений введенного вектора **x.**

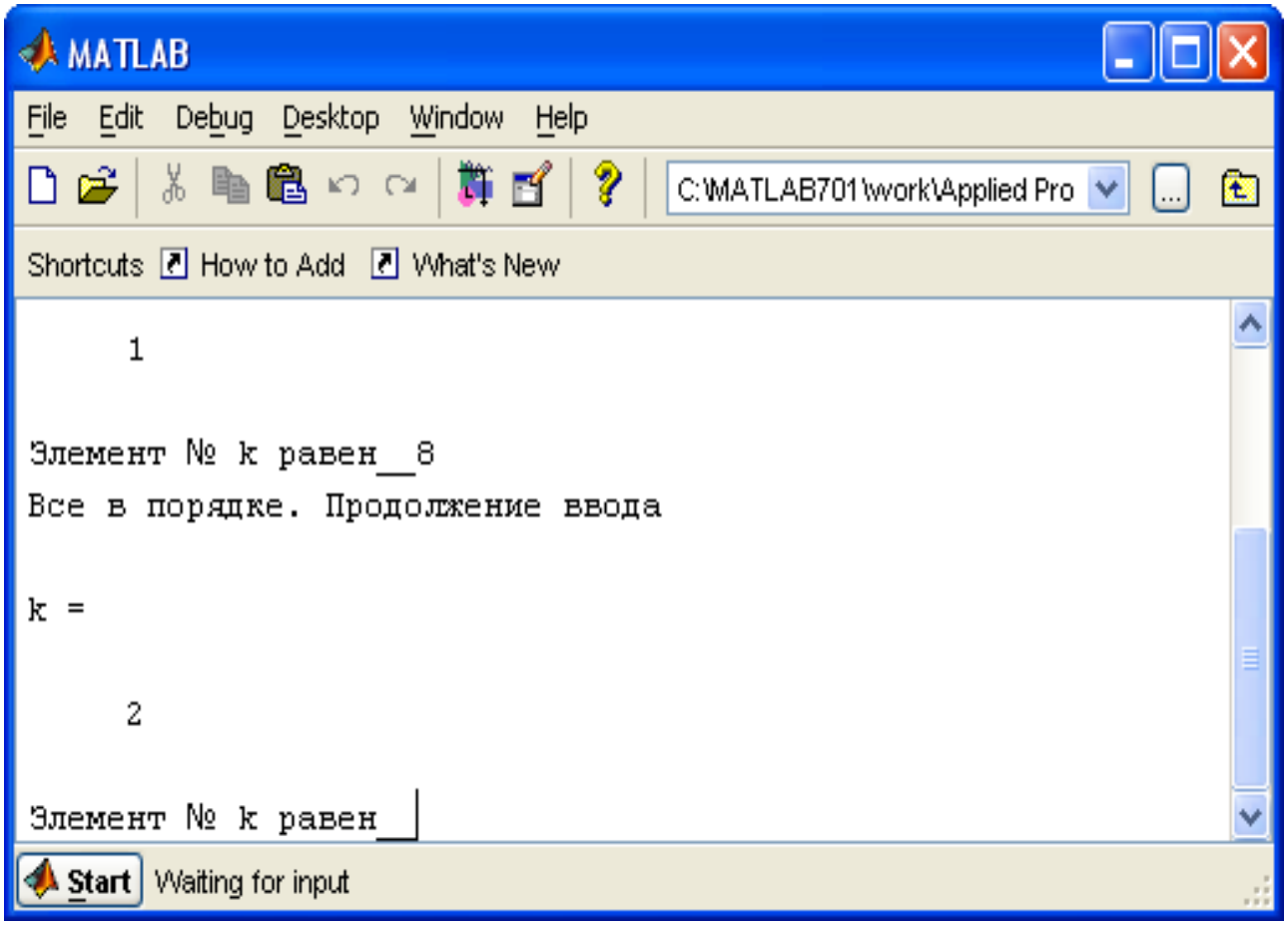


Рисунок 4.22 – Вывод на экран текущего значения счетчика k

После завершения работы сценария в рабочей области Matlab содержатся два массива одинаковой длины **x** и **y**.

Цикл for и ветвление ***if…elseif…else***

Рассмотрим работу цикла **for** и ветвления **if…elseif…else** на примере следующей задачи:

– найти сумму отрицательных элементов массива **y**;

– вычислить количество элементов массива **y**, превышающих 100;

– вычислить количество неотрицательных и не превышающих значения 100 элементов массива **y** больших, чем элементы массива **x** с тем же индексом;

– найти сумму всех остальных элементов массива **y**.

Блок-схема алгоритма и сценарий решения данной задачи приведены на рисунках 4.23 и 4.24.

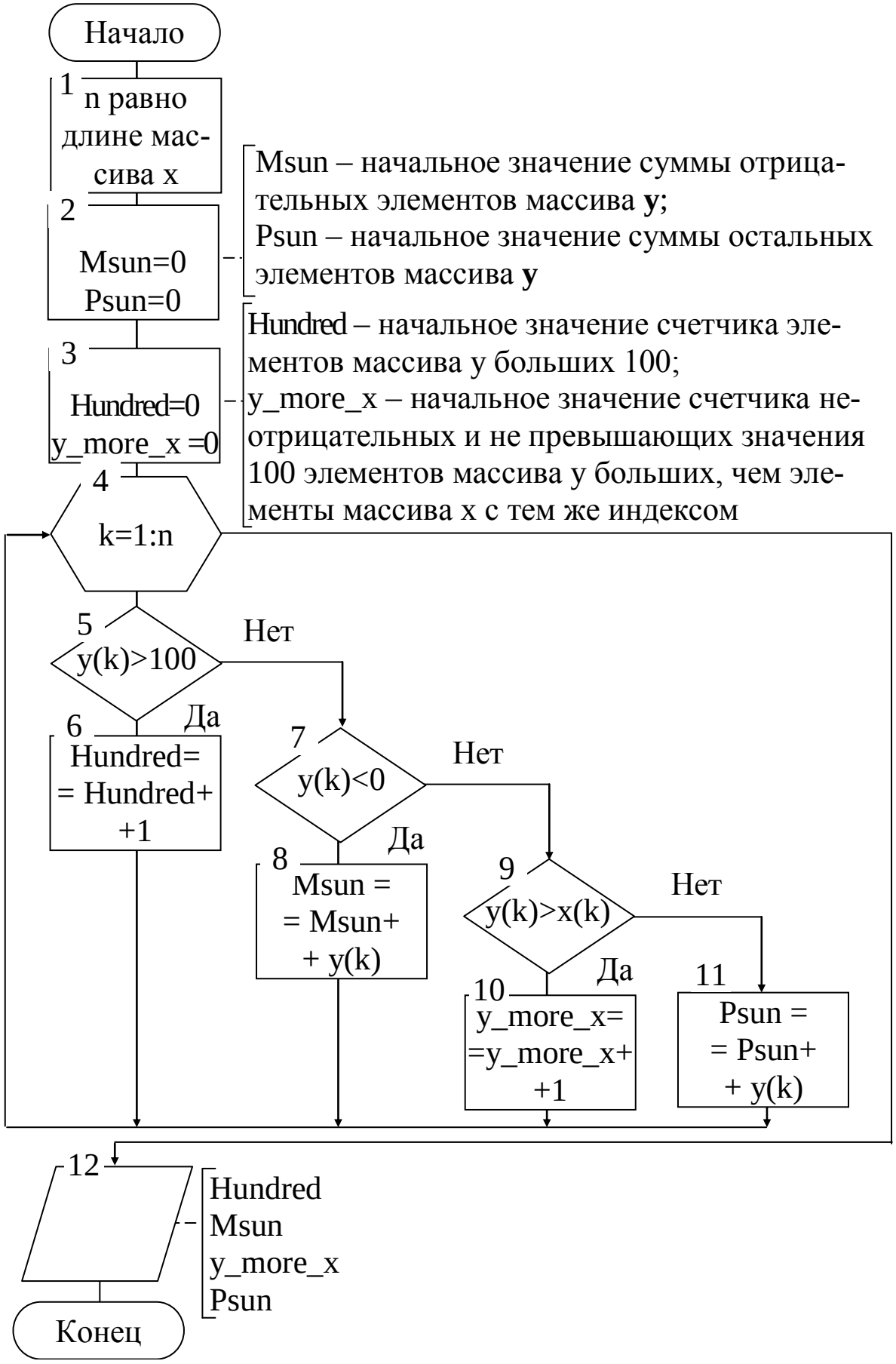


Рисунок 4.23 – Блок-схема алгоритма решения

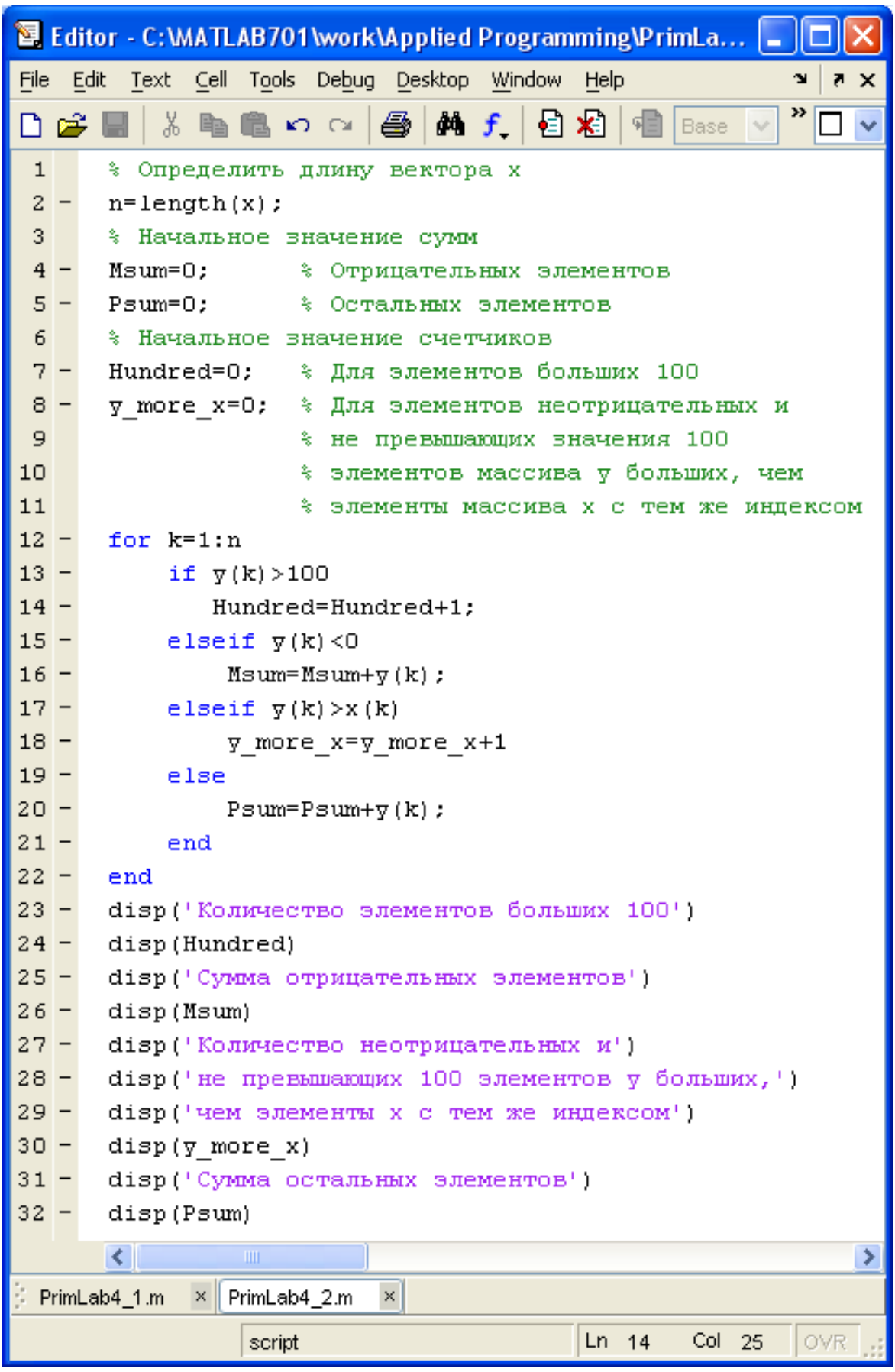


Рисунок 4.24 – Сценарий решения

Рассмотрим работу алгоритма и программы.

Пусть имеются векторы **x** и **y**, которые получились при работе сценария, описанного в разделе 0:

$$x=\begin{bmatrix}-10.00\\6.00\\9.00\\15.00\\8.00\\-11.00\\-17.00\\14.00\\100.00\\0\end{bmatrix};\qquad y=\begin{bmatrix}19.29\\-3.51\\17.58\\124.06\\1.73\\-0.15\\48.90\\-16.68\\749842.50\\0\end{bmatrix}.$$

Для решения поставленной задачи необходимо просмотреть все элементы вектора **y** и отыскать в нем элементы с заданными в задаче признаками, это можно сделать с помощью цикла. Так как длина вектора **y** известна, то для просмотра всех его элементов целесообразно воспользоваться циклом с заранее заданным числом итераций, т. е. циклом **for**. Число итераций естественно равно длине вектора. Поэтому перед началом цикла определим длину вектора **x** или вектора **y**, что даст одинаковый результат, так как длины этих векторов одинаковы. Добавим в блок-схему **блок № 1**. В сценарий – функцию, возвращающую в переменную **n** длину вектора **x**, и комментарий к ней:

```
% определить длину вектора x
n=length(x);
```

Вектор **x**, собственно как и вектор **y**, состоит из 10 элементов, поэтому получится **n=10**.

При работе цикла необходимо будет подсчитывать количество элементов массива **y**, превышающих 100 и больших, чем значения элементов массива **x** с тем же индексом, увеличивая на 1 соответствующие счетчики при появлении элемента, удовлетворяющего одному из данных требований. Поэтому перед началом цикла инициализируем два счетчика, присвоив им значение **0**. В блок-схеме смотри **блок № 3**. Строки сценария с комментариями:

```
% Начальное значение счетчиков
Hundred=0;    % для элементов больших 100
y_more_x=0;  % для элементов неотрицательных и
              % не превышающих значения 100
              % элементов массива y больших, чем
              % элементы массива x с тем же индексом
```

К текущему значению сумм отрицательных и остальных элементов массива **y** будет прибавляться значение элемента массива **y** при появлении элемента, удовлетворяющего одному из условий. Перед началом просмотра элементов в цикле значение сумм равно 0. В блок-схеме смотри **блок № 2**. Строки сценария и комментарии к ним:

% Начальное значение сумм

Msum=0; % Отрицательных элементов

Psum=0; % Остальных элементов

Начало цикла с заранее заданным числом итераций, равным **n** (**блок № 4**):

for k=1:n

Начальное значение переменной цикла **k=1**. Размер шага изменения не указан, поэтому он будет принят равным **1**. Конечное значение переменной цикла **k=n**, т. е. **10**. Таким образом, цикл повторится **10** раз, переменная цикла **k** при первой итерации будет равна **1**, при каждой следующей итерации на **1** больше, чем на предыдущей.

