РОССИЙСКАЯ АКАДЕМИЯ НАУК
ДАЛЬНЕВОСТОЧНОЕ ОТДЕЛЕНИЕ
СЕВЕРО-ВОСТОЧНЫЙ НАУЧНЫЙ ЦЕНТР
СЕВЕРО-ВОСТОЧНЫЙ КОМПЛЕКСНЫЙ НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ ИНСТИТУТ

В. Ю. БЕЛАШОВ, Н.М. ЧЕРНОВА

Магадан 1997

УДК 681.3.06

Белашов В.Ю., Чернова Н.М. **Эффективные алгоритмы и программы вычислительной математики**. Магадан: СВКНИИ ДВО РАН, 1997. 160 с.

В книге представлены результаты работы авторов по отбору наиболее эффективных (оптимальных) алгоритмов, реализующих методы как традиционных, так и практически не встречающихся в монографической и справочной литературе разделов вычислительной математики, например вычислительные методы в теории чисел, комбинаторике, теории спецфункций, теории спектральных преобразований и т.п. Каждый тематический раздел включает краткое введение в теорию с постулированием основных положений и серию оптимальных алгоритмов, реализующих тот или иной метод, с подробными комментариями и примерами тестовых расчетов. В книге более 200 текстов процедур, написанных на языке *TURBO PASCAL*.

Для специалистов, работа которых связана с решением задач на ЭВМ, аспирантов и студентов вузов по рекомендации Министерства общего и профессионального образования Российской Федерации.

Табл. 90. Ил. 25. Библиогр.: 97 назв.

Ответственный редактор
доктор физико-математических наук В.Н.Доровский

Утверждено к печати Ученым советом СВКНИИ ДВО РАН

Belashov V.Yu., Chernova N.M. **Effective algorithms and programs of applied mathematics**. Magadan: NEISRI FEB RAS, 1997. 160 ð.

This book is a result of selection of the most effective (optimum) algorithms realizing the methods of traditional topics and nontraditional ones which are not found in the monograph and reference books (for example, calculational methods in the number theory, combinatorics, the theory of mathematical functions and theory of orthogonal transforms etc.). Each subject section includes brief summary of the theory with postulating of main principles and a number of optimum algorithms being provided with the detail comments and the examples of their testing. The book includes more than 200 procedures written in Turbo Pascal.

ISBN 5-7442-1013

В настоящее время существует обширная литература, в которой рассматриваются отдельные разделы вычислительной математики (численное интегрирование, в том числе интегрирование как обыкновенных дифференциальных уравнений, так и уравнений в частных производных, численное дифференцирование, методы решения задач линейной алгебры, обработка данных эксперимента и пр.). Однако все известные издания посвящены либо изложению теоретических аспектов соответствующих разделов вычислительной математики, т.е. в них отсутствуют указания на конкретные приложения и практически не включены примеры конкретных алгоритмов и текстов программ (не говоря уже о примерах тестовых расчетов), либо представляют собой сборники чисто алгоритмов и программ без какого бы то ни было, пусть элементарного, введения в теорию используемых методов вычислений. Все это создает искусственный барьер между специалистами в области вычислительной математики и пользователями не математиками, которым по роду своей деятельности в той или иной сфере (естественные и экономические науки, техника, система образования и т.п.) приходится решать задачи с помощью соответствующих методов вычислений.

К сожалению, в литературе часто не уделяется внимание уже разработанным и признанным весьма эффективными алгоритмам, которые широко использовались для решения прикладных задач на ЭВМ серий ЕС и БЭСМ.

Отметим также, что практически отсутствует специальная литература (включающая как теоретические аспекты методов, так и примеры программ), посвященная наиболее эффективным и оптимальным с точки зрения затрат машинного времени и памяти ЭВМ алгоритмам и доступная широкому кругу пользователей. Справочники или слишком элементарны (написаны на уровне первых версий языка Бейсик и программируемых микрокалькуляторов) или содержат описание методов и примеры программ лишь по отдельным разделам вычислительной математики. Этими же недостатками страдает как переводная, так и оригинальная иностранная литература, не говоря уже о разработках иностранных фирм (к сожалению, малодоступных широкому кругу отечественных пользователей).

Авторы сделали попытку в той или иной мере устранить отмеченные недостатки путем систематизированного изложения алгоритмов с краткими предварительными сведениями по теории используемых методов по всем, включенным в издание, разделам вычислительной математики.

Наряду с наиболее эффективными алгоритмами, разработанными в последнее время, книга содержит оригинальные полученные авторами результаты. Что касается алгоритмов, разработанных другими специалистами, то при подготовке настоящего издания их тщательно отбирали, учитывая эффективность, оптимальность (оценка проводилась по скорости получения результатов и затратам ресурсов ЭВМ). Мы повторили тестирование на персональных компьютерах типа IBM PC и выполнили решение контрольных примеров для широкого диапазона входных параметров.

Основное содержание книги составляют лекционные курсы “Численные методы”, “Вычислительная математика” и “Вычислительная физика”, которые авторы читают в течение последних пяти лет в Международном педагогическом университете г. Магадана для студентов физико-математического факультета и аспирантов соответствующих специальностей. О развитии методов вы-

числительной математики и построения алгоритмов численного решения прикладных задач мы неоднократно докладывали на различных отечественных и международных конференциях и симпозиумах.

Материал разбит на тематические разделы, соответствующие традиционно выделяемым областям с включением таких, практически не встречающихся в изданиях подобного плана разделов, как, например, вычислительные методы в теории чисел, комбинаторике, теорий спецфункций и спектральных преобразований и т.п. Каждый тематический раздел содержит краткое введение в теорию с постулированием основных положений (поэтому в книге отсутствует такой раздел, как общее введение) и серию оптимальных алгоритмов с их программной реализацией на языке *TURBO PASCAL* того или иного метода с подробными комментариями и примерами тестовых расчетов.

Хотелось бы отметить, что мы старались построить книгу так, чтобы пользователь, работающий в конкретной предметной области и не являющийся специалистом непосредственно в вычислительной математике, мог без особого труда выбрать наиболее эффективный метод численного решения своей задачи. И, наконец, авторы надеются, что методика подбора и изложения материала позволит специалистам использовать это издание в качестве настольной книги-справочника по эффективным алгоритмам и программам вычислительной математики.

Главы 1-3 написаны авторами совместно, главы 4, 6-8 - Н.М.Черновой, предисловие, главы 5, 9 и заключение - В.Ю.Белашовым. Работа по отбору, написанию и тестированию алгоритмов проводилась соответственно.

Авторы выражают свою признательность Л.И.Измайлову за поддержку, без которой публикация книги вряд ли была бы возможной, а также М.А.Трумпе за техническую помощь в подготовке графического материала.

1.

§ 1. ЭЛЕМЕНТАРНАЯ АЛГЕБРА

В главе представлены алгоритмы элементарной алгебры, которые обычно не включаются в библиотеки математического обеспечения ЭВМ, хотя довольно часто оказываются необходимыми при решении соответствующих задач.

1.1. КОМПЛЕКСНАЯ АРИФМЕТИКА. ИЗВЛЕЧЕНИЕ КОРНЕЙ *n*-й СТЕПЕНИ ИЗ КОМПЛЕКСНОГО ЧИСЛА

При извлечении корней n-й степени из комплексного числа $z = x + iy = re^{i\varphi}$ ($i^2 = -1$), где $x = \mathrm{Re}\, z$ - действительная часть; $y = \mathrm{Im}\, z$ - мнимая часть; $y = |z| = \sqrt{x^2 + y^2}$ - модуль; $\varphi = \arg z$ - главное значение аргумента комплексного числа, используется формула Муавра:

$$\sqrt[n]{z} = \sqrt[n]{r} \cdot \exp\left(\frac{\varphi + 2\pi k}{n} \cdot i\right) =$$

$$= \sqrt[n]{r} \cdot \left(\cos(\varphi + 2k\pi)/n + i \cdot \sin(\varphi + 2k\pi)/n\right),$$

$$k = 0,\ 1,\ 2,\ \ldots,\ n-1\ .$$

При этом алгоритм предполагает вычисление n корней уравнения $z^n = x + iy$. Область применения его очевидна - это задачи, связанные с комплексной арифметикой.

Формальные параметры процедуры. *Входные*: *x, y* (тип ***real***) - действительная и мнимая части комплексного числа; *n* (тип ***integer***) - показатель степени. *Выходные*: одномерные массивы *Re*[1:*n*], *Im*[1:*n*], в которых запоминаются соответственно действительные и мнимые части комплексных корней.

Алгоритм реализован в следующей процедуре:

```
PROCEDURE CROOT (X,Y: REAL; N: INTEGER;
                 VAR RE, IM : MAS1);
CONST PI = 3.14159265;
VAR A, M, R, T : REAL; I : INTEGER;
BEGIN    M:=1/N;
         R := EXP((M/2) *LN (X*X+Y*Y));
         IF X=0 THEN T := SIGN(Y)*PI/2;
         IF X>0 THEN T := ARCTAN(Y/X) ELSE
                T := ARCTAN(Y/X)+PI;
         T := M*T;              A:=2*PI*M;
         FOR I:=1 TO N DO
         BEGIN
            RE[I]:=S*COS(T);
            IM[I]:=S*SIN(T);
            T:=T+A;
         END;
END { *** CROOT *** };
```

Точность вычислений по данной подпрограмме определяется точностью представления действительных чисел в используемой ЭВМ и точностью вычисления функций sin и cos в соответствующих библиотеках подпрограмм.

Подпрограмма тестировалась на примерах извлечения корней 3-й и 5-й степени из комплексных чисел $z_1 = 8 + i6$ и $z_2 = -8 + i6$ (выбор чисел определялся требованием сравнимости результатов с вычислениями по аналогичной процедуре, описанной Агеевым и др. (1976)). Результаты расчетов представлены в табл. 1.1.

Данные результаты совпадают с результатами работ Агеева и ручного счета с точностью 10^{-6}.

Замечание. В работе Нестора [Nestor, 1961] в целях экономии машинного времени при $n > 3$ предлагается вместо обращения к стандартным функциям sin и cos в цикле использовать рекуррентные формулы (1.1):

$$\left.\begin{aligned}\sin(n+1)\theta &= \sin(n\theta)\cdot\cos\theta + \cos(n\theta)\cdot\sin\theta;\\ \cos(n+1)\theta &= \cos(n\theta)\cdot\cos\theta - \sin(n\theta)\cdot\sin\theta,\end{aligned}\right\} \quad (1.1)$$

что, однако, требует некоторых дополнительных затрат памяти ЭВМ [Агеев и др., 1976]. По оценкам специалистов [Nestor, 1961] такая процедура оказывается весьма эффективной в задачах, связанных с разложением в ряд Фурье, как это было предложено в работе [Goerzel, 1961].

Таблица 1.1

Формула расчета	
$x^n = 8 + i\,6$	$x^n = -8 + i\,6$
$n = 3$	
$x = 2.1050612 + i\,0.4585914$	$x = 1.4496824 + i\,1.5937408$
$x = -1.4496824 + i\,1.5937408$	$x = -2.1050612 + i\,0.4585914$
$x = -0.6553788 - i\,2.0523322$	$x = 0.6553788 - i\,2.0523322$
$n = 5$	
$x = 1.5717854 + i\,0.2034135$	$x = 1.3911646 + i\,0.7593073$
$x = 0.2922507 + i\,1.5577150$	$x = -0.2922507 + i\,1.5577150$
$x = -1.3911645 + i\,0.7593073$	$x = -1.5717854 + i\,0.2034135$
$x = -1.1520377 - i\,1.0884372$	$x = -0.6791661 - i\,1.4319985$
$x = 0.6791661 - i\,1.4319985$	$x = 1.1520377 - i\,1.0884372$

Однако наши расчеты, выполненные на компьютере PC/AT-286 с частотой 12 МГц, показали, что сколько-нибудь значительного выигрыша во времени использование рекуррентных соотношений (1.1) практически не дает.

1.2. КОМБИНАТОРИКА. БИНОМ НЬЮТОНА И РОДСТВЕННЫЕ ФОРМУЛЫ

Число перестановок из n элементов определяется формулой

$$P_n = n! \; ; \qquad (1.2)$$

число размещений из n элементов по m элементов

$$A_n^m = \frac{n!}{(n-m)!} \; ; \qquad (1.3)$$

число сочетаний из n элементов по m элементов

$$C_n^m = \frac{n!}{(n-m)!\,m!} \; . \qquad (1.4)$$

Число перестановок с повторениями (*кортежами*) *состава* (*спецификации*) (k_1, k_2, ..., k_r) определяется формулой

$$C_n(k_1, k_2, \ldots, k_r) = \frac{n!}{k_1!\,k_2!\ldots k_r!}, n = k_1 + k_2 + \ldots + k_r \; ; \qquad (1.5)$$

число сочетаний из n элементов по m с *повторениями*

$$f_n^m = C_{m+n-1}^{n-1} = C_{m+n-1}^m \; . \qquad (1.6)$$

При численных расчетах также можно использовать рекуррентную формулу

$$f_n^m = f_{n-1}^m + f_n^{m-1} \; . \qquad (1.7)$$

В формулах (1.2) - (1.4) $n!$, $m!$, $k!$ и $(n\text{-}m)!$ - факториалы, которые легко могут быть вычислены из определения

$$0! = 1, \; n! = \prod_{k=1}^{n} k = 1 \cdot 2 \cdot 3 \times \ldots \times (n-1) \cdot n, \; n > 0. \qquad (1.8)$$

Однако для больших n использование выражения (1.8) в связи со значительным количеством операций умножения чисел с плавающей точкой приводит к неоправданным затратам процессорного времени. В таких случаях для приближенного вычисления факториалов удобно использовать *асимптотическое разложение Стирлинга*, в результате получаем ***формулу Стирлинга*** для $n!$:

$$n! \approx n^n e^{-n} \sqrt{2n\pi} \;\; \text{при} \; n \to \infty \; . \qquad (1.9)$$

Относительная ошибка формулы Стирлинга, как это видно из выражения (1.9), убывает с возрастанием n. Для достаточно больших $n \to \infty$, однако, более точного результата можно добиться, если использовать специальную формулу [Корн, Корн, 1978]:

$$n! \approx n^n \sqrt{2n\pi} \cdot \exp\left(-n + \frac{1}{12n} - \frac{1}{360n^2} + \ldots\right), \qquad (1.10)$$

поскольку $n! \approx n^n e^{-n}$; $n! \approx n^n e^{-n} \sqrt{2n\pi}$ при $n \to \infty$. Отмеченные особенности учтены нами в процедуре вычисления факториала, которая может быть с успехом использована для нахождения P_n, A_n^m, C_n^m, C_n, f_n^m в соответствии с формулами (1.2) - (1.6).

Формальные параметры процедуры. *Входные*: n (тип ***integer***) - натуральное число, факториал которого требуется вычислить; k (тип ***integer***) - ключ, значение которого отвечает выбору одной из формул (1.8) - (1.10): $k < 0$ - формула (1.8), $k = 0$ - формула (1.9), $k > 0$ - формула (1.10). *Выходной*: идентификатор процедуры-функции *fct* (тип ***real***)- значение факториала.

```
{*** FCT ***}
FUNCTION FCT (N, K : INTEGER) : REAL;
CONST PI = 3.14159265;
VAR S1, S2, S3 : REAL; I : INTEGER;
BEGIN
  IF N=0 THEN
  BEGIN
      FCT:=1;
      EXIT;
  END;
  IF K < 0 THEN
  BEGIN
      FCT:=1;
      FOR I:=1 TO N DO
          FCT:=FCT*I;
      EXIT;
  END;
  S1:=SQRT(2*PI*N);
  IF K=0 THEN
  BEGIN
      S3:=0;
      S2:=0
  END
  ELSE
  BEGIN
      S3:=1/(360*N*N*N);
      S2:=1/(12*N)
  END;
  FCT:=EXP(N*LN(N))*S1*EXP(-N+S2-S3);
END .
```

Оценка точности вычислений проводилась при тестировании программы для n = 5, 10, 15, 20, 30; k = -1, 0, 1. Результаты, полученные при n = 5, 10, 15, и соответствующие им величины относительной погрешности представлены в табл.1.2.

Результаты тестирования показывают, что с ростом n погрешность вычисления $n!$ уменьшается, причем во всех случаях формула (1.10) дает более точный результат. Однако следует отметить, что при дальнейшем увеличении n погрешность формулы (1.10) остается практически на том же уровне (~1.0 e - 5 %) и при необходимости получить более точный результат требуется удержать следующий член в разложении (1.10). Дальнейший рост n приводит к появлению ошибок округления, связанных с конечностью разрядной сетки ЭВМ.

Таблица 1.2

n	Формула (1.8) (точное значение)	Формула (1.9)	Формула (1.10)
5	120	118.019172668 (1.65 %)	119.9999771 (1.9 *e* - 5 %)
10	3628800	3598695.75 (0.82 %)	3628800 (0 %)
15	1307674279936	1300430716928 (0.55 %)	1307674411008 (1.0 *e* - 5 %)

Процедура FCT может быть эффективно использована при вычислении бинома Ньютона:

$$(a+b)^n = \sum_{k=0}^{n} C_n^k a^{n=k} b^k \qquad (1.11)$$

и величины полинома

$$(a_1 + a_2 + \ldots + a_r)^n = \sum_{k_1+k_2+\ldots+k_r=n} C_n(k_1, k_2, \ldots, k_r) \cdot a_1^{k_1} \cdot a_2^{k_2} \cdot \ldots \cdot a_r^{k_r}, \qquad (1.12)$$

где суммирование проводится по всем наборам неотрицательных целых чисел (k_1 ,k_2 ,....,k_r), для которых $k_1 + k_2 + \ldots + k_r = n$. При этом $C_n(k_1, k_2, \ldots, k_r)$, вычисляемые по формуле (1.5), называются *полиномиальными коэффициентами.* Формула для определения количества полиномиальных коэффициентов имеет вид:

$$\sum_{k_1+k_2+\ldots+k_r=n} C_n(k_1, k_2, \ldots, k_r) = r^n \quad .$$

Рассмотрим процедуру-функцию вычисления бинома Ньютона, в которой используется функция FCT в соответствии с формулой (1.11).

Формальные параметры процедуры. *Входные*: *a, b* (тип ***real***)- значения слагаемых *a* и *b* в выражении (1.11); *n* (тип ***integer***) - показатель степени бинома; *k* (тип ***integer***) - ключ, определяющий выбор алгоритма вычисления факториала (см. описание параметров процедуры-функции FCT). *Выходной:* функция *binn* (тип ***real***) - значение бинома Ньютона.

```
FUNCTION BINN (VAR A, B : REAL;
               VAR N, K : INTEGER;): REAL;
VAR S,F1,F2,F3 : REAL; L,M : INTEGER;
BEGIN
  BIN:=0;
  F1:=FCT(N,K);
  FOR L:=0 TO N DO
  BEGIN
     F2:=FCT(L,K);
     M :=N-L;
     F3:=FCT(M,K);
     S :=F1/(F2*F3);
     BIN:=BIN+S*EXP(A*LN(M)+B*LN(L));
  END;
END; {*** B I N N  ***} .
```

Оценка точности вычислений бинома проводилась при тестировании программы для *n* = 5, 10, 15, 20, 30; *k* = - 1 (данное значение *k* соответствует точному вычислению факториала в процедуре-функции FCT) и различных параметров *a* и *b*. Результаты, полученные при *n* = 5, 7, 10, 15; *a* = =1.5; *b* = 2.5, -2.5, и соответствующие им величины относительной погрешности представлены в табл. 1.3.

Таблица 1.3

Параметр			Значение бинома при использовании формулы	
n	*a*	*b*	(1.8)	(1.10)
5	1.5	2.5	1024 (+ 0%)	1024.1940 (+1.89e-2%)
		- 2.5	— 1 (+ 0%)	— 0.8770 (+11.3%)
7	1.5	2.5	16384 (+ 0%)	16385.5449 (+9.4e-3%)
		- 2.5	— 1 (+ 0%)	0.125 (+112.6%)
10	1.5	2.5	1048576 (+ 0%)	1048610 (+3.2e-3%)
		- 2.5	1.00098 (+9.766e-2%)	—
15	1.5	2.5	21073741696 (+1.2e-5%)	1073747328 (+5.1e-4%)
		- 2.5	—	—

Из таблицы видно, что при $b > 0$ с увеличением *n* точность вычислений растет и практически ограничена только ошибками округления, для отрицательных же *b* уже при $n>10$ в случае использования формулы (1.8) и при $n \sim 5$ в случае использования формулы (1.10) для факториала погрешность резко возрастает, что связано с вычитанием больших чисел в правой части равенства (1.11). Поэтому при $b < 0$ рекомендуется использовать элементарную формулу $(a + b)^n$, на современных быстродействующих ЭВМ это даст некоторый выигрыш во времени счета.

Заметим, что совершенно аналогично может быть построена процедура вычисления величины полинома (1.12), поэтому она здесь не приводится.

Когда известно некоторое сочетание из *n* по *m*, тогда вычисление следующего по порядку сочетания может быть эффективно осуществлено с помощью процедуры генератора сочетаний [Агеев и др., 1976], которая образует следующее по порядку сочетание из *n* целых чисел по *m*, если заданы *n, m* и предшествующее сочетание.

Формальные параметры процедуры. *Входные: n, m* (тип ***integer***); вектор ***j*** [1:*m*] (тип ***integer***) - предшествующее сочетание. *Выходной:* тот же вектор ***j***, в котором целые числа меняются от 0 до *n* - 1 и всегда при входе и выходе из процедуры составляют монотонную строго возрастающую последовательность. Если входной вектор ***j*** состоит из нулей, то в качестве первого сочетания будет получено (*n* - *m, ..., n* - 1). Такое же сочетание получится и

после сочетания (0, 1,..., k - 1), являющегося последним значением вектора ***j*** в этом цикле.

```
PROCEDURE COMB (N, M : INTEGER; VAR J : MAS1);
VAR A, B, L : INTEGER;
BEGIN
  FOR B:= 1 TO M DO
  IF J[B]>B THEN
  BEGIN
     A:=J[B]-B-1;
     FOR L:=1 TO B DO     J[L]:=L+A;
     EXIT;
  END;
  B:=N-M-1;
  FOR L:=1 TO K DO
     J[L]:=B+L;
END; {*** C O M B  ***}.
```

Данный алгоритм был переведен авторами с языка *ALGOL* на язык *PASCAL* и проверен на машине IBM PC/AT-286 для тех же значений входных параметров, что и в работе Агеева и др. (1976), n = 4, k = 3. В качестве начального вектора ***j*** был выбран вектор (0,0,0). В результате последовательных обращений к процедуре получены следующие сочетания: (1,2,3), (0,2,3), (0,1,3), (0,1,2), что полностью соответствует результатам тестирования *ALGOL*-программы, приведенными в работе Агеева и др. (1976).

1.3. ВЫЧИСЛЕНИЕ СИМВОЛА ЯКОБИ $\left(\frac{a}{P}\right)$

Символ Якоби $\left(\frac{a}{P}\right)$ - функция, определяемая для всех целых a, взаимнопростых, с заданным нечетным целым числом $P > 1$. Так, если $P = p_1 p_2 ... p_r$ - разложение числа P на простые сомножители не обязательно различные, то

$$\left(\frac{a}{P}\right) = \left(\frac{a}{p_1}\right) \times \left(\frac{a}{p_2}\right) \times \ldots \times \left(\frac{a}{p_r}\right),$$

где $\left(\frac{a}{p_i}\right)$ - *символы Лежандра* [Виноградов, 1953], т.е. арифметические функции чисел a и p_i, определенные для простых нечетных p_i и целых a, не делящихся на P, причем $\left(\frac{a}{p_i}\right) = 1$, если сравнение $x^2 \equiv a (\mathrm{mod}\ p_i)$ разрешимо, а в противном случае $\left(\frac{a}{p_i}\right) = -1$. Часто символ Лежандра, а следовательно, и символ Якоби, который является его обобщением, доопределяют для чисел a, делящихся на p_i (для символа Якоби соответственно на P), полагая, что в этом случае $\left(\frac{a}{p_i}\right) = 0$. Символ Якоби обладает свойствами, аналогичными свойствам символа Лежандра, а именно:

1) если $a \equiv b (\mathrm{mod}\ P)$, то $\left(\frac{a}{P}\right) = \left(\frac{b}{P}\right)$;

2) $\left(\frac{1}{P}\right) = 1$;

3) $\left(\frac{a}{P}\right) \equiv a^{(P-1)/2} (\mathrm{mod}\ P)$;

4) $\left(\frac{ab}{P}\right) = \left(\frac{a}{P}\right) \cdot \left(\frac{b}{P}\right)$;

5) $\left(\frac{P}{Q}\right) \cdot \left(\frac{Q}{P}\right) = (-1)^{(P-1)/2 \cdot (Q-1)/2}$,

где P, Q - положительные нечетные взаимно простые числа (*квадратичный закон взаимности*, который впервые доказан для символа Лежандра Гауссом в 1876 г.);

6) $\left(\frac{-1}{P}\right) = (-1)^{(P-1)/2}$;

7) $\left(\frac{2}{P}\right) = (-1)^{(P \times P-1)/8}$.

Перечисленные свойства позволяют легко вычислять символ Якоби, не прибегая к решению сравнений. Заметим, что при фиксированном P символ Якоби является действительным характером мультипликативной группы классов вычетов по модулю P.

Процедура SYJAC, построенная по алгоритму, учитывающему приведенные свойства, вычисляет значение символа Якоби $\left(\frac{a}{P}\right)$ по квадратичному закону взаимности. При этом полагается следующее (табл. 1.4):

Таблица 1.4

Условие	Значение символа Якоби ...
P - четное	Не определено
P - нечетное	2
P - простое и a является квадратичным невычетом числа m	- 1
Если a и P имеют нетривиальный общий множитель	0
Остальные случаи	1

Формальные параметры процедуры. *Входные*: a, p (тип ***integer***). *Выходной: r* (тип ***integer***) - значение символа Якоби.

```
PROCEDURE SYJAC (A, P : INTEGER; VAR R : INTEGER);
VAR K : INTEGER; Z, Q : BOOLEAN;
LABEL : START, CYCLE;
FUNCTION PARITY (X:INTEGER): BOOLEAN;
  { *** ???????? ??????? PARITY TRUE, ???? ? -
       ??????, ? FALSE - ? ????????? ??????    *** }
BEGIN
  IF (X DIV 2 * 2) = X THEN
     PARITY := TRUE
  ELSE
     PARITY := FALSE;
END;{***P A R I T Y ***}
```

```
BEGIN
START:
  IF NOT PARITY (P) THEN
  BEGIN
     R := 2:
     EXIT;
  END;
  Z := TRUE;
CYCLE:
  REPEAT
     A:=A - A DIV P * P;
     Q := FALSE;
     IF A > 1 THEN
     BEGIN
        WHILE NOT PARITY (A) DO
        BEGIN
           Q:= NOT Q;
           A:=A DIV 2;
        END;
        IF Q AND PARITY ((SQR(P)-1) DIV 8) THEN
           Z := NOT Z;
        IF A ≠ 1 THEN
        BEGIN
           IF PARITY((P-1)*(A-1) DIV 4) THEN
              Z:= NOT Z;
           K:=P;
           P:=A;
           A:=K;
        END;
  UNTIL A <= 1;
  IF A=0 THEN
     R:= 0
  ELSE
     IF Z THEN
        R := 1
     ELSE
        R:= -1;
END.
```

Приведенный алгоритм был переведен авторами с языка *ALGOL* на язык *PASCAL* и проверен на машине IBM PC/AT-286 для тех же значений входных параметров, что и в работе Агеева (1976). В результате тестирования было получено:

$\left(\frac{5}{19}\right) = 1$ ($a = 5$ - квадратичный вычет);

$\left(\frac{3}{31}\right) = -1$ ($a = 3$ - квадратичный невычет);

$\left(\frac{7}{12}\right) = 2$ ($p = 12$ - четное), что полностью соответствует результатам тестирования программы на языке *ALGOL*, приведенным в работе Агеева и др. (1976).

1.4. РАЗЛОЖЕНИЕ МНОГОЧЛЕНА НА МНОЖИТЕЛИ

Многочлен (*целая рациональная функция*) относительно $x_1, x_2, \ldots, x_m$ есть сумма конечного числа членов вида $ax_1^{k_1} \cdot x_2^{k_2} \times \ldots \times x_m^{k_m}$, где каждое k_j - целое неотрицательное число. Наибольшее значение суммы $k_1 + k_2 + \ldots + k_m$, встречающееся в каком-либо из членов, называется *степенью многочлена*. В простейшем случае, когда $x_1 = x_2 = \ldots = x_m = x$ и наибольшее значение суммы степеней $\sum_j^m k_j = n$, многочлен принимает вид

$$P_n \equiv \sum_{i=0}^{n} a_{n-i} \cdot x^i = a_0 x^n + a_2 x^{n-1} + \ldots + a_{n-1} x + a_n . \quad (1.13)$$

Выражение (1.13) называется *канонической формой* представления целой рациональной функции. Если многочлен (1.13) при $n > 1$ может быть представлен в виде произведения многочленов низших степеней, он называется *приводимым*. Возможность разложения на множители многочленов степени $n > 1$ зависит от области определения коэффициентов. В частном случае, когда область определения коэффициентов есть множество действительных чисел, любая целая рациональная функция n-й степени может быть разложена на множители первой и второй степени (*основная теорема алгебры* [Бронштейн, Семендяев, 1981]). Приведенная нами процедура FACTORS позволяет найти все рациональные линейные множители $(u_i x + v_i)$, $1 < i < r$ многочлена (1.13) в частном случае, когда область определения коэффициентов сужена до множества целых чисел. При этом находится также наибольший общий делитель c коэффициентов a_i. Суть метода состоит в том, что находятся все делители p коэффициента a_0 и все делители q коэффициента a_n, (причем $1 < p < |a_0|$ и $1 < q < |a_n|$). Далее составляются все возможные пары из найденных p и q и проверяется, не является ли двучлен (px - q) множителем многочлена.

Формальные параметры процедуры. *Входные: n* (тип ***integer***)- степень многочлена; *a*[0:*n*] (тип ***integer***) - массив коэффициентов многочлена. *Выходные: u*[1:*r*], *v*[1:*r*] (тип ***integer***) - массивы коэффициентов соответственно при первых и нулевых степенях *x* в рациональных линейных множителях*; r* (тип ***integer***) - количество линейных множителей в разложении многочлена; *c* (тип ***integer***) - наибольший общий делитель коэффициентов a_i многочлена.

```
PROCEDURE FACTORS (N : INTEGER; A : MAS1;
          VAR U,V : MAS1; VAR R,C : INTEGER);
LABEL ZERO;
VAR I, F, G, P, Q : INTEGER;
BEGIN
  R:=0;
  C:=1;
```

```
ZERO:
  IF A[N]=0 THEN
  BEGIN
      N:=N-1;
      R:=R+1;
      U[R]:=1;
      V[R]:=0;
      GOTO ZERO;
  END;                          { *** ZERO***}
  FOR P:=1 TO ABS(A[0]) DO
      IF (A[0] DIV P * P) = A[0] THEN
      BEGIN
{ *** ??????? ???????? P, ???????????
      ????????? ???????????? A0    ***}
          FOR Q:=1 TO ABS(A[N]) DO
          BEGIN
{ *** ?????? Q ≠ 0, ?????????? ?????????? AN ***}
              IF Q<>1 THEN
              REPEAT
                  IF (A[N] DIV Q * Q)=A[N] THEN
                  BEGIN
                      FOR I:=0 TO N DO
                          IF (A[I] DIV Q * Q) = A[I] THEN
                              FOR I:=0 TO N DO
                                  A[I]:=A[I] DIV Q;
                      C := C * Q;
                   END
               UNTIL (A[N] DIV Q * Q)<>A[N];
              REPEAT
                  F:=A[0];
                  G:=1;
                  FOR I:=1 TO N DO
                  BEGIN
                      G:=G*P;
                      F:=F*Q+G*A[I]
                  END;
                  IF F=0 THEN INC(R);
                      U[R]:=P;
                  V[R]:=Q;
                  FOR I:=0 TO N DO
                  BEGIN
                      A[I]:=F:=(A[I]+F) DIV P;
                      F:=F*Q;
                  END;
              N:=N-1;
              IF N=0 THEN
              BEGIN
                  C:=C*A[0];
                  EXIT
              END
              ELSE
                  Q:=-Q;
          UNTIL (N<>0) OR (Q<0);
      END;
  IF N=0 THEN C:=C*A[0];
END.
```

Данный алгоритм был переведен авторами с языка *ALGOL* стереотипного переиздания [Агеев и др., 1976] сокращенной и ординарно переработанной процедуры, опубликованной в работе [Relph, 1962], на язык *PASCAL* и проверен на машине IBM PC/AT-286 для многочленов, что из работы Агеева и др. (1976). При этом были получены следующие разложения, полностью совпадающие с результатами Агеева и др. (1976):

$$P_3 \equiv \quad 3x^3 - 29\,x^2 + 78x - 40 = 1*(x - 4)\,(x - 5)\,(3x - 2);$$
$$P_3 \equiv \; x^3 - 6x^2 + 32 = 1*(x + 2)\,(x - 4)\,(x - 4).$$

1.5. ВЫЧИСЛЕНИЕ КОРНЕЙ ПОЛИНОМОВ С ЦЕЛЫМИ КОЭФФИЦИЕНТАМИ В ФОРМЕ ПРОСТЫХ ДРОБЕЙ

Число x_j называется *корнем* (*нулем*) целой рациональной функции $f(x)$ с действительными коэффициентами, если

$$f(x_j) = \sum_{k=0}^{n} a_{n-k} \cdot x_j^k = 0 \cdot \qquad (1.14)$$

Вычисление корней полиномов, таким образом, сводится к решению алгебраических уравнений (1.14). Для нахождения рациональных корней полиномов с целыми коэффициентами обычно используется хорошо известная схема Горнера [Корн, Корн, 1978; Бронштейн, Семендяев, 1981]. В алгоритме применено предложенное в работе [Pek, 1961] расширение этой схемы, суть которого состоит в следующем. Полином (1.13) с целыми коэффициентами при $a_0 \cdot a_n \neq 0$ имеет в качестве корня несократимую дробь p/q тогда и только тогда, когда

$$\sum_{k=0}^{n} a_k \cdot p^k \cdot q^{n-k} = 0 \cdot$$

Кроме того, q должно быть множителем a_n, а p - множителем a_0. Ненулевые рациональные корни p/q могут быть выведены на внешний носитель информации в файл, имя которого передается при вызове процедуры.

Формальные параметры процедуры. *Входные: a* (тип ***integer***) - одномерный массив размером от 0 до *n,* в котором размещают коэффициенты полинома (1.13); *n* (тип ***integer***) - степень полинома; *k* (тип ***string***) - имя файла на внешнем носителе (магнитном диске), в который будут записываться результаты счета. *Выходные: p, q* (тип ***integer***) - числители и знаменатели найденных рациональных корней, выводятся на внешний носитель.

```
PROCEDURE RATF (A : MAS1; VAR N : INTEGER;
                K: STRING);
VAR I, P, Q, R, T, F, G, A0, AN : INTEGER; F : TEXT;
BEGIN
  A0:=ABS(A[0]);
  AN:=ABS(A[N]);
  ASSIGN (F,K);
  REWRITE (F);
  FOR P:=1 TO A0 DO
  BEGIN
```

```
{ *** ???? P ? Q ?? ???????? ??????????? ????????????
?????????????? A[0] ? A[N], ????????? ?? ????? ????? ???
??????????? ? ???????????????? ?????? ????? ***}
      IF (A0 DIV P * P) = A0 THEN
      BEGIN
          FOR Q:=1 TO AN DO
          BEGIN
              IF (AN DIV Q * Q) = AN THEN
              BEGIN
{*??????????? ???????? ??? ????????, ???????? ??   P/Q
  ??????, ??? ???? ??????????? ???????? R ???? ????????
?????? ?????????? ?????????? ? ???????? ***}
                  F := A[0];
                  G := A[0];
                  T := P;
                  FOR I:=1 TO N DO
                  BEGIN
                      R := A[I]*T;
                      F:=F*Q+R;
                      G:=-G*Q+R;
                      T:=T*P
                  END;
                  IF F=0 THEN
                      WRITE (F,P:10:5,' ',K:5,Q:10:5);
                  IF G=0 THEN
                      WRITE (F,P:10:5,' ',K:5,Q:10:5);
              END;
          END; {**Q**}
      END;
  END; {**P**}
END.
```

Процедура, в которой представлен исправленный, ординарно переработанный и модифицированный алгоритм [Perry, 1962], переведена авторами с языка *ALGOL* [Агеев и др., 1976] на язык *PASCAL* с сохранением оригинального синтаксиса процедуры и проверена на машине IBM PC/AT-286 для тех же полиномов, что и в указанной выше работе :

$$P_3^{(1)} \equiv -36x^3 + 3x^2 + 14x + 3; \qquad P_3^{(2)} \equiv -24x^3 - 22x^2 - x + 3.$$

При этом были получены результаты, полностью совпадающие с результатами Агеева и др. [1976]:

для $P_3^{(1)}$: $x_1 = -{}^1/_3, x_2 = {}^3/_4, x_3 = -{}^3/_9$;

для $P_3^{(2)}$: $x_1 = {}^1/_2, x_2 = -{}^1/_3, x_3 = {}^3/_4, x_4 = {}^3/_6$.

Из результатов тестирования процедуры видно, что во втором случае $x_4 = x_1$, и, кроме того, выдается избыточное значение корня p/q, в котором p и q содержат общий множитель (это было также отмечено в работе [Halstead, 1962] для алгоритма Перри [Perry, 1962]).

§ 2. РЕШЕНИЕ НЕЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ С ОДНОЙ ПЕРЕМЕННОЙ

Нелинейное алгебраическое уравнение с одной переменной в общем случае может быть записано в виде

$$F(x) = 0, \qquad (1.15)$$

где функция $F(x)$ определена и непрерывна на конечном или бесконечном интервале $a < x < b$.

Всякое значение $\xi \in [a, b]$, обращающее функцию $F(x)$ в нуль, т.е. когда $F(\xi) = 0$, называется *корнем уравнения* (1.15) или *нулем функции* $F(x)$. Число ξ называется *корнем k-й кратности*, если при $x = \xi$ вместе с функцией $F(x)$ равны нулю и ее производные до порядка $(k - 1)$ включительно:

$$F(x) = F'(x) = \ldots = F^{(n-1)}(x) = 0.$$

Однократный корень называется *простым*. Два уравнения называются *равносильными* (*эквивалентными*), если множества их решений совпадают. Нелинейные уравнения с одной переменной подразделяются на *алгебраические*, когда функция $F(x)$ в формуле (1.15) является алгебраической, и *трансцендентные* в противном случае. Большинство алгебраических и трансцендентных нелинейных уравнений вида (1.15) аналитически (т.е. точно) не решается, поэтому на практике для нахождения корней часто используются численные методы. Рассмотрим некоторые из них [Березин, Жидков, 1962; Бахвалов, 1973а, 1973б и др.].

Задача численного нахождения действительных и комплексных корней уравнения [1.15] обычно состоит из двух этапов: отделения корней, т.е. нахождения достаточно малых окрестностей рассматриваемой области, в которых содержится одно значение корня, и уточнения корней, т.е. их вычисления с заданной степенью точности в некоторой окрестности. В связи с этим рассмотрим вначале задачу отделения корней, а затем ряд итерационных методов их уточнения.

2.1. ЗАДАЧА ОТДЕЛЕНИЯ КОРНЕЙ. УТОЧНЕНИЕ КОРНЕЙ МЕТОДОМ ПОЛОВИННОГО ДЕЛЕНИЯ (МЕТОД ДИХОТОМИИ)

В общем случае редко удается точно найти все корни в алгебраических уравнениях, а если к тому же коэффициенты в уравнении даны с погрешностью, то вопрос о точном определении корней вообще теряет всякий смысл. Однако если предположить, что задано уравнение типа (1.15), то тогда без ограничения общности можно утверждать, что $F(x)$ имеет корни, для которых существует окрестность $(\pm\delta)$, содержащая только один простой корень. Такой корень иногда называют *изолированным*. В результате общая задача нахождения корней или нулей функции будет состоять из следующих этапов:

1) отделения корней, т.е. установления интервала $[-\delta,+\delta]$, где содержится один и только один корень уравнения;

2) задачи уточнения одним из известных методов найденного корня ξ с заданной погрешностью ε.

Предположим теперь, что найден отрезок [*a, b*] такой, что $F(a)F(b) < 0$. Тогда, согласно теореме Больцано-Коши [Бахвалов, 1973б], внутри отрезка [*a, b*] существует точка ξ, в которой $F(\xi) = 0$. Далее необходимо убедиться, что найденная точка ξ единственная на отрезке [*a, b*]. Одним из методов является деление отрезка на несколько частей, например на четыре, и проверка на концах каждого из отрезков знака функции.

Нули функции на практике вычисляют приближенно несколькими способами. Одним из самых распространенных и не очень точных является *графический метод*, заключающийся в том, что $F(x)$ представляют как $F(x) = \varphi(x) + \psi(x)$, где $\varphi(x)$ и $\psi(x)$ более простые по сравнению с $F(x)$ функции. Далее строят два графика $y = \varphi(x)$; $y = \psi(x)$ и определяют точки их пересечения. Этим методом выгодно решать уравнения вида $x^n + ax + b = 0$ или $ax + b + \sin(cx) = 0$ и т.п. Но следует помнить, что этот метод дает лишь грубое приближение решения.

Другим, не менее распространенным является *метод производных.* Он заключается в том, что ищут и приравнивают к нулю производную функции $F'(x)$. Затем на отрезках $(-\infty, x_1)$, (x_1, x_2), …, $(x_n, +\infty)$ рассматривают знак функции $F'(x)$, где x_i - корни уравнения $F'(x) = 0$. Таким образом, всю числовую ось разбивают на два интервала и более. Этот метод еще называют *методом экстремумов функции.*

Если исследуемая функция представлена полиномом *n*-й степени, то используют метод удаления корней: определяют один корень, и по теореме Виетта функцию $F(x)$ представляют как $F(x) = g(x)(x - x_1)$, где x_1 - первый найденный корень, а $g(x)$ - полином степени $(n - 1)$. Для проверки кратности корня x_1 следует подставить в $g(x)$, и если $g(x_1) = 0$, то говорят, что x_1 является *кратным корнем*, а $F(x)$ записывается $F(x) = g(x)(x - x_1)^2$, где $g(x)$ - теперь полином степени $(n - 2)$. Следуя этому процессу, можно удалить все корни, т.е. представить

$$f(x) = \prod_{i=0}^{n} (x - x_i) \cdot$$

Чтобы погрешность с каждым шагом не увеличивалась, а очередной корень определялся с высокой степенью точности, следует уточнение корня делать по $F(x)$, а не по $g(x)$. Это особенно важно, когда удалено много (больше половины) корней.

На практике предполагаемые корни уточняют различными специальными вычислительными методами. Одним из них можно назвать *метод дихотомии* (*бисекции, половинного деления*), относящийся к итерационным. Он состоит в построении последовательности вложенных отрезков, на концах которых $F(x)$ имеет разные знаки. Каждый последующий отрезок получают делением пополам предыдущего. Этот процесс построения последовательности вложенных отрезков позволяет найти нуль функции ($F(x) = 0$) с любой заданной точностью.

Опишем подробно один шаг итерации. Пусть на *k*-м шаге найден отрезок $[a_k, b_k]$, на концах которого $F(x)$ меняет знак. Заметим, что обязательно $[a_k, b_k] \in [a, b]$. Разделим теперь отрезок $[a_k, b_k]$ пополам и выделим $F(\xi)$, где ξ - середина $[a_k, b_k]$. Здесь возможны два случая: первый, когда $F(\xi) = 0$, тогда мы говорим, что корень найден; второй, когда $F(\xi) \neq 0$, тогда сравниваем знак $F(\xi)$ с $F(a_k)$ и $F(b_k)$ и из двух половин $[a_k, \xi]$ и $[\xi, b_k]$ выбираем ту, на концах которой функция меняет свой знак. Таким образом, $a_k = a$, $b_k = \xi$, если $F(\xi)F(a_k) < 0$, и $a_k = \xi$, $b_k = b$, если $F(\xi)F(b_k) < 0$.

При заданной точности ε деление пополам продолжают до тех пор, пока длина отрезка не станет меньше 2ε, тогда координата середины последнего найденного отрезка и есть значение корня требуемой точности.

Метод дихотомии — простой и надежный метод поиска простого корня[1] уравнения $F(x) = 0$. Он сходится для любых непрерывных функций $F(x)$, в том числе и недифференцируемых. Недостатки метода:

1) проблема определения отрезка, на котором функция меняет свой знак (как правило, это отдельная вычислительная задача, наиболее сложная и трудоемкая часть решения);

2) если корней на выделенном отрезке несколько, то нельзя заранее сказать, к какому из них сойдется процесс;

3) не применим к корням четной кратности;

4) для корней нечетной, но высокой кратности метод неустойчив, дает большие ошибки;

5) медленно сходится. Для достижения ε необходимо выполнить N итераций[2], т.е. для получения 3 верных цифр ($\varepsilon = 0.0005$) надо выполнить около 10 итераций, если отрезок имеет единичную длину.

Программа, по которой можно вычислить корни методом дихотомии, построена по следующему алгоритму:

Øàã 1. Определить входные параметры *A, B, EPS.*

Øàã 2. Присвоить: $A1 \Leftarrow A$; $B1 \Leftarrow B$; $K \Leftarrow 0$.

Øàã 3. Присвоить: $X1 \Leftarrow A1$; $X2 \Leftarrow B1$; $K \Leftarrow K + 1$; $X3 \Leftarrow (B1+A1)/2$.

Øàã 4. Если $F(X1) \times F(X3) < 0$, то перейти на шаг 5 иначе на шаг 7.

Øàã 5. Присвоить: $B1 \Leftarrow X3$.

Øàã 6. Если $|A1 - B1| < EPS$, то перейти на шаг 10 иначе на шаг 3.

[1] Корень x называется простым, если $F(x) = 0$, а $F'(x) \neq 0$.

[2] $N = \log_2 (b - a) / \varepsilon$.

Øàã 7. Если $F(X2) \times F(X3) < 0$, то перейти на шаг 8 иначе на шаг 11.

Øàã 8. Присвоить: $A1 \Leftarrow X3$.

Øàã 9. Перейти на шаг 6.

Øàã 10. Печать: $X3$ - корень уравнения; K - количество итераций.

Øàã 11. $|A1 - B1| / 2$ - погрешность решения.

Øàã 12. Конец программы.

Это наиболее простое решение задачи, но не самое эффективное. Эффективность можно повысить, если:

1) заменить произведения $F(x_1)\cdot F(x_3)$ и $F(x_2)\cdot F(x_3)$ на использование встроенной функции sign(x, y). В тех версиях языка, где нет этой встроенной функции, можно заранее написать соответствующую процедуру;

2) определить процедуру-функцию, вычисляющую $F(x)$ только один раз;

3) заменить в операторе цикла медленный оператор $(A+B)/2$ на более быстрый $(A+B)$*0.5. Заметим, что именно для этой программы данное усовершенствование будет незаметно, хотя в случае больших программ учет скорости выполнения операций в машине дает ощутимый результат.

Формальные параметры процедуры. *Входные*: a, b (тип ***real***) - определяют длину отрезка; *eps* (тип ***real***) - определяет заданную точность вычислений; *it* (тип ***integer***) - определяет наибольшее разрешенное количество итераций (для избежания зацикливания процесса в случае неправильного определения отрезка). *Выходные:* x (тип ***real***) - в нем содержится искомый корень сравнения; k (тип ***integer***) - в него заносится количество выполненных итераций.

Учитывая все замечания, окончательный вариант процедуры BISECT может быть следующим:

```
PROCEDURE BISECT (A,B,EPS :REAL; IT:INTEGER;
          VAR X : REAL; VAR K:INTEGER);
VAR A1, B1: REAL; X1, X2, X3 : INTEGER;
BEGIN
  K := 0;
  X1 := SIGN (FUNC(A));
  X2 := SIGN (FUNC(B));
  A1 := A;
  B1 := B;
  REPEAT
     INC (K);
     X := (A1+B1)*0.5;
     X3 := SIGN (FUNC (X));
     IF X3=0 THEN EXIT;
     IF ABS(B1-A1)<(2*EPS) THEN EXIT;
     IF (X1=X2) AND (X2=X3) THEN EXIT;
     IF X1=X3 THEN
        BEGIN
           A1 := X;
           X1 := X3;
        END
     ELSE
        BEGIN B1 := X;
           X2 := X3;
        END;
  UNTIL K>IT;
END.
```

Перед началом работы программы определяют FUNC(x) - процедуру-функцию, по которой вычисляют значения $F(x)$. Тип функции должен быть вещественным. Если в библиотеке стандартного математического обеспечения отсутствует процедура-функция SIGN, то ее следует написать самостоятельно.

Предложенная процедура проверялась на примере решения уравнения $x^2 - 5 \sin x = 0$.

Графическим методом находился отрезок, на котором располагался один из корней данного уравнения [1.57; 3.14]; (второй корень тривиальный, $x = 0$ находится легко). Для того, чтобы найти корень на отрезке [1.57; 3.14] с указанной точностью, полагали *eps* = 0.0005.

Результаты проверки работы предлагаемой процедуры приводятся в табл. 1.5.

Таблица 1.5

Отрезок						Номер итерации
Левый конец		Правый конец		Центральная точка		
A	sign (A)	B	sign (B)	x	sign (x)	k
1.0	— 1	4.0	+1	2.5000		0
1.000000	— 1	2.500000	+1	1.750000	+1	1
1.750000	— 1	2.500000	+1	2.125000	+1	2
1.750000	— 1	2.125000	+1	1.937500	— 1	3
1.937500	— 1	2.125000	+1	2.031250	— 1	4
2.031250	— 1	2.125000	+1	2.078125	— 1	5
2.078125	— 1	2.125000	+1	2.101563	— 1	6
2.101563	— 1	2.125000	+1	2.089844	+1	7
2.101563	— 1	2.089844	+1	2.083984	+1	8
2.101563	— 1	2.083984	+1	2.086914	— 1	9

2.086914	— 1	2.083984	+1	2.085449	+1	10

Окончание таблицы 1.5

Отрезок						Номер итерации
Левый конец		Правый конец		Центральная точка		
A	sign (A)	B	sign (B)	x	sign (x)	k
2.086914	— 1	2.085449	+1	2.086182	— 1	11
2.086182	— 1	2.085449	+1	2.085815	+1	12
2.085815	— 1	2.086182	+1	2.085999	— 1	13
2.085815	— 1	2.085999	+1	2.085907	+1	14
2.085815	— 1	2.085907	+1	2.085953	— 1	15
2.085907	— 1	2.085953	+1	2.085930	+1	16
2.085930	— 1	2.085953	+1	2.085941	— 1	17
РЕШЕНИЕ: $X = 2.085936$; $F(x) = 0.0000066938$; $K = 18$						

2.2. ПРИБЛИЖЕННОЕ РЕШЕНИЕ УРАВНЕНИЯ $F(x) = 0$ МЕТОДОМ ХОРД (СЕКУЩИХ)

После того, как отрезок [a, b], на котором $F(x)$ меняет свой знак, определен (см. п. 2.1), можно уточнять корень разными методами. Метод дихотомии требует большого количества итераций и не всегда сходится к искомому корню [Бахвалов, 1973а] [3].

Если отрезок [a, b] делят не пополам, а в отношении $F(a):F(b)$, то получают точку x, расположенную ближе к предполагаемому корню уравнения по сравнению с точкой, найденной методом дихотомии. Новое приближенное значение корня принимается:

$$x^{(n+1)} = Q + h_n, \text{ где } h_n = \frac{(a-b)\cdot f(x_0)}{f(b)-f(x_n)} \quad . \qquad (1.16)$$

При $n \neq 0$ полагаем $x_0 = a$. Геометрически этот способ эквивалентен замене кривой $F(x)$ хордой A_0B, проходящей через A_0 (b, $F(b)$) и $B(a, F(a))$ (см. рис. 1.1). Из аналитической геометрии найдем x_1 как точку пересечения хорды A_0B с осью Ox:

$$\frac{x-a}{b-a} = \frac{y-f(a)}{f(b)-f(a)},$$

полагая при этом $y = 0$, имеем

$$x_1 = a - \frac{(b-a)\cdot f(a)}{f(b)-f(a)}.$$

Точку x_2 ищем как пересечение хорды A_1B, где A_1 (x_1, $F(x_1)$), с осью $0x$ (см. рис 1.1):

$$\frac{x-x_1}{b-x_1} = \frac{y-f(x_1)}{f(b)-f(x_1)},$$

полагая опять $y = 0$, имеем:

$$x_2 = x_1 - \frac{(b-x_1)\cdot f(x_1)}{f(b)-f(x_1)}.$$

Далее каждое очередное x_n будем определять по формуле

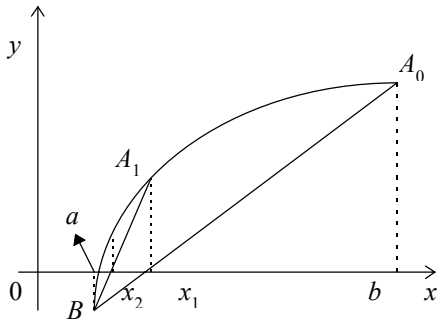

Рис. 1.1. *Графическая интерпретация метода хорд (секущих)*

$$x_n = x_{n-1} - \frac{(b-x_{n-1})\cdot f(x_{n-1})}{f(b)-f(x_{n-1})} \quad . \qquad (1.17)$$

Процесс продолжаем до тех пор, пока не начнет выполняться условие $|x_n - x_{n-1}| < \varepsilon$.

Оценим погрешность этого метода [Березин, Жидков, 1962]. Пусть найдутся x_n и x_{n-1} и пусть $F'(x)$ непрерывна и на всем отрезке [a, b] сохраняет свой знак, причем выполняется условие:

$$0 < m_1 < |F'(x)| < M_1 < +7.$$

Примем, что x_n определен по формуле (1.17), где $n = 1, 2, \ldots$, a; b - неподвижная точка. Учитывая, что $F(\xi) = 0$, имеем

[3] Если на выделенном отрезке находятся три корня функции и более или если корни близко расположены (в пределах заданной погрешности).

$$f(\xi) - f(x_{n-1}) = \frac{f(x_{n-1}) - f(b)}{x_{n-1} - b} \cdot (x_n - x_{n-1})$$

и, применив теорему Лагранжа о конечном приращении функции, получим:

$$(\xi - x_{n-1})\, F'(\xi_{n-1}) = (x_n - x_{n-1})\, F'(X_{n-1}),$$

где $\xi_{n-1} \in [x_{n-1}, \xi]$, $X_{n-1} \in [x_{n-1}, \xi]$, следовательно

$$|\xi - x_{n-1}| = \frac{|f'(X_{n-1}) - f'(\xi_{n-1})|}{|f'(\xi_{n-1})|} \cdot |x_n - x_{n-1}| \cdot \quad (*)$$

Так как $F'(x)$ сохраняет постоянный знак на отрезке $[a, b]$, причем $\xi_{n-1} \in [x_{n-1}, \xi]$ и $X_{n-1} \in [x_{n-1}, \xi]$, то числитель дроби можно ограничить разностью между max ($F'(\xi)$) и min($F'(\xi)$) на отрезке $[a, b]$:

$$|F'(X_{n-1}) - F'(\xi_{n-1})| < M_1 - m_1 . \quad (**)$$

Из выражений (*) и (**) находим

$$|\xi - x_{n-1}| \le \frac{M_1 - m_1}{m} \cdot |x_n - x_{n-1}|, \quad (1.18)$$

где $M_1 = \sup(|F'(x)|)$ и $m_1 = \inf(|F'(x)|)$ на отрезке $[a, b]$. Если при этом отрезок $[a, b]$ мал, то имеет место неравенство $M_1 < m_1$, тогда оценка $|\xi - x_{n-1}| \le |x_n - x_{n-1}|$ еще более упрощается, т.е., как только обнаружили, что расстояние между двумя последовательно вычисленными корнями $|x_n - x_{n-1}| \le \varepsilon$, где ε - заданная предельная погрешность, то можно гарантировать, что $|\xi - x_{n-1}| \le \varepsilon$. Для вычисления значения корня на отрезке $[a, b]$ с заданной точностью ε можно воспользоваться процедурой HORD.

Формальные параметры процедуры. *Входные: a, b* (тип ***real***) - отрезок, на котором ищется корень; *eps* (тип ***real***) - точность вычисления корня; *k* (тип ***integer***) - разрешенное число итераций; *FUNC* - внешняя процедура-функция. *Выходные: k* (тип ***integer***) - количество выполненных итераций; *x* (тип ***real***) - найденное значение корня с заданной точностью *eps*.

```
PROCEDURE HORD (A,B,EPS:REAL; IT:INTEGER;
          VAR X : REAL; VAR K:INTEGER);
VAR X1,X2,X3 : REAL; K1 : INTEGER;
BEGIN
  K := 1;
  X1 := FUNC(A);
  X2 := FUNC(B);
  X3 := B - X2*(B-A)/(X2-X1);
  IF SIGN(X2)=SIGN(FUNC(X3)) THEN
     K1:=1
  ELSE
  BEGIN
     K1:=2;
     X1:=X2;
  END;
  REPEAT
     INC (K);
     X := X3;
     CASE K1 OF
        1: X2:= X3-FUNC(X3)*(X3-A)/(FUNC(X3)-X1);
        2: X2:= X3-FUNC(X3)*(B-X3)/(X1-FUNC(X3));
     END;
     X3 := X2;
  UNTIL (K>IT) OR (ABS(X-X2)<EPS);
END.
```

Для проверки процедуры решалось уравнение

$$2\cos(x + \pi/6) + x^2 - 3x + 2 = 0.$$

Первоначально корни уравнения определяли с точностью 0.1 графическим методом, а затем найденное значение корня уточняли методом хорд до 0.0001.

Перепишем уравнение в виде

$$2\cos(x + \pi/6) = -x^2 + 3x - 2,$$

и если построить два графика: $y = 2\cos(x + \pi/6)$ и $y = -x + 3x - - 2$, то можно убедиться, что один корень ~1.1, а второй ~2.9. Поэтому первый интервал выбираем [0.9; 1.3], второй - [2.7; 3.1]. Точность установим $\varepsilon = 0.0005$.

Процедура-функция может выглядеть так:

```
FUNCTION FUNC (X :REAL) : REAL;
BEGIN FUNC := X*X - 3*X +2.0 + 2.0*COS(X+PI/6);
END.
```

Результаты расчетов с использованием подпрограммы HORD приводятся в табл. 1.6.

Таблица 1.6

Первый корень на интервале [0.9; 1.3]			Второй корень на интервале [2.7; 3.1]		
x_i	x_{i+1}	№ итерации	x_i	x_{i+1}	№ итерации
1.044879	1.032336;	$k = 1$	2.915101	2.953624;	$k = 1$
1.032336	1.031823;	$k = 2$	2.953624	2.959635;	$k = 2$
1.031823	1.031803;	$k = 3$	2.959635	2.960551;	$k = 3$
1.031803	1.031802;	$k = 4$	2.960551	2.960690;	$k = 4$
			2.960690	2.960711;	$k = 5$
			2.960711	2.960714;	$k = 6$
$x = 1.031803$; $F(x) = —\ 0.0000025773$; $k = 4$			$x = 2.960711$; $F(x) = —\ 0.0000135768$; $k = 6$		

2.3. ПРИБЛИЖЕННОЕ РЕШЕНИЕ УРАВНЕНИЯ $F(x) = 0$ МЕТОДОМ КАСАТЕЛЬНЫХ (Ньютона)

Если известно начальное приближение решения уравнения $F(x) = 0$ на отрезке $[a, b]$, то уточнить корень можно, как уже говорилось, любым способом. Одним из самых эффективных и точных решений является метод Ньютона (метод касательных), который состоит в построении итерационной последовательности [Бахвалов, 1973б]

$$x_{n+1} = x_n - F(x_n)/F'(x_n),$$

сходящейся к корню уравнения $F(\xi) = 0$. Для применения этого метода необходимо существование первой и второй производных и их знакопостоянство на исследуемом отрезке.

Рассмотрим метод более подробно. Пусть найдено некоторое $x_n = \xi$, где $\xi \in [a, b]$. При уточнении корня по методу Ньютона полагаем $\xi = x_n + h_n$, где h_n - достаточно малое число. Тогда, считая ξ корнем уравнения, находим h_n, применив формулу Тейлора:

$$0 = F(\zeta) = F(x_n + h_n) = F(x_n) + h_n F'(x_n),$$

откуда $h_n = -F(x_n)/F'(x_n)$, и окончательно получаем рекуррентную формулу

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}. \qquad (1.19)$$

Достаточные условия сходимости определяются теоремой.

Теорема: *Пусть $F(x)$ определена и дважды дифференцируема на отрезке $[a, b]$, причем $F(a)F(b) < 0$, а производные $F'(x)$ и $F''(x)$ сохраняют знак на отрезке $[a, b]$. Тогда, исходя из начального приближения $x_0 \in [a, b]$, удовлетворяющего неравенству $F'(x_0)F''(x_0) > 0$, можно построить последовательность*

$$x_{n+1} = x_n - F(x_n)/F'(x_n), \quad n = 0, 1, ...,$$

сходящуюся к единственному на отрезке $[a, b]$ решению ξ уравнения $F(\xi) = 0$.

Если нулевое приближение выбрано достаточно близко к корню, то итерации сходятся очень быстро, со скоростью геометрической прогрессии.

Алгоритм данного метода очень простой и ясный и имеет, как и метод хорд, графическую интерпретацию (рис. 1.2).

1. Через точку $A_0(x, y)$ с координатами $x = b$, $y = F(b)$ проводим касательную.

2. Находим пересечение касательной с осью $0x$ (точка x_1).

3. Находим значение функции в точке $x = x_1$; $y = F(x_1)$.

4. Проводим касательную в точке $x = x_1$; $y = F(x_1)$ (точка A_1).

5. Находим точку пересечения касательной с осью $0x$ (точку x_2), и так далее до выполнения условия

$|x_n - x_{n-1}| \le \varepsilon$. За корень ξ принимаем x_n.

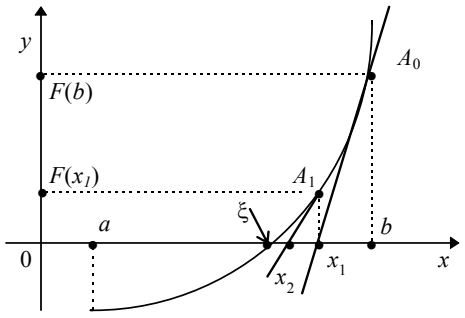

Рис. 1.2. *Геометрическая интерпретация метода касательных (Ньютона)*

Для оценки погрешности n-го приближения корня [Березин, Жидков, 1962] можно воспользоваться неравенством:

$$|\xi - x_n| \le |x_n - x_{n-1}| M_2 / (2m_2),$$

где $M_2 = \sup |F''(x)|$ на отрезке $[a, b]$, $m_2 = \inf |F'(x)|$ на отрезке $[a, b]$. Таким образом, если

$$|x_n - x_{n-1}| \le \varepsilon, \text{ то } |\xi - x_n| \le M_2 \cdot \varepsilon^2 / (2m_2).$$

Последнее неравенство означает, что при удачном начальном приближении корня после каждой итерации количество верных цифр удваивается. Следовательно, процесс вычисления корня можно прекратить, если

$$|x_n - x_{n-1}| \le \varepsilon = \sqrt{2m_2\varepsilon / M_2}.$$

Заметим еще раз, что метод Ньютона эффективен, если выбрано хорошее начальное приближение корня и график функции имеет большую крутизну в окрестности корня. В этом случае процесс быстро сходится. Если же численное значение $F'(x)$ вблизи корня мало, т.е. график почти параллелен оси $0x$, то процесс вычисления корня будет долгим и ошибки округления чисел в машине могут вызвать обратный процесс - решение начнет расходиться. В этом случае можно посоветовать воспользоваться для уточнения корня другими более эффективными методами.

Алгоритм решения задачи достаточно простой и требует особых пояснений. Процедура NEWTON вычисления приближенного значения корня уравнения $F(x) = 0$ методом Ньютона (касательных) по формуле (1.19) приводится ниже. Она была написана на языке *FORTRAN*, [Плис, Сливина, 1983] и переведена авторами на язык *PASCAL*.

Формальные параметры процедуры. *Входные*: $x0$ (тип ***real***) - начальное приближение; n (тип ***integer***) - максимально допустимое количество итераций; *eps* (тип ***real***) - значение ε в условии окончания итерационного процесса; *func* - имя внешней подпрограммы-функции (тип ***real***), по которой вычисляется значение функции $F(x)$;

funcp - имя внешней подпрограммы-функции (тип ***real***), по которой вычисляется значение производной функции $F'(x)$. *Выходные: i* (тип ***integer***) - количество выполненных итераций; *ih* (тип ***integer***) - указатель причины окончания процесса вычислений: если *ih* = 0, то корень уравнения найден с заданной точностью; если *ih* = 1, то итерации не сходятся к корню уравнения, и это значит, что на отрезке $[a,b]$ не выполняется условие теоремы; если *ih* = 2, то количество совершенных итераций превзошло максимально допустимое число *N*, что говорит либо о плохом начальном приближении, либо о малости величины $|F'(x)|$ в окрестности корня; *x* (тип ***real***) - вычисленное значение корня.

```
PROCEDURE NEWT (N: INTEGER; X0,EPS:REAL;
                VAR I,IH:INTEGER; VAR X:REAL);
VAR DEL,XS,DEL0,Y: REAL;
BEGIN          DEL0 := 1.E12;
  XS := X0;       I := 0;
  IH := 0;
  REPEAT         Y := FUNCP (XS);
     X := XS - FUNC (XS) / Y;
     DEL := ABS ( X - XS);
     IF DEL <= DEL0 THEN
     BEGIN
        IF I>N THEN
        BEGIN
           IH := 2;     EXIT;
        END;
        INC(I);      XS := X;
        DEL0 := DEL;
     END
     ELSE
     BEGIN
        IH := 1;
        EXIT;
     END;
  UNTIL DEL < EPS;
END.
```

Для проверки процедуры решалось уравнение

$$F(x) = e^x - 10x.$$

Все вычисления выполнялись с точностью до 10^{-4}. Очевидно, что это уравнение не может быть решено аналитическими методами.

Корни уравнения ζ_1 и ζ_2 легко определить графически (рис. 1.3). Если преобразовать исходное уравнение к виду $y = e^x$; $y = 10x$, то первый корень будет лежать на отрезке [0, 1], а второй - [3, 4]. Определим ε_0. Для этого оценим производные $F'(x)$ и $F''(x)$ на найденных отрезках:

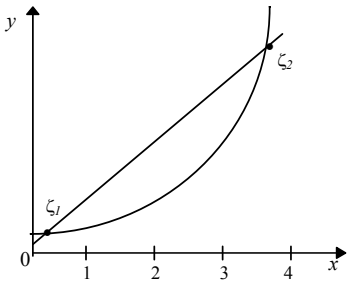

Рис. 1.3. *Графическое решение уравнения* $F(x) = e^x - 10x$

$$F'(x) = e^x - 10;\ F''(x) = -e^x.$$

Очевидно, что $F''(x)$ сохраняет свой знак во всей области определения, в том числе и на указанных отрезках. При этом

$$m_1^{(1)} = \min_{[0,1]} |f'(x)| = \min_{[0,1]} |e^x - 10| \cong 7;$$

$$m_1^{(2)} = \min_{[3,4]} |f'(x)| = \min_{[3,4]} |e^x - 10| \cong 16;$$

$$M_1^{(1)} = \sup_{[0,1]} |f''(x)| = \sup_{[0,1]} |e^x| \cong e;$$

$$M_1^{(2)} = \sup_{[3,4]} |f''(x)| = \sup_{[3,4]} |e^x| \cong e^4.$$

Если требуемая точность 0.0001, то в условии окончания процесса:

$$\varepsilon^{(1)} = \sqrt{2m_1\varepsilon / M_1} = \sqrt{2 \cdot 7 \cdot 10^{-4} \cdot e^{-1}};$$

$$\varepsilon^{(2)} = \sqrt{2m_1\varepsilon / M_1} = \sqrt{2 \cdot 16 \cdot 10^{-4} \cdot e^{-4}} = 4 \cdot 10^{-2} \cdot \sqrt{2} \cdot e^{-2}.$$

В качестве начального приближения на отрезке [0, 1] выбираем $x_0 = 0$, а на отрезке [3, 4] - $x_0 = 3$.

Процедуры-функции FUNC и FUNCP будут выглядеть следующим образом:

```
FUNCTION FUNC (X:REAL): REAL;
  BEGIN
     F UNC := 10.0*X - EXP(X);
  END;

FUNCTION FUNCP (X:REAL): REAL;
  BEGIN
     FUNCP :=10.-EXP(X);
  END;
```

Результаты работы приведены в табл. 1.7.

Таблица 1.7

Первый корень на отрезке [0, 1]			Второй корень на отрезке [3, 4]		
Итерация	x_i	x_{i+1}	Итерация	x_i	x_{i+1}
$i = 0$	0.000000	0.111111	$i = 0$	3.000000	3.983038
$i = 2$	0.111111		$i = 2$	3.983038	3.665970
		0.111833	$i = 3$	3.665970	3.582299
			$i = 4$	3.582296	3.577170
$x = 0.111833$			$x = 3.577170$		

2.4. ПРИБЛИЖЕННОЕ РЕШЕНИЕ УРАВНЕНИЯ $F(x) = 0$ КОМБИНИРОВАННЫМ МЕТОДОМ (ХОРД И КАСАТЕЛЬНЫХ)

Методы хорд (1.17) и касательных (1.19) дают приближения корня с разных сторон. Поэтому их часто применяют в сочетании. Комбинации методов могут быть различны, но во всех случаях уточнение корня идет быстрее. Рассмотрим несколько вариантов расчетных схем комбинированного метода.

Пусть дано уравнение (1.15), корень ξ отделен и находится на отрезке $[a, b]$. Применим комбинированный метод хорд и касательных с учетом графика функции (рис. 1.4 и 1.5).

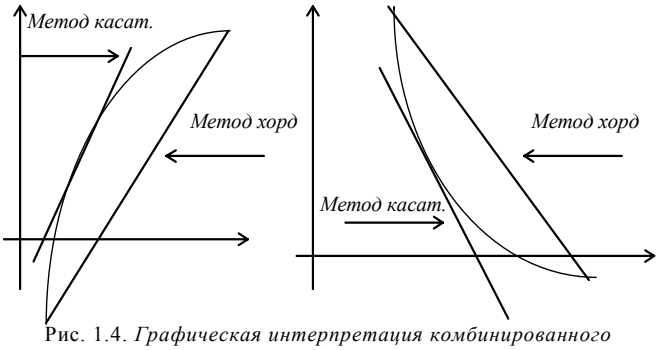

Рис. 1.4. *Графическая интерпретация комбинированного метода при $F'(x)F''(x) > 0$*

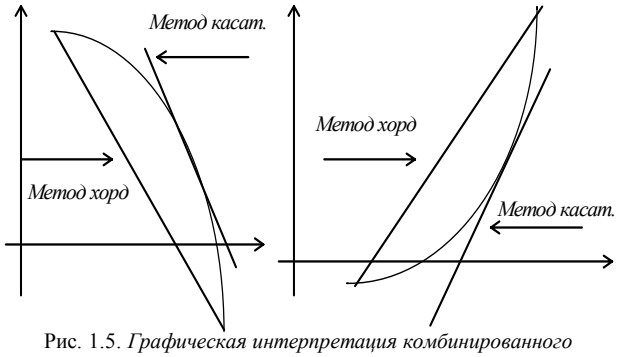

Рис. 1.5. *Графическая интерпретация комбинированного метода при $F'(x)F''(x) < 0$*

Если $F'(x)F''(x) > 0$ (рис. 1.4), то метод хорд дает приближение корня с недостатком, а метод касательных - с избытком. Если же $F'(x)F''(x) < 0$ (рис. 1.5), то, наоборот, метод хорд дает приближение корня с избытком, а касательных - с недостатком.

Однако во всех случаях истинный корень ξ заключен между промежуточными корнями, получающимися по методу хорд (1.17) и методу касательных (1.19), т.е. выполняется неравенство

$$a < x < \zeta < X < b,$$

где x - значение корня с недостатком, а X - с избытком.

Из этих соображений складывается следующая схема вычислений: в первом случае, когда выполняется условие $F'(x)F''(x) > 0$, то последовательность $\{x_i\}$ (слева) образуется по методу хорд, а последовательность $\{X_i\}$ (справа) образуется по методу касательных:

$$x_i = x_{i-1} - \frac{f(x_{i-1})\cdot(X_{i-1} - x_{i-1})}{f(X_{i-1}) - f(x_{i-1})};$$

$$X_i = X_{i-1} - \frac{f(X_{i-1})}{f'(X_{i-1})}.$$

Во втором случае, когда $F'(x)$ $F''(x)<0$, слева лежит приближенное значение корня, найденное по методу касательных, а справа - по методу хорд. При этом расчетные формулы будут несколько иные:

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

$$X_i = X_{i-1} - \frac{f(X_{i-1})\,(x_{i-1} - X_{i-1})}{f(x_{i-1}) - f(X_{i-1})}.$$

Другая вычислительная схема этого же метода дает более быструю сходимость:

$$x_i = x_{i-1} - \frac{f(x_{i-1})\,(X_{i-1} - x_{i-1})}{f(X_{i-1}) - f(x_{i-1})};$$

$$X_i = x_i - \frac{f(x_i)}{f'(x_i)}.$$

Для начала процесса в этом случае полагают $x_0 = a$, $X_0 = b$.

За приближенное значение для окончания вычислений при любой из рассмотренных схем принимают среднее между X_n и x_n: $\xi \sim {}^1/_2 (X_n + x_n)$.

При организации вычислений можно воспользоваться процедурами из п. 2.2 и 2.3. Здесь предлагается оригинальная программа HORDKAS, объединяющая оба алгоритма.

Формальные параметры процедуры. *Входные*: *a*, *b* (тип ***real***) - заданный отрезок, на котором ищется решение; *eps* (тип ***real***) - точность решения (погрешность); *it* - наибольшее разрешенное число итераций. *Выходные*: *x* (тип ***real***) - решение, найденное с заданной точностью; *k* (тип ***integer***) - целое число, равное 0, если процесс решения прошел удачно, и 1, если решение расходится.

```
PROCEDURE HORDKAS (A,B,EPS:REAL; IT:INTEGER;
                   VAR X : REAL; VAR K:INTEGER);
VAR X1,X2,X3,A1,B1 : REAL;
    K1 : INTEGER;
BEGIN
  K := 1;
  X1 := FUNC(A);
  X2 := FUNC(B);
  A1 :=A;
  B1 := B;
```

```
    x3 := b - x2*(b-a)/(x2-x1);
    IF SIGN(x2)=SIGN(FUNC(x3)) THEN k1:=1
    ELSE   k1:=2;
    REPEAT
        INC (k);
        x := x3;
        CASE k1 OF
            1:x2:= x3-FUNC(x3)*(x3-a1)/
                    (FUNC(x3)-FUNC(a1));
            2:x2:= x3-FUNC(x3)*(b1-x3)/
                     (FUNC(b1)-FUNC(x3));
         END;
        x3 := x2;
        CASE k1 OF
            1: BEGIN
                    a1 := a1 - FUNC(a1)/FUNC1(a1);
                    x1 := a1;
               END;
            2: BEGIN
                    b1 := b1 - FUNC(b1)/FUNC1(b1);
                    x1 := b1;
              END;
            END;
    UNTIL (k>it) OR (ABS(x-x2)<eps);
END.
```

Для тестирования процедуры комбинированным методом хорд и касательных уточним до 0.001 корни уравнения $x^3 + +3x^2 - 24x + 1 = 0$. Для этого сначала отделим корни методом исследования производных:

$$F'(x) = 3x^2 + 6x - 24; \qquad F''(x) = 6x + 6.$$

Очевидно, что $F'(x) = 0$ при $x_1 = -4$; $x_2 = 2$.

Тогда можно составить таблицу знаков функции (табл. 1.8), из которой видно, что уравнение действительно имеет три корня, расположенных на интервалах:

$$x_1 \in [-\infty, -4]; x_2 \in [-4, 2] \text{ и } x_3 \in [2, +\infty].$$

Таблица 1.8

Параметр	Характеристики интервалов			
x	$-\infty$	-4	+2	$+\infty$
sign($F(x)$)	-	+	-	+

Уменьшив найденные интервалы до единичной длины, получим $x_1 \in [-7, -6]$; $x_2 \in [0, 1]$; $x_3 \in [3, 4]$.

Теперь уточним корни комбинированным методом хорд и касательных. Для этого определим знак произведения $F'(x)F''(x)$ на указанных интервалах (табл. 1.9). Анализируя результаты, представленные в табл. 1.9, делаем вывод, что на первом интервале [-7,-6] за начальное приближение ξ_0 следует принять -6.5, на втором [0,1] - $\xi_0 = 0.5$, а на третьем [3, 4] - $\xi_0 = 3.5$ (как центральные точки указанных интервалов). Результаты проверки процедуры приводятся в табл. 1.10.

Таблица 1.9

Параметр	Знаки функций на интервалах		
	[-7, -6]	[0, 1]	[3, 4]
$F'(x)$	> 0	< 0	> 0
$F''(x)$	< 0	> 0	> 0
$F'(x)\,F''(x)$	< 0	< 0	> 0

Таблица 1.10

Первый корень на интервале [-7,-6]			Второй корень на интервале [0, 1]			Третий корень на интервале [3, 4]		
Итерация	Метод хорд	Метод касат.	Итерация	Метод хорд	Метод касат.	Итерация	Метод хорд	Метод касат.
$i = 1$	-6.5781	-6.6330	$i = 1$	0.05000	0.04193	$i = 1$	3.50000	3.58282
$i = 2$	-6.6666	-6.6381	$i = 2$	0.04166	0.04189	$i = 2$	3.64583	3.59601
$i = 3$	-6.6383	-6.6381	$i = 3$	0.04188	0.04188	$i = 3$	3.59717	3.59626
$i = 4$	-6.6381	-6.6381				$i = 4$	3.59627	3.59626
$x = -6.638156$;			$x = 0.041889$;			$x = 3.596267$;		
$F(x) = 0.0000001204$; $i = 4$			$F(x) = -0.0000000306$; $i = 3$			$F(x) = -0.0000031128$; $i = 4$		

2.5. ПРИБЛИЖЕННОЕ РЕШЕНИЕ УРАВНЕНИЯ $F(x) = 0$ МЕТОДОМ ИТЕРАЦИЙ

Часто этот метод называют еще *методом последовательных приближений.*

Заменим уравнение (1.15) равносильным ему $w(x) - x = 0$, преобразовав для этого функцию $F(x)$. Из этого уравнения получим $w(x) = x$ и выберем любым способом $x_0 \in [a, b]$, которое затем подставим в левую часть уравнения $w(x_0) = x_1$. Полученное значение x_1 снова подставим в левую часть и получим $w(x_1) = x_2$. Продолжая этот процесс, запишем последовательность чисел $x_1, x_2, \ldots, x_n$, которая может либо сходиться, т.е. *иметь предел*, либо расходиться, т.е. *не иметь предела*. Тогда в соответствии с полученным результатом в первом случае этот предел назовем корнем уравнения (1.15), во втором случае сделаем вывод о невозможности получения решения данным способом. Оба этих варианта определяются теоремой сходимости итерационного процесса.

Теорема. *Пусть на отрезке* $[a, b]$ *имеется единственный корень уравнения* $x = w(x)$ *и во всех точках этого отрезка производная* $F'(x)$ *удовлетворяет неравенству* $|F'(x)| < q < 1$. *Если при этом выполняется и условие* $a < w(x) < b$, *то итерационный процесс сходится, а за ну-*

левое приближение x_0 можно взять любое число из отрезка [*a, b*].

Последнее утверждение означает, что $\{x_i\} \in [a, b]$. Чем меньше $|w'(x)|$, тем лучше сходимость итерационного процесса.

Пусть теперь ξ - точное значение корня, а x_k - приближенное. Попробуем оценить погрешность метода. Согласно теореме [Бахвалов, 1973а], q определяется из $|w'(x_k)| < q < 1$. Тогда должно быть справедливо соотношение

$$|\xi - x_k| < q(q - 1)^{-1} |x_k - x_{k-1}|.$$

Если положить, что ξ отличается от x_k на величину, меньшую ε, то $|\xi - x_k| < \varepsilon$ и последовательность $\{x_k\}$ надо вычислять до тех пор, пока не будут выполнены условия

$$q(q - 1)^{-1} |x_k - x_{k-1}| < \varepsilon \text{ или } |x_k - x_{k-1}| < \varepsilon (q - 1)/q ,$$

откуда можно сделать следующий вывод: так как уравнение $F(x) = 0$ приводится к виду $w(x) = x$ разными способами, то для метода итераций следует выбрать такое уравнение для $w(x)$, для которого выполняется условие теоремы.

В заключение отметим, что при использовании метода простых итераций основным и, пожалуй, самым важным моментом является выбор функции $w(x)$ в уравнении $x = w(x)$, эквивалентном исходному. Для метода итераций следует подбирать функцию $w(x)$ так, чтобы обязательно выполнялось условие $|w'(x)| < q < 1$. При этом не нужно забывать, что скорость сходимости последовательности приближений $\{x_i\}$ к корню ξ тем выше, чем меньше число q.

Если предположить, что все условия для реализации расчетов по методу простой итерации выполнены, то программа (без проверок) может быть очень простой:

```
FUNCTION ITER1 (X0: REAL; EPS : REAL;
                VAR K : INTEGER; KI : INTEGER) : REAL;
VAR X, Y : REAL;
BEGIN   K := 0;
  Y := X0;
  REPEAT    X := Y;
     Y := FUNCI (X);
     INC (K);
  UNTIL (ABS(X-Y) < EPS) OR ( K > KI); ITER1 := X;
END.
```

Но если заранее неизвестно: выполняются условия или нет, то в процедуру надо включить дополнительную проверку, что не намного усложнит программу:

```
FUNCTION ITER2 (X0: REAL; EPS : REAL;
                VAR K,L : INTEGER; KI : INTEGER) : REAL;
VAR X, Y, EPS1, EPS2 : REAL;
BEGIN   K := 0;
  Y := X0;    L := 0;
  X := Y;
  Y := FUNCI (X);
  K := 1;
  EPS1 := ABS (X-Y);
  REPEAT    X := Y;
     Y := FUNCI (X);
     INC (K);
     EPS2 := ABS (X-Y);
     IF EPS2 > EPS1 THEN L := 1;
     IF K > KI THEN L := 2;
     EPS1 := EPS2;
  UNTIL (EPS2 < EPS) OR ( L<>0 );
  ITER2 := X;
END.
```

Формальные параметры обеих процедур. *Входные: x*0 (тип ***real***) - начальное приближение корня; *eps* (тип ***real***) - параметр, используемый для окончания итерационного процесса; *ki* (тип ***integer***) - максимальное количество разрешенных итераций; *func* - имя внешней процедуры-функции (тип ***real***), возвращающей значение *w*(*x*). *Выходные: x*0 (тип ***real***) - приближенное значение корня, вычисленное с заданной точностью; *k* (тип ***integer***) - количество выполненных итераций; *l* (тип ***integer***) - параметр, контролирующий работу процедуры (только в *iter*2): *l* = 0 - вычисления выполнены с заданной точностью и в *x*0 находится значение корня; *l* = 1 - вычисления прерваны из-за того, что итерационный процесс стал расходиться (не выполнены условия применимости метода, сформулированные в теореме); *l* = 2 - слишком много итераций (больше, чем *ki*). Следует выбрать другое приближение, более близкое к ξ, или подобрать иную функцию *w*(*x*).

Для проверки процедуры методом итераций уточнялся корень с точностью до 0,00001 для уравнения $F(x) = 5x^3 - 20x + 3$.

Для отделения корней исследовалась производная уравнения $F'(x) = 15x^2 - 20$, корни которой легко определились аналитически: это $\pm 2/\sqrt{3}$. Определим знаки функции на интервалах

$$]-\infty; -2/\sqrt{3}\,]; [-2/\sqrt{3}\,;\ 2/\sqrt{3}\,]; [\,2/\sqrt{3}\,; +\infty[.$$

Таблица 1.11

Параметр	Характеристики интервалов				
Интервал	- 3	-2	0	+ 1	+ 2
sign (*F* (*x*))	-	+	+	-	+

Следовательно, корни расположены на интервалах [-3; -2]; [0; 1] и [1; 2]. Теперь уравнение $F(x) = 0$ следует привести к виду $x = w(x)$, что можно сделать разными способами:

1. $x = x + (5x^3 - 20x + 3)$; $\quad w_1(x) = 5x^3 - 19x + 3$;
2. $x = (5x^3 + 3)/20$; $\quad w_2(x) = (5x^3 + 3)/20$;
3. $x = \sqrt[3]{(20 \cdot x - 3)/5}$; $\quad w_3(x) = \sqrt[3]{(20 \cdot x - 3)/5}$.

Определим теперь, какая из полученных $w(x)$ подходит к использованию в итерационном процессе

$$|\omega_1'(x)| = |15 \cdot x^2 - 19| \quad \begin{cases} > 1, & [-3,2] \\ > 1, & [0,1] \\ > 1, & [1,2] \end{cases};$$

$$|\omega_2'| = |3x^2 / 4| \quad \begin{cases} > 1, & [-3,2] \\ < 1, & [0,1] \\ > 1, & [1,2] \end{cases}.$$

Методом итераций уточним корень на отрезке [0, 1], выбрав вторую форму представления функции $w(x)$. Два других корня уравнения находим по теореме Виетта. За начальное приближение возьмем $x_0 = 0{,}75$ и $q_0 = x_0 = 0{,}75$. Пользуясь формулой для вычисления погрешности, определим ε так, чтобы разность между двумя последовательными приближениями была бы заданной точности:

$$|x_n - x_{n-1}| < 0.00001(1 — 0.75) / 0.75 =$$

$$= 0.00001 — 0.25/0.75 = 0.000003,$$

т.е. когда $|x_n - x_{n-1}| < 0.000003$, то итерационный процесс можно завершить, считая, что заданная точность достигнута.

Процедура-функция, по которой вычисляют $w(x)$, имеет вид

```
FUNCTION FUNCI (X :REAL) : REAL;
BEGIN    FUNCI := (5.0*X*SQR(X) + 3) / 20.0;  END.
```

Результаты работы программы можно оформить так (табл. 1.12).

Таблица 1.12

Уточнение корня методом итераций на [0, 1]		
x_i	$\omega(x_i)$	Итерация
0.100000	0.150250	$k = 1$
0.150250	0.150848	$k = 2$
0.150848	0.150858	$k = 3$
0.150858	0.150858	$k = 4$

Два остальных корня вычисляют по теореме Виетта: $F(x) = (x - 0.1514)(5x^2 + 0.757x - 19.8853)$, откуда x_2 и x_3 определяются аналитически.

Уточним корни на отрезках [-3; -2] и [1; 2] любым изученным ранее методом, например дихотомии или комбинированным. Используя программы из п. 2.2 и 2.3, получаем оставшиеся два корня (табл. 1.13).

Таблица 1.13

Интервал отрезка					
[-3; -2]			[1; 2]		
Метод хорд	Метод касат.	Итерация	Метод хорд	Метод касат.	Итерация
-2.057690	2.373913	$k = 1$	1.912281	1.925000	$k = 1$
-2.068692	2.119586	$k = 2$	1.920268	1.920317	$k = 2$
-2.071076	2.072719	$k = 3$	1.920299	1.920299	$k = 3$
-2.071157	2.071159	$k = 4$	1.920299	1.920299	$k = 4$
-2.071157	2.071157	$k = 5$			
$x = -2.071157$; $F(x) = 0.0000039561$; $k = 5$			$x = 1.920299$; $F(x) = -0.0000000159$; $k = 4$		

2.6. ПРИБЛИЖЕННОЕ РЕШЕНИЕ УРАВНЕНИЯ $F(x) = 0$ МЕТОДОМ ПАРАБОЛ

Если в методе Ньютона (1.19) вместо производной вычислять разность первого порядка, то это можно рассматривать как замену функции $F(x)$ интерполяционным многочленом первой степени, построенным по узлам x_n, x_{n-1}. Но по трем последним итерациям можно построить интерполяционный многочлен второй степени, т.е. заменить график функции $F(x)$ параболой. Если воспользуемся вторым интерполяционным многочленом Ньютона, то приравняв его к нулю, получим квадратное уравнение:

$$az^2 + bz + c = 0,$$

где $z = x - x_n$; $a = F(x_n, x_{n-1}, x_{n-2})$; $b = a(x - x_n) + F(x_n, x_{n-1})$; $c = F(x_0)$.

Тогда, решив уравнение, выбирают меньший по модулю корень, который будет определять новое приближение: $x_{n+1} = x_n + z$.

Очевидно, что для начала расчетов надо определить первые три приближения x_0, x_2 и x_3. Обычно задают любые три числа из отрезка, на котором ищется корень, но можно определить эти значения и иначе.

Метод парабол относится к трехшаговым методам. По сравнению с ранее изученными он сходится гораздо медленнее, но имеет следующее достоинство: даже если все предыдущие приближения действительны, этот метод может привести к комплексным числам. Во всех же остальных методах для сходимости к комплексному корню требуется задание комплексного начального приближения. Кроме того, метод парабол исключительно эффективен для нахождения всех корней многочлена высокой степени. Интересно заметить, что если $F(x)$- некоторый многочлен

высокой степени, то, хотя сходимость метода не доказана [Касаткин, 1978] при произвольном начальном приближении, на практике итерации *всегда* сходятся к какому-либо корню. По теореме Горнера для алгебраического многочлена частное $F(x)/(x - x_n)$ будет тоже многочленом, но меньшей степени. Поэтому, последовательно удаляя найденные корни, можно найти все корни многочлена.

Один из лучших имеющихся машинных алгоритмов для нахождения действительного нуля функции сочетает в себе безотказность метода бисекции с асимптотической скоростью метода парабол в случае гладких функций. Он называется ZEROIN и был изобретен в 1960-х гг. в Математическом центре Амстердама Ван Вейнгарденом, Деккером и др. Описание алгоритма и его анализ даны в публикации Уилкинсона (1967, с. 8-12). Сам алгоритм впервые был опубликован в 1969 г. Деккером, а в 1973 г. улучшен Брентом. Реализация программы на языке *FORTRAN* была предложена Форсайтом в 1980 г. по алгоритму Деккера в версии Брента. Здесь же предлагается реализация алгоритма на языке *PASCAL*, выполненная авторами пособия.

Подпрограмма оформлена как процедура-функция ZEROIN. Все названия переменных по сравнению с фортранной версией сохранены. Обращение к процедуре-функции имеет простой вид:

$$zz := zeroin\ (a,\ b,\ tol).$$

Здесь *a* и *b* (тип ***real***) - точки интервала, на котором ищется нуль, *tol* (тип ***real***) - граница погрешности вычисления результата. Процедура ZEROIN обращается также к процедуре-функции FUNC(*x*) (тип ***real***), которая возвращает значение функции в заданной точке. Без проверки в ZEROIN предполагается, что FUNC(*a*) и FUNC(*b*) имеют разные знаки.

В процедуре ZEROIN выполняется итерационный процесс, в котором на каждом шаге присутствуют три абсциссы *a, b* и *c*. Обычно абсцисса *a* - предыдущее приближение (x_{n-1}); *b* - последнее и наилучшее приближение x_n; *c* - предыдущее, но еще более раннее приближение, причем sign(FUNC(*b*)) = sign(FUNC(*c*)). Таким образом, *b* и *c* ограничивают нуль. При этом |FUNC(*b*)|< |FUNC(*c*)|. Если |*b* - *c*| уменьшилось настолько, что выполняется условие

$$|b - c| < tol + 4.0 \cdot eps \cdot \mathrm{abs}(b),$$

то *b* выдается как найденное значение процедуры-функции ZEROIN. Кроме параметра *tol* в проверке сходимости участвует еще один параметр - *eps*, "машинный нуль", чтобы подстраховаться на случай, если *tol* задано слишком маленьким, например равным нулю.

На каждой очередной итерации в ZEROIN выбирается очередное приближение к корню из двух - один получен алгоритмом бисекции, а другой - интерполяцией. Если *a, b* и *c* различны, то используется квадратичная интерполяция (метод парабол), если нет, то метод секущих (линейная интерполяция). Если полученное x_{n+1} находится в "разумных" пределах, то выбирают его, в противном случае выбирают точку бисекции. Определение "разумности" означает, что $x_{n+1} \in [b,\ c]$, т.е. лежит внутри интервала и не совпадает с его концами. Таким образом, длина интервала гарантированно убывает с каждой итерацией, а если функция еще к тому же ведет себя хорошо, то и достаточно быстро. Более подробно об алгоритме, программе и описании метода можно посмотреть в книге Брента (1973).

Еще несколько важных замечаний по поводу алгоритма. Для того чтобы точка бисекции всегда лежала внутри расчетного интервала и желаемая величина получалась как малая поправка к найденному приближению, ее вычисляют по формуле $xm := b + 0.5*(c - b)$.

Большое внимание в алгоритме уделяется проблеме машинных нулей. Это выражается в необычной проверке знаков FUNC(*b*) и FUNC(*c*) (см. соответствующие подпрограммы).

Итоговая точка, получаемая алгоритмом, записывается в виде $b + p/q$, но деление не выполняется до тех пор, пока оно не станет безопасным и действительно необходимым, так как если будет выбрана точка по методу бисекций, то деление вообще не нужно.

Версия алгоритма Брента, во-первых, всегда сходится, даже при плавающей арифметике. Во-вторых, количество итераций не может стать больше числа

$$\left[\log_2\left(\frac{B - A}{tol1}\right)\right]^2,$$

где $tol1 := 0.5*tol + 2.0*eps*\mathrm{abs}(B)$. В-третьих, нуль *R*, получаемый алгоритмом, таков, что FUNC гарантированно меняет знак в определенном интервале, приблизительно совпадающим с

$$[R - \exp(tol*\ln 2),\ \ R + \exp(tol*\ln 2)].$$

В-четвертых, сам Брент тщательно проверял свой алгоритм на самых разнообразных функциях и установил, что в типичном случае для гладкой функции требуется не более 10 итераций. В общем случае Брент ни разу не встретил функции, для которой количество итераций было больше трехзначного числа. Более подробно смотрите работу Брента (1973).

Формальные параметры процедуры. *Входные: ax, bx* (тип ***real***) - определяют отрезок, на котором ищется корень уравнения; *tol* (тип ***real***) - точность, с которой должен быть определен корень уравнения. *Выходные:* процедура-функция возвращает найденное с заданной точностью значение корня на отрезке [*ax*, *bx*].

