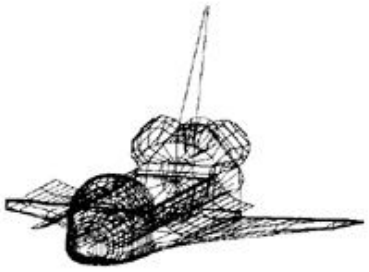
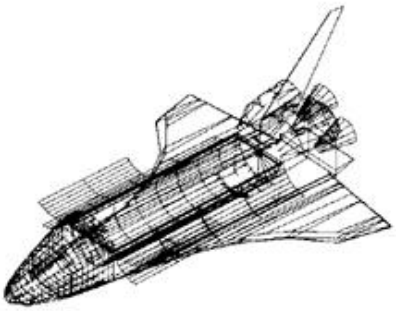
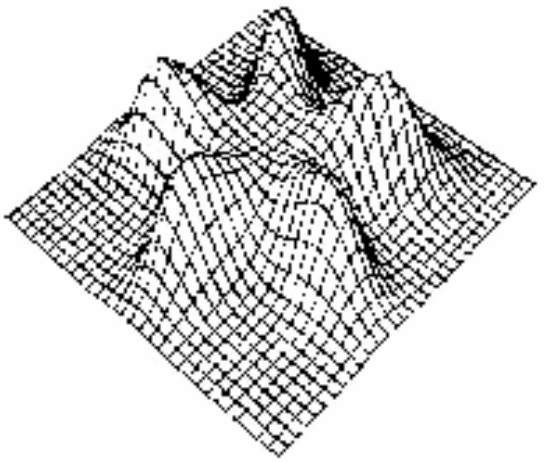




ИНФОРМАТИКА

ПРОГРАММИРОВАНИЕ И ЧИСЛЕННЫЕ МЕТОДЫ

Лабораторный практикум



Краснодар
2010

Министерство образования и науки Российской Федерации
КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКА

ПРОГРАММИРОВАНИЕ И ЧИСЛЕННЫЕ МЕТОДЫ

Лабораторный практикум

Краснодар
2010

УДК 004 (076.5)
ББК 22.18 я7
И 741

Рецензент:
Кандидат химических наук, доцент
С.Н. Болотин

И 741. Информатика. Программирование и численные методы: лабораторный практикум / сост. В.А. Волынкин, И.В. Сухно, В.Ю. Бузько. Краснодар: Кубанский государственный университет, 2010. 75 с.

Предлагаемый лабораторный практикум содержит указания к выполнению практических работ по второй части курса «Информатика», выполнение которых позволит студентам получить и закрепить знания по основным приемам программирования, разработки алгоритмов, научиться применять полученные навыки с учетом специфики основной специальности.

Адресуется студентам I, II курсов, специализирующимся по направлениям 020100.62 – Химия, 200500.62 – Метрология, стандартизация и сертификация, 260500.62, 260501.65 – Технология продуктов общественного питания, 280200.62 – Защита окружающей среды.

УДК 004(076.5)
ББК 22.18 я7
И 741

ВВЕДЕНИЕ

> В наши дни у людей существует странное мнение, что будто бы всему можно научиться, слушая лекции. Между тем я не могу признать, что лекции приносят такую же пользу, как чтение тех книг, по которым составлены эти лекции.
>
> *Д-р Джонсон (1766)*

Профессор Технического университета в Цюрихе Н. Вирт в 1971 г. создал язык программирования и назвал его в честь Блеза Паскаля – PASCAL. При создании языка программирования PASCAL, учитывая его средства и возможности, Н. Вирт большое внимание уделял хорошему стилю программирования. Стиль программирования – это выражение опыта общения людей, занимающихся разработкой и использованием программ, опыта, выработанного в результате многолетней практики такого общения.

В частности, программы, разработанные программистом, должны быть хорошо прокомментированы, интерфейс понятен пользователю. Каждый объект программы: константы, переменные, процедуры, функции и т.п. имеют свое, отличное от других объектов, имя (идентификатор). Правильный выбор идентификатора позволяет сделать программу гораздо понятнее, уменьшить число комментариев в ней. При написании идентификаторов программист должен помнить и придерживаться следующих правил:

- согласные важнее гласных;
- начало слова важнее его конца;
- начальные буквы всегда должны быть включены в сокращение.

Программист должен знать, что от того, как он разместит операторы в программе, существенно зависит наглядное восприятие программы. Такое правило не только улучшает читаемость программы, но и облегчает исправление возможных ошибок. Рекомендуется делать отступы разных уровней от левого края программы. Следует помнить, что существуют основные принципы создания программ, но индивидуальный стиль программиста и творческий подход оказывают огромное влияние на полученную программу.

АЛГОРИТМЫ

Алгоритм — точное и понятное предписание исполнителю совершить последовательность действий, направленных на решение поставленной задачи. Алгоритмы могут быть записаны в вербальной форме (словами), в графической форме (в виде блок-схем) и в виде программного кода.

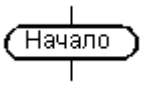
Вербальный (словесный) способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке. Например, алгоритм нахождения **наибольшего общего делителя (НОД)** двух натуральных чисел:

1) задать два числа;
2) если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма;
3) определить большее из чисел;
4) заменить большее из чисел разностью большего и меньшего из чисел;
5) повторить алгоритм с шага 2.

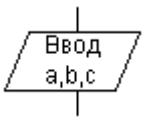
Графический способ представления алгоритмов более компактный и наглядный по сравнению с вербальным. При графическом представлении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий. Такое графическое представление называется схемой алгоритма или блок-схемой.

В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде блочного символа. Блочные символы соединяются линиями переходов, определяющими очередность выполнения действий. Наиболее часто используемые элементы блок-схем приведены далее.

Начало (конец) алгоритма

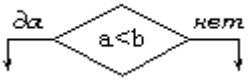


Ввод / вывод данных

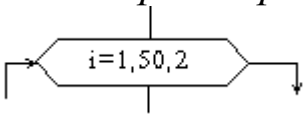


Вывод на печать

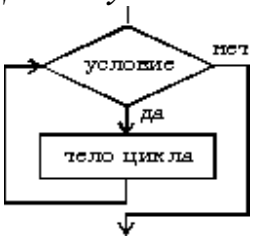
Ветвление по условию



Цикл с параметром



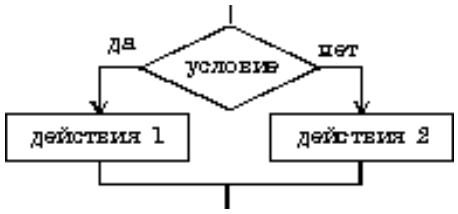
Цикл с условием



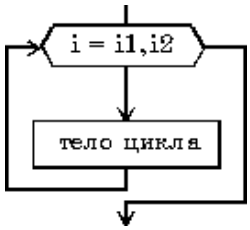
Алгоритмы можно представлять как некоторые структуры, состоящие из отдельных **базовых** (т.е. основных) **элементов.** Все алгоритмы строятся на основе трех основных структур

следование

ветвление



цикл



При записи алгоритма в словесной форме, или в виде блок-схемы, или псевдокода допускается определенный произвол при изображении команд. Вместе с тем такая запись точна настолько, что позволяет человеку понять суть дела и исполнить алгоритм.

Однако на практике алгоритм, предназначенный для исполнения на компьютере, должен быть записан на «понятном» ему языке. И здесь на первый план выдвигается необходимость точной записи команд, не оставляющей места для произвольного их толкования. Следовательно, язык для записи алгоритмов должен быть формализован. Такой язык принято называть языком программирования, а запись алгоритма на этом языке — программой для компьютера.

В настоящее время в мире существует несколько сотен реально используемых языков программирования. Для каждого есть своя область применения. В зависимости от степени детализации предписаний обычно определяется уровень языка про-

граммирования — чем меньше детализация, тем выше уровень языка. По этому критерию можно выделить следующие уровни языков программирования:

- машинные;
- машинно-ориентированные (ассемблеры);
- машинно-независимые (языки высокого уровня).

Машинные языки и машинно-ориентированные языки — это языки низкого уровня, требующие указания мелких деталей процесса обработки данных. Языки же высокого уровня имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы. Эти языки более удобны для человека. Языки высокого уровня делятся на:

- алгоритмические (Basic, Pascal, C и др.), которые предназначены для однозначного описания алгоритмов;
- логические (Prolog, Lisp и др.), в отличие от алгоритмических, ориентированы не на разработку алгоритма решения задачи, а на систематическое и формализованное описание задачи с тем, чтобы решение следовало из составленного описания;
- объектно-ориентированные (Object Pascal, C++, Java и др.). В основе лежит понятие объекта, сочетающего в себе данные и действия над ними. Описание задачи в форме системы взаимодействующих объектов естественнее, чем в форме взаимодействующих процедур.

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PASCAL

Компиляторы, интерпретаторы. Понятие программы

Язык программирования Pascal относится к алгоритмическим языкам высокого уровня. Программа на таком языке представляет собой формальную запись некоторого алгоритма (конечной последовательности действий, приводящих к решению некоторой задачи). Запись ведется в одном (или несколько) текстовых файлах с расширением *.pas.

Программа состоит из двух основных частей: описания последовательности действий, которые необходимо выполнить, и описания данных, которыми оперируют эти действия. Действия

представляются операторами языка, данные вводятся посредством описаний и определений. Описания данных текстуально предшествуют описанию действий и должны содержать упоминание всех объектов, используемых в действиях (операторах). Это необходимо для корректного размещения программы в памяти компьютера.

Компьютеры работают с информацией, представленной в двоичном коде; программа, непосредственно исполняемая компьютером, представляет собой машинный код. Поэтому для работы программы, написанной на языке высокого уровня, используются два типа программ, называемые соответственно компиляторами и интерпретаторами. Последние пошагово обрабатывают и выполняют строки программы. Компиляторы преобразуют текст программы в машинный код и записывают его в файл (с расширением *.exe), который можно загрузить в оперативную память и затем выполнить. При загрузке программы выделяется память для сегмента команд, для хранения данных: переменных, констант (сегмент данных) и организации стека[1].

Структура текста программы

Любая программа на языке Pascal состоит из раздела описаний и раздела операторов. В разделе описаний перечисляют все величины (константы, переменные и т.д.) и подпрограммы (процедуры, функции), используемые в программе. Раздел описаний обычно состоит из нескольких секций. Каждая секция начинается со слова, которое ее характеризует (**`program, uses, const, var`** …). Программа может содержать столько разделов описаний, сколько нужно для корректного описания всех объектов.

В разделе операторов записывается собственно алгоритм программы. Он начинается со служебного слова **`begin`** и заканчивается служебным словом **`end`**. Раздел операторов может быть только один. Компилятор не рассматривает текст, расположенный после финального **`end`** (end с точкой).

[1] Стек – область памяти программы, используемая для организации механизма передачи параметров в процедурах и функциях, а также некоторых других целей.

Заголовок программы	**Program** <name>;	*Имя программы (должно соответствовать правилам именования идентификаторов)*
Описание меток	**Label** 1, 2, out;	*Совокупность идентификаторов и/или целых чисел, предназначенных для организации последовательности вычислений*
Описание констант	**Const** Pi=3.14; S ='строка';	*Имена и значения постоянных величин, которые будут использоваться в программе*
Описание типов	**Type** MyType= **Array** [1..10]**of** real;	*Предназначено для задания конкретных множеств значений. Указанные множества обозначаются именами (идентификаторами) и в дальнейшем могут служить для описания переменных*
Описание переменных	**Var** I, j :integer; A, b :real; C :MyType; D:**record** A,H:longint; **End**;	*Служит для указания имен и типа переменных. Переменная — это величина, значение которой может меняться в процессе выполнения программы. Переменная обозначается идентификатором; с каждой переменной связывается ее тип, определяющий множество допустимых значений этой переменной и соответственно набор допустимых операций*[2]
Описание процедур и функций	**Procedure** MyProc; **function** MyFunc:real;	*Определяет часть программы как отдельную синтаксическую единицу и сопоставляет с ней имя*
Тело программы (раздел операторов)	**Begin** ... ... **end.**	*В разделе операторов, который всегда начинается со слова* **begin***, последовательно указываются те действия, которые должна выполнить программа. Заканчивается программа словом* **end.** *(с точкой) — это также обязательное требование*

[2] Чаще всего значениями переменных являются числа. Но это не единственная возможность: значением может быть буква, строка символов, допустимы и другие варианты. Для каждой переменной множество возможных ее значений строго фиксировано и называется **типом** переменной. Тип переменной также обязательно указывается при ее описании в секции **var**.

Числа

Для записи чисел в Pascal используются три формы:

– целое число записывается в виде последовательности цифр и знака минус (для отрицательных чисел). Целая константа в шестнадцатеричном формате имеет в качестве префикса знак доллара ($): `0 345 -15;`

– для записи дробных чисел используется десятичная точка: `3.1415926 -2.718;`

– очень большие и очень маленькие числа обычно записываются в экспоненциальной форме[3]:
`6.022e23 (= 6.022*10^23)  -5E-7 (= -0.0000005).`

Идентификаторы

Для обозначения типов, имен переменных, процедур и функций используются идентификаторы. Идентификатором может быть любая последовательность из латинских букв, цифр, знаков подчеркивания (значимыми являются только первые 63 символа). Другие символы (например, пробелы) в именах использовать нельзя! Имя не может начинаться с цифры. Внутри имен недопустимы пробелы. Прописные и строчные буквы не различаются. В качестве имен каких-либо объектов программы нельзя использовать зарезервированные слова (**`var, begin, uses`**…).

Типы данных

В языке Pascal (как и во многих других) для любой используемой переменной должен быть указан её тип. Это связано с отведением памяти под переменные и последующими операциями над переменными различных типов (совместимость типов).

Иерархия типов в языке Pascal следующая:

[3] Латинские буквы «E» и «e» не различаются! Показатель степени, который ставится после *латинской* буквы «E», может быть только целым. Мантисса должна обязательно присутствовать. Например, число 10^9 записывается как 1e9, а запись e9 воспринимается как имя переменной, а не как число.

Существует семь встроенных (предопределенных) порядковых типов: **integer**, **shortint**, **longint**, **byte**, **word**, **boolean** и **char**. Кроме того, имеется два других класса определяемых пользователем порядковых типов: перечислимые типы и отрезки типов (интервалы). К любому значению порядкового типа можно применить стандартную функцию **Ord**, возвращающую порядковый номер этого значения, а также функцию **Pred**, возвращающую предшествующее этому значению значение. Если эта функция применяется к первому значению в этом порядковом типе, то выдается сообщение об ошибке. К любому значению порядкового типа можно применить стандартную функцию **Succ**, возвращающую следующее за этим значением значение. Если эта функция применяется к последнему значению в этом порядковом типе, то выдается сообщение об ошибке.

Логические (булевские величины)

Описатель типа	Длина (байт)	Кол-во значений	Допустимые значения
`boolean`	1	2	`true, false`

Целые числа

Описатель типа	Длина (байт)	Минимальное число	Максимальное число
integer	2 (знак)	-32768	+32767
Shortint	1 (знак)	-128	+127
longint	4 (знак)	-2147483648	+21474836474
byte	1(б/зн.)	0	255
word	2(б/зн.)	0	65535

Вещественные числа

Описатель типа	Длина (байт)	Диапазон значений	Число цифр мантиссы
Real	6	$2 \cdot 10^{-39}$ .. $1.7 \cdot 10^{38}$	11-12
Single	4	$1.5 \cdot 10^{-45}$..$3.4 \cdot 10^{38}$	7-8
Double	8	$5.0 \cdot 10^{-324}$..$1.7 \cdot 10^{308}$	15-16
Extended	10	$3.4 \cdot 10^{-4932}$..$1.1 \cdot 10^{-4932}$	19-20
Comp	8	(цел. число,64-bit)	-

Литеры (символьные величины)

Описатель типа	Длина, байт	Кол-во значений	Допустимые значения
Char	1	256	литера (символ)

Интервалы

```
type имя_типа =
            мин_целое_значение .. макс_целое_значение;
```

Интервалом называется тип, содержащий ограниченное количество порядковых значений. Например, можно создать тип TColors, содержащий номера цветов, которые можно использовать в графике (для версии Turbo Pascal). Таких цветов 16: от 0 до 15. Соответственно тип будет определен как

```
type Tclolors = 0 .. 15;
```

Перечисления

```
type имя_типа=(значение1, значение2, ... значениеN);
```

Перечисления позволяют создавать для значений определенные названия. Например, для цветов из предыдущего примера удобнее создать перечисление, в котором будут не порядковые номера, а названия цветов. Название каждого цвета кодирует его номер. Нумерация начинается с 0.

```
type Tclolors = (clBlack, clRed, clGreen, clYel-
low,clBlue,clGrey);
```

В данном случае присвоение значения переменной может выглядеть так:

```
ScreenColor := clGreen;
```

Что эквивалентно:

```
ScreenColor:= 2;
```

Строки

Строки используются для хранения символьной информации. Строка символов представляет собой последовательность, содержащую нуль и более символов из расширенного набора символов кода ASCII, записанную в одной строке программы и заключенную *в одиночные кавычки* (апострофы). Строка символов, ничего не содержащая между апострофами, называется нулевой строкой. Два последовательных апострофа в строке символов обозначают один символ – апостроф. В первом байте строки располагается информация о её длине (не более 255 символов), далее следует собственно набор символов (коды ASCII), определяющий строку. Атрибут длины строки символов выражается действительным количеством символов между апострофами.

Модель организации данных строки
(s[0]=длина строки, 0<=s[0]<=255)

s[0]	**s[1]**	**s[2]**	**s[3]**	**s[4]**	**s[n]**

Описание переменной типа «строка» выглядит следующим образом:

```
var переменная: string;{длина строки <= 255 символов}
var переменная: string[кол_во_символов_в_строке];
```

В языке Си принят другой способ построения строк. Строка является массивом символов, а для указания её конца используется специальный символ с кодом ASCII, равным 0 (символ \0). Такие строки называют ASCIIZ строками. В Pascal для них предопределен специальный тип Pchar. (указатель на символ). Необходимость введения такого типа связана с широким применением ASCIIZ строк в ОС Windows.

Операции

Операции подразделяются на арифметические, логические, строковые, операции над множествами, операции отношения и операцию @ (операция получения адреса).