В теле цикла имеется оператор ветвления **if**. Очередной **k**-й текущий элемент вектора **y**, т. е. **y(k),** сравнивается с константой **100,** что описывается **блоком № 5** блок-схемы и строкой сценария:

if y(k)>100

Если результат сравнения окажется равным значению «**истина**», то будет выполняться действие **блока № 6** блок-схемы, а в сценарии строка:

Hundred=Hundred+1;

После увеличения на единицу счетчика **Hundred** работа оператора **if…elseif…else** закончится, управление программой перейдет к операторам, расположенным после соответствующего слова **end**. Так как никаких операторов в теле цикла больше нет, то начнется следующая итерация цикла, если **k<n**, или цикл будет завершен, если значение оператора цикла **k** уже достигло **n.**

Эта ситуация сложится два раза: при четвертой и девятой итерациях цикла, так как **y(4)=124.06**, **y(9)=749842.50**, поэтому после завершения цикла значение переменной **Hundred** окажется равным **2.**

Если результат сравнения окажется равным значению «**ложь**», то управление перейдет к **блоку № 7** блок-схемы, а в сценарии – к строке, в которой определяется, меньше ли **y(k)** нуля:

elseif y(k)<0

При результате сравнения, равном «**истина**», работает **блок № 8** блок-схемы, строка сценария:

Msum=Msum+y(k);

После прибавления к **Msum** значения **y(k)** работа оператора **if…elseif…else** закончится, управление программой перейдет к операторам, расположенным после соответствующего слова **end**. Так как никаких операторов в теле цикла больше нет, то начнется следующая итерация цикла, если **k<n**, или цикл будет завершен, если значение оператора цикла **k** уже достигло **n**.

В векторе **y** имеется три отрицательных элемента: **y(2)= -3.51**, **y(6)= -0.15**, **y(8)= -16.68**, в результате накопленная сумма окажется равной **Msum= -20.34**.

В том случае, если и последняя проверка условия вернет значение «**ложь**», то будет работать **блок № 8** блок-схемы, строка сценария:

elseif y(k)>x(k)

Заметим, что данную проверку, а именно больше ли элемент вектора **y**, чем элемент вектора **x**, будут проходить только элементы не большие 100 и неотрицательные, другие элементы вектора **y** никогда не будут подвергнуты данному сравнению, так как будут обработаны одной из предыдущей уже описанной частью оператора **if…elseif…else**, после чего будет начинаться новая итерация цикла, все манипуляции будут производиться со следующим элементом вектора **y**.

При результате сравнения **y(k)>x(k)**, равном «**истина**», произойдет увеличение на единицу счетчика **y_more_x**:

y_more_x=y_more_x+1,

после чего, как и в предыдущих случаях, закончится итерация цикла. Описанных случаев получится три: при **k=1 y=19.29, x=-10.00,** при **k=3 y=17.58, x=9.00** и при **k=7 y=48.90, x=-17.00.**

Если результат сравнения окажется равным «**ложь**», то управление программой перейдет к **блоку № 11** блок-схемы и строкам программы:

else

Psum=Psum+y(k);

Таким образом, во всех случаях, когда предыдущие сравнения вернули результат «**ложь**», произойдет прибавление значения текущего элемента вектора **y** к **Psum** (на пятой и десятой итерациях цикла).

Конец оператора ветвления **if**:

end

Конец цикла **for**:

end

После тела цикла в программе предусмотрен вывод результатов счета с помощью встроенных функций Matlab **disp**:

disp('Количество элементов больших 100')
disp(Hundred)
disp('Сумма отрицательных элементов')
disp(Msum)
disp('Количество неотрицательных и')
disp('не превышающих 100 элементов у больших,')
disp('чем элементы x с тем же индексом')
disp(y_more_x)
disp('Сумма остальных элементов')
disp(Psum)

Результат работы программы в виде текстовых сообщений приведен на рисунке 4.25.

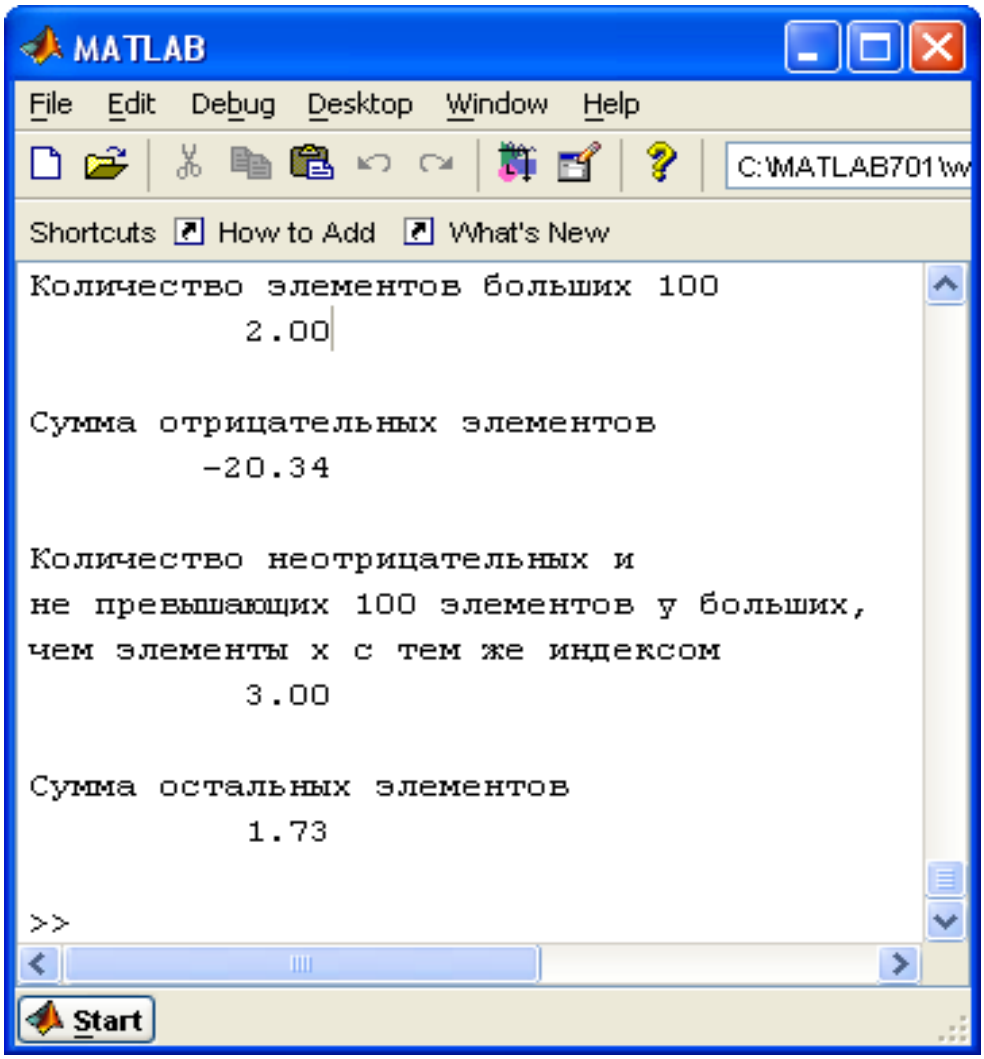


Рисунок 4.25 – Результат работы программы в виде текстовых сообщений

4.4. Варианты заданий

Номер варианта	Функция Q	Организовать ввод	Вычислить
1	$Q=x\times e^{x}+0.2$	Не более 50 элементов вектора *x* в интервале [2; 0], вычислить функцию Q от вектора *x*	Количество элементов вектора Q, меньших 0.1; сумму остальных отрицательных элементов вектора Q; количество неотрицательных элементов вектора Q
2	$Q=\ln(x/(x20)^{3/2})$	Не более 33 элементов вектора *x* в интервале [0; 19], вычислить функцию Q от вектора *x*	Количество положительных элементов вектора Q, меньших 0.5; сумму остальных положительных значений вектора Q; количество отрицательных и равных 0 элементов вектора Q
3	$Q=\frac{x^2-3x+2}{x^2+2x-1}$	Не более 50 элементов вектора *x* в интервале [1.3; 1.4], вычислить функцию Q от вектора *x*	Количество элементов вектора Q, меньших 0.0645; сумму элементов вектора Q, больших 0.0639; количество остальных элементов вектора Q
4	$Q=\operatorname{arctg}(x)-0.5\times\ln(1+2x)$	Не более 15 элементов вектора *x* в интервале [0.5; 1], вычислить функцию Q от вектора *x*	Количество элементов вектора Q, меньших 0.2; сумму элементов вектора Q, больших 1.2; количество остальных элементов вектора Q
5	$Q=\lvert x\rvert\times e^{\lvert x1\rvert}$	Не более 22 элементов вектора *x* в интервале [0.5; 0.5], вычислить функцию Q от вектора *x*	Количество элементов вектора Q, меньших 0.05; количество элементов вектора Q, больших 0.22; сумму остальных элементов вектора Q
6	$Q=x^{2}+e^{1.1x+3}$	Не более 77 элементов вектора *x* в интервале [3; 1], вычислить функцию Q от вектора *x*	Сумму элементов вектора Q, меньших 0.05; количество элементов вектора Q, больших 0.22; сумму остальных элементов вектора Q
7	$Q=x^{4}1.5\operatorname{arctg}(x)$	Не более 25 элементов вектора *x* в интервале [0; 1], вычислить функцию Q от вектора *x*	Количество элементов вектора Q, больших 0.1; сумму элементов вектора Q, меньших 0.6; сумму остальных элементов вектора Q
8	$Q=0.4x+e^{\lvert x2\rvert}$	Не более 54 элементов вектора *x* в интервале [1; 3], вычислить функцию Q от вектора *x*	Сумму элементов вектора Q, больших 1.5; количество элементов вектора Q, меньших 0.5; количество остальных элементов вектора Q

Номер варианта	Функция Q	Организовать ввод	Вычислить
9	$Q=\mathrm{tg}(x^2)+2x$	Не более 19 элементов вектора *x* в интервале [1; 1], вычислить функцию Q от вектора *x*	Количество элементов вектора Q, меньших 0.5; сумму элементов вектора Q, больших 1; сумму остальных элементов вектора Q
10	$Q=2*x+\sin(x^2)$	Не более 32 элементов вектора *x* в интервале [1; 2], вычислить функцию Q от вектора *x*	Сумму элементов вектора Q, больших 3.5; количество элементов вектора Q, меньших 3; количество остальных элементов вектора Q
11	$Q= e^x\times\ln(x)$	Не более 45 и элементов вектора *x* в интервале [3; 1], вычислить функцию Q от вектора *x*	Количество элементов вектора Q, меньших 0.08; сумму элементов вектора Q, больших 0.02; сумму остальных элементов вектора Q
12	$Q= =\sqrt{x+5.5}+x^4$	Не более 81 элемента вектора *x* в интервале [1; 1], вычислить функцию Q от вектора *x*	Сумму элементов вектора Q, больших 3.1; количество элементов вектора Q, больших 2.4; количество остальных элементов вектора Q
13	$Q=2^x e^x$	Не более 64 элементов вектора *x* в интервале [3; 0], вычислить функцию Q от вектора *x*	Количество элементов вектора Q, меньших 0.05; сумму элементов вектора Q, больших 0.1; сумму остальных элементов вектора Q
14	$Q= =x^4 14x^3+ +60x^2 70x$	Не более 31 элемента вектора *x* в интервале [2; 1], вычислить функцию Q от вектора *x*	Сумму элементов вектора Q, меньших 10; количество элементов вектора Q, больших 100; количество остальных элементов вектора Q
15	$Q= =\frac{x^2-30}{x^4+7.7x^2+3}$	Не более 105 элементов вектора *x* в интервале [7; 9], вычислить функцию Q от вектора *x*	Количество элементов вектора Q, меньших 6.9×10^3; сумму элементов вектора Q, больших 7.3×10^3; сумму остальных элементов вектора Q
16	$Q=2x^2+3e^x$	Не более 117 элементов вектора *x* в интервале [0.1; 1], вычислить функцию Q от вектора *x*	Сумму элементов вектора Q, меньших 2.5; количество элементов вектора Q, больших 2.9; количество остальных элементов вектора Q

Номер варианта	Функция Q	Организовать ввод	Вычислить
17	$Q=2+xx^2$	Не более 93 элементов вектора *x* в интервале [0; 1], вычислить функцию Q от вектора *x*	Количество элементов вектора Q, меньших 2.05; сумму элементов вектора Q, больших 2.2; сумму остальных элементов вектора Q
18	$Q=(1x)^4$	Не более 23 элементов вектора *x* в интервале [0; 2], вычислить функцию Q от вектора *x*	Сумму элементов вектора Q, меньших 0.2; количество элементов вектора Q, больших 0.7; количество остальных элементов вектора Q
19	$Q=\cos(x)+x$	Не более 94 элементов вектора *x* в интервале [1; 2], вычислить функцию Q от вектора *x*	Количество элементов вектора Q, меньших 1.55; сумму элементов вектора Q, больших 1.57; сумму остальных элементов вектора Q
20	$Q=$ $=x^{1/2}+(1x)^{2/3}$	Не более 18 элементов вектора *x* в интервале [8; 10], вычислить функцию Q от вектора *x*	Сумму элементов вектора Q, меньших 0.997; количество элементов вектора Q, больших 0.999; сумму остальных элементов вектора Q
21	$Q=$ $=x^3 6x^2+9x+4$	Не более 73 элементов вектора *x* в интервале [2; 4], вычислить функцию Q от вектора *x*	Количество элементов вектора Q, меньших 5; сумму элементов вектора Q, больших 6; количество остальных элементов вектора Q
22	$Q=2x^2x^4$	Не более 23 элементов вектора *x* в интервале [0.5; 1.5], вычислить функцию Q от в ектора *x*	Сумму элементов вектора Q, меньших 0.2; количество элементов вектора Q, больших 0.4; количество остальных элементов вектора Q
23	$Q=x+1/\lvert x+1\rvert$	Не более 46 элементов вектора *x* в интервале [0.9; 1], вычислить функцию Q от вектора *x*	Количество элементов вектора Q, меньших 2; сумму элементов вектора Q, больших 6; количество остальных элементов вектора Q
24	$Q=x^3\sqrt{x-1}+4$	Не более 49 элементов вектора *x* в интервале [1; 2], вычислить функцию Q от вектора *x*	Сумму элементов вектора Q, меньших 2; количество элементов вектора Q, больших 6; сумму остальных элементов вектора Q

Результаты работы по индивидуальным заданиям лабораторной работы 4 необходимы для выполнения лабораторной работы 5.