Описанная процедура выглядит:

```
FUNCTION ZEROIN (AX, BX, TOL : REAL) : REAL;
LABEL 20, 30;
VAR A,B,C,D,E,EPS,FA,FB,FC,TOL1,XM,P,Q,R,S:REAL;
      DN : BOOLEAN;
BEGIN  EPS := 1.0;
  DN := FALSE;
  REPEAT
     EPS := EPS / 2.0;
     TOL1 := 1.0 + EPS;
  UNTIL TOL1 <= 1.0;
```

```
  A := AX;
  B := BX;
  FA := FUNC (A);
  FB := FUNC (B);
20: C := A;
  FC := FA;
  D := B - A;
  E := D;
  REPEAT
    IF ABS (FC) < ABS (FB) THEN
    BEGIN              A := B;
      B := C;          C := A;
      FA := FB;        FB := FC;
      FC := FA;
    END;
                {*** ???????? ?????????? ***}
    TOL1 := 2.0*EPS*ABS(B) + 0.5*TOL;
    XM := 0.5 * (C-B);
    IF ABS (XM) <= TOL1 THEN DN := TRUE
    ELSE
      IF FB=0.0 THEN DN := TRUE
      ELSE
      BEGIN
        {*** ?????? ? ????????????? ???????? ***}
        IF (ABS(E)>=TOL1) AND
           (ABS(FA)>ABS(FB)) THEN
        BEGIN
         {*** ???????? ?? ?????????. ???????????? ***}
          IF A<>C THEN
          BEGIN
            Q := FA / FC;       R := FB / FC;
            C := FB / FA;
            P:= S*(2.0*XM*Q*(Q-R)- (B-A)*(R-1.0));
            Q := (Q-1.0)*(R-1.0)*(S-1.0);
          END
          ELSE
          BEGIN S := FB / FA;
            P := 2.0*XM*S;
            Q := 1.0 - S;
          END;
            {*** ????? ????? ??? Q ***}
        IF P > 0.0 THEN Q := -Q;
        P := ABS (P);
      {** ?????? ? ???????????? ?????? ???????????? **}
        IF ((2.0*P) < (3.0*XM*Q-ABS(TOL1*Q)))
           AND (P < (ABS(0.5*E*Q))) THEN
        BEGIN                  E := D;
          D := P/Q;            GOTO 30;
        END;
      END;
      D := XM;          E := D;
30:   A := B;
      FA := FB;
      IF ABS(D) > TOL1 THEN B := B + D;
      IF ABS(D) <= TOL1 THEN
            B := B + SIGN(TOL1)*XM;
          FB := FUNC (B);
          IF (FB*(FC / ABS(FC))) > 0.0 THEN GOTO 20;
        END;
    UNTIL DN;
    ZEROIN := B;
END.
```

Процедура-функция проверялась на примере решения уравнения из п. 2.5 (см. табл. 1.11 и 1.12). Результаты работы данной программы приводятся ниже:

на отрезке [0, 1]:

$$x = 0.150858;\ F(x) = -0.0000000000;\ i = 5.$$

на отрезке [-3,-2]:

$$x = -2.071157;\ F(x) = -0.0000000001;\ i = 7.$$

на отрезке [2, 3]:

$$x = 1.920299;\ F(x) = 0.0000000001;\ i = 7.$$

2.7. МЕТОДЫ УСКОРЕНИЯ СХОДИМОСТИ

В применении к некоторым уравнениям рассмотренные итерационные методы сходятся довольно медленно. В таких случаях особое значение приобретают способы, позволяющие оптимизировать итерационный процесс путем ускорения сходимости метода итераций. Рассмотрим два наиболее часто употребляющихся алгоритма.

Метод кратного корня. Ускорение сходимости можно получить, если подобрать правую часть уравнения (1.15) так, что

$$F'(\xi) = F''(\xi) = \ldots = F^{(m-1)}(\xi) = 0,\ F^{(m)}(\xi) \neq 0,$$

где ξ - корень уравнения.

Тогда для $m = 3$ итерационный процесс будет определяться формулой

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{f''(x_n)\cdot f^2(x_n)}{2\cdot f'^3(x_n)}, \qquad (1.20)$$

т.е. будет отличаться от уравнения (1.19) последним членом. Следовательно, процедура метода касательных, рассмотренная в п. 2.3, должна быть дополнена соответствующим блоком, в котором производится вычисление этого члена.

Другой вариант - использование на каждом итерационном шаге процедуры-функции ACCEL1. Значение и тип параметров ясны из самого текста процедуры-функции ACCEL1, которая, в свою очередь, использует функции, возвращающие значения

$$F(x_n),\ F'(x_n) \text{ и } F''(x_n).$$

```
FUNCTION ACCEL1(X:REAL): REAL;
VAR F,D1,D2:REAL;
BEGIN
{ *** ??????? FUNC, DER1, DER2 ?????????
      ?????? X ***}
  F:=FUNC(X);
  D1:=DER1(X);
```

```
  ACCEL1:=X-F/D1-DER2(X)*F*F/(2*D1 3)
END.
```

Метод Эйткена - Стеффенсона. Если функция $F(x)$ непрерывна и трижды непрерывно дифференцируема на отрезке $[a, b]$, причем $|F'(x)| < 1$, то сходимость итерационного процесса может быть значительно увеличена, если использовать следующий алгоритм. На каждом шаге вычисления проводятся в три этапа: находим $x'_n = F(x_n)$, затем на основе x'_n рассчитываем $x''_n = F(x'_n)$ и далее по трем значениям x_n, x'_n, x''_n определяем приближение x_{n+1} по следующей формуле:

$$x_{n+1} = \frac{x_n \cdot x''_n - [x'_n]^2}{x''_n - 2 \cdot x'_n + x_n} = \frac{x_n \cdot f[f(x_n)] - f^2(x_n)}{f[f(x_n)] - 2 \cdot f(x_n) + x_n}. \quad (1.21)$$

Итерации заканчиваются, когда знаменатель становится близким к нулю. Описанный алгоритм вычисления следующего приближения уже по имеющемуся реализован в процедуре-функции ACCEL2. Значение и тип параметров ясны из текста процедуры-функции ACCEL2, которая, в свою очередь, обращается к функции, возвращающей значение $F(x)$.

```
FUNCTION ACCEL2(X:REAL) : REAL;
VAR F1,F2 : REAL;
BEGIN
{ ** FUNC □□□□□□□□□ □□□□□□□□ □□□□□□□ □ □□□□□ X **}
  F1:=FUNC(X);
  F2:=FUNC(F1);
  ACCEL2:=(X*F2-F1*F1)/(F2-2*F1+X)
END; { *** ACCEL2 ***} .
```

Сравнительные расчеты, выполненные с использованием процедур ACCEL1, ACCEL2 (по формулам 1.20 и 1.21) и NEWT, реализующей метод Ньютона (1.19), показали, что при использовании "ускоряющих" процедур для некоторых функций удается добиться ускорения сходимости в 3 - 5 раз.

§ 3. МЕТОДЫ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ И НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Система m линейных алгебраических уравнений с n неизвестными в общем виде может быть записана следующим образом:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1; \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2; \\ \ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \end{cases} \quad (1.22)$$

или в матричном виде $\boldsymbol{AX} = \boldsymbol{B}$, где $\boldsymbol{A}$ - прямоугольная матрица размерности $m \times n$, $\boldsymbol{X}$ - вектор n-го порядка, $\boldsymbol{B}$ - вектор m-го порядка. *Решением системы* (1.22) называется такая упорядоченная совокупность чисел $x_1 = c_1, x_2 = c_2, \ldots, x_n = c_n$, которая обращает все уравнения системы в верные равенства. Две системы называются *эквивалентными* (*равносильными*), если множества их решений совпадают.

Система линейных уравнений называется *совместной*, если она имеет хотя бы одно решение, и *несовместной* - в противном случае. Совместная система называется *определенной*, если она имеет единственное решение, и *неопределенной* - в противном случае. Система является определенной, если rang $\boldsymbol{A}$ = rang $\boldsymbol{B}$, где матрица $\boldsymbol{B}$, полученная из матрицы $\boldsymbol{A}$ добавлением столбца свободных членов, называется *расширенной*.

Если матрица $\boldsymbol{A}$ - квадратная и $\det \boldsymbol{A} \neq 0$, то она называется *неособенной* (*невырожденной*), при этом система уравнений, имеющая неособенную матрицу $\boldsymbol{A}$, совместна и имеет единственное решение.

Если уравнения (1.22) являются нелинейными относительно неизвестного вектора $\boldsymbol{X}$, то соответствующая система, записанная в векторной форме

$$f(x) = 0, \quad (1.23)$$

называется *системой нелинейных уравнений*. Она может быть также представлена в координатном виде:

$$f_k(x_1, x_2, \ldots, x_n) = 0, \quad 1 < k < n.$$

Многообразие численных методов решения систем линейных алгебраических уравнений можно разделить на два класса: *прямые* (или *точные*) и *итерационные* (или *приближенные*) методы. В настоящем параграфе рассматриваются наиболее эффективные алгоритмы, реализующие ряд методов из обоих классов. Однако заметим, что нелинейные системы решают только итерационными методами, один из которых (метод Ньютона) рассматривается в настоящем параграфе.

3.1. МЕТОД ИСКЛЮЧЕНИЯ ГАУССА ДЛЯ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ

Из курса линейной алгебры [Крылов и др., 1972; Курош, 1962; Фадеев, Фадеева, 1963 и др.] известно, что решение системы линейных уравнений можно просто найти по *правилу Крамера* - через отношение определителей. Но этот способ не очень удобен для решения систем уравнений с числом неизвестных > 5, т.е. когда найти определитель сложно, а при числе неизвестных > 10 нахождение определителя с достаточно высокой степенью точности

становится самостоятельной вычислительной задачей. В этих случаях применяют иные методы решения, среди которых самым распространенным является *метод Гаусса.*

Запишем систему линейных уравнений (1.22) в виде

$$\sum_{i=1}^{n} a_{ij} \cdot x_j = b_i, 1 \le i \le n. \qquad (1.24)$$

Если матрица системы верхняя треугольная, т.е. ее элементы ниже главной диагонали равны нулю, то все x_j можно найти последовательно, начиная с x_n, по формуле

$$x_j = \frac{1}{a_{jj}} \left(b_j - \sum_{k=j+1}^{n} a_{jk} \cdot x_k \right), j = n,\ n-1,\ \ldots,\ 1. \qquad (1.25)$$

При $j > k$ и $a_{jj} \neq 0$ этот метод дает возможность найти решение системы.

Метод Гаусса для произвольной системы (1.22) основан на приведении матрицы $\boldsymbol{A}$ системы к верхней или нижней треугольной. Для этого вычитают из второго уравнения системы первое, умноженное на такое число, при котором $a_{21} = 0$, т.е. коэффициент при x_1 во второй строке должен быть равен нулю. Затем таким же образом исключают последовательно a_{31}, a_{41}, ..., a_{m_1}. После завершения вычислений все элементы первого столбца, кроме a_{11}, будут равны нулю.

Продолжая этот процесс, исключают из матрицы $\boldsymbol{A}$ все коэффициенты a_{ij}, лежащие ниже главной диагонали.

Построим общие формулы этого процесса. Пусть исключены коэффициенты из k - 1 столбца. Тогда ниже главной диагонали должны остаться уравнения с ненулевыми элементами:

$$\sum_{i=1}^{n} a_{ij}^{(k)} x_j = b_j^{(k)}, k \le j \le n.$$

Умножим k-ю строку на число

$$c_{mn} = a_{mk}^{(k)} \Big/ a_{kk}^{(k)}, \quad (m > k)$$

и вычтем ее из m-й строки. Первый ненулевой элемент этой строки обратится в нуль, а остальные изменятся по формулам

$$\begin{cases} a_{ml}^{(k+1)} = a_{ml}^{(k)} - c_{ml} a_{kl}^{(k)}; \\ b_m^{(k+1)} = b_m^{(k)} - c_{ml} b_k^{(k)}, k < m, l \le n. \end{cases} \qquad (1.26)$$

Произведя вычисления по формулам (1.26) при всех указанных индексах, исключим элементы k-го столбца. Такое исключение назовем *циклом*, а выполнение всех циклов назовем *прямым ходом* исключения.

После выполнения всех циклов получим верхнюю треугольную матрицу $\boldsymbol{A}$ системы (1.24), которая легко решается обратным ходом по формулам (1.25). Если при прямом ходе коэффициент a_{jj} оказался равен нулю, то перестановкой строк перемещаем на главную диагональ ненулевой элемент и продолжаем расчет.

Если предположить, что $a_{jj} \neq 0$, то тогда можно применить подпрограмму GAUS, решающую систему из n линейных уравнений. Это одна из подпрограмм, традиционно использующаяся в библиотеках стандартных программ для ЕС-ЭВМ. Ее перевод на язык *PASCAL* выполнен авторами.

Формальные параметры процедуры. *Входные: n* (тип ***integer***) - порядок системы; *a* (тип ***real***) - массив коэффициентов системы размером $n \times n$; *b* (тип ***real***) - массив-строка свободных членов. *Выходные: x* (тип ***real***) - массив-строка, в который помещается решение системы.

```
PROCEDURE GAUS (N:INTEGER; A : MAS; B : MAS1;
                VAR X : MAS1);
TYPE MST = ARRAY [1..N] OF REAL;
     MSS = ARRAY [1..N] OF MST;
VAR A1 : MSS; B1 : MST;
    I, J, M, K : INTEGER;   H : REAL;
BEGIN
  FOR I := 1 TO N DO
  BEGIN
     B1[I] := B[I];
     FOR J := 1 TO N DO
        A1 [I,J] := A[I,J];
  END;
  FOR I := 1 TO N-1 DO
     FOR J := I+1 TO N DO
     BEGIN
        A1[J,I] :=- A1[J,I] / A1[I,I];
        FOR K := I+1 TO N DO
           A1[J,K] := A1 [J,K] + A1[J,I]*A1[I,K];
        B1[J] := B1[J] + B1[I]*A1[J,I];
     END;
  X[N] := B1[N] / A1[N,N];
  FOR I := N-1 DOWNTO 1 DO
  BEGIN
     H := B1[I];
     FOR J := I+1 TO N DO
        H := H - X[J]*A1[I,J];
     X[I] := H / A1[I,I];
  END;
END.
```

Если контроль за невырожденностью матрицы $\boldsymbol{A}$ необходим, то можно предложить воспользоваться другой процедурой GAUS1.

В отличие от первой процедуры здесь в выходных параметрах есть переменная *tol*, возвращающая 0 при нормальном завершении работы процедуры, или 1, если на главной диагонали один из элементов равен 0, или 2, если матрица $\boldsymbol{A}$ размерностью больше, чем 50×50.

```
PROCEDURE GAUS1 (N:INTEGER; A : MAS; B : MAS1;
                 VAR X : MAS1; VAR TOL : INTEGER);
TYPE MST = ARRAY [1..50] OF REAL;
     MSS = ARRAY [1..50] OF MST;
VAR A1 : MSS; B1 : MST;
```

```
        I, J, M, K : INTEGER;    H : REAL;
BEGIN
  FOR I := 1 TO N DO
  BEGIN
     B1[I] := B[I];
     FOR J := 1 TO N DO   A1 [I,J] := A[I,J];
  END;
  TOL := 0;
  IF N > 50 THEN
  BEGIN
     TOL := 2;
     EXIT;
  END;
  FOR I := 1 TO N-1 DO
     FOR J := I+1 TO N DO
     BEGIN
        IF ABS(A1[I,I])< 1.0E-10 THEN
        BEGIN
           TOL := 1;
           EXIT;
        END;
        A1[J,I] :=- A1[J,I] / A1[I,I];
        FOR K := I+1 TO N DO
           A1[J,K] := A1 [J,K] + A1[J,I]*A1[I,K];
        B1[J] := B1[J] + B1[I]*A1[J,I]
     END;
  IF ABS(A1[N,N]) < 1.0E-10 THEN
  BEGIN
     TOL := 1;
     EXIT;
  END;
  X[N] := B1[N] / A1[N,N];
  FOR I := N-1 DOWNTO 1 DO
  BEGIN
     IF ABS(A1[I,I]) < 1.0E-10 THEN
     BEGIN
        TOL := 1;
        EXIT;
     END;
     H := B1[I];
     FOR J := I+1 TO N DO   H := H - X[J]*A1[I,J];
     X[I] := H / A1[I,I];
  END;
END.
```

По поводу приведенных алгоритмов можно сказать, что они не самые эффективные и пригодны для решения систем, имеющих невырожденные матрицы $\boldsymbol{A}$ и $\boldsymbol{B}$ с рангом не более 50, составленные из приблизительно одинаковых коэффициентов. Если между наибольшим и наименьшим из коэффициентов расстояние более чем 10^4, то эти алгоритмы применять не рекомендуется, так как погрешность вычислений будет такова, что может либо привести к вырожденной матрице A, либо дать в результате вектор X с большой вычислительной погрешностью.

Для проверки работы процедур решим систему линейных уравнений методом Гаусса с точностью до 0.0001:

$$\left.\begin{aligned} 0.68x_1 + 0.05x_2 - 0.11x_3 + 0.08x_4 &= 2.15;\\ 0.21x_1 - 0.13x_2 + 027x_3 - 0.80x_3 &= 0.44;\\ 0.11x_1 + 0.84x_2 - 0.28x_3 - 0.06x_4 &= 0.83;\\ -0.08x_1 + 0.15x_2 - 0.5x_3 - 0.12x_4 &= 1.16. \end{aligned}\right\}$$

Коэффициенты, промежуточные результаты и окончательное решение приводятся в табл. 1.14. Подобные таблицы удобно использовать для ручного счета и проверки хода решения.

Таблица 1.14

Коэффициенты при неизвестных				Свободные	Строчные
x_1	x_2	x_3	x_4	члены	суммы
0.68	0.55	-0.11	0.88	2.15	2.85
0.21	-0.13	0.27	0.80	0.44	-0.01
0.11	0.84	0.28	-0.06	0.83	1.44
-0.08	0.15	0.50	0.12	1.16	0.61
1	**0.0735**	**-0.1618**	**0.1176**	**3.1618**	**4.1912**
	-0.1454	0.30398	-0.8247	-0.22398	-0.88015
	-0.18319	0.26220	0.0729	-0.48220	-0.97847
	0.15590	-0.51290	-0.1106	1.41290	0.94530
	1	**-2.09060**	**5.6719**	**1.54040**	**6.1217**
		-1.47643	4.79139	0.79920	4.1136
		-0.18697	-0.99480	1.17230	-0.0095
		1	**-3.24410**	**-0.54110**	**-2.7851**
			-1.60130	1.07110	-0.5302
			1	**-0.66890**	**0.3311**
2.8264	-0.3337	-2.7110	-0.6689	: РЕШЕНИЕ	

3.2. МЕТОД ГЛАВНЫХ ЭЛЕМЕНТОВ

На практике применяют множество вариантов вычислительных схем метода Гаусса. Например, при приведении матрицы к верхней треугольной форме выбирают наибольший элемент (в строке или столбце), уменьшая вычислительную погрешность за счет деления на не самый маленький элемент. Такая вычислительная схема называется *методом Гаусса с выбором ведущего элемента.* Если же выбирать при приведении матрицы самый большой (по модулю) элемент из всех оставшихся, то такая схема будет называться *методом Гаусса с выбором главного элемента.* Последняя схема относится к наиболее популярным. Главное ее отличие от метода Гаусса, рассмотренного в п. 3.1, состоит в том, что при приведении матрицы ***A*** к верхней (или нижней) треугольной форме ее строки и столбцы переставляют так, чтобы наибольший из всех оставшихся элементов матрицы стал ведущим, и на него выполняется деление. Если матрица хорошо обусловлена, то в методе Гаусса с выбором главного элемента погрешности округления невелики.

Описанный алгоритм, как и алгоритм метода Гаусса, представленный в п. 3.1, имеет множество программных реализаций и всегда есть в библиотеках стандартных программ. Один из относительно эффективных алгоритмов есть в БСП [1] на ЕС ЭВМ для транслятора *FORTRAN-77*. Он реализован в виде процедуры SINQ. Перевод на язык *PASCAL* выполнен авторами.

Формальные параметры процедуры. *Входные: n* (тип ***integer***) - целое положительное число, равное порядку исходной системы; *aa* (тип ***real***) - массив из $n \times n$ действительных чисел, содержащий матрицу коэффициентов системы ($aa[1] = a_{11}$; $aa[2] = a_{21}$; $aa[3] = a_{31}$; ...; $aa[n] = a_{n1}$; $aa[n+1] = a_{12}$; $aa[n+2] = a_{22}$; ...; $aa[2*n] = a_{n2}$; ...; $aa[n*n] = a_{nn}$); *b* (тип ***real***) - массив из *n* действительных чисел, содержащий столбец свободных членов исходной системы ($b[1] = b_1$; $b[2] = b_2$; ...; $b[n] = b_n$). *Выходные: b* (тип ***real***) - массив из *n* действительных чисел (он же был входным) при выходе из подпрограммы, содержащий решения исходной системы ($b[1] = x_1$; $b[2] = x_2$; ...; $b[n] = x_n$); *ks* (тип ***integer***) - признак правильности решения или код ошибки: если *ks* = =0, то в массиве *b* содержится решение исходной системы; если *ks* = 1, то исходная система не имеет решения (главный определитель ее равен нулю).

```
PROCEDURE  SINQ (VAR A: MAS11; VAR B : MAS1;
                 VAR KS : INTEGER; N : INTEGER);
VAR TOL, BIGA, AA, SAVE : REAL;
    JJ,JY,IJ,IT,J,I, I1, IMAX, K, IQS, IXJ, IXJX : INTEGER;
    JJX, NY, IA, IB, IO, I2, IX, JX, KY : INTEGER;
LABEL CONT10;
BEGIN    TOL := 0.0;
    KS := 0;
    JJ := -N;
    FOR J := 1 TO N DO
    BEGIN      JY  := J + 1;
       JJ := JJ + N + 1;
       BIGA := 0.0;
       IT := JJ - J;
       FOR I := J TO N DO
       BEGIN
          IJ := IT + I;
          AA := ABS (A[IJ]);
          IF (ABS(BIGA)-AA) < 0.0 THEN
          BEGIN
             BIGA := A[IJ];
             IMAX := I; END;
          END;
          IF (ABS(BIGA)-TOL) <= 0.0 THEN
          BEGIN
             KS := 1;
             EXIT;
          END;
          I1 := J + N*(J-2);
          IT := IMAX - J;
          FOR K := J TO N DO
          BEGIN
             INC (I1,N);
             I2:= I1 + IT;
             SAVE := A[I1];
             A[I1] := A[I2];
             A[I2] := SAVE;
             A[I1] := A[I1] / BIGA;
          END;
          SAVE := B[IMAX];
          B[IMAX] := B[J];
          B[J] := SAVE / BIGA;
          IF J=N THEN GOTO CONT10;
          IQS := N*(J-1);
          FOR IX := JY TO N DO
          BEGIN
             IXJ := IQS + IX;
             IT := J - IX;
             FOR JX := JY TO N DO
             BEGIN
                IXJX := N*(JX-1) + IX;
                JJX := IXJX + IT;
                A[IXJX] := A[IXJX] - A[IXJ]*A[JJX];
             END;
             B[IX] := B[IX] - B[J]*A[IXJ];
          END;
       END;
CONT10:    NY := N - 1;
       IT := N*N;
       I := 1;
       J :=1;
       FOR KY := 1 TO NY DO
```

[1] Библиотека стандартных программ. Поставляется вместе с транслятором языка.

```
      BEGIN  IA := IT - KY;
          IB := N - KY;
          IO := N;
          FOR K := 1 TO KY DO
          BEGIN
              B[IB] := B[IB] - A[IA]*B[IO];
              DEC (IA,N);
              DEC (IO);
          END;
      END;
END.
```

Для проверки процедуры и сравнения с работой процедур в п. 3.1 решалась система уравнений, приведенная в качестве примера в п. 3.1. Вычисления проводились с точностью до 10^{-5}. Результаты работы программы приведены далее.

Исходная матрица A				**Матрица** B
0.680000	0.050000	-0.110000	0.080000	2.150000
0.210000	-0.130000	0.270000	-0.800000	0.440000
-0.110000	-0.840000	0.280000	0.060000	-0.830000
-0.080000	0.150000	-0.500000	-0.120000	1.160000
Преобразованная матрица A				**Матрица** B
1.000000	0.210000	-0.110000	-0.080000	3.161765
0.000000	1.000000	-0.145441	0.155882	0.579636
0.000000	0.000000	1.000000	0.258130	-2.851572
0.000000	0.000000	0.000000	1.000000	-0.669070

РЕШЕНИЕ СИСТЕМЫ

2.826351 -0.333733 -2.711759 -0.669070 .

3.3. РЕШЕНИЕ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ НЬЮТОНА

Очень распространенной является вычислительная задача нахождения некоторых или всех решений системы (1.23) из n нелинейных алгебраических или трансцендентных уравнений с n неизвестными.

Обозначим через $\boldsymbol{X}$ вектор-столбец $(x_1, x_2, ..., x_n)^T$ и запишем систему уравнений в виде формулы (1.23) $\boldsymbol{F}(\boldsymbol{X}) = 0$, где $\boldsymbol{F} = (f_1, f_2, ..., f_n)^T$.

Подобные системы уравнений могут возникать непосредственно, например при конструировании физических систем, или опосредованно. Так, к примеру, при решении задачи минимизации некоторой функции $G(x)$ часто необходимо определить те точки, в которых градиент этой функции равен нулю. Полагая $\boldsymbol{F}$ = grad G, получаем нелинейную систему.

Основная *идея метода Ньютона* состоит в *выделении из уравнений линейных частей*, которые являются главными при малых приращениях аргументов. Это позволяет свести исходную задачу к решению последовательности линейных систем. При решении единственного уравнения (n=1) получаем рассмотренный в п. 2.3 метод Ньютона (1.19).

Метод Ньютона для n уравнений применим только тогда, когда могут быть вычислены все частные производные функций f_i по переменным x_i. Пусть $\boldsymbol{J}(x)$ обозначает матрицу Якоби и ее (i, j)-й элемент есть значение производной $\partial f_i / \partial x_i$ в точке x_i. Как и в одномерном случае (n = 1), метод Ньютона начинается с произвольного $\boldsymbol{X}$, обозначенного $\boldsymbol{X}^0$. Далее $\boldsymbol{F}$ линеаризуют для $\boldsymbol{X}^0$, разлагая его в ряд Тейлора и учитывая лишь первые члены:

$$\boldsymbol{F}(\boldsymbol{X}) = \boldsymbol{F}(\boldsymbol{X}^0) + \boldsymbol{J}(\boldsymbol{X}^0)(\boldsymbol{X} - \boldsymbol{X}^0) + \ ...$$

Линейное приближение к $\boldsymbol{F}$ около $\boldsymbol{X}^0$ в операторной форме задается

$$\boldsymbol{L}(\boldsymbol{X}) = \boldsymbol{F}(\boldsymbol{X}^0) + \boldsymbol{J}^0(\boldsymbol{X} - \boldsymbol{X}^0), \quad \text{где } \boldsymbol{J}^0 = \boldsymbol{J}(\boldsymbol{X}^0).$$

Чтобы найти следующее приближение $\boldsymbol{X}$ к решению системы $\boldsymbol{F}(\boldsymbol{X}) = 0$, решают уравнение $\boldsymbol{F}(\boldsymbol{X}^0) + \boldsymbol{J}^0(\boldsymbol{X}^1 - \boldsymbol{X}^0) = 0$. Естественно, решение можно также записать в форме $\boldsymbol{X}^1 = \boldsymbol{X}^0 - (\boldsymbol{J}^0)^{-1} \boldsymbol{F}(\boldsymbol{X}^0)$, сильно напоминающей одномерную формулу метода Ньютона.

Однако для большинства систем из n уравнений с n неизвестными вычисление обратной матрицы $(\boldsymbol{J}^0)^{-1}$ не является необходимым, а наоборот, предпочтительнее решать линейную систему относительно поправки $\boldsymbol{X}^1 - \boldsymbol{X}^0$.

В общем случае, имея $\boldsymbol{X}^k$, можно найти $\boldsymbol{X}^{k+1}$ прибавлением к $\boldsymbol{X}^k$ поправки $\boldsymbol{X}^{k+1} - \boldsymbol{X}^k$, полученной из решения линейной системы

$$\boldsymbol{F}(\boldsymbol{X}^k) + \boldsymbol{J}^k(\boldsymbol{X}^{k+1} - \boldsymbol{X}^k) = 0. \qquad (1.27)$$

Сходимость итерационного процесса (1.27) доказывается теоремой Дж.Форсайта и др. (1980), которую сформулируем неформально.

Пусть r - решение системы $\boldsymbol{F}(\boldsymbol{X}) = 0$ такое, при котором $\boldsymbol{J}(n)$ не вырождена и вторые частные производные функции $\boldsymbol{F}$ непрерывны вблизи r. Тогда, если $\boldsymbol{X}^0$ достаточно близко к r, то ньютоновы итерации сходятся. Более того, для $e^k = x^k - r$ при $k \to \infty$ отношение

$$\left\| e^{k+1} \right\| \Big/ \left\| e^k \right\|^2$$

ограничено. Сходимость процесса будет 2-го порядка.

Как и в одномерном случае, здесь основная проблема состоит в удачном выборе начального приближения, которое желательно было бы выбрать достаточно близко к предполагаемому корню, чтобы могла начаться быстрая сходимость.

На практике такое приближение достигается или очень большим везением (удачно выбран $\boldsymbol{X}^0$), или мужеством и настойчивостью исследователя (выполняется очень много итераций до того, как процесс начнет быстро сходиться).

Еще одно замечание по поводу матрицы Якоби. Если вычисление производной функции уже в одномерном случае бывает более сложной задачей, чем отыскание значения самой функции, то для системы n уравнений вычисление $\boldsymbol{F}'(\boldsymbol{X})$ во много раз более трудоемко, чем вычисление $\boldsymbol{F}(\boldsymbol{X})$ (не забывайте, что это матрицы, а не просто функции !).

Попытки избежать перечисленные трудности превратились в отдельную вычислительную задачу. Прямое обобщение метода секущих оказалось неудовлетворитель-

ным, так как приближения к ***J***(***X***), получающиеся как *n*-мерный аналог метода секущей (метод хорд), часто оказываются вырожденными. Популярны так называемые квазиньютоновские методы, которые начинаются с очень трудных начальных итераций, но по мере приближения к решению точность их возрастает. Эти методы широко используют в задачах многомерной оптимизации.

Полная информация по рассмотренным методам имеется в работах Островского (1963) и Ортега, Рейнболда (1975), последнюю можно считать полным справочником по методам решения систем нелинейных уравнений.

В данной работе можно воспользоваться подпрограммой ZEROIN, подробно описанной в п. 2.6 с небольшими изменениями. Но здесь приводится другая процедура, взятая из БСП для БЭСМ (язык *FORTRAN*), переведенная на язык *PASCAL* авторами. Эту подпрограмму используют традиционно для решения систем не выше 10-го порядка, она работает достаточно эффективно и быстро.

Формальные параметры процедуры. *Входные: n* (тип ***integer***) - количество уравнений системы (совпадает с количеством неизвестных, причем $n < 10$); *x* (тип ***real***) - массив из *n* действительных чисел, содержащий перед обращением *newts* начальное приближение решения; *funcf* - имя внешней подпрограммы, при помощи которой выполняют вычисления текущих значений функции ***F*** по заданным значениям, расположенных в элементах массива *x*, и размещение их в элементах массива *y*; *funcg* - имя внешней подпрограммы, предназначенной для вычисления по заданным значениям *x* из массива {*x*} элементов матрицы $d\boldsymbol{F}/dx$, размещенной в двухмерном массиве *a* размером [*n*×*n*]; *eps* (тип ***real***) - значение ε из условия окончания итерационного процесса. *Выходные: x* (тип ***real***) - массив из *n* действительных чисел (он же входной) содержит при выходе из *newts* приближенное значение решения; *k* (тип ***integer***) - разрешенное количество итераций.

```
PROCEDURE  NEWTS (CONST N: INTEGER;
                  VAR X : ARRAY OF REAL; EPS : REAL;
                  VAR XKIT : INTEGER);
VAR Y, X1 : ARRAY [0..2] OF REAL;
       A : ARRAY [0..4] OF REAL;
       L, M : ARRAY [0..10] OF INTEGER;
       J, I, NK : INTEGER;       S, D, XX,YY : REAL;
BEGIN
   XKIT := 0;
   REPEAT
       FOR J := 1 TO N DO X1[J] := X[J];
       XX := X1[1];
       YY := X1[2];
       FUNCF (N, XX,YY, Y);
       FUNCG (N, XX,YY, A);
       MINV (A, N, D, L, M);
       FOR J := 1 TO N DO
       BEGIN
           X[J] := X1[J];
           FOR I := 1 TO N DO
           BEGIN
               NK := J + N*(I-1);
               X[J] := X[J] - A[NK]*Y[I];
           END;
       END;
       INC (XKIT);
       S := 1.0;
       FOR J := 1 TO N DO   S := S* SQR(X[J]-X1[J]);
       S := SQRT(S);
   UNTIL S > EPS;
END.
```

Внимание: подпрограмма NEWTS содержит обращение к подпрограмме MINV, которая входит в БСП БЭСМ. Полный текст подпрограммы приводится ниже (перевод на язык *PASCAL* выполнен авторами).

```
PROCEDURE MINV (VAR A : ARRAY  OF REAL;
                N : INTEGER;  VAR D : REAL;
                VAR  L, M : ARRAY OF INTEGER);
VAR NK,KK,K,IZ,IJ,I,KI,JI,JP, JK, IK, KJ, JQ : INTEGER;
       BIGA, HOLD : REAL;
       JR : INTEGER;  DN : BOOLEAN;
BEGIN    D := 1.0;
   NK := -N;
   DN := FALSE;
   FOR K := 1 TO N DO
   BEGIN      INC(NK,N);
       L[K] := K;
       M[K] := K;
       KK := NK + K;
       BIGA := A[KK];
       FOR J := K TO N DO
        BEGIN
           IZ := N*(J-1);
       FOR I := K TO N DO
           BEGIN
               IJ := IZ + I;
               IF (ABS(BIGA)-ABS(A[IJ])) < 0.0 THEN
               BEGIN
                   BIGA := A[IJ];
                   L[K] := I;
                   M[K] := J;
               END;
           END;
       END;
       J :=L[K];
       IF J > K THEN
       BEGIN
           KI := K - N;
           FOR I := 1 TO N DO
           BEGIN
               INC(KI,N);
               HOLD := - F[KI];
               JI := KI - K + J;
```

```
            A[KI] := A[JI];
            A[JI] := HOLD;
        END;
    END;
    I := M [K];
    IF I > K THEN
    BEGIN
        JP := N*(I-1);
        FOR J := 1 TO N DO
        BEGIN
            JK:= NK + J;
            JI:= JP + J;
            HOLD := - A[JK];
            A[JK] := A[JI];
            A[JI] := HOLD;
        END;
    END;
    IF ABS(BIGA)< 1.0E-10 THEN
    BEGIN
        D := 0;
        EXIT;
    END;
    FOR I := 1 TO N DO
    BEGIN
        IF I<> K THEN
        BEGIN
            IK := NK + I;
            A[IK] := A[IK] / (-BIGA);
        END;
    END;
    FOR I := 1 TO N DO
    BEGIN
        IK := NK + I;
        HOLD := A[IK];
        IJ := I - N;
        FOR J := 1 TO N DO
        BEGIN
            INC(IJ,N);
            IF  I<>K THEN
            BEGIN
                KJ := IJ - I + K;
                A[IJ] := HOLD*A[KL] + A[IJ];
            END;
        END;
    END;
    KJ := K - N;
    FOR J := 1 TO N DO
    BEGIN
        INC (KJ, N);
        IF J <> K THEN A[KJ] := A[KJ] / BIGA;
    END;
    D := D * BIGA;
    A[KK] := 1.0 / BIGA;
END;
K := N-1;
REPEAT
    I := L[K];
    IF I>K THEN
    BEGIN
        JQ := N*(K-1);
        JR := N*(I-1);
        FOR J := 1 TO N DO
        BEGIN
            JK := JQ + J;
            HOLD := A[JK];
            JI := JR + J;
            A[JK] := -A[JI];
            A[JI] := HOLD;
        END;
    END;
    J := M[K];
    IF J>K THEN
    BEGIN          KI := K - N;
        FOR I := 1 TO N DO
        BEGIN
            KI := KI + N;
            HOLD := A[KI];
            JI := KI - K + J;
            A[KI] := -A[JI];
            A[JI] := HOLD;
        END;
    END;
    DEC (K);
UNTIL K=0;
END.
```

Для проверки работы процедур решим систему нелинейных уравнений с точностью до 0.001, используя метод Ньютона:

$$\begin{cases} \sin(2x - y) - \frac{1}{2}x = 0.4; \\ 0.8x^2 - 1.5y^2 = 1. \end{cases}$$

После отделения корней выяснилось, что система имеет решение по x на интервале $0.4 < x < 0.5$, а по y - $0.75 < y <$ <-0.73. Таким образом, за начальные приближения можно взять: $x_0 = 0.4$; $y_0 = -0.75$. Далее имеем

$F(x,y) = \sin(2x - y) - 0.4 - 1.2x$;

$G(x,y) = 0.8x^2 + 1.5y^2 - 1$;

$F_x' = 2\cos(2x - y) - 1.2$; $G_x' = 1.6x$;

$F_y' = -\cos(2x - y)$; $G_y' = 3y$.

Уточнение корней системы проведем методом Ньютона. Взяв за основу предлагаемые процедуры и доопределив процедуры для вычисления $F(x)$, $G(x)$ и $F'(x)$, $G'(x)$ как

```
PROCEDURE FUNCF (N : INTEGER; XX1,XX2 : REAL;
                 VAR Y : ARRAY OF REAL);
BEGIN
  Y[1] := XX1 - EXP(-XX2);
  Y[2] := XX2 - EXP (XX1);
```

```
END.

PROCEDURE FUNCG (N : INTEGER; XX1,XX2: REAL;
          VAR A : ARRAY OF REAL);
BEGIN
  A[1] := 1.0;
  A[2] := - EXP(XX1);
  A[3] := EXP(-XX2);
  A[4] := 1.0;
END
```

получим решение поставленной задачи:

$X_1 = 0.273332;\ Y_1 = 1.275009;$ итерация = 1;

$X_2 = 0.273332;\ Y_2 = 1.275009;$ итерация = 2.

3.4. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ КВАДРАТНОГО КОРНЯ

Метод квадратного корня основан на представлении матрицы $\boldsymbol{A}$, составленной из коэффициентов системы в форме произведения треугольных матриц, что позволят свести решение заданной системы к последовательному решению двух систем с треугольными матрицами.

Метод квадратного корня применяется для решения системы линейных уравнений, коэффициенты которой образуют *эрмитову симметрическую матрицу* (эрмитова матрица совпадает с комплексно-сопряженной транспонированной $\boldsymbol{A}^* = \boldsymbol{A}$).

Представим матрицу $\boldsymbol{A}$ в виде $\boldsymbol{A} = \boldsymbol{S}^* \boldsymbol{D}\, \boldsymbol{S}$, где $\boldsymbol{S}$ - верхняя треугольная матрица с положительными элементами на главной диагонали; $\boldsymbol{S}^*$- транспонированная к матрице $\boldsymbol{S}$; $\boldsymbol{D}$- диагональная матрица, на диагонали которой находятся числа (+1) или (— 1).

Если матрицы $\boldsymbol{S}$ и $\boldsymbol{D}$ найдены, то заданная система $\boldsymbol{AX} = \boldsymbol{F}$ может быть решена следующим путем:

$$\boldsymbol{AX} = \boldsymbol{S}^* \boldsymbol{D}\, \boldsymbol{S}\, \boldsymbol{X} = (\boldsymbol{S}^* \boldsymbol{D})\, \boldsymbol{S}\, \boldsymbol{X} = \boldsymbol{B}\, \boldsymbol{Y} = \boldsymbol{F}, \quad (1.28)$$

где $\boldsymbol{S}^* \boldsymbol{D} = \boldsymbol{B}$ есть нижняя треугольная матрица и $\boldsymbol{Y} = \boldsymbol{S}\, \boldsymbol{X}$ - вспомогательный вектор.

Таким образом, решение системы $\boldsymbol{AX} = \boldsymbol{F}$ равносильно решению двух треугольных систем $\boldsymbol{BY} = \boldsymbol{F}$ и $\boldsymbol{SX} = \boldsymbol{Y}$.

Пусть $\boldsymbol{S} = (s_{ik})$ при $i > k$ и $s_{ik} \neq 0$, $s_{ii} > 0$; $\boldsymbol{S}^* = (s_{ik}^*)$; $\boldsymbol{D} = (d_{ik})$, $d = \pm 1$; $i \neq k$. Тогда из сравнения матриц $\boldsymbol{A}$ и $\boldsymbol{S}^* \boldsymbol{D} \boldsymbol{S}$ получим

$$a_{kl} = \sum_{i=1}^{n} s_{ik}^* \cdot d_{ii} \cdot s_{il} = \sum_{i=1}^{\min(k,l)} d_{ii} \cdot s_{ik}^* \cdot s_{il} \cdot$$

Ограничение в сумме получается из учета того факта, что в матрице $\boldsymbol{S}$ ниже главной диагонали элементы обращаются в нуль. Последнее равенство можно переписать несколько иначе, приняв для определенности $k = \min(k, l)$:

$$a_{kk} = \sum_{i=1}^{k} d_{ii} \cdot |s_{ik}|^2 = \sum_{i=1}^{k-1} d_{ii} \cdot |s_{ik}|^2 + d_{kk} \cdot s_{kk}^2 ;$$

$$a_{kl} = \sum_{i=1}^{k} d_{ii} \cdot s_{ik}^* \cdot s_{il} = \sum_{i=1}^{k-1} d_{ii} \cdot s_{ik}^* \cdot s_{il} + d_{kk} \cdot s_{kk} \cdot s_{kl} ,$$

откуда окончательно получим формулы для вычисления элементов матриц $\boldsymbol{S}$ и $\boldsymbol{D}$:

$$\begin{cases} d_{kk} = Sign\left(a_{kk} - \sum_{i=1}^{k-1} d_{ii} \cdot |s_{ik}|^2 \right) ; \\ s_{kk} = \sqrt{\left| a_{kk} - \sum_{i=1}^{k-1} d_{ii} \cdot |s_{ik}|^2 \right|} \quad ; \\ s_{kl} = \dfrac{a_{kl} - \sum_{i=1}^{k-1} d_{ii} \cdot s_{ik}^* \cdot s_{il}}{d_{kk} \cdot s_{kk}} \quad . \end{cases} \quad (1.29)$$

Единственным условием возможности определения s_{ik} является $s_{kk} \neq 0$. Для построения матриц полагаем $k = 1$ и последовательно вычисляем все элементы первой строки s по формуле (1.29); затем полагаем $k = 2$ - и определяем элементы второй строки и т.д. Когда $k = n$, тогда найдены все элементы матриц $\boldsymbol{S}$ и $\boldsymbol{D}$, а следовательно и $\boldsymbol{S}^*$. Затем последовательно выполняем вычисления (1.28):

$$\boldsymbol{S}^* \boldsymbol{Z} = \boldsymbol{F};\ \boldsymbol{D}\, \boldsymbol{Y} = \boldsymbol{Z};\ \boldsymbol{S}\, \boldsymbol{X} = \boldsymbol{Y}$$

обычным ходом по формулам

$$\begin{cases} y_1 = f_1 / (s_{11} \cdot d_{11}) \ ; \\ y_i = \dfrac{\left(f_i - \sum_{l=1}^{i-1} d_{ll} \cdot y_l \cdot s_{li}^* \right)}{s_{ii} \cdot d_{ii}} \ ; \\ x_n = y_n / s_{nn} \quad ; \\ x_i = \dfrac{y_i - \sum_{l=i+1}^{n} s_{li} \cdot x_l}{s_{ii}} \end{cases} \quad (1.30)$$

при $i = 2, 3, \ldots, n$.

Попутно заметим, что определитель матрицы $\boldsymbol{A}$ можно вычислить из выражения

$$det\ A = \prod_{i=1}^{n} d_{ii} \cdot s_{ii}^2 \cdot \quad (1.31)$$

Этот метод экономичен, требует $n^3/3$ арифметических действий и при больших n ($n > 50$) вдвое быстрее метода Гаусса. Если за основу процедуры принять алгоритм П5.6

Дьяконова [1987], то тогда метод квадратного корня может быть реализован с помощью следующей процедуры:

```
PROCEDURE KVK (M, N : INTEGER;
               AA: ARRAY OF REAL; BB : ARRAY OF REAL;
               VAR C: ARRAY OF REAL);
TYPE MAS1=ARRAY [0..4] OF REAL;
     MAS=ARRAY [0..4] OF MAS1;
VAR A : MAS; J, K, I : INTEGER;   S, C1 : REAL;
BEGIN
  I := 0;
  FOR J := 1 TO M DO
     FOR K := 1 TO N DO
     BEGIN     INC (I);     A[J,K] := AA [I];        END;
     FOR J := 1 TO N DO
     BEGIN
        FOR K := J TO N DO
        BEGIN
           S := 0.0;
           FOR I := 1 TO M DO   S := S + A[I,J] * A[I,K];
           C[K] := S;
        END;
        C1 := 0.0;
        FOR I := 1 TO M DO   C1 := C1 + A[I,J]*BB[I];
        FOR I := J TO N DO  A[I,J] := C[I];
        C[J] := C1;
     END;
     A[1,1] := SQRT (A[1,1]);
     FOR J := 2 TO N DO  A[1,J] := A[J,1] / A[1,1];
     FOR I := 2 TO N DO
     BEGIN
        S := 0.0;
        FOR K := 1 TO I-1 DO  S := S + A[K,I]*A[K,I];
        A[I,I] := SQRT (A[I,I] - S);
        FOR J := I+1 TO N DO
        BEGIN
           S := 0.0;
           FOR K := 1 TO I-1 DO
              S := S + A[K,I]*A[K,J];
           A[I,J] := (A[J,I] - S) / A[I,I];
        END;
     END;
     C[1] := C[1] / A[1,1];
     FOR I := 2 TO N DO
     BEGIN     S := 0.0;
        FOR K := 1 TO I-1DO
           S := S + A[K,I] * C[K];
        C[I] := (C[I] - S) / A[I,I];
     END;
     C[N] := C[N] / A[N,N];
     FOR I := N-1 DOWNTO 1 DO
     BEGIN              S := 0.0;
        FOR K := I+1 TO N DO   S := S + A[I,K] * C[K];
        C[I] := (C[I] - S) / A[I,I];
     END;
END.
```

Формальные параметры процедуры.*Входные: m, N* (тип ***integer***) - *m* уравнений и *n* неизвестных определяют размер матрицы ***A.*** Для этой процедуры обязательно выполнение $m > n$, т.е. количество уравнений должно быть больше количества неизвестных (система переопределена), предельный разрешенный случай $m = n$; *a* - матрица, составленная из коэффициентов при неизвестных; *B* - массив, составленный из столбца свободных членов. *Выходные: C* - массив, в котором содержится решение системы.

Для проверки правильности работы процедуры решалась система 4×4 линейных уравнений методом квадратных корней с точностью 0.0001:

$$\begin{aligned} 0.68X_1 + 0.05X_2 + 0.11X_3 + 0.08X_4 &= 2.15; \\ 0.05X_1 + 0.13X_2 + 0.27X_3 + 0.80X_4 &= 0.44; \\ 0.11X_1 + 0.27X_2 + 0.28X_3 + 0.06X_4 &= 0.83; \\ 0.08X_1 + 0.80X_2 + 0.06X_3 + 0.12X_4 &= 1.16, \end{aligned}$$

решение которой, полученное методом квадратных корней, будет следующим

$$X_1 = 2.97; X_2 = 1.11; X_3 = 0.74; X_4 = -0.07.$$

3.5. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ ИТЕРАЦИЙ

Итерационные методы решения систем линейных уравнений дают возможность вычислить решение системы как предел бесконечной последовательности промежуточных решений. Причем каждое последующее решение в случае сходимости итерационного процесса считается более точным. В этих методах, в отличие от точных (см. п. 3.1 - 3.4), ошибки в начальном приближении и последующих вычислениях компенсируются, т.е. итерационные методы (в случае сходимости) позволяют получить решение более точное, чем прямые. Поэтому итерационные методы относят к *самоисправляющимся*.

Условия и скорость сходимости процесса в большей степени зависят от свойств уравнений, т.е. от свойств матрицы системы и от выбора начального приближения.

Пусть дана система линейных алгебраических уравнений (1.22) с неособенной матрицей. В методе простой итерации если $a_{ii} \neq 0$, то исходная система может быть преобразована к виду $x_i = b_i + a_{ij} x_j$, $i \neq j$, т.е. из каждого уравнения последовательно выражают x_i.

Здесь $b_i = \boldsymbol{F}_i / a_{ii}$; $a_{ij} = - a_{ij} / a_{ii}$. Таким образом, в матричном виде имеем $\boldsymbol{X} = \boldsymbol{B} + \boldsymbol{AX}$. Полученную систему будем решать *методом последовательных приближений*. За нулевое приближение $\boldsymbol{X}^{(0)}$ можно принять матрицу ***B:*** $\boldsymbol{X}^{(0)} = = \boldsymbol{B}$, и далее, подставив найденные значения в исходную систему, получим $\boldsymbol{X}^{(1)} = \boldsymbol{B} + \boldsymbol{A}\,\boldsymbol{X}^{(0)}$.

При бесконечном повторении этой вычислительной схемы имеем

$$\lim_{n \to \infty} \boldsymbol{X}^{(n)} = \hat{\boldsymbol{X}},$$

где $\hat{\boldsymbol{X}}$ и будет искомое решение системы.

Условия сходимости итерационного процесса определяются теоремами, которые приводятся нами без доказательств.

Теорема 1. *Для того, чтобы последовательность приближений* $\boldsymbol{X}^{(n)}$ *сходилась, достаточно, чтобы все собственные значения матрицы* $\boldsymbol{A}$ *были по модулю меньше единицы:* $|\lambda_i| < 1$, $i = 1, 2, ..., n$.

Теорема 2. *Если требовать, чтобы последовательность* $\boldsymbol{X}^{(n)}$ *сходилась к* $\hat{\boldsymbol{X}}$ *при любом начальном приближении* $\boldsymbol{X}^{(0)}$, *то условие теоремы* 1 *является необходимым.*

Применение теорем 1 и 2 требует знания всех собственных значений матрицы $\boldsymbol{A}$, нахождение которых является очень не простой задачей. Поэтому на практике ограничиваются более простой теоремой, дающей достаточные условия сходимости.

Теорема 3. *Если для системы* $\boldsymbol{X} = \boldsymbol{B} + \boldsymbol{AX}$ *выполняется хотя бы одно из условий :*

$$\sum_{i=1}^{n} \left|a_{ij}\right| < 1, j = 1,2,\ldots,n;$$

$$\sum_{j=1}^{n} \left|a_{ij}\right| < 1, i = 1,2,\ldots,n,$$

то итерационный процесс сходится к единственному решению этой системы независимо от выбора начального приближения.

Для многих приложений важно знать, какой является скорость сходимости процесса, и оценить погрешность полученного решения.

Теорема 4. *Если какая-либо норма матрицы* $\boldsymbol{A}$, *согласованная с рассматриваемой нормой вектора* $\boldsymbol{X}$, *меньше единицы, то верна следующая оценка погрешности приближения в методе простой итерации:*

$$\left\|\boldsymbol{X}^{*} - \boldsymbol{X}^{(k)}\right\| \le \|\boldsymbol{A}\|^{k} \cdot \left\|\boldsymbol{X}^{(0)}\right\| + \frac{1}{1-\|\boldsymbol{A}\|} \cdot \|\boldsymbol{A}\|^{k} \cdot \|\boldsymbol{B}\|.$$

В библиотеках стандартного математического обеспечения ЭВМ всегда можно найти несколько вариантов программы, выполняющей решение системы линейных уравнений методом простой итерации.

Так, можно предложить к использованию следующую удобную и достаточно эффективную процедуру, взятую из БСП БЭСМ для транслятора ALFA. Перевод на язык *PASCAL* выполнен авторами.

```
PROCEDURE ITER (N, IK :INTEGER; EPS : REAL;
                A : MAS1; B : MAS; VAR X : MAS);
VAR  X1 : MAS;   S : REAL;   I, J, K : INTEGER;
BEGIN   X1 := B;
  X := X1;   K := 0;
  REPEAT        S := 0.0;
     INC(K);
     FOR I := 1 TO N DO
     BEGIN
        FOR J := 1 TO N DO X[I] := A[I,J]*X1[J] + B[J];
        S := S + ABS (X[I]-X1[I]);
     END;
     S := S / N;      X1 := X;
  UNTIL (S<EPS) AND (K>IK);
END.
```

Формальные параметры процедуры. *Входные:* $\boldsymbol{A}$ (тип ***real***) - матрица, составленная из коэффициентов при $\boldsymbol{X}$ преобразованного уравнения; $\boldsymbol{B}$ (тип ***real***) - матрица, составленная из свободных членов; N (тип ***integer***) - размерность матриц $\boldsymbol{A}$ ($N \times N$) и $\boldsymbol{B}(N)$; *IK* (тип ***integer***) - предельно возможное количество итераций (введено для того, чтобы в случае расхождения процесса выйти из подпрограммы. Обычно решение достигается за 3-6 итераций); *EPS* (тип ***real***) - заданная погрешность решения. *Выходные:* $\boldsymbol{X}$ (тип ***real***) - матрица, в которой находится решение системы.

Для примера методом простых итераций решена система 4×4 линейных уравнений с точностью до 0.0001:

$$\begin{aligned}
X_1 &= 0.08X_1 + 0.05X_2 + 0.11X_3 + 0.08X_4 + 2.15\\
X_2 &= 0.05X_1 + 0.13X_2 + 0.27X_3 + 0.28X_4 + 0.44\\
X_3 &= 0.11X_1 + 0.27X_2 + 0.28X_3 + 0.06X_4 + 0.83\\
X_4 &= 0.08X_1 + 0.18X_2 + 0.06X_3 + 0.12X_4 + 1.16
\end{aligned}$$

Результаты вычислений представлены в табл. 1.16.

Таблица 1.16

X[1]	*X*[2]	*X*[3]	*X*[4]	№ итерации
2.150000	0.440000	0.830000	1.160000	*ITER* = 0
1.252800	1.484800	1.229600	1.299200	*ITER* = 1
1.263936	1.523776	1.237952	1.315904	*ITER* = 2
1.265272	1.528453	1.238954	1.317908	*ITER* = 3
1.265433	1.529014	1.239075	1.318149	*ITER* = 4
1.265452	1.529082	1.239089	1.318178	*ITER* = 5
1.265454	1.529090	1.239091	1.318181	*ITER* = 6
1.265455	1.529091	1.239091	1.318182	*ITER* = 7
1.265455	1.529091	1.239091	1.318182	:РЕШЕНИЕ

3.6. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ ЗЕЙДЕЛЯ

Этот метод относится к итерационным, имеет более быструю сходимость по сравнению с методом простых итераций (см. п. 3.5), но часто приводит к громоздким вычислениям.

Метод Зейделя применяют в двух расчетных схемах. Рассмотрим каноническую форму (для схемы итераций). Пусть система приведена к виду $\boldsymbol{X} = \boldsymbol{Bx} + \boldsymbol{b}$. В схеме простой итерации следующее приближение $\boldsymbol{X}^{(k+1)}$ находится путем подстановки предыдущего приближения $\boldsymbol{X}^{(k)}$ в правую часть выражения $\boldsymbol{X}^{(k+1)} = \boldsymbol{B}\,\boldsymbol{X}^{(k)} + \boldsymbol{b}$.

Для удобства рассуждений полагаем, что левые части уравнений содержат x_i (элементы матрицы $\boldsymbol{X}^{(k+1)}$) с воз-

растающими номерами, т.е. сначала x_1, затем x_2, x_3, ..., x_n. Тогда решение системы уравнений $\boldsymbol{X} = \boldsymbol{Bx} + \boldsymbol{b}$ на очередной $(k+1)$ итерации будет имет вид

$$x_i^{(k+1)} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} \cdot x_j^{(k+1)} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} \cdot x_j^{(k)} + \frac{f_i}{a_{ii}}, \quad (1.32)$$

т.е. каждое очередное найденное приближение $x_i^{(k+1)}$ сразу же используется для определения $x_{i+1}^{(k+1)}$. Условия сходимости для итерационного метода Зейделя дают те же теоремы, что и в методе простых итераций.

Вторая форма метода Зейделя требует предварительного приведения системы (1.22) к виду, когда все диагональные элементы отличны от нуля. Если разложить матрицу $\boldsymbol{A}$ на сумму двух матриц $\boldsymbol{R} + \boldsymbol{C}$, где $\boldsymbol{R}$ - нижняя диагональная матрица, а $\boldsymbol{C}$ - верхняя с нулями на главной диагонали, то систему (1.22) можно записать как

$$\boldsymbol{Ax} = (\boldsymbol{R} + \boldsymbol{C})\boldsymbol{x} = \boldsymbol{R}\,\boldsymbol{x}^{(k+1)} + \boldsymbol{C}\,\boldsymbol{x}^{(k)} = \boldsymbol{B}$$

$$\text{или } \boldsymbol{x}^{(k+1)} = \boldsymbol{R}^{-1}\boldsymbol{B} - \boldsymbol{R}^{-1}\boldsymbol{C}\,\boldsymbol{x}^{(k)}$$

и тогда становится ясно, что метод Зейделя в канонической форме равносилен методу простой итерации, примененному к системе

$$\boldsymbol{X} = \boldsymbol{R}^{-1}\boldsymbol{B} - \boldsymbol{R}^{-1}\boldsymbol{C}\,\boldsymbol{X}.$$

Для решения системы линейных уравнений методом Зейделя можно использовать процедуру из п. 3.5, слегка ее модифицировав для случая системы n уравнений:

```
PROCEDURE ITER (N, IK :INTEGER; EPS : REAL;
          A : MAS1; B : MAS; VAR X : MAS);
VAR  X1 : MAS;   S : REAL;   I, J, K : INTEGER;
BEGIN        X1 := B;
  X := X1;
  K := 0;
  REPEAT        S := 0.0;
     INC(K);
     FOR I := 1 TO N DO
     BEGIN
        FOR J := 1 TO N DO
           X[I] := A[I,J]*X1[J] + B[J];
        S := S + ABS (X[I]-X1[I]);
        X1 := X;
     END;
     S := S / N; X1 := X;
  UNTIL (S<EPS) AND (K>IK);
END.
```

Формальные параметры процедуры такие же, как и в п. 3.5. Для сравнения решим такую же систему линейных уравнений, как в методе итераций (см. п. 3.5). Результат решения системы методом Зейделя (табл. 1.17) можно сравнить с результатами табл. 1.16.

Таблица 1.17

$x[1]$	$x[2]$	$x[3]$	$x[4]$	№ итерации
2.150000	0.440000	0.830000	1.160000	$ITER = 0$
1.252800	1.484800	1.229600	1.299200	$ITER = 1$
1.263936	1.523776	1.237952	1.315904	$ITER = 2$
1.265272	1.528453	1.238954	1.317908	$ITER = 3$
1.265433	1.529014	1.239075	1.318149	$ITER = 4$
1.265452	1.529082	1.239089	1.318178	$ITER = 5$
1.265452	1.529082	1.239089	1.318178	:РЕШЕНИЕ

§ 4. АЛГЕБРА МАТРИЦ

Матрицей называется прямоугольная таблица из чисел, содержащая некоторое количество m строк и n столбцов, при этом числа m и n называются *порядками матрицы*. Если $m = n$, матрица называется *квадратной*, а число $m = n$ - ее *порядком*. Для записи матриц используются следующие обозначения:

$$\boldsymbol{A} = \|a_{ij}\| = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \quad (1.33)$$

где числа a_{ij} называются *элементами матрицы* $\boldsymbol{A}$. В случае, если $m = n$ и определитель матрицы $\det\boldsymbol{A} \neq 0$, матрица $\boldsymbol{A}$ называется *невырожденной* и для нее можно найти обратную матрицу. *Обратной* по отношению к данной матрице $\boldsymbol{A}$ называется матрица $\boldsymbol{A}^{-1}$, которая, будучи умноженной как справа, так и слева на $\boldsymbol{A}$, дает единичную матрицу:

$$\boldsymbol{A} \cdot \boldsymbol{A}^{-1} = \boldsymbol{A}^{-1} \cdot \boldsymbol{A} = \boldsymbol{E} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix}.$$

Матрица $\boldsymbol{A}^T$, полученная перестановкой строк со столбцами в матрице $\boldsymbol{A}$, называется *транспонированной*. Квадратная матрица $\boldsymbol{A}$ называется *симметрической*, если $\boldsymbol{A}^T = \boldsymbol{A}$, и *ортогональной*, если $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{E}$.

Характеристическим уравнением матрицы $\boldsymbol{A}$ называется матричное уравнение $||\boldsymbol{A} - \lambda\boldsymbol{E}|| = \boldsymbol{0}$, в котором λ_i - *собственные* (или характеристические) *числа* (или значения) матрицы $\boldsymbol{A}$. *Собственным вектором*, отвечающим собственному числу λ_i, называется вектор

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix},$$

который удовлетворяет матричному уравнению $\boldsymbol{Ax} = \lambda_i \boldsymbol{x}$. Когда говорят о вычислении собственных чисел, то различают *полную* и *частичную проблему* собственных чисел. В полной проблеме вычисляют все собственные числа и соответствующие им собственные векторы матриц, а под частичной проблемой обычно понимают задачу нахождения одного или нескольких собственных чисел и соответствующих им собственных векторов.

В настоящем параграфе рассматривается несколько эффективных алгоритмов, реализующих как операции над матрицами, так и задачи решения полной проблемы собственных значений.

Задача определения собственных значений и *собственных векторов* матриц имеет большое значение при решении очень широкого спектра задач. Методы решения указанной задачи по типу примененной вычислительной схемы можно разделить на *прямые* (точные) и *итерационные* (приближенные).

В *прямых методах* по некоторому правилу вычисляются коэффициенты матрицы с заранее известными свойствами, а затем собственные значения находятся как корни характеристического многочлена по какому-либо известному численному методу. После этого определяются собственные векторы, что считается не очень сложной задачей.

В *итерационных методах* коэффициенты характеристического уравнения не вычисляют, а составляют некоторые итерационные последовательности, позволяющие найти одно или несколько, а иногда и все собственные значения исходной матрицы $\boldsymbol{A}$. Итерационные методы почти всегда более трудоемкие, однако они надежнее прямых методов, так как менее чувствительны к ошибкам округления.

К прямым можно отнести методы Крылова, Данилевского, Самуэльсона и др., а к итерационным - степенной и метод Якоби, или, как его еще называют, метод вращений. Последний метод считается наиболее эффективным из всех известных [Крылов и др., 1972].

4.1. ВЫЧИСЛЕНИЕ СОБСТВЕННЫХ ВЕКТОРОВ И СОБСТВЕННЫХ ЗНАЧЕНИЙ МАТРИЦ ПО МЕТОДУ КРЫЛОВА

Пусть $\boldsymbol{A}$ - некоторая квадратная матрица $||a_{ij}||$ порядка n. Рассмотрим связанное с ней уравнение $||\boldsymbol{A} - \lambda\boldsymbol{E}|| = 0$, определитель которого есть алгебраический многочлен степени n от λ:

$$||\boldsymbol{A} - \lambda\boldsymbol{E}|| = (-1)^n (\lambda^n - P_1\lambda^{n-1} - \ldots - P_{n-1}\lambda - P_n) . \quad (1.34)$$

Корнями этого многочлена являются собственные значения (собственные числа) матрицы $\boldsymbol{A}$. Эта система имеет ненулевое решение тогда и только тогда, когда $\lambda = \lambda_1, \lambda_2, \ldots, \lambda_n$.

А.Н.Крылов[1] предложил эффективный метод построения характеристического многочлена, основанный на применении минимального многочлена матрицы, аннулирующего заданный вектор.

Возьмем произвольный вектор $\boldsymbol{c}^{(0)} \neq 0$, размерности n и составим последовательность линейно независимых векторов по правилу

$$\boldsymbol{c}^{(1)} = \boldsymbol{A}\,\boldsymbol{c}^{(0)};\ \boldsymbol{c}^{(2)} = \boldsymbol{A}\,\boldsymbol{c}^{(1)};\ \ldots;\ \boldsymbol{c}^{(n)} = \boldsymbol{A}\,\boldsymbol{c}^{(n-1)}.$$

Очевидно, что $\boldsymbol{c}^{(n)}$ будет являться линейной комбинацией предыдущих линейно независимых векторов.

Расписав полученную на последнем шаге систему

$$\begin{cases} q_1 c_1^{(n-1)} + q_2 c_1^{(n-2)} + \ldots + q_n c_1^{(0)} = c_1^{(n)}; \\ q_1 c_2^{(n-1)} + q_2 c_2^{(n-2)} + \ldots + q_n c_2^{(0)} = c_2^{(n)}; \\ \vdots \\ q_1 c_n^{(n-1)} + q_2 c_n^{(n-2)} + \ldots + q_n c_n^{(0)} = c_n^{(n)}, \end{cases} \quad (1.35)$$

где c_i - соответствующие координаты вектора $\boldsymbol{c}^{(i)}$, а q_i - неопределенные коэффициенты, получим неоднородную систему из n алгебраических уравнений. Определитель этой системы

$$D = \det = \begin{vmatrix} c_1^{(n-1)} c_1^{(n-2)} \ldots c_1^{(0)} \\ c_2^{(n-1)} c_2^{(n-2)} \ldots c_2^{(0)} \\ \vdots \\ c_n^{(n-1)} c_n^{(n-2)} \ldots c_n^{(0)} \end{vmatrix} \neq 0$$

будет отличен от нуля тогда и только тогда, когда $\boldsymbol{c}^{(i)}$ образуют систему линейно независимых векторов [Крылов и др., 1972]. Полученные значения q_i будут равны P_i - коэффициентам характеристического уравнения матрицы $\boldsymbol{A}$ (1.34).

Для определения q_i систему (1.35) решают методом Гаусса (с обязательной проверкой). Если систему не удается привести в ходе решения к треугольному виду, то говорят о зависимости построенной системы векторов, и тогда можно записать только делитель характеристического многочлена матрицы. Сам многочлен решается одним из известных методов решения линейных алгебраических уравнений.

После того как найдены собственные значения λ_i матрицы $\boldsymbol{A}$, ищут собственные векторы матрицы $\boldsymbol{A}$, что приводит к решению следующей однородной системы линейных алгебраических уравнений:

$$(\boldsymbol{A} - \lambda\,\boldsymbol{E})\boldsymbol{x} = 0. \quad (1.36)$$

Так как λ_i являются корнями минимального аннулирующего вектора $\boldsymbol{c}^{(0)}$ многочлена (1.34), то решения системы (1.36) представляют в виде линейной комбинации независимых векторов $\boldsymbol{c}^{(i)}$

1 Крылов А.Н. О численном решении уравнения высокой степени // ИАН ОМЕН. 1931. № 4. С. 491-539

$$\boldsymbol{x}^{(k)} = \beta_{i1}\,\boldsymbol{c}^{(n-1)} + \beta_{i2}\,\boldsymbol{c}^{(n-2)} + \ldots + \beta_{in}\,\boldsymbol{c}^{(0)},$$

где коэффициенты β_{ij} удовлетворяют условию $\boldsymbol{A}\boldsymbol{x}^{(n)}=\lambda_k \boldsymbol{x}^{(n)}$. Умножая $\boldsymbol{x}^{(n)}$ на $\boldsymbol{A}$ и учитывая, что $\boldsymbol{c}^{(j)}=\boldsymbol{A}\boldsymbol{c}^{(j-1)}$, получают

$$\lambda_i(\beta_{i1}\boldsymbol{c}^{(n-1)}+\beta_{i2}\boldsymbol{c}^{(n-2)}+\ldots+\beta_{in}\boldsymbol{c}^{(0)}) =$$
$$= \beta_{i1}\,\boldsymbol{c}^{(n)} + \beta_{i2}\,\boldsymbol{c}^{(n-1)} + \ldots + \beta_{in}\,\boldsymbol{c}^{(1)}.$$

Если же учесть, что $\varphi(\boldsymbol{A})\boldsymbol{c}^{(n)} \equiv 0$, то после приведения подобных (с учетом формул 1.35) последнее равенство можно переписать в виде

$$(q_n\,\beta_{i1} - \lambda_i\,\beta_{in})\boldsymbol{c}^{(0)} + (q_{n-1}\,\beta_{i1} + \beta_{in} - \lambda_i\,\beta_{in-1})\boldsymbol{c}^{(1)} +$$

$$+ (q_{n-2}\beta_{i1} + \beta_{in-1} - \lambda_i\beta_{in})\boldsymbol{c}^{(2)} + \ldots + (q_1\beta_{i1} + \beta_{i2} - \lambda_i\beta_{i1})\boldsymbol{c}^{(n-1)} = 0.$$

В силу линейной независимости векторов $\boldsymbol{c}^{(i)}$ последнее уравнение может быть равно нулю только тогда, когда коэффициенты будут равны нулю, т.е.

$$q_n \beta_{i1} - \lambda_i \beta_{in} = 0 ;$$
$$q_{n-1} \beta_{i1} + \beta_{in} - \lambda_\iota \beta_{in-1} = 0 ;$$
$$q_{n-2} \beta_{i1} + \beta_{in-1} - \lambda_i \beta_{in} = 0 ;$$
$$\ldots$$
$$q_1 \beta_{i1} + \beta_{i2} - \lambda_i \beta_{i1} = 0 \quad ,$$

из которых находим β_{ik}, начиная с последнего:

$$\beta_{i2} = (\lambda_i - q_1)\, \beta_{i1} ;$$
$$\beta_{i3} = (\lambda_i^2 - q_1 \lambda_i - q_2)\, \beta_{i1} ;$$
$$\ldots$$
$$\beta_{in} = (\lambda_i^{n-1} - q_1 \lambda_i^{n-2} - \ldots - q_{m-1})\, \beta_{i1} ;$$
$$0 = (\lambda_i^{n} - q_1 \lambda_i^{n-1} - \ldots - q_m)\, \beta_{i1} .$$

Теперь, полагая $\beta_{i1} = K$, где K - любое отличное от нуля число, можно вычислить собственный вектор - β_{ij}. Используя различные λ_i, получаем систему собственных векторов β_{ij} .

Так как в методе Крылова применяются изученные ранее методы, то для его программной реализации дополнительно требуются только три несложные оригинальные процедуры.

Первая VECT - формирует матрицу $\boldsymbol{c}^{(k)}$ и "столбец свободных членов" для решения по схеме Гаусса. Вторая SVECT - формирует собственный вектор, соответствующий очередному собственному значению матрицы $\boldsymbol{A}$. Вычисления выполняются по схеме Горнера. Решение самого характеристического полинома относительно λ_i выполняется по программе LAMBDA Дьяконова [Справочник..., 1987], реализующей метод Хичкока. Найденное очередное решение полинома сохраняется последовательно в массиве *xx*, который после нахождения всех λ_i упорядочивают любым методом. Для определения q_i используют подпрограмму GAUSS из п. 3.2.

Алгоритм вычисления собственных векторов и собственных значений по методу Крылова приводится ниже.

Øàã 1. Формирование матрицы $\boldsymbol{C}^{(i)}$ и столбца β_i (процедура VECT).

Øàã 2. Решение системы $\boldsymbol{C}^{(i)} = \beta_i$ методом Гаусса (процедура GAUSS).

Øàã 3. Определение собственных значений матрицы $\boldsymbol{A}$ (процедура LAMBDA).

Øàã 4. Определение собственных векторов матрицы $\boldsymbol{A}$ (процедура SVECT).

Если предложенную схему принять за основу, то подпрограммы VECT, LAMBDA и SVECT могут выглядеть следующим образом.

```
PROCEDURE VECT (CONST N : INTEGER; A:MAS1;
                VAR C: MAS1; VAR B : MAS11);
VAR I,J,K : INTEGER; X : MAS;  S : REAL;
BEGIN
  FOR I := 1 TO N DO
  BEGIN
     X[I] := 0.0;
     C[I,N] := X[I];
     C[N,I] := 0.0;
  END;
  X[1] := 1.0;
  C[1,N] := X[1];
  FOR I := 2 TO N+1 DO
  BEGIN
     FOR J := 1 TO N DO
     BEGIN
        S := 0.0;
        FOR K := 1 TO N DO
           S := S + A[J,K]*X[K];
           IF I<>N+1 THEN C[J,N+1-I] := S
           ELSE B[J] := S;
     END;
     FOR J := 1 TO N DO
        X[J] := C[J,N+1-I];
  END;
END.

PROCEDURA LAMBDA (CONST N : INTEGER;E:REAL;
                  QQ : MAS11;  VAR XX : MAS11);
LABEL CONT, CONT1;
VAR I,J,K,NN : INTEGER;  Q : ARRAY [0..30] OF REAL;
     T,C,P,Q1,D,U,V,F,W,H,Y,Z,L,R,S,X,M : REAL;
     A, B, BIG : REAL;   DN : BOOLEAN;
BEGIN
  K := 1;
  DN := FALSE;
  NN:= N;
  X := 0.0;
  BIG := 1.0E-10;
  FOR I := 1 TO N+1 DO
     Q[I] := QQ[I];
  REPEAT
     T := 1.0;
     C := Q[2] / Q[1];
     IF NN =1 THEN
     BEGIN
        P := -C;
        Q1 := 0;
     END
     ELSE
     BEGIN
     IF NN = 2 THEN
           H := C*C/4.0 - Q[3]/Q[1];
        ELSE
        BEGIN
           M := 10;
           C := 4.0;
           D := 8.0;
```

```
        U := 4.0;
        F := 1.0;
        W := 2.0;
        T := 0.0;
        V := 8.0;
CONT:   IF ABS(M-10.0)<BIG THEN
        BEGIN
            P := C;
            M := 0.0;
            Q1 := D;
            C := U;
            D := V;
            U := P;
            V := Q1;
            Y := C;
            Z := D;
            F := -F;
        END;
        M := M + 1.0;
        H := 0.0;
        Q1 := Q[1];
        P := Q[2] - C*Q1;
        L := Q1;
CONT1:  FOR J := 3 TO NN DO
        BEGIN
            R := P;
            P := Q[J] - C*R - D*Q1;
            Q1 := R;        R := L;
            L := Q1 - C*R - H*D;
            H := R;
        END;
        Q1 := Q[NN+1] - D*Q1;
        S := L + C*R;
        IF ABS(T)<=BIG THEN
        BEGIN
            X := D*R;
            H := R*X + S*L;
            IF ABS(H)<=BIG THEN
                GOTO CONT;
        END;
        C := C + (P*S - Q1*R)/H;
        D := D + (P*X+Q1*L)/H;
        IF ABS (C-Y+D-Z) < BIG THEN
        BEGIN
            F := -W;
            EXIT;
        END;
        W  := -F;
        H := C*C/4.0 - D;
        IF SQRT((Q1-P*C/2)*(Q1-P*C/2)+
          P*P*ABS(H))>(E/NN) THEN
                GOTO CONT;
        T := 0.0;           Q[2] := Q[2] - C*Q[1];
        FOR J := 3 TO NN-1 DO
        Q[J] := Q[J] - C*Q[J-1] - D*Q[J-2];
    END;
            P := -C/2.0;
            Q1 := SQRT(ABS(H));
            IF H>= 0.0 THEN
            BEGIN
                M := P+Q1;
                P := P-Q1;
                Q1 := 0.0;
            END
            ELSE
                M := P;
            XX[K] := M;
            INC(K);
        END;
        XX[K] := P;
        INC(K);
        IF ABS(T) >1.0E-20 THEN
                EXIT;
        NN:= NN-2;
        IF NN=0 THEN
                EXIT;
        A := 0.0;
        IF SQRT((S-R*C/2)*(S-R*C/2)+
            R*R*ABS(H)) <= E  THEN
                A := 1;
        B := 0.0;
        IF NN>=2 THEN
            B := 1;
        IF (A+B)=2 THEN
        BEGIN
            T := 1.0;
            GOTO CONT1;
        END;
    UNTIL    DN;
END.

PROCEDURA SVECT (CONST N : INTEGER; LAM : REAL;
                 VAR Q:MAS11);
VAR I,J,K : INTEGER; SUM : REAL;
BEGIN
  SUM := LAM;
  FOR I := 1 TO N DO
  BEGIN
      SUM := SUM -Q[I];
      Q[I] := SUM;
      SUM := SUM*LAM;
  END;
END.
```

Процедуру GAUSS (или SINQ), как уже говорилось, можно взять из п. 3.1, 3.2.

Опишем **формальные параметры** процедур.

Процедура VECT**.** *Входные*: *N* (тип ***integer***) - размер матрицы ***A***(*n*×*n*) и вычисляемой матрицы ***C*** (*n*×*n*)*;* ***A*** (тип ***real***) - исходная матрица ***A*** (*n*×*n*), для которой вычисляют собственные значения и собственные векторы. *Выходные*: ***C*** (тип ***real***) - матрица (*n*×*n*), определяющая коэффициенты системы, решая которую находим q_i; ***B*** (тип ***real***) - матрица

$(1\times n)$ - столбец свободных членов системы, равен последнему $\boldsymbol{c}^{(n)}$.

Процедура LAMBDA**.** *Входные*: *N* (тип ***integer***) - размер матрицы $\boldsymbol{A}(n\times n)$ и количество собственных значений матрицы; *q* (тип ***real***) - массив $(1\times n)$ коэффициентов характеристического уравнения для определения λ_i. *Выходные*: *xx* (тип ***real***) - массив $(1\times n)$ собственных значений матрицы $\boldsymbol{A}$.

Процедура SVECT**.** *Входные*: *N* (тип ***integer***) - количество собственных векторов матрицы $\boldsymbol{A}$; *LAM* (тип ***real***)-собственное значение матрицы $\boldsymbol{A}$. *Выходные*: *Q* (тип ***real***) - массив $(n\times n)$ собственных векторов матрицы $\boldsymbol{A}$ для ее собственных значений.

Для примера найдены собственные значения и собственные векторы матрицы

$$\begin{pmatrix} 0.68 & 0.05 & 0.11 & 0.08 \\ 0.05 & 0.13 & 0.27 & 0.08 \\ 0.11 & 0.27 & 0.28 & 0.06 \\ 0.08 & 0.08 & 0.06 & 0.12 \end{pmatrix}.$$

Матрица коэффициентов взята из п. 3.4. Вычисления выполнялись с точностью до 10^{-4} с использованием предложенных процедур. Для контроля все промежуточные матрицы распечатаны. Результаты работы данной программы приводятся в табл. 1.18.

Таблица 1.18

Параметр	Номер ячейки				
	[4]	[3]	[2]	[1]	[0]
Матрица независимых векторов (массив *C*)	0.279269	0.357899	0.483400	0.680000	1.000000
	0.207845	0.163549	0.134200	0.050000	0.000000
	0.130137	0.130736	0.123900	0.110000	0.000000
	0.187324	0.166738	0.110600	0.080000	0.000000
Приведенная система для определения *Q* методом Гаусса	1.000000	0.163549	0.130736	0.166738	0.780301
	0.000000	1.000000	-0.052680	-0.086699	-0.499257
	0.000000	0.000000	1.000000	-0.029549	-0.452717
	0.000000	0.000000	0.000000	1.000000	0.101987
	0.000000	0.000000	0.000000	0.000000	1.000000
Решение системы (P_i)	-1.2100000	-0.2915000	0.5833630	0.101987	—
Собственные значения (λ_i)	0.214779	-0.699093	1.043229	0.651085	—
Матрица вспомогательных коэффициентов (*Beta*)	0.000000	1.000000	1.000000	1.000000	1.000000
	0.000000	-0.995221	-1.909093	-0.166771	-0.558915
	0.000000	-0.505252	1.043134	-0.465480	-0.655401
	0.000000	0.474846	-0.145884	0.097761	0.156641
Собственные векторы (ненормированные)	0.008083	-0.001510	0.058516	-0.201312	0.000000
	0.004728	-0.040495	0.117894	0.055773	0.000000
	-0.048150	0.008944	0.058870	-0.010608	0.000000
	0.016246	0.039043	0.111055	0.052490	0.000000
Собственные векторы (нормированные)	0.167872	-0.037291	0.496345	-1.000000	0.000000
	0.098187	-1.000000	1.000000	0.277045	0.000000
	-1.000000	0.220871	0.499348	-0.052693	0.000000
	0.337413	0.964153	0.941985	0.260739	0.000000

4.2. ВЫЧИСЛЕНИЕ СОБСТВЕННЫХ ВЕКТОРОВ И СОБСТВЕННЫХ ЗНАЧЕНИЙ МАТРИЦ ПО МЕТОДУ ДАНИЛЕВСКОГО

Очень простой и экономичный способ решения проблемы собственных значений был предложен в конце 30-х гг. А.М.Данилевским[1]. Этот метод основан на известном из линейной алгебры факте [Крылов и др., 1972; Курош, 1962], что подобные преобразования матрицы $\boldsymbol{A}$ не изменяют ее собственного многочлена. В самом деле, если $\boldsymbol{B} = \boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S}$, то

$$|\boldsymbol{B} - \lambda\boldsymbol{E}| = |\boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S} - \lambda\boldsymbol{S}^{-1}\boldsymbol{E}\boldsymbol{S}| = |\boldsymbol{S}^{-1}|\,|\boldsymbol{A} - \lambda\boldsymbol{E}|\,|\boldsymbol{S}| = |\boldsymbol{A} - \lambda\boldsymbol{S}|,$$

поэтому, удачно подобрав матрицу подобия, можно привести $\boldsymbol{A}$ к такой форме, что собственный многочлен будет строиться только по виду самой матрицы. А.М. Данилевский предложил приводить исходную матрицу $\boldsymbol{A}$ к канонической форме Фробениуса

[1] Матем.сб., 1937, N2, с.169-171.

$$\boldsymbol{\Phi} = \begin{pmatrix} p_1 & p_2 & \cdots & p_{n-1} & p_n \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & & & & \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}. \qquad (1.37)$$

Характеристический полином матрицы Фробениуса есть

$$|\boldsymbol{\Phi} - \lambda \boldsymbol{E}| = \begin{pmatrix} p_1 - \lambda & p_2 & \cdots & p_{n-1} & p_n \\ 1 & -\lambda & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & & & & \\ 0 & 0 & \cdots & 1 & -\lambda \end{pmatrix} =$$

$$= (p_1 - \lambda)\cdot\left(-\lambda^{n-1}\right) - \begin{pmatrix} p_2 & p_3 & \cdots & p_{n-1} & p_n \\ 1 & -\lambda & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & & & & \\ 0 & 0 & \cdots & 1 & -\lambda \end{pmatrix} =$$

$$= \ldots = (p_1 - \lambda)(-\lambda^{n-1}) - p_2(-\lambda^{n-2}) - \ldots - p_{n-1}\lambda^1 - p_n =$$
$$= (-1)^n P_n(\lambda).$$

Иными словами, элементы первой строки матрицы Фробениуса (1.37) есть соответствующие коэффициенты ее собственного многочлена (1.34). Если $\boldsymbol{\Phi}$ получена из матрицы $\boldsymbol{A}$ подобными преобразованиями, то собственный многочлен матрицы $\boldsymbol{A}$ совпадает с собственным многочленом матрицы $\boldsymbol{\Phi}$. Больше того оказывается, что найденная матрица подобия $\boldsymbol{S}$ в выражении $\boldsymbol{\Phi} = \boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S}$ может быть использована для нахождения собственных векторов матрицы $\boldsymbol{A}$ [Крылов и др., 1972].

Таким образом, основная задача сводится к отысканию матрицы подобия $\boldsymbol{S}$. А.М. Данилевский в своем алгоритме предложил строить матрицу $\boldsymbol{S}$, начиная процесс с последней строки. При этом последовательно, строка за строкой, исходная матрица приводится к форме Фробениуса. Рассмотрим эту вычислительную схему.

Предположим, что элемент матрицы $a_{n\,n-1} \neq 0$ (если это не так, то простой перестановкой столбцов можно всегда добиться выполнения условия), тогда разделим на него $n(n\text{-}1)$-й столбец матрицы $\boldsymbol{A}$. Полученный столбец умножим на a_{ni} и вычтем из i-го столбца. Проделав эту процедуру для $i = 1, 2, \ldots, n\text{-}1, n$, приведем последнюю строку к виду Фробениуса. Однако это равносильно умножению матрицы $\boldsymbol{A}$ на матрицу $\boldsymbol{M}_{n-1}$ вида

$$\boldsymbol{M}_{n-1} = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ \dfrac{-a_{n1}}{a_{nn-1}} & \dfrac{-a_{n2}}{a_{nn-1}} & \cdots & \dfrac{-1}{a_{nn-1}} & \dfrac{a_{nn}}{a_{nn-1}} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}. \qquad (1.38)$$

Но чтобы выполнить преобразование подобия, надо матрицу $\boldsymbol{A}$ домножить слева на матрицу $(\boldsymbol{M}_{n-1})^{-1}$, которая, как легко убедиться, имеет вид

$$\boldsymbol{M}_{n-1}^{-1} = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn-1} & a_{nn} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}. \qquad (1.39)$$

Аналогично строится последовательность матриц

$$\boldsymbol{M}_{n-1},\ \boldsymbol{M}_{n-2},\ \boldsymbol{M}_{n-3},\ \ldots,\ \boldsymbol{M}_{n-1}^{-1},\ \boldsymbol{M}_{n-2}^{-1},\ \boldsymbol{M}_{n-3}^{-1},\ \ldots,$$

и если при этом все $a_{ii\text{-}1} \neq 0$, то после n -1 шага метода Данилевского будем иметь

$$\boldsymbol{M}_1^{-1}\boldsymbol{M}_3^{-1}\ldots\boldsymbol{M}_{n-2}^{-1}\boldsymbol{M}_{n-1}^{-1}\boldsymbol{A}\boldsymbol{M}_{n-1}\boldsymbol{M}_{n-2}\ldots\boldsymbol{M}_2\boldsymbol{M}_1 = \boldsymbol{A}^{(\Phi)}.$$

Если теперь обозначить

$$\boldsymbol{S}^{-1} = \boldsymbol{M}_1^{-1}\boldsymbol{M}_3^{-1}\ldots\boldsymbol{M}_{n-2}^{-1}\boldsymbol{M}_{n-1}^{-1};\ \ \boldsymbol{S} = \boldsymbol{M}_{n-1}\boldsymbol{M}_{n-2}\ldots\boldsymbol{M}_2\boldsymbol{M}_1,$$

то равенство, полученное на n-1 шаге, может быть представлено в форме $\boldsymbol{A}^{(\Phi)} = \boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S}$. Когда исходная матрица $\boldsymbol{A}$ приведена к канонической форме Фробениуса, то только по виду первой строки можно выписать собственный многочлен матрицы $\boldsymbol{A}$.

Наши рассуждения относились к случаю, когда все коэффициенты $a_{ii\text{-}1} \neq 0$, который называется *регулярным*. Рассмотрим *нерегулярный* случай.

Предположим, что процесс приведения $\boldsymbol{A}$ к виду Фробениуса доведен до k-й строки и выполнено $n - k$ шагов преобразований. Следующий $n - k + 1$ шаг не может быть выполнен, так как элемент $a_{kk\text{-}1}$ равен нулю. Дальнейшие действия зависят от существования в строке k-го номера слева от элемента с номером $(k,\ k - 1)$ отличного от нуля элемента. Пусть $a_{ki} \neq 0$ при $(i < k - 1)$. Тогда простой перестановкой столбцов можно вновь перейти к регулярному случаю $\boldsymbol{T}\boldsymbol{A}^{(n-k)}$, где

$$T = \begin{pmatrix} 1 & 0 & (i) & & & & (k-1) & & \\ & \ddots & \vdots & & & & \vdots & & \\ 0 & \cdots & 1 & \vdots & & & \vdots & & \\ \cdots & & \ddots & \vdots & & & \vdots & & \\ & & & 1 & \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots & & 1 & \cdots\ \cdots\ \cdots\ \cdots & (i) \\ & & & \vdots & \ddots & & \vdots & & \\ & & & \vdots & 0 & & \vdots & & \\ & & & \vdots & & \ddots & \vdots & & \\ & & & \vdots & & 1 & \vdots & & \\ & & & \vdots & & & \ddots & \vdots & \\ & & & \vdots & & & 1 & \vdots & \\ & & & \vdots & & & & \ddots & \vdots \\ & & & 1 & \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots & & 0 & \cdots\ \cdots\ \cdots\ \cdots & (k-1) \\ & & & & & & & \ddots & \\ & & & & & & & 1\ \ 0 & \\ & & & & & & & \ddots\ \ddots & \\ & & & & & & & & 1\ \ 0 \end{pmatrix}$$

Легко убедиться, что $\boldsymbol{T}\boldsymbol{A}^{(n-k)}\boldsymbol{T}$ есть преобразование подобия, так как $(\boldsymbol{T})^2 = \boldsymbol{E}$; $(\boldsymbol{T})^{-1} = \boldsymbol{T}$. После этого преобразования вычисления продолжаются, как и в регулярном случае.

Более интересен случай, когда все элементы строки k, левее a_{kk-1} на $(n\text{-}k)$ шаге равны нулю, т.е. невозможно подобрать очередной элемент непреобразованной части строки, на который будет выполняться деление.

Матрица, которая должна быть преобразована на очередном шаге вычислительной схемы, условно делится на четыре блока, причем в силу особенностей строения этой матрицы один из блоков будет состоять исключительно из нулевых членов, т.е. будет нулевой матрицей $\boldsymbol{O}$, второй будет матрицей Фробениуса $\boldsymbol{\Phi}$, а два оставшихся - обычными матрицами, которые обозначим $\boldsymbol{B}$ и $\boldsymbol{C}$ (очевидно, что все матрицы ранга n - k):

$$\boldsymbol{A}_{n-k} = \left(\begin{array}{cccc|ccccc} a_{11}^{(n-k)} & a_{12}^{(n-k)} & \cdots & a_{1k-1}^{(n-k)} & a_{1k}^{(n-k)} & \cdots & a_{1n-1}^{(n-k)} & a_{1n}^{(n-k)} \\ \vdots & & \cdots & & & & & \\ a_{k-1,1}^{(n-k)} & a_{k-1,2}^{(n-k)} & \cdots & a_{k-1,k-1}^{(n-k)} & a_{k-1,k}^{(n-k)} & \cdots & a_{k-1,n-1}^{(n-l)} & a_{k-1,n}^{(n-k)} \\ --- & --- & --- & --- & --- & --- & --- & --- \\ 0 & 0 & \cdots & 0 & a_{kn}^{(n-k)} & \cdots & a_{k,n-1}^{(n-k)} & a_{kn}^{(n-k)} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 & 0 \end{array}\right) = \begin{pmatrix} \boldsymbol{B}_{n-k} & \boldsymbol{C}_{n-k} \\ \boldsymbol{0} & \Phi_{n-k} \end{pmatrix}.$$

Тогда в силу теоремы Лапласа о разложении определителя [Курош, 1962] имеет место равенство (в правой части индексами обозначены порядки единичных матриц)

$$|\boldsymbol{A}^{(n-k)} - \lambda\boldsymbol{E}| = |\boldsymbol{B}^{(n-k)} - \lambda\boldsymbol{E}_{k-1}|\,|\boldsymbol{\Phi}^{(n-k)} - \lambda\boldsymbol{E}_{n-k+1}|.$$

А так как $\boldsymbol{\Phi}^{(n-k)}$ есть матрица Фробениуса, то ее характеристический многочлен выписывается по виду первой строки. Остается только привести $\boldsymbol{B}^{(n-k)}$ к каноническому виду Фробениуса уже изложенным методом.

Можно подсчитать, что в регулярном случае необходимо для вычисления характеристического многочлена выполнить n^3 действий, поэтому метод Данилевского относят к самым эффективным.

Для повышения точности вычислений на каждом шаге преобразований при помощи перестановки строк или столбцов выбирают наибольший элемент. Для контроля вычислений на каждом шаге проверяют след матрицы, который не должен изменяться.

Если найдены все собственные значения λ_i матрицы $\boldsymbol{A}$ и известна неособенная матрица $\boldsymbol{S}$, то в методе Данилевского, как и в методе Крылова, для определения собственных значений матрицы $\boldsymbol{A}$ можно обойтись без решения системы однородных линейных алгебраических уравнений $\boldsymbol{A}^{(\Phi)}\boldsymbol{X} = \lambda_i\boldsymbol{X}$ и использовать уже известную матрицу $\boldsymbol{S}$:

$$\boldsymbol{S} = \boldsymbol{M}_{n-1}\boldsymbol{M}_{n-2}\ldots\boldsymbol{M}_2\boldsymbol{M}_1. \qquad (1.40)$$

И хотя собственные значения матрицы $\boldsymbol{A}^{(\Phi)}$ и $\boldsymbol{A}$ различны, они имеют одинаковый спектр [Курош, 1962], следовательно, между собственными значениями матрицы $\boldsymbol{A}^{(\Phi)}$ и $\boldsymbol{A}$ существует связь, основанная на преобразовании подобия матриц [Курош, 1962]. Если вектор $\boldsymbol{X}$ есть собственный вектор матрицы $\boldsymbol{A}^{(\Phi)}$, принадлежащий собственному значению λ_i, а вектор $\boldsymbol{Y}$ - собственный вектор подобной ей матрицы $\boldsymbol{\Phi} = \boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S}$, принадлежащий тому же собственному значению λ_i, то вектор $\boldsymbol{S}\boldsymbol{Y}$ также будет собственным вектором матрицы $\boldsymbol{A}$, соответствующим собственному значению λ_i.

В этом утверждении можно легко убедиться. Пусть $\boldsymbol{Y}$ - собственный вектор матрицы $\boldsymbol{\Phi}$, тогда

$$\boldsymbol{\Phi}\boldsymbol{Y} = \lambda\,\boldsymbol{Y}; \qquad \boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S}\,\boldsymbol{Y} = \lambda\,\boldsymbol{Y};$$
$$\boldsymbol{S}\,\boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S}\,\boldsymbol{Y} = \boldsymbol{S}\,\lambda\,\boldsymbol{Y}; \qquad \boldsymbol{A}\boldsymbol{S}\,\boldsymbol{Y} = \boldsymbol{S}\,\lambda\,\boldsymbol{Y};$$
$$\boldsymbol{A}\boldsymbol{S}\,\boldsymbol{Y} = \lambda\boldsymbol{S}\,\boldsymbol{Y}; \qquad \boldsymbol{A}\,\boldsymbol{X} = \lambda\,\boldsymbol{X}.$$

Таким образом, собственные векторы исходной матрицы $\boldsymbol{A}$ легко находятся по собственным значениям матрицы Фробениуса $\boldsymbol{\Phi}\boldsymbol{Y} = \lambda\boldsymbol{Y}$ или, записав непосредственно, получим:

$$\begin{cases} \lambda y_1 = p_1 y_1 + p_2 y_2 + \ldots + p_n y_n; \\ \lambda y_2 = y_1; \\ \lambda y_3 = y_2; \\ \ldots \\ \lambda y_{n-1} = y_n. \end{cases} \qquad (1.41)$$

И так как собственный вектор определяется с точностью до полиномного множителя, положим $y_n = 1$. Тогда из системы (1.41) легко найти вектор $\boldsymbol{Y}$, а первое уравнение при этом используют исключительно для контроля вычислений. Вектор $\boldsymbol{X}$ матрицы $\boldsymbol{A}$ вычисляется по формуле

$$\boldsymbol{X} = \boldsymbol{S}\boldsymbol{Y} = \boldsymbol{M}_{n-1}\boldsymbol{M}_{n-2}\ldots\boldsymbol{M}_2\boldsymbol{M}_1\boldsymbol{Y}. \qquad (1.42)$$

Заметим, что не обязательно предварительно перемножать матрицы $\boldsymbol{M}$, удобнее последовательно умножать $\boldsymbol{Y}$ на $\boldsymbol{M}_1$, $\boldsymbol{M}_2$, ..., $\boldsymbol{M}_{n-1}$, при этом от умножения на $\boldsymbol{M}_i$ у вектора $\boldsymbol{Y}$ будет изменяться только i-я координата.