Операция присваивания

```
Имя_переменной := выражение;
```

Арифметические операции

Сложение	Вычитание	Умножение	Деление
`A + b`	`A - b`	`a * b`	`a / b`

Целочисленное деление	Остаток от деления	Двоичный сдвиг влево[4]	Двоичный сдвиг вправо[5]
`a div b`	`a mod b`	`a shl b`	`a shr b`
`20 div 3 ≡ 6`	`20 mod 3 ≡ 2`	`32 shl 2 ≡ 128`	`32 shr 2 ≡ 8`

Логические операции[6]

Операции булевой алгебры (высший приоритет)

1. Операция логического отрицания **not**:

Not true ≡ **false**; **not false** ≡ **true**;

[4] Часто используется для организации умножения на степень 2.

[5] Часто используется для организации деления на степень 2.

[6] Результат таких операций всегда булевского типа (true – false).

2. Бинарные логические операции **and** (логическое И), **or** (логическое ИЛИ) и **xor** (исключающее или) работают следующим образом[7]:

X	*Y*	*X* **and** *Y*	*X* **or** *Y*	*X* **xor** *Y*
false	false	false	false	false
false	true	false	true	true
true	false	false	true	true
true	true	true	true	false

Операции отношения (низший приоритет)

Равно	Не равно	Меньше	Меньше или равно	Больше	Больше или равно
a **=** b	a **<>** b	a **<** b	a **<=** b	a **>** b	a **>=** b

Составные операторы

Составные операторы задают порядок выполнения операторов, являющихся их элементами. Они должны выполняться в том порядке, в котором они записаны. Составные операторы обрабатываются как один оператор, что имеет решающее значение там, где синтаксис Паскаля допускает использование только одного оператора. Операторы заключаются в операторные скобки **begin** и **end** и отделяются друг от друга точкой с запятой. Приведем пример составного оператора:

```
begin
   Z := X;
   X := Y;
   Y := Z;
end;
```

[7] Кроме того, эти операции используются для выполнения побитовых операций с числами.

Стандартные процедуры и функции

Как и любой другой язык высокого уровня Pascal имеет библиотеки стандартных подпрограмм, позволяющие выполнять основные наиболее часто используемые операции. К ним относятся математические, тригонометрические, системные подпрограммы. Использование библиотек позволяет решить проблему эффективности (использование уже созданных, отлаженных и проверенных подпрограмм) и переносимости (для разных операционных систем и машин). В языке Pascal для удобства изучения и использования подпрограммы разделены на функции и процедуры. Принципиальной разницы между ними нет: вместо процедуры можно создать функцию с аналогичным функционалом и наоборот. Из всех подпрограмм процедуры read (readln) и write (writeln) используются настолько часто, что их обычно называют операторами.

Ввод данных оператором READ

Оператор **`read`** имеет следующий вид:

`read` (<переменная>, <переменная>, . . . , <переменная>);

Оператор **`read`** принимает информацию из входного потока, интерпретирует ее как значения и присваивает эти значения соответствующим переменным. Количество значений и порядок их следования во входном потоке должны находиться в строгом соответствии со списком переменных в операторе **`read`**. Если эти значения являются числами, то каждое из них набирается по правилам записи чисел.

Между значениями во входном потоке должен стоять один или несколько разделительных символов, которыми могут быть только пробел или символ новой строки. Если при считывании данных оператор **`read`** не обнаруживает в строке числа, он считывает следующую строку и т.д. Если же число набрано неправильно или в его записи встречаются недопустимые символы (например, запятая или точка с запятой), программа прекращает работу и выдает сообщение об ошибке (*invalid numeric format*). Отметим, что при вводе с клавиатуры информация попадает во входной поток программы только после того, как нажата клавиша

<**Enter**>. Если при наборе допущена ошибка и информация еще не отправлена во входной поток, можно воспользоваться клавишей <**Backspace**> для внесения исправлений. После нажатия <**Enter**> изменить значение уже нельзя. У оператора **read** существует модификация **readln**, отличающаяся тем, что после нажатия <**Enter**> происходит переход курсора на следующую строку.

Вывод данных оператором WRITE

Оператор **write** может быть представлен как

write (<переменная>, <переменная>, . . . , <переменная>);

Его исполнение заключается в последовательном выводе указанных элементов в выходной поток. В качестве элементов вывода в простейшем случае можно указывать следующие конструкции:

- <строка>
- <переменная>
- <переменная>: <формат>
- <переменная> : <формат> : <дробная часть>

Перечисленные ранее варианты элементов вывода в операторе **write** не исчерпывают всех возможностей. В частности, в любом из них вместо переменной может стоять арифметическое выражение, например:

write (Mass/MolMass*Na:15:5,' молекул');

Данная команда выведет на экран значение выражения с заданным форматом (цифры :15:5 после переменной означают – использовать для вывода 15 символов, из них после запятой до 5 знаков) и слово « молекул». Кроме оператора **write**, для вывода информации в выходной поток часто используют оператор **writeln**, отличающийся тем, что следующий оператор **write (writeln)** производит вывод уже в следующей строке.

Некоторые часто используемые функции

Sin(x)	Синус[8]	**abs**(x)	Модуль	**inc**(x)	Увеличить на 1
cos(x)	Косинус	**exp**(x)	Экспонента	**dec**(x)	Уменьшить на 1
tan(x)	Тангенс	**ln**(x)	Нат. логарифм		

Операции управления

Операторы управления позволяют изменить порядок выполнения программы. К ним относятся условные операторы и оператор безусловного перехода. Условные операторы позволяют выбрать для выполнения один из составных операторов (или не выбрать ни одного).

Оператор выбора

```
if логическое_выражение then оператор1
                          [else оператор2]
```

В логическом выражении должен получаться результат, имеющий стандартный булевский тип. Если результатом выражения является истинное значение true, то выполняется оператор, следующий за ключевым словом then. Если результатом выражения является значение False и присутствует ключевое слово else, то выполнятся оператор, следующий за ключевым словом else. Если ключевое слово else отсутствует, то никакой оператор не выполняется. При необходимости выбора в программе из большого количества вариантов более предпочтительно использование оператора варианта case … of … else … end.

Оператор варианта

Этот оператор представляет собой переключатель, в котором различным значениям некоторой переменной сопоставляются соответствующие операторы.

Ниже приведен пример использования операторов **`if`** и **`case`**.

[8] Аргумент тригонометрических функций указывается в радианах.

```
Program if_then_else_case;
Var x:integer;
begin
 Readln(x);
 If x<0 then writeln ('число меньше 0')
  else
  begin {начало ветки else оператора if}
   Case x of  {начало оператора case}
     0: writeln ('число равно 0');
     1: writeln ('число равно 1');
         …
         …
     10:writeln ('число равно 10');
     {ветка else оператора case}

         Else writeln ('число больше 10')
     end{case};
  End;{else}
End.
```

Оператор безусловного перехода GOTO

Синтаксис: **goto** метка;

Служит для немедленного перехода на необходимый оператор программы. Оператор перехода должен быть помечен меткой. В качестве метки может быть использована цифра (от 0 до 9999) или обычные идентификаторы. Метка от оператора отделяется двоеточием, например **1: Quit: Label1:**. Не допускается использование в качестве меток служебных слов. Все метки должны быть описаны в разделе описаний блока, в котором они используются. Метка, описанная в одном блоке, не видна в других. Не допускается переход на оператор, находящийся в другом блоке (например, переход из одной подпрограммы в другую или из основной программы в подпрограмму).

Циклы

Циклы служат для многократного выполнения в программе определенной последовательности действий. В Паскале для реализации циклов существуют три конструкции:

For … **to** … **to**	**while** … **do**	**repeat** … **until**
Оператор цикла с параметром	*Оператор цикла с предусловием*	*Оператор цикла с послеусловием*
Используется для выполнения определенной последовательности действий заданное число раз	Необходимость выполнения тела цикла проверяется до его начала	Необходимость выполнения итерации проверяется в конце тела цикла. Выполняется хотя бы 1 раз

Организовать выполнение цикла можно и без использования оператора цикла. Например: *рассчитать сумму чисел от 1 до 10.*

Вариант с использованием цикла ***for****:*

```
S:=0
for i:=1 to 10 do
begin
s:=s+i;
end;
```

Вариант с использованием меток:

```
S:=0; i:=1;
{инициализация переменных}
1: s:=s+i;
 i:=i+1;
if i<=10 then goto 1;
{Проверка условия выполнения
следующего прохода}
```

Запись **for** `i:=1` **to** `10` **do** означает, что в процессе выполнения цикла переменная `i` (она называется переменной цикла) будет изменяться от начального значения 1 до конечного значения 10, увеличиваясь на 1. Эту переменную можно использовать в самом цикле, например, для индексации элементов массива. Недостатком цикла **for** является то, что в качестве переменной цикла может быть использована только переменная целого типа, т.е. шаг цикла равен **1** (или **–1**: вариант **for** … **downto** … **do**).

Если шаг переменной цикла отличен от 1, то более целесообразно использование других операторов цикла. Однако в них переменную цикла нужно инициализировать до начала цикла и самостоятельно изменять ее в цикле.

Например: необходимо рассчитать сумму $S = \sum_{i=1}^{10} i$ с шагом 0,1.

While…do	***repeat…until***

`S:=0; i:=1;`
`While` `i<=10` **`do`**
` `**`Begin`**
`  s:=s + i;`
`  i:=i + 0,1;`
` `**`end;`
	`s:=0; i:=1;`
	`repeat`
	`  s:=s+i;`
	`  i:=i+0,1;`
	`until` `i>10;`

В цикле **`While`** условие (`i<=10`) контролируется в начале выполнения тела цикла (тело цикла будет выполняться до тех пор, пока переменная `i` не станет больше 10). Характер изменения переменной задает программист в самом цикле (`i:=i + 0,1;`).

Цикл **`repeat`** … **`until`** эквивалентен предыдущему, но в отличие от него выполняется до тех пор, пока условие (`i>10`) не станет истинным. В циклах **`for`** и **`While`** выполняется один оператор, следующий после **`do`**. Для выполнения в цикле нескольких операторов необходимо использовать составной оператор. Цикл **`repeat`** … **`until`** не требует заключения операторов в блок **`Begin`** … **`end`.**

В некоторых случаях возникает необходимость немедленного выхода из цикла. Для этой цели служит оператор **break**. Если необходимо пропустить итерацию, используется оператор **continue.**

Например: *вводить последовательно 10 чисел и считать их сумму до тех пор, пока не будет введено отрицательное число.*

```
for i:=1 to 10 do
 begin
  readln (a);
  if a >=0 then {число положительное}
  begin
   s:=s+ a; {прибавляем к сумме}
   continue; {пропускаем операторы до конца цикла}
  end;
  writeln('введено отрицательное число');
  break; {число отрицательное, выходим из цикла}
 end;
```

Комбинированные типы

Данные, обрабатываемые программой, в большинстве случаев представляют собой не единичные значения, а определенный набор. Например, данные эксперимента могут быть представлены как в виде ряда чисел, так и в виде матрицы и т.д. Если значений немного (5–10), то для работы с ними можно объявить соответствующее число переменных (A1, A2 … A10).

А если их 100, 1000? В этом случае для организации работы можно использовать *тип-массив*, представляющий собой группу однородных элементов, доступ к которым осуществляется по индексу, указывающему место элемента в массиве. Например, массив из 10 целых чисел объявляется следующим образом:

```
Masint: array [1 .. 10] of integer;
```

Индексация массива может начинаться с любого числа (целого), например, `[-5 .. 5]`, `[0 .. 20]`. Размер массива должен быть указан явно, так как отведение памяти под массив производится на этапе компиляции программы.

Массивы могут быть многомерными, причем размерность массива ограничена размером памяти, отводимой под переменную[9]. В памяти массивы располагаются следующим образом:

Модель организации данных одномерного массива: a[1 .. n]

a[1]	a[2]	a[3]	a[4]	a[5]	...	a[n]

Модель организации данных двумерного массива: a[1 .. m,1 .. n]

a[1,1]	a[1,2]	a[1,3]	a[1,4]	a[1,5]	...	a[1,n]
a[2,1]	a[2,2]	a[2,3]	a[2,4]	a[2,5]	...	a[2,n]
a[3,1]	a[3,2]	a[3,3]	a[3,4]	a[3,5]	...	a[3,n]
a[m,1]	a[m,2]	a[m,3]	a[m,4]	a[m,5]	...	a[m,n]

[9] Необходимо помнить, что с увеличением размерности объем отводимой памяти резко возрастает: A[1 .. 10] of integer – 20 byte; A[1 .. 10, 1 .. 10] of integer – 200 byte; A[1 .. 10, 1 .. 10, 1 .. 10] of integer – 2000 byte.

Работа с массивом сводится к работе над его элементами. Для массива в целом доступна лишь операция присваивания, причем оба массива должны быть одного типа и размерности.

```
program massive;
 const mas_size=100;
 var A, B:array [1 .. mas_size] of integer;
  i:integer;
begin
 for i:=1 to mas_size do
 begin
 writeln ('введите ',i,'элемент массива');
 readln (A[i]); {считывание элемента массива}
 end;
 B:=A; {присваиваем одному массиву значение другого}
 for i:=1 to mas_size do
 writeln ('B[',i,']=',B[i]);{поэлементный ввод масси-
ва}
end.
```

В некоторых случаях возникает необходимость в группировке разнородных данных. Например, такими данными могут быть данные о каком-либо объекте. Данные такого типа объединяются в ***типы-записи*** (**record**). Приведем пример объявления типа-записи:

```
type
 Person:record
   Firstname, lastname: string[100];
   age: integer;
   weight, height: real; end;

var oneman: person;
     {объявление переменной типа person}
Peoples: array[1 .. 1000] of person;
    {записи могут быть элементами других
        структурированных типов}
```

Элементы записи называются полями. Работа с записями заключается в работе с её полями.

```
{********** пример работы с записями ************}
for i:=1 to 1000 do
begin
 writeln ('person number ',i);
 write('Введите Ваше имя '):
 readln (Peoples[i].firstname);
                              {Заполняем поле firstname}
 write ('Введите Вашу фамилию ');
 readln (Peoples[i].lastname);
                               {Заполняем поле lastname}
 write ('Введите Ваш возраст ');
 readln (Peoples[i].age);
 write ('Введите Ваш вес ');
 readln (Peoples[i]. weight);
 write ('Введите Ваш рост ');
readln (Peoples[i].height);
end;
```

Для работы с записями часто используется оператор **with** (вместе с…). В этом случае нет необходимости полностью указывать имя записи, например:

```
For i:=1 to 1000 do
with Peoples[i] do
begin
 writeln('person number ',i);
 writeln ('Имя: ',firstname);
                               {Выводим поле firstname}
 writeln ('Фамилия: ',lastname)
                               { Выводим поле lastname}
 writeln ('Возраст: ',age);
 writeln ('Вес: ', weight);
 writeln ('Рост: ', height);
end;
```

Тип-запись может иметь вариантную часть, изменяющуюся при необходимости, например:

```
Type
 Figure = (Square, Triangle, Circle);
 Param = record
  X,Y: real;
  Case fig: figure of {начало вариантной части }
```

```
    Square: (Side: real);
    Triangle:(side1, side2, angle: real);
    Circle: (radius:real);
  End;
var Mysquare, Mycircle: Param;
```

В этом случае объявленные переменные могут иметь различные поля в зависимости от значения поля выбора. Это позволяет динамически изменять набор полей записи, например:

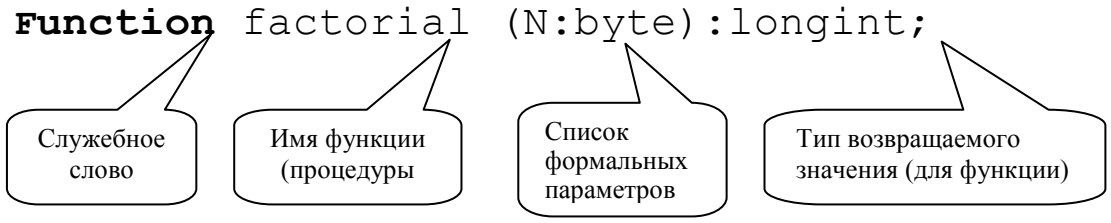
```
Fig:=Square;
MySquare.Side:=10;

Fig:=Circle;
MySquare.Radius:=5;
```

Процедуры и функции

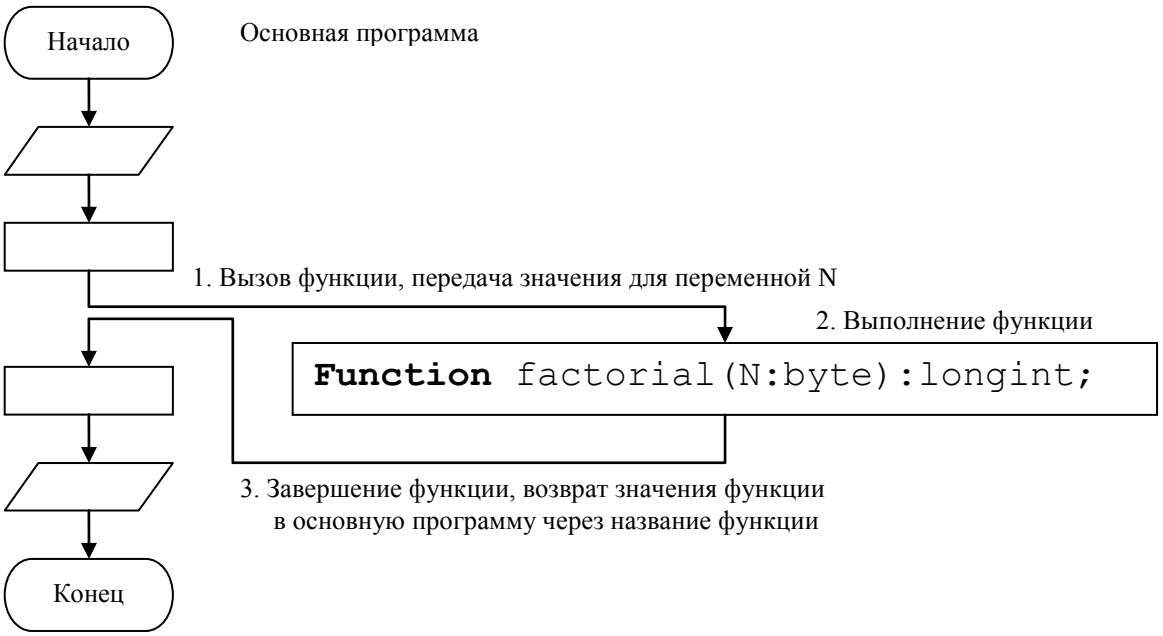
Понятие подпрограммы существует практически в каждом языке программирования. В Pascal существует две разновидности подпрограмм: функции и процедуры. Структура подпрограммы повторяет структуру программы. Она должна быть описана до того, как она будет использована в программе или другой подпрограмме в разделе описания процедур и функций. Описание выглядит следующим образом (подпрограмма вычисления факториала):

ФУНКЦИЯ	ПРОЦЕДУРА

Function factorial
(N:byte):longint;
 	
var Fact:longint;i:byte;
 	
begin
Fact:=N;
For i:=N-1 **downto** 2 **do**
Fact:=Fact*i;
Factorial:=Fact;
End;
	Procedure Factorial
	(N:byte;**var** Fact:longint);
	
	var i:byte;
	
	begin
	Fact:=N;
	for i:=N-1 **downto** 2 **do**
	Fact:=Fact*i;
	End;



Использование процедур и функций позволяет значительно упростить программу, структурировать её, сделать более наглядной, позволяет исключить повторение участков кода в нескольких местах. Кроме того, вынос стандартных структур и функций в отдельные модули делает язык не зависящим от конкретной платформы (операционной системы).