Контрольные вопросы

1. Цикл while.
2. Цикл for.
3. Ветвление switch.
4. Ветвление if…elseif…else.
5. Оператор continue.
6. Оператор break.
7. Операторы отношения.
8. Логические операторы.
9. Оператор if.
10. Оператор if…else…end.
11. Оператор if… elseif …end.
12. Оператор if… elseif … else…end.
13. Оператор переключения Switch.
14. Цикл типа for…end.
15. Цикл типа while…end.
16. Оператор continue.
17. Оператор break.

Лабораторная работа № 5
М-ФУНКЦИИ И ПОДСИСТЕМЫ SIMULINK

Цель работы: получение практических навыков, необходимых для программирования пользовательских функций и подсистем Simulink с помощью пакета Matlab.

Файлы, которые содержат коды языка Matlab, называются **M**-файлами. Для создания **M**-файла используется текстовый редактор.

Типы **M**-файлов. Существует два типа **M**-файлов. Это **M**-сценарии и **M**-функции со следующими характеристиками:

1. **M**-сценарий не использует входных и выходных аргументов, оперирует с данными из рабочей области, предназначен для автоматизации последовательности шагов, которые нужно выполнять много раз.

2. **M**-функция использует входные и выходные аргументы по умолчанию, внутренние переменные являются локальными по отношению к функции, предназначена для расширения возможностей языка Matlab.

Подсистема модели Simulink является своеобразной функцией с входами, выходами и внутренними вычислениями.

5.1. Создание М-файлов

M-файлы являются обычными текстовыми файлами, которые создаются с помощью текстового редактора. Для операционной среды персонального компьютера система Matlab поддерживает специальный встроенный редактор/отладчик, хотя можно использовать и любой другой текстовый редактор с ASCII-кодами.

Открыть редактор можно двумя способами:

– из меню **File** выбрать опцию **New**, а затем **M-File** (т. е. **File/New/M-File**);

– использовать команду редактирования **edit.**

Например, команда

edit FileName

запускает редактор и открывает файл **FileName.m**. Если имя файла опущено, то запускается редактор и открывается файл без имени. Если файла с указанным именем не существует, то такой файл будет создан.

Требования к именам файлов аналогичны требованиям к идентификаторам массивов Matlab.

5.2. M-сценарии

M-сценарии предназначены для автоматизации многократно выполняемых вычислений. Сценарии оперируют данными из рабочей области и могут генерировать новые данные для последующей обработки в этом же файле. Данные, которые используются в сценарии, сохраняются в рабочей области после завершения сценария и могут быть использованы для дальнейших вычислений.

В работе, посвященной вопросам программирования, программы поиска экстремумов функции сохранялись в **M**-файлах-сценариях.

5.3. M-функции

M-функции являются **M**-файлами, которые допускают наличие входных и выходных аргументов. Они работают с переменными в пределах собственной рабочей области, отличной от общей рабочей области системы Matlab.

Для примера оформим программу поиска максимального значения функции $\mathbf{y=x^3-6x^2+9x+4}$ методом дихотомии, приведенную в качестве примера в лабораторной работе, посвященной вопросам программирования, в виде **M**-функции Matlab.

*Структура **M**-функции*

M-функция имеет определенную структуру. Текст **M**-функция состоит:

– из строки определения функции;

– первой строки комментария;

– собственно комментария;

– тела функции, в том числе внутренних функций;

– строчных комментариев.

Строка определения функции сообщает системе Matlab, что файл является **M**-функцией, а также определяет список входных аргументов.

Строка определения функции **dichotomy** имеет вид:

function y = dichotomy (x),

где **function** – ключевое слово, определяющее **M**-функцию;

y – выходной аргумент;

dichotomy – имя функции, имя файла, в котором содержится функция должно совпадать с именем функции, в данном случае файл назовем **dichotomy.m**;

x – входной аргумент.

Каждая функция в системе Matlab содержит строку определения функции.

Если функция имеет более одного выходного аргумента, список выходных аргументов помещается в квадратные скобки. Входные аргументы, если они присутствуют, помещаются в круглые скобки. Для отделения аргументов во входном и выходном списках применяются запятые. Например:

function [x0, y0] = dichotomy (a, b, h)

Имена входных переменных могут, но не обязаны совпадать с именами, указанными в строке определения функции.

Первая строка комментария. Для нашей функции **dichotomy** первая строка комментария пусть выглядит так:

% **dichotomy** поиск максимума функции

Это первая строка текста, которая появляется, когда пользователь набирает команду **help dichotomy**. Кроме того, первая строка комментария выводится на экран по команде поиска **lookfor**.

Для **M**-файлов можно создать online-подсказку, вводя текст в одной или более строках комментария.

Например, сформируем несколько строк комментария:

% **dichotomy** поиск максимума функции
% функция **dichotomy (a, b, h)**
% вычисляет максимум функции
% методом дихотомии
% в интервале от **a** до **b**,
% шаг изменения аргумента **h**.

Тогда при вводе команды подсказки **help dichotomy** система Matlab отображает строки комментария, которые размещаются между строкой определения функции и первой пустой строкой либо началом программы; комментарии, размещенные вне этой области, игнорируются.

Тело функции содержит код языка Matlab, который выполняет вычисления и присваивает значения выходным аргументам. Операторы в теле функции могут состоять из вызовов функций, программных

конструкций для управления потоком команд, интерактивного ввода/вывода, вычислений, присваиваний, комментариев и пустых строк.

В нашем примере тело функции:

```
interval=abs(a-b);
while interval>=h*2;
  x0=a+(b-a)/2;
  x1=x0-h/2;
  x2=x0+h/2;
  Q1=x1.^3-6*x1.^2+9*x1+4;
  Q2=x2.^3-6*x2.^2+9*x2+4;
  if Q1>Q2;
     b=x2;
  else
     a=x1;
  end
  interval=abs(b-a);
end
x0=a+(b-a)/2;
y0=x0^3-6*x0^2+9*x0+4;
```

Последние две строки «**x0=a+(ba)/2**» (ордината максимального значения функции $\mathbf{y=x^3-6x^2+9x+4}$) и «**y0=x0^36*x0^2+9*x0+4**» (максимальное значение функции $\mathbf{y=x^3-6x^2+9x+4}$) обязательны, так как **x0** и **y0** – выходные аргументы функции, поэтому их необходимо вычислить в теле функции.

Для вычислений в теле функции также необходимы значения переменных: **a** – левая граница интервала неопределенности, **b** – правая граница интервала неопределенности, **h** – шаг изменения аргумента функции. Их указывают в строке определения функции. Затем при вызове **M**-функции эти переменные должны принимать численные значения.

*Пример **M**-функции*

Полный текст функции в редакторе Matlab, содержащий строку определения функции, online-подсказку, программный код в виде операторов и функции, строчные комментарии, поясняющие работу операторов и функции, смотри на рисунке 5.1.

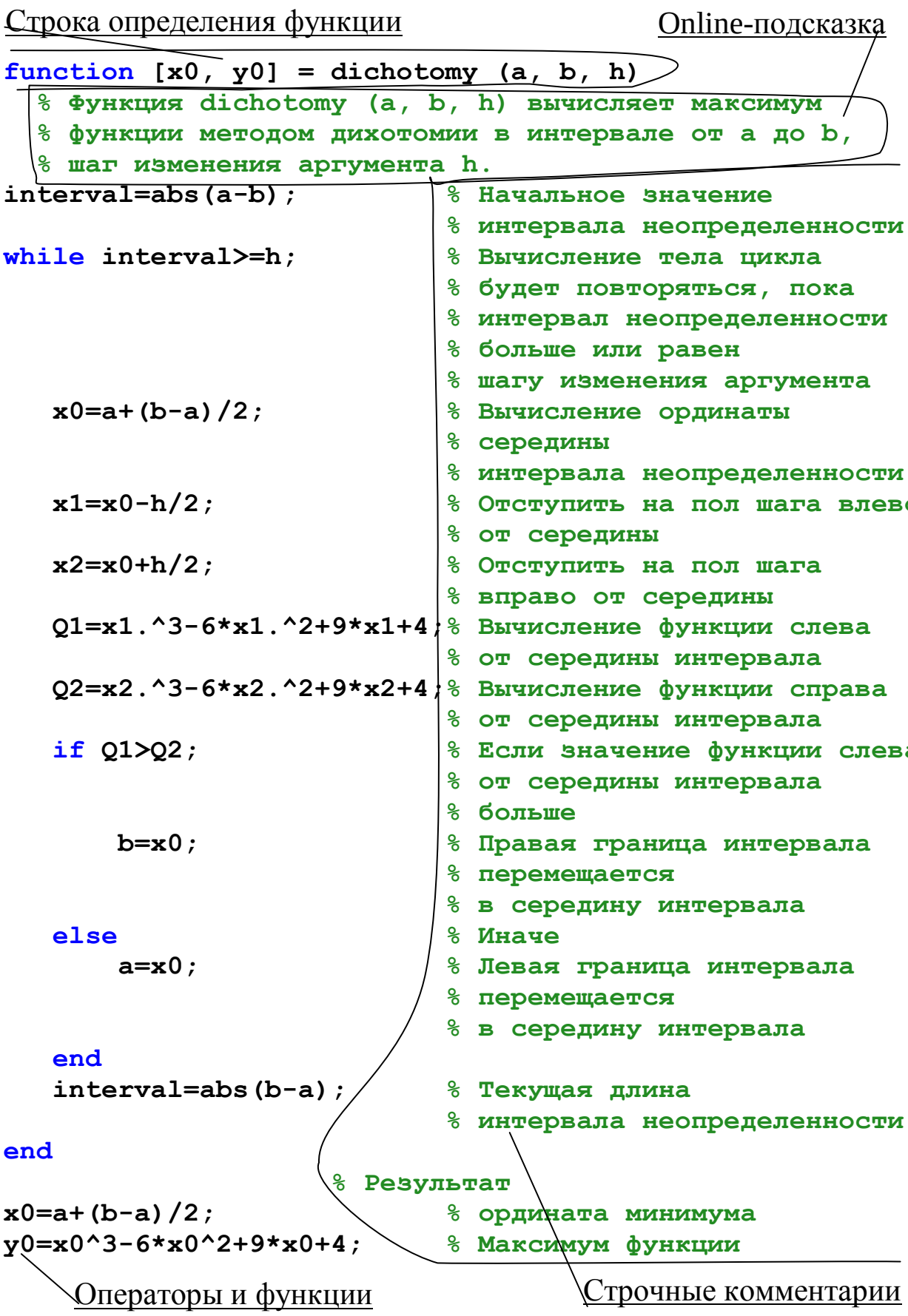


Рисунок 5.1 – Текст функции в редакторе Matlab

*Вызов **M**-функции*

Вызвать **M**-функции **dichotomy** можно из командной строки, из сценария, из другой **M**-функции, из блока Simulink и так далее, важно только, чтобы заранее были известны значения входных аргументов **a**, **b** и **h**.

Пример вызова **M**-функции **dichotomy** можно из командной строки, из сценария или другой **M**-функции:

[x,y]=dichotomy (0.2, 1.5, 0.02)

При вычислении в теле **M**-функции переменная **a** примет значение **0.2**, и **b** – **1.5**, а **h** – **0.02**.

Когда вычисления будут окончены, в выходную переменную **x** **M**-функция вернет значение своей выходной переменной **x0**, а в **y** – значение своей выходной переменной **y0**, в данном случае **x=0.9973**, **y=8.0000**.

Если используются переменные, которые в результате вычислений получили необходимые численные значения, например

L=0.2;

R=1.5;

h=0.02;

то в этом случае вызов **M**-функции:

[x,y]=dichotomy (L, R, h)

Функция возвращает тот же результат, что и при предыдущем вызове.

Пользовательская функция в Simulink

Существуют различные способы вызова **M**-функций при работе моделей Simulink.

Рассмотрим один из вариантов использования **M**-функции в модели Simulink. Для того чтобы можно было задать **M**-функцию, вычисляющую максимум функции $\mathbf{y=x^3-6x^2+9x+4}$, модель содержит блок «Embedded MATLAB Function» из библиотеки «User-Defined Function» (рис. 5.2).

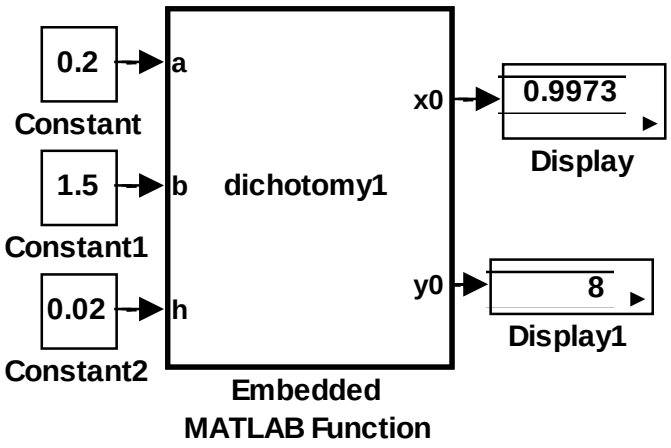


Рисунок 5.2 – Блок «Embedded MATLAB Function»
из библиотеки «User-Defined Function»

Двойной щелчок левой кнопкой мыши по блоку «Embedded MATLAB Function» приводит к открытию редактора с заготовкой **M**-функции. Заменим имеющийся текст текстом из **M**-функции **dichotomy**, но в данном случае не допускаются символы кириллицы, поэтому уберем все комментарии, а функция пусть называется **dichotomy1**. Текст функции:

```
function [x0,y0] = dichotomy1 (a,b,h)
interval=abs(a-b);
while interval>=h;
  x0=a+(b-a)/2;
  x1=x0-h/2;
  x2=x0+h/2;
  Q1=x1.^3-6*x1.^2+9*x1+4;
  Q2=x2.^3-6*x2.^2+9*x2+4;
  if Q1>Q2;
     b=x0;
  else
     a=x0;
  end
  interval=abs(b-a);
end
x0=a+(b-a)/2;
y0=x0^3-6*x0^2+9*x0+4;
```

Так как строка определения **M**-функции содержит два выходных и три входных параметра, то блок автоматически после сохранения **M**-функции и закрытия окна редактора преобразуется в блок с тремя

входами и двумя выходами. Осталось только присоединить необходимые источники и регистраторы сигналов.

При таких же, как в предыдущих вызовах, входных данных m-функция вернет такие же результаты.

Внутренние функции

M-функция может содержать внутренние функции. Особенности таких функций заключаются в следующем:

1) текст внутренних функций расположен после текста основной **M**-функции в том же **M**-файле;

2) использоваться внутренние функции могут только одной основной **M**-функцией (расположенной в том же **M**-файле);

3) синтаксические правила и правила вызова внутренних функций такие же, как у **M**-функций.