Если же случай нерегулярный, то этим приемом непосредственно пользоваться нельзя, надо предварительно вычислить матрицу $\boldsymbol{S}$ полностью.

Подпрограмма DANIL предназначена для вычисления собственных значений λ_i матрицы $\boldsymbol{A}$, положительно определенной, эрмитовой по методу Данилевского. Собствен-

ные значения λ_i не упорядочены по возрастанию. Подпрограмма VECTD предназначена для вычисления собственных векторов $\boldsymbol{X}$ матрицы $\boldsymbol{A}$, соответствующих собственным значениям λ_i. Векторы записывают в столбцы матрицы X, они также не упорядочены по возрастанию λ_i.

Вычислительный алгоритм подпрограммы DANIL следующий:

Øàã 1. Строится единичная матрица.

Øàã 2. Определяется очередной номер строки (последняя строка предыдущего преобразования). Для первого шага i полагается равным n, а далее уменьшается на единицу с каждым шагом.

Øàã 3. Формируется (i - 1)-я строка матрицы $\boldsymbol{S}$ как результат деления $a_{ij} / a_{ij_{-1}}$, где j - номер столбца, i - номер строки (по формуле 1.40).

Øàã 4. Формируется (i - 1)-я строка матрицы $\boldsymbol{M}$ как a_{ij} (по формуле 1.38).

Øàã 5. умножается $\boldsymbol{A}$ слева на $\boldsymbol{M}_i$, справа на $\boldsymbol{M}_i^{-1}$, т.е. $\boldsymbol{A}_i = \boldsymbol{M}_i^{-1}\boldsymbol{A}\boldsymbol{M}$.

Øàã 6. формируется (i - 1)-я строка матрицы $\boldsymbol{S}$ (по формуле 1.40) как (i - 1)-я строка матрицы $\boldsymbol{M}$ (так как не умножается!).

Øàã 7. Если $i > 1$, то повторить с шага 1.

Øàã 8. Решаем характеристическое уравнение (можно воспользоваться программой из п. 4.1).

```
PROCEDURE DANIL (CONST N:INTEGER;
                 VAR A,M:MAS1;  VAR B: MAS11);
TYPE MAS = ARRAY [1..N] OF REAL;
                 MAS1 = ARRAY [1..N] OF MAS;
VAR I, J, NS,NSP1, K : INTEGER;
                 X : MAS11; AIJ : REAL;
                 R, E, S, S1, A1 : MAS1;
BEGIN
  FOR I := 1 TO N DO
    {□□□□□□□□□□ □□□□□□□□□□ □□□□□□□□□}
     FOR J := 1 TO N DO      {□□□□□□□□  }
     BEGIN
         IF  I=J THEN
                  S[I,J] := 1
             ELSE
                  S[I,J] := 0.0;
         IF I=J THEN
                  M[I,J] := 1
             ELSE
                  M[I,J] := 0.0;
         A1[I,J] := A[I,J];
     END;
    {□□□□□□□□□□□ □□□□□□□□□□ □□□□□□□□}
  NS := N- 1;
  E:= S;
  REPEAT
     S := E;
     NSP1 := NS + 1;
     S1 := E;
     FOR I := 1 TO N DO
        S[NS,I] := -A1[NSP1,I] / A1[NSP1,NS];
     S[NS,NS] := 1 / A1[NSP1,NS];
     FOR I := 1 TO N DO
     BEGIN
        M[NS,I] := S[NS,I];
        S1[NS,I] := A1[NSP1,I];
     END;
     FOR I := 1 TO N DO
           { □□□□□□□□□□□□□ □□□□□□□□ □   }
        FOR J := 1 TO N DO
        BEGIN
           AIJ := 0.0;
           FOR K := 1 TO N DO
           AIJ := AIJ + A1[I,K] * S[K,J];
           R[I,J] := AIJ;
        END;
     FOR I := 1 TO N DO
     FOR J := 1 TO N DO
        BEGIN
           AIJ := 0.0;
           FOR K := 1 TO N DO
              AIJ := AIJ + S1[I,K]*R[K,J];
           A1[I,J] := AIJ;
        END;
     DEC (NS);
  UNTIL NS = 0;
{ □□□□□□□□□□□ □□□□□□□□□□□ □□□□□□□□ }
   FOR I := 2 TO N+1 DO
      B[I] :=-A1 [1,I-1];
   B[1] := 1.0;
  LAMBDA (4,0.0000001,B,X);
  B := X;
END.
```

Формальные параметры процедуры. *Входные*: N (тип ***integer***) - размерность матрицы $\boldsymbol{A}$, для которой ищутся собственные значения λ; A (тип ***real***) - матрица $\boldsymbol{A}$ (исходная) размером ($n \times n$). *Выходные*: A (тип ***real***) - матрица Фробениуса; M (тип ***real***) - массив, содержащий строки, отличные от нуля и единицы матрицы преобразований $\boldsymbol{M}_{n-1}$; B (тип ***real***) - массив собственных значений матрицы $\boldsymbol{A}$. Процедура DANIL в работе обращается к подпрограмме LAMBDA, описанной в п. 4.1.

Вычислительный алгоритм процедуры VECT:

Øàã 1. Вычисляется вспомогательный вектор $\boldsymbol{Y}$, соответствующий очередному значению λ (по формуле 1.41).

Øàã 2. Вычисляется собственный вектор матрицы $\boldsymbol{A}$ и заносится в соответствующий столбец массива V (по формуле 1.42).

Øàã 3. Если выбраны не все значения λ, то переход на шаг 1, иначе - конец подпрограммы.

```
PROCEDURE VECT (CONST N: INTEGER; X: MAS11;
                 M:MAS1;  VAR V : MAS1);
TYPE MAS = ARRAY [1..N] OF REAL;
```

```
VAR Y : MAS;  I,J, K, NS : INTEGER;  SUM : REAL;
BEGIN
  NS := 1;
  REPEAT
     Y[N] := 1;
     FOR I := N-1 DOWNTO 1 DO
           Y[I] := Y[I+1]*X[NS];
     FOR I := 1 TO N DO
     BEGIN
        SUM := 0.0;
        FOR J := 1 TO N DO
           SUM := SUM + M[I,J]*Y[J];
        Y[I] := SUM;
     END;
     FOR I := 1 TO N DO
     V[I,NS] := Y[I];
     INC (NS);
  UNTIL NS>N;
END.
```

Формальные параметры процедуры. *Входные*: *M*1 (тип ***integer***) - начальный номер собственного значения λ матрицы ***A***, начиная с которого надо искать собственные векторы матрицы ***A;*** *M*2 (тип ***integer***) - последний номер собственного значения λ матрицы ***A,*** до которого надо искать собственные векторы (очевидно, что *M*1 < *M*2); *M* (тип ***real***) - вспомогательный массив, содержащий строки матрицы преобразований *M,* используемый для пересчета собственного вектора матрицы Фробениуса в собственный вектор матрицы ***A***; *B* (тип ***real***) - массив собственных значений матрицы ***A***. *Выходные*: *V* (тип ***real***) - массив собственных векторов матрицы ***A***. Если определяются не все λ, то собственные векторы располагаются с *M*1 по *M*2 столбец, а остальные значения массива *V* не определены.

Примечание. В процедуре VECT отсутствует контроль вычислений собственных значений.

Для проверки работы программы (см. табл. 1.19) собственные значения и собственные векторы матрицы, взятой из п. 4.1, находятся с точностью до 0.0001. Все промежуточные матрицы для контроля за работой программы выписаны.

Таблица 1.19

Параметр	Номер элемента [1]	[2]	[3]	[4]
		Итерация первая		
Матрица ***A***	0.533333	-1.416667	1.833333	-0.140000
	-0.310000	-3.470000	4.500000	0.260000
	-0.221133	-3.097133	4.146667	0.166800
	0.000000	0.000000	1.000000	0.000000
Матрица $(\boldsymbol{M}_3)^{-1}$	1.000000	0.000000	0.000000	0.000000
	0.000000	1.000000	0.000000	0.000000
	0.080000	0.080000	0.060000	0.120000
	0.000000	0.000000	0.000000	1.000000
Матрица $\boldsymbol{M}_3$	1.000000	0.000000	0.000000	0.000000
	0.000000	1.000000	0.000000	0.000000
	-1.333333	-13.333333	16.666667	-2.000000
	0.000000	0.000000	0.000000	1.000000
		Итерация вторая		
Матрица ***A***	0.634482	0.457412	-0.063403	-0.2162961
	0.052473	0.575518	0.632654	-0.178628
	0.000000	1.000000	0.000000	0.000000
	0.000000	0.000000	1.000000	0.000000
Матрица $(\boldsymbol{M}_2)^{-1}$	1.000000	0.000000	0.000000	0.000000
	-0.221133	-3.097133	4.146667	0.166800
	0.000000	0.000000	1.000000	0.000000
	0.000000	0.000000	0.000000	1.000000
Матрица $\boldsymbol{M}_2$	1.000000	0.000000	0.000000	0.000000
	-0.071399	-0.322879	1.338873	0.053856
	0.000000	0.000000	1.000000	0.000000
	0.000000	0.000000	0.000000	1.000000
		Итерация третья		
Матрица ***A***	1.210000	0.291500	-0.583363	0.101987
	1.000000	0.000000	0.000000	0.000000

	0.000000	1.000000	0.000000	0.000000
	0.000000	0.000000	1.000000	0.000000

Окончание таблицы 1.19

Параметр	Номер элемента			
	[1]	[2]	[3]	[4]
Матрица $(\boldsymbol{M}_1)^{-1}$	0.052473	0.575518	0.632654	-0.178628
	0.000000	1.000000	0.000000	0.000000
	0.000000	0.000000	1.000000	0.000000
	0.000000	0.000000	0.000000	1.000000
Матрица $\boldsymbol{M}_1$	19.057255	-10.967785	-12.056645	3.404166
	0.000000	1.000000	0.000000	0.000000
	0.000000	0.000000	1.000000	0.000000
	0.000000	0.000000	0.000000	1.000000
Собственные значения матрицы $\boldsymbol{A}$	0.214779	-0.699093	1.043229	0.651085
Собственные векторы матрицы $\boldsymbol{A}$ (не нормированные)	0.497526	-0.038678	0.526914	-3.835256
	0.291000	-1.037180	1.061588	1.062538
	-2.963725	0.229082	0.530102	-0.202091
	1.000000	1.000000	1.000000	1.000000
Собственные векторы матрицы $\boldsymbol{A}$ (нормированные)	-0.167872	0.037291	0.496345	1.000000
	-0.098187	1.000000	1.000000	-0.277045
	1.000000	-0.220871	0.499348	0.052693
	-0.337413	-0.964153	0.941985	-0.260739

4.3. ВЫЧИСЛЕНИЕ СОБСТВЕННЫХ ВЕКТОРОВ И СОБСТВЕННЫХ ЗНАЧЕНИЙ СИММЕТРИЧЕСКОЙ МАТРИЦЫ МЕТОДОМ ЯКОБИ

Всякая симметричная матрица $\boldsymbol{A}$ может быть записана в виде

$$\boldsymbol{A} = \boldsymbol{V}^*\boldsymbol{D}\boldsymbol{V}, \qquad (1.43)$$

где $\boldsymbol{V}^*$- ортогональная матрица; $\boldsymbol{D}$ - диагональная с элементами $\lambda_1, \lambda_2, ..., \lambda_m$ - собственными значениями матрицы $\boldsymbol{A}$. После умножения (1.43) слева на $\boldsymbol{V}^*$ получим $\boldsymbol{A}\boldsymbol{V}^* = \boldsymbol{V}^*\boldsymbol{D}$, и если расписать это матричное равенство по столбцам, то окажется, что каждый i-й столбец матрицы $\boldsymbol{V}^*$ является собственным вектором, отвечающим собственному значению λ_i [Бахвалов, 1973а]. Преобразуя систему (1.43) к виду

$$\boldsymbol{V}\boldsymbol{A}\boldsymbol{V}^* = \boldsymbol{D},$$

построим последовательность ортогональных матриц $\boldsymbol{V}_1, ..., \boldsymbol{V}_n$ так, чтобы при $\boldsymbol{W}_n = \boldsymbol{V}_n ... \boldsymbol{V}_1$, $n \to \infty$ иметь $\boldsymbol{W}_n \boldsymbol{A} \boldsymbol{W}_n^* \to \to \boldsymbol{D}$. Если теперь обозначить через $\boldsymbol{V}_{kl}(\phi)$ матрицу с элементами $v_{kk} = v_{ll} = \cos\phi$, $v_{kl} = -v_{lk} = \sin\phi$, $v_{ii} = 1$ при $i \neq k$, $i \neq 1$, $v_{ij} = 0$ при остальных (i, j), то она будет ортогональной.

Построим последовательность матриц $\boldsymbol{V}_n$ и $a^{(n)} = \left[a_{ij}^{(n)} \right]$ по следующему правилу:

$$\boldsymbol{A}^{(1)} = \boldsymbol{V}_1 \boldsymbol{A} \boldsymbol{V}_1^*, ..., \boldsymbol{A}^{(n)} = \boldsymbol{V}_n \boldsymbol{A}^{(n-1)} \boldsymbol{V}_n^*. \qquad (1.44)$$

Матрица

$$V_n = B_{kl}[\phi] = \begin{pmatrix} b_{kk}(\phi) & b_{kl}(\phi) \\ b_{lk}(\phi) & b_{ll}(\phi) \end{pmatrix}_{\phi = 1/2 \cdot \mathrm{arctg}\left(\frac{2 \cdot b_{kl}}{b_{kk} - b_{ll}} \right)}$$

строится в зависимости от матрицы $\boldsymbol{A}^{(n)}$ так, чтобы выполнялось неравенство [Бахвалов, 1973а]

$$\sum_{i \neq j} \left(a_{ij}^{(n)} \right)^2 \leq \left(1 - \frac{2}{m(m-1)} \right) \cdot \sum_{i \neq j} \left(a_{ij}^{(n-1)} \right)^2. \qquad (1.45)$$

Выражая каждую матрицу $\boldsymbol{A}^{(n)}$ через предыдущую, будем строить итерационную последовательность $\boldsymbol{A}^{(n)} = \boldsymbol{W}\boldsymbol{A}\boldsymbol{W}_n^*$. Согласно оценке (1.45), имеем

$$\sum_{i \neq j} \left(a_{ij}^{(n)} \right)^2 \leq \left(1 - \frac{2}{m(m-1)} \right)^n \cdot \sum_{i \neq j} (a_{ij})^2 \cdot$$

Пусть матрица $\boldsymbol{D}^{(n)}$ получена из $\boldsymbol{A}^{(n)}$ простой заменой всех недиагональных элементов на нули. Тогда нетрудно доказать [Бахвалов, 1973а], что

$$\left| \frac{(A_{x,x}^{(n)})}{(x,x)} - \frac{(D_{x,x}^{(n)})}{(x,x)} \right| \leq \rho_n = \sqrt{\sum_{i \neq j} (a_{ij}^{(n)})^2}, \qquad (1.46)$$

откуда вытекает $|\lambda_n^{(n)} - \lambda_n| \leq \rho_n$ при всех n, где λ_i - собственные значения матрицы $\boldsymbol{A}^{(n)}$ (значит, и матрицы $\boldsymbol{A}$), занумерованные в порядке убывания, а $\lambda_n^{(n)}$ - собственные значения матрицы $\boldsymbol{D}^{(n)}$, т.е. ее диагональные элементы, также занумерованные в порядке убывания; r_n - верхняя оценка погрешности.

Столбцы матрицы $\boldsymbol{W}^{(n)}$ оказываются приближениями к собственным векторам матрицы $\boldsymbol{A}^{(n)}$. Все столбцы произведения матриц $\boldsymbol{V}(\phi_{kl})$ и $\boldsymbol{A}_{kl}$, кроме k-го и первого, совпа-

дают с соответствующими столбцами матрицы $\boldsymbol{A}^{(n)}$, k-й и первый столбцы новой матрицы являются линейными комбинациями этих же столбцов матрицы $\boldsymbol{A}$. Такого же количества действий потребует дальнейшее умножение на матрицу $\boldsymbol{V}^{*}(\phi_{kl})$ в соответствии с формулами (1.44), когда происходит изменение k-й и первой строк.

Общее количество арифметических операций для получения каждой последующей матрицы $\boldsymbol{A}^{(n)}$ составит при этом $\sim 6n$.

Рассмотренный алгоритм (с некоторой модификацией, предложенной в работе [Greenstadt, 1961]) реализован в процедуре EIGJAC, используя которую можно вычислить все собственные значения и собственные векторы заданной симметричной матрицы $\boldsymbol{A}[1{:}n,\ 1{:}n]$. На выходе из процедуры l-й столбец матрицы $\boldsymbol{S}$ (начальное значение массива $\boldsymbol{S}$ несущественно) содержит первый из n собственных векторов данной матрицы, диагональный элемент $a[\mathrm{l},\ \mathrm{l}]$ массива $\boldsymbol{A}$ соответствует первому собственному значению. Итерационный процесс заканчивается, когда для каждого недиагонального элемента $a[i, j]$ выполняется неравенство

$$\left|a_{ij}\right| \le (\varepsilon / n)\sqrt{\sum_{i=2}^{n}\sum_{j=1}^{i-1} 2\left(a_{ij}\right)^2}\ ,$$

где параметр ε, связанный с оценкой погрешности r_n в формуле (1.46) и введенный вместо r_n для оценки качества решения в работе [Greenstadt, 1961], и есть задаваемая точность.

Формальные параметры процедуры. *Входные*: n (тип ***integer***) - порядок матрицы $\boldsymbol{A}$; *eps* (тип ***real***) - задаваемая точность; $A[1{:}n,1{:}n]$ (тип ***real***) - исходная симметричная матрица. *Выходные*: $A[1{:}n,\ 1{:}n]$ (тип ***real***) - матрица, диагональные элементы которой $A[i,\ i]$ являются соответственно λ_i-ми собственными значениями; $S[1{:}n,\ 1{:}n]$ (тип ***real***) - матрица, k-й столбец которой содержит k-й из n собственных векторов исходной матрицы $\boldsymbol{A}$.

```
PROCEDURE EIGJAC (N:INTEGER;EPS:REAL;
                  VAR A, S:MAS1);
VAR  NORM1,NORM2,THR,MU,OMEGA,SINT,COST,
     INT1,V1, V2,V3,T : REAL;
     I,J,P,Q,IND : INTEGER;
LABEL MAIN, MAIN1;
BEGIN
                  { *** □□□□□□□□□□□□□ □ □□□□□□□□ S
                    □□□□□□□□□□ □□□□□□□□ *** }
  FOR I:=1 TO N DO
    FOR J:=1 TO I DO
    IF I=J THEN
       S[I,J]:=1
    ELSE
    BEGIN
       S[I,J]:=0;
       S[J,I]:=0;
    END;
{ *** □□□□□□□□□□□ □□□□□□□□□□ □□□□□ NORM1,
 □□□□□□□□ □□□□□ NORM2 □ □□□□□□ THR    *** };
    INT1:=0;
    FOR I:=2 TO N DO
      FOR J:=1 TO I-1 DO
        INT1:=INT1+2*A[I,J]*A[I,J];
    IF INT1=0 THEN
      EXIT;
    NORM1:=SQRT(INT1);
    THR:=SQRT(INT1);
    NORM2:=(EPS/N)*NORM1;
    IND:=0;
MAIN:
    THR:=THR/N;
    {**□□□□□□ □□□□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□□□**};
MAIN1:
    FOR Q:=2 TO N DO
      FOR P:=1 TO Q-1 DO
        IF ABS(A[P,Q]) >= THR THEN
        BEGIN
          IND:=1;
          V1:=A[P,P];
          V2:=A[P,Q];
          V3:=A[Q,Q];
          MU:=0.5*(V1-V3);
        IF MU=0 THEN
            OMEGA:=-1
          ELSE
            OMEGA:=-SIGN(MU)*
              V2/SQRT(V2*V2+MU*MU);
          SINT:=OMEGA/SQRT(2*(1+
              SQRT(1-OMEGA*OMEGA));
          COST:=SQRT(1-SINT*SINT);
          FOR I:=1 TO N DO
          BEGIN
            IF I<>P AND I<>Q THEN
          BEGIN
              INT1:=A[I,P];
              MU:=A[I,Q];
            A[Q,I]:=INT1*SINT+MU*COST;
            A[I,Q]:=INT1*SINT+MU*COST;
              A[P,I]:=INT1*COST-MU*SINT;
            A[I,P]:=INT1*COST-MU*SINT;
            END;
          INT1:=S[I,P];
          MU:=S[I,Q];
        S[I,Q]:=INT1*SINT+MU*COST;
          S[I,P]:=INT1*COST-MU*SINT
        END {***I***};
        MU:=SINT*SINT;
        OMEGA:=COST*COST;
        INT1:=SINT*COST;
      T:=2*V2*INT1;
        A[P,P]:=V1*OMEGA+V3*MU-T;
      A[P,P]:=V1*MU+V3*OMEGA+T;
      A[P,Q]:=(V1-V3)*INT1+V2*(OMEGA-MU);
```

```
        A[Q,P]:=(V1-V3)*INT1+V2*(OMEGA-MU);
    END ;
            {***□□□□□□□□□ □□□□□□□□□□□ □□□□□□□□□
                    □□□□□□□□□***};
    IF IND=1 THEN
    BEGIN
        IND:=0;
        GO TO MAIN1
    END;
  IF THR > NORM2 THEN GO TO MAIN;
END.
```

Процедура EIGJAC получена из процедуры JACOBI [Агеев, 1976] переводом последней с языка *ALGOL* на язык *PASCAL*, при этом было сделано несколько тождественных преобразований, оптимизирующих затраты машинного времени. Процедура EIGJAC была проверена для тех же исходных данных, что и процедура JACOBI [Агеев, 1976]: $n = 4$, $eps=10^{-8}$ и

$$a = \begin{pmatrix} 1.00 & 0.42 & 0.54 & 0.66 \\ 0.42 & 1.00 & 0.32 & 0.44 \\ 0.54 & 0.53 & 1.00 & 0.22 \\ 0.66 & 0.44 & 0.22 & 1.00 \end{pmatrix}.$$

Вычисления, выполненные на IBM PC/AT-386 с частотой 40 МГц, дали следующие результаты, которые совпадают с результатами тестирования

$$S = \begin{pmatrix} 0.579642502 & -0.050328450 & -0.718845953 & 0.380449881 \\ 0.459996665 & 0.237226458 & -0.095698981 & -0.850275473 \\ 0.433459111 & -0.812846170 & 0.387435463 & -0.035889606 \\ 0.514325614 & 0.529595844 & 0.569206432 & 0.361941215 \end{pmatrix};$$

$$a = \begin{pmatrix} 2.32274880 & 0.1000514e-11 & 0.4799251e-14 & 0 \\ 0.1000514e-11 & 0.796706689 & 0 & 0.1795160e-14 \\ 0.4799251e-14 & 0 & 0.242260708 & 0 \\ 0 & 0.1795162e-14 & 0 & 0.638283803 \end{pmatrix},$$

приведенными в работе Агеева [Агеев и др., 1976], и имеют максимальную погрешность по отношению к собственным значениям λ_i и собственным векторам $\boldsymbol{X}$ исходной матрицы [в сравнении с их величинами из работы Фадеева и др. (1963)] max $r_i = 4*10^{-8}$, max$S_i = 3*10^{-6}$.

4.4. ЗАДАЧА ОБРАЩЕНИЯ МАТРИЦ И ВЫЧИСЛЕНИЯ ГЛАВНОГО ОПРЕДЕЛИТЕЛЯ ПО СХЕМЕ ГАУССА

Метод Гаусса, подробно рассмотренный в п. 3.1 и 3.2, с успехом может быть использован для вычисления определителей.

Пусть дана неособенная матрица (1.33) и надо найти обратную матрицу и det$\boldsymbol{A}$. Если матрица (1.33) имеет ранг больше трех, то даже задача нахождения детерминанта не является простой задачей. Надо отметить, что данный метод нельзя применять, если исходная матрица особенная. Рассмотрим линейную однородную систему уравнений $\boldsymbol{Ax} = \boldsymbol{0}$, при решении которой матрица $\boldsymbol{A}$ заменяется верхней треугольной матрицей $\boldsymbol{B}$:

$$\boldsymbol{B} = \begin{pmatrix} 1 & b_{12} & \cdots & b_{1n} \\ 0 & 1 & \cdots & b_{2n} \\ \vdots & & & \\ 0 & 0 & \cdots & 1 \end{pmatrix},$$

тогда уравнение преобразуется к виду $\boldsymbol{Bx} = 0$. Элементы матрицы $\boldsymbol{B}$ получаются из элементов матрицы $\boldsymbol{A}$ по формулам единственного деления (1.26). Но заметим, что деление на ведущий элемент матрицы эквивалентно выносу за определитель сомножителя a_{ii}, тогда

$$\det\boldsymbol{A} = a_{11}\det\boldsymbol{A}^{(1)} = \\ = a_{11}a_{22}\det\boldsymbol{A}^{(2)} = \ldots = a_{11}a_{22}\ldots a_{nn}\det\boldsymbol{B},$$

но det $\boldsymbol{B} = 1$, следовательно

$$\det \boldsymbol{A} = a_{11}a_{22}\ldots a_{nn}, \qquad (1.47)$$

т.е. определитель матрицы равен произведению "ведущих" элементов для соответствующей схемы Гаусса. Если при приведении матрицы $\boldsymbol{A}$ к треугольному виду использовать модифицированный метод Гаусса с выбором главного элемента, то формула (1.47) примет следующей вид:

$$\det \boldsymbol{A} = (-1)^k a_{11}a_{22}\ldots a_{nn}, \qquad (1.48)$$

где k - количество перестановок строк при реализации метода.

Очень часто метод Гаусса применяют при решении задачи обращения матриц. Обозначим элементы обратной матрицы через a_{ij}, тогда задача обращения матрицы $\boldsymbol{A}$ сводится к решению системы линейных уравнений $a_{ij} \cdot a_{jk} = d_{ik}$, или в матричной записи $\boldsymbol{AA}^{-1} = \boldsymbol{E}$, где $\boldsymbol{E}$ - единичная матрица. Решая систему уравнений методом Гаусса - Жордана, легко получаем обратную матрицу:

$$\left(\begin{array}{cccc|cccc} a_{11} & a_{12} & \cdots & a_{1n} & 1 & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & a_{2n} & 0 & 1 & \cdots & 0 \\ \vdots & & & & \vdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} & 0 & 0 & \cdots & 1 \end{array}\right) \Rightarrow$$

$$\Rightarrow \left(\begin{array}{cccc|cccc} 1 & a_{12}^{(k)} & \cdots & a_{1n}^{(k)} & b_{11}^{(k)} & 0 & \cdots & 0 \\ 0 & 1 & \cdots & a_{2n}^{(k)} & b_{21}^{(k)} & b_{22}^{(k)} & \cdots & 0 \\ \vdots & & & & \vdots & & & \\ 0 & 0 & \cdots & 1 & b_{n1}^{(k)} & b_{n2}^{(k)} & \cdots & b_{nn}^{(k)} \end{array}\right) \Rightarrow$$

$$\Rightarrow \left(\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & b_{11} & b_{12} & \cdots & b_{1n} \\ 0 & 1 & \cdots & 0 & b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & & & & \vdots & & & \\ 0 & 0 & \cdots & 1 & b_{n1} & b_{n2} & \cdots & b_{nn} \end{array}\right).$$

Правая часть преобразованной матрицы и будет являться искомой обратной матрицей A^{-1}.

Как уже указывалось, метод Гаусса является очень распространенным вычислительным методом, и поэтому практически любая БСП имеет вычислительную процедуру, реализующую метод Гаусса. Рассмотрим одну из таких процедур ЕС-1066 для транслятора *FORTRAN-77*, несколько модифицированную для обращения матрицы и вычисления определителя (перевод текста процедуры на язык *PASCAL* выполнен авторами):

```
PROCEDURE GAUS_OBR (A : MAS; VAR V : MAS;
                    VAR TOL : INTEGER; VAR DET : REAL);
TYPE MST = ARRAY [1..N*2]OF REAL;
                    MSS = ARRAY [1..N] OF MST;
VAR A1 : MSS;   I, J, M, K : INTEGER;  H : REAL;
BEGIN
  FOR I := 1 TO N DO
        { □□□□□□□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□□□ Д1   }
      FOR J := 1 TO N*2 DO
      IF J<= N THEN
          A1 [I,J] := A[I,J]
      ELSE
          IF I=J-N THEN
              A1  [I,J] := 1.0
          ELSE
              A1[I,J] :=0.0;
  TOL := 0;
  DET := 1;
{ □□□□□□□□□ □□□□□□□□ □□NS□-□□□}
  IF N > 50 THEN
  BEGIN
      TOL := 2;
      EXIT;
  END;
  IF ABS(A1[N,N]) < 1.0E-10 THEN
  BEGIN
      TOL := 1;
      EXIT;
  END;
  FOR I := 1 TO N DO
  BEGIN
      IF ABS(A1[I,I])<1.0E-10 THEN
      BEGIN TOL:= 1;   EXIT;   END;
      H := A1[I,I];
      DET := DET * H;
      FOR J := I TO N*2 DO
          A1[I,J] := A1[I,J] / H;
      IF I<N THEN
      FOR J := I+1 TO N DO
      BEGIN
          H := A1[J,I];
      FOR K := 1 TO N*2 DO
              A1[J,K] := A1 [J,K] - H*A1[I,K];
      END;
  END;
  FOR I := N DOWNTO  1 DO
  FOR K := I-1 DOWNTO 1 DO
  BEGIN
      H := A1[K,I];
      FOR J:=1 TO 2*N DO
          A1[K,J] := A1[K,J] - A1[I,J]*H;
  END;
  FOR I := 1 TO N DO
      FOR J := N+1 TO N*2 DO
          V [I,J-N] := A1[I,J];
END.
```

Формальные параметры процедуры. *Входные*: *A* (тип ***real***) - матрица, для которой ищется обратная; *N* (тип ***integer***) - размерность матрицы (матрица должна быть квадратной, т.е. размером *N*×*N*). *Выходные*: *tol* (тип ***real***) - целое число, которое при нормальном завершении процедуры равно 0. Если при обращении матрицы один из диагональных элементов стал равен 0, то процедура возвращает значение данного параметра 1; ***V*** (тип ***real***) - обратная для ***A*** матрица; *SUM*- определитель матрицы.

В процедуре GAUS_OBR вводится вспомогательная матрица ***A***1, размером [*N*×2**N*], в которой первая половина равна введенной матрице ***A***, а вторая (от *N*+1 до *N**2) - единичной. Выполняя преобразования над первой половиной матрицы ***A***1 (от 1 до *N*) для приведения ее к виду единичной матрицы, получаем во второй половине матрицы ***A***1 матрицу, обратную по отношению к ***A***, т.е. A^{-1}. При выполнении деления на ведущий элемент a_{ii} подсчитывается и определитель матрицы ***A***, который накапливается в переменной *SUM*.

Для примера и проверки процедуры находим обратную матрицу методом Гаусса и вычисляем det***A***. Матрица взята из п. 3.1. Результаты вычислений приводятся в табл. 1.20.

Определитель матрицы равен -0.1020.

Таблица 1.20

Обратная матрица				Проверка $A*A^{-1}$			
1.5505	-0.1315	-0.5025	0.0941	1.000	-0.0000	-0.0000	-0.0000
-0.1315	-0.1786	-0.0560	1.3062	0.000	1.0000	-0.0000	-0.0000
-0.5025	-0.0560	4.1116	-1.3471	-0.000	0.0000	1.0000	0.0000
0.0941	1.3062	-1.3471	0.2365	0.000	0.0000	-0.0000	1.0000

4.5. ОБРАЩЕНИЕ СИММЕТРИЧЕСКОЙ МАТРИЦЫ МЕТОДОМ КВАДРАТНЫХ КОРНЕЙ

Для обратной матрицы $\boldsymbol{X} = \boldsymbol{A}^{-1}$ справедливо равенство $\boldsymbol{AX} = \boldsymbol{BDX} = \boldsymbol{E}$, где $\boldsymbol{D}$ - диагональная матрица с элементами $d = \pm 1$; $\boldsymbol{E}$ - единичная матрица; $\boldsymbol{B}$ - матрица, удовлетворяющая равенству $\boldsymbol{A} = \boldsymbol{BD}$, т.е. в некотором смысле достаточно близкая к матрице $\boldsymbol{A}$. Следовательно, для нахождения матрицы $\boldsymbol{X}$ достаточно последовательно решить два матричных уравнения: $\boldsymbol{BY}=\boldsymbol{E}$ и $\boldsymbol{DX}=\boldsymbol{Y}$ (детали алгоритма метода квадратного корня см. в п. 3.4). Общий объем вычислений при этом составит $\sim 2n$ арифметических операций. Уточнения приближения к обратной матрице в случае необходимости можно проводить, включая в алгоритм итерации

$$\boldsymbol{X}_k = \boldsymbol{X}_{k-1}(2\boldsymbol{E} - \boldsymbol{AX}_{k-1}).$$

В работе Бахвалова (1973а) доказано, что при достаточно хорошем начальном приближении, когда выполняется неравенство $\|\boldsymbol{E} - \boldsymbol{AX}_0\| << 1$, этот итерационный процесс быстро сходится.

В работе Фадеева [Фадеев, Фадеева, 1963] предложен другой алгоритм обращения симметрической матрицы с ненулевыми ведущими минорами на основе упрощенного варианта метода квадратных корней. Этот алгоритм построен таким образом, что в матрице $\boldsymbol{A}^{-1}$ заменяются $n(n+1)/2$ диагональных и наддиагональных элементов элементами матрицы $\boldsymbol{A}$, при этом поддиагональные элементы матрицы $\boldsymbol{A}$ остаются неизменными. Преимущество такого подхода состоит в том, что требуется только n рабочих ячеек, не нужно никакой единичной матрицы, не извлекаются никакие квадратные корни, выполняется лишь n операций деления. При больших n количество операций умножения приближается к $n^3/2$, что в несколько раз меньше, чем при использовании стандартного алгоритма.

Рассмотрим процедуру INVERS, построенную на основе такого упрощенного алгоритма.

Формальные параметры процедуры. *Входные*: n (тип ***integer***) - порядок матрицы $\boldsymbol{A}$; $A[1{:}n, 1{:}n]$ (тип ***real***) - исходная матрица. *Выходные*: $A[1{:}n,1{:}n]$ (тип ***real***) - матрица, размещенная на месте исходной матрицы $\boldsymbol{A}$, в которой главная диагональ и наддиагональные элементы представляют собой элементы обратной матрицы $\boldsymbol{A}^{-1}$.

```
PROCEDURE INVERS (N: INTEGER; VAR A MAS2);
VAR  Y,P : REAL; I,J,K : INTEGER;
          V : ARRAY [1..N-1] OF REAL;
BEGIN
  FOR K:=1 TO N DO
  BEGIN
    {**** ЕСЛИ A[1,1]=0, ТО ВЫХОД ***}
     IF A[1,1]=0 THEN          EXIT;
     P:=1.0/A[1,1];
     FOR I:=2 TO N DO
         V[I-1]:=A[1,I];
     FOR I:=1 TO N-1 DO
     BEGIN
         A[I,N]:=-V[I]*P;
         Y:=-V[I]*P;
         FOR J:=I TO N-1 DO
             A[I,J]:=A[I+1,J+1]+V[J]*Y
      END ;
     A[N,N]:=-P;
  END ;
  FOR I:=1 TO N DO
     FOR J:=I TO N DO
         A[I,J]:=-A[I,J]
END .
```

Приведенная процедура была получена путем перевода процедуры INVERS66 [Агеев и др., 1976] с языка *ALGOL* на язык *PASCAL* и проверена на тех же примерах, что и в подтверждениях к рассматриваемому алгоритму, приведенных в работах [Randel, Brouden, 1962; Caffrey, 1962]. В частности, для матрицы Вильсона

$$\boldsymbol{A} = \begin{pmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{pmatrix}$$

получено

$$\boldsymbol{A}^{-1} = \begin{pmatrix} 67.9995 & -40.997 & -16.9997 & 9.9999 \\ -40.9997 & 24.9998 & 9.9999 & -5.9999 \\ -16.9997 & 9.9999 & 4.9999 & -2.9999 \\ 9..9999 & -5..9999 & -2.9999 & 1.9999 \end{pmatrix}.$$

Для матрицы пятого порядка, использованной в качестве теста в работе (Randel, Brouden, 1962), результат оказался следующим:

$$\boldsymbol{A}^{-1} = \begin{pmatrix} 0.0000 & 0.9999 & 0.0000 & 0.0000 & 0.9999 \\ 0.9999 & 1.5333 & -0.7333 & -0.1333 & 0.7999 \\ 0.0000 & -0.7333 & -0.8666 & -1.0666 & -0.5999 \\ 0.0000 & -0.1333 & -1.0666 & -1.4666 & -0.1999 \\ 0.9999 & 0.7999 & -0.5999 & -0.1999 & 0.2000 \end{pmatrix}.$$

Результаты тестирования на IBM PC/AT-386 предлагаемых процедур соответствуют результатам, приведенным в работах [Randel, Brouden,1962; Caffrey, 1962] , с достаточной степенью точности ($\varepsilon < 10^{-5}$).

Замечание. В рамках приведенного алгоритма можно легко вычислять определитель исходной матрицы как результат последовательного перемножения значений главных элементов (на что указано и в работе [Caffrey, 1962]. Так, если a_{ii}, равный $a[i, i]$, на каждом k-м шаге алгоритма является главным элементом матрицы порядка n, то определитель матрицы может быть вычислен довольно просто: как произведение ведущих элементов. Если при этом вы-

полнялись перестановки столбцов или строк, то появляется сомножитель -1 в степени, равной числу выполненных перестановок.

4.6. ОБРАЩЕНИЕ МАТРИЦЫ И РЕШЕНИЕ СИСТЕМЫ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

Рассматриваемый здесь алгоритм итерационного типа объединяет задачи обращения матрицы $\boldsymbol{A}$ и решения системы линейных алгебраических уравнений, задаваемых в матричной форме как

$$\boldsymbol{A}^T \boldsymbol{x} = \boldsymbol{b}, \qquad (1.49)$$

где $\boldsymbol{A}^T$ - транспонированная матрица $\boldsymbol{A}$; $\boldsymbol{x}$ - вектор-строка неизвестных членов; $\boldsymbol{b}$ - вектор-строка свободных членов. В случае если задано $\boldsymbol{b}$ = (1, 0, ..., 0), то процедура INVEQU будет вычислять обращенную матрицу $\boldsymbol{A}^{-1}$, для других $\boldsymbol{b}$ решается уравнение (1.49)

Формальные параметры процедуры. *Входные*: *n* (тип ***integer***) - размер матрицы $\boldsymbol{A}$; *a*[1:*n*, 1:*n*] (тип ***real***) - результат транспонирования матрицы $\boldsymbol{A}$; *b*[1:*n*] (тип ***real***) - вектор-строка $\boldsymbol{b}$; *count* (тип ***integer***) - параметр, задающий количество итераций (в работе Роека (1962), в которой приведена первоначальная, не оптимизированная версия алгоритма, не рекомендуется задавать *count* больше 6); *eps* (тип ***real***) - число, характеризующее точность полученного приближения к теоретическому результату. *Выходные*: *w*[1:*n*, 1:*n*] (тип ***real***) - матрица $\boldsymbol{A}^{-1}$ в случае $\boldsymbol{b}$ = (1,0,...,0); *x*[1:*n*] (тип ***real***) - вектор-строка решения уравнения (1.23); *c*[1:*n*,1:*n*] (тип ***real***) - матрица, служащая для контроля точности. После выполнения процедуры в качестве первой строки матрицы $\boldsymbol{c}$ будет вектор $\boldsymbol{b}$ вдоль главной ее диагонали (1,0,...,0). В нулевом цикле первой итерации процедура INVEQU определяет в качестве строк матрицы $\boldsymbol{w}$ столбцы матрицы $\boldsymbol{a}$, для всех последующих итераций строки матрицы $\boldsymbol{w}$, вычисленные в *n*-м цикле последней итерации, являются строками матрицы $\boldsymbol{w}$ нулевого цикла.

```
PROCEDURE INVEQU (A:MAS2; B:MAS1;
                  N,COUNT: INTEGER;EPS:REAL;
                  VAR W: MAS2; X: MAS1; C:MAS2);
 VAR BH,Z : REAL; I,J,P : INTEGER;
LABEL BEG;
BEGIN
  FOR J:=1 TO N DO
      FOR I:=1 TO N DO
          W[J,I]:=A[I,J];
BEG:
  FOR J:=1 TO N DO
      FOR I:=1 TO N DO
          C[I,J]:=0;
  BH:=B[1];
  FOR J:=1 TO N DO
  BEGIN
      FOR I:=1 TO N DO
          C[J,J]:=C[J,J]+W[J,I]*A[I,J];
      Z:=BH/C[J,J];
      FOR I:=1 TO N DO
          W[J,I]:=Z*W[J,I];
      FOR I:=1 TO N DO
          IF I<>J THEN
          BEGIN
              FOR P:=1 TO N DO
                  C[I,J]:=C[I,J]+A[P,J]*W[I,P];
          IF I=1 THEN    Z:=B[J] -C[I,J];
              ELSE       Z:=0-C[I,J];
              FOR P:=1 TO N DO
                  W[I,P]:=BH*W[I,P]+Z*W[J,P]
          END;
      BH:=1
  END;
  Z:=ABS(B[1]);
  FOR J:=1 TO N DO
      IF ABS(ABS(C[J,J])-Z) > EPS THEN
      BEGIN
          COUNT:=COUNT-1;
          IF COUNT > 0 THEN        GO TO BEG
      END;
  FOR J:=1 TO N DO
      X[J]:=W[1,J]
END.
```

Процедура INVEQU получена из процедуры *simult* [Агеев, 1976] [1] путем ее перевода с языка *ALGOL* на язык *PASCAL*.

Проверка процедуры INVEQU производилась в целях контроля для тех же входных параметров, что и в работе Агеева (1976). При этом для входных параметров

$$a = \begin{pmatrix} 2 & 1 & 3 \\ 1 & -2 & 2 \\ 3 & 1 & 2 \end{pmatrix}, \quad b = (1, \ 0, \ 0), (9, \ -2, \ 7)$$

n = 3, *count* = 3, 6, 10 и *eps* = 1*e* - 3, 1*e* - 5, 1*e* - 8, при всех значениях *count* и *eps* были получены следующие результаты, совпадающие с полученными Агеевым (1976):

1) для $\boldsymbol{b}$ = (1, 0, 0)

$$\boldsymbol{w} = \begin{pmatrix} -0.46153846 & 0.07692308 & 0.61538461 \\ 0.30769230 & -0.38461538 & -0.07692308 \\ 0.53846153 & 0.07692308 & -0.38461538 \end{pmatrix};$$

2) для $\boldsymbol{b}$ = (9, -2, 7), $\boldsymbol{x}$ = (-1, 2, 3)

$$\boldsymbol{c} = \begin{pmatrix} 9 & -2 & 7 \\ -0.36379788e-11 & 9 & 0.21827872e-10 \\ 0 & -0.58207661e-10 & 9 \end{pmatrix}.$$

Для матрицы, транспонированной от матрицы $\boldsymbol{A}$, использованной в первом тесте, результаты также хорошо

[1] Процедура SIMULT является результатом существенной переработки алгоритма, впервые опубликованного в работе [Roek,1962], с целью сокращения его записи и оптимизации по расходуемой памяти и времени выполнения.

совпали с результатами, приведенными М.И. Агеевым (1976):

1) для $\boldsymbol{b} = (1, 0, 0)$:

$$\boldsymbol{w} = \begin{pmatrix} -0.46153846 & 0.30769230 & 0.53846153 \\ 0.07692308 & -0.38461538 & 0.07692308 \\ 0.61538461 & -0.07692308 & -0.38461538 \end{pmatrix};$$

2) для $\boldsymbol{b} = (9, -2, 7)$: $\boldsymbol{x} = (-0.42536367e\text{-}10, 3, 2)$,

$$\boldsymbol{c} = \begin{pmatrix} 9 & -2 & 7 \\ -0.14551915e-10 & 9 & 0.72759576e-11 \\ 0.29103830e-10 & 0 & 9 \end{pmatrix},$$

точное решение уравнения (1.23), как указано Агеевым (1976), при этих входных данных $\boldsymbol{x} = (0, 3, 2)$.

4.7. УМНОЖЕНИЕ УПЛОТНЕННОЙ СИММЕТРИЧЕСКОЙ МАТРИЦЫ НА ПРЯМОУГОЛЬНУЮ

Произведение матрицы

$$\boldsymbol{A}=||a_{ij}|| \ (i = 1, ..., m; j = 1, ..., n)$$

на матрицу

$$\boldsymbol{B}=||b_{ij}|| \ (i = 1, ..., n; j = 1, ..., p)$$

есть матрица

$$\boldsymbol{C}=||c_{ij}|| \ (i = 1, ..., m; j = 1, ..., p),$$

элементы которой определяются формулой

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \ , \ i=1, ..., \ m; \ j = 1,..., p. \quad (1.50)$$

Из равенства (1.50) следует, что операция перемножения матриц определена только тогда, когда число столбцов матрицы $\boldsymbol{A}$ равно количеству строк матрицы $\boldsymbol{B}$.

В случае если одна из перемножаемых матриц, например матрица $\boldsymbol{A}$, является симметрической, т.е. $a_{ij} = a_{ji}$, то она может быть уплотнена путем сведения к почти треугольной матрице (верхняя треугольная часть матрицы $\boldsymbol{A}$, включая главную диагональ), которую из практических вычислительных соображений (экономия машинной памяти, ускорение поиска нужного элемента) удобно построчно переписать в виде одномерного массива $d\,[1{:}n\cdot(n{+}1)/2]$. Заметим, что в этом случае в процессе выполнения соответствующей процедуры значения элементов исходной матрицы $a[i, j]$ могут присваиваться элементам $d[m_i{+}j]$, где $m_i = (2n - i)\,(i - 1) / 2$. Отсюда следует, что $m_{i+1}= m_i + n - i$, т.е. значения m_i могут вычисляться рекурсивно, если положить $m_0 = 0$. Такое уплотнение симметрической матрицы может быть полезно, например, в том случае, когда она целиком не помещается в свободную часть оперативной памяти ЭВМ. Размещение результирующей матрицы $\boldsymbol{C}$ на месте вводимой матрицы $\boldsymbol{B}$ также позволяет оптимизировать затраты оперативной памяти.

Изложенный алгоритм реализован в процедуре MULTCRAM.

Формальные параметры процедуры. *Входные*: n (тип ***integer***) - порядок квадратной матрицы $\boldsymbol{A}$ и количество строк прямоугольной матрицы $\boldsymbol{B}$; $d[1{:}n\cdot(n{+}1)/2]$ (тип ***real***) - вводимая в виде одномерного массива d уплотненная матрица $\boldsymbol{A}$; r (тип ***integer***) - количество столбцов вводимой прямоугольной матрицы $\boldsymbol{B}$; $b[1{:}n, \ 1{:}r]$ (тип ***real***) - матрица $\boldsymbol{B}$. *Выходные*: $b[1{:}n, \ 1{:}r]$ (тип ***real***) - выводимая на месте входной матрицы $\boldsymbol{B}$ матрица-произведение $\boldsymbol{C} = \boldsymbol{A}*\boldsymbol{B}$.

```
PROCEDURE MULTCRAM (D: MAS1; N,R: INTEGER;
          VAR B:MAS2);
VAR S:REAL; I,J,K,M : INTEGER; V ARRAY [1..N] OF REAL;
BEGIN
  FOR J:=1 TO R DO
  BEGIN
     FOR I:=1 TO N DO
     BEGIN
        S:=0;          M:=I-N;
        FOR K:=1 TO N DO
        BEGIN
           IF K <= I THEN M := M+N+1-K
              ELSE M:=M+1;
           S:=S+D[M]*B[K,J]
        END {K};          V[I]:=S
     END {I};
     FOR I:=1 TO N DO
        B[I,J]:=V[I]
  END {J};
END {MULTCRAM}.
```

Процедура MULTCRAM была получена из процедуры CRAM [Агеев, 1976][1] путем ее перевода с языка *ALGOL* на язык *PASCAL*. Тестирование процедуры проводилось на IBM PC/AT-386 для тех же исходных данных, что и в работе Каффрей (1961):

$$\boldsymbol{A} = \begin{pmatrix} 0 & 4 & 10 \\ 4 & 8 & 18 \\ 10 & 18 & 40 \end{pmatrix}; \qquad \boldsymbol{B} = \begin{pmatrix} 5 & 3 \\ 2 & 6 \\ 1 & 8 \end{pmatrix}.$$

При этом был получен верный результат:

$$\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B} = \begin{pmatrix} 18 & 104 \\ 54 & 204 \\ 126 & 458 \end{pmatrix}.$$

4.8. КОРРЕКТИРОВКА ОБРАТНОЙ МАТРИЦЫ ПОСЛЕ ИЗМЕНЕНИЯ ОДНОГО ЭЛЕМЕНТА В ПРЯМОЙ МАТРИЦЕ

Пусть дана матрица $\boldsymbol{A} = \boldsymbol{M}^{-1}$ размерностью $[n\times n]$ и матрица $\boldsymbol{M'}$, полученная в результате увеличения элемента $m[i,j]$ матрицы $\boldsymbol{M}$ на величину d. Тогда скорректированная матрица $\boldsymbol{B} = (\boldsymbol{M'})^{-1}$ может быть эффективно вычислена по

[1] Процедура CRAM является переработкой алгоритма [Caffrey, 1961] с целью его упрощения и модификации.

элементам матрицы ***A*** без обращения матрицы ***M*** при помощи представленной процедуры ADJUST.

Формальные параметры процедуры. *Входные*: *a*[*n,n*] - (тип ***real***) - исходная матрица ***A***; *n* (тип ***integer***) - порядок матрицы ***A***; *i, j* (тип *integer*) - индексы измененного элемента *m*[*i, j*] матрицы ***M***; *d* (тип ***real***) - величина изменения элемента *m*[*i, j*]. *Выходные*: *b*[*n, n*] (тип ***real***) - скорректированная матрица ***B***.

```
PROCEDURE ADJUST (A:MAS1;N,I,J: INTEGER;
                  D:REAL; VAR B:MAS1);
VAR  T : REAL;  R,S : INTEGER;
BEGIN    T:=D/(A[J,I]*D+1);
   FOR R:=1 TO N DO
       FOR S:=1 TO N DO
           B[R,S]:=A[R,S]-T*A[R,I]*A[J,S]
END {ADJUST};
```

Процедура ADJUST представляет собой перевод на язык *PASCAL* опубликованной в работе [Агеев и др., 1976], исправленной, сокращенной и ординарно переработанной процедуры, опубликованной в работе [Herndon, 1961]. Тестирование процедуры ADJUST было выполнено на IBM PC/AT-286 для следующих исходных данных:

$$\boldsymbol{A} = \begin{pmatrix} -2.0 & 1.0 \\ 1.5 & -0.5 \end{pmatrix}, \quad n = 2,\ i = 1,\ j = 2,\ d = 3.$$

Результат корректировки

$$\boldsymbol{B} = \begin{pmatrix} -0.36363636 & 0.45454545 \\ 0.27272727 & -0.90909091 \end{pmatrix}$$

с точностью до ошибок округления совпал с точным решением

$$\boldsymbol{B} = \begin{pmatrix} -4/11 & 5/11 \\ 3/11 & -1/11 \end{pmatrix}.$$

Замечание. Интересно, что исходной алгоритм [Herndon, 1961] проверялся дополнительно Р. Георгом (1962) для матриц порядка n = 2, 3, ..., 10 при произвольных значениях *i, j* и случайных приращениях элемента *m*[*i, j*] прямой матрицы ***M***: -1.0 < *d* << 1.0. По результатам корректировки рассчитывалось произведение ***C*** = ***B*******M*** ' и вычислялась погрешность каждого опыта по формуле

$$\varepsilon = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \right) - n,$$

при этом во всех произведенных опытах значение ε не превышало 2*e*-8.

4.9. МАТРИЦА ПРИЧИННО-СЛЕДСТВЕННЫХ ОТНОШЕНИЙ

Задача построения матрицы причинно-следственных связей возникает при исследовании сетевых диаграмм и в этом смысле стоит несколько особняком в ряду рассматриваемого в настоящем параграфе алгебры матриц. Тем не менее вследствие того, что данный логический алгоритм, суть которого описана в этом пункте, оперирует с матричными объектами, мы посчитали целесообразным рассмотреть его именно здесь.

Пусть задана матрица ***M*** размером *n*×*n*, в которой каждый элемент m_{ij} = ***true***, если индивидуум *i* является непосредственной причиной ("родителем") индивидуума *j*, в противном случае m_{ij} = ***false***. Тогда, если существует последовательность чисел *k*, l, ..., *p* такая, что в матрице ***M*** все m_{ik}, m_{kl}, ..., m_{pj} эквивалентны ***true***, т.е. если индивидуум *i* является косвенной причиной ("прародителем") индивидуума *j*, то элементы преобразованной матрицы ***M*** - m_{ij} = ***true***.

Данный алгоритм реализован в процедуре ANCES.

Формальные параметры процедуры. *Входные*: *m*[1:*n*, 1:*n*] (тип ***boolean***) - исходная матрица ***M***; *n* (тип ***integer***) - порядок матрицы ***M***. *Выходные*: *m* [1:*n*, 1:*n*] (тип ***boolean***) - преобразованная матрица ***M***.

```
PROCEDURE ANCES (N:INTEGER;
          VAR M : ARRAY [1..N,1..N] OF BOOLEAN);
VAR I, J, K : INTEGER;
BEGIN
  FOR I:=1 TO N DO
      FOR J:=1 TO N DO
      IF M[J,I] THEN
          FOR K:=1 TO N DO
              IF M[I,K] THEN M[J,K]:=TRUE
END {ANCES};
```

Процедура ANCES является переводом с языка *ALGOL* на язык *PASCAL* стереотипного переиздания [Агеев и др., 1976] ординарно переработанного алгоритма Р.Б. Флойда (1962). Тестирование процедуры проводилось на IBM PC/AT-286 для тех же исходных матриц, что и в работе Х.Ц.Тачера (1963), в которой он подтвердил эффективность алгоритма Р.Б. Флойда (1962). Нами при тестировании были получены результаты, совпадающие с результатами работы [Thacher, 1963]:

1) при n = 5

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix};$$

2) при n = 9

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \rightarrow$$

$$\rightarrow \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Правильность этих результатов, как указано в работе [Thacher, 1963], была убедительно подтверждена при исследовании сетевых диаграмм.

2.

Предположим, что задано некоторое упорядоченное множество вещественных абсцисс $x_1, x_2, ..., x_n$ и связанное с ним множество вещественных ординат $y_1, y_2, ..., y_n$. Пусть $x_1 < x_2 < ... < x_n$ и каждое y_i есть некоторое вещественное число, отвечающее x_i, которое определяется математически или в результате каких-либо наблюдений (рис. 2.1). Точки (x_i, y_i) называются *узлами интерполяции*. Кривая, которая точно проходит через эти узлы, называется *интерполяционной кривой*.

Рис. 2.1. *Узлы интерполяции и интерполяционная кривая f (x, y)*

При решении задач на ЭВМ достаточно часто возникает обратная задача, т.е. необходимо подобрать некоторую простую функцию, которая, с одной стороны, имела бы в известных точках заданные значения, а с другой — вычислялась бы быстрее и проще исходной.

Исходя из изложенного можно сформулировать (не строго) следующие определения. Задача *одномерной интерполяции* заключается в построении такой непрерывной функции *f*, при которой $f(x_i) \equiv y_i$ для всех x_i, т.е. функция *f* обязательно должна проходить абсолютно точно через заданные узлы интерполяции. Однако при этом $f(x_k)$ должна принимать "разумные" значения для всех x_k, лежащих между заданными точками x_i. Кроме того, на интерполирующую функцию накладывается еще ряд очевидных ограничений: не иметь особых точек на интервале интерполирования; быть достаточно гладкой; иметь необходимое количество производных и пр. Если функция определена и непрерывна на $[y_1, y_2]$, ее интерполяционные узлы расположены на $[x_1, x_2]$, а точка $\xi \notin [x_1, x_2]$ или $f(\xi) \notin [y_1, y_2]$, то тогда говорят о задаче *экстраполирования функции.*

Таким образом, задачи интерполирования и экстраполирования возникают в следующих случаях:

1) если решаются прогнозные задачи (экстраполирование);

2) практически всегда при решении экономических задач (интерполирование и экстраполирование);

3) при обработке данных, полученных в экспериментах (интерполирование);

4) всегда при решении задач на ЭВМ (интерполирование и экстраполирование);

5) при задании функции графиком или таблицей (интерполирование и экстраполирование);

6) во всех остальных не описанных здесь случаях, если это необходимо.

§ 1. ПОСТРОЕНИЕ ИНТЕРПОЛЯЦИОННОГО ПОЛИНОМА ЛАГРАНЖА ПО ЗАДАННЫМ ЗНАЧЕНИЯМ ФУНКЦИИ

Пусть теперь функция $f(x)$ известна лишь в $x_1, x_2, ..., x_n$ узлах некоторой сетки. Попытаемся построить такую другую функцию $\varphi(x; \boldsymbol{a})$, которая полностью совпадала бы с заданной $f(x)$ в этих узлах, т.е. $\varphi(x; \boldsymbol{a}) = \varphi(x, a_1, a_2, ..., a_m) \equiv \equiv f(x_к)$. Тогда из последнего равенства можно будет определить компоненты $\boldsymbol{a}$. При $k \le m$, а в частном случае $k = m$, компоненты $\boldsymbol{a}$ определяются из решения системы

$$\sum_{k=1}^{m} a_k \cdot \varphi_k(x_i) = f(x_i) \ , \qquad (*)$$

которая в указанных случаях имеет единственное решение, если ее главный определитель отличен от нуля:

$$\Delta \equiv \det\{\varphi_k(x_i)\} = \begin{vmatrix} \varphi_1(x_1) & \varphi_2(x_1) & \dots & \varphi_n(x_1) \\ \varphi_1(x_2) & \varphi_2(x_2) & \dots & \varphi_n(x_2) \\ \dots & \dots & \dots & \dots \\ \varphi_1(x_m) & \varphi_{21}(x_m) & \dots & \varphi_n(x_m) \end{vmatrix} \neq 0 \forall x_i \neq x_j \ .$$

Система функций, удовлетворяющая условию (*), является *Чебышевской* [Бахвалов, 1973а].

Одним из самых важных классов среди интерполирующих функций является множество алгебраических полиномов вида

$$\varphi(x) = \sum_{i=0}^{n} a_i \cdot x^i \ ,$$

где a_i - некоторые неизвестные коэффициенты. Для того чтобы их определить однозначно, необходимо иметь n+1 точек для построения системы из n линейных уравнений относительно a_i:

$$y_j = \sum_{i=0}^{n} a_i \cdot x_j^i \ , (j = 0, 1, ..., N+1).$$

При этом потребуем, чтобы построенный полином в заданных узлах $x_i \in \{X\}$ в точности совпадал со значениями $f(x_i)$ в указанных точках. Если среди множества x_i нет совпадающих, тогда система линейных уравнений, полученная относительно a_i,

$$\left.\begin{array}{llll} a_0 + & a_1 x_0 + & \ldots + & a_m x_0^m = \varphi_0; \\ a_0 + & a_1 x_1 + & \ldots + & a_m x_1^m = \varphi_1; \\ \ldots & \ldots & \ldots & \ldots \\ a_0 + & a_1 x_1 + & \ldots + & a_m x_n^m = \varphi_n, \end{array}\right\}$$

определена, имеет решение и это решение единственное. Заметим, что главный определитель системы - это определитель Вандермонда:

$$\Delta = \prod_{0 \le p < q \le n} \left(x_q - x_p \right) \neq 0 \ ,$$

который при выполнении названных условий всегда отличен от 0. Значит система имеет решение и это решение будет единственное. Очевидно, что решением системы будет полином n-й степени, который называют *полиномом Лагранжа* и записывают

$$L_n(x) = \sum_{i=0}^{n} y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}. \qquad (2.1)$$

Важно отметить, что существует один и только один полином степени меньше n, который интерполирует заданные $n + 1$ точки. При использовании всего известного набора точек интерполирующий полином называют *глобальным интерполирующим полиномом.* Однако по теореме Фабера при *специальном расположении узлов интерполяции n всегда можно найти такой полином P_n, непрерывный на заданном отрезке $[x_0, x_n]$, который не сходится к f, несмотря на то, что $P_n(x_i) \equiv y_i$* . Поэтому на практике редко пользуются глобальным интерполирующим полиномом, заменяя его полиномом степени не выше 5. Тогда говорят о *скользящей интерполяции*.

Процедура-функция L выполняет интерполирование по заданным значениям x и y. Следует отметить, что ее эффективно использовать, если количество точек не превышает 10, в противном случае необходимо применять *интерполяцию сплайнами* (для уменьшения погрешности интерполирования).

Формальные параметры процедуры. *Входные*: b (тип ***real***) - точка на оси Ox, для которой ищутся значение функции; n (тип ***integer***) - количество точек, по которым строится интерполирующий полином. Степень полинома должна быть на единицу меньше n; x, y (тип ***real***) - массивы, в которых находятся заданные значения точек. *Выходные*: L (тип ***real***) - значение функции в точке b.

```
FUNCTION L ( B : REAL; N : INTEGER;
             X, Y : MAS ) : REAL;
VAR SUM, PR : REAL; I, J : INTEGER;
 (* □□□□□□□ □□□□□□□□□□ □□□□□□□□ "□" □ □□□□□□□□
 □□□□□ "□", □□□□ □□□□□□□□ □□□□□□□ "□" □ "Y" *)
BEGIN
  SUM := 0.0;
  FOR J:=1 TO N DO
  BEGIN
     PR := 1.0;
     FOR I := 1 TO N DO
     IF I<>J THEN
        PR:=PR*(B-X[I])/ (X[J]-X[I]);
     SUM := SUM + Y[J] * PR;
  END;
  L := SUM;
END.
```

При большом количестве точек n могут оказаться решающими ошибки, связанные с округлением при вычислении высших степеней x, что приведет к расхождению решения. Кроме того, ошибки округления накапливаются на каждом очередном шаге, поэтому оценка ошибки интерполирования должна обязательно вычисляться в каждом конкретном случае на каждом расчетном шаге.

Один из возможных способов оценки ошибки при интерполировании заключается в следующем. Во-первых, предполагают, что заданные числа y_i являются в действительности значениями некоторой функции $f(x_i)$. Во-вторых, что $f(x_i)$ имеет n +1 непрерывную производную для всех x_i. В-третьих, что P_n - единственный полином степени < n, интерполирующий заданные точки $\{(x_i , y_i)\}$. Тогда можно доказать, что для любого действительного x_i выполняется

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{n+1} \prod_{i=0}^{n} (x - x_i),$$

где x - некоторая неизвестная точка на интервале, определяемом точками x_0, x_1, ..., x_n и x_{n+1}. Когда известны границы для $f(x)$, эта разность дает оценку ошибки.

Приводимые результаты были получены на персональном компьютере IBM386 DX2 фирмы “R-Style”. Для подтверждения корректности процедуры L были использованы два набора значений X и Y для неравноотстоящих (вариант А) и равноотстоящих (вариант Б) узлов. Результаты вычислений и исходные данные приведены в табл. 2.1.

Таблица 2.1

Параметр	№ i	Вариант А		Вариант Б	
		X	Y	X	Y
Исходные данные	1	0.11	9.0	0.1	2.0
	2	0.15	6.6	0.2	4.0
	3	0.21	4.7	0.3	5.0
	4	0.29	3.4	0.4	5.2
	5	0.35	2.7	0.5	3.8
	6	0.40	2.4	0.6	1.5
Заданная точка		0.263	?	0.253	?

Окончание таблицы 2.1

Параметр	№ i	Вариант А X	Вариант А Y	Вариант Б X	Вариант Б Y	Параметр	№ i	Вариант А X	Вариант А Y	Вариант Б X	Вариант Б Y
Расчетные точки, через которые проходит интерполяционная кривая	1	0.11000	9.00000	0.10000	2.00000	Расчетные точки, через которые проходит интерполяционная кривая	14	0.26708	3.71492	0.37083	5.27809
	2	0.12208	8.13717	0.12083	2.60852		15	0.27917	3.54559	0.39167	5.23644
	3	0.13417	7.40129	0.14167	3.08333		16	0.29125	3.38354	0.41250	5.12278
	4	0.14625	6.77476	0.16250	3.46364		17	0.30333	3.22789	0.43333	4.93141
	5	0.15833	6.24154	0.18333	3.77930		18	0.31542	3.07873	0.45417	4.66007
	6	0.17042	5.78712	0.20417	4.05183		19	0.32750	2.93704	0.47500	4.31085
	7	0.18250	5.39849	0.22500	4.29535		20	0.33958	2.804663	0.49583	3.89127
	8	0.19458	5.06405	0.24583	4.51758		21	0.35167	2.68421	0.51667	3.41516
	9	0.20667	4.77360	0.26667	4.72085		22	0.36375	2.57910	0.53750	2.90372
	10	0.21875	4.51827	0.28750	4.90302		23	0.37583	2.49342	0.55833	2.38646
	11	0.23083	4.29049	0.30833	5.05853		24	0.38792	2.43193	0.57917	1.90219
	12	0.24292	4.08391	0.32917	5.17932		25	0.40000	2.40000	0.60000	1.50000
	13	0.25500	3.89339	0.35000	5.25586	Заданная точка		0.26300	3.77409	0.25300	4.58967

§ 2. ПОСТРОЕНИЕ ИНТЕРПОЛЯЦИОННОГО ПОЛИНОМА НЬЮТОНА ПО ЗАДАННЫМ ЗНАЧЕНИЯМ ФУНКЦИИ

В силу единственности многочлена степени n, построенного по $n + 1$ значениям функции $f(x)$, *многочлен Ньютона* $P_n(x)$ является разновидностью записи интерполяционного многочлена и полностью совпадает с многочленом, построенным по формуле Лагранжа (2.1). Однако с помощью многочлена Ньютона удобнее проводить интерполяцию в начале и конце таблицы значений функции.

В начале таблицы используют *первую интерполяционную формулу Ньютона*, а в конце - *вторую*. Рассмотрим один из способов построения первой интерполяционной формулы.

Пусть некоторая функция $f(x)$ задана табличными значениями $y_0 = f(x_0)$; $y_1 = f(x_1)$; ...; $y_n = f(x_n)$ в равноотстоящих узлах интерполяции $\{x_0 , x_1 = x_0 + h;...; x_n = x_0 + nh\}$. Требуется построить интерполяционный полином Ньютона $P_n(x)$ степени n, при котором

$$P_n(x_0) \equiv y_0;\ P_n(x_1) \equiv y_1;\ ...;\ P_n(x_n) \equiv y_n. \quad (2.2)$$

Будем искать полином в виде

$$P_n(x_0) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + ... \\ ... + a_n(x - x_0)(x - x_1)...(x - x_n), \quad (2.3)$$

где неизвестные коэффициенты a_i находятся из очень простых зависимостей. Для того чтобы найти a_0, положим $x = x_0$. Очевидно, что при этом $P_n(x_0) \equiv y_0\, a_0$ [это следует из формулы (2.2)]. Но так как все члены уравнения (2.3) , кроме первого, содержат сомножитель $(x - x_0)$, следовательно, они все станут равными нулю, и из формулы (2.3) с учетом формулы (2.2) имеем $P_n(x_0) = a_0 \equiv y_0$.

Для того чтобы найти a_1, положим $x = x_1$. Повторив все рассуждения и учитывая, что значение полинома в указанной точке будет тождественно равно y_1 [формула (2.2)], после подстановки в формулу (2.3) x_1 имеем

$$P_n(x_1) = a_0 + a_1(x_1 - x_0) = y_0 + a_1 h = y_1.$$

Все остальные сомножители при неизвестных коэффициентах a_i будут равны нулю. Преобразуя последнее выражение, находим a_1 как $a_1 = \Delta_1/h$ [здесь $\Delta_1 = P_n(x_0 + h) - P_n(x_0) = y_1 - y_0$].

Для того чтобы определить a_2, положим $x = x_2$ и, рассуждая аналогично, определим третий коэффициент как $a_2 = \Delta_2 / (2!\, h^2)$.

Подставляя в выражение (2.3) последовательно все x_n, приходим к общей формуле для получения коэффициентов a_i:

$$a_i = \Delta_i / (\, i!\, h^i),$$

которые подставим в формулу (2.3) и получим первую интерполяционную формулу Ньютона:

$$P_n(x) = y_0 + \frac{\Delta_1}{1!\cdot h}(x - x_0) + \frac{\Delta_2}{2!\cdot h^2}(x - x_0)(x - x_1) + \cdots \\ \ldots + \frac{\Delta_n}{n!\cdot h^n}\prod_{i=0}^{n-1}(x - x_i). \quad (2.4)$$

В формуле (2.4) обычно выполняют замену переменных: $q = (x - x_0)/h$, где h - шаг интерполирования. Тогда

первая интерполяционная формула Ньютона записывается несколько иначе:

$$P_n(x) = y_0 + \Delta_1 \cdot q + \frac{\Delta_2 \cdot q \cdot (q-1)}{2!} + \ldots + \frac{\Delta_n \cdot q \cdot (q-1) \times \ldots \times (q-n+1)}{n!} . \quad (2.5)$$

При $n = 1$ получаем формулу линейного интерполирования; при $n = 2$ - параболического интерполирования и т.д.

Вторую интерполяционную формулу Ньютона получают, если узлы интерполяции в $P_n(x)$ берут в несколько ином порядке:

$$P_n(x_0) = a_0 + a_1(x - x_n) + a_2(x - x_n)(x - x_{n-1}) + \ldots + a_n(x - x_n)(x - x_{n-1}) \ldots (x - x_0) .$$

Тогда, рассуждая как и в случае первой интерполяционной формулы, получаем искомую форму записи полинома Ньютона, которая известна как вторая интерполяционная формула:

$$P_n(x) = y_n + \frac{\Delta_1}{1! \cdot h}(x - x_n) + \frac{\Delta_2}{2! \cdot h^2}(x - x_n)(x - x_{n-1}) + \ldots + \frac{\Delta_n}{n! \cdot h^n}\prod_{i=0}^{n-1}(x - x_i) . \quad (2.6)$$

Выполнив подстановку $q = (x - x_n)/h$, получим иную запись интерполяционого многочлена Ньютона:

$$P_n(x) = y_n + \Delta_1 \cdot q + \frac{\Delta_2 \cdot q \cdot (q+1)}{2!} + \ldots + \frac{\Delta_n \cdot q \cdot (q+1) \times \ldots \times (q+n-1)}{n!} . \quad (2.7)$$

Первая интерполяционная формула Ньютона используется для интерполирования в начале отрезка $[x_i, x_{i+1}]$ и экстраполирования до первой точки x_0, т.е. для интерполирования вперед и экстраполирования назад. При таком интерполировании $q = (x - x_i)/h > 0$. При экстраполировании назад по первой интерполяционной формуле $q = (x - x_i)/h < 0$. При интерполировании в конце таблицы, т.е. при интерполировании назад, когда шаг интерполяции постоянен, применяют вторую интерполяционную формулу Ньютона, где $q = (x - x_{i+1}) / h < 0$. Эту же формулу используют при экстраполировании вперед (в конце таблицы), но тогда $q = (x - x_{i+1})/ h > 0$.

Как уже отмечалось, полином Ньютона является иной записью полинома Лагранжа для функции, заданной в равноотстоящих узлах интерполяции. Поэтому при вычислении полинома Ньютона возможно использовать процедуру-функцию L из § 1. Здесь же приводятся оригинальные процедуры-функции для вычисления полинома Ньютона по первой и второй интерполяционным формулам, в которых применяется перед интерполяцией метод двоичного поиска интервала [Бахвалов, 1973а] .

Процедура-функция NEW1 применяется при интерполировании первой интерполяционной формулой (метод интерполирования вперед), а процедура-функция NEW2 - при интерполировании второй интерполяционной формулой (метод интерполирования назад). Обе процедуры применяются, если узлы интерполяции расположены равномерно. Процедуры-функции могут с успехом применяться для поиска значений во многих точках заданного интервала, например при решении задачи субтабулирования функций. Однако тогда следует рекомендовать вынести блок расчета разностей первого, второго и третьего порядков из процедур. Это значительно ускорит их работу. Но тогда следует позаботиться о передаче массивов разностей в процедуры. Для этого можно их объявить глобальными или включить их имена в список формальных параметров при обращении к процедурам.

Формальные параметры процедур. Процедура-функция NEW1. *Входные*: *x*, *y* (тип ***real***) - массивы заданных значений x_i и y_i; *n* (тип ***integer***) - количество точек в массивах *x* и *y*; *x*1 (тип ***real***)- искомая точка. *Выходные*: процедура-функция NEW1 возвращает вещественное значение функции в точке *x*1 согласно построенному полиному.

Процедура-функция NEW2. *Входные*: *x*, *y* (тип ***real***) - массивы заданных значений x_i и y_i; *n* (тип ***integer***) - количество точек в массивах *x* и *y*; *x*1 (тип ***real***)- искомая точка. *Выходные*: процедура-функция NEW2 возвращает вещественное значение функции в точке *x*1 согласно построенному полиному.

```
                {□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□□ □□□□□□□*}
FUNCTION NEW1 (X,Y: MAS; N: INTEGER; X1:REAL):REAL;
LABEL 30;
VAR I,J,K, N : INTEGER; Q : REAL; R1,R2,R3: MAS;
BEGIN    I := 1;  J := N;
                  { **** □□□□□□ □□□□□□□□ □□□□□□□□□ ****
              **** □□□□□□□, □□□□□□□ □ □□□□□□□□ □□□□□□□□ ****}
  FOR I := 2 TO N DO
  BEGIN
     R1[I-1] := Y[I] - Y[I-1];
     IF I>2 THEN       R2 [I-2] := R1[I-1] - R1[I-2];
     IF I>3 THEN       R3 [I-3] := R2[I-2] - R2[I-3];
  END;
            {**** □□□□□ □□□□□□□□□ □□□□□□ □□□□□□□□□ ****}
  IF X1 < X[I] THEN GOTO 30;
  REPEAT
     K := (I+J) DIV 2;
     IF X1< X[K] THEN
        J := K;
     IF X1>=X[K] THEN
        I := K;
  UNTIL J <= I+1;
 30:      Q := (X1 - X[I]) / (X[2]-X[1]);
  NEW1: = Y[I] + Q * (R1[I] + (Q-1)*(R2[I]/2.0 +
             (Q-2) * R3[I] / 6.0));
END.
```

```
        {***□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□□ □□□□□□□ ***}
FUNCTION NEW2 (X,Y: MAS; N: INTEGER; X1:REAL):REAL;
LABEL 30;
VAR I,J,K,N : INTEGER; Q : REAL; R1,R2,R3: MAS;
BEGIN    I := 1;  J := N;
                {**** □□□□□□ □□□□□□□□ □□□□□□□□□ ****
          **** □□□□□□□, □□□□□□□ □ □□□□□□□□ □□□□□□□□ ****}
  FOR I := 2 TO N DO
  BEGIN      R1[I-1] := Y[I] - Y[I-1];
     IF I>2 THEN    R2 [I-2] := R1[I-1] - R1[I-2];
     IF I>3 THEN    R3 [I-3] := R2[I-2] - R2[I-3];
  END;
           {**** □□□□□ □□□□□□□□□ □□□□□□ □□□□□□□□□ ****}
  IF X1 > X[J] THEN      GOTO 30;
  REPEAT         K := (I+J) DIV 2;
     IF X1< X[K] THEN           J := K;
     IF X1>=X[K] THEN              I := K;
  UNTIL J <= I+1;
30:      Q := (X1 - X[I]) / (X[2]-X[1]);
  NEW2 := Y[J] + Q * (R1[J-1] +
          (Q+1) * (R2[J-2]/2.0 + (Q+2)*R3[J-3]/6.0));
END.
```

Оценка погрешности первой и второй интерполяционных формул Ньютона получается из оценки погрешности интерполяционного полинома Лагранжа. Введем обозначение $h = x_{i+1} - x_i$ и полагая для первой интерполяционной формулы $q = (x - x_0)/h$, а для второй $q = (x - x_n)/h$, получим для первой интерполяционной формулы

$$R_n(x) = h^{n+1}\,\frac{q\cdot(q-1)\times\ldots\times(q-n)}{(n+1)!}\cdot f^{(n+1)}(x)$$

и для второй интерполяционной формулы

$$R_n(x) = h^{n+1}\,\frac{q\cdot(q+1)\times\ldots\times(q+n)}{(n+1)!}\cdot f^{(n+1)}(x)\cdot$$

Здесь x - некоторая точка из отрезка интерполирования $[x_0,\ x_n]$ в случае задачи интерполирования или не принадлежащая указанному отрезку в случае задачи экстраполирования. Выполняя предельный переход для

$$f^{(n+1)}(x) = \Delta_{n+1} / h^{n+1}$$

и подставляя последнее выражение в формулу для погрешности, окончательно будем иметь для первой интерполяционной формулы

$$R_n(x) = \frac{q\cdot(q-1)\times\ldots\times(q-n)}{(n+1)!}\cdot\Delta_{n+1}$$

и для второй интерполяционной формулы

$$R_n(x) = \frac{q\cdot(q+1)\times\ldots\times(q+n)}{(n+1)!}\cdot\Delta_{n+1}\cdot$$

По последним формулам производится оценка погрешности интерполирования.

Для проверки и тестирования процедур-функций, используя первую или вторую интерполяционные формулы Ньютона, вычисляли значения функции при заданных значениях аргумента. Все расчеты контролировались при составлении таблицы разностей.

Вычисления производились с точностью 10^{-5}. Исходные данные и промежуточные вычисления для контроля за работой процедур приведены в табл. 2.2. По формуле (2.3) в точке x_1 = 1.908 было получено значение функции y_1 = 0.9687; в точке x_2 = 2.135 значение функции y_2 = 0.85550. По формуле (2.6) в точке x_1 = 2.248 было получено значение функции y_1 = =0.76912; в точке x_2 = 2.359 значение функции y_2 = 0.44132.

Таблица 2.2

№ п/п	Массивы исходных данных		Массивы конечных разностей		
	$x[i]$	$y[i]$	$r1[i]$	$r2[i]$	$r3[i]$
1	2.02500	0.91517	-0.012853	-0.0004161	-0.00000271
2	2.05000	0.90232	-0.013269	-0.0004188	-0.00000183
3	2.07500	0.88905	-0.013687	-0.0004206	-0.00000097
4	2.10000	0.87536	-0.014108	-0.0004216	-0.00000012
5	2.12500	0.86126	-0.014530	-0.0004217	0.00000072
6	2.15000	0.84673	-0.014951	-0.0004210	0.00000153
7	2.17500	0.83178	-0.015372	-0.0004195	0.00000232
8	2.20000	0.81640	-0.015792	-0.0004171	0.00000309
9	2.22500	0.80061	-0.016209	-0.0004140	0.00000384
10	2.25000	0.78440	-0.016623	-0.0004102	0.00000456
11	2.27500	0.76778	-0.017033	-0.0004056	0.00000526
12	2.30000	0.75075	-0.017439	-0.0004004	0.000000
13	2.32500	0.73331	-0.017839	0.000000	0.000000
14	2.35000	0.71547	0.000000	0.000000	0.000000
Сумма:			-0.199704	-0.0049866	0.00001570
$y_{14} - y_1$:		-0.199704	-0.004987	0.0000157	

§ 3. ИНТЕРПОЛЯЦИЯ ПО ЭЙТКЕНУ

Для построения интерполяционного полинома Лагранжа по функции $y(x)$, заданной таблично $\{y_i, x_i\}$, весьма эффективной является итерационная схема Эйткена:

$$y_{i+1} = \frac{y_{i+1}\cdot(x - x_i) - y_i \cdot (x - x_{i+1})}{x_{i+1} - x_i}, \; i = 0, \ldots, \; m\text{-}1,$$

где m - количество итераций, символ равно означает оператор присваивания. При этом для любого значения $x = x_p$ из интервала $[x_0, x_n]$ ордината $y_p = y_m$.

Рассмотренный алгоритм реализован в процедуре AITKEN, которую можно использовать как для равных, так и неравных интервалов $[x_i, x_{i+1}]$. Процедура строит полином Лагранжа $L_n(x)$ степени n таким образом, что $L_n(x_i) \equiv y(x_i)$. Поскольку в процедуре массив y используется для хранения промежуточных значений, его начальное значение не сохраняется.

Формальные параметры процедуры. *Входные*: $x[0{:}n]$, $y[0{:}n]$ (тип ***real***) - массивы абсцисс и ординат таблично заданной функции $y = [y_i(x_i)]$; n (тип ***integer***) - количество значений табличной функции (количество узлов интерполяции); xp (тип ***real***) - абсцисса, для которой вычисляется интерполированное значение функции. *Выходные*: l (тип ***real***) - интерполированное значение функции y (значение полинома Лагранжа в точке x_p).

```
PROCEDURE AITKEN(X:REAL;Y:MAS1;
          N:INTEGER;XP:REAL; VAR L : REAL);
VAR I,J,N1: INTEGER;
BEGIN
  N1:=N-1;
  FOR J:=0 TO N1 DO
     FOR I:=J+1 TO N DO
        Y[I]:=((XP-X[J])*Y[I]-(XP-X[I])*Y[J])/(X[I]-X[J]);
  L:=Y[N];
END { ****** AITKEN  ******}.
```

Процедура AITKEN получена в результате сокращения записи, оптимизации по временным затратам и перевода с языка *ALGOL* на язык *PASCAL* одноименной процедуры, которая опубликована в работе Агеева (1976) и является, в свою очередь, результатом исправления, сокращения и ординарной переработки алгоритма С.Дж. Мифсуда (1966). Процедура проверена для ряда элементарных функций на машине IBM PC/AT-286, некоторые примеры результатов тестирования представлены в табл.2.3. Погрешность интерполирования вычислялась как модуль разности интерполированного значения функции $L(x)$ в точке $x_n = x_p$ и ее значения, рассчитанного на ЭВМ с точностью $\varepsilon = 10^{-12}$.

Таблица 2.3

Функция	Шаг аргумента	Значение x	$L_n(x)$	Δ
$y = \exp(x)$	$\Delta x = \text{const}$	0.74	2.09593558	0
$y = \exp(x)$	$\Delta x \neq \text{const}$	0.74	2.09593558	0
$y = \exp(x)$	$\Delta x = \text{const}$	0.333	1.39514732	0
$y = \exp(x)$	$\Delta x \neq \text{const}$	0.333	1.39514625	1.073*e*-62
$y = \cos(x)$	$\Delta x = \text{const}$	0.74	0.73846864	1.192*e*-72
$y = \cos(x)$	$\Delta x \neq \text{const}$	0.74	0.73846852	0
$y = \cos(x)$	$\Delta x = \text{const}$	0.333	0.94506603	5.960*e*-82
$y = \cos(x)$	$\Delta x \neq \text{const}$	0.333	0.94506591	5.960*e*-82

§ 4. ИНТЕРПОЛЯЦИЯ ФУНКЦИИ КУБИЧЕСКИМИ СПЛАЙНАМИ

На практике редко проводят интерполяцию полиномами высших степеней, так как, во-первых, они дают значительные погрешности и, во-вторых, при бесконечном увеличении порядка n интерполяционного полинома $P_n(x)$ последовательность P_n расходится (согласно теореме Фабера). Этот факт впервые обнаружил Рунге в 1901 г. Им же была высказана идея об интерполировании движущимися полиномами. Однако, так как сплайн-интерполяция давала более хорошие и устойчивые результаты, эта идея не получила широкого распространения.

Наибольшей популярностью сегодня на практике при интерполяции пользуются полиномы 2-й и 3-й степени. Остановимся подробнее на полиномах 3-й степени, так называемых кубических сплайнах.

Кубические сплайн-функции моделируют очень старое механическое устройство, которым пользовались чертежники. Они брали гибкие рейки, изготовленные из достаточно упругого материала, например из дерева. Эти рейки закрепляли, подвешивая грузики в точках интерполяции, называемых *интерполяционными узлами*. Рейка или *механический сплайн* принимали форму с наименьшей потенциальной энергией. Последнее условие имеет свое математическое выражение: $f^{(IV)}(x) \equiv 0$. Если при этом сплайн н е разрушается, то тогда следует, что f и f' должны быть непрерывны на $[x_0, x_n]$. Из теории балок известно, что функция $f(x)$ между каждой парой заданных точек может быть предствлена полиномом 3-й степени

$$f(x_i) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3,$$

где $x_{i-1} < x < x_i$. При этом между каждой парой соседних узлов полиномы соединяются непрерывно (так же, как их первые и вторые производные).

Кубическую сплайн-функцию, удовлетворяющую условиям $f''(x_1) = f''(x_n) = 0$, называют *естественным куби-*

ческим сплайном. С математической точки зрения было доказано [Алберг, 1972], что она является единственной функцией, обладающей минимальной кривизной среди всех функций, интерполирующих данные точки и имеющих квадратично интегрируемую вторую производную. В этом смысле кубический сплайн будет самой гладкой функцией, интерполирующей заданные точки.

Построение кубического сплайна - простой и численно устойчивый процесс. Вычислительных схем, реализующих построение кубического сплайна по заданным значениям функции, в математике известно довольно много. Здесь нами рассмотрена одна из эффективных и простых вычислительных схем, обладающая свойством абсолютной устойчивости. Рассмотрим подынтервал $[x_i, x_{i+1}]$ и пусть

$$h_i = x_{i+1} - x_i;\ w = (x - x_i)/h\ ;\ W = 1 - w.$$

Очевидно, когда x пробегает значения из указанного подынтервала, w изменяется от 1 до 0, а W - от 0 до 1. Рассуждая, приходим к следующему представлению сплайна на этом подынтервале:

$$s(x) = w \cdot y_{i+1} + W \cdot y_i + h_i^2 \cdot [(w^3 - w) \cdot \sigma_{i+1} + (W^3 - W) \cdot \sigma_i],$$

где σ_i и σ_{i+1} - некоторые константы, которые предстоит определить.

Первый и второй члены в этой формуле соответствуют линейной интерполяции, а член, взятый в квадратные скобки,- это кубическая поправка, которая обеспечивает необходимую гладкость функции. Отметим, что она на концах подынтервала обращается в 0 и тогда $s(x_i) = y_i$; $s(x_{i+1}) = y_{i+1}$. Таким образом, $s(x)$ будет интерполировать заданные значения независимо от выбора σ_i .

Продифференцируем трижды $s(x)$ как сложную функцию от x и, учитывая, что $w' = 1/h_i$, $W' = -1/h_i$, получим производные $s(x)$ 1-, 2- и 3-го порядков

$$s'(x) = \frac{y_{i+1} - y_i}{h_i} + h_i \cdot \left[\left(3w^2 - 1\right) \cdot \sigma_{i+1} - \left(3W^2 - 1\right) \cdot \sigma_i \right],$$

$$s''(x) = 6 \cdot w \cdot \sigma_{i+1} + 6 \cdot W \cdot \sigma_i, \qquad s'''(x) = \frac{6 \cdot (\sigma_{i+1} - \sigma_i)}{h_i}.$$

Заметим, что $s''(x)$ - линейная функция, интерполирующая значения $6\sigma_i$ и $6\sigma_{i+1}$ в точке $x = x_i$. Следовательно, коэффициент $\sigma_i = s''(x_i)/6$, а $s'''(x)$ является константой на каждом подынтервале и поэтому четвертая производная тождественно равна 0.

На концах подынтервала $s'_+(x_i) = s'_-(x_i)$, так как ранее было выставлено требование непрерывности функции $s(x)$. Но, с другой стороны,

$s'_+(x_i) = \Delta_i - h_i(\sigma_{i+1} + 2\sigma_i)$ и $s'_-(x_{i+1}) = \Delta_i + h_i(2\sigma_{i+1} + \sigma_i)$,

где $\Delta_i = (y_{i+1} - y_i)/h_i$. С учетом равенства производных слева и справа в окрестности каждой точки для всех i получаем

$$\Delta_i - h_i(\sigma_{i+1} + 2\sigma_i) = \Delta_{i-1} - h_{i-1}(2\sigma_i + \sigma_{i-1}),$$

откуда имеем трехточечную систему из n - 2 линейных уравнений относительно σ_i

$$h_{i-1}\,\sigma_{i-1} + 2(h_{i-1} + h_i)\sigma_i + h_i\,\sigma_{i+1} = \Delta_i - \Delta_{i-1},$$

где $i = 1, 2, ..., n$.

Заменим третью производную по s разностью третьего порядка. Тогда условие для $s'''(x)$ в краевых точках $i = 1$ и $i = n$ запишется

$$(\sigma_2 - \sigma_1)/h_1 = \Delta_1^{(3)}\ (\sigma_n - \sigma_{n-1})/h_{n-1} = \Delta^{(3)}_{n-3}.$$

Здесь введены следующие обозначения:

$$\Delta_i^{(3)} = \frac{\Delta_{i+1}^{(2)} - \Delta_i^{(2)}}{x_{i+3} - x_i}; \quad \Delta_i^{(2)} = \frac{\Delta_{i+1} - \Delta_i}{x_{i+2} - x_i};$$

$$\Delta_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}.$$

Умножим полученные равенства для разностей третьего порядка на квадрат их знаменателей и получим:

$$-h_1\sigma_1 + h_1\sigma_2 = h_1^2\,\Delta_1^{(3)};\ h_{n-1}\,\sigma_n - h_{n-1}\,\sigma_{n-1} = -h_{n-1}^2\,\Delta^{(3)}_{n-3}.$$

$$\begin{pmatrix} -h_1 & h & 0 & 0 & \cdots & 0 \\ h_1 & 2(h_1 + h_2) & h_2 & 0 & \cdots & 0 \\ 0 & h_2 & 2(h_2 + h_3) & h_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & & \\ 0 & 0 & 0 & 0 & \cdots & h_{n-2} \\ 0 & 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ & \\ 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ h_{n-1} & -h_{n-1} \end{pmatrix} \times \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \vdots \\ \sigma_{n-1} \\ \sigma_n \end{pmatrix} = \begin{pmatrix} h_1^2 \cdot \Delta_1^{(3)} \\ \Delta_2 - \Delta_1 \\ \Delta_3 - \Delta_2 \\ \vdots \\ \Delta_{n-1} - \Delta_{n-2} \\ -h_{n-1}^2 \cdot \Delta_{n-3}^{(3)} \end{pmatrix}.$$

Таким образом, для сплайна с заданными выше граничными условиями коэффициенты σ_i удовлетворяют следующей системе из n линейных уравнений с n неизвестными:

Последнюю систему из n линейных уравнений с n неизвестными можно решать методом исключения, но однако если при решении использовать особые свойства матрицы системы (она трехдиагональная, симметричная, для любого выбора $x_1 < x_2 < ... < x_n$ невырожденная и диагонально доминирующая), то всегда существует единственное решение системы. Обозначим это решение через $\sigma_1, \sigma_2, ..., \sigma_n$. Можно также показать, поскольку для всех $x_1, x_2, ..., x_n$ матрица коэффициентов хорошо обусловлена, то Гауссово исключение приведет исходную систему к ленточной форме:

$$\begin{pmatrix} \alpha_1 & h_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \alpha_2 & h_2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \alpha_3 & h_3 & \cdots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \cdots & 0 & \alpha_n \end{pmatrix} \times \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \vdots \\ \sigma_n \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \beta_n \end{pmatrix}.$$

Здесь вычисляются

1) диагональные элементы α_i:

$$\alpha_1 = -h_1\ ;\ \alpha_i = 2(h_{i-1} + h_i) - h^2_{i-1} / \alpha_{i-1}\ ;$$
$$\alpha_n = -h_{n-1} - h^2_{n-1} / \alpha_{n-1};$$

1) правые части матрицы β_i:

$$\beta_1 = h^2_1\, \Delta^{(3)}_1;\ \beta_i = (\Delta_i - \Delta_{i-1}) - h_{i-1}\, \beta_{i-1} / \alpha_{i-1};$$
$$\beta_n = -h^2_{n-1}\, \Delta^{(3)}_{n-3} - h_{n-1}\beta_{n-1} / \alpha_{n-1},$$

(здесь для всех соотношений $i = 2, 3, \ldots, n\text{-}1$). Тогда искомые σ_i определяются через обратную подстановку

$$\sigma_n = \beta_n / \alpha_n;\ \sigma_i = (\beta_i - h_i\, \sigma_{i+1}) / \alpha_i$$

для всех $i = n - 1,\ n - 2,\ \ldots,\ 1$.

Вычислительные схемы, используемые для решения систем с трехдиагональной матрицей, традиционно называют *методом прогонки*.

В некоторых случаях по различным причинам предпочтительнее вычислять и сохранять коэффициенты b_i, c_i и d_i для функции $s(x)$, записанной в несколько ином виде :

$$s(x) = y_i + b_i\,(x - x_i) + c_i\,(x - x_i)^2 + d_i\,(x - x_i)^3\ ,$$
$$x \in [x_i,\ x_{i+1}],$$

где на каждом подынтервале $[x_i,\ x_{i+1}]$ указанные коэффициенты выражаются через σ_i и h_i:

$$b_i = (y_{i+1} - y_i) / h_i - h_i\,(\sigma_{i+1} + 2\sigma_i);\ c_i = 3\,\sigma_i;$$
$$d_i = (\sigma_{i+1} - \sigma_i) / h_i.$$

Использование такой формы хранения сплайнов упрощает некоторые операции с ними, например при вычислении производных и интегралов. Заметим попутно, что при $N = 1$ сплайн совпадает с многочленом Ньютона первой степени (см. § 2).

На практике, если дано достаточно много точек, то для построения кубического сплайна выбирают 4 последовательно расположенные, по которым строят первый полином. Затем сдвигаются на 3 точки и снова по 4 последовательным точкам строят следующий сплайн. И так поступают до тех пор, пока все точки не будут выбраны.

Один из методов вычисления кубической сплайн-функции реализован в подпрограмме SPLINE и процедуре-функции SEVAL [Дж. Форсайт и др., 1980], которые написаны на алгоритмическом языке *FORTRAN* и реализованы в БСП для ЭВМ БЭСМ-6 (к сожалению, эта эффективная процедура практически не используется для персональных ЭВМ!). В работе подпрограмма и процедура-функция приводятся на языке *PASCAL* (перевод выполнен авторами).

Перед применением программы следует убедиться, что среди точек нет повторяющихся. Если это не так, то повторяющиеся точки не должны быть использованы для построения *одного* сплайна. Для построения разных сплайнов такие точки использовать можно.

Процедуры SPLINE и SEVAL выполняются последовательно. Сначала в первой из названных процедур вычисляются коэффициенты для построения сплайна на выбранном отрезке, а во второй процедуре - значение функции в заданной точке. Отрезок, на котором строится сплайн, следует выбирать таким образом, чтобы заданная точка располагалась около его середины. Это уменьшит погрешность вычислений. Выбор последовательности из четырех точек осуществляется в основной программе.

Если требуется применить сплайн для построения непрерывной функции на всем пространстве экспериментальных данных, то в основной программе следует организовать скользящее окно из 4 точек с перекрытием в две точки. В области перекрытия рекомендуется вычислять значение функции как среднее из одного и другого сплайна.

