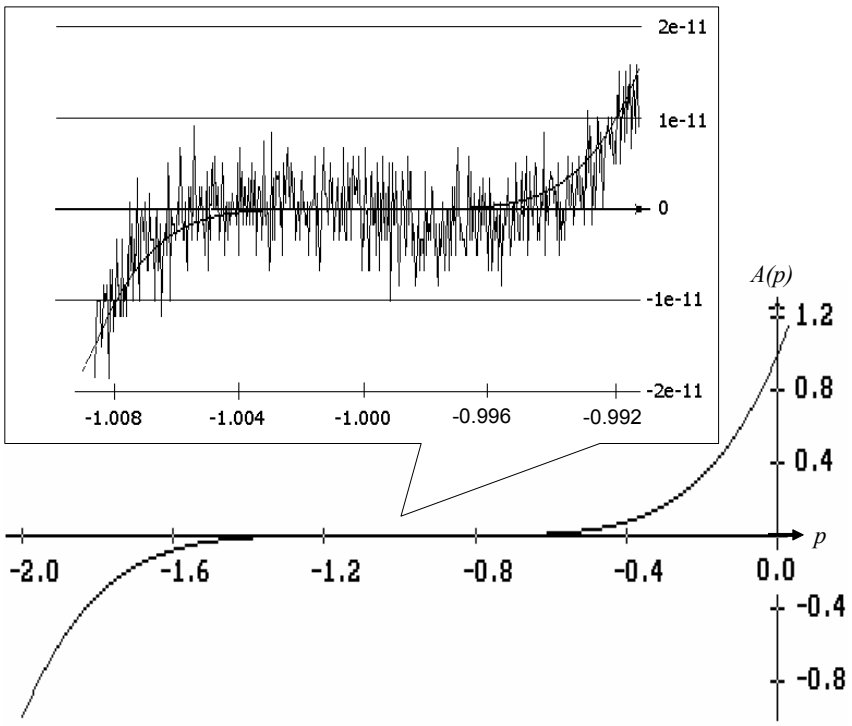
А.И. Жученко
Н.А. Кубрак
И.М. Голинко

БАЗОВЫЕ АЛГОРИТМЫ ЧИСЛЕННОГО АНАЛИЗА



Киев
НУТУ “КПІ”
2006

А.И. Жученко
Н.А. Кубрак
И.М. Голинко

БАЗОВЫЕ АЛГОРИТМЫ ЧИСЛЕННОГО АНАЛИЗА

Рекомендовано Министерством образования и науки Украины в качестве учебного пособия для студентов вузов, обучающихся по специальности «Автоматизация и компьютерно-интегрированные технологии»

Киев
НУТУ “КПІ”
2006

УДК 519.6(075.8)
Ж94

Рецензенты: А.П. Ладанюк, д.т.н., проф. (Национальный университет пищевых технологий)
Ю.К. Тодорцев, д.т.н., проф. (Одесский политехнический университет)

Жученко А.И., Кубрак Н.А., Голинко И.М. Базовые алгоритмы численного анализа: Учебн. Пособие / –К.: НУТУ “КПІ”, 2006. -236с.
ISBN 966-622-200-0

Рассматриваются компьютерные алгоритмы вычисления биноминальных коэффициентов, чисел Фибоначчи, сумм бесконечных рядов. Приводятся алгоритмы анализа одномерных массивов – вычисление сумм, локализация минимальных и максимальных элементов, сортировка массивов, вставка и удаление отдельных элементов, численное дифференцирование и интегрирование функций, заданных таблично.

Демонстрируется широкий спектр операций над полиномами – арифметические действия над ними, определение всех корней полинома различными способами.

Отдельно рассматриваются задачи, связанные с использованием двумерных числовых массивов – решение систем линейных алгебраических уравнений, интерполиррование и сглаживание, аппроксимация полиноминальными и дробно–рациональными зависимостями.

Для студентов направления подготовки “Автоматизация и компьютерно-интегрированные технологии” всех форм обучения, в частности для студентов-иностранцев, обучающихся по данной специальности.

ISBN 966-622-200-0

© А.И. Жученко, Н.А. Кубрак, И.М. Голинко, 2005

Введение

На кафедре автоматизации химических производств Национального технического университета Украины (КПИ) на первом курсе читаются “Численные методы в инженерных и научных расчетах”.

В этом курсе ставится задача подготовить студентов к изучению алгоритмов численного анализа и синтеза систем автоматического управления технологическими процессами и установками. Подобные расчеты могут быть реализованы только на компьютерах. Существует большое количество программных продуктов общематематического и специализированного характера, призванных поддерживать упомянутые расчеты. Но это не снижает необходимости обучать студентов инженерных специальностей элементам программирования. Помимо необходимости разработки пусть даже относительно несложных нестандартных алгоритмов для компьютера понимание принципов и овладение навыками программирования полезно с точки зрения углубления понимания сущности, возможностей и ограничений даже стандартных методов и алгоритмов. Если студент, знакомясь с каким-то алгоритмом, демонстрирует умение зафиксировать его в виде фрагмента программы или, что еще лучше, подпрограммы на каком-то алгоритмическом языке, то это надежно свидетельствует о том, что он разобрался в данном алгоритме настолько, что может обучить ему такого “безмозглого” исполнителя как компьютер.

В настоящем пособии в качестве алгоритмического языка выбран Турбо Паскаль. Почему он? Это один из наиболее простых, понятных (иногда говорят - прозрачных) и доступных языков. Мы не сочли необходимым приводить в пособии описание необходимых средств Турбо Паскаля, для этого существует множество пособий.

Рассмотрению нетривиальных алгоритмов исследования автоматических систем посвящено также много учебных пособий, в частности, созданных преподавателями кафедры, например, [1-4]. А вот как возвести число в степень, вычислить значение факториала, или биноминального коэффициента, найти минимальный, или максимальный элемент массива, вычислить определитель или

решить систему линейных алгебраических уравнений – обучать в институте таким тривиальным вещам считается несолидным. А между тем для многих первокурсников овладение подобными премудростями вызывает определенные проблемы.

Авторы располагают материал по мере нарастания сложности. В каждом параграфе вначале излагается существо рассматриваемого алгоритма, затем этот алгоритм реализуется в виде подпрограммы и уже после этого данная подпрограмма комментируется. Таким образом, читатель, ознакомившись с подпрограммой и убедившись, что ему в ней все понятно, может не терять времени на чтение комментария. Естественно, вариант реализации того или иного алгоритма может показаться читателю не лучшим из возможных. Но, во-первых, “на вкус и цвет товарищей нет”, а во-вторых, авторы и не ставили перед собой задачи создавать идеальные программные продукты. Главным считалась корректность (предлагаемое решение должно приводить к получению правильного результата), а также простота, понятность даже начинающему программисту, на которого, собственно, и ориентировано пособие.

Авторы не сочли целесообразным снабжать пособие заданиями для самостоятельного решения (упражнениями), поскольку заключение о правильности выполнения задания может дать либо эксперт, либо компьютер. Отдельно взятые подпрограммы в Турбо Паскале отдельно не компилируются и не выполняются. Чтобы их оживить их нужно включить в какую-то программу или модуль (в составе модуля их, по крайней мере, можно будет пытаться компилировать), но даже в этом случае оценить правильно ли решается поставленная задача должен специалист. Правильные ответы на ряд контрольных задач являются необходимым, но отнюдь не достаточным условием правильности программного продукта. Итак, доводка до необходимого уровня мастерства программиста должна осуществляться под руководством преподавателя, тогда как предлагаемое пособие должно служить для отработки навыков программирования.

1. Вычисление сумм, произведений, степеней

Обучение программированию задач численного анализа обычно начинают с объяснения правил программирования арифметических выражений. В Турбо Паскале эти правила практически не отличаются от общепринятых в алгебре. Вот разве что для использования в арифметических выражениях в Турбо Паскале применяются только круглые скобки, так как квадратные и фигурные зарезервированы для других целей. Начинающему программисту чтобы не ошибиться при программировании заданной последовательности вычисления арифметического выражения можно рекомендовать “не жалеть скобок”. Можно также разбивать сложные выражения на элементарные фрагменты, вычислять каждый из фрагментов отдельно, а потом уже конструировать выражения из указанных блоков. В данном же разделе мы остановимся на вычислении наиболее часто встречающихся функций, которые не перекрываются стандартными функциями и процедурами Турбо Паскаля.

1.1. Возведение числа типа real в степень типа integer

Итак, требуется вычислить x^n, где x:real, n:integer. В Турбо Паскале есть стандартная функция `sqr(x)`, которая вычисляет x^2.

Если же $n>2$, то $x^n = x \cdot x \cdot x \cdot ... x$, где в произведении участвует n множителей. Если же $n < 0$, то $x^{-|n|} = \frac{1}{x^{|n|}}$, $x^0 = 1$ - согласно определению. Оформим вычисления x^n в виде подпрограммы – функции `Xstn`:

```
Function Xstn(x:real; n:integer):real;
var s:integer; Xs:real;
begin
  Xs:=1;
  for s:=1 to abs(n) do
    Xs:=Xs*X;
  if n<0 then Xs:=1/Xs;
  Xstn:=Xs
end;
```

Прокомментируем эту подпрограмму. Начнем с того, что задача решается в формате подпрограммы – функции (а не подпрограммы-процедуры). Как известно [5], подпрограммы-функции в Турбо Паскале целесообразно использовать тогда, когда результатом работы подпрограммы является отдельное значение. В нашем случае x^n - это отдельное число. Следовательно, задача может быть решена в виде именно подпрограммы-функции.

Подпрограммы-функции в Турбо Паскале вызываются непосредственно в процессе вычисления выражений, тогда как для вызова подпрограммы-процедуры требуется обращаться к оператору вызова процедуры. Вычисленное процедурой значение требуется где-то запоминать, и только затем его можно будет использовать при вычислении соответствующего выражения. Поэтому с подпрограммой-функцией работать проще, почему мы и отдали ей предпочтение.

Назвали мы нашу функцию `Xstn`. Почему так? Вообще-то, кто создает подпрограмму, тот ее и называет (как считает нужным). Лишь бы идентификатор (имя) функции удовлетворял требования Турбо Паскаля к идентификаторам вообще и к именам подпрограмм в частности. Это должна быть произвольная комбинация букв и цифр, начинающаяся с буквы . Наше имя `Xstn` удовлетворяет данному требованию. Второе требование (скорее - пожелание) к идентификатору – он должен ассоциироваться с его сущностью. Мы надеемся, что `Xstn` ассоциируется с текстом “X в степени n”, в чем и состоит сущность подпрограммы.

В скобках после имени функции перечисляются ее параметры `(x:real; n:integer)`. Порядок расположения параметров в списке произволен. Но когда мы воспринимаем x^n, то сначала упоминаем x, а уже потом n. Поэтому мы и поставили x первым, а n - вторым. После каждого параметра через двоеточие указывается его тип. Описание одного параметра отделяется от описания другого точкой с запятой. После списка параметров (в скобках) стоит двоеточие и слово `real`. Это означает, что значение x^n (идентификатор `Xstn`) имеет тип `real`. Указание типа результата в конце

заголовка подпрограммы функции в Турбо Паскале обязательно.

Описательная часть подпрограммы в нашем случае начинается служебным словом `var` (variable), после которого идет описание `S:integer; Xs:real;`. Здесь `s` - счетчик множителей при вычислении выражения $x^n = 1 \cdot x \cdot x \cdot ... \cdot x$, а `Xs` – идентификатор промежуточного значения X в степени.

Обращаем внимание читателя на множитель 1 в выражении для x^n. В данном случае 1 это x^0. А n ведь может быть и нулем (а почему бы и нет?). Вы скажете, зачем прибегать к услугам функции $Xstn$ для вычисления значения x^0, когда ответ и так очевиден – единица? Да, конечно, в отдельно взятом случае замените x^0 единицей – и все дела! Но необходимость вычислять x^0 может возникнуть, например, при вычислении ряда, где в одном из его членов (причем, вовсе не обязательно первом или последнем) может возникнуть ситуация, когда степень x оказывается равной нулю. Так что останавливать процесс, учитывать $x^0 = 1$, а потом только продолжать его? Нелогично. Да и кроме того, создаваемая Вами подпрограмма-функция должна быть достаточно интеллектуальной (умной), чтобы ситуация x^0 не ставила ее в тупик. Ведь подпрограмма пишется не для разового использования, и пользоваться ею, возможно, будете не только Вы и не только для решения отдельной (сиюминутной) проблемы, и поэтому ее желательно сделать по возможности универсальной. Писать программу (и подпрограмму) на один раз – не стоит, да и неинтересно.

Итак, начинаем операторную часть подпрограммы с ее решения для $n = 0$. Если окажется, что $n \neq 0$, то мы этот результат подкорректируем (домножим необходимое число раз на x), а нет, то так и оставим.

В цикле `for s:=1 to abs(n) do` … мы как раз и доводим результат (Xs) “до кондиции”. “Старое” значение Xs (справа от символа присваивания (:=) мы домножаем на x и запоминаем в качестве нового его значения (Xs слева от символа присваивания) – и так столько раз (n раз), чтобы в конце концов получить искомое x^n.

Если $n \geq 0$, то вычисление x^n на этом и завершается, если же $n < 0$, то

полученный результат нужно инвертировать, что и выполняется условным оператором `if n<0 then Xs:=1/Xs.`

И наконец, завершающий штрих `Xstn:=Xs.` А это зачем? Дело в том, что в Турбо Паскале существует четкое правило: в операторной части описания подпрограммы-функции должен присутствовать как минимум один оператор присваивания, в левой части которого фигурирует идентификатор этой подпрограммы. В результате данный идентификатор (имя функции) получает значение, которым возвращается в программу значение вычисляемой функции. Так вот оператор `Xstn:=Xs` как раз и выполняет указанное требование.

У пытливого читателя может возникнуть идея-предложение. А почему бы не начать с того, что вместо `Xs:=1` написать `Xstn:=1`. Это, действительно, можно было бы сделать. Смысл предложения – обойтись без промежуточной переменной `Xs`. Тогда в теле цикла возникла бы конструкция вроде следующей `Xstn:=Xstn*x`. В правой части данного оператора присваивания вызывается `Xstn`, т.е. описываемая функция вызывает сама себя. Это явление носит название рекурсии. В принципе, в Тубо Паскале рекурсия легальна и ничего противозаконного в ее использовании нет. Надо только уметь ею пользоваться. Это своего рода “высший пилотаж” – и это, конечно, шик! Но первоначальное обучение пилотированию начинается с освоения “подскоков” и горизонтального полета. А фигуры высшего пилотажа – потом. Вот и мы рекомендуем читателю оставить рекурсию на потом. Тем более, что практически не существует проблем, которые нельзя было бы решить не прибегая к рекурсии. Элементарный прием – введите промежуточную переменную, вот как мы `Xs` - и рекурсия перестает быть необходимой. Плата не слишком велика, не так ли? Ну, а переписать в конце `Xs` в `Xstn` также не слишком обременительно. Описание подпрограммы в Турбо Паскале заканчивается точкой с запятой.

1.2. Возведение вещественного числа в вещественную степень

Если более формально, то речь идет о вычислении

$$x = a^b, \tag{1.1}$$

где `a:real` и `b:real`. Естественно, что `x:real`. Стандартной функции для реализации указанной операции в Турбо Паскале нет. Возьмем логарифм натуральный от (1.1)

$$\ln x = b \ln a. \tag{1.2}$$

Логарифм, в том числе и натуральный, определен только для положительных чисел, следовательно, в (1.1) число a должно быть положительным. Чтобы определить значение x в (1.1), от выражения (1.2) необходимо взять антилогарифм (натуральный, естественно). А ведь это не что иное как экспонента! Следовательно

$$x = e^{b \ln a}. \tag{1.3}$$

Программно это выглядит так

$$x := \exp(b * \ln(a)). \tag{1.4}$$

Это и есть решение поставленной задачи. Его можно оформить в виде подпрограммы

```
function Astb(a,b:real):real;
begin
  Astb:=exp(b*ln(a))
end;
```

Но “городить” подпрограмму под один единственный оператор присваивания? Проще при программировании соответствующего выражения a^b представить как $\exp(b * \ln(a))$ - только и всего.

1.3. Вычисление факториала

Итак, речь идет о вычислении

$$x = n!, \tag{1.5}$$

где n - целое неотрицательное число.

Кстати, а чему равен $0!$? Некоторые уверены, что он равен нулю. И совершенно напрасно, так как $0!=1$ (по определению!). Ну и, естественно, $1!=1$. Далее – очевидно.

Согласно определению

$$n!=1\cdot 1\cdot 2\cdot 3\cdot 4...n. \qquad (1.6)$$

Вас удивляет наличие “лишней” первой единицы! Так это же $0!$, о чем мы только что говорили. При таком определении, понятно, что $n!$ число целое положительное. При программной реализации может возникнуть естественное желание отнести его к типу `integer` или, например, `word`. Однако… Давайте посмотрим коротенькую табличку $0!=1$; $1!=1$; $2!=2$; $3!=6$; $4!=24$; $5!=120$; $6!=720$; $7!=5040$; $8!=40320$; $9!=362880$.

Известно, что в Турбо Паскале диапазон допустимых значений переменной типа `integer` – `(maxint+1)..maxint`, где `maxint=32767`. Соответственно, верхняя граница диапазона для чисел типа `word` будет `2*maxint+1=65535`.

Из таблицы следует, что начиная с $8!$ значения факториала выходят за пределы допустимого диапазона типа integer , а $9!$ - и из диапазона типа `word`. Можно, конечно, выбрать какой-то более мощный целочисленный тип, но в подавляющем числе практических расчетов столь высокая точность вычисления значения факториала (с большим числом значащих цифр – для больших значений n) – вовсе не требуется. В технических расчетах число с количеством значащих цифр выше 3-4 уже вызывает абсолютно обоснованные сомнения. Поэтому будет вполне логичным (целесообразным) рассматривать значение факториала как, например, число типа `real`, (верхняя граница допустимых значений где-то около 10^{38}), что более чем достаточно для технических и научных расчетов. В тех же случаях, где факториал должен в принципе быть целочисленным – его значение можно округлить до ближайшего целого. Кстати, если Вы в качестве аргумента функции `round` возьмете число, выходящее за допустимый диапазон, то компилятор совершенно резонно будет “возражать”.

С учетом вышесказанного, подпрограмму по вычислению $n!$ можно

представить так

```
function Fact(n:byte):real;
var s:integer; r:real;
begin
  r:=1;
  for s:=1 to n do r:=r*s;
  Fact:=r
end;
```

Здесь, как и функции `Xstn`, вводится промежуточная переменная `r`, чтобы избежать рекурсии, `r:=1` решает задачу вычисления $0!$. Цикл `for s:=1 to n ..` вычисляет факториал числа $n>0$. Если $n=0$, то тело цикла не выполняется ни разу, т.е. остается в силе `r:=1`. Последний оператор `Fact:=r` переносит вычисленное значение факториала в идентификатор подпрограммы-функции.

Итак, проблема формирования подпрограммы для вычисления факториала решается. Но, как будет показано ниже, чаще всего имеется возможность организовать вычислительный процесс таким образом, чтобы обойти определения факториалов и тем самым избежать целого ряда проблем.

1.4. Вычисление биномиальных коэффициентов

Биномиальный коэффициент

$$C_n^k = \binom{n}{k} = \frac{n!}{k!(n-k)!}. \qquad (1.7)$$

Если подпрограмма-функция `Fact(n:integer):real` имеется в наличии (доступна), то реализация формулы (1.7) “лежит на поверхности”.

```
function Cnk(n,k:byte):real;
begin
  Cnk:=Fact(n)/Fact(k)/ Fact(n-k)
end;
```

Уверяем читателя, что не забыли, что биномиальные коэффициенты – целые числа. Если это обстоятельство является принципиальным, то выше

приведенную подпрограмму можно легко “подкорректировать”, например, так

```
function Cnk(n,k:byte):integer;
begin
  Cnk:=round(Fact(n)/Fact(k)/ Fact(n-k))
end;
```

Если бы результат функции `Fact` был целочисленным, то можно было бы для осуществления операции деления воспользоваться операцией `div` (целочисленное деление) и тогда отпадала бы необходимость в использовании функции round. Короче, предмет для размышлений и вариантов.

Но, возможно, более радикальным будет представить формулу (1.7) следующим образом

$$C_n^k = \frac{n(n-1)(n-2)...(k+1)k!}{k!(n-k)!}.$$

После сокращения на $k!$ получаем

$$C_n^k = \frac{(k+1)(k+2)...(k+n-k)}{1\cdot 2...(n-k)}.$$

Для наглядности запишем последнюю формулу так

$$C_n^k = 1\cdot\frac{k+1}{1}\cdot\frac{k+2}{2}...\frac{k+(n-k)}{n-k} \qquad (1.8)$$

В последней формуле, слагаемое при k в каждом из числителей-сомножителей совпадает с его знаменателем, а эти знаменатели образуют отрезок натурального ряда от 1 до $n-k$ включительно. Это дает возможность вычислять C_n^k, например, так

```
function Cnk1(n,k:byte):word;
var s:byte; r:real;
begin
  r:=1;
  for s:=1 to n-k do
    r:=r*(k+s)/s;
  Cnk1:=round(r)
end;
```

Или, скажем, так

```
function Cnk2(n,k:byte):word;
var s,r:word;
begin
  r:=1;
  for s:=1 to n-k do
    r:=r*(k+s) div s;
  Cnk2:=r
end;
```

Вариации на эту увлекательную тему можно, конечно, продолжить. Но мы только хотели бы очередной раз напомнить читателю об абсолютной необходимости думать, анализировать, принимать взвешенные решения, учитывая все аспекты задачи.

1.5. Числа Фибоначчи

Последовательность 0, 1, 1, 2, 3, 5, 8, 13, 21, 34…, в которой каждое следующее число (начиная с третьего) равно сумме двух предшествующих, носит название чисел Фибоначчи. Эта последовательность используется в целом ряде алгоритмов, в частности, при поиске экстремумов функций. Итак, формально числа Фибоначчи можно записать так

$$F_s = F_{s-2} + F_{s-1},\ s = 2,3,4... \qquad (1.9)$$

где $F_0 = 0$, $F_1 = 1$.

Таким образом, мы принимаем, что нумерация (индексация) чисел Фибоначчи начинается с нуля. Сформируем подпрограмму, которая определяла бы F_n.

```
function Fib(n:byte):word;
var s,F,Fp,Fpp:word;
begin
  case n of
    0: Fib:=0;
    1: Fib:=1;
    else
      begin
        Fpp:=0; Fp:=1;
```

```
        for s:=2 to n do
          begin
            F:=Fpp+Fp; Fpp:=Fp; Fp:=F
          end;
        Fib:=F
      end
  end
end;
```

Здесь использован оператор выбора вариантов `case`. Если $n \geq 2$, то число Фибоначчи, предшествующее определяемому, обозначается как `Fp`, а число, предшествующее `Fp` соответственно как `Fpp`.

В теле цикла `for s:=2 to n do` … сразу же после вычисления `F` - очередного числа Фибоначчи, производится переприсваивание: `Fp` переводится в разряд `Fpp`, а только что полученное `F` в `Fp`, после чего можно выполнять очередной “виток” цикла.

Тип `word` позволяет вычислять числа Фибоначчи до F_{24} включительно. Для практических расчетов это более чем достаточно.

1.6. Приближенные вычисления сумм бесконечных рядов

В качестве элементарного примера алгоритма приближенного вычисления суммы бесконечного ряда рассмотрим ряд

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots + (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!} + \ldots \quad (1.10)$$

Конечно же в Турбо Паскале есть стандартная функция $\sin(x)$. Но тем лучше, будет с чем сравнивать разрабатываемый алгоритм.

Мы не будем ставить сейчас задачу рассмотрения всех аспектов задачи оценки суммы ряда (1.10). Просто примем к сведению известную из математики оценку: погрешность за счет ограничения бесконечной суммы n первыми членами ряда не превышает по модулю модуля $(n+1)$-го члена этого ряда (ряд (1.10) – сходится и является знакопеременным – именно к таким рядам применима приведенная оценка).

Итак, чтобы вычислить значение ряда (1.10) с погрешностью, не

превышающей некоторого наперед заданного числа ε, требуется учесть в сумме все слагаемые, превышающие по модулю величину ε. Предполагается, что модули слагаемых по мере роста их номеров равномерно уменьшаются. Для $x<1$ это очевидно, а вот для больших x? Синус, как известно, функция периодическая, и перед обращением к ряду (1.10) вероятно, целесообразно сузить диапазон значений x до разумного предела. Но это отдельная проблема, в которую мы сейчас не хотели бы углубляться. “Мы еще не волшебники – мы только учимся”, учащемуся же простительно начинать с попытки решения задачи “в лоб”. Ну, например, так

```
function sinx(x,Eps:real):real;
var Sum, Modul:real; Zn,s:integer;
begin
  Sum:=x; Zn:=-1; s:=1;
  repeat
    inc(s,2); Modul:=Xstn(s)/Fact(s);
    Sum:=Sum+Zn*Modul;
    Zn:=-Zn
  until Modul<=Eps;
  sinx:=Sum
end;
```

А теперь давайте посмотрим, что получилось. Тут в идентификаторе `Sum` накапливается значение суммы членов ряда (1.10). `Modul` - это модуль очередного члена ряда, который (член ряда) добавляется в накапливаемую сумму. С другой стороны, сравнение `Modul`’я с допустимой погрешностью `Eps` (идентификатор для ε) позволяет принимать решение о целесообразности продолжения процесса “накопления”. Тут, кстати, обращаем внимание читателя на одну деталь. При придумывании идентификатора для обозначения модуля может возникнуть идея сокращенно обозначить модуль как `Mod`. Так вот делать этого как раз нельзя. Почему? А потому, что слово `mod` (прописными или строчными буквами записанное – в Турбо Паскале - несущественно), входит в число так называемых зарезервированных слов Турбо Паскаля и в этом качестве обозначает операцию определения остатка от целочисленного деления. Чтобы

“отстроиться” от “запретного” слова mod мы и ввели идентификатор Modul. Идентификатор Zn предназначен для учета знака очередного члена ряда. Поскольку ряд (1.10) является знакопеременным, то в цикле repeat знак очередного слагаемого изменяется на противоположный на каждом “витке” (оператором Zn:=-Zn). Идентификатор s - это счетчик членов ряда.

Начинаем с того, что принимаем накапливаемую сумму равной первому члену ряда (Sum:=x). Очередной член ряда будет иметь знак минус (Zn:=-1). Здесь и далее речь будет идти о формальном знаке члена ряда, поскольку x ведь и сам может быть отрицательным (а почему бы и нет, если, скажем, диапазон x, к которому он сводится, содержит и $x<0$). Счетчик s “запускается” с (s:=1) – это значение s для слагаемого x (уже учтенного). В цикле repeat очередное значение s на каждом “витке” увеличивается на 2 (оператором inc(s,2)), т.е. s в нашем случае это фактически степень x и аргумент факториала, на который делится x^s. Тогда модуль очередного слагаемого вычисляется так: Modul:=Xstn(s)/Fact(s). Это слагаемое (со знаком Zn) вводится в накапливаемую сумму (Sum:=Sum+Zn*Modul), после чего знак (для очередного слагаемого, если в таковом возникнет необходимость) изменяется на противоположный (Zn:=-Zn). Модуль только что сформированного (и учтенного в сумме) слагаемого сравнивается с Eps. По результатам сравнения принимается решение о продолжении или прекращении накапливания суммы. Так, если Modul<=Eps, то, очевидно, что процесс можно останавливать, т.е. выходить из цикла. После выхода из цикла остается только сформированное значение суммы перенести в идентификатор sinx функции. Внимательный читатель, очевидно, заметил, что мы решили задачу “с перевыполнением плана”. Мы слагаемое, меньшее по модулю, чем ε, в сумму включили, тогда как согласно выше сформулированному критерию, мы этого не обязаны были делать. В данном случае такая “старательность” простительна (принцип – “кашу маслом не испортишь”), но в программировании такой подход (делать более того, чем требуется) не приветствуется. В очередном варианте программы постараемся

избавиться от этого недостатка. Ну, а раз мы уже решили переделывать подпрограмму, то заодно постараемся избавиться в ней от обращения к подпрограммам `Xstn` и `Fact`. Как? А вот как. Представим n-й член ряда (1.10) так

$$R_n = (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!}. \qquad (1.11)$$

Тут $n = 1, 2, 3 \ldots$.

Заменяем в (1.11) n на $(n-1)$, т.е. определяем предшествующий n-му член ряда

$$R_{n-1} = (-1)^{n-2} \frac{x^{2n-3}}{(2n-3)!}. \qquad (1.12)$$

А теперь поделим (1.11) на (1.12), и запишем результат деления так

$$\frac{R_n}{R_{n-1}} = (-1) \frac{x^2 (2n-3)!}{(2n-1)!},$$

или после упрощения

$$R_n = -R_{n-1} \frac{x^2}{(2n-1)(2n-2)}. \qquad (1.13)$$

Последнее соотношение положено в основу следующего варианта подпрограммы

```
function Sinx1(x,Eps:real):real;
var Sum,R,x2:real; n:integer;
begin
  R:=x; n:=2; x2:=x*x; Sum:=0;
  while abs(R)>Eps do
    begin
      Sum:=Sum+R; R:=-R*x2/(2*n-1)/(2*n-2);
      inc(n)
    end;
  Sinx1:=Sum
end;
```

В данном варианте мы вместо цикла типа `repeat` (в котором при первом “проходе” тело цикла выполняется всегда) используем цикл типа `while` (в

котором тело цикла может не выполняться и ни разу). В нашем случае если первое слагаемое (первый член ряда (1.10)) не превышает по модулю величины `Eps`, то и вносить его в сумму нет надобности. И сумма, первоначально принятая равной нулю (`Sum:=0`) так и останется нулем. Если же `abs(R)>Eps`, то слагаемое `R` вносится в сумму (`Sum:=Sum+R`), после чего заготавливается очередное слагаемое по формуле (1.13), заготавливается (на всякий случай) очередное `n` (`inc(n)`), и цикл повторяется очередной раз. И так до тех пор пока очередное слагаемое не окажется не заслуживающим внимания – выход из цикла. Прием организации вычисления членов ряда с использованием рекуррентных соотношений типа (1.13), когда задается первый член ряда, а все последующие вычисляются путем модифицирования предыдущего, является весьма эффективным. И мы настоятельно рекомендуем этот прием в подобных ситуациях.

1.7. Операции над комплексными числами

Пусть имеем два комплексных числа

$$x_1 = R_1 + jI_1,$$

$$x_2 = R_2 + jI_2,$$

где R_1, R_2 - их вещественные, а I_1 и I_2 - мнимые компоненты (составляющие).

Тут $j = \sqrt{-1}$ или, что то же $j^2 = -1$. Таким образом, j - мнимая единица.

Рассмотрим простейшие арифметические операции над комплексными числами

$$\begin{cases} x_1 + x_2 = (R_1 + R_2) + j(I_1 + I_2); \\ x_1 - x_2 = (R_1 - R_2) + j(I_1 - I_2); \\ x_1 \cdot x_2 = (R_1R_2 - I_1I_2) + j(I_1R_2 + R_1I_2); \\ \dfrac{x_1}{x_2} = \dfrac{(R_1 + jI_1)(R_2 - jI_2)}{(R_2 + jI_2)(R_2 - jI_2)} = \dfrac{R_1R_2 + I_1I_2}{R_2^2 + I_2^2} + j\dfrac{I_1R_2 - R_1I_2}{R_2^2 + I_2^2}. \end{cases} \quad (1.14)$$

Последнюю операцию мы рассмотрели более детально чем предыдущие. Суть ее в том, чтобы избавиться от комплексности в знаменателе дроби

$(R_1 + jI_1)/(R_2 + jI_2)$. Это достигается путем умножения числителя и знаменателя на число, сопряженное знаменателю.

Приведенный набор операций над двумя комплексными числами можно “перекрыть” с помощью подпрограммы типа

```
procedure OpComp(R1,I1,R2,I2:real; OP:char; var R,I:real);
var Zn:real;
begin
  case OP of
    '+': begin R:=R1+R2; I:=I1+I2 end;
    '-': begin R:=R1-R2; I:=I1-I2 end;
    '*': begin
           R:=R1*R2-I1*I2;
           I:=I1*R2+R1+I2
         end;
    '/': begin
           Zn:=sqr(R2)+ sqr(I2);
           R:=(R1*R2+I1*I2)/Zn;
           I:=(I1*R2-R1*I2)/Zn
         end
  end
end;
```

Возведение комплексного числа в целую степень n можно реализовать по схеме

$$(R_1 + jI_1)^n = 1 \cdot (R_1 + jI_1)(R_1 + jI_1)...(R_1 + jI_1). \qquad (1.15)$$

Количество скобок в (1.15) равно n. Если же $n < 0$, то очевидно, что

$$(R_1 + jI_1)^{-abs(n)} = \frac{1}{(R_1 + jI_1)^{abs(n)}} = \frac{1}{R + jI} = \frac{R}{R^2 + I^2} - j\frac{I}{R^2 + I^2}.$$

Итак,

```
Procedure Compstn(R1,I1:real; n:integer; var R,I;real);
var s:integer; Zn:real;
begin
  R:=1; I:=0;
  for s:=1 to abs(n) do
    begin
      Zn:=R*R1-I*I1;
      I:=R*I1+I*R1;
```

```
      R:=Zn
    end;
  if n<0 then
    begin
      Zn:=sqr(R)+sqr(I);
      R:=R/Zn; I:=-I/Zn
    end
end;
```

Первый множитель 1 в правой части (1.15) мы представляем как $R := 1$; $I := 0$ (вещественное число 1 имеет нулевую мнимую часть). В процессе умножения $(R + jI)$ – текущего значения результата на $(R_1 + jI_1)$ вещественную часть произведения записываем в промежуточную переменную `Zn:real`. После того, как вычислена мнимая часть (`I:=R*I1+I*R1`) значения соответствующей вещественной части, хранившееся в `Zn`, переносится в переменную R (`R:=Zn`). Если $n > 0$, выполнением цикла по `s` процесс вычисления $(R1 + jI1)^n$ завершается, как это и следует из формулы (1.15).

Если же $n < 0$, то дополнительно осуществляются вычисления, показанные в формуле (1.16).

Извлечение корня n-ой степени из комплексного числа приводит к n ответам, которые удобно представлять в виде массивов. Мы вернемся к этому вопросу, после того, как ознакомимся с массивами.

2. Операции с одномерными числовыми массивами

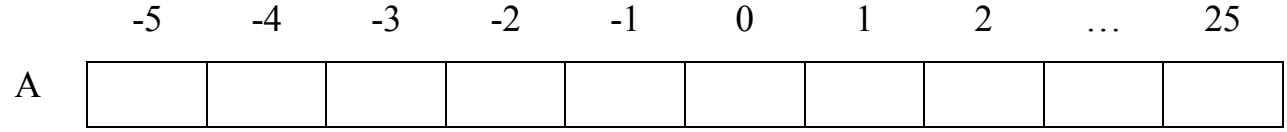
Очень широкий круг задач численного анализа предполагает оперирование с таблицами данных, в частности, чисел. Это и результаты наблюдений, начиная от метеорологических, статистических, финансовых и кончая представлением результатов решения дифференциальных уравнений и их систем, работа с матрицами. Этот список можно продолжать. В Турбо Паскале таблицы однородных (однотипных) данных оформляются в виде массивов. Начнем с так называемых одномерных массивов.

2.1. Формирование типов одномерных массивов

Одномерный массив (массив-строка) в Турбо Паскале описывается так. Если речь идет об описании отдельной переменной или группы однотипных переменных, то такое описание имеет вид

```
var A,B:array[-5..25] of real.
```

Здесь A, B – массивы, тип элементов которых указывается после двоеточия. Служебное (зарезервированное) слово `array` означает массив. В квадратных скобках после слова array указывается диапазон индексов (номеров ячеек) данного типа – две последовательных точки (символ диапазона) отделяют начальное (слева от двух точек) от конечного значения индексов. Естественно, что края диапазона должны быть указаны в виде целых чисел. Индексы (и соответственно их крайние значения) могут быть как положительными, так и отрицательными. Важно лишь, чтобы начальное значение было меньше конечного. Ячейки нумеруются с шагом единица и, естественно, нумерация идет по нарастающей (“слева – направо”). Для наглядности одномерный массив можно представлять себе в виде длинного ящика, разделенного поперечными перегородками на ячейки, соответствующим образом пронумерованные.



Тут показана схема выше описанного массива. В каждой из ячеек согласно описанию размещается (записывается, хранится) число типа `real`. Обращение к отдельному элементу массива осуществляется так `A[-3], A[0], …, A[25].` Например, запись `A[-2]:=5` означает, что в ячейку с номером -2 записывается число 5. Запись `A[0]:=7+A[12]` означает, в ячейку массива A с номером ноль записывается сумма числа 7 и содержимого ячейки того же массива с номером 12.

Описание массивов `A` и `B`, приведенное в начале параграфа, удобно при создании нестандартных массивов (условно говоря, массивов разового пользования). Это имеет смысл с точки зрения экономии машинной памяти, чтобы, не обращаться к более мощным (более емким) “стандартным” массивам. Слово стандартным взято в кавычки потому, что в Турбо Паскале стандартных массивов не предусмотрено (разработчики языка не сочли целесообразным их вводить, отдавая принятие решений по этому поводу пользователям языка). Вместе с тем “стандартизовать” массивы необходимо. Дело в том, что при создании подпрограмм удобно передавать им в качестве параметров различные массивы и получать в качестве результата новые или соответствующим образом трансформированные массивы тех или иных типов. Но для этого передаваемые и возвращаемые массивы должны быть типизированы. Описание типа одномерного массива может выглядеть, например, так

```
        type Coef=array[-1..31] of real
```

или `Coefr=array[-1..201] of real`

или же `Coefl=array[-1..601] of real.`

Первый из типов (тип `Coef`) определяет массив чисел типа real с нумерацией ячеек от -1 до 31. Два последующих типа отличаются от первого только конечными значениями индексов. Указанные три типа рассматриваются как “стандартные” на кафедре автоматизации химических производств НТУУ (КПИ). Разработанное на кафедре программное и методическое обеспечение ориентировано на использование этих типов, а сами эти типы сформированы из соображений удобства их использования при решении вычислительных задач

чаще всего встречающихся при исследовании автоматических систем управления.

Так, тип `Coef` предназначен в первую очередь для размещения коэффициентов (откуда и его название) полиномов.

При расчете передаточных функций (имеются в виду в первую очередь дробно-рациональные передаточные функции, т.е. функции в виде отношения полиномов) полиномы приходится складывать, вычитать, перемножать, делить, возводить в степень и т.д. – в результате чего степени полиномов могут принимать самые различные значения, от нуля до 30. Почему до 30? Ну, на чем-то же нужно было остановиться. Вот мы и подумали, что 30 хватит. У вас другое мнение? Поменяйте верхний предел по своему усмотрению. Мы принципиально возражать не станем. Ну, а пока пусть будет так. Мы пока не объяснили назначение ячеек с номерами -1 и 31. Договоримся записывать в ячейку с номером -1 старший номер задействованной ячейки. В случае полинома это будет его степень n. Дело в том, что в массиве типа Coef может храниться информация о полиноме любой степени (как мы уже отмечали от 0 до 30). А какую степень имеет конкретный полином, хранящийся в массиве в данный момент? Вот в ячейке с номером -1 эта степень и хранится. Это своего рода бирочка на ящике, на которой обозначено количество “предметов” (чисел) “упакованных” в ящике. Мы ведь не брали на себя обязательства загружать ящик “под завязку”. Тип Coef это своего рода стандартная тара для упаковки различных “товаров”. Итак, если имеется полином вида

$$A(p) = a_0 + a_1 p + a_2 p^2 + ... + a_n p^n, \qquad (2.1)$$

то его коэффициенты в массиве (назовем его массивом A) типа Coef расположатся таким образом

	-1	0	1	2	3	4	…	n	…	31
A	n	a_0	a_1	a_2	a_3	a_4	…	a_n	…	

Форма (2.1) записи полинома не является единственно возможной. Во

многих книгах полином записывается так

$$a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_n . \qquad (2.2)$$

С нашей точки зрения форма (2.1) предпочтительнее, в первую очередь, с точки зрения удобства программирования. В этой форме записи индекс коэффициента, степень переменной при этом коэффициенте (кто при ком - несущественно), а также номер ячейки массива, где этот коэффициент хранится – это одно и то же число.

Мы в дальнейшем будем пользоваться именно формой записи (2.1). Теперь об обозначении переменной. В форме (2.1) – это p. Почему? Математики обычно обозначают независимую переменную буквой x. Дело, конечно, не в названии – “называй хоть горшком, только в печь не сади”. Буква p выбрана потому, что в теории автоматического управления (как она излагается на кафедре АХП) параметр передаточной функции принято обозначать как p. То есть запись дробно-рациональной передаточной функции выглядит так

$$W(p) = \frac{b_0 + b_1 p + b_2 p^2 + \ldots + b_m p^m}{a_0 + a_1 p + a_2 p^2 + \ldots + a_n p^n} .$$

Полиномы, фигурирующие здесь, - это полиномы от p. И именно с подобными полиномами нам чаще всего придется иметь дело в дальнейшем, отсюда и p в (2.1). Символ p в $W(p)$ - это, по сути, символ комплексной переменной в формуле преобразования Лапласа

$$L[f(t)] = \int_0^\infty f(t) e^{-pt} dt . \qquad (2.3)$$

Дотошный читатель найдет источники, в которых вместо t используется x, а вместо p - буква s. Но на чем-то ведь нужно остановиться!

Запись n в A[-1] может вызвать возражение типа: массив то типа `real`, а n - целочисленное. Таким образом, n - чужеродное (“чужетипное”) в этом массиве, ведь массив, по определению, таблица однотипных элементов. Упрек – справедливый. Но мы руководствовались соображениями “революционной целесообразности”. Это нарушение, но, скажем так, уголовно ненаказуемое.

Против записи целого числа в ячейку типа `real` Турбо Паскаль (его компилятор) не возражает, т.е. при записи n в A[-1] проблем не возникает. А вот при извлечении этого n, чтобы не было проблем можно поступить, например, так

```
n:=round(A[-1]).
```

В итоге - "чужеродность" n в массиве (n, по сути, не является элементом массива), окупается соображениями удобства (не требуется сопровождать массив идентификатором, несущим информацию о фактической длине или, применительно к полиному, о его степени). Судить о степени полинома, анализируя содержание ячеек массива – нет возможности. Записывать в `A[n+1]` ноль? А чем нулевое значение коэффициента хуже всех прочих? Нет формального признака, по которому число типа `real` не может (не имеет права!) рассматриваться как потенциальный коэффициент полинома. Поэтому предложенный нами вариант (хотя и несколько искусственный) позволяет решить задачу однозначно и, согласитесь, несложно. К слову, в Турбо Паскале не предусматривается "чистка" памяти перед началом прогона программы. Поэтому, если какая-то переменная описана, но не "записана" (в нее не записано конкретное значение), то в ней может оказаться все что угодно ("мусор"), поэтому когда мы записываем n в `A[-1]`, то мы тем самым утверждаем, что за содержимое ячеек с номерами от 0 до n мы отвечаем, а за остальных – нет. Если мы туда ничего не записывали, то незачем и пытаться оттуда что-то считывать. Записывать в "лишние" ячейки нули – зачем? А вот то, что находится в ячейках от 0-й до n-ой – за это несем ответственность мы – с вытекающими отсюда последствиями. Мы пока ничего не скажем о наших видах на ячейку с номером 31. Если хотите, можете считать, что это на случай, если n окажется равным 31. Возможны и иные ситуации, но о них лучше будет говорить по мере их возникновения.

Массивы типов `Coefr` (последняя буква `r` от слов: рабочий, расширенный) используются при решении задач, где рамки типа `Coef` оказываются тесными, например, при обработке массивов экспериментальных данных соответствующей

длины, при решении систем дифференциальных уравнений (особенно в частных производных) и в других случаях.

Массивы типа `CoefL` (буква `L` от слова `long` - длинный) будут нами использоваться при построении графиков всевозможный функций. В массиве такого типа удобно размещать таблицу значений функции, анализировать эту таблицу на предмет формирования под нее подходящей системы координат, с тем, чтобы затем визуализировать эту таблицу на экране компьютера в виде качественного (построенного с наименьшим возможным шагом по оси аргумента) графика.

2.2. Анализ одномерного массива на наличие в нем элементов с заданными свойствами

Простейшей задачей, которая может возникнуть при анализе информации, сосредоточенной в массиве, является просмотр массива с целью выяснить имеются ли в нем элементы (или группы элементов), обладающие некоторыми представляющими интерес для исследователя свойствами. Ну, например, предположим, анализируемый массив `A:Coef` содержит значения температур в январе 2005 года в г. Киеве, скажем, на 12 часов каждого дня месяца. В январе 31 день – чем не оправдание для цифры 31 в описании типа `Coef`? Правда, нет дня с номером 0. Ну, что же, ячейка с номером ноль в таком случае останется незадействованной. Важно только будет не забыть об этом обстоятельстве при анализе. А если бы речь шла не о январе, а о феврале (где число дней бывает 28 или 29) или, например, об апреле (с его 30 днями), то наличие ячейки с номером -1 становится вполне оправданным.

При анализе температур вопрос может состоять в том, например, были ли в данном месяце дни с температурами ниже нуля (мороз) или, скажем, таковых не было. Применительно к полиномам выяснение вопроса есть ли среди его коэффициентов нулевые или отрицательные может быть важным при оценке устойчивости системы, для которой данный полином является характеристическим(необходимым условием устойчивости системы является строгая положительность всех коэффициентов ее характеристического

уравнения). Поэтому, для конкретности, рассмотрим именно такую задачу. Есть массив A:Coef с информацией о полиноме. Требуется выяснить, есть ли среди коэффициентов данного полинома хотя бы один коэффициент меньший или равный нулю. Возможен ряд алгоритмов решения этой задачи. Начнем с такого. Посчитаем сколько в данном полиноме коэффициентов меньших или равных нулю. Если окажется, что их количество равно нулю, значит таковых нет ни одного, если больше нуля – то, значит, есть.

```
function NegPol(A:Coef):Boolean;
var s,n,m:integer;
begin
  n:=round(A[-1]); m:=0;
  for s:=0 no n do
    if A[s]<=0 then inc(m);
  if m=0 then NegPol:=false
         else NegPol:=true
end;
```

Имя функции не будем комментировать. А вот тип функции (boolean) договоримся интерпретировать так: если среди коэффициентов будет обнаружен хотя бы один коэффициент меньше или равный нулю, то функция NegPol возвращают значение true (есть - правда), в противном случае false (нет - ложь). s - счетчик элементов массива, n - степень полинома, m - число обнаруженных коэффициентов, удовлетворяющих требованию A[s]<=0.

Оператор n:=round(A[-1]) уже ранее комментировался. Оператор m:=0 фиксирует то обстоятельство, что пока мы не начнем просмотр массива, то ни одного числа меньшего нуля или равного нулю мы, естественно, видеть не будем. В цикле мы перебираем всех “подозреваемых” и как только обнаруживаем “виновного” увеличиваем число “уличенных” на единицу (inc(m)).

И, наконец, последний условный оператор “докладывает наверх” о результатах “дознания”, в результате идентификатор подпрограммы функции приобретает требуемое значение.

В приведенном варианте подпрограммы мы, собственно делали больше чем

от нас требовалось. Считать количество “дефектных” коэффициентов нам “не задавали”. Это – самодеятельность, в программировании энтузиазм не поощряется. Виноваты – исправимся.

```
function NegPol1(A:Coef):Boolean;
var s:integer;
begin
  NegPol1:=false;
  for s:=0 to round(A[-1]) do
    if A[s]<=0 then NegPol1:=true
end;
```

Здесь, пока мы не начали “расследование” ответ отрицателен (`NegPol1:=false`). Если же на каком-то шаге (при “допросе” `s`-го подозреваемого) выяснится его виновность, то ответ подпрограммы станет утвердительным (`NegPol1:=true`).

Здесь мы еще “сэкономили” на переменной n, воспользовавшись возможностью задавать начальное и конечное (как в нашем случае) значение параметра цикла s в виде выражений. `round(A[-1])` - это выражение типа `integer`, разрешенное в качестве конечного значения параметра `S:integer`.

Но и к `NegPol1` можно “придраться”. Какой смысл обязательно просматривать весь массив (от 0 до n), если может оказаться, что неположительный коэффициент находится где-то в начале “списка”. Логично обнаружив первый попавшийся элемент с разыскиваемыми свойствами прекратить дальнейший поиск, поскольку он становится излишним (ответ `NegPol1:=true`) очевиден. Итак

```
function NegPol2(A:Coef):Boolean;
var s,n:integer; F:Boolean;
begin
  n:=round(A[-1]); s:=0; F:=false;
  repeat
    if A[s]<=0 then F:=true;
    inc(s)
  until F or (s>n);
  NegPol2:=F
```

```
end;
```

В данном варианте использован цикл типа repeat (мы ведь заранее не знаем сколько элементов массива придется перебрать, но один-то, по крайней мере с номером 0 (`s:=0`) посмотреть придется). `F` - вспомогательная переменная (“флажок”). Пока мы не начали просмотр флажок указывает на то, что требуемого не найдено (`F:=false`). Как только искомое будет обнаружено, флажок устанавливается в положение `F:=true`. Состояние флажка используется как одно из показаний к выходу из цикла (в составе логического выражения после слова `until`). Может читателю было бы понятнее если бы мы записали `until (F=true)`… Но для компилятора (и, надеемся, после этого для читателя) понятно и более лаконичное выражение until `F`… Ну, и понятно второе показание к выходу из цикла – это исчерпание списка “подозреваемых субъектов” (`s>n`). Здесь `s` - следующий (после только что рассмотренного) номер ячейки. Следующим его сделал оператор `inc(s)`.

При желании можно было бы обойтись без промежуточных переменных `n` и `F`. Но стоит ли? Экономии - чуть, а так понятнее… Но это, конечно же, субъективно.

2.3. Поиск минимального и (или) максимального элемента массива

Мы ранее уже отмечали, что массивы типа `CoefL` часто используются как носители таблиц значений ординат функции, предназначенных для формирования их графиков. Чтобы график “смотрелся”, необходимо чтобы система координат под него была адекватной ему (кому нужен график, который в предложенной системе координат находится за пределами экрана или выглядит жалкой закорючкой, отрезком прямой или даже точкой на экране?). С диапазоном значений аргумента – проще. Раз мы сформировали массив ординат, то, очевидно, с диапазоном значений x мы определились, итак `Xmin, Xmax` считаем известными. А вот `Ymin, Ymax` – их еще необходимо определить, просмотрев данный массив. Этим мы сейчас и займемся.

```
Procedure YminYmax(A:coefL; var Ymin,Ymax:real);
var s:integer;
begin
  Ymin:=1e25; Ymax:=-1e25;
  for s:=0 to round(A[-1]) do
    begin
      if A[s]<Ymin then Ymin:=A[s];
      if A[s]>Ymax then Ymax:=A[s]
    end
end;
```

У читателя, естественно, должен возникнуть вопрос: что за магические числа `1e25` фигурируют в первых двух операторах присваивания? Авторы просто надеялись, что искомый минимум не больше `1e25` (т.е. 10^{25}), тогда как искомый максимум уж, по крайней мере, не меньше чем `-1e25`. Ну, не каждый день выпадает радость строить графики функций, выходящих за указанные (весьма просторные!) пределы. Но это, конечно, не лучшее решение. Может, лучше было бы так?

```
Procedure YminYmax1(A:coefL; var Ymin,Ymax:real);
var s:integer; r:real;
begin
  Ymin:=A[0]; Ymax:=A[0];
  for s:=1 to round(A[-1]) do
    begin
      r:=A[s];
      if r<Ymin then Ymin:=r;
      if r>Ymax then Ymax:=r
    end
end;
```

Тут мы начали с того, что предположили, что одним из возможных претендентов на титул “наибольшего”, равно как и “наименьшего” является “крайний слева” или, вообще-то говоря, любой из претендентов (любой элемент массива – от элемента с номером 0 до элемента с номером `round(A[-1])`) может оказаться либо наименьшим либо наибольшим). Но на чем-то нужно остановиться, почему бы не на “крайнем слева”? Далее, в цикле перебор

оставшихся претендентов. Поскольку элемент с номером 0 уже учтен, то за начальное значение s принято (s:=1). Ну, и чтобы упростить последующие сравнения и присваивания мы ввели промежуточную переменную r:real.

Обратите внимание на слово var, стоящее в заголовке процедуры перед параметрами Ymin, Ymax. Слово var как раз и обязывает процедуру возвратить значение Ymin, Ymax в вызывающий ее блок. Не будь этого var значения Ymin, Ymax были бы определены процедурой, но они немедленно

“умерли бы” после завершения процедурой своей работы – и работа пошла бы насмарку. И, напоследок, одно небольшое соображение. Если система координат под формирование графика зависимости, представленной массивом F:CoefL, будет определяться значениями Ymin, Ymax, возвращенными процедурой YminYmax1, то не исключено, что горизонтальная ось системы может оказаться за пределами экрана (выше или ниже его). Иногда это оправдано, иногда – не желательно. Если мы все таки настаиваем на присутствии оси x на экране, то следует позаботиться о включении значения $y = 0$ (на оси) в число претендентов на Ymin, Ymax. Это можно обеспечить задавая первыми операторами Ymin:=0; Ymax:=0.

Естественно, что этим приемом мы можем исказить либо истинное значение Ymin, либо Ymax. Но, что вам дороже? Решайте сами…

И, наконец, еще одно соображение. В приведенных подпрограммах мы определяем значения Ymin и Ymax, не интересуясь (не запоминая) их “адреса”, т.е. где мы их обнаружили. Возможно, это и не существенно. Но возможны случаи, когда необходимо знать именно адреса, а уж сами то значения Ymin или там Ymax при наличии соответствующего адреса определить – не проблема. Давайте для определенности установим адрес, например, минимума в массиве A:Coefr.

```
function Adrmin(A:Coefr):integer;
var s,n,Am:integer;
begin
  n:=round(A[-1]); Am:=0;
```

```
  for s:=1 to n do
    if A[s]<A[Am] then Am:=s;
  Adrmin:=Am
end;
```

Здесь Am - текущий адрес наименьшего элемента массива (в процессе поиска). Начинаем с предположения, что искомым элементом (наименьшим), является A[0] с адресом (индексом) Am:=0. В процессе перебора в случае обнаружения более “достойного” претендента, адрес последнего запоминается в Am (вместо находившегося в нем перед этим адреса предшествующего претендента “не оправдавшего, как оказалось, доверия”). Хотя и последний (на текущем s-ом шаге поиска) может также подлежать замене… Итак, алгоритм, реализованный в Adrmin, после просмотра всего массива (после завершения цикла for s:=1 to n…) выдаст (с последующим переносом в Adrmin) адрес самого “достойного” (наименьшего) элемента массива A. Если в массиве A окажется несколько одинаковых элементов, претендующих на титул наименьшего, тогда, если это существенно, можно возвращать либо адрес первого (или соответственно) последнего из этих равновеликих претендентов. В Adrmin это будет первый встреченный (первый слева, поскольку мы сканировали массив идя слева направо) истинный минимум, а вот если бы мы вместо проверки условия A[s]<A[Am] да проверяли бы условие A[s]<=A[Am], то Adrmin возвратила бы адрес последнего (среди равных) истинного минимума.

Чтобы переориентировать подпрограмму типа Adrmin на поиск адреса максимума (вместо минимума), очевидно, достаточно в проверяемом условии заменить знак < на >. Ну, возможно, еще изменить заголовок (имя) функции, скажем на Adrmax, что ли. Хотя это уже на усмотрение разработчика подпрограммы.

2.4. Сдвиг элементов массива

При анализе, сортировке, удалении одних и (или) при введении в массив других элементов возникает необходимость сдвигать весь массив или какую-то его часть в одну, или другую сторону. Будем это называть сдвигом влево, или, соответственно, вправо. Начнем со сдвигов на одну позицию. Рассмотрим несколько примеров (задач).