Преобразуем **M**-функцию **dichotomy**. Новая **M**-функция будет называться **dichotomy2**. В ней расчет текущего значения функции $\mathbf{y=x^3-6x^2+9x+4}$ слева и справа от середины интервала неопределенности происходит с помощью вызова внутренней функции **curr_res**, которая рассчитывает значение функции, ее текст:

```
function curr_y=curr_res(x)
curr_y=x.^3-6*x.^2+9.*x+4;
```

с соответствующим входным параметром (**x1** или **x2**):

```
  Q1=curr_res(x1);  % Вычисление функции слева
                    % от середины интервала
  Q2=curr_res(x2);  % Вычисление функции справа
                    % от середины интервала
```

Для того чтобы вычислить окончательный результат, внутренняя функция вызывается с входным параметром (**x0**):

```
y0=curr_res(x0);        % Максимум функции
```

Полный текст m-функции с вызовом внутренней функции:

```
function [x0, y0] = dichotomy2 (a, b, h)
% dichotomy поиск максимума функции
% Функция dichotomy (a, b, h) вычисляет максимум функции
% методом дихотомии в интервале от a до b,
% шаг изменения аргумента h.
interval=abs(a-b);    % Начальное значение
                      % интервала неопределенности
while interval>=h;    % Вычисление тела цикла
                      % будет повторяться, пока
                      % интервал неопределенности
```

```
                    % больше или равен
                    % шагу изменения аргумента
  x0=a+(b-a)/2;     % Вычисление ординаты
                    % середины
                    % интервала неопределенности
  x1=x0-h/2;        % Отступить на пол шага влево
                    % от середины
  x2=x0+h/2;        % Отступить на пол шага
                    % вправо от середины
  Q1=curr_res(x1);  % Вычисление функции слева
                    % от середины интервала
  Q2=curr_res(x2);  % Вычисление функции справа
                    % от середины интервала
  if Q1>Q2;         % Если значение функции слева
                    % от середины интервала
                    % больше
    b=x0;           % Правая граница интервала
                    % перемещается
                    % в середину интервала
  else              % Иначе
    a=x0;           % Левая граница интервала
                    % перемещается
                    % в середину интервала
  end
  interval=abs(b-a); % Текущая длина
                    % интервала неопределенности
end
                    % Результат
x0=a+(b-a)/2;       % Середина последнего
                    % интервала неопределенности
y0=curr_res(x0);    % Максимум функции
                    % Расчет текущего значения функции
function curr_y=curr_res(x)
curr_y=x.^3-6*x.^2+9*x+4;
```

Количество внутренних функций неограниченно. Вторая внутренняя функция будет строить график функции $y=x^3-6x^2+9x+4$, вызывая при этом внутреннюю функцию

function plotter(l, p, delta)
absc=l:delta:p; % Вектор значений аргумента
ordinat=curr_res(absc); % Вектор значений функции
plot(absc,ordinat) % Построение графика

Входные аргументы функции **plotter**:

l – минимальное значение вектора аргументов функции;

p – максимальное значение вектора аргументов функции;

delta – шаг приращения вектора аргументов функции.

Выходных аргументов функция не имеет.

В первой строке тела функции задается вектор **absc**. Во второй вычисляется функция $\mathbf{y=x^3-6x^2+9x+4}$ для всех значений вектора с помощью вызова внутренней функции «**curr_res**» с вектором **absc** в качестве аргумента, поэтому функция «**curr_res**» возвращает результат **ordinat** – вектор той же длины. В третьей строке строится график, при этом по оси абсцисс откладывается вектор **absc**, а по оси ординат вектор **ordinat**.

Вызов функции:

plotter(a, b, h)

Аргументы при вызове функции «**plotter**» – это те левая и правая границы определения функции и шаг приращения аргумента, которые получит главная **M**-функция при вызове, таким образом, график будет построен для всего интервала поиска максимума функции.

Окончательно оформим **M**-функцию поиска максимального значения функции $\mathbf{y=x^3-6x^2+9x+4}$ и построения ее графика в интервале поиска:

```
function [x0, y0] = dichotomy3 (a, b, h)
  % dichotomy поиск максимума функции
  % функция dichotomy (a, b, h) вычисляет максимум функции
  % методом дихотомии в интервале от a до b,
  % шаг изменения аргумента h.
plotter(a, b, h)            % Построение графика
interval=abs(a-b);          % Начальное значение
                            % интервала неопределенности
while interval>=h;          % Вычисление тела цикла
                    % будет повторяться, пока
                    % интервал неопределенности
                    % больше или равен
                    % шагу изменения аргумента
```

```
x0=a+(b-a)/2;          % Вычисление ординаты
                       % середины
                       % интервала неопределенности
  x1=x0-h/2;           % Отступить на пол шага влево
                       % от середины
  x2=x0+h/2;           % Отступить на пол шага
                       % вправо от середины
  Q1=curr_res(x1);     % Вычисление функции слева
                       % от середины интервала
  Q2=curr_res(x2);     % Вычисление функции справа
                       % от середины интервала
  if Q1>Q2;            % Если значение функции слева
                       % от середины интервала
                       % больше
     b=x0;             % Правая граница интервала
                       % перемещается
                       % в середину интервала
  else                 % Иначе
     a=x0;             % Левая граница интервала
                       % перемещается
                       %в середину интервала
  end
  interval=abs(b-a);   % Текущая длина
                       % интервала неопределенности
end
                % Результат
x0=a+(b-a)/2;          % Середина последнего
                       % интервала неопределенности
y0=curr_res(x0);       % Минимум функции

% Расчет текущего значения функции
function curr_y=curr_res(x)
curr_y=x.^3-6*x.^2+9.*x+4;

% Построение графика функции
function plotter(l, p, delta)
absc=l:delta:p;        % Вектор значений аргумента
ordinat=curr_res(absc); % Вектор значений функции
plot(absc,ordinat)     % Построение графика
```

5.4. Подсистемы в моделях Simulink

Модель Simulink также может иметь внутренние функции, оформленные как подсистемы. Для создания подсистем в библиотеке Simulink предусмотрен рад блоков. Подсистема – это фрагмент Simulink-модели, оформленный в виде отдельного блока.

Если блок-схема модели слишком сложная и имеет большие размеры, ее можно упростить, группируя блоки в подсистемы. Использование подсистем дает следующие преимущества:

1) сокращается количество одновременно отображаемых блоков на экране, что облегчает восприятие модели (в идеале модель полностью должна отображаться на экране монитора);

2) появляется возможность объединить в одну группу (подсистему) функционально связанные блоки;

3) появляется возможность создания иерархических блок-схем;

4) позволяет создавать и отлаживать фрагменты модели по отдельности, что повышает технологичность создания модели;

5) позволяет создавать собственные библиотеки;

6) дает возможность синхронизации параллельно работающих подсистем;

7) позволяет включать в модель собственные справочные средства;

8) дает возможность связывать подсистему с каким-либо m-файлом, обеспечивая запуск этого файла при открытии подсистемы (нестандартное открытие подсистемы).

Использование подсистем и механизма их блоков позволяет создавать блоки, не уступающие стандартным по своему оформлению (собственное окно параметров блока, пиктограмма, справка и т. д.).

Количество подсистем в модели не ограничено, кроме того, подсистемы могут включать в себя другие подсистемы. Уровень вложенности подсистем друг в друга также не ограничен.

Связь подсистемы с моделью (или подсистемой верхнего уровня иерархии) выполняется с помощью входных (блок **In** библиотеки **Sources**) и выходных (блок **Out** библиотеки **Sinks**) портов. Добавление в подсистему входного или выходного порта приводит к появлению на изображении подсистемы метки порта, с помощью которой внешние сигналы передаются внутрь подсистемы или выводятся в основную модель. Переименование блоков **In** или **Out** позволяет из-

менить метки портов, отображаемые на пиктограмме подсистемы со стандартных (**In** и **Out**) на те, которые нужны пользователю.

Создание подсистем

Подсистему можно создать двумя способами:

1) добавить блок **Subsystem** в модель, войти в этот блок и создать подсистему в появившемся окне подсистемы;

2) выделить часть блок-схемы модели и объединить ее в подсистему.

Создание подсистемы путем добавления блока **Subsystem**:

1) скопировать блок **Subsystem** в окно модели, перетянув его из библиотеки **Port & Systems.** После того как блок подсистемы скопирован из библиотеки в модель, он становится доступным для редактирования;

2) открыть окно блока **Subsystem**, дважды щелкнув на изображении блока в блок-схеме;

3) в пустом окне модели создать подсистему, используя блоки **In** и **Out** для входов и выходов подсистемы.

Создание подсистемы группировкой блоков

Если блок-схема содержит блоки, которые нужно объединить в подсистему, последнюю можно создать так:

1) выделить при помощи рамки блоки и соединяющие их линии, которые нужно включить в состав подсистемы;

2) выбрать **Create Subsystem** из меню **Edit**. **Simulink** заменит выделенные блоки одним блоком **Subsystem**. Чтобы увидеть блок-схему созданной подсистемы, дважды щелкнуть на блоке **Subsystem.** Simulink добавил в блок-схему блоки **In** и **Out.**

Блоки **In** и **Out** можно найти в библиотеке **Port & Systems**, а также в библиотеках **Source** и **Sink.**

*Использование блока **In** в подсистемах*

Создает входной порт для подсистемы или модели верхнего уровня иерархии.

Блоки **In** подсистемы являются ее входами. Сигнал, подаваемый на входной порт подсистемы через блок **In**, передается внутрь подсистемы. Название входного порта будет показано на изображении подсистемы как метка порта.

При создании подсистем и добавлении блока **In** в подсистему Simulink использует следующие правила:

1. При создании подсистемы с помощью команды **Edit/Create subsystem** входные порты создаются и нумеруются автоматически, начиная с **1.**

2. Если в подсистему добавляется новый блок **In**, то ему присваивается следующий по порядку номер.

3. Если какой-либо блок **In** удаляется, то остальные порты переименовываются таким образом, чтобы последовательность номеров портов была непрерывной.

4. Если в последовательности номеров портов имеется разрыв, то при выполнении моделирования Simulink выдаст сообщение об ошибке и остановит расчет. В этом случае необходимо вручную переименовать порты таким образом, чтобы последовательность номеров портов не нарушалась.

Параметры:

1. **Port number** – номер порта.
2. **Port dimensions** – размерность входного сигнала. Если этот параметр равен –1, то размерность входного сигнала будет определяться автоматически.
3. **Sample time** – шаг модельного времени.
4. **Data type** – тип данных входного сигнала:

а) auto;
б) double;
в) single;
г) int8;
д) uint8;
е) int16;
ж) uint16;
и) int32;
к) uint32;
л) Boolean.

5. **Signal type** – тип входного сигнала:

а) **auto** – автоматическое определение типа;
б) **real** – действительный сигнал;
в) **complex** – комплексный сигнал.

6. **Interpolate data** (флажок) – интерполировать входной сигнал. В случае если временные отсчеты входного сигнала, считываемого из рабочей области Matlab, не совпадают с модельным временем, то блок будет выполнять интерполяцию входного сигнала. При использовании блока Inport в подсистеме данный параметр недоступен.

*Использование блока **Out** в подсистемах*

Блок выходного порта **Out** создает выходной порт для подсистемы или для модели верхнего уровня иерархии.

Блоки **Out** подсистемы являются ее выходами. Сигнал, подаваемый в блок **Out** внутри подсистемы, передается в модель (или подсистему) верхнего уровня. Название выходного порта будет показано на изображении подсистемы как метка порта.

При создании подсистем и добавлении блока **Out** в подсистему Simulink использует следующие правила:

- При создании подсистемы с помощью команды **Edit/Create subsystem** выходные порты создаются и нумеруются автоматически, начиная с **1.**
- Если в подсистему добавляется новый блок **Out**, то ему присваивается следующий по порядку номер.
- Если какой-либо блок **Out** удаляется, то остальные порты переименовываются таким образом, чтобы последовательность номеров портов была непрерывной.
- Если в последовательности номеров портов имеется разрыв, то при выполнении моделирования Simulink выдаст сообщение об ошибке и остановит расчет. В этом случае необходимо вручную переименовать порты таким образом, чтобы последовательность номеров портов не нарушалась.

Параметры:

1. **Port number** – номер порта.
2. **Output when disabled** – вид сигнала на выходе подсистемы, в случае если подсистема выключена. Используется для управляемых подсистем. Может принимать значения (выбираются из списка):

а) **held** – выходной сигнал подсистемы равен последнему рассчитанному значению;

б) **reset** – выходной сигнал подсистемы равен значению, задаваемому параметром **Initial output.**

3. **Initial output** – начальное значение входного сигнала. Числовое значение (скаляр или вектор) или вычисляемое выражение

Настройка параметров подсистемы

Доступ к окну параметров подсистемы осуществляется через опцию **Edit** главного меню окна редактирования модели командой **Subsystem Parameters**.

Параметры:

1. **Show port labels** – показать метки портов;

2. **Read/Write permissions** – доступность подсистемы для изменений. Выбирается из списка:

а) **ReadWrite** – пользователь может открывать и изменять подсистему;

б) **ReadOnly** – пользователь может открывать подсистему только для просмотра;

в) **NoReadOrWrite** – пользователь не может открывать и изменять подсистему.

3. **Name of error callback function** – имя функции используемой для обработки ошибок возникающих в данной подсистеме.

4. **Treat as atomic unit** (флажок) – считать подсистему монолитной.

Остальные параметры подсистемы доступны при разработке приложений с использованием Real-Time Workshop.

Маскированные подсистемы

Для создания маскированной подсистемы необходимо выделить на блок-диаграмме значок подсистемы, выбрать из меню **Edit** строку **Mask Subsystem**, в окне **Mask Editor** установить свойства маскированной подсистемы. Окно **Mask Editor** имеет четыре закладки.

Закладка **Icon** позволяет настроить следующие свойства пиктограммы подсистемы:

1. **Drawing commands** – поле ввода команд, предназначенных для создания графического изображения на иконке.

2. **Icon frame** – вывод рамки:

Visible – есть рамка;

Invisible – нет рамки.

3. **Icon transparence** – прозрачность рамки:

Opaque – изображение иконки скрывает стандартный образ;

Transparent – новая иконка «прозрачная»;

Drawing coordinates – «Прозрачность» рамки;

Autoscale – размер иконки изменяется пропорционально изменению контуров блока;

Normalized – для вывода иконки устанавливается постоянный масштаб;

Pixel – значения координат x и y должны указываться в пикселах. Для автоматического масштабирования иконки в параметрах рисования необходимо указывать текущую высоту и ширину (**height** и **width**).

Initialization

Mask type – Название создаваемой маскированной подсистемы. Допускается использовать русские символы.