Формальные параметры процедур. Процедура SPLINE. *Входные*: *n* (тип ***integer***) - количество точек, по которым строится сплайн (оно должно быть не меньше 2, но не больше 4); *x, y* (тип ***real***) - два массива размером $1\times n$, в которые предварительно записаны данные для построения очередного сплайна. *Выходные*: *b, c, d* (тип ***real***) - массивы коэффициентов кубического сплайна, если были предложены 4 точки.

Процедура SEVAL. *Входные*: *n* (тип ***integer***) - количество точек, по которым строится сплайн; *u* (тип ***real***) - точка, в которой надо определить значение функции; *x, y* (тип ***real***) - два массива размером $1\times n$, в которые предварительно записаны данные для построения очередного сплайна; *b, c, d* (тип ***real***) - массивы коэффициентов кубического сплайна, если были предложены 4 точки. *Выходные*: процедура-функция SEVAL возвращает вещественное значение функции в указанной точке *u*.

```
          { ***????????? ?????????? ???????????
         ??????? ?? ??????? ???????? 4 ?????? ***}
PROCEDURE SPLINE (N:INTEGER; X,Y: MAS; VAR B,C,D : MAS);
VAR NM1, I : INTEGER; T : REAL;
BEGIN     NM1 := N-1;
  IF N<2 THEN   EXIT;
  IF N>=3 THEN
          { *** ????????? ???????????? ??? ???????;
           *** ?????? 3-? ???????????? ????????: ***
            *** ? - ?????????, D - ????????????, ***
                  *** ? - ?????? ????? ***}
  BEGIN       D[1] := X[2]-X[1];
      C[2] := (Y[2]-Y[1]) / D[1];
      FOR I := 2 TO NM1 DO
      BEGIN  D[I] := X[I+1]-X[I];
          B[I] := 2.0*(D[I-1]+D[I]);
          C[I+1]:= (Y[I+1]-Y[I])/D[I];
          C[I] := C[I+1] - C[I];
      END;
                  { *** ????????? ??????? *** }
      B[1] := -D[1];       B[N] := -D[N-1];
```

```
    C[1] := 0.0;          C[N] := 0.0;
    IF N<>3 THEN
    BEGIN
        C[1] := C[3] / (X[4]-X[2])-C[2]/(X[3]-X[1]);
        C[N] := C[N-1]/(X[N]+X[N-2])-
                    C[N-2]/(X[N-1]-X[N-3]);
        C[1] := C[1]*D[1]*D[1]/(X[4]-X[1]);
        C[N] := -C[N]*D[N-1]*D[N-1]/(X[N]-X[N-3]);
    END;
                    { *** ?????? ??? *** }
    FOR I := 2 TO N DO
    BEGIN       T:= D[I-1]/B[I-1];
        B[I] := B[I] - T*D[I-1];      C[I] := C[I] - T*C[I-1];
    END;
              { *** ???????? ??????????? ***}
    C[N] := C[N] / B[N];
    FOR I:=NM1 DOWNTO 1 DO C[I] := (C[I]-D[I]*C[I+1])/B[I];
             { *** ? ?(I) ???????? SIG??(I) *** }
    B[N] := (Y[N]-Y[NM1]) / D[NM1] +
                D[NM1]*(C[NM1+2.0*C[N]);
    FOR I := 1 TO NM1 DO
    BEGIN
        B[I] := (Y[I+1]-Y[I]) / D[I]-D[I]*(C[I+1]+2*C[I]);
        D[I] := (C[I+1]-C[I])/D[I];
        C[I] := 3.0*C[I];
    END;
    C[N] := 3*C[N]; D[N] := D[N-1];
  END
  ELSE
  BEGIN  B[1] := (Y[2]-Y[1])/(X[2]-X[1]);
      C[1] := 0;       D[1] := 0;
      B[2] := B[1];   C[2] := 0;       D[2] := 0;
  END;
END.

          { *** ???????????? ????????? ???????? ***
        ***??????????? ??????? ?? ????? ???????*** }
FUNCTION SEVAL (N:INTEGER; U:REAL;
          X,Y,B,C,D:MAS): REAL;
LABEL 10,30;
VAR I,J,K: INTEGER; DX: REAL;
BEGIN
  I:= 1;
  IF I>= N THEN I:=1;
  IF U< X[I] THEN          GOTO 10;
  IF U<= X[I+1] THEN    GOTO 30;
10:   I := 1;  J := N+1;
  REPEAT          K:=(I+J) DIV 2;
      IF U<X[K] THEN  J:=K;
      IF U>=X[K] THEN I:=K;
  UNTIL J<=I+1;
30:   DX := U-X[I];
  SEVAL := Y[I] + DX*(B[I] + DX*(C[I] + DX*D[I]));
END.
```

Погрешность сплайн-интерполяции можно оценить, пользуясь методом, описанным в § 1.

Для проверки программы использовались данные § 1 (табл. 2.1), варианты А и Б. Здесь определялось значение функции в средней точке отрезка $[a, b]$ - $X_s = (x_3 + x_4)*0{,}5$. При этом предварительно выполнялась сплайн-интерполяция полиномом третьей степени. Результат сравнивался с приближением функции полиномом Лагранжа (алгоритм из § 1) для обоих вариантов.

Заметим, что расчетных точек в примере (табл. 2.1) дано 6, а для построения сплайна необходимо только 4, поэтому для каждого варианта строились два сплайна - первый по точкам 1 - 4 и второй по точкам 2 - 6. Таким образом обеспечивалось перекрытие в две точки по центру отрезка именно там, где задавалась точка X_s. Истинное значение Y определялось как среднее между значениями функции, полученными по первому и второму сплайну.

Точность вычислений задавалась 10^{-5}.

Интересно заметить, что сплайны (табл. 2.4) проходят более гладко по сравнению с полиномами пятой степени (табл. 2.1), что подтверждает тезис о неустойчивости процесса интерполирования полиномами высоких степеней.

Таблица 2.4

**** SPLINE 1 ****		**** SPLINE 2 ****	
$x[i]$	$y[i]$	$x[i]$	$y[i]$
Вариант А			
0.11000	9.00000	0.21000	4.70000
0.12500	7.96525	0.22583	4.42303
0.14000	7.09653	0.24167	4.15337
0.15500	6.37437	0.25750	3.89339
0.17000	5.78052	0.27333	3.64543
0.18500	5.29858	0.28917	3.41185
0.20000	4.91234	0.30500	3.19515
0.21500	4.60550	0.32083	2.99849
0.23000	4.35744	0.33667	2.82525
0.24500	4.14079	0.35250	2.67879
0.26000	3.92762	0.36833	2.56082
0.27500	3.69000	0.38417	2.46892
Расчетные точки варианта А			
0.25000	4.07072	0.25000	4.01518
Вариант Б			
0.10000	2.00000	0.30000	5.00000
0.12500	2.60260	0.32500	5.23547
0.15000	3.13472	0.35000	5.34000
0.17500	3.59948	0.37500	5.32453
0.20000	4.00000	0.40000	5.20000
0.22500	4.33906	0.42500	4.97687
0.25000	4.61806	0.45000	4.66375
0.27500	4.83802	0.47500	4.26875
0.30000	5.00000	0.50000	3.80000

0.32500	5.10677	0.52500	3.26797
0.35000	5.16806	0.55000	2.69250
0.37500	5.19531	0.57500	2.09578

Расчетные точки варианта Б

0.35000	5.16806	0.35000	5.34000

§ 5. ТРИГОНОМЕТРИЧЕСКАЯ ИНТЕРПОЛЯЦИЯ

Пусть функция $f(x)$ представлена на некотором отрезке $[0, 2\pi]$ таблицей значений $f(x_i)$ в равноотстоящих узлах $x_i = 2\pi\,(i - 1) / (2N + 1)$, $i = 1, 2, ..., 2N + 1$. Тогда *тригонометрическим интерполирующим многочленом* назовем многочлен степени m вида:

$$P_m(x) = \frac{a_0}{2} + \sum_{k=1}^{m}\left[a_k \cdot \cos(kx) + b_k \cdot \sin(kx)\right] \quad .$$

Задача тригонометрической интерполяции состоит в построении тригонометрического полинома, который бы наиболее полно удовлетворял условиям $P_m(x_i) = f(x_i)$ для любого $i = 1, 2, ..., 2N+1$.

Можно показать, что решением этой задачи является полином именно того вида, коэффициенты которого вычисляют по следующим формулам:

$$a_0 = \frac{1}{2m+1}\sum_{i=1}^{2m+1} f(x_i);$$

$$a_k = \frac{1}{2m+1}\sum_{i=1}^{2m+1} f(x_i)\cdot\cos\left(\frac{2\pi\cdot k}{2m+1}\cdot i\right);$$

$$b_k = \frac{1}{2m+1}\sum_{i=1}^{2m+1} f(x_i)\cdot\sin\left(\frac{2\pi\cdot k}{2m+1}\cdot i\right).$$

Для вычисления коэффициентов a_0, a_k и b_k можно воспользоваться подпрограммой, входящей в состав БСП ЕС ЭВМ. Она написана на языке *FORTRAN*, однако здесь приведен вариант на языке *PASCAL*.

Формальные параметры процедуры. *Входные*: *fnt* (тип ***real***) - массив из $2N + 1$ чисел, содержащий значения таблично заданной функции в равноотстоящих узлах $x_i = 2\pi(i - 1) / (2N+ +1)$, где $i = 1, 2, ..., 2N +1$; N (тип ***integer***) - число, задающее количество равноотстоящих узлов x_i; M (тип ***integer***) - количество вычисляемых пар коэффициентов Фурье. *Выходные*: A (тип ***real***) - массив из $M + 1$ чисел, содержащий значения коэффициентов Фурье $a_0, a_1, ..., a_m$; B (тип ***real***) - массив из $M +1$ чисел, содержащий значения коэффициентов Фурье $b_0, b_1, ..., b_m$; *IER* (тип ***integer***) - признак ошибки во входных параметрах: *IER* = 0 нет ошибки; *IER* = 1, если $M > N$; *IER* = 2, если $M < 0$.

```
PROCEDURE FORINT (FNT:MAS21;N,M:INTEGER;
          VAR A,B:MAS; VAR IER : INTEGER);
LABEL 70,100;
VAR I, AN, J : INTEGER; DN : BOOLEAN;
          Q,CONS,COEF,S1,S,C1,C,FNTZ,
          U0,U1,U2: REAL;
BEGIN
  IER := 0;
  IF M<0 THEN
  BEGIN
     IER := 2;
     EXIT;
  END;
  IF M-N>0 THEN
  BEGIN
     IER := 1;
     EXIT;
  END;
  AN := N;
  COEF := 2.0 / (2.0*AN+1.0);
  DN := FALSE;
  CONS := 3.14159265*COEF;
  S1 := SIN (CONS);     C1 := COS (CONS);
  C := 1.0;   S := 0.0;   J := 1;
  FNTZ := FNT[1];
  REPEAT
     U2 := 0.0;          U1 := 0.0;
     I := 2*N + 1;
     REPEAT
        U0 := FNT [I] + 2.0 *C*U1-U2;
        U2:= U1;        U1 := U0;
        DEC (I);
     UNTIL I<=1;
     A[J] := COEF*(FNTZ+C*U1-U2);
     B[J] := COEF*S*U1;
     IF J>=(M+1) THEN DN := TRUE
     ELSE
     BEGIN
        Q := C1*C-S1*S;          S := C1*S + S1*C;
        C := Q;              INC (J);
     END;
  UNTIL DN;
  A[1] := A[1]/2.0;
END.
```

Интересно заметить, что с возрастанием m многочлен $P_m(x)$ аппроксимирует функцию $f(x)$ с возрастающей степенью точности, т.е. ошибка интерполирования

$$|f(x) - P_m(x)| \to 0 \text{ при } m \to \infty .$$

Именно этим свойством тригонометрическая интерполяция отличается от полиноминальной, где при возрастании степени полинома сам многочлен может принимать какие угодно большие значения для всех точек x, кроме самих узлов интерполяции.

Тригонометрическая интерполяция полностью свободна от этого недостатка. Однако при увеличении точности уве-

личивается и количество необходимых вычислений, что делает метод тригонометрической интерполяции не совсем удобным. Тогда применяют специальные вычислительные методы для расчетов коэффициентов Фурье, которые намного уменьшают количество операций и ускоряют процесс построения полинома. Эти методы называют *быстрым преобразованием Фурье,* или БПФ. Более подробно алгоритм БПФ рассматривается в главе 9 § 2 (спектральный анализ временных рядов).

Для проверки работы процедуры строился интерполяционный тригонометрический полином, аппроксимирующий функцию, заданную таблицей значений в точках $x_i = 2\pi(i - 1) / (2N+1)$, где $i = 1, 2, ..., 2N+1$, если $2N + 1 = 21$ (табл. 2.5). Результаты вычисления коэффициентов ряда Фурье приводятся в табл. 2.6.

Таблица 2.5

№ п/п	*f*(*x*)	№ п/п	*f*(*x*)	№ п/п	*f*(*x*)
1	-5.0000	8	-1.3505	15	-2.6495
2	-4.8855	9	-1.2120	16	-3.4816
3	-4.4737	10	-1.1162	17	-4.1411
4	-3.7691	11	-1.0190	18	-4.533
5	-2.9241	12	-1.0257	19	-4.7249
6	-2.1578	13	-1.2797	20	-4.8312
7	-1.6283	14	-1.8456	21	-4.9368

Таблица 2.6

Номер *i*	*a* (*i*)	*b* (*i*)
0	-2.99965	0.00000
1	-2.00051	0.75047
2	0.00005	-0.00069
3	0.00043	-0.24946
4	-0.00069	-0.00010
5	0.00057	-0.00039

§ 6. ПОЛИНОМИАЛЬНАЯ АППРОКСИМАЦИЯ ПРОИЗВОДНЫХ ЛЮБОГО ПОРЯДКА ТАБЛИЧНО ЗАДАННОЙ ФУНКЦИИ

Если имеется таблично заданная функция $y = \{y_i; x_i\}$, $i = 1, ..., n$, то, построив n-точечный интерполяционный полином Лагранжа (см. § 1), можно найти аналитические выражения для k-й производной каждого коэффициента в интерполяционной формуле и вычислить это выражение в заданной точке x_p. Тогда, имея табличные значения функции y_i в узловых точках x_i, легко получить значение как самой функции ($k = 0$), так и ее производной любого порядка в точке x_p по формуле

$$y^{(n)} = \sum_{i=1}^{n} C_i y_i \,, \qquad (2.8)$$

где C_i - вычисленные значения коэффициентов полиномиальной аппроксимации производной в соответствующих узлах.

По приведенной процедуре DERCOEF можно вычислить коэффициенты C_i для n ординат, соответствующих абсциссам x_i в n-точечном конечно-разностном выражении для k-й производной, значение которой необходимо определить в точке x_p.

При $k = 0$ вычисляются интерполяционные коэффициенты Лагранжа.

Формальные параметры процедуры. *Входные*: *k* (тип ***integer***) - порядок производной; *n* (тип ***integer***) - количество точек конечно-разностного представления производной; *kk* (тип ***integer***) - параметр, идентифицирующий формулу вычисления факториала (см. описание параметров процедуры-функции *fct* в п. 1.1, гл. 1); *xp* (тип ***real***) - абсцисса, в которой вычисляется производная; *xtab*[1:*n*] (тип ***real***) - абсциссы x_i точек, в которых вычисляются коэффициенты конечно-разностного выражения *k*-й производной. *Выходные*: *c*[1:*n*] (тип *real*) - массив коэффициентов C_i *n*-точечной конечно-разностной аппроксимации *k*-й производной.

```
PROCEDURE DERCOEF(K,N,KK,XP:REAL;XTAB:MASA;
          VAR C: MAS1);
VAR FK,SUM,DN,PT : REAL; I,T,J,M,H : INTEGER;
          XU ARRAY [1..N-1] OF INTEGER;
LABEL ZZ, XX, YY,AA;
BEGIN
          {*** □□□□□□□□□□ □□□□□□□□□□ N! □ □□□□□□□
               □□□□□□□□□-□□□□□□□ FCT; ****}
  FK:=FCT(N,KK);
  T:=N-K-1;
  IF T<0 THEN
     GO TO ZZ;
  FOR J:=1 STEP 1 UNTIL N DO
     BEGIN
        SUM:=0;
        DN:=PT:=1;
        FOR I:=1 STEP 1 UNTIL N DO
           IF I<>J THEN
              DN:=DN*(XTAB[J]-XTAB[I]);
        IF T=0 THEN
           GO TO YY;
        M:=H:=1;
AA:
        IF (H=J)X(XTAB[H]=XP) THEN
        BEGIN
           H:=H+1;
           GO TO AA
        END;
     IF H>N THEN
     BEGIN
```

```
        M:=M-1;
        IF M<0 THEN
            GO TO XX;
        H:=XU[M]+1;
        GO TO AA
    END;
    XU[M]:=H;
    M:=M+1;
    IF M<T THEN
    BEGIN
        H:=H+1;   GO TO AA
    END;
    FOR I:=1 STEP 1 UNTIL T DO
        PT:=PT*(XP-XTAB[XU[I]]);
    SUM:=SUM+PT;
    M:=T;
    PT:=1;
    H:=XU[T]+1;
    GO TO AA;
YY: SUM:=1;
XX: C[J]:=SUM*FK/DN
  END {*** J ***};
 EXIT;
ZZ: FOR I:=1 STEP 1 UNTIL N DO     C[I]:=0;
END { *** DERCOEF *** }.
```

Процедура DERCOEF получена в результате перевода с языка *ALGOL* на язык *PASCAL* предварительно сокращенной и подвергнутой некоторой модификации (в целях оптимизации временных затрат) процедуры DICOL [Агеев и др., 1976], которая является сокращенным и ординарно переработанным алгоритмом Т.П. Жиаммо (1962)**.** Тестирование процедуры проводилось на машине IBM PC/AT-386 для тех же исходных данных, что и в работе Агеева (1976), при этом для $n = 3$ были получены совершенно идентичные результаты, в которых погрешность определяется только ошибками округления, т.е. зависит лишь от точности представления чисел в ЭВМ.

Замечание. В исследовании алгоритма Т.П. Жиаммо (1962) Е.С. Кларк (1963) отметил, что при возрастании n ($n > 12$) и k точность результатов снижается и, кроме того, рост n приводит к быстрому росту времени выполнения процедуры. Если $k = 0$, то при увеличении n от 4 до 8 и 12 возрастает количество машинных операций, необходимых для выполнения алгоритма, соответственно от $1.3\,e + 3$ до $3.8\,e + 4$ и $8.6\,e + 5$.

3.

Существует много машинных методов интегрирования и дифференцирования. Метод, более всего подходящий для данной задачи, в значительной степени зависит от информации о соответствующей функции. Особый интерес с точки зрения вычислительной математики при этом вызывают задачи для одной функции $f(x)$ одного действительного переменного $x \in [a, b]$, которые условно подразделяют на четыре категории.

1. Значения функции $f(x)$ заданы только на *фиксированном* конечном множестве точек x_i интервала $[a, b]$.

2. Функция $f(x)$ определена и может быть вычислена для любого *действительного* x из интервала $[a, b]$, однако первообразной для нее не существует.

3. Определение функции может быть аналитически продолжено на *комплексные* значения x.

4. Имеется явная формула, пригодная для символического манипулирования.

Функции, входящие в первую категорию, чаще всего появляются в результате некоторого эксперимента для заданных x_i, которые, как правило, распределены неравномерно или расположены в виде таблицы.

Поэтому ясно, что для функций первых двух категорий численное дифференцирование является более трудной вычислительной задачей, чем численное интегрирование. Очевидно причина в том, что математическая операция численного дифференцирования увеличивает любую ошибку, присутствующую в данных, в то время как численное интегрирование обычно сглаживает и уменьшает такие ошибки. Если значения функции известны или могут быть вычислены с большой точностью и если требуются производные относительно невысоких порядков, то часто методы, основанные на интерполяции сплайнами или полиномами, дают вполне удовлетворительные результаты. Однако если нужны производные высокого порядка или если значения функции *зашумлены*, то результаты могут быть, мягко говоря, неточными.

Примером функций, входящих в третью категорию, могли бы служить сложные выражения, составленные из тригонометрических или других элементарных функций. Для задач этого типа следует извлекать выгоду из комплексного расширения, если оно доступно. Тогда производные функции $f(x)$ можно выразить через комплексные контурные интегралы и сглаживающий эффект численного интегрирования можно использовать для получения хороших приближений к производным высокого порядка [Линес, Моулер, 1967; Линес, Санде, 1971].

В основном настоящая глава посвящена методам работы с функциями 1, 2 и 3-й категории.

Задачи 4-й категории в данной работе не рассматриваются, но сведения о данных такого типа можно найти в работах Мозеса (1972).

Для того чтобы избежать путаницы с *численным интегрированием* обыкновенных дифференциальных уравнений, для численного приближения определенных интегралов используется понятие *квадратура*.

§ 1. ЧИСЛЕННОЕ ДИФФЕРЕНЦИРОВАНИЕ С ПОМОЩЬЮ ИНТЕРПОЛЯЦИОННЫХ ФОРМУЛ НЬЮТОНА И ЛАГРАНЖА

Пусть некоторая функция $F(x)$ задана таблично в $n+1$ точке, расположенной на интервале $[a, b]$. Требуется вычислить первую и вторую производные данной функции в заранее определенных точках.

В качестве аппроксимирующей функции выберем интерполяционный многочлен. Если узлы интерполяции расположены не равномерно, то таким многочленом могут быть полиномы Лагранжа или Лежандра, а если равномерно, то лучше использовать полином Ньютона, так как он дает меньшую вычислительную погрешность по сравнению с другими.

Пусть узлы x_i, в которых известны значения функции $F(x_i)$, расположены равномерно на интервале $[a, b]$, т.е. $x_{i+1}-x_i = h$, где $i = 0, 1, ..., n$. Тогда в качестве интерполирующей функции $P_n(x)$ выберем полином Ньютона

$$f(x) = y_0 + \Delta_1 \cdot q + \frac{\Delta_2 \cdot q \cdot (q-1)}{2!} + \ldots + \frac{\Delta_n \cdot q \cdot (q-1) \cdot \ldots \cdot (q-n+1)}{n!},$$

где $q = (x - x_0) / h$; $h = x_{i_{+1}} - x_{i.}$. Здесь использована запись первой интерполяционной формулы Ньютона (см. § 2, гл. 2). Раскроем скобки, приведем подобные, получим

$$f(x) = y_0 + \Delta_1 \cdot q + \frac{\Delta_2 \cdot \left(q^2 - q\right)}{2!} + \frac{\Delta_3 \cdot \left(q^3 - 3q^2 + 2q\right)}{3!} + \frac{\Delta_4 \cdot \left(q^4 - 6q^3 + 11q^2 - 5q\right)}{4!} + \ldots \quad (3.1)$$

Заметим, что $\frac{df(x)}{dx} = \frac{df(x)}{dq} \cdot \frac{dq}{dx} = \frac{1}{h} \cdot \frac{df(x)}{dq}$.

Продифференцируем $f(x)$ из выражения (3.1) и с учетом последнего замечания получим выражение для первой производной

$$f'(x) \cong \frac{1}{h}\left(\Delta_1 + \frac{\Delta_2 \cdot (2q-1)}{2} + \frac{\Delta_3 \cdot \left(3q^2 - 6q + 2\right)}{6} + \right.$$
$$\left. + \frac{\Delta_4 \cdot \left(4q^3 - 9q^2 + 11q - 3\right)}{12} + \ldots \right) . \qquad (3.2)$$

Продифференцируем последнее выражение еще раз:

$$f''(x) \cong \frac{1}{h^2}\left(\Delta_2 + \Delta_3 \cdot (q-1) + \frac{\Delta_4 \cdot \left(6q^2 - 18q + 11\right)}{12} + \ldots \right) . \qquad (3.3)$$

Здесь учитывалось, что

$$\frac{df'(x)}{dx} = \frac{df'(x)}{dq} \cdot \frac{dq}{dx} = \frac{1}{h} \cdot \frac{df'(x)}{dq} .$$

Аналогично можно определить и следующие производные функции $f(x)$. Но при вычислении производных более высоких порядков следует учитывать большее количество членов ряда Ньютона для обеспечения точности результата.

Если производные функции $f(x)$ надо определить в узлах интерполяции, т.е. в $x = x_i$, то тогда формулы (3.2), (3.3) для первой и второй производных заметно упрощаются, так как в этом случае $q \equiv 0$:

$$f'(x) = (\Delta_1 - \Delta_2/2 + \Delta_3/3 - \Delta_4/4 + \ldots) / h ; \qquad (3.4)$$

$$f''(x) = (\Delta_2 - \Delta_3 + 11\,\Delta_4/12 - 5\,\Delta_5/6 + \ldots) / h^2 . \qquad (3.5)$$

Погрешность в определении производных можно приближенно оценить как [Плис, Сливина, 1983]

$$R_k(x) \cong \frac{1}{h} \cdot \frac{q \cdot (q-1) \times \ldots \times (q-k)}{(k+1)!} \cdot y^{(k+1)}(\xi), \forall \xi \in [a,b] \setminus x_i .$$

Для того чтобы получить значения производных функции $f(x)$ в точках, расположенных в конце таблицы, следует, как и в § 2, гл. 2, воспользоваться второй интерполяционной формулой Ньютона. Повторив аккуратно и последовательно все рассуждения, получим для первой производной следующее выражение:

$$f'(x) \cong \frac{1}{h}\left(\Delta_1(y_{n-1}) + \frac{\Delta_2(y_{n-2}) \cdot (2q+1)}{2} + \right.$$
$$+ \frac{\Delta_3(y_{n-3}) \cdot \left(3q^2 + 6q + 2\right)}{6} +$$
$$\left. + \frac{\Delta_4(y_{n-4}) \cdot \left(2q^3 + 9q^2 + 11q + 3\right)}{12} + \ldots \right) . \qquad (3.6)$$

Если же для равноотстоящей системы узлов x_i построить полином Лагранжа, то, записав его в виде

$$f(x) \cong \sum_{i=0}^{n} \frac{(-1)^{n-i}}{i! \cdot (n-i)!} \cdot \frac{q \cdot (q-1) \cdot (q-2) \times \ldots \times (q-n)}{q-i} \cdot y_i ,$$

где $q = (x - x_0)/h$ - шаг интерполяции; $h = x_{i+1} - x_i$ - расстояние между узлами интерполяции; $y_i = f(x_i)$- значения функции $f(x)$ в заданных узлах x_i, и с учетом, что $dx/dq = h$, получим выражение для первой производной:

$$f'(x) \cong \frac{1}{h} \cdot \sum_{i=0}^{n} \frac{(-1)^{n-i}}{i! \cdot (n-i)!} \cdot y_i \cdot \frac{d}{dq} \frac{q \cdot (q-1) \cdot (q-2) \times \ldots \times (q-n)}{q-i} .$$

Для оценки погрешности последнего выражения можно воспользоваться формулой, приведенной в работе Плис и Сливиной (1983)

$$R_k(x) \cong (-1)^{n-i} \cdot h^n \cdot \frac{i!(n-i)!}{(n+1)!} \cdot y^{(n+1)}(\xi), \forall \xi \in [a,b] \setminus x_i .$$

В заключение отметим, что рассматриваемые здесь формулы численного дифференцирования являются менее точными по сравнению с интерполяционными (см. § 1 и 2 в гл. 2). Однако их применяют в практических расчетах достаточно часто, так как они удобны, просты, не требуют специальных программ. И, что особенно важно, позволяют быстро сделать предварительные (прикидочные) расчеты.

Процедуры NEW1, NEW2, NEWTON из § 1 и 2 (гл.2) вполне пригодны для вычисления производных. Но с учетом решаемой задачи они должны быть несколько модифицированы, хотя формальные параметры остаются у процедур без изменений.

```
{***?????? ???????????????? ??????? ???????**}
FUNCTION NEW1 (KEY:INTEGER;X,Y,R1,R2,R3,R4: MAS;
               X1:REAL):REAL;
LABEL 30;
VAR I,J,K : INTEGER; Q : REAL;
BEGIN
  I := 1;
  J := 14;
  IF X1 < X[I] THEN
     GOTO 30;
  REPEAT
     K := (I+J) DIV 2;
     IF X1< X[K] THEN
        J := K;
     IF X1>=X[K] THEN
        I := K;
  UNTIL J <= I+1;
30:
  Q := (X1 - X[I]) / (X[2]-X[1]);
              { ****????? ???? ?????? ?????????****}
CASE KEY OF
     0:NEW1:=Y[I]+Q*(R1[I]+(Q-1)*(R2[I]/2.0+
          (Q-2)*R3[I]/6.0));
     1: NEW1 := R1[I] + 0.5*(2*Q-1)*R2[I] +
          ((3*Q-6)*Q+2)*R3[I]/6+
```

```
                (((2*Q-9)*Q+11)*Q-3)*R4[I]/12;
        2: NEW1 := R2[I]+(Q-1)*R3[I]+
                ((6*Q-8)*Q+11)*R4[I]/12;
    END;
END.
          {***Вычисление интеграла по формулам***}
FUNCTION NEW2 (KEY:INTEGER;X,Y,R1,R2,R3,R4:MAS;
               X1:REAL):REAL;
LABEL 30;
VAR I,J,K : INTEGER; Q : REAL;
BEGIN
  I := 1;
  J := 14;
  IF X1 > X[J] THEN GOTO 30;
 {****Поиск отрезка содержащего****}
  REPEAT
      K := (I+J) DIV 2;
      IF X1< X[K] THEN J := K;
      IF X1>=X[K] THEN I := K;
  UNTIL J <= I+1;
 30:
  Q := (X1 - X[I]) / (X[2]-X[1]);
              {****Выбор нужной****}
  CASE KEY OF
        0: NEW2 := Y[J]+Q*(R1[J-1]+
                (Q+1)*(R2[J-2]/2.0+(Q+2)*R3[J-3]/6.0));
        1: NEW2 := R1[J-1]+0.5*(2*Q+
                1)*R2[J-2]+((3*Q+6)*Q+2)*R3[J-3]/6+
                (((2*Q+9)*Q+11)*Q+3)*R4[J-4]/12;
        2: NEW2 := R2[J-2]+(Q+
                1)*R3[J-3]+((6*Q+18)*Q+11)*R4[J-4]/12;
    END;
END.
```

Для проверки и тестирования процедур вычислялись значения первой и второй производных функции при заданных значениях аргумента с использованием первой или второй интерполяционных формул Ньютона. Расчеты контролировались при составлении таблицы разностей. Вычисления выполнялись с точностью 10^{-5}. Данные взяты из табл. 2.2. Результаты работы процедур приводятся в табл. 3.1.

Таблица 3.1

Первая инт.формула Ньютона			Вторая инт.формула Ньютона		
x_i	y'	y''	x_i	y'	y''
1.90800	-0.01067	-0.00043	2.24800	-0.01679	-0.00041
2.13500	-0.01449	-0.00042	2.35900	-0.01999	0.00025

§ 2. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ ПО ПРОСТЕЙШИМ ФОРМУЛАМ

Пусть теперь некоторый конечный интервал $[a, b]$ на оси Ox разбит на n подынтервалов $[x_i, x_{i+1}]$, которые в дальнейшем будем называть элементарными отрезками. Ясно, что без ограничения общности можно положить $x_0 = a$; $x_n = b$ и $x_0 < <x_1 < ... < x_n$. Через h_i обозначим длину элементарного отрезка $(x_{i+1} - x_i)$. Если заданный отрезок $[a, b]$ разбит равномерно, то тогда h_i будет постоянна для любой $\xi_i \in [a, b]$. Пусть теперь на $[a, b]$ определена некоторая функция $f(x)$. Предположим, что необходимо найти приближение к определенному интегралу, которое обозначим $\Im(f) = \int_a^b f(x)dx$. Очевидно также, что если $f(x)$ непрерывна $\forall x_i \in [a, b]$, то тогда $\Im(f)$ можно представить как $\Im(f) = \sum_{i=0}^{N} \Im_i$, где $\Im_i$ - интеграл функции $f(x)$ на элементарном отрезке $[x_{i+1}, x_i]$, т.е.:

$$\Im_i = \Im_i(f) = \int_{x_i}^{x_{i+1}} f(x)dx .$$

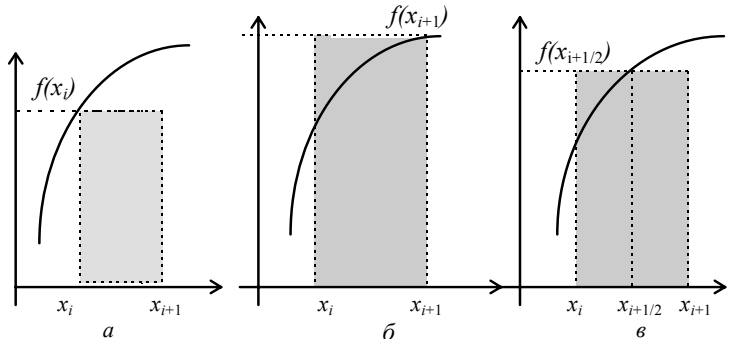

Рис. 3.1. *Вычисление интеграла по формулам прямоугольников*

Всякая простая формула, аппроксимирующая отдельный интеграл $\Im_i$, называется *квадратурной*. *Составная квадратурная формула* - это формула, дающая приближение интеграла $\Im(f)$ в виде суммы приближений интегралами $\Im_i$ по данной квадратурной формуле.

Двумя простейшими квадратурными формулами являются формулы *прямоугольников и трапеций*, которые в ряде случаев оказываются наиболее эффективными.

Известны три разновидности формул прямоугольников: это формулы *левых, правых и средних прямоугольников*. Все они основаны на аппроксимации каждого

интеграла $\mathfrak{I}_i$ площадью прямоугольника, одной из сторон которого является h_i, а второй - либо значение функции на левом конце отрезка (рис.3.1, *а*), либо значение функции на правом конце отрезка (рис. 3.1, *б*), либо значение функции в средней точке отрезка (рис. 3.1, *в*).

Квадратурные формулы, аппроксимирующие $\mathfrak{I}_i$, будут иметь вид:

левых прямоугольников: $\mathfrak{I}_i = h_i f(x_i)$;

правых прямоугольников: $\mathfrak{I}_i = h_i f(x_{i+1})$;

средних прямоугольников: $\mathfrak{I}_i = h_i f(x_{i+1/2})$.

С учетом представления $\mathfrak{I}_i$ на элементарном отрезке составные квадратурные формулы прямоугольников могут быть записаны так:

левых прямоугольников

$$\sum_{i=0}^{n-1} \mathfrak{I}_i = \sum_{i=0}^{n-1} h_i \cdot f(x_i);$$

правых прямоугольников

$$\sum_{i=1}^{n} \mathfrak{I}_i = \sum_{i=1}^{n} h_i \cdot f(x_{i+1});$$

средних прямоугольников

$$\mathfrak{I} = \sum_{i=0}^{n-1} \mathfrak{I}_i = \sum_{i=0}^{n-1} h_i \cdot f(x_{i+h/2}).$$

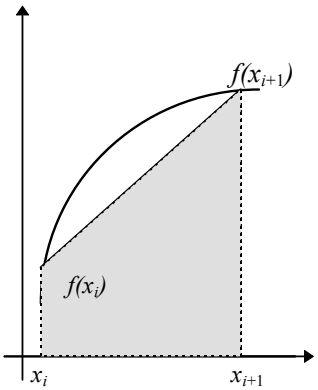

Рис. 3.2. *Геометрическая интерпретация "метода трапеций"*

В формуле трапеций используются значения функции в концевых точках элементарных отрезков. В этом случае $\mathfrak{I}_i$ аппроксимируется площадью трапеции с основаниями $f(x_i)$ и $f(x_{i+1})$ и высотой Δx (рис. 3.2).

Тогда площадь фигуры может быть определена из формулы площади прямоугольной трапеции

$$S_i = (f_i + f_{i+1})\, h_i/2.$$

Если теперь просуммируем последнюю формулу по всем элементарным отрезкам, то получим с учетом выполненных элементарных преобразований следующее выражение:

$$\mathfrak{I} = \sum_{i=1}^{n} \mathfrak{I}_i = (b-a)\cdot\left[(f_a + f_b)/2 + \sum_{i=2}^{n-1}\left(f(x_i) + f(x_{i+1})\right)\right]/n \quad .$$

Заметим, что при бесконечном уменьшении длин элементарных отрезков формулы обоих типов (прямоугольников и трапеций) сходятся к точному значению интеграла $\mathfrak{I}$. Однако не ясно, как быстро они сходятся ? Попытаемся выяснить данный вопрос, воспользовавшись разложением функции в ряд Тейлора относительно центра элементарного отрезка $[x_i, x_{i+1}]$

$$f(x) = f(y_i) + (x - y_i) f'(y_i) + (x - y_i)^2 f''(y_i)/2 + (x - y_i)^3 f'''(y_i)/6 + (x-y_i)^4 f^{(IV)}(y_i)/24 + \ldots$$

Затем, проинтегрировав полученный ряд по каждому из отрезков в предположении, что оставшиеся члены ряда намного меньше выписанных, с учетом значений коэффициентов ряда Тейлора на элементарном отрезке

$$\int_{X_i}^{X_{i+1}} (x - y_i)^p\, dx = \begin{cases} h_i, p = 0 \ ; \\ 0, p = 1 \ ; \\ h_i^3/12, p = 2 \ ; \\ 0, p = 3 \ ; \\ h_i^5/80, p = 4, \end{cases}$$

получим

$$\int_{X_i}^{X_{i+1}} f(x)dx = h_i \cdot f(y_i) + \frac{1}{24} h_i^3 \cdot f''(y_i) + \frac{1}{1920} h_i^5 \cdot f^{(IV)}(y_i) + \ldots$$

Член $\frac{1}{24} h_i^3 \cdot f''(y_i)$ показывает ошибку формулы прямоугольников без учета членов более высокого порядка. Если теперь подставим в формулу трапеций значения функции в точках $x = x_i$ и $x = x_{i+1}$, то получим

$$\int_{X_i}^{X_{i+1}} f(x)dx = h_i \frac{f(x_i) + f(x_{i+1})}{2} + \frac{1}{12} h_i^3 \cdot f''(y_i) + \ldots$$

Можно видеть, что ошибки формул средних прямоугольников и трапеций одного порядка, т.е. дают почти одинаковую точность.

Примеров вычисления определенных интегралов по простейшим формулам (прямоугольников и трапеций) можно привести множество. Ни одна БСП не обходится без этих процедур. Одним из примеров такой вычислительной процедуры может служить следующая:

```
PROCEDURE INTLPS (VAR S:REAL; K:INTEGER);
BEGIN
  S := 0.0;
  X := A;
  FOR I := 0 TO N DO
  BEGIN
     IF K <>3 THEN X := A+ H*I
          ELSE X:=A+H/2+H*I;
     F := FUNC (A,B,X);
     IF K=4 THEN F := F/2.0;
     IF (I=0) AND (K<>2) THEN S := S + F;
     IF (I=N) AND ((K=2) OR (K=4)) THEN S := S + F;
     IF (I<>0) AND (I<>N) THEN
          IF K<>4 THEN S := S+F
```

```
            ELSE S := S+F*2;
    END;   S := S * H;
  END.
```

Эта процедура обращается к процедуре-функции FUNC для вычисления подынтегральной функции в заданных

заданных точках. Для каждого из вариантов эта процедура-функция должна быть своя.

Формальные параметры процедуры. *Входные*: *X*0, *X*1 (тип ***real***) - границы отрезка, на котором вычисляется определенный интеграл; *N* (тип ***integer***) - количество узлов (элементарных отрезков), на которые разбивается заданный интервал [*a, b*]; *к* (тип ***integer***) - параметр, определяющий тип расчета (*к* = 1 - формула левых прямоугольников; *к* = 2 - формула правых прямоугольников; *к* = 3 - формула средних прямоугольников; *к* = 4 - формула трапеций). *Выходные*: *FX* (тип ***real***) - массив значений функции в заданных узлах (x_i); *fint* (тип ***real***) - вычисленные значения интеграла; *err* (тип ***real***) - погрешность вычисленного значения.

Программу проверяли по формулам левых, правых и средних прямоугольников при разном количестве точек разбиения. Для сравнения интеграл $\int_0^1 \frac{dx}{10+x^2}$ вычисляли по формуле трапеций с теми же параметрами разбиения отрезка. Точность вычислений полагалась 10^{-5}. Результаты работы данной программы приводятся в табл. 3.2.

Таблица 3.2

Кол-во узлов	Формула прямоугольников			Формула трапеций
	левых	правых	средних	
2	0.0987804878	0.0942350333	0.0970267191	0.0965077605
4	0.0979036035	0.0956308762	0.0968965226	0.0967672398
8	0.0974000630	0.0962636994	0.0968641736	0.0968318812
12	0.0971949075	0.0964956068	0.0968581918	0.0968438419
20	0.0970772373	0.0966226919		
30	0.0970033929	0.0967003626		
40	0.0969661837	0.0967389110		
50	0.0969437664	0.0967619482		
60	0.0969287832	0.0967772680		
65	0.0969230123	0.0967831521		
68	0.0969199553	0.0967862654		
69		0.0967872428		

§ 3. ВЫЧИСЛЕНИЕ ОПРЕДЕЛЕННОГО ИНТЕГРАЛА МЕТОДОМ СИМСОНА

Метод Симпсона часто называют в литературе *методом парабол*. Очевидно, что точность вычислений приближенного интеграла возрастет по сравнению с точностью вычислений, выполненных по формулами трапеций и прямоугольников, если подынтегральную функцию $f(x)$ заменить на отрезке $[x_i,x_{i+1}]$ квадратичной параболой, которая в узлах разбиения x_i принимает значения $f(x_i)$ и при этом $x_0 = a; f(x_0) = f(a) = y_0; x_n = b; f(x_n) = f(b) = y_n$.

Разобьем равномерно отрезок $[a, b]$ на N элементарных отрезков $[x_i,x_{i+1}]$ и на каждом из них заменим подынтегральную функцию $f(x)$ интерполяционным многочленом Ньютона (или Лагранжа, в принципе, без разницы!) второй степени. Тогда для каждого элементарного отрезка $[x_i,x_{i+1}]$ имеем следующее:

$$\Im_i = \int_{X_i}^{X_{i+1}} f(x)dx = \int_{X_i}^{X_{i+1}} P_2(x)dx =$$

$$= \int_{X_i}^{X_{i+1}} \left(y_i + \frac{\Delta(y_i)}{h}(x - x_i) + \frac{\Delta_2(y_i)}{2h^2}(x - x_i)(x - x_{i+1}) \right) dx =$$

$$= 2hy_i + \frac{\Delta(y\ )}{h} \cdot \frac{4h^2}{2} + \frac{\Delta_2(y_i)}{2h^2} \cdot \frac{8h^3}{3} - \frac{\Delta_2(y_i)}{2h^2} \cdot \frac{4h^3}{2} =$$

$$= 2hy_i + 2h\Delta(y_i) + h\Delta_2(y_i)/3 =$$

$$= h \cdot (y_i + 4y_{i+1/2} + y_{i+1})/6 \ .$$

Так же, как и в § 2, просуммируем полученное выражение по всем элементарным отрезкам, и если подставим $h = (b - a) / n$, то окончательно получим

$$\Im_i = \sum_{i=0}^{N} h \cdot (y_i + 4y_{i+1/2} + y_{i+1})/6 =$$

$$= \frac{b-a}{6N} \cdot \left((y_0 + y_n) + 4\sum_{i=0}^{N} y_{i+1/2} + 2\sum_{i=1}^{N-1} y_i \right).$$

Данное выражение называется *формулой Симсона*. Оно относится к формулам повышенной точности и является точной для многочленов второй и третьей степени [Бахвалов, 1973]. Погрешность формулы Симсона оценивается, как и в § 2, по формуле Тейлора и имеет вид [Данилина и др., 1976]

$$R = -\frac{h^5}{90}\cdot y^{(IV)}(\xi), \forall \xi \in [a,b]\setminus x_i.$$

Если воспользоваться процедурой INTLPS (из § 2), то после некоторой модификации процедуры будем иметь следующее:

```
PROCEDURE INTLPS (VAR S:REAL; K:INTEGER);
BEGIN
  S := 0.0;
  X := A;
  IF K<5 THEN
     FOR I := 0 TO N DO
     BEGIN
        IF K <>3 THEN X := A+ H*I
           ELSE X:=A+H/2+H*I;
        F := FUNC (X);
        IF K=4 THEN F := F/2.0;
        IF (I=0) AND (K<>2) THEN S := S + F;
        IF (I=N) AND ((K=2) OR (K=4)) THEN S := S + F;
        IF (I<>0) AND (I<>N) THEN
        IF K<>4 THEN S := S+F
           ELSE S := S + F*2;
     END
  ELSE
  BEGIN
     FOR I := 1 TO N-1 DO
        S := 2*FUNC(X+H*I) + 4*FUNC(X+H*I-H/2) + S;
     S := S + 4*FUNC(B-H/2) + FUNC(A) + FUNC(B);
     S := S / 6;
  END;
  S := S * H;
END.
```

По сравнению с § 2 в данном случае изменился интервал значений параметра *k*, который определяет тип расчетной формулы. Теперь этот параметр должен быть равным 5, если применяется формула Симсона. Все остальные параметры процедуры (входные и выходные) остались без изменений.

Для контроля и проверки процедуры использовалась функция из § 2, значение которой здесь вычислялось методом Симсона. Получены следующие результаты: при $n = 5$ интеграл равен 0.0968534164, а при $n = 2$: 0.0968537329, т.е. точность 0,00001 достигалась уже при разбиении отрезка пополам.

§ 4. ИНТЕГРИРОВАНИЕ КВАДРАТУРНЫМИ ФОРМУЛАМИ НЬЮТОНА - КОТЕСА И МЕТОДОМ "ТРИ ВОСЬМЫХ"

Пусть некоторая функция $f(x)$, как и раньше, задана в виде таблицы значений $y_i = f(x_i)$ в узлах интерполяции $x_i = x_0 + ih$ на отрезке $[a, b]$. Требуется найти значения интеграла $\Im = \int_a^b f(x)dx$ на указанном отрезке.

По заданным значениям подынтегральной функции $y_i = f(x_i)$ построим интерполяционный полином Лагранжа

$$L_n = \sum_{i=0}^{n} y_i \cdot \prod_{i\neq j} \frac{x - x_i}{x_j - x_i},$$

который для равноотстоящих узлов примет вид

$$L_n = \sum_{i=0}^{n} y_i \cdot \frac{(-1)^{n-i}}{i!\cdot(n-i)!}\cdot\frac{q(q-1)\ldots(q-n)}{q-i},$$

где $q = (x - x_0) / h$.

Теперь заменим подынтегральную функцию $f(x)$ построенным полиномом, считая, что узлы интерполяции расположены равномерно:

$$\Im = \int_a^b f(x)dx = \int_a^b L_n dx =$$

$$= \int_a^b \sum_{i=0}^{n} y_i \cdot \frac{(-1)^{n-i}}{i!\cdot(n-i)!}\cdot\frac{q(q-1)\ldots(q-n)}{q-i}dx.$$

Проведя необходимые элементарные преобразования, выполнив замену переменных $dq = dx/n$ и сменив в соответствии с подстановкой пределы интегрирования, получим

$$\Im = \int_a^b f(x)dx = \sum_{i=0}^{n} y_i \cdot \frac{(-1)^{n-i}}{i!\cdot(n-i)!}\cdot h \times$$

$$\times \int_0^n q(q-1)\times\ldots\times(q-i+1)\cdot(q-i-1)\times\ldots\times(q-n)dq \quad .$$

Здесь h - шаг, который для равноотстоящих узлов интерполяции определяется как $h = (b - a)/n$. Подставив значения h в последнюю формулу, окончательно получим

$$\Im = \int_a^b f(x)dx = (b-a)\cdot\sum_{i=0}^{n} y_i \cdot H_i,$$

где

$$H_i = \frac{(-1)^{n-i}}{i!\,(n-i)!}\cdot\frac{1}{n}\cdot\int_0^n q(q-1)\times\ldots\times(q-i+1)\cdot(q-i-1)\times\ldots\times(q-n)dq;$$

H_i - *коэффициенты Ньютона - Котеса*. Они не зависят от значений функции $f(x)$ и являются функциями только количества узлов, на которые разбит отрезок $[a, b]$. Поэтому H_i обычно вычисляют заранее:

$N = 1$: $H_0 = H_1 = {}^1/_2$;
$N = 2$: $H_0 = H_2 = {}^1/_6$; $H_1 = {}^2/_3$;
$N = 3$: $H_0 = H_3 = {}^1/_8$; $H_1 = H_2 = {}^3/_8$;
$N = 4$: $H_0 = H_4 = {}^7/_{90}$; $H_1 = H_3 = {}^{16}/_{45}$; $H_2 = {}^2/_{13}$;

$N = 5$: $H_0 = H_5 = {}^{19}/_{288}$; $H_1 = H_4 = {}^{25}/_{96}$; $H_2 = H_3 = {}^{25}/_{144}$.

Этот список при необходимости можно продолжить. Но если теперь рассмотреть частные случаи формулы Ньютона - Котеса, то:

1) при $n = 1$ получаем формулу трапеций:

$$\Im = \int_a^b f(x)dx = (b-a)\cdot(y_0 + y_1)/2;$$

1) при $n = 2$ получаем формулу Симсона:

$$\Im = \int_a^b f(x)dx = (b-a)\cdot(y_0 + 4y_1 + y_2)/6;$$

2) при $n = 3$ получаем формулу "трех восьмых":

$$\Im = \int_a^b f(x)dx = 3\cdot(b-a)\cdot(y_0 + 3y_1 + 2y_2 + y_3)/8.$$

Погрешность последней формулы оценивается [Численные..., 1976] соотношением

$$R = -(3/80)\, h^5 y^{(IV)}(\xi) \text{ для всех } \xi \in [a, b] \setminus x_i .$$

Достроим таблицу коэффициентов до $n = 7$:

$N = 6$: $H_0 = H_6 = {}^{41}/_{840}$; $H_1 = H_5 = {}^{9}/_{35}$;
$H_2 = H_4 = {}^{9}/_{280}$, $H_5 = {}^{34}/_{105}$;

$N = 7$: $H_0 = H_7 = {}^{751}/_{17280}$; $H_1 = H_6 = {}^{3577}/_{17280}$;
$H_2 = H_5 = {}^{1323}/_{17280}$; $H_3 = H_4 = {}^{2989}/_{17280}$.

Если эти коэффициенты записать в специальный справочный массив, который может храниться на магнитном диске, то программа для вычисления определенного интеграла выглядит так:

```
FUNCTION NEWCOT (A,B:REAL;N:INTEGER) : REAL;
TYPE MS = ARRAY [0..7] OF INTEGER;
     MS2 = ARRAY [3..7] OF MS;
     FF = FILE OF MS2;
VAR H1,H2 : MS2; FC : TEXT;
     X1, SUM,H, Y1 : REAL;
BEGIN
  ASSIGN (FC,'A:\');
  RESET (FC);
  FOR I := 3 TO 7 DO
     FOR J := 0 TO 7 DO
        READ (FC,H1[I,J]);
  FOR I := 3 TO 7 DO
     FOR J := 0 TO 7 DO
        READ (FC,H2[I,J]);
  CLOSE (FC);
  H := (B-A) / N;
  X1 := A;
  SUM := 0.0;
  FOR I := 0 TO 7 DO
  BEGIN
     SUM := SUM + FUNC (X1) * H1[N,I]/H2[N,I];
     X1 := X1 + H;
  END;
  NEWCOT := (B-A) * SUM;
END.
```

В программе массивы $H1(i)$ и $H2(i)$ считываются с диска, где готовятся пользователем предварительно (имя файла надо доопределить). При этом $H1(i)$ содержит числители, а $H2(i)$ - знаменатели коэффициентов формул Ньютона - Котеса H_i, которые окончательно вычисляются в программе (для уменьшения ошибки вычислений); *FUNC* (*X*1) - процедура-функция, определяется пользователем по подынтегральному выражению.

Эту программу можно оформить как процедуру и использовать ее на каждом элементарном отрезке $[x_i, x_{i+1}]$, что значительно повысит точность вычисления определенного интеграла. Но тогда следует помнить, что n будет относиться не ко всему отрезку $[a, b]$, на котором вычисляется интеграл, а только к одному элементарному отрезку $[x_i, x_{i+1}]$.

Для проверки работы программы вычислялся определенный интеграл функции $\int_0^{\pi/2} \frac{\cos(x)}{1+x}dx$ на отрезке $[a, b]$, который пересчитывался параллельно для сравнения с помощью простейших формул интегрирования. Для вычислений по формулам Ньютона - Котеса $n = 3, 4, 5, 6, 7$; по остальным формулам значение n варьировалось от 10 до 100. Все вычисления выполнялись с точностью до 10^{-5}. Результаты работы приводятся в табл. 3.3.

Таблица 3.3

Кол-во узлов	Формула прямоугольников			Формула трапеций	Формулы Ньютона - Котеса	
	левых	правых	средних			
10	0.7533486992	0.5963973464	0.6729958921	0.6748730228	$N = 3$	0.6750341569
20	0.7131722957	0.6346966193	0.6734642885	0.6739344575	$N = 4$	0.6737460190
40	0.6933182921	0.6540804539	0.6735817747	0.6736993730	$N = 5$	0.6736947228
50	0.6893662861	0.6579760155	0.6735958836	0.6736711508	$N = 6$	0.6736281749
100	0.6814810848	0.6657859496	0.6736146990	0.6736335172	$N = 7$	0.6736255684

§ 5. ИНТЕГРИРОВАНИЕ С АВТОМАТИЧЕСКИМ ВЫБОРОМ КОЛИЧЕСТВА УЗЛОВ МЕТОДОМ РУНГЕ

Погрешность в вычислении определенных интегралов по приближенным формулам, описанным в п. 3.1 - 3.4, зависит от шага разбиения интервала интегрирования h и от гладкости интегрируемой функции $f(x)$, поэтому в общем случае заранее вычислить погрешность интегрирования невозможно. На практике для оценки погрешности пользуются удобным *правилом Рунге.* Опишем это правило более подробно.

Пусть некоторый интервал $[a, b]$ разбит равномерно на n отрезков $[x_i ; x_{i+1}]$, $i = 1, 2, ..., N$; $x_0 = a$; $x_N = b$. На каждом отрезке применим формулу трапеций:

$$J_i = \int_{x_{i-1}}^{x_i} f(x)dx \approx \frac{f_i + f_{i-1}}{2} \cdot h_i = \tilde{J}_{hi}.$$

Слева в последней формуле стоит *точное* значение определенного интеграла J_i, а справа - его *приближенное* значение $\tilde{J}_{hi}$. Найдем погрешность последней формулы с учетом оценок, полученных в § 2,

$$J_i - \tilde{J}_{hi} \approx \frac{1}{24} h_i^3 \cdot f''(y_i) = c_i \cdot h_i^3. \qquad (*)$$

Уменьшим теперь шаг разбиения вдвое и, рассуждая подобным образом, получим

$$J_i - \tilde{J}_{(h/2),i} \approx \frac{1}{24} (h_i / 2)^3 \cdot f''(y_i) = c_i \cdot (h_i / 2)^3. \quad (**)$$

Интегрируемая функция оставалась неизменной в том и другом случае, формула интегрирования была также одинаковой, следовательно, константа c_i в обоих случаях должна быть тоже одинаковой. Тогда, выражая $c_i\ h_i$ из формул (*) и (**) и выполнив элементарные преобразования полученного выражения, будем иметь

$$J_i - \tilde{J}_{hi} \approx 8\left(J_i - \tilde{J}_{(h/2),i}\right),$$

откуда, представив $8 = 2^3 + 1 - 1$, раскрыв скобки и приведя подобные, окончательно получим

$$J_i - \tilde{J}_{(h/2),i} = \frac{\tilde{J}_{(h/2),i} - \tilde{J}_{hi}}{2^3 - 1}. \qquad (3.7)$$

В выражении (3.7) в правой части стоят только известные величины - значения интегралов, вычисленные для шагов h_i и $h_i / 2$.

В общем случае, когда интегрируется функция некоторой квадратурной формулой порядка m, тогда формула (3.7) будет немного иной:

$$J_i - \tilde{J}_{(h/2),i} = \frac{\tilde{J}_{(h/2),i} - \tilde{J}_{hi}}{2^m - 1}. \qquad (3.8)$$

Используя формулу (3.7) или (3.8), в зависимости от типа формулы приближенного интегрирования (прямоугольников, трапеций, Симсона или Ньютона-Котеса) можно, задав необходимую точность вычислений, интегрировать с автоматическим выбором шага интегрирования. Процедуры, реализующие данный метод, можно взять из предыдущих § 2 - 4 и после некоторой переработки использовать для автоматического выбора шага интегрирования. На примере процедуры INTLPS покажем, как это можно выполнить. Во входные параметры указанной процедуры добавляем новую переменную - *eps,* задающую необходимую точность вычислений. Текст переработанной процедуры приводится ниже.

```
PROCEDURE INTLPS (VAR S:REAL; K:INTEGER;
      EPS:REAL);
BEGIN
  S := 0.0;  N := 1;
  REPEAT
     S1 := S;   H := (B-A) / N;
     X := A;
     IF K<5 THEN
        FOR I := 0 TO N DO
        BEGIN
           IF K <>3 THEN X := A+ H*I
           ELSE  X:=A+H/2+H*I;
           F := FUNC (X);
           IF K=4 THEN  F := F/2.0;
           IF (I=0) AND (K<>2) THEN S := S + F;
           IF (I=N) AND ((K=2) OR (K=4)) THEN
              S := S + F;
           IF (I<>0) AND (I<>N) THEN
              IF K<>4 THEN S := S+F
              ELSE  S := S + F*2;
        END
     ELSE
        BEGIN
           FOR I := 1 TO N-1 DO
              S := 2*FUNC(X+H*I)+4*FUNC(X+H*I-H/2)+ S;
           S := S + 4*FUNC(B-H/2) + FUNC(A) + FUNC(B);
           S := S / 6;
        END;
     S := S * H;     N := N * 2;
  UNTIL ABS (S1 - S) > EPS;
END.
```

Для контроля и проверки процедуры использовалась функция, заданная в § 2, значения которой вычислялись методом Симсона. Результаты получились следующие: при заданной точности 0,00001 потребовалось только одно разбиение отрезка, и значение определенного интеграла при $n = 2$ было равно 0.0968537329.

§ 6. ИНТЕГРИРОВАНИЕ С АВТОМАТИЧЕСКИМ ВЫБОРОМ КОЛИЧЕСТВА УЗЛОВ

В ряде задач, например при численном решении эволюционных интегродифференциальных уравнений высокого порядка, когда вычисление определенного интеграла должно выполняться на сетке с изменяющимся от шага к шагу количеством узлов квадратурной формулы (переменные находятся "внутри" функционирующего алгоритма, применяется один или несколько пределов интегрирования), а порядок аппроксимации должен при этом соответствовать порядку разностной схемы для дифференциальной части уравнения, возникает весьма нетривиальная проблема автоматического выбора количества узлов квадратурной формулы, имеющей достаточно высокую точность [Белашов, 1989, 1991а, б, 1997]. Рассмотрим в качестве примера, каким образом могут в этом случае использоваться такие известные квадратурные формулы, как Симпсона и Ньютона - Котеса с количеством узлов $m>3$.

Пусть, например, уравнение имеет вид

$$\partial_t u + A(t,u)u = f, \qquad (3.9)$$

где $A(t,u)$ - некоторый дифференциальный оператор; f - интегральный член, и пусть порядок аппроксимации дифференциальной части уравнения (для простоты будем считать его двумерным по пространственным координатам) $O\left(\tau^2, h_{x,y}^2\right)$. В этом случае для аппроксимации его интегральной части f достаточно выбрать квадратурную формулу Симпсона

$$f_{ij} = \frac{h_x}{3}\sum_{i=1}^{m}\left(f'_{2i-1,j} + 4f'_{2i,j} + f'_{2i+1,j}\right); \qquad (3.10)$$

$$m = (N-1)/2,$$

(f'_{ij} - аппроксимация соответствующей подынтегральной функции, N - нечетное), которая согласно оценкам, полученным Белашовым [1989, 1991б], аппроксимирует интеграл на решениях уравнения (3.9) с точностью $O\left(h_{x,y}^2\right)$ для шагов сетки $h_x, h_y \le 0.075$ (зависит от постоянных коэффициентов уравнения). Меньшие требования к размеру шага и расположению узлов пространственной сетки имеют аппроксимации интеграла квадратурными формулами Ньютона - Котеса с количеством узлов $m>3$:

$$f_{ij} = \sum_{k=1}^{n}\sum_{i=1}^{m} A_i^{(m)} f'_{l\,j}, \quad l = i + (m-1)(k-1); \qquad (3.11)$$

$$A_i^{(m)} = (m-1)\,h_x C_i^{(m)},$$

где $C_i^{(m)}$ - коэффициенты квадратурной формулы. При этом следует помнить, что m должно выбираться таким, чтобы выражение $(N-1)/(m-1)$ было целым. Аппроксимация (3.11) позволяет вычислять интеграл f с точностью $O(h_{x,y}^4)$ уже для $h_x, h_y \le 0.1$ [Белашов, 1989].

Ограничение на шаг по времени, которое дают формулы (3.7), (3.10), при учете двумерности задачи весьма существенно. Кроме того, на каждом временном слое требуется запоминать значения функции на двух предыдущих слоях. Поэтому такие сравнительно простые схемы, как (3.2) и (3.8), вряд ли целесообразно использовать для получения решения в асимптотике $t \to \infty$. Однако они могут оказаться полезными для моделирования динамики становления решения на начальной стадии эволюции.

Ниже приводится подпрограмма FNK, использующая функцию KU, которая осуществляет автоматический выбор квадратурной формулы из семейства формул Ньютона-Котеса в зависимости от количества узлов сетки, и вычисляющая соответствующий интеграл.

Формальные параметры процедур. *Входные: p* - одномерный (***real***) массив значений подынтегральной функции размерностью *N*; *h* (тип ***real***) - шаг сетки. *Выходные:* функция *fnk* возвращает вычисленное значение интеграла.

```
FUNCTION FNK ( P : ARRAY OF REAL;
        K, N : INTEGER; H: REAL) : REAL;
VAR C,C8 : ARRAY [1..8] OF REAL;
        C2 : ARRAY [1..2] OF REAL;
        C3 : ARRAY [1..3] OF REAL;
        C4 : ARRAY [1..4 OF REAL;
        C5 : ARRAY [1..5] OF REAL;
        C6 : ARRAY [1..6] OF REAL;
        C7 : ARRAY [1..7] OF REAL;
        NA, I, J, NJ, N1, D, NZ, JJJ, III : INTEGER;
        F : REAL;

 FUNCTION KU ( VAR N : INTEGER) : INTEGER;
 VAR N1, J, I, ND : INTEGER;
 BEGIN
    N1:=N-1;
   FOR I := 1 TO 7 DO
    BEGIN
       J := 8 - I;
       ND := MOD (N1, J);
       IF ND := 0 THEN
       BEGIN
          KU := J + 1;
          EXIT;
       END;
    END;
    KU := J + 1;
 END;

BEGIN   C2 [1] := 0.5;
  C2 [2] := 0.5;
  C3 [1] :=0.1666667;
  C3 [2] :=0.6666667;
```

```
C3 [3] :=0.1666667;
C4 [1] := 0.125;    C4 [2] := 0.375;
C4 [3] := 0.375;    C4 [4] := 0.125;
C5 [1] := 0.0777778;
C5 [2] := 0.3555556;
C5 [3] := 0.1333333;
C5 [4] := 0.3555556;
C5 [5] := 0.0777778;
C6 [1] := 0.0659722;
C6 [2] := 0.2604167;
C6 [3] := 0.1736111;
C6 [4] := 0.1736111;
C6 [5] := 0.2604167;
C6 [6] := 0.0659722;
C7 [1] := 0.0488095;
C7 [2] := 0.2571429;
C7 [3] := 0.0321429;
C7 [4] := 0.3238095;
C7 [5] := 0.0321429;
C7 [6] := 0.2571429;
C7 [7] := 0.0488095;
C8 [1] := 0.0434606;
C8 [2] := 0.2070023;
C8 [3] := 0.0765625;
C8 [4] := 0.1729745;
C8 [5] := 0.1729745;
C8 [6] := 0.0765625;
C8 [7] := 0.2070023;
C8 [8] := 0.0434606;
NA := N - K + 1;
IF (N - K) <= 0 THEN
BEGIN
    F := 0.0;
    FNK := F;
    EXIT;
END;
NJ := KU (NA);
CASE NJ OF
    1 : BEGIN
            F := 0.0;    FNK := F;
            EXIT;
        END;
    2 : FOR I := 1 TO NJ
            C [I] := C2 [I];
    3 : FOR I := 1 TO NJ
            C [I] := C3 [I];
    4 : FOR I := 1 TO NJ
            C [I] := C4 [I];
    5 : FOR I := 1 TO NJ
            C [I] := C5 [I];
    6 : FOR I := 1 TO NJ
            C [I] := C6 [I];
    7 : FOR I := 1 TO NJ
            C [I] := C7 [I];
    8 : FOR I := 1 TO NJ
            C [I] := C8 [I];
END;  { **** CASE **** }
N1 := NJ - 1;
D   := N1*H;
NZ := NA DIV N1;
IF (NJ - 2) < = 0 THEN NZ := NZ - 1;
F   := 0.0;
FOR J := 1 TO NZ DO
BEGIN
    JJJ := N1*(J - 1) + K - 1;
    FOR  I := 1 TO NJ DO
    BEGIN
        III := I + JJJ;
        F := F + C[I]*D*P[III];
    END;
END;
FNK := F;
END.
```

Подпрограмма FNK тестировалась на ЭВМ *Pentium*-166 для массивов от 21 до 5001 членов. Результаты показали, что подпрограмма обеспечивает заданную точность вычислений при разумных значениях шага сетки h, кроме того, она эффективно использовалась в системе моделирования динамики нелинейных волн в плазме, описываемых уравнениями типа уравнения Кадомцева - Петвиашвили, представленной в серии работ В.Ю.Белашова [Белашов, 1989, 1991а, б, 1997].

§ 7. ВЫЧИСЛЕНИЕ ИНТЕГРАЛА ПО РОМБЕРГУ

При решении проблемы построения оптимальной квадратуры часто используют различные методы уточнения приближений к интегралу, полученные при помощи простейших квадратурных формул. Одна из наиболее распространенных формул, реализующих такой подход, - это формула Ромберга.

Алгоритм Ромберга детально описан в работе [Romberg, 1955] и является весьма распространенным в западных странах (в частности, он один из наиболее рекомендуемых в библиотеках стандартных программ фирмы IBM). Суть алгоритма состоит в следующем. Пусть с помощью некоторой квадратурной формулы были вычислены приближенные значения интеграла $S_{M_k}(f)$ при $M_k = M_0\, 2^k$ отрезках разбиения интервала интегрирования $[a, b]$, но при этом требуемая точность не была достигнута. Тогда в соответствии с правилом Ромберга по полученной совокупности значений $S^{(1)}_{M_0}(f),\ldots,S^{(1)}_{M_p}(f)$ можно применить при каждом k из интервала $[0, p]$ для вычисления последовательности приближений

$$S^{(2)}_{M_k}(f),\ldots S^{(k+1)}_{M_k}(f)$$

рекуррентную формулу

$$S^{(i)}_{M_k}(f) = S^{(i-1)}_{M_k}(f) + \frac{1}{2^i - 1}\cdot\left(S^{(i-1)}_{M_k}(f) - S^{(i-1)}_{M_{k-1}}(f)\right), \quad (3.12)$$

При этом порядок погрешности будет определяться выражением $2*k + 2$.

Начальные приближения $S^{(1)}_{M_k}(f)$ при некотором задаваемом начальном M_0 могут быть получены, например, по формуле трапеций, которую целесообразно в данном случае использовать в следующем виде:

$$S^{(1)}_{M_k}(f) = \frac{1}{2} S^{(1)}_{M_{k-1}}(f) + \frac{b-a}{M_k} \sum_{j=1}^{M_{k-1}} f\left(a + \frac{2j-1}{M_k}(b-a)\right) . \quad (3.13)$$

Достижение заданной точности ε вычисления интеграла может контролироваться проверкой выполнения неравенства

$$\left| \left(S^{(i)}_{M_k}(f) - S^{(i-1)}_{M_k}(f) \right)(b-a) \right| < \varepsilon ,$$

при выполнении которого итерационный процесс (3.11) прекращается.

Алгоритмы (3.12), (3.13) реализованы в процедуре-функции INTROMB для начального числа отрезков разбиения $M_0 = 2$.

Формальные параметры процедуры. *Входные*: *a, b* (тип ***real***) - нижний и верхний пределы интегрирования; *k* (тип ***integer***) - количество вычисленных по формуле трапеций приближений интеграла для разбиений M_0, ..., M_k; *f* (тип ***real function***) - процедура-функция, по которой вычисляют значения подынтегральной функции $f(x)$ в узлах разбиения $a = x_0, x_1, ..., x_{Mk} = b$, $0 < M_1 < M_k$; *e* (тип ***real***) - задаваемая точность. *Выходные*: *intromb* (тип ***real***) - значение интеграла, вычисленное с точностью ε.

```
FUNCTION  INTROMB (A,B: REAL;K:INTEGER;
            EXTENAL F : FUNCTION; EPS: REAL) : REAL;
VAR D,S,H : REAL; I,J,M,N : INTEGER;
            T : ARRAY [1..K+1] OF REAL;
BEGIN    D:=B-A;
  T[1]:=(F(A)+F(B))/2; N:=1;
  FOR I:=1 TO K DO
  BEGIN
     S:=0;      N:=2*N;      H:=D/N;
     FOR J:=1 TO N DO  S:=S+F(A+J*H);
     T[I+1]:=(2*S/N+T[I])/2;
     M:=1;
     FOR J:=I DOWNTO 1 DO
     BEGIN
        M:=4*M;
        T[J]:=T[J+1]+(T[J+1]-T[J])/(M-1)
     END {J};
     IF ABS((T[1]-T[2])*D) < EPS THEN
        EXIT;
  END { **** I **** };
  INTROMB:=T[1]*D
END { ***** INT ROMB **** }.
```

Процедура-функция INTROMB была получена путем перевода на язык *PASCAL* процедуры-функции, опубликованной в работах Агеева и др. (1966; 1976) и являющейся, в свою очередь, результатом оптимизации и ординарной переработки алгоритма Ф.Л.Бауэра Х. Татчером (1962). Тестирование производилось на машине IBM PC/AT-386 для ряда подынтегральных функций (в том числе и для тех же $f(x)$) и пределов интегрирования, что и в работах Агеева и др. (1966; 1976), а также Хенниона (1962). При этом для $\varepsilon = 10^{-10}$ были получены следующие результаты:

1) $\mathfrak{I} = \int_0^{\pi} \cos(x)dx = 1.408858\ e\text{-}7 \pm 9.47\ e\text{-}11$ ($k = 4$);

2) $\mathfrak{I} = \int_2^5 \frac{x^3 dx}{\left(4 + x^2\right)^3} = 3.082269\ e\text{-}2 \pm 0.0$ (k=4);

3) $\mathfrak{I} = \int_{0.01}^{1.1} x^{-5} dx = 2.500138\ e +7 \pm 0.0$ (k=10);

4) $\mathfrak{I} = \int_1^2 \ln(x)dx = 0.3862944 \pm 0.0$ (k=4).

Замечание. Х. Татчером в работе (1964) указывалось, что использовавшиеся с успехом многими исследователями алгоритмы (3.7), (3.8) все же не вполне совершенны, так как существует значительный класс подынтегральных функций, для которых экстраполяционные значения (3.1) являются менее точными оценками интеграла, чем соответствующие суммы формулы трапеций. Точность процедуры Ромберга, вообще говоря, зависит от возможности разложения погрешности метода трапеций по степеням h^r, например в ряд Эйлера - Маклорена. Коэффициенты при h^{2r} такого ряда пропорциональны разности производных на концах интервала интегрирования $f^{(2r+1)}(a) - f^{(2r+1)}(b)$. Следовательно, любой интеграл, для которого эта разность равна нулю, не сходится при экстраполяции по Ромбергу. Простыми примерами таких функций являются интегралы периодических функций при $(b - a) >> T$, где T-период функции, и интегралы, для которых производные равны нулю на обоих концах интервала, например интегральная ап7проксимация модифицированной функции Ханкеля:

$$e^x K_p(x) = \int_0^L e^{x(1-\cosh(t))} \cdot \cosh(pt)dt ,$$

где L выбирается настолько большим, что погрешность интеграла от L до ∞ можно считать пренебрежимо малой. Алгоритм непригоден также в тех случаях, когда разложение остаточного члена содержит другие степени h или хотя бы одну такую степень. И, наконец, тривиальным с точки зрения вычислительной математики вляется исключение, когда подынтегральная функция $f(x)$ разрывна.

§ 8. ИНТЕРПОЛЯЦИЯ, ДИФФЕРЕНЦИРОВАНИЕ И ИНТЕГРИРОВАНИЕ ФУНКЦИЙ В ОДНОЙ ПРОЦЕДУРЕ

Поскольку в вычислительной математике подход к решению задач дифференцирования и интегрирования функций основывается на построении интерполирующих многочленов, зачастую бывает целесообразным объединить эти задачи в одной процедуре. Например, при использовании интерполяционной схемы Лагранжа, приспособленной для осредненных квадратных трехчленов $(a_k x^2 + b_k x + c_k)$, где

$$a_k = \sum_{k=1}^{m} t_i\,,\quad b_k = a_k \cdot \sum_{\substack{j=1\\(j\neq i)}}^{m} x_j\,,\quad c_k = a_k \cdot \prod_{\substack{j=1\\(j\neq i)}}^{m} x_j\,,$$

$$t_j = y_j \,/ \prod_{j=1(j\neq i)}^{m} (x_j - x_i),\, y_j = f\left(x_j\right), j = 1,2,\ldots,m\,,$$

производная может быть вычислена при $m = 3$, $k = 1, 2$ для функции, определенной в точках 1, 2, 3, 4 путем усреднения коэффициентов парабол, проведенных через точки 1, 2, 3 и 2, 3, 4:

$$y'(x_2) = 2ax + b, \quad a = (a_1 + a_2)/2\,,$$

$$b = (b_1 + b_2)/2, \quad c = (c_1 + c_2)/2\,.$$

Интерполирующий многочлен, построенный по двум параболам, в этом случае будет вычисляться по формуле $F_2(x) = (ax^2 + bx + c)$. Интегрирование производится путем осреднения интегралов от парабол между границами независимых координат, т.е. для каждого интервала вдоль абсциссы, поскольку полученные результаты накапливаются для вычисления определенного интеграла.

Рассмотренный алгоритм реализован в процедуре DIFINT, которая в зависимости от способа обращения к ней выполняет интерполяцию, дифференцирование или интегрирование функции одной переменной.

Формальные параметры процедуры. *Входные*: $x[1{:}n]$, $y[1{:}n]$ (тип ***real***) - массивы табличных значений аргумента и функции, причем значения x_i расположены в возрастающем порядке и $x_i \neq xj$ для $i \neq j$; jt (тип ***integer***) - параметр, задающий операцию: при $jt = 1$ выполняется интерполирование, при $jt = 2$ и $jt = 3$ соответственно дифференцирование и интегрирование; xk (тип ***real***)- точка, для которой строится полином или вычисляется производная, при $xk \in [x_1, x_{n-1}]$ осуществляется операция экстраполирования, при $jt = 3$ значение xk несущественно; $x1$, $x2$ (тип ***real***) - нижний и верхний пределы интегрирования, в случае $jt = 1, 2$ значения $x1$, $x2$ несущественны. *Выходные*: *res* (тип ***real***) - значение результата.

```
PROCEDURE DIFINT (X,Y:MAS1;N,JT:INTEGER;
                  XK,X1,X2:REAL; VAR RES : REAL);
VAR CA,CB,CC,A,B,C,S1,S2,T1,T2,T3,SUM : REAL;
                  J,JS,J1,J2,I : INTEGER;
LABEL ST,INT,TERM,NO5,NO6,NO9,INTRP,DIFF,
                  NO17, LAGR;
BEGIN
ST:
  IF JT=3 THEN GOTO INT;
  IF XK > X[N-1] THEN
  BEGIN
     J:=N-1;
     JS:=1;
     GOTO TERM
  END;
  IF XK < X[2] THEN
  BEGIN
     J:=2;
     JS:=1;
     GOTO TERM
  END;
     {***** □□□□□□□□□□□ □□□□□ □□□□□□□□□ □ □□□□□□□ *****};
  JS:=2;
  FOR I:=2 TO N DO
  BEGIN
     IF XK < X[I] THEN GOTO TERM;
     J:=I;
  END;
NO5:
  CA:=A;
  CB:=B;
  CC:=C;
  JS:=33;
  J:=J+1;
  GOTO TERM;
NO6:
   A:=(CA+A)/2;
  B:=(CB+B)/2;
  C:=(CC+C)/2;
NO9:
  IF JT=2 THEN GOTO DIFF;
INTRP:
  RES:=XK*(A*XK+B)+C;
  RETURN;
DIFF:
  RES:=2*A*XK+B;
  RETURN;
INT:  SUM:=0;
  S1:=X1;
  J1:=2;     J2:=N;
  FOR I:=1 TO N DO
  BEGIN
       IF X1 < X[I] THEN GOTO NO17;
```

```
      J1:=J1+1
    END;
NO17:
    FOR I:=1 TO N DO
    BEGIN
      J2:=J2-1;
      IF X2 > X[J2+1] THEN
      GOTO LAGR
    END {I};
TERM:
    J1:= J;
    J2:=J;
LAGR:
    FOR J:=J1 TO J1+1 DO
    BEGIN
      A:=X[J-1]-X[J];
      B:=X[J-1]-X[J+1];
      C:=A-B;
      T1:=Y[J-1]/(A*B);
      T2:=Y[J]/(A*C);
      T3:=-Y[J+1]/(B*C);
      A:=T1+T2+T3;
      B:=-(X[J]+X[J+1])*T1-(X[J-1]+X[J+1])*T2-
           (X[J-1]+X[J])*T3;
      C:=X[J]*X[J+1]*T1+X[J-1]*X[J+1]*T2+X[J-1]*X[J]*T3;
      IF JT <> 3 THEN
      CASE OF JS
        1: GOTO NO9;
        2: GOTO NO5;
        3: GOTO NO6;
      END; {CASE}
      IF J=J1 THEN
      BEGIN
        CA:=A;
        CB:=B;
        CC:=C;
      END
      ELSE
      BEGIN          CA:=(A+CA)/2;
        CB:=(B+CB)/2;
        CC:=(C+CC)/2
      END;
      S2:=X[J];
      SUM:=SUM+CA*(SQR(S2)*S2-SQR(S1)*S1)/3+
           CB*(SQR(S2)-SQR(S1))/2+CC*(S2-S1);
      CA:=A;
      CB:=B;
      CC:=C;
      S1:=S2
    END {J};
    RES:=SUM+CA*(SQR(X2)*X2-
         SQR(S1)*S1)/3+CB*(SQR(X2)-
    SQR(S1))/2+CC*(X2-S1);
END {DIFINT};
```

Процедура DIFINT была получена из одноименного алгоритма Агеева (1976) путем его ординарной переработки и перевода с языка *ALGOL* на язык *PASCAL* с учетом всех модификаций первоначального варианта Р.Е.Хеньона (1962). Тестирование проводилось на IBM PC/AT-286 для параметров $x_1 = 0$, $x_n = 2$, $x_i = x_1 + (i\text{-}1)(x_n - x_1) / n$ при $n = 10$, 100, 200; $y_i = \exp(x_i)$, $i = 1, 2, ...,$ $n + 1$; $xk = 1.57$. Результаты приведены в табл. 3.4.

Контрольные значения для вычисления производной функции $y = \exp(xk)$ получены на ЭВМ, а интеграл вычислен по формуле Ньютона - Лейбница

$$\int_0^2 e^x dx = e^2 - 1.$$

Таблица 3.4

Результаты расчетов			
n	$jt = 1$	$jt = 2$	$jt = 3$
10	4.8068337	4.8055027	6.3859727
100	4.8066484	4.8067283	6.3890554
200	4.8066479	4.8066064	6.3890563
Контр. знач.	4.8066481	4.8066481	6.33890561

§ 9. ВЫЧИСЛЕНИЕ КОМПЛЕКСНОГО КРИВОЛИНЕЙНОГО ИНТЕГРАЛА

Для вычисления комплексного криволинейного интеграла обычно используется конечная сумма Римана - Стилтьеса [Лаврентьев, 1965]:

$$S = S_1 + iS_2 = \int_a^b f(z)z'(t)dt \approx \sum_{i=1}^{n} f(\xi)\cdot(z_i - z_{i-1}),$$

где $a \le i \le b$, $\xi \in [z_{i-1}, z_i]$. Этот алгоритм реализован в процедуре CINTEG, в которой использованы процедуры ZET и FZET, выполняющие вычисления функции $z(t)$ соответственно на кривой G и $f(\xi)$.

Формальные параметры процедуры. *Входные*: *a, b* (тип ***real***) - границы интервала интегрирования ($a < b$); *n* (тип ***integer***) - количество частичных интервалов, на которые разбивается [*a, b*]. *Выходные*: *s*1, *s*2 (тип ***real***) - действительная S_i и мнимая S_{i-1} части интеграла *S*.

```
PROCEDURE CINTEG (A,B: REAL;N:INTEGER;
                  VAR S1,S2: REAL);
VAR D,ZT11,ZT12,DZ1,DZ2,P1,P2: REAL; I : INTEGER;
                  ZT,ZK,FZ ARRAY [1..2] OF REAL;
BEGIN
  S1:= 0.0;
  S2:=0.0;
  D:=(B-A)/N;
  T:=A;
```

```
  WHILE TRUE DO
  BEGIN
     ZET(T,ZT);
     IF T<>A THEN
     BEGIN
        DZ1:=ZT[1]-ZT11;
        DZ2:=ZT[2]-ZT12;
        ZK[1]:=ZT11+DZ1/2;
        ZK[2]:=ZT12+DZ2/2;
        FZET(ZK,FZ);
        P1:=FZ[1]*DZ1-FZ[2]*DZ2;
        P2:=FZ[1]*DZ2-FZ[2]*DZ1;
        S1:=S1+P1;
        S2:=S2+P2;
        IF T >= B-0.25*D THEN
           RETURN;
     END;
     ZT11:=ZT[1];
     ZT12:=ZT[2];
     T:=T+D;
  END;
END { **** CINTEG **** }.
```

Процедура CINTEG была получена путем переработки (с целью сокращение записи) и перевода на язык *PASCAL* алгоритма COMPLINT, опубликованного в справочнике [Агеев и др., 1976] и являющегося результатом исправления и оптимизации алгоритма, разработанного Дж. Пфальцем (1962). Процедура была протестирована на нескольких задачах, аналогичных рассмотренным в работах [Агеев и др., 1976; Pfaltz, 1962] на машине IBM PC/AT-286. Примеры результатов вычислений приведены ниже.

Пример 1. Вычисление интеграла $\int z^{-1/2} dz$ по полуокружности $| z | = 1$, $y > 0$. Процедуры ZET и FZET имели вид:

```
PROCEDURE ZET(T:REAL;Z:MAS1);
BEGIN
  Z[1]:=COS(T);
  Z[2]:=SIN(T)
END;

PROCEDURE FZET(Z,F: MAS1);
BEGIN
  F[1]:=SQRT(0.5*(1+Z[1]));
  F[2]:=-SQRT(0.5*(1-Z[1]))
END;
```

при $t \in [0,\ p]$.

Формальные параметры процедур. *Входные*: $a = 0$, $b = 3.14159265$, $n = 10, 50, 100, 500$. Результаты интегрирования приведены в табл. 3.5.

Пример 2. Выполнялось вычисление интегралов $\Im = \int_1^0 x dx$ и $\Im = \int_2^1 y dz$ по радиусу-вектору комплексной точки $z = 2 + i$. Тела процедур ZET и FZET задавались для вычисления I_1 в виде

```
BEGIN
  Z[1]:=2*T;
  Z[2]:=T
END;


BEGIN
  F[1]:=Z[1];
  F[2]:=0
END;
```

- для вычисления I_2 в виде

```
BEGIN
  F[1]:=Z[2];
  F[2]:=0
END.
```

Входные параметры: $a = 0$, $b = 1$. Как и в примере, приведенном в работе [Агеев др., 1976], уже при $n = 1$ были получены точные результаты: $I_1 = 2 + i$, $I_2 = 1 + i/2$.

Т а б л и ц а 3.5

n	$S = S + iS$
10	-1.980219 + i81.980219
50	-1.998826 + i81.998825
100	-1.999665 + i81.999664
500	-1.999995 + i81.999981

4.

При решении многих физических и геометрических задач приходится искать неизвестную функцию по данному соотношению между неизвестной функцией, ее производными и независимыми переменными. Такое соотношение называется *дифференциальным уравнением*, а отыскание функции, удовлетворяющей уравнению, называется *решением* или *интегрированием* данного уравнения.

Простейшее дифференциальное уравнение первого порядка $y' = f(y, t)$ имеет семейство решений $y = y(t)$, являющихся интегральными кривыми второго порядка, чаще всего вида $y = Ce^t$, с произвольной постоянной C. Тогда выбор начального значения $y(0)=y_0$ определяет одну из кривых семейства решений, которая и будет считаться решением поставленной задачи.

Основной считается *задача Коши* в разделе высшей математики "решение дифференциальных уравнений приближенными методами", формулирующаяся традиционно так: *требуется найти решение уравнения* $y' = f(y, t)$ *в виде* $y = y(t)$, *удовлетворяющее начальному условию* $y(0) = y_0$. Иными словами, *требуется найти среди семейства интегральных кривых, являющихся частными решениями дифференциального уравнения* $y' = f(y, t)$, *интегральную кривую* $y(t)$, *которая проходит через заданную точку* $M(t_0, y_0)$.

Для двумерного случая задача Коши формулируется так: *если* $f(x,y)$ *непрерывна в некоторой области* **R**, *определенной как* $|x - x_0| < a$; $|y - y_0| < b$, *то существует, по крайней мере, одно решение* $y = y(x, t)$, *определенное в окрестности* $|x - x_0| < h$, *где* $h > 0$, *и это решение будет единственным, если выполняется условие Липшица*

$$|f(x,\psi) - f(x,y)| < N \cdot |\psi - y|,$$

где N - некоторая константа Липшица, зависящая в общем случае от a и b.

Заметим, что решение системы дифференциальных уравнений первого порядка

$$\begin{cases} y' = f(x, y, t) \\ x' = g(x, y, t) \end{cases};$$

при условии, что производные *df/dy, df/dx, dg/dy* и *dg/dx* существуют на каждом интервале интегрирования, содержит уже две постоянные интегрирования и, следовательно, нужны два начальных условия, чтобы однозначно определить эти константы.

Как правило, аналитическими методами можно решить дифференциальные уравнения только определенного вида. Если уравнение не сводится никакими тождественными преобразованиями ни к одному из известных типов дифференциальных уравнений, то решение такой задачи может быть найдено только специальными численными методами. Рассмотрим наиболее часто применяемые на практике.

§ 1. ПРИБЛИЖЕННОЕ РЕШЕНИЕ ОБЫКНОВЕННОГО ДИФФЕРЕНЦИАЛЬНОГО УРАВНЕНИЯ МЕТОДОМ ЭЙЛЕРА

При обсуждении методов решения задачи Коши ограничимся рассмотрением дифференциального уравнения первого порядка $y' = f(x,y)$, удовлетворяющего начальному условию $y(x_0) = y_0$.

Численное решение задачи состоит в построении таблицы приближенных значений $y_1, y_2, ..., y_n$, являющихся решениями уравнения $y = y(x)$ в точках $x_1, x_2, ..., x_n$. Чаще всего точки x_i расположены равномерно: $x_i = x_0 + h \cdot i$, где $i = 1, 2, ..., n$. Точки x_i называют *узлами сетки, h - шагом сетки*. Ясно, что $h > 0$.

Для получения решения задачи Коши воспользуемся разложением функции в ряд Тейлора. Для этого продифференцируем исходное уравнение $y' = f(x,y)$ по x n раз. Тогда с учетом правила дифференцирования сложной функции получим следующие соотношения:

$$y'' = f_x(x, y) + f_x(x, y) \cdot y';$$
$$y''' = f_{xx}(x,y) + 2f_{xy}(x,y) \cdot y' + f_{yy}(x,y) \cdot (y') + f_y(x,y) \cdot y''; \ \ldots$$

Полагая $x = x_0$ и $y = y_0$, получаем ряд $y'(x_0)$; $y''(x_0)$; $y'''(x_0)$; ... ; $y^{(n)}(x_0)$, который сходится к решению поставленной задачи, т.е.

$$y(x) \cong \sum_{i=0}^{n} y^{(i)}(x_0) \cdot \frac{(x - x_0)^i}{i!}. \qquad (4.1)$$

Надо заметить, что если $|x - x_0|$ больше радиуса сходимости ряда $y'(x_0)$; $y''(x_0)$; $y'''(x_0)$; ...; $y^{(n)}(x_0)$, то погрешность $|\psi - y|$ не стремится к нулю при $n \to \infty$, т.е. ряд расходится. Тогда поступают следующим образом. Отрезок $[x_0, x_0+X]$, на котором ряд расходится, разбивают на меньшие отрезки $[x_i, x_{i+1}]$, где $i = 1, 2, ..., n$, и получают новые последовательные приближения y_j к решению $y(x_j)$ при $j = 1, 2, ..., m$, используя следующую схему:

1) y_i считаем найденным (для первого шага им может быть y_0);

2) вычисляем в точке x_i производные $y_i^{(k)}$;

3) полагаем $y_x \cong z_i(x) \cong \sum_{k=0}^{n} y^{(k)}(x_0) \cdot \frac{(x-x_0)^k}{k!}$;

4) считаем $y_{i+1} = z_i(x_{i+1})$;

5) повторяем с первого пункта.

Если же в формуле (4.1) положить n=1, то получим основную расчетную формулу метода Эйлера: $y_{i+1}=y_i+h\times$ $\times f(x,y)$.

Этот метод относится к группе одношаговых, в которых для расчета точки $(x_{i+1},\ y_{i+1})$ требуется информация только о последней вычисленной точке $(x_i,\ y_i)$. Он допускает простую геометрическую интерпретацию [Демидович и др., 1962]. Получается, что на каждом новом приближении решение переходит на другую кривую из семейства решений. Этот факт для некоторых дифференциальных уравнений может привести к большим ошибкам и решения задачи Коши начнет расходиться. Заметим, что хотя метод Эйлера относится к итерационным, но ошибки, сделанные на ранних этапах, не уменьшаются из-за неустойчивости вычислительной схемы и ее чувствительности к шагу разбиения отрезка, на котором ищется решение. Чтобы обойти эту трудность, применяют другие вычислительные схемы или методы.

Для оценки погрешности метода Эйлера на одном из шагов сетки разложим точное решение в ряд Тейлора в окрестности узла x_i:

$$y(x_{i+1}) = y(x_{i+h}) = y(x_i) + y'(x_i)\cdot h + \Theta(h^2) =$$

$$= y(x_i) + f(x_i,y_i)\cdot h + \Theta(h^2).$$

Сравнение разложения с основной расчетной формулой метода показывает, что они совпадают до членов первого порядка по h, а погрешность формулы равна $\Theta(h^2)$, т.е. метод Эйлера относится к методам первого порядка.

Программа, реализующая метод Эйлера, может быть представлена простой процедурой-функцией [Плис, Сливина, 1983] (перевод на язык *PASCAL* выполнен авторами):

```
FUNCTION EYLER (X, H, Y : REAL) : REAL;
BEGIN    EYLER := Y + H*FUNC(X,Y) END.
```

Формальные параметры процедуры. *Входные*: x (тип ***real***) - очередное значение x_i; h (тип ***real***) - величина шага разбиения отрезка, на котором ищется решение; y (тип ***real***) - предыдущее значение y_i; *FUNC*(x) (тип ***real***) - процедура-функция, вычисляющая значение правой части уравнения по заданным x и y. Результатом работы данной процедуры-функции явится очередное значение y_{i+1}.

Для проверки и тестирования процедуры решалась задача Коши методом Эйлера на отрезке [0.2; 1.2] с шагом 0.1 при начальном условии $y(0.2) = 0.25$. Вычисления выполнялись с точностью 10^{-4}:

$$y' = 0.185(x^2 + \cos(0.7x) + 1.843y.$$

Составим процедуру-функцию для вычисления правой части уравнения:

```
FUNCTION FUNC (X, Y : REAL) : REAL;
BEGIN    FUNC := 1.843*Y + 0.185*(X*X + COS(0.7*X));
END.
```

Результаты работы программы приведены в табл. 4.1.

Таблица 4.1

i	x_i	y_i
1	0.200000	0.250000
2	0.300000	0.315134
3	0.400000	0.392972
4	0.500000	0.486136
5	0.600000	0.597734
6	0.700000	0.731449
7	0.800000	0.891643
8	0.900000	1.083487
9	1.000000	1.313107
10	1.100000	1.587762
11	1.200000	1.916053

§ 2.

Метод Эйлера (§ 1) относится к методам первого порядка, потому что обладает не только неустойчивостью решения, но и низкой точностью. Однако в смысле погрешности он дает удовлетворительные результаты даже при не очень больших h. К сожалению, ошибки при вычислении накапливаются от шага к шагу, и к концу отрезка решение содержит значительные погрешности. Заметим также, что источником ошибок очень часто являются и исходные данные. Существуют, однако, методы, позволяющие использовать все достоинства метода Эйлера и одновременно повышающие точность вычислений. В нашей работе рассматривается несколько модифицированных методов Эйлера (расчетные схемы), по любому из них можно выполнять свои вычисления.

Рассмотрим основную расчетную формулу метода Эйлера (§ 1): $y_{i+1} = y_i + hf(x,y)$. Одна из модификаций метода заключается в том, что сначала вычисляют промежуточные значения

$$x_{i+1/2} = x_i + h/2 \quad \text{и} \quad y_{i+1/2} = y_i + f_i \cdot h/2$$

и находят направление поля интегральных кривых в средней точке $(x_{i+1/2},\ y_{i+1/2})$ отрезка $[x_i,\ x_{i+1}]$, т.е.

$$f_{i+1/2} = f(x_{i+1/2},\ y_{i+1/2}),$$

а затем полагают $y_{i+1} = y_{i+1/2} + f_i \cdot h/2$.

Другой модификацией этого же метода является нелинейная вычислительная схема: зная предыдущее значение y_i, вычисляют $y\tilde{}_{i+1}$ как $y\tilde{}_{i+1} = y_i + h\,y_i$, а затем определяют поправку к решению $\Delta y\tilde{}_{i+1}$ по формуле $\Delta y\tilde{}_{i+1} = f(x_{i+1}, y_{i+1})$. Решение находится из уравнения

$$y_{i+1} = y_i + h/2\cdot [\,f(x_i,y_i) + \Delta y\tilde{}_{i+1}\,] .$$

Эту вычислительную схему еще называют *методом Эйлера - Коши.*

Другая вычислительная схема, которую называют *методом Эйлера с упорядочением решения,* заключается в том, что каждое значение $y_{i+1} = y(x_{i+1})$, где $y(x)$ - искомое решение, а $x_{i+1} = x_0 + h\cdot(I+1)$, $i = 0, 1, ..., N$, определяется по следующей схеме:

1) за начальное приближение берется

$$y_{i+1}^{(0)} = y_i + h\cdot f\left(x_i . y_i\right),$$

2) найденное значение уточняется по формуле

$$y_{i+1}^{(0)} = y_i + \frac{h}{2}\cdot\left(f\left(x_i . y_i\right) + f\left(x_i . y_{i-1}\right)\right),$$ где $i = 1, 2, ..., N$;

3) уточнение продолжают до тех пор, пока в пределах требуемой точности (заранее заданной) два последовательных приближения не совпадут, т.е.

$$\left| y_{k+1}^{(i-1)} - y_{k+1}^{(i)} \right| \le \varepsilon ,$$

где ε - погрешность или любое малое число. После этого полагают $y_{k+1} = y_{k+1}^{-(i)}$, где $y_{k+1}^{-(i)}$ общая часть полученного ряда решений $y_{k+1}^{(i)}$.