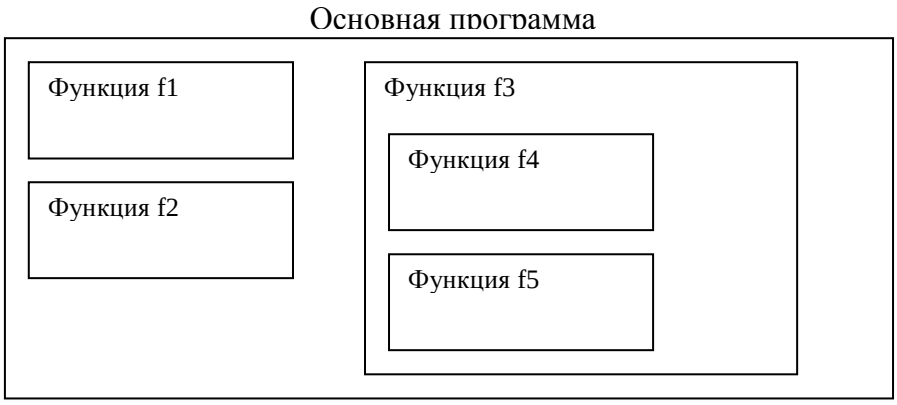


Взаимодействие процедур и функций с остальными частями программы осуществляется с помощью механизма формальных параметров, задаваемых в описании процедуры или функции. Каждый параметр, описанный в списке формальных параметров, является локальной переменной (константой) в описываемой процедуре или функции. Формальные параметры могут быть трех типов:

- параметры значения,
- параметры константы,
- параметры-переменные.

При вызове процедуры либо функции указываются действительные значения (числа, строки и т.д.), либо переменные, значения которых нужны для ее работы. Они называются актуальными параметрами.

Переменные и константы, определенные в основной программе, называются глобальными по отношению ко вложенным подпрограммам и видны для них. Переменные и константы, определенные во вложенных подпрограммах, называются локальными и видны только в данных подпрограммах и во вложенных в них подпрограммах. Например, переменные, объявленные в функции f3, будут видны только в функциях f3, f4, и f5.



При передаче параметров-значений и параметров-констант значения передаваемых переменных копируются, и в подпрограмму передается копия. В случае параметров-констант дополнительно накладывается ограничение на изменение в подпрограмме их значений. Так как в подпрограмме используются копии, то их изменение никак не отражается на значениях этих переменных в основной программе. В случае параметров-переменных в программу передается не копия, а адрес самой переменной. Результатом является то, что все манипуляции в подпрограмме производятся с самой переменной, а не с копией. Поэтому **VAR** параметры используются для передачи результатов обратно в вызывающую программу (подпрограмму)

```
procedure procname (i,j:integer;{параметры-значения }
    const n:integer;{параметры-константы }
    var vesult:real;{параметры-переменные}
```

В отличие от процедуры функция имеет тип возвращаемого значения. Это позволяет использовать функции непосредственно в выражениях для вычисления какого-либо значения.

```
Program Use_function;
 var j: byte;
Function Factorial(N:byte):longint;
 var Fact:longint; i:byte;
begin
 Fact:=N;
 for i:=N-1 downto 2 do Fact:=fact*i;
 Factorial:=Fact;
End;
{начало основной программы}
begin
 writeln ('введите целое число');
 readln (j);
 writeln ('факториал числа',j,'равен',
Factorial(j));
end.
```

В случае процедуры вычисляемое значение передается или через глобальные параметры программы, либо посредством **var** параметров.

Передача результата через глобальные переменные.

```
Program Use_global_var;
Var i:integer;
 result:longint;
{описание процедуры}
procedure Fact1 (N:byte);
var j:byte;
begin
 {глобальная переменная}
 Result:=N;
 For j:=N-1 downdo 2 do
 Result:=result*j;
end;
{основная программа }
begin
writeln('введите
   целое число');
```

*Передача результата через **var**-параметры.*

```
Program Use_var_parametres;
 var j:integer;
  result:longint;
{описание процедуры}
procedure Fact2 (N:byte;
  var Fact:longint);
 var i:byte;
 begin
 Fact :=N;
 for i:=N-1 downto 2 do
 Fact:=Fact*i;
 end;
{основная программа }
begin
writeln('введите
   целое число');
```

```
readln(i);
{вызов процедуры}
Fact1(i);
Writeln('Факториал числа',
  i,'равен ', result);
end.
```

```
readln (j);
{вызов процедуры}
  Fact2(j,result);
Writeln('Факториал числа `,
  j,'равен', result);
end.
```

Модули

При наличии в программе большого числа подпрограмм их удобно размещать в отдельных файлах, называемых модулями.

Модуль представляет собой набор процедур и функций. В основной программе модуль подчиняется с помощью служебного слова **uses** в разделе описания модулей. Этим служебным словом начинается секция используемых программных модулей. В ней указывается одно или несколько перечисленных через запятую названий модулей. В Pascal имеется ряд модулей, служащих для выполнения определенных задач:

– **crt**– модуль для работы в текстовом режиме;

– **graph** – модуль, содержащий графические процедуры и функции;

– **strings** – модуль для работы со строками;

– **dos** – модуль для работы с OS;

– **system** – основной модуль, содержащий основные процедуры и функции, такие как **read**, **write** и т.д. Единственный модуль, не требующий указания в разделе **uses**.

Структура модуля напоминает структуру программы. Модуль начинается со служебного слова **unit**. Имя модуля должно совпадать с именем файла, так как компьютер производит поиск модуля по имени файла.

Далее модуль разбивается на две основные части: интерфейсную (начинается со служебного слова **interface**) и часть реализации (**implementation**). Первая часть служит для связи модуля с другими модулями и вызываемой программой. Вызывающими программами должны использоваться только переменные, типы, процедуры и функции, описанные в разделе **interface**. Каждая описанная функция должна быть реализо-

вана в разделе **implementation**. В конце модуля также ставится **end.**

Приведем пример модуля:

```
Unit MyUnit;
interface {начало интерфейсной части}
uses crt;
type MyType =
   array [1 .. 100] of record a, b: integer end;
var A1: mytype;{объявление глобальных переменных }
  B1: real; {которые могут быть использованы }
     {в вызывающей программе   }
function f1( i, j :integer):real;
procedure proc10 (var: s:string);
implementation { интерфейсная часть закончилась, }
     { начинается реализация описанных}
     { процедур и функций    }
function f1 (i, j: integer):real;
 begin
 … {код реализации }
 end;
procedure proc10;{в модуле реализации указывать  }
     { список формальных параметров, }
     { а также результат функции или }
     { процедуры необязательно   }
begin
 … {код реализации }
 end;
end. {конец модуля}
```

Использование модулей позволяет значительно структурировать программу, облегчить ее читаемость и процесс отладки, а также дает возможность повторного использования кода (в других программах).

Для использования подпрограмм из модуля, его необходимо объявить в программе в разделе описания модулей.

```
Program UseModule;
Uses Crt, Graph, MyModule;
     …
```

Файлы. Операции над файлами

В языке Pascal под ФАЙЛОМ понимается область памяти на внешнем запоминающем устройстве, способная хранить некоторую совокупность информации. В эту область внешней памяти можно как помещать определенные данные, так и извлекать их из нее. Эти действия имеют общее название «ввод – вывод».

Для организации работы по вводу – выводу в программе могут быть определены специальные переменные файловых типов, которые считаются *представителями* файлов в Pascal-программе. Файловая переменная в Паскале – это любая переменная файлового типа. В Паскале имеются три класса файлов: типизованный файл, текстовый файл и нетипизованный файл. Перед использованием файловой переменной она должна быть связана с внешним файлом с помощью вызова процедуры **Assign**. Внешний файл также может представлять собой устройство, например, клавиатуру или дисплей. Когда связь с внешним файлом установлена, для подготовки ее к операции ввода или вывода файловая переменная должна быть «открыта». Существующий файл можно открыть с помощью процедуры **Reset**, а новый файл можно создать и открыть с помощью процедуры **Rewrite**. Текстовые файлы, открытые с помощью процедуры **Reset** доступны только по чтению, а текстовые файлы, открытые с помощью процедуры **Rewrite** – только по записи. Типизованные и нетипизованные файлы всегда допускают как чтение, так и запись, независимо от того, были они открыты с помощью процедуры **Reset** или с помощью процедуры **Rewrite**.

Когда начинается выполнение программы, всегда автоматически открываются стандартные текстовые файловые переменные *Input* и *Output* (ввод и вывод). *Input* – это доступный только по чтению файл, связанный с клавиатурой, а *Output* – это доступный только по записи файл, связанный с дисплеем. Любой файл представляет собой линейную последовательность элементов, каждый из которых имеет тип элемента (или тип записи) файла. Каждый элемент файла имеет номер. Первый элемент файла считается нулевым. Обычно доступ к файлам организуется последовательно, т. е. когда элемент считывается с помощью стандарт-

ной процедуры **Read** или записывается с помощью стандартной процедуры **Write**, текущая позиция файла перемещается к следующему по порядку элементу файла. Однако к типизованным и нетипизованным файлам можно организовать прямой доступ с помощью стандартной процедуры **Seek**, которая перемещает текущую позицию файла к заданному элементу. Для определения текущей позиции в файле и текущего размера файла можно использовать стандартные функции **FilePos** и **Filesize.**

Когда программа завершает обработку файла, он должен закрываться с помощью стандартной процедуры **Close**. После полного закрытия файла связанный с ним внешний файл обновляется. Затем файловая переменная может быть связана с другим внешним файлом. По умолчанию при всех обращениях к стандартным функциям и процедурам ввода – вывода автоматически производится проверка на наличие ошибок. При обнаружении ошибки программа прекращает работу и выводит на экран сообщение об ошибке. С помощью директив компилятора {$I+} и {$I–} эту автоматическую проверку можно включить или выключить. Когда автоматическая проверка отключена, т. е. когда процедура или функция была скомпилирована с директивой {$I–}, ошибки ввода – вывода, возникающие при работе программы, не приводят к ее останову.

При этом, чтобы проверить результат выполнения операции ввода – вывода, нужно использовать стандартную функцию **IOResult**. Описание переменной вида

```
Var F : file of integer;
```

понимается как определение в программе под именем F списка неопределенного количества целых чисел, расположенного на некотором внешнем запоминающем устройстве (например, на магнитном диске). Как правило, все действия с файлом (чтение из файла, запись в файл) производятся поэлементно, причем в этих действиях участвует тот элемент файла, который обозначается текущим указателем. В результате совершения операций текущий указатель может перемещаться, настраиваясь на тот или иной элемент файла. Все элементы файла считаются пронумерованными; начальный элемент имеет НУЛЕВОЙ номер.

Итак, файловый тип задается в программе с помощью служебных слов **file** и **of**, за которыми следует тип элементов файла (базовый тип). Базовый тип может быть любым типом, кроме файлового; в качестве базового типа не допускается комбинированный тип, одним из полей которого является файл. Turbo Pascal допускает определение так называемых бестиповых файлов, для которых тип компонентов не устанавливается.

Примеры описаний файловых типов и переменных:

```
type
Sequence = file of char;
var
F1, F2 : Sequence;
Table : file of string[80];
DataBase : file of Person;
InputData : file of real;
```

Пример. Из текстового файла T прочитать числа и, считая в каждой паре первое число действительной, а второе — мнимой составляющей, записать их в двоичный файл комплексных чисел C.

```
program Work_with_files;
type Complex = record Re, Im : real; end;
Var F2: file of Complex; (объявляем двоичный файл}
  F1: text: {объявляем файловую переменную типа text}
  Comp: Complex ;
  I:integer; a, b: real;
  C : char;

Procedure Errors (ICode: integer);
{сообщения об ошибках}.
begin
case ICode of
1 :writeln ('Файл не найден');
2:writeln ('Путь не найден');
3: writeln ('Ошибка открытия файла');
end;
if icode <>0 then halt; {если ошибка, то
                                  выходим из программы}.
end.

Begin
   {$I-} {отключаем автоматическую проверку
```

```
                                  правильности ввода-вывода}
assign (F1,'T'); {связываем файловую переменную F1
                                       с файлом с именем 'T'}
  reset (F1);                  {открываем файл для чтения}
  assign (F2,'C');
  rewrite (F2);    {открываем двоичный файл для записи}
    IOCode:= IOResult;
    if IOCode<>0 then Errors (IOCode);
    i: = 0;
  while not eof(F1) do
                          {пока не кончится текстовый файл}
  begin
  inc(i);
  readln (F1,a);
  if not eof(F1) then readln (F1 ,b)else b:=0;
       {если файл не закончился после считывания первого
                          числа, то считываем второе (Im)}.
  Comp.Re:=a; Comp.Im:=b;
      write(F2,Comp);{Записываем считанную пару чисел
                                          в двоичный файл}
  end;
  {текстовый файл закончился,
                               закрываем его и двоичный файл}
close (F2); {$I+}                      {включаем проверку}
writeln ('Данные перенесены, кол-во точек :',
                                    I, 'просмотреть (Y,N)');
readln (C);
if C<>'Y' then exit
else
begin
  {просмотр двоичного файла}
  assign (F2,'C');
  reset (F2);i:=0;
  while not Eof(F2) do
   begin
   read(F2,comp); inc (i);
  writeln ('точка:',i,' Re:', comp.Re,' Im:', comp.Im)
  end;
  writeln ('Всего точек: ',i);
  close (F2);
  readln;
end.
```

Указатели

Каждая переменная занимает определенное место в оперативной памяти компьютера, называемое адресом. Существует два способа доступа к переменным в программе: по имени (обычный способ) и по адресу. Для этого используется специальный вид переменных, называемых указателями. Указатель – это переменная, хранящая адрес какой-либо другой переменной, структуры, подпрограммы. Указатели могут быть как определенного типа (например, указатель на целое вещественное), так и бестиповые, которые могут указывать на переменную любого типа. В программе указатели описываются аналогично обычным переменным, например:

```
a:integer;  pj^ : real;
P:pointer; {бестиповой указатель}.
```

Для работы с указателями в Pascal предусмотрено 2 оператора:

^ – разыменование указателя;

@ – взятие адреса переменной.

Например:

```
pi:= @a; {записываем в pi адрес переменной a}
b:= pj^; {присваиваем переменной b значение перемен-
ной, на которую указывает pj}
```

Над указателями можно проводить обычные математические операции, например:

```
p:= pj; {присвоить p адрес переменной,
                                  на которую указывает pj}
pi:= pi + 2; {увеличить значение адреса,
                           записанного в pi, на 2 байта}
```

Но:

```
pi ^ := pi ^ + 2; {увеличить значение переменной,
                         на которую указывает pi, на 2}
```

Указатели часто используются при работе с различными структурами данных, а также при работе с динамическими переменными.

ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Корни линейных и квадратных уравнений можно найти по известным формулам. Хотя алгебраические уравнения третьей и четвертой степени можно решить аналитическими методами, соответствующие формулы достаточно сложны. Например, нельзя аналитически (точно) найти корни следующих уравнений:

$x^6 + 4x^5 - 5x^4 + x^3 + 3x^2 - 9x + 11 = 0$ (алгебраическое уравнение);

$x^2 - \sin x = 0$ (трансцендентное уравнение),

но с помощью численных методов можно найти приближенные корни[10] уравнений.

Метод деления отрезка пополам (метод полуинтервалов, метод дихотомии)

Пусть нам удалось найти отрезок [a, b], в котором расположено искомое значение корня x = c, т.е. a < c < b. В качестве начального приближения корня c_o примем середину этого отрезка: $c_o = (a + b) / 2$. Далее исследуем значения функции f(x) на концах отрезков $[a, c_o]$ и $[c_o, b]$ в точках a, c_o, b. Тот из них, на концах которого f(x) принимает значения разных знаков, содержит корень, поэтому его принимаем в качестве нового отрезка. В качестве следующего приближенного значения корня принимаем середину нового отрезка и т.д. Каждый новый шаг называется итерацией, а сама методика последовательного приближения к корню − итерационным методом. После каждой итерации отрезок, на котором расположен корень, уменьшается вдвое, т. е. после n итераций он сокращается в 2^n раз.

Пусть f(a) < 0, f(b) > 0. Пусть начальное приближение $c_o = (a + b) / 2$, так как $f(c_o) < 0$, то $c_o < c < b$ и рассматриваем отрезок $[c_o, b]$. Следующее приближение $c_1 = (c_o + b) / 2$, так как $f(c_1) > 0$, то $c_o < c < c_1$ и рассматриваем отрезок $[c_o, c_1]$. Следующее приближение $c_2 = (c_o + c_1) / 2$ и т.д. (рис. 1).

[10] Корнем уравнения f(x) называется такое значение аргумента, при котором f(x) обращается в нуль.