Окно элементов списка параметров. Позволяет добавлять элементы в список параметров подсистемы помощью кнопки **Add**. Максимальное количество элементов списка – 14.

В случае если элемент списка помечен, с ним возможны следующие действия: удалять, перемещать вверх и вниз с помощью кнопок **Delete**, **Up** и **Down** и устанавливать параметры элемента следующим образом:

Prompt – название параметра (можно по-русски);

Variable – имя переменной для записи параметра;

Control type – установка способа ввода значения параметра:

Edit – с помощью строки редактирования;

Checkbox – с помощью переключателя;

Popup – с помощью выпадающего меню.

Assignment – тип редактируемой переменной:

Evaluate – численное значение;

Literal – символьный

Popup strings – ввод элементов выпадающего меню (если **Control type** установлено **Popup**). Разделение элементов меню с помощью символа «|»

Initialization commands – ввод списка команд инициализации маски. Команды инициализации оперируют с переменными, находящимися в рабочей области подсистемы (**Mask Workspace**). Они представляют собой обычные команды системы **Matlab**, т. е. поле **Initialization commands** можно рассматривать, как командное окно системы **Matlab**, область действия которого ограничена рабочей областью маскированной системы.

Команды инициализации выполняются в следующих случаях:

– при открытии блок-диаграммы модели;

– запуске модели на исполнение;

– обновлении блок-диаграммы (по команде **Update diagram**);

– вращении блока маскированной подсистемы (с целью перерисовки иконки);

– для автоматического изменения иконки, зависящей от параметров блока.

5.5. Задания для лабораторной работы

Необходимо преобразовать сценарии, составленные при выполнении лабораторной работы № 4, в самостоятельный программный файл с расширением «exe». Для этого необходимо скомпилировать программный код, но такой компиляция применима только к функциям Matlab. Прежде всего перепишем уже готовые сценарии в форме функций. Пусть функция для ввода данных называется «**VecInput**», сохраненная в файле «**VecInput.m**» она будет возвращать вектор, который оператор введет при работе с программой, и вектор, рассчитанный функцией, т. е. получится два выходных параметра. Вторая функция – «**MyFunc**» в файле «**MyFunc.m**», эта функция будет иметь два входных параметра – векторы, которые возвращает функция «**VecInput**», выходными параметрами будут результаты ее расчета (четыре выходных параметра):

– количество элементов больших 100;

– сумма отрицательных элементов;

– количество неотрицательных и не превышающих 100 элементов **y** больших, чем элементы **x** с тем же индексом;

– сумма остальных элементов.

5.5.1. Программирование m-функции, не имеющей входных и выходных аргументов

Составим главную функцию, из которой с помощью меню можно будет вызывать другие функции.

Строка меню:

```
k=menu('Главное меню',...
        'Ввод элементов массива',...
        'Просмотр элементов массива',...
        'Просмотр элементов массива функции',...
        'Вычисления',...
        'Количество элементов больших 100',...
        'Сумма отрицательных элементов',...
        'Количество элементов y больших, чем x',...
        'Сумма остальных элементов',...
        'Конец работы');
```

Так как функция имеет длинный список параметров, то для удобства применен специальный знак «…», позволяющий перенос на

новую строку. При воспроизведении программы всплывающее меню будет выглядеть как на рисунке 5.3.

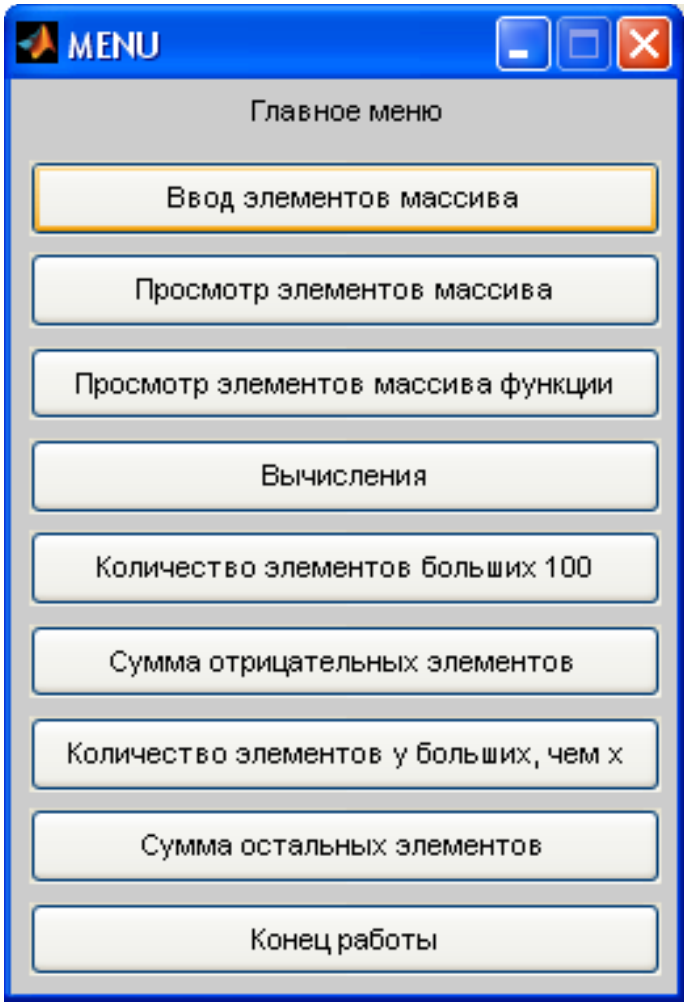


Рисунок 5.3 – Всплывающее меню

Входной параметр «*Главное меню*» отображается как заголовок меню, остальные параметры – надписи на кнопках. Число кнопок равно числу строк минус 1. При этом, если оператор нажмет кнопку меню, то переменная **k** примет значение, равное номеру кнопки, например, если это будет первая кнопка с надписью: «*Ввод элементов массива*», переменная **k** примет значение **1**.

Этим свойством функции «*menu*» воспользуемся для организации выбора из меню с помощью ветвления «**if…elseif…else**». Для начала строки выбора будут выглядеть, как на рисунке 5.4.

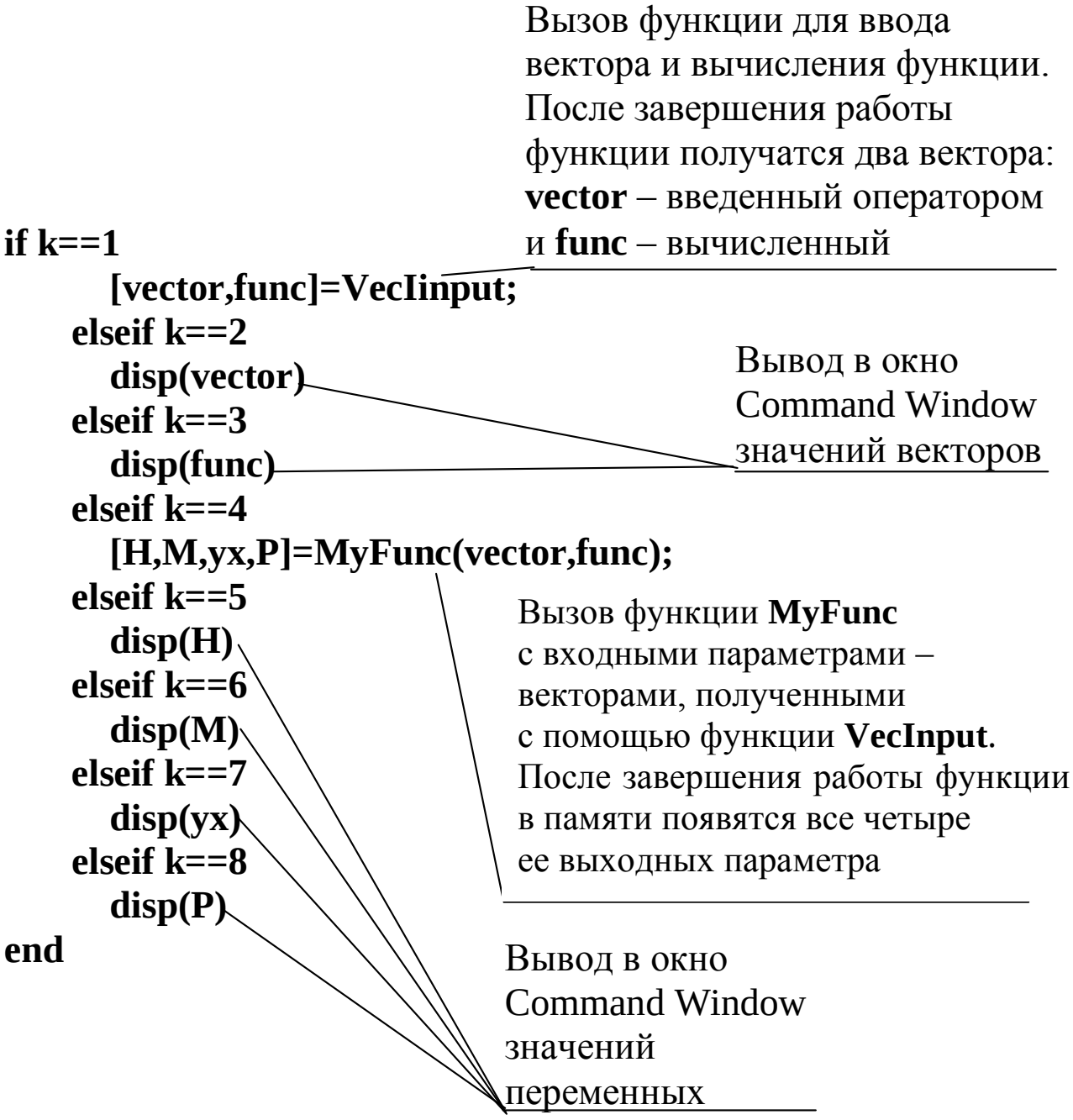


Рисунок 5.4 – Организации выбора из меню

Весь текст функции смотри на рисунке 5.5.

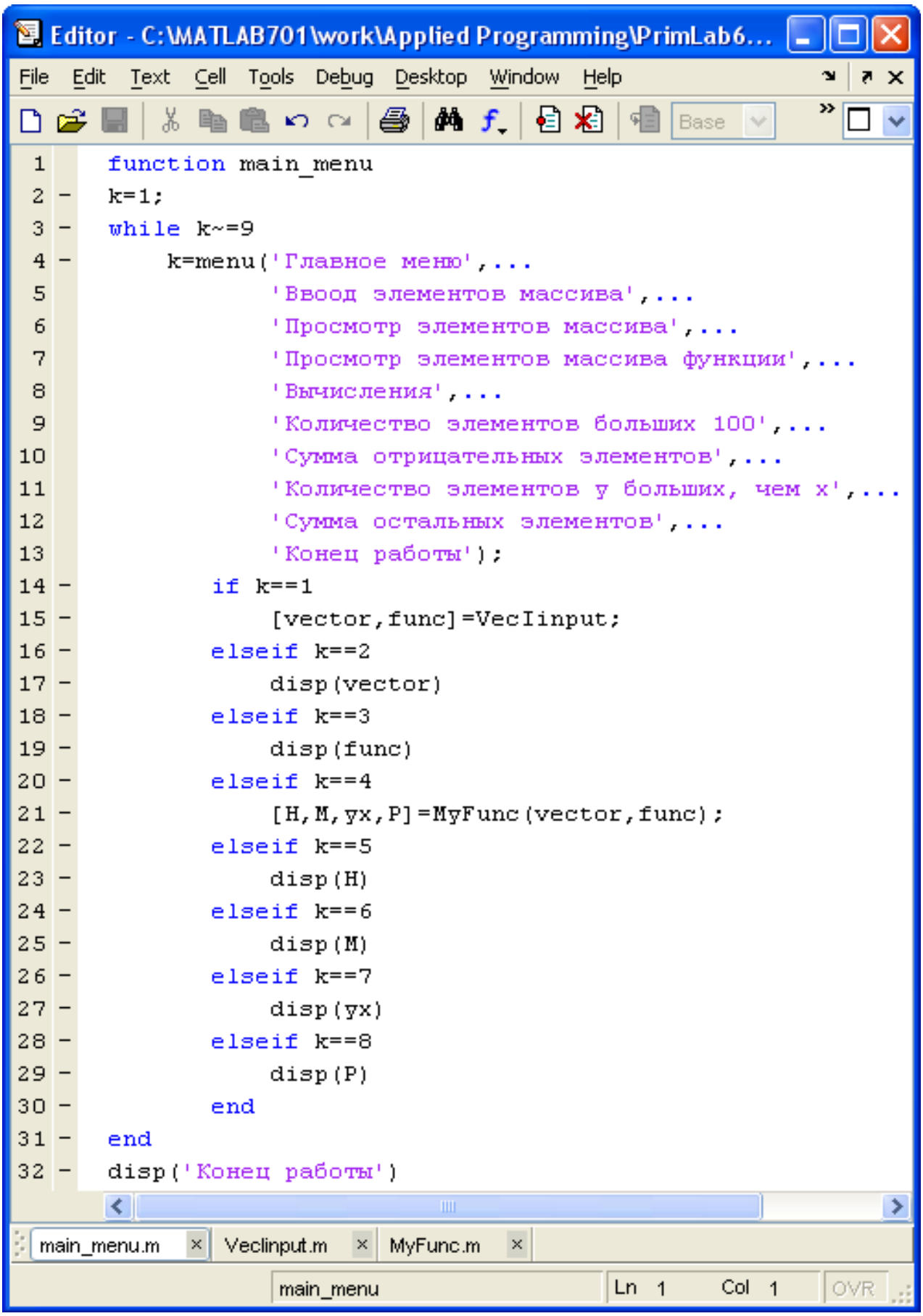


Рисунок 5.5 – Текст функции выбора

Цикл «**while k~=9**» будет повторяться после любого нажатия кнопки и соответствующего выбора с помощью «**if…elseif…else**», кроме случая нажатия кнопки № 9 с надписью «*Конец работы*». В этом случае переменная **k** примет значение **9**, оператор сравнения «**k~=9**» вернет значение «**ложь**», управление программой перейдет к строке, стоящей после цикла, т. е. «**disp(**'*Конец работы*'**)**». Затем работа программы будет закончена.

Перед началом цикла необходимо, чтобы переменная **k** имела некоторое значение, так как цикл начинается со сравнения **k** константой **9**, поэтому перед оператором «**while k~=9**» введем выражение: «**k=1**» (значение, присвоенное переменной **k** может быть любым, кроме 9, так как в этом случае выражение **k~=9** будет равно «**ложь**», и цикл не будет выполняться).