Задача №1. В массиве `A:Coef` осуществить сдвиг массива влево на одну позицию, после чего в n-ую (освободившуюся) ячейку записать число `b:real`.

```
Procedure Shiftl(var A:Coef; b:real);
var s,n:integer;
begin
  n:=round(A[-1]);
  for s:=0 to n-1 do
    A[s]:=A[s+1];
  A[n]:=b
end;
```

Надеемся, понятно, что бывшее `A[0]` мы при этом теряем. Но если Вам его не жалко, то так тому и быть. Если бы оно было Вам дорого (“как память”, например), то его можно было бы вернуть, добавив в список параметров, скажем, такую конструкцию: `var A0:real`. И где-то в самом начале операторной части процедуры нужно было бы записать в A0 значение еще “живого” `A[0]` оператором присваивания `A0:=A[0]`.

Обращаем внимание читателя на цикл. Его переменная `s` принимает последовательно значения от `0` до `n-1`. Что такое `s`? Счетчик, переменная цикла – это правда, но не вся правда. Что за ним стоит, что оно (`s`) обозначает? Это обязательно необходимо уточнить (уяснить!) чтобы правильно задать пределы его изменения и корректно сформировать тело цикла. Так вот в нашем случае s - это номер ячейки в массиве A, в которую помещается (записывается) сдвигаемый элемент массива. А поскольку сдвиг осуществляется влево, то в s-ую ячейку записывается ее сосед справа, т.е. `(s+1)`−ый элемент. Это и отражено в теле цикла как `A[s]:=A[s+1]`. Первым в ячейку с номером 0 (`for s:=0` …)

записывается A[0+1] т.е. A[1], а последним в ячейку с номером s=n-1 переносится A[n]. Именно поэтому переменная *s* и изменяется от 0 до n-1. Перебор сдвигаемых элементов в указанной последовательности (по мере увеличения индекса) представляется естественным. Но это при сдвиге влево (именно такой сдвиг мы сейчас и реализовали). А если бы нужно было сдвигать вправо? Вернемся еще к этому вопросу. А пока что для завершения процесса в освободившуюся (n -ую) ячейку (содержимое которой перекочевало в ячейку с номером n-1 на последнем "витке" цикла) как и обещали (как было оговорено в постановке задачи) заносится число b (A[n]:=b).

На этом решение задачи заканчивается. Стоит только обратить внимание читателя на слово var в начале списка параметров в заголовке подпрограммы. Наличие этого слова показывает, что подпрограмма "обязана" вернуть предоставленный ей массив A:Coef (после выполнения предписанных операций над ним). Это удобная конструкция. Вместо того, чтобы передавать один массив (исходный), а возвращать другой (переделанный) – можно обойтись одним массивом как в нашем случае. Естественно, с массивом, который необходимо вернуть, требуется обращаться аккуратно, чтобы "не запороть" его, но это само собой.

Ну, а теперь, как и обещали, возвращаемся к рассмотрению варианта сдвига вправо.

Задача №2. Что если мы этот сдвиг (вправо) попытаемся организовать, например, так

```
for s:=0 to n-1 do
  A[s+1]:=A[s]
```

На первом шаге ("витке" цикла) содержимое A[0] перекочует в A[1]. На втором витке - A[1] (по сути то же A[0]) перейдет в A[2], затем A[2] (по сути то же A[0]) будет записано в A[3] и т.д. "Эффект катка". После этого так называемого сдвига у нас во всех ячейках массива окажется продублированным A[0]. Это, конечно, интересно, но мы ведь не это собирались делать! Ну, а

теперь как это можно было бы выполнить корректно.

```
Procedure Shiftr(var A:Coef; b:real);
var s,n:integer;
begin
  n:=round(A[-1]);
  for s:=n downto 1 do
    A[s]:=A[s-1];
  A[0]:=b
end;
```

Здесь `s` - номер ячейки, куда заносится передвигаемый элемент массива ((`s-1`)-ый).

Задача №3. Удалить из массива `A:Coef` содержимое k-ой ячейки (подразумевается $0 \le k \le n$). Термин “удалить” будем понимать как команду “выйти из строя (шеренги)” с последующим (по умолчанию – автоматическим) “смыканием строя”. Смыкание (снова же по умолчанию) предполагается в сторону головы колонны (“подтянись!”). В случае одномерного массива его “голова” там, где находится ячейка с минимальным номером (в типе Coef “головной” является A[0]).

```
Procedure UdElM(var A:Coef; k:integer);
var s,n:integer;
begin
  n:=round(A[-1]);
  for s:=k to n-1 do
    A[s]:=A[s+1];
  A[-1]:=n-1
end;
```

Тут мы в целях “экономии”, получаем массив A, его же и возвращаем (слово var перед `A:Coef` в списке параметров). `k` - номер ячейки, содержимое которой подлежит удалению (ликвидации). В цикле `s` - это номер ячейки куда записывается содержимое сдвигаемой (т.е. (`s+1`)-ой) ячейки. Сдвиг осуществляется влево. Как уже ранее отмечалось при сдвиге нужно “тянуть”, а не

“толкать” во избежание “эффекта катка” или “утюга”, после которых оставляется ровная поверхность. Поэтому первой заполняется (обновляется) k-ая ячейка, последней (n-1)-ая. Массив при этой операции – “теряет бойца”, соответственно, это фиксируется в ячейке с номером -1.

Задача №4. Вставить между k -ой и (k+1) ячейками массива A:=Coef число b:real.

```
Procedure VstElM(var A:Coef; k:integer; b:real);
var s,n:integer;
begin
  n:=round(A[-1]);
  for s:=n+1 downto k+1 do
    A[s]:=A[s-1];
  A[k]:=b; A[-1]:=n+1
end;
```

Комментарии, как говорится, излишни.

2.5. Сортировка одномерного массива

Под термином “сортировка” будем понимать упорядочение массива с целью расположить его элементы либо в порядке возрастания (точнее неубывания) либо убывания (невозрастания). Уточнение в скобках сделано на тот случай, если в массиве окажутся повторяющиеся элементы. Понятно, что после упорядочения одинаковые элементы должны располагаться подряд, а среди равных понятие возрастания (убывания) теряют смысл. При упорядочивании информация о прежних “постах” (номерах) теряется, если не оговорено обратное. Так вот, мы это оговаривать не будем (для упрощения задачи). Снова же (по умолчанию – для простоты) будем подразумевать, что упорядочивание осуществляется по неубыванию.

Существует множество алгоритмов сортировки. Ограничимся рассмотрением двух.

Первый. Идя вдоль массива (слева направо) начиная с номера s:=0 и кончая s:=n-1 определяем для каждого (s-го) номера ячейки “наиболее

достойного претендента”. Таковым будет, очевидно, наименьший элемент массива среди элементов расположенных правее s-го (предполагается, что элементы левее s-го уже упорядочены). После того, как адрес претендента окончательно установлен (скажем, это k) k -ый и s-ый элемент меняются местами. И только после этого переходим к следующему s, чем гарантируется упорядоченность “левее s”. Почему процесс упорядочивания заканчивается на s:=n-1? Да если (s:=n-1) элементов упорядочены, то n-ому куда деваться? В процессе перестановок, когда “вправо” сбрасывались “некондиционные” элементы, самый некондиционный (худший среди наименьших, т.е. наибольший) не мог не оказаться ни в какой ячейке кроме n-ой. А даже если бы мы попытались найти для n-ой ячейки более достойного претендента, то среди кого его искать? Среди единственного n-го? Но если бы мы по небрежности не обратили внимание на высказанные соображения и все-таки взяли бы за конечное значение n, то не исключено, что (n+1)-ый элемент занял бы место n-го. Но ведь в (n+1)-ой ячейке – мусор!

```
Procedure SortMas(var A:Coefr);
var z,s,n,k:integer; r:real;
begin
  n:=round(A[-1]);
  for s:=0 to n-1 do
    begin
      k:=s;
      for z:=s+1 to n do
        if A[z]<A[k] then k:=z;
      if k>s then
        begin
          r:=A[k]; A[k]:=A[s];
          A[s]:=r
        end
    end
end;
```

Здесь цикл по s перебирает номера ячеек, куда надо подобрать и занести наиболее достойного претендента. Первым из возможных претендентов на место в s-ой ячейке рассматривается он сам (k:=s). Альтернативные претенденты

“скрываются” где-то в диапазоне номеров от (s+1)-го до n-го. При обнаружении “перспективного” (более достойного), чем “ныне действующий” - k-ый, значение k (адрес) уточняется (k:=z). После того, как окончательный адрес кандидата, выигравшего конкурс, установлен (после завершения цикла по z), нужно осуществить “ротацию”. Но устраивать церемонию имеет смысл только в том случае, если победитель новый (if k>s), иначе – зачем?

Составной оператор в теле условного оператора меняет местами k- ый и s-ый элементы массива с использованием “третьего стаканчика” r:real. Вначале в r переносится содержимое одного из двух меняющихся местами элементов, в нашем случае k-го. На освободившееся место переносится содержимое другого элемента (A[k]:=A[s]) и, наконец, в свободную (уже свободную) s-ую ячейку “переливается” содержимое “третьего стаканчика” (A[s]:=r). Рекомендуем читателю взять на заметку идею “третьего стаканчика”. Авось пригодится.

Только что рассмотренный алгоритм сортировки уязвим для критики. Нет, правильность результата не ставится под сомнение, но вот трудоемкость, громоздкость… Ну, например, представим себе ситуацию, что на сортировку передан уже отсортированный массив. Каждому понятно что это – “подарок судьбы”. И как же в данном алгоритме “оприходуется” привалившая удача? А никак! Алгоритм упорно будет перебирать ячейки с 0-ой по (n−1)-ую. Для каждой из них он методично будет искать наиболее достойного претендента, будет обнаруживать, что наиболее достойным является сам этот элемент. Слава богу (мы об этом благоразумно позаботились) не будет устраивать перестановку s-го элемента с самим собой. Таким образом, алгоритм будет прокручен на полную катушку – а результат-то – нулевой! Нехорошо! А что, есть идея получше? Пожалуй. Рассмотрим вариант второй.

Это будет так называемая “пузырьковая” сортировка. Суть ее в следующем. Устраиваем смотр массиву. Идем вдоль него от s:=0 до s:=n−1. И на каждом шаге оцениваем “правильно” ли стоят s-ый и (s+1)-ый элементы. Если

сортировка по неубыванию, то правильной будет ситуация, когда A[s+1]>=A[s]. При невыполнении этого условия – они стоят неправильно. Устраним это безобразие – поменяем их местами (через “третий стаканчик” – помните?) и сделаем замечание “сержанту” – непорядок! Если по завершению смотра сержант не получил ни единого “прокола” массив можно возвращать – он в полном порядке. Если же хотя бы одно замечание имело место, назначается повторный смотр и так до тех пор, пока очередной смотр пройдет без сучка, без задоринки.

```
Procedure SortMasD(var A:Coefr);
var s,n:integer;
    r:real;
    F:Boolean;
begin
  n:=round(A[-1]);
  repeat
    F:=true;
    for s:=0 to n-1 do
      if A[s+1]<A[s] then
        begin
          r:=A[s]; A[s]:=A[s+1];
          A[s+1]:=r; F:=false
        end
  until F
end;
```

Особенность реализации этого алгоритма – переменная F:Boolean. Это – флажок, состояние которого демонстрирует состояние талона (типа ГАИ-шного). Нет проколов (F=true), есть проколы (неважно даже сколько) – F=false. Цикл repeat - смотр. Начинаем с чистого талона (F:=true - оптимистическая гипотеза). Если же непорядок зафиксирован (A[s+1]<A[s]), то – принятие мер (перестановка и дырка в талон). В конце смотра в зависимости от состояния талона либо добро на окончание мероприятия (until F – когда F=true - выходим из цикла) либо – все сначала.

2.6. Удаление из массива “дефектных” элементов

Представим себе, что в массиве `A:Coef` находятся результаты некоего эксперимента или, скажем, серии экспериментов. Причем, условия эксперимента таковы, что некоторые “замеры” (результаты) в силу каких-то обстоятельств – “не лезут ни в какие ворота”. И подобные “результаты” лучше отсеять с самого начала, т.к. они только искажают картину. Подобный прием применяется при статистических исследованиях, при оценке сигналов, поступающих с датчиков технологических параметров, и во многих других случаях. Для простоты примем, что отсеиванию подлежат элементы массива с отрицательным знаком. Тогда, если при просмотре массива обнаруживается `A[s]<0`, то этот элемент подлежит удалению (путем “смыкания рядов”), при этом после каждого удаления количество игроков уменьшается на единицу, а значение s увеличивается (на 1) только тогда, когда `s`-ый элемент “устоял”.

```
Procedure Clean(var A:Coefr);
var s,z,n:integer;
begin
  n:=round(A[-1]); s:=0;
  repeat
    if A[s]<0
      then
        begin
          for Z:=s to n-1 do A[z]:=A[z+1];
          dec(n)
        end
      else inc(s)
  until s>n;
  A[-1]:=n;
end;
```

2.7. Перезапись массива в обратном порядке

Чтобы массив “вывернуть наизнанку” надо поменять местами его первый элемент с последним, второй с предпоследним, третий – с предпредпоследним и т.д. Например, так

```
Procedure InvMas(var A:Coefr);
```

```
var s,n:integer; r:real;
begin
  n:=round(A[-1]);
  for s:=0 to (n-1) div 2 do
    begin
      r:=A[s]; A[s]:=A[n-s]; A[n-s]:=r
    end
end;
```

Проверим алгоритм на примерах. Пусть `n=2`. `1 div 2 = 0`. Таким образом, конечное значение `s` в цикле равно нулю, и тело цикла будет выполнено один раз (при `s:=0`). При этом поменяются местами `A[0]` и `A[2]`. `A[1]` останется нетронутым. Так и должно быть – оно на своем месте.

Пусть теперь `n=3`. Тогда `3 div 2 = 1`. Цикл по `s` будет "прокручен" дважды – при `s=0` и `s=1`. На первом "витке" поменяются местами `A[0]` и `A[3]`, на втором витке - `A[1]` и `A[2]`. Тоже - правильно. Итак, алгоритм работает правильно при четных и нечетных значениях `n`. Предлагаем читателю самостоятельно модифицировать алгоритм на случай, когда нумерация элементов массива начинается не с нуля, а, скажем, с `k` (неважно положительного или отрицательного). Как будет выглядеть цикл? "Ищи, выдумывай, пробуй!" (В.В. Маяковский)

2.8. Численное дифференцирование функций, заданных таблично

Пусть некая функция $f(x)$ задана в диапазоне $x_n..x_k$ массивом `A:Coef` своих значений с шагом

$$h = \frac{x_k - x_n}{L} \tag{2.3}$$

где L - некоторое целое положительное число. Схема размещения информации такова

	-1	0	1	2	3	4		…		L	L+1	…	601
A	L	f_0	f_1	f_2	f_3	f_4				f_L	h		

Значение x_s, соответствующее f_s ($f(x)\big|_{x=x_n+sh}$), как это видно из приведенной записи, определяются так $x_s = x_n + sh$, $0 \le s \le L$, где s - номер шага, h - величина шага.

Предполагается, что шаг h достаточно мелок, чтобы таблица (массив A) с необходимой полнотой отражала все существенные особенности исследуемой функции в рассматриваемом диапазоне. Для этого, с одной стороны, ширина диапазона не должна быть слишком большой, с другой - L должно быть достаточно большим, ну, и с "третьей стороны" – функция должна вести себя "прилично". Все приведенные рассуждения явно страдают неконкретностью. Конкретнее можно будет говорить только применительно к конкретной функции (и конкретным значениям x_n, x_k и L). Проблемам численного дифференцирования и интегрирования можно посвящать отдельные книги. Рассмотрим самые элементарные подходы.

Если требуется определить производную в точке (например, в x_s), то речь может идти о производной слева

$$\frac{df(x)}{dx}\Big|_{x=x_S-\varepsilon} \approx \frac{f_s - f_{s-1}}{h},\ 1 \le s \le L \qquad (2.4)$$

или о производной справа

$$\frac{df(x)}{dx}\Big|_{x=x_S+\varepsilon} \approx \frac{f_{s+1} - f_s}{h},\ 0 \le s \le L-1. \qquad (2.5)$$

Естественно, формулы (2.4) и (2.5) дают весьма грубую оценку значения производной в точке x_s. Они тем лучше соответствовали бы классическому определению производной, чем меньше была бы величина шага h (в пределе при $h \to 0$). Вот почему желательно сужение диапазона и увеличение L в формуле (2.3). Однако, при конечном h у нас возникает конфликт ("кому верить"), поскольку в общем случае формулы (2.4) и (2.5) будут давать разные оценки значения производной в точке x_s. Чтобы "никому не было обидно" возьмем среднее арифметическое

$$\left.\frac{df(x)}{dx}\right|_{x=x_s} \approx \frac{f_{s+1}-f_{s-1}}{2h},\ 1 \le s \le L-1. \qquad (2.6)$$

К слову, выражения $f_s - f_{s-1}$, $f_{s+1} - f_s$ и $(f_{s+1} - f_{s-1})/2$ называют соответственно разностью назад, разностью вперед и усредненной центральной разностью. Интуитивно чувствуется, что от формулы (2.6) можно ожидать большей точности, чем от предшествующих, почему ей при прочих равных условиях, следует отдать предпочтение. Но как следует из (2.6) применять ее можно только для внутренних узлов ($1 \le s \le L-1$). Формулу (2.4) можно использовать для обслуживания крайнего правого узла ($s = L$), а формула (2.5) пригодится для крайнего левого узла ($s = 0$). Обращаем внимание читателя на ячейку с номером ($L+1$) – там хранится h - шаг таблицы – пригодится при расчете производных.

Места для x_n и x_k в массиве A не нашлось, придется хранить их где-то отдельно (по крайней мере - одного из них, т.к. другой легко восстановить, зная L и h). Но для выполнения самого процесса дифференцирования знание x_n, x_k и не обязательно. Если окажется, что $x_n = 0$, то тем проще. Итак, решаем задачу – имея массив A ординат исследуемой функции сформировать массив B значений ее производной в тех же узлах, что и сама функция

```
Procedure DifMas1(var A:CoefL; var B:CoefL);
var L,s:integer; h:real;
begin
  L:=round(A[-1]); h:=A[L+1];
  B[-1]:=L; B[L+1]:=h;
  for s:=1 to L-1 do
    B[s]:=(A[s+1]-A[s-1])/(2*h);
  B[0]:=(A[1]-A[0])/h;
  B[L]:=(A[L]-A[L-1])/h
end;
```

При использовании этой процедуры может возникнуть идея возвращать массив-производную в том же массиве, который был передан процедуре в качестве массива-первообразной. В таком случае массив B будет неправильным. Понятно – почему? Чтобы исключить подобную ситуацию слегка переделаем процедуру

```
Procedure DifMas(var A:CoefL);
var L,s:integer;
    h:real;
    B:CoefL;
begin
  L:=round(A[-1]); h:=A[L+1];
  B[-1]:=L; B[L+1]:=h;
  for s:=1 to L-1 do
    B[s]:=(A[s+1]-A[s-1])/(2*h);
  B[0]:=(A[1]-A[0])/h;
  B[L]:=(A[L]-A[L-1])/h;
  A:=B
end;
```

2.9. Численное интегрирование функций, заданных таблично

Как и в предыдущем параграфе исходная функция $f(x)$ задается массивом `A:CoefL`. Пусть требуется составить таблицу значений

$$y(x) = \int_{x_n}^{x_k} f(x)dx \tag{2.7}$$

на той же сетке, на которой сформирован массив `A`. Воспользуемся общеизвестным методом трапеций. Тогда

$$y_0 = 0;$$
$$y_s \cong y_{s-1} + \frac{f_s + f_{s-1}}{2} h,\ 1 \le s \le L. \tag{2.8}$$

Программно это может выглядеть, например, так

```
Procedure IntMas(var A:CoefL);
var L,s:integer;
    h:real;
    B:CoefL;
begin
  L:=round(A[-1]); h:=A[L+1];
  B[-1]:=L; B[L+1]:=h; B[0]:=0;
  for s:=1 to L do
    B[s]:=B[s-1]+(A[s-1]+ A[s])/2*h;
  A:=B
end;
```

При необходимости можно было бы сэкономить массив B (обойтись без посредника). Предлагаем читателю осуществить этот проект самостоятельно.

3. Работа с полиномами

В п 2.1 мы специально ввели в рассмотрение

```
type Coef = array [-1..31] of real;
```

ориентированный в первую очередь на работу с полиномами вида

$$A(p) = a_0 + a_1 p + a_2 p^2 + ... + a_n p^n .$$

Информация о полиноме располагается в массиве типа `Coef` по схеме

-1	0	1	2	3	…	n	…	31
n	a_0	a_1	a_2	a_3	…	a_n	…	

После этого напоминания приступим к рассмотрению элементарных операций над полиномами.

3.1. Алгебраическое суммирование полиномов

$$C(p) = A(p) + B(p).$$

Если есть два полинома, например, $A(p)$ и $B(p)$ произвольных степеней, то их алгебраическое суммирование (сложение или вычитание) сводится к соответствующему суммированию их компонентов с одинаковыми степенями p. Степень полинома суммы (разности), очевидно, равна большей из степеней, если только старшие коэффициенты при соответствующем суммировании не сократятся, тогда фактическая степень суммы может оказаться меньше формальной (большей из степеней полиномов-слагаемых). Оформим рассматриваемый алгоритм так:

```
procedure AlSumPol (A, B:Coef; OP:char;     var C:Coef);
var Na, Nb, Nc, S: integer;
begin
  Na:= round(A[-1]);
  Nb:=round(B[-1]);
  if Na > Nb then Nc:=Na
             else Nc:=Nb;
  C[-1]:= Nc;
  for S:=0 to Nc do
```

```
    begin
      if S <= Na
        then C[S]:=A[S]
        else C[S]:=0;
      if S<=Nb then
        if OP='+' then C[S]:=C[S]+B[S]
                  else C[S]:=C[S]-B[S]
    end
end;
```

Итак, в приведенной процедуре `A`, `B` – массивы с информацией о полиномах-слагаемых, а `C` – массив полинома-результата. Параметр `OP:char` определяет тип математического суммирования. Если `OP = '+'` то это сложение, при `OP = '–'` имеем вычитание `C(p)=A(p)-B(p)`. Степень `Nc`, зарегистрированная в `C[-1]` – формальная. `C[n], C[n-1]` и т.д. могут оказаться равными `0` и тогда фактическая степень полинома C(p) может оказаться меньше зарегистрированной. А кто сказал, что у полиномов A(p) и (или) B(p) степени не завышены? Редактирование полинома (любого), т.е. приведеие в соответствие его формальной и фактической степеней можно реализовать в виде, скажем, такой процедуры:

```
procedure RedPol (Eps:real; var A:coef);
var n:integer;
begin
  n:=round (A[-1]);
  while (n>0) and (abs(A[n])<Eps) do dec(n);
  A[-1]:=n;
end;
```

Оператор `n:=round(A[-1])` определяет формальную степень. В цикле `while` оценивается «весомость» коэффициентов, претендующих на право рассматриваться старшими (коэффициенты при старших степенях аргумента *p*). Несолидные (меньше `Eps` по модулю) в расчет не принимаются. Дело в том, что и исходные коэффициенты полиномов, стоящие у истоков исследования, известны, как правило, не точно, а приблизительно, т.к. получены на базе параметров, определенных с какой-то погрешностью (поверхности, веса,

теплоемкости, коэффициенты тепло- и массоотдачи). Далее полином может быть получен в результате каких-либо действий, операций, выполняемых над величинами типа `real`, а результаты компьютерных расчетов над подобными операндами – непредсказуемы. Последнее утверждение, возможно, излишне категорично, но тем не менее это так. Представьте для примера результат деления на число, модуль которого сопоставим или меньше возможной погрешности, которую это число может в себе содержать. Результат будет «плюс-минус миллион». И это не преувеличение. Мы говорим это для того чтобы убедить читателя в том, что за старшим коэффициентом полинома с малым модулем может скрываться и настоящий ноль (возможно, этот коэффициент был бы настоящим нулем, если бы все вычисления выполнялись точно; но ведь они, как правило, не точны.) Это объективная реальность, а против нее «не попрешь!». С нею можно только считаться («благодарно принимать»). Именно поэтому в процессе редактирования полинома мы сравниваем `Abs (A[n])` не с чистым нулем, а `Eps` (идентификатор малой величины ε). И последнее. Мы ничего не делаем с «несерьезными коэффициентами». «Бог им судья». Мы просто исключаем их из рассмотрения, корректируя `A[-1]`. Этого достаточно.

3.2. Умножение полиномов

Итак, требуется вычислить:

$$C(p) = A(p) * B(p). \tag{3.2}$$

Алгоритм перемножения полиномов общеизвестен. Каждое слагаемое одного полинома множится на каждое слагаемое другого, после чего (или в процессе перемножения)сводятся подобные члены (складываются коэффициенты при одинаковых степенях p). Степень полинома - произведения равна сумме степеней полиномов-сомножителей.

```
Procedure UmnPol (A,B:Coef; Var C:Coef);
var Na, Nb, Nc, S, Z:integer;
begin
  Na:=round (A[-1]); Nb:=round(B[-1]);
```

```
  Nc:=Na+Nb; C[-1]:=Nc;
  for S:=0 to Nc do C[S]:=0;
  for Z:=0 to Na do
    for S:=0 to Nb do
      C[Z+S]:=C[Z+S]+A[Z]*B[S];
end;
```

Ну, со степенями полиномов все очевидно. Цикл `for S:=0 to Nc do C[S]:=0;` решает задачу чистки массива `C` – обнуления его активных элементов с тем, чтобы подготовить их для накопления в них ранее упоминавшихся «подобных членов». При принятой схеме размещения коэффициентов полиномов в соответствующих ячейках обслуживающих их массивов, индекс ячейки совпадает со степенью p при размещенном в ней коэффициенте. Когда множится `A[Z]` на `B[S]`, то одновременно множится p^s на p^z – в итоге получаем p^{s+z}, а произведение `A[Z]*B[S]` заносится (добавляется к ранее накопленной сумме подобных – с такой же суммой индексов – слагаемых) в ячейку с номером (`Z+S`). Таким образом, вложенные циклы (внешний по `Z`, перебирающий слагаемые полинома A(p) и внутренний по `S` , перебирающий слагаемые полинома B(p)) одновременно решают две задачи: собственно перемножение полиномов A(p) и B(p) и сведение подобных членов в произведении. Что и требовалось. Предварительная чистка массива C, оказывается, имела таки смысл.

3.3. Деление полиномов

Речь идет об операции вида:

$$\frac{A(p)}{B(p)} = C(p) + \frac{D(p)}{B(p)}. \qquad (3.3)$$

Здесь: $C(p)$ - полином – частное, $D(p)$ - полином – остаток (его степень меньше степени полинома – делителя $B(p)$).

Если степень полинома – делимого $A(p)$ ниже степени полинома – делителя $B(p)$, то полином – частное $C(p)$ будет отсутствовать. ($C(p) = 0$ – полином нулевой степени, в котором $C(0) = 0$). В таком случае $D(p) = A(p)$. Это – элементарно. Интереснее, когда степень $A(p)$ больше или равна степени $B(p)$. Начнем с

определения степеней Nc и Nd полиномов $C(p)$ и $D(p)$. Очевидно Nc = Na – Nb. Степень Nd формально будет равна Nd = Nb – 1, ну, а фактически может быть меньше вплоть до нулевой так как не исключено, например, что полином $A(p)$ делится на $B(p)$ без остатка. Тогда, $D(p) = 0$. Правда, последнее утверждение при компьютерной реализации может выполняться только приближенно, когда $D[0]$ получается просто малым по модулю, а не нулем (за счет неизбежных погрешностей вычислений). Как мы ранее упоминали, результаты компьютерных вычислений над вещественными числами бывают непредсказуемы. Для разработки алгоритмов вычисления коэффициентов полиномов $C(p)$ и $D(p)$ рассмотрим пример с целью выявления закономерностей, которые затем должны быть обобщены. Итак, пусть Na = 6, Nb = 2, тогда согласно (3.3) можно записать Nc = Na – Nb = 4, Nd = Nb – 1=1.

$$\frac{a_6p^6+a_5p^5+a_4p^4+a_3p^3+a_2p^2+a_1p+a_0}{b_2p^2+b_1p+b_0}= \\ =c_4p^4+c_3p^3+c_2p^2+c_1p+c_0+\frac{d_1p+d_0}{b_2p^2+b_1p+b_0}. \tag{3.4}$$

Умножим (3.4) на полином $b_2p^2+b_1p+b$. Получим

$$\left(c_4p^4+c_3p^3+c_2p^2+c_1p+c_0\right)\left(b_2p^2+b_1p+b_0\right)+\left(d_1p+d_0\right)= \\ =a_6p^6+a_5p^5+a_4p^4+a_3p^3+a_2p^2+a_1p+a_0. \tag{3.5}$$

Раскрываем скобки, сводим подобные и приравниваем коэффициенты при равных степенях p в левой и правой частях. (3.5)

$$p^6:\quad c_4b_2=a_6,$$

$$p^5:\quad c_4b_1+c_3b_2=a_5,$$

$$p^4:\quad c_4b_0+c_3b_1+c_2b_2=a_4,$$

$$p^3:\quad c_3b_0+c_2b_1+c_1b_2=a_3,$$

$$p^2:\quad c_2b_0+c_1b_1+c_0b_2=a_2,$$

$$p^1:\quad c_1b_0+c_0b_1+d_1=a_1,$$

$$p^0:\quad c_0b_2+d_0=a_0.$$

Выписанную систему соотношений можно рассматривать как систему уравнений относительно искомых коэффициентов полиномов $C(p)$ и $D(p)$. Имеет смысл обратить внимание на два обстоятельства: первое – данная система распадается на две независимые системы, а именно: первые Nc+1 уравнения содержат в качестве переменных коэффициенты полинома $C(p)$, а последующие Nb+1 уравнений – кроме c_0 и c_1 – коэффициенты полинома $D(p)$. Итак, удобно вначале решить первую подсистему, т.е. определить c_i (0<i<Nc), затем имея коэффициенты полинома $C(p)$, из второй подсистемы найти коэффициенты полинома $D(p)$.

Второе обстоятельство – первая система элементарно решается путем последовательного определения неизвестных. Итак, из первого уравнения определяется C_n (в общем случае c_{N_c}), из второго – c_3 (т.е. c_{N_c-1})и т.д. пока из (Nc+1)-го уравнение определятся c_0, на чем решение первой подсистемы и заканчивается. Из второй подсистемы (при уже найденных коэффициентах C_i, 0<i<Nc коэффициенты полинома $D(p)$ определяются в произвольной последовательности – по одному из каждого уравнения этой системы. Описанный алгоритм решения первой подсистемы можно обобщить такой формулой:

$$c_z = \frac{a_{Nb+Z} - \sum_{S=0}^{Nd} b_s c_{Nb+Z-S}}{b_{Nb}},\ \mathrm{Nc} \geq \mathrm{z} \geq 0. \qquad (3.6)$$

Здесь для удобства записи принято, что коэффициенты c_z со значениями индекса больше Nc (от Nc+1 до Na) равны нулю. Формулой (3.6) необходимо пользоваться перебирая z от Nc до 0 (именно в таком порядке!). Решение же второй подсистемы можно представить так:

$$d_z = a_z - \sum_{S=0}^{Z} b_s c_{z-s},\ 0 \leq \mathrm{z} \leq \mathrm{Nd}. \qquad (3.7)$$

Мы предлагаем читателю убедиться в корректности этих формул. Откуда мы их взяли? Придумали! А вы сами попробуйте подумать и придумать их (или что-то эквивалентное) самостоятельно. Может у вас получится даже лучше (по

крайней мере изящнее) чем у нас. Важно только, чтобы оно было правильным. Чтобы оно сработало при любых значениях Na и Nb. Процесс улавливания (и формулирования!) закономерностей в программировании очень важен. Трудно дать универсальный алгоритм как это сделать, но если вы понимаете суть алгоритма и его механизм, то «увидеть» подобные закономерности пусть, даже на ощупь, путем проб и ошибок (пробуете – проверяете – корректируете, снова корректируете, и так, пока не получится т.е. пока не убедитесь сами или не убедите потенциальных оппонентов в том, что сформулированная вами закономерность действительно имеет место и, следовательно, имеет право на существование – на то, чтобы ею в дальнейшем можно было пользоваться не сомневаясь в ее правильности).

С учетом сказанного:

```
procedure DelPol(A,B:Coef; var C,D:Coef);
var Z,S,Na,Nb,Nc,Nd: integer;
    Sum:real;
begin
  Na:=round(A[-1]); Nb:=round(B[-1]);
  if Na<Nb
    then
      begin
        D:=A; Nd:=Na; Nc:=0; C[0]:=0
      end;
    else
      begin
        Nd:=Nb-1; Nc:=Na-Nb;
        for Z:= Nc+1 to Na do C[Z]:=0;
        for Z:=Nc downto 0 do
          begin
            Sum:=0;
            for S:=0 to Nd do
            Sum:=Sum+B[S]*C[Nb+Z-S];
            C[Z]:=(A[Nb+Z]-Sum)/B[Nb]
          end;
        for Z:=0 to Nd do
          Begin
            Sum:=0;
            for S:=0 to Z do
            Sum:= Sum+B[S]*C[Z-S];
```

```
            D[Z]:=A[Z]-Sum;
          end
      end;
  C[-1]:= Nc; D[-1]:= Nd;
end;
```

Итак, переменная `Sum:real;` предназначена для вычисления сумм, фигурирующих в формулах (3.6) и (3.7). Na, Nb – степени полиномов $A(p)$ и $B(p)$. `Nc`, `Nd` – соответственно для полиномов $C(p)$ и $D(p)$. Вариант Na<Nb ранее обсуждался, он реализуется после слова `then` в составном операторе. Здесь, пожалуй, следует обратить внимание на оператор `D:=A`. Это копирование массива A в массив D (целиком). Подобное присваивание допускается в Турбо Паскале для однотипных массивов (а в нашем случае и `D` и `A` – массивы одного и того же типа `Coef` – смотри заголовок программы `DelPol`). Операторы: `C[-1]:=0; C[0]:=0` задают полином $C(p)$=0. (оператор `Nc:=0` и `C[-1]:=0` – эквивалентны).

Если же условие Na<Nb не выполняется, то – смотри составной оператор после `else.`

`Nd:=Nb-1.` Это формальная степень $D(p)$.

`Nc:=Na-Nb` – степень $C(p)$. `For Z:=Nc+1 to Na do C[Z]:=0;` это обнуление коэффициентов полинома $C(p)$ при степенях превышающих Nc (до Na) как уже выше упоминулось – условие корректности формулы (3.6).

В цикле `For Z:=Nc to downto 0 do …`

`Sum:=0;` Это подготовка к вычислению суммы

$$\sum_{S=0}^{Nd} b_s c_{Nb+Z-S} \cdot$$

Цикл по `S` эту сумму накапливает (вычисляет по формуле, в чем читатель может убедиться сам – цикл, как говорится, один в один реализует формулу суммы). Оператор `C[Z]:=(A[Nb+Z]-Sum)/B[Nb]` буквально реализует формулу (3.6). Следующий цикл (по Z) буквально реализует формулу (3.7) – вначале вычисляется значение суммы:

$$\sum_{S=0}^{Z} b_s c_{Z-S},$$

а потом значение суммы подставляется в (3.7). Заполнение ячейки `C[-1]` и `D[-1]` – понятно зачем.

3.4. Разложение дробно-рациональной функции в ряд Маклорена

Речь идет о преобразовании вида:

$$\frac{B(p)}{A(p)} = c_0 + c_1 p + c_2 p^2 + c_3 p^3 + ... + c_n p^n + ... \qquad (3.8)$$

Раскроем выражения для $B(p)$ и $A(p)$

$$\frac{b_0 + b_1 p + b_2 p^2 + ... + b_{Nb} p^{Nb}}{a_0 + a_1 p + a_2 p^2 + ... + a_{Nb} p^{Nb}} = c_0 + c_1 p + c_2 p^2 + c_3 p^3 + ... + c_n p^n + ... \quad (3.9)$$

Ряд в правой части (3.8) бесконечен, но на практике будем его ограничивать членом $c_n p^n$, где n – некоторое наперед заданное целое положительное число, что и реализовано в (3.9). Помножим левую и правую части на знаменатель левой части – и приравняем коэффициенты при одинаковых степенях p в обеих частях равенства

$$p^0: \quad c_0 a_0 = b_0,$$

$$p^1: \quad c_0 a_1 + c_1 a_0 = b_1,$$

$$p^2: \quad c_0 a_2 + c_1 a_1 + c_2 a_0 = b_2,$$

$$p^3: \quad c_0 a_3 + c_1 a_2 + c_2 a_1 + c_3 a_0 = b_3,$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots..$$

$$p^n: \quad c_0 a_n + c_1 a_{n-1} + c_2 a_{n-2} + ... + c_n a_0 = b_n.$$

В приведенных уравнениях принимается, что коэффициенты в полиномах $B(p)$ и $A(p)$ при степенях p, превышающих соответственно Nb и Na равняются нулю. Т.е. мы фактически «подтягиваем» степени полиномов $B(p)$ и $A(p)$ до степени полинома $C(p)$ – в правой части (3.9). Если окажется, что n меньше или равен Nb или Na (или обоим вместе) – ну, что ж, тогда необходимость в «подтягивании» отпадает сама собой, соответствующий цикл не сработает –

только и всего («на нет и суда нет»).

Если (3.9) рассматривать как систему линейных алгебраических уравнений относительно искомых коэффициентов полинома $C(p)$ (а ведь ряд в правой части (3.9) – не что иное как полином!) то эта система элементарно решается путем последовательного определения неизвестных: из первого уравнения c_0, затем из второго – c_1, из третьего – c_2 и т.д., пока, наконец, из очередного c_n. Напрашивается обобщающая формула:

$$\begin{cases} z_0 = \dfrac{b_0}{a_0}, \\ c_z = \dfrac{b_z - \sum\limits_{S=0}^{Z-1} c_s a_{Z-S}}{a_0}, \quad 1 < \mathrm{z} < \mathrm{n}. \end{cases} \tag{3.10}$$

Оформим все сказанное в виде подпрограммы `McLDrf` (ряд Маклорена для дробно-рациональной функции).

```
Procedure McLDrf (B,A:Coef; n:integer; var C:Coef);
var Nb,Na,Z,S:integer;
    Sum:real;
Begin
  Nb:=round(B[-1]); Na:=round(A[-1]);
  for S:=Nb+1 to n do B[S]:=0;
  for S:=Na+1 to n do A[S]:=0;
  C[-1]:=n; C[0]:=B[0]/A[0];
  for Z:=1 to n do
    begin
      Sum:=0;
      for S:=0 to Z-1 do Sum:=Sum+C[S]*A[Z-S];
      C[Z]:=(B[Z]-Sum)/A[0];
    end
end;
```

Внимание! В полиноме $A(p)$ коэффициент a_0 не должен равняться нулю (компьютер на ноль делить не любит). А что если? Вынесите в $A(p)$ за скобки p^q, где q – такое минимальное положительное число, при котором в скобке младший коэффициент (при p^0) станет неравным нулю. Вместо $A(p)$ возьмите упомянутую

только что скобку, а после того как полином $C(p)$ будет получен, поделите все его члены на p^q. Правда, он после этого престанет быть полиномом, но что делать! «Такова жизнь» -- как говорят французы.

3.5. Алгебраическое сложение отрезков бесконечных степенных рядов

Итак, имеем два бесконечных ряда B(p) и A(p):

$$A(p) = a_0 + a_1 p + a_2 p^2 + ... + a_{Na} p^{Na} + ... \qquad (3.11)$$

$$B(p) = b_0 + b_1 p + b_2 p^2 + ... + b_{Nb} p^{Nb} + ... \qquad (3.12)$$

представленных массивами `A,B:Coef`, в первом из которых ряд $A(p)$ ограничен его начальным отрезком с последним слагаемым $a_{Na} p^{Na}$, тогда как в $B(p)$ последним учитываемым слагаемым есть $b_{Nb} p^{Nb}$. Требуется найти начальный отрезок ряда $C(p)$, представляющего алгебраическую сумму этих рядов:

$$C(p) = A(p) \pm B(p). \qquad (3.13)$$

Естественной является идея оперировать с рядами $A(p)$ и $B(p)$. как с полиномами, т.е. свести задачу к ранее рассмотренной в п. 3.1. Отличие данной задачи от задачи в п. 3.1. в том, что полиномы, «закодированные» в массивах `A,B:Coef`, это лишь «надводные части айсбергов», а оперировать нужно с «айсбергами» целиком, а не только с их надводными частями. При сложении (алгебраическом) «настоящих» полиномов степень полинома суммы равнялась большей из степеней полиномов-слагаемых. А в случае бесконечных рядов? Наоборот! Ведь здесь «недостающие» старшие коэффициенты в «младшем» полиноме заменить нулем нельзя, т.к. они в общем случае не нули, но информацией об их конкретных значениях мы не располагаем – ну нет этой информации в соответствующем массиве! Поэтому и приходится ограничиваться в сумме сложением (вычитанием) только тех коэффициентов, которых (оба!) имеются в наличии.

```
procedure AlSumSer(A,B:Coef; Zn:integer; var C:Coef;);
var Na,Nb,Nc,S: integer;
begin
```

```
  Na:=round(A[-1]); Nb:=round(B[-1]);
  if Na<Nb then Nc:=Na else Nc:=Nb;
  C[-1]:=Nc;
  for S:=0 to Nc do
    C[S]:=A[S]+Zn*B[S]
end;
```

В данной процедуре в отличие от процедуры `AlSumPol`, рассмотренной в п. 3.1, тип операции задается параметром `Zn:integer`, который может принимать значения +1 или, соответственно –1 (сложение/вычитание). Это, пожалуй, удобнее параметра `Op:char`, когда операции требуется «разводить» с помощью логического оператора. Более того, если Zn отнести к типу `real`, то тогда он становится полноценным коэффициентом в сумме типа:

$$C(p) = A(p) + Zn\ B(p), \tag{3.14}$$

тем самым возможности процедуры `AlSumSer` расширяются.

3.6. Умножение рядов

Итак, требуется перемножить ряды (3.11) и (3.12), представленные так же, как в п. 3.5. Легко сообразить, что в дело можно запускать полиномы, степень которых не превышает меньшую из степеней полиномов-сомножителей (иначе при сведении подобных членов мы недосчитаемся части слагаемых, а это обессмысливает значение «частичной» суммы.). Итак

```
procedure UmnSer (A,B:Coef; var C:Coef);
var Na,Nb,Nc,S,Z:integer;
begin
  Na:=round(A[-1]); Nb:=round(B[-1]);
  if Na<Nb then Nc:=Na
           else Nc:=Nb;
  for S:=0 to Nc do C[S]:=0;
  C[-1]:=Nc;
  for Z:=0 to Nc do
    for S:=0 to Nc do
      if Z+S<=Nc then C[Z+S]:=C[Z+S]+A[Z]*B[S]
end;
```

3.7. Деление рядов

Будем делить ряд (3.11) на ряд (3.12). Применительно к настоящим полиномам задача рассматривалась в 3.4. Проанализируем формулу (3.10) для определения коэффициентов частного. Очевидно, что C_z не может быть определено для Z, превышающих Nb (иначе откуда мы возьмем C_z?) С C_z в сумме проблем нет , т.к. формула для C_z применяется при последовательном наращивании Z, когда на момент вычисления C_z все его предшественники уже определены (это C_s, 0<=s<=n-1). При S=0 , $a_{z-s} = a_z$, а при s=z-1 (верхний предел индекса сумма) $a_{z-s} = a_1$. Итак, для вычисления C_z необходимо B_z и все коэффициенты от a_0 до a_z (как без a_0, если оно стоит в знаменателе формулы для C_z?). Так что получается, что мы можем сформировать начальный отрезок ряда-частного, у которого старший член имеет P в степени не выше меньшей из степеней полиномов, представляющих ряд-делимое и ряд-делитель.

```
procedure DelSer (A,B:Coef; var C:Coef);
var Na, Nb, Nc, Z, S: real;
    Sum:real;
begin
  Na:=round(A[-1]); Nb:=round(B[-1]);
  if Na<Nb then Nc:=Na
           else Nc:=Nb;
  C[-1]:=Nc;  C[0]:=A[0]/B[0];
  for Z:=1 to Nc do
    begin
      Sum:=0;
      for S:=0 to Z-1 do
        Sum:=Sum+C[S]*B[Z-S];
      C[Z]:=(A[Z]-Sum)/B[0]
    end
end;
```

Интересно обратил ли внимание читатель на то обстоятельство, что в п. 3.4. делится $B(p)$ на $A(p)$, а в (3.7) наоборот? Зачем авторы запутывают бедного студента? Ну, во-первых, чтобы последний не терял бдительности. А во-вторых, у авторов есть некоторые соображения, оправдывающие их, но об этом долго рассказывать.

3.8. Дифференцирование полиномов

Продифференцируем полином $A(p)$ по p

$$\frac{dA(p)}{dp}=\frac{d}{dp}\left(a_0+a_1p+a_2p^2+...+a_Np^N\right)=a_1+2a_2p+3a_3p^2...+na_np^{n-1}. \quad (3.15)$$

Внесем коэффициенты полинома-производной в массив типа `Coef`

	-1	0	1	2	3	…	n-1	…	31
B	n-1	a_1	$2a_2$	$3a_3$	$4a_4$	…	na_n	…	

Очевидно, что заполнение ячеек массива B можно выполнить по схеме

$$B_S=(s+1)a_{S+1},\ 0<=S<=n\text{-}1. \quad (3.16)$$

Итак, очевидно, дифференцирование полинома можно представить такой подпрограммой.

```
procedure DifPol (var A:Coef);
var S, N:integer;
begin
  N:=round(A[-1]); A[-1]:=N-1;
  for S:=0 to N-1 do A[S]:=(S+1)*A[S+1]
end;
```

3.9. Замена в полиноме p на p_1-L

Выполним в полиноме $A(p)$ замену его переменной p на p_1-L, где p_1 - новая переменная, L – константа (типа `real` положительная или отрицательная). Получим полином $B(p_1)$.

$$A(p)\big|_{p=p_1-L}=B(p_1). \quad (3.17)$$

Задача заключается в определении коэффициентов полинома $B(p_1)$. Степень полинома $B(p_1)$, очевидно, будет такой же как у полинома $A(p)$. Пусть для конкретности Na=Nb=5. Тогда

$$A(p)=a_0+a_1p+a_3p^3+a_4p^4+a_5p^5, \quad (3.18)$$

$$B(p)=b_0+b_1p+b_3p^3+b_4p^4+b_5p^5. \quad (3.19)$$

Произведем в п. (3.17) еще одну замену, а именно заменим p_1 на (p+L).

Получим

$$A(p) = B(p + L) \tag{3.20}$$

или в развернутом виде

$$A(p) = b_0 + b_1(p + L) + b_2(p + L)^2 + b_3(p + L)^3 + b_4(p + L)^4 + b_5(p + L)^5. \tag{3.21}$$

Выражение (3.21) – тождество. Оно должно выполняться при любом значении переменной p. Возмем p=-L. При этом скобки в (3.21) превращаются в нули и

$$b_0 = A(p)\big|_{p=-L}. \tag{3.22}$$

Один коэффициент полинома $B(p_1)$ найден. Продифференцируем (3.21) по p

$$\frac{dA(p)}{dp} = b_1 + 2b_2(p + L) + 3b_3(p + L)^2 + 4b_4(p + L)^3 + 5b_5(p + L)^4. \tag{3.23}$$

Рассмотрим последнее выражение при p=-L. Получим

$$b_1 = \left.\frac{dA(p)}{dp}\right|_{p=-L}. \tag{3.24}$$

Дифференцируем (3.23) по p

$$\frac{d^2A(p)}{dp^2} = 2b_2 + 3*2b_3(p + L) + 4*3b_4(p + L)^2 + 5*4b_5(p + L)^3. \tag{3.25}$$

Из (3.25)
$$b_2 = \frac{1}{2}\left.\frac{d^2A(p)}{dp^2}\right|_{p=-L}.$$

Продолжая этот процесс получаем

$$b_3 = \frac{1}{3!}\left.\frac{d^3A(p)}{dp^3}\right|_{p=-L}, \tag{3.27}$$

$$b_4 = \frac{1}{4!}\left.\frac{d^4A(p)}{dp^4}\right|_{p=-L}, \tag{3.28}$$

$$b_5 = \frac{1}{5!}\left.\frac{d^5A(p)}{dp^5}\right|_{p=-L} \tag{3.29}$$

Формулы (3.22), (3.24), (3.26), (3.27)—(3.29) можно обобщить

$$b_s = \frac{1}{s!}\left.\frac{d^sA(p)}{dp^s}\right|_{p=-L},\ 0 \le S \le n.$$

А $b_n = a_n$ – заметили? Оформим рассмотренный алгоритм в виде подпрограммы ZamPPL1.

```
procedure ZamPPL1 (A:Coef; L:real; var B:Coef);
var S, n :integer;
begin
  N:=round(A[-1]); B[-1]:=n; B[n]:=A[n];
  for S:=0 to n-1 do
    begin
      B[S]:=HorReal(A,-L)/Fact(S);
      DifPol(A);
    end
end;
```

Приведенная подпрограмма подкупает своей компактностью. Но обратной стороной этого достоинства является необходимость иметь под рукой подпрограммы HorReal, Fact, DifPol. Каждая из этих программ сама по себе так проста, что имеет смысл, алгоритмы, реализуемые в HorReal, Fact и DifPol, включить непосредственно в подпрограмму для замены переменной. Тогда получаем программу ZamPPL.

```
procedure ZamPPL (A:Coef; L:real; var B:Coef);
var Z, S, N: integer; F:real;
begin
  F:=1; n:=round(A[-1]);
  B[-1]:=N; B[N]:=A[N];
  for S:=0 to N-1 do
    begin
      C:=A[N-S];
      for Z:=N-S-1 downto 0 do C:=-C*L+A[Z];
      B[S]:=C/F;
      for Z:= 1 to N-S do A[Z-1]:=A[Z]*Z;
      F:=F*(S+1)
    end
end;
```

Почему F (факториал) отнесен к типу real – мы уже обсуждали раньше. Где здесь фрагменты от подпрограмм HorReal, DifPol и Fact – рекомендуем читателю разобраться самостоятельно.

4. Определение корней полиномов

Определение корней полиномов, а в более общей постановке, определение нулей функций произвольного вида – это обширная область математики. Разработано и применяется множество методов решения данной задачи как общих так и ориентированных на исследование специальных типов функций. Полиномы – частный и весьма своеобразный тип функций. Своеобразие полиномов, в частности, состоит в том, что применительно к ним может ставиться вопрос об определении не только вещественных (как для большинства функций) корней. Отдельно взятая функция (не полином) может иметь то или иное количество корней (или не иметь таковых, или иметь их бесконечное множество). А вот полином - дело особое. Полином n-ой степени имеет (обязан иметь!) ровно n корней (не больше и не меньше!). Среди этих корней могут быть простые и кратные, вещественные и комплексные, и серьёзное исследование предполагает возможность определения их всех (до единого) каковыми бы они ни оказались.

4.1. Вычисления значения полинома при вещественном значении его аргумента

Корень полинома – это такое значение его аргумента , при котором данный полином становится равным нулю. Итак , нужно иметь возможность, по крайней мере, проверить не является ли нулём полинома какое-то значение его аргумента. А для этого нужно вычислить значение полинома при этом значении аргумента и уже тогда делать выводы . Итак, полином имеет вид

$$A(p) = a_n p^n + a_{n-1} p^{n-1} + \ldots + a_1 p + a_0 . \qquad (4.1)$$

Это тот же полином, что и (2.1), но только записанный в обратном порядке. Информацию о полиноме (4.1) будем размещать в массиве типа `Coef` точно так же, как это было оговорено для полинома (2.1) – ведь это один и тот же полином , а перестановка слагаемых суммы не меняет. Особых проблем с вычислением выражения (2.1) или, что то же, (4.1) вроде бы и нет. Ну, например, почему бы не вычислить его так?

```
function Polreal (A:Coef; p:real):real;
var s,n:integer; Sum:real;
begin
  n:=round (A[-1]); Sum:=0;
  for S:=0 to n do
    Sum:=Sum+A[s]*Xstn(p,s);
  Polreal:=Sum
end;
```

Всё верно. Вот только нужно “иметь на подхвате” функцию `Xstn`, возводящую число типа `real` в степень типа `integer`. А если бы обойтись без неё? Ну, например, так

```
function Polreal1( A:Coef; p:real):real;
var s,n:integer; Ps:real;
begin
  n:=round (a[-1]); Sum:=0; Ps:=1;
  for s:=0 to n do
    begin
      Sum:=Sum+A[s]*Ps;
      Ps:=Ps*P
    end;
  Polreal1:=Sum
end;
```

Здесь переменная `Ps:real` – это p в степени. Для начала `Ps:=1` – это когда степень равна нулю (для `S:=0` ,когда к текущему значению суммы (`Sum:=0`) прибавляется слагаемое $a_0 * p^0$ (проще говоря, a_0). По мере наращивания `S` (в цикле) синхронно с ним наращивается степень в p^s (путём элементарного домножения предшествующего значения `Ps` на `P` – оператор `Ps:=Ps*P`). Вот ведь как просто!

Большой популярностью у математиков пользуется так называемая схема Хорнера (Горнера) вычисления значения полинома, которая, например, для полинома третьей степени выглядит так

$$A(p) = ((a_3 p + a_2) p + a_1) p + a_0. \qquad (4.2)$$

Легко видеть, что запись (4.2) полностью эквивалентна записи

$$A(p) = a_3 p^3 + a_2 p^2 + a_1 p + a_0 . \qquad (4.3)$$

Раскройте в (4.2) скобки и это станет очевидным. Вычисления в форме (4.2) идут в направлении “изнутри - наружу”. Как принято в алгебре (и в Турбо Паскале) сначала вычисляется выражение в самых внутренних скобках, затем переходим к вычислению выражения в скобках, непосредственно охватывающих предыдущие скобки, - и так, пока не будет реализовано все выражение.

Применительно к полиному произвольной степени программно это может выглядеть так

```
function HorReal (A:Coef; p:real):real;
var s,n:integer; r:real;
begin
  n:=round (A[-1]); r:=A[n];
  for S:=n-1 downto 0 do
    r:=r*p+A[S];
  HorReal:=r
end;
```

Здесь в идентификаторе функции отражены слова Хорнер (в знак признательности господину Хорнеру) и `real` (чтобы не забыть о том, что `p:real`). Переменная `S` цикла – это индекс коэффициента полинома (`A[S]`), который прибавляется к текущему значению накапливаемой в переменной r суммы. Начальное значение `r` – это `A[n]` (`r:=A[n]`). Первым “прибавляемым” коэффициентом является `A[n-1]`, последним – `A[0]`, чем и объясняется начальное (`S:=n-1`) и конечное (`S:=0`) значения `S` в цикле.

Чем схема Хорнера лучше варианта, рассмотренного перед этим? В ней меньше арифметических операций. Посчитайте сами. А чем больше операций, тем больше накапливается погрешностей вычислений (неизбежных при операциях с числами типа `real`). В итоге - значения найденных корней могут существенно отличаться от их истинных значений. А это чревато… Поэтому в дальнейшем, когда это не будет оговорено отдельно, будем отдавать предпочтение схеме Хорнера.

