

**МИНИСТЕРСТВО ТРАНСПОРТА
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Российский университет транспорта (МИИТ)»**

Кафедра «Электропоезда и локомотивы»

Е.К. Рыбников, Е.В. Сердобинцев, С.В. Володин

**ПРОГРАММИРОВАНИЕ В СРЕДЕ TURBO BASIC
И ПРОГРАММНОМ ПАКЕТЕ MATHCAD**

Учебное пособие по дисциплине «Информатика»

Москва – 2018

**МИНИСТЕРСТВО ТРАНСПОРТА
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Российский университет транспорта (МИИТ)»**

Кафедра «Электропоезда и локомотивы»

Е.К. Рыбников, Е.В. Сердобинцев, С.В. Володин

**ПРОГРАММИРОВАНИЕ В СРЕДЕ TURBO BASIC
И ПРОГРАММНОМ ПАКЕТЕ MATHCAD**

**Учебное пособие
для студентов специальности
23.05.03 «Подвижной состав железных дорог»,
специализация
«Электрический транспорт железных дорог»**

Москва – 2018

УДК 681.3.06.5

Р93

Рыбников Е.К., Сердобинцев Е.В., Володин С.В.
Программирование в среде Turbo Basic и программном пакете Mathcad: Учебное пособие по дисциплине «Информатика». – М.: РУТ (МИИТ), 2018. – 116 с.

В учебном пособии приведены основы работы в среде Turbo BASIC и программном пакете MathCad. Даны примеры блок схем и программ на языке программирования BASIC. Рассказано о наиболее часто употребляемых операторах профессиональной версии языка BASIC. Рассмотрены примеры применения программной среды MathCad для решения типовых задач, представлены материалы для расширенного ознакомления с программным пакетом.

Рецензенты:

Профессор кафедры «Управление и защита информации»
РУТ (МИИТ), д.т.н. Сидоренко В.Г.

Главный специалист отдела новых локомотивов
Департамента технической политики ОАО «РЖД»
Углянкин Д.М.

© РУТ (МИИТ), 2018

СОДЕРЖАНИЕ

1	Процесс постановки и решения задач.....	5
1.1	Этапы решения задачи.....	5
1.2	Понятие об алгоритме решения задачи.....	6
1.3	Реализация алгоритмов с помощью языков программирования	12
2	Алгоритмический язык BASIC	14
2.1	Общие сведения о среде TurboBASIC.....	14
2.2	Команды и подкоманды главного меню оболочки Turbo BASIC. Написание программы	14
2.3	Набор текста программы	16
2.4	Понятие имени переменной	16
2.5	Команды присваивания	18
2.6	Операторы вывода результата	19
2.7	Понятие встроенных функций	22
2.8	Построение сложных вычислений.....	24
2.9	Операторы условного и безусловного переходов	25
2.10	Оператор безусловного перехода	27
2.11	Организация циклических вычислений	28
2.12	Применение массивов.....	33
2.13	Организация файлов последовательного и прямого доступа	36
2.14	Работа со строковыми переменными	42
2.15	Применение графических средств.....	46
2.16	Организация подпрограмм и подпрограмм- процедур	52
2.17	Нестандартные математические функции, вводимые пользователем	56
2.18	Типовые задачи	57
3	Программный пакет MathCad	68

3.1 Основные понятия.....	68
3.2 Начало работы с MathCad	69
3.3 Работа с документами.....	70
3.4 Запись математических выражений	72
3.5 Структура документа	73
3.6 Ввод математических выражений	75
3.7 Ввод текста	77
3.8 Операторы присваивания	77
3.9 Понятие переменной	79
3.10 Вывод результатов	82
3.11 Построение графиков.....	84
3.12 Программирование в MathCad.....	89
3.13 Оператор условного перехода.....	89
3.14 Операторы циклических вычислений	92
3.15 Оператор Break.....	94
3.16 Программа-константа	95
3.17 Программа-переменная:	95
3.18 Программа-функция:	95
3.19 Работа с массивами	96
3.20 Ввод числовых значений в таблицу	99
3.21 Сортировка элементов массива	99
3.22 Решение линейной системы уравнений	103
3.23 Решение обыкновенных дифференциальных уравнений.....	103
3.24 Решение обыкновенного дифференциального уравнения первого порядка методом Рунге-Кутты	104
3.25 Решение системы обыкновенных дифференциальных уравнений первого порядка.....	106
3.26 Работа с файлами данных.....	108
3.27 Работа с комплексными числами.....	109
3.28 Пример расчёта электрической цепи при синусоидальном напряжении.....	112

1 Процесс постановки и решения задач

1.1 Этапы решения задачи

Решение задач средствами программирования на цифровых вычислительных машинах (ЦВМ) является достаточно сложным процессом, состоящим из нескольких этапов:

- Постановка задачи.
- Математическая формулировка задачи.
- Выбор численного метода решения.
- Разработка алгоритма решения задачи.
- Написание программы.
- Ввод исходных данных и программы.
- Отладка программы.
- Решение задачи на ЦВМ.

В зависимости от вида задачи, каждый этап может носить свой собственный характер. Рассмотрим более подробно каждый из этих этапов.

Постановка задачи – определяет цель решения задачи, раскрывает её содержание. Задача формулируется на уровне профессиональных понятий, должна быть корректной и понятной исполнителю (пользователю). Ошибка в постановке задачи, обнаруженная на последующих этапах, приведёт к тому, что работа по подготовке к решению должна начаться с самого начала.

Математическая формулировка задачи осуществляет формализацию задачи путём описания её с помощью формул, определяет перечень исходных данных и получаемых результатов, начальные условия, точность вычисления. По существу, разрабатывается математическая модель решения задачи.

Выбор численного метода решения. В ряде случаев одна и та же задача может быть решена с помощью

различных численных методов. Выбор метода должен определяться многими факторами, основными из которых являются точность результатов решения и время решения на ЦВМ. В каждом конкретном случае в качестве критерия для выбора численного метода принимают какой-либо из указанных критериев или некоторый обобщённый критерий.

Разработка алгоритма решения задачи. На данном этапе устанавливается необходимая логическая последовательность вычислений с учётом выбранного численного метода решения и других действий, с помощью которых будет получен результат.

Написание программы осуществляется по разработанному алгоритму с помощью языка программирования.

Ввод программы и исходных данных выполняется с помощью клавиатуры ЦВМ.

Отладка программ представляет собой процесс обнаружения и устранения синтаксических и логических ошибок.

Решение задачи на ЦВМ обычно проводится в диалоговом режиме. В этом режиме пользователь с помощью клавиатуры ЭВМ может осуществлять ввод программы и её корректировку, трансляцию программы (перевод программы с языка программирования на машинный), исправление синтаксических и логических ошибок при отладке, получение на выходе результатов и вспомогательной информации.

1.2 Понятие об алгоритме решения задачи

Алгоритм – заранее определённая последовательность действий, выполнение которой приводит к однозначному решению задачи.

Запись алгоритма может быть осуществлена многими способами. Самым распространённым является графический метод. Графическая запись алгоритма должна осуществляться в соответствии с ГОСТ 19.701–90.

Схема алгоритма представляет собой последовательность блоков, предписывающих выполнение определённых действий и связи между ними. Наиболее часто используемые в графическом методе символы приведены в таблице 1.

При решении задач могут применяться алгоритмы трёх основных типов: линейный, разветвляющийся и циклический.

Линейный алгоритм. В таком типе алгоритма все действия выполняются в строгой последовательности – один за другим.




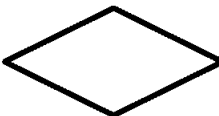

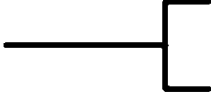
Разветвляющийся алгоритм – алгоритм, в котором те или иные действия осуществляются в зависимости от выполнения или невыполнения некоторого условия.

Циклический алгоритм – алгоритм, в котором присутствуют многократно повторяющиеся действия.

Каждый из этих типов в отдельности применяется при решении простых задач, например, проезд поездом сигнала светофора (разветвлённый алгоритм) или загрузка грузового вагона (циклические действия). Однако, на практике очень трудно встретить задачу, решаемую одним только типом алгоритма, поэтому при решении задач используют комбинацию всех трёх типов. При этом при написании алгоритма решения задачи, он должен обладать следующими свойствами:

– *массовостью*, алгоритм должен позволять решать не одну конкретно поставленную задачу, а целый ряд однотипных задач;

Таблица 1 – Символы, используемые в графическом методе

Обозначение	Действие
	Начало, конец алгоритма
	Ввод данных
	Действие или последовательность вычислений
	Проверка условия, выбор направления выполнения
	Циклические вычисления (параметры цикла: начальное и конечное значения, шаг переменной цикла)
	Предопределённый процесс (уже созданный алгоритм)
	Вывод документа
	Определение разорванных линий связи
	Комментарии

– *детерминированностью*, когда результаты выполнения этого алгоритма разными пользователями при одинаковых исходных данных будут одинаковы;

– *результативностью*, когда после определённого количества шагов *обязательно* будет достигнут результат.

При решении задачи любого уровня сложности можно составить несколько алгоритмов, приводящих к одному результату. Из всего многообразия алгоритмов следует выбирать наилучший, удовлетворяющий какому-либо критерию.

Приведём несколько примеров составления алгоритмов различных типов

Пример. Определить площадь треугольника по формуле

$$s = \sqrt{p(p-a)(p-b)(p-c)},$$

где a , b , c – длины сторон;

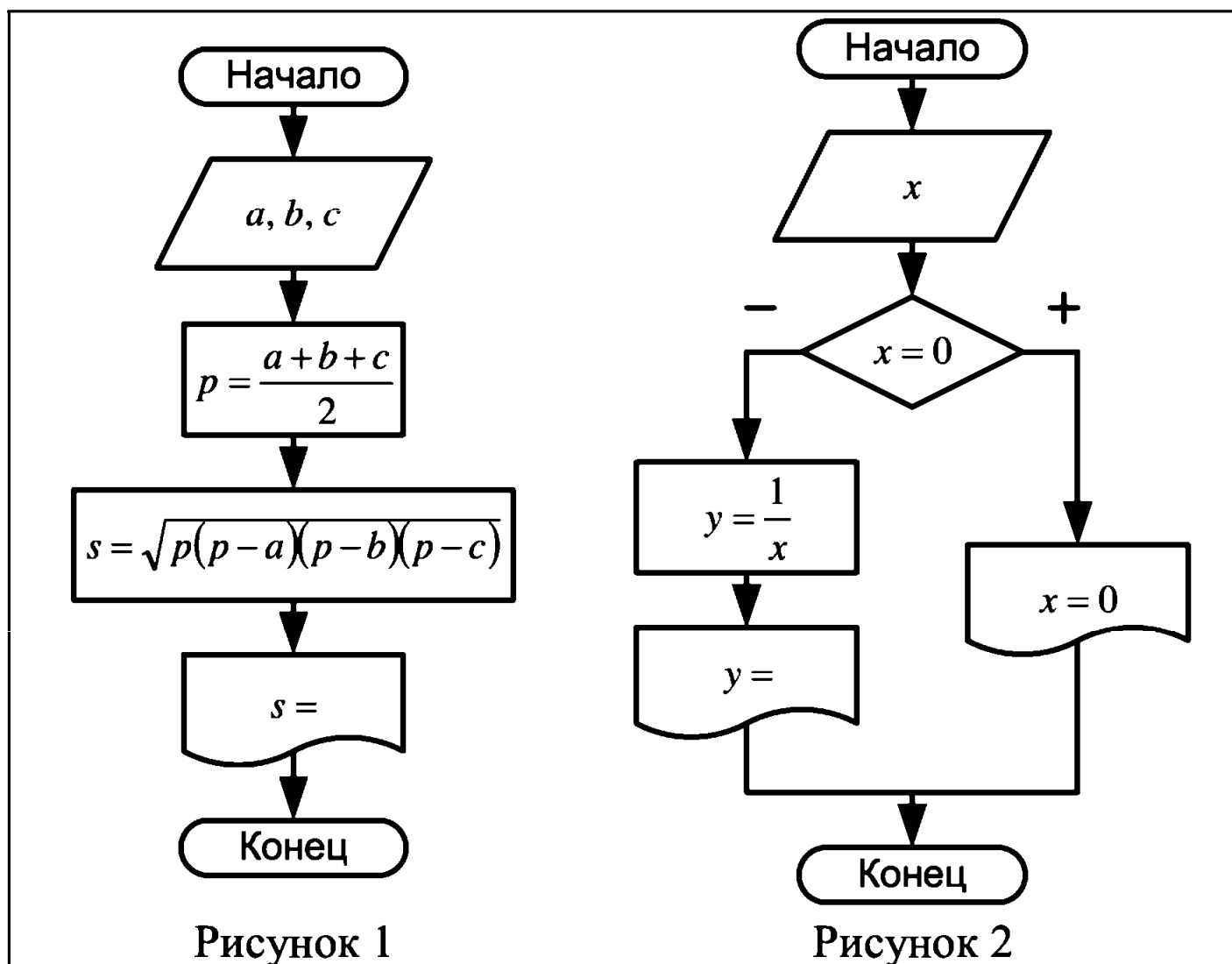
$$p = \frac{a+b+c}{2}.$$

Алгоритм решения данной задачи представляет собой линейный алгоритм, приведённый на рисунке 1.

Пример. Вычислить значение функции

$$y = \frac{1}{x}.$$

Построение алгоритма данной задачи сводится к созданию разветвлённого алгоритма (рисунок 2), т.к. деление на ноль неосуществимо.

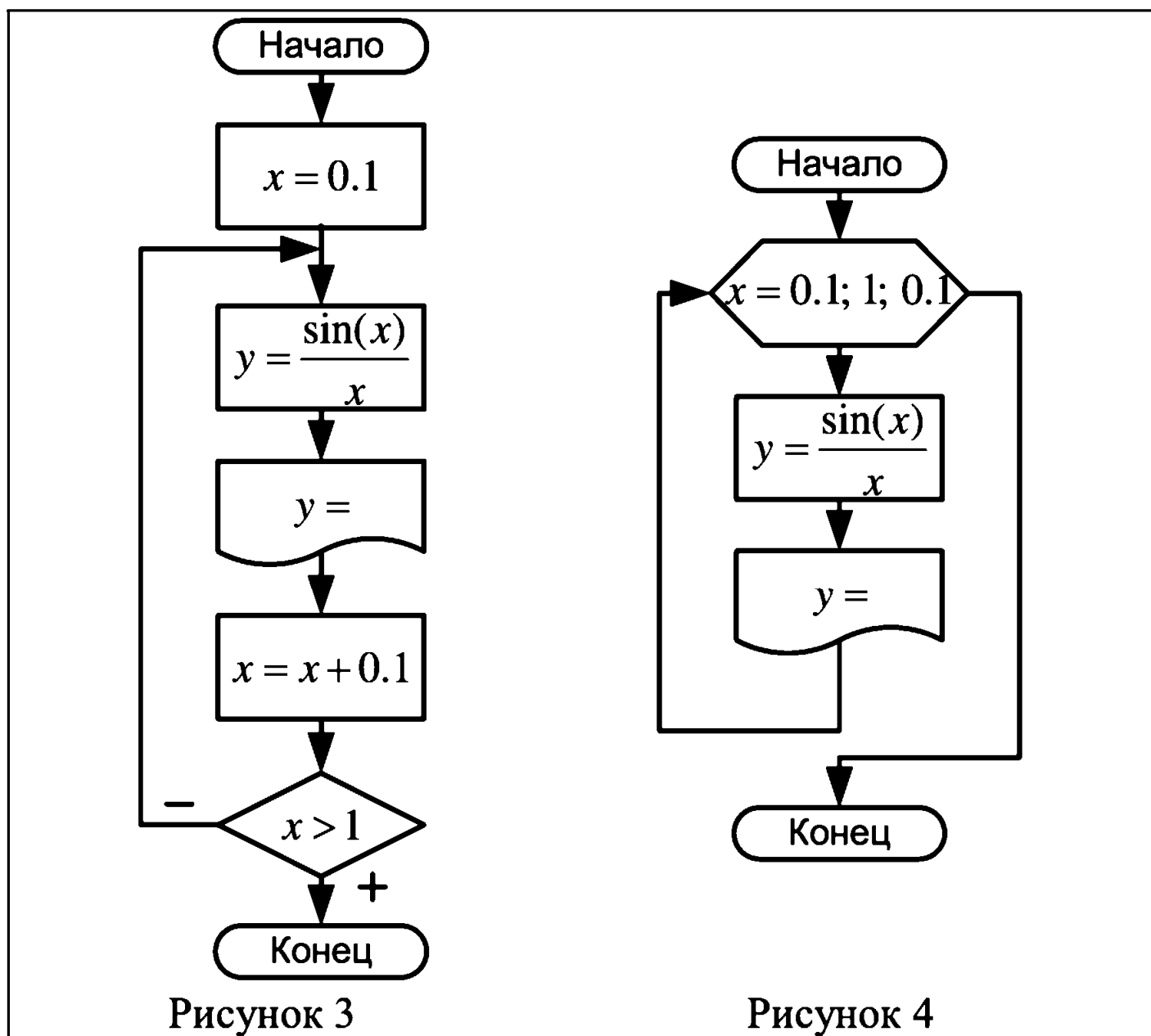


Пример. Вычислить значения функции

$$y = \frac{\sin(x)}{x},$$

при $x = 0,1, 0,2, \dots, 1$, т.е. при x изменяющимся от 0,1 до 1 с шагом 0,1.

Алгоритм решения этой задачи можно реализовать несколькими способами. В первую очередь, это уже рассмотренный разветвлённый алгоритм, с использованием блока проверки условия (рисунок 3). Но при реализации данного алгоритма средствами языка программирования могут возникнуть некоторые трудности, которые будут описаны ниже. Поэтому здесь необходимо использовать алгоритм циклического типа. Формальное представление которого представлено на рисунке 4.



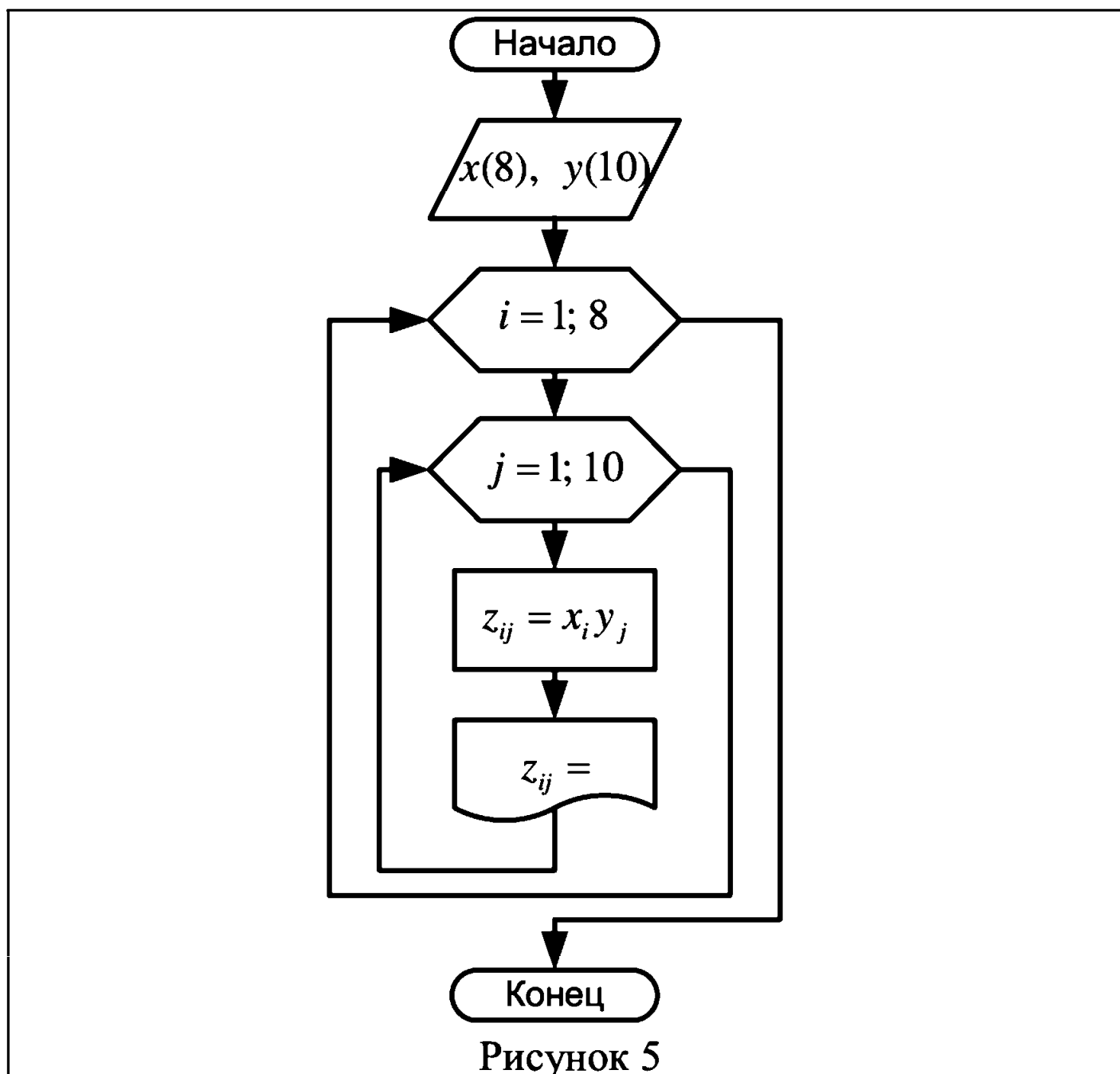
Очень часто встречаются задачи, для которых в алгоритме необходимо использовать вложенные циклические вычисления.

Пример. Вычислить значения матрицы

$$z_{ij} = x_i \cdot y_j,$$

где $x = 1, 2, \dots, 8$; $y = 1, 2, \dots, 10$.

Пример алгоритма решения данной задачи представлен на рисунке 5.



1.3 Реализация алгоритмов с помощью языков программирования

После того как на основании поставленной задачи был сформирован алгоритм решения, необходимо осуществить написание программы с помощью языка программирования.

Программа – последовательность операторов языка программирования, записанных в соответствии со схемой (алгоритмом) программы.

Язык **BASIC** является языком высокого уровня, т.е. он является как бы посредником между человеком и компьютером. Перевод алгоритма с языка программирования на язык машинных кодов осуществляют программы трансляторы.

Программы трансляторы бывают двух типов: интерпретаторы и компиляторы.

Интерпретаторы покомандно переводят алгоритм с языка программирования на язык машинных кодов и тут же исполняют переведённую команду. Интерпретаторы позволяют быстро осуществить отладку программы. К таким программам-интерпретаторам относится **QBasic**.

Компиляторы сначала переводят в машинные коды всю программу, записанную на алгоритмическом языке, и после этого её выполняют. К компиляторам принадлежат **Pascal, C, Turbo BASIC**.

Современные алгоритмические языки имеют интерпретирующий и компилирующий режимы, которые используются соответственно при отладке программы и вычислениях.

2 Алгоритмический язык BASIC

2.1 Общие сведения о среде TurboBASIC

Turbo BASIC (TB) – является оболочкой профессионального алгоритмического языка **BASIC**. Она применяется для упрощения написания и редактирования программ на языке программирования **BASIC**.

Для запуска оболочки необходимо произвести следующие действия:

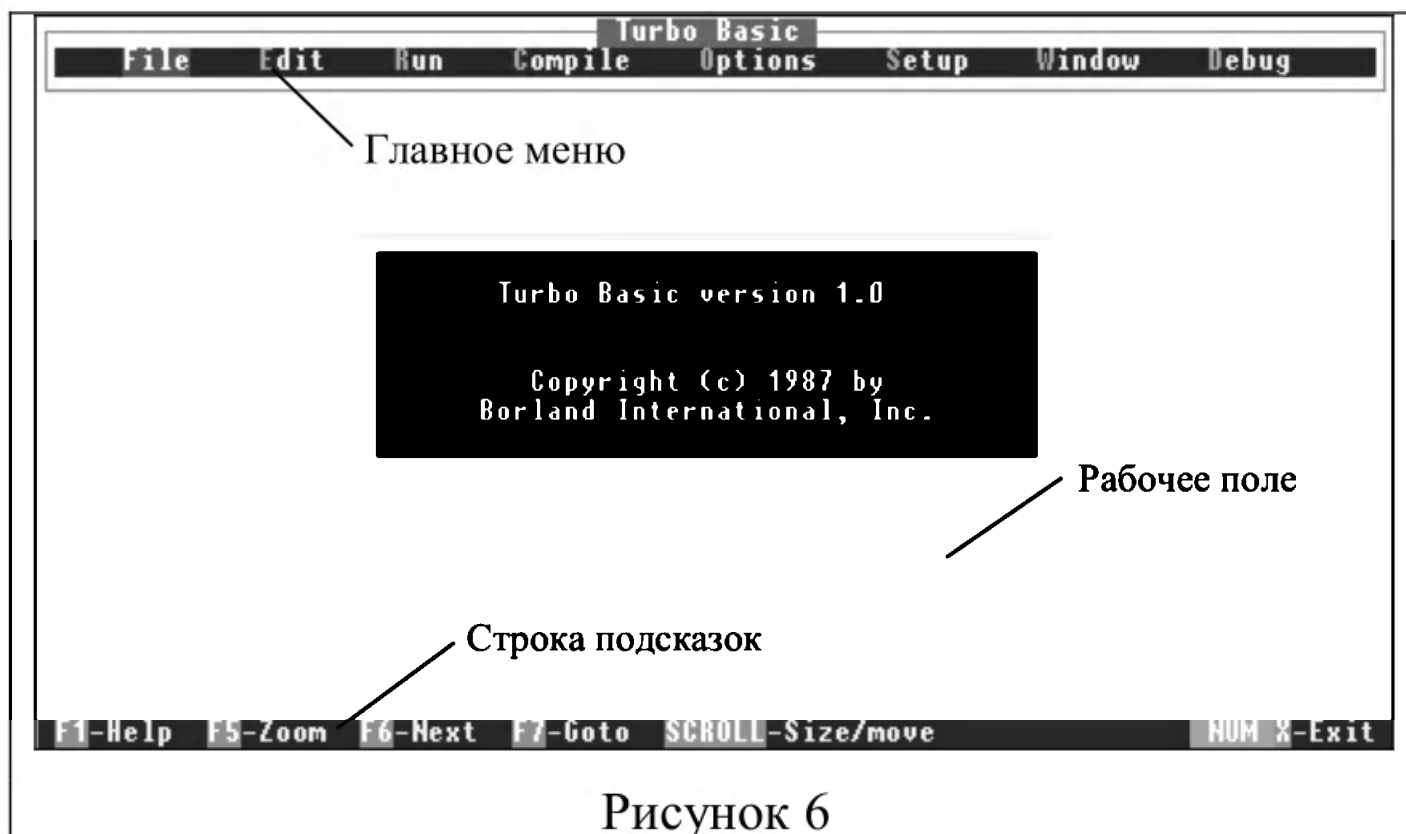
- найти с помощью проводника или папки «Мой компьютер» запускающий файл **TB** – **tb.exe** (обычно файл располагается по адресу **C:/TB/tb.exe**);
- нажать **Enter**.

На экране появится приглашение оболочки **TB** с главным меню (рисунок 6). При нажатии любой клавиши приглашение исчезает и начинается работа в среде **TB**.

2.2 Команды и подкоманды главного меню оболочки Turbo BASIC. Написание программы

Для перехода в главное меню среды **TB** необходимо нажать клавишу **Esc**, после чего, перемещая клавишами вправо–влево, меткой выделяется необходимая команда, выбор команды главного меню подтверждается нажатием клавишей **Enter**. С помощью команд главного меню можно осуществлять следующие действия:

- **File** – набор подкоманд для работы с файлами:
 - **Load** – загрузка файла (на запрос необходимо набрать имя файла с указанием пути расположения файла);
 - **New** – очистка оперативной памяти и экрана для записи новой программы;



- **Save** – сохранение набранной программы (на запрос необходимо набрать имя файла с указанием пути расположения файла);
- **Write to** – сохранение исправленной программы под другим именем (на запрос необходимо набрать имя файла с указанием пути расположения файла);
- **Main file** – указание главного файла при работе со сложной программой;
- **Directory** – указание директории, в которой находятся справочные файлы;
- **Change dir** – Изменение имени директории с файлами справки;
- **OS shell** – временный выход из оболочки ТВ в операционную систему (на экране появляется командная строка операционной системы), для возврата в среду ТВ необходимо набрать командное слово (для ОС DOS это **exit**);
- **Quit** – выход из среды ТВ.
- **Edit** – переход в режим редактирования, вход во встроенный редактор ТВ;

– **Run** – команда запуска написанной программы на исполнение;

– **Compile** – команда компиляции (перевода текста программы, записанного на алгоритмическом языке, в машинный код) написанной программы и создание исполняемого файла. Файл создаётся с расширением **.exe** и может быть запущен в любой операционной системе, например **DOS**, без применения оболочки **TB**.