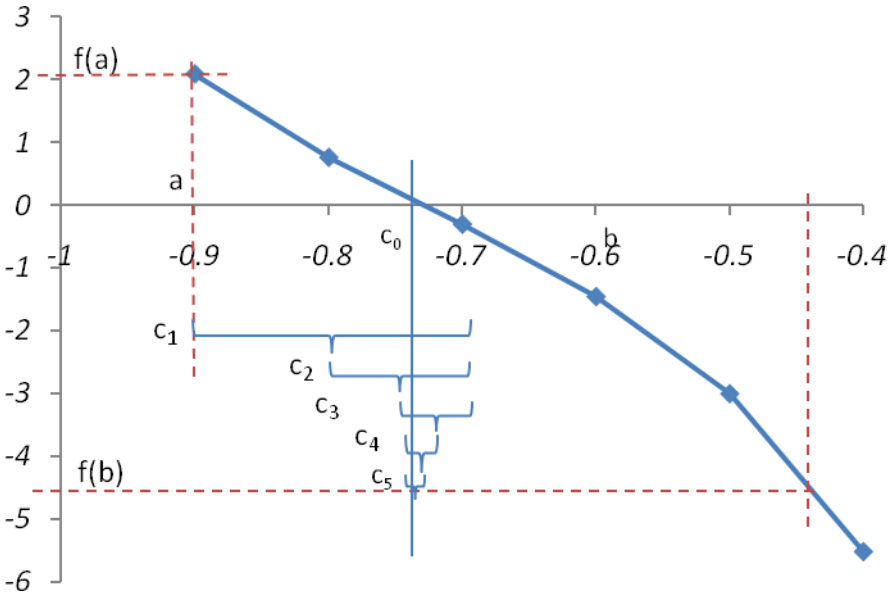


Рис. 1

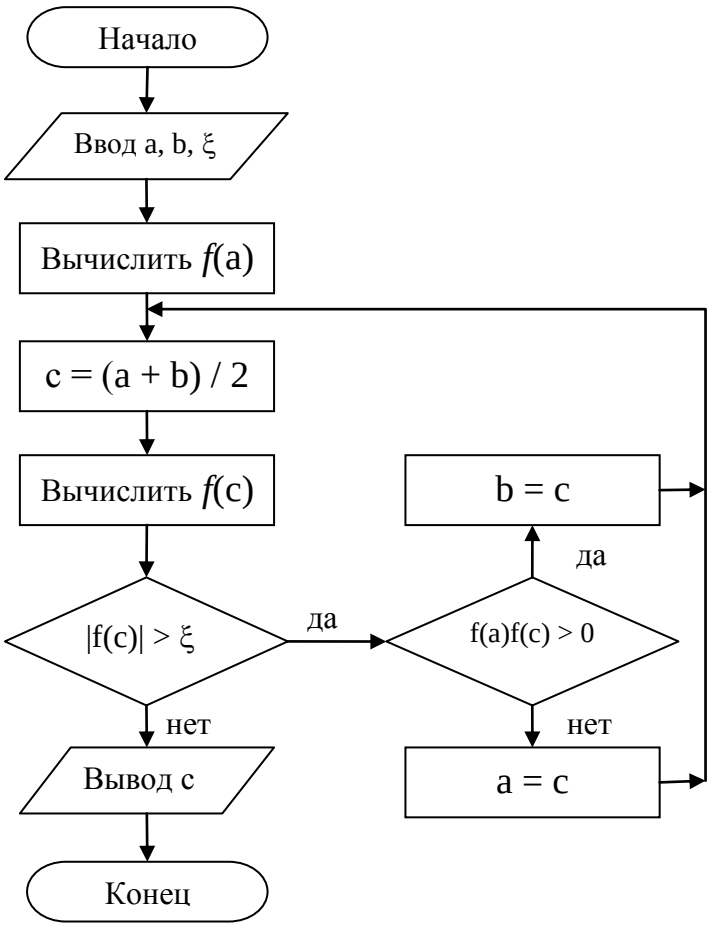


Рис. 2

Итерационный процесс продолжается до тех пор, пока значение функции f(x) после n-й итерации не станет меньшим по

модулю некоторого заданного малого ξ, т.е. $|f(c_n)| < \xi$. Можно также потребовать, чтобы расчет прекратился, когда точность вычисления корня составит 1/1000000 от исходного отрезка, или задать число итераций. Блок-схема метода показана на рис. 2.

Метод деления отрезка пополам довольно медленный, однако этот итерационный процесс всегда сходится, т.е. при его использовании решение получается всегда, так как при каждой последующей итерации происходит приближение к корню.

Пример

Найти с точностью $\xi = 0{,}01$ корень уравнения $x^2 \cdot \log_{0,5}(x + 1) = 1$.

Решение

Сначала определим отрезок существования корня — отделим корень графически: $\log_{0,5}(x + 1) = \frac{1}{x^2}$ (рис. 3).

Видно, что корень — один. Определим знак функции слева и справа от корня:

$$f(x) = x^2 + \log_{0,5}(x+1) - 1 = x^2 \frac{\lg(x+1)}{\lg 0{,}5} - 1 = x^2 \frac{\lg(x+1)}{-0{,}31} - 1$$

x	– 0,8	– 0,5
f(x)	+	–

Далее уточним корень с точностью $\xi = 10^{-2}$:

n	a_n	b_n	$x_n = (a_n + b_n)/2$	$f(a_n)$	$f(b_n)$	$f(x_n)$
0	-0,8	-0,5	-0,65	0,486182	-0,74998	-0,36
1	-0,8	-0,65	-0,725	0,486182	-0,36003	-0,0209
2	-0,8	-0,725	-0,7625	0,486182	-0,02093	0,20596
3	-0,7625	-0,725	-0,74375	0,205957	-0,02093	0,08673
4	-0,74375	-0,725	-0,734375	0,086731	-0,02093	0,03155
5	-0,734375	-0,725	-0,7296875	0,031547	-0,02093	0,00498
6	-0,7296875	-0,725	-0,72734375	0,004981	-0,02093	-0,0081

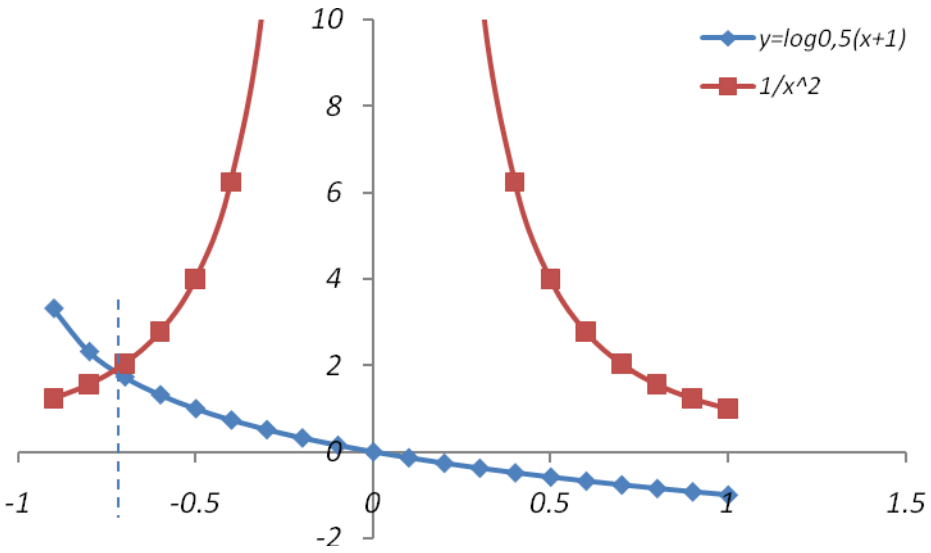


Рис. 3

Корень ≈ – 0,73.

Метод хорд (метод ложного положения, метод линейного интерполирования, метод пропорциональных частей)

В этом методе нелинейная функция f(x) на определенном интервале [a, b] заменяется линейной, в качестве которой берется хорда – прямая, стягивающая концы нелинейной функции. Хорда определяется как прямая, проходящая через точки с координатами (a, f(a)) и (b, f(b)). Имея уравнение хорды y = cx + d, можно найти точку ее пересечения с горизонтальной осью. Полученную точку x_1 считают новой границей отрезка, где содержится корень, и т.д. Получают последовательность приближений x_2, x_3,… сходящуюся к корню (рис. 4). Метод применим только для монотонных функций.

Алгоритм метода зависит от свойств функции f(x). Если $f(b) \cdot f''(b) > 0$, то строящаяся на каждом этапе хорда имеет правый фиксированный («закрепленный») конец и алгоритм имеет вид

$$x_{i+1} = x_i - \frac{f(x_i)}{f(b) - f(x_i)}(b - x_i).$$

При этом последовательность x_1, x_2,… приближается к корню слева.

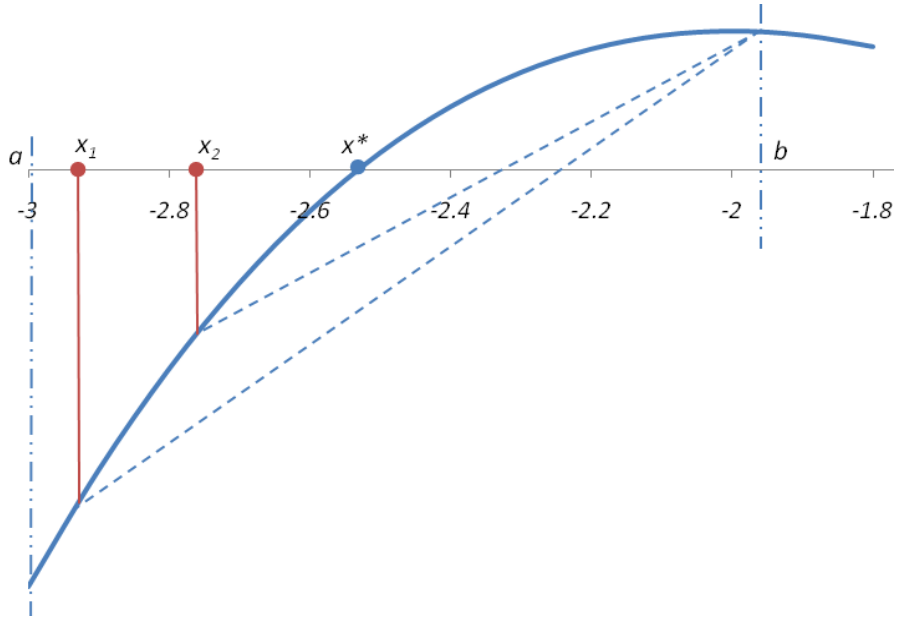

a
x1
x2
x*
b
-3
-2.8
-2.6
-2.4
-2.2
-2
-1.8

Рис. 4

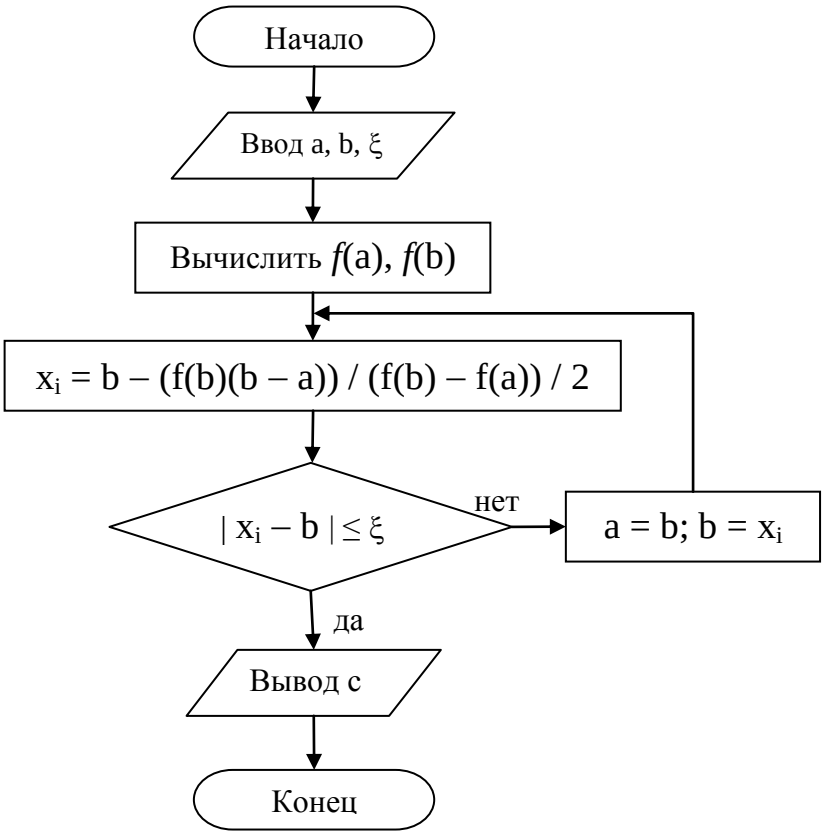

Начало
Ввод a, b, ξ
Вычислить f(a), f(b)
xi = b – (f(b)(b – a)) / (f(b) – f(a)) / 2
| xi – b | ≤ ξ
нет
a = b; b = xi
да
Вывод c
Конец

Рис. 5.

Если f(a) f″(a) > 0, то строящаяся на каждом этапе хорда имеет левый фиксированный («закрепленный») конец, и алгоритм имеет вид

$$x_{i+1} = a + \frac{f(a)}{f(a) - f(x_i)}(x_i - a).$$

При этом последовательность x_1, x_2,… будет приближаться к корню справа. Теоретически доказано, что если первые производные на концах интервала при монотонной и выпуклой функции f(x) не различаются более чем в 2 раза, то справедливо соотношение $|x^* - x_i| < |x_i - x_{i-1}|$ и условием прекращения пополнения последовательности может быть $|x_{i+1} - x_i| \le \xi$, а в качестве корня принято x_{i-1}. Можно также окончить процесс и при достижении $f(x_i) \le \xi$. На практике можно указанные условия применять и без предварительной проверки производных.

Пример

Найти с точностью $\xi = 10^{-3}$ меньший корень уравнения $x^3 + 3x^2 - 3 = 0$.

Решение

Определим отрезок существования меньшего корня – отделим корень. Отделим корни аналитически: приравняем $f'(x) = 0$. $f(x) = x^3 + 3x^2 - 3 = 0$; $f'(x) = 3x^2 + 6x = 0$; $x_1 = 0$, $x_2 = -2$.

Составим таблицу знаков функции f(x), полагая x равным:

– критическим значениям (корни производной) или ближайшим к ним;

– граничным значениям (исходя из области допустимых значений аргумента).

x	$-\infty$	-2	-1	0	1	$+\infty$
f(x)	–	+	–	–	+	+

Левый корень лежит в интервале $(-\infty; -2)$, для уточнения интервала найдем $f(-3) = -3$, следовательно,

x	–3	–2	–1	0	1
f(x)	–	+	–	–	+

Таким образом, корни содержатся в интервалах: (–3, –2); (–2, –1); (0, 1). Уточним корень методом хорд в интервале (–3, –2), т.е. a = –3, b = –2.

Найдем вторую производную f "(x) = 6x + 6, f "(x) < 0 на интервале (–3; –2), поэтому используем формулу

$x_{i+1} = a + \frac{f(a)}{f(a) - f(x_i)}(x_i - a)$, где x_o= b = – 2, f(a) = f(–3) = –3.

i	$\mathbf{x_i}$	$\mathbf{f(x_i)}$	$\mathbf{x_i - a}$
0	–2	1	1
1	–2,25	0,796875	0,75
2	–2,4074074	0,434435808	0,592593
3	–2,4823669	0,189730569	0,517633
4	–2,5131566	0,074881701	0,486843
5	–2,5250125	0,028372102	0,474987
6	–2,5294626	0,010583631	0,470537
7	–2,5311176	0,003925032	0,468883
8	–2,5317294	0,001452481	0,468271
9	–2,531956		

Корень ≈ –2,532.

Метод Ньютона (метод касательных)

В этом методе нелинейная функция f(x) на отделенном интервале [a, b] заменяется линейной, в качестве которой берется касательная, проводимая в текущей точке последовательности. Уравнение касательной находится по координате одной точки и углу наклона (значение производной). В качестве начальной точки в зависимости от свойств функции берется или левая точка $x_o = a$, если $f(a)\, f''(a) > 0$, или правая точка $x_o = b$, если $f(b)\, f''(b) > 0$. Алгоритм записывается следующим образом:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Получаем последовательность приближенных значений x_1, x_2, … , каждый член которой ближе к истинному корню x^*, чем предыдущий (рис. 6). Если производная $f'(x)$ мало изменяется на отрезке [a, b], то используют упрощенный вариант

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

(геометрически это означает, что касательные в точках $(x_i, f(x_i))$ заменяются прямыми, параллельными касательной, проведенной к кривой $y = f(x)$ в точке $(x_o, f(x_o))$).

Условия окончания поиска аналогичны методу хорд. Главным достоинством метода является то, что на каждой итерации погрешность возводится в квадрат, т.е. число верных знаков корня удваивается. Если начальное приближение выбрано удачно, то после 5 – 6 итераций величина погрешности будет $\sim 2^{-64}$. Для получения такой малой погрешности в методе дихотомии необходимо ~50 итераций. Блок-схема метода приведена на рис. 8.

Пример

Методом Ньютона уточнить до $\xi = 0{,}0001$ корень уравнения $x - \sin(x) = 0{,}25$.