4.2. Схема Хорнера – Руффини отделения вещественного корня

Суть идеи. Есть исследуемая функция , т.е. имеется алгоритм вычисления её значения при заданном значении аргумента . Задан диапазон поиска (конечная цель поиска – определение вещественного корня функции), ну, скажем, $X_n..X_k$. Разбиваем диапазон на m одинаковых (для удобства) отрезков, длина отрезка $h=(X_n-X_k)/m$. Если предположить, что на отрезке $X_n..X_k$ исследуемая функция имеет вещественный корень, то идя вдоль отрезка с шагом *h* можно надеяться, что на некотором шаге будет обнаружена перемена знака функции . Если функция непрерывна на отрезке, то имея две точки, например, X_{s-1} (где *x*=(s-1)*h*) и X_s (где *x*=s*h*), в которых функция имеет разные знаки, можно утверждать, что где-то между этими двумя точками функция принимает нулевое значение – *x* в этой точке является корнем функции. Определить такие X_{s-1} и X_s - это и означает отделить (локализовать) корень. Описанный алгоритм не гарантирует успеха, сканируя отрезок с шагом *h* можно "переступить" потенциальный корень , не заметив его (ну, во-первых, на заданном отрезке заданная функция может вообще не иметь корней – нельзя найти то, чего нет; во-вторых, корень может оказаться не один), особенно эффективно "прячутся" корни, если они группируются в виде близких по значению пар – в пределе при 2-кратной паре корней график функции вообще не пересекает оси x, а только касается в точке 2-кратного (в общем случае – n-кратного, где n-чётное) корня. Следовательно, нет ожидаемой перемены знака , на что так надеются Хорнер вместе с Руффини: Но… - не сидеть же сложа руки. Хоть какие-то попытки делать надо !

```
procedure HorRuf (A:Coef; Xn,Xk:real; m:integer;
                  var F:Boolean; var Xl,Xr :real);
var y, yp, h:real; s:integer;
begin
  h:=(Xk-Xn)/h; F:=false;
  yp:=HorReal(A,n); X:= Xn +h;
  repeat
    y:=HorReal(A,X);
```

```
    if y*yp<0 then
      begin
        Xl:=X-h; Xr:=X; F:=true
      end;
    yp:=y; X:=X+h;
  until (X>Xk) or F
end;
```

В списке параметров подпрограммы фигурирует `F:Boolean`. Это флажок, сигнализирующий об успехе (или провале) попытки локализовать (засечь) корень полинома, закодированного в массиве `A:Coef`. Итак, если после завершения "экспедиции" на матче подпрограммы гордо реет флаг `F:=true`, то – победа!

Если же `F:=false` – "пришёл невод с тиной морскою" (А.С. Пушкин). Об успехе в процессе поиска свидетельствует выполнение условия $y*yp<0$ (y на предыдущем шаге, т.е. *yp* отличается по знаку от *y* текущего). Компьютеру, возможно, накладнее вычислить произведение, чем просто сравнить знаки *yp* и *y* но нам программировать проще (пусть уж компьютер нам извинит такой эгоизм !). Запоминание `Xl` - левого и `Xr` - правого краев области локализации корня в случае успеха – понятно. Как объяснимо и вывешивание победного флага (чтобы не забыть потом, как это случилось с легендарным Тесеем, который, будучи расстроенным разлукой с Ариадной, забыл сменить цвет паруса с чёрного на белый – читатель знает ,чем это закончилось) - `F:=true`. Поиск прекращается либо при исчерпании диапазона поиска (очередной *x* уже за *xk*) либо когда искомое найдено.

Итак, возвращаемые процедурой `HorRuf` значения *xl*, *xr* имеют смысл только когда вместе с ними возвращается `F:=true`, в противном случае в них – "тина морская".

4.3.Уточнение локализованного корня методом половинного деления

В артиллерии есть такое понятие – "взять цель в вилку "(недолёт, перелёт – огонь на поражение). Результат работы предыдущего алгоритма именно в том и состоит , чтобы взять корень "в вилку". Способов добивания попавшего в

ловушку зверя – множество. Один из простейших (и эффективных!) – метод дихотомии или по-нашему – половинного деления.

Суть метода: исходная ситуация – на концах отрезка `Xl…Xr` исследуемая функция (в нашем случае полином) принимает значения противоположных знаков. Делим отрезок пополам (“половинное деление”) и определяем знак функции в середине отрезка ; в точке `X:=(Xl+Xr)/2`, если знак совпадает со знаком в точке `Xl`, то (очевидно, корня в левой половине отрезка нет) – переносим точку `Xl` в точку `X`, в противном случае - переносим `Xr` в `X`. В итоге ширина отрезка сужется вдвое. Этот суженный отрезок снова делим пополам – и так до тех пор пока ширина отрезка не станет меньше 2ε, где ε - допустимая погрешность в определении искомого корня. Середину последнего отрезка и принимаем за искомое значение корня. Программно это может выглядеть так

```
function Dych (A:Coef; Xl,Xr,Eps:real):real;
varX,F,Fl:real;
begin
  F:=HorReal(A,Xl);
  repeat
    Fl:=F; X:=(Xl+Xr)/2;
    F:=HorReal(A,X);
    if F*Fl>0 then Xl:=X
              else Xr:=X
  until Xl-Xr<=2*Eps;
  Dych:= (Xl+Xr)/2
end;
```

Здесь `Eps` – допустимая погрешность ε в определении корня. Но не следует надеяться ,что таким образом вы локализируете корень с погрешностью в действительности не превышающей `Eps`. Это было бы возможно в случае, если бы значения функции вычислялись точно, а на самом-то деле компьютер вычисляет их, хоть и весьма точно, но всё-таки с погрешностью (“результат компьютерных операций над числами типа `real` - непредсказуем”). Возможно, вы надеетесь, что “не так страшен волк, как его малюют” – компьютер, как-никак, достаточно серьёзный инструмент для вычислений. Возможно, что вы правы,

если вы ловите одинокого зверя, т.е. ищете изолированный вещественный корень, в окрестности которого не бродят его “собратья”, и в этой окрестности функция изменяется довольно “круто” (угол пересечения графика с осью в искомом корне относительно велик). В такой ситуации можно ожидать (надеяться), что на концах зоны $(X \pm \varepsilon)$ погрешность вычисления значения функции не будет превышать модуля этой самой функции. Но читатель мог обратить внимание на то, что абсолютная величина функции в алгоритме `Dych` не фигурирует. С одной стороны – это хорошо, но с другой стороны – это «расслабляет» исследователя. И он может не обратить внимание на необходимость подготовки полинома к определению его корней. Ну, в самом деле, корни полинома, как известно, не должны изменяться, если мы все коэффициенты умножим (или разделим) на какой-то (большой или малый) коэффициент. Вот значение полинома в процессе поиска корня от такого умножения (деления) изменится, и ещё как! А вот как при этом изменяется (или не изменяется?) погрешность вычисления этих самых значений полинома читатель знает? Ну, представьте, что вы поделили все коэффициенты полинома на миллион (или миллиард)! Скажется это на результате, который возвратит функция `Dych`? А знает ли читатель, что график, например, полинома $(p - X_k)^5$ в окрестности точки $-X_k$ очень пологий рис. 4.1?

Ширина зоны, в пределах которой даже знак значений полинома точно (гарантированно) не определяется, может быть сколько угодно шире заданной априори зоны $Xk \pm \varepsilon$. Вот и уточняйте после этого положение корня! Между тем, что мы хотим, и тем, что мы можем, может быть существенный разрыв. Поэтому выбирать требуемое ε и оценивать достигнутую точность локализации корня нужно здраво. Возможно, это легче сказать, чем сделать, но, по крайней мере, иметь это в виду – необходимо!

Есть более “утончённые” алгоритмы уточнения корня, например, метод хорд, метод касательных (Ньютона), но с учётом вышесказанного – стоит ли “овчинка выделки”?

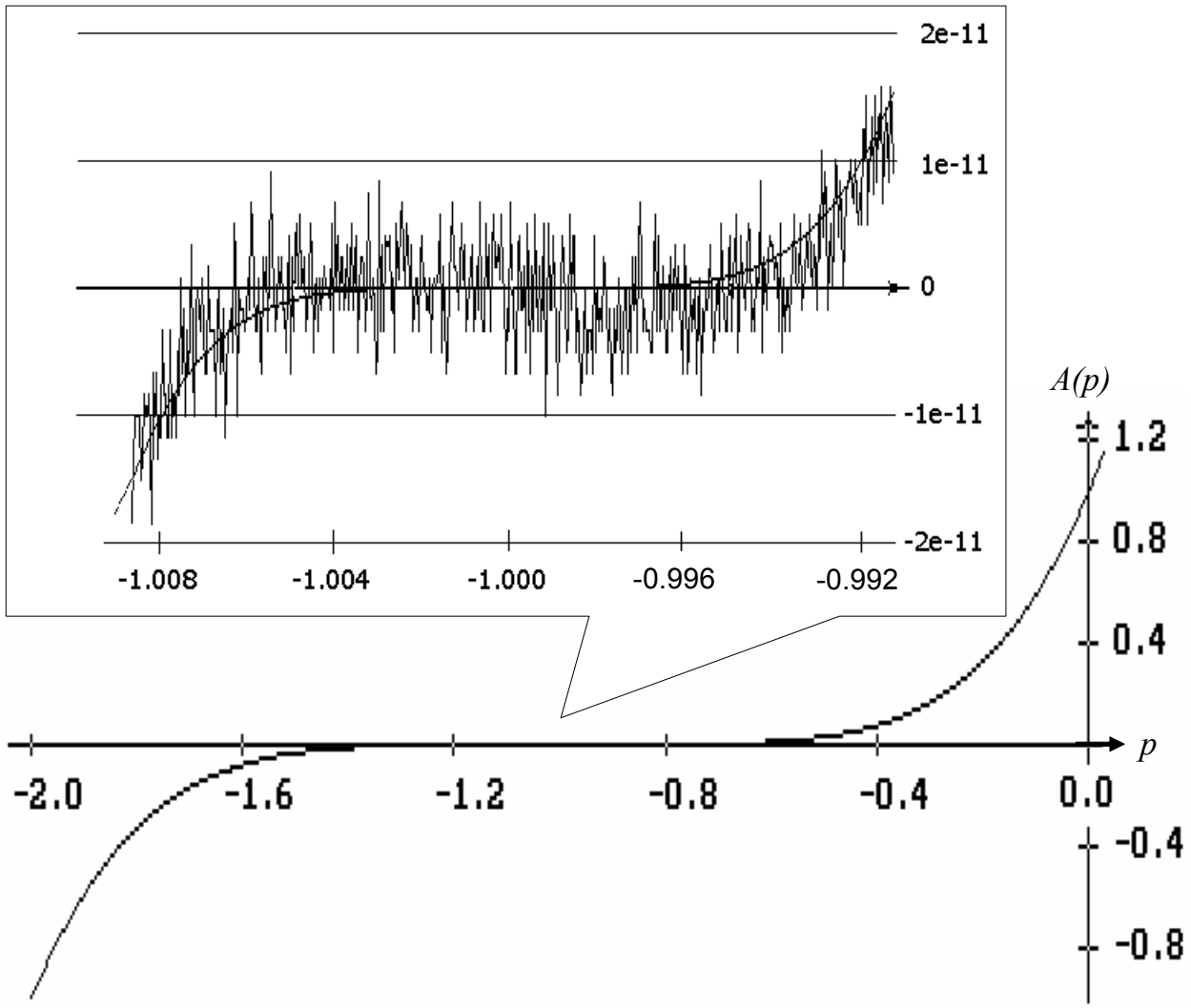


Рис.4.1. График полинома $A(p) = (p-1)^5$.

4.4. Вычисление значения полинома при комплексном значении аргумента

Ни построение графика полинома, ни метод Хорнера – Руффини вместе с методом половинного деления комплексных корней полинома, если таковые в полиноме имеются, нам определить не позволят. А при решении ряда задач наличие комплексных корней в соответствующих полиномах обязательно! И находить их обязательно! Так что от необходимости определять комплексные корни нам некуда деться! По крайней мере, вычислять значение полинома при комплексном значении аргумента – нужно! Итак, требуется вычислить значение полинома

$$A(p) = a_n p^n + a_{n-1} p^{n-1} + ... + a_1 p + a_0 ,$$

при $p=R+jI$, где R - вещественная, I – мнимая части аргумента Р.

Ранее мы рекламировали схему Хорнера для вычисления значения полинома. Почему бы её не применить и в этом случае?

Очередной «виток» цикла в схеме Хорнера можно обозначить так

$$\mathrm{Re}^{н} + j\,\mathrm{Im}^{н} = (\mathrm{Re}^{с} + j\,\mathrm{Im}^{с})(R + jI) + a_s, \qquad (4.4)$$

$\mathrm{Re}^{с} + j\,\mathrm{Im}^{с}$ - старое (полученное на предыдущем «витке») приближение значения полинома (при комплексном значении аргумента можно предположить, что и значение полинома тоже, в общем случае, будет комплексным). Это “старое” значение полинома множится на само значение аргумента (R+*j*I) и к полученному произведению добавляется очередной (S-ый) коэффициент полинома. Весь же процесс вычисления значения полинома можно представить процедурой Horcomp (метод Хорнера для комплексного аргумента).

```
procedure HorComp(A:Coef;R,J:real;var Re,Im :real);
var S,n,:integer;R1:real;
begin
  n:=round(A[-1]);Re:=A[n]; Im:=0;
  for S:=n-1 downto 0 do
    begin
      R1:=Re*R-Im*I+A[S];
      Im:=Re*I+Im*R;
      Re:=R1
    end
end;
```

Здесь мы пользуемся известным свойством оператора присваивания: в правой части его присутствует старое значение идентификатора, тогда как в левой части этот же идентификатор выступает уже в качестве своего обновлённого значения. Поэтому индексы “н” и “с” (новый – старый) в подпрограмме опущены, а «новое» значение `Re` во избежание утери его “старого” значения (необходимого для обновления `Im`) временно помещается во вспомогательное хранилище `R1`. После того, как необходимость в хранении отпадает, значение `R1` возвращается его “законному владельцу” `Re` (`Re:=R1`). Родство процедуры `Horcomp` с функцией `Horreal` просматривается достаточно чётко. А вот почему в последнем случае пришлось оформлять процесс в виде

процедуры – тоже понятно. Ведь возвращать надо не одно значение (с такой задачей может справиться и подпрограмма-функция), а два (`Re` и `Im`) – а это под силу процедуре. Заметили `var` перед `Re` и `Im:real` в заголовке процедуры? Это обязательство вернуть. Без этого var найденные значения `Re` и `Im` так бы и умерли, не родившись (в момент окончания работы процедуры).

4.5. Выделение круга, ограничивающего корни полинома

Чтобы целенаправленно искать корни полинома желательно было бы хотя бы ориентировочно представлять себе где они могут находиться. Такую "наводку" дают математики. Они утверждают, что корни полинома не выходят за пределы круга на комплексной плоскости с центром в её начале и радиусом R.

$$R = 1 + \frac{b}{|a_n|}, \tag{4.5}$$

где $b = \max(|a_0|, |a_1|, |a_2|, ..., |a_{n-1}|)$.

Вообще-то говоря, математики говорят о кольце, внешний радиус которого определяется (4.5), но мы не будем обращать внимание на существование «дырки» в кольце, т.к. по нашему мнению обход «дырки» в процессе поиска корней создаёт больше проблем, чем уменьшает зону поиска. Ну проверит компьютер несколько точек на комплексной плоскости, где заведомо нет корней – какая в том трагедия? Если корней там нет, то и обнаружить их там он (компьютер) не сможет, а небольшое увеличение работы для компьютера окупается упрощением программирования для нас с вами. Раньше программисты вынуждены были экономить машинное время (оно было дорогим, значительно дороже их собственного!), теперь ситуация изменилась на противоположную, поэтому сейчас ценится простота («прозрачность») алгоритма. Если процесс поиска отдельного корня полинома занимает у компьютера малые доли секунды, то что за беда, если это время увеличится на 10%? Реализуем формулу (4.5) программно

```
function Radius (A:Coef;):real;
```

```
var S,n:integer;b:real;
begin
  n:=round(A[-1]);b:=abs(A[0]);
  for S:=1 to n-1 do;
    if abs(A[S])>b then B:=abs(A[S]);
  Radius:=1+b/abs(A[n])
end;
```

Мы надеемся, что читатель видит угрозу деления на ноль в формуле (4.5), если степень полинома окажется завышенной. Так что не стоит предлагать функции `Radius` полином, формальная степень которого выше его фактической степени. К слову, формула (4.5) свидетельствует нам о том, что формально можно считать, что полином, кроме *n* «фактических» корней, расположенных в конечной области (внутри круга с радиусом `R`), имеет ещё бесчисленное множество «виртуальных» корней в бесконечности (при `A[n]=0` радиус `R` становится равным бесконечности). Но, к счастью, неучёт “виртуальных” корней нам ничем не грозит - и мы ими интересоваться не будем.

И ещё относительно ранее упоминавшейся “дырки”. Полином ведь может иметь и нулевой (нулевые) корень! А почему бы нет? Положите $a_0 = 0$, и один нулевой корень – налицо. В этом случае “дырка” - исчезает. Это ещё соображение в пользу её «игнорирования» при разработке общего алгоритма поиска корней.

4.6. Сканирование заданной области комплексной плоскости с целью поиска корней полинома

Корень полинома ещё называют его нулём, т.к. в корне полином обращается в ноль. Если же говорить о комплексном корне (а это общий случай), то в корне и вещественная Re и мнимая Im части значения полинома должны одновременно обращаться в нули. Чтобы не рассеивать внимание на двух «зайцев» введём в рассмотрение «интегрального зайца» - модуль значения полинома, равный корню квадратному из суммы квадратов Re и Im. И тогда в потенциальном корне этот модуль должен будет обращаться в ноль.

Таким образом, поиск корней полинома на комплексной плоскости можно организовать как поиск таких точек, где модуль данного полинома обращается в

нуль. Можно предложить множество алгоритмов решения этой задачи. Т.к. нуль – это абсолютный минимум значения модуля, то корни можно искать как точки абсолютных минимумов (все абсолютные минимумы для модулей в области, содержащей корни полинома, равны нулю). Мы ограничимся самым примитивным, но простым в программном отношении (а это немаловажное достоинство!) методом поиска, а именно – перебором (сканированием) заданной области с некоторым фиксированным шагом. Для простоты используем одно и то же значение шага `h` и по вертикали и по горизонтали. Тем самым мы как бы покрываем исследуемую область (участок комплексной плоскости) квадратной сеткой (с размером ячейки `h*h`), вычисляем в узлах сетки значение полинома, определяем модуль этого значения и выбираем тот узел, где модуль значения полинома будет наименьшим. Рассчитывать на то, что в каком-то из рассматриваемых узлов модуль полинома окажется точно равным нулю – неконструктивно (во-первых, вычисляемое значение модуля будет определяться неточно, а во-вторых с какой это стати узел сетки обязан угодить точно на корень?). Да и вообще, как мы уже отмечали ранее, по значению полинома (и его модуля в частности) весьма трудно оценить расстояние от рассматриваемой точки до корня. Если бы модуль точно равнялся нулю, тогда, конечно… Но нуль с погрешностью может оказаться и не нулём вовсе! Так что же делать? А не нужно требовать абсолюта! Будем для начала просто искать узел, где модуль полинома принимает минимальное значение. Найденная таким образом точка (узел) и будет наилучшим претендентом на роль искомого корня. Достаточно ли хорош этот претендент – это уже другой вопрос. Судить о достоинствах по величине претендента – какую величину модуля можно считать достаточно (допустимо) малой? Проще судить по величине ячейки в сетке, чем меньше ячейка, тем вероятнее приближение лучшего претендента к истинному корню. Но и количество узелков в такой сетке будет возрастать пропорционально квадрату числа узлов в одном направлении. В тысячу раз уменьшите шаг – в миллион раз возрастёт число узлов. Не станет ли перебор такого количества узелков обременительным, даже для компьютера? Думайте, а мы пока что выполним

грубую прикидку

```
procedure KorPolSc(A:Coef;k:integer;Eps:real;
                   var Rk,Ik:real);
var z,S:integer;
    R,J,Re,Im,M2,MinMod,Rad,Rc,Ic:real;
begin
  Rad:=Radius(A);Rk:=0;Jk:=0;
  MinMod:=1e25; Rc:=0; Jc:=0; H:=Rad/k;
  repeat
    for z:=-k to k do
      begin
        I:=Ic+z*h;
        for S:=-k to k do
          begin
            R:=Rc+S*h: HorComp(A,R,I,Re,Im);
            M2:=sqr(Re)+sqr(Im);
            if M2<MinMod then
              begin
                Rk:=R; Ik:=I;
                MinMod:=M2
              end
          end;
      end;
    if h>Eps then
      begin
       Rc:=Rk; Ic:=Ik; h:=h/k
      end
  until h<Eps
end;
```

Здесь `Eps` – допустимая погрешность в определении искомого корня. Раньше мы уже обсуждали условность этого термина. Скажем так, здесь `Eps` – это тот шаг сетки, дробить который дальше считается нецелесообразным. `k` – это количество шагов (влево, а затем вправо) от точки с координатами Rc, Ic, принимаемой за центр области сканирования. Вначале за такой центр принимается начало координат (центр круга, охватывающего все корни исследуемого полинома, откуда `Rc:=0; Ic:=0`). Шаг сканирования на первом этапе принимает `h:=Rad/k`, где `Rad` – радиус пресловутого круга. Тем самым, делая `k` шагов вниз, `k` шагов вверх от центра (это в цикле по `z`) и на каждом `z-k`

шагов влево, затем k шагов вправо, мы перекрываем сеткой квадрат, описанный вокруг круга с радиусом Rad (с запасом, чтобы не отвлекаться на отсев лишних узлов). В процессе сканирования (узелок за узелком) в каждом узле вычисляется Re, Im (процедурой HorComp) и затем Modul:=sqr(Re)+sqr(Im). Вообще-то это не сам модуль, а его квадрат. Мы решили сэкономить на операции Sqrt. В самом деле, где минимум модуля полинома там и минимум корня квадратного из него. Поэтому мы обозначили его как M2. Тогда и величина MinMod, с которой сравнивается M2 - это минимальное значение квадрата модуля. До того, как начать сканирование, мы предположили, что в процессе найдётся хоть одно M2 меньше первоначально принятого (MinMod:=1e25). Или, может, стоило взять побольше?

Так или иначе, когда в очередном узле M2 окажется меньшим текущего значение MinMod, то это событие заносится в "книгу рекордов"

```
(if M2<MinMod then
  begin
    Rk:=R;Ik:=I: MinMod:=M2
  end)
```

После того как «просмотр» области сканирования завершён, проверяем с каким шагом это сканирование было произведено. Если шаг слишком велик (h>Eps), то найденную наиболее перспективную точку (узел), принимаем за центр квадрата со сторонами 2h*2h (по шагу h влево–вправо и вверх–вниз от центра), шаг h уменьшается в k раз и процесс сканирования этого редуцированного квадрата повторяется по уже описанной "технологии".

Если же шаг признан удовлетворительным, то найденная наиболее перспективная точка (Rk+jIk) принимается за искомый корень.

Выбирая по своему усмотрению k и Eps пользователь может регулировать «надёжность» процесса поиска (если k слишком мало – можно «проморгать» корень, т.е. переступить его не заметив). Увеличение же k как и уменьшение Eps увеличивает трудоёмкость (время поиска). Увлекаться слишком малым Eps – это

как правило, обманывать самих себя (и заказчика!). Но – решение за пользователем.

Если бы искомый корень был один – то процедура `KorPolSc` решала бы задачу исчерпывающим образом. Но кто же станет привлекать такую "махину" для решения уравнения

$$a_1 p + a_0 = 0 ?$$

И даже для полинома второй степени рассматриваемый алгоритм вряд ли оправдан. Хотя – как сказать. Если алгоритм в компьютере, то какое вам дело – пусть пашет.

Но, в принципе, если корней много (n в полиноме n-ой степени), то который из них найдёт `KorPolSc`? А кто его знает? Какой попадётся. В общем случае – это дело случая. Но Бог с ним – нумеровать корни можно в любом порядке, считайте что найденный корень – первый. А вот как насчёт остальных (n-1) корней? Их как искать? Самый простой способ – исключить найденный корень из исследуемого полинома, чтобы он не мешал искать остальные корни. На механизме исключения корня (пары корней, если найденный корень окажется комплексным) и на том, куда "складировать" корни по мере их «поимки» мы остановимся, когда мы будем рассматривать операции с двумерными массивами.

4.7. Критерий Гурвица-Рауса

Нарушим естественную последовательность изложения материала. Торжественно обещаем обосновать вариант реализации алгоритма Гурвица-Рауса в разделе, посвящённом рассмотрению двумерных массивов. Смысл такого «перескока» в том, что результатом обещаемого рассмотрения будет сведение алгоритма к элементарным действиям над одномерным массивам, а именно, над массивом типа `Coef`, несущим информацию об исследуемом полиноме. А суть критерия Гурвица-Рауса состоит в выяснении имеет ли исследуемый полином корни в правой половине комплексной плоскости. На первый взгляд – не бог весть какая информация, но не спешите с выводами. Для проблемы поиска корней – появляется возможность вдвое уменьшить зону поиска (по крайней

мере, на первом этапе, если настаивать на только что рассмотренном алгоритме). Но, как знать, может этот самый критерий откроет нам путь к другим заслуживающим интереса алгоритмам? Не отвлекаясь сейчас на формулировку критерия, ограничимся констатацией факта, что возможно существование подпрограммы-функции `Hurvic`, заголовок которой выглядит следующим образом

```
function Hurvic (A:Coef):integer;
```

Эта функция возвращает (в идентификаторе `Hurvic`) значение `+1`, если в полиноме, представляемом массивом `A:Coef`, корней в правой половине комплексной плоскости нет. Соответственно ответ `−1` означает, что в правой полуплоскости имеются в наличии (ну, хотя бы один!) корни данного полинома. Чтобы рассеять сомнения читателя – приведём эту подпрограмму (а от подробного комментирования в данном параграфе воздержимся).

```
function Hurvic (A:Coef):integer;
const Eps=1e-9;
var z,S,Hurv,n:integer;
    R:real;
begin
  Hurv:=1;N:=round(A[-1]);
  if n>0 then
    while (abs(a[n])<Eps) and (n>0) do dec(n);
      if n>0 then
        begin
          if A[n]<0 then
            for z:=0 to n do A[z]:=-A[z];
              for z:=0 to n-1 do
                if A[z]<=Eps then Hurv:=1;
          if (Hurv=1) and (n>2) then
            for z:=1 to n-2 do
              begin
                if A[n-z]<=Eps
                  then Hurv:=-1
                  else
                    begin
                      R:=A[n-z+1]/A[n-z]
                      S:=N-z;
                      while s>=2 do
```

```
                        begin
                          A[s-1]:=A[s-1]-A[s-2]*R;
                          dec(S,2)
                        end
                    end
                end;
            if A[1]<=Eps then Hurv:=-1
        end;
  Hurvic:=Hurv
end;
```

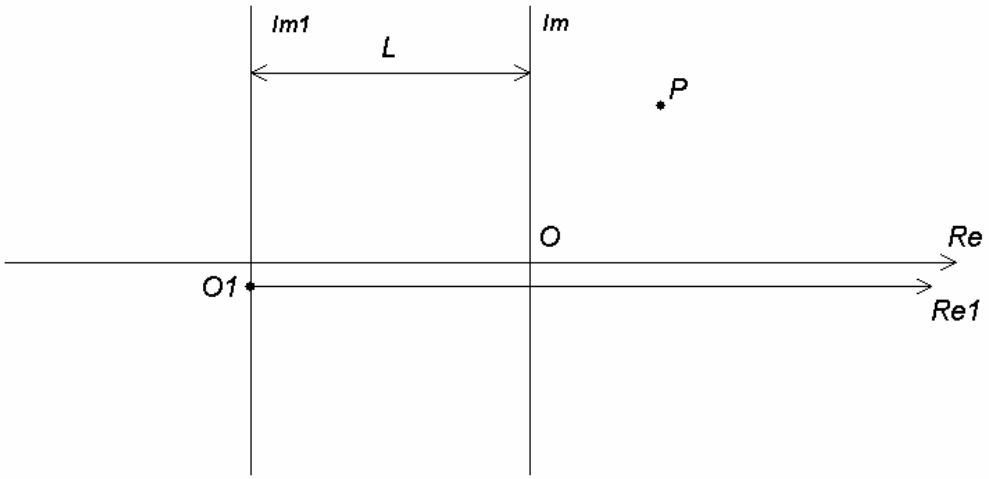
4.8. Определение степени устойчивости

Критерий Гурвица, задекларированный в предыдущем параграфе применяется для исследования устойчивости динамических систем, в частности, систем автоматического управления. В теории автоматического управления показывается, что динамическая система будет устойчива, если все корни её характеристического полинома будут находиться в левой половине комплексной плоскости. Таким образом, если предметом рассмотрения является именно характеристический полином, то если `function Hurvic` возвращает значение +1, то это свидетельствует об устойчивости исследуемой системы, в противном случае (ответ –1) – система неустойчива. В этом смысле иногда говорят об устойчивости полинома (полином устойчив, если он не имеет корней в правой полуплоскости). Это, естественно, условность, т.к. в общем случае конкретный полином может и не иметь никакого отношения к динамической системе, а сам по себе полином – как он может быть устойчивым или не устойчивым? Но если применять критерий Гурвица-Рауса к полиному, то можно говорить не только о его «устойчивости», но, более того, можно определять степень его устойчивости.

Степень устойчивости для полинома – это расстояние от мнимой оси плоскости корней до самого правого из корней полинома. Степень устойчивости считается положительной, если этот самый правый корень (или группа корней, находящихся на одной вертикали – с общей вещественной частью) находится левее мнимой оси (полином устойчив). Если же самый правый корень – правее мнимой оси, то степень устойчивости считается отрицательной (система – то ведь неустойчива!). Мы сообщаем читателю сколько сведений о вещах, которые на

первый взгляд могут показаться надуманными, но для студента, готовящегося стать специалистом в области автоматизации – это профессиональная информация – и она будет востребована! Итак, попытаемся определить эту самую степень устойчивости для произвольного полинома A(p).

В п.4.6. при поиске корня методом сканирования мы уже прибегали к услугам функции `Radius` для определения радиуса круга, охватывающего все корни полинома (п.4.5.). Если искать положение вертикальной (параллельной мнимой оси комплексной плоскости) прямой, проходящей через самый правый корень полинома A(p), то, очевидно, эта прямая не может быть левее, чем на R и не правее, чем на R от мнимой оси, где `R:=Radius(A)`. Итак, примем `Pmin:=-R` и `Pmax:=R`. Найдём среднюю точку между `Pmin` и `Pmax`, а именно `P:=(Pmin+Pmax)/2` и сместим мнимую ось плоскости корней в эту точку (замена p на $p_1 - L$ смещает мнимую ось влево на L, если $L>0$).



Одна и та же точка в системе координат Re, Im будет обозначаться как p, тогда как в системе координат $\mathrm{Re}_1, \mathrm{Im}_1$ (сдвинутой влево на L относительно системы Re, Im) – это будет p_1. И поскольку $p = p_1 - L$, то $p_1 = p + L$, что наглядно видно на приведённом эскизе.

Смещение мнимой оси “влево” на p реализуется вызовом `ZamPPL(A,-P,B)`. Если теперь проанализировать устойчивость «смещённого» полинома $B(p_1)$ (с помощью критерия Гурвица, естественно), то если $B(p_1)$

окажется устойчивым, то левую границу зоны поиска Pmin можно перенести в точку *p*, в противном случае переносим в *p* правую границу, т.е. Pmax. В итоге зона поиска сужается вдвое (чем не половинное деление?). Полученную (суженную) зону снова делим пополам – и так до тех пор, пока ширина суженной зоны не станет меньше или равна `2*Eps`, где `Eps` – допустимая погрешность в определении искомой степени устойчивости. Об известной условности такого определения Eps мы уже достаточно много говорили в п.4.6. Зону, ширина которой не превышает `2*Eps` ещё раз (последний раз!) делим пополам – и полученную точку *p* принимаем за искомое значение степени устойчивости (не забудем только, что отрицательному *p* соответствует положительная степень устойчивости, и наоборот).

```
function StepUst( A:Coef;Eps:real):real;
var Pmin,Pmax:integer;P:real;
    B:Coef;
begin
  Pmax:=Radius(A,Eps); Pmin:=-Pmax;
  P:=0;
  while Pmax-Pmin>2*Eps do
    begin
      ZamPPL(A,-P,B);
      if Hurvic(B)=1 then Pmax:=P
      Pmin:=P;
      P:=(Pmin+Pmax)/2
    end;
  StepUST:=-P
end;
```

4.9. Определение самого правого корня полинома

Итак, только что сгенерированная подпрограмма StepUst может локализовать вертикаль на комплексной плоскости, в окрестности которой находится самый правый корень полинома (а может несколько корней). Если исследуемый полином имеет вещественные коэффициенты, то появление комплексного (с не нулевой мнимой частью) корня будет обязательно сопровождаться корнем-двойником (комплексные корни могут появляться только в виде попарно-сопряженных пар). Для локализации такой пары достаточно

определить один (неважно – верхний или нижний) её компонент. Поэтому при поиске самого правого корня можно ограничиться сканированием упомянутой выше вертикали начиная от $I = 0$ до $I = Iv$, где $Iv = \sqrt{Rad^2 - R^2}$. Это высота катета при условии, что другой катет равен R, а гипотенуза равна `Rad`. Здесь `R:=-StepUst(A,Eps)`, а `Rad` – радиус круга `Rad:=Radius(A)`. Первоначально шаг сканирования принимается равным `H:=Iv` (это пока не сканировали), а потом он уменьшается в k раз (k – локальная константа, например, k = 10). На каждом шаге сканирования вычисляется значение полинома Re+jIm с помощью подпрограммы `HorComp`, определяется квадрат модуля значения полинома (`Modul:=sqr(Re)+sqr(Im)`) и запоминается точка с минимальным значения `Modul`'я. Вокруг этой точки формируется суженная зона поиска (нижняя граница – на шаг ниже найденной точки, верхняя – на шаг выше). Шаг в очередной раз уменьшается в k раз – и так до тех пор, пока не будет найдена (с шагом `H<=Eps`) точка, где наблюдается минимальное значение модуля значения полинома. Эта точка и принимается за искомое значение корня. Описанный процесс оформим в виде подпрограммы `KorPolSU`

```
procedure KorPolSU (A:Coef;Eps:real; var R,I:real);
const K=10;
var Re,Im,Modul,I0,H,Iv,Ini,Rad,Min:real;
begin
  Rad:=Radius(A);R:=-StepUst(A,Eps);
  Iv:=sqrt(sqr(Rad)-sqr(R);
  Ini:=0; H:=Iv; Min:=1e25, I0:=0;
  While H>Eps do
    begin
      I:=Ini; H:=H/k;
      While I<Iv do
        begin
          HorComp(A,R,I,Re,Im);
          Modul:=sqrt(Re)+sqr(Im);
          If Modul<Min then
            begin
              I0:=I;Min:=Modul
            end;
          I:=J+H
```

```
        end;
      Iv:=I0+H; Im:=I0-H
    end;
  I:=I0
end;
```

Здесь `R, I` – вещественная и мнимая части искомого корня (самого правого). `Iv, Ini` – верхняя и нижняя границы зоны поиска (постоянно суживается в процессе поиска). Почему `Ini` не обозначена, например, как `In`? А дело в том, что идентификатор `In` зарезервирован (используется при определении принадлежности элемента к множеству).

`I0` – текущее "лучшее" значение мнимой части искомого корня. Значение I в I0 заносится тогда, когда обнаруживается улучшение текущего значения модуля полинома (когда `Modul<Min`).

`I0`, полученная в конце процесса поиска, и принимается за искомое значение I (подлежащее возврату).

4.10. Определение корней полинома, степень которого не выше двух

Корни полиномов первой и второй степени, как известно, можно без проблем определить по известным формулам.

Для *n*=1 (полином $a_1 p + a_0$) это

$$p_1 = -\frac{a_0}{a_1}, \tag{4.6}$$

Для *n*=2

$$p_{1,2} = \begin{cases} \dfrac{-a_1 \pm \sqrt{a_1{}^2 - 4a_2a_0}}{2a_2}, \text{если } a_1{}^2 - 4a_2a_0 \geq 0, \\ -\dfrac{a_1}{2a_2} \pm j\dfrac{\sqrt{4a_2a_0 - a_1{}^2}}{2a_2}, \text{если } a_1{}^2 - 4a_2a_0 < 0. \end{cases}, \tag{4.7.}$$

Если исходный полином имеет степень равную 1 или 2, то проблем с программированием можно избежать решая их по формулам (4.6.) или (4.7.). Более интересен вариант, когда исходный полином имеет степень n>2. Первый

вопрос, который возникает при этом. Если корней много, то где и как их размещать (хранить)? Можно предложить бездну вариантов, например, R и I – вещественные и мнимые части корней размещать не в переменных типа real, а в массивах, ну хотя бы типа `Coef`, где `R[S]` и `I[S]`, $1 \le s \le n$ – вещественная и мнимая части *s*-го корня. В ячейке `R[-1]` можно записывать `k` – количество уже найденных (в процессе поиска) или *n* – общее количество корней данного полинома. Но эти два массива нужно всегда таскать вместе, как одно целое (ведь один без другого, как правило, теряет право на существование)! Неудобно это. А что, если эти два массива "склеить", например, вводя такое определение типа

```
type Kor=array[1..2, -1..30] of real.
```

Это массив на две строки (с номерами строк 1 и 2), а столбики в нём нумеруются от –1 до 30. Схематически корни полинома степени n в массиве типа Kor будут выглядеть так

		-1	0	1	2	3	…	*n*	…	30
	1	*n*	-	R_1	R_2	R_3	…	R_n	…	
Korni	2	-	-	I_1	I_2	I_3	…	I_n	…	

Здесь `Korni[1, S]` – вещественная часть `S`-го корня, `Korni[2, S]` – мнимая часть `S`-го корня. Если какие-то (а хотя бы и все!) корни – вещественные, то нет проблем – соответствующая мнимая часть будет равна нулю – только и всего! Ячейки `Korni[1,0], Korni[2,-1], Korni[2,0]` – пустые. Можно было их "сэкономить", начав нумерацию столбцов с 0, например, но назначать номер столбца, равный –1 для размещения "бирки" - это, вроде, у нас уже стало традицией. Поскольку на чём-то всё-таки необходимо остановиться, то и остановимся на приведенном варианте типа Kor. Это уже двумерный массив. Во втором измерении в нём всего-то две строки, но нам больше пока и не нужно.

И как мы уже раньше предлагали, будем заносить каждый очередной найденный корень полинома под его номером (номер столбца) в массив Korni, параллельно корректируя содержимое ячейки `Korni[1,-1]`.

“Оприходованный” таким образом корень будем исключать из исследуемого полинома (чтобы не мешал искать оставшиеся корни). На самой технологии исключения остановимся подробнее в следующем параграфе. А сейчас только обратим внимание на то, что после исключения корня (корней) исходный полином редуцируется (его степень понижается) – и когда, например, степень полинома снизится до одного или двух, то можно отказаться от мощного (универсального) алгоритма, изложенного в п.4.9, и “добить” ослабленный полином “из рогатки” - по формулам (4.6.) или (4.7.). В роли терминатора планируется использовать процедуру `Kvadrat`, задачей которой является “доукомплектование” массива `Korni`: Kor с уже найденными *m* корнями какого-то исходного полинома, за счёт корней “остаточного” полинома A(*p*), который передаётся процедуре `Kvadrat` в качестве параметра (в виде одномерного массива типа `Coef`). Важно только гарантировать, чтобы степень остаточного полинома A(p) была равна 1 или 2 (не больше, но и не меньше).

```
procedure Kvadrat (A:Coef;var Korni:Kor);
var D,Z,R,I:real;
    m,n:integer;
begin
  n:round(A[-1]); m:=round(Korni[1,-1]);
  case n of
    1: begin
         inc(m);Korni[1,m]:=-A[0]/A[1];
         Korni [2,m]:=0;
       end;
    2: begin
         D:=sqr(A[1])-4*A[0]*A[2];
         Z:=2*A[2];
         if D>=0
           then
             begin
               inc(m); Korni[1,m]:=(-A[1]+sqrt(D))/Z;
               Korni[2,m]:=0; inc(m);
               Korni[1,m]:=(-A[1]-sqrt(D))/Z;
               Korni[2,m]:=0
             end
           else
             begin
```

```
                R:=-A[1]/Z; I:=sqrt(-D)/abs(Z);
                inc(m);Korni[1,m]:=R; Korni[2,m]:=I;
                inc(m);Korni[1,m]:=R; Korni[2,m]:=-I;
              end
        end
  end;
  Korni[1,-1]:=m
end;
```

Тут, как легко видеть, `D` – дискриминант квадратного полинома; `Z` – знаменатель в формулах (4.7.). Остальное очевидно.

4.11. Нахождение полного комплекта корней полинома

Как найти отдельный корень (пару комплексно спряженных корней) – мы рассмотрели (даже в нескольких вариантах). О необходимости исключить найденный корень из рассматриваемого полинома (чтобы он не мешал искать остальные корни), тоже упоминали. Остановимся на последнем вопросе подробнее. Если p_s, $1 \le s \le n$ – корни полинома (не важно – вещественные они или комплексные), то полином

$$A(p) = a_n p^n + a_{n-1} p^{n-1} + ... + a_1 p + a_0 . \qquad (4.8)$$

можно представить в виде

$$A(p) = a_n (p - p_1)(p - p_2)...(p - p_n) . \qquad (4.9)$$

Если p равно одному из корней (не важно – какому по номеру) соответствующая скобка в формуле (4.9) обращается в 0, а вместе с ней становится равным нулю и значение полинома. Пусть для определённости речь идёт о корне p_1 (нумерация корней – произвольная). Тогда, если полином A(p) в формуле (4.9) поделить на $(p - p_1)$, то получим

$$B(p) = \frac{A(p)}{p - p_1} = a_n (p - p_2)(p - p_3)...(p - p_n) \qquad (4.10)$$

Полином $B(p)$ из (4.10) имеет корнями $p_2, p_3, ..., p_n$, т.е. имеет тот же набор корней (за исключением p_1), что и полином (4.9), а значит, и (4.8). Итак, делением $A(p)$ на $(p - p_1)$ мы исключили из $A(p)$ корень p_1. Оставшиеся корни

можно искать в полиноме $B(p)$. Кроме того, что $B(p)$ не имеет корня p_1, степень полинома $B(p)$ будет ниже степени исходного полинома $A(p)$, что тоже – плюс. Если полином $B(p)$ "переименовать" в полином $A(p)$, то для дальнейшего поиска (и последующего исключения найденных) корней можно применить тот же алгоритм, что и для исходного полинома $A(p)$.

А что делать, если p_1 в (4.8) – комплексный, скажем, равный $p_1 = R + jI$? Делить на $(p - p_1) = (p - R - jI)$? В принципе, можно. Выше были рассмотрены алгоритмы операций над полиномами с вещественными коэффициентами. Совершенно не сложно модифицировать их так, чтобы можно было оперировать полиномами с комплексными коэффициентами. Не верите – попробуйте! Но если немножко подумать, то можно обойти возникшее затруднение. Вспомним, что полином с вещественными коэффициентами (а таковым предполагается исходный полином (4.8)) может иметь комплексный корень только в комплекте с его "двойником" - сопряжённым ему корнем. Итак, если есть корень $p_1 = R + jI$, то обязательно есть и $p_2 = R - jI$ (для удобства сопряжённые корни будем располагать рядышком). И исключать из (4.9), в этом случае удобнее не каждый из корней пары в отдельности, а оба сразу, т.е. делить $A(p)$ на

$$(p - p_1)(p - p_2) = (p - R - jI)(p - R + jI).$$

Правую часть приведённого выражения можно представить так (раскрывая скобки)

$$(p - p_1)(p - p_2) = (p - R)^2 + I^2 = p^2 - 2Rp + R^2 + I^2$$

Это полином второй степени, у которого

$$e_2 = 1, \quad e_1 = -2R, \quad e_0 = R^2 + I^2.$$

А при исключении вещественного корня приходилось делить на полином первой степени, у которого $e_1 = 1, \quad e_0 = -p_1$.

С учётом сказанного выше формирование массива корней (`Korni:Kor`) полинома $A(p)$ можно оформить в виде процедуры `KorPolMS`.

```
procedure KorPolMS (A:Coef;Eps:real; var Korni:Kor);
```

```
var m,n:integer;
    R,I:real;
    E:Coef;
begin
  RedPol(Eps,A);n:=round(A[-1]);m:=0;
  while n>2 do
    begin
      KorPolSu(A,Eps,R,I); inc(m);
      Korni [1,m]:=R; Korni[2,m]:=abs(I);
      if abs(I)<=Eps
        then
          begin
            E[-1]:=1; E[0]:=-R; E[1]:=1;
            dec(n)
          end
        else
          begin
            E[-1]:=2; E[0]:=sqr(R)+sqr(I);
            E[1]:=-2*R;E[2]:=1; inc(m);
            Korni[1,m]:=R; Korni[2,m]:=-abs(I);
            dec(n,2)
          end;
      DelPol(A,E,A,E)
    end;
  Korni[1,-1]:=m; Kvadrat(A,Korni)
end;
```

Насколько известно авторам, алгоритм определения корней, реализованный в `KorPolMS`, не излагается ни в каких книгах, кроме книг самих авторов. Авторы не утверждают, что изложенный алгоритм – лучше всех существующих. Но у него есть преимущество, которое авторы считают существенным. Определение корней в этом алгоритме осуществляется «справа-налево». Применительно к характеристическим полиномам устойчивых динамических систем это означает, что первыми определяются самые правые корни. А чем раньше определяется корень, тем меньше накапливается погрешностей счёта – тем точнее значение корня. Наибольшее влияние на динамические свойства системы оказывают именно самые правые корни (они ближе всего к мнимой оси – “особы приближённые к императору”). Именно поэтому важно вычислять эти “VIP-корни” с максимально доступной точностью. Мы уже и раньше обращали

внимание читателя на параметр `Eps` – рекомендуем вспомнить, а при практической работе – при нахождении корней полиномов высоких степеней на компьютере – самостоятельно проверить обоснованность скептицизма авторов.

5. Работа с двумерными массивами

Двумерные массивы (таблицы) в Турбо Паскале описываются таким образом (как переменные)

`A: array [Zn..Zk , Sn..Sk] of` < тип элемента массива >.

Здесь `Zn..Zk` – начальный и конечный номера строк, `Sn..Sk` – начальный и конечный номера столбцов.

Схематически такой массив можно представить себе так

		$S=S_n$	S_n+1	S_n+2	S_n+3	...	S_k
	$Z=Z_n$						
	Z_n+1						
A	Z_n+2						
	...						
	Z_k						

Элемент в `Z` – ой строке и `S` – ом столбце обозначается как `A[Z,S]` (так к нему принято обращаться как при считывании так и при записи).

Массивы бывают разных размеров и размерностей, так кроме дву-мерных, могут быть трех –, четырех – т.д. мерные массивы, в них могут размещаться данные различной физической и математической природы (см. < тип элемента массива >). Для упорядочения работы с такими массивами их удобно относить к тем или иным типам (типизировать).

Введем, например, такой тип

```
Const n=15; m=16;
type Matr=array[1..n,-1..m] of real;
```
(5.1)

Это матрица, имеющая n строк и m+2 столбцов, где n и m – некоторые константы. Договоримся, что в ячейке с адресом `[1,-1]` хранится n, а в ячейке `[2,-1]` значения m. Тогда, однотипные массивы могут описываться так

```
A,B,C: Matr;
```

Помимо того, что подобная типизация позволяет использовать массивы как

параметры подпрограммы (а это чрезвычайно существенно с точки зрения возможностей структурирования программных продуктов), подобная фиксация однотипности позволяет в Турбо Паскале использовать такое, например, присваивание как

```
A:=C
```

когда массив C копируется в массив A. Это удобнее, чем переносить ячейку за ячейкой. Договоримся в дальнейшем считать в массивах типа Matr рабочими столбцы с номерами от 1 до *m*.

5.1. Локализация минимума в двумерном массиве

Алгоритм локализации экстремума (для конкретности рассматриваем минимум) в двумерном массиве, в принципе, ничем не отличается от алгоритма аналогичной задачи для одномерного массива. Нужно перебрать все элементы массива и запомнить (возвратить) адрес “самого достойного претендента”. Ну, например, так

```
Procedure LocMin2 (A: Matr; var Zm, Sm: integer);
var n, m, Z, S: integer; Min: real;
begin
  n:=round(A[1,-1]); m:=round(A[2,-1]);
  Zm:=1; Sm:=1; Min:=A[1,1];
  for Z:=1 to n do
    for S:=1 to m do
      if A[Z,S]<Min then
        begin
          Min:=A[Z,S]; Zm:=Z; Sm:=S
        end
end;
```

Для начала назначаем “исполняющей обязанности” минимальной ячейку A[1,1] и запоминаем ее адрес (Zm:=1; Sm:=1)

Затем строка за строкой (в цикле по Z) поэлементно (в цикле по S) перебираем элементы массива, сравнивая их по величине с текущим претендентом. Как только обнаруживаем элемент “более достойный” – тут же ставим его на место предыдущего претендента (и запоминаем адрес вновь

назначенного). После завершения перебора “кресло лидера” займет, действительно самый достойный, а последний запомненный (в `Zm` и `Sm`) адрес и будет искомым.

При необходимости можно было бы возвратить и значения `Min`. Для этого достаточно перенести описания `Min:real` из описательной части процедуры в список параметров процедуры (поставив перед ним слово `var`).

5.2. Перестановка строк и столбцов в двумерном массиве

Пусть имеется массив `A:Matr` – такой, как рассмотренный в предыдущем параграфе. Необходимо поменять местами строки с номерами k и l ($1 \le k \le n$, $1 \le l \le n$). Будем переставлять их поэлементно с использованием “третьего стаканчика”.

```
Procedure PerStr (var A: Matr; k, L: integer);
var S, m: integer; r: real;
begin
  m:=round(A[2,-1]);
  for S:=1 to m do
    begin
      r:=A[k,S]; A[k,S]:=A[L,S]; A[L,S]:=r
    end
end;
```

Если бы k и l обозначали не номера переставляемых строк, а, например, номера переставляемых столбцов, то решение выглядело бы так

```
Procedure PerStolb (var A: Matr; k, L: integer);
var Z, n: integer; r: real;
begin
  n:=round(A[1,-1]);
  for Z:=1 to n do
    begin
      r:=A[Z,k]; A[Z,k]:=A[Z,L]; A[Z,L]:=r
    end
end;
```

5.3. Перезапись двумерного массива в одномерный

Пусть, например, требуется строку за строкой перенести информацию из массива A:Matr в массив B:CoefL (естественно, необходимо позаботиться о том, чтобы в массиве B хватило места !).

Решение такой задачи может быть таким

```
Procedure Transf21 (A: Matr; var B: CoefL);
var n,m,Z,S,u:integer;
begin
  n:=round(A[1,-1]); m:=round(A[2,-1]);
  u:=0;
  for Z:=1 to n do
    for S:=1 to m do
      begin
        B[u]:=A[Z,S]; inc(u)
      end;
    B[-1]:=u-1
end;
```

Здесь мы хотели бы обратить внимание читателя на счетчик u заполняемых ячеек в массиве B. Мы начали его заполнять с ячейки с индексом 0 (u:=0). Далее этот счетчик наращивается на единицу (inc(u)), при записи в массив B каждого следующего числа. Можно было бы искать формулу, связывающую между собой адрес переносимого элемента (из массива A) с соответствующим ему адресом в массиве B – и тогда надобность в счетчике отпала бы. Но стоит ли? Ведь действует счетчик так просто. А формула в случае, когда начальный номер столбца в массиве A не единица – усложняется…

Как организовать перезапись столбца за столбцом или переписывать одномерный массив в двумерный предлагаем придумать читателю самостоятельно.

5.4. Решение систем линейных алгебраических уравнений методом Гаусса

Итак, требуется разработать компьютерный алгоритм решения систем уравнений вида

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + ... + a_{1,n}x_n = a_{1,n+1}, \\ a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + ... + a_{2,n}x_n = a_{2,n+1}, \\ a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 + ... + a_{3,n}x_n = a_{3,n+1}, \\ ..... \\ a_{n,1}x_1 + a_{n,2}x_2 + a_{n,3}x_3 + ... + a_{n,n}x_n = a_{n,n+1}. \end{cases} \quad (5.2)$$

Здесь $x_1, x_2, x_3, ..., x_n$ – неизвестные, подлежащие определению. В качестве метода решения применяем метод Гаусса (метод последовательного исключения неизвестных) с выбором главного элемента.

Занесем коэффициенты системы (5.2) в масив `A: Matr` по такой схеме

		1	2	3	…	*n*	*n*+1
	1	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$		$a_{1,n}$	$a_{1,n+1}$
	2	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$		$a_{2,n}$	$a_{2,n+1}$
A	3	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$		$a_{3,n}$	$a_{3,n+1}$
	…	…	…	…		…	…
	n	$a_{n,1}$	$a_{n,2}$	$a_{n,3}$		$a_{n,n}$	$a_{n,n+1}$

Суть метода Гаусса решения системы (5.2) заключается в следующем. Из 1 – го уравнения определяется x_1 (это уравнения решается относительно x_1)

$$x_1 = \left(a_{1,n+1} - a_{1,2}x_2 - a_{1,3}x_3 - ... - a_{1,n}x_n\right) / a_{1,1} \quad (5.3)$$

и это выражение для x_1 подставляется во 2–е, 3–е, … , *n*–е уравнения. В результате после сведения подобных в этих уравнениях x_1 как таковое исчезает (коэффициенты $a_{2,1}, a_{3,1}, ..., a_{n,1}$ превращаются в нули). Естественно, речь не идет о коэффициентах с такими индексами в исходной системе уравнений, а следовательно, и в массиве *A*. В процессе реализации алгоритма мы будем последовательно трансформировать массив *A*. Поэтому на каждой стадии процесса решения речь будет идти о содержимом ячеек с соответствующими индексами, а это содержимое будет постоянно обновляться по мере реализации алгоритма.

После того как обнулены коэффициенты при x_1 в уравнениях со 2–го по *n* – е (элементы матрицы *A* в первом столбце начиная со 2–го и ниже), определяем x_2

из 2–го уравнения и подставляем его в 3–е, 4–е, … , *n*–е. В итоге обнулятся элементы 2–го столбца, (начиная с 3–го) и так далее. Это так называемый прямой ход в алгоритме Гаусса. В итоге выполнения прямого хода обнуляются все элементы массива коэффициентов ниже главной диагонали. Последний (*n*+1) – ый столбец преобразуется синхронно с остальными, хотя он формально и выходит за пределы "квадратной части" расширенной матрицы (расширение то как раз и состоит в присоединении столбца свободных членов к квадратной части матрицы коэффициентов). Понятие главная диагональ относится к квадратной части матрицы.

Если выполнение прямого хода завершилось благополучно (а это, вообще-то говоря, вовсе не обязательно, но об этом несколько позже), то последняя строка трансформированной расширенной матрицы будет соответствовать такому уравнению

$$a_{n,n}x_n = a_{n,n+1}, \tag{5.4}$$

которое может быть решено

$$x_n = a_{n,n+1}/a_{n,n}. \tag{5.5}$$

Это уже компонент вектора – решения. Предпоследняя строка соответствует уравнению

$$a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n = a_{n-1,n+1},$$

из которого определяется x_{n-1}

$$x_{n-1} = \left(a_{n-1,n+1} - a_{n-1,n}x_n\right)/a_{n-1,n-1} \tag{5.6}$$

и т.д.

Таким образом идя "снизу вверх" мы получаем возможность последовательно определять $x_n, x_{n-1}, x_{n-2}, ..., x_1$. Это уже обратный ход схемы (алгоритма) Гаусса.

Обратим внимание на то, что в формулах (5.3), (5.6) предусмотрено деление на "диагональные" элементы. А что если они окажутся нулями? А почему бы и нет? На стадии выполнения прямого хода, если в `k`–ом уравнении не окажется слагаемого с `k`–ой неизвестной (`A[k,k]=0`), то логично попытаться

среди уравнений, расположенных ниже k–го, найти такое уравнение, у которого бы коэффициент при x_k не был бы нулем. Если таковое будет найдено, то можно ведь и поменять его местом с k–ым. Ведь от перестановки уравнений в системе ее решение не должно измениться. Но – это если найдется подходящее. А если не найдется? Оставлять нуль на месте A[k,k] мы не имеем права – ведь, с одной стороны, окажутся не обнуленными элементы матрицы, расположенные под ним, а с другой стороны, если бы довелось выполнять обратный ход, то пришлось бы на этот нуль делить, как в формулах (5.5), (5.6). Но достаточно "во – первых"! И что же делать? А ничего! Ведь бывают системы, которые решения не имеют! Вот такая – то система нам как раз и попалась. Остается остановить процесс поиска решения – и доложить "куда следует". Это не мы виноваты – это система виновата. Ну, представьте себе две параллельные прямые на плоскости (это эквивалент системы двух линейных алгебраических уравнений). Точка пересечения прямых – решение системы. А поскольку прямые не пересекаются, то и решения нет. От того, что параллельные прямые пересекаются где-то в бесконечности – не легче, т.к. бесконечное решение нас не устраивает.