Остальные команды главного меню предназначены для установок внутри оболочки **TB**.

2.3 Набор текста программы

Набор текста программы осуществляется в окне редактирования. Для этого необходимо переместит курсор в главное меню **TB** на команду **Edit** и нажать клавишу **Enter**.

В оболочке **TB** перемещение по экрану или главному меню осуществляется с помощью клавиш перемещения курсора (стрелки вправо, влево, вверх, вниз).

Написание и редактирование программы в редакторе **TB** осуществляется как в обычном текстовом редакторе.

Применение оболочки **TB** позволяет при написании программы обходиться без индексации строк, что необходимо при использовании простейших версий алгоритмического языка **BASIC**.

2.4 Понятие имени переменной

Во время работы с **TB** в основном осуществляются операции над числовыми значениями, но при решении сложных задач их количество может быть настолько велико, что возникает необходимость эти значения называть собственными именами. Таким образом, вводится понятие переменной и её имени.

Таким образом, в начале программирования необходимо задать имена переменных, которым будут присваиваться числовые значения или которые будут вычисляться в программе. В данной версии **ТВ** именем переменной может быть сочетание латинских буквенных символов и цифр любого количества.

В **ТВ** существует несколько типов переменных. Наибольшее распространение получили два типа переменных строковый и числовой. Тип переменной определяется последним символом в имени переменных:

- **\$** – строковая переменная – группа литер (букв или цифр), заключённая в двойные кавычки, например **x\$="IVAN"**;

- **%** – целая переменная – числовые значения, лежащие в диапазоне от -32768 до 32767 , например **i%=10**;

- **&** – длинная целая переменная – числовые значения, лежащие в диапазоне от -2^{31} до 2^{31} ;

- **!** – вещественная переменная единичной точности – числовые значения, лежащие в диапазоне от -10^{38} до 10^{38} , характеризуется точностью в шесть значащих цифр, например **x=3.14159**;

- **#** – переменная двойной точности, то же что и переменная единичной точности, но значащих цифр – 16.

По «умолчанию», если в имени переменной нет символа, определяющего тип переменной, будет использоваться переменная единичной точности. Тип переменной в **ТВ** помимо явного описания может быть описан неявно с помощью операторов описания:

- **DEFINT** – для целой переменной;
- **DEFSNG** – для вещественной переменной одинарной точности;

- **DEFDBL** – для вещественной переменной двойной точности;

– **DEFSTR** – для строковых переменных.

Пример. **defint a,b,i**

2.5 Команды присваивания

В среде **TB** существует несколько способов присваивания значений числовым или строковым переменным.

1 Непосредственное присваивание: осуществляется с помощью оператора **LET**:

LET /имя переменной/= /присваиваемое значение/

Для среды **TB** указание команды **LET** необязательно и его можно опустить.

Пример.

Команда программы	Результат
a=10	Переменной a присвоено значение 10
tok=0	Переменной tok присвоено значение 0

2 Присваивание значения с помощью клавиатуры: осуществляется оператором **INPUT**, с указанием имени переменной:

INPUT /имя переменной/

При выполнении этого оператора программа приостанавливает свою работу и на экране появиться символ **?**, после чего необходимо задать значение переменной с помощью клавиатуры и нажать **Enter**, выполнение программы будет продолжено. В операторе **INPUT** можно вводить несколько переменных и применять комментарии. Комментарии должны быть заключены в кавычки и при выполнении команды они будут выведены на экран.

Пример.

Команда программы	Результат
input a	?
input a,b,c,	?
input "Введите переменные a,b,c ";a,b,c	Введите переменные a,b,c?

3 Присваивание значения для большого количества переменных осуществляется с помощью пары операторов **READ-DATA**:

READ /имена переменных/

...

DATA /значения переменных/

Количество значений переменных должно соответствовать количеству переменных или быть больше (т.е. на 10 переменных должно быть не менее 10 данных). Операторы **READ** и **DATA** могут быть расположены в разных местах программы. Применения такого вида присваивания целесообразно при количестве переменных более десяти и в том случае, если они не будут изменяться при выполнении вычислений.

Пример.

Команда программы	Результат
read a,b,c	a=1, b=2.5, c=13
...	
data 1,2.5,13	

2.6 Операторы вывода результата

В алгоритмическом языке **BASIC** для вывода полученных результатов применяется оператор **PRINT** в различных формах.

PRINT /имена переменных, выводимых на экран/

В таком виде оператор применяется при выводе числовых данных на экран.

Пример.

Команда программы	Результат
a=2	
b=2.5	
c=13	
print a	2
print a,b,c	2 2.5 13

При выводе большого количества данных, в операторе **PRINT** целесообразно применять комментарии

PRINT *"/комментарии"/, /имена переменных /*

В результате на экран будет выведен комментарий и значения переменных.

Пример.

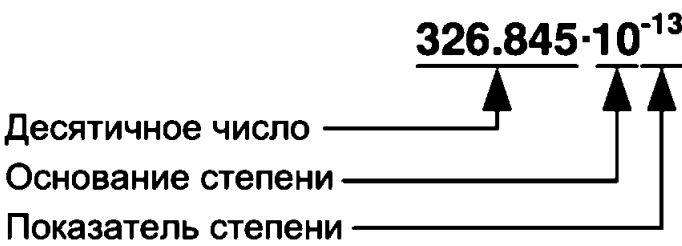
Команда программы	Результат
v=25	
print "Скорость движения";v	Скорость движения 25
a=2	
b=2.5	
c=13	
print "a=";a,"b=";b,"c=";c	a=2 b=2.5 c=13

Если количество значащих цифр в выводимом результате необходимо ограничить, то применяется оператор **PRINT** в следующей форме:

PRINT USING *"/комментарий и формат"/, /список переменных/*

Числовые значения могут выводиться в двух формах: десятичного числа с фиксированной запятой (десятичной

точкой) и экспоненциальной форме. Экспоненциальная форма предполагает представление числа в виде комбинации десятичного числа, основания степени (обычно 10 и обозначается буквой «Е») и показателя степени этого основания.



Для отображения формата числа в операторе используются специальные символы: «-» – для знакоместа знака десятичного числа, «#» – для знакомест десятичного числа, «^» – для знакоместа показателя степени экспоненциального числа. Необходимо отметить, что постановка знака «-» (минус) перед десятичным числом обязательна. Если десятичное число будет положительным, то символ «-» высвечиваться не будет.



Пример.

Команда программы	Результат
v=-12.3625	
print using "Ускорение -###.## км/ч";v	Ускорение -12.36 км/ч
a=0,0000012756	
b=-123456789	
print using "a=-#.###^>^>^> b=-#.###^>^>^>";a,b	a= 1.276E-06 b=-1.235E+08

При ошибке в формате записи оператора **PRINT USING** перед числовым значением с неправильным форматом будет выведен знак процентов: %.

Для вывода результата в файл применяется следующая запись:

PRINT #/номер канала/, /список выводимых переменных/

Более подробно эта форма вывода результатов будет рассмотрена отдельно.

Следует отметить, что в операторе **PRINT** большое значение имеет знак, разделяющий переменные. Для вывода результатов экран монитора условно разделён на 5 зон по 16 символов (всего 80 символов). Если для разделения переменных применена «,» (запятая), то данные будут выводиться в следующей зоне экрана (переход табулятора), если же применена «;» (точка с запятой), то данные будут выводиться непосредственно за предыдущей переменной (подавляется переход в следующую зону экрана).

2.7 Понятие встроенных функций

В алгоритмическом языке **BASIC** для облегчения программирования некоторые сложные функции уже записаны под определёнными именами. Для того чтобы осуществить вычисление с помощью этих функций нужно только ввести имя необходимой функции и указать её аргумент. Основное правило – аргумент функции должен быть заключён в скобки. В таблице 2 приведён список обозначений математических действий и наиболее часто употребляемых встроенных функций.

Также имеются встроенные функции, которые не имеют принятого математического обозначения, например: **INT(X)** – преобразует числовое выражение в целое число с

Таблица 2 – Стандартные функции ТВ

Математическое обозначение	Обозначение в среде ТВ	Математическое обозначение	Обозначение в среде ТВ
+	+	>	>
–	–	<	<
·	*	=	=
÷	/	≤	<=
x	ABS(x)	≥	>=
x^2	x^2	≠	<>
x^n	x^n	\sqrt{x}	SQR(x)
e^x	EXP(x)	$\sqrt[n]{x}$	x^(1/n)
$\sin x$	SIN (x)	$\ln x$	LOG(x)
$\cos x$	COS (x)	$\lg x$	LOG10(x)
$\operatorname{tg} x$	TAN (x)	$\sin^5 x$	SIN(x)^5
$\operatorname{arctg} x$	ATN (x)	целочисленное ÷	\

округлением в низшую сторону (с недостатком); **RND(X)** – выдаёт случайные числа с равновероятным законом распределения в диапазоне от 0 до 1.

Для функций, которых нет в языке **BASIC**, необходимо применить формулы перевода. Наиболее часто при вычислениях встречаются функции **arcsin** и **arccos**:

$$\arcsin(x) = \operatorname{arctg} \frac{x}{\sqrt{1-x^2}}; \quad \arccos(x) = \frac{\pi}{2} - \arcsin(x).$$

2.8 Построение сложных вычислений

При решении задач в ТВ формулы и выражения записываются в строчной форме. Поэтому при записи сложных арифметических выражений необходимо учитывать последовательность (приоритет) действий. В ТВ последовательность действий в сложных вычислениях осуществляется так же, как и в математических вычислениях:

- в первую очередь – выполняются действия, заключённые в скобки;
- во вторую – выполняются действия встроенных функций (**sin**, **cos**, и т.д.);
- в третью – выполняются действия возведения в степень;
- в четвёртую – выполняются действия умножения и деления;
- в пятую – выполняются действия сложения и вычитания.

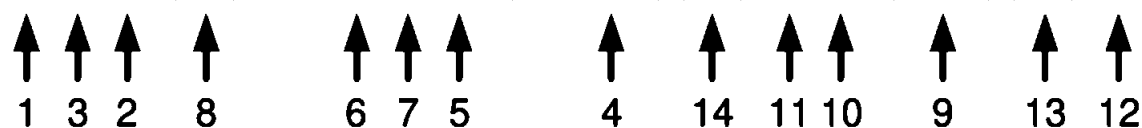
Равнозначные действия выполняются слева направо. Для повышения приоритета действия его необходимо заключить в скобки. Наиболее часто такая ситуация возникает при записи сложных дробных выражений.

Рассмотрим пример записи сложного арифметического выражения (в скобках указана последовательность выполнения действий)

Пример.

$$y = \frac{3x + 5z}{0.25z - \cos 0.5x} + \frac{5 \sin(x + z)}{x + z}$$

$$y = (3 * x + 5 * z) / (0.25 * z - \cos(0.5 * x)) + (5 * \sin(x + z)) / (x + z)$$



2.9 Операторы условного переходов

Часто при выполнении расчётов необходимо выполнять те или иные действия в зависимости от условий, поставленных задач, т.е. использовать разветвлённый тип алгоритма. Для этого в алгоритмическом языке **BASIC** существует специальный оператор – оператор условного перехода **IF**.

Существуют три варианта написания оператора условного перехода:

1 короткий строчный:

IF /условие/ THEN /действие 1/,

где *действие 1* – действие, производимое при выполнении условия;

2 длинный строчный:

IF /условие/ THEN /действие 1/ ELSE /действие 2/,

где *действие 1* – действие, производимое при выполнении условия;

действие 2 – действие, производимое при невыполнении условия;

3 блочный:

IF /условие/ THEN

... /действия, производимые при выполнении условия/

ELSE

... /действия, производимые при невыполнении условия/

END IF

При наличии нескольких взаимоисключающих условий в блочной конструкции оператора **IF** можно

применить оператор **ELSEIF** и добавить блок действий соответствующего условия.

```

IF /условие 1/ THEN
... /действия, производимые при выполнении условия 1/
ELSEIF /условие 2/ THEN
... /действия, производимые при выполнении условия 2/
ELSE
... /действия, производимые при невыполнении условий/
END IF

```

Так же в операторе условного перехода можно использовать проверку по двум условиям для этого проверяемые условия объединяются оператором **AND**, если условия являются взаимоисключающими (проверяется выполнение хотя бы одного условия), то между условиями ставится оператор **OR**.

Приведём примеры программ по определению значения функции y

$$y = \begin{cases} x^2 & \text{при } x < 0 \\ 0 & \text{при } x = 0, \\ \sqrt{x} & \text{при } x > 0 \end{cases}$$

для всех вариантов использования оператора **IF**.

Пример.

Команда программы	Результат
gem Короткая строчная input "Введите x";x if x<0 then y=x^2 if x=0 then y=0 if x>0 then y=sqr(x)	Для всех трёх вариантов Введите x? -2 y= 4

print "y=";y end	или
rem Длинная строчная input "Введите x";x if x<0 then y=x^2 else if x=0 then y=0 else y=sqr(x) print "y=";y end	
rem Блочная input "Введите x";x if x<0 then y=x^2 elseif x=0 then y=0 else y=sqr(x) end if print "y=";y end	
	Введите x? 0 y= 0
	или
	Введите x? 4 y= 2

2.10 Оператор безусловного перехода

Для организации безусловного перехода применяется оператор **GOTO**:

GOTO /метка строки перехода/

Меткой строки может быть целое число в диапазоне от –32768 до 32767, стоящее в начале строки программы.

Рассмотрим пример определения корня квадратного введённого числа.

Пример.

Команда программы	Результат
rem	Введите x? -100

5 input "Введите x";x if x<0 then print "НЕТ КОРНЯ, попробуйте еще раз" goto 5 else y=sqr(x) end if print "y=";y end	НЕТ корня, попробуйте еще раз или Введите x? 100 y= 10
---	--

2.11 Организация циклических вычислений

В алгоритмическом языке **BASIC** имеются средства, позволяющие многократно использовать формулы, математические выражения, относительно независимые части программ (подпрограммы). Это позволяет сокращать длину программ, экономить оперативную память, сокращать время вычисления. Фактически при решении таких задач используется алгоритм циклических вычислений.

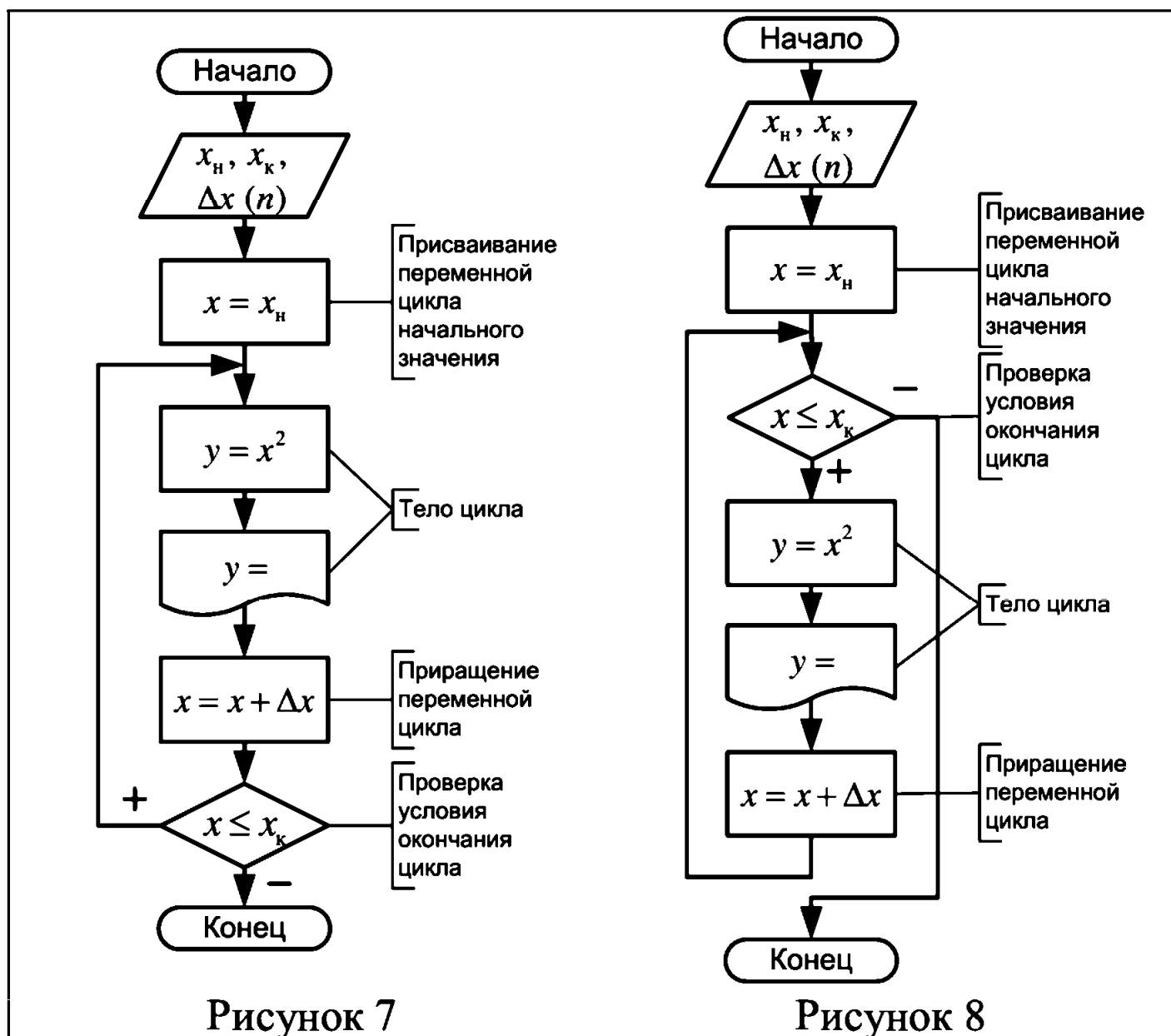
Схема алгоритма циклических вычислений, которая является типовой независимо от того, какими средствами он реализован, представлена на рисунках 3 и 4.

Реализаций алгоритмов циклических вычислений несколько.

Во-первых, такие алгоритмы можно реализовать сочетанием операторов условного и безусловного переходов. Рассмотрим пример такой реализации для нахождения значений функции

$$y = x^2,$$

где $x = 0, 2, \dots, 8$.



Исходя из условий задачи, реализовать циклические вычисления с помощью оператора **IF** можно двумя способами: с проверкой условия цикла в начале циклических вычислений, и с проверкой условия в конце циклических вычислений. Алгоритмы реализованных программ представлены на рисунках 7 и 8.

Пример.

Команда программы	Результат
rem Цикл через IF в конце	x= 0 y= 0
xn=0: xk=8: dx=2	x= 2 y= 4
x=xn	x= 4 y= 16
5 y=x^2	x= 6 y= 36

<pre> print "x=";x,"y=";y x=x+dx if x<=xk then 5 end </pre>	<pre> x= 8 y= 64 </pre>
<pre> rem Цикл через IF в начале xn=0: xk=8: dx=2 x=xn 5 if x<=xk then y=x^2 print "x=";x,"y=";y x=x+dx goto 5 end if end </pre>	

Во-вторых, алгоритмы циклических вычислений с заранее известным числом повторений реализуются с помощью оператора **FOR-NEXT** (цикл «ДО»):

FOR /переменная цикла = начальное значение/ **TO**_
 /конечное значение/ **STEP** /шаг/
 ... /тело цикла/
NEXT /переменная цикла/

Рассмотрим предыдущий пример нахождения функции $y = x^2$ с использованием оператора **FOR-NEXT**.

Пример.

Команда программы	Результат
<pre> rem Цикл через FOR xn=0: xk=8: dx=2 for x=xn to xk step dx y=x^2 print "x=";x,"y=";y next x end </pre>	<pre> x= 0 y= 0 x= 2 y= 4 x= 4 y= 16 x= 6 y= 36 x= 8 y= 64 </pre>

В данном примере, в строке с оператором **NEXT** производятся два действия: изменяется значение переменной цикла и проверяется условие превышения ею конечного значения.

В-третьих, циклический алгоритм с заранее неизвестным количеством повторений реализуется с помощью оператора **WHILE–WEND**:

WHILE /условие, при котором цикл выполняется/
 ... /тело цикла/
WEND

Такой тип циклических вычислений получил название цикл «ПОКА», т.е. циклические вычисления будут осуществляться до тех пор, пока будет соблюдаться условие циклических вычислений.

Пример нахождения функции $y = x^2$ с использованием оператора **WHILE–WEND** представлен ниже.

Пример.

Команда программы	Результат
rem Цикл через WHILE	x= 0 y= 0
xn=0: xk=8: dx=2	x= 2 y= 4
x=xn	x= 4 y= 16
while x<=xk	x= 6 y= 36
y=x^2	x= 8 y= 64
print "x=";x,"y=";y	
x=x+dx	
wend	
end	

Следует отметить, что при использовании цикла «ПОКА» в теле цикла обязательно должна присутствовать строка с приращением переменной цикла (в примере это

строка $x=x+dx$), в случае её отсутствия цикл становится «бесконечным» и его завершение невозможно.

Как уже говорилось выше, очень часто существует необходимость применения вложенных циклов (рисунок 5). При этом в качестве переменных цикла используются разные переменные. При окончании циклических вычислений первым необходимо закрыть внутренний цикл, а затем внешний. Количество применяемых вложенных циклов практически не ограничено. Наиболее часто такие алгоритмы используются при работе с матрицами или массивами. Рассмотрим в качестве примера заполнение двумерного массива $A(5,5)$ исходными данными (фрагмент программы)

Пример.

Команда программы	Результат
rem	1 2 3 4 5
dim A(5,5)	6 7 8 9 10
for i=1 to 5	11 12 13 14 15
for j=1 to 5	16 17 18 19 20
read A(i,j)	21 22 23 24 25
print A(i,j);	
next j	
print	
next i	
data 1,2,3,4,5	
data 6,7,8,9,10	
data 11,12,13,14,15	
data 16,17,18,19,20	
data 21,22,23,24,25	
end	

2.12 Применение массивов

В **ТВ**, как и в любом другом алгоритмическом языке, имеются средства, позволяющие хранить в оперативной памяти числовые значения в виде совокупности пронумерованных по порядку чисел – ***массивов чисел***.

Массивы принято делить на одномерные и многомерные, в частности двумерные или матрицы. В общем случае **ТВ** позволяет объявлять 8-ми мерные массивы.

Для возможности применения массивов чисел в программе их необходимо сначала объявить, т.е. указать их имена, размерность и объём, т.е. количество элементов (чисел), составляющих массив. Для объявления массива используется оператор **DIM** (от английского **dimension** – размер):

DIM /список имён массивов с указанием их объёма/

В скобках указывается объём массивов, т.е. сколько элементов будет содержать массив, начиная отсчёт с нулевого (n-го) элемента.

Пример.

Команда программы	Результат
dim A(100)	В оперативной памяти зарезервировано место для одномерного массива A , состоящего из 100 чисел
dim B(10,10), C(1001)	В оперативной памяти зарезервировано место для двумерного массива B размером 10×10 (100 чисел) и одномерного массива C , состоящего из 1001 числа

Двумерные массивы (матрицы) задаются количеством строк и количеством столбцов. Для записи элементов массивов можно применить операторы присваивания **READ-DATA** или **INPUT**. При занесении значений элементов одномерного массива оператор присваивания ставится внутри операторов цикла (в тело цикла) с количеством шагов равных количеству элементов массива. При занесении значений элементов двумерного массива оператор присваивания ставится внутри двойного (вложенного) цикла и общее количество шагов циклов должно также равняться числу элементов массива.

Рассмотрим несколько примеров организации массивов.

Пример. Заполнить одномерный массив **A(10)** произвольными числами

Команда программы	Результат
rem Заполнение массива dim A(12) for i= 1 to 12 input A(i) next i end	В оперативной памяти содержится 12 введённых пользователем чисел, сформированных в массив A

Пример. Заполнить массив **A(5×5)** произвольными числами

Команда программы	Результат
rem Заполнение массива dim B(5,5) for i= 1 to 5 for j=1 to 5 read B(i,j)	В оперативной памяти содержится двумерный массив B, состоящий из 25 введённых пользователем чисел

next j next i data 1,2,3,4,5 data 6,7,8,9,10 data 11,12,13,14,15 data 16,17,18,19,20 data 21,22,23,24,25 end	
---	--

Достаточно часто при решении задач с массивами чисел встречается необходимость определить максимальный и минимальный элементы массива. Для этого используется циклический алгоритм решения (рисунок 4 или 5). Начальным условием нахождения максимума или минимума в массиве является присваивание этим переменным значения одного из элементов массива, обычно это первый элемент: $A(1)$ или $A(1,1)$. Это обусловлено тем, что если оставить значения переменных, определяющих максимум или минимум без присваивания, они будут численно равны 0, что может привести к некорректному решению задачи.

Следует заметить, что при нахождении максимума (минимума) среди числовых значений, не сформированных в массив, в качестве начальных значений должны приниматься следующие числа: для максимума – бесконечно большое отрицательное число ($\max = -10^{38}$), для минимума – бесконечно большое положительное число ($\min = 10^{38}$).

Рассмотрим пример нахождения максимального и минимального элементов в одномерной матрице $A(10)$, с произвольными числами.

Пример.

Команда программы	Результат
rem	? 10
dim A(10)	? 9
for i= 1 to 10	? 8
input A(i)	? 7
next i	? 6
max=A(1):min=A(1)	? 5
for i=1 to 10	? 4
if A(i)>max then max=A(i)	? 3
if A(i)<min then min=A(i)	? 2
next i	? 1
print "Min=";min	Min= 1
print "Max=";max	Max= 10
end	

2.13 Организация файлов последовательного и прямого доступа

Файлы – это последовательности записей, объединённых каким-либо общим назначением и хранящиеся на внешнем накопителе (накопителе, винчестере и т.д.). Файлы различают по способу записи и считывания данных из них.

В файле в отличие от массивов можно хранить значительно больше данных и не занимать при этом оперативную память. Но из-за обращения к устройствам с низким быстродействием (таким как дисководы гибких магнитных дисков), время доступа увеличивается, что соответственно увеличивает время выполнения программ.

Файлы, используемые **ТВ**, делят на файлы последовательного и прямого доступов.

а) Файлы последовательного доступа

Для организации файла с последовательным доступом его необходимо «открыть». При этом в оперативной памяти создаётся временная буферная зона, используемая для хранения и пересылки данных на внешний носитель информации или считывания с внешнего носителя в оперативную память. Для открытия файла используется оператор: **OPEN**, с указанием номера канала перекачки данных и имени файла с данными.

Далее применяются операторы записи (**PRINT**) или считывания (**INPUT**) данных с указанием номера канала и имени переменной (или нескольких переменных).

Для окончания работы с файлом данных необходимо осуществить «закрытие» файла с помощью оператора **CLOSE**.

Примеры фрагментов программ:

– при записи в файл

```
OPEN "o",#1,"имя файла.dat"  
PRINT #1,x  
CLOSE #1
```

– при считывании из файла

```
OPEN "i",#1,"имя файла.dat"  
INPUT #1,x  
CLOSE #1
```

Отличительной особенностью файлов последовательного доступа является то, что при считывании данных из файла все значения считываются последовательно и для того чтобы найти, например, пятую запись необходимо считать четыре предыдущих.

Рассмотрим пример использования файла последовательного доступа для записи значений функции

$$y = 20 \sin x ,$$

где $x = 0 \dots 2\pi$,

$$\Delta x = \frac{2\pi}{n};$$

n – количество записываемых ординат.

Пример.