5.5.2. Программирование m-функции с выходными аргументами

При выборе первой кнопки меню с надписью «Ввод элементов массива» переменная **k** примет значение **1**, оператор «**if k==1**» будет равен «**истина**», строка «**[vector,func]=VecIinput**» вызовет функцию «**VecIinput**» из файла «**VecIinput.m**». Эта функция должна обеспечить возможность ввода оператором элементов вектора и вычислить математическое выражение от всех элементов вектора, а также вернуть в вызывающую программу **введенный вектор** в качестве **первого** выходного параметра и **вычисленный вектор** в качестве **второго** выходного параметра. Сценарий ввода элементов вектора и вычисления математического выражения, составленный при выполнении лабораторной работы № 4, преобразуем в функцию, приведенную на рисунке 5.6. На рисунке стрелками показаны переменные, с которыми работает программа, и их положение в списке выходных параметров функции. Вводимый оператором вектор **x** стоит первым в списке параметров, поэтому при вызове данной функции строкой «**[vector,func]=VecIinput**» в первый параметр «**vector**» будет возвращено значение вектора **x**, во второй параметр «**func**» – значение второго выходного параметра **y**.

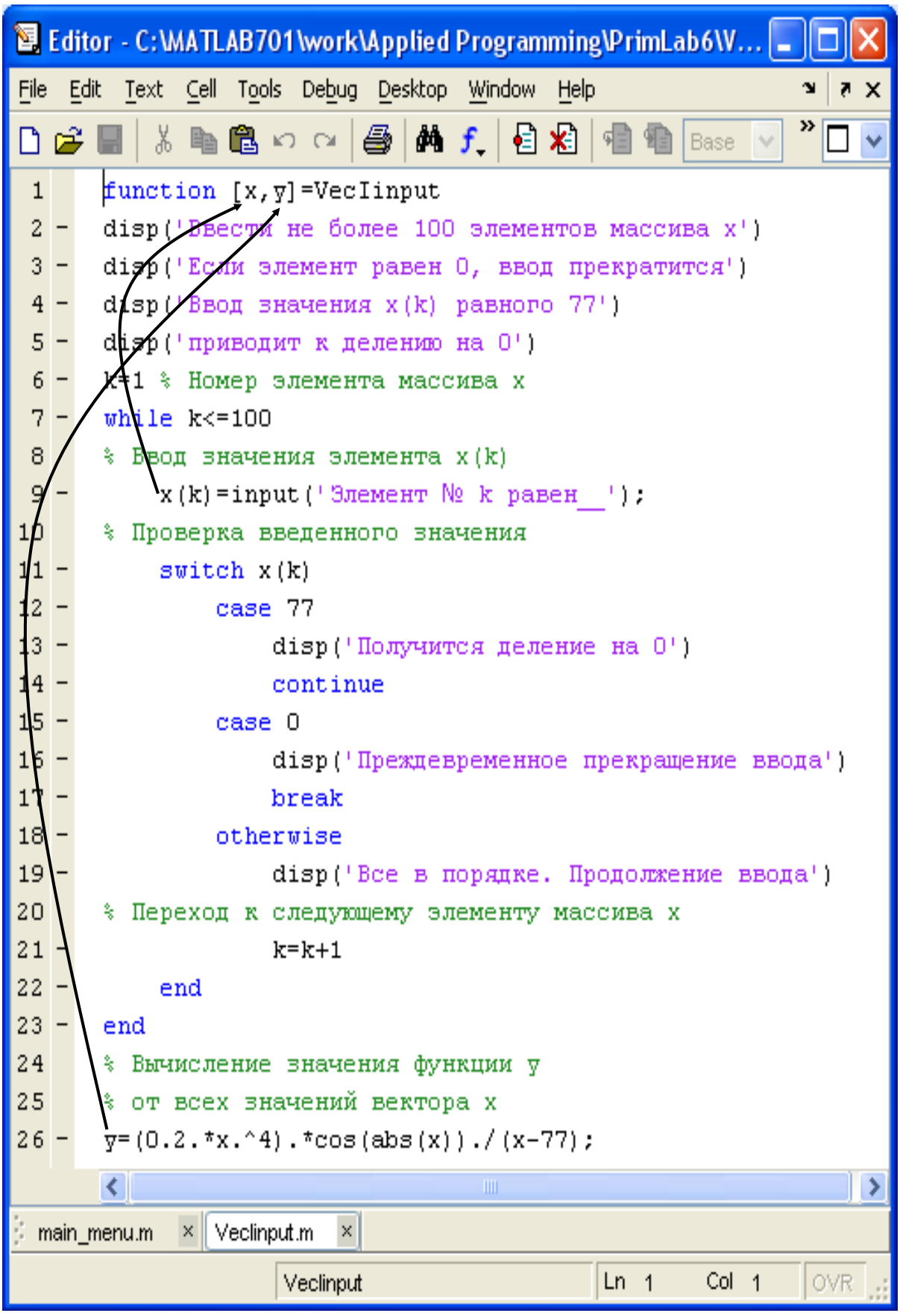


Рисунок 5.6 – Переменные, работающие в программе, и их положение в списке выходных параметров функции

5.5.3. Программирование М-функции с входными и выходными аргументами и внутренней функцией

При выборе четвертой кнопки меню с надписью «Вычисления» переменная **k** примет значение **4**, оператор «**elseif k==4**» будет равен «**истина**», строка «**[H,M,yx,P]=MyFunc(vector,func)**» вызовет функцию **MyFunc** из файла «**MyFunc.m**». Эта функция должна получить в качестве двух своих входных параметров вектор **x** (первый параметр: в вызывающей программе нужный вектор имеет идентификатор «**vector**», поэтому при вызове функции **vector** – первый входной параметр) и **y** (второй – сектор «**func**» в вызывающей программе), затем вычислить две суммы и два количества элементов вектора **y** (см. задание к лабораторной работе № 4,), вернуть в вызывающую программу **вычисленные значения** в качестве выходных параметров. Сценарий из лабораторной работы № 4 преобразуем в функцию, приведенную на рисунке 5.7.

Итак, входные параметры вызова функции **vector** и **func** воспринимаются в тексте функции как **x** и **y** соответственно согласно порядку их следования.

В списке возвращаемых параметров первым стоит **Hundred**, поэтому при вызове функции его значение попадет в переменную вызывающей программы **H**, стоящую первой при вызове. Согласно этому принципу значение выходного параметра **Msum** попадет в **M**, **y_more_x** – в **yx**, а **Psum** – в **P**.

Для вывода сообщений в функции **MyFunc** организуем внутреннюю функцию:

```
function dispmess(mess,res)
disp(mess);
disp(res);
```

Эта функция отображает на экране содержимое своих входных параметров **mess** и **res**. При первом вызове функции **dispmess** на ее вход будут поданы строка «**Количество элементов больших 100**» и численная переменная **Hundred**.

При втором вызове – «Сумма отрицательных элементов» и **Msum**.

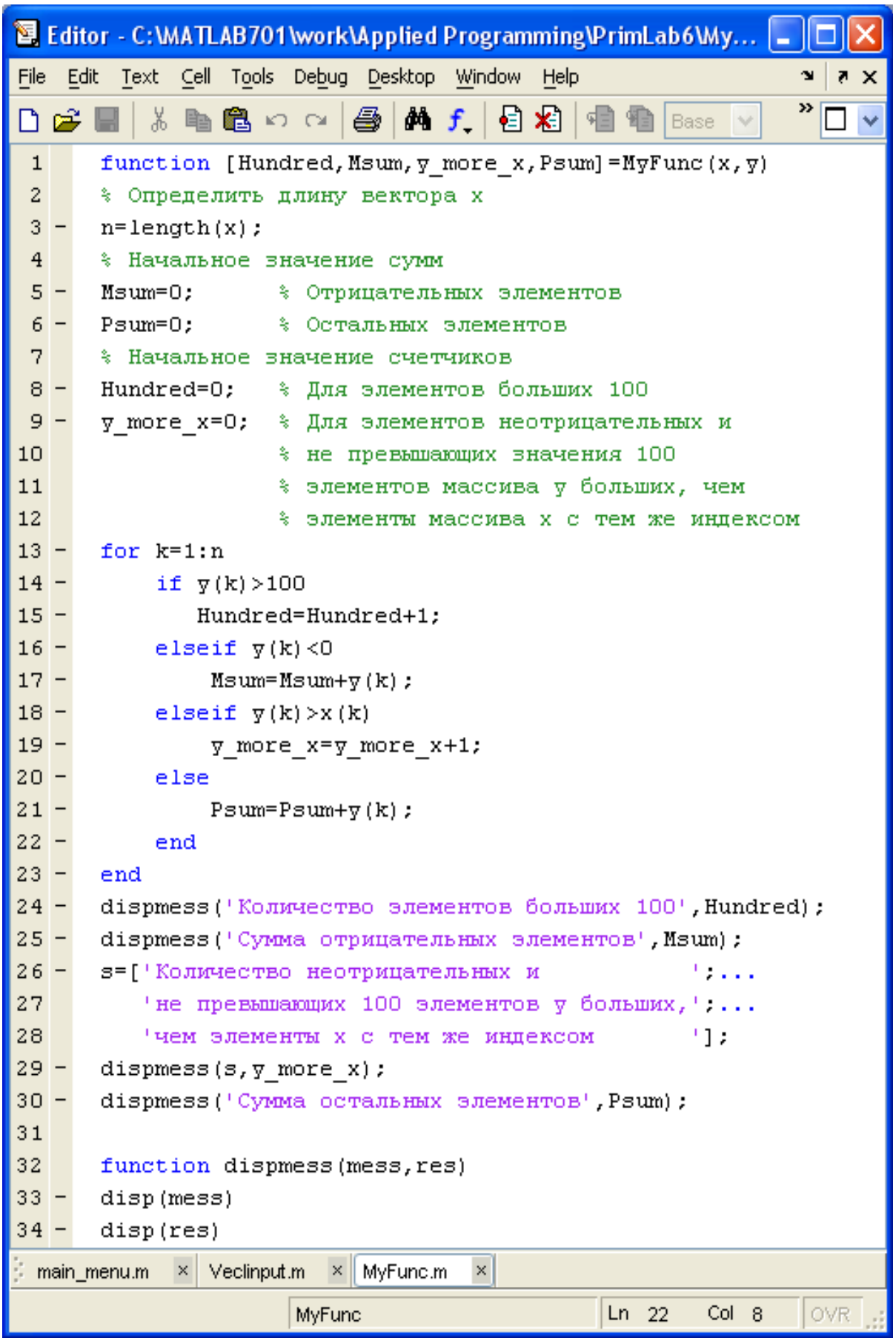


Рисунок 5.7 – Сценарий из лабораторной работы № 4, преобразованный в функцию

При третьем вызове первым параметром будет двумерный символьный массив **s**. Так как длины строк матрицы должны быть одинаковыми, то выровняем их за счет пробелов, символ «**;**» отделяет одну строку от другой, перенос текста программы на следующую строку – символ «**…**». Второй параметр **y_more_x**.

Последний вызов с входными параметрами: «**Сумма остальных элементов**» и **Psum**.

Результат всех четырех вызовов смотри на рисунке 5.8.

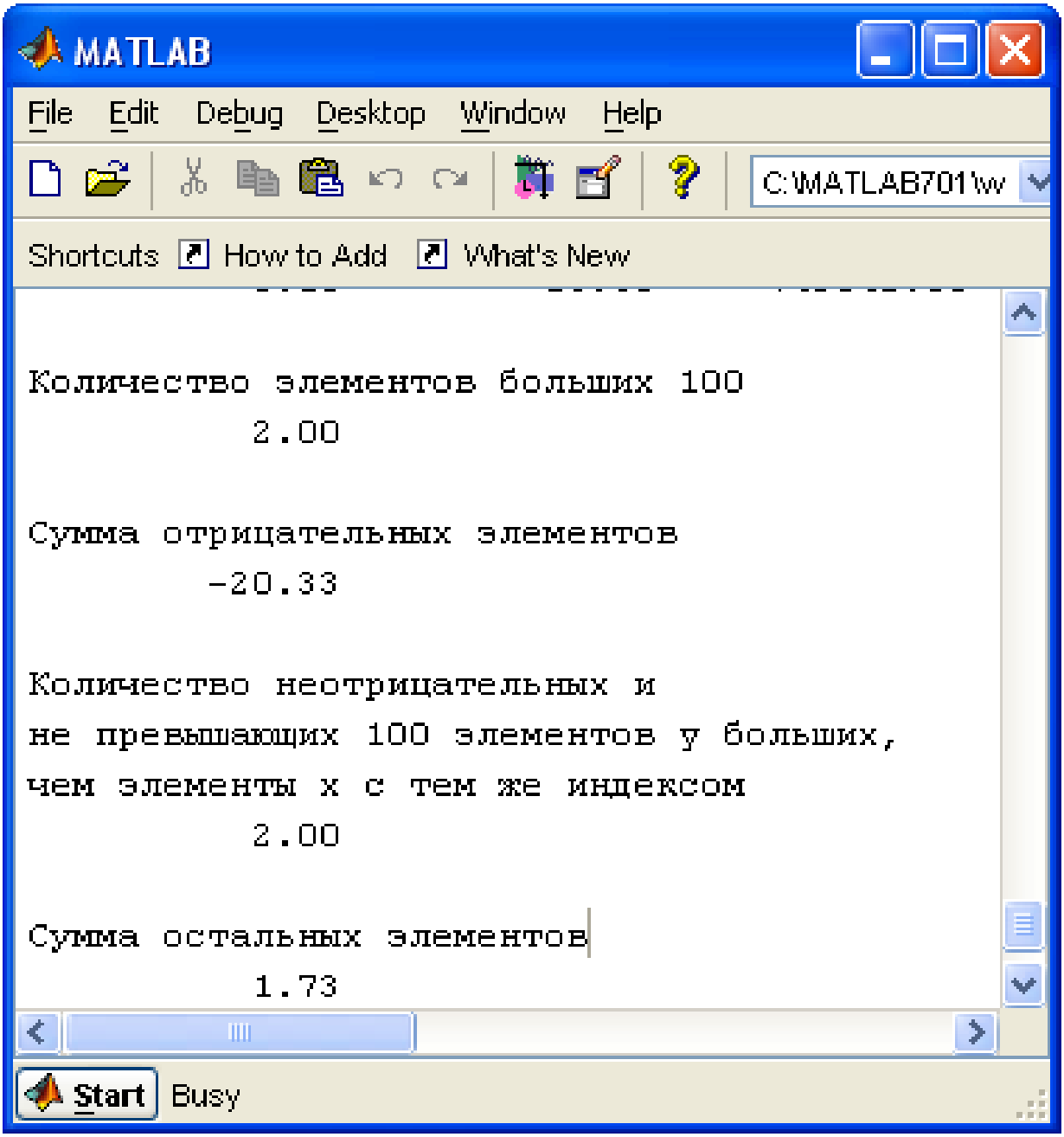


Рисунок 5.8 – Результат всех четырех вызовов

*Графические функции отображения данных в функции «**main_menu**»*

Функции Matlab можно скомпилировать в выполняемый файл с расширением «**exe**» с помощью встроенной функции **mcc**. Полученный файл будет работать без среды Matlab, поэтому функции **disp** и **input**, использующие окно Command Window, необходимо заменить графическими функциями вывода и ввода данных.