А теперь еще раз вернемся к прямому ходу. Там мы говорили о поиске уравнения, у которого коэффициент при x_k не был бы нулем. А вы не забыли, что все коэффициенты известны не точно, а только приближенно? И представьте, что вы нашли уравнение, где коэффициент при x_k не нуль, но "не внушает доверия"… Скажем, его величина по модулю близка к возможной погрешности в его определении. А ведь на него делить придется (и в прямом и в обратном ходе!). А при делении на "потенциальный ноль" можно наломать дров! Чтобы уменьшить вероятность подобных неприятностей при поиске "подходящего для обмена" уравнения, будем искать не просто уравнение где бы этот коэффициент при x_k не был бы нулем, а такое уравнения, где бы этот коэффициент был по возможности больше (по модулю). В этом и состоит суть поиска (выбора) "главного элемента", (тогда метод приобретает название – метод Гаусса с выбором главного элемента). И только тогда, когда "главный элемент"

оказывается несостоятельным (меньше по модулю от ε), тогда процесс решения признается несостоявшимся. Договоримся результаты решения системы уравнений размещать в массиве типа `Coef` по схеме

	-1	0	1	2	3	…	n	…	31
x	n	-	x_1	x_2	x_3	...	x_n		

Здесь ячейка с номером 0 – “гуляет”, а в ячейку с номером –1 заносится число найденных корней уравнения. Если решить систему не удалось, то договоримся честно записать в `x[-1]` ноль. Когда пользователь увидит ноль в этой ячейке – он все поймет!

```
Procedure SystUr(n:integer; A:Matr; var x:Coef);
const Eps=1e-9;
var Zmax,z,s,i,j:integer;
    R:real;
    B:boolean;
  begin
    B:=true;
    if n=1
      then if abs(A[1,1])<Eps then B:=false
                              else X[1]:=A[1,2]/A[1,1]
      else
        begin
          i:=1;
          while (i<=n-1) and B do
            begin
              Zmax:=i;
              for z:=i+1 to n do
               if abs(A[z,i])>abs(A[Zmax,i]) then Zmax:=z;
              if abs(A[Zmax,i])<Eps
                then B:=false
                else
                  begin
                    if Zmax>i then
                      for j:=i to n+1 do
                        begin
                          R:=A[i,j]; A[i,j]:=A[Zmax,j];
                          A[Zmax,j]:=R
                        end;
                    for z:=i+1 to n do
```

```
                    if abs(a[z,i])>Eps then
                      begin
                        R:=A[z,i]/A[i,i];
                        for s:=i+1 to n+1 do
                          A[z,s]:=A[z,s]-R*A[i,s]
                      end
                  end;
              inc(i)
            end;
          if B then if abs(A[n,n])<Eps then B:=false;
          if B then
            begin
              X[n]:=A[n,n+1]/A[n,n];
              for i:=n-1 downto 1 do
                begin
                  R:=0;
                  for s:=i+1 to n do
                    R:=R+A[i,s]*X[s];
                    X[i]:=(A[i,n+1]-R)/A[i,i]
                end
            end
        end;
     if B then X[-1]:=n else X[-1]:=0
  end;
```

Кратко прокомментируем подпрограмму. В списке параметров *n* – порядок системы передаем отдельно. Как вариант можно было договориться, например, передавать его в `A[1,-1]`, если бы в определении типа Matr наличие столбца с номером –1 было предусмотрено. Решения системы принимаем в массив `X: Coef`. Слово var перед ним об этом напоминает (нам и компилятору).

Константа `Eps` – это та малая величина, ниже которой мы перестаем “всерьез” воспринимать значения коэффициентов (элементов матрицы) в процессе их преобразования. Элемент по модулю меньший `Eps`, приравнивается нулю. `Zmax` – номер строки, в которой обнаружен больший по модулю элемент – претендент на диагональный. `Z, S` – номера строки и столбца соответственно; `i, j` – вспомогательные счетчики, `R` – промежуточная переменная. `B:Boolean` – флажок. Пока `B=true` надежда на успех имеется, когда `B=false` – поражение.

Начинается операторная часть на оптимистической ноте (`B:=true`) –

непреодолимых препятствий для успеха предприятия еще не обнаружено (правда, мы еще ничего и не делали, потому-то сдаваться еще явно рановато!).

Если `n=1`, то речь идет о решении системы (правда, это несколько сильно сказано)

$$a_{1,1}x_1 = a_{1,2}.$$

Ну, решение, естественно

$$x_1 = a_{1,2}/a_{1,1}. \qquad (5.7)$$

Оказывается, что даже в этом примитивнейшем случае можно схлопотать себе неприятности, если вдруг $a_{1,1}$ окажется не состоятельным (`if abs(A[1,1]<Eps)`). Это как если бы вы вдруг стали решать уравнения вида

$$0 = a_{1,2}.$$

Если же $a_{1,1}$ явно не ноль, то реализуется решение (5.7).

Вариант с `n=1`, таким образом, решаем “в лоб” не прибегая к алгоритму Гаусса (нет потребности), а вот, когда условие `n=1` не выполняется, то реализуем алгоритм в полном объеме.

Буквой `i` обозначаем номер диагонального элемента, под которым производится обнуление элементов матрицы. Естественно, что начинаем с первого (`i:=1`). В цикле `while` значение `i` постепенно (с шагом 1) увеличивается до тех пор, пока оно остается меньше или равно `n-1` или же пока не возникнет ситуация (неустранимый ноль в очередном `i`–ом диагональном элементе), когда процесс прямого хода дальше окажется невозможным. Итак

```
while (i<=n-1) and B do …
```

Почему до `n-1`? А дело в том, что под `n`–ым диагональным элементом нечего обнулять.

Тело цикла `while` начинается с поиска `Zmax` – номера строки, в которой элемент `A[Zmax,i]` является наибольшим по модулю среди элементов, у которых первый индекс (номер строки) изменяется от `i` до `n` (от `i` – го

диагонального и до крайнего нижнего). В итоге имеем конструкцию

```
Zmax:= i;
for Z:= i+1 to n do
  if abs(A[Z, i])>abs(A[Zmax, i])
    then Zmax:= Z;
```

После того как Zmax найдено, проверяем abs(A[Zmax,i]) на солидность (стоит ли менять шило на мыло?). Если нет – то “сдаемся”

```
if abs(A[Zmax, i])<Eps
  then B:=false …
```

Если же элемент A[Zmax,i]) заслуживает доверия, то – следует поменять местами Zmax – ую и i – ую строки. Да, следует, если это не одна и та же строка (начали мы ведь с того, что Zmax:=i), а что если затем в цикле

```
for Z:= i+1 to n do..
```

не нашлось более достойного претендента? В принципе, поменять строку саму с собой – никаких проблем, но зачем? Итак

```
if Zmax> i then
  for j:= i to n do
    begin
      R:= A[i, j]; A[i, j]:= A[Zmax, j]; A[Zmax, j]:= R
    end;
```

Это классический обмен с использованием “третьего стаканчика” R.

Итак, i – ая строка заняла свое окончательное положение. Теперь нужно обнулить элементы, стоящие под i – ым диагональным элементом. Опять же – стоит ли обнулять элемент, если он и так нуль? “Лежачего не бьют”. В целом процесс обнуления выглядит так

```
for Z:=i+1 to n do
  if abs(A[Z,i])> Eps then
     begin
       R:= A[Z,i]/A[i,i];
       for S:=i+1 to n+1 do
         A[Z,S]:=A[Z,S]-R*A[i,S]
```

```
end;
```

Технология обнуления A[Z,S] выглядит так. Будем вычитать из Z – ой строки i – ую строку, умноженную на некоторый коэффициент R, подобранный таким образом, чтобы

```
A[Z,i]-R*A[i,i]=0,
```

откуда получаем требуемое значение `R`

```
R:=A[Z,i]/A[i,i].
```

Дальше – дело техники. В цикле по `S`, где `S` изменяется от `i+1` до `n+1` выполняем упомянутое вычитание. Да, а почему от `i+1`, а не от `i`? А мы ведь и так знаем, что элемент `A[Z,i]` должен превратиться в нуль, так что в дальнейшем мы даже не будем туда заглядывать (зачем интересоваться пустым местом?), так что физически его ликвидировать и нет надобности. Закончив обнуление требуемых элементов i – го столбца можно переходить к очередному `i` (`inc(i)`). После того как все i перебраны или мы вышли из игры (из цикла `while`) по второму условию (не выполняется `and B`), то в любом случае прямой ход закончен. Но совсем не безразлично как он закончен. “Со щитом или на щите”?

```
if B then if abs(A[n,n])<Eps
  then B:=false;
```

Если `B:=false`, то все ясно – прямой ход закончился неудачей, а вот если `B=true (if B)`, то следует еще посмотреть `abs(A[n,n])`. Если окажется, что последний диагональный элемент “не тянет”, то `X[n]` нам не видать, а вместе с ним и остальных неизвестных не определить. Это последняя проверка. Жаль, конечно, если ее не удастся пройти, но “c’est la vie!”.

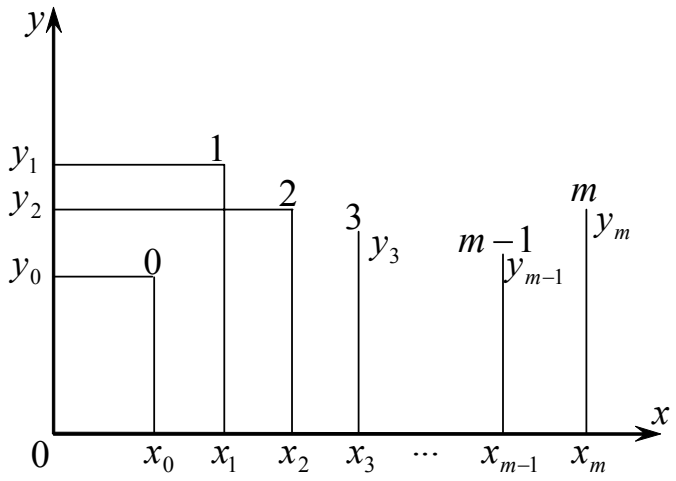
Но зато если “знамя реет” (`B=true`), то после завершения прямого хода пора приступать к выполнению обратного (ведь прямой ход это всего только подготовка к основному, завершающему обратному ходу).

```
X[n]:=A[n,n+1]/A[n,n].
```

Тут все ясно. В цикле по i перебираем последовательно номера определяемых неизвестных (от (n−1)–го до 1–го включительно). Для этого в переменной R формируем сумму слагаемых i–го уравнения с уже определенными значениями неизвестных. Затем эту сумму переносим в правую часть и полученную разность делим на A[i,i]. Результат – в X[i]. Последний штрих – заполнения X[-1]–как договорились раньше.

5.5. Формирование интерполяционного полинома

Рассмотрим следующую задачу



Есть набор значений x_S, $0 \le s \le m$. Назовем их узлами интерполирования. Есть соответствующие данным узлам значения y_S, $0 \le s \le m$ т. е. задан набор точек некоторой функциональной зависимости.

Требуется сформировать непрерывную (гладкую) функцию $f(x)$, значения которой в узлах x_S совпадали бы со значениями y_S.

$$f(x_S) = y_S,\ 0 \le s \le m. \tag{5.8}$$

Такую функцию будем называть интерполяционной (интерполирующей). Термин интерполяция, подразумевает то обстоятельство, что функцию $f(x)$ можно использовать для целей интерполирования т. е. для определения значений функции в промежутках между узлами.

Набор узлов интерполирования может быть произвольным (как в смысле

порядка их нумерации, так и в смысле расстояний между соседними (пространственно) узлами). Естественным ограничением будет отсутствие совпадающих узлов, а в более широком смысле – отсутствие очень близких между собой узлов. Если допустить существование нескольких совпадающих узлов (для которых значения y_S не совпадают), то из этого вытекает требование для интерполирующей функции иметь разрывы в местах совпадения узлов. Если же узлы хотя и не совпадают полностью, но достаточно близки, то в промежутке между близкими узлами аппроксимирующая функция должна иметь большие по модулю производные (быть "крутой" на промежутке). Подобные особенности аппроксимирующей функции чаще всего нежелательны.

Вариантов интерполирующих функций можно предложить множество. Будем для определенности искать решение в классе полиномов. Полином – любимая функция математиков. И есть за что! (Имеется в виду – есть за что любить полиномы). Полином – простая по структуре и удобная для использования функция. Для нее легко запрограммировать всевозможные операции. Даже мы с вами не смогли удержаться, чтобы не написать кучу подпрограмм для работы с полиномами. Вместе с тем – полином достаточно гибкая функция – ее гибкость легко регулировать подбирая ту или иную степень. Итак, пусть интерполирующая функция имеет вид

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n . \qquad (5.9)$$

Тогда условия (5.8) прохождения графика интерполяционного полинома (5.9) через все заданные точки на плоскости x, y будет выглядеть так

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 + a_3 x_0^3 + \ldots + a_n x_0^n = y_0, \\ a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 + \ldots + a_n x_1^n = y_1, \\ a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3 + \ldots + a_n x_2^n = y_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_0 + a_1 x_m + a_2 x_m^2 + a_3 x_m^3 + \ldots + a_n x_m^n = y_m. \end{cases} \qquad (5.10)$$

Это система уравнений относительно искомых значений коэффициентов интерполирующего полинома (5.9). Такая система будет иметь единственное

решение при условии, что количество уравнений и количество искомых неизвестных будут равны друг другу

$$n+1=m+1$$

или что то же

$$n=m. \qquad (5.11)$$

Таким образом, формально степень n интерполяционного полинома должна равняться старшему номеру узла интерполяции (нумерация с 0, равно как и нумерация коэффициентов интерполяционного полинома). Фактически (если повезет?!) степень полинома (фактическая степень) может оказаться ниже формальной n. Но это выяснится только после определения коэффициентов (5.9), т. е. после решения системы (5.10).

Расширенная матрица `D:Matr` коэффициентов системы уравнений (5.10) выглядит так,

		1	2	3	…	n+1	n+2
	1	1	x_0	x_0^2	…	x_0^n	y_0
	2	1	x_1	x_1^2	…	x_1^n	y_1
D	3	1	x_2	x_2^2	…	x_2^n	y_2
	…	…			…	…	…
	$n+1$	1	x_n	x_n^2	…	x_n^n	y_n
		a_0	a_1	a_2	…	a_n	Правые части

Формирование матрицы D, решение соответствующий системы уравнений, а также “редактирование” полученного массива решений реализуется в подпрограмме `IntPol`.

```
Procedure IntPol (x,y:CoefR; var A:Coef);
var z,s,n:integer; D:Matr;
begin
  n:=round(X[-1]);
  for z:=1 to n+1 do
    begin
      D[z,1]:=1; D[z,n+2]:=Y[z-1];
      for s:=2 to n+1 do
```

```
        D[z,s]:=D[z,s-1]*X[z-1]
    end;
  SystUr(n+1,D,A);
  if A[-1]>0 then
    begin
      A[-1]:=n;
      for s:=0 to n do A[s]:=A[s+1]
    end
end;
```

Итак, в массиве `X:Coefr` передается набор узлов интерполирования, в ячейке `X[-1]` хранится значение `m`, которое для решения задач интерполирования принимается равным `n` согласно (5.11). В массиве `Y:Coefr` – значения y_s в соответствии с (5.8). Тип `Coefr` для решения данной задачи, пожалуй, слишком просторный. Ну, не собираемся же мы всерьез формировать полином (5.9) со степенью выше 30 – й! Почему именно 30 – й? Да потому, что для формирования интерполяционных полиномов более низких степеней можно было бы обойтись и массивами `X` и `Y` типа `Coef`. Но необходимость (и целесообразность) использования именно такого типа будет оправданной при решении родственных задач обработки данных (например, экспериментальных данных). Массив же `A:Coef` предназначен для возвращения (слово var перед соответствующим параметром в заголовке подпрограммы) информации об интерполяционном полиноме, который как раз и должен быть сформирован в данной подпрограмме. Для интерполяционного полинома типа `Coef` заведомо достаточно, т. е. мы всерьез не допускаем поиска интерполяционных полиномов фантастически высоких степеней (а степень выше 30 – й – это уже действительно фантастика!). Массив `D:Matr` – для размещения расширенной матрицы коэффициентов системы (5.10).

Начинается операторная часть подпрограммы с определения $n=m$ (`n:= round(X[-1])`).

Цикл по `Z` (`for Z:= 1 to n+1 do …`) перебирает строки, вложенный в него цикл по `S` – столбцы расширенной матрицы. В итоге формируется расширенная матрица `D`.

Процедура `SystUr` решает систему (5.10). Решение (искомые коэффициенты полинома (5.9)) возвращаются `SystUr` в массиве `X:Coef` в виде

	-1	0	1	2	3	…	n+1	…	31
x	n+1	-	a_0	a_1	a_2		a_n		

Здесь вся необходимая информация об интерполяционном полиноме – налицо, но в несколько, скажем так, нестандартном виде. Но это легко исправить

```
if A[-1]> 0 then
  begin
    A[-1]:= n;
    for S:= 0 to n do A[S]:= A[S+1]
  end.
```

Перестраховка (`if A[-1]> 0`…) – на тот случай, когда решить систему (5.10) не удастся. Часть возможных причин мы уже обсуждали, возможны и другие. Ну, например, в каких единицах измеряются `X`–ы и `Y`–и? Особенно `X`–ы! Ведь они участвуют в уравнениях системы в высоких степенях (n ведь может быть не маленьким!). Желательно бы чтобы `X` – ы были по возможности близки к единице. Но вместе с тем мы ведь предостерегали читателя, чтобы не брал узлы слишком близко! Приходится вертеть головой во все стороны (“хочешь жить – умей вертеться”).

Редактирование полинома (на случай превышения его формальной степени по сравнению с фактической) мы уже ранее рассматривали. Но где заканчиваются “малые“ коэффициенты и начинаются “большие” – “думайте сами, решайте сами – жить или не жить!”.

5.6. Формирование сглаживающего полинома

Алгоритм формирования интерполяционного полинома провокационно прост. И степень его определить – “легко и просто!” но – “иная простота – похуже воровства!”. Будет у вас 25 экспериментальных точек (а если уже представилась возможность поэкспериментировать, то, очевидно, чем больше –

тем лучше). И станете вы искать интерполяционный полином 25 – й (или 24 – й – не все ли равно?) степени? Ну, во-первых, его еще нужно получить! И проблемы с памятью (объем типа `Matr`), да и при решении систем такого высокого порядка могут возникнуть проблемы (возведите какой-то x_S в 25 – ую степень, я на вас посмотрю…). И, наконец, – ну, получите вы полином 25 – й степени – что вы с ним делать будете? Если вы уповаете на то, что компьютер справится, то еще, как говорится, – "бабка надвое говорила". Короче – полиномы высоких степеней – даже интерполяционные – нам не нужны. Ну, так что – выбрасывать часть точек, добытых таким трудом (не говоря уже о времени, расходах и т. д.)? Так жалко ведь. А с другой стороны – полученные экспериментальные точки они ведь тоже "не без греха" – без погрешностей там вряд ли обошлось (и дело не только в злом умысле или разгильдяйстве, хотя и этого исключить нельзя). Просто эксперимент, он на то и эксперимент – "лес рубят - щепки летят". А точно отражать в интерполяционном полиноме все погрешности эксперимента – стоит ли? Может, истина – попроще? Может, если бы удалось каким-то образом "отшелушить" "руду золотую" от "породы пустой", то "золото" поместилось бы в полином 3 – ей или там 4 – ей степени? Идея заманчива, да вот как это сделать? Ну, 100 – процентной гарантии вам никто не даст, а вот кое-что можно предпринять. Давайте не будем настаивать на прохождении графика аппроксимирующего полинома (заданной степени `n<m`) через все рассматриваемые точки, а ограничимся пожеланием, чтобы этот график прошел все – таки по возможности ближе к этим точкам. Формулировка, прямо скажем, не больно конкретная… Попробуем конкретизировать ее таким образом. В качестве показателя приближения (качества аппроксимации) примем такой

$$E = \sum_{S=0}^{m} \left(a_0 + a_1 x_S + a_2 x_S^2 + \ldots + a_n x_S^n - y_S \right)^2 . \qquad (5.12)$$

Очевидно, чем меньше будет E – тем лучше. Интерполяционный полином в смысле минимизации данного показателя вообще вне конкуренции. Для него E=0 (это, естественно, если не учитывать погрешностей расчета коэффициентов полинома, и вычисления его значений в узлах интерполирования и вычисления

самой суммы квадратов). Но все же чем меньше E, тем – лучше. Логично будет предположить, что увеличивая n – степень аппроксимирующего полинома – можно надеяться на уменьшение E. По крайней мере, если наращивая n добраться до $n=m$, то должен получиться по сути интерполяционный полином (с E=0). Если m – велико, то до этого, пожалуй, не дойдет, но тенденция просматривается. Кстати, постепенно наращивая степень полинома можно "нащупать" подходящее значение n. Какое значения E следует принимать за "приемлемое". Да это ведь зависит от "зашумленности" данных, и от количества точек. Ну, от последнего фактора можно было бы, например, отстроиться поделив E на (m+1). Но это – детали, будем искать полином (заданной степени n), минимизирующий (5.12).

В (5.12) искомыми являются коэффициенты $a_0, a_1, a_2, ..., a_n$. Будем рассматривать их как неизвестные (так ведь оно и есть!), тем самым функция E оказывается функцией не одной, а (n +1) – ой неизвестных. Условие (условия!) ее минимума будет

$$\begin{cases} \dfrac{\partial E}{\partial a_0} = 0, \\ \dfrac{\partial E}{\partial a_1} = 0, \\ \dfrac{\partial E}{\partial a_2} = 0, \\ \dfrac{\partial E}{\partial a_n} = 0. \end{cases} \quad \begin{matrix} \\ \\ \\ \cdots \end{matrix} \tag{5.13}$$

Раскроем уравнения (5.13) и сразу сократим каждое из них на 2

$$\begin{cases} \sum_{S=0}^{m} \left(a_0 + a_1 x_S + a_2 x_S^2 + ... + a_n x_S^n - y_S\right) x_S^0 = 0, \\ \sum_{S=0}^{m} \left(a_0 + a_1 x_S + a_2 x_S^2 + ... + a_n x_S^n - y_S\right) x_S^1 = 0, \\ \sum_{S=0}^{m} \left(a_0 + a_1 x_S + a_2 x_S^2 + ... + a_n x_S^n - y_S\right) x_S^2 = 0, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \sum_{S=0}^{m} \left(a_0 + a_1 x_S + a_2 x_S^2 + ... + a_n x_S^n - y_S\right) x_S^n = 0. \end{cases} \tag{5.14}$$

Если раскрыть скобки и перенести в правые части слагаемые с y_S в (5.14),

то расширенная матрица коэффициентов системы уравнений приобретет вид

D

	1	2	3	…	n+1	n+2
1	$\sum_{S=0}^{m} x_S^0$	$\sum_{S=0}^{m} x_S^1$	$\sum_{S=0}^{m} x_S^2$	…	$\sum_{S=0}^{m} x_S^n$	$\sum_{S=0}^{m} y_S x_S^0$
2	$\sum_{S=0}^{m} x_S^1$	$\sum_{S=0}^{m} x_S^2$	$\sum_{S=0}^{m} x_S^3$	…	$\sum_{S=0}^{m} x_S^{n+1}$	$\sum_{S=0}^{m} y_S x_S^1$
3	$\sum_{S=0}^{m} x_S^2$	$\sum_{S=0}^{m} x_S^3$	$\sum_{S=0}^{m} x_S^4$	…	$\sum_{S=0}^{m} x_S^{n+2}$	$\sum_{S=0}^{m} y_S x_S^2$
…	...	...	...	…	...	...
n+1	$\sum_{S=0}^{m} x_S^n$	$\sum_{S=0}^{m} x_S^{n+1}$	$\sum_{S=0}^{m} x_S^{n+2}$	…	$\sum_{S=0}^{m} x_S^{2n}$	$\sum_{S=0}^{m} y_S x_S^n$
	a_0	a_1	a_2	…	a_n	правые части

Тут под каждым столбцом матрицы показано неизвестное, коэффициентами при котором являются элементы данного столбца.

Даже само формирование подобной матрицы (особенно, если n и m не маленькие) – задача не для слабонервных. Хотя – не так страшен черт как его малюют! Да и не собираемся же мы, в самом деле, считать его вручную. Так что попробуем все это запрограммировать. Сам метод называют сокращенно МНК – метод наименьших квадратов (это из-за (5.12)), что отразилось в названии подпрограммы.

```
Procedure NaimKv(x,y:CoefR;n:integer; var E:real;
                 var A:Coef);
var m,z,s:integer;
    Sum:real;
    B,C:CoefR; D:Matr;
begin
  m:=round(X[-1]);
  for s:=0 to m do B[s]:=1;
  for z:=0 to 2*n do
    begin
      C[z]:=0; Sum:=0;
      for s:=0 to m do
        begin
          C[z]:=C[z]+B[s];
```

```
          if Z<=n then Sum:=Sum+B[s]*Y[s];
          B[s]:=B[s]*X[s]
        end;
      if z<=n then D[z+1,n+2]:=Sum
    end;
  for z:=1 to n+1 do
    for s:=1 to n+1 do
      D[z,s]:=C[z+s-2];
  SystUr(n+1,D,A); A[-1]:=n; E:=0;
  for s:=0 to n do A[s]:=A[s+1];
  for s:=0 to m do E:=E+sqr(HorReal(A,X[s])-Y[s])
end;
```

В этой процедуре `X,Y:CoefR` – массивы с заданным набором точек: узлов (в X) и соответствующих им значений функции (в Y). В ячейке `X[-1]` – число *m*, старший номер узла, нумерация точек (узлов) с нуля.

Здесь, *n* – заданная степень аппроксимирующего (иногда говорят – сглаживающего, чем меньше *n* – тем сильнее эффект сглаживания) полинома; `E` – величина показателя (5.12) – возвращается после того как найден аппроксимирующий полином; `A:Coef` – массив с информацией о найденном аппроксимирующем полиноме; `m` – старший номер узла (точки), Z, S – номер строки и номер столбца в массиве `D` коэффициентов; `D:Matr` – сама матрица коэффициентов системы (5.14); `B,C:CoefR` – вспомогательные массивы такой структуры

	-1	0	1	2	3	…	*m*	…	201
B		x_0^R	x_1^R	x_2^R	x_3^R	…	x_m^R	…	

	-1	0	1	2	3	…	2*n*	…	201
C		$\sum_{S=0}^{m} x_S^0$	$\sum_{S=0}^{m} x_S^1$	$\sum_{S=0}^{m} x_S^2$	$\sum_{S=0}^{m} x_S^3$	…	$\sum_{S=0}^{m} x_S^{2n}$	…	

Можно было бы “для полного счастья” в `B[-1]` записать *m*, а в `C[-1]` соответственно 2*n*, но это массивы для внутреннего пользования, так что можно обойтись и без такой формальности. `R` в массиве `B` будет последовательно принимать значения `0, 1, 2, … , 2n`. Когда при текущем R, например, при

R=0 массив B заполнен значениями x_0^R т. е. единицами, то суммируя содержимое ячеек массива B, начинается от 0 – й до *m* – ой, найденную сумму можно занести в C[0]. Умножая поэлементно массив B с x_0^R на массив X, повышаем R на единицу. Сумму элементов массива B в этом случае заносим в C[1] и т. д.

Итак, предварительно формируем массив B для R=0 (`for S:= 0 to m do B[S]:=1;`). Затем в цикле по Z (`for Z:= 0 to 2*n do…`) заполняем ячейки массива C. Сумму элементов массива B с текущим R накапливаем в соответствующей (с номером Z равным R) ячейке того же массива C. Поэтому начинаем с "чистки" ячейки (`C[Z]:=0`). В идентификаторе Sum: real накапливаем сумму для последнего элемента Z – ой строки, (n+2) – го столбика массива D. Начинаем, естественно с `Sum:=0`; в цикле по S (`for S:= 0 to m do…`) "собираем урожай" для ячейки C[Z] (суммируем массив B), если Z<=n, то вычисляем свободный член Z – го уравнения. После того как из массива B "выжато" все необходимое – он "идет в переплавку" – повышаем в нем R на 1, умножая поэлементно на массив X. После выхода из только что рассмотренного цикла по S, если накопленная сумма Sum востребована (у нас правых частей как-никак меньше 2n), то значение Sum записывается в D[Z+1,n+2]. Этим тело цикла по Z заканчивается, т. е. осуществляется переход к очередному Z, пока заказанный ряд значений Z не будет исчерпан.

И вот, наконец, он таки исчерпан, т. е. массив C сформирован полностью. Теперь его можно разнести по ячейкам массива D (еще не заполненным). Это и выполняется

```
for Z:= 1 to n+1 do
  for S:= 1 to n+1 do D[Z, S]:= C[Z+S-2];
```

Этим формирование матрицы D завершается. Остается передать ее в распоряжение процедуры SystUr, которая возвратит массив A:Coef с информацией о сглаживающем полиноме. Здесь ситуация совершенно аналогична той, что имела место при получении интерполяционного полинома.

Аналогичен и алгоритм коррекции. Ну, наконец, мы обещали подсчитать и возвратить значение показателя E. Начальное значения (`E:=0`) и цикл по `S` (`for S:= 0 to m do…`) выполняют задачу расчета `E`. Тут мы прибегаем к услугам функции `HorReal`, так ведь для этого мы ее и создавали!

5.7. Аппроксимация функциональной зависимости интерполяционным дробно-рациональным выражением

В двух предыдущих параграфах в качестве аппроксимирующих (приблизительно – описывающих), функций применялись полиномы той или иной степени. Имеется еще одна весьма популярная аппроксимирующая структура, а именно, отношение полиномов

$$f(x)=\frac{b_0+b_1x+b_2x^2+\ldots+b_{Nb}x^{Nb}}{1+a_1x+a_2x^2+\ldots+a_{Na}x^{Na}}. \qquad (5.15)$$

Выражение (5.15) еще называют дробно-рациональной функцией.

Итак, пусть имеются массивы `X,Y:CoefR` с координатами x и y точек исследуемой функциональной зависимости. Точки пронумерованы от 0 до m. Неизвестных коэффициентов в (5.15) ($Na+Nb+1$). Чтобы решить задачу интерполяции (провести график (5.15) через все заданные точки), необходимо выдержать условие

$$Na+Nb=m. \qquad (5.16)$$

Если характер поведения аппроксимирующей функции в исследуемом диапазоне не уточнить, то давать какие бы то ни было рекомендации по выбору Na и Nb – весьма затруднительно. Чтобы действовать более осмысленно, слегка изменим постановку задачи. Предположим, что требуется так подобрать Na и Nb, чтобы обеспечить достаточно точно аппроксимацию некоторой неизвестной функции структурой типа (5.15). Речь, понятно, идет не только о совпадении функций в узлах интерполирования (это – само собой), но и о близости функций в промежутках между узлами. Подобная постановка задачи может быть использована на практике, когда, например, речь идет о табулировании функций, вычисление которых очень трудоемко. Хотя, если честно, понятие трудоемкости

в век компьютеров все больше утрачивает свой устрашающий смысл. Ну, да речь не об этом – мы учимся программировать, а для учебных целей такие задачи полезны.

Итак, пусть имеется некая подпрограмма `function Fx (x: real): real;` к помощи которой можно прибегнуть при необходимости сформировать массив `Y:Coef` значений функции в узлах аппроксимации, которые равномерно (с постоянным шагом `Dx`) распределены на отрезке $x_n \ldots x_k$. Значения Na и Nb степеней числителя и знаменателя функции (5.15) считаем заданными. При этом автоматически выдерживается соотношение (5.16), поскольку речь идет о задаче интерполирования. Условие прохождения графика (5.15) через все точки x_S, y_S $0 \le S \le m$ на плоскости xoy будет

$$\frac{b_0 + b_1 x_S + b_2 x_S^2 + \ldots + b_{Nb} x_S^{Nb}}{1 + a_1 x_S + a_2 x_S^2 + \ldots + a_{Na} x_S^{Na}} = y_S,\ 0 \le S \le m.$$

Перепишем последнее соотношение в виде

$$\left(a_1 x_S + a_2 x_S^2 + \ldots + a_{Na} x_S^{Na}\right) y_S - b_0 - b_1 x_S - b_2 x_S^2 - \ldots - b_{Nb} x_S^{Nb} = -y_S,\ 0 \le S \le m. \quad (5.17)$$

Это система линейных алгебраических уравнений относительно искомых коэффициентов $a_1, a_2, a_3, \ldots, a_{Na}, b_0, b_1, b_2, \ldots, b_{Nb}$ (итого $Na + Nb + 1$ неизвестных). Если как уже отмечалось, выполняется условие (5.16), то можно рассчитывать на получение решения системы (5.17).

Поставленная таким образом задача решается подпрограммой `DrapInt`. Она работает с расширенной матрицей `D` коэффициентов системы (5.17) такого вида

D		1	2	3	…	*Na*	*Na*+1	*Na*+2	*Na*+3	…	*m*+1	*m*+2
	1	$x_0 y_0$	$x_0^2 y_0$	$x_0^3 y_0$	…	$x_0^{Na} y_0$	-1	$-x_0$	$-x_0^2$	…	$-x_0^{Na}$	$-y_0$
	2	$x_1 y_1$	$x_1^2 y_1$	$x_1^3 y_1$	…	$x_1^{Na} y_1$	-1	$-x_1$	$-x_1^2$	…	$-x_1^{Na}$	$-y_1$
	3	$x_2 y_2$	$x_2^2 y_2$	$x_2^3 y_2$	…	$x_2^{Na} y_2$	-1	$-x_2$	$-x_2^2$	…	$-x_2^{Na}$	$-y_2$
	…	…	…	…	…	…	…	…	…	…	…	…
	m+1	$x_m y_m$	$x_m^2 y_m$	$x_m^3 y_m$	…	$x_m^{Na} y_m$	-1	$-x_m$	$-x_m^2$	…	$-x_m^{Na}$	$-y_m$

Сама же подпрограмма приводится ниже

```
Procedure DrapInt(Xn,Xk:real;Na,Nb:integer; var B,A:Coef);
var m,z,s:integer; Dx:real; D:Matr; X,Y:Coef;
begin
  m:=Na+Nb; Dx:=(Xk-Xn)/m;
  for s:=0 to m do
    begin
      X[s]:=Xn+s*Dx; Y[s]:=Fx(X[s])
    end;
  for z:=1 to m+1 do
    begin
      if Na>0 then
        D[z,1]:=X[z-1]*Y[z-1];
      for s:=2 to Na do
        D[z,s]:=D[z,s-1]*X[z-1]
      D[z,Na+1]:=-1;
      for s:=2 to Nb+1 do
        D[z,Na+s];=D[z,Na+s-1]*X[z-1];
      D[z,m+2]:=-Y[z-1]
    end;
    SystUr(m+1,D,A); A[-1]:=Na; A[0]:=1; B[-1]:=Nb;
    for s:=0 to Nb do B[s]:=A[Na+1+s]
 end;
```

5.8. Аппроксимация функции сглаживающей дробно-рациональной структурой

Более содержательным (и, пожалуй, более гибким) представляется вариант, когда жесткое условие (5.17) заменяется условием минимизации показателя качества вида

$$E=\sum_{i=0}^{m}\left[\left(a_1x_i+a_2x_i^2+...+a_{Na}x_i^{Na}\right)y_i-b_0-b_1x_i-b_2x_i^2-...-b_{Nb}x_i^{Nb}+y_i\right]^2. \quad (5.18)$$

Условия минимизации E получаем путем приравнивая нулю частных производных от E по всем искомым коэффициентам (как неизвестным). Имеем

$$\begin{cases}\sum_{i=0}^{m}\left[\left(a_1x_i+a_2x_i^2+...+a_{Na}x_i^{Na}\right)y_i-\right.\\ \qquad\left.-b_0-b_1x_i-b_2x_i^2-...-b_{Nb}x_i^{Nb}+y_i\right]x_i^z y_i=0,\ 1\le z\le Na, \quad (5.19)\\ \sum_{i=0}^{m}\left[\left(a_1x_i+a_2x_i^2+...+a_{Na}x_i^{Na}\right)y_i-\right.\\ \qquad\left.-b_0-b_1x_i-b_2x_i^2-...-b_{Nb}x_i^{Nb}+y_i\right]\left(-x_i^r\right)=0,\quad 0\le r\le Nb. \quad (5.20)\end{cases}$$

Вопреки ставшей уже привычной читателю традиции демонстрировать структуру расширенной матрицы коэффициентов системы (5.19), (5.20) на этот раз воздержимся от этого из-за ее громоздкости. Если у читателя окажется под рукой лист бумаги достаточного размера он при желании может визуализировать такую схему самостоятельно. Если он до сих пор этому не научился – пусть пишет на себя жалобу. А мы приведем листинг подпрограммы

```
Procedure DrApMNK(X,Y:CoefR;Na,Nb:integer; var B,A:Coef);
var m,z,s,i:integer; u:real; D:Matr; Y2:CoefR;
  function XS(X:real; k:integer):real;
  var r:real; s:integer;
  begin
    r:=1;
    for s:=1 to k do r:=r*X; Xs:=r
  end;
begin
  m:=round(X[-1]);
  for s:=0 to m do Y2[s]:=sqr(Y[s]);
  for z:=1 to Na do
    begin
      for s:=1 to Na do
```

```
        begin
          u:=0;
          for i:=0 to m do
            u:=u+Xs(X[i],z+s)*Y2[i];
          D[z,s]:=u
        end;
      for s:=1 to Nb+1 do
        begin
          u:=0;
          for i:=0 to m do
            u:=u+Xs(X[i],z+s-1)*y[i];
          D[z,Na+s]:=-u
        end;
      u:=0;
      for i:=0 to m do
        u:=u+Xs(X[i],z)*Y2[i];
      D[z,Na+Nb+2]:=-u
    end;
  for z:=1 to Nb+1 do
    begin
      for s:=1 to Na do
        begin
          u:=0;
          for i:=0 to m do
            u:=u+Xs(X[i],z+s-1)*Y[i];
          D[z+Na,s]:=-u
        end;
      for s:=1 to Nb+1 do
        begin
          u:=0;
          for i:=0 to m do
            u:=u+Xs(X[i],z+s-2);
          D[Na+z,Na+s]:=u
        end;
      u:=0;
      for i:=0 to m do
        u:=u+Xs(X[i],z-1)*Y[i];
      D[Na+z,Na+Nb+2]:=u
    end;
  SystUr(Na+Nb+1,D,A); A[-1]:=Na; A[0];=1; b[-1]:=Nb;
  for s:=0 to Nb do
    B[s]:=A[Na+1+s]
end;
```

Предлагаем читателю проверить корректность подпрограммы. Уверены,

что у него неизбежно появится желание записать ее компактнее. Успеха!

Теперь, когда читатель вошел во вкус создания подпрограмм для работы с двумерными массивами ему можно посоветовать попробовать свои силы в программирование матричных операций. Но – это уже, скорее, профессиональные задачи (т. е. не для начинающих). Они, естественно, уже запрограммированы (следовательно, продать их не удастся). Именно поэтому и мы воздерживаемся от их рассмотрения в данном пособии. Да и места нет. Благодарим за внимание.

6. Сплайны

В предыдущем разделе был рассмотрен ряд способов аппроксимации функциональных зависимостей – интерполяционным полиномом, сглаживающим полиномом и дробно-рациональным выражением (отношением двух полиномов). При этом отмечалось, что полиномы высоких степеней могут иметь нежелательные свойства. Например, интерполяционный полином в промежутках между узлами интерполяции может вести себя вовсе не так, как это хотелось бы исследователю. Значительные возможности для улучшения качества аппроксимации функций открывает использование так называемых сплайнов, когда диапазон аппроксимации разбивается на ряд последовательно расположенных интервалов, в каждом из которых функция аппроксимируется полиномом той или иной степени. Наибольшей популярностью сейчас пользуются кубические сплайны, когда степень аппроксимирующих полиномов равна трем. К рассмотрению таких сплайнов мы и переходим.

6.1. Интерполяционные кубические сплайны

Пусть задано множество значений аргумента x (узлов аппроксимации)

$$x_0<x_1<x_2<..<x_{m-1}<x_m$$

и каждому узлу поставлено в соответствие значение функции

$$y_0, y_1, y_2, .., y_{m-1}, y_m.$$

Этот набор данных на плоскости *XOY* можно представить рядом точек с координатами (x_S, y_S), $0 \le s \le m$.

Функциональную зависимость y от x будем искать в виде кубического полинома типа

$$\psi_s(x) = a_{0,s} + a_{1,s}(x - x_{s-1}) + a_{2,s}(x - x_{s-1})^2 + a_{3,s}(x - x_{s-1})^3,\ 1 \le s \le m \quad (6.1)$$

Первая и вторая производные от (6.1)

$$\psi_s'(x) = a_{1,s} + 2a_{2,s}(x - x_{s-1}) + 3a_{3,s}(x - x_{s-1})^2,\ 1 \le s \le m, \quad (6.2)$$

$$\psi_s''(x) = 2a_{2,s} + 6a_{3,s}(x - x_{s-1}),\ 1 \le s \le m. \quad (6.3)$$

Тут индекс s означает, что полином (6.1) относится к s-му интервалу

между узлами аппроксимации (s-ый интервал начинается в точке x_{S-1}, заканчивается – в x_S). Так, что полиномов (6.1) будет m – по одному на каждый из m интервалов.

Система полиномов (6.1) описывает сплайн (гибкую линейку), которая проходит через каждую из упомянутых точек на плоскости XOY. Коэффициенты $a_{0,s}, a_{1,s}, a_{2,s}, a_{3,s}$ полинома (6.1) подбираются таким образом, чтобы обеспечить на интервале x_0 .. x_m непрерывность не только аппроксимирующей функции (6.1), но и ее первой (6.2) и второй (6.3) призводной.

Непрерывность (6.1) обеспечивается тем, что сплайн должен проходить через каждую из упомянутых точек, а каждая точка (кроме крайних) является концом одного и началом следующего интервала. Так, что условие непрерывности на левом конце s-го интервала (в точке x_{s-1}) требует, чтобы

$$\psi_s(x_{s-1}) = y_{s-1},$$

то есть согласно (6.1)

$$a_{0,s} = y_{s-1},\ 1 \le s \le m. \qquad (6.4)$$

На правом конце s-го интервала сплайн должен пройти через точку с координатами (x_S, y_S), значит, согласно (6.1)

$$a_{0,s} + a_{1,s}h_s + a_{2,s}h_s^2 + a_{3,s}h_s^3 = y_s,\ 1 \le s \le m. \qquad (6.5)$$

где h_S-ширина s-го интервала, $h_s = x_s - x_{s-1}$.

Условие непрерывности первых производных на стыках интервалов (в точке x_s, которая является концом s-го и началом (s+1)-го интервалов) согласно (6.2)

$$a_{1,s} + 2a_{2,s}h_s + 3a_{3,s}h_s^2 = a_{1,s+1},\ 1 \le s \le m-1, \qquad (6.6)$$

Для вторых производных аналогично согласно (6.3)

$$a_{2,s} + 3a_{3,s}h_s = a_{2,s+1},\ 1 \le s \le m-1. \qquad (6.7)$$

Уравнение (6.4) определяет коэффициенты, первым индексом которых есть 0, итак $a_{0,s}$ ($1 \le s \le m$) можно считать уже определенными. А уравнения (6.5)-(6.7)

можно рассматривать как систему из (3m-2) уравнений относительно 3m неизвестных коэффициентов $a_{1,s}, a_{2,s}, a_{3,s}$, $1 \le s \le m$.

Если упомянутую систему дополнить двумя уранениями граничных условий(речь о них пойдет ниже), то будем иметь 3m уравнений относительно 3m неизвестных, которую можно попробовать решить.

Определим сначала $a_{3,s}$ из (6.7)

$$a_{3,s} = \frac{a_{2,s+1} - a_{2,s}}{3h_s} \tag{6.8}$$

и подстаим (6.8) в (6.5) и (6.6), избавившись в (6.5) от $a_{0,S}$ в соответствии (6.4). После упрощения получим

$$3a_{1,s}h_s + 2a_{2,s}h_s^2 + a_{2,s+1}h_s^2 = 3\left(y_s - y_{s-1}\right), \tag{6.9}$$

$$a_{1,s} + a_{2,s}h_s + a_{2,s+1}h_s = a_{1,s+1} \tag{6.10}$$

Тогда из (6.9) определям $a_{1,S}$

$$\alpha_{1,\mathrm{s}} = \frac{y_s - y_{s-1}}{h_s} - \frac{2}{3}\alpha_{2,s}h_s - \frac{1}{3}\alpha_{2,s+1}h_s. \tag{6.11}$$

Подставляем (6.11) в (6.6) вместо $a_{1,S}$ и $a_{1,S+1}$ и упрощаем

$$h_s a_{2,s} + e_s a_{2,s+1} + h_{s+1}a_{2,s+2} = r_s, 1 \le s \le m-1, \tag{6.12}$$

где

$$e_s = 2\left(h_s + h_{s+1}\right),$$

$$r_s = 3\left(\frac{y_{s+1} - y_s}{h_{s+1}} - \frac{y_s - y_{s-1}}{h_s}\right).$$

Система (6.12) является трехдиагональной, и мы будем ее решать методом прогонки. В процессе прямого хода будем приводить уравнения этой системы к виду

$$a_{2,s+1} = A_s a_{2,s+2} + B_s, \ 1 \le s \le m-1. \tag{6.13}$$

Подставляем (6.13) в (6.12) вместо $a_{2,s}$. Результат представим в форме

$$\alpha_{2,s+1} = \frac{-h_{s+1}}{e_s + A_{s-1}h_s}\alpha_{2,s+2} + \frac{r_s - B_{s-1}h_s}{e_s + A_{s-1}h_s}. \tag{6.14}$$

Сравнивая (6.13) с (6.14), получаем рекуррентные формулы для A_s и B_s

$$A_s = \frac{-h_{s+1}}{e_s + A_{s-1}h_s}, \tag{6.15}$$

$$B_s = \frac{r_s - B_{s-1}h_s}{e_s + A_{s-1}h_s}. \tag{6.16}$$

Чтобы формулами (6.15), (6.16) можно было воспользоватся, надо сначала определить A_0 и B_0. Используем для этого граничное условие на левом конце сплайна. Рассмотрим два простейших варианта условий на концах сплайна: 1-го типа, когда задается значение первой производной и 2-го типа – со значением второй производной на соответствующем конце.

Так, например, если на левом конце сплайна первая производная равна y'_0, то, в соответствии с (3.9) для точки X_0 при s=1 будем иметь

$$a_{1,1} = y'_0. \tag{6.17}$$

Подставляем (6.17) в (6.16) при s=1

$$\frac{y_1 - y_0}{h_1} - \frac{2}{3}\alpha_{2,1}h_1 - \frac{1}{3}\alpha_{2,2}h_1 = y'_0,$$

откуда

$$\alpha_{2,1} = -\frac{1}{2}\alpha_{2,2} + \frac{\frac{3}{2}\left(\frac{y_1 - y_0}{h_1} - y'_0\right)}{h_1}. \tag{6.18}$$

Сравниваем (6.18) с (6.13) при s=0, откуда

$$\begin{cases} A_0 = -\dfrac{1}{2}, & (6.19) \\ B_0 = \dfrac{\frac{3}{2}\left(\frac{y_1 - y_0}{h_1} - y'_0\right)}{h_1}. & (2.20) \end{cases}$$

Если же на левом конце сплайна задано значение второй производной, то есть y''_0, то согласно (6.3) для точки X_0 при s=1 будем иметь

$$2a_{2,1} = y''_0. \tag{6.21}$$

Сравнивая (6.21) с (6.13) при s=0, получим

$$\begin{cases} A_0 = 0, & (6.22) \\ B_0 = \dfrac{y_0^{''}}{2}. & (6.23) \end{cases}$$

Имея A_0, B_0, по формулам (3.22), (3.23) последовательно определяются A_s, B_s, $1 \le s \le m$-1. При $s=m$-1 формула (6.13) дает

$$a_{2,m} = A_{m-1} a_{2,m+1} + B_{m-1}. \quad (6.24)$$

Чтобы начать обратный ход воспользуемся граничным условием на правом конце сплайна. Когда на правом конце имеем граничное условие I рода

$$\psi_m^{'}(x_m) = y_m^{'}. \quad (6.25)$$

Если в (6.25) подставить (6.2) при $s=m$, то получим

$$a_{1,m} + 2a_{2,m}h_m + 3a_{3,m}h_m^2 = y_m^{'}. \quad (6.26)$$

Когда мы теперь заменим в (6.26) $a_{1,m}$ по формуле (6.11) при $s=m$, а $a_{3,m}$ по формуле (6.8) при $s=m$, то после упрощения будем иметь

$$\alpha_{2,m}h_m + 2\alpha_{2,m+1}h_m = 3\left(y'_m - \frac{y_m - y_{m-1}}{h_m} \right). \quad (6.27)$$

Решаем систему уравнений, которую образуют (6.24) и (6.27)

$$\alpha_{2,m+1} = \frac{3\left(y'_m - \dfrac{y_m - y_{m-1}}{h_m} \right) - B_{m-1}h_m}{(A_{m-1} + 2)h_m}. \quad (6.28)$$

Когда же на правом конце сплайна имеем граничное условие II рода

$$\psi_m^{''}(x_m) = y_m^{''}, \quad (6.29)$$

то, учитывая (6.3), имеем

$$2a_{2,m} + 6a_{3,m}h_m = y_m^{''}, \quad (3.37)$$

а если учесть (6.8) при $s=m$, то получим

$$\alpha_{2,m+1} = \frac{y''_m}{2}. \quad 6.31)$$

После определения $a_{2,m+1}$ обратный ход реализуем по формуле (6.13), изменяя s от $s=m$-1 до 1 включительно. Этим завершается определение массива коэффициентов $a_{2,S}$, $1 \le s \le m$. По формулам (6.8) и (6.11) для всех нужных s (от 1

до m) определяются $a_{3,S}$ и $a_{1,S}$ (коэффициенты $a_{0,S}$ были определены формулой (6.4)).

Описанный алгоритм реализуется подпрограммой `SplineN`.

```
procedure SplineN
(X,Y:CoefR;Ngl,Ngr:integer;D0,Dm:real;A0,A1,A2,A3:CoefR);
var m,s:integer;
    H,H1,E,R,V,Z:real;
    A,B:CoefR;
begin
  m:=round(X[-1]);
  case Ngl of
    1: begin
         H:=X[1]-X[0]; A[0]:=-0.5;
         B[0]:=1.5*((Y[1]-Y[0])/H-D0)/H
       end;
    2: begin
         A[0]:=0; B[0]:=D0/2
       end
  end;
  for s:=1 to m-1 do
    begin
      H:=X[s]-X[s-1]; H1:=X[s+1]-X[s];
      E:=2*(H+H1);
      R:=3*((Y[s+1]-Y[s])H1-(Y[s]-Ys[S-1])/H);
      Z:=E+A[s-1]*H; A[s]:=-H1/Z;
      B[s]:=(R-B[s-1]*H)/Z
    end;
  case Ngr of
    1: begin
         H:=X[m]-X[m-1];
         A2[m+1]:=(3*(Dm-(Y[m]-Y[m-1])/H)-B[m-1]*H)/
         ((A[m-1]+2)*H)
       end;
    2: A2[m+1]:=Dm/2
  end;
  for s:=m-1 downto 0 do
    A2[s+1]:=A[s]*A2[s+2]+B[s];
  for s:=1 to m do
    begin
      H:=X[s]-X[s-1];
      A1[s]:=(Y[s]-Y[s-1])/H-2/3*A2[s]*H-1/3*A2[s+1]*H;
      A3[s]:=(A2[s+1]-A2[s])/(3*H);
```

```
      A0[s]:=Y[s-1]
    end;
  A0[-1]:=m
end;
```

После того, как сплайн сформирован (то есть сформированы массивы `A0, A1, A2, A3`), интерполяция выполняется подпрограммой-функцией `SplintN`, которая фактически реализует формулу (3.8), определяя сначала значение *S*-номер интервала, в который попадает заказанное значение *X*.

```
function SplintN(A0,A1,A2,A3,Mx:CoefR; X:real):real;
var m,s:integer;
  begin
    s:=1;
    while Mx[s-1] < x then inc(s); x:=x-Mx[s-1];
    SplintN:=((A3[s]*x+A2[s])*x+A1[s])*x+A0[s]
  end;
```

При практическом использовании часто возникает ситуация, когда узлы аппроксимации создают арифметическую прогрессию с постоянным шагом h_S=const. Тогда можно обойтись без массива `Mx`, а чтобы еще упростить задачу, можно за узлы аппроксимации принять номера этих узлов, т.е. нормировать `X` таким образом, чтобы он принимал значения в диапазоне 0..*m*.

Если кроме того рассматривать массивы `A0, A1, A2, A3` как глобальные, а граничные условия на концах сплайна по умолчанию считать условиями II рода при нулевых значениях вторых производных, то упрощенный вариант подпрограммы `SplineN` можно представить, как подпрограмму `Spline`.

```
procedure Spline(Y:CoefR);
var m,s:integer;
    Z,R:real;
begin
  m:=round(Y[-1]); A0[-1]:=m;A0[0]:=0;
  A1[0]:=0;
  for s:=1 to m-1 do
    begin
      R:=3*(Y[s+1]-2*Y[s]+Y[s-1]);
```

```
      Z:=4+A0[s-1];A0[s]:=-1/Z;
      A1[s]:=(R-A1[s-1])/Z
    end;
  A2[m+1]:=0;
  for s:=m-1 downto 0 do
    A2[s+1]:=A0[s]*A2[s+2]+A1[s];
  for s:=1 to m do
    begin
      A1[s]:=Y[s]-Y[s-1]-(2*A2[s]+A2[s+1])/3;
      A3[s]:=(A2[s+1]-A2[s])/3;A0[s]:=Y[s-1]
    end;
end;
```

Функция `SplintN` в подобной ситуации принимает вид (для нормированного *X*)

```
function Splint(x:real):real;
var m,s:integer;
begin
  s:=trunc(x)+1; x:=frac(x);
  Splint:=((A3[s]*x+A2[s])*x+A1[s])*x+A0[s]
end;
```

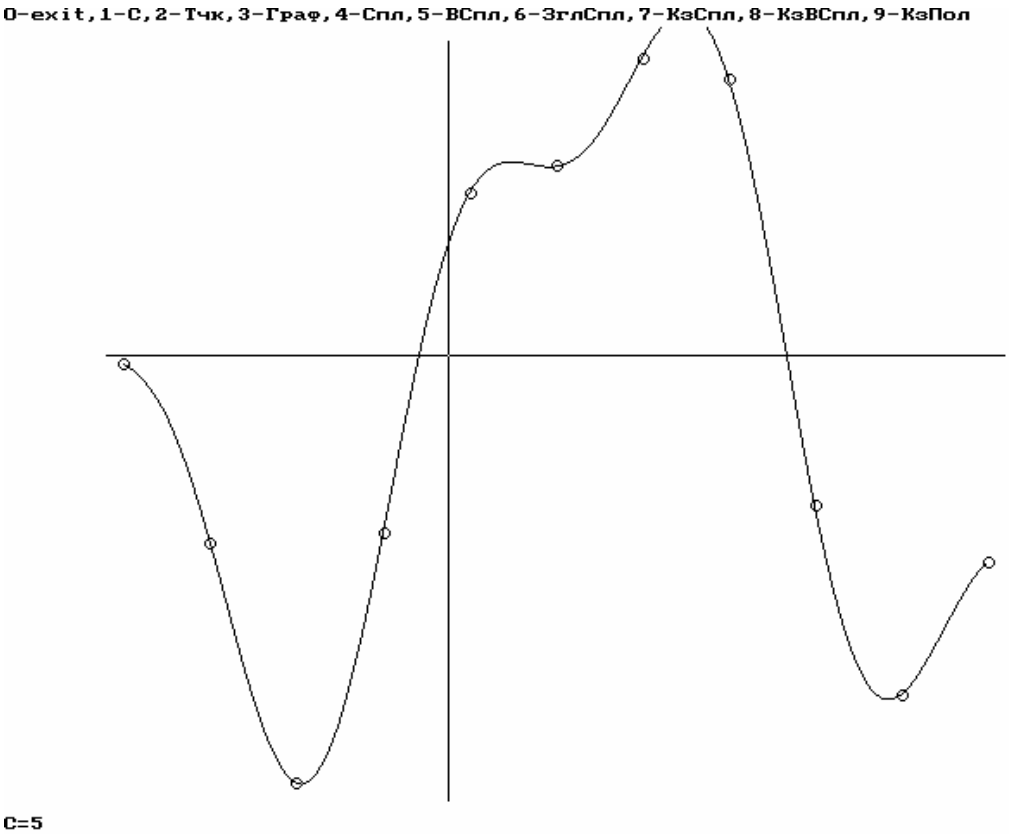


Рис. 3.2. Аппроксимация при помощи интерполяционного кубического сплайна.