Команда программы	Результат
rem Файл последовательного доступа n=200 x=0 xk=2*3.1415 dx=xk/n open "o",#1,"a:sin.dat" for i=1 to n+1 y=20*sin(x) x=x+dx print #1,x,y next i close #1 end	На носителе (a:) будет создан файл sin.dat, в котором будут содержаться значения аргумента x и соответствующие им значения функции y. (Файл можно просмотреть с помощью программы «Блокнот» операционной системы Windows)

б) Файлы прямого доступа

В файлах прямого доступа обращение к данным и работа с результатами упрощена. Это достигнуто за счёт применения разбивки буферной зоны по переменным и номерам записи.

Для создания файла прямого доступа необходимо:

1 Открыть файл в режиме прямого доступа

OPEN "r", #1, "имя файла", 32

Символ **r** указывает на режим использования файла – прямой или произвольный (**random**) доступ, цифра **32**

указывает размер записи в байтах/ По умолчанию этот размер равен 128 байтам, **#1** – указание номера канала передачи данных.

2 Для распределения количества ячеек памяти в буфере для размещения переменных, которые будут записаны в файл, используется оператор **FIELD** (внутри оператора используется командное слово **as** – как).

FIELD #1, 20 as a\$, 2 as b\$, 10 as c\$

Сумма размеров полей размещения переменных должна быть равна количеству байтов, указанных в операторе **OPEN**, в нашем случае – **20+2+10=32**.

3 Для записи данных в буфер используется оператор **LSET** или **RSET** (для выравнивания записи по левому или правому краю отведённого поля соответственно).

Перед помещением в буфер цифровых данных их рекомендуется преобразовывать в строковые для компактности записей в файле. Для этого используются функции **MKI\$** – для целых чисел, **MKS\$** – для чисел с одинарной точностью, **MKD\$** – для чисел с двойной точностью.

4 Для переноса данных из буфера на диск или носитель информации используется оператор **PUT**.

PUT #1, C%,

где **#1** – номер канала,

C% – номер записи, обязательно выраженный в целых числах.

Для обращения к файлу прямого доступа требуется следующее:

1 Открыть файл в режиме прямого доступа – **г**.

2 Распределить буфер для считывания переменных – это две строки в программе, идентичные строкам программы при записи данных в файл.

3 Для переноса данных из файла в буфер используется оператор **GET**

GET #1, C%.

После этого данные в буфере доступны для использования их в программе.

4 Для обратного перевода цифрового содержимого файла прямого доступа в числовой формат используются функции: **CVI** – для целых чисел; **CVS** – для чисел с одинарной точностью, **CVD** – для чисел с двойной точностью.

Работа программ записи и считывания данных из файла прямого доступа происходит по схеме, изображённой на рисунке 9. Фрагменты программ, описывающих работу операторов, приведены ниже:

– для оператора **PUT**

...

OPEN "r",#1,"a:spisok.dat"

FIELD #1,5 as a\$,5 as b\$,25 as c\$, 12 as d\$

LSET a\$,b\$,c\$,d\$

PUT #1,1

close #1

...

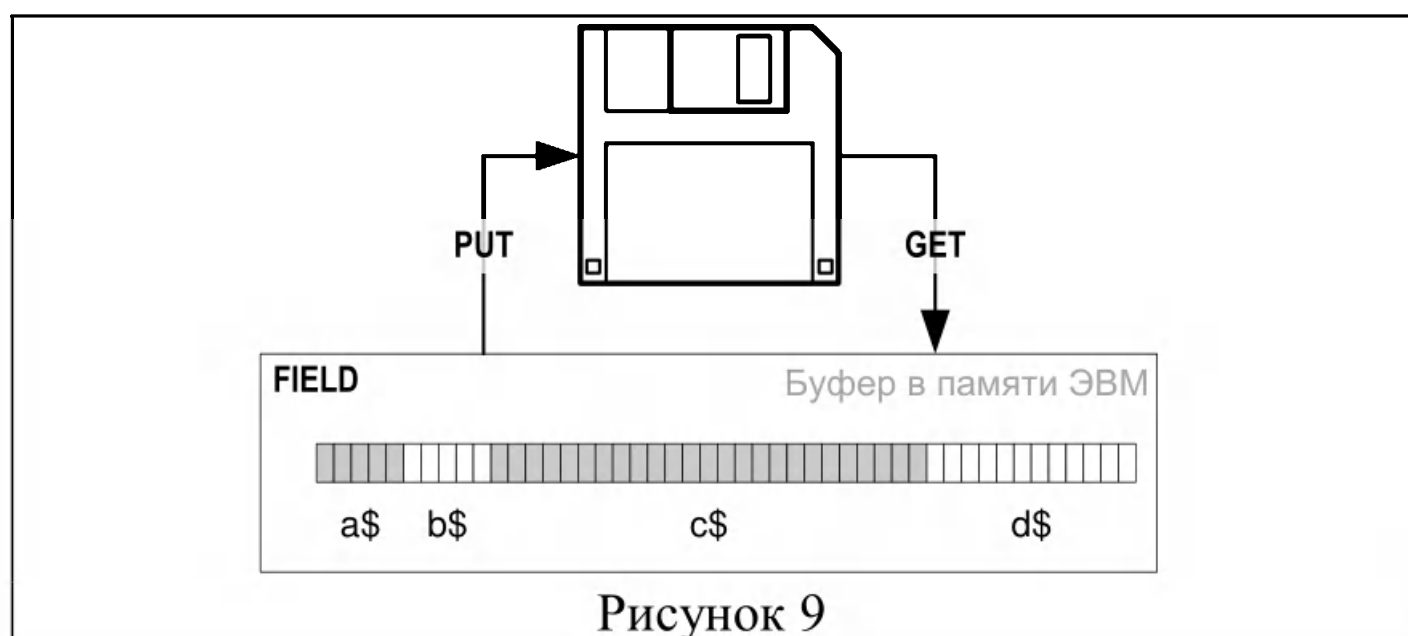


Рисунок 9

– для оператора **GET**

```
...  
OPEN "r",#1,"a:spisok.dat"  
FIELD #1,5 as a$,5 as b$,25 as c$, 12 as d$  
GET #1,1  
close #1  
...
```

Рассмотрим пример использования файла прямого доступа для записи значений функции $y = 20\sin x$, где $x = 0 \dots 2\pi$, $\Delta x = \frac{2\pi}{n}$; n – количество записываемых ординат.

Пример.

Команда программы	Результат
rem Файл прямого доступа - запись n=10 x=0 xk=2*3.1415 dx=xk/n open "r",#1,"a:sin.dat",20 field #1,10 as x\$,10 as y\$ for i%=1 to n+1 lset x\$=mks\$(x) y=20*sin(x) lset y\$=mks\$(y) x=x+dx put #1,i% next i% close #1 end	На носителе информации (a:) будет создан файл sin.dat, в котором будут содержаться значения аргумента x и соответствующие им значения функции y.

rem Файл прямого доступа - чтение n=10 x=0 xk=2*3.1415 dx=xk/n open "r",#1,"a:sin.dat",20 field #1,10 as x\$,10 as y\$ for i%=1 to n+1 get #1,i% x=cvs(x\$) y=cvs(y\$) print using "x=-#.## y=-##.##";x,y next i% close #1 end	x= 0.00 y= 0.00 x= 0.63 y= 11.76 x= 1.26 y= 19.02 x= 1.88 y= 19.02 x= 2.51 y= 11.76 x= 3.14 y= 0.00 x= 3.77 y=-11.75 x= 4.40 y=-19.02 x= 5.03 y=-19.02 x= 5.65 y=-11.76 x= 6.28 y= -0.00
--	---

2.14 Работа со строковыми переменными

Строковые переменные находят широкое применение при выполнении поставленных задач. Характерным обозначением строковых переменных является знак **\$** в имени переменной (**x\$, name\$**). Значением переменной является не числовые величины, а строка символов, например:

```
x$="hello"  
name$="ivanov"
```

Так, например, строковая переменная часто используется для задания любого имени файла при написании программ для работы с файлами, например (фрагмент программы):

```

...
rem задание имени файла
input "Задайте имя файла (8 латинских символов)";f$
fn$="a:"+f$+".dat"
open "o",#1,fn$
...

```

В данном примере строка **fn\$="a:"+f\$+".dat"** позволяет сформировать строковую переменную, содержащую полный адрес создаваемого файла с указанием его местоположения. Результатом выполнения данного фрагмента программы будет запрос на имя создаваемого файла и получение файла на носителе **a:**.

В процессе работы с символьными переменными возникает необходимость в их обработке. Обработка символьных последовательностей сводится к следующему:

- нахождению в заданной символьной последовательности необходимой строки символов или последовательности символов;
- сравнение предложенных последовательностей с заданной последовательностью символов с целью выделения из предложенных последовательностей, удовлетворяющих заданному условию;
- вычисление длины (количества символов) заданной последовательности;
- нахождение в ряде заданных последовательностей или символьных переменных совпадающих символов и выборка их по этому критерию.

Все названные операции широко используются при обработке баз данных (совокупности символьных переменных, отражающих сведения об объектах).

Наиболее распространённые функции, обрабатывающие символьные последовательности следующие:

- MID\$(a,n,[p])** – выделение подстроки символов
 где **a** – строка символов, символьное выражение или символьные переменные;
n – номер позиции в строке символов, с которой начинается анализ, **n<255**;
[p] – длина выделяемой подстроки символов; может быть не указана, тогда выдаются все символы от **n** до конца строки;
- LEFT\$(a,n)** – выделение левых символов строки **a**
 где **n** – длина строки формируемых символов;
- RIGHT\$(a,n)** – выделение правых символов строки **a**;
- INSTR([n],a,s)** – поиск символьной подстроки **s** в заданной строке символов **a**, начиная с **n**-го символа.
 Если длина исходной строки равна 0 или подстрока не найдена в исходной строке, то результат выполнения функции **INSTR** – нуль. Если выражение **n** не задано, то предполагается, что оно равно 1.
- LEN(a)** – определение длины строки символов (результат: константа целого типа).

Рассмотрим примеры использования указанных выше функций.

Пример.

Команда программы	Результат
st\$="функция mid\$" nm\$=mid\$(st\$,2,6) print nm\$	ункци
st\$="время не ждёт" rs\$=right\$(st\$,7) print rs\$	не ждёт
a\$="программа на Бэйсике" b\$=left\$(a\$,9) print b\$	программа
a\$="dbcaebc" b\$="bc" print instr(a\$,b\$);instr (3,a\$,b\$)	2 6
x\$="переменная тип" print len(x\$)	14

Примером использования символьных переменных может служить программа создания базы данных.

Пример. Создать в файле прямого доступа базу данных, состоящую из фамилий, имён, отчеств, адресов и телефонов жильцов.

Команда программы	Результат
rem Файл прямого доступа (база) open "r",#2,"a:spisok.dat",90 field #2,10 as m1\$,10 as m2\$,10 as m3\$,45 as _ adr\$,10 as tel\$ for c%=1 to 5 read n1\$:read n2\$:read n3\$:read ad\$: read tt\$ lset m1\$=n1\$ lset m2\$=n2\$ lset m3\$=n3\$	На носителе (a:) будет создан файл spisok.dat, в котором будут содержаться сведения о жильцах.

<pre> lset adr\$=ad\$ lset tel\$=tt\$ put #2,c% next c% close #2 data "Ivanov","Ivan","Ivanovich","1 avenue,15","12-34" data "Petrov","Petr","Petrovich","2 avenue,1","42-14" data "Sidorov","Sidor","Sidorovich","3 avenue,24","65-59" data "Vasilev","Vasilii","Vasilich","4 avenue,5","13-45" data "Victorov","Victor","Victorovich","5 avenue,16","77-77" end </pre>	
--	--

2.15 Применение графических средств

Очень часто при решении задач невозможно оценить результаты из-за большого количества численных значений, поэтому для оценки таких результатов необходимо использовать графические средства **ТВ**. Графический редактор **ТВ** позволяет строить двумерные графики по числовым значениям аргумента и функции.

Для построения графиков математических зависимостей используется группа операторов, которые соответствуют действиям человека при ручном построении графиков математических функций, а именно: определение формата бумаги для размещения графиков, разметка осей для функции и независимой переменной с учётом масштабов различных физических величин, расстановка точек на поле графика, оформление осей графика.

При написании программы для построения графиков функций используются следующие операторы:

- для перевода экрана монитора в графический режим с соответствующим разрешением (в дальнейшем будем использовать режимы: 2 – 640×200 точек, или 12 – 640×480 точек):

SCREEN 2
(SCREEN 12)

- для определения поля расположения графика в виде прямоугольника с координатами его диагональных углов:

VIEW (x1,y1)–(x2,y2), ,1

где **x1,y1** и **x2,y2** – координаты соответственно верхнего левого и правого нижнего углов;

- 1** – указывает на то, что при выполнении оператора на экране будет нарисован прямоугольник с выделенным полем.

- для наложения на выделяемое поле экрана прямоугольного поля, размеры которого выражены в физических единицах функции (по вертикали) и аргумента (по горизонтали):

WINDOW (x1,y1)–(x2,y2)

где **x1, y1** – значения минимумов соответственно аргумента и функции;

x2, y2 – координаты максимумов соответственно аргумента и функции;

- для расстановки на поле экрана точек графика с координатами x , y , выраженных в физических единицах:

PSET (x,y)

- для построения прямой линии (координатной оси), соединяющей две точки с координатами x_1 , y_1 и x_2 , y_2 .

LINE (x1,y1)–(x2,y2)

где x_1, y_1 – координаты начала линии в физических величинах;
 x_2, y_2 – координаты конца линии в физических величинах;

- для обозначения координатной оси в заданном месте экрана

LOCATE y,x: PRINT “/комментарий/”

где **LOCATE** – команда переноса курсора;
 y, x – координаты начала обозначения, выраженные в знакахместах экрана.

Если максимум и минимум функции неизвестны, то рассчитанные значения функции должны быть сохранены в массиве или файле. После чего необходимо определить максимальное и минимальное значения функции, которые затем подставляются в оператор **WINDOW**. Таким образом, график функции в любом случае будет виден в поле экрана. В случае, когда график функции совпадает с границей рабочего окна, необходимо в команде **WINDOW** добавить к значениям максимума и минимума функции небольшое произвольное значение.

Рассмотрим пример построения графика функции

$$y = 20\sin x ,$$

где $x = 0 \dots 2\pi$,

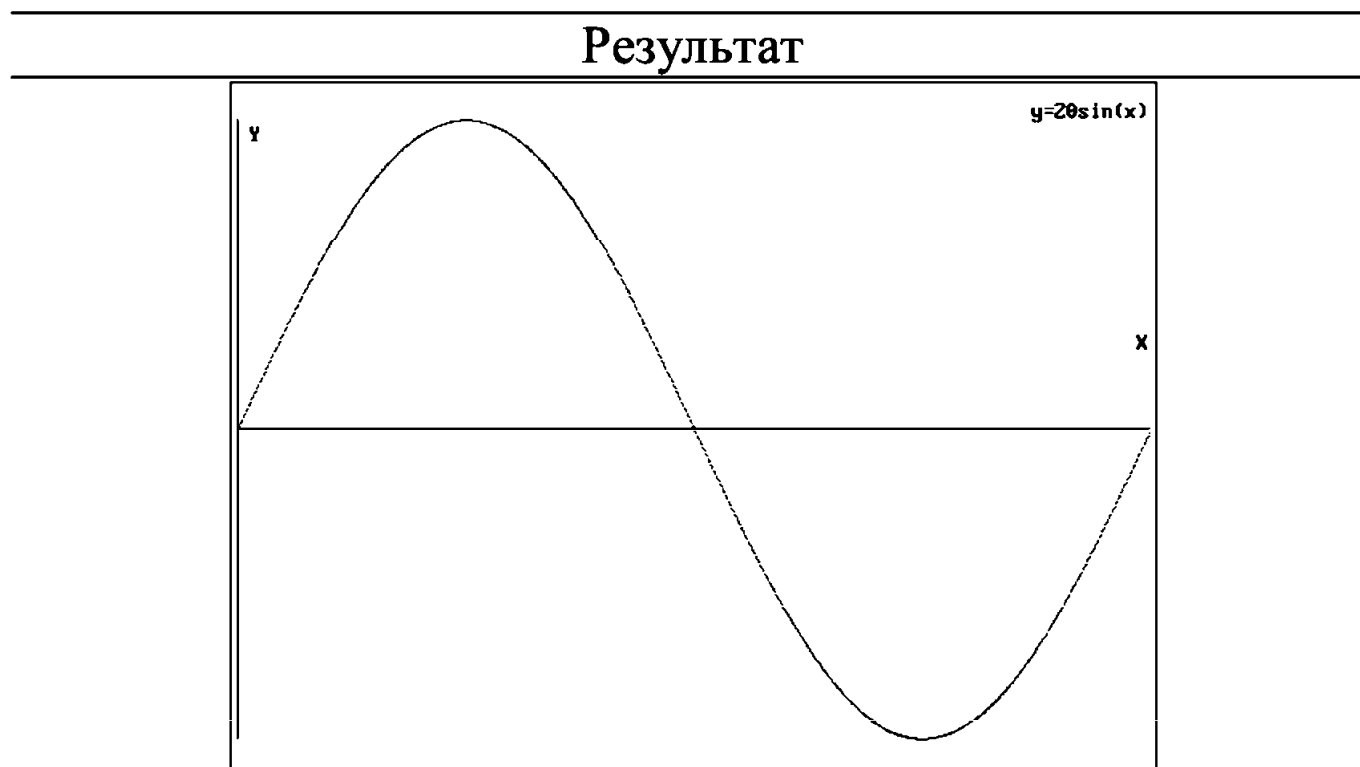
$$\Delta x = \frac{2\pi}{n} ;$$

n – количество точек.

Пример.

Команда программы

```
rem Построение графика
n=1000
x=0
xk=2*3.1415
dx=xk/n
max=20: min=-20
screen 12
view (4,4)-(636,478),,15
view (8,8)-(632,476)
window (1,max+max*.1)-(n,min+min*.1)
for i=1 to n
  y=20*sin(x)
  x=x+dx
  pset (i,y)
next i
line (1,min)-(1,max)
locate 3,3: print "Y"
line (1,0)-(n,0)
locate 12,79: print "X"
locate 2,70
print "y=20sin(x)"
end
```



Для построения на одном экране нескольких окон с графиками математических функций необходимо повторить все шаги для каждого окна, при этом следуют учесть, что окна будут располагаться на экране в зависимости от того, как они заданы операторами **VIEW**.

Рассмотрим пример построения на экране двух графиков: функции

$$u(t) = 220 \sin \omega t$$

и изменения тока в цепи при отключении дополнительного сопротивления в момент времени $T/3$ (фрагмент программы).

Пример.

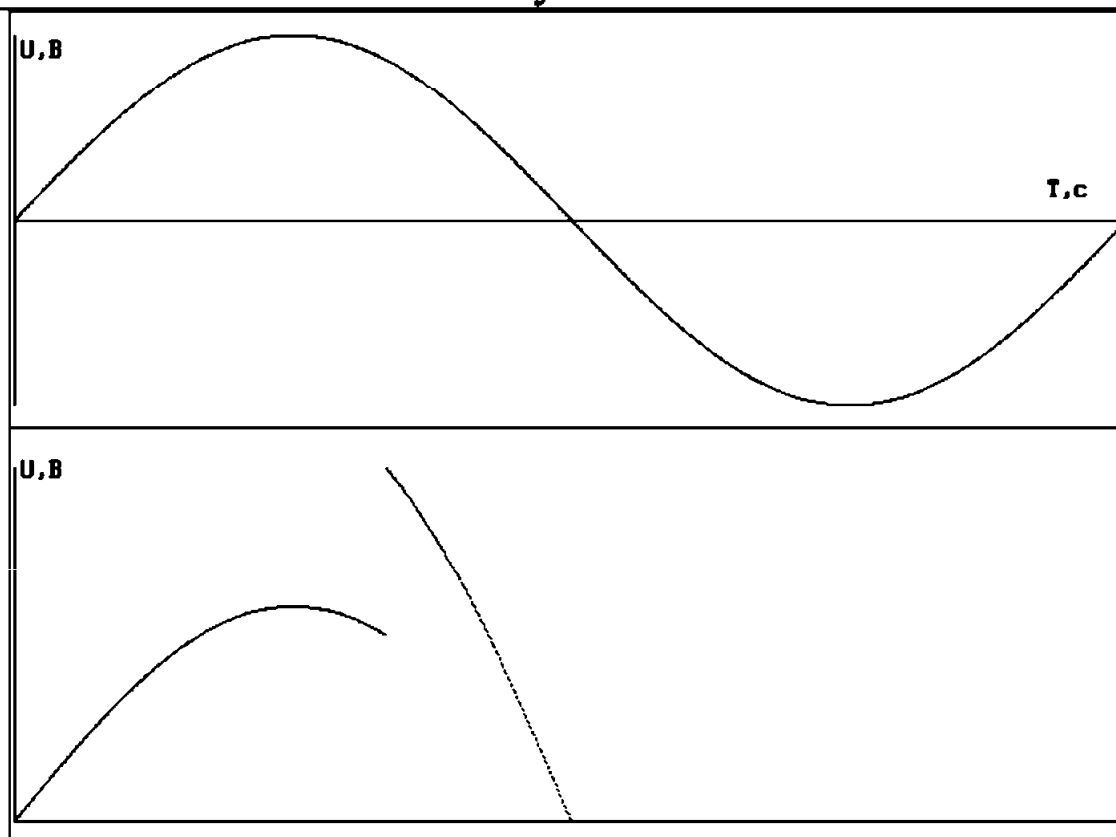
Команда программы
<p>...</p> <p>gem построение двух графиков</p> <p>screen 12</p> <p>view (2,2)-(638,238),,1</p> <p>view (4,4)-(636,234)</p> <p>window (1,amin*1.1)-(m,amax*1.1)</p>

```

for i=1 to m
  pset (i,a(i))
next i
line (1,amin)-(1,amax)
line (1,0)-(m,0)
locate 2,2:print "U,B"
locate 7,75:print "T,c"
view (2,240)-(638,478),,1
view (4,242)-(636,476)
window (1,bmin-0.25)-(m,bmax*1.1)
for i=1 to m
  pset (i,b(i))
next i
line (1,bmin)-(1,bmax)
line (1,0)-(m,0)
locate 17,2:print "U,B"
...

```

Результат



2.16 Организация подпрограмм и подпрограмм-процедур

а) Подпрограммы

Подпрограмма – это выполняющая конкретную функцию группа операторов, к которой можно обратиться из любого места основной программы.

Подпрограмма обязательно должна начинаться со строки, имеющей метку, а заканчиваться оператором **RETURN**. Располагается подпрограмма за оператором **END**, т.е. после завершения основной программы. К подпрограмме нельзя обращаться с помощью оператора **GOTO**.

Достаточно часто подпрограммы используются для написания сложной программы, состоящей из отдельных блоков. Для удобства разработки таких программ, алгоритм программы делится на несколько относительно независимых по смыслу частей. Каждая из этих частей программируется отдельно, а потом эти части соединяются в основной программе (рисунок 10).

Обращение к подпрограмме производится оператором **GOSUB**, а возврат из неё только оператором **RETURN**:

GOSUB /номер строки начала подпрограммы/

Рассмотрим пример использования подпрограмм в задаче, построенной по схеме, изображённой на рисунке 10

Недостатки использования подпрограмм:

- подпрограммы не могут использоваться в пределах основной программы;
- для использования написанных подпрограмм в других программах, необходимо согласовывать обозначения общих переменных, т.е. изменять их имена по всей основной программе или в подпрограмме.

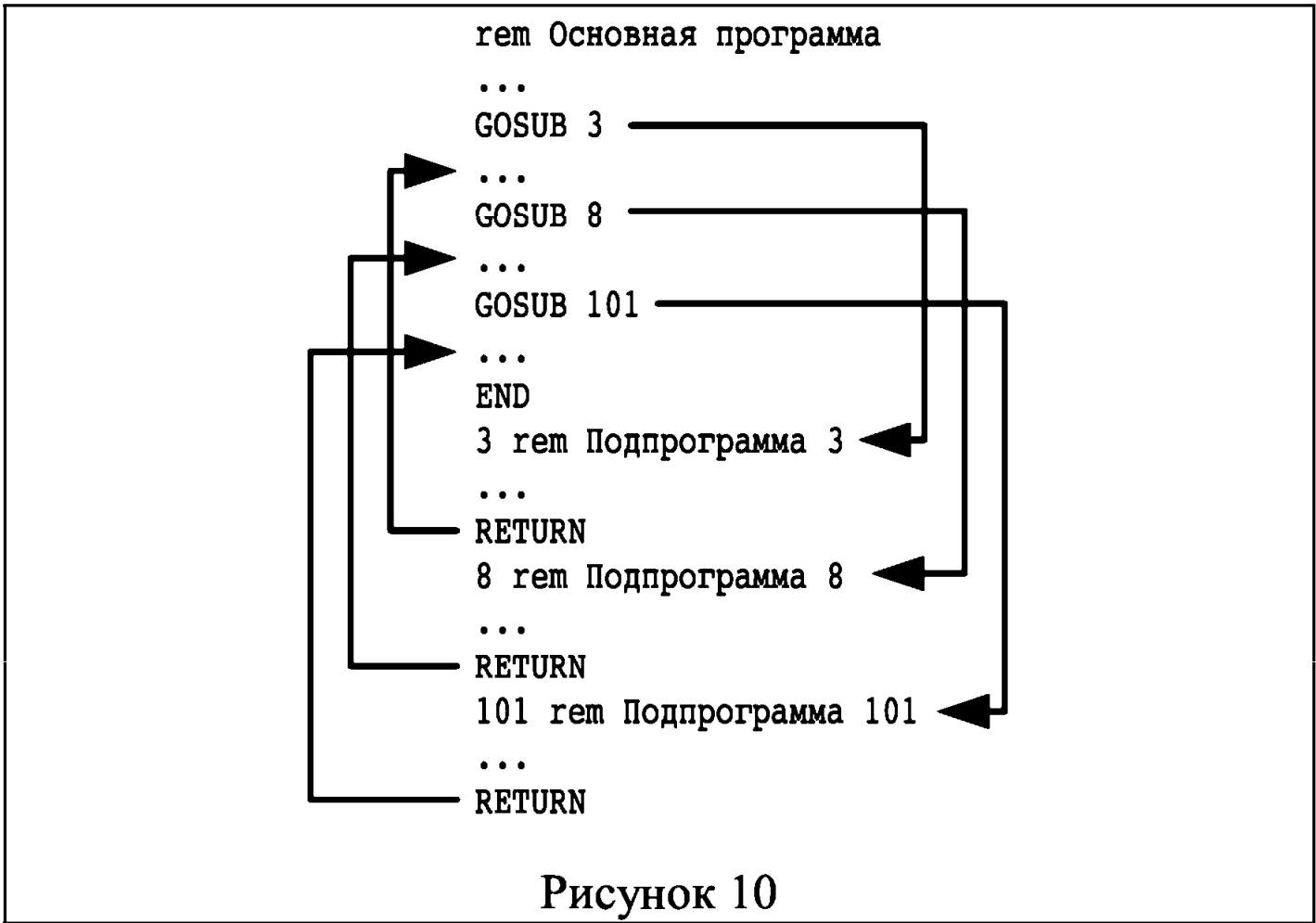


Рисунок 10

Пример.

Команда программы	Результат
rem Подпрограммы x=2 gosub 3 print "y=";y gosub 8 print "y=";y gosub 101 print "y=";y end 3 y=x^2 return 8 y=x^3 return 101 y=sqr(x) return	y= 4 y= 8 y= 1.414213538169861

Второй недостаток можно устранить, если использовать подпрограммы-процедуры.

б)Подпрограммы-процедуры

Применение процедур позволяет снять все ограничения на применение подпрограмм в других программах. Для этого вводятся понятия фактические и формальные переменные.

Фактические переменные – это переменные, которые используются в основной (главной) программе.

Формальные переменные – это переменные, которые используются при написании подпрограмм-процедур.

Вызов процедуры из основной программы производится с помощью команды:

CALL */имя процедуры/(/список фактических переменных/)*

Подпрограмма-процедура записывается после основной программы (после **END**). Последовательность используемых операторов:

SUB */имя процедуры/(/список формальных параметров /*
.../программа-процедура/
END SUB

Списки фактических и формальных переменных, используемых при работе с подпрограммами-процедурами должны полностью совпадать, т.е. по количеству передаваемых переменных и их типу. Имена фактических и формальных переменных могут быть произвольными, т.е. они могут как совпадать, так и быть различными.

