Министерство образования Российской Федерации

Томский политехнический университет

А.В. Кравцов, О.Е. Мойзес, Е.А. Кузьменко,

Д.А. Баженов, П.И. Коваль

Информатика

и вычислительная математика

УЧЕБНОЕ ПОСОБИЕ ДЛЯ СТУДЕНТОВ ХИМИЧЕСКИХ СПЕЦИАЛЬНОСТЕЙ ТЕХНИЧЕСКИХ ВУЗОВ

Допущено Министерством образования Российской Федерации в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлениям подготовки дипломированных специалистов Химическая технология неорганических веществ и материалов, химическая технология органических веществ и топлива, биотехнология

Томск 2003

УДК 519.682 (075.8)

Кравцов А.В., Мойзес О.Е., Кузьменко Е.А., Баженов Д.А., Коваль П.И., Информатика и вычислительная математика: Учебное пособие для студентов химических специальностей технических вузов. - Томск: Изд. ТПУ, 2003 - 263 с.

В учебном пособии рассмотрены вопросы архитектуры ЭВМ, особенности работы в системах MS DOS, WINDOWS. Приводятся основы программирования на алгоритмическом языке Паскаль. Пособие содержит основные сведения о численных методах. Основные приемы и методы программирования рассмотрены на конкретных химических задачах. Приведены типовые алгоритмы для решения задач различных классов. Излагаемый материал сопровождается большим количеством примеров вычислений и обработки опытных данных. Учебное пособие подготовлено на кафедре химической технологии топлива и химической кибернетики ТПУ и предназначено для студентов химических специальностей технических вузов.

Рецензенты:

Л.С.Гордеев- доктор технических наук, профессор, зав.кафедрой кибернетики ХТП РХТУ им.Менделеева, член-корреспондент международной инженерной академии.

А.И.Рубан- доктор технических наук, профессор, зав. кафедрой информатики Красноярского государственного технического университета, академик Международной АН высшей школы,

А.С.Носков- доктор технических наук, профессор, зам.директора института катализа Сибирского отделения РАН, зав. кафедрой инженерных проблем экологии Новосибирского государственного университета.

Темплан 2003

СОДЕРЖАНИЕ

Предисловие

Конец XX и начало XI века ознаменовались широким внедрением и практическим использованием в химической науке и технологической практике метода математического моделирования. При этом решаются проблемы повышения эффективности и совершенствования технологии химического производства с использованием компьютерных моделирующих систем, создание которых требует, с одной стороны, глубокого понимания и адекватного количественного описания физико-химических закономерностей протекающих процессов, а с другой –знания вычислительных методов и основ программирования с применением ЭВМ и компьютерных технологий.

Настоящее учебное пособие удачно объединяет эти два направления. В нем помещены разделы, рассматривающие сведения об аппаратурных и программных средствах вычислительной техники, алгоритмы и основы программирования, а также численные методы решения прикладных задач.

В пособии нашли отражение основные теоретические и практические направления, что дает возможность получить разносторонние знания при изучении дисциплины «Информатика».

В целом пособие состоит из пяти глав.

В 1,2 главах рассмотрены принципы представления информации, описание основных и периферийных устройств современных компьютеров, приведены сведения об операционных системах MS DOS и WINDOWS. Изложение материала построено таким образом, что обучающиеся могут самостоятельно освоить работу на персональном компьютере в наиболее распространенных программных средах.

В главах 3,4 авторами подробно излагаются особенности работы в интегрированной среде Турбо Паскаль и необходимые действия пользователя при построении и выполнении его первой программы, что является, несомненно, важным для приобретения практических навыков.

Достоинством пособия является также то, что теоретический материал по основам программирования на алгоритмическом языке Турбо Паскаль сопровождается конкретными химическими примерами. Это делает осваиваемые студентами алгоритмы более доступными и понятными.

Излагаемый в пятой главе теоретический материал по численным методам решения также иллюстрируется большим количеством примеров алгоритмов для решения химико-технологических задач. Даны примеры работающих программ для решения прикладных задач, встречающихся в деятельности будущего специалиста.

Представляемое учебное пособие ориентировано, прежде всего, на студентов химических специальностей вузов. Сведения и навыки, полученные в ходе изучения разделов данного пособия, успешно могут быть использованы и при изучении других дисциплин, которые требуют использования вычислительной техники и основ программирования, при обработке результатов химического эксперимента и математическом моделировании химико-технологических процессов.

В настоящее время учебной литературы, которая бы комплексно рассматривала вопросы изучения программно-технических средств, программирования и практического применения численных методов для решения прикладных задач химического профиля практически не существует, поэтому предлагаемое пособие в какой-то степени восполняет этот пробел и является необходимым при изучении дисциплины «Информатика» студентами химиками.

1.ЧТО ТАКОЕ ИНФОРМАТИКА

Термин «информатика» возник в 60-е годы во Франции и образован слиянием двух слов: information (информация) и automatique (автоматика). Это наука об автоматической обработке информации.

Информатика – это наука, которая изучает общие законы, методы накопления, передачи и обработки информации с помощью ЭВМ [1,2]. Информатика появилась благодаря развитию компьютерной техники, базируется на ней и не мыслима без нее.

Под информацией понимаются любые сведения об объективно существующих объектах и процессах, их связях и взаимодействии, доступные для практического использования в деятельности людей [2].

Во всех областях человеческой деятельности приходится обрабатывать информацию. Однако человек сам не может обработать огромное количество информации, поэтому на помощь ему приходят ЭВМ.

ЭВМ – электронное устройство для автоматической обработки информации.

В информатике в узком смысле можно выделить три части [1–3]: технические средства, программные средства, алгоритмы и теоретические методы решения задач на ЭВМ.

1.1. ЗНАКОМСТВО С ВЫЧИСЛИТЕЛЬНОЙ МАШИНОЙ

В настоящее время существует огромное разнообразие электронных вычислительных машин (ЭВМ) различных типов и марок. ЭВМ представляет собой устройство (точнее, совокупность взаимосвязанных устройств), которое способно выполнять определенный набор элементарных арифметических и логических операций. Выполнив одну операцию, ЭВМ автоматически переходит к выполнению следующей и, таким образом, может выполнить длинные цепочки операций без вмешательства человека.

В составе вычислительной машины различают аппаратуру и программное обеспечение. К последнему относят совокупность программ, определяющих функционирование аппаратуры [1 – 3].

Общий вид персонального компьютера представлен на рис.1. Принципиальная функциональная схема вычислительной машины приведена на рис.2. Назначение основных, а также периферийных устройств компьютера будут далее рассмотрены.

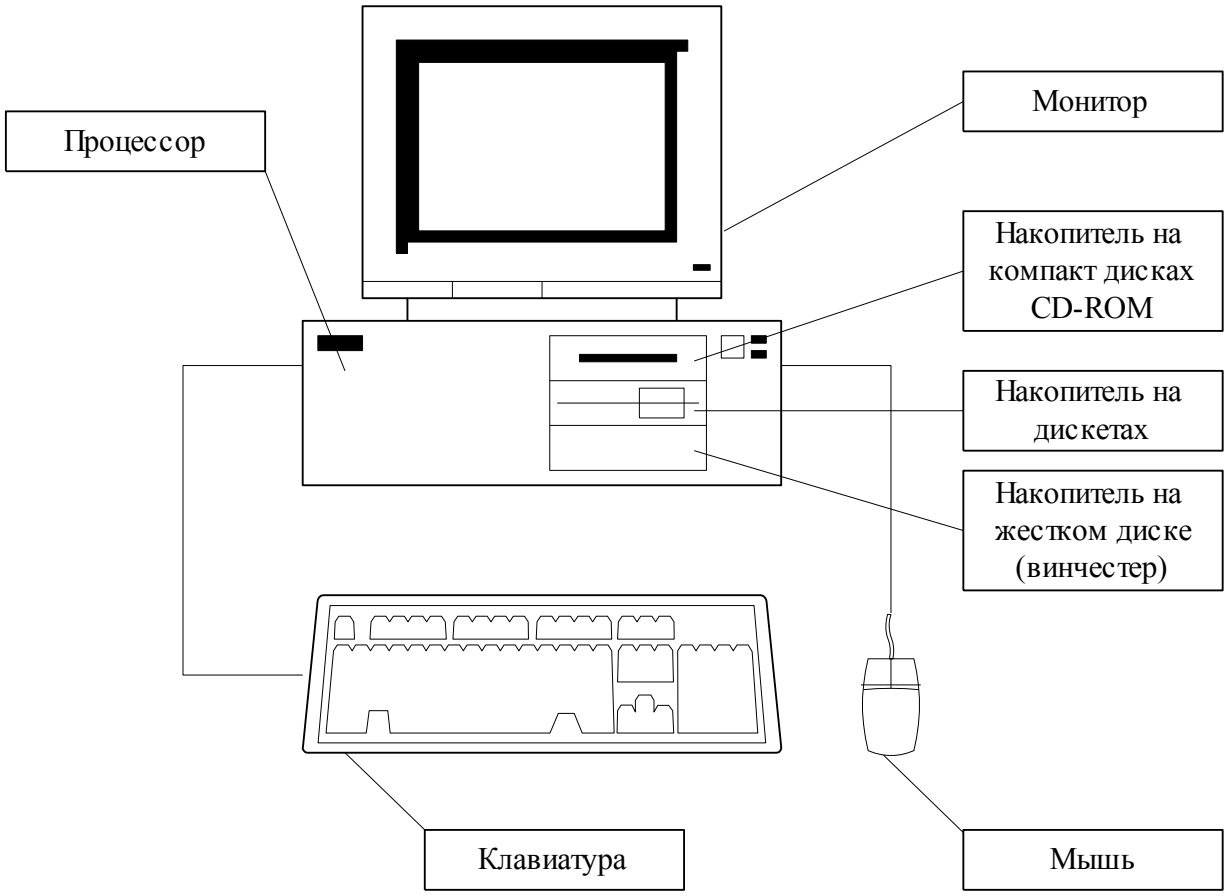


Рис. 1. Общий вид персонального компьютера

1.2. Представление информации в эвм

ЭВМ обрабатывает информацию только в закодированном виде. Информация называется закодированной, если любая ее элементарная часть представлена в виде числа. Такие числа называются кодами. Вся информация, циркулирующая в центральном устройстве, имеет двоичное представление, т.е. записана в виде последовательности из 0 и 1, что вызывает необходимость кодирования и декодирования информации. Запись символов в двоичной системе более громоздка, чем в десятичной, но более приспособлена к использованию в компьютерах, в которых имеется два состояния: наличие сигнала (1) и отсутствие сигнала (0).

Рис. 2. Принципиальная схема ЭВМ

Пример записи некоторых чисел в двоичном виде:

0 – 0	4 – 100	8 – 1000
1 – 1	5 – 101	9 – 1001
2 – 10	6 – 110	10 – 1010
3 – 11	7 – 111	11 – 1011

Минимальная единица информации – ***бит***(***bit***). Один бит информации – это одна двоичная цифра: 0 или 1. Это очень маленькое количество информации, поэтому в компьютерах для обработки информации используется более крупная единица – ***байт***(***byte***)**. 1 байт = 8 битам.**

Байт – это также элементарная ячейка памяти ЭВМ. Каждая ячейка имеет адрес (номер ячейки) и содержимое (двоичный код), которое хранится в ней.

Для измерения памяти ЭВМ используются также килобайты и мегабайты:

1 Кбайт = 1024 байтам;

1 Мбайт = 1024 Кбайтам.

При обработке информации процессор находит по адресу нужную ячейку, читает из нее содержимое, выполняет необходимые действия и записывает результат в другую ячейку памяти.

1.3. ЭТАПЫ РЕШЕНИЯ ЗАДАЧ НА ЭВМ

Процедуру подготовки и решения задач на ЭВМ можно представить в виде следующих этапов:

1. Постановка задачи. Задача формулируется пользователем или выдается ему в виде специального задания. На данном этапе осуществляется выбор общего подхода к решению задачи, определение конечной цели.

2. Математическая формулировка задачи, т.е. представление ее в виде уравнений, соотношений и т.д.

3. Выбор метода решения, либо процедуры, позволяющих свести решаемую задачу к последовательности элементарных действий.

4. Разработка алгоритма решения задачи.

5. Написание программы на языке программирования. На данном этапе происходит перевод разработанного алгоритма на язык программирования.

6. Ввод программы и исходных данных в ЭВМ.

7. Отладка программы. На этом этапе производится обнаружение с помощью ЭВМ ошибок в программе и их исправление.

8. Решение задачи на ЭВМ. Обработка и интерпретация результатов расчета в соответствии с поставленной задачей.

2. ПРОГРАММНО–ТЕХНИЧЕСКИЕ РЕДСТВА НФОРМАТИКИ

2.1. Назначение основных и периферийных устройств компьютера

Персональный компьютер включает следующие основные устройства[4]:

– процессор, выполняющий управление компьютером, вычисление и т.д.;

– клавиатура, позволяющая вводить символы в компьютер;

– монитор (дисплей) для изображения текстовой и графической информации;

– накопители (дисководы) для гибких магнитных дисков, используемые для чтения и записи на гибкие магнитные диски (дискеты);

– накопитель на жестком магнитном диске, предназначенный для чтения и записи на несъемный магнитный диск (винчестер).

Для передачи информации между системным блоком и внешними устройствами используется магистраль.

Рассмотрим более подробно назначение, структуру и разновидности основных устройств компьютера.

2.1.1. Процессор

Процессор персонального компьютера IBM PC содержит следующие элементы:

– Основной микропроцессор, управляющий работой компьютера и выполняющий все вычисления. Определяет быстродействие компьютера. Наряду с основным микропроцессором большинство современных компьютеров содержат математический сопроцессор, предназначенный для выполнения операций с числами с плавающей запятой. При этом скорость расчетов возрастает в пять и более раз.

– Оперативная память, в которой располагаются программы, выполняемые компьютером, и используемые программами данные.

– Электронные схемы (контроллеры), управляющие работой различных устройств, входящих в компьютер.

– Порты ввода–вывода, через которые процессор обменивается данными с внешними устройствами.

Оперативная память. Оперативная память компьютера разделяется на несколько типов:

– conventional;
– upper;
– extended;
– high.

Conventional – базовая память**.** Под базовой памятью понимают первые 640 Кбайт оперативной памяти компьютера. В младшие ее адреса загружается операционная система и драйверы устройств. Этот размер памяти называется lower. Оставшуюся свободную часть занимают пользовательские программы.

Upper. Это область памяти между 640 Кбайт и 1 Мбайт. Часть этой памяти используется для загрузки видеоадаптера. При использовании специальных программ эта часть памяти становится доступной для загрузки пользовательских программ.

Extended. Раздел памяти свыше 1 Мбайт принято называть extended. С помощью специальных программ пользователь может использовать эту память для создания временного логического диска, как буфер для ускорения обмена с жестким диском и т.д.

High. Так называют первые 64 Кбайта extended памяти.

Expended. Для использования памяти свыше 640 Кбайт в прикладных программах фирмы Lotus, Intel, Microsoft разработали стандарт LIM EMS (Lotus, Intel, Microsoft Expanded Memory Specification), позволяющий адресовать до 32 Мбайт оперативной памяти.

Для достаточно быстрых компьютеров необходимо обеспечить быстрый доступ к оперативной памяти, иначе микропроцессор будет простаивать и быстродействие компьютера уменьшится. Для этого такие компьютеры могут оснащаться **кэш–памятью**, т.е. «сверхоперативной» памятью относительно небольшого объема, в которой хранятся наиболее часто используемые участки оперативной памяти. Кэш–память располагается «между» микропроцессором и оперативной памятью, и при обращении микропроцессора к памяти сначала производится поиск нужных данных в кэш–памяти.

2.1.2. МАГНИТНЫЕ НОСИТЕЛИ И НАКОПИТЕЛИ

Основным носителем информации IBM PC – совместимого компьютера является магнитный диск. Различают жесткий магнитный диск (винчестер) и гибкий магнитный диск (дискета или флоппи–диск) [1,4-6].

Накопители на жестком диске предназначены для постоянного хранения информации, используемой при работе с компьютером: программ операционной системы, часто используемых пакетов программ, редакторов документов, трансляторов с языков программирования и т.д.

Гибкие диски позволяют переносить программы и документы с одного компьютера на другой, хранить информацию, не используемую постоянно на компьютере, делать архивные копии информации, содержащейся на жестком диске. Существует два типа дискет: 5,25 и 3,5 дюйма (рис.3,4). Для чтения–записи гибких дисков служат внешние магнитные накопители, называемые дисководами (рис.5,6).

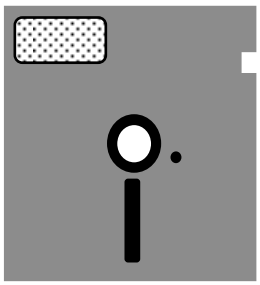


Рис. 3. Дискета размером 5,25 дюйма

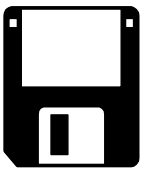


Рис. 4. Дискета размером 3,5 дюйма

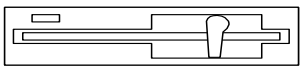


Рис. 5. Дисковод для дискет размером 5,25 дюйма

Рис. 6. Дисковод для дискет

Магнитные ленты и накопители на них. В настоящее время используются только для архивного хранения больших объемов информации. Наиболее массовый вид устройств для этой цели – кассетные накопители, так называемые стримеры. Стримеры могут подключаться к контроллеру дисковода для гибких дисков и позволяют сохранять на ленте 250, 510 или 680 Мбайт данных.

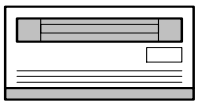


Рис. 7. Накопитель на CD – ROM

Оптические (лазерные) компакт-диски (CD) и дисководы. CD-ROM-Compakt Disk Read Only Memory. Наиболее распространены диски «только для чтения» – CD–ROM и дисководы для них. В этих устройствах применяется технология цифровой записи, для чтения и записи данных используют лазерный луч. Благодаря этому CD обладают достаточно большой скоростью чтения данных (150,300,600 и т.д. Кбайт/с) и чрезвычайно высокой емкостью: от 127 до 650 Мбайт (рис.7).

Bernoulli. Накопители на гибких магнитных носителях кассетного типа большой емкости (до 230 Мбайт). Их работа основана на аэродинамическом эффекте («эффекте Бернулли») между головкой

чтения–записи и магнитной поверхностью. Bernoulli–накопители являются надежными, быстродействующими и достаточно дорогими устройствами.

Магнитооптические диски и дисководы. Основаны на записи данных при помощи дополнительного магнитного поля (поля смещения) и луча лазера. Емкость магнитооптических дисков достигает сотен и тысяч мегабайт.

2.1.3. ВИДЕОСИСТЕМА

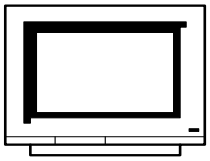


Рис. 8. Дисплей

Видеосистема считается стандартным устройством вывода IBM PC. Видеосистема состоит из ***дисплея*** и ***видеоконтроллера*** (видеоадаптера) [6]. Дисплей (рис.8) является достаточно простым устройством на основе электронно–лучевой трубки и во всем подчиняется командам, поступающим с платы видеоадаптера. Изображение, создаваемое видеоадаптером, может быть текстовым и графическим, поэтому различают текстовой и графический режимы работы. В графическом режиме работы на экран выводится любое изображение, состоящее из точек, в текстовом режиме выводятся только символы ASCII–кодировки (256 заранее заданных символов) – зато с более высокой скоростью. Текущее место вывода на экран в текстовом режиме всегда отмечается курсором – мерцающим значком, похожим на символ подчеркивания. Видеосистема компьютера характеризуется целым рядом показателей: разрешающей способностью, палитрой, числом видеорежимов и видеопамятью. В зависимости от этих показателей различают следующие видеоадаптеры: MDA (Monochrome Display Adapter), CGA (Color Graphics Adapter), HGC (Hercules Graphics Card), EGA (Enhanced Graphics Adapter), VGA (Video Graphics Array), SuperVGA (SVGA).

2.1.4. КЛАВИАТУРА

Клавиатура IBM PC предназначена для ввода в компьютер информации от пользователя [5]. На рис. 9 показана клавиатура для IBM PC AT, для других моделей компьютера расположение и число клавиш может несколько отличаться, но назначение одинаковых клавиш на различных компьютерах совпадает. Все клавиши на клавиатуре можно разбить на 4 группы: алфавитно–цифровая часть клавиатуры; функциональные клавиши; клавиши управления курсором; цифровая клавиатура.

Хотя применение клавиатуры в той или иной программе бывает совершенно индивидуальным, следует рассмотреть некоторые специальные клавиши клавиатуры (табл. 1).

Имеются комбинации клавиш, обрабатываемые специальным образом. Некоторые из них показаны в табл.2. Комбинация клавиш означает, что необходимо сначала нажать первую клавишу, не отпуская ее, нажать вторую и т.д.

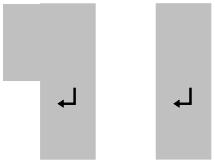
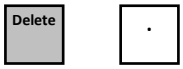
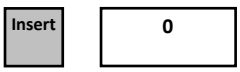
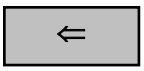
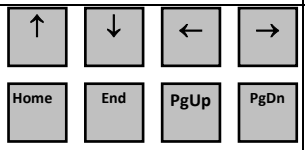
2.1.5. Печатающие устройства

Печатающие устройства предназначены для вывода информации на бумагу [4, 6]. Исторически первыми печатающими устройствами для ЭВМ были **литерные принтеры** с ограниченным набором печатаемых символов. Как правило, в настоящее время такие принтеры не применяются. Современные принтеры отличаются способами нанесения красителя на бумагу.

Рис. 10. Матричный принтер

Матричные принтеры (рис.10). Принцип печати этих принтеров таков: печатающая головка принтера содержит вертикальный ряд тонких металлических стержней (иголок). Головка движется вдоль печатаемой строки, а стержни в нужный момент ударяют по бумаге через красящую ленту. Это и обеспечивает формирование на бумаге символов и изображений.

Таблица 1

Клавиша	Назначение
1	2
	Клавиша **Enter** (Return или CR) предназначена для окончания ввода строки
	Клавиша **Del** (**Delete** – удаление) используется для удаления символа, находящегося под курсором
	Клавиша **Ins** (**Insert** – вставка) предназначена для переключения между двумя режимами ввода символов
	**Backspace** (стрелка влево над клавишей **Enter**) удаляет символ, слева от курсора
	**Клавиши перемещения курсора** предназначены, как правило, для перемещения курсора или «перелистывания» на экране текста
	Клавиша **NumLock** (блокировка цифр) предназначена для переключения между двумя режимами работы

	цифровой клавиатуры
Esc	Клавиша **Esc** (escape – убегать, спасаться), как правило, используется для отмены какого-либо действия, выхода из режима программы и т.д.
F1 – F12	Функциональные клавиши предназначены для различных специальных действий. Их действие определяется выполняемой программой
⇑	Клавиша **Shift** для ввода прописных букв (режим **CapsLock** отключен)
Caps Lock	**CapsLock** включает и выключает режим ввода прописных букв
Control Alt	**Control** (**Ctrl**) и **Alt** вводятся в комбинации с другими клавишами, и программа может особым образом реагировать на такие комбинации
PrtSc	Клавиша **Print Screen** предназначена для распечатки текстового (графического) экрана
Scrol	Клавиша **Scroll Lock –** служебная клавиша для фиксации регистра

Таблица 2

Control Break	Завершение работы выполняемой программы или команды
Control Alt Delete	Перезагрузка DOS
Control C	Прекращение работы любой программы или команды MS-DOS (для DOS)
Control S	Приостановление выполнения программ (для DOS)
Control Num	Приостановление выполнения программ
Control PrtSc	Включение и выключение режима копирования на принтер выводимой на экран информации
Control P	Включение и выключение режима копирования на принтер выводимой на экран информации (для DOS)

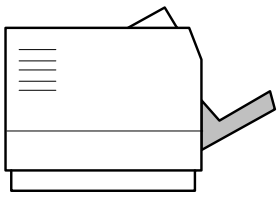


Рис. 11. Струйный принтер

Струйные принтеры (рис.11). В этих принтерах изображение формируется микрокаплями специальных чернил, выдуваемых на бумагу, с помощью сопел. Этот

способ печати обеспечивает более высокое качество печати по сравнению с матричными, он очень удобен для цветной печати.

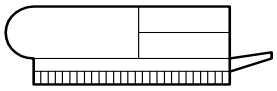


Рис. 12. Лазерный принтер

Лазерные принтеры (рис.12). В этих принтерах для печати используется принцип ксерографии: изображение переносится на бумагу со специального барабана, к которому электрически притягиваются частички краски, при этом печатающий барабан электризуется с помощью лазера по командам из компьютера. Лазерные принтеры обеспечивают в настоящее время наилучшее качество и скорость печати.

2.1.6. Ручные манипуляторы

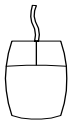


Рис.14. Мышь

Ручные манипуляторы или координаторные устройства предназначены для облегчения перемещения курсора по экрану, изменения рабочей среды, выбора действия, отметки блоков информации, рисования объектов и тому подобное [6].

Наиболее популярный тип координатного устройства – манипулятор типа ***мышь*** (Mouse) (рис.14). Мышь представляет собой легко умещающуюся в ладони коробочку с кнопками и проводом. Перемещение мыши по столу вызывает перемещение курсора мыши по экрану, а команды передаются нажатием кнопок. Наиболее распространены механические мыши с шариком в основании. Самые точные, надежные и дорогие мыши – оптические, построенные на основании светового индикатора. Известны также «безхвостные» мыши с автономным питанием и радиопередачей сигналов. Помимо манипуляторов типа мышь используются ***шаровые манипуляторы*** (***trackboll***) и ***джойстики*** (***joystick***).

2.1.7. Устройства ввода изображений

Устройства ввода изображений предназначены для считывания графической информации в компьютер [5]. Для ввода изображения наиболее часто используется ***сканер*** (***scanner***) (рис. 15).

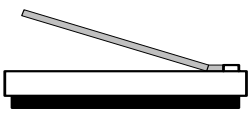


Рис.15. Сканер

Сканер создает в компьютере электронную копию изображения. Изображение может быть текстом, рисунком, фотографией, диаграммой, проекцией трехмерного предмета на плоскость или чем-нибудь другим. Оно считывается многоэлементными фотоприемными линейками с использованием протяженного осветителя и

объектива. Существуют: ручные сканеры (оператор сам должен перемешать ручной сканер по изображению); планшетные сканеры (для считывания изображения в него кладется лист бумаги или целая пачка при наличии автоподатчика); роликовые сканеры (сами протягивают бумагу сквозь себя); проекционные сканеры (обеспечивают ввод как документов, так и проекций трехмерных объектов). Кроме ввода графической информации, в последнее время сканеры стали использовать для ввода текстовой информации с переводом ее в формат ASCII. Для этого используют аппаратные и программные средства оптического распознавания символов. В системах автоматического конструирования (САПР) для ввода чертежей в компьютер используется ***графический планшет***.

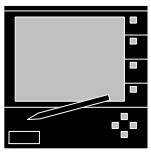


Рис.16. Диджитайзер

Для ручного создания рисунка можно использовать такое устройство как, ***диджитайзер*** (***digitizer –оцифровщик***)(рис. 16). Фактически – это планшет для рисования специальным пером. В настоящее время ведутся активные разработки в области перьевого ввода текста, но о повсеместном применении pen–распознавания говорить пока рано.

2.1.8. Коммуникационное оборудование

Связь между ЭВМ необходима для обмена информацией с другими компьютерами, кроме того, широко применяется управление удаленными компьютерами. В зависимости от степени удаленности машин для обеспечения компьютерной коммуникации необходимо различное оборудование [5,6].

Связь между двумя компьютерами, находящимися в одном месте, осуществляется посредством кабеля, которым нужно соединить последовательные или параллельные порты. Для совместного функционирования многих компьютеров в системе используются ***локальные вычислительные сети*** (ЛВС). Персональный компьютер подключается к ЛВС посредством сетевого адаптера. Наиболее простые, так называемые ***одноранговые***, ЛВС предназначены для рабочих групп. Они позволяют операторам нескольких компьютеров использовать общие диски, принтеры и другие устройства, передавать друг другу сообщения и выполнять другие коллективные операции. В ***многоранговых*** ЛВС для хранения разделяемых данных и программ используются серверы – более мощные компьютеры, чем машины пользователя. В последнее время большое значение приобретает объединение ЛВС для совместного функционирование в так называемых ***сетях RAN*** и ***WAN – Regional Area Network*** и ***Wide Area Network***.

Для передачи данных на действительно большие расстояния используются линии телефонной сети и выделенные линии. Преобразование цифровой информации в сигналы речевого диапазона (модуляцию) и звуковых сигналов в цифровую информацию (демодуляцию) выполняет ***модем*** – модулятор–демодулятор (рис.17). Используются также ***факс–модемы*** – устройства, сочетающие возможности модема и средства для обмена факсимильными изображениями с другими факс-модемами и обычными телефаксными аппаратами (рис.18).

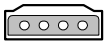


Рис. 17. Модем

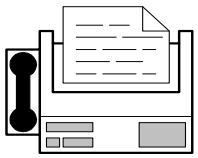


Рис. 18. Факс–модем

2.2. ОПЕРАЦИОННАЯ СИСТЕМА

2.2.1. Начальные сведения об операционной истеме

Операционная система – это программа, которая загружается при включении компьютера. Она производит диалог с пользователем, осуществляет управление компьютером, его ресурсами (оперативной памятью, местом на дисках и т.д.), запускает другие (прикладные) программы на выполнение. Операционная система обеспечивает пользователю и прикладным программам удобный способ общения (интерфейс) с устройствами компьютера. Основная причина необходимости операционной системы состоит в том, что элементарные операции для работы с устройствами компьютера и управления ресурсами компьютера – это операции очень низкого уровня, поэтому действия, которые необходимы пользователю и прикладным программам, состоят из нескольких сотен или тысяч таких элементарных операций. Операционная система скрывает от пользователя эти сложные и ненужные ему подробности и предоставляет ему удобный интерфейс для работы. Она выполняет также различные вспомогательные действия, например копирование и печать файлов. Кроме того, операционная система осуществляет загрузку в оперативную память всех программ, передает им управление в начале их работы, выполняет различные вспомогательные действия по запросу выполняемых программ и освобождает занимаемую программами оперативную память при их завершении [1,5].

В последнее время получили распространение следующие операционные системы: MS DOS, PC DOS, DR DOS, OS/2, Windows 3.1x, Windows-95, Windows 98, Windows 2000, Windows-NT.

2.2.2. Основные составные части операционной системы ms–dos

Операционная система состоит из следующих частей [5,6].

Базовая система ввода-вывода (BIOS), находящаяся в постоянной памяти (ПЗУ) компьютера. Эта часть операционной системы является «встроенной» в компьютер. Ее назначение состоит в выполнении наиболее простых и универсальных услуг операционной системы, связанных с осуществлением ввода–вывода. Базовая система ввода–вывода содержит также тест функционирования компьютера, проверяющий работу памяти и устройств компьютера при включении его электропитания. Кроме того, базовая система ввода–вывода содержит программу вызова загрузчика операционной системы.

Загрузчик операционной системы – это программа, функция которой заключается в считывании в память модулей операционной системы, которые и завершают процесс загрузки.

Дисковые файлы – два файла (*IO.SYS* и *MSDOS.SYS*), которые загружаются в память загрузчиком операционной системы и остаются в памяти компьютера постоянно. Файл *IO.SYS* – дополнение к базовой системе ввода–вывода. Файл *MSDOS.SYS* реализует основные высокоуровневые услуги DOS.

Командный процессор MS-DOS обрабатывает команды, выводимые пользователем. Командный процессор находится в дисковом файле *COMMAND.COM* на диске, с которого загружается операционная система. Некоторые команды пользователя (внутренние) командный процессор выполняет сам, для выполнения остальных (внешних) команд пользователя командный процессор ищет на дисках программу с соответствующим именем, и если находит ее, то загружает в память и передает ей управление. По окончании работы программы командный процессор удаляет программу из памяти и выводит сообщение о готовности к выполнению команд.

Внешние команды MS-DOS – это программы, поставляемые вместе с операционной системой в виде отдельных файлов. Эти программы выполняют действия обслуживающего характера.

Драйверы устройств – это специальные программы, которые дополняют систему ввода–вывода MS–DOS и обеспечивают обслуживание новых устройств или нестандартное использование имеющихся устройств.

2.2.3. Основные понятия операционной системы Ms–Dos

Как ранее было упомянуто, операционная система предназначена для осуществления управления компьютером, его ресурсами (оперативной памятью, местом на дисках и т.д.) и обеспечения пользователю и прикладным программам удобного способа общения (интерфейс) с устройствами компьютера. При работе с операционной системой используются некоторые общие понятия, которыми необходимо оперировать при работе с MS–DOS [1,2,6].

Программа (Program). Последовательность операций, выполняемых вычислительной машиной для реализации какой–либо задачи, часто называют программным обеспечением, или прикладной программой. Программы пишутся на специальных машинных языках и хранятся в файлах. Выделяют резидентные программы, которые после завершения работы оставляют в памяти машины какую-либо свою часть, которая позволяет либо осуществлять постоянно заданные функции, либо вызывать в любой момент, в том числе и во время выполнения другой программы, данную программу.

Файл (File). Это поименная область на диске или другом машинном носителе, содержащая определенную совокупность данных. В файлах могут храниться тексты программ, документы, готовые к выполнению программы и т.д. Использование понятия файла существенно облегчает работу пользователя, т.к. в этом случае он не обязан знать, в каком физическом порядке и где именно находятся его данные. Чтобы различить файлы, каждый имеет собственное имя, причем компьютеру безразлично, какое имя носит тот или иной файл, поскольку он получает от операционной системы инструкции низкого уровня.

Имя файла (Filename). Принято, что каждое название файла состоит из двух частей: собственно имени файла (filename) и расширения (extension). Имя файла имеет длину от 1 до 8 символов. В имени файла можно использовать любые символы из следующих: A...Z a...z 0...9 ’ ` ! @ # $ % ^ & ( ) _ - { } . Эти символы можно применять как в имени файла, так и в его расширении. В русифицированных версиях MS–DOS можно применять русские буквы в именах файлов. Расширение имени файла (extension) состоит из точки, за которой следует от одного до трех символов. Использование расширения является необязательным. Оно, как правило, описывает содержимое файла, поэтому использование расширения весьма удобно. По расширению можно узнать, какая программа создала файл. Примеры:

.com , .exe	готовые к выполнению программы
.bat	командные (Batch) файлы
.pas	программы на Паскале
.arj , .rar , .zip	архивные файлы
.bak	копия файла, делаемая перед его изменением

Каталог (Directory). Каталог – это специальное место на диске, в котором хранятся имена файлов, сведения о размере файлов, времени их последнего обновления, атрибуты (свойства) файлов и т.д. Если в каталоге хранится имя файла, то говорят, что этот файл находится в данном каталоге. Все каталоги, кроме корневого, на самом деле являются файлами специального вида. Каждый каталог имеет имя, и он может быть зарегистрирован в другом каталоге. Требования к именам каталогов те же, что и к именам файлов. На каждом магнитном диске имеется один главный, или корневой, каталог. В нем регистрируются файлы и подкаталоги (каталоги первого уровня). В каталогах 1–го уровня регистрируются файлы и каталоги 2–го уровня и т.д. Получается иерархическая древообразная структура каталогов на магнитном диске.

Дисковод (Disk Drive). Под дисководом понимают устройство, которое вращает диск и обеспечивает перемещение головок при записи/чтении. Иногда этим именем называют и накопитель на магнитных дисках. Приводы гибких дисков обозначаются как A и B, жесткого – как C.

В операционной системе MS–DOS можно разделить жесткий диск на несколько частей и работать с ними как с отдельными дисками. Эти диски называются логическими дисками или разделами жесткого диска. Каждый логический диск имеет имя (букву), по которому к нему можно обращаться (C: , D: , E: , F: ...).

Команда (Command). Под командой понимают как команду, которую пользователь вводит с клавиатуры, так и программу, которая эту программу исполняет. Работа с MS–DOS сводится к выполнению программ – прикладных программ пользователя и сервисных программ MS–DOS, в том числе встроенных в командный процессор. Программы обоих типов вызываются командами, которые дает пользователь. Чтобы дать команду, необходимо набрать ее текст после приглашения MS–DOS и нажать кнопку ввода – **Enter**. Все команды MS–DOS можно разделить на внутренние и внешние команды. Внутренние команды MS–DOS встроены непосредственно в командный процессор (***COMMAND.COM***). Программы, находящиеся на магнитных дисках в виде COM– и EXE–файлов, вызываются на выполнение посредством внешних команд MS–DOS. Кроме того, внешней командой можно вызывать файлы с расширением BAT, которые не являются программами, а списками команд MS–DOS.

К внутренним командам MS–DOS относятся следующие.

Таблица 3

Команда	Назначение
VER	Вывод номера версии MS–DOS
DATE	Вывод на экран или установка даты
TIME	Вывод или установка системного времени

CLS	Очистка экрана
DIR	Вывод содержимого каталога
COPY	Копирование файла в другое место
MD	Создание каталога
CD	Вывод на экран текущего каталога или его смена
REN	Изменение имени файла
DEL	Удаление одного или нескольких файлов

Здесь приведен неполный список внешних команд MS-DOS

2.3. Программы–оболочки

Основное назначение программ–оболочек – облегчить пользователю работу с файлами на диске, запуск программ, доступ к функциям управления файлами, каталогами и дисками, и, кроме того, они предоставляют пользователю ряд дополнительных возможностей. Рассмотрим особенности работы в некоторых из них [5,6].

2.3.1. Norton Commander

Оболочка Norton Commander была выпущена в 1986 году фирмой Peter Norton Computing [2,6]. Программа Norton Commander позволяет выполнять большое количество различных функций, в частности:

– наглядно изображать содержание каталогов на дисках;

– изображать дерево каталогов с возможностью перехода в нужный каталог с помощью указания его на этом дереве, а также создания, переименования и удаления каталогов;

– удобно копировать, переименовывать, пересылать и удалять файлы;

– просматривать текстовые файлы, документы, сделанные с помощью различных редакторов текстов, базы данных и таблицы табличных процессоров;

– редактировать текстовые файлы;

– выполнять любые команды DOS;

– изменять атрибуты файлов;

– с помощью одного нажатия клавиши выполнять стандартные действия для каждого типа файлов;

– ... и многое другое.

Запуск Norton Commander осуществляется вводом команды ***NC.EXE***. После запуска экран имеет вид, приведенный на рис.20

На экране можно использовать одновременно до двух панелей любого типа, кроме того, можно вывести панели от верха до середины экрана. Для управления панелями Norton Commander можно использовать комбинации клавиш [6].

Ctrl – O	убрать панели с экрана или вывести панели на экран
Ctrl – P	убрать одну из панелей (не текущую) с экрана/вывести панель на экран

Ctrl – U	поменять панели местами
Ctrl – F1	убрать левую панель с экрана или вывести левую панель на экран
Ctrl – F2	убрать правую панель с экрана или вывести правую панель на экран
Alt – F1	вывести в левой панели оглавление другого диска
Alt – F2	вывести в правой панели оглавление другого диска

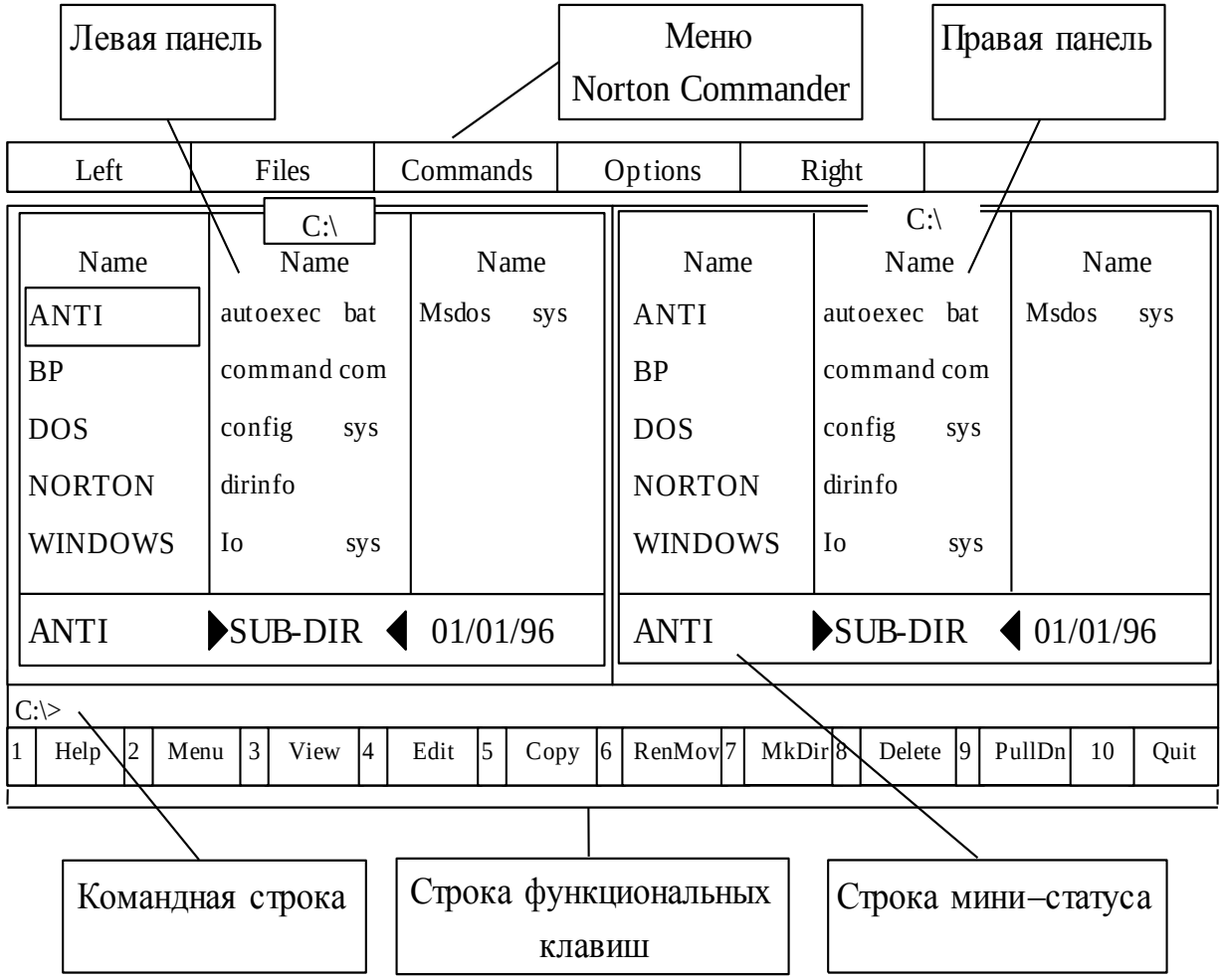


Рис. 20. Вид экрана при работе с Norton Commander

Оглавление каталога в панели. Если в панели Norton Commander выводится оглавление каталога, то сверху панели изображается имя этого каталога. При выводе имен файлов и подкаталогов Norton Commander использует такое правило: имена файлов выводятся строчными буквами, а имена подкаталогов – прописными.

Один из файлов или каталогов на экране выделен определенным цветом. Будем называть такой файл или каталог выделенным. Клавишами перемещения курсора можно перемещать выделенный участок на экране,

выделяя другой файл или каталог. Клавишей **Tab** можно перевести выделенный участок на другую панель Norton Commander.

Действия при нажатой клавиши Enter. Действие Norton Commander при нажатии клавиши **Enter** зависит от того, имеется что–либо в командной строке внизу экрана или нет. Если командная строка не пуста, то при нажатии клавиши **Enter** будет выполнена команда, содержащаяся в командной строке. Эта команда выполняется так же, как если бы она была введена в ответ на приглашение MS-DOS. Если же командная строка не содержит каких–либо символов, то действие Norton Commander зависит от того, что выделено на панели: имя файла или имя каталога. Если выделено имя каталога, то при нажатии клавиши **Enter** Norton Commander «войдет» в этот каталог и выведет его оглавление. Если выделено имя файла, то действие Norton Commander при нажатии клавиши **Enter** зависит от расширения этого файла, например:

.com, .exe, .bat – начнет выполнение этого файла;
.pas – будет вызван Turbo–Pascal;
.arj – будет вызвано оглавление архива.

Действие, выполняемое для файла с данным расширением при нажатии клавиши **Enter**, задается файлом ***NC.EXT***. При отсутствии файла ***NC.EXT*** и для расширений, не упомянутых в файле ***NC.EXT***, никаких действий выполнено не будет.

Информационная панель. В панели Norton Commander можно вывести сводную информацию о диске и каталоге на другой панели. Наверху информационной панели выводится строка «***Info***». В панели изображаются следующие сведения:

– емкость оперативной памяти компьютера в байтах (...*Bytes Memory*);

– количество свободной оперативной памяти в байтах (...*Bytes Free*);

– емкость текущего дисковода в байтах (... *bytes on drive* ...);

– количество свободного места на текущем диске (... *bytes free on drive*...);

– количество файлов в каталоге, выведенном на другой панели Norton Commander, и их общий размер в байтах (... *files use* ... *bytes in* ...).

Ниже, в информационной панели, выводится содержание файла с именем ***DIRINFO***, а если этого файла в каталоге нет, то сообщение

«*No ‘dirinfo’ file in this directory*»

«В этом каталоге нет файла dirinfo».

Для вызова на экран информационной панели необходимо нажать комбинацию клавиш **Ctrl – L**. При этом та панель, которая не является текущей, станет информационной. Для возврата в режим просмотра каталогов необходимо опять нажать **CTRL – L**.

Использование функциональных клавиш. В нижней строке экрана Norton Commander выводит напоминание о значении функциональных клавиш [5]. Ниже кратко описывается их назначение:

F1 – *Help* – краткая информация о назначении клавиш при работе с Norton Commander;

F2 – *Menu* – запуск команд, указанных в списке, заданном пользователем (пользовательское меню);

F3 – *View* – просмотр файла. Можно просматривать текстовые файлы, базы данных, таблицы табличных процессоров, рисунки графических редакторов, просматривать содержимое архивных файлов, проигрывать звуковые WAV–файлы;

F4 – *Edit* – редактирование файла. Для редактирования может быть использован встроенный редактор Norton Commander или любой другой редактор, указанный пользователем;

F5 – *Copy* – копирование файла. В середине экрана появляется запрос о том, куда копировать файл. По умолчанию файл копируется в каталог, изображенный на другой панели. Можно набрать и другое имя файла (каталога). Затем для копирования надо нажать **Enter**, для отмены команды –**Esc;**

F6 – *Renmov* – переименование файла (каталога) или пересылка файла в другой каталог. Можно задать новое имя файла (каталога) или имя каталога, в который надо пересылать данный файл. Для начала переименования или пересылки надо нажать **Enter**, для отмены команды – **Esc;**

F7 – *MkDir* – создание подкаталога;

F8 – *Delete* – уничтожение файла или подкаталога;

F9 – *PullDn* – вывод меню, содержащего режимы работы Norton Commander;

F10 – *Quit* – выход из Norton Commander.

Если нажать клавишу **Alt**, то последняя строка экрана изменится. В ней будет выводиться подсказка о значении комбинации клавиш (**Alt – F1**) – (**Alt – F10**). Ниже кратко описано назначение комбинаций этих клавиш.

Alt – F1 – ***Left* –** выбор диска, изображаемого на левой панели;

Alt – F2 – ***Right* –** выбор диска, изображаемого на правой панели;

Alt – F3 – ***View* –** просмотр текстового файла. Этот режим вызывается быстрее, но позволяет просматривать только текстовые файлы и не имеет некоторых возможностей, доступных при просмотре с помощью клавиши **F3;**

Alt – F4 – ***Edit* –** редактирование файла с помощью альтернативного редактора;

Alt – F7 – ***Find* –** поиск файла на диске;

Alt – F8 – ***History* –** просмотр и повторное выполнение ранее введенных команд;

Alt – F9 – ***Ega Ln* –** переключение с 25 на 43 (для монитора EGA) или 50 (для монитора VGA) строк на экране, и наоборот;

Alt – F10 – быстрый переход в другой каталог.

Работа с файлами. Norton Commander позволяет выполнять различные действия над файлами (просмотр, редактирование, копирование, переименование, пересылка, удаление, поиск файла на диске, изменение атрибутов) и группами выделенных файлов (копирование, перемещение, переименование, уничтожение, изменение атрибутов) [4]. Выбор отдельного файла (т.е. помещение этого файла в группу) осуществляется нажатием клавиши **Insert** (**Ins**). Отмена выбора – повторным нажатием клавиши. Чтобы выбрать группу файлов по маске необходимо нажать клавишу «+» (плюс на цифровой клавиатуре) и задать маску для выбора. Для отмены выбора необходимо нажать клавишу «–» (минус на цифровой клавиатуре) и задать маску для отмены выбора.

Просмотр файлов. При нажатии клавиши **F3** Norton Commander позволяет просматривать выделенный курсором файл. Для перемещения по просматриваемому файлу можно использовать клавиши перемещения курсором, при просмотре баз данных можно использовать клавиши «+» «–» (плюс и минус на цифровой клавиатуре) для перемещения по полям записи.

При просмотре текстовых файлов и документов имеются следующие дополнительные возможности:

– **F2** – переносить или нет на другую строку длинные строки документов;

– **F4** – вывод файла в шестнадцатеричном виде и выход из этого режима;

– **F7** – поиск подстроки в документе. Нужную подстроку нужно ввести в ответ на запрос;

– **Shift – F7** – повторение поиска той же подстроки в документе;

– **F8** – выбор режима просмотра документа;

– **F10** – выход из режима просмотра.

Редактирование файла. Для редактирования выделенного курсором файла следует нажать **F4**, если используется встроенный редактор, или **Alt – F4**, если используется альтернативный редактор. Опишем возможности встроенного редактора Norton Commander.

Курсор. Курсор (мигающий символ на экране, похожий на знак подчеркивания) указывает на текущую позицию в тексте. Все изменения в тексте и вставки нового текста происходят в той позиции, на которую показывает курсор.

Перемещение курсора по тексту. Курсор можно перемещать с помощью клавиш управления курсором **(← → ↑ ↓)** на одну позицию влево, вправо, вверх и вниз. Кроме того, курсор можно перемещать по тексту с помощью следующих клавиш:

– **PgUp** и **PgDn** на страницу (размер экрана) вверх и вниз;
– **Home** и **End** на начало и конец редактируемого файла;
– **Ctrl – ←** и **Ctrl – →** на слово влево и вправо.

Ввод текста. Для ввода текста нужно переместить курсор в то место, в которое надо вводить новый текст, и начать набор текста, нажимая соответствующие буквенно–цифровые клавиши. Вводимый текст помещается в ту позицию, в которой находится курсор. Для окончания строки надо нажать клавишу **Enter**.

Ввод символа из верхнего регистра клавиатуры. Если необходимо ввести символ из верхнего регистра клавиатуры, необходимо нажать клавишу **Shift** и, не отпуская ее, нажать клавишу с нужным символов.

Удаление символов и строк. Для удаления символов и строк можно использовать следующие клавиши:

– **Delete** или **Del** – удаление символа под курсором;
– **BackSpace** – удаление символа слева от курсора;
– **Ctrl – Y** – удаление строки;
– **Ctrl – K** – удаление текста от текущей позиции курсора до конца строки.

Назначение функциональных клавиш. При редактировании файла с помощью встроенного в Norton Commander редактора можно использовать следующие функциональные клавиши:

– **F2** – сохранить отредактированный файл;
– **Shift – F2** – сохранить отредактированный файл под другим именем (новое имя запрашивается);
– **F10** – выйти из режима редактирования (это можно сделать также, нажав клавишу **Esc**);
– **Shift – F10** – сохранить отредактированный файл и выйти из режима редактирования;
– **F7** – поиск подстроки в документе. Нужная подстрока вводится в ответ на запрос;
– **Shift – F7** – повторный поиск той же подстроки в документе;
– **F1** – вывод на экран справки о назначениях клавиш при редактировании.

Создание файла с помощью редактора. Чтобы создать новый файл, необходимо нажать комбинацию клавиш **Shift – F4**. Если на диске

нет файла с введенным именем, то он будет создан и Norton Commander перейдет в режим редактирования файла.

Копирование файла (файлов). Для копирования файлов с помощью Norton Commander надо выделить нужный файл или выбрать группу файлов и нажать клавишу **F5**. В центре экрана появится запрос о том, куда надо копировать файл или файлы. В ответ на запрос можно:

– ввести имя каталога, в который надо копировать файл (файлы);

– ввести новое имя файла (если копируется более одного файла, то в этом имени должны быть символы * или ?);

– нажав клавишу **F10**, вывести на экран дерево каталогов текущего диска и выбрать в нем каталог, в который надо скопировать файл (файлы). Для выбора каталога надо выделить его с помощью клавиш перемещения курсора и затем нажать **Enter**.

После этого надо нажать клавишу **Enter** для начала копирования файлов или **Esc** для отмены копирования. Если на появившийся запрос просто нажать клавишу **Enter**, то файлы (файл) будут скопированы в каталог на другой (неактивной) панели с теми же именами. При копировании возможно возникновение особых ситуаций (нехватка места на диске при копировании и перезапись файла при копировании), при этом будет выдано соответствующее сообщение.

Переименование и пересылка файлов. Для переименования или пересылки в другой каталог файлов с помощью Norton Commander надо выделить переименовываемый или пересылаемый файл или выбрать группу файлов и нажать клавишу **F6**. Если файл пересылается в другой каталог того же диска, то содержимое файла остается на том же месте диска, а в другой каталог включается только ссылка на этот файл (т.е. элемент каталога), а из исходного каталога эта ссылка исчезает. Если же файл пересылается на другой диск, то он просто копируется на другой диск, и после успешного окончания копирования исходный файл уничтожается. Процесс пересылки или переименования файлов с помощью Norton Commander происходит аналогично копированию.

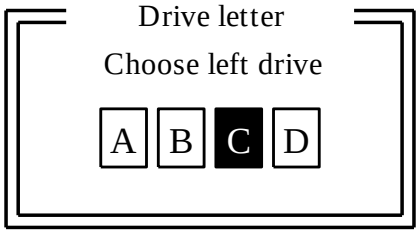
Удаление файлов. Для удаления файлов с помощью Norton Commander надо выделить нужный файл или выбрать группу файлов и нажать клавишу **F8**. При удалении файлов на экран выводится запрос на подтверждение удаления. Для подтверждения удаления нужно нажать клавишу **Enter**, для отмены – **Esc**. Можно также с помощью клавиш ←, → выбрать нужный ответ (**Delete** – удалить, **Cancel** – не удалять) и нажать клавишу **Enter**.

Работа с каталогами. Создание каталога. Для создания каталога следует нажать клавишу **F7**. Norton Commander выведет на экран запрос об имени каталога (*Create the directory*). Необходимо ввести это имя и нажать

Enter. Чтобы отменить создание подкаталога, следует нажать клавишу **Esc**.

Удаление каталога. Для удаления каталога следует указать его курсором и нажать клавишу **F8**.

Переход на другой диск. Для того чтобы в панели Norton Commander вывести оглавление другого диска, следует нажать **Alt – F1** – для левой панели, **Alt – F2** – для правой панели. На экран будет выведен список доступных дисков.



Затем надо выбрать (клавишами ←, →, либо нажав соответствующую букву) имя нужного диска и нажать **Enter**. Norton Commander прочтет оглавление текущего каталога на указанном диске и выведет его на экран.

Меню Norton Commander. С помощью меню Norton Commander можно установить наиболее удобный вид представления информации на экране, изменить режимы работы Norton Commander, а также выполнять некоторые другие действия [5,6].

Работа с меню. Для входа в меню следует нажать клавишу **F9**. В верхней строке экрана появится строка, содержащая пункты меню. Один из пунктов меню является выделенным. Для выбора нужного пункта меню следует использовать клавиши перемещения курсора ← и →. Выбрав нужный пункт меню, следует нажать клавишу **Enter**, и под ним откроется соответствующее ему подменю. Для выбора нужного пункта подменю следует использовать клавиши ↑ и ↓. Выбрав нужный пункт подменю, следует нажать клавишу **Enter**.

Выход из меню. Для выхода из меню или подменю Norton Commander следует использовать клавишу **Esc**.

Пункты меню «***Left»*** и «***Right»*** задают режимы вывода информации соответственно в левой и правой панелях Norton Commander.

Пункт меню «***Files»*** дает возможность производить различные операции с файлами. Многие из этих операций закреплены за функциональными клавишами.

Пункт меню «***Commands»*** позволяет выполнять различные команды Norton Commander (вывод на экран дерева каталогов, поиск файла на диске и др.).

Пункт меню «***Options»*** позволяет задавать конфигурацию и режимы работы Norton Commander и указывать, какой редактор будет использоваться при редактировании файлов.

Меню команд пользователя. При нажатии пользователем клавиши **F2** Norton Commander выводит на экран список команд, указанный пользователем в файле ***NC.MNU***. Пользователь может клавишами перемещения курсора ↑ и ↓ выделить нужный пункт этого списка и, нажав клавишу **Enter**, выполнить соответствующие команды.

2.3.2. MICROSOFT WINDOWS

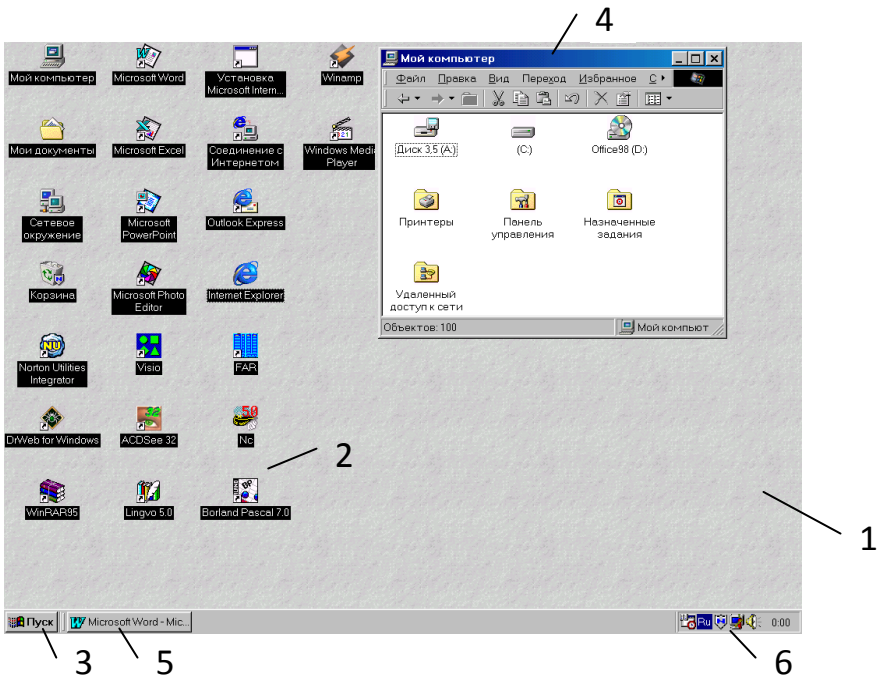
Операционная система Windows 98 является, по сути, эволюционным продолжением Widows 95, которая полностью изменила принцип управления файлами и интерфейс, что сделало операционную систему доступной даже для новичков. Эта система обладает высокой универсальностью имеет самое широкое распространение, и соответственно, имеет особую поддержку со стороны производителей аппаратного и программного обеспечения. Для компьютера, работающего в этой системе, наиболее просто подобрать прикладные программы и драйверы устройств.

Почти все сказанное об операционной Windows 98, можно отнести и к Windows 95, а многое справедливо и для Windows NT.

Windows 98 является графической операционной системой для компьютеров платформы IBM PC, представляет собой высокопроизводительную, многозадачную и многопотоковую 32–разрядную операционную систему с графическим интерфейсом и расширенными сетевыми возможностями, работающую в защищенном режиме [7–9]. Её основные средства управления – графический манипулятор (мышь или иной аналогичный) и клавиатура. Система предназначена для управления автономным компьютером, но также содержит все необходимое для создания локальной компьютерной сети и имеет средства для интеграции компьютера во всемирную сеть (Интернет).

Запуск Windows 98 осуществляется автоматически при включении компьютера. После загрузки на экране появится рабочий стол Windows (рис.21).

Работа с манипулятором мышь. При работе с операционной системой Windows большую часть операций можно осуществлять с помощью компьютерной мыши. Компьютерная мышь является манипуляторным устройством, позволяющим общаться с компьютером через операционную систему. Мышь на экране управляет указателем (курсором), вид которого



зависит от выполняемой операции. Движение мыши по ровной поверхности (столу) автоматически повторяется соответствующим перемещением указателя на экране. В Windows 98 используются две клавиши мыши:

– ***левая клавиша*** – для выполнения действий (основная);

– ***правая клавиша*** – для получения информации по свойствам любого объекта.

С помощью мыши пользователь может выполнить следующие операции [7, 8]:

– ***щелчок*** (***фиксация***). Для выполнения необходимо быстро нажать и отпустить кнопку мыши в тот момент, когда вершина стрелки указателя мыши неподвижна и находится на нужном объекте. Одна из самых распространенных операций, выполняемая щелчком мыши – выбор объекта.

– ***двойной щелчок*** производится быстрым двойным нажатием на кнопку мыши без ее перемещения. Двойной щелчок выполняют после того, как указатель мыши помещен на объект (элемент), при этом щелкнуть надо быстро иначе система воспримет их как два одиночных щелчка и выполнит другие команды. Двойной щелчок используется при открытии документов, запуске программ.

1 – рабочий стол; 2 – папки и значки; 3 – кнопка «**Пуск**»; 4 – открытое окно; 5 – кнопка свернутого окна приложения; 6 – значки утилит, запускаемых в фоновом режиме.

– ***перетаскивание*** (***перемещение***) объекта производится после установки указателя мыши на нужном объекте (элементе). Затем, нажав левую кнопку мыши и не отпуская ее, перемещают мышь, двигая объект или элемент к конечному положению. Кнопку мыши отпускают в тот момент, когда хотят закончить операцию. Перетаскивание широко

применяется для перемещения значка папки или файла, окна или его границы.

Использование меню «Пуск». Меню «**Пуск**» является основным рабочим инструментом для запуска программ. Чтобы активизировать меню «**Пуск**», необходимо курсор мыши подвести к кнопке «**Пуск**» и нажать левую клавишу мыши. На экране появится меню «**Пуск**». После этого, перемещая курсор мыши вдоль пунктов меню, можно раскрывать соответствующие подменю (рис. 22).

Когда нужное имя программы, документа или команды будет подсвечено, для запуска или открытия необходимо еще раз нажать левую кнопку мыши. Можно запустить программу, документ или команду из меню и без использования мыши. Для этого необходимо нажать комбинацию клавиш **Ctrl – Esc**, а далее перемещение по пунктам меню осуществляется с помощью клавиш перемещения курсора (←, ↑, →, ↓). Для запуска или открытия необходимо нажать клавишу **Enter**. Например, запустите программу «Калькулятор» (***Пуск–Программы–Стандартные–Калькулятор***) (рис.22). На экране появится программа «Калькулятор» (рис. 23). Для закрытия программы подведите указатель мыши к кнопке закрытия окна (рис.24) и нажмите левую кнопку мыши, или закройте программу с помощью комбинаций клавиш **Alt –F4.**

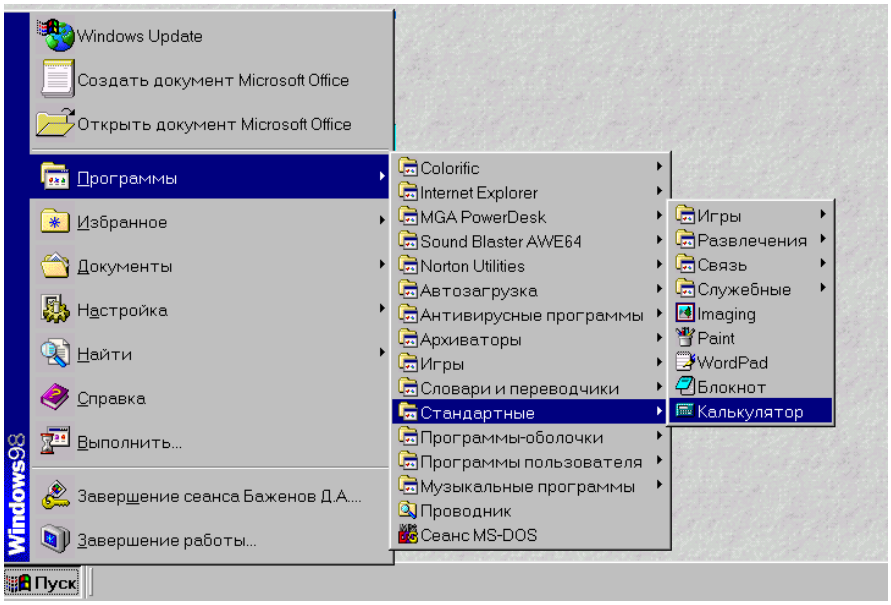


Рис. 22. Меню «**Пуск**»

Использование ярлыков на рабочем столе. Ярлыки – это значки быстрого доступа к программам и документам. Двойным щелчком мыши на ярлыке на рабочем столе вы запускаете соответствующую программу и

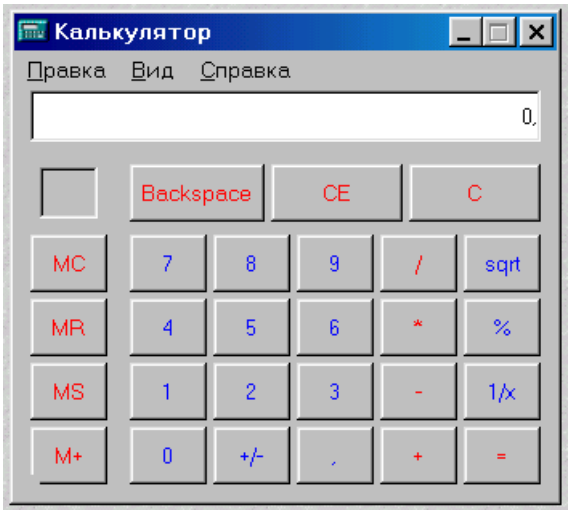


Рис. 23. Программа «Калькулятор»

затем загружаете нужный документ (в Windows 98 существует возможность запуска программ одним щелчком мыши при использовании активного рабочего стола). Чтобы запустить программу или открыть документ с помощью ярлыка, необходимо подвести указатель мыши к необходимому значку на рабочем столе и два раза быстро нажать левую кнопку мыши. Соответствующая программа или документ откроются автоматически. Аналогично открываются и окна. Например, найдите на рабочем столе ярлык «***Мой компьютер***» и откройте его (рис. 24).