6.2. Сглаживающие кубические сплайны

Как и в предыдущих параграфах будем считать, что задано множество узлов аппроксимации

$$x_s = sh,\ s=0,1,2,\ldots,m \text{ (шаг h - постоянный)}$$

и соответствующих им значений функций

$$y_s,\ s=0,1,2,\ldots,m.$$

Ставится задача сформировать аппроксимирующую функцию в виде кубического сплайна типа

$$\psi_s(x) = a_{0,s} + a_{1,s}(x - x_{s-1}) + a_{2,s}(x - x_{s-1})^2 + a_{3,s}(x - x_{s-1})^3. \qquad (6.32)$$

От этого сплайна будем требовать, чтобы его график прошел как можно ближе к точкам (x_s,y_s), $0 \le s \le m$ и чтобы первая и вторая производные от сплайна были бы непрерывные в диапазоне $[x_0 \,..\, x_m]$.

Итак, сплайн должен сглаживать исследуемую зависимость. Сравнительно с интерполяционным кубическим сплайном, рассмотренным выше, данный сплайн должен формировать более градкую зависимость. Для определения коэффициентов сплайна (6.32) разложим исследуемую функцию $y(x)$ в ряд Тейлора в окрестности x_{s-1}, ограничившись первыми четырьмя слагаемыми

$$y(x) = y\Big|_{x=x_{s-1}} + \frac{dy}{dx}\Big|_{x=x_{s-1}} (x - x_{s-1}) + \frac{1}{2!}\frac{d^2y}{dx^2}\Big|_{x=x_{s-1}} (x - x_{s-1})^2 + \frac{1}{3!}\frac{d^3y}{dx^3}\Big|_{x=x_{s-1}} (x - x_{s-1})^3 \qquad (6.33)$$

Сравнивая (6.33) с (6.32), видим, что коэффициенты $a_{0,s}, a_{1,s}, a_{2,s}, a_{3,s}$ могут быть выражены таким образом

$$\begin{cases} a_{0,s} = y\Big|_{x=x_{s-1}}, \\ a_{1,s} = \dfrac{dy}{dx}\Big|_{x=x_{s-1}}, \\ a_{2,s} = \dfrac{1}{2}\dfrac{d^2y}{dx^2}\Big|_{x=x_{s-1}}, \\ a_{3,s} = \dfrac{1}{3!}\dfrac{d^3y}{dx^3}\Big|_{x=x_{s-1}}. \end{cases} \qquad (6.34)$$

Формула для a_0 в (6.34) требует, чтобы сплайн проходил через точку (x_{s-1}, y_{s-1}),

то есть был бы интерполяционным. Поскольку речь идет о сглаживающем сплайне, то это условие следует смягчить. Примем, что

$$a_{0,s} = \frac{y_{s-2} + 4y_{s-1} + y_s}{6}, \tag{6.35}$$

тогда как остальные коэффициенты определяем через разностные соотношения, которые приближенно представляют производные в формулах (3.41)

$$a_{1,s} = \frac{y_s - y_{s-2}}{2h}, \tag{6.36}$$

$$a_{2,s} = \frac{y_{s-2} - 2y_{s-1} + y_s}{2h^2}, \tag{6.37}$$

$$a_{3,s} = \frac{-y_{s-2} + 3y_{s-1} - 3y_s + y_{s+1}}{6h^3}, \tag{6.38}$$

где h – шаг по x (считаем его постоянным, что было отмечено в начале параграфа, $x_s = sh$)

$$h = x_s - x_{s-1}.$$

Целесообразность выбора именно таких формул для коэффициентов сплайна (6.32) будет подтверждена ниже.

Проверим сначала непрерывность функции (6.32) на границах интервалов (s-тый интервал находится между точками x_{s-1} и x_s).

На конце s-того интервала (в точке x_s) согласно формулам (6.32) и (6.35)-(6.38) получим

$$\psi_s(x_s) = \frac{y_{s-2} + 4y_{s-1} + y_s}{6} + \frac{y_s - y_{s-2}}{2h}h + \frac{y_{s-2} - 2y_{s-1} + y_s}{2h^2}h^2 + \\ + \frac{-y_{s-2} + 3y_{s-1} - 3y_s + y_{s+1}}{6h^3}h^3,$$

что после упрощения дает

$$\psi_s(x_s) = \frac{y_{s-1} + 4y_s + y_{s+1}}{6}. \tag{6.39}$$

В начале же (s+1)-го интервала (в той же точке x_s) будем иметь соответственно

$$\psi_{s+1}(x_s) = \frac{y_{s-1} + 4y_s + y_{s+1}}{6}. \qquad (6.40)$$

Тут $\psi_{s+1}(x_s) = a_{0,s+1}$, поскольку скобки в формуле (6.32) становятся равными нулю при замене s на s+1 и при $x=x_s$. Сравнение (6.39) и (6.40) подтверждает непрерывность функции на стыке s-того и (s+1)-го интервалов (в точке x_s). Продифференцируем (6.32) относительно x

$$\psi_s'(x) = a_{1,s} + 2a_{2,s}(x - x_{s-1}) + 3a_{3,s}(x - x_{s-1})^2. \qquad (6.41)$$

Вычислим значение $\psi_s'(x)$ на конце s-го интервала (в точке x_s)

$$\psi_s'(x_s) = \frac{y_s - y_{s-2}}{2h} + 2\frac{y_{s-2} - 2y_{s-1} + y_s}{2h^2}h + 3\frac{-y_{s-2} + 3y_{s-1} - 3y_s + y_{s+1}}{6h^3}h^2,$$

что после упрощения дает

$$\psi_s'(x_s) = \frac{y_{s+1} - y_{s-1}}{2h}. \qquad (6.42)$$

В начале (s+1)-го интервала

$$\psi_{s+1}'(x_s) = a_{1,s+1},$$

то есть, согласно (6.36) при замене s на s+1

$$\psi_{s+1}'(x_s) = \frac{y_{s+1} - y_{s-1}}{2h}. \qquad (6.43)$$

Сравнение (6.42) и (6.43) подтверждает непрерывность первой производной от сплайновой кривой на стыке s-го и (s+1)-го интервалов.

Вторая производная

$$\psi_s''(x) = 2a_{2,s} + 6a_{3,s}(x - x_{s-1}). \qquad (6.44)$$

На конце s-го интервала

$$\psi_s''(x) = 2\frac{y_{s-2} - 2y_{s-1} + y_s}{2h^2} + 6\frac{-y_{s-2} + 3y_{s-1} - 3y_s + y_{s+1}}{6h^3}h$$

или после упрощения

$$\psi_s''(x) = \frac{y_{s-1} - 2y_s + y_{s+1}}{h^2}. \qquad (6.45)$$

В начале (s+1)-го интервала

$$\psi_{s+1}''(x_s) = 2a_{2,s+1}$$

или, что то же самое, согласно (6.37) при замене s на s+1

$$\psi''_{s+1}(x) = \frac{y_{s-1} - 2y_s + y_{s+1}}{h^2} = 2a_{2,s+1}. \qquad (6.46)$$

Сравнение (6.45) и (6.46) подтверждает непрерывность второй производной на стыке s-го и (s+1)-го интервалов. Третья производная

$$\psi'''_s(x) = 6a_{3,s}. \qquad (6.47)$$

На конце s-го интервала

$$\psi'''_s(x_s) = 6a_{3,s}$$

или после подстановки (6.38)

$$\psi'''_s(x_s) = \frac{-y_{s-2} + 3y_{s-1} - 3y_s + y_{s+1}}{h^3}. \qquad (6.48)$$

В начале (s+1)-го интервала

$$\psi'''_{s+1}(x_s) = 6a_{3,s+1}$$

или, если учесть (6.38),

$$\Psi'''_{s+1}(x_s) = \frac{-y_{s-1} + 3y_s - 3y_{s+1} + y_{s+2}}{h^3}. \qquad (6.49)$$

Сравнение (6.48) с (6.49) показывает, что непрерывность третьей производной на стыках интервалов не гарантируется (график третьей производной, как это следует из (6.47) – это ступенчатая линия).

Рассмотрим теперь определение коэффициентов (6.35)-(6.38) на концах сплайна. В точке x_0 согласно (6.32)

$$\Psi_1(x_0) = a_{0,1}$$

или же после подстановки (6.35)

$$\Psi_1(x_0) = \frac{y_{-1} + 4y_0 + y_1}{6}. \qquad (6.50)$$

Но y_{-1} у нас нету. Так что нам надо его придумать. Как? А согласно граничному условию на левом конце сплайна.

Пусть на левом конце имеет место граничное условие 0-го рода (*Ngl*=0), а именно – условие прохождения сплайновой кривой через точку (x_0,y_0). Тогда

$$y_0 = \frac{y_{-1} + 4y_0 + y_1}{6},$$

откуда (Ngl=0)

$$y_{-1} = 2y_0 - y_1. \qquad (6.51)$$

Очередной вариант – на левом конце задано значение первой производной $y_0^{'}$, то есть граничное условие 1-го рода (*Ngl*=1). Согласно (12) при *s*=0

$$\frac{y_1 - y_{-1}}{2h} = y_0^{'},$$

откуда (*Ngl*=1)

$$y_{-1} = y_1 - 2hy_0^{'}. \qquad (6.52)$$

Следующий вариант – на левом конце задано значение второй производной $y_0^{''}$ (*Ngl*=2 – граничное условие 2-го рода). Согласно (6.46) при *s*=0

$$\frac{y_{-1} - 2y_0 + y_1}{h} = y^{''}{}_0,$$

откуда (*Ngl*=2)

$$y_{-1} = 2y_0 - y_1 + h^2 y_0^{''}. \qquad (6.53)$$

Граничное условие 3-го рода (*Ngl*=3) гарантирует повышенную гладкость сплайна на его начальном участке (третья производная на стыке 1-го и 2-го интервалов не терпит разрыва), то есть согласно (6.47)

$$a_{3,1} = a_{3,2}$$

или после подстановки (6.48)

$$-y_{-1} + 3y_0 - 3y_1 + y_2 = -y_0 + 3y_1 - 3y_2 + y_3,$$

Откуда (*Ngl*=3)

$$y_{-1} = 4y_0 - 6y_1 + 4y_2 - y_3. \qquad (6.54)$$

При граничных условиях 4-го рода (*Ngl*=4) требуют, чтобы выполнялось условие периодичности, т.е. значение самой функции (6.32) и ее первой и второй производных в точках x_0 и x_m соответственно совпадали. Итак,

$$\Psi_1(x_0) = \Psi_m(x_m)$$

или

$$a_{0,1} = a_{0,m} + a_{1,m}h + a_{2,m}h^2 + a_{3,m}h^3.$$

Подставляем значение коэффициентов

$$\frac{y_{-1} + 4y_0 + y_1}{6} = \frac{y_{m-2} + 4y_{m-1} + y_m}{6} + \frac{y_m - y_{m-2}}{2h}h + \\ + \frac{y_{m-2} - 2y_{m-1} + y_m}{2h^2}h^2 + \frac{-y_{m-2} + 3y_{m-1} - 3y_m + y_{m+1}}{6h^3}h^3$$

или после упрощения

$$y_{-1} + 4y_0 + y_1 = y_{m-1} + 4y_m + y_{m+1}. \qquad (6.55)$$

Условие непрерывности первых производных

$$\Psi_1'(x_0) = \Psi_m'(x_m),$$

или $$a_{1,1} = a_{1,m} + 2a_{2,m}h + 3a_{3,m}h^2.$$

И, наконец, после подстановки значений коэффициентов

$$\frac{y_1 - y_{-1}}{2h} = \frac{y_m - y_{m-2}}{2h} + 2\frac{y_{m-2} - 2y_{m-1} + y}{2h^2}h + 3\frac{-y_m + 3y_{m-1} - 3y_m + y_{m+1}}{6h^3}h^2$$

Упростим это выражение

$$-y_{-1} + y_1 = -y_{m-1} + y_{m+1}. \qquad (6.56)$$

Условие непрерывности второй производной

$$\Psi_1''(x_0) = \Psi_m''(x_m)$$

или

$$2\frac{y_{-1} - 2y_0 + y_1}{2h^2} = 2\frac{y_{m-2} - 2y_{m-1} + y_m}{2h^2} + 6\frac{-y_{m-2} + 3y_{m-1} - 3y_m + y_{m+1}}{6h^3}h.$$

Упрощаем

$$y_{-1} - 2y_0 + y_1 = y_{m-1} - 2y_m + y_{m+1}. \qquad (6.57)$$

Рассматриваем теперь (6.55)-(6.57) как систему уравнений.

$$y_{-1} + 4y_0 + y_1 = y_{m-1} + 4y_m + y_{m+1}; \qquad (6.58)$$

$$-y_{-1} + y_1 = -y_{m-1} + y_{m+1}; \qquad (6.59)$$

$$y_{-1} + 2y_0 + y_1 = -y_{m-1} - 2y_m + y_{m+1}. \qquad (6.60)$$

Вычитаем (6.60) из (6.58)

$$6y_0 = 6y_m,$$

откуда $(Ngr = Ngl = 4)$

$$y_0 = y_m. \tag{6.61}$$

Это очевидно – условие непрерывности периодической функции.

Вычитаем (6.59) из (6.58)

$$2y_{-1} + 4y_0 = 2y_{m-1} + 4y_m,$$

откуда $(Ngr = Ngl = 4)$

$$y_{-1} = y_{m-1} + 2y_m - 2y_0. \tag{6.62}$$

Складываем (6.58) и (6.59)

$$4y_0 + 2y_1 = 4y_m + 2y_{m+1},$$

откуда $(Ngr = Ngl = 4)$

$$y_{m+1} = 2y_0 + y_1 - 2y_m. \tag{6.63}$$

Рассмотрим теперь варианты граничных условий на правом конце сглаживающего сплайна.

$$Ngr = 0;\ \psi_m(x_m) = y_m.$$

Подставляем вместо $\psi_m(x_m)$ формулу (6.32) при $s = m,\ x = x_m$

$$a_{0,m} + a_{1,m}h_m + a_{2,m}h_m^{\ 2} + a_{3,m}h_m^{\ 3} = y_m,$$

С учетом (6.35) - (6.38) имеем:

$$\frac{y_{m-2} + 4y_{m-1} + y_m}{6} + \frac{y_m - y_{m-2}}{2} + \frac{y_{m-2} - 2y_{m-1} + y_m}{2} + \frac{-y_{m-2} + 3y_{m-1} - 3y_m + y_{m+1}}{6} = y_m$$

или после упрощения:

$$y_{m+1} = 2y_m - y_{m-1};\ (Ngr = 0) \tag{6.64}$$

$$Ngr = 1;\ \psi_m'(x_m) = y_m'.$$

Подставляем вместо $\psi'_m(x_m)$ его значения с (6.41) при $s = m,\ x = x_m$

$$a_{1,m} + 2a_{2,m}h_m + 3a_{3,m}h_m^{\ 2} = y_m'.$$

Подставляем значения коэффициентов из (6.36) - (6.38) при $s = m$

$$\frac{y_m - y_{m-2}}{2h} + \frac{y_{m-2} - 2y_{m-1} + y_m}{h} + \frac{-y_{m-2} + 3y_{m-1} - 3y_m + y_{m+1}}{2h} = y_m',$$

После упрощения получаем:

$$y_{m+1} = y_{m-1} + 2hy_m'; \ (N_{gr} = 1) \qquad (6.65)$$

$$Ngr = 2; \ \psi_m''(x_m) = y_m''.$$

Используем (6.44) при $S = m$, $x = x_m$

$$2a_{2,m} + 6a_{3,m}h_m = y_m'',$$

Подставляем значения коэффициентов из (6.37), (6.38) при $S = m$

$$\frac{y_{m-2} - 2y_{m-1} + y_m}{h^2} + \frac{-y_{m-2} + 3y_{m-1} - 3y_m + y_{m+1}}{h^2} = y_m'',$$

После упрощения:

$$y_{m+1} = 2y_m - y_{m-1} + h^2 y_m''; \ (Ngr = 2) \qquad (6.66)$$

$$Ngr = 3; \ \psi_{m-1}'''(x_{m-1}) = \psi_m'''(x_{m-1}).$$

Согласно (6.47) $6a_{3,m-1} = 6a_{3,m}$,

или после упрощения $a_{3,m-1} = a_{3,m}$.

Подставляем значение коэффициентов из (6.38)

$$-y_{m-3} + 3y_{m-2} - 3y_{m-1} + y_m = -y_{m-2} + 3y_{m-1} - 3y_m + y_{m+1},$$

откуда

$$y_{m+1} = 4y_m - 6y_{m-1} + 4y_{m-2} - y_{m-3}; \ (Ngr = 2) \qquad (6.67)$$

Описанный алгоритм реализуется для сглаживания массива в подпрограмме `SplSgl`.

```
procedure SplSgl(Y:Coefr;Ngl,Ngr:integer;D0,Dm:real;var
Ys:Coefr) ;
var m,s:integer;
    H,H2:real;
begin
  m := round (Y[-1]); Ys[-1]:= m; H:=1; H2:= sqr (H);
  if (Ngl=4) and (Ngl=4)
    then
      begin
        Y[-1]:=Y[m-1]+2*Y[m]+2*Y[0];
        Y[m+1]:= 2*Y[0]+Y[1] -2*Y[m]
      end
    else
```

```
      begin
        case Ngl of
          0: Y[-1]:= 2*Y[0]-Y[1];
          1: Y[-1]:= Y[1]-2*H*D0;
          2: Y[-1]:= 2*Y[0]-Y[-1]+H2*D0;
          3:Y[-1]:= 4*Y[0]-6*Y[1]+4*Y[2]-Y[3]
        end;
        case Ngl of
          0: Y[m+1]:= 2*Y[m]-Y[m-1];
          1: Y[m+1]:= Y[m-1]+2*H*Dm;
          2: Y[m+1]:= 2*Y[m]-Y[m-1]+H2*Dm;
          3: Y[m+1]:= 4*Y[m]-6*Y[m-1]+4*Y[m-2]-Y[m-3]
        end;
    end;
    for s:= 0 to m do
      Ys[s]:= (Y[s-1]+4*Y[s]+Y[s+1])/6
end;
```

Работа метода продемонстрирована в программе DemSpl, которая позволяет проводить сглаживание заранее определенных сплайнов. Это продемонстрировано на рис. 3.3, где а) – это кубический сплайн, который аппроксимирует заданные точки, б) и в) – сплайны, которые сглаживают кривые а) и б), соответственно.

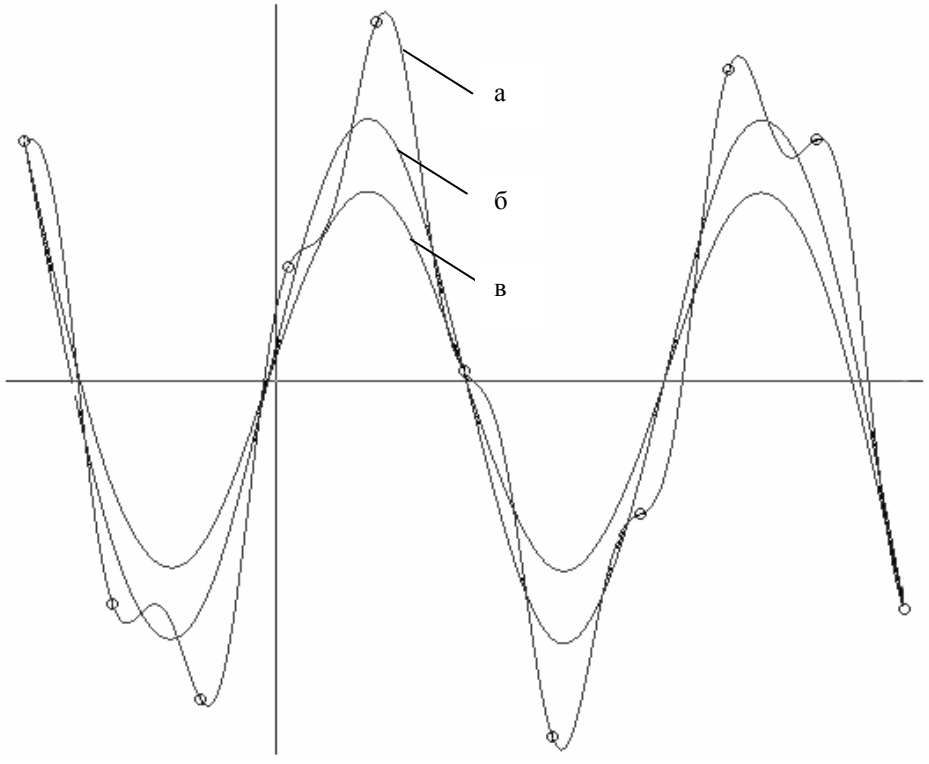


Рис. 3.3. Последовательное сглаживание кубическими сплайна а) сглаживаемая кривая; б) первое сглаживание; в) второе сглаживание.

6.3. Интерполяционные B-сплайны

Пускай, как и в предыдущем параграфе, задано множество значений аргумента x (узлов аппроксимации)

$$x_0<x_1<x_2<..<x_{m-1}<x_m$$

с постоянным шагом $h_s=x_s-x_{s-1}$=const.

Каждому значению x из указанного множества поставлено в соответствие значение функции

$$y_0, y_1, y_2, \dots, y_{m-1}, y_m.$$

Кубический B-сплайн (базовый сплайн) для интервала $[x_{s-2}..x_{s+2}]$ определяется таким образом

$$B_s(x)=\begin{cases}\frac{1}{6}(u_s+2)^3, & x\in[x_{s-2},x_{s-1}],\\ \frac{2}{3}-\frac{1}{2}\left(u_s^3+2u_s^2\right), & x\in[x_{s-1},x_s],\\ \frac{2}{3}+\frac{1}{2}\left(u_s^3-2u_s^2\right), & x\in[x_s,x_{s+1}],\\ \frac{1}{6}(2-u_s)^3, & x\in[x_{s+1},x_{s+2}],\\ 0 \text{ для других } x,\end{cases} \tag{6.68}$$

где $u_s=\dfrac{x-x_s}{h}$.

Вычислим значения первой, второй и третьей производных от сплайна (6.68) по x.

$$B_s'(x)=\begin{cases}\frac{1}{2h}(u_s+2)^2, & x\in[x_{s-2},x_{s-1}],\\ -\frac{1}{2h}\left(3u_s^2+4u_s\right), & x\in[x_{s-1},x_s],\\ \frac{1}{2h}\left(3u_s^2-4u_s\right), & x\in[x_s,x_{s+1}],\\ -\frac{1}{2h}(2-u_s)^2, & x\in[x_{s+1},x_{s+2}],\\ 0 \text{ для других } x.\end{cases} \tag{6.69}$$

$$B_s''(x)=\begin{cases}\dfrac{1}{h^2}(u_s+2)\ , & x\in[x_{s-2},x_{s-1}],\\ -\dfrac{1}{2h^2}(6u_s+4)=-\dfrac{1}{h^2}(3u_s+2), & x\in[x_{s-1},x_s],\\ \dfrac{1}{2h^2}(6u_s-4)=\dfrac{1}{h^2}(3u_s-2), & x\in[x_s,x_{s+1}],\\ \dfrac{1}{h^2}(2-u_s)\ , & x\in[x_{s+1},x_{s+2}],\\ 0 \quad \text{для других } x. \end{cases} \tag{6.70}$$

$$B'''_s(x)=\begin{cases}\dfrac{1}{h^3}, & x\in[x_{s-2},x_{s-1}],\\ -\dfrac{1}{3h^3}, & x\in[x_{s-1},x_s],\\ \dfrac{1}{3h^3}, & x\in[x_s,x_{s+1}],\\ -\dfrac{1}{h^3}, & x\in[x_{s+1},x_{s+2}],\\ 0 \ \text{для других } x. \end{cases} \tag{6.71}$$

Графически кубические B-сплайны и их производные имеют вид как на рис.6.4. Отрезок $[x_{s-2},\ x_{s+2}]$ называется носителем функции $Bs(x)$. Дополняем сетку $x_0<x_1<x_2<..<x_m$, де $x_0=a$, $x_m=b$, дополнительными узлами

$$x_{-3}<x_{-2}<x_{-1}<a<\ldots<b<x_{m+1}<x_{m+2}<x_{m+3}.$$

По расширенной сетке

$$x_{-3}<x_{-2}<x_{-1}<a<x_1<x_2<\ldots<x_{m-1}<b<x_{m+1}<x_{m+2}<x_{m+3}$$

можно построить семейство из $(m+3)$ кубических B-сплайнов типа (6.68) $Bs(X)$, $s=-1,0,1,...,m-1,m,m+1$. Это семейство образует базис в пространстве кубических сплайнов на отрезке $[a,b]$.

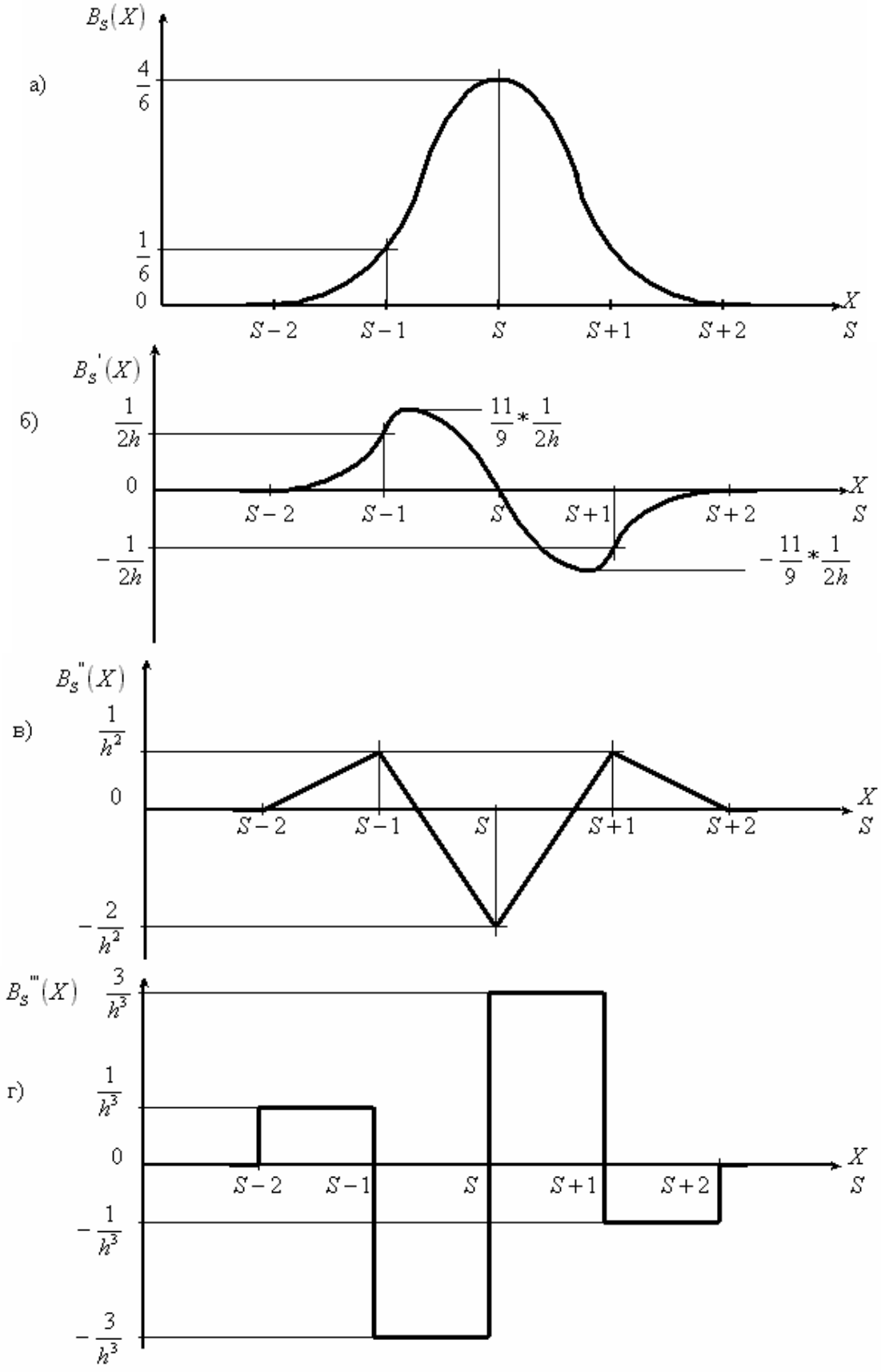


Рис.3.4. Графики B-сплайна и его производных

Тем самым, произвольный кубический сплайн $\psi(x)$, построенный на

отрезке [a,b] по сетке из (m+1) узлов, может быть представлен на этом отрезке в виде линейной комбинации

$$\psi(x)=\sum_{s=-1}^{m+1} b_s B_s(x). \qquad (6.72)$$

Итак, задача формирования B-сплайна сводится к определению b_s, $-1\le s\le m+1$. B-сплайн должен удовлетворять трем условиям:

1) быть непрерывным от x и обеспечивать равенство значений B-сплайна соответствующим значениям y_s в узлах интерполяции;
2) непрерывной на отрезке [a,b] должна быть первая производная от B-сплайна;
3) непрерывной на отрезке [a,b] должна быть вторая производная от B-сплайна.

Проверим сначала непрерывность отдельно взятого элемента сплайна, то есть $B_s(x)$.

- Точка $x = x_{S-2}$, $u_S = -2$

Левая граница $B_S(x_{S-2}) = 0$ (0 во всех других точках) .

Правая граница $B_s(x_{s-1}) = \frac{1}{6}(u_s+2)\Big|_{U_S=-2} = 0$.

Итак в точке $x = x_{S-2}$ B - сплайн является непрерывным.

- Точка $x = x_{S-1}$, $u_S = -1$.

Левая граница $B_s(x_{s-1}) = \frac{1}{6}(u_s+2)\Big|_{U_S=-1} = \frac{1}{6}$.

Правая граница $B_S(x_{S-1}) = \frac{2}{3} - \frac{1}{2}u_S{}^2(u_S+2)\Big|_{U_S=-1} = \frac{2}{3}-\frac{1}{2} = \frac{1}{6}$.

Выходит, что и в точке x_{S-1} B-сплайн является непрерывным.

- Точка $x = x_S$, $u_S = 0$.

Левая граница $B_S(x_S) = \frac{2}{3} - \frac{1}{2}u_S{}^2(u_S+2)\Big|_{U_S=0} = \frac{2}{3} = \frac{4}{6}$.

Правая граница $B_S(x_S) = \frac{2}{3} + \frac{1}{2}u_S{}^2(u_S - 2)\Big|_{U_S=0} = \frac{2}{3} = \frac{4}{6}.$

И в точке x_S непрерывность имеет место.

- Точка $x = x_{S+1}$, $u_S = 1$.

Левая граница $B_S(x_{S+1}) = \frac{2}{3} + \frac{1}{2}u_S{}^2(u_S - 2)\Big|_{U_S=1} = \frac{2}{3} - \frac{1}{2} = \frac{1}{6}.$

Правая граница $B_S(x_{S+1}) = \frac{1}{6}(2 - u_S)^3\Big|_{U_S=1} = \frac{1}{6}.$

- Точка $x = x_{S+2}$, $u_S = 2$.

Левая граница $B_S(x_{S+2}) = \frac{1}{6}(2 - u_S)^3\Big|_{U_S=2} = 0.$

Правая граница $B_S(x_{S+2}) = 0$ (во всех других точках)

Таким образом, мы убедились, что B-сплайн является непрерывной функцией на всем отрезке его существования. Проверим теперь непрерывность первой производной от B-сплайна (6,69).

- Точка $x = x_{S-2}$, $u_S = -2$.

Левая граница $B_S{}'(x_{S-2}) = 0$ (во всех других точках).

Правая граница $B_S{}'(x_{S-2}) = \frac{1}{2h}(u_s + 2)^2\Big|_{U_S=-2} = 0.$

Вывод: в точке x_{S-2}, $B_S{}'(x)$ является непрерывной функцией.

- Точка $x = x_{S-1}$, $u_S = -1$.

Левая граница $B_S{}'(x_{S-1}) = \frac{1}{2h}(u_s + 2)^2\Big|_{U_S=-1} = \frac{1}{2h}.$

Правая граница $B_S{}'(x_{S-1}) = \frac{1}{2h}(3u_s^2 + 4u_s)\Big|_{U_S=-1} = \frac{1}{2h}.$

В точке x_{S-2} первая производная от B-сплайна - непрерывная.

- Точка $x = x_S$, $u_S = 0$.

Левая граница $$B_s^{'}(x_s) = -\frac{1}{2h}(3u_s^2 + 4u_s)\Big|_{U_S=0} = 0.$$

Правая граница $$B_s^{'}(x_s) = \frac{1}{2h}(3u_s^2 + 4u_s)\Big|_{U_S=0} = 0.$$

В точке $x = x_s$ функция $B_S^{'}(x_S)$ непрерывна.

- Точка $x = x_{S+1}$, $u_S = 1$.

Левая граница $$B_s^{'}(x_{s+1}) = \frac{1}{2h}\left(3u_s{}^2 - 4u_s\right)\Big|_{U_S=1} = -\frac{1}{2}.$$

Правая граница $$B_s^{'}(x_{s+1}) = -\frac{1}{2h}(2-u_s)^2\Big|_{U_S=1} = -\frac{1}{2h}.$$

Непрерывность $B_S^{'}(x)$ в точке x_{S+1} имеет место.

- Точка $x = x_{S+2}$, $u_S = 2$.

Левая граница $$B_s^{'}(x_{s+2}) = -\frac{1}{2h}(2-u_s)^2\Big|_{U_S=2} = 0.$$

Правая граница $B_s^{'}(x_{s+2}) = 0$ (во всех других точках).

Итак, и в точке x_{s+2} наблюдаем непрерывность функции $B^{'}(x)$.

Проверим теперь непрерывность второй производной от B-сплайна (6.70).

- Точка $x = x_{S-2}$, $u_S = -2$.

Левая граница $B_S^{''}(x_{S-2}) = 0$ (во всех других точках) .

Правая граница $$B_s^{''}(x_{s-2}) = \frac{1}{h^2}(u_s + 2)\Big|_{U_S=-2} = 0.$$

Вторая производная в точке x_{S-2} непрерывна.

- Точка $x = x_{S-1}$, $u_S = -1$.

Левая граница $$B_s''(x_{s-2}) = \frac{1}{h^2}(u_s + 2)\Big|_{U_S=-1} = \frac{1}{h^2}.$$

Правая граница $$B_s''(x_{s-1}) = -\frac{1}{2h^2}(6u_s + 4)\Big|_{U_S=-1} = \frac{1}{h^2}.$$

Вторая производная в точке x_{S-1} непрерывна.

- Точка $x = x_S$, $u_S = 0$.

Левая граница $$B_s''(x_s) = -\frac{1}{2h^2}(6u_s + 4)\Big|_{U_S=0} = -\frac{2}{h^2}.$$

Правая граница $$B_s''(x_s) = -\frac{1}{2h^2}(6u_s - 4)\Big|_{U_S=0} = -\frac{2}{h^2}.$$

Итак, и в точке x_S имеем вторую непрерывную производную от B-сплайна.

- Точка $x = x_{S+1}$, $u_S = 1$.

Левая граница $$B_s''(x_{s-1}) = \frac{1}{2h^2}(6u_s - 4)\Big|_{U_S=1} = \frac{1}{h^2}.$$

Правая граница $$B_s''(x_{s-1}) = \frac{1}{h^2}(2 - u_s)\Big|_{U_S=1} = \frac{1}{h^2}.$$

И в точке $x = x_{S+1}$ вторая производная непрерывна.

- Точка $x = x_{S+2}$, $u_S = 2$.

Левая граница $$B_s''(x_{s+2}) = \frac{1}{h^2}(2 - u_s)\Big|_{U_S=2} = 0.$$

Правая граница $B_s''(x_{S+2}) = 0$ (во всех других точках)

На конце интервала (в точке x_{s+2}) вторая производная непрерывна.

Таким образом показано, что на всем интервале существования сам B-сплайн и его первая и вторая производная являются непрерывными функциями.

Непрерывность этих функций в пределах любого подынтервала гарантируется тем, что полином является непрерывной функцией своего аргумента.

Третья производная от B-сплайна, как это следует из формулы (6.71), не является непрерывной функцией – имеет разрывы на стыках подынтервалов.

Условие прохождения B-сплайна $\psi(x)$ через точку (x_s,y_s) на плоскости *XOY*

$$b_{s-1}B_{s-1}(x_s)+b_sB_s(x_s)+b_{s+1}B_{s+1}(x_s)=y_s,\ 0\le x\le m. \qquad (6.73)$$

Пользуясь формулами (6.68), определим $B_{s-1}(x_s)$, $B_s(x_s)$, и $B_{s+1}(x_s)$.

$$B_{s-1}(x_s)=\frac{1}{6}(2-1)^3=\frac{1}{6},$$
$$B_s(x_s)=\frac{2}{3}-\frac{1}{2}(0^3+2\cdot 0^2)=\frac{2}{3},$$
$$B_{s+1}(x_s)=\frac{1}{6}(-1+2)^3=\frac{1}{6}.$$

Итак, (6.73) получает следующий вид

$$b_{s-1}+4b_s+b_{s+1}=6y_s,\ 0\le s\le m. \qquad (6.74)$$

Это система из (m+1) уравнений относительно (m+3) неизвестных b_s, -1≤s≤m+1. Ее надо дополнить двумя уравнениями граничных условий – на левом и на правом концах B-сплайна. Пускай на левом конце имеем граничное условие I рода, то есть залано значение первой производной y_0' в точке x_0 (*Ngl*=1). Дифференцируем (6.73) при s=0

$$b_{-1}B'_{-1}(x_0)+b_0B'_0(x_0)+b_1B'_1(x_0)=y'_0,\ 0\le x\le m. \qquad (6.75)$$

Пользуясь формулами (6.69), определяем $B'_{-1}(x_0), B'_0(x_0)$ и $B'_1(x_0)$

$$B'_{-1}(x_0)=-\frac{1}{2h}(2-1)^2=-\frac{1}{2h},$$
$$B'_0(x_0)=-\frac{1}{2h}(3\cdot 0^2+4\cdot 0)=0,$$
$$B'_1(x_0)=\frac{1}{2h}(-1+2)^2=\frac{1}{2h}.$$

И тогда (6.75) приобретает вид (Ngl=1)

$$-b_{-1}+b_1=2hy'_0. \qquad (6.76)$$

Когда же на левом конце B-сплайна имеет место граничное условие II рода (*Ngl*=2), то, дифференцируя (6.73) дважды при *s*=0

$$b_{-1}B''_{-1}(x_0) + b_0 B''_0(x_0) + b_1 B''_1(x_0) = y''_0 . \qquad (6.77)$$

Определим $B''_{-1}(x_0), B''_0(x_0)$ и $B''_1(x_0)$ из (6.70)

$$B''_{-1}(x_0) = \frac{1}{h^2}(-1+2) \ = \frac{1}{h^2},$$

$$B''_0(x_0) = -\frac{1}{h^2}(3\cdot 0 \ +2) = -\frac{2}{h^2},$$

$$B''_1(x_0) = \frac{1}{h^2}(-1+2) \ = \frac{1}{h^2}.$$

Итак, (6.77) приобретает вид (*Ngl*=2)

$$b_{-1} - 2b_0 + b_1 = y''_0 h^2 . \qquad (6.78)$$

А теперь обратим внимание на то, что система (6.74) является трехдиагональной. Будем решать ее методом прогонки, последовательно исключая из каждого из уравнений $B_{s\text{-}1}$, то есть приводя очередное (*s*-тое) уравнение к виду

$$b_s = C_s b_{s+1} + D_s , \ 0 \le s \le m. \qquad (6.79)$$

Подставим (6.79) в (6.74) вместо $b_{s\text{-}1}$

$$C_{s-1}b_s + D_{s-1} + 4b_s + b_{s+1} = 6y_s .$$

Сводим подобные и решаем относительно b_s

$$b_s = \frac{-1}{C_{s-1}+4} b_{s+1} + \frac{6y_s - D_{s-1}}{C_{s-1}+4} . \qquad (6.80)$$

Сравнивая (6.80) с (6.79), видим, что

$$\begin{cases} C_s = \dfrac{-1}{C_{s-1}+4}, \ \ 0 \le s \le m+1, & (6.81) \\ D_s = \dfrac{6y_s - D_{s-1}}{C_{s-1}+4}, \ 0 \le s \le m+1. & (6.82) \end{cases}$$

Формулы (6.81), (6.82) – рекуррентные, чтобы они “заработали”, определим из “левого” граничного условия C_0 и D_0.

Уравнение (6.74) при *s*=0 имеет вид

$$b_{-1} + 4b_0 + b_1 = 6y_0. \tag{6.83}$$

При *Ngl*=1 уравнение (6.76) образует с (6.83) систему

$$\begin{cases} b_{-1} + 4b_0 + b_1 = 6y_0, \\ -b_{-1} + b_1 = 2hy_0^{'}. \end{cases}$$

Складываем уравнения этой системы

$$4b_0 + 2b_1 = 6y_0 + 2y_0^{'}h$$

или

$$b_0 = -0{,}5b_1 + \frac{3y_0 + hy_0^{'}}{2} \tag{6.84}$$

Сравниваем (6.84) с (6.79) при *s*=0 (*Ngl*=1)

$$\begin{cases} C_0 = -0.5, & (6.85) \\ D_0 = \dfrac{3y_0 + hy_0^{'}}{2}. & (6.85) \end{cases}$$

При *Ngl*=2 систему образуют уравнения (6.83) и (6.78)

$$\begin{cases} b_{-1} + 4b_0 + b_1 = 6y_0, \\ b_{-1} - 2b_0 + b_1 = y_0^{''}h^2. \end{cases}$$

Вычтем второе уравнение из первого

$$6b_0 = 6y_0 - y_0^{''}h^2$$

или

$$b_0 = 0 \cdot b_1 + y_0 - y_0^{''}\frac{h^2}{6} \tag{6.87}$$

Сравниваем (6.87) с (6.79) при *s*=0, откуда (*Ngl*=2)

$$\begin{cases} C_0 = 0, & (6.88) \\ D_0 = y_0 - y_0^{''}\dfrac{h^2}{6}. & (6.89) \end{cases}$$

При *s*=*m*-1 (6.79) приобретает вид

$$b_{m-1} = C_{m-1}b_m + D_{m-1}. \tag{6.90}$$

Граничное условие I рода на правом конце B-сплайна

$$b_{m-1}B_{m-1}^{'}(x_m) + b_mB_m^{'}(x_m) + b_{m+1}B_{m+1}^{'}(x_m) = y_m^{'}. \tag{6.91}$$

Легко увидеть из (6.70), что

$$B'_{m-1}(x_m) = -\frac{1}{2h},$$
$$B'_m(x_m) = 0,$$
$$B'_{m+1}(x_m) = \frac{1}{2h},$$

откуда (6.91) приобретает вид

$$-b_{m-1} + b_{m+1} = 2hy'_m. \qquad (6.92)$$

С другой стороны из (6.74) при $s=m$ имеем

$$b_{m-1} + 4b_m + b_{m+1} = 6y_m. \qquad (6.93)$$

Если из (6.93) вычесть (6.92), то

$$2b_{m-1} + 4b_m = 6y_m - 2hy'_0.$$

или, деля на 2,

$$b_{m-1} + 2b_m = 3y_m - hy'_0. \qquad (6.94)$$

Вычитаем теперь (6.94) из (6.90)

$$0 = (C_{m-1} - 2)b_m + D_{m-1} - 3y_m + hy'_m,$$

окуда (*Ngr*=1)

$$b_m = \frac{D_{m-1} - 3y_m + hy'_m}{2 - C_{m-1}}. \qquad .95)$$

При *Ngr*=2 граничное условие на правом конце B-сплайна

$$b_{m-1}B''_{m-1}(x_m) + b_mB''_m(x_m) + b_{m+1}B''_{m+1}(x_{m+1}) = y''_m. \quad 6.96)$$

Воспользуемся формулами (6.70) для определения вторых производных в (6.96)

$$B''_{m-1}(x_m) = \frac{1}{2}(2-1) = \frac{1}{h^2},$$
$$B''_m(x_m) = \frac{1}{2h^2}(6\cdot 0 - 4) = \frac{-2}{h^2},$$
$$B''_{m+1}(x_m) = \frac{1}{h^2}(-1+2) = \frac{1}{h^2}.$$

Следовательно, (6.96) приобретает вид

$$b_{m-1}\frac{1}{h^2} - b_m\frac{2}{h^2} + b_{m+1}\frac{1}{h^2} = y''_m$$

или

$$b_{m-1} - 2b_m + b_{m+1} = y_0^{''} h^2 . \qquad (6.97)$$

Вычитаем (6.97) из (6.93)

$$6b_m = 6y_m - y_m^{''} h^2 ,$$

откуда (*Ngr*=2)

$$b_m = y_m - y_m^{''} \frac{h^2}{6} . \qquad (6.98)$$

Обратный ход выполняется по формуле (6.79), в которой *s* меняется от *m*-1 до 0. Значение b_{m+1} тогда можно вычислить по формуле, которая получается из (6.93)

$$b_{m+1} = 6y_m - b_{m-1} - 4b_m . \qquad (6.99)$$

Аналогично из формулы (6.83)

$$b_{-1} = 6y_0 - b_1 - 4b_0 . \qquad (6.100)$$

Подпрограмма `Bspline`, которая формирует массив *B* коэффициентов b_s ($-1 \le s \le m+1$) может иметь такой вид при нормированном *X*, который меняется в диапазоне $x \in [0,m]$.

```
procedure BSpline (Y:Coefr; Ngl,Ngr:integer;D0,Dm:real;
var m:integer; var B:Coefr);
var s:integer;
    z,h:real;
    A,C:Coefr;
begin
  m:=round(Y[-1]); h:=1;
  case Ngl of
    1: begin A[0]:=-0.5; C[0]:=(3*Y[0]+h*D0)/2 end;
    2:begin A[0]:=0; C[0]:=Y[0]-D0*sqr(h)/6 end
  end;
  for s:=1 to m-1 do
    begin
      z:=A[s-1]+4;
       A[s]:=-1/z; C[s]:=(6*Y[s]-C[s-1])/z
    end;
  case Ngr of
    1: B[m]:=(-C[m-1]+3*Y[m]-h*Dm)/(2+A[m-1]);
    2: B[m]:=Y[m]-Dm*sqr(h)/6
  end;
```

```
  for s:=m-1 downto 0 do
    B[s]:=A[s]*B[s+1]+C[s];
  B[-1]:=6*Y[0]-B[1]-4*B[0];
  B[m+1]:=6*Y[m]-B[m-1]-4*B[m]
end;
```

Подпрограмма-функция `BSplint` для нормированного диапазона $X \in [0,m]$ будет выглядеть следующим образом

```
function BSplint(x:real):real;
var s:integer;
begin
  s:=trunc(x); x:=frac(x);
  if s>m
    then  BSplint:=(B[m-1]+4*B[m]+B[m+1])/6
    else
      BSplint:= B[s-1]*(1-x)*sqr(1-x)/6+
                B[s]*(2/3+0.5*sqr(x)*(x-2))+
                B[s+1]*(2/3-0.5*sqr(x-1)*(x+1))+
                B[s+2]*x*sqr(x)/6
end;
```

6.4. Сравнение обычного интерполяционного кубического сплайна и интерполяционного B-сплайна

Обычный интерполяционный сплайн является совокупностью кубических полиномов для каждого из подынтервалов. Интерполяционный B-сплайн в каждой точке (на каждом из подынтервалов) является суммой кубических полиномов, следовательно, и сам является кубическим полиномом. Поскольку при заданных граничных условиях, как обычный, так и B-сплайны определяются однозначно, то очевидно, что на каждом из подынтервалов мы будем иметь один и тот же кубический полином, т.е. обычный сплайн и B-сплайн в математическом плане являются полностью эквивалентными. Следовательно, при необходимости можно пересчитать один в другой и наоборот.

Для обычного кубического интерполяционного сплайна функция `Splint` как было показано раньше, имеет вид

```
function Splint(x:real):real;
var s:integer;
begin
  s:=trunc(x)+1;  x:=frac(x);
  Splint:=((A3[s]*x+A2[s])*x+A1[s])*x+A0[s]
end;
```

Для B-сплайна аналогично

```
function BSplint(x:real):real;
var s:integer;
begin
  s:=trunc(x)+1;  x:=frac(x)
  BSplint:=B[s-1]*sqr(1-x)*(1-x)/6+B[s]*(2/3+0.5*sqr(x)*
           (x-2))+B[s+1]*(2/3-0.5*sqr(x1)*(x+1))+
           B[s+2]*sqr(x)*x/6
end;
```

В этих подпрограммах x считается нормированным ($0 \le x \le m$) и, кроме того, принимаем, что соответствующие обычные и B-сплайны являются интерполяционными и сформированы при одинаковых граничных условиях.

Раскроем скобки и приведем подобные в формуле для `Bsplint`

$$\texttt{Bsplint:=}x^3\{\texttt{-B[s-1]/6+B[s]/2-B[s+1]/2+B[s+2]/6}\}+x^2\{\texttt{B[s-1]/2-B[s]+B[s+1]/2}\}+x\{\texttt{-B[s-1]/2+B[s+1]/2}\}+\{\texttt{B[s-1]/6+B[s]*2/3+B[s+1]/6}\}$$

Сравнивая это выражение с аналогичным выражением из *Splint*, приходим к выводу, что

$$\begin{aligned} &\texttt{A3[s+1]=(-B[s-1]+3·B[s]-3·B[s+1]+B[s+1])/6;}\\ &\texttt{A2[s+1]=(B[s-1]-2·B[s]+ B[s+1])/2;}\\ &\texttt{A1[s+1]=(B[s+1]-B[s-1])/2;}\\ &\texttt{A0[s+1]=(B[s-1]+4·B[s]+ B[s+1])/6;} \end{aligned} \qquad (6.101)$$

Вычислим `3·A0[s+1]-A2[s+1]`. Будем иметь

$$\texttt{3·A0[s+1]-A2[s+1]=3·B[s]},$$

откуда

```
B[s]=A0[s+1]-A2[s+1]/3.
```
(6.102)

Вычитаем `4/6 B[s]` из `A0[s+1]`

```
A0[s+1]-(4/6)·B[s]=(B[s-1]+B[s+1])/6
```

или то же самое

```
6A0[s+1]-4B[s]=B[s-1]+B[s+1].
```
(6.103)

Из $A1[s+1]$ можно записать

```
2A1[s+1]=B[s+1]-B[s-1].
```
(6.104)

Складывая (3.110) и (3.111) получим

```
B[s+1]=3A0[s+1]-2B[s]+ A1[s+1].
```
(6.105)

Из `A1[s+1]` получаем

```
B[s-1]:=B[s+1]-2A1[s+1].
```
(6.106)

И, наконец, из `A3[s+1]` имеем

```
B[s+2]:=6A3[s+1]+B[s-1]-3B[s]+3B[s+1].
```
(6.107)

Формулы (6.102), (6.105)-(6.107), если их применить последовательно, дают возможность пересчитать значения `A3[s+1], A2[s+1], A1[s+1], A0[s+1]` в значения `B[s-1], B[s], B[s+1], B[s+2]`, которые определяют значения B-сплайновой аппроксимирующей функции на интервале между точками X_S и $X_{S+!}$.

Формирование B-сплайна (массива `B:Coefr`) по массивам коэффициентов `A0, A1, A2, A3:Coefr` обычного интерполяционного сплайна можно представить в виде подпрограммы

```
procedure TranSplB(A0, A1, A2, A3:Coefr; var m:integer;
                   var B:Coefr);
var s:integer;
begin
  m:=round(A0[-1]);
  for s:=0 to m-1 do
    B[s]:=A0[s+1]-A2[s+1]/3;
  B[-s]:= B[1]-2*A0[1]);
  B[m]:=3*A0[m]-2*B[m-1]+ A1[m];
  B[m+1]:=6*A3[m]+B[m-2]-3*B[m-1]+3*B[m]
end;
```

Формирование же массивов A0, A1, A2, A3:Coefr по массиву B, сформированному под B-сплайн, можно представить процедурой TranBSpl в соответствии с формулами (6.101).

```
procedure TranBSpl(B:Coefr;m:integer;
                   var A0,A1,A2,A3:Coefr);
var s:integer;
begin
  A0[-1]):=m;
  for s:=0 to m do
    begin
      A3[s+1]:=(-B[s-1]+3*B[s]-3*B[s+1]+B[s+2])/6;
      A2[s+1]:=(B[s-1]-2*B[s]+B[s+1])/2;
      A1[s+1]:=(B[s+1]-B[s-1])/2;
      A0[s+1]:=(B[s-1]+4*B[s]+B[s+1])/6
    end
end;
```

Работу со сплайнами демонстрируют программы DemSpl и DemBSpl листинги которых приводятся в приложении.

7. Визуализация результатов расчетов

В большинстве выше рассмотренных алгоритмов результатом расчета является не отдельное число или конечный набор чисел (как, например, корни полинома или решение системы линейных алгебраических уравнений), а функции (интерполяционные и сглаживающие полиномы, сплайны, B – сплайны и т. д.). Причем функции могут зависеть не только от одного, но и от большего количества независимых переменных. Аппроксимировать функциональные зависимости, исследуя их формулы или соответствующие таблицы при наличии компьютера – нелогично. Современный компьютер представляет почти неограниченные возможности для визуализации функций в виде графиков, систем изолиний, аксонометрических изображений и т. п. Ниже мы остановимся на рассмотрении простейших идей и алгоритмов решения подобных задач.

7.1. Упрощенный графический интерфейс и формирование графиков

Мы не собираемся конкурировать с *Windows* и другими операционными системами, но для решения многих интересных задач можно ограничиться элементарными средствами – “не святые горшки обжигают”.