При передаче в процедуру массивов, в команде **CALL** указывается только имя массива, а в команде **SUB** – объявляется имя массива и в скобках указывается размерность массива (а не количество элементов массива!). Так при передача двумерного массива **A(5,5)** в процедуру в

операторе **CALL** необходимо указать **A()**, а в операторе **SUB** – **B(2)**.

Как уже говорилось выше блоки подпрограмм-процедур можно легко передавать между программами, согласовывая только количество, расположение и тип передаваемых фактических и формальных переменных. Рассмотрим пример использования подпрограммы-процедуры для нахождения максимального значения массива.

Пример. Дан массив **A(n)**, распечатать из этого массива только те элементы, которые соответствуют условию

$$\frac{A(i)}{A_{\max}} < 0.35,$$

где A_{\max} – максимальное значение элемента в массиве **A(n)**.

Команда программы	Результат
rem Программа с применением процедуры MAX	max= 386
n=12	A(1)= 3
dim A(n)	A(2)= 18
for i=1 to n	A(3)= 16
read A(i)	A(4)= 39
next i	A(5)= 40
data 3,18,16,39,40,45	A(6)= 45
data 101,18.5,386,10,11,38	A(7)= 101
call max(A(),max,n)	A(8)= 18.5
print "max=";max	A(9)= 386
for i=1 to n	A(10)= 10
if A(i)<=A(i)/max<0.35 then print "A(";i;")=";A(i)	A(11)= 11
next i	A(12)= 38
end	
sub max(B(1),max,k)	

max=B(1) for i=1 to k if B(i)>=max then max=B(i) next i end sub	
---	--

2.17 Нестандартные математические функции, вводимые пользователем

В процессе решения многих задач часто встречаются математические выражения с повторяющимися действиями или функциями, которых нет в **ТВ**. Поэтому в **ТВ** реализована возможность создания нестандартных функций – функций пользователя.

Все нестандартные функции, используемые в программе, должны быть объявлены или описаны с помощью команды:

DEF FN */имя функции/(/формальные переменные/)=_*
/нестандартная функция/

Нестандартная функция всегда описывается через формальные переменные.

Рассмотрим пример применения нестандартной функции пользователя при вычислении значения функций

$$y = \frac{\boxed{x^3 + 2x^2} + 8 + \cos(x)}{\sin(\boxed{x^3 + 2x^2} + 5)};$$

$$z = \frac{\boxed{t^3 + 2t^2} + 5 + 3\sin(t)}{\boxed{t^3 + 2t^2} + 18},$$

при $x=5, t=8.5$.

В уравнениях пунктиром выделены одинаковые действия с разными переменными, к которым будет применена функция пользователя.

Пример.

Команда программы	
rem Применение нестандартной функции	
def fn f1(m)=m^3+2*m^2	
x=5	
t=8.5	
y=(fn f1(x)+8+cos(x))/sin(fn f1(x)+5)	
z=(fn f1(t)+5+3*sin(t))/(fn f1(t)+18)	
print "x=";x,"y=";y	
print "t=";t,"z=";z	
end	
Результат	
x= 5	y=-228.7749633789062
t= 8.5	z= .9863453507423401

2.18 Типовые задачи

Задача №1

Написать программы для вычисления

$$a = \frac{\sqrt{|x-1|} - \sqrt[3]{|y|}}{2}, \quad b = x(\operatorname{tg} z + e^{-(x+3)}),$$

$$1 + \frac{x}{2} + \frac{y}{4}$$

где $x=-0.622$, $y=3.325$, $z=5.541$

$$f = \begin{cases} \frac{ia^2 + (-1)^{i+1} i^2 (b-a)}{\sqrt{5a^2 + ai + b^2 i^2}} & \text{при } a > 0 \\ \frac{ia^2 + i^2 (b-a)}{\sqrt{5a^2 + ai + b^2 i^2}} & \text{при } a \leq 0 \end{cases}$$

где $i=1, 2, 3, \dots, n$

Команда программы

```
x=-0.622:y=3.325:z=5.541
a=(sqr(abs(x-1))-abs(y)^(1/3))/(1+x^2/2+y^2/4)
b=x*(atn(z)+exp(-(x+3)))
print "a=";a,"b=";b
input "Ddtlbnt n";n
for i=1 to n
  if a>0 then
    f=(i*a^2+(-1)^(i+1)*i^2*(b-a))/sqr(5*a^2+a*i+b^2*i^2)
  else
    f=(i*a^2+i^2*(b-a))/sqr(5*a^2-a*i+b^2*i^2)
  end if
  print "f=";f
next i
end
```

Результат

```
a=-5.533454939723015E-002  b=-.9236583113670349
Введите n? 10
f=-.9002460241317749
f=-1.843156337738037
f=-2.784279108047485
f=-3.724903345108032
f=-4.665319919586182
f=-5.605630397796631
f=-6.54587984085083
f=-7.486090183258057
```

f=-8.426275253295898
f=-9.366440773010254

Задача №2

Занести в оперативную память по условию квадратную матрицу $A(5,5)$:

$$A := \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Команда программы	Результат
dim a(5,5)	0 0 0 1 0
for i=1 to 5	0 0 1 0 0
for j=1 to 5	0 1 0 0 0
if i+j=5 then a(i,j)=1 else a(i,j)=0	1 0 0 0 0
next j	0 0 0 0 0
next i	
for i=1 to 5	
for j=1 to 5	
print a(i,j);	
next j	
print	
next i	
end	

Задача №3

Дан двумерный массив $A(5,5)$. Найти сумму нечётных положительных элементов массива и выдать её на печать, указать номера строк и столбцов этих элементов

Команда программы

```
dim a(5,5)
for i=1 to 5
  for j=1 to 5
    read a(i,j)
  next j
next i
data 18,28,66,32,84
data 1.3,6,-7,3,101
data 2.6,5,1,0,-26
data 4.8,-42,41,-36,22
data 12,0.7,73,52,3.2
for i=1 to 5
  for j=1 to 5
    print using " -###.#";a(i,j);
  next j
  print
next i
print "Ytxtnyst gjkj;bntkmyst 'ktvtyns vfccbdf:"
for i=1 to 5
  for j=1 to 5
    if int(a(i,j)/2)<>a(i,j)/2 and a(i,j)>0 then
      print "a(";i;",";j;")=";a(i,j)
      s=s+a(i,j)
    end if
  next j
next i
print
print using " Сумма нечетных элементов ###.#";s
end
```

Результат				
18.0	28.0	66.0	32.0	84.0
1.3	6.0	-7.0	3.0	101.0
2.6	5.0	1.0	0.0	-26.0

4.8 -42.0 41.0 -36.0 22.0
12.0 0.7 73.0 52.0 3.2

Нечетные положительные элементы массива:

a(2 , 1)= 1.299999952316284

a(2 , 4)= 3

a(2 , 5)= 101

a(3 , 1)= 2.599999904632568

a(3 , 2)= 5

a(3 , 3)= 1

a(4 , 1)= 4.800000190734863

a(4 , 3)= 41

a(5 , 2)= .699999988079071

a(5 , 3)= 73

a(5 , 5)= 3.200000047683716

Сумма нечетных элементов 236.6

Задача №4

Построить в разных окнах три графика функций:

$$y_1 = \sin \omega t ,$$

$$y_2 = \sin 3\omega t + \sin 5\omega t + \sin 7\omega t ,$$

$$y_3 = \sin 2\omega t + \sin 4\omega t + \sin 6\omega t ,$$

где $0 \leq t \leq t_p$, $f = 50$, $t_p = \frac{1}{f}$, $dt = \frac{t_p}{1000}$

Команда программы

n=1000:f=50:t=0:tk=1/f:dt=tk/n:w=2*3.14*f

dim y1(n),y2(n),y3(n)

for i=1 to n

 y1(i)=sin(w*t)

 y2(i)=sin(3*w*t)+sin(5*w*t)+sin(7*w*t)

 y3(i)=sin(2*w*t)+sin(4*w*t)+sin(6*w*t)

```

    print y1(i),y2(i),y3(i)
    t=t+dt
next i
max1=y1(1):min1=y1(1)
max2=y1(1):min2=y1(1)
max3=y1(1):min3=y1(1)
for i=1 to n
    if y1(i)<min1 then min1=y1(i)
    if y2(i)<min2 then min2=y2(i)
    if y3(i)<min3 then min3=y3(i)
    if y1(i)>max1 then max1=y1(i)
    if y2(i)>max2 then max2=y2(i)
    if y3(i)>max3 then max3=y3(i)
next i
print max1,max2,max3
print min1,min2,min3
screen 12
view (2,2)-(319,239),,1
view (4,4)-(317,237)
window (1,min1)-(n,max1)
for i=1 to n
    pset (i,y1(i))
next i
line (1,0)-(n,0)
line (1,min1)-(1,max1)
view (321,2)-(635,239),,1
view (323,4)-(633,237)
window (1,min2)-(n,max2)
for i=1 to n
    pset (i,y2(i))
next i
line (1,0)-(n,0)
line (1,min2)-(1,max2)
view (2,241)-(319,475),,1

```

```

view (4,243)-(317,473)
window (1,min3)-(n,max3)
for i=1 to n
  pset (i,y3(i))
next i
line (1,0)-(n,0)
line (1,min3)-(1,max3)
end

```

Результатом выполнения программы будут три графика функций, расположенные в разных окнах.

Задача №5

Записать в файл последовательного доступа фамилии студентов с указанием года рождения. Вывести на экран список студентов 1979 года рождения.

Команда программы

```

cls
open "o",#1,"a:f10.dat"
open "o",#2,"a:f11.dat"
print "Введите фамилию и год рождения"
for i=1 to 5
  input "введите данные";nam$,year
  print #1,nam$
  print #2,year
next i
close #1
close #2
print
print "Получены данные:"
open "i",#1,"a:f10.dat"
open "i",#2,"a:f11.dat"
for i=1 to 5

```

```

    input #1,nam$
    input #2,year
    print nam$,year
next i
close #1
close #2
print
print "В 1979 году родились:"
open "i",#1,"a:f10.dat"
open "i",#2,"a:f11.dat"
for i=1 to 5
    input #1,nam$
    input #2,year
    if year=1979 then print nam$
next i
close #1
close #2
end

```

Результат

Введите фамилию и год рождения
 введите данные? Иванов,1978
 введите данные? Петров,1979
 введите данные? Сидоров,1980
 введите данные? Васечкин,1979
 введите данные? Орлов,1989

Получены данные:

Иванов	1978
Петров	1979
Сидоров	1980
Васечкин	1979
Орлов	1989

В 1979 году родились:

Петров
Васечкин

Задача №6

Составить список студентов с полученными оценками.
Рассчитать средний бал студента и вывести его на экран.
Указать неуспевающих студентов.

Команда программы

```
dim nam$(5),srbal(5)
print "          Средний бал"
for i%=1 to 5
    sum%=0
    read nam$(i%)
    for j%=1 to 5
        read ocen%
        sum%=sum%+ocen%
    next j%
    srbal(i%)=sum%/5
    print nam$(i%),srbal(i%)
next i%
print
print "Неуспевающие ученики:"
for i%=1 to 5
    if srbal(i%)<3 then print nam$(i%)
next i%
data "Иванов",2,3,5,4,4
data "Петров",5,5,5,5,5
data "Сидоров",4,4,4,4,4
data "Васечкин",2,2,3,5,2
data "Орлов",1,2,5,1,1
end
```

Результат

Средний бал	
Иванов	3.599999904632568

Петров	5
Сидоров	4
Васечкин	2.799999952316284
Орлов	2

Неуспевающие ученики:
Васечкин
Орлов

Задача №7

Написать программу обработки базы данных, в которой предусмотрены: поиск записи по номеру в списке, поиск по фамилии, запись в базу данных по указанному номеру записи.

Команда программы

```
open "r",#2,"a:spisok",90
field #2,10 as m1$,10 as m2$,10 as m3$,45 as adr$,10 as tel$
1 print "Поиск по: 1 - номеру в списке"
  print "      2 - фамилии клиента"
  print
  print "      0 - запись нового клиента"
input num
if num=1 then input "Номер? ";c%:gosub 123
if num=2 then input "Фамилия? ";fam$:gosub 234
if num=0 then gosub 345
input "Продолжить? у-да";y$
if y$="y" or y$="Y" then goto 1
close #2
end
123 get #2,c%
  print m1$;m2$;m3$
  print adr$;tel$
  return
```

```
234 for c%=1 to 10
    get #2,c%
    if instr(m1$,fam$)<>0 then gosub 123
next C%
return
345 input "Номер записи? ";c%
    input "Фамилия? ";n1$
    input "Имя? ";n2$
    input "Отчество? ";n3$
    input "Адрес? ";ad$
    input "Телефон? ";t$
    lset m1$=n1$
    lset m2$=n2$
    lset m3$=n3$
    lset adr$=ad$
    lset tel$=t$
    put #2,c%
return
```

3 Программный пакет MathCad

3.1 Основные понятия

Наряду с языками программирования для решения различных задач широко используются различные программные пакеты. В программных пакетах имеется большое количество уже запрограммированных алгоритмов, к которым при решении задачи пользователь может обращаться. К таким пакетам относится **MathCad**.

Математический пакет программ **MathCad** – это математический редактор, позволяющий осуществлять математические расчёты различной степени сложности: от простых арифметических вычислений, до сложных инженерных расчётов. Простота использования **MathCad** заключается в использовании метода реального представления математического документа, т.е. то, что видно на экране будет использовано для расчётов и получено в окончательном документе при печати его на принтере.

В **MathCad** математические выражения представляют собой обычные формулы и уравнения. Для их вычисления не надо писать программы, как, например, в **Turbo BASIC**, что значительно облегчает работу с программой. Помимо проведения расчётов, **MathCad** может использоваться для создания пояснительных записок, научных и технических документов. Любой документ обычно содержит в себе формулы, результаты расчётов, таблицы, графики, текстовые комментарии, иллюстрации. В **MathCad** всё это заменяется текстовыми и расчётными блоками, создавая структуру законченного документа. При этом таблицы и графики в **MathCad** являются блоками формул, а текстовые блоки при проведении расчётов игнорируются.

Работа в программе аналогична работе в формульном редакторе программы **Word**.

3.2 Начало работы с MathCad

Для запуска **MathCad** необходимо найти ярлычок, изображённый на рисунке 11. Обычное расположение этого ярлычка находится по пути **Пуск→Все Программы→Mathcad→MathCad14→MathCad14**. Данный ярлычок может быть выведен на рабочий стол.

После запуска программы на экране монитора появляется рабочий экран **MathCad** (рисунок 12), который



Рисунок 11

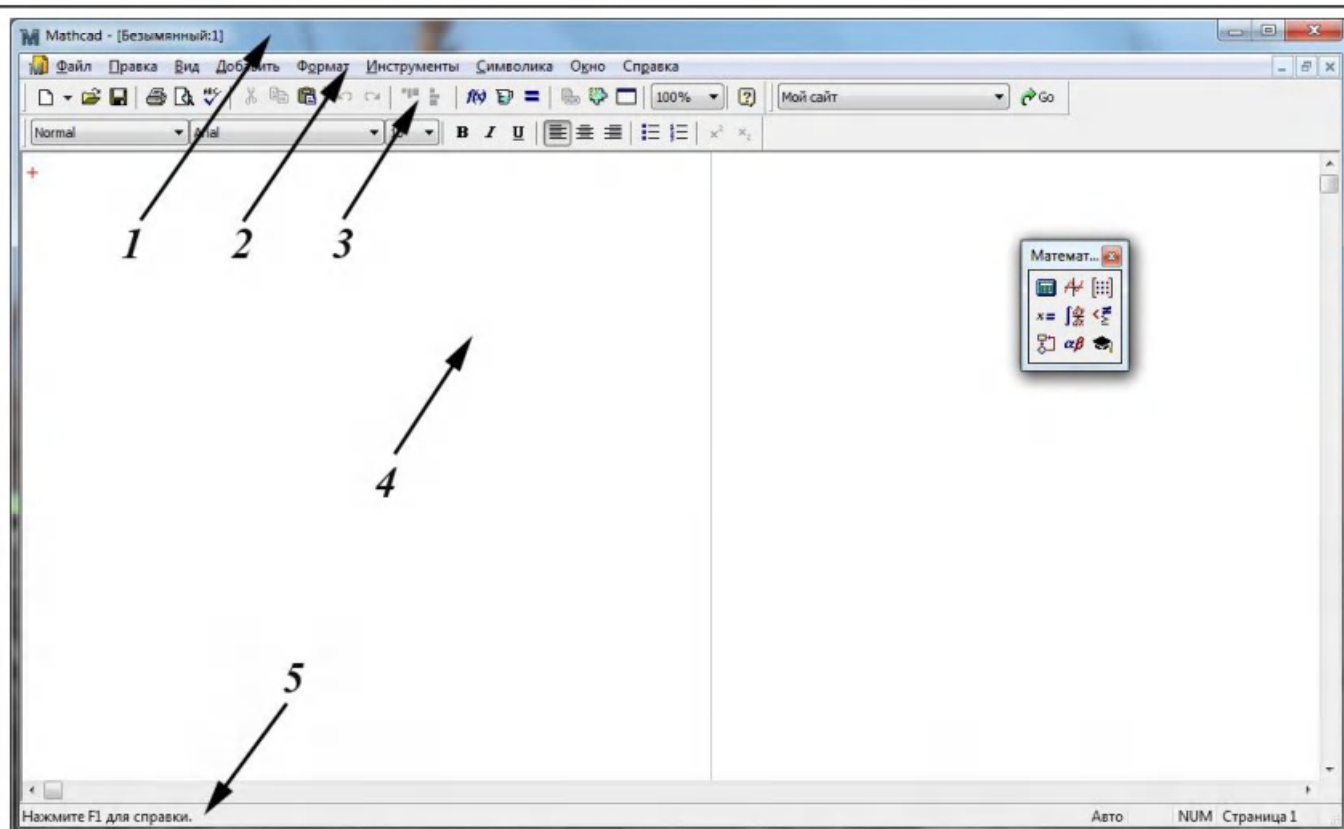






Рисунок 12

представляет собой стандартный рабочий экран программ-приложений операционной системы **Windows**. Основными элементами экрана программы являются:

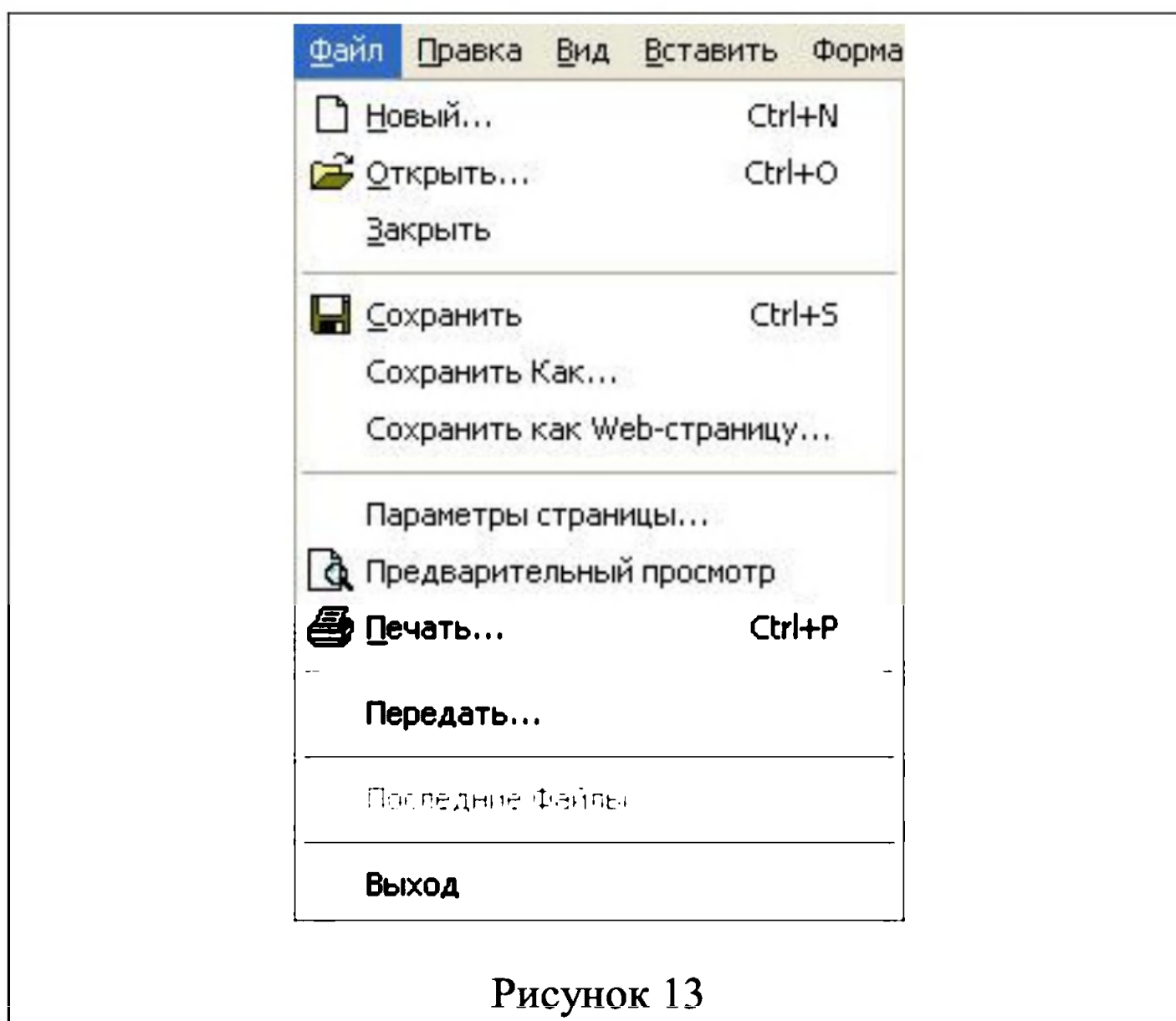
- заголовок окна приложения (1), содержащий управляющие кнопки приложения;
- главное меню программы-приложения (2), содержащее все доступные в приложении команды;
- панели инструментов программы-приложения (3), кнопки быстрого доступа до наиболее часто используемых команд;
- рабочая область (4), в которой непосредственно создаётся документ;
- строка состояния (5), которая выводит режим работы программы-приложения.

При работе с программой используются несколько видов курсоров мыши, каждый из них позволяет осуществить то или иное действие в разных областях рабочего окна:

-  – указатель – стандартный вид курсора;
-  – курсор ввода – определяет на рабочем листе верхний левый угол вводимого блока;
-  – курсор редактирования формулы – определяет направление и зависимость элементов формулы;
-  – знакоместо – элемент выражения, требующий заполнения.

3.3 Работа с документами

При работе с **MathCad** используются те же принципы, что и при работе со всеми остальными программами-приложениями в операционной системе **Windows**. Так при работе с документом пользуются командами главного меню (рисунок 13):



Файл → Новый – для создания нового документа **MathCad**;

Файл → Открыть – для открытия уже созданного документа;

Файл → Заккрыть – для закрытия текущего документа;

Файл → Сохранить – для сохранения текущего документа;

Файл → Сохранить как – для создания копии текущего документа.

Следует обратить внимание на то, что все перечисленные действия осуществляются только с тем документом, который в данный момент используется программой, т.е. является рабочим. Поскольку **MathCad**, как и любая другая программа **Windows**, является

многозадачной, следует быть внимательным при одновременной работе с большим количеством документов.

3.4 Запись математических выражений

Ввод выражения в **MathCad** достаточно прост. Для этого необходимо выполнить следующие действия:

- выбрать место в документе, где будет записано выражение;
- ввести левую часть выражения;
- поставить знак равенства.

Последовательность, приведённая выше, описывает так называемый режим калькулятора, когда результат решения следует непосредственно за выражением.

На самом деле не все выражения являются простыми и требуют применения более сложных режимов вычисления.

Для ввода таких выражений в **MathCad** можно воспользоваться несколькими способами. Самый простой – использовать панели инструментов, доступные в **MathCad**. Так для ввода выражений в программе предусмотрена специальная панель инструментов **Математика** (рисупок 14), которая содержит девять управляющих кнопок. Под каждой кнопкой скрывается соответствующая ей панель инструментов с командами, сгруппированными по определённым свойствам. Например, нажатие на кнопку с изображением калькулятора выводит панель **Арифметика**, содержащую команды ввода цифр, арифметических действий, простейших функций.

Необходимо отметить, что для начинающих пользователей использование панели **Арифметика** позволяет быстро освоить программу и записывать достаточно сложные математические выражения.

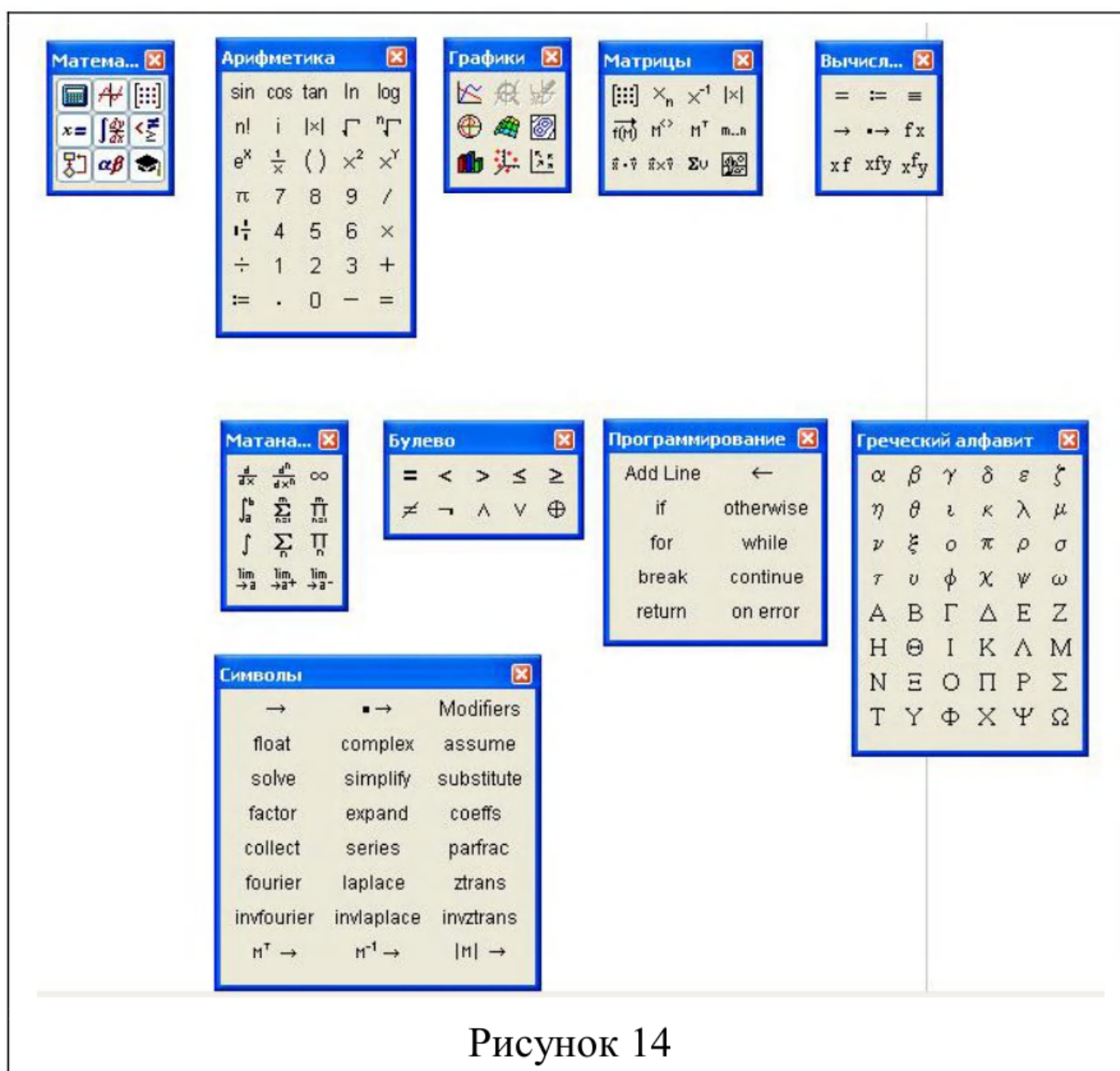


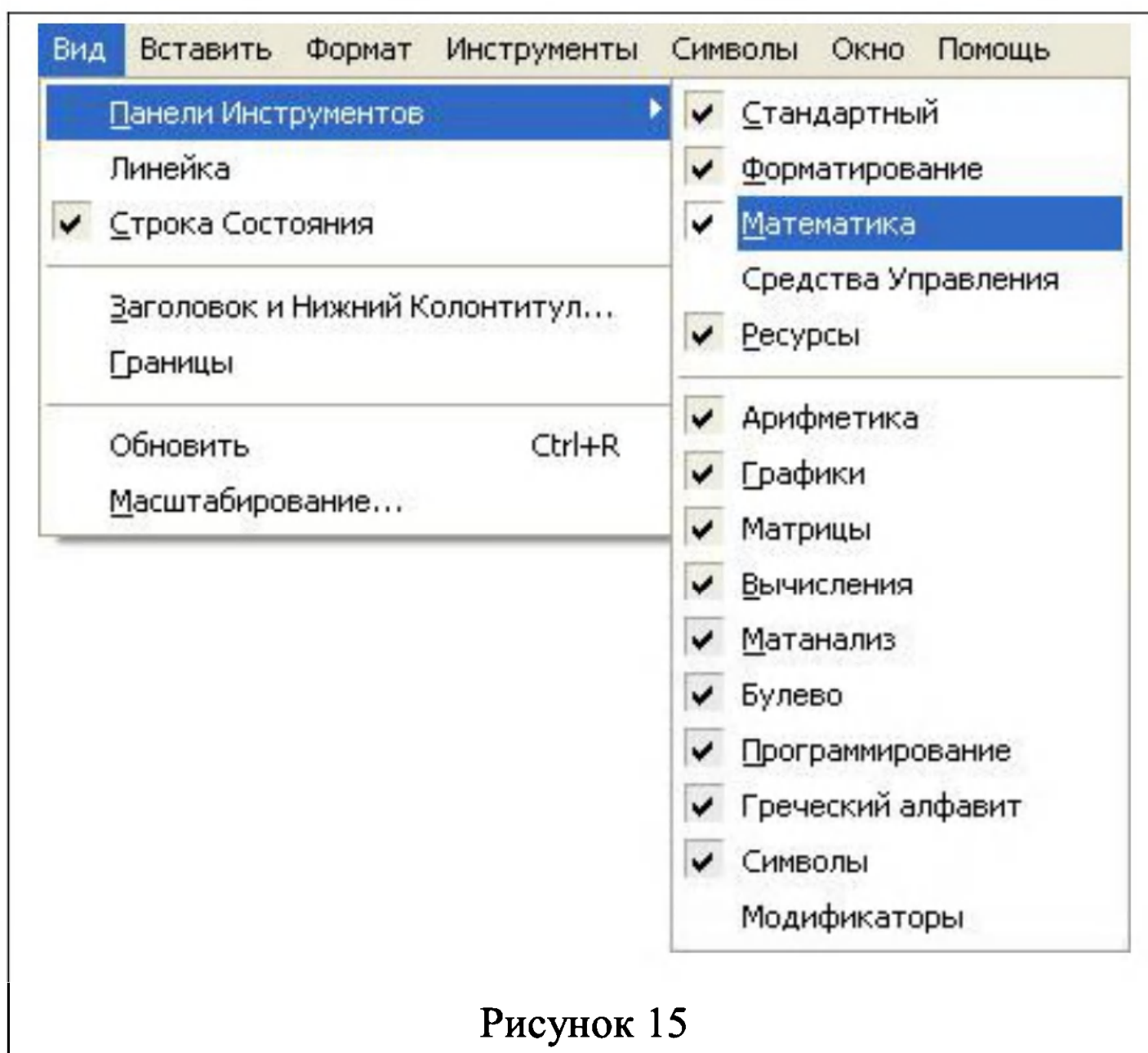
Рисунок 14

Если данная панель не представлена в рабочем окне **MathCad**, то её можно вывести с использованием команд главного меню **Вид**→**Панели инструментов**→**Математика** (рисунок 15).