Существует несколько способов открытия окна:

– щелкнуть на кнопке «**Пуск**», а затем на имени команды, меню, папки или документа;

– щелкнуть на нужном вам ярлыке на рабочем столе;

– в окне «***Мой компьютер***» щелкнуть на значке программы или документа;

– щелкнуть на кнопке панели, например панели Microsoft Office.

Переключение между окнами [1,2,8]**.** Одновременно вы можете запускать несколько программ, и окна каждой из них будут располагаться на рабочем столе, но только одно окно из всех будет активным. Если вы хотите работать в окне, его строка заголовка должна быть выделена.

Активное окно отличается от других цветом строки заголовка, – как правило, более темным, а в других окнах строки заголовков светлее. Если окна перекрываются, активное окно будет располагаться поверх других, а соответствующая ему кнопка на панели задач будет выглядеть нажатой.

Вы можете выбрать окно и, таким образом, переключиться с одной программы на другую. Это можно сделать с помощью следующих действий:

– щелкнуть по кнопке окна на панели задач;

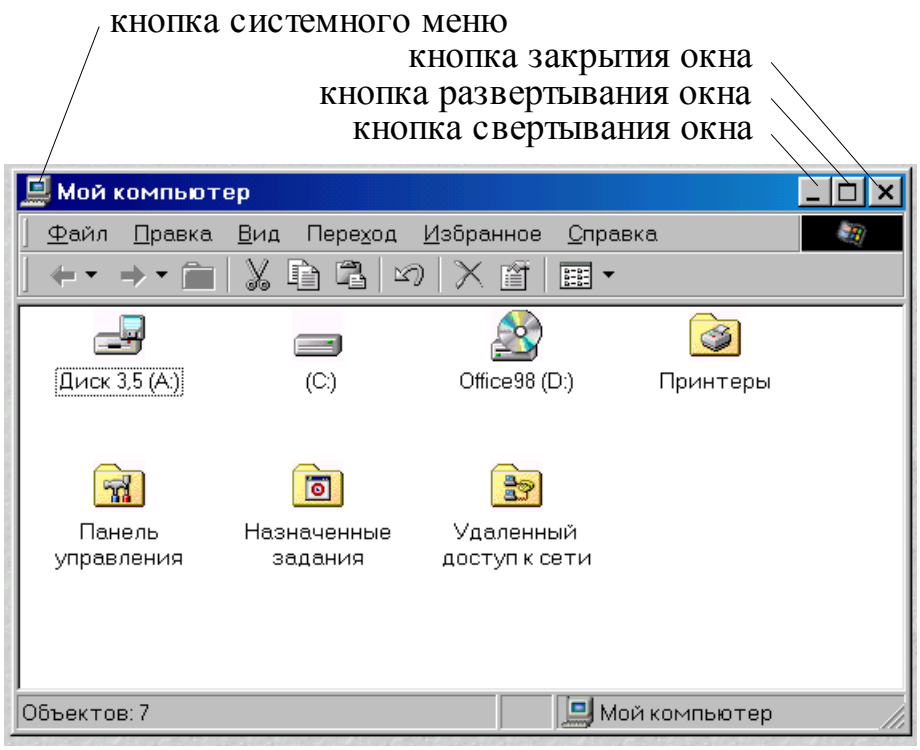


Рис. 24. Окно «***Мой компьютер***»

– щелкнуть по строке заголовка окна;

– щелкнуть по любой видимой части окна;

– удерживая клавишу **Alt**, нажимать клавишу **Tab** до тех пор, пока не будет выделено окно нужной вам программы. Когда вы окажетесь в нужном вам окне, отпустить клавишу **Alt**.

Закрытие окна. Существует несколько способов закрытия окна:

– щелкнуть по кнопке закрытия окна в верхнем правом углу окна (рис.24);

– щелкнуть на значке в верхнем левом углу окна и выбрать опцию «***Закрыть***»;

– раскрыть меню «***Файл***» и выбрать команду «***Exit***» (Выход, Закрыть и т.п.);

– щелкнуть правой кнопкой мыши по кнопке окна в панели задач и выбрать команду «***Закрыть***»;

– нажать комбинацию клавиш **Alt – F4**.

Свертывание и развертывание окна. В Windows существует возможность свертывания и развертывания окна. Существует несколько способов свернуть и развернуть окно:

– щелкнуть по кнопке свертывания или развертывания в верхнем правом углу окна;

– щелкнуть по значку в верхнем левом углу окна и из появившегося системного меню выбрать команду «***Свернуть***» или «***Развернуть***» (рис. 25);

– щелкнуть правой кнопкой мыши по кнопке окна в панели задач и выбрать из появившегося системного меню команду «***Свернуть***» или «***Развернуть***» (рис. 26).

Перемещение окна. Самый простой способ переместить окно на рабочем столе – перетащить его с помощью мыши. Для этого необходимо поместить указатель мыши на строку заголовка окна, щелкнуть левой кнопкой мыши и, удерживая ее нажатой, перетащить окно на нужное место,отпустить кнопку мыши. Можно также переместить окно с помощью клавиатуры. Для этого необходимо нажать комбинацию клавиш

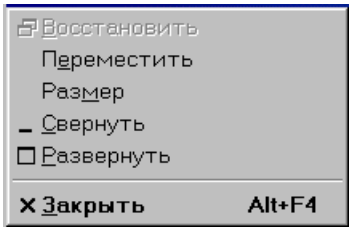


Рис. 25. Системное меню

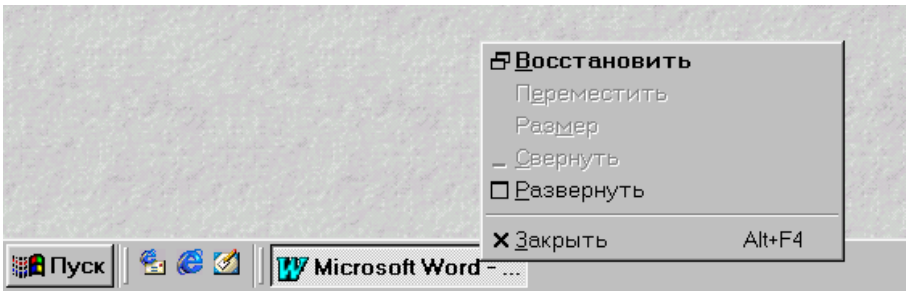


Рис. 26. Системное меню в панели задач

Alt – пробел, чтобы открыть системное меню в верхнем левом углу окна, выбрать команду «***Переместить***» и с помощью клавиш управления курсором поместить окно в нужное место, завершив перемещение, нажать клавишу **Enter**.

Изменение размеров окна. Для изменения размеров окна необходимо поместить указатель мыши в какой – либо угол или на одну из границ окна (указатель мыши при этом приобретает вид двунаправленной стрелки), щелкнуть левой кнопкой мыши и, удерживая ее нажатой, перемещать указатель мыши до придания окну желаемых размеров, отпустить кнопку мыши. Можно изменять размеры окна с помощью клавиатуры. Для этого необходимо нажать **Alt – пробел**, чтобы открыть

системное меню, выбрать команду «***Размер***» и затем с помощью клавиш управления курсором изменить размеры окна, нажать клавишу **Enter**.

Контекстное меню. Одной из самых полезных в Windows 98 является концепция контекстно–зависимых меню. Контекстное меню появляется на экране при щелчке правой кнопкой мыши. Контекстное меню предоставляет пользователю доступ к набору команд, соответствующих выполняемой в данный момент задаче. Чтобы отобразить контекстное меню на экране, необходимо поместить указатель мыши на любой пустой области рабочего стола и щелкнуть правой кнопкой мыши. В зависимости от местоположения указателя мыши в момент вызова контекстного меню его вид различается.

Окно «*Мой компьютер*». Окно «***Мой компьютер***» (рис. 24) является средством управления файлами, с помощью которого пользователь упорядочивает содержимое своего жесткого диска, подключает сетевые диски, периферийные устройства, управляет папками и файлами. Используя окно «***Мой компьютер***», пользователь может открыть окно диска или папки, открыть файл, запустить программу, перемещать, копировать, переименовывать, удалять любые файлы или папки, получать сведения об объектах, просматривать другие составляющие своей системы, просматривать Web – страницы.

Для навигации с помощью средств «***Мой компьютер***» можно использовать кнопки панели инструментов, меню или клавиатуру (рис. 27).

Чтобы просмотреть содержимое диска или папки, необходимо два раза щелкнуть на соответствующих значках мышью. Чтобы перейти к диску или папке более высокого уровня иерархии, необходимо щелкнуть на кнопке «***Вверх***» в панели инструментов. Чтобы вернуться к диску или папке, выбранной раньше, необходимо щелкнуть в панели инструментов на кнопке «***Назад***». Чтобы выбрать диск или папку, открытую раньше, необходимо щелкнуть на кнопке со стрелкой вниз, расположенной около кнопки «***Назад***», и выбрать нужный элемент из списка. Если к каким–то папкам и дискам осуществлялся переход с помощью кнопки «***Назад***», то с помощью кнопки «***Вперед***» можно проделать обратный путь. Перейти к любому диску или папке можно, набрав путь в поле адреса и нажав клавишу **Enter**.

Файлы и папки [7, 8]. Каждый файл на жестком диске имеет свое имя и занимает определенное место. В Windows 98 имена файлов могут содержать до 256 символов, включая пробелы и другие специальные символы. Файлы на диске хранятся в папках. Папка является аналогом каталога в операционной системе MS DOS. В каждой папке могут содержаться как файлы, так и другие папки. Точное местоположение файла называется путем к файлу, и в путь включаются все названия папок,

которые нужно открыть, чтобы получить доступ к файлу. Управление файлами и папками является основой работы с компьютером. Рассмотрим некоторые средства управления файлами.

Создание папок. Создать новую папку можно с помощью окна «***Мой компьютер***». Для этого необходимо:

1. Открыть окно «***Мой компьютер***».
2. Два раза щелкнуть левой кнопкой мыши на значке диска или папки, в которой вы хотите создать новую папку.
3. Подвести курсор мыши к опции меню «***Файл***» и щелкнуть левой кнопкой.
4. В раскрывшемся меню «***Файл***» выбрать опцию «***Создать***» → «***Папку***» и щелкнуть левой кнопкой мыши.

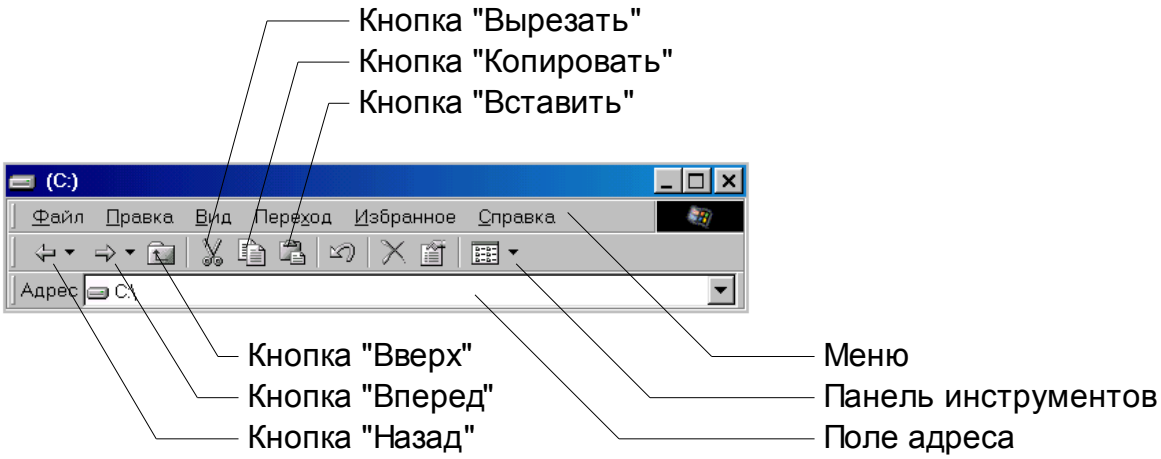


Рис. 27. Панель инструментов и меню окна «***Мой компьютер***»

На экране появится значок пустой новой папки. Имя этой папки – «***Новая папка»*** – будет подсвечено. Необходимо назвать эту папку любым именем и нажать клавишу **Enter.**

Можно создать новую папку с помощью контекстного меню. Для этого необходимо после того, как открыта папка или диск, в которой необходимо создать новую папку, щелкнуть правой кнопкой мыши в пустой области экрана, чтобы раскрылось контекстное меню. В нем необходимо выбрать опцию «***Создать***», а затем – «***Папку***».

Переименование файлов и папок. Для переименования файлов и папок необходимо щелкнуть правой кнопкой мыши на значке папки или файла и в контекстном меню выбрать опцию «***Переименовать***». Имя файла подсветится, после чего необходимо набрать новое имя и нажать клавишу **Enter**. Существует другой способ переименования файлов и папок. Для этого необходимо выделить папку или файл с помощью щелчка левой кнопки мыши, затем подвести указатель мыши к имени и

еще раз нажать левую кнопку. Имя файла подсветится, после чего необходимо набрать новое имя и нажать клавишу **Enter**.

Удаление файлов и папок. Для удаления необходимо выделить файл или папку, которую вы хотите удалить, и нажать клавишу **Delete** или же щелкнуть правой кнопкой мыши для открытия контекстного меню и выбрать в нем опцию «***Удалить***».

Восстановление удаленных файлов. Windows 98 позволяет некоторое время сохранять удаленные файлы в «***Корзине»***, что позволяет их восстанавливать в случае необходимости. Для восстановления необходимо открыть корзину, щелкнув на рабочем столе на значке «***Корзина***». Когда откроется окно «***Корзина***» (рис. 28), необходимо найти имя файла или папки, которую надо восстановить. После этого необходимо щелкнуть на нужном значке правой кнопкой мыши и выбрать в контекстном меню опцию «***Восстановить***». То же можно проделать и с помощью опций меню «***Файл***» → «***Восстановить***».

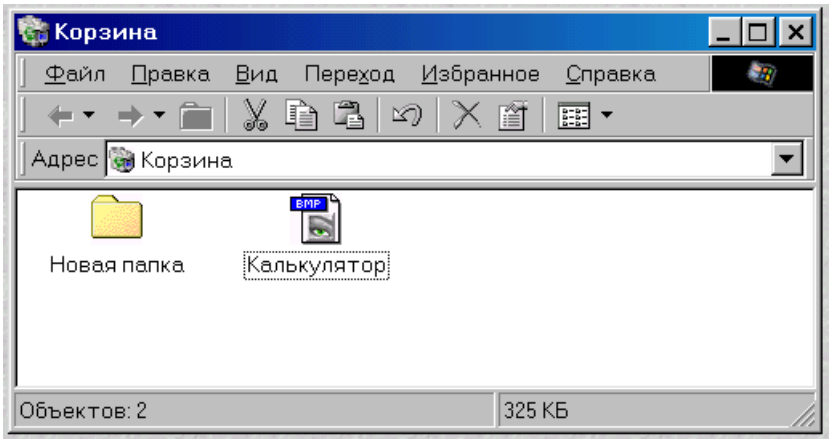


Рис. 28. Корзина

Копирование (Перемещение) файлов и папок. Для копирования (перемещения) файлов и папок необходимо выполнить следующую последовательность действий.

1. В окне «***Мой компьютер***» выбрать файл или папку, которую необходимо скопировать (переместить).

2. Щелкнуть на нужном значке правой кнопкой мыши и выбрать команду «***Копировать***» («***Вырезать***») из появившегося контекстного меню (можно также щелкнуть левой кнопкой мыши по соответствующей кнопке в панели инструментов (рис. 27)).

3. Выбрать папку для копируемого (перемещаемого) объекта.

4. Щелкнуть правой кнопкой мыши на пустом месте окна выбранной папки и выполнить команду «***Вставить***» из контекстного меню (можно также щелкнуть левой кнопкой мыши по соответствующей кнопке в панели инструментов (рис. 27)).

Выход из Windows и выключение компьютера. Завершив работу с программами, необходимо правильно выйти из Windows, используя команду «***Завершение работы***».

1. Закрыть все прикладные программы.
2. Щелкнуть по кнопке «**Пуск**» и выбрать команду «***Завершение работы***».
3. В появившемся диалоговом окне завершения работы с Windows выбрать опцию «***Выключить компьютер***» и щелкнуть по кнопке «**ОК**» (рис. 29).
4. После того, как на экране появится сообщение, что теперь питание компьютера можно отключать, выключить компьютер.

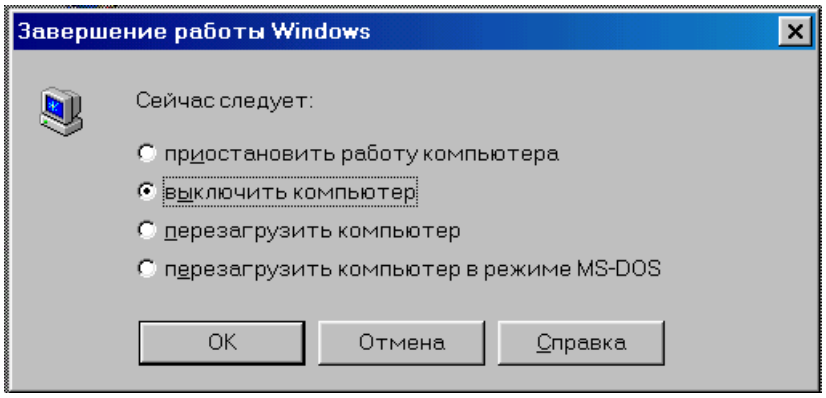


Рис. 29. Выход из Windows

3. ИНТЕГРИРОВАННАЯ СРЕДА ЯЗЫКА ТУРБО–ПАСКАЛЬ

Интегрированная среда – это средство организации диалога программиста с компьютером, разработанное для облегчения написания, запуска и отладки программ на языке Паскаль [3, 10, 14].

Интегрированная среда Турбо–Паскаль позволяет выполнить все этапы работы с программой:

- создать исходный текст программы на языке Паскаль;
- сохранить исходный текст программы на жёстком или гибком диске;
- отредактировать текст программы;
- выполнить программу.

Таким образом, при работе с Турбо – Паскалем уже нет необходимости использовать отдельный редактор (программу для набора и исправления текстов), компилятор (программу для создания пускового файла), отладчик (программу для поиска ошибок в процессе счета) и справочник, поскольку все эти средства, а также средства работы с файлами, запуска сервисных программ и некоторые другие встроены в Турбо-Паскаль и ко всем из них есть доступ из интегрированной среды.

3.1. Вход в интегрированную среду

Для входа в интегрированную среду следует вызвать на выполнение файл ***TURBO.EXE***. При необходимости можно задать файл, с которым следует работать, и использовать ключи командной строки, с помощью которых задаются параметры интегрированной среды.

После загрузки файла ***TURBO.EXE*** на экране дисплея появится основной экран интегрированной среды, имеющий вид, показанный на рис.30. Изображение в этом случае состоит из трех основных частей: строки основного меню, поля экрана и строки состояния. Строка основного меню содержит имена меню следующего уровня (подменю). Поле экрана предназначено для размещения открываемых окон. Строка состояния отражает состояние вычислительного процесса, а также содержит подсказки по использованию функциональных клавиш:

- **F1 - Help** (помощь);
- **F2 - Save** (записать файл);
- **F3 - Open** (вызвать файл на экран);
- **ALT - F9 - Compile** (компилировать файл);
- **F9 - Make** (компилировать файл);
- **F10 - Menu** (выход в основное меню).

Вызов определенной команды в интегрированной среде может быть осуществлен двумя методами:

- нажатием клавиш оперативного вмешательства;
- с помощью вызова функций меню (рис. 30).

Порядок нажатия клавиш при вызове команд следующий: если названия клавиш разделены запятой (например **F10**, **Enter)**, то их следует нажимать последовательно, если они разделены тире (например **Ctrl – F9**), то следует нажать первую клавишу и, не отпуская ее, нажать вторую. Наиболее употребляемые клавиши оперативного вмешательства приведены в строке состояния (рис. 30) .

Меню Турбо – Паскаля содержит набор функций. Для удобства работы однотипные функции сгруппированы в подразделы и разделы. Например, функция ***Color*** находится в разделе ***Options*** в подразделе ***Enviroment*** (условная запись ***Option***/***Enviroment***/***Color***). Для вызова функции меню требуется, во-первых, нажать **F10** для входа в меню и затем, используя **клавиши со стрелками** совместно с клавишей **Enter**, выбрать раздел, подраздел, если он есть, и функцию.

Названия разделов меню, подразделов и функций содержат символы, выделенные другим цветом. Наборные клавиши, соответствующие этим символам, называются **«горячими» клавишами** (**hot keys**) и используются для ускоренного вызова функций меню. Сначала для входа в меню необходимо нажать **Alt – <«горячая» клавиша>**, после чего следует просто нажимать «горячие» клавиши. Например, для вызова функции ***Color*** необходимо нажать **Alt – O, E, C**.

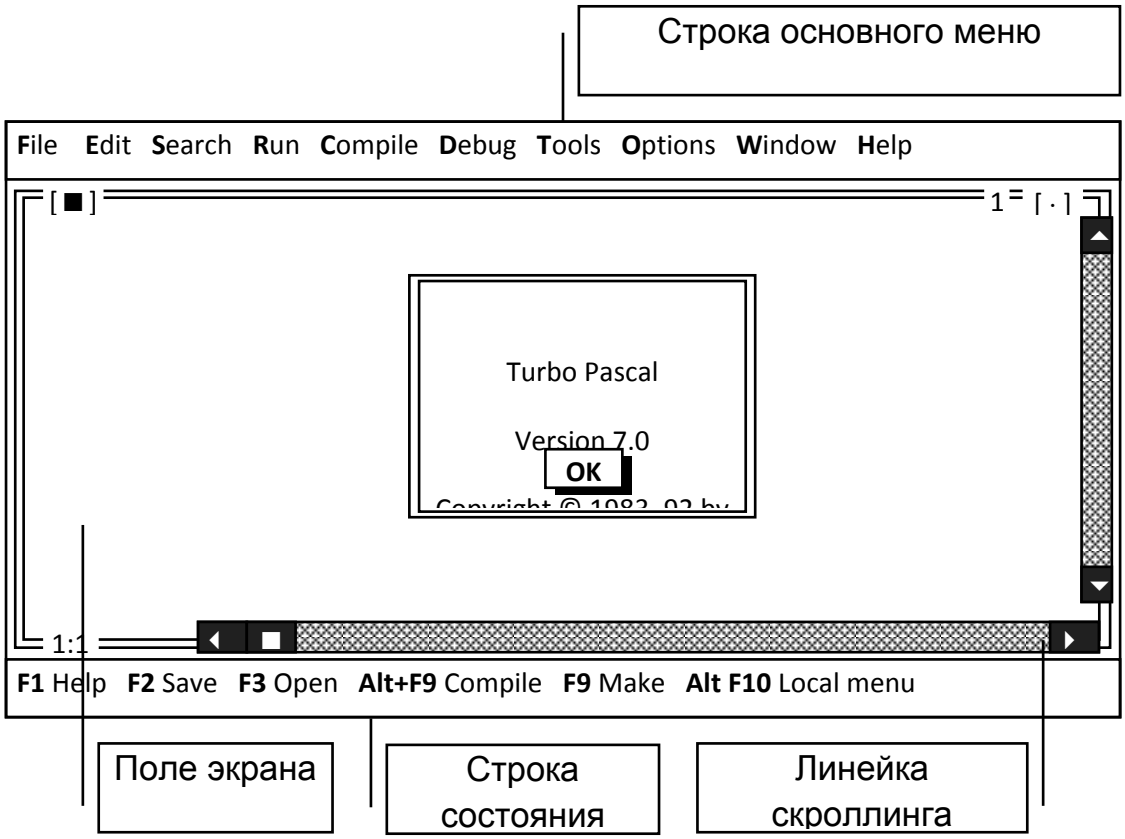


Рис. 30. Основной экран интегрированной среды

3.2. Окна диалога

Некоторым функциям после их вызова необходимо указать дополнительную информацию. В этом случае на экране появится «диалоговое» окно, как, например, при нажатии **Ctrl – Q, A** (рис. 31). Элементами «диалогового» окна могут быть строки ввода текстовой информации, кнопки, списки выбора информации, переключатели с независимой и зависимой фиксацией. Для передвижения внутри «диалогового» окна от элемента к элементу используется **Tab** и **Shift – Tab** для передвижения в обратном направлении [3, 10].

Чтобы вызвать функцию кнопки, необходимо подвести подсветку на кнопку и нажать на **Enter**. Чтобы выбрать информацию из списка, следует, используя **клавиши со стрелками**, подвести подсветку в данную строку и нажать **Enter**. Переключатели, как правило, располагаются группами: в каждой группе несколько переключателей. Перемещение подсветки внутри группы осуществляется **клавишами со стрелками**, «включение» и «выключение» – клавишей **Space (пробел).**

Для передвижения используются также «горячие» клавиши (в том случае, когда нажатие клавиши приводит к вводу текста, необходимо использовать **Alt – <«горячая» клавиша>**), но при переходе на кнопку одновременно вызывается функция данной кнопки, а при переходе на переключатель он автоматически «включается» или «выключается».

Для продолжения выполнения функции меню необходимо перейти на кнопку **«ОК»** и нажать **Enter** или просто нажать **Enter**. Для прерывания функции меню следует перейти на кнопку **«Cancel»** и нажать **Enter** или нажать **Esc**.

Особенности работы с конкретными «диалоговыми» окнами будут рассмотрена ниже.

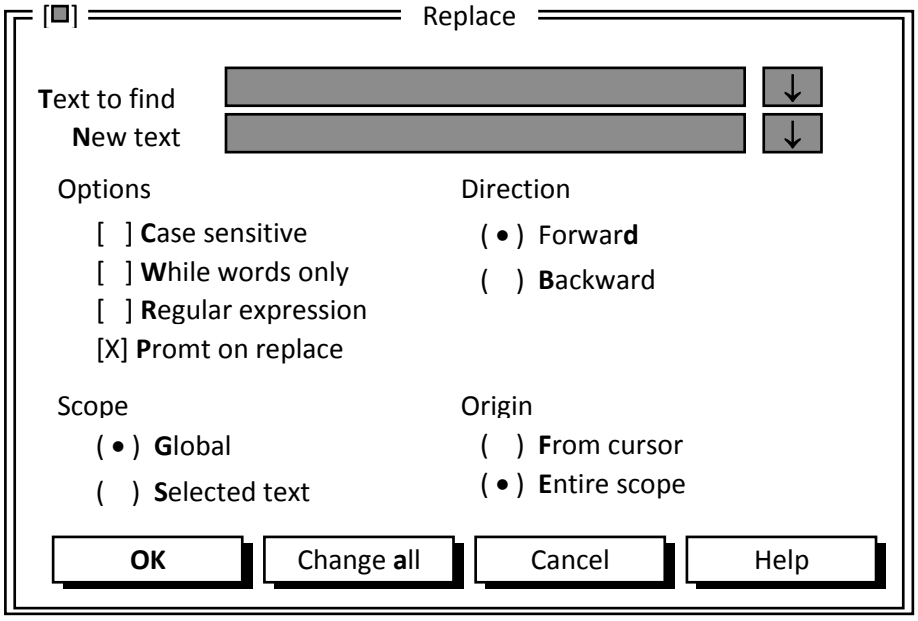


Рис. 31. Диалоговое окно

3.3. Первая программа

Этот раздел кратко описывает последовательность действий оператора-программиста при создании программы: ввести текст программы ⇒ сохранить ⇒ отредактировать ⇒ выполнить.

Чтобы начать работу на Турбо – Паксале, необходимо войти в подраздел, где находятся ваши программы, нажать клавишу **F2**, запустить TURBO – PASCAL. После загрузки на экране появляется основное окно компилятора Турбо – Паскаля (рис. 30).

Создание текста программы. Для создания нового файла необходимо выполнить команду ***New*** меню ***File***. Для этого следует выполнить такую последовательность действий:

– **F10** (выход в главное меню);
– ***File*** **<Enter>**;
– ***New*** **<Enter>** либо нажать клавиши **ALT – F, N**.

В окне редактирования появится окно со стандартным именем файла *NONAMEOO.PAS*. Далее следует набирать текст программы. При наборе можно воспользоваться командами редактора.

Сохранение текста программы. Для сохранения набранного текста программы следует выполнить команду ***Save*** меню ***File***. Для этого необходимо выполнить следующие действия:

– **F10**;
– ***File*** **<Enter>**;
– ***Save*** **<Enter>**.

Также можно воспользоваться горячей клавишей команды ***Save*** – **F2** (информация о «горячих» клавишах на строке подсказки). После выполнения команды ***Save*** на экране появляется окно, в котором требуется ввести имя программы, например

PROGR1.PAS <**Enter**>.

Файл с именем *PROGR1.PAS* будет сформирован в текущем каталоге. А в окне редактирования стандартное имя *NONAMEOO.PAS* будет изменено на текущее.

Выполнение. После сохранения программы можно перейти к шагу выполнения. Для этого необходимы действия:

– **F10** (выход в главное меню);
– ***Run*** **<Enter>**;
– ***Run*** **<Enter>** либо нажать **CTRL – F9** (горячая клавиша команды ***Run***).

Перед стартом команды ***Run*** автоматически проводится компиляция программы, если она не была предварительно выполнена.

При выполнении программы возможны ошибки двух типов:

– ошибки компиляции;
– ошибки времени выполнения.

При этом курсор устанавливается в строку, приведшую к ошибке. После внесения соответствующих изменений можно вновь переходить к шагу выполнения программы.

Просмотр результатов. Просмотр результатов позволяют сделать команды:

– **F10**;
– ***Debug*** **<Enter>**;
– ***User screen*** **<Enter>** либо команда **ALT – F5**.

После просмотра результатов для возвращения в окно Турбо–Паскаля надо нажать **любую клавишу**.

Повторное обращение к программе. Для загрузки в окно редактора текста любой имеющейся программы необходимо выполнить команды:

– **F10**;
– ***File*** **<Enter>**;
– ***Open*** **<Enter>** либо горячая клавиша **F3**.

На экране появляется список всех имеющихся программ текущего каталога. С помощью клавиш **Tab** (табуляция), ↑, ↓ выбрать нужную программу и загрузить её командой **Enter**.

3.4. Главное меню

Главное меню Турбо–Паскаля содержит 10 основных разделов: ***File***, ***Edit***, ***Search***, ***Run***, ***Compile***, ***Debug***, ***Tools***, ***Options***, ***Window***, ***Help*** [3, 10].

Раздел ***File*** (работа с файлами) позволяет осуществлять операции с файлами (создавать, записывать, считывать и т.п.).

Раздел ***Edit*** (редактирование) позволяет выполнять операции с фрагментами текста (копировать, вставлять, удалять фрагмент и т.п.).

Раздел ***Search*** (поиск) позволяет осуществлять поиск фрагментов текста, заменять фрагмент текста, находить ошибки и подпрограммы.

Раздел ***Run*** (выполнить) служит для компиляции и выполнения программы.

Раздел ***Compile*** (компилировать) предназначен для компиляции программы и её записи в оперативную память или на магнитный диск. Следует отметить, что на магнитный диск программа компилируется в случае создания загружаемого файла (с расширением *.EXE*) или модуля (с расширением *.TPU*).

Раздел ***Debug*** (отладка) служит для упрощения процесса отладки пользовательских программ.

Раздел ***Tools*** (инструменты) позволяет задать программы, которые можно запускать не выходя из интегрированной среды.

Раздел ***Options*** (варианты) позволяет изменить некоторые параметры системы Турбо–Паскаль, связанные с конфигурацией компьютера.

Раздел ***Window*** (работа с окнами) позволяет работать с окнами (открывать, закрывать, перемещать, изменять размеры окон и т.п.).

Раздел ***Help*** (помощь) может быть выполнен с помощью зарезервированной клавиши **F1**. Информация в системе ***Help*** снабжена комментариями, что ускоряет поиск нужной темы. Не выходя из окна редактирования, можно установить курсор на любое слово и нажать **Ctrl – F1** – на экране появится информационное сообщение.

File* – работа с файлами.** Команда ***File позволяет загружать с диска и записать на диск файлы, просматривать оглавление дисков, осуществлять временный или окончательный выход из системы Турбо–Паскаль. Рассмотрим более подробно функции из подменю ***File*.**

Функция ***New*** открывает окно редактора для набора новой программы или любого произвольного текста. После подачи команды на экране отображается пустое окно с наименованием *NONAMEOO.PAS*. Если программа находится на диске или дискете, её можно считать, вызвав функцию меню ***Load*** (**F3**).При этом будет открыто «диалоговое» окно для ввода имени файла (рис. 32) [10].

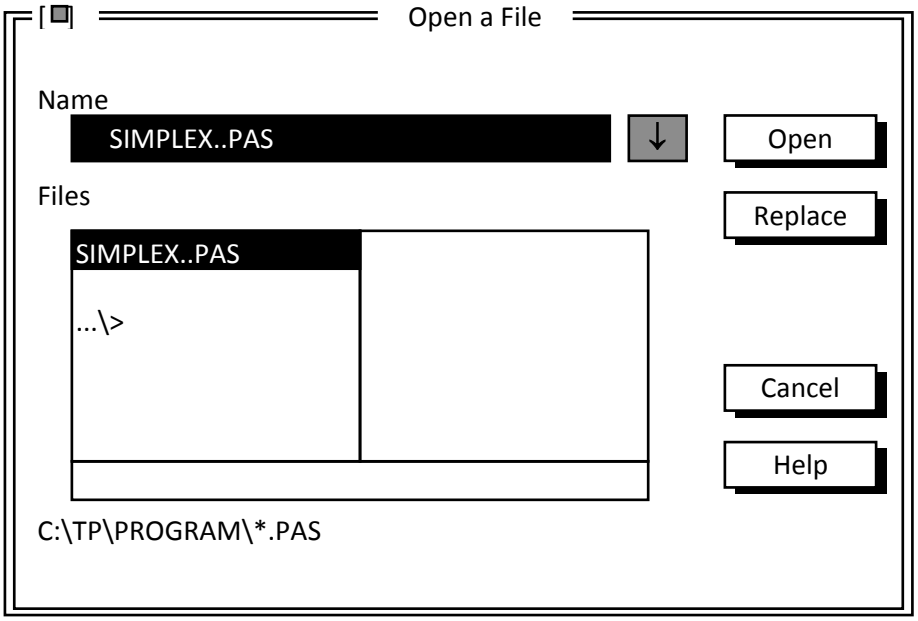


Рис. 32. Диалоговое окно для ввода имени файла

Имя файла можно ввести тремя способами:

а) набрать с клавиатуры (для программ на Паскале расширение указывать необязательно, (например: *matr, simplex* и т.д.), для прочих текстовых файлов требуется указывать имя и расширение (например: *data.txt*, *result*. и т.д.);

б) указать формат просматриваемых файлов (например: **.pas*, **.**, **.* и т.д.), нажать **Enter**, выбрать имя файла из списка файлов текущего каталога;

в) нажать клавишу ↓ и выбрать имя файла из списка уже указывавшихся файлов.

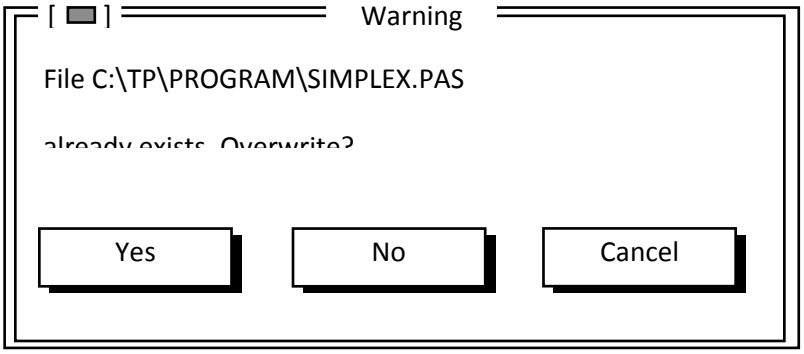
Также файл можно ввести, используя список файлов, находящийся в разделе меню ***File***. Здесь указываются редактировавшиеся, но к данному моменту удаленные из редактора файлы.

Для записи программы под новым именем используется функция ***Save as***. При этом, как и в предыдущем случае, будет открыто **«диалоговое» окно** для ввода имени файла. Для записи программы под прежним именем используется функция ***Save*** (**F2**).Однако при записи только что набранных программ эта функция работает так же, как и

функция ***Save as***. Команду ***Save*** рекомендуется использовать при наборе текста периодически, каждые несколько минут, во избежание потери набранного текста при сбое компьютера.

Если во время записи появится сообщение:

«*File <имя файла> already exists. Overwrite?*»



«*Файл <имя файла> уже существует. Переписать?*»,

то, если находящийся на диске файл представляет собой более раннюю версию вашей программы и он вам не нужен, нажмите – **Y**, если нет, то – **N.**

Функция ***Save all*** записывает все редактировавшиеся файлы.

Функция ***Change dir*** позволяет изменить текущий каталог. После вызова этой функции и появления **«диалогового» окна** установить каталог можно тремя способами:

– ввести с клавиатуры;

– используя «дерево каталогов» (***Dyrectory tree***) и **клавиши передвижения** совместно с **Enter;**

– используя список упоминавшихся каталогов (для вызова этого списка необходимо, находясь в ***Directory name***, нажать ↓).

Для окончания ввода нажмите **Alt – K**. Для отмены ввода – **Esc**. Для восстановления первоначального состояния дерева каталогов используется кнопка ***Revert.***

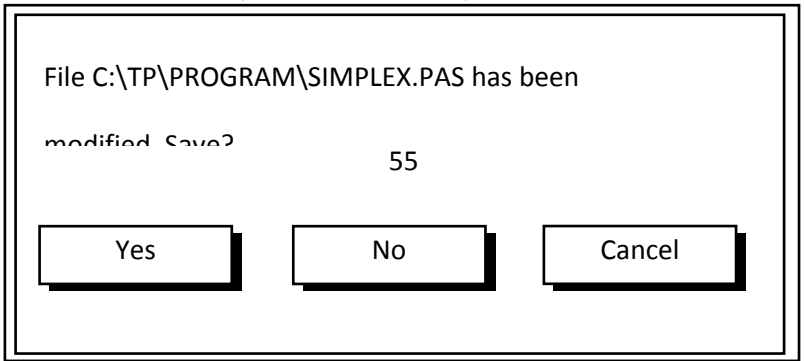
Функция ***Print*** позволяет напечатать текст программы. Функция ***Printer setup*** устанавливает параметры работы принтера.

Функция ***DOS shell*** осуществляет временный выход в DOS. Для возврата в Турбо-Паскаль необходимо завершить все программы, набрать ***Exit*** и нажать **Enter.**

Функция ***Exit*** (**Alt – X**)- выход из Турбо-Паскаля. Если, выходя из Турбо – Паскаля, вы не записали редактируемые файлы, то появится сообщение:

«*Файл <имя файла> был изменен. Записать?*»

«Если «да», нажмите – **Y**, если «нет» – **N».**



Работа с окнами (*Window*). ***Окно*** – это один из видимых элементов интегрированной среды Турбо – Паскаля, в котором может отображаться различная информация: редактируемый текст, значения переменных и т.д. Окно имеет заголовок и может иметь порядковый номер от 1 до 9. Окно, с которым в данный момент работает программист, называется **активным** и имеет двойную рамку [3, 10].

Чтобы перейти из одного окна в другое, необходимо нажать **Alt – <номер окна>,** при этом окно с указанным номером станет активным. Эту же операцию можно осуществить, нажав **Alt – 0** (функция меню ***Window*/*List***), выбрав, используя **стрелки**, необходимый заголовок окна и нажав **Enter**.

Интегрированная среда запоминает порядок вашей работы с окнами. Нажав **Shift – F6** (функция меню ***Window*/*Previos***), вы перейдете в окно, в котором только что работали, нажав **F6** (функция меню ***Window*/*Next***), вы перейдете в окно, к которому дольше всего не обращались.

Можно менять размер окон. После нажатия **Ctrl – F5** (функция меню ***Window*/*Size*/*Move***) можно, нажимая **клавиши со стрелками**, **Home, End, PgUp, PgDn, Shift – <стрелки>** и затем **Enter**, изменить размер и положение окна. Клавишей **F5** (функция меню ***Window*/*Zoom***) можно увеличить размер окна до максимально возможного размера. Повторное нажатие **F5** уменьшает окно до первоначальных размеров. Функция меню ***Window*/*Title*** располагает окна так, чтобы полностью было видно каждое окно. Функция меню **Window/Cascad** располагает окна друг за другом так, чтобы был виден заголовок и номер каждого окна.

Комбинация клавиш **Alt – F3** (функция меню ***Window*/*Close***) закрывает активное окно. Эту же операцию можно осуществить, нажав **Alt – 0**, выбрав, используя стрелки, необходимый заголовок окна, и нажав **Del**. Функция меню ***Window*/*Close all*** закрывает все окна.

Функция меню ***Window*/*Refresh display*** восстанавливает (перерисовывает) экран интегрированной среды.

Редактор (*Edit, Search*). **Редактор** – это программа, предназначенная для набора текстов и исправления ошибок в них. В данном разделе рассмотрены возможности редактора языка Турбо – Паскаль.

В табл. 4 приведены функции некоторых клавиш, используемых при работе с редактором.

Таблица 4

Клавиша (и)	Выполняемая операция
1	2
Передвижение курсора	

Ctrl–S или **стрелка влево**	на символ влево
Ctrl–D или **стрелка вправо**	на символ вправо
Ctrl–A или **Ctrl–стрелка влево**	на слово влево
Ctrl–F или **Ctrl–стрелка вправо**	на слово вправо
Ctrl–E или **стрелка вверх**	на строку вверх
Ctrl–X или **стрелка вниз**	на строку вниз
Home	в начало строки
End	в конец строки
Ctrl–R или **PgUp**	на страницу вверх
Ctrl–C или **PgDn**	на страницу вниз
Ctrl–Home	к верху окна
Ctrl–End	к низу окна
Ctrl–PgUp	в начало текста
Ctrl–PgDn	в конец текста
Ctrl–Q,B	в начало блока
Ctrl–Q,K	в конец блока
Del или **Ctrl–G**	удаление символа
BackSpace или **Ctrl–H**	удаление символа слева
Ctrl–Y	удаление строки
Ctrl–Q,Y	удаление до конца строки
Ctrl–T	слова справа
Восстановление	
Alt–BackSpace	восстановление текста
Работа с блоками	
Shift–<клавиши передвижения>	выделение блока
Ctrl–K,B	отметка начала блока
Ctrl–K,K	отметка конца блока

Продолжение табл. 4.

1	2
Ctrl–K,T	выделить слово
Ctrl–K,C	копировать блок
Ctrl–K,V	передвинуть блок
Ctrl–K,H	снятие/восстановление выделения
Ctrl–K,Y	удаление блока
Ctrl–K,W	считывание блока из файла
Ctrl–K,R	запись блока в файл
Ctrl–K,P	печать блока
Ctrl–K,I	смещение текста в блоке вправо
Ctrl–K,U	смещение текста в блоке влево

Поиск и замена текста	
Ctrl–Q,F	поиск текста
Ctrl–L	повторный поиск текста
Ctrl–Q,A	замена текста
Прочие команды	
Ctrl–W	смещение текста вверх
Ctrl–Z	смещение текста вниз
Ctrl–K,n (n = 0..9)	установка метки
Ctrl–Q,n (n = 0..9)	переход на метку
Tab или **Ctrl–I**	табуляция
Enter	раздвижение, перенос строк
Ctrl–N	раздвижение строк
Ctrl–P,Ctrl–<символ>	ввод управляющего символа
Ctrl–F1	контекстная справка
Esc	отмена команды
Переключение режимов редактора	
Ins или **Ctrl–V**	вставки/замещения
Ctrl–O,I	при нажатии Enter курсор переводится в следующую строку и помещается в начало строки/под первый символ в строке
Ctrl–O,T	Установка/отмена фиксированной табуляции
Для получения справки о прочих режимах редактора необходимо войти в **Option**/**Enviroment**/**Editor** и использовать клавишу F1	

Из перечисленных в таблице функций только поиск и замена требуют ввода дополнительной информации, поэтому рассмотрим их применение на примере. При нажатии **Ctrl – Q, A** на экране появляется диалоговое окно (рис. 31). Для выполнения операции требуется ввести заменяемый текст в строке ***Text to find***, заменяющий текст в строке ***New text***, установить, если необходимо, режимы поиска и замены:

- ***Case sensitive*** – отличие строчных и прописных букв;
- ***Whole words only*** – поиск отдельных слов;
- ***Regular expression*** – поиск математических выражений;
- ***Promt on replace*** – запрос подтверждения при замене;
- ***Forward*** – поиск сверху вниз;
- ***Backward*** – поиск снизу вверх;
- ***Global*** – поиск во всем тексте;
- ***Selected text*** – поиск в выделенном тексте (блоке);
- ***From cursor*** – поиск от курсора;
- ***Entire scope*** – поиск от начала текста.

Для однократной замены подведите клавишей **Tab** подсветку на кнопку **«ОК»** и нажмите **Enter** или нажмите **Alt – K**. Чтобы поиск продолжался автоматически до тех пор, пока искомая строка встречается в тексте, используйте кнопку **«Change all»** или нажмите **Alt – A**.

Функции поиска и замены можно вызвать также, используя меню ***Search***. Здесь же содержатся функции:

– ***Go to line number*** – переход в строку с данным номером;

– ***Show last compiler error*** – показать последнюю ошибку компилятора (**Ctrl – Q, W**);

– ***Find error*** – нахождение в тексте программы ошибки, возникшей при выполнении программы;

– ***Find procedure –*** нахождение в тексте программы процедуры с данным именем.

В разделе меню ***Edit*** содержатся функции для копирования блока из одного окна в другое. Порядок копирования следующий:

а) выделить блок в одном окне;

б) вызвать функцию ***Edit/Cut*** – перенести выделенный блок в специальное окно, называемое ***ClipBoard***, – или ***Edit/Copy*** –скопировать выделенный блок в ***ClipBoard***;

в) перейти в другое окно и подвести курсор к месту вставки;

г) вызвать функцию ***Edit/Paste***.

Другие функции раздела меню ***Edit***:

– ***Undo, Redo*** - восстановление удаленного текста;
– ***Clear*** - удаление блока (**Ctrl – Q, Y**);
– ***Show ClipBoard*** - показать ***ClipBoard*** в отдельном окне.

Компиляция и выполнение программы (*Compile*). Персональный компьютер PC, как и большинство микрокомпьютеров, имеет процессор, который представляет собой его рабочий механизм. Программист, как правило, составляет программу на алгоритмических языках высокого уровня, а компьютер обрабатывает программы только в машинных кодах. Для перевода программы в машинные коды служит компилятор. Компилятор Турбо – Паскаля транслирует (или переводит) программу, написанную на Паскале, в команды, которые могут быть восприняты компьютером. Компилятор, таким образом, является программой, пересылающей данные: она считывает текст вашей программы и записывает его на соответствующем машинном коде.

Компиляция начинается нажатием клавиш **Alt – F9**. Если во время компиляции обнаруживается ошибка, компиляция прекращается, курсор подводится к месту ошибки и указывается номер и тип ошибки. Если при этом нажать **F1** ,то можно получить информацию о данном виде ошибки на английском языке. Выход из справки – **Esc**. По завершению компиляции на экран выдается сообщение (рис. 33):

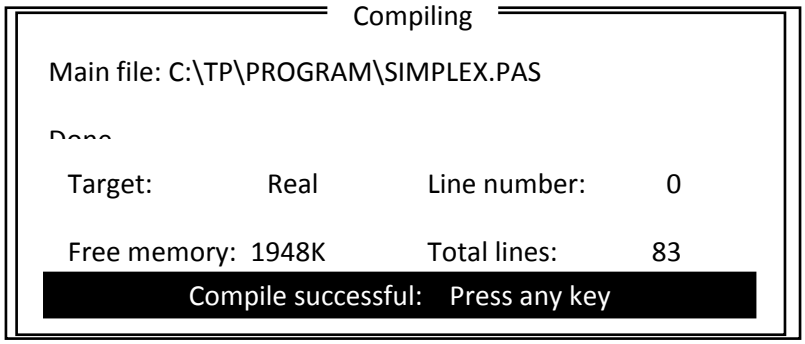


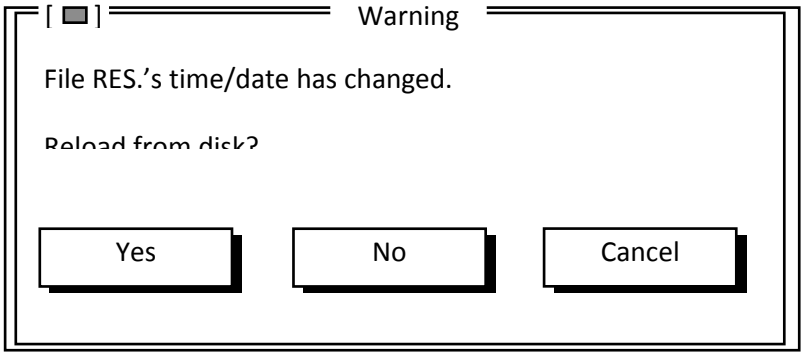
Рис. 33. Окно сообщения о завершении компиляции

Для запуска программы необходимо нажать **Ctrl – F9**.

Если результаты расчетов вашей программы выводятся на экран, то по окончании работы программы их можно просмотреть, нажав **Alt – F5**. Выход из режима просмотра – **Esc**. Если результаты расчетов выводятся в файл, то по окончании работы программы этот файл можно считать с диска или дискеты. Если после этого вы вновь запустите программу, то по окончании расчетов появится сообщение:

«*Файл был изменен. Перезагрузить?*»

«Если «да», нажмите – **Y**, если «нет» – **N**».



Отладка программы (*Run, Debug*). В программе могут быть разнообразные ошибки: синтаксические, семантические и логические. Ошибки на этапе компиляции (или синтаксические ошибки) возникают в том случае, если написанные вами операторы Паскаля не удовлетворяют требованиям Паскаля. Другой возможный тип ошибок – это ошибки этапа выполнения (или семантические ошибки). Это происходит в том случае, если при выполнении программы предпринимается попытка выполнить недопустимое действие: открыть несуществующий файл, например, или разделить число на ноль. В этом случае Турбо – Паскаль печатает на экране сообщение об ошибке, которое выглядит примерно так:

Runtime error at Seg:Ofs (Ошибка этапа выполнения в Сегменте: Смещении).

Наконец, программа может содержать логические ошибки, связанные с неправильным составлением алгоритма. Этот тип ошибок наиболее труден для обнаружения, и поэтому он может быть одной из основных причин возникновения необходимости использования отладчика.

Перед применением отладчика программа должна быть откомпилирована. Затем, нажимая **F7** или **F8**, можно начать выполнение программы «по шагам». При этом на экране появится подсветка, которая при нажатии **F7** или **F8** будет перемещаться от оператора к оператору в той же последовательности, как если бы программа выполнялась в режиме счета. Отличие клавиш **F7** и **F8** заключается в том, что при нажатии **F7** будет выполнятся трассировка процедур и функций, а при **F8** вызовы процедур и функций выполняются как один шаг.

Если требуется пропустить выполнение циклов и других малоинтересных частей программы и сразу перейти к той строке, где вы хотите начать отладку, то необходимо подвести курсор в эту строку и нажать **F4.**

Точки останова. Отдельные строки программы можно пометить как точки останова. Когда вы запускаете программу, и она попадает в точку останова, ее выполнение приостанавливается, и на экран выводится текст программы в точке останова. При этом вы можете проверить значение переменных, начать трассировку (**F7, F8**) или запустить программу, пока она не достигнет следующей точки останова (**Ctrl – F9**).

Чтобы установить (или убрать) точку останова, требуется подвести курсор в требуемую строку и нажать **Ctrl – F8**.Установить точку останова можно также, вызвав функцию ***Debug*/*Add Breakpoint***. При этом в «диалоговом» окне требуется указать в строке ***Condition*** – логическое выражение – условие остановки (используется при отладке циклов ***Repeat Until*** и ***While Do***), ***Path count*** – число, сколько раз будет пропущена точка останова перед остановкой на ней (используется при отладке цикла ***For***), ***File name*** – имя файла, в котором поставлена строка останова, ***Line number*** – номер строки. Работать с уже установленными точками останова можно используя функцию меню ***Debug*/*Breakpoints***. При этом в появившемся «диалоговом» окне можно, используя клавиши со стрелками, выбрать необходимую точку останова и, вызывая функции кнопок:

- изменить параметры точки останова (кнопка **Edit**);
- удалить точку останова (кнопка **Delete** или клавиша **Del**);
- перейти на выбранную точку останова (кнопка **View**);
- удалить все точки останова (кнопка **Clear all**).

В добавление к использованию в программе точек останова работающую программу в любой момент можно приостановить либо

остановить, нажав **Ctrl – Break**. Как правило, однократное нажатие **Ctrl – Break** приводит к приостановке программы с возможностью дальнейшей отладки, а двукратное – к остановке.

Окно просмотра. Турбо – Паскаль дает программисту возможность во время отладки просматривать и изменять текущие значения переменных. Для просмотра переменных служит окно просмотра.

Окно просмотра можно открыть двумя способами: вызовом функции меню ***Debug*/*Watch*** или первым добавлением переменной, структуры данных или выражения в окно просмотра. При этом окно просмотра помещается в нижней части экрана, и его размеры и положение на экране можно изменить, используя клавиши работы с окнами.

Добавление переменных, структур данных и выражений может быть осуществлено нажатием **Ctrl – F7** (или вызовом функции меню ***Debug*/*Add watch***).

При этом появится «диалоговое» окно. Необходимый текст можно ввести с клавиатуры или, нажав ↓, выбрать его из списка. Также для ускорения добавления можно подвести курсор на необходимую переменную в редакторе, нажать **Ctrl – F7** и, если не весь требуемый текст появился в окне, нажимая ™, добиться появления всего текста.

Добавленные переменные, структуры данных и выражения могут быть изменены или удалены. Для этого необходимо войти в окно просмотра, выбрать, используя клавиши со стрелками, требуемую строку и для изменения нажать на **Enter** или для удаления – на **Del.**

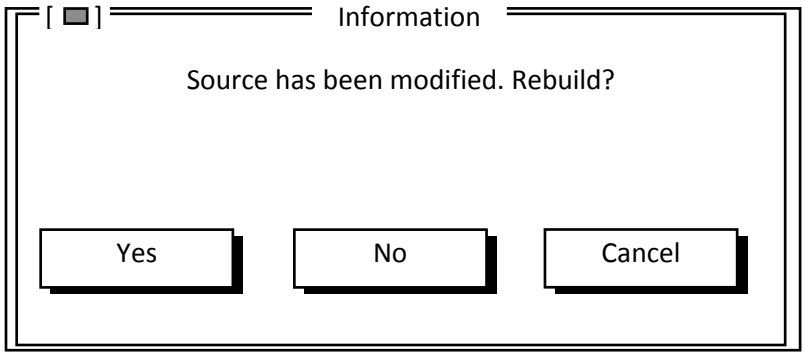
Кроме окна просмотра в отладчике можно использовать специальные окна.

Окно «стек вызова» (функция ***Debug*/*Call Stack*** или **Ctrl – F3**).При каждом вызове процедуры или функции Турбо – Паскаль запоминает вызов и передаваемые ей параметры. Когда вы выходите из этой процедуры или функции, то Паскаль про этот вызов «забывает». Это называется стеком вызова. Используя окно «стек вызова», можно просмотреть список вызовов процедур и функций и переданные им фактические параметры. Окно «стек вызова» несет также другую важную функцию: оно позволяет выполнять обратный просмотр последовательности вызовов. Когда на экране появится стек вызова, самый верхний вызов подсвечивается. Для перемещения вверх и вниз по стеку можно использовать клавиши передвижения курсора. Если вы нажмете клавишу **Enter**, то переместитесь к последней активной точке в программе или подпрограмме.

Окно просмотра (функция ***Debug*/*Output***) позволяет просматривать всю информацию, выводимую программой на экран.

Завершение сеанса отладки осуществляется нажатием **Ctrl – F2** (функция ***Run/Program Reset***).С другой стороны, если вы во время отладки модифицировали какую–либо часть программы, то при нажатии одной из клавиш выполнения: **F7, F8, Ctrl – F9** и т.д. – вы получите сообщение:

«*Source modified, rebuild? (Y/N)*»



«Исходный код изменен, сформировать заново?».

Если вы нажмете **Y** (да), то Турбо – Паскаль выполнит повторное создание вашей программы и начнет отладку сначала. Если вы нажмете **N** (нет), то Турбо – Паскаль, предполагая, что вы знаете, что делаете, продолжит сеанс отладки (изменения не будут учтены).

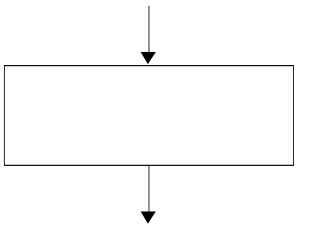
4. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ТУРБО–ПАСКАЛЬ

4.1. АЛГОРИТМЫ

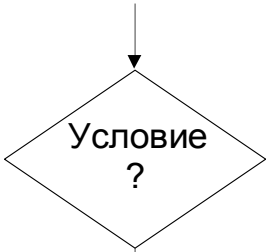
Практически решение любой задачи на ЭВМ начинают с составления алгоритма.

Алгоритм – это точно определенная последовательность элементарных действий, необходимых для решения задачи. Алгоритм содержит несколько шагов, которые должны выполняться в определенной последовательности. Каждый шаг алгоритма может состоять из одной или нескольких простых операций [2,13,17].

Наиболее наглядным способом описания алгоритмов является описание его в виде блок–схем. При этом алгоритм представляется последовательностью блоков, выполняющих определенные функции, и связей между ними. Внутри блоков указывается информация, которая характеризует, какие функции выполняют данные блоки. На блок–схеме каждый шаг алгоритма обозначается специальной геометрической фигурой, а внутри записываются простые операции. Для изображения блок–схем программ используются следующие наиболее часто встречающиеся обозначения.



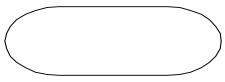
Выполнение операции или групп операций



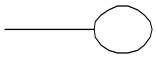
Проверка условия

Выполнение циклических операций

Ввод исходных данных и вывод результатов



Начало, конец

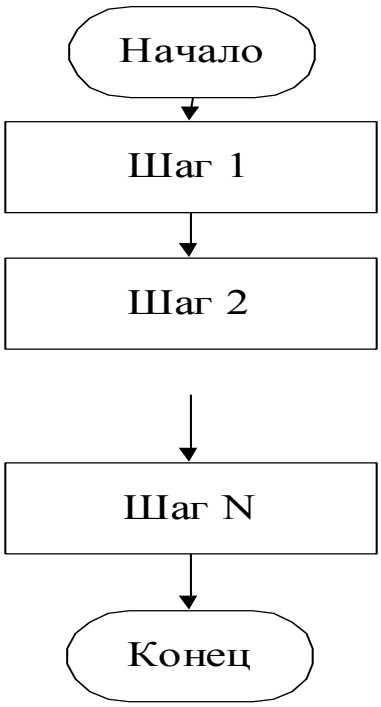


Указание связи между прерванными линиями, связывающими блоки

Существует несколько типов алгоритмов.

4.1.1. ЛИНЕЙНЫЙ АЛГОРИТМ

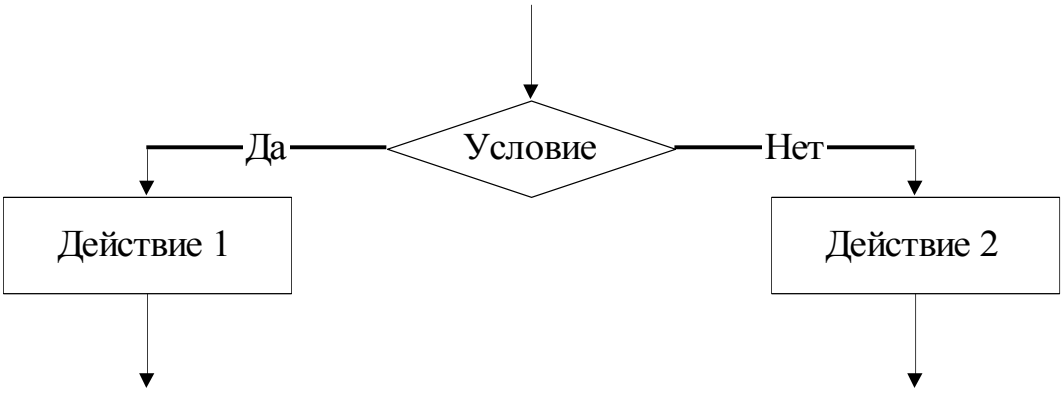
Линейным называется алгоритм, который содержит N шагов, и все шаги выполняются последовательно друг за другом без разветвлений.



4.1.2. РАЗВЕТВЛЯЮЩИЙСЯ АЛГОРИТМ

Разветвляющимся называется такой алгоритм, в котором последовательность выполнения операций (шагов) изменяется в зависимости от некоторых условий и продолжается в различных направлениях.

Если условие верно, то выполняется действие 1, в противном случае– действие 2.



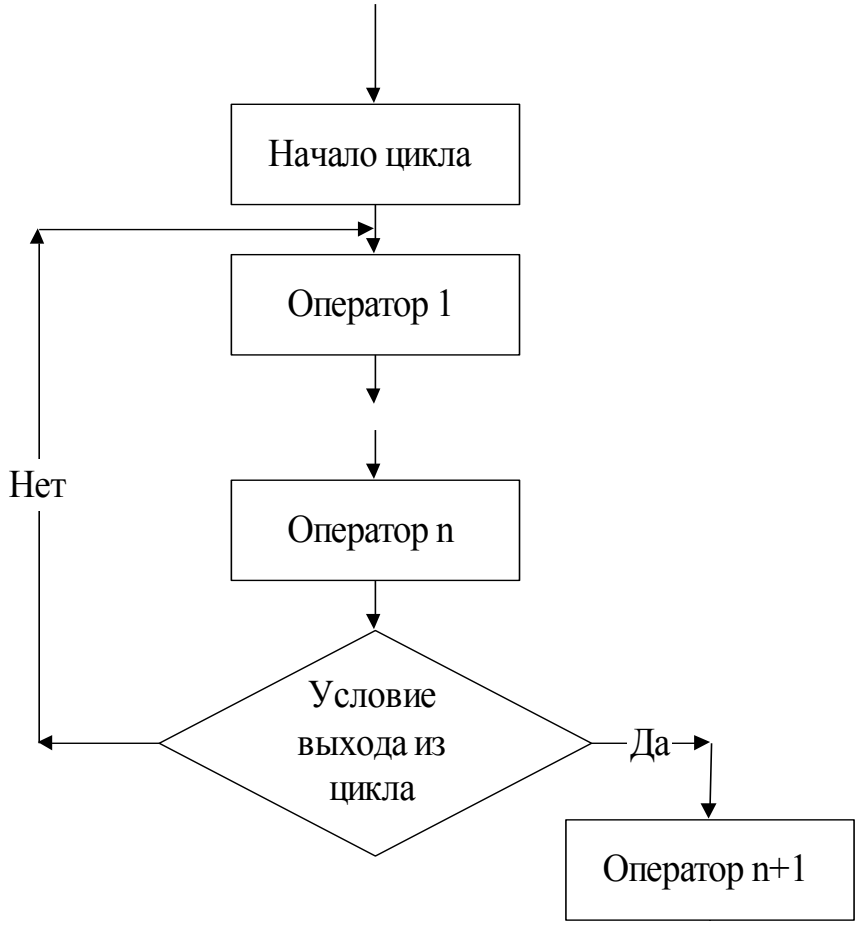
4.1.3. ЦИКЛИЧЕСКИЙ АЛГОРИТМ

Циклическим называется такой алгоритм, который содержит многократно повторяющийся участок в зависимости от заданной величины (параметра цикла). Ниже приведена блок–схема циклического алгоритма с проверкой условия для параметра в конце цикла.

4.2. Введение в турбо–Паскаль

Любая ЭВМ обрабатывает информацию по программам.

Программой называется алгоритм, записанный на языке программирования. Существуют различные языки программирования, и



каждый из них предназначен для решения определенного круга задач. Для решения вычислительных задач используются операторные языки программирования. К таким языкам относятся FORTRAN, PASCAL, BASIC и некоторые другие. С помощью этих языков удобно решать математические, физические, инженерные задачи. Но программы для решения нечисловых задач лучше писать на других языках [1,11].

В операторных языках каждый элемент алгоритма записывается с помощью какого–либо оператора. Каждый оператор выполняет определенное действие (определенную операцию) в программе.

Алгоритмический язык Паскаль появился в 1970 г. Автор языка Паскаль – Никлаус Вирт (профессор, директор института Информатики в Швейцарии). Язык назван в честь французского ученого Блеза Паскаля, разработавшего одно из первых суммирующих устройств.

Язык программирования Паскаль является достаточно простым и компактным языком, который широко применяется в мини–ЭВМ и ПЭВМ. Конструкции языка позволяют работать не только со стандартными типами данных, но и предоставляют пользователю возможность выполнять операции над файлами, множествами и записями, использовать динамические структуры данных [10–17].

4.2.1. Символы, простейшие конструкции языка

В языке Турбо-Паскаль используются следующие символы.

Буквы: 26 латинских букв (прописных и строчных).

Цифры: 0, 1, ..., 9.

.Специальные символы:

: + - □ / < > = ( ) [ ] / , ; _ { }

Ключевые слова: **AND** – и, **ARRAY** – массив, **BEGIN** – начало, **CASE** – вариант, **CONST** – константа, **DIV** – деление нацело, **DO** – выполнять, **DOWNTO** – уменьшать до, **END** – конец, **FILE** – файл, **FOR** – для, **FUNCTION** – функция, **GOTO** – перейти на, **IF** – если, **IN** – включение, **LABEL** – метка, **MOD** – модуль, **NIL** – отсутствие указателя, **NOT** – не, **OF** – из, **OR** – или, **PACKED** – упакованный, **PROCEDURE** – процедура, **PROGRAM** – программа, **RECORD** – запись, **REPEAT** – повторять , **THEN** – то, **TO** – до, **TYPE** – тип, **UNTIL** – до, **VAR** – переменная, **WHILE** – пока, **WITH** – с.

Знаки операций:

- арифметических: + (сложение), – (вычитание), □ (умножение), / (деление), DIV (деление нацело с отбрасыванием остатка), MOD (нахождение остатка от деления нацело);
- отношения: > (больше), < (меньше), <= (меньше или равно), >= (больше или равно), = (равно), <> (не равно);
- логических: NOT (отрицание), OR (логическое сложение), AND (логическое умножение).

Имена (идентификаторы). Для обозначения различных объектов в языке Паскаль используются их имена или идентификаторы.

Имя (идентификатор) в Паскале – это произвольная последовательность букв и цифр, начинающаяся с буквы. Русские буквы использовать нельзя:

A X12 UM1 TY134B.

Если необходимо разделить имя, можно использовать символ подчеркивания:

DATE_27_sep A_B.

Длина идентификатора может быть любой, но значимыми являются только первые 63 символа.

Константы. Константами называются параметры программы, значения которых не меняются в процессе её выполнения.

В Турбо–Паскале используются три вида констант: 1) числовые, 2) символьные и строковые, 3) логические.

1). Числовые константы. В Паскале используются целые и вещественные (действительные) числа. ***Целые*** числа имеют обычный арифметический вид (знак, цифры):

0, 1, +100, –12.

Диапазон изменения целых чисел зависит от типа ЭВМ.

Вещественные числа имеют две формы представления: в виде обычной десятичной дроби с фиксированной точкой:

2.75, –11.2

и числа с плавающей точкой, представленные в форме: mEp, где m – мантисса, E – признак записи числа, p – порядок числа;

$1.2 \cdot 10^{-5}$ соответствует 1.2E–5.

Представление чисел

целое	с фиксированной точкой	с плавающей точкой
–257	–257.0	–2.57E2
3	3.0	3.0E0

Константы с фиксированной точкой обязательно должны содержать как целую, так и дробную часть: 2.0; 0.5.

2) Символьные константы и константы–строки.

Символьная константа – это символ, заключенный в апостроф: 'A', '!', '22'.

Строковая константа – это последовательность символов, заключенных в апострофы:

'MASSA', 'РЕАКЦИЯ', 'C2H6'.

3) ***Логические*** константы – принимают два значения: True (истина) и False (ложь).

Переменные. Переменными называются параметры программы, значения которых могут изменяться в процессе ее выполнения. Переменная в Паскале имеет **имя** и **тип.**

Тип:

- real – вещественный 1.5; 7.32E-5;
- integer – целый 5; -10;
- boolean – логический true, false;
- char – литерный 'a'.

Различают простые переменные и переменные с индексом, в записи которых указывается имя массива и индекс этого элемента.

Например: VOL, X, A, B[1], P[1,3], C[1].

Комментарии. Служат для пояснений в программе. Комментарии – это любой текст, заключенный в { } или (□ □). Например:

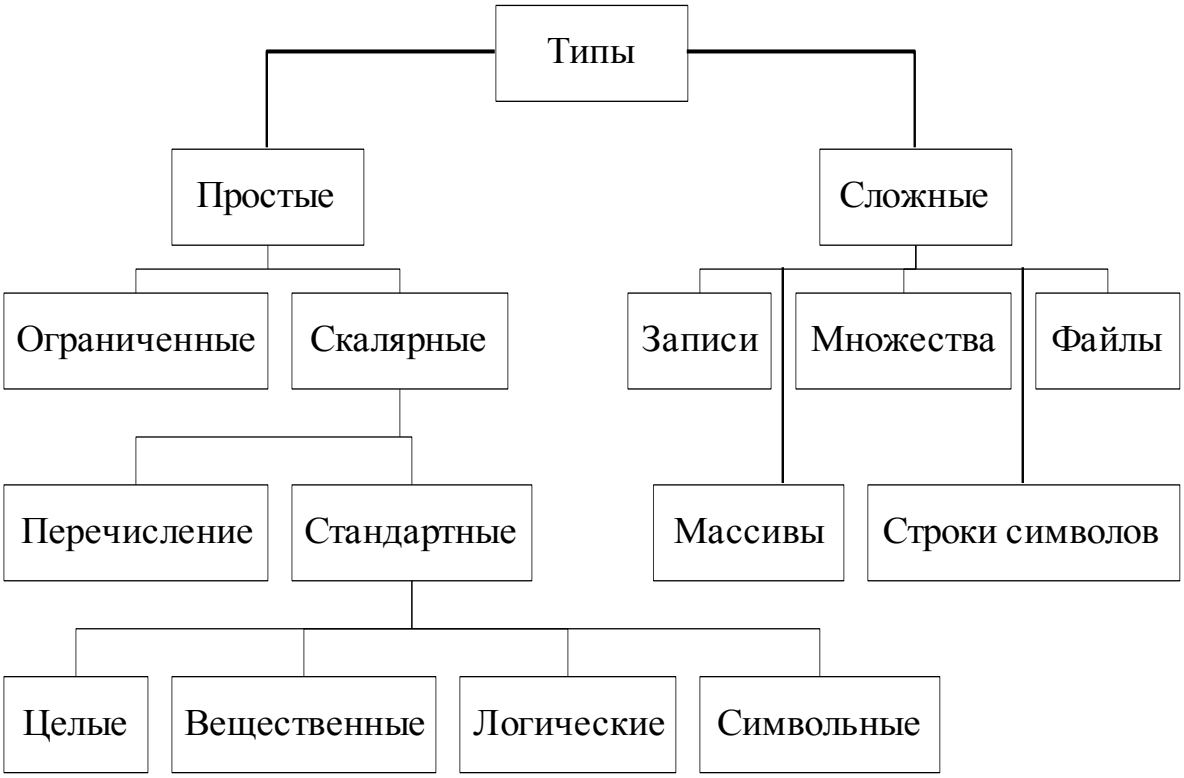
{ Расчет суммы } или (□Расчет суммы □).

Комментарии можно помещать в **любом месте** программы, они не воспринимаются и не обрабатываются ЭВМ.

4.2.2. Типы данных

В языке Турбо–Паскаль под типом понимается множество значений, которые может принимать переменная, а также совокупность операций, которые можно выполнять с этими значениями.

В Турбо–Паскале можно выделить группы простых и сложных типов. Основные типы переменных в Турбо–Паскале можно представить в виде следующей схемы:



4.2.2.1. Простые типы данных

В Турбо–Паскале имеются следующие группы простых типов:

- целый тип (integer);
- вещественный тип (real);
- символьный тип (char);
- логический тип (boolean);
- перечисляемый тип;
- ограниченный тип (тип–диапазон).

Среди типов, используемых в языке, есть стандартные (предопределенные) и определяемые программистом.

К стандартным типам относятся целый, вещественный, логический и символьный типы. Все другие используемые типы данных (нестандартные) должны быть определены в разделе объявления типов, который начинается словом **type**, за которым следует имя типа и список его значений:

type <имя типа>=<определение типа>.

Целый тип. Обеспечивает задание целых чисел. В Турбо–Паскале имеется пять стандартных типов целых чисел. Характеристики этих типов приведены в табл. 5.

Таблица 5

Целые типы данных

ТИП	ДИАПАЗОН	ФОРМАТ	РАЗМЕР В БАЙТАХ
Shortint	–128 ÷ 127	Знаковый	1
Integer	–32768 ÷ 32767	Знаковый	2
Longint	–2147483648 ÷2147483647	Знаковый	4
Byte	0 ÷ 255	Беззнаковый	1
Word	0 ÷ 65535	Беззнаковый	2

Над переменными целого типа определены следующие арифметические операции: сложение (+), вычитание (–), умножение (□), деление (/); div – деление нацело, mod – вычисление остатка от целочисленного значения. Результат выполнения этих операций будет целого типа, кроме операции деления, результат которой всегда вещественного типа. Например:

7 div 2 = 3; 7 mod 3 = 1; (–7) div 2 = –3; 14 mod 3 = 2; 3 div 5 = 0;

8/4 = 2.0; 12/3=4.0.

Вещественный тип. В отличие от стандартного языка Паскаль, где определен только один вещественный тип – real, в Турбо–Паскале имеется пять стандартных вещественных типов (табл.6).

Типы Single, Double, Extended и Comp можно использовать в программах только при наличии арифметического сопроцессора.

Таблица 6

Вещественные типы данных

ТИП	ДИАПАЗОН	ФОРМАТ (точнось)	РАЗМЕР В БАЙТАХ
Real	2.9E–39 ÷ 1.7E+38	11–12 знаков	6

Single	1.5E–45 ÷ 3.4E+38	7–8 знаков	4
Double	5.0E–324 ÷ 1.7E+308	15–16 знаков	8
Extended	3.4E–4932 ÷ 1.1E+4932	9–20 знаков	10
Comp	–9E18 ÷ 9E18	19–20 знаков	8

Символьный тип. Стандартный символьный тип char определяется множеством значений кодовой таблицы ПЭВМ. Все символы клавиатуры упорядочены (пронумерованы) и каждая литера имеет свой порядковый номер. Над символьными значениями не предусмотрены никакие операции, но они могут сравниваться, участвовать в чтении, печати и в операциях присваивания. Например:

```
var C1,C2: char;
 . . . . .
C1:='A';
C2:='Z';
if C1<C2 then C1:='Z' else C2:='A';
. . . .
```

Логический тип. Стандартный логический тип Boolean (размер – 1 байт) – это тип данных, любой элемент которого может принимать только два значения: True и False.

Над логическими переменными можно осуществлять логические операции:

- and (и) – логическое умножение;
- or (или) – логическое сложение;
- not (не) – логическое отрицание.

Перечисляемый тип. Этот тип задается перечислением всех значений, которые может принимать переменная данного типа. Описание этих переменных имеет вид:

type <имя типа> = (список значений);

var <имя переменной> : <имя типа>;

или

var <имя переменной> : <список значений>;

Например:

type God=(Zima, Vesna, Leto, Osen);
var A1,A2:God;

или

var A1,A2: (Zima, Vesna, Leto, Osen);

Здесь A1 и A2 переменные типа «перечисление», которые могут принимать любые из заданных значений.

Для переменных перечисляемого типа применимы операции отношения: <, >, >=, <=, =, если оба компонента отношения имеют один тип. Описание типа переменной одновременно упорядочивает ее значения, так: Zima < Vesna < Leto < Osen.

Ограниченный тип (тип–диапазон). Этот тип позволяет задавать две константы, определяющие границы диапазона значений для данной переменной. Описание имеет следующий вид:

type <имя типа> = Min .. Max;

var <имя переменной> : <имя типа>;

или

var <имя переменной> : Min .. Max;

Например:

var P:1..10.

Здесь Min и Max – соответственно константы, определяющие левую и правую границы значений, которые может принимать ограниченная переменная. Обе константы должны принадлежать одному из стандартных типов, кроме real. Обязательно: Min < Max.

К значениям ограниченного типа применимы все операции и функции, которые определены над значениями базового типа.

4.2.3.Структура программы

Общую структуру программы на Турбо–Паскале для простых алгоритмов можно представить:

Program <имя>;
<раздел описаний>

label <список меток>;

const <список констант>;

type <список типов>;

var <список переменных>;

procedure, function <список процедур, функций>;

Begin

<тело программы>

End.

Раздел описаний.

1) Раздел меток (label).

label 5, 10, M1, M2; { список меток }.

Метка может содержать цифры от 0 до 9999, но не может иметь более четырех знаков.

Например:

10: A:=A+1.

2). Раздел констант (const).

const a=2.56; R=1.198; P=1.75E+2;

const f:real=–0.5;

Изменять значения простых констант в программе нельзя.

3). Раздел типов.

В этом разделе описываются имена типов переменных, отличные от стандартных. Например, массивы:

type mas= **array** [1..5] **of** real;

4). Раздел описания переменных.

Каждая переменная в программе должна быть описана в разделе описания переменных:

var <переменная>:<тип;

Например:

```
var a,cl,SK:real;
    P1,P2: char;
    P: array[1.5] of real; {описание массива}
    b: boolean;
    a,x: integer;
```

5). Раздел операторов.

Тело программы начинается словом **Begin** и заканчивается словом **End** с точкой, которая является признаком конца программы. Раздел операторов – выполняемая часть программы, которая записывается в свободной форме. Операторы отделяются друг от друга точкой с запятой. Допускается располагать несколько операторов в одной строке, а также переносить с одной строки на другую части описаний или операторов (но без разделения ключевых слов и идентификаторов). Пробелы допускаются

в любом месте программы и в неограниченном количестве. Между ключевыми словами обязателен пробел. Внутри ключевого слова пробел не допускается.

4.2.4. Стандартные функции

Стандартные функции используются для вычисления часто встречающихся функций. При обращении к стандартным функциям необходимо записать имя функции, а в скобках указать аргумент. Перечень основных стандартных функций и процедур приведен в табл. 7.

Таблица 7

Основные стандартные функции

Функция	Назначение	Тип	
		аргумента	функции
abs(x)	$\lvert x \rvert$	real или	real или
sqr(x)	x^2	integer	integer
sin(x)	sin x		
cos(x)	cos x		
exp(x)	e^x	real или	real
ln(x)	ln x	integer	
sqrt(x)	$\sqrt{x}$		
arctan(x)	arctg x		
trunc(x)	Вычисление целой части числа x	real	integer
int(x)	Вычисление целой части числа x	real	real
frac(x)	Вычисление дробной части числа x	real	real
round(x)	Округление числа x в сторону ближайшего целого	real	integer
pred(x)	Нахождение предшествующего элемента	integer или char или	integer или char или
succ(x)	Нахождение последующего элемента	boolean	boolean
ord(x)	Определение порядкового номера символа	char или	

	в наборе символов	boolean	integer
chr(i)	Определение символа из набора символов по номеру *i*	integer	char
inc(x)	Увеличение значения *x* на единицу		
dec(x)	Уменьшение значения *x* на единицу		
odd(x)	Определение четности числа: true, если *x* нечетное false, если *x* четное	integer	boolean

Примеры использования некоторых стандартных функций:

Функция	Результат	Функция	Результат
Trunc(6.3)	6	Round(8.3)	8
Trunc(6.7)	6	Round(8.9)	9
Int(7.3)	7.0	Frac(9.3)	0.3
Int(7.8)	7.0		

В Турбо–Паскале нет операции возведения в степень, ее заменяют выполнением следующей операции:

$$x^a = \exp(a*\ln(x)).$$

Вычисление логарифмов производят по соотношениям:

$$\log_a x = \ln x / \ln a; \quad \lg x = \ln x / \ln 10.$$

В Паскале определены только три тригонометрические функции: sin, cos, arctg. Для вычисления остальных тригонометрических функций необходимо использовать соотношения:

$$\operatorname{tg} x = \sin x / \cos x;$$
$$\operatorname{ctg} x = \cos x / \sin x;$$
$$\operatorname{csc} x = 1 / \sin x;$$
$$\operatorname{sc} x = 1 / \cos x;$$
$$\arcsin x = \operatorname{arctg}(x/(1-x^2))^{1/2};$$
$$\arccos x = \pi/2 - \arcsin x;$$
$$\operatorname{arcctg} x = \pi/2 - \operatorname{arctg} x.$$

4.2.5. Выражения

Выражение – это синтаксическая единица языка, определяющая способ вычисления некоторого значения. Выражения в языке Паскаль

формируются в соответствии с определенными правилами из констант, переменных, функций, знаков операций и круглых скобок [10–16].

Начинается вычисление с определения переменных и констант, входящих в выражение. Дальнейшие действия выполняются в соответствии с их приоритетом. В первую очередь вычисляются выражения, заключенные в круглые скобки, далее – значения входящих в выражение функций и т.д. Операции одного приоритета выполняются последовательно слева направо.

При вычислении выражений принят следующий приоритет операций:

- арифметических:
  - вычисление значений стандартных функций;
  - умножение и деление;
  - сложение и вычитание;
- логических:
  - not;
  - □, /, div, mod, and;
  - +, –, or;
  - <=, >=, <, >, =.

Тип результата выражения зависит от типов операндов, участвующих в операции. Тип результата операций «+», «□», «–» является INTEGER, если оба операнда имеют тип INTEGER, и REAL – в противном случае. Результатом операции «/» всегда является тип REAL. Результат выполнения логических операций NOT, OR, AND всегда имеет тип BOOLEAN. Аргументы операций сравнения на равенство и неравенство (=, < >) могут иметь любой тип переменных и констант, а результат всегда имеет тип BOOLEAN. В операциях сравнения (>, <, >=, <=) аргументы могут быть любого типа, а результат имеет только тип BOOLEAN.

Примеры записи выражений:

A□EXP(T□T)–SQRT(X□Y□Z);

A□X□X+(4.0□A□B–X□C/2.0).

4.3. Операторы языка

Операторы языка описывают некоторые алгоритмические действия, которые необходимо выполнить для решения задачи. Тело программы можно представить как последовательность таких операторов. Операторы программы разделяются точкой с запятой. Все операторы языка Паскаль можно разбить на две группы: простые и структурированные[10–17].

4.3.1. Простые операторы

Простыми являются те операторы, которые не содержат других операторов. К ним относятся:

- оператор присваивания;
- оператор безусловного перехода GOTO;
- пустой оператор.

4.3.1.1. Оператор присваивания

С помощью этого оператора переменной присваивается значение выражения. Для этого используется знак присваивания **«:=».** Общий вид оператора следующий:

<имя переменной>:= выражение.

В операторе присваивания переменная и выражение должны иметь один и тот же тип. Однако допускается присваивать переменной типа real выражение типа integer. Присваивание же переменной целого типа выражения вещественного типа запрещается.

Пример. Вычислить значение концентрации вещества по формуле $C=P/RT$

при P=10 ат ; T= 513 К; R= 0,001986 ккал/моль·К.

```
Program Conc;
 const R=1.986E–3;
 var P,T:integer;
    C: real;
begin
 P:=10;
 T:=513;
 C:=P/(R□T);
 writeln('C=',C);
end.
```

4.3.1.2. Оператор безусловного перехода GOTO

Оператор GOTO позволяет изменить стандартный последовательный порядок выполнения операторов в программе и

перейти к выполнению программы, начиная с заданного оператора. Общий вид оператора следующий:

goto n ,

где n – метка оператора.

Метки, используемые в Турбо–Паскале, могут быть двух типов:

- целым числом в пределах от 0 до 9999;
- обычным идентификатором.

Метка должна быть описана в разделе **label**. Одной меткой можно пометить только один оператор. Например:

```
goto 20;
10: B:=3;
. . . .
20: X:=X/B;
goto 10;
```

4.3.1.3. Пустой оператор

Пустой оператор – это оператор, не выполняющий никакого действия. Он используется для выхода из середины программы или составного оператора. Пустому оператору соответствует символ «;». Чаще всего пустой оператор встречается с меткой. Например:

```
goto 10;
. . .
10: ;
End.
```

Символ «;» можно опустить, тогда можно записать

10: **End**.

4.3.1.4. Ввод-вывод данных

Для ввода и вывода данных в Турбо–Паскале существуют стандартные процедуры ввода–вывода, вызываемые соответственно операторами READ и WRITE [10–16].

Операторы ввода:

1) Read (<список переменных>) – последовательный ввод переменных из списка;

2) Readln (<список переменных>) – то же, что и оператор Read, только после ввода данных происходит переход на новую строку, т.е. ввод осуществляется каждый раз с новой строки;

3) Readln – происходит переход на новую строку без ввода данных.

Значения вводимых переменных должны соответствовать типам переменных из списка ввода. В Турбо–Паскале допускается вводить значения следующих данных: целых (integer), вещественных (real), символьных (char), строковых (string).

С помощью оператора ввода нельзя ввести:

1) значение логической переменной;

2) значение переменной типа «массив» (необходимо вводить значения отдельных элементов массива);

3) значение переменной типа «перечисление»;

4) значение переменной типа «запись».

Ввод в языке Паскаль может быть только бесформатный. Данные набираются на терминале, разделителем между числами служит пробел. Разделитель между символами, между числом и символом не нужен.

Пример 1. Требуется ввести значения следующих переменных:
A=2.5; S=7.42; P= –0.34E–01; M=10; N=15; C1='B'; C2='K'; C3='E'

```
var A,S,P:real;
   M,N:integer;
   C1,C2,C3:char;
. . . . . .
   Readln(A,S,P);
   Read(M,N,C1,C2,C3);
```

Значения переменных вводятся следующим образом:

```
 2.5 7.42 –0.34E–01
10 15BKE (без апострофов)
```

или

```
  2.5
  7.42
–0.34E–01
10 20BKE
```

Операторы вывода.

Оператор вывода данных имеет три формы записи:

1) Write (<список переменных>) – выводит последовательно значения переменных из списка;

2) Writeln (<список переменных>) – то же, что и оператор Write, но после вывода переменных осуществляется переход на новую строку (следующий оператор вывода будет выводить данные с начала новой строки);

3) Writeln – осуществляет переход на новую строку без вывода данных.

В Турбо–Паскале допустим вывод значений данных следующих типов: вещественных, символьных, логических и строковых.

Значения величин действительного типа выводятся в нормализованном виде (мантисса числа с порядком), а целого типа – в обычной форме.

Пример 2. Пусть в результате выполнения программы переменные получили следующие значения:

K= –7; P=8.74; S='+'; C=True.

Выведем их на печать:

```
 var K: integer; P: real; S: char; C: boolean;
Begin
 Writeln ('  Пример');
 Writeln ('K=',K,' P=',P);
 Writeln ('S=',S);
 Write ('C=',C);
End.
```

Информация будет выведена в следующем виде:

```
 Пример
K=–7 P=8.7400000000E+0000
S=+
C=True
```

В Турбо–Паскале предусмотрен вывод данных по формату. Общий вид записи оператора для вывода значений *целого* типа : **Write (p:m);**

где p – имя переменной, m – число позиций, отводимое для выводимой величины p.

Для вывода значений *действительного* типа:

1) Write (p:m); 2) Write (p:m:n);

где m – общее число позиций для выводимой переменной p, включая знак числа, целую часть, точку и дробную часть; n – число позиций дробной части.

Оператором (1) переменная вещественного типа p выводится в виде константы с плавающей точкой) с шириной поля m.

Оператором (2) переменная p выводится в виде константы с фиксированной точкой.

Пример 3. Используем форматный вывод для следующих переменных:

K= –7; P=8.74; X=3.524; X1= $0.264*10^{-5}$; S='+'; C=True.

```
Writeln ('K=',K:3,' P=',P:5:2);
Writeln ('X=',X:10,' X1=',X1:10);
Writeln ('S=',S:2,' C=',C:6);
```

В результате вывода получим:

K= –7 P= 8.74

X= 3.5240E+00 X1=0.2640E-05;

S= + C= True

4.3.1.5. Программирование линейных алгоритмов

Рассмотрим пример программы для линейного алгоритма.

Пример. Вычислить скорость химической реакции по выражению

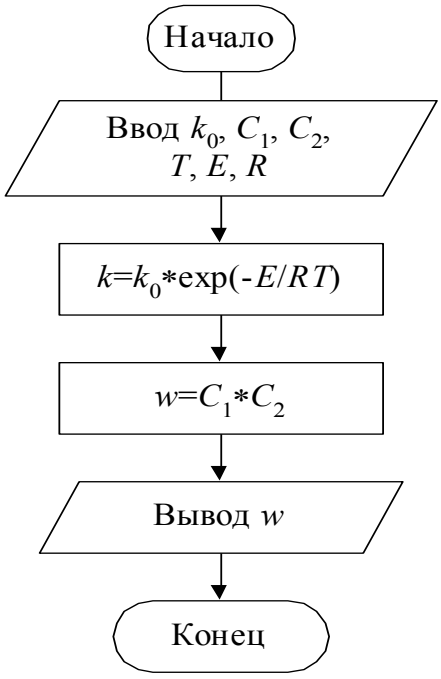
$$w = k \cdot C_1 \cdot C_2,$$

в котором константа скорости рассчитывается по формуле Аррениуса

$$k = k_0 \cdot e^{-E/RT},$$

где k_0= $0.34 \cdot 10^7$, C_1= 1.2 моль/л, C_2= 0.2 моль/л, E= 22.1 ккал/моль, T=473 К,

R = 0.00198 ккал/моль·К.



```
Program Scor;
 var k0,k,C1,C2,E:real;
     T:integer;
 const R=0.198E–2;
Begin
 k:=0.34E7; E:=22.1; C1:=1.2; C2:=0.2;
 writeln('введи T=');
 readln(T);
 k:=k0□exp(–E/(R□T));
 W:=k□C1□C2;
 writeln('k=',k,' W=',W);
End.
```

4.3.2. Структурированные операторы паскаля

Структурированными являются такие операторы, которые состоят из других операторов [10-16]. К ним относятся:

- составной оператор;
- условный оператор IF;
- условный оператор CASE;
- операторы цикла WHILE, REPEAT, FOR.

4.3.2.1. Составной оператор

Составной оператор позволяет объединить несколько операторов Паскаля в одну конструкцию, которая рассматривается как составной оператор. Общий вид оператора следующий:

```
begin
 оператор 1;
 оператор 2;
  . . . . . .
 оператор n
end;
```

В этой конструкции слова **begin** и **end** выполняют роль операторных скобок. Составной оператор можно включать в любое место программы, где допускается использование только одного оператора. В свою очередь, любой из операторов составного оператора также может быть составным. Извне составного оператора нельзя передавать управление внутрь его (оператором **goto**).

4.3.2.2. Условный оператор

Условный оператор позволяет выбирать одно из двух условий. Существуют следующие виды записи условного оператора:

If <условие> **then** <оператор>;

If <условие> **then** <оператор1> **else** <оператор2>;

If <условие> **then** <оператор1> **else if** <условие> **then** <оператор2>

else <оператор3>;

Для условного оператора первого вида, если условие истинно, то выполняется оператор, стоящий после **then**. Если же условие ложно, то этот оператор не выполняется, а выполняется оператор, следующий за условным. Например:

if x<0 **then** y=x+x.

Второй вид записи оператора позволяет производить выполнение оператора 1, если условие истинно. Если условие ложно, то выполняется оператор 2. Например**:** **if** x>0 **then** y:=sqrt(x) **else** y:=x.

В третьей форме записи условный оператор расширен за счет вложенности новых условий. Это приводит к сокращению числа условных операторов, но снижает наглядность программы. Новые условия могут записываться за ключевыми словами **then** и **else**. Ключевое слово **else** всегда относится к ближайшему **if**. Например:

```
if x<a then p:=ln(x)
else if x>b then p:=sin(x)
else      p:=cos(x).
```

Следует помнить, что условный оператор управляет только одним оператором. Поэтому, если возникает необходимость выполнения группы операторов, то их надо объединить в один, взяв в операторные скобки (т.е. использовать составной оператор **begin–end**). Кроме того, при необходимости учета нескольких условий используются логические операции: and (и), or (или), not (не) .

Например, алгоритм : если A<D и A>C то Y1:=A^2 и Y2:=A*C; будет записан следующим образом

If (A<D) and (A>C) then begin Y1:= sqr(A); Y2:=A*C end;

.**Пример.** Вычислить скорость осаждения капелек воды в неподвижной среде электродегидратора в зависимости от значения Рейнольдса:

$$U = \begin{cases} \dfrac{d^2 g(\rho_1 - \rho_2)}{18\nu_2\rho_2}, & \text{если } 1\cdot 10^{-4} \le \mathrm{Re} \le 2.0, \\ \sqrt{3.03 dg(\rho_1 - \rho_2)/\rho_2}, & \text{если } \mathrm{Re} \le 500. \end{cases}$$

где *d* –диаметр капелек воды, м; ρ_1, ρ_2 – плотности воды и нефти, кг/$м^3$; ν_2 – кинематическая вязкость нефти, $м^2$/с.

```
Program SKOR;
 var d,g,V2,r1,r2,U,Re:real;
Begin
 writeln('Введите значение d g r1 r2 v2 Re=');
 readln(d,g,r1,r2,v2,Re);
 if (Re>=1.0E–4) and (Re<=2.0) then
 U:=sqr(d)□g□(r1–r2)/(18□V2□r2);
 if Re>500 then U:=sqrt(3.03□d□g□(r1–r2)/r2);
 writeln('U=',U);
end.
```

4.3.2.3. Оператор выбора CASE

Оператор CASE предназначен для программирования алгоритмов с большим числом разветвлений. Этот оператор обеспечивает выполнение одного оператора (простого или составного) из нескольких возможных.

Общий вид оператора CASE:

case <выражение–селектор> **of**

<список меток 1>: оператор 1;

<список меток 2>: оператор 2;

. . . . .

<список меток n>: оператор n

else <оператор>

end;

Здесь значение выражения должно быть одного и того же скалярного типа (кроме real), что и метки. Оператор выбора действует следующим образом. Если значение выражения равно одной из меток, то выполняется соответствующий ей оператор. Затем управление передается за пределы оператора выбора.

Замечание. Метки оператора CASE не описываются в разделе **label**, и на них нельзя переходить оператором GOTO. Метки внутри одного оператора выбора должны быть различными.

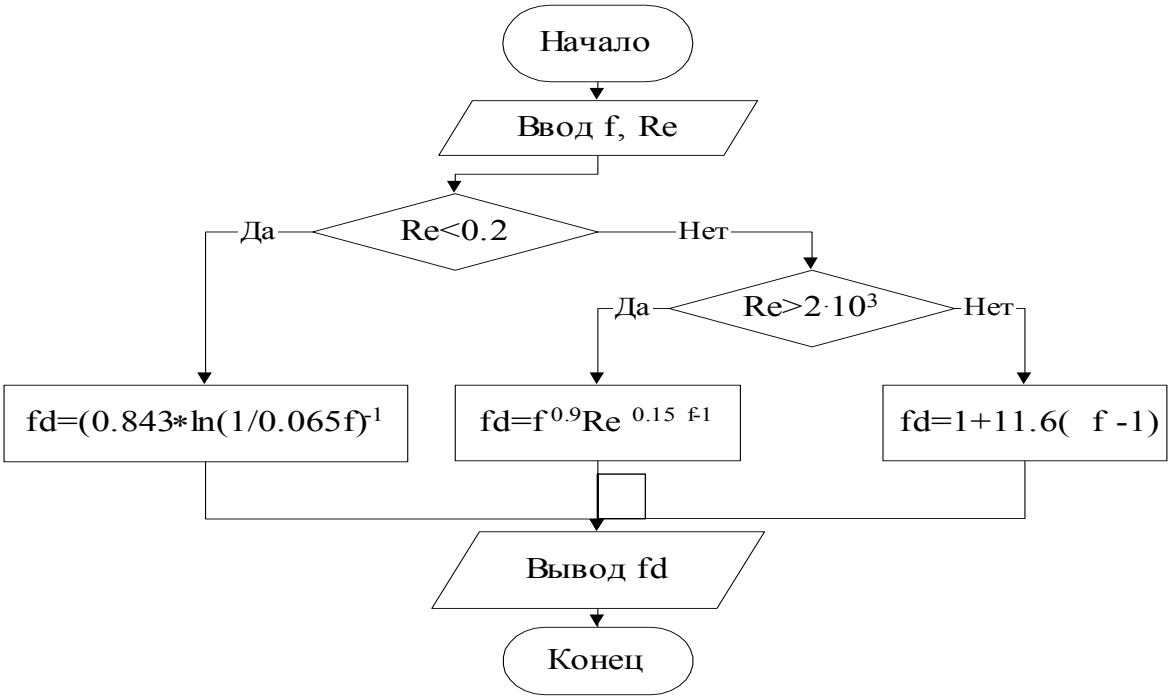
4.3.2.4.Программирование разветвляющихся алгоритмов

Рассмотрим пример программы для разветвляющегося алгоритма.

Пример. Вычислить динамический коэффициент формы сферической частицы катализатора

$$f_g = \begin{cases} \left(0.834 \ln \dfrac{1}{0.065 f}\right)^{-1}, & \mathrm{Re} \le 0.2, \\ f^{0.9}\,\mathrm{Re}^{0.15\sqrt{f-1}}, & 0.2 < \mathrm{Re} < 2\cdot 10^3, \\ 1 + 11.6\left(\sqrt{f-1}\right), & \mathrm{Re} \ge 2\cdot 10^3. \end{cases}$$

где f=1.16. Обозначим f_g = fd.



Программа.

```
Program Koef;
 var fd,f,Re:real;
Begin
 write('Введите fd,Re');
 readln(fd,Re);
 if Re<=0.2 then fd:=1/(0.843□ln(1/0.065□f))
 else if Re>=2.0E3 then fd:=exp(0.9□ln(f))□
 exp(0.15□sqrt(f–1)□ln(Re))
 else fd:=1+11.6(sqrt(f)–1));
 writeln('fd=',fd:10);
end.
```

4.3.2.5. Операторы цикла

Для организации циклов (повторов) при записи алгоритмов на языке Паскаль используются три вида операторов цикла:

WHILE – оператор цикла с предварительным условием;

REPEAT – оператор цикла с последующим условием;

FOR – оператор цикла с управляющим параметром.

Оператор цикла WHILE.

Общий вид оператора следующий:

while <условие> **do** <оператор>;

где

<условие> – логическое выражение;

<оператор> – тело цикла (простой или составной оператор).

Оператор действует следующим образом. Проверяется условие, если оно истинно, выполняются операторы циклической части. Как только оно становится ложным, происходит выход из цикла. На литературном языке это будет выглядеть примерно так: «Пока указанное условие верно, выполнять следующее».

Пример: Зависимость удельной теплоемкости химического соединения от температуры выражается формулой

$C_p = a + bT + cT^2$,

где a, b, c – постоянные коэффициенты; T – температура.

Вычислить удельную теплоемкость в интервале температур от 200 до 800 К с шагом 50 К.

Программа.

```
Program Tepl;
 var a,b,Cp:real;
     T,h:integer;
Begin
 writeln('Введите коэффициенты');
 readln(a,b,c);
 T:=200; h:=50;
 while T<=800 do
  begin
   Cp:=a+b□T+c□T□T;
   T:=T+h;
   writeln('T=',T:7:3,' Cp=',Cp:10);
  end;
End.
```

Действие цикла можно прокомментировать: «Пока температура меньше 800 К, вычислять значение теплоемкости». При выполнении программы все нужные нам значения теплоемкости будут выданы на печать.

Оператор цикла REPEAT.

Общий вид оператора следующий:

repeat

<оператор 1>;

. . . . . {операторы циклической части}

<оператор n>

until <условие>;

Оператор действует следующим образом. Выполняются операторы циклической части, проверяется условие. Если оно ложно, то вновь выполняется тело цикла, если оно истинно, то происходит выход из цикла. Это может быть выражено так: «Повторять действие до тех пор, пока не выполнится условие».

Примечание. Так как границы цикла обозначены словами REPEAT и UNTIL, нет необходимости заключать операторы циклической части в операторные скобки **begin – end**, хотя их использование не является ошибкой.

Пример. Вычислить значение теплоемкости Ср с использованием оператора REPEAT.

```
Program Tepl;
 var a,b,c,Cp:real;
     T,h:integer;
Begin
 writeln('Введите a,b,c=');
 readln(a,b,c);
 T:=200; h:=50;
 repeat
  Cp:=a+b□T+c□T□T;
  T:=T+h;
  writeln('T=',T:7:3,' Cp=',Cp:10);
 until T>800;
End.
```

Примечание: действие оператора REPEAT, противоположно действию оператора WHILE, т.к. в первом условие выхода из цикла должно быть истинным, а во втором – ложным. Значения переменных, входящих в условие операторов WHILE и REPEAT должны обязательно

изменяться в теле цикла, иначе цикл не будет завершен. (В приведенном нами примере - это значение переменной T).

Оператор цикла FOR.

Оператор цикла FOR используется для организации цикла, когда известно число повторений. Существует два варианта оператора:

– при увеличении значения параметра (цикл с положительным шагом: +1)

for i:=n1 **to** n2 **do** <оператор>;

– при уменьшении значения параметра (цикл с отрицательным шагом: –1)

for i:=n1 **downto** n2 **do** <оператор>,

где i – параметр цикла; n1 и n2 – начальные и конечные значения параметра цикла; <оператор> – тело цикла (простой или составной операторы). Параметры i, n1, n2 должны иметь один и тот же тип, кроме real, шаг параметра цикла всегда 1.

Цикл действует таким образом. Параметру i присваивается начальное значение n1 и сравнивается со значением n2. До тех пор, пока параметр i меньше или равен конечному значению n2 (в первом варианте) или больше, или равен n2 (во втором варианте), выполняются операторы циклической части; в противном случае происходит выход из цикла.

Примечание:

1. Внутри цикла нельзя изменять начальное (n1) и конечное (n2) значения параметра цикла, а также само значение i.
2. После завершения цикла значение параметра i становится неопределенным (т.е. ничему неравным), за исключением выхода из цикла при помощи GOTO.
3. Во всех трех операторах цикла внутри цикла можно использовать операторы IF, GOTO. Разрешается в любой момент выходить из цикла, не дожидаясь его завершения. Но запрещено при помощи этих операторов передавать управление извне цикла внутрь цикла.

Пример: Рассмотрим расчет теплоемкости с использованием оператора FOR.

```
Program Tepl;
 var a,b,c,Cp:real;
    T,h,i:integer;
Begin
 writeln('Введи a,b,c=');
 readln(a,b,c);
 T:=200; h:=50;
```

```
 for i:=1 to 13 do
  begin
   Cp:=a+b□T+c□T□T;
   T:=T+h;
   writeln('T=',T:7:3,' Cp=',Cp:10);
  end;
End.
```

Параметр цикла i изменяется от 1 до 13, т.к. на заданном интервале температуры от 200 до 800 К с шагом 50 К должно быть вычислено тринадцать значений теплоемкости.

Следует помнить. Операторы цикла WHILE и FOR могут содержать в теле цикла только один оператор. Поэтому при необходимости вычисления нескольких операторов необходимо заключать их в операторные скобки (т.е.использовать составной оператор begin ..end).

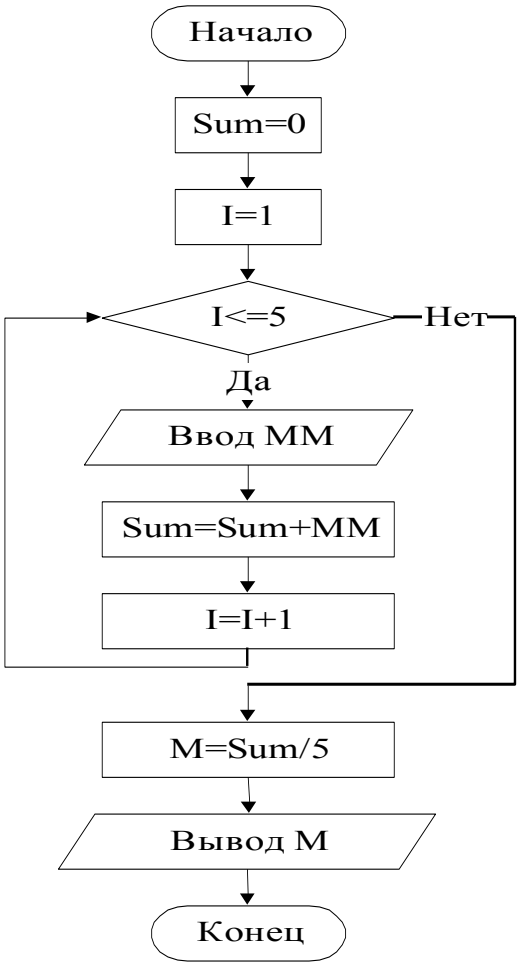
4.3.2.6. Программирование циклических алгоритмов

Рассмотрим примеры программ для циклических алгоритмов.

Пример 1. Составить программу вычисления средней молекулярной массы пяти органических соединений от CH_4 до C_5H_{12}.

Для этого вначале необходимо найти сумму молекулярных масс всех 5 соединений: Sum=ΣMM , а затем найти ее среднее значение M=Sum/5.

```
Program Molmas;
 var M,MM,Sum:real;
     i:integer;
Begin
 Sum:=0; i:=1;
 while i<=5 do
  begin
   writeln('Введи мол. массу');
   readln(MM);
   Sum:=Sum+MM;
   i:=i+1;
  end;
 M:=Sum/5;
 writeln('M=',M:10:3);
End.
```



Пример 2. Химическая формула углеводорода предельного гомологического ряда в общем случае имеет вид: C_nH_{2n+2} , где n – количество атомов углерода в молекуле. Вычислить молекулярную массу углеводородов от CH_4 до $C_{10}H_{22}$.

Обозначим: MM – молекулярная масса углеводорода; MC, MH – молекулярные массы углерода и водорода.

```
Program Mol;
 const MC=12; MH=1;
 var MM:real;
     n:integer;
Begin
 for n:=1 to 10 do
  begin
   MM:=MC□n+MH□(2□n+2);
   writeln('Мол. масса C',n,'H',2□n+2,
           '=',MM:10:3);
```

```
  end;
End.
```

В результате вычислений при каждом значении n будет выводиться на экран запись:

Мол. масса CH4 = 16

. . . . .

Мол. масса C10H22 = 142

Пример 3. Вычислить значение константы скорости по формуле Аррениуса

$$k = k_0 \cdot \exp(-E/RT);$$

$k_0 = 2.4 \cdot 10^6$; R= 0.001986 ккал/моль·К; E= 12.3 ккал/моль.

Температура изменяется от 400 до 500 К с шагом 20 К.

```
Program Constant;
 const R=0.001986; E=12.3;
 var k,k0:real;
     T:ineger;
Begin
 k0:=2.4E6;
 repeat
  k:=k0*exp(–E/(R*T));
  writeln ('при T=',T:5,' k=',k:10);
  T:=T+20;
 until T>500;
End.
```

4.4. Структурированные типы данных

4.4.1. Массивы

Массив – это упорядоченная последовательность элементов одного типа, обозначенных одним именем [10–17].

Отдельная величина последовательности называется элементом массива (переменная с индексом). Индекс указывает положение (адрес) элемента в массиве.

Любой массив имеет имя, размерность и длину (размер). Количество индексов у переменной с индексом определяет размерность массива. Длина массива- это общее число его элементов.

Примерами массивов могут быть:

1) вектор **x** = $\{x_1, x_2, ., x_{10}\}$ – это одномерный массив состоящий из десяти элементов x_i, где i=1, …, 10.

2) матрица $a_{11}\ a_{12}\ a_{13}$

$a_{21}\ a_{22}\ a_{23}$

это двумерный массив из шести элементов a_{ij}, где i=1,2; j=1,2,3.

Описание массивов. Возможны два способа описания массивов:

1) **type** <имя типа> = **array**[<тип индексов>] **of** <тип компонент>;
var <имя массива>:<имя типа> .

Вначале определяется некоторый тип со структурой массива, а затем описывается переменная, как имеющая данный тип.

2) **var** <имя массива> : **array** [<тип индексов>] **of** <тип компонент>.

Например, массивы вещественных чисел $a_1, a_2, \ldots, a_{10}$ и $b_1, b_2, \ldots, b_{10}$ можно описать:

1) **type** mas = **array**[1..10] **of** real;
var a,b:mas;

2) **var** a,b: **array** [1..10] **of** real;

Размерность массивов в Турбо–Паскале не ограничена.

Доступ к каждому элементу массива можно выполнить путем указания имени массива, за которым в квадратных скобках следует индекс элемента.

Индекс элемента может быть задан переменной – a[i]; числом – a[5]; выражением – a[2□i–1].

Примеры описания массивов:

```
var
{одномерный массив целых чисел}
x:array[1..10]of integer;
{одномерный массив вещественных чисел}
y:array[1..5]of real;{двумерный массив вещественных чисел}
a:array[1..3,1..5]of real;
{трехмерный массив символьных данных}
b:array[1..2,1..5,1..3]of char;.
```

Ввод–вывод массивов.

Для ввода–вывода массивов используются циклы. Рассмотрим ввод и вывод массивов на примерах.

Пример ввода–вывода **одномерного** массива.

Пусть для решения задачи необходимо ввести численные значения молекулярных масс десяти химических веществ: M_1, M_2, ... , M_{10}.

```
 var M:array[1..10]of real;
    i:integer;
Begin
 {ввод значений массива M в столбце}
 writeln('Введите M');
 for i:=1 to 10 do
  begin
   writeln('M',i);
   readln(M[i]);
  end;
 {вывод элементов массива M в строку}
 for i:=1 to 10 do
  write(M[i]);
End.
```

Ввод значений с экрана монитора будет происходить следующим образом. После появления записи: «введи M», следует записать численное значение M[1] и нажать на «**Enter**» и так до десятого элемента включительно.

Введи M
M1 <значение M[1]> ;
M2 <значение M[2]> ;
. . . . .
M10 <значение M[10]> .

Пример ввода–вывода **двумерного** массива.

Требуется ввести значения теплоемкостей пяти органических соединений, представляющих три гомологических ряда: алканы, алкены, спирты.

Cp_{11}, Cp_{12}, ... , Cp_{15}
Cp_{21}, Cp_{22}, ... , Cp_{25}
Cp_{31}, Cp_{32}, ... , Cp_{35}

```
var Cp:array[1..3,1..5]of real;
   i,j:integer;
Begin
{ввод значений массива}
 for i:=1 to 3 do
  for j:=1 to 5 do
   begin
```

```
   writeln('Введите Cp',i,j);
   readln(Cp[i,j]);
  end;
{вывод элементов массива по строкам}
 for i:=1 to 3 do
  begin
   for j:=1 to 5 do
    write(Cp[i,j]:7:3,' ');
   writeln;
  end;
 . . . . .
End.
```

Ввод массива будет осуществляться таким же образом, как и в случае одномерного массива.

При выводе массива на экран появляются три строки по пять численных значений теплоемкостей:

```
 Cp[1,1] Cp[1,2] ... Cp[1,5]
 . . . . . .
 Cp[3,1] Cp[3,2] ... Cp[3,5]
```

Численные значения элементов массива могут быть заданы в разделе const:

```
Type   mas = array[1..3] of real;
       mas1 = array[1..2,1..3] of integer;
Const  M:mas = (12.3,14.6,18.4);          {одномерный массив}
       Cp:mas1 = ((10,16,8), (6,20,12));  {двумерный массив}
```

4.4.2. ФАЙЛЫ

Удобным способом хранения информации служит запись этой информации на магнитный носитель (жесткие, гибкие диски, магнитные ленты). Запись удобна особенно тогда, когда объем информации велик и в дальнейшем предполагается использовать эту информацию в других программах. В Паскале предусмотрен специальный тип данных – файлы, операции над которыми сводятся к работе с внешними носителями [8,9,11].

Файл – это поименованная область памяти на каком–либо носителе информации, предназначенная для хранения данных. Этим носителем может быть гибкий или жесткий диск (магнитная лента).

Внешние файлы должны быть описаны в разделе описаний программы. Описание файлов в общем случае имеет следующий вид:

type <имя файла> = **file of** <тип компонент>;

var <имя файловой переменной>:<имя типа>;

или

var <имя файловой переменной> **file of** <тип компонент>;

Файловая переменная (обозначим ее как f) служит для доступа к файлу.

В Турбо–Паскале существуют следующие категории файлов:

- типизированные;
- нетипизированные;
- текстовые.

В зависимости от категории объявление файлов соответственно будет:

var f : **file of** <тип компонент>;

var f : **file**;

var f : **text.**

Например:

var f1:**file of** char;

f2:**file**;

f3**:text.**

Так как по определению, число элементов файла не задается, то для нахождения конца файла введена стандартная функция

Eof(var:file):boolean;

Когда достигнут конец файла, Eof принимает значение True, в противном случае – False.

Стандартные процедуры для работы с файлами.

Работа с файлами производится посредством специальных стандартных процедур. Рассмотрим некоторые из них.

ASSIGN (f, '<имя внешнего файла>') – эта процедура связывает файловую переменную f с именем внешнего файла на диске.

Например: Assign (f1, 'fl.d'), здесь имя файловой переменной **f1** связывается с файлом f1.d на диске.

Процедуру Assign необходимо использовать до начала работы с файлом (до его открытия процедурой Reset или Rewrite).

RESET (f) – процедура открывает существующий файл **f** для чтения.

REWRITE (f) – создает и открывает новый файл для записи.

APPEND (f) – открывает существующий файл для добавления данных.

READ (f, X1,...,Xn) или READLN (f, X1,...,Xn)– считывает из файла f значения переменных X1 … Xn.

WRITE (f, X1,...,Xn) или WRITELN (f, X1,...,Xn) – записывает в файл f значения переменных X1 … Xn.

CLOSE (f) – закрывает файл f после окончания работы с ним.

Особым типом файлов являются ***текстовые файлы***. Эти файлы содержат некоторый текст, который состоит из обычных символов (например, букв алфавита и цифр). Символы текстового файла разбиты на строки.

Описание текстового файла:

Var <имя файла>: text;

Текстовый файл состоит из последовательности строк различной длины. Для определения конца строки используется функция

Eoln(var F:text) : Boolean;

Она принимает значение True, если достигнут конец строки, и значение False – в противном случае.

Для чтения из текстового файла или записи в текстовый файл можно использовать **процедуры Write (f, X1,...,Xn), Writeln(f, X1,...,Xn) Read(f, X1,...,Xn) , Readln (f, X1,...,Xn) .**

Следует отметить, что, несмотря на то, что текстовый файл является набором символьных значений, он может использоваться (и часто используется) для хранения численных значений. При считывании или записи значений в файл происходит автоматическое преобразование из числового формата в символьный и наоборот.

Продемонстрируем работу с файлами на примере.

Пример. Составить программу пересчета концентраций химических веществ, заданных в мольных долях, в весовые :

$$BD_i = \frac{MD_i \cdot MB_i}{\sum MD_i \cdot MB_i};\ i = 1,...,5,$$

где

BD_i – концентрация в весовых долях;

MD_i – концентрация в мольных долях;

MB_i – молекулярный вес веществ.

Исходные данные ввести из файла, результат вычислений поместить в файл.

Программный файл.

```
Program Conz;
 type mas=array[1..10]of real;
 var BD,MD,MB:mas;
    s:real;
    i:integer;
    f1,f2:text; {объявление файлов}
Begin
 Assign(f1,'dat'); Assign(f2,'rez');
 Reset(f1); Rewrite(f2);
 {ввод данных из файла dat}
```

```
for i:=1 to 5 do
read(f1,MD[i]);
readln(f1);
for i:=1 to 5 do
read(f1,MB[i]);
s:=0.0;
for i:=1 to 5 do
s:=s+MD[i]□MB[i];
for i:=1 to 5 do
BD[i]:=MD[i]□MD[i]/s;
{вывод результатов в файл rez}
for i:=1 to 5 do
write(f2,BD[i]:6:2,' ');
close(f1); close(f2);
End.
```

После написания и сохранения программного файла в новый файл согласно последовательности ввода данных в программе (1–я строка – массив значений концентраций MD_i ; 2–я строка – массив значений молекулярных весов MB_i) записывают через пробел исходные численные данные и сохраняют файл под именем dat.

После выполнения вычислений ПЭВМ создает файл с именем rez, в который помещаются результаты расчетов (строка численных значений массива BD_i).

4.4.3. Строки

Строковые данные предназначены для обработки текстовой информации [8,9,11]. Данные типа string делятся на константы и переменные.

Строковые константы могут быть описаны в разделе констант и не меняются в ходе выполнения программы. Например:

```
const St='строка';
    C='константа';
```

Описание строковых переменных имеет вид:

type <имя типа> = **string**[N];

var <имя переменнй> : <имя типа>;

или

var <имя переменной> : **string**[N];

Здесь N – целая константа, указывающая максимальную длину строки (количество символов в строке). В Турбо–Паскале 1 <N<255. Если размер строки не указан, то считается равным 255. В этом случае описание строки будет следующим:

var <имя переменной> : **string**;

Например, описание строк 'ДАВЛЕНИЕ' и 'MOL' следующее:

```
var St1:string[8];
   St2:string[3].
```

Строковые переменные аналогичны массивам типа char. Их отличие в том, что число символов (текущая длина строки) может динамически меняться в интервале от нуля до заданного верхнего значения N. Для ссылки на специальные компоненты, как обычно в случае массивов, используются переменные с индексом St1[2], St2[3]. Допускается описание строковых данных в разделе const:

```
const St1:string[8]='ДАВЛЕНИЕ';
     St2:string[3]='MOL'.
```

Операции над строками

Для строк применимы операции присваивания, сцепления и сравнения.

Операции присваивания

```
Str1:='SR';  Str2:='SKOR';
Str1:=Str2;
```

В результате будет присвоено: Str1:='SK'. Ввод и вывод строковых переменных осуществляется без апострофов. Например, строковая переменная Str имеет значение 'TEMP'. Для выполнения оператора Readln (Str) необходимо набрать TEMP, начиная с 1–й позиции.

Замечание. Для исключения ошибок ввода–вывода строковых переменных следует всегда использовать оператор Readln (вместо Read).

Операции сцепления

Эти операции применяются для сцепления нескольких строк в одну строку. Операция сцепления обозначается знаком +.

Выражение	Результат
'ВК'+'12'+'15'	ВК1215

Пример.

```
Program St;
 var C:char;
    S1:string[3];
    S2:string[7];
```

```
 const S1='PAS';
Begin
 readln(c); {c='L'}
 S2:=S1+'CA'+C;
 writeln('S2=',S2);
End.
```

В результате выполнения получим

S2=PASCAL.

Операции сравнения

Операции сравнения (=, <>, >, < и т.д.) проводят сравнение двух строк и имеют более низкий приоритет, чем операции сцепления. Сравнение строк производится слева направо до первого несовпадающего символа, и та строка считается большей, в которой первый несовпадающий символ имеет больший номер в кодовой таблице ПЭВМ. Результат выполнения операций сравнения над строками всегда имеет логический тип (true или false). Строки считаются равными, если они полностью совпадают по текущей (а не по объемной) длине и содержат одни и те же символы.

Например:

```
Program Pr;
 var St1:string[3];
     St2:string[8];
Begin
 St1:='Str'; St2:=St1;
 if St1=St2 then writeln('St1 равно St2')
 else writeln('St1 не равно St2');
End.
```

В результате будет напечатано:

St1 равно St2.

Следует учесть, что одинаковые заглавные и строчные буквы в строках имеют разные значения строковых переменных. Так, если St1:='STR', а St2:='Str', то St1 не равно St2.

Стандартные процедуры и функции обработки строк

Пусть переменные St, St1, St2 – переменные типа string. Рассмотрим некоторые стандартные процедуры и функции для обработки строк.

1) Delete (St, i, n) – удаляет из строки St подстроку длиной n, начиная с позиции i. Результат – новая строка без подстроки.

Например:

St:='TURBOBAS';

Delete (St,6,3);

Результат: St:='TURBO'.

2) Insert (St1,St2,i) – вставляет строку St1 в строку St2, начиная с позиции i.

Например:

St:='TURBO';

Insert('BAS',St,6);

Результат: St:='TURBOBAS'.

3) Str(val,St) – преобразует число в строковую переменную.

Например:

Str(1342,St1);

Результат: St1:='1342'.

4) Val (St,v,cod) – преобразует строковое значение в численное данное. Если во время преобразования ошибки не обнаружено, Cod=0. В противном случае Cod будет содержать номер позиции первого ошибочного символа.

5) Copy (St,i,n) – выделяет из St подстроку длиной n, начиная с позиции i.

Например:

St:=Copy('Turbo–Pascal',7,6);

Результат: St:='Pascal'.

6) Concat (St1,St2,...,Stn) – выполняет сцепление строк в указанном порядке.

Например:

St3:=Concat(St1,St2,St4).

Эта запись эквивалентна:

St3:=St1+St2+St4.

7) Length (St) – определяет длину строки.

Например:

St:='MOLEKULA';

L:=length(St);

Результат: L=8.

8) Pos (St1,St2) – определяет наименьший номер элемента в St2, начиная с которого St1 входит в St2 как подстрока.

Например:

St:='Давление, атм';

P:=Pos('атм',St);

Результат: P=10.

4.4.4. ЗАПИСИ

Во многих информационных задачах при обработке документов, списков часто возникает необходимость объединения различного типа данных в одну группу.

Для этого в ТП существует тип данных - ***записи***, позволяющий хранить вместе переменные, имеющие различные типы данных.

Запись – это структура данных, состоящая из фиксированного числа элементов одного или нескольких типов. В одном поле данные имеют один и тот же тип, а в разных полях могут иметь разные типы.

Общий вид описания записи:

```
 type  <имя типа >=record
 < список элементов 1 > : тип 1;          {поле 1};
 < список элементов 2 > : тип 2;          {поле 2};
. . . . . .
 < список элементов n > : тип n;          {поле n};
end;

 var   <имя записи >: имя типа ;


 var  <имя записи> : record
 < список элементов 1 > : тип 1           {поле 1};
 < список элементов 2 > : тип 2;          {поле 2};
 . . . . . .
 < список элементов n > : тип n;    {поле n};
  end;
```

<список элементов >- перечень имен элементов через запятую;
тип 1, тип 2, … **тип n**..- типы полей.

Между ключевыми словами **record** и **end** заключен список элементов, называемых ***полями*** с указанием имен полей и типа каждого поля. Поля отделяются друг от друга (**;)**, количество полей записи может быть любым.

Тип данных **record** предоставляет программисту возможность объединить в одну связную структуру различные по смыслу и типу элементы (поля). Причем элементами записи могут быть и структурированные типы данных, например массивы и другие (подчиненные) записи. Для обработки доступна как вся запись целиком, так и отдельные ее поля.

Обращение к элементам (полям) записи выполняется с помощью составного имени: Указывается имя всей записи и имя отдельного поля, причем, они разделяются точкой.

< имя записи >. < имя элемента >

Пример 1.

Опишем запись, содержащую информацию об имени, домашних и рабочих телефонах лиц, с которыми общается пользователь.

```
Program Zap1;
Type  St1 = string[30];
St2 = string[10];
St3 = string[100];
Zap = record                                    { запись }
        Name1, Name2 : st1;                     {Ф.И.О.}
        Rtel1, Rtel2, Dtel1, Dtel2 : st2; {рабрчие и домашние телефоны}
        Inform : st3;
end;
var  rv: zap;                                   {rv - имя записи}
f1, f2 : text;
Begin
clrscr;
assign (f1,   ‘zapd.pas’); reset (f1);
assign (f2,   ‘zapr.pas’); rewrite (f2);
readln (f1, rv. Name1);
readln (f2, rv. Name2);
```

```
readln (f1, rv.RTel1, rv. RTel2);
readln (f1, rv.Inform);
writeln (f2,  rv. Name1, ‘   : р.т.,  ‘, rv. RTel1, ‘   д.т.  ‘, rv. Dtel1);
writeln (f2,  rv. Name2, ‘   : р.т.,  ‘, rv. RTel2, ‘   д.т.  ‘, rv. Dtel2);
writeln (f2);
writeln (f2, rv. Inform);
close (f1); close(f2);
End.
```

Файл с исходными данными (zapd.pas):

Петров Андрей Васильевич
Иванова Татьяна Николаевна
26-96-22 41-55-90
55-42-32 23-17-05
Петров А.В. – гл. инженер комбината; Иванова Т.Н. – гл.технолог комбината

Файл с результатами (zapr.pas):

Петров Андрей Васильевич: р.т. 26-96-22 д.т. 41-55-90
Иванова Татьяна Николаевна: р.т. 55-42-32 д.т. 23-17-05
Петров А.В. – гл. инженер комбината; Иванова Т.Н. – гл.технолог комбината

Следует помнить. Переменные типа «запись» могут участвовать в операторах присваивания, но никакие операции над ними выполняться не могут. Арифметические или какие - либо другие операции могут выполняться только над отдельными полями записи.

Обращение к полям связано с некоторыми неудобствами, так как иногда получаются длинные составные идентификаторы (имена). Для сокращения записи составных имен используется оператор **with … do**.

В операторе, стоящем после **do**, к полям переменной типа «запись» можно обращаться просто по имени поля, не указывая имя переменной. Это облегчает выполнение многократных ссылок к полям записи.

Общий вид оператора:

with<имя записи > do < оператор>

Благодаря этому оператору отпадает необходимость при обращении к полю записи указывать кроме имени поля, еще и имя записи. Идентификатор записи указывается однократно в начале фрагмента программы обработки записи.

Например:

```
with rv do begin
readln (f1, name1); readln (f1, name2);
readln (f1, Rtel 1, RTel2);
readln (f1,Dtel1,Dtel2);
writeln (name, Rtel1, Dtel1);
end;
```

Пример 2.

Даны сведения о результатах сессии студентов различных групп. Определить количество студентов со средним баллом не менее 4.75**.**

```
Program Student;
 type zap=record
      Fio : string[10];              {Ф.И.О.}
      Grup:string[4];                {Номер группы}
      Oz:array[1..3] of integer;     {Массив оценок}
     end;
 var  Spisok : zap;
     kol:integer;                    {Искомое количество студентов}
     n:  integer;                    {Общее число студентов }
     bal:real;                       {Средний балл}
     i:integer;
     f1,f2:text;
 Begin
      Assign(f1,'d.pas'); Reset(f1);
      Assign(f2,'r.pas'); Rewrite(f2);
      Write('Введите число студентов  ');
      Readln(n);
      kol:=0;
      for i:=1 to n do
```

```
        with Spisok  do begin
          read(f1,fio);
         read(f1,grup);
         readln(f1,oz[1],oz[2],oz[3]);
         bal:=(oz[1]+oz[2]+oz[3])/3;
         if bal>=4.75 then kol:=kol+1
        end;
     Write(f2,'  kol=',kol);
     close(f1); close(f2);
 End.
```

В Паскале имеется и другая разновидность записей – ***запись с вариантами*** *(с переменными* или *вариантными* частями). В различных ситуациях эти записи могут иметь разную структуру.

Описание типа «*запись с вариантами*» имеет следующий вид:

```
type
rec = record
 {Описание фиксированных частей}
        v1,v2 : integer;
 {Описание вариантной части}
 case n : Word of
          0 :   (Список полей - описаний переменных);
          1:    (Список полей - описаний переменных);
          . . . . . .
          7:    (Список полей - описаний переменных);
  end;
```

В этом случае описание включает вариантную часть, в которой используется оператор **case.** Описание завершается единственным зарезервированным словом **end.** В качестве селектора оператора **case** используется идентификатор типа. В нашем случае это **n: Word of**. Переменная часть должна следовать после фиксированной части, если таковая имеется.

4.5. Подпрограммы

При решении различных задач часто возникает необходимость проводить вычисления по одним и тем же алгоритмам, но с различными данными в одной программе. Такие вычисления целесообразно оформить в виде отдельных подпрограмм. Подпрограммы являются основой модульного программирования [10–17].

В языке Паскаль имеется две разновидности подпрограмм: ***процедуры*** и ***функции***.

Структура любой подпрограммы аналогична структуре всей программы. Текст подпрограммы должен быть помещен в тексте программы непосредственно перед основным блоком (после объявления констант, типов и переменных).

Все параметры, которые используются в подпрограмме, можно разделить на две категории: глобальные и локальные.

Глобальные – это параметры, объявленные в разделе описаний основной программы, они действуют как в основной программе, так и в любой ее подпрограмме.

Локальные – это параметры, объявленные внутри подпрограммы и доступны только ей самой. Они недоступны для операторов основной программы и других подпрограмм.

Использование подпрограмм связанно с двумя этапами:

- описание подпрограмм;
- обращение к подпрограмме.

4.5.1. ПРОЦЕДУРЫ

Процедуры используются в том случае, если подпрограмма имеет несколько результатов вычислений или результат является многомерной величиной, или не имеет результата.

Описание процедуры имеет вид:

Procedure <имя> (формальные параметры);
 <раздел описаний>
  label <список меток>;
  const <список констант>;
  type <список типов>;
  var <список переменных>
Begin
  <тело процедуры>
End;

и помещается в основной программе в разделе описаний.

Формальные параметры представляют собой список переменных с указанием их типа. Эти переменные не описываются в разделе описания процедур, используются только в теле процедуры и локальны по отношению к ней.

Формальные параметры подразделяются:

1) на параметры–значения – это входные данные для работы процедур и функций, в списке формальных параметров они описываются в виде

(X1:T1; X2:T2;...) или (X1,X2:T;...),

где Xi– имена параметров; Ti – тип параметров.

Например:

(a,b:real; c:integer; ...)

2) параметры–переменные – это, как правило, выходные данные процедуры (результаты выполнения процедуры), которые передаются в основную программу. Описание выходных параметров следующее:

(... **var** x,s:real; **var** y:integer)

В целом заголовок процедуры с описанием формальных параметров может быть таким:

Procedure Prim (a,b:real; c:integer;
var x,s: real; **var** y:integer).

Процедура не может выполняться сама, ее необходимо вызвать из основной программы или другой подпрограммы по имени с указанием в скобках фактических параметров.

Обращение к процедуре:

<имя процедуры> (фактические параметры).

Например:

Prim (a,b,c,x,y).

При обращении к процедуре формальные параметры заменяются на фактические.

Следует запомнить. Между формальными и фактическими параметрами должно быть соответствие по количеству, порядку следования и типу данных.

Входные фактические параметры – это те, которые передаются в процедуру. В приведенном выше примере- это параметры **a, b, c.**

Входными параметрами могут быть: константы, переменные, выражения.

Выходными фактическими параметрами (которые получают значения из процедуры) могут быть только переменные (это параметры **x, y).**

Имена соответствующих формальных и фактических параметров могут быть одинаковыми или разными.

При использовании в качестве параметров процедур сложного типа данных (массивы, множества, записи) в основной программе необходимо описать имя типа этих данных, а затем указать их в списке формальных параметров процедуры.

Пример. Вычислить расход тепла на нагрев стеклобоя при варке стекла

$$Q = Cp \cdot B(T_м - 25)/100,$$

где B – количество стекломассы из боя; $T_м$ – максимальная температура варки; Ср – средняя удельная теплоемкость стекломассы, равная

$$C_р = \frac{\left(\sum P_i A_i T_м + \sum P_i C_i\right) \cdot 4{,}19}{0{,}00146 T_м + 1},$$

где P_i – содержание окислов в стекле, i = 1, 5; A_i, C_i – коэффициенты Шарпа и Гинтера, i = 1,5.

```
Program Steklo;
 type mas=array[1..5]of real;
 var P,A,C:mas;
    Cp,S1,S2,B,Q,Тм:real;
    i:integer;
Procedure Sum2(P,A,C:mas;Тм:real;var S1,S2:real);
 var i:integer;
Begin
 S1:=0.0; S2:=0.0;
 for i:=1 to 5 do
  begin
   S1:=S1+P[i]*A[i]*Тм;
   S2:=S2+P[i]*C[i];
  end;
end;
Begin {основная программа}
 for i:=1 to 5 do
  readln(P[i]);
 for i:=1 to 5 do
  readln(A[i],C[i]);
 readln(Тм,B);
 Sum2(P,A,C,Тм,S1,S2); {обращение к процедуре}
 Cp:=(S1+S2)*4.19/(0.00146*Тм+1);
 Q:=Cp*B*(Тм–25)/100;
 writeln ('Cp=',Cp:10:4,' Q=',Q:10:4);
End.
```

4.5.2. ФУНКЦИИ

Подпрограмма функция предназначена для вычисления какого–либо одного параметра, значение которого присваивается имени функции.

Общая структура записи функции имеет вид:

Function <имя> (формальные параметры): <тип функции>;
<Раздел описания локальных переменных, меток, констант>
Begin
<тело функции>;
end;

Формальные параметры – это параметры, необходимые для выполнения данной функции, т.е. – исходные данные.

Тип функции может быть только скалярным: real, integer, char, boolean, string.

Особенности функций следующие:

- функция имеет только один результат выполнения (однако может иметь несколько входных параметров);
- результат выполнения функции должен быть обозначен именем функции, т.е. внутри подпрограммы–функции должна иметь место конструкция следующего вида:

Имя функции := значение (результат вычислений).

Обращение к функции из основной программы или другой подпрограммы осуществляется непосредственно в выражении с указанием имени функции со списком фактических параметров:

<Переменная>:=<имя функции>(список фактических параметров).

Порядок обращения к подпрограмме–функции следующий.

Если при компиляции программы встречается имя подпрограммы (процедуры или функции), это имя отыскивается в описании.

В описании подпрограммы формальным параметрам присваиваются соответствующие фактические. Формальные параметры заменяются на фактические, после чего выполняется тело подпрограммы. Результат выполнения функции присваивается имени функции и передается в основную программу.

Следует запомнить. Так как происходит замена формальных параметров на фактические, то число , тип и порядок следования формальных и фактических параметров должен обязательно совпадать.

Пример. Составить программу вычисления затрат тепла на образование силикатов при варке стекла по выражению

$$QS_j = (100 - B_j) \cdot \sum_{i=1}^{n} Q_i \cdot P_i / 100 \ ,$$

где B_j – массив значений стеклобоя, j = 1,2, … 5; Q_i – удельный расход тепла, i = 1,2, … 6; P_i – концентрация реагентов в шихте, i = 1,2, … 6.

```
Program Teplo;
 type mas1=array[1..5]of real;
      mas2=array[1..6]of real;
 var QS,B:mas1;
     Q,P:mas2;
     i,j:integer;
Function Sum(Q,P:mas2):real;
 var i:integer;
     s:real;
Begin
 s:=0.0;
 for i:=1 to 5 do
  s:=s+Q[i]□P[i];
  Sum:=s;
end;
 {Основная программа}
Begin
 for i:=1 to 6 do
  readln (Q[i],P[i]);
 for j:=1 to 5 do
  readln (B[j]);
 for j:=1 to 6 do
  begin
   {Обращение к функции Sum(Q,P)}
   QS[j]:=(100–B[j])□Sum(Q,P)/100;
   writeln ('QS', j, '=',QS[j]:10:2);
  end;
End.
```

4.6. МОДУЛИ

Наличие модулей в Turbo-Pascal позволяет программировать и отлаживать программу по частям, создавать библиотеки программ и данных. Набор процедур и функций, объединенных в один блок (UNIT), может компилироваться независимо от главной программы. Благодаря этому, время компиляции для больших программ существенно сокращается. Модульный принцип построения особенно важен при

разработке программ расчета сложных химико-технологических процессов (ХТП), математическое описание которых представляет собой комплекс математических описаний блоков ХТП.

4.6.1. СТРУКТУРА МОДУЛЯ

Модуль состоит из следующих частей:

-заголовка модуля;

-интерфейса модуля;

-исполнительной (реализационной) части модуля;

-секции инициализации.

Все разделы модуля, за исключением секции инициализации, являются обязательными.

СТРУКТУРА МОДУЛЯ

<Заголовок модуля>

UNIT <ИМЯ МОДУЛЯ>;

<Интерфейсная часть>

INTERFACE {начало раздела объявлений} ;
USES <СПИСОК ИСПОЛЬЗУЕМЫХ МОДУЛЕЙ>;
LABEL
CONST {открытые объявления}
TYPE
VAR
PROCEDURE {только заголовки}
FUNCTION

<Исполнительная (реализационная) часть>

IMPLEMENTATION
USES <ИСПОЛЬЗУЕМЫЕ ПРИ РЕАЛИЗАЦИИ МОДУЛИ>;
LABEL
CONST {собственные объявления}
TYPE
VAR
PROCEDURE
FUNCTION {тела процедур и функций}

<Инициализационная часть>

BEGIN

…..

END.

Указанная последовательность разделов обязательна.

Заголовок модуля состоит из зарезервированного слова **unit** и имени модуля. Имя модуля должно быть единственным. Модуль должен быть помещен в файл, имя которого совпадает с именем модуля, а его расширение- **.pas.**

Пример заголовка: **UNIT mod;**

Имя модуля не может состоять более чем из восьми символов.

Интерфейсная часть начинается словом **interface.** Через интерфейс осуществляется взаимодействие основной программы с модулем (модуля с модулем).

В интерфейсе указываются константы, типы, переменные, процедуры и функции, которые могут быть использованы основной программой (модулем) при вызове этого модуля.

В разделе объявления процедур и функций указываются лишь заголовки подпрограмм. Сами подпрограммы приводятся в исполнительной части.

Исполнительная (реализационная) часть начинается словом **implementation** и заканчивается словом **end.**

Эта часть включает все программы модуля, а также локальные метки, константы, типы, переменные. Раздел **uses** необязателен. Если какой-то модуль уже указан в интерфейсе модуля, то в исполнительной части его повторять не следует.

За разделами объявления локальных меток, локальных типов, локальных переменных идут описания подпрограмм модуля (тела процедур и функций).

Инициализационная часть. Если между ключевыми словами **implementation** и **end** появляется **begin,** то полученный составной оператор **begin……end** представляет раздел инициализации модуля.

Этот раздел обычно используется для открытия файлов (например с помощью процедуры Assign) и для формирования структур данных и переменных. Например:

```
begin
 Assign (f1, Dan.dat);
end.
```

Инициализационная часть – это основной блок модуля. Операторы, приведенные в ней, выполняются после запуска программы первыми, т.е. перед операторами основного блока главной программы, в которую включен данный модуль.

Использование модуля в основной программе. Для использования модулей в программах, следует их имена указать после слова **USES**.

Например**: USES crt, mod;**

После этого в основной программе можно использовать идентификаторы, указанные в интерфейсах перечисленных модулей.

Разработанный модуль помещается в файл с именем, имеющим расширение **.pas**, например **mod.pas.**

Имя модуля в заголовке (Unit mod) должно совпадать с именем файла. Модуль транслируется отдельно, получает расширение .tpu. Например, mod.tpu.

При трансляции основной программы все используемые в ней модули (tpu-файлы) подсоединяются автоматически.

Пример 4.1. Вычислить молекулярную массу смеси по формуле:

$$MS = \sum_{i=1}^{4} MM_i \cdot МД_i ,$$

а также скорость реакции по выражению

$$W = k \cdot C_1 \cdot C_{2_2} \ ,$$

где MM_i – молекулярная масса i-того компонента;

$МД_i$ – мольная доля i–того компонента,

$$k = k_0 \cdot e^{\frac{-E}{RT}}$$

Для расчета средней молекулярной массы и константы скорости сформировать модуль.

Модуль

```
UNIT MOL;              {заголовок модуля}
INTERFACE              {раздел интерфейса}
 Type mas=array[1..4] of real;
 Var MD,MM:mas;
    i:integer;
 {объявление функции и процедуры}
 Function K(ko,e,r,t:real):real;
```

```
 Procedure MASS(MM,MD:mas;var MS:real);
IMPLEMENTATION          {раздел реализации}
 Function K(k0,e,r,t:real):real;
  var k1:real;
      i:integer;
 begin
  k1:=k0*exp(-e/(r*t));
  k:=k1
 end;
 Procedure MASS(MM,MD:mas;var MS:real);
  var i:integer;
 begin
  MS:=0;
  for i:=1 to 4 do
   MS:=MS+MM[i]*МД[i];
 end
end.
```

Текст модуля записывается в файл с именем **MOL.pas** и транслируется.

Основная программа

```
Program MolMas;
 Uses Mol;          {подключение модуля}
  Var MM,МД:mas;
      C1,C2,W,k0,e,r,t,ms:real;
begin
 write('Введите k0 e r t C1 C2');
 readln(k0,e,r,t,C1,C2);
 for i:=1 to 4 do
  readln(MM[i],MD[i]);
 MASS(MM,МД,MS);       {обращение к процедуре}
 writeln('Значение молекулярной массы=', MS:7:3);
```

);{обращение к функции k(k0,e,r,t)}

writeln (’Значение скорости=’,W:10:5);

end.

Запишем программу в файл, например с именем **Skor.pas,** и запустим её на выполнение. В каталоге библиотеки, в которой вы работаете, должны находиться файлы: **Skor.pas ; Mol.pas; Mol.tpu, Skor.exe**

4.6.2. МОДУЛИ CRT, GRAPH

Богатство алгоритмических возможностей языка Турбо-Паскаль в значительной степени достигается благодаря использованию модулей. Так, все математические функции, в том числе sqr, sqrt, exp, ln и другие, описаны в модуле System, который автоматически (по умолчанию) подключается при компиляции программы.

Модуль представляет собой набор констант, типов данных, переменных, процедур и функций, что позволяет использовать его как своеобразную библиотеку описаний (как правило, используются процедуры и функции).

Турбо-Паскаль располагает восемью стандартными модулями. Это System, Dos, Overlay, Graph, Crt, Printer, Turbo3, Graph3. Два последних модуля предназначены для совместимости программ, написанных в версии 3.0. Файл, содержащий модуль, имеет расширение *.tpu. Все перечисленные модули (кроме Graph, Graph3, Turbo3) объединены и хранятся в файле Turbo.tpl.

Модуль System поддерживает все стандартные процедуры и функции, обеспечивает ввод-вывод данных, обработку строк, динамическое распределение оперативной памяти и ряд других возможностей Турбо-Паскаля.

Модуль Dos содержит многочисленные стандартные процедуры и функции, многие из которых по своему действию эквивалентны соответсвующим командам MS-DOS (GetTime, DiskSize и др.).

Модуль Overlay обеспечивает поддержку оверлеев.

Модуль Crt поддерживает ряд стандартных процедур и функций, которые обеспечивают работу с экраном дисплея в текстовом режиме, управление звуком и работу с клавиатурой.

Модуль Printer содержит драйвер печатающего устройства и позволяет организовывать вывод информации на принтер.

Модуль Graph обеспечивает работу с экраном дисплея в графическом режиме.

Для того чтобы использовать модули в программах, их имена следует указать в предложении uses, всегда находящемся после заголовка программы. Например,

program Pr;
uses Crt, Graph;
...

Модуль Crt.

Работа с экраном в текстовом режиме

Практически любая программа использует дисплей для отображения вводимой и выводимой информации. Всю выводимую на экран дисплея информацию подразделяют на текстовую и графическую, соответственно выделяют текстовый и графический режимы.

Для инициализации (установки) текстового режима используется процедура

TextMode(Mode:word). Выполнение этой процедуры приводит к очистке экрана и активации указанного режима. Для задания режимов экрана используются следующие константы.

Наименование константы	Значение константы	Размер экрана	Цвет символов	Вид адаптера
Bw40	0	40×25	ч/б	цветной
Co40	1	40×25	цветной	цветной
Bw80	2	80×25	ч/б	цветной
Co80	3	80×25	цветной	цветной
Mono	4	80×25	ч/б	ч/б
Font8x8	256	80×43 (EGA)	цветной	цветной
		80×50 (VGA)	цветной	цветной

Например, TextMode(2) –черно-белый текстовый режим, 25x40.

TextColor(Color:byte) - устанавливает цвет выводимых на экран символов.

TextBackGround(Color:byte) - устанавливает цвет фона, т.е. цвет области, которая окружает отображаемый на экране символ.

ClrScr - очищает активное окно и устанавливает курсор в верхний левый угол.

GotoXY(x,y:byte) - перемещает курсор в позицию с координатами X, Y в рамках активного окна.

WhereX - возвращает X-координату текущей позиции курсора.
WhereY - возвращает Y-координату текущей позиции курсора.

Пример 1. Массив X(N) напечатать на экране в виде столбцов по M элементов в каждом. Выделить элементы, превышающие по значению величину K.

```
program P1; uses Crt;
 var x:array[1..20]of real;
     i,N,M:integer; K:real;
Begin
 writeln(' N M K');readln(N,M,K);
 for i:=1 to N do
  begin write('X[',i,']=');readln(x[i]) end;
 ClrScr;
 for i:=1 to N do
  begin
   if x[i]>K then
    begin
     TextColor(14);
     TextBackGround(4);
    end
   else
    begin
     TextColor(15);
     TextBackGround(1);
    end;
   GotoXY(10□((i-1) div M)+1,(i-1) mod M+1);
   write(i:2,' ',x[i]:6:4)
  end;
 writeln
 End.
```

Работа с клавиатурой

Для работы с клавиатурой в Crt предусмотрены следующие функции:

KeyPressed:boolean - возвращает значение True, если на клавиатуре была нажата какая-либо клавиша. В противном случае эта функция возвращает значение False.

ReadKey:char - считывает символ с клавиатуры. Считываемый символ на экране не отображается.

Работа со звуком

Основные процедуры работы со звуком:

Sound(F:word) – генерирует звук частотой F гц.

NoSound – отключает динамик.

Delay(Time:word) – делает задержку (приостанавливает выполнение программы) на определенное количество миллисекунд .

Модуль Graph.
Работа с экраном в графическом режиме

Модуль Graph поддерживает графический режим работы дисплея. В этом режиме любое изображение на экране дисплея синтезируется из множества мельчайших элементов, называемых *пикселями*. Каждый пиксель представляет собой светящуюся точку таких размеров, при которых промежутки между отдельными пикселями отсутствуют. Если группа смежных пикселей светится, то они воспринимаются не как совокупность отдельных точек, а как сплошной участок. Таким образом, на экране дисплея может быть синтезировано любое графическое изображение.

В графическом режиме экран дисплея разделяется прямоугольной сеткой, каждый элемент которой имеет свои координаты. Левый верхний угол экрана имеет координаты (0,0). Значение левой координаты (X) увеличивается в горизонтальном направлении слева направо. Значение правой координаты (Y) увеличивается в вертикальном направлении сверху вниз. Количество точек по горизонтали и вертикали называется разрешающей способностью.

Координаты правой нижней границы экрана можно определить, используя функции **GetMaxX и GetMaxY.**

Реализация графического режима в ПЭВМ обеспечивается благодаря наличию специальной схемы (электронная плата), называемой графическим *адаптером.* ПЭВМ может комплектоваться следующими типами графических адаптеров: CGA, VCGA, EGA, VGA, Hercules, AT&T, PC-3270, IBM-8514. Работу графического адаптера поддерживает специальная программа, называемая *драйвером.* Загрузочный модуль драйвера хранится в специальном файле с расширением bgi. Используемый адаптер может функционировать в различных режимах.

Адаптер	Режим	Разрешающая способность монитора	Число цветов
1.CGA (1)	0-3	320□200	4
	4	640□200	2
2.EGA (3)	0	640□200	16
	1	640□350	16
3.Hercules (7)	0	720□348	2

4.VGA (9)	0	640□200	16
	1	640□480	16

Модуль Graph подключается при помощи USES Graph;. С момента подключения модуля становятся доступными все находящиеся в нем модули.

Процедура инициализации:

InitGraph(var GraphDriver:integer ; {тип адаптера}
Var GraphMode:integer; {режим графики}
Var DriverPath:string {путь к драйверу}) .

Рассмотрим пример инициализации (установки) графического режима:

```
uses Graph;
 var
 Gd,Gm:integer;
begin
 Gd:=Detect;
 InitGraph(Gd,Gm,'');
 if GraphResult<>grOk then Halt(1);
 ...
 CloseGraph
end.
```

Графический режим инициализируется с помощью стандартной процедуры **InitGraph**. При этом переменным Gd и Gm необходимо указать номер адаптера и номер графического режима. Если переменной Gd предварительно присвоить значение константы Detect, описанной в модуле Graph (ее значение 0), то при загрузке драйвера программа выполнит автоматическое распознавание типа адаптера. При этом, если есть выбор графических режимов, устанавливается тот из них, который обеспечивает более высокое качество изображения. Третий параметр процедуры InitGraph - путь до файла с загрузочным модулем драйвера. Если путь отсутствует, то поиск этого файла будет осуществляться в текущем каталоге. Ошибки, которые могут возникать при инициализации графического режима, анализируют с помощью функции GraphResult. Для выхода из графического режима используется стандартная процедура CloseGraph. Эта процедура восстанавливает режим, существовавший до инициализации графики.

Для создания графических изображений модуль Graph предоставляет широкий набор процедур и функций.

Вывод точки и линии.

PutPixel(X,Y:integer;Color:word) - ставит на экране точку с координатами (X,Y) цвета Color.

Line(X1,Y1,X2,Y2:integer) - выводит на экран линию, соединяющую точки с координатами (X1,Y1) и (X2,Y2).

Rectangle(X1,Y1,X2,Y2:integer) - выводит на экран изображение прямоугольника с координатами диагонали (X1,Y1) и (X2,Y2).

Circle(X,Y:integer;Radius:word) - выводит на экран изображение окружности с координатами центра (X,Y) и радиусом (Radius).

Ellipse(X,Y :integer; StAngle, EndAngle, XRadius, YRadius: word) - выводит на экран изображение эллиптической дуги с центром в точке (X,Y) от начального угла StAngle до конечного угла EndAngle с горизонтальной полуосью XRadius и вертикальной YRadius. Отсчет углов осуществляется относительно горизонтальной оси в направлении против часовой стрелки. (3 часа - 0, 12 часов - 90 и т.д.) Если StAngle=0, а EndAngle=360, то будет выведено изображение полного эллипса.

SetColor(Color:integer) -устанавливает цвет линий**.**

SetLineStyle(LineStyle,Pattern,Thickness:Word) –задает тип (толщину) линии. Первый параметр изменяется от 0 до 4 и определяет тип линии. В модуле Graph описаны следующие константы:

- SolidLn - 0 - непрерывная линия,
- DottedLn - 1 - точечная линия,
- CenterLn - 2 - штрихпунктирная линия,
- DashedLn - 3 - штриховая линия,
- UserBitLn - 4 - тип линии, указываемый пользователем.

Второй параметр задает тип линии, он необходим, если первый параметр равен 4. Третий параметр задает толщину линии.

- NormWidth - 1 - тонкая линия,
- ThickWidth - 3 - толстая линия.

Рассмотрим пример:

```
uses Graph;
 var Gd,Gm:integer;
begin
 Gd:=Detect;
 InitGraph(Gd,Gm,'c:\bp\bgi');
 rectangle(100,50,200,100);
 line(100,50,200,100);
 line(100,100,200,50);
 ellipse(150,75,0,360,50,25);
 readln;
 CloseGraph
end.
```

Закрашенные области

Bar(X1,Y1,X2,Y2:integer) - выводит на экран закрашенный прямоугольник с координатами диагонали (X1,Y1) и (X2,Y2).

Bar3D(X1,Y1,X2,Y2:integer;Depth:word;Top:boolean) - выводит на экран изображение закрашенного прямоугольного параллелепипеда, который рисуется в изометрическом изображении с глубиной Depth. Параметр Top определяет, рисовать ли верхнюю грань параллелепипеда. Значение Top выбирается из соответствующего списка констант модуля Graph. Если рисовать, Top =TopOn, если нет - Top = TopOff.

FloodFill(X,Y:integer;Border:word) - заполняет (закрашивает) ограниченную область текущем цветом. Граница закрашиваемой области высвечивается цветом, заданным в Border.

PieSlice(X,Y:integer;StAngle,EndAngle,Radius:word) - выводит на экран изображение закрашенного сектора круга, используя в качестве центра круга точку (X,Y), начального угла StAngle, конечного угла - EndAngle и радиуса - Radius. Контур сектора высвечивается текущим цветом. Если StAngle=0, а EndAngle=360 градусам, то PieSlice выводит на экран закрашенную окружность.

Sector(X,Y:integer;StAngle,EndAngle,XRadius,YRadius:word)- выводит на экран изображение эллиптического сектора, используя в качествецентра круга точку (X,Y), начального угла StAngle, конечного угла - EndAngle, а в качестве горизонтальной и вертикальной полуосей - XRadius и YRadius. Контур сектора высвечивается текущим цветом. Если StAngle=0, а EndAngle=360, то на экране будет выведено изображение закрашенного эллипса.

Тип и цвет закраски можно установить с помощью процедуры SetFillStyle (Pattern,Color: Word). В модуле Graph предопределены следующие типы закрасок.

Константа	Значение	Константа	Значение
EmptyFill	0	LtBkSlashFill	6
SolidFill	1	HatchFill	7
LineFill	2	XhatchFill	8
LtSlashFill	3	InterleaveFill	9
SlashFill	4	WideDotFill	10
BkSlashFill	5	CloseDotFill	11

Вывод текстовой информации

В графическом режиме вывод текстовой информации осуществляется с помощью штриховых и побитовых шрифтов. Каждый символ в штриховом шрифте определен серией отрезков, что позволяет использовать любой коэффициент увеличения символов без ухудшения качества изображения. Побитовый шрифт определен матрицей 8□8 пикселей для каждого символа. Для увеличения побитового шрифта

используется коэффициент масштабирования, однако, большое увеличение побитового шрифта делает изображение грубым. Каждый штриховой шрифт хранится в соответствующем файле с расширением CHR.

OutTextXY(X,Y:integer;Text:string) - выводит строку, начиная с точки, имеющей координаты (X,Y).

SetTextJustify(Horiz,Vert:word) - устанавливает значения выравнивания текста. Для установки значения выравнивания в модуле Graph определены следующие константы:

а) горизонтальное выравнивание;

- LeftText = 0 - выравнивание слева,
- CenterText = 1 - выравнивание по центру,
- RightText = 2 - выравнивание справа,

б) вертикальное выравнивание;

- Bottom Text = 0 - выравнивание снизу,
- CenterText = 1 - выравнивание по центру,
- TopText = 2 - выравнивание сверху.

SetTextStyle(Font,Direction,CharSize:word) - устанавливает текущий шрифт, тип и коэффициент увеличения символов. Параметр Font - тип шрифта. Для установки типа шрифта в модуле Graph описаны следующие константы:

- DefaultFont = 0 - побитовый шрифт ,
- TriplexFont = 1 - тройной шрифт,
- SmallFont = 2 - малый шрифт,
- SansSerifFont = 3 - гротесковый шрифт,
- GothicFont = 4 - готический шрифт

и другие.

Direction задает направление вывода (0 - горизонтальное, слева направо, 1 - вертикальное, снизу вверх). CharSize - коэффициент увеличения символов.

4.6.2.1. Построение графиков

График, как и любое графическое изображение, строится в Турбо-Паскале с помощью стандартных процедур модуля Graph, которые были описаны выше. Однако при этом возникают сложности, заключающиеся в следующем. Допустим, мы строим график функции $y=\sin x$ на интервале

$[-\pi,\pi]$. Поскольку современные мониторы имеют разрешение 640 точек по горизонтали на 480 точек по вертикали (VGA), то, очевидно, что если мы не увеличим график, то он будет располагаться в верхнем левом углу и

практически не будет виден. Кроме того, часть точек, имеющих отрицательную координату *x* или *y*, вообще не будут отображаться на экране. Есть и еще одна проблема: нумерация точек на экране монитора по вертикали ведется сверху вниз, в то время как при построении графика значения *y* должны увеличиваться снизу вверх.

Таким образом, при построении графика возникает задача масштабирования изображения.

Рассмотрим рисунок (рис.34). Здесь внешняя рамка обозначает экран монитора, внутренняя – ту область экрана, в которой должен быть нарисован график. x_1, x_2 – границы интервала, на котором строится график (в данном случае $x_1 = -\pi$, $x_2 = \pi$). y_1, y_2 – пределы изменения функции на выбранном интервале (в данном случае $y_1 = -1$, $y_2 = 1$). N*x*1, N*x*2 – левая и правая границы рамки, N*y*1, N*y*2 – верхняя и нижняя границы. Для составления программы нам потребуются формулы, которые по известным значениям *x* и *y* позволяют определить N*x* и N*y* – координаты точки на экране монитора. В общем виде эти формулы могут быть записаны следующим образом:

$$Nx = \Delta N_x + M_x \cdot (x - x_1) \; ; \qquad Ny = \Delta N_y - M_y \cdot (y - y_1),$$

где ΔN_x и ΔN_y выражают смещение графика вдоль осей OX и OY; M_x, M_y – масштабные коэффициенты осей OX и OY; знак "–" во второй формуле выражает изменение направления оси OY.

Значения ΔN_x и M_x могут быть найдены из условий: при $x=x_1$ N*x* = Nx_1, при $x=x_2$ N*x* = Nx_2. ΔN_x = Nx_1, а $M_x = (Nx_2 - Nx_1) / (x_2 - x_1)$.

Значения ΔN_y и M_y могут быть найдены из условий: при $y=y_1$ N*y* = Nx_2, при $y=y_2$ N*y* = Ny_2. ΔN_y = Ny_2, а $M_y = (Ny_2 - Ny_1) / (y_2 - y_1)$.

Пример программы построения графика функции $y=\sin x$ приведен ниже.

```
uses Graph;
{Описание функции y=sin(x)}
function f(x:real):real;
begin f:=sin(x) end;
var  gd,gm,Nx,Nx1,Nx2,Ny,Ny1,Ny2:integer;
       x,x1,x2,h,y,y1,y2:real;
BEGIN
 {Переход в графический режим}
```

```
gd:=Detect;
InitGraph(gd,gm,'c:\bp\bgi');
x1:=0; x2:=4*Pi; h:=0.01; y1:=-1; y2:=1;
{Размеры рамки}
Nx1:=10; Nx2:=GetMaxX-Nx1;
Ny1:=10; Ny2:=GetMaxY-Ny1;
{Зарисовка рамки}
rectangle(Nx1,Ny1,Nx2,Ny2);
{Построение графика по точкам}
x:=x1;
repeat
y:=f(x);
{Вычисление координаты точки
 экрана по заданным  x и y}
Nx:=round(Nx1+(Nx2-Nx1)*(x-x1)/(x2-x1));
Ny:=round(Ny2-(Ny2-Ny1)*(y-y1)/(y2-y1));
PutPixel(Nx,Ny,12);
x:=x+h
until x>x2;
readln; {Пауза}
closegraph {Выход из графического режима}
END.
```

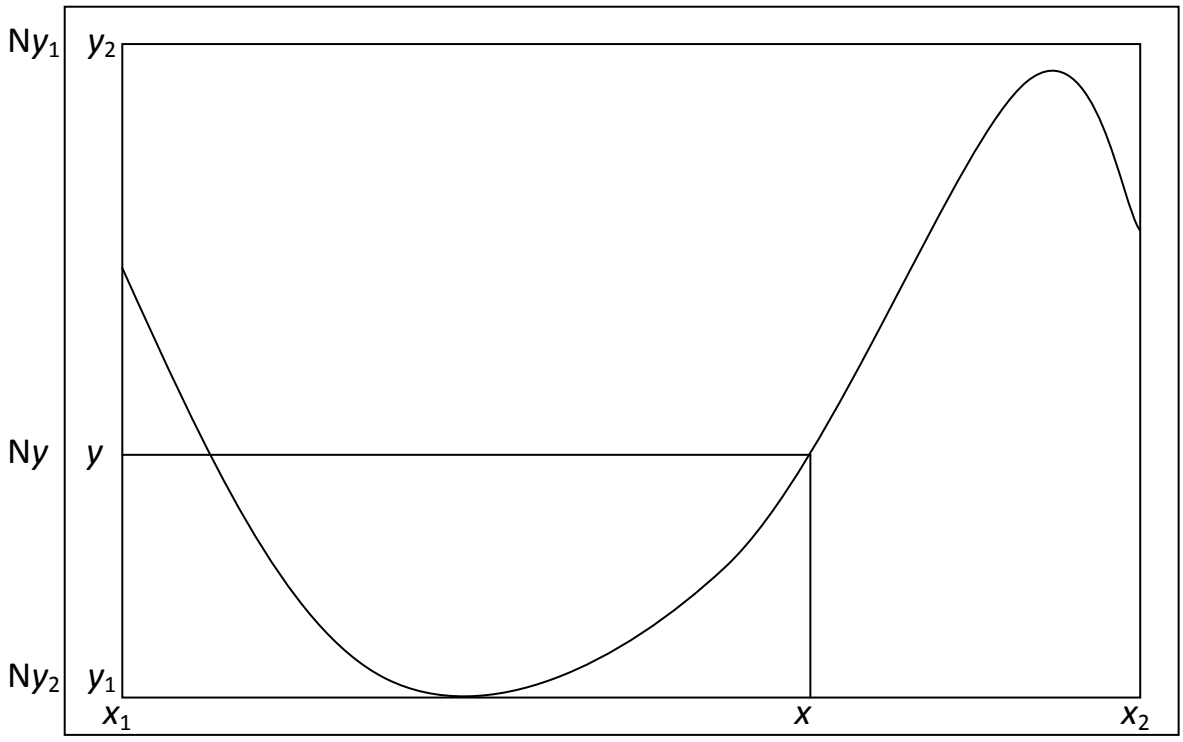


Рис.34. График функции *y*=sin *x*

4.7. ПРОГРАММИРОВАНИЕ ТИПОВЫХ АЛГОРИТМОВ ВЫЧИСЛЕНИЙ

Вычисление суммы и произведения

Пусть требуется вычислить сумму значений некоторой последовательности $s = a_1 + a_2 + ... + a_{20} = \sum_{i=1}^{20} a_i$,

где a_i – массив исходных данных.

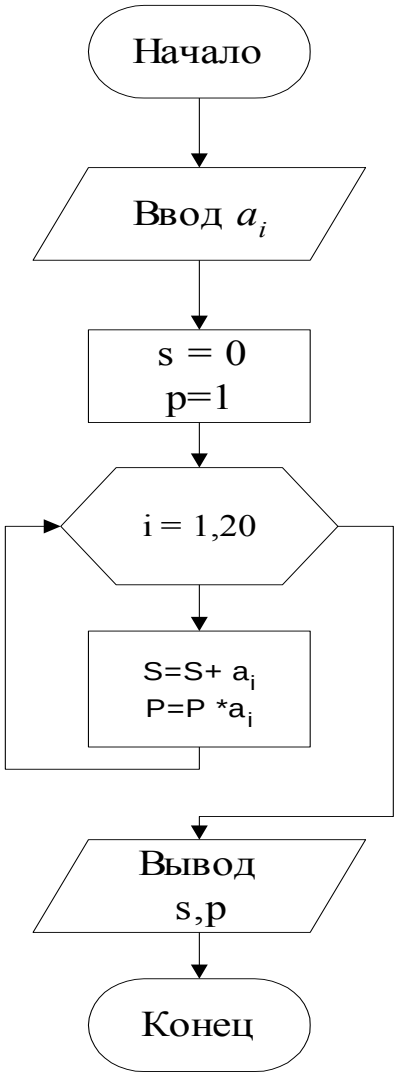
При вычислении суммы используется прием накопления по выражению

$$s = s + a_i,\ i = 1,2,..., n.$$

По данному выражению каждое новое значение вычисляется добавлением полученного слагаемого к сумме предыдущих. Начальное значение суммы принимается равным нулю (s = 0). При первом выполнении цикла (i=1) вычисляется значение s = 0 + a[1], на втором шаге (i = 2) – значение s = (0 + a[1]) + a[2] или s = s + a[2]. В результате повторения этой операции 20 раз получим искомую сумму.

Аналогично вычисляется и произведение $z = a_1 \cdot a_2 \cdot \ldots \cdot a_n = \prod_{i=1}^{20} a_i$, но с той разницей, что для его накопления используется выражение $P = P \cdot a_i$, а начальное значение произведения должно быть равно единице.

Схема алгоритма и программа вычисления суммы и произведения




```
Program SumPr;
var a:array[1..20]of real;
    i: integer;
    P,s: real;
Begin
 for i:=1 to 20 do
  begin
   writeln('Введите a',i);
   readln(a[i]);
  end;
 s:=0.0; p:=1;
 for i:=1 to 20 do
 begin
 s:=s+a[i];
 p:=p·a[i];
 end;
 writeln('s=',s:10,
        ' p=',p:10);
End.
```

Нахождение наибольшего (наименьшего) значения и порядкового номера

Пример. В массиве плотностей десяти химических элементов (d_1, d_2, ..., d_{10}) найти элемент с наибольшей плотностью (d_{max}) и его порядковый номер (n).

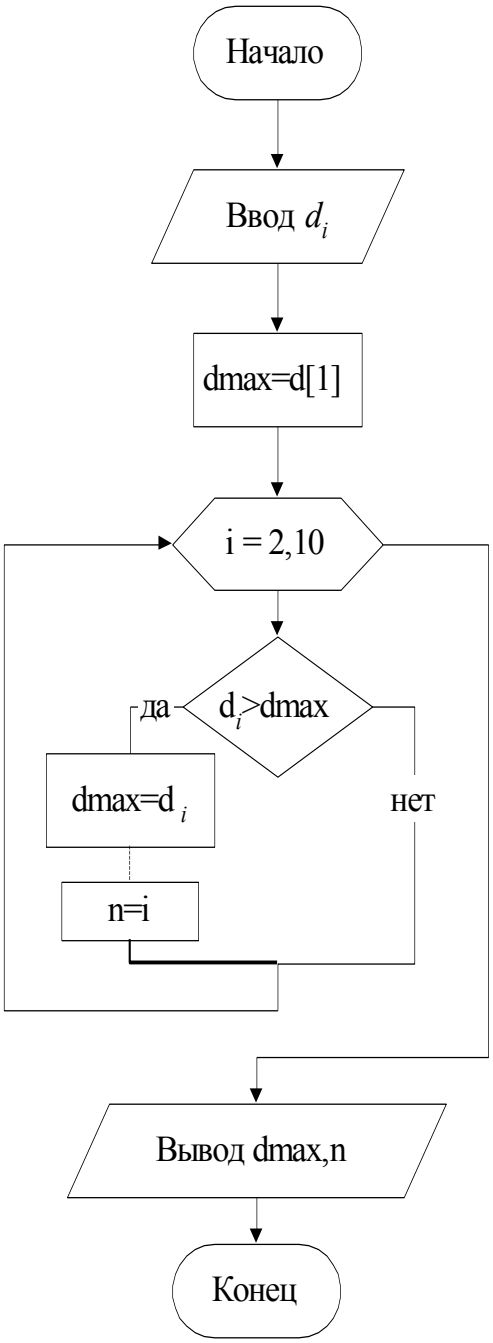
Нахождение наибольшего (наименьшего) значения из последовательности чисел осуществляется при помощи алгоритма

попарного сравнения. В качестве начального значения наибольшего (наименьшего) принимается первый элемент массива, с которым сравниваются все остальные элементы массива. Если сравниваемый член последовательности больше d_{max} , то d_{max} присваивается его значение, в противном случае d_{max} остается без изменения. Можно за начальное приближение d_{max} принять число, которое заведомо меньше (больше) всех элементов массива, например 10^{-5}.

В том случае, когда не требуется нахождение самого значения максимального (минимального) элемента массива, а требуется определить только его номер, алгоритм поиска номера будет следующий:

```
Nmax:=1;
for i:=2 to 10 do
if d[i]> d[Nmax] then Nmax:= i;
```

Схема и программа вычисления наибольшего значения и его номера

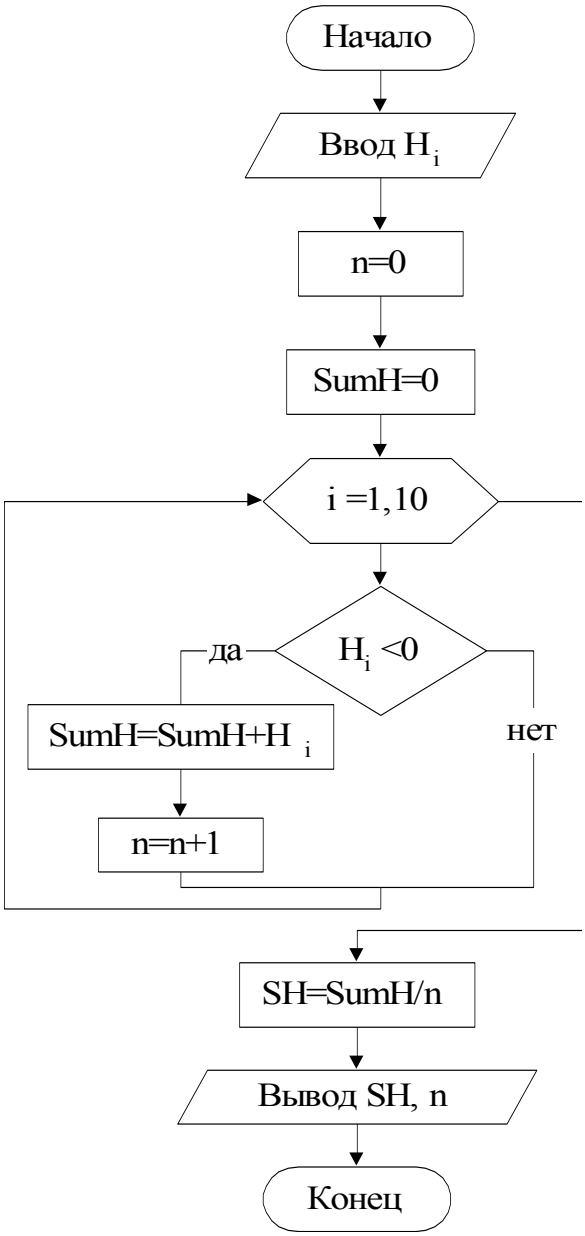



```
Program Max;
 var d:array[1..10]of
              real;
    dmax:real;
    i,n:integer;
Begin
 for i:=1 to 10 do
  readln(d[i]);
 dmax:=d[1];
 for i:=2 to 10 do
  if d[i]>dmax then
   begin
    dmax:=d[i];
    n:=i
   end;
 writeln('dmax=',
 dmax:10:3,' n=',n:3);
End.
```

Нахождение количества чисел

Пример. Вычислить средний тепловой эффект (SH) экзотермических реакций (H1, H2, ..., H10) химико–технологического процесса.

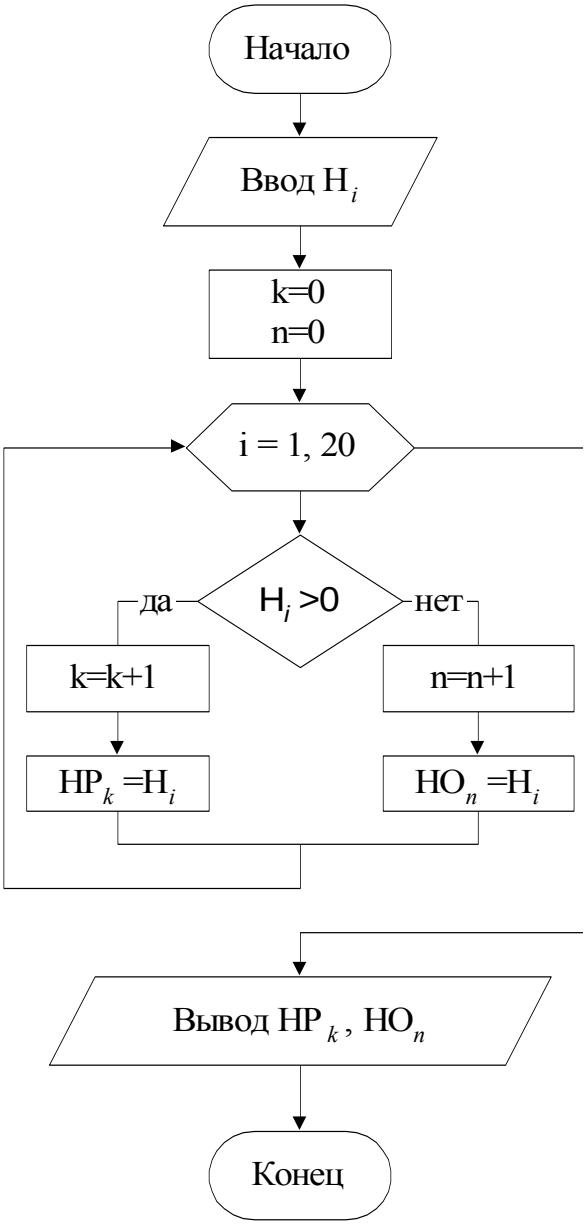
Для того, чтобы найти средний тепловой эффект, необходимо найти суммарный тепловой эффект (SumH) и количество реакций (n) с отрицательными значениями ΔHi .




```
Program Tepl;
var H:array[1..10]of
                real;
   SumH,SH:real;
   i,n:integer;
Begin
 for i:=1 to 10 do
  readln(H[i]);
 SumH:=0; n:=0;
 for i:=1 to 10 do
  if H[i]<0 then
   begin
    Sum:=SumH+H[i];
    n:=n+1;
   end;
 SH:=SumH/n;
 writeln('SH=',SH:10;
        ' n=',n:3);
End.
```

Преобразование массивов

Пример. Дан массив энтальпий экзо– и эндотермических реакций образования химических соединений: H_1, H_2, ..., H_{20}. Сформировать массив (Hp_1, Hp_2, ..., $Hp_к$), состоящий из положительных значений энтальпий и массив, состоящий из отрицательных значений энтальпий (Ho_1, Ho_2, ...,Ho_n). Обозначим: k, n – количество положительных и отрицательных элементов в массивах Hp и Ho соответственно.




```
Program Entalp;
type mas=array[1..20]of
                  real;
var H,Hp,Ho:mas;
   k,n,i:integer;
Begin
 for i:=1 to 20 do
  readln(H[i]);
 k:=0;n:=0;
 for i:=1 to 20 do
  if H[i]>0 then
   begin k:=k+1;
    Hp[k]:=H[i];
   end
  else
   begin n:=n+1;
    Ho[n]:=H[i];
   end;
 for i:=1 to k do
  writeln(Hp[i]:10,' ');
 writeln;
 for i:=1 to n do
  writeln(Ho[i]:10,' ');
End.
```

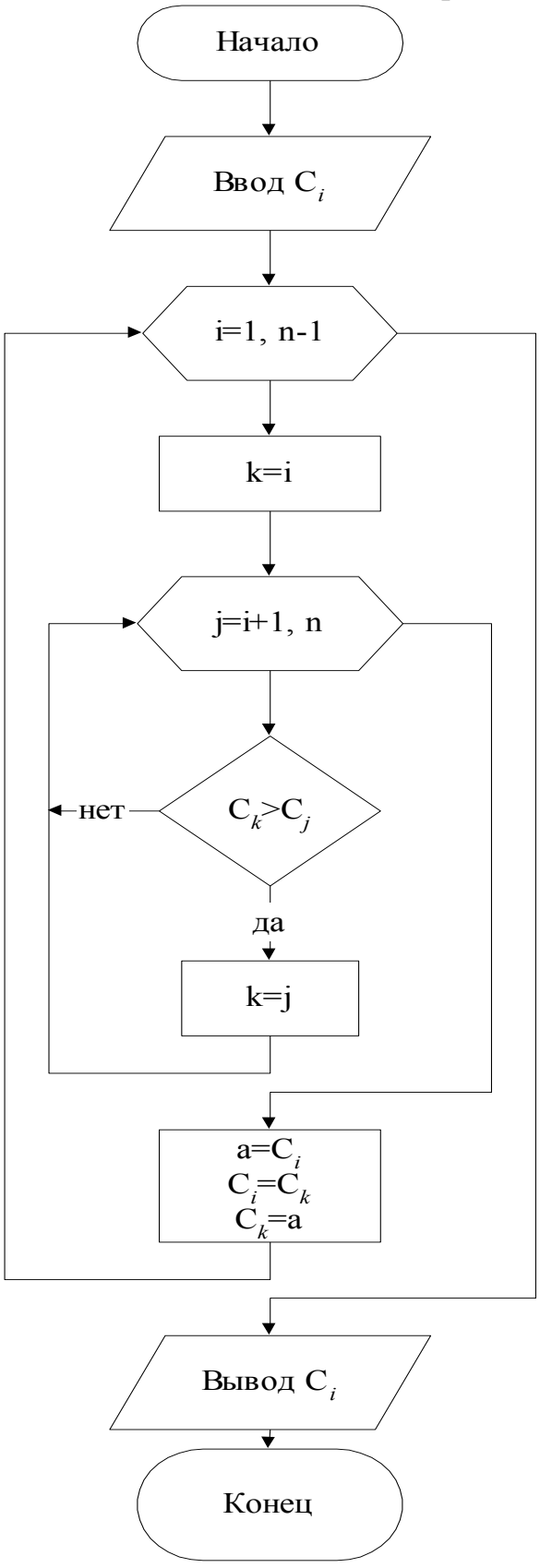
Сортировка данных методом отыскивания наименьшего элемента

Одним из наиболее важных и часто выполняемых на ЭВМ задач является сортировка данных, т.е. расположение элементов в соответствии с заданным порядком. Известно достаточно большое количество методов сортировки. Рассмотрим один из них: сортировку методом отыскивания наименьшего элемента.

Пример: Дана смесь углеводородов C_1, ..., C_5. Расположить компоненты в порядке возрастания по числу атомов углерода в молекуле.

Массив исходных данных: 4 3 1 5 2.

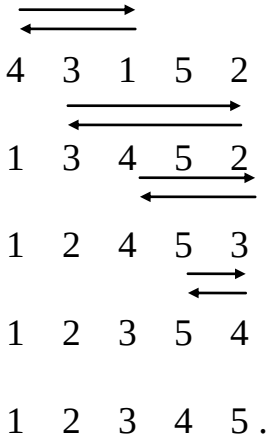
Обозначим: i – место, на которое устанавливается очередной минимальный элемент; j – текущий номер элемента в усеченной последовательности; k – номер минимального элемента.




```
Program Sort;
 var C:array[1..5]of
                integer;
    a,i,j,k:integer;
 const n=5;
Begin
 for i:=1 to n do
  readln(C[i]);
 for i:=1 to n–1 do
  begin
   k:=i;
   for j:=i+1 to n do
   if C[k]>C[j] then k:=j;
   a:=C[i];
   C[i]:=C[k];
   C[k]:=a;
  End;
 for i:=1 to 5 do
  writeln(C[i]);
End.
```

В последовательности чисел отыскивается наименьший элемент C_k и ставится на первое место. Первый элемент ставится на место минимального (т.е. C_1 и C_k меняются местами). Затем в усеченной последовательности C_2, ..., C_n отыскивается следующий наименьший элемент и ставится на второе место и т.д. (n–1) раз. Самый максимальный сдвигается в конец последовательности.

Сортировка осуществляется по следующей схеме:



Примеры алгоритмов работы с двумерными массивами. Вычисление суммы элементов матрицы

$$S = \sum_{i=1}^{n} \sum_{j=1}^{m} b_{ij} \ .$$

```
s:=0.0;
for i:=1 to n do
 for j:=1 to m do
  s:=s+b[i,j];
```

Вычисление суммы диагональных элементов матрицы (следа матрицы)

$$S = \sum_{j=1}^{n} b_{ii} \ .$$

```
s:=0.0;
for i:=1 to n do
 s:=s+b[i,i]
```

Вычисление суммы элементов строки матрицы

$S_i = \sum_{j=1}^{m} b_{ij}$, i = 1,..., n (количество строк) .

```
for i:=1 to n do
 begin
  s[i] :=0;
  for j:=1 to m do
   s[i]:=s[i]+b[i,j];
 end;
```

Вычисление суммы элементов столбца матрицы

$S_j = \sum_{i=1}^{n} b_{ij}$, j = 1,...,m (количество столбцов) .

```
for j:=1 to m do
 begin
  s[j]:=0;
  for i:=1 to n do
   s[j]:=s[j]+b[i,j];
 end;
```

Нахождение наибольшего (наименьшего) элемента в матрице:

```
max:=b[1,1];
for i:=1 to m do
 for j:=1 to n do
  if b[i,j]>max then max:=b[i,j];
```

Нахождение наибольшего элемента в строке матрицы

```
for i:=1 to m do
 begin
  max[i]:=b[i,1];
  for j:=1 to n do
  if b[i,j]>max[i] then max[i]:=b[i,j];
 end;
```

5. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ ХИМИЧЕСКИХ ЗАДАЧ

Инженеров химиков-технологов математика интересует не сама по себе, а как средство решения конкретных инженерных задач. Рассмотрим, как может решаться любая инженерная практическая задача, допустим, определение максимального выхода продукта в химическом реакторе. Одним из способов решения является эксперимент. Можно построить установку, реактор даже завод и затем измерить интересующие нас характеристики. Если результат получится неудачным, то проект изменяется, создаются новые объекты. Таким образом будет достигнут требуемый результат, но слишком медленным и дорогим способом.

Другой способ решения – математический анализ объекта или явления. Но такой анализ применяется не к реальным явлениям, а к математическим моделям этих объектов и явлений.

Математическая модель химико-технологического процесса (ХТП) – совокупность математических структур: уравнений, неравенств и т.д., адекватно описывающая исследуемые свойства объекта[ ].

Математическое моделирование ХТП – исследование свойств процесса на математических моделях с целью предсказания результатов его протекания в реальных условиях.

Метод математического моделирования включает в себя следующие этапы.

1. Формирование математической модели объекта (уравнения материального и теплового балансов ХТП).

2. Выбор метода решения полученной системы уравнений, формирование алгоритма и программы решения задачи. Проведение расчетов на ЭВМ.

3. Установление соответствия (адекватности) модели изучаемому процессу.

4. Рекомендации по наиболее эффективному проведению исследуемого процесса.

Для некоторых моделей удается получить аналитическое решение. Однако большинство практических задач в химии и химической технологии не имеют аналитического решения. Основными методами решения таких задач являются численные методы, которые позволяют добиться хорошего количественного описания явления или процесса.

Реализация численных методов требует большого объема вычислений. Поэтому применение методов вычислительной математики возможно лишь при наличии современных компьютеров.

В настоящей главе рассматриваются численные методы решения нелинейных и линейных уравнений, методы обработки экспериментальных данных, решения дифференциальных уравнений и численное интегрирование.

5.1. Приближенные числа

5.1.1. Абсолютная и относительная погрешности

Приближенным числом a называется число, незначительно отличающееся от точного A и заменяющее последнее в вычислениях. Под *ошибкой* или *погрешностью* Δa приближенного числа a обычно понимают разность между соответствующим точным числом и данным приближенным, т.е.

$$\Delta a = A - a.$$

Если $A > a$, то ошибка положительна $\Delta a > 0$; если же $A < a$, то ошибка отрицательна $\Delta a < 0$. Во многих случаях знак ошибки неизвестен. Тогда целесообразно пользоваться *абсолютной погрешностью приближенного числа*

$$\Delta = |\Delta a|.$$

Определение 1. *Абсолютной погрешностью* □ приближенного числа *a* называется абсолютная величина разности между соответствующим точным числом *A* и числом *a*, т.е.

$$\square = | A - a |. \qquad (5.1.1)$$

На практике чаще всего число *A* нам не известно, и, следовательно, мы не можем определить и абсолютную погрешность □ по формуле (5.11). В этом случае вводят ее оценку сверху, так называемую *предельную абсолютную погрешность*.

Определение 2. *Под предельной абсолютной погрешностью* приближенного числа понимается всякое число, не меньшее абсолютной погрешности этого числа. Таким образом, если $\square_a$ - предельная абсолютная погрешность приближенного числа *a*, заменяющее точное *A*, то

$$\square = | A - a | \le \square_a. \qquad (5.1.2)$$

Отсюда следует, что точное число A заключено в границах

$$a - \square_a \le A \le a + \square_a. \qquad (5.1.3)$$

В этом случае для краткости пользуются записью

$$A = a \pm \square_a.$$

Пример 5.1.1. Определить предельную абсолютную погрешность числа *a*=3.14, заменяющего число π.

Решение. Так как имеет место неравенство $3.14 < \pi < 3.15$, то $| a - \pi | < 0.01$ и, следовательно, можно принять $\square_a = 0.01$. Однако если учесть, что $3.14 < \pi < 3.142$, то будем иметь лучшую оценку: $\square_a = 0.002$.

Заметим, что сформулированное выше понятие предельной абсолютной погрешности является весьма широким, а именно: *под предельной абсолютной погрешностью приближенного числа a понимается любой представитель бесконечного множества неотрицательных чисел* $\square_a$, *удовлетворяющих неравенству* (5.1.2). Отсюда логически вытекает, что всякое число, большее абсолютной погрешности данного приближенного числа, также может быть названо предельной абсолютной погрешностью этого числа. Практически удобно в качестве $\square_a$ выбирать возможно меньшее при данных обстоятельствах число, удовлетворяющее неравенству (5.1.2).

В записи числа, полученного в результате измерения, обычно отмечают его предельную абсолютную погрешность. Например, если

длина отрезка $l = 214$ *см* с точностью до 0.5 *см*, то пишут $l = 214$ *см* $\pm$ 0.5 *см*. Здесь предельная абсолютная погрешность $\Delta_l = 0.5$ *см*, а точная величина длины l отрезка заключена в границах $213.5\ см \le l \le 214.5\ см$.

Абсолютная погрешность (или предельная абсолютная погрешность) недостаточна для характеристики точности измерения или вычисления. Так, например, если при измерении длин двух стержней получены результаты $l_1 = 100.8$ *см* $\pm$ 0.5 *см* и $l_2 = 5.2$ *см* $\pm$ 0.5 *см*, то, несмотря на совпадение предельных абсолютных погрешностей, качество первого измерения выше, чем второго. Для точности данных измерений существенна абсолютная погрешность, приходящаяся на единицу длины, которая носит название *относительной погрешности*.

Определение 3. *Относительной погрешностью* δ приближенного числа a называется отношение абсолютной погрешности Δ к приближенному значению этого числа.

$$\delta = \frac{\Delta}{|A|}. \qquad (5.1.4)$$

Отсюда $\Delta = |A|\,\delta.$

Введем понятие *предельной относительной погрешности*.

Определение 4. *Предельной относительной погрешностью* δ_a данного приближенного числа a называется всякое число, не меньшее относительной погрешности этого числа.

$$\delta \le \delta_a, \qquad (5.1.5)$$

т.е. $\Delta / |A| \le \delta_a$, отсюда $\Delta \le |A| \cdot \delta_a.$

Таким образом, за предельную абсолютную погрешность числа a можно принять:

$$\Delta_a = |A|\,\delta_a. \qquad (5.1.6)$$

Так как на практике $A \approx a$, то вместо формулы (5.1.6) часто пользуются формулой

$$\Delta_a = |a|\,\delta_a. \qquad (5.1.7)$$

Отсюда, зная предельную погрешность δ_a, получают границы для точного числа. То обстоятельство, что точное число лежит между $a(1-\delta_a)$ и $a(1+\delta_a)$, условно записывают так:

$$A = a(1 \pm \delta_a).$$

Пример 5.1.2. Вес 1 л воды при 0°С P = 999.847 ± 0.001 г. Определить предельную относительную погрешность результата взвешивания.

Решение. Очевидно, что $\Delta_P = 0.001$ г. Следовательно, $\delta_P = 0.001/999.847 \approx 10^{-4}\%$.

Пример 5.1.3. При определении газовой постоянной для воздуха получили R=29.25. Зная, что относительная погрешность этого измерения равна 0.1 % найти пределы, в которых заключается R.

Решение. Имеем $\delta_R = 0.001$, тогда $\Delta_R = R\delta_R \approx 0.03$.Следовательно, $29.22 \le R \le 29.28$.

5.1.2. ОСНОВНЫЕ ИСТОЧНИКИ ПОГРЕШНОСТЕЙ

Погрешности, встречающиеся в математических задачах, могут быть в основном разбиты на пять групп.

1. Погрешности, связанные с самой постановкой математической задачи. Математические формулировки редко точно отображают реальные явления: обычно они дают лишь более или менее идеализированные модели. Как правило, при изучении тех или иных явлений природы мы вынуждены принять некоторые упрощающие задачу условия, что вызывает ряд погрешностей (*погрешности задачи*).

Иногда бывает и так, что решить задачу в точной постановке трудно или даже невозможно. Тогда ее заменяют близкой по результатам приближенной задачей. При этом возникает погрешность, которую можно назвать *погрешностью метода*.

2. Погрешности, связанные с наличием бесконечных процессов в математическом анализе. Функции, фигурирующие в математических формулах, часто задаются в виде бесконечных последовательностей или рядов, например $\left(\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots\right)$. Более того, многие математические уравнения можно решить, лишь описав бесконечные процессы, пределы которых и являются искомыми решениями. Так как бесконечный процесс, вообще говоря, не может быть завершен в конечное число шагов, то мы вынуждены остановиться на некотором члене последовательности, считая его приближением к искомому решению.

Понятно, что такой обрыв процесса вызывает погрешность, называемую обычно *остаточной погрешностью*.

3. Погрешности, связанные с наличием в математических формулах числовых параметров, значения которых могут быть определены лишь приближенно. Таковы, например, все физические константы. Условно назовем эту погрешность *начальной*.

4. Погрешности, связанные с системой счисления. При изображении даже рациональных чисел в десятичной системе или другой позиционной системе справа от запятой может быть бесконечное число цифр (например, может получиться бесконечная десятичная периодическая дробь). При вычислениях, очевидно, можно использовать лишь конечное число этих цифр. Так возникает *погрешность округления*. Например, полагая 1/3 = 0.333, получаем погрешность $\Delta \approx 3 \cdot 10^{-4}$. Приходится так же округлять и конечные числа, имеющие большое количество знаков.

5. Погрешности, связанные с действиями над приближенными числами (*погрешности действий*). Понятно, что, производя вычисления с приближенными числами, погрешности исходных данных в какой-то мере мы переносим в результат вычислений. В этом отношении погрешности действий являются неустранимыми.

При решении конкретной задачи те или иные погрешности иногда отсутствуют, или влияние их ничтожно. Но для полного анализа погрешностей следует учитывать все их виды.

5.1.3. ПОГРЕШНОСТЬ СУММЫ И РАЗНОСТИ

Теорема 1. Абсолютная погрешность алгебраической суммы нескольких приближенных чисел не превышает суммы абсолютных погрешностей этих чисел.

Доказательство. Пусть $x_1, x_2, \ldots, x_n$ - данные приближенные числа. Рассмотрим их алгебраическую сумму

$$u = \pm x_1 \pm x_2 \pm \ldots \pm x_n.$$

Очевидно, что

$$\Delta u = \pm \Delta x_1 \pm \Delta x_2 \pm \ldots \pm \Delta x_n.$$

И, следовательно,

$$|\Delta u| = \pm |\Delta x_1| \pm |\Delta x_2| \pm \ldots \pm |\Delta x_n|. \qquad (5.1.8)$$

Следствие. За предельную абсолютную погрешность алгебраической суммы можно принять сумму предельных абсолютных погрешностей слагаемых

$$\Delta_U = \Delta_{X1} + \Delta_{X2} + ... + \Delta_{Xn}.$$

Рассмотрим разность двух приближенных чисел $u = x_1 - x_2$. По формуле (5.1.8) предельная абсолютная погрешность Δ_U разности

$$\Delta_U = \Delta_{X1} + \Delta_{X2},$$

т.е. *предельная абсолютная погрешность разности равна сумме предельных абсолютных погрешностей уменьшаемого и вычитаемого.*

Отсюда предельная относительная погрешность разности

$$\delta = \frac{\Delta_{X1} + \Delta_{X2}}{A}. \qquad (5.1.9)$$

Замечание о потере точности при вычитании близких чисел. Если приближенные числа x_1 и x_2 достаточно близки друг к другу и имеют малые абсолютные погрешности, то число A мало. Из формулы (5.1.9) вытекает, что относительная погрешность в этом случае может быть весьма большой, в то время как погрешности уменьшаемого и вычитаемого остаются малыми, т.е. здесь происходит *потеря точности*.

ПОГРЕШНОСТЬ ПРОИЗВЕДЕНИЯ И ЧАСТНОГО

Теорема 2. *Относительная погрешность произведения нескольких приближенных чисел, отличных от нуля, не превышает суммы относительных погрешностей этих чисел.*

Доказательство. Пусть $u = x_1 x_2 ... x_n$. Предполагая для простоты, что приближенные числа положительны, будем иметь:

$$\ln u = \ln x_1 \ln x_2 ... \ln x_n.$$

Отсюда, используя приближенную формулу $\Delta \ln x = \frac{\Delta x}{x}$, находим:

$$\frac{\Delta u}{u} = \frac{\Delta x_1}{x_1} + \frac{\Delta x_2}{x_2} + ... + \frac{\Delta x_n}{x_n}.$$

Оценивая последнее выражение по абсолютной величине, получим:

$$\left|\frac{\Delta u}{u}\right| \le \left|\frac{\Delta x_1}{x_1}\right| + \left|\frac{\Delta x_2}{x_2}\right| + ... + \left|\frac{\Delta x_n}{x_n}\right|.$$

Если A_i (i = 1,2, ... ,n) - точные значения сомножителей x_i и $|\Delta x_i|$, как это бывает обычно, малы по сравнению с x_i, то приближенно можно положить:

$$\left|\frac{\Delta x_i}{x_i}\right| \approx \left|\frac{\Delta x_i}{A_i}\right| = \delta_i$$

и

$$\left|\frac{\Delta u}{u}\right| = \delta,$$

где δ_i - относительные погрешности сомножителей x_i (i=1,2, ... ,n) и δ - относительная погрешность произведения. Следовательно,

$$\delta \le \delta_1 + \delta_2 + ... + \delta_n. \qquad (5.1.10)$$

Формула (5.1.10), очевидно, остается верной также, если сомножители x_i (i=1,2, ... ,n) имеют различные знаки.

С л е д с т в и е. Предельная относительная погрешность произведения равна сумме предельных относительных погрешностей сомножителей, т.е.

$$\delta_U = \delta_{X1} + \delta_{X2} + ... + \delta_{Xn}.$$

Если $u = \frac{x}{y}$, то $\ln u = \ln x - \ln y$ и

$$\frac{\Delta u}{u} = \frac{\Delta x}{x} - \frac{\Delta y}{y}.$$

Отсюда

$$\left|\frac{\Delta u}{u}\right| = \left|\frac{\Delta x}{x}\right| + \left|\frac{\Delta y}{y}\right|.$$

Из последней формулы вытекает, что теорема 2 верна и для частного.

ОБЩАЯ ФОРМУЛА ДЛЯ ПОГРЕШНОСТИ

Основная задача теории погрешности заключается в следующем: известны погрешности некоторой системы величин, требуется определить погрешность данной функции от этих величин.

Пусть задана дифференцируемая функция

$$u = f(x_1, x_2, ... , x_n)$$

и пусть $|\Delta x_i|$ (i=1,2, ... ,n) - абсолютные погрешности аргументов функции. Тогда абсолютная погрешность функции

$$|\Delta u| = |f(x_1+\Delta x_1, x_2+\Delta x_2, ... , x_n+\Delta x_n) - f(x_1, x_2, ... , x_n)|.$$

Обычно на практике | Δx_i | - малые величины, произведениями, квадратами и высшими степенями которых можно пренебречь. Поэтому можно положить:

$$| \Delta u | \approx |df(x_1, x_2, \ldots, x_n)| = \left| \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \Delta x_i \right| \le \sum_{i=1}^{n} \left| \frac{\partial f}{\partial x_i} \right| |\Delta x_i| .$$

Итак,

$$| \Delta u | \le \sum_{i=1}^{n} \left| \frac{\partial f}{\partial x_i} \right| |\Delta x_i| .$$

Отсюда, обозначая через Δ_{Xi} (i=1, 2, ... , n) предельные абсолютные погрешности аргументов x_i и через Δ_U - предельную погрешность функции u, для малых Δ_{Xi} получим:

$$\Delta_U = \sum_{i=1}^{n} \left| \frac{\partial f}{\partial x_i} \right| \Delta_{Xi} . \qquad (5.1.11)$$

Пример 5.1.4. Найти предельную абсолютную и относительную погрешности объема шара $V=\pi d^3/6$, если диаметр d=3.7 см ± 0.05 см, а $\pi \approx 3.14$.

Решение. Рассматривая π и d как переменные величины, вычисляем частные производные:

$$\frac{\partial V}{\partial \pi} = \frac{1}{6} d^3 = 8.44,$$

$$\frac{\partial V}{\partial d} = \frac{1}{2} \pi d^2 = 21.5.$$

В силу формулы (5.1.11), предельная абсолютная погрешность объема

$$\Delta_V = \left| \frac{\partial V}{\partial \pi} \right| |\Delta \pi| + \left| \frac{\partial V}{\partial d} \right| |\Delta d| = 8.44 \cdot 0.0016 + 21.5 \cdot 0.05 = 0.013 + 1.075 = 1.088$$

см^3 ≈ 1.1 см^3.

Поэтому

$$V = \pi d^3/6 \approx 27.4 \text{ см}^3 \pm 1.1 \text{ см}^3.$$

Отсюда предельная относительная погрешность объема

$$\delta_V = 1.088 \text{ см}^3 / 27.4 \text{ см}^3 = 0.0397 \approx 4 \%.$$

5.2. ПРИБЛИЖЕННЫЕ МЕТОДЫ РЕШЕНИЯ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Одной из важнейших и наиболее распространённых задач прикладной математики является задача решения нелинейных уравнений, встречающихся в разных областях научных исследований.

В практической деятельности инженеру химику-технологу достаточно часто приходится сталкиваться с решением нелинейных уравнений. Например, при расчетах процессов однократного испарения, тепловых балансов в смесителях, при расчетах процессов отстаивания, диаметров нефтепроводов, расчете точки росы и многих других процессах. Нелинейные уравнения широко представлены в расчетах физико-химических свойств систем, например, при вычислении констант фазового равновесия.

Любое уравнение в общем случае можно представить в виде

$$f(x) = 0. \qquad (5.2.1)$$

Нелинейные уравнения можно разделить на два класса - алгебраические и трансцендентные.

Алгебраическими уравнениями называются уравнения, содержащие только алгебраические функции (целые, рациональные, иррациональные). Алгебраическое уравнение в общем виде можно представить многочленом n-ой степени с действительными коэффициентами:

$$f(x) = a_0x^n + a_1x^{n-1} + \ldots + a_n = 0. \qquad (5.2.2)$$

Например, $x^3 + x^2 + 2x = 0$.

Трансцендентными называются уравнения, содержащие другие функции (тригонометрические, показательные, логарифмические и т.д.), например:

$$2x - \sin x = 0.$$

Задача решения уравнения (5.2.1) заключается в нахождении таких значений **x**, которые обращают (5.2.1) в тождество, т.е. в 0.

$$f(\xi) = 0,$$

где ξ - корень уравнения.

Методы решения нелинейных уравнений делятся на прямые и итерационные. Прямые методы позволяют записать корни в виде

формулы. Однако встречающиеся на практике уравнения не всегда удаётся решить простыми методами. Для их решения используются итерационные методы, т.е. методы последовательных приближений.

Приближённое определение корней проводится в два этапа:

1. Отделение корней, т.е. установление достаточно малых отрезков, в каждом из которых содержится только один корень уравнения.

2. Уточнение приближённого значения корней до некоторой заданной степени точности.

5.2.1. ОТДЕЛЕНИЕ КОРНЕЙ

Приближенное значение корня может быть найдено различными способами.

Графический метод отделения корней.

Пусть требуется отделить корни уравнения (5.2.1). Для этого строим график функции f(x)=0. Абсциссы точек пересечения графика с осью OX будут приближёнными значениями корней уравнения (5.2.1).

Часто на практике уравнение (5.2.1) преобразовывают к более простому виду. Допустим:

$$f(x) = \varphi_1(x) - \varphi_2(x) = 0, \qquad (5.2.3)$$

$$\varphi_1(x) = \varphi_2(x).$$

Строим графики функций: $y_1=\varphi_1(x)$; $y_2=\varphi_2(x)$. Корнями уравнения (5.2.3) будут абсциссы пересечения этих графиков.

Пример 5.2.1. Отделить корни уравнения f(x)= x∗lgx-1 = 0. Преобразуем f(x) к виду:

$$\lg x = 1/x.$$

Построим графики функций $y_1 = \lg x$ и $y_2 = 1/x$ (рис.35).

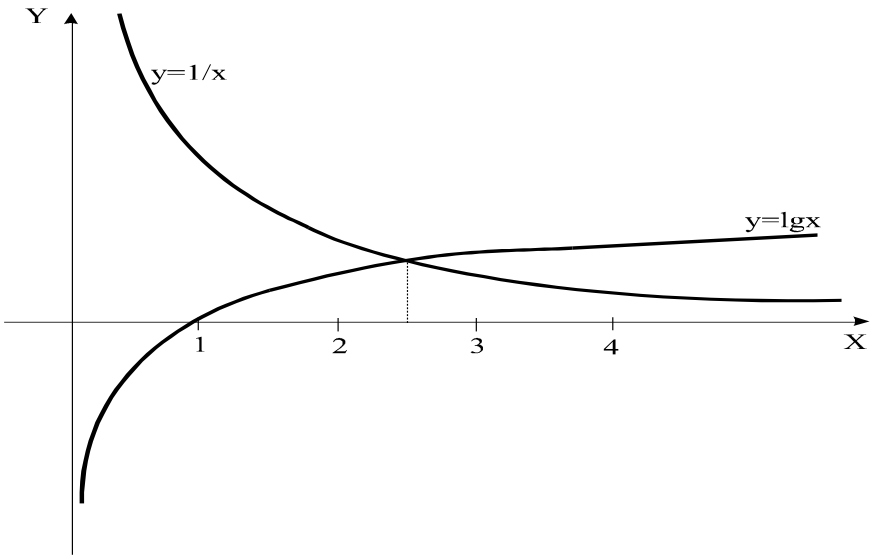


РИС.35. ГРАФИЧЕСКИЙ МЕТОД ОТДЕЛЕНИЯ КОРНЕЙ

Точка пересечения этих графиков даёт приближённое значение единственного корня $\xi \approx 2.5$.

Аналитический метод отделения корней.

Теорема. Если непрерывная функция f(x) принимает значения разных знаков на концах отрезка [a, b], т.е. $f(a)*f(b)<0$, то между точками a и b имеется хотя бы один действительный корень уравнения f(x)=0, т.е. существует такое число ξ, принадлежащее [a,b], что $f(\xi)=0$ (рис.36).

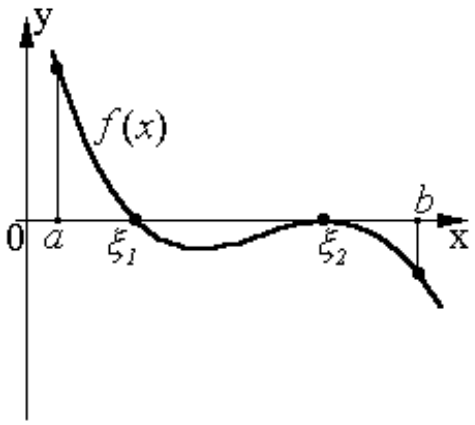


Рис.36

При этом, если на заданном отрезке [a,b] существует первая производная f'(x), сохраняющая знак внутри [a,b] (f'(x)>0 или f'(x)<0), то корень ξ будет единственным рис.(37).

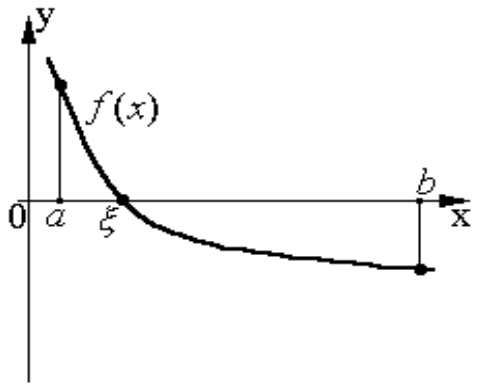


РИС.37

Процесс отделения корней начинается с установления знаков функции в граничных точках **a** и **b**. Затем определяются знаки в ряде промежуточных точек. После чего выделяются отрезки, на границе которых функция меняет знак на противоположный. Выделенные отрезки и содержат корень данного уравнения.

Пример 5.2.2. Отделить корни уравнения $f(x) = x^3 - 7x + 3 = 0$.

При заданных значениях x от - ∞ до + ∞ определяем знаки f(x). Результаты поиска приведены в табл.8.

Таблица 8

X	Знак f (x)
-∞	-
- 3	-
- 1	+
0	+
+1	-
+3	+
+ ∞	+

В результате поиска выделены три интервала, на которых функция f(x) имеет действительные корни: [-3,-1]; [0,1]; [1,3] .

5.2.2. УТОЧНЕНИЕ КОРНЕЙ

Итерационный процесс состоит в последовательном уточнении начального приближения $\mathbf{x_0}$. Каждый такой шаг называется *итерацией*.

Рассмотрим некоторые итерационные методы решения нелинейных уравнений.

5.2.2.1. Метод деления отрезка пополам (метод бисекций)

Пусть дано уравнение f(x)=0. Допустим, нам удалось найти такой отрезок [a,b], на котором расположено значение корня ξ, т.е. a<ξ<b. В качестве начального приближения корня x_0 (рис.39.) принимаем середину отрезка $x_0=(a+b)/2$. Далее исследуем значения функции: если $f(x_0)=0$, то x_0 является корнем уравнения, т.е. $\xi=x_0$. Если $f(x_0)\neq 0$, то выбираем одну из половин отрезка $[a,x_0]$ или $[x_0,b]$, на концах которой функция f(x) имеет противоположные знаки, т.е. содержит искомый корень, поэтому его принимаем в качестве нового отрезка $[x_0,b]$. Вторую половину отрезка на концах которого знак f(x) не меняется, отбрасываем: в данном случае $[a,x_0]$. Отрезок $[x_0,b]$ вновь делим пополам. Новое приближение: $x_1=(x_0+b)/2$. Вновь исследуем функцию f(x) на концах отрезка и отбрасываем отрезок $[x_1,b]$, т.к. $f(x_1)>0$ и $f(b)>0$.

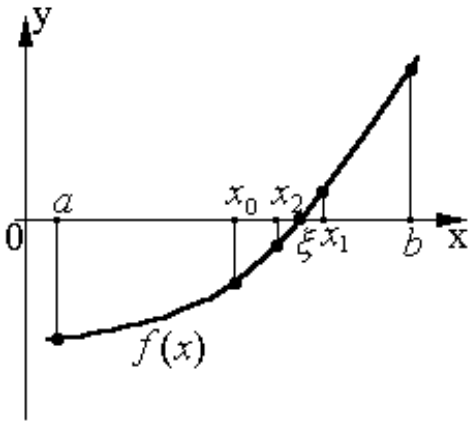


РИС.38. МЕТОД ДЕЛЕНИЯ ОТРЕЗКА ПОПОЛАМ

Отрезок $[x_0,x_1]$, на концах которого функция имеет противоположные знаки $f(x_1)>0$, $f(x_0)<0$, вновь делим пополам и получаем

новое приближение корня $x_2=(x_0+x_1)/2$ и т.д. Итерационный процесс продолжаем до тех пор, пока длина отрезка после n-ой итерации не станет меньше некоторого заданного малого числа (погрешности) ε, т.е.

$$|b-a|\leq \varepsilon . \qquad (5.2.4)$$

Тогда за искомое значение корня принимают полученное приближение x_n: $\xi=_{xт.}$ Говорят, что решение данного уравнения найдено с точностью ε.

Пример 5.2.3. Найти корни обобщенного кубического уравнения, описывающего уравнение состояния газа относительно коэффициента сжимаемости, записанного в следующем виде:

$$x^3 - x^2 - x(m^2 + m - d\cdot m) - d\cdot m^2 = 0,$$

где m=0,4; d=3,4 с точностью 0,01.

На этапе отделения корней был выделен интервал, на котором функция f(x) меняет знак: [0,1]. Определим корень уравнения, используя метод деления отрезка пополам.

Последовательность решения. Определим знак функции на концах отрезка [0,1]:

$$f(0)=-0.544; \quad f(1)= 0.256.$$

Делим отрезок пополам:

$$(1-0)/2= 0.5.$$

Значение функции в этой точке

$$f(0.5)=--0.269$$

имеет положительное значение. Отбрасываем половину отрезка, на концах которого функция имеет отрицательные знаки, а именно - отрезок [0,0.5]. Полученный отрезок [0.5,1] вновь делим пополам :

$$(1+0.5)/2= 0.75 .$$

f(0.75) <0, следовательно, отбрасываем отрезок [0.5,0.75], а отрезок [0.75,1] делим пополам

$$(1+0.75)/2= 0.875.$$

f(0.875)<0, следовательно, рассматриваем отрезок [0.75,0.875]:

$$(0.875+0.750)/2= 0.813.$$

f(0.813)<0, выбираем отрезок [0.813,0.875]:

(0.813+0.875)/2=-0.844.

f(0.844)>0, следовательно, вновь полученный отрезок : [0.813,0.844].
(0.813+0.844)/2=-0.828.

Проверим условие окончания вычислений по формуле (5.2.4):

$|0.844 - 0.813| = 0.012$,

0.012<0.1.

Таким образом, за искомое значение корня принимаем значение

x=0.828 .

На рис.40. представлена блок-схема нахождения корня уравнения (5.2.1) методом деления отрезка пополам.

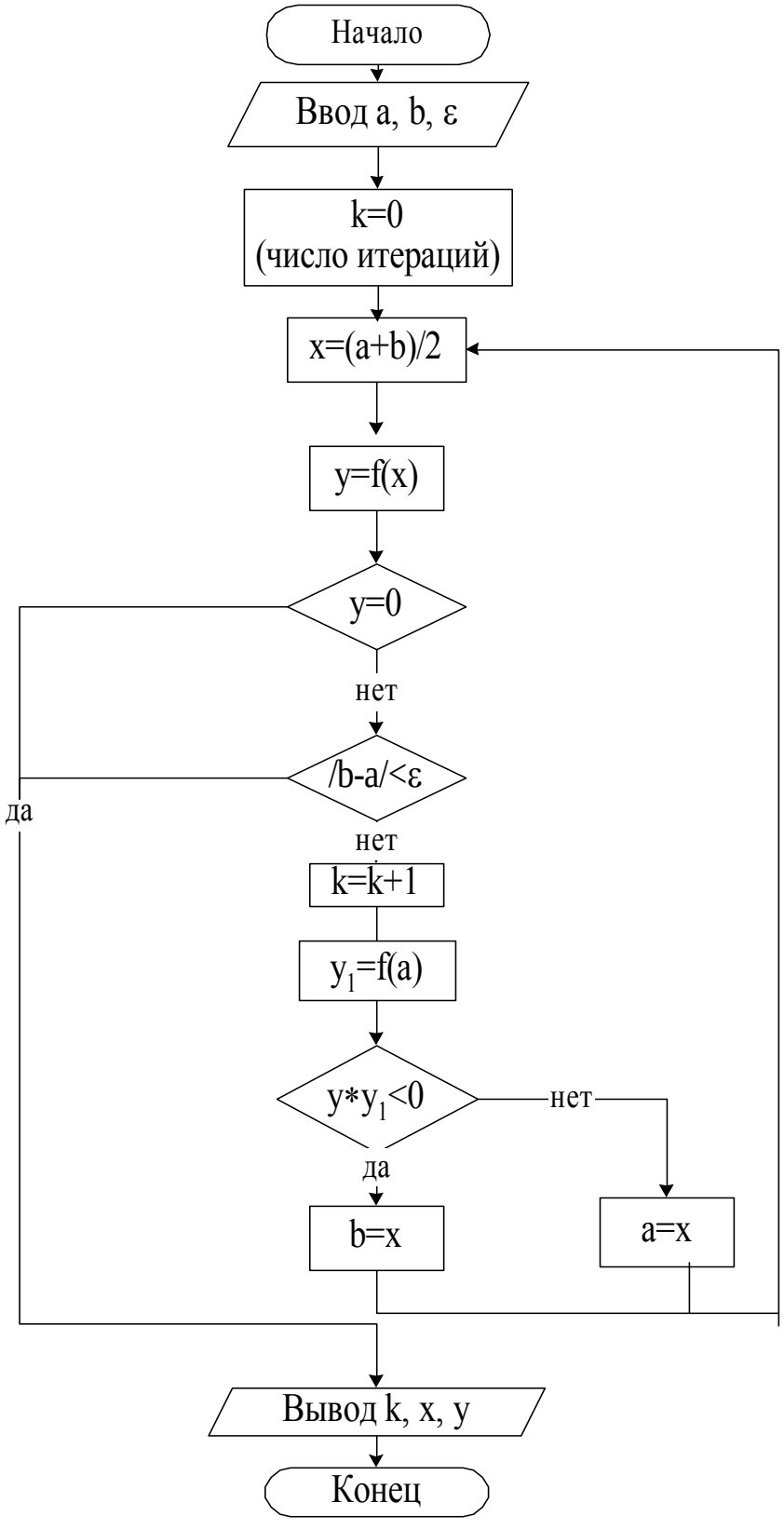


Рис.40. Блок-схема алгоритма метода деления отрезка пополам

5.2.2.2. Метод Ньютона (метод касательных)

Если известно начальное приближение к корню уравнения f(x)=0, то эффективным методом уточнения корней является метод Ньютона (метод касательных).

Сформулируем достаточные условия сходимости метода.

Пусть функция f(x) имеет первую и вторую производную на отрезке [a,b], причем выполнено условие знакопеременности функции f(a)∗f(b)<0, а производные f□'(x), f''(x) сохраняют знак на отрезке [a,b]. Тогда, исходя из начального приближения $x_0 \in [a,b]$, удовлетворяющего неравенству f(x)∗f''(x)>0, можно построить итерационную последовательность

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n=0,1,2,..., \qquad (5.2.5)$$

сходящуюся к единственному на [a,b] решению ξ уравнения f(x)=0.

В данном методе процесс итераций состоит в том, что в качестве приближений к корню принимаются значения x_0, x_1, x_2... точек пересечения касательной к кривой y=f(x) с осью абсцисс. То есть, геометрически метод Ньютона эквивалентен замене небольшой дуги кривой y=f(x) касательной. При этом не обязательно задавать отрезок [a,b], содержащий корень уравнения, а достаточно лишь найти некоторое начальное приближение корня $x=x_0$ (рис.40)

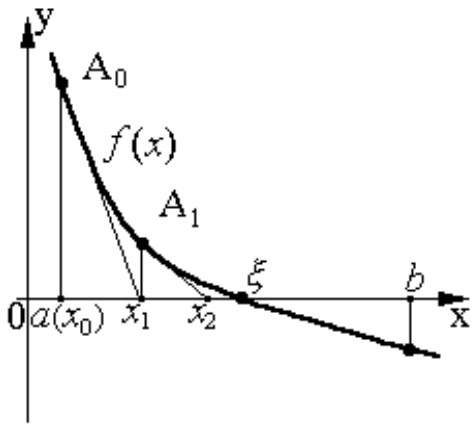


РИС.41

В качестве начального приближения выберем $x_0=a$, для которого выполняется условие $f(x_0)\cdot f''(x_0)>0$. Проведем касательную в точке $A_0[x_0,f(x_0)]$. Первым приближением корня будет точка пересечения этой касательной с осью абсцисс x_1. Через точку $A_1[x_1,f(x_1)]$ снова проводим

касательную, точка пересечения которой с осью OX даст нам второе приближение корня x_2 и т.д.

На рис.42 приведены возможные варианты выбора правого или левого конца отрезка в качестве начального приближения.

Условие выбора: $f(x)\cdot f''(x)>0$.

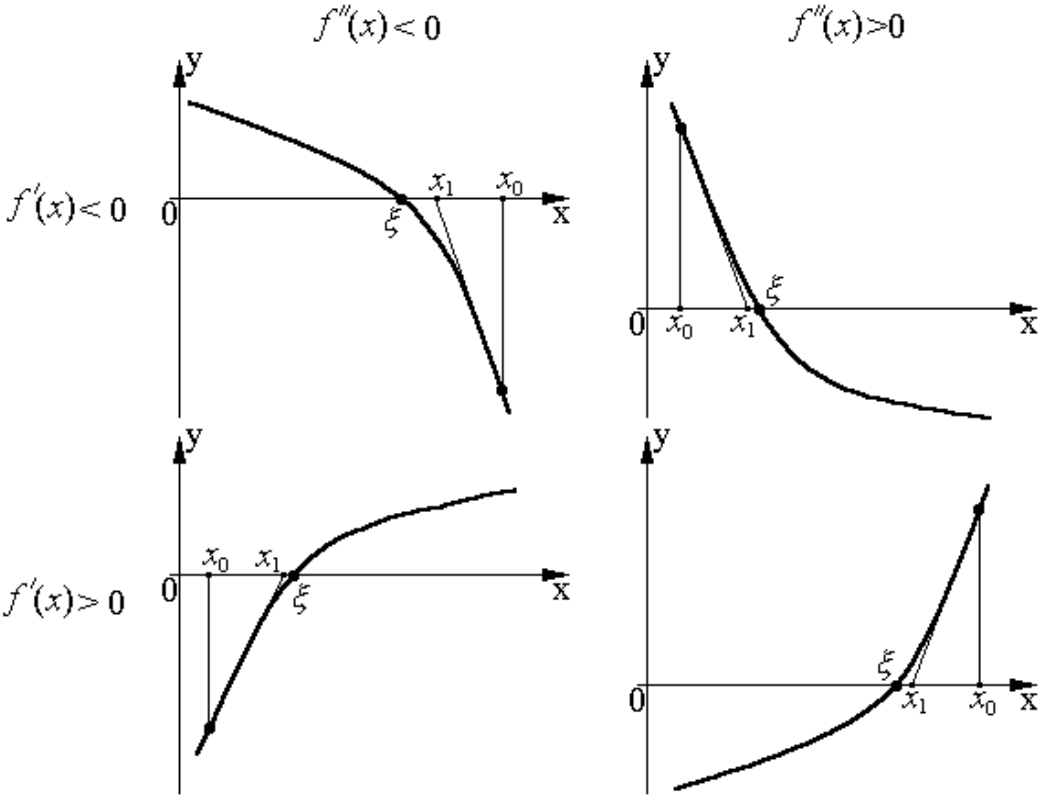


РИС.42

Вывод формулы Ньютона.

Уравнение касательной, проведенной к кривой y=f(x) в точке $A_0[x_0,f(x_0)]$, имеет вид:

$$y - f(x_0) = f'(x_0)(x-x_0). \qquad (5.2.6)$$

Отсюда найдем следующее приближение корня. Примем $x=x_1$ (y=0), тогда

$$-f(x_0) = f'(x_0)(x_1-x_0), \qquad (5.2.7)$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} .$$

Аналогично могут быть найдены и следующие приближения, как точки пересечения с осью OX касательных, проведенных в точках A_1, A_2 и т.д. Формула Ньютона для n+1-го приближения будет иметь вид:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \qquad (5.2.8)$$

Для окончания итерационного процесса может быть использовано условие:

$$|x_{n+1} - x_n| \le \varepsilon. \qquad (5.2.9)$$

Пример 5.2.4. Для уравнения, приведенного в примере 5.2.3. найти действительные корни, используя метод Ньютона :

$x^3 - x^2 - x(m^2 + m - d \cdot m) = 0,$

где m=0,4; d=3,4.

На этапе отделения корней был выделен интервал, на котором функция f(x) меняет знак: [0,1], следовательно, уравнение на данном интервале имеет действительный корень. Вычислим по методу Ньютона значение корня на отрезке [0,1]. Выберем начальное приближение так, чтобы выполнялось условие $f(x_0)*f''(x_0)>0$.

Запишем первую и вторую производную функции f(x):

$$f'(x)=3x^2-2x-(m^2+m-d\cdot m);$$

$$f''(x)=6x-2.$$

Вычислим значения $f(x_0)$ и $f''(x_0)$:

x=1 f(1)= 0,256; f″(1)=4;

$$f(1)*f''(1)>0.$$

За начальное приближение принимаем $x_0=1$.

Найдем корень уравнения по формуле (5.2.5) с точностью $\varepsilon=0.001$.

1) $x_1=x_0 - f(x_0)/f'(x_0)$;

f(1)=0.256; f′(1)=-1.80;

$x_1=1 - 0.256/1.8=0.858$;

2) $x_2=x_1 - f(x_1)/f'(x_1)$;

f(0.858)=0.0376; f ' (0.858)=1.29;

$x_2=0.858 - 0.0376/1.129=0.829$;

3) $x_3=x_2-f(x_2)/f'(x_2)$;

f(0.829)=0.0013; f′(0.829)=1.203;

$x_3=0.829 -0.0013/1.203=0.828$;

4) $x_4=x_3-f(x_3)/f'(x_3)$;

f(0.828)=1.4e-6; f ' (0.828)=1,2;

$x_4=0.828 - 1.4e-6/1.2=0.828$

Таким образом, корнем данного уравнения с точностью ε=0,001 (после третьей итерации) будет значение x=0,828.

Проверим по формуле (5.2.9) условие окончания вычислений

|0.829-0.828|=0.001<=0.001.

Блок схема алгоритма метода Ньютона приведена рис.43.

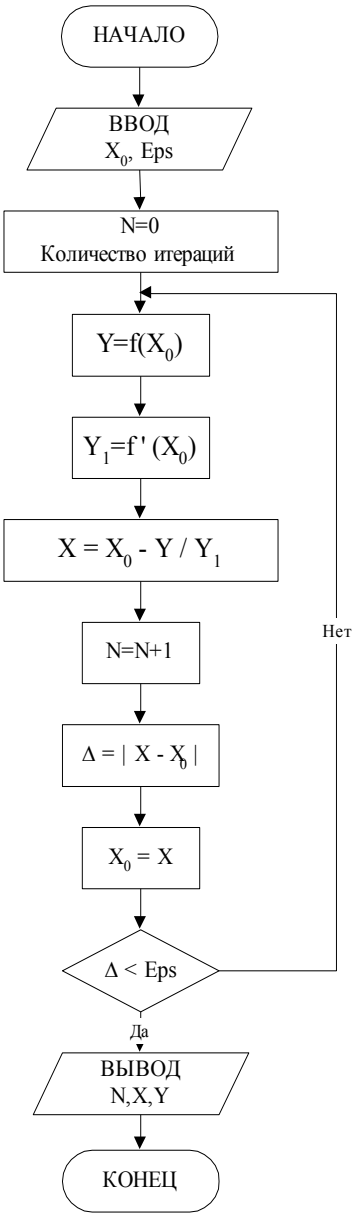


Рис.43. Блок-схема метода Ньютона

X_0 - начальное приближение; X - последовательное приближение

5.2.2.3. Метод простых итераций

Одним из наиболее важных численных методов решения нелинейных уравнений является метод итераций (метод последовательных приближений). Сущность метода заключается в следующем.

Заменим исходное нелинейное уравнение (5.2.1) эквивалентным ему уравнением вида:

$$x = \varphi(x) . \qquad (5.2.10)$$

Пусть известно начальное приближение корня $x=x_0$. Подставляя это значение в правую часть уравнения (5.2.10), получаем новое приближение:

$$x_1 = \varphi(x_0) , \qquad (5.2.11)$$

затем аналогичным образом получим

$$x_2 = \varphi(x_1) . \qquad (5.2.12)$$

Далее, подставляя каждый раз новое значение корня в (5.2.11), получаем последовательность значений

$$x_{n+1} = \varphi(x_n) , \qquad n=1,2, \ldots . \qquad (5.2.13)$$

Итерационный процесс продолжается до тех пор, пока не станут близки результаты двух последовательных итераций.

$$|x_{n+1}-x_n| \le \varepsilon . \qquad (5.2.14)$$

Достаточным условием сходимости метода простых итераций является условие

$$|\varphi'(x)| < 1, \qquad (5.2.15)$$

выполненное для любого x, принадлежащего некоторому отрезку [a,b], содержащему корень уравнения.

Рассмотрим геометрическую интерпретацию метода. Построим графики функций $y=x$ и $y=\varphi(x)$ (рис.44). Корнем ξ уравнения $x=\varphi(x)$ является абсцисса точки пересечения кривой $y=\varphi(x)$ с прямой $y=x$.

Из графиков видно, что при $\varphi'(x)>0$ (рис.44,а,б) и при $\varphi'(x)<0$ (рис.44,в,г) возможны как сходящиеся, так и расходящиеся итерационные процессы. Скорость сходимости зависит от абсолютной величины производной $\varphi'(x)$. Чем меньше $|\varphi'(x)|$ вблизи корня, тем быстрее

сходится процесс. Таким образом, при переходе от уравнения (5.2.1) к уравнению (5.2.10) необходимо, чтобы выполнялось условие (5.2.15).

Итерационные процессы могут быть односторонними, если φ'(x)>0 и двусторонними, если φ'(x)<0.

Блок-схема метода простых итераций приведена на рис.45.

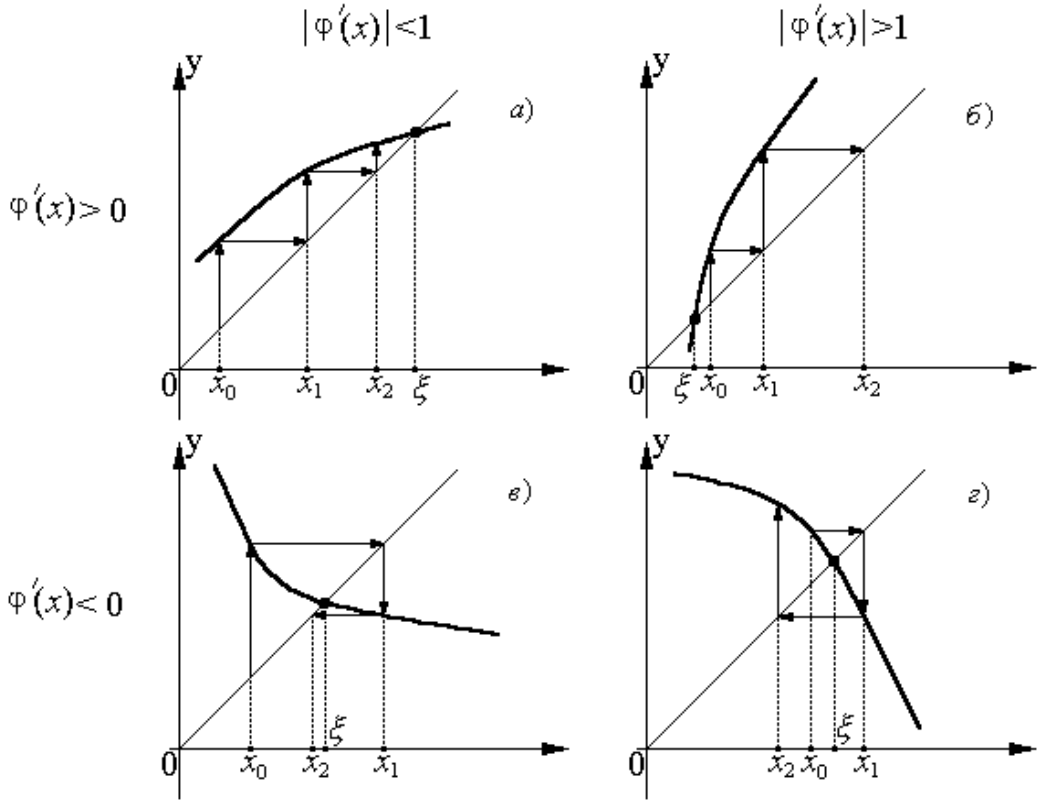


Рис. 44. Метод простых итераций

Пример 5.2.5. Требуется решить нелинейное уравнение $e^{-x}-x^2=0$. Допустим, что в результате аналитического отделения корней было найдено, что корень уравнения находится на интервале [0,1].

x	-2	-1	0	1	2
f(*x*)	3.389	1.718	1	-0.632	-3.865

Приведем исходное уравнение $f(x)= e^{-x}-x^2=0$ к эквивалентному виду $x=\varphi(x)$. Возможны два варианта.

1) $e^{-x}-x^2=0$; $e^{-x}=x^2$; $x=-2\cdot\ln(x)$; $\varphi(x)=-2\cdot\ln(x)$,

2) $e^{-x}-x^2=0$; $e^{-x}=x^2$; $x=\sqrt{e^{-x}}$; $\varphi(x)=\sqrt{e^{-x}}$.

Проверим для каждого из них условие сходимости $|\varphi'(x)|<1$:

1) $\varphi'(x)=(-2\cdot\ln(x))'=-2/x$ и очевидно, что на интервале [0,1] $|\varphi'(x)|>1$,

x	0	0.25	0.5	0.75	1
$\lvert -2/x\rvert$	$+\infty$	8	4	2.667	2

следовательно, метод итераций при таком эквивалентном уравнении

$$x=-2\cdot\ln(x)$$

сходиться не будет.

2) $\varphi'(x)=\dfrac{e^{-x}}{2\cdot\sqrt{e^{-x}}}=\dfrac{e^{-x/2}}{2}$. На интервале [0,1] $|\varphi'(x)|<1$

x	0	0.25	0.5	0.75	1
$\dfrac{e^{-x/2}}{2}$	0.5	0.441	0.384	0.345	0.303

и ,следовательно, метод итераций при таком эквивалентном уравнении

$x=\sqrt{e^{-x}}$

сходится, что хорошо видно из приведенных ниже результатов расчета. В качестве начального приближения может быть выбрано любое число, принадлежащее исходному интервалу, например, $x_0=0$.

n	x_n	$\varphi(x)=\sqrt{e^{-x}}$
0	0	1
1	1	0.607
2	0.607	0.738
3	0.738	0.691
4	0.691	0.708
5	0.708	0.702
6	0.702	0.704
7	0.704	0.703

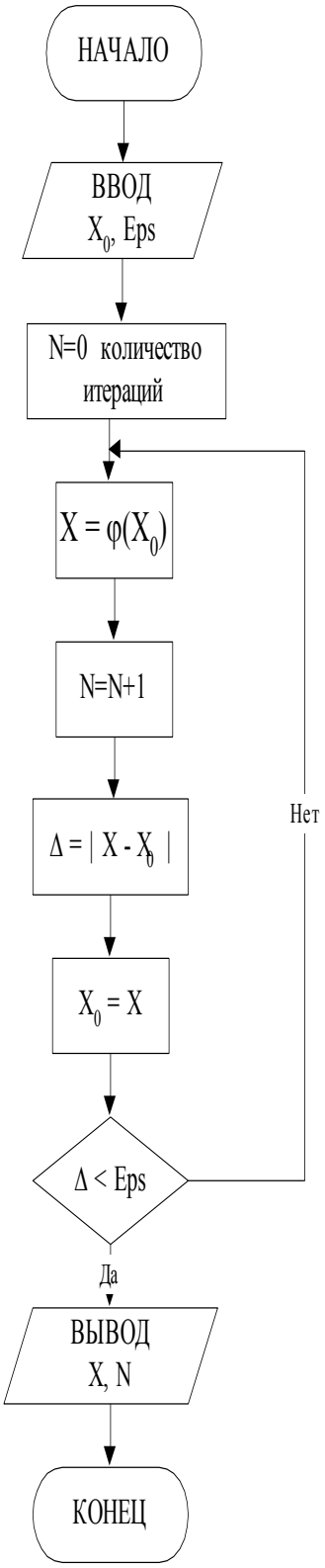


Рис.45. Блок-схема метода простых итераций

5.3. МЕТОДЫ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ

5.3.1. Постановка задачи

К решению систем линейных уравнений сводятся многочисленные практические задачи.

Запишем систему **n** линейных алгебраических уравнений с **n** неизвестными:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n. \end{cases} \qquad (5.3.1)$$

Совокупность коэффициентов этой системы запишем в виде таблицы, или матрицы:

$$\mathbf{A} = \begin{bmatrix} a_{11}a_{12}\ldots a_{1n} \\ a_{21}a_{22}\ldots a_{2n} \\ \ldots\ldots\ldots\ldots \\ a_{n1}a_{n2}\ldots a_{nn} \end{bmatrix} . \qquad (5.3.2)$$

Используя понятие матрицы **A**, систему уравнений (5.3.1) можно записать в матричном виде:

$$\mathbf{AX}=\mathbf{B}, \qquad (5.3.3)$$

где **X** и **B** - вектор-столбец неизвестных и вектор-столбец правых частей соответственно:

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{bmatrix} .$$

Определителем, или детерминантом, матрицы **A** (det A) называется число **D**, равное

$$\mathbf{D} = \begin{vmatrix} a_{11}a_{12}\ldots a_{1n} \\ a_{21}a_{22}\ldots a_{2n} \\ \ldots\ldots\ldots\ldots \\ a_{n1}a_{n2}\ldots a_{nn} \end{vmatrix} = \sum (-1)^k a_{1\alpha}a_{2\beta}\ldots a_{n\varpi}. \qquad (5.3.4)$$

Здесь индексы $\alpha, \beta, \ldots, \varpi$ пробегают все возможные **n!** перестановок номеров 1,2,...,*n*; *k* - число инверсий в данной перестановке.

Необходимым и достаточным условием существования единственного решения системы линейных уравнений является условие **D** ≠ 0. В случае равенства нулю определителя системы матрица называется *вырожденной* ; при этом система линейных уравнений (5.3.1) либо не имеет решения, либо имеет их бесчисленное множество.

Методы решения систем линейных уравнений делятся на две группы - прямые и итерационные. *Прямые методы* используют конечные соотношения (формулы) для вычисления неизвестных. Они дают решение после выполнения заранее известного числа операций. Эти методы достаточно просты и пригодны для широкого класса линейных систем.

К прямым методам решения систем линейных уравнений относятся метод Гаусса, метод прогонки, схема Жордана, метод квадратного корня, метод оптимального исключения, клеточные методы. Среди прямых методов наиболее распространён метод Гаусса.

Итерационные методы - это методы последовательных приближений. В них необходимо задать некоторое приближённое решение - *начальное приближение*. После этого с помощью некоторого алгоритма проводится цикл вычислений, называемый *итерацией*. В результате итерации находят новое приближение. Итерации проводятся до получения решения с требуемой точностью. Итерационные методы могут использоваться для уточнения решений, полученных с помощью прямых методов. Одним из самых распространённых итерационных методов, отличающихся простотой и лёгкостью программирования, является метод Гаусса-Зейделя.

5.3.2. Метод Гаусса (метод исключений)

Методом Гаусса называют точный метод решения невырожденной системы линейных уравнений, состоящий в том, что последовательным исключением неизвестных систему

$$\sum_{j=1}^{n} a_{ij} \cdot x_j = b_i, \qquad i=1,2,..,n \qquad (5.3.5)$$

или

$$\begin{cases} a_{11}x_1 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + \ldots + a_{nn}x_n = b_n \end{cases} \qquad (5.3.6)$$

приводят к эквивалентной системе с треугольной матрицей:

$$\begin{aligned} x_1 + c_{12}x_2 + \ldots + c_{1n}x_n &= d_1 \\ x_2 + \ldots + c_{2n}x_n &= d_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots&, \\ x_n &= d_n \end{aligned} \qquad (5.3.7)$$

решение которой находят по рекуррентным формулам :

$$x_i = d_n,$$

$$x_i = d_i - \sum_{k=j+1}^{n} c_{ik} x_k, \quad i = n\text{-}1, n\text{-}2, \ldots, 1. \qquad (5.3.8)$$

Существует много вариантов этого метода. Рассмотрим схему с выбором главного элемента.

Предположим, что $a_{11} \neq 0$ и разделим обе части первого уравнения системы на a_{11}, в результате получим уравнение

$$x_1 + b_{12}x_2 + \ldots + b_{1n}x_n = b_1. \qquad (5.3.9)$$

С помощью полученного уравнения исключаем во всех уравнениях системы, начиная со второго, слагаемые, содержащие x_1. Для этого умножаем последовательно обе части уравнения на a_{21}, a_{31}, **...,** a_n и вычитаем из соответствующих уравнений. В результате получаем систему, порядок которой на единицу меньше порядка исходной. Аналогично преобразуем полученную систему. В результате **n**-кратного повторения этого преобразования получим систему с треугольной матрицей.

Основным условием применимости данной схемы является неравенство нулю элементов главной диагонали матрицы коэффициентов – $a_{ii} \neq 0$, i=1,...,n. В противном случае необходимо сделать перестановку уравнений системы и добиться выполнения этого условия.

Пример 5.3.1.

$$\begin{aligned} 3.1x_1 - 0.6x_2 - 2.3x_3 &= -1.5, \\ 0.8x_1 + 7.4x_2 - 0.5x_3 &= 6.4, \\ 1.4x_1 - 2.9x_2 + 9.7x_3 &= 4. \end{aligned} \qquad (5.3.10)$$

Элементы главной диагонали матрицы коэффициентов не равны нулю. Исключим x_1 из 2-го и 3-го уравнений системы:

$$x_1 - \frac{0.6}{3.1}x_2 - \frac{2.3}{3.1}x_3 = -\frac{1.5}{3.1},$$

$$\left(7.4 + \frac{0.6}{3.1}\cdot 0.8\right)x_2 + \left(-0.5 + \frac{2.3}{3.1}\cdot 0.8\right)x_3 = 6.4 + \frac{1.5}{3.1}\cdot 0.8, \qquad (5.3.11)$$

$$\left(-2.9 + \frac{0.6}{3.1}\cdot 1.4\right)x_2 + \left(9.7 + \frac{2.3}{3.1}\cdot 1.4\right)x_3 = 4 + \frac{1.5}{3.1}\cdot 1.4.$$

Перейдём к десятичным дробям с точностью до 3-го знака после запятой:

$$\begin{aligned} x_1 - 0.194\,x_2 - 0.742\,x_3 &= -0.484, \\ 7.555\,x_2 + 0.094\,x_3 &= 6.787, \\ -2.629\,x_2 + 10.739x_3 &= 4.677. \end{aligned} \qquad (5.3.12)$$

Исключим x_2 из 3-го уравнения:

$$\begin{aligned} x_1 - 0.194\,x_2 - 0.742\,x_3 &= -0.484, \\ x_2 + \frac{0.094}{7.555}x_3 &= \frac{6.787}{7.555}, \\ \left(10.739 + \frac{0.094}{7.555}\cdot 2.629\right)x_3 &= 4.677 + \frac{6.787}{7.555}\cdot 2.629 \end{aligned} \qquad (5.3.13)$$

и получим эквивалентную систему с треугольной матрицей:

$$x_1 - 0.194\,x_2 - 0.742\,x_3 = -0.484,$$

$$x_2 - 0.012 - x_3 = 0.898, \qquad (5.3.14)$$
$$x_3 = 0.653.$$

Теперь очевидно, что надо делать для решения системы. Необходимо определить x_3 из (5.3.14), подставить этот результат во второе уравнение системы и определить x_2:

$$x_2 = 0.012 \cdot 0.653 + 0.898 = 0.906, \qquad (5.3.15)$$

подставить x_2 и x_3 в первое уравнение системы (5.3.14) и определить x_1 :

$$x_1 = 0.194 \cdot 0.906 + 0.742 \cdot 0.653 - 0.484 = 0.176. \qquad (5.3.16)$$

Этот процесс обычно называют обратной подстановкой.

Блок-схема алгоритма решения системы линейных алгебраических уравнений методом Гаусса представлена на рис.46, программа расчёта на языке Паскаль приведена в Приложении. Рассмотрен случай, когда $a_{ii} \neq 0$, $i=1,...,n$, и перестановка уравнений системы не требуется.

Для удобства реализации алгоритма вектор-столбец правых частей уравнений включен *(n+1)*-м столбцом в матрицу коэффициентов **A** системы **n** линейных уравнений.

5.3.3.Интерполяционный метод гаусса- зейделя

Этот метод исключительно удобен для использования на ЭВМ.

Рассмотрим систему из трёх уравнений с тремя неизвестными.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1, \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2, \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3. \end{cases} \qquad (5.3.17)$$

Предположим, что $a_{11} \neq 0$; $a_{22} \neq 0$; $a_{33} \neq 0$ и перепишем систему в следующем виде:

$$x_1 = \frac{1}{a_{11}}\left(b_1 - a_{12}x_2 - a_{13}x_3\right), \qquad (5.3.18)$$
$$x_2 = \frac{1}{a_{22}}\left(b_2 - a_{21}x_1 - a_{23}x_3\right),$$
$$x_3 = \frac{1}{a_{33}}\left(b_3 - a_{31}x_1 - a_{32}x_2\right). \qquad (5.3.19)$$

(5.3.20)

Теперь возьмём некоторое первое приближение к решению этой системы, обозначив его через $x_1^{(0)}$, $x_2^{(0)}$, $x_3^{(0)}$. Подставим это решение в (5.3.18) и вычислим новое значение $x_1^{(1)}$:

$$x_1^{(1)} = \frac{1}{a_{11}}\left(b_1 - a_{12}x_2^{(0)} - a_{13}x_3^{(0)} \right). \qquad (5.3.21)$$

Используя только что вычисленное значение $x_1^{(1)}$ и начальное значение $x_3^{(0)}$, вычислим из уравнения (5.3.19) новое значение x_2:

$$x_2^{(1)} = \frac{1}{a_{22}}\left(b_2 - a_{21}x_1^{(1)} - a_{23}x_3^{(0)} \right). \qquad (5.3.22)$$

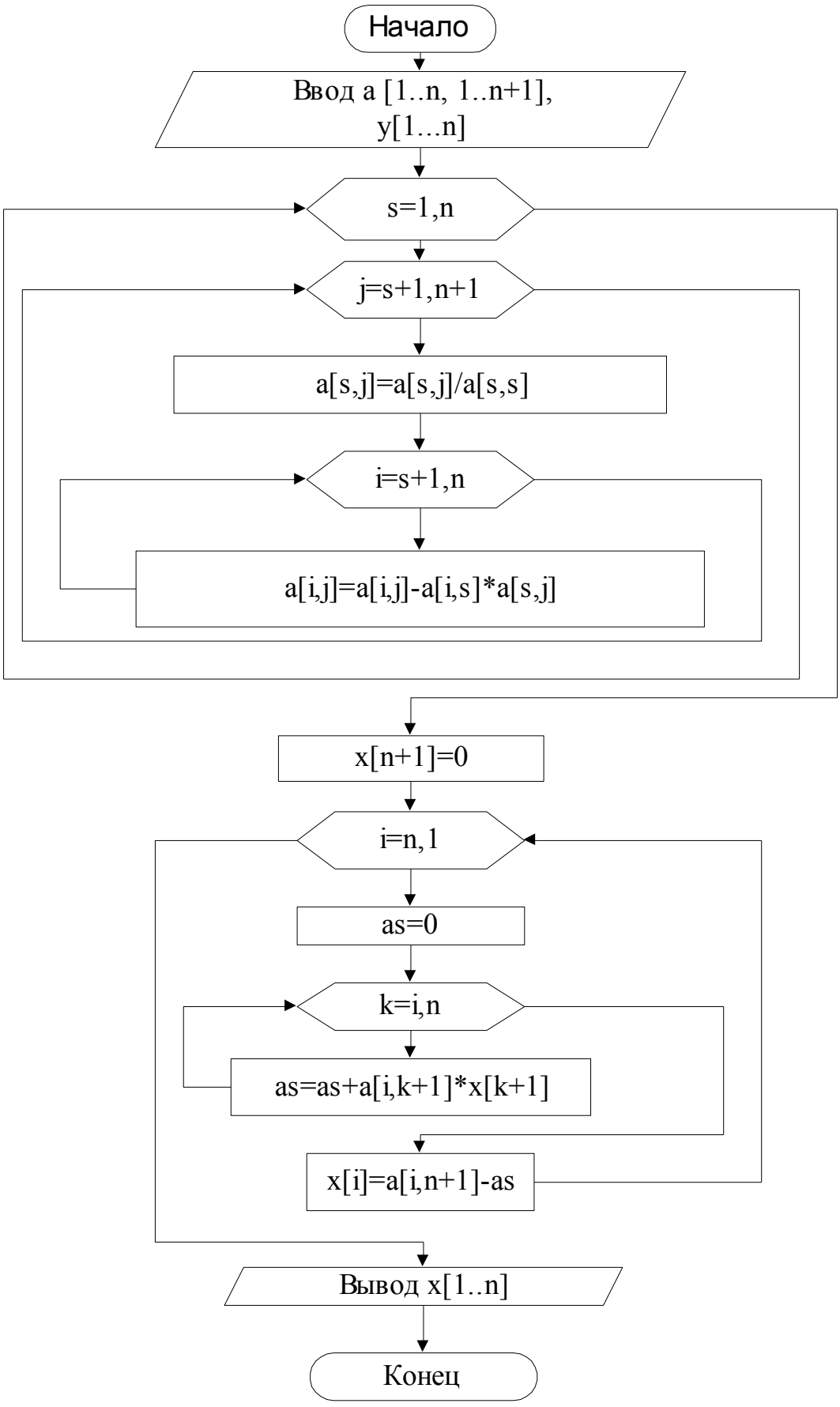


Рис. 46. Блок - схема метода Гаусса

Наконец, используя только что вычисленные значение $x_1^{(1)}$ и $x_2^{(1)}$, найдём из (5.3.20) новое значение x_3:

$$x_3^{(1)} = \frac{1}{a_{33}}\left(b_3 - a_{31}x_1^{(1)} - a_{32}x_2^{(1)} \right). \qquad (5.3.23)$$

Этим заканчивается первая итерация. Теперь можно заменить исходные значения $x_1^{(0)}$, $x_2^{(0)}$, $x_3^{(0)}$ на $x_1^{(1)}$, $x_2^{(1)}$, $x_3^{(1)}$ и вычислить следующее приближение.

В общем случае k-е приближение определяется формулами:

$$x_1^{(k)} = \frac{1}{a_{11}}\left(b_1 - a_{12}x_2^{(k-1)} - a_{13}x_3^{(k-1)} \right),$$

$$x_2^{(k)} = \frac{1}{a_{22}}\left(b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k-1)} \right), \qquad (5.3.24)$$

$$x_3^{(k)} = \frac{1}{a_{33}}\left(b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)} \right).$$

Пример 5.3.2.

$$\begin{aligned} 4x_1 - x_2 - x_3 &= 4, \\ x_1 + 6x_2 + 2x_3 &= 9, \\ -x_1 - 2x_2 + 5x_3 &= 2. \end{aligned} \qquad (5.3.25)$$

Нетрудно убедиться, что точное решение этой системы равно x_1=1, x_2=1, x_3=1. Положим $x_1^{(0)} = 0$, $x_2^{(0)} = 0$, $x_3^{(0)} = 0$, как это обычно делается для начального приближения. Тогда, согласно формулам:

$$\begin{aligned} x_1 &= \frac{1}{4}(4 + x_2 - x_3), \\ x_2 &= \frac{1}{6}(9 - x_1 - 2x_3), \\ x_3 &= \frac{1}{5}(2 + x_1 + 2x_2). \end{aligned} \qquad (5.3.26)$$

получаем следующее приближение:

$$\begin{aligned} x_1^{(1)} &= 1, \\ x_2^{(1)} &= \frac{4}{3}, \\ x_3^{(1)} &= \frac{17}{15}. \end{aligned} \qquad (5.3.27)$$

Последовательные приближения, вычисленные каждый раз с точностью до четырёх значащих цифр, приведены в табл. 9.

Таблица 9

Итерация	x_1	x_2	x_3
0	0	0	0
1	$0.1000\cdot10^1$	$0.1333\cdot10^1$	$0.1133\cdot10^1$
2	$0.1050\cdot10^1$	$0.9473\cdot10^0$	$0.9889\cdot10^0$
3	$0.9896\cdot10^0$	$0.1005\cdot10^1$	$0.9999\cdot10^0$
4	$0.1001\cdot10^1$	$0.9999\cdot10^0$	$0.1000\cdot10^1$
5	$0.1000\cdot10^1$	$0.1000\cdot10^1$	$0.1000\cdot10^1$

Для системы из **n** уравнений с **n** неизвестными (диагональные элементы отличны от нуля) *k*-ое приближение к решению будет задаваться функцией:

$$x_i^{(k)} = \frac{1}{a_{ii}}\left(b_i - a_{i1}x_1^{(k)} - \ldots - a_{i,i-1}x_{i-1}^{(k)} - a_{i,i+1}x_{i+1}^{(k-1)} - \ldots - a_{in}x_n^{(k-1)}\right), \quad (5.3.28)$$

i=1,2,...,n.

Интерполяционный процесс продолжается до тех пор, пока все $x_i^{(k)}$ не станут достаточно близки к $x_i^{(k-1)}$. Критерий близости можно задавать в следующем виде:

$$M^{(k)} = \max\left|x_i^{(k)} - x_i^{(k-1)}\right| < \varepsilon, \quad (5.3.29)$$

где определяется максимальное значение разности для всех i, а ε - некоторое положительное число.

Достаточным условием сходимости метода Гаусса- Зейделя является то, что диагональные члены должны преобладать в уравнении, т.е. они должны быть по абсолютной величине не меньше, а по крайней мере в одном случае, больше недиагональных элементов:

$$|a_{ii}| \geq |a_{i1}| + \ldots + |a_{i,i-1}| + |a_{i,i+1}| + \ldots + |a_{in}| \quad (5.3.30)$$

для всех или по крайней мере для одного i.

Блок-схема алгоритма метода Гаусса-Зейделя представлена на рис.47, программа – в Приложении.

Таким образом, рассмотрено два метода решения систем линейных уравнений и естественно, возникает вопрос, какой метод предпочтительнее?

Метод исключения имеет то преимущество, что он конечен, и теоретически с его помощью можно решить любую невырожденную систему уравнений ($\det \neq 0$). Итерационный метод сходится только для специальных систем уравнений. Для некоторых систем метод исключения является единственно возможным.

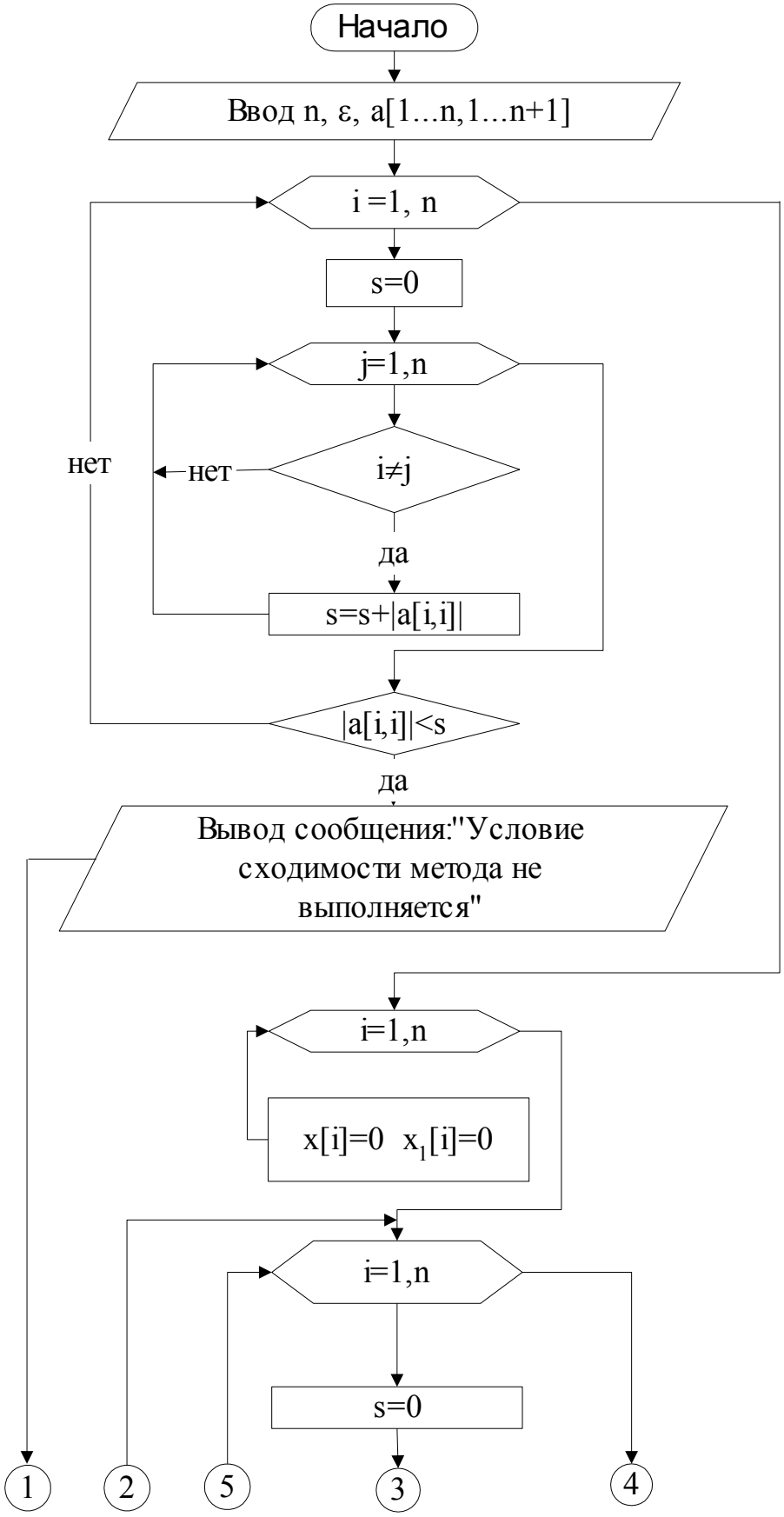
Однако, если итерационные методы сходятся, они обычно предпочтительнее:

1. Время вычислений пропорционально $\mathbf{n}^2$ на итерацию, в то время как для метода исключения время вычислений пропорционально $\mathbf{n}^3$; если для решения системы требуется менее **n** итераций, то общие затраты машинного времени будут меньше.

2. Как правило, ошибки округления при итерационном методе меньше; иногда это соображение может оказаться достаточно важным и оправдывает дополнительные затраты машинного времени.

Многие системы уравнений, возникающие на практике, имеют среди коэффициентов больший процент нулей, например, уравнения, получаемые при решении уравнений в частных производных. В этих случаях итерационные методы в высшей степени предпочтительны, так как при решении на ЭВМ можно проверять коэффициенты и не производить умножение, если они равны нулю. В треугольной матрице практически нет нулевых коэффициентов, а некоторые системы уравнений настолько велики, что их практически нельзя точно решить методом исключений.

В качестве практического примера использования методов решения систем линейных уравнений можно привести поиск коэффициентов параболической аппроксимации по набору экспериментальных точек (см. пример 2, раздел 5.5, формулы 5.5.31-5.5.36,). Решение этого примера надежнее проводить с помощью прямого метода Гаусса, т.к. условие сходимости итерационного метода Гаусса-Зейделя может не выполниться. Блок-схема алгоритма расчёта представлена на рис. 48. Программа расчета на языке Паскаль - в Приложении. Матрица коэффициентов системы линейных уравнений обозначена двумерным массивом **aa**, искомые коэффициенты параболической аппроксимации - массивом **a.**


Начало
Ввод n, ε, a[1…n,1…n+1]
i =1, n
s=0
j=1,n
нет
нет
i≠j
да
s=s+|a[i,i]|
|a[i,i]|<s
да
Вывод сообщения:"Условие сходимости метода не выполняется"
i=1,n
x[i]=0 x1[i]=0
i=1,n
s=0
1
2
5
3
4

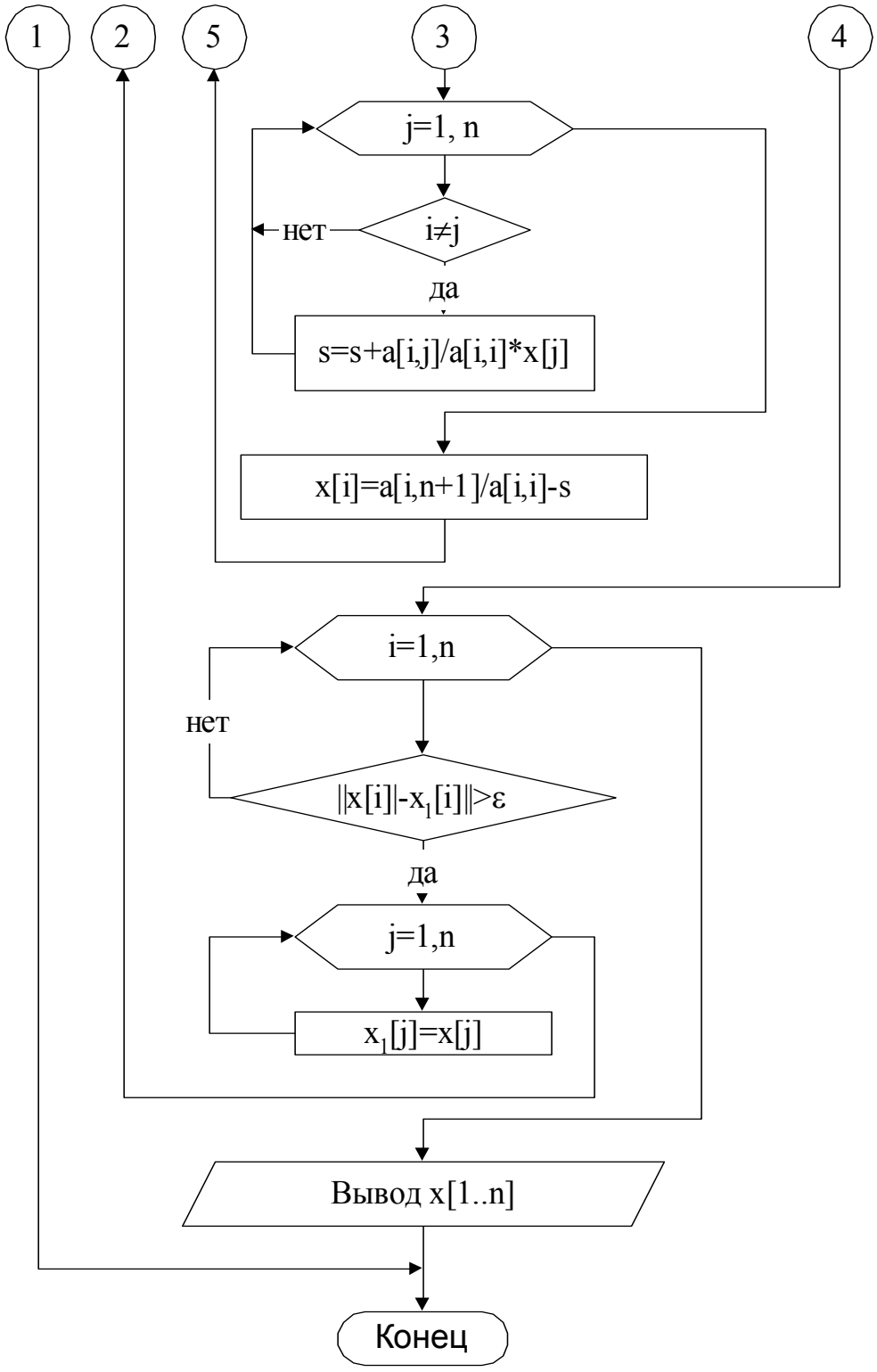
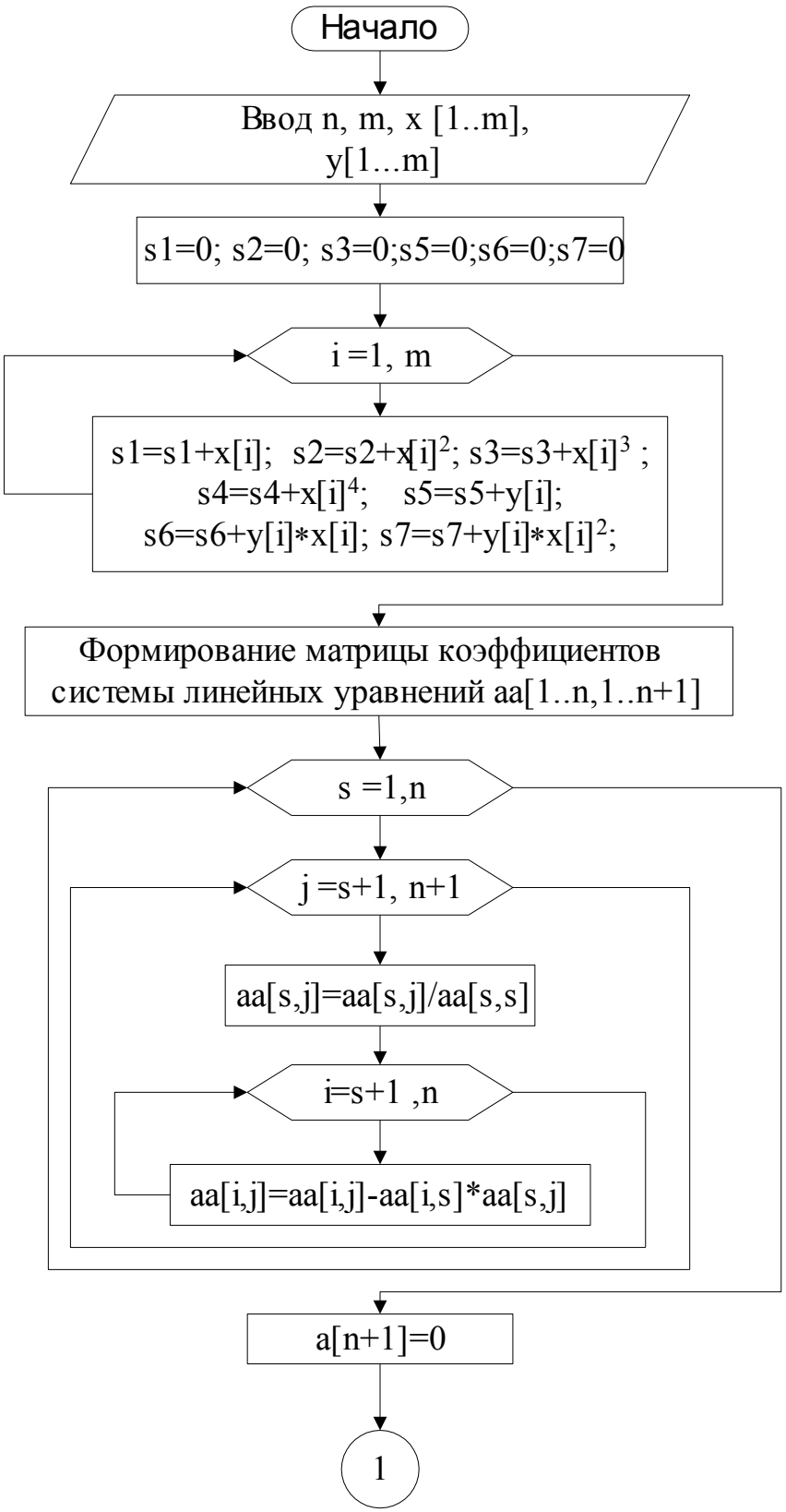


Рис.47. Блок-схема решения системы линейных уравнений методом Гаусса-Зейделя


Начало
Ввод n, m, x [1..m],
y[1…m]
s1=0; s2=0; s3=0;s5=0;s6=0;s7=0
i =1, m
s1=s1+x[i]; s2=s2+x[i]2; s3=s3+x[i]3 ;
s4=s4+x[i]4; s5=s5+y[i];
s6=s6+y[i]*x[i]; s7=s7+y[i]*x[i]2;
Формирование матрицы коэффициентов
системы линейных уравнений aa[1..n,1..n+1]
s =1,n
j =s+1, n+1
aa[s,j]=aa[s,j]/aa[s,s]
i=s+1 ,n
aa[i,j]=aa[i,j]-aa[i,s]*aa[s,j]
a[n+1]=0
1

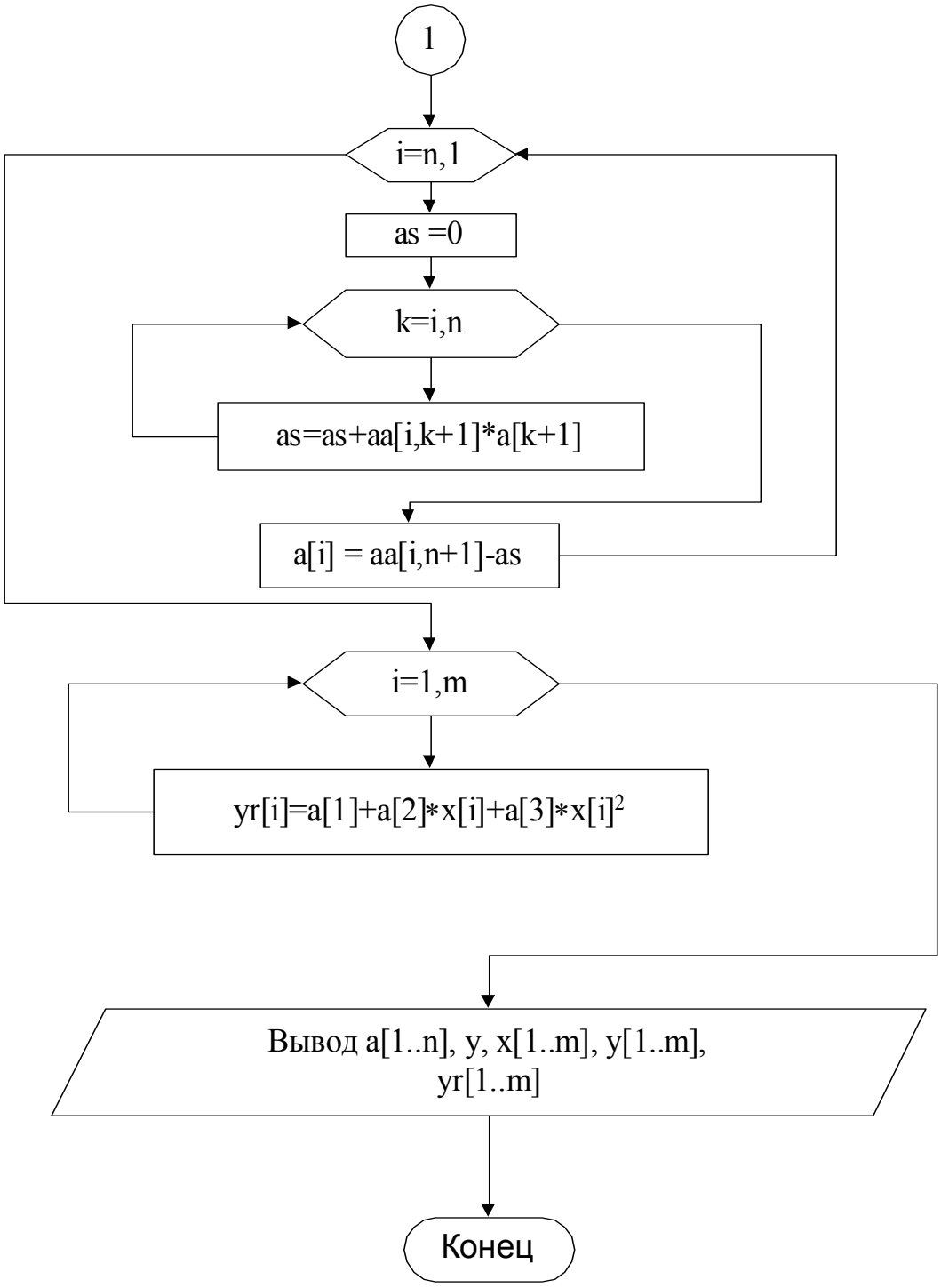


Рис. 48. Блок-схема расчета коэффициентов квадратичной аппроксимации зависимости теплоемкости циклопропана от температуры с использованием метода Гаусса

5.4. Системы нелинейных уравнений

Многие практические задачи сводятся к решению системы нелинейных уравнений.

Пусть для вычисления неизвестных $\mathbf{x_1, x_2, ... , x_n}$ требуется решить систему **n** нелинейных уравнений [ ].

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0, \\ f_2(x_1, x_2, \ldots, x_n) = 0, \\ \ldots\ldots\ldots\ldots\ldots, \\ f_n(x_1, x_2, \ldots, x_n) = 0. \end{cases} \qquad (5.4.1)$$

В отличие от систем линейных уравнений не существует прямых методов решения нелинейных систем общего вида. Лишь в некоторых случаях систему (5.4.1) можно решить непосредственно. Например, для случая двух уравнений иногда удается выразить одно неизвестное через другое и таким образом свести задачу к решению одного нелинейного уравнения относительно одного неизвестного. Для решения систем нелинейных уравнений обычно используют итерационные методы. Рассмотрим два из них - метод простой итерации и метод Ньютона.

5.4.1. Метод простой итерации

Систему уравнений (5.4.1) представим в следующем виде:

$$\begin{cases} x_1 = \varphi_1(x_1, x_2, \ldots, x_n), \\ x_2 = \varphi_2(x_1, x_2, \ldots, x_n), \\ \ldots\ldots\ldots\ldots\ldots, \\ x_n = \varphi_n(x_1, x_2, \ldots, x_n). \end{cases} \qquad (5.4.2)$$

Запишем систему (5.4.2) в векторной форме:

$$\mathbf{x} = \boldsymbol{\varphi}\,(\mathbf{x}) \qquad (5.4.3)$$

где $$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix}, \quad \varphi(\mathbf{x}) = \begin{bmatrix} \varphi_1(x) \\ \varphi_2(x) \\ . \\ . \\ . \\ \varphi_n(x) \end{bmatrix} . \qquad (5.4.4)$$

Для нахождения вектор - корня $\boldsymbol{x}^* = \left(x_1^*, x_2^*, ..., x_n^*\right)$ уравнения (5.4.3) часто удобно использовать *метод итерации*

$$\boldsymbol{x}^{(p+1)} = \boldsymbol{\varphi}(\boldsymbol{x}^{(p)}) \qquad (p = 0, 1, 2, ...), \qquad (5.4.5)$$

где начальное приближение $x^{(0)} \approx x^*$. Это грубое значение искомого корня. Заметим, что если процесс итерации (5.4.5) сходится, то предельное значение

$$\xi = \lim_{p\to\infty} \mathbf{x}^{(\mathbf{p})} \qquad (5.4.6)$$

обязательно является корнем уравнения (5.4.3). Действительно, предполагая, что соотношение (5.4.6) выполнено, и переходя к пределу в равенстве (5.4.5) при $p\to\infty$, в силу непрерывности функции $\boldsymbol{\varphi}(\boldsymbol{x})$ будем иметь:

$$\lim_{p\to\infty} \mathbf{x}^{(p+1)} = \varphi\left(\lim_{p\to\infty} \mathbf{x}^{(p)} \right),$$

т.е.

$$\xi = \boldsymbol{\varphi}(\xi).$$

Таким образом, ξ есть корень векторного уравнения (5.4.3).

Рассмотрим метод простой итерации на примере системы двух нелинейных уравнений с двумя неизвестными:

$$\begin{aligned} F_1(x, y) &= 0; \\ F_2(x, y) &= 0. \end{aligned} \qquad (5.4.7)$$

Решением системы (5.4.7) для x∈ [a,b] и y∈ [c,d] будут такие значения x* и y*, которые обращают эту систему в тождество. Необходимо найти x* и y* с заданной степенью точности ε.

Запишем систему (5.4.7) в эквивалентной форме:

$$\begin{aligned} x &= \varphi_1(x, y), \\ y &= \varphi_2(x, y). \end{aligned} \qquad (5.4.8)$$

Последовательные приближения будут вычисляться по формулам:

$$\begin{aligned} x_1 &= \varphi_1(x_0, y_0), \\ y_1 &= \varphi_2(x_0, y_0), \\ x_2 &= \varphi_1(x_1, y_1), \\ y_2 &= \varphi_2(x_1, y_1), \\ &\cdots\cdots\cdots\cdots \\ x_{n+1} &= \varphi_1(x_n, y_n), \\ y_{n+1} &= \varphi_2(x_n, y_n), \end{aligned} \qquad (5.4.9)$$

где x_0, y_0 - начальные приближения значения искомого корня.

Итерационный процесс можно считать законченным, как только выполнится неравенство:

$$|x_{n+1} - x_n| + |y_{n+1} - y_n| \le \varepsilon \quad \text{или}$$
$$|y_{n+1} - y_n| \le \varepsilon \qquad (5.4.10)$$

Для определения сходимости процесса имеет место следующая ***теорема.***

Пусть в некоторой заданной области x∈[a,b] и y∈[c,d] имеется единственное решение x*, y* системы (5.4.8) тогда, если:

- $\varphi_1(x,y)$ и $\varphi_2(x,y)$ определены и непрерывно дифференцируемы в заданной области;
- начальные приближения x_0, y_0 и все последующие приближения x_n, y_n принадлежат заданной области;
- в рассматриваемой области выполняются неравенства:

$$\left|\frac{\partial \varphi_1}{\partial x}\right| + \left|\frac{\partial \varphi_1}{\partial y}\right| \le 1,$$
$$\left|\frac{\partial \varphi_2}{\partial x}\right| + \left|\frac{\partial \varphi_2}{\partial y}\right| \le 1,$$

или (5.4.11)

$$\left|\frac{\partial \varphi_1}{\partial x}\right| + \left|\frac{\partial \varphi_2}{\partial x}\right| \le 1,$$

$$\left|\frac{\partial \varphi_1}{\partial y}\right| + \left|\frac{\partial \varphi_2}{\partial y}\right| \le 1.$$

то процесс последовательных приближений (5.4.9) сходится к решению системы уравнений (5.4.11).

Пример 5.4.1. Методом итерации приближенно решить систему

$$\begin{cases} x_1^2 + x_2^2 = 1, \\ x_1^3 - x_2 = 0. \end{cases}$$

Решение. Преобразуем данную систему к виду (5.4.2):

$\begin{cases} x_2 = \sqrt{1 - x_1^2}, \\ x_1 = \sqrt[3]{x_2}. \end{cases}$ Из графического построения (рис.49) видно, что

система имеет два решения, отличающиеся только знаком.

Ограничимся нахождением положительного решения. Из чертежа видим, что за начальное приближение положительного решения системы можно принять:

$$\mathbf{x}^{(0)} = \begin{bmatrix} 0.8 \\ 0.5 \end{bmatrix}.$$

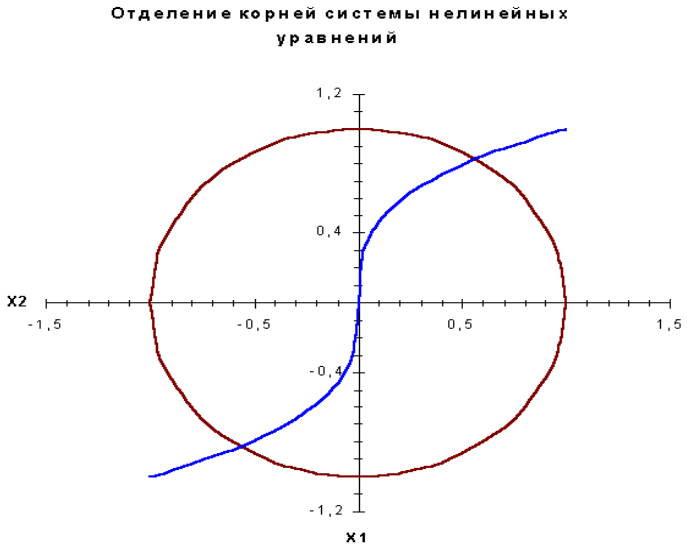


Рис. 49

Полагая

$$\varphi(\mathbf{x}) = \begin{cases} \sqrt[3]{x_2} \\ \sqrt{1 - x_1^2} \end{cases},$$

будем иметь

$$\mathbf{x}^{(1)} = \begin{bmatrix} \sqrt[3]{0.5} \\ \sqrt{1 - 0.8^2} \end{bmatrix} = \begin{bmatrix} 0.7937 \\ 0.6000 \end{bmatrix}.$$

Аналогично,

$$\mathbf{x}^{(1)} = \begin{bmatrix} \sqrt[3]{0.6} \\ \sqrt{1 - 0.7937^2} \end{bmatrix} = \begin{bmatrix} 0.8434 \\ 0.6083 \end{bmatrix},$$

$$\mathbf{x}^{(2)} = \begin{bmatrix} \sqrt[3]{0.6083} \\ \sqrt{1 - 0.8434^2} \end{bmatrix} = \begin{bmatrix} 0.8473 \\ 0.5372 \end{bmatrix}$$

и т.д.

Точное решение системы $x_1=0.8261$, $x_2=0.5636$.

Блок-схема метода итераций приведена на рис.50. Имеют место следующие обозначения неравенств (5.4.11):

$$\begin{aligned} \left|\frac{\partial \varphi_1}{\partial x}\right| + \left|\frac{\partial \varphi_1}{\partial y}\right| &= D_1, \\ \left|\frac{\partial \varphi_2}{\partial x}\right| + \left|\frac{\partial \varphi_2}{\partial y}\right| &= D_2, \\ \left|\frac{\partial \varphi_1}{\partial x}\right| + \left|\frac{\partial \varphi_2}{\partial x}\right| &= D_3, \\ \left|\frac{\partial \varphi_1}{\partial y}\right| + \left|\frac{\partial \varphi_2}{\partial y}\right| &= D_4. \end{aligned} \qquad (5.4.12)$$

5.4.2. Метод ньютона

Этот метод обладает более быстрой сходимостью, чем метод простой итерации. В случае одного уравнения f(x)=0 алгоритм метода Ньютона может быть получен путем записи уравнения касательной к кривой y=f(x) (линеризация кривой y=f(x)). В основе метода Ньютона для системы уравнений также лежит идея линеризации, а именно, использование разложения функций $f_i(x_1, x_2, ..., x_n)$ в ряд Тэйлора. Причем члены, содержащие вторые (и более высоких порядков) производные, отбрасываются.

Запишем систему (5.4.1) в векторной форме:

$$\mathbf{f(x)=0,} \qquad (5.4.13)$$

где $f = \begin{bmatrix} f_1 \\ f_2 \\ . \\ . \\ . \\ f_n \end{bmatrix}$, $x = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix}$.

Для решения системы (5.4.1) будем использовать метод последовательных приближений. Предположим, что найдено *p*-е приближение

$$\mathbf{x}^{(p)} = \left(x_1^{(p)}, x_2^{(p)}, ..., x_n^{(p)}\right)$$

одного из изолированных корней $\mathbf{x} = \left(x_1, x_2, ..., x_n\right)$ векторного уравнения (5.4.13). Тогда точный корень уравнения (5.4.1) можно представить в виде

$$\mathbf{x} = \mathbf{x}^{(p)} + \boldsymbol{\varepsilon}^{(p)}, \qquad (5.4.14)$$

где $\varepsilon^{(p)} = \left(\varepsilon_1^{(p)}, \varepsilon_2^{(p)}, ..., \varepsilon_n^{(p)}\right)$ - поправка (погрешность корня).

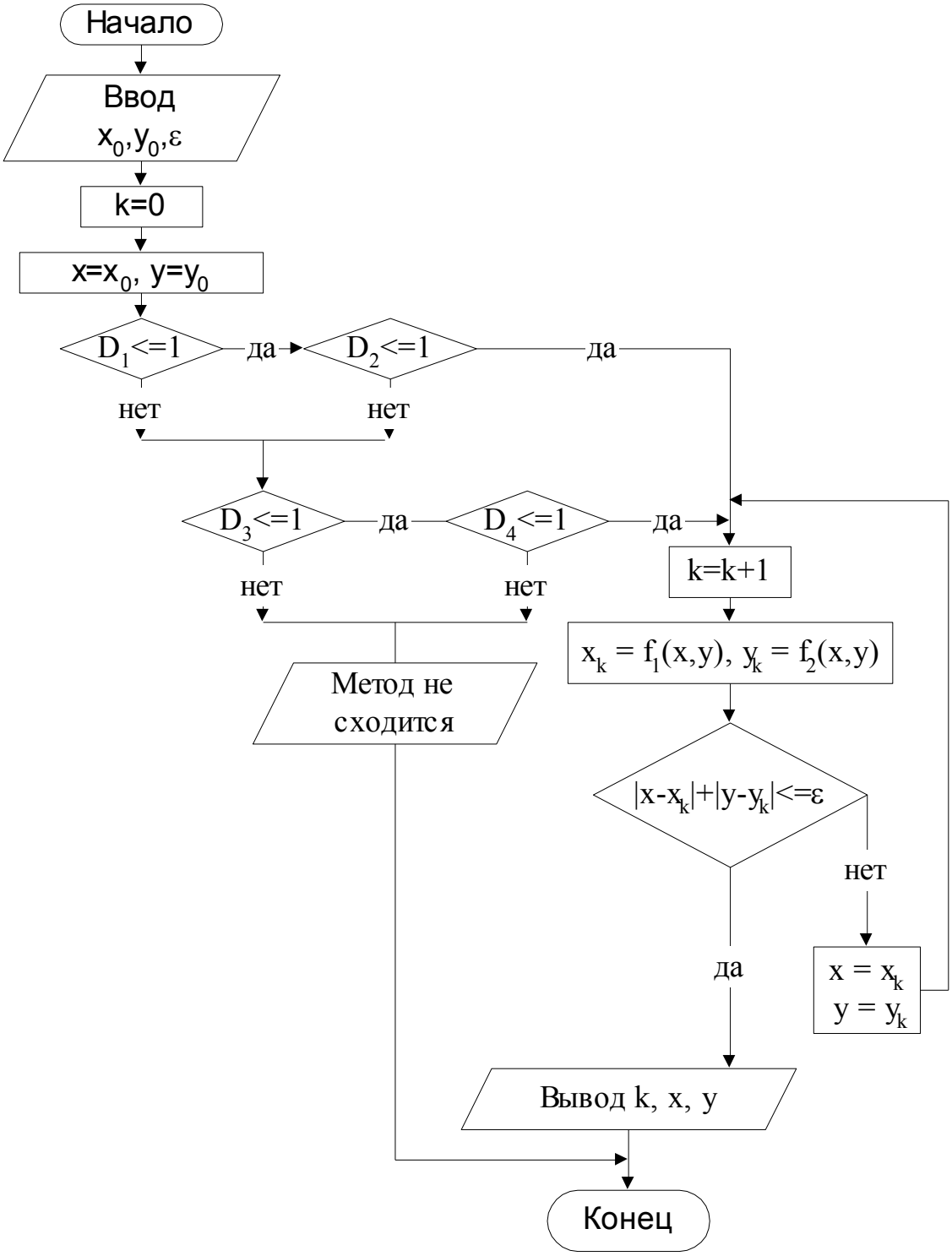


Рис. 50. Блок-схема метода простых итераций

Подставляя выражение (5.4.14) в уравнение (5.4.13), будем иметь:

$$\mathbf{f}\,(\mathbf{x}^{(p)} + \boldsymbol{\varepsilon}^{(p)}) = \mathbf{0}. \qquad (5.4.15)$$

Предполагая, что функция **f(x)** непрерывно дифференцируема в некоторой выпуклой области, содержащей **x** и $\mathbf{x}^{(p)}$, разложим левую часть уравнения (5.4.15) по степеням малого вектора $\boldsymbol{\varepsilon}^{(p)}$, ограничиваясь линейными членами.

$$\mathbf{f}\,(\mathbf{x}^{(p)} + \boldsymbol{\varepsilon}^{(p)}) = \mathbf{f}\,(\mathbf{x}^{(p)}) + \mathbf{f}\,'(\mathbf{x}^{(p)})\cdot\boldsymbol{\varepsilon}^{(p)} = \mathbf{0}. \qquad (5.4.16)$$

Из формулы (5.4.16) вытекает, что под производной **f′(x)** следует понимать *матрицу Якоби* системы функций f_1, f_2, ... , f_n относительно переменных x_1, x_2, ..., x_n, т.е.

$$\mathbf{f}'(\mathbf{x}) = \mathbf{W}(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_1}{\partial x_2} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_2}{\partial x_2} & \cdots & \dfrac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial f_n}{\partial x_1} & \dfrac{\partial f_n}{\partial x_2} & \cdots & \dfrac{\partial f_n}{\partial x_n} \end{bmatrix}, \qquad (5.4.17)$$

или в краткой записи

$$\mathbf{f}'(\mathbf{x}) = \mathbf{W}(\mathbf{x}) = \left[\frac{\partial f_i}{\partial x_j} \right], \quad i, j = 1, 2, \ldots, n. \qquad (5.4.18)$$

Система (5.4.16) представляет собой линейную систему относительно поправок $\varepsilon_i^{(p)}$ (i = 1, 2, ... , n) с матрицей **W(x)**, поэтому формула (5.4.16) может быть записана в следующем виде:

$$\mathbf{f}\,(\mathbf{x}^{(p)}) + \mathbf{W}\,(\mathbf{x}^{(p)})\cdot\boldsymbol{\varepsilon}^{(p)} = \mathbf{0}. \qquad (5.4.19)$$

Отсюда, предполагая, что матрица $\mathbf{W}\,(\mathbf{x}^{(p)})$ - неособенная, получим:

$$\boldsymbol{\varepsilon}^{(p)} = -\,\mathbf{W}^{-1}\,(\mathbf{x}^{(p)}) \cdot \mathbf{f}\,(\mathbf{x}^{(p)}).$$

Следовательно,

$$\mathbf{x}^{(p+1)} = \mathbf{x}^{(p)} - \mathbf{W}^{-1}(\mathbf{x}^{(p)}) \cdot \mathbf{f}(\mathbf{x}^{(p)}), \qquad p = 0, 1, 2, \ldots . \tag{5.4.20}$$

(*метод Ньютона*).

За нулевое приближение $\mathbf{x}^{(0)}$ можно взять грубое значение искомого корня.

В качестве примера рассмотрим метод Ньютона для решения системы двух уравнений:

$$\begin{aligned} F_1(x, y) &= 0, \\ F_2(x, y) &= 0. \end{aligned} \tag{5.4.21}$$

Пусть x_0 и y_0 - начальные приближенные значения неизвестных.

Последовательные приближения к решению системы по методу Ньютона вычисляются по формулам:

$$\begin{aligned} x_1 &= x_0 - \frac{1}{W(x_0, y_0)}\left(F_1(x_0, y_0)\frac{\partial F_2(x_0, y_0)}{\partial y} - F_2(x_0, y_0)\frac{\partial F_1(x_0, y_0)}{\partial y} \right), \\ y_1 &= y_0 - \frac{1}{W(x_0, y_0)}\left(F_2(x_0, y_0)\frac{\partial F_1(x_0, y_0)}{\partial x} - F_1(x_0, y_0)\frac{\partial F_2(x_0, y_0)}{\partial x} \right) \end{aligned} \tag{5.4.22}$$

$$\begin{aligned} x_{n+1} &= x_n - \frac{1}{W(x_n, y_n)}\left(F_1(x_n, y_n)\frac{\partial F_2(x_n, y_n)}{\partial y} - F_2(x_n, y_n)\frac{\partial F_1(x_n, y_n)}{\partial y} \right) \\ y_{n+1} &= y_n - \frac{1}{W(x_n, y_n)}\left(F_2(x_n, y_n)\frac{\partial F_1(x_n, y_n)}{\partial x} - F_1(x_n, y_n)\frac{\partial F_2(x_n, y_n)}{\partial x} \right) \end{aligned}$$

где

$$W(x_0, y_0) = \begin{vmatrix} \dfrac{\partial F_1(x_0, y_0)}{\partial x} & \dfrac{\partial F_1(x_0, y_0)}{\partial y} \\ \dfrac{\partial F_2(x_0, y_0)}{\partial x} & \dfrac{\partial F_2(x_0, y_0)}{\partial y} \end{vmatrix} \neq 0,$$

. . . . . ., (5.4.23)

$$W(x_n, y_n) = \begin{vmatrix} \dfrac{\partial F_1(x_n, y_n)}{\partial x} & \dfrac{\partial F_1(x_n, y_n)}{\partial y} \\ \dfrac{\partial F_2(x_n, y_n)}{\partial x} & \dfrac{\partial F_2(x_n, y_n)}{\partial y} \end{vmatrix} \neq 0.$$

Процесс вычисления по итерационным формулам (5.4.22) продолжается до тех пор, пока не будут выполнены условия:

$$|x_{n+1} - x_n| + |y_{n+1} - y_n| \le \varepsilon$$

или (5.4.24)

$$|x_{n+1} - x_n| \le \varepsilon \qquad |y_{n+1} - y_n| \le \varepsilon .$$

Блок-схема алгоритма метода Ньютона приведена на рис.51.

В формулах для расчета x_{n+1} и y_{n+1} системы (5.4.22) обозначим числитель: D_1 и D_2, а знаменатель- W.

Представим уравнение (5.4.22) в следующем виде:

$$x_{n+1} = x_n - \frac{F_1 \cdot F'_{2y} - F_2 \cdot F'_{1y}}{F'_{1x} \cdot F'_{2y} - F'_{2x} \cdot F'_{1y}};$$

$$y_{n+1} = y_n - \frac{F_2 \cdot F'_{1x} - F_1 \cdot F'_{2x}}{F'_{1x} \cdot F'_{2y} - F'_{2x} \cdot F'_{1y}}. \qquad (5.4.25)$$

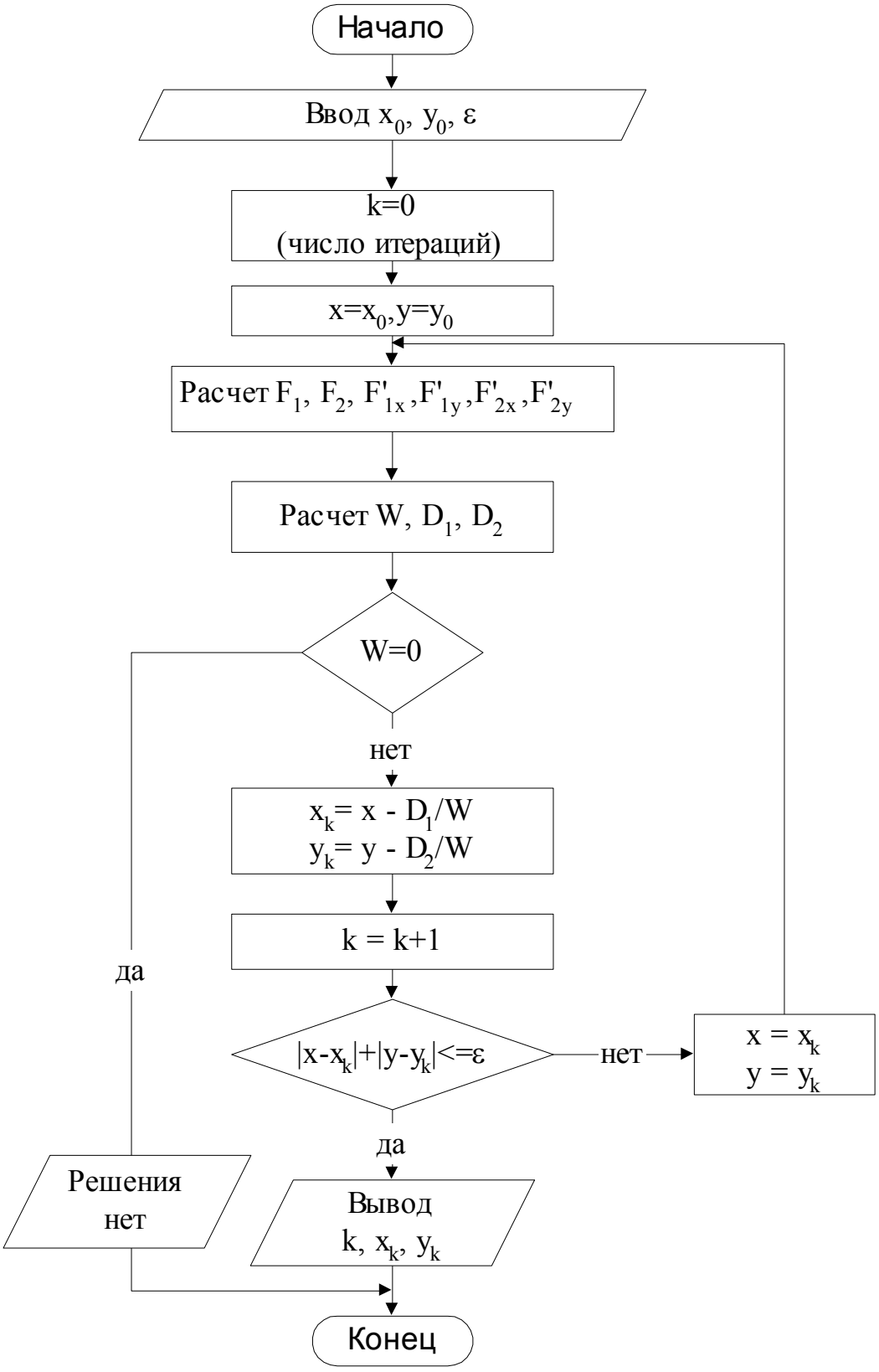


Рис. 51. Блок-схема метода Ньютона

5.5. МЕТОДЫ ОБРАБОТКИ ЭКСПЕРИМЕНТАЛЬНЫХ ДАННЫХ

5.5.1. Понятие о приближении функции

При исследовании химических и химико-технологических процессов, как правило, возникает необходимость в обработке и анализе данных, полученных в эксперименте, с последующим применением результатов обработки при моделировании и проектировании реальных процессов. Пусть дана некоторая функция y=f(x). Например, это функция выхода продуктов реакции от концентрации исходного компонента. Это означает, что любому значению **x** (концентрации исходного компонента) ставится в соответствие значение **y** (выход продукта реакции). На практике часто оказывается, что найти это значение достаточно сложно: функция f(x) является решением сложной задачи (сложный хроматографический анализ) или f(x) измеряется в дорогостоящем эксперименте. В этом случае вычисляют небольшую таблицу значений выходного параметра от аргумента и по некоторым точкам строят функцию y=f(x), где x – концентрация исходного компонента, y – концентрация продукта реакции.

Задача о приближении (*аппроксимации*) функции ставится следующим образом. Функцию f(x) , где x изменяется в некоторой области , требуется приближенно заменить (аппроксимировать) некоторым многочленом P_m (x) так, чтобы отклонение P_m (x) от f(x) в заданной области было наименьшим. Полином P_m (x) при этом называется *аппроксимирующим* . Для практики важен случай аппроксимации функции многочленом степени **m**

$$\mathbf{P_m(x) = a_0 + a_1x + a_2x^2 + \ldots + a_mx^m .} \qquad (5.5.1)$$

Коэффициенты $\mathbf{a_0}$, $\mathbf{a_1}$, ... , $\mathbf{a_m}$ должны подбираться так, чтобы отклонение многочлена от данной функции было наименьшим, а следовательно, задача о приближении функции состоит в определении a_0, $a_{1, \ldots}$, a_m полинома (5.5.1).

Если приближение строится на заданном дискретном множестве точек x_i, то аппроксимация называется *точечной*. К ней относятся интерполирование, среднеквадратичное приближение, равномерное приближение и т.д.

5.5.2. Интерполирование

В инженерных расчетах часто требуется установить функцию f(x) для всех значений **x** отрезка **[a,b]**, если известны ее значения в некотором конечном числе точек этого отрезка. Одним из способов приближения функции является *интерполяция* .

задача интерполяции может возникнуть в практике инженера при

- интерполировании табличных данных;
- получении функциональной зависимости по экспериментальным данным, представленным в табличной форме;
- замене сложной с вычислительной точки зрения функции, более простой зависимостью;
- при дифференцировании и интегрировании.

Постановка задачи

Пусть на отрезке $[x_0,x_n]$ заданы n+1 точки x_0, x_1, x_2,...,x_n, называемые узлами интерполяции, и значения некоторой интерполируемой функции y=f(x) в этих точках, т.е. имеется таблица экспериментальных значений функции y=f(x): y_0, y_1, y_2.....y_n.

$$\mathbf{y_0=f(x_0);\ \ y_1=f(x_1);\ \ ...;\ \ y_n=f(x_n).} \qquad \mathbf{(5.5.2)}$$

Требуется найти значения этой функции для промежуточных значений аргумента, не совпадающих с приведенными в таблице. Получить аналитическое выражение функции y= f(x) по таблице ее значений (5.5.2) в большинстве случаев невозможно. Поэтому вместо нее строят другую функцию, которая легко вычисляется и имеет ту же таблицу значений, что и f(x), т.е.

$$\mathbf{P_m(x_0)=f(x_0)=y_0,}$$
$$\mathbf{.....}$$
$$\mathbf{.....} \qquad \mathbf{(5.5.3)}$$
$$\mathbf{P_m(x_i)=f(x_i)=y_i\ ,\ \ где\ i = 0,1,2, ... , n.}$$

Такую задачу называют задачей *интерполирования.* Точки $\mathbf{x_i}$ называются *узлами интерполяции,* функция **f(x)** - *интерполируемой функцией,* многочлен $\mathbf{P_m(x)}$ - *интерполяционным многочленом.* Задачей интерполяции, в узком смысле слова, считают нахождение приближенных значений табличной функции при аргументах **x,** не совпадающих с

узловыми. Если значение аргумента **x** расположено между узлами $\mathbf{x_0 \le x \le x_n}$, то нахождение приближенного значения функции f(x) называется *интерполяцией*, если аппроксимирующую функцию вычисляют вне интервала $[x_0,x_n]$, то процесс называют *экстраполяцией*. Происхождение этих терминов связано с латинскими словами inter - между, внутри, pole - узел, extra - вне.

Графически задача интерполирования заключается в том, чтобы построить такую интерполирующую функцию, которая бы проходила через все узлы интерполирования (рис.52).

Близость интерполяционного многочлена к заданной функции состоит в том, что их значения совпадают на заданной системе точек.

При решении задачи интерполирования обычно принимается, что:

1) интерполируемая функция непрерывна на отрезке [a, b] и в каждой точке имеет конечные производные любого порядка;
2) узлы интерполирования отличны друг от друга.

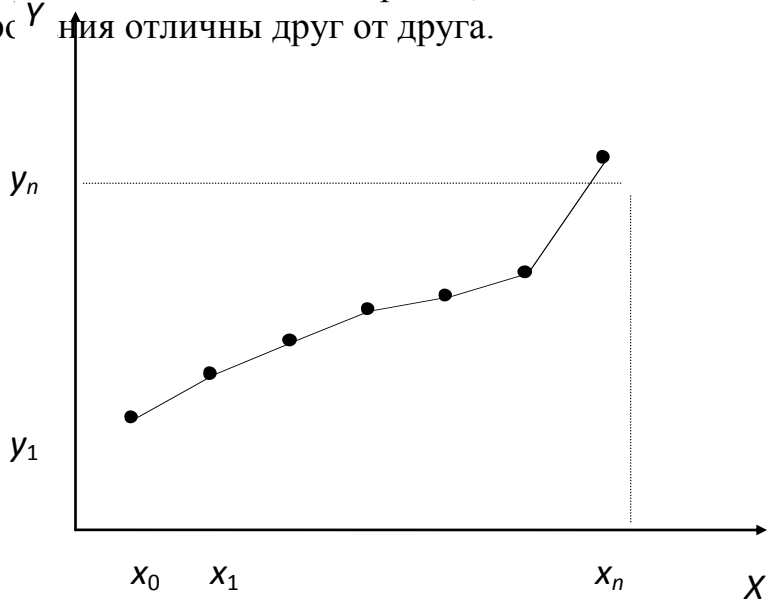


Рис.52

5.5.2.1.Линейная интерполяция

Простейшим и часто используемым видом интерполяции является *линейная интерполяция.* Она состоит в том, что заданные точки x_i, y_i при i=0,1,2,... n соединяются прямолинейными отрезками и функцию f(x) можно приближенно представить ломаной с вершинами в данных точках.

Уравнения каждого отрезка ломаной в общем случае разные. Поскольку имеется **n** интервалов (x_{i-1},x_i), то для каждого из них в качестве уравнения интерполяционного многочлена используется уравнение прямой, проходящей через две точки. В частности, для i-го интервала

можно написать уравнение прямой, проходящей через точки (x_{i-1},y_{i-1}) и (x_i,y_i), в виде

$$\frac{y-y_{i-1}}{y_i-y_{i-1}} = \frac{x-x_{i-1}}{x_i-x_{i-1}} \quad .$$

Отсюда

$$\mathbf{y = a_i x + b_i,} \qquad \mathbf{x_{i-1} \le x \le x_i,} \qquad (5.5.4)$$

$$\mathbf{a_i} = \frac{y_i-y_{i-1}}{x_i-x_{i-1}} \quad , \quad \mathbf{b_i = y_{i-1} - a_i x_{i-1}.}$$

Следовательно, при использовании линейной интерполяции сначала нужно определить интервал, в который попадает значение аргумента **x**, а затем подставить его в формулу (5.5.4) и найти приближенное значение функции в этой точке.

5.5.2.2. Интерполяционный многочлен Лагранжа

Пусть функция f(x) задана таблично. Это могут быть, например, значения концентраций продуктов реакции в зависимости от температуры, полученные экспериментально.

x	x_0	x_1	. . . .	x_n
f(x)	f_0	f_1	. . . .	f_n

Значения $x_0,x_1,...,x_n$ называются узлами таблицы. Считаем, что узлы в общем случае не являются равноотстоящими (шаг таблицы неравномерный).

Построим интерполяционный многочлен на отрезке $[x_0,x_n]$. Запишем искомый многочлен в виде

$$\mathbf{P_m(x) = a_0 + a_1 x + a_2 x^2 + . . . + a_m x^m .} \qquad (5.5.5)$$

Геометрически задача интерполирования сводится к построению кривой через заданные точки.

Аналитически задача сводится к решению системы уравнений

$$Y_i(x) = a_0 + a_1 x_i + a_2 x_i^2 + ... + a_m x_i^m . \qquad i = 0...n \qquad (5.5.6)$$

Для определения коэффициентов многочлена (5.5.5) необходимо располагать n+1 узловой точкой.

Чтобы система уравнений имела единственное решение, необходимо, чтобы количество неизвестных (коэффициенты полинома a_j) – **m+1** – равнялось количеству уравнений **n+1** или **m=n.**

Пусть в n+1-ой точках x_0, x_1,..., x_n определены значения y_0, y_1,..., y_n. Требуется построить многочлен Pn(x), принимающий в узловых точках заданные значения y_i, т.е. такой, что

$$\mathbf{P_n(x_i) = y_i, \qquad i= 0,1, ...,n.}$$

Лагранж предложил следующую форму интерполяционного полинома

$$Pn(x) = \sum_{i=0}^{n} Y_i \cdot L_i(x), \tag{5.5.7}$$

где $L_i(x)$ - множитель Лагранжа, имеющий вид:

$$L_i(x) = \frac{(X - X_0)..(X - X_{i-1})(X - X_{i+1})..(X - X_n)}{(X_1 - X_0)..(X_i - X_{i-1})(X_i - X_{i+1})..(X_i - X_n)} = \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{(X - X_k)}{(X_i - X_k)}. \tag{5.5.8}$$

Следовательно , формулу Лагранжа можно представить в виде:

$$\mathrm{P_n}(x) = \sum_{i=0}^{n} y_i \left(\prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{x - x_k}{x_i - x_k} \right). \tag{5.5.9}$$

Числитель и знаменатель не должны включать в себя значения $\mathbf{x = x_i}$, так как результат будет равен нулю. В развернутом виде формулу Лагранжа можно записать:

$$P_n(x) = Y_0 \frac{(X - X_1)(X - X_2)..(X - X_n)}{(X_0 - X_1)(X_0 - X_2)..(X_0 - X_n)} +$$

$$+ Y_1 \frac{(X - X_0)(X - X_2)..(X - X_n)}{(X_1 - X_0)(X_1 - X_2)..(X_1 - X_n)} + ... \tag{5.5.10}$$

$$...+ Y_n \frac{(X - X_0)(X - X_1)(X - X_2)..(X - X_{n-1})}{(X_n - X_0)(X_n - X_1)(X_n - X_2)..(X_n - X_{n-1})}.$$

Блок - схема метода Лагранжа приведена на рис.53.

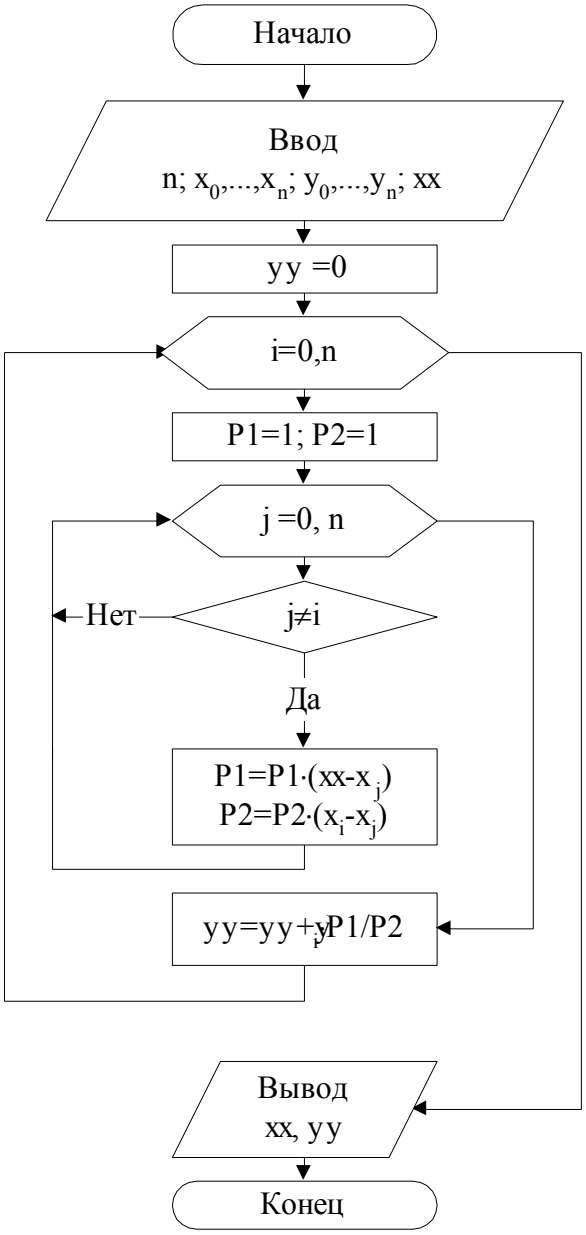


Рис. 53. Блок-схема метода Лагранжа

Пример 5.5.1. Для функции **y=sinπx** построить интерполяционный полином Лагранжа, выбрав узлы: x_0, x_1, x_2.

X	0	$\frac{1}{6}$	$\frac{1}{2}$
Y	0	$\frac{1}{2}$	1

Применяя формулу (5.5.10), получим:

$$P(X)=\frac{\left(X-\frac{1}{6}\right)\left(X-\frac{1}{2}\right)}{\left(0-\frac{1}{6}\right)\left(0-\frac{1}{2}\right)}\cdot 0+\frac{(X-0)\left(X-\frac{1}{2}\right)}{\left(\frac{1}{6}-0\right)\left(\frac{1}{6}-\frac{1}{2}\right)}\cdot\frac{1}{2}+\frac{(X-0)\left(X-\frac{1}{6}\right)}{\left(\frac{1}{2}-0\right)\left(\frac{1}{2}-\frac{1}{6}\right)}\cdot 1,$$

$$P(X)=\frac{7}{2}X-3\cdot X^2.$$

Пример 5.5.2. Дана таблица значений теплоёмкости вещества в зависимости от температуры C_p =f(T).

x (T)	300	400	500	600
y (Cp)	52.89	65.61	78.07	99.24

Вычислить значение теплоёмкости в точке T=450 K.

Для решения воспользуемся формулой (5.5.10).

$$f(450)=\frac{(450-400)(450-500)(450-600)}{(300-400)(300-500)(300-600)}\cdot 52.89+$$

$$+\frac{(450-300)(450-500)(450-600)}{(400-300)(400-500)(400-600)}\cdot 65.61+\frac{(450-300)(450-400)(450-600)}{(500-300)(500-400)(500-600)}\cdot 78.07+$$

$$+\frac{(450-300)(450-400)(450-500)}{(600-300)(600-400)(600-500)}\cdot 99.24\ .$$

Значение теплоемкости при температуре 450 К получим:

$$Cp(450) = 71.31 \text{ Дж} / (\text{моль} \cdot \text{К}).$$

5.5.2.3. Интерполяционные многочлены Ньютона

Конечные разности.

Пусть задана функция y=f(x) на отрезке $[x_0,x_n]$, который разбит на n одинаковых отрезков (случай равноотстоящих значений аргумента). $\Delta x=h=const$. Для каждого узла x_0, $x_1=x_0+h$, ..., $x_n=x_0+n\cdot h$ определены значения функции в виде:

$$f(x_0)=y_0,\ f(x_1)=y_1,\ ...,\ f(x_n)=y_n \qquad (5.5.11)$$

Введем понятие ***конечных разностей***.

Конечные разности первого порядка

$$\Delta y_0 = y_1 - y_0;$$
$$\Delta y_1 = y_2 - y_1;$$
$$. \quad . \quad . \quad . \quad .$$
$$\Delta y_{n-1} = y_n - y_{n-1}.$$

Конечные разности второго порядка

$$\Delta^2 y_0 = \Delta y_1 - \Delta y_0;$$
$$\Delta^2 y_1 = \Delta y_2 - \Delta y_1;$$
$$. \quad . \quad . \quad . \quad . \quad .$$
$$\Delta^2 y_{n-2} = \Delta y_{n-1} - \Delta y_{n-2}.$$

Аналогично определяются конечные разности высших порядков:

$$\Delta^k y_0 = \Delta^{k-1} y_1 - \Delta^{k-1} y_0;$$
$$\Delta^k y_1 = \Delta^{k-1} y_2 - \Delta^{k-1} y_1;$$
$$. \quad . \quad . \quad . \quad . \quad .$$
$$\Delta^k y_i = \Delta^{k-1} y_{i+1} - \Delta^{k-1} y_i \quad , \quad i = 0,1,...,n-k.$$

Конечные разности функций удобно располагать в таблицах, которые могут быть горизонтальными (табл.10) или диагональными (табл.11) таблицами разностей.

Таблица 10

Горизонтальная таблица

x_i	y_i	Δy_i	$\Delta^2 yi$	$\Delta^3 y_i$	$\Delta^4 y_i$	$\Delta^5 y_i$
x_0	y_0	Δy_0	$\Delta^2 y_0$	$\Delta^3 y_0$	$\Delta^4 y_0$	$\Delta^5 y_0$
x_1	y_1	Δy_1	$\Delta^2 y_1$	$\Delta^3 y_1$	$\Delta^4 y_1$	
x_2	y_2	Δy_2	$\Delta^2 y_2$	$\Delta^3 y_2$		
x_3	y_3	Δy_3	$\Delta^2 y_3$			
x_4	y_4	Δy_4				
x_5	y_5					

Таблица 11

Диагональная таблица

x_i	y_i	Δy_i	$\Delta^2 yi$	$\Delta^3 y_i$	$\Delta^4 y_i$	$\Delta^5 y_i$
x_0	y_0					
		Δy_0				
x_1	y_1		$\Delta^2 y_0$			
		Δy_1		$\Delta^3 y_0$		
x_2	y_2		$\Delta^2 y_1$		$\Delta^4 y_0$	
		Δy_2		$\Delta^3 y_1$		$\Delta^5 y_0$
x_3	y_3		$\Delta^2 y_2$		$\Delta^4 y_1$	
		Δy_3		$\Delta^3 y_2$		
x_4	y_4		$\Delta^2 y_3$			
		Δy_4				
x_5	y_5					

Первая интерполяционная формула Ньютона

Пусть для функции y = f(x) заданы значения $y_i = f(x_i)$ для равностоящих значений независимых переменных:

$\mathbf{x_n = x_0 + nh,}$ где h - шаг интерполяции.

Необходимо найти полином $P_n(x)$ степени не выше n, принимающий в точках (узлах) x_i значения:

$$P_n(x_i) = y_i, \qquad i=0,...,n. \tag{5.5.12}$$

Интерполирующий полином ищется в виде:

$$\begin{gathered} P_n(X) = a_0 + a_1(X - X_0) + a_2(X - X_0)(X - X_1) + ... \\ ... + a_n(X - X_0)..(X - X_{n-1}). \end{gathered} \tag{5.5.13}$$

Задача построения многочлена сводится к определению коэффициентов a_i из условий:

$$\begin{gathered} P_n(x_0)=y_0, \\ P_n(x_1)=y_1, \\ . \quad . \quad . \quad . \\ P_n(x_n)=y_n. \end{gathered} \tag{5.5.14}$$

Полагаем в (5.5.13) x = x_0, тогда, т.к. второе, третье и другие слагаемые равны 0,

$$P_n(x_0) = y_0 = a_0; \quad a_0=y_0.$$

Найдем коэффициент $\mathbf{a_1}$.

При $\mathbf{x = x_1}$ получим:

$$P_n(x_1) = y_1 = y_0 + a_1(x_1-x_0);$$

$$y_1 = y_0 + a_1(x_1-x_0),$$

$$a_1 = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\Delta y_0}{h}.$$

Для определения $\mathbf{a_2}$, составим конечную разность второго порядка.

При $\mathbf{x = x_2}$ получим:

$$P_n(x_2) = y_2 = y_0 + \frac{\Delta y_0}{h}(x_2 - x_0) + a_2(x_2-x_0)(x_2-x_1) = y_0 + 2\Delta y_0 + a_2 2h^2,$$

$$a_2 = \frac{y_2 - y_0 - 2\Delta y_0}{2h^2} = \frac{y_2 - y_0 - 2y_1 + 2y_0}{2h^2} = \frac{y_2 - 2y_1 + y_0}{2h^2} =$$

$$= \frac{(y_2 - y_1) - (y_1 - y_0)}{2h^2} = \frac{\Delta y_1 - \Delta y_0}{2h^2} = \frac{\Delta^2 y_0}{2!h^2}.$$

Аналогично можно найти другие коэффициенты. Общая формула имеет вид.

$$a_k = \frac{\Delta^k y_0}{k! h^k} \quad , \qquad k = 1 \ldots n. \qquad (5.5.15)$$

Подставляя эти выражения в формулу (5.5.13), получаем:

$$P_n(X) = Y_0 + \frac{\Delta Y_0}{1! \cdot h}(X - X_0) + \frac{\Delta^2 Y_0}{2! \cdot h^2}(X - X_0)(X - X_1) + \ldots$$
$$\ldots + \frac{\Delta^n Y_0}{n! \cdot h^n}(X - X_0)\ldots(X - X_{n-1}), \qquad (5.5.16)$$

где $\mathbf{x_i}$ **,**$\mathbf{y_i}$ - узлы интерполяции; **x** - текущая переменная; **h** - разность между двумя узлами интерполяции h – величина постоянная, т.е. узлы интерполяции равноотстоят друг от друга.

Этот многочлен называют *интерполяционным полиномом Ньютона* для интерполяции в начале таблицы (интерполирование «вперед») или *первым полиномом Ньютона.*

Для практического использования этот полином записывают в преобразованном виде, вводя обозначение $t=(x-x_0)/h$, тогда

$$P_n(X) = Y_0 + t \cdot \Delta Y_0 + \frac{t \cdot (t-1)}{2!} \cdot \Delta^2 Y_0 + \ldots + \frac{t \cdot (t-1)\ldots(t-n+1)}{n!} \cdot \Delta^n Y_0 . \quad (5.5.17)$$

Эта формула применима для вычисления значений функции для значений аргументов, близких к началу интервала интерполирования.

Блок-схема алгоритма метода Ньютона для интерполирования «вперед» приведена на рис. 54, программа - в Приложении.

Пример 5.5.3. Дана таблица значений теплоёмкости вещества в зависимости от температуры Ср =f(T) (табл.12).

Таблица 12

x (T)	300	400	500	600
Y (Cp)	52.88	65.61	78.07	99.24

Построить интерполяционный многочлен Ньютона для заданных значений функции.

n=3; h=100.

Составим таблицу конечных разностей функции.

Таблица 13

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$
300	52.88	12.73	-0.27	8.98
400	65.61	12.46	8.71	
500	78.07	21.17		
600	99.24			

Воспользуемся формулой (5.5.16):

$$P_3(x) = y_0 + \frac{\Delta y_0}{h}(x - x_0) + \frac{\Delta^2 y_0}{2!h^2}(x - x_0)(x - x_1) +$$

$$+ \frac{\Delta^3 y_0}{3!h^3}(x - x_0)(x - x_1)(x - x_2),$$

$$P_3(x) = 52.88 + \frac{12.73}{100}(x - 300) - \frac{0.27}{2!100^2}(x - 300)(x - 400) +$$

$$+ \frac{8.98}{3!100^3}(x - 300)(x - 400)(x - 500).$$

После выполнения преобразований получим интерполяционный многочлен вида:

$$P_3(x) = 1.5 \cdot 10^{-6} x^3 - 0.00181x^2 + 0.842x - 76.94.$$

Полином имеет третью степень и дает возможность вычисления при помощи найденной формулы значения y для неизвестного x .

Пример 5.5.4. в табл. 12 приведены значения теплоемкости в зависимости от температуры. определить значение теплоёмкости в точке т= 450 к.

Воспользуемся первой интерполяционной формулой ньютона. конечные разности рассчитаны в предыдущем примере, табл. 13, запишем интерполяционный многочлен при x=450 к.

$$\mathrm{P}_3(450) = 52.88 + \frac{12.73}{100}(450 - 300) - \frac{0.27}{2!100^2}(450 - 300)(450 - 400) +$$

$$+ \frac{8.98}{3!100^3}(450 - 300)(450 - 400)(450 - 500) = 71.31$$

Таким образом, теплоемкость при температуре 450 к будет

$$Cp(450)=71,31 Дж/(моль \cdot К).$$

Значение теплоемкости при Т=450 К получили такое же, что и рассчитанное по формуле Лагранжа.

Вторая интерполяционная формула Ньютона

Для нахождения значений функций в точках, расположенных в конце интервала интерполирования, используют второй интерполяционный полином Ньютона.
запишем интерполяционный многочлен в виде

$$P_n(X) = a_0 + a_1(X - X_n) + a_2(X - X_n)(X - X_{n-1}) + \ldots$$
$$\ldots + a_n(X - X_n)(X - X_{n-1})\ldots(X - X_1). \qquad (5.5.18)$$

Коэффициенты $a_0, a_1, \ldots, a_n$ определяем из условия:

$$P_n(x_i) = y_i, \quad i=0,\ldots n.$$

Полагаем в (5.5.18) $x = x_n$, тогда

$$P_n(x_n) = a_0,$$
$$P_n(x_n) = y_n = a_0,$$
$$a_0 = y_n.$$

Полагаем $x=x_{n-1}$, тогда

$$P_n(x_{n-1})=y_{n-1}=y_n+a_1(x_{n-1}-x_n), \qquad h=x_n-x_{n-1},$$

следовательно,

$$a_1 = \frac{y_n - y_{n-1}}{h} = \frac{\Delta y_{n-1}}{h}.$$

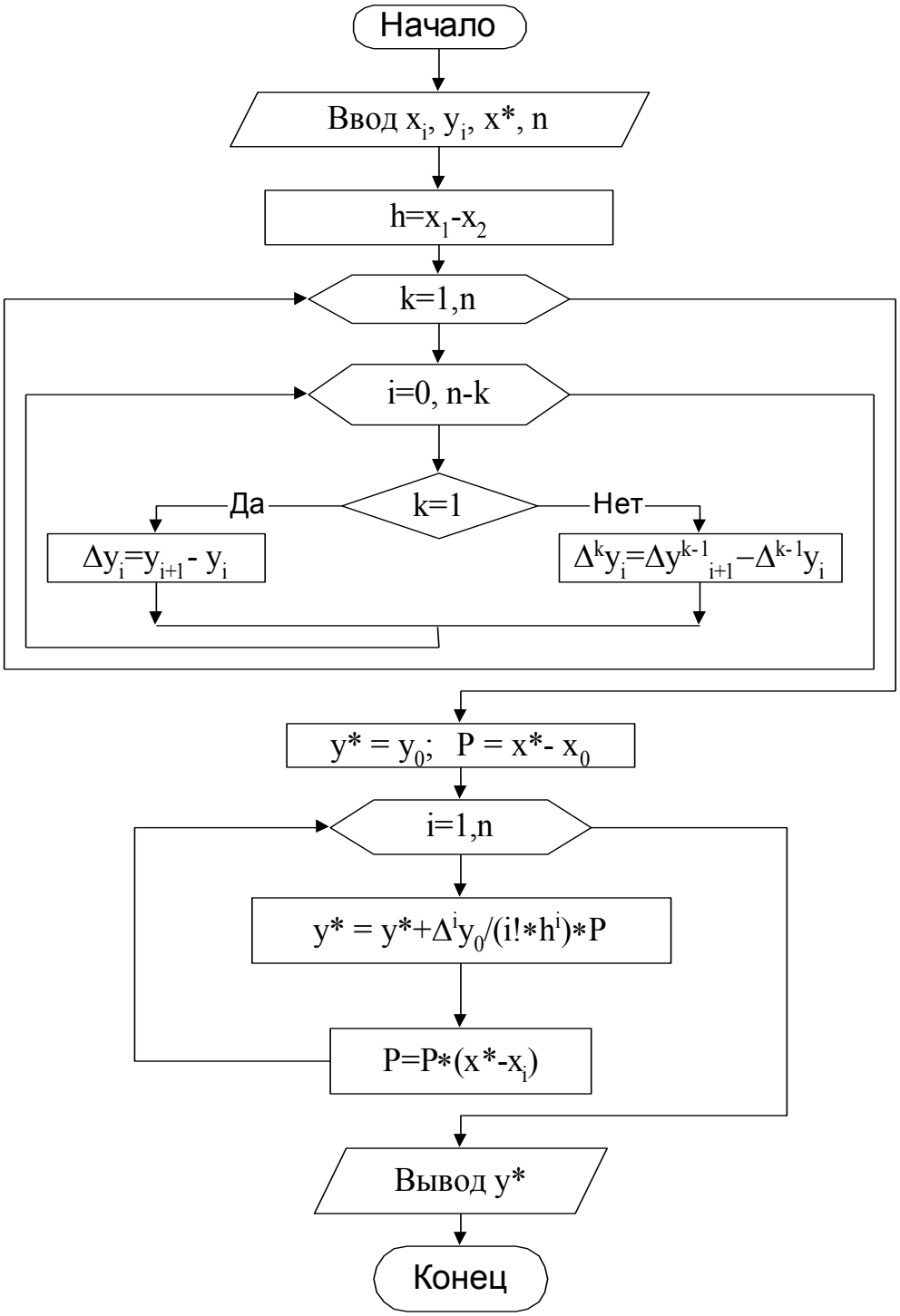


Рис. 54. Блок-схема метода Ньютона для интерполирования "вперед"

Если $\mathbf{x=x_{n-2}}$, то

$$P_n(x_{n-2}) = y_{n-2} = y_n + \frac{\Delta y_{n-1}}{h}(x_{n-2} - x_n) + a_2(x_{n-2} - x_n)(x_{n-2} - x_{n-1}),$$

$$y_{n-2} = y_n + \frac{\Delta y_{n-1}}{h}(-2h) + a_2 \cdot 2h^2 = y_n - 2\Delta y_{n-1} + a_2 2h^2,$$

$$a_2 = \frac{\Delta^2 y_{n-2}}{2!h^2}.$$

Аналогично можно найти другие коэффициенты многочлена (5.5.18).

$$a_3 = \frac{\Delta^3 y_{n-3}}{3!h^3}, \qquad (5.5.19)$$

$$a_n = \frac{\Delta^n y_0}{n!h^n}.$$

Подставляя эти выражения в формулу (5.5.18), получим ***вторую интерполяционную формулу Ньютона*** или многочлен Ньютона для интерполирования «назад».

$$P_n(x) = y_n + \frac{\Delta y_{n-1}}{h}(x - x_n) + \frac{\Delta^2 y_{n-2}}{2!h^2}(x - x_n)(x - x_{n-1}) +$$
$$+ \frac{\Delta^3 y_{n-3}}{3!h^3}(x - x_n)(x - x_{n-1})(x - x_{n-2}) + \ldots \qquad (5.5.20)$$
$$\ldots + \frac{\Delta^n y_0}{n!h^n}(x - x_n)(x - x_{n-1})\ldots(x - x_1).$$

Введем обозначения:

$$\frac{x - x_n}{h} = t \quad \text{или} \quad x = x_n + th,$$

$$\frac{x - x_{n-1}}{h} = \frac{x - (x_n - h)}{h} = t + 1,$$

$$\frac{x - x_{n-2}}{h} = \frac{x - (x_{n-1} - 2h)}{h} = t + 2,$$

……..……..……..……

$$\frac{x - x_1}{h} = \frac{x(x_n - 2h)}{h} = t + n - 1.$$

Произведя замену в (5.5.20), получим

$$P_n(x) = y_n + t\Delta y_{n-1} + \frac{t(t+1)}{2!}\Delta^2 y_{n-2} + \ldots + \frac{t(t+1)\ldots(t+n-1)}{n!}\Delta^n y_0 \qquad (5.5.21)$$

Это вторая формула Ньютона для интерполирования «назад».

Пример 5.5.5. Вычислить теплоемкость (табл. 12) для температуры Т=550 К.

Воспользуемся второй формулой Ньютона (5.5.19) и соответствующими конечными разностями (табл.13)

$$P_3(x) = y_3 + \frac{\Delta y_2}{h}(x - x_3) + \frac{\Delta^2 y_1}{2!h^2}(x - x_3)(x - x_2) +$$

$$+ \frac{\Delta^3 y_0}{3!h^3}(x - x_3)(x - x_2)(x - x_1),$$

$$P_3(550) = 99.24 + \frac{21.17}{100}(550 - 600) + \frac{8.71}{2!100^2}(550 - 600)(550 - 500) +$$

$$+ \frac{8.98}{3!100^3}(550 - 600)(550 - 500)(550 - 400) = 87.01.$$

Следовательно, значение теплоемкости при температуре 550 К равно:

Ср(550)=87,01Дж/(моль ∗К).

5.5.3. Аппроксимация функций

В предыдущих разделах был рассмотрен один из способов приближения функции к табличным данным – интерполяция. Отличительной особенностью ее являлось то, что интерполирующая функция строго проходила через узловые точки таблицы, т.е. рассчитанные значения совпадали с табличными – $y_i=f(x_i)$. Эта особенность обуславливалась тем, что количество коэффициентов в интерполирующей функции (m) было равно количеству табличных значений (n). Однако, если для описания табличных данных будет выбрана функция с меньшим количеством коэффициентов (m<n), что часто встречается на практике, то уже нельзя подобрать коэффициенты функции так, чтобы функция проходила через каждую узловую точку. В лучшем случае она будет проходить каким – либо образом между ними и очень близко к ним (см. рис. 55). Такой способ описания табличных данных называется аппроксимацией, а функция – аппроксимирующей.

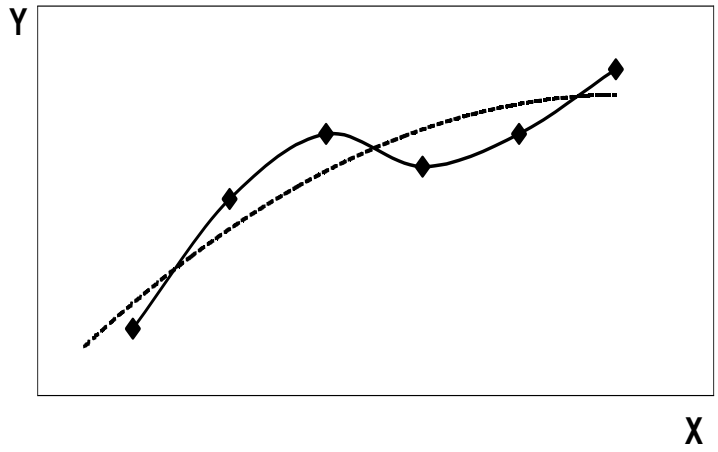


Рис. 55

– интерполирующая функция, - - аппроксимирующая функция.

Казалось бы, с помощью метода интерполяции можно описать табличные данные более точно, чем аппроксимации, тем не менее, на практике возникают ситуации, когда последний предпочтительнее.

1. Когда количество табличных значений очень велико. В этом случае интерполирующая функция будет очень громоздкой. Удобнее выбрать более простую в применении функцию с небольшим количеством коэффициентов, хотя и менее точную.

2. Когда вид функции заранее определен. Такая ситуация возникает, если требуется описать экспериментальные точки какой – либо теоретической зависимостью. Например, константа скорости химической реакции зависит от температуры по уравнению Аррениуса $k=k_0 \cdot \exp(-E/RT)$, в котором два определяемых параметра k_0 – предэкспоненциальный множитель, E – энергия активации. А так как почти всегда экспериментальных точек бывает больше двух, то и возникает необходимость в аппроксимации.

3. Аппроксимирующая функция может сглаживать погрешности эксперимента, в отличие от интерполирующей функции. Так, на рис.56. точками показаны табличные данные – результат некоторого эксперимента. Очевидно, что Y монотонно возрастает с увеличением X, а разброс данных объясняется погрешностью эксперимента.

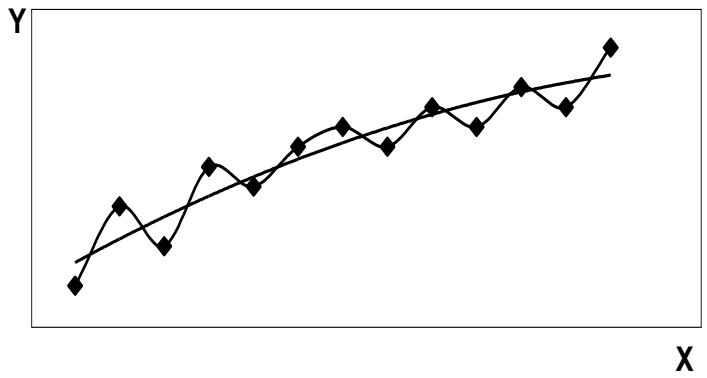


Рис 56

Однако интерполирующая функция, проходя через каждую точку, будет повторять ошибки эксперимента, иметь множество экстремумов: минимумов и максимумов – и в целом неверно отображать характер зависимости Y от X. Этого недостатка лишена аппроксимирующая функция.

3. И, наконец, интерполирующей функцией невозможно описать табличные данные, в которых есть несколько точек с одинаковым значением аргумента. А такая ситуация возможна, если один и тот же эксперимент проводится несколько раз при одних и тех же исходных данных. Однако это не является ограничением для использования аппроксимации, где не ставится условие прохождения графика функции через каждую точку.

Постановка задачи

Пусть, изучая неизвестную функциональную зависимость $y=f(x)$, был произведен ряд измерений величин x и y.

x	x_1	x_2	x_3	…	x_n
y	y_1	y_2	y_3	…	y_n

Если аналитическое выражение функции $f(x)$ неизвестно или весьма сложно, то возникает практически важная задача: найти такую эмпирическую формулу

$$\widetilde{y} = \widetilde{f}(x) , \qquad (5.5.22)$$

значения которой при $x=x_i$ возможно мало отличались бы от опытных данных y_i ($i=1, 2, \ldots , n$).

Как правило, указывают достаточно узкий класс функций K (например, множество функций линейных, степенных, показательных и т.п.), которому должна принадлежать искомая функция $\widetilde{f}(x)$. Таким образом, задача сводится к нахождению наилучших значений параметров.

Геометрически задача построения эмпирической формулы состоит в проведении кривой Г, «возможно ближе» примыкающей к системе точек

$$M_i(x_i, y_i) \quad (i = 1, 2, \ldots, n) \quad (\text{рис.57}).$$

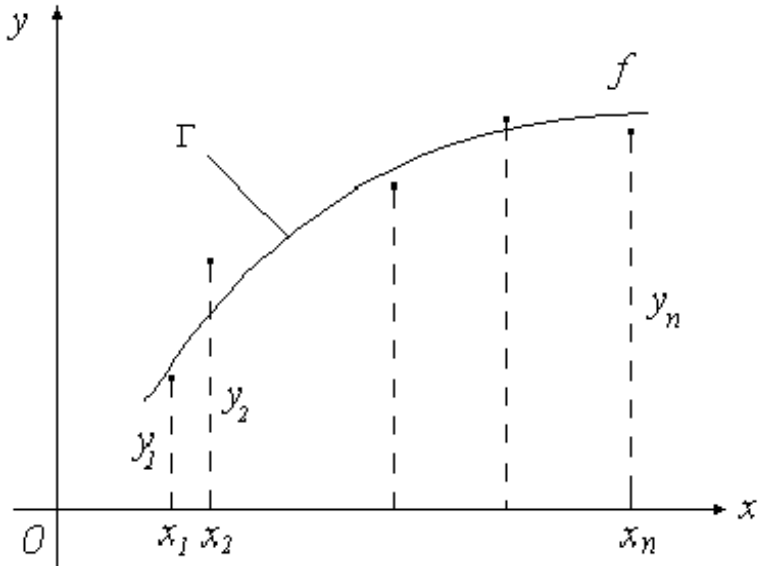


Рис. 57

Следует отметить, что задача построения эмпирической формулы отлична от задачи интерполирования. Известно, что эмпирические данные x_i и y_i, как правило, приближенные и содержат ошибки. Поэтому интерполяционная формула повторяет эти ошибки и не является идеальным решением поставленной задачи. Весьма вероятно, что более простая эмпирическая зависимость будет сглаживать данные и не будет повторять ошибки, как в случае интерполирования. График эмпирической зависимости не проходит через заданные точки, как это имеет место в случае интерполяции.

Построение эмпирической зависимости слагается из двух этапов:

- выяснение общего вида формулы;
- определение наилучших параметров эмпирической зависимости.

Если неизвестен характер зависимости между данными величинами x и y, то вид эмпирической формулы является произвольным. Предпочтение отдается простым формулам, обладающим хорошей точностью. Если отсутствуют сведения о промежуточных данных, то обычно предполагается, что эмпирическая функция аналитическая, без точек разрыва, и график ее – плавная кривая.

Удачный подбор эмпирической формулы в значительной мере зависит от опыта и искусства составителя. Во многих случаях задача состоит в аппроксимации неизвестной функциональной зависимости между x и y многочленом заданной степени m

$$P_m(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m = \sum_{k=0}^{m} a_k x^k$$

или

$$y = \sum_{k=0}^{m} a_k \cdot x^k \ .$$

Нередко употребляются другие элементарные функции (дробно-линейная, степенная, показательная, логарифмическая и т.п.). Что касается определения наилучших значений параметров, входящих в эмпирическую формулу, то эта задача более легкая и решается регулярными методами. Наиболее употребительным методом определения параметров эмпирической формулы является *метод наименьших квадратов*.

5.5.3.1. Метод наименьших квадратов

Пусть, например, в результате эксперимента получена таблица значений функции y_i $(i=1,...,n)$. Задача состоит в аппроксимации неизвестной функциональной зависимости между x и y эмпирической формулой :

$$y(x) = \tilde{f}(x; a_0, a_1, \ldots, a_m), \qquad (5.5.23)$$

где m – число параметров;

$a_1 \ldots a_m$ – неизвестные коэффициенты.

Для отыскания неизвестных коэффициентов применим метод наименьших квадратов, который является универсальным методом решения задач аппроксимации.

Идея метода наименьших квадратов заключается в следующем: определить искомые коэффициенты a_j зависимости (5.5.23) таким образом, чтобы этот полином наилучшим образом описывал экспериментальные данные, а сумма квадратов отклонений экспериментальных значений y_i от соответствующих значений, вычисленных по аппроксимирующему многочлену (5.5.23), была минимальной.

$$F(a_0, a_1, \ldots, a_m) = \sum_{i=1}^{n} \left[y_i - \tilde{f}(x_i; a_0, a_1, \ldots, a_m) \right]^2 = \min \ , \qquad (5.5.24)$$

где $F(a_0, a_1, \ldots, a_m)$ – функция коэффициентов.

В точке минимума функции F ее производные обращаются в нуль. Отсюда получаем так называемую *нормальную систему* для определения коэффициентов a_j $(j=0, 1, ..., m)$.

$$\frac{\partial F}{\partial a_0}=0,\ \frac{\partial F}{\partial a_1}=0,\ \ldots,\ \frac{\partial F}{\partial a_m}=0. \qquad (5.5.25)$$

Если система (5.5.25) имеет единственное решение, то оно будет искомым. Система (5.5.25) упрощается, если эмпирическая функция $\tilde{f}(x;a_0,a_1,...,a_m)$ линейная относительно параметров $a_0, a_1, ..., a_m$.

Рассмотрим наиболее распространенный частный случай, когда эмпирическая функция представлена полиномом:

$$P(x)=a_0+a_1\cdot x+a_2\cdot x^2+...+a_m\cdot x^m\ . \qquad (5.5.26)$$

$$F(a_0,a_1,\ldots,a_m)=\sum_{i=1}^{n}\left[y_i-P(x_i)\right]^2\to\min.$$

Используя систему (5.5.25), получаем математическое условие минимума для уравнения (5.5.26):

$$\frac{\partial F}{\partial a_0}=-2\cdot\sum_{i=1}^{n}\Big(y_i-(a_0-a_1\cdot x_i+a_2\cdot x^2+\ldots+a^m\cdot x_i^m)\Big)\cdot 1=0,$$

$$\frac{\partial F}{\partial a_1}=-2\cdot\sum_{i=1}^{n}\Big(y_i-(a_0-a_1\cdot x_i+a_2\cdot x^2+\ldots+a^m\cdot x_i^m)\Big)\cdot x_i=0, \qquad (5.5.27)$$

……………………………………………………………………………

$$\frac{\partial F}{\partial a_m}=-2\cdot\sum_{i=1}^{n}\Big(y_i-(a_0-a_1\cdot x_i+a_2\cdot x^2+\ldots+a^m\cdot x_i^m)\Big)\cdot x_i^m=0.$$

В результате решения системы линейных уравнений получим коэффициенты $a_0,a_1,...,a_m$ многочлена (5.5.26).

Метод наименьших квадратов обладает тем преимуществом, что, если сумма квадратов отклонений *F* мала, то сами эти отклонения также малы по абсолютной величине.

Недостатком метода наименьших квадратов является громоздкость вычислений. Поэтому к нему прибегают обычно при обработке

наблюдений высокой точности, когда нужно получить также весьма точные значения параметров.

5.5.3.2. Линейная аппроксимация

Часто при обработке экспериментальных данных оказывается возможным построить линейный аппроксимирующий полином, т.е. описать закон изменения ***x*** линейным уравнением (рис.58)

$$P_1(x)=a_0+a_1 \cdot x\ . \qquad (5.5.28)$$

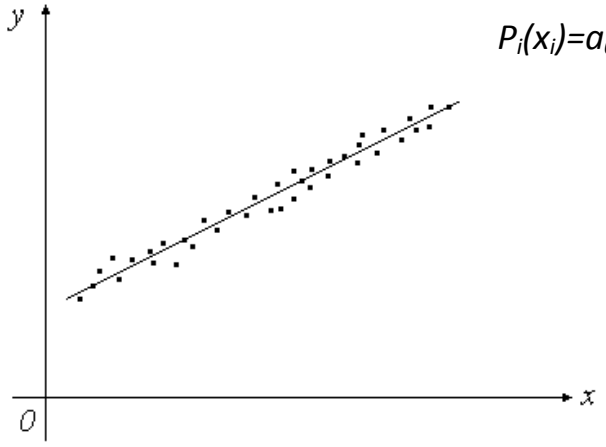


Рис.58

Выведем формулы для расчета неизвестных коэффициентов a_0 и a_1 линейного аппроксимирующего полинома (5.5.28) по методу наименьших квадратов.

$$F=\sum_{i=1}^{n}(y_i-a_0-a_1 \cdot x_i)^2 \to \min\ . \qquad (5.5.29)$$

$$\begin{cases} \dfrac{\partial F}{\partial a_0}=-2 \cdot \sum_{i=1}^{n}(y_i-a_0-a_1 \cdot x_i) \cdot 1=0, \\ \dfrac{\partial F}{\partial a_1}=-2 \cdot \sum_{i=1}^{n}(y_i-a_0-a_1 \cdot x_i) \cdot x_i=0. \end{cases}$$

$$\begin{cases} \sum_{i=1}^{n} y_i-a_0 \cdot n-a_1 \cdot \sum_{i=1}^{n} x_i=0, \\ \sum_{i=1}^{n}(y_i \cdot x_i)-a_0 \cdot \sum_{i=1}^{n} x_i-a_1 \cdot \sum_{i=1}^{n} x_i^2=0. \end{cases}$$

$$\begin{cases} a_0 \cdot n + a_1 \cdot \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i, \\ a_0 \cdot \sum_{i=1}^{n} x_i + a_1 \cdot \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} (x_i \cdot y_i). \end{cases}$$

Решая систему уравнений, выражаем коэффициенты a_0 и a_1.

$$a_0 = \frac{\begin{vmatrix} \sum_{i=1}^{n} y_i & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} (x_i \cdot y_i) & \sum_{i=1}^{n} x_i^2 \end{vmatrix}}{\begin{vmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{vmatrix}} = \frac{\sum_{i=1}^{n} y_i \cdot \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} (x_i \cdot y_i)}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2}$$

$$a_1 = \frac{\begin{vmatrix} n & \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} (x_i \cdot y_i) \end{vmatrix}}{\begin{vmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{vmatrix}} = \frac{n \cdot \sum_{i=1}^{n} (x_i \cdot y_i) - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} y_i}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2}$$

. (5.5.30)

Определитель системы: $D = \begin{vmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{vmatrix} \neq 0.$

Определение коэффициентов a_0 и a_1 возможно, если определитель системы $\neq 0$. Если определитель D=0, то система или не имеет решений (т.е. несовместна), или имеет бесконечно много решений (т.е. система неопределенная).

Пример 5.6. Дана табличная зависимость теплоемкости оксида углерода от температуры

T, K	300	400	500	600	700	800	900	1000
$C_P, \frac{Дж}{моль \cdot K}$	6.97	7.01	7.12	7.28	7.45	7.62	7.79	7.93

Необходимо построить аппроксимирующий полином в виде $y=a_0+a_1 \cdot x$. Для вычисления коэффициентов по функции (5.5.30) составим таблицу (табл.14).

Введем обозначение $x = \frac{T-300}{h}$, где h=100, Ср=у.

Таблица 14

I	T_i	x_i	y_i	x_i^2	$x_i \cdot y_i$
1	300	0	6.97	0	0
2	400	1	7.01	1	7.01
3	500	2	7.12	4	14.24
4	600	3	7.28	9	21.84
5	700	4	7.45	16	29.8
6	800	5	7.62	25	38.1
7	900	6	7.79	36	46.74
8	1000	7	7.93	49	55.51
∑		28	59.17	140	213.24

$$\begin{cases} 8 \cdot a_0 + 28 \cdot a_1 = 59.17, \\ 28 \cdot a_0 + 140 \cdot a_1 = 213.24. \end{cases}$$

$$a_0 = \frac{\begin{vmatrix} 59.17 & 28 \\ 213.24 & 140 \end{vmatrix}}{\begin{vmatrix} 8 & 28 \\ 28 & 140 \end{vmatrix}} = 6.884, \qquad a_1 = \frac{\begin{vmatrix} 8 & 59.17 \\ 28 & 213.24 \end{vmatrix}}{\begin{vmatrix} 8 & 28 \\ 28 & 140 \end{vmatrix}} = 0.146.$$

$$y=6.884+0.146 \cdot x.$$

$$Cp = 6.884 + \frac{T-300}{100} \cdot 0.146 = 6.445 + 0.146 \cdot 10^{-2} T.$$

$$C_P = 6.445 + 0.146 \cdot 10^{-2} \cdot T$$

Сделаем проверку, насколько хорошо полученный полином аппроксимирует (описывает) эксперимент (табл.15).

Таблица 15

i	T_i	$C_{Pi}^{э}$	C_{Pi}^{p}	$/C_{Pi}^{p} - C_{Pi}^{э}/$
1	300	6.97	6.9	0.07
2	400	7.01	7.03	0.02
3	500	7.12	7.17	0.05
4	600	7.28	7.32	0.04
5	700	7.45	7.46	0.01
6	800	7.62	7.62	0.00
7	900	7.79	7.76	0.03
8	1000	7.93	7.91	0.02

Разность между исходными данными и результатами расчета по полученному выражению определяет погрешность аппроксимации. Если погрешность велика, то выбирают другой вид аппроксимирующего полинома.

Блок-схема алгоритма линейной аппроксимации приведена на рис.59.

Приняты следующие обозначения:

$$S1 = \sum_{i=1}^{n} X_i; \quad S2 = \sum_{i=1}^{n} Y_i; \quad S3 = \sum_{i=1}^{n} X_i \cdot Y_i; \quad S4 = \sum_{i=1}^{n} X_i^2$$

5.5.3.3. Параболическая аппроксимация

В том случае, если экспериментальные данные не удается с достаточной степенью точности аппроксимировать линейным полиномом, применяют аппроксимацию 2-го и большего порядков. Такая

аппроксимация называется нелинейной. Рассмотрим случай многочлена 2-ой степени.

$$P_2(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2. \qquad (5.5.31)$$

Как и в случае линейной аппроксимации, коэффициенты a_j определяются по методу наименьших квадратов. Запишем квадратичное отклонение.

$$F = \sum_{i=1}^{n} \left(y_i - a_0 - a_1 \cdot x_i - a_2 \cdot x_i^2 \right)^2 \to \min. \qquad (5.5.32)$$

Приравняем к нулю частные производные:

$$\begin{aligned} \frac{\partial F}{\partial a_0} &= -2 \cdot \sum_{i=1}^{n} \left(y_i - a_0 - a_1 \cdot x_i - a_2 \cdot x_i^2 \right) \cdot 1 = 0, \\ \frac{\partial F}{\partial a_1} &= -2 \cdot \sum_{i=1}^{n} \left(y_i - a_0 - a_1 \cdot x_i - a_2 \cdot x_i^2 \right) \cdot x_i = 0, \\ \frac{\partial F}{\partial a_2} &= -2 \cdot \sum_{i=1}^{n} \left(y_i - a_0 - a_1 \cdot x_i - a_2 \cdot x_i^2 \right) \cdot x_i^2 = 0. \end{aligned} \qquad (5.5.33)$$

Выполнив преобразования, получим систему линейных уравнений с тремя неизвестными (a_0, a_1,a_2).

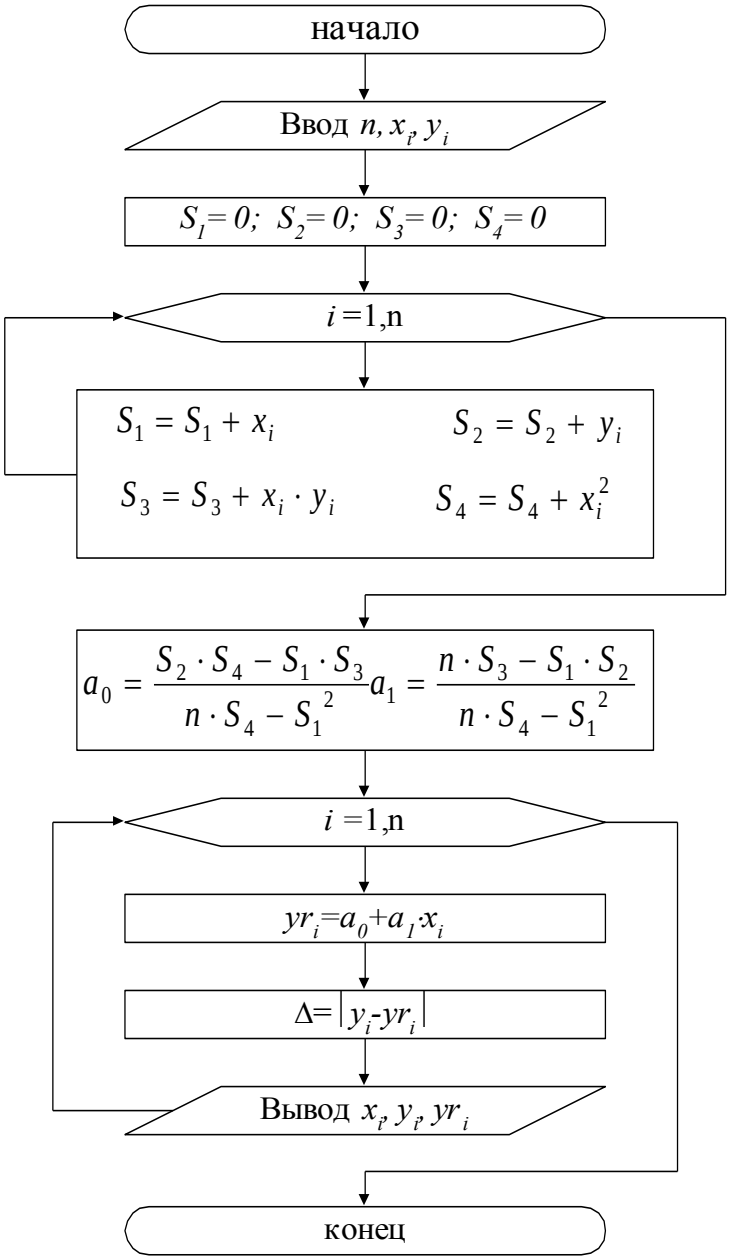


Рис. 59. Блок - схема линейной аппроксимации

$$\begin{cases} a_0 \cdot n + a_1 \cdot \sum_{i=1}^{n} x_i + a_2 \cdot \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} y_i, \\ a_0 \cdot \sum_{i-1}^{n} x_i + a_1 \cdot \sum_{i=1}^{n} x_i^2 + a_2 \cdot \sum_{i=1}^{n} x_i^3 = \sum_{i=1}^{n} (x_i \cdot y_i), \\ a_0 \cdot \sum_{i=1}^{n} x_i^2 + a_1 \cdot \sum_{i=1}^{n} x_i^3 + a_2 \cdot \sum_{i=1}^{n} x_i^4 = \sum_{i=1}^{n} (x_i^2 \cdot y_i). \end{cases} \quad (5.5.34)$$

Введем обозначения:

$$S_1 = \sum_{i=1}^{n} x_i ; \qquad S_2 = \sum_{i=1}^{n} x_i^2 ; \qquad S_3 = \sum_{i=1}^{n} x_i^3 ; \qquad S_4 = \sum_{i=1}^{n} x_i^4 ;$$

5.5.35)

$$S_5 = \sum_{i=1}^{n} y_i ; \qquad S_6 = \sum_{i=1}^{n} (x_i \cdot y_i); \qquad S_7 = \sum_{i=1}^{n} (x_i^2 \cdot y_i).$$

С учетом принятых обозначений система (5.5.34) будет иметь следующий вид:

$$\begin{cases} a_0 \cdot n + a_1 \cdot S_1 + a_2 \cdot S_2 = S_5, \\ a_0 \cdot S_1 + a_1 \cdot S_2 + a_2 \cdot S_3 = S_6, \\ a_0 \cdot S_2 + a_1 \cdot S_3 + a_2 \cdot S_4 = S_7. \end{cases} \quad (5.5.36)$$

Определим неизвестные коэффициенты a_0, a_1, a_2.

$$a_0 = \frac{\begin{vmatrix} S_5 & S_1 & S_2 \\ S_6 & S_2 & S_3 \\ S_7 & S_3 & S_4 \end{vmatrix}}{\begin{vmatrix} n & S_1 & S_2 \\ S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \end{vmatrix}}, \quad a_1 = \frac{\begin{vmatrix} n & S_5 & S_2 \\ S_1 & S_6 & S_3 \\ S_2 & S_7 & S_4 \end{vmatrix}}{\begin{vmatrix} n & S_1 & S_2 \\ S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \end{vmatrix}}, \quad a_2 = \frac{\begin{vmatrix} n & S_1 & S_5 \\ S_1 & S_2 & S_6 \\ S_2 & S_3 & S_7 \end{vmatrix}}{\begin{vmatrix} n & S_1 & S_2 \\ S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \end{vmatrix}}$$

. (5.5.37)

Необходимое условие: определитель системы

$$D=\begin{vmatrix} n & S_1 & S_2 \\ S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \end{vmatrix} \neq 0.$$

Пример 5.5.7. Дана табличная зависимость y от x (табл.16). Необходимо построить аппроксимирующий полином в виде $y=a_0+a_1 \cdot x+a_2 \cdot x^2$.

Таблица 16

x_i	y_i	x_i^2	x_i^3	x_i^4	$x_i \cdot y_i$	$x_i^2 \cdot y_i$
0	1	0	0	0	0	0
1	3	1	1	1	3	3
2	4	4	8	16	8	16
3	5	9	27	81	15	45
S_1=6	S_5=13	S_2=14	S_3=36	S_4=98	S_6=26	S_7=64

$$\begin{cases} 4 \cdot a_0 + 6 \cdot a_1 + 14 \cdot a_2 = 13, \\ 6 \cdot a_0 + 14 \cdot a_1 + 36 \cdot a_2 = 26, \\ 14 \cdot a_0 + 36 \cdot a_1 + 98 \cdot a_2 = 64. \end{cases}$$

Эта система линейных алгебраических уравнений относительно неизвестных a_0, a_1, a_2. Определитель системы $\neq 0$, т.е. существует единственное решение.

$$a_0 = \frac{\begin{vmatrix} 13 & 6 & 14 \\ 26 & 14 & 36 \\ 64 & 36 & 98 \end{vmatrix}}{\begin{vmatrix} 4 & 6 & 14 \\ 6 & 14 & 36 \\ 14 & 36 & 98 \end{vmatrix}} = 1.05,\ a_1 = \frac{\begin{vmatrix} 4 & 13 & 14 \\ 6 & 26 & 36 \\ 14 & 64 & 98 \end{vmatrix}}{\begin{vmatrix} 4 & 6 & 14 \\ 6 & 14 & 36 \\ 14 & 36 & 98 \end{vmatrix}} = 2.05,\ a_2 = \frac{\begin{vmatrix} 4 & 6 & 13 \\ 6 & 14 & 26 \\ 14 & 36 & 64 \end{vmatrix}}{\begin{vmatrix} 4 & 6 & 14 \\ 6 & 14 & 36 \\ 14 & 36 & 98 \end{vmatrix}} = -0.25.$$

Запишем аппроксимирующее уравнение с учетом найденных коэффициентов: $y=a_0+a_1 \cdot x+a_2 \cdot x^2=1.05+2.05 \cdot x-0.25 \cdot x^2$. Проведем оценку правильности полученного уравнения (табл.17, рис.60).

Таблица 17

i	x_i	$y_i^э$	y_i^p	$\left(y_i^э - y_i^p\right)$
1	0	1	1.05	-0.05
2	1	3	2.85	0.15
3	2	4	4.15	-0.05
4	3	5	4.95	0.05

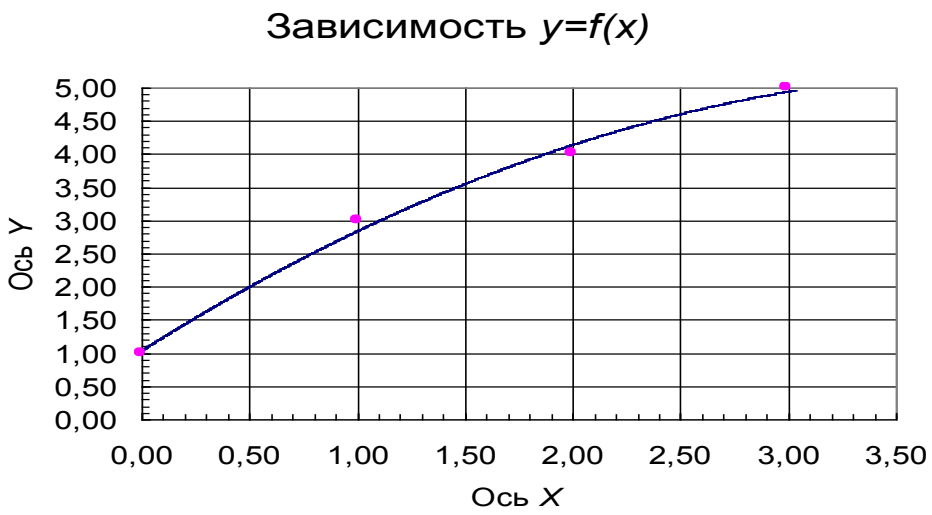


Рис.60

Пример 5.5.8. Дана зависимость теплоемкости циклопропана от температуры.

T	298	300	400	500	600	700	800	900	1000
Cp	13.37	13.44	18.31	22.65	26.15	29.02	31.45	33.57	35.39

Аппроксимировать экспериментальные данные полиномом второго порядка (5.5.31). Коэффициенты a_0,a_1,a_2 определяем на ПЭВМ по формулам (5.5.37). Блок-схема параболической аппроксимации 2-ого порядка приведена на рис.61.

В результате решения получаем следующее уравнение для расчета теплоемкости циклопропана:

$$Cp(T) = -3.766 + 0.0657T - 0.00003T^2 .$$

Результаты расчета приведены в табл.18.

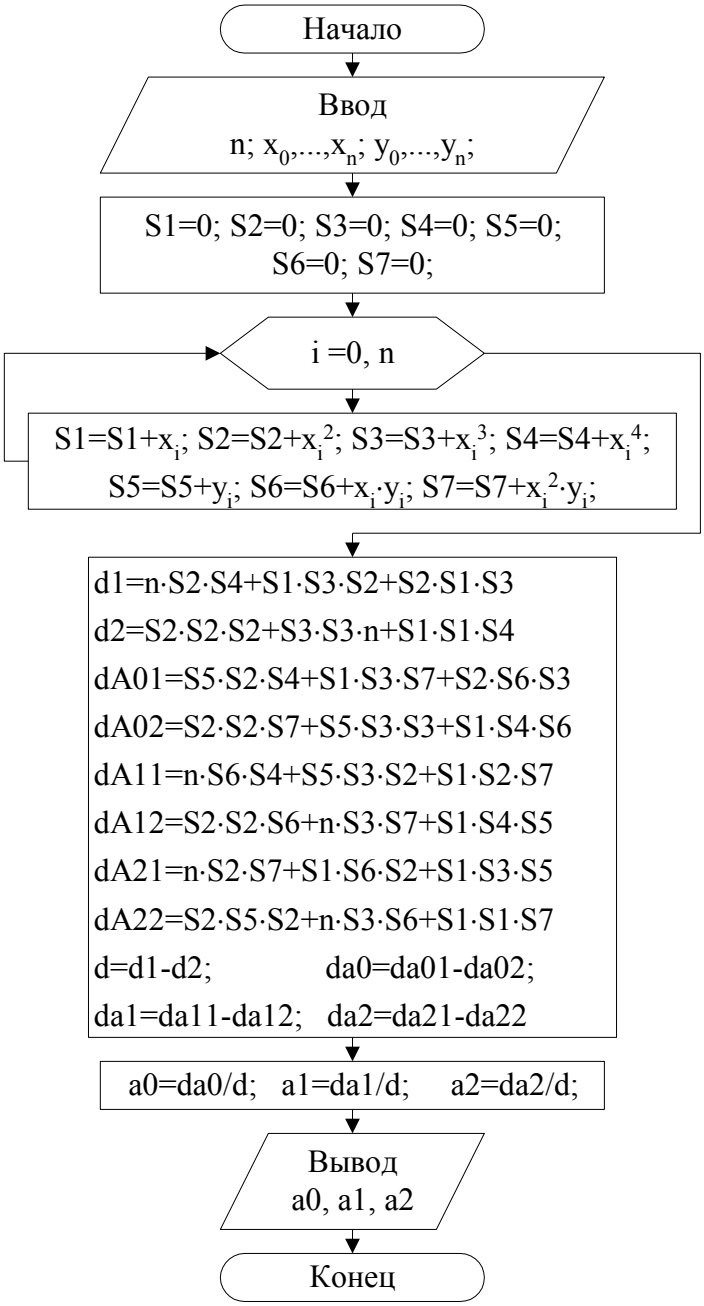


Рис.61. Блок – схема параболической аппроксимации

Таблица 18

T	$Cp^э$	Cp^p	$\left\| Cp_i^p - Cp_i^э \right\|$
298	13.37	13.44	0.07
300	13.44	13.54	0.01
400	18.31	18.24	0.07
500	22.65	22.40	0.25
600	26.15	26.03	0.12
700	29.02	29.12	0.10
800	31.45	31.68	0.23
900	33.57	33.70	0.13
1000	35.39	35.19	0.20

5.5.3.4. Аппроксимация в виде показательной и степенной функции

а) показательная (экспоненциальная) функция

Часто при обработке экспериментальных данных возникает необходимость воспользоваться зависимостью вида:

$$y = a \cdot e^{b \cdot x}, \tag{5.5.38}$$

где a, b – неизвестные коэффициенты.

Представим уравнение (5.5.38) в виде:

$$\ln(y) = \ln(a) + b \cdot x.$$

Введем обозначения: $Y=\ln(y)$; $A_0=\ln(a)$; $A_1=b$.

$$Y=A_0+A_1 \cdot x. \tag{5.5.39}$$

Последнее выражение представляет собой линейный многочлен 1-ой степени. Решаем систему по методу наименьших квадратов относительно A_0 и A_1.

$$F = \sum_{i=1}^{n} \left(Y_i - \left(A_0 + A_1 \cdot x_i\right)\right)^2 \to \min.$$

Последовательность вывода формул вычисления коэффициентов A_0 и A аналогична последовательности для случая линейной аппроксимации. (разд.5.5.3.2)

$$A_0 = \frac{\begin{vmatrix} \sum_{i=1}^{n} Y_i & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} (x_i \cdot Y_i) & \sum_{i=1}^{n} x_i^2 \end{vmatrix}}{\begin{vmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{vmatrix}}, \qquad A_1 = \frac{\begin{vmatrix} n & \sum_{i=1}^{n} Y_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} (x_i \cdot Y_i) \end{vmatrix}}{\begin{vmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{vmatrix}}. \quad (5.5.40)$$

После определения коэффициентов A_0 и A_1 вернемся к принятым ранее обозначениям:

$$a = e^{A_0}, \quad b = A_1, \qquad y_i = e^{Y_i}.$$

Пример 5.5.9. В табл. 19 приведены данные по конверсии пропана в зависимости от времени работы катализатора.

Таблица 19

t, час	5	10	15	20	25	30
$\mathbf{X_{C_3H_8}}$,%	66.1	60.0	53.5	48.4	44.3	42.3

Аппроксимировать экспериментальные данные экспоненциальной зависимостью:

$$X_{C_3H_8} = a \cdot e^{-\alpha t},$$

где α - показатель активности катализатора.

Представим уравнение в виде:

$$y = a \cdot e^{-bx}$$

и прологарифмируем

$$\ln y = \ln a - b \cdot x.$$

Обозначим

$$Y = \ln y; \quad a_0 = \ln a; \quad a_1 = b, \quad \text{получим} \quad Y = a_0 - a_1 x.$$

Таблица 20

x_i	$y_i^э$	$\ln y_i$	x_i^2	$x_i \cdot \ln y_i$	$y_i^р$	$\lvert y_i^э - y_i^р \rvert$
5	66.1	4.1912	25	20.95558	65.2	0.9
10	60.0	4.0943	100	40.9435	59.5	0.5
15	53.5	3.9797	225	59.6952	54.2	0.7
20	48.4	3.8795	400	77.5900	49.3	0.9
25	44.3	3.7910	625	94.7746	45.1	0.8
30	42.3	4.7448	900	112.3436	41.4	0.9
105		23.6805	2275	406.3027		

Коэффициенты a_0 и a_1 определяем по формулам (5.5.40).

$$a_0 = 4{,}271; \quad a_1 = -0{,}0185.$$

Вернемся к прежним обозначениям:

$$a = e^{a0} = \exp(4{,}271);$$

$$a = 71.59,$$

$$b = a_1 = -0.0185.$$

Получим вид аппроксимирующего полинома

$$y = 71.59 \cdot e^{-0.0185x},$$

или

$$x_{C_3H_8} = 71.59 \cdot e^{-0.0185t}.$$

В табл.20 приведены результаты расчета y_i^{p} по полученному уравнению и относительная погрешность. Блок-схема аппроксимации показательной функцией приведена на рис.62.

b) степенная функция.

Степенная функция имеет вид:

$$y = a \cdot x^b. \tag{5.5.41}$$

Логарифмируя последнее уравнение получим:

$$\lg(y_i) = \lg(a) + b \cdot \lg(x_i). \tag{5.5.42}$$

Введем обозначения: $Y=\lg(y)$; $A_0=\lg(a_0)$; $A_1=b$; $X=\lg(x)$.

Используя метод наименьших квадратов, найдем неизвестные коэффициенты A_0 и A_1.

$$F = \sum_{i=1}^{n} \left(Y_i - \left(A_0 + A_1 \cdot X_i\right)\right)^2 \to \min$$

Определим неизвестные коэффициенты A_0 и A_1:

$$A_0 = \frac{\begin{vmatrix} \sum_{i=1}^{n} Y_i & \sum_{i=1}^{n} X_i \\ \sum_{i=1}^{n} \left(X_i \cdot Y_i\right) & \sum_{i=1}^{n} X_i^2 \end{vmatrix}}{\begin{vmatrix} n & \sum_{i=1}^{n} X_i \\ \sum_{i=1}^{n} X_i & \sum_{i=1}^{n} X_i^2 \end{vmatrix}}, \quad A_1 = \frac{\begin{vmatrix} n & \sum_{i=1}^{n} Y_i \\ \sum_{i=1}^{n} X_i & \sum_{i=1}^{n} \left(X_i \cdot Y_i\right) \end{vmatrix}}{\begin{vmatrix} n & \sum_{i=1}^{n} X_i \\ \sum_{i=1}^{n} X_i & \sum_{i=1}^{n} X_i^2 \end{vmatrix}}. \tag{5.5.43}$$

После определения коэффициентов A_0 и A_1 вернемся к принятым ранее обозначениям: $a_0 = 10^{A_0}$, $b = A_1$, $y_i = 10^{Y_i}$, $x_i = 10^{X_i}$.

Блок-схема степенной аппроксимации приведена на рис.63.

Недостатком метода наименьших квадратов является громоздкость вычислений. Поэтому к нему прибегают обычно при обработке наблюдений высокой точности, когда нужно получить также весьма точные значения параметров.

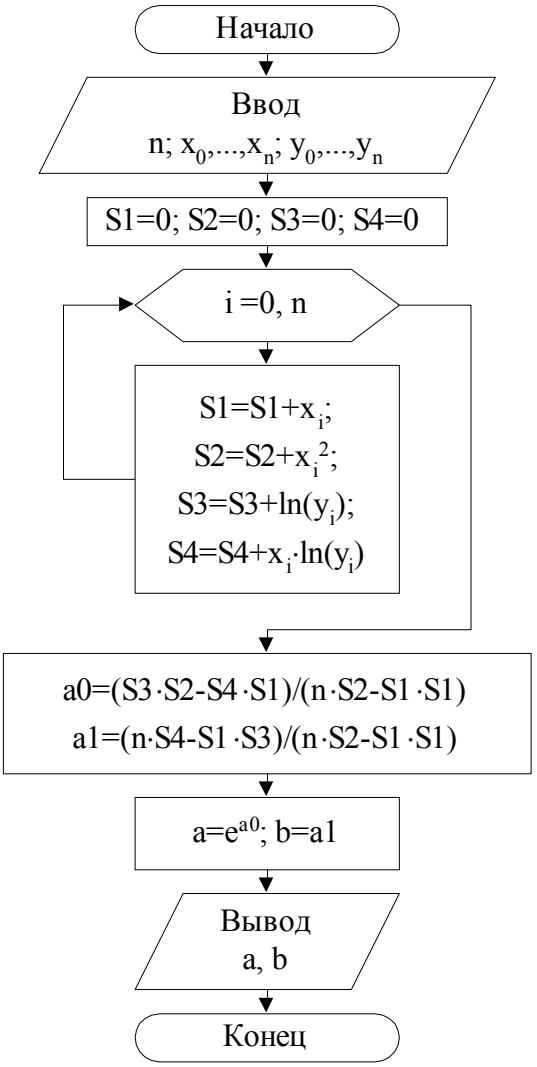


Рис. 62. Блок - схема аппроксимации показательной функцией

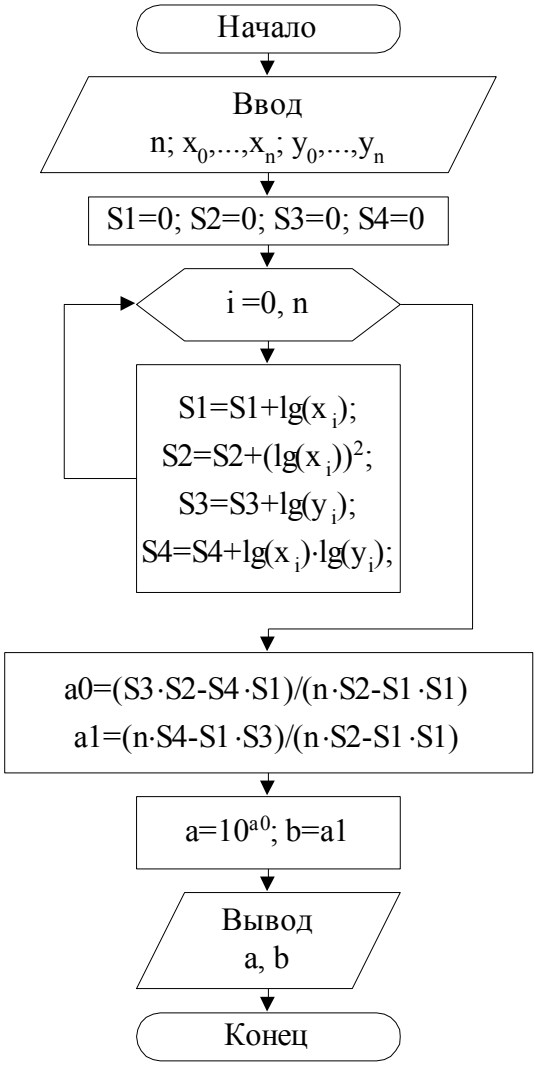


Рис. 63. Блок - схема степенной аппроксимации

5.6. Приближенное дифференцирование
5.6.1. Постановка задачи

При решении практических задач часто нужно найти производную указанных порядков от функции $y = f(x)$, заданной таблично. Возможно

также, что в силу сложности аналитического выражения функции $f(x)$ непосредственное дифференцирование ее затруднительно. В этих случаях обычно прибегают к *приближенному дифференцированию*.

Для вывода формул приближенного дифференцирования заменяют данную функцию $f(x)$ на интересующем отрезке $[a,b]$ интерполирующей функцией $P(x)$ (чаще всего полиномом), а затем полагают:

$$f'(x) = P'(x) \qquad (5.6.1)$$

при

$$a \le x \le b.$$

Аналогично поступают при нахождении производных высших порядков функции $f(x)$.

Если для интерполирующей функции $P(x)$ известна погрешность

$$R(x) = f(x) - P(x),$$

то погрешность производной $P'(x)$ выражается формулой

$$r(x) = f'(x) - P'(x) = R'(x), \qquad (5.6.2)$$

т.е. погрешность производной интерполирующей функции равна производной от погрешности этой функции. То же самое справедливо и для производных высших порядков.

Следует отметить, что, вообще говоря, приближенное дифференцирование представляет собой операцию менее точную, чем интерполирование. Действительно, близость друг к другу ординат двух кривых

$$y = f(x) \quad \text{и} \quad Y = P(x)$$

на отрезке $[a,b]$ еще не гарантирует близости на этом отрезке их производных $f'(x)$ и $P'(x)$, т.е. малого расхождения угловых коэффициентов касательных к рассматриваемым кривым при одинаковых значениях аргумента. (см. рис. 64).

5.6.2. Формулы, основанные на первой интерполяционной формуле Ньютона

Пусть имеем функцию $y(x)$, заданную в равноотстоящих точках x_i ($i = 0, 1, 2, \dots, n$) отрезка $[a,b]$ с помощью значений $y_i = f(x_i)$. Для нахождения на $[a,b]$ производных $y' = f'(x)$, $y'' = f''(x)$ и т.д. функцию y приближенно заменим интерполяционным полиномом Ньютона, построенным для системы узлов $x_0, x_1, \dots, x_k$ ($k \le n$).

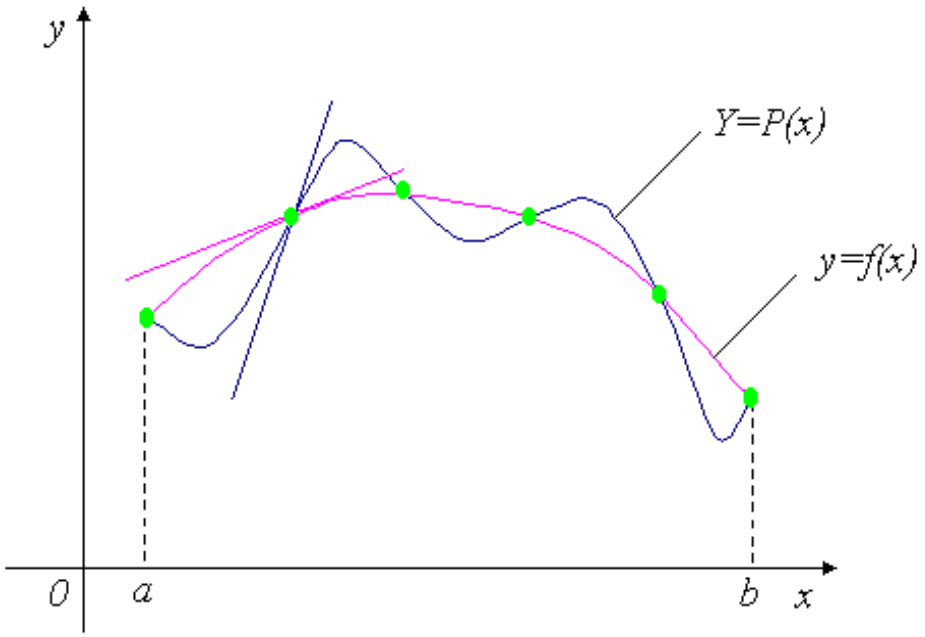


Рис.64

Имеем:

$$y(x) = y_0 + q\Delta y_0 + \frac{q(q-1)}{2!}\Delta^2 y_0 + \frac{q(q-1)(q-2)}{3!}\Delta^3 y_0 +$$
$$+ \frac{q(q-1)(q-2)(q-3)}{4!}\Delta^4 y_0 + \frac{q(q-1)(q-2)(q-3)(q-4)}{5!}\Delta^5 y_0 + \ldots ,$$

где

$$q = \frac{x - x_0}{h} \qquad \text{и} \qquad h = x_{i+1} - x_i \qquad (i = 0, 1, \ldots).$$

Производя перемножение биномов, получим:

$$y(x) = y_0 + q\Delta y_0 + \frac{q^2 - q}{2}\Delta^2 y_0 + \frac{q^3 - 3q^2 + 2q}{6}\Delta^3 y_0 +$$
$$+ \frac{q^4 - 6q^3 + 11q^2 - 6q}{24}\Delta^4 y_0 + \qquad (5.6.3)$$
$$+ \frac{q^5 - 10q^4 + 35q^3 - 50q^2 + 24q}{120}\Delta^5 y_0 + \ldots$$

Так как

$$\frac{dy}{dx} = \frac{dy}{dq}\cdot\frac{dq}{dx} = \frac{1}{h}\cdot\frac{dy}{dq},$$

то

$$y'(x)=\frac{1}{h}\left[\begin{array}{l}\Delta y_0+\dfrac{2q-1}{2}\Delta^2 y_0+\dfrac{3q^2-6q+2}{6}\Delta^3 y_0+\\ +\dfrac{2q^3-9q^2+11q-3}{12}\Delta^3 y_0+\ldots\end{array}\right]. \qquad (5.6.4)$$

Аналогично, так как

$$y''(x)=\frac{d(y')}{dx}=\frac{d(y')}{dq}\cdot\frac{dq}{dx}=\frac{d(y')}{dq}\cdot\frac{1}{h},$$

то

$$y''(x)=\frac{1}{h^2}[\Delta^2 y_0+(q-1)\Delta^3 y_0+\frac{6q^2-18q+11}{12}\Delta^4 y_0+ \\ +\frac{2q^3-12q^2+21q-10}{12}\Delta^5 y_0+\ldots]. \qquad (5.6.5)$$

Таким же способом, в случае надобности, можно вычислить и производные функции $y(x)$ любого порядка. Заметим, что при нахождении производных $y'(x)$, $y''(x)$, ... в фиксированной точке x в качестве x_0 следует выбирать ближайшее табличное значение аргумента. Отметим, что можно вывести также формулы приближенного дифференцирования, исходя из второй интерполяционной формулы Ньютона.

Пример 5.6.1. Найти $y'(50)$ функции $y = \lg x$, заданной таблично.

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$
50	1.6990	0.0414	-0.0036	0.0005
55	1.7404	0.0378	-0.0031	
60	1.7782	0.0347		
65	1.8129			

Решение. Здесь h=5. Дополняем таблицу столбцами конечных разностей. Так как x= 50 и $q = (x - x_0)/h = 0$, то $y'(50)$ = (0.0414 + 0.0018 + 0.0002) = 0.0087.

Для оценки точности найденного значения заметим, что, так как табулированная выше функция есть $y = \lg x$, то

$$y'(x)=\frac{1}{x\ln 10}.$$

Следовательно, $y'(50)$ = 1/50 · 1/ln(10) = 0.0087. Таким образом, результаты совпадают с точностью до четвертого десятичного знака.

5.7. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

Ряд технологических задач требует увязки в математическое описание всей информации о процессе. Например, для математических моделей химико-технологических процессов одними из основных параметров, характеризующих процессы, являются концентрации реагирующих веществ, температура процесса и др. Как правило, большинство балансовых уравнений в химической технологии представлены системой интегральных и дифференциальных уравнений, в результате решения которых могут быть получены зависимости, характеризующие протекание процесса. Интегральные уравнения встречаются при описании гетерогенной кинетики (поверхности раздела фаз), в теории активированного комплекса, при описании процессов в химических реакторах и т.д.

Часто на практике не удается вычислить интеграл аналитическим путем. В этих случаях применяют приближенные методы численного интегрирования.

Постановка задачи

Вычислить определенный интеграл

$$I = \int_a^b f(x)dx \qquad (5.7.1)$$

при условии, что **a** и **b** конечны и F(x) является непрерывной функцией **x** на всем интервале $x \in [a,b]$. Во многих случаях, когда подынтегральная функция задана в аналитическом виде, интеграл от этой функции в пределах от **a** до **b** может быть вычислен по формуле *Ньютона-Лейбница*:

$$\int_a^b f(x)dx = F(x)\Big|_a^b = F(b) - F(a). \qquad (5.7.2)$$

Однако этой формулой часто нельзя воспользоваться по следующим причинам:

- первообразная функция f(x) слишком сложна и ее нельзя выразить в элементарных функциях;
- функция f(x) задана в виде таблицы, что особенно часто встречается в задачах химической технологии при обработке экспериментальных данных.

В этих случаях используются методы численного интегрирования.

Задача численного интегрирования состоит в нахождении приближенного значения интеграла (5.7.1) по заданным или вычисленным значениям.

Общий подход к решению задачи будет следующим. Определенный интеграл I представляет собой площадь, ограниченную кривой f(x), осью x и прямыми x=**a** и x=**b**. Необходимо вычислить интеграл, разбивая интервал [a,b] на множество меньших интервалов, находя приблизительно площадь каждой полоски и суммируя их.

В зависимости от способа вычисления подынтегральной суммы существуют различные методы численного интегрирования (методы прямоугольников, трапеций, парабол, сплайнов и др.).

5.7.1. Метод прямоугольников

Простейшим методом численного интегрирования является *метод прямоугольников*. Он непосредственно использует замену определенного интеграла интегральной суммой:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} f(\xi_i)\cdot \Delta x_i; \qquad \xi_i \in [x_{i-1}, x_i]. \qquad (5.7.3)$$

Разобьём интервал интегрирования [a,b] на n равных частей. Обозначим $\Delta x_i = h$ шаг разбиения. Формула прямоугольника применяется к каждому отрезку. В качестве точек ξ_i выбираются левые ($\xi_i = x_{i-1}$) или правые ($\xi_i = x_i$) границы элементарных отрезков (рис.65).

Соответственно, для этих двух случаев можно записать формулы метода прямоугольников:

$$\int_a^b f(x)dx = h_1 \cdot f(x_0) + h_2 \cdot f(x_1) + \ldots + h_n \cdot f(x_{n-1}); \qquad (5.7.4)$$

$$\int_a^b f(x)dx = h_1 \cdot f(x_1) + h_2 \cdot f(x_2) + \ldots + h_n \cdot f(x_n). \qquad (5.7.5)$$

Более точным является вид формулы прямоугольников, использующий значения функции в средних точках элементарных отрезков: точка $\overline{x_i}$. Таким образом, площадь криволинейной трапеции заменяется суммой прямоугольников с основанием h и высотами, равными значениям функции f(x) в середине оснований $f(\overline{x_i})$ (рис.66).

Получим формулу:

$$\int_a^b f(x)dx = \frac{b-a}{n} \cdot \sum_{i=1}^{n} f\left(\overline{x_i}\right), \qquad \text{где} \quad \frac{b-a}{n} = h, \qquad (5.7.6)$$

или

$$\int_a^b f(x)dx \approx h \cdot \sum_{i=1}^{n} f\left(\frac{x_{i+1} + x_i}{2}\right). \qquad (5.7.7)$$

На рис.67. приведена блок-схема метода прямоугольников, в Приложении – программа с графическим выводом интегральной кривой.

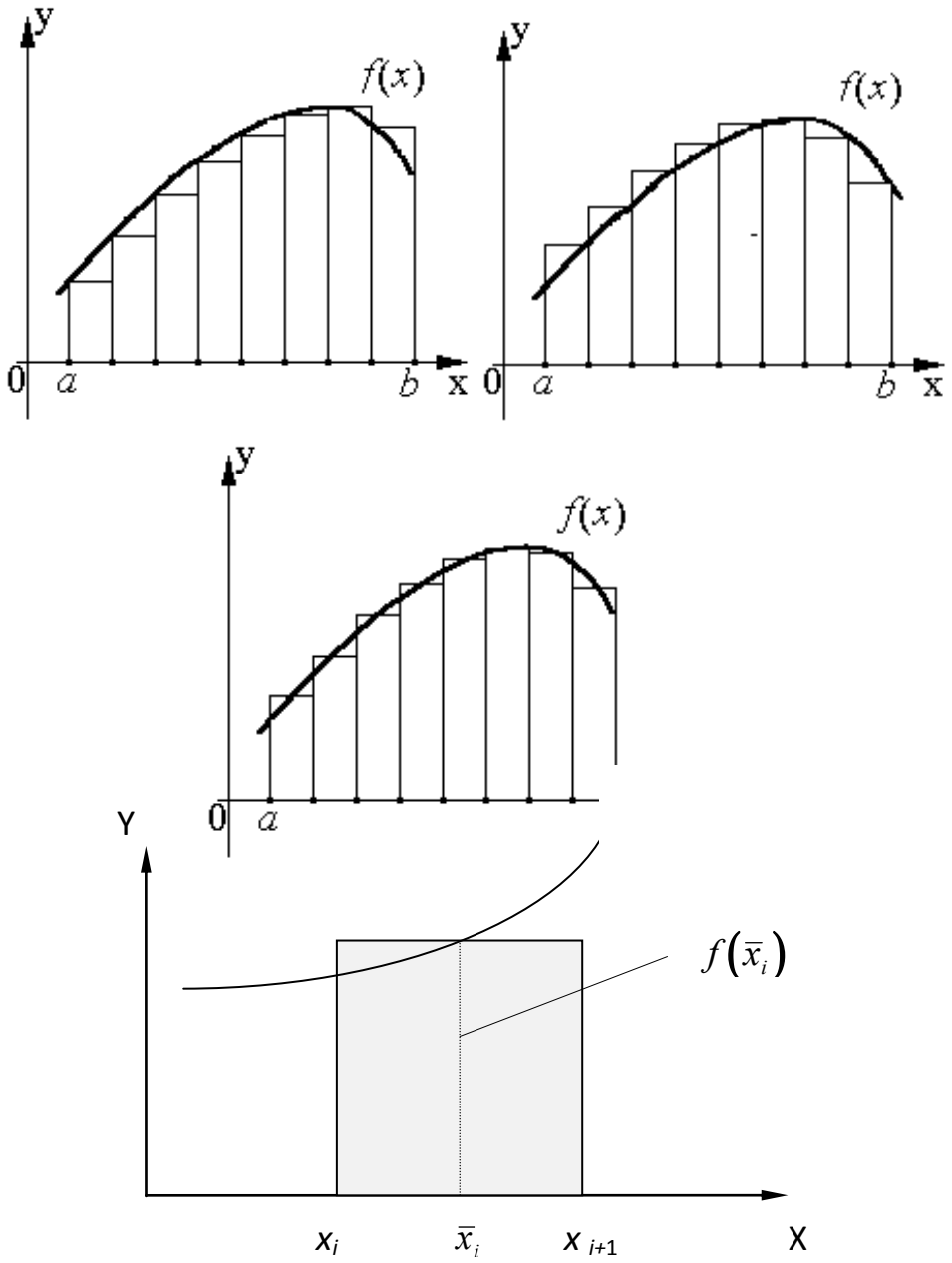


Рис.66

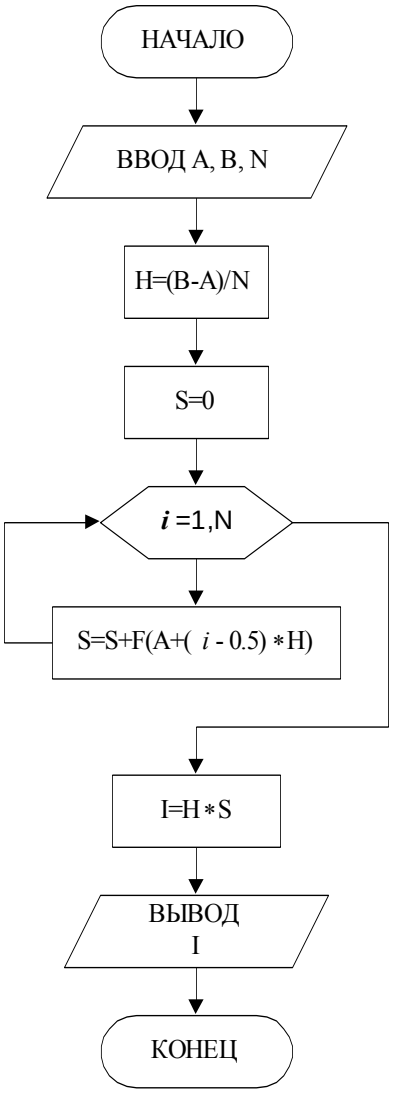


Рис.67. Блок-схема метода прямоугольников

5.7.2. Метод трапеций

Метод трапеций использует линейную интерполяцию, т.е. график функции y=f(x) представляется в виде ломаной, соединяющей точки (x_i, y_i). В этом случае площадь всей криволинейной трапеции складывается из площадей элементарных прямоугольных трапеций (рис.68, 69)

Площадь каждой такой трапеции определяется по формуле:

$$S_i = \frac{y_{i-1} + y_i}{2} \cdot h_i, \qquad \text{i=1,2,...,n} \quad . \tag{5.7.8}$$

$h = \dfrac{b-a}{n}$, где n - число интервалов разбиения.

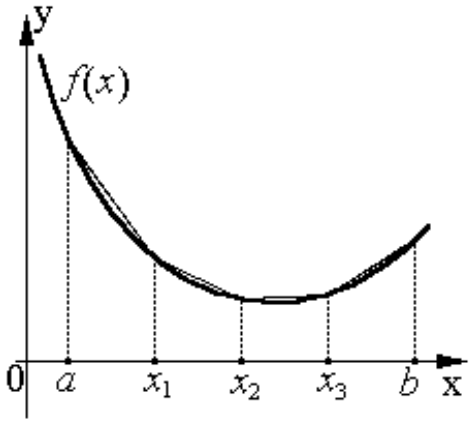


Рис.68

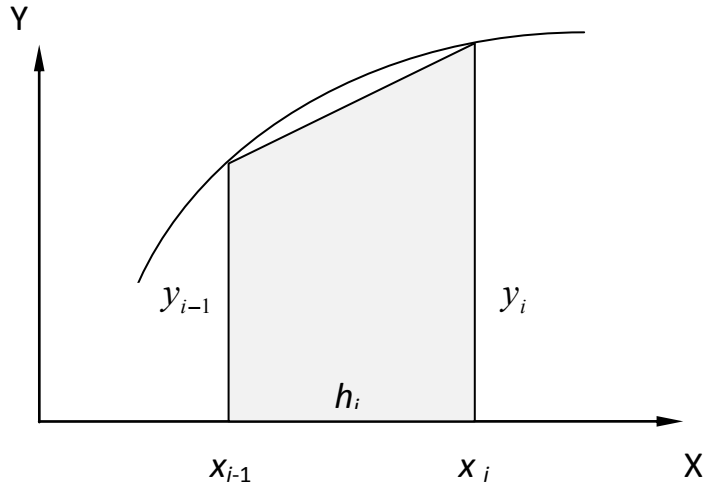


Рис.69

Складывая все эти равенства, получим *формулу трапеций* для численного интегрирования:

$$\int_a^b f(x)dx = \sum_{i=1}^{n} s_i = \frac{h}{2} \cdot \sum_{i=1}^{n} (y_{i-1} + y_i), \qquad (5.7.9)$$

или

$$\int_a^b f(x)dx = h \cdot \sum_{i=1}^{n} \frac{f(x_{i-1}) + f(x_i)}{2}. \qquad (5.7.10)$$

Формулы (5.7.9) и (5.7.10) можно представить в виде:

$$\int_a^b f(x)dx = h \cdot (\frac{y_0 + y_1}{2} + \frac{y_1 + y_2}{2} + \ldots + \frac{y_{n-1} + y_n}{2}), \qquad (5.7.11)$$

$$\int_a^b f(x)dx = h \cdot \left(\frac{y_0 + y_n}{2} + \sum_{i=1}^{n-1} y_i \right). \qquad (5.7.12)$$

Блок - схема алгоритма метода трапеций приведена на рис.70.

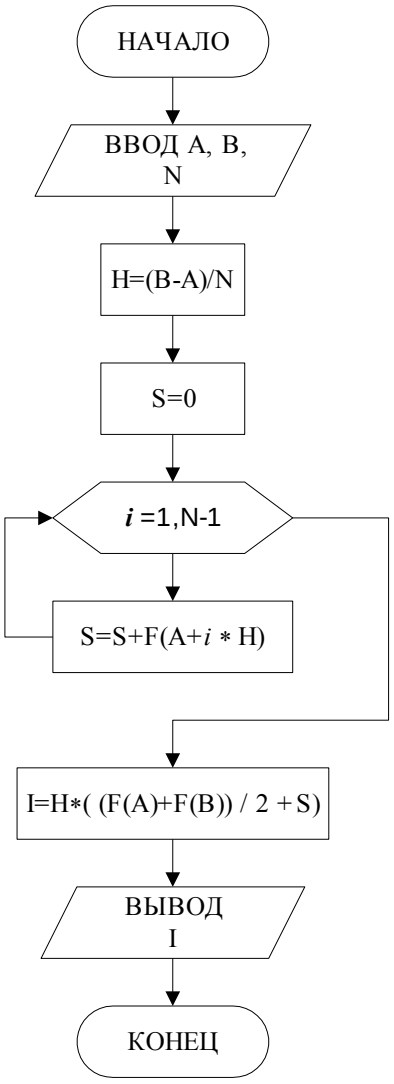


Рис.70

5.7.3. Метод парабол (формула Симпсона)

Этот метод более точный по сравнению с методами прямоугольников и трапеций.

В основе формулы Симпсона лежит квадратичная интерполяция подынтегральной функции на отрезке [a,b] по трем равноотстоящим узлам.

Разобьем интервал интегрирования [a, b] на четное число **n** равных отрезков с шагом **h**.

Примем: $x_0=a$, $x_1=x_0+h$, ... , $x_n=x_0+nh=b$.

Значения функций в точках обозначим соответственно:

$$y_0=f(a);\quad y_1=f(x_1);\quad y_2=f(x_2);\quad \ldots\ ;\quad y_n=f(b).$$

На каждом отрезке $[x_0,x_2]$, $[x_2,x_4]$, ..., $[x_{i-1},x_{i+1}]$ подынтегральную функцию f(x) заменим интерполяционным многочленом второй степени.

$$f(x) \approx P_i(x) = a_i x^2 + b_i x + c_i, \qquad \text{где } x_{i-1} \le x \le x_{i+1}. \qquad (5.7.13)$$

В качестве $P_i(x)$ можно принять интерполяционный многочлен Лагранжа второй степени, проходящий через концы каждых трех ординат:

$$y_0, y_1, y_2\ ; \quad y_2, y_3, y_4\ ; \quad y_4, y_5, y_6; \quad \ldots\ ; \quad y_{n-2}, y_{n-1}, y_n.$$

Формула Лагранжа для интервала $[x_{i-1},x_{i+1}]$:

$$P_i = \frac{(x-x_i)(x-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})}\cdot y_{i-1} + \frac{(x-x_{i-1})(x-x_{i+1})}{(x_i-x_{i-1})(x_i-x_{i+1})}\cdot y_i + \\ + \frac{(x-x_{i-1})(x-x_i)}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)}\cdot y_{i+1} \quad .$$

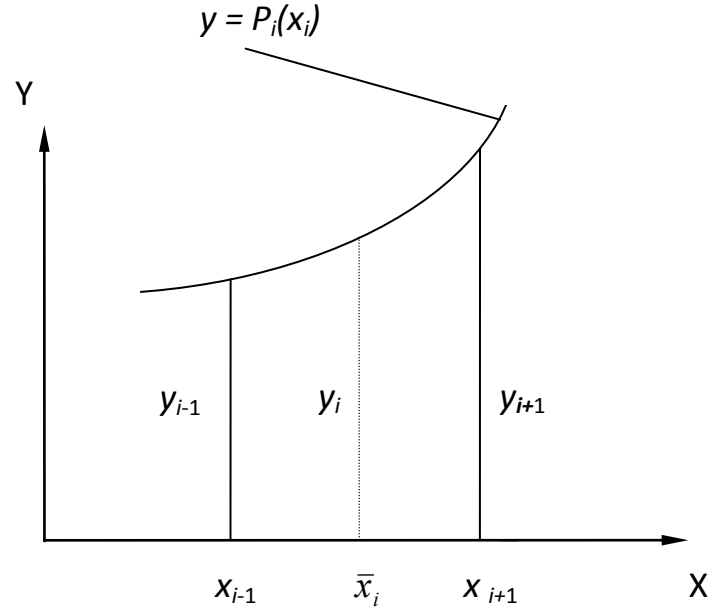


Рис. 71

Элементарная площадь s_i (рис.71) может быть вычислена с помощью определенного интеграла. Учитывая, что $x_i-x_{i-1}=x_{i+1}-x_i=h$ и, проведя вычисления, получим для каждого элементарного участка:

$$s_i = \int_{X_{i-1}}^{X_{i+1}} P_i(x)dx = \frac{h}{3}\cdot(y_{i-1}+4y_i+y_{i+1}) \ . \qquad (5.7.14)$$

После суммирования интегралов по всем отрезкам, получим составную формулу Симпсона:

$$\int_a^b f(x)dx \approx \frac{h}{3}\cdot\left[y_0+4(y_1+y_3+\ldots+y_{n-1})+2(y_2+y_4+\ldots+y_{n-2})+y_n\right] \qquad (5.7.15)$$

Часто пользуются простой формулой Симпсона:

$$\int_a^b f(x)dx = \frac{h}{3}\cdot\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right) . \qquad (5.7.16)$$

Пример 5.7.1. Вычислить интеграл

$$I=\int_0^1 \frac{dx}{1+x^2}.$$

Разбиваем интервал интегрирования на 10 равных частей: n=10. Шаг интегрирования h = (1-0)/10=0.1. Результаты вычислений подынтегральной функции приведены в табл. 21.

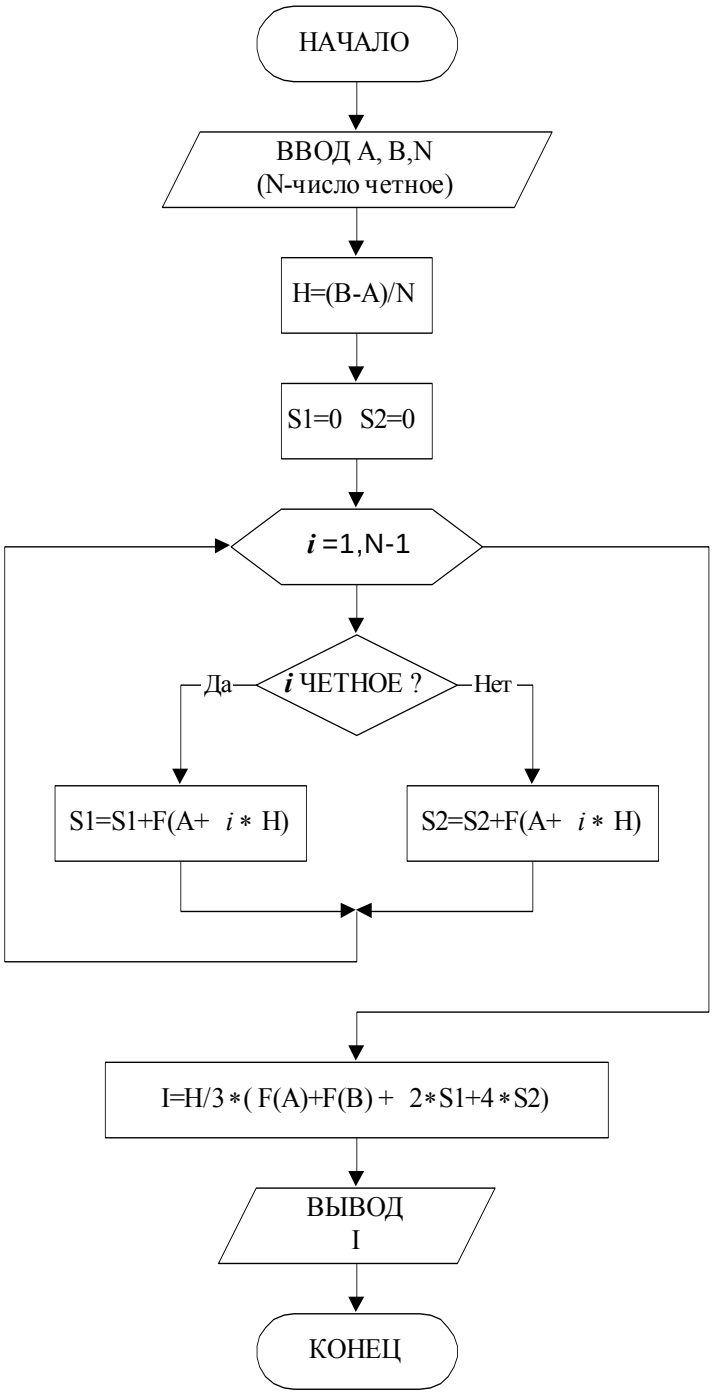


Рис.72. Блок-схема метода Симпсона

Таблица 21

$f(x)=\frac{1}{1+x^2}$					
x_i	x_i^2	$1+x_i^2$	$f(x_i)$ i=1,3,...	$f(x_i)$ i=2,4,...	$f(x_0),f(x_{10})$
0.0	0.00	1.00	—	—	1.00000
0.1	0.01	1.01	099010	—	—
0.2	0.04	1.04	—	0.96154	—
0.3	0.09	1.09	0.91743	—	—
0.4	0.16	1.16	—	0.76207	—
0.5	0.25	1.25	0.70000	—	—
0.6	0.36	1.36	—	0.073529	—
0.7	0.49	1.49	0.67114	—	—
0.8	0.64	1.64	—	0.60976	—
0.9	0.71	1.71	0.55249	—	—
1.0	1.00	2.00	—	—	0.50000
		Σ	3.93116	3.16766	1.50000

Для вычисления интеграла по методу прямоугольников, необходимо вычислить значения функции в середине каждого элементарного отрезка: $f\left(\frac{x_{i+1}+x_i}{2}\right)$ или $f(\overline{x_i})$. Результаты вычислений подынтегральной функции приведены в табл.22.

Таблица 22

$f(x)=\frac{1}{1+x^2}$			
x_i	$\overline{x}_i$	$\overline{x}_i^2$	$f(x)=\frac{1}{1+\overline{x}^2}$
0.0	0.05	0.0025	0.9975
0.1	0.15	0.0225	0.9770
0.2	0.25	0.0625	0.9412
0.3	0.35	0.1225	0.7907
0.4	0.45	0.2025	0.7316
0.5	0.55	0.3025	0.7677
0.6	0.65	0.4225	0.7030
0.7	0.75	0.5625	0.6400

0.8	0.75	0.7225	0.5705
0.9	0.95	0.9025	0.5256
1.0	1.00	1.00	—
		Σ	7.756

Вычислим интеграл по формуле трапеций (5.7.12):

$$I = \int_0^1 \frac{dx}{1+x^2} = 0.1(\frac{1+0.5}{2} + 3.93116 + 3.16866) = 0.78498,$$

по формуле Симпсона (5.7.15):

$$I = \int_0^1 \frac{dx}{1+x^2} = \frac{0.1}{3}(1+0.5+4*3.93116+2*3.16866) = 0.78540.$$

По формуле прямоугольников (5.7.6) получим:

$$I = \int_0^1 \frac{dx}{1+x^2} = 0.1 \cdot 7.856 = 0.78560.$$

Найдем точное значение интеграла:

$$I = \int_0^1 \frac{dx}{1+x^2} = \frac{\pi}{4} = 0.785398.$$

Относительная погрешность при применении формулы трапеций составляет 0,05 %, формулы прямоугольников - 0,026 %, формулы Симпсона - 0,00025 %. Таким образом, точность вычислений по формуле Симпсона выше, чем по формулам трапеций и прямоугольников.

Пример 5.7.2. Вычислить значение энтропии воды при нагревании ее от 400 до 500 К по формуле:

$$\Delta S = n \int_{400}^{500} \frac{C_v dT}{T} \quad .$$

Принимаем количество молей n=1, значение теплоемкости при v=const:

Cv=35.0 Дж/моль∗К .

Разобьем интервал интегрирования на 10 равных частей. Шаг интегрирования будет равен h=(500 – 400) /10 =10.

Результаты вычислений подынтегральной функции поместим в табл. 23.

Вычислим интеграл, используя данные табл.23:

по формуле трапеций (5.7.12):

$$\Delta S = \int_{400}^{500} \frac{C_v dT}{T} = 10(\frac{0.1575}{2} + 0.39044 + 0.31189) = 7.8108,$$

по формуле Симпсона (5.7.15):

$$\Delta S = \int_{400}^{500} \frac{C_v dT}{T} = \frac{10}{3}(0.1575 + 4 * 0.39044 + 2 * 0.31189) = 7.8101 ,$$

Таблица 23

$f(T) = \frac{Cv}{T} = \frac{35}{T}$					
T	**F (T$_I$) I=1,3 , …**	f (T$_i$) i=2,4, …	f (T$_0$) f (T$_{10}$)	$\overline{T}$	$f(\overline{T}) = \frac{35}{\overline{T}}$
400	—		0.0875	405	0.08642
410	0.08536			415	0.08434
420		0.08333		425	0.08235
430	0.08140			435	0.08046
440		0.07955		445	0.07865
450	0.07778			455	0.07692
460		0.07609		465	0.07527
470	0.07447			475	0.07368
480		0.07292		485	0.07216
490	0.07143			495	0.07071
500			0.0700		
Σ	0.39044	0.31189	0.1575		0.78096

по формуле прямоугольников (5.7.6):

$$\Delta S = \int_{400}^{500} \frac{C_v dT}{T} = 10 * 0.78096 = 7.8096 .$$

Найдем точное значение интеграла:

$$\Delta S = \int_{400}^{500} \frac{C_v dT}{T} = Cv * \ln \frac{T2}{T1} = 7.8100 .$$

Относительная погрешность вычислений по формуле трапеций, Симпсона и прямоугольников составляет соответственно: 0.01 %, 0.001 %, 0.005 %. Таким образом, наибольшую точность вычислений получили по формуле Симпсона.

5.8. Численное решение обыкновенных дифференциальных уравнений

5.8.1. Постановка задачи

Большинство балансовых уравнений в химии и химической технологии представлены системой дифференциальных уравнений, в результате решения которых могут быть получены зависимости, характеризующие протекание процесса. Уравнения, содержащие производную функции одной переменной, возникают во многих областях прикладной математики. Любая физическая ситуация, где рассматривается степень изменения одной переменной по отношению к другой, описывается дифференциальным уравнением, а такие ситуации встречаются весьма часто.

Обыкновенные дифференциальные уравнения широко используются для математического моделирования химико-технологических процессов. С помощью обыкновенных дифференциальных уравнений, например, исследуется кинетика химических реакций, процессы, протекающие в химических реакторах, массообменных и теплообменных аппаратах.

Дифференциальные уравнения устанавливают связь между независимыми переменными, искомыми функциями и их производными. Если искомая функция зависит от одной переменной, то дифференциальное уравнение называется *обыкновенным.*

Например, структура движения потока в реакторе идеального перемешивания описывается обыкновенным дифференциальным уравнением:

$$\frac{dC}{dt} = \frac{1}{\tau}(C_0 - C).$$

Здесь искомая функция (концентрация вещества) C(t) зависит от одной переменной t (времени).

В том случае, если искомая функция зависит от нескольких переменных, дифференциальное уравнение будет уравнением в *частных производных.*

Например, структуру потока в реакторе идеального вытеснения можно описать уравнением в частных производных:

$$\frac{\partial C}{\partial t} = -U\frac{\partial C}{\partial l}.$$

В этом уравнении функция C(t,l) зависит от времени (t) и длины аппарата (l).

В настоящей главе нами будут рассмотрены методы решения обыкновенных дифференциальных уравнений первого порядка.

Обыкновенными дифференциальными уравнениями (ОДУ) называются уравнения, которые содержат одну или несколько производных от искомой функции y = y(x):

$$F(x, y, y', \ldots, y^{(n)}) = 0, \qquad (5.8.1)$$

где x - независимая переменная.

Наивысший порядок *n*, входящей в уравнение (5.8.1) производной, называется *порядком дифференциального уравнения.*

Например:

$F(x, y, y') = 0$ – уравнение первого порядка;

$F(x, y, y', y'') = 0$ – уравнение второго порядка.

Из общей записи дифференциального уравнения (5.8.1) можно выразить производную в явном виде:

$$y' = f(x, y),$$
$$y'' = f(x, y, y'). \qquad (5.8.2)$$

Уравнение (5.8.2) имеет бесконечное множество решений. Для получения единственного решения необходимо указать дополнительные условия, которым должны удовлетворять искомые решения.

В зависимости от вида таких условий рассматривают три типа задач, для которых доказано существование и единственность решений.

Первый тип – это задачи с начальными условиями.

Для таких задач кроме исходного уравнения (5.8.1) в некоторой точке x_0 должны быть заданы начальные условия, т.е. значения функции y (x) и её производных: $y(x_0) = y_0$,

$$y'(x_0) = y'_0, \ldots, y^{(n-1)}(x_0) = y^{n-1}{}_0.$$

Второй тип задач – это, так называемые, граничные, или краевые, в которых дополнительные условия задаются в виде функциональных соотношений между искомыми решениями.

Количество условий должно совпадать с порядком уравнения или системы **n**. Если решение задачи определяется в интервале $x \in [x_0, x_k]$, то такие условия могут быть заданы как на границах, так и внутри интервала. Минимальный порядок обыкновенных дифференциальных уравнений, для которых может быть сформулирована граничная задача, равен двум.

Третий тип задач для обыкновенных дифференциальных уравнений – это задачи на собственные значения.

Такие задачи отличаются тем, что кроме искомых функций y(x) и их производных в уравнения входят дополнительно m неизвестных параметров $\lambda_1, \lambda_2, \ldots, \lambda_m$, которые называются собственными значениями. Для единственности решения на интервале $[x_0, x_k]$ необходимо задать n + m граничных условий.

Большинство методов решения обыкновенных дифференциальных уравнений основано на ***задаче Коши.***

Сформулируем задачу Коши.

Дано обыкновенное дифференциальное уравнение (ОДУ) первого порядка, разрешенное относительно производной

$$y' = f(x, y), \qquad (5.8.3)$$

удовлетворяющее начальному условию

$$y(x_0) = y_0. \qquad (5.8.4)$$

Необходимо найти на отрезке $[x_0,x_n]$ такую непрерывную функцию y = y(x), которая удовлетворяет дифференциальному уравнению (5.8.3) и начальному условию (5.8.4), т.е. найти решение дифференциального уравнения. Нахождение такого решения называют решением *задачи Коши*. Численное решение этой задачи состоит в построении таблицы приближенных значений $y_1,y_2,...,y_n$ решения уравнения y(x) в точках $x_1,x_2,...,x_n$ с некоторым шагом h.

$$x_i = x_0 + i \cdot h, \qquad i = 1,2,\ldots,n.$$

К численному решению обыкновенных дифференциальных уравнений приходится обращаться, когда не удаётся построить аналитическое решение задачи через известные функции, хотя для некоторых задач численные методы оказываются более эффективными даже при наличии аналитических решений.

5.8.2. Методы Рунге – Кутта

Широкая категория методов, наиболее часто применяемых на практике для решения дифференциальных уравнений, известна под общим названием: методы Рунге - Кутта. Различные методы этой категории требуют большего или меньшего объема вычислений и, соответственно, обеспечивают большую или меньшую точность.

Методы Рунге - Кутта обладают следующими отличительными свойствами:

- эти методы являются одноступенчатыми: чтобы найти значение функции в точке y_{i+1} нужна информация только о предыдущей точке (y_i,x_i);
- они согласуются с рядом Тейлора вплоть до членов порядка h^k, где степень k определяет порядок метода;
- эти методы не требуют вычисления производных от f(x,y), а требуют

вычисления самой функции.

Именно благодаря последнему свойству методы Рунге - Кутта более удобны для практических вычислений.

5.8.2.1. Метод Эйлера (метод Рунге - Кутта первого порядка)

Простейшим из численных методов решения дифференциальных уравнений является *метод Эйлера*. Это один из самых старых и широко известных методов. Метод Эйлера является сравнительно грубым методом решения дифференциальных уравнений, однако идеи, положенные в его основу, являются, по существу, исходными для очень широкого класса численных методов.

Пусть требуется найти приближенное решение дифференциального уравнения первого порядка

$$y' = f(x, y) \qquad (5.8.5)$$

с начальным условием

$$y(x_0)=y_0, \qquad (5.8.6)$$

т.е. необходимо решить задачу Коши.

В окрестности точки x_0 функцию $y(x)$ разложим в ряд Тейлора

$$y(x) = y(x_0) + (x - x_0) \cdot y'(x_0) + \frac{(x - x_0)^2}{2} \cdot y''(x_0) + \ldots, \qquad (5.8.7)$$

который можно применить для приближенного определения искомой функции $y(x)$. В точке x_0+h при малых значениях h можно ограничиться двумя членами ряда (5.8.7), тогда,

$$y(x) = y(x_0 + h) = y(x_0) + y'(x_0) \cdot \Delta x + 0(h^2), \qquad (5.8.8)$$

где $O(h^2)$ - бесконечно малая величина порядка h^2. Заменим производную $y'(x_0)$, входящую в формулу (5.8.7), на правую часть уравнения (5.8.5)

$$y(x_0 + h) \approx y_0 + h \cdot f(x_0, y_0), \qquad (5.8.9)$$

Теперь приближенное решение в точке $x_1=x_0+h$ можно вновь рассматривать как начальное условие и по формуле (5.8.9) найти значение искомой функции в следующей точке $x_2=x_1+h$. В результате получен простейший алгоритм решения задачи Коши, который называется *методом Эйлера* или методом *ломаных.*

Метод Эйлера можно представить в виде последовательного применения формул:

для точки $x_1 = x_0 + h, \quad y_1 = y_0 + h \cdot y_0' = y_0 + h \cdot f(x_0, y_0),$

$$x_2 = x_1 + h, \quad y_2 = y_1 + h \cdot y_1' = y_1 + h \cdot f(x_1, y_1), \qquad (5.8.10)$$

. . . . . .

$$x_{i+1} = x_i + h, \quad y_{i+1} = y_i + h \cdot y_i' = y_i + h \cdot f(x_i, y_i). \qquad (5.8.11)$$

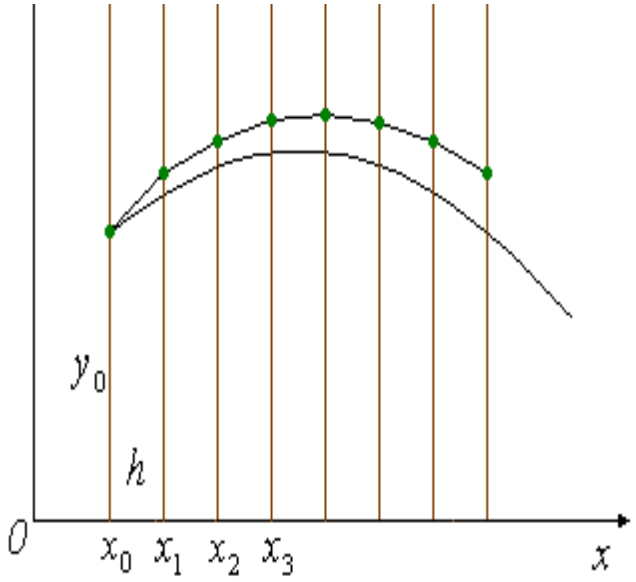


Рис.73

Таким образом, **формула Эйлера** в общем случае имеет вид:

$$y_{i+1} = y_i + h \cdot f(x_i, y_i), \qquad x_{i+1} = x_i + h. \qquad (5.8.12)$$

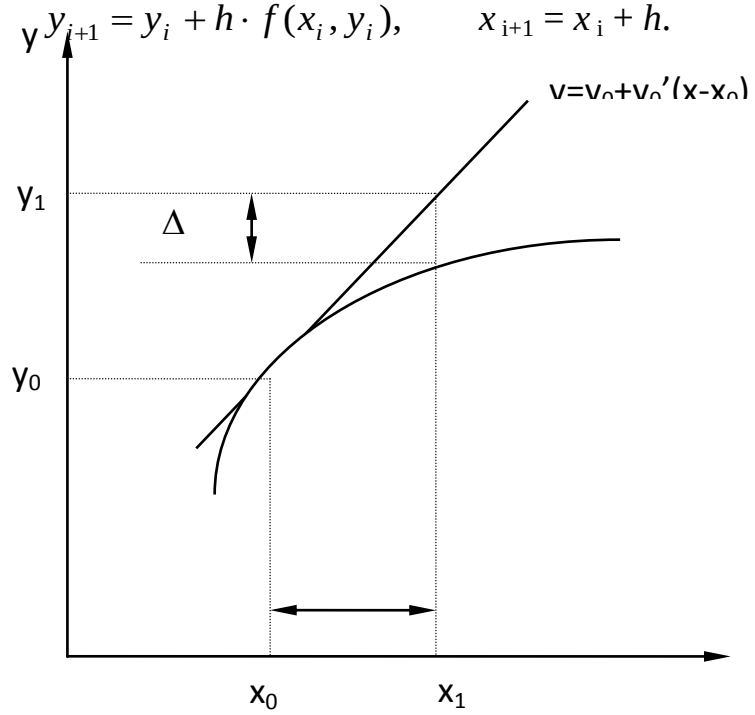


Рис.74

Название “метод ломаных” связано с его геометрической интерпретацией (рис.73): искомая функция $y(x)$ заменяется ломаной линией,
представляющей собой отрезки касательных к этой функции в узлах x_0, x_1,

Выведем формулы на основе геометрических аналогий.

Предположим, что нам известна точка (x_0,y_0) на искомой интегральной кривой (рис.74).

Через точку (x_0,y_0) проведем касательную с тангенсом угла наклона

$$tg\alpha = y_0' = f(x_0, y_0). \qquad (5.8.13)$$

Уравнение касательной имеет вид:

$$y = y_0 + y_0' \cdot (x - x_0).$$

Тогда в точке $x_1=x_0+h$, с учетом (5.8.13) получим решение:

$$y = y_0 + y_0' \cdot (x_0 + h - x_0);$$

$$y_1 = y_0 + h \cdot f(x_0, y_0).$$

Ошибка решения в точке $x=x_1$ показана в виде отрезка Δ.

Формула (5.8.12) является методом Рунге - Кутта первого порядка,

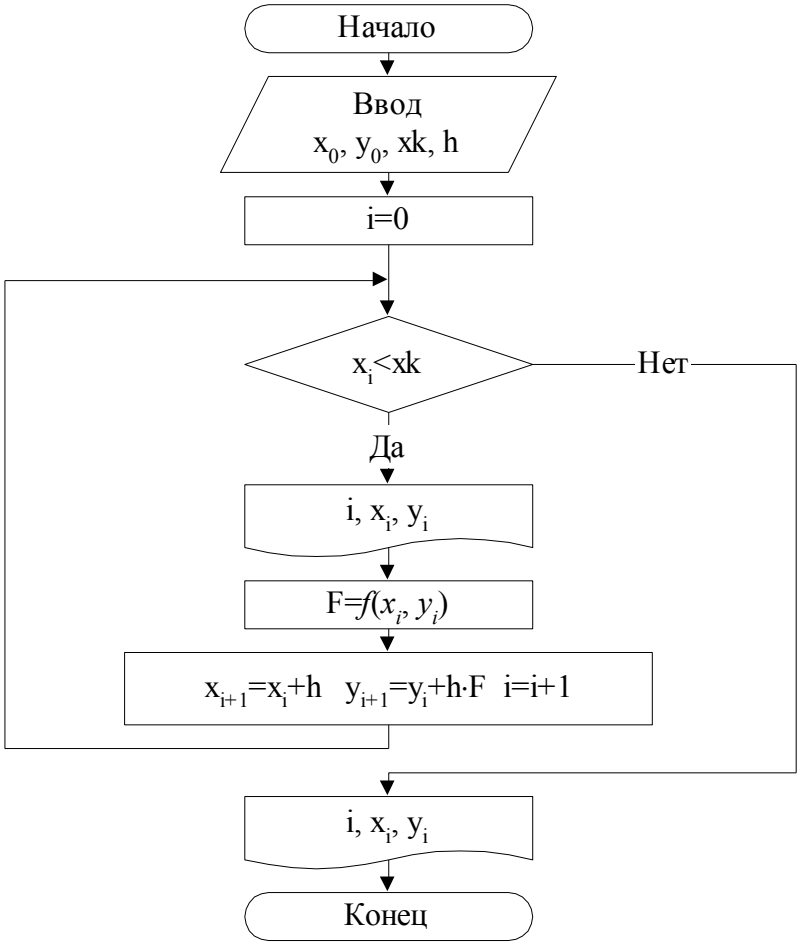


Рис. 75. Блок - схема метода Эйлера

т.к. она согласуется с разложением в ряд Тейлора вплоть до членов порядка h^1. Метод Эйлера имеет довольно большую погрешность вычисления: $\Delta \approx 0(h)$. Кроме того, он очень часто оказывается неустойчивым – малая ошибка (например, заложенная в исходных данных) увеличивается с ростом **x**. На рис.75 приведена блок – схема метода Эйлера, в Приложении – программа расчета.

Пример 5.8.1. Для химической реакции

$$A \xrightarrow{k} B.$$

Изменение концентраций веществ А и В можно описать следующими кинетическими уравнениями.

$$\frac{dC_A}{dt} = -k \cdot C_A;$$

$$\frac{dC_B}{dt} = k \cdot C_A \quad ,$$

с начальными условиями:

при t=0, $C_A(0) = C_{A,0}$; $C_B(0) = C_{B,0}$.

Требуется получить зависимость изменения концентрации вещества А от времени, т.е. необходимо решить дифференциальное уравнение (решить задачу Коши).

Исходные данные:

$C_{A,0}$ = 1моль/л;

$C_{B,0}$ = 0;

$k = 0{,}2c^{-1}$, интервал интегрирования t = [0,5].

Обозначим $C_A = y$, тогда

$$f(y,t) = \frac{dy}{dt} = -k \cdot y. \qquad (5.8.14)$$

Примем величину шага h = 0,1. Решим данное уравнение методом Эйлера (5.8.12).

Последовательность решения.

Найдем решение в точке:

1. $t_1 = t_0 + h = 0 + 0{,}1 = 0{,}1;$

$$y_1 = y_0 + h \cdot f(y_0, t_0);$$

$$f(y_0, t_0) = -0{,}2 \cdot 1 = -0{,}2;$$

$$y_1 = 1 + 0{,}1 \cdot (-0{,}2) = 0{,}98;$$

2. $t_2 = t_1 + h = 0{,}1 + 0{,}1 = 0{,}2;$

$$y_2 = y_1 + h \cdot f(y_1, t_1);$$

$$f(y_1, t_1) = -0{,}2 \cdot 0{,}98 = -0{,}196;$$

$$y_2 = 0{,}98 + 0{,}1 \cdot (-0{,}196) = 0{,}9604;$$

3. $t_3 = t_2 + h = 0{,}2 + 0{,}1 = 0{,}3;$

$$y_3 = y_2 + h \cdot f(y_2, t_2);$$
$$f(y_2, t_2) = -0{,}2 \cdot 0{,}9604 = -0{,}1921;$$
$$y_3 = 0{,}9604 + 0{,}1 \cdot (-0{,}1921) = 0{,}9412;$$

4. $t_4 = t_3 + h = 0{,}3 + 0{,}1 = 0{,}4;$

$$y_4 = y_3 + h \cdot f(y_3, t_3);$$
$$f(y_3, t_3) = -0{,}2 \cdot 0{,}9412 = -0{,}1882;$$
$$y_4 = 0{,}9412 + 0{,}1 \cdot (-0{,}1882) = 0{,}9224.$$

Для того чтобы проинтегрировать данное уравнение на интервале t=[0,5] с шагом h=0.1, потребуется n=t/h=5/0.1=50 шагов вычислений.

На примере первых четырех шагов мы показали последовательность вычислений по методу Эйлера. Если аналогично выполнить расчеты во всех точках, то в конечный момент времени t=5 сек., концентрация вещества А будет равна:

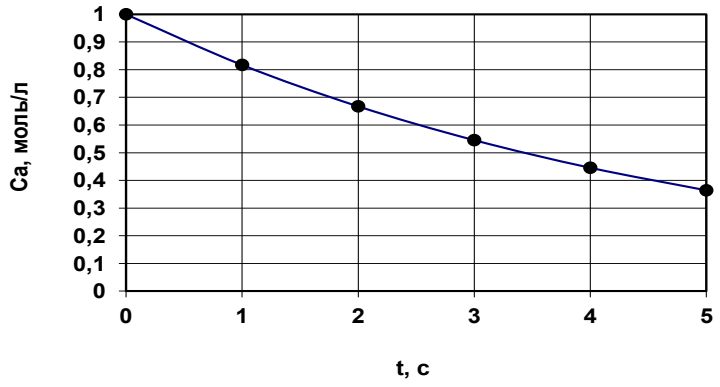
C_A=0,3642 моль/л.

Точное решение дифференциального уравнения (5.8.14) в точке t=5: C_A=0,3679 моль/л. Относительная ошибка метода Эйлера составляет ≈1%. Значения концентраций вещества А, рассчитанные по методу Эйлера и точные значения C_A в точках t=1,2,3,4,5 сек. приведены в табл.24.

Таблица 24

t_i, Сек.	C_A, моль/л		
	Метод Эйлера	Метод Рунге - Кутта	Точное решение
1	0,8171	0,8185	0,8187
2	0,6676	0,6699	0,6703
3	0,5455	0,5483	0,5488
4	0,4457	0,4487	0,4493
5	0,3642	0,3673	0,3679

Построим график изменения концентрации вещества А в зависимости от времени.



Для получения более точных результатов по методу Эйлера используют различные приемы. Рассмотрим уточненный метод Эйлера.

Уточненный метод Эйлера

При получении формулы Эйлера (5.8.12) на основании (5.8.5) полагали функцию f(x, y(x)) равной ее значению $f(x_i, y_i)$ на левом конце отрезка интегрирования.

Более точное значение получится, если принять функцию f(x, y(x)) в формуле (5.8.5) равной значению в центре отрезка. Так как значение производной между точками x_i и x_{i+1} не вычисляется, то возьмем двойной участок (x_{i-1}, x_{i+1}).

Из (5.8.5) следует

$$y_{i+1} = y_i + \int_{x_i}^{x_{i+1}} y' dx = y_i + \int_{x_i}^{x_{i+1}} f(x, y(x))dx; \qquad (5.8.15)$$

С учетом двойного участка, формулу (5.8.15) можно представить в виде:

$$y_{i+1} - y_{i-1} = \int_{x_{i-1}}^{x_{i+1}} f(x, y(x))dx. \qquad (5.8.16)$$

Центром интервала (x_{i-1}, x_{i+1}) является точка x_i. Поэтому, заменив в формуле (5.8.16) под интегралом функцию $f(x,y(x))$ ее значением в точке x_i, равным $y'_i = f(x_i, y_i)$, придем к формуле:

$$y_{i+1} = y_{i-1} + 2 \cdot h \cdot y'_i. \qquad (5.8.17)$$

Эта формула и выражает уточненный метод Эйлера. Однако она применима лишь при $i \neq 1$, а значение y_1 по ней получить нельзя. Следовательно, мы не можем выполнить и дальнейших вычислений, т.к. для нахождения y_2 надо иметь y'_1, для чего, в свою очередь, надо иметь значение y_1. В данном случае рекомендуется сначала определить y_1 по формуле (5.8.12), а последующие значения y_i ($i \neq 1$) определять по формуле (5.8.17).

Более точные результаты получатся, если сначала найти $y_{1/2}$ в точке $x_{1/2}=x_0+h/2$ по формуле (5.8.12), а затем искать y_1 по формуле (5.8.17) с шагом h/2. Иначе говоря, рекомендуется следующая последовательность действий:

$$\begin{aligned} y_{1/2} &= y_0 + \frac{h}{2} \cdot y'_0 = y_0 + \frac{h}{2} \cdot f(x_0, y_0), \\ y_1 &= y_0 + h \cdot y'_{1/2} = y_0 + h \cdot f(x_{1/2}, y_{1/2}), \\ y_2 &= y_0 + 2 \cdot h \cdot y'_1 = y_0 + 2 \cdot h \cdot f(x_1, y_1), \end{aligned} \qquad (5.8.18)$$

После этого дальнейшие вычисления идут уже по формуле (5.8.17) без изменений.

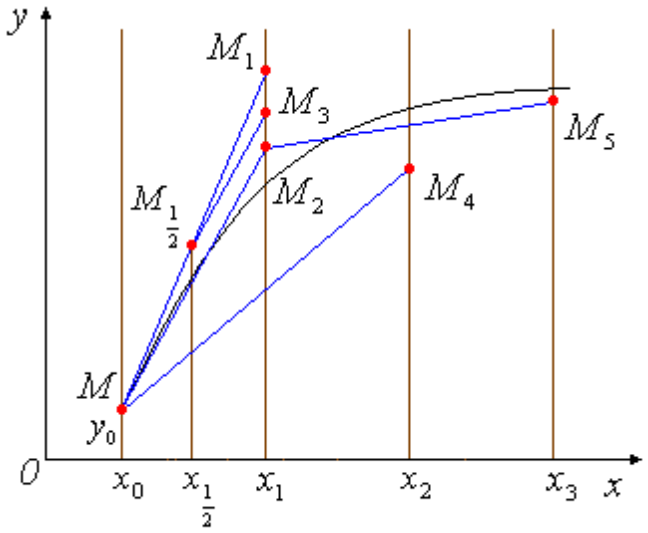


Рис. 76

Рассмотрим геометрический смысл уточненного метода Эйлера (рис. 76). Исходя из начальной точки $M(x_0,y_0)$, получаем по методу Эйлера для $x=x_{1/2}$ точку $M_{1/2}$. Для $x=x_1$ способ Эйлера дал бы точку M_1, находящуюся на касательной к интегральной кривой в точке M. Уточненный способ состоит в том, что из точки M проводится отрезок MM_2, параллельный отрезку, направленному в соответствии со значением $M_{1/2}$ углового коэффициента в точке M. Точка M_2, которая получена по уточненному методу Эйлера, находится ближе к истинной кривой, чем точка M_1. В дальнейшем для получения каждой следующей точки проводится отрезок над участком 2h параллельно направлению, которое определяется значением производной в середине этого участка: MM_4, M_2M_5.

5.8.2.2. Методы Рунге - Кутта второго порядка

Пусть дано дифференциальное уравнение первого порядка:

$$y' = f(x, y)$$

с начальным условием $x=x_0$, $y=y_0$.

В окрестности точки x_0 функцию $y(x)$ разложим в ряд Тейлора:

$$y(x) = y(x_0) + (x - x_0) \cdot y'(x_0) + \frac{(x - x_0)^2}{2} \cdot y''(x_0) + \ldots,$$

который можно применить для приближенного определения искомой функции $y(x)$. Для уменьшения погрешности метода интегрирования дифференциального уравнения необходимо учитывать большее количество членов ряда. Однако при этом возникает необходимость аппроксимации производных от правых частей дифференциального уравнения. Основная идея методов Рунге-Кутта заключается в том, что производные аппроксимируются через значения функции $f(x,y)$ в точках на

интервале $[x_0, x_{0+h}]$, которые выбираются из условия наибольшей близости алгоритма к ряду Тейлора. В зависимости от старшей степени h, с которой учитываются члены ряда, построены вычислительные схемы Рунге-Кутта разных порядков точности.

Так, например, общая форма записи метода Рунге - Кутта второго порядка следующая:

$$y_{i+1} = y_i + h[(1-\alpha)\cdot f(x_i, y_i) + \\ + \alpha \cdot f(x_i + \frac{h}{2\alpha}, y_i + \frac{h}{2\alpha}\cdot f(x_i, y_i))] + 0(h^3), \qquad (5.8.19)$$

где $0 < \alpha \le 1.$

Решение ОДУ, полученное по этой схеме имеет погрешность $0(h^2)$.

Для параметра α наиболее часто используют значения

α=0,5 и α=1.

Рассмотрим *первый* вариант метода Рунге - Кутта второго порядка.

При α=0,5 формула (5.8.19) примет вид:

$$y_{i+1} = y_i + \frac{h}{2}[f(x_i, y_i) + f(x_i + h, y_i + h\cdot f(x_i, y_i))]. \qquad (5.8.20)$$

Формулу (5.8.20) можно представить в виде следующей схемы:

$$y_{i+1} = y_i + \Delta y_i, \qquad (5.8.21)$$

где

$$\Delta y_i = \frac{1}{2}(k_1 + k_2),$$

$$k_1 = h\cdot f(x_i, y_i),$$

$$k_2 = h\cdot f(x_i + h, y_i + k_1).$$

Это метод Рунге - Кутта второго порядка (1-ый вариант) или *исправленный* метод Эйлера.

Геометрически процесс нахождения точки x_1,y_1 можно проследить по рис.77. По методу Эйлера находится точка $x_0+h, y_0+h\cdot y'_0$, лежащая на прямой L1. В этой точке снова вычисляется тангенс угла наклона касательной (прямая L2). Усреднение двух тангенсов дает прямую $\overline{L}$. Проводим через точку x_0,y_0 прямую L, параллельную $\overline{L}$. Точка, в которой прямая L пересечется с ординатой $x=x_1=x_0+h$, и будет искомой точкой x_1,y_1.

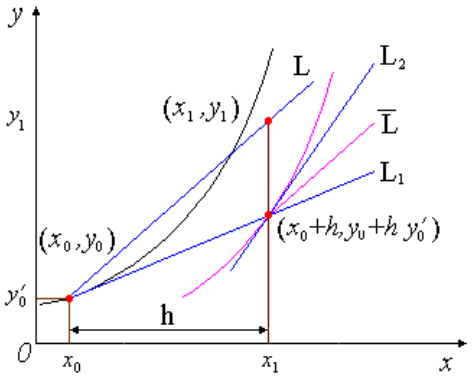
Тангенс угла наклона прямой $\overline{L}$ и L равен:

$$y'_0 = \frac{1}{2}[f(x_{0,}y_0) + f(x_0 + h, y_0 + h\cdot y'_0]. \qquad (5.8.22)$$

Уравнение прямой L запишется в виде:

$$y = y_0 + (x - x_0)\cdot y'_0, \qquad (5.8.23)$$

тогда в точке x=x1=x0+h с учетом (5.8.22) получим решение:



$$y_1 = y_0 + h \cdot \frac{1}{2}[f(x_{0,}y_0) + f(x_0 + h, y_0 + h \cdot y_0^{'}], \qquad (5.8.24)$$

Данная формула описывает метод Рунге – Кутта второго порядка при α=0.5.

В случае *второго* варианта метода Рунге – Кутта второго порядка принимают

$$\alpha=1.$$

Тогда формула (5.8.19) примет вид:

$$y_{i+1} = y_i + h \cdot f(x_i + \frac{h}{2}, y_0 + h \cdot \frac{f(x_i, y_i)}{2}. \qquad (5.8.25)$$

Представим формулу (5.8.25) в виде схемы:

$$\begin{aligned} &y_{i+1} = y_i + \Delta y_i, \\ &\Delta y_i = k_2, \\ &k_1 = h \cdot f(x_i, y_i), \\ &k_2 = h \cdot f(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}). \end{aligned} \qquad (5.8.26)$$

Это метод Рунге – Кутта второго порядка (2-ой вариант) или *модифицированный метод Эйлера.*

Рассмотрим геометрическую интерпретацию метода Рунге – Кутта при α=1 (рис. 78).

Через точку x_0,y_0 проводим касательную (прямая L1) с тангенсом угла наклона равным

$$y_0^{'} = f(x_{0,}y_0). \qquad (5.8.27)$$

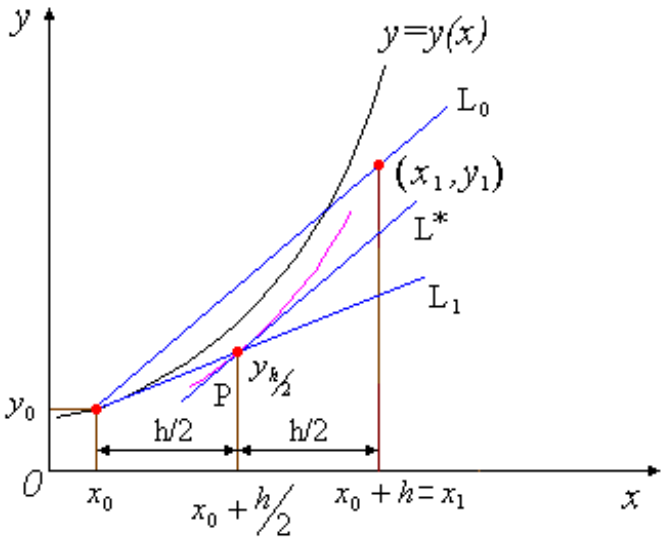


Рис.78

По методу Эйлера в точке $x=x_0+h/2$ находится приближенное решение ОДУ

$$y = y_0 + \frac{h}{2} \cdot y_0'.$$

В точке P определяется тангенс угла наклона касательной (прямая L*) интегральной кривой:

$$y_0' = f(x_0 + \frac{h}{2}, y_0 + \frac{h}{2} \cdot y_0'), \qquad (5.8.28)$$

$$\text{где } y_0' = f(x_0, y_0).$$

Проводим через точку x_0,y_0 прямую, параллельную L^* (прямая L_0). Пересечение этой прямой с ординатой $x=x_0+h$ и дает искомую точку x_1,y_1. Уравнение прямой L_0 можно записать в виде

$$y = y_0 + (x - x_0) \cdot y_0'. \qquad (5.8.29)$$

Тогда с учетом (5.8.28) в точке $x= x_0+h$ получим решение:

$$y_1 = y_0 + h \cdot f(x_0 + \frac{h}{2}, y_0 + \frac{h}{2} y_0'). \qquad (5.8.30)$$

Формула (5.8.30) описывает метод Рунге – Кутта второго порядка при $\alpha=1$.

5.8.2.3. Метод Рунге – Кутта четвертого порядка

Методы Рунге – Кутта третьего и четвертого порядков можно вывести аналогично тому, как это делалось при выводе методов первого и второго порядков.

Не будем воспроизводить эти выкладки, а приведем формулы, описывающие метод четвертого порядка, один из самых применимых

методов интегрирования ОДУ. Этот метод применяется настолько широко, что в литературе просто называется ”методом Рунге – Кутта” без указаний на тип и порядок. Этот классический метод Рунге – Кутта описывается системой следующих соотношений:

$$y_{i+1} = y_i + \Delta y_i \qquad (5.8.31)$$

или

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

где

$$k_1 = h \cdot f(x_i, y_i), \qquad (5.8.32)$$

$$k_2 = h \cdot f(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}),$$

$$k_3 = h \cdot f(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}),$$

$$k_4 = h \cdot f(x_i + h, y_i + k_3).$$

Геометрическая интерпретация метода представлена на рис. 79.

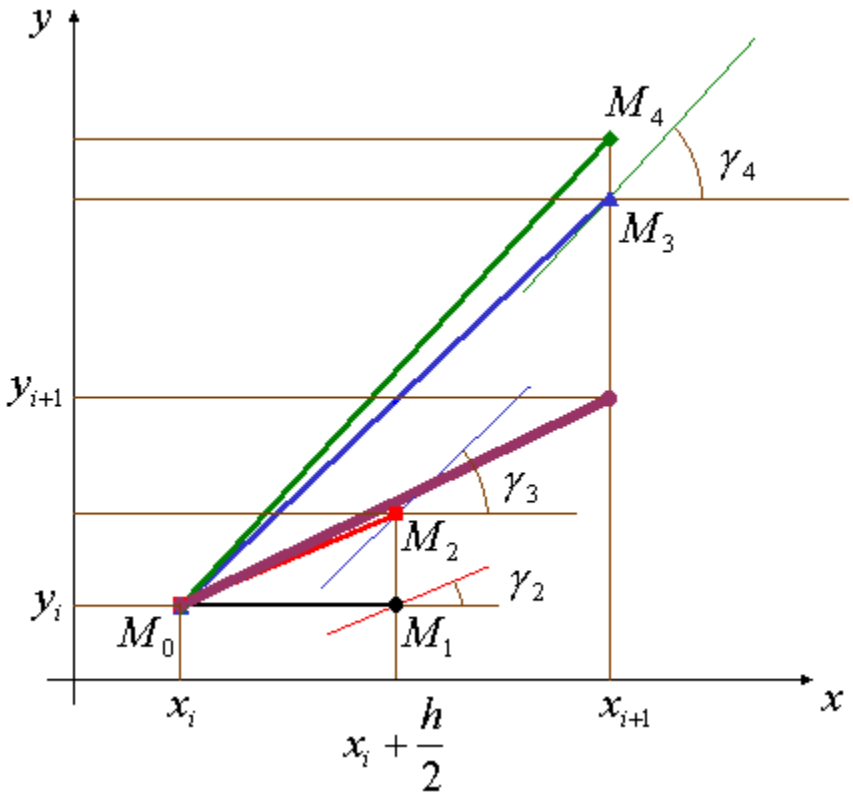


Рис.79

Порядок построения:

1. С шагом h/2 из точки $M_0(x_i,y_i)$ под углом $y_1=tg\ (k_1/h)$ проводим прямую в точку $M_1(x_i+h/2,y_i+h/2)$.
2. В точке M_1 вычисляем направление $tg(y_2)=k_2/h$ и, делая шаг в этом

направлении, из точки M_0 попадаем в точку $M_2(x_i+h/2,y_i+h/2)$.

3. В точке M_2 вычисляем $tg(y_3)=k_3/h$ и, делая шаг в этом направлении, из точки M_0 попадаем в точку $M_3(x_i+h/2,y_i+k_3)$.
4. В точке M_3 вычисляем $tg(y_3)=k_3/h$.
5. Полученные величины k1,k2,k3,k4 усредняются по формуле:

$$\Delta y_i = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \qquad (5.8.33)$$

6. Используя величину Δy_i делаем окончательный шаг из (x_i,y_i) в (x_{i+1},y_{i+1}).

Блок-схема метода Рунге-Кутта четвертого порядка приведена на рис.80.

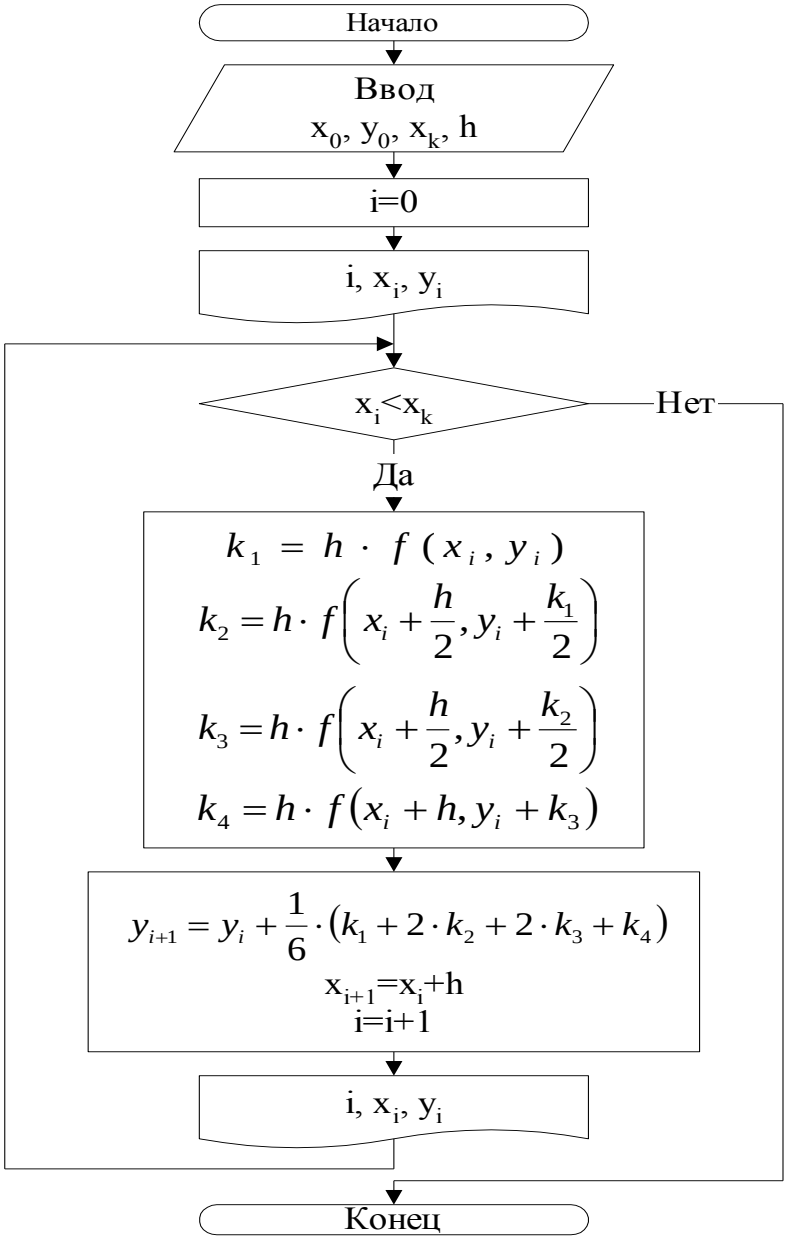


Рис. 80. Блок – схема метода Рунге - Кутта

Пример 5.8.2. Для дифференциального уравнения, приведенного в примере 5.8.1. покажем последовательность решения методом Рунге - Кутта четвертого порядка

$$\frac{dC_A}{dt} = -k \cdot C_A;$$

Или

$$\frac{dy}{dt} = -k \cdot y,$$

начальные условия : при t=0, y_0=1.

t=[0,5]; h=0,1; k=0.2c^{-1}.

Найдем решения в точках:

1. $t_1 = t_0+h = 0+0,1=0,1;$

$$k_1 = h \cdot f(t_0, y_0) = 0.1 \cdot (-0.2) \cdot 1 = -0.02;$$

$$k_2 = h \cdot f(t_0 + \frac{h}{2}, y_0 + \frac{k_1}{2}) = 0.1 \cdot (-0.2) \cdot (1 - \frac{0.02}{2}) = -0.0198;$$

$$k_3 = h \cdot f(t_0 + \frac{h}{2}, y_0 + \frac{k_2}{2}) = 0.1 \cdot (-0.2) \cdot (1 - \frac{0.0198}{2}) = -0.0198;$$

$$k_4 = h \cdot f(t_0 + \frac{h}{2}, y_0 + k_3) = 0.1 \cdot (-0.2) \cdot (1 - 0.0198) = -0.0198;$$

$$y_1 = y_0 + \frac{1}{6} \cdot (k_1 + 2k_2 + 2k_3 + k_4)/6 =$$

$$= 1 + (-0,02 - 2 \cdot 0,0198 - 2 \cdot 0,0198 - 0,0198)/6 =$$

$$= 1 - \frac{0,0198}{6} = 0,980;$$

2. $t_2 = t_1+h = 0,1+0,1=0,2;$

$$k_1 = h \cdot f(t_1, y_1) = 0.1 \cdot (-0.2) \cdot 0,98 = -0,0196;$$

$$k_2 = h \cdot f(t_1 + \frac{h}{2}, y_1 + \frac{k_1}{2}) = 0.1 \cdot (-0.2) \cdot (0,98 - \frac{0.0196}{2}) = -0.0194;$$

$$k_3 = h \cdot f(t_1 + \frac{h}{2}, y_1 + \frac{k_2}{2}) = 0.1 \cdot (-0.2) \cdot (0,98 - \frac{0.0194}{2}) = -0.0194;$$

$$k_4 = h \cdot f(t_1 + \frac{h}{2}, y_1 + k_3) = 0.1 \cdot (-0.2) \cdot (0,98 - 0,0194) = -0.0194;$$

$$y_2 = y_1 + (k_1 + 2k_2 + 2k_3 + k_4)/6 =$$

$$= 0,98 + (-0,0196 - 2 \cdot 0,0194 - 2 \cdot 0,0194 - 0,0194)/6 =$$

$$= 0,98 - \frac{0,1166}{6} = 0,9607.$$

Выполнив аналогичным образом вычисления во всех последующих точках интервала [1,5] с шагом h=0.1, получим в момент времени t=5 концентрацию вещества: C_A=0.3673 моль/л. Ранее показано, что точное решение - C_A=0.3679, следовательно, относительная ошибка составляет

≈0,16%. В табл. 24 приведены результаты расчетов по методу Эйлера и Рунге - Кутта, а также точное решение дифференциального уравнения (5.8.14). Как видно из таблицы, наиболее точным является решение, полученное по методу Рунге - Кутта.

5.8.2.3. Системы дифференциальных уравнений

Очень часто приходится иметь дело с задачей, в которой необходимо решить систему нескольких дифференциальных уравнений.

Будем рассматривать нормальные системы дифференциальных уравнений, в которых уравнения разрешены относительно производных и число уравнений равно числу неизвестных функций. Например, система двух уравнений с двумя неизвестными функциями *y, z* от одного и того же аргумента *x* в нормальной форме имеет вид:

$$\begin{cases} y' = f_1(x, y, z) \\ z' = f_2(x, y, z) \end{cases}, \qquad (5.8.34)$$

причем штрих означает производную по *x*. Общий вид нормальной системы *n* уравнений с *n* неизвестными функциями x_1, x_2, ..., x_n от переменного *t* имеет вид:

$$\begin{aligned} \frac{dx_1}{dt} &= f_1(t, x_1, x_2, \ldots, x_n) \\ \frac{dx_2}{dt} &= f_2(t, x_1, x_2, \ldots, x_n) \\ &\ldots\ldots\ldots\ldots\ldots\ldots \\ \frac{dx_n}{dt} &= f_n(t, x_1, x_2, \ldots, x_n) \end{aligned} \qquad (5.8.35)$$

Рассмотренные численные методы решения дифференциального уравнения вида $y' = f(x, y)$ без труда переносятся на системы вида (5.8.35): каждый раз при переходе к следующей точке параллельно вычисляются приращения каждой из неизвестных функций по аналогичным формулам.

Так , для нормальной системы двух уравнений , $\begin{matrix} y' = f_1(x, y, z) \\ z' = f_2(x, y, z) \end{matrix}$

с начальными условиями (5.8.36)

$y(x_0) = y_0$,

$z(x_0) = z_0$,

используя метод Эйлера, можно записать расчетные формулы так:

$$\begin{cases} \Delta y_i = y_i^{'} \cdot h = f_1(x_i, y_i, z_i) \cdot h_i \\ \Delta z_i = z_i^{'} \cdot h = f_2(x_i, y_i, z_i) \cdot h_i \\ y_{i+1} = y_i + \Delta y_i \\ z_{i+1} = z_i + \Delta z_i . \end{cases} \qquad (5.8.37)$$

ЛИТЕРАТУРА

1. Информатика: Базовый курс./ Под ред. С.В. Симоновича .- С-Петербург: Питер, 2002.-640с.
2. Информатика: Учебник/ Под ред. Н.В. Макаровой. – М.: Финансы и статистика, 1997.-768 с.
3. Основы информатики и вычислительной техники: Учебное пособие/ А.И. Громов, М.Я. Сафин и др. - М.: Изд. УДМ, 1991. - 88 с.
4. Персональный компьютер для всех: В 4 кн. Кн.2 Подготовка и редактирование документов: Практ. пособие для вузов/ А.Я. Савельев, В.А. Сазонов, С.Э. Лукьянов; Под ред. А.Я. Савельева.- М.: Высшая школа, 1991.-190 с.
5. IBM PC для пользователя./В.Э. Фигурнов. -Уфа: Партнерская компания «Дегтярев и сын», 1993. - 300 с.
6. Курс молодого бойца./К.С. Ахметов. -М.: ТОО фирма «Компьютер Пресс», 1996. - 380 с.
7. Работа на IBM PC./А.М. Кенин, Н.С. Печенкина. -М.: АО «Книга и бизнес», 1992. - 368 с.
8. Миллер М. Использование Windows 98: Пер. с англ. – К.; М.; СПб.: Изд. Дом «Вильямс», 1998. – 336 с.
9. Андердал Б. Самоучитель Windows 98. – СПб.: «Питер», 1998. – 368с.
10. Леонтьев Б. Как установить и настроить Microsoft Windows 98. –М.: «Познавательная книга +», 1998. – 192 с.
11. Немнюгин С.А. Turbo Pascal. Учебник. –С-Петербург: 2000.- 491 с.
12. Программирование в среде Turbo PASCAL 6.0: Справ. пособие./Ю.С. Климов, А.И. Касаткин, С.М. Мороз. - М.: Высшая школа, 1992. - 160 с.
13. Офицеров Д.В., Старых В.А. Программирование в интегрированной среде Turbo-Pascal 6.0. - Минск:: Белорусь, 1992. - 240 с.
14. Алексеев В.Е. и др. Вычислительная техника и программирование. - М.: Высшая школа, 1991. - 400 с.
15. Епанешников В., Епанешников А.. Программирование в среде Турбо-Паскаль-7.- М.: 1994. -316 с.

16. Епанешников А.М. Программирование в среде Турбо– Паскаль-7.М.: «ДИАЛОГ-МИФИ»,1996.- 288с.
17. Турбо-Паскаль 7.0.- Киев: 1995. - 448 с.
18. Васюкова Н.Д., Тюляева В.В. Практикум по основам программирования. Язык Паскаль. - М.:Высшая школа, 1991. - 160 с.
19. Колесников А. EXCEL 97 – Издательская группа BHV, 1997 – 528 с.
20. Демидович Б.П., Марон И.А. Основы вычислительной математики.М.:Наука,1970 .-660с.
21. Турчак Л.И. Основы численных методов. - М.:Наука,1987.-318с.
22. Плис А.И., Сливина Н.А. Лабораторный практикум по высшей математике. -М.: Высшая школа,1983.-207с.
23. Мудров А.Е. Численные методы для ЭВМ на языках Бэйсик, Фортран,Паскаль. -Томск: МП ’РАСКО’,1991.-272с.
24. Калиткин Н.Н. Численные методы.-М.:Наука,1978.-512с.
25. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы.-М.:Наука,1987.-600с.
26. Ракитин В.И. Практическое руководство по методам вычислений.-М:Высшая школа,1998.-383с.
27. Гутер Р.С., Резниковский П.Т. Программирование и вычислительная математика.-М.:Наука,1971.-264с.
28. Гутер Р.С., Овчинский Б.В. Элементы численного анализа и математической обработки результатов опыта.-М.:Наука,1970.-432с.
29. Мак- Кракен Д., Дорн У. Численные методы и программирование на ФОРТРАНе. – М.: Мир,1977.-584с.
30. Воробьева Г.Н., Данилова А.Н. Практикум по вычислительной математике.-М.:Высшая школа,1990.-208с.
31. Джонсон К. Численные методы в химии: Пер. с англ.-М.: Мир, 1983.-504с.

Анатолий Васильевич Кравцов

Ольга Ефимовна Мойзес

Елена Анатольевна Кузьменко

Дмитрий Анатольевич Баженов,
Павел Иванович Коваль

Информатика и вычислительная математика

Научный редактор доктор техн. наук,
профессор А.В.Кравцов

Редактор Н.Я.Горбунова

ПОДПИСАНО К ПЕЧАТИ 17,02,2003.
Формат 60x84/16. Бумага офсетная
Печать RISO. Усл.печ.л. 15,5 Уч-изд.л.14,6
Тираж 250 экз. Заказ N . Цена свободная
Издательство ТПУ. Лицензия ЛТ N 1 от 18.07.94
Типография ТПУ. 634034, Томск, пр.Ленина,30.