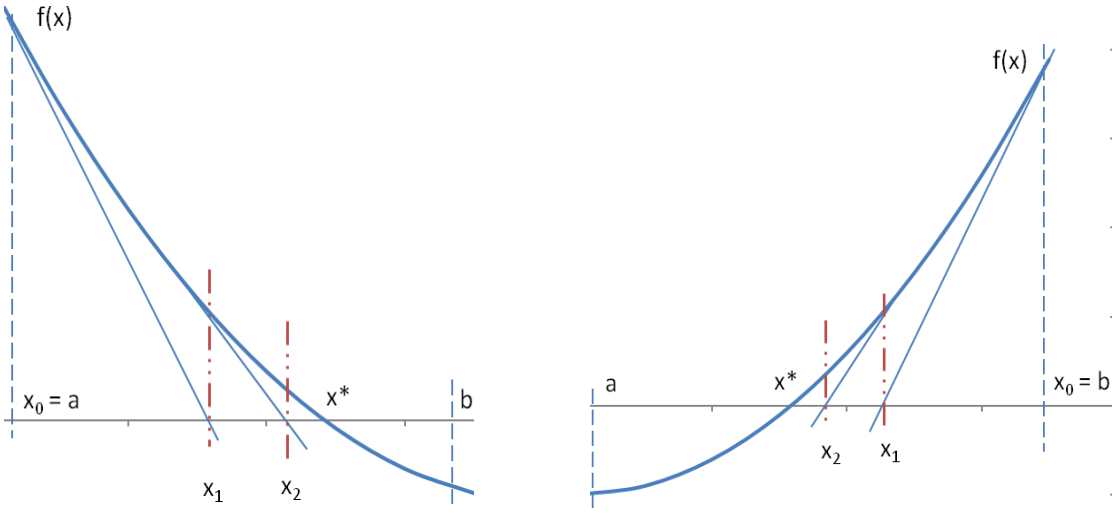


Рис. 6

Решение

Отделим корень графически.

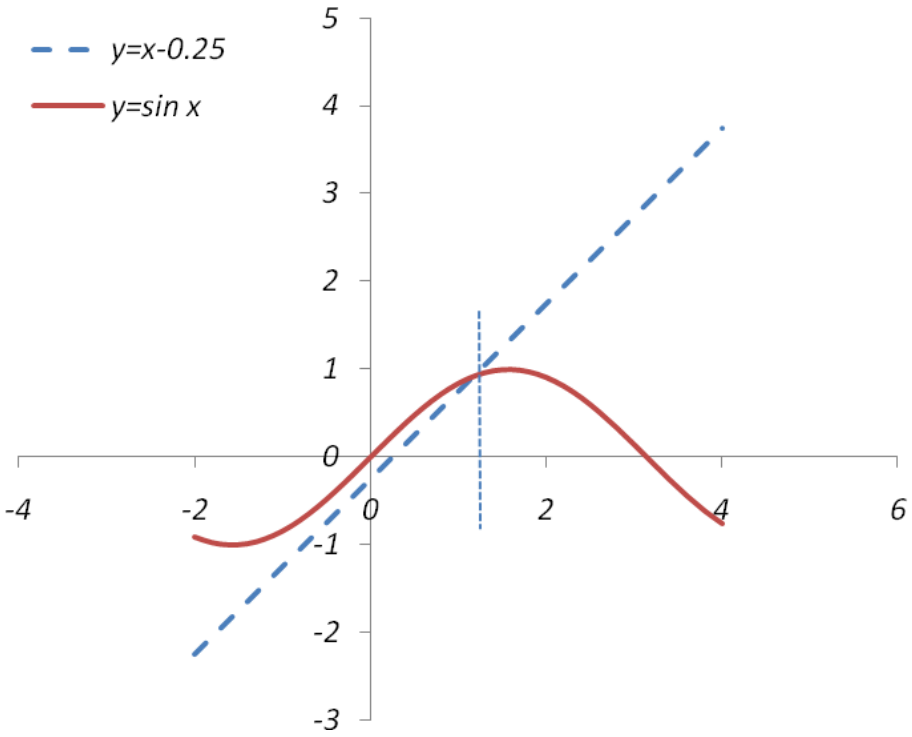


Рис. 7

Корень находится в интервале [0,982; 1,178]: a = 0,982, b = 1,178.

f ′(x) = 1 – cos(x); f″(x) = sin(x) > 0 на [0,982; 1,178], f(1,178)f″(x) > 0, поэтому x_o = 1,178.

i	**x_i**	**$f(x_i) = x_i - \sin(x_i) - 0{,}25$**	**$f'(x_i) = 1 - \cos(x_i)$**
0	1,178	0,00416	0,61723
1	1,1715	0,00017	0,61123
2	1,1713	0,00003	0,61110
3	1,17125		

Корень ≈1,1712.

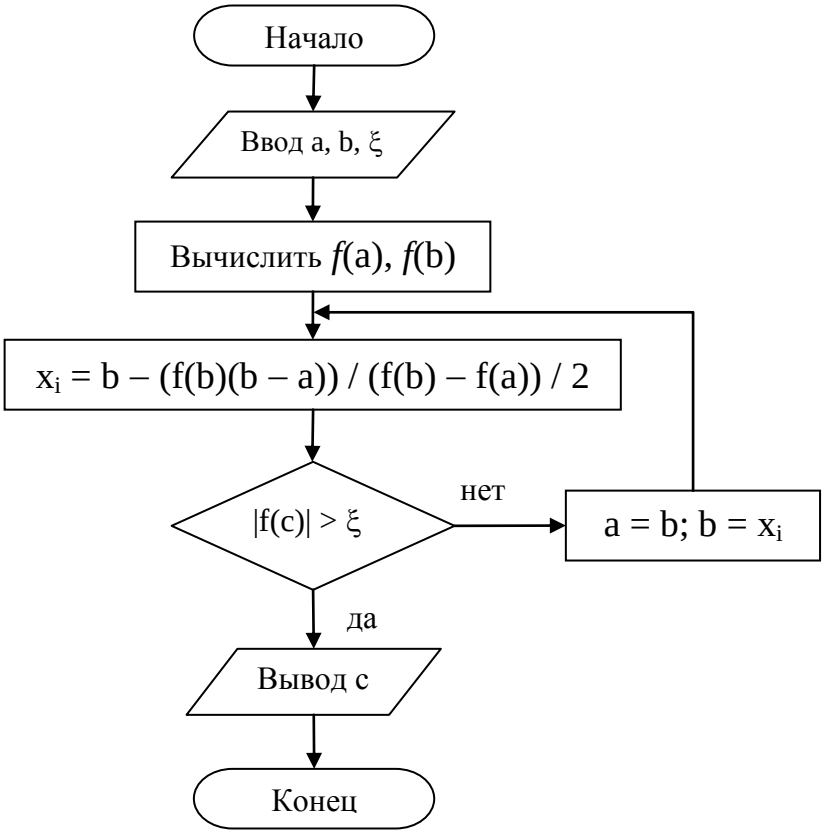


Рис. 8

Комбинированный метод

Метод также базируется на замене нелинейной функции f(x) линейной, но с учетом стремления к корню метода хорд и метода Ньютона с разных сторон. Для повышения эффективности используют оба алгоритма одновременно. Один шаг делается методом хорд, а следующий, с другой стороны – методом Ньютона.

Интервал, где содержится корень, сокращается с обеих сторон. Поиск можно прекратить, как только разница станет меньше предварительно заданной погрешности ξ.

Пример

Уточнить корни уравнения $x^3 - 2x^2 - 4x + 7 = 0$ с погрешностью $\xi < 0{,}001$.

Решение

Отделим корни графически (рис. 9).

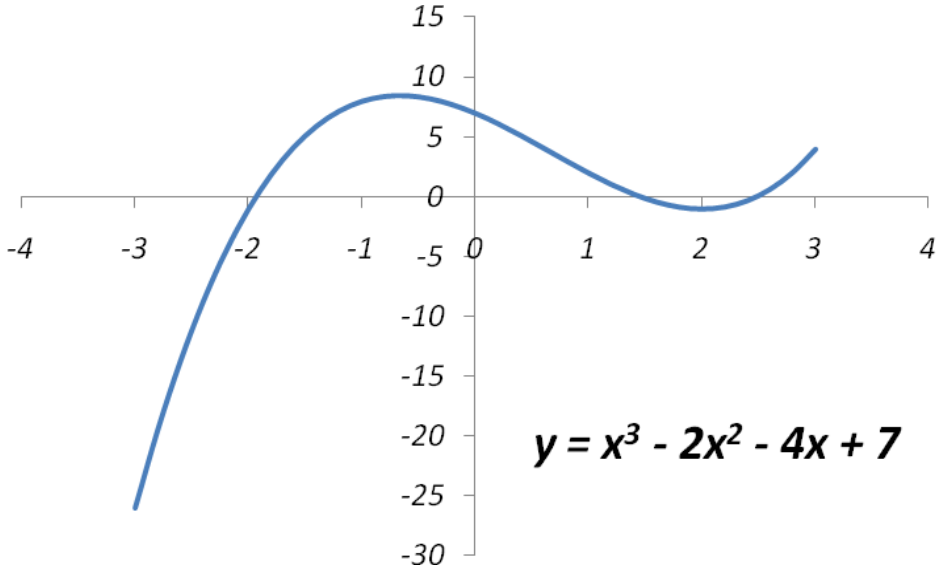


Рис. 9

Три корня лежат в интервалах $x_1 \in [-2; -1]$, $x_2 \in [1; 2]$, $x_3 \in [2; 3]$. Уточним корень x_1. Учитывая, что $f(-2) < 0$, $f(-1) < 0$, $f''(x) = 6x - 4$ и при $-2 \le x \le -1$, $f''(x) < 0$, для расчетов примем следующие формулы:

$$x_{\Lambda(i+1)} = x_{\Lambda i} - \frac{f(x_{\Lambda i})}{f'(x_{\Lambda i})}; \quad x_{\Pi(i+1)} = x_{\Lambda i} - \frac{f(x_{\Lambda i})}{f(x_{\Pi i}) - f(x_{\Lambda i})}(x_{\Pi i} - x_{\Lambda i}),$$

где $x_{\Lambda i}$ и $x_{\Pi i}$ – соответственно значение корня по недостатку (слева) и избытку (справа), $x_{\Lambda o} = -2$, $x_{\Pi o} = -1$.

i	$x_{Лi}$	$x_{Пi}$	$x_{Пi}-x_{Лi}$	$f(x_{Лi})$	$f(x_{Пi})$
1	–2	–1	1	–1	8
2	–1,9400	–1,8900	0,0500	–0,0686	0,0645
3	–1,9355	–1,9353	0,0002	–0,0011	0,0020

$x_1 \approx -1{,}935$.

За приближенное значение корня можно взять $\frac{1}{2}(x_{Лi} + x_{Пi})$.

Метод параболической аппроксимации

В этом методе функция f(x) заменяется не линейной, а параболической функцией, что является более точной заменой. Следовательно, метод может обеспечить более быструю сходимость к решению. На первом этапе параболу обычно строят по трем точкам: крайним и средним точкам интервала (a, b), где отделен корень, т.е. (a, f(a)), ((a + b) / 2, f((a + b) / 2)), (b, f(b)).

По полученному уравнению параболы $y = c_2x^2 + c_1x + c_o$ находят приближенный корень, для чего решают уравнение $c_2x^2 + c_1x + c_o = 0$. На втором этапе строят параболу по трем точкам: найденному приближенному корню и двум предыдущим точкам (слева и справа от этой точки), лежащим по разные стороны оси x. Такой вариант выбора точек на практике быстрее приводит к решению по сравнению с вариантом, когда для построения параболы берутся последовательно три последние точки. Такая процедура повторяется до тех пор, пока величина отрезка, внутри которого находится корень, не будет меньше ξ – предварительно заданной погрешности.

Пример

Для уравнения $x \cdot 2^x - 1 = 0$ сделать две итерации методом параболической аппроксимации.

Решение

Отделим корень графически (рис. 10). Корень находится в промежутке [0, 1], т.е. a = 0, b = 1. Выбираем среднюю точку интервала x = 0,5 и вычисляем значение функции в этих точках:

x = 0; f(0) = –1;

x = 0,5; f(0,5) ≈ – 0,293;

x = 1; f(1) = 1.

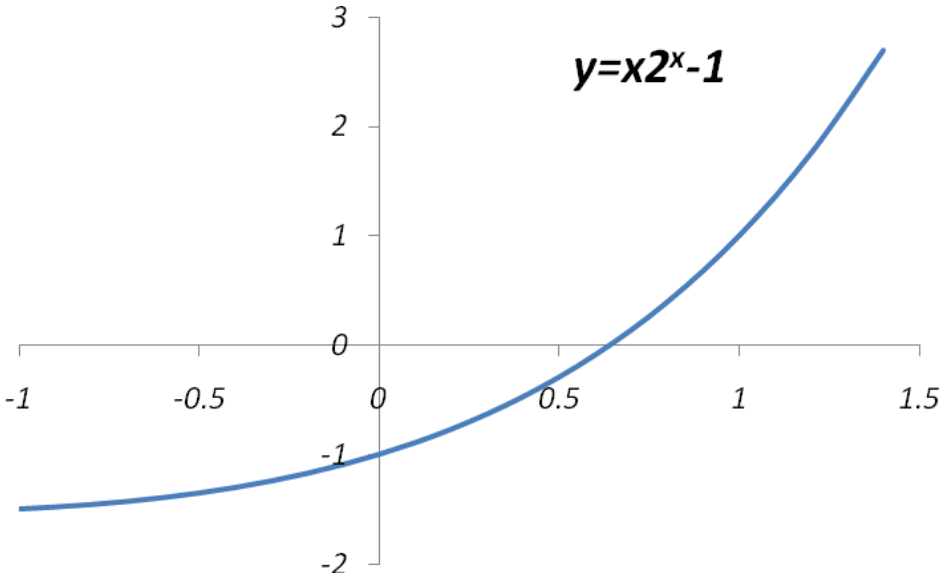


Рис. 10

Через эти три точки проводим параболу $y = c_2x^2 + c_1x + c_o$. Для того чтобы парабола проходила через заданные точки, коэффициенты c_i должны удовлетворять следующим уравнениям:

$$\begin{cases} 0^2 c_2 + 0c_1 + c_0 = -1 \\ (1/2)^2 c_2 + 1/2c_1 + c_0 = -0{,}293 \\ 1^2 c_2 + 1c_1 + c_0 = 1 \end{cases} \quad \begin{cases} c_0 = -1 \\ c_1 = -0{,}828 \\ c_2 = 1{,}172 \end{cases}$$

Для нахождения приближенного значения корня решим уравнение $1{,}172x^2 + 0{,}828x - 1 = 0$, что на участке (0, 1) дает x = 0,636. Найдем значения функции в этой точке

$$f(0{,}636) = 0{,}636 \cdot 2^{0{,}636} - 1 = -0{,}011647.$$

Полученное значение не равно нулю, так как парабола приблизительно описывает исходную функцию. Используя три точки: (0,5; –0,293); (0,636; –0,011647); (1; 1), построим новую параболу. Коэффициенты этой параболы

$$\begin{cases} c_0 = -1 \\ c_1 = 0{,}828. \\ c_2 = 1{,}172 \end{cases}$$

Метод итераций (метод последовательных приближений)

Предварительно исходное уравнение f(x) = 0 преобразуют к виду φ(x) = x. Затем выбирают начальное значение x_o и подставляют его в левую часть уравнения, но $\varphi(x_o) \neq x_o$, так как x_o взято произвольно и не является корнем уравнения. Полученное $\varphi(x_o) = x_1$ рассматривают как очередное приближение к корню. Его снова подставляют в левую часть уравнения $\varphi(x_1)$ и получают следующее значение x_2 ($x_2 = \varphi(x_1)$) и т.д. В общем случае $x_{i+1} = \varphi(x_i)$. Получаемая таким образом последовательность x_o, x_1, x_2, … при определенных условиях может сходиться к корню x* (рис. 11).

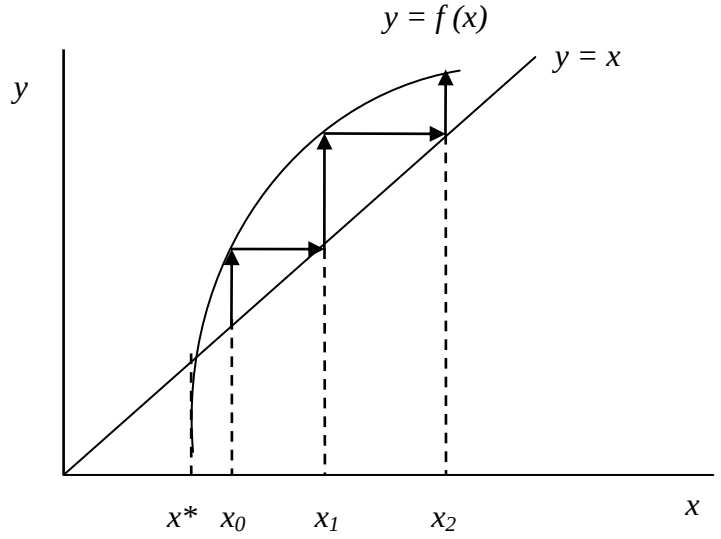


Рис. 11

Таким условием является $|\varphi'(x)| \leq 1$ на [a, b] причем, чем ближе модуль к нулю, тем выше окажется скорость сходимости к решению. В противном случае последовательность расходится от искомого решения (говорят, что «метод не сходится»).

Видно, что последовательность x_o, x_1, x_2, … удаляется от корня. Это всегда будет иметь место, когда тангенс угла наклона φ(x) в окрестности корня по модулю больше единицы.

Существуют различные способы преобразования уравнения f(x)=0 к виду φ(x) = x. Одни могут привести к выполнению условия сходимости всегда, другие – в отдельных случаях.

Самый простой способ: f(x) + x = 0 + x, f(x) + x = φ(x), φ(x) = x , но он не всегда успешен.

Другой способ: φ(x) = x – f(x)/k, причем k выбирается так, чтобы $|k| > Q/2$, где $Q = \max\limits_{[a,b]} |f'(x)|$ и знак k совпадал бы со знаком f ′(x) на [a, b]. Погрешность можно оценить из соотношения

$$|x^* - x_i| \le \frac{q}{1-q}|x_i - x_{i+1}|,$$

где $q = \max\limits_{[a,b]} \varphi'(x)$. Вследствие этого для окончания вычислений в методе итераций применяют соотношение $\frac{q}{1-q}|x - x_{i+1}| \le \xi$.

Часто используют упрощенное окончание поиска $|x_i - x_{i+1}| \le \xi$, не вычисляя максимальное значение производной, но в этом случае погрешность решения может не соответствовать заданной (т.е. быть больше или меньше).

Пример

Уточнить корень погрешностью ξ < 0,001 уравнения 2x + lg(2x + 3) = 1.

Решение

Корень уравнения f(x) = 2x + lg(2x + 3) – 1 = 0 находится в [0; 0,5], т.е. a = 0, b = 0,5. Приведем уравнение к виду, удобному для итераций φ(x) = x. Функцию φ(x) будем искать из соотношения φ(x) = x – f(x)/k, считая для повышения сходимости, что $|k| \ge Q/2$, где $Q = \max\limits_{[0;0,5]} |f'(x)|$; число k имеет тот же знак, что и f ′(x) на [0; 0,5]. Находим

$$f'(x) = 2 + \frac{0{,}8686}{2x+3}, \quad Q = \max_{[0;0,5]} f'(x) = 2 + \frac{0{,}8686}{2 \cdot 0 + 3} \approx 2{,}2895,$$

f ′(x) > 0 при 0 ≤ x ≤ 0,5.

Примем k = 2, тогда $\varphi(x) = x - f(x)/2 = 0{,}5 - 0{,}5 \cdot \lg(2x + 3)$. За начальное приближение примем $x_o = 0$, остальные приближения будем определять из $x_{i+1} = 0{,}5 - 0{,}5 \cdot \lg(2x_i + 3)$.

i	$\mathbf{x_i}$	$\mathbf{\varphi(x) = x_{i+1}}$
0	0	0,2614
1	0,2614	0,2266
2	0,2266	0,2309
3	0,2309	0,2303
4	0,2303	0,2304
5	0,2304	

$x \approx 0{,}230$. Блок-схема метода итераций приведена на рис. 12.

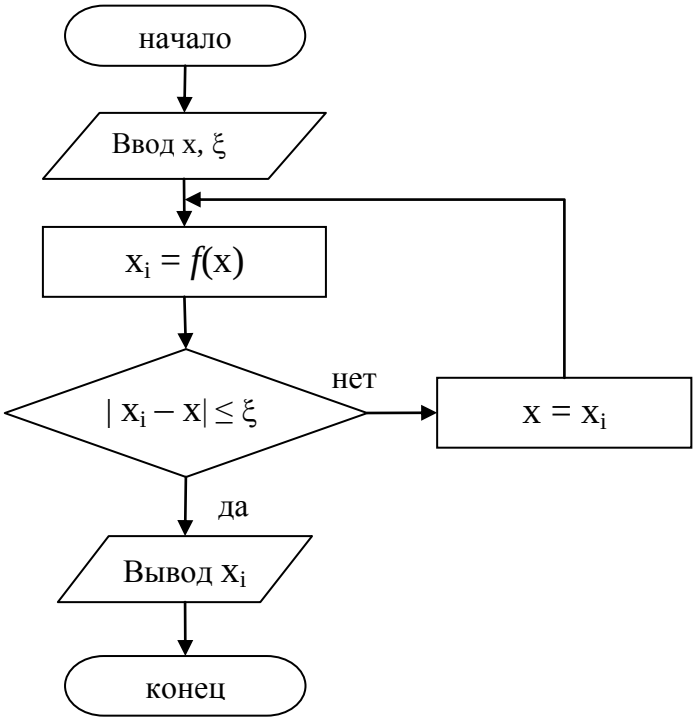


Рис. 12.

ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Пусть для вычисления неизвестных x_1, x_2, …, x_n требуется решить систему n нелинейных уравнений:

$$\begin{cases} f_1(x_1, x_2, ...x_n) = 0 \\ f_2(x_1, x_2, ...x_n) = 0 \\ ... \\ f_n(x_1, x_2, ...x_n) = 0. \end{cases}$$

Метод Ньютона – Рафсона

Метод базируется на разложении функций $f_j(x_1, x_2, …, x_n)$, j=1, 2, …, n в ряд Тейлора с отбрасыванием всех нелинейных членов разложения (т.е. функции линеаризуют). Затем осуществляется переход по всем переменным из текущей точки x^i в следующую x^{i+1}, которую считают решением, т.е. находят $h^i = x^{i+1} - x^i$, поэтому полагают $f_j(x^{i+1}) = 0$ (j = 1, 2 … n). Получается система линейных уравнений относительно приращений h_j^i (j = 1, 2 … n):

$$\begin{cases} f_1(x_1^i, x_2^i, ... x_n^i) + \dfrac{\partial f_1}{\partial x_1}(x_1^i, x_2^i, ... x_n^i) \cdot h_1^i + \dfrac{\partial f_1}{\partial x_2}(x_1^i, x_2^i, ... x_n^i) h_2^i + ... \dfrac{\partial f_1}{\partial x_n}(x_1^i, x_2^i, ... x_n^i) h_n^i = 0 \\ \quad ... \\ f_n(x_1^i, x_2^i, ... x_n^i) + \dfrac{\partial f_n}{\partial x_1}(x_1^i, x_2^i, ... x_n^i) \cdot h_1^i + \dfrac{\partial f_n}{\partial x_2}(x_1^i, x_2^i, ... x_n^i) h_2^i + ... \dfrac{\partial f_n}{\partial x_n}(x_1^i, x_2^i, ... x_n^i) h_n^i = 0, \end{cases}$$

которую решают. С найденными приращениями h_j переходят в новую точку по всем переменным $x_j^{i+1} = x_j^i + h_j^i$ (j = 1, 2, … n). Однако вследствие линейного приближения новая точка x^{i+1} не является решением, т.е. $f_j(x^{i+1}) \neq 0$ (j = 1, 2, … n), ее считают следующим приближением и повторяют всю процедуру до тех пор, пока итерационный процесс не сойдется при удачно выбранном начальном приближении. Причем

$$x^{i+1} = x^i - W^{-1}(x^i) f(x^i),$$

где

$x = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}$, матрица аргументов; $f = \begin{pmatrix} f_1 \\ f_2 \\ \dots \\ f_n \end{pmatrix}$, матрица функций;

W(x) – матрица Якоби системы функций $f_1, f_2, \dots, f_n$ относительно переменных $x_1, x_2, \dots, x_n$

$$W(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ & \dots & & \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

$W^{-1}(x)$ – обратная матрица. Причем определитель матрицы – якобиан J – должен быть $\neq 0$ на каждой итерации.

Окончание поиска решения осуществляется по условию малости шагов по всем переменным одновременно, например, при выполнении условия $|h_j^i| \le \xi$ j = 1, 2, … n, ξ – заданная погрешность. Можно также использовать условие отклонений функций $f(x_1, x_2, \dots, x_n)$ от нуля, например $\max\{f_j(x^j)\} \le \xi$, или потребовать выполнения обоих условий одновременно.

Пример

Найти положительные решения системы с точностью ξ=0,001:

$$\begin{cases} f_1(x_1, x_2) \equiv x_1 + 3\lg x_1 - x_2^2 = 0 \\ f_2(x_1, x_2) \equiv 2x_1^2 - x_1 x_2 - 5x_1 + 1 = 0. \end{cases}$$

Решение

Отделим корни графически.

Кривые системы пересекаются приблизительно в точках M_1 и M_2(3,4; 2,2). Начальное приближение

$$x^{(0)} = \begin{pmatrix} 3{,}4 \\ 2{,}2 \end{pmatrix}.$$

Вычислим второе приближение, полагая

$$f(x) = \begin{pmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{pmatrix},$$

имеем

$$f(x^{(0)}) = \begin{pmatrix} 3{,}4 + 3\lg 3{,}4 - 2{,}2^2 \\ 2 \cdot 3{,}4^2 - 3{,}4 \cdot 2{,}2 - 5 \cdot 3{,}4 + 1 \end{pmatrix} = \begin{pmatrix} 0{,}1544 \\ -0{,}3600 \end{pmatrix}.$$

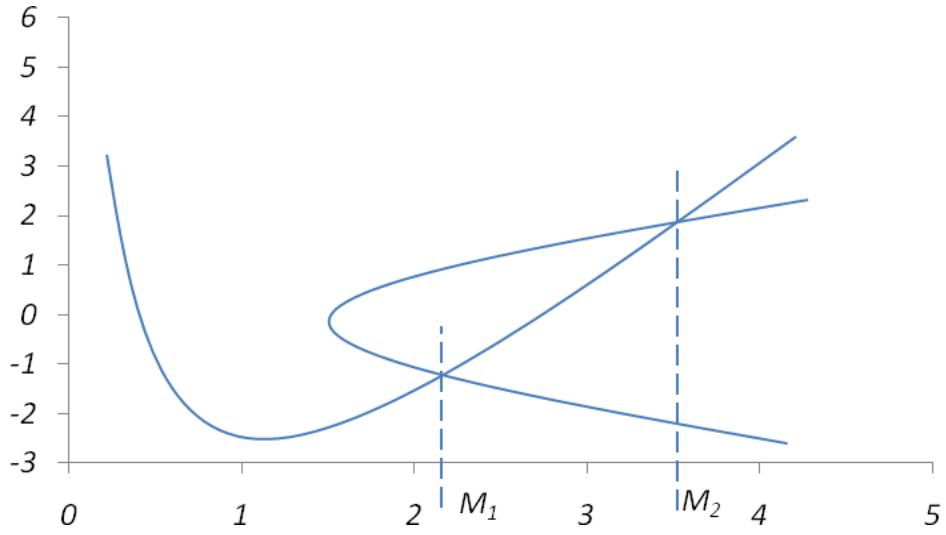


Рис. 13

Составим матрицу Якоби

$$W(x) = \begin{pmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_1}{\partial x_2} \\ \dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 1 + \dfrac{3 \cdot 0{,}43429}{x_1} - 2x_2 \\ 4x_1 - x_2 - 5 - x_1 \end{pmatrix}.$$

Отсюда

$$W(x^{(0)}) = \begin{pmatrix} 1 + \dfrac{3 \cdot 0{,}43429}{3.4} - 2 \cdot 2{,}2 \\ 4 \cdot 3{,}4 - 2{,}2 - 5 - 3{,}4 \end{pmatrix},$$

причем якобиан

$$J = \det W(x^{o}) = 23{,}4571 \neq 0.$$

Обратная матрица

$$W^{-1}(x^{(0)}) = \frac{1}{J}\begin{pmatrix} -3,4 & 4,4 \\ -6,4 & 1,3832 \end{pmatrix}.$$

Следующее приближение:

$$x^{(1)} = \begin{pmatrix} 3,4 \\ 2,2 \end{pmatrix} - \frac{1}{23,4571}\begin{pmatrix} -3,4 & 4,4 \\ -6,4 & 1,3832 \end{pmatrix}\begin{pmatrix} 0,1500 \\ -0,3600 \end{pmatrix} =$$

$$= \begin{pmatrix} 3,4 \\ 2,2 \end{pmatrix} - \frac{1}{23,4571}\begin{pmatrix} -2,10896 \\ -1,48604 \end{pmatrix} = \begin{pmatrix} 3,4 \\ 2,2 \end{pmatrix} + \begin{pmatrix} 0,0899 \\ 0,0633 \end{pmatrix} = \begin{pmatrix} 3,4899 \\ 2,2633 \end{pmatrix}$$

Аналогично для $x^{(2)}$, $x^{(3)}$…

i	$\mathbf{x_1}$	$\mathbf{h_1}$	$\mathbf{x_2}$	$\mathbf{h_2}$
0	3,4	0,0899	2,2	0,0633
1	3,4899	–0,0008	2,2633	–0,0012
2	3,4891	–0,0016	2,2621	–0,0005
3	3,4875		2,2616	

Останавливаемся на $x^{(3)}$

$x_1 \approx 3,4875$; $x_2 \approx 2,2616$, причем

$$f(x^{(3)}) = \begin{pmatrix} 0,0002 \\ 0,0000 \end{pmatrix}.$$

Метод итераций

В этом методе предварительно все уравнения приводят к виду $x = \varphi(x)$, т.е.

$$\begin{cases} x_1 = \varphi_1(x_1, x_2, ...x_n) \\ x_2 = \varphi_2(x_1, x_2, ...x_n) \\ ... \\ x_n = \varphi_2(x_1, x_2, ...x_n) \end{cases}$$

Берется произвольное начальное значение $x^{(o)}$ ($x_1^{(o)}$, $x_2^{(o)}$, … , $x_n^{(o)}$) и подставляется в уравнение. Из каждого уравнения системы находятся новые значения $x^{(1)}$ ($x_1^{(1)}$, $x_2^{(1)}$, … , $x_n^{(1)}$) и т.д. При выполнении условий

$\sum_{i=1}^{n} M_{ki} < 1$ или $\sum_{k=1}^{n} M_{ki} < 1$, где $M_{ki} = \max \left| \frac{\partial \varphi_k}{\partial x_i} \right|$

последовательность $x^{(0)}$, $x^{(1)}$, $x^{(2)}$… сходится к решению.

Для обеспечения сходимости метода можно использовать следующий прием преобразования исходной системы к виду, удобному для итераций. Пусть $x = (x_1, x_2, \ldots, x_n)$ – матрица-строка аргументов; $\varphi(x) = (\varphi_1(x), \varphi_2(x), \ldots \varphi_n(x))$ матрица-строка функций $\varphi(x)$. Тогда система уравнений примет вид $x = \varphi(x)$.

Перепишем систему в виде $x^{(i+1)} = x^{(i)} + \Lambda f(x) = \varphi(x)$, где матрица Λ определяется как $\Lambda = -(f'(x^{(0)}))^{-1}$, причем $\det f'(x^{(0)}) \neq 0$

Пример

Методом итерации решить систему

$$\begin{cases} x_1^2 + x_2^2 = 1 \\ x_1^3 - x_2 = 0 \end{cases}.$$

Решение

Отделим корни графически (рис. 14). Система имеет два решения, отличающихся знаком.

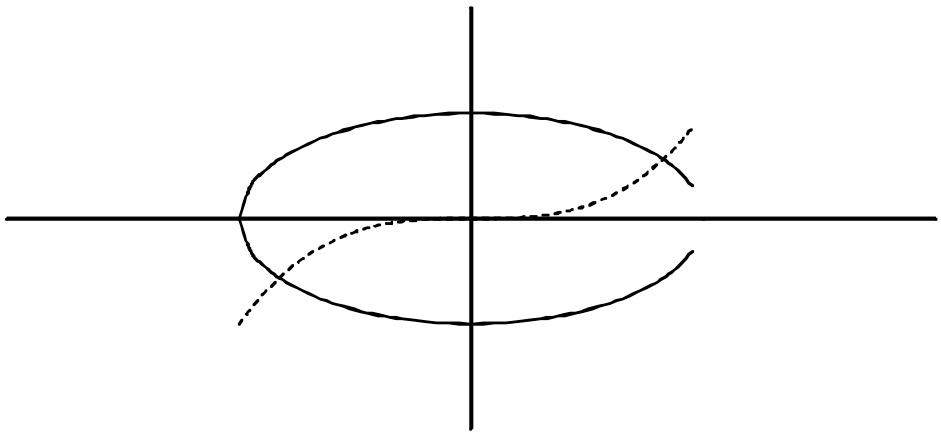


Рис. 14

Найдем положительное решение системы. Примем $x^{(0)} = \begin{pmatrix} 0{,}9 \\ 0{,}5 \end{pmatrix}$.

Полагая $f(x) = \begin{pmatrix} x_1^2 + x_2^2 - 1 \\ x_1^3 - x_2 \end{pmatrix}$, имеем $f'(x) = \begin{pmatrix} 2x_1 & 2x_2 \\ 3x_1^2 & -1 \end{pmatrix}$.

Отсюда $f'(x^{(0)}) = \begin{pmatrix} 1{,}8 & 1 \\ 2{,}43 & -1 \end{pmatrix}$, $\det f'(x^{(0)}) = -1{,}8 - 2{,}43 = -4{,}23 \neq 0$.

Обратная матрица $(f'(x^{(0)}))^{-1} = -\frac{1}{4{,}23}\begin{pmatrix} -1 & -1 \\ -2{,}43 & 1{,}8 \end{pmatrix}$.

Таким образом $\Lambda = -(f'(x^{(0)}))^{-1} = \frac{1}{4{,}23}\begin{pmatrix} -1 & -1 \\ -2{,}43 & 1{,}8 \end{pmatrix}$.

Так как

$$\varphi(x) = x + \Lambda f(x) = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \frac{1}{4{,}23}\begin{pmatrix} 1 & 1 \\ 2{,}43 & -1{,}8 \end{pmatrix}\begin{pmatrix} x_1^2 + x_2^2 - 1 \\ x_1^3 - x_2 \end{pmatrix},$$ то

$$x^{(1)} = \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \end{pmatrix} - \frac{1}{4{,}23}\begin{pmatrix} 1 & 1 \\ 2{,}43 & -1{,}8 \end{pmatrix}\begin{pmatrix} x_1^{(0)2} + x_2^{(0)2} - 1 \\ x_1^{(0)3} - x_2^{(0)} \end{pmatrix} =$$

$$= \begin{pmatrix} 0{,}9 \\ 0{,}5 \end{pmatrix} - \frac{1}{4{,}23}\begin{pmatrix} 1 & 1 \\ 2{,}43 & -1{,}8 \end{pmatrix}\begin{pmatrix} 0{,}060 \\ 0{,}229 \end{pmatrix} = \begin{pmatrix} 0{,}9 \\ 0{,}5 \end{pmatrix} - \begin{pmatrix} 0{,}0683 \\ -0{,}0630 \end{pmatrix} = \begin{pmatrix} 0{,}8317 \\ 0{,}5630 \end{pmatrix};$$

$$x^{(2)} = \begin{pmatrix} 0{,}8317 \\ 0{,}5630 \end{pmatrix} - \frac{1}{4{,}23}\begin{pmatrix} 1 & 1 \\ 2{,}43 & -1{,}8 \end{pmatrix}\begin{pmatrix} 0{,}8317^2 + 0{,}5630^2 - 1 \\ 0{,}8317^3 - 0{,}5630 \end{pmatrix} = \begin{pmatrix} 0{,}8268 \\ 0{,}5633 \end{pmatrix};$$

$$x^{(3)} = \begin{pmatrix} 0{,}8268 \\ 0{,}5633 \end{pmatrix} - \begin{pmatrix} 0{,}0007 \\ 0{,}0002 \end{pmatrix} = \begin{pmatrix} 0{,}8261 \\ 0{,}5631 \end{pmatrix};$$

$$x^{(4)} = \begin{pmatrix} 0{,}8261 \\ 0{,}5631 \end{pmatrix} - \begin{pmatrix} 0{,}0000 \\ -0{,}0005 \end{pmatrix} = \begin{pmatrix} 0{,}8261 \\ 0{,}5636 \end{pmatrix}.$$

Ограничиваясь $x^{(4)}$, получаем $x_1 \approx 0{,}8261$, $x_2 \approx 0{,}5636$, причем $f(x) = \begin{pmatrix} 0{,}0000 \\ 0{,}0002 \end{pmatrix}$ с точностью 10^{-4}.

МЕТОДЫ ОБРАБОТКИ ТАБЛИЧНЫХ ДАННЫХ

Пусть величина y является функцией аргумента x. Это означает, что любому значению x из области определения поставлено в соответствие значение y. На практике часто неизвестна явная связь между y и x, т.е. невозможно записать эту связь в виде некоторой зависимости y = f(x). В других случаях даже при известной зависимости y = f(x) она настолько сложна (например, содержит трудновычисляемые выражения, интегралы и т.п.), что ее использование в практических целях затруднено.

Пусть дана таблица значений $\{x_i, y_i\}$, i=0, 1, … n и необходимо получить значение величины y в других точках, отличных от x_i, причем точная связь y = f(x) неизвестна. Для этой цели служит задача о приближении (аппроксимации) функции: данную функцию f(x) требуется приближенно заменить (аппроксимировать) некоторой функцией φ(x) так, чтобы отклонение φ(x) от f(x) в заданной области было наименьшим. Функция φ(x) при этом называется аппроксимирующей.

Если приближение строится на заданном дискретном множестве точек $\{x_i\}$, то аппроксимация называется точечной. При построении приближения на непрерывном множестве точек аппроксимация называется непрерывной, или интегральной.

Интерполяция

Одним из основных типов точечной аппроксимации является интерполяция – нахождение значения таблично заданной функции в тех точках внутри данного интервала, где она не задана. Если функция восстанавливается в точках за пределами заданного интервала, то используют термин экстраполяция.

При построении интерполяционной функции φ(x), проходящей через все заданные точки – узлы интерполяции, возникают три проблемы:

- выбор интерполяционной функции φ(x);
- оценка погрешности R(x);
- размещение узлов интерполяции для обеспечения возможной наивысшей точности восстановления функции.

Чаще всего интерполяционную функцию представляют в виде полинома, принимающего в заданных точках x_i те же значения y_i, что и функция f(x), т.е. $\varphi(x_i) = y_i$, i = 0, 1, … n. Точки x_i называются узлами интерполяции, φ(x) – интерполяционный многочлен $\varphi(x) = a_o + a_1x + a_2x^2 + \ldots + a_nx^n$.

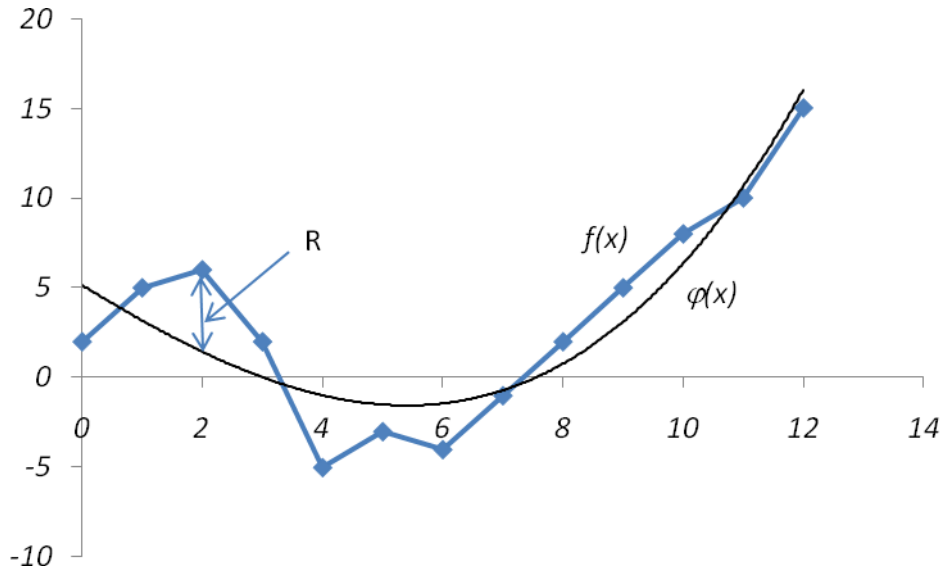


Рис. 15

Метод Лагранжа

Пусть известны значения некоторой функции f(x) в n+1 произвольных различных точках $y_i = f(x_i)$, i = 0,1 … n. Для интерполирования (восстановления) функции в какой-либо точке $x \in [x_o, x_n]$ необходимо построить интерполяционный полином n-го порядка:

$$L_n(x) = y_o Q_o(x) + \ldots + y_j Q_j(x) + \ldots + y_n Q_n(x),$$

где

$$Q_j(x) = \frac{(x-x_0)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_n)}{(x_j-x_0)\ldots(x_j-x_{j-1})\ldots(x_j-x_n)}.$$

Оценить погрешность интерполяции в точке $x \in [x_o, x_n]$ можно по формуле

$$R(x) = \left|f(x) - L_n(x)\right| \le \frac{M_{n+1}}{(n+1)!}\prod_{i=0}^{n}\left|(x-x_i)\right|,$$

где $M_{n+1} = \max\left|f^{(n+1)}(x)\right|$ – максимальное значение (n + 1)-й производной исходной функции f(x) на отрезке $[x_o, x_n]$.

Пример

Дана таблично заданная функция:

x	0	1	2	6
y	–1	–3	3	1187

Требуется найти y при x = 4.

Решение

n = 3

$$L_3(x) = y_0 \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} + y_1 \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} +$$

$$+ y_2 \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} + y_3 \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} =$$

$$= -1\frac{(4-1)(4-2)(4-6)}{(0-1)(0-2)(0-6)} - 3\frac{(4-0)(4-2)(4-6)}{(1-0)(1-2)(1-6)} + 3\frac{(4-0)(4-1)(4-6)}{(2-0)(2-1)(2-6)} +$$

$$+ 1187\frac{(4-0)(4-1)(4-2)}{(6-0)(6-1)(6-2)} = 255.$$

Так как отсутствует информация о значении четвертой производной исходной функции, то оценить погрешность сложно.

Пример

Дана функция y = $\sqrt{x}$ (для иллюстрации метода считаем ее «сложной»). Найти погрешность интерполяции функции при x = 115.

Решение

Знаем, что

x	100	121	144
y	10	11	12

Всего три узла интерполяции, n = 2. Оценим максимальное значение третьей производной

$$y' = \frac{x^{-1/2}}{2},\ y'' = -\frac{x^{-3/2}}{4},\ y''' = 3\frac{x^{-5/2}}{8}.$$

$$M_3 = \max\left|y'''\right| = 3\frac{(100)^{-5/2}}{8}.$$

Погрешность при интерполяции по трем узлам:

$$R \le \frac{3 \cdot 100^{-5/2} \left|(115-100)(115-121)(115-144)\right|}{8 \cdot 3!} \approx 1{,}6 \cdot 10^{-3}.$$

Метод Ньютона

Пусть известны значения некоторой функции f(x) в n + 1 произвольных, попарно не совпадающих точках y = f(x), i = 0, … , n. В общем случае интерполяция по формулам Ньютона может производиться для произвольно расположенных точек – узлов интерполяции, но чаще применяется для равномерно расположенных.

Рассмотрим случай с равномерным расположением узлов

$x_{i+1} = x_i + h$, где $h = (x_o - x_n)/n$.

Замечание

Пусть y=f(x) – заданная функция. Обозначим через $\Delta x = h$ фиксированную величину аргумента (шаг), тогда выражение

$$\Delta y \equiv \Delta f(x) = f(x + \Delta x) - f(x)$$

называется первой конечной разностью функции y. Аналогично определяются конечные разности высших порядков.

$$\Delta^n y = \Delta(\Delta^{n-1} y), \quad (n = 2, 3, \ldots).$$

Например, $\Delta^2 y = \Delta[f(x + \Delta x) - f(x)] = [f(x + 2\Delta x) - f(x + \Delta x)] - [f(x + \Delta x) - f(x)]$.

Пример

Построить конечные разности для функции $P(x) = x^3$, считая шаг $\Delta x = 1$.

Решение

$\Delta P(x) = (x + 1)^3 - x^3 = 3x^2 + 3x + 1.$

$\Delta^2 P(x) = [3(x + 1)^2 + 3(x + 1) + 1] - (3x^2 + 3x + 1) = 6x + 6.$

$\Delta^3 P(x) = [6(x + 1) + 6] - (6x + 6) = 6.$

$\Delta^n P(x) = 0, \; n > 3.$

Интерполяционный многочлен Ньютона имеет вид

$$P_n(x) = y_0 + \frac{\Delta y_0 (x - x_0)}{h} + \frac{\Delta^2 y_0 (x - x_0)(x - x_1)}{2! h^2} + \frac{\Delta^3 y_0 (x - x_0)(x - x_1)(x - x_2)}{3! h^3} + \ldots +$$
$$+ \frac{\Delta^n y_0 (x - x_0)(x - x_1)\ldots(x - x_{n-1})}{n! h^n}.$$

Часто вводят безразмерную величину q, показывающую, сколько содержится шагов от x_o до заданной точки x

$q = (x - x_o)/h.$

$$P_n(x) = y_0 + \Delta y_0 q + \frac{\Delta^2 y_0 q(q-1)}{2!} + \frac{\Delta^3 y_0 q(q-1)(q-2)}{3!} + \ldots +$$
$$+ \frac{\Delta^n y_0 q(q-1)\ldots(q-n+1)}{n!}.$$

Обе формулы называются первой интерполяционной формулой Ньютона и применяются при интерполяции вперед (в сторону увеличения x) или при экстраполяции назад (левее x_o).

Второй интерполяционный многочлен Ньютона применяется при интерполяции назад (т.е. в конце интервала, но левее x_n) или при экстраполяции вперед (правее x_n):

$$P_n(x) = y_n + \Delta y_{n-1} q + \frac{\Delta^2 y_{n-2} q(q+1)}{2!} + \frac{\Delta^3 y_{n-3} q(q+1)(q+2)}{3!} + \ldots +$$
$$+ \frac{\Delta^n y_0 q(q+1)\ldots(q+n-1)}{n!}.$$

Оценить погрешность интерполяции можно по формуле

$$R(x) = \left| f(x) - P_n(x) \right| \le \frac{\max\{\left|\Delta^{(n+1)} y(x)\right|\}}{(n+1)!} \left| q(q-1)(q-2)\ldots(q-n) \right|.$$

Пример

Даны следующие точки:

x	2,0	2,1	2,2	2,3	2,4	2,5	2,6
y	0,0540	0,0440	0,0355	0,0283	0,0224	0,0175	0,0136

Необходимо найти y(2,05).

Решение

Используем для интерполяции первые три точки, остальные используем для оценки погрешности.

Таким образом n=2, h=0,1, $Q = (x - x_o)/h = 0,5$.

Составим таблицу конечных разностей и воспользуемся интерполяционной формулой Ньютона

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$
2,0	0,0540	–0,0100	0,0015	–0,0002
2,1	0,0440	–0,0085	0,0013	0
2,2	0,0355	–0,0072	0,0013	–0,0003
2,3	0,0283	–0,0059	0,0010	–0,0010
2,4	0,0224	–0,0049	0	
2,5	0,0175	–0,0049		
2,6	0,0136			

$y(2,05) = 0,0540 + 0,5(-0,01) + 0,5(0,5 - 1)0,0015/2! = 0,0488125.$

Оценим погрешность найденного y

$$\max\{ |\Delta^3 y(x)| \} = 0,0010.$$

$$R \le \frac{0,001|0,5(0,5-1)0,5-2)|}{3!} = 0,0000625.$$

Метод сплайнов

Часто возникает задача восстановления не только значений функции, но и ее первой и второй производной. Для решения такой задачи применяется сплайновая интерполяция.

Сплайн – это функция, которая на каждом межузловом интервале совпадает с некоторым полиномом, своим для каждого интервала. Полиномы соседних интервалов стыкуются так, чтобы функция была непрерывной. Дополнительно требуется непрерывность нескольких производных.

Наибольшее распространение получила интерполяция кубическими сплайнами. Кубический сплайн склеивается из полино-

мов третьей степени, которые для i-го участка записываются следующим образом:

$$y = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i.$$

Для всего интервала будет n кубических полиномов, отличающихся коэффициентами a_i, b_i, c_i, d_i. Чаще всего узлы при сплайновой интерполяции располагаются равномерно, $x_{i+1} - x_i = h = \text{const}$.

Таким образом, необходимо найти четыре коэффициента при условии прохождения каждого полинома через две точки (x_i, y_i) и (x_{i+1}, y_{i+1}). Первое условие соответствует прохождению полинома через начальную точку, второе – через конечную точку.

$$y_i = d; \qquad y_{i+1} = a_i h^3 + b_i h^2 + c_i h + d_i \quad (i = 0, 1, 2, \ldots n - 1).$$

Найти все коэффициенты нельзя, так как уравнений больше, чем неизвестных. Дополним уравнения условием гладкости функции (т.е. непрерывности первой производной) и гладкости первой производной (т.е. непрерывности второй производной) в узлах интерполяции. Математически эти условия записываются как равенства первой и второй производных в конце i-го и в начале (i + 1)-го участков. Так как

$$y' = 3a_i(x - x_i)^2 + 2b_i(x - x_i) + c_i$$
$$y'' = 6a_i(x - x_i) + 2b_i,$$

то

$$3a_i h^2 + 2b_i h + c_i = c_{i+1}, \quad (i = 0, 1, 2, \ldots, n - 2).$$
$$6a_i h + 2b_i = 2b_{i+1}, \quad (i = 0, 1, 2, \ldots, n - 2).$$

Получают систему линейных уравнений для всех участков, содержащую 4n – 2 уравнения с 4n неизвестными (a_1, a_2, …a_n, b_1, …d_n – коэффициенты сплайнов). Для решения системы добавляют два граничных условия одного из следующих видов:

1) $y''(x_o) = y''(x_n) = 0$
2) $y'(x_o) = g_o$, $y'(x_n) = g_n$
3) $y'(x_o) = y'(x_n) = 0$
4) $y'(x_o) = y'(x_n)$, $y''(x_o) = y''(x_n)$.

Совместное решение 4n уравнений позволяет найти все 4n коэффициента.

Среднеквадратичное приближение

При большом количестве узлов интерполяции получается высокая степень многочлена $\varphi(x) = a_o + a_1 x + \ldots + a_n x^n$. Кроме того, табличные данные могли быть получены путем измерений и содержать ошибки. Построение аппроксимирующего многочлена с условием обязательного прохождения его графика через эти экспериментальные точки означало бы повторение допущенных при измерениях ошибок. В этом случае строится аппроксимирующая функция, в целом наиболее близко проходящая около данных точек.

Критерием близости исходной и аппроксимирующей функций при среднеквадратичном приближении является минимальное значение суммы квадратов отклонений между значениями аппроксимирующего многочлена и заданных таблично

$$R = \sum_{i=0}^{n} \beta_i [\varphi(x_i) - f(x_i)]^2 ,$$

где β – весовые коэффициенты, учитывающие важность i-й точки.

При среднеквадратичном приближении степень аппроксимирующего полинома m меньше n:

$\varphi(x) = a_o + a_1 x + \ldots + a_m x^m$ (m < n).

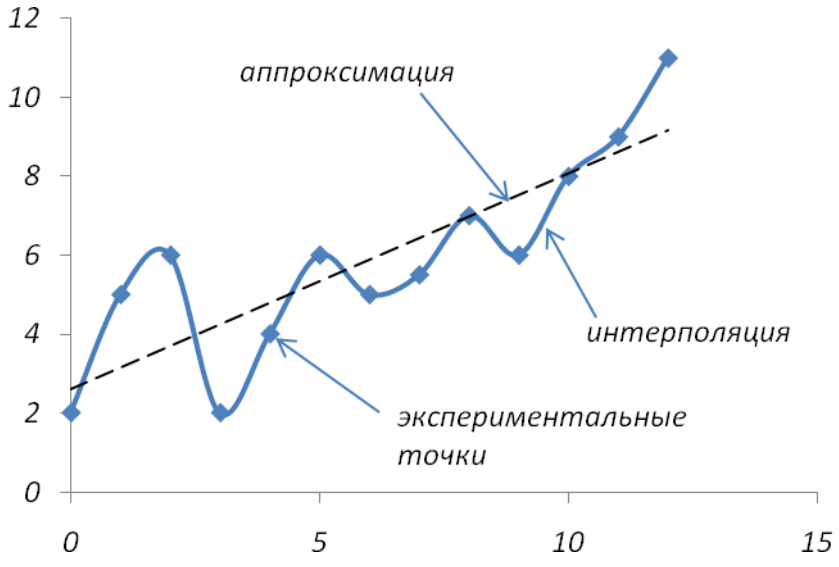


Рис. 16

На практике стараются подобрать аппроксимирующий многочлен как можно меньшей степени (m = 1, 2, 3). При построении аппроксимирующего полинома подбираются коэффициенты a_o,

a_1, … , a_m таким образом, чтобы величина R была наименьшей – это так называемый метод наименьших квадратов (МНК).

Замечание

В теории вероятностей доказывается, что полученные таким методом значения параметров a_o, a_1, … a_m наиболее вероятны, если отклонения $\varphi(x_i) - f(x_i)$ подчиняются нормальному закону распределения.

Коэффициенты a_o, a_1, … , a_m находят из необходимого условия минимума функции R, для которой эти коэффициенты являются независимыми переменными (β=1):

$$\begin{cases} \frac{\partial R}{\partial a_0} = 0 \\ \frac{\partial R}{\partial a_1} = 0 \\ \dots \\ \frac{\partial R}{\partial a_m} = 0 \end{cases} \qquad \begin{cases} \frac{\partial R}{\partial a_0} = 2\sum_{i=0}^{n}(a_0 + a_1 x_i + \dots a_m x_i^m - f(x_i)) = 0 \\ \frac{\partial R}{\partial a_1} = 2\sum_{i=0}^{n}(a_0 + a_1 x_i + \dots a_m x_i^m - f(x_i)) \cdot x_i = 0 \\ \dots \\ \frac{\partial R}{\partial a_m} = 2\sum_{i=0}^{n}(a_0 + a_1 x_i + \dots a_m x_i^m - f(x_i)) \cdot x_i^m = 0. \end{cases}$$

После сокращения на два, раскрытия скобок и изменения порядка суммирования получим так называемую систему нормальных уравнений, из решения которой находим параметры a_o, a_1, … a_m:

$$\begin{cases} (n+1)a_0 + a_1\sum_{i=0}^{n} x_i + a_2\sum_{i=0}^{n} x_i^2 + \dots a_m\sum_{i=0}^{n} x_i^m = \sum_{i=0}^{n} y_i \\ a_0\sum_{i=0}^{n} x_i + a_1\sum_{i=0}^{n} x_i^2 + a_2\sum_{i=0}^{n} x_i^3 + \dots a_m\sum_{i=0}^{n} x_i^{m+1} = \sum_{i=0}^{n} x_i y_i \\ \dots \\ a_0\sum_{i=0}^{n} x_i^m + a_1\sum_{i=0}^{n} x_i^{m+1} + a_2\sum_{i=0}^{n} x_i^{m+2} + \dots a_m\sum_{i=0}^{n} x_i^{2m} = \sum_{i=0}^{n} x_i^m y_i \end{cases}$$

Пример

Необходимо найти аппроксимирующую функцию в виде линейного полинома $y=a_o+a_1x$ по имеющимся экспериментальным данным

x	–26	–22	–16	–11	–5	3	10	25	42
y	66,7	71,0	76,3	80,6	85,7	92,9	99,4	113,6	125,1

Решение

Составим систему линейных уравнений относительно a_o и a_1

$$\begin{cases} a_0(n+1) + a_1\sum_{i=0}^{n} x_i = \sum_{i=0}^{n} y_i \\ a_0\sum_{i=0}^{n} x_i + a_1\sum_{i=0}^{n} x_i^2 = \sum_{i=0}^{n} x_i y_i \end{cases}$$

$n = 8 \quad \Sigma x_i = 0 \quad \Sigma x_i^2 = 4060 \quad \Sigma y_i = 811{,}3 \quad \Sigma y_i x_i = 3534{,}8$

$$\begin{cases} 9a_0 = 811{,}3 \\ 4060a_1 = 3534{,}8 \end{cases} \qquad \begin{cases} a_0 = 90{,}1 \\ a_1 = 0{,}87. \end{cases}$$

Аппроксимирующая функция $y = 90{,}1 + 0{,}87x$.

Пример

Используя метод МНК, вывести эмпирическую формулу для функции y = f(x), заданной в табличном виде

x	0,75	1,50	2,25	3,00	3,37
y	2,50	1,20	1,12	2,25	4,28

Решение

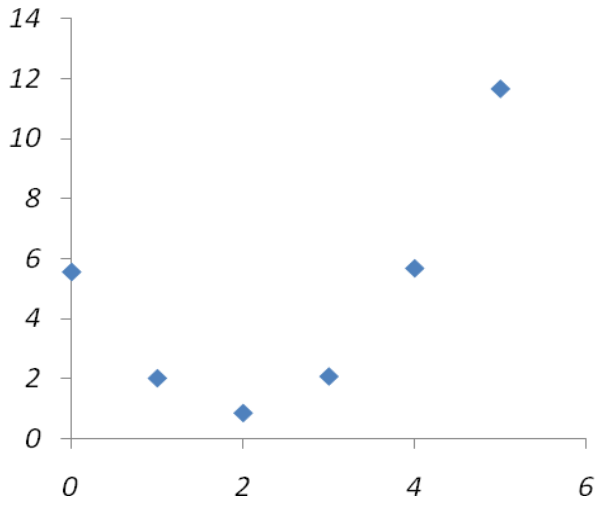


Рис. 17

Изобразим табличные значения графически, откуда видно, что в качестве аппроксимирующей функции можно взять параболу

$$\varphi(x) = a_o + a_1x + a_2x^2,$$

$m = 2 \quad n = 4.$

Система нормальных уравнений имеет вид

$$\begin{cases} 5a_0 + a_1\sum_{i=0}^{n} x_i + a_2\sum_{i=0}^{n} x_i^2 = \sum_{i=0}^{n} y_i \\ a_0\sum_{i=0}^{n} x_i + a_1\sum_{i=0}^{n} x_i^2 + a_2\sum_{i=0}^{n} x_i^3 = \sum_{i=0}^{n} x_i y_i \\ a_0\sum_{i=0}^{n} x_i^2 + a_1\sum_{i=0}^{n} x_i^3 + a_2\sum_{i=0}^{n} x_i^4 = \sum_{i=0}^{n} x_i^2 y_i \end{cases}$$

$\Sigma x_i = 11{,}25$ $\Sigma x_i^2 = 30{,}34$ $\Sigma x_i^3 = 94{,}92$ $\Sigma x_i^4 = 309{,}76$

$\Sigma y_i = 11{,}35$ $\Sigma x_i y_i = 29{,}00$ $\Sigma x_i^2 y_i = 90{,}21.$

$$\begin{cases} 5a_0 + 11{,}25a_1 + 30{,}94a_2 = 11{,}35 \\ 11{,}25a_0 + 30{,}94a_1 + 94{,}92a_2 = 29{,}00 \\ 30{,}94a_0 + 94{,}92a_2 + 309{,}76a_2 = 90{,}21 \end{cases} \qquad \begin{cases} a_0 = 5{,}54 \\ a_1 = -4{,}73 \\ a_2 = 1{,}19 \end{cases}$$

$\varphi(x) = 5{,}54 - 4{,}73x + 1{,}19x^2.$

ЗАДАЧИ

1. Напечатать значение заданного двузначного (в десятичной записи) натурального числа:

а) в десятичном представлении;

б) в двоичном представлении;

в) в троичном представлении;

г) в десятичном представлении с инверсным (обратным) порядком цифр.

2. Проверить для любых чисел x, y, z истинность неравенства $x - y < z$. Ответ напечатать:

а) явно;

б) в виде значения булевского выражения.

3. Проверить принадлежность точки с заданными координатами кругу с заданным радиусом и координатами центра (система координат – декартовая).

4. Вычислить значение числовой функции, заданной кусочной схемой

$$y=\begin{cases} x^2-1, \ \textit{если} \ \ 0<|x|\le 1; \\ \pi/x, \ \ \textit{если} \ \ |x|>1 \\ \pi, \ \textit{если} \ \ x=0 \end{cases}$$

5. Написать программу, печатающую по заданному значению десятичной цифры ее эквивалент в римской нумерации:

а) используя (если это возможно) оператор варианта;

б) моделируя (а) другими структурами управления.

6. Для заданных целых переменных m и n ($m \ge 0$) вычислить:

а) $y = m \times n,$ не используя операцию умножения;

б) $y = n^m$, не используя операцию возведения в степень.

7. Для заданного целого значения переменной m ($m \ge 0$) вычислить y = m!

8. Для заданного натурального значения переменной m вычислить y = m!!. По определению, $m!!=\begin{cases} 1\cdot 3\cdot \ldots \cdot m, \text{если m нечетное} \\ 2\cdot 4\cdot \ldots \cdot m, \text{если m четное} \end{cases}$.

9. Для заданного натурального n найти суммы

а) $y=\sum_{i=1}^{n} i^2$; б) $y=\sum_{i=1}^{n} i!$; в) $y=\sum_{i=1}^{n} i^i$; г) $y=\sum_{i=1}^{n} i!!$.

10. Для заданных натуральных значений переменных m и n, не используя операцию деления, вычислить:

а) результат целочисленного деления m на n;

б) остаток от целочисленного деления m на n.

11. Вычислить значение $y=\sqrt{x}$ с заданной точностью $\varepsilon > 0$ по итерационной формуле Ньютона $y_{i+1}=\frac{1}{2}\left(\frac{x}{y_i}+y_i\right)$, выбрав в качестве $y_0 = x/n$. Необходимая степень точности достигается при $|y_{i+1}-y_i|>\varepsilon$.

12. Для заданного числового значения $x(|x|<1)$ с заданной точностью $\varepsilon > 0$ вычислить e^x по формуле: $e^x = 1+x+\frac{x^2}{2!}+\ldots+\frac{x^n}{n!}+\ldots$ Необходимая степень точности достигается при $\left|\frac{x^n}{n!}\right|<\varepsilon$.

13. Для заданного натурального значения n вычислить сумму $S = \sum_{i=1}^{n} i \times i!$:

а) непосредственным суммированием;

б) используя подпрограмму-функцию;

в) используя подпрограмму-процедуру.

14. Заданы координаты двух точек $P = (p_1, p_2)$, $F = (f_1, f_2)$ и окружность радиуса r с координатами центра (a, б). Проверить, сколько точек (нуль, одна или две) принадлежит внутренности окружности. Проверку принадлежности точки кругу оформить в виде подпрограммы.

15. Заданы стороны двух треугольников ABC (стороны a, b, c) и PLF (стороны p, l, f). Найти сумму и разность площадей треугольников. Вычисление площади треугольника по формуле Герона оформить в виде подпрограммы.

16. Три точки заданы своими декартовыми координатами x = (x_1, x_2), y = (y_1, y_2), z = (z_1, z_2). Вычислить и выдать на печать полярные координаты этих точек, если известно, что полярный радиус p и полярный угол φ для точки x вычисляются по формулам: $p = \sqrt{x_1^2 + x_2^2}, \varphi = arctg \frac{x_2}{x_1}$. Перевод в полярные координаты оформить в виде подпрограммы.

17. Последовательность Фибоначчи (члены которой называются числами Фибоначчи) определяется рекуррентными соотношениями: $F_1 = F_2 = 1$; $F_n = F_{n-1} + F_{n-2}$ для n > 2. Для заданного натурального k найти F_k.

18. Для заданного натурального n ($n \geq 3$) найти все элементы начального отрезка из n членов последовательности Фибоначчи, являющиеся квадратами натуральных чисел.

19. Натуральное число n называется совершенным, если оно равно сумме всех своих делителей (исключая само это число). Определить, является ли заданное натуральное n совершенным или нет.

20. Определить, является ли заданное натуральное число $m \triangleright 1$ простым или нет.

21. Известно (теорема Евклида), что если число $p = 1 + 2 + 2^2 + \ldots + 2^n = 2^{n+1}$-1 простое, то число $2^n p$ является совершенным. Решить задачу 26, воспользовавшись этим фактом.

22. Вычислить для заданных a, x, n: $R = \frac{1}{(n+1)!}\prod_{k=1}^{n}(x - z_k)$ $z_k = a + k \times 0.1$.

23. Вычислить сумму k членов разложения в ряд $\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots + (-1)^k \frac{x^{2k}}{(2k)!} + \ldots$ и результат сравнить с результатом работы встроенной функции cos(x) для различных k.

24. Даны вектор a и значение x. Найти величину $\sum_i a_i\ (x - i)$.

25. Найти в целочисленном векторе первый элемент с простым значением (решить с использованием и без использования процедур).

26. Компоненты a_i вектора a, начиная со второго и заканчивая предпоследним, удовлетворяющие условию $a_{i-1} \le a_i \le a_{i+1}$, заменить на величину $a_i^2 + ba_i + c$, где b и c – заданные значения.

27. Заданный вектор с различными компонентами упорядочить по возрастанию, используя метод простого обмена (поиск минимального элемента и обмен значения с первым элементом неотсортированной части вектора).

28. Заданный вектор упорядочить по возрастанию методом «пузырька».

29. Даны векторы a и x. Преобразовать вектор x по правилу: если $a_i = 0$, то x_i увеличить на 10, в противном случае x_i заменить нулем.

30. Ввести построчно двумерную матрицу известной размерности и вывести ее по столбцам.

31. Каждый элемент трехмерной целочисленной матрицы домножить на сумму его индексов.

32. Поменять местами k-ю и i-ю строки матрицы.

33. Дана строка x. Получить строку Y, состоящую из символов строки x, расположенных в обратном порядке (вертикальное отражение строки).

34. Даны строки x и y. Определить позицию первого слева символа строки x, который входит в строку y.

35. Дан текст, состоящий из слов, чисел и знаков арифметических операций. Заменить в нем слова, описывающие цифры, на сами цифры. Например, ввод: один два 5 + пять ноль – шесть. Вывод: 125 + 50 – 6. Между словами, числами и знаками располагается по одному пробелу.

36. Составьте алгоритм и программу для определения сдачи после покупки в магазине товара: перчаток стоимостью a рублей, портфеля стоимостью b рублей, галстука стоимостью c рублей. Исходная сумма, выделенная на покупку - d рублей. В случае нехватки денег сдача получится отрицательной.

37. Ежегодный прирост рыбы в пруду составляет 15%. Запасы рыбы оценены в A тонн. Ежегодный план отлова – B тонн. Наименьший запас рыбы, ниже которого запас уже не восстанавливается, составляет C тонн. Составьте алгоритм и программу, подсчитывающую, сколько лет можно выдерживать заданный план?

38. Начальный вклад в сберкассу составил A рублей. Через сколько лет он станет больше B рублей? (Каждый год вклад увеличивается на 3%).

39. Составить программу, в которой используется подпрограмма, анализирующая текст, выдающая количество различных букв в этом тексте и печатающая эти буквы в алфавитном порядке. (Например: в тексте «Пример текста» – 9 различных букв: а, е, и, к, м, п, р, с, т).

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

Банди Б. Методы оптимизации: Вводный курс. М., 1988.

Боон К. Паскаль для всех. М., 1988.

Васильков Ю.В., Василькова Н.Н. Компьютерные технологии вычислений в математическом моделировании. М., 1999.

Вирт Н. Алгоритмы и структуры данных: пер. с англ. М., 1989.

Вычислительная математика / Н.И. Данилина и др. М., 1985.

Гулд Х., Тобочник Я. Компьютерное моделирование в физике: в 2 ч. М., 1990.

Демидович Б.П., Марон И.А. Основы вычислительной математики. М., 1966.

Демидович Б.П., Марон И.А., Шувалова Э.З. Численные методы анализа. М., 1967.

Джонс Ж., Харроу К. Решение задач в системе Турбо Паскаль. М., 1991.

Епанешников А., Епанешников В. Программирование в среде Turbo Pascal 7.0. М., 1996.

Зуев Е.А. Язык программирования Turbo Pascal 6.0, 7.0. М., 1993.

Касьянов В.Н., Сабельфельд В.К. Сборник заданий по практикуму на ЭВМ. М., 1986.

Кнут Д. Искусство программирования для ЭВМ: пер. с англ.; в 2 т. М., 1976. Т.1.

Мак-Кракен Д., Дорн У. Численные методы и программирование на Фортране. М., 1977.

Программирование в среде Турбо Паскаль 7.0. М., 1994.

Турчак Л.И. Основы численных методов. М., 1987.

Эберт К., Эдерер Х. Компьютеры. Применение в химии. М., 1988.

П Р И Л О Ж Е Н И Я

Приложение 1

ОБЩИЕ ПРАВИЛА РАБОТЫ В КОМПЬЮТЕРНОМ КЛАССЕ

1. До начала занятия необходимо внимательно ознакомиться с темой работы, используя методические пособия, учебник и конспект лекций. Любая работа только тогда продуктивна, когда выполняется сознательно, с пониманием её теоретического содержания. *Важно*: используйте в работе справочные материалы по изучаемому языку, как бумажные, так и электронные (help).

2. Студенты допускаются к работе в компьютерных залах (классах) только в присутствии преподавателя.

3. В компьютерных классах *категорически запрещается* снимать и развешивать верхнюю одежду, громко разговаривать, принимать пищу, включать и выключать рубильники и оборудование, не относящиеся к работе.

4. Лишние книги и тетради не должны находиться на рабочем столе. Портфели, сумки и другие вещи необходимо помещать в специально отведенные места.

5. Воспрещается запускать программное обеспечение, не относящееся к данной работе, изменять настройки и параметры работы компьютера.

6. Использование внешних накопителей (flash) допускается только с разрешения преподавателя и должно быть сведено к необходимому минимуму.

7. Звуковые сигналы мобильных телефонов во время занятий должны быть *отключены*.

8. Разговаривать разрешается только тихо. Если нужно обратиться к преподавателю или товарищу, находящемуся далеко, следует к нему подойти.

9. По окончании работы следует сохранить при необходимости результаты работы и закрыть все использовавшиеся Вами программы.

(Из документа «Порядок планирования и проведения занятий в компьютерных классах вычислительного центра»).

Приложение 2

ОСНОВНЫЕ ФУНКЦИОНАЛЬНЫЕ КЛАВИШИ IDE BORLAND PASCAL (FREE PASCAL)

Клавиша	Назначение
F1	Вызов справки
Ctrl+F1	Вызов контекстной справки. Вызывается справка по конкретной процедуре (функции) или ошибке компилятора, на которую указывает курсор
F2	Запись активного файла
F3	Открыть файл
F4	Запуск программы с остановкой в месте расположения курсора
F7	Пошаговое выполнение программы с заходом во все процедуры
F8	То же, что и F7, но пропуская процедуры
F9	Компиляция программы
F10	Вход в меню IDE
Ctrl+F9	Запуск программы на выполнение
Ctrl+F2	Сброс выполнения программы (в режиме отладки)
Ctrl+F7	Добавить переменную в окно просмотра
Alt+X	Выход из IDE
Alt+F5	Вызов рабочего окна программы для просмотра результатов её работы
Shift+ cursor	Выделение блока текста
Shift+ Del	Вырезать блок текста
Shift+ Insert	Вставить блок текста из буфера
Ctrl+ Insert	Скопировать блок текста в буфер
Alt+ цифра	Переключение между окнами редактора
Ctrl+ Break	Остановка выполнения программы

СОДЕРЖАНИЕ

Учебное издание

ИНФОРМАТИКА.
ПРОГРАММИРОВАНИЕ И ЧИСЛЕННЫЕ МЕТОДЫ
Лабораторный практикум

В о л ы н к и н Виталий Анатольевич

С у х н о Игорь Владимирович

Б у з ь к о Владимир Юрьевич

Подписано в печать 20.10.2010. Печать цифровая.
Формат 60×84 1/16. Уч. изд. л. 5,3. заказ № 123, тираж 100 экз.
Кубанский государственный университет
350040, г. Краснодар, ул. Ставропольская, 149.

Издательско-полиграфический центр
кубанского государственного университета
350040, г. Краснодар, ул. Ставропольская, 149.