Иногда описанная выше схема *называется методом Эйлера с итерационным уточнением решения.*

Если после трех итераций решения в пределах требуемой погрешности не совпали, то следует либо уменьшить шаг, либо увеличить погрешность.

В заключение краткого обзора отметим, что метод Эйлера с итерационным уточнением наиболее часто применяется в вычислительной практике, так как дает погрешность порядка h^2, т.е. относится к методам второго порядка, несмотря на то, что сам метод Эйлера является методом первого порядка.

При решении обыкновенных дифференциальных уравнений методом Эйлера с итерационным уточнением значения y_{k+1} удобно пользоваться подпрограммой-функцией, формальные значения параметров которой не приводятся в виду простоты и понятности предлагаемой процедуры:

```
FUNCTION EYLER ( X,Y,H,EPS : REAL) : REAL;
VAR YK, Y0 : REAL;
          DN : BOOLEAN;
BEGIN
   DN := FALSE;
   Y0 := Y + FUNC (X,Y) * H;
   REPEAT
      YK := Y + FUNC (X,Y0) * H * 0.5;
      IF ABS ( YK - Y0) < EPS THEN
         DN := TRUE;
      Y0 := YK;
   UNTIL DN;
   EYLER := Y0;
END.
```

В подпрограмме-функции EYLER используется процедура-функция FUNC(*x,y*) для вычисления правой части задачи. Ее следует составлять для каждого случая отдельно.

Для проверки и тестирования процедуры составим таблицу приближенных значений интеграла дифференциального уравнения $y' = x + \cos(0.7 - y)$ на отрезке [0.7; 1.7] с шагом $h = 0.1$, удовлетворяющего начальным условиям $y_0(0.7) = 1.1$, используя метод Эйлера с уточнением. Все вычисления произведем с точностью 10^{-5}.

Процедура-функция, по которой вычисляется правая часть уравнения, выглядит:

```
FUNCTION FUNC (X,Y: REAL) : REAL;
BEGIN
   FUNC := X + COS (0.7 * Y);
END;
```

Результаты работы программы приводятся в табл. 4.2.

Таблица 4.2

Решение дифференциального уравнения $y' = x + \cos(0.7 - y)$			
при $h = 0.1$		при $h = 0.05$	
$x = 0.70000$	$y = 1.100000$	$x = 0.700000$	$y = 1.100000$
		$x = 0.750000$	$y = 1.135017$
$x = 0.80000$	$y = 1.169169$	$x = 0.800000$	$y = 1.170830$
		$x = 0.850000$	$y = 1.207420$
$x = 0.90000$	$y = 1.241447$	$x = 0.900000$	$y = 1.244765$
		$x = 0.950000$	$y = 1.282844$
$x = 1.000000$	$y = 1.316671$	$x = 1.000000$	$y = 1.321637$
		$x = 1.050000$	$y = 1.361123$
$x = 1.100000$	$y = 1.394676$	$x = 1.100000$	$y = 1.401280$
		$x = 1.150000$	$y = 1.442088$
$x = 1.200000$	$y = 1.475300$	$x = 1.200000$	$y = 1.483526$
		$x = 1.250000$	$y = 1.525575$
x = 1.300000	$y = 1.558385$	$x = 1.300000$	$y = 1.568215$
		$x = 1.350000$	$y = 1.611427$
$x = 1.400000$	$y = 1.643779$	$x = 1.400000$	$y = 1.655191$
		$x = 1.450000$	$y = 1.699491$
$x = 1.500000$	$y = 1.731338$	$x = 1.500000$	$y = 1.744308$
		$x = 1.550000$	$y = 1.789626$
$x = 1.600000$	$y = 1.820929$	$x = 1.600000$	$y = 1.835429$
		$x = 1.650000$	$y = 1.881701$
$x = 1.700000$	$y = 1.912428$	$x = 1.700000$	$y = 1.928429$
		$x = 1.750000$	$y = 1.975598$

§ 3. ПРИБЛИЖЕННОЕ РЕШЕНИЕ ОБЫКНОВЕННОГО ДИФФЕРЕНЦИАЛЬНОГО УРАВНЕНИЯ МЕТОДОМ АДАМСА

Если дифференциальное уравнение $y' = f(x, y)$ имеет в правой части сложное аналитическое выражение, то тогда применяют *экстраполяционный метод Адамса,* который не требует многократного подсчета правой части.

Пусть для дифференциального уравнения заданы начальные условия $x = x_0$, $y = y_0$. Требуется найти решение уравнения $y' = f(x, y)$ на отрезке $[a, b]$.

Разобьем отрезок $[a, b]$ равномерно на n частей точками $x_i = x_0 + hi$, $i = 0, 1, ..., n$, $h = (b - a)/n$. Выберем произвольно элементарный отрезок, на котором проинтегрируем дифференциальное уравнение

$$y_{i+1} = y_i + \int_{X_i}^{X_{i+1}} y'dx \quad \text{или} \quad \Delta y_i = \int_{X_i}^{X_{i+1}} y'dx .$$

Для нахождения производной воспользуемся второй интерполяционной формулой Ньютона, ограничившись разностями третьего порядка с учетом $t = (x - x_i)/h$:

$$y' = y'_i + t \cdot \Delta y'_{i-1} + \frac{t \cdot (t+1)}{2!}\Delta^2 y'_{i-2} +$$

$$+ \frac{t \cdot (t+1) \cdot (t+2)}{3!}\Delta^3 y'_{i-3} \quad .$$

Подставим полученное выражение для y' в интегральное уравнение и, учитывая, что $dx = hdt$, имеем

$$\Delta y_i = h \cdot \int_0^h (y'_i + t \cdot \Delta y'_{i-1} + \frac{t^2 + t}{2}\Delta^2 y'_{i-2} +$$

$$+ \frac{t^3 + 3t^2 + 2t}{6}\Delta^3 y'_{i-3})dt = hy'_i + \frac{1}{2}\Delta(h \cdot y'_{i-1}) +$$

$$+ \frac{5}{12}\Delta^2(h \cdot y'_{i-2}) + \frac{3}{8}\Delta^3(h \cdot y'_{i-2}) \quad .$$

Обозначим через $q_i = y_i'$; $h = f(x_i, y_i) \cdot h$, $\forall i \in \overline{0, N}$, тогда для любой разности $\Delta^m q_i = \Delta^m (y_i' h)$ имеем выражение

$$\Delta y_i = q_i + 1/2 \cdot \Delta q_{i-1} + 5/12 \cdot \Delta^2 q_{i-2} + 3/8 \cdot \Delta^3 q_{i-3},$$

используемое для получения решения уравнения

$$y_{i+1} = y_i + \Delta y_i.$$

Две последние формулы являются основными в экстраполяционном методе Адамса.

Схема предлагается из работы [Численные ..., 1976]. Для начала процесса вычисления нужны четыре начальных значения y_0, y_1, y_2 и y_3, которые можно определить любым известным методом. Далее, зная y_0, y_1, y_2 и y_3, находят $q_0 = hy_0' = h\,f(x_0, y_0)$; $q_2 = hy_2' = h\,f(x_2, y_2)$; $q_3 = hy_3' = h\,f(x_3, y_3)$; $q_4 = hy_4' = h\,f(x_4, y_4)$ и составляют таблицу конечных разностей величин q (табл. 4.3)

Таблица 4.3

№ п/п	x_i	y_i	Δy_i	$y_i' = f(x_0, y_0)$	$q_i = hy_i'$	Конечные разности		
0	x_0	y_0		$f(x_0, y_0)$	—	—	—	—
1	x_1	y_1		$f(x_1, y_1)$	q_0	Δq_0	$\Delta^2 q_0$	$\Delta^3 q_0$
2	x_2	y_2		$f(x_2, y_2)$	q_1	Δq_1	$\Delta^2 q_1$	
3	x_3	y_3	Δy_3	$f(x_3, y_3)$	q_2	Δq_2		
4	x_4	y_4		$f(x_4, y_4)$	q_3			
5	x_4	y_5		...	...			
...	...	...						

Метод Адамса заключается в продолжении данной таблицы разностей с помощью формулы для Δy_i. Используя уже вычисленные q_3, Δq_2, $\Delta^2 q_1$ и $\Delta^3 q_0$, расположенные в таблице диагонально, по формуле для Δy_i получают, полагая $n = 3$,

$$\Delta y_3 = q_3 + 0.5\Delta q_2 + (5/12) \cdot \Delta^2 q_1 + (3/8) \cdot \Delta^3 q_0,$$

Δy_3 вносят в таблицу и находят $y_4 = y_3 + \Delta y_3$. Затем, используя x_4 и y_4 находят $f(x_4, y_4)$, q_4, Δq_3, $\Delta^2 q_2$ и $\Delta^3 q_1$, т.е. новую диагональ. По этим данным определяют значение Δy_4, которое тут же вносят в таблицу, и находят $y_5 = y_4 + \Delta y_4$.

Таблицу продолжают по описанному алгоритму до ее заполнения, вычисляя правую часть формулы при этом только один раз. Чтобы оценить погрешность полученного результата, можно применить правило Рунге или просто следить за третьими разностями $\Delta^3 q_i$, которые считаются постоянными. Этого можно добиться, выбирая h каждый раз такой, чтобы выражение для оценки погрешности было $|\Delta^3 q_{i-1} - \Delta^3 q_i| < \varepsilon$. На практике h выбирают из неравенства $h^4 < \varepsilon$, где ε - заданная точность решения.

Метод Рунге состоит в том, что сначала находится решение дифференциального уравнения при шаге h, а затем значение h удваивается и находится решение при новом шаге. Погрешность оценивается по формуле

$$\varepsilon = (2^m - 1) \cdot |y_n^{\sim} - y_{2n}^{\sim}| ,$$

где $y_n^{\sim}$ - значение приближенного вычисления при двойном шаге; m - порядок метода.

Процедура, реализующая указанную схему, может быть следующей:

```
PROCEDURE ADAMS (A,B,H,EPS,YN : REAL;
                 VAR YI:MAS1);
TYPE MAS = ARRAY [1..20] OF REAL;
VAR X0, Y0, Y : REAL; FF : TEXT; I : INTEGER;
                 XI, DYI, YSH, QI, DQI, D2QI, D3QI : MAS;
```

```
      {**** □□□□□□□□□ "□□□□□ □□□□□ □ □□□□□□□□□" ***}
  FUNCTION EYLER ( X,Y,H,EPS : REAL) : REAL;
  VAR YK, Y0 : REAL; DN : BOOLEAN;
  BEGIN
     DN := FALSE;
     Y0 := Y + FUNC (X,Y) * H;
     REPEAT
        YK := Y + FUNC (X,Y0) * H * 0.5;
        IF ABS ( YK - Y0) < EPS THEN DN := TRUE;
        Y0 := YK;
     UNTIL DN;
     EYLER := Y0;
  END;

        { ** □□□□□□□□□ "□□□□□ □□□□□ " ** }
BEGIN
  FOR I := 1 TO 20 DO XI[I] := 0.0;
  DYI := XI;
  Y0 := YN;
  YI:=XI;
  QI:= XI;
  DQI := XI;
  D2QI:= XI;
  D3QI:=XI;
  YSH:=XI;
  X0 := A;
  XI[1] := X0;
  YI[1] := Y0;
  I := 1;
  YSH [1] := FUNC (X0,Y0);
  QI[1] := H*YSH[1];
  REPEAT
     Y := EYLER (X0,Y0,H,EPS);
     INC (I);
     X0 := X0 + H;
     Y0 := Y;
     XI[I] := X0;
     YI[I] := Y;
     YSH [I] := FUNC (X0,Y0);
     QI[I] := H*YSH[I];
     DQI[I-1] := QI[I] - QI[I-1];
     IF I>2 THEN
           D2QI[I-2] := DQI[I-1]-DQI[I-2];
     IF I>3 THEN
           D3QI[I-3] := D2QI[I-2]-D2QI[I-3];
  UNTIL I >= 4;
  REPEAT
     DYI[I]:= QI[I]+0.5*DQI[I-1]+5*D2QI[I-2]/12+
           3*D3QI[I-3]/8; INC(I); X0 := X0 + H;
     XI[I] := X0;
     YI[I] := YI[I-1] + DYI[I-1];
     YSH [I] := FUNC (XI[I],YI[I]);
     QI[I] := H*YSH[I];
     DQI[I-1] := QI[I] - QI[I-1];
     D2QI[I-2] := DQI[I-1]-DQI[I-2];
     D3QI[I-3] := D2QI[I-2]-D2QI[I-3];
  UNTIL I=10;
END;
```

Формальные параметры процедуры. *Входные*: *a, b* (тип ***real***) - начало и конец отрезка, на котором ищут решение; *h* (тип ***real***) - шаг разбиения отрезка [*a, b*]; *EPS* (тип ***real***) - заранее заданное малое число (используется для определения первых четырех значений y_i методом Эйлера с уточнением); y_N - начальное y_0. Перед началом работы следует определить процедуру-функцию, по которой вычисляют правую часть дифференциального уравнения *f* (*x, y*). *Выходные*: массив *yi* (тип ***real***) значений y_i, найденных методом Адамса.

Для тестирования и проверки процедуры предлагается вычислить при $x = 0.1$ с точностью 10^{-4} по методу Адамса значения решения дифференциального уравнения $y' = f(x, y)$ с начальными условиями $x = x_0$; $y(x_0) = y_0$; $y' = 0.185 \diamond \diamond (x^2 + \cos 0.7x) + 1.843\, y$; $x_0 = 0.2$; $y_0 = 0.25$; $h = 0.1$. Как и в § 2, здесь необходимо доопределить процедуру-функцию, вычисляющую значение *y* по заданным *x*:

```
FUNCTION FUNC (X,Y: REAL) : REAL;
BEGIN
  FUNC := 0.185 ; (X;X + COS (0.7 * X))+ 1.843;Y;

END;
```

Результаты работы программы приведены в табл. 4.4 (в табл. 4.5 представлено также решение того же уравнения, полученное по методу Эйлера с уточнением).

Таблица 4.4

Номер	Массив							
i	*xi*[*i*]	*yi*[*i*]	*dyi*[*i*]	*ysh*[*i*]	*qi*[*i*]	*dqi*[*i*]	*d2qi*[*i*]	*d3qi*
1	0.20	0.2500	0.00000	0.65134	0.06513	0.00731	0.00102	0.00011
2	0.30	0.2858	0.00000	0.72445	0.07245	0.00833	0.00113	0.00855
3	0.40	0.3257	0.00000	0.80780	0.08078	0.00946	0.00968	-.00373
4	0.50	0.3702	0.09549	0.90244	0.09024	0.01915	0.00595	-.00162
5	0.60	0.4657	0.12621	1.09391	0.10939	0.02510	0.00433	0.00181
6	0.70	0.5919	0.14811	1.34487	0.13449	0.02942	0.00614	0.00166
7	0.80	0.7400	0.17982	1.63910	0.16391	0.03556	0.00779	0.00133
8	0.90	0.9198	0.22048	1.99469	0.19947	0.04335	0.00912	0.0000
9	1.00	1.1403	0.26836	2.42821	0.24282	0.05248	0.00000	0.0000
10	1.10	1.4087	0.00000	2.95297	0.29530	0.00000	0.00000	0.0000

Таблица 4.5

Точки интервала	Решение уравнения	Точки интервала	Решение уравнения
		$x_5 = 0.700000$	$y_5 = 0.475552$
$x_0 = 0.200000$	$y_0 = 0.250000$	$x_6 = 0.800000$	$y_6 = 0.537801$
$x_1 = 0.300000$	$y_1 = 0.285875$	$x_7 = 0.900000$	$y_7 = 0.607540$
$x_2 = 0.400000$	$y_2 = 0.325772$	$x_8 = 1.000000$	$y_8 = 0.685688$
$x_3 = 0.500000$	$y_3 = 0.370259$	$x_9 = 1.100000$	$y_9 = 0.773265$
$x_4 = 0.600000$	$y_4 = 0.419957$	$x_{10} = 1.200000$	$y_{10} = 0.871391$

§ 4. ПРИБЛИЖЕННОЕ РЕШЕНИЕ ОБЫКНОВЕННОГО ДИФФЕРЕНЦИАЛЬНОГО УРАВНЕНИЯ МЕТОДОМ РУНГЕ - КУТТА

Методы Рунге - Кутта образуют семейство методов решения задачи Коши. Они относятся к многошаговым методам повышенной точности (от второго порядка и выше). Здесь будет рассмотрен одношаговый метод Рунге - Кутта четвертого порядка точности. Заметим попутно, что методы Рунге - Кутта отличаются от разностных тем, что все они допускают вычисление функции не только в заранее заданных узлах сетки, но и в некоторых промежуточных точках, тем самым позволяя уменьшать шаг вычислений во время самого процесса построения решетки.

Итак, пусть требуется найти решение дифференциального уравнения $y' = f(x, y)$, удовлетворяющего начальному условию $y(x_0) = y_0$.

Численное решение задачи состоит в построении таблицы приближенных значений y_0, y_1, ..., y_n решения уравнения $y' = f(x,y)$ в точках x_0, x_1, ..., x_n, которые называют *узлами сетки*. Для краткости обозначим $x_i = x_0 + ih$; $y_i = y(x_i)$, где h - шаг сетки (ясно, что $h > 0$).

Обычно методом Рунге-Кутта в литературе называют метод, согласно которому y искомой функции вычисляют по формулам:

$$y_{i+1} = y_i + \Delta/6,$$

где обозначено:

$$\Delta = k_1 + 2k_2 + 2k_3 + k_4; \qquad k_1 = hf(x_i, y_i);$$

$$k_2 = h\,f(x_i + h/2, y_i + k_1/2); \quad k_3 = h\,f(x_i + h/2, y_i + k_2/2);$$

$$k_4 = h\,f(x_i + h, y_i + hk_3).$$

Погрешность метода на одном шаге сетки равна Mh^5, где M - некоторое число. Но на практике даже приближенно очень трудно оценить M, поэтому при оценке погрешности используют *правило Рунге*. Для чего сначала проводят вычисления с шагом h, а затем повторяют их с числом $h/2$. Если $y_i^{(h)}$ - приближение, вычисленное с шагом h, а $y_i^{(h/2)}$ - с шагом $h/2$, то справедлива оценка

$$\left| y_i^{(h/2)} - y(x_i) \right| \le \frac{16}{5} \cdot \left| y_i^{(h/2)} - y_i^{(h)} \right|.$$

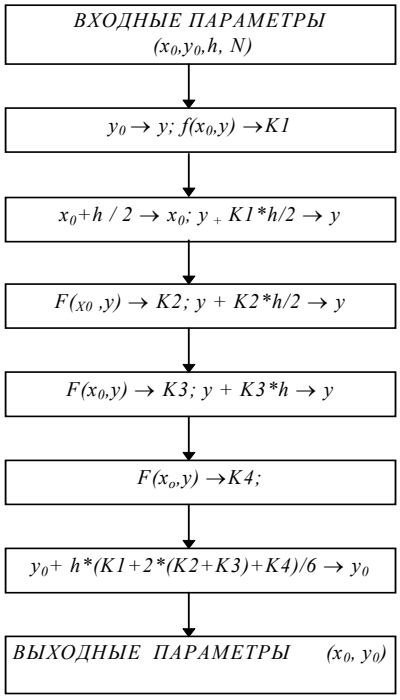

Рис. 4.1. *Вычислительная схема алгоритма Рунге - Кутта для решения обыкновенного дифференциального уравнения*

Именно поэтому при реализации метода на ЭВМ обычно на каждом шаге делают двойной пересчет. Если полученные значения отличаются в пределах допустимой погрешности, то шаг h удваивают. В противном случае шаг уменьшают вдвое. На рис. 4.1 приводится схема этого алгоритма.

Метод Рунге - Кутта обладает повышенной точностью и, несмотря на трудоемкость, широко используется в вычислительной математике для решения дифференциальных уравнений. Его легко обобщить для решения систем линейных дифференциальных уравнений.

В БСП ЭВМ обычно используют эффективную программу RKF45, написанную Уотсом и Шампэнь в 1974 г. Однако решению этой же задачи посвящены работы и других авторов, сведения о которых можно найти в книге [Форсайт, 1980], где все вычислительные методы подробно рассмотрены, в том числе и RKF45.

Здесь же предлагается к применению один из самых простых алгоритмов [Плис, Сливина, 1983], реализованный на ЭВМ МИР-2 (перевод на язык *PASCAL* выполнен авторами), который довольно эффективен, но, к сожалению, незаслуженно забыт. Алгоритм реализован в процедуре RGK.

```
PROCEDURE RGK (N:INTEGER; Y,H,X : REAL;  VAR YR : MAS);
VAR I : INTEGER; F1, F, Y0, X0 : REAL;
BEGIN I := 2;  Y0 := Y;
```

```
    X0 := X;
    YR[1] := Y0;
    REPEAT
        F1 := H * FUNC (X0,Y0);
        F := H * FUNC (X0+H/2,Y0+F1/2);
        F1 := F1 + F*2;
        F := H * FUNC (X0+H/2,Y0+F/2);
        F1 := F1 + F*2;
        F := H * FUNC (X0+H,Y0+F);
        F1 := F1 + F;
        YR[I]:=YR[I-1] + F1 / 6;
        Y0 := YR [I];
        X0 := X0 + H;
        INC (I);
    UNTIL I>N;
END.
```

Формальные параметры процедуры. *Входные*: *N* (тип ***integer***) - количество разбиений отрезка [*a, b*]; *X, Y* (тип ***real***) - начальные значения, соответствующие x_0 и y_0; *h* (тип ***real***) - шаг интегрирования системы; *FUNC* - имя процедуры-функции, по которой вычисляют значения правой части уравнения $y' = f(x, y)$. *Выходные*: *YR* (тип ***real***) - массив, содержащий решения системы в точках $x_i = x_0 + hi$.

Для сравнения и тестирования процедуры здесь, по данным § 1 и 2, решалась задача Коши методом Рунге - Кутта для заданного шага *h*. Все вычисления сведены в табл. 4.6 и выполнены с точностью 10^{-5}. В табл. 4.6 сравниваются результаты, полученные по методу Эйлера (столбец *EYLER*) и по методу Эйлера с уточнением (столбец *EYLER2*).

Заметим, что решение, полученное методом Рунге - Кутта (столбец Runge - Kutt), располагается несколько выше по сравнению с решениями, полученными по методам Эйлера. Для того, чтобы выяснить вопрос, какое из решений ближе к истинному, необходимо дополнительное исследование функции.

Таблица 4.6

N п/п	Узлы *X*	Runge-Kutt	*EYLER*	*EYLER2*
1	0.20	0.250000	0.250000	0.250000
2	0.30	0.321868	0.285875	0.285871
3	0.40	0.409199	0.325772	0.325768
4	0.50	0.515431	0.370259	0.370254
5	0.60	0.644700	0.419957	0.419952
6	0.70	0.801984	0.475552	0.475547
7	0.80	0.993267	0.537801	0.537795
8	0.90	1.225753	0.607540	0.607533
9	1.00	1.508101	0.685688	0.685681
10	1.10	1.850732	0.773265	0.773257
11	1.20	1.935423	0.871391	0.871382

§ 5. ПРИБЛИЖЕННОЕ РЕШЕНИЕ ОБЫКНОВЕННОГО ДИФФЕРЕНЦИАЛЬНОГО УРАВНЕНИЯ МЕТОДОМ ПРОГОНКИ

Рассмотрим линейное дифференциальное уравнение второго порядка

$$y'' + p(x)y' + q(x)y = f(x) \qquad (4.2)$$

с краевыми условиями:

$$\begin{cases} \alpha_0 \cdot y(a) + \alpha_1 \cdot y'(a) = A; \\ \beta_0 \cdot y(b) + \beta_1 \cdot y'(b) = B. \end{cases} \qquad (4.3)$$

где $|\alpha_0| + |\alpha_1| > 0$ и $|\beta_0| + |\beta_1| > 0$, а $p(x)$, $q(x)$, $f(x)$ - известные функции, непрерывные на отрезке [*a, b*].

Численное решение задачи (4.2) - (4.3) состоит в нахождении приближенных значений y_0, y_1, ..., y_n искомого решения $y(x)$ в точках x_0, x_1, , ..., x_n.

Одним из наиболее распространенных методов решения этой краевой задачи является сведение ее к системе конечно-разностных уравнений. Используя равномерную сетку, образованную системой равноотстоящих узлов $x_i = x_0 + ih$, $i = 0, 1, ..., n$, аппроксимируем $y'(x)$ и $y''(x)$ в каждом внутреннем узле центральными разностями

$$y'(x_i) = \frac{y_{i+1} - y_{i-1}}{2h} + \Theta(h^2);$$

$$y''(x_i) = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + \Theta(h),$$

а на концах отрезка - односторонними

$$y'(x_0) = \frac{y_1 - y_0}{h} + \Theta(h); \quad y'(x_n) = \frac{y_n - y_{n-1}}{h} + \Theta(h).$$

Обозначив $x_0 = a$; $x_n = b$; $h = (b - a) / n$; $p(x_i) = p_i$; $q(x_i) = q_i$; $f(x_i) = f_i$; $y'(x_i) = y_i'$; $y''(x_i) = y_i''$; $f(x_i) = y_i$, получим систему линейных уравнений

$$\begin{cases} \dfrac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + p_i \cdot \dfrac{y_{i+1} - y_{i-1}}{2h} + q_i \cdot y_i = f_i; \\ \alpha_0 y_0 + \alpha_1 \cdot \dfrac{y_1 - y_0}{h} = A; \\ \beta_0 y_n + \beta_1 \cdot \dfrac{y_n - y_{n-1}}{h} = B. \end{cases} \qquad (4.4)$$

Теперь, чтобы найти приближенные значения y_0, y_1, ..., y_n искомого решения, надо решить эту систему из $n+1$ линейных уравнений с $n+1$ неизвестными, что можно сделать любым стандартным методом решения линейных систем, который будет называться по типу разложения *конечно-разностным*. Однако матрица последней системы трехдиагональная, поэтому для ее решения применима специальная вычислительная схема, называемая *методом прогонки*.

Перепишем систему (4.4) в несколько ином виде:

$$\begin{cases} y_{i+1} + m_i \cdot y_i + n_i \cdot y_{i-1} = h^2 \cdot f_i; \\ a_0 y_0 + a_1 \cdot \dfrac{y_1 - y_0}{h} = A; \\ b_0 y_n + b_1 \cdot \dfrac{y_n - y_{n-1}}{h} = B, \end{cases} \qquad (4.5)$$

где $m_i = -2 + h p_i$; $n_i = 1 - h p_i + h^2 q_i$.

Решением уравнения (4.5) относительно y_i будет

$$y_i = f_i h^2 / m_i - n_i y_{i-1} / m_i - y_{i+1} / m_i . \qquad (4.6)$$

Предположим, что y_i уже найдено, тогда (4.6) можно записать

$$y_i = c_i (d_i - y_{i+1}), \qquad (4.7)$$

где надо определить неизвестные c_i и d_i .

Если $i = 0$, то на основании одного из краевых условий (4.5) имеем

$$y_0 = (\alpha_1 y_1 - A h) / (\alpha_1 - \alpha_0 h) . \qquad (4.8)$$

Подставим выражение (4.8) в (4.5) и, считая $i = 0$, получим

$$y_1 = \frac{f_0}{m_0} h^2 - \frac{n_0}{m_0} \cdot \frac{\alpha_1 y_1 - Ah}{\alpha_1 - \alpha_0 h} - \frac{1}{m_0} \cdot y_2 ,$$

откуда выразим явно y_1

$$y_1 = \frac{\alpha_1 y_1 - Ah}{m_0 \cdot (\alpha_1 - \alpha_0 h) + n_0 \alpha_0} \cdot \left(\frac{n_0 Ah}{\alpha_1 - \alpha_0 h} + f_0 h^2 - y_2 \right) .$$

Сравнив последнее равенство с равенством (4.7), определим c_0 и d_0

$$\begin{cases} c_0 = \dfrac{a_1 y_1 - Ah}{m_0 \cdot (a_1 - a_0 h) + n_0 a_0} ; \\ d_0 = \dfrac{n_0 Ah}{a_1 - a_0 h} + f_0 h^2 . \end{cases} \qquad (4.9)$$

Теперь последовательно будем искать c_i и d_i. Для этого выразим y_{i-1} из равенства (4.7) и подставим полученное выражение в выражение (4.6), из которого после элементарных преобразований найдем y_i :

$$y_i = \frac{1}{m_i - n_i c_{i-1}} \cdot \left(f_i h^2 - n_i c_{i-1} d_{i-1} - y_{i+1} \right) .$$

Сравним последнее выражение с равенством (4.7), определим c_i и d_i :

$$\begin{cases} c_i = \dfrac{1}{m_i - n_i c_{i-1}} ; \\ d_i = f_i h^2 - n_i c_{i-1} d_{i-1} . \end{cases} \qquad (4.10)$$

И, наконец, при $i = n$ используем второе краевое условие из системы (4.5), подставив в него y_{n-1} из равенства (4.7):

$$y_n = \frac{\beta_1 c_{n-1} d_{n-1} + B \cdot h}{\beta_1 \cdot (1 + c_{n-1}) + \beta_0 \cdot h} . \qquad (4.11)$$

Общая схема расчета по методу прогонки следующая:

1) по уравнениям (4.10) и (4.9) определяем значения коэффициентов c_i и d_i , $i = 0, 1, ..., n - 1$;

2) по уравнениям (4.11) находим y_n ;

3) по уравнениям (4.7) вычисляем $y_{n-1}, ..., y_2, y_1$;

4) по уравнениям (4.8) находим y_0 .

Таким образом, данный метод позволяет найти решение системы (4.5), а значит, полная погрешность метода будет определяться только погрешностью разностной аппроксимации, которая равна $\Theta(h)$. Так как $h = (b - a)/n$, то, выбирая n, можно добиться уменьшения погрешности. Но при этом следует заметить, что, во-первых, из-за некорректности операции численного дифференцирования, и, во-вторых, из-за возрастания вычислительной погрешности обычно используют двойной пересчет и правило Рунге (см. § 4). Тогда сумма погрешности решения y_i определится формулой

$$|y\tilde{}_i - y(x_i)| \cong |y_i - y\tilde{}_i| / 3,$$

где $y\tilde{}_i$ - решение при $4h$; y_i - решение при h; $y(x_i)$ - точное решение.

Как метод Рунге - Кутта, так и метод прогонки на практике используется достаточно часто, поэтому БСП всегда содержат такую процедуру. Здесь приводится процедура из БСП ЕС ЭВМ для языка *FORTRAN*. (Перевод на язык *PASCAL* выполнен авторами).

```
PROCEDURE PROG ( N:INTEGER;
          A,B,ALFA0,ALFA1,AL,BETA0,BETA1,BT:REAL;
          VAR Y : MAS);
VAR H, R1, R2,X, T : REAL; J, I, I1 : INTEGER;
          P, Q : MAS;
BEGIN
  H := (B-A)/N;
  R1 := H*H;
  R2 := H*0.5;
  P[1] := -ALFA1 / (ALFA0*H-ALFA1);
  Q[1] := -AL*H*P[1]/ALFA1;
  X := A;
  FOR I := 2 TO N DO
  BEGIN      I1 := I-1;
     X := X + H;
     T := 1.0 - NI(X)*R2;
     P[I] := (T-2.0) / (MI(X)*R1+T*P[I1] - 2);
     Q[I] := (FI(X)*R1 - T*Q[I1])*P[I]/(T-2);
  END;
  P[N+1] := 0;
  Q[N+1]:=(BT*H+BETA1*Q[N]) / (BETA0*H+
            BETA1-BETA1*P[N]);
  Y[N+1] := Q[N+1];
  FOR J := 1 TO N DO
  BEGIN
     I := N-J+1; I1 := I+1;
```

```
    Y[I] := P[I]*Y[I1]+Q[I];
  END;
END.
```

Формальные параметры процедуры. *Входные*: *a*, *b* (тип ***real***) - границы отрезка, на котором ищется решение; *n* (тип ***integer***) - количество узлов сетки; *alfa*1, *alfa*0, *al*, *beta*0, *beta*1, *bt* (тип ***real***) - значения коэффициентов α_1, α_0, *A*, β_0, β_1, *B*; *N*(*x*, *i*), *M*(*x*, *i*), *F*(*x*, *i*) - встроенные внешние процедуры-функции (тип ***real***), вычисляющие $p(x_i)$, $q(x_i)$ и $f(x_i)$ в точках x_i сетки. *Выходные*: *Y* (тип ***real***) - массив решений $y_0, y_1, \ldots, y_n$.

Для примера здесь решается на отрезке [*a*, *b*] методом прогонки линейная краевая задача (4.2) с краевыми условиями (4.3), если $p(x) = e^x$; $q(x) = x/2$; $\alpha_0 = 1$; $\alpha_1 = -1.2$; $A = 0$; $\beta_0 = 2$; $\beta_1 = -2.5$; $B = -4$; $f(x) = x^2$; $a = 0.1$; $b = 1.1$. Вычисления выполнялись с точностью 10^{-5}.

Подставив данные, получим следующую задачу:

$$y'' + e^x y' + 0{,}5\,x\,y = x^2$$

с краевыми условиями

$$y(0.1) - 1.2\,y'(0.1) = 0;\ 2\,y(1.1) - 2.5\,y'(1.1) = -4\ .$$

Результаты проверки работы данной процедуры приведены в табл. 4.7.

Таблица 4.7

i	x_i	y_i	*i*	x_i	y_i
1	0.10	-1.257594	11	0.60	-1.625558
2	0.15	-1.309994	12	0.65	-1.642122
3	0.20	-1.359142	13	0.70	-1.654892
4	0.25	-1.404951	14	0.75	-1.663893
5	0.30	-1.447334	15	0.80	-1.669171
6	0.35	-1.486207	16	0.85	-1.670797
7	0.40	-1.521492	17	0.90	-1.668859
8	0.45	-1.553121	18	0.95	-1.663469
9	0.50	-1.581036	19	1.00	-1.654759
10	0.55	-1.605191	20	1.05	-1.642880
			21	1.10	-1.628000

5.

Под специальными функциями в широком смысле понимают совокупность отдельных классов функций, возникающих при решении как теоретических, так и ряда прикладных задач в самых разнообразных областях математики, физики и техники. В более узком смысле под специальными функциями понимаются функции математической физики, появляющиеся при интегрировании дифференциальных уравнений с частными производными методом разделения переменных.

Специальные функции могут быть определены с помощью степенных, тригонометрических рядов и рядов по ортогональным функциям, производящих функций, бесконечных произведений, последовательного дифференцирования, интегральных представлений, а также дифференциальных, разностных, интегральных и функциональных уравнений.

К наиболее важным классам специальных функций, чаще всего встречающихся в приложениях, относятся гамма-функция и связанные с ней функции, функции Бесселя и Эйри, интегральные синус и косинус, эллиптические функции Якоби и эллиптические интегралы, а также некоторые другие функции.

Вообще говоря, теория специальных функций (спецфункций) связана с представлениями групп, методами интегральных представлений, опирающихся на обобщение формулы Родрига для классических ортогональных многочленов, а также с методами теории вероятностей. Однако рассмотрение теоретических аспектов спецфункций как математических объектов не входит в задачу настоящей главы, цель которой изучение вычислительных аспектов теории, для того чтобы эффективно применять аппарат спецфункций при решении задач, возникающих в математических, физических и технических приложениях.

В настоящее время имеется много справочников по специальным функциям, среди которых, в первую очередь, следует отметить книгу Ю.Люка [1980], содержащую большой табличный материал. Однако развитие численного анализа и усовершенствование вычислительных машин делают экономически невыгодным хранить таблицы в памяти ЭВМ с тем, чтобы с помощью специальных программ затем осуществлять их поиск для последующей интерполяции. Поэтому весьма актуальной является задача построения оптимальных алгоритмов, позволяющих выполнять вычисление широких классов специальных функций.

В данную главу включены подготовленные к непосредственному использованию на ЭВМ специально отобранные программные модули, реализующие наиболее эффективные алгоритмы вычисления спецфункций. Материал главы может оказаться весьма полезным для выбора наиболее оптимальных подходов к построению алгоритмов соответствующего класса.

Результаты, представленные здесь, можно использовать как дополнительный учебный материал для курсов "Численные методы" и "Вычислительная физика", которые авторы читают на физико-математическом факультете МПУ. Содержание главы включает в себя как краткое изложение основных определений и формулировку разного рода представлений, так и примеры эффективных алгоритмов вычисления спецфункций с процедурами на языке *PASCAL* (перевод с языка *FORTRAN* в версии *FORTRAN*-77 [Белашов, 1997] выполнен авторами), снабженными результатами тестовых расчетов.

Представленный материал, кроме того, можно эффективно использовать при изучении ряда дисциплин теоретической физики (например, при решении задач дифракции на физических объектах, обладающих высокой степенью симметрии, при изучении некоторых точно интегрируемых математических моделей в теории солитонов и т.п.), а также быть полезным преподавателям, ведущим соответствующие курсы, и специалистам, работа которых требует применения средств вычислительной техники для решения соответствующих математических задач.

§ 1. ГАММА-ФУНКЦИЯ И СВЯЗАННЫЕ С НЕЙ ФУНКЦИИ

Гамма-функция $\Gamma(z)$, по определению, есть решение функционального уравнения

$$\Gamma(z+1) = z\,\Gamma(z), \quad \Gamma(1) = 1 \qquad (5.1)$$

и является мероморфной функцией аргумента $z = x + iy$ с простыми полюсами в точках $z = -n$, $n = 0, -1, -2, \ldots$. Для целых значений аргумента $m = 0, 1, 2, \ldots$ гамма-функция определяется через факториал

$$\Gamma(m + 1) = m\,! \quad . \qquad (5.2)$$

Отметим также, что гамма-функция удовлетворяет уравнению [Янке и др., 1968]

$$\Gamma(z)\,\Gamma(-z) = -(\pi / z)\sin(\pi z) .$$

В пределах настоящего параграфа рассмотрим алгоритмы вычисления гамма-функции и связанных с ней функций только для $z \equiv \mathrm{Re}\ z = x$, при этом для больших x имеет место асимптотическое разложение Стирлинга [Корн, Корн, 1984]:

$$\Gamma(x) = \sqrt{\frac{2\pi}{x}} e^{x(\ln(x)-1)} \times \times\left(1 + \frac{1}{12x} + \frac{1}{288x^2} - \frac{139}{51840x^3} - \ldots\right), \quad (5.3)$$

где абсолютная величина ошибки меньше модуля последнего удержанного члена. В ряде приложений большое значение играет не сама гамма-функция, а обратная ей функция $1/\Gamma(x)$, которая при $x \in [-1, 1]$ может быть разложена в ряд [Справочник ..., 1979]

$$\frac{1}{\Gamma(x)} = x(1+x)\left(1 + \sum_{i=1}^{\infty} a_i x^i\right), \quad (5.4)$$

коэффициенты которого представлены в табл. 5.1.

Таблица 5.1

i	a_i	i	a_i
1	- 0.422784335092	7	- 0.000804341335
2	- 0.233093736365	8	- 0.000360851496
3	0.191091101162	9	0.000145624324
4	- 0.024552490887	10	1.7527917e-5
5	- 0.017645242118	11	2.625721e-6
6	0.008023278113	12	1.328554e-6

Логарифмическая производная $\psi(x)$ *гамма-функции* определяется выражением [Справочник ..., 1979]

$$\psi(z) = \Gamma'(z)/\Gamma(z), \quad \psi(1) = -0.5772156649015...$$

Вычисление $\psi(x)$ можно проводить как с учетом формулы (5.4), так и на основании соотношений

$$\psi(x) = \Psi(x+1) - 1/x; \quad \Psi(x) = -\pi \operatorname{ctg}(\pi x) + \psi(1-x) \quad (5.5)$$

или асимптотического разложения

$$\psi(x) = \ln(x) - \frac{1}{2x} - \frac{1}{12x^2} + \frac{1}{120x^4} + \frac{1}{252x^6} - \ldots \quad (5.6)$$

Полная бета-функция есть, по определению Корн и др. (1984),

$$B(p,q) = \frac{\Gamma(p)\Gamma(q)}{\Gamma(p+q)}.$$

Неполные гамма-функция $\Gamma_z(p)$ и *бета-функция* $B_z(p,q)$ определяются аналитическим продолжением интегралов в верхние полуплоскости соответственно p и p, q:

$$\Gamma_z(p) = \int_0^z t^{p-1} e^{-t} dt, \quad \operatorname{Re}(p) > 0;$$

$$B_z(p,q) = \int_0^z t^{p-1}(1-t)^{q-1} dt; \quad (5.7)$$

$$\operatorname{Re}(p) > 0;\ \operatorname{Re}(q) > 0;\ 0 \le z \le 1.$$

Отметим, что значения неполной гамма-функции при $x = z \equiv \operatorname{Re} z$, $a = p \equiv \operatorname{Re} p$ легко могут быть вычислены с помощью разложения в степенной ряд [Янке и др., 1968]

$$\Gamma_x(a) = x^a \sum_{i=0}^{\infty} \frac{(-1)^i x^i}{i!(a+i)}. \quad (5.8)$$

Еще одна величина, связанная с гамма-функцией и называемая *отношением неполной бета-функции*, определяется по формуле

$$I_z(p,q) = \frac{B_z(p,q)}{B(p,q)}, \quad (5.9)$$

при этом для $x = z \equiv \operatorname{Re} z$ справедливо следующее соотношение симметрии:

$$I_x(p,q) = 1 - I_{1-x}(q,p). \quad (5.10)$$

На основании введенных определений построены процедуры вычисления $\Gamma(x)$, $1/\Gamma(x)$, $\psi(x)$, $\Gamma_x(p)$ и $I_x(p,q)$.

С помощью процедуры-функции GAMF в зависимости от способа ее вызова вычисляют $\Gamma(x)$ или $1/\Gamma(x)$, исключая вначале (в первом случае) точки $x = 0, -1, -2, \ldots$, где $\Gamma(x)$ имеет полюсы, и приводя затем значения аргумента к $x \in [-1, 1]$ на основании соотношения (5.1). В случае если при заданном значении аргумента вычисляемая функция $\Gamma(x)$ имеет полюс, выход из программы осуществляется путем обращения к программе обработки ошибки. Основной вычислительный процесс строится на использовании разложения обратной гамма-функции (5.4).

Формальные параметры процедуры. *Входные*: x (тип ***real***) - аргумент гамма-функции; k (тип ***integer***) - переключатель: при $k = -1$ вычисляется $1/\Gamma(x)$, при всех других значениях k полагается равным $\Gamma(x)$. *Выходной*: *gamf* (тип ***double***) - значение $\Gamma(x)$ или $1/\Gamma(x)$.

```
FUNCTION GAMF (X: REAL;K:INTEGER) : REAL;
VAR G : REAL;
LABEL A0, A1, A2, A3;
BEGIN
  IF (K<> -1) AND INT(X)=-ABS(X) THEN
  BEGIN
     WRITE ("ОШИБКА В ИСХОДНЫХ ДАННЫХ ");
     EXIT;
  END;
  GAMF:=1;
  WHILE  X>1 DO
  BEGIN
     X:=X-1;
     GAMF=GAMF*X;
  END;
  GAMF:=1/GAMF;
  WHILE X<-1 DO
  BEGIN
     GAMF:=GAMF*X;
     X:=X+1;
  END;
  IF X<>1 THEN
```

```
  BEGIN
     G:=(((((((((-0.00000018122*X+0.000001328554)*X
      -0.000002625721)*X-0.000017527917)*X+
      0.000145624324)*X-0.000360851496)*X-
      0.000804341335)*X+0.008023278113)*X
      -0.017645242118)*X-0.024552490887)*X;
  GAMF:=GAMF*((((G+0.191091101162)*X-
      0.233093736365)*X-0.422784335092)*X+
      1)*X*(1+X);
  END;
  IF K = -1 THEN EXIT;
  GAMF:=1/GAMF
END {GAMF}.
```

Процедура-функция GAMF была получена путем некоторой переработки и перевода программы вычисления гамма-функции, приведенной в работе Янке и др. (1968), с языка Бейсик вначале на язык *FORTRAN* [Белашов, 1997], а затем на язык *PASCAL* и протестирована на IBM PC/AT-286 при x = 5, -2.5 для k = 1, -1, когда табличные значения $\Gamma(x)$ равны соответственно

$$\Gamma(5) = 24, \quad \Gamma(-2.5) = -0.945308720... \ .$$

Полученные при этом результаты (см. табл. 5.2) совпадают с точными до восьмого десятичного знака.

Таблица 5.2

x	$k = 1$	$k = -1$
5	24.00000000	0.04166667
-2.5	-0.94530872	-1.05785544

Другой алгоритм вычисления $\Gamma(x)$ и $1/\Gamma(x)$, основанный на использовании асимптотического разложения Стирлинга (5.3), реализован в процедуре-функции GAMF1, значения формальных параметров в которой те же, что и в процедуре-функции GAMF.

```
FUNCTION GAMF1(X: REAL;K: INTEGER) : REAL;
BEGIN
  GAMF1:=1;
  IF (K<> -1) AND INT(X)=-ABS(X) THEN
  BEGIN
     WRITE ("ОШИБКА В ИСХОДНЫХ ДАННЫХ ');
     EXIT;
  END;
  WHILE  X<15 DO
  BEGIN
     GAMF1:=GAMF1/X;
     X:=X+1;
  END;
  GAMF1:=GAMF1*SQRT(2*3.141592655/X)
           *EXP(-X+X*LN(X));
  X:=1/X;
  GAMF1:=GAMF1*(1+X*(1/12+X*(1/288-
           X*(1339/51840+X*571/2488320))));
  IF K <>-1 THEN EXIT;
  GAMF1:=1/GAMF1
END {****GAMF1****}.
```

Процедура-функция GAMF1 была получена путем переработки и перевода Бейсик-программы вычисления гамма-функции, приведенной в работе Гринчишина и др. (1988), на язык *FORTRAN* [Белашов, 1997], а затем на язык *PASCAL* и протестирована на IBM PC/AT-286. Полученные при этом результаты совпадают с точностью 10^{-8} с результатами вычислений процедуры-функции GAMF, приведенными в табл. 5.2.

Замечание. Процедуру-функцию GAMF1 рекомендуется использовать для значений аргумента $x \geq 15$, однако ее можно с успехом применять и для малых x, так как в этом случае предусматривается приведение значения аргумента к x = 15 в соответствие с соотношением (5.1), на что, правда, расходуется дополнительное процессорное время.

Вычисление логарифмической производной гамма-функции $\psi(x)$ с использованием формул (5.5) и асимптотического разложения (5.6) реализовано в процедуре-функции GLOGD. Исключение полюсов функции $\psi(x)$ при x = 0, -1, -2, ... производится так же, как в процедурах-функциях GAMF и GAMF1**.**

Формальные параметры процедуры. *Входной*: x (тип ***double***) - значение аргумента x. *Выходной*: *glogd* (тип ***double***) - значение функции $\psi(x)$. Как и в процедурах-функциях GAMF и GAMF1, при x = 0, -1, -2, ... , когда $\psi(x)$ имеет полюсы, осуществляется выход к внешней процедуре обработки ошибки.

```
FUNCTION GLOGD(X: DOUBLE) : DOUBLE;
VAR PI : REAL;
BEGIN
  IF INT(X)=-ABS(X) THEN
  BEGIN
     WRITE ("ОШИБКА В ИСХОДНЫХ ДАННЫХ ');
     EXIT;
  END;
  IF X=1 THEN
  BEGIN
     GLOGD:=-0.5772156649;
     EXIT;
  END;
  GLOGD:=0;
  PI:=3.1415926535;
  IF X<-2 THEN
  BEGIN
     GLOGD:=-PI*COS(PI*X)/SIN(PI*X);
     X:=1-X
  END;
  WHILE  X<15 DO
  BEGIN
     GLOGD:=GLOGD-1/X;
     X:=X+1;
  END;
  X:=1/(X-1);
  GLOGD:=GLOGD-LN(X)+X/2;
```

```
    X:=X*2;
    GLOGD:=GLOGD-X*((X/21-0.1)*X+1)/12
  END {****GLOGD****}.
```

Процедура-функция GLOGD была получена путем переработки и перевода на язык *PASCAL* *FORTRAN*-программы вычисления $\psi(x)$ [Белашов, 1997] и протестирована на IBM PC/AT-286. При этом результаты, полученные для $x =$ = 0.5, 1,

$$\psi(0.5) = -1.96351003; \ \psi(1) = -0.577215665$$

совпадают с точностью 10^{-8} с результатами, приведенными в работе Гринчишина и др. (1988).

Замечание. Для процедуры GLOGD справедливо то же замечание, что и для процедуры GAMF1.

Вычисление неполной гамма-функции $\Gamma_x(a)$ с использованием ее разложения в степенной ряд (5.8) реализовано в процедуре-функции UGAMF. При этом вначале исключаются точки a = 0, -1, -2, ... , в которых функция не определена. Точность вычисления $\Gamma_x(a)$ задается при вызове функции.

Формальные параметры процедуры. *Входные*: *x, a* (тип ***real***) - амплитуда и аргумент функции $\Gamma_x(a)$; *eps* (тип ***real***) - точность вычисления. *Выходной*: *ugamf* (тип ***double***) - значение неполной гамма-функции с точностью *eps*.

```
FINCTION UGAMF (X,A,EPS : REAL) ; REAL;
VAR S, S1: REAL I : INTEGER;
LABEL A1;
BEGIN
IF INT(A)=-ABS(A) THEN
  BEGIN
     WRITE ("ОШИБКА В ИСХОДНЫХ ДАННЫХ ');
     EXIT;
  END;
  UGAMF:=1/A; I:=1;   S:=1;
  REPEAT
    S:=-S*X/I;       S1:=S/(A+I);
     I:=I+1;         UGAMF:=UGAMF+S1;
  UNTIL  ABS(S1) <= EPS*ABS(UGAMF);
  UGAMF:=UGAMF*X A
END { *** UGAMF *** }.
```

Процедура-функция UGAMF получена путем переработки и перевода на язык *PASCAL* программы вычисления неполной гамма-функции $\Gamma_x(a)$ [Белашов, 1997] и протестирована на IBM PC/AT-286 для тех же значений x и a. Результат вычисления $\Gamma_2(3)$ = 0.646647167 при задававшейся точности ε = 1e-8 совпадает с контрольным с точностью до восьми десятичных знаков.

Вычисление отношения неполной бета-функции $I_x(p,q)$ (5.9) при $x = z \equiv$ Re z, $p \equiv$ Re p, $q \equiv$ Re q реализовано в процедуре-функции UBETF на основании разложения $I_x(p,q)$ в степенной ряд, при этом в целях оптимизации процесса сходимости ряда при $x > 0.5$ используется соотношение симметрии (5.10). В процедуре предварительно проверяется соответствие вводимых значений амплитуды и аргументов областям их определения [см. выражение (5.7)]; в случае если неравенства (5.7) не удовлетворяются, осуществляется выход из процедуры к внешней программе обработки ошибки. Точность вычисления задается при вызове процедуры-функции.

Формальные параметры процедуры. *Входные*: *x, p, q* (тип ***real***) - амплитуда и аргументы функции $I_x(p,q)$; *eps* (тип ***real***) - задаваемая точность. *Выходной*: *ubetf* (тип ***double***) - вычисленное с точностью *eps* значение отношения неполной бета-функции $I_x(p,q)$.

```
FUNCTION UBETF (X,P,Q,EPS : REAL ) : REAL;
VAR W,T,T1,S1,S2,R,R1,Q1,I,U : REAL ;
LABEL A2, A3, A4, A5, A6;
{*** ??????? ???????????? ?????????-??????? G,
      ???????????? ? ???????? ?????????-???????
      UBETF, ???????????? ? ????? BEG; ***}
  FUNCTION G(U: REAL) : REAL;
  VAR GL : REAL;
  BEGIN     G:=1;
     WHILE  U>1 DO
     BEGIN
        U:=U-1;    G:=G/U;
     END;
    G1:=(((((((((-0.00000018122*U+0.000001328554)*U
          -0.000002625721)*U-0.000017527917)*U+
           0.000145624324)*U-0.000360851496)*U-
           0.000804341335)*U+0.008023278113)*U
           -0.017645242118)*U-0.024552490887)*U;
     G:=G*((((G1+0.191091101162)*U-
           0.233093736365)*U-0.422784335092)*U+
           1)*U*(1+U);
     G:=1/G
  END { *** G *** };

{ **** ?????? ???????? ????????? *** }
BEGIN
  IF X>1 OR X<0 OR P<0 OR Q<0 THEN
  BEGIN
     WRITE ("ОШИБКА В ИСХОДНЫХ ДАННЫХ ');
     EXIT;
  END;
  IF X=0 OR X=1 THEN
  BEGIN
     UBETF:=X;
     EXIT;
  END;
  W:=0;
  IF X>0.5 THEN    BEGIN
     T:=P;
     P:=Q;      Q:=T;
     X:=X-1;    W:=1
  END;
  S2:=0;
```

```
R:=1;
T:=1-X;
Q1:=Q;
I:=Q;.
REPEAT
    DEC (I);
    IF I>0 THEN
    BEGIN
        Q1:=I;
        R:=R*(Q1+1)/T/(P+Q1);
        S2:=S2+R;
    END
UNTIL I = 0;
R:=1;
S1:=1;
I:=0;
REPEAT
    I:=I+1;
    IF R>=EPS*S1 THEN
    BEGIN
        R:=R*X*(I-Q1)*(P+I-1)/I/(P+I);
        S1:=S1+R;
    END;
UNTIL R >= EPS*S1;
U:=Q1;
T:=G(U);
T1:=T;
U:=Q1+P;
R:=G(U);
R1:=R;
I:=Q1;
 WHILE I<= Q-0.5 DO
BEGIN
    T1:=T1*I;
    R1:=R1*(I+P);
    INC (I);
END;
T:=X P*(S1*R/P/T+S2*R1*(1-X) Q/Q/T1);
U:=P;
T:=T/G(U);
IF W=1 THEN `
BEGIN
    UBETF:=1-T;
    EXIT
END;
UBETF:=T
END { *** UBETF *** }.
```

Процедура-функция UBETF получена путем переработки и перевода вначале на язык *FORTRAN* [Белашов, 1997], а затем на язык *PASCAL* программы вычисления $I_x(p,q)$, опубликованной в работе Гринчишина и др. (1988), и протестирована на IBM PC/AT-286 для тех же, что и в указанной последней работе, значений *x, p* и *q*. Результаты вычислений функций $I_{0.7}(0.5;\ 0.5) = 0.63098988$, $I_{0.2}(2;1.5) = =0.06979572$ при заданной точности $\varepsilon = 1e\text{-}8$ с точностью до восьми десятичных знаков совпадают с полученными в работах Белашова (1997), Гринчишина и др. (1988).

§ 2. НЕКОТОРЫЕ ИНТЕГРАЛЬНЫЕ ФУНКЦИИ

Интегральная показательная функция $E_k(z)$ определяется интегралом [Справочник ..., 1979]

$$E_k(z) = \int_1^\infty \frac{e^{-zt}}{t^k}\,dt,\quad k = 0, 1, 2, ...;\ z = x + iy,\ x > 0\,.$$

Наряду с функцией $E_k(z)$ часто используется связанная с ней комплексная интегральная функция

$$W_k(z) = U + iV = z^k e^z E_k(z), \qquad (5.11)$$

представляемая в виде непрерывной дроби [Справочник ..., 1979]

$$W_k(z) = \cfrac{z}{z + \cfrac{k}{1 + \cfrac{1}{z + \cfrac{k+1}{1 + \cfrac{2}{z+\ldots}}}}},$$

вычислить которую можно итерационным методом в соответствии с формулами [Библиотека ..., 1975]

$$W_n = W_{n-1} + \left(\frac{z}{z + mD_{n-1}} - 1\right) R_{n-1}\ ;$$

$$W_n = U_n + iV_n\ ;\qquad R_n = a_n + ib_n\ ;$$

$$D_n = c_n + id_n\ \ ;\quad C_1 = R_1 = D_1 = 1\ ;$$

$$m = \begin{cases} k - 1 + (n-2)/2\ ; & n - \text{even}\ ; \\ (n-2)/2\ ; & n - \text{odd}. \end{cases}$$

Итерационный процесс завершается, когда выполняется $|W_n - W_{n-1}| \le \varepsilon$, где ε - заданная точность вычислений.

Данный алгоритм в соответствии с областью определения *z* позволяет получить результат на вещественной полуплоскости Re $z > 0$, однако, как указывается в работе Гринчишина и др. (1988), сходимость его очень медленная для $|z| < 0.05$ и в области $|y| < 2$, $x < 0$. Алгоритм реализован в процедуре WKZ, полученной путем переработки и перевода вначале на язык *FORTRAN* [Белашов, 1997], а затем на язык *PASCAL* *ALGOL*-программы вычисления $W_k(z)$ [Библиотека ..., 1975].

Формальные параметры процедуры. *Входные*: *k* (тип ***real***) - показатель *k* в формуле (5.11); *x, y* (тип ***real***) - действительная и мнимая части аргумента $z = x + iy$; *eps* (тип

real) - задаваемая точность. *Выходной*: *wkz* (тип ***comp***) - функция $W_n = u + iv$.

```
FUNCTION WKZ(K:INTEGER; X,Y,EPS:REAL) : COMPEX;
VAR A,B,C,D,W,M,P,Q: REAL; R,N : INTEGER;
BEGIN
  EPS:=EPS*EPS;
  A:=1.;
  C:=1.;
  U:=1;
  V:=0.0;
  D:=0.0;
  B:=0;
  N:=1;
  W:=K-1;
  REPEAT
     N:=N+1;
     R:=INT(N/2);
     M:=R;
     IF R*2=N THEN M:=R+W;
     P:=X+M*C;
     Q:=Y+M*D;
     M:=P*P+Q*Q;
     C:=(X*P+Y*Q)/M;
     D:=(Y*P-X*Q)/M;
     P:=C-1;
     Q:=A;
     A:=A*P-B*D;
     B:=Q*D+P*B;
     U:=U+A;
     V:=V+B;
  UNTIL (A*A+B*B)/(U*U+V*V) <= EPS;
END {****WKS****}.
```

Процедура WKZ была протестирована на машине IBM PC/AT-386 для $k = 1$, $z = 1 + i$ и $k = 2$, $z = 4$; $eps = 1e\text{-}7$. Полученные при этом результаты

$$W_1(1+i) = 0.673321227 + i\,0.147863858,$$

$$W_2(4) = 0.698469604$$

с точностью до шести десятичных цифр совпали с табличными [Библиотека ..., 1975].

Вещественная интегральная показательная функция $E_1(x)$, являющаяся частным случаем $E_k(z)$ при $k = 1$, $x = z$ ⏲ ⏲ Re z, определяется интегралом [Корн, Корн, 1984]

$$E_1(x) = -Ei(-x) = \int_x^\infty \frac{e^{-t}}{t}\,dt$$

и может быть аппроксимирована степенными рядами следующим образом [Библиотека ..., 1975]:

а) при $0 < x < 1$

$$E_1(x) = -\ln(x) - c_0 + c_1 x + \ldots + c_5 x + \varepsilon_1(x)\ ; \quad (5.12)$$

б) при $x \ge 1$

$$E_1(x) = \frac{e^{-x}}{x}\left[\frac{a_0 + a_1 x + a_2 x^2 + a_3 x^3 + x^4}{b_0 + b_1 x + b_2 x^2 + b_3 x^3 + x^4} + \varepsilon_2(x)\right], \quad (5.13)$$

где $|\varepsilon_1| < 2\cdot10^{-7}$, $|\varepsilon_2| < 2\cdot10^{-8}$, а значения коэффициентов представлены в табл. 5.3.

Таблица 5.3

i	a_i	b_i	c_i
0	0.2677737343	3.9584969228	0.57721566
1	8.6347608925	21.0996530827	0.99999193
2	18.0590169730	25.6329561486	-0.24991055
3	8.5733287401	9.5733223454	0.05519968
4			-9.76004e-3
5			1.07857e-3

С помощью процедуры-функции E1X вычисляются значения вещественной интегральной функции $E_1(x)$ в соответствии с формулами (5.12), (5.13).

Формальные параметры процедуры. *Входной*: x (тип **real**) - аргумент функции. *Выходной*: идентификатор функции $e1x$ (тип ***double***) - значение функции $E_1(x)$.

```
FUNCTION E1X(X:REAL) : DOUBLE;
BEGIN
  IF X>=1 THEN
  BEGIN
     E1X:=(((X+8.5733287401)*X+18.059016973)*X
          +8.6347608925)*X +0.2677737343;
     E1X:=EXP(-X)*E1X/X/((((X+9.5733223454)*X
          +25.6329561486)*X
          +21.0996530827)*X+3.9584969228);
     EXIT;
  END
  ELSE
     E1X:=-LN(X)-0.57721566+((((0.00107857*X-
          0.00976004)*X+0.05519968)*X-
          0.24991055)*X+0.99999193)*X
END {***E1X***}.
```

При тестировании процедуры-функции E1X на IBM PC/AT-286 для $x = 0.59$, 10 были получены следующие результаты (погрешность вычисления указана по отношению к табличным значениям [Библиотека ..., 1975]):

$$E_1(0.59) = 0.463649765 + 8.4e\text{-}8;$$

$$E_1(10) = 4.15696901\text{e-}6 + 2.0e\text{-}15,$$

что совпадает по точности с результатами, полученными в работе [Гринчишин и др., 1988] при расчетах по Бейсик-программе аналогичного назначения для тех же значений x.

Интегральный синус Si(x) действительного аргумента

$$\mathrm{Si}(x) = \int_0^x \frac{\sin t}{t}\,dt$$

может быть представлен в виде степенного ряда

$$\mathrm{Si}(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)(2n+1)!}\ , \quad (5.14)$$

а при больших $x \ge 1$ аппроксимирован по формуле из Справочника ... [1979]

$$\mathrm{Si}(x) = \pi/2 - f(x)\cos x - g(x)\sin x \qquad (5.15)$$

с рациональными коэффициентными функциями

$$f(x) = \frac{1}{x} \times \frac{x^8 + a_1 x^6 + a_2 x^4 + a_3 x^2 + a_4}{x^8 + b_1 x^6 + b_2 x^4 + b_3 x^2 + b_4} + \varepsilon_1(x); \qquad (5.16)$$

$$f(x) = \frac{1}{x} \times \frac{x^8 + c_1 x^6 + c_2 x^4 + c_3 x^2 + c_4}{x^8 + d_1 x^6 + d_2 x^4 + d_3 x^2 + d_4} + \varepsilon_2(x); \qquad (5.17)$$

где $|\varepsilon_1| < 5\cdot10^{-7}$, $|\varepsilon_2| < 3\cdot10^{-7}$, а значения коэффициентов a_i, b_i, c_i, d_i ($i = 1, 2, 3, 4$) представлены в табл. 5.4.

Т а б л и ц а 5.4

i	a_i	b_i	c_i	d_i
1	38.027264	40.021433	42.242855	48.196927
2	265.187033	322.624911	302.757865	482.485984
3	335.677320	570.236280	352.018498	1114.978885
4	38.102495	157.105423	21.821899	449.690326

Интегральный косинус Ci(x) действительного аргумента [Корн, Корн, 1984]

$$\mathrm{Ci}(x) = C + \ln x + \int_0^x \frac{\cos t - 1}{t} dt$$

может быть также представлен в виде ряда

$$\mathrm{Ci}(x) = C + \ln x + \sum_{n=1}^{\infty} \frac{(-1)^n x^{2n}}{2n(2n)!}, \qquad (5.18)$$

а в области $x \ge 1$ аппроксимирован с помощью выражения [Справочник ..., 1979]

$$\mathrm{Ci}(x) = f(x)\sin x - g(x)\cos x, \qquad (5.19)$$

в котором функции $f(x)$ и $g(x)$ определяются по формулам (5.16), (5.17), в которых использованы коэффициенты, представленные в табл. 5.4.

Вычисление Si(x) и Ci(x) на основе разложений (5.14), (5.18) для $0 < x < 1$ (погрешность $|\varepsilon_2| < 3\cdot10^{-8}$) и (5.15), (5.19) для $x \ge 1$ (погрешность $\varepsilon < 8\cdot10^{-7}$) реализовано в приведенной процедуре-функции SCI.

Формальные параметры процедуры. *Входные*: x (тип ***real***) - значение аргумента ($x > 0$); k (тип ***integer***) - переключатель: при $k \ne 2$ вычисляется Si(x), при $k = 2$ - Ci(x). *Выходной параметр*: идентификатор функции *sci* (тип ***double***) - вычисленное значение функции.

```
FUNCTION  SCI ( X : DOUBLE; K : INTEGER) : DOUBLE;
VAR  SC, S, C : DOUBLE;
BEGIN
  X1:=X;
  X:=X X;
  IF X >= 1 THEN
  BEGIN
     SCI:=1/X1;
```

```
SCI:=SCI*((((X/100+0.38027264)*X+2.65187033)*X
          +3.3567732)*X
          +0.38102495)/((((X/100+0.40021433)*X+
          3.22624911)*X+5.7023628)*X+1.57105423);
    S:=SIN(X1);
    C:=COS(X1);
     IF K=2 THEN
    BEGIN
        SCI:=SCI*S;
        SC:=C;
    END
    ELSE
    BEGIN
        SCI:=1.570796326-SCI*C;
        SC:=S;
    END;
    SCI:=SCI-SC/X*((((X/100+0.42242855)*X+
        3.02757865)*X+3.52018498)*X
        +0.21821899)/((((X/100+0.48196927)*X

+4.82485984)*X+11.14978885)*X+4.49690326);
END
ELSE
IF K=2 THEN
BEGIN
    SCI:=0.577215664+LN(X1);
    SCI:=SCI+(((X/322560-2.3148148E-4)*X
        +0.010416666667)*X-0.25)*X;
END;
ELSE
 BEGIN
    SCI:=X1;
    SCI:=SCI*((((X/3265920-2.834467E- 5)*X+
        1.66666666E-3)*X-0.0555555556)*X+1)
END
END { ***** SCI **** }.
```

Процедура-функция SCI была получена путем существенной переработки и перевода вначале на язык *FORTRAN* [Белашов, 1997], а затем на язык *PASCAL* Бейсик-программ вычисления интегральных синуса и косинуса, опубликованных в работе Гринчишина и др. (1988), и в целях контроля протестирована для тех же значений аргумента. В результате было получено (погрешность указана по отношению к табличным значениям [Справочник ..., 1979])

$$Si(0.5) = 0.493107418 + 0.0e\text{-}9;$$

$$Si(10) = 1.65834785 - 2.6e\text{-}7;$$

$$Ci(0.5) = -0.177784079 + 2.0e\text{-}10;$$

$$Ci(10) = -0.045456288 - 1.5e\text{-}7,$$

что также с достаточной точностью совпадает с результатами, полученными в работе Гринчишина и др. (1988).

§ 3. НЕПОЛНЫЕ ЭЛЛИПТИЧЕСКИЕ ИНТЕГРАЛЫ I И II РОДА

Неполный эллиптический интеграл I рода в нормальной форме Лежандра определяется выражением [Корн, Корн, 1984; Янке и др., 1968]

$$K(k)=\int_0^1 \frac{dt}{\sqrt{(1-t^2)(1-k^2t^2)}}, \qquad 0\le k<1,$$

(k - модуль), которое может быть аппроксимировано многочленом

$$K(k)=a_0+a_1n+a_2n^2+a_3n^3+a_4n^4-(b_0+b_1n+ \\ +b_2n^2+b_3n^3+b_4n^4)\ln n+\varepsilon(k)\ , \qquad (5.20)$$

где $n = 1 - k^2$, $|\varepsilon(k)| \le 2\cdot10^{-8}$; a_i, b_i - значения коэффициентов, которые при $i = 0, 1, 2, 3, 4$ приведены в табл. 5.5.

Таблица 5.5

№ i	Значения коэффициентов	
	a_i	b_i
0	1.38629436112	0.5
1	0.09666344259	0.12498593597
2	0.03590092383	0.06880248576
3	0.03742563713	0.03328355346
4	0.01451196212	0.00441787012

Процедура-функция ELLIPT1 вычисления $K(k)$ построена на основании разложения (5.20).

Формальные параметры процедуры. *Входной*: k (тип ***real***) - модуль эллиптического интеграла $K(k)$. *Выходной*: *ellipt*1 (тип ***double***) - значение $K(k)$.

```
FUNCTION ELLIPT1(K : DOUBLE) : DOUBLE;
BEGIN
  K:=1.-K;
  ELLIPT1:=(((0.01451196212*K+0.03742563713)*K+
            0.03590092383)*K+0.09666344259)*K+
             1.38629436112-LN(K)*((((0.00441787012*
             K+0.03328355346)*K+
            0.06880248576)*K+0.12498593597)*K+0.5)
END.
```

Неполный эллиптический интеграл II рода в нормальной форме Лежандра определяется выражением [Корн, Корн, 1984; Янке и др., 1968]

$$E(k)=\int_0^1 \sqrt{\frac{1-k^2t^2}{1-t^2}}, \qquad 0\le k<1,$$

которое может быть аппроксимировано многочленом [Янке и др., 1968]

$$E(k)=1+a_1n+a_2n^2+a_3n^3+a_4n^4-(b_1n+ \\ +b_2n^2+b_3n^3+b_4n^4)\ln n+\varepsilon(k)\ , \qquad (5.21)$$

где $n = 1 - k^2$, $|\varepsilon(k)| \le 2\cdot10^{-8}$, a_i, b_i - значения коэффициентов, которые при $i = 0, 1, 2, 3, 4$ приведены в табл. 5.6.

Таблица 5.6

№ i	Значения коэффициентов	
	a_i	b_i
1	0.44325141463	0.24998368310
2	0.06260601220	0.09200180037
3	0.04757383546	0.04069697526
4	0.01736506451	0.00526449639

Вычисление $E(k)$ в процедуре-функции ELLIPT2 построено на основании разложения (5.21).

Формальные параметры процедуры. *Входной*: k (тип ***real***) - модуль эллиптического интеграла $E(k)$. *Выходной*: *ellipt2* (тип ***double***) - значение $E(k)$.

```
FUNCTION ELLIPT2(K : DOUBLE) : DOUBLE;
BEGIN
  K=1.-K;
  ELLIPT2=(((0.01736506451*K+0.04757383546)*K+
            0.06260601220)*K+0.44325141463)*K+1-
            LN(K)*K*(((0.00526449639*K+
            0.04069697526)*K+0.09200180037)*K+
            0.24998368310);
END.
```

Тестирование процедур ELLIPT1 и ELLIPT2 проводилось на IBM PC/AT-286 для разных значений модуля k, примеры результатов приведены в табл. 5.7.

Таблица 5.7

k	$K(k)$	$E(k)$
0	1.570796	1.570796
0.01	1.574746	1.563021
0.44	1.806328	1.277917
0.99	3.695638	1.004439
0.999999	8.287456	1

§ 4. ПОЛНЫЕ ЭЛЛИПТИЧЕСКИЕ ИНТЕГРАЛЫ I И II РОДА

Полные эллиптические интегралы I и II рода $K(\pi/2,k)$ и $E(\pi/2,k)$ выражены как

$$K(\pi/2,k)=\int_0^{\pi/2}(1-k^2\sin^2\phi)^{-1/2}d\phi\ ;$$

$$E(\pi/2,k)=\int_0^{\pi/2}(1-k^2\sin^2\phi)^{1/2}d\phi\ ,$$

где $k^2=1-\cos^2\alpha$, и могут быть вычислены с помощью арифметико-геометрического среднего. При этом вначале

задаются числа $a_0 = 1$, $b_0 = \cos\alpha$, $c_0 = \sin\alpha$, а затем по рекуррентным формулам вычисляются значения a_i и b_i:

$$a_i = (a_{i-1} + b_{i-1})/2, \qquad b_i = (a_{i-1}b_{i-1})^{1/2}, \qquad i = 1,\ldots, N.$$

Тогда на N-м шаге, когда с заданной относительной погрешностью ε выполняется равенство $a_N = b_N$,

$$K(\alpha) = \pi/(2a_N),$$

$$E(\alpha) = \left[1 - \tfrac{1}{2}(c_0^2 + 2c_1^2 + 4c_2^2 + \ldots + 2^N c_N^N)\right]K(\alpha),$$

где $c_i = (a_{i-1} - b_{i-1})/2$, $i = 1,\ldots, N$. С помощью процедуры ELLIPT3 можно вычислить значения интегралов $K(\pi/2, k)$ и $E(\pi/2, k)$ по приведенному алгоритму, предварительно выполнив проверку аргумента, значение которого должно лежать в диапазоне $0 < n = 1 - k^2 \le 1$. В случае если это условие не выполняется, осуществляется выход к программе обработки ошибки.

Формальные параметры процедуры. *Входной*: *n* (тип ***real***) - аргумент $n = 1 - k^2$ вычисляемых функций. *Выходные*: *k, e* (тип ***double***) - значения полных эллиптических интегралов соответственно I и II рода.

```
PROCEDURE  ELLIPT3(N: DOUBLE;VAR EPS,K,E: DOUBLE);
VAR A, B, T, S, C : DOUBLE;  I : INTEGER;
BEGIN
  IF(N>1) OR (N<=0) THEN EXIT;
  N1:=DBLE(N);
  A:=1.;
  I:=1;
  B:=DSQRT(N1);
  T:=1.-N1;
  S:=0.;
  REPEAT
    S:=S+T;
    C:=(A-B)/2.;
    I:=2*I;
    T:=(A+B)/2.;
    B:=SQRT(A*B);
    A:=T;
    T:=I*C*C;
  UNTIL (ABS(C) <= EPS*A)  OR  (T<=EPS*S);
  K:=3.14159265359D0/(A+B);
  S:=S+T;
  E:=K*(1.-S/2.);
END.
```

Процедура ELLIPT3 получена с помощью перевода на язык *PASCAL* *FORTRAN*-программы, представленной в работе Белашова (1997) и являющейся переводом с языка *ALGOL* алгоритма, опубликованного в Библиотеке алгоритмов [Библиотека ..., 1975]. Тестирование процедуры проводилось на машине IBM PC/AT-286 для различных значений аргумента, некоторые примеры результатов вычислений представлены в табл. 5.8. Полученные значения с необходимой точностью совпадают с табличными [Беляков и др., 1962].

Таблица 5.8

n	$K(\pi/2, k)$	$E(\pi/2, k)$
0.1	2.578092	1.104775
0.6	1.777519	1.399392
1	1.570796	1.570796

Отметим, что наряду с эллиптическими функциями эллиптические интегралы находят важное применение в различных областях анализа, геометрии и физики, в частности в задачах механики, астрономии и геодезии.

§ 5. ФУНКЦИИ БЕССЕЛЯ ЦЕЛОГО ПОРЯДКА

Функции Бесселя есть решения дифференциального уравнения

$$z^2 w'' + zw' + (z^2 - \nu^2)w = 0, \qquad z = x + iy, \qquad (5.22)$$

при этом *функцией Бесселя I рода* называется такое решение уравнения (5.22), которое для произвольного порядка ν и аргумента z имеет представление

$$J_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{k=0}^{\infty} \frac{(-1)^k (z/2)^{2k}}{k!\,\Gamma(k + \nu + 1)}, \qquad (5.23)$$

функцией Бесселя II рода [*функцией Вебера* $Y_\nu(z)$ или *Неймана* $N_\nu(z)$] называется такая функция, при которой

$$N_\nu(z) = \left[J_\nu(z)\cos\pi z - J_{-\nu}(z)\right]/\sin\pi z, \qquad (5.24)$$

а *функциями Бесселя III и IV рода (функциями Ханкеля)* называются функции соответственно

$$H_\nu^{(1)}(z) = J_\nu(z) + iN_\nu(z) \text{ и } H_\nu^{(2)}(z) = J_\nu(z) - iN_\nu(z). \qquad (5.25)$$

Функции (5.22) - (5.25) являются аналитическими функциями z во всей плоскости, разрезанной вдоль отрицательной части действительной оси. При $\nu = \pm n$ функция $J_\nu(z)$ является целой функцией аргумента и не имеет особых точек. Отметим, что все функции Бесселя удовлетворяют рекуррентному соотношению [Справочник ..., 1979]

$$B_{\nu+1}(z) = (2\nu/z)B_\nu(z) - B_{\nu-1}(z). \qquad (5.26)$$

На основании введенных определений построены процедуры, с помощью которых вычисляются $J_\nu(z)$, $Y_\nu(x)$, $H_\nu^{(1)}(x)$ и $H_\nu^{(2)}(x)$ для любых целых ν и вещественных x.

В процедуре-функции JNX вычисления производятся с использованием разложения в ряд выражения (5.23), при этом точность задается пользователем.

Формальные параметры процедуры. *Входные*: *n* (тип ***integer***) - порядок ν функции; *x* (тип ***real***) - значение аргу-

мента; *eps* (тип ***real***) - задаваемая точность (при этом абсолютная точность $\varepsilon < eps \cdot (x/2)^n / n!$ [Гринчишин, 1988]). *Выходной*: *jnx* (идентификатор процедуры-функции, тип ***double***) - значение $J_\nu(x)$.

```
FUNCTION JNX(N:INTEGER;X,EPS:DOUBLE):DOUBLE;
VAR T,S,X1: DOUBLE;
BEGIN
  JNX:=1.;
  IF(X<>0) OR (N<>0) THEN
  BEGIN
     X1:=DBLE(0.5*X);
     N1:=ABS(N);
     IF(N <> 0) THEN
      FOR I:=1 TO N1 DO
         JNX:=JNX*X1/I;
  END;
  X1:=X1*X1;
  T:=1.;
  I:=1;
  S:=1.;
  REPEAT
     T:=-T*X1/(I*(I+N1));
     S:=S+T;
     INC (I);
  UNTIL ABS(T) > EPS;
  IF (N < 0) THEN
     IF N MOD 2 = 1 THEN
        JNX := -JNX;
  JNX:=N1*JNX;
  JNX:=JNX*S;
END.
```

Процедура-функция JNX была получена путем переработки и перевода программы вычисления $J_\nu(x)$, приведенной в работе Гринчишина и др. (1988), с языка Бейсик на язык *FORTRAN* [Белашов, 1997], а затем на язык *PASCAL* и протестирована на IBM PC/AT-286 при следующих значениях входных параметров: $n = 0$; $x = \pm 5$; $n = \pm 1$; $x = 1.4$ и $n = 2$, $x = 1.4$, $eps = 1e\text{-}8$. Полученные при этом результаты

$$J_0(5) = -0.1775967712, \quad J_1(1.4) = 0.5419477138,$$
$$J_0(-5) = 0.1775967712, \quad J_{-1}(1.4) = -0.5419477138,$$
$$J_2(1.4) = 0.2073558994, \quad J_{-2}(1.4) = 0.2073558994$$

совпадают с табличными [Справочник..., 1979] вплоть до девятого десятичного знака.

В случаях когда $\nu = 0$, 1, для вычисления функции Бесселя I рода вместо формулы (5.23) может быть использовано ее разложение в ряд [Справочник ..., 1979]:

а) при $|x| \le 3$

$$x^{-\nu} J_\nu(x) = 2^{-\nu} + \sum_{i=1}^{6} a_i y^{2i} + \varepsilon(x), \qquad y = x/3; \quad (5.27)$$

б) при $x > 3$

$$x^{1/2} J_\nu(x) = \cos\Theta \times S;$$
$$\Theta = x + \sum_{i=0}^{6} b_i y^{-i} + \varepsilon_1(x); \qquad S = \sum_{i=0}^{6} c_i y^{-i} + \varepsilon_2(x), \quad (5.28)$$

где коэффициенты a_i, b_i, c_i при $\nu = 0$ и $\nu = 1$ выбирают соответственно из табл. 5.9а и 5.9б

для $\nu = 0$

$$|\varepsilon(x)| < 5\cdot10^{-8}, \ |\varepsilon_1(x)| < 7\cdot10^{-8}, \ |\varepsilon_2(x)| < 5\cdot10^{-8};$$

для $\nu = 1$

$$|\varepsilon(x)| < 1.3\cdot10^{-8}, \ |\varepsilon_1(x)| < 9\cdot10^{-8}, \ |\varepsilon_2(x)| < 4\cdot10^{-8}.$$

Таблица 5.9а

i	a_i	b_i	c_i
0		-0.78539816	0.79788456
1	-2.2499997	-0.04166397	-7.7e-7
2	1.2656208	-0.00003954	-0.00552740
3	-0.3163866	0.00262573	-0.00009512
4	0.0444479	-0.00054125	0.00137237
5	-0.0039444	-0.00029333	-0.00072805
6	0.0002100	0.00013558	0.00014476

Таблица 5.9б

i	a_i	b_i	c_i
0		-2.35619449	0.79788456
1	-0.56249985	0.12499612	0.00000156
2	0.21093573	0.00005650	0.01659667
3	-0.03954289	-0.00637879	0.00017105
4	0.00443319	0.00074348	-0.00249511
5	-0.00031761	0.00079824	0.00113653
6	0.00001109	-0.00029166	-0.00020033

В приведенной далее процедуре-функции JNX01 для вычисления функции Бесселя $J_\nu(x)$ при $\nu = 0$; $\nu = 1$ используются формулы (5.27), (5.28). При этом значения коэффициентов a_i, b_i, c_i передаются в процедуру в виде массивов из вызывающей программы.

Формальные параметры процедуры. *Входные*: *n* (тип ***integer***) - значение порядка ν функции (0 или 1); *x* (тип ***real***) - значение аргумента; *a*[1:6], *b*[0:6], *c*[0:6] (тип ***double***) - значения коэффициентов разложений (5.27), (5.28). *Выходной*: *jnx*01 (идентификатор процедуры-функции, тип ***double***) - вычисленное значение функции Бесселя $J_0(x)$ или $J_1(x)$.

```
FUNCTION JNX01(N : INTEGER; X : DOUBLE;
          A,B,C : ARRAY OF DOUBLE) : DOUBLE;
VAR  Y,S1,S2 : DOUBLE;
BEGIN
  X1:=DOUBLE(X);    S1:=0.;
  IF (X <=3.) THEN
  BEGIN
     Y:=X1*X1/9.;
     FOR I:=1 TO 6 DO
        S1:=S1+A(I)*EXP (I*LN(Y));
```

```
      JNX01:=EXP (N*LN(X1))*(EXP (N*LN(0.5))+S);
      EXIT;
   END;
   S2:=0.;
   Y:=3./X1;
   FOR I:=0 TO 6 DO
   BEGIN
      S1:=S1+B(I)*EXP (I*LN(Y));
      S2:=S2+C(I)* EXP (I*LN(Y));
   END;
   JNX01=COS(X1+S1)*S2/SQRT(X1);
END.
```

Процедура-функция JNX01 тестировалась на IBM PC/AT-286 при n = 0, 1; x = 2.9, 4, 15. Полученные при этом результаты представлены в табл. 5.10, значения погрешности указаны в сравнении с табличными величинами $J_0(x)$ и $J_1(x)$ [Справочник ..., 1979].

Таблица 5.10

x	$n = 0$	$n = 1$
2.9	-0.2243115953+4.96e-8	0.3754275162+3.44e-8
4	-0.3971498118 -1.20e-8	-0.0660433224-0.56e-8
15	-0.0142244714 -0.14e-8	0.2051040490-1.04e-8

Вычисление функции $J_\nu(x)$ для любого целого ν реализовано наряду с процедурой *jnx* в процедуре JANX, в которой используются внешние процедуры-функции JNX и JNX01. При этом вначале рассчитывается $J_{|\nu|}(|x|)$, а далее используются соотношения

$$J_{-\nu}(x) = (-1)^\nu J_\nu(x); \qquad J_\nu(-x) = (-1)^\nu J_\nu(x).$$

Формальные параметры процедуры. *Входные*: n (тип ***integer***) - порядок ν функции; x (тип ***real***) - значение аргумента; *eps* (тип ***real***) - задаваемая точность (при $|x| \le 3$, при $|x| > 3$ значение не используется); a[1:6], b[0:6], c[0:6] (тип ***double***) - значения коэффициентов разложений (5.27), (5.28) (используются при $|x| > 3$). *Выходной*: *janx* (идентификатор процедуры-функции, тип ***double***) - значение $J_\nu(x)$.

```
FUNCTION JANX (N : INTEGER; X,EPS : DOUBLE;
               A,B,C : ARRAY OF DOUBLE) : DOUBLE;
VAR  JNX,JNX01 : DOUBLE;
BEGIN
 { ***????????????? ??????? ??????? JNX ? JNX01 *** }
  JANX:=1.;
  IF (X =0)  AND (N=0) THEN EXIT;
  NA:=ABS(N);
  XA:=ABS(X);
  IF (XA > 3) THEN
     JANX:=JNX01(NA,XA,A,B,C)
  ELSE
     IF (X =0) THEN
     BEGIN
        JANX:=0.;
        EXIT;
     END
     ELSE
        JANX:=JNX(NA,XA,EPS)
  END;
  IF(N=0) THEN EXIT;
  IF(N < 0) THEN
     IF (NA MOD 2 = 1) THEN  JANX := -JANX;
  IF(X < 0.) THEN
     IF (NA MOD 2 = 1) THEN  JANX := -JANX;
END.
```

Тестирование процедуры-функции JANX на IBM PC/AT-386 при тех же значениях n и x, что и для процедур JNX и JNX01, дало аналогичные значения $J_\nu(x)$; результаты при отрицательных значениях порядка и аргумента представлены в табл. 5.11 (ср. с табл. 5.10).

Таблица 5.11

x	n = -1	n = 1
2.9	-0.3754275162-3.44e-8	0.3754275162+3.44e-8
-2.9	0.3754275162+3.44e-8	-0.3754275162-3.44e-8
4	0.0660433224+0.56e-8	-0.0660433224-0.56e-8
-4	-0.0660433224-0.56e-8	0.0660433224+0.56e-8
15	-0.2051040490+1.04e-8	0.2051040490-1.04e-8
-15	0.2051040490-1.04e-8	-0.2051040490+1.04e-8

Вычисление функции Бесселя II рода для любого целого $\nu = n$ и вещественного аргумента x может быть выполнено на основании ее разложения [Справочник ..., 1979]

$$\left.\begin{aligned} N_n(x) = &-\frac{(x/2)^{-n}}{\pi}\sum_{k=0}^{n-1}\frac{(n-k-1)!}{k!}(x^2/4)^k + \\ &+ (2/\pi)\ln(x/2)J_n(x) - \\ &-\frac{(x/2)^n}{\pi}\sum_{k=0}^{\infty}\left[\psi(k+1)+\psi(k+n+1)\right]\frac{(-x^2/4)^k}{k!(n+k)!}; \\ \psi(m) = &-C + \sum_{l=1}^{m-1} 1/l, \qquad \psi(1) = -C, \end{aligned}\right\} \quad (5.29)$$

где C = 0.57721566490153 - постоянная Эйлера.

Такой алгоритм реализован в приведенной далее процедуре-функции *ynx*, использующей константы $2C$= =1.1544... и $1/\pi$ = 0.318... . При этом, если $n < 0$, вначале вычисляется функция $N_{|n|}(x)$, а затем используется соотношение $N_{-n}(x) = (-1)^n N_n(x)$.

Формальные параметры процедуры. *Входные*: n (тип ***integer***) - порядок n функции; x (тип ***real***) - значение аргумента. *Выходной*: *ynx* (идентификатор процедуры-функции, тип ***double***) - значение $N_n(x)$. При x = 0, когда функция Неймана (Вебера) не определена, осуществляется выход к программе обработки соответствующей ошибки.

```
FUNCTION YNX (N : INTEGER; X : DOUBLE) : DOUBLE;
```

```
VAR  X1,X2,A,R,B,S,D,P,T : DOUBLE;
BEGIN
   IF(X = 0.) THEN
         EXIT;
   YNX := 0.;
   NA := IABS(N);
   X1 := DOUBLE(X/2.);
   X2 := X1*X1;
   A := 1.;
   R := 1.;
   B := 0.;
   S := 0.;
   IF(N <>0)THEN
      FOR I :  := 1 TO NA DO
      BEGIN
         R := R*I;
         S := S+1./I;
      END;
   END;
   D := 0.;
   IF(N <>0) THEN
      D := R/NA
   R := 1./R;
   P := EXP (NA * LN(X2));
   T := LN (X2)+1.1544313298031D0;
    I := 0;
   WHILE (I > NA ) AND (B = YNX) DO
   BEGIN
      B := YNX;
      YNX := YNX+A*R*(T-S);
      IF (I < NA) THEN
         YNX := YNX-A*D/P;
      I1 := I+1;
      A := A*X2/I1;
      R := -R/(I1+NA);
      S := S+1./I1+1./(I1+NA);
      IF (I1 < NA) THEN
         D := D/(NA-I1);
      I := I1;
   END;
   P := EXP ( NA * LN(X1));
   YNX := 0.318309886184D0*YNX*P;
   IF(N < 0) THEN
      YNX := (-1)**NA*YNX;
END.
```

Процедура-функция YNX тестировалась на IBM PC/AT-386 при следующих значениях входных параметров: $n = 1$, $x = \pm 8$ и $n = \pm 7$, $x = 4$. Полученные при этом результаты

$N_1(8) = -0.1580554$, $N_1(-8) = 0.1580554$,

$N_7(4) = -3.706224$, $N_{-7}(4) = 3.706224$

совпадают с табличными [Справочник ..., 1979] с точностью до шести - восьми десятичных знаков.

Вычисление функции Вебера $Y_\nu(x)$ в частных случаях $\nu = n = 0, 1$ выполняется в процедуре-функции YNX01 на основании следующих полиномиальных аппроксимаций:

а) при $0 \le x \le 3$

$$x^n Y_n(x) = J_n(x)(2x^n/\pi)\ln(x/2) + \sum_{i=0}^{6} a_i y^{2i} + \varepsilon(x), \quad (5.30)$$

где $J_n(x)$ и y определяются выражениями (5.27), значения a_i представлены в табл. 5.12; $|\varepsilon(x)| < 1.4\cdot10^{-8}$ - при n=0 и $|\varepsilon(x)| << 11\cdot10^{-7}$ при $n = 1$;

б) при $x > 3$

$$x^{1/2} Y_n(x) = \sin\Theta \cdot S, \quad (5.31)$$

где θ и S определяются так же, как и в выражении (5.28).

Таблица 5.12

i	a_i	
	$n = 0$	$n = 1$
0	0.36746691	-0.6366198
1	0.60559366	0.2212091
2	-0.74350384	2.1682709
3	0.25300117	-1.3164827
4	-0.04261214	0.3123951
5	0.00427916	-0.0400976
6	-0.00024846	0.0027873

Формальные параметры процедуры. *Входные*: *n* (тип ***integer***) - значение порядка *n* функции (0 или 1); *x*1 (тип ***real***) - значение аргумента; *a*[0:6], *b*[0:6], *c*[0:6] (тип ***double***) - значения коэффициентов разложений (5.28), (5.30). *Выходной*: *ynx*01 (идентификатор процедуры-функции, тип ***double***) - вычисленное значение функции Вебера $Y_0(x)$ или $Y_1(x)$. В процедуре YNX01 используется внешняя процедура-функция JNX.

```
FUNCTION YNX01(N : INTEGER; X1 : DOUBLE;
      A,B,C : ARRAY OF DOUBLE) : DOUBLE;
VAR  X,PI,Y,S1,S2 : DOUBLE;
BEGIN
  {*** ???????????? ??????? ??????? JNX ***}
  PI := 3.14159265359D0;
  S1 := 0.;
  X := DOUBLE(X1);
  IF(X <=3)THEN
  BEGIN
     Y := (X/3.)* (X/3.);
     FOR I := 0 TO 6 DO
        S1 := S1+A(I)*EXP (I *LN(Y));
     YNX01 := S/X**N+2./PI*DLOG(X/2.)*JNX(N,X1,1E-8);
     EXIT;
  END;
  S2 := 0.;  Y := 3./X;
  FOR I := 0 TO 6 DO
  BEGIN
     S1 := S1+B(I)* EXP (I *LN(Y));
     S2 := S2+C(I)* EXP (I *LN(Y));
  END;
  YNX01 := SIN(X+S1)*S2/SQRT(X);
END.
```

Процедура-функция YNX01 тестировалась на IBM PC/AT-386 при n = 0, 1; x = 2.9, 4, 10. Полученные при этом результаты представлены в табл. 5.13, значения погрешности указаны в сравнении с табличными величинами $Y_0(x)$ и $Y_1(x)$ [Справочник ..., 1979].

Таблица 5.13

x	n = 0	n = 1
2.9	0.4079117580+1.12*e*-8	0.2959400312+2.34*e*-8
4	-0.0169407231-1.62*e*-8	0.3979257124-0.18*e*-8
10	0.0556711676+0.03*e*-8	0.2490154233+0.09*e*-8

Вычисление функций Ханкеля $H_\nu^{(1)}(z)$ и $H_\nu^{(2)}(z)$ (функций Бесселя III и IV рода) при $z \equiv \mathrm{Re}\, z = x$ и целом значении порядка $\nu = n$ реализовано в процедуре HANX12, в которой на основании формул (5.25) использованы разложение (5.23) функции Бесселя I рода и аппроксимация функции Неймана $N_n(x)$ многочленом вида (5.29). При вычислении $\mathrm{Re}[H_n(x)] \equiv J_n(x)$ в процедуре применяется внешняя функция JNX, а при вычислении $\mathrm{Im}[H_n(x)] \equiv Y_n(x)$ - функция YNX.