Для вывода сообщений в функции **main_menu** применим графическую функцию **warndlg(сообщение, заголовок окна)**. Рассмотрим работу этой функции, поставив ее вместо **disp ('Конец работы')** в конце программы:

warndlg('До свидания', 'Конец работы')

Результат работы этой графической функции смотри на рисунке 5.9.

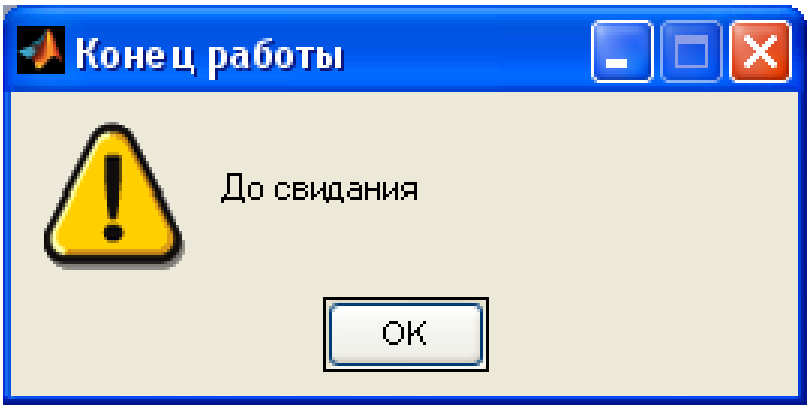


*Рисунок 5.9 – Результат работы функции **warndlg** ('До свидания', 'Конец работы')*

Из рисунка 5.9 видно, что первая строка «*До свидания*» отображается как сообщение, вторая «*Конец работы*» – как заголовок окна. Окно является модальным, т. е. для продолжения работы оно должно быть закрыто нажатием кнопки «**OK**».

В остальных случаях функция **disp** выводила численные данные, поэтому при замене ее на функцию **warndlg** необходимо преобразовать численный массив в строку, применив функцию **num2str**. Например, массив:

vector=[-10 6 9 15 8 -11 14 100 0]

преобразуем

num2str(vector)

результат – строка символов ‘**-10 6 9 15 8 -11 14 100 0**’

Пусть окна будут без заголовков, для этого вторую строку сделаем пустой. Пример вызова функции **warndlg** вместо **disp(vector)**:

warndlg(num2str(vector),'')

Результат смотри на рисунке 5.10.

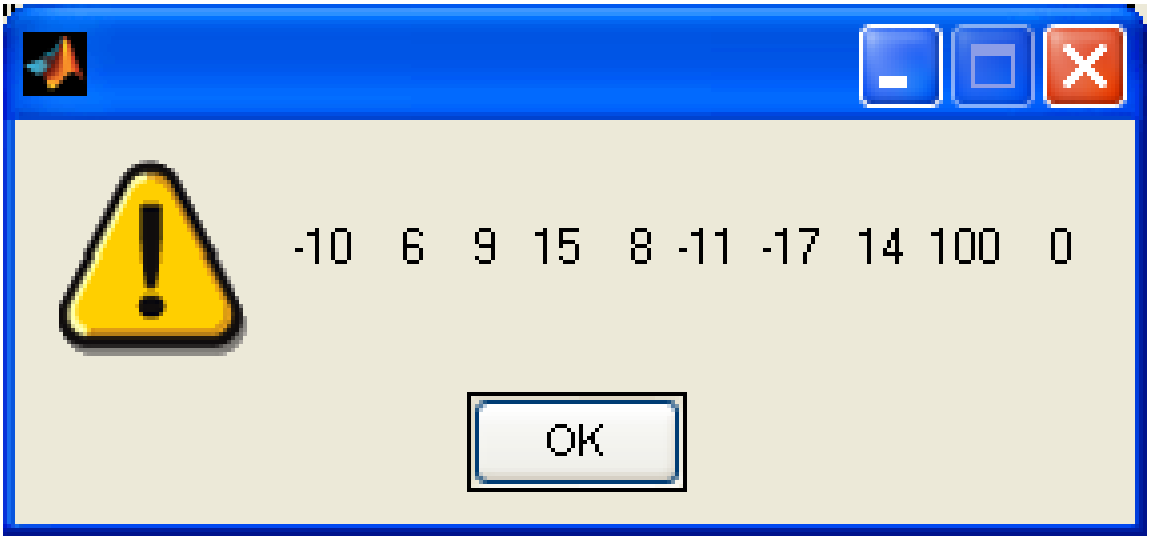


Рисунок 5.10 – Результат работы функции ***warndlg(num2str(vector),'')***

Остальные замены функции **disp** на функцию **warndlg**:

– вместо disp(func) функция warndlg(num2str(func),");
– вместо disp(H) функция warndlg(num2str(H),");
– вместо disp(M) функция warndlg(num2str(M),");
– вместо disp(yx) функция warndlg(num2str(yx),");
– вместо disp(P) функция warndlg(num2str(P),").

Весь текст функции смотри на рисунке 5.11.

Графические функции ввода и отображения данных в функции ***VecInput***

В функции **VecInput** сообщение **disp**('Преждевременное прекращение ввода') заменим на

h = warndlg('Конец ввода',...
'Преждевременное прекращение ввода')

Сообщение **disp**('Получится деление на 0') заменим на

h = errordlg('Получится деление на 0',...
'Ошибочный ввод')

Работа данной функции аналогична предыдущей, другой будет только пиктограмма в окне (рис. 5.12).

Рисунок 5.11 – Текст функции

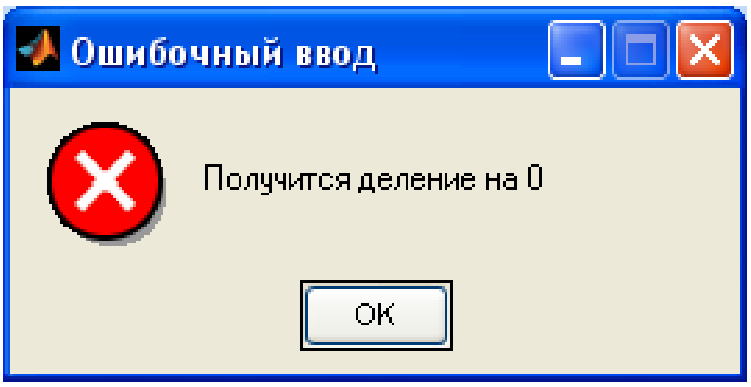


*Рисунок 5.12 – Сообщение **disp**('Получится деление на 0')*

Осталось заменить функцию **input** на графическую функцию ввода **inputdlg**.

s=inputdlg('To enter number','Ввод данных')

Получится окно для ввода числа (рис. 5.13).

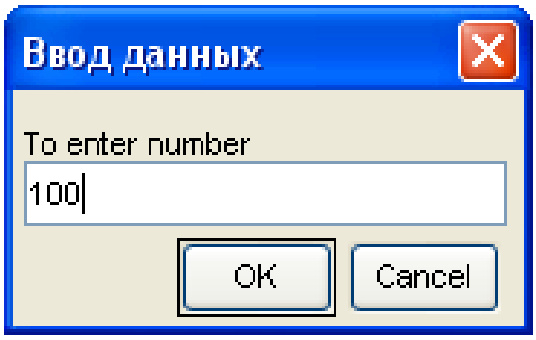


*Рисунок 5.13 – Функция **s=inputdlg**('To enter number', 'Ввод данных')*

Данный диалог является модальным, поэтому для продолжения работы программы оператору необходимо ввести число в поле редактора и нажать одну из кнопок: **Ok** или **Cancel**.

Символы, введенные оператором, составляют строку. Чтобы получить элемент численного массива **y**, необходимо преобразовать строку в число (естественно, строка должна состоять из подходящих для этого символов, а именно (допустимые символы): «0», «1», «2», «3», «4», «5», «6», «7», «8», «9» и точка для отделения целой части числа от дробной, а также пробел, запятая, точка с запятой и квадратные скобки для ввода массива данных)

x(k)=str2num(s{1})

Так как вводимые данные при использовании данной функции являются массивом ячеек, то необходимо указать индекс ячейки в фигурных скобках (в данном случае ячейка одна и поэтому она первая). В лабораторной работе, посвященной вопросам ввода данных, описаны варианты использования функции **inputdlg**.

После предлагаемых преобразований получится функция, текст которой приведен на рисунке 5.14.

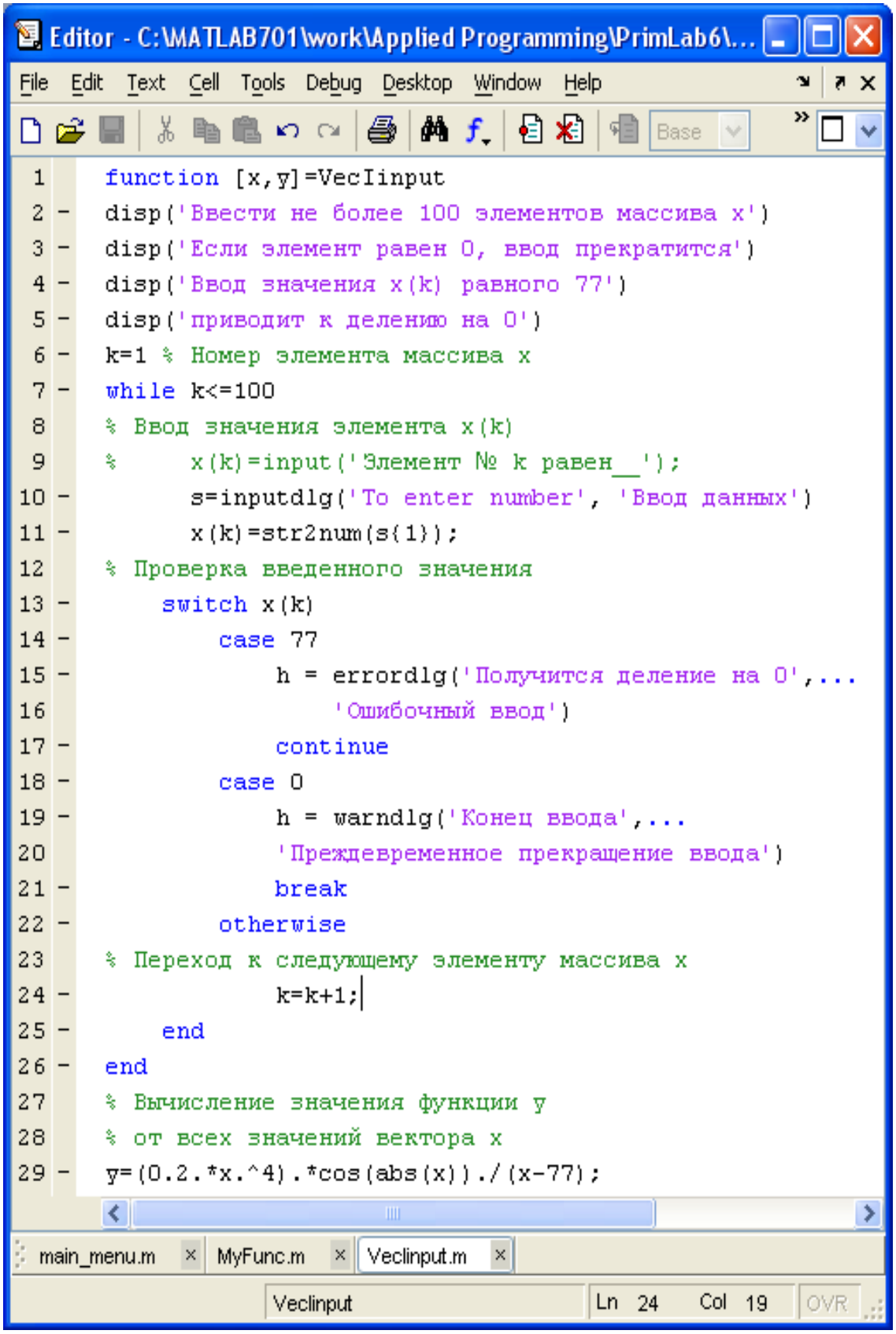


Рисунок 5.14 – Использование функции ***inputdlg***

*Графические функции отображения данных в функции **MyFunc***

В функции **MyFunc** также необходимо заменить функцию **disp** на графическую функцию.

Сделаем это с помощью функции **uicontrol**, позволяющей программировать различные элементы управления: кнопки, меню, редактор и др. Запрограммируем статический текст:

uicontrol('style','text','position',[10 h 300 100],'string',mess)

Рассмотрим работу этой функции:

– пара параметров «style» и «text» обеспечат стиль статического текста;

– пара «**position**» и «**[10 h 300 100]**» описывают позицию текста в окне: от левого нижнего угла влево 10 пикселов, вверх **h** пикселов. Параметр **h** будем добавлять при вызове внутренней функции «**dispmess**», заботясь о том, чтобы каждый следующий вызов функции обеспечивал позицию вывода текста ниже, чем предыдущий вывод. Соответственно в заголовок функции также необходимо добавить входной параметр **h. 300** и 100 – ширина и высота текста;

– пара «**string**» и «**mess**» – указание на конкретную строку, которая будет выведена как статический текст.

В качестве параметра mess в нашей программе применяются строки сообщений.

Второй случай данной функции:

uicontrol('style','text',…

'position',[300 h 100 100],'string',num2str(res))

Отличия этого вызова от предыдущего заключаются в следующем:

– **res** – это численные данные, поэтому для преобразования числа в строку использована функция **num2str(res)**;

– Отступ позиции начала вывода текста от левого нижнего угла **300** пикселов, что левее предыдущего текста на **290** пикселов.

Результат всех четырех вызовов функции **dispmess(mess,res,h)** смотри на рисунке 5.15.