3.5 Структура документа

Весь документ **MathCad** представляет собой набор блоков, в которых введены данные. Такие блоки подразделяются на два типа:

- текстовые блоки, в которых введены пояснения, описания и т.д.;



– блоки формул – блоки, содержащие выражения для расчёта.

Такие блоки как, например, блоки построения графиков (вывода результатов) являются частным случаем блока формул, т.к. для построения графика зачастую необходимо осуществить вычисление функции, что и делается в поле ввода значения функции. Пример данного решения будет приведён ниже.

Расчёт рабочего документа осуществляется последовательно, в направлении слева – направо и сверху – вниз. Это важно знать, когда в документе существует последовательность зависимых выражений.

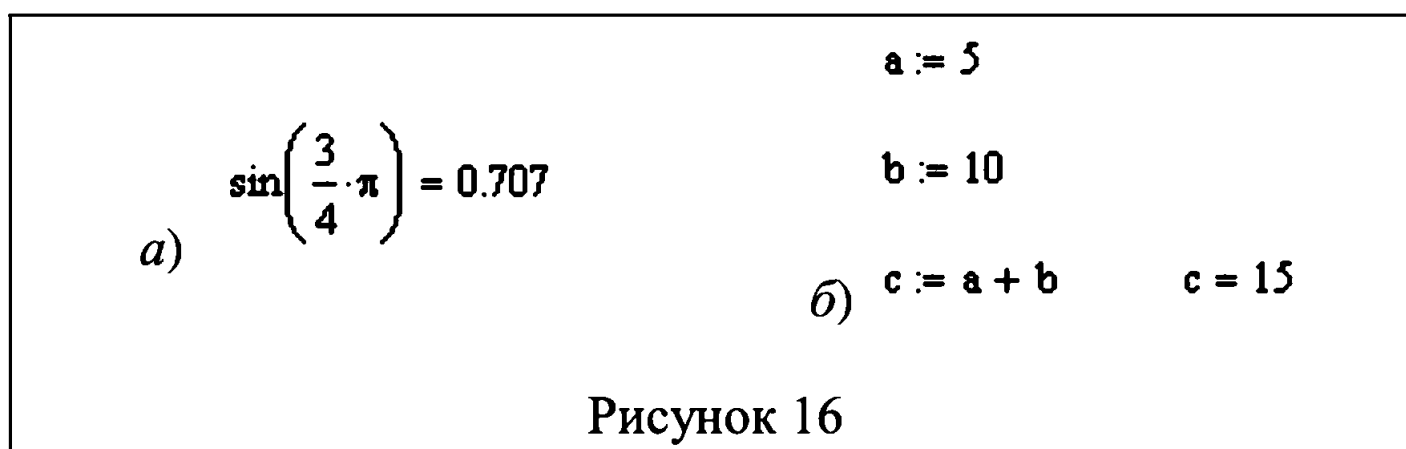
3.6 Ввод математических выражений

При выборе места на листе документа, куда будет вводиться выражение (блок формулы), на рабочем листе появляется курсор ввода, как уже говорилось выше, он определяет верхний левый угол блока. Выбор места осуществляется щелчком мыши.

Для ввода формул и зависимостей, используются команды с панелей, вызываемых из панели **Математика**.

























После определения места в документе, куда будет помещено выражение, в блок вносятся математические выражения. На рисунке 16 представлен пример такого ввода. При использовании режима калькулятора (рисунок 16, а) выражение помещается в одну строчку, при вводе сложных (зависимых) выражений может потребоваться значительно большее количество действий (рисунок 16, б), при этом необходимо отметить, что каждое выражение является блоком.

При вводе сложных выражений необходимо строго соблюдать последовательность ввода. Для этого в блоке формулы служит курсор редактирования, который состоит из двух частей – вертикальной и горизонтальной линий. Вертикальная линия курсора определяет направление записи формулы, горизонтальная – указывает часть выражения, к которой будет относиться ввод.



Рассмотрим последовательность ввода для сложного выражения

$$y = \frac{\sin(3x + 0.5x)}{\cos(0.5x - 3x)} + \frac{3x}{0.5x}.$$

Для того чтобы ввести данное математическое выражение в **MathCad** необходимо осуществить следующую последовательность действий: «у», , , , , «х», «пробел», , , , , , «х», «пробел», «пробел», «пробел», «пробел», , , , , , , «х», «пробел», , , , «х», «пробел», «пробел», «пробел», «пробел», «пробел», , , , «х», «пробел», , , , , , «х». Здесь в кавычках указаны действия, вводимые с клавиатуры, а остальные действия вводятся с панели инструментов **Арифметика**. Как видно в последовательности несколько раз присутствует клавиша «пробел». Дело в том, что при написании выражения курсор редактирования следует за последним введённым элементом формулы. Но иногда возникают моменты, когда курсор должен охватывать не последний элемент, а, например, целое выражение. Для изменения внешнего вида курсора редактирования и предназначено использование пробела при написании формулы. Рассмотрим простой пример неправильного написания формулы (рисунок 17).

При вводе дроби $\frac{2+2}{2}$ в знаменателе дроби стоит сумма,

$a) \quad 2 + \frac{2}{2} = 3$	$б) \quad \frac{2+2}{2} = 2$	$в) \quad \frac{(2+2)}{2} = 2$
Рисунок 17		

если не отнести знак дроби к сумме, то получится совсем другой результат (рисунок 17, а). Фактически использование клавиши пробел заменяет использование скобок при написании математических выражений (рисунок 17, в). Использование же скобок в **MathCad**, так же как и в арифметике, повышает приоритет действия.

3.7 Ввод текста

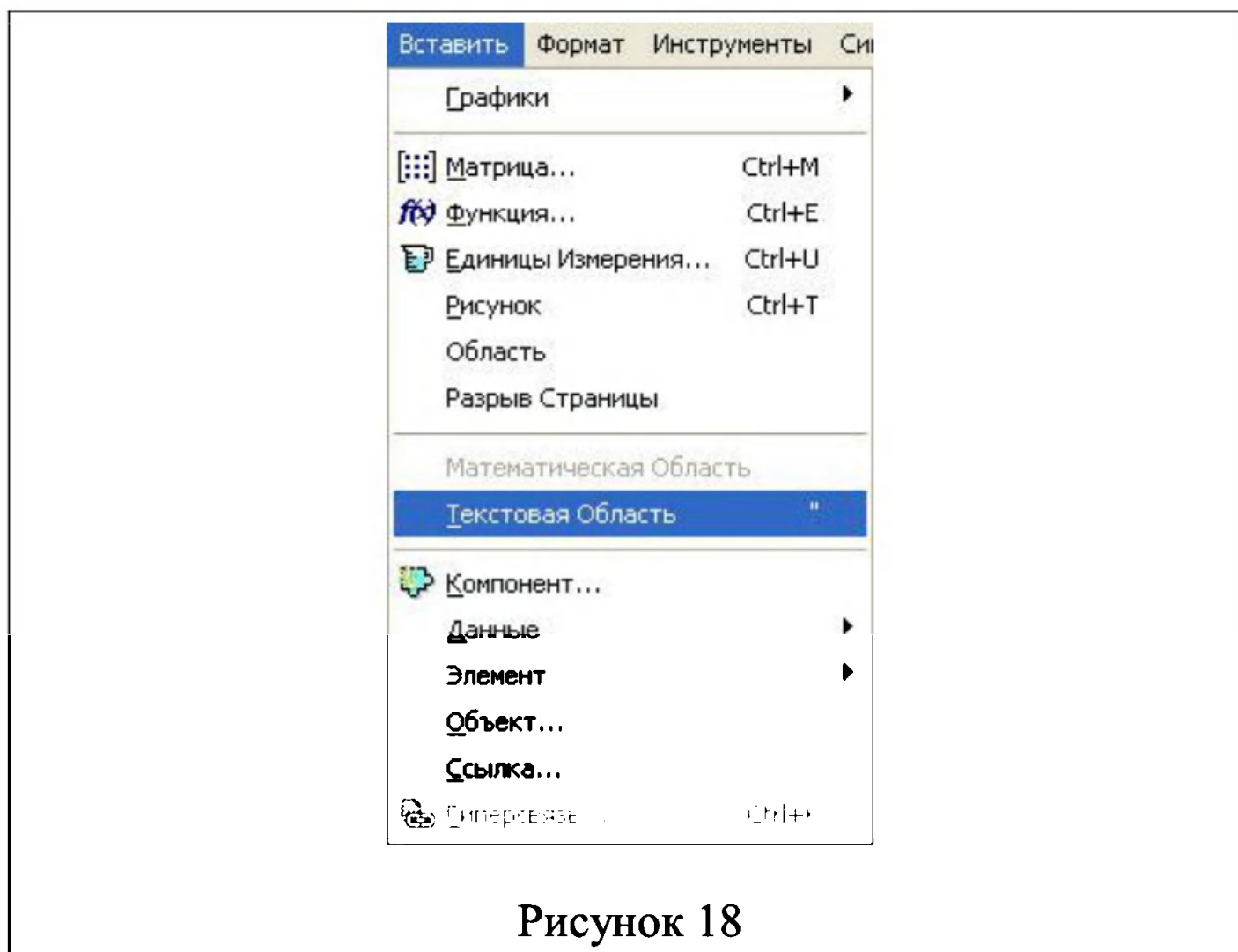
В **MathCad** наиболее часто используются блоки формул, но для составления пояснительной записки в ней необходимо использовать пояснения, замечания и т.д. Таким образом, необходимо осуществлять ввод текстовых блоков. Если в **MathCad** начать вводить символы, то программа расценит это как ввод имени переменной и соответственно присвоит этому блоку статус формулы. Для того, чтобы в документе корректно вводился текстовый блок необходимо использовать специальную команду главного меню **Вставить→Текстовая область** или ввести с клавиатуры символ кавычек (рисунок 18).

Редактирование внутри текстового блока осуществляется также как и в любом другом текстовом редакторе. Для выхода из текстового блока применяется щелчок мыши в поле документа или нажатие клавиши **→**.

Текстовый блок имеет отличительные от формульного границы – на них присутствуют маркеры изменения размера, «потянув» за которые, можно изменить размер блока.

3.8 Операторы присваивания

На рисунке 16 видно, что при использовании различных режимов расчётов в **MathCad** знаки равенства различны. Дело в том, что во втором случае (рисунок 16, б) до вывода результата, где используется знак равенства,



осуществляются присваивания исходных данных и выражения переменным. Фактически знак $:=$ является оператором присваивания. Рассмотрим возможные варианты использования операторов присваивания в **MathCad**:

$=$ — оператор присваивания, осуществляет присваивание переменной числовых значений или математического выражения;

\equiv — оператор глобального присваивания, позволяет осуществлять присваивание в любом месте программы;

\rightarrow — выразить символически;

\leftarrow — оператор локального присваивания, осуществляет присваивание переменной числовых значений или математических выражений внутри блока программы.

Все эти операторы представлены на панелях инструментов **Вычисления** и **Программирование**.

Помимо этих операторов присваивания существует ещё два символа равенства, которые используются для:

- = – определения равенства величин (знак отношения);
- = – вывода результата.

3.9 Понятие переменной

В **MathCad** как и в любом другом математическом процессоре допускается присваивание численных значений переменным. В качестве имени переменной в **MathCad** используется буква или сочетание букв английского алфавита. Необходимо отметить, что используя в имени переменной различный регистр, **MathCad** воспринимает эти переменные как разные. Т.е. X и x – для **MathCad** разные величины. То же правило распространяется и на использование разного шрифта (например, x и \mathcal{X}).

Так как наиболее часто используется режим, когда осуществляется ввод зависимых выражений, то необходимо знать, что в **MathCad** существует шесть типов переменных, используемых в вычислениях:

- **простая переменная** – характеризуется одним числовым выражением, пример ввода $x := 2$;
- **дискретная переменная** – переменная, изменяющаяся на определённом интервале с определённым шагом, пример ввода $x := 0, 0.1, \dots, 10$;
- **функциональная переменная** – переменная, принимающая свои значения на основе аргумента, пример ввода $x := 0, 0.1, \dots, 4 \cdot \pi$, $y(x) := \sin(x)$;
- **локальная переменная** – переменная, используемая в блоке программирования, её значения не передаются из программы в общий документ, пример ввода $x \leftarrow 12$;

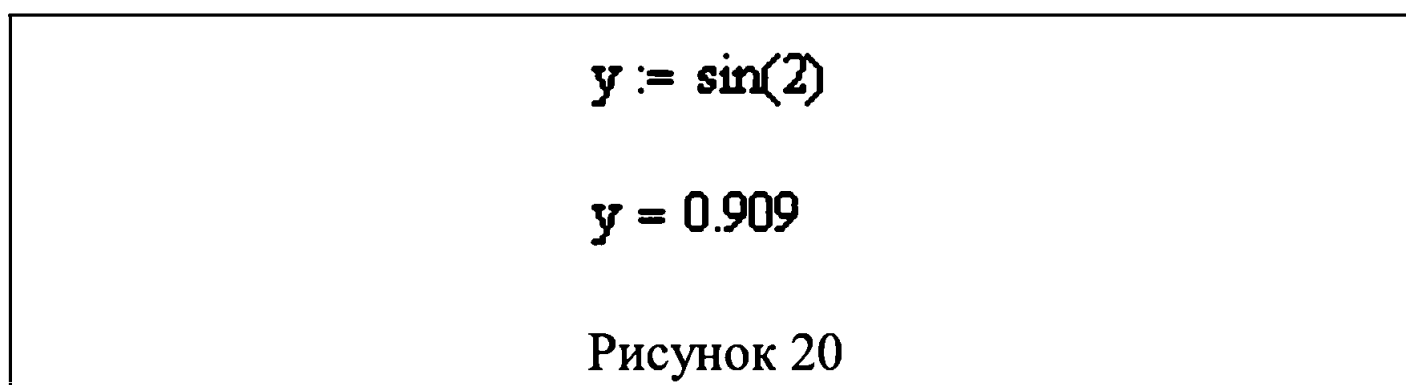
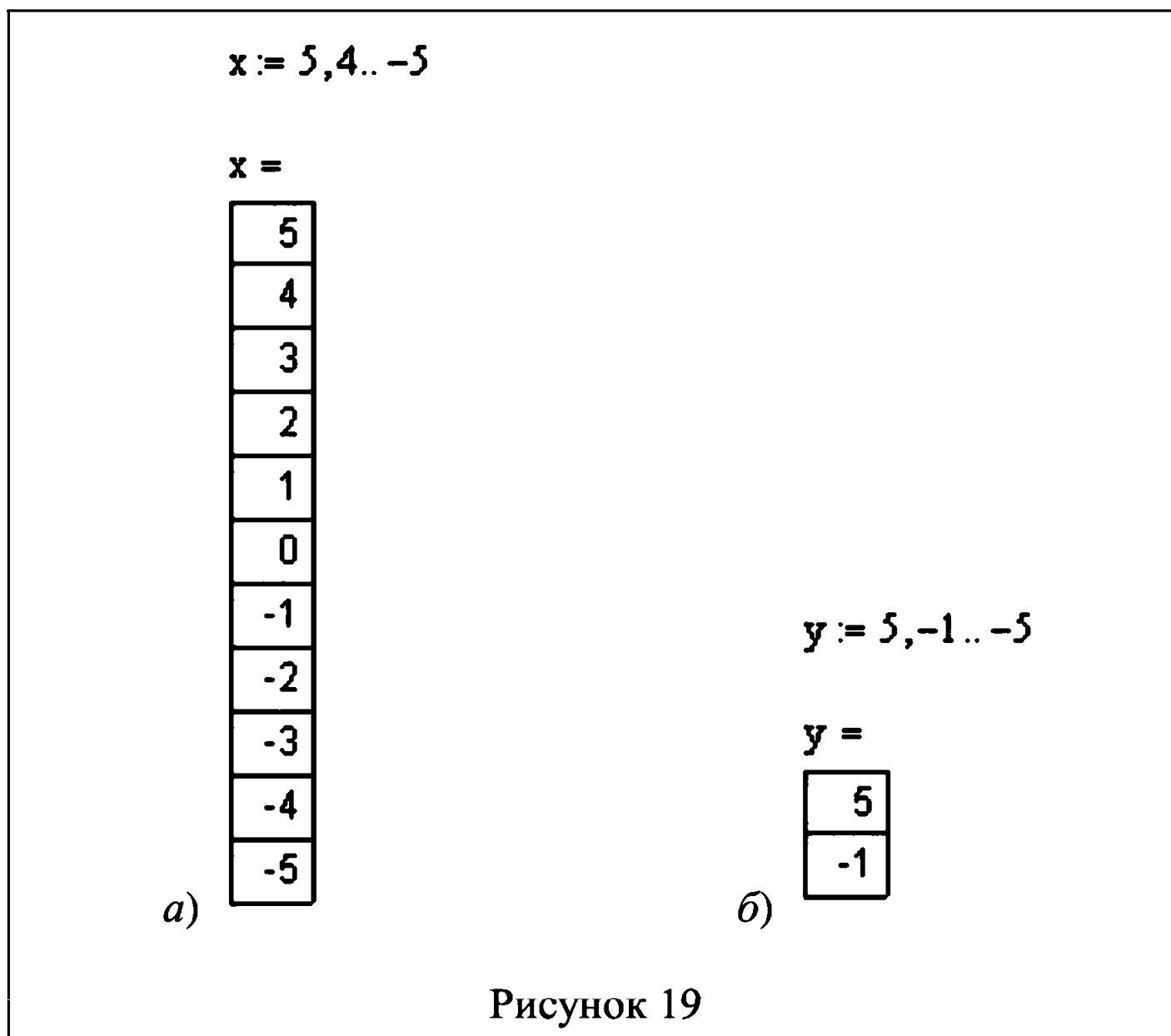
– **индексированная переменная** – переменная, используемая при определении матриц, характеризуется индексом – целым положительным числом, пример ввода $a_1 := 10$;

– **глобальная переменная** – содержит единственное числовое значение, действие этой переменной распространяется на весь документ, т.е. независимо от того, где определена глобальная переменная, она может использоваться в любом месте документа, пример ввода переменной $const \equiv 3.142$.

При использовании переменных существуют следующие особенности

Для ввода дискретной переменной используется команда m...n с панели инструментов **Матрицы**. При использовании в данном случае двух точек программа выведет сообщение об ошибке. После запятой в дискретной переменной указывается не шаг переменной, а следующее значение, т.е. начальное значение плюс шаг. Это особенно важно знать, когда вычисления осуществляются с отрицательным шагом. На рисунке 19. представлены правильный (а) и неправильный (б) ввод дискретной переменной с шагом -1 . В дискретной переменной, изменяющейся с шагом 1 указывать следующее значение необязательно.

При вводе функциональной переменной в скобках обязательно должно указываться имя переменной являющейся аргументом функции – это может быть простая или дискретная переменная. Допускается использование функциональной переменной по нескольким аргументам, например $t(x, y, z) := \sin(x) + \sin(y) + \sin(z)$.



При вводе индексированной переменной часто вместо матричной величины используют простую переменную с подписью и наоборот. Переменная, характеризующая матричную величину (индексированная переменная), вводится с помощью команды x_n , а простая переменная с подписью вводится с помощью «.» (точки) вводимой с клавиатуры после основного имени переменной.

3.10 Вывод результатов

Как уже говорилось выше результат вычислений можно получить, применив команду $=$ с панели инструментов **Арифметика** или **Вычисления**.

При решении простого выражения ответом будет являться числовое значение (рисунок 20). Если решение представляет из себя вычисление функции по многим аргументам – функциональную переменную, то вывод результата будет представлен в табличной форме (рисунок 21). В данном примере явно видно, что количество решений превышает количество выведенных результатов. Это вызвано тем, что некоторые решения могут быть представлены значительными количествами результатов, которые просто невозможно вывести на поле документа. Поэтому количество результатов, выводимых на экран, составляет 16. Если необходимо проконтролировать результаты, не выведенные на экран или в документ, то можно осуществить щелчок на блоке результатов и справа от таблицы результатов будет выведена полоса прокрутки, которая позволит переместиться в любое место таблицы (рисунок 22).

При использовании **MathCad** для расчётов можно воспользоваться расчётами с помощью функций. По внешнему виду данная запись напоминает ввод функциональной переменной, но отличается тем, что аргумент функции на начальном этапе не определён (рисунок 23). Такое действие в документе **MathCad** называется «задание функции пользователя». Определение аргумента для такой функции может быть осуществлено разными способами: заданием непосредственно переменных (рисунок 23, а) или вводом аргументов непосредственно в функцию (рисунок 23, б)

$x := 0, 0.1 \dots 4 \cdot \pi$

$y(x) := \sin(x)$

$y(x) =$

0
0.1
0.199
0.296
0.389
0.479
0.565
0.644
0.717
0.783
0.841
0.891
0.932
0.964
0.985
0.997

Рисунок 21

$x := 0, 0.1 \dots 4 \cdot \pi$

$y(x) := \sin(x)$

$y(x) =$

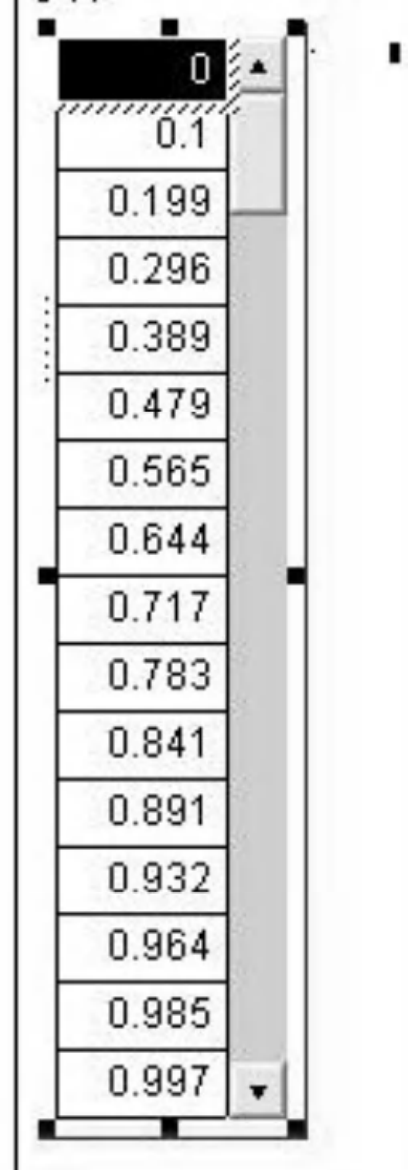


Рисунок 22

$x := 10$

$y(x) := \sin(x)$

a) $y(x) = -0.544$

$y(x) := \sin(x)$


$y(10) = -0.544$

b)

Рисунок 23

3.11 Построение графиков

При расчётах, особенно значений функций, часто бывают ситуации, когда в результате получается большое количество числовых значений. Вывод такого результата в табличной форме не всегда можно оценить. Например, когда берется временная функция на значительном интервале времени с минимальным шагом по времени (для обеспечения точности решения). В таком случае вместо табличной формы в качестве представления результата можно воспользоваться графическим модулем **MathCad**.

На панели инструментов **Математика** есть команда  вызова панели **Графики** (рисунок 24).

С помощью этой панели можно использовать графический модуль **MathCad** для решения и визуального представления данных.


Наиболее часто используется команда построения обычного двумерного графика . При использовании данной команды на экране появляется заготовка для графика (рисунок 25), в которой необходимо указать два параметра: аргумент функции на горизонтальной оси и функциональную переменную, которая содержит результат вычислений – на вертикальной.



Рисунок 24

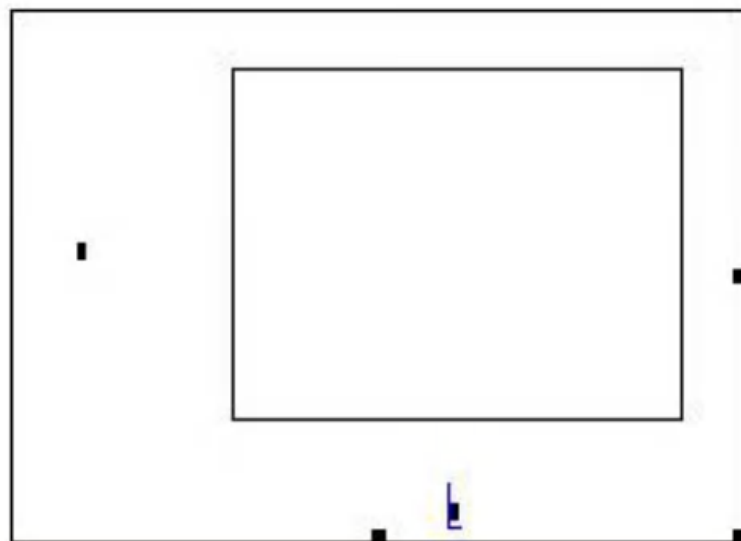


Рисунок 25

Пример использования данной команды для построения функции

$$y = \sin x \cdot e^{-0.1x}$$

приведён на рисунке 26. Последовательность действий для построения графика состоит в вводе аргумента функции – переменной x , вводе рассчитываемой функции y , построении графика.