Формальные параметры процедуры. *Входные*: n (тип ***integer***) - порядок функции; x (тип ***double***) - значение аргумента; k (тип ***integer***) - задает знак мнимой части функции Ханкеля: при k = 1 вычисляется $H_n^{(1)}(x)$, при k = -1- $H_n^{(2)}(x)$; *eps* (тип ***double***) - задаваемая точность. *Выходной*: *hanx*12 (тип ***complex***) - идентификатор вычисленной функции Ханкеля. В случае если значение аргумента x = 0 (когда функция не определена), осуществляется выход к программе обработки ошибки.

```
FUNCTION HANX12 (N,K : INTEGER;
                 X,EPS : DOUBLE):DOUBLE;
VAR  RE,IM,JNX,YNX : DOUBLE;
BEGIN
  IF (X =0.)  THEN EXIT;
  RE  := JNX(N,X,EPS);
  IM  := K*YNX(N,X);
  HANX12  := COMPLEX(RE,IM);
END.
```

Альтернативный вариант вычисления функций Ханкеля реализован в более экономичной по времени процедуре HANX120, не требующей использования внешних функций, тело которой построено аналогично телу функции YNX. Назначение формальных параметров такое же, что и в функции HANX12, однако в отличие от нее процедура организована как подпрограмма, результатом работы которой являются передающиеся через формальные параметры значения действительной и мнимой частей вычисляемой функции, что иногда может оказаться более удобным. В процедуре используются константы: $2C$ = 1.1544... и $1/\pi$ = =0.318... .

```
PROCEDURE HANX120 (NN , K : INTEGER;
                   VAR X1,RE,IM : DOUBLE);
VAR RE,IM,X,X2,A,R,B,S,D,P,T : DOUBLE;
          X1 : REAL; NN,K,NNA,I,I1 : INTEGER;
BEGIN     IF(X1 = 0.) THEN          EXIT;
  RE  := 0.;   IM  := 0.;
  NNA  := ABS(NN);
  X  := DOUBLE(X1/2.);
  X2  := X*X;
  A  := 1.;   R  := 1.;
  B  := 0.;   S  := 0.;
  IF (NN <> 0) THEN
  FOR  I  := 1 TO NNA DO
  BEGIN       R  := R*I;
      S  := S+1./I;
  END;
  D  := 0.;
  IF(NN <>0) THEN
      D  := R/NNA;
  R  := 1./R;
  P  := EXP (NNA * LN(X2));
  T  := LN(X2)+1.1544313298031D0;
  I  := 0;
  WHILE (I <=NNA) AND (B <>IM) DO
  BEGIN       B  := IM;
      RE  := RE+A*R;
      IM  := IM+A*R*(T-S);
      IF(I < NNA) THEN
          IM  := IM-A*D/P;
      I1  := I+1;      A  := A*X2/I1;
      R  := -R/(I1+NNA);
      S  := S+1./I1+1./(I1+NNA);
      IF(I1.LT.NNA)D  := D/(NNA-I1)
      I  := I1
  END;
  P  := EXP ( NNA * LN(X));
  RE  := RE*P;
  IM  := 0.318309886184D0*IM*P;
  IF(NN <>0)THEN
  BEGIN       RE  := (-1)**NNA*RE;
     IM  := (-1)**NNA*IM;
  END;
  IF(K < 0) THEN     IM  := -IM;
END.
```

Тестирование процедур HANX12 и HANX120 на IBM PC/AT-386 показало, что их точностные характеристики примерно одинаковы (при задаваемом в процедуре HANX12 *eps* = 1*e*-8), но процедура HANX120 ра ботает несколько быстрее. Результаты расчетов для k = 1 приведены в табл. 5.14.

Таблица 5.14

n/x	HANX12	HANX120
1/8	0.2346363-i*0.1580554	0.2346331-i*0.1580554
1/-8	-0.2346363+i*0.1580554	0.2346331+i*0.1580554
7/4	1.517608e-2-i*3.706224	1.517607e-2-i*3.706224
-7/4	-1.517608e-2-i*3.706224	-1.517607e-2+i*3.706224

§ 6. МОДИФИЦИРОВАННЫЕ ФУНКЦИИ БЕССЕЛЯ

Полагая аргумент z функций Бесселя комплексным и изменяя его на $\pi/2$, получаем *модифицированные функции Бесселя* [Справочник ..., 1979]:

$$\left.\begin{aligned} I_\nu(z) &= e^{-i\pi\nu/2} J_\nu(ze^{i\pi/2}), \quad (-\pi < \arg z \le \pi/2); \\ I_\nu(z) &= e^{3i\pi\nu/2} J_\nu(ze^{-3i\pi/2}), \quad (\pi/2 < \arg z \le \pi); \end{aligned}\right\} \tag{5.32}$$

$$\left.\begin{aligned} K_\nu(z) &= \tfrac{1}{2} i\pi e^{i\pi\nu/2} H_\nu^{(1)}(ze^{i\pi/2}), \quad (-\pi < \arg z \le \pi/2); \\ K_\nu(z) &= -\tfrac{1}{2} i\pi e^{-i\pi\nu/2} H_\nu^{(2)}(ze^{-i\pi/2}), \quad (-\pi/2 < \arg z \le \pi), \end{aligned}\right\} \tag{5.33}$$

(последние также называются *модифицированными функциями Ханкеля*), которые являются решениями дифференциального уравнения

$$z^2 w'' + zw' - (z^2 + \nu^2) w = 0$$

и могут быть аппроксимированы рядами

$$I_\nu(z) = (z/2)^\nu \sum_{k=0}^{\infty} \frac{(z^2/4)^k}{k!\Gamma(\nu + k + 1)}; \tag{5.34}$$

$$\begin{aligned} K_\nu(z) = {} & \tfrac{1}{2}(z/2)^{-\nu} \sum_{k=0}^{\nu-1} \frac{(\nu - k - 1)!}{k!} (-z^2/4)^k + \\ & + (-1)^{\nu+1} \ln(z/2) I_\nu(z) + \\ & + \tfrac{1}{2}(-z/2)^\nu \sum_{k=0}^{\infty} \left[\psi(k+1) + \psi(k+\nu+1)\right] \frac{\left(z^2/4\right)^k}{k!(\nu+k)!}, \end{aligned} \tag{5.35}$$

где функция ψ определяется так же, как в формуле (5.29). Для функций $I_\nu(z)$ и $K_\nu(z)$ справедливы следующие соотношения симметрии:

$$I_{-n}(z) = I_n(z); \ (n \text{ - целое}), \ K_{-\nu}(z) = K_\nu(z). \tag{5.36}$$

В частных случаях ν=0, 1 функция $I_\nu(x)$ (при $x = z \equiv \mathrm{Re}\, z$) может быть представлена в виде многочленов [Справочник ..., 1979]:

а) при $|x| \le 3.75$

$$\begin{aligned} I_0 = {} & 1 + 3.5156229t^2 + 3.0899424t^4 + \\ & + 1.2067492t^6 + 0.2659732t^8 + 0.0360768t^{10} + \\ & + 0.0045813t^{12} + \varepsilon_1(x); \end{aligned} \tag{5.37}$$

$$\begin{aligned} x^{-1} I_1(x) = {} & 0.5 + 0.87890594t^2 + 0.51498869t^4 + \\ & + 0.15084934t^6 + 0.02658733t^8 + 0.0030152t^{10} + \\ & + 0.00032411t^{12} + \varepsilon_2(x), \end{aligned} \tag{5.38}$$

где $t = x/3.75$, $|\varepsilon_1(x)| < 1.6\cdot10^{-7}$; $|\varepsilon_2(x)| < 8\cdot10^{-9}$;

б) при $3.75 \le x \le \infty$

$$\begin{aligned} x^{1/2} e^{-x} I_0(x) = {} & 0.39894228 + 0.01328592t^{-1} + \\ & + 0.00225319t^{-2} - 0.00157565t^{-3} + 0.00916281t^{-4} - \\ & - 0.02057706t^{-5} + 0.02635537t^{-6} - 0.01647633t^{-7} + \\ & + 0.00392377t^{-8} + \varepsilon_3(x); \end{aligned} \tag{5.39}$$

$$\begin{aligned} x^{1/2} e^{-x} I_1(x) = {} & 0.39894228 - 0.03988024t^{-1} - \\ & - 0.00362018t^{-2} + 0.00163801t^{-3} - 0.01031555t^{-4} + \\ & + 0.02282967t^{-5} - 0.02895312t^{-6} + 0.01787654t^{-7} - \\ & - 0.00420059t^{-8} + \varepsilon_4(x), \end{aligned} \tag{5.40}$$

где $|\varepsilon_3(x)| < 1.9\cdot10^{-7}$; $|\varepsilon_4(x)| < 2.2\cdot10^{-7}$.

Отметим также, что функция $e^{-x} K_\nu(x)$ может быть представлена интегралом

$$e^{-x} K_\nu(x) = \int_0^\infty e^{x[1-\mathrm{ch}(t)]} \mathrm{ch}(\nu t)\, dt. \tag{5.41}$$

При повороте аргумента z в функциях Бесселя на $3\pi/4$ получим *функции Кельвина* $ber_\nu(z)$, $bei_\nu(z)$ и $ker_\nu(z)$, $kei_\nu(z)$, которые при $x = z \equiv \mathrm{Re}\, z \ge 0$ и действительном ν определяются из следующих соотношений:

$$\left.\begin{aligned} ber_\nu(x) + ibei_\nu(x) &= J_\nu(xe^{3i\pi/4}); \\ ker_\nu(x) + ikei_\nu(x) &= (i\pi/2) H_\nu^{(1)}(xe^{3i\pi/4}). \end{aligned}\right\} \tag{5.42}$$

При произвольном целом n функции Кельвина $ber_n(x)$ и $bei_n(x)$ могут быть представлены в виде следующих разложений [Справочник ..., 1979]:

$$\begin{aligned} ber_n(x) = {} & (x/2)^n \sum_{k=0}^{\infty} \frac{\cos(3\pi n/4 + k\pi/2)}{k!(n+k)!} (x^2/4)^k = \\ = {} & (x/2)^n \left[\frac{\cos(3\pi n/4)}{0!n!} - \frac{\sin(3\pi n/4)}{1!(n+1)!} \cdot \frac{x^2}{4} - \right. \\ & \left. - \frac{\cos(3\pi n/4)}{2!(n+2)!} \cdot \left(\frac{x^2}{4}\right)^2 + \frac{\sin(3\pi n/4)}{3!(n+3)!} \cdot \left(\frac{x^2}{4}\right)^3 - \ldots \right]. \end{aligned} \tag{5.43}$$

$$bei_n(x) = (x/2)^n \sum_{k=0}^{\infty} \frac{\sin(3\pi n/4 + k\pi/2)}{k!(n+k)!}(x^2/4)^k =$$

$$= (x/2)^n \left[\frac{\sin(3\pi n/4)}{0!n!} + \frac{\cos(3\pi n/4)}{1!(n+1)!} \cdot \frac{x^2}{4} - \right. \quad (5.44)$$

$$\left. - \frac{\sin(3\pi n/4)}{2!(n+2)!} \cdot \left(\frac{x^2}{4}\right)^2 - \frac{\cos(3\pi n/4)}{3!(n+3)!} \cdot \left(\frac{x^2}{4}\right)^3 + \ldots \right].$$

Если же порядок функций Кельвина v=0 (в этом случае индекс v обычно опускается), то имеют место следующие разложения [Справочник ..., 1979]:

$$\left.\begin{aligned} ber(x) &= 1 - \frac{(x^2/4)^2}{2!} + \frac{(x^2/4)^4}{(4!)^2} - \ldots ; \\ bei(x) &= \frac{x^2}{4}\left[1 - \frac{(x^2/4)^2}{(3!)^2} + \frac{(x^2/4)^4}{(5!)^2} - \ldots \right]; \end{aligned}\right\} \quad (5.45)$$

$$ker(x) = -ber(x)l(x/2) + (\pi/4)bei(x) +$$

$$+ \sum_{k=0}^{\infty} (-1)^k \frac{\psi(2k+1)}{((2k)!)^2} \cdot \left(\frac{x^2}{4}\right)^{2k} =$$

$$= \sum_{k=0}^{\infty} (-1)^k \frac{(x^2/4)^{2k}}{((2k)!)^2} \times \quad (5.46)$$

$$\times \left[-\ln(x/2) + (\pi/4)\frac{x^2/4}{(2k+1)^2} + \psi(2k+1) \right];$$

$$kei(x) = -bei(x)l(x/2) + (\pi/4)ber(x) +$$

$$+ \sum_{k=0}^{\infty} (-1)^k \frac{\psi(2k+2)}{((2k+1)!)^2} \cdot \left(\frac{x^2}{4}\right)^{2k+1} =$$

$$= \sum_{k=0}^{\infty} (-1)^k \frac{(x^2/4)^{2k}}{((2k)!)^2} \times \quad (5.47)$$

$$\times \left[-(\pi/4) + (x^2/4)\frac{\psi(2k+2) - \ln(x/2)}{(2k+1)^2} \right].$$

При отрицательных значениях x и $v = n$ (где n - целое) используются следующие соотношения симметрии:

$$ber_n(-x) = (-1)^n ber_n(x); \quad ber_{-n}(x) = (-1)^n ber_n(x), \quad (5.48)$$

которые справедливы и для функций $bei_n(x)$, $ker_n(x)$, $kei_n(x)$.

По приведенным далее процедурам осуществляется вычисление модифицированных функций Бесселя в соответствии с рассмотренными разложениями (отметим также, что значения модифицированных функций Бесселя могут быть получены с помощью формул (5.32), (5.33) и (5.42) при использовании для вычисления функций $J_v(z)$, $H_v^{(1)}(z)$ и $H_v^{(2)}(z)$ соответствующих процедур из предыдущего параграфа).

С помощью процедуры-функции INX вычисляется значение $I_v(x)$ для $v=n$ (n-целое) и любого вещественного x = = = $z \equiv$ Re z на основании разложения (5.34). При этом в случае отрицательного n рассчитывается значение функции $I_{|n|}(x)$, которое с учетом соотношения (5.36) равно искомому значению $I_n(x)$. Поскольку значение $|n|$ является натуральным числом, при вычислении Г-функции в формуле (5.34) используется равенство Γ(η) = (η - 1)!. Вычисления продолжаются до достижения задаваемой точности ε, когда оказывается выполненным неравенство $|S_m - S_{m-1}| \le \varepsilon$, где S_m - значение суммы в формуле (5.34) при $k = m$.

Формальные параметры процедуры. *Входные*: *n* (тип ***integer***) - значение порядка; *x* (тип ***real***) - значение аргумента функции; *eps* (тип ***real***) - задаваемая точность. *Выходной*: *inx* (идентификатор процедуры-функции, тип ***double***).

```
FUNCTION INX( VAR N INTEGER; X,EPS : DOUBLE): DOUBLE;
VAR  X1,X2,S : DOUBLE;
BEGIN
  INX  := 1.;
  IF(N <> 0) OR (X <> 0) THEN
  BEGIN      N  := ABS(N);
     X1  := DOUBLE(X);
     X2  := EXP ( N * LN((X1/2.)));
     X1  := X1*X1/4.;
     FOR I  := 1 TO N DO  INX  := INX/I;
     I  := 1;
     S  := INX;
     REPEAT
        S  := S*X1/I/(I+N);
        INC (I);
        INC (INX, S);
     UNTIL  (ABS(S) >= EPS);
     INX  := INX*X2;
  END;
END .
```

Процедура-функция INX тестировалась на IBM PC/AT-386 для ряда значений порядка n и аргумента x, некоторые результаты вычислений при ε =1e-10^{-8} приведены в табл. 5.15. Сравнение полученных значений с табличными [Справочник..., 1979] дает погрешность, не превышающую задаваемой точности вычислений.

Таблица 5.15

n	x		
	2	4	5
± 3	0.212739960654	3.337275851226	10.331150908245
± 4	0.050728570426	1.416275740233	5.108234887217
± 5	9.82567932297e-3	0.504724367966	2.157974595789

С помощью следующих процедур-функций I0X и I1X, структура которых аналогична, вычисляются значения соответственно $I_0(x)$ и $I_1(x)$ на основании разложений (5.37) - (5.40). Точность вычислений определяется погрешностью соответствующих разложений.

Формальные параметры процедур. *Входной*: *x* (тип ***real***) - значение аргумента. *Выходной*: *i*0*x* или *i*1*x* (идентификатор процедуры-функции, тип ***double***).

```
FUNCTION I0X(X1 : REAL) : DOUBLE;
VAR X,T : DOUBLE;
BEGIN
  X := DOUBLE(X1);
  IF X <=3.75 THEN
  BEGIN
     X := (X/3.75)*(X/3.75);
     I0X := 1.+X*(3.5156229D0+X*(3.0899424D0+
            X*(1.2067492D0+X*(0.26597332D0+
            X*(0.0360768D0+X*0.0045813D0)))));
  END
  ELSE
  BEGIN
     T := 3.75/X;
     I0X := (((((((0.00392377D0*T-0.01647633D0)*T+
             0.02635537D0)*T-0.02057706D0)*T+
            0.00916281D0)*T- 0.00157565D0)*T+
            225319D0)*T+0.01328592D0)*T+
            0.39894228D0)*EXP(X)/SQRT(X);
   END;
END.

FUNCTION I1X(X1 : REAL) : DOUBLE;
VAR X,T : DOUBLE;
BEGIN
  X := DOUBLE(X1);
  IF X <=3.75 THEN
  BEGIN
     T := (X/3.75)*(X/3.75);
     I1X := (0.5+T*(0.87890594D0+T*(0.51498869D0+
            T*(0.15084934D0+T*(0.02658733D0+
            T*(0.00301532D0+T*0.00032411D0))))))*X;
  END
  ELSE
  BEGIN
     T := 3.75/X;
     I1X := (((((((-0.00420059D0*T+0.01787654D0)*T-
            0.02895312D0)*T+0.02282967D0)*T-
            0.01031555D0)*T+0.00163801D0)*T-
            0.00362018D0)*T-0.03988024D0)*T+
             0.39894228D0)*EXP(X)/SQRT(X);
   END;
END.
```

Процедуры-функции *I0X* и *I1X* тестировались на IBM PC/AT-286 для *x* = 2.9, 10. Полученные при этом результаты (см. табл. 5.16) совпадают с контрольными [Справочник ..., 1979] и результатами, представленными в работах Белашова (1997) и Гринчишина и др. (1988), с точностью до восьми десятичных знаков.

Таблица 5.16

n	*x* = 2.9	*x* = 10
0	4.50274867	3.61260722
1	2815.71665	2670.98831

В процедуре-функции KNX реализовано вычисление значения функции $e^x K_\nu(x)$ при $\nu = n$ (*n* - целое) на основании ее интегрального представления (5.41), при этом требуемая точность ε задается при обращении к KNX.

Формальные параметры процедуры. *Входные*: *n* (тип ***integer***) - значение порядка; *x* (тип ***real***) - значение аргумента; *eps* (тип ***real***) - требуемая точность вычислений. *Выходной*: *knx* (идентификатор процедуры-функции, тип ***double***) - значение $e^x K_\nu(x)$.

```
FUNCTION KNX(N : INTEGER; X1,EPS : REAL) : DOUBLE;
VAR X,H,G,S,Z,U,ZN,F : DOUBLR;
BEGIN
  X := DOUBLE(X1);
  KNX := 0.;
  H := 1.;
  REPEAT
     G := KNX;
     S := 0.;
     Z := EXP(H/2.);
     U := Z*Z;
     REPEAT
        ZN := EXP (N * LN(Z));
        F := 0.5*DEXP(X*(1.-0.5*(Z+1./Z)))*(ZN+1./ZN);
        S := S+F;
        Z := Z*U;
     UNTIL (F <=EPS);
     KNX := H*S;
     H := H/2.;
  UNTIL (ABS(KNX-G) <=KNX*EPS);
END.
```

Процедура-функция KNX была получена путем некоторой модификации и перевода вначале на язык *FORTRAN* [Белашов, 1997], а затем на язык *PASCAL* Бейсик-программы вычисления $e^x K_n(x)$, приведенной в работе Гринчишина и др. (1988), и проверена при тех же значениях порядка *n* и аргумента *x* на машине IBM PC/AT-386. Полученные при этом для ε = 1*e*-8 результаты

$$e^4 K_9(4) = 1325.16874, \qquad e^{17} K_1(17) = 0.310561234$$

совпали с контрольными [Гринчишин и др., 1988] с точностью до девяти десятичных знаков.

В процедуре-функции BENX в зависимости от способа обращения реализовано вычисление функций $ber_n(x)$ или $bei_n(x)$ на основании их разложений в ряды (5.43), (5.44), при этом вычисления прекращаются, когда совпадут зна-

чения сумм слагаемых $2i$ и $2i + 2$ в разложениях. При отрицательном n вычисления вначале ведутся для $|n|$, а затем используется соотношение (5.48).

Формальные параметры процедуры. *Входные*: x (тип ***real***) - значение аргумента; n (тип ***integer***) - значение порядка; k (тип ***integer***) - параметр, по значению которого выбирается вычисляемая функция: при k = -1 вычисляется функция $ber_n(x)$, при k = 1 - $bei_n(x)$. *Выходной*: *benx* (идентификатор процедуры-функции, тип ***double***) - вычисленное значение функции.

```
FUNCTION BENX(X : REAL;N,K : INTEGER) : DOUBLE;
VAR X0,X1,F,F1,F2 : DOUBLE;
BEGIN
  X0 := DOUBLE(X);
  X1 := X0*X0/4.;
  NA := IABS(N);
  N1 := NA;
  WHILE (N1 > 8) DO
     N1 := N1-8
  IF (K =-1) THEN
  BEGIN
     F1 := DCOS(N1*2.35619449019D0);
     F2 := X1*DSIN(N1*2.35619449019D0);
  END
  ELSE
  BEGIN
     F2 := X1*DCOS(N1*2.35619449019D0);
     F1 := DSIN(N1*2.35619449019D0);
  END;
  N1 := 1;
  FOR I := 1 TO NA DO
     N1 := N1*I;
  X1 := X1*X1;
  F1 := F1/N1;
  F2 := F2/N1/(NA+1.);
  N1 := 1;
  I := 2;
  BENX := F1+K*F2;
 REPEAT
     F1 := -X1*F1/(I*(I-1)*(NA+I-1)*(NA+I));
     F2 := -X1*F2/(I*(I+1)*(NA+I+1)*(NA+I));
     F := BENX+F1+K*F2;
     IF (F<>BENX)THEN
     BEGIN
        BENX := F;
        INC (I, 2);
     END;
  UNTIL F = BENX;
  BENX := BENX*EXP (LN(X0/2.)*NA);
  IF(N < 0) THEN
     IF NA MOD 2 = 1 THEN BENX := (-1)*BENX;
END.
```

Процедура-функция BENX была получена путем некоторой модификации и перевода вначале на язык *FORTRAN* [Белашов, 1997], а затем на *PASCAL* Бейсик-программ вычисления $ber_n(x)$ и $bei_n(x)$, опубликованных в работе Гринчишина и др. (1988), и протестирована на IBM PC/AT-286 для n = 0, 2; x = 3. Полученные результаты

$ber_0(3) = -0.221380250$, $ber_2(3) = 0.808368465$,
$bei_0(3) = 1.937586785$, $bei_2(3) = -0.891022363$

совпадают с табличными значениями [Справочник ..., 1979] и результатами контрольных примеров, приведенными в работе Гринчишина и др. (1988).

В процедуре-функции BE0X в зависимости от способа обращения реализовано вычисление функций $ber(x)$ или $bei(x)$ на основании их разложений в ряды (5.45) с точностью ε, задаваемой при вызове процедуры. Вычисления прекращаются при выполнении неравенства

$$(x^2/4)^m/(m!)^2 \le \varepsilon,$$

где $m = 2k$ для функции $ber(x)$ и $m = 2k + 1$ - для функции $bei(x)$.

Формальные параметры процедуры. *Входные*: x (тип ***double***) - значение аргумента; *eps* (тип ***double***) - задаваемая точность вычислений; n (тип ***integer***) - параметр, по значению которого выбирается вычисляемая функция: при n = 0 вычисляется функция $ber(x)$, при n = 1 - $bei(x)$. *Выходной*: *be0x* (идентификатор процедуры-функции, тип ***double***) - вычисленное значение функции.

```
FUNCTION BE0X (X1,EPS : DOUBLE;N : INTEGER) : DOUBLE;
VAR  F,F1,X : DOUBLE;
BEGIN
  X := DOUBLE(X1*X1/4.);
  BE0X := 1.;
  I := 1+N;  K := 1;  F := 1.;
  REPEAT
     F := F/(I*I*(I+1)*(I+1));
     F1 := EXP ( LN(X)*(2*K))*F;
     IF  K MOD 2 = 1 THEN F1 := - F1;
     BE0X := BE0X+F1;
     I := I+2;       K := K+1;
  UNTIL (ABS(F1) <=EPS);
  IF(N.NE.0)BE0X := BE0X*X;
END.
```

Процедура-функция BE0X тестировалась на IBM PC/AT-286 для x = 1, 2, 3. Полученные результаты, представленные в табл. 5.17, показывают совпадение с контрольными значениями функций [Справочник ..., 1979] с задаваемой в процедуре точностью.

Таблица 5.17

x	$ber(x)$	$bei(x)$
1	0.984381781213	0.249566040033
3	-0.221380249601	1.937586785250
5	-6.230082506865	0.116034583788

В заключение заметим, что вычисление функций $ker(x)$ и $kei(x)$ может быть организовано аналогично в соответствии с разложениями (5.46), (5.47).

§ 7. ФУНКЦИИ БЕССЕЛЯ ДРОБНОГО ПОРЯДКА

К функциям Бесселя дробного порядка относят сферические и модифицированные сферические функции Бесселя, а также функции Эйри.

Сферические и модифицированные сферические функции Бесселя определяются как решения уравнений

$$z^2w''+2zw'\pm[z^2 \mp n(n+1)]w=0, \quad n=0,\pm1,\pm2,\ldots \qquad (5.49)$$

(здесь и далее верхние знаки отвечают случаю сферических, а нижние - модифицированных сферических функций). Так, *сферическая и модифицированная сферическая функции Бесселя* есть

I рода

$$j_n(z)=\sqrt{\pi/2z}J_{n+1/2}(z) \quad \text{и} \quad i_n(z)=\sqrt{\pi/2z}I_{n+1/2}(z),$$

II рода

$$y_n(z)=\sqrt{\pi/2z}Y_{n+1/2}(z) \quad \text{и} \quad i_{-n}(z)=\sqrt{\pi/2z}I_{-n-1/2}(z),$$

III рода

$$h_n^{(1)}(z)=j_n(z)+iy_n(z), \quad h_n^{(2)}(z)=j_n(z)-iy_n(z),$$
$$k_n(z)=(-1)^{n+1}(\pi/2)\left[i_n(z)-i_{-n}(z)\right].$$

Все эти функции удовлетворяют следующим рекуррентным соотношениям [Справочник ..., 1979]:

$$f_{n+1}(z)=\mp f_{n-1}(z)\pm\frac{2n+1}{z}f_n(z), \qquad (5.50)$$

при этом

$$j_0(z)=\frac{\sin z}{z}, \qquad j_1(z)=\frac{\sin z}{z^2}-\frac{\cos z}{z};$$

$$y_0(z)=-j_{-1}(z)=-\frac{\cos z}{z}, \quad y_1(z)=j_{-2}(z)=-\frac{\cos z}{z^2}-\frac{\sin z}{z};$$

$$i_0(z)=\frac{\mathrm{sh}\,z}{z}, \qquad i_1(z)=-\frac{\mathrm{sh}\,z}{z^2}+\frac{\mathrm{ch}\,z}{z};$$

$$i_{-0}(z)=\frac{\mathrm{ch}\,z}{z}, \qquad i_{-1}(z)=-\frac{\mathrm{ch}\,z}{z^2}+\frac{\mathrm{sh}\,z}{z};$$

$$k_0(z)=\sqrt{\pi/2z}e^{-z}, \qquad k_1(z)=\sqrt{\pi/2z}e^{-z}(1+z^{-1}).$$

Пары функций $j_n(z)$, $y_n(z)$ и $h_n^{(1)}(z)$, $h_n^{(2)}(z)$; $i_n(z)$, $i_{-n}(z)$ и $i_n(z)$, $k_n(z)$ являются линейно независимыми решениями соответствующего уравнения (5.49) для любого n.

Сферические и модифицированные сферические функции Бесселя I и II рода могут быть представлены в виде степенных рядов [Справочник ..., 1979]:

$$\left.\begin{aligned} f_n(z)&=\frac{z^n}{(2n+1)!!}\left[1\mp\frac{z^2/2}{1!(2n+3)}+\right.\\ &\left.+\frac{(z^2/2)^2}{2!(2n+3)(2n+5)}\mp\ldots\right];\\ f_n(z)&=\mp\frac{(2n-1)!!}{(-1)^{(s-1)n}z^{n+1}}\left[1\mp\frac{z^2/2}{1!(1-2n)}+\right.\\ &\left.+\frac{(z^2/2)^2}{2!(1-2n)(3-2n)}\mp\ldots\right], \end{aligned}\right\} \qquad (5.51)$$

где $s=1$ для функций I рода и $s=2$ для функций II рода.

Рассмотрим приведенные в работе Белашова (1997) и переведенные авторами с языка *FORTRAN* на язык *PASCAL* процедуры, в соответствии с которыми вычисляются сферические функции Бесселя на основе рекуррентных формул (5.50) и разложений (5.51) для аргумента $x = z \equiv$ Re z.

С помощью процедуры-функции JINX1 вычисляется в зависимости от способа ее вызова значение сферической $j_n(x)$ или модифицированной сферической $i_n(x)$ функции Бесселя на основании рекуррентного соотношения (5.50) для произвольного целого $n \geq 0$. При $x = 0$, когда вычисляемые функции не определены, осуществляется выход к внешней программе обработки ошибки.

Формальные параметры процедуры. *Входные*: n (тип ***integer***) - порядок; x (тип ***double***) - аргумент функции; k (тип ***integer***) - ключ, значение которого определяет выбор вычисляемой функции: при k = -1 вычисляется $j_n(x)$, при k = 1 - $i_n(x)$. *Выходной*: *jinx*1 (идентификатор процедуры-функции, тип ***double***) - вычисленное значение соответствующей функции.

```
FUNCTION JINX1(N,K : INTEGER;X1 : DOUBLE) : DOUBLE;
VAR  X,G,C : DOUBLE;
BEGIN
  IF(X1 = 0.) THEN EXIT;
  X := DOUBLE(X1);
  IF(K.EQ.-1)THEN
  BEGIN
     JINX1 := SIN(X)/X ;
     G := COS(X)/X ;
  END
  ELSE
  BEGIN
     JINX1 := (EXP(X)-EXP(-X))/2./X;
     G := SQRT(1.+(JINX1*X)* (JINX1*X))/X;
  END;
  IF(N <>0)THEN
```

```
  FOR I := 1 TO N DO
  BEGIN
     C := JINX1;
     JINX1 := K*(G-JINX1*(2*I-1)/X);
     G := C;
  END;
END.
```

Процедура-функция JINX1 тестировалась на IBM PC/AT-286 для n = 2, 3; x = 1, 3. Полученные результаты, приведенные в табл. 5.18, совпадают с табличными значениями функций [Справочник ..., 1979] и результатами контрольных примеров, приведенными в работах [Белашов, 1997; Гринчишин и др., 1988].

Таблица 5.18

n	$j_n(x)$		$i_n(x)$	
	$x = 1$	$x = 3$	$x = 1$	$x = 3$
2	6.2035052e-22	0.29863749	7.1562870e-22	1.09650152
3	9.0065812e-32	0.15205166	1.0065091e-22	0.41528758

В процедуре-функции YINX1 в зависимости от способа ее вызова выполняется вычисление сферической $y_n(x)$ или модифицированной сферической $i_{-n}(x)$ функций Бесселя II рода в соответствии с соотношением (5.50) для произвольного целого $n \geq 0$. Значения формальных параметров и структура процедуры те же, что и для JINX1.

```
FUNCTION YINX1(N,K : INTEGER; X1 : DOUBLE) : DOUBLE;
VAR  X,G,C : DOUBLE;
BEGIN    IF(X1 = 0.) THEN EXIT;
  X := DOUBLE(X1);
  IF(K = -1) THEN
  BEGIN
     YINX1 := -COS(X)/X;       G := SIN(X)/X;
  END
  ELSE
  BEGIN
     YINX1 := (EXP(X)+EXP(-X))/2./X ;
     G := SQRT((YINX1*X)* (YINX1*X)-1)/X
  END;
  IF(N <> 0)THEN
  FOR I := 1 TO N DO
  BEGIN      C := YINX1;
     YINX1 := K*(G-YINX1*(2*I-1)/X);
     G := C ;
   END ;
END.
```

Процедура-функция YINX1 тестировалась на IBM PC/AT-286 для n = 2, 3; x = 2, 3. Полученные результаты, приведенные в табл. 5.19, совпадают с табличными значениями функций [Справочник ..., 1979].

Таблица 5.19

x	$y_n(x)$		$i_{-n}(x)$	
	n=2	n=3	n=2	n=3
2	-0.73399142	-1.48436655	0.57177592	-0.55655852
3	-0.26703834	-0.50802305	1.13522480	0.32862120

С помощью процедуры-функции JINX2 вычисляется в зависимости от способа ее вызова значение функции $j_n(x)$ или $i_n(x)$ в соответствии с их разложениями (5.51) для $n \geq 0$.

Формальные параметры процедуры. *Входные*: n (тип ***integer***) - порядок; x (тип ***real***) - аргумент функции; k (тип ***integer***) - параметр, значение которого определяет выбор вычисляемой функции: при k = -1 вычисляется $j_n(x)$, при k = 1 - $i_n(x)$. *Выходной*: *jinx2* (идентификатор процедуры-функции, тип ***double***) - вычисленное значение соответствующей функции. Процесс суммирования прекращается, когда для двух последующих членов ряда выполняется равенство $S_k = S_{k+1}$.

```
FUNCTION JINX2 (N,K : INTEGER; X : DOUBLE) : DOUBLE;
VAR 8X1,X2,T,S : DOUBLE;
BEGIN
  X1 := DOUBLE(X);
  X2 := X1*X1/2.;
  T := 1.;
  S := 0.;
  JINX2 := 1.;
  REPEAT
     S := S+1.;
     T := K*T*X2/S/(N+N+S+S+1.);
     JINX2 := JINX2+T;
  UNTIL (JINX2 <> JINX2+T);
  S := 1.;
  FOR I := 1 TO N DO
     S := S*X1/(I+I+1);
  JINX2 := S*JINX2;
END.
```

По процедуре-функции YINX2 вычисляется значение сферической $y_n(x)$ или модифицированной сферической $i_{-n}(x)$ функции Бесселя II рода с помощью разложения в ряд (5.51) для произвольного целого $n \geq 0$. Значения формальных параметров и структура процедуры те же, что и для процедуры-функции JINX2.

```
FUNCTION YINX2 (N, K : INTEGER; X: DOUBLE) : DOUBLE;
VARX1,X2,T,S : DOUBLE;
BEGIN
  X1 := DOUBLE(X);
  X2 := X1*X1/2.;
  T := 1.;
  S := 0.;
  YINX2 := 1.;
  REPEAT
     S := S+1.;
     T := K*T*X2/S/(S+S-N-N-1.)
     YINX2 := YINX2+T;
  UNTIL (YINX2 <>YINX2+T);
  S := 1.;
  FOR I := 1 TO N DO
     S := -K*S/X1*(I+I-1);
```

```
    S := K*S/X1;
    YINX2 := S*YINX2;
END.
```

Результаты тестирования процедур JINX2, YINX2 на IBM PC/AT-286 для значений порядка n = 2, 3 и аргументов соответственно x = 1, 3 и x = 2, 3 совпадают с результатами, приведенными в табл. 5.18 и 5.19, с точностью до тринадцати - четырнадцати десятичных знаков.

Сферические и модифицированные сферические функции III рода, как это видно из определяющих их формул, легко вычисляются через соответствующие функции I и II рода с помощью приведенных процедур.

Функции Эйри $Ai(x)$ и $Bi(x)$ представляют собой линейно независимые решения дифференциального уравнения второго порядка - *уравнения Эйри* $w'' - zw = 0$
и могут быть представлены в виде степенных рядов [Справочник ..., 1979]:

$$\left.\begin{aligned} Ai(z) &= C_1 f(z) - C_2 g(z); \\ Bi(z) &= \sqrt{3}\left[C_1 f(z) + C_2 g(z)\right]; \\ f(z) &= 1 + \frac{1}{3!}z^3 + \frac{1\times 4}{6!}z^6 + \frac{1\times 4\times 7}{9!}z^9 + \ldots = \\ &= \sum_{k=0}^{\infty} 3^k \left(\frac{1}{3}\right)_k \frac{z^{3k}}{(3k)!}; \\ g(z) &= z + \frac{2}{4!}z^4 + \frac{2\times 5}{7!}z^7 + \frac{2\times 5\times 8}{10!}z^{10} + \ldots = \\ &= \sum_{k=0}^{\infty} 3^k \left(\frac{2}{3}\right)_k \frac{z^{3k+1}}{(3k+1)!}, \end{aligned}\right\} \quad (5.52)$$

где

$$C_1 = Ai(0) = Bi(0)/\sqrt{3} = 3^{-2/3}/\Gamma(\tfrac{2}{3}) \approx 0.355028053888,$$

$$C_2 = -Ai'(0) = Bi'(0)/\sqrt{3} = 3^{-1/3}/\Gamma(\tfrac{1}{3}) \approx 0.258819403793,$$

$$\left(\alpha+\tfrac{1}{3}\right)_0 = 1, \quad 3^k\left(\alpha+\tfrac{1}{3}\right)_k = (3\alpha+1)(3\alpha+4)\ldots(3\alpha+3k-2)$$

(α- произвольное, k = 1, 2, 3,...).

Процедура AIRYX реализует вычисление значения функций $Ai(x)$ и $Bi(x)$ для $x = z \equiv \mathrm{Re}\, z$ с помощью разложения (5.52). Вычисления прекращаются, когда для двух последовательных сумм в разложении (5.52) выполняется равенство $S_k = S_{k-1}$.

Формальные параметры процедуры. *Входной*: x (тип ***real***) - аргумент функции. *Выходные*: *a, b* (тип ***double***) - вычисленные значения функций Эйри $Ai(x)$, $Bi(x)$. В процедуре помимо C_1 и C_2 используется константа

$$\sqrt{3} \approx 1.732050807568877.$$

```
PROCEDURE AIRYX(X0 : DOUBLE;
          VAR A,B : DOUBLE) : DOUBLE;
 VAR A,B,X,F,S1,S2,G,C1,C2,F1,G1 : DOUBLE;
          I, J : INTEGER;
BEGIN
   X := DOUBLE(X0);
   F := 1.;
   I := 1;
   S1 := 1.;
   S2 := X;
   G := X;
   X := X*X*X;
   C1 := 0.355028053887817D0;
   C2 := 0.258819403792807D0;
  REPEAT
      S1 := S1*X/(3*I*(3*I-1));
      S2 := S2*X/((3*I+1)*3*I);
      F1 := F+S1;
      G1 := G+S2;
      F := F1;
      G := G1;
      I := I+1;
   UNTIL (F1 <> F) OR  (G1<>G);
   A := C1*F-C2*G;
   B := (C1*F+C2*G)*1.732050807568877D0;
END.
```

Процедура AIRYX получена путем переработки и перевода вначале на язык *FORTRAN* [Белашов, 1997], а затем на язык *PASCAL* Бейсик-программы вычисления функций Эйри, приведенной в работе Гринчишина и др. (1988). Тестирование процедуры проводилось на IBM PC/AT-286 для значений аргумента x = -3, 0.8. Полученные при этом результаты

$$Ai(-3) = -0.378814293677658,$$

$$Bi(-3) = -0.198289626374926;$$

$$Ai(0.8) = 0.169846315284426,$$

$$Bi(0.8) = 1.042422180129392$$

совпадают с контрольными с точностью до девяти - десяти десятичных знаков.

6.

§ 1. ОСНОВНЫЕ ПОНЯТИЯ МАТЕМАТИЧЕСКОЙ СТАТИСТИКИ

Численные методы, как правило, применяют при математическом моделировании процессов, которые имеют место в самых различных областях знаний: физике, экономике, психологии, геологии, медицине и др. Результаты моделирования при этом сравнивают с экспериментальными данными. По степени их согласованности делают заключение о соответствии или несоответствии выбранной математической модели моделируемому процессу. Чтобы обоснованно делать это заключение, а также уметь из экспериментальных данных извлечь необходимую информацию об объекте исследования, экспериментатор должен владеть методами статистической, регрессионной и корреляционной обработки экспериментальных данных.

Статистические методы направлены в основном на *установление закономерностей*, которым подчиняются массовые случайные явления - некоторые реальные процессы, подвергающиеся случайным воздействиям. Случайный эксперимент или опыт есть процесс, при котором возможны различные исходы, так что заранее нельзя предсказать, каков будет результат. Однако опыт характеризуется тем, что его можно повторить много раз, меняя или нет условия эксперимента. Кроме того, полученные результаты можно обрабатывать различными способами, группируя их так, как это необходимо для лучшего выявления закономерностей. Таким образом, в задачи математической статистики входят:

1) указание способов сбора и группировки сведений, полученных в результате наблюдений или как результат некоторых экспериментов (измерений);

2) разработка методов анализа полученных экспериментальных данных в зависимости от целей исследований.

Напомним коротко некоторые понятия и определения.

Случайной величиной (элементом) *назовем такую величину, которая принимает в результате эксперимента одно и только одно возможное значение из некоторой их совокупности и неизвестно заранее, какое именно.*

По результату отдельного эксперимента нельзя заранее указать точное значение случайной величины, но по совокупности, полученной в эксперименте, можно изучать некоторые закономерности ее поведения, т.е определять вероятностные характеристики случайной величины.

Под ***совокупностью*** *будем понимать множество (конечное или бесконечное) элементов, рассматриваемых как однотипные измерения, полученные на объектах заданного типа.*

Под ***выборкой*** *будем понимать подмножество элементов, выбранных из некоторой совокупности.*

Подмножество элементов, из которого производится выборка, будем называть генеральной совокупностью, а количество объектов (в генеральной совокупности или в выборке) - объемом совокупности.

Если из выборки заранее исключают элементы с заданными свойствами, то такую выборку будем называть ***смещенной.*** Если отобранный из выборки объект возвращается в генеральную совокупность перед выбором очередного объекта, то такую выборку будем называть ***повторной***, если же объект не возвращается, то ***бесповторной.*** Если по данной выборке можно уверенно судит об изучаемом признаке генеральной совокупности, то такую выборку будем называть ***репрезентативной (представительной).***

Параметрами *будем называть статистические характеристики, определенные для совокупности, а если эти характеристики относятся к выборке, то* ***статистиками.*** Заметим попутно, что статистики можно использовать для оценки параметров исходных совокупностей, для проверки гипотез и пр. Все эти признаки можно условно разделить на качественные и количественные.

Случайные величины можно условно подразделить на *дискретные* и *непрерывные.* В то время как первые принимают вполне определенные значения x_1, x_2, ..., $x_к$ с вероятностями соответственно p_1, p_2, ..., $p_к$, то вторые распределены с некоторой плотностью по некоторому отрезку прямой *OX*.

§ 2. МАТЕМАТИЧЕСКИЕ ОЦЕНКИ ЭКСПЕРИМЕНТАЛЬНЫХ ДАННЫХ. ПРОВЕРКА ГИПОТЕЗЫ НОРМАЛЬНОГО РАСПРЕДЕЛЕНИЯ

Пусть теперь из генеральной совокупности извлечена выборка, причем x_1 наблюдалось n_1 раз, x_2 - n_2 раза, ..., $x_к$ - $n_к$ раз, тогда наблюдаемые значения назовем ***вариантами,*** последовательность вариант, записанных в возрастающем порядке, ***вариационным рядом,*** количество наблюдений $n_к$ назовем ***частотами***, а отношение частот к общему объему выборки $n_i/N = W_i$ - ***относительными частотами.***

Если, например, задано распределение частот выборки объема $N = 20$:

x_i	2	6	12
n_i	3	10	7 ,

то тогда относительные частоты будут: $W_1 = 3/20 = 0{,}15$; $W_2 = 10/20 = 0{,}5$; $W_3 = 7/20 = 0{,}35$. Если эти числа сложить, то в результате получим единицу $(0{,}15 + 0{,}5 + 0{,}35 = 1)$.

Наиболее используемая характеристика совокупности - это ее *среднее значение*

$$M = \overline{X} = \frac{1}{N}\sum_{i=1}^{N} x_i \,, \qquad (6.1)$$

которое определяется как сумма результатов наблюдений, деленная на их количество, т.е. это *среднее арифметическое*, если экспериментальные данные считать независимыми случайными событиями. Если данные представлены зависимыми событиями, то математическое ожидание определяется из формулы

$$M = \overline{X} = \sum_{i=1}^{N} p_i x_i \,, \qquad (6.2)$$

где p_i - вероятность появления i-го случайного события. Отметим, что при достаточно большом числе наблюдений N величина $\overline{X}$, вычисленная по формуле (6.1) или (6.2), стремится к истинному математическому ожиданию ζ, которое также характеризуется как случайная величина: $\zeta = \sum_{i=1}^{N} \xi_i p_i$, здесь p_i - вероятность появления случайного события ξ_i. Чем больше N, тем с большей надежностью можно утверждать, что $\zeta \approx \overline{X}$.

Ошибка равенства $\overline{X} \approx Mx_i = \zeta_n$ носит вероятностный характер и описывается некоторым интервалом

$$r_0 = \left|\overline{X} - Mx_i\right| = \left|\frac{1}{N}\sum_{i=1}^{N} x_i - \sum_{i=1}^{N} \xi_i p_i\right| .$$

Этот интервал зависит от закона распределения случайной величины, который также является ее универсальной характеристикой.

Функция распределения определяет для каждого значения x_i на числовой оси вероятность того, что случайная величина X примет значение, меньшее чем x_i, т.е. $F(x_i) = P(X < x_i)$. Функция распределения $F(x_i)$ существует для непрерывных и дискретных величин. Она обладает следующими свойствами:

1) $F(x_i)$ - непрерывна; 2) $\lim\limits_{n \to +\infty} F\left(x_i\right) = 1$;

3) $\lim\limits_{n \to -\infty} F\left(x_i\right) = 0$; 4) $P(x_1 < X < x_2) = F(x_1) - F(x_2)$.

Можно также определить *плотность распределения*

$$\lim_{\Delta X \to 0} F\left(x + \Delta x\right) / \Delta x = f\left(x\right) = F'\left(x\right),$$

которая характеризует *плотность* распределения случайной величины в данной точке. Эта характеристика применяется в случае непрерывного распределения данных.

Для наглядности строят различные функции распределения, в частности полигон и гистограмму. ***Полигоном частот*** назовем ломаную, отрезки которой соединяют точки (x_1, n_1), (x_2, n_2) ,..., $(x_к, n_к)$, а если вместо $n_к$ взяты $W_к$, то тогда говорят о ***полигоне относительных частот***. (рис. 6.1). В случае непрерывного признака строят гистограмму, для чего весь интервал наблюдений разбивается на несколько частичных подынтервалов шириной h и находят для каждого подынтервала $n_к$ сумму частот вариант, попавших в данный интервал.

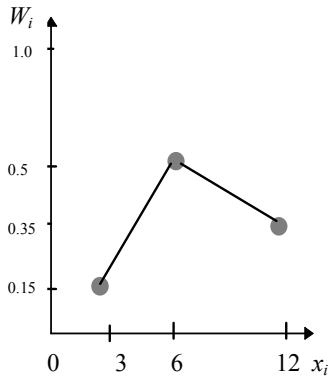

Рис. 6.1 *Полигон частот*

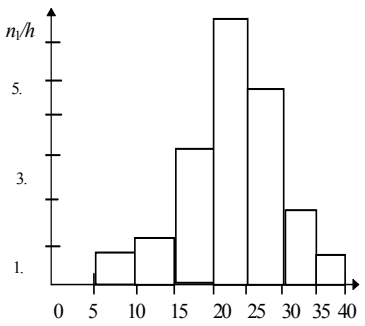

Рис. 6.2. *Гистограмма частот*

Гистограммой частот будем называть ступенчатую фигуру, состоящую из прямоугольников, основаниями которых служат частичные отрезки длиной в h, а высоты равны отношению $n_к/h$, которое называется *плотностью частоты*.

На рис. 6.2 изображена гистограмма частот распределения объема 100 для примера, приведенного в табл. 6.1. Заметим попутно, что площадь гистограммы частот равна сумме всех частот, т.е. объему выборки.

Таблица 6.1

Частичный интервал длиной $h = 5$	Сумма частот вариант	Плотность частоты n_i/h
5-10	4	0.8
10-15	6	1.2
15-20	16	3.2
20-25	36	7.2
25-30	24	4.8
30-35	10	2.0
35-40	4	0.8

Для построения гистограммы на оси $0x$ (абсцисс) откладывают частичные интервалы, а над ними проводят отрезки, параллельные оси $0x$ на расстоянии $n_к/h$. Иными словами строят прямоугольники со сторонами x_i, $n_к / h$. Если по оси ординат откладывают относительные частоты, то тогда говорят о построении *гистограммы относительных частот*.

Другой важной характеристикой распределения является *дисперсия* - мера разброса отдельных значений относительно среднего значения. Квадратный корень из дисперсии называют *стандартным отклонением*. Эти величины обозначаются соответственно D и σ.

Вычисляют эти характеристики по формулам:

$$D = \frac{1}{N} \cdot \sum_{i=1}^{N} \left(x_i - \overline{X}\right)^2 ; \sigma = \sqrt{D} \,.$$

Другие основные числовые характеристики случайной величины приводятся в табл. 6.2. Заметим, что σ характеризует группировку наблюдений вокруг центрального значения; A - скошенность графика функции $f(x)$ (если $A = 0$, то график $f(x)$ симметричен относительно центрального значения, $A > 0$ - вытянут правый конец, $A < 0$ - вытянут левый); E - показывает остроту пика кривой по сравнению с нормальным законом: $E > 0$ - более острый пик, а $E < 0$ - менее.

Таблица 6.2

Обозначения	Название функции	Математическое выражение
M_0	Мода	Такое значение x_i, при котором $f(x) = \max$
V	Коэфф. вариации	$V = \sigma / M$
$\widetilde{X}_i$	Центрированное нормированное уклонение	$\widetilde{X}_i = (x_i - M) / \sigma$
μ_k	Начальный момент k-го порядка	$\mu_k = M(X - M)^k = \frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - \overline{X})^k$
A	Асимметрия распределения	$A = \mu_3 / \sigma^3$
E	Эксцесс распределения	$E = \mu_4 / \sigma^4 - 3$

Величины A и E на практике используют для оценки нормальности закона распределения через вспомогательные коэффициенты [Гольцман, 1971], которые определяются по формулам:

$$\sqrt{D(A)} = a_3 = \sqrt{\frac{6 \cdot (N-1)}{(N+1) \cdot (N+3)}};$$

$$\sqrt{D(E)} = a_4 = \sqrt{\frac{24 \cdot (N-2) \cdot (N-3)}{(N+1)^2 \cdot (N+3) \cdot (N+5)}}.$$

Если выполняются соотношения

$$|A| \le 3 \cdot \sqrt{D(A)}; |E| \le 3 \cdot \sqrt{D(E)},$$

то по критерию Чебышева [Калиткин, 1978] отличие A и E от нуля недостоверно и можно принять гипотезу[1] о нормальном распределении случайной величины X.

Для оценки нормальности распределения случайной величины можно также воспользоваться известным правилом:

$$P(-3\sigma < x < 3\sigma) = 99.7\% = 0.997,$$

т.е. вероятность того, что в ширину интервала $[-3\sigma, 3\sigma]$ попадет 99.7% всех случайных величин данной выборки. Если это не так, то закон распределения случайной величины нельзя считать нормальным.

[1] Этот критерий верен, если N достаточно большое. Если же N невелико, то к данному критерию следует относиться с осторожностью. В этом случае рекомендуется для надежности выводов воспользоваться другими методами анализа нормальности распределения случайной величины.

Возможен вариант, когда сама величина X распределена не по нормальному закону, в то время как $\ln(X)$ или $\lg(X)$ - по нормальному. В этом случае говорят о логнормальном распределении случайной величины X.

После того, как вычислены основные статистики распределения, необходимо подтвердить или опровергнуть гипотезу о нормальном распределении случайной величины. Для этого надо сравнить функции распределения $f^*(X)$ случайной величины X и функции распределения $f(X)$ нормального закона

$$f_n(X) = \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot e^L, \quad L = \frac{(X-M)^2}{2 \cdot \sigma^2}.$$

Для сравнения рядов обычно используют критерии согласия, но можно выполнить простой алгоритм, выполняя последовательно следующие шаги:

1) построить графики функций распределений случайной величины X экспериментального $f^*(X)$ и нормального $f(X)$законов в одном масштабе и совместить их наложением друг на друга;

2) если получают два максимума с разными X, то центрируют обе совокупности по правилу $x' = x_i - M$, т.е. вычитают из каждого наблюдения соответствующее среднее. Это преобразование совмещает максимумы функций распределений;

3) выполняют стандартизацию, т.е. переходят к безразмерным единицам: $Z_i = (x_i - M) / \sigma$.

После выполнения указанных действий можно полученное распределение сравнивать с нормальным. Такая оценка называется *кумулятивной функцией распределения.* Она также имеет довольно широкое распространение в математической статистике при первичной обработке экспериментальных данных.

При программировании вычислений одномерных статистик желательно предусмотреть следующие возможности:

1) накопление сумм x_k, $k = 1, 2, 3, 4...$;

2) возможность исключения ошибочно введенного числа x_i;

3) подсчет N в ходе ввода x_i;

4) подсчет и просмотр статистических характеристик в любой момент (до окончания ввода всех x_i).

Пункты 2 - 4 предусматривают "ручной" ввод данных. В случае ввода данных с магнитных носителей необходимость в них отпадает.

Оформим несколько вспомогательных процедур. Их входные и выходные параметры не представляют собой каких-либо сложностей и поэтому специально не описываются.

```
{ ***□□□□□□□□ □□□ □□□□□□□□ □□□□ (□ - □) : *** }

FUNCTION SUMX (X,S,M : REAL; K:INTEGER) : REAL;
VAR SS : REAL;
BEGIN
  CASE K OF
     1: SS := S+X;
     2: SS := S + (X-M)*(X-M);
```

```
        3: Ss := s + sqr(x-m)*(x-m);
        4: Ss := s + sqr(x-m)*sqr(x-m);
    end;
    SumX := Ss;
end.


          { ***□□□□□□□□□ □□□ □□□□□□□□□□ □□□□□□□□□ : *** }

Procedure Stat1 (x : mas1; var st : mas4;
                 n : integer);
Var i,j : integer;  sm, s, r : real;
begin
    for i := 1 to n do
        st[i] := 0.0;
    sm := 0.0;
    for i := 1 to n do
    begin
        s := x[i];
        sm := SumX (s,sm,0.0,1);
    end;
    st[1] := sm;
    sm := st[1]/n;
    for i := 1 to n do
        for j := 2 to 4 do
        begin
            s := st[j];
            r := x[i];
            st[j] := SumX (r,s,sm,j);
        end;
 end.
```

Если теперь разделим элементы массива *ST* на *N* (или *N* - 1 для несмещенных оценок), то получим

```
Dis := st[2] / NN;
disn := st[2] / (NN-1);
sig := SQRT (dis);
SigN := Sqrt (DisN);
As := st[3] / (NN*Dis*sqrt(Dis));
Ex  := st[4] / (NN * sqr(Dis)) - 3.0;
Xm := st[1] / nn;
A3 := Sqrt (6*(NN-1)/(nn+3)/(nn+1));
A4 := Sqrt (24*(NN-2)*(NN-3)*NN/
           (NN+1)/(nn+1)/(nn+3)/(NN+5)).
```

Введенные значения *X* накапливаются в массиве *X*[*i*]. Если ввод в программе выполнять по алгоритму

Шаг 1. Ввод очередного *X* [*i*];
Шаг 2. *N* = *N* + 1;
Шаг 3. Вычисление *ST* [*к*], где *к* = 1, ..., 4;
Шаг 4. Опрос клавиатуры (*ch*);
Шаг 5. Если *ch* = ESC то
    Шаг 5.1. Если "конец ввода" то конец
    Шаг 5.2. иначе Нач
        Шаг 5.2.1. Вывод *X*, которое было введено ошибочно;
        Шаг 5.2.2. Ввод нового *X*
    Шаг 5.3. Конец;
Шаг 6. перейти на Шаг 1,

то тогда обеспечиваются все желаемые условия работы программы.

Без блока ввода (каждый автор программы может сам написать вариант этого блока) программа, по которой выполняют расчет основных статистик и оценки нормальности распределения случайной величины, может выглядеть так (данные вводятся из файла на магнитном носителе).

```
Program Stat;
Const NN = 40;
Type mas1 = Array [1..NN] of real;
           mas4 = Array [1..4 ] of real;
Var x : mas1; st : mas4; i,j,k,n : integer;
           Xm,Dis,DisN,Sig,SigN,A3,A4,As,Ex : real;
           f : text;
begin
  Assign (f,'');
  Reset (f);
  for i := 1 to NN do
     read (f,x[i]);
  Close (f);
  Stat1 (x, st, N);
  Dis := st[2] / NN;
  disN := st[2] / (NN-1);
  sIG := SQRT (dIS);
  SigN := SQRT (DisN);
  As := st[3] / (NN*Dis*sqrt(Dis));
  Ex  := st[4] / (NN * sqr(Dis)) - 3.0;
  Xm := st[1] / nn;
  A3 := Sqrt (6*(NN-1)/(nn+3)/(nn+1));
  A4 := Sqrt (24*(NN-2)*(NN-3)*NN/
         (NN+1)/(nn+1)/(nn+3)/(NN+5));
  writeln ('*SM** ','Xm ',Xm:7:2,' Dis ',Dis:9:6,
         ' DisN ', DisN:9:6,' Sig ',Sig:9:6);
  Write (' SigN ', SigN:9:6,' As ',  As:8:5,
         ' Ex ',Ex:8:5,' A3 ',
        As/A3:8:5,' A4 ',ex/A4:8:5);
end.
```

Для проверки и тестирования предложенных процедур рассмотрим следующий пример. Пусть задан некоторый ряд данных (*N* = 80):

Таблица 6.3

Исходные данные								
0,068	0,27	1	0,1	0,46	1,1	0,9	0,81	0,86
0,05	0,28	0,5	1,6	0,13	2,1	0,71	0,9	0,87
0,11	0,06	2	2	0,85	0,75	0,61	0,94	1,01
0,08	0,08	1	0,41	0,8	0,97	0,67	0,97	1,1
0,05	0	0,5	0,35	0,82	0,85	0,76	0,91	1,2
0,13	2,5	2	0,16	2,3	1,2	0,85	0,82	1,1
0,27	0,06	0,5	0,23	0,91	0,9	0,75	1,2	1,23
0,8	0,12	1,6	0,2	0,84	0,84	0,74	0,97	0,92
0,11	2,2	2	0,41	0,83	0,93	0,85	0,99	

Необходимо:

1) построить полигон частот, перейдя к вариационному ряду, и подсчитать основные статистические характеристики полученного ряда;

2) преобразовать данные, чтобы перейти к непрерывному ряду и построить гистограмму частот.

Отсортируем сначала массив данных и получим следующую таблицу

Таблица 6.3а

Отсортированные исходные данные								
0	0,11	0,27	0,61	0,81	0,85	0,93	1,1	2
0,05	0,11	0,28	0,67	0,82	0,86	0,94	1,1	2
0,05	0,12	0,35	0,71	0,82	0,87	0,97	1,1	2
0,06	0,13	0,41	0,74	0,83	0,9	0,97	1,2	2
0,06	0,13	0,41	0,75	0,84	0,9	0,97	1,2	2,1
0,068	0,16	0,46	0,75	0,84	0,9	0,99	1,2	2,2
0,08	0,2	0,5	0,76	0,85	0,91	1	1,23	2,3
0,08	0,23	0,5	0,8	0,85	0,91	1	1,6	2,5
0,1	0,27	0,5	0,8	0,85	0,92	1,01	1,6	

из которой можно сформировать вариационный ряд (табл 6.4).

Таблица 6.4

Вариационный ряд	Частота	Отн. частота	Вариационный ряд	Частота	Отн. частота	Вариационный ряд	Частота	Отн. частота
0	0	0	0,46	1	0,0125	0,91	2	0,025
0,05	2	0,025	0,5	3	0,0375	0,92	1	0,0125
0,06	2	0,025	0,61	1	0,0125	0,93	1	0,0125
0,68	1	0,0125	0,67	1	0,0125	0,94	1	0,0125
0,08	2	0,025	0,71	1	0,0125	0,97	3	0,0375
0,08	2	0,025	0,74	1	0,0125	0,99	1	0,0125
0,1	1	0,0125	0,75	2	0,025	1	2	0,025
0,11	2	0,025	0,76	1	0,0125	1,01	1	0,0125
0,12	1	0,0125	0,8	2	0,025	1,1	3	0,0375
0,13	2	0,025	0,81	1	0,0125	1,2	3	0,0375
0,16	1	0,0125	0,82	2	0,025	1,23	1	0,0125
0,2	1	0,0125	0,83	1	0,0125	1,6	2	0,025
0,23	1	0,0125	0,84	2	0,025	2	4	0,05
0,27	2	0,025	0,85	4	0,05	2,1	1	0,0125
0,28	1	0,0125	0,86	1	0,0125	2,2	1	0,0125
0,35	1	0,0125	0,87	1	0,0125	2,3	1	0,0125
0,41	2	0,025	0,9	3	0,0375	2,5	1	0,0125

Разделив весь интервал равномерно на частичные отрезки шириной по 0,1, получим непрерывный ряд (см. табл. 6.5).

Таблица 6.5

Непрерыв. ряд	Частота	Отн. частота	Непрерыв. ряд	Частота	Отн. частота
0-01	6	0.075	1,3-1,4	0	—
0,1-0,2	5	0.0625	1,4-1,5	0	—
0,2-0,3	4	0.05	1,5-1,6	0	—
0,3-0,4	1	0.0125	1,6-1,7	1	0.0125
0,4-0,5	2	0.025	1,7-1,8	0	—
0,5-0,6	1	0.0125	1,8-1,9	0	—
0,6-0,7	2	0.025	1,9-2,0	0	—
0,7-0,8	4	0.05	2,0-2,1	1	0.0125
0,8-0,9	8	0.1	2,1-2,2	1	0.0125
0,9-1,0	7	0.0875	2,3-2,4	1	0.0125
1,0-1,1	2	0.025	2,4-2,5	0	—
1,1-1,2	1	0.0125	2,5-2,6	1	0.0125
1,2-1,3	2	0.025			

Полигон частот вариационного ряда, представленного в табл. 6.4, изображен на рис. 6.3, а гистограмма для данных из табл. 6.5 - на рис. 6.4. Здесь по горизонтальной оси отложены значения вариационного ряда, а по вертикальной оси - частота появления указанного значения.

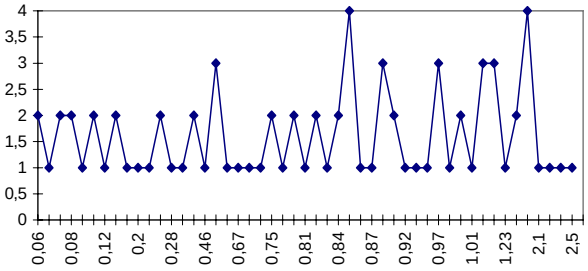

Рис. 6.3. *Полигон частот*

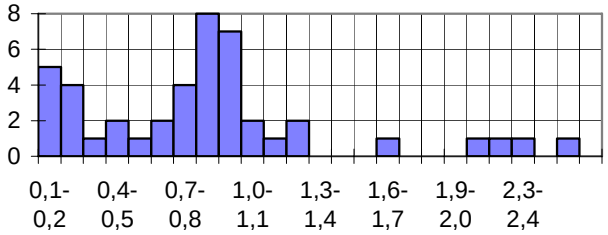

Рис. 6.4. *Гистограмма частот*

Основные статистические характеристики рассматриваемого ряда будут следующими:

математическое ожидание 0.812725;

дисперсия 0.340662; сигма 0.58366;

интервал “трех сигм” [-0.93826; 2.563713].

Очевидно, что данный ряд можно сравнивать по характеристикам с нормальным распределением, однако для того чтобы более уверенно ответить на этот вопрос, необходимо дополнительное исследование ряда другими методами.

В более общем случае, если наблюдается большая дисперсия ряда по сравнению с нормальным законом, можно посоветовать, например, перейти к логнормальному ряду, т.е. прологарифмировать (если среди данных нет отрицательных и нулевых), извлечь корень нечетной степени. Эти преобразования данных позволяют уменьшать первоначальную дисперсию ряда и дают возможность применять стандартные методы обработки данных.

§ 3. КЛАССИФИКАЦИЯ ПО ОДНОМУ ПРИЗНАКУ

3.1. ВВЕДЕНИЕ. ТИПЫ ФАКТОРОВ

В течение многих лет главным вопросом математической статистики было изучение разброса между физическими наблюдениями и соответствие наблюдаемых данных некоторым заранее известным или заданным критериям. Однако в любом эксперименте всегда есть множество внешних условий, которые либо экспериментатор не контролирует, либо их контроль оказывается слишком дорогим и трудоемким (например, температура, атмосферное давление, нервная обстановка, в которой некоторое время назад был испытуемый, износ измерительных приборов и пр.). Значительная часть внешних условий может совершенно не влиять на выполненные измерения, но тем не менее может случиться так, что именно эти неконтролируемые условия окажут решающее влияние на результаты эксперимента. Конечно, можно много раз повторить эксперимент, а потом для обработки выполнить математические преобразования, уменьшающие влияние этих случайных внешних воздействий, но не всегда есть возможности повторения. Чаще всего это связано с оплатой труда испытуемых или с некоторыми уникальными условиями эксперимента, которые готовятся заранее. Поэтому возникает вопрос: можно ли без повторных экспериментов в полученном ряду выявить те условия, воздействие которых значительно, поддается проверке, и по возможности уменьшить их влияние.

Внешние неконтролируемые условия назовем *факторами.* Факторы бывают двух видов: *систематические* и *случайные.* К факторам первого вида можно отнести разные варианты опытов, а к факторам второго вида - отсутствие однородности свойств элементов в испытуемой группе. В первом случае экспериментатор сам выбирает такие условия эксперимента, которые наиболее интересны ему, а во втором ему необходимо так разделить исследуемый материал, чтобы обеспечить его наибольшую однородность в каждой отдельной группе.

Но чаще всего на практике невозможно однозначно определить случайное или систематическое влияние фактора. Например, предположим, что экспериментатор имеет дело с фабриками или предприятиями одного экономического района. Если включить в рассмотрение все имеющиеся предприятия, то такой фактор, как их неоднородность, будет систематическим и, исключив его, можно исследовать предприятия только одного типа. Но если делать выборку из всех имеющихся в районе предприятий, то результаты эксперимента должны быть применены ко всем заводам, включенным в выборку.

Ответ на вопрос, является тот или иной фактор систематическим или случайным, служит ответом на другой вопрос: можно ли результаты эксперимента применять для других выборок или нет, например в нашем случае для анализа деятельности предприятий другого района.

Некоторые факторы, например, такие, как варианты опытов, если опыт готовится специально, могут оказывать только систематическое воздействие, выявляя именно те свойства, которые наиболее интересны экспериментатору. Другие факторы в зависимости от условий опыта могут оказывать как систематическое, так и случайное воздействие. Поэтому есть смысл разделить все виды экспериментов на три группы:

1) эксперименты, в которых все факторы имеют систематические (фиксированные) уровни;

2) эксперименты, в которых все факторы имеют случайные уровни;

3) эксперименты, в которых есть факторы, имеющие случайные уровни, и факторы, имеющие систематические (фиксированные) уровни.

Последняя группа экспериментов известна в математической статистике как *смешанная модель*, а две первые - модели типа I и II. Для того чтобы пользоваться описанными в работе методиками, следует признать, что количество возможных уровней случайных факторов бесконечно. В противном случае описываемые здесь методики неприменимы. И еще одно замечание. При анализе каждого эксперимента необходимо очень серьезно подходить к анализу условий, в которых он проведен, так как соответствующие критерии зависят в большей части своей именно от этого фактора.

3.2. КЛАССИФИКАЦИЯ ПО ОДНОМУ ПРИЗНАКУ С РАЗНЫМ КОЛИЧЕСТВОМ НАБЛЮДЕНИЙ

3.2.1. РАВНОЕ ЧИСЛО НАБЛЮДЕНИЙ

Рассмотрим некоторый фактор, который принимает p различных уровней, и предположим, что выполнено n наблюдений на каждом уровне. Тогда имеем таблицу x_{ij} (i = =1, 2, ..., p; j = 1, 2, ..., n). Будем полагать, что для каждого уровня n средняя равна общей средней. Тогда можно записать следующее равенство:

$$x_{ij} = \mu + F_i + \varepsilon_{ij} \quad , \qquad (6.3)$$

где μ - общая средняя, F_i - эффект, обусловленный i-ым уровнем фактора, ε_{ij} - вариация результатов внутри отдельного фактора. При помощи последнего члена принимаются в расчет все неконтролируемые факторы.

Будем предполагать, что наблюдения на фиксированном уровне фактора распределены нормально относительно среднего значения $\mu + F_i$ с общей дисперсией σ^2. Введем обозначение, в котором точка вместо индекса будет обозначать усреднение по этому индексу. Тогда можно будет записать

$$x_{ij} - x_{..} = (x_{i.} - x_{..}) + (x_{ij} - x_{i.}).$$

Возведя обе части этого равенства в квадрат и приведя подобные, получим следующее выражение:

$$\sum_{i,j}\left(x_{ij}-x_{..}\right)^2 = n\sum_{i,j}\left(x_{i.}-x_{..}\right)^2 + \sum_{i,j}\left(x_{ij}-x_{i.}\right)^2 \quad (6.4)$$

Таблица 6.6

Источник изменчивости	Суммы квадратов	Степени свободы	Средние квадраты	Отношение
Различия между уровнями	$S_1 = \dfrac{\sum_i \left(\sum_j x_{ij}\right)^2}{p} - \dfrac{\left(\sum_{i,j} x_{ij}\right)^2}{N}$	p - 1	$M_1 = \dfrac{S_1}{(p-1)}$	$\dfrac{M_1}{M_2}$
Различия внутри уровней	$S_2 = \sum_{i,j} x_{ij}^2 - \dfrac{\sum_i \left(\sum_j x_{ij}\right)^2}{p}$	$N - p$	$M_2 = \dfrac{S_2}{(N-p)}$	
Сумма	$S = \sum_{i,j} x_{ij}^2 - \dfrac{\left(\sum_{i,j} x_{ij}\right)^2}{N}$	N-1		F_α

или кратко $S = S_1 + S_2$, где S_1 показывает различия между уровнями и вычисляется исходя из отклонений p средних для независимых классов от общего среднего и, следовательно, имеет (p - 1) степеней свободы; S_2 - определяет различия внутри уровней и вычисляется по отклонениям N наблюдений от p выборочных средних и, следовательно, имеет p (n - 1) степеней свободы. А само S имеет N - 1 степеней свободы. И если теперь подставить x_{ij} из уравнения (6.3) в выражение (6.4), то после преобразований можно видеть, что S_2 - есть несмещенная оценка p (n - 1) σ^2 различий внутри уровней, в то время как S_1 имеет смысл несмещенной оценки $\dfrac{n}{(p-1)}\sum_i \left(F_i - F_.\right)^2 + \sigma^2$ различий между уровнями. В случае справедливости гипотезы, согласно которой влияние всех уровней одинаково, отношение второй оценки к первой должно быть сравнимо с F-распределением с (p - 1) и (N - p) степенями свободы. Если вычисленное значение больше табличного, то гипотеза ложна при указанном уровне значимости, т.е. различия несущественны.

Результаты удобно оформлять в таблицу.

В табл. 6.6 приведены удобные расчетные формулы для вычисления необходимых параметров ряда, которыми можно пользоваться в практических расчетах. Последний столбец заполняется так: первая строка - рассчитанное значение M_1/M_2, вторая - табличный критерий Фишера, найденный по соответствующей таблице для указанных степеней свободы, в третью строку вписывается результат: принимается или нет гипотеза.

3.2.2. НЕРАВНОЕ КОЛИЧЕСТВО НАБЛЮДЕНИЙ

Если ряды данных разной длины, то надо либо усечь более длинный ряд, либо добавить требуемое количество измерений в более короткий. Иметь равное количество наблюдений на каждом уровне проведения эксперимента желательно из-за простоты интерпретации результатов. Но если все же на разных уровнях имеется разное количество наблюдений, то тогда формула для S_1 несколько усложнится и будет выглядеть так:

$$\sum_i n_i \left(x_{i.} - x_{..}\right)^2 ,$$

для S_2 расчетная формула останется без изменений.

3.3. ПРИМЕР ПРОВЕРКИ ГИПОТЕЗЫ

Предположим, что необходимо проверить разницу между результатами некоторых тестов четырех испытуемых групп при проведении по 10 экспериментов для каждой группы. Данные приводятся в табл. 6.7.

Таблица 6.7

№п/п	1 гр.	2 гр.	3 гр.	4 гр.	№п/п	1 гр.	2 гр.	3 гр.	4 гр.
1	10	21	9	17	6	11	23	10	18
2	10	22	11	15	7	11	19	9	19
3	12	22	10	19	8	12	20	9	10
4	10	20	10	22	9	14	25	16	20
5	15	27	15	20	10	10	21	12	10

Этот тип проверки представляет собой данные для классификации по одному признаку с равным числом наблюдений на каждом уровне. В этом случае p = 4, n = 10. Подставив данные из табл. 6.7, значения p и n в формулы табл. 6.6, получим табл. 6.8.

Таблица 6.8

Источник изменчивости	Суммы квадратов	Степени свободы	Средние квадраты	Отношение
Различия 1	163,1	3	54,367	2,11
Различия 2	928,5	36	25,792	2,87 (>)
Сумма	1091,6	39		Принимается

Иными словами, данных, что между группами существуют различия, нет, т.е. группы равнозначны. Поэтому, с одной стороны, можно обрабатывать группы вместе как единый ряд наблюдений, а с другой стороны, результаты обработки данных по одной из групп могут быть обобщены на все группы, так как они имеют сходные характеристики.

3.4. РАЦИОНАЛЬНЫЕ СХЕМЫ ВЫЧИСЛЕНИЙ

Для облегчения вычислений, если последние выполняются вручную, можно предложить несколько иные формы записи переменных для построения таблиц дисперсионного анализа (табл. 6.6)

$$S = \sum_{i,j} (x_{ij} - x_{..})^2 = \sum_{i,j} (x_{ij})^2 - \frac{\left(\sum_{i,j} x_{ij}\right)^2}{N};$$

$$S_1 = n \cdot \sum_{i,j} (x_{i.} - x_{..})^2 = \frac{1}{p}\sum_{i}\left(\sum_{j} x_{ij}\right)^2 - \frac{\left(\sum_{i,j} x_{ij}\right)^2}{N};$$

$$S_2 = \sum_{i,j} (_{ij} - x_{i.})^2 = \sum_{i,j} (x_{ij})^2 - \frac{\sum_{i}\left(\sum_{j} x_{ij}\right)^2}{p},$$

анализируя которые можно предложить следующий алгоритм вычисления данных сумм.

Шаг 1. Просуммировать наблюдения для каждого уровня и получить $\sum_j x_{ij}$ для каждого j, а также найти общую сумму $\sum_{i,j} x_{ij}$.

Шаг 2. Возвести каждое наблюдение в квадрат и повторить подсчет, как в первом пункте, т.е. найти суммы квадратов наблюдений по строкам $\sum_j x_{ij}^2$ и общую сумму квадратов $\sum_{i,j} x_{ij}^2$.

Шаг 3. Найденные суммы квадратов на шаге 1 возводим в квадрат. Таким образом будут найдены суммы квадратов $\sum_i \left(\sum_j x_{ij}\right)^2$. Полученный результат надо разделить на p.

Шаг 4. Общую сумму, найденную на первом шаге, возвести в квадрат и разделить на N.

Шаг 5. Теперь легко образовать S, S_1 и S_2 из всего, что было вычислено в шагах 1 - 4 по формулам табл. 6.6.

§ 4. КЛАССИФИКАЦИЯ ПО НЕСКОЛЬКИМ ПРИЗНАКАМ

В исследованиях, включающих в себя много факторов, неэффективно и неэкономично изучать поочередно действие каждого из них на результат. Такой способ не дает достаточной информации о возможных взаимодействиях между соответствующими факторами. Следовательно, необходимо рассмотреть эксперименты, в которых учитывается сразу несколько факторов. Для логичности изложения рассмотрим вначале случай с двумя факторами, затем с тремя и обобщим на более общий случай со многими факторами. Условимся, что когда в эксперименте действуют два фактора, то оба они могут имет фиксированные (модель I) или случайные (модель II) уровни. Если один фактор имеет фиксированные уровни, а другой случайные, то это будет пример смешанной модели.

Все дальнейшие рассуждения будем строить относительно модели I, а затем указывать необходимые изменения для применения полученных соотношений в случае модели II или смешанной. Для удобства будем пользоваться теми же обозначениями, что и для классификаций по одному признаку (§ 3).

Пусть теперь два фактора A и B имеют соответственно p и q уровней свободы и пусть n наблюдений составляют двумерную таблицу размером $p \times q$ так, что всего будет $N = = npq$ наблюдений. Тогда можно записать наблюдения следующим образом:

$$x_{ij\alpha} = \mu + F_i + G_j + I_{ij} + \varepsilon_{ij\alpha}, \quad \left(i = \overline{1,p}; j = \overline{1,q}; \alpha = \overline{1,n}\right), \quad (6.5)$$

где μ - общее среднее; F_i - влияние, обусловленное i-м уровнем первого фактора; G_j - влияние, обусловленное j-м уровнем второго фактора; I_{ij} - член, соответствующий взаимодействию, которое представляют собой отклонения среднего по наблюдениям в (ij)-й ячейке от суммы первых трех членов в равенстве (1), а $\varepsilon_{ij\alpha}$ - учитывает вариацию внутри отдельной ячейки. Предположим, что $\varepsilon_{ij\alpha}$ нормально распределено вокруг нулевого среднего с дисперсией σ^2. Также предположим, что математические ожидания F_i, G_j, $I_{i.}$, I_j равны нулю. Это условие не служит жестким ограничением, и если оно не соблюдается, то к нему можно перейти или с помощью корректировки других факторов, или применив специальные методы обработки экспериментального ряда (см. § 1 и 2).

4.1. ДВУСТОРОННЯЯ КЛАССИФИКАЦИЯ С ПОВТОРЕНИЯМИ

Примем для наблюдений $x_{ij\alpha}$ модель I. Пользуясь точечными обозначениями, как и в предыдущий раз, запишем

$$x_{ij\alpha} - x_{...} = (x_{i..} - x_{...}) + (x_{.j.} - x_{...}) + (x_{ij.} - x_{i..} - x_{.j.} + x_{...}) + (x_{ij\alpha} - x_{ij.})$$

и, возведя обе части в квадрат, просуммировав по i, j, α, получим

$$\sum_{i,j,\alpha}\left(x_{ij\alpha} - x_{...}\right)^2 = nq\sum_{i}\left(x_{i..} - x_{...}\right)^2 + np\sum_{j}\left(x_{.j.} - x_{...}\right)^2 + \tag{6.6}$$

$$+n\sum_{i,j}\left(x_{ij.} - x_{i..} - x_{.j.} + x_{...}\right)^2 + \sum_{i,j,\alpha}\left(x_{ij\alpha} - x_{ij.}\right)^2$$

или после переобозначения некоторых членов и приведения подобных

$$S = S_1 + S_2 + S_3 + S_4 .$$

Члены с перекрестными произведениями обращаются в нуль, как и в случае с одним фактором. Теперь S будет иметь $(N - 1)$, S_1 - $(p - 1)$, S_2 - $(q - 1)$, S_4 - $(N - pq)$ степеней свободы, поскольку они все вычислены исходя из отклонений наблюдений от различных выборочных средних. Для того чтобы в обеих частях равенства (6.6) количество степеней свободы было равным, S_3 должно иметь $(p - 1)(q - - 1)$ степеней свободы. Общее количество наблюдений при этом $N = npq$.

Соответственно этому получим таблицу дисперсионного анализа (табл. 6.8).

Теперь существование эффекта взаимодействия можно проверять, сравнивая соотношения M_3 / M_4 с F-распределением с $(p - 1)(q - 1)$ и $(N - pq)$ степенями свободы. Таким же образом главные влияния, обусловленные факторами A и B, можно проверить при помощи соотношений соответственно M_1 / M_4 и M_2 / M_4. В случае модели II для проверки соотношения M_3 / M_4 с F-распределением можно использовать F-распределение с теми же степенями свободы. Однако для проверки гипотез относительно главного влияния двух факторов соответствующие квадраты следует разделить на средний квадрат взаимодействия, а не на средний квадрат "внутри ячеек"или остаточный средний квадрат. Это одно из самых важных различий между этими моделями.

Для смешанной модели можно положить, что фактор A имеет случайные уровни, а фактор B - фиксированные. Тогда гипотеза относительно члена, соответствующего взаимодействию между уровнями, проверяется из сравнения соотношения M_3 / M_4 с F-распределением с $(p - 1)(q - 1)$ и $(N - pq)$ степенями свободы. Существование эффектов случайности для фактора A проверяется делением среднего квадрата M_1 на средний квадрат "внутри ячеек" или остаточный средний квадрат. Критерий для главного эффекта фактора B получают, разделив M_2 на средний квадрат взаимодействия. Таким образом, критерии для проверки гипотез относительно главных эффектов в смешанной модели действуют обратно критериям для моделей I и II.

4.2. УДОБНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ФОРМУЛЫ

Как и в случае с одним фактором, различные суммы, приведенные в табл. 6.8, можно выразить в формулах, удобных для выполнения вычислений. Формулы приведем без вывода:

$$S = \sum_{i,j,\alpha} \left(x_{ij\alpha} - x_{...} \right)^2 = \sum_{i,j,\alpha} x_{ij\alpha}{}^2 - \frac{\left(\sum_{i,j,\alpha} x_{ij\alpha} \right)^2}{N} ; \quad (6.7)$$

$$S_1 = nq \sum_i \left(x_{i..} - x_{...} \right)^2 = \frac{\sum_i \left(T_{i.} \right)^2}{p} - \frac{\left(\sum_{i,j,\alpha} x_{ij\alpha} \right)^2}{N} ; \quad (6.8)$$

$$S_2 = np \sum_j \left(x_{.j.} - x_{...} \right)^2 = \frac{\sum_j \left(T_{.j} \right)^2}{q} - \frac{\left(\sum_{i,j,\alpha} x_{ij\alpha} \right)^2}{N} ; \quad (6.9)$$

$$S_3 = n \sum_{i,j} \left(x_{ij.} - x_{i..} - x_{.j.} + x_{...} \right)^2 =$$

$$= \frac{\sum_{i,j} \left(T_{ij} \right)^2}{n} - \frac{\sum_i \left(T_{i.} \right)^2}{nq} - \frac{\sum_j \left(T_{.j} \right)^2}{np} + \frac{\left(\sum_{i,j,\alpha} x_{ij\alpha} \right)^2}{N} ; \quad (6.10)$$

Таблица 6.8

Источник изменчивости	Сумма квадратов	Степени свободы	Средние квадраты
Влияние фактора A	$S_1 = nq \sum_i \left(x_{i..} - x_{...} \right)^2$	p - 1	$M_1 = \frac{S1}{p\text{-}1}$
Влияние фактора B	$S_2 = np \sum_j \left[x_{.j.} - x_{...} \right]^2$	q - 1	$M_2 = \frac{S2}{p\text{-}1}$
Взаимодействие AxB	$S_3 = n \sum_{i,j} \left(x_{ij.} - x_{i..} - x_{.j.} + x_{...} \right)^2$	$(p\text{-}1)\,(q\text{-}1)$	$M_3 = \frac{S3}{(q\text{-}1)(p\text{-}1)}$
Различия внутри ячеек	$S_4 = \sum_{i,j,\alpha} \left(x_{ij\alpha} - x_{ij.} \right)^2$	$N - pq$	$M_4 = \frac{S4}{N - pq}$
Сумма	$S = \sum_{i,j,\alpha} \left(x_{ij\alpha} - x_{...} \right)^2$	N - 1	

$$S_4 = \sum_{i,j,\alpha} \left(x_{ij\alpha} - x_{ij.}\right)^2 = \sum_{i,j,\alpha} \left(x_{ij\alpha}\right)^2 - \frac{\sum_{i,j} \left(T_{ij}\right)^2}{n} . \quad (6.11)$$

В формулах (6.7) - (6.11) использованы обозначения: T_{ij} - сумма наблюдений в *ij*-й ячейке; $T_{i.}$ - сумма наблюдений для *i*-го уровня; $T_{.j}$ - сумма наблюдений для *j*-го уровня.

Таким образом, необходимо образовать суммы внутри каждой ячейки T_{ij} и суммы по рядам $T_{i.}$ и столбцам $T_{.j}$. Одновременно следует проверить равенство сумм $\sum_i T_{i.} = \sum_j T_{.j} = \sum_{i,j,\alpha} x_{ij\alpha}$. При этом если количество наблюдений велико, то рекомендуется использовать персональный компьютер и, в частности, электронные таблицы.

4.3. ИЕРАРХИЧЕСКАЯ КЛАССИФИКАЦИЯ ПО ДВУМ ПРИЗНАКАМ

Довольно часто при практических исследованиях встречается случай, когда один фактор “сгруппирован” внутри другого. Тогда с первым фактором нельзя сравнивать никакой другой главный фактор, а в предположении, что фактор *B* полностью содержится в факторе *A*, модель процесса будет задаваться уравнением

$$x_{ij\alpha} - x_{...} = (x_{i..} - x_{...}) + (x_{ij.} - x_{i..}) + (x_{ij\alpha} - x_{ij.}) .$$

Так как предполагается, что фактор *B* не имеет главного воздействия на результат, то в основное модельное уравнение не надо включать член $(x_{.j.} - x_{...})$. Возведя обе стороны в квадрат и просуммировав по *i, j,* α, получим $S = = S_1 + S_2 + S_3$. Члены с перекрестными произведениями опять обращаются в нуль при суммировании. Переменная *S* имеет (*N* - 1), S_1 - (*p* - 1); S_2 - *p*(*q* - 1); S_3 - (*N* - *pq*) степеней свободы, так как их вычисляют, исходя из отклонений наблюдений от различных выборочных средних. И таблица дисперсионного анализа может быть составлена, как табл. 6.9.

Вторая строка табл. 6.9 представляет собой вариацию между теми ячейками двусторонней классификационной таблицы, которые соответствуют фиксированным уровням фактора *A*. Третья строка - это вариация внутри ячеек, обусловленная повторением эксперимента при фиксированных уровнях факторов *A* и *B*.

Существование главного фактора *A* и влияние эффекта фактора *B* внутри *A* теперь должны проверяться делением соответствующего среднего квадрата на M_3 и сравнением результата с *F*-распределением.

Вычислительные формулы в табл. 6.9 можно преобразовать, использовав формулы, приведенные в п. 4.2.

4.4. МНОГОСТОРОННЯЯ КЛАССИФИКАЦИЯ С ПОВТОРЕНИЯМИ

Для характеристики общего случая достаточно будет рассмотреть три фактора, которые условно назовем *A, B, C*. Модель процесса, когда факторы имеют *p*, *q* и *r* уровней, а в каждой ячейке - *n* наблюдений, можно записать

$$x_{ijk\alpha} = \mu + F_i + G_j + H_k + I_{ij} + J_{ik} + K_{jk} + L_{ijk} + \varepsilon_{ijk\alpha} .$$

Здесь F_i , G_j , H_k - главные влияния, обусловленные соответствующими факторами; I_{ij}, J_{jk}, K_{jk} - любые возможные взаимодействия между парами факторов; L_{ijk} - отвечает за возможные взаимодействия между всеми тремя факторами. При этом полагают, что

$$i = \overline{(1,p)} \; ; \; j = \overline{(1,q)} \; ; \; k = \overline{(1,r)} \; ; \; \alpha = \overline{(1,N)} \; ; \; N = npqr .$$

Нетрудно убедиться, что количество главных влияний и взаимодействий, относительно которых проверяются гипотезы, равно (2^m - 1), где *m* - предполагаемое количество факторов в эксперименте. Теперь, как и прежде, разложив общую сумму квадратов, построим таблицу дисперсионного анализа (табл. 6.10).

Средние квадраты с M_1 по M_8 получают делением соответствующих сумм квадратов на число степеней свободы. В предположении, что $\varepsilon_{ijk\alpha}$ распределено нормально с нулевой дисперсией, для проверки существования какого-либо главного влияния или взаимодействия делим соответствующие средние квадраты на средний квадрат ошибки и результаты сравниваем с *F*-распределением с соответствующим числом степеней свободы.

Проверку значимых эффектов следует начинать с взаимодействий высших порядков. При проверке гипотез относительно взаимодействия трех факторов M_7 делится на средний квадрат ошибки соответствующего фактора и результат сопоставляется с *F*-распределением, а при проверке гипотез относительно взаимодействия двух факторов средние квадраты этих взаимодействий делятся на M_7. Однако если у экспериментатора нет данных (опыт проведен не

Таблица 6.9

Источник изменчивости	Суммы квадратов	Степени свободы	Средние квадраты
Влияние фактора A	$S_1 = nq \sum_i \left(x_{i..} - x_{...}\right)^2$	(*p* - 1)	$M_1 = \frac{S_1}{p-1}$
Различия между ячейками (внутри фактора A)	$S_2 = n \sum_{i,j} \left(x_{ij.} - x_{i..}\right)^2$	*p* (*q* - 1)	$M_2 = \frac{S_2}{p(q-1)}$
Различия внутри ячеек	$S_3 = \sum_{i,j,\alpha} \left(x_{ij\alpha} - x_{ij.}\right)^2$	(*N* - *pq*)	$M_3 = \frac{S_3}{N - pq}$
Сумма по всем ячейкам	$S = \sum_{i,j,\alpha} \left(x_{ij\alpha} - x_{...}\right)^2$	(*N* - 1)	

Таблица 6.10

Источник изменчивости	Сумма квадратов	Степени свободы
Главные влияния:		
Фактор A	$S_1 = nqr\sum_i \left(x_{i\ldots} - x_{\ldots.}\right)^2$	$(p - 1)$
Фактор B	$S_2 = npr\sum_j \left(x_{.j.} - x_{\ldots.}\right)^2$	$(q - 1)$
Фактор C	$S_3 = npq\sum_k \left(x_{..k} - x_{\ldots.}\right)^2$	$(r - 1)$
Взаимодействие двух факторов		
$A\times B$	$S_4 = nr\sum_{i,j} \left(x_{ij.} - x_{i\ldots} - x_{.j.} + x_{\ldots.}\right)^2$	$(p - 1)(q - 1)$
$A\times C$	$S_5 = nq\sum_{i,k} \left(x_{i.k} - x_{i\ldots} - x_{..k} + x_{\ldots.}\right)^2$	$(p - 1)(r - 1)$
$B\times C$	$S_6 = np\sum_{j,k} \left(x_{.jk} - x_{.j.} - x_{..k} + x_{\ldots.}\right)^2$	$(q - 1)(r - 1)$
Взаимодействие трех факторов $A\times B\times C$	$S_7 = n\sum_{i,j,k} \left(x_{ijk} - x_{ij.} - x_{i.k} - x_{.jk} + x_{i\ldots} + x_{.j.} + x_{..k} - x_{\ldots.}\right)^2$	$(p-1)(q-1)(r-1)$
Различия внутри ячеек	$S_8 = \sum_{i,j,k,\alpha} \left(x_{ijk\alpha} - x_{ijk}\right)^2$	$(N - pqr)$
Сумма по всем ячейкам	$S = \sum_{i,j,k,\alpha} \left(x_{ijk\alpha} - x_{\ldots.}\right)^2$	$(N - 1)$

тщательно, результат предыдущего испытания не подтвердил гипотезу, что одно или более взаимодействий двух факторов равно нулю), то тогда нет точных критериев для проверки гипотезы относительно главных влияний факторов на результат эксперимента. Например, чтобы проверить существование эффекта главного фактора A, следует сравнить M_1 с величиной, которая является несмещенной оценкой $(nr\sigma_i^2 + nq\sigma_j^2 + n\sigma_L^2 + \sigma^2)$, но именно в таком виде ее нет в таблице. Однако такую величину можно образовать, если, например, взять $M_4 + M_5 - M_7$, но эта линейная комбинация не имеет распределения среднего квадрата. Приближенно можно считать, что она распределена как средний квадрат с числом степеней свободы, равным

$$g = \frac{\left(M_4 + M_5 - M_7\right)^2}{\frac{M_4^2}{f_4} + \frac{M_5^2}{f_5} + \frac{M_7^2}{f_7}},$$

где f_4, f_5, f_7 - степени свободы, определенные по табл. 6.10 дисперсионного анализа соответственно для S_4, S_5, S_7. Таким образом, если оба взаимодействия $A\times B$ и $A\times C$ существенны, то для оценки влияния главного фактора A надо сопоставлять отношение $M_1/(M_4+M_5- M_7)$ с F-распределением с f_1 и g степенями свободы[1].

Если можно принять, что взаимодействия $A\times B$ и $A\times C$ равны нулю, то точные критерии для оценки влияния главного фактора A получают, сравнив отношение M_1/M_5 или M_1/M_4 с F-распределением с соответствующим числом степеней свободы, данным в таблице 6.10.

4.5. РАЦИОНАЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СХЕМЫ ДЛЯ ТАБЛ. 6.10

Запишем суммы квадратов несколько в иной, но более удобной форме. Для этого получим вспомогательные суммы по каждой ячейке (индекс j) и по соответствующим факторам $(T_{ij}, T_{i.k}, T_{.jk})$. Нам будут также нужны суммы по двум факторам $(T_{i..}, T_{.j.}, T_{..k})$, а также общая сумма

$$T_{\ldots} = \sum_{i,j,k,\alpha} x_{ijk\alpha}.$$

Правильность подсчета сумм можно контролировать по вспомогательным равенствам:

[1] Этот метод приближения линейной комбинацией средних квадратов с помощью среднего квадрата с некоторым новым числом степеней свободы принадлежит Саттерсвэйту [Satterthwaite, 1946] и Уэлчу [Welch, 1946].

$$\sum_i T_{ij.} = \sum_k T_{.jk} = T_{.j.} .$$

С учетом сказанного, суммы, приведенные в табл. 6.10, можно расписать следующим образом:

$$S = \sum_{i,j,k,\alpha} \left(x_{ijk\alpha} - x_{....}\right)^2 = \sum_{i,j,k,,\alpha} \left(x_{ijk\alpha}\right)^2 - \frac{\left(T_{....}\right)^2}{N} ;$$

$$S_1 = nqr \sum_i \left(x_{i...} - x_{....}\right)^2 = \frac{\sum_i \left(T_{i...}\right)^2}{p} - \frac{\left(T_{....}\right)^2}{N} ;$$

$$S_4 = nr \sum_{i,j} \left(x_{ij..} - x_{i...} - x_{.j..} + x_{....}\right)^2 =$$

$$= \frac{\sum_{i,j} \left(T_{ij.}^2\right)}{pq} - \frac{\sum_i \left(T_{i...}^2\right)}{p} - \frac{\sum_j \left(T_{.j.}^2\right)}{q} + \frac{\left(T_{...}\right)^2}{N} ;$$

$$S_8 = \sum_{i,j,k,\alpha} \left(x_{ijk\alpha} - x_{ijk.}\right)^2 = \sum_{i,j,k,\alpha} x_{ijk\alpha}^2 - \frac{\sum_{i,j,k} T_{ijk}^2}{pqr} .$$

Другое главное влияние взаимодействия двух соответствующих факторов можно получить по аналогии с оценкой S_1 и S_4. Член S_7, соответствующий взаимодействию трех факторов, можно также выразить с помощью вспомогательных сумм, но так как в него будут входить уже восемь членов, то обычно эту сумму получают вычитанием всех других сумм из общей S. Если все же выражение S_7 нужно для проверки предыдущих арифметических выкладок, то его можно вычислить как

$$S_7 = \frac{\sum_{i,j,k} T_{ijk}^2}{pqr} - \frac{\sum_{i,j} T_{ij.}^2}{pq} - \frac{\sum_{i,k} T_{i.k}^2}{pr} - \frac{\sum_{j,k} T_{.jk}^2}{qr} +$$

$$+ \frac{\sum_i T_{i..}^2}{p} + \frac{\sum_j T_{.j.}^2}{q} + \frac{\sum_k T_{..k}^2}{r} - \frac{\left(T_{...}\right)^2}{N} .$$

Как видно из формул, даже при малом количестве данных эти вычисления становятся довольно громоздкими и поэтому такой вид дисперсионного анализа выполняется, как правило, с использованием электронных таблиц.