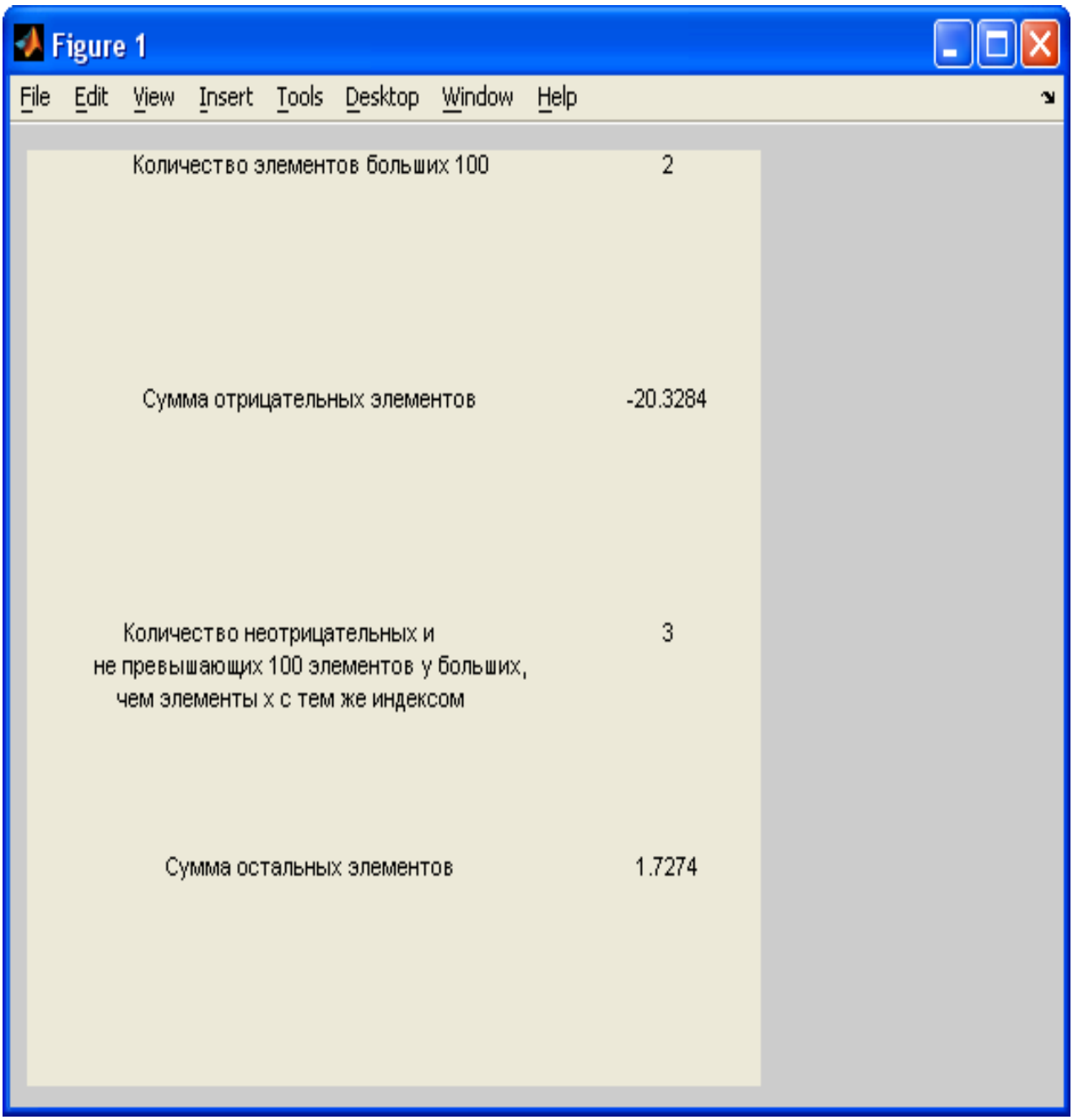
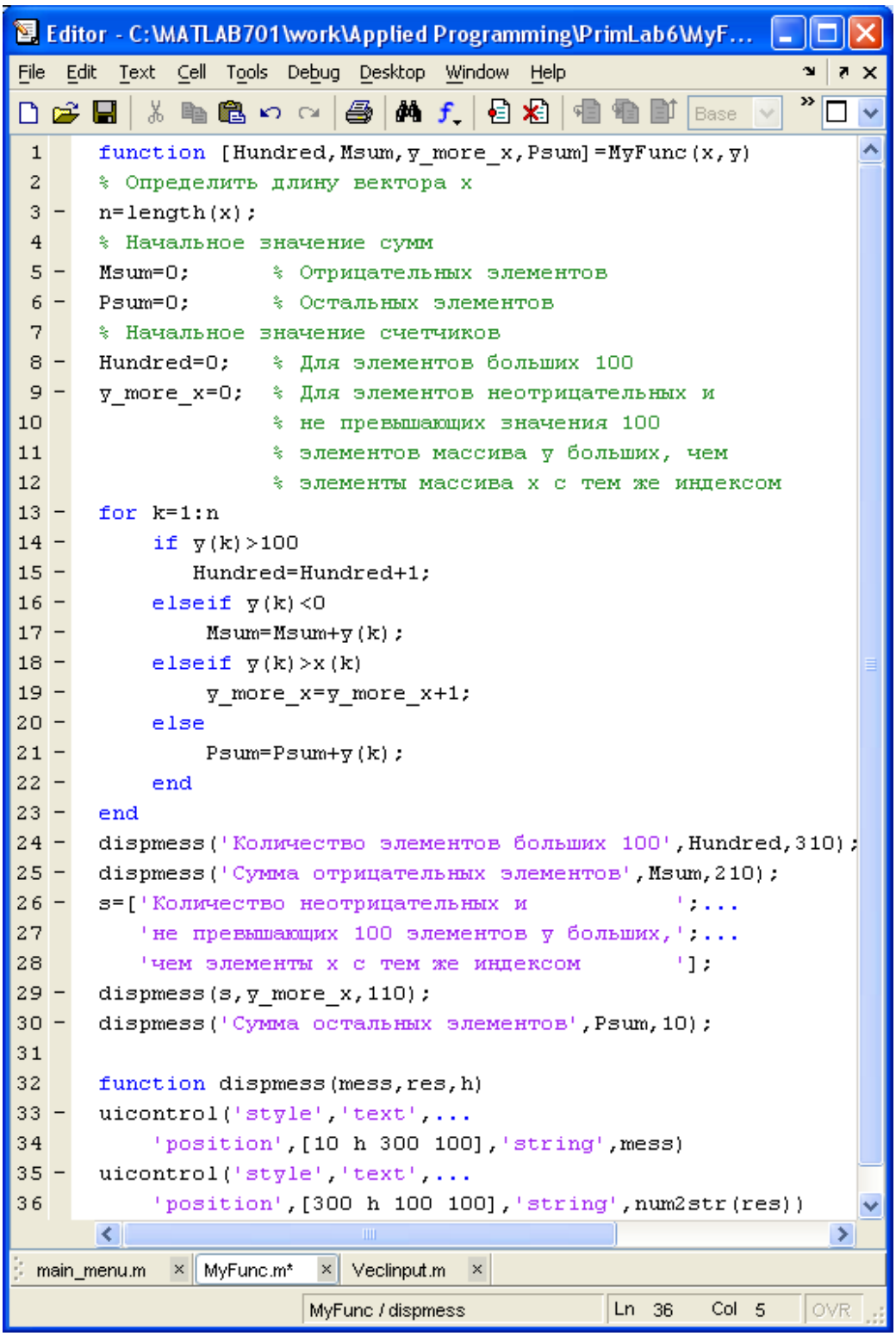


*Рисунок 5.15 – Вызовов функции «**dispmess(mess,res,h)***

После предлагаемых преобразований текст функции **MyFunc** смотри на рисунке 5.16.




```
function [Hundred,Msum,y_more_x,Psum]=MyFunc(x,y)
% Определить длину вектора x
n=length(x);
% Начальное значение сумм
Msum=0;       % Отрицательных элементов
Psum=0;       % Остальных элементов
% Начальное значение счетчиков
Hundred=0;    % Для элементов больших 100
y_more_x=0;   % Для элементов неотрицательных и
              % не превышающих значения 100
              % элементов массива y больших, чем
              % элементы массива x с тем же индексом
for k=1:n
    if y(k)>100
       Hundred=Hundred+1;
    elseif y(k)<0
        Msum=Msum+y(k);
    elseif y(k)>x(k)
        y_more_x=y_more_x+1;
    else
        Psum=Psum+y(k);
    end
end
dispmess('Количество элементов больших 100',Hundred,310);
dispmess('Сумма отрицательных элементов',Msum,210);
s=['Количество неотрицательных и           ';...
   'не превышающих 100 элементов y больших,';...
   'чем элементы x с тем же индексом       '];
dispmess(s,y_more_x,110);
dispmess('Сумма остальных элементов',Psum,10);

function dispmess(mess,res,h)
uicontrol('style','text',...
    'position',[10 h 300 100],'string',mess)
uicontrol('style','text',...
    'position',[300 h 100 100],'string',num2str(res))
```

Рисунок 5.16 – Текст функции ***MyFunc***

Компиляция файла «main_menu.exe»

Для получения независимого выполняемого файла с расширением «exe» применим функцию компиляции в форме:

mcc -m mcc -m main_menu MyFunc VecIinput

mcc – функция;

m – ключ компиляции;

main_menu MyFunc VecIinput – список функций, которые необходимо включить в файл.

Получится файл «main_menu. exe», так как имя функции «**main_menu**» стояло первым в списке.

В процессе компиляции получились временные файлы на языке С «main_menu_main.c» и «main_menu_mcc_component_data.c», которые можно удалить.

Файл «main_menu.ctf» и папка с файлами «main_menu_mcr» необходимы для работы программы. При переносе файла «main_menu.exe» в другую папку или на другой компьютер они должны быть перенесены вместе с ним.

Тестовые вопросы:

– программирование **M**-функции, не имеющей входных и выходных аргументов;

– программирование **M**-функции с выходными аргументами;

– программирование **M**-функции, с входными и выходными аргументами и внутренней функцией;

– графические функции отображения данных warndlg;

– графические функции отображения данных errordlg;

– графические функции ввода данных в функции inputdlg;

– графические функции отображения данных uicontrol ('Style', 'Text',…);

– компиляция независимого файла с расширением «exe».

Контрольные вопросы

1. Порядок создания **M**-файла, требования к имени **M**-файла.
2. Что такое **M**-функция?
3. Структура **M** функции.
4. Привести пример **M**-функции.
5. Вызов **M**-функции.
6. Пользовательская функция в Simulink.
7. Внутренние функции.
8. Подсистемы в моделях Simulink.
9. Порядок создания подсистемы в модели Simulink.
10. Использование блока In в подсистемах.
11. Использование блока Out в подсистемах.
12. Настройка параметров подсистемы.
13. Маскированные подсистемы.

ЗАКЛЮЧЕНИЕ

В учебном пособии по дисциплине «Моделирование в агроинженерии» автор попытался найти доступные формы изложения сложного материала, познакомить с основными способами моделирования и показать необходимость их познания.

Учебное пособие задумано как самоучитель по математическому моделированию в программе Matlab. В каждом из пяти разделов в доступной форме кратко представлен теоретический и практический материал, даны контрольные вопросы для проверки знаний и задания для самостоятельной работы.

Учебное пособие написано и построено таким образом, чтобы студент самостоятельно мог разобраться в терминах, понятиях, теории вопроса и других нюансах дисциплины. Автор полагает, что такое изложение и расположение материала будет способствовать его лучшему усвоению.

Учебное пособие предназначено главным образом для студентов вузов и лиц, самостоятельно изучающих методы математического моделирования и программирования в программе Matlab. Оно может также оказаться полезным преподавателям, руководителям предприятий и специалистам в энергетике. Автор будет благодарен читателям за замечания и предложения как по улучшению содержания учебного пособия, так и по форме изложения материала.

ЛИТЕРАТУРА

1. Андриевский, Б. Р. Избранные главы теории автоматического управления с примерами на языке MATLAB / Б. Р. Андриевский, А. Л. Фрадков. – Санкт-Петербург: Наука, 1999.

2. Говорухин, В. Компьютер в математическом исследовании: Maple, MATLAB, LaTeX / В. Говорухин, В. Цибулин. – Санкт-Петербург: Питер, 2001.

3. Гультяев, А. К. Визуальное моделирование в среде Matlab: учебник / А. К. Гультяев. – Санкт-Петербург: Питер, 2000.

4. Дьяконов, В. П. MATLAB. Полный самоучитель / В. П. Дьяконов. – Москва: ДМК Пресс, 2012. – 768 с.

5. Дьяконов, В. П. MATLAB 6/6.1/6.5 + Simulink 4/5. Основы применения. Полное руководство пользователя / В. П. Дьяконов. – Санкт-Петербург: Солон-Пресс, 2002.

6. Дьяконов, В. П. MATLAB 6: учебное пособие / В. П. Дьяконов. – Санкт-Петербург: Питер, 2001. – 560 с.

7. Дьяконов, В. П. MATLAB с пакетами расширений / В. П. Дьяконов, И. В. Абраменкова, В. В. Круглов. – Москва: Нолидж, 2001.

8. Дьяконов, В. П. MATLAB. Анализ, идентификация и моделирование систем: специальный справочник / В. П. Дьяконов, В. В. Круглов. – Санкт-Петербург: Питер, 2002. – 448 с.

9. Дьяконов, В. П. Математические пакеты расширения MATLAB: специальный справочник / В. П. Дьяконов, В. В. Круглов. – Санкт-Петербург: Питер, 2001. – 480 с.

10. Кондрашов, В. Е. Matlab как система программирования научно-технических расчетов / В. Е. Кондрашов, С. Б. Королев. – Москва: Мир, 2002.

11. Мартынов, Н. Н. Введение в MATLAB 6 / Н. Н. Мартынов. – Москва: Кудиц-образ, 2002.

12. Мэтьюз, Д. Г. Численные методы. Использование MATLAB / Д. Г. Мэтьюз, К. Д. Финк. – Москва: Вильямс, 2001.

13. Потемкин, В. Г. MATLAB 6: среда проектирования инженерных приложений / В. Г. Потемкин. – Москва: Диалог-МИФИ, 2003.

14. Потемкин, В. Г., Введение в MATLAB / В. Г. Потемкин. – Москва: Диалог-МИФИ, 2000.

15. Потемкин, В. Г. Вычисления в среде MATLAB / В. Г. Потемкин. – Москва: Диалог-МИФИ, 2004.

16. Потемкин, В. Г. Система MATLAB: справочное пособие / В. Г. Потемкин. – Москва: Диалог-МИФИ, 1997.

17. Ткачев, Н. Н. Статистические методы в математическом моделировании и научных исследованиях: лабораторный практикум / Н. Н. Ткачев; Красноярский государственный аграрный университет. – Красноярск, 1996. – 148 с.

18. Чен, К. MATLAB в математических исследованиях / К. Чен, П. Джиблин, А. Ирвинг. – Москва: Мир, 2001.

19. Черных, И. В. Simulink: среда создания инженерных приложений / И. В. Черных. – Москва: Диалог-МИФИ, 2003.

ОГЛАВЛЕНИЕ

МОДЕЛИРОВАНИЕ В АГРОИНЖЕНЕРИИ

Учебное пособие

СЕМЕНОВ Александр Федорович

Электронное издание

Редактор М. М. Ионина

Подписано в свет 07.10.2024. Регистрационный номер 44
Редакционно-издательская служба Красноярского государственного аграрного университета
660017, Красноярск, ул. Ленина, 117
e-mail: rio@kgau.ru