На график можно выводить значения нескольких функций, полученных по одной переменной (рисунок 27). Для этого необходимо в знакоместе, где вводится функция, поставить запятую, после чего в этом месте появится второе знакоместо для ввода дополнительной функции. Пример такого использования графического модуля показан на рисунке 28.

$x := 0,0.1..10 \cdot \pi$

$y(x) := \sin(x) \cdot e^{-0.1 \cdot x}$

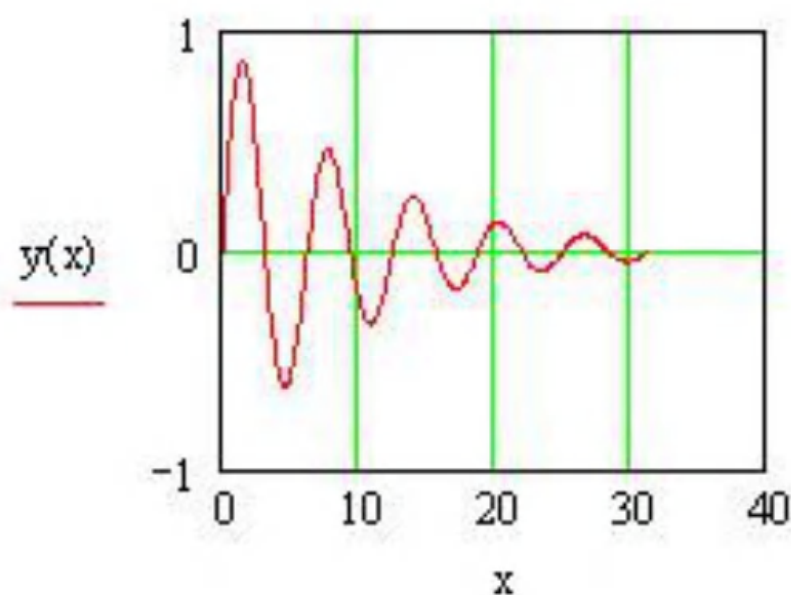


Рисунок 26

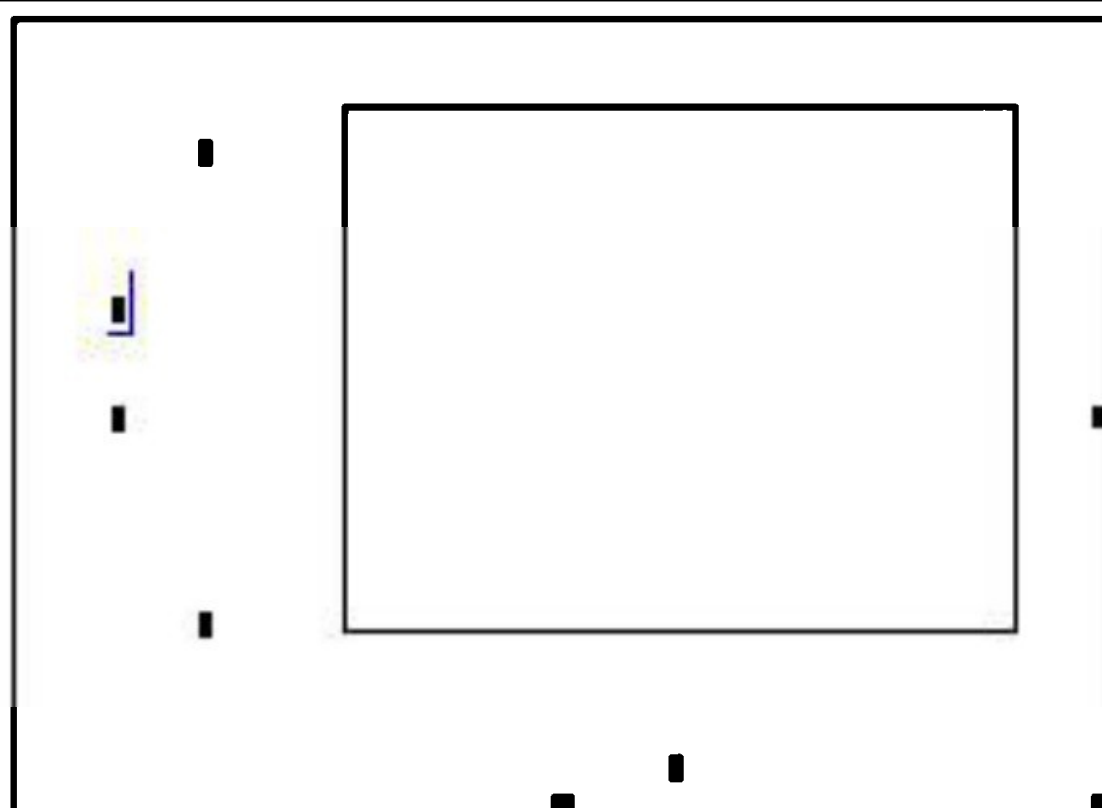


Рисунок 27

$x := 0,0.01 \dots 4\pi$

$y(x) := \sin(x)$ $z(x) := \cos(x)$

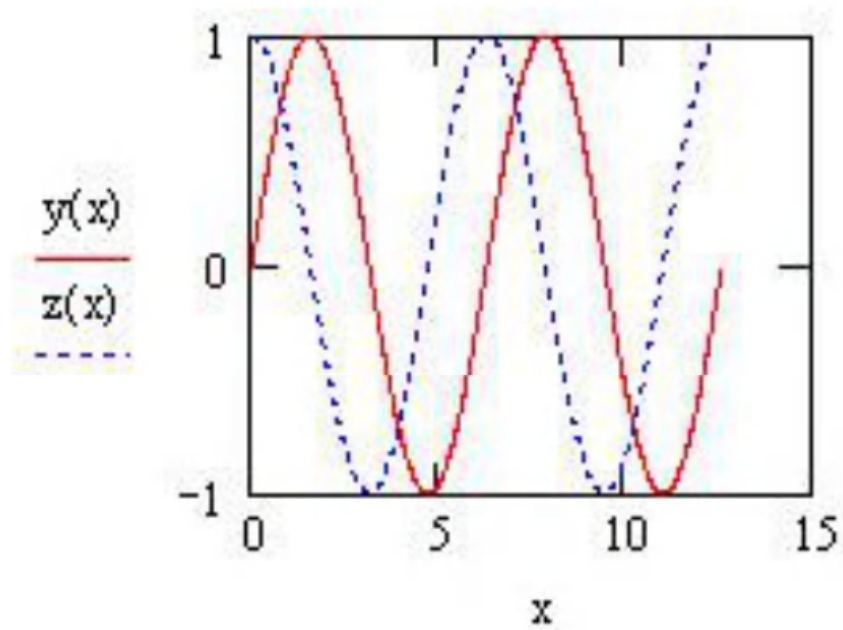



Рисунок 28

Параметры оформления графика в документе **MathCad** вызываются двойным щелчком по полю графика. Изменяемыми параметрами являются: наличие сетки, представление осей, представления линий графика и т.д. Панель параметров графика представлена на рисунке 29.

Помимо построения двумерного графика пакет **MathCad** позволяет осуществлять построение трёхмерных графиков – графиков поверхности. Для этого значения функций, определяющие график поверхности необходимо перевести в матрицу, имя которой указывается при построении графика. Для вызова заготовки графика поверхности на панели **Графики** (рисунок 24) необходимо выбрать команду .

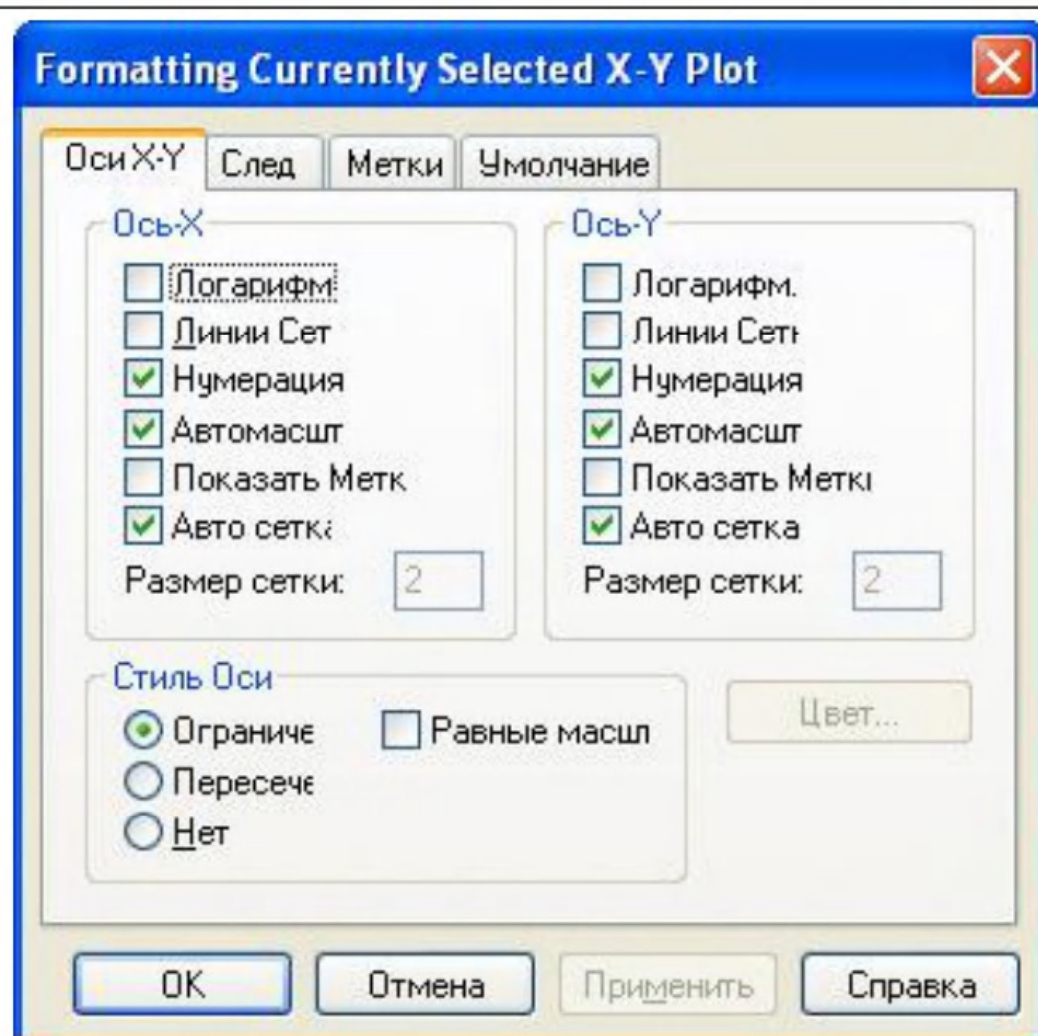
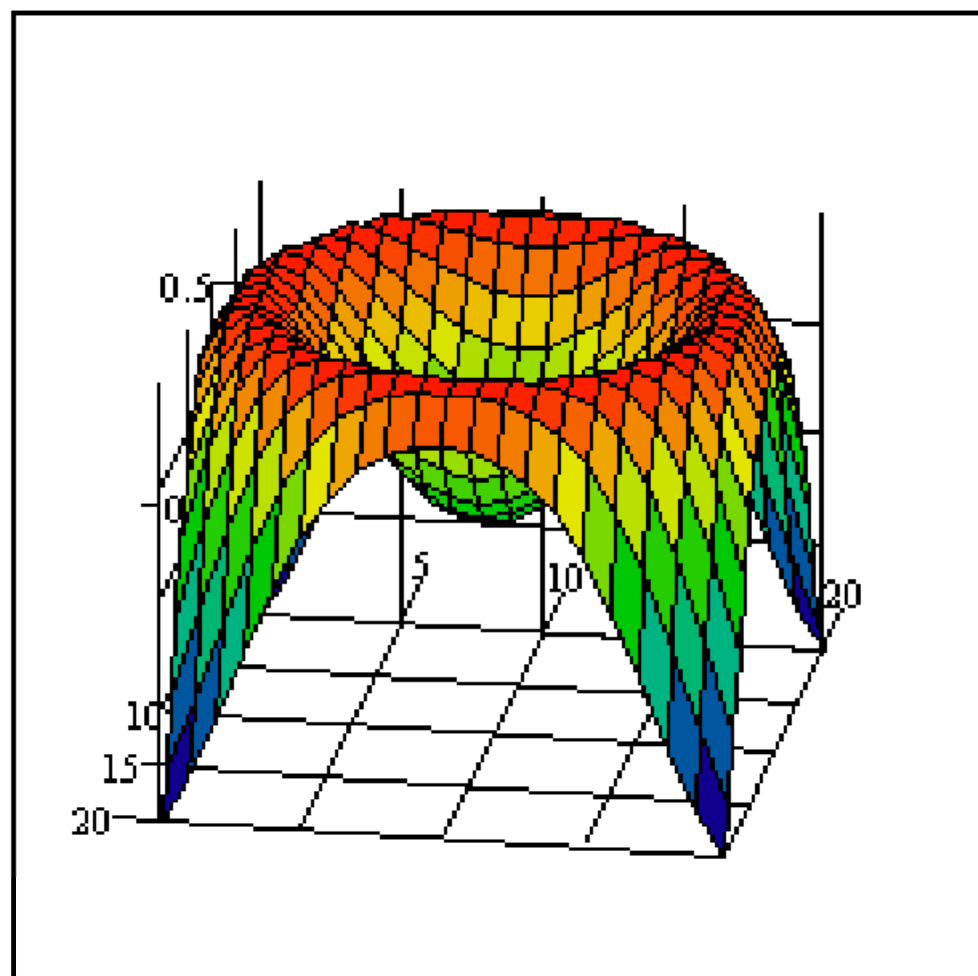


Рисунок 29

Пример построения графика поверхности представлен на рисунке 30.

```
n := 20  
i := 0..n      bi := -1.5 + 0.15·i  
j := 0..n      cj := -1.5 + 0.15·j  
f(b, c) := sin(b2 + c2)  
Mi,j := f(bi, cj)
```



M


Рисунок 30

3.12 Программирование в MathCad

В **MathCad** содержатся конструкции, во многом подобные конструкциям, содержащимся в языках программирования: условные операторы передачи управления, операторы циклов, области видимости переменных, использование подпрограмм и рекурсии.

Программа в **MathCad** – это выражение, состоящее из последовательности операторов, каждый из которых является в свою очередь выражением.

Как и любое выражение, программа выдаёт в качестве результата значение, которое является значением последнего выражения, выполняемого программой.

Для вставки в **MathCad** блока программы имеется специальная панель **Программирование** (рисунок 14), которая содержит все операторы доступные для программирования. Вызывается панель **Программирование** нажатием на кнопку  панели **Математика**.

Блок программы в **MathCad** (группа операторов) обозначается сплошной вертикальной чертой, которая ставится слева от заключённых в программу операторов. Для начала блока необходимо выбрать команду **Add Line**, после этого в документе появляется заготовка блока программы (рисунок 31, а), при необходимости использования нескольких операторов в блоке программ нужно применить команду несколько раз (рисунок 31, б).

3.13 Оператор условного перехода

В процессе решения задачи встречаются случаи, когда необходимо выполнять действие в зависимости от выполнения или невыполнения некоторого условия. Этого можно добиться с помощью операторов условного перехода **If** и **Otherwise**. Обе команды находятся на панели **Программирование** (рисунок 14).

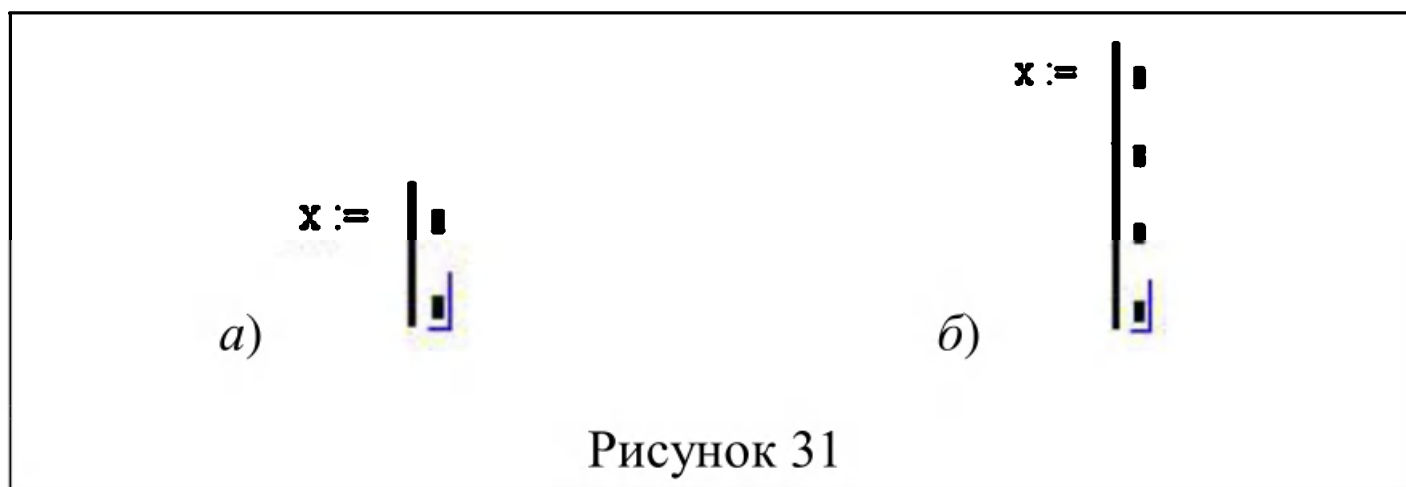


Рисунок 31

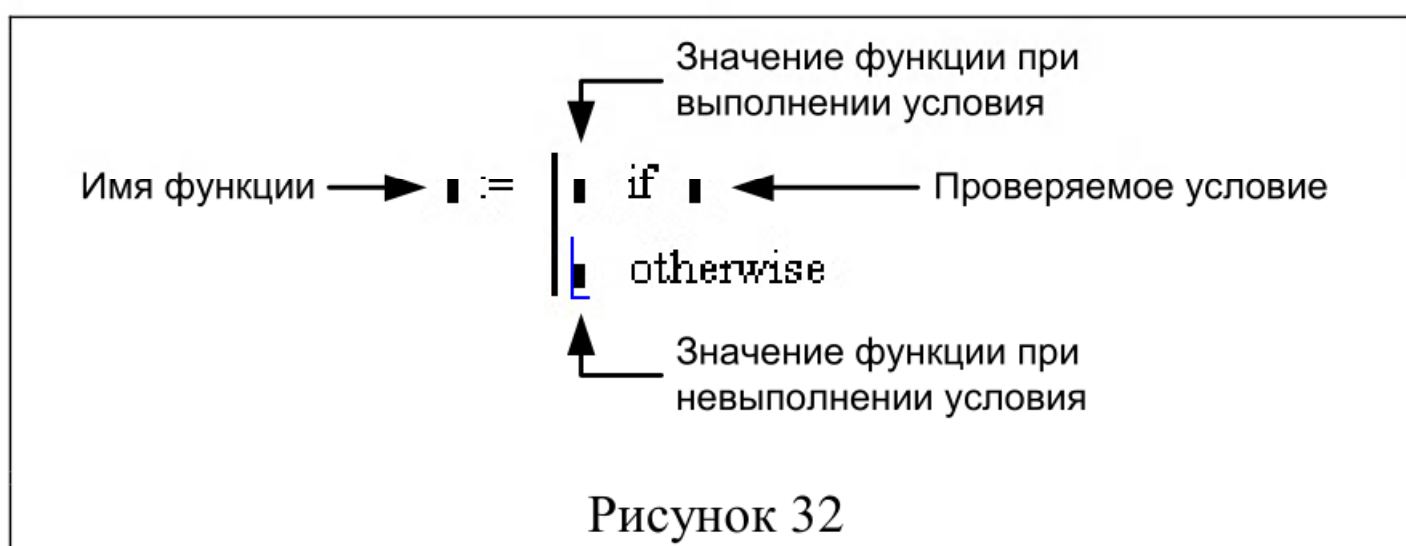


Рисунок 32

В качестве параметров оператора условного перехода указываются следующие величины – рисунок 32. Примером использования операторов условного перехода является решение функции по условию:

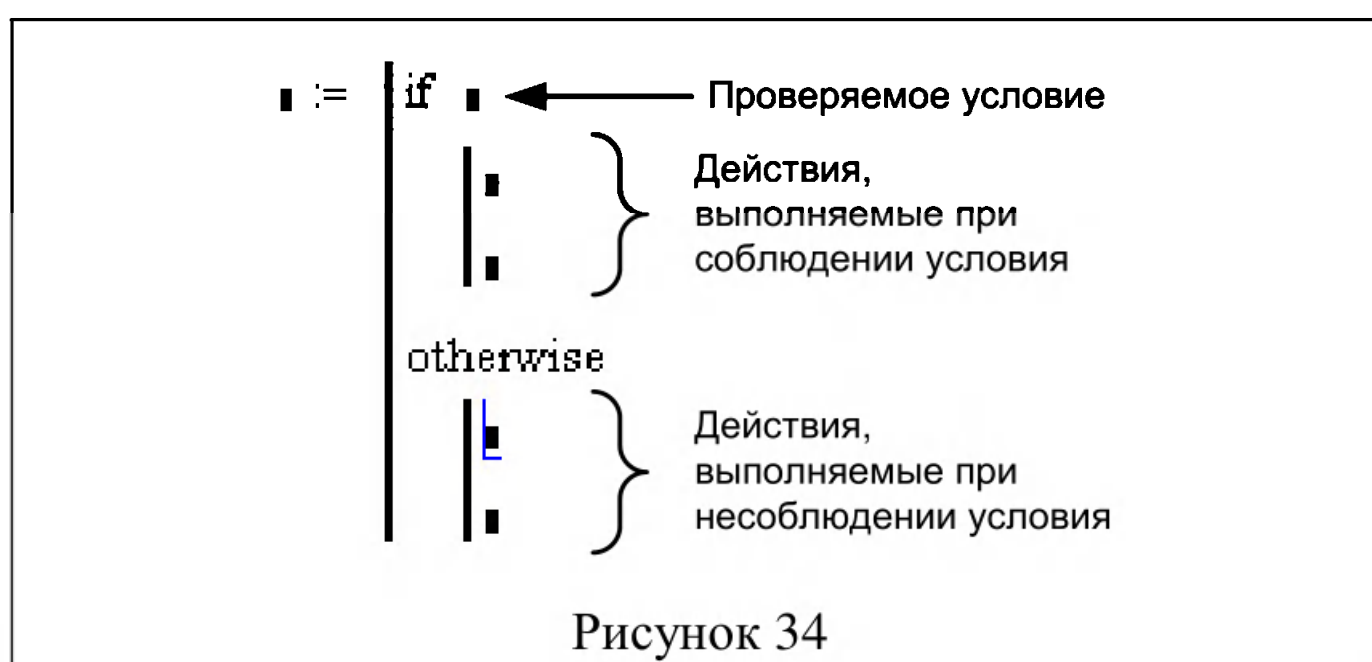
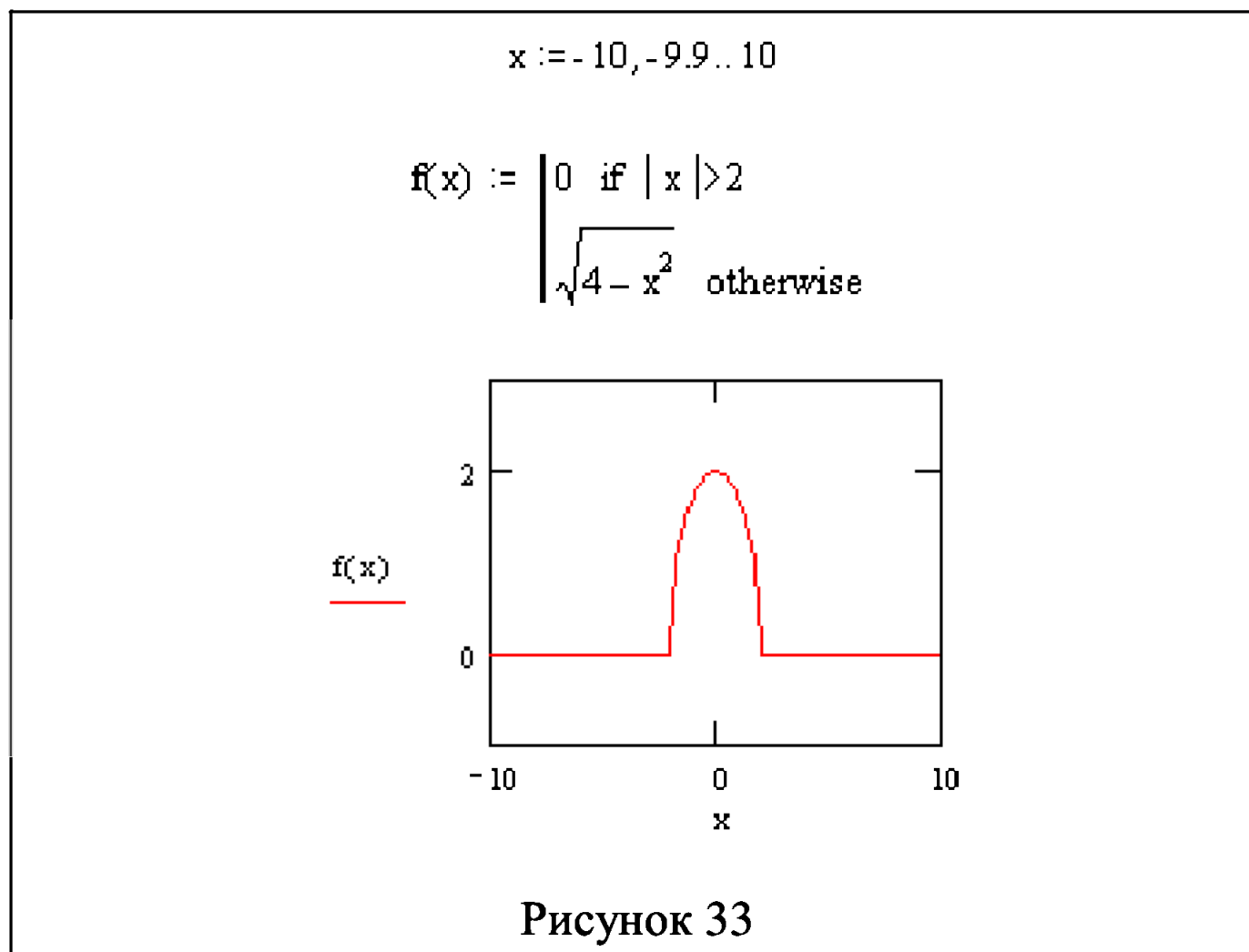
$$f(x) = \begin{cases} 0 & \text{при } |x| > 2 \\ \sqrt{4 - x^2} & \text{при } |x| \leq 2 \end{cases},$$

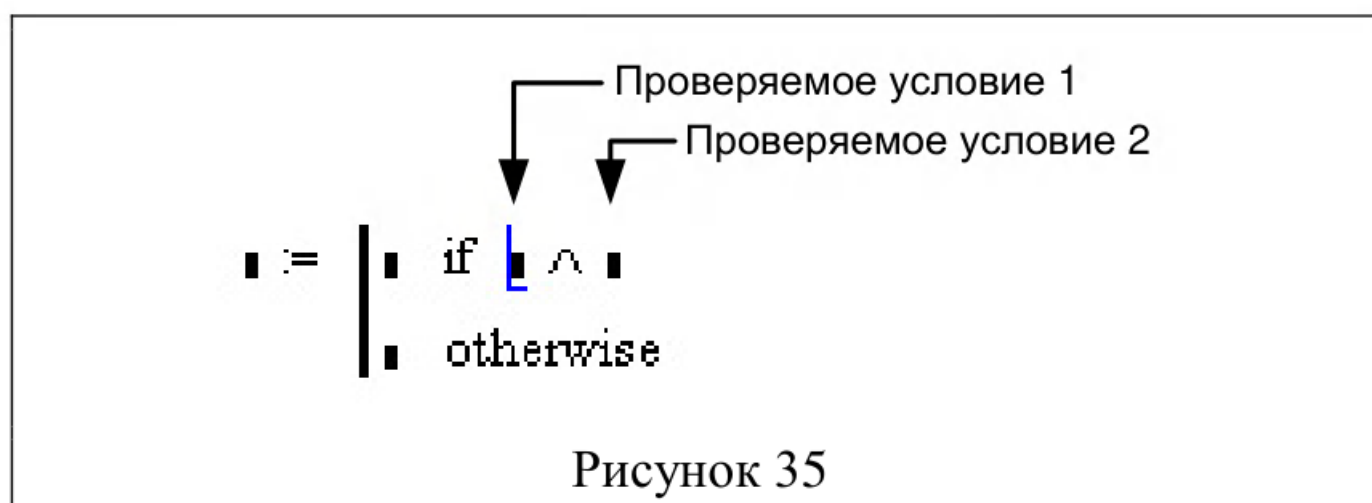
где x – изменяется в интервале от -10 до 10 .