§ 5. НЕКОТОРЫЕ ВОПРОСЫ ПРЕОБРАЗОВАНИЯ ДАННЫХ

Довольно часто предполагается, что генеральная совокупность распределена нормально. Когда это не так, то все предложенные к рассмотрению и использованию методы не могут применяться, так как построенные отношения в таблицах дисперсионного анализа будут соответствовать некоторому неизвестному распределению, а точные критерии значимости становятся непригодными. Изучать эффект отклонения от нормального закона для описанных в работе критериев - длительная и утомительная работа. Данных может быть достаточно большое разнообразие, следовательно, и подобных критериев должно быть не меньше. Значит, самый надежный путь - это преобразовать данные таким образом, чтобы отклонения от нормального закона распределения были бы небольшими[1].

Одним из методов приведения данных к нормальному виду считается логарифмирование[2]. Когда дисперсия представляет собой некоторую функцию средней, можно воспользоваться *методом стабилизации совокупности.*

Предположим, что x и y - переменные, связанные между собой некоторой функциональной зависимостью $y = f(x)$ (например, $f(x)$ может быть регрессионной зависимостью). Нам надо подобрать функцию $g(x)$ таким образом, чтобы дисперсия у y была бы более стабильной, чем дисперсия у x[3]. Предположим, что переменная x распределена относительно средней m с небольшим стандартным отклонением, тогда в первом приближении можно предположить, что $y = f(m) + (x - m)\, g(x)$, откуда среднее значение y равно $f(m)$, а дисперсия y может быть вычислена как $|g(m)|^2$(дисперсия x).

Если при этом допустить, что дисперсия x может быть выражена некоторой функцией от m, которую обозначим как $q(m)$, а дисперсия y при этом должна оставаться стабильной и постоянной (допустим, равной A), то тогда на основании двух последних выражений имеем

$$g(m) = \sqrt{\frac{A}{q(m)}} \quad \text{или} \quad G(m) = \int \sqrt{\frac{A}{q(m)}}\, dm ,$$

что дает подходящую форму для преобразования данных. Например, если предложить к рассмотрению распределение Пуассона, в котором $q(m) = m$, то тогда последний интеграл может быть вычислен как

$$G(m) = \int \sqrt{\frac{A}{m}}\, dm = 2\sqrt{Am} .$$

Это значит, что подходящим преобразованием для данных, делающих дисперсию независящей от среднего значения, должно быть $y = \sqrt{x}$. Это преобразование с извлечением квадратного корня обычно применяется, когда есть данные о том, что исходный ряд может иметь распределение Пуассона или быть преобразованным к нему. Если это не так, то можно предположить иное преобразование. Довольно полная сводка таких преобразований приводится в работе [Bartlett, 1947].

[1] По общему мнению современных авторов при применении дисперсионного анализа можно допустить умеренные отступления от нормального закона распределения.

[2] Этот способ преобразования данных применим в случае, когда данные имеют довольно большую дисперсию по сравнению с нормальной.

[3] Без ограничения общности можно и наоборот. Самое главное при этом добиться стабилизации дисперсии хотя бы по одной из переменных. Здесь предложено стабилизировать дисперсию по y, так как y является независимой переменной.

7.

§ 1. ЭМПИРИЧЕСКИЕ ЛИНЕЙНЫЕ ЗАВИСИМОСТИ

1.1. МЕТОДЫ ПОСТРОЕНИЯ ЛИНЕЙНЫХ ЗАВИСИМОСТЕЙ И УТОЧНЕНИЕ ИХ ПАРАМЕТРОВ

Анализ экономических, технических, физических процессов приводит к необходимости выявления существенных факторов, влияющих на исследуемый процесс, а также выбора формы связи между этими факторами и оценки параметров полученных уравнений связи.

Будем считать, что некоторое явление характеризуется двумя величинами $\{X\}$ и $\{Y\}$, связанными между собой некоторой неизвестной экспериментатору функциональной зависимостью. Любую из этих величин с одинаковой степенью можно считать независимой, тогда как другая будет считаться зависимой. Пусть, например, независимой положим переменную $\{X\}$. Тогда говорят, что переменная Y связана с $\{X\}$ некоторой зависимостью, которую без ограничения общности можно представить как $Y = F(X)$, где F - некоторый неизвестный оператор, определяющий правило перехода от множества X ко множеству Y. Для простоты можно считать преобразование взаимно однозначным, хотя на практике это выполняется далеко не всегда.

Теперь математически задача сводится к построению явного вида оператора F и затем его уточнению. Методов решения указанной задачи существует достаточно много. Рассмотрим методы *линейного регрессионного анализа*.

Одним из самых простых операторов F является линейный, опеределяющий линейную зависимость вида $Y = AX+ + B$. Для начала положим $B = 0$ и определим связи между переменными X и Y, вычислив параметр A.

Метод выбранных точек. Проведем прямую как можно ближе к нанесенным точкам (рис. 7.1) и выберем на ней произвольную точку $M(x,y)$. Тогда параметр A определим из отношения $A=y/x$. Преимущество этого метода состоит в его наглядности. Но заметим, что значения A могут колебаться довольно сильно, так как прямая строится произвольно и в выборе точек, через которые ее проводят, нет однозначности.

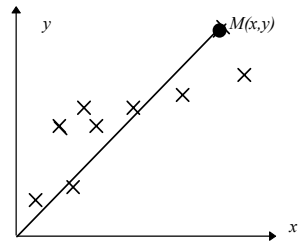

Рис. 7.1. *Множество экспери - ментальных точек {X} и {Y}, нанесенных на плоскость.M(x,y)- -выбранная точка для регрессии*

Метод средних дает лучшие результаты по сравнению с методом выбранных точек. Если предположим, что зависимость построена, тогда $y_i = ax_i$ даст приближенные значения y_i. Определим параметр a из условия минимума средней ошибки

$$\sum_i \left(y_i - y_i^*\right) = \sum_i \left(y_i - ax_i\right) = 0 .$$

Перепишем последнее выражение в виде

$$\sum_i y_i - a \cdot \sum_i x_i = 0 ,$$

преобразуя которое получим выражение для

$$a = \sum_I y_i \Big/ \sum_i x_i .$$

Метод наименьших квадратов дает еще более точные результаты по сравнению с рассмотренными. В этом методе параметр a определяется из условия минимальной суммы квадратов отклонений табличных значений y_i от полученных y_i*:

$$\sum_i \left(y_i - ax_i\right)^2 = \min(F).$$

Условие минимума F, как известно, дает равенство нулю ее первой производной, т.е. $\left.\frac{\partial F}{\partial a}\right|_{a_i} = 0$. Продифференцировав F по a, получим $2\sum_i \left(y_i - ax_i\right) \cdot x_i = 0$, откуда находим $a = \sum_i x_i \cdot y_i \Big/ \sum_i x_i$.

Каждый из приведенных методов является более точным по сравнению с предыдущим, поэтому рекомендуется сначала воспользоваться методом выбранных точек, а затем - одним из двух оставшихся.

Пусть теперь $B \neq 0$. Посмотрим, как изменятся методы. Общий вид зависимости теперь $Y_i = AX_i + B$.

Для уточнения параметров A и B воспользуемся рассмотренными методами.

Метод выбранных точек. Выберем на построенном графике две произвольные точки $M_1(x_1,y_1)$ и $M_2(x_2,y_2)$ (рис. 7.2). Из аналитической геометрии известно, что уравнение прямой будет

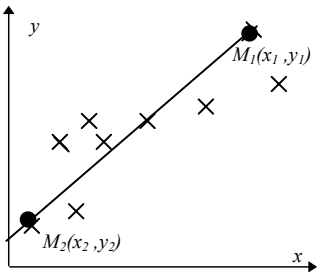

Рис. 7.2. *Пример построения линейной регрессии по двум точкам*

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1} ,$$

откуда получаем

$$y = \frac{y_2 - y_1}{x_2 - x_1} \cdot x + \left(y_1 - \frac{\left(y_2 - y_1\right) \cdot x_1}{x_2 - x_1} \right) .$$

Тогда выражения для параметров A и B можно определить как

$$a = \frac{y_2 - y_1}{x_2 - x_1}; b = y_1 - \frac{\left(y_2 - y_1\right) \cdot x_1}{x_2 - x_1}$$

.

Метод средних. Согласно ему A и B ищем такими, чтобы алгебраическая сумма всех уклонений от вычисленных значений была бы равна нулю:

$$\sum_i \left(y_i - y_i^*\right) = \sum_i \left(y_i - ax_i - b\right) = 0 \cdot$$

Для определения A и B разобьем все данные на две группы так, чтобы сумма алгебраических уклонений каждой группы от среднего была бы равна нулю. Иными словами среднее для одной группы точек было бы равным (или не очень сильно отличалось) среднему другой группы точек. Тогда для каждой группы запишем

$$\begin{cases} \sum_i y_i^{(1)} - A \cdot \sum_{i=1}^{L} x_i^{(1)} - LB = 0; \\ \sum_i y_i^{(2)} - A \cdot \sum_{i=L+1}^{N} x_i^{(2)} - B\left(N - L\right) = 0, \end{cases}$$

где L - число элементов в I группе. Из последней системы найдем A и B :

$$A = \frac{B\left(N - L\right) - \sum_{i=L+1}^{N} y_i^{(2)}}{\sum_{i=L+1}^{N} x_i^{(2)}}; B = \frac{\sum_{i=1}^{L} y_i^{(1)} - A\sum_{i=1}^{L} x_i^{(1)}}{L} \quad .$$

Выполнив над последними выражениями элементарные алгебраические преобразования, получим окончательно выражения для коэффициентов A и B:

$$A = \frac{\sum_{i=L+1}^{N} y_i^{(2)} - \left(N - L\right) \cdot \sum_{i=1}^{L} y_i^{(1)}}{L \cdot \sum_{i=L+1}^{N} x_i^{(2)} + \left(N - L\right) \cdot \sum_{i=1}^{L} x_i^{(1)}};$$

$$B = \frac{\sum_{i=L+1}^{N} y_i^{(2)} \cdot \sum_{i=1}^{L} x_i^{(1)} - \sum_{i=1}^{L} y_i^{(1)}}{N \cdot \sum_{i=1}^{L} x_i^{(1)} - L \cdot \left(1 + \sum_{i=1}^{L} x_i^{(1)}\right)}.$$

Метод наименьших квадратов. Согласно ему ищем минимум функции

$$F = \sum_{i=1}^{N} \left(y_i - Ax_i - B\right)^2 \cdot$$

Используя условие экстремума функции F, находим

$$\begin{cases} \frac{\partial F}{\partial A} = 2\sum_{i=1}^{N} \left(y_i - Ax_i - B\right) \cdot x_i = 0 \ ; \\ \frac{\partial F}{\partial B} = 2\sum_{i=1}^{N} \left(y_i - Ax_i - B\right) = 0. \end{cases}$$

От последней системы можно перейти к более простой, выполнив элементарные алгебраические преобразования

$$\begin{cases} \sum_{i=1}^{N} y_i \cdot x_i - A \cdot \sum_{i=1}^{N} x_i \cdot x_i - B \cdot \sum_{i=1}^{N} x_i = 0 \ ; \\ \sum_{i=1}^{N} y_i - A \cdot \sum_{i=1}^{N} x_i - B = 0. \end{cases}$$

Решая последнюю систему относительно A и B, получаем

$$A = \frac{\sum_{i=1}^{N} x_i \cdot \sum_{i=1}^{N} y_i - N \cdot \sum_{i=1}^{N} x_i \cdot y_i}{\left(\sum_{i=1}^{N} x_i\right)^2 - N \cdot \sum_{i=1}^{N} x_i^2};$$

$$B = \frac{1}{N} \cdot \left(\sum_{i=1}^{N} y_i - A \cdot \sum_{i=1}^{N} x_i\right).$$

1.2. ВЫЧИСЛИТЕЛЬНЫЙ АЛГОРИТМ, ПРОЦЕДУРЫ И ФОРМАЛЬНЫЕ ПАРАМЕТРЫ

Обращает на себя внимание наличие в формулах для A и B конструкций типа $\sum_{i=1}^{N} x_i$ и $\sum_{i=1}^{N} x_i \cdot y_i$. Поэтому для вычисления значений этих сумм следует заранее предусмотреть процедуры-функции:

```
FUNCTION SUMX (N,K:INTEGER; X: MAS1) : REAL;
VAR SS : REAL; I : INTEGER;
BEGIN
  SS := 0.0;
  FOR I := 1 TO N DO
     CASE K OF
        1: SS := SS+X[I];
        2: SS := SS + SQR(X[I]);
     END;
  SUMX := SS;
END.
```

Формальные параметры процедуры. *Входные*: N (тип ***integer***) - общее число точек, которые суммируем; K (тип ***integer***) - параметр, определяющий тип суммирования**:** $K = 1$ - суммируем первые степени x_i; $K = 2$ - суммируем квадраты чисел x_i. *Выходной:* процедура возвращает число (тип ***real***), равное искомой сумме.

```
FUNCTION SUMXY (N,K:INTEGER; X,Y: MAS1) : REAL;
VAR SS : REAL; I : INTEGER;
BEGIN
  SS := 0.0;
  FOR I := 1 TO N DO
      SS := SS+X[I]*Y[I];
  SUMXY := SS;
END.
```

Формальные параметры процедуры. *Входные*: *N* (тип ***integer***) - общее число точек, которые суммируем; *K* (тип ***integer***) - параметр, определяющий тип суммирования: *K* = 1 - суммируем произведение первых степеней x_i на y_i. *Выходной:* процедура возвращает число (тип ***real***), равное искомой сумме.

В параграфах этой главы предложенные процедуры будут модифицироваться за счет включения в оператор CASE новых вариантов конструкций сумм.

Теперь, используя предложенные процедуры-функции, можно определить процедуры, выполняющие вычисления неизвестных параметров *A* и *B*. Заметим, что все вычисления можно было бы свести к одной процедуре, но для ясности изложения было решено составить для каждого варианта вычисления свою.

Метод выбранных точек при *B* = 0 настолько прост, что вычислительная процедура не составляется.

Метод средних и метод наименьших квадратов. Работой функции управляет параметр *K,* который при *K* = 1 вызывает определение *A* методом средних, а при *K* = 2 - методом наименьших квадратов.

```
FUNCTION A (N : INTEGER; X,Y : MAS1; K : INTEGER) : REAL;
 VAR S : REAL;
 BEGIN
   CASE K OF
      1: S := SUMX (N,1,Y) / SUMX (N,1,X);
      2: S := SUMXY(N,1,X,Y) / SUMX (N,2,X);
   END;
   A := S;
END.
```

```
{ **** МЕТОД ВЫБРАННЫХ ТОЧЕК ДЛЯ A ≠0; B ≠0*** }
POCEDURE AB1 (X1,Y1,X2,Y2 : REAL; VAR A,B : REAL;
              VAR K : INTEGER);
BEGIN
  IF ABS(X2-X1)<1.0E-10 THEN
  BEGIN
      K := 1;
      EXIT;
  END;
  A := (Y2-Y1) / (X2-X1);   B := -A*X1 + Y1; K := 0;
END.
```

```
{ **** МЕТОД СРЕДНИХ ДЛЯ A ≠0; B ≠0 **** }
PROCEDURE AB2 (N:INTEGER; X,Y:MAS1;
           VAR A,B : REAL);
VAR X1,Y1:MAS1; L,I : INTEGER; S1, S2, S3, S4 : REAL;
BEGIN     L := N DIV 2;
  FOR I := L+1 TO N DO
  BEGIN       X1[I-L] := X[I]; Y1[I-L] := Y[I];
  END;
  S1 := SUMX (L,1,Y);
  S2 := SUMX (N-L,1,Y1);
  S3 := SUMX (L,1,X);
  S4 := SUMX (N-L,1,X1);
  A := (L*S2 - (N-L)*S1) / (L*S4 + (N-L)*S3);
  B := S1/L - A*S3 /L;
END.
```

```
{****МЕТОД НАИМЕНЬШИХ КВАДРАТОВ ДЛЯ A ≠0; B ≠0 ****}
PROCEDURE AB3 (N:INTEGER; X,Y:MAS1;
            VAR A,B:REAL);
VAR S1, S2,S3, S4 : REAL;
BEGIN
  S1 := SUMX(N,1,X);  S2 := SUMX (N,1,Y);
  S3 := SUMXY(N,1,X,Y);  S4 := SUMX(N,2,X);
  A := (N*S3 - S1*S2) / (N*S4- S1*S1);
  B := (S2 - A*S1) / N;
END.
```

1.3. КОНТРОЛЬНЫЙ ПРИМЕР

Для проверки и тестирования предлагаемых процедур, по данным эксперимента (табл. 7.1), строится линейная зависимость для случаев: 1) $B = 0$; $A \neq 0$; 2) $A \neq 0$; $B \neq 0$. Затем найденные *A* и *B* уточняются.

Таблица 7.1

Параметры	Данные эксперимента						
x	0.1	0.2	0.3	0.4	0.5	0.6	0.7
y	2.59	3.40	3.07	2.81	2.51	2.16	1.80
x	0.8	0.9	1.0	1.1	1.2	1.3	1.4
y	1.60	1.18	0.13	0.69	0.47	0.01	-0.13
x	1.5	1.6	1.7	1.8	1.9	2.0	
y	-0.46	-0.79	-1.16	-1.45	-1.95	-1.75	

Результаты вычислений с использованием приведенных процедур даются ниже.

Метод средних при $B = 0$: $A = 0.78985$.
Метод наименьших квадратов при $B=0$: $A = -0.08371$.

УТОЧНЕНИЕ КОЭФФИЦИЕНТОВ.

Метод выбранных точек: $A = -2.91832$; $B = 3.88663$.
Метод средних: $A = -1.41004$; $B = 3.08540$.
Метод наименьших квадратов: $A = -2.98023$; $B = 3.95859$.

Анализируя полученные результаты по определению коэффициентов разными методами, видим, что $A = -2.95$; $B = 3.92$. Наиболее близкие к этим значениям коэффициенты были нами получены методом наименьших квадратов. Однако при достаточном практическом опыте исследователя можно остановиться на методе выбранных точек. Для наглядности можно построить график, что, впрочем, не обязательно.

§ 2. ВЫБОР ЭМПИРИЧЕСКИХ ФОРМУЛ ДЛЯ АНАЛИЗА НЕЛИНЕЙНЫХ ЗАВИСИМОСТЕЙ

Нами была рассмотрена линейная зависимость вида $y=Ax+B$ для случаев, когда $A \neq 0$; $B = 0$ и $A \neq 0$, $B \neq 0$. Но, к сожалению, построение этой зависимости не дает ответа на вопрос о том, какая аналитическая зависимость наилучшим образом подходит к имеющемуся распределению. Наиболее популярные на практике эмпирические зависимости имеют вид:

1) линейная функция: $y = Ax + B$;
2) показательная функция: $y = AB^x$;
3) дробно-рациональная функция: $y = (Ax+B)^{-1}$;
4) логарифмическая функция: $y = A \cdot \ln(x)+B$;
5) смешанная функция: $y = Ax^B$.

В зависимости от параметра B она определяет параболическую ($B > 0$), гиперболическую ($B < 0$) и линейную ($B = 0$) зависимости;

6) гиперболическая функция: $y = A + B/x$;
7) дробно-рациональная функция: $y=x/(Ax+B)$.

Для того чтобы выбрать теперь вид аналитической зависимости, которая наилучшим образом соответствует исходным экспериментальным данным, поступим следующим образом. Выполним промежуточные вычисления. Из области определения независимой переменной (мы в § 1 условились, что это будет x_i) выберем две точки, достаточно надежные и по возможности как можно дальше отстоящие друг от друга. Обозначим их X_1 и X_2. Этим точкам соответствуют значения Y_1 и Y_2. Найдем теперь среднее арифметическое, среднее геометрическое и среднее гармоническое для выбранных точек:

$$X_{\text{АР}} = \frac{X_1 + X_2}{2}; \; X_{\text{ГЕОМ}} = \sqrt{X_1 \cdot X_2}; \; X_{\text{ГАРМ}} = \frac{2 \cdot X_1 \cdot X_2}{X_1 + X_2}; \; Y_{\text{АР}} = \frac{Y_1 + Y_2}{2};$$

$$Y_{\text{ГЕОМ}} = \sqrt{Y_1 \cdot Y_2}; \; Y_{\text{ГАРМ}} = \frac{2 \cdot Y_1 \cdot Y_2}{Y_1 + Y_2}.$$

Построим график, который, по нашему мнению, наилучшим образом будет соответствовать имеющимся экспериментальным данным. И зная $X_{\text{АР}}$, $X_{\text{ГЕОМ}}$ и $X_{\text{ГАРМ}}$, найдем из графика приближенные $Y^*_{\text{АР}}$, $Y^*_{\text{ГЕОМ}}$ и $Y^*_{\text{ГАРМ}}$. При построении графика можно использовать метод построения интерполяционной кривой по выбранным точкам [Дорот и др., 1977; Крылов и др., 1972; Троицкий, Иванова 1975] или методы, описанные в §1 главы 7.

Теперь найдем погрешности результатов сравнений:

$$|Y^*_{ГЕОМ} - Y_{АР}| = \varepsilon_4;$$

$$|Y^*_{АР} - Y_{АР}| = \varepsilon_1; \quad |Y^*_{ГЕОМ} - Y_{ГЕОМ}| = \varepsilon_5;$$

$$|Y^*_{АР} - Y_{ГЕОМ}| = \varepsilon_2; \quad |Y^*_{ГАРМ} - Y_{АР}| = \varepsilon_6;$$

$$|Y^*_{АР} - Y_{ГАРМ}| = \varepsilon_3; \quad |Y^*_{ГАРМ} - Y_{ГАРМ}| = \varepsilon_7$$

и выберем $\varepsilon = \min \{ \varepsilon_1, \varepsilon_2, ..., \varepsilon_7 \}$.

1. Если наименьшим среди всех абсолютных значений окажется ε_1 , то в качестве аналитической зависимости для данных точек будет служить линейная функция вида $y = Ax + B$.

2. Если наименьшей абсолютной ошибкой является ε_2, то в качестве эмпирической зависимости следует выбрать показательную функцию $y = AB^x$.

3. Если наименьшая из абсолютных ошибок есть ε_3, то искомая эмпирическая зависимость определяется дробно-рациональной функцией вида $y = (Ax + B)^{-1}$.

4. Если наименьшая из абсолютных ошибок есть ε_4 , то хорошим приближением будет служить логарифмическая функция $y=A \ln(x) + B$.

5. Если наименьшая абсолютная ошибка окажется ε_5, то в качестве эмпирической зависимости рекомендуется выбрать смешанную функцию $y = Ax^B$.

6. Если наименьшей из абсолютных ошибок окажется ε_6, то за искомую зависимость следует выбрать гиперболическую функцию $y= A + B/x$.

7. Если наименьшая из всех абсолютных ошибок есть ε_7, то в качестве зависимости следует выбрать дробно-рациональную функцию вида $y = x/(Ax + B)$.

Для уточнения коэффициентов выбранной аналитической зависимости $y = f(x, A, B)$ воспользуемся, как и в § 1 настоящей главы, тремя методами.

Метод выбранных точек. На кривой, которую предварительно построим по множеству экспериментальных точек, выберем две произвольные $S_1(x_1^*, y_1^*)$ и S_2 (x_2^*, y_2^*). Зная вид зависимости $f(x,A,B)$, составим систему

$$\begin{cases} y_1^* = f(x_1^*, A, B); \\ y_2^* = f(x_2^*, A, B), \end{cases}$$

разрешая которую относительно параметров A и B, находим их числовые значения.

Метод средних. В эмпирическую формулу $y = f(x,A, B)$ подставляем последовательно x_i и получаем y_i, которые будут отклоняться от табличных на $e_i = y_i - f(x_i,A,B)$. Согласно методу средних надо определить так A и B, чтобы $e = 0$. Для этого вся совокупность значений разбивается на две группы так, чтобы алгебраическая сумма уклонений в каждой группе равнялась нулю. Таким образом, для определения параметров A и B имеем

$$\begin{cases} \sum_{i=1}^{L} \left[y_i^* - f\left(x_i^*, A, B\right) \right] = 0; \\ \sum_{i=L+1}^{N} \left[y_i^* - f\left(x_i^*, A, B\right) \right] = 0, \end{cases}$$

откуда получаем из совместного решения системы значения двух параметров A и B.

Метод наименьших квадратов. Согласно этому методу A и B должны быть определены так, чтобы выполнялось условие минимума функции

$$F = \sum_{i=1}^{N} \left(y_i - Ax_i - B \right)^2 .$$

В силу необходимого условия экстремума функции находим частные производные функции F по неизвестным коэффициентам A и B и приравниваем их к нулю, откуда получаем систему

$$\begin{cases} \dfrac{\partial F}{\partial A} = 2\sum\limits_{i=1}^{N} \left(y_i - Ax_i - B \right) \cdot x_i = 0 \ ; \\ \dfrac{\partial F}{\partial B} = 2\sum\limits_{i=1}^{N} \left(y_i - Ax_i - B \right) = 0 \ , \end{cases}$$

из решения которой находим A и B.

В табл. 7.2 приводятся коэффициенты A и B для всех рассматриваемых здесь видов зависимостей. Эти коэффициенты были получены методом наименьших квадратов.

Таблица 7.2

Вид зависимости	Система уравнений для определения A и B
$y = Ax + B$	$A = \dfrac{\sum\limits_{i=1}^{N} x_i \cdot \sum\limits_{i=1}^{N} y_i - N \cdot \sum\limits_{i=1}^{N} x_i \cdot y_i}{\left(\sum\limits_{i=1}^{N} x_i \right)^2 - N \cdot \sum\limits_{i=1}^{N} x_i^2}; \qquad B = \dfrac{1}{N} \cdot \left(\sum\limits_{i=1}^{N} y_i - A \cdot \sum\limits_{i=1}^{N} x_i \right);$
$y = AB^x$	$\ln(A) = \dfrac{1}{N} \cdot \left(\sum\limits_{i=1}^{N} \ln(y_i) - \ln(B) \cdot \sum\limits_{i=1}^{N} x_i \right);$ $\ln(B) = \dfrac{\sum\limits_{i=1}^{N} x_i \cdot \sum\limits_{i=1}^{N} \ln(y_i) - N \cdot \sum\limits_{i=1}^{N} x_i \cdot \ln(y_i)}{\left(\sum\limits_{i=1}^{N} x_i \right)^2 - N \cdot \sum\limits_{i=1}^{N} x_i^2};$
$y = (Ax + B)^{-1}$	$A \cdot \sum\limits_{i-1}^{N} y_i^2 + B \cdot \sum\limits_{i-1}^{N} x_i \cdot y_i^2 = \sum\limits_{i-1}^{N} y_i^2 ; \quad A \cdot \sum\limits_{i-1}^{N} x_i \cdot y_i^2 + B \cdot \sum\limits_{i-1}^{N} x_i^2 \cdot y_i^2 = \sum\limits_{i-1}^{N} x_i \cdot y_i \ ;$
$y = A \cdot \ln(x) + B$	$A = \dfrac{\sum\limits_{i=1}^{N} \ln(x_i) \cdot \sum\limits_{i=1}^{N} y_i - N \cdot \sum\limits_{i=1}^{N} \ln(x_i) \cdot y_i}{\left(\sum\limits_{i=1}^{N} \ln(x_i) \right)^2 - N \cdot \sum\limits_{i=1}^{N} (\ln(x_i))^2}; \qquad B = \dfrac{1}{N} \cdot \left(\sum\limits_{i=1}^{N} y_i - A \cdot \sum\limits_{i=1}^{N} \ln(x_i) \right);$
$y = Ax^B$	$\ln(A) = \dfrac{\sum\limits_{i=1}^{N} \ln(x_i) \cdot \sum\limits_{i=1}^{N} \ln(y_i) - N \cdot \sum\limits_{i=1}^{N} \ln(x_i) \cdot \ln(y_i)}{\left(\sum\limits_{i=1}^{N} \ln(x_i) \right)^2 - N \cdot \sum\limits_{i=1}^{N} (\ln(x_i))^2};$ $B = \dfrac{1}{N} \cdot \left(\sum\limits_{i=1}^{N} \ln(y_i) - A \cdot \sum\limits_{i=1}^{N} \ln(x_i) \right);$
$y = x/(Ax+B)$	$A \cdot \sum\limits_{i-1}^{N} y_i^2 + B \cdot \sum\limits_{i-1}^{N} x_i \cdot y_i^2 = \sum\limits_{i-1}^{N} x_i \cdot y_i \ ;$ $A \cdot \sum\limits_{i-1}^{N} x_i \cdot y_i^2 + B \cdot \sum\limits_{i-1}^{N} x_i^2 \cdot y_i^2 = \sum\limits_{i-1}^{N} x_i^2 \cdot y_i$

Для проверки изложенного построим эмпирическую зависимость для некоторой функции, заданной таблично, и, пользуясь рассмотренными методами, уточним коэффициенты найденной зависимости:

x	1	2	3	4	5	6	7	8	9
y	521	308	240.5	204	183	171	159	152	147

В этой работе речь идет об очень простых вычислениях, поэтому программа для выполнения задания не приводится. Однако в случае большого числа N можно воспользоваться процедурами, описанными в п. 1.2.

Для примера рассмотрим упрощенный метод расчета без использования ЭВМ, которым можно воспользоваться для предварительного (грубого) анализа экспериментальных данных.

1. Предположим, что в данном примере крайние табличные значения достаточно надежны. Проведем вспомогательные вычисления, найдем для $x_0 = 1$ и $x_8 = 9$ средние арифметическое, геометрическое и гармоническое: $x_{АР} = 5$; $x_{ГЕОМ} = 3$; $x_{ГАРМ} = 1.8$.

2. Из графика (рис.7.2) найдем значения функции, соответствующие вычисленным значениям аргумента.

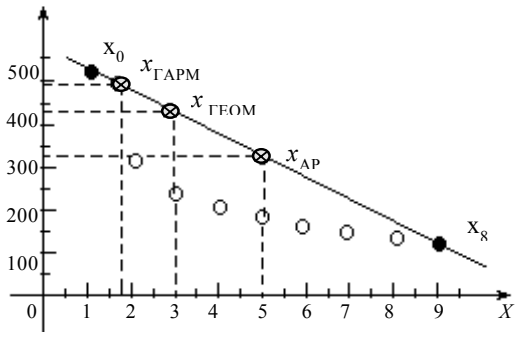

Рис. 7.2. *Экспериментальные данные из таблицы с выделенными точками x_0 и x_8 для построения регрессии разными методами:* ○ *- точки, нанесенные согласно экспериментальным данным;* ● *-выбранные для расчетов точки x_1 и x_8;* ⊕ *- точки, полученные на графике для определения $Y^*_{АР}$, $Y^*_{ГЕОМ}$ и $Y^*_{ГАРМ}$*

Из графика видно, что для $x_{АР} = 5$ имеем $Y^*_{АР} \sim 330$; для $x_{ГЕОМ} = 3$ имеем $Y^*_{ГЕОМ} \sim 430$; для $x_{ГАРМ} = 1.8$ имеем $Y^*_{ГАРМ} \sim 480$.

3. Выполним дополнительные расчеты для соответствующих значений зависимой переменной

$$Y_{АР} = (y_1 + +y_9)/2 = (521 + 147)/2 = 334;$$

$$Y_{ГЕОМ} = (521 \cdot 147)^{1/2} = 276{,}74;$$

$$Y_{ГАРМ} = (2 \cdot 521 \cdot 147)/(521+147) = 229{,}3.$$

4. Сравним графические значения зависимой переменной с вычисленными и найдем $\varepsilon_1, ..., \varepsilon_7$:

$\varepsilon_1 = |330 - 334| = 4$; $\varepsilon_4 = |430 - 334| = 96$;

$\varepsilon_2 = |330 - 277| = 53$; $\varepsilon_5 = |430 - 277| = 153$;

$\varepsilon_3 = |330 - 229| = 101$; $\varepsilon_6 = |480 - 334| = 146$;

$\varepsilon_7 = |480 - 229| = 251$.

5. Поскольку наименьшие из абсолютных ошибок с учетом погрешности есть ε_2 или ε_4, то в качестве аналитической зависимости следует выбрать либо показательную функцию, либо логарифмическую. Возьмем для примера логарифмическую: $y = A\ln x + B$. Погрешность ε_1 в расчетах не учитывается, так как при вычислениях использовались те же точки, что и для построения линейной зависимости.

6. Уточнение коэффициентов проведем по методу наименьших квадратов. Получим: $A \sim 102.59$; $B \sim 405.66$.

7. Проверим полученную зависимость и вычислим отклонение экспериментальных данных от функции (табл. 7.3):

Таблица 7.3

Параметр	Расчетные и экспериментальные данные								
x	1	2	3	4	5	6	7	8	9
y	521	308	240.5	204	183	171	159	152	147
y^*	508.3	305.4	237.8	204	183.7	170.2	160.5	153.3	147.7
D	12.75	2.58	2.69	0	-0.72	0.8	-1.5	1.3	-0.66

§ 3. ПРЕОБРАЗОВАНИЕ НЕЛИНЕЙНОЙ ЗАВИСИМОСТИ В ЛИНЕЙНУЮ МЕТОДОМ ПРЕОБРАЗОВАНИЯ КООРДИНАТ

Рассмотрим в системе координат xOy некоторую линейную зависимость $f(x_i,A,B)$, непрерывную и монотонную на отрезке $[x_1,x_N]$. Перейдем к новым переменным $q = v(x)$ и $z = u(y)$, чтобы в новой системе координат QOZ эмпирическая зависимость стала линейной $z = a'q + b'$.

Заметим, что точки с координатами $[v(x_i),u(y_i)]$ в плоскости QOZ практически лежат на одной прямой. И верно обратное утверждение: если при построении на плоскости QOZ окажется, что точки лежат на одной прямой, то между переменными q и z имеет место линейная зависимость $z = a'q + b'$.

Попытаемся теперь рассмотренные в § 2 нелинейные зависимости преобразовать в линейные.

1. Показательная функция $y = AB^x$. Прологарифмируем зависимость $\lg(y) = x \lg(B) + \lg(A)$, полагая $B' = \lg(B)$; $A' = \lg(A)$; $x = q$; $z = \lg(y)$, получаем $z = B'q + A'$.

2. Дробно-линейная зависимость $y = (Ax + B)^{-1}$. Введем новые переменные $z = 1/y$; $q = x$ и получим $z = A'q+B'$. Заметим, что $B' \equiv B$ и $A' \equiv A$.

3. Логарифмическая зависимость $y = A \ln(x) + B$. Если ввести новые переменные $q = \ln(x)$ и $z = y$, то получим линейную зависимость $z = Aq + B$; A и B не изменились.

4. Степенная зависимость $y = Ax^B$. Пусть $A > 0$ и $B > 0$, тогда логарифмируем $\lg(y) = \lg(A) + B \lg(x)$ и, заменяя переменные $z = \lg(y)$; $A' = \lg(A)$; $B' = B$; $q = \lg(x)$, получаем линейную зависимость $z = A' + B'q$.

Пусть $A > 0$ и $B < 0$, тогда имеем зависимость вида $y = A/x^B$. Логарифмируем последнее выражение $\lg(y) = \lg(A) - B\lg(x)$ и, делая замену переменных $z = \lg(y)$; $A' = \lg(A)$; $B' = -B$; $q = \lg(x)$, получаем линейную зависимость $z = A' + B'q$.

Пусть $A < 0$ и B - любое число. Тогда выполняем замену $A' = -A$, приходим в зависимости от знака либо к первому варианту, либо ко второму.

5. Гиперболическая зависимость $y = A + B/x$ приводится к линейной заменой $z = y$; $q = 1/x$; A и B остаются без изменений $z = A + Bq$.

6. Дробно-рациональная функция $y = x/(Ax + B)$ приводится к линейной заменой $z = 1/y$; $A' = A$; $B' = B$; $q = 1/x$; $z = A + Bq$.

Для наглядности и проверки предлагаемого способа анализа экспериментальных данных преобразуем нелинейную зависимость в линейную и сравним результаты с полученными в п.2.2.

Рассмотрим новую переменную $q = \ln(x)$, для которой, как мы предполагаем, наша зависимость должна стать линейной.

Для того чтобы в этом убедиться, достаточно построить график в указанных координатах, для чего составим новую таблицу значений (табл. 7.4).

Таблица 7.4

Параметры	Экспериментальные данные								
x	1	2	3	4	5	6	7	8	9
$\ln(x)$	0.0	0.69	1.1	1.39	1.61	1.79	1.95	2.08	2.20
y	521	308	240.5	204	183	171	159	152	147

Как видим, экспериментальные точки действительно ложатся на прямую в плоскости $Y0X$ (рис.7.3), т.е. наши предположения оказались верными.

Для того чтобы найти A и B, достаточно выбрать две точки в новых координатах M_1 (0,0; 521) и M_2 (2.20; 147), через которые будет проходить новая регрессионная прямая. Выполнив несложные подсчеты, получаем A = -170; B = 521. Составим новую таблицу (см. табл. 7.5).

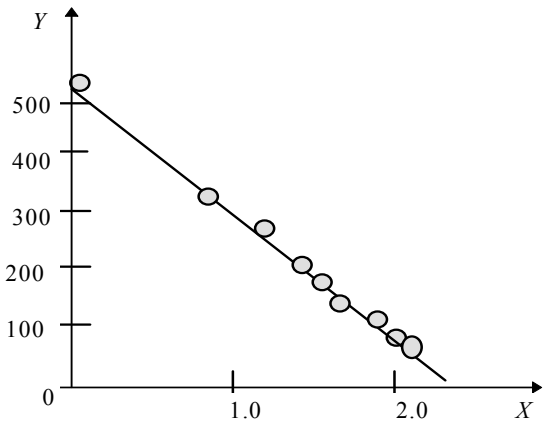

Рис. 7.3. *Преобразование нелинейной зависимости в линейную методом пересчета координат (данные взяты из п. 2.2; выполнялось преобразование координаты X: x* = ln (x)*

и сравним полученные данные (y**) с данными (y*), рассчитанными в п. 2.2. Анализ табл. 7.5 показывает, что Y** отклоняется от Y* незначительно (в пределах ошибок округления), что подтверждает правильность выполненных преобразований. На практике обычно удовлетворяются 10%-й относительной погрешностью вычислений. Если необходима более высокая точность, то в исходных данных надо учитывать большее число значащих цифр (не менее шести).

Таблица 7.5

Параметры	Экспериментальные данные								
X	1	2	3	4	5	6	7	8	9
Y	521	308	240.5	204	183	171	159	152	147
Y *	508.3	305.4	237.8	204	183.7	170.2	160.5	153.3	147.7
D *	12.75	2.58	2.69	0	-0.72	0.8	-1.5	1.3	-0.66
LN(X)	0.0	0.69	1.1	1.39	1.61	1.79	1.95	2.08	2.20
Y **	521	402.3	331.8	281.9	244.1	213.1	185.6	163.2	142.6
D **	0. 0	94.3	89.3	77.92	61.01	42.12	26.2	11.24	-4.4

§ 4. ГАРМОНИЧЕСКИЙ АНАЛИЗ ЭКСПЕРИМЕНТАЛЬНЫХ ДАННЫХ

Гармоническим анализом называют разложение функции $f(x)$, заданной на отрезке $[0, 2\pi]$, в ряд по функциям $\sin(kx)$ и $\cos(kx)$, где k - целое число. Если функция задана на другом отрезке $[x_i, x_{i+1}]$, то линейной заменой переменной задачу можно свести к отрезку $[0, 2\pi]$. А каждой абсолютно интегрируемой на отрезке $[0, 2\pi]$ функции можно поставить в соответствие ее тригонометрический ряд Фурье:

$$f(x) \approx \frac{a_o}{2} + \sum_{k=1}^{\infty} \left[a_k \cdot \cos(kx) + b_k \cdot \sin(kx) \right] \cdot$$

Коэффициенты ряда вычисляются по формулам Фурье - Эйлера при $k = 0, 1, \ldots$, [Хемминг, 1972]:

$$a_k = \frac{1}{\pi} \cdot \int_0^{2\pi} f(x) \cdot \cos(kx)\, dx;$$

$$b_k = \frac{1}{\pi} \cdot \int_0^{2\pi} f(x) \cdot \sin(kx)\, dx$$

и называются *коэффициентами ряда Фурье.*

Сформулируем без доказательства следующую теорему о сходимости ряда Фурье [Хемминг, 1972].

Теорема. Если функция $f(x)$ кусочно-гладкая на отрезке $[0, 2\pi]$, то ее тригонометрический ряд Фурье сходится к каждой точке этого отрезка.

Если $S(x)$ - сумма тригонометрического ряда Фурье, т.е.

$$S(x) \approx \frac{a_0}{2} + \sum_{k=1}^{\infty} \left[a_k \cdot \cos(kx) + b_k \cdot \sin(kx) \right],$$

то справедливы следующие равенства:

1. $S(x) = [f(x+0) + f(x-0)]/2$ - для любой точки из интервала $[0, 2\pi]$.
2. $S(0) = S(2\pi) = [f(0+0) + f(2\pi-0)]/2$.

Таким образом, гармонический анализ состоит в вычислении коэффициентов Фурье a_k, b_k. Для приближенного вычисления интегралов можно использовать любые квадратурные формулы, если $f(x)$ задана таблично. Однако специальный вид подынтегральной функции $f(x)\sin(kx)$ и $f(x)\cos(kx)$ позволяет получить простые квадратурные формулы, если интерполировать алгебраически многочленом только функцию $f(x)$, а не всю подыинтегральную функцию. В частности, в случае кусочно-линейной интерполяции имеем

$$a_0 \cong \frac{1}{2N+1} \cdot \sum_{i=0}^{2N} f(x_i);$$

$$a_k \cong \frac{1}{2N+1} \cdot \sum_{i=0}^{2N} f(x_i) \cdot \cos\left(\frac{2\pi k}{2N+1} i\right);$$

$$b_k \cong \frac{1}{2N+1} \cdot \sum_{i=0}^{2N} f(x_i) \cdot \sin\left(\frac{2\pi k}{2N+1} i\right).$$

Здесь $2N+1$ - количество узлов квадратурной формулы; $x_i = 2\pi/(2N+1)$ - узлы квадратурной формулы; $i = 0, 1, ..., 2N$.

Для вычисления приближенных значений коэффициентов ряда Фурье по предложенным формулам можно воспользоваться подпрограммой *FORIF* из БСП ЕС ЭВМ. Подпрограмма написана на алгоритмическом языке *FORTRAN*, переведена на язык *PASCAL* авторами.

Формальные параметры подпрограммы. *Входные*: *FUN* - имя внешней процедуры-функции (тип ***real***), возвращающей значения $f(x)$ на отрезке $[0, 2\pi]$; N (тип ***integer***) - число, задающее разбиение отрезка $[0, 2\pi]$ на равные части длиной $2\pi/(2N+1)$; M (тип ***integer***) - количество вычисляемых пар коэффициентов Фурье. *Выходные*: A (тип ***real***) - массив из $M+1$ чисел, содержащий значения коэффициентов Фурье $a_1, a_2, ..., a_M$; B (тип ***real***) - массив из $M+1$ чисел, содержащий значения коэффициентов Фурье $b_1, b_2, ..., b_M$; *IER* (тип ***integer***) - признак ошибки во входных параметрах. Если $IER = 0$ - нет ошибки; $IER = 1$ при $N < M$; $IER = 2$ при $M<0$.

```
PROCEDURE FORINT (FNT:MAS21;N,M:INTEGER;
                  VAR A,B:MAS; VAR IER : INTEGER);
LABEL 70,100;
VAR I, AN, J : INTEGER;
                  Q,CONS,COEF,S1,S,C1,C,FNTZ,U0,U1,U2:REAL;
BEGIN
  IER := 0;
  IF M<0 THEN
  BEGIN
     IER := 2;
     EXIT;
  END;
  IF M-N>0 THEN
  BEGIN
     IER := 1;
     EXIT;
  END;
  AN := N;
  COEF := 2.0 / (2.0*AN+1.0);
  CONS := 3.14159265*COEF;
  S1 := SIN (CONS);
  C1 := COS (CONS);
  C:=1.0;
  S :=0.0;
  J := 1;
  FNTZ := FNT[1];
70:
  U2 := 0.0;     U1 := 0.0;
  I := 2*N + 1;
  REPEAT
     U0 := FNT [I] + 2.0 *C*U1 - U2;
     U2 := U1;
     U1 := U0;
     DEC (I);
  UNTIL I<=1;
  A[J] := COEF*(FNTZ+C*U1-U2);
  B[J] := COEF*S*U1;
  IF J>=(M+1) THEN GOTO 100;
  Q := C1*C - S1*S;
  S := C1*S + S1*C;
  C := Q; INC (J);
  GOTO 70;
100:
  A[1] := A[1]/2.0;
END.
```

Для проверки работы процедуры и ее тестирования приведены результаты (табл. 7.6), полученные на отрезке $[0, 2\pi]$ при разложении функции

$$f(x) = \sin^3(x) - 2\cos(x) - 3$$

в ряд Фурье. Для расчетов $M = 6$; $N = 10$; $2N + 1 = 21$.

Таблица 7.6

Номер п/п	Коэффициент Фурье	
	A_i	B_i
1	-3.0000	0.0000
2	-2.0000	0.75000
3	0.00000	0.00000
4	0.00000	-0.2500
5	0.00000	0.00000
6	0.00000	0.00000

§ 5. КОРРЕЛЯЦИОННЫЙ АНАЛИЗ ЭКСПЕРИМЕНТАЛЬНЫХ ДАННЫХ

5.1. ПОСТРОЕНИЕ КОРРЕЛЯЦИОННОЙ МАТРИЦЫ

Математическая теория корреляции изучает наличие зависимости между исследуемыми величинами. Основным применением корреляционных методов обработки экспериментальных данных являются прогнозные задачи, связанные с определением пределов, в которых наперед с заданной надежностью будут содержаться интересующие исследователя величины. Так, например, исследователя может интересовать влияние на то или иное явление некоторых факторов, связь некоторых признаков, выделение связных групп исследуемых объектов и пр.

Довольно часто случайная величина, получаемая из эксперимента, представлена таблицей порядка $m \times n$, где m - количество наблюдений; n - количество фиксируемых переменных. Для того чтобы выяснить, зависимы или нет эти переменные, вычисляют меру взаимной изменчивости пары переменных. Эта мера называется *ковариацией* и является характеристикой совместного изменения двух переменных по отношению к их общему среднему значению. Иными словами, ковариация является мерой разброса значений относительно общего среднего.

Для вычисления коэффициента ковариации введем величину, аналогичную сумме квадратов, назовем ее *центрированной суммой смешанных произведений* (SP_{jk}) и будем определять по формуле

$$SP_{jk} = \sum_{i=1}^{N} (x_{ij} - \bar{x}_j) \cdot (x_{ik} - \bar{x}_k),$$

где x_{ij} - i-е значение j-й переменной; x_{ik} - i-е значение k-й переменной; $\bar{x}_j, \bar{x}_k$ - средние значения j-й и k-й переменных. Перепишем SP_{jk} в форме, удобной для вычислений:

$$SP_{jk} = \sum_{i=1}^{N} x_{ij} x_{ik} - \frac{1}{N} \sum_{i=1}^{N} x_{ij} \cdot \sum_{i=1}^{N} x_{ik} .$$

Если выбрать $j = k$, то последняя формула преобразуется к виду

$$SS_j = \sum_{i=1}^{N} x_{ij} x_{ij} - \frac{1}{N} \sum_{i=1}^{N} x_{ij} \cdot \sum_{i=1}^{N} x_{ij} .$$

Из этой формулы видно, что SS_j - дисперсия величин x_{ij} при условии, что SS_j будет разделена на (N - 1) или N для центрированной величины.

Предположим, что в результате некоторого эксперимента были исследованы три переменные *A, B* и *C*. Подсчитав центрированные произведения переменных *A, B* и *C*, получим табл. 7.7.

Таблица 7.7

	A	*B*	*C*
A	SS_a	SP_{ab}	SP_{ac}
B	SP_{ba}	SS_b	SP_{bc}
C	SP_{ca}	SP_{cb}	SS_c

Легко заметить, что эта таблица симметрична относительно диагональных элементов, так как $SP_{ab} = = SP_{ba}$; $SP_{bc} = SP_{cb}$; $SP_{ac} = SP_{ca}$.

Если теперь разделить каждый элемент таблицы на (N - 1), то получим функцию ковариации

$$COV_{jk} = \frac{SP_{jk}}{N \cdot (N-1)} = \frac{1}{N \cdot (N-1)} \times$$

$$\times \left(N \cdot \sum_{i=1}^{N} x_{ij} x_{ik} - \sum_{i=1}^{N} x_{ij} \cdot \sum_{i=1}^{N} x_{ik} \right),$$

которая, как уже отмечалось, характеризует совместное изменение двух переменных по отношению к их общему среднему значению. Заметим попутно, что интерпретация значений оценок ковариаций должна проводиться таким же образом, как и для дисперсий (см. § 1), но при этом следует помнить, что рассматриваемые значения зависят от единиц измерения, так как являются размерными.

Для оценки степени взаимной связи между переменными, которая не зависит от единиц измерения, на практике пользуются коэффициентом взаимной корреляции $\Re_{jk}$, который вычисляется как отношение ковариации двух переменных к произведению их стандартных отклонений, т.е. $\Re_{jk} = \dfrac{COV_{jk}}{SS_j \cdot SS_k}$. Как видно из формулы, $\Re_{jk}$ - это безразмерная величина. При этом ковариация может равняться произведению стандартных отклонений рассматриваемых переменных, но не может быть больше его. Поэтому $\Re_{jk}$ изменяется на интервале [-1, 1]. Причем, если $\Re_{jk} = 1$, то это указывает на прямую линейную связь между переменными, а если $\Re_{jk} = -1$, то это указывает, что одна переменная является полной противоположностью другой. Между этими двумя крайними случаями имеется спектр значений, который показывает сильные связи между переменными, а если $\Re_{jk} = 0$, то на полное отсутствие линейной связи.

На рис.7.4 показано несколько типов очевидной связи между переменными.

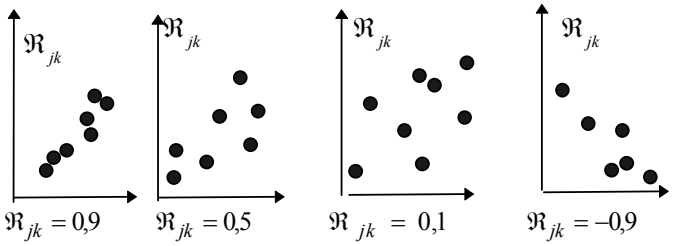

Рис. 7.4. *Расположение экспериментальных точек и значение* $\Re_{jk}$

Коэффициент линейной корреляции удобно вычислять по следующей формуле:

$$\Re_{jk} = \frac{COV_{jk}}{SS_j \cdot SS_k} =$$

$$= \frac{N\sum_{i=1}^{N} x_{ij} \cdot x_{ik} - \sum_{i=1}^{N} x_{ij} \cdot \sum_{i=1}^{N} x_{ik}}{\sqrt{\left(N \cdot \sum_{i=1}^{N} (x_{ij})^2 - \left(\sum_{i=1}^{N} x_{ij} \right)^2 \right) \cdot \left(N \cdot \sum_{i=1}^{N} (x_{ik})^2 - \left(\sum_{i=1}^{N} x_{ik} \right)^2 \right)}}.$$

Заметим, что коэффициент корреляции есть мера линейной зависимости между двумя переменными. Но на практике часто оказывается, что переменные связаны между собой нелинейными зависимостями, тогда коэффициентом линейной корреляции пользоваться нельзя. В этом случае вычисляют коэффициент нелинейной корреляции.

При оценке $\Re_{jk}$ возникает проблема определения предельно допустимого отклонения от нуля выборочного коэффициента корреляции. Это отклонение можно рассчитать из распределения Стьюдента [Корн Г., Корн Т., 1978; Самарский, Гулин, 1989; Численные..., 1976], если положить N - 2 - число степеней свободы и сформулировать нулевую гипотезу H_0:$R^* = 0$, где R^*- коэффициент связности (нулевая гипотеза говорит, что связи между величинами нет):

$$|R_\alpha| = \frac{t_{\alpha,f}}{\sqrt{t^2_{\alpha,f} + N - 2}},$$

где α - уровень значимости; $f = N - 2$ - количество наблюдений; $t_{\alpha,f}$ - выбирается из табл. 7.8, в которой приведены некоторые критические значения для выборочного коэффициента корреляции согласно критерию Фишера [Самарский, Гулин, 1989].

Таблица 7.8

N	$t_{0.05}$	$t_{0.10}$	N	$t_{0.05}$	$t_{0.10}$	N	$t_{0.05}$	$t_{0.10}$
1	0.997	1.00	**16**	0.468	0.590	**110**	0.186	0.242
2	0.950	0.980	**17**	0.456	0.575	**120**	0.178	0.232
3	0.878	0.959	**18**	0.444	0.561	**130**	0.171	0.223
4	0.811	0.917	**19**	0.433	0.549	**140**	0.165	0.215
5	0.754	0.875	**20**	0.423	0.537	**150**	0.160	0.210
6	0.707	0.834	**21**	0.381	0.487	**200**	0.139	0.182
7	0.666	0.798	**30**	0.349	0.449	**300**	0.113	0.148
8	0.632	0.765	**35**	0.325	0.418	**400**	0.098	0.129
9	0.603	0.735	**40**	0.304	0.393	**500**	0.088	0.115
10	0.576	0.708	**50**	0.273	0.354	**600**	0.080	0.105
11	0.553	0.684	**60**	0.250	0.325	**700**	0.074	0.097
12	0.552	0.661	**70**	0.232	0.302	**800**	0.069	0.091
13	0.514	0.614	**80**	0.217	0.283	**900**	0.065	0.086
14	0.497	0.624	**90**	0.205	0.267	**1000**	0.062	0.081
15	0.482	0.606	**100**	0.195	0.254			

Если теперь $|\Re_{jk}| < t_\alpha$, то гипотеза принимается, т.е. между величинами нет значимой связи. Если $|\Re_{jk}| > t_\alpha$, то гипотеза отклоняется, а коррелируемые признаки считают линейно связанными.

5.2. ВЫЧИСЛИТЕЛЬНЫЙ АЛГОРИТМ, ПРОЦЕДУРЫ И ФОРМАЛЬНЫЕ ПАРАМЕТРЫ

Прежде чем составить основную процедуру, выполняющую вычисление корреляционной матрицы, предложим две вспомогательные процедуры-функции:

```
FUNCTION SUMXR (N,J,K:INTEGER; X: MAS2) : REAL;
VAR SS : REAL; I : INTEGER;
BEGIN
  SS := 0.0;
  FOR I := 1 TO N DO
     CASE K OF
        1: SS := SS+X[I,J];
        2: SS := SS + SQR(X[I,J]);
     END;
  SUMXR := SS;
END.

FUNCTION SUMXYR (N,J,K:INTEGER; X,Y: MAS2) : REAL;
VAR SS : REAL; I : INTEGER;
BEGIN
  SS := 0.0;
  FOR I := 1 TO N DO
     SS := SS+X[I,J]*Y[I,K];
  SUMXYR := SS;
END.
```

Тогда процедура, выполняющая вычисление корреляционной функции, может быть:

```
PROCEDURE CORR (N,M : INTEGER; X: MAS2;
                VAR R : MAS2);
VAR I, J, K : INTEGER; XIJ,XIK,XJK,XJ,XJ2,XI,XI2 : REAL;
BEGIN
  FOR J := 1 TO N DO
  BEGIN
     XJ := SUMXR (M,J,1,X);
     XJ2 := SUMXR (M,J,2,X);
     FOR K := 1 TO N DO
     BEGIN
        XIJ := SUMXYR (M,J,K,X,X);
        XI := SUMXR (M,K,1,X);
        XI2 := SUMXR (S,K,2,X);
        R[J,K] := (M*XIJ - XI*XJ) / (SQRT(M*XI2-
                  XI*XI)* SQRT(M*XJ2 - XJ*XJ));
     END;
  END;
END.
```

Формальные параметры процедуры. *Входные*: N (тип ***integer***) - количество измеряемых признаков; M (тип ***integer***) - количество экспериментов (количество измерений N признаков); X (тип ***real***) - матрица $m \times n$, содержащая экспериментальные данные. *Выходные*: R - корреляционная матрица.

5.3. КОНТРОЛЬНЫЙ ПРИМЕР

Для проверки и тестирования предлагаемых процедур и функций просчитаем и проанализируем корреляционную матрицу по уровням значимости 0.05 и 0,10. Вычисления выполнялись с точностью 10^{-5}.

Пусть экспериментальные данные по измерению пяти некоторых признаков представлены табл. 7.9.

Таблица 7.9

Номер эксперимента	Результаты эксперимента				
	I	II	III	IV	V
1	2.30	2.08	2.89	4.85	0.10
2	2.20	4.57	3.37	2.30	0.74
3	2.30	2.30	4.97	2.30	-0.29
4	2.30	2.40	4.94	2.94	-0.03
5	2.30	2.48	4.91	2.89	-0.16
6	2.30	2.56	4.87	2.94	0.18
7	2.30	2.64	4.87	3.26	-0.11
8	2.30	2.71	4.86	3.30	-0.17
9	2.30	2.77	4.70	3.40	-0.07
10	2.30	2.83	4.76	3.37	-0.11
11	2.30	2.08	4.85	2.89	0.10
12	2.30	2.20	4.57	3.37	0.74
13	2.30	2.30	4.97	2.30	-0.29
14	2.30	2.40	4.94	2.94	-0.03
15	2.30	2.48	4.91	2.89	-0.16
16	2.30	2.56	4.87	2.94	0.18
17	2.30	2.64	4.87	3.26	-0.11
18	2.30	2.71	4.86	3.30	-0.17
19	2.30	2.77	4.70	3.40	-0.07
20	2.30	2.83	4.76	3.37	-0.11

Результат работы данной программы приводится в табл. 7.10.

Таблица 7.10

Корреляционная функция				
1.00000	0.33842	0.58387**	-0.29250	-0.45483*
0.33842	1.00000	-0.05452	0.52663*	-0.45056*
0.58387**	-0.05452	1.00000	-0.72840**	-0.73536**
-0.29250	0.52663*	-0.72840**	1.00000	0.36006
-0.45483*	-0.45056*	-0.73536**	0.36006	1.00000

В матрице одной звездочкой обозначены значения связей, соответствующие критерию значимости 0.05, а двумя звездочками - критерию значимости 0.10 (уровни значимости выбраны из таблицы 7.6 для $N = 20$).

Как видно из анализа корреляционной матрицы (табл. 7.10), можно уверенно говорить о линейной связи между I и III и между IV и II признаками. Признак V является антагонистом по отношению к признакам I, II и III. О взаимной линейной связи признаков I и II, II и III, I и IV, V и IV ничего определенного сказать нельзя. Необходимо или провести дополнительные измерения (удлинить экспериментальный ряд), или применить иные методы анализа.

8. **( )**

§ 1. АНАЛИЗ КОРРЕЛЯЦИОННОЙ МАТРИЦЫ МЕТОДОМ КОРРЕЛЯЦИОННЫХ ПЛЕЯД

Этот метод относится к группе методов, включающих в себя самые простые приемы изучения структуры исследуемых систем. Впервые его применил в 1925 г. для анализа геохимических систем П.В.Терентьев и назвал методом *корреляционных плеяд*. В 1944 г. этот метод в несколько модифицированном виде был предложен Ф.Сэттэлом и назван методом *ветвящихся связей*.

Метод удобнее описывать в терминах теории графов. Определим для этого некоторые основные понятия.

Пусть дано конечное множество $A = \{a_1\ a_2, \ldots, a_n\}$, между элементами которого существует вполне определенное соотношение R_g . Каждому элементу a_i можно поставить в соответствие точку на плоскости. Любые две точки a_k и a_l соединим прямыми, если выполняется между ними соотношение $a_k R_g a_l$. Полученная таким образом геометрическая схема называется *графом G(A)*. Элементы множества *A* называют *вершинами графа*, а соединяющие их линии - *ребрами*. Различают *неориентированные ребра*, если отношение R_g симметрично, и *ориентированные ребра*, если R_g антисимметрично. Ориентированные ребра называют еще *дугами*, на схеме их обозначают линией со стрелкой. Неориентированное ребро всегда можно представить в виде двух противоположно ориентированных дуг. С этой точки зрения графы можно разделить на *неориентированные* (к ним относятся и графы, построенные по корреляционной матрице), *ориентированные* и *смешанные*.

Вершины a_k и a_l, соединенные ребром g_{kl}, называют *смежными.* Говорят, что вершины a_k и a_l инцидентны ребру g_{kl}, и наоборот, ребро g_{kl} инцидентно вершинам a_k и a_l. Вершина, которая не инцидентна никакому ребру, называется *изолированной,*а соответствующий граф - *нуль-графом.*

Если все вершины смежные и при этом реализованы все возможные для данного набора вершин соединения, то граф называется *полным.* Последовательность ребер $\{\ldots, g_{kl}, g_{lm}, \ldots\}$, позволяющая обойти более чем две вершины, называется *цепью.* Цепь называется *элементарной,* если в ней ни одна из вершин не повторяется дважды. Последовательность вида $\{g_{kl} \ldots g_{mk}\}$ называется *циклом.*

Условимся, что если все множества точек разбиты на изолированные цепи, циклы и графы, то такие множества будем называть *подграфами* $G^{(1)}(A)$, $G^{(2)}(A)$, ..., $G^{(n)}(A)$ графа $G(A)$.

Если теперь элементы корреляционой матрицы представить вершинами, отношение R_g положить равным r_a, критическому значению выборочного коэффициента корреляции[1], то, заменяя $|\Re_{ij}|$, которые больше r_a, единицами, а все остальные - нулями, получим *матрицу смежности G*, состоящую из групп нулей и единиц.

Если получилось так, что каждый из элементов матрицы смежности равен единице, то граф *G(A)*, построенный на основе данной матрицы, будет полным, и, следовательно, можно сделать вывод, что все элементы множества *A* принадлежат одному классу.

Последний может рассматриваться либо как класс *эквивалентности,* если судить о нем с позиции матрицы смежности, либо как *класс толерантности*, если помнить, что исходные выборочные коэффициенты корреляции не обладают свойством транзитивности.

Есть другой предельный вариант, когда вся матрица смежности *G* состоит из нулей, т.е. все вершины *G(A)* изолированы, что ведет к построению нуль-графа. Тогда весь набор экспериментальных данных разбивается на *p* классов, каждый из которых состоит только из одного элемента a_{ij}. Однако на практике чаще всего наблюдается ситуация, как бы промежуточная между описанными. В этом случае встает вопрос о выделении обособленных групп, решение которого часто оказывается сложной проблемой и, заметим, большей частью неоднозначно.

Эта трудность связана с нетранзитивностью коэффициентов выборочной корреляции $\Re$, что ведет к многовариантности результатов группирования.

Анализ матрицы смежности *G* целесообразно начать с выделения компонент связности. Если последние будут полными подграфами, то они одновременно будут считаться максимальными. В этом случае имеем для выбранного порогового значения r_a однозначное разбиение множества *A* на классы связности.

Здесь возможны два варианта продолжения анализа. Первый ориентируется на матрицу смежности *G* и включает алгоритм нахождения полных подграфов. Другой основан на применении коэффициентов Танимото и описан в работе Р.Бонера [1969], его рассмотрим более подробно.

На базе уже построенной матрицы *G* сформулируем максимально обособленные группы. Для этого будем рассматривать матрицу *G* как совокупность таких ***p*** векторов $g_1, g_2, \ldots g_з$, в которых $g_k = \{g_{1k}, g_{2k}, \ldots, g_{nk}\}$, и введем меру близости между парами подобных векторов, основанную на коэффициенте Танимото:

[1]Критическое значение выборочного коэффициента корреляции r_a можно положить равным уровню значимости по любому из критериев Фишера (табл. 7.8) для соответствующего количества *N* экспериментальных данных.

$$s_{KL} = \frac{C_{KL}}{C_{KK} + C_{LL} - C_{KL}},$$

где C_{KK} и C_{LL} - количество единиц в векторах g_k и g_l, принадлежащих соответственно элементам a_k и a_l; C_{KL} - количество совпадений по единицам.

Полученные значения являются элементами новой матрицы ***S***, отражающей согласованность нуль-единичных векторов $\boldsymbol{g}_k$, $k = 1, ..., p$. Выбрав пороговое значение r_a (см. гл. 7, табл. 7.8), можно матрицу связей S снова трансформировать в матрицу смежности $G^{(1)}$ по правилу $g_{kl}^{(1)} = 1$, если $s_{kl} > r_a$; $g_{kl}^{(1)} = 0$, если $s_{kl} < r_a$. Процедуру повторяют до получения групп с необходимой степенью обособленности. Обычно требуется не больше 2 - 3 итераций.

Для написания процедуры, выполняющей построение матрицы связей, составим две вспомогательные процедуры-функции:

```
FUNCTION SUM1 (N:INTEGER; A1:MASINT) : INTEGER;
            VAR J,I : INTEGER;
BEGIN
   J := 0;
   FOR I := 1 TO N DO
     IF A1[I]=1 THEN
        INC(J);
  SUM1 := J;
END;

FUNCTION SUMED (N:INTEGER; A1, A2:MASINT) : INTEGER;
VAR J,I : INTEGER;
BEGIN
  J := 0;
  FOR I := 1 TO N DO
     IF (A1[I]+A2[I])=2 THEN
        INC(J);
  SUMED := J;
END.
```

Теперь процедура, осуществляющая построение матрицы смежности, имеет вид

```
PROCEDURE MATSMEG (N:INTEGER;RALFA:REAL;
            R : MAS2;VAR G:MAS2INT);
VAR J,I : INTEGER;
BEGIN
  FOR I := 1 TO N DO
     FOR J := 1 TO N DO
        IF R[I,J]>=RALFA THEN
           G[I,J] := 1
        ELSE
           G[I,J] := 0;
END.
```

Формальные параметры процедур. *Входные*: *N* (тип ***integer***) - размер матриц корреляции и смежности; *ralfa* (тип ***real***) - пороговое значение коэффициента корреляции; *R* (тип ***real***) - исследуемая матрица. *Выходные*: *G* (тип ***real***) - матрица смежности.

Для построения матрицы связей S можно предложить следующую процедуру:

```
PROCEDURE COMUN (N:INTEGER; G:MAS2INT;
            VAR S : MAS2);
VAR J,I : INTEGER; A1,A2 : MASINT;
BEGIN
  FOR I := 1 TO N DO
  BEGIN
     FOR J := 1 TO N DO
     BEGIN
        FOR K := 1 TO N DO
        BEGIN
           A1[K] := G[K,I];
           A2[K] := G[K,J];
        END;
        S[I,J] := SUMED (N, A1, A2) / (SUM1 (N,A1)+
              SUM1(N,A2) - SUMED (N,A1,A2));
     END;
     FOR J := I TO N DO        S[J,I] := S[I,J];
  END;
END.
```

Формальные параметры процедур. *Входные*: *N* (тип ***integer***) - размер матриц корреляции; *G* (тип ***integer***) - массив целых чисел, состоящий из 0 и 1, называемый матрицей смежности. *Выходные*: *S* (тип ***integer***) - матрица связей.

Для проверки работы процедур, используя матрицу взаимной корреляции из гл. 7, п. 5.2, табл. 7.10, строится матрица смежности для r_a = 0.05; 0.1. На основании матрицы смежности выполняется анализ связанных групп методом корреляционных плеяд. Для проверки выводов строятся матрица связи и матрица смежности. Выделяются связанные группы. Вычисления выполнялись с точностью 10^{-4}. Их результаты приведены в табл. 8.1.

Таблица 8.1

r_a	Матрица смежности					Матрица связи				
	Итерация первая									
0.05	1.0	0.0	1.0	0.0	1.0	1.0	0.2	0.8	0.2	0.8
	0.0	1.0	0.0	1.0	1.0	0.2	1.0	0.4	0.5	0.4
	1.0	0.0	1.0	1.0	1.0	0.8	0.4	1.0	0.4	0.6
	0.0	1.0	1.0	1.0	0.0	0.2	0.5	0.4	1.0	0.4
	1.0	1.0	1.0	0.0	1.0	0.8	0.4	0.6	0.4	1.0
	Итерация вторая									
	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0
	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0
	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0
	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0
	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0
	Итерация первая									
0.10	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.5	0.3	0.3
	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
	1.0	0.0	1.0	1.0	1.0	0.5	0.0	1.0	0.5	0.5
	0.0	0.0	1.0	1.0	0.0	0.3	0.0	0.5	1.0	0.3
	0.0	0.0	1.0	0.0	1.0	0.3	0.0	0.5	0.3	1.0
	Итерация вторая									
	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0

§ 2. МЕТОД БУРКОВА

С некоторой долей условности к иерархическим методам группирования можно отнести метод Ю.К.Буркова [1973], который широко применяется в геохимии и называется также методом многократной корреляции (ММК). Он хотя дает конечные результаты, имеющие иерархическую структуру, но сам характер классифицирующий процедуры заметно отличается от иерархического.

В ММК объединение элементов в классы производится на основе меры сходства, роль которой играют корреляционные моменты высших порядков. Корреляционные моменты рассчитываются на каждом новом шаге по формуле

$$r_{KL}^{(n+1)} = \frac{\sum_{i=1}^{P}\left(r_{ik}^{(n)} - \bar{r}_k^{(n)}\right)\cdot\left(r_{iL}^{(n)} - \bar{r}_L^{(n)}\right)}{\sqrt{\sum_{i=1}^{P}\left(r_{ik}^{(n)} - \bar{r}_k^{(n)}\right)^2 \cdot \left(\sum\left(r_{iL}^{(n)} - \bar{r}_L^{(n)}\right)\right)^2}},$$

где r_{ik}, r_{iL} - компоненты k-го и L-го столбцов матрицы $\mathfrak{R}^*$, полученной последовательным преобразованием матрицы взаимной корреляции; $\bar{r}_k$ и $\bar{r}_L$ - средние арифметические числа для k-го и L-го столбцов: $\bar{r}_k^{(n)} = 1/p \cdot \sum_{i=1}^{N} r_{ik}^{(n)}$; p - размерность матрицы $\mathfrak{R}^*$. Как следует из приведенной формулы, суть метода Ю.К. Буркова заключается в отыскании таких групп элементов, которые обладают сходным характером связи как между собой (внутри группы), так и с элементами других групп.

Каждому элементу из множества A поставим в соответствие вектор-столбец исходной корреляционной матрицы $\mathfrak{R}$ и обозначим новую матрицу $\mathfrak{R}^{(1)}$. Если теперь вычислим взаимный коэффициент корреляции от элементов матрицы $\mathfrak{R}^{(1)}$, то получим матрицу $\mathfrak{R}^{(2)}$. При этом сами $r_{ik}^{(2)}$ будут превышать исходный коэффициент корреляции $r_{ik}^{(1)}$, если векторы имеют достаточно уверенную взаимосвязь. В противном случае $r_{ik}^{(1)} > r_{ik}^{(2)}$. В целом обнаруживается тенденция: с увеличением количества шагов n величина $r_{ik}^{(n)}$ стремится либо к +1, либо к -1, причем для некоторых групп значения $r_{ik}^{(n)}$ достигают +1 уже при $n < 4$. Если нужно для достижения +1 больше пяти пересчетов матрицы $\mathfrak{R}$, то говорят, что эти элементы не столь тесно связаны в данной группе.

Конечный результат соответствует разбиению матрицы (множества A) на две "антагонистические" ассоциации: элементы a_j, связанные в одной ассоциации, имеют $r^{(n)}$ равное +1, а элементы a_L, связанные в противоположной ассоциации, $r^{(n)}$ равное - 1.

Метод многократной корреляции привлекает простотой алгоритма, но вместе с тем имеет серьезный недостаток. Его нецелесообразно применять в ситуациях, когда есть данные, говорящие о существовании более двух групп связанных элементов, что соответствует действию более чем двух факторов. Однако для прикладных или предварительных расчетов этот метод можно использовать довольно успешно.

Вычислительная процедура, используемая здесь, уже применялась для расчета корреляционной матрицы (глава 7, § 5). Единственное отличие ее состоит в самой вычислительной схеме, т.е. в основной программе. Поэтому для выполнения данной работы можно воспользоваться процедурой расчета корреляционной матрицы без изменений.

Для демонстрации и проверки процедур корреляционная матрица (табл. 8.2) анализируется уже описанным методом. Используются данные из табл. 7.9, по которым ранее строилась корреляционная матрица. Анализ проводится по уровням значимости 0.05 и 0.1. Вычисления выполнены с точностью 10^{-5}. Результат работы приводится в табл. 8.3. Обозначения сохранены, как в главе 7.

Из анализа матрицы корреляционных профилей видно (табл. 8.3, итерации 4 - 6), что все элементы уверенно разделились на две антагонистические группы: к первой можно отнести элементы I, II, III столбцов, а ко второй IV и V (табл. 8.2). Связь элементов II и III не очевидна, поэтому результат нуждается в дополнительной проверке. Аналогичной проверке необходимо подвергнуть группу II и IV, так как корреляционная матрица дала в указанном случае противоположный результат (по связи элементов). Делая вывод, можно предположить следующую группировку элементов: I - III; IV - V; II. Положение элемента II необходимо уточнить другими методами.

Таблица 8.2

№	I	II	III	IV	V
I	1.00000	0.33842	0.58387**	-0.29250	-0.45483*
II	0.33842	1.00000	-0.05452	0.52663*	-0.45056*
III	0.58387**	-0.05452	1.00000	-0.72840**	-0.73536**
IV	-0.29250	0.52663*	-0.72840**	1.00000	0.36006
V	-0.45483*	-0.45056*	-0.73536**	0.36006	1.00000

Таблица 8.3

	I	II	III	IV	V	I	II	III	IV	V	I	II	III	IV	V
	Итерация первая					Итерация третья					Итерация пятая				
I	1.00	0.28	0.89	-0.72	-0.88	1.00	0.49	1.00	-0.98	-1.00	1.00	0.99	1.00	-1.00	-1.00
II	0.28	1.00	0.01	0.39	-0.47	0.49	1.00	0.42	-0.30	-0.55	0.99	1.00	0.99	-0.99	-0.99
III	0.89	0.01	1.00	-0.91	-0.88	1.00	0.42	1.00	-0.99	-0.99	1.00	0.99	1.00	-1.00	-1.00
IV	-0.72	0.39	-0.91	1.00	0.61	-0.98	-0.30	-0.99	1.00	0.96	-1.00	-0.99	-1.00	1.00	1.00
V	-0.88	-0.47	-0.88	0.61	1.00	-1.00	-0.55	-0.99	0.96	1.00	-1.00	-0.99	-1.00	1.00	1.00
	Итерация вторая					Итерация четвертая					Итерация шестая				
I	1.00	0.30	0.99	-0.92	-0.99	1.00	0.83	1.00	-1.00	-1.00	1.00	1.00	1.00	-1.00	-1.00
II	0.30	1.00	0.15	0.11	-0.43	0.83	1.00	0.82	-0.79	-0.85	1.00	1.00	1.00	-1.00	-1.00
II	0.99	0.15	1.00	-0.97	-0.96	1.00	0.82	1.00	-1.00	-1.00	1.00	1.00	1.00	-1.00	-1.00
IV	-0.92	0.11	-0.97	1.00	0.85	-1.00	-0.79	-1.00	1.00	0.99	-1.00	-1.00	-1.00	1.00	1.00
V	-0.99	-0.43	-0.96	0.85	1.00	-1.00	-0.85	-1.00	0.99	1.00	-1.00	-1.00	-1.00	1.00	1.00

§ 3. АНАЛИЗ КОРРЕЛЯЦИОННОЙ МАТРИЦЫ МЕТОДОМ КОРРЕЛЯЦИОННЫХ ПРОФИЛЕЙ

Этот метод известен как *метод корреляционных профилей (МКП) Фишера* или *метод пробных графиков (МПГ)*.

Прежде чем рассматривать метод, введем некоторые критерии, позволяющие достаточно объективно судить о сходстве или различии корреляционных векторов $\mathbf{r}_1$, $\mathbf{r}_2$, ..., $\mathbf{r}_n$, дающих в совокупности матрицу коэффициентов корреляции $\mathfrak{R}$ размерностью $n \times n$. Запишем любую пару сравниваемых векторов в развернутом виде:

$$\mathbf{r}_k = \{r_{kh},\ h = 1, ..., k, ..., L, ..., N\};$$

$$\mathbf{r}_L = \{r_{Lh},\ h = 1, ..., k, ..., L, ..., N\}.$$

Очевидно, что они идентичны лишь в том случае, если соответствующие парные корреляционные коэффициенты совпадают и по величине и по знаку. На практике решение о равенстве r_{kh} и r_{Lh} приходится принимать, опираясь на их выборочные значения, которые являются, как уже неоднократно отмечалось, лишь приближенными оценками истинных коэффициентов корреляции.

Поэтому вывод относительно векторов $\mathbf{r}_k$ и $\mathbf{r}_L$ формулируются в форме статистических гипотез:

$$H_0 : r_{kh} = r_{Lh}; \qquad H_1 : r_{kh} \neq r_{Lh};\ h \neq k;\ h \neq L.$$

Если хотя бы для одного из возможных значений h нулевая гипотеза H_0 будет отвергнута для уровня зависимости L, то принимается решение о существенном различии векторов $\mathbf{r}_k$ и $\mathbf{r}_L$. А так как распределение выборочных коэффициентов корреляции в условиях $r_0 \neq 0$ обладает значительной асимметрией, то для проверки гипотезы H_0 используют не сами коэффициенты корреляции, а их преобразованные с помощью критерия Фишера значения

$$z_{kL} = \frac{1}{2} \cdot \ln\left(\frac{1 + r_{kL}}{1 - r_{kL}}\right), \forall\ k \neq L.$$

В условиях нулевой гипотезы статистика $(z_{kh} - z_{Lh}) \cdot s \cdot \sqrt{2}$ распределена асимптотически нормально с нулевым средним и дисперсией, равной единице. Величина s в последнем выражении является выборочной оценкой стандартного отклонения z. Она зависит только от объекта выборки M. Так как $M_k = M_L\ \forall\ k$ и L, то $s_k = s_L = s$.

Запишем критерий для проверки гипотезы H_0:

$$P\left\{\frac{|z_{kh} - z_{Lh}|}{s \cdot \sqrt{2}} < c\right\} \approx 1 - \alpha,$$

где P - вероятность выполнения неравенства в скобках; α - уровень значимости; c - константа, которая с учетом асимптотически нормального распределения статистики может быть найдена по соответствующим таблицам. Например, для $\alpha = 0.05$ $c = 1.46$. Таким образом, гипотеза о равенстве $\mathbf{r}_k$ и $\mathbf{r}_L$ не противоречит выборочным данным, если при

$$\alpha = 0.05 \quad |z_{kh} - z_{Lh}| < 1{,}96 \cdot S \cdot (2)^{1/2} \sim 2{,}77,$$

где S можно определить с использованием:

а) обычных коэффициентов корреляции и коэффициентов, основанных на нормальных метках,

$$S = 1/(M - 3)^{1/2};$$

б) коэффициента корреляции Стирлинга

$$S = 1.03/(M - 3)^{1/2};$$

в) коэффициента корреляции Кенндалла

$$S = 0.66/(M - 3)^{1/2}.$$

Рассматриваемый здесь метод относится к самым простым методам поиска ассоциаций, легко реализуется на ЭВМ любого класса, работает в режиме реального време-

ни. А если составить таблицу соответствия z_{kh} коэффициентам r_k , то тогда его несложно применить и для "ручной" обработки матриц $\mathfrak{R}$, даже значительного размера.

Анализ полученной матрицы $\boldsymbol{Z}$ можно выполнять двумя способами. Первый относится к аналитическим и основан на анализе максимальных расхождений между векторами $\mathbf{z}_k$ и $\mathbf{z}_L$. По этому способу вычисляют $|z_{kh} - z_{Lh}|$, где $k \neq L$, и отбирают для анализа лишь максимальные. Так как матрица $\boldsymbol{Z}$ симметрична, то достаточно это выполнить лишь для одной ее половины. Тогда вторую половину матрицы можно использовать для результатов проверки гипотез H_0 и H_1, после сопоставления вычисленных **max**$|z_{kh} - z_{Lh}|$ с предельно допустимой в условиях ненулевой гипотезы величины отклонения α.

Второй способ оказывается более удобным для "ручной" обработки полученных матриц (даже больших размерностей) - это графический вариант МКП и в отечественной литературе он получил название "*метод пробных графиков*". Для каждого из классифицируемых элементов строится график в координатах ось абсцисс - номер элемента, ось ординат - значение z_{kh}. Таким образом, каждому элементу z_k матрицы $\boldsymbol{Z}$ на графике соответствует совокупность точек N - 1 (не вычисляются диагональные элементы z_{nn}). Если точки соединить ломаной, то получим корреляционный профиль элемента z_k. Даже беглое визуальное сопоставление этих профилей позволяет быстро скомпоновать группы согласующихся профилей. А дальнейшее уточнение групп элементов может быть выполнено любым из рассмотренных аналитических способов.

Основным недостатком МКП является ошибочное объединение в одну группу элементов z_k с невысокими r_{ij}, различие между которыми ниже заданного статистического критерия. Поэтому окончательные результаты классификации должны быть откорректированы с привлечением других аналитических методов.

Для выполнения данной работы следует воспользоваться процедурами из гл. 7, § 1, вычисляя критерий Фишера в основной программе; M следует положить равным количеству испытаний (для данного случая - 20).

Далее выполнен анализ корреляционной матрицы, полученной по экспериментальным данным, взятым из табл. 7.9, а результаты анализа МКП (табл. 8.4) сравниваются с выводами § 1 и 2 настоящей главы.

Таблица 8.4

№	I	II	III	IV	V
I	***	0.352	0.668	-0.301	-0.491
II	0.352	***	-0.055	0.585	-0.485
III	0.668	-0.055	***	-0.925	-0.940
IV	-0.301	0.585	-0.925	***	0.377
V	-0.491	-0.485	-0.940	0.377	***

Анализируя последнюю матрицу, получаем, что можно выделить элементы групп I - III, IV - V, а элемент II, как и предполагалось в § 1 и 2, образует отдельную группу.

§ 4. МЕТОД ГЛАВНЫХ КОМПОНЕНТ (МНОГОФАКТОРНЫЙ АНАЛИЗ)

Одно из центральных мест в теории факторного анализа занимает *метод главных компонент* (МГК). Его основная идея может быть записана системой

$$X_i = \sum_{j=1}^{N} w_{ij} \cdot f_j + e_i \ , \quad i = 1,..., M; N < M,$$

где f_j - простой j-й фактор (f_j независимы друг от друга и погрешностей измерения); N - заданное число простых (начальных) факторов; e_i - остаточный член с дисперсией $d\,(e_i)$, действующей только на X_i; w_{ij} - коэффициенты, называемые нагрузкой i-й случайной величины на j-й фактор. В матричной записи эту систему можно представить

$$\vec{\boldsymbol{X}} = \vec{\boldsymbol{W}} \cdot \vec{\boldsymbol{F}} + \vec{\boldsymbol{E}} \ .$$

Максимально возможное количество факторов m при заданной величине N определяется неравенством $(N + m) > (N - m)$, которое, безусловно, должно выполняться, чтобы задача не выродилась в тривиальную.

Основная теорема (в вольном изложении) факторного анализа сводится к следующему. Допустим, что исходные величины X_1 и X_2 имеют один фактор f_i , тогда $r_{12} = w_{ij} \cdot w_{2j}$. В общем случае, когда N величин имеют m факторов, коэффициент корреляции может быть представлен как

$$r_{12} = \sum_{j=1}^{M} w_{1j} \cdot w_{2j} \ ,$$

т.е. коэффициент корреляции любых двух независимых величин выражается суммой произведений коэффициентов (нагрузок) некоррелированных факторов.

Построив матрицу $\boldsymbol{W}$ размерностью $N \times m$, элементами которой служат w_{ij} , получим основную теорему анализа в следующем виде: $\mathfrak{R} = \boldsymbol{W} \cdot \boldsymbol{W}'$, где $\boldsymbol{W}'$ - матрица, транспонированная к $\boldsymbol{W}$. Таким образом, задача сводится к линейному преобразованию N-мерного пространства в m-мерное. Эта задача не решается однозначно, так как представить корреляционную матрицу факторами можно бесконечным числом способов, например вращением на разный угол. Но к новым случайным величинам модели

$$X_i = \sum_{j=1}^{N} w_{ij} \cdot f_j + e_i \, , i = 1,..., M$$

можно перейти, ориентируясь на поведение дисперсий. При этом ищут некоррелированные комбинации вида

$Y_j = \sum_{i=1}^{m} w_{ij} \cdot X_i$. Дисперсии Y_j располагают в убывающем порядке, т.е. $\sigma^2(Y_1) > \sigma^2(Y_2) > ... > \sigma^2(Y_m)$.

Корреляционная матрица оказывается расщепленной на M ортогональных компонент. Если при этом $m = = N$, то матрица $\mathfrak{R}$ в факторном анализе будет эквивалентна методу главных компонент.

На практике в предположении малости σ^2 для e_i изучают корреляционную матрицу $\mathfrak{R}$ для исходных величин $\boldsymbol{X}$.

Как уже отмечалось, матрицу $\mathfrak{R}$ можно связать с матрицей факторных нагрузок: $\mathfrak{R} = \boldsymbol{W}\cdot\boldsymbol{W}'$.

Учитывая, что $\mathfrak{R}$ - симметричная положительно определенная матрица, представим ее в виде $\mathfrak{R} = \boldsymbol{U\Lambda U}'$, где $\boldsymbol{U}$ - ортогональная матрица собственных векторов матрицы $\mathfrak{R}$, а $\boldsymbol{\Lambda}$ - диагональная матрица собственных чисел матрицы $\mathfrak{R}$. Тогда, сравнивая два последних равенства, имеем $\boldsymbol{W} = \boldsymbol{U}\cdot\boldsymbol{\Lambda}^{1/2}$.

Теперь, учитывая, что $\boldsymbol{Y} = \boldsymbol{WX}$, можно вычислить матрицу $\boldsymbol{Y}$ главных компонент, для которой λ_i являются дисперсиями соответствующих компонент.

Система собственных векторов $\boldsymbol{U}$ является ортогональной, а из ортогональности следует их некоррелированность. Иными словами, корреляционная матрицы $\mathfrak{R}$ оказывается в результате расщепленной на N ортогональных некоррелированных компонент.

Традиционно λ_i располагают в порядке убывания, т.е. λ_1 соответствует самое большое значение из λ_i, которое называют первой главной компонентой. Второе значение λ_2 - второй главной компонентой. Метод построения комбинацией вида $\boldsymbol{Y} = \boldsymbol{WX}$ называется *компонентным анализом,* или *методом главных компонент.*

Геометрически определение главных компонент приводит к новой ортогональной системе координат. Причем первая координатная ось вычисляется таким образом, чтобы соответствующая ей линейная форма извлекала возможно большую дисперсию. Далее находится ортогональная этой форме ось, которая делает то же самое с оставшейся дисперсией. И так далее, т.е. в N-мерном пространстве величин $X_1,\ldots, X_N$ ось наибольшей протяженности N-мерного эллипсоида рассеяния X_N определится направляющими косинусами, равными компонентам вектора $\boldsymbol{W}$.

Проекция X на направление w_1 имеет наибольшую дисперсию по сравнению с их проекциями на другие направления. Кроме того, учитывая ортогональность системы $\boldsymbol{W}$, можно перейти к старым координатам $X_i = \sum_{j=1}^{M} w_{ij}\cdot Y_j$, где Y_j - j-я главная компонента; w_{ij} - вес i-й компоненты в j-й случайной величине.

Последнее соотношение - основное в МГК. Оно не содержит остаточной составляющей ε и получается, что все N_j главных компонент исчерпывают всю дисперсию исходных данных. В МГК поэтому нет необходимости делать какие-либо предположения, величины X_i даже не обязательно считать случайными. Единственным недостатком рассматриваемого метода является то, что главные компоненты неинвариантны относительно изменения масштаба тех шкал, по которым отсчитываются разные случайные величины.

Именно поэтому анализ МГК целесообразно использовать тогда, когда все X_i измерены в одних и тех же единицах. Для этого обычно берут либо нормированные данные $\left(X_{ij} - \bar{x}_i\right) / \delta_i$, либо приводят экспериментальные данные к нормальному распределению. Однако надо заметить, что ни первый, ни второй приемы не имеют строгого математического обоснования.

На практике обычно для оценки количества независимых факторов оставляют только те λ_i, которые в сумме дают 90 - 95 % от всех λ_i. Количество оставшихся собственных значений и дает оценку числа независимых факторов. Чаще всего сохраняют все $\lambda_i > 1$, т.е. только дающие наибольший вклад в дисперсию.

Алгоритм, реализующий МГК, может быть записан следующей вычислительной схемой:

1. Вычислить матрицу взаимных корреляций $\mathfrak{R}$ для экспериментальных данных X.
2. Пользуясь любым методом, вычислить собственные значения матрицы $\mathfrak{R}(\lambda_i)$.
3. Определить матрицу факторных нагрузок $\boldsymbol{W}$.
4. Найти матрицу главных компонент $\boldsymbol{Y}$.
5. Выделить y_i, соответствующие λ_i, которые в сумме дают не менее 95 % от всей суммы $\sum_{i=1}^{N} \lambda_i$.
6. Пересчитать X_i в соответствии с оставшимися факторами, получить новую матрицу $\boldsymbol{X}^{(1)}$.
7. Проанализировать матрицы $\boldsymbol{W}$, $\boldsymbol{Y}$ и новую матрицу $\boldsymbol{X}^{(1)}$ с целью выделения групп.

При программировании МГК целесообразно пункты1, 2 и 3 оформить отдельными процедурами. Матрицу $\mathfrak{R}$ лучше вычислять из ковариационной матрицы для уменьшения вычислительной погрешности.

По процедуре CORR вычисляется матрица $\mathfrak{R}$ взаимных корреляций (взять из гл. 7, § 5).

Собственные значения матрицы $\mathfrak{R}$ все положительны. Это вытекает из свойств самой матрицы (эрмитова матрица). Поэтому вычислять λ_i можно, пользуясь методами квадратных корней, Крылова или же Данилевского, которые дают удовлетворительную точность, но, чтобы не увеличивать вычислительную погрешность, рекомендуется выполнять вычисления с шестью или семью знаками.

Далее приведем краткое описание процедур для облегчения понимания алгоритма.

В программе для определения собственных значений применена процедура, написанная по эффективному алгоритму Хаусхолдера [Уилкинсон, Райнш, 1976], который использует симметричность матрицы $\mathfrak{R}$ (процедуры TRED1 и TRBAK1 выполняют вычисления собственных значений матрицы $\mathfrak{R}$ по методу Хаусхолдера). Основное назначение этого алгоритма состоит в том, чтобы заменить вычисление собственных значений произвольной симметрической матрицы вычислением собственных значений трехдиагональной матрицы.

Процедура TRED1 приводит действительную симметрическую матрицу $\boldsymbol{A}$ к симметрической трехдиагональной форме $\boldsymbol{A}_{n\text{-}1}$* с помощью преобразования Хаусхолдера. Эту процедуру можно использовать также для матриц с кратными или очень близкими собственными значениями.

Процедура TQL1 предназначена для определения всех собственных значений трехдиагональной симметрической матрицы. Если трехдиагональная матрица получена из исходной преобразованием Хаусхолдера, то данная процедура определяет непосредственно и собственные векторы матрицы $\boldsymbol{A}_{n\text{-}1}$ без предварительного вычисления собственных векторов исходной матрицы. Вычисленные векторы всегда ортонормированы с точностью, определяемой точностью используемой ЭВМ.

Процедура TRBAK1 предназначена для восстановления совокупности собственных векторов матрицы $\boldsymbol{A}$, пронумерованных от $M1$ до $M2$, если известны собственные векторы матрицы $\boldsymbol{A}_{n\text{-}1}$. Это преобразование выполняют только после первых двух процедур с использованием промежуточных результатов, полученных при приведении матрицы $\boldsymbol{A}$ к виду $\boldsymbol{A}_{n\text{-}1}$.

По процедуре GAUSS выполняется расчет обратной матрицы $\boldsymbol{A}$ размером $N\times N$.

Остальные процедуры играют вспомогательную роль и обеспечивают вывод результатов счета на экран, печатающее устройство или магнитый диск в файл с заданным именем. Есть также процедура, которая в графическом режиме позволяет увидеть расположение элементов исходной матрицы в факторных осях.

Формальные параметры.

Процедура TRED1. *Входные*: N (тип ***integer***) - размер квадратной матрицы $N\times N$, для которой вычисляются собственные значения; *tol* (тип ***real***) - константа, зависящая от свойств конкретной вычислительной системы; A (тип ***real***) - массив размером $N\times N$ для размещения элементов симметрической матрицы, для которой вычисляются собственные значения. *Выходные*: A (тип ***real***) - массив размером $N\times N$, в котором только поддиагональные элементы содержат информацию о преобразовании Хаусхолдера, элементы верхнего треугольного массива определяют исходную матрицу A; D (тип ***real***) - массив размером $N\times 1$, содержащий диагональные элементы трехдиагональной матрицы $\boldsymbol{A}_{n\text{-}1}$; E - массив размером $N\times 1$, N - 1 элементов которого от $E[2]$ до $E[N]$ определяют внедиагональные элементы трехдиагональной матрицы $\boldsymbol{A}_{n\text{-}1}$. Первый элемент массива $E[1]$ равен нулю; $E2$ (тип ***real***) - массив, содержащий служебную информацию для дальнейших расчетов.

Процедура GAUSS. _*Входные*: N (тип ***integer***) - размер квадратной матрицы $N\times N$; *AAA* (тип ***real***) - исходная квадратная матрица $\boldsymbol{A}$. *Выходные*: *AA* (тип ***real***) - матрица, обратная данной.

Процедура TQL1. *Входные*: N (тип ***integer***) - порядок трехдиагональной матрицы $\boldsymbol{A}_{n\text{-}1}$; *massheps* (тип ***real***) - константа, зависящая от свойств конкретной вычислительной системы, представляет наименьшее число, для которого еще выполняется условие 1 + *massheps* > 1; D (тип ***real***) - массив размером $N\times 1$, содержащий диагональные элементы трехдиагональной матрицы $\boldsymbol{A}_{n\text{-}1}$; E - массив размером $N\times 1$, N - 1 элементов которого от $E[2]$ до $E[N]$ определяют внедиагональные элементы трехдиагональной матрицы $\boldsymbol{A}_{n\text{-}1}$. Первый элемент массива $E[1]$ равен нулю. Надо заметить, что он вообще не используется и может быть произвольным. Массив E в программе используется как рабочий. *Выходные*: D (тип ***real***) - массив размером $N\times 1$, содержащий N собственных значений матрицы $\boldsymbol{A}_{n\text{-}1}$, расположенных в порядке их возрастания; E (тип ***real***) - массив используется как рабочий для хранения промежуточных результатов; *ZA* (тип ***real***) - массив размером $N\times N$, в котором предварительно записывается единичная матрица, если надо вычислять собственные векторы трехдиагональной матрицы.

Процедура TRBAK. *Входные*: N (тип ***integer***) - порядок действительной симметрической матрицы $\boldsymbol{A}$; $M1$ и $M2$ (тип ***integer***) - граничная пара обозначения массива нормированных собственных векторов трехдиагональной матрицы $\boldsymbol{A}_{n\text{-}1}$, определенной в процедуре TRED1; *AZ* (тип ***real***) - массив размером $N\times N$, в котором существенны лишь поддиагональные элементы (соответствует выходному массиву A процедуры TRED1); E (тип ***real***) - массив размером $N\times 1$, N - 1 элементов которого от $E[2]$ до $E[N]$ определяют внедиагональные элементы трехдиагональной матрицы $\boldsymbol{A}_{n\text{-}1}$. Первый элемент массива $E[1]$ равен нулю. *Выходные*: *ZA* (тип ***real***) - массив размером $N \times (M2 - M1 + 1)$, содержащий нормированные собственные векторы матрицы $\boldsymbol{A}$, пронумерованные от $M1$ до $M2$.

Остальные процедуры не имеют прямого отношения к рассматриваемой задаче, выполняют вспомогательную роль, указанную в комментариях к каждой процедуре.

В заключение следует отметить, что МГК - сложный с вычислительной точки зрения метод и требует максимума внимательности. Программа, реализующая МГК, довольно сложная и трудоемкая. Один расчет матрицы 25×60 занимает до 20 минут на ЭВМ типа IBM-286 с частотой 12 МГц. Ввиду сложности текст программы приводится полностью. Для настройки программы перед запуском следует указать полный путь, где находятся графический драйвер, файл с исходными данными, и определить значения следующих констант:

PR - если $PR = 0$, то результаты будут записываться в файл на магитный диск C: с именем *BURKOV*; если $PR = 1$, то результаты будут выводится на печатающее устройство; если $PR = 2$, то на экран дисплея;

START - строковая константа, указывающая полное имя файла, в котором хранятся исходные данные;

N - количество параметров, которые измеряют в ходе эксперимента, определяет количество столбцов в матрице данных;

M - количество измерений параметров, определяет количество строк в матрице данных.

В матрице исходных данных первые три строки содержат служебную информацию:

название массива, где и когда получены данные;

фамилию и имя исследователя;

условные названия измеряемых параметров (лучше, если название будет состоять из одного или двух символов. На каждое имя обязательно должно отводиться по 3 позиции).

Для проверки работоспособности своей программы рекомендуется выполнить приведенный как образец тес-

товый расчет, если результаты совпадут (до 6-го знака), то можно выполнять расчеты по своим данным.

```
{*****  ПРОГРАММА РАСЧЕТА
     " МЕТОД ГЛАВНЫХ КОМПОНЕНТ " ***** }
PROGRAM BURCORR;
USES CRT, DOS, GRAPH;
CONST   PR  = 0; START = ' '; N     = 5;       M     = 20;
        EPS = 0.00000001; EPS1 = 2E-32; EPS2 = EPS1;
LABEL 50;
TYPE MAS=ARRAY [1..N] OF REAL;
             MAS2 = FRRAY [1..N] OF MAS;
             MAS1 = ARRAY [1..N] OF MAS;
             STRING1=STRING[3];
             FILEMAS = FILE OF MAS;
             MAS3 = ARRAY [1..N] OF STRING1;
VAR Z,I,IMAX,J,K,K1,LON,N1,A1,A2,A3,GMY,
             GMX,DX,DY,DELTA,XI, YI : INTEGER;
             MNR1,LLON: STRING [1]; FT,FF,LST: TEXT;
             CH,CHX,CHY : CHAR;
             FM, COV, COVS, F1, F2, F3 : FILEFAS;
             SS,SI,SC,SX,X,D,E,E2,Y,ZZ: MAS;
             SVEC,A,AA,RRCC: MAS1;
             SUM,SUM1,SUM2,XMIN,TOL: REAL; RRC: MAS2;
             RR :-10..10; DN : BOOLEAN;
             MNR : STRING1; XCOL,MINER:MAS3;
             STR1, STR2, STRNG : STRING;
             GRAPHDRIVER, GRAPHMODE,
             GRAPHREZULT : INTEGER;

{$F+}
PROCEDURE GAUSS (N:INTEGER; AAA:MAS1;
             VAR AA : MAS1);
(;□□□□□□□□□ □□□□□□□□□ □□□□□□□□□ □□□□□□□□□□ □□□□□□□ □□□ □□□□□□□□
N □ N □□□□□□□ □□□□□□. □□□□□□□□ □□□□□□□ □□□□□□□□□□□□□□□ □ □□□□□□□ □□
;)
VAR I,J,J1,K : INTEGER; A:MAS1;
             B,X:MAS; SUM,AIJ,AII : REAL;
BEGIN
  A:= AAA;          (; □□□□□□□□□□□□□ □□□□□□□□□□ □□□□□□□□ ;)
  FOR J1 := 1 TO N DO
  BEGIN
     FOR I := 1 TO N DO              B[I] := 0.0;
          (; □□□□□□□□□□□□□□□ □□□□□□□ □□ □□□□□□□ □□□□□□□ ;)
     B[J1] := 1.0;
     A := AAA;
     X := B;
     FOR I := 1 TO N DO
     BEGIN
        AIJ := A[I,I];
        B[I] := B[I] / AIJ;
        FOR J := I TO N DO
           A[I,J] := A[I,J] / AIJ;
        IF I<>N THEN
        BEGIN
           FOR J := I+1 TO N DO
           BEGIN
              AII := A[J,I];
              B[J] := B[J] - B[I]*AII;
              FOR K := I TO N DO
                 A[J,K] := A[J,K] - A[I,K] * AII;
           END;
        END;
     END;
     X[N] := B[N];
     FOR I := N-1 DOWNTO 1 DO
     BEGIN
        SUM := B[I];
        FOR J := I+1 TO N DO SUM := SUM - X[J] * A[I,J];
        X[I] := SUM;
     END;
     FOR I := 1 TO N DO        AA[I,J1] := X[I];
  END;
END;

{$F+}
PROCEDURE TQL1 (N:INTEGER;MASHEPS:REAL;
             VAR D,E:MAS; VAR ZA:MAS1);
(; □□□□□□□□□□ □□□□□□□□□□□ □□□□□□□□ □ □□□□□□□□□□□  □□□□□□□□ □
□□□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□□ ZTZ, □□□ □□□□□□□□□□□□□□□□
□□□□□□□□ T □□□□□□□ □□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□□□□□, □□□□□□□□□□□ □
□□□□□□□□ D[1:N], □ □□□□□□□□□□□□□□□□□□ □□□□□□□□□□□, □□□□□□□□□□□ □
□□□□□□□□□□ N - 1 □□□□□□□□ □□□□□□□□ □[1:N]. □□□□□□□□□□□ □□□□□□□□□□□
□□□□□□□□□ □□□□□□□□□□□□□ □ □□□□□□□□ □□□□□□□□□□□□□□ □ □□□□□□□□ D □□ □□□□□□
□□□□□□□□□□□□□ □□□□□□□□□□.
   □□□□□□□□□□□ □□□□□□□□ □□□□□□□□□□□□ □ □□□□□□□□ Z [1:N, 1:N] □□ □□□□□□
□□□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□ Z.  □□□ □□□□□□□ □□□□□□□□□□
□□□□□□□□□□□ □□□□□ □□ □□□□□□□□ EXIT, □□□□□ □□□□ □□□□□□□□□□□□□
□□□□□□□□□□□□□ □□□□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□ 30 □□□□□□□□□ ;)
LABEL CONT, ROOT,NEXTIT;
VAR I, J, K, L, M : INTEGER; B, C, F, G, H, P, R, S : REAL;
BEGIN
  FOR I := 2 TO N DO
     E[I-1] := E[I];
  FOR I := 1 TO N DO
     FOR J := 1 TO N DO
        IF I=J THEN              ZA[I,J] := 1.0
        ELSE             ZA[I,J] := 0.0;
  E[N] := 0.0;
  B:= 0.0;
  F := 0.0;
(; □□□□□□□□□□□ □□□□□□□ □□□□□□□□□□□ □□□□□□. □□□□□□□□□□□ □□□□□□ □□□□□□□□
□□ □□□□□ □□□□□□□□□□□□□□ □□□ ;)
  FOR L := 1 TO N DO
  BEGIN
     J := 0;
     H := MASHEPS * (ABS(D[L])+ABS(E[L]));
     IF B<H THEN
        B:= H;
(;□□□□□□□□□□□ □□□□□□□□□□□□□□ □□□□□□□ □□□□□□□□□□□□□□□□□□ □□□□□□□□□.
□□□□□□□□ □□ □□□□□ □□□□□□□□□□□□□□ □□□□;)
     FOR M := L TO N DO
        IF ABS(E[M])<=B THEN
           GOTO CONT;
```

```
CONT:
        IF M=L THEN
            GOTO ROOT;
NEXTIT:
        IF J=30 THEN EXIT;
        INC(J);
(; □□□□□□□□□□□□ □□□□□□ ;)
        G := D[L];
        P:= (D[L+1]-G) / (2*E[L]);
        R:= SQRT (P*P+1.0);
        IF P<0.0 THEN
            D[L] := E[L]/(P-R)
        ELSE
            D[L] := E[L]/(P+R);
        H := G - D[L];
        FOR I := L+1 TO N DO
            D[I] := D[I] - H;
        F := F + H;
(; QL - ПРЕОБРАЗОВАНИЕ ;)
        P:= D[M];
        C := 1;
        S := 0;
        FOR I := M-1 DOWNTO L DO
        BEGIN
            G := C*E[I]; H := C*P;
            IF ABS(P)>=ABS(E[I]) THEN
            BEGIN
                C := E[I]/P;
                R := SQRT(C*C+1);
                E[I+1] := S*P*R;
                S := C / R;
                C := 1.0 / R;
            END
            ELSE
            BEGIN
                C:= P/E[I];
                R := SQRT(C*C+1);
                E[I+1] := S*E[I]*R;
                S := 1.0/R; C := C / R;
            END;
            P := C*D[I] - S*G;
            D[I+1] := H + S*(C*G + S*D[I]);
(; □□□□□□□□□□□□ □□□□□□□□□□□□ □□□□□□□ ;)
            FOR K := 1 TO N DO
            BEGIN
                H := ZA[K,I+1];
                ZA[K,I+1] := S*ZA[K,I] + C*H;
                ZA[K,I] := C*ZA[K,I] - S*H;
            END;
        END;
        E[L] := S*P;
        D[L] := C*P;
        IF ABS(E[L])>B THEN
            GOTO NEXTIT;
ROOT:
        D[L] := D[L] + F;
    END;
            (; □□□□□□□□□□□ □□□□□□□□□□□□ □□□□□□□□□
                □ □□□□□□□□□□□□ □□□□□□□□□;)
        FOR I := 1 TO N DO
      BEGIN
        K := I;
        P := D[I];
        FOR J := I+1 TO N DO
            IF D[J] < P THEN
            BEGIN
                K:=J;
                P:=D[J];
            END;
        IF K<>I THEN
        BEGIN
            D[K] := D[I];
            D[I] := P;
            FOR J := 1 TO N DO
            BEGIN
                P := ZA[J,I];
                ZA[J,I] := ZA[J,K];
                ZA[J,K] := P;
            END;
        END;
      END;
END;

{$F+}
PROCEDURE TRBAK (N,M1,M2:INTEGER;AZ:MAS1;
                 E:MAS;     VAR ZA:MAS1);
(; □□□□□□□□□ □□□□□□□□□□□ □□□□□□□□ □□□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□,
□□□□□ □□□□□□□□ □□□□□□□□□□□ □□□□□□□ □□□□□□□□□□□□□□□□□ □□□□□□□,
□□□□□□□□□□□ □ □□□□□□□ Z[1:N,M1:M2]. □□□□□□□□□ □□□□□□□ □□□□□□□□□□□□□
□□ □□□□□ □□□□□□□. □□□□□□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□□□□□□□□□ □□□□□□□
□□□□□□□□□ □ □□□□□□□ □ [1:N], □ □□□□□□□□ □□□□□□□ □□□□□□□□□□□□□□□□
□□□□□□□□ □□□□□□ □□□□□□□□□□□□ □□□□□□□ □ [1:N, 1:N]. □□□□□ □□□□□□□□□□□
□□□□□□□□□□□□□□ □□□□□□□□□□□ TRED1. □□□□ □□□□□□□□ Z □□□□□□□□□□□
□□□□□□□□□□□□□ □□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□□ Z, □□ □□ □□□□□□□□□□□□
□□□□□ □□□□□□□□□□□□□□ □□□□□□□ Z□Z=Z(INPUT)□Z(INPUT) ;)
VAR I,J,K,L : INTEGER; H,S : REAL;
BEGIN
  FOR I := 2 TO N DO
  IF E[I]<>0 THEN
  BEGIN
    L := I-1;
    H := E[I]*AZ[I,I-1];
    FOR J := M1 TO M2 DO
    BEGIN
      S := 0.0;
      FOR K := 1 TO L DO   S := S + AZ[I,K]*ZA[K,J];
      S := S/H;
      FOR K := 1 TO L DO   ZA[K,J] := ZA[K,J] + S*AZ[I,K];
    END;
  END;
END;

{$F+}
PROCEDURE PRNGRAF (FACTX,FACTY:CHAR;
                   VAR JF: INTEGER);
```

```
(; [illegible], [illegible] ([illegible], [illegible]) [illegible]. [illegible]. [illegible]. [illegible]. [illegible]. [illegible], [illegible]. 132 [illegible]. [illegible]
90 [illegible], [illegible] ;)
CONST  A1 = '|';   A2 = ' ';   A3 = '-';   A4 = '/';
       A5 = '\';   A6 = '>';   A7 = '.';   A8 = '*';   S1 = ' ';
LABEL CONT;
VAR SX,SY :ARRAY[1..N] OF REAL;
            X :ARRAY[1..102] OF REAL;
            Y: ARRAY [1..52] OF REAL;
            STX:ARRAY[1..102]OF STRING [1];
            I,J,FX,FY,K,K1,KX,KM,KK,KMN,IND:INTEGER;
            MNR : STRING [1];
            MIN : ARRAY [1..30] OF STRING [1];
BEGIN
  VAL (FACTX,FX,JF);
  IND := 0;
  IF JF=0 THEN
     VAL (FACTY,FY,J)
  ELSE
     GOTO CONT;
  IF J <> 0 THEN
  BEGIN
     JF := J;
     GOTO CONT;
  END;
  FOR I :=1 TO N DO
  BEGIN
     SX[I] := SVEC [FX,I];
     SY[I] := SVEC [FY,I];
  END;
  FOR I := 1 TO 101 DO       X[I] := I / 50.0 - 1.0;
  FOR I := 1 TO 51 DO     Y[I] := 1.0 - I / 25;
  FOR J := 1 TO 30 DO     MIN[J] := A2;
  FOR J := 1 TO 102 DO
     CASE J OF
        49: STX [J] := A4;
        50: STX [J] := A5;
        52: STX [J] := 'F';
        53: STX [J] := FACTY;
     ELSE
        STX[J] := A2;
     END;
  FOR J :=1 TO 102 DO
  BEGIN
     WRITE (LST,STX[J]);
     STX[J] := A2;
  END;
  WRITELN (LST,' ');
  FOR I :=1 TO 50 DO
  BEGIN
     IF ((I DIV 5 * 5) = I) OR (I=1) THEN
     FOR J := 1 TO 102 DO
        IF J=50 THEN          STX[J] := A1
        ELSE
           IF (J DIV 2 * 2) = J THEN          STX[J] := A2
           ELSE               STX[J] := A7;
     IF I=25 THEN
     BEGIN
        FOR J :=2 TO 101 DO        STX[J] := A3;
        STX[101] := A6;
     END;
     IF I = 26 THEN
     BEGIN
        STX[101] := 'F';
        STX[102] := FACTX;
     END;
     STX[1]:=A7;
     KMN := 1;
     FOR K := 1 TO 10 DO
     IF K =5 THEN          STX[K*10] := A1
     ELSE          STX[K*10] := A7;
     IF I=1 THEN
     BEGIN
        K1 := 0;
        KM := 0;
        FOR K := 1 TO N DO
        BEGIN
           IF SY[K]>Y[I] THEN    K1 := K;
           IF K1>0 THEN
           BEGIN
              KX :=0;
              INC(KM);
              KK:=1;
              IF SX[K1]<X[KK] THEN    KX := 1
              ELSE
              BEGIN
                 REPEAT
                    IF(SX[K1]>=X[KK])AND
                       (SX[K1]<X[KK+1]) THEN
                         KX := KK;
                    INC(KK);
                 UNTIL (KK=100) OR (KX<>0);
                 IF KX=0 THEN
                    KX := 100;
           END;
           FOR KK := KX TO KX+2 DO
           IF (STX[KK]=A1) OR (STX[KK]=A3)
              OR (STX[KK]=A2) OR (STX[KK]=A7) THEN
                 STX[KK] := COPY (MINER[K1],KK-KX+1,1)
           ELSE
           BEGIN
              STX[KK] := A8;
              IND := 1;
           END;
           FOR KK := KX TO KX+2 DO
           BEGIN
              MNR := COPY (MINER[K1],KK-KX+1,1);
              IF MNR<>A2 THEN
              BEGIN
                 MIN[KMN] := MNR;
```

```
                IF KMN<31 THEN INC (KMN);
            END;
        END;
        K1 := 0;
    END;
END
ELSE
BEGIN
    K1 := 0;
    KM := 0;
    FOR K :=1 TO N DO
    BEGIN
        IF (SY[K]>=Y[I+1]) AND (SY[K]<Y[I]) THEN
            K1 := K;
        IF K1>0 THEN
        BEGIN
            KX := 0;
            INC (KM);
            KK:=1;
            IF SX[K1]<X[1] THEN
                KX := 1
            ELSE
            BEGIN
                REPEAT
                    IF(SX[K1]>=X[KK])AND (SX[K1]<X[KK+1])
                        THEN                    KX := KK;
                    INC(KK);
                UNTIL (KK=100) OR (KX<>0);
                IF KX=0 THEN
                    KX:=100;
            END;
            FOR KK := KX TO KX+2 DO
            IF (STX[KK]=A1) OR (STX[KK]=A3)
                OR (STX[KK]=A2) OR (STX[KK]=A7) THEN
                    STX[KK] := COPY (MINER[K1],KK-KX+1,1)
            ELSE
            BEGIN
                STX [KK] := A8;
                IND := 1;
            END;
            FOR KK := KX TO KX+2 DO
            BEGIN
                MNR := COPY (MINER[K1],KK-KX+1,1);
                IF MNR<>A2 THEN
                    BEGIN
                        MIN[KMN] := MNR;
                        IF KMN<31 THEN
                            INC (KMN);
                    END;
                END;
                K1 := 0;
            END;
        END;
    END;
    FOR K :=1 TO 102 DO
    BEGIN
        WRITE (LST,STX[K]);
        STX[K] := A2;
    END;
    IF KMN<=28 THEN         WRITE (LST,' ');
    FOR K := 1 TO KMN-1 DO
    BEGIN
        IF IND=1 THEN       WRITE (LST,MIN[K]);
        MIN[K] := A2;
    END;
    IF IND =1 THEN      WRITELN (LST,MIN[KMN])
    ELSE            WRITELN (LST,' ');
    MIN [KMN] := A2;
    IND := 0;
  END;
  J := 1;
  K1 := 10;
  IF K1>N THEN
      K1 := N;
  REPEAT
      WRITE (LST,' ');
      FOR Z:=J TO K1-1 DO     WRITE (LST,' '+MINER[Z]+' ');
      WRITELN(LST,' '+MINER[K1]);
      WRITE (LST,' (F',FACTX:1,') :');
      FOR Z:=J TO K1-1 DO     WRITE (LST,SX[Z]:11:5);
      WRITELN (LST,SX[K1]:11:5);
      WRITE (LST,' Y (F',FACTY:1,') :');
      FOR Z:=J TO K1-1 DO         WRITE (LST,SY[Z]:11:5);
      WRITELN (LST,SY[K1]:11:5);
      WRITELN (LST,' ');
      J := K1+1;
      INC(K1,10);
      IF K1>N THEN              K1 := N;
  UNTIL J>K1;
  WRITELN (LST,' ');
  WRITELN (LST,' ');
CONT:
END;

{$F+}
PROCEDURE GRAFIK (FACTX,FACTY: CHAR;
            VAR JF: INTEGER);
(; [illegible] ;)
LABEL CONT;
VAR FX, FY, J: INTEGER;
BEGIN
  VAL (FACTX,FX,JF);
  IF JF=0 THEN      VAL (FACTY,FY,J)
  ELSE        GOTO CONT;
  IF J <> 0 THEN
  BEGIN
      JF := J;
      GOTO CONT;
  END;
  MNR := 'F'+FACTX;
```

```
  GMX:= GETMAXX;
  GMY := GETMAXY;
  LINE (GMX DIV 2,1,GMY DIV 2,GMY-10);
  LINE (1,GMY DIV 2,GMX-10,GMY DIV 2);
  A1 := GMX DIV 2 - 2;
  A2 := A1 + 4;
  DY := GMY DIV 20;
  FOR I := 1 TO 20 DO
  BEGIN
     LINE (A1,I*DY,A2,I*DY);
  END;
  OUTTEXTXY (A2+2,0,MNR);
  MNR := 'F'+FACTY;
  A1 := GMY DIV 2 - 2;
  A2 := A1 + 4;
  DX := GMX DIV 20;
  DELTA := GMX DIV 2 - DX*10;
  FOR I:= 1 TO 20 DO
  BEGIN
     LINE (I*DX+DELTA,A1,I*DX+DELTA,A2);
  END;
  OUTTEXTXY (GMX-28,A2+2,MNR);
  A1 := GMX DIV 2;
  A2 := GMY DIV 2;
  FOR I := 1 TO N DO
  BEGIN
     A3 := TRUNC (SVEC[FX,I]*10);
     XI := A1+DX*A3+DELTA;
     A3 := TRUNC (SVEC[FY,I]*10);
     YI := A2 - DY*A3;
     IF YI<=14 THEN
        INC(YI,14);
     IF XI>GETMAXX-16 THEN
        DEC (XI,8);
     OUTTEXTXY (XI-8,YI,MINER[I]);
  END;
  CH := READKEY;
CONT:
END;

{$F+}
PROCEDURE TRED1 (N:INTEGER; TOL:REAL;
                 VAR A:MAS1; VAR D,E,E2:MAS);
(; □□□□□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□ □ □ □□□□□□□□□□□□□□□□□ □□□□□□ □
□□□□□□□ □□□□□□□□□□□□□□□ □□□□□□□□□□□□ □ □□□□□□□□□□□□□□□□ □□□□□
□□□□□□□□□□□□□ □ □□□□□□□□□□□□□□□□□ □□□□□□□□□□. □□□ □□□□□□□ □ □□□□□□□□□
□ □□□□□□□ □[1:N, 1:N]. □□□□□□□□□□□□□ □□□□□□□□ □□□□□□□□□□□□□□□□ □□□-
□□□□□ □□□□□□□□□□□□□□□ □ □□□□□□□ D[1:N], □ □□□□□□□□□□□□□□□□□ □
□□□□□□□□□□ N-1 □□□□□□□ □□□□□□□ □ [1:N]. (□□□□□□□ □ [1] = 0).
□□□□□□□ □2 [I] □□□□□ □ [I]; □2 [I]. □□□□□□ □□□□□□ □□□□□□□□□□□□□
□□□□□ □□□□□□□ □ □□□□□□□□□□ □ □□□□□□□□□ □ □□□□□□□□□□□□□□□ □ □□□□□□□□□□
TRBAK1; □ □□□□ □□□□□□□□□□□ □□□□□□□□□□□ □ □□□□□□□□□□□□□□□□□
□□□□□□□□□□□. □□□□□□□ □□□□□□□□□□□□□ □□□□□□□ □ □□□□□□□□ □□□□□□□□□□□ □□□□□-
□□ □ □□□□□□□□□□□□□ □□□□ □□□□□□□□□□□ ;)
LABEL 10;
VAR I, J, K, L : INTEGER; F, G, H : REAL;
BEGIN
  FOR I :=1 TO N DO
     D[I] := A[I,I];
  FOR I := N DOWNTO 1 DO
  BEGIN
     L := I-1;
     H := 0.0;
     FOR K := 1 TO L DO
        H:= H + A[I,K]*A[I,K];
(; □□□□ □□□□□□□□□ H □□□□□□□ □□□□, □□□□□□ □□□□□□□□□□□□□□□ □□-
□□□□□□□□□□□□□□□, □□ □□□□□□□□□□□□□□□□□ □□□□□□□□□□□□ □□ □□□□□□□□□□□□□,
□□□□□□□□□□□ □□□□□□□□ □ □□□□□□ 10 □ □□□□□□ □□ □□□□□□□□□□□ ;)
        IF H <= TOL THEN
        BEGIN
           E[I] := 0.0;
           E2[I]:= 0.0;
           GOTO 10;
        END;
        E2[I] := H;
        F := A[I,I-1];
        IF F>=0 THEN
           G :=-SQRT(H)
        ELSE
           G := SQRT(H);
        E[I] := G;
        H := H - F*G;
        A[I,I-1] := F-G;
        F := 0.0;
        FOR J := 1 TO L DO
        BEGIN
           G := 0;
(; □□□□□□□□□□□□□ □□□□□□□□□□ □□□□□□□□ □ □ U ;)
           FOR K := 1 TO J DO
              G := G + A[J,K]*A[I,K];
           FOR K:= J+1 TO L DO
              G := G+A[K,J]*A[I,K];
           G := G / H;
           E[J] := G;
           F := F+G*A[I,J];
        END;
(; □□□□□□□□□□□ □□□□□□□□□□ □ ;)
        H := F/(H+H);
(; □□□□□□□□□□□□□□□ □□□□□□□□ □ ;)
        FOR J := 1 TO L DO
        BEGIN
           F := A[I,J];
           G := E[J]-H*F;
           E[J] := G;
           FOR K := 1 TO J DO
              A[J,K]:= A[J,K]-F*E[K]-G*A[I,K];
        END;
10:
        H := D[I];
        D[I] := A[I,I];
        A[I,I] := H;
  END;
END;

{$F+}
```

```
PROCEDURE PRINTR (N:INTEGER; Y:MAS);
(; [illegible] Y [illegible];)
VAR J,K1,Z:INTEGER;
BEGIN
  J := 1;
  K1 := 10;
  WRITELN (LST,' ');
  IF K1>N THEN
     K1 := N;
  REPEAT
     FOR Z:=J TO K1-1 DO
        WRITE (LST,Я[Z]:12:6);
     WRITELN (LST,Y[K1]:12:6);
     J := K1+1;
     INC(K1,10);
     IF K1>N THEN
        K1 := N;
  UNTIL J>K1;
END;

PROCEDURE CORR (NN,MM : INTEGER; X:MAS2;
          VAR R : MAS1);
(; [illegible]. [illegible], [illegible].
;)
FUNCTION SUMXR (NN,J,K: INTEGER; X:MAS2) : REAL;
VAR SS : REAL; I : INTEGER;
BEGIN
  SS:= 0.0;
  FOR I := 1 TO NN DO
     CASE K OF
        1: SS := SS + X[I,J];
        2: SS := SS + SQR (X[I,J]);
     END;
  SUMXR := SS;
END;

FUNCTION SUMXYR (NN,J,K: INTEGER; X,Y:MAS2) : REAL;
VAR SS : REAL; I : INTEGER;
BEGIN
  SS:= 0.0;
  FOR I := 1 TO NN DO
     SS := SS + X[I,J]*Y[I,K];
  SUMXYR := SS;
END;

VAR I, J, K : INTEGER; XIJ,XIK, XJK, XJ, XJ2, XI, XI2 : REAL;
BEGIN
  FOR J := 1 TO NN DO
  BEGIN
     XJ := SUMXR (MM,J,1,X);
     XJ2 := SUMXR (MM,J,2,X);
     FOR K := 1 TO NN DO
     BEGIN
        XIJ := SUMXYR (MM,J,K,X,X);
        XI := SUMXR (MM,K,1,X);
        XI2 := SUMXR (MM,K,2,X);
        R[J,K] := (MM*XIJ - XI*XJ) / (SQRT (MM*XI2 - XI*XI)*
              SQRT (MM*XJ2 - XJ*XJ));
     END;
  END;
END;

{$F+}
PROCEDURE PRINTR2 (N1:INTEGER;A:MAS1;MINER:MAS3);
(; [illegible]. [illegible] MINER. [illegible] 3 [illegible]. ;)
VAR J,K1,Z,I:INTEGER;
BEGIN
  J := 1;
  IF N1 > 13 THEN
     K1 := N1 DIV 2
  ELSE
     K1 := N1;
  WRITELN (LST);
  WRITE (LST,' ');
  REPEAT
     FOR I:=J TO K1 DO
        WRITE (LST,MINER[I]:8);
     WRITELN(LST,' ');
     FOR I := 1 TO N DO
     BEGIN
        WRITE (LST,MINER[I]:4);
        FOR Z:=J TO K1-1 DO
           WRITE (LST,A[I,Z]:8:2);
        WRITELN (LST,A[I,K1]:8:2);
     END;
     J := K1+1;
     K1:= K1*2+1;
     IF K1>N1 THEN
        K1 := N1;
     WRITELN (LST,' ****** ');
     WRITE (LST,' ');
  UNTIL J>K1;
END;

     (;**[illegible]**;)
BEGIN
  LON := 0;
  IF PR=1 THEN
     ASSIGN (LST,'LPT1')
  ELSE
     ASSIGN (LST,'C:\BURCOV');
  REWRITE (LST);
  ASSIGN (FF,START);
  RESET (FF);
{[illegible]}
  FOR I := 1 TO 3 DO
  BEGIN
     STRNG := ' ';
     READLN (FF,STRNG);
     WRITELN (LST,STRNG);
  END;
  STRNG := ' ';
  READLN (FF,STRNG);
  K := 1;
```

```
  I := 1;
  REPEAT
     MNR := '';
     MNR := COPY(STRNG,K,3);
     STR(I:3,XCOL[I]);
     MINER[I] := MNR;
     INC (I);
     INC(K,3);
  UNTIL I>N;
  WRITELN(LST,' ');
  WRITELN(LST,' **** ****');
  WRITELN(LST,' ');
  FOR I := 1 TO N DO
     WRITE (LST,MINER[I]);
  WRITELN(LST,' ');
  WRITELN(LST,' ');
50:
  TEXTCOLOR (14);
  TEXTBACKGROUND (10);
  CLRSCR;
  WRITELN (LST);
  WRITELN (LST, ' ******************');
  WRITELN (LST,' N= ',N:4,'; M= ',M:4);
  WRITELN (LST, ' **********');
  IF LON=0 THEN
  BEGIN
     WRITE (' ВВЕДИТЕ ДЛИНУ СТРОКИ ');
     READLN (A1);
     FOR J :=1 TO N DO
        SI[J] :=0;
     SX := SI;
     SC := SI;
     FOR I := 1 TO M DO
     BEGIN
        FOR J :=1 TO A1 DO
        IF J<=N THEN
           READ (FF,SX[J])
        ELSE
           READ (FF,SUM);
        FOR J :=1 TO N DO
        BEGIN
           IF ABS(SX[J])<EPS THEN
              SX[J]:= 0.001;
           IF ABS(1.0-SX[J])<EPS THEN
              SX[J] := 1.001;
           SI[J]:= SX[J] + SI[J];
           RRC[I,J] := SX[J];
        END;
     END;
(; □□□□□□ □□□□□□□□□□□□□□□ □□□□□□□□ ;)
     FOR I := 1 TO N DO
        SS[I] := SI[I] / (M-1);
     WRITE(LST,'****** КОРРЕЛЯЦИОННАЯ ФУНКЦИЯ ****');
     PRINTR (N,SS);
     CORR(N,M,RRC,RRCC);
     PRINTR2 (N,RRCC,MINER);
     CLOSE (FF);
     IF LON=0 THEN
           INC (LON);
     END
     ELSE
        INC (LON);
     IF LON >1 THEN
     BEGIN
        FOR I:= 1 TO N DO
           FOR J := 1 TO N DO
              RRC[I,J] := RRCC[I,J];
        CORR (N,N,RRC,RRCC);
     END;
     A := RRCC;
     AA:= RRCC;
{**** □□□□□□ □□□□□□□□□□□ □□□□□□□□ □ □□□□□□□□□□□ □□□□□□□□ □□□□□□□
□ □□ □□□□□□ □□□□□□□□□□□ □ □□□□□□□□□□□□□□□ □□□□□□□□□□ □□
□□□□□□□□□□□ ****}
     TRED1 (N,EPS,A,SI,SC,SX);
     Y := SC;
     TQL1 (N,EPS,SI,SC,AA);
     SC:=Y;
     TRBAK (N,1,N,A,SC,AA);
     WRITELN (LST,'*** СОБСТВЕННЫЕ ЗНАЧЕНИЯ***');
     PRINTR (N,SI);
     X := SI;
     SUM := 0.0;
     WRITELN (LST,' ');
     WRITELN (LST,' *** СОБСТВЕННЫЕ ВЕКТОРЫ ***');
     FOR I :=1 TO N DO
        SUM := SUM+X[I];
     FOR I:=1 TO N DO
        SC[I]:= X[I]/SUM*100;
     PRINTR (N,SC);
     XMIN:=0.0;
     WRITELN (LST,' ');
     WRITELN(LST,'***СУММА ВКЛАДОВ В ОБРАТНОМ ПОРЯДКЕ
**');
     FOR I:=N DOWNTO 1 DO
     BEGIN
        XMIN:= XMIN + X[I]/SUM*100;
        SC[N+1-I] := XMIN;
        IF XMIN<90 THEN
           IMAX:=N+2-I;
     END;
     PRINTR (IMAX,SC);
     WRITELN (LST,' ');
{ □□□□□□ □□□□□□□□ □□□□□□□□ □ □□□□□□□□□□□ □□□□□□□□ }
     FOR I := 1 TO N DO
     BEGIN
        J := N+1-I;
        SC[J] := X[I];
        FOR K := 1 TO N DO
           A[K,J] := AA[K,I];
     END;
     X:= SC;
     AA:= A;
     FOR I := 1 TO N DO
        FOR J :=1 TO N DO
        BEGIN
```

```
          AA[J,I] := AA[J,I] * SQRT (X[I]);
          SVEC[I,J]:=AA[J,I]; END;
          WRITELN (LST,'*** СОБСТВЕННЫЕ ВЕКТОРА ***');
          PRINTR2 (IMAX,AA,XCOL);
          WRITELN (LST,' ');
{**** □□□□□□□□□□□ □□□□□□□□, □□□□□□□□□ □□□□□□□ ******}
       GAUSS (N,AA,A);
{********* □□□□□□□ ***********
□ □□□□□□□□□□ INITGRAPH (GRAPHDRIVER,GRAPHMODE,'...') □□□□□□□□
□□□□□□□□ □□□□□□□ □□□□□ □□□ □□□□□□□□□□□□□□ □□□□□□□□□ }
       GRAPHDRIVER:=DETECT;
       INITGRAPH(GRAPHDRIVER,GRAPHMODE,'...');
       IF GRAPHRESULT <> GROK THEN
          HALT (1);
       SETBKCOLOR (WHITE);
       SETCOLOR (BLUE);
       SETLINESTYLE (SOLIDLN,0,NORMWIDTH);
{*********** □□□□□ □□□□□□□ **********}
       SETFILLSTYLE (0,1);
       DN := FALSE;
       CHX:=' ';
       CHY:=' ';
       REPEAT
          BAR (0,0,GETMAXX-1,GETMAXY-1);
          OUTTEXTXY (GETMAXX DIV 2,GETMAXY DIV 2,
             'ДЛЯ ВЫХОДА НАЖМИТЕ ПРОБЕЛ ...');
          OUTTEXTXY (GETMAXX DIV 2,GETMAXY -20,
             ' ВВЕДИТЕ НОМЕР ФАКТОРА ПО X:');
          CHX:=READKEY;
          IF CHX<>' ' THEN
          BEGIN
             OUTTEXTXY(GETMAXX-30,GETMAXY -
                20,CHX);
             OUTTEXTXY (GETMAXX DIV 2+22*8,
                GETMAXY -10,'PO Y:');
             CHY:=READKEY;
             IF CHY<>' ' THEN
             OUTTEXTXY (GETMAXX-30,GETMAXY -
                10,CHY);
             BAR (0,0,GETMAXX-1,GETMAXY-1);
             IF CHY<> ' ' THEN
                GRAFIK (CHY,CHX,J);
             IF J<>0 THEN
             BEGIN
                OUTTEXTXY (GETMAXX DIV 3,
                   GETMAXY DIV 2,
                   'ВЫ ОШИБЛИСЬ ПРИ ВВОДЕ!');
                OUTTEXTXY (GETMAXX DIV 3,GETMAXY
                   DIV 2 + 10,
                   'ПОВТОРИТЕ ЗАНОВО ВВОД ФАКТОРОВ ! ');
                CHX := READKEY;
             END ;  {FACTOR}
(; □□□□ □□□□ □□□□□□□□□□□ □□ □□□□□□□□ □□□□□□□, □□□□□□□ □□ □□□□□□□
□A □□□□□□, □□ □□□□ □□□□□□ □□□□□ □□□□□□□□□□□ □ □□□□□□□□□□
□□□□□□□□□□□ □□□□□□□. □□□□□ □□ □□□□□□□□ □□□□□ □□□□□ □□ □□□□□□□, □□□ □
□□ □□□□□□□. □□□□□□□□ □□□□□□ □□□□ □□□□□□□! ;)
{;              ELSE
                   PRNGRAF (CHX,CHY,J);}
             END
             ELSE
                IF CHX=' ' THEN
                   DN := TRUE;
       UNTIL DN;
       CLOSEGRAPH;
       TEXTCOLOR (14);
       TEXTBACKGROUND (10);
       CLRSCR;
       WRITE (' ЕЩЕ РАЗ? 1-ДА, 0-НЕТ ');
       READLN (I);
       IF I=1 THEN GOTO 50;
       IF PR=1 THEN
          WRITELN (LST,' ');
       CLOSE (LST);
       WINDOW (0,0,80,24);
       CLRSCR;
END.
```

Как и ранее, для проверки и тестирования процедур и программы методом главных компонент выполнен анализ экспериментальных данных. Данные взяты из табл. 7.9 по расчету корреляционной матрицы.

РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ ***BURCOV***

n= 5; m= 20.

СРЕДНИЕ ПО СТОЛБЦАМ (для контроля за вводом данных):

2.421053 2.628421 5.084211 3.224211 0.008421.

КОРРЕЛЯЦИОННАЯ МАТРИЦА

1.00	0.34	0.58	-0.29	-0.45
0.34	1.00	-0.05	0.53	-0.45
0.58	-0.05	1.00	-0.73	-0.74
-0.29	0.53	-0.73	1.00	0.36
-0.45	-0.45	-0.74	0.36	1.00

СОБСТВЕННЫЕ ЗНАЧЕНИЯ

0.054739 0.128314 0.564726 1.633568 2.618654

*** ВКЛАД В ПРОЦЕНТАХ ***

1.094772 2.566273 11.294512 32.671368 52.373075

СУММА ВКЛАДОВ В ОБРАТНОМ ПОРЯДКЕ (учитываются только те значения, которые дают существенный вклад в общую сумму (до 96 %), т.е., начиная с максимального, суммируют вклад каждого из λ_i пока не получится нужная сумма)

52.373075 85.044443 96.338955

Êàê âèäíî èç àíàëèçà ñóìì âêëàäîâ, äàííûå îïèñûâàþòñÿ òðåìÿ ôàêòîðàìè. Îâîçíà÷èì èõ óñëîâíî *Õ, Y* è *Z* èëè *F*1, *F*2, *F*3.

ФАКТОР ПО X
(F1) : 0.72662 0.12109 0.94921 -0.70266 0.82541
ФАКТОР ПО Y
(F2) : -0.30741 -0.97541 0.16630 -0.66483 0.34349
ФАКТОР ПО Z
(F3) : -0.61074 0.06987 0.07228 -0.02193 0.42560

Теперь, зная факторы, можно вернуться обратно к исходной матрице данных, используя основное уравнение метода главных компонент. Полученная ошибка в данных составит 3.67 % (см. сумму вкладов). После выполненных преобразований рассчитывается матрица первых корреляционных моментов восстановленной матрицы ***X*** и все вычисления повторяются вновь. Каждый шаг (возврат к исходной матрице и пересчет факторов) называется итерацией. Как правило, в практических расчетах достаточно пяти-шести итераций.

МАТРИЦА ПЕРВЫХ КОРРЕЛЯЦИОННЫХ МОМЕНТОВ

1.00 0.28 0.89 -0.72 -0.88

0.28 1.00 0.01 0.39 -0.47

0.89 0.01 1.00 -0.91 -0.88

-0.72 0.39 -0.91 1.00 0.61

-0.88 -0.47 -0.88 0.61 1.00

СОБСТВЕННЫЕ ЗНАЧЕНИЯ

0.000000 0.005314 0.116924 1.406921 3.470841

ВКЛАД В ПРОЦЕНТАХ

0.000000 0.106284 2.338487 28.138412 69.416818

** СУММА ВКЛАДОВ В ОБРАТНОМ ПОРЯДКЕ **

69.416818 97.555230

По сравнению с предыдущей итерацией учет только двух факторов уже дает 97 %, но, чтобы не увеличивать ошибку, рассмотрим опять три основных фактора (они в сумме дают более 99 %).

ФАКТОР ПО X
(F1) : -0.13737 -0.98749 0.14254 -0.53192 0.33115
ФАКТОР ПО Y
(F2) : -0.28075 0.00584 0.08834 -0.02209 -0.17256
ФАКТОР ПО Z
(F3) : 0.94981 0.15406 0.98548 -0.84466 -0.92754

****** **2-я ИТЕРАЦИЯ** ******

МАТРИЦА ВТОРЫХ КОРРЕЛЯЦИОННЫХ МОМЕНТОВ

1.00 0.30 0.99 -0.92 -0.99

0.30 1.00 0.15 0.11 -0.43

0.99 0.15 1.00 -0.97 -0.96

-0.92 0.11 -0.97 1.00 0.85

-0.99 -0.43 -0.96 0.85 1.00

*** СОБСТВЕННЫЕ ЗНАЧЕНИЯ ***

0.000036 0.000153 0.060399 22.182735 77.756678

*** ВКЛАД В ПРОЦЕНТАХ ***

77.756678 99.939413

ФАКТОР ПО X 0.99849
(F1) : 0.26510 0.99323 -0.92980 -0.98466
ФАКТОР ПО Y -0.03160 -0.96422
(F2) : 0.11533 -0.36804 0.17223

****** **3-я ИТЕРАЦИЯ** ******

МАТРИЦА ТРЕТЬИХ КОРРЕЛЯЦИОННЫХ МОМЕНТОВ

1.00 0.49 1.00 -0.98 -1.00

0.49 1.00 0.42 -0.30 -0.55

1.00 0.42 1.00 -0.99 -0.99

-0.98 -0.30 -0.99 1.00 0.96

-1.00 -0.55 -0.99 0.96 1.00

СОБСТВЕННЫЕ ЗНАЧЕНИЯ

0.000000 0.000000 0.000032 15.937216 84.062752

*** ВКЛАД В ПРОЦЕНТАХ ****

84.062752 99.999968

ФАКТОР ПО X
(F1): 0.99771 0.54424 0.99058 -0.96450 -1.00000
ФАКТОР ПО Y
(F2): -0.06758 0.83893 -0.13694 0.26408 -0.00120

ВЫВОД. Из анализа матриц факторных нагрузок видно, что зависимыми являются переменные I и III; IV и V, а переменная II, хотя и тяготеет к первой группе, но ее нельзя отнести ни к той ни к другой группировке: по первому фактору II является антагонистом по отношению ко второй группировке, в по второму фактору - к первой. Третий фактор несущественен, так как первых два дают почти стопроцентный вклад, поэтому его можно не учитывать.

Данный вывод подтверждает все сделанные ранее выводы о группах для исследуемых экспериментальных данных.

§ 2.

В большинстве физических экспериментов, результатом которых является некоторый измеренный физический процесс $x(t)$, помимо статистических характеристик этого процесса необходимо исследовать спектр сигнала, поскольку реально измеренный сигнал $x(t)$ обычно содержит большое (иногда бесконечное) число гармонических составляющих. Исследование спектра - это определение вклада гармоник в измеренный сигнал, мощности и амплитуды спектральных составляющих, фазового сдвига между составляющими и т.д. Измеренный сигнал может быть аналоговым или дискретным, в связи с чем в этих двух случаях используются соответствующие ортогональные преобразования различной формы, к которым относятся, например, преобразования Фурье, Уолша - Адамара, Хаара и др. В настоящем параграфе, после введения основных определений, рассматриваются алгоритмы Фурье и Уолша-Адамара, наиболее часто используемые при спектральной обработке дискретных и аналоговых сигналов.

2.1. РЯД ФУРЬЕ, ПРЕОБРАЗОВАНИЕ ФУРЬЕ И АЛГОРИТМ БПФ

Пусть $x(t)$ - измеренный действительный сигнал, заданный на интервале $[t_0, t_0 + T]$[1] и представленный в виде ряда

$$x(t) = \sum_{n=0}^{\infty} a_n \cdot u_n(t) \quad , \tag{9.7}$$

где $u_n(t)$ - множество непрерывных функций действительного аргумента t. Тогда, если

$$\{ u_n(t) \} = \{ 1, \cos(nw_0 t), \sin(nw_0 t) \}$$

есть множество синусоидальных функций, то измеренный сигнал может быть представлен в виде ***ряда Фурье***

$$x(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos(n \cdot \omega_0 \cdot t) + \sum_{n=1}^{\infty} b_n \sin(n \cdot \omega_0 \cdot t), \tag{9.8}$$

где $w_0 = 2\pi/T$ - основная угловая частота, а коэффициенты вычисляются по формулам

$$a_0 = \frac{1}{T}\int_T x(t)dt; \quad a_n = \frac{2}{T}\int_T x(t)\cos(n \cdot w_0 \cdot t)dt;$$

$$b_n = \frac{2}{T}\int_T x(t)\sin(n \cdot w_0 \cdot t)dt$$

Таким образом, сигнал $x(t)$ можно представить множеством действительных чисел $\{ a_0, a_n, b_n \}$.

Разложение (9.8) часто удобнее записывать в комплексной форме:

$$x(t) = \sum_{n=-\infty}^{\infty} c_n e^{inw_0 t}, \quad c_n = \frac{1}{T}\int_T x(t) \cdot e^{-inw_0 t} dt \cdot \tag{9.9}$$

В этом случае справедливы следующие определения.

Фурье-спектром мощности называется последовательность

$$P_n = |c_n|^2, \; n = 0, \pm 1, \pm 2, \ldots, \tag{9.10}$$

где P_n - мощность n-й спектральной составляющей.

Амплитудным фурье-спектром называется последовательность

$$p_n = \sqrt{P_n}, \; n = 0, \pm 1, \pm 2, \ldots, \tag{9.11}$$

а ***фазовый фурье-спектр*** определяется из следующего выражения:

$$\psi_n = \begin{cases} \operatorname{arctg}\dfrac{\operatorname{Im}(c_n)}{\operatorname{Re}(c_n)}, & n = \pm 1, \pm 2, K \\ 0, & n = 0. \end{cases} \tag{9.12}$$

Основные свойства спектра мощности и амплитудного спектра можно сформулировать следующим образом:

1) инвариантность величине временного сдвига τ,

2) неотрицательность ($P_n, p_n > 0$):

3) четность функций P_n и p_n относительно аргумента n.

Свойства фазового спектра:

1) ψ_n является функцией τ (изменяется при сдвиге $x(t)$ вдоль оси t),

2) ψ_n инвариантен к усилению и ослаблению сигнала (тогда как P_n и p_n являются функциями амплитуды),

3) ψ_n является нечетной функцией n.

Переход от ряда Фурье к преобразованию Фурье осуществляется следующим образом. Предположим, что время измерений $T \to \infty$. Процесс в этом случае становится апериодическим и $w_0 \to dw$, $nw_0 \to w$. В результате имеем:

$$\lim_{\tau \to \infty} c_n = \int_{-\infty}^{\infty} x(t) \cdot e^{-iwt} dt \equiv F_x(w), \tag{9.13a}$$

где $F_x(w)$ называется преобразованием Фурье функции $x(t)$. Выражение для ряда Фурье (9.9) при этом приобретает вид обратного преобразования Фурье функции $F_x(w)$:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F_x(w) \cdot e^{iwt} dw, \tag{9.13b}$$

Сравнение формул (9.9) и (9.13) позволяет заключить, что, если сигнал задан на конечном интервале времени T, то его преобразование Фурье $F_x(w)$ точно определяется рядом Фурье на множестве точек, равномерно расположенных по оси w на расстоянии $2\pi/T$ одна от другой.

Все сказанное относилось к Фурье-представлению непрерывных (аналоговых) сигналов. В физических же экспериментах, а также, например, в экономических и социологических исследованиях, особенно учитывая

[1] При этом предполагается, что влево и вправо от этого интервала функция $x(t)$ периодически продолжена.

цифровую регистрацию и обработку данных на ЭВМ, чаще всего приходится иметь дело с *временными последовательностями*. При этом для исследования спектра в качестве ортогонального преобразования используется дискретное преобразование Фурье (ДПФ) этих последовательностей и соответствующие формулы (9.9) принимают вид:

$$C_x(k) = \frac{1}{N}\sum_{m=0}^{N-1} X(m)\cdot W^{km}, \quad k = 0,\ 1,\ \ldots,\ N-1, \quad (9.14а)$$

$$X(m) = \sum_{m=0}^{N-1} C_x(k)\cdot W^{-km}, \quad m = 0,\ 1,\ \ldots,\ N-1, \quad (9.14б)$$

$$W = e^{-i2\pi/N}, \quad i = \sqrt{-1}\ .$$

Операция (9.14а) называется ***дискретным преобразованием Фурье*** (ДПФ) последовательности $X(m)$, обратная операция (9.14б) - ***обратным ДПФ*** (ОДПФ), соответственно спектр мощности, амплитудный и фазовый Фурье-спектры определяются, в отличие от выражений (9.10) - (9.12), формулами

$$P(k) = \left|C_x(k)\right|^2, \quad p(k) = \sqrt{P(k)}, \quad (9.15)$$

$$\psi_x(k) = \operatorname{arctg}\frac{\left[\operatorname{Im} C_x(k)\right]}{\left[\operatorname{Re} C_x(k)\right]}, \quad k = 0,\ 1,\ \ldots,\ N-1. \quad (9.16)$$

Отметим, что непосредственное применение формул (9.14) сопряжено с большим объемом вычислительной работы и требует выполнения примерно $2N^2$ арифметических операций (комплексных умножений с последующим сложением или вычитанием), что приводит при увеличении N к резкому росту временных затрат. Это обстоятельство обусловило, начиная с работы Дж.Кули и Дж.Тьюки [1965], интерес к поиску "быстрых" алгоритмов вычисления ДПФ - реализации так называемого быстрого преобразования Фурье (БПФ). В настоящее время имеется большое количество алгоритмов БПФ, минимизирующих как время выполнения процедуры, так и объем требуемой памяти (описание некоторых из них приведено в книге Н.Ахмеда и К.Р.Рао [1980], эффективная реализация БПФ в ряде конкретных систем обработки данных - в работах О.В. Гулинского, В.Ю.Белашова, Л.И.Дормана и др. [1988], А.А.Белашовой и В.Ю.Белашова [1990]). Рассмотрим реализацию алгоритма Кули - Тьюки вычисления БПФ, предложенную в последней из цитированных выше работ, сделав несколько предварительных замечаний.

Будем предполагать, что в формулах (9.14) $N = 2^n$, $n = 1, 2, \ldots, n_{max}$. При этом общность не теряется, поскольку N выбирается достаточно большим для того, чтобы удовлетворять теореме Котельникова[1] $N \geq 2BL$, где B (Гц) - полоса частот сигнала $x(t)$, L (с) - его длительность.

При расчете коэффициентов Фурье по формуле (9.14а) индексные выражения k, если их представить в двоичном виде, появляются в $C_x(k)$ в инвертированном порядке, в отличие от индексных выражений m в $X(m)$, где они представлены в естественном порядке [Ахмед, Рао, 1980]. Вызывающая в общем случае большие затраты времени двоичная инверсия может быть при $N = 2^n$ эффективно осуществлена с помощью метода перестановки данных, использующего только десятичную арифметику в соответствии с алгоритмом, предложенным Н.Ахмедом и К.Р.Рао [1980], суть которого заключается в следующем.

1. Число N выражается в терминах n множителей:

$$n_s = N/2^s,\ s = 1,2,\ldots,\ n = \log_2 N. \quad (9.17)$$

2. Формируется таблица T следующего вида:

$$0$$
$$n_1$$
$$n_2\ (n_1 + n_2)$$
$$n_3\ (n_1 + n_3)\ (n_2 + n_3)\ (n_1 + n_2 + n_3)$$
$$n_4\ (n_1 + n_4)\ (n_2 + n_4)\ (n_1 + n_2 + n_4)\ \ldots$$
$$\ldots$$
$$n_n,$$

элементы которой представляют собой двоичную инверсию

$$\{k\} = \{0,\ n_1,\ n_2,\ (n_1 + n_2),\ n_3,\ (n_1 + n_3),\ \ldots,\ (n_1 + n_2 + \ldots + n_n)\}$$

исходной "естественной" последовательности $\{m\}$. Например, для $N=8$ имеем: $n_1 = 4$, $n_2 = 2$, $n_3 = 1$ и вместо исходной последовательности $\{m\}=\{0,1,2,3,4,5,6,7\}$ получим инвертированную: $\{k\}=\{0,4,2,6,1,5,3,7\}$.

Принимая во внимание изложенное, процедуру, реализующую алгоритм Кули - Тьюки вычисления БПФ, можно сформулировать следующим образом [Белашова, Белашов, 1990].

1. Получение методом перестановки инвертированной последовательности $\{k\}$ из ряда $\{m\}$, представленного в естественном порядке, в соотвествии с формулой (9.17).

2. Получение последовательности степеней W (см. формулы (9.14)) с показателями, представляющими собой последовательность $\{k\}$:

$$\left\{W^{(k)}\right\} = W^0,\ W^{k_1},\ W^{k_2},\ \ldots,\ W^{k_{N/(2-i)}}$$

3. Выполнение $n = \log_2 N$ итераций согласно графу, пример которого для $N = 8$ показан на рис. 9.1. При этом каждой группе i-й итерации соответствует свой множитель W^{k_i} из последовательности $\{W^{(k)}\}$. Полученные после i-й итерации промежуточные значения записываются на место исходного массива $\{X(m)\}$, что, как видно из рис. 9.1, вполне осуществимо при итерационном методе реализации алгоритма БПФ и дает значительную экономию оперативной памяти.

4. Вычисление коэффициентов $C_x(k)$ по формуле (9.14а).

Замечание. Рассмотренный алгоритм БПФ не зависит от того, являются ли исходные величины $X(m)$ действительными или комплексными. Поэтому он может быть эффективно использован и для вычисления ОДПФ с незначительными изменениями, которые следуют из сравнения формул для ДПФ (9.14а) и ОДПФ (9.14б):

1) множители W заменяются на комплексно-сопряженные W^*;

2) опускаются множители $1/N$, стоящие после последней итерации.

[1] В зарубежной литературе обычно употребляются другие названия - "теорема отсчетов", "теорема дискретизации".

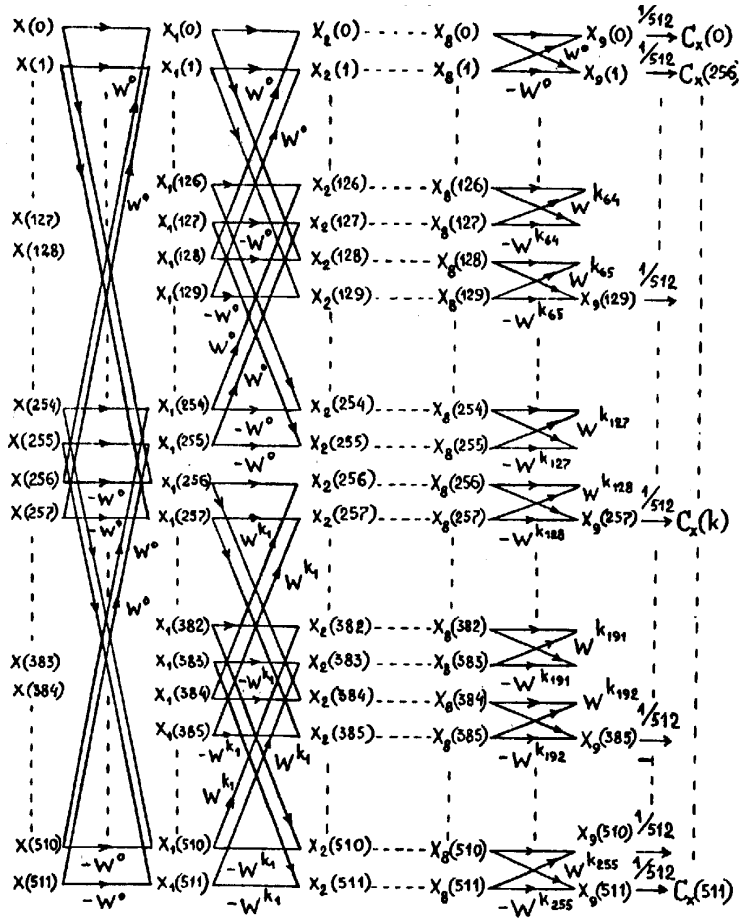

Рис. 9.1. Граф для быстрого преобразования Фурье при $N = 8$.

Такие изменения обычно предусматриваются в единой процедуре, по которой вычисляются в зависимости от значения некоторого задаваемого пользователем ключа БПФ либо ОБПФ.

Как видно из вышеизложенного, рассмотренный алгоритм включает $\sim N \log_2 N$ (т.е. порядка N) арифметических операций, что, по сравнению с 2 опускаются множители $1/N$, стоящие после последней итерации.

Отметим, что реализация алгоритма БПФ, предложенная в работе А.А.Белашовой и В.Ю.Белашова [1990] и выполненная на языке МНЕМОКОД в операционной системе АСПО СМ ЭВМ, позволяет сэкономить (не в ущерб времени выполнения вычислений) еще N ячеек памяти[1] по сравнению с предложенной Дж.Кули и Дж.Тьюки [1965] и модификациями алгоритма, рассмотренными в работах Н.М.Бреннера [1967], Дж. Р.Фишера [1970] и Р. Синглтона [1969]. Отметим также, что рассмотренный алгоритм БПФ значительно уменьшает ошибку округления, связанную с арифметическими операциями. По сравнению с прямым методом он снижает ее в $N/\log_2 N$ раз [Ахмед, Рао, 1980].

Представленная процедура реализует БПФ для последовательностей комплексных либо действительных (в этом случае полагается $\mathrm{Im}X(m) = = 0$) чисел с произвольной в пределах ресурсов памяти ЭВМ, длиной массива входных данных. При этом для $N \neq =2$ обрабатывается количество членов ряда, равное максимальному $N' = 2^n < N$.

Формальные параметры процедуры. *Входные*: *x* (тип ***real***) - двумерный массив $x[1{:}n,1{:}2]$, в котором $x[m,1] = \mathrm{Re}\ X(m)$ и $x[m,2] = \mathrm{Im}X(m)$ - при вычислении прямого БПФ, или $x[m,1] = \mathrm{Re}C_x(k)$ и $x[m,2] = \mathrm{Im}C_x(k)$ - при вычислении обратного БПФ, соответственно; *t, ns* (тип ***word***) - одномерные массивы размерностью $[1{:}n]$, использующиеся для перестановки (инверсии) данных; *n* (тип ***word***) - длина обрабатываемой реализации; *sign* (тип ***integer***) - ключ, значение которого определяет вид преобразования: при *sign* = -1 выполняется прямое, а при *sign* = 1 - обратное БПФ. *Выходные*: *x* (тип ***real***) - двумерный массив, сформированный на месте одноименного входного (см. выше), содержащий действительные и мнимые части коэффициентов Фурье, расположенных в естественном порядке, - при выполнении прямого БПФ, или комплексный ряд $X(m)$ - при выполнении обратного БПФ.

Примечание. Тип и длина массивов задается в вызывающей (головной) программе, например, следующим образом:

```
  . . . . . . . . . . . . .
    TYPE
      RT = ARRAY[0..5000] OF WORD;
      RP = ^RT;
      DRT = ARRAY[0..5000,0..1] OF REAL;
      DRP = ^DRT;
VAR NS,T :RP;  X :DRP;
. . . . . . . . . . . . .
PROCEDURE FFT(X:DRP; T,NS:RP; N:WORD;
              SIGN:INTEGER);
VAR  Y,K,INDX2,I,NR,INDX,IDX,U,P :WORD;
              A10,A11,A2,D10,D11,D20,D21,TT :REAL;
BEGIN
  IF N=0 THEN
      HALT(0); { □□□□ N=0 - □□□□□ □□ □□□□□□□□□ }
{ □□□□□□□□ □□ □□, □□□ N=2^N. □□□□ N □□ □□□□ 2^N ,
□□ □ □□□□□□□□ N □□□□□□□□□□ □□□□□□□□□□ N'=2^N<N }
  IDX:=0;
  FOR INDX:=0 TO 12 DO
  IF ROUND(POW(2,INDX))<=N THEN
      IDX:=INDX;
  N:=ROUND(POW(2,IDX));
  NR:=ROUND(LN(N)/LN(2));
{ □ □□□□□□□□□□□ □□ □□□□□□□□ □□□□□ SIGN □□□□□□□□□□□□
□□□ (SIGN=-1) □□□ □□□□ (SIGN=1)}
  FOR INDX:=1 TO NR DO
  BEGIN { □□□□□□□□ □□□□ □□□□□□ A □□□ INDX-□ □□□□□□□□ }
      A2:=(SIGN*2*PI)/( 1 SHL INDX);
      FOR I:=1 TO INDX-1 DO
          NS^[I]:=(1 SHL (INDX-I-1));
```

[1] Под ячейкой в данном случае понимается количество двоичных разрядов, занимаемых в данной ЭВМ числом с плавающей точкой.

```
      K:=0;
      FOR I:=1 TO INDX-1 DO
      BEGIN
         FOR Y:=0 TO K DO
            T^[K+Y+1]:=T^[Y]+NS^[I];
         INC(K,Y+1);
      END;
      U:=0;
      INDX2:=0;
      P:=(N SHR (INDX-1));
      Y:=P SHR 1;
      REPEAT
{ □□□□□□□□□□ INDX-□ □□□□□□□□ □□□□□□□□ □□□□□ □□□ }
         A10:=COS(A2*T^[U]);
         A11:=SIN(A2*T^[U]); INC(U);
         FOR I:=INDX2 TO (INDX2-1+(P SHR 1)) DO
         BEGIN
            IDX:=I+(P SHR 1);
            D10:=X^[I,0]+X^[IDX,0]* A10 -X^[IDX,1]* A11;
            D11:=X^[I,1]+X^[IDX,0]* A11 +X^[IDX,1]* A10;
            D20:=X^[I,0]+X^[I+Y,0]*(-A10)-
                 X^[I+Y,1]*(-A11);
            D21:=X^[I,1]+X^[I+Y,0]*(-A11)+
                 X^[I+Y,1]*(-A10);
            X^[I,0] :=D10;
            X^[I,1] :=D11;
            X^[I+Y,0]:=D20;
            X^[I+Y,1]:=D21;
         END;
         INC(INDX2,P);
      UNTIL (INDX2=N);
{ □□□□□ □□□□□ □□□□□□□ INDX-□ □□□□□□□□ }
      FOR I:=0 TO N-1 DO
         T^[I]:=0;
   END; { □□□□□ □□□□□□□□□ □□□□□ }
{ □□□□□□□□ □□□□□□□□ □□□□□□□□□□□□□ □□□□□ Cx }
   FOR I:=1 TO NR+1 DO
      NS^[I]:=(1 SHL (NR-I));
   K:=0;
   FOR I:=1 TO NR DO
   BEGIN
      FOR Y:=0 TO K DO
         T^[K+Y+1]:=T^[Y]+NS^[I];
      INC(K,Y+1);
   END;
{ □□□□□□□□□□ □□□□□□□□ □□□□□□□□□□□□□ □□□□□ Cx }
   FOR I:=0 TO N-1 DO
   BEGIN
      NS^[I]:=0;
      IF SIGN=(-1) THEN
      BEGIN
         X^[I,0]:=X^[I,0]/N;
         X^[I,1]:=X^[I,1]/N;
      END;
   END;
   FOR I:=0 TO N-1 DO
   BEGIN
      IF NS^[I]=0 THEN
      BEGIN
         TT:=X^[I,0];
         X^[I,0]:=X^[T^[I],0];
         X^[T^[I],0]:=TT;
         TT:=X^[I,1];
         X^[I,1]:=X^[T^[I],1];
         X^[T^[I],1]:=TT;
         NS^[T^[I]]:=1;
      END
      ELSE
   END;
  END {* □□□□□ □□□□□□□□□ □□□ *}
```

Процедура FFT тестировалась на *IBM PC/AT*-286 для последовательностей $X(m)$, задававшихся как в виде различного рода функций, так и виде произвольных рядов действительных и комплексных чисел при различных n. Пример результатов вычислений прямого и обратного БПФ для $n = 8$ приведен в табл. 9.1.

Таблица 9.1

m	$X(m)$ (исходный ряд)	k	$C_x(k)$	$X(m)$ (ряд после ОБПФ)
0	$1 + 0i$	0	$2 + 0i$	$1 + 0i$
1	$2 + 0i$	4	$0.051777 - 0.728553i$	$2.000001 - 0i$
2	$1 + 0i$	2	$-0.25 + 0.25i$	$1 + 0i$
3	$0 + 0i$	6	$-0.301777 + 0.021447i$	$0 + 0i$
4	$2 + 0i$	1	$0 + 0i$	$2 + 0i$
5	$3 + 0i$	5	$-0.301777 - 0.021447i$	$2.999998 - 0i$
6	$4 + 0i$	3	$-0.25 - 0.25i$	$4 - 0i$
7	$3 + 0i$	7	$0.051777 + 0.728553i$	$3 + 0i$

2.2. ВЫЧИСЛЕНИЕ АМПЛИТУДНОГО СПЕКТРА, СПЕКТРА МОЩНОСТИ И ФАЗОВОГО СПЕКТРА МЕТОДОМ БПФ

Вычисленные коэффициенты Фурье $C_x(k)$ позволяют найти амплитудный спектр, спектр мощности и фазовый спектр продискретизированного сигнала $X(m) = x(mT)$, для которого $m = 0, 1, ..., 2^n - 1$ (T - интервал дискретизации) в соответствии с формулами (9.15), (9.16). Проиллюстрируем это на следующем примере. Пусть измеренный сигнал представляет собой функцию $x(t) = e^{-t}$, $0 < t < 4$ *мс* и требуется с помощью алгоритма ДПФ найти его амплитудный и фазовый спектры. Для применения алгоритма БПФ сигнал $x(t)$ необходимо продискретизировать и из полученной последовательности $X(m)$ взять $N = 2^n$ дискретных значений. Далее, для последовательности $X(m)$, $m = 0, 1, ..., N$ можно использовать приведенную процедуру FFT, обращение к которой может быть построено, например, следующим образом:

```
. . . . . .
 TYPE
 RT = ARRAY[0..5000] OF WORD;  RP = ^RT;
DRT=ARRAY[0..5000,0..1] OF REAL;
DRP = ^DRT;
 VAR  NS,T :RP;   X :DRP;   N,I :WORD;
. . . . . .
 BEGIN
   FFT(X,N,-1); {□□□}
   FOR I:=0 TO N-1 DO
   BEGIN
{ □□□□□□□□□□ □□□□□□□□, □□□□□□□□□□□□□: }
   X^[I,0]:=SQRT(X^[I,0]*X^[I,0]+`
          X^[I,1]*X^[I,1]);
{ □ □□□□□□□□: }
    X^[I,1]:=ARCTAN((X^[I,1])/(X^[I,0]));
   END;
 END.
```

Пример результатов для экспоненциально убывающего сигнала в случае N =32 показан на рис. 9.2. (амплитудный спектр нормирован), который хорошо иллюстрирует свойства амплитудного и фазового спектров, рассмотренных в п. 2.1. При основной частоте f_0 = 1/ 4 *мс* = 250 *Гц* (T = 0.25 *мс*) полоса частот сигнала x(t) предполагается, в соответствии с теоремой Котельникова (см. п. 2.1) равной $B = (N/2)\times \times f_0$ = 4000 *Гц*. Пользуясь результатами, полученными с помощью процедуры FFT, можно теперь построить графики, представляющие собой амплитудный $|F_x(kw_0)|$ и фазовый $\psi_x(kw_0)$ ($w_0 = 2pf_0$, k = 0, +1, +2, ..., N/2) спектры последовательности $X(m)$, m = 0,1, ..., 31 (рис. 9.3).

2.3. ВЫЧИСЛЕНИЕ КОРРЕЛЯЦИОННЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ И СВЕРТКИ МЕТОДОМ БПФ

Алгоритм БПФ может быть эффективно использован для вычисления **корреляционных** последовательностей (взаимно- или кросс-корреляционных функций) и **свертки** сигналов [Ахмед, Рао, 1980].

Пусть, например, имеются две действительные последовательности $\{X(m)\}$ и $\{Y(m)\}$ с периодом N. Тогда последовательность, полученная в результате их корреляции, будет определяться по следующей формуле [Г.Корн, Т.Корн, 1978]:

$$\hat{Z}(m) = \frac{1}{N}\sum_{l=0}^{N-1} X(l)\cdot Y(m+l), \quad m = 0,\ 1,\ \ldots,\ N-1 \ . \qquad (9.18)$$

Теорема корреляции [Ахмед, Рао, 1980] гласит, что, если функция корреляции последовательностей $\{X(m)\}$ и $\{Y(m)\}$ действительных чисел определяется формулой (9.12), причем имеют место взаимно однозначные отображения $X(m) \leftrightarrow C_x(k)$ и $Y(m) \leftrightarrow C_Y(k)$, то

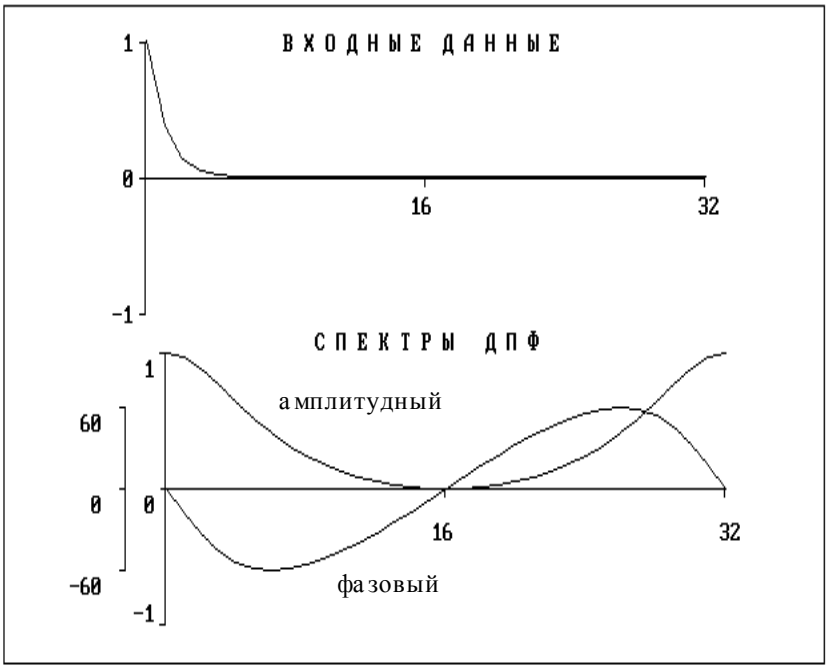

Рис. 9.2. Спектры ДПФ экспоненциально убывающего сигнала

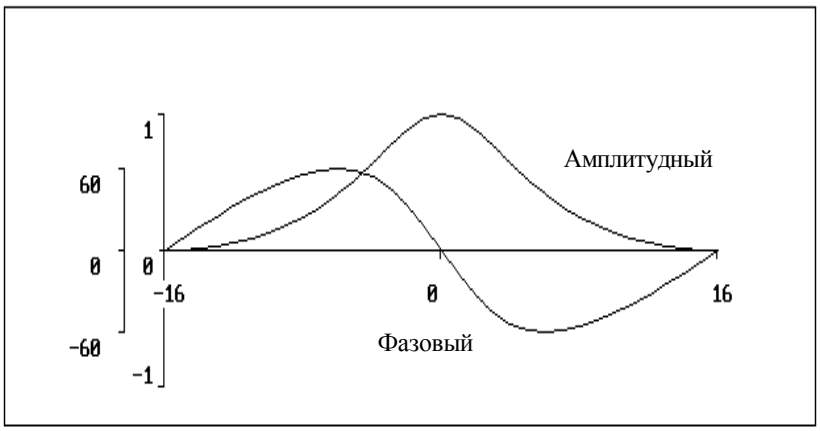

Рис. 9.3. Спектры Фурье для экспоненциально убывающего сигнала

$$C_{\hat{Z}}(k) = \overline{C}_x(k)\cdot C_Y(k), \qquad k = 0,\ 1,\ \ldots,\ N-1 \ .$$

Отсюда следует, что последовательность $\{\hat{Z}(m)\}$ может быть вычислена при помощи алгоритма БПФ в соответствии со схемой, показанной на рис. 9.4. На рис. 9.5, в качестве примера, приведена последовательность $\{\hat{Z}(m)\}$, полученная с помощью процедуры FFT в результате автокорреляции экспоненциально убывающей последовательности $x(t) = e^{-t}$ для N = *32*.

Свертка двух непрерывных T-периодических сигналов $x(t)$ и $y(t)$ определяется интегралом

$$Z_{xy}(\tau) = \frac{1}{T}\int_{T} X(t)\cdot Y(\tau - t)dt \cdot \qquad (9.19)$$

$$Z(m) = \frac{1}{N}\sum_{l=0}^{N-1} X(l)\cdot Y(m-l), \quad \text{m} = 0, 1, \ldots, \text{N} - 1 \cdot \qquad (9.20)$$

В случае дискретных последовательностей *{X(m)}, {Y(m)}* формула (9.19) приобретает вид:

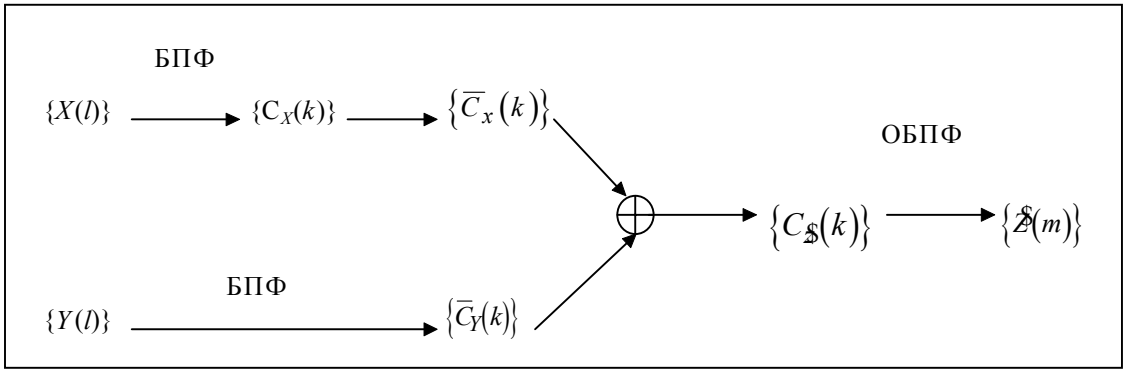

Рис. 9.4. Схема алгоритма вычисления корреляционной последовательности $\{\hat{Z}(m)\}$

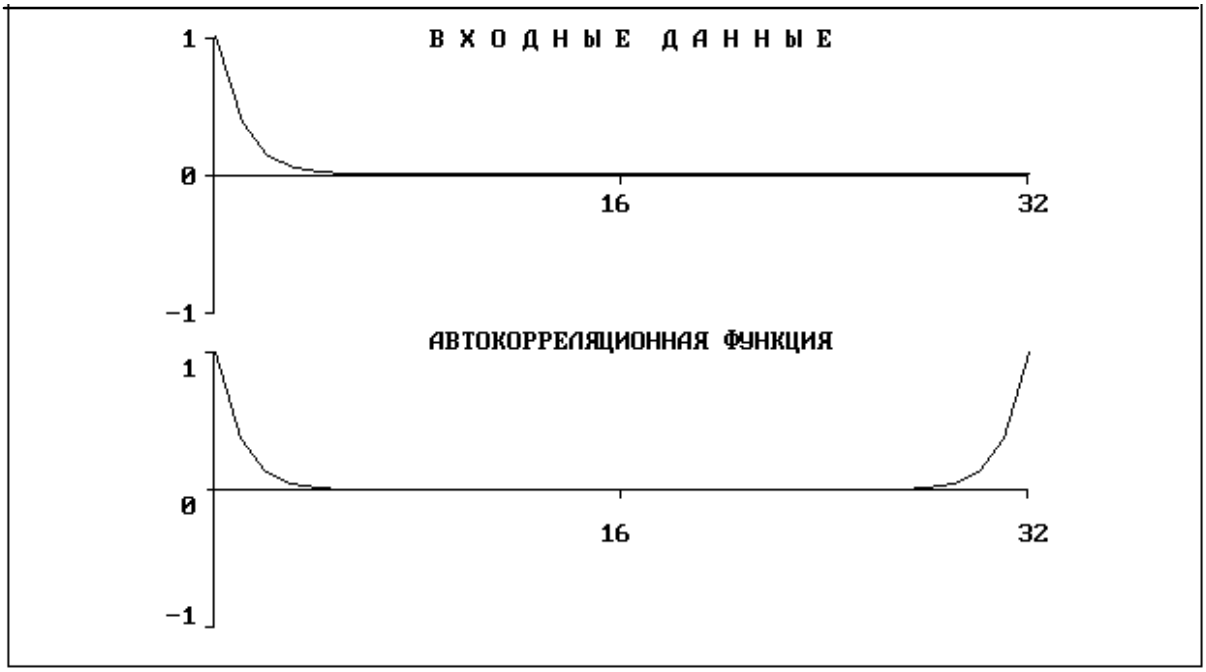

Рис. 9.5. Нормированная автокорреляционная функция
экспоненциально убывающего сигнала

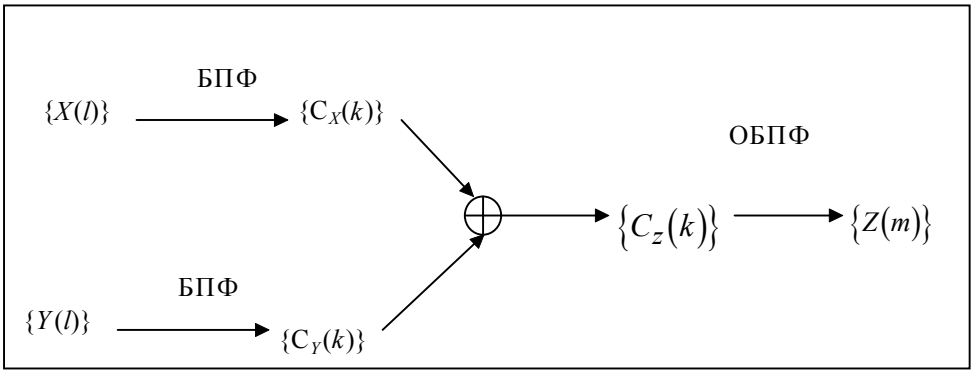

Рис. 9.6. Схема алгоритма вычислениясвертки двух
последовательностей $\{Z(m)\}$

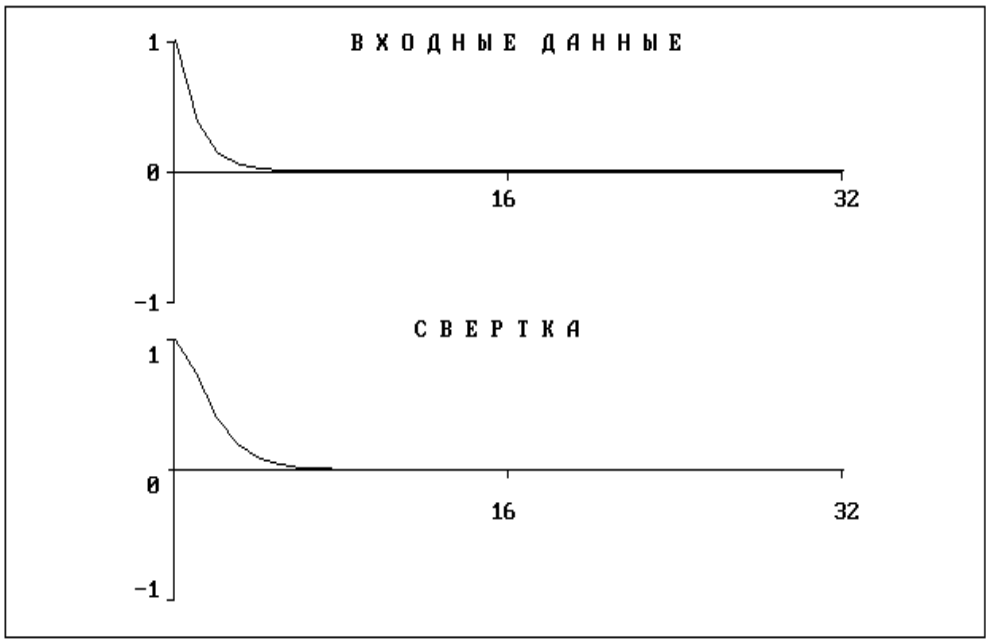

Рис. 9.7. Свертка экспоненциалдьно убывающего сигнала (нормированное значение)

Согласно *теореме о свертке* [Корн, Корн, 1978], в этом случае справедливо равенство $C_z(k) = C_x(k)C_Y(k)$, $k = 0, 1, ..., N - 1$, из которого следует, что свертка $Z(m)$ может быть вычислена с помощью БПФ в соответствии со схемой, изображенной на рис. 9.6. На БПФ рис. 9.7. показана в качестве примера свертка экспоненциально убывающей последовательности $x(t) = e^{-t}$ с самой собой, полученная с использованием процедуры fft.

2.4. СИНТЕЗ СИГНАЛОВ С ПОМОЩЬЮ БПФ

Алгоритм БПФ, изложенный выше, может быть использован для *синтеза* L-периодического продолжения дискретизированного сигнала $x(t)$ по его амплитудному и фазовому спектрам [Ахмед, Рао, 1980]. Пусть дискретизированный сигнал представляет собой последовательность $\{X(m)\}$, $m = 0, 1, ..., N$ при $X(0)=X(N)$, которая отвечает непрерывному сигналу $x(t)$, измеренному на промежутке времени длиной L. Пусть также заданы $|F_x(kw_0)|$ и $\psi_x(kw_0)$ при $k = 0, 1, ..., N/2$ и $N=2^n$, $w_0 = 2\pi/L$. Тогда сигнал $X(m)$ может быть восстановлен (синтезирован) в соответствии со следующим алгоритмом:

1) находим $F_x(kw_0) = |F_x(kw_0)| e^{i\psi_x(kw_0)}$, $k = 0, 1, ..., N/2$;

2) определяем $C_x(k) = (1/L) F_x(kw_0)$, $k = 0, 1, ..., N-1$, где, в соответствии с теоремой о комплексной сопряженности [Ахмед, Рао, 1980], $C_x(N/2 + l) = \overline{C}_x(N/2 - l)$, $l = 1, 2, ..., N/2 - 1$;

3) с помощью алгоритма БПФ производим обратное преобразование Фурье и находим

$$X(m) = \sum_{k=0}^{N-1} C_x(k) \cdot W^{-km},$$

$m = 0, 1, \ldots, N - 1$

Данный алгоритм тестировался с использованием процедуры FFT, пример полученных результатов для $N = 8$, в сравнении с контрольными, представлен в табл. 9.2.

Таблица 9.2

k	$\lvert F_x(kw_0)\rvert$	$\psi_x(kw_0)$	
0	3.5	0	0
1	1.306563	-1.178097	1
2	0.707107	-0.785398	2
3	0.541196	-0.392699	3
4	0.5	-	4
5	-	-	5
6	-	-	6
7	-	-	7

Айвазян С.А., Бежаева З.И., Староверов О.В. Классификация многомерных наблюдений. М.: Статистика, 1974.

Айвазян С.А., Енюков И.С., Мешалкин Л.Д. Прикладная статистика. Основы моделирования и первичная обработка данных. М.: Финансы и статистика, 1983.

Абрамович М., Стиган И. Справочник по специальным функциям. М.: Наука, 1979.

Агеев М.И., Алик В.П., Малюк Л.В., Марков Ю.И. Алгоритмы (51-100). М.: ВЦ АН СССР, 1966.

Агеев М.И., Алик В.П., Марков Ю.И. Библиотека алгоритмов 51б-100б. (Справочное пособие.). Вып. 2. М.: Сов. радио, 1976,

Алберг Дж., Нильсон Э., Уолш Дж. Теория сплайнов и ее приложения / Пер. с англ.: М.: Мир, 1972.

Ашкрофт Дж., Элдридж Р., Полсон Р. и др. Программирование на ФОРТРАНЕ 77 / Пер. с англ. Н.А.Геодакова, Д.П. Матюшина. М.: Радио и связь, 1990.

Бахвалов Н.С. Численные методы (анализ, алгебра, обыкновенные дифференциальные уравнения). Т. 1. М.: Наука, 1973а.

Бахвалов Н.С. Численные методы. М.: Наука, 1973б.

Белашова А.А. Тестирование информации на стационарность в системе регистрации и обработки радиокосмических данных в реальном масштабе времени // Исследование явлений в ионосфере и магнитосфере Земли. Владивосток: ДВО АН СССР, 1990. С. 92 - 97.

Белашова А.А., Белашов В.Ю. Реализация алгоритма Кули - Тьюки вычисления БПФ в системе регистрации и обработки радиокосмофизических данных в реальном масштабе времени // Исследование явлений в ионосфере и магнитосфере Земли. Владивосток: ДВО АН СССР, 1990. С. 87 - 91.

Белашов В.Ю. О численных методах решения эволюционных уравнений типа уравнения Кадомцева-Петвиашвили: Препринт ИКИР. Магадан, 1989. 21 с.

Белашов В.Ю. Численное исследование динамики неодномерных нелинейных волн в слабодиспергирующих средах: Дис... канд. физ.-мат. наук. М.: ИЗМИРАН, 1991.

Белашов В.Ю. Специальные функции и алгоритмы их вычисления. М.: Магадан, 1997.

Белашов В.Ю. Уравнение КП и его обобщения. Теория, приложения. Магадан: СВКНИИ ДВО РАН, 1997.

Белецкий Я. Турбо Паскаль с графикой для персональных компьютеров / Пер. с польск. Д.И.Юренкова. М.: Машиностроение. 1991.

Беляков В.М., Кравцова Р.И., Раппопорт М.Г. Таблицы эллиптических интегралов. Т. 1, 2. М.: 1962.

Березин И.С., Жидков Н.П. Методы вычислений. Т. 2. М.: Физматгиз, 1962.

Библиотека алгоритмов 1б-50б: Справочное пособие. Вып. 1. М.: Советское радио, 1975.

Библиотека алгоритмов 151б-200б: Справочное пособие. М.: Радио и связь, 1981.

Брич З.С., Капилевич Д.В., Котик С.Ю. и др. Фортран ЕС ЭВМ. М.: Статистика, 1978.

Бронштейн И.Н., Семендяев К.А. Справочник по математике для инженеров и учащихся втузов / Пер. с нем. под ред. Г.Гроше и В.Циглера. М.: Наука, 1981.

Вазов В.Р., Форсайт Дж. Разностные методы решения дифференциальных уравнений в частных производных / Пер. с англ. М.: Мир, 1969.

Ван дер Варден Б.Л. Математическая статистика. М.: ИЛ, 1960.

Виноградов И.М. Основы теории чисел. М.: Гостехиздат, 1953.

Воробьева Г.Н., Данилова А.Н. Практикум по вычислительной математике. Изд. 2. М.: Высшая школа, 1990.

Гольцман Ф.М. Статистические методы интерпретации. М.: Наука, 1971.

Гринчишин Я.Т., Ефимов В.И., Ломакович А.Н. Алгоритмы и программы на Бейсике. М.: Просвещение, 1988.

Данциг Дж. Б. Линейное программирование, его применение и обобщения / Пер. с англ. М.: Прогресс, 1966.

Демидович В.М., Марон Б.В. Численные методы вычислительной математики. М.: Высшая школа, 1962

Дорот В.Л., Троицкий В.А., Шелест В.Д. Элементы вычислительной математики. Л.: Изд-во ЛПИ им. Калинина, 1977.

Дорот Л.И., Пименов И.А., Сацук В.В. Математическое обеспечение исследований геофизических закономерностей на примере космических лучей. М.: Наука, 1978.

Дяконов В.П. Справочник по алгоритмам и программам на языке Бейсик. М.: Наука, 1981.

Иберла К. Факторный анализ. М.: Статистика, 1980.

Калиткин В.Г. Численные методы. М.: Наука, 1978.

Касаткин В.И. Лекции по методам вычислений. М.: Наука, 1978.

Кнут Д.Е. Искусство программирования. Т. 2. Получисленные алгоритмы / Пер. с англ. М.: Мир, 1977.

Корн Г., Корн Т. Справочник по математике для научных работников и инженеров / Пер. с англ. под ред. И.Г. Арамановича. М.: Наука, 1978.

Курош А.Г. Курс высшей алгебры. М.: Физматгиз, 1962 .

Крылов В.И., Бобков В.В., Монастырный П.И. Вычислительные методы высшей математики. Т. 1. Минск: Вышэйшая школа, 1972.

Лаврентьев М.А., Шабат Б.В. Методы теории функций комплексного переменного. М.: Физматгиз, 1965.

Люк Ю. Специальные функции и их аппроксимации / Пер. с англ. под ред. К.И.Бабенко. М.: Мир, 1980.

Мизрохи С.В. TURBO PASCAL и объектно-ориентированное программирование. М.: Фин. и стат., 1992.

Ортега Дж., Рейнболд В. Итерационные методы решения нелинейных уравнений со многими неизвестными / Пер. с англ. М.: Мир, 1975.

Островский А.М. Решение уравнений и систем уравнений/Пер. с англ. М.: ИЛ, 1963.

Поляков Д.Б., Круглов И.Ю. Программирование в среде Турбо Паскаль (версия 5.5) / Справочно-методическое пособие. М.: Изд-во МАИ, 1992.

Плис А.И., Сливина Н.А. Лабораторный практикум по высшей математике. М.: Высшая школа, 1983.

Самарский А.А., Гулин А.В. Численные методы: Учебное пособие для вузов. М.: Наука, 1989.

Справочник по специальным функциям / Под ред. М. Абрамовича, И.Стиган. М.: Наука, 1979.

Теория вероятностей и математическая статистика/Под. ред. Колемаева В.А.: Учебное пособие для эконом. спец. вузов. М.: Высшая школа, 1992.

Троицкий В.А., Иванова И.М. Методы вычислительной математики. Ч. 1. Л.: Изд-во ЛПИ им. Калинина, 1975.

Уилкинсон Дж. Алгебраическая проблема собственных значений / Пер. с англ.М.: Наука, 1970.

Уилкинсон Дж. Райнш. Справочник алгоритмов на языке АЛГОЛ. Линейная алгебра / Пер. с англ. М.: Машиностроение, 1976.

Фаддеев Д.К., Фаддеева В.Н. Вычислительные методы линейной алгебры. М.: Физматгиз, 1963.

Фишер Р.А. Статистические методы для исследования. М.: Госстатиздат, 1958.

Форсайт Дж., Малькольм М., Моулер К. Машинные методы математических вычислений / Пер. с англ. Х.Д. Икрамова. М.: Мир, 1980.

Форсайт Дж., Молер К. Численное решение систем линейных алгебраических уравнений / Пер. с англ. М.: Мир, 1969.

Хемминг Р.В. Численные методы для научных работников и инженеров / Пер. с англ. М., Наука, 1972.

Численные методы. Учебник для техникумов. / Под ред. Данилина Н.И., Дубровской Н.С., Кваши О.П. и др. М.: Высшая школа, 1976.

Янке Е., Эмде Ф., Леш Ф. Специальные функции. М.: Наука, 1968.

Bartlett M.S. The use of transformatioms. Biometrics, 3, 39, 1947.

Belashov V.Yu. The methods for numerical integration of nonlinear evolutional KP-class equations // XX Int. Conf. on Phenomena in Ionized Gases, Italy, Pisa, July 8-12, 1991. Contributed papers. V. 6. P. 1241-1242.

Brent R.P. Algorithms for minimization without derivatives. Englewood Cliffs, N. J.: Prenctice-Hall. 1973.

Caffrey J. Communications of the ACM. Algorithms, 1962, N 6.

Clark Eva S. Communications of the ACM // Algorithms, 1963, N 3.

Cjjley J.W., Tukey J.W. An Algorithm for Machine Computation of Complex Fourier Series // Mathematics of Computation, 1965. V. 19. P. 297 - 301.

Dekker T.J. Finding a zero by means of successive linear interpolation, in B. Dejon and P. Henrici (eds.), Constructive aspects of the fundamental thejrem of algebra. New York: Wiley-Interscoence. 1969.

Fisher J.R. Fortran Program for Fast Fourier Transform // Naval Research Laboratory Report. N 1074, Washington, D.C., 1970.

Floyd R.W. Communications of the ACM // Algorithms, 1962, N 6.

Georg R. Communications of the ACM // Algorithms, 1962, N 7.

Goerzel G. Mathematical Methods for Digital Computers / A.Ralston and H.S.Wilf, eds., 1961.

Giammo T.P. Communications of the ACM // Algorithms, 1962, N 2.

Greenstadt J. The determination of the characteristic roots of a matrix by the Jacobi method//Mathemetical methods for Digital Computers/A.Ralston and H.S.Wilf, eds., 1961.

Gulinsky O.V., Belashov V.Yu., Dorman L.I. e.a. Analysis of the Small-scale Cosmic Ray Fluctuations Spectrum Inferred From Ground-based Cosmic Ray Observation Data // Geofisica International, Mexico, D.F., Io, de enero de 1988. V. 27. N 1. P. 3 - 36.

Halstead M.H. Communications of the ACM // Algorithms, 1962, N 3.

Hennion P.E. Communications of the ACM // Algorithms, 1962, N 2.

Herndon J.R. Communications of the ACM // Algorithms, 1961, N 4.

Mifsud C.J. Communications of the ACM // Algorithms, 1961, N 11.

Nestor C.W. Communications of the ACM // Algorithms, 1961, N 7.

Peck J.E.L. Polynomial curve fitting with constraint // Soc. Indust. Appl. Math. Rev., 1961.

Perry C. Communications of the ACM // Algorithms, 1962, N 2.

Pfaltz J.L. Communications of the ACM // Algorithms, 1962, N 6.

Randell B., Broyden C.G. Communications of the ACM // Algorithms, 1962, N 1.

Relph A.P. Communications of the ACM // Algorithms, 1962, N 7.

Roek D.J. Communications of the ACM // Algorithms, 1962, N 5.

Romberg W. Vereinfachte numerische Integration. Det. Konglinge Norske Videnskaber Selskab Forhandlinger, 1955. N 28. P. 30-36.
Satterthwaite F.E. An approximate distribution of estimates of variance components. Biometrics. 2, 10, 1946.
Thacher H.C. Communications of the ACM // Algorithms, 1962, N 3.
Thacher H.C. Communications of the ACM // Algorithms, 1963, N 3.
Thacher H.C. Communications of the ACM // Algorithms, 1964, N 7.
Welch B.L. On linear combinations of several variances // J. Amer. Statist. Assoc., 51, 132, 1956.
Wilkinson J.H. Rounding errors in algebraic processes. Englewood Cliffs, N.J.:Prentice-Hall. 1967.