Итак, что в первою очередь необходимо включить в заявленный интерфейс? Ну, инициировать графический режим нужно? Нужно. Сформировать меню и подкрепить его информационной строкой, которая бы демонстрировала текущие значения переменных, предлагаемых для изменения. Тоже нужно вводить в интерактивном режиме значения переменных, фигурирующих в текущем меню. Рассчитывать параметры системы координат, которая позволяла бы отображать на заданную область экрана необходимые значения переменных x и y, так чтобы формируемый график или годограф оказался в нужном месте экрана и имел бы желательные размеры. Рисовать оси системы координат и формировать график. Как видите, заявка не так уж скромна. Но удовлетворить ее можно – и средства решения сосредоточены в модуле `Serv`, полный листинг которого приводится в Приложении. Ниже мы

кратко прокомментируем содержимое этого модуля. Начнем с интерфейсной части.

```
uses Crt, Graph;
```

Модуль Crt поддерживает текстовый режим программы, модуль Graph – графический. Поскольку рассматриваемый интерфейс графический, то и реализовать его без модуля Graph невозможно, а модуль Crt понадобится для реализации процедуры GotoXY (см. ниже).

```
Const Hj: real=2.1; C: integer=15;
```

Первая типизованная константа *Hj* равна отношению расстояния между соседними пикселями по вертикали (шаг развертки по вертикали) к таковому по горизонтали (шаг развертки по горизонтали). Отношение этих шагов изменяется в зависимости от выбранного драйвера и режима его работы. В процессе инициирования графического режима величина Hj будет рассчитываться по результатам вызова процедуры GetAspectRatio.

C – цвет изображения по умолчанию считаем C равным 15, т. е. цвет – ярко белый.

```
Type Coef=array[-1..31] of real;
     Coefr=array[-1..201] of real;
     CoefL=array[-1..601] of real;
```

Эти три типа ранее уже неоднократно использовались. Здесь они как бы узакониваются.

```
Var L,H,L0,H0,Xu,Yu,X0,Y0,S,Z: integer;
```

Отметим пока, что L и H – ширина и высота в пикселях прямоугольника, отведенного на экране для формирования графика или годографа. S, Z – счетчики. Остальные переменные будут объясняться ниже.

```
Xn,Yn,Xk,Yk,X,Y,Dx,Dy,Xmin,Xmax,Ymin,Ymax: real;
```

Начнем с переменных X, Y. Это обычные *x* и *y* (аргумент и функция). Dx, Dy - шаг изменения *x* и *y* (при формировании таблиц, графиков и т. д.). Xmin, Xmax, Ymin, Ymax – идентификаторы говорят сами за себя. Xn, Xk, Yn, Yk – копии соответственно Xmin, Xmax, Ymin, Ymax,

предназначенные для разметки и оцифровки шкал, формирования координатных сеток. `Xn, Xk, Yn, Yk` в процессе решения перечисленных задач можно свободно корректировать (округляя, согласовывая), тогда как исходные `Xmin, Xmax, Ymin, Ymax` остаются нетронутыми.

```
T10: string[10]; Ts: string[80].
```

Это текстовые переменные, используемые при превращении чисел в текст (это, в первую очередь *T10*) и при формировании ранее упоминавшейся информационной строки (`Ts`).

```
J, J1, J2, J3: char;
```

Диалог с пользователем удобно организовать через систему меню разных уровней. Выбор той или иной опции предлагаемого меню пользователь осуществляет путем нажатия соответствующий клавиши, код которой запоминается в переменной типа `J, J1, J2, J3: char`. Чем больше цифра, тем, как правило, выше уровень вложенности меню. Так `J` используется для основного меню. `J1` – для подменю первого уровня, `J2` – второго и т. д. При необходимости, уровень вложенности можно организовать любой, “на первый случай” приведенных переменных достаточно.

Далее в интерфейсной части модуля *Serv* идет список заголовков подпрограмм – процедур. Прокомментируем их названия.

```
procedure Clear (X1, Y1, X2, Y2: integer);
```

Предназначена для стирания изображения в пределах прямоугольника, левый верхний угол которого имеет экранные координаты `X1, Y1`, а правый нижний угол соответственно `X2, Y2.`

```
procedure PutA;
```

Очищает прямоугольник с координатами *0, 0* (левый верхний угол), `GetMaxX-1, 12` (правый нижний угол). Это полоса высотой *12* пикселей (по высоте строки – меню) в верхней части экрана (чтобы освободить место для нового меню или какого-то иного сообщения).

```
procedure Ou(Text: string);
```

Выводит текст, переданный в качестве параметра, в верхнюю строку

экрана. Это может быть, например, очередное меню.

```
procedure Our(Text: string; var X: real);
procedure Oui(Text: string; var X: integer);
```

Осуществляют ввод в интерактивном режиме значения переменной `X: real (Our)` или `X: integer (Oui)`.

```
procedure Info;
```

Очищает место (полосу высотой 10 пикселей) в нижней части экрана и на освобожденное таким образом место выводит текст Ts информационной строки.

```
procedure X0Y0(God: Boolean);
```

Определяет параметры системы координат (`Xu, Yu, Dx, Dy, X0, Y0, Xn, Xk, Yn, Yk`). Исходным материалом для этой процедуры являются глобальные переменные `L, H, Xmin, Xmax, Ymin, Ymax`. Параметр `God` отвечает на вопрос – формируется ли система координат под годограф (`God=true`) или не под годограф (очевидно, под график – `God=false`). В первом случае предполагается , что шкалы по осям требуется согласовывать, во втором случае – в этом нет надобности.

```
procedure SystGoor;
```

Формирует горизонтальную и вертикальную оси системы координат (инверсным цветом).

```
procedure Graphic(Mo: CoefL; C: integer);
```

Формирует график по массиву ординат Mo цветом C.

```
procedure InGraph;
```

Инициализирует графический режим.

А теперь краткий комментарий к каждой из перечисленных процедур, тексты которых приводятся в разделе `Implementation модуля Serv`.

```
procedure Clear(X1,Y1,X2,Y2:integer);
begin
  SetViewPort(X1,Y1,X2,Y2,true);
```

```
  ClearViewPort;
  SetViewPort(0,0,GetMaxX-1,GetMaxY-1,false)
end;
```

Процедура устанавливает временное окно с координатами прямоугольной области, подлежащей очистке. Вытирает это окно (`ClearViewPort`). И затем восстанавливает окно размером во весь экран. Предполагается, что таковым было перед этим окно (по умолчанию графическое окно занимает весь экран).

Обратим внимания читателя на то обстоятельство, что процедурой Clear можно пользоваться только тогда, когда окном действительно является весь экран, в противном случае процедура `Clear` отменит текущее окно, распространит его на весь экран.

И еще одна небольшая особенность, на которою может оказаться полезным обратить внимание читателя. Стандартная процедура Турбо Паскаля `SetViewPort` “не любит”, когда ей велят установить окно, выходящее за рамки экрана. Она просто молча игнорирует такое указание. Иногда она отказывается устанавливать даже окна, края которых совпадают с границами экрана. Поэтому при распространии окна на весь экран горизонтальная и вертикальная координаты правого нижнего угла прямоугольника уменьшены на 1.

```
procedure PutA;
begin
  Clear(0,0,GetMaxX-1,10)
end;
```

Единственный смысл введения процедуры `PutA` заключается в том, что ее вызов короче чем вызов процедуры `Clear` в ее операторной части. Нам придется очень часто вызывать `PutA`, поэтому экономия на ее вызове оказывается весьма кстати.

```
procedure Our(Text:String; var X:real);
begin
  PutA;
```

```
  OutTextXY(150,0,'Введіть     '+Text);
  GotoXY(1,1);
  read(X)
end;
```

Процедура `Our` на очищенное предварительно процедурой `PutA` место в верхней части экрана выводит текст – приглашение `'Enter ' + Text`, который поясняет, что именно требуется вести. Процедура `GotoXY(1,1)` переводит текстовый курсор в левое верхнее знакоместо экрана (с координатами 1,1). Это необходимо чтобы эхо-повтор вводимого пользователем ответа (на сделанное предложение) оказался в левой верхней части экрана (где его легко отыскать, чтобы проконтролировать). Оператор `read(x)` считывает введенный ответ и заносит его в идентификатор `X` (формальный).

Процедура `Oui` отличается от `Our` только типом вводимой переменной (в `Oui` – это переменная `X: integer`).

```
procedure Info;
begin
  Clear(0,GetMaxY-12,GetMaxX-1,GetMaxy-1);
  OutTextXY(0,GetMaxY-12,Ts)
end;
```

Тут автоматически (по умолчанию) носителем текста информационной строки является `Ts: string[80]`.

Чаще всего на экране при стандартном размере шрифта (по умолчанию) помещается максимум 80 символов. Из этого соображения и выбрана максимальная длина переменной *Ts*. Пользователь обязан позаботиться о том, чтобы к моменту вызова процедуры `Info` была полностью заготовлена (сформирована) переменная `Ts` соответствующего содержания.

```
procedure X0Y0(God:boolean);
begin
  Dx:=(Xmax-Xmin)/L; L0:=L;
  Dy:=(Ymax-Ymin)/H; H0:=H;
```

```
  if God then
    if Dy>Dx*Hj
      then
        begin
           L0:=round((Xmax-Xmin)/Dy*Hj);
           if L0<>0 then Dx:=(Xmax-Xmin)/L0
                    else  Dx:=0
        end
      else
        begin
          H0:=round((Ymax-Ymin)/(Dx*Hj));
          if H0<>0
            then Dy:=(Ymax-Ymin)/H0
            else  Dy:=0
        end;
    Xu:=(GetMaxX-L0) div 2; Yu:=(GetMaxY-H0) div 2;
    X0:=Xu-round(Xmin/Dx); Y0:=Yu+round(Ymax/Dy);
    Xn:=Xmin; Xk:=Xmax; Yn:=Ymin; Yk:=Ymax
end;
```

Здесь принято, что при формировании системы координат под график (`God=false`) размеры прямоугольника должны быть `L*H`, а для годографа (`God=true`) `L0*H0`. Итак `L0<=L, H0<=H`. Первоначально (на всякий случай, `L0:=L; H0:=H`). Затем по мере необходимости пересчитывается либо `L0` (когда `Dy>Dx*Hj`), либо `H0`, а заодно с ними корректируется соответственно `Dx` или `Dy`. После этого откорректированный прямоугольник помещается в центре экрана. Левый верхний угол этого прямоугольника оказывается в точке с координатами `Xu, Yu`. Оси же системы координат пересекаются в точке `X0, Y0`. Относительно `Xn, Xk, Yn, Yk` речь шла выше.

```
procedure SystCoor;
begin
  ClearDevice;
  SetWriteMode(1);
  Line(Xu-10,Y0,Xu+L0+10,Y0);
  Line(X0,Yu+H0+10,X0,Yu-10);
  SetWriteMode(0)
end;
```

Первая линия изображает горизонтальную ось, вторая – вертикальную.

Чтобы эти оси как-то отличались от остальных линий на формируемом прямоугольнике, они выступают на 10 пикселей за пределы заданного прямоугольника.

Цвет изображения – инверсный. Это обеспечивается процедурой `SetWriteMode`. Первый вызов этой процедуры устанавливает инверсный цвет второй – отменяет. Инверсный цвет обеспечивает возможность "примерки" системы координат (при повторным вызове `SystCoor` оси системы координат бесследно исчезают, не повреждая сформированного изображения).

```
procedure Graphic(Mo:CoefL; C:integer);
var x,y,s:integer;
begin
  if C>=0 then SetColor(C)
          else  SetWriteMode(1);
  MoveTo(Xu,Y0-round(Mo[0]/Dy));
  for s:=1 to L do
    begin
      x:=X0+s; y:=Y0-round(Mo[s]/Dy);
      LineTo(x,y);
      if C<0 then PutPixel(x,y,GetMaxColor-GetPixel(x,y))
    end;
  if C>=0 then SetColor(15)
          else SetWriteMode(0)
end;
```

Ширина графика предполагается равной `L` пикселей. Под номера пикселей от `0` до `L` формируется массив ординат `Mo` соответствующей функции (величина L заключается в нем в ячейке с номером `-1`). Цвет графика `C`. Если `C<0` цвет считается инверсным. График формируется в виде ломаной, причем шаг ломаной по горизонтали равняется 1 пикселю, т. е. является минимально возможным. Этим обеспечивается наивысшее достижимое (на данном экране, при данном драйвере и режиме) качество графика в смысле его детальности. Можно обратить внимание читателя на то обстоятельство, что при инверсном цвете (*C<0*) точки излома ломаной, т. е. точки стыка отдельных ее участков оказываются закрашенными дважды, а следовательно, не закрашенными.

Образующие таким образом “дырки” затыкает процедура PutPixel(X, Y, GetMaxColor - GetPixel(X, Y)).

```
procedure InGraph;
var E,Gd,Gm:integer;
    Aw,Bw:word;
begin
  Gd:=Detect;
  InitGraph(Gd,Gm,' ');
  E:=GraphResult;
  if E<>0
    then
      begin
        write(GraphErrorMsg(E));
        readln; Halt(1)
      end
    else
      begin
        GetAspectRatio(Aw,Bw);
        Hj:=Bw/Aw;
        L:=GetMaxX-139;
        H:=GetMaxY-59
       end
end;
```

Процедура InGraph пользуется услугами стандартной процедуры из модуля Graph, а именно InitGraph. Последней передается параметр Gd=Detect. Тем самым пользователь предоставляет компилятору самому принять решения о выборе подходящего драйвера и режима. В первых двух параметрах процедуры InitGraph будут возвращены коды выбранных драйвера и режима. Это в том случае, если графический режим процедуре InitGraph удается успешно реализовать. Но в общем случае успех не гарантирован. Результаты попытки фиксируются в переменной E. Если все в порядке, то E=0, иначе процедурой write (в текстовом режиме, поскольку в графический режим перейти не удалось!) выведется сообщение о причине неудачи.

Но если E=0, то вызывается процедура GetAspectRatio. По

результатам ее работы уточняется Hj. Ну, и заодно на всякий случай определяются значения L и H (по умолчанию). Текст модуля Serv заканчивается конструкцией

```
begin
  InGraph
end.
```

Это инициирующая часть модуля. Она инициализирует графический режим. Тем самым уже на стадии подключения модуля Serv (в конструкции uses Serv) автоматически устанавливается графический режим работы программы (снова же по умолчанию).

7.2 Оформление графиков и годографов (модуль GromS)

Листинг модуля GromS приводится в приложении. Любознательный читатель может самостоятельно разобраться в механизме его функционирования. Мы же ограничимся кратким анализом возможностей, которые он предоставляет в распоряжение пользователя, т.е. прокомментируем его содержание, объясним суть обозначений, используемых в опциях меню в его подпрограммах.

Начнем с интерфейсной части модуля, т.е. сначала.

```
Unit GromS;
Interface
uses Crt, Graph, Serv;
const
  Hx : real = 0.5;
  Hy : real =0.5;
  Se : integer = 5;
  Sd : integer = 2;
  Sh : integer = 2;
  Sv : integer = 2;
  Rs : integer = 3;
  Xsc : real = 0;
  Ysc : real = 0;
  Zmh : integer = -15;
  Zmv : integer = -10;
  Fr : integer = 1000;
```

```
procedure Mark;
procedure Net;
procedure Num;
procedure Inscr;
procedure Service;
```

Более подробно назначение и техника использования каждой из перечисленных выше процедур, входящих в модуль, будет рассмотрена ниже. Сейчас же только кратко обозначим назначение каждой из них.

`procedure Mark`

предназначена для нанесения маркировки (разметки) шкал на осях, что после соответствующей оцифровки позволяет воспринимать графики (годографы) количественно.

`procedure Net`

формирует координатную сетку.

`procedure Num`

проводит оцифровку шкал – напротив каждой (или через несколько) меток или линий координатной сетки проставляются числа, равные соответствующим значениям X и (или) Y.

`procedure Inscr`

позволяет помещать в нужных местах изображения надписи (формируются и перемещаются в желаемое место в интерактивном режиме).

`procedure Service`

- это процедура-диспетчер, с ее помощью пользователь получает доступ к процедурам, упомянутым выше.

А теперь несколько слов о константах модуля.

```
Hx: real=0.5;
```

Это шаг разметки (в `Mark`) или координатной сетки (в `Net`) или элементов оцифровки (в `Num`) по горизонтали.

```
Hy: real=0.5;
```

То же, что и `Hx`, но только по вертикали. Начальные значения этих и нижеследующих типизированных констант заданы условно. Конкретные

значения им присваиваются в процессе работы с процедурами модуля (под конкретную решаемую задачу).

```
Se: integer=5;
```

Количество символов (ширина) элемента оцифровки. Ну, например, элемент оцифровки – `0.25` содержит `5` символов (`Se=5`).

```
Sd: integer=2;
```

Это количество символов после десятичной точки (в элементе оцифровки).

Именно таково значение `Sd` в элементе оцифровки – `0.25`

```
Sh: integer=2;
```

Это шаг по горизонтали (в `Inscr`). Текст, формируемый в `Inscr`, перемещается в требуемое место на экране. Перемещение осуществляется дискретно, шаг за шагом. Так вот, очередной кадр смещается по горизонтали относительно предыдущего на `Sh` пикселей (вправо, если `Sh>0`, и влево, если `Sh<0`).

```
Sv: integer=2;
```

То же, что и Sh, но по вертикали.

```
Rs: integer=3;
```

Это (в `Mark`) – размер штриха, точнее, его половина. Штрих выступает за ось, на которой он нанесен, на `Rs` пикселей влево и `Rs` пикселей вправо (на вертикальной оси) или соответственно на `Rs` пикселей вверх и столько же вниз на горизонтальной оси.

```
Xsc: real=0;
```

Это (в `Num`) координата `X` (горизонтальная) для вертикальной линии, которую мы собираемся использовать в качестве шкалы (т.е. собираемся на ней нанести оцифровку). Это, в принципе, может быть любая вертикальная линия (крайняя левая, крайняя правая, центральная, соответствующая `X=0` и т. п.)

```
Ysc: real=0;
```

То же для горизонтальной линии, используемой в качестве шкалы.

```
Zmh: integer=-15;
```

Это смещение элементов оцифровки (в `Num`) по горизонтали (для вертикальной

шкалы). Если Zmh=0, то элементы оцифровки будут накладываться на оцифровываемую линию (заданную Xsc). Чтобы сдвинуть элементы оцифровки (все вместе относительно соответствующей оси), например, на 10 пикселей вправо, необходимо задать Zmh=10, если влево, знак Zmh изменяется на отрицательный.

```
Zmv: integer=-10;
```

Это смещение по вертикали элементов оцифровки (в Num). Для смещения вверх Zmv должно быть положительным, вниз – отрицательным. В процедуре Num и горизонтальная и вертикальная оцифровки могут при желании убираться (не повреждая оси). Но это возможно, если оцифровка убирается непосредственно после ее нанесения. Так что можно подбирать подходящее смещение, пробуя и убирая отдельные варианты.

Следует обратить внимание на то обстоятельство, что Zmh подбирается с учетом величины Se (см. выше). Чем больше Se, тем больше по модулю должно быть Zmv.

```
Fr: integer=200;
```

При формировании надписей (процедура Inscr) надпись формируется в верхнем левом углу экрана, и затем эта надпись “перегоняется” в нужное место на экране. При такой перегонке, перемещение надписи осуществляется путем последовательной смены кадров (каждый очередной кадрик проецируется смещенным в направлении перемещения). Если компьютер медленный, то надпись перемещается слишком медленно, на компьютере с высоким быстродействием изображение смещается настолько быстро, что за ним трудно уследить и трудно остановить именно в желательном месте. Для регулирования темпа перемещения и используется параметр Fr. Это аргумент процедуры Delay(Fr). По идее, процедура Delay на новых компьютерах не подстраивается под их фактическую тактовую частоту, поэтому *Fr* приходится задавать во много-много раз больше, чем требуется (задержка в миллисекундах).

7.2.1 Процедура `Mark` (разметка шкал)

Имеет меню

```
'0-exit,1-Rs,2-Sd,3-Xn,4-Xk,5-Hx,6-Yn,7-Yk,8-Hy,9-Mark'
```

Расшифровываем опции меню:

`'0-exit'` – это явно выход из меню.

`'1-Rs'` – размер штриха (от оси в одну сторону) – в пикселях.

`'2-Sd'` – символов после десятичной точки. Задавая `Sd` мы тем самым определяем форму представления чисел в информационной строке, обслуживающей выше приведенное меню.

`'3-Xn'` – начальное значение `X` (значения абсциссы крайнего левого штриха. Обычно в качестве `Xn` задается `Xmin`, но оно может нуждаться в уточнении (округлении, приближении к удобному для восприятия крайнего левого элемента оцифровки) которое, возможно, будет выполняться после разметки данной шкалы.

`'4-Xk'` – конечное значение – то же для крайнего правого штриха (с возможностью модифицирования как и `Xn` – ведь необходимо еще учесть удобство выбора шага *Hx* штрихов (см. следующий пункт)).

`'5-Hx'` – шаг штрихов вдоль оси `X` (в единицах `X`).

`'6-Yn'`, `'7-Yk'`, `'8-Hy'` – это аналоги `Xn`, `Xk`, `Hx`, но предназначенные для вертикальной шкалы.

`'9-Mark'` – эта опция выполняет разметку шкал, все предыдущие только готовили материал для этого завершающего этапа.

Отметим, что разметка выполняется инверсным цветом, так что повторный вызов опции `'9-Mark'` приводит к удалению только что нанесенной разметки.

7.2.2. Процедура `Net` (нанесение координатной сетки)

Имеет меню

```
'0-exit,1-Sd,2-Xn,3-Xk,4-Hx,5-Yn,6-Yk,7-Hy,7-Net'
```

В этом меню по сравнению с таковым для процедуры Mark отсутствует только опция '1-Rs' – размер штриха, т.к. штрихов тут нет. Все остальные опции полностью повторяют набор параметров подпрограммы Mark. Оно и понятно – процедура Net там, где должны были бы быть штрихи, формирует линии на всю ширину или, соответственно, высоту сетки. Краями сетки как раз и служат крайние (слева и справа, а также снизу и сверху) прямые, проведенные в местах, где могли бы быть штрихи.

7.2.3. Процедура Num (оцифровка шкал)

Чтобы график (годограф) воспринимался количественно, его шкалы должны быть оцифрованы. Элементы оцифровки (числа) должны располагаться напротив соответствующих им штрихов (если шкалы размечены, например, с помощью процедуры Mark) или же напротив линий координатной сетки (при формировании этой сетки, например, процедурой Net). Из этого следует, что процедура Num должна вызываться после срабатывания Mark или Net. В принципе это требование можно и нарушить, оставив на потом маркировку шкал или формирование координатной сетки, но каков в этом смысл ?

Основное меню процедуры Num выглядит так

'0-exit,1-Se,2-Sd,3-NumH,4-NumV'

Как уже отмечалось, оцифровка состоит из отдельных чисел, поставленных около (левее, правее, выше или ниже) соответствующих штрихов или линий.

Отдельное число (элемент оцифровки) характеризуется двумя параметрами (они уже упоминались среди типизированных констант модуля) Se и Sd.

Se - количество символов в элементе оцифровки,

Sd - количество символов после десятичной точки в этом элементе.

Задание определенных значений этим параметрам необходимо во-первых, чтобы определить формат представления других параметров оцифровки (см.

ниже) в соответствующей информационной строке (формируемой процедурой Info), и, во-вторых, задать формат самих элементов оцифровки.

Опция

'3-NumH' – организует нанесение оцифровки на горизонтальную ось,

'4-NumV' – то же для вертикальной оси.

В свою очередь, опция '3-NumH' имеет свое меню (подменю)

'0-exit,1-Ysc,2-Xn,3-Xk,4-Hx,5-Zmv,6-NnmH,7-DelNnmH'

'1-Ysc' – это Y - шкалы, т.е. координата Y горизонтальной линии, выбранной для привязки к ней оцифровки. Эта линия выполняет роль шкалы (отсюда и обозначение опции У для шкалы).

'2-Xn', '3-Xk', '4-Hx' – это соответственно, Xn – начальное значение X, Xk – конечное значение и Hx – шаг элементов оцифровки на формируемой шкале.

Обращаем внимание пользователя на связь этих параметров с одноименными параметрами, используемыми при маркировке шкал и формировании координатных сеток. Это глобальные параметры (описанные в модуле Serv) и если процедуры Mark и (или) Net вызывались до Num, то там были уточнены их значения и они определили размещение штрихов или линий сетки. Так что в первом приближении можно попытаться довольствоваться уже имеющимися значениями (унаследованными от предшественников) Xn, Xk и Hx. Но не обязательно. Может, штрихи (линии) слишком густо расположены, чтобы оцифровывать каждую из них.

Тогда, очевидно, стоит увеличить в целое число раз Hx, чтобы наносить оцифровку через несколько штрихов (линий). В принципе не обязательно оцифровывать крайние штрихи (линии). Тогда корректируется Xn и Xk так, чтобы они были кратны Hx, чтобы под оцифровку подпадала точка (линия), соответствующая нулевому значению X. Хотя как раз в подпрограмме Num данный элемент оцифровки оцифровываться не будет, иначе точка пересечения шкал подлежит оцифровке дважды (по одной шкале и по другой).

Элементы оцифровки разных шкал для точки их пересечения могут “конфликтовать” (накладываться или дублироваться – как для начала координат – друг на друга). Во избежание подобных осложнений, было решено пропускать элемент оцифровки, соответствующий точке пересечения шкал. Недостающий элемент (или элементы) могут быть сформированы в процедуре *Inscr* как соответствующая надпись (с помещением ее в место, выбранное пользователем).

’5-Zmv’ – смещение элементов оцифровки относительно оцифрованной линии. Смещение задается в пикселях. Это будет смещение вверх, если Zmv>0 или вниз, если Zmv<0.

’6-NumH’ – завершающий (активный) этап оцифровки – выполнение (формирование) заказанной оцифровки.

’7-DelNumH’ – стирание только что нанесенной оцифровки. Может оказаться, что выполненная только что оцифровка чем-то не устраивает пользователя (не те пределы *Xn* и (или) *Xk*, не тот шаг, не то смещение), тогда DelNumH убирает оцифровку, восстанавливая фон под ней (перед нанесением оцифровки фон под ней предусмотрительно запоминается).

Опция ’4-NumV’ подпрограммы Num оцифровывает вертикальную шкалу. Ее меню

’0-exit,1-Xsc,2-Yn,3-Yk,4-Hy,5-Zmh,6-NumV,7-DelNumV’

Это меню умышленно (для унификации) построено полностью аналогично таковому для опции ’3-NumH’, которая была рассмотрена только что. Но, как говорится, “с точностью до наоборот”.

’1-Xsc’ – X для шкалы,

’2-Yn’, ’3-Yk’, ’4-Hy’ – Y начальное, Y конечное и шаг по Y для элементов оцифровки линии, у которой Y равняется Ysc.

’5-Zmh’ – смещение по горизонтали в пикселях. Понятно, что элементы оцифровки вертикальной шкалы имеет смысл смещать только по горизонтали. Если Zmh>0 – смещение происходит вправо, при Zmh<0 – влево от шкалы.

'7-DelNumV' – стирание только что нанесенной оцифровки. Как и в аналогичной опции DelNumH, фон под наносимой оцифровкой предусмотрительно запоминается, так что при необходимости (когда вызывается опция DelNumV) этот запомненный фон восстанавливается. Тем самым создается возможность экспериментировать с оцифровкой, добиваясь оптимального (с точки зрения пользователя) решения.

7.2.4. Процедура Inscr (нанесение надписей)

После того, как график (годограф) сформирован, выполнена маркировка шкал (Mark) или нанесена координатная сетка (Net), проведена оцифровка шкал (Num) может возникнуть необходимость "нанести последние штрихи" (обозначить шкалы, пронумеровать или как-то прокомментировать отдельные кривые, подписать рисунок или его отдельные фрагменты и т.д.). Для решения подобных задач и предназначена процедура Inscr. Она имеет меню

'0-exit,1-Text,2-Sh,3-Sv,4-Fr,5-Move,6-DelText'

Опция '1-Text' –формирует текст (надпись), которую пользователь собирается поместить в каком-то месте экрана. Этот текст запоминается как изображение в пределах прямоугольника, охватывающего этот текст (с запасом в половину размера отдельного символа влево, вправо, вверх и вниз – поля вокруг текста). Затем запомненное изображение помещается на "стартовую позицию" в левой верхней части экрана (фон под стартовой позицией также запоминается, чтобы иметь возможность восстановить его, когда изображение начнет смещаться).

'2-Sh', '3-Sv' – смещение по горизонтали (Sh) и по вертикали (Sv). Эффект движения элемента изображения на экране телевизора (и компьютера) достигается путем последовательной демонстрации кадров, на каждом из которых перемещаемый объект смещается относительно его положения на предшествующем кадре. Так вот, Sh – смещение (в пикселях) по горизонтали и, соответственно, Sv – по вертикали очередного кадра относительно

положения предшествующего. Кадр у нас, собственно, один – это изображение перемещаемого текста, и этот кадр последовательно демонстрируется через определенный (величиной Fr) интервал времени, причем каждый раз место демонстрации смещается на Sh пикселей по горизонтали и Sv пикселей по вертикали. Чтобы не повредить фон при таком перемещении изображения текста по уже сформированной “картинке” перед каждой демонстрацией текста запоминается фон под тем местом, которое текст займет при данной демонстрации, после выдержки демонстрируемого текста на данном месте, он замещается фоном (сам текст при этом исчезает с экрана), запоминается новая область фона, демонстрируется текст на этом новом месте, выдерживается и т.д.

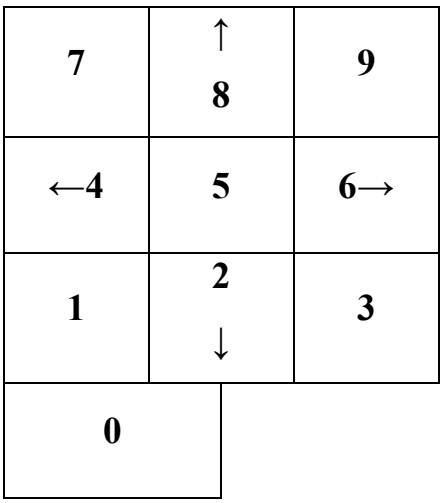
Величины Sh и Sv выбираются из следующих соображений. Чем меньше модуль Sh или Sv, тем медленнее (с меньшим шагом) будет перемещаться изображение текста по экрану и его будет возможно точнее установить в требуемом месте экрана. Sh>0 – дает перемещение вправо, Sh<0 – влево, Sv>0 - вверх, Sv<0 – вниз.

’4-Fr’ – как уже отмечалось в начале параграфа – это своего рода “коэффициент торможения”. Чем больше Fr, тем медленнее перемещается изображение по экрану (больше задержка при смене кадров). Итак, Fr – это параметр процедуры Delay(Fr), Fr подбирается под конкретный компьютер (ну, и с учетом темперамента пользователя).

’5- Move’ – организует движение в требуемом направлении. Эта опция имеет свое подменю

’0-exit, 2468-Move, 5-Stop’

Здесь ’2468-Move’ следует понимать так. Рекомендуется пользоваться цифровой клавиатурой, клавиши которой, как известно, расположены по следующей схеме



Стрелочки при цифрах 2, 4, 6, 8 как раз и указывают горизонтальную (4,6) и вертикальную (2,8) компоненты перемещения.

Модуль горизонтальной компоненты задается параметром `Sh` (в пикселях на кадр), вертикальной – `Sv` (пикселей на кадр). Остановка движения, как это следует из опции `'5-Stop'` - нажатием клавиши `'5'` (в центре цифровой клавиатуры).

Остановленный клавишей `'5'` текст может либо остаться на этом месте насовсем или же можно возобновить процесс перемещения одной (или двумя) из перечня `(2, 4, 6, 8)` клавишами до очередной остановки. Если текст “не ко двору” (или пользователь передумал или решил заменить данный текст каким-то другим), то этот (текущий) текст можно ликвидировать с помощью опции.

```
'6-DelText',
```

которая убирает (и забывает) этот текст, восстанавливая фон под его последним местом демонстрации. Т.е., удаление текста до его окончательной фиксации не создает никаких дополнительных проблем. А фиксация текста (на месте его последней “стоянки”) осуществляется опцией `'0-exit'`, т.е. в момент выхода из рассматриваемого подменю.

7.2.5. Процедура Service (сервисное обслуживание изображения)

Эта процедура настолько компактна, что ее не грех привести и полностью.

```
procedure Service;
  Var J1:char;
begin
  repeat
    putA;
    Ou('0-exit,1-Mark,2-Net,3-Num,4-Inscr');
    J1:=readkey;
    case J1 of
      '1':Mark;
      '2':Net;
      '3':Num;
      '4':Inscr
    end
  until J1='0'
end;
```

Структура ее настолько очевидна, что в дополнительном комментарии не нуждается.

7.3. Формирование годографов (файл Godogr)

Листинг файла Godogr приведен в приложении. Но подключить этот файл к программе следует не раньше процедуры UrGod, которая формируется под конкретную задачу. Имя процедуры UrGod означает уравнение годографа, и в этой процедуре вычисляются (возвращаются) значения двух переменных (глобальных, описанных в Serv) X,Y:real;

Здесь X - значение горизонтальной, Y - соответственно вертикальной координат очередной точки годографа. Применительно к задаче формирования годографов амплитудно-фазовых характеристик X, Y – это значения вещественно-частотной и мнимо-частотной характеристик для текущего (w:real) значения частоты (w-глобальный параметр).

Для примера рассмотрим вариант расчета точки амплитудно-фазовой

характеристики, соответствующей дробно-рациональной передаточной функции с запаздыванием τ

$$W(p)=\frac{B(p)}{A(p)}e^{-p\tau}.$$

Соответствующая амплитудно-фазовая характеристика:

$$W(j\omega)=\frac{B(j\omega)}{A(j\omega)}e^{-j\omega\tau}.$$

Воспользуемся услугами процедуры `HorComp`, чтобы вычислить значения полинома-числителя $B(p)$ при $p=0+j\omega$ (результат R_1+jI_1) и знаменателя $A(p)$ при $p=0+j\omega$ (результат R_2+jI_2) и поделим числитель на знаменатель:

$$\frac{R_1+jI_1}{R_2+jI_2}=\frac{(R_1+jI_1)\cdot(R_2-jI_2)}{(R_2+jI_2)\cdot(R_2-jI_2)}=R_3+jI_3,$$

где

$$R_3=\frac{R_1\cdot R_2+I_1\cdot I_2}{R_2^2+I_2^2},$$

$$I_3=\frac{R_2\cdot I_1-R_1\cdot I_2}{R_2^2+I_2^2}.$$

Обозначая заново $R_1=cos(\omega\tau)$ и $I_1=sin(\omega\tau)$ представляем:

$$e^{-j\omega\tau}=R_1-jI_1.$$

А теперь остается перемножить (R_3+jI_3) на (R_1-jI_1), чтобы получить окончательный результат:

$$W(j\omega)=(R_3+jI_3)\cdot(R_1-jI_1)=X+jY,$$

где

$$X=R_1\cdot R_3+I_3\cdot I_1,$$

$$Y=R_1\cdot I_3-I_1\cdot R_3.$$

Рассмотренный алгоритм реализуется в данном случай так:

```
procedure UrGod;
var R1,I1,R2,I2,R3,I3,Zn,Wtau:real;
begin
  HorComp(B,0,W,R1,I1);
  HorComp(A,0,W,R2,I2);
```

```
  Zn:=Sqr(R2)+Sqr(I2);
  R3:=(R1*R2+I1*I2)/Zn;
  I3:=(R2*I1-R1*I2)/Zn;
  Wtau:=W*Tau;
  R1:=Cos(Wtau); I1:=Sin(Wtau);
  X:=R3*R1+I3*I1;
  Y:=R1*I3-I1*R3
end;
```

Естественно, что в каждом случае начинка подпрограммы `UrGod` будет своей (под решаемую задачу или задачи).

Итак, если выше описана процедура `UrGod`, можно рассматривать описание процедуры `SZ` из файла `Godogr`. Приводим ее.

7.3.1. Процедура SZ

```
procedure SZ;
begin
  UrGod;
  if Dx<>0 then s:=x0+round(x/Dx)
           else s:=x0;
  if Dy<>0 then z:=y0-round(y/Dy)
           else z:=y0
end;
```

Процедура `SZ` превращает ответ процедуры `UrGod` (глобальные `X, Y: real`) в глобальные `S,Z: integer`. Эти `S` и `Z` – это экранные координаты точки годографа, соответствующие текущему `W` (идентификатор для ω), для которого процедура `UrGod` (вызванная для этого в `SZ`) вычислила `X` и `Y`. Совершенно очевидно, что к услугам процедуры `SZ` можно прибегать только тогда, когда в наличии имеется уже сформированная система координат (для которой определены `X0, Y0, Dx, Dy` – положение начала координат и цены шкал).

А вот чтобы подходящая система координат таки была сформирована (если в этом направлении кто-то не подсуетился заранее) вызывают подпрограмму `Scan`.

7.3.2. Процедура Scan

```
procedure Scan;
var s:integer;
begin
  Dw:=(Wk-Wn)/Nsc;
  Xmin:=0; Xmax:=0; Ymin:=0; Ymax:=0;
  for s:=0 to Nsc do
    begin
      W:=Wn+S*Dw;  UrGod;
      if X<Xmin then Xmin:=X;
      if X>Xmax then Xmax:=X;
      if Y<Ymin then Ymin:=Y;
      if Y>Ymax then Ymax:=Y
    end;
  X0Y0(true);ClearDevice; SystCoor
end;
```

Подпрограмма `Scan` выполняет сканирование диапазона `Wn..Wk` изменения параметра `W`. Диапазон разбивается на Nsc шагов, соответственно шаг сканирования `Dw` оказывается разным

```
Dw:=(Wk-Wn)/Nsc;
```

В процессе сканирования (в цикле по *s*) на каждом шаге определяется текущее значение параметра `W` (`W:=Wn+S*Dw`), для этого `W` процедура `UrGod` определяет текущие значения `X` и `Y`, для которых по ходу сканирования определяют их крайние значения (`Xmin`, `Xmax`, `Ymin`, `Ymax`). Перед началом сканирования эти крайние значения принимаются равными нулю. Это делается затем, чтобы гарантировать присутствие на экране начала координат формируемой на их базе системы координат.

После выхода из цикла, когда `Xmin`, `Xmax`, `Ymin`, `Ymax` приобретают свои окончательные (истинные) значения, они передаются процедуре `X0Y0(true)` для определения параметров системы координат. Параметру `God` в `X0Y0` присваивается значения true, когда система координат формируется под годограф (шкалы необходимо согласовывать между собой). Затем экран очищается, и на чистом экране формируются оси системы

координат.

7.3.3. Процедура PointGod

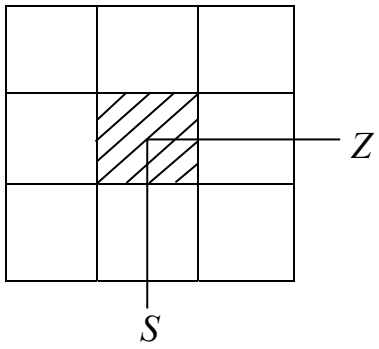
```
procedure PointGod;
const Kw=1.2;
var Mdx,Mdy,Md,Sp,Zp:integer;
    Wp:real;
begin
  Sp:=s;Zp:=z;Wp:=w;
  repeat
    W:=Wp+Dw; SZ; Mdx:=abs(S-Sp);Mdy:=abs(Z-Zp);
    if Mdx>Mdy then Md:=Mdx
               else Md:=Mdy;
    if Md<>1 then
      if Md<Min then Dw:=Dw*Kw
                else Dw:=Dw/Md;
  until (Md=1) or (W>Wk);
  if Md=1 then
  if c>=0 then PutPixel (S,Z,c)
          else PutPixel (S,Z,GetMaxColor-GetPixel(s,z))
end;
```

Процедура `PointGod` предназначена для формирования (закрашивания) очередного пикселя в изображении годографа. В данной процедуре заложен алгоритм, который, во-первых, гарантирует непрерывность линии, изображающей годограф (каждый очередной пиксель касается предыдущего), а во-вторых, каждый пиксель годографа закрашивается один и только один раз. Последнее обстоятельство открывает возможность формирования годографа в инверсном цвете.

Это может оказаться удобным, если годограф наносится поверх ранее сформированного изображения (которое жалко было бы потерять или испортить). Если после нанесения годографа (инверсным цветом) возникает желание его убрать или переделать, то повторное рисование в инверсном цвете помогает решить задачу. Читатель, очевидно, помнит, что в стандартном Турбо Паскале в инверсном цвете(вызывая процедуру `SetWriteMode(1)`) можно формировать только фигуры, состоящие из отрезков прямых линий. На

формирование кривых (а годограф – это, как правило, кривая) действие процедуры `SetWriteMode(1)` не распространяется. А вот в процедуре `PointGod` такая возможность предусмотрена. И как же она реализуется? Предположим, что, по крайней мере, один пиксель годографа на экране определен (изображен). Первый пиксель соответствует `W=Wn`. А система координат предполагается уже сформированной (ну, например, усилиями выше рассмотренной процедуры `Scan`), так что этот первый пиксель вполне можно определить : `W=Wn; UrGod; SZ`.

В дальнейшем это будет очередной (последний по времени) сформированный поточный пиксель годографа. На приводимой ниже схеме – его изображает заштрихованный прямоугольник с координатами `S,Z`.



Чтобы гарантировать непрерывность годографа в окрестности рассматриваемого пикселя, очередной пиксель должен соприкасаться с данным, т.е. совпадать с одним из его 8 ближайших “соседей”, показанных на схеме. Запомним координаты `S, Z` как `Sp, Zp (Sp:=S , Zp:=Z)`, а значение `W` как `Wp (Wp:=W)`.

Дадим некоторое приращение `Dw` параметру `W`

```
W:=Wp+Dw,
```

и для нового значения W вычислим экранные координаты соответствующего ему пикселя для формируемого годографа. Для этого достаточно вызвать процедуру `SZ`, которая определит новые значения S и Z. Оцениваем

```
Mdx:=abs(S-Sp); Mdy:=abs(Z-Zp).
```

Это расстояния в пикселях по горизонтали и по вертикали между пикселями с координатами Sp, Zp и S, Z. Выбираем Md – большее их них.

```
If Mdx>Mdy then Md:=Mdx
          else Md:=Mdy.
```

Если окажется, что `Md=1`, то это как раз то, что нам требуется. При выполнении этого условия очередной пиксель будет удовлетворять условию непрерывности (это будет один из ближайших, находящихся в непосредственном контакте с `Sp, Zp,` пикселей). Но от этой идиллии возможны и отклонения.

`S` и `Z` в процедуре `SZ` получаются с использованием опции `round`, а эта операция может округлить до нуля даже разные числа. Так что, если приращение Dw окажется недостаточно большим, то мы вполне можем “округлиться” в предыдущий пиксель (`Sp:=S, Zp:=Z`), следовательно: `Mdx=0; Mdy=0, Md=0`. Что делать? Да увеличивать `Dw`. Что тут еще можно придумать?

```
If Md<1 then Dw:=Dw*Kw
        else Dw:=Dw/Md;
```

Итак, предыдущий вариант Dw предлагается умножить на коэффициент `Kw` (const `Kw = 1.2`). Почему именно на 1.2 ? Что это за “магическое число” ? Дело тут вот в чем. Годографы, с которыми приходится иметь дело при решении задач автоматизации производственных процессов, характерны тем, что метки параметра `w` (частот) на них могут распределяться чрезвычайно неравномерно. Так, например, может оказаться, что на начальном участке годографа значение `Dw`, соответствующее переходу на очередной пиксель, будет равно порядка 0,001, а на конечном участке, чтобы перейти на очередной пиксель `Dw`, должно будет исчисляться сотнями, тысячами – вплоть до ∞. Не исключено даже, что последний пиксель годографа окажется зарисованным уже при каком-то конечном значении w, хотя теоретически этот последний пиксель должен соответствовать w=∞. Причина – в округлении, упоминавшемся выше.

В этом случае попытка выхода из этого (фактически последнего) пикселя

заведомо обречена на неудачу. Это мы отмечаем затем, чтобы читатель мог оценить проблему выбора величины Kw. Если выбрать Kw побольше, то вероятность выбраться из Sp, Wp за одну попытку возрастает (и это хорошо!), но зато возрастает и вероятность “выпрыгнуть” из этого пикселя дальше, чем хотелось бы. А хотелось бы, чтобы Md=1. Но не больше, т.к. тогда придется повторять попытку, уменьшая “прыть”, то бишь Dw. А уменьшение, как видно из последнего приведенного оператора, осуществляется в Md раз. Если взять Kw достаточно большое (больше 1.2), то может так статься, что для некоторого годографа на некотором его участке увеличение Dw в Kw раз окажется таким, что Md станет намного больше 1, а после деления Dw на это Md, мы получим Md=0, т.е. вернемся туда, откуда пытались “выпрыгнуть”. Это может привести к зацикливанию, что совершенно нежелательно. Чем ближе Kw к *1* (но все же больше единицы!), тем вероятность зацикливания уменьшается, но зато снижается быстродействие алгоритма, т.к. выход из точки Sp, Zp может происходить только после многократно повторяющихся попыток. Мы остановились на Kw = 1.2 путем подбора. Для тех годографов, с которыми нам приходилось иметь дело до сих пор, это число оказалось приемлемым. А вообще-то это настроечный параметр.

Выход из цикла repeat очевиден: или же будет найдено такое Dw, что Md=1 (искомая точка – пиксель найдена!) или же W>Wk (заданный диапазон параметра W исчерпан).

В первом случае обретенный пиксель необходимо зафиксировать (процедура PutPixel), иначе к уже сформированному годографу ничего добавлять не требуется.

Если c>=0, то очередной пиксель зарисовывается заданным цветом, при невыполнении же этого условия – рисуем его инверсным цветом.

7.3.4. Процедура LineGod

```
procedure LineGod;
  var Mdx,Mdy,Md,Sp,Zp:integer;
      Wp,R:real;
begin
  Sp:=s;Zp:=z;Wp:=w; R:=(Min+Max)/2;
  repeat
    W:=Wp+Dw; SZ;
    Mdx:=abs(S-Sp);Mdy:=abs(Z-Zp);
    if Mdx>Mdy then Md:=Mdx
               else Md:=Mdy;
    if (Md>Max) or (Md<Min) then
    if Md<Min then Dw:=Dw*Kw
              else Dw:=Dw/Md*r;
  until ((Md<=Max) and (Md>=Min)) or (W>Wk);
  if (Md<=Max) and (Md>=Min) then
    Begin
      Line(Sp,Zp,S,Z);
      If c<=0 then
        PutPixel(S,Z,GetMaxColor-GetPixel(S,Z))
    end
end;
```

Процедура LineGod формирует очередной отрезок ломаной, которая представляет изображаемый годограф. Необходимость прибегать к услугам LineGod возникает тогда, когда трудоемкость вычисления координат очередной точки годографа слишком велика для компьютера. Так, например, на относительно медленных компьютерах рисование некоторых годографов пиксель за пикселем (процедура PointGod) занимало время порядка часа (на один годограф!). И это не предел. Это утомительно (скучно!). Процесс можно ускорить, если не настаивать на предельно высоком качестве формируемого годографа (с точностью до пикселя!), а согласиться на более грубое (но зато и более быстрое!) решение, а именно, формировать годограф в виде ломаной, размеры отрезков в которой не должны выходить за диапазон Min..Max. Размер отрезка оцениваем величиной Md (большей из его горизонтальной и

вертикальной проекций Mdx и Mdy). Величины Min и Max задаются как типизированные константы (при необходимости их можно корректировать).

Структура процедуры LineGod в основном аналогична такой для PointGot. Здесь только дополнительно введен коэффициент

R:=(Min+Max)/2,

задающий среднюю длину отрезков ломаной. Корректировка Dw производится при условии, что длина очередного (предлагаемого) отрезка ломаной не укладывается в "интервал" между Min..Max. Если отрезок короток (Md <Min), то его "вытягиваем" (Dw:=Dw*Kw), если же он слишком длинен – то, как и положено, укорачиваем (Dw:=Dw*R/Md).

И вот только когда (Md<=Max) и (Md>=Min), тогда отрезок считается "в самый раз " – выходим из цикла и рисуем этот отрезок. Из цикла выходим и тогда, когда (W>Wk), т.е. когда исчерпан заданный диапазон изменения параметра *W*. На тот случай, когда c<0 (инверсный цвет) точка с координатами (S,Z) зарисовывается еще раз (второй) инверсным цветом, т.к. после этого она будет зарисовываться еще один раз (третий) как начало очередного отрезка ломаной.

7.3.5. Процедура Godo

```
procedure Godo;
var J3:char; m:integer;
begin
  PutA;Ou('0-Exit'); J3:='1';
  Dw:=(Wk-Wn)/Nsc; W:=Wn; Sz;
  If c>=0 then PutPixel(S,Z,C)
          else PutPixel(S,Z,GetMaxColor-GetPixel(s,z));
  If c>=0 then SetColor(C)
          else SetWriteMode(1);
  repeat
    case Tg of
      P: PointGod;
      L: LineGod
    end;
    Str(W,T20); Clear(450,0,GetMaxX-1,10);
```

```
    Outtextxy(450,0,'W='+T20);
    if KeyPressed then J3:=ReadKey
  until (W>Wk) or (J3='0');
  if c>=0 then SetColor(15)
          else SetWriteMode(0)
end;
```

Краткий комментарий.

```
PutA; Ou('0-exit'); J3:='1';
```

Как уже выше отмечалось, процесс построения годографа может затягиваться. Возможных причин для этого может быть несколько: неудачно задан диапазон (`Wn..Wk`), выбрано значение `Tg` равным `'P'` вместо `'L'`, или же `Min, Max` заданы слишком малыми и т.п. Более того может оказаться, что в процессе формирования годографа (если он затягивается) выявляется нецелесообразность дальнейшего его продолжения. Ну, например, требуется оценить относительное расположение годографа и так называемой М-окружности (или какой-то другой кривой). Так вот по мере формирования годографа может наступить момент, когда ответ на интересующий нас вопрос уже получен, так что дальнейшая прорисовка годографа теряет свою актуальность. На этот случай и предусматривается возможность досрочного завершения процесса. На такую возможность указывает надпись

```
Ou('0-exit').
```

Смысл надписи очевиден – хочешь выйти – нажми на клавишу `0`. Выход из цикла repeat по указанной причине производится когда переменная `J3:char` в результате нажатия клавиши `'0'` получит значение `'0'` `(J3:=ReadKey)`. На всякий случай перед началом процесса `J3:='1'` (чтобы не ноль!)

```
Dw:=(Wk-Wn)/Nsc;
```

Шаг Dw приращения параметра `W` для начала выбираем равным в `1/Nsc` ширины заданного диапазона. Если этого не сделать, он может оказаться либо

равным 0 (и тогда его долго придется умножать на Kw, чтобы превратить во что-то подходящее. Верно, долго?). Не намного лучше, если `Dw` – большое (а таким оно могло стать в конце формирования какого-то предыдущего годографа), тогда его придется уменьшить до требуемого размера. И приведенный вариант `Dw:=(Wk-Wn)/Nsc` тоже не идеален, но как правило, лучше так, чем полагаться на волю чистого случая.

```
W:=Wn; SZ;
```

Эти два оператора задают начальное значение параметра *W* и определяют заданные координаты (`S,Z`) первого пикселя годографа.

```
If c>=0
  then PutPixel(s,z,c)
  else PutPixel(S,Z,GetMaxColor-GetPixel(s,z ));
```

Это рисуется тот самый первый пиксель явным (c) или инверсным (если *c<0*) цветом. На тот случай, если годограф планируется рисовать в виде ломаной (`Tg='L'`) задается цвет годографа – явный через *SetColor*(*c*) или инверсный через `SetWriteMode(1)`. Оператор

```
case Tg of
  P: PointGod;
  L: LineGod
end;
```

рисует очередной пиксель или отрезок ломаной.

```
Str(W,T20); Clear(450,0,GetMaxX-1,10);
OutTextXY (450,0,' W='+T20);
```

Эти операторы организуют вывод на экран текущего значения `W`. Сопоставляя его с `Wn` и `Wk` пользователь может в каждый момент времени оценить “на каком свете он находится”, т.е. сколько уже сделано, сколько осталось- и стоит ли продолжать?

```
If KeyPressed then J3:=ReadKey;
```

Этот оператор следит за возможными “ценными указаниями” от пользователя. Если тот что-то нажимает, то код нажатой клавиши запоминается в `J3`. И если будет нажата клавиша `0` (как было договорено выше `Ou('0-exit')`), то, значит, пора выходить из цикла `repeat`. Другая причина выхода, это когда (`W>Wk`).

После завершения формирования годографа, установки по цвету восстанавливаются до значений, имеющих место по умолчанию,

```
if c>=0 then SetColor(15)
        else SetWriteMode(0).
```

На этом процедура `Godo` завершает свою работу.

7.4. Формирование изображения поверхности системой изолиний

Предметом анализа могут быть не только графики функции или параметрические зависимости в виде годографов, но и поверхность в случае функции двух независимых переменных. Так при поиске комплексных корней полинома путем сканирования заданного участка комплексной плоскости изучалась поверхность модуля значения полинома. При изучении незатухающих колебаний маятника предметом анализа может служить поверхность значений его полной энергии (суммы кинетической и потенциальной) в функции от угла отклонения маятника и его угловой скорости. Список примеров можно неограниченно продолжать. Одним из наиболее простых способов отобразить на плоскости особенности поверхности функции двух переменных может быть такой. Пересекаем изучаемую поверхность серией плоскостей (с шагом Hiz) перпендикулярных оси Z, вдоль которой откладывается значение функции *f*. Линии пересечения поверхности каждой из упомянутых плоскостей проектируются на плоскость XOY. Это так называемые изолинии, т.к. каждой точке отдельно взятой изолинии соответствует одно и тоже значение функции *f* (при котором соответствующая

секущая плоскость пересекает ось Z), откуда и термин «изолиния». Вообще изолинии – это линии одинаковых значений функций. Система изолиний, если ее к тому же снабдить метками уровней (значений функции для каждой из них) позволяет составить представление об особенностях поведения изучаемой функции в заданной области изменения ее аргументов. Будем задавать область аргументов в виде прямоугольника ограниченного Xmin, Xmax, Ymin, Ymax, которая отображается на экране компьютера как показано на схеме рис.7.1.

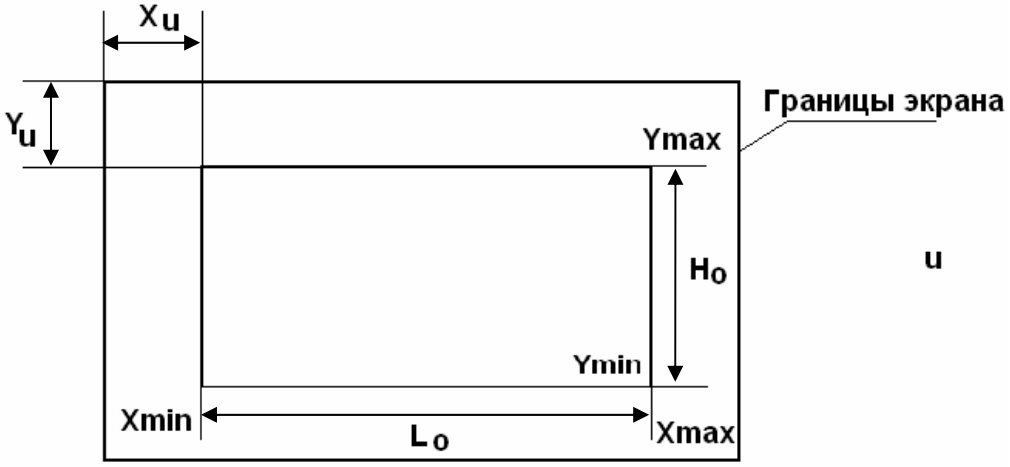


Рис.7.1. Схема отображаения границ изображения на экране монитора.

Расстояние в один пиксель по горизонтали (вдоль оси X) соответствует шагу `Dx`

```
Dx:=(Xmax-Xmin)/L0,
```

а по вертикали, соответственно `Dy`

```
Dy:=(Ymax-Ymin)/H0.
```

Будем сканировать прямоугольник `L0*H0`, показанный на схеме, пиксель за пикселем, например, вертикаль за вертикалью слева направо. В каждом пикселе (которому соответствует вполне определенное значение X и Y) вычисляется значение функции *f*(*x*,*y*). Предполагается, что у нас имеется подпрорамма-функция `Fxy`, которая вычислит функцию при переданных ей значениях аргументов. Полученное значение фукции делится на `Hiz` – получается `Nrr` (номер уровня в формате `real`). Целая часть `Nrr` заносится в

массив `Mnr:array{-1..500} of integer`. Таким образом, `Mnr[pz]:=trunc(Fxy(x,y)/Hiz)` в массиве `Mnr` формируется как бы вертикальный срез функции при некотором фиксированном *x*. По мере возрастания номера ячейки в массиве `Mnr` увеличивается значение *y* (с шагом `Dy`). Пока значение элементов массива Mnr не меняется, то это значит, что значение функции находится между двумя соседними секущими плоскостями, а вот факт изменения содержимого ячейки массива `Mnr` при переходе от одного к другому пикселю на рассматриваемой вертикали свидетельствует, что значение функции переходит из промежутка между двумя соседними секущими плоскостями к следующему (снизу или сверху) промежутку, а это значит, что где-то между двумя этими пикселями значение функции равняется индексу какой-то изолинии. Поскольку изображение на экране формируется с точностью до пикселя, то один из соседних пикселей, у которых содержимое соответствующих ячеек массива не совпало, можно считать принадлежащим одной из возможных изолиний. Чтобы не усложнять без нужды алгоритм, будем считать, что это текущий пиксель (при прохождении через который обнаружено изменение целой части `Nrr`). Описанный алгоритм позволяет выявить точки изолиний, направление которых в окрестностях рассматриваемого пикселя “скорее горизонтально, чем вертикально”. А если наоборот? Тогда желательно сопоставлять текущий пиксель и его ближайшего соседа слева. Если и (или) в этих двух пикселях обнаруживается несовпадение целых частей `Nrr`, то – закрашиваем текущий пиксель. Чтобы иметь информацию о целой части `Nrr` в пикселе слева от текущего (анализируемого) в соответствующей z-ой ячейке массива `Mnr` будем хранить соответствующее значение, полученное при сканировании предшествующей вертикали и обновлять ее (под текущую вертикаль) только после сравнения предыдущего значения с текущим (хранится временно в `Nr=trunc(Nrr)`). Итак, какие пиксели (принадлежащие изолиниям) нужно запрашивать – мы разобрались. А вот каким цветом? Да каким хотите. Можно предложить, например, такой

вариант. Изолинию №1 (f=Hiz) закрашивать цветом C=1, изолинию №2 – цветом 2 и т. д. Это хорошо если число изолиний не превышает 15. А если их больше? Определим цвет как

```
round (Abs(Nrr))mod 16
```

Цвета будут периодически повторяться, но в пределах данной цветовой шкалы это лучшее решение, которое можно предложить. При более богатых цветовых палитрах можно придумать более эффектные решения данной проблемы. Описанный алгоритм можно оформить в виде процедуры Izo.

```
procedure Izo (HIZ:real; God:boolean);
  const Mi:integer=5000;
  var Z,S,Nr:integer;
      Nrr:real;
      Mnr:array[-1..500] of integer;
begin
  X0Y0(God);  ClearDevice; {SystCoor}
  for Z:=-1 to H0 do
    begin
      Nrr:=Fxy(Xmin-Dx, Ymin+Z*Dy)/HIZ;
      if Abs(Nrr)>Mi then Mnr[z]:=Mi
                     else Mnr[z]:=trunc(Nrr)
    end;
  for S:=0 to L0 do
    begin
      X:=Xmin+S*Dx;
      for Z:=-1 to H0 do
        begin
          Nrr:=Fxy(X,Ymin+Dy)/HIZ;
          if Abs(Nrr)>Mi then Nrr:=Mi;
          Nr:=trunc(Nrr);
          if Z>-1 then
```

```
            if (Nr<>Mnr[z-1]) or (Nr<>Mnr[z])then
              if Abs(Nrr)<Mi then
                PutPixel(Xn+S,Yn+H0-Z);
          round(Abs(Nrr)) mod 16;
          Mnr[-z]:=Nr;
        end;
    end;
end.
```

Кратко прокомментируем процедуру. Начнем с параметров, передаваемых в заголовке. `Hiz:real` – шаг изолинии (расстояние между соседними секущими плоскостями, параллельными оси Z). `God:Boolean` – параметр, определяющий необходимо ли согласовывать шкалы по осям X и У. Если `God=true`, то необходимо. Выбор значения параметра God зависит от того совпадают ли по физической природе X и У. Например, при формировании изображения модуля значений полинома X и У – это вещественная и мнимая части аргумента полинома – величины одной размерности, логично задать `God=true`. А вот при изучении свободных колебаний маятника по оси X откладывается угол отклонения, а по оси У – угловая скорость – это величины разной размерности и `God=false`.

`Const Mi:integer=5000;` Это еще что? Дело вот в чем. Мы собираемся в массив `Mnr:array[-1..500] of integer` заносить целые части от `Nrr`. Выделение целой части осуществляется функцией `Trunc`. Кроме того, как уже упоминалось, придется пользоваться услугами функции `Round`: `round(Abs(Nrr))`

Компилятор отказывается выполнять операции `Round` и `Trunc` над аргументами, которые выходят за допустимый диапазон целых чисел `(-32768..32676)`. Поэтому в тех случаях, когда `Nrr:=Fxy(X, Y)/Hiz;` оказывается по модуля больше, чем `Mi`, то в соответствующую ячейку массива `Mnr` просто заносится число `Mi`. Число `Mi` выбирается достаточно большим,

чтобы при формировании системы изолиний потеря истинного значения Nrr не приводила бы к искажениям. Ну, судите сами. Размеры прямоугольника отводимого на экране под отображение заданной области плоскости ХОУ, исчисляются сотнями пикселей по каждой из осей. Следовательно, различных изолиний (формируемых с точностью до пикселя) на изображении может присутствовать (так, чтобы не накладываться друг на друга, т. е. не сливаться) в лучшем случае несколько сотен. При больших значениях `Nrr` вероятнее всего соседним пикселям будут, очевидно, соответствовать `trunc(Nrr)`, отличающиеся не на единицу, а на значительно большие числа. В такой ситуации закрашиваться будут практически все пиксели в данной области, а цвет закраски становится хаотичным. Картинка получается красивая, но рассмотреть на ней отдельные изолинии – невозможно. Рекомендуем читателю понаблюдать на мониторе изолинии модуля значений полинома при значительном удалении от его корней. В такой ситуации заботиться о сохранении истинного значения Nrr с целью формирования неискаженной картины изолиний – нет смысла.

В первом цикле по Z формируется массив `Mnr` для `X:=Xmin-Dx`. Это вертикаль, находящаяся на один пиксель левее левого края заданной области. При заполнении этого массива никаких решений о запросе тех или иных пикселей не принимается. Этот массив необходим для сравнения его с аналогом в процессе прохождения вдоль вертикали с `X=Xmin` (когда сравнивается значение `trunc(Nrr)` в текущем пикселе с его значением в пикселе, соседствующем с текущим слева).

Заполнение ячейки `Mnr[-1]` необходимо для решения той же задачи, а именно сравнения содержимого ячеек `Mnr[-1]` и `Mnr[0]`, что позволяет принять решение о закраске пикселя на уровне Ymin. Вопрос о закраске пикселя, соответствующего ячейке `Mnr[-1]` (при z=-1) не рассматривается, так как закрашивать этот пиксель не требуется (он ниже нижнего края заданной области).

7.5. Формирование аксонометрического изображения поверхности

Представление об особенностях рельефа поверхности функции двух переменных $Z=f(x,y)$ можно получить формируя на экране компьютера изображение системы сечений этой поверхности плоскостями, перпендикулярными осям X и (или) У. Для общности будем рассматривать вариант, когда пользователю программы предоставляется возможность по его усмотрению менять ориентацию исследуемой поверхности относительно наблюдателя.

Рассмотрим систему координат XOYZ, связанную с экраном компьютера так, как показано на рис. 7.2.

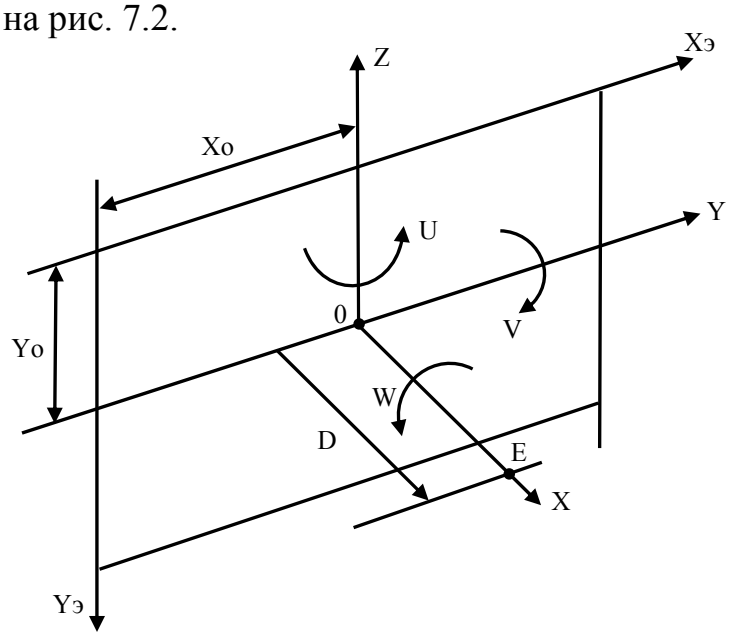


Рис. 7.2. Система координат и углы поворота преображаемого объекта.

Итак, начало системы XOYZ помещается в точку на экране с экранными координатами X0, Y0. Ось Z направлена вверх, ось Y – вправо, а ось X – перпендикулярна плоскости экрана и направлена к зрителю. К слову, сам зритель (его глаз) находится на оси X на расстоянии D от экрана (точка E). А теперь представим, что исследуемая поверхность (объект исследования) помещается таким образом, чтобы его “центр” совмещался с точкой 0. Под словом центр будем понимать точку, находящуюся посредине между крайними точками объекта в направлении осей X, У и Z соответственно. Привязка центра

к точке 0 оправдана тем соображением, что мы собираемся вращать объект вокруг осей Z (на угол U), У (на угол V) и X (на угол W). И если не выполнить упомянутой привязки, то в процессе вращения какие-то части изображения могут оказаться за пределами экрана, а уменьшать размеры изображения во избежание этого эффекта – нежелательно.

Договоримся указанные повороты тела совершать в такой последовательности (U → V → W). При каждом повороте каждая точка объекта будет изменять свое положение в пространстве. Будет также меняться ее проекция на экран (проецирование будет выполняться из точки E). Определим эти перемещения на примере точки A(x,y,z).

Начнем с разворота на угол U вокруг оси Z. Положительным, как это обычно принято, будем считать вращение в направлении против часовой стрелки, если смотреть из точки на положительной ветви соответствующей оси. Спроектируем точку A на плоскость XOY – рис.7.3.

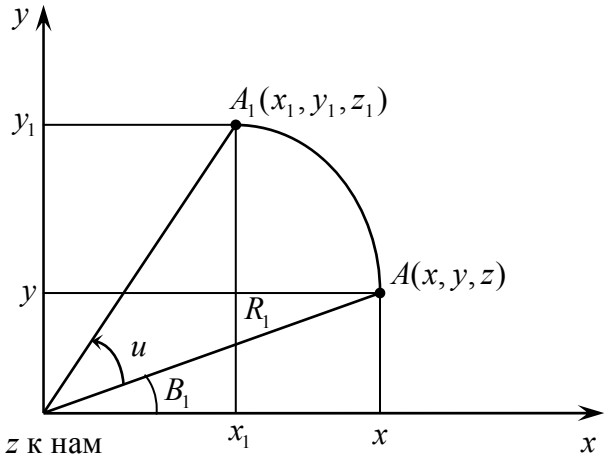


Рис.7.3. Поворот на угол *u* вокруг оси *z*.

В результате поворота точка $A(x,y,z)$ переместится в $A_1(x_1,y_1,z_1)$. Очевидно, что

$$\begin{cases} x_1 = R_1 \cos(B_1 + u), \\ y_1 = R_1 \sin(B_1 + u), \\ z_1 = z. \end{cases}$$

Учитывая известные формулы для синуса и косинуса суммы получаем

$$\begin{cases} x_1 = R_1(\cos B_1 \cos u - \sin B_1 \sin u), \\ y_1 = R_1(\sin B_1 \cos u + \cos B_1 \sin u), \\ z_1 = z. \end{cases}$$

Но

$$\begin{cases} R\cos B_1 = x, \\ R\sin B_1 = y, \end{cases}$$

следовательно,

$$\begin{cases} x_1 = x\cos u - y\sin u, \\ y_1 = y\cos u + x\sin u, \\ z_1 = z. \end{cases} \qquad (7.1)$$

Выполним теперь поворот на угол V вокруг оси y. Спроектируем точку $A_1(x_1, y_1, z_1)$ на плоскость XOZ – рис.7.4.

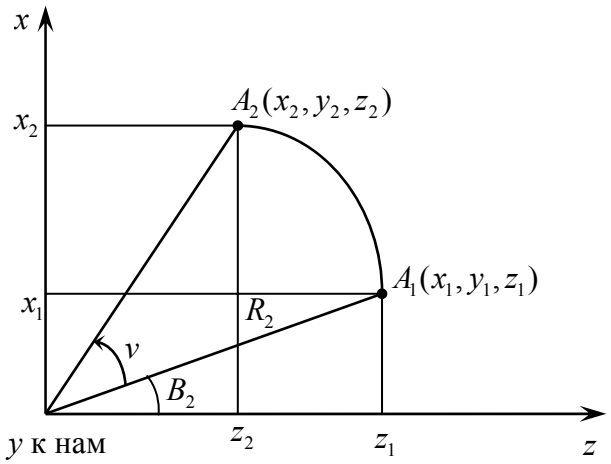


Рис.7.4. Поворот на угол V вокруг оси y.

Точка $A_1(x_1, y_1, z_1)$ переходит в положение $A_2(x_2, y_2, z_2)$.

$$\begin{cases} x_2 = R_2 \sin(B_2 + v), \\ y_2 = y_1, \\ z_2 = R_2 \cos(B_2 + v). \end{cases}$$

Или, что то же

$$\begin{cases} x_2 = R_2(\sin B_2 \cos v + \cos B_2 \sin v), \\ y_2 = y_1, \\ z_2 = R_2(\cos B_2 \cos v - \sin B_2 \sin v). \end{cases}$$

Но

$$\begin{cases} R_2 \cos B_2 = x_1, \\ R_2 \sin B_2 = y_1, \end{cases}$$

следовательно,

$$\begin{cases} x_2 = x_1 \cos v + z_1 \sin v, \\ y_2 = y_1, \\ z_2 = z_1 \cos v - x_1 \sin v. \end{cases}$$

Подставляем в последние выражения вместо x_1, y_1, z_1 их значения из системы (7.1)

$$\begin{cases} x_2 = (x \cos u - y \sin u) \cos v + z \sin v, \\ y_2 = y \cos u + x \sin u, \\ z_2 = z \cos v - (x \cos u - y \sin u) \sin v. \end{cases} \qquad (7.3)$$

И, наконец, последний поворот (W вокруг *x*). Проецируем изображение на плоскость *YOZ*.

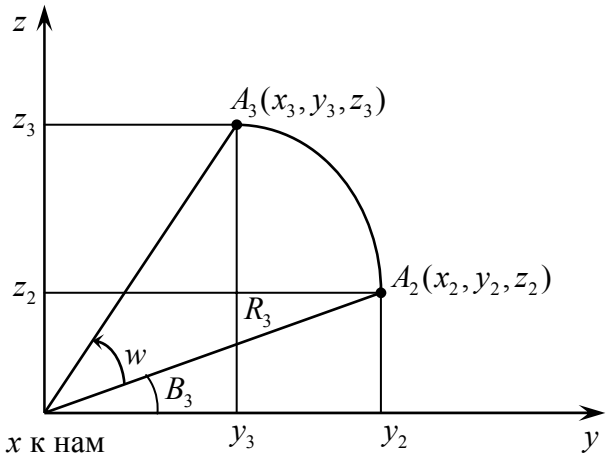


Рис.7.5. Поворот на угол W вокруг оси *x*.

Точка $A_2(x_2, y_2, z_2)$ переходит в точку $A_3(x_3, y_3, z_3)$.

$$\begin{cases} x_3 = x_2, \\ y_3 = R_3 \cos(B_3 + w), \\ z_3 = R_3 \sin(B_3 + w), \end{cases}$$

или

$$\begin{cases} x_3 = x_2, \\ y_3 = R_3(\cos B_3 \cos w - \sin B_3 \sin w), \\ z_3 = R_3(\sin B_3 \cos w + \cos B_3 \sin w). \end{cases}$$

Но

$$\begin{cases} R_3 \cos B_3 = z_2, \\ R_3 \sin B_3 = y_2. \end{cases}$$

Итак

$$\begin{cases} x_3 = x_2, \\ y_3 = y_2 \cos w - z_2 \sin w, \\ z_3 = z_z \cos w + y_2 \sin w. \end{cases}$$

Заменяем в последних выражениях значения x_2, y_2, z_2 их значениями из (7.2).

$$\begin{cases} x_3 = (x\cos u - y\sin u)\cos v + z\sin v, \\ y_3 = (y\cos u + x\sin u)\cos w - [z\cos v - (x\cos u - y\sin u)\sin v]\sin w, \\ z_3 = [z\cos v + (x\cos u - y\sin u.)\sin v]\cos w + (y\cos u + x\sin u)\sin w. \end{cases}$$

Группируя в последних выражениях слагаемые с x, y, z можно записать

$$\begin{cases} x_3 = C_{xx}x + C_{xy}y + C_{xz}z, \\ y_3 = C_{yx}x + C_{yy}y + C_{yz}z, \\ z_3 = C_{zx}x + C_{zy}y + C_{zz}z, \end{cases} \tag{7.3}$$

где

$$\begin{cases} C_{xx} = \cos u \cos v, \\ C_{xy} = -\sin u \cos v, \\ C_{xz} = \sin v, \\ C_{yx} = \sin u \cos w + \cos u \sin v \sin w, \\ C_{yy} = \cos u \cos w - \sin u \sin v \sin w, \\ C_{xz} = -\cos v \sin w \\ C_{zx} = -\cos u \sin v \cos w + \sin u \sin w, \\ C_{zy} = \sin u \sin v \cos w + \cos u \sin w, \\ C_{zz} = \cos v \cos w. \end{cases} \tag{7.4}$$

А теперь спроектируем точку $A_3(x_3, y_3, z_3)$ на экран (лучом, выходящим из точки E).

Начнем с проекции луча EA_3 на плоскость XOZ.

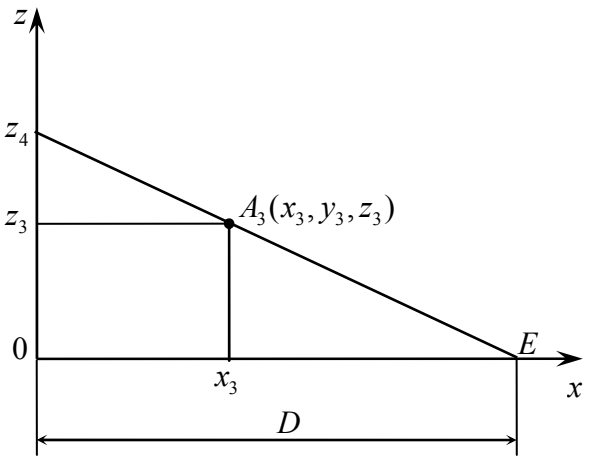


Рис.7.6. Проекция на плоскость XOZ.

Рассматривая подобные треугольники, можно записать, что

$$z_4 = z_3 \frac{D}{D - x_3}. \tag{7.5}$$

Аналогично проекция на плоскость *XOY*.

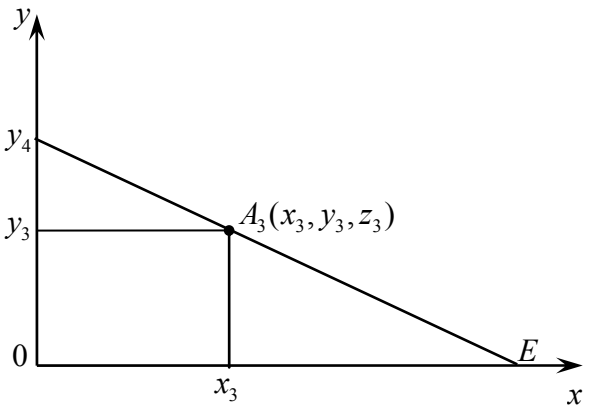


Рис.7.7. Проекция на плоскость *XOY*.

$$y_4 = y_3 \frac{D}{D - x_3}. \tag{7.6}$$

С учетом того, что начало координат системы *OXYZ* привязано к точке с экранными координатами *X0, Y0*, из рис.7.2 легко видеть, что точка $A_3(x_3, y_3, z_3)$ будет проектироваться в точку с экранными координатами

$$x_э = x_0 + round\left(y_3 \frac{D}{D - x_3} \right),$$

$$y_э = y_0 - round\left(z_3 \frac{D}{D - x_3} \right).$$

Обозначаем $\frac{D}{D - x_3} = k_p$ – коэффициент перспективы, получаем окончательно

$$\begin{cases} x_э = x_0 + round\left(y_3 k_p\right), \\ y_э = y_0 - round\left(z_3 k_p\right) \end{cases} \tag{7.7}$$

Итак, формулы (7.3) (с учетом коэффициентов (7.4)) и формулы (7.7) позволяют определить экранные координаты точки $A(x, y, z)$ после выполнения трех последовательных поворотов U, V, W и проектирования ее на экран из точки $E(D,0,0)$. Чтобы сформировать на экране изометрическое изображение поверхности $z = f(x, y)$ над областью, ограниченной `Xmin..Xmax, Ymin..Ymax`, будем последовательно пересекать эту поверхность плоскостями, перпендикулярными оси *X*, с шагом `Dx=(Xmax-Xmin)/Nx` и (или) плоскостями, перпендикулярными оси *Y*, с шагом `Dy=(Ymax-Ymin)/Ny`, где `Nx, Ny` – заданные целые положительные числа. Линии пересечения в виде ломаных (для простоты) будем проектировать на экран.

Изучение поверхности будем осуществлять рассматривая ее на экране при различных комбинациях U, V, W и, возможно, *D*. Обратим внимание на то, что при всех этих поворотах массив функции в узлах сетки на плоскости *XOY*, остается неизменным. Поэтому сформируем его (глобальный массив `Mo:Matr`) для заданных значений `Xmin, Xmax, Ymin, Ymax, Nx, Ny`.

```
procedure FormMo;
  var f:real;
begin
  Dx:=(Xmax-Xmin)/Nx; Dy:=(Ymax-Ymin)/Ny;
  Y:=Fxy(Xmin, Ymin); Zmin:=Y; Zmax:=Y;
```

```
  Xu:=(GetMaxX-Ny) div 2;
  PutA; Rectangle(Xu, 0, Xu+Ny, 5);
  for z:=0 to Ny do
    begin
      Y:=Ymin+Z*Dy; Line(Xu+Z,1,Xu+Z,4);
      for s:= 0 to Nx do
        begin
          X:=Xmin+S*Dx; f:=Fxy(X,Y);
          Mo[Z,S]:=f;
          if f<Zmin then Zmin:=f;
          if f>Zmax then Zmax:=f
        end
    end;
  Xs:=(Xmin+Xmax)/2;  Ys:=(Ymin+Ymax)/2;
  Zs:=(Zmin+Zmax)/2;  Kx:=(Xmax-Xmin)/Rx;
  Ky:=(Ymax-Ymin)/Ry; Kz:=(Zmax-Zmin)/Rz;
end;
```

Здесь помимо массива `Mo:Matr` вычисляются также `Xs,Ys,Zs:real` (глобальные) – средние значения *X, Y* и *Z*, а также `Kx,Ky,Kz:real` (также глобальные) – количество *X, Y, Z*, приходящееся на 1 пиксель вдоль соответствующей оси (при условии, что изучаемая поверхность сориентирована так, что соответствующий размер проектируется на экран в натуральную величину).

При расчете экранных координат характерных точек поверхности (в узлах сетки) согласно формулам (7.3), (7.4), (7.7) используются одни и те же углы U, V, W, расстояние до наблюдателя *D*, а, следовательно, одни и те же синусы и косинусы этих узлов и соответственно одни и те же коэффициенты формул (7.3).

Поэтому имеет смысл для заданного положения поверхности в

пространстве вычислить один раз все коэффициенты формул (7.3), а затем по ним сформировать массивы `Mxe,Mye:array[0..Ny,0..Nx] of integer` – экранных координат характерных точек поверхности (в узлах сетки). Примем также, что в этих массивах не учитываются значения *X0* и *Y0* в формулах (7.7), чтобы удобнее было при необходимости изменять положение изображения на экране без дополнительного (относительно трудоемкого) пересчета экранных координат.

```
procedure FormMe;
  var f,sinU,cosU,sinV,cosV,sinW,cosW,Cxx,Cxy,Cxz,
      Cyx,Cyy,Cyz,Czx,Czy,Czz,X3,Y3,Z3,Kp:real;
begin
  X:=U*Pi/180; sinU:=sin(x); cosU:=cos(x);
  V:=U*Pi/180; sinV:=sin(x); cosV:=cos(x);
  X:=W*Pi/180; sinW:=sin(x); cosW:=cos(x);
  Cxx:=cosU*cosV;   Cxy:=-sinU*cosV; Cxz:=sinV;
  Cyx:=sinU*cosW+cosU*sinV*sinW;
  Cyy:=cosU*cosW-sinU*sinV*sinW;
  Cyz:=-cosV*sinW;
  Czx:=sinU*sinW-cosU*sinV*cosW;
  Czy:=cosU*sinW+sinU*sinV*cosW;
  Czz:=cosV*cosW;
  Xu:=(GetMaxX-Wy)div2;
  PutA; Rectangle(Xu,0,Xu+Ny,5);
  for z:=0 to Ny do
    begin
      Y:=(Ymin+Z*Dy-Ys)/Ky; Line(Xu+Z,1, Xu+Z,4);
      for s:=0 to Nx do
        begin
          X:=(Xmin+S*Dx-Xs)/Kx;
          f:=(Mo[z,s]-Zs)/Kz;
          X3:=Cxx*X+Cxy*Y+Cxz*f;
          Y3:=Cyx*X+Cyy*Y+Cyz*f;
          Z3:=Czx*X+Czy*Y+Czz*f;
          Kp:=D/(D-X3);
          Mxe[z,s]:=round(Y3*Kp);
          Mye[z,s]:=-round(Z3*Kp/Hj);
        end
    end
end;
```

Обращаем внимание читателя на вычисление *X, Y* и *f* (значения функции) в цикле по *z*. Из текущего значения переменной вычитается ее среднее значение, после чего полученная разница делится на коэффициент (*Kx, Ky* или *Kz* соответственно). Этим самым изображение “центрируется” относительно точки *X0, Y0* и координаты преобразуемой точки поверхности выражаются в пикселях. И уже эти координаты далее используются при последующих поворотах и проектировании на экран.

При вычислении *Mye*[*z,s*] учитывается коэффициент *Hj* – отношение расстояния между соседними пикселями по вертикали к таковому по горизонтали.

И, наконец, завершающий этап – формирование собственно изображения поверхности.

```
procedure FormRel;
  var z,s:integer;
begin
  SetColor(c);
  for z:=0 to Ny do
    begin
      MoveTo(X0+Mxe[z,0],Y0+Mye[z,0]);
      for s:=1 to Nx do
      LineTo(X0+Mxe[z,s],Y0+Mye[z,s]);
    end;
  for z:=0 to Nx do
    begin
      MoveTo(X0+Mxe[z,o],Y0+Mye[z,o]);
      for z:=1 to Ny do
        LineTo(X0+Mxe[z,s],Y0+Mye[z,s]);
    end
  SetColor(15)
end;
```

В процедуре, `FormRel` сначала формируется Ny сечений поверхности плоскостями, перпендикулярными оси *Y*, а затем Nx сечений плоскостями параллельными оси *X*. Цвет изображения задается параметром *C*. Не рекомендуем задавать инверсный цвет, т.к. при наложении отдельных отрезков

ломаных или целых кривых они будут взаимно уничтожаться, что отрицательно скажется на качестве изображения.

Описанная последовательность этапов формирования изображения поверхности (массив `Mo`, массивы `Mxe, Mye`, формирование собственно изображения) позволяет при желании формировать движущиеся изображения поверхности. Проще всего перетаскивать изображение из одного места на экране на другое. Для этого по уже сформированным массивам `Mxe, Mye` изменяя с каким-то шагом *X0* и (или) *Y0* можно последовательно формировать изображение, выдерживать его требуемое время на экране,стирать, заменять новым (в новом месте). Если требуется демонстрировать в динамике эффекты вращения, то для каждого очередного кадра (с какими-то новыми значениями углов U, V, W или D) пересчитываются заново массивы `Mxe, Mye` (при неизменном массиве `Mo`). Пока идет пересчет предыдущий кадр “стоит” на экране. По мере готовности массивов `Mxe, Mye` старый кадр заменяется новым с помощью процедуры `FormRel`. Если бы требовалось замедлить темп вращения (вращений) выдержку очередного кадра можно было бы увеличить с применением процедуры `Delay(R)`, где `R` – время задержки в миллисекундах.

Приложения

```
Unit Serv;
   Interface
      uses crt, graph;
      const Hj:real=2.1; c:integer=15;
      type coef=array[-1..31] of real;
           coefr=array[-1..201] of real;
           coefl=array[-1..601] of real;
      var  l,h,l0,h0,xu,yu,x0,y0,s,z:integer;
           xn,yn,xk,yk,x,y,dx,dy,
      xmin,xmax,ymin,ymax:real;
           T10:string[10];
           Ts:string[80];
           j,j1,j2,j3:char;
           aw,bw:word;
      procedure Clear(x1,y1,x2,y2:integer);
      procedure PutA;
      procedure Ou(text:string);
      procedure Our(text:string; var x:real);
      procedure Oui(text:string; var x:integer);
      procedure Info;
      procedure X0Y0(God:boolean);
      procedure SystCoor;
      procedure Graphic(mo:coefl; c:integer);
      procedure InGraph;
   Implementation
       procedure Clear(x1,y1,x2,y2:integer);
       begin
        SetViewPort(x1,y1,x2,y2,true);
        ClearViewPort;
        SetViewPort (0,0,GetMaxX-1,GetMaxY-1,false);
       end;
      procedure PutA;
          begin
           Clear(0,0,GetMaxX-1,10);
          end;
      procedure Ou(text:string);
         begin
           OutTextXY(0,0,text);
         end;
      procedure Our(text:string; var X:real);
        begin
          PutA; OuttextXY(150,0,'Enter '+text);
          GoToXY(1,1);
          read(x);
```

```
  end;
procedure Oui(text:string; var x:integer);
 begin
   PutA;
   OuttextXy(150,0,'Enter '+text);
   GoToXY(1,1);
   read(x);
 end;
procedure Info;
  begin
     clear(0,Getmaxy-10,getmaxx-1,getmaxy-1);
     Outtextxy(0,getmaxy-10,ts);
   end;
procedure X0Y0(god:boolean);
   begin
     dx:=(xmax-xmin)/L; L0:=L;
     dy:=(ymax-ymin)/h; h0:=h;
     if god then
      if dy>dx*hj
        then
          begin
           L0:=round((xmax-xmin)/dy*hj);
           if L0<>0 then dx:=(xmax-xmin)/L0
                    else dx:=0;
           end
       else
         begin
           H0:=round((ymax-ymin)/(dx*hj));
           if H0<>0
               then dy:=(ymax-ymin)/H0
               else dy:=0;
         end;
    xu:=(getmaxx-L0) div 2;
    yu:=(getmaxy-H0) div 2;
    x0:=xu-round(xmin/dx);
    y0:=yu+round(ymax/dy);
    xn:=Xmin; xk:=Xmax;
    yn:=Ymin; yk:=Ymax;
 end;
 procedure SystCoor;
   begin
    SetWriteMode(1);
    Line(xu-10,y0,xu+L0+10,y0);
    Line(x0,yu+h0+10,x0,yu-10);
    SetwriteMode(0);
```

```
      end;
    procedure Graphic(mo:coefL; c:integer);
    var s,x,y,L:integer;
       begin
         L:=round(mo[-1]);
         if c>=0 then setcolor(c)
                 else setwritemode(1);
         moveto(xu,y0-round(mo[0]/dy));
         for s:=1 to L-1 do
           begin
             x:=xu+s; y:=y0-round(Mo[s]/dy);
             lineto(x,y);
             Putpixel(x,y,getmaxcolor-getpixel(x,y))
           end;
         if c>=0 then setcolor(15)
                   else setwritemode(0)
       end;
     procedure Ingraph;
       var e,gd,gm:integer;
           aw,bw:word;
       begin
           gd:=detect;
           initgraph(gd,gm,'d:\tp ');
           e:=graphresult;
           if e<>0
            then
              begin
                   write(grapherrormsg(e));
                   readln;
                   Halt(1);
              end
            else
              begin
                  getaspectratio(aw,bw);
                  Hj:=bw/aw;
                  L:=getmaxx-139;
                  H:=getmaxy-59;
       end;
     end;
begin
  ingraph
end.
```

```
Unit Groms;
interface
  uses crt,graph,serv;
  const xl :real=  0;
        xr :real=  5;
        hx :real=0.5;
        hy :real=0.5;
        se :integer=  5;
        sd :integer=  2;
        sh:integer=   2;
        sv:integer=2;
        rs :integer=  3;
        xsc:real=  0;
        ysc:real=  0;
        zmh:integer=-15;
        zmv:integer=-10;
        fr :integer=200;
        j4:char='1';
  procedure MARK;
  procedure NET;
  procedure NUM;
  procedure INSCR;
  procedure Service;
   Implementation
  procedure mark;
   var s,z,sn,sk,zn,zk,ss,zs:integer;
       J2:char;
   begin
     setwritemode(1);
     repeat
      puta;
      Ou('0-exit,  1-Rs,  2-Sd,  3-Xn,  4-Xk,  5-Hx,  6-Yn,
          7-Yk, 8-Hy, 9-Mark');
      Str(Rs,T10); Ts:='Rs='+T10;
      Str(Sd,T10); Ts:=Ts+', Sd='+T10;
      Str(Xn:1:Sd,T10); Ts:=Ts+', Xn='+T10;
      Str(Xk:1:Sd,T10); Ts:=Ts+', Xk='+T10;
      Str(Hx:1:Sd,T10); Ts:=Ts+', Hx='+T10;
      Str(Yn:1:Sd,T10); Ts:=Ts+', Yn='+T10;
      Str(Yk:1:Sd,T10); Ts:=Ts+', Yk='+T10;
      Str(Hy:1:Sd,T10); Ts:=Ts+', Hy='+T10;
      Info; J2:=Readkey;
      Case J2 of
        '1':Oui('Rs-a®§¬?a иваЁе ',Rs);
        '2':Oui('Sd-Є?«мЄ?бвм бЁ¬ў®«?ў Ї?б«п ¤Ґбпв€®ў®х
```

```
                Єа ЇЄЁ',Sd);
          '3':Our('Xn',Xn);
          '4':Our('Xk',Xk);
          '5':Our('Hx-Єа®Є иваЁе?ў ў§¤®ў¦ ®б? e',Hx);
          '6':Our('Yn',Yn);
          '7':Our('Yk',Yk);
          '8':Our('Hy- Єа®Є иваЁе?ў ў§¤®ў¦ ®б? Y',Hy);
          '9':Begin
               Sn:=round(Xn/Hx);
               Sk:=round(Xk/Hx);
               Zn:=round(Yn/Hy);
               Zk:=round(Yk/Hy);
               for S:=sn to Sk do
                if S<>0 then
                  begin
                    SS:=X0+round(S*Hx/Dx);
                    Line(SS,Y0-Rs,SS,Y0+Rs);
                  end;
               for Z:=Zn to Zk do
                if Z<>0 then
                 begin
                  Zs:=Y0-round(Z*Hy/Dy);
                  Line(X0-Rs,Zs,X0+Rs,Zs);
                 end
             end
           end
       until J2='0';
       SetWriteMode(0)
    end;

procedure Net;
  var sn,sk,zn,zk,sl,sr,zb,zt,ss,zs:integer;
    J2:char;
  begin
       SetWriteMode(1); SetLineStyle(1,0,1);
       repeat
         puta;
         ou('0-exit, 1-Xn, 2-Xk, 3-Hx, 4-Yn, 5-Yk, 6-Hy,
             7-Net');
         Str(Xn:1:3, t10); ts:='Xn='+t10;
         Str(Xk:1:3, t10); ts:=ts+', Xk='+t10;
         Str(Hx:1:3, t10); ts:=ts+', Hx='+t10;
         Str(Yn:1:3, t10); ts:=ts+', Yn='+t10;
         Str(Yk:1:3, t10); ts:=ts+', Yk='+t10;
         Str(Hy:1:3, t10); ts:=ts+', Hy='+t10;
```

```
        info; J2:=readkey;
        case J2 of
          '1':Our('Xn',Xn);
          '2':Our('Xk',Xk);
          '3':Our('Hx',Hx);
          '4':Our('Yn',Yn);
          '5':Our('Yk',Yk);
          '6':Our('Hy',Hy);
          '7':begin
                sn:=round(Xn/hx);
                sk:=round(xk/hx);
                zn:=round(yn/hy);
                zk:=round(yk/hy);
                sl:=X0+round(xn/dx);
                sr:=X0+round(xk/dx);
                zb:=Y0-round(yn/dy);
                zt:=Y0-round(yk/dy);
                for s:=sn to sk do
                  if s<>0 then
                     begin
                       ss:=X0+round(s*hx/dx);
                       Line(ss,zt,ss,zb)
                     end;
                for z:=zn to zk do
                  if z<>0 then
                     begin
                       zs:=Y0-round(z*hy/dy);
                       Line(sl,zs,sr,zs)
                     end
              end
        end
     until J2='0';
     SetWriteMode(0); SetLineStyle(0,0,1)
  end;
procedure Inscr;
  const  ds:integer=0;
         dz:integer=0;
  var Size:word;
      p,pf:pointer;
      s,z,h1,h2,L1,L2,Lt,ht,Sm,Zm,Sp,Zp:integer;
      J3,J4:char;
  begin
   L1:=TextWidth('2'); L2:=L1 div 2;
   H1:=TextHeight('2'); h2:=h1 div 2;
   Size:=ImageSize(0,0,L1,h1); ht:=2*h1;
```

```
GetMem(P,Size); GetMem(pf,Size);
repeat
  puta;
  Ou('0-exit,1-Text,2-Sh,3-Sv,4-Fr,5-Move,6-DelText');
  Str(Sh,t10); ts:='Sh='+t10;
  Str(Sv,t10); ts:=ts+', Sv='+t10;
  Str(Fr,t10); ts:=ts+', Fr='+t10;
  info; J3:=readkey;
  case J3 of
    '1':begin
          s:=0; z:=15;
          FreeMem(p, Size); FreeMem(pf,Size);
          puta; Ou('Text: ');
          GotoXy(7,1); readln(ts);
          Lt:=TextWidth(ts)+L1;
          Size:=ImageSize(0,0,Lt,Ht);
          GetMem(p,Size); GetMem(pf,Size);
          GetImage(s,z,s+Lt,z+ht,pf^);
          Clear(s,z,s+Lt,z+ht);
          OutTextXY(s+L2,z+h2,ts);
          GetImage(s,z,s+Lt,z+ht,p^);
          sm:=GetMaxX-Lt-1; zm:=GetMaxY-ht-1
        end;
   '2':Oui('Sh',Sh);
   '3':Oui('Sv',Sv);
   '4':Oui('Fr',Fr);
   '5':repeat
         puta; Ou('0-exit,2 4 6 8-Move, 5-Stop');
         J4:=ReadKey;
         If J4<>'0' then
           repeat
             If KeyPressed then J4:=ReadKey;
             case J4 of
               '2': Dz:=Sv;
               '4': Ds:=-Sh;
               '5': begin Ds:=0; Dz:=0 end;
               '6': Ds:=Sh;
               '8': Dz:=-Sv
             end;
             Sp:=s; zp:=z; s:=s+ds; z:=z+dz;
             If s<0 then s:=0;
             If s>sm then s:=sm;
             If z<0 then z:=0;
             If z>zm then z:=zm;
             PutImage(sp,zp,pf^,0);
```

```
                GetImage(s,z,s+Lt,z+ht,pf^);
                PutImage(s,z,p^,0); Delay(Fr)
              until J4='0'
            until J4='0';
      '6':PutImage(s,z,pf^,0)
     end
  until J3='0';
  FreeMem(p,Size); FreeMem(Pf,Size)
 end;

procedure Num;
  var Size:word;
      p,pf:pointer;
sn,sk,zn,zk,s,z,Lt,ht,L1,L2,Le,Le2,h1,h2,st,zt,x1,y1:inte
ger;
      J,J1:char;
  begin
    H1:=TextHeight('2'); L1:=TextWidth('2');
    H2:=h1 div 2; L2:=L1 div 2; Ht:=2*h1;
    repeat
      Puta; Ou('0-exit, 1-NumH, 2-NumV');
      clear(0,getmaxY-10,getmaxX-1,getmaxY-1);
      J:=ReadKey;
      Case J of
       '1': begin
              Le:=(se+1)*L1; Le2:=Le div 2;
              repeat
               Puta;
               Ou('0-ex, 1-Se, 2-Sd, 3-Ysc, 4-Xn, 5-Xk,
                   6-Hx, 7-ZmV, 8-NumH, 9-DelNumH');
               Str(Se,t10); ts:='Se='+t10;
               Str(Sd,t10); ts:=ts+', Sd='+t10;
               Str(Ysc:1:Sd,t10); ts:=ts+', Ysc='+t10;
               Str(Xn:1:Sd,t10); ts:=ts+', Xn='+t10;
               Str(Xk:1:Sd,t10); ts:=ts+', Xk='+t10;
               Str(Hx:1:Sd,t10); ts:=ts+', Hx='+t10;
               Str(ZmV,t10); ts:=ts+', ZmV='+t10;
               Info; J1:=ReadKey;
               case J1 of
                 '1': Oui('Se',Se);
                 '2': Oui('Sd',Sd);
                 '3': Our('Ysc',Ysc);
                 '4': Our('Xn',Xn);
                 '5': Our('Xk',Xk);
                 '6': Our('Hx',Hx);
```

```
                '7': Oui('ZmV', ZmV);
                '8': begin
                       sn:=round(Xn/Hx);
                       sk:=round(Xk/Hx);
                       Lt:=round((Sk-sn)*Hx/dx)+Le;
                       Size:=ImageSize(0,0,Lt,ht);
                       GetMem(pf,Size);
                       St:=x0-round(sn*hx/dx)-Le2;
                       Zt:=y0-round(ysc/dy)-ZmV-h1;
                       GetImage(st,zt,st+Lt,zt+ht,pf^);
                       for s:=sn to sk do
                        if s<>0 then
                         begin
                           X:=s*hx;
                           Str(X:se:sd,ts);
                           x1:=x0+round(x/dx)-Le2;
                           Clear(x1,zt,x1+Le,zt+ht);
                           OutTextXY(x1+L2,
                              zt+h2,ts)
                         end
                     end;
              '9':PutImage(st,zt,pf^,0)
            end
        until J1='0'
      end;
'2':begin
      Le:=(se+1)*L1; Le2:=Le div  2;
      repeat
        Puta;
        Ou('0-ex, 1-Se, 2-Sd, 3-Xsc, 4-Yn, 5-Yk,
            6-Hy,7-Zmh, 8-NumV, 9-DelNumV');
        Str(Se,t10); ts:='Se='+t10;
        Str(Sd,t10); ts:=ts+', Sd='+t10;
        Str(xsc:1:Sd,t10); ts:=ts+', Xsc='+t10;
        Str(Yn:1:Sd,t10); ts:=ts+', Yn='+t10;
        Str(Yk:1:Sd,t10); ts:=ts+', Yk='+t10;
        Str(Hy:1:Sd,t10); ts:=ts+', Hy='+t10;
        Str(ZmH,t10); ts:=ts+', ZmH='+t10;
        Info; J1:=ReadKey;
        case J1 of
          '1': Oui('Se',Se);
          '2': Oui('Sd',Sd);
          '3': Our('Xsc',xsc);
          '4': Our('Yn',Yn);
          '5': Our('Yk',Yk);
```

```
            '6': Our('Hy',Hy);
            '7': Oui('ZmH', ZmH);
            '8': begin
                   zn:=round(yn/Hy);
                   zk:=round(yk/Hy);
                   ht:=round((yk-yn)/dy)+2*h1;
                   Size:=ImageSize(0,0,Le,ht);
                   GetMem(pf,Size);
                   St:=x0+round(Xsc/dx)-Le2+Zmh;
                   if st<0 then st:=0;
                   Zt:=y0-round(yk/dy)-h1;
                   GetImage(st,zt,st+Le,zt+ht,pf^);
                   for z:=zn to zk do
                     if z<>0 then
                       begin
                         y:=z*hy; Str(y:se:sd,ts);
                         y1:=y0-round(y/dy)-h1;
                         Clear(st,y1,st+Le,y1+2*h1);
                         OutTextXY(st+L2,y1+h2,ts)
                       end
                 end;
             '9':PutImage(st,zt,pf^,0)
           end
         until J1='0'
       end
     end
   until J='0'
  end;

 Procedure Service;
  begin
    repeat
      PutA; Ou('0-exit, 1-Mark, 2-Net, 3-Num, 4-Inscr');
      j4:=readkey;
      case j4 of
        '1': Mark;
        '2': Net;
        '3': Num;
        '4': Inscr;
      end;
    until j4='0'
  end;
end.
```

```
Unit Polynom;
  interface
   uses serv;
   type kor=array[1..2,-1..30] of real;
   const Eps:real=1e-9;
  procedure Redpol(eps:real;var a:coef);
  function Radius(a:coef):real;
  function Horreal(a:coef;p:real):real;
  procedure Horcomp(a:coef;r,i:real;var re,im:real);
  procedure Alsumpol(a,b:coef;op:char;var c:coef);
  procedure Umnpol(a,b:coef;var c:coef);
  procedure Delpol(a,b:coef; var C,D:coef);
  function Hurvic(A:coef):integer;
  procedure Zamppl (A:Coef; L:real;var B:coef);
  function StepUst (A:coef; Eps:real):real;
  procedure KorPolSu (A:coef; Eps:real; var R,I:real);
  procedure Kvadrat (A:coef; var Korni:Kor);
  procedure KorpolMs (A:coef; Eps:real; var Korni:Kor);
  Implementation

  Procedure RedPol (Eps:real; var a:coef);
   var n:integer;
    begin n:=round(a[-1]);
      if abs(A[N])<Eps then n:=n-1;
     while (n>1) and  ( abs(A[n])<Eps) do dec(n);
     A[-1]:=n;
    end;

  function Radius (A:coef):real;
   var z,n:integer;
       amax:real;
   begin
       RedPol( Eps,A); n:=round(a[-1]);
       amax:=abs(a[0]);
       for z:=1 to n-1 do
        if abs(a[z])>amax then amax:=abs(a[z]);
       radius:=1+amax/abs(a[n]);
   end;
  function Horreal (A:coef; P:real):real;
   var S,N:integer;
       R:real;
   begin
     N:=round (A[-1]);
     R:=A[N];
     for S:=N-1 downto 0 do R:=R*P+A[S];
```

```
     Horreal:=R;
   end;

procedure Horcomp (A:coef; R,I:real; var Re,Im:real);
    var R1:real; n,z:integer;
    begin
       N:=round(A[-1]);Re:=A[n]; Im:=0;
       for z:=N-1 downto 0 do
        begin
          R1:=Re*R-Im*I+A[z];
          Im:=Re*I+Im*R; Re:=R1
        end;
    end;

procedure Alsumpol (A,B:coef; Op:char; var C:coef);
    var Na, Nb, Nc,S:integer;
    begin
       Na:=round(A[-1]); Nb:=round(B[-1]);
       if Na>Nb
         then Nc:=Na
         else Nc:=Nb;
       C[-1]:=Nc;
       if Na<Nc
        then
          for S:=Na+1 to Nc do A[S]:=0
        else for S:=Nb+1 to Nc do
          B[S]:=0;
       for S:=0 to Nc do
         if op='+' then C[S]:=A[S]+B[S]
                   else C[S]:=A[S]-B[S];
    end;

Procedure Umnpol (A,B:coef; var C:Coef);
  var Z,S,NA,NB,Nc:integer;
    begin
      Na:=round(A[-1]); Nb:=round(B[-1]); Nc:=Na+Nb;
      C[-1]:=Nc;
      for S:=0 to Nc do  C[s]:=0;
      for Z:=0 to Na do
        for S:=0 to Nb do
          C[Z+S]:=C[Z+S]+A[Z]*B[s]
    end;

Procedure Delpol (A,B:coef; var C,D:coef);
   var Z,S,Na,Nb,Nc,Nd:integer; Sum:real;
```

```
   begin
     Na:=round(A[-1]); Nb:=round(B[-1]);
     if Na<Nb
      then
       begin
         D:=A; Nd:=Na; Nc:=0; C[0]:=0
       end
      else
       begin
          Nd:=Nb-1; Nc:=Na-Nb;
          for z:=0 to Na do C[z]:=0;
          for Z:=Na-Nb downto 0 do
           begin
            Sum:=0;
            for S:=0 to nb-1 do
                Sum:=Sum+B[S]*C[Nb+Z-S];
            C[Z]:=(A[Nb+Z]-Sum)/B[Nb]
           end;
          for Z:=0 to Nd do
            begin
              Sum:=0;
              for S:=0 to Z do sum:=sum+B[S]*C[Z-S];
              D[Z]:=A[Z]-Sum;
            end
       end;
     C[-1]:=Nc; D[-1]:=Nd
   end;

Function Hurvic (A:coef):integer;
  var z,s,Hurv,n:integer;R:real;
  begin
    Hurv:=1; n:=round(a[-1]);
    if n>0 then
    while (abs(a[n])<Eps) and (n>0) do dec(n);
    if n>0 then
      begin
        if A[n]<0 then for z:=0 to n do A[z]:=-A[z];
        for z:=0 to n-1 do
          if A[z]<=Eps then Hurv:=-1;
        if (Hurv=1) and (N>2) then
          for z:=1 to n-2 do
            begin
              if A[n-z]<=Eps
                then Hurv:=-1
                else
```

```
                    begin
                      r:=A[n-z+1]/A[n-z];
                      s:=N-z;
                      while s>=2 do
                        begin
                          A[s-1]:=A[s-1]-A[s-2]*r;
                          dec(S,2)
                        end
                    end
              end;
          if A[1]<=Eps then Hurv:=-1
        end;
      Hurvic:=Hurv
    end;

Procedure Zamppl (a:Coef;L:REAL; var b:coef);
    var s,z,n:integer;
        f:real;
        Sum:real;
    begin
      f:=1; n:=round(A[-1]); B[-1]:=n;
      for s:=0 to n do
        begin
          Sum:=A[n-s];
          for z:=n-s-1 downto 0 do Sum:=-Sum*L+A[z];
          B[s]:=Sum/f;
          for z:=1 to n-s do A[z-1]:=A[z]*z;
          f:=f*(s+1)
        end
    end;
Function Stepust (A:coef; Eps:real):real;
    var Pmin,Pmax,P:real;
        B:Coef;
    begin
      Pmax:=Radius(A); Pmin:=-Pmax; P:=0;
      while Pmax-Pmin>2*Eps do
        begin
          ZamPPL(A,-P,B);
          if Hurvic(B)=1 then Pmax:=P
                         else Pmin:=P;
          P:=(Pmax+Pmin)/2
        end;
      StepUst:=-P
    end;
```

```
procedure KorPolSu (A:coef; Eps:real; var R,I:real);
  var Re,Im,Modul,Io,Min,H,Iv,Ini,Rad:real;
  begin
    Rad:=Radius(A); r:=-StepUst(A,Eps);
    Iv:=sqrt(sqr(Rad)-sqr(r)); Ini:=0; H:=Iv/10;
    Min:=1e25; Io:=0;
    while 2*H>Eps do
      begin
        I:=Ini;
        while I<=Iv do
          begin
            HorComp(A,R,I,Re,Im);
            Modul:=sqr(Re)+sqr(Im);
            if Modul<Min then
               begin
                 Io:=I; Min:=Modul
               end;
            I:=I+H
          end;
        Iv:=Io+h; Ini:=Io-H; H:=H/2
      end;
    I:=Io
  end;

procedure Kvadrat (A:coef; var korni:kor);
  var d,z,r,i:real;
      n,m:integer;
  begin
    n:=round(A[-1]); m:=round(Korni[1,-1]);
    case n of
      1:begin
          inc(m);
          Korni[1,m]:=-A[0]/A[1]; Korni[2,m]:=0
        end;
      2:begin
          d:=sqr(A[1])-4*A[0]*A[2]; z:=2*A[2];
          if d>=0
             then
               begin
                 inc(m); Korni[1,m]:=
                    (-A[1]+sqrt(d))/z;
                 Korni[2,m]:=0; inc(m);
                 Korni[1,m]:=(-A[1]-sqrt(d))/z;
                 Korni[2,m]:=0
               end
```

```
              else
                begin
                  r:=-A[1]/z; i:=sqrt(-d)/abs(z);
                  inc(m);
                  Korni[1,m]:=r; Korni[2,m]:=i;
                  inc(m);
                  Korni[1,m]:=r; Korni[2,m]:=-i
                end
        end
    end;
   Korni[1,-1]:=m
  end;

procedure KorpolMs (A:coef; Eps:real; var Korni:kor);
  var m,n:integer;
      i,r:real;
      E:Coef;
  begin
    RedPol(0,A);n:=round(A[-1]); m:=0;
    Korni[1,-1]:=n;
    while N>2 do
      begin
        KorPolSu(A,Eps,R,I); inc(m);
        Korni[1,m]:=r; Korni[2,m]:=abs(i);
        if abs(i)<=Eps
           then
             begin
               E[-1]:=1; E[0]:=-r; E[1]:=1; dec(n)
             end
           else
             begin
               E[-1]:=2; E[0]:=sqr(r)+sqr(i);
               E[1]:=-2*r;
               E[2]:=1; inc(m); Korni[1,m]:=r;
               Korni[2,m]:=-abs(i); dec(n,2)
             end;
        DelPOl(A,E,A,E)
      end;
    Korni[1,-1]:=m; Kvadrat(A,Korni)
  end;
end.
```

```
Program KorPols;
  uses Crt,Graph,Serv,Polynom;
  const  Eps:real=1e-9;
         Hiz:real=0.2;
         Mi:integer=5000;
  var A:coef;
      Korni:Kor;
      Rad:real;
      Mo:Coefl;

  procedure InpPol(Id:char; var A:Coef);
    var s,n:integer;
    begin
      Oui('n-step.pol. '+id,n); A[-1]:=n;
      for s:=0 to n do
        begin
          Str(s,T10);
          Our(Id+'['+T10+']',A[s])
        end
    end;

  procedure OutPol(Id:char; A:coef; x:integer);
    var S,n:integer; T20:string;
    begin
      n:=round(A[-1]);
      for S:=0 to n do
        begin
          Str(S,T10); Str(A[s],Ts);
          OutTextxy(x,(s+2)*10, Id+'['+T10+']='+Ts)
        end
    end;

  procedure OutRoots(Korni:Kor);
    var S,n:integer; T20:string;
    begin
      n:=round(Korni[1,-1]);
      for s:=1 to n do
        begin
          Str(s,T10); Str(Korni[1,s],Ts);
          Str(korni[2,s],T20);
          Ts:='p['+t10+']='+Ts;
          if Korni[2,s]>0 then Ts:=Ts+'+j*'
                          else Ts:=Ts+'-j*';
          OutTextXY(20, (s+2)*10, Ts+T20)
        end
```

```
  end;

function  Fxy(x,y:real):real;
  var Re,Im:real;
  begin
    HorComp(A,x,y,Re,Im);
    Fxy:=sqrt(sqr(Re)+Sqr(Im));
  end;

procedure Izo;
  const Mi=20000;
  var Nr,z,s:integer;
      Nrr,x,y:real;
      Mnr:array[-1..600] of integer;
  begin
    X0Y0(true);
    Cleardevice;
    for z:=-1 to H0 do
      begin
        Nrr:=Fxy(Xmin-Dx,Ymin+z*Dy)/Hiz;
        if abs(Nrr)>Mi then Mnr[z]:=Mi
                       else Mnr[z]:=trunc(Nrr);
      end;
    for s:=0 to L0 do
      begin
        x:=xmin+s*dx;
        for z:=-1 to H0 do
          begin
            Nrr:=Fxy(x,Ymin+z*dy)/Hiz;
            if abs(Nrr)>Mi then Nr:=Mi
                           else Nr:=trunc(Nrr);
            if Z>-1 then
              if (Nr<>Mnr[z-1]) or (Nr<>Mnr[z]) then
                if abs(Nrr)<Mi then
                  PutPixel(Xu+s,Yu+H0-z,
                    round(abs(Nrr)) mod 16);
            Mnr[z]:=Nr;
          end
      end
  end;

procedure IzoModP;
  begin
    Rad:=Radius(A);
    repeat
```

```
        PutA;
        Ou('0-exit, 1-Xmin, 2-Xmax, 3-Ymin, 4-Ymax,
            5-Hiz, 6-Izo');
        Str(Xmin:1:3,T10); Ts:='Xmin='+T10;
        Str(Xmax:1:3,t10); Ts:=Ts+', Xmax='+T10;
        Str(Ymin:1:3,T10); Ts:=Ts+', Ymin='+T10;
        Str(Ymax:1:3,T10); Ts:=Ts+', Ymax='+T10;
        Str(Hiz:1:3,T10);  Ts:=Ts+', Hiz='+T10;
        Info; J1:=ReadKey;
        case J1 of
          '1': Our('Xmin', Xmin);
          '2': Our('Xmax', Xmax);
          '3': Our('Ymin', Ymin);
          '4': Our('Ymax', Ymax);
          '5': Our('Hiz', Hiz);
          '6': Izo;
        end
      until J1='0';
  end;

  begin
    A[-1]:=5; A[0]:=1; A[1]:=0; A[2]:=0;
    A[3]:=0; A[4]:=0; A[5]:=1;
    Xmin:=-1.2; Xmax:=1.2;
    Ymin:=-1.2; Ymax:=1.2; Eps:=1e-9;
    repeat
      PutA;
      Ou('Esc-Exit,1-InpPol,2-IzoLines,3-Eps,
          4-OutRoots,5-OutPol');
      J:=ReadKey;
      case J of
        '1': InpPol('A',A);
        '2': IzoModP;
        '3': Our('Eps',Eps);
        '4': begin
               KorPolMs(A,Eps,Korni);
               Clear(0,20,GetMaxX-1,GetMaxY-15);
               OutRoots(Korni);
             end;
        '5': OutPol('A', A,300);
       end;
    until J=#27;
    Closegraph
  end.
program OSystUr;
```

```
  uses Crt,Serv,Graph;
  const Nmax=10;
  type Matr=array[1..Nmax,1..Nmax+1] of real;
  var Xk,X:coef; A:Matr; z,s,n:integer;
      sum:real; T20:string[20];

procedure SystUr(n:integer; A:Matr; var x:Coef);
  const Eps=1e-9;
  var Zmax,z,s,i,j:integer;
      r:real;
      b:boolean;
  begin
    b:=true;
    if n=1
      then
        if abs(A[1,1])<Eps then b:=false
                           else X[1]:=A[1,2]/A[1,1]
      else
        begin
          i:=1;
          while (i<=n-1) and b do
            begin
              Zmax:=i;
              for z:=i+1 to n do
                if abs(A[z,i])>abs(A[zMax,i])
                  then Zmax:=z;
              if abs(A[Zmax,i])<Eps
                then b:=false
                else
                  begin
                    if Zmax>i then
                      for j:=i to n+1 do
                        begin
                          r:=A[i,j]; A[i,j]:=A[Zmax,j];
                          A[Zmax,j]:=r
                        end;
                    for z:=i+1 to n do
                      if abs(A[z,i])>Eps then
                        begin
                          r:=A[z,i]/A[i,i];
                          for s:=i+1 to n+1 do
                            A[z,s]:=A[z,s]-r*A[i,s]
                        end
                  end;
             inc(i)
```

```
          end;
        if b then
          if abs(A[n,n])<Eps then b:=false;
        if b then
          begin
            X[n]:=A[n,n+1]/A[n,n];
            for i:=n-1 downto 1 do
              begin
                r:=0;
                for s:=i+1 to n do
                  r:=r+A[i,s]*X[s];
                X[i]:=(A[i,n+1]-r)/A[i,i]
              end
          end
    end;
  if b then X[-1]:=n else X[-1]:=0
end;

 begin
   A[1,1]:=5; A[1,2]:=2; n:=1; Xk[1]:=0.4;
   repeat
     PutA;
     Ou('Esc-exit, 1-n, 2-inpXk, 3-inpA, 4-outX');
     str(n,t10); Ts:='n='+t10;
     Info; j:=ReadKey;
     case j of
       '1': Oui('n=',n);
       '2': for s:=1 to n do
            begin
              str(s,t10);
              Our('Xk['+t10+']', Xk[s]);
            end;
       '3': for z:=1 to n do
              begin
                Sum:=0; Str(z,t10);
                for s:=1 to n do
                  begin
                    Str(s,Ts);
                    Our('A['+t10+','+Ts+']',A[z,s]);
                    Sum:=Sum+A[z,s]*Xk[s]
                  end;
                A[z,n+1]:=Sum
              end;
       '4': begin
              SystUr(n,A,X);
```

```
              Clear(0,15,getmaxx-1,getmaxy-15);
              if X[-1]=0
                then
          OutTextXY(50,20,'Solution is not found')
                else
                  for s:=1 to n do
                    begin
                      Str(s,T10); Str(Xk[s],Ts);
                      Ts:='Xk['+T10+']='+Ts;
                      Str(X[s],t20);
                      Ts:=Ts+'   '+'X['+T10+']='+T20;
                      OutTextXY(50,(s+2)*10,Ts)
                     end
            end{'4'}
     end{case}
    until j=#27;
    CloseGraph
end.
```

```
Program OSystUr;
   uses Crt,Serv,Graph;
   const Nmax=10;
   type Matr=array[1..Nmax,1..Nmax+1] of real;
   var Xk,X:coef; A:Matr; z,s,n:integer;
       sum:real; T20:string[20];

 procedure SystUr(n:integer; A:Matr; var x:Coef);
   const Eps=1e-9;
   var Zmax,z,s,i,j:integer;
       r:real;
       b:boolean;
   begin
     b:=true;
     if n=1
       then
         if abs(A[1,1])<Eps then b:=false
                            else X[1]:=A[1,2]/A[1,1]
       else
         begin
           i:=1;
           while (i<=n-1) and b do
             begin
               Zmax:=i;
               for z:=i+1 to n do
                 if abs(A[z,i])>abs(A[zMax,i]) then
                   Zmax:=z;
               if abs(A[Zmax,i])<Eps
                 then b:=false
                 else
                   begin
                     if Zmax>i then
                       for j:=i to n+1 do
                         begin
                           r:=A[i,j]; A[i,j]:=A[Zmax,j];
                           A[Zmax,j]:=r
                         end;
                     for z:=i+1 to n do
                       if abs(A[z,i])>Eps then
                         begin
                           r:=A[z,i]/A[i,i];
                           for s:=i+1 to n+1 do
                             A[z,s]:=A[z,s]-r*A[i,s]
                         end
                   end;
```

```
            inc(i)
          end;
        if b then
          if abs(A[n,n])<Eps then b:=false;
        if b then
          begin
            X[n]:=A[n,n+1]/A[n,n];
            for i:=n-1 downto 1 do
              begin
                r:=0;
                for s:=i+1 to n do
                  r:=r+A[i,s]*X[s];
                X[i]:=(A[i,n+1]-r)/A[i,i]
              end
          end
    end;
  if b then X[-1]:=n else X[-1]:=0
end;

 begin
   A[1,1]:=5; A[1,2]:=2; n:=1; Xk[1]:=0.4;
   repeat
     PutA;
     Ou('Esc-exit, 1-n, 2-inpXk, 3-inpA, 4-outX');
     str(n,t10); Ts:='n='+t10;
     Info; j:=ReadKey;
     case j of
       '1': Oui('n=',n);
       '2': for s:=1 to n do
            begin
              str(s,t10);
              Our('Xk['+t10+']', Xk[s]);
            end;
       '3': for z:=1 to n do
              begin
                Sum:=0; Str(z,t10);
                for s:=1 to n do
                  begin
                    Str(s,Ts);
                    Our('A['+t10+','+Ts+']',A[z,s]);
                    Sum:=Sum+A[z,s]*Xk[s]
                  end;
                A[z,n+1]:=Sum
              end;
       '4': begin
```

```
                SystUr(n,A,X);
                clear(0,15,getmaxx-1,getmaxy-15);
                if X[-1]=0
                  then
                    OutTextXY(50,20,'Solution is not
                              found')
                  else
                    for s:=1 to n do
                      begin
                        Str(s,T10); Str(Xk[s],Ts);
                        Ts:='Xk['+T10+']='+Ts;
                        Str(X[s],t20);
                        Ts:=Ts+'   '+'X['+T10+']='+T20;
                        OutTextXY(50,(s+2)*10,Ts)
                       end
            end{'4'}
     end{case}
   until j=#27;
   CloseGraph
end.
```

```
Program AprPol;
 uses Crt,Graph,Serv;
 const Nmax=15;
 type Matr= array[1..Nmax, 1..Nmax+1] of real;
 const
  m:integer=5;
  n:integer=0;
 var
  Ak,A:coef; D:Matr;
  Mo:coefl; E:real;
  Mx,My:coefr;
  T20:string[20];

 procedure SystUr(n:integer; A:Matr; var x:Coef);
  const Eps=1e-9;
  var zmax,z,s,i,j:integer;
      r:real;
      b:boolean;
  begin
   b:=true;
   if n=1
    then
     if abs(A[1,1])<Eps then b:=false
                        else X[1]:=a[1,2]/a[1,1]
    else
     begin
      i:=1;
      while (i<=n-1) and b do
       begin
        zmax:=i;
        for z:=i+1 to n do
         if abs(A[z,i])>abs(A[zmax,i]) then zmax:=z;
        if abs(A[zmax,i])<Eps
         then b:=false
         else
           begin
             if zmax>i then
              for j:=i to n+1 do
                begin
                  r:=a[i,j]; a[i,j]:=a[zmax,j];
                  a[zmax,j]:=r
                end;
             for z:=i+1 to n do
               if abs(a[z,i])>Eps then
                 begin
```

```
                    r:=a[z,i]/a[i,i];
                    for s:=i+1 to n+1 do
                      a[z,s]:=a[z,s]-r*a[i,s]
                  end
            end;
          inc(i)
        end;
      if b then
        if abs(a[n,n])<Eps then b:=false;
      if b then
        begin
          x[n]:=a[n,n+1]/a[n,n];
          for i:=n-1 downto 1 do
            begin
              r:=0;
              for s:=i+1 to n do
                r:=r+a[i,s]*x[s];
              x[i]:=(a[i,n+1]-r)/a[i,i]
            end
        end
      end;
     if b then x[-1]:=n else x[-1]:=0
  end;

function HorReal(A:coef;X:real):real;
 var s,n:integer;
     sum:real;
 begin
   n:=round(A[-1]); sum:=A[n];
   for s:=n-1 downto 0 do
     sum:=sum*x+a[s];
   HorReal:=sum
 end;

procedure intpol(x,y:coefr; var a:coef);
  var z,s,n:integer;
  begin
    n:=round(x[-1]);
    for z:=1 to n+1 do
      begin
        d[z,1]:=1;
        d[z,n+2]:=y[z-1];
        for s:=2 to n+1 do
          d[z,s]:=d[z,s-1]*x[z-1]
      end;
```

```
    systur(n+1,d,a);
    if a[-1]>0 then
      begin
        a[-1]:=n;
        for s:=0 to n do
          a[s]:=a[s+1]
      end
  end;

 procedure NaimKV(x,y:coefr; n:integer; var E:real;
                  var A:coef);
  var m,z,s:integer;
      Sum:real;
      B,C:coefr;
      D:matr;
  begin
    m:=round(x[-1]);
    for s :=0 to m do b[s]:=1;
    for z :=0 to 2*n do
      begin
        c[z]:=0; sum:=0;
        for s:=0 to m do
          begin
            c[z]:=c[z]+b[s];
            if z <=n then sum:=sum+b[s]*y[s];
            b[s]:=b[s]*x[s]
          end;
        if z<=n then d[z+1,n+2]:=sum
      end;
    for z:=1 to n+1 do
      for s:=1 to n+1 do D[z,s]:=C[z+s-2];
    systur(n+1,D,A); A[-1]:=n;
    for s:=0 to n do A[s]:=A[s+1]; E:=0;
    for s:=0 to m do
      E:=E+sqr(HorReal(A,x[s])-y[s]);
  end;

procedure InpPol(Id:string; var A:Coef);
  var s,n:integer;
  begin
    Oui('n-бвЃЇi-м Ї®«i-®¬  '+Id,n);
    A[-1]:=n;
    for s:=n downto 0 do
      begin
        Str(s,t10);
```

```
       Our(Id+'['+t10+']',a[s]);
     end
 end;

procedure OutPol(Id:String; A:Coef; x:integer);
  var s,n:integer;
  begin
    n:=round(A[-1]);
    for s:=0 to n do
      begin
        Str(s,t10); Str(A[s],Ts);
        OutTExtXY(x, (s+2)*10,Id+'['+t10+']='+ts)
      end
  end;

Procedure FormMas(a:Coef);
  var
    s:integer;
  begin
    Mo[-1]:=L;
    Dx:=(Xk-Xn)/L;
    for s:=0 to L do
      Mo[s]:=HorReal(A,Xn+s*Dx)
  end;

Procedure Points;
  var s:integer;
  begin
    Dx:=(Xk-Xn)/m; Mx[-1]:=m; My[-1]:=m;
    Xmin:=Xn; Xmax:=Xk; Ymin:=0; Ymax:=0;
    for s:=0 to m do
      begin
        Mx[s]:=Xn+s*Dx;
        y:=HorReal(Ak,Mx[s]); My[s]:=y;
        if y<Ymin then Ymin:=y;
        if y>Ymax then Ymax:=y;
      end;
    X0Y0(false); Cleardevice;
    SystCoor; Setcolor(15);
    for s:=0 to m do
      circle(X0+round(Mx[s]/Dx), Y0-round(My[s]/Dy),3)
  end;

Procedure Diap;
 begin
```

```
   repeat
     PutA; Ou('0-exit, 1-InpAk, 2-Xn, 3-Xk, 4-m');
     Str(Xn:1:3,T10); Ts:='Xn='+T10;
     Str(Xk:1:3,T10); Ts:=Ts+', Xk='+T10;
     Str(m,T10); Ts:=Ts+', m='+T10;
     Info; J1:=ReadKey;
     case J1 of
       '1': InpPol('Ak',Ak);
       '2': Our('Xn',Xn);
       '3': Our('Xk',Xk);
       '4': Oui('m',m);
     end;
   until J1='0';
   Points;
 end;

Procedure FormGraph(a:Coef);
  begin
    FormMas(a);
    Graphic(Mo,c);
  end;

 begin
   Ak[-1]:=2; Ak[0]:=1; Ak[1]:=2; Ak[2]:=1;
   Diap; Xn:=-2; Xk:=3; c:=15;
   repeat
     PutA;
     Ou('Esc-exit,1-Diap,2-C,3-GrAk,4-GrInt,5-n,
         6-GrMNK,7-OutAk,8-OutA');
     Str(C,T10); Ts:='C='+T10;
     Str(n,T10); Ts:=Ts+', n='+T10;
     Info; J:=ReadKey;
     case J of
       '1': Diap;
       '2': Oui('C',C);
       '3': FormGraph(Ak);
       '4': begin
              IntPol(Mx,My,A);
              FormGraph(A)
            end;
       '5': Oui('n',n);
       '6': begin
              NaimKv(Mx,My,n,E,A); FormGraph(A);
              Str(E,Ts); Clear(400,30,GetMaxX-1,40);
              OutTextXY (400,30,'E='+Ts)
```

```
          end;
      '7': OutPol('Ak',Ak,50);
      '8': begin
             Clear(320,20,GetMaxX-1,
                   20+10*round(a[-1]));
             OutPol('A',A,320);
           end;
    end
  until J=#27;
  CloseGraph
end.
```

```
Program BSpLGrs;
  uses Crt, Graph, Serv;
  const Np:integer=0;
  var Mo:coefL;
  function f(Np:integer; x:real):real;
    var y:real;
     begin
       if (x<=-2) or (x>=2) then f:=0;
       if (x>-2) and (x<=-1) then
         begin
           y:=x+2;
           case Np of
             0: f:=y*y*y/6;
             1: f:=sqr(y)/2;
             2: f:=y;
             3: f:=1
           end
         end;
       if (x>-1) and (x<=0) then
         case Np of
           0: f:=2/3-0.5*sqr(x)*(x+2);
           1: f:=-0.5*x*(3*x+4);
           2: f:=-3*x-2;
           3: f:=-3;
         end;
       if (x>0) and (x<=1) then
         case Np of
           0: f:=2/3+0.5*sqr(x)*(x-2);
           1: f:=0.5*x*(3*x-4);
           2: f:=3*x-2;
           3: f:=3;
         end;
       if (x>1)  and (x<=2) then
         begin
           y:=2-x;
           case Np of
             0: f:=y*sqr(y)/6;
             1: f:=-sqr(y)/2;
             2: f:=y;
             3: f:=-1
           end
         end
    end;
   begin
      c:=15;
```

```
  repeat
    PutA;
    Ou('esc-exit, 1-L, 2-h, 3-Np, 4-c, 5-graphic');
    Str(L,t10); Ts:='L='+t10;
    Str(h,t10); Ts:=Ts+', h='+t10;
    Str(Np,t10); Ts:=Ts+', Np='+t10;
    Str(c,t10); Ts:=Ts+', c='+t10;
    info; J:=Readkey;
    case J of
      '1': oui('L',L);
      '2': oui('h',h);
      '3': oui('Np',Np);
      '4': oui('c',c);
      '5': begin
             xmin:=-2; xmax:=2;
             ymin:=0;  ymax:=0;
             Dx:=(xmax-xmin)/L;
             Mo[-1]:=L;
             for s:=0 to L-1 do
               begin
                 x:=xmin+s*Dx; y:=f(np,x);
                 if (s=L) and (Np=3) then y:=0;
                 Mo[s]:=y;
                 if y<ymin then ymin:=y;
                 if y>ymax then ymax:=y;
               end;
             Mo[L]:=0;
             x0y0(false); Cleardevice; Systcoor;
             Graphic(Mo,c)
           end
    end
  until j=#27;
  Closegraph
end.
```

```
Program DemSpl;
 uses Crt,Graph,Serv;
 const
  c1:real=2;
  w1:real=1.5;
  c2:real=0.5;
  w2:real=5;
  x1:real=-2;
  x2:real=3;
  m:integer=10;
  Ngl:integer=1;
  Ngr:integer=2;
  D0:real=0;
  Dm:real=0;
 var
  Mx,My,A0,A1,A2,A3:Coefr;
  Mo:coefl;
  Dx1:real;
  i:integer;

 function SplintN(A0,A1,A2,A3:coefr;x:real):real;
  var
   m,s:integer;
  begin
   s:=1;
   while Mx[s-1]<x do inc(s);
   if s>1 then dec(s);
   x:=x-Mx[s-1];
   SplintN:=((A3[s]*x+A2[s])*x+a1[s])*x+a0[s];
  end;

 procedure SplineN(X,Y:coefr;Ngl,Ngr:integer;
                   D0,Dm:real);
   var
    m,s:integer;
    h,h1,e,v,z,r:real;
    a,b:coefr;
   begin
    m:=round(x[-1]);
    case Ngl of
     1: begin
         h:=x[1]-x[0]; A[0]:=-0.5;
         B[0]:=1.5*((y[1]-y[0])/h-d0)/h;
        end;
     2: begin
```

```
        A[0]:=0; b[0]:=d0/2;
       end;
   end;
   for s:=1 to m-1 do
    begin
     h:=x[s]-x[s-1]; h1:=x[s+1]-x[s];
     e:=2*(h+h1);
     r:=3*((y[s+1]-y[s])/h1-(y[s]-y[s-1])/h);
     z:=e+a[s-1]*h; a[s]:=-h1/z;
     b[s]:=(r-b[s-1]*h)/z;
    end;
   case Ngr of
    1: begin
        h:=x[m]-x[m-1];
        a2[m+1]:=(3*(dm-(y[m]-y[m-1])/h)-
                 b[m-1]*h)/((a[m-1]+2)*h);
       end;
    2: a2[m+1]:=dm/2;
   end;
   for s:= m-1 downto 0 do
     a2[s+1]:=a[s]*a2[s+2]+b[s];
   for s:=1 to m do
    begin
     h:=x[s]-x[s-1];
     a1[s]:=(y[s]-y[s-1])/h-2/3*a2[s]*h-1/3*a2[s+1]*h;
     a3[s]:=(a2[s+1]-a2[s])/(3*h);
     a0[s]:=y[s-1];
    end;
   a0[-1]:=m;
 end;

procedure InpD;
 begin
  repeat
   putA;
   Ou('0-exit,1-C1,2-w1,3-C2,4-w2,5-x1,6-x2,7-m,
       8-SystCoor,9-points');
   str(c1:1:3,T10); Ts:='C1='+T10;
   str(w1:1:3,T10); Ts:=Ts+', w1='+T10;
   str(c2:1:3,T10); Ts:=Ts+', C2='+T10;
   str(w2:1:3,T10); Ts:=Ts+', w2='+T10;
   str(x1:1:3,T10); Ts:=Ts+', x1='+T10;
   str(x2:1:3,T10); Ts:=Ts+', x2='+T10;
   str(m:1,T10);    Ts:=Ts+', m='+T10;
   Info; j1:=readKey;
```

```
   case j1 of
    '1': Our('C1',C1);
    '2': Our('w1',w1);
    '3': Our('C2',C2);
    '4': Our('w2',w2);
    '5': Our('x1',x1);
    '6': Our('x2',x2);
    '7': Oui('m',m);
    '8': begin
          xMax:=x2; xMin:=x1;
          yMax:=0;  yMin:=0;
          Mx[-1]:=m; My[-1]:=m;
          Dx:=(x2-x1)/m;
          for s:=0 to m do
           begin
            x:=x1+s*Dx; Mx[s]:=x;
            y:=C1*cos(w1*x)+C2*sin(w2*x);
            My[s]:=y;
            if y<yMin then yMin:=y;
            if y>yMax then yMax:=y;
           end;
          X0Y0(false); cleardevice;
          SystCoor;
         end;
    '9': for s:=0 to m do
          circle(X0+round(Mx[s]/Dx),
                 Y0-round(My[s]/Dy),3);
   end;
  until j1='0';
end;

begin
 c:=15;
 repeat
  PutA;
  Ou('Esc-exit,1-InpD,2-c,3-Grf,4-Ngl,5-Ngr,6-D0,7-Dm,
      8-GrSpl');
  str(c,T10);        Ts:='c='+T10;
  str(Ngl,T10);      Ts:=Ts+', Ngl='+T10;
  str(Ngr,T10);      Ts:=Ts+', Ngr='+T10;
  str(D0:1:3,T10);   Ts:=Ts+', D0='+T10;
  str(Dm:1:3,T10);   Ts:=Ts+', Dm='+T10;
  Info; j:=ReadKey;
  case j of
   '1': InpD;
```

```
    '2': Oui('c',c);
    '3': begin
          SetColor(c); Mo[-1]:=L;
          for s:=0 to L do
           begin
            x:=x1+s*Dx;
            Mo[s]:=c1*cos(w1*x)+c2*sin(w2*x);
           end; {X0Y0(false); SystCoor;}
          Graphic(Mo,c);
         end;
    '4': Oui('Ngl',Ngl);
    '5': Oui('Ngr',Ngr);
    '6': Our('D0',D0);
    '7': Our('Dm',Dm);
    '8': begin
          Dx1:=(x2-x1)/m; Mx[-1]:=m;
          for s:=0 to m do
           begin
            Mx[s]:=x1+s*Dx1;
            My[s]:=c1*cos(w1*Mx[s])+c2*sin(w2*Mx[s]);
           end;
          SplineN(Mx,My,Ngl,Ngr,D0,Dm);
          Mo[-1]:=L; Dx:=(x2-x1)/L;
          for i:=0 to L do
           begin
            x:=x1+i*Dx;
            Mo[i]:=splintn(A0,A1,A2,A3,x);
         end;
   Graphic(mo,c);
  end;
  end;
 until j=#27;
end.
```

```
Program DemBSpl;
 uses Crt,Graph,Serv;
 const
  c1:real=2;
  w1:real=1.5;
  c2:real=0.5;
  w2:real=5;
  x1:real=-2;
  x2:real=3;
  m:integer=10;
  Ngl:integer=1;
  Ngr:integer=2;
  D0:real=0;
  Dm:real=0;
 var
  Mx,My,B:Coefr;
  Mo:coefl;
  Dx1:real;
  i:integer;

 function BSplint(x:real):real;
  var
   s:integer;
  begin
   s:=trunc(x); x:=frac(x);
   if s>m then BSplint:=(B[m-1]+4*B[m]+B[m+1])/6
          else BSplint:=B[s-1]*(1-x)*sqr(1-x)/6+
                        B[s]*(2/3+0.5*sqr(x)*(x-2))+
                        B[s+1]*(2/3-0.5*sqr(x-1)*(x+1))+
                        B[s+2]*x*sqr(x)/6;
  end;

 procedure BSpline(Y:CoefR; Ngl,Ngr:integer; D0,Dm:real);
   var
    s:integer;
    h,z:real;
    A,C:CoefR;
   begin
    m:=round(Y[-1]); h:=1;
    case Ngl of
     1: begin
         A[0]:=-0.5;
         C[0]:=(3*Y[0]+h*D0)/2;
        end;
     2: begin
```

```
         A[0]:=0;
         C[0]:=Y[0]-D0*sqr(h)/6;
        end;
    end;
    for s:=1 to m-1 do
     begin
       z:=A[s-1]+4; A[s]:=-1/z;
       C[s]:=(6*Y[s]-C[s-1])/z
     end;
    case Ngr of
     1: B[m]:=(-C[m-1]+3*Y[m]-h*Dm)/(2+A[m-1]);
     2: B[m]:=Y[m]-Dm*sqr(h)/6
    end;
    for s:= m-1 downto 0 do
      B[s]:=A[s]*B[s+1]+C[s];
    B[-1]:=6*Y[0]-B[1]-4*B[0];
    B[m+1]:=6*Y[m]-B[m-1]-4*B[m];
 end;

procedure InpD;
 begin
  repeat
   putA;
   Ou('0-exit,1-C1,2-w1,3-C2,4-w2,5-x1,6-x2,7-m,
       8-SystCoor,9-points');
   str(c1:1:3,T10); Ts:='C1='+T10;
   str(w1:1:3,T10); Ts:=Ts+', w1='+T10;
   str(c2:1:3,T10); Ts:=Ts+', C2='+T10;
   str(w2:1:3,T10); Ts:=Ts+', w2='+T10;
   str(x1:1:3,T10); Ts:=Ts+', x1='+T10;
   str(x2:1:3,T10); Ts:=Ts+', x2='+T10;
   str(m:1,T10);    Ts:=Ts+', m='+T10;
   Info; j1:=readKey;
   case j1 of
    '1': Our('C1',C1);
    '2': Our('w1',w1);
    '3': Our('C2',C2);
    '4': Our('w2',w2);
    '5': Our('x1',x1);
    '6': Our('x2',x2);
    '7': Oui('m',m);
    '8': begin
           xMax:=x2; xMin:=x1;
           yMax:=0;  yMin:=0;
           Mx[-1]:=m; My[-1]:=m;
```

```
          Dx:=(x2-x1)/m;
          for s:=0 to m do
           begin
            x:=x1+s*Dx; Mx[s]:=x;
            y:=C1*cos(w1*x)+C2*sin(w2*x);
            My[s]:=y;
            if y<yMin then yMin:=y;
            if y>yMax then yMax:=y;
           end;
          X0Y0(false); cleardevice;
          SystCoor;
         end;
    '9': for s:=0 to m do
          circle(X0+round(Mx[s]/Dx),
                 Y0-round(My[s]/Dy),3);
   end;
  until j1='0';
end;

begin
 c:=15;
 repeat
  PutA;
  Ou('Esc-exit,1-InpD,2-c,3-Grf,4-Ngl,5-Ngr,6-D0,7-Dm,
      8-GrBSpl');
  str(c,T10);       Ts:='c='+T10;
  str(Ngl,T10);     Ts:=Ts+', Ngl='+T10;
  str(Ngr,T10);     Ts:=Ts+', Ngr='+T10;
  str(D0:1:3,T10);  Ts:=Ts+', D0='+T10;
  str(Dm:1:3,T10);  Ts:=Ts+', Dm='+T10;
  Info; j:=ReadKey;
  case j of
   '1': InpD;
   '2': Oui('c',c);
   '3': begin
         SetColor(c); Mo[-1]:=L;
         for s:=0 to L do
          begin
           x:=x1+s*Dx;
           Mo[s]:=c1*cos(w1*x)+c2*sin(w2*x);
          end; {X0Y0(false); SystCoor;}
         Graphic(Mo,c);
        end;
   '4': Oui('Ngl',Ngl);
   '5': Oui('Ngr',Ngr);
```

```
    '6': Our('D0',D0);
    '7': Our('Dm',Dm);
    '8': begin
           Dx1:=(x2-x1)/m; Mx[-1]:=m;
           for s:=0 to m do
            begin
             x:=x1+s*Dx1;
             My[s]:=c1*cos(w1*x)+c2*sin(w2*x);
            end;
           BSpline(My,Ngl,Ngr,D0,Dm);
           Mo[-1]:=L; Dx:=(x2-x1)/L;
           for i:=0 to L do
             Mo[i]:=BSplint(i*m/L);
           Graphic(mo,c);
         end;
  end;
 until j=#27;
end.
```

```
Program Godogr;
  uses Crt,Graph,Serv;
  const
    R0:integer=50;
    R1:integer=10;
    Lst:integer=20;
    Wn:real=0;
    Wk:real=6.28;
    Nsc:integer=50;
    Nvr:integer=1;
    Tg:char='P';{'L'}
    Min:integer=5;
    Max:integer=15;
    Kw:real=1.5;
  var
    W,Dw,A1,A2:real;
    T20:string;

  procedure UrGod;
    begin
      case Nvr of
        1: begin
             x:=R0*cos(w);
             y:=R0*sin(w);
           end;
        2: begin
             x:=R0*cos(w);
             y:=R1*sin(w);
           end;
        3: begin
             A1:=R0+R1; A2:=A1/R1;
             x:=A1*cos(w)-Lst*cos(A2*w);
             y:=A1*sin(w)-Lst*sin(A2*w);
           end;
        4: begin
             A1:=R0-R1; A2:=A1/R1;
             x:=A1*cos(w)-Lst*cos(A2*w);
             y:=A1*sin(w)-Lst*sin(A2*w);
           end;
      end
    end;

  procedure SZ;
    begin
      UrGod;
```

```
    if Dx<>0 then S:=X0+round(x/Dx)
             else S:=X0;
    if Dy<>0 then Z:=Y0-round(Y/Dy)
             else Z:=Y0
  end;

procedure Scan;
  var S:integer;
  begin
    Dw:=(Wk-Wn)/Nsc;
    Xmin:=0; Xmax:=0; Ymin:=0; Ymax:=0;
    for S:=0 to Nsc do
      begin
        W:=Wn+S*Dw; UrGod;
        if X<Xmin then Xmin:=X;
        if X>Xmax then Xmax:=X;
        if Y<Ymin then Ymin:=Y;
        if Y>Ymax then Ymax:=Y;
      end;
    X0Y0(true);
    ClearDevice;
    SystCoor;
  end;

procedure PointGod;
  const Kw=1.2;
  var Mdx,Mdy,Md,Sp,Zp:integer;
      Wp:real;
  begin
    Sp:=S; Zp:=Z; Wp:=W;
    repeat
      W:=Wp+Dw; Sz;
      Mdx:=abs(S-Sp);
      Mdy:=abs(Z-Zp);
      if Mdx>Mdy then Md:=Mdx
                 else Md:=Mdy;
      if Md<>1 then
        if Md<1 then Dw:=Dw*Kw
                else Dw:=Dw/Md
    until (Md=1) or (w>Wk);
    if C>=0
      then PutPixel(S,Z,C)
      else PutPixel(S,Z,GetMaxColor-GetPixel(S,Z))
  end;
```

```
procedure InpPol(Id:char; var A:coef);
  var S,n:integer;
  begin
    Oui('n-step.pol. '+Id,n); A[-1]:=n;
    for S:=0 to n do
      begin
        str(s,T10);
        Our(Id+'['+T10+']',A[s])
      end
  end;

procedure LineGod;
  var Mdx,Mdy,Md,Sp,Zp:integer;
      Wp,R:real;
  begin
    Sp:=S; Zp:=Z; Wp:=W; R:=(Min+Max)/2;
    repeat
      W:=Wp+Dw; Sz;
      Mdx:=abs(S-Sp); Mdy:=abs(Z-Zp);
      if Mdx>Mdy then Md:=Mdx
                 else Md:=Mdy;
      if (Md>Max) or (Md<Min) then
        if Md<Min then Dw:=Dw*Kw
                  else Dw:=Dw*R/Md
    until ((Md<=Max)  and(Md>=Min)) or (W>Wk);
    if (Md<=Max) and (Md>=Min) then Line(Sp,Zp,S,Z);
    if C<0 then PutPixel(S,Z,GetMaxColor-GetPixel(S,Z))
  end;

procedure Godo;
  var J3:char;
  begin
    PutA; Ou('0-exit'); J3:='1';
    Dw:=(Wk-Wn)/Nsc; W:=Wn; Sz;
    if C>=0 then SetColor(C)
            else SetWriteMode(1);
    PutPixel(S,Z,C);
    repeat
      case Tg of
        'P':PointGod;
        'L':LineGod;
      end;
      Str(w,T20); Clear(450,0,GetMaxX-1,10);
      OutTextXY(450,0,'w='+T20);
      if KeyPressed then J3:=ReadKey;
```

```
      until (w>wk) or (J3='0');
      if C>=0 then SetColor(15)
              else SetWriteMode(0)
    end;

  procedure Nastr;
    begin
      repeat
        PutA;
        Ou('0-Exit,1-R0,2-R1,3-Lst,4-Min,5-Max,6-Kw,
            7-Tg');
        Str(R0,T10);          Ts:='R0='+T10;
        Str(R1,T10);          Ts:=Ts+', R1='+T10;
        Str(Lst,T10);         Ts:=Ts+', Lst='+T10;
        Str(Min,T10);         Ts:=Ts+', Min='+T10;
        Str(Max,T10);         Ts:=Ts+', Max='+T10;
        Str(Kw:1:3,T10);      Ts:=Ts+', Kw='+T10;
        Ts:=Ts+', Tg='+Tg;
        Info; J1:=ReadKey;
        case J1 of
          '1':Oui('R0',R0);
          '2':Oui('R1',R1);
          '3':Oui('Lst',Lst);
          '4':Oui('Min',Min);
          '5':Oui('Max',Max);
          '6':Our('Kw',Kw);
          '7':if Tg='P' then Tg:='L'
                        else Tg:='P';
        end;
      until J1='0';
    end;

begin
  repeat
    PutA;
    Ou('Esc-Exit,1-Nastr,2-Nvr,3-Wn,4-Wk,5-Nsc,6-Tg,
       7-C,8-Sc,9-Godo');
    Str(Nvr,T10);           Ts:='Nvr='+T10;
    Str(Wn:1:3,T10);        Ts:=Ts+', Wn='+T10;
    Str(Wk:1:3,T10);        Ts:=Ts+', Wk='+T10;
    Str(Nsc,T10);           Ts:=Ts+', Nsc='+T10;
    Ts:=Ts+', Tg='+Tg;
    Str(C,T10);             Ts:=Ts+', C='+T10;
    Info; J:=ReadKey;
    case J of
```

```
      '1':Nastr;
      '2':Oui('Nvr',Nvr);
      '3':Our('Wn',Wn);
      '4':Our('Wk',Wk);
      '5':Oui('Nsc',Nsc);
      '6':if Tg='P' then Tg:='L'
                    else Tg:='P';
      '7':Oui('C',C);
      '8':Scan;
      '9':Godo
    end;
  until J=#27;
  Closegraph;
end.
```

```
Program Izolines;
  Uses Crt, Graph, Serv, groms;
  const Hiz: real=0.2;
        Nvr: integer=1;
        God: boolean=true;
        Mi: integer=32000;
  var Nrr,Rad: real;
      Nr:integer;
      Mnr: array [-1..800] of integer;
      A: Coef;
      Mo: Coefl;

  function HorReal(A:coef; x:real):real;
   var s,n:integer;
        sum:real;
   begin
     n:=round(A[-1]); sum:=A[n];
     for s:=n-1 downto 0 do
       sum:=sum*x+a[s];
     HorReal:=sum
   end;

  procedure InpPol(Id:string; var A:Coef);
   var s,n:integer;
   begin
     Oui('n-degree of polynom '+Id,n); A[-1]:=n;
     for s:=n downto 0 do
       begin
         Str(s,t10);
         Our(Id+'['+t10+']',a[s]);
       end
   end;

  procedure HorComp (A:Coef;r,i:real; var Re,Im:real);
  var
      r1:real;
      n,s:integer;
  begin
   n:=round(A[-1]);
   Re:=A[n];
   Im:=0;
   for s:=n-1 downto 0 do
     begin
       R1:=Re*r-Im*i+A[s];
       Im:=Re*i+Im*r;
```

```
     Re:=R1
   end
end;

function Fxy(x,y:real):real;
var re,im:real;
 begin
   case Nvr of
    1: Fxy:=0.5*(1-cos(pi*x)+2*sqr(y));
    2: begin
        HorComp(a,x,y,re,im);
        Fxy:=sqrt(sqr(re)+sqr(im));
       end;
    3: Fxy:=sin(x)+cos(y);
    4: Fxy:=sin(sin(x))+cos(cos(y));
   end
 end;

 procedure GrPol;
 var dx,y:real;
     s:integer;
 begin
   dx:=(Xmax-Xmin)/L;
   mo[-1]:=L;
   Ymin:=0; Ymax:=0;
   for s:=0 to L do
   begin
     y:=HorReal(A,Xmin+s*dx);
     Mo[s]:=y;
     if y<Ymin then Ymin:=y;
     if y>Ymax then Ymax:=y
   end;
   X0Y0(false);
   ClearDevice;
   SystCoor;
   Graphic(Mo,15);
 end;

 procedure Sc;
 begin
  repeat
  PutA;
  Ou('0-exit, 1-Xmin, 2-Xmax, 3-Ymin, 4-Ymax, 5-God');
  str(Xmin:1:3,t10); ts:='Xmin='+t10;
  str(Xmax:1:3,t10); ts:=ts+', Xmax='+t10;
```

```
    str(Ymin:1:3,t10); ts:=ts+', Ymin='+t10;
    str(Ymax:1:3,t10); ts:=ts+', Ymax='+t10;
    str(rad:1:3,t10); ts:=ts+', rad='+t10;
    if god then ts:=ts+',god=true'
           else ts:=ts+',god=false';
    info; J1:=ReadKey;
    case J1 of
     '1': our('Xmin',Xmin);
     '2': our('Xmax',Xmax);
     '3': our('Ymin',Ymin);
     '4': our('Ymax',Ymax);
     '5': if god then god:=false
                 else god:=true
    end;
    until J1='0';
    X0Y0(god);
    ClearDevice;
    SystCoor
   end;

   function Radius(a:coef):real;
   var z,n:integer;
       b:real;
   begin
     n:=round(a[-1]);
     b:=abs(a[0]);
     for z:=1 to n-1 do
       if abs(a[z])>b then b:=abs(a[z]);
     Radius:=1+b/abs(a[n])
   end;

begin
 Xmin:=-2; Xmax:=2; Ymin:=-3; Ymax:=3;
 a[-1]:=3; a[0]:=0; a[1]:=5; a[2]:=4; a[3]:=1;
 Rad:=Radius(a);
 repeat
   PutA;
   Ou('Esc-exit,1-Nvr,2-Sc,3-InpPol,4-GrPol,5-L,6-H,
       7-Hiz,8-Izo,9-GromS');
   Str(Nvr,T10);     Ts:='Nvr='+T10;
   Str(L,T10);       Ts:=Ts+', L='+T10;
   Str(H,T10);       Ts:=Ts+', H='+T10;
   Str(Hiz:1:3,T10); Ts:=Ts+', Hiz='+T10;
   Info; J:=ReadKey;
   case j of
```

```
      '1': Oui('Nrv',Nvr);
      '2': Sc;
      '3': Inppol('a',a);
      '4': Grpol;
      '5': Oui('L',L);
      '6': Oui('h',h);
      '7': Our('hiz',hiz);
      '8': begin
             for z:=-1 to h0 do
               begin
                 Nrr:=Fxy(Xmin-dx,Ymin+z*dy)/hiz;
                 if abs(Nrr)>mi
                   then Mnr[z]:=Mi
                   else Mnr[z]:=trunc(Nrr)
               end;
             for s:=0 to L0 do
               begin
                 x:=Xmin+s*dx;
                 for z:=-1 to H0 do
                   begin
                     Nrr:=Fxy(x,Ymin+z*dy)/Hiz;
                     if abs(Nrr)>Mi
                       then Nr:=Mi
                       else Nr:=trunc(Nrr);
                     if z>-1 then
                      if(Nr<>Mnr[z-1]) or (Nr<>Mnr[z])
                       then
                        if abs(Nrr)<Mi then
                         if abs(Nrr)=0
                          then PutPixel (xu+s,yu+h0-z,15)
                          else PutPixel (xu+s,yu+h0-z,
                                  abs(round(Nrr) mod 16));
                     Mnr[z]:=Nr
                   end
               end
           end;
      '9':Service
    end;
 until J=#27;
 CloseGraph
end.
```

```
Program Relief;
uses Crt, Graph, Serv;
const
  Nvr:integer=1;
  Nx:integer=10;
  Ny:integer=10;
  U:integer=15;
  V:integer=20;
  W:integer=0;
  D:integer=500;
  Rx:integer=100;
  Ry:integer=120;
  Rz:integer=100;
var
  Xs,Ys,Zs,Kx,Ky,Kz,Zmin,Zmax:real;
  A:Coef;
  Mxe,Mye:array[0..20,0..20] of integer;
  Mo:array[0..20,0..20] of real;

  procedure HorComp(A:Coef; R,i:real; var Re,Im:real);
    var r1:real;
        n,s:integer;
    begin
      n:=round(A[-1]); Re:=A[n]; Im:=0;
      for s:=n-1 downto 0 do
        begin
          R1:=Re*R-Im*i+A[s];
          Im:=Re*i+Im*R;
          Re:=R1
        end
    end;

    function Fxy(x,y:real):real;
      var Re,Im:real;
      begin
        case Nvr of
          1: begin
               HorComp(A,x,y,Re,Im);
               Fxy:=sqrt(sqr(Re)+sqr(Im))
             end;
          2: Fxy:=sin(x)+cos(y);
        end
      end;

    procedure FormMo;
```

```
  var s,z:integer;
      x,y,dx,dy,f:real;
  begin
    dx:=(Xmax-Xmin)/Nx;
    dy:=(Ymax-Ymin)/Ny;
    f:=Fxy(Xmin,Ymin); Zmin:=f; Zmax:=f;
    for z:=0 to Ny do
      begin
        y:=Ymin+z*dy;
        for s :=0 to Nx do
          begin
            f:=Fxy(Xmin+s*dx,y);
            Mo[z,s]:=f;
            if f<Zmin then Zmin:=f;
            if f>Zmax then Zmax:=f;
          end;
      end;
    Xs:=(Xmin+Xmax)/2;
    Ys:=(Ymin+Ymax)/2;
    Zs:=(Zmin+Zmax)/2;
    Kx:=(Xmax-Xmin)/Rx;
    Ky:=(Ymax-Ymin)/Ry;
    Kz:=(Zmax-Zmin)/Rz;
  end;

procedure FormMe;
  var f,x,y,sinU,cosU,sinV,cosV,sinW,cosW,
      Cxx,Cxy,Cxz,Cyx,Cyy,Cyz,Czx,Czy,Czz,
      x3,y3,z3,Kp,dx1,dy1:real;
  begin
    dx1:=(Xmax-Xmin)/Nx;
    dy1:=(Ymax-Ymin)/Ny;
    x:=U*Pi/180; sinU:=sin(x); cosU:=cos(x);
    x:=V*Pi/180; sinV:=sin(x); cosV:=cos(x);
    x:=W*Pi/180; sinW:=sin(x); cosW:=cos(x);
    Cxx:=cosU*cosV; Cxy:=-sinU*cosV; Cxz:=sinV;
    Cyx:=sinU*cosW+cosU*sinV*sinW;
    Cyy:=cosU*cosW-sinU*sinV*sinW;
    Cyz:=-cosV*sinW;
    Czx:=sinU*sinW-cosU*sinV*cosW;
    Czy:=cosU*sinW+sinU*sinV*cosW;
    Czz:=cosV*cosW;
    for z:=0 to Ny do
      begin
        y:=(Ymin+z*dy1-Ys)/Ky;
```

```
            for s:=0 to Nx do
              begin
                x:=(Xmin+s*dx1-Xs)/Kx;
                f:=(Mo[z,s]-Zs)/Kz;
                x3:=Cxx*x+Cxy*y+Cxz*f;
                y3:=Cyx*x+Cyy*y+Cyz*f;
                z3:=Czx*x+Czy*y+Czz*f;
                Kp:=D/(D-x3);
                Mxe[z,s]:=round(y3*Kp);
                Mye[z,s]:=-round(z3*Kp/Hj);
              end
          end
      end;

    procedure Geom;
      begin
        repeat
          PutA;
Ou('0-Exit,1-Xmin,2-Xmax,3-Ymin,4-Ymax,5-Rx,6-Ry,7-Rz');
          str(Xmin:1:3,T10); Ts:='Xmin='+T10;
          str(Xmax:1:3,T10); Ts:=Ts+', Xmax='+T10;
          str(Ymin:1:3,T10); Ts:=Ts+', Ymin='+T10;
          str(Ymax:1:3,T10); Ts:=Ts+', Ymax='+T10;
          str(Rx,T10); Ts:=Ts+', Rx='+T10;
          str(Ry,T10); Ts:=Ts+', Ry='+T10;
          str(Rz,T10); Ts:=Ts+', Rz='+T10;
          Info; J1:=ReadKey;
          case J1 of
            '1': Our('Xmin',Xmin);
            '2': Our('Xmax',Xmax);
            '3': Our('Ymin',Ymin);
            '4': Our('Ymax',Ymax);
            '5': Oui('Rx',Rx);
            '6': Oui('Ry',Ry);
            '7': Oui('Rz',Rz);
          end
        until J1='0';
        FormMo;
      end;

    procedure Orient;
      begin
        repeat
          PutA;
          Ou('0-Exit, 1-U, 2-V, 3-W, 4-D');
```

```
        str(U,T10); Ts:='U='+T10;
        str(V,T10); Ts:=Ts+', V='+T10;
        str(W,T10); Ts:=Ts+', W='+T10;
        Info; J1:=ReadKey;
        case J1 of
          '1': Oui('U',U);
          '2': Oui('V',V);
          '3': Oui('W',W);
          '4': Oui('D',D);
        end;
      until J1='0';
      FormMe
    end;

  procedure FormRel;
    var z,s:integer;
    begin
      SetColor(C);
      for z:=0 to Ny do
        begin
          MoveTo(x0+Mxe[z,0],y0+Mye[z,0]);
          for s:=1 to Nx do
            LineTo(x0+Mxe[z,s],y0+Mye[z,s]);
        end;
      for s:=0 to Nx do
        begin
          MoveTo(x0+Mxe[0,s],y0+Mye[0,s]);
          for z:=1 to Ny do
            LineTo(x0+Mxe[z,s],y0+Mye[z,s]);
        end;
      SetColor(15);
    end;

  procedure InpPol(Id:char; var A:Coef);
    var n,s:integer;
    begin
      Oui('n-degre of polynom '+Id, n);
      A[-1]:=n;
      for s:=0 to n do
        begin
          Str(s,T10); Our(Id+'['+T10+']', A[s]);
        end
    end;

begin
```

```
  A[-1]:=4; A[0]:=1; A[1]:=0; A[2]:=0;
  A[3]:=0; A[4]:=1;
  Xmin:=-1.2; Xmax:=1.2; Ymin:=-1.2; Ymax:=1.2;
  x0:=200; y0:=200; C:=10;
  repeat
    PutA;
    Ou('Esc-Exit,1-Nvr,2-InpPol,3-Geom,4-Orient,5-x0,
        6-y0,7-C,8-Rel,9-Del');
    str(Nvr,T10); Ts:='Nvr='+T10;
    str(x0,T10);  Ts:=Ts+', x0='+T10;
    str(y0,T10);  Ts:=Ts+', y0='+T10;
    str(C,T10);   Ts:=Ts+', C='+T10;
    Info; J:=ReadKey;
    case J of
      '1': Oui('Nvr',Nvr);
      '2': InpPol('A',A);
      '3': Geom;
      '4': Orient;
      '5': Oui('x0',x0);
      '6': Oui('y0',y0);
      '7': Oui('C',C);
      '8': FormRel;
      '9': Clear(0,15,GetMaxX-1,GetMaxY-15);
    end;
  until J=#27;
  Closegraph;
end.
```

Литература

1. Айнберг В.Д. Основы программирования на алгоритмическом языке Алгол-60. Изд. 2-е, испр. и доп. –М.: Машиностроение, 1978. -152с.
2. Горинштейн А.М. Практика решения инженерных задач на ЭВМ. – М.: Радио и связь, 1984. -232с.
3. Григас Г. Начала программирования: Кн. для учащихся: пер. с лит. / Под. ред. Ю.А. Первина. –М.: Просвещения, 1987. -112с.
4. Карачун В.В., Кваско М.З., Кубрак Н.А. Прикладний аналіз і візуалізація характеристик динамічних систем: Навч. посібник. –К.: ІЗМН, 1999. -157с.
5. Кваско М.З., Кубрак А.І., Жученко А.І. Числові методи комп’ютерного моделювання автоматичних систем: Алгоритми і програми: Навч. посіб. –К.: ІВЦ “Видавництво <<Політехніка>>” 2003. -360с.
6. Кубрак А.И., Жученко А.И., Ярощук Л.Д. Численные методы и программы при исследовании автоматических систем: Учеб. пособие. К.: УМКВО, 1988. -207с.
7. Кубрак А.И., Ярощук Л.Д., Аверина Т.В. Програмування на Турбо Паскалі: Початковий курс: Навч. посібник. –К.: НМЦ ВО, 2000. -188с.
8. Мак-Кракен Д., Дорн У. Численные методы и программирование на Фортране. –М.: Мир, 1977. -584с.
9. Пильщиков В.Н. Сборник упражнений по языку Паскаль: Учеб. пособие для вузов. –М.: Наука, 1989. 160с.
10. Попов Б.О. Монцібович Є.Р. Розв’язування задач на машинах для інженерних розрахунків. –К.: Наукова думка, 1978. -156с.
11. Фаронов В.В. Турбо Паскаль 7.0. Начальный курс: Учеб пособие. – М.: Нолидж, 1997. -616с.

Содержание

Ввдение 3

1. Вычисление сумм, произведений, степеней 5

1.1. Возведение числа типа real в степень типа integer 5

1.2. Возведение вещественного числа в вещественную степень 9

1.3. Вычисление факториала 9

1.4. Вычисление биномиальных коэффициентов 11

1.5. Числа Фибоначчи 13

1.6. Приближенные вычисления сумм бесконечных рядов 14

1.7. Операции над комплексными числами 18

2. Операции с одномерными числовыми массивами 21

2.1. Формирование типов одномерных массивов 21

2.2. Анализ одномерного массива на наличие в нем элементов с заданными свойствами 26

2.3. Поиск минимального и (или) максимального элемента массива 29

2.4. Сдвиг элементов массива 33

2.5. Сортировка одномерного массива 36

2.6. Удаление из массива “дефектных” элементов 40

2.7. Перезапись массива в обратном порядке 40

2.8. Численное дифференцирование функций, заданных таблично 41

2.9. Численное интегрирование функций, заданных таблично 44

3. Работа с полиномами 45

3.1. Алгебраическое суммирование полиномов 45

3.2. Умножение полиномов 47

3.3. Деление полиномов 48

3.4. Разложение дробно-рациональной функции в ряд Маклорена 53

3.5. Алгебраическое сложение отрезков бесконечных степенных рядов 55

3.6. Умножение рядов 56

3.7. Деление рядов 57

3.8. Дифференцирование полиномов 58

3.9. Замена в полиноме р на $p_1 - L$ 58
4. Определение корней полиномов 61
4.1. Вычисления значения полинома при вещественном значении его аргумента 61
4.2. Схема Хорнера – Руффини отделения вещественного корня 64
4.3. Уточнение локализованного корня методом половинного деления 65
4.4. Вычисление значения корня при комплексном значении аргумента 68
4.5. Выделение круга, ограничивающего корни полинома 70
4.6. Сканирование заданной области комплексной плоскости с целью поиска корней полинома 71
4.7. Критерий Гурвица-Рауса 75
4.8. Определение степени устойчивости 77
4.9. Определение самого правого корня полинома 79
4.10. Определение корней полинома, степень которого не выше двух 81
4.11. Нахождение полного комплекта корней полинома 84
5. Работа с двумерными массивами 88
5.1. Локализация минимума в двумерном массиве 89
5.2. Перестановка строк и столбцов в двумерном массиве 90
5.3. Перезапись двумерного массива в одномерный 91
5.4. Решение систем линейных алгебраических уравнений методом Гаусса 91
5.5. Формирование интерполяционного полинома 100
5.6. Формирование сглаживающего полинома 104
5.7. Аппроксимация функциональной зависимости интерполяционным дробно-рациональным выражением 110
5.8. Аппроксимация функции сглаживающей дробно-рациональной структурой 113
6. Сплайны 116
6.1. Интерполяционные кубические сплайны 116
6.2. Сглаживающие кубические сплайны 124
6.3. Интерполяционные В-сплайны 133

6.4. Сравнение обычного интерполяционного кубического сплайна и интерполяционного B-сплайна 145
7. Визуализация результатов расчетов 149
7.1. Упрощенный графический интерфейс и формирование графиков 149
7.2 Оформление графиков и годографов 158
7.2.1. Процедура `Mark` (разметка шкал) 162
7.2.2. Процедура `Net` (нанесение координатной сетки) 162
7.2.3. Процедура `Num` (оцифровка шкал) 163
7.2.4. Процедура `Inscr` (нанесение надписей) 166
7.2.5. Процедура `Service` (сервисное обслуживание изображения) 169
7.3. Формирование годографов (файл Godogr) 169
7.3.1. Процедура `SZ` 171
7.3.2. Процедура `Scan` 172
7.3.3. Процедура `PointGod` 173
7.3.4. Процедура `LineGod` 177
7.3.5. Процедура `Godo` 178
7.4. Формирование изображения поверхности системой изолиний 181
7.5. Формирование аксонометрического изображения поверхности 187
Приложения 198
`Unit Serv` 198
`Unit Groms` 201
`Unit Polynom` 208
`Program KorPols` 213
`Program OSystUr` 220
`Program AprPol` 223
`Program BSpLGrs` 229
`Program DemSpl` 231
`Program DemBSpl` 235
`Program Godogr` 239

Program Izolines 244
Program Relief 248
Литература 253

Жученко Анатолий Иванович
Кубрак Наталия Анатольевна
Голинко Игорь Михайлович

БАЗОВЫЕ АЛГОРИТМЫ ЧИСЛЕННОГО АНАЛИЗА