Решение этой задачи в **MathCad** будет иметь вид, представленный на рисунке 33.

Если необходимо под оператором условного перехода объединить несколько действий, то можно сформировать блок операторов с помощью применения команды **Add Line**, которая применяется в метке «**Значение функции по**

умолчанию». В этом случае будет создан блок операторов второго уровня, выполнение которых будет осуществляться при выполнении условия (рисунок 34). То же справедливо и для использования блока действий под оператором **Otherwise**.





Для выполнения действия с одновременной проверкой нескольких независимых условий можно использовать оператор объединения \wedge с панели инструментов **Булево**. Пример заготовки для такой формы записи представлен на рисунке 35. Для проверки выполнения хотя бы одного условия используется оператор исключения \vee .

3.14 Операторы циклических вычислений

Циклические вычисления в **MathCad** проводятся с помощью операторов **For** и **While**, находящихся на панели инструментов **Программирование**.

В **MathCad** имеется два типа цикла

- если известно необходимое число операций, то применяется оператор **For** – рисунок 36;
- если число циклов неизвестно или цикл управляется истинностью некоторого условия, то применяется цикл **While** – рисунок 37.

Пример использования оператора **For** будет приведён ниже. Примером программы с применением оператора **While** может быть программа нахождения номера элемента вектора в зависимости от заданной величины функции (рисунок 38). В результате величина 1,98 превосходитя восьмым элементом вектора.

$\text{■} := \text{for } \text{■} \in \text{■}$
 ■

Имя переменной цикла
 Диапазон значений в котором
 изменяется переменная цикла
 Тело цикла

Рисунок 36

$\text{■} := \text{while } \text{■}$
 ■

Условие выполнения цикла
 Тело цикла

Рисунок 37

```

m := 0..2500
Vm := 1 + sin(m)
t(V, th) :=
  j ← 0
  while Vj ≤ th
    j ← j + 1
  j
t(V, 1.98) = 8
  
```

Рисунок 38

Так же как и в других языках программирования в **MathCad** допускается использование вложенных циклов. Если в теле цикла необходимо объединить несколько операторов, то используется команда **Add Line**.

3.15 Оператор Break

Оператор **Break** используется для выхода из цикла или остановки выполнения программы при выполнении некоторого условия. Пример использования оператора **Break** приведён на рисунке 39. Решение предыдущей задачи в данном случае – это вывод номера элемента и его величины.

Все программные блоки, используемые в **MathCad**, могут быть представлены тремя различными типами программ.

```

m := 0..2500

Vm := 1 + sin(m)

t(V, th) := | j ← 0
              break if max(V) ≤ th
              while Vj ≤ th
                j ← j + 1
              [ j ]
              [ Vj ]

t(V, 1.98) = [ 8 ]
              [ 1.989 ]

```

Рисунок 39

$$s \equiv \left| \begin{array}{l} s \leftarrow 0 \\ \text{for } i \in 1..64 \\ \quad \left| \begin{array}{l} \text{return } s \text{ if } s \geq 10^{12} \\ n \leftarrow 2^{i-1} \\ s \leftarrow s + n \end{array} \right. \end{array} \right| s$$

$$s = 1.0995 \cdot 10^{12}$$

Рисунок 40

$$x := 9$$

$$y := \left| \begin{array}{l} \sqrt[3]{|x|} \text{ if } x < 0 \\ e^x \text{ if } 0 \leq x \leq 10 \\ \int_0^x \frac{\sin(t)}{t} dt \text{ if } x > 10 \end{array} \right|$$

$$y = 8.103 \cdot 10^3$$

Рисунок 41

$$V(r, h) := \left| \begin{array}{l} s \leftarrow \pi r^2 \\ v \leftarrow \frac{1}{3} \cdot s \cdot h \end{array} \right|$$

$$V(1, 2) = 2.094$$

Рисунок 42

3.16 Программа-константа

Переменная при выполнении этой программы будет принимать единственное значение (рисунок 40).

3.17 Программа-переменная:

В результате выполнения программы переменная y принимает значение, зависящее от начального x (рисунок 41).

3.18 Программа-функция:

Нахождение значения функции осуществляется вне блока программирования по явно заданным значениям аргументов r и h (рисунок 42).



3.19 Работа с массивами

Массив – упорядоченная и пронумерованная последовательность чисел. Каждый массив состоит из элементов, которые характеризуются адресом. В качестве адресов используются целые положительные числа.

В то время как простые переменные содержат всего одно значение, массивы хранят много значений. Всего имеется три способа формирования массива:

- заполнение пустых полей (для небольших массивов);
- заполнение с помощью дискретного аргумента (имеется функция для вычисления элемента массива через его индекс);
- считывание массива из файлов данных.

Различают несколько типов матриц: одномерные (вектор), двумерные и тензоры (многомерные или вложенные массивы).

Организация массива в **MathCad** осуществляется с помощью команды **Вставить→Матрица** (рисунок 43), главного меню программы, или с помощью команды , панели инструментов **Матрицы**. Напомним, что данная панель вызывается кнопкой  с панели инструментов **Математика** (рисунок 14). При использовании команды вставки массива на экране появляется запрос о параметрах матрицы (рисунок 44), в которой указывается количество строк и столбцов в создаваемой матрице. После подтверждения введенных значений командой **Ok**, в документе будет создана основа матрицы с количеством знакомест соответствующих количеству элементов матрицы (рисунок 45). В этом же запросе осуществляют ввод параметров исправления матриц: удаления или добавления строк или столбцов. Для этого используют кнопки **Вставить** и **Удалить** (рисунок 44).

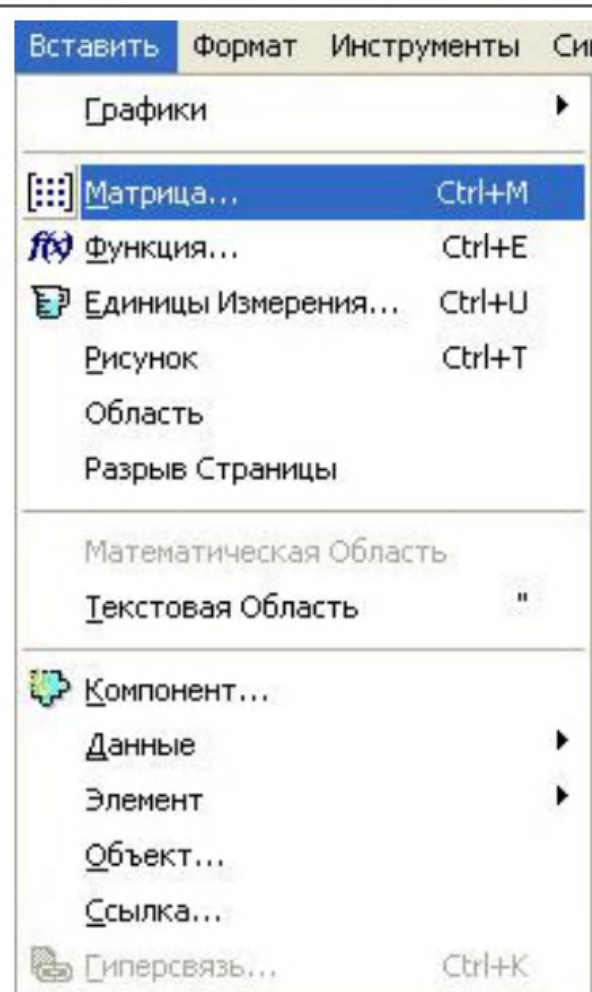


Рисунок 43

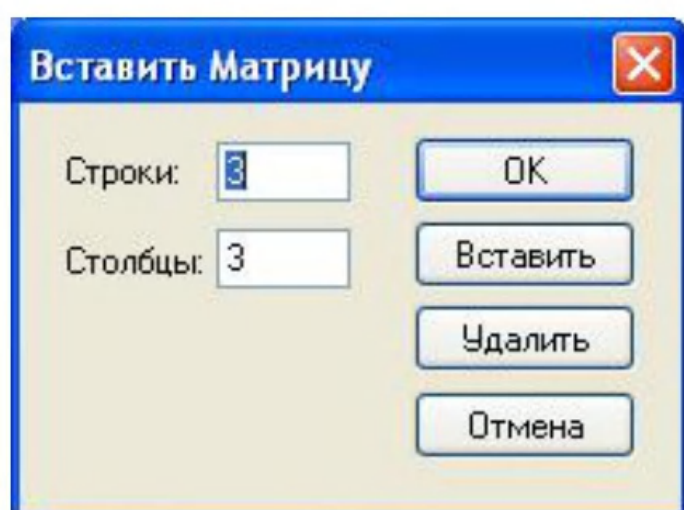


Рисунок 44

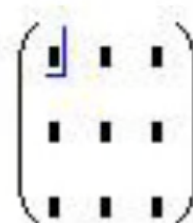


Рисунок 45


Начало нумерации элементов массива начинается с нуля. Это определено системной переменной **ORIGIN**, изменение данной переменной позволяет изменить номер первого элемента массива. При этом рекомендуется переменную **ORIGIN** определять оператором глобального

присваивания. Это гарантирует одинаковое определение всех массивов во всём документе.

При работе с массивами элементы можно задавать или выводить как в целом, так и индивидуально, для этого используется команда \times_n с панели инструментов **Матрицы**. Если используется многомерный массив, то номер элемента массива указывается через запятую (рисунок 46).

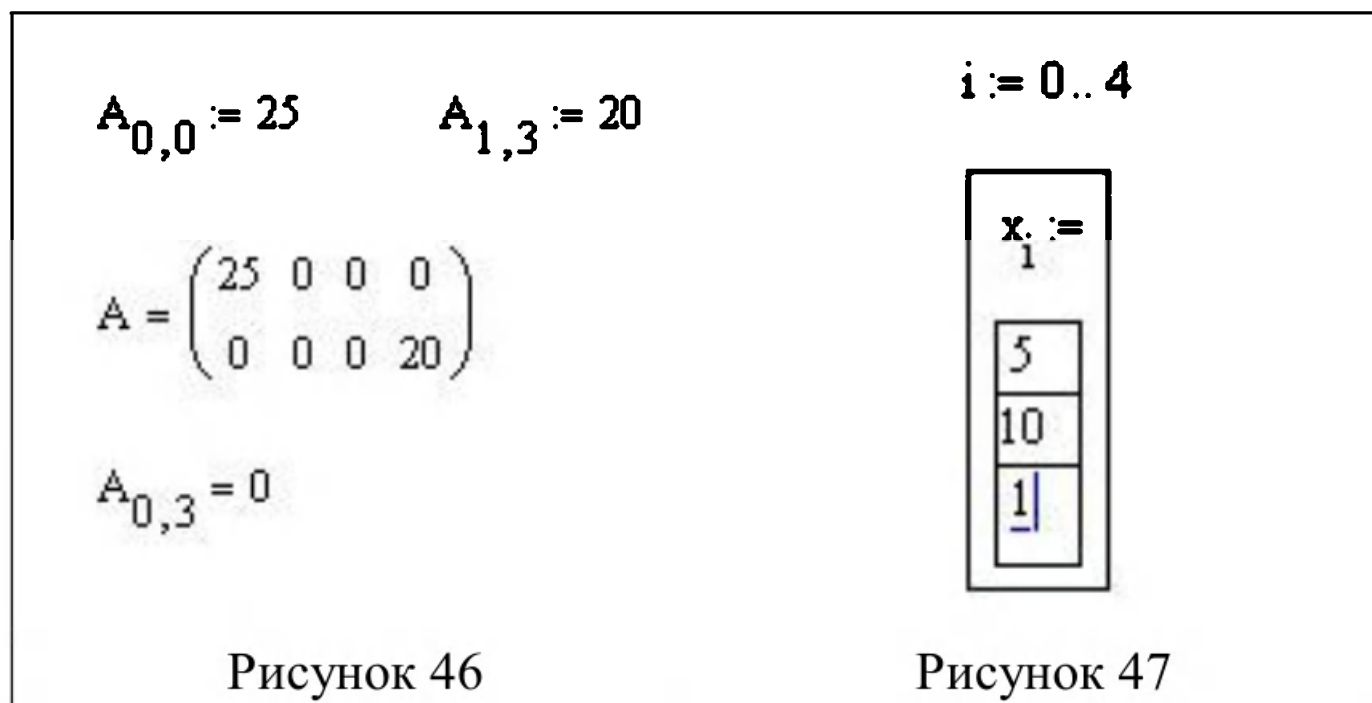
При необходимости вывода на экран столбца матрицы используется команда '' панели инструментов **Матрицы**, где в верхних угловых скобках указывается номер выводимого столбца. Если необходимо вывести строку матрицы, то перед выводом необходимо применить команду транспонирования элементов матрицы ''^T , после чего используется команда вывода элемента массива.

Помимо этого, индексированные переменные применяются при выводе результатов из-под команды программирования **Add Line**. Пример такого использования индексированной переменной представлен на рисунке 39.

Для изменения размера матрицы необходимо щёлкнуть на элементе массива и выбрать команду  на панели инструментов **Матрицы**. После появления запроса (рисунок 44) необходимо в поля **Строки** или **Столбцы** ввести количество изменяемых параметров, далее для их добавления удаления необходимо нажать кнопку **Вставить**, для удаления – **Удалить**.

При работе с массивами можно использовать следующие действия:

- умножение матрицы на скаляр;
- умножение матрицы на матрицу;
- деление элементов матрицы на скаляр;
- сложение матриц;
- увеличивает элемент матрицы на скаляр;
- вычитание матриц;



- уменьшение элементов матрицы на скаляр;
- возведение матрицы в степень (степень n – целое число, если n – меньше 0, то это степень обращённой матрицы);
- обращение матрицы (возведение в степень -1);
- транспонирование;
- сортировка значений;
- суммирование элементов.

3.20 Ввод числовых значений в таблицу


Для ввода числовых значений матрицы-вектора необходимо задать параметры счётчика и осуществить ввод числовых значений переменной через запятую. На экране будет сформирована таблица, содержащая значения вектора (рисунок 47).

3.21 Сортировка элементов массива

При обработке экспериментальных данных иногда существует необходимость сортировки данных. Для того чтобы отсортировать данные по возрастанию их величины

необходимо применить одну из трёх функций сортировки, из списка встроенных функций:

- **sort(V)** – для сортировки элементов вектора (матрицы-строки или матрицы-столбца);
- **rsort(A,n)** – для сортировки элементов матрицы **A** в порядке возрастания элементов строки **n** этой матрицы;
- **csort(A,n)** – для сортировки элементов матрицы **A** в порядке возрастания элементов столбца **n** этой матрицы;

Для того чтобы ввести функцию сортировки необходимо на панели инструментов **Стандартная** выбрать команду  – **Вставить функцию**. После этого появится запрос со списком всех функций (рисунок 48), которые встроены в **MathCad**. В списке функций, расположенных в алфавитном порядке (опция **Всё**), необходимо выбрать нужную функцию и нажать **Ok**. Далее в функции указываются параметры:

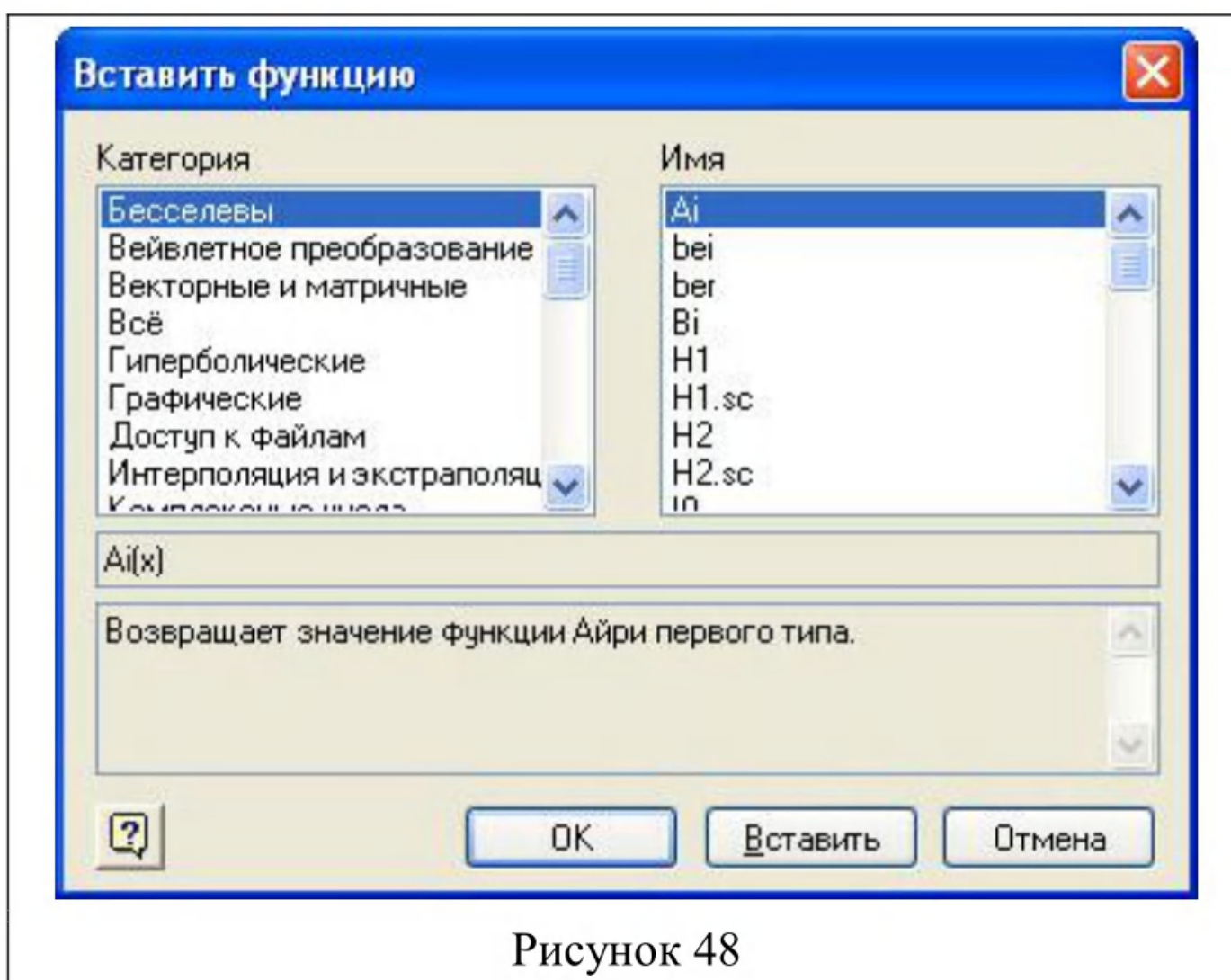
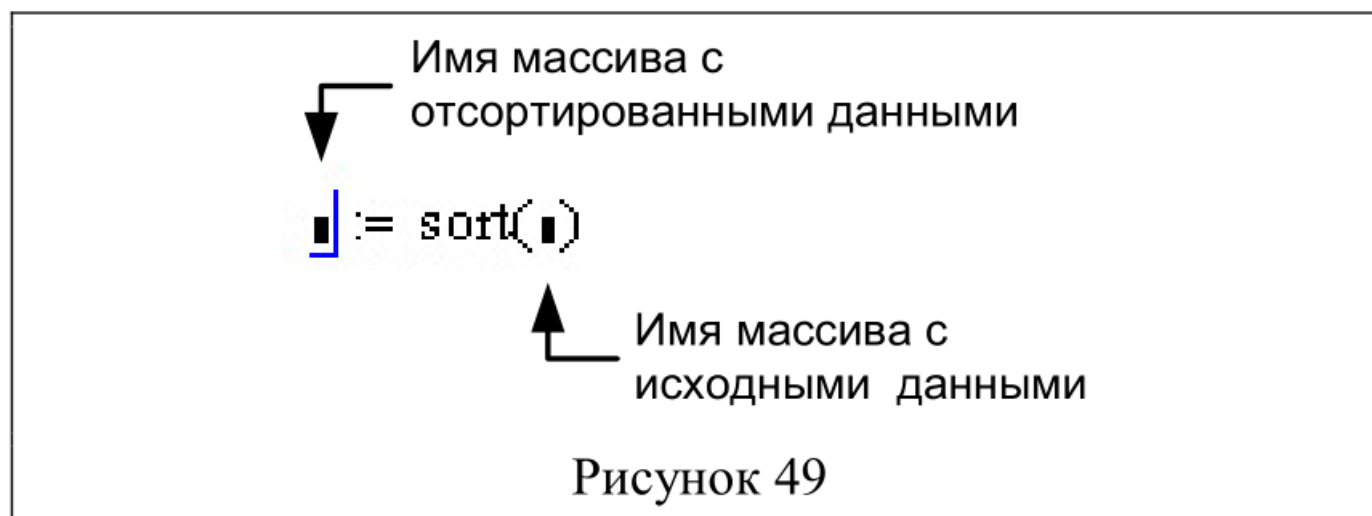


Рисунок 48



- для матрицы-вектора – рисунок 49;
- для двумерной матрицы – рисунок 50.

При сортировке данных в матрице-векторе (матрица-столбец или матрица-строка), полученная матрица будет иметь тот же вид, что и исходная матрица. При сортировке двумерной матрицы полученная матрица меняет элементы в зависимости от расположения сортируемого столбца или строки.

Внимание! Если значение **ORIGIN** не изменено, матрицы будут пронумерованы, начиная с нулевого столбца и нулевой строки. Если необходимо отсортировать матрицу по первому столбцу, то в функцию **csort** необходимо ввести параметры **(A,0)**.

Далее приведены примеры применения функции сортировки:

- для матрицы-строки – на рисунке 51;
- для матрицы-столбца – на рисунке 52;
- для сортировки матрицы по номеру строки – на рисунке 53;
- для сортировки матрицы по номеру столбца – на рисунке 54.

$M := (1 \ 5 \ 3 \ 2 \ 4)$
 $B := \text{sort}(M)$
 $B = [1 \ 2 \ 3 \ 4 \ 5]$

Рисунок 51

$L := \begin{bmatrix} 11 \\ 13 \\ 15 \\ 14 \\ 12 \end{bmatrix}$ $A := \text{sort}(L)$ $A = \begin{bmatrix} 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{bmatrix}$

Рисунок 52

$N := \begin{bmatrix} 1 & 12 & 13 & 17 & 19 \\ 16 & 2 & 14 & 10 & 22 \\ 15 & 11 & 3 & 8 & 24 \\ 21 & 9 & 18 & 4 & 7 \\ 25 & 20 & 23 & 6 & 5 \end{bmatrix}$

$P := \text{rsort}(N, 3)$

$P = \begin{bmatrix} 17 & 19 & 12 & 13 & 1 \\ 10 & 22 & 2 & 14 & 16 \\ 8 & 24 & 11 & 3 & 15 \\ 4 & 7 & 9 & 18 & 21 \\ 6 & 5 & 20 & 23 & 25 \end{bmatrix}$

Рисунок 53

$N := \begin{bmatrix} 1 & 12 & 13 & 17 & 19 \\ 16 & 2 & 14 & 10 & 22 \\ 15 & 11 & 3 & 8 & 24 \\ 21 & 9 & 18 & 4 & 7 \\ 25 & 20 & 23 & 6 & 5 \end{bmatrix}$

$O := \text{csort}(N, 2)$

$O = \begin{bmatrix} 15 & 11 & 3 & 8 & 24 \\ 1 & 12 & 13 & 17 & 19 \\ 16 & 2 & 14 & 10 & 22 \\ 21 & 9 & 18 & 4 & 7 \\ 25 & 20 & 23 & 6 & 5 \end{bmatrix}$

Рисунок 54

3.22 Решение линейной системы уравнений

MathCad можно использовать при решении систем линейных уравнений. Для этого необходимо применить функцию **Isolve**.

Пусть задана система уравнений:

$$\begin{cases} 3x + 2y - 8z + 3 = 0 \\ 4x - 5y + 6z + 8 = 0 \\ -7x + 4y - z + 5 = 0 \end{cases}$$

Для решения заданной системы необходимо ввести две матрицы: матрицу коэффициентов и матрицу правых частей уравнений, где помещены свободные члены уравнений (рисунок 55).

Далее введя функцию **Isolve** с указанием заданных параметров можно определить корни уравнения.

Существует также возможность решить систему линейных уравнений матричным методом (рисунок 56).

3.23 Решение обыкновенных дифференциальных уравнений

Для алгоритма, который используется при решении дифференциальных уравнений, необходимо, чтобы были заданы следующие величины, необходимые для поиска решения:

- начальные условия;
- набор точек, в которых нужно найти решение (интервал и число точек);

$A := \begin{bmatrix} 3 & 2 & -8 \\ 4 & -5 & 6 \\ -7 & 4 & -1 \end{bmatrix}$	$B := \begin{bmatrix} -3 \\ -8 \\ -5 \end{bmatrix}$	$\text{Isolve}(A, B) = \begin{bmatrix} 23 \\ 44 \\ 20 \end{bmatrix}$
Рисунок 55		

$$A := \begin{bmatrix} 3 & 2 & -8 \\ 4 & -5 & 6 \\ -7 & 4 & -1 \end{bmatrix} \quad B := \begin{bmatrix} -3 \\ -8 \\ -5 \end{bmatrix}$$

$$X := A^{-1} \cdot B \quad X = \begin{bmatrix} 23 \\ 44 \\ 20 \end{bmatrix}$$

Проверка :

$$B1 := A \cdot X \quad B1 = \begin{bmatrix} -3 \\ -8 \\ -5 \end{bmatrix}$$

Рисунок 56

- само дифференциальное уравнение, записанное в виде равенства: в левой части – первая производная, а в правой – члены уравнения, не содержащие производных.

