УДК 536.2
Б 53

Беспалов В. В.
Б 53 Основы применения вычислительной техники и программирование: учебное пособие./В.В. Беспалов – Томск: Изд-во Томского политехнического университета, 2007. – 107 с.

В учебном пособии кратко изложен базовый курс изучения языка программирования Паскаль в интерактивной среде Турбо Паскаля (версии 7.0). Основной акцент сделан на практическое применение конструкций языка в наиболее типичных задачах программирования. По каждой теме приведены примеры программ. Также в пособие включены главы, рассматривающие численные методы решения инженерных задач и способы их реализации на языке Паскаль.

Пособие подготовлено на кафедре атомных и тепловых электрических станций ТПУ и предназначено для студентов всех специальностей направления 140100 – «Теплоэнергетика».

УДК 536.2

Рекомендовано к печати Редакционно-издательским советом Томского политехнического университета

Рецензенты:

Кандидат технических наук, директор ООО ПФ «Котлосиб» А.А. Купрюнин.

Кандидат технических наук, директор ООО «Теплоуниверсал» Б.В. Лебедев.

Оглавление

ВВЕДЕНИЕ

Целью курса «Основы применения вычислительной техники и программирование» является обучение студентов программированию с применением методов вычислительной математики, использованием современных средств вычислительной техники и компьютерных технологий, дальнейшее развитие компьютерной грамотности на основе дисциплин «Информатика», «Высшая математика», «Механика жидкости и газов», «Теоретические основы теплотехники».

Задачи изучения дисциплины заключаются в практическом освоении языка и среды Турбо Паскаля (версии 7.0), в приобретении студентами навыков составления алгоритмов задач теплоэнергетического профиля, отладки программ, в умении проводить анализ полученных результатов и корректировать свои действия с целью улучшения качественных показателей программ.

Язык Турбо Паскаль является классическим языком программирования, широко применяемым в инженерных расчётах. Его изучение позволяет сформировать у студентов особый вид мышления – алгоритмический. Студентам, успешно овладевшим этим языком, не составит особого труда в будущей своей трудовой деятельности применять свои знания и составлять программы не только на языке Паскаль, но и на других языках программирования. Особенно важным является то, что знание языка Паскаль нужно для составления программ в среде Windows при помощи прикладного пакета Delphi, всё более популярного в последнее время.

К настоящему моменту имеется огромное количество библиотек программ, процедур и функций с примерами реализации большинства инженерных задач на языке Паскаль и в среде визуального программирования Delphi. Умелое применение этих наработок предполагает хорошее базовое знание языка Паскаль.

В период обучения студенты должны освоить некоторые численные методы и способы их реализации на языке Паскаль, в том числе с использованием библиотек подпрограмм и внешних файлов данных.

1. ОСНОВНЫЕ ПОНЯТИЯ СИСТЕМЫ ПРОГРАММИРОВАНИЯ ТУРБО ПАСКАЛЬ

Большинство программ создаются для решения какой-либо задачи. В процессе ее решения на ПК необходимо: ввести данные, указать способ их обработки, задать способ вывода полученных результатов. Поэтому нужно знать следующее:

- Как ввести информацию в память (ввод).
- Как хранить информацию в памяти (данные).
- Как указать правильные команды для обработки данных (операции).
- Как передать данные из программы пользователю (вывод).

Необходимо также уметь упорядочивать команды так, чтобы:

- некоторые из них выполнялись только в случае, если соблюдается некоторое условие или ряд условий (условное выполнение);
- другие выполнялись повторно некоторое число раз (циклы);
- третьи выделялись в отдельные части, которые могут быть неоднократно выполнены в разных местах программы (подпрограммы).

Таким образом, нужно уметь использовать семь основных элементов программирования – ввод, данные, операции, вывод, условное выполнение, циклы и подпрограммы – и на их основе строить программы.

Этот список не является полным, однако, он содержит те элементы, которые присущи обычно всем программам. Многие языки программирования имеют еще и дополнительные средства, в том числе и Паскаль.

Основные файлы пакета Турбо Паскаль:

- **Turbo.exe** – интегрированная среда программирования;
- **Turbo.hlp** – файл, содержащий данные для оперативной подсказки;
- **Turbo.tp** – файл конфигурационной системы;
- **Turbo.tpl** – библиотека стандартных модулей Турбо Паскаля.

Для работы в графическом режиме необходимы **Graph.tru** – модуль с графическими процедурами и функциями Турбо Паскаля, несколько файлов с расширением ***.BGI** – драйверы различных типов видеосистем ПК, несколько файлов с расширением ***.CHR**, содержащих векторные шрифты.

После загрузки системы экран разделен на три части: основное (или рабочее) окно, главное меню и строка, в которой указывается на-

значение основных функциональных клавиш. Переход из основного окна в главное меню и обратно осуществляется посредством клавиши **F10**.

В рабочем окне осуществляется набор текста программы, запуск же происходит следующим образом: выход в меню, выбор пункта **Run – Run**.

Для того чтобы сохранить программу, необходимо: выйти в меню, выбрать **File – Save (Save as …)**, в появившемся окне ввести имя файла и нажать клавишу **Enter**.

Выход из системы программирования: выход в меню, пункт **File – Exit**.

1.1. Алфавит и словарь языка Паскаль

Язык – совокупность символов, соглашений и правил, используемых для общения. При записи алгоритма решения задачи на языке программирования необходимо четко знать правила написания и использования языковых единиц. Основой любого языка является алфавит (набор знаков, состоящий из букв, десятичных и шестнадцатеричных цифр, специальных символов).

Алфавит Паскаля составляют:

- прописные и строчные буквы латинского алфавита:
  A, B, C…Y, Z, a, b, c…y, z;
- десятичные цифры: **0, 1, 2…9;**
- специальные символы:
  + - * / > < = ; # ' , . : {} [] ( );
- комбинации специальных символов, которые нельзя разделять пробелами, если они используются как знаки операций:

«:=», «..», «<>», «<=», «>=», «{}».

Неделимые последовательности знаков алфавита образуют слова, отделенные друг от друга разделителями. Ими могут быть пробел, комментарий или символ конца строки. Словарь Паскаля можно разделить на три группы слов: зарезервированные слова, стандартные идентификаторы и идентификаторы пользователя.

Зарезервированные слова (см. табл. 1.1) имеют фиксированное написание и навсегда определенный смысл. Они не могут изменяться программистом, и их нельзя использовать в качестве имен для обозначения величин.

Идентификатор – имя (*identification* – установление соответствия объекта некоторому набору символов). Для обозначения определенных разработчиками языка функций, констант и т. д. служат стандартные идентификаторы, например, **Sqr, Sqrt** и т. д. В этом примере **Sqr** вызывает функцию, которая возводит в квадрат данное число, а **Sqrt** – корень квадратный из заданного числа. Пользователь может переопределить любой стандартный идентификатор, но чаще всего это приводит

к ошибкам, поэтому на практике их используют без изменения. Идентификаторы пользователя – это те имена, которые дает сам программист. При записи программ нужно соблюдать общие правила написания идентификаторов:

- Идентификатор начинается только с буквы (исключение составляют специальные идентификаторы меток).
- Идентификатор может состоять из букв, цифр и знака подчеркивания.
- Максимальная длина – 127 символов.
- При написании идентификаторов можно использовать прописные и строчные буквы.
- Между двумя идентификаторами должен стоять хотя бы один пробел.

Некоторые зарезервированные слова версии Турбо Паскаль.

Таблица 1.1

Absolute	Абсолютный	**Library**	Библиотека
And	Логическое И	**Mod**	Остаток от деления
Array	Массив	**Not**	Логическое НЕ
Begin	Начало блока	**Or**	Логическое ИЛИ
Case	Вариант	**Of**	Из
Const	Константа	**Object**	Объект
Div	Деление нацело	**Procedure**	Процедура
Goto	Переход на	**Program**	Программа
Do	Выполнять	**Repeat**	Повторять
Downto	Уменьшить до	**String**	Строка
Else	Иначе	**Then**	То
End	Конец блока	**To**	Увеличивая до
File	Файл	**Type**	Тип
For	Для	**Until**	До тех пор, пока не выполнится
Function	Функция	**Uses**	Использовать
If	Если	**Var**	Переменная
Interrupt	Прерывание	**While**	Пока
Interface	Интерфейс	**With**	С
Label	Метка	**Xor**	Исключающее ИЛИ

Группа слов, имеющая смысл, называется *словосочетанием*. В языке программирования словосочетание, состоящее из слов и символов и задающее правило вычисления некоторого значения, называется *выражением*. Минимальная конструкция языка, представляющая законченную мысль, есть *предложение*. Если предложение языка программирования задает полное описание действия, которое необходимо выполнить, то оно называется *оператором*. Предложение, описывающее структуру и организацию данных, называется *описанием*.

1.1.1. Величины в Паскале

Решение задач на ПК – это процесс сбора, обработки и передачи информации. Поэтому задача любой программы состоит в обработке данных. В Паскале данные делятся на константы и переменные. Они определяются идентификаторами (именами).

Константами называются такие данные, которые не изменяются в процессе выполнения программы в отличие от переменных, которые могут менять свои значения. Имя переменной подобно ящичку, который можно заполнить различными значениями, что нельзя сделать с константой. Переменная характеризуется именем, типом (см. 1.2) и значением.

Кроме констант и переменных, существуют так называемые типизированные константы, которые являются как бы промежуточным звеном между переменными и константами (в данном пособии не рассматриваются. Рекомендуется дополнительная литература, например, [3]).

1.1.2. Структура программы

В программе программист записывает последовательность действий, выполняемых над определенными данными с помощью различных операций для реализации заданной цели. Основные требования, предъявляемые к программе:

- точность полученного результата;
- время выполнения;
- объем требуемой памяти.

Максимальный размер программы ограничен. Компилятор позволяет обрабатывать программы, в которых объем данных и генерируемый машинный код не превышают 64 Кбайт каждый. Если объем программы требует большего количества памяти, то необходимо использовать дополнительные средства.

Структура программы:

1. Заголовок, состоящий из зарезервированного слова **`program`** и имени программы. Заголовок несет смысловую нагрузку и может отсутствовать, однако рекомендуется всегда его записывать для быстрого распознавания нужной программы.

2. Раздел описаний, в котором должны быть описаны все идентификаторы, встречающиеся в программе. Он представляет собой:

- список имен подключаемых библиотечных модулей (определяется зарезервированным словом **`uses`**);
- описание меток (**`label`**);
- описание констант (**`const`**);
- определение типов данных (**`type`**);
- описание переменных (**`var`**);

- описание процедур и функций.

Раздел описания начинается соответствующим каждому элементу списка служебным словом (табл. 1), после которого идет последовательность величин одного типа, разделенных запятой. После списка имен ставится двоеточие и указывается тип данных (см. 1.2).

Любой элемент данного списка в программе может отсутствовать.

3. Раздел операторов.

Данный раздел начинается со служебного слова **Begin** и заканчивается служебным словом **End**. В нём задаются действия над объектами программы, введенными в разделе описаний. Операторы, посредством которых эти действия производятся, разделяются точкой с запятой. После последнего слова **End** ставится точка.

Для лучшего восприятия текста программы и поиска ошибок рекомендуется следующая схема:

- зарезервированные слова **program, procedure, function** и т. д. пишутся строчными буквами;
- идентификаторы начинаются с прописных букв;
- операторы записываются строчными буквами;
- логически подчиненные структуры записываются на 1 строку ниже и на 1–2 позиции правее по отношению к более старшим.

1.2. Типы данных

При решении задач выполняется обработка информации различного свойства, например: дробные и целые числа, слова, строки и т. д. Для описания множества допустимых значений величины и совокупности операций, в которых участвует данная величина, используется указание ее типа данных. Тип данных – это множество величин, объединенных определенной совокупностью допустимых операций. Каждый тип имеет свой диапазон значений и специальное зарезервированное слово для описания. Все типы данных можно разделить на две группы: скалярные (простые) и структурированные (составные). Простые типы данных также делятся на стандартные и пользовательские. Стандартные – предлагаются разработчиками Турбо Паскаля, а пользовательские разрабатывают сами программисты.

Представим типы данных в виде схемы:

1. Простые типы:
   - порядковые;
   - целые;
   - логический;
   - символьный;

- перечисляемый;
- интервальный;
- вещественные;
- ссылочный.

2. Структурированные типы:
   - строковый;
   - регулярный;
   - комбинированный;
   - множественный;
   - файловый.
3. Процедурные типы.

В данном учебном пособии будут рассмотрены лишь основные типы данных, используемые наиболее часто. С другими интересующими типами данных можно познакомиться в специальной литературе (например, [3]). Рассмотрим пока лишь простые типы данных, структурированные типы требуют отдельного, более тщательного рассмотрения.

1.2.1. Целочисленные типы данных

Таблица 1.2

Тип	Диапазон	Требуемая память (байт)
Byte	**0 … 255**	**1**
Shorint	**-128 … 127**	**1**
Integer	**-32768 … 32767**	**2**
Word	**0 … 65535**	**2**
Longint	**-2147483648 … 2147483647**	**4**

Значения целых типов могут изображаться в программе 2 способами: в десятичном виде и в шестнадцатеричном. Если число представлено в шестнадцатеричной системе, перед ним без пробела ставится знак $, а цифры старше 9 обозначаются латинскими буквами от A до F. Диапазон изменений таких чисел от $0000 до $FFFF.

1.2.2. Вещественные типы данных

Вещественные (действительные) типы данных представляют собой значения, которые используются в арифметических выражениях и могут быть представлены двумя способами: с фиксированной и с плавающей точкой.

Таблица 1.3

Тип	Диапазон	Мантисса	Требуемая память (байт)
Real	**2.9E-39 … 1.7E38**	**11–2**	**6**
Single	**1.5E-45 … 3.4E38**	**7–8**	**4**
Double	**5.0E-324 … 1.7E308**	**15–16**	**8**
Extended	**1.9E-4951 … 1.1E4932**	**19–20**	**10**
Comp	**-2E+63+1 … 2E+63-1**	**10–20**	**8**

Действительные числа с фиксированной точкой записываются по обычным правилам арифметики, только целая часть от дробной отделяется точкой. Если точка отсутствует, число считается целым. Перед числом может стоять знак «+» или «–». Если знака нет, то число считается положительным.

Числа в форме с плавающей точкой представляются в экспоненциальном виде: **mE+p**, где **m** – мантисса (целое или дробное число), **E** означает 10 в степени, **p** – порядок (целое число).

Например: $5.18E+2 = 5.18 * 10^{2} = 518$;
$10E-03 = 10 * 10^{-3} = 0.01$.

1.2.3. Символьный тип

Литерный (символьный) тип **char** определяется множеством значений кодовой таблицы ПК. Каждому символу приписывается целое число в диапазоне от 0 до 255. Для кодировки используется код ASCII. Например, код символа 'A' при русской раскладке клавиатуры будет равен 192.

Для размещения в памяти переменной литерного типа нужен 1 байт.

1.2.4. Логический тип

Логический (булевский) тип **boolean** определяется двумя значениями: **true** (истина) и **false** (ложь). Он применяется в логических выражениях и выражениях отношения. Для размещения в памяти –1 байт.

1.3. Арифметические операции и стандартные функции

Арифметическим называется выражение, составленное из операндов – величин, над которыми производится операция; скобок и знаков операций. В результате вычисления выражения получается значение определенного типа. Порядок вычисления выражения определяется скобками и старшинством операций. Они делятся на арифметические отношения, логические и др. Операции могут быть унарными и бинарными.

1.3.1. Арифметические операции

Таблица 1.4

Операция	Действие	Тип операндов	Тип результата
Бинарные			
`+`	Сложение	Целый, вещественный	Целый, вещественный
`-`	Вычитание	Целый, вещественный	Целый, вещественный
`*`	Умножение	Целый, вещественный	Целый, вещественный
`/`	Деление	Целый, вещественный	Вещественный
`DIV`	Целочисленное деление	Целый	Целый
`MOD`	Остаток от деления	Целый	Целый
Унарные			
`+`	Сохранение знака	Целый, вещественный	Целый, вещественный
`-`	Отрицание знака	Целый, вещественный	Целый, вещественный

1.3.2. Операции отношения

Операции отношения выполняют сравнение двух операндов и определяют, истинно значение или ложно. Сравниваемые величины могут принадлежать к любому типу данных, и результат всегда имеет логический тип, принимая одно значение из двух: истина или ложь.

Таблица 1.5

Операция	Название	Выражение
`=`	Равно	`A=B`
`<>`	Неравно	`A<>B`
`>`	Больше	`A>B`
`<`	Меньше	`A<B`
`>=`	Больше или равно	`A>=B`
`<=`	Меньше или равно	`A<=B`

1.3.3. Стандартные математические функции

Таблица 1.6

Обращение	Тип аргумента	Тип результата	Функция
Abs(x)	Целый, вещественный	Целый, вещественный	Модуль аргумента
Arctan(x)	Целый, вещественный	Вещественный	Арктангенс
Cos(x)	Целый, вещественный	Вещественный	Косинус
Exp(x)	Целый, вещественный	Вещественный	e^x – экпонента
Frac(x)	Целый, вещественный	Вещественный	Дробная часть x
Int(x)	Целый, вещественный	Вещественный	Целая часть x
Ln(x)	Целый, вещественный	Вещественный	Натуральный логарифм
Random		Вещественный	Псевдослучайное число [0,1]
Random(x)	Целый	Целый	Псевдослучайное число [0,x]
Round(x)	Вещественный	Целый	Округление до ближайшего целого
Sin(x)	Целый, вещественный	Вещественный	Синус
Sqr(x)	Целый, вещественный	Вещественный	Квадрат x
Sqrt(x)	Целый, вещественный	Вещественный	Корень квадратный из x
Trunc(x)	Вещественный	Целый	Ближайшее целое, не превышающее x по модулю

1.3.4. Логические операции

Логические выражения в результате вычисления принимают логические значения **True** и **False**. Операндами этих выражений могут быть логические константы, переменные, отношения. Идентификатор логического типа в Паскале: **boolean**.

В Паскале имеется 4 логические операции: отрицание – **NOT**, логическое умножение – **AND**, логическое сложение – **OR**, исключающее «или» – **XOR** . Использованы обозначения: T – **true**, F – **false**.

Таблица 1.7

A	B	Not A	A and B	A or B	A xor B
T	T	F	T	T	F
T	F	F	F	T	T
F	F	T	F	F	F
F	T	T	F	T	T

Приоритеты операций: **not, and, or, xor**. Операции отношения (**=** , **<>** ...) имеют более высокий приоритет, чем логические операции, поэтому их следует заключать в скобки при использовании по отношению к ним логических операций.

1.3.5. Приоритет операций (в порядке убывания):

- вычисление функции;
- унарный минус, **not**;
- умножение, деление, **div, mod, and**;
- сложение, вычитание, **or, xor**;
- операции отношения.

2. ОПЕРАТОРЫ ЯЗЫКА ПАСКАЛЬ

Оператором называется предложение языка программирования, задающее полное описание некоторого действия, которое необходимо выполнить. Основная часть программы на языке Турбо Паскаль представляет собой последовательность операторов. Разделителем операторов служит точка с запятой. Операторы, не содержащие других операторов, называются *простыми*. К ним относятся операторы присваивания, безусловного перехода, вызова процедуры, пустой. Структурные операторы представляют собой конструкции, построенные из других операторов по строго определенным правилам. Эти операторы можно разделить на три группы: составные, условные и повтора.

2.1. Оператор присваивания

Оператор присваивания – один из наиболее часто встречающихся операторов языка. Он имеет следующую структуру:

```
переменная := выражение;
```

`:=` – знак присваивания. Слева от знака присваивания может стоять только переменная. Справа от знака присваивания стоит выражение. Частным случаем выражения может служить переменная, константа или вызов функции. Тип выражения должен соответствовать типу переменной.

Наиболее распространен случай арифметического выражения. Примеры оператора присваивания:

```
a:=4.5*b-13*c;   x:=y-sin(P/4)+8.1;
A:=C;            x:=0;
```

Менее распространен случай логического выражения. Примеры оператора присваивания:

```
L:=x<2;          Z:=true;
```

Работает оператор следующим образом. Сначала вычисляется значение выражения, а затем это значение присваивается переменной, стоящей слева от знака присваивания.

2.2. Ввод и вывод данных

Решение даже самой простой задачи на компьютере не обходится без операций ввода – вывода информации. Ввод данных – это передача информации от внешнего носителя в оперативную память для обработки. Вывод – обратный процесс, когда данные передаются после обработки из оперативной памяти на внешний носитель (экран монитора, принтер, дискету или винчестер и другие устройства). Выполнение этих операций производится путем обращения к стандартным процедурам: **`Read, Readln, Write, Writeln.`**

2.2.1. Вывод данных на экран

Процедура вывода **Write** производит вывод данных.

Общий вид: **Write(<список вывода>);**

В списке вывода могут быть представлены выражения допустимых типов данных (**integer, real, char** и т. д.) и произвольный текст, заключенный в апострофы.

Например: **Write('Привет'); Write(34.7);**
Write(45+55); Write(b, d);

Если в качестве параметра в списке вывода стоит строковая константа в апострофах, то она выводится в том же виде. Если в качестве параметра в списке вывода стоит переменная, то выводится её значение. Если в качестве параметра в списке вывода стоит выражение, то сначала вычисляется его значение, а затем оно выводится на экран.

Процедура **Writeln** аналогична процедуре **Write**. Отличие в том, что после вывода последнего в списке выражения курсор переходит на начало новой строки.

В процедурах вывода **Write** и **Writeln** имеется возможность записи выражения, определяющего ширину поля вывода.

При рассмотрении форматов вывода примем следующие обозначения:

- **I** – целочисленное выражение;
- **R** – выражение вещественного типа;
- **_** – пробел.

Таблица 2.1

Значение I	**Выражение**	**Результат**
324 34 324 312	Write (I); Write (I,I,I); Write (I:6); Write (I+I:7);	324 343434 ___324 ____624
Значение R	**Выражение**	**Результат**
123.432 -1.34E+01 304.55	Write (R); Write (R); Write (R:15);	1.2343200000E+02 -1.3400000000E+01 3.045500000E+02
Значение R	**Выражение**	**Результат**
304.66 45.322	Write (R:9:4); Write (R:7:2);	_304.6600 __45.32

Оператор **Writeln;** без параметров просто переводит курсор на новую строку, ничего не выводя.

Пример 2.1. Использование операторов вывода.

```
program primer;
  var a,b,c,sum:integer;
begin
a:=4; b:=6; c:=55;
Write(a:3); Write(b:3); Write(c:3); Writeln;
Sum:=a+b+c;
Writeln ('A=',a);
Writeln ('B=',b);
Writeln ('C=',c);
Writeln ('Сумма A+B+C равна ', sum);
end.
```

Результат выполнения:

```
  4  6 55
A=4
B=6
C=55
Сумма A+B+C равна 65
```

2.2.2. Ввод данных с клавиатуры

Процедура чтения **Read** обеспечивает ввод данных для последующей их обработки программой. Чаще эту процедуру называют *оператором ввода*.

Общий вид: **Read (<список переменных>)**;

В списке перечисляются имена переменных. Значения этих переменных набираются через пробел на клавиатуре и высвечиваются на экране после запуска программы. После набора данных для одной процедуры **Read** нажимается клавиша ввода **Enter**. Значения переменных должны вводиться в строгом соответствии с синтаксисом языка Паскаль. Если соответствие нарушено, то возникают ошибки.

Процедура чтения **Readln** аналогична процедуре **Read**, единственное отличие в том, что после считывания последнего в списке значения курсор переходит на начало новой строки.

Пример 2.2. Использование операторов ввода.

```
program primer;
    var   i, k:integer;  c,d, s: real;
begin
readln (c,d);
read(i,k);
...
end.
```

В данном случае необходимо ввести сначала два действительных числа через пробел. Переменной **c** присваивается значение, равное первому введенному числу, а переменной **d** – значение, равное второму введенному числу. После ввода этих значений курсор переходит на начало новой строки (за это отвечает **ln**, следующий за словом **Read**). Далее требуется ввести еще два целых числа, которые будут присвоены значениям переменных **i** и **k** соответственно.

Оператор ввода работает следующим образом. После запуска программы, когда её выполнение доходит до оператора ввода, программа приостанавливает свою работу и ждёт от вас ввода значений переменных с клавиатуры. При этом на экране мерцает курсор. После ввода требуемых значений программа продолжает работу дальше, выполняя последующие операторы.

Как вы могли заметить, такая организация работы не совсем удобна, т. к. нужно помнить, значения каких переменных нужно ввести. При большом их количестве легко запутаться.

Гораздо удобнее для ввода значений переменных организовать диалог с компьютером. Для этого непосредственно перед каждым оператором ввода переменной нужно поставить оператор вывода **Write** для вывода на экран поясняющего текста:

```
Write ('X='); Readln(X);
Write ('Z='); Readln(Z);
```

или

```
Write ('Введите значение A > '); Readln(A);
```

Предпочтительно использовать именно такой ввод переменных с клавиатуры.

Пример 2.3. Организация ввода переменных.

```
program primer;
    var   X,Z,Sum: real;
begin
Write ('X='); Readln(X);
Write ('Z='); Readln(Z);
Sum:=X+Z;
Writeln ('Сумма равна ', Sum:10:2);
end.
```

2.3. Оператор безусловного перехода

Оператор безусловного перехода **goto** означает «перейти к» и применяется в случаях, когда после выполнения некоторого оператора надо выполнить не следующий по порядку, а какой-либо другой, отмеченный меткой, оператор.

Общий вид: **goto <метка>;**

Метка объявляется в разделе описания меток и состоит из имени и следующего за ним двоеточия. Имя метки может содержать цифровые и буквенные символы, максимальная длина имени ограничена 127 знаками. Раздел описания меток начинается зарезервированным словом Label, за которым следует имя метки.

Пример 2.4. Использование меток.

```
program primer;
     label 999, metka;
begin
…
goto 999;
…
999: write (‘ Имя’);
…
goto metka;
…
metka: write(‘Фамилия’);
…
end.
```

Использование безусловных передач управления в программе считается теоретически избыточным и подвергается критике, т. к. способствует созданию малопонятных и трудномодифицируемых программ, которые вызывают сложности при отладке. Поэтому рекомендуется минимальное использование оператора безусловного перехода с соблюдением следующих правил:

- следует стремиться применять операторы перехода для передачи управления только вниз (вперед) по тексту программы;
- расстояние между меткой и оператором перехода на нее не должно превышать одной страницы текста (или высоты экрана дисплея).

2.4. Пустой оператор

Пустой оператор не содержит никаких символов и не выполняет никаких действий. Используется для организации перехода к концу блока в случаях, если необходимо пропустить несколько операторов, но не выходить из блока. Для этого перед зарезервированным словом **end** ставятся метка и двоеточие.

2.5. Структурные операторы

Структурные операторы представляют собой конструкции, построенные из других операторов по строгим правилам. Их можно разделить на три группы: составные, условные и повтора. Применение структурных операторов в вашей программе очень часто просто незаменимо, потому что они позволяют программисту сделать его программу зависимой от каких-либо условий, например введенных пользователем. К тому же, применяя операторы повтора, вы получаете возможность обрабатывать большие объемы данных за сравнительно малый отрезок времени.

2.6. Составной оператор

Этот оператор представляет собой совокупность произвольного числа операторов, отделенных друг от друга точкой с запятой, и ограниченную операторными скобками **begin** и **end**. Он воспринимается как единое целое и может находиться в любом месте программы, где возможно наличие оператора.

Иными словами, составной оператор позволяет объединить несколько операторов в один.

2.7. Условные операторы

Условные операторы предназначены для выбора к исполнению одного из возможных действий, в зависимости от некоторого условия (при этом одно из действий может отсутствовать). Для программирования ветвящихся алгоритмов в Турбо Паскале есть специальные операторы.

2.7.1. Условный оператор If

Оператор If – это одно из самых популярных средств, изменяющих порядок выполнения операторов программы.

Он может принимать одну из форм:

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

или

```
If <условие> then <оператор>;
```

Оператор выполняется следующим образом. Сначала вычисляется выражение, записанное в условии. В результате его вычисления получается значение логического (булевского) типа. Если это значение – «истина», то выполняется **оператор1**, указанный после слова **then**. Если же в результате имеем «ложь», то выполняется **оператор2**. В случае

если вместо **оператора1** или **оператора2** следует серия операторов, то эту серию операторов необходимо заключить в операторные скобки **begin** … **end**.

Обратить внимание, что перед словом **else** точка с запятой не ставится.

Пример 2.5. Составить программу, которая запрашивает возраст ребенка и затем выдает решение о приеме ребенка в школу (возраст от 7 до 16 лет).

Решение:

```
program sh;
     var v: integer;
begin
Write('Введите возраст ребенка');
Readln(v);
if (v>=7)and(v<=16)
              then writeln('Принимаем в школу')
               else writeln ('Не принимаем в школу');
end.
```

Пример 2.6. Даны два числа. Меньшее из этих чисел заменить суммой данных чисел, большее – произведением.

```
program sh;
     var x, y,s,p: integer;
begin
Write('Введите 2 числа');
Readln(x, y);
s:=x+y; p:=x*y;
if x>=y then begin y:=s; x:=p; end
         else begin x:=s; y:=p; end;
writeln('x=', x);
writeln('y=', y);
end.
```

Если оператор **if** обеспечивает выбор из двух альтернатив, то существует оператор, который позволяет сделать выбор из произвольного числа вариантов.

2.7.2. Оператор выбора

Он организует переход на один из нескольких вариантов действий в зависимости от значения выражения, называемого *селектором*.

Общий вид:

```
case k of
<const1>: <оператор1>;
<const2>: <оператор2>;
...
<constN>: <операторN>
else <операторN+1>
end;
```

Здесь **k** – выражение-селектор, которое может иметь только простой порядковый тип (целый, символьный, логический). **<const1>, ...<constN> –** константы того же типа, что и селектор.

Оператор **case** работает следующим образом. Сначала вычисляется значение выражения-селектора, затем обеспечивается реализация того оператора, константа выбора которого равна текущему значению селектора. Если ни одна из констант не равна значению селектора, то выполняется оператор, стоящий за словом **else**. Если же это слово отсутствует, то активизируется оператор, находящийся за границей **case**, т. е. после слова **end**.

При использовании оператора **case** должны выполняться следующие правила:

1. Выражение-селектор может иметь только простой порядковый тип (целый, символьный, логический).
2. Все константы, которые предшествуют операторам альтернатив, должны иметь тот же тип, что и селектор.
3. Все константы в альтернативах должны быть уникальны в пределах оператора выбора.

Формы записи оператора:

Селектор интервального типа:

```
case I of
 1..10 : writeln('число в диапазоне 1-10');
11..20 : writeln('число в диапазоне 11-20');
else writeln('число вне пределов нужных диапазонов')
end;
```

Селектор целого типа:

```
case I of
1 : y:=I+10;
2 : y:=I+20;
3 : y:=I+30;
end;
```

Пример 2.7. Составить программу, которая по введенному номеру месяца выводит на экран название времени года.

```
program m;
    var k:byte;
begin
write('Введите номер месяца');
readln(k);
case k of
1,2,12 : writeln('Зима');
3,4,5  : writeln('Весна');
6,7,8  : writeln('Лето');
9,10,11: writeln('Осень')
else writeln('Такого месяца нет');
end;
end.
```

2.8. Операторы цикла (повтора)

Если в программе возникает необходимость неоднократного выполнения некоторых операторов, то для этого используются операторы повтора (цикла). В языке Паскаль различают три вида операторов цикла: цикл с предусловием (**while**), цикл с постусловием (**repeat**) и цикл с параметром (**for**).

Если число повторений заранее неизвестно, а задано лишь условие его повторения (или окончания), то используются операторы **while** и **repeat**. Если число требуемых повторений заранее известно, то используется оператор, называемый *оператором цикла с параметром*.

2.8.1. Оператор цикла с параметром

Оператор цикла с параметром имеет два варианта записи:

```
for <имя переменной> := <начальное значение> to
<конечное значение> do <тело цикла>
```

и

```
for <имя переменной> := <начальное значение> downto
<конечное значение> do <тело цикла>
```

Имя переменной – параметр цикла, простая переменная целого типа; **<тело цикла>** – один оператор. Если в теле цикла нужно выполнить несколько операторов, то их нужно заключить в операторные скобки **begin** ... **end**, т. е. применить составной оператор.

Цикл повторяется до тех пор, пока значение параметра лежит в интервале между начальным и конечным значениями. В первом вариан-

те при каждом повторении цикла значение параметра увеличивается на 1, во втором – уменьшается на 1.

При первом обращении к оператору **for** вначале определяются начальное и конечное значения и присваивается параметру цикла начальное значение. После этого циклически повторяются следующие действия:

1. Проверяется условие параметр цикла <= конечному значению.
2. Если условие выполнено, то оператор продолжает работу (выполняется оператор в теле цикла); если условие не выполнено, то оператор завершает работу и управление в программе передается на оператор, следующий за циклом.
3. Значение параметра изменяется (увеличивается на 1 или уменьшается на 1).

Пример 2.8. Вывести на экран таблицу перевода из градусов по шкале Цельсия в градусы по Фаренгейту для значений от 15 °С до 30 °С с шагом в 1 °С. Перевод осуществляется по формуле **F=C*1.8+32** .

```
program zf;
    var i:integer; f:real;
begin
writeln('Температура');
for i:=15 to 30 do
  begin
  f:=i*1.8+32;
  writeln('по Цельсию ',i,' по Фаренгейту', f:5:2);
  end;
end.
```

Пример 2.9. Вывести на экран натуральные числа от 1 до 9 в обратном порядке.

```
program z;
    var i:integer;
begin
for i:=9 downto 1 do
    writeln(i);
end.
```

2.8.2. Оператор цикла с предусловием

Оператор **while** называют *оператором цикла с предусловием*, т. к. проверка условия выполнения цикла производится в самом начале оператора.

Общий вид:

```
while <условие> do <тело цикла>;
```

Условие является условием продолжения повторений. Это логическое выражение.

Тело цикла – простой или составной оператор. Если операторов в теле цикла несколько, то тело цикла заключается в операторные скобки **begin** … **end**.

Перед каждым выполнением тела цикла вычисляется значение выражения условия. Если результат – «истина», тело цикла выполняется и снова вычисляется выражение условия. Если результат – «ложь», происходят выход из цикла и переход к следующему после **while** оператору.

Пример 2.10. Найти сумму 10 произвольных чисел.

```
program z;
     const  n=10;
     var k,x,s: integer;
begin
k:=0; s:=0;  {k- количество введенных чисел}
while k < n do
      begin
      k:=k+1;
      write('Введите число '); readln(x);
      s:=s+x;
      end;
writeln('Сумма чисел равна ', s);
end.
```

2.8.3. Оператор цикла с постусловием

Оператор цикла **repeat** аналогичен оператору **while**, но отличается от него, во-первых, тем, что условие проверяется после очередного выполнения операторов тела цикла и таким образом гарантируется хотя бы однократное выполнение цикла. Во-вторых, тем, что критерием прекращения цикла является выполнение условия. За это данный оператор часто называют *циклом с постусловием*, т. к. он прекращает выполняться, как только условие, записанное после слова **until**, выполнится. Оператор цикла **repeat** состоит из заголовка, тела и условия окончания.

Общий вид:

```
repeat
<оператор>;
…
<оператор>;
until <условие окончания цикла>
```

Вначале выполняется тело цикла, затем проверяется условие выхода из цикла. В любом случае этот цикл выполняется хотя бы один раз. Если условие не выполняется, т. е. результатом выражения является **False**, то цикл активизируется еще раз. Если условие выполнено, то происходит выход из цикла. Использования операторных скобок, в случае если тело цикла состоит из нескольких операторов, не требуется.

Пример 2.11. Составить программу, которая вводит и суммирует целые числа. Если введено значение 999, то на экран выводится результат суммирования.

```
program s;
     var x, s:integer;
begin
s:=0;
repeat
     Write('Ввести число ');  Readln(x);
     if x<>999 then s:=s+x;
until x=999;
Writeln('Сумма введенных чисел ', s);
end.
```

2.8.4. Типовые задачи с использованием циклов

Пример 2.12. Составить программу для вычисления конечной суммы ряда n нечётных чисел:

$$S = 1 + 3 + 5 + 7 + ... + (2 \cdot n - 1)$$

для произвольного значения n, введённого с клавиатуры.

Решение:

```
program summa;
     var i,n,s:integer;
begin
write('n='); readln(n);
s:=0;
for i:=1 to n do
     s:=s+(2*i-1);
writeln('Сумма ряда = ',s);
end.
```

Пример 2.13. Составить программу для вычисления суммы ряда (с точностью до $\varepsilon = 10^{-5}$)

$$S = 1 - 1/2^2 + 1/3^2 - 1/4^2 + ...$$

Полученный результат сравнить с точным $S = \pi^2/12$.

Решение. При решении подобных задач заранее неизвестно число элементов ряда. Поэтому применяются операторы цикла **while** или **repeat**. К переменной, в которой накапливается сумма ряда, добавляется очередной элемент и производится проверка условия. Если предыдущая сумма отличается от последующей менее чем на заданную погрешность вычислений, то можно выходить из цикла и выводить результат. Предыдущая сумма ряда сохраняется в дополнительной переменной **s1** для сравнения. При сравнении разницу берем по модулю, используя функцию **abs( )**.

```
program summa;
    var i,z:integer; x,s,s1:real;
begin
e:=1e-5; {погрешность вычислений}
s:=0; i:=1;
z:=1; {знак элемента}
repeat
    s1:=s;            {сохраняем старую сумму}
    x:=1/(i*i);       {вычисляем очередной элемент}
    s:=s+z*x;    {добавляем к сумме с учётом знака}
    i:=i+1;
    z:=-z;            {меняем знак элемента}
until abs(s-s1)<=e;  {проверяем условие выхода}
writeln('Сумма ряда = ',s);
end.
```

Пример 2.14. Составить программу для вычисления факториала числа *n* (для произвольного значения *n*, введённого с клавиатуры):

$$S = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \ldots \cdot n = n!$$

Решение:

```
program fact;
    var i,n,s:integer;
begin
write('n='); readln(n);
s:=1; {начальное значение для произведения - 1}
for i:=1 to n do
    s:=s*i;
writeln('n!=',s);
end.
```

3. ПРОЦЕДУРЫ И ФУНКЦИИ

Процедуры и функции в общем случае называются *подпрограммами* и применяются для упрощения структуры программы.

3.1. Функции

Подпрограмма-функция обрабатывает данные, переданные ей из главной программы, и затем возвращает полученный результат (в отличие от процедуры). В языке Паскаль есть стандартные функции, которые описаны в модулях. Ими можно пользоваться. Для использования некоторых стандартных функций требуется объявление соответствующего модуля в секции **uses** раздела описаний. Наиболее часто используемые математические функции приведены в табл. 1.6.

Часто возникает необходимость описать свою функцию и многократно использовать её в программе. Описываются функции в разделе описаний программы, а используются – в основной части.

Функция, определенная пользователем, состоит из заголовка, своего раздела описаний и тела функции. Заголовок содержит зарезервированное слово **function**, имя, список формальных параметров (заключенный в скобки) и тип возвращаемого функцией значения. Раздел описаний аналогичен разделу описаний программы и может содержать те же секции, включая описание собственных функций и процедур. Раздел описаний может и отсутствовать, если в нём нет необходимости. Тело функции представляет собой локальный блок, по структуре сходный с программой.

Общий вид описания функции:

```
function <имя> (<параметры>): <тип результата>;
const … ;
 …          {раздел описания}
var  … ;
begin
<операторы>;
end;
```

В скобках после имени функции описываются формальные параметры. Параметры одного типа можно перечислять через запятую, затем ставится двоеточие и указывается их тип. Далее через точку с запятой могут описываться параметры других типов. Порядок следования параметров имеет значение!

В собственном разделе описания могут быть описаны локальные константы, переменные и т. д. Область их действия ограничена локальным блоком функции.

В разделе операторов должен находиться хотя бы один оператор, присваивающий имени функции значение. Обращение к функции осуществляется по имени с указанием списка аргументов. Каждый аргумент должен соответствовать формальным параметрам и иметь тот же тип. Механизм передачи параметров будет подробнее описан ниже.

Описанную функцию можно использовать в основной части программы. Функция используется в выражениях по имени с фактическими параметрами.

Пример 3.1. Найти значение следующего выражения:

$$\frac{f(t)-f(c)}{f(t+c)},$$

где $f(x)=\sqrt{x}+2\cdot x^2+3\cdot x$.

Решение:

```
program prf;
var t,c,r:real;
     function F(x:real):real;
     begin
     F:=sqrt(x)+2*sqr(x)+3*x;
     end;
begin
Writeln('Введите числа');
Write('t='); Readln(t);
Write('c='); Readln(c);
r:=(F(t)-F(c))/(F(t+c));
writeln('результат = ', r:8:3);
end.
```

3.2. Примеры полезных функций

Часто возникает необходимость описать математическую функцию, которой нет в списке стандартных функций. Например, в Паскале нет операции возведения в произвольную степень или отсутствует функция тангенса и факториала числа. Приведу несколько полезных функций.

Пример 3.2. Описание функции тангенса:

```
function tg(x:real):real;
begin
     tg:=sin(x)/cos(x);
end;
```

Пример 3.3. Описание функции возведения любого числа в натуральную степень x^n :

```
function stn(x:real; n:integer):real;
    var i,n:integer; s:real;
begin
s:=1;
for i:=1 to n do
    s:=s*x;
stn:=s;
end;
```

Пример 3.4. Описание функции возведения любого числа в любую вещественную степень x^y.

Для решения этой задачи воспользуемся свойством натурального логарифма $\ln(x^y) = y \cdot \ln(x)$. Тогда $x^y = e^{y \cdot \ln(x)}$.

```
function st(x,y:real):real;
begin
stn:=exp(y*Ln(x));
end;
```

Описанные выше функции теперь могут быть использованы в основной части программы, например, в операторе присваивания:

```
z:=(3*tg(x/pi)+4.3*st(r,8.65))/(12*stn(a,5));
```

Данный оператор вычислит значение выражения $\dfrac{3 \cdot \mathrm{tg}(x / pi) + 4{,}3 \cdot r^{8{,}65}}{12 \cdot a^5}$.

3.3. Процедуры

Для использования подпрограммы-процедуры необходимо сначала описать процедуру, а затем обращаться к ней (обращение к процедуре – отдельный оператор). Описание процедуры включает заголовок, свой раздел описаний и тело процедуры. Заголовок состоит из зарезервированного слова **procedure**, имени процедуры и заключенного в скобки списка формальных параметров с указанием типа. Название «формальные» эти параметры получили в связи с тем, что в этом списке заданы только имена для обозначения исходных данных и результатов работы процедуры, а при вызове подпрограммы на их место будут поставлены конкретные значения. Допускается отсутствие параметров, если в них нет необходимости. Тело процедуры – блок, по структуре аналогичный программе.

При создании программ, использующих процедуры, следует учитывать, что все объекты, которые описываются после заголовка в собственном разделе описаний, называются *локальными объектами* и доступны только в пределах этой процедуры.

Все объекты, описанные в вызывающей программе, называются *глобальными* и являются доступными внутри процедур, вызываемых этой программой.

Общий вид описания процедуры:

```
procedure <имя> (список формальных параметров);
const … ;
  …     {раздел описания}
var … ;
begin
<операторы>;
end;
```

3.4. Оператор вызова процедуры

Оператор вызова процедуры служит для активизации стандартной процедуры или процедуры, определенной пользователем. Стандартные процедуры находятся в файлах – модулях. Для их использования достаточно указать имя процедуры и, если необходимо, дополнительные параметры. Для того чтобы вызвать свою процедуру, её надо описать перед началом программы в разделе описаний программы, а затем уже использовать.

Например, в рассматриваемом ниже примере будут использованы стандартные процедуры **ClrScr**, **Gotoxy** и **Textcolor**, описанные в модуле **crt**, а также будет описана своя процедура **name**.

Пример 3.4. Вывести по четырем углам экрана свое имя цветными буквами с эффектом мерцания.

```
program names;
{объявление использования процедур модуля crt}
uses crt;
     procedure name(x,y,c:byte);
     begin
     Gotoxy(x,y); {позиционирование курсора}
     Textcolor(c+16); {установка цвета с и мерцания}
     Write('Иван');
     end;
begin
Clrscr; {очистка экрана}
name(2,2,14); {вызов процедуры}
name(2,22,8);
name(75,2,3);
name(75,22,5);
end.
```

3.5. Механизм передачи параметров в подпрограммах

Как вы уже заметили, параметры в процедуры и функции передаются согласно порядку следования.

При описании процедур и функций в скобках после имени подпрограммы описываются формальные параметры. Параметры одного типа можно перечислять через запятую, затем ставится двоеточие и указывается их тип. Далее через точку с запятой могут описываться параметры других типов. При описании некоторых параметров, через которые планируется передать значение из подпрограммы в основную программу, перед описанием этого параметра ставят ключевое слово **var**. Это обозначает, что при вызове подпрограммы в качестве фактического параметра здесь обязательно должна подставляться переменная соответствующего типа. Эта переменная получит результирующее значение из подпрограммы в результате её выполнения. Применение таких параметров особенно важно для процедур, т. к., в отличие от функций, процедуры не возвращают результирующее значение.

При вызове подпрограммы в скобках указываются фактические параметры. Именно они подставляются в подпрограмму для проведения необходимых действий. При этом соблюдаются следующие правила:

1. Число фактических параметров должно быть равным числу формальных параметров.
2. Формальные параметры замещаются фактическими согласно порядку следования.
3. Тип каждого фактического параметра должен соответствовать типу своего описанного формального параметра.
4. Если формальный параметр описан с ключевым словом **var**, то в качестве соответствующего фактического может стоять только переменная. В противном случае в качестве фактических параметров могут быть переменные, константы и выражения.
5. Если в качестве фактического параметра стоит выражение, то при вызове подпрограммы сначала вычисляется значение этого выражения, а затем это значение передаётся в подпрограмму соответствующему формальному параметру.
6. Имена переменных в фактических и формальных параметрах могут быть одинаковыми или нет. При этом в любом случае для машины это разные переменные, т. к. они имеют разные области действия.

3.6. Стандартные библиотечные модули

В систему Турбо Паскаль версии 6.0 и старше включены 8 модулей: **System, Crt, Dos, Graph, Graph3, Overlay, Printer, Turbo3** и специализированная библиотека Turbo Vision.

Модуль **System** подключается по умолчанию, поэтому в любой программе становятся доступными все его встроенные процедуры и функции (см. табл. 1.6). Остальные модули должны подключаться с помощью зарезервированного слова **uses** с добавлением имени модуля. Например: **uses crt;**.

Рассмотрим кратко назначение каждого модуля:

- **System** – сердце Турбо Паскаля. Подпрограммы, содержащиеся в нем, обеспечивают работу всех остальных модулей системы.
- **Crt** – содержит средства управления дисплеем и клавиатурой компьютера.
- **Dos** – включает средства, позволяющие реализовывать различные функции Dos.
- **Graph3** – поддерживает использование стандартных графических подпрограмм.
- **Overlay** – содержит средства организации специальных оверлейных программ.
- **Printer** – обеспечивает быстрый доступ к принтеру.
- **Turbo3** – обеспечивает максимальную совместимость с версией Турбо Паскаль 3.0.
- **Graph** – содержит пакет графических средств.
- **Turbo Vision** – библиотека объектно-ориентированных программ для разработки пользовательских интерфейсов.

На примере модуля **Crt** познакомимся поближе с работой встроенных процедур и функций. Он устанавливает режим работы адаптера дисплея, организует вывод в буфер экрана, регулирует яркость свечения символов и т. д. С момента подключения пользователю доступны все содержащиеся в нем стандартные средства. Рассмотрим некоторые из них.

TextMode(Mode:integer); – Установка текстового режима. Значение **Mode** равно 1 (40 символов / 25 строк) или 3(80 / 25).

ClrScr; – полностью очищает экран;

ClrEol; – стирает все символы в строке, начиная с текущей позиции до конца строки;

GotoXY(x,y); – перемещает курсор в позицию, заданную координатами x, y.

TextColor(Color:byte); – установка цвета выводимых символов;

TextBackGround(Color); – установка цвета фона.

Чтобы добавить при выводе эффект мерцания, при установке цвета указывается **Blink** (16). Смотрите пример 3.4.

Подробное описание модулей и описанных в них процедур и функций смотрите в [3].

4. РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

При решении нелинейных уравнений невозможно выразить переменную. В этих случаях целесообразно применить ряд численных методов нахождения корней уравнения. Ниже рассмотрены некоторые из них.

Любое уравнение можно представить в виде $f(x) = 0$, перенеся всё в одну сторону, тогда поиск корней уравнения сводится к поиску точек пересечения функции $f(x)$ с осью абсцисс. Для более удобной реализации методов в языке Паскаль целесообразно сразу описать функцию $f(x)$ как подпрограмму:

```
function F(x:real):real;
begin
F:=  .... ;
end;
```

Существует ряд методов численного решения нелинейных уравнений, целесообразность применения каждого из которых определяется видом уравнения, его порядком, требуемой точностью и т. д. Эти методы подробно рассмотрены в [1,2,5].

4.1. Метод отделения корней

Итак, дано уравнение $f(x) = 0$, где $f(x)$ – непрерывная функция. Поиск корней уравнения сводится к поиску точек пересечения функции $f(x)$ с осью абсцисс. Все рассматриваемые ниже методы подразумевают, что уже найден отрезок $[a,b]$, в котором существует один корень уравнения. В зависимости от вида функции таких отрезков может быть несколько, а для периодических функций – бесконечное множество. Метод отделения корней осуществляет поиск таких отрезков.

Наиболее наглядным является графический способ отделения корней. Для реализации этого метода необходимо построить график функции. Это будет легко сделать, если составить программу, которая будет выдавать таблицу значений функции при меняющемся с некоторым шагом h аргументе x (см. рис. 4.1).

Если есть такая таблица значений функции, то график функции можно и не строить. Достаточно найти две строчки, где значение функции меняет знак на противоположный. Такой способ называется табличным методом отделения корней (см. пример 4.1).

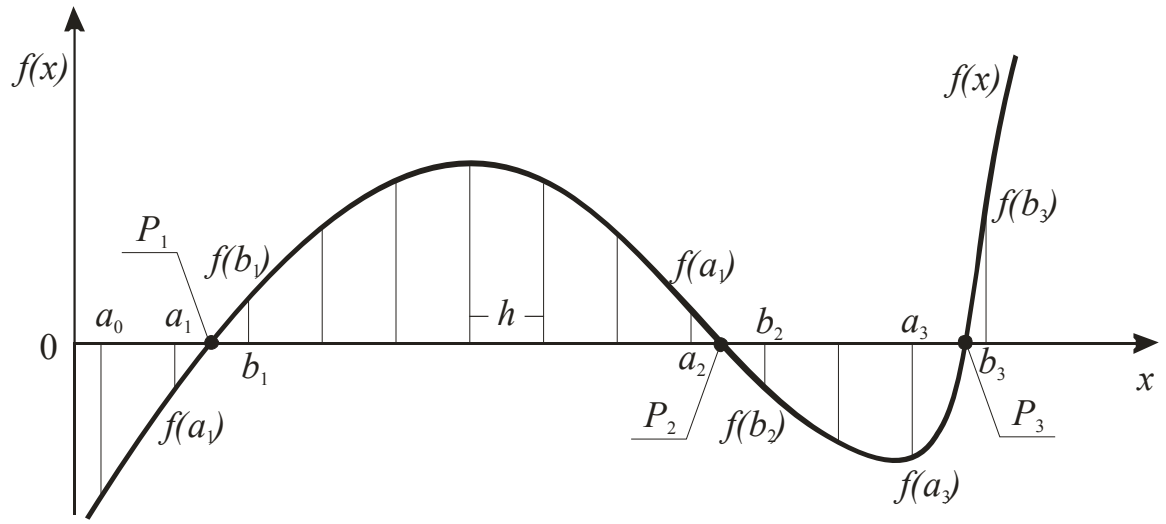

Рис. 4.1. Графическая интерпретация метода отделения корней

Пример. 4.1. Реализация табличного метода отделения корней.

Здесь x пробегает значения от x_n до x_k с шагом h и при этом на экран выводятся значения x и $f(x)$. Отрезок $[x_n, x_k]$ и шаг h нужно подбирать для каждой функции, исходя из её характера. Шаг должен быть меньше, чем расстояние между корнями уравнения, чтобы исключить попадание в шаг двух корней.

```
program tablica;
     var xn,xk,x,h:real;
function F(x:real):real; {описание функции}
     begin F:=sqrt(x)+2*sqr(x)+3*x; end;
begin
write('Введите начало интервала > '); readln(xn);
write('Введите конец интервала > '); readln(xk);
write('Введите шаг > '); readln(h);
x:=xn;
while x<xk do
     begin
     writeln('x=',x,' f(x)=',f(x));
     x:=x+h;
     end;
end.
```

Процесс выбора отрезка $[a,b]$, содержащего корень, можно автоматизировать. Для этого нужно, начиная с какого-то начального значения, смещать отрезок длиной h в цикле и каждый раз анализировать значения функции на концах этого отрезка. Корень присутствует, если эти значения разного знака. Можно использовать сложное условие `(f(a)>0 AND f(b)<0) OR (f(a)<0 AND f(b)>0)`,

а можно просто найти произведение и проверить его знак. Корень присутствует, если **f(a)*f(b)<0**. После этого можно уточнять значение корня, а можно перейти к поиску следующего отрезка, задав начало поиска от конца найденного.

Пример. 4.2. Поиск ближайшего отрезка, содержащего корень.

```
program poisk;
    var a,b,h:real;
function F(x:real):real; {описание функции}
    begin F:=sqrt(x)+2*sqr(x)+3*x; end;
begin
write('Введите начало поиска > '); readln(b);
write('Введите начальное шаг > '); readln(h);
repeat
    a:=b;
    b:=a+h;
until f(a)*f(b)<0;
writeln('a=',a,' b=',b);
end.
```

4.2. Метод половинного деления

Пусть дано уравнение $f(x) = 0$, где $f(x)$ – непрерывная функция, корень P отделен на отрезке $[a,b]$, т. е. $f(a) \cdot f(b) > 0$, причем $|b - a| < E$. Требуется найти значение корня P с точностью до E (см. рис. 4.2).

Если корень P не отделен на заданном отрезке, т. е. $f(a)$ и $f(b)$ одного знака и, следовательно, $f(a) \cdot f(b) > 0$, то вычисляются значения функции в точках, расположенных через равные интервалы на оси X. Когда $f(a_n)$ и $f(b_n)$ имеют противоположные знаки, то значения $a = a_n$ и $b=b_n$ принимаются в качестве начальных и находят середину отрезка $[a,b]$, т. е. $c=(a+b)/2$. Тогда отрезок $[a,b]$ точкой c разделится на два равных отрезка $[a,c]$ и $[c,b]$, длина которых равна $(b–a)/2$. Из двух этих образовавшихся отрезков выбирается тот, на концах которого функция $f(x)$ принимает значения противоположных знаков; обозначим его $[a_1,b_1]$. Затем отрезок $[a_1,b_1]$ делим пополам и проводим те же действия. Получим отрезок $[a_2,b_2]$, длина которого равна $(b–a)/2^2$. Процесс деления отрезка пополам производится до тех пор, когда на каком-то k-м этапе будет получен отрезок $[a_k,b_k]$, такой, что

$$b_k - a_k = (b-a)/2^k \le E \quad \text{и} \quad a_k \le P \le b_k,$$

где число k указывает на количество проведенных делений. Числа a_k и b_k – корни уравнения $f(x) = 0$ с точностью до E. За приближенное значение корня следует взять $P=(a_k+b_k)/2$, причем погрешность не превысит $(b–a)/2^{k+1}$.

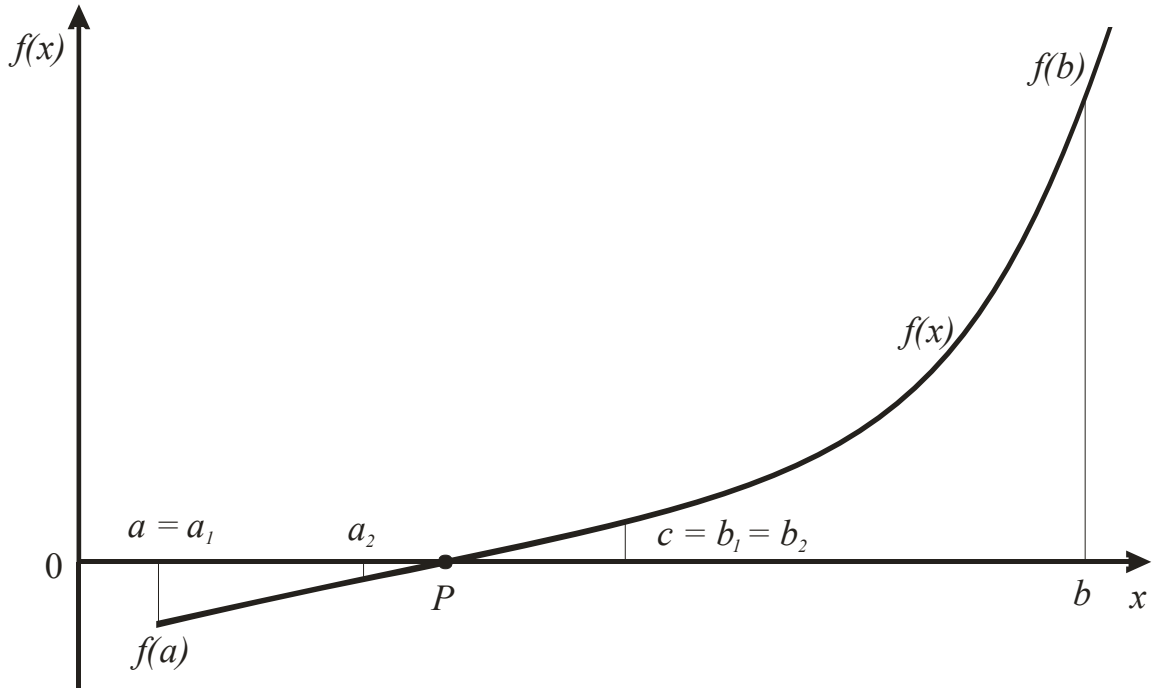

Рис. 4.2. Графическая интерпретация метода половинного деления

Отметим, что в качестве условия прекращения счета более целесообразно пользоваться условием $E \ge |b_k - a_k|$.

Блок-схема алгоритма представлена на рис. 4.3.

Рассмотренный метод имеет относительно малую скорость сходимости, но отличается от других методов простотой реализации алгоритма, не требующего вычисления производных заданной функции.

На блок-схеме видно два цикла. Первый реализует поиск отрезка [*a*,*b*] длиной **h**, на котором есть корень уравнения. Второй цикл уменьшает этот отрезок методом половинного деления до тех пор, пока его длина не станет меньше заданной погрешности **e**. Удобнее всего для организации циклов применить оператор **repeat** … **until**. Внутри второго цикла размещён условный оператор, который проверяет, с какой стороны нужно уменьшить отрезок [*a*,*b*].

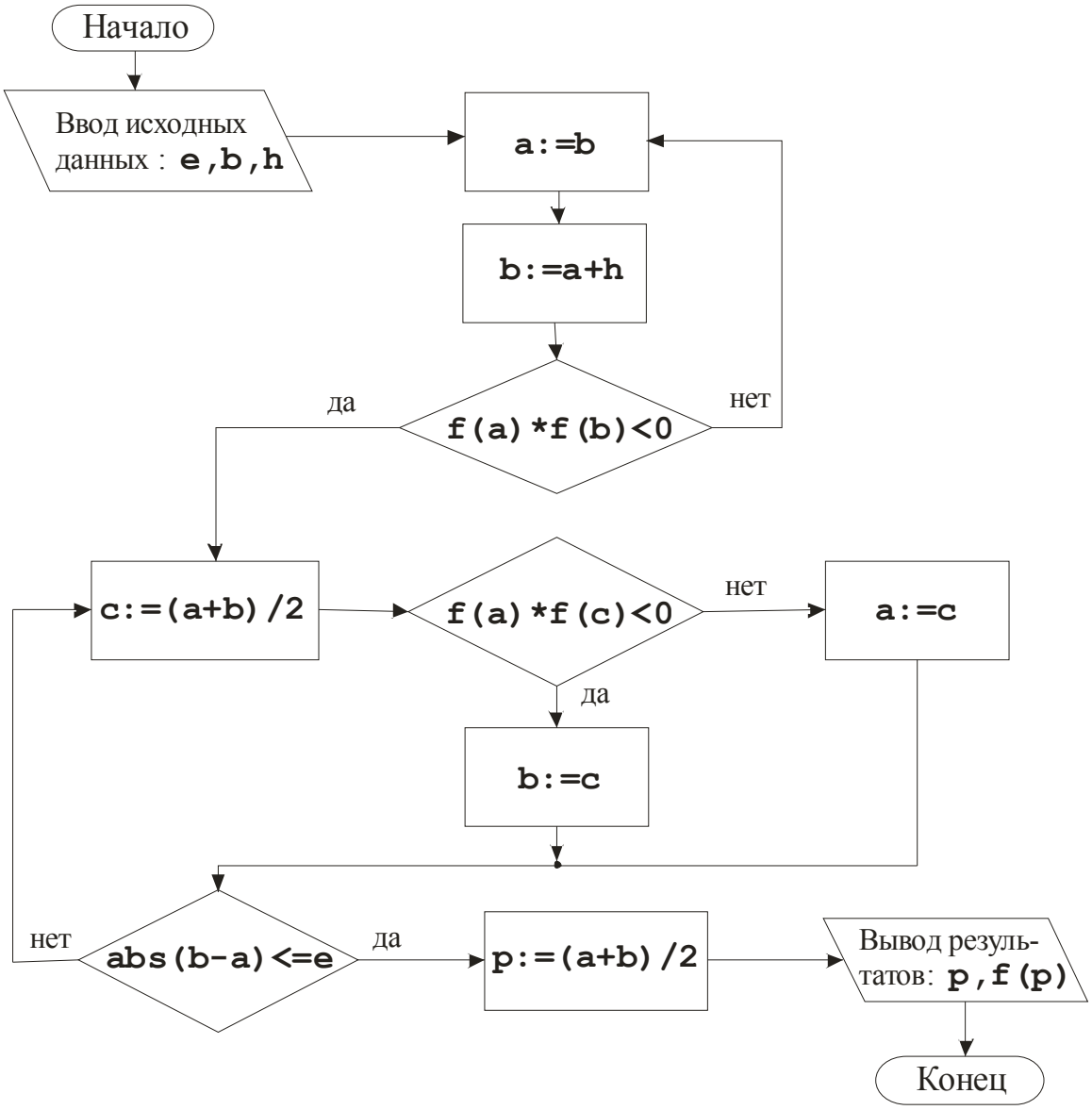

Рис. 4.3. Блок-схема алгоритма метода половинного деления

4.3. Метод касательных

Расчетная формула метода касательных (или метод Ньютона-Рафсона) получается из разложения функции $f(x) = 0$ в ряд Тейлора в окрестности точки x_n. При ограничении разложения двумя членами ряда получим

$$f(x) = f(x_n) + (x–x_n)\cdot f'(x_n) + O(f''(x_n)).$$

Здесь O (от английского order) означает порядок остаточного члена в разложении, который в дальнейшем считается малым.

Из соотношения

$$f(x) \approx f(x_n) + (x - x_n)\cdot f'(x_n) \approx 0$$

получаем

$$x = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Обычно окончательная формула записывается в виде

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Таким образом, зная какое-либо предыдущее приближение x_n, где n – номер приближения или итерации ($n \geq 0$), можно определить последующее приближенное значение корня x_{n+1}. Если заданное (x_n) и расчетное (x_{n+1}) значения совпадают с точностью ε, т. е.

$$|x_{n+1} - x_n| \leq \varepsilon,$$

то значение x_{n+1} считается приближенным значением корня уравнения $f(x) = 0$.

Кроме предыдущего условия окончания счета, можно использовать условие малости функций $f(x)$ около корня, т. е. $|f(x_n)| \leq \varepsilon_f$ или $|f(x_{n+1})| \leq \varepsilon_f$, где ε_f – заданная погрешность.

Рассмотрим геометрическое толкование метода касательных (см. рис. 4.4), где значение корня P определяется следующим образом.

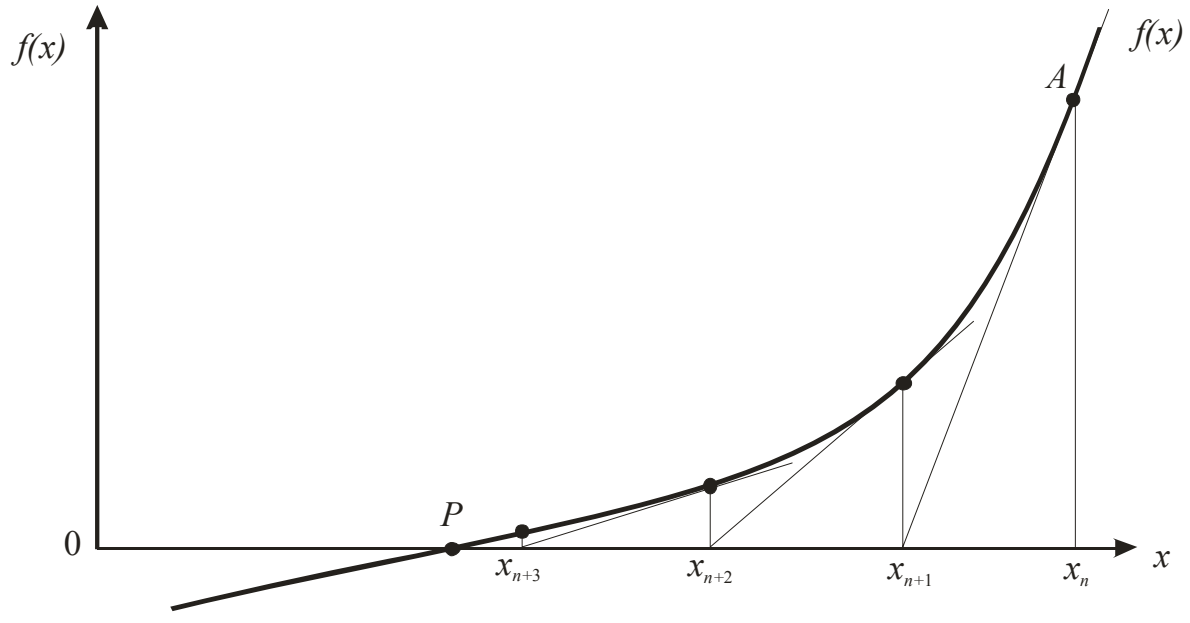

Рис. 4.4. Графическая интерпретация метода касательных

Исходя из некоторого начального приближения x_n, находим соответствующее ему значение $f(x_n)$ (точка A), проводим касательную к кривой $f(x)$ через точку A и ищем точку пересечения этой касательной

с осью X. Эта точка будет значением x_{n+1}, т. к. требовалось провести через точку с координатами x_n, $f(x_n)$ прямую с угловым коэффициентом $f'(x_n)$ и затем найти её пересечение с осью X.

Величина отрезка $(x_n - x_{n+1})$ больше заданной погрешности ε, поэтому поиск значения корня продолжается аналогично. Принимая последнее найденное значение x_{n+1} за исходное, определяем следующее значение x_{n+2} по той же формуле

$$x_{n+2} = x_{n+1} - \frac{f(x_{n+1})}{f'(x_{n+1})},$$

далее опять проверяется условие

$$|x_{n+2} - x_{n+1}| \le \varepsilon.$$

Повторение поиска следующей точки продолжается до тех пор, пока не выполнится условие окончания поиска приближенного значения корня.

Для составления программ можно руководствоваться блок-схемой, представленной на рис. 4.5.

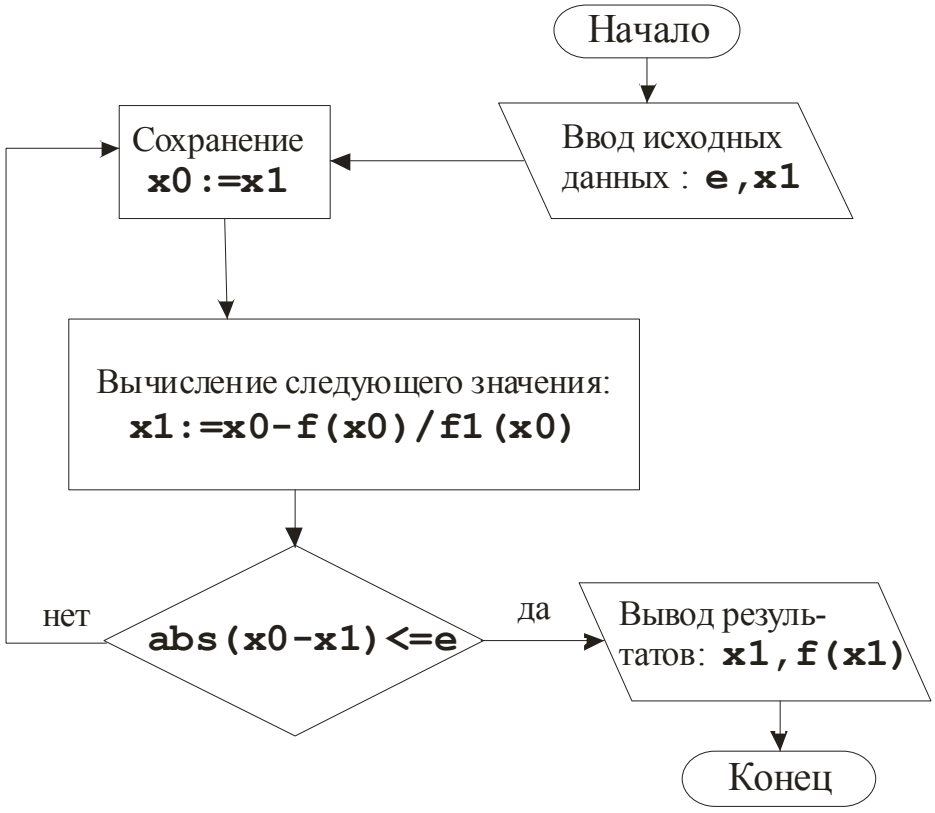

Рис. 4.5. Блок-схема алгоритма метода касательных

Наличие в блок-схеме вывода $f(x_1)$ означает дополнительную проверку правильности определения корня, т. к. в этом случае значение функции должно быть близко к нулю.

Замечание. При реализации этого метода целесообразно функцию $f(x)$ и её производную $f'(x)$ описать как подпрограммы:

```
function F(x:real):real; {Для функции f(x)}
begin   F:=  .... ; end;
function F1(x:real):real;{Для производной f'(x)}
begin   F1:=  .... ; end;
```

При этом функция должна быть предварительно продифференцирована вручную.

Основной цикл удобнее всего организовать при помощи оператора **repeat** … **until**.

Метод касательных обладает относительно большой скоростью сходимости при выполнении следующих условий:

1. Начальное приближение x_0 выбрано достаточно близко к корню уравнения $f(x) = 0$.
2. Вторая производная $f''(x)$ не принимает больших значений.
3. Первая производная $f'(x)$ не слишком близка к нулю.

Важно помнить, что успешность поиска корня напрямую зависит от того, насколько близко к корню выбрано начальное приближение. Поэтому иногда целесообразно вначале провести поиск отрезка, содержащего корень, методом отделения корней, как это было сделано перед реализацией метода половинного деления. Подбирая необходимый шаг, можно легко найти такой отрезок. Тогда в качестве первого приближения можно взять любой из его концов или середину этого отрезка.

4.4. Модифицированный метод Ньютона

Модифицированный метод Ньютона лишь немного отличается от метода касательных и обладает меньшей скоростью сходимости. Здесь значение производной вычисляется всего один раз в точке первого приближения и больше не изменяется. Следовательно, её вычисление будет стоять до оператора цикла. Общая формула вычисления последующего приближения будет выглядеть так:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_0)}.$$

5. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

Задача вычисления интегралов возникает во многих областях прикладной математики. В большинстве случаев не удается найти аналитической формулы, т. е. выразить неопределенный интеграл в виде алгебраических и трансцендентных функций. Даже если аналитическая формула находится, то она получается настолько сложной, что вычислять интеграл с ее помощью труднее, чем другими способами. Распространенными являются также случаи, когда подынтегральная функция задается графиком или таблицей экспериментально полученных значений. В таких ситуациях используют различные методы численного интегрирования, которые основаны на том, что интеграл представляется в виде предела интегральной суммы (суммы площадей), и позволяют определить эту сумму с приемлемой точностью.

Пусть требуется вычислить определенный интеграл

$$I = \int_A^B f(x)dx$$

при условии, что A и B конечны и $f(x)$ является непрерывной функцией x во всем интервале $A \le x \le B$. Значение интеграла I представляет собой площадь, ограниченную кривой $f(x)$, осью x и прямыми $x = A$, $x = B$. Вычисление I проводится путем разбиения интервала от A до B на множество меньших интервалов, приближенным нахождением площади каждой полоски, получающейся при таком разбиении, и дальнейшем суммировании площадей этих полосок.

Суть методов численного интегрирования состоит в замене подынтегральной функции $f(x)$ вспомогательной, интеграл от которой легко вычисляется в элементарных функциях. Обычно $f(x)$ заменяется некоторым интерполяционным многочленом, что приводит к квадратурным формулам:

$$I = \int_A^B f(x)dx = \sum_{i=0}^{n} C_i f(x) + R,$$

где x_i – узлы интерполяции;

i – произвольный номер узла;

C_i – коэффициенты;

R – остаточный член, или погрешность метода.

Неучет (отбрасывание) R приводит к ***погрешности усечения***. К этим погрешностям в процессе вычислений добавляются ***погрешности округления***. Геометрическая интерпретация численного интегрирования представлена на рис. 5.1. и 5.2.

5.1. Методы прямоугольников

Самыми простыми методами численного интегрирования являются методы прямоугольников. При этом непосредственно используется замена определенного интеграла интегральной суммой:

$$I = \int_A^B f(x)dx = \sum_{i=1}^{n} f(z_i)h_i + R \quad , \qquad h_i = x_i - x_{i-1} \quad , \quad x_{i-1} \le z_i \le x_i$$

В качестве точек z_i могут выбираться левые ($z_i = x_{i-1}$) или правые ($z_i = x_i$) границы элементарных отрезков. Обозначая $y_i = f(x_i)$, получим формулы:

- метод левых прямоугольников: $I = h_1 y_0 + h_2 y_1 + ... + h_n y_{n-1} + R$;
- метод правых прямоугольников: $I = h_1 y_1 + h_2 y_2 + ... + h_n y_n + R$;
- более точным является метод средних прямоугольников, использующий значения функции в средних точках элементарных отрезков:

$$I = \sum_{i=1}^{n} h_i \cdot f(x_{i-1/2}) + R;$$

$x_{i-1/2} = 0{,}5(x_{i-1} + x_i) = x_{i-1} + 0{,}5\, h_i, \qquad i = 1, 2, \ldots, n.$

Для частного случая $h_i = h$ = const формулы примут вид

$$I = h \cdot \sum_{i=1}^{n} f(x_{i-1}) + R;$$

$$I = h \cdot \sum_{i=1}^{n} f(x_i) + R;$$

$$I = h \cdot \sum_{i=1}^{n} f(x_{i-1} + 0{,}5 \cdot h) + R, \qquad h = (B - A) / n.$$

Если координату выразить через начальную точку и принять, что $I \approx S$, то получим формулы, готовые для применения в операторе цикла с переменной:

$S = h \cdot \sum_{i=1}^{n} f(a + h \cdot (i-1))$ – для метода левых прямоугольников;

$S = h \cdot \sum_{i=1}^{n} f(a + h \cdot i)$ – для метода правых прямоугольников;

$S = h \cdot \sum_{i=1}^{n} f(a + h \cdot (i - 0{,}5))$ – для метода средних прямоугольников.

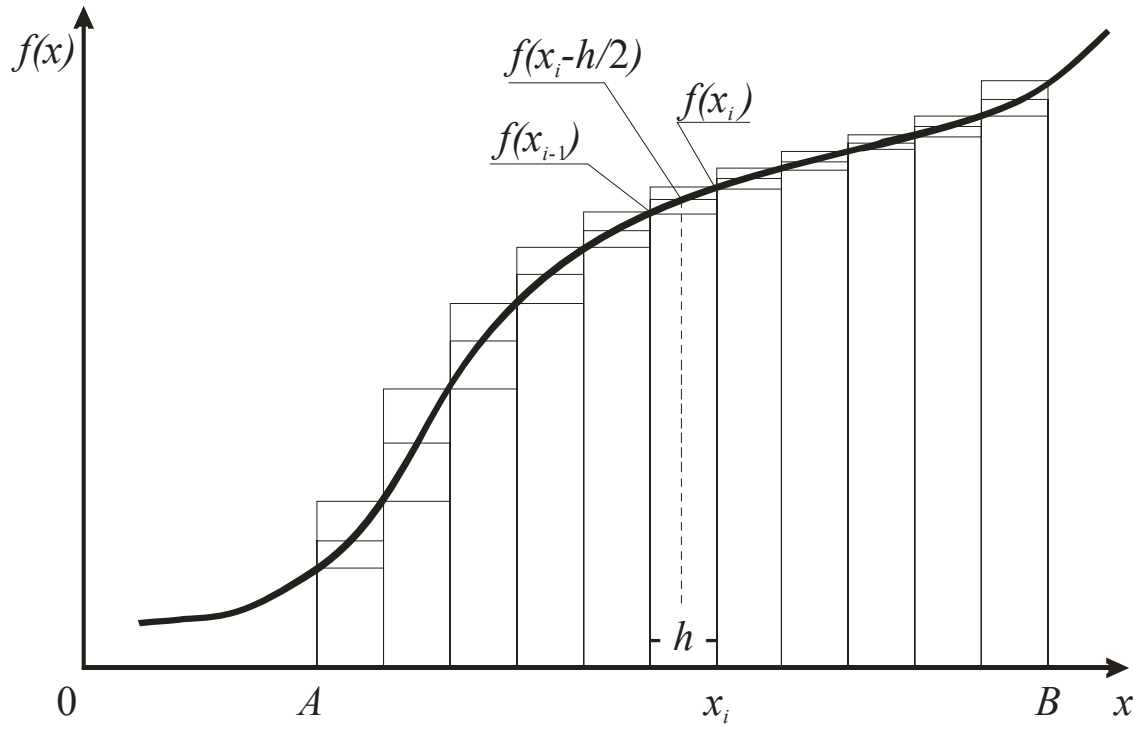

Рис. 5.1. Графическая интерпретация методов прямоугольников

5.2. Метод трапеций

В данном методе $f(x)$ заменяется на линейный интерполяционный многочлен, т. е. на элементарном отрезке $[x_{i\ -1},\ x_i]$ подынтегральная функция представляет собой отрезок прямой линии. Значение I в пределах $[x_{i\ -1},\ x_i]$, равное площади криволинейной фигуры, заменяется площадью прямоугольной трапеции с высотой h_i и основаниями $f(x_{i\ -1}), f(x_i)$:

$$S_i = 0{,}5\ (y_{i\ -1} + y_i)\ h_i, \qquad i = 1, 2, ..., n.$$

После сложения этих соотношений получим формулу трапеций

$$I = 0{,}5 \cdot \sum_{i=1}^{n} h_i \cdot (y_{i-1} + y_i) + R\,.$$

Если шаг интегрирования постоянный ($h_i = h = \text{const}$), то

$$I = h \cdot ((y_0 + y_n)/2 + \sum_{i=1}^{n-1} y_i) + R\,.$$

Если координату выразить через начальную точку и принять, что $I \approx S$, то получим формулу, готовую для применения в операторе цикла с переменной

$$S = 0{,}5 \cdot h \cdot \sum_{i=1}^{n} (f(a + h \cdot (i-1)) + f(a + h \cdot i))\,.$$

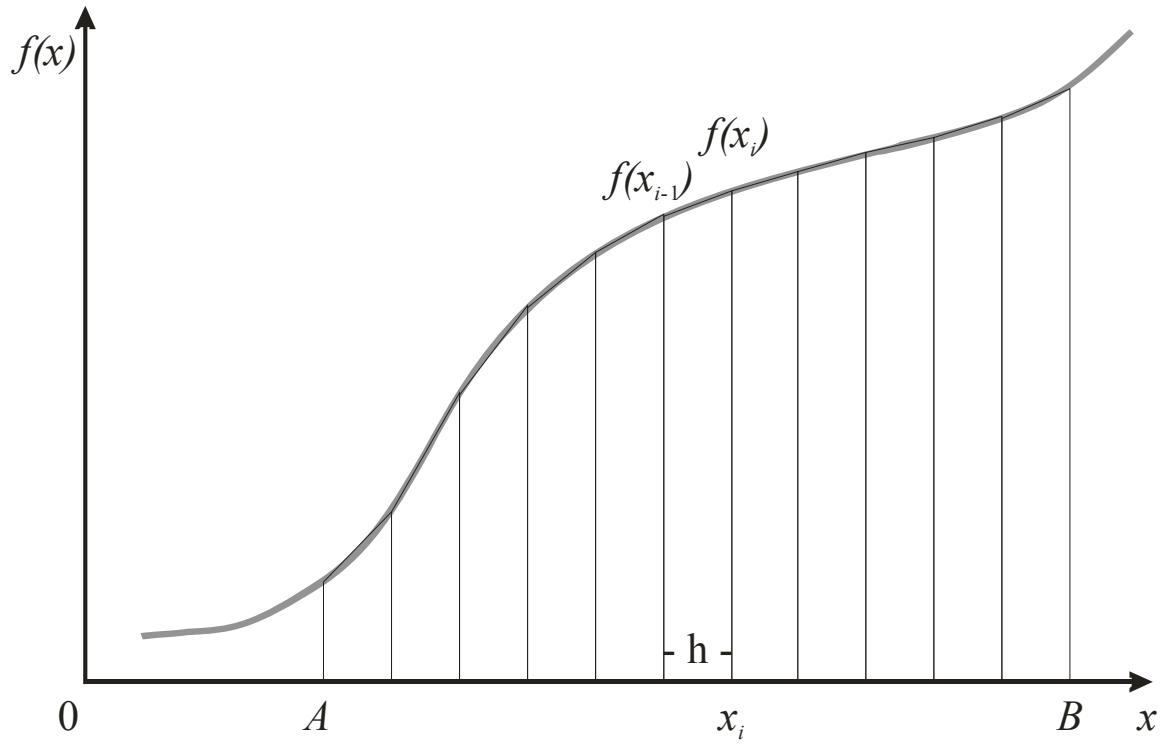

Рис. 5.2. Графическая интерпретация метода трапеций

5.3. Процедура вычисления интеграла

Процедура вычисления интеграла при заданном числе разбиений и границами интегрирования должна иметь входные параметры этих величин и выходной параметр результата. Выходной параметр должен содержать перед собой ключевое слово **var** для передачи параметра по ссылке.

Здесь должен вычисляться шаг в соответствии с числом разбиений.

Сумма может вычисляться при помощи цикла **for** по числу разбиений или при помощи циклов **repeat** или **while** с перемещением текущей точки от начальной до конечной. Например, для метода правых прямоугольников процедура может быть описана так:

```
procedure integral(a,b:real;n:integer; var s:real);
    var i:integer; h:real;
begin
h:=abs(b-a)/n; s:=0;
for i:=1 to n do
    s:=s+h*f(a+h*i);
end;
```

Конечно, перед описанием процедуры необходимо описать подынтегральную функцию $f(x)$. Смотрите описание функций в 3.1.

Замечание! При вычислении интеграла функция должна браться по модулю, т. к. площадь фигуры всегда положительна, независимо от того, что функция может быть и отрицательной. Для упрощения программы модуль можно вставить сразу в описание функции.

5.4. Вычисление интегралов с заданной точностью и оценка методов интегрирования

Оценка погрешности усечения R в формулах численного интегрирования оказывается трудоемкой и малоэффективной из-за трудностей оценки производных высокого порядка подынтегральных функций. Поэтому в практических расчетах для достижения требуемой точности вычислений или допустимой погрешности E используют правило Рунге.

Согласно этому правилу вычисление заданного интеграла проводят для разных интервалов разбиения отрезка $[A,B]$. Так, если начальное число интервалов разбиения есть n и соответствующее ему приближенное значение интеграла – I_n, то для числа интервалов $2n$ получим значение интеграла I_{2n}. Число интервалов можно увеличивать в 2, 3 и т. д. раз по сравнению с базовым значением n. При двукратном увеличении числа отрезков погрешность Δ приближенного значения интеграла для методов прямоугольников и трапеций оценивается как

$$\Delta = | I_n - I_{2n} | / 3.$$

Если $\Delta > E$, то количество интервалов разбиения $[A,B]$ опять увеличивают вдвое, т. е. значение I вычисляют для $4n$. Такое удвоение повторяют до тех пор, пока не выполнится условие $\Delta < E$. Необходимо помнить, что общая погрешность вычислений, равная сумме погрешностей ***усечения*** и ***округления***, сначала с ростом интервалов разбиения уменьшается за счет уменьшения ошибки усечения до некоторого «критического» значения $n_{кр}$, а затем увеличивается из-за увеличения ошибок округления.

Таким образом, не для всякой функции можно получить результат с заданной погрешностью.

- Методы левых и правых прямоугольников практически не применяются из-за их низкой точности.
- В некоторых случаях метод средних дает лучшую точность, чем метод трапеций.

5.5. Основная часть программы

Основная часть программы должна содержать ввод с клавиатуры начального числа разбиений и погрешности вычислений. Для обеспечения заданной точности интеграла процедура его вычисления должна выполняться в цикле. Причём сначала сохраняется старое значение интеграла, затем увеличивается число разбиений и вычисляется новое значение интеграла. Цикл работает пока разница между старым и новым значениями интеграла не станет меньше заданной точности.

При выводе результата полезно также вывести конечное число разбиений, чтобы знать, на сколько частей нужно разбить отрезок для обеспечения заданной точности вычисления интеграла.

6. МАССИВЫ

Все простые типы данных, рассматриваемые ранее, имеют два характерных свойства: неделимость и упорядоченность их значений. Составные, или структурированные типы данных задают множество сложных значений с одним общим именем. Существует несколько методов структурирования, каждый из которых отличается способом обращения к отдельным компонентам. В данном учебном пособии будут рассмотрены только два структурированных типа данных: регулярный тип (массивы) и строковый тип.

С понятием «массив» приходится встречаться при решении научно-технических, экономических задач обработки большого количества однотипных значений.

Таким образом, массив – это упорядоченная последовательность данных, состоящая из фиксированного числа элементов, имеющих один и тот же тип, и обозначаемая одним именем.

Название *регулярный тип* массивы получили за то, что в них объединены однородные элементы, упорядоченные (урегулированные) по индексам, определяющим положение каждого элемента в массиве.

Массиву присваивается имя, посредством которого можно ссылаться на него как на единое целое. Элементы, образующие массив, упорядочены так, что каждому элементу соответствует совокупность номеров (индексов), определяющих его место в общей последовательности. Индексы представляют собой выражения простого типа. Доступ к каждому отдельному элементу осуществляется обращением к имени массива с указанием индекса нужного элемента:

<имя массива>[<индекс>].

Чтобы использовать массивы в программах, нужно их описать в разделе описаний. Тип массива не является стандартным, поэтому его необходимо описать в части описания типов. Описание типа массива определяет его имя, размер массива и тип данных:

type <имя нового типа данных>=array[<тип индекса>] of <тип компонентов>;

Далее, в перечне переменных указывается имя массива и через двоеточие указывается имя нового типа данных:

var <имя массива>:<имя типа>;

Массив может быть описан и без представления типа в разделе описания типов данных:

var <имя массива>:array[<тип индекса>] of <тип компонентов>;

Этот вариант описания короче, но в некоторых случаях, когда описание переменных типа массив встречается несколько раз в различных частях программы, необходимо описание этого типа отдельно, как приведено в первом варианте.

Чаще всего в качестве типа индекса используется интервальный целый тип.

6.1. Одномерные массивы

Линейный (одномерный) массив – массив, у которого в описании задан только один индекс; если два индекса, то это двумерный массив и т. д. Одномерные массивы часто называют *векторами*, т. е. они представляют собой конечную последовательность пронумерованных элементов. Пример описания одномерного массива:

```
type vec=array[1..5] of real;
var x:vec;
```

или

```
var x: array[1..5] of real;
```

Оба из вариантов описывают одномерный массив **x**, состоящий из 5 вещественных элементов.

Присваивание начальных значений (заполнение массива) заключается в присваивании каждому элементу массива некоторого значения заданного типа. Наиболее эффективно эта операция осуществляется при помощи оператора **for**. Ввод данных может осуществляться: с клавиатуры, из файла данных, при помощи различных формул, в том числе и датчика случайных чисел.

Индексированные элементы массива называются *индексированными переменными* и могут быть использованы так же, как и простые переменные. Например, они могут находиться в выражениях в качестве операндов, им можно присваивать любые значения, соответствующие их типу, и т. д.

Алгоритм решения задач с использованием массивов:

- описание массива;
- заполнение массива;
- вывод (распечатка) массива;
- выполнение условий задачи;
- вывод результата.

6.1.1. Заполнение массива

Рассмотрим типичные варианты заполнения массива **x**, описанного выше.

Заполнение всех элементов массива числом 1 :

```
for i:=1 to 5 do x[i]:=1;
```

Заполнение всех элементов массива случайными числами из диапазона 1–9 :

```
for i:=1 to 5 do x[i]:=random(9);
```

Заполнение всех элементов массива при помощи ввода с клавиатуры:

```
for i:=1 to 5 do
    begin
    write('x[',i,']=');
    readln(x[i]);
    end;
```

6.1.2. Вывод массива на экран

Вывод массива на экран в одну строку без пояснений:

```
for i:=1 to 5 do writeln(x[i]:6:1);
```

Вывод массива на экран в столбец с пояснениями. Этот вариант гораздо нагляднее. Старайтесь использовать его:

```
for i:=1 to 5 do writeln('x[',i,']=',x[i]:6:1);
```

6.1.3. Работа с массивами

Пример 6.1. Определить самую высокую температуру и самый теплый день в мае.

```
program massiv;
uses crt;
var t:array[1..31] of integer;
    i,max,n:integer;
begin
Clrscr;
for i:=1 to 31 do
    begin
    t[i]:=random(20);
    write(t[i],' ');
    end;
writeln;
max:=t[1]; n:=1;
for i:=2 to 31 do
    begin
    if t[i]>max then
        begin max:=t[i]; n:=i; end;
    end;
writeln('t-макс.= ',max,' в ',n, 'день');
end.
```

6.2. Двумерные массивы

Двумерный массив – структура данных, хранящая прямоугольную матрицу. В матрице каждый элемент определяется номером строки и номером столбца, на пересечении которых он расположен. В Паскале двумерный массив представляется массивом, элементами которого являются одномерные массивы. Два следующих описания двумерных массивов тождественны:

```
var a:array [1..5] of array [1..6] of real;
var a:array [1..5,1..6] of real;
```

Чаще всего при описании двумерного массива используют второй способ. Так же как и для одномерных массивов, для двумерных можно использовать отдельно описание нового типа массива, а затем описывать переменную, используя этот тип:

```
type matr=array [1..5,1..6] of integer;
var a:matr;
```

Доступ к каждому отдельному элементу осуществляется обращением к имени массива с указанием индексов (первый индекс – номер строки, второй индекс – номер столбца). Все действия над элементами двумерного массива идентичны действиям над элементами линейного массива. Только для инициализации двумерного массива используется конструкция, когда один цикл **for** вложен в другой. Например:

```
for i:=1 to 5 do
    for j:=1 to 6 do
        a[i,j]:=0;
```

При организации вложенных (сложных) циклов необходимо учитывать:

- Все правила, присущие простому циклу, должны соблюдаться.
- Имена параметров для циклов, вложенных один в другой, должны быть различными.
- Внутренний цикл должен полностью входить в тело внешнего цикла. Пересечение циклов недопустимо.

6.2.1. Заполнение матрицы

Рассмотрим типичные варианты заполнения матрицы **a**, описанной выше.

Заполнение всех элементов матрицы случайными числами из диапазона 1 – 9 :

```
for i:=1 to 5 do
    for j:=1 to 6 do
        a[i,j]:=random(9);
```

Заполнение всех элементов матрицы при помощи ввода с клавиатуры:

```
for i:=1 to 5 do
    for j:=1 to 6 do
        begin
        write('a[',i,',',j,']=');
        readln(a[i,j]);
        end;
```

6.2.2. Вывод матрицы на экран

Вывести на экран матрицу 5 × 6 можно следующим образом:

```
for i:=1 to 5 do
    begin
    for j:=1 to 6 do
        write(a[i,j]:4); {вывод строки}
    writeln;             {перевод на новую строку}
    end;
```

6.2.3. Работа с матрицами

Работа с матрицами осуществляется также поэлементно.

Пример 6.2. Сформировать таблицу Пифагора (таблица умножения) и вывести ее на экран.

```
program pifagor;
uses crt;
var p:array [1..9,1..9] of integer;
    i,j:integer;
begin
Clrscr;
for i:=1 to 9 do
    for j:=1 to 9 do
        p[i,j]:=i*j;
for i:=1 to 9 do
    begin
    for j:=1 to 9 do
        write(p[i,j]:4);
    writeln;
    end;
end.
```

Пример 6.3. Задан двумерный массив B(10, 10), заполненный случайными числами из диапазона -10 – 10. Найти и вывести на экран произведение элементов побочной диагонали.

```
program massiv;
uses crt;
var b:array[1..10,1..10] of integer;
    i,j,s:integer;
begin
Clrscr;
for i:=1 to 10 do
    begin
    for j:= 1 to 10 do
        begin
        b[i,j]:=random(20)-10;
        write(b[i,j]:4);
        end;
    writeln;
    end;
s:=1;
for i:=1 to 10 do
    s:=s*b[i,11-i];
writeln('Произведение = ',s);
end.
```

Пример 6.4. Ввести с клавиатуры матрицу B(5, 5) и поменять местами первый и последний столбец.

```
program mest;
var b:array[1..5,1..5] of integer;
    i,j,s:integer;
begin
for i:=1 to 5 do
    for j:=1 to 5 do
        begin
        write('b[',i,',',j,']=');
        readln(b[i,j]);
        end;
for i:=1 to 5 do
    begin
    for j:=1 to 5 do
        write(b[i,j]:4);
```

```
    writeln;
    end;
for i:=1 to 5 do
    begin
    s:=b[i,1]; b[i,1]:=b[i,5]; b[i,5]:=s;
    end;
writeln;
writeln('Изменённая матрица');
writeln;
for i:=1 to 5 do
    begin
    for j:=1 to 5 do
        write(b[i,j]:4);
    writeln;
    end;
end.
```

7. РАБОТА С ФАЙЛАМИ ДАННЫХ

Очень важным моментом в программировании инженерных задач является работа с файлами данных. Особенно часто ввод данных из файла используется для ввода массивов и матриц.

Файловый тип данных (или файл) определяет упорядоченную совокупность произвольного числа однотипных компонент.

При работе с файлами выполняются операции ввода – вывода. Операция ввода означает перепись данных с внешнего устройства (из входного файла) в основную память ЭВМ, операция вывода – это пересылка данных из основной памяти на внешнее устройство (в выходной файл).

Файлы на внешних устройствах часто называют ***физическими файлами***. Их имена определяются операционной системой. В программах на языке Паскаль имена файлов задаются с помощью строк. Например, имя файла на диске может иметь вид

```
'A:\LAB1.DAT'
'c:\ABC150\pr.pas'
'dat.pas'.
```

Для работы с файлами в программе необходимо определить ***файловую переменную***. Турбо Паскаль поддерживает три файловых типа: текстовые файлы, типизированные файлы, нетипизированные файлы.

Описание типизированных файлов имеет вид

```
var f1: File of T;
```

Здесь **T** – тип компоненты файла. Примеры описания файловой переменной компонентного типа:

```
type M=array[1..500] of Longint;
var fLi: File of M;
    f1: File of Real;
    f2: File of Integer;
```

Бестиповые файлы описываются с помощью служебного слова File:

```
var f1: File;
```

Особое место в языке Паскаль занимают текстовые файлы, компоненты которых имеют символьный тип. Для описания текстовых файлов в языке определен стандартный тип Text:

```
var TF1,TF2: Text;
```

Текстовые файлы представляют собой последовательность строк, а строки – последовательность символов. Строки имеют переменную длину, каждая строка завершается признаком конца строки. Иначе говоря, описанные выше файловые переменные можно было описать так:

```
var TF1,TF2: File of Char;
```

Будем использовать первый вариант описания.

Поскольку работа с текстовыми файлами более удобна в использовании и чаще применяется, то дальше рассматриваем именно её.

Файловые переменные, которые описаны в программе, называют *логическими файлами*. Все основные процедуры и функции, обеспечивающие ввод-вывод данных, работают только с логическими файлами. Физический файл должен быть связан с логическим до выполнения процедур открытия файлов.

Турбо Паскаль вводит ряд процедур и функций, применимых для любых типов файлов:

```
Assign (var f; FileName: String);
```

Процедура связывает логический файл **f** с физическим файлом, полное имя которого задано в строке **FileName**.

```
Reset (var f);
```

Процедура открывает логический файл **f** для последующего чтения данных, или, как говорят, открывает входной файл. Если физического файла не существует, то возникает ошибка. После успешного выполнения процедуры **Reset** файл готов к чтению из него первого элемента. Указатель файла устанавливается на начало.

```
Rewrite (var f);
```

Процедура открывает логический файл **f** для последующей записи данных (открывает выходной файл). Если физического файла не существует, то процедура создаст и откроет его. Если физический файл уже существует, то процедура откроет его и уничтожит все находящиеся в нём данные. После успешного выполнения этой процедуры файл готов к записи в него первого элемента. Указатель файла устанавливается на начало.

```
Append (var f: Text);
```

Процедура служит для специального открытия выходных текстовых файлов. Она применима к уже существующим физическим файлам и открывает их для дозаписи в конец файла. Указатель файла устанавливается за последним элементом.

```
Close (var f);
```

Процедура закрывает открытый до этого логический файл. Вызов процедуры **Close** необходим при завершении работы с файлом. Если по какой-то причине процедура **Close** не будет выполнена, файл все

же будет создан на внешнем устройстве, но содержимое последнего буфера в него не будет перенесено. Для входных файлов использование оператора закрытия файла необязательно.

EOF (var f): Boolean;

Логическая функция возвращает значение TRUE, когда при чтении достигнет конца файла. Это означает, что уже прочитан последний элемент в файле или файл после открытия оказался пуст.

Rename (var f; NewName: String);

Процедура позволяет переименовать физический файл на диске, связанный с логическим файлом **f**. Переименование возможно после закрытия файла.

Erase(var f);

Процедура уничтожает физический файл на диске, который был связан с файловой переменной **f**. Файл к моменту вызова процедуры **Erase** должен быть закрыт.

IOResult: Integer;

Функция возвращает целое число, соответствующее коду последней ошибки ввода-вывода. При нормальном завершении операции функция вернет значение 0. Значение функции **IOResult** необходимо присваивать какой-либо переменной, т. к. при каждом вызове функция обнуляет свое значение. Функция **IOResult** работает только при выключенном режиме проверок ошибок ввода-вывода или с ключом компиляции {$I-}.

Для операций ввода-вывода используют стандартные процедуры **Read** и **Write**, в которых первым параметром в списке стоит файловая переменная, указывающая, с каким файлом требуется провести операцию.

Read(T,a,b,c);

– вводит значения переменных **a,b,c** из файла **T**.

Write(T,x,y);

– выводит значения переменных **x,y** в файл **T**.

7.1. Особенности работы с текстовыми файлами.

Текстовые файлы наиболее удобны при подготовке исходных данных. Их можно создавать, просматривать и редактировать непосредственно в среде Турбо Паскаля, используя меню File.

Во время подготовки файла исходных данных значения переменных нужно отделять друг от друга пробелами. Перевод на новую строчку нужно учитывать при организации ввода переменных в тексте программы.

При вводе и выводе в текстовый файл происходит автоматическое преобразование типов переменных. Значение каждой переменной при вводе преобразовывается из текста в тип соответствующей переменной и присваивается ей. При выводе переменной её значение переводится в текст в соответствии с заданным форматом вывода.

Для операций над текстовыми файлами, кроме перечисленных, определены также операторы обращения к процедурам Readln и Writeln.

`Readln(T,a,b,c);`

– вводит значения переменных **`a,b,c`** из файла **`T`** и затем переводит указатель файла на новую строку.

`Writeln(T,x,y);`

– выводит значения переменных **`x,y`** в файл **`T`** в одну строчку и затем переводит указатель файла на новую строку.

`Readln(T);`

– пропускает строку до начала следующей, ничего не вводя.

`Writeln(T);`

– завершает строку файла, в которую производится запись, признаком конца строки и переходит к началу следующей.

7.1.1. Общий алгоритм ввода из файла данных

Общий алгоритм ввода из файла данных выглядит следующим образом:

1. Описываем файловую переменную в разделе описаний.
2. В основной части связываем файловую переменную с именем файла на диске. Процедура **`assign`**.
3. Открываем файл для чтения. Процедура **`reset`**.
4. Вводим значения переменных. Процедуры **`read`** и **`readln`**.
5. Закрываем файл. Процедура **`close`**.

Пример 7.1. Ввести из текстового файла данных с именем dat.pas два массива **`x[0..7]`** и y**`[0..7]`**, состоящих из вещественных чисел.

Создадим на диске текстовый файл с именем dat.pas и запишем в него следующие две строчки:

```
0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8
0.1   -0.1    0.5    1.1    2.1    3      4      10
```

Сохраним этот файл и перейдём в файл основной программы.

Пусть эти массивы и файловая переменная описаны в разделе описаний так:

```
var x,y:array [0..7] of real;
    fl:text;
```

Кусочек программы, который будет осуществлять ввод, может выглядеть так:

```
assign(fl,'dat.pas'); {имя файла данных на диске}
reset(fl);            {открыть файл для ввода}
for i:=0 to 7 do read(fl,x[i]); {ввод 1 строки}
readln(fl);           {перевод строки}
for i:=0 to 7 do read(fl,y[i]); {ввод 2 строки}
close(fl);            {закрыть файл}
```

Пример 7.2. Сделать то же самое другим способом.

Создадим на диске текстовый файл с именем dat.pas и запишем в него следующие два столбца:

```
 X      Y
0.1    0.1
0.2   -0.1
0.3    0.5
0.4    1.1
0.5    2.1
0.6    3
0.7    4
0.8    10
```

Сохраним этот файл и перейдём в файл основной программы.

Пусть эти массивы и файловая переменная описаны в разделе описаний также, как и в предыдущем примере.

Кусочек программы, который будет осуществлять ввод, может выглядеть так:

```
assign(fl,'dat.pas'); {имя файла данных на диске}
reset(fl);            {открыть файл для ввода}
readln(fl);           {пропуск первой строки}
for i:=0 to 7 do
    readln(fl,x[i],y[i]);
close(fl);            {закрыть файл}
```

Этот вариант короче. Здесь одним оператором ввода считываются очередные значения массивов **x** и **y**, и указатель переводится на новую строку.

7.1.2. Общий алгоритм вывода в файл результатов

Общий алгоритм вывода в файл результатов выглядит следующим образом:

1. Описываем файловую переменную в разделе описаний.
2. В основной части связываем файловую переменную с именем файла на диске (процедура **assign**).
3. Открываем файл для записи (процедура **rewrite** или **append**).
4. Осуществляем вывод в файл результатов (процедуры **write** и **writeln**).
5. Закрываем файл (процедура **close**).

Пример 7.3. Вывести матрицу **a[1..5,1..5]**, состоящую из вещественных чисел, в файл результатов с именем rezultat.pas.

Пусть матрица и файловая переменная описаны в разделе описаний так:

```
var a:array [1..5,1..5] of real;
    fr:text;
```

Кусочек программы, который будет осуществлять вывод, может выглядеть так:

```
assign(fr,'rezultat.pas');
rewrite(fr);          {открыть файл для ввода}
writeln(fr,'Матрица A'); {вывод поясняющей строки}
writeln(fr);              {пропуск строки}
for i:=1 to 5 do
    begin
    for j:=1 to 5 do
        write(fr,a[i,j]:8:2); {вывод строки}
    writeln(fr);          {перевод на новую строку}
    end;
close(fl);                {закрыть файл}
```

Примечание. Вводить данные из одного файла и выводить в другой можно одновременно, но файловые переменные и имена файлов на диске для файла данных и файла результатов при этом должны быть разными.

8. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

Системы линейных алгебраических уравнений (СЛАУ) появляется почти в каждой области прикладной математики. В некоторых случаях эти СЛАУ непосредственно составляют ту задачу, которую необходимо решить, в других случаях задача сводится к решению такой системы. Примерами таких задач являются системы нелинейных уравнений, дифференциальных уравнений в частных производных, задачи аппроксимации и интерполяции и др. Решение СЛАУ является одной из самых распространенных и важных задач вычислительной математики.

Систему n линейных алгебраических уравнений с n неизвестными запишем как:

$$\left.\begin{array}{l} a_{11}\cdot x_1+a_{12}\cdot x_2+\cdots+a_{1n}\cdot x_n=b_1\ ; \\ a_{21}\cdot x_1+a_{22}\cdot x_2+\cdots+a_{2n}\cdot x_n=b_2\ ; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}\cdot x_1+a_{n2}\cdot x_2+\cdots+a_{nn}\cdot x_n=b_n\ . \end{array}\right\} \qquad (8.1)$$

Коэффициенты $a_{i,j}$ (i=1, 2,...,n; j=1,2,...,n) этой СЛАУ можно представить в виде квадратной матрицы $n \times n$:

$$A = \begin{vmatrix} a_{11}\ a_{12}\ldots a_{1n} \\ a_{21}\ a_{22}\ldots a_{2n} \\ \ldots\ldots\ldots\ldots \\ a_{n1}\ a_{n2}\ldots a_{nn} \end{vmatrix} . \qquad (8.2)$$

Систему (8.1) можно записать в матричном виде $A\cdot X=B$, где X – вектор-столбец неизвестных; B – вектор-столбец правых частей.

Рассмотрим некоторые специальные виды матриц :

$$C = \begin{vmatrix} 1 & 2 & 3 \\ 2 & 7 & 4 \\ 3 & 4 & 8 \end{vmatrix} , \quad T = \begin{vmatrix} 1 & 2 & 3 \\ 0 & 5 & 7 \\ 0 & 0 & 9 \end{vmatrix} ,$$

$$K = \begin{vmatrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 0 & 0 & 5 & 6 \\ 0 & 0 & 7 & 8 \end{vmatrix} , \quad Д = \begin{vmatrix} 2 & 3 & 0 & 0 & 0 \\ 1 & 2 & 3 & 0 & 0 \\ 0 & 3 & 4 & 5 & 0 \\ 0 & 0 & 4 & 5 & 6 \\ 0 & 0 & 0 & 1 & 2 \end{vmatrix} .$$

Здесь C – симметричная матрица (ее элементы расположены симметрично относительно главной диагонали, т. е. $a_{ij}=a_{ji}$); T – верхняя тре-

угольная матрица с равными нулю элементами, расположенными ниже диагонали; *К* – клеточная матрица (ее ненулевые элементы составляют отдельные группы-клетки); *Д* – ленточная матрица (ее ненулевые элементы составляют «ленту», параллельную диагонали; в данном случае данная матрица *Д* одновременно является также трехдиагональной).

Методы решения СЛАУ делятся на две группы – прямые и итерационные.

8.1. Прямые методы

Прямые методы используют заранее известное, зависящее от n, количество соотношений (формул) для вычисления неизвестных. К ним относятся правило Крамера, метод Гаусса (или метод последовательного исключения неизвестных), метод Гаусса с выбором главного элемента, метод прогонки (частный случай метода Гаусса для СЛАУ с трехдиагональной матрицей), метод Жордана (часто используется для нахождения обратной матрицы), метод квадратного корня (применяется тогда, когда матрица системы является симметричной), клеточные методы (используются для решения больших СЛАУ, когда матрица и вектор правых частей целиком не помещаются в оперативной памяти ЭВМ) и др. Алгоритмы этих методов сравнительно просты и наиболее универсальны, т. е. пригодны для решения широкого класса СЛАУ. К недостаткам таких методов относится требование хранения в оперативной памяти ЭВМ сразу всей исходной матрицы, и, следовательно, при больших значениях n используется большой объем памяти. Прямые методы не учитывают конкретный вид матрицы или ее структуру, т. е. при большом числе нулевых элементов в ленточных или клеточных матрицах эти элементы занимают место в памяти ЭВМ, и над ними проводятся практически бесполезные арифметические операции. Кроме того, существенным недостатком прямых методов является увеличение погрешностей в процессе получения решения, т. к. вычисления на последующем этапе используют результаты предыдущих операций, полученных, в свою очередь, с какой-то погрешностью. Особенно актуальным становится вышеуказанное обстоятельство для больших СЛАУ вследствие резкого увеличения общего количества числа арифметических действий, а также для плохо обусловленных СЛАУ из-за их чувствительности к погрешностям.

Поэтому прямые методы применяются для относительно небольших ($n < 200$) СЛАУ с плотно заполненной матрицей и не близким к нулю определителем.

8.2. Метод Гаусса

Наиболее распространенные прямые методы основаны на приведении СЛАУ к «треугольному» виду (см. матрицу T). Это достигается последовательным исключением неизвестных из уравнений исходной системы. Сначала с помощью первого уравнения исключается X_1 из всех последующих уравнений системы, затем, используя второе уравнение, исключается X_2 из третьего и всех последующих уравнений. Этот процесс, называемый прямым ходом метода Гаусса, продолжается до тех пор, пока в левой части последнего (n-го) уравнения не останется лишь один член с неизвестным X_n, т. е. матрица системы будет приведена к треугольному виду. (Замечание: к такому виду приводится только невырожденная матрица, иначе данный метод неприменим).

Пусть методом исключения Гаусса требуется решить СЛАУ

$$\left.\begin{aligned} x_1 + x_2 + x_3 - x_4 &= 2\ ; \\ x_1 - x_2 - x_3 + x_4 &= 0\ ; \\ 2x_1 + x_2 - x_3 + 2x_4 &= 9\ ; \\ 3x_1 + x_2 + 2x_3 - x_4 &= 7\ . \end{aligned}\right\}$$

Для удобства обозначим уравнения буквами и будем выписывать только коэффициенты при неизвестных и свободные члены уравнений. Тогда исходная СЛАУ примет вид

A_1	1	1	1	-1	2
A_2	1	-1	-1	1	0
A_3	2	1	-1	2	9
A_4	3	1	2	-1	7

Исключая члены, содержащие x_1, получим

$B_1 = A_1/1$	1	1	1	-1	2
$B_2 = A_2 - B_1$	0	-2	-2	2	-2
$B_3 = A_3 - 2B_1$	0	-1	-3	4	5
$B_4 = A_4 - 3B_1$	0	-2	-1	2	1

После исключения членов с x_2 имеем

B_1	1	1	1	-1	2
$C_2 = B_2/(-2)$	0	1	1	-1	1
$C_3 = B_3 + C_2$	0	0	-2	3	6
$C_4 = B_4 + 2C_2$	0	0	1	0	3

Исключения членов с x_3 дает

B_1	1	1	1	-1	2
C_2	0	1	1	-1	1
$D_3 = C_3/(-2)$	0	0	1	-3/2	-3
$D_4 = C_4 - D_3$	0	0	0	3/2	6

Продолжая аналогичные действия с последним рядом, получим

B_1	1	1	1	-1	2
C_2	0	1	1	-1	1
D_3	0	0	1	-3/2	-3
$E_4 = 2D_4/3$	0	0	0	1	4

Возвращаясь к общепринятой форме уравнений, запишем

$$\left.\begin{aligned} x_1 + x_2 + x_3 - x_4 &= 2; \\ x_2 + x_3 - x_4 &= 1; \\ x_3 - 1.5\,x_4 &= -3; \\ x_4 &= 4. \end{aligned}\right\}$$

Обратный ход метода Гаусса состоит в последовательном вычислении исходных неизвестных. Из последнего уравнения находим единственное неизвестное x_4, подставляя значение x_4 в третье уравнение, x_3 – во второе и т. д., находим решение заданной СЛАУ:

$$x_1 = 1, \quad x_2 = 2, \quad x_3 = 3, \quad x_4 = 4.$$

Аналогично строится алгоритм для СЛАУ с произвольным числом уравнений. Необходимо помнить, что при решении СЛАУ может потребоваться операция перестановок уравнений (т. е. перестановки соответствующих коэффициентов), служащая для предотвращения деления на нулевой элемент.

Данный метод целесообразно использовать для решения систем с плотно заполненной матрицей. Все элементы матрицы и правые части системы уравнений (все вместе есть расширенная матрица) находятся в оперативной памяти ЭВМ. Объем вычислений определяется порядком системы n: число арифметических операций примерно равно $(2/3)n^3$.

Вычисленное по методу Гаусса решение X^* для СЛАУ, записанной в матричном виде $A \cdot X = B$, отличается от точного решения X из-за погрешностей округления, связанных с ограниченностью разрядной сетки ЭВМ. Существуют две величины, характеризующие степень отклонения полученного решения X^* от точного X. Одна из них – погрешность E, равная разности этих значений: $E = X - X^*$.

Другая – невязка r, равная разности между правой и левой частями уравнений при подстановке в них решения: $r = B - A \cdot X^*$.

Реализация метода на языке Паскаль

Потребуется описание используемых типов для расширенной матрицы коэффициентов и для вектора значений переменных:

```
type mat=array[1..20,1..21] of real;
     vec=array[1..20] of real;
var n:integer; {Реальное число уравнений}
    a:mat; x:vec;
```

Ввод матрицы наиболее эффективно реализовать из файла данных. Для этого потребуется файловый тип и файловая переменная

```
var  f:text;
```

В основной части программы ввод можно организовать следующим образом:

```
Write('Введите число уравнений N = '); readln(n);
assign(f,'alex.dat'); {имя файла данных на диске}
reset(f);
for i:=1 to n do
    begin
    for j:=1 to n+1 do read(f,a[i,j]);
    readln(f);
    end;
close(f);
```

Прямой ход метода Гаусса (формирование треугольной матрицы коэффициентов) может выглядеть так:

```
for k:=1 to n do
    begin
    s:=a[k,k];j:=k;
    for i:=k+1 to n do
        begin
        r:=a[i,k];
        if abs(r)>abs(s) then
            begin
            s:=r; j:=i;
            end;
        end;
    if s=0 then
        begin
        writeln('Переставьте уравнения чтобы на
        главной диагонали не было нулевых коэффици-
        ентов !');
        halt;
        end;
    if j<>k then
        for i:=k to n+1 do
            begin
            r:=a[k,i]; a[k,i]:=a[j,i];a[j,i]:=r;
            end;
    for j:=k to n+1 do a[k,j]:=a[k,j]/s;
    for i:=k+1 to n do
        begin
```

```
        r:=a[i,k];
        for j:=k to n+1 do
            a[i,j]:=a[i,j]-a[k,j]*r;
        end;
    end;
```

Обратный ход метода Гаусса (вычисление вектора значений переменных) может выглядеть так:

```
if s<>0 then
    for i:=n downto 1 do
        begin s:=a[i,n+1];
        for j:=i+1 to n do s:=s-a[i,j]*x[j];
        x[i]:=s;
        end;
```

Остаётся только вывести результат и сделать проверку.

8.3. Метод прогонки

Он является модификацией метода Гаусса для частного случая разреженных систем – системы уравнений с трехдиагональной матрицей. Такие системы получаются при моделировании некоторых инженерных задач, а также при численном решении краевых задач для дифференциальных уравнений.

Запишем систему уравнений в виде

$$\left.\begin{aligned} &b_1x_1 + c_1x_2 = d_1 ;\\ &a_2x_1 + b_2x_2 + c_2x_3 = d_2 ;\\ &a_3x_2 + b_3x_3 + c_3x_4 = d_3 ;\\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\ &a_{n-1}x_{n-2}+b_{n-1}x_{n-1}+c_{n-1}x_n = d_{n-1} ;\\ &a_nx_{n-1} + b_nx_n = d_n. \end{aligned}\right\} \qquad (8.3)$$

На главной диагонали матрицы этой системы стоят элементы $b_1, b_2,...,b_n$, над ней – элементы $c_1, c_2,..., c_{n-1}$, под ней – элементы a_2,a_3,a_n. При этом обычно все коэффициенты b_i не равны нулю.

Метод прогонки состоит из двух этапов – прямой прогонки (аналога прямого хода метода Гаусса) и обратной прогонки (аналога обратного хода метода Гаусса). Прямая прогонка состоит в том, что каждое неизвестное x_i выражается через x_{i+1} с помощью прогоночных коэффициентов a_i, b_i:

$$x_i=a_ix_{i+1}+b_i, \quad i=1, 2,..., n-1. \qquad (8.4)$$

Из первого уравнения системы (8.3) найдем $x_1=(-c_1/b_1)\, x_2 + d_1/b_1$.

С другой стороны, по формуле (8.4) $x_1=a_1x_2+b_1$.

Приравнивая коэффициенты в обоих выражениях для x_1,получаем

$$a_1 = -c_1/b_1, \quad b_1 = d_1/b_1. \tag{8.5}$$

Из второго уравнения системы (8.3) выразим x_2 через x_3, заменяя x_1 по формуле (8.4):

$$a_2(a_1x_2 + b_1) + b_2x_2 + c_2x_3 = d_2.$$

Отсюда найдем $x_2 = (-c_2x_3 + d_2 - a_2b_1) / (a_2a_1 + b_2)$,

или $x_2 = a_2x_3 + b_2$;

$a_2 = -c_2/e_2$;

$b_2 = (d_2 - a_2b_1)/e_2$;

$e_2 = a_2a_1 + b_2$.

Аналогично можно вычислить прогоночные коэффициенты для любого номера i:

$$a_i = -c_i/e_i, \quad b_i = (d_i - a_ib_{i-1})/e_i \ ; \tag{8.6}$$

$$e_i = a_ia_{i-1} + b_i, \quad i = 2, 3, ..., n - 1 .$$

Обратная прогонка состоит в последовательном вычислении неизвестных x_i. Сначала нужно найти x_n . Для этого воспользуемся выражением (8.4) при $i = n - 1$ и последним уравнением системы (8.3). Запишем их:

$$x_{n-1} = a_{n-1}x_n + b_{n-1} \ ;$$

$$a_nx_{n-1} + b_nx_n = d_n \ .$$

Отсюда, исключая x_{n-1}, находим $x_n = (d_n - a_nb_{n-1})/(b_n + a_na_{n-1})$.

Далее, используя формулы (8.4) и выражения для прогоночных коэффициентов (8.5), (8.6), последовательно вычисляем все неизвестные $x_{n-1}, x_{n-2}, \ldots, x_1$.

8.4. Итерационные методы

Итерационные методы используют последовательные приближения (итерации, итерационные циклы), в процессе вычисления по которым образуется последовательность значений $X^0, X^1, \ldots, X^S, \ldots$, сходящаяся к некоторому пределу X^P, т. е.

$$\lim_{S \to \infty} X^S = X^P,$$

где каждое последующее значение X^S определяется через предыдущее X^{S-1} (S – номер итерации).

К таким методам относятся метод Якоби (метод простой итерации, или метод одновременных смещений), метод Зейделя (Гаусса-Зейделя, или метод последовательных смещений), метод верхней релаксации.

Итерации начинаются с задания начального приближенного решения X^0, которое может быть получено из физических или других разумных соображений. Чем ближе исходное приближение X^0 к решению

X^P, тем меньше итераций необходимо для его получения. Итерации проводятся до получения решения с требуемой точностью, которая не должна превышать заданной погрешности вычислений E, т. е. итерации заканчиваются при выполнении условия

$$|X^S - X^{S-1}| <= E_a \ , \qquad (8.7)$$

или условия

$$|(X^S - X^{S-1}) / X^S| <= E_o \ , \qquad (8.8)$$

где E_a – заданная абсолютная погрешность;

E_o – заданная относительная погрешность.

Алгоритмы решения СЛАУ итерационными методами обычно более сложные по сравнению с прямыми методами. Объем вычислений заранее определить трудно, т. к. он зависит от степени близости начального приближения X^0 к «точному» значению X^P, от величины E_a или E_o, от скорости (условий) сходимости используемого алгоритма (чем меньше конечное значение S при выполнении условий (8.7) или (8.8), тем скорость больше).

Преимущества данных методов по сравнению с прямыми заключаются в следующем:

1. В ряде случаев удается хранить в памяти ЭВМ не всю матрицу системы, а лишь несколько векторов (см. матрицы Т, К, Д).

2. Погрешности итерационных методов ограничены, (не увеличиваются), т. к. точность вычислений на итерации S определяется результатами предыдущей итерации $S - 1$ и практически не зависит от результатов на итерациях $S - 2$, $S - 3$, ... Это является достоинством итерационных методов, особенно полезным в случае большого числа уравнений, плохо обусловленных СЛАУ (когда малые погрешности вычислений или исходных данных приводят к большим погрешностям в решении).

Для реализации итерационного метода исходную СЛАУ (8.1) обычно приводят к виду :

$$\left.\begin{array}{l} x_1 = (b_1 - a_{12}x_2 - a_{13}x_3 - ... - a_{1n}x_n) / a_{11} \ ; \\ x_2 = (b_2 - a_{21}x_1 - a_{23}x_3 - ... - a_{2n}x_n) / a_{22} \ ; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n = (b_n - a_{n1}x_1 - a_{n2}x_2 - ... - a_{n(n-1)}x_{n-1}) / a_{nn} \ . \end{array}\right\} \qquad (8.9)$$

8.5. Метод Зейделя

Этот метод является итерационным. На каждой итерации уточняются значения переменных. Система сходится, если на главной диагонали матрицы коэффициентов расположены максимальные элементы. В противном случае необходимо провести эквивалентные преобразования (перестановка строк и др.).

Систему (8.9) приведём к виду

$$\left.\begin{array}{l} x_1 = (b_1 - a_{11}x_1 - a_{12}x_2 - \ldots - a_{1n}x_n) / a_{11} + x_1 \ ; \\ x_2 = (b_2 - a_{21}x_1 - a_{22}x_2 - \ldots - a_{2n}x_n) / a_{22} + x_2 \ ; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n = (b_n - a_{n1}x_1 - a_{n2}x_2 - \ldots - a_{nn}x_n) / a_{nn} + x_n \ . \end{array}\right\} \qquad (8.10)$$

Левую часть уравнений будем считать новыми (последующими) значениями переменных, а x_i в правой части – предыдущими значениями переменных. Первоначально массив X имеет нулевые значения. Затем на каждой итерации вычисляется S – добавка для каждого x_i, которая приближает его к искомому значению. Процесс продолжается до тех пор, пока добавка для каждого x_i будет меньше заданной погрешности вычислений.

Реализация метода на языке Паскаль

Ввод данных рекомендуется организовать так же, как и в методе Гаусса. Далее приводится возможный вариант реализации метода:

```
for k:=1 to m do
    begin
    z:=k;
    for i:=1 to n do
        begin
        s:=a[i,n+1];
        for j:=1 to n do s:=s-a[i,j]*x[j];
        s:=s/a[i,i];
        x[i]:=x[i]+s;
        if abs(s)>e then z:=0
        end;
    if z<>0 then Break;
end;
```

Здесь **m** – максимальное число итераций (для избежания бесконечного цикла); **z** – переменная, обнуляющаяся, если хотя бы одна из переменных отличается от предыдущего значения более чем на E, и содержащая число итераций, если система сошлась.

Остаётся только вывести результат и сделать проверку.

8.6. Метод простых итераций

Метод простых итераций немного отличается от метода Зейделя. Здесь имеется два вектора переменных: с предыдущими значениями **X0**

и с последующими значениями **X1**. В конце каждой итерации производится переприсвоение значений из последующих в предыдущие:

```
for k:=1 to m do
    begin
    z:=k;
    for i:=1 to n do
        begin
        s:=a[i,n+1];
        for j:=1 to n do s:=s-a[i,j]*X0[j];
        s:=s/a[i,i];
        X1[i]:=X0[i]+s;
        if abs(s)>e then z:=0
        end;
    if z<>0 then Break
            else
            for i:=1 to n do X0[i]:=X1[i];
    end;
```

8.7. Вывод результатов и проверка

Чтобы проверить, правильно ли решена система уравнений, нужно подставить найденные значения *x* в левую часть каждого уравнения и найти относительную погрешность между полученным значением и правой частью этого уравнения:

```
    for i:=1 to n do
        writeln('X[,i,]=',X[i]);
    writeln('Проверка:');
    for i:=1 to n do
        begin
        s:=0;
        for j:=1 to n do s:=s+a[i,j]*X[j];
        write('Уравнение ',i,' ',s,' = ',a[i,n+1]);
        writeln(' Погрешность ',
                abs((s-a[i,n+1])/s));
        end;
```

Замечание. При решении системы методом Гаусса перед проверкой нужно заново произвести ввод исходной матрицы из файла данных. Это нужно сделать потому, что исходная матрица *A* была преобразована в треугольную при прямом проходе и теперь требуется перечитать её исходное состояние для проведения проверки.

9. АППРОКСИМАЦИЯ ФУНКЦИЕЙ. МЕТОД НАИМЕНЬШИХ КВАДРАТОВ

В инженерной деятельности часто возникает необходимость описать в виде функциональной зависимости связь между величинами, заданными таблично или в виде набора точек с координатами (x_i,y_i), i=0,1,2,...,n, где n – общее количество точек. Как правило, эти табличные данные получены экспериментально и имеют погрешности. При аппроксимации желательно получить относительно простую функциональную зависимость (например, полином), которая позволила бы «сгладить» экспериментальные погрешности, получить промежуточные и экстраполяционные значения функций, изначально не содержащиеся в исходной табличной информации.

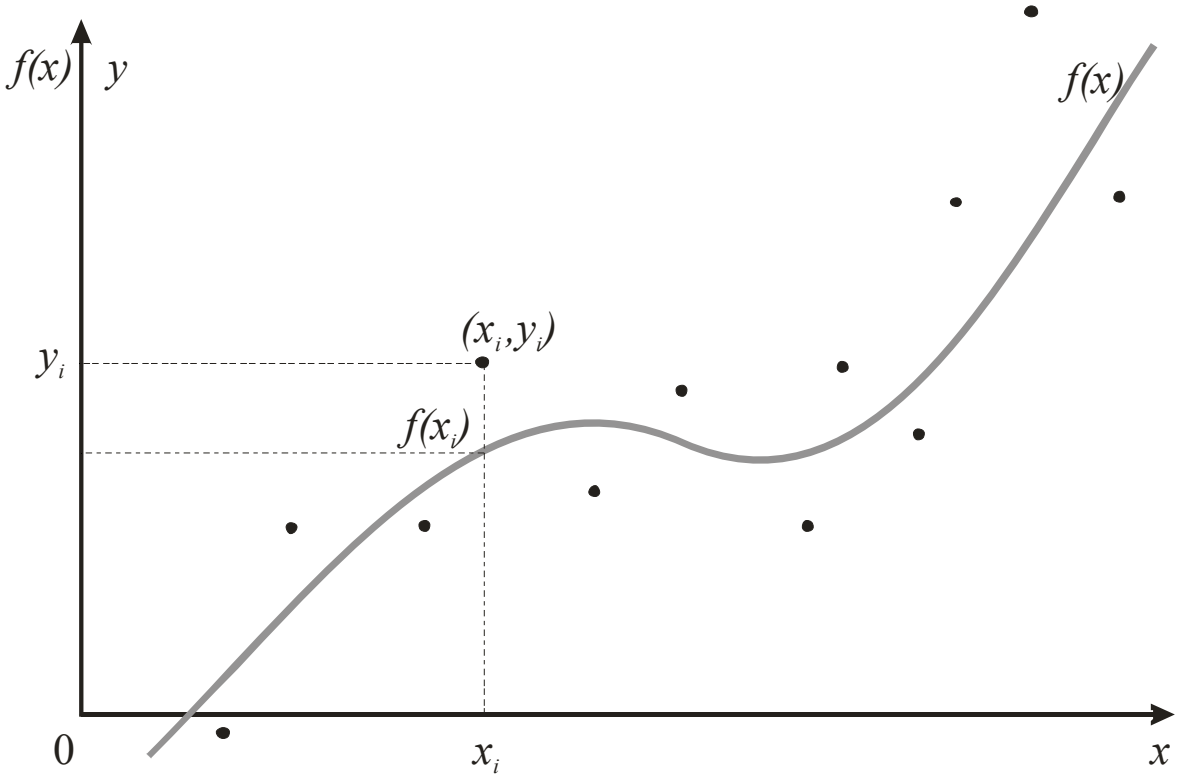

Рис. 9.1. Графическая интерпретация аппроксимации

Эта функциональная (аналитическая) зависимость должна с достаточной точностью соответствовать исходной табличной зависимости. Критерием точности или достаточно «хорошего» приближения могут служить несколько условий.

Обозначим через f_i значение, вычисленное из функциональной зависимости для $x = x_i$ и сопоставляемое с y_i.

Одно из условий согласования можно записать как

$$S = \sum_{i=0}^{n} (f_i - y_i) \ \rightarrow \ \min,$$

т. е. сумма отклонений табличных и функциональных значений для одинаковых $x = x_i$ должна быть минимальной (метод средних). Отклонения могут иметь разные знаки, поэтому достаточная точность в ряде случаев не достигается.

Использование критерия

$$S = \sum_{i=0}^{n} |f_i - y_i| \ \rightarrow \ \min$$

также неприемлемо, т. к. абсолютное значение не имеет производной в точке минимума.

Учитывая вышеизложенное, используют критерий ***наименьших квадратов***, т. е. определяют такую функциональную зависимость, при которой

$$S = \sum_{i=0}^{n} (f_i - y_i)^2 \qquad (9.1)$$

обращается в минимум.

В качестве функциональной зависимости рассмотрим многочлен

$$f(x) = C_0 + C_1 x + C_2 x^2 + ... + C_m x^m. \qquad (9.2)$$

Обозначим далее $\sum_{i=0}^{n}$ как Σ. Формула (1) примет вид

$$S = \Sigma \, (C_0 + C_1 x_i + C_2 x_i^2 + ... + C_m x_i^m - y_i)^2.$$

Условия минимума S можно записать, приравнивая нулю частные производные S по независимым переменным $C_0, C_1, ..., C_m$:

$$\left.\begin{array}{l} S_{C0} = 2 \, \Sigma \, (C_0 + C_1 x_i + C_2 x_i^2 + ... + C_m x_i^m - y_i) \ = 0 \ ; \\ S_{C1} = 2 \, \Sigma \, (C_0 + C_1 x_i + C_2 x_i^2 + ... + C_m x_i^m - y_i) \, x_i = 0 \ ; \\ ..... \\ S_{CM} = 2 \, \Sigma \, (C_0 + C_1 x_i + C_2 x_i^2 + ... + C_m x_i^m - y_i) \, x_i^m = 0. \end{array}\right\} \qquad (9.3)$$

Тогда из (3) можно получить систему нормальных уравнений

$$\left.\begin{array}{l} C_0(n+1) + C_1\Sigma x_i \quad + C_2\Sigma x_i^2 \; +\ldots+ \; C_m\Sigma x_i^m \; = \Sigma y_i\,; \\ C_0\Sigma x_i \; + \; C_1\Sigma x_i^2 \; + C_2\Sigma x_i^3 \; +\ldots+ \; C_m\Sigma x_i^{m+1} = \Sigma y_i x_i\,; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ C_0\Sigma x_i^m + C_1\Sigma x_i^{m+1} + C_2\Sigma x_i^{m+2} +\ldots+ C_m\Sigma x_i^{2m} \; = \Sigma y_i x_i\,. \end{array}\right\} \qquad (9.4)$$

Для определения коэффициентов C_i и, следовательно, искомой зависимости (9.2) необходимо вычислить суммы и решить систему уравнений (9.4). Матрица системы (9.4) называется *матрицей Грама* и является симметричной и положительно определенной. Эти полезные свойства используются при ее решении:

$$\left|\begin{array}{lllll} (n+1) & \Sigma\, x_i & \Sigma\, x_i^2 & \ldots & \Sigma\, x_i^m & \Sigma\, y_i \\ \Sigma\, x_i & \Sigma\, x_i^2 & \Sigma\, x_i^3 & \ldots & \Sigma\, x_i^{m+1} & \Sigma\, y_i x_i \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \Sigma\, x_i^m & \Sigma\, x_i^{m+1} & \Sigma\, x_i^{m+2} & \ldots & \Sigma\, x_i^{2m} & \Sigma\, y_i x_i^m \end{array}\right| \qquad (9.5)$$

Нетрудно видеть, что для формирования расширенной матрицы Грама (9.5) достаточно вычислить только элементы первой строки и двух последних столбцов, остальные элементы не являются «оригинальными» и заполняются с помощью циклического присвоения.

9.1. Процедура заполнения расширенной матрицы Грама

Реализация процедуры заполнения расширенной матрицы Грама на языке Паскаль может выглядеть так:

```
function ct(x:real;n:integer):real;
{Функция возведения X в целую степень n}
    var i:integer;s:real;
begin
s:=1;
for i:=1 to n do s:=s*x;
ct:=s;
end;

procedure gram(n,m:integer;var a:tr);
    var i,j:integer; p,q,r,s:real;
begin
for j:=0 to m do
    begin
    s:=0; r:=s; q:=s;
    for i:=0 to n do
        begin
```

```
        p:=ct(x[i],j); s:=s+p;
        r:=r+p*y[i];
        q:=q+p*ct(x[i],m);
        end;
    a[0,j]:=s; a[j,m+1]:=r; a[j,m]:=q;
    end;
for i:=1 to m do
    for j:=0 to m-1 do a[i,j]:=a[i-1,j+1];
a[0,0]:=n+1;
end;
```

где **a[i,j]** – расширенная матрица Грама, **x[i],y[i]** – исходные точки, **n** – число исходных точек, **m** – степень полинома.

Далее можно переходить к решению этой системы уравнений. Для решения систем уравнений с матрицей Грама разработаны методы сингулярного разложения. Если же **m** < 5, то такие системы можно решать и более простым методом исключения Гаусса (см. выше).

После решения системы мы нашли искомые коэффициенты C_0, C_1, ..., C_m аппроксимирующей функции

$$f(x)=C_0 + C_1 x + C_2 x^2+...+C_m x^m.$$

Для проверки нужно подставить данные **x[i]** в найденную функцию и сравнить значения с заданными **y[i]**, а также вычислить относительную погрешность для этих точек.

Можно составить функцию **F1** для вычисления значений полинома:

```
function F1(x:real; m:integer):real;
    var i:integer; s:real;
begin
s:=0;
for i:=0 to m do s:=s+c[i]*ct(x,i);
F1:=s;
end;
```

9.2. Алгоритм решения задачи

Весь алгоритм решения задачи будет выглядеть следующим образом.

1. Ввод исходных данных:
   - **n** – число заданных точек;
   - **m** – степень полинома;
   - **x[i],y[i]** – заданные точки. Смотрите примеры 7.1 и 7.2.
2. Формирование расширенной матрицы Грама **a[m,m+1]**.
3. Решение системы нормальных уравнений.

4. Вывод коэффициентов **c[m]**.

5. Циклический перебор точек **x[i]**, вычисление значения аппроксимирующей функции, относительной погрешности в каждой точке и вывод результатов на экран.

6. Далее рекомендуется воспользоваться графическими возможностями Турбо Паскаля и построить график найденной функции, а также нанести исходные точки. Подробнее смотрите в следующей главе.

МНК можно применить не только к многочленам, но и к другим функциональным зависимостям. Например, вместо (9.2) ищется зависимость в виде показательной функции

$$f(x) = y = a\, x^b.$$

Непосредственное использование МНК приведет к нелинейным нормальным уравнениям, но эта проблема решается, если использовать вместо экспериментальных значений их логарифмы:

$$\ln(y) = \ln(a) + b \ln(x).$$

Теперь минимизируется сумма квадратов разностей между логарифмами экспериментальных значений y_i и логарифмами ординат функции

$$y = a \quad x^b.$$

При этом получаем

$$S = \Sigma\, [\ln(y_i) - \ln(a) - b \ln(x_i)]^2.$$

Дифференцируя по a и b, получим систему нормальных уравнений

$$\left.\begin{aligned} &(n+1) \ln(a) \; + \Sigma \ln(x_i)\, b \; = \; \Sigma \ln(y_i)\,; \\ &\Sigma \ln(x_i) \ln(a) + \Sigma \ln(x_i)^2\, b = \Sigma\, [\ln(x_i) \ln(y_i)]\,. \end{aligned}\right\}$$

Неизвестными в этой системе являются ln (a) и b. После их определения a находится по величине логарифма.

10. ГРАФИКА В СИСТЕМЕ ТУРБО ПАСКАЛЬ

Экран дисплея ПК представляет собой прямоугольное поле, состоящее из большого количества точек. Дисплей может работать в текстовом и графическом режимах. Но в отличие от текстового режима в графическом режиме имеется возможность изменять цвет каждой точки.

Чтобы сделать процесс графического программирования более эффективным, фирма Borland International разработала специализированную библиотеку Graph (в этом библиотечном модуле содержится 79 графических процедур, функций, различных стандартных констант и типов данных), набор драйверов, позволяющих работать с разными типами мониторов, и набор шрифтов для вывода на графический экран текстов разной величины и формы.

Аппаратная поддержка графики ПК обеспечивается двумя основными модулями: видеомонитором и видеоадаптером. Какой бы адаптер ни был установлен на компьютере, мы можем использовать один и тот же набор графических процедур и функций Турбо Паскаля благодаря тому, что их конечная настройка на конкретный адаптер осуществляется автоматически. Эту настройку выполняют графические драйверы.

10.1. Запуск и завершение работы в графической системе

Запуск и завершение работы в графической системе осуществляется следующим образом:

1. Подключить модуль Graph (библиотеку графических процедур):

```
uses Graph;
```

2. Установить графический режим:

- описываем переменные, которые определяют графический драйвер и монитор:

```
var gd, gm: integer;
```

- в основной части программы задаем команду ПК для самовыбора значений переменных:

```
gd:=Detect;
```

значение **gm** после этой команды определяется автоматически;

- инициализируем графический режим:

```
InitGraph(gd,gm,'c:\tp7\bin');
```

(в апострофах указывается путь к драйверу: чем подробнее, тем лучше). С этого момента все графические средства доступны пользователю.

3. Завершить работу в графической системе:

```
CloseGraph;
```

10.2. Базовые процедуры и функции

Для построения изображений на экране используется система координат. Отсчет начинается от верхнего левого угла экрана, который имеет координаты (0,0). Значение *X* (столбец) увеличивается слева направо, значение *Y* (строка) увеличивается сверху вниз. Чтобы строить изображения, необходимо указывать точку начала вывода. В текстовых режимах эту точку указывает курсор, который присутствует на экране. В графических режимах видимого курсора нет, но есть невидимый текущий указатель *CP* (Current Pointer). Фактически это тот же курсор, но он невидим.

10.2.1. Процедуры модуля Graph

`SetColor(a:word);`

Устанавливает цвет, которым будет осуществляться рисование.

`SetBkColor(a:word);`

Устанавливает цвет фона.

`SetFillStyle(a,b:word);`

Устанавливает стиль и цвет закраски: **`a`** – стиль закраски, **`b`** – цвет.

`SetLineStyle(a,b,c:word);`

Устанавливает стиль и толщину линии: **`a`** – стиль линии, **`b`** – образец построения линии (может устанавливаться пользователем), **`c`** – толщина линии.

`SetTextStyle(a,b,c:word);`

Устанавливает шрифт, стиль и размер текста.

`SetFillPattern(Pattern:FillpatternType;Color:word);`

Выбирает шаблон заполнения, определенный пользователем. **`Pattern`** – маска.

`ClearDivice;`

Очищает экран и устанавливает текущий указатель в начало.

`SetViewPort(x1,y1,x2,y2:integer;Clip:boolean);`

Устанавливает текущее окно для графического вывода.

`ClearViewPort;`

Очищает окно.

`PutPixel(a,b,c:integer);`

Рисует точку цветом **`c`** в (x, y).

`Line(x1,y1,x2,y2:integer);`

Рисует линию от (x_1, y_1) к (x_2, y_2).

`Rectangle(x1,y1,x2,y2:integer);`

Рисует прямоугольник; (x_1, y_1) – координаты левого верхнего угла, (x_2, y_2) – координаты правого нижнего угла прямоугольника. Область внутри прямоугольника не закрашена и совпадает по цвету с фоном.

Bar(x1,y1,x2,y2:integer);

Рисует закрашенный прямоугольник.

Bar3D(x1,y1,x2,y2,d:integer;a:boolean);

Рисует трехмерную полосу (параллелепипед).

Circle(x,y,r:word);

Рисует окружность радиуса **r** с центром в точке (*x*, *y*).

Arc(x,y,a,b,r:integer);

Рисует дугу из начального угла к конечному, используя (*x*, *y*) как центр; **a**, **b** – начальный и конечный углы в градусах.

Ellipse(x,y,a,b,Rx,Ry:integer);

Рисует эллиптическую дугу от начального угла к конечному, используя *(x,y)* как центр; **a**, **b** – начальный и конечный углы в градусах; **Rx**, **Ry** – вертикальная и горизонтальная оси.

FillEllipse(x,y,Rx,Ry:integer);

Рисует закрашенный эллипс.

MoveTo(x,y:integer);

Передвигает текущий указатель в (*x*, *y*).

MoveRel(x,y:integer);

Передвигает текущий указатель на заданное расстояние от текущей позиции на **x** по горизонтали и на **y** по вертикали.

OutText(text:string);

Выводит текст от текущего указателя.

OutTextxy(x,y:integer;text:string);

Выводит текст с позиции (*x*, *y*).

Sector(x,y,a,b,Rx,Ry:integer);

Рисует и заполняет сектор эллипса; **a**, **b** – начальный и конечный углы в градусах.

10.2.2. Функции модуля Graph

GetBkColor

Возвращает текущий фоновый цвет.

GetColor

Возвращает текущий цвет.

GetX

Возвращает координату *X* текущей позиции.

GetY

Возвращает координату *Y* текущей позиции.

GetPixel

Возвращает цвет точки в (*x*, *y*).

10.3. Экран и окно в графическом режиме

По аналогии с текстовыми режимами графический экран может рассматриваться как одно большое или несколько меньших по размеру окон. После установки окна вся остальная площадь экрана как бы не существует, и весь ввод-вывод осуществляется только через окно. В каждый отдельный момент может быть активным только одно окно. Если окон несколько, за переключение ввода-вывода в нужное окно отвечает программист.

По умолчанию окно занимает весь экран, значения координат его левого верхнего и правого нижнего угла устанавливаются автоматически процедурой инициализации **InitGraph**.

Если требуется создать окно, следует воспользоваться процедурой
```
SetViewPort(x1,y1,x2,y2:integer;Clip:boolean);
```
Здесь **x1**, **y1** – координаты левого верхнего угла; **x2**, **y2** – координаты правого нижнего угла окна. Параметр **Clip** определяет, будет ли рисунок отсекаться при выходе за границы окна (**Clip:= True**) или нет (**Clip:=False**). После создания окна за точку отсчета принимается верхний левый угол окна, имеющий координаты (0,0).

Координатную систему полного экрана можно восстановить, в частности, с помощью **ClearDevice** или задав в процедуре установки окна максимально возможные значения:
```
SetViewPort(0,0,GetMaxX,GetMaxY,true);
```
Необходимо помнить, что, в отличие от текстовых окон, графические окна после команды установки фона **SetBkColor** и очистки с помощью **ClearViewPort** меняют фон вместе с общим фоном экрана. Поэтому фон (точнее «закраску») графического окна следует устанавливать с помощью процедур **SetFillStyle** и **SetFillPattern**.

10.4. Вывод простейших фигур

10.4.1. Вывод точки

Какие бы изображения не выводились на экран, все они построены из точек. Теоретически можно создать любое изображение путем построения точек определенного цвета в нужном месте экрана. В библиотеке **Graph** вывод точки осуществляется процедурой
```
PutPixel(x,y:integer,color:word);
```
Здесь **x**, **y** – координаты расположения точки, **color** – цвет.

Возможные значения **Color** приведены в табл. 10.1.

Пример. Операторы выводят в центре экрана точку красного цвета:
```
PutPixel(320,240,4);
```
или
```
PutPixel(320,240,Red);
```

10.4.2. Цветовая шкала

Таблица 10.1

Цвет	Код	Цвет	Код
Black – черный	0	DarkGray – темно-серый	8
Blue – синий	1	LightBlue – голубой	9
Green – зеленый	2	LightGreen – ярко-зеленый	10
Gyan – бирюзовый	3	LightGyan – ярко-бирюзовый	11
Red – красный	4	LightRed – ярко-красный	12
Magenta – малиновый	5	LightMagenta – ярко-малиновый	13
Brown – коричневый	6	Yellow – желтый	14
LightGray – светло-серый	7	White – белый	15

10.4.3. Вывод линии

Из точек строятся линии (отрезки прямых). Это можно сделать с помощью процедуры

```
Line(x1,y1,x2,y2:integer);
```

Здесь **x1,y1** – координаты начала, **x2,y2** – координаты конца линии, например

```
Line(1,1,600,1);
```

В процедуре **Line** нет параметра для установки цвета. В этом случае цвет задается процедурой **SetColor(цвет:word);** где цвет из табл. 10.1.

Пример:

```
SetColor(Gyan);
Line(1,1,600,1);
```

Для черчения линий применяются еще две процедуры: **LineTo** и **LineRel**. Процедура

```
LineTo(x,y:integer);
```

строит линию из точки текущего положения указателя в точку с координатами *(x,y)*. Процедура

```
LineRel(dx,dy:integer);
```

проводит линию от точки текущего расположения указателя (x, y) в точку ($x + dx$, $y + dy$).

Турбо Паскаль позволяет вычерчивать линии самого различного стиля: тонкие, широкие, штриховые, пунктирные и т. д. Установка стиля производится процедурой

```
SetLineStyle(a,b,c:word);
```

Здесь **a** устанавливает тип строки, возможные значения которого приведены в табл. 10.2, **b** – образец, **c** – толщина линии, определяемая константами, указанными в табл. 10.3. Если применяется один из стандарт-

ных стилей, то значение **b** равно 0. Если пользователь хочет активизировать собственный стиль, то значение **b** равно 4. В этом случае пользователь сам указывает примитив (образец), из которого строится линия.

Например:

```
SetLineStyle(1,0,1);
Line(15,15,150,130);
```

или

```
SetLineStyle(UserBitLn,$5555,ThickWidth);
Line(15,15,150,130);
```

10.4.4. Стандартные типы и толщина линий

Таблица 10.2

Константа	Значение	Описание
SolidLn	0	Непрерывная линия
DottedLn	1	Линия из точек
CenterLn	2	Линия из точек и тире
DashedLn	3	Штриховая линия
UserBitLn	4	Тип пользователя

Таблица 10.3

Константа	Значение	Описание
NormWidth	1	Нормальная толщина (1 пиксель)
ThickWidth	3	Жирная линия (3 пикселя)

10.4.5. Построение прямоугольников

Для построения прямоугольных фигур имеется несколько процедур. Первая из них – вычерчивание одномерного прямоугольника:

```
Rectangle(x1,y1,x2,y2:integer);
```

Здесь **x1,y1** – координаты левого верхнего угла, **x2,y2** – координаты правого нижнего угла прямоугольника. Область внутри прямоугольника не закрашена и совпадает по цвету с фоном.

Более эффектные для восприятия прямоугольники можно строить с помощью процедуры, которая рисует закрашенный прямоугольник:

```
Bar(x1,y1,x2,y2:integer);
```

Цвет закраски устанавливается с помощью **SetFillStyle**. Еще одна эффектная процедура –

```
Bar3D(x1,y1,x2,y2,d:integer;a:boolean);
```

– вычерчивает трехмерный закрашенный прямоугольник (параллелепипед). При этом используются тип и цвет закраски, установленные с помощью **SetFillStyle**. Параметр **d** представляет собой число пикселей, задающих глубину трехмерного контура. Чаще всего его значение равно четверти ширины прямоугольника (**d:=(x2-x1) div 4**). Параметр **a** определяет, строить над прямоугольником вершину (**a:=True**) или нет (**a:=False**).

Примеры использования:

```
SetColor(Green);
Rectangle(200,100,250,300);

SetFillStyle(1,3);
Bar(10,10,50,100);

SetFillStyle(1,3);
Bar3D(10,10,50,100,10,True);
```

10.4.6. Построение многоугольников

Многоугольники можно рисовать самыми различными способами, например, с помощью процедуры Line. Однако в Турбо Паскале имеется процедура **DrawPoly**, которая позволяет строить любые многоугольники линией текущего цвета, стиля и толщины. Она имеет формат

```
DrawPoly(a:word;var PolyPoints);
```

Параметр **PolyPoints** является нетипизированным параметром, который содержит координаты каждого пересечения в многоугольнике. Параметр **a** задает число координат в **PolyPoints**. Необходимо помнить, что для вычерчивания замкнутой фигуры с *N* вершинами нужно передать при обращении к процедуре **DrawPoly** *N*+1 координату, где координата вершины с номером *N* будет равна координате вершины с номером 1.

10.4.7. Построение дуг и окружностей

Процедура вычерчивания окружности текущим цветом имеет следующий формат:

```
Cicrle(x,y,r:word);
```

Здесь **x,y** – координаты центра окружности, **r** – ее радиус.

Например, фрагмент программы обеспечит вывод ярко-зеленой окружности с радиусом 50 пикселей и центром в точке (450, 100):

```
SetColor(LightGreen);
Circle(450,100,50);
```

Дуги можно вычертить с помощью процедуры

```
Arc(x,y:integer;a,b,R:integer);
```

Здесь **x,y** – центр окружности, **a,b** – начальный и конечный углы в градусах, **R** – радиус. Для задания углов используется полярная система координат.

Цвет для вычерчивания устанавливается процедурой **SetColor**. В случае a=0° и b=360°, вычерчивается полная окружность.

Например, выведем дугу красного цвета от 0° до 90° в уже вычерченной с помощью **Circle(450,100,50)** окружности:

```
SetColor(Red);
Arc(450,100,0,90,50);
```

Для построения эллиптических дуг предназначена процедура

```
Ellipse(x,y,a,b,Rx,Ry:integer);
```

Здесь **x,y** – центр эллипса, **Rx,Ry** – горизонтальная и вертикальная оси. В случае a = 0° и b = 360° вычерчивается полный эллипс. Например, построим голубой эллипс:

```
SetColor(9);
Ellipse(100,100,0,360,50,50);
```

Фон внутри эллипса совпадает с фоном экрана. Чтобы создать закрашенный эллипс, используется специальная процедура

```
FillEllipse(x,y:integer,Rx,Ry:integer);
```

Закраска эллипса осуществляется с помощью процедуры

```
SetFillStyle(a,b:word);
```

Здесь **a** – стиль закраски (см. табл. 10.4), **b** – цвет закраски (см. табл. 10.1). Например, нарисуем ярко-красный эллипс, заполненный редкими точками зеленого цвета:

```
SetFillStyle(WideDotFill,Green); {стиль заполнения}
SetColor(12);          {цвет вычерчивания эллипса}
FillEllipse(300,150,50,50);
```

Для построения секторов можно использовать следующие процедуры:

```
PieSlice(x,y:integer;a,b,R:word);
```

Рисует и заполняет сектор круга. Координаты **x,y** – центр окружности, сектор рисуется от начального угла **a** до конечного угла **b**, а закрашивание происходит при использовании специальных процедур;

```
Sector(x,y:integer;a,b,Rx,Ry:word);
```

Создает и заполняет сектор в эллипсе. Координаты **x,y** – центр, **Rx,Ry** – горизонтальный и вертикальный радиусы, и сектор вычерчивается от начального угла **a** до конечного угла **b**.

10.4.8. Стандартные стили заполнения

Таблица 10.4

Константа	Значение	Маска
EmptyFill	0	Заполнение цветом фона
SolidFill	1	Заполнение текущим цветом
LineFill	2	Заполнение символами --, цвет – color
LtslashFill	3	Заполнение символами // нормальной толщины, цвет – color
SlashFill	4	Заполнение символами // удвоенной толщины, цвет – color
BkslashFill	5	Заполнение символами \\ удвоенной толщины, цвет – color
LtbkSlahFill	6	Заполнение символами \\ нормальной толщины, цвет – color
HatchFill	7	Заполнение вертикально-горизонтальной штриховкой тонкими линиями, цвет – color
XhatchFill	8	Заполнение штриховкой крест-накрест по диагонали «редкими» тонкими линиями, цвет – color
InterLeaveFill	9	Заполнение штриховкой крест-накрест по диагонали «частыми» тонкими линиями, цвет – color
WideDotFill	10	Заполнение «редкими» точками
CloseDotFill	11	Заполнение «частыми» точками
UserFill	12	Заполнение по определенной пользователем маске заполнения, цвет – color

10.5. Построение графиков функций

Для построения графиков функций при помощи графического режима предполагается свободное владение студентами понятием функции, ее графическим и аналитическим представлением. Необходимо также использовать операторы цикла, которые помогут избежать однообразного труда по вычислению ординаты каждой точки.

До сих пор при создании рисунков использовали только первый квадрант системы координат. Для построения большинства функций в требуемом интервале изменения необходимо работать хотя бы в двух квадрантах. В общем случае полезно изображать систему координат в любой части плоскости, но наиболее наглядно располагать ее в центре экрана. В таких случаях, установив начало координат в точке (x_0, y_0) на экране, можно координаты (x, y) произвольной точки кривой определять разностью ($x - x_0$, $y - y_0$). После этого в программе можно употреблять не только положительные, но и отрицательные значения.

Рисунок получается маленьким, поэтому требуется увеличить масштаб изображения. Если для функции будет использован весь экран, надо увеличить рисунок по x и по y в зависимости от выбранного экрана.

Выбрать масштаб увеличения можно следующим образом:

- определить горизонтальный и вертикальный размеры графика (для этого вводятся границы области значений и определяются максимальное и минимальное значения функции на заданной области определения, затем вычисляются разности максимального и минимального значений аргументов и функции, которые и являются горизонтальным и вертикальным размерами графика соответственно);
- определить масштаб (сначала определяются масштабы изображения по горизонтали и вертикали) с учетом размеров выбранного экрана по формуле

масштаб(г/в) = размер экрана (по г/в) / размер графика (по г/в),

затем из них выбирается меньший, который и принимается за необходимый масштаб. В нашем случае графический экран имеет размеры 640 на 480.

В любом случае, чтобы высветить на экране точку, надо взять x, вычислить по данной абсциссе y и выполнить рисование точки. Так как на экране можно получить лишь ограниченное количество значений x, то их перебираем с помощью цикла.

Пример 10.1. Построить график функции $y = x^2$.

```
program parabola;
     uses graph,crt;
     var gd,gm: integer;
         x,y,mx,my,m,x1,x2,y1,y2,h:real;
function f(x:real):real;
begin
     f:= Sqr(x);
end;

begin
clrscr;
Writeln('Введите границы отрезка');
Write('x1='); Readln(x1);
Write('x2='); Readln(x2);
y1:=f(x1); y2:=f(x2);
mx:=640/(x2-x1);      {масштаб по x}
my:=480/(y2-y1);      {масштаб по y}
if mx<my then m:=mx
         else m:=my;  {min масштаб}
h:=1/m;               {определяем шаг}
x:=x1;
gd:=detect;       {инициализируем графический режим}
Initgraph(gd,gm,'c:\tp7\bgi');
```

```
SetColor(5);
Line(0,240,640,240); {чертим оси}
Line(320,0,320,480);
While x<=x2 do        {основной цикл}
    begin
    y:=-f(x);  {вычисляем значение функции}
    PutPixel(x*m+320,y*m+240,15); {выводим точку}
    x:=x+h;                       {делаем шаг}
    end;
end.
```

10.6. Построение графика аппроксимирующей функции

Для задачи нахождения аппроксимирующей функции по заданным точкам, рассмотренной в гл. 9, напишем процедуру, которая построит нам на экране график найденной функции и кружочками пометит исходные точки. Это наглядно продемонстрирует метод наименьших квадратов.

Напоминаю, что исходные точки находятся в двух массивах: **x[0..n]** и **y[0..n]**, а результат – в массиве **C[0..m]**, где **m** – порядок степенной функции. Это массив коэффициентов степенной функции

$$f(x) = C_0 + C_1 x + C_2 x^2 + ... + C_m x^m.$$

При описании процедуры эти массивы будут использованы как глобальные переменные. Так же в описании процедуры будет использована ранее описанная функция **F1** для вычисления значений полинома. В разделе описаний программы нужно не забыть подключить графический модуль

```
    uses graph;
```

Размер графического экрана принимаем 640 на 480 точек.

```
procedure grafic;
    var i,gd,gm,dx,dy: integer;
    x0,y0,miny,maxy,minx,maxx,mx,my,h:real;
begin
maxy:=y[0]; miny:=y[0];
maxx:=x[0]; minx:=x[0];
for i:=0 to n do     {определяем max и min}
    begin
    if y[i]<miny then miny:=y[i];
    if y[i]>maxy then maxy:=y[i];
    if x[i]<minx then minx:=x[i];
    if x[i]>maxx then maxx:=x[i];
    end;
```

```
mx:=640/(maxx-minx);     {масштаб по x}
my:=480/(maxy-miny);     {масштаб по y}
h:=1/mx;                 {определяем шаг по x}
x0:=minx;
gd:=detect;      {инициализируем графический режим}
Initgraph(gd,gm,'c:\tp7\bgi');
SetColor(5);
Rectangle(0,0,639,479);          {чертим обрамление}
dx:=round(x0*mx);          {определяем смещение по x}
dy:=round(miny*my);        {определяем смещение по y}
While x0<=maxx do          {основной цикл}
  begin
  y0:=f1(x0,m);
  {выводим точку аппроксимирующей функции}
  PutPixel(round(x0*mx)-dx,480-round(y0*my)+dy,14);
  x0:=x0+h;                {делаем шаг}
  end;
for i:=0 to n do      {выводим исходные точки}
 Circle(round(x[i]*mx)-dx,480-round(y[i]*my)+dy,3);
end;
```

11. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

11.1. Решение нелинейных уравнений

Часто при решении нелинейных уравнений невозможно выделить переменную. В этих случаях целесообразно применить ряд численных методов нахождения корней уравнения.

Любое уравнение можно представить в виде $f(x)=0$, перенеся всё в одну сторону, тогда поиск корней уравнения сводится к поиску точек пересечения функции $f(x)$ с осью абсцисс. Для более удобной реализации методов в языке Паскаль целесообразно сразу описать функцию $f(x)$ как подпрограмму:

```
function F(x:real):real;
begin
     F:=  .... ;
end;
```

В рамках изучаемой дисциплины предполагается изучить три метода решения нелинейных уравнений.

Метод половинного деления основан на поиске отрезка, содержащего корень и последующим уменьшением его размеров до достижения заданной точности вычислений. Уменьшение размеров отрезка осуществляется циклическим делением его пополам и отбрасыванием половинки, не содержащей корня.

Метод касательных предполагает произвольное задание начальной точки и последующее итерационное приближение этой точки к истинному значению корня до достижения заданной точности. Каждая последующая точка вычисляется, зная предыдущую точку и значение производной функции в этой точке.

Модифицированный метод Ньютона лишь немного отличается от метода касательных и обладает меньшей скоростью сходимости. Здесь значение производной вычисляется всего один раз в точке первого приближения и больше не изменяется.

Подробнее описание методов смотрите в разделе 4. Дополнительно рекомендуется использовать [2,3,9].

Задание на выполнение лабораторной работы № 1.

Тема : Решение нелинейных уравнений.

Цель : Изучение численных методов решения нелинейных уравнений.

Задачи : Освоить метод половинного деления, метод касательных и модифицированный метод Ньютона для решения нелинейного уравнения, научиться численно определить действенный корень нелинейного уравнения, составить алгоритм и соответствующую программу, развить практические навыки решения задач на ЭВМ.

Программа лабораторной работы.

1. Изучить численные методы решения нелинейных уравнений.
2. Составить алгоритм решения своего нелинейного уравнения по методу, предложенному преподавателем.
3. Составить программу на языке Pascal и получить допуск у преподавателя для работы на ПЭВМ.
4. Отладить программу.
5. Получить результаты вычислений.
6. Составить отчет и защитить работу у преподавателя.

Содержание отчета.

1. Постановка задачи, исходные данные.
2. Краткое описание метода решения нелинейных уравнений.
3. Текст программы.
4. Результаты вычислений.
5. Анализ результатов и выводы по работе.

Контрольные вопросы.

1. Приведите недостатки и преимущества используемого метода расчета по сравнению с другими известными методами.
2. Зависит ли значение искомого корня от выбора начальной точки для его поиска.
3. Как зависит значение функции, взятой в корне уравнения, от точности вычисления корня.
4. Какое значение функции, взятой в корне уравнения, мы ожидаем при предельной точности.

Варианты:

Две последние цифры номера зачётной книжки соответствуют вашему варианту.

Варианты			Задание	Метод решения
1	31	61	1	метод половинного деления
2	32	62	2	метод касательных
3	33	63	3	модифицированный метод Ньютона
4	34	64	5	метод половинного деления
5	35	65	6	метод касательных
6	36	66	7	модифицированный метод Ньютона
7	37	67	8	метод половинного деления
8	38	68	9	метод касательных
9	39	69	11	модифицированный метод Ньютона
10	40	70	12	метод половинного деления
11	41	71	1	метод касательных
12	42	72	2	модифицированный метод Ньютона
13	43	73	3	метод половинного деления
14	44	74	5	метод половинного деления
15	45	75	6	модифицированный метод Ньютона
16	46	76	7	метод половинного деления
17	47	77	8	метод касательных
18	48	78	9	модифицированный метод Ньютона
19	49	79	11	метод половинного деления
20	50	80	12	метод касательных
21	51	81	1	модифицированный метод Ньютона
22	52	82	2	метод половинного деления
23	53	83	3	метод касательных
24	54	84	5	метод половинного деления
25	55	85	6	метод половинного деления
26	56	86	7	метод касательных
27	57	87	8	модифицированный метод Ньютона
28	58	88	9	метод половинного деления
29	59	89	11	метод касательных
30	60	90	12	модифицированный метод Ньютона

Задания.

1. Определить с точностью ε температуру стенки печи T при радиационно-конвективном теплообмене стенки с греющей средой, используя выражение

$$q = \alpha(T_c - T) + \sigma (T_c^4 - T^4) ,$$

где q – плотность теплового потока, Вт/м^2 ; α – коэффициент конвективного теплообмена, Вт/м^{2}К ; $\sigma = \varepsilon_n \cdot \sigma_0$ – эффективный коэффициент теплообмена излучением, Вт/м^2·К^4 ; ε_n – приведенная степень черноты; σ_0 – постоянная Стефана-Больцмана, Вт/м^2·К^4 ; T_c – температура греющей среды, К.

Исходные данные : $q = 2 \cdot 10^4$; $\alpha = 30.2$; $\varepsilon_n = 0.28$; $\sigma_0 = 5.67 \cdot 10^{-8}$; $T_c = 1390$ К ; $\varepsilon = 0.01$ К .

2. При решении задач нестационарной теплопроводности в плоской неограниченной пластине необходимо определять корни μ характеристического уравнения

$$\text{ctg}\, \mu = \mu / Bi,$$

где $Bi = \alpha \cdot l / \lambda$ – критерий Био, α – коэффициент конвективного теплообмена, Вт/м^2·К; l – линейный размер (полутолщина пластины), м; λ – коэффициент теплопроводности материала пластины, Вт/м·К.

Вычислить с точностью ε два первых корня μ_1 и μ_2 характеристического уравнения ($0 < \mu_1 < \pi$, $\pi < \mu_2 < 2\pi$) .

Исходные данные: α=60.0, l=0.5, λ=10.0, ε=10^{-4} .

3. Определить с точностью ε относительную толщину γ динамического ламинарного пограничного слоя при обтекании пластины газом, используя выражения

а) $v = v_\infty [3/2 \cdot \gamma - 1/2 \cdot \gamma^3]$ б) $v = v_\infty [2 \cdot \gamma - 2 \cdot \gamma^3 + \gamma^4]$,

где v – скорость газа в поперечном сечении пограничного слоя, м/с; v_∞ – скорость невозмущенного потока газа, м/с.

Сравнить полученное значение с точным.

Исходные данные : v_∞=2, v=0.52, ε=10^{-4}.

4. Определить с точностью ε температуру самовоспламенения T горючей смеси, заключенной в сосуд объемом Vс площадью стенок F, используя выражения

$$Q_v = Q_w,\ \ Q_v = Q_g V,\ \ \ Q_w = \alpha F(T - T_w),\ \ \ g = k_0 \cdot c^n \cdot e^{-\frac{E}{R \cdot T}},$$

где Q_v – тепловыделение газовой смеси объема V в единицу времени, Вт; Q – тепловой эффект реакции горения, Дж/кмоль; g – скорость реакции, кмоль/(м^{3}с); α – коэффициент конвективного теплообмена, Вт/м^{2}К; k_0 – предэкспоненциальный множитель, кмоль/(м^{3}с); C – концентрация топлива в горючей смеси; n – порядок реакции горения; E – энергия активации, Дж/кмоль; R – универсальная газовая постоянная (8314 Дж/кмоль К); T_w – температура стенки сосуда, К.
Исходные данные получить у преподавателя.

5. При решении задач нестационарной теплопроводности в шаре необходимо определять корни μ характеристического уравнения

$$\mu = \operatorname{tg} \mu.$$

Вычислить с точностью $\varepsilon = 10^{-4}$ два корня μ_2 и μ_3 характеристического уравнения ($\pi < \mu_2 < 2\pi$, $2\pi < \mu_3 < 3\pi$).

6. Определить длину волны λ_m с точностью ε, которая соответствует максимальной плотности излучения, при температуре абсолютно черного тела Т в условиях термодинамического равновесия. В соответствии с законом смещения Вина

$$e^{\frac{-c_2}{\lambda_m \cdot T}} + \frac{c_2}{5\lambda_m \cdot T} - 1 = 0,$$

(примечание: для удобства обозначьте за $x = \dfrac{-C_2}{\lambda_m T}$)
где $c_2 = 1{,}4388 \cdot 10^{-2}$ мК –вторая константа излучения, $T = 1000$ К, $E = 10^{-2}$ мкм. После вычисления λ_m показать, что $\lambda_m \cdot T = 2{,}8978 \cdot 10^{-3}$.

7. При решении задач нестационарной теплопроводности в плоской неограниченной пластине при конвективном теплообмене необходимо определять корни μ характеристического уравнения

$$tg(\mu) = \frac{\mu(B_{i1} + B_{i2})}{\mu^2 - B_{i1}B_{i2}}$$

вычислить с точностью $\varepsilon=10^{-4}$ корень μ $(0.2 < \mu < 2.0)$ при заданных критериях Био: $B_{i1}=0.5$, $B_{i2}=0.9$.

8. В шаре радиуса $R = 0.05$ м выделяется энергия с мощностью $P = 100$ Вт . Выделяющаяся энергия с помощью конвекции и излучения передается в окружающую среду с температурой $T = 300$ К. Коэффициент теплоотдачи от поверхности шара $\alpha = 5$ Вт/м^{2}К , степень черноты его поверхности $\varepsilon_n = 0.7$. Требуется определить температуру поверхности шара с точностью $\varepsilon = 0.1$ К из уравнения теплового баланса

$$\alpha(T-T_c)+\varepsilon_n\sigma_0(T^4-T_c^4) = P/(4\pi R^2) ,$$

где $\sigma_0=5.67\cdot10^{-8}$ Вт/м$^2\cdot$К^4 – постоянная Стефана-Больцмана.

9. Определить с точностью ε температуру поверхности T твердого тела при радиационно-конвективном теплообмене, если известно экспериментально измеренное значение температуры $T_э$ в точке , расположенной на расстоянии h от поверхности. Для расчета использовать уравнение

$$q= \alpha(T_c-T) + \varepsilon_n\sigma_0(T_c^4-T^4) \quad , \text{или}$$

$$\lambda(T-T_э)/h = \alpha(T_c-T) + \varepsilon_n\sigma_0(T_c^4-T^4) ,$$

где λ – коэффициент теплопроводности материала тела, Вт/м·К;
h – глубина закладки датчика (термопары), м; T_c – температура греющей среды, К; α – коэффициент конвективного теплообмена, Вт/м$^2\cdot$К;
ε_n – приведенная степень черноты; $\sigma_0 = 5.67\cdot10^{-3}$ Вт/м^{2}К^4 – постоянная Стефана-Больцмана.
Исходные данные: $\lambda = 8.5$; $h = 8\cdot10^{-3}$; $T_э = 953.5$; $T_c = 1273$; $\alpha = 25$; $\varepsilon_n = 0.3$; $\varepsilon = 0.1$.

10. Определить с точностью ε температуру греющей среды T_c в камерной печи при нагреве слитков. Для расчета использовать уравнение теплового баланса печи в стационарном режиме

$$Q^p_н\cdot V + q_э = \alpha\cdot F\cdot n(T_c-T) + \varepsilon_n\cdot\sigma_0\cdot F\cdot n(T_c^4-T^4) + K(T_c-T_{ok}) + a\cdot T_c\cdot V .$$

Члены в левой части уравнения соответствуют приходным статьям теплового баланса, а в правой – расходным (затратным) статьям. Первый член в левой части уравнения характеризует поступающую в

печь теплоту в результате горения топлива, второй – теплоту экзотермических реакций из-за угара металла. Первый и второй члены правой части уравнения учитывают теплоту, переданную слиткам конвекцией и излучением, третий – определяет теплопотери через кладку печи, четвертый – теплоту уходящих продуктов сгорания. Здесь $Q_н^р$ – теплота сгорания топлива, Дж/м^3; V – расход топлива, м^3/с ; $q_э$ – тепловая мощность экзотермических реакций, Вт ; α – коэффициент конвективного теплообмена, Вт/м^{2}К ; F – площадь боковой поверхности одного слитка, м^2 ; n – количество слитков в печи ; T – температура поверхности слитков , К; ε_n – приведенная степень черноты поверхности слитка ; σ_o – постоянная Стефана-Больцмана ($5{,}67\cdot10^{-8}$ Вт/м^{2}К^4) ; K – соответствует коэффициенту теплопередачи, Вт/К; $T_{ок}$ – температура окружающей среды, К ; Q – поправочный коэффициент, Дж/м^{3}К . Исходные данные получить у преподавателя.

11. Обычно измерение термо-Э.Д.С. проводят двумя соединенными навстречу друг другу термопарами, например, термопарами N/N_i . При этом один спай термостатируют при 0 ^{0}С, в то время как другой имеет температуру объекта. Экспериментально полученную зависимость термо-Э.Д.С. от температуры в определенном температурном интервале часто описывают полиномом. Пусть разность Э.Д.С. V холодного и горячего спаев термопары и температура T горячего спая связаны следующей зависимостью :

$$T = 25{,}45V - 0{,}5592V^2 + 0{,}1045V^3 - 8{,}776\cdot10^{-3}V^4 + 3{,}76\cdot10^{-4}V^5 - -8{,}649\cdot10^{-6}V^6 .$$

Определить с точностью ε значение термо-Э.Д.С. V при заданной температуре $T_з$. Исходные данные : $T_з$=200 ^{o}С , ε=0,01 , 5<V<10 мВ .

12. При развитом турбулентном (хаотичном, неупорядоченном) течении связь между коэффициентом сопротивления ς и числом Рейнольдса Re в гладкой трубе имеет вид

$$\frac{1}{\sqrt{\varsigma}} = 0.88 \cdot \ln(\mathrm{Re}\sqrt{\varsigma}) - 0.9 ,$$

где $\mathrm{Re} = \frac{Ud}{\nu}$, U – скорость среды , ν – вязкость среды, d – характерный линейный размер. Величина ς хорошо аппроксимируется в области 10< Re< 10^5 эмпирической формулой Блазиуса

$$\varsigma = \frac{0.3164}{\mathrm{Re}^{0.25}} ,$$

а в области Re > 10^5 эмпирической формулой Никурадзе

$$\varsigma = 0.0032 + \frac{0.221}{\mathrm{Re}^{0.237}}$$

Определить с точностью ε коэффициент ς для Re = $5\cdot10^4$, Re =$5\cdot10^5$ и сравнить расчетные значения с данными, полученными по формуле Блазиуса и Никурадзе. Принять $\varepsilon = 10^{-5}$.

13. Мощность радиоактивного излучения пропорциональна концентрации радиоактивного вещества. Период полураспада *T* одного из изотопов углерода составляет 8 дней. В начальный момент времени мощность излучения Q_0 составляет 2*P/r* (1*P* = 2 .58·10^{-4} Кл/кг). Определить с точностью ε через сколько дней *t* мощность излучения *Q* снизится до 0.1 *P/r*. Изменение концентрации и, следовательно мощности излучения описывается формулой

$Q = Q_0 \cdot e^{-\lambda t}$, где $\lambda = \ln 2/T$.

Принять ε=1 день. Сравнять расчетное значение с точным.

11.2. Численное интегрирование

Задача вычисления интегралов возникает во многих областях прикладной математики. Суть методов численного интегрирования состоит в замене подынтегральной функции ***f(x)*** вспомогательной, интеграл от которой легко вычисляется в элементарных функциях.
Самыми простыми методами численного интегрирования являются методы прямоугольников. При этом непосредственно используется замена определенного интеграла суммой прямоугольников.
В методе трапеций определенный интеграл заменяется на сумму площадей прямоугольных трапеций.

Точность вычисления интеграла зависит от числа разбиений отрезка интегрирования, иначе говоря, от числа элементарных прямоугольников или трапеций.

Подробнее описание методов смотрите в разделе 5. Дополнительно рекомендуется использовать [2,3,4].

Задание на выполнение лабораторной работы N2.

Тема : Численное интегрирование.
Цель : Изучение методов численного интегрирования.
Задачи : Изучить методы численного интегрирования. Составить программу вычисления определенного интеграла на языке Pascal по методу, предложенному преподавателем и получить результат вычислений.

Оценить точность вычислений и его влияние на требуемое число разбиений интервала интегрирования.

Программа лабораторной работы.

1. Получить у преподавателя задание на проведение лабораторной работы в соответствии с номером варианта.
2. Полностью ознакомиться с методикой решения задачи и ответить на контрольные вопросы.
3. Составить программу. С целью достижения заданной точности алгоритм должен учитывать вычисление интеграла с удвоенным количеством шагов. Подынтегральная функция должна быть оформлена в виде функции языка Паскаль. Вычисление интеграла при заданном числе разбиений и границами интегрирования должно быть оформлено в виде процедуры. Начальное число разбиений и точность вычисления должна вводиться с клавиатуры.
4. Отладить программу, получить результат с точностью E . Расчеты проводить задавая различную точность вычислений.
5. Составить отчет.
6. Сдать отчет на проверку и защитить работу преподавателю.

Содержание отчета.

1. Постановка задачи, исходные данные.
2. Краткое описание метода вычисления интеграла.
3. Текст программы.
4. Результаты вычислений.
5. Анализ результатов и выводы по работе.

Контрольные вопросы

1. В чем заключается суть методов численного интегрирования?
2. Приведите известные вам методы численного интегрирования.
3. Как вычисляется интеграл с заданной точностью?
4. Как оценивается погрешность усечения?
5. Как влияют ошибки усечения и округления на результат вычислений?

Варианты:

Две последние цифры номера зачётной книжки соответствуют вашему варианту.

Варианты			Задание	Метод решения
1	31	61	1	метод правых прямоугольников
2	32	62	2	метод левых прямоугольников
3	33	63	3	метод средних прямоугольников
4	34	64	4	метод трапеций
5	35	65	5	метод правых прямоугольников
6	36	66	6	метод левых прямоугольников
7	37	67	7	метод средних прямоугольников
8	38	68	8	метод трапеций
9	39	69	9	метод правых прямоугольников
10	40	70	10	метод левых прямоугольников
11	41	71	11	метод средних прямоугольников
12	42	72	12	метод трапеций
13	43	73	13	метод правых прямоугольников
14	44	74	14	метод левых прямоугольников
15	45	75	15	метод средних прямоугольников
16	46	76	16	метод трапеций
17	47	77	1	метод трапеций
18	48	78	2	метод правых прямоугольников
19	49	79	3	метод левых прямоугольников
20	50	80	4	метод средних прямоугольников
21	51	81	5	метод трапеций
22	52	82	6	метод правых прямоугольников
23	53	83	7	метод левых прямоугольников
24	54	84	8	метод средних прямоугольников
25	55	85	9	метод трапеций
26	56	86	10	метод правых прямоугольников
27	57	87	11	метод левых прямоугольников
28	58	88	12	метод средних прямоугольников
29	59	89	13	метод трапеций
30	60	90	14	метод средних прямоугольников

Задания.

1. $$I = \int_0^{\pi/4} (1 - x \cdot \sin^2 t)^{-1/2} \; dt \quad ; \quad x = 1.5 \; ;$$

2. $\tau = K \int_{T_í}^{T_ê} 1/(\alpha(T_{ñð} - T) + \sigma(T_{ñç}^4 - T^4))\ dT$; $K = 9.0 \cdot 10^6$;

α=20; $\sigma = 0.2 \cdot 10^{-7}$; $T_н = 293$; $T_{ср} = 1273$; $T_к = 1173$

3. $I = \int_0^{0,5} \exp(-x^n)\ dx$; $n = 2$;

4. $I = \int_0^1 x^2 \cdot e^{ax}\ dx$; $a = 2.0$;

5. $I = \int_{1,5}^{3,5} (Ln(x))^n\ dx$; $n = 4$;

6. $I = \int_{0,1}^{0,3} tg^n(x)\ dx$; $n = 4$;

7. $I = \int_0^{\pi/2} \cos^n(x)\ dx$; $n = 5$;

8. $I = \int_0^{\pi/2} \sin^n(x)\ dx$; $n = 5$;

9. $I = \int_0^1 x^5/(a^3 + x^3)\ dx$; $a = 0.5$;

10. $I = \int_{0,1}^{1,1} x^2/(a^3 - x^3)\ dx$; $a = 0.4$;

11. $I = \int_{\pi/6}^{\pi/2} \frac{\cos(x)}{\sin^3(x)}\ dx$;

12. $I = \int_{\pi/6}^{\pi/2} \frac{\cos^3(x)}{\sin(x)}\ dx$;

13. $I = \int_{0,1}^{0,5} x \cdot sh(x)\ dx$;

14. $I = \int_{0,5}^1 sh^2(x)\ dx$;

15. $I = \int_{1}^{2} x \cdot ch(x)\ dx$;

16. $I = \int_{0}^{1} Ln(1+x)\ dx$;

11.3. Решение систем линейных алгебраических уравнений

Методы решения СЛАУ делятся на две группы – прямые и итерационные.

Прямые методы используют заранее известное, зависящее от числа уравнений, количество соотношений (формул) для вычисления не известных. К ним относятся правило Крамера, метод Гаусса или метод последовательного исключения неизвестных, метод Гаусса с выбором главного элемента, метод прогонки и др.

Итерационные методы используют последовательные приближения (итерации, итерационные циклы), в процессе вычисления по которым вычисляемые корни сходятся к некоторому пределу – значению корней, вычисленных с заданной точностью. К таким методам относятся метод простых итераций, метод Зейделя и др.

Подробнее описание методов смотрите в разделе 8. Дополнительно рекомендуется использовать [2,3].

Задание на выполнение лабораторной работы N3.

Тема : Решение системы линейных алгебраических уравнений (СЛАУ).
Цель : Изучение численных методов решения СЛАУ.
Задачи : Изучить численные методы решения СЛАУ. Составить программу решения СЛАУ на языке Pascal по методу, предложенному преподавателем и получить результат вычислений. Исходные данные ввести из файла.

Программа лабораторной работы.

1. Изучить численные методы решения СЛАУ.
2. Составить алгоритм решения СЛАУ методом, согласно варианту.
3. Составить программу на языке Pascal. Исходные данные должны вводиться из файла.
4. Отладить программу.
5. Получить результаты вычислений.
6. Составить отчет и защитить работу у преподавателя.

Содержание отчета.

1. Задание.
2. Краткое описание метода решения СЛАУ.
3. Текст программы. Текст файла исходных данных.
4. Результаты вычислений.
5. Выводы.

Варианты:

Две последние цифры номера зачётной книжки соответствуют вашему варианту.

Варианты			Задание	метод решения
1	31	61	1	метод простых итераций
2	32	62	2	метод Зейделя
3	33	63	3	метод Гаусса
4	34	64	4	метод простых итераций
5	35	65	5	метод Зейделя
6	36	66	6	метод Гаусса
7	37	67	7	метод простых итераций
8	38	68	8	метод Зейделя
9	39	69	9	метод Гаусса
10	40	70	10	метод простых итераций
11	41	71	1	метод Зейделя
12	42	72	2	метод Гаусса
13	43	73	3	метод простых итераций
14	44	74	4	метод Зейделя
15	45	75	5	метод Гаусса
16	46	76	6	метод простых итераций
17	47	77	7	метод Зейделя
18	48	78	8	метод Гаусса
19	49	79	9	метод простых итераций
20	50	80	10	метод Зейделя
21	51	81	1	метод Гаусса
22	52	82	2	метод простых итераций
23	53	83	3	метод Зейделя
24	54	84	4	метод Гаусса
25	55	85	5	метод простых итераций
26	56	86	6	метод Зейделя
27	57	87	7	метод Гаусса
28	58	88	8	метод простых итераций
29	59	89	9	метод Зейделя
30	60	90	10	метод Гаусса

Задания.

1	$13x_1 + 2{,}1x_2 + 4x_3 + 0{,}8x_4 + 3x_5 = 9{,}2$ $1{,}3x_1 + 12x_2 - 4x_3 + 8x_4 - 0{,}3x_5 = 2$ $3x_1 + 1x_2 + 4x_3 - 0{,}8x_4 = 0{,}8$ $2{,}1x_2 + 4x_3 + 8x_4 - 3x_5 = 2{,}4$ $x_1 - 2{,}1x_2 + 1{,}8x_4 + 3x_5 = 1{,}2$
2	$13x_1 + x_2 + 0{,}4x_3 + 0{,}6x_4 + 0{,}11x_5 = 7{,}6$ $3x_1 + 10x_2 - 0{,}6x_3 + 3x_5 = 8{,}7$ $x_1 - 2{,}1x_2 + 4{,}9x_3 + 2{,}8x_4 = 9{,}2$ $1{,}1x_1 - x_2 + 18x_4 - 3x_5 = 5{,}5$ $x_1 + x_2 + x_3 + x_4 + 5x_5 = 4{,}2$
3	$15x_1 - 2{,}13x_2 - 0{,}24x_3 + 0{,}18x_4 + 3{,}7x_5 = 4{,}01$ $x_1 + 24x_2 - 4{,}5x_3 + 0{,}3x_4 - 6{,}4x_5 = 8{,}71$ $0{,}58x_1 + 0{,}54x_2 - 1{,}56x_3 + 0{,}98x_4 + x_5 = 6$ $x_1 - x_3 + 7x_4 + 4x_5 = 7$ $- x_1 + 0{,}5x_2 - x_3 + 1{,}7x_4 + 3{,}5x_5 = 0{,}54$
4	$5x_1 - 4x_2 + 1{,}9x_3 - x_4 + x_5 = 0{,}85$ $x_1 + 6x_2 - x_3 + x_4 + x_5 = 2{,}86$ $x_1 + x_2 + x_3 + x_4 + x_5 = 0{,}67$ $- x_1 + 3{,}14x_3 + 11x_4 - x_5 = 5{,}81$ $x_1 + 0{,}52x_2 + 3{,}95x_3 - 2x_4 + 8x_5 = 4$
5	$x_1 + 0{,}85x_2 - 0{,}24x_3 + 0{,}53x_4 + 0{,}11x_5 = 0{,}32$ $0{,}55x_1 + 1{,}42x_2 + 0{,}45x_3 - 0{,}75x_4 + 0{,}91x_5 = 1{,}52$ $0{,}66x_1 + 0{,}56x_2 + 2{,}11x_3 - x_4 - x_5 = 0{,}84$ $0{,}99x_1 + 1{,}57x_2 - x_3 + 3{,}54x_4 - 2{,}16x_5 = 1{,}78$ $- 1{,}8x_1 + 2x_2 - 4{,}1x_3 + 3{,}4x_4 + 6{,}3x_5 = 5$
6	$13x_1 - 3{,}1x_2 + 4x_3 + 8x_4 + 3x_5 = 9{,}2$ $1{,}3x_1 + 12x_2 - 4x_3 + 8x_4 - 0{,}3x_5 = 2$ $3x_1 - 1x_2 + 4x_3 - 2{,}8x_4 = 0{,}8$ $2{,}1x_2 + 4x_3 + 8x_4 - 3x_5 = 2{,}4$ $x_1 - 1{,}1x_2 + 1{,}8x_4 + 3x_5 = 1{,}2$
7	$13x_1 + x_2 + 0{,}4x_3 + 0{,}6x_4 + 0{,}11x_5 = 7{,}6$ $3x_1 + 11x_2 - 0{,}6x_3 + 1{,}3x_5 = 8{,}7$ $x_1 - 2{,}1x_2 + 4{,}9x_3 + 2{,}8x_4 = 9{,}2$ $1{,}1x_1 - x_2 + 18x_4 + 3x_5 = 5{,}5$ $x_1 + x_2 + x_3 + x_4 + 5x_5 = 2{,}2$
8	$15x_1 - 2{,}13x_2 - 0{,}24x_3 + 0{,}18x_4 + 3{,}7x_5 = 4{,}01$ $x_1 + 24x_2 - 4{,}5x_3 + 0{,}48x_4 - 6{,}4x_5 = 8{,}7$ $0{,}58x_1 + 0{,}54x_2 - 1{,}56x_3 + 0{,}98x_4 + x_5 = 6$ $x_1 - 0{,}3x_3 + 7x_4 + 4x_5 = 7$ $x_1 + 0{,}5x_2 - x_3 + 1{,}7x_4 + 3{,}5x_5 = 0{,}54$

9	$5x_1 - 2{,}4x_2 - 1{,}9x_3 - x_4 + x_5 = 0{,}85$ $x_1 + 6x_2 - x_3 + x_4 + x_5 = 2{,}86$ $x_1 + x_2 + x_3 - x_4 + x_5 = 0{,}67$ $- x_1 - 3{,}14x_3 + 11x_4 - x_5 = 5{,}81$ $x_1 + 0{,}52x_2 + 3{,}95x_3 - 2x_4 + 8x_5 = 4$
10	$x_1 + 0{,}8x_2 - 0{,}4x_3 + 0{,}3x_4 + 0{,}11x_5 = 0{,}32$ $0{,}55x_1 + 1{,}42x_2 - 0{,}4x_3 - 0{,}75x_4 + 0{,}91x_5 = 1{,}52$ $0{,}66x_1 + 0{,}5x_2 + 2{,}1x_3 - x_4 - x_5 = 0{,}84$ $0{,}9x_1 - 1{,}57x_2 - x_3 + 3{,}54x_4 - 2{,}16x_5 = 1{,}78$ $- 1{,}8x_1 - 2x_2 - 4{,}1x_3 + 3{,}4x_4 + 6{,}3x_5 = 5$

11.4. Аппроксимация функцией. Метод наименьших квадратов

Подробнее описание метода смотрите в разделе 9. Дополнительно рекомендуется использовать [2,3].

Задание на выполнение лабораторной работы N4

Тема : Метод наименьших квадратов (МНК).
Цель : Изучение метода и практическое его применение для обработки данных.
Задачи : Изучить метод наименьших квадратов. Составить программу на языке Pascal для нахождения аппроксимирующей функции по исходным точкам, полученным в результате эксперимента. Исходные данные ввести из файла.

Программа лабораторной работы.

1. Изучить метод наименьших квадратов.
2. Составить алгоритм решения задачи.
3. Составить программу на языке Pascal.
4. Отладить программу.
5. Получить результаты вычислений.
6. Начертить график аппроксимирующей функции (желательно на экране компьютера) и нанести исходные точки. Оценить погрешность.
7. Составить отчет о проделанной работе.

Содержание отчета.

1. Задание.
2. Краткое описание МНК.
3. Блок-схема алгоритма.
4. Текст программы. Текст файла исходных данных.
5. Результаты вычислений.

6. График аппроксимирующей функции.
7. Выводы по работе.

Варианты и исходные данные.

Две последние цифры номера зачётной книжки соответствуют вашему варианту.

Варианты			Экспериментально полученные точки (X,Y).								
1	21	41	X	1	2	3	4	5	6	7	8
			Y	0.1	-0.1	0.5	1.1	2.1	3	4	10
2	22	42	X	1	2	3	4	5	6	7	8
			Y	4.9	9.3	10	23	20	29	14.2	10.4
3	23	43	X	1	2	3	4	5	6	7	8
			Y	5.5	4.8	2.0	0.3	1.5	2.1	4.3	10
4	24	44	X	1	2	3	4	5	6	7	8
			Y	77	23	45	20	12	5.4	3.1	0.3
5	25	45	X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
			Y	-0.4	-0.1	0.5	1.1	2	5	4	10
6	26	46	X	-0.1	-0.2	-0.3	-0.4	-0.5	-0.6	-0.7	-0.8
			Y	-0.4	-0.1	0.5	1.1	2	5	4	10
7	27	47	X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
			Y	10	12	9.9	6.4	3.2	1.3	-1.5	-2.0
8	28	48	X	-0.1	-0.2	-0.3	-0.4	-0.5	-0.6	-0.7	-0.8
			Y	10	12	9.9	6.4	3.2	1.3	-1.5	-2.0
9	29	49	X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
			Y	76	34	40	22	20	18	10	5
10	30	50	X	-0.1	-0.2	-0.3	-0.4	-0.5	-0.6	-0.7	-0.8
			Y	76	34	40	22	20	18	10	5
11	31	51	X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
			Y	4.9	9.3	10	23	20	29	14.2	10.4
12	23	52	X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
			Y	0.1	-0.1	-0.5	1.1	2.1	3	4	10
13	33	53	X	-0.1	-0.2	-0.3	-0.4	-0.5	-0.6	-0.7	-0.8
			Y	4.9	9.3	10	23	20	29	14.2	10.4
14	34	54	X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
			Y	5.5	4.8	2.0	0.3	-1.5	2.1	4.3	10
15	35	55	X	-0.1	-0.2	-0.3	-0.4	-0.5	-0.6	-0.7	-0.8
			Y	5.5	4.8	2.0	0.3	-1.5	2.1	4.3	10
16	36	56	X	-0.1	-0.2	-0.3	-0.4	-0.5	-0.6	-0.7	-0.8
			Y	0.1	-0.1	0.5	1.1	2.1	3	4	10
17	37	57	X	1	2	3	4	5	6	7	8
			Y	7.8	3.1	-7.7	-10	-22	-5.6	0	-1.6
18	38	58	X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
			Y	7.8	3.1	-7.7	-10	-22	-5.6	0	-1.6

19	39	59	X	-0.1	-0.2	-0.3	-0.4	-0.5	-0.6	-0.7	-0.8
			Y	7.8	3.1	-7.7	-10	-22	-5.6	0	-1.6
20	40	60	X	1	2	3	4	5	6	7	8
			Y	-0.4	-0.1	0.5	1.1	2	5	4	10

ЗАКЛЮЧЕНИЕ

Учебное пособие не позволяет рассмотреть подробно все вопросы программирования на языке Паскаль. Мы с вами рассмотрели лишь основные, наиболее часто используемые, приёмы программирования. Курс позволяет научиться и начать составлять программы для инженерных расчётов. Освоение других численных методов решения задач предполагается в рамках смежных курсов «Методы оптимизации и расчеты на ЭВМ технико-экономических задач», «Математическое моделирование и расчеты теплотехнических систем на ЭВМ», «Математические методы моделирования физических процессов». Рекомендуется также воспользоваться литературой [1,2,5–8].

Что же касается дальнейшего углубления знаний языка Паскаль, то это выносится на самостоятельное изучение. Можно использовать любую литературу по языку Паскаль, однако наиболее удачно материал изложен в [3]. Прежде всего, это касается работы с типами данных, которые редко используются в технических расчётах. В пособии совсем не рассмотрены типы данных «запись» и «указатели», а также объектно-ориентированное программирование. Отчасти этому есть разумное объяснение. Тип «запись» обычно применяется для программирования баз данных. Сейчас разрабатывать базы данных в Паскале совсем не актуально, т. к. существуют более простые и эффективные средства проектирования баз данных. Изучать объектно-ориентированное программирование и работу с «указательным» типом данных в рамках языка Паскаль тоже нерационально. Гораздо более полезно перейти к изучению среды программирования Delphi и изучить эти темы там.

Пусть изучение языка Паскаль в рамках данного курса послужит вам хорошей базой для дальнейшего изучения современных средств программирования.

СПИСОК ЛИТЕРАТУРЫ

Основная

1. Кацман Ю. Я. Прикладная математика. Численные методы. Учебное пособие.–Томск: Изд-во. ТПУ, 2000.–68 с.
2. Мудров А. Е. Численные методы для ПЭВМ на языках Бейсик, Фортран, Паскаль. –Томск: МП «РАСКО», 1991.–227 с.
3. Офицеров Д. В., Старых В. А. Программирование в интерактивной среде Турбо-Паскаль: Справ. пособие.–Мн.: Беларусь, 1992.–240 с.: ил.

Дополнительная

4. Двайт Г. Б. Таблицы интегралов и другие математические формулы. М.: Наука, 1977.–228 с.
5. Бахвалов Н. С., Жидков Н. Л., Кобельков Г. М. Численные методы.– М.: Наука, 1987.–598 .
6. Щуп Т. Решение инженерных задач на ЭВМ.–М.: Мир, 1982.–235 с.
7. Меликов В. Т., Кветный Р. Н. Вычислительные методы и применение ЭВМ: учебное пособие.–Киев: Высш. шк., 1989.– 213 с.
8. Данилина Н. И., Дубровская Н. С., Кваша О. Л. и др. Численные методы: учебник для техникумов.–М.: Высш. шк., 1976.–368 с.
9. Кутателадзе С. С. Основы теории теплообмена.– М.: Гос. Машиностроение, 1962. – 456 с.