3.24 Решение обыкновенного дифференциального уравнения первого порядка методом Рунге-Кутты

Пусть задано обыкновенное дифференциальное уравнение первого порядка:

$$\frac{dy}{dx} + 3 \cdot y = 0,$$

представим его в виде

$$\frac{dy}{dx} = -3 \cdot y,$$

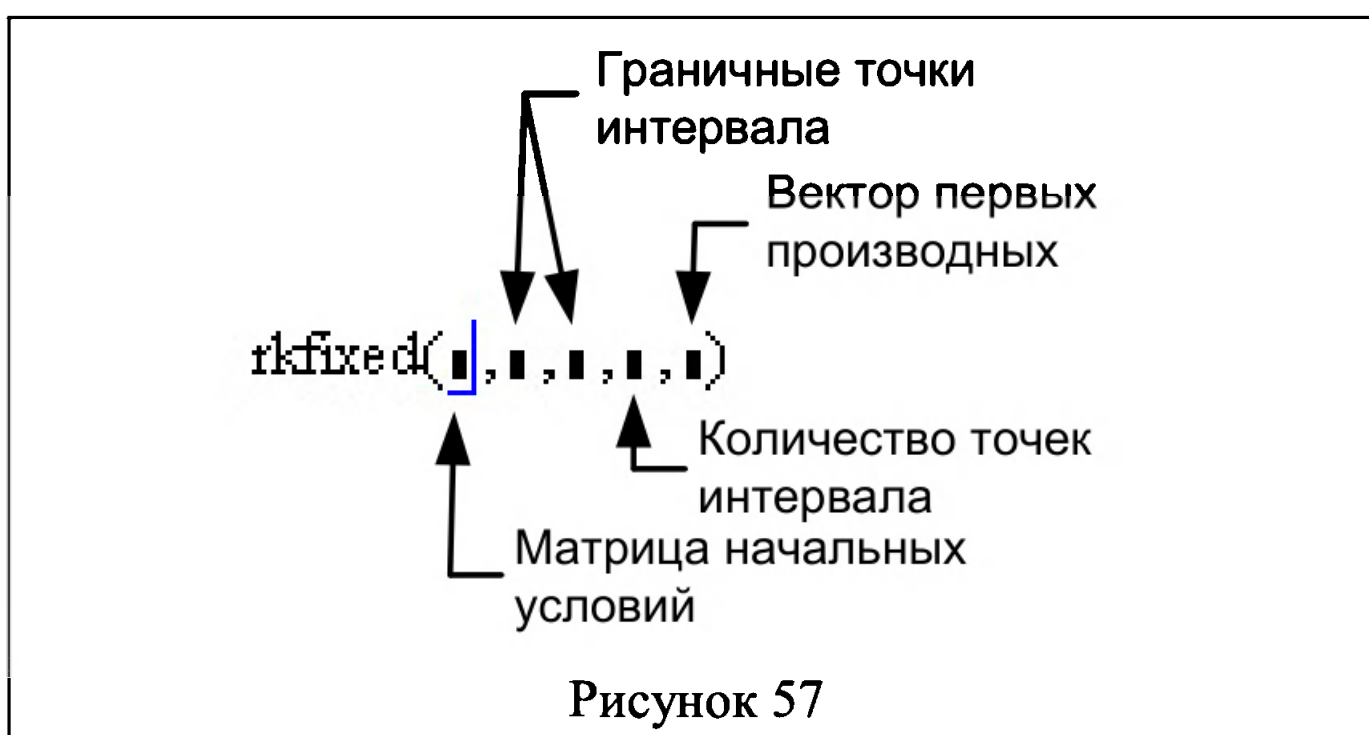
с начальными условиями:

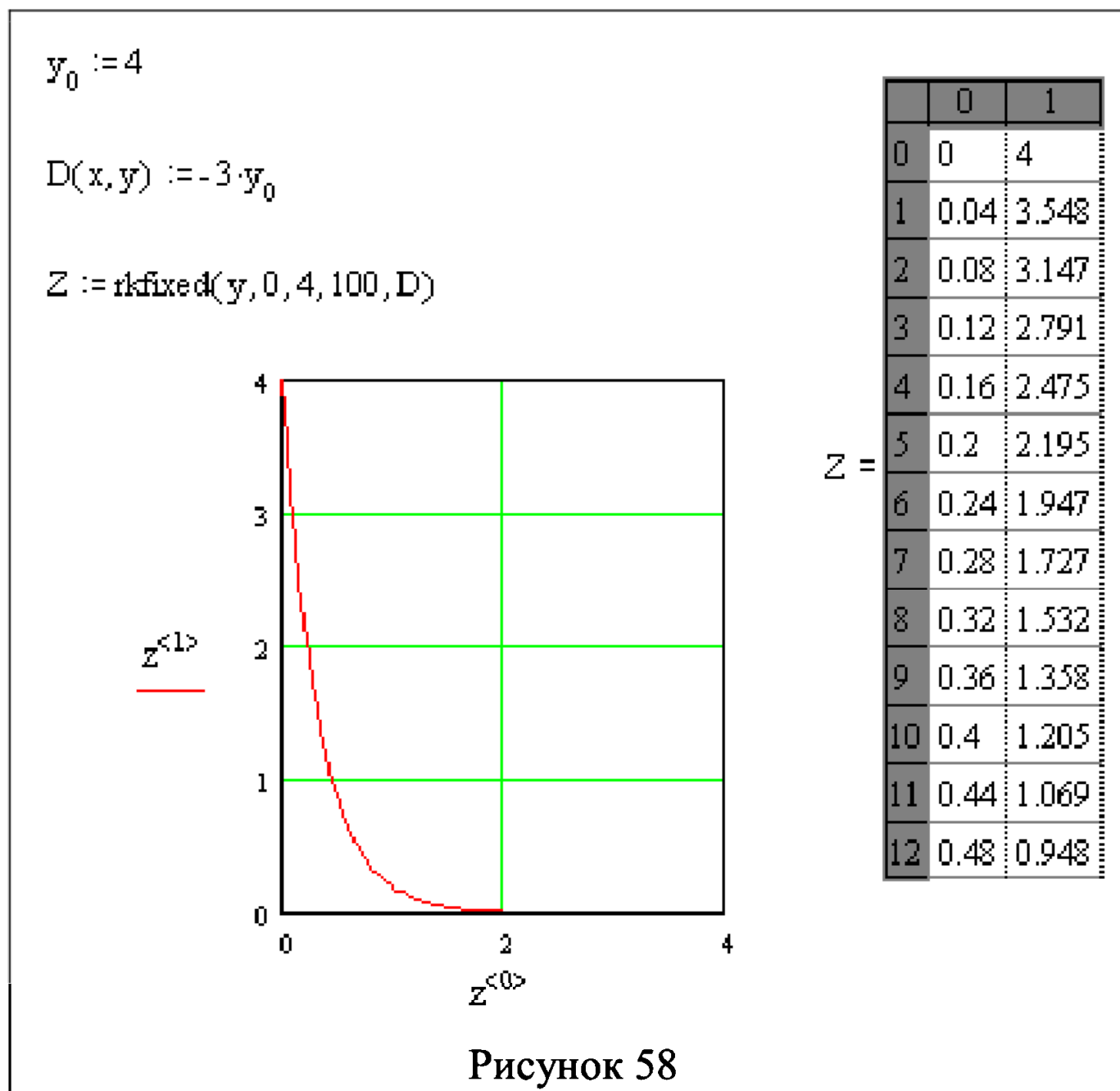
$$y(0) = 4.$$

Используем для решения метод Рунге-Кутты 4-го порядка. Для этого используется встроенная в **MathCad** функция **rkfixed**, с указанием параметров (рисунок 57).

В результате решения получается матрица, состоящая из двух столбцов: первый – значения независимой переменной для которых ищется решение дифференциального уравнения; второй – значения, найденного решения при соответствующих значениях аргумента.

На рисунке 58 представлено решение дифференциального уравнения в **MathCad**.





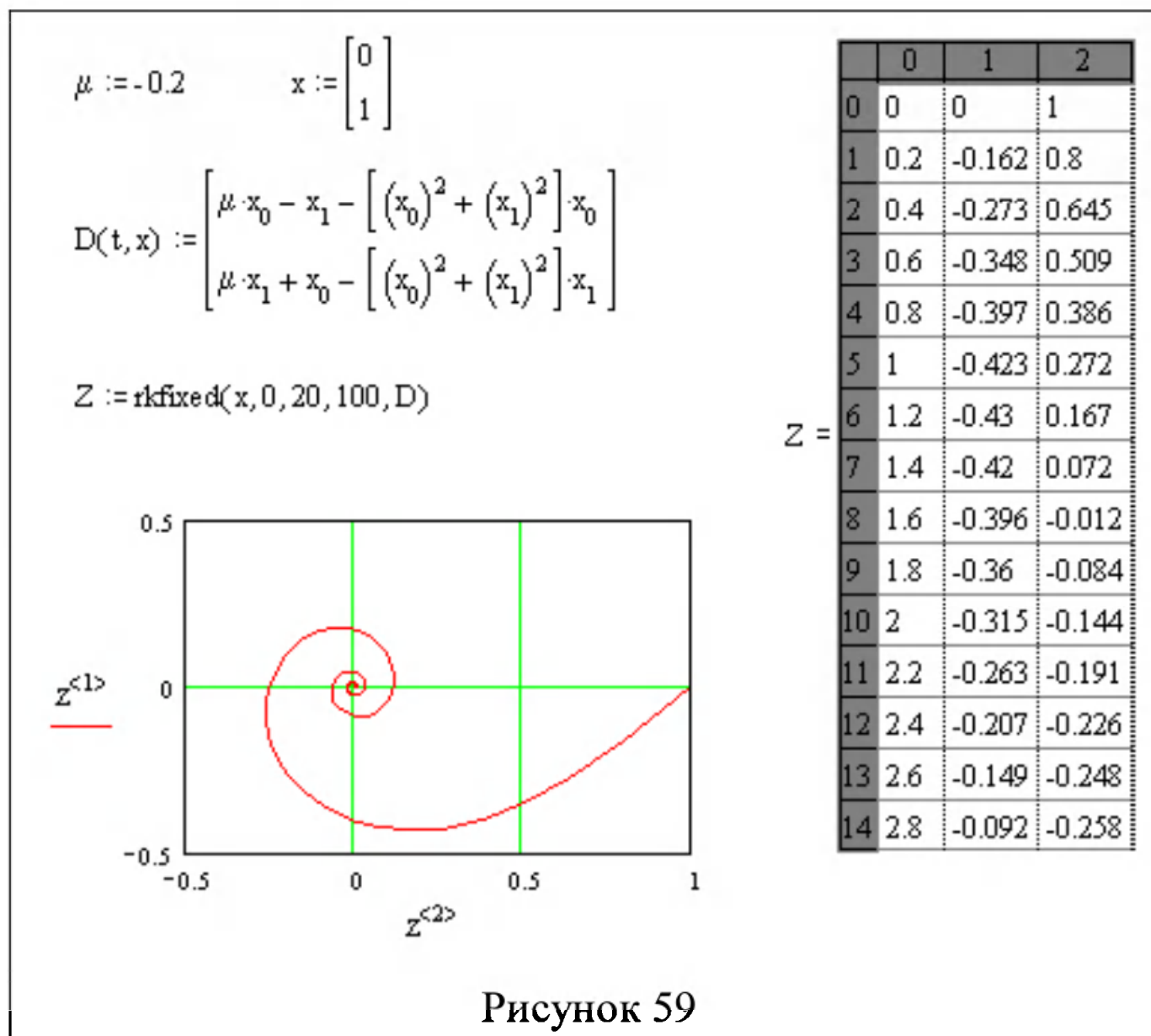
3.25 Решение системы обыкновенных дифференциальных уравнений первого порядка

Пусть задана система дифференциальных уравнений первого порядка:

$$\begin{aligned}
 x_0'(t) &= \mu \cdot x_0(t) - x_1(t) - \left(x_0(t)^2 + x_1(t)^2 \right) \cdot x_0(t) \\
 x_1'(t) &= \mu \cdot x_1(t) + x_0(t) - \left(x_0(t)^2 + x_1(t)^2 \right) \cdot x_1(t)
 \end{aligned}$$

с начальными условиями:

$$x_0(0) = 0 \quad \text{и} \quad x_1(0) = 1.$$



Для того чтобы решить систему дифференциальных уравнений первого порядка, необходимо:

- определить вектор, содержащий начальные значения для каждой неизвестной функции;
- определить функцию, возвращающую значение в виде вектора из n элементов, которые содержат первые производные каждой из неизвестных функций;
- выбрать точки, в которых нужно найти приближённое решение;
- передать эту информацию в функцию **rkfixed**.

Решение данной системы дифференциальных уравнений в **MathCad** будет иметь вид представленный на рисунке 59.

3.26 Работа с файлами данных

MathCad может считывать файлы с данными, записанными: регистрирующей аппаратурой, выводом данных из электронных таблиц, в столбец данных; набранные в текстовом редакторе, экспортированные из базы данных.

Данные в файле **MathCad** могут иметь любые форматы записи:

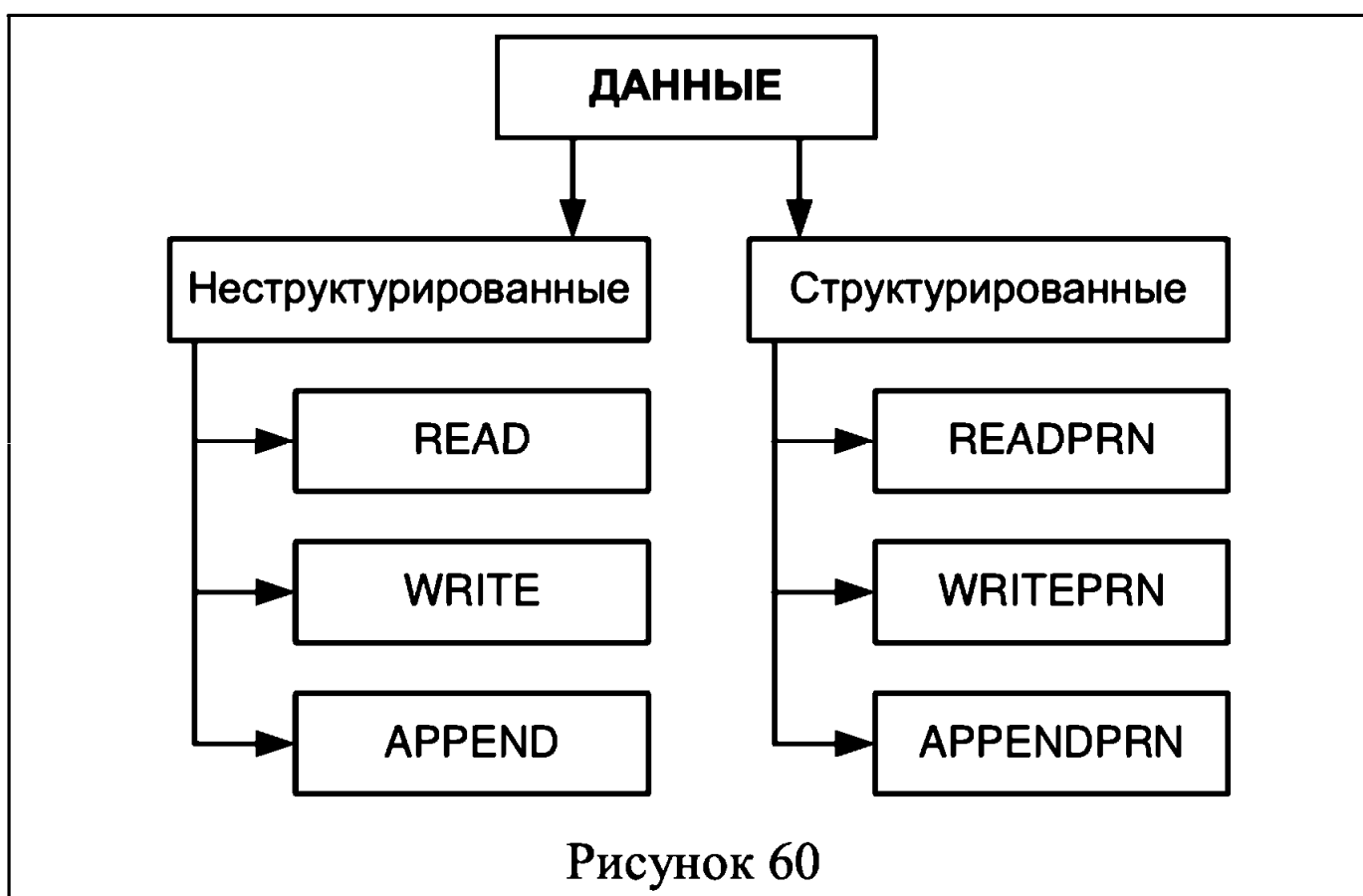
- целое число: **3, 2, -5**;
- число с плавающей запятой: **2.54, 3.1415**;
- число в экспоненциальной форме: **4.51e-5, 34.1e5**.

В **MathCad** существует шесть функций для работы с файлами данных (рисунок 60).

Основные из них – это:

- **READ** (“*имя файла.dat*”) – считывает значение из файла данных. Применяется для записи скалярных величин.

$V_i := \text{READ}(\text{"fp"})$



- **WRITE** (“*имя файла.dat*”) – записывает значения в файл данных. Действителен при скалярной величине.

$$\text{WRITE}(\text{"fp"}) := F(n, m)$$

- **READPRN** (“*имя файла.dat*”) – считывает структурированный файл данных. Возвращает матрицу. Каждая строка файла данных становится строкой матрицы. Обязательное условие – данных в строках должно быть одинаковое количество.

$$A_{i,j} := \text{READPRN}(\text{"matr.dat"})$$

- **WRITEPRN** (“*имя файла.dat*”) – записывает матрицу в файл данных. Каждая строка матрицы становится строкой в файле.

$$\text{WRITEPRN}(\text{"matr.dat"}) := M$$

Примерами записи этих операторов в **MathCad** могут служить следующие случаи:

- запись значений функции в файл – рисунок 61;
- считывание значений функции из файла – рисунок 62;
- запись матрицы в файл данных – рисунок 63;
- считывание матрицы из файла данных – рисунок 64.

3.27 Работа с комплексными числами

Комплексное число в математике представляется в виде:

- алгебраической формы: $z = \text{Re}(z) + \text{Im}(z)$;
- показательной формы: $z = r \cdot e^{j\Theta}$, где Θ – фаза; r – модуль комплексного числа.

$$x := 0, \frac{3 \pi}{100} \dots 3 \pi$$

$$y(x) := \sin(x)$$

$$\text{WRITE}(\text{"sin.dat"}) := y(x)$$

$$y(x)$$

0
0.094
0.187
0.279
0.368
0.454
0.536
0.613

Рисунок 61

$$x := 0, \frac{3 \pi}{100} \dots 3 \pi$$

$$z(x) := \text{READ}(\text{"sin.dat"})$$

$$z(x)$$

0
0.094
0.187
0.279
0.368
0.454
0.536
0.613

Рисунок 62

$$M := \begin{bmatrix} 3 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

WRITEPRN("matr.dat") := M

Рисунок 63

A := READPRN("matr.dat")

$$A = \begin{bmatrix} 3 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Рисунок 64

В **MathCad** комплексные числа заносятся в формате:

$$z = a + bi,$$

где a и b – обычные числа.

Вместо i в мнимом числе можно использовать j . При вводе комплексных чисел нельзя использовать i или j сами по себе. Нужно всегда писать $1i$ или $1j$, в противном случае **MathCad** воспринимает i или j как переменную. При вводе $1i$ или $1j$ **MathCad** скрывает ненужную 1.

В **MathCad** есть следующие специальные функции и операторы для работы с комплексными числами:

- $\text{Re}(z)$ – вещественная часть;
- $\text{Im}(z)$ – мнимая часть;
- $\text{arg}(z)$ – угол в радианах между $-\pi$ и π ;
- $|z|$ – модуль;
- \bar{z} – число, комплексно сопряжённое к z (для получения сопряжения необходимо набрать имя переменной комплексного числа и затем нажать кавычку “”).

Далее приведены примеры записи и некоторых действий с комплексными числами:

- задание характеристик комплексного числа – рисунок 65;
- определение комплексных переменных – рисунок 66;

– примеры действий с комплексными числами –
 рисунок 67.

$r := 2$	$\theta := \frac{3\pi}{4}$	$z1 := \sqrt{-1}$	$z2 := r \cdot e^{i \cdot \theta}$
		$z1 = i$	$z2 = -1.414 + 1.414i$
Рисунок 65		Рисунок 66	

$z1 + z2 = -1.414 + 2.414i$	$\text{Re}(z2) = -1.414$
$z1 \cdot z2 = -1.414 - 1.414i$	$\text{Im}(z2) = 1.414$
$\overline{z2} = -1.414 - 1.414i$	$\frac{z2}{z1} = 1.414 + 1.414i$
$ z2 = 2$	$\arg(z2) = 2.356$
Рисунок 67	

3.28 Пример расчёта электрической цепи при синусоидальном напряжении

Пример иллюстрирует применение матричного метода решения алгебраических уравнений с комплексными коэффициентами.

Результаты расчётов представляются в виде векторов синусоидальных токов, протекающих по ветвям с резисторами R_1 , R_2 , R_3 . Вектора токов построены в полярной системе координат.

Дана электрическая цепь, состоящая из резисторов R_1 , R_2 , R_3 , ёмкости C и индуктивности L , и включённая на генератор синусоидальной эдс E (рисунок 68). Подобрать величину резистора R_2 с целью осуществления работы цепи в режиме резонанса токов. Для этого в примере применён оператор глобального присваивания.

В такой электрической цепи при определённом соотношении между L и C возникает резонанс токов, характеризуемый нулевым сдвигом фаз между током i_3 и питающим напряжением.

Ток i_1 протекает в цепи с резистором R_1 , i_2 – в цепи с резистором R_2 , i_3 – в цепи с резистором R_3 .

Решение задачи в программе **MathCad** будет иметь следующий вид.

Исходные данные:

$$R_1 := 30 \quad R_3 := 70 \quad R_2 := 100$$

$$E := 220 \quad L := 0.09$$

$$f := 50 \quad \omega := 2 \cdot \pi \cdot f$$

Из условия резонанса токов при заданной индуктивности определим величину ёмкости по формуле

$$C := \frac{1}{(2 \cdot \pi \cdot f)^2 \cdot L} \quad C = 1.126 \times 10^{-4}$$

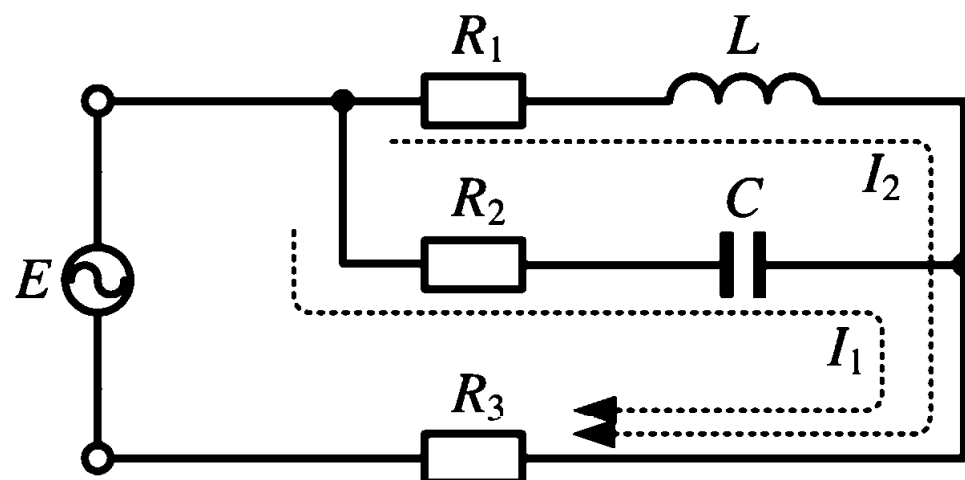


Рисунок 68

Комплексные сопротивления в ветвях электрической цепи:

$$Z_1 := R_1 + i \cdot \omega \cdot L \qquad Z_2 := R_2 - i \cdot \frac{1}{\omega \cdot C}$$

Обозначим контурные токи через I_1 и I_2 .

На основании второго закона Кирхгофа запишем уравнения равенства напряжений:

$$Z_1 \cdot I_2 + R_3 \cdot (I_1 + I_2) = E$$

$$Z_2 \cdot I_1 + R_3 \cdot (I_1 + I_2) = E$$

В матричной форме уравнения имеют вид:

$$\mathbf{A} \cdot \{\mathbf{I}\} = \{\mathbf{E}\}$$

где $\{\mathbf{I}\}$ - вектор контурных токов;

$\{\mathbf{E}\}$ - вектор эдс;

\mathbf{A} - матрица комплексных сопротивлений.

$$\mathbf{A} := \begin{pmatrix} R_3 & Z_1 + R_3 \\ Z_2 + R_3 & R_3 \end{pmatrix} \qquad \mathbf{E} := \begin{pmatrix} 220 \\ 220 \end{pmatrix}$$

Решение системы уравнений в матричной форме имеет вид:

Вектор контурных токов имеет вид:

$$\mathbf{I} := \mathbf{A}^{-1} \cdot \mathbf{E} \qquad \mathbf{I} = \begin{pmatrix} 0.572 + 0.394i \\ 1.594 - 0.727i \end{pmatrix}$$

Здесь \mathbf{A}^{-1} - обратная матрица комплексных сопротивлений.

Токи, протекающие по ветвям с резисторами R_1 , R_2 и R_3 , их модули и фазовые углы равны:

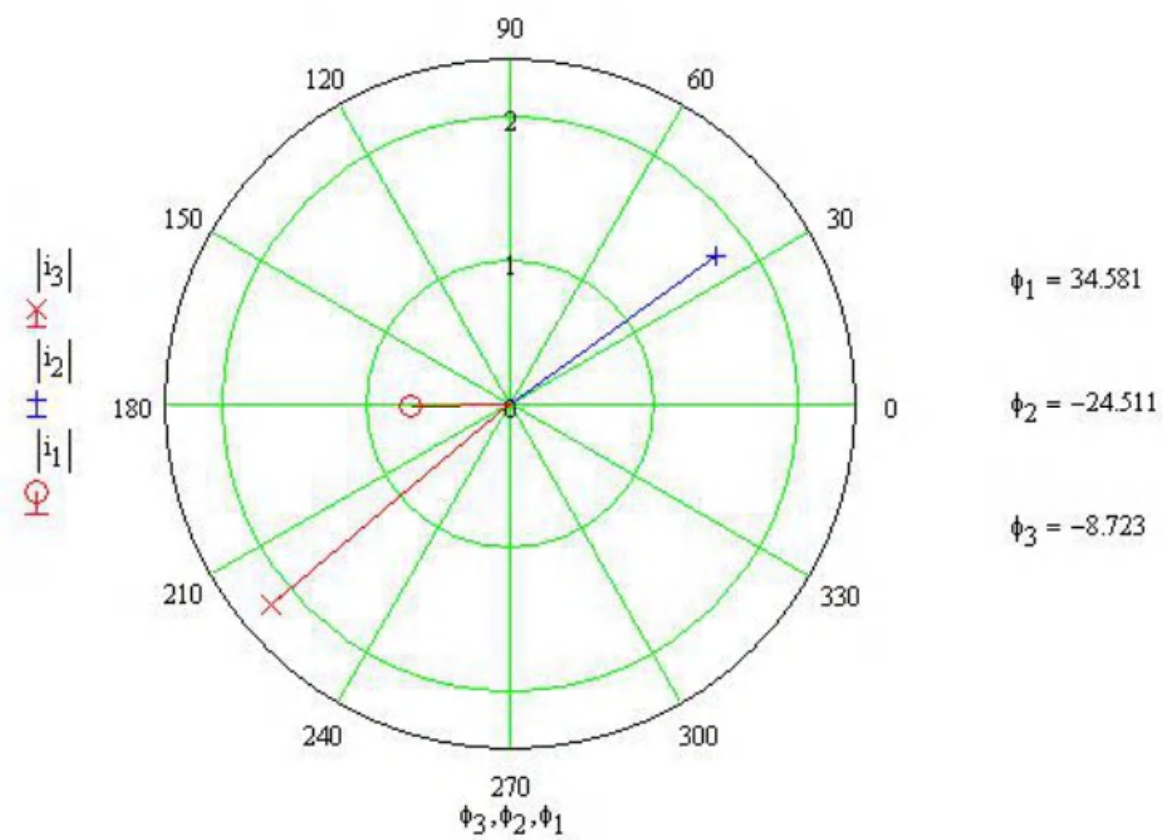
$$i_1 := I_{0,0} \qquad |i_1| = 0.695 \qquad \phi_1 := \frac{\arg(i_1) \cdot 180}{\pi}$$

$$i_2 := I_{1,0} \qquad |i_2| = 1.752 \qquad \phi_2 := \frac{\arg(i_2) \cdot 180}{\pi}$$

$$i_3 := I_{0,0} + I_{1,0} \qquad i_3 = 2.166 - 0.332i \qquad |i_3| = 2.192 \qquad \phi_3 := \frac{\arg(i_3) \cdot 180}{\pi}$$

Изменяя величину резистора R_2 можно добиться резонанса токов, который возникает при значении фазового угла $\phi_3=0$.

Результаты изменения величины R_2 можно наблюдать на векторной диаграмме, построенной в полярной системе координат.



Св.план 2018 г., поз.99

**Рыбников Евгений Константинович,
Сердобинцев Евгений Васильевич,
Володин Сергей Вячеславович**

**ПРОГРАММИРОВАНИЕ В СРЕДЕ TURBO BASIC
И ПРОГРАММНОМ ПАКЕТЕ MATHCAD**

УЧЕБНОЕ ПОСОБИЕ
по дисциплине «Информатика»
для студентов I курса
специальности
23.05.03 «Подвижной состав железных дорог»

Тираж 100 экз.

Москва, Издательский